Expert Systems With Applications 65 (2016) 332–344

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Discussion

A fast incremental extreme learning machine algorithm for data streams classification

Shuliang Xu [a], Junhong Wang [a,b,*]

[a] School of Computer and Information Technology, Shanxi University, Taiyuan 030006, China
[b] Key Laboratory of Computational Intelligence and Chinese Information Processing of Ministry of Education, Taiyuan 030006, China

ARTICLE INFO

Article history:
Received 31 March 2016
Revised 17 August 2016
Accepted 18 August 2016
Available online 25 August 2016

Keywords:
Data mining
Extreme learning machine
Data streams
Classification
Concept drift

ABSTRACT

Data streams classification is an important approach to get useful knowledge from massive and dynamic data. Because of concept drift, traditional data mining techniques cannot be directly applied in data streams environment. Extreme learning machine (ELM) is a single hidden layer feedforward neural network (SLFN), comparing with the traditional neural network (e.g. BP network), ELM has a faster speed, so it is very suitable for real-time data processing. In order to deal with the challenge in data streams classification, a new approach based on extreme learning machine is proposed in this paper. The approach utilizes ELMs as base classifiers and adaptively decides the number of the neurons in hidden layer, in addition, activation functions are also randomly selected from a series of functions to improve the performance of the approach. Finally, the algorithm trains a series of classifiers and the decision results for unlabeled data are made by weighted voting strategy. When the concept in data streams keeps stable, every classifier is incrementally updated by using new data; if concept drift is detected, the classifiers with weak performance will be cleared away. In the experiment, we used 7 artificial data sets and 9 real data sets from UCI repository to evaluate the performance of the proposed approach. The testing results showed, comparing with the conventional classification methods for data streams such as ELM, BP, AUE2 and Learn++.MF, on most data sets, the new approach could not only be simplest in the structure, but also get a higher and more stable accuracy with lower time consuming.

1. Introduction

With the development of information society, many fields have produced huge amount of data streams, such as communication, electronic commerce, stock market et al. Data streams classification is different from conventional classification approaches (Padmalatha, Reddy, & Rani, 2014); due to infinite number, fast arrival and concept drift, so how to derive useful knowledge and patterns from massive stream data becomes a challenge in data mining and machine learning (Lemaire, Salperwyck, Bondu, Zimányi, & Kutsche, 2015). Since the model of data stream was proposed, it has attracted much attention from Scholars (Farid et al., 2013; Gama, Sebasti, & Rodrigues, 2009). Domingos et al. proposed VFDT algorithm (Hulten, Spencer, & Domingos, 2001), VFDT establishes a decision tree based on Hoeffding inequality in the training model process and makes a leaf node become a decision node according to the statistics of the attributes values. After a new sample arriving, by traversing the decision tree with top-down method, we will get the labels of the unlabeled data. Lenco et al. proposed CDCStream algorithm (Ienco, Bifet, Pfahringer, & Poncelet, 2014). CDCStream treats a data block as a processing unit and it compares the statistical information of the current data and historical data to determine whether the distribution of the data changing or not. If an alarm is sent, the algorithm starts to collect data. If concept drift is detected, the collect data will be used to train a new classifier to learn new concept. Du et al. proposed an algorithm based on information entropy and adaptive sliding window, called ADDM (Du, Song, & Jia, 2014), it detects concept drift only when the number of instances in sliding window satisfies Hoeffding bound. In ADDM algorithms, an increased error rate will lead to an increase in entropy. If the calculated entropy is equal to 1, the system will think a change has happened and MB-GT (Nikovski & Jain, 2010) algorithm will be used to retrain classifiers. For partially labeled data, Wu et al. proposed CDRT algorithm (Wu, Li, & Hu, 2012), the algorithm is based on random decision tree model and uses semi supervised learning techniques to deal with unlabeled data. CDRT is similar to CVFDT (Hulten et al., 2001), when concept drift appearing, the subtree replacement strategy will be taken to update classifiers. Aiming at the problem of periodic concept drift, Ortiz

* Corresponding author. Tel.: +8613834560616.
E-mail addresses: xushulianghao@126.com (S. Xu), wjhwjh@sxu.edu.cn (J. Wang).

http://dx.doi.org/10.1016/j.eswa.2016.08.052
0957-4174/

proposed an ensemble algorithm called FAE (Ortíez et al., 2014), FAE utilizes Hoeffding trees (Hulten et al., 2001) as base classifiers, when the weight of old classifier is greater than a threshold, the classifier will be activated and then go to affect the decision of the classification results. Shao et al. proposed an algorithm (Shao, Ahmadi, & Kramer, 2014) based on P-tree structure and it uses PCA to detect concept drift. The difference between the two adjacent blocks is measured by the first principal components of the adjacent blocks; if the cosine angle of the two first principal component vectors exceeds a range, a change in data streams will be thought to appear. Farid et al. proposed an algorithm for detecting novel classes in data streams by a decision tree (Farid & Rahman, 2012). The approach unites classification and clustering techniques and treats concept drift as a novel class. By calculating the percentage of the data of each class in leaf nodes, a novel class will be detected. Shaker et al. proposed an instances-based algorithm, called IBLStream (Shaker & Húllermeier, 2012); IBLStream is similar to KNN, by saving the relevant examples, the labels of new data are decided according to the several examples of the nearest neighbors. Li et al. extended the classical SVM to an on-line classifier (Li & Yu, 2015), which was called OLSVM, the algorithm can be directly applied for data streams classification with kernel function method. Xing et al. proposed a lifelong learning method for Support Vector Machine (SVM) training called L^3-SVM (Youlu, Shen, Chaomin, & Zhao, 2015). The algorithm introduces a Prototype Support Layer (PSL) before SVM training. L^3-SVM learns the concepts of data streams in an online way.

ELM, proposed by Huang, Zhu, and Siew (2006b), is an effective machine learning algorithm (Huang, Huang, Song, & You, 2015a; Huang, 2014; Huang, Zhou, Ding, & Zhang, 2012; Yu, Dai, & Tang, 2015b) for SLFN and can get a much faster speed with a good generalization performance than SVM and BP network (Liu, Gao, & Li, 2012). ELM randomly generates the input weights and the hidden layer biases, the output weights can be determined analytically. Since ELM was proposed, it has been widely applied and developed. Tavares et al. proposed an extreme learning algorithm with parallel layer perceptrons, called PLP-ELM (Tavares, Saldanha, & Vieira, 2015). Different from original ELM, PLP-ELM has two parallel hidden layers. For the activation functions in the hidden layers, one is linear and another is nonlinear. The final classification result for each instance is determined by two parallel hidden layers outputs. Zhang et al. proposed a self-adaptive algorithm called RAE-ELM (Zhang & Yang, 2015). RAE-ELM introduces AdaBoost.RT algorithm to train several classifiers, only the classifiers that the error rate is within a certain range will be accepted. Li et al. proposed an extreme learning machine algorithm based on transfer learning, called TL-ELM (Li, Mao, & Jiang, 2014). In TL-ELM, the difference between the prior knowledge and the target learning model needs to be considered. TL-ELM can use the existing knowledge to learn similar knowledge without retraining classifiers. Xin et al. proposed extreme learning machine for big data classification, called Elastic-ELM (Xin, Wang, Qu, & Wang, 2015); Elastic-ELM uses MapReduce to parallel handle data and it is very suitable for the problem of the real-time data processing. Stasic et al. proposed an extreme learning machine called V-QELM which based on voting mechanism (Stosic, Stosic, & Ludermir, 2016). In V-QELM algorithm, for the activation functions in hidden layer nodes, q-Gaussian activation function is adopted. Each classifier selects an optimal value of the Parameter q_k from a number of candidate values. Final classification results are determined by k classifiers with the voting mechanism. In the article (Huang, Bai, Kasun, & Chi, 2015b; Zeng, Xu, & Fang, 2015), convolutional neural network (CNN) is used in extreme learning machine. The algorithm utilizes CNN to extract effective features from a feature set and then ELM is used to decide the class labels of unlabeled data.

For the most algorithms of ELM, the number of the nodes in hidden layer needs to be specified in advance; sometimes it is difficult for users to choose an appropriate number of the nodes without priori knowledge. To solve this problem, Xue et al. proposed an ensemble extreme learning machine algorithm called GE-ELM (Xue, Yao, Wu, & Yang, 2014) based on Genetic algorithm. In GE-ELM, several ELM classifiers with different numbers of hidden layer nodes are created at first, and then the algorithm uses genetic operations and a validation set getting from training set to select the top M classifiers with minimum error rate and the least norm of the output weight matrix. Miche et al. prosed an optimally pruned Extreme Learning Machine called OP-ELM (Miche et al., 2010); OP-ELM ranks the best neurons using MRSR and selects the optimal number of neurons by LOO algorithm. Huang et al. proposed SRM-ELM algorithm based on structural risk minimization (Huang & Lai, 2012). SRM-ELM can obtain an optimal number of neurons by Particle Swarm Optimization (PSO) with minimal structural risk. Lima et al. proposed an extreme learning machine algorithm for nonlinear regression called H-ELM (Lima, Cannon, & Hsieh, 2015). In H-ELM, after the parameters are initialized, the algorithm will automatically select the best number of neurons by training set and feedback mechanism. But in fact, when classifiers are trained by the above algorithms, classifier will not change any more, so it is evident that the algorithms cannot handle data streams hidden concept drift.

Because of simple structure, short training time and good generalization ability, extreme learning machine satisfies the requirements of classification algorithms in data streams. Aiming at the problem of data stream classification, the existing ELM algorithms have already made some achievements; most of them ,such as OS-ELM etc (Liang, Huang, Saratchandran, & Sundararajan, 2006), uses incremental learning strategy to update classifiers. For the algorithms, gradual concept drift can be dealt with well, however, it can not handle abrupt concept drift. So in this paper, a fast incremental extreme learning machine algorithm for data streams (IDS-ELM) is proposed. The approach takes ELM as base classifiers and uses an incremental learning mechanism to improve the performance of classifiers. When the concept hidden in data streams changing and evolving, the classifiers which do not meet the requirements of the data streams classification will be eliminated and retrained.

