Han DH, Zhang X, Wang GR. Classifying uncertain and evolving data streams with distributed extreme learning machine. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(4): 874–887 July 2015. DOI 10.1007/s11390-015-1566-6

Classifying Uncertain and Evolving Data Streams with Distributed Extreme Learning Machine

Dong-Hong Han [1,2] (韩东红), *Member, CCF*, Xin Zhang [1] (张 昕)
and Guo-Ren Wang [1,2] (王国仁), *Senior Member, CCF*

[1] *College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*
[2] *Key Laboratory of Medical Image Computing (NEU), Ministry of Education, Shenyang 110819, China*

E-mail: handonghong@mail.neu.edu.cn; 185332896@qq.com; wangguoren@mail.neu.edu.cn

Received January 31, 2015; revised May 15, 2015.

Abstract Conventional classification algorithms are not well suited for the inherent uncertainty, potential concept drift, volume, and velocity of streaming data. Specialized algorithms are needed to obtain efficient and accurate classifiers for uncertain data streams. In this paper, we first introduce Distributed Extreme Learning Machine (DELM), an optimization of ELM for large matrix operations over large datasets. We then present Weighted Ensemble Classifier Based on Distributed ELM (WE-DELM), an online and one-pass algorithm for efficiently classifying uncertain streaming data with concept drift. A probability world model is built to transform uncertain streaming data into certain streaming data. Base classifiers are learned using DELM. The weights of the base classifiers are updated dynamically according to classification results. WE-DELM improves both the efficiency in learning the model and the accuracy in performing classification. Experimental results show that WE-DELM achieves better performance on different evaluation criteria, including efficiency, accuracy, and speedup.

Keywords uncertain data stream, classification, extreme learning machine, distributed computing, concept drift

1 Introduction

The development of the Internet, together with improved data collection techniques, has created overwhelming amounts of streaming data continuously being generated at high speeds, such as financial service data, web application data, network intrusion data, sensor data, and surveillance video streams[1]. Data streams are inherently associated with uncertainty in many real-world applications, such as sensor networks measuring temperature and light, radio frequency identification (RFID) networks, global positioning systems (GPS), and weather radar networks[2-4]. Traditional data mining algorithms that focus on certain and static data are not adequate for classifying uncertain streaming data.

Data uncertainty can be described by possible worlds model[5], in which every uncertain tuple is represented by several instances with the probabilities of the instances summing to 1. The uncertainty can mainly appear in attribute values, class labels, or instance level. In real-world applications, instance-level uncertainty is a common phenomenon. For example, in a mobile object monitoring system, it is impossible to track the exact location of the moving object O_1 at all times due to the limitation of resource. Instead, the location of O_1 at time t_i can be described by a few specific possible positions, or instances, and their associated probabilities in a region. It is assumed that the uncertain data stream consists of infinitely many tuples. Given that there are several instances per tuple, the task of classifying an overwhelming amount of data becomes even more difficult. In this study, the classification of data stream with instance-level uncertainty is explored.

The underlying data stream may evolve over time

Regular Paper
Special Section on Data Management and Data Mining
This work was supported by the National Natural Science Foundation of China under Grant Nos. 61173029 and 61272182.

and the streaming data distribution may change continuously, which is a phenomenon known as concept drift. It is essential for classifying uncertain data stream to detect concept drift and update the learning model in a timely way. Obviously, the approach to handling concept drift is different for precise and uncertain streaming data.

Due to uncertain data stream properties, the challenge of uncertain data stream classification is how to deal with uncertainty and concept drift in a timely way. To the best of our knowledge, there is no research so far on classifying uncertain streaming data based on distributed extreme learning machine (ELM). In this paper, we tackle the challenge here and investigate uncertain data stream classification. Our main contributions are:

• Based on the parallel ELM (PELM) algorithm[6], we first propose the distributed ELM (DELM) algorithm to handle large-scale datasets with a faster speed.

• We then extend DELM into the weighted ensemble DELM (WE-DELM) algorithm to classify uncertain streaming data in the presence of concept drift. In order to track evolving concepts, the weight of the base classifiers is updated dynamically according to the accuracy of the base classifiers. Meanwhile, the ensemble model is updated in the light of the weight of the base classifiers or classification error of the ensemble model.

• Because there is no real uncertain data stream set available, the dataset we used in the experiments is synthesized from real datasets. Experimental results show that the proposed algorithms have better performance on several evaluation criteria, including efficiency, accuracy, and speedup.

