Hindawi Publishing Corporation
ISRN Signal Processing
Volume 2013, Article ID 748914, 11 pages
http://dx.doi.org/10.1155/2013/748914

Research Article

A Novel Neuron in Kernel Domain

Zahra Khandan[1] and Hadi Sadoghi Yazdi[1,2]

[1] *Department of Computer Science, Ferdowsi University of Mashhad, Mashhad, Iran*
[2] *Center of Excellence on Soft Computing and Intelligent Information Processing, Ferdowsi University of Mashhad, Iran*

Correspondence should be addressed to Hadi Sadoghi Yazdi; h-sadoghi@um.ac.ir

Received 23 June 2013; Accepted 22 July 2013

Academic Editors: A. Krzyzak and L. Zhang

Copyright © 2013 Z. Khandan and H. Sadoghi Yazdi. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Kernel-based neural network (KNN) is proposed as a neuron that is applicable in online learning with adaptive parameters. This neuron with adaptive kernel parameter can classify data accurately instead of using a multilayer error backpropagation neural network. The proposed method, whose heart is kernel least-mean-square, can reduce memory requirement with sparsification technique, and the kernel can adaptively spread. Our experiments will reveal that this method is much faster and more accurate than previous online learning algorithms.

1. Introduction

Adaptive filter is the heart of most neural networks [1]. LMS method and its kernel-based methods are potential online methods with iterative learning that are used for reducing mean squared error toward optimum Wiener weights. Due to simple implementation of LMS [1], this method became one of the candidates for online kernel-based learning. The kernel-based learning [2] utilizes Mercer kernels in order to produce nonlinear versions of conventional linear methods.

After the introduction of the kernel, kernel least-mean-square (KLMS) [3, 4] was proposed. KLMS algorithm tries to solve LMS problems in reproducing kernel hilbert spaces (RKHS) [3] using a stochastic gradient methodology. KNN has such characteristics as kernel abilities and LMS features, easy learning over variants of patterns, and traditional neurons capabilities. The experimental results show that this classifier has better performance than the other online kernel methods, with suitable parameters.

Two main drawbacks of kernel-based methods are selecting proper value for kernel parameters and series expansions whose size equals the number of training data, which make them unsuitable for online applications.

This paper concentrates only on Gaussian kernel (for similar reasons to those discussed in [5]), while KNN uses other kernels too. In [6], the role of kernel width in the smoothness of the performance surfaces. Determining the kernel width of Gaussian kernel in kernel-based methods is very important. Controlling kernel width can help us to control the learning rate and the tradeoff between overfitting and underfitting.

Use of cross-validation is one of the simplest methods to tune this parameter which is costly and cannot be used for datasets with too many classes. So, the parameters are chosen using a subset of data with a low number of classes in [7]. In some methods, genetic algorithm [8] or grid search [5] is used to determine the proper value of such parameters. However, in all the mentioned methods, the kernel width is chosen as a preprocess which is against the principle of online learning methods. In addition, it has time overhead problem. Therefore, we follow methodologies that are consistent with online applications. In other works, the kernel width is scaled in a distribution-dependent way for SVM methods [9], kernel Polarization is used as a kernel optimality criterion independent of the learning machine [10], and in [11] a data-dependent kernel optimization algorithm is employed whcih maximizes the class separability in the empirical feature space. The computational complexity of these methods is quadratic with respect to the number of selected training data. Lately, an adaptive kernel width method to optimize

Table 1: Notations.

	Description	Examples
Scalers	Small *italic* letters	$\mu, \sigma, \gamma, \eta$
Vectors	Small **bold** letters	$\mathbf{w}, \mathbf{x}, \mathbf{e}, \boldsymbol{\alpha}$
Matrixes	Capital **bold** letters	$\mathbf{X}, \mathbf{G}, \boldsymbol{\Phi}$
Time or iteration	Indices in parentheses	$\mathbf{w}(t), \mathbf{k}(t)$
Components of vector	Subscript indices	$\mathbf{w}_i(t)$

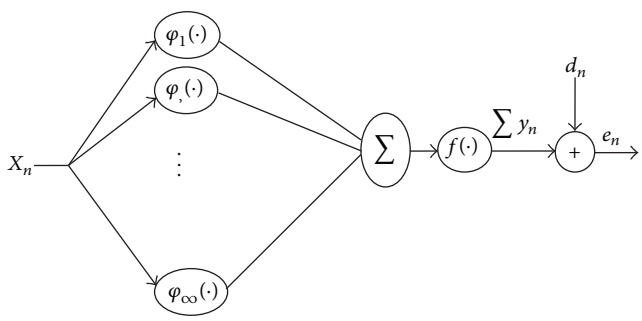


Figure 1: Structure of neural network KLMS.

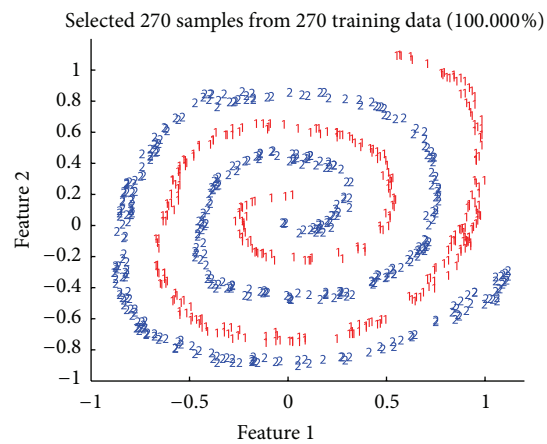


Figure 2: Spiral dataset by 800 instances.

minimum error entropy (MEE) criterion that has a linear complexity with respect to the length of the window used for computing the density estimate is proposed [12].

We proposed an adaptive kernel width learning in KNN method to maintain the online nature of it without any preprocess and reach convergence. We use the gradient property of KNN in order to estimate the best kernel width value during the process. Therefore, KNN method with adaptive Kernel width remains online and improves its accuracy as compared to the versions with fixed kernel width.

In other sides, it is needed to decrease computational complexity of kernel-based method to be useful in online application. Usually, used pruning [13–15] or fixed size models [13, 14, 16–18] in batching methods and truncation [17, 18] in online methods. We focus on online model reduction. KRLS and KLMS algorithms proposed some sparsification techniques based on ALD criterion [19, 20], by checking the linear dependency of the input feature vectors, which have a quadratic complexity. A sliding-window kernel RLS algorithm consists in only taking the last L pairs of arrived data, but it is a local method [21]. In [22, 23], proposed an online coherence based sparsification method on kernel-based affine projection (KAP) and KLMS that has only linear complexity. Coherence parameter is a fundamental quantity that characterizes the behavior of dictionaries in sparse approximation problems. In this paper, we combine the coherence criterion with proposed AKNN to control growth number of instances.