For the paper, the contributions are as follows:

(1) A new data streams classification algorithm is developed via a weighted extreme learning machine strategy (Zong, Huang, & Chen, 2013). The algorithm will be incrementally and continuously updating classifier in the stable data, when concept drift appears, the algorithm will rebuild a classifiers system and can get a better accuracy.
(2) The existed ELM algorithms (Li et al., 2014; Shao & Japkowicz, 2012; Shao et al., 2014; Wang, Lu, Dong, & Zhao, 2015; Xue et al., 2014; Zhang, Lan, Huang, & Xu, 2013) are very sensitive to the number of the neurons in hidden layer. Although most of them have used some search techniques to select the optimal number of the neurons, the methods produces much time overhead and it is not suitable for data stream environment. IDS-ELM utilizes an approach similar to the binary search method to find an appropriate number of the neurons in hidden layer. Comparing with the existed algorithms, the speed of IDS-ELM is faster with a high accuracy.
(3) According to the literatures (Zhang, Zhu, Shi, & Wu, 2009), the more differences the classifiers are, the better performances the algorithm will have. In order to increase the differences among the classifiers, in this paper, activation functions in hidden nodes are randomly selected from a series of functions.

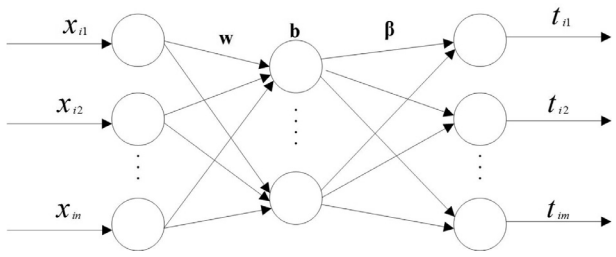

Fig. 1. ELM structure.

The rest of the paper is organized as follows. Section 2 briefly reviews classification techniques for data streams and ELM basics. In Section 3, a new fast incremental extreme learning machine algorithm called IDS-ELM is proposed and in Section 4, IDS-ELM is evaluated with 7 artificial data sets and 9 real data sets. Finally, the conclusions are made in Section 5.

2. Related work

2.1. Data streams classification

Let $\mathbf{S}=\{\ldots,\mathbf{d}_{t-1},\mathbf{d}_t,\mathbf{d}_{t+1},\ldots\}$ denotes a data stream, where $\mathbf{d}_t=\{\mathbf{x}_i,y_i\}$, $\mathbf{x}_i$ is the value set of the ith datum in each attribute and y_i is the class label of the instance. The goal of the data stream classification is to train a classifier f: $\mathbf{x}\rightarrow y$ that establishes a mapping relationship between feature vectors and class labels (Li, Wu, Hu, & Wang, 2015). Data stream is different from conventional static data, which often has infinite number, continuously arrives and is accessed only once. To facilitate the description of the data stream problem, we have the following notions.

A number of instances are organized into a data set with generating order and we call the data set as a data block, so it is know that all data streams are composed of many data blocks. Because of massive data and high speed for data streams, the data is only allowed to be accessed once, and the sliding window mechanism is applied (Soares & Rui, 2015a). In sliding window, there is one or several data blocks in a window at the same time, only if the data in the current window is processed completely, the next data block can be allowed to enter into the sliding window.

At time t, the data distribution in data streams is $p_t(y|\mathbf{x})$, after Δt time, the data distribution is changing to $p_{t+\Delta t}(y|\mathbf{x})$; if $p_{t+\Delta t}(y|\mathbf{x})\neq p_t(y|\mathbf{x})$, it is said that concept drift has happened in Δt time (liobait, Technol, Bifet, Pfahringer, & Holmes, 2014). If Δt is a short time, the conceptual transformation is referred to as gradual concept drift; if Δt is a long time, it is called abrupt concept drift.

When a concept in data stream has changed, classifiers will not function well, so the accuracy of the classification will decline. By observing the change of the neighboring accuracies, if the change of the accuracies is more than a range, we can say the concept has changed in the data streams.

2.2. ELM basics

ELM is a single hidden layer feedforward neural network (Huang et al., 2006b). The input weights and biases for hidden node are assigned randomly and the weights connecting the hidden layer and output layer are determined analytically (Huang et al., 2012). The structure of ELM is showed in Fig. 1.

For N arbitrary distinct samples $(\mathbf{x}_i,\ \mathbf{t}_i)$, where $\mathbf{x}_i=[x_{i1},x_{i2},x_{i3},\ldots,x_{in}]^T$, the output of a standard ELM with L hidden nodes and an activation function $g(x)$ which is nonlinear infinitely differentiable in any interval is mathematically modeled as

$$\sum_{i=1}^{L}\boldsymbol{\beta}_i g(a_i,b_i,\mathbf{x}_j)=\mathbf{o}_j,\quad j=1,2,\ldots,N \tag{1}$$

Where a_i is the weight connecting the input neurons with the ith hidden neuron; $\boldsymbol{\beta}_i$ is the weight connecting the ith hidden neuron with the output neurons; b_i is the bias of the ith hidden neuron. The ELM performance can be guaranteed by the two theorems in Huang, Chen, and Siew (2006a); Huang et al. (2006b) and ELM can approximate the N samples with zero error if the conditions of the two theorems are satisfied (Huang et al., 2006a; Huang et al., 2006b), so we have

$$\sum_{i=1}^{L}\boldsymbol{\beta}_i g(a_i,b_i,\mathbf{x}_j)=\mathbf{t}_j,\quad j=1,2,\ldots,N \tag{2}$$

The formula (2) can be written in a more compact format as

$$\mathbf{H}\boldsymbol{\beta}=\mathbf{T} \tag{3}$$

where

$$H=\begin{bmatrix} g(a_1,b_1,\mathbf{x}_1) & \ldots & g(a_L,b_L,\mathbf{x}_1)\\ \vdots & \ddots & \vdots\\ g(a_1,b_1,\mathbf{x}_N) & \ldots & g(a_L,b_L,\mathbf{x}_N)\end{bmatrix} \tag{4}$$

$$\boldsymbol{\beta}=\begin{bmatrix}\boldsymbol{\beta}_1^T\\ \vdots\\ \boldsymbol{\beta}_L^T\end{bmatrix}_{L\times m}\quad\text{and}\quad T=\begin{bmatrix}\mathbf{t}_1^T\\ \vdots\\ \mathbf{t}_N^T\end{bmatrix}_{N\times m} \tag{5}$$

$\mathbf{H}$ is the output matrix of the hidden layer; the ith column of $\mathbf{H}$ is the ith hidden neuron output vector for samples and the ith row is the ith sample output vector for all hidden neurons. For activation function, six nonlinear functions are most commonly used (Fernandez-Navarro, Hervas-Martínez, Sanchez-Monedero, & Gutierrez, 2011; Zuo, Huang, Wang, Han, & Westover, 2014):

(1) Sigmoid function

$$G(\mathbf{a},b,\mathbf{x})=\frac{1}{1+exp(-(\mathbf{a}\cdot\mathbf{x}+b))} \tag{6}$$

(2) Sinusoid function

$$G(\mathbf{a},b,\mathbf{x})=\sin(\mathbf{a}\cdot\mathbf{x}+b) \tag{7}$$

(3) Multiquadric function

$$G(\mathbf{a},b,\mathbf{x})=\sqrt{\|\mathbf{x}-\mathbf{a}\|^2+b^2} \tag{8}$$

(4) Gaussian function

$$G(\mathbf{a},b,\mathbf{x})=exp(-\frac{\|\mathbf{x}-\mathbf{a}\|^2}{b}) \tag{9}$$

(5) Hardlim function

$$G(\mathbf{a},b,\mathbf{x})=\begin{cases}1 & if\ \ \mathbf{a}\cdot\mathbf{x}+b\geqslant 0\\ 0 & \text{otherwise}\end{cases} \tag{10}$$

(6) Triangular basis function

$$G(\mathbf{a},b,\mathbf{x})=\begin{cases}1-|\mathbf{a}\cdot\mathbf{x}+b| & if\ \ -1\leqslant\mathbf{a}\cdot\mathbf{x}+b\leqslant 1\\ 0 & \text{otherwise}\end{cases} \tag{11}$$

According to the relevant definition of the *Moore-penrose* generalized inverse (Yong & Meng, 2016), the smallest norm least-squares of (3) is as

$$\hat{\boldsymbol{\beta}}=\mathbf{H}^{\dagger}\mathbf{T} \tag{12}$$

Where $\mathbf{H}\dagger$ is the *Moore-penrose* generalized inverse of the matrix $\mathbf{H}$. $\mathbf{H}\dagger$ can be calculated through orthogonal projection method,

orthogonalization method and singular value composition (SVD) (Zhang, 2013).

In order to improve the generalization performance, when dealing with classification problems, ELM is generally regularized as the following optimization problem (Yu, Sun, Yang, Yang, & Zuo, 2015a):

$$\begin{aligned} &min \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{1}{2}C\sum_{i=1}^{N}\|\boldsymbol{\xi}_i\|^2 \\ &s.t. \quad h(\mathbf{x}_i)\boldsymbol{\beta} = \mathbf{t}_i^T - \boldsymbol{\xi}_i^T \end{aligned} \tag{13}$$

Where $\boldsymbol{\xi_i}$ is the training error of the ith sample; C is a relevant penalty factor. We can transform the problem (13) into the solution of the dual problem and construct the following Lagrangian function:

$$L = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{1}{2}C\sum_{i=1}^{N}\|\boldsymbol{\xi}_i\|^2 - \sum_{i=1}^{N}\sum_{j=1}^{L}\alpha_{ij}(h(\mathbf{x}_i)\boldsymbol{\beta}_j - t_{ij} + \xi_{ij}) \tag{14}$$

From the partial derivatives of the function and KKT conditions (Huang, Wang, & Lan, 2011; Huang et al., 2012; Sun, Yuan, & Wang, 2014), we can conclude, if $L < N$, the size of the matrix $\boldsymbol{H}^T\boldsymbol{H}$ is less than that of the matrix$\boldsymbol{H}\boldsymbol{H}^T$, combining (1)–(3), the solution of the equations is

$$\boldsymbol{\beta} = \boldsymbol{H}^{\dagger}\boldsymbol{T} = \left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}^T\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{T} \tag{15}$$

So the final output of ELM is

$$f(x) = h(\boldsymbol{x})\boldsymbol{\beta} = h(\boldsymbol{x})\left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}^T\boldsymbol{H}\right)^{-1}\boldsymbol{H}^T\boldsymbol{T} \tag{16}$$

If $L > N$, the size of the matrix $\boldsymbol{H}\boldsymbol{H}^T$ is less than that of the matrix $\boldsymbol{H}^T\boldsymbol{H}$, the solution of the equations is

$$\boldsymbol{\beta} = \boldsymbol{H}^T\left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}\boldsymbol{H}^T\right)^{-1}\boldsymbol{T} \tag{17}$$

So the final output of ELM is

$$f(x) = h(\boldsymbol{x})\boldsymbol{\beta} = h(\boldsymbol{x})\boldsymbol{H}^T\left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}\boldsymbol{H}^T\right)^{-1}\boldsymbol{T} \tag{18}$$

In the binary classification problem (Singh, Kumar, & Singla, 2015; Sun et al., 2014), the decision function of ELM is formulated as

$$f(x) = \text{sign}(h(\boldsymbol{x})\boldsymbol{\beta}) \tag{19}$$

For multiclass case (Singh et al., 2015; Sun et al., 2014), the class label of sample is formulated as

$$label(x) = \arg\max_{1\leqslant i\leqslant m}\{f_i(x)\} \tag{20}$$

and

$$f(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})]^T \tag{21}$$

The ELM training algorithm can be summarized as follows Huang et al. (2006b).