The remainder of the paper is organized as follows. In Section 2, we introduce background and related work. Section 3 reviews ELM and MapReduce. Our novel algorithms are presented in Section 4, with experimental results and evaluations discussed in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

Broadly, most research work addresses two kinds of uncertainty, tuple-level uncertainty and attribute value uncertainty[7]. If the independence assumption is used, the former means the probability whether the tuple exists or not. In the latter case, the individual attributes of a number of tuples are modeled by probability density functions (PDFs) or other statistical parameters such as standard deviation. In many real applications, one does not access to the PDFs. Hence, a discrete case of the PDF is generally used to describe uncertainty. An uncertain data x_i consists of m possible values as its instances, denoted by $x_i = \{(x_i^1, p_{x_i^1}), (x_i^2, p_{x_i^2}), \ldots, (x_i^m, p_{x_i^m})\}$, where x_i^j denotes the j-th instance of tuple x_i, and $p_{x_i^j}$ expresses the probability of the j-th instance of tuple x_i. The sum of the probabilities of the m instances of an uncertain tuple x_i is equal to 1.

We first briefly introduce previous work related to our study on two aspects: streaming data classification and uncertain streaming data classification. And then, we present a brief overview of ELM and MapReduce.

2.1 Related Work

2.1.1 Streaming Data Classification

Classification over data streams has recently attracted a lot of attention due to the increasing number of streaming data applications. Streaming data classification algorithms must be designed to deal with concept drift and an overwhelming amount of data[8]. There are two main types of classification approaches: single classifier and ensemble model[9].

Among the best known and most used cases of single classifiers, there are algorithms based on decision tree learning. Domingos and Hulten[10] proposed the very fast decision tree (VFDT) algorithm based on Hoeffding bounds, which requires only a single scan of the streaming data. Many algorithms that extend the VFDT algorithm have been put forward. The concept-adapting very fast decision tree (CVFDT) algorithm was proposed in order to handle concept drift[11]. The dynamic update of CVFDT is implemented by deleting unfeasible tree nodes and adding a new sub-tree structure, instead of constructing a new tree. The VFDTc system extends the VFDT algorithm in order to reduce the variance component for large-size datasets[12]. In [13], an ambiguous CVFDT (aCVFDT) algorithm is presented to integrate ambiguities into the CVFDT algorithm. On the other hand, decision rules are a classification model similar to decision tree, with the advantage of having individual rules that can be managed independently. There exist some algorithms for learning decision rules from streaming data[14-15]. Moreover, some approaches have studied the problem of data stream classification in supervised incremental learning. [16] presents a family of methods for monitoring over time the performance metrics measured during the learning

process, to trigger drift signals when a significant variation has been detected.

With the growing amount of streaming data, the single classifiers usually have difficulty in handling concept drift. To improve the accuracy, ensemble methods have been proposed. The ensemble model consists of several base classifiers, which are learned from training data chunks with classical algorithms, such as C4.5, Naïve Bayes. In general, every incremental ensemble approach adopts some criteria to dynamically update the base classifiers and the final classification result arises from a weighted vote among the base classifiers. The streaming ensemble algorithm (SEA) was proposed in [17]. The SEA algorithm builds base classifiers on sequential chunks of streaming data, combining them into a fixed-size ensemble classifier with a heuristic strategy.

Building upon the SEA algorithm, Stanley proposed the concept drift committee (CDC) algorithm[18], which uses a weighted committee of hypotheses that vote on the current classification. Wang *et al.* proposed the weighted ensemble classifier (WEC) algorithm[19] in which base classifiers are judiciously weighted based on their expected classification accuracy on the test data. In [20], the adaptive concept ensemble (ACE) system includes one online learner and a drift detection mechanism for quick response. Li *et al.* proposed an ensemble model of random decision tree algorithms to distinguish various types of concept drifts from noisy data streams[21]. In [22], a decision tree ensemble for large-scale data was proposed.

Ensemble learning algorithms have proven to be highly effective compared with single classifiers. Moreover, ensemble classifiers can be easier to learn with distributed techniques[19]. To classify a huge volume of uncertain streaming data efficiently and accurately, we adopt a learning model implemented on the Hadoop platform.