This paper is organized as follows. In Section 2, KLMS and KNN are introduced. Section 3 discusses adapting kernel parameter and sparsing instances in KNN. In Section 4, experiments illustrate the effectiveness of our approach compared to other existing methods. Finally, Section 5 summarizes the conclusions and points out avenues for further research.

2. Background

In this section, a short review of LMS and KLMS algorithms is presented. We introduce notations in Table 1 for better understanding of the formulation.

2.1. LMS Algorithm. The main purpose of the LMS algorithm is to find a proper weight vector, which can reduce the MSE of the system output based on a set of examples (x_i, d_i). Therefore, LMS cost function is

$$J = \min_{\mathbf{w}} \sum_{i=1}^{t} \left(\mathbf{d}_i - \mathbf{x}_i^T \mathbf{w}\right)^2 = \min_{\mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|^2. \quad (1)$$

The LMS algorithm approximates weight vector $\mathbf{w}$ using gradient method [1]:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \widehat{\nabla}_{\mathbf{w}} \left(\mathbf{e}_t^2\right) = \mathbf{w}(t) + \eta \mathbf{e}_t \mathbf{x}_t, \quad (2)$$

where η is the stepsize parameter. LMS algorithm is a famous linear filter because it has easy implementation and low computation.

2.2. KLMS Algorithm. The LMS algorithm can learn linear patterns very well but it is poor in learning nonlinear patterns. To overcome this problem, Puskal derived LMS algorithm directly in the kernel feature space [3, 4]. Kernel methods map the input data into a high dimensional space (HDS) [24]. The mapping procedure (Φ) helps to compute nonlinear problems using linear algorithms. After that, Mercer's theorem provides a kernel function to compute inner product of data

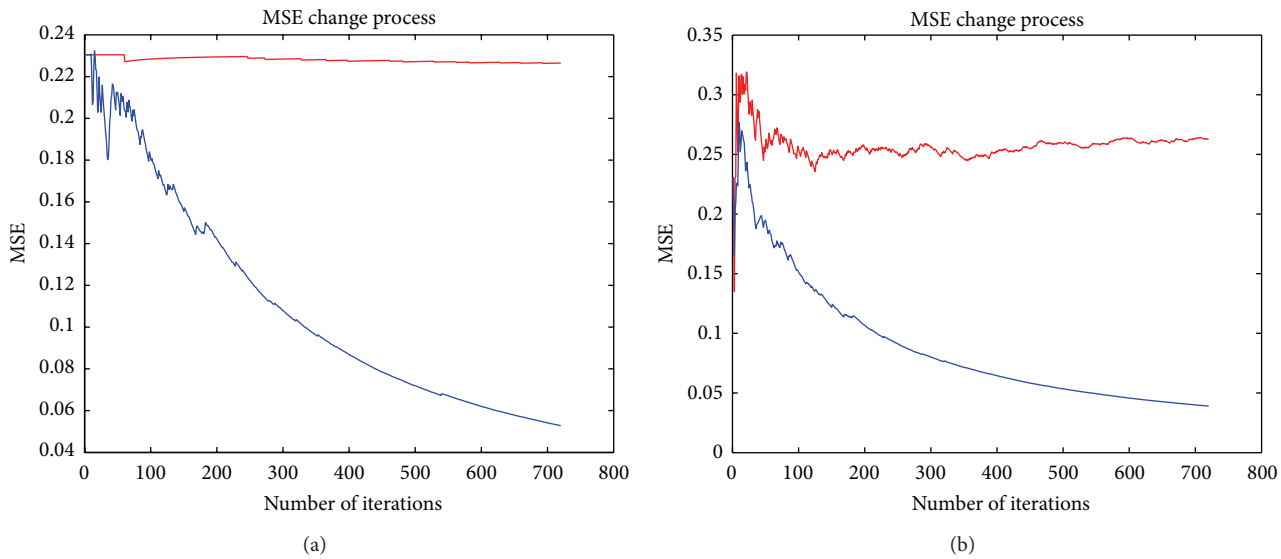


Figure 3: Evolution curves of the MSE of KNN and AKNN methods for (a) $\sigma = 0.001$, (b) $\sigma = 1$.

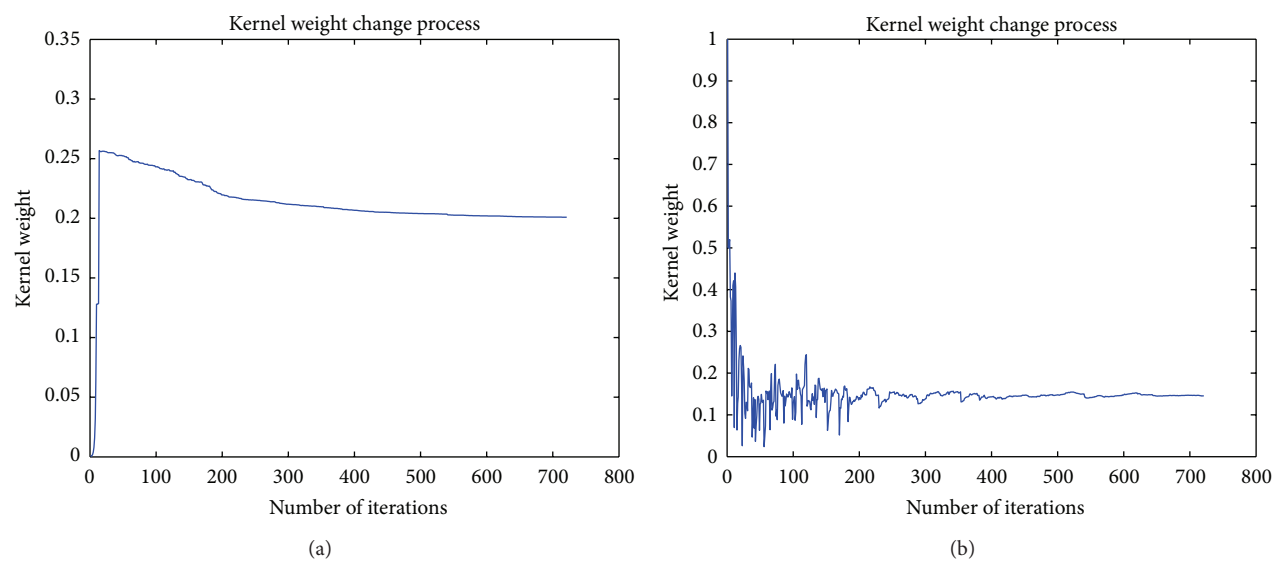


Figure 4: Evolution curves of the σ changes in AKNN method for different (a) $\sigma = 0.001$, (b) $\sigma = 1$.