Algorithm 1. A training set $\mathscr{X} = \{(\boldsymbol{x}_i, \boldsymbol{t}_i)|\boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{t}_i \in \mathbb{R}^m\}$, activation function $g(x)$, the number of hidden neurons L.

Step 1. Randomly generate input node weight w_i and hidden neurons bias b_i, $i = 1, 2, \ldots, L$.
Step 2. Calculate the hidden layer output matrix **H** for the training samples.
Step 3. Calculate the output layer weight $\boldsymbol{\beta}$ based on the formula (15) or (17).

From the above, it can be known that, for ELM, the output layer weight vector is the only parameter that needs to be calculated and not to be tuned repeatedly. Comparing with the traditional neural network algorithm e.g. BP neural network, ELM has the advantage in the training speed, so it meets the requirements of the algorithm in the data stream environment.

3. IDS-ELM algorithm

3.1. The approach determining the number of the neurons in hidden layer

For ELM, the number of neurons has a great influence on the performance of the algorithm (Mirza, Lin, & Liu, 2015). If the number is too few, ELM can complete the classification task in a short time, but the classifier is unable to cope with data well due to the under fitting, so the error rate is very high. If the number of neurons is too many, the structure of ELM will be very complicated and the over-fitting will happen. To obtain the optimal number of neurons, one method for conventional ELM algorithms (Liang et al., 2006; Silva, Pacifico, & Ludermir, 2011) is to train a series of ELMs with different numbers of neurons, and then selects a ELM with minimum error rate as the last result; another way is to add neuron into hidden layer one by one until the accuracy is no longer increased or the increasing level is very small.

It is obvious that the above methods are time-consuming, so they cannot be applied in data streams environment. For the sake of getting the suitable number of neurons in a short time, in this paper, an algorithm similar to the binary search is proposed. For ELM, the number of neurons L should not be too large; in the algorithm, we limit L in (0, num], where num= min(N, q); q is the dimensions of the training data and N is the number of samples. We compare the current accuracy of ELM with the last one, if the difference between the two accuracies is less than ε(ε is determined by *Hoeffding bound* and the details can be seen in Section 3.2), the number of the neurons in hidden layer of the current ELM is the final result. The algorithm is showed as follows:

Algorithm 2. A training set, a validation set, the upper bound of L is num, L is initialized to $\frac{1}{2}num$, m=1;

Step 1. Utilize Algorithm 1 to produce ELM with L hidden neurons and calculate the accuracy v_1 of the validation set.
Step 2. $L_0 = \frac{1}{2}(L+m)$, train an ELM with L_0 hidden neurons and calculate the accuracy v_2 of the validation set.
Step 3. If $v_2 - v_1 > \varepsilon$, $L = L_0$ and $v_1 = v_2$, go to *step 2*; else go to *step 4*;
Step 4. $L_2 = \frac{1}{2}(L + num)$, train an ELM with L_2 hidden neurons and calculate the accuracy v_3 of the validation set.
Step 5. If $v_3 - v_1 > \varepsilon$, m=L, L=L_2 and $v_1 = v_3$ goto *step 2*; else goto *step 6*.
Step 6. Terminate Algorithm 2 and output L.

From Algorithm 2, a suitable number L for ELM hidden neurons can be found, then we use L to train a series of ELMs.

3.2. Incrementally updating for IDS-ELM

In data streams, when the change of the data is not apparent, e.g. gradual concept drift, conventional data streams classification algorithms deem no change has happened, so classifiers will keep immobile (Webb, Hyde, Cao, Nguyen, & Petitjean), however, it is not consistent with the fact, so the performance of classifiers will be dropping with time going on. In order to handle a slight change in data streams, we use online sequential learning algorithm (OS-ELM) (Liang et al., 2006) to update every classifier. From Section 3.2, we have known that the number of the neurons L in ELM is less than N, so the output weight is calculated as the formula (15).

Given an initial data block $D = \{\boldsymbol{x}_i, t_i\}_{i=1}^{N_0}$, the hidden layer output matrix is

$$H_0 = \begin{bmatrix} G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_1) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_{N_0}) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_{N_0}) \end{bmatrix} \tag{22}$$

And

$$T_0 = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_{N_0}^T \end{bmatrix} \tag{23}$$

Form the formula (15), we have

$$\boldsymbol{\beta}^{(0)} = \left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}_0^T\boldsymbol{H}_0\right)^{-1}\boldsymbol{H}_0\boldsymbol{T}_0 \tag{24}$$

Let $\boldsymbol{K}_0 = \frac{\boldsymbol{I}}{C} + \boldsymbol{H}_0^T\boldsymbol{H}_0$, the formula (24) can be transformed into the formula (25)

$$\boldsymbol{\beta}^{(0)} = \boldsymbol{K}_0^{-1}\boldsymbol{H}_0^T T_0 \tag{25}$$

We suppose a new data block $D_1 = \{(\boldsymbol{x}_i, \boldsymbol{t}_i)\}_{i=N_0+1}^{N_0+N_1}$ coming after getting $\boldsymbol{\beta_0}$, then we use the new data block to update classifier, so $\boldsymbol{\beta^{(1)}}$ is

$$\begin{aligned}\boldsymbol{\beta}^{(1)} &= \left(\frac{\boldsymbol{I}}{C} + \begin{bmatrix}\boldsymbol{H}_0 \\ \boldsymbol{H}_1\end{bmatrix}^T\begin{bmatrix}\boldsymbol{H}_0 \\ \boldsymbol{H}_1\end{bmatrix}\right)^{-1}\begin{bmatrix}\boldsymbol{H}_0 \\ \boldsymbol{H}_1\end{bmatrix}^T\begin{bmatrix}\boldsymbol{T}_0 \\ \boldsymbol{T}_1\end{bmatrix} \\ &= \boldsymbol{K}_1^{-1}(\boldsymbol{H}_0^T\boldsymbol{T}_0 + \boldsymbol{H}_1^T\boldsymbol{T}_1)\end{aligned} \tag{26}$$

Where

$$\mathbf{H}_1 = \begin{bmatrix} G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_{N_0+1}) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_{N_0+1}) \\ \vdots & \ddots & \vdots \\ G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_{N_0+N_1}) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_{N_0+N_1}) \end{bmatrix} \tag{27}$$

$$\boldsymbol{T}_1 = [\boldsymbol{t}_{N_0+1}, \dots, \boldsymbol{t}_{N_0+N_1}]_{N_1\times m}^T \tag{28}$$

$$\begin{aligned}\boldsymbol{K}_1 &= \frac{\boldsymbol{I}}{C} + \begin{bmatrix}\boldsymbol{H}_0 \\ \boldsymbol{H}_1\end{bmatrix}^T\begin{bmatrix}\boldsymbol{H}_0 \\ \boldsymbol{H}_1\end{bmatrix} \\ &= \boldsymbol{K}_0 + \boldsymbol{H}_1^T\boldsymbol{H}_1\end{aligned} \tag{29}$$

We can transform the formula (26) into (30)

$$\begin{aligned}\boldsymbol{\beta}^{(1)} &= \boldsymbol{K}_1^{-1}(\boldsymbol{H}_0^T\boldsymbol{H}_0 + \boldsymbol{H}_1^T\boldsymbol{H}_1) \\ &= \boldsymbol{K}_1^{-1}\boldsymbol{K}_0\boldsymbol{\beta}^{(0)} + \boldsymbol{K}_1^{-1}\boldsymbol{H}_1^T\boldsymbol{T}_1\end{aligned} \tag{30}$$

Unite the formulas (29) and (30), we can conclude

$$\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(0)} + \boldsymbol{K}_1^{-1}\boldsymbol{H}_1^T(\boldsymbol{T}_1 - \boldsymbol{H}_1\boldsymbol{\beta}^{(0)}) \tag{31}$$

From the formulas (29) and (31), when the (k+1)th data block $\mathbf{D}_{k+1} = \{(\boldsymbol{x}_i, \boldsymbol{t}_i)\}_{i=\sum_{j=0}^{k} N_j+1}^{\sum_{i=0}^{k+1} N_j}$ coming, we can summarize

$$\boldsymbol{K}_{k+1} = \boldsymbol{K}_k + \boldsymbol{H}_{k+1}^T\boldsymbol{H}_{k+1} \tag{32}$$

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \boldsymbol{K}_{k+1}^{-1}\boldsymbol{H}_{k+1}^T(\boldsymbol{T}_{k+1} - \boldsymbol{H}_{k+1}\boldsymbol{\beta}^{(k)}) \tag{33}$$

Where

$$\mathbf{H}_k = \begin{bmatrix} G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_{\sum_{j=0}^{k} N_j+1}) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_{\sum_{j=0}^{k} N_j+1}) \\ \vdots & \ddots & \vdots \\ G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_{\sum_{j=0}^{k+1} N_j}) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_{\sum_{j=0}^{k+1} N_j}) \end{bmatrix} \tag{34}$$

$$\boldsymbol{T}_{k+1} = \left[\boldsymbol{t}_{1+\sum_{j=0}^{k} N_j}^T, \dots, \boldsymbol{t}_{1+\sum_{j=0}^{k+1} N_j}^T\right]_{N_{k+1}\times m}^T \tag{35}$$

From the formulas (32) and (33), when output weight $\boldsymbol{\beta}$ is calculated, we need to carry out an matrix inversion calculation, but the calculated amount of the pseudo inverse of the matrix is very large, so we use Woodbury formula (Zhang, 2013) to diminish the computation and the method is as

$$\begin{aligned}\boldsymbol{K}_{k+1}^{-1} &= (\boldsymbol{K}_k + \boldsymbol{H}_{k+1}^T\boldsymbol{H}_{k+1})^{-1} \\ &= \boldsymbol{K}_k^{-1} - \boldsymbol{K}_k^{-1}\boldsymbol{H}_{k+1}^T(\boldsymbol{I} + \boldsymbol{H}_{k+1}\boldsymbol{K}_k^{-1}\boldsymbol{H}_{k+1}^T)^{-1}\boldsymbol{H}_{k+1}\boldsymbol{K}_k^{-1}\end{aligned} \tag{36}$$

From the formulas (33) and (36), we can learn a new data block incrementally based on current classifiers. If the change in data streams is sight, owing to updating mechanism, the classifier can recursively learn the new concept which does not need to retrain classifiers over and over again.