2.1.2 Uncertain Streaming Data Classification

The uncertainty of streaming data poses additional challenges for classification algorithms. So far, only a few publications are available. Liang *et al.* introduced the uncertainty-handling and conception-adapting very fast decision tree (UCVFDT) algorithm[23], which is based on the CVFDT[11] algorithm and the UDT[24] algorithm. In [25], Pan *et al.* proposed two types of algorithms, the static classifier ensemble (SCE) algorithm and the dynamic classifier ensemble (DCE) algorithm for classifying uncertain data streams. The authors extended the NS-PDT algorithm and then used this approach to learn base classifiers in the ensemble model[26]. ECluds, a framework for classifying uncertain and evolving data streams, was proposed in [9]. In this approach, the supervised k-means algorithm was adopted to build base classifiers. Liu *et al.* proposed an approach to one-class-based uncertain data steam learning[27]. Based on the proposed local kernel-density-based method, the authors constructed base classifiers and then built the ensemble model to cope with concept drift in the uncertain data stream environment. In [28], a weighed ensemble classifier based on ELM (WEC-ELM) algorithm was proposed, in which base classifier is trained by an ELM technique to efficiently classify uncertain streaming data with evolving concept.

In general, due to the characteristic of uncertain data streams, the algorithm should be designed to classify efficiently and accurately. The ensemble model is suitable for learning base classifiers with distributed techniques, namely, the base classifiers are built simultaneously. To the best of our knowledge, there is no literature on classifying uncertain streaming data with distributed techniques. In this paper, we adopt distributed ELM to learn base classifiers and construct an ensemble classifier on the Hadoop platform.

2.2 Review of ELM and MapReduce

The feedforward neural network may be one of the most popular neural networks. A feedforward neural network consists of one input layer receiving the stimuli from the external environment, one or more hidden layers, and one output layer sending the network output to the external environment[29]. Huang Babri showed that, for function approximation in a finite training set, a single-hidden layer feedforward neural network (SLFN), with at most $\tilde{N}$ hidden nodes and almost any nonlinear activation function, can exactly learn N distinct observations[30]. Unlike the conventional SLFN implementation in which all adjustments of parameters of the networks are iterative, an ELM algorithm was proposed[31]. As a generalization of SLFN, the essence of ELM is that, if hidden nodes, input weights, and hidden layer biases can be chosen randomly, the output weights can be computed analytically. The processing of the ELM is performed in one pass without iteration. Therefore, the ELM algorithm tends to provide good generalization performance at extremely fast learning speed.

Let N be the number of arbitrary distinct samples $(\boldsymbol{x}_i, \boldsymbol{t}_i)$, where $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})^{\mathrm{T}} \in \mathbb{R}^n$ and

$\boldsymbol{t}_i = (t_{i1}, t_{i2}, \ldots, t_{im})^{\mathrm{T}} \in \mathbb{R}^m$. The output $\boldsymbol{O}_j$ of an SLFN for input $\boldsymbol{x}_j$ is

$$\boldsymbol{O}_j = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g_i(\boldsymbol{x}_j) = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g(\boldsymbol{w}_i \boldsymbol{x}_j + \boldsymbol{b}_i),$$

where $\boldsymbol{w}_i = (w_{i1}, w_{i2}, \cdots, w_{in})^{\mathrm{T}}$ is the weight vector connecting the i-th hidden node and the input nodes, $\boldsymbol{\beta}_i = (\beta_{i1}, \beta_{i2}, \cdots, \beta_{im})^{\mathrm{T}}$ is the weight vector connecting the i-th hidden node and the output nodes, and $\boldsymbol{b}_i$ is the threshold of the i-th hidden node. The activation function $g(x)$ can be a Sigmoid function, Sine function or RBF function. SLFN can approximate these N samples with zero error means, in other words, there exist $\boldsymbol{\beta}_i$, $\boldsymbol{w}_i$, and $\boldsymbol{b}_i$ such that $\sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g_i(\boldsymbol{x}_j) = \boldsymbol{t}_j, j = 1, \ldots, N$.

The above N equations can be expressed compactly as $\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{T}$, where

$$\boldsymbol{H} = \begin{pmatrix} g(\boldsymbol{w}_1 \boldsymbol{x}_1 + \boldsymbol{b}_1) & \cdots & g(\boldsymbol{w}_{\tilde{N}} \boldsymbol{x}_1 + \boldsymbol{b}_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\boldsymbol{w}_1 \boldsymbol{x}_N + \boldsymbol{b}_1) & \cdots & g(\boldsymbol{w}_{\tilde{N}} \boldsymbol{x}_N + \boldsymbol{b}_{\tilde{N}}) \end{pmatrix}_{N \times \tilde{N}},$$

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{t}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{t}_N^{\mathrm{T}} \end{pmatrix}_{N \times m} \quad \text{and} \quad \boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\beta}_{\tilde{N}}^{\mathrm{T}} \end{pmatrix}_{\tilde{N} \times m}.$$

$\boldsymbol{H}$ is an independent variables matrix of N samples and $\boldsymbol{T}$ is a sample observation value matrix. $\boldsymbol{H}$ is called the hidden layer output matrix of the neural network. Both $\boldsymbol{H}$ and $\boldsymbol{T}$ are known. In order to calculate the output weight matrix $\boldsymbol{\beta}$, the smallest norm least-squares solution of the above linear system is $\hat{\boldsymbol{\beta}} = \boldsymbol{H}^{+}\boldsymbol{T}$, where $\boldsymbol{H}^{+}$ is the Moore-Penrose generalized inverse of matrix $\boldsymbol{H}$.