from HDS directly in the input space that is called the kernel trick:

$$k\left(x, x'\right) = \left\langle \varphi\left(x\right), \varphi\left(x'\right)\right\rangle. \tag{3}$$

The basic idea of the KLMS algorithm is to perform the linear LMS algorithm on $\{(\varphi(\mathbf{x}_1), d_1), \ldots, (\varphi(\mathbf{x}_n), d_n)\}$ in the feature space. So, KLMS cost function is given by

$$J = \min_{\mathbf{w}} \|\mathbf{d} - \mathbf{\Phi}\mathbf{w}\|^2, \tag{4}$$

where $\mathbf{\Phi}$ and $\mathbf{w}$ are matrixes of input vectors and weight vectors in feature space (RKHS), respectively. For convenience assuming that $\mathbf{w}(0) = 0$, therefore

$$\mathbf{w}\left(t\right) = \mathbf{w}\left(t\right) + \eta\widehat{\nabla}_{\mathbf{w}}\left(\mathbf{e}_t^2\right) = \eta\sum_{i=0}^{t-1}\mathbf{e}_i\varphi\left(\mathbf{x}_i\right). \tag{5}$$

By exploiting the kernel trick, output is given by

$$\mathbf{y}_t = \eta\sum_{i=0}^{t-1}\mathbf{e}_i k\left(\mathbf{x}_i, \mathbf{x}_t\right) = \eta\mathbf{e}(t-1)^T\mathbf{k}, \tag{6}$$

where $\mathbf{k} = [k(\mathbf{x}_1, \mathbf{x}_t), \ldots, k(\mathbf{x}_{t-1}, \mathbf{x}_t)]^T$. Good prediction ability in nonlinear channels is an advantage of the KLMS algorithm, but the complexity of this method for each input is $O(n)$, and n is the number of training data, which is a problem especially in online application.

3. Kernel-Based Neural Network

This section includes five parts for better presentation of the proposed kernel-based neural network. First, KLMS based

Initialization:
learning step η
learning kernel width step γ
primal kernel width σ
sparsity threshold μ
$t = 1$;
While $\{\mathbf{x}(t), \mathbf{d}(t)\}$ available ***do***
  for l instances ***do***
    % compute distances of dictionary instances to tth instance
    $\mathbf{dis}_i = \|\tilde{x}_i - x_t\|^2$; $O(m_{t-1})$
    % compute kernel vector, output and error for tth instance
    $\tilde{\mathbf{k}} = \exp(-\mathbf{dis}^T/\sigma^2)$, $y_t = f(\eta\boldsymbol{\alpha}^T\tilde{\mathbf{k}}), \mathbf{e}_t = \mathbf{d}_t - \mathbf{y}_t$ $O(m_{t-1})$
    % compute coefficient $\boldsymbol{\alpha}_t$ using (19)
    % save σ_{pred} for σ tracking
    $\sigma_{\text{pred}} = \text{avg}[\sigma_{\text{pred}}, \text{avg}(\mathbf{dis})]$
    % update kernel width σ
    $\sigma(t) = \sigma(t-1) + \gamma\boldsymbol{\alpha}_t\tilde{\boldsymbol{\alpha}}^T((\mathbf{dis}/\sigma^3)\cdot\tilde{\mathbf{k}})$ $O(m_{t-1})$
    if σ became negative ***then***
    % decrease γ until σ becomes non-negative
    % update kernel width σ again.
    end if
    if $(\max(\mathbf{k}(t-1)) \le \mu_0)$
    % add $\mathbf{x}_t$ to dictionary and update $\tilde{\boldsymbol{\alpha}}$ using (23);
    else
    % update Coefficients $\tilde{\boldsymbol{\alpha}}$ using (24);
    end if
    $t = t + 1$
end for
if $(\text{mse}_l \ge \delta)$
    % manipulated step size γ using (20)
else if $(|\text{mse}_{l-1} - \text{mse}_l| < \varepsilon)$
  % exit training process
end if
end while

Algorithm 1: The adaptive KNN algorithm (AKNN_μ). Complexity for each instance.

neuron is explained; then kernel adaptation and stepsize of adaptation and termination condition are discussed, and final subsection includes the sparsification.

3.1. The KLMS Neural Network (KNN). The KLMS neural network (KNN) performs the classification task by adding a nonlinear logistic function to KLMS structure; Figure 1 illustrates the structure of KNN.

Similar to the KLMS algorithm, we perform a gradient search in order to find the optimum weight. If $y_t = f(\varphi(\mathbf{x}_t)^T\mathbf{w})$ is tth output and $\mathbf{w}(0) = 0$, then

$$\mathbf{w}(t) = \eta\sum_{i=0}^{t-1}\mathbf{e}_i f'\left(\varphi(\mathbf{x}_i)^T\mathbf{w}(t-1)\right)\varphi(\mathbf{x}_t). \quad (7)$$