3.3. Concept drift detection based on IDS-ELM

Concept drift is very common in data streams; different types of concept drift have different processing methods (Webb, Hyde, Cao, Nguyen, & Petitjean). From Section 3.2, we know IDS-ELM ceaselessly corrects the classification models, so it can handle gradual concept drift. When the concept in data streams changes greatly, the current classification models are not fit for the current data stream, so the error rate will be distinctly increased; many of them have to be deleted and the remainders need to learn the new concept. In data streams classification problem, accuracy is usually used to detect the change of concept; if accuracy changes significantly, the algorithm will deem a concept drift has appeared. To measure the significance, *Hoeffding Bound* (Domingos & Hulten, 2000; Rutkowski, Pietruczuk, Duda, & Jaworski, 2013) is used.

Theorem 1 (Hoeffding Bound). *By observing an independent random variable* r *for* n *times, the range of* r *is* R *and the observed mean values of* r *is* $\bar{r}$. *With the confidence level 1-*α, *the true value of* R *is at least* $\bar{r} - \varepsilon$, *where* $\varepsilon = \sqrt{\frac{R^2\ln(\frac{1}{\alpha})}{2n}}$.

According to the theorem, we know if the difference of the accuracies of the two adjacent data blocks is more than ε, it is thought that concept drift has occurred; if the difference is less than or equal to ε, it reveals that the data distribution is stable in data streams, at this time, the algorithm will incrementally update classifiers using the method showed in Section 3.2. The literature (Zhang et al., 2009) tells us that the performance of the ensemble classifiers can be improved by increasing the diversity of classifiers, so in this paper, the activation function of ELM is randomly selected from the formulas(6)–(11) and the ensemble classifiers use a weighted voting mechanism to make decision. Assuming the accuracy of a classifier C_j is v_k, for a data block B_k, the weight of the classifier is as

$$C_j.weight = \frac{1}{1 - v_k + \mu} \tag{37}$$

Where μ is a positive and small constant to avoid the divisor becoming zero.

From the above, IDS-ELM algorithm is summarized as follows.

Algorithm 3. IDS-ELM

Data stream S; the maximum number of the ensemble classifiers k; the neurons'number of ELM in hidden layer L=0; the number of instances in data block *winsize*; threshold ε; ensemble classifiers *ensemble=NULL; signal*=1;

Step 1. If *S=NULL*, goto *step 7*; else get *winsize* new instances to form a data block B_i from S;

Step 2. If *size*(*ensemble*) < k, use *algorithm 2* and the data block to determine the number of hidden layer neurons L, goto *step 3*; else goto *step 5*;

Step 3. Use *algorithm 1* to train a new ELM C_j as a base classifier; the number of the neurons in hidden layer of C_j is L; the activation function of ELM is randomly selected from the formulas(6)–(11); $C_j.weight$ =1; *ensemble* ← *ensemble* ∪ C_j; if *signal*=1, goto *step 4*; else goto *step 5*;

Step 4. Repeat *step 1* – *step 3* until there are k ELMs existing in system; *signal*=0;

Step 5. Use classifiers to calculate the accuracy v_i of B_i with weighted voting mechanism, if $v_{i-1} - v_i \leqslant \varepsilon$, incrementally update *ensemble*, recalculate the weights of *ensemble*, then goto *step 1*; else concept drift has appeared, then goto *step 6*;

Step 6. Recalculate the weights of *ensemble*; sort *ensemble* according to the weights in ascending order; delete the first half of classifiers in *ensemble*; then goto *step 1*;

Step 7. Terminate *algorithm 3* and output *ensemble*

In the Algorithm 3, owing to the new search method, IDS-ELM can obtain a good number of neurons in hidden layer with a fast speed. Additionally, the incremental learning makes IDS-ELM only adjust some parameters of neural network to learn new data blocks one by one when no concept drift happens in data streams. The above measures ensure the algorithm can reach a high accuracy in a short time.

4. Experimental evaluation

In this section, we conduct our IDS-ELM algorithm on the artificial data sets and practical data sets; the artificial data sets are produced by MOA[1] platform (Bifet, Holmes, Kirkby, & Pfahringer, 2010) and the practical data sets are selected from UCI data sets[2]. Here we compare our algorithm with original ELM[3] (Huang et al., 2012; Huang et al., 2006b), Back Propagation neural network (BP) (Han, Kamber, & Pei, 2011), AUE2 (Brzezinski & Stefanowski, 2014), Learn++.MF (Polikar, Depasquale, Mohammed, Brown, & Kuncheva, 2010), H-ELM, SRELM, OP-ELM[4]. The experimental configurations of the computer running algorithms are as follows: Windows 7 OS, 4 GB RAM memory, Intel Core 2.94 G dual cores processor. All algorithms are implemented in MATLAB R2013a. The related parameters in this paper are set as follows: significance level α =0.05, k=5, n=10 × *winsize*, $\mu = 1 \times 10^{-6}$.

4.1. Data sets descriptions

In order to fully validate the performance of the proposed algorithms, different kinds of data sets are carried out in the experiments, including categorical, numerical and mixed. The related information of the data sets is showed in the Table 1. In this section, we only give a brief introduction to the artificial data sets. Except from the above, the 16 data sets are normalized into [-1,1].

In STAGGER data set, there are three attributes for every instance and each attribute has three values: *color*={*red, blue, green*}, *size*= {*small, medium, large*}, *shape*={*circle, square, triangle*}; *class*={*false, true*}; the numbers of the data for each class are balanced. In Waveform data set, the goal of the task is to differentiate the three types of waveform, each of which is generated from a combination of two or three base waves; in the process of generating the data set, 2% noise data is added. LED data set represents the results of seven LED digital tubes display; the display number is from 0 to 9 and all attributes values are either 0 or 1; according to whether the corresponding light is on or not for the decimal digit, so the first 7 attributes represent the lighted state of the digital tubes, and the last 14 attributes are redundant. Hyperplane data set is a gradual concept drift data set; in the data set, a d dimensions hyperplane **X** satisfies the following mathematical expression:$\sum_{i=1}^{d} a_i x_i = a_0$, where $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$, $x_i \in [01]$, $a_i \in [-10, 10]$; if $\sum_{i=1}^{d} a_i x_i \geqslant a_0$, the label of instance is marked as a positive example, otherwise the label of the instance is marked as a negative example; the data set contains 10% noise. In Agrawal data set, instances are produced with nine attributes; the six of them are numerical and the remaining three attributes are categorical. At last, these instances are divided into two classes: group A and group B, which represents whether the loan should be approved or not. The generator producing RandomTree data set constructs a decision tree by choosing attributes at random to split and assigns a random class label to each leaf. Once the tree is built, new examples can be generated by assigning uniformly distributed random values to attributes to determine the class labels by traversing the tree. We can adjust the data set by adjusting the parameters of the tree. In RandomTree dataset, the first three attributes are categorical and the last 20 attributes are numerical; all data are divided into six classes. SEAGenerator is a unbalanced and binary class data set; it contains three attributes and all of them are numerical, but only the first two attributes relate with the labels and the last one is redundant; there are four concepts in the data set and the data of every data block satisfy inequation: $f_1 + f_2 \leqslant \theta$, where f_1 and f_2 are the values of the first two attributes and θ is a threshold which is usually selected from 9, 8, 7 and 9.5; four different values of θ produces four concepts for SEAGenerator.

Table 1
Specifications of the data sets.

Data sets		Attri	class	instance	Types
Artificial data sets	STAGGER	3	2	50,000	categorical
	Waveform	21	3	50,000	numerical
	LED	24	10	50,000	categorical
	Hyperplane	40	2	50,000	numerical
	Agrawal	9	2	50,000	mixed
	RandomTree	25	6	50,000	mixed
	SEAGenerator	3	2	50,000	numerical
Practical data sets	Bank	16	2	45,211	numerical
	Letter	15	26	20,000	numerical
	EEG Eye	14	2	14,980	numerical
	Magic	10	2	19,020	numerical
	Mushroom	21	2	8124	categorical
	Nursery	7	5	12,960	categorical
	Student	32	3	5820	mixed
	Adult	14	2	32,561	mixed
	Shuttle	8	5	58,000	numerical

4.2. Experimental results

In order to compare the performance of IDS-ELM with the conventional algorithms: ELM, BP, AUE2 and Learn++.MF, we test the algorithms on the experimental datasets and we selected a variety of functions as the ELM and BP activation functions. Considering the weights of IDS-ELM, ELM and BP were randomly chosen, so we ran these algorithms for 10 times, the final result was from the average of the 10 results. The test results and time overhead are showed in the Tables 2, 3 and 4.

From the Table 2, we can see that the accuracy of IDS-ELM is better than AUE2, Learn++.MF and ELM on most data sets. For BP algorithms, when the activation function is *sigmoid*, the algorithm is better than IDS-ELM on 13 data sets; when the activation function is *radbas*, BP algorithm is also better than IDS-ELM on 13 data sets; but when the activation function is *hardlim*, BP algorithm is only better than IDS-ELM on 6 data sets, in other words, the performance of BP algorithm is worse than IDS-ELM. It is obvious that

[1] http://moa.cms.waikato.ac.nz/.
[2] http://archive.ics.uci.edu/ml/datasets.html.
[3] http://www.ntu.edu.sg/home/egbhuang/elm_codes.html.
[4] http://www.gnu.org/copyleft/gpl.html.