MapReduce is a programming model, proposed by Google, to process large datasets in parallel on large clusters[32]. It is the heart of the Hadoop platform, which consists of several relevant sub-projects and establishes a distributed system with high reliability and scalability. Relevant characteristics of the MapReduce model are automatic allocation of documents, load balancing, node failure processing, and data transfer between different nodes. Hence, users can easily run their programs without expertise in distributed programming.

Algorithm 1 gives the pseudo-code of the ELM algorithm.

Algorithm 1. ELM Algorithm

Require: training dataset: $N = \{(\boldsymbol{x}_i, \boldsymbol{t}_i) \mid \boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{t}_i \in \mathbb{R}^m, i = 1, 2, \cdots, N\}, \boldsymbol{x}_i = (x_{i1}, x_{i2}, \cdots, x_{in})^{\mathrm{T}} \in \mathbb{R}^n, \boldsymbol{t}_i = (t_{i1}, t_{i2}, \cdots, t_{im})^{\mathrm{T}} \in \mathbb{R}^m$;

activation function: $g(x)$;

hidden layer nodes: $\tilde{N}$

Ensure: output weight: $\boldsymbol{\beta}$

1: **for** $i = 1$ to $\tilde{N}$ **do**

2: Randomly assign input weight $\boldsymbol{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in})^{\mathrm{T}}$ and bias $\boldsymbol{b}_i, i = 1, 2, \cdots, \tilde{N}$;

3: **end for**

4: Calculate the hidden layer output matrix $\boldsymbol{H}$;

5: Calculate the output weight $\boldsymbol{\beta} = \boldsymbol{H}^{+}\boldsymbol{T}$, and $\boldsymbol{H}^{+}$ is Moore-Penrose generalized inverse of $\boldsymbol{H}$, $\boldsymbol{H} = (t_1, t_2, \ldots, t_N)^{\mathrm{T}}$;

The principle of MapReduce model is that $(key, value)$ pairs are computed from input $(key, value)$ pairs. A MapReduce program is composed of a $Map()$ procedure that performs filtering and sorting, and a $Reduce()$ procedure that performs a summary operation. The Map function and the $Reduce$ function are specified by users. Namely, a MapReduce program includes two steps.

In the first step, when the large-scale data is processed, data can be transformed into $(key, value)$ pairs and divided into different blocks for corresponding Map procedure. Every Map procedure is independent and runs simultaneously on the Hadoop platform. When input $(key, value)$ pairs are fed to a corresponding Map function, some intermediate pairs are generated. In the second step, after these intermediate $(key, value)$ pairs are merged, grouped, or sorted by key value, the $Reduce$ function combines them into output $(key, value)$ pairs as the final result. The structure of the MapReduce model is shown in Fig.1.

Both Map and $Reduce$ functions can run in parallel across a Hadoop cluster. That is to say, the MapReduce model provides sufficient high-level parallelization. To address the issue of how to classify large volumes of uncertain streaming data with fast speed, in this paper, we adopt the MapReduce model to learn base classifiers in parallel.

3 Proposed Approach

In this section, we will demonstrate the corresponding definitions and our algorithms.

3.1 Problem Definition

Suppose we have an uncertain data stream $D = \{D_1, D_2, \cdots, D_M, \cdots\}$, in which each D_i is called