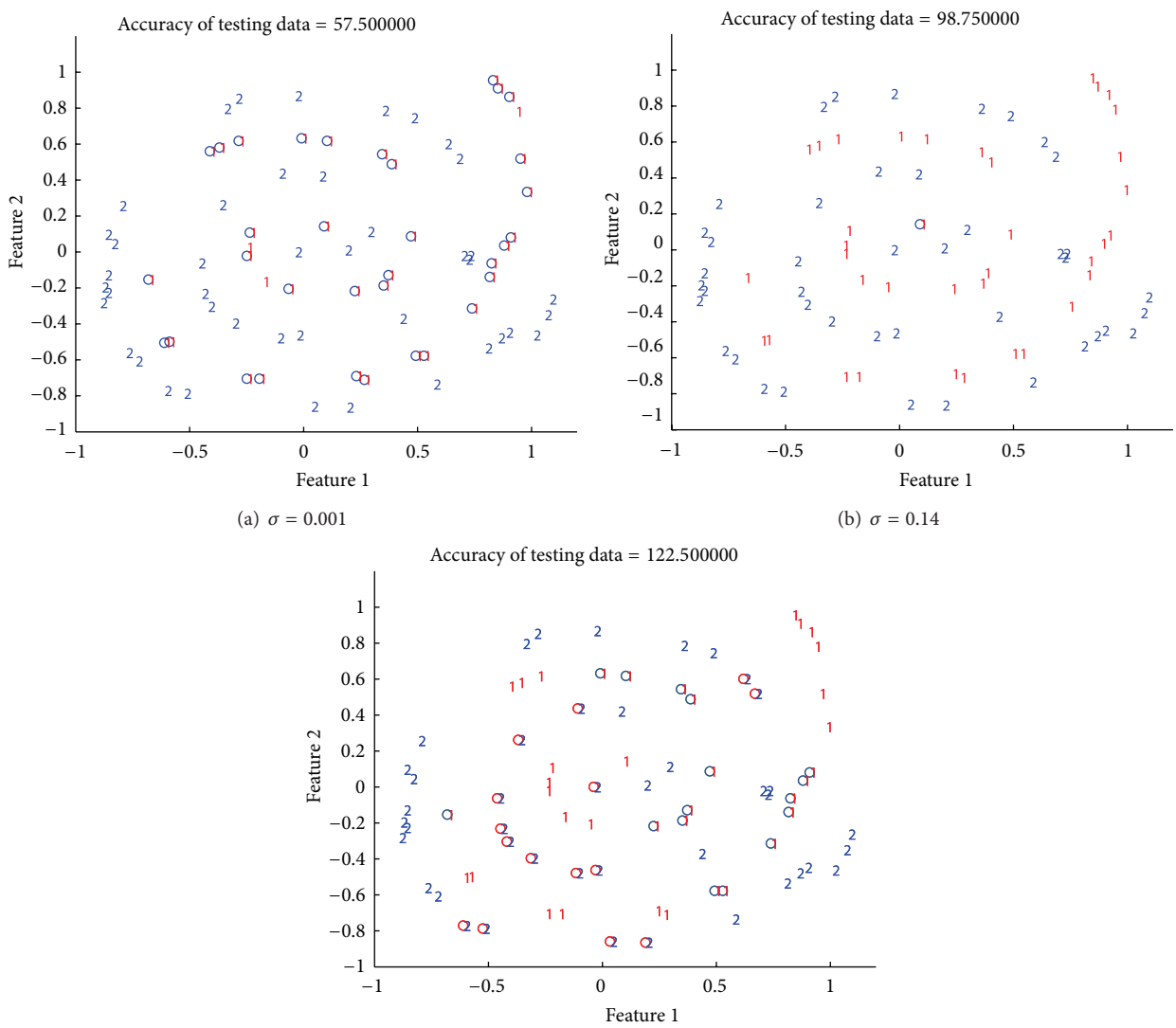
By using kernel trick, $\mathbf{y}_t$ is given as follows:

$$\mathbf{y}_t = f\left(\eta\sum_{i=0}^{t-1}\boldsymbol{\alpha}_i k(\mathbf{x}_i, \mathbf{x}_t)\right) = f\left(\eta\boldsymbol{\alpha}(t-1)^T\mathbf{k}\right), \quad (8)$$

where $\boldsymbol{\alpha}_t = \mathbf{e}_t f'(\eta\boldsymbol{\alpha}(t-1)^T\mathbf{k})$. Therefore, KNN can determine the classifier output by calculating coefficients $\boldsymbol{\alpha}$ in learning state and input vectors.

According, what was said, finding a proper kernel play an important role in kernel-based learning. The best kernel function for learning each dataset is different. One solution for improving kernel function is finding the best kernel parameters. We try to determine the best kernel width σ in KNN (assuming a Gaussian kernel). Whereas KNN is an online learning method, we need an iterative method to find appropriate σ instead of methods by high complexity or methods that find it by preprocessing on data. A good suggestion is to select a random value for σ at the beginning of the training step and update it until convergence is obtained.

3.2. Adaptation of Kernel Width. The goal of LMS family's methods is to reduce mean square error, and this aim will be achieved by using gradient search. It can be proved that the KNN, such as KLMS algorithm, converges when there are infinite numbers of samples. If kernel space structure and cost function were derivable, then gradient methods can be



(a) $\sigma = 0.001$

(b) $\sigma = 0.14$

(c) $\sigma = 1$

FIGURE 5: Classification result of KNN method for different values of σ.

used for finding kernel's parameters. Due to the derivability of Gaussian kernel function and MSE cost function, gradient search method can be used for updating kernel weight to reach the least mean square error. If the proposed modified KNN cost function is defined as

$$J_t(\boldsymbol{\alpha}, \sigma) = \mathbf{e}_t^2, \quad \mathbf{e}_t = \mathbf{d}_t - f\left(\eta \boldsymbol{\alpha}^T \mathbf{k}\right) \tag{9}$$

and the kernel formulation is

$$k_\sigma(x, z) = \exp\left(\frac{-\|x - z\|^2}{\sigma^2}\right), \tag{10}$$

then there is an error function which depends on weight Ω and kernel weight σ. Using gradient of $\mathbf{e}_t^2$ with respect to σ, the least mean square error is achieved:

$$\sigma(t+1) = \sigma(t) - \gamma' \nabla_{\sigma(t)}\left(\mathbf{e}_t^2\right), \tag{11}$$