Table 2
Results of algorithms testing on the different data sets.

	Sigmoid		Radbas		Hardlim		IDS-ELM	AUE2	Learn++.MF	C	winsize
	ELM	BP	ELM	BP	ELM	BP					
STAGGER	1	0.9990	0.9988	0.9997	0.9986	1	0.949	0.9427	1.0000	3	1000
Waveform	0.7120	0.8302	0.5951	0.8073	0.6688	0.7320	0.7950	0.8283	0.7806	3	1000
LED	0.7840	0.9315	0.4362	0.9219	0.7513	0.5838	0.7995	0.7589	0.6206	3	1000
Hyperplane	0.7377	0.8513	0.5188	0.7925	0.7250	0.6927	0.7775	0.7846	0.6616	10	1000
Agrawal	0.6974	0.9635	0.6618	0.9496	0.6114	0.6588	0.6722	0.6282	0.6204	10	1000
RandomTree	0.3107	0.3989	0.2277	0.3847	0.3039	0.3501	0.4069	0.3994	0.3872	10	1000
SEAGenerator	0.8622	0.9689	0.8706	0.9670	0.8774	0.9167	0.8880	0.9374	0.9467	3	1000
Bank	0.7282	0.8549	0.7885	0.8396	0.7901	0.8522	0.8530	0.8846	0.8739	3	1000
Letter	0.5345	0.4463	0.4274	0.4600	0.4190	0.2220	0.4151	0.7917	0.6576	8	500
EEG Eye	0.6445	0.6710	0.6516	0.7051	0.6725	0.6937	0.6750	0.572	0.5854	3	500
Magic	0.9659	0.9734	0.9675	0.9614	0.9677	0.9635	0.8400	0.6316	0.8717	3	500
Mushroom	0.9693	0.9604	0.9228	0.9496	0.9461	0.8819	0.9170	0.8250	0.9268	3	200
Nursery	0.4720	0.8077	0.6035	0.7936	0.6459	0.6737	0.6582	0.6265	0.6488	3	500
Student	0.8995	0.8994	0.9050	0.8994	0.8994	0.8994	0.7979	0.4979	0.6407	3	200
Adult	0.7161	0.8226	0.7104	0.7985	0.7212	0.7845	0.7731	0.7692	0.7719	3	1000
Shuttle	0.9441	0.8346	0.9629	0.7622	0.9874	0.7205	0.9218	0.8665	0.9877	3	1500

Table 3
Training time of algorithms testing on the different data sets.

	Sigmoid		Radbas		Hardlim		IDS-ELM	AUE2	Learn++.MF
	ELM	BP	ELM	BP	ELM	BP			
STAGGER	15.1879	14.8460	25.7894	15.1672	19.2133	11.0526	22.6291	1.1056	0.4771
Waveform	18.186	180.192	17.3215	113.8716	19.2032	121.8453	37.7122	1.7999	0.9129
LED	19.2827	283.4271	9.7364	172.1357	11.8113	224.7418	66.7037	3.8668	1.0318
Hyperplane	15.4838	88.2735	17.0222	162.3093	18.1988	253.0390	55.3992	2.1491	1.2242
Agrawal	17.0919	36.9174	16.9129	42.6376	13.6972	28.2382	18.7083	1.1929	0.4831
RandomTree	16.2520	368.8589	11.9754	515.5934	19.7426	198.4962	44.1668	3.0365	1.1087
SEAGenerator	11.9504	18.7776	14.6287	19.2220	14.0446	13.8478	13.4078	0.99352	0.41556
Bank	14.0328	44.2776	12.2594	44.0951	16.1663	27.7447	14.4432	1.0300	0.5750
Letter	2.6160	240.4929	1.9515	474.6340	2.1083	328.5714	13.7545	3.7381	0.3695
EEG Eye	1.1651	5.1201	1.4788	5.4994	0.9122	6.3952	5.4400	0.3575	43.6958
Magic	1.5035	6.1444	1.8268	5.3827	2.2720	4.8014	5.2734	0.4803	0.2850
Mushroom	0.3684	5.6977	0.3579	6.3101	0.2868	5.8755	3.5942	0.6578	0.2651
Nursery	1.1248	7.5555	1.1968	7.8134	1.0472	7.9494	5.5113	1.6967	0.1902
Student	0.2333	3.5093	0.2394	3.5559	0.2362	3.6291	3.2706	0.2915	10.3783
Adult	13.4052	76.7632	11.0449	68.2534	9.0949	24.3345	17.8900	0.5370	0.4146
Shuttle	40.9569	100.3948	28.8437	162.7517	34.7804	61.5044	36.8103	8.4959	0.5257

Table 4
Testing time of algorithms testing on the different data sets.

	Sigmoid		Radbas		Hardlim		IDS-ELM	AUE2	Learn++.MF
	ELM	BP	ELM	BP	ELM	BP			
STAGGER	14.8161	10.2161	25.9204	14.7542	18.7005	11.4936	1.1861	6.0040	2.302
Waveform	18.2566	172.8051	17.5097	111.9578	19.1882	107.1093	4.3724	15.8932	4.5529
LED	19.3385	268.3405	9.8040	165.2984	12.1022	209.3725	10.2065	16.5520	5.0687
Hyperplane	15.3238	81.4869	17.0282	171.4107	17.8698	251.8899	11.8561	21.9077	11.8561
Agrawal	17.0776	37.5925	16.8574	50.7133	13.2745	27.7827	2.1739	6.3993	2.5046
RandomTree	16.8190	373.4720	11.8713	543.3526	19.4801	193.5932	7.1649	18.6196	7.4000
SEAGenerator	11.7450	18.7473	14.5776	20.1848	13.7882	13.2722	1.2249	5.2388	2.2558
Bank	13.8060	48.8350	12.2264	53.5312	15.9988	28.4303	2.3149	10.6173	2.9135
Letter	2.6006	238.5782	1.8361	237.1080	2.0602	295.3646	2.0418	3.9935	1.7646
EEG Eye	1.0614	4.9174	1.4549	5.6615	0.8192	5.9366	1.1066	1.9884	1.2057
Magic	1.4026	5.9597	1.8007	5.2261	2.1682	4.6649	0.8047	3.1935	1.3933
Mushroom	0.3458	5.4773	0.3246	5.9739	0.2504	5.7682	0.9911	2.5805	1.1772
Nursery	1.1328	6.8940	1.0770	8.2423	1.0786	7.5415	0.5680	6.7584	0.88329
Student	0.2126	3.3724	0.2086	3.4638	0.2116	3.4756	1.0014	1.3859	0.8070
Adult	13.3351	75.3223	11.0908	70.2657	9.1142	25.2618	2.1159	5.4663	2.2030
Shuttle	40.8890	71.3626	28.6940	128.5360	34.5192	56.1131	2.8543	20.8237	3.7421

BP algorithm is very time-consuming and the time consumption of BP algorithm is several or even several hundred times as much as IDS-ELM. So if the active function of BP algorithm is selected properly, it will get a very high accuracy, but we can also know that activation functions have a great influence on the performance of BP algorithm and IDS-ELM is better than BP in the aspect of stability. From the Tables 3 and 4, the time overhead of IDS-ELM is mainly spent on the training process; once the classifier is trained, the testing time of IDS-ELM will be much better than the other algorithms. After analyzing the data in the Tables 3 and 4, the testing time overhead of IDS-ELM is better than ELM, AUE2 and

Table 5
Winsize values of algorithms testing on the different data sets.

Data set	Adult	Agrawal	Bank	EEG Eye	Hyperplane	Magic	Mushroom	SEAGenerator	STAGGER
winsize	500	1000	1000	500	1000	600	500	1000	1000

Table 6
Results of algorithms testing on the different data sets.

Data set	IDS-ELM	H-ELM	SRELM	OP-ELM
Adult	0.8064 ± 0.0206	0.7815 ± 0.0202	0.7675 ± 0.0277	0.8109 ± 0.0172
Agrawal	0.6932 ± 0.0301	0.6481 ± 0.0674	0.6722 ± 0.0152	0.6660 ± 0.0252
Bank	0.8598 ± 0.1398	0.8803 ± 0.1084	0.8641 ± 0.1465	0.8915 ± 0.1136
EEG Eye	0.7480 ± 0.2515	0.6296 ± 0.4118	0.4196 ± 0.3770	0.6939 ± 0.2281
Hyperplane	0.8011 ± 0.0206	0.8853 ± 0.0219	0.5880 ± 0.0373	0.7957 ± 0.0193
Magic	0.7963 ± 0.2654	0.7058 ± 0.4530	0.8588 ± 0.3082	0.9476 ± 0.0802
Mushroom	0.9560 ± 0.0174	0.9870 ± 0.0123	0.7638 ± 0.1990	0.9755 ± 0.0215
SEAGenerator	0.9239 ± 0.0265	0.9535 ± 0.0146	0.9120 ± 0.0699	0.9650 ± 0.0091
STAGGER	0.9794 ± 0.0046	1.0000 ± 0.0000	0.9435 ± 0.0852	1.0000 ± 0.0000
Average	0.8405 ± 0.0568	0.8301 ± 0.1233	0.7544 ± 0.1407	0.8606 ± 0.0571

Table 7
Numbers of neurons testing on the different data sets.