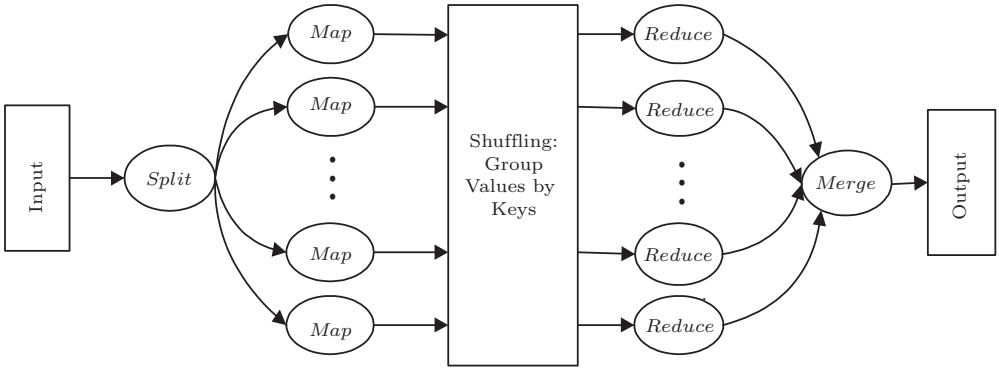


Fig.1. Structure of MapReduce model.

"chunk". The first M chunks in the beginning of the uncertain data stream consist of training data and the later chunks are filled with test data.

For clarity, Table 1 lists the notations used in this paper. In each chunk D_i, there exists the same amount of uncertain tuples. Namely, D_i is a set of uncertain tuples x_j, $D_i = \{(x_j, t_j)\}$, where $j = i+1, i+2, \cdots i+n$, and t_j is the arriving time of tuple x_j. The possible world instances set of x_j is $S_{x_j} = \{(x_j^k, p_{x_j^k}, c_j^k)\}(k = 1, 2, \cdots, m)$, where the k-th instance of uncertain tuple x_j is denoted by x_j^k. Each instance is a d-dimensional vector including d attributes. c_j^k is the class label of the instance x_j^k and $p_{x_j^k}$ is the probability of x_j^k. The sum of the probabilities of the instances belonging to uncertain tuple x_j is $\sum_{k=1}^{m} p_{x_j^k} = 1$.

Table 1. Notions and Meaning

Notion	Meaning
D	Uncertain data stream
D_i	The i-th uncertain data stream chunk
x_j	The j-th uncertain tuple in data chunk
M	Number of training data chunks
n	Number of tuples in each chunk
x_j^k	The k-th instance of uncertain tuples x_j
$p_{x_j^k}$	Probability of instance x_j^k
L	Number of class labels
S_{x_j}	Possible world instances set of uncertain tuple x_j
$P_{x_j}^{C_l}$	Classification probability of uncertain tuple x_j belonging to C_l
S_C	Set of class labels
E_i	The i-th base classifier
WE_i	Weight of the i-th base classifier

According to the model of the uncertain data stream noted above, the corresponding definitions used in this paper are as follows.

Definition 1 (Classification of Instance of Uncertain Tuple). *Let S_c be the set of class labels, where $S_c = \{C_1, C_2, \cdots, C_L\}$. After using M base classifiers to predict the k-th instance x_j^k of uncertain tuple x_j, C_{ji}^k denotes the class label of the instance x_j^k predicted by the i-th base classifier E_i. The final classification of instance x_j^k of uncertain tuple x_j is referred to as $c_j^{'k}$, which can be calculated by the votes of all base classifiers.*

Definition 2 (Classification Probability of Uncertain Tuple). *Let S_{x_j} be the set of possible world instances of x_j, where $S_{x_j} = \{(x_j^1, p_{x_j^1}, c_j^1), (x_j^2, p_{x_j^2}, c_j^2), \cdots, (x_j^m, p_{x_j^m}, c_j^m)\}$. Suppose that each instance of x_j is independent of each other. If the instance x_j^k belongs to C_l $(1 \leqslant l \leqslant L)$, the classification probability, which uncertain tuple x_j belongs to, can be calculated as:*

$$P_{x_j}^{C_l} = \sum_{x_j^k \in S_{x_j} \cap C_j^k = C_l} P_{x_j^k}. \tag{1}$$

Definition 3 (Classification of Uncertain Tuple). *Let S_c be the set of class labels. If $P_{x_j}^{C_l} > \forall P_{x_j}^{C_k} (1 \leqslant k \leqslant L)$, then the class label of uncertain tuple x_j is C_l.*

We can use (1) to calculate the real classification probability of uncertain tuple. Similarly, according to Definition 3, we can obtain the real class label of uncertain tuple.

Definition 4 (Classification Error Rate of the Base Classifiers). *For a test data chunk with n tuples, let E_err_i be the classification error rate of the i-th base classifier, which can be calculated as:*

$$E\_err_i = \frac{\sum_{j=1}^{n} err_{ji}}{n}, \tag{2}$$

where err_{ji} denotes whether the tuple x_j is wrongly classified by E_i. Namely, if x_j is wrongly classified, $err_{ji} = 1$; otherwise