Classifying With Adaptive Hyper-Spheres: An Incremental Classifier Based on Competitive Learning

Tie Li, Gang Kou, Yi Peng, and Yong Shi

***Abstract*—Nowadays, datasets are always dynamic and patterns in them are changing. Instances with different labels are intertwined and often linearly inseparable, which bring new challenges to traditional learning algorithms. This paper proposes adaptive hyper-sphere (AdaHS), an adaptive incremental classifier, and its kernelized version: Nys-AdaHS. The classifier incorporates competitive training with a border zone. With adaptive hidden layer and tunable radii of hyper-spheres, AdaHS has strong capability of local learning like instance-based algorithms, but free from slow searching speed and excessive memory consumption. The experiments showed that AdaHS is robust, adaptive, and highly accurate. It is especially suitable for dynamic data in which patterns are changing, decision borders are complicated, and instances with the same label can be spherically clustered.**

***Index Terms*—Adaptive algorithms, Nyström method, pattern clustering, self-organizing feature maps (SOFMs).**

I. Introduction

ADAPTIVE incremental learning, also called online learning, aims to handle dynamic data arriving in real-time [1]. Data in the real world are not always processed all at once, such as real-time financial data analysis, network intrusion detection, and dynamic Webpages mining [2]–[4].

Zhou and Chen [5] proposed three types of incremental learning: 1) example-incremental learning; 2) class-incremental learning; and 3) attribute-incremental learning (A-IL). The first two types of incremental learning with assumption that the attributes are fixed have been studied more than A-IL [6]–[9].

With the availability of increasing amount of dynamic data, a great deal of researches on incremental learning have been conducted in recent years [6]–[10]. Some of them categorize the modeling strategies of online learning into statistical learning models and adversarial models. A latent assumption of statistical learning models is that variables are "independent and identically distributed." However, patterns are evolving and changing over time, and the statistical assumptions are hard to be satisfied [10]. For example, fraudulent information and censorship are adversarial in credit risk assessment. Fraud and anti-fraud strategies are evolving interactively, which results in changing patterns [11]. Analogous phenomenon can also be found in stock market and network intrusion detection. In practice, given the complexity of data sources, uniform patterns do not always exist across the entire dataset. Specific patterns only fit in certain parts of the datasets at a particular time. So it is necessary for learning algorithms to adapt to new patterns in dynamic data.

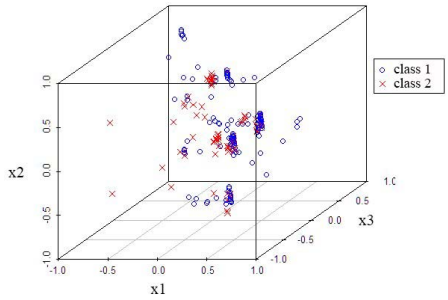


Fig. 1. Data points that are not linearly separable.

Fig. 1 shows an example of changing patterns in different areas of a dataset.

Some feasible solutions to identify changing patterns in dynamic data are described as follows.

1) *Instance-Based Learning:* A well-known instance-based learning algorithm is k-nearest neighbors (k-NNs). But the problem with k-NN is that the search time is usually unbearable and the memory consumption is too high in large-scale data applications [12]. Though indexing technologies such as ball-tree, KD-tree, R-tree, locality sensitive hashing, and other hashing technologies [13] can make the search of the similar points faster, the model of k-NN is too simple and lacks characterization of the data distribution [14].
2) *SVM and Kernel Methods:* SVMs use hyper-planes to divide space [15]. However, hyper-planes are not

Manuscript received October 31, 2016; accepted September 26, 2017. Date of publication October 24, 2017; date of current version March 17, 2020. This work was supported by the National Natural Science Foundation of China under Grant 71325001 and Grant 71771037. This paper was recommended by Associate Editor Z. Liu. *(Corresponding author: Yi Peng.)*

T. Li and Y. Peng are with the School of Management and Economics, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: lteb2002@163.com; pengyi@uestc.edu.cn).

G. Kou is with the School of Business Administration, Southwestern University of Finance and Economics, Chengdu 611130, China (e-mail: kougang@swufe.edu.cn).

Y. Shi is with the Key Laboratory of Big Data Mining and Knowledge Management, Chinese Academy of Sciences, Beijing 100190, China (e-mail: yshi@ucas.ac.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2761360

adaptive enough to divide space with highly complicated data distributions (such as Fig. 1). In fact a decision border of hyper-plane is not realistic in most cases. SVMs rely on kernel tricks to project instances into a reproducing kernel Hilbert space [16]. The problem is that kernel methods are hard to be applied for large-scale datasets because the time cost of computing the kernel matrix is $O(n^2)$. Furthermore, the search of the optimal parameters for kernel functions is also time consuming.

Given the limitations of hyper-planes, a straightforward intuition is to use hyper-spheres to divide the space [17]. Many methods, such as support vector data description (SVDD), ball trees, competitive learning, and clustering algorithms, explicitly or implicitly, use hyper-spheres to divide the space.

SVDD is a representative model that uses mathematical method to obtain an appropriate hyper-sphere. It builds a minimum radius hyper-sphere around the data. The primal form of the optimization problem of SVDD [18] is

$$\begin{aligned} &\min \ R^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t.} \ \ \|x_i - \alpha\|^2 \le R^2 + \xi_i, \xi_i \ge 0 \end{aligned} \tag{1}$$

where $x_i \in \mathbb{R}^m$, $i = 1, \ldots, n$, is the training data; R represents the radius; ξ_i is the slack variables; α is the center of the hyper-sphere; and C is the penalty constant. SVDD was first introduced for single class classification and outlier detection [18] and then many improvements had been made for multiclass classification by building one hyper-sphere for each class [19]. The dual problem of SVDD can be expressed as inner-product form. When the data distribution is complex, SVDD also uses kernel methods to project the instances into a reproducing kernel Hilbert space, in which instances of a particular class are more likely to be enclosed by one single hyper-sphere [18].

The main advantage of SVDD is that it can be solved via mathematical optimization method and easy to use kernel tricks. The limitations of SVDD include the following.

1) There is only one hyper-sphere for each class. If the data distribution of the same class is complex, one hyper-sphere is obviously not enough [19].
2) It is hard for SVDD to determine the number of hyper-spheres adaptively.

1) *Clustering-Based Classification Models:* Numerous studies revealed that there is a connection between clustering and classification [17], [20], [21]. Such studies include radial basis function networks (RBFNs) [20] and functional link neural network [21]. RBFN is a clustering-classification style neural network classifier and has its incremental version IRBFN, but the number of clusters is fixed and this limits its adaptivity [20]. Adaptive resonance theory (ART) can use clusters for classification, add clusters adaptively, and be trained incrementally [22].
2) *Competitive Learning:* ART is a type of competitive neural networks. It tries to fit each new input pattern in an existing subclass. If no matching subclass can be found, a new subclass is created containing the new pattern.

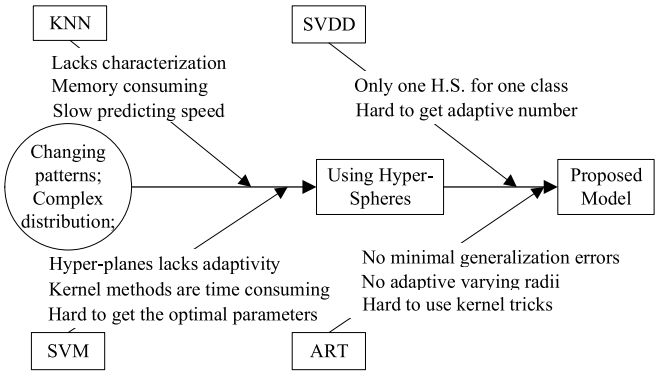


Fig. 2. Causations of the proposed model.

In the past two decades, extensive research has been conducted regarding ART, including ART-1, ART-2, ART-3, fuzzy ART, and ARTMAP [23]–[25]. The primary goal of ARTMAP, fuzzy ARTMAP, and Gaussian ARTMAP, is to solve the plasticity–stability dilemma [25]. In general, the structure of ART is too complicated for most cases.

ART, counter propagation network (CPN), [26], [27] and learning vector quantization [28] are usually used for classification or supervised clustering. They are partially based on self-organizing feature map (SOFM), namely "Kohonen learning" [29], which is an incremental clustering algorithm and can be trained in linear time. Xiao and Chaovalitwongse [30] proposed analogous model based on k-NN, and the hyper-spheres are referred as "prototypes." Valente and Abrão [31] proposed a multi-input multioutput transmit scheme based on morphological perceptron. Dai and Song [32] proposed a supervised competitive learning algorithm for the generation of multiple classifier systems.