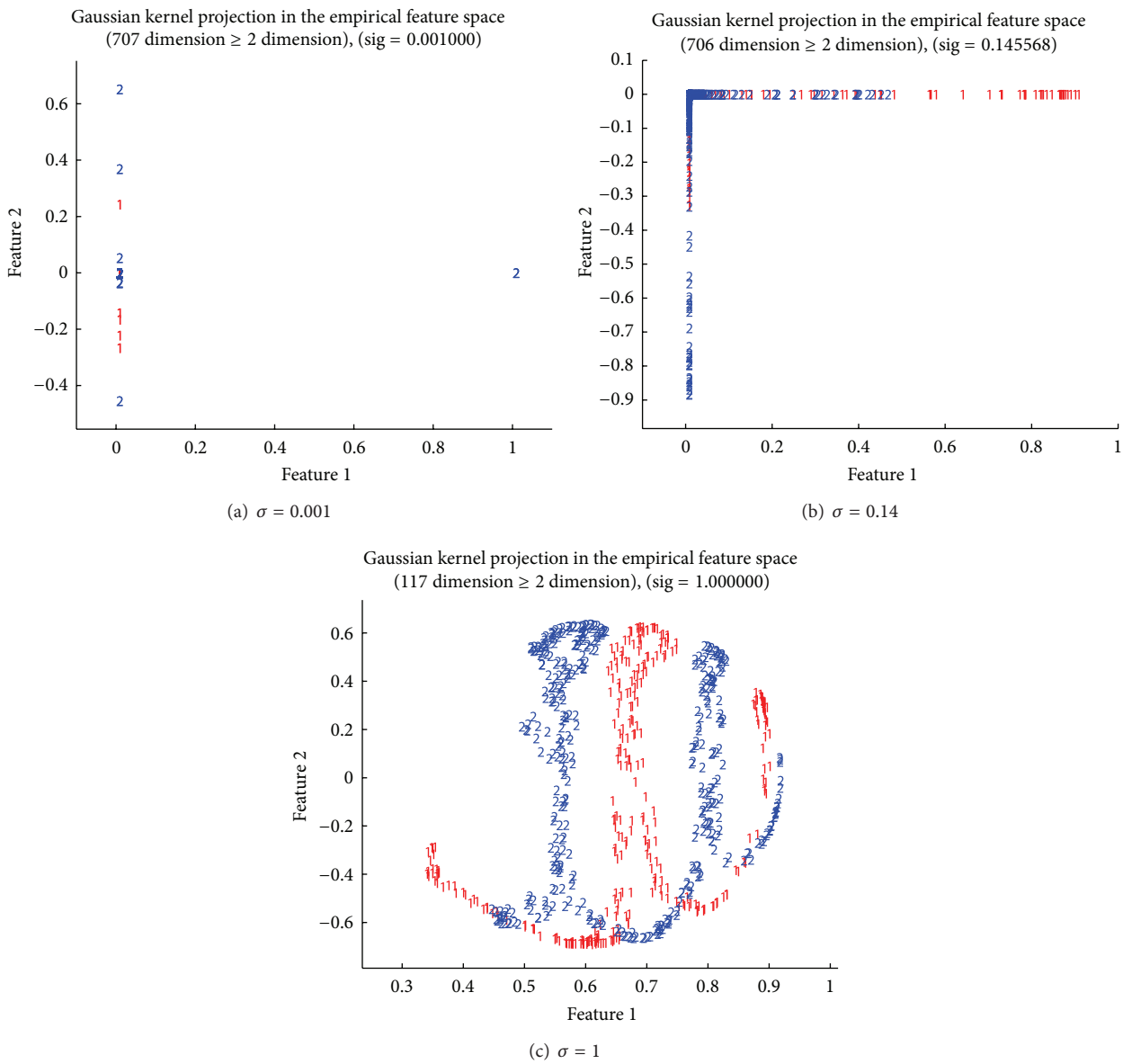
where γ' is stepsize and

$$\nabla_\sigma\left(\mathbf{e}_t^2\right) = 2\mathbf{e}_t\left(\frac{\partial \mathbf{y}_t}{\partial \sigma(t)}\right) = 2e_t\left(\frac{\partial f}{\partial u}\frac{\partial u}{\partial \sigma(t)}\right), \quad \text{where } u = \eta \sum_{i=0}^{t-1} \boldsymbol{\alpha}_i k\left(\mathbf{x}_i, \mathbf{x}_t\right)_\sigma. \tag{12}$$

At first, $\partial k / \partial \sigma$ can be solved that depends on σ:

$$\frac{\partial k}{\partial \sigma} = 2\frac{\|x - z\|^2}{\sigma^3} \exp\left(\frac{-\|x - z\|^2}{\sigma^2}\right) \tag{13}$$

$$\Longrightarrow \frac{\partial u}{\partial \sigma(t)} = \eta \sum_{i=0}^{t-1} \boldsymbol{\alpha}_i \frac{\|\mathbf{x}_i - \mathbf{x}_t\|^2}{\sigma(t)^3} k\left(\mathbf{x}_i, \mathbf{x}_t\right). \tag{14}$$



(a) $\sigma = 0.001$

(b) $\sigma = 0.14$

(c) $\sigma = 1$

Figure 6: Two-dimensional projection in empirical feature space for different initial values of σ.

By substituting (14) in (12), error gradient is given by:

$$\nabla_\sigma\left(\mathbf{e}_t^2\right) = -\eta \mathbf{e}_t f'(u) \sum_{i=0}^{t-1} \frac{\left\|\mathbf{x}_i - \mathbf{x}_t\right\|^2}{\sigma(t)^3} \boldsymbol{\alpha}_i k\left(\mathbf{x}_i, \mathbf{x}_t\right). \quad (15)$$

As a result, by substituting error gradient in (11) and using $\boldsymbol{\alpha}_t = \mathbf{e}_t f'(u)$, σ is updated by

$$\sigma(t+1) = \sigma(t) - \gamma \boldsymbol{\alpha}_t \sum_{i=0}^{t-1} \frac{\left\|\mathbf{x}_i - \mathbf{x}_t\right\|^2}{\sigma(t)^3} \boldsymbol{\alpha}_i k\left(\mathbf{x}_i, \mathbf{x}_t\right), \quad (16)$$

where $\gamma = \gamma' \eta$ is the stepsize that controls the convergence and speed of the kernel weight process. Choosing proper value for γ is a main challenge that some of its problems are explained in Section 3.2. The AKNN output can be calculated with coefficients $\boldsymbol{\alpha}$, kernel width σ, achieved from training state, and input vectors X as follows:

$$\mathbf{y}_n = f\left(\eta \sum_{i=0}^{n-1} \boldsymbol{\alpha}_i k_\sigma\left(\mathbf{x}_i, \mathbf{x}_n\right)\right), \quad (17)$$

where $f(\cdot)$ is a sigmoid function that covers all ranges of data $(-\infty < u < +\infty)$ and is bounded between 0 and 1. This function is derivable, which is important for using gradient method, and its derivative can be achieved from sigmoid function, but you can use other suitable functions. Sigmoid

function and its derivatives are equal to

$$f(u) = \frac{1}{1+e^{\beta u}},$$
$$f'(u) = -\frac{\beta e^{\beta u}}{1+e^{\beta u}} = \beta f(u)\left(1 - f(u)\right). \tag{18}$$

According to (18), we can reformulate $\boldsymbol{\alpha}_t$ for the next sample as below that decrease training time:

$$\boldsymbol{\alpha}_t = \mathbf{e}_t f'(u) = \beta \mathbf{e}_t \mathbf{y}_t \left(1 - \mathbf{y}_t\right). \tag{19}$$

Algorithm 1 describes the proposed adaptive KNN method that has a unique solution the same as KLMS algorithm.

3.3. Modulation of Stepsize γ. There are some problems with choosing a fixed γ that we intend to solve them with a simple way without adding a new procedure. Some of these solutions are as follows:

(*a*) *Preventing Negation of Kernel Width σ.* By choosing large γ, large jumps occurs which sometimes lead to σ passing from positive range to negative range. In order to prevent negation of σ, new σ value is checked in each time step. If σ is negative, γ has been decreased until σ stays in positive range.

(*b*) *Tracking Kernel Width σ.* When current σ is far from real $\sigma(\sigma_r)$ and γ is too small, adaptation procedure cannot reach σ_r very well. Or when σ is nearby σ_r, occasionally σ will be far from σ_r under a too large γ.