Data set	IDS-ELM	H-ELM	SRELM	OP-ELM
Adult	10.2333 ± 0.5731	50.7188 ± 1.7083	5.1250 ± 2.7327	32.8125 ± 9.6668
Agrawal	6.1283 ± 0.5970	158.6800 ± 6.8577	8.9327 ± 2.1237	31.2000 ± 6.5000
Bank	9.4441 ± 1.5172	100.1500 ± 0.9881	7.2273 ± 2.4089	36.1364 ± 9.7507
EEG Eye	10.4704 ± 0.5534	49.5833 ± 0.9962	5.6429 ± 1.7368	36.4286 ± 15.8634
Hyperplane	35.2108 ± 2.9075	100.4783 ± 1.4100	5.5600 ± 2.2000	53.4000 ± 4.0104
Magic	6.8600 ± 0.8222	59.0769 ± 0.2774	5.3333 ± 2.4976	19.6667 ± 12.0218
Mushroom	16.7000 ± 1.7163	49.6667 ± 0.8165	6.5000 ± 3.0237	55.6250 ± 8.2104
SEAGenerator	1.8775 ± 0.2823	100.7826 ± 3.2746	5.9600 ± 2.5080	5.4000 ± 11.4492
STAGGER	2.4000 ± 0.1985	99.0000 ± 0.0000	6.5200 ± 2.2383	27.8000 ± 4.5826
Average	11.0360 ± 1.0186	85.3485 ± 1.8143	6.3112 ± 2.3855	37.6077 ± 9.1174

Learn++.MF on most data sets and the lager the data set is, the more obvious the advantages will be.

According to the results, although BP is better than IDS-ELM in accuracy, the time overhead of BP is far more than IDS-ELM; in data streams environment, such a high time consumption is inadequate for dealing with massive data. In test results, the difference of the time overheads between ELM and IDS-ELM is small, but IDS-ELM has a better accuracy. As for AUE2 and Learn++.MF, although the algorithms are better than IDS-ELM in training time overhead, IDS-ELM is better than the two algorithms in accuracy and testing time; if we require a high accuracy, IDS-ELM should be used with a higher priority. From the above, we can see IDS-ELM is a relatively better algorithm.

BP, ELM and IDS-ELM algorithms are neural networks. The number of the neurons in hidden layer reflects the complexity of the neural networks and the more neurons the hidden layer has, the higher the complexity of the neural networks will be. To compare the complexity of the neural networks, we test H-ELM, SRELM, OP-ELM and IDS-ELM algorithms on 9 representative data sets; in the experiment, the value of winsize on different data sets is shown in theTable 5 and the test results are shown in theTables 6 and 7.

From the Table 6, we can see that, in the aspect of average accuracy, H-ELM and OP-ELM are better than SRELM and IDS-ELM is better than H-ELM and SRELM; OP-ELM is the best of all. By analysing the result, we can observe that OP-ELM is only more 0.0201 than IDS-ELM in average accuracy; however, the average number of neurons of OP-ELM is 3.4077 times as much as IDS-ELM which means the structure of neural network of OP-ELM is more complex than IDS-ELM; so IDS-ELM gets a balance between accuracy and structure of neural network. If we observe the Table 7, we can find SRELM can select fewer neurons for ELM than IDS-ELM. But combining the data from the Tables 6 and 7, it is obvious the accuracy of IDS-ELM is better than SRELM. From the execution of SRELM, we have found because particle swarm optimization algorithm (PSO) is used; SRELM costs so much time to find the optimal number of neurons, and the elapsed time of SRELM is far more than IDS-ELM. The advantages of IDS-ELM are more obvious than SRELM although the structure of neural network of SRELM is more simple.

In order to investigate the effect of sliding window size on the performance of the algorithms, we test IDS-ELM, AUE2 and Learn++.MF with different winsize values. Because the weights are randomly selected, we perform IDS-ELM for 10 times; every final result of IDS-ELM is the average value of the 10 results. The test results are showed in the Tables 8–16. In the experiments, we choose KNN as the base classifiers of AUE2 and Learn++.MF. To measure the performance of IDS-ELM and the conventional data streams classification algorithms: AUE2 and Learn++.MF, we use coefficient of variation (CV) (Brown, 2008) to judge the stability of the algorithms and *CV* is computed as the formula (38).

$$CV = \frac{std}{mean} \tag{38}$$

Where *std* is standard deviation and *mean* is average value. The smaller the *CV* value is, the fewer differences the data will have and the performance of algorithm is more steady, otherwise, the performance is more fluctuant.

Table 8
Accuracies of IDS-ELM testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	0.9623	0.9559	0.9500	0.9541	0.9349	0.9507	0.0096
Waveform	0.7953	0.7923	0.7978	0.7940	0.8004	0.7939	0.0037
LED	0.7679	0.7808	0.7589	0.7513	0.7587	0.7561	0.0139
Hyperplane	0.7562	0.7622	0.7763	0.7643	0.7389	0.7455	0.0178
Agrawal	0.6727	0.6617	0.6582	0.6617	0.6645	0.6507	0.0109
RandomTree	0.3875	0.3968	0.3894	0.3970	0.4021	0.3975	0.0139
SEAGenerator	0.8901	0.8833	0.8501	0.8746	0.8628	0.8621	0.0171
Bank	0.8254	0.8312	0.8310	0.8455	0.8330	0.8424	0.0091
Adult	0.7884	0.7846	0.7847	0.7874	0.7874	0.7771	0.0053
winsize	100	200	300	400	450	550	CV
Letter	0.3768	0.4315	0.3871	0.3847	0.3687	0.3956	0.0562
EEG eye	0.6546	0.6253	0.6439	0.5773	0.5194	0.6637	0.0905
Magic	0.9692	0.9357	0.8838	0.9330	0.8601	0.8266	0.0595
Nursery	0.6804	0.6272	0.6383	0.6296	0.6342	0.6276	0.0320
winsize	50	100	150	180	210	250	CV
Mushroom	0.9635	0.9322	0.9347	0.9225	0.9117	0.9093	0.0213
Student	0.9635	0.9322	0.7382	0.8211	0.8538	0.9263	0.0969

Table 9
Training time of IDS-ELM testing with different winsize.

winsize	100	300	500	700	900	1200	CV
STAGGER	5.4017	6.0509	8.4669	10.2734	15.7565	18.8592	0.5019
Waveform	28.5457	31.6793	40.1537	34.6976	35.8236	49.3659	0.1997
LED	37.0325	38.5264	48.4504	46.6290	60.9928	45.5152	0.1856
Hyperplane	57.6496	57.6974	47.7636	64.0210	51.9482	66.8727	0.1241
Agrawal	11.0167	9.6954	12.0931	15.3048	13.7612	16.4409	0.1980
RandomTree	47.5039	50.1525	55.3900	50.1951	45.8725	64.1405	0.1279
SEAGenerator	5.3085	7.1264	9.6005	11.7092	12.8913	19.5737	0.4566
Bank	13.4965	14.2977	11.2994	18.0477	14.4821	20.2007	0.3386
Adult	9.6695	8.7514	11.2994	18.0477	14.4821	20.2007	0.3386
winsize	100	200	300	400	450	550	CV
Letter	11.2838	10.8711	10.1828	11.2241	12.2078	9.3462	0.0910
EEG eye	3.8596	4.8860	4.5188	3.4731	4.2372	6.4224	0.2266
Magic	5.8507	5.6173	6.0242	7.5456	6.9145	7.1226	0.1205
Nursery	3.9804	3.6900	3.5504	3.4152	2.7649	2.8859	0.1393
winsize	50	100	150	180	210	250	CV
Mushroom	3.1894	4.0149	4.9884	4.3959	4.8927	4.0802	0.1553
Student	3.3276	4.5508	3.2049	2.8304	4.6967	3.1990	0.2160

Table 10
Testing time of IDS-ELM testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	1.1606	1.1812	0.9499	1.0104	0.7809	0.8537	0.1629
Waveform	6.3590	6.2994	6.2076	6.2074	6.3745	5.3050	0.0667
LED	7.6761	7.4219	8.0537	7.6485	8.1970	7.5520	0.0389
Hyperplane	12.6290	12.6731	11.9149	10.7473	8.4256	10.4123	0.1122
Agrawal	2.9517	2.3963	2.6189	2.5994	2.1567	2.4259	0.1060
RandomTree	10.2111	9.7626	11.9149	10.7473	8.4256	10.4123	0.1122
SEAGenerator	1.1961	1.1543	1.3023	1.1265	0.8296	0.9021	0.1673
Bank	3.6127	2.8925	3.4595	3.6347	3.6526	2.9415	0.1053
Adult	2.3461	2.1352	2.1283	2.2998	1.8368	1.7958	0.1099
winsize	100	200	300	400	450	550	CV
Letter	2.1862	2.2888	2.2485	2.1244	2.3340	1.6540	0.1164
EEG eye	1.1510	1.2227	1.0105	0.6936	0.9756	1.0714	0.2030
Magic	1.5628	1.3715	1.1506	1.4549	1.3335	1.1802	0.1179
Nursery	0.6670	0.5501	0.6293	0.5938	0.5157	0.3581	0.1981
winsize	50	100	150	180	210	250	CV
Mushroom	0.8961	1.1553	1.0303	0.9210	1.223	0.9115	0.1358
Student	1.0390	1.3399	0.7292	0.9409	0.9796	0.9052	0.2035

Table 11
Accuracies of AUE2 testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	0.9554	0.9347	0.9621	0.9634	0.9598	0.9611	0.0113
Waveform	0.8574	0.8508	0.8440	0.8438	0.8355	0.8313	0.0118
LED	0.7606	0.7335	0.7168	0.6883	0.7973	0.7815	0.0550
Hyperplane	0.7300	0.7118	0.6936	0.6795	0.6735	0.7716	0.0516
Agrawal	0.6696	0.6607	0.6522	0.6392	0.6391	0.6242	0.0271
RandomTree	0.4498	0.4444	0.4319	0.4245	0.4173	0.39829	0.0440
SEAGenerator	0.9550	0.9470	0.9479	0.9467	0.9453	0.9634	0.0074
Bank	0.8857	0.8861	0.8939	0.8935	0.8887	0.9011	0.0066
Adult	0.7705	0.7661	0.7756	0.7760	0.7713	0.7606	0.0077
winsize	100	200	300	400	450	550	CV
Letter	0.7582	0.7604	0.7846	0.7956	0.7988	0.8192	0.0292
EEG eye	0.5669	0.5612	0.5293	0.5864	0.5124	0.5387	0.0571
Magic	0.6455	0.6523	0.6486	0.6522	0.6383	0.6471	0.0080
Nursery	0.6489	0.6525	0.6414	0.6352	0.6471	0.5990	0.0310
winsize	50	100	150	180	210	250	CV
Mushroom	0.7704	0.7530	0.8101	0.7912	0.8258	0.8103	0.0347
Student	0.6121	0.6121	0.5877	0.6250	0.5916	0.5546	0.0421