The main criticism of competitive learning was that the accuracy was not as high as mathematical optimization ones, especially when the size of a dataset is small. In incremental learning, most mathematical optimization problems resort to stochastic gradient descent (SGD) or its variants, as the substitute for gradient descent methods, and SGD suffers from "regret error" [33]. Other disadvantages of competitive learning include the following.

1) They do not have a mechanism to minimize generalization errors, which may cause over-fitting.
2) Unlike SVDD, they lack the explicit definition of each hyper-sphere's varying boundary, or "decision border."
3) It is difficult to denote them in inner-product forms, and thus hard to use kernel tricks.

Given the limitations of existing models, this paper proposes a new model based on the following sequence of causations (Fig. 2).

The purpose of this paper is to propose a new incremental learning model which can be used for the aforementioned complicated dynamical scenario. Adaptability, locality, and bounded memory consumption are the key requirements.

Assumption 1: In order to bind this paper within a specific framework, we make the following assumptions.

1) *Regarding Clustering:* Data distribution is always complex, with instances consisting of different labels

TABLE I
COMPARISON OF THE HYPER-SPHERES-BASED MODELS

	ISVDD	IRBFN	CPN	ART	Proposed
Incremental learning method	SGD	SGD	C.L.	C.L.	C.L.
Number of Hyper-spheres	1/class	Many	Many	Many	Many
Adaptive hyper spheres number	No	No	No	Yes	Yes
Boundary/Radii definition	Yes	No	No	Yes	Yes
Adaptive radii	No	No	No	No	Yes
Explicit hyper sphere definition	Yes	No	No	No	Yes
Splitting and merging	No	No	No	No	Yes
Clustering-based	No	Yes	Yes	Yes	Yes
Kernelization	Yes	Yes	No	No	Yes
Training time	High	High	Low	Low	Low
Applications	Outlets detection	Classification;	Classification;	Classification;	Classification;

C.L. refers to "competitive learning".

intertwined. There are usually a large number of clusters containing instances with the same labels in local dense areas. Local clustering is practical in most cases and easy to accomplish using methods such as local distance metrics learning [34].

2) *Regarding Local Consistency:* We assume that instances near to one another tend to have the same label. This assumption is used in many semisupervised learning research studies [35] and conforms to basic theories of local models, such as RBFN and k-NN.

Contributions: The main contributions of the proposed model are as follows.

1) It utilizes the adaptivity of competitive neural networks while recognizes decision borders of data points like SVM.
2) It can be incrementally trained and does not require retraining when new patterns emerge. It can be applied to datasets that are not linearly separable and maintains reasonable memory consumption.
3) It can apply feasible kernel methods on small- and large-scale datasets.

The main differences of the proposed model with the existing ones are summarized in Table I.

The remainder of this paper is organized as follows. Section II describes the basic theory, including the analysis of competitive learning and kernel methods. Section III represents the proposed algorithms. Section IV reports the results and discussion of the experiments. Section V concludes this paper with conclusion and future works.

II. BASIC THEORY

A. Basic Theory of Supervised Competitive Learning

We partially borrow the topological structure of CPN to introduce our model. CPNs are a combination of competitive networks and Grossberg's [22] outstar networks. The topological structure of CPN has three layers: 1) input layer; 2) hidden layer; and 3) output layer (Fig. 3).

Suppose there are N elements in the input layer, M neurons in the hidden layer, and L neurons in the output layer. Let vector $V_i = (v_{i1}, \ldots, v_{iN})^T$ denote the weights of neuron i in the hidden layer connecting to each of the elements of the input

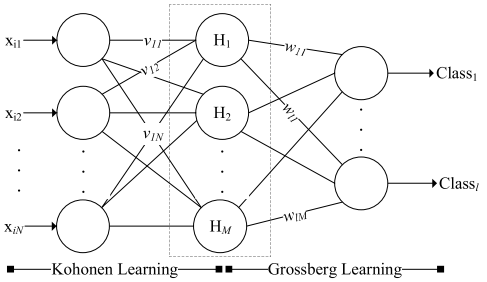


Fig. 3. Topological structure of CPN.

layer. Then $V = (V_1, \ldots, V_M)$ denotes weight matrix of the instars. If the training in stage 1 can be viewed as a clustering process, then neuron i is cluster c_i and V_i is the centroid of cluster c_i.

When an instance is coming, it will compute the proximity between the instance and each V_i in the weight matrix, i.e., the centroid of cluster c_i. Here, proximity can be measured by computing inner product $\text{net}_j = V_j^T x$, $(j = 1, 2, \ldots, m)$. It adopts a winner-takes-all strategy to determine which neuron's weights are to be adjusted. The winner is $\text{net}_{j*} = \max\{\text{net}_j\}$. In other words, the winner is c_{j*} whose centroid is the closest to the incoming instance. The winning neuron's weights would be adjusted as follows:

$$V_{j*}(t+1) = V_{j*}(t) + \alpha\big[x - V_{j*}(t)\big] \tag{2}$$

where α is the learning rate, indicating that the centroid of the winning cluster will move in the direction of x. As instances keep coming, the weights vector—i.e., the centroid of the hyper-spheres—tend to move toward the densest region of the space. This first stage of the CPN's training algorithm is a process of self-organizing clustering, although it is structured using a network.

The second part of the structure is a Grossberg [22] learning. We will redesign a different hidden layer and different connection from the hidden layer to the output layer.

B. Advantages and Disadvantages of the Original Model

To illustrate the advantage and disadvantage of original model, a set of 2-D artificial data were created and visualized in Fig. 4.

In Fig. 4(a), instances can be grouped into six clusters. Setting the number of neurons in the hidden layer to six, the first training stage of the model in Fig. 3 can automatically find the centroids of the six clusters, which are represented by the weights of the six neurons. The second training stage can learn each cluster's connection to the right class. The distance from each instance in Fig. 4(a) to its cluster centroid is smaller than the distances to the centroids of other clusters. The dataset shown in Fig. 4 is ideal for CPN to classify.

Data distribution in Fig. 4(a) is simplified and idealistic. Data with distribution similar to Fig. 4(b) will cause two kinds of problems to the original model.

1) First, the self-organized clustering process depends on the similarity measures between data points and hyper-sphere's centroid. Points closer to one cluster's

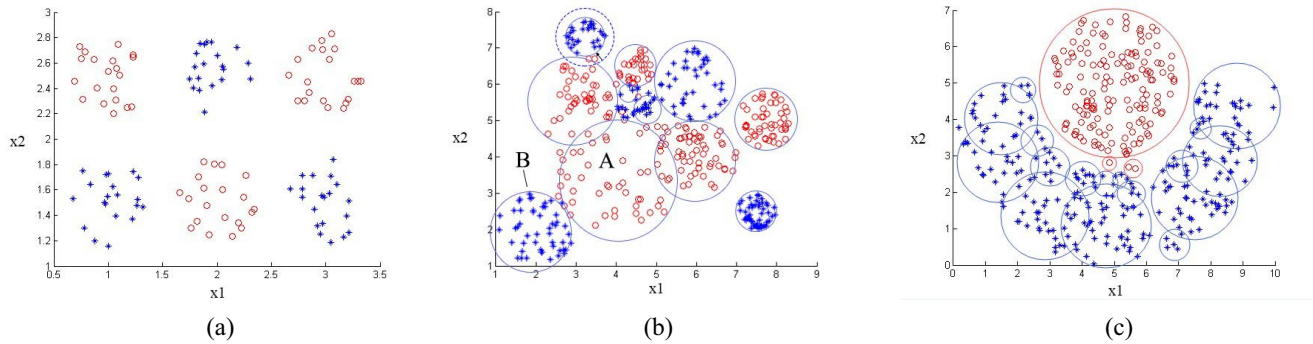


Fig. 4. Artificial datasets and the proposed clustering solutions.

centroid may belong to another cluster. Therefore, every cluster should have a definite scope or radius, and the scope should be as far away from others as possible.

2) Second, the number of clusters in the hidden layer is fixed in the original model. However, it is difficult to estimate the number of clusters in advance. Given different numbers of neurons in the hidden layer, the accuracy varies dramatically. The training of the instar layer—i.e., the clustering process—is contingent on this fixed number.

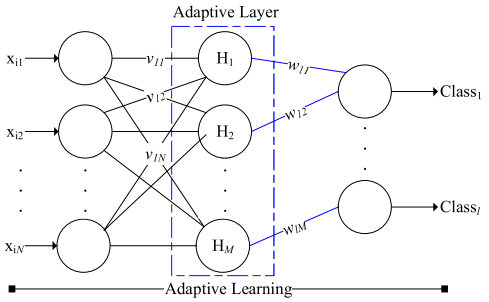


Fig. 5. Topological structure of the proposed model.