We can track σ value change by controlling stepsize γ. But when does this problem happen? Assuming that δ is a danger threshold, we expect that the error learning does not exceed this threshold and MSE_l is mean square error for l instances; if $\mathrm{MSE}_l \geq \delta$, then we can change γ according to distance of current σ from σ_r:

$$\gamma = \left|\sigma_{\text{pred}} - \sigma\right|, \tag{20}$$

where σ_{pred} is a prediction of σ_r. It means that MSE is too large when γ is too small to change quickly σ by (16). So, when σ is far from σ_r, γ will increase to σ reaching to σ_r quickly, or when σ is nearby σ_r, γ will decrease for convergence. A simple way for expecting σ_{pred} is using the average distance of l passed instances:

$$\sigma_{\text{pred}} = \sum_{i=1}^{l}\sum_{j=1}^{m}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|. \tag{21}$$

According to (10), Gaussian kernel has a good resolution when $0.1 < \|x - z\|/\sigma^2 < 10$. Therefore, this average could give me an approximation of σ_r.

3.4. Termination Condition. When MSE is an acceptable range ($\mathrm{MSE}_l < \delta$) and its change is not noticeable ($|\mathrm{MSE}_{l-1} - \mathrm{MSE}_l| < \varepsilon$), the procedure can be terminated.

Table 2: Datasets used in the experiments.

Dataset	No. of instances	No. of features
Small		
Sonar	208	60
Ionosphere	351	34
Pima	768	8
Medium		
German	1,000	24
Splice	1,000	60
Cloud	2048	10
Large		
Football	4288	13
Spambase	4601	58
MITFace	6,977	361
Mushrooms	8,124	112

3.5. Adaptive Sparsification in KNN. Growing network with arriving training inputs is the other drawback of kernel-based online learning algorithms. In this section, we use a proper criterion to cope with this problem and to produce sparse approximation of functions based on RKHS [22, 23]. As we show, this sparsification approach yields an efficient algorithm that has a linear complexity with respect to the number of dictionary elements. We incorporate the coherence criterion into the new AKNN method to increase the number of variables controlled by the coherence parameter (μ_0).

In online coherence-based sparsification algorithm using coherence criterion, whenever a new data pair $(\varphi(x_t), d_t)$ are arrives, a decision is made of whether the new data add to the dictionary $\widetilde{X}_{m_{t-1}} = \{\widetilde{x}_j\}_{j=1}^{m_{t-1}}$. If $\widetilde{\mathbf{k}} = [k(\widetilde{x}_1, x_t), \ldots, k(\widetilde{x}_{m_{t-1}}, x_t)]^T$ is kernel vector for tth instance, the coherence rule has two modes [23].

(a) If $\max_{j=1\cdots m}|\widetilde{\mathbf{k}}| > \mu_0$. The new instance is not added to $\widetilde{X}_m$ because $\varphi(\widetilde{x}_t)$ can be reasonably well represented by the kernel functions of the dictionary elements. We suggest a gradient approach for applying effects of removed instance error on dictionary elements coefficients to minimize e_t^2:

$$\frac{\partial e_t^2}{\partial \widetilde{\boldsymbol{\alpha}}} = 2e_t\left(\frac{\partial f}{\partial u}\frac{\partial u}{\partial \widetilde{\boldsymbol{\alpha}}}\right) = 2E_t\left(\frac{\partial u}{\partial \widetilde{\boldsymbol{\alpha}}}\right) = \eta'\boldsymbol{\alpha}_t\widetilde{\mathbf{k}}. \tag{22}$$

So, recursive update equation for $\widetilde{E}$ is obtained by

$$\widetilde{\boldsymbol{\alpha}}(t) = \widetilde{\boldsymbol{\alpha}}(t-1) + \eta'\widetilde{K}_{t-1}\boldsymbol{\alpha}_t, \tag{23}$$

where $\eta' = \eta/m_{t-1}$ is the stepsize with forgetting factor capability.

(b) If $\max_{j=1\cdots m}|\widetilde{\mathbf{k}}| \leq \mu_0$. The new instance is added to $\widetilde{X}_m$ and $\widetilde{\boldsymbol{\alpha}}$ vector is updated by

$$\widetilde{\boldsymbol{\alpha}}(t) = \begin{bmatrix}\widetilde{\boldsymbol{\alpha}}(t-1)\\ 0\end{bmatrix} + \eta'\begin{bmatrix}\widetilde{K}_{t-1}\\ 1\end{bmatrix}\boldsymbol{\alpha}_t. \tag{24}$$

Adaptive kernel width and sparsity techniques in KNN method (AKNN_μ) which is described in Algorithm 1, are

Table 3: Evaluation of online learning algorithms with sparsing ability on the large datasets.

Algorithm	Football				Spambase			
	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)
Perceptron	21.493	6.002	21.493	41.043	25.325	8.179	25.325	40.881
ROMMA	45.605	12.550	20.282	40.905	46.360	15.474	23.403	39.708
ALMA	23.807	6.646	20.430	40.416	28.203	9.440	23.306	39.361
PA	49.531	13.864	20.256	40.694	54.262	17.776	23.757	38.731
DOUL	45.457	19.883	21.775	40.041	49.744	27.720	23.422	39.382
$AKNN_{\mu}$	**6.626**	**0.699**	**5.631**	**17.840**	**23.627**	**4.619**	**23.174**	**17.648**
Algorithm	MITFace				Mushrooms			
	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)
Perceptron	**20.764**	15.333	20.764	50.366	41.085	40.077	41.085	47.587
ROMMA	32.428	24.932	19.347	50.724	61.679	60.266	41.437	49.569
ALMA	21.138	15.822	18.888	50.365	43.330	42.153	40.501	45.335
PA	43.635	32.526	19.828	48.674	62.218	61.115	41.234	47.378
DOUL	39.419	49.529	19.279	50.193	63.467	192.678	41.075	47.082
$AKNN_{\mu}$	37.366	**9.903**	**7.420**	**16.038**	**0.480**	**0.442**	**9.992**	**39.931**

*Bold number shows the algorithm with best efficiency measurement in each dataset.

produced from combination of this online sparsification strategy with AKNN methods. This algorithm has high accuracy because errors of removed instances are committed in learning process, and it is capable of online application. Algorithm 1 shows that its complexity for each instance is $O(m)$, if m is the number of dictionaries.

4. Experiments and Results

Two experiments have been designed. In the first experiment effects of kernel width parameter on performance of the proposed KNN method with adaptive kernel width and the KNN method with fixed kernel width were demonstrated. In the second experiment, simulations on some classification problems to compare performance of the proposed method with other online classification methods were conducted.

4.1. Evaluation Effect of Parameter σ on the Fixed and Adaptive KNN Methods. This experiment is an example that visualizes effect of choosing initial σ on learning error. It was performed on an artificial two-dimensional (2D) dataset with 720 samples (Figure 2), whose design is complex. A comparison was carried out between the performance of the adaptive kernel width KNN method and the fixed kernel width KNN approach in the same conditions. The plot of Figure 3 depicts the average square error rate in both methods for small (Figure 3(a)) and large σ (Figure 3(b)). The changing process of σ in each time step of online learning algorithm in AKNN is plotted in Figure 4(a) (for small σ) and in Figure 4(b) (for large σ).