Table 12
Training time of AUE2 testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	86.4369	9.5433	3.5644	2.6798	1.3104	0.7206	1.9555
Waveform	118.7110	15.3804	6.2450	3.4311	2.3616	1.4859	1.8852
LED	184.0658	31.8465	13.1409	7.4093	4.8869	3.1282	1.7421
Hyperplane	113.4361	14.9986	6.7833	3.7935	2.6126	1.6941	1.8477
Agrawal	89.9786	11.1038	4.3476	2.3850	1.5053	0.9140	1.9200
RandomTree	141.2959	25.5404	10.2509	5.6624	3.8518	2.4168	1.8212
SEAGenerator	83.5994	10.1535	3.9122	2.1271	1.3221	0.7991	1.9318
Bank	17.6726	9.9511	4.0319	2.1864	1.5230	0.9022	1.0881
Adult	39.6253	4.9927	1.9956	1.1480	0.7454	0.4488	1.9000
winsize	100	200	300	400	450	550	CV
Letter	57.3281	23.6322	10.0918	6.4147	4.7862	3.3782	1.1813
EEG eye	6.7568	1.9257	0.9278	0.5769	0.4684	0.3453	1.3520
Magic	11.2505	2.6676	1.2906	0.7562	0.6364	0.4569	1.4760
Nursery	7.3178	1.8908	0.9046	0.5681	0.4365	0.2842	1.4290
winsize	50	100	150	180	210	250	CV
Mushroom	6.8047	2.5559	1.2594	0.8590	0.7004	0.4990	1.1419
Student	4.2456	1.1630	0.5787	0.4167	0.3220	0.2293	1.3353

Table 13
Testing time of AUE2 testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	396.3004	45.4808	18.0450	11.9620	6.7754	3.8641	1.7625
Waveform	377.7649	54.7613	28.2951	19.8391	16.7249	14.4533	1.6884
LED	271.6658	53.6601	28.3774	20.4123	17.3045	14.9604	1.4892
Hyperplane	389.5625	59.9526	34.9469	26.2310	22.9216	20.2062	1.5854
Agrawal	406.2628	49.7001	20.5667	11.6399	7.6559	5.0340	1.9043
RandomTree	287.3797	59.5329	30.7544	22.7320	19.4149	17.3394	1.4580
SEAGenerator	381.7968	46.7684	18.6917	9.9837	6.5192	4.0371	1.9196
Bank	69.0791	42.9587	21.4443	15.3006	12.8944	10.3231	0.8036
Adult	159.5460	21.9050	10.7438	7.7054	6.5839	5.2984	1.7329
winsize	100	200	300	400	450	550	CV
Letter	34.2934	17.4387	8.222	6.1741	4.7602	3.7146	0.9480
EEG eye	32.8668	9.1448	4.7331	2.9223	2.5626	2.1636	1.3172
Magic	60.8081	15.1438	7.4669	4.6601	4.0788	3.0367	1.4150
Nursery	27.3966	7.3897	3.7455	2.4804	2.2467	1.5572	1.3362
winsize	50	100	150	180	210	250	CV
Mushroom	25.7782	9.94394	4.6839	3.2831	2.6430	1.9725	1.1455
Student	19.0960	5.1483	2.5339	1.8248	1.3758	1.0610	1.3485

Table 14
Accuracies of Learn++.MF testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	0.9587	0.8463	0.9548	0.9581	0.9363	0.9275	0.0463
Waveform	0.7796	0.7896	0.7864	0.7892	0.7788	0.7854	0.0059
LED	0.5564	0.6189	0.6397	0.6235	0.6550	0.6713	0.0636
Hyperplane	0.6491	0.6582	0.6608	0.6618	0.6734	0.6533	0.0127
Agrawal	0.6399	0.6397	0.6858	0.6182	0.6548	0.6760	0.0386
RandomTree	0.2972	0.2996	0.3218	0.3284	0.3433	0.3217	0.0551
SEAGenerator	0.9434	0.9448	0.9542	0.9379	0.9379	0.9557	0.0069
Bank	0.8918	0.8937	0.8958	0.8926	0.8870	0.8974	0.0040
Adult	0.7657	0.7794	0.7681	0.7918	0.7779	0.7696	0.0125
winsize	100	200	300	400	450	550	CV
Letter	0.4596	0.5837	0.6126	0.6589	0.6717	0.6883	0.1378
EEG eye	0.6203	0.6142	0.4521	0.6259	0.4857	0.6639	0.1447
Magic	0.9718	0.9502	0.9067	0.8696	0.8764	0.8643	0.0550
Nursery	0.6852	0.6625	0.6975	0.5252	0.5395	0.5648	0.1270
winsize	50	100	150	180	210	250	CV
Mushroom	0.9746	0.9755	0.9775	0.97146	0.9065	0.8725	0.0479
Student	0.9152	0.8486	0.7456	0.7500	0.6418	0.6455	0.1449

Table 15
Training time of Learn++.MF testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	3.5689	1.2331	0.7732	0.5738	0.4520	0.3718	1.0485
Waveform	3.5367	1.3525	1.0362	0.9315	0.9511	1.0181	0.6951
LED	3.5904	1.3955	1.1046	0.9938	1.1194	1.1430	0.6449
Hyperplane	3.8505	1.5147	1.1328	1.1541	1.2228	1.3807	0.6196
Agrawal	76.5215	24.9305	0.8562	0.6694	0.5269	0.4204	1.7661
RandomTree	3.6991	1.4269	1.1388	1.0606	1.0816	1.2714	0.6392
SEAGenerator	3.5589	1.2577	0.7695	0.5847	0.4692	0.3863	1.0343
Bank	3.4015	1.2195	0.8061	0.6733	0.5701	0.5545	0.9168
Adult	2.3996	0.8627	0.5730	0.5103	0.5011	0.3347	0.8940
winsize	100	200	300	400	450	550	CV
Letter	1.4719	0.7389	0.5591	0.4624	0.4540	0.3776	0.6033
EEG eye	67.2887	74.7111	115.3149	114.8503	66.6204	71.3951	0.2759
Magic	12.0567	0.6731	11.7310	13.0676	13.3667	0.2764	0.7351
Nursery	0.8898	0.4490	0.3096	0.2475	0.2350	0.2164	0.6609
winsize	50	100	150	180	210	250	CV
Mushroom	1.2206	0.5260	0.3649	0.3062	0.2682	0.2362	0.7672
Student	0.7389	0.3782	10.0766	0.2360	0.1929	12.0131	1.4067

Table 16
Testing time of Learn++.MF testing with different winsizes.

winsize	100	300	500	700	900	1200	CV
STAGGER	15.8610	5.9542	3.9385	3.0836	2.4853	2.0443	0.9407
Waveform	15.5342	6.5430	5.2735	4.7351	4.7344	4.8844	0.6129
LED	15.4308	6.7125	5.3122	5.0081	5.3506	5.4103	0.5790
Hyperplane	15.4308	6.9472	5.6906	5.6798	6.0222	6.5979	0.4927
Agrawal	17.7274	6.5224	4.3272	3.4938	2.8739	2.2494	0.9417
RandomTree	15.6474	6.8312	5.5149	5.3069	5.3105	5.9445	0.5473
SEAGenerator	15.9702	6.0673	3.9822	3.0256	2.5343	2.1233	0.9369
Bank	14.8611	5.9201	4.1278	3.6502	3.0429	2.8818	0.7998
Adult	10.7585	4.1659	2.9174	2.5639	2.4579	1.6559	0.8245
winsize	100	200	300	400	450	550	CV
Letter	6.4091	3.4819	2.6907	2.2360	2.3449	1.9146	0.5253
EEG eye	4.4751	2.5025	1.1388	1.0777	1.1039	1.1987	0.7146
Magic	5.7820	3.0563	2.1276	1.6198	1.5386	1.3109	0.6574
Nursery	3.8084	2.1287	1.4780	1.1962	1.1566	1.0024	0.5927
winsize	50	100	150	180	210	250	CV
Mushroom	4.4478	2.3413	1.5961	1.3726	1.1854	1.0643	0.6401
Student	3.1245	1.6075	1.0482	0.9834	0.7813	0.6260	0.6800

From the Tables 8 and 11, we can see, the accuracies of IDS-ELM and AUE2 varies with the change of the winsize values; for the CV values of the two algorithms, IDS-ELM is better than AUE2 on 8 data sets; it tells us when the number of data increasing in sliding window, the change of the accuracy of IDS-ELM is smaller, in other words, the stability of IDS-ELM is better than AUE2. From the Tables 9, 10 and 13, for CV values, IDS-ELM is better than AUE2 on 15 data sets, so IDS-ELM has more advantages in time consumption with winsize increasing. From the Tables 8 and 14, for CV values, the accuracy of IDS-ELM is better than AUE2 in 15 data sets. From the Table 9, 10, 15 and 16, for the time overhead, IDS-ELM is better than Learn++.MF on 15 data sets; the test results tell us IDS-ELM is better than Learn++.MF. In summary, we can conclude that, in data streams environment, comparing with the conventional algorithms for data streams classification with different winsizes, IDS-ELM is more stable than the conventional algorithms for data streams classification.

5. Conclusions

In this paper, a fast incremental extreme learning machine algorithm for data streams classification called IDS-ELM was proposed. IDS-ELM uses a fast search method that can effectively reduce the number of the neurons in hidden layer and ensure the performance of the algorithm is not affected. In addition, IDS-ELM uses a series of measures to deal with concept drift in data streams environment which extends the scope of the algorithm. In the experiments, the results showed that, comparing with the traditional algorithms, IDS-ELM can improve classification accuracy and speed; the performance of the algorithm keeps a quite good stability when the sliding window changing.

In the work, in order to eliminate the influence of an ill-conditioned matrix on the classification results, a ridge parameter C is used in IDS-ELM. However, how to select a reasonable ridge parameter ensuring the accuracy, speed and stability is our future study.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant no.61202018. Authors would like to gratefully acknowledge the reviewers for their valuable comments.

References

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning and Research, 11*, 1601–1604.

Brown, C. (2008). *Encyclopedia of statistical sciences*. John Wiley & Sons, Inc.

Brzezinski, D., & Stefanowski, J. (2014). Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks & Learning Systems, 25*, 81–94.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 71–80). Boston, Massachusetts, USA: ACM.