C. Building of the DMZ

To solve the first aforementioned problem, we should have a general knowledge of the scope of the clusters. For example, points of cluster A [in Fig. 4(b)] near the border may be closer to the centroid of cluster B, so these points will be considered belong to cluster B in the original model. We must identify the decision border that separates clusters according to their labels. When two instances with conflicting labels fall into the same cluster, it gives us an opportunity to identify the border point that is somewhere between the two conflicting instances (as long as the instance is not an outlier). To maintain the maximum margin and for the sake of simplicity, the median point of two instances could be selected as a point in a zone called a demilitarized zone (DMZ), and clusters should be as far away from the DMZ as possible. As the number of conflicting instances increases, a general zone gradually forms as the DMZ. This mechanism can find borders of any shapes that are surrounded by many hyper-spheres.

To solve the second problem, the number of clusters should not be predetermined. The clusters should be formed dynamically and merged or split if necessary. The scope of the hyper-spheres, represented by the corresponding radii, should be adjusted on demand. As an example, consider the situation presented in Fig. 4(b): with instances of conflicting labels found in the top cluster, the original cluster should tune its radius. After training, a new cluster would be formed beneath the top cluster containing instances of different labels from the ones in the top cluster. The radii of the two clusters should be tuned according to their distance to the borders.

One single hyper-sphere may not enclose an area whose shape is not hyper-spherical [36]. However, any shape could be enclosed as long as the number of the formed hyper-spheres is unlimited. Consider the clusters represented by the 2-D circles in Fig. 4(c). All of the instances can be clustered no matter what the data distribution is and what the shape of the border is, as long as there are enough hyper-spheres of varying radii and are properly arranged.

D. Proposed Topological Structure

Given the solutions above, the structure of our improved model is shown in Fig. 5.

The first difference is that our model has an adaptive dynamic hidden layer and the number of neurons in hidden layer is adaptive. The second difference is that each neuron H_i connects to only one particular neuron in the output layer, and w_{ij} is used to record the radius of neuron H_i.

E. Kernelization

It is challenging for competitive learning models to apply kernel methods because they cannot be denoted in inner-product forms. Some previous studies use approximation methods for the kernelization of competitive learning [37], [38]. This paper uses Nyström method to kernelize the proposed model [39], [40].

Let the kernel matrix written in blocks form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}. \tag{3}$$

Let $C = [A_{11} \quad A_{21}]^T$, Nyström method uses A_{11} and C to approximate large matrix A. Suppose C is a uniform sampling of the columns, Nyström method generates a rank-k

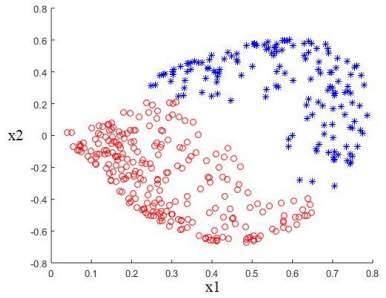


Fig. 6. Artificial dataset 3 after Nyström and SVD transformation.

approximation of $A(k \leq n)$ and is defined by

$$A_k^{\text{nys}} = CA_{11}^{+}C^T = \begin{bmatrix} A_{11} & A_{21} \\ A_{21} & A_{21}A_{11}^{+}A_{21}^T \end{bmatrix} \approx A \tag{4}$$

where A_{11}^{+} denotes the generalized pseudo inverse of A_{11}.

There exists an Eigen decomposition $A_{11}^{+} = V\Lambda^{-1}V^T$, such that each element ${A_k}^{\text{nys}}{}_{ij}$ in A_k^{nys} can be decomposed as

$$\begin{aligned} {A_k}^{\text{nys}}{}_{ij} &= \left(C_i^T V \Lambda^{-1} V^T C_j\right) \\ &= \left(\Lambda^{-1/2} V^T C_i\right)^T \left(\Lambda^{-1/2} V^T C_i\right) \\ &= \left(\Lambda^{-1/2} V^T (\kappa(x_i, x_1), \ldots \kappa(x_i, x_m))\right)^T \\ &\quad \bullet \left(\Lambda^{-1/2} V^T \left(\kappa\left(x_j, x_1\right), \ldots \kappa\left(x_j, x_m\right)\right)\right) \end{aligned} \tag{5}$$

where $\kappa(x_i, x_j)$ is the base kernel function, $x_1, x_2, \ldots, x_m$ are representative data points and can be obtained by uniform sampling or clustering methods such as K-means and SOFM.

Let

$$\underset{m}{\widetilde{\phi}}(x) = \Lambda^{-1/2} V^T (\kappa(x, x_1), \ldots, \kappa(x, x_m))^T \tag{6}$$

such that

$$A_{k_{ij}}^{\text{nys}} = \underset{m}{\widetilde{\phi}}(x_i)^T \underset{m}{\widetilde{\phi}}\left(x_j\right) = \widetilde{\kappa}\left(x_i, x_j\right). \tag{7}$$

With Nyström method, we can get an explicit approximation of the nonlinear projection $\phi(x)$, which is

$$x \to \underset{m}{\widetilde{\phi}}(x). \tag{8}$$

To justify why we use kernel methods for our model, we first used Nyström method to raise the dimension of dataset 3 to 403, then used singular value decomposition (SVD) to reduce the dimension to 2 for the purpose of visualization. Fig. 6 illustrates the transformed dataset 3 from Fig. 4(c).

Compared with Fig. 4(c), the data in Fig. 6 can be covered with less hyper-spheres, or each hyper-sphere can enclose more data points. Because the sampling points in Nyström methods can be obtained dynamically, the projection of (8) can be used for every single instance in competitive learning and can be applied directly to our incremental model.

Without loss of generality, we use $\phi(x)$ to denote a potential projection of x in the reminder of this paper. If it works in the original space, the projection of x is to itself.

III. Proposed Classifier: AdaHs

The main characteristic of the proposed model is to adaptively build hyper-spheres. Therefore, we call the model adaptive hyper-spheres (AdaHSs), and the version after Nyström projection is called Nys-AdaHS.

A. Training Stages

Our algorithms are trained in three stages, which are described below.

Stage 1 (Forming Hyper-Spheres and Adjusting Centroids and Radii):

1) *Forming Hyper-Spheres and Adjusting Centroids:* Given that instances are read dynamically, there is no hyper-sphere at the beginning. The first instance inputed forms a hyper-sphere whose centroid is itself and initial radius is set to a large value. When a new instance is inputed and does not fall into any existing hyper-spheres, a new hyper-sphere will be formed in the same way. If a new instance falls into one or more existing hyper-spheres, the winner is the one whose centroid is the closest to the new instance. The winning cluster's centroid is recalculated as

$$c_i(t+1) = c_i(t) + \alpha[\phi(x) - c_i(t)] \tag{9}$$

where x is the new inputed instance, $c(t)$ is the original centroid of the hyper-sphere, $c(t+1)$ is the new centroid, and α is the learning rate. When the number of instances that fall within a particular hyper-sphere grows, its centroid tends to move toward the densest zone. In order to speed up the search of the winner, we build simple k-dimension trees for all hyper-spheres. With the knowledge of the radius, it is easy to figure out the upper and lower bounds of the selected k dimensions. In this way, it avoids extensive computation of all Euclidean distance of instance and hyper-sphere pairs.

2) *Building Decision Border Zone—DMZ:* The goal of this step is to find the DMZ's median points that approximate the shape of the DMZ. We find the points using the following technique. The first time a labeled instance falls into a hyper-sphere, the hyper-sphere will be labeled using the label of this instance. If another instance with a conflicting label falls into the same hyper-sphere, it indicates that the hyper-sphere has entered the DMZ. We identify the nearest data point in the hyper-sphere to the newly inputed conflicting instance, and let p_i represent the median point as follows:

$$p_i = \frac{1}{2}\left(\phi(x_{\text{conflicting}}) + c_i\right) \tag{10}$$

where $\phi(x_{\text{conflicting}})$, $p_i \in c_i$, and p_i is recorded and used in the posterior clustering process.