The empirical feature space preserves the geometrical structure of Φ in the feature space [11, 26]. Figure 5 gives the 2D projection of the training data (90% of data) in the empirical feature space when the Gaussian kernel function with different σ is employed. It is based on only the 2D projection of the embedding that means the first two largest eigenvalues of kernel Gram matrix. Classification results of testing data (10% of data) for three σ values are presented in Figure 6 that shows misclassified data by circles.

Figure 3 demonstrates the effects of initial σ change on the MSE changing process for both methods. We observe that AKNN achieves significantly smaller MSE than the KNN algorithm in both cases. This shows that the proposed AKNN approach is effective for different initial σ values and can reach appropriate σ value (from Figure 4). Achieved σ from AKNN approach for spiral dataset is about 0.14. In addition, effect of σ on result classification is shown clearly in Figure 6. It means that KNN too has misclassified data for inappropriate σ and least misclassified data for the best σ that the achieved from proposed method.

Figure 5 gets some intuitive feeling about the position of data into the empirical feature space. From Figure 5(b), it is seen that the embedded data are too dense in empirical feature space and they cannot explain class separability. In Figure 5(c) with large σ, the embedded data are not dense, but they have complex structure yet that cannot be separated by linear hyper plans. Opposed to the two mentioned cases, the embedded data have a good resolution and a linear structure that can be separated by linear hyper plans.

4.2. Performance Comparison of AKNN with Online Learning Methods. The experiments have been carried out to evaluate the performance of the proposed method in comparison with a number of online classification methods. They include the kernel perception algorithm [25], the aggressive version

Table 4: Evaluation of online learning algorithms on the small and medium datasets.

Algorithm	Sonar				Ionosphere			
	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)
Perceptron	35.722	**0.031**	35.722	39.952	**15.886**	**0.038**	20.675	48.117
ROMMA	69.946	0.066	33.048	39.405	68.829	0.084	22.829	42.260
ALMA	42.139	0.042	33.961	34.643	45.949	0.057	23.601	46.907
PA	77.112	0.069	33.636	37.571	67.785	0.094	**18.714**	45.470
DOUL	71.551	0.078	31.818	39.428	68.449	0.114	24.727	48.126
KNN	98.716	0.056	27.166	26.857	99.968	0.139	27.331	39.395
AKNN	100	0.059	28.343	25.357	95.032	0.132	32.894	39.361
$AKNN_\mu$	**20.267**	0.038	**25.561**	**24.024**	38.418	0.077	45.338	**38.269**
Algorithm	Pima				German			
	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)
Perceptron	30.679	0.313	30.679	45.955	40.289	0.637	40.289	35.0
ROMMA	51.185	0.527	30.824	45.957	99.922	1.574	34.555	33.7
ALMA	32.442	0.345	29.003	43.481	99.722	1.574	34.600	34.4
PA	61.257	0.609	29.176	44.400	99.911	1.578	34.544	33.3
DOUL	55.390	0.635	31.228	49.086	99.911	2.539	34.544	31.4
KNN	97.471	0.489	25.462	**23.689**	100	0.824	45.022	39.6
AKNN	94.957	0.494	**25.144**	24.735	100	0.876	31.733	**29.7**
$AKNN_\mu$	**26.084**	**0.237**	28.367	25.916	**3.422**	**0.169**	**30.189**	29.9
Algorithm	Splice				Cloud			
	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)	Density (%)	Training time (s)	Training mistake (%)	Test mistake (%)
Perceptron	**46.033**	0.715	46.033	49.3	**0.222**	**0.048**	0.222	25.243
ROMMA	99.611	1.538	43.389	46.4	14.248	1.191	0.108	25.441
ALMA	95.933	1.479	44.022	43.9	0.494	0.101	0.125	24.661
PA	98.455	1.507	43.378	45.5	3.076	0.332	0.108	25.292
DOUL	98.455	2.305	43.378	45.6	3.076	0.339	0.108	23.536
KNN	36.000	**0.176**	**13.967**	42.4	44.927	0.697	0.054	**0**
AKNN	92.000	0.765	39.289	41.7	44.927	0.737	0.108	**0**
$AKNN_\mu$	81.188	0.694	40.155	**41.5**	0.488	0.144	**0.103**	**0**

*Bold number shows the algorithm with the best efficiency measurement in each dataset.

of ROMMA [27], the $ALMA_p(\alpha)$ algorithm [28], and the passive-aggressive algorithms (PA) [29]. We test all the algorithms on ten real datasets which are listed in Table 2 and can be downloaded from LIBSVM, UCI, and MIT CBCL face websites.

To make a fair comparison, all algorithms adopt the same experimental setup. For all algorithms in comparison, we set the penalty parameter $C = 5$ and train all datasets with the Gaussian kernel with $\sigma = 8$. For the $ALMA_p(\alpha)$ algorithm, parameters p and α are set to 2 and 0.9, respectively. We fix sparsity threshold to be 0.7 (ρ in DUOL algorithm and μ in $AKNN_\mu$).

We scale all training and testing data to be in [0, 1]. Then we use a 10-fold cross-validation to estimate the efficiency measurements the same as *mistake rate, density rate,* and *running time*. The *mistake rate* evaluates the online learning performance that is the percentage of instances that are misclassified by the learning algorithm. We measure the sparsity of the learned classifiers by the *density rate* that is the percentage of remained instances (or support vectors in some of the methods). The *running time* evaluates computational efficiencies of all the algorithms. All the experiments are run in MATLAB over a windows machine.

Table 3 presents the result of comparing eight online learning algorithms over the large datasets. And Table 4 summarizes the performance of six compared online learning algorithms over the small and medium datasets. KNN and AKNN methods are not in Table 3 because they do not

have sparsification technique. The online learning efficiency measurements are presented in both tables.

According to the experimental results shown in Tables 3 and 4, we can see that

(i) although there is little difference in the training mistake rate among all methods, the proposed method has the best *training mistake rate* especially for large datasets;

(ii) the proposed method achieves significantly smaller *testing mistake rates* than the other online approaches except in pima dataset;

(iii) it can be seen that the perceptron, ALMA, and AKNN_μ methods return more density rate than other approaches especially for large datasets. That means they need to keep fewer training data. Therefore, we can say that these methods are faster than other approaches because in general the training time is proportional to the size of dictionary.