Du, L., Song, Q., & Jia, X. (2014). Detecting concept drift: An information entropy based method using an adaptive sliding window. *Intelligent Data Analysis, 18*, 337–364.

Farid, D. M., & Rahman, C. M. (2012). Novel class detection in concept-drifting data stream mining employing decision tree. In *Electrical & computer engineering (ICECE), 2012 7th international conference on* (pp. 630–633).

Farid, D. M., Zhang, L., Hossain, A., Rahman, C. M., Strachan, R., Sexton, G., & Dahal, K. (2013). An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications, 40*, 5895–5906.

Fernandez-Navarro, F., Hervas-Martínez, C., Sanchez-Monedero, J., & Gutiérrez, P. A. (2011). MELM-GRBF: A Modified version of the extreme learning machine for generalized radial basis function neural networks. *Neurocomputing, 74*, 2502–2510.

Gama, J., Sebasti, R., & Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 329–338). Paris, France: ACM.

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques*. Morgan Kaufmann Publishers Inc.

Huang, G., Huang, G. B., Song, S., & You, K. (2015a). Trends in extreme learning machines: A review. *Neural Networks the Official Journal of the International Neural Network Society, 61*, 32–48.

Huang, G. B. (2014). An insight into extreme learning machines: Random neurons, random features and kernels. *Cognitive Computation, 6*, 376–390.

Huang, G. B., Bai, Z., Kasun, L. L. C., & Chi, M. V. (2015b). Local receptive fields based extreme learning machine. *Computational Intelligence Magazine IEEE, 10*, 18–29.

Huang, G. B., Chen, L., & Siew, C. K. (2006a). Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks, 17*, 879–892.

Huang, G. B., Wang, D. H., & Lan, Y. (2011). Extreme learning machines: A survey. *International Journal of Machine Learning & Cybernetics, 2*, 107–122.

Huang, G. B., Zhou, H., Ding, X., & Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. In *IEEE transactions on systems man & cybernetics part b cybernetics a publication of the IEEE systems man & cybernetics society: 42* (pp. 513–529).

Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006b). Extreme learning machine: Theory and applications. *Neurocomputing, 70*, 489–501.

Huang, Y. W., & Lai, D. H. (2012). Hidden node optimization for extreme learning machine. *Aasri Procedia, 3*, 375–380.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 97–106). San Francisco, California: ACM.

Ienco, D., Bifet, A., Pfahringer, B., & Poncelet, P. (2014). Change detection in categorical evolving data streams. In *Proceedings of the 29th annual ACM symposium on applied computing* (pp. 792–797). Gyeongju, Republic of Korea: ACM.

Lemaire, V., Salperwyck, C., & Bondu, A. (2015). A survey On supervised classification on data streams. In E. Zimányi, & R.-D. Kutsche (Eds.), *Business intelligence: 4th European summer school, eBISS 2014, Berlin, Germany, July 6–11* (pp. 88–125). Cham: Springer International Publishing. Tutorial Lectures.

Li, P., Wu, X., Hu, X., & Wang, H. (2015). Learning concept-drifting data streams with random ensemble decision trees. *Neurocomputing, 166*, 68–83.

Li, X., Mao, W., & Jiang, W. (2014). Multiple-kernel-learning-based extreme learning machine for classification design. *Neural Computing & Applications*, 1–10.

Li, X., & Yu, W. (2015). Data stream classification for structural health monitoring via on-line support vector machines. In *Big data computing service and applications (bigdataservice), 2015 IEEE first international conference on* (pp. 400–405).

Liang, N. Y., Huang, G. B., Saratchandran, P., & Sundararajan, N. (2006). A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks, 17*, 1411–1423.

Lima, A. R., Cannon, A. J., & Hsieh, W. W. (2015). Nonlinear regression in environmental sciences using extreme learning machines: A comparative evaluation. *Environmental Modelling & Software, 73*, 175–188.

liobait, I., Technol, X., Bifet, A., Pfahringer, B., & Holmes, G. (2014). Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems, 25*, 27–39.

Liu, X., Gao, C., & Li, P. (2012). A comparative analysis of support vector machines and extreme learning machines. *Neural Networks, 33*, 58–66.

Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., & Lendasse, A. (2010). OP-ELM: Optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks, 21*, 158–162.

Mirza, B., Lin, Z., & Liu, N. (2015). Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift. *Neurocomputing, 149*, 316–329.

Nikovski, D., & Jain, A. (2010). Fast adaptive algorithms for abrupt change detection. *Machine Learning, 79*, 283–306.

Ortíez, D. A., Del, C.-A. J., Ramos-Jimenez, G., Frias, B. I., Caballero, M. Y., Mustelier, H. A., & Morales-Bueno, R. (2014). Fast adapting ensemble: A new algorithm for mining data streams with concept drift. *Scientific World Journal, 2015*.

Padmalatha, E., Reddy, C. R. K., & Rani, B. P. (2014). Classification of concept drift data streams. In *information science and applications (ICISA), 2014 international conference on* (pp. 1–5).

Polikar, R., Depasquale, J., Mohammed, H. S., Brown, G., & Kuncheva, L. I. (2010). Learn ++ .MF: A random subspace approach for the missing feature problem. *Pattern Recognition, 43*, 3817–3832.

Rutkowski, L., Pietruczuk, L., Duda, P., & Jaworski, M. (2013). Decision trees for mining data streams based on the mcDiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering, 25*, 1272–1279.

Shaker, A., & Hüllermeier, E. (2012). IBLSTreams: A system for instance-based classification and regression on data streams. *Evolving Systems, 3*, 235–249.

Shao, H., & Japkowicz, N. (2012). Applying least angle regression to ELM. In L. Kosseim, & D. Inkpen (Eds.), *Advances in artificial intelligence: th Canadian conference on artificial intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28–30, 2012. Proceedings* (pp. 170–180). Berlin, Heidelberg: Springer Berlin Heidelberg.

Shao, J., Ahmadi, Z., & Kramer, S. (2014). Prototype-based learning on concept-drifting data streams. In *proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 412–421).

Silva, D. N. G., Pacifico, L. D. S., & Ludermir, T. B. (2011). An evolutionary extreme learning machine based on group search optimization. In *Evolutionary computation (CEC), 2011 IEEE congress on* (pp. 574–580).

Singh, R., Kumar, H., & Singla, R. K. (2015). An intrusion detection system using network traffic profiling and online sequential extreme learning machine. *Expert Systems with Applications, 42*, 8609–8624.

Soares, S. G., & Rui, A. (2015a). An adaptive ensemble of on-line extreme learning machines with variable forgetting factor for dynamic system prediction. *Neurocomputing, 171*, 693–707.

Stosic, D., Stosic, D., & Ludermir, T. (2016). Voting based q-generalized extreme learning machine. *Neurocomputing, 174*, 1021–1030.
Sun, Y., Yuan, Y., & Wang, G. (2014). Extreme learning machine for classification over uncertain data. *Neurocomputing, 128*, 500–506.
Tavares, L. D., Saldanha, R. R., & Vieira, D. A. G. (2015). Extreme learning machine with parallel layer perceptrons. *Neurocomputing, 166*, 164–171.
Wang, G. G., Lu, M., Dong, Y. Q., & Zhao, X. J. (2015). Self-adaptive extreme learning machine. *Neural Computing & Applications, 27*, 1–13.
Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (0000). Characterizing Concept Drift. arXiv1511.03816.
Wu, X., Li, P., & Hu, X. (2012). Learning from concept drifting data streams with unlabeled data. *Neurocomputing, 92*, 145–155.
Xin, J., Wang, Z., Qu, L., & Wang, G. (2015). Elastic extreme learning machine for big data classification. *Neurocomputing, 149*, 464–471.
Xue, X., Yao, M., Wu, Z., & Yang, J. (2014). Genetic ensemble of extreme learning machine. *Neurocomputing, 129*, 175–184.
Yong, Z., & Meng, J. E. (2016). Sequential active learning using meta-cognitive extreme learning machine. *Neurocomputing, 173*, 835–844.
Youlu, X., Shen, F., Chaomin, L., & Zhao, J. (2015). I^3-SVM: A lifelong learning method for SVM. In *2015 international joint conference on neural networks (IJCNN)* (pp. 1–8).
Yu, H., Sun, C., Yang, W., Yang, X., & Zuo, X. (2015). AL-ELM: One uncertainty-based active learning algorithm using extreme learning machine. *Neurocomputing, 166*, 140–150.
Yu, L., Dai, W., & Tang, L. (2015). A novel decomposition ensemble model with extended extreme learning machine for crude oil price forecasting. *Engineering Applications of Artificial Intelligence, 47*.
Zeng, Y., Xu, X., Fang, Y., & Zhao, K. (2015). Traffic sign recognition using deep convolutional networks and Extreme learning machine. In X. He, X. Gao, Y. Zhang, Z.-H. Zhou, Z.-Y. Liu, B. Fu, F. Hu, & Z. Zhang (Eds.), *Intelligence science and big data engineering. Image and video data engineering: 5th international conference, IScIDE 2015, Suzhou, China, June 14–16, 2015* (pp. 272–280). Cham: Springer International Publishing. Revised Selected Papers, Part I
Zhang, P., & Yang, Z. (2015). Ensemble extreme learning machine based on a new self-adaptive adaboost. RT. In J. Cao, K. Mao, E. Cambria, Z. Man, & K.-A. Toh (Eds.), *Proceedings of ELM-2014 Volume 1: Algorithms and theories* (pp. 237–244). Cham: Springer International Publishing.
Zhang, P., Zhu, X., Shi, Y., & Wu, X. (2009). *An aggregate ensemble for mining concept drifting data streams with noise*. Springer Berlin Heidelberg.
Zhang, R., Lan, Y., Huang, G. B., & Xu, Z. B. (2013). Dynamic extreme learning machine and its approximation capability. *IEEE Transactions on Cybernetics, 43*, 2054–2065.
Zhang, X. (2013). *Matrix analysis and application, 2nd ed*. Tsinghua University Press.
Zong, W., Huang, G. B., & Chen, Y. (2013). Weighted extreme learning machine for imbalance learning. *Neurocomputing, 101*, 229–242.
Zuo, B., Huang, G. B., Wang, D., Han, W., & Westover, M. B. (2014). Sparse extreme learning machine for classification. *IEEE Transactions on Cybernetics, 44*, 1858–1870.