3) *Adjusting the Radii of Hyper-Spheres:* Once a DMZ point is found in a hyper-sphere, the radius of the hyper-sphere should be updated such that it does not enter the DMZ. The new radius of hyper-sphere c_i should therefore be set as

$$r_i = d(p_i, c_i) - d_{\text{safe}} \tag{11}$$

Algorithm 1 Forming of Hyper-Spheres and the Adjusting of the Centroids and Radii

Input:
x, the newly read instance;
Output:
C : A set of hyper-spheres whose centroids and radii are tuned properly;
DMZ : A set of points who shape the decision border approximately.
Method:
(1) $c_t = Null$, $len = -1$;
(2) **For Each** c_i **in** C
(3) **If** $\phi(x)$ falls into c_i
//Find the winner of the hyper-spheres
(4) **If** $label(x) = label(c_i)$ **and** ($len = -1$ **or** $d_E(\phi(x), c_i) < len$)
(5) $c_t = c_i$; //Store the present temporary nearest hyper-sphere
(6) $len = d_E(\phi(x), c_i)$;//Store the present temp nearest distance
(7) **Else If** $label(x) \neq label(c_i)$ //Split the hyper-sphere
(8) $p_i = \frac{1}{2}(\phi(x_{conflicting}) + c_i)$, $\phi(x_{conflicting}), p_i \in c_i$;
(9) Add p_i to DMZ;
(10) $r_i = d(p_i, c_i) - d_{safe}$; //Adjusting radii r_j of hyper-sphere c_j;
(11) Mark c_i as "support hyper-sphere";
(12) **End If**
(13) **End If**
(14)**End For**
(15)**If** $c_t \neq Null$
// Adjust the winning hyper-sphere's centroid
(16) $c_i(t+1) = c_i(t) + \alpha[\phi(x) - c_i(t)]$;
(17)**Else**
(18) Form a new hyper-sphere, and make $\phi(x)$ be the centroid;
(19) Let the label of the new hyper-sphere be $label(x)$.
(20)**End If**

Algorithm 2 Merging of Hyper-Spheres

Input:
C : A set of hyper-spheres which are formed in stage 1;
Output:
C : The remaining hyper-spheres after merging.
Method:
(1)**For Each** c_i **in** C
(2) **For Each** c_j **in** C **except** c_i
(3) $c_{big} = \max_{radius}(c_i, c_j)$, $c_{small} = \min_{radius}(c_i, c_j)$,$d_t = d(c_{big}, c_{small})$;
(4) **If** $d_t + r_{small} <= r_{big} + \theta \times r_{small}$ //θ is the merging coefficient
(5) Merge c_i and c_j;
(6) **End If**
(7) **End For**
(8)**End For**

where d_{safe} represents a safe distance at which a hyper-sphere should be from the closest DMZ point.

The logics of this stage are outlined in Algorithm 1 below.

Stage 2 (Merging Hyper-Spheres): Hyper-spheres may overlap with one another or even be contained in others. Therefore, after certain period of training, a merging operation should be performed. Suppose that we have two hyper-spheres, c_A and c_B, and the radii of them are not the same. Let $c_{\text{big}} = \max_{\text{radius}}(c_A, c_B)$, $c_{\text{small}} = \min_{\text{radius}}(c_A, c_B)$, $d_t = d(c_{\text{big}}, c_{\text{small}})$, and θ be the merging coefficient. If $d_t + r_{\text{small}} <= r_{\text{big}} + \theta \times r_{\text{small}}$, the prerequisite to merge is met. Then let $r_{\text{temp}} = d_t + r_{\text{small}}$, and the new radius of the c_{big} will be $r_{\text{new}} = \max(r_{\text{temp}}, r_{\text{big}})$.

The details of this stage are outlined in Algorithm 2.

Stage 3 (Selecting Hyper-Spheres): Since the training process is entirely autonomous, the number of generated hyper-spheres could be large. Therefore, the final stage needs to select hyper-spheres.

There are three types of hyper-spheres that are prominent, which are described as follows.

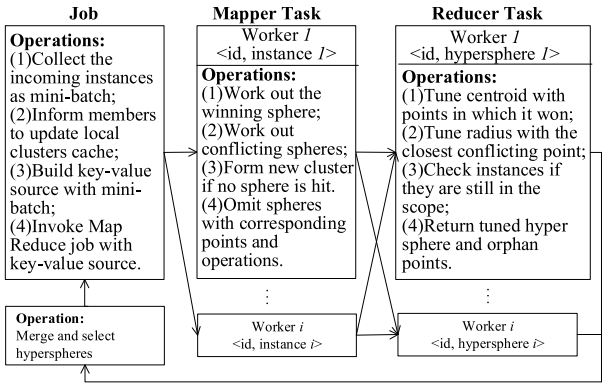


Fig. 7. MapReduce computing model.

Algorithm 3 Selection of Hyper-Spheres

Input:
C : The set of hyper-spheres which are formed in preceding stages;
Output:
C : The remaining hyper-spheres after selection.
Method:
(1)**For Each** c_i in C
// T is the threshold of the instances number which one hyper-sphere must at least have.
// $num(c)$ is a function computing the number of instances in a hyper-sphere.
(2) **If** $num(c_i) < T$
//Let $d(c_i, DMZ)$ be the distance from the centroid of c_i to the nearest data point in DMZ
(3) **If** $r_i < d(c_i, DMZ)$
(4) Discard c_i;
(5) **End If**
(6) **Else**
(7) Mark c_i as "core hyper-sphere";
(8) **End If**
(9)**End For**

1) The first type of hyper-spheres includes large number of instances. Because these are the fundamental hyper-spheres that contain most data points, they are marked as "core hyper-spheres."
2) The second type of hyper-spheres has less instances but locates near the border. They are marked as "support hyper-spheres" because such hyper-spheres can be found by measuring the distance between hyper-spheres and the nearest DMZ points.
3) The third type of hyper-spheres has small number of instances and is far away from the border. These hyper-spheres can be discarded.

To achieve high classification accuracy, both core hyper-spheres and support hyper-spheres should be selected. The logic of the third stage is outlined in Algorithm 3.

B. Mini-Batch Learning and Distributed Computing

To make it applicable in large-scale applications, we encapsulate the proposed algorithms into a MapReduce framework. We can collect the incoming instances as mini-batch set and then train them in MapReduce tasks. The computing model of the algorithms is illustrated in Fig. 7.

The collected mini-batch instances can be encapsulated in key-value pairs and mapped into mapper tasks.

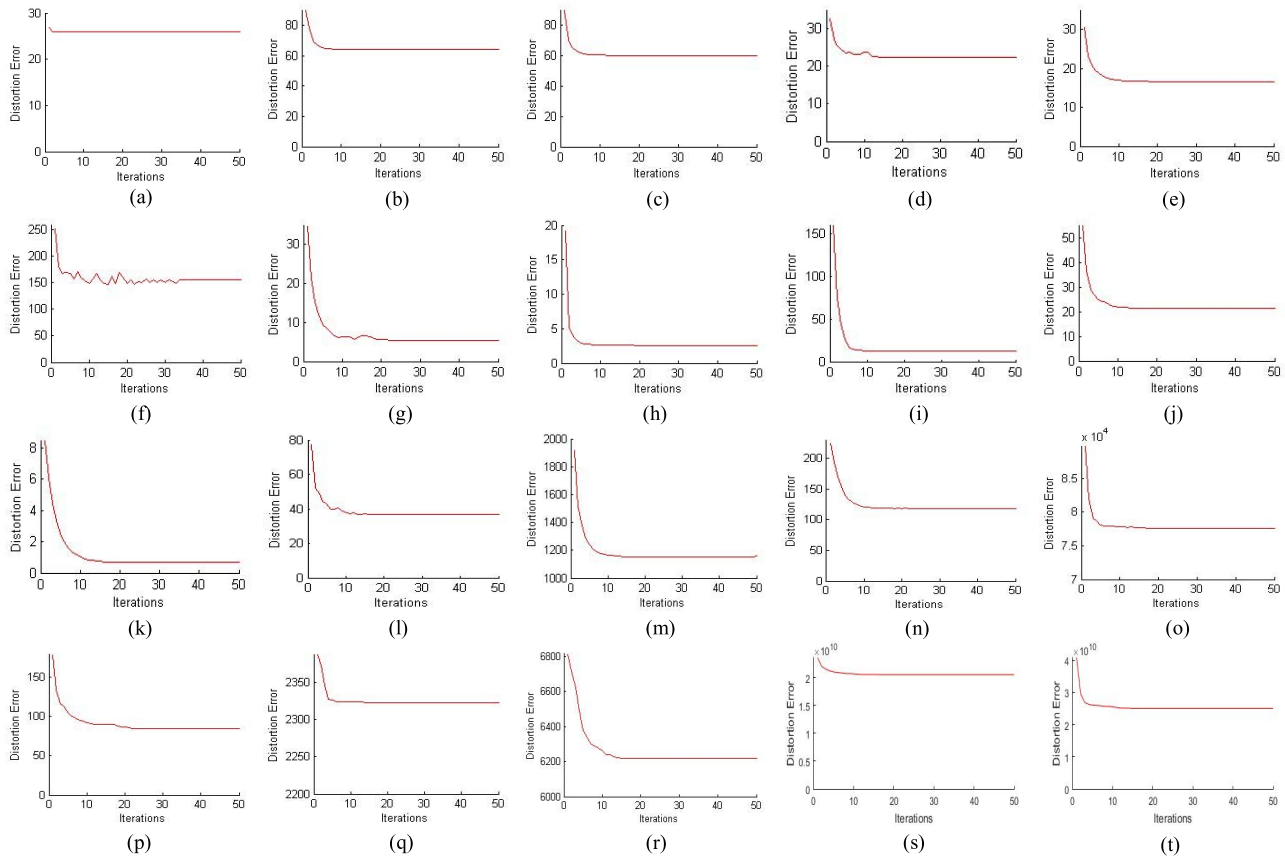


Fig. 8. Convergence test on all of the datasets. (a) Dataset 1. (b) Dataset 2. (c) Dataset 3. (d) Iris. (e) Seeds. (f) Segment. (g) Wholesale. (h) Glass. (i) Diabetes. (j) Wine. (k) Credit-g. (l) Credit-rating. (m) Phishing websites. (n) Credit card. (o) Pendigits. (p) Shuffle. (q) Occupation. (r) HAPT. (s) Loans. (t) URLs.

In each mapper tasks, the operations are based on instances. It queries local cache for every instance to find out in which hyper-spheres the instance falls, marks the winning hyper-sphere and the conflicting ones, and sends the hyper-spheres along with the description of the needed operations in another form of key-value<id, hyper-sphere> pairs.

In each reducer task, the operations are based on every hyper-sphere, which is aggregated according to the hyper-sphere id emitted from mapper tasks. The competitive learning can be conducted collectively with the aggregated instances. The tuning of a radius can be performed for only once with the closest conflicting instance, and it should find out the orphan points and return the tuned hyper-sphere at the end.

After a turn of the MapReduce tasks, the merging and selection of the hyper-spheres should be performed. After all of the operations, the tuned hyper-spheres should be saved to the cache. The orphan points should be retrained in the next turn. In the whole MapReduce process, subtasks do not coordinate with each other. Thus the hyper-spheres and DMZ are not updated in real time in a mini-batch turn, and they are updated collectively after all reducer tasks return.

C. Predicting Labels

Just like other supervised competitive neural networks, AdaHS must determine the winning hyper-sphere in the hidden layer to predict the label of a new instance. There are two situations. In the first situation, the new instance falls into an existing hyper-sphere and the label of the instance is determined by the label of the hyper-sphere. In the second situation, the new instance does not fall into an existing hyper-sphere, and the label of the new instance is coordinated by the k nearest hyper-spheres' labels

$$y = \arg\max_{l_j} \sum_{c_i \in N_k(x)} w_j I\big(y_i = l_j\big) \tag{12}$$

where $w_j = \exp(-([d_E(\phi(x), c_j)^2]/[2r_j^2]))$; $i = 1, 2, \ldots, L$; $j = 1, 2, \ldots, k$; $N_k(x)$ is the k nearest hyper-spheres; and I is the indicator function. The default value of k is set to 3.

IV. Experiments

We implemented our classifier using Java, with the help of third-party Jars including common-math3, weka, joptimizer, and a local caching framework. The distributed MapReduce implementation of AdaHS was built upon Hazelcast [41], which also provides distributed caching system. Most experiments were conducted on computer of i7-4560U (4CPU, 2-GHz), 8-GB RAM, and Ubuntu OS. The distributed deploy of AdaHS was conducted on a cluster of two and four machines, respectively, using the same configuration.

A. Benchmark Datasets

To evaluate the AdaHS, we used 20 datasets as the benchmarks. Among them, three were the 2-D artificial datasets

TABLE II
DETAILS OF THE DATASETS

Data set	Cla.	Attr.	Ins.	Class Distribution
Data set 1	2	2	120	{60, 60}
Data set 2	2	2	430	{230, 200}
Data set 3	2	2	403	{153, 250}
Iris	3	4	150	{50, 50, 50}
Seeds	3	7	210	{70, 70, 70}
Segment	7	19	1500	{205, 220, 208, 220, 204, 236, 207}
Wholesale	2	6	440	{298,142}
Glass	6	9	214	{70, 76, 17, 13, 9, 29}
Diabetes	2	8	768	{500, 268}
Wine	3	13	178	{59, 71, 48}
Credit-g	2	61	1000	{700, 300}
Credit-rating	2	42	690	{307, 383}
Phishing sites	2	30	2456	{1094, 1362}
Credit card	2	24	30000	{23364,6636}
Pendigits	10	16	10992	{1143,1143,1144,1055,1144, 1055,1056,1142,1055,1055}
Shuttle	7	9	43500	{34108,37,132,6748,2458,6,11}
Occupancy	2	6	20560	{15810, 4750}
HAPT	12	561	10929	{1722,1544,1407,1801,1979, 1958,70,33,107,85,139,84}
Loans	2	37	224858	{205229, 19629}
URLs	2	63	331622	{97925,233697}

TABLE III
OPTIMAL VALUE FOR γ IN KERNEL FUNCTION

Data set	SVM	Accu.	Nys.	Accu.	Data set	SVM	Accu.	Nys.	Accu.
Data set1	2^{7}	99.17	2^{0}	100	Credit-g	2^{-3}	76.10	2^{-9}	73.00
Data set2	2^{1}	98.60	2^{2}	96.97	Credit-rating	2^{-1}	86.52	2^{-2}	82.61
Data set3	2^{2}	100.00	2^{0}	99.75	Phishing sites	2^{-1}	97.60	2^{-9}	93.20
Iris	2^{1}	96.67	2^{-7}	98.66	Credit card	2^{-6}	78.57	2^{-12}	77.68
Seeds	2^{1}	91.90	2^{-9}	93.81	Pendigits	2^{-12}	99.58	2^{-12}	99.34
Segment	2^{3}	95.67	2^{0}	95.20	Shuttle	2^{6}	99.79	2^{-1}	99.84
Wholesale	2^{4}	90.91	2^{-3}	91.59	Occupancy	2^{-8}	98.93	2^{-9}	98.64
Glass	2^{3}	71.50	2^{-1}	72.43	HAPT	2^{-9}	94.28	2^{-8}	95.12
Diabetes	2^{0}	77.86	2^{-2}	72.66	Loans	2^{1}	91.27	2^{1}	91.27
Wine	2^{1}	99.44	2^{-2}	96.07	URLs	2^{-5}	92.11	2^{-5}	93.13

The search for Loans and URLs was performed on a uniform samples of 10000 points.

TABLE IV
NUMBER OF HYPER-SPHERES

Data set	Orig.	Nys.	Data set	Orig.	Nys.
Data set1	6	10	Credit-g	337	187
Data set2	39	31	Credit-rating	124	63
Data set3	60	19	Phishing sites	985	363
Iris	31	15	Credit card	1613	312
Seeds	87	33	Pendigits	405	125
Segment	323	174	Shuttle	221	103
Wholesale	125	74	Occupancy	1605	405
Glass	91	46	HAPT	969	355
Diabetes	249	163	Loans	2212	683
Wine	69	20	URLs	713	306

mentioned in Section II; "loans" and "URLs" were real datasets that we collected from other real projects; the other datasets were sourced from the University of California at Irvine, Machine Learning Repository [42] and LibSVM [43]. All attributes were numeric and the details of datasets are summarized in Table II.

B. Kernel Approximation With Nyström Method

As shown in (6), two types of elements need to be determined. The first is the sampling points and the second is the kernel function.

For datasets with less than 500 dimensions, we select all data points as the samples. For datasets with more than 500 fields, we use SOFM, which can be viewed as an independent parallel process of AdaHS, to obtain 500 cluster centers and use the centers as the representative points in (6). Previous research showed that sampling with clustering can enable Nyström method to have a much better approximation than uniform sampling [36].

We used radial basis function as the base kernel function for Nyström method

$$\kappa(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right). \tag{13}$$

The optimal values of parameter γ were obtained by grid search on $[2^{-12}, 2^{-11}, \ldots, 2^{12}]$. By comparison, we also performed grid search for kernelized SVM in the same way. The optimal values, which make the classifiers perform best on each dataset with regarding to accuracy, were recorded in Table III.

It can be observed that the optimal values for SVM and our model were not the same.

C. Training: Clustering With Classes Constraints

AdaHS uses labels of instances during the clustering phase. Training AdaHS consists of a clustering process, a monitoring process that watches boundary points, and a building process to construct DMZ. Thus, training is a process of adaptive clustering, with the constraints that all instances in a hypersphere must have the same label.

Both AdaHS and Nys-AdaHS record the number of hyper-spheres after training. Table IV showed the number of hyper-spheres for all datasets. It could be observed that the Nyström method generated much less number of hyper-spheres than the original model.

D. Convergence Tests for AdaHS

We use "distortion error" to monitor the training process and test for convergence. The distortion error is defined as follows [37], [38]:

$$\text{error} = \sum_{i=1}^{n} \|\phi(x_i) - c_i\|^2 \tag{14}$$

where c_i is the centroid of hyper-sphere to which x_i belongs. If AdaSH converges, it should find the globally optimal solution. In such a situation, both number of hyper-spheres and distortion error should be stable and converge to a particular value. Otherwise, the value of distortion error oscillates and does not converge. As long as the constraints on the same dataset are satisfied, the smaller the distortion error, the better the quality of clustering. We performed this test on the 20 benchmark datasets.

On datasets that could be easily clustered, such as dataset 1, the distortion error decreases to the bottom of the curve in only a few iterations and remains stable. On datasets that could not

TABLE V
DETAILS OF THE ACCURACY(%) IN SITUATIONS I AND II

Data set	I num	II num	I right num	II right num	I accu.	II accu.	Overall accu.
Data set 1	120	0	120	0	100	\	**100.00**
Data set 2	418	12	406	12	97.13	100	**97.21**
Data set 3	402	1	402	0	100.00	0.00	**99.75**
Iris	144	6	140	4	97.22	66.67	**96.00**
Seeds	188	20	183	8	97.34	40.00	90.95
Segment	1345	155	1303	120	96.88	77.42	94.87
Wholesale	432	8	397	2	91.90	25.00	90.68
Glass	201	13	147	6	73.13	46.15	**71.50**
Diabetes	560	208	455	113	81.25	54.33	73.96
Wine	174	4	169	1	97.13	25.00	95.51
Credit-g	854	146	643	94	75.29	64.38	73.70
Credit-rating	665	25	559	8	84.06	32.00	82.17
Phishing sites	2281	175	2224	74	97.50	42.29	93.57
Credit card	22170	7830	18749	5590	84.57	71.39	81.13
Pendigits	10087	905	10043	861	99.56	95.14	99.20
Shuttle	41861	1639	41855	1606	99.99	97.98	**99.91**
Occupancy	14005	6555	14005	6415	100	97.86	**99.32**
HAPT	9082	1847	9057	1308	99.72	70.82	94.84
Loans	183369	41489	169652	33012	92.52	79.57	90.13
URLs	287459	44163	273815	33134	95.25	75.03	**92.56**

be easily hyper-spherically clustered, such as dataset 2, distortion error values converge to the minimum after relatively more iterations of training. As it is shown in Fig. 4(b), the borders are relatively complex, so it took more hyper-spheres to enclose the entire set of instances and more time to converge. On some datasets, such as iris, segment, wholesale, credit-rating, and HAPT, the classifier converged after several small oscillations.

Results of the convergence tests on all 20 datasets showed that, given enough hyper-spheres and with the constraints that all instances in the same cluster have the same label, the competitive learning was able to provide a clustering solution no matter how complex and irregular the decision border is and what the data distribution is.

E. Performance Evaluation

Tenfold cross-validation was used to test the accuracy of AdaHS. To examine the details of the resulting predictions, performance on the two types of prediction was studied separately.

1) Two Types of Prediction: As discussed in Section III, our algorithm may be confronted with two situations in prediction, i.e., there is an explicit winning hyper-sphere or an instance does not fall into any existing hyper-sphere. Experiments on the 20 datasets showed that prediction accuracies of the two situations varied. Accuracies of the first situation were much higher than the second situation, as shown in Table V.

2) Accuracy and Time Cost Comparison: To evaluate the relative performance of AdaHS, we selected several other well-known algorithms, including naïve Bayes, LDA, SVM, C4.5, RBFN, and other incremental learning algorithms for comparison. Both accuracy and time cost were recorded. The comparative results are shown in Tables VI and VII.

Indices in Tables VI and VII show that C4.5 performed best on datasets phishing_sites and loans whose attributes were mostly nominal. LDA and L-SVM performed well on datasets with a globally consistent pattern, such as iris, seeds, wine, and wholesale, but perform poorly on datasets 1–3, segment, glass, and URLs. k-NN, k-SVM, and LWL were slow on large-scale datasets such as loans and URLs. Kernel methods improved the performance on most datasets, both in Nys-AdaSH and k-SVM.

AdaHS fit quite well for specific datasets, such as datasets 1–3, shuttle, occupancy, loans, and URLs, while maintained an acceptable accuracy on other datasets. As a local model, AdaHS works well on datasets which are linearly inseparable. In addition, because of the build-in clustering mechanism, its accuracy is comparable to k-NN and even SVM using kernel methods, but free from slow searching speed and excessive memory consumption.

We observed slightly lower performance of AdaHS and k-NN on diabetes, wine, credit-g, and credit-rating. This is due to the "bad distance metrics" noted in [31], which is crucial to distance-based models and implying that the features are not selected or scaled properly. Besides, one assumption of AdaHS is that instances in local areas can be clustered well. If this assumption is violated, such as in occupancy and loans, AdaHS will retrogress like k-NN. There will be too many clusters and DMZ points in the memory, and the searching speed will drop accordingly.

F. Discussion

1) Time Complexity and Space Complexity: It is obvious that the time costs of Algorithms 1–3 are $n \times m$, m^2, and m, where n is the number of data points and m is the number of clusters. The original form of the time complexity is $O(nm + m^2 + m)$. If the number of clusters m is constant, the total computational cost is $O(n)$. That means if the assumption of "clustering" holds, AdaHS runs in linear time. Data kept in memory are clusters and DMZ information, so the space cost is $O(m + l)$, where l is the data size in DMZ.

In Nys-AdaHS, the time cost of SOFM is $O(nk)$, where k is the cluster number of SOFM and the target dimension of Nyström method. The time cost of SVD on A_{11}^{+} in (4) is $O(rk^2)$, where r is the rank of A_{11}^{+} [40] and the multiplication with the vector in (6) also takes $O(nk)$. So the total time cost is $O(rk^2 + 2nk + nm + m^2 + m)$. With the cluster center of SOFM kept in memory, the space cost of Nys-AdaHS is $O(k + m + l)$.

It can be observed from Table VII that the time cost of Nys-AdaHS is far less than that of k-SVM, especially on the last 7 datasets. Because the computation of kernel matrix in k-SVM takes $O(n^2)$, which makes it hardly feasible in real applications. For example, on URLs, k-SVM took 6.085E5 s (about seven days), and that is not realistic in practice. Nys-AdaHs only took 894 s.

2) Significance of Nyström Methods for AdaHS: Our motivation to apply kernel methods to AdaSH is not exactly the same as SVM's. Based on the data in Tables IV, VI, and VII, the benefits of Nyström method for AdaHS are summarized as follows.

a) Improving classification accuracy: On datasets 1–3, segment, wholesale, glass, and loans, kernel methods improved the accuracy of SVM dramatically. That is because RBF kernel brings the local learning ability to SVM, and improves the

TABLE VI
ACCURACY (%) COMPARISON WITH OTHER ALGORITHMS

	AdaHS	Nys-AdaHS	k-NN	NB	LDA	L-SVM	K-SVM	C4.5	RBFN	LWL	RILB
Data set1	**100.00**	**100**	**100.00**	66.67	66.67	66.67	**99.17**	85.80	75.83	80.00	50.00
Data set2	**97.21**	**96.97**	96.74	76.04	55.12	55.12	**98.60**	95.01	74.88	79.53	53.49
Data set3	**99.75**	**99.75**	99.20	93.79	80.64	80.64	**100.00**	98.75	99.50	87.34	62.03
Iris	**96.00**	**98.66**	96.00	94.66	**98.00**	92.67	**96.67**	**96.00**	95.33	93.33	33.33
Seeds	90.95	**93.81**	89.05	91.42	**96.67**	**95.24**	91.90	91.90	92.38	91.90	33.33
Segment	94.87	**95.20**	95.07	81.07	91.93	89.93	**95.67**	**95.33**	86.87	84.20	86.40
Wholesale	90.68	**91.59**	**91.14**	89.09	84.77	88.86	**90.91**	90.00	88.63	90.66	67.72
Glass	**71.50**	**72.43**	71.02	49.53	62.62	59.81	**71.50**	66.82	63.55	44.86	35.51
Diabetes	73.96	72.66	72.14	75.75	**77.34**	**77.34**	**77.86**	73.83	75.39	71.22	65.10
Wine	95.51	96.07	95.51	96.63	**98.88**	**99.31**	**99.44**	93.82	98.31	88.76	39.89
Credit-g	73.70	73.00	72.80	74.70	**75.70**	**76.20**	**76.10**	69.50	71.00	70.00	70.00
Credit-rating	82.17	82.61	81.88	79.56	**85.80**	**85.94**	**86.52**	83.62	76.81	85.50	55.51
Phishing sites	93.57	94.20	93.65	**94.05**	**94.05**	94.83	**97.60**	**94.99**	92.63	88.68	93.36
Credit card	**81.13**	77.68	77.72	68.55	81.13	80.18	78.57	80.45	79.10	**81.87**	**81.61**
Pendigits	99.20	**99.34**	**99.35**	85.75	87.66	85.40	**99.58**	96.56	95.19	64.70	89.83
Shuttle	**99.91**	99.84	**99.89**	92.77	94.28	91.95	99.79	**99.91**	98.04	86.94	99.79
Occupancy	**99.32**	98.64	**98.93**	96.76	98.51	97.32	98.93	**99.17**	97.35	98.93	98.90
HAPT	94.84	**95.12**	95.00	74.12	**96.20**	**97.07**	94.28	92.30	84.21	48.93	87.85
Loans	90.13	**91.27**	89.88	40.86	79.62	63.18	**91.27**	**92.58**	91.27	\	**92.27**
URLs	**92.56**	**93.13**	**92.87**	51.87	88.08	86.08	92.11	90.34	86.14	\	85.89

L-SVM and K-SVM respectively refer to linear SVM and kernelized SVM, and the implementation packages are lib-linear and lib-svm; K-SVM used Radial Basis Function kernel, and the optimal gammas were obtained by grid search; NB, LWL, RILB respectively refer to Naive Bayes, Locally Weighted Learning and Raced Incremental Logit Boost; All nominal attributes are converted to binary numeric ones in advance.

TABLE VII
TIME COST (SECONDS) COMPARISON WITH OTHER ALGORITHMS

	AdaHS	Nys-AdaHS	k-NN	NB	LDA	L-SVM	K-SVM	C4.5	RBFN	LWL	RILB
Data set1	0.7	0.9	<0.1	<0.1	<0.1	0.1	1.1	0.1	0.1	0.1	<0.1
Data set2	1.3	1.4	0.3	<0.1	<0.1	0.8	3.5	0.2	0.2	0.2	0.1
Data set3	0.8	0.5	0.2	<0.1	<0.1	0.2	3.2	0.2	0.2	0.2	0.1
Iris	0.6	0.9	0.1	<0.1	<0.1	<0.1	1.2	0.1	0.3	0.1	<0.1
Seeds	0.6	0.4	0.1	<0.1	<0.1	0.1	1.2	0.1	1.4	0.3	<0.1
Segment	1.5	1.9	1.1	<0.1	0.5	0.4	14.3	0.3	39.1	21.3	0.9
Wholesale	1.5	1.2	0.3	<0.1	<0.1	0.1	3.3	0.2	0.1	0.9	<0.1
Glass	0.8	0.5	0.4	<0.1	0.1	0.2	2.6	0.1	4.1	0.7	<0.1
Diabetes	2.1	1.4	0.6	<0.1	<0.1	<0.1	5.9	0.2	0.3	2.1	<0.1
Wine	0.7	0.5	0.7	<0.1	<0.1	<0.1	3.7	0.3	0.1	0.3	<0.1
Credit-g	3.2	4.5	1.4	0.2	0.2	0.3	2.5	0.7	0.5	8.7	0.1
Credit-rating	1.3	2.4	2.7	0.2	0.1	0.2	6.0	1.7	0.6	3.1	0.1
Phishing sites	8.7	17.0	3.81	0.1	1.0	0.9	34.2	3.4	1.2	12.3	0.8
Credit card	184.8	165.9	241.3	2.3	2.8	18.33	1056	76.2	7.2	11063	7.8
Pendigits	9.04	24.7	73.2	1.2	1.3	32.19	756	8.7	2565.5	532.1	17.1
Shuttle	186.5	157.2	170.3	1.8	1.8	27.57	319	18.3	598.8	4753.5	41.4
Occupancy	63.5	42.5	68.8	2.11	1.3	21.32	3782	3.1	6.1	517.6	3.7
HAPT	90.1	53.6	129.6	13.6	431.2	91.23	667	287.9	552.1	29878	30.36
Loans	1092.3 / 859.5 / 669.8	746.9 / 543.6 / 267.4	1953.2	26.2	16.5	1453.4	4.732E5	1407.2	1772	\	86.1
URLs	1238.3 / 865.7 / 477.2	893.6 / 594.0 / 303.6	7272.3	42.9	223.9	1570.2	6.085E5	2151.8	1275.9	\	194.9

On Loans, URLs, we used both single and multiple machines to perform the experiments; the time cost in the corresponding cells was recorded respectively for single machine, 2 machines cluster, and 4 machines cluster. LWL cannot finish within 30 days.

accuracy of SVM on datasets that are not linearly separatable. AdaSH does not rely heavily on kernel methods like SVM in terms of accuracy. AdaSH is a local model. Nevertheless, if kernel method can make the dissimilar points apart and linearly separable in the new space [16], it can make the hyper-spheres more easily to enclose the points and classify. The effect of this benefit can be observed on most of the dataset in Table VI, for most accuracies were improved slightly.

b) Increasing hyper-spheres' ability for data definition: It can be observed from Table IV that the number of clusters was reduced significantly on all datasets with Nyström method. That means each hyper-sphere in the new space can enclose more data points. In SVDD, this phenomenon was stated as "kernel methods increase the hyper-sphere's ability for data definition" [18]. The reason is that AdaHS is a clustering-based method and now each cluster contains more information. It is especially useful when we analyze the evolving trends or the similar instances contained in the same cluster.

c) Improving the training speed: Nyström method does increase a time cost of $O(2nk+rk^2)$. By projecting data points to a new space of simpler distribution, the number of clusters can be reduced significantly, and the time cost saved from this benefit is $O((m_1 - m_2)(n + m_1 + m_2 + 1))$, where m_1 and m_2 refer to the number of clusters in AdaHS and Nys-AdaHS, respectively. So there is a tradeoff between the two terms. On large-scale datasets of complex data distribution (i.e., the original number of clusters is very large, like credit

card, occupancy, loans, and URLs), the total learning time of Nys-AdaHS could be reduced.

V. Conclusion

To deal with dynamic data and changing patterns, this paper proposed a new algorithm AdaHS, which incorporates the adaptivity of competitive neural networks and the idea of building a border zone. It has a strong capability for local learning. By keeping only cluster and DMZ information in memory, it avoids the problem of excessive memory consumption and improves the searching speed dramatically. The experiments using 20 datasets showed that AdaHS is especially suitable for datasets whose patterns are changing, decision borders are complex, and instances with the same label can be spherically clustered. AdaHS has great potentials in fields like anti-fraud analysis, network intrusion detection, stock market, and credit scoring.

AdaHS is proposed as a classifier to deal with changing patterns, which is a subtopic in system uncertainties [44]–[47]. System uncertainties theory has great significance to many important applications such as "actuator dynamics" [46], "multiagent-based systems" [47], and "nonlinear systems" [47]. One of our future works will take these problems into consideration and explore the potential applications to those areas.

References

[1] A. Bouchachia, "Adaptation in classification systems," in *Foundations of Computational Intelligence*, vol. 2. Heidelberg, Germany: Springer, 2009, pp. 237–258.

[2] G. Kou, Y. Peng, and G. Wang, "Evaluation of clustering algorithms for financial risk analysis using MCDM methods," *Inf. Sci.*, vol. 275, pp. 1–12, Aug. 2014.

[3] G. Kou, Y. Peng, Y. Shi, Z. Chen, and X. Chen, "A multiple-criteria quadratic programming approach to network intrusion detection," in *Data Mining and Knowledge Management*, vol. 3327. Heidelberg, Germany: Springer, 2005, pp. 145–153. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-540-30537-8_16#citeas

[4] Y. Huang and G. Kou, "A kernel entropy manifold learning approach for financial data analysis," *Decis. Support Syst.*, vol. 64, pp. 31–42, Aug. 2014.

[5] Z.-H. Zhou and Z.-Q. Chen, "Hybrid decision tree," *Knowl. Based Syst.*, vol. 15, no. 8, pp. 515–528, 2002.

[6] C. Alippi, D. Liu, D. Zhao, and L. Bu, "Detecting and reacting to changes in sensing units: The active classifier case," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 3, pp. 353–362, Mar. 2014.

[7] V. Bruni and D. Vitulano, "An improvement of kernel-based object tracking based on human perception," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 11, pp. 1474–1485, Nov. 2014.

[8] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1901–1914, Dec. 2011.

[9] M. Pratama, S. G. Anavatti, P. P. Angelov, and E. Lughofer, "PANFIS: A novel incremental learning machine," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 55–68, Jan. 2014.

[10] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 730–742, May 2010.

[11] Y. Guo, W. Zhou, C. Luo, C. Liu, and H. Xiong, "Instance-based credit risk assessment for investment decisions in P2P lending," *Eur. J. Oper. Res.*, vol. 249, no. 2, pp. 417–426, 2016.

[12] C.-H. Chen, "Feature selection for clustering using instance-based learning by exploring the nearest and farthest neighbors," *Inf. Sci.*, vol. 318, pp. 14–27, Oct. 2015.

[13] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," in *Proc. ACM SIGKDD*, Halifax, NS, Canada, 2017, pp. 445–454.

[14] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[15] P. Laskov, C. Gehl, S. Krüger, and K.-R. Müller, "Incremental support vector learning: Analysis, implementation and applications," *J. Mach. Learn. Res.*, vol. 7, pp. 1909–1936, Sep. 2006.

[16] S. Agarwal, V. V. Saradhi, and H. Karnick, "Kernel-based online machine learning and support vector reduction," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1230–1237, 2008.

[17] B. Li, M. Chi, J. Fan, and X. Xue, "Support cluster machine," in *Proc. ICML*, Corvallis, OR, USA, 2007, pp. 505–512.

[18] G. Chen, X. Zhang, Z. J. Wang, and F. Li, "Robust support vector data description for outlier detection with noise or uncertain data," *Knowl. Based Syst.*, vol. 90, pp. 129–137, Dec. 2015.

[19] G. Huang, H. Chen, Z. Zhou, F. Yin, and K. Guo, "Two-class support vector data description," *Pattern Recognit.*, vol. 44, no. 2, pp. 320–329, 2011.

[20] Z. Uykan, C. Guzelis, M. E. Celebi, and H. N. Koivo, "Analysis of input–output clustering for determining centers of RBFN," *IEEE Trans. Neural Netw.*, vol. 11, no. 4, pp. 851–858, Jul. 2000.

[21] B. Chandra and M. Gupta, "A novel approach for distance-based semi-supervised clustering using functional link neural network," *Soft Comput.*, vol. 17, no. 3, pp. 369–379, 2013.

[22] S. Grossberg, "Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world," *Neural Netw.*, vol. 37, pp. 1–47, Jan. 2013.

[23] G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," *Neural Netw.*, vol. 4, no. 5, pp. 565–588, 1991.

[24] J. R. Williamson, "Gaussian ARTMAP: A neural network for fast incremental learning of noisy multidimensional maps," *Neural Netw.*, vol. 9, no. 5, pp. 881–897, 1996.

[25] B. Vigdor and B. Lerner, "The Bayesian ARTMAP," *IEEE Trans. Neural Netw.*, vol. 18, no. 6, pp. 1628–1644, Nov. 2007.

[26] R. Hecht-Nielsen, "Counter propagation networks," *Appl. Opt.*, vol. 26, no. 23, pp. 4979–4983, 1987.

[27] Y. Dong, M. Shao, and X. Tai, "An adaptive counter propagation network based on soft competition," *Pattern Recognit. Lett.*, vol. 29, no. 7, pp. 938–949, 2008.

[28] P. Schneider, M. Biehl, and B. Hammer, "Adaptive relevance matrices in learning vector quantization," *Neural Comput.*, vol. 21, no. 12, pp. 3532–3561, 2009.

[29] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.

[30] C. Xiao and W. A. Chaovalitwongse, "Optimization models for feature selection of decomposed nearest neighbor," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 2, pp. 177–184, Feb. 2016.

[31] R. A. Valente and T. Abrão, "MIMO transmit scheme based on morphological perceptron with competitive learning," *Neural Netw.*, vol. 80, pp. 9–18, Apr. 2016.

[32] Q. Dai and G. Song, "A novel supervised competitive learning algorithm," *Neurocomputing*, vol. 191, pp. 356–362, May 2016.

[33] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-Newton method for online convex optimization," *J. Mach. Learn. Res.*, pp. 436–443, 2007.

[34] W. Bian and D. Tao, "Constrained empirical risk minimization framework for distance metric learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1194–1205, Aug. 2012.

[35] K. Chen and S. H. Wang, "Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 129–143, Jan. 2011.

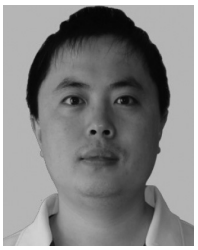
[36] M. Zia-ur Rehman, T. Li, Y. Yang, and H. Wang, "Hyper-ellipsoidal clustering technique for evolving data stream," *Knowl. Based Syst.*, vol. 70, pp. 3–14, Nov. 2014.

[37] J. Lai and C. Wang, "Kernel and graph: Two approaches for nonlinear competitive learning clustering," *Front. Elect. Electron. Eng.*, vol. 7, no. 1, pp. 134–146, 2012.

[38] J.-S. Wu, W.-S. Zheng, and J.-H. Lai, "Approximate kernel competitive learning," *Neural Netw.*, vol. 63, pp. 117–132, Mar. 2015.

[39] S. Kumar, M. Mohri, and A. Talwalkar, "Sampling methods for the Nyström method," *J. Mach. Learn. Res.*, vol. 13, pp. 981–1006, Apr. 2012.

[40] J. Lu, S. C. H. Hoi, J. Wang, P. Zhao, and Z.-Y. Liu, "Large scale online kernel learning," *J. Mach. Learn. Res.*, vol. 17, no. 47, pp. 1–43, 2016.
[41] Hazelcast. *Hazelcast: The Leading In–Memory Data Grid*. Accessed: Apr. 5, 2016. [Online]. Available: http://hazelcast.com
[42] *UC Irvine Machine Learning Repository*. Accessed: Dec. 13, 2015. [Online]. Available: http://archive.ics.uci.edu/ml/index.php
[43] LibSVM. *LIBSVM Data: Classification, Regression, and Multi-Label*. Accessed: Dec. 16, 2015. [Online]. Available: https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/
[44] C. Chen *et al.*, "Adaptive fuzzy asymptotic control of MIMO systems with unknown input coefficients via a robust Nussbaum gain-based approach," *IEEE Trans. Fuzzy Syst.*, vol. 25, no. 5, pp. 1252–1263, Oct. 2017.
[45] Z. Liu, G. Lai, Y. Zhang, X. Chen, and C. L. P. Chen, "Adaptive neural control for a class of nonlinear time-varying delay systems with unknown hysteresis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 12, pp. 2129–2140, Dec. 2014.
[46] C. Chen, Z. Liu, Y. Zhang, C. L. P. Chen, and S. L. Xie, "Saturated Nussbaum function based approach for robotic systems with unknown actuator dynamics," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2311–2322, Oct. 2016.
[47] C. Chen *et al.*, "Adaptive consensus of nonlinear multi-agent systems with non-identical partially unknown control directions and bounded modelling errors," *IEEE Trans. Autom. Control*, vol. 62, no. 9, pp. 4654–4659, Sep. 2017.



Tie Li received the M.S. degree in management science and engineering from Shanxi University, Taiyuan, China, in 2009. He is currently pursuing the doctoral degree with the School of Management and Economics, University of Electronic Science and Technology of China, Chengdu, China. He has authored four software and ten papers. His current research interests include big data mining and information management.

Gang Kou received the B.S. degree in physics from Tsinghua University, Beijing, China, and the M.S. degree in computer science and the Ph.D. degree in information technology from the University of Nebraska at Omaha, Omaha, NE, USA.

He is a Distinguished Professor of Chang Jiang Scholars Program and the Executive Dean of the School of Business Administration, Southwestern University of Finance and Economics, Chengdu, China.

Dr. Kou is the Managing Editor of the *International Journal of Information Technology and Decision Making*, and the Editor-in-Chief of Springer book series on Quantitative Management.

Yi Peng received the B.S. degree in management information systems from Sichuan University, Chengdu, China, in 1997, and the M.S. degree in management information systems and the Ph.D. degree in information technology from the University of Nebraska at Omaha, Omaha, NE, USA, in 2007.

From 2007 to 2011, she was an Assistant Professor with the School of Management and Economics, University of Electronic Science and Technology of China, Chengdu, China, where she has been a Professor with the School of Management and Economics, University of Electronic Science and Technology of China, since 2011. She has authored three books and over 100 articles. Her current research interests include data mining, multiple criteria decision making, and data mining applications.

Yong Shi received the Ph.D. degree in management science and computer systems from the University of Kansas, Lawrence, KS, USA, in 1991.

He is the Director of the Key Research Laboratory on Big Data Mining and Knowledge Management and the Research Center on Fictitious Economy and Data Science, Chinese Academy of Sciences, Beijing, China. Since 1999, he has been the Charles W. and the Margre H. Durham Distinguished Professor of information technology with the College of Information Science and Technology, Peter Kiewit Institute, Omaha, NE, USA, and the University of Nebraska, Lincoln, NE, USA. He has authored 17 books and over 200 papers. His current research interests include business intelligence, data mining, and multiple criteria decision making.