So, we can say that among all online learning, the AKNN_μ method yields the least *mistake rate* with the smallest density rate and running time for most of the cases. It happens because AKNN_μ method is able to find a proper kernel function by finding the best kernel parameters and it keep the least and the best possible instances that are selected by online sparsification method.

5. Conclusion and Future Works

The goal of the present paper was to present a novel adaptive kernel least-mean-square neural network method. This method (AKNN) has an adaptive kernel width and sparsification processes simultaneously. We briefly touched history of learning algorithms based on LS. Then, we proposed an adaptive kernel that iteratively decreases least mean square error in form (4) to select proper kernel width for Gaussian kernel. For improving the method in online application, we have used a sparsification methodology for controlling the model order increasing that its computational complexity is only linear in the dictionary size. By use of the acquired μ-coherence dictionary from the training samples, the algorithm needs less computational cost and memory requirement compared with conventional KNN. The conducted experiments show that varying σ did not affect the proposed algorithm. Experiments results also prove that the given algorithm has faster learning rate than conventional KNN algorithm and better classification performance than other online classification methods. However, selecting proper values for other parameters is still a challenge.

Our future works deal with the following questions: can we use a more efficient function as the sigmoid function in the KNN neuron? Can we adapt other kernel functions to use in neuron KNN? How to improve the performance of AKNN on imbalanced data and noisy data?

References

[1] B. Widrow, "Adaptive filters I: fundamentals," Tech. Rep. SEL-66-126 (TR-6764-6), Stanford Electronic Laboratories, Stanford, Calif, USA, 1966.

[2] J. Kivinenm, A. J. Smola, and R. C. Williamson, *Online Learning with Kernels*, IEEE, New York, NY, USA, 2004.

[3] P. P. Pokharel, L. Weifeng, and J. C. Principe, "Kernel LMS," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '07)*, pp. III1421–III1424, Honolulu. Hawaii, USA, April 2007.

[4] W. Liu, P. P. Pokharel, and J. C. Principe, "The kernel least-mean-square algorithm," *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 543–554, 2008.

[5] B. Sch?lkopf and A. J. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, Mass, USA, 2002.

[6] D. Erdogmus and J. C. Principe, "Generalized information potential criterion for adaptive system training," *IEEE Transactions on Neural Networks*, vol. 13, no. 5, pp. 1035–1044, 2002.

[7] R. Herbrich, *Learning Kernel Classifiers: Theory and Algorithms*, MIT Press, Cambridge, Mass, USA, 2002.

[8] V. Vapnik, *Statistical Learning Theory*, Wiley, New York, NY, USA, 1998.

[9] Q. Chang, Q. Chen, and X. Wang, "Scaling Gaussian RBF kernel width to improve SVM classification," in *Proceedings of the International Conference on Neural Networks and Brain Proceedings (ICNNB '05)*, pp. 19–22, Beijing, China, October 2005.

[10] Y. Baram, "Learning by kernel polarization," *Neural Computation*, vol. 17, no. 6, pp. 1264–1275, 2005.

[11] H. Xiong, M. N. S. Swamy, and M. O. Ahmad, "Optimizing the kernel in the empirical feature space," *IEEE Transactions on Neural Networks*, vol. 16, no. 2, pp. 460–474, 2005.

[12] A. Singh and J. C. Príncipe, "Kernel width adaptation in information theoretic cost functions," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '10)*, pp. 2062–2065, Dallas, Tex, USA, March 2010.

[13] J. A. K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle, "Weighted least squares support vector machines: robustness and sparce approximation," *Neurocomputing*, vol. 48, pp. 85–105, 2002.

[14] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machine*, World Scientific Publishing, River Edge, NJ, USA, 2002.

[15] B. J. De Kruif and T. J. A. De Vries, "Pruning error minimization in least squares support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 696–702, 2003.

[16] G. C. Cawley and N. L. C. Talbot, "Improved sparse least-squares support vector machines," *Neurocomputing*, vol. 48, pp. 1025–1031, 2002.

[17] L. Hoegaerts, *Eigenspace methods and subset selection in kernel based learning [Ph.D. thesis]*, Katholieke Universiteit Leuven, Leuven, Belgium, 2005.

[18] L. Hoegaerts, J. A. K. Suykens, J. Vandewalle, and B. De Moor, "Subset based least squares subspace regression in RKHS," *Neurocomputing*, vol. 63, pp. 293–323, 2005.

[19] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[20] P. P. Pokharel, W. Liu, and J. C. Principe, "Kernel least mean square algorithm with constrained growth," *Signal Processing*, vol. 89, no. 3, pp. 257–265, 2009.

[21] S. Van Vaerenbergh, J. Vía, and I. Santamaría, "A sliding-window kernel RLS algorithm and its application to nonlinear channel identification," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, pp. V789–V792, May 2006.

[22] P. Honeine, C. Richard, and J. C. Bermudez, "On-line nonlinear sparse approximation of functions," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT '07)*, pp. 956–960, Nice, France, June 2007.

[23] C. Richard, J. C. M. Bermudez, and P. Honeine, "Online prediction of time series data with kernels," *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 1058–1067, 2009.

[24] H. J. Bierens, "Introduction to Hilbert Spaces," Lecture notes, 2007.

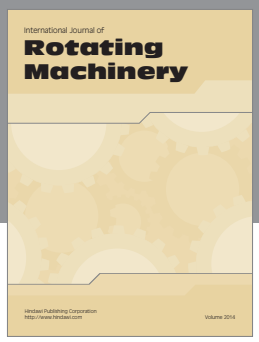
[25] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

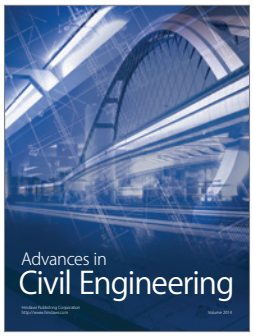
[26] B. Schölkopf, S. Mika, C. J. C. Burges et al., "Input space versus feature space in kernel-based methods," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

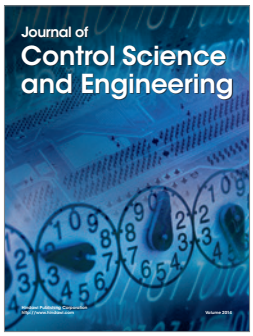
[27] Y. Li and P. M. Long, "The relaxed online maximum margin algorithm," *Machine Learning*, vol. 46, no. 1–3, pp. 361–387, 2002.

[28] C. Gentile, "A new approximate maximal margin classification algorithm," *Journal of Machine Learning Research*, vol. 2, pp. 213–242, 2002.

[29] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *Journal of Machine Learning Research*, vol. 7, pp. 551–585, 2006.







Hindawi

Submit your manuscripts at
http://www.hindawi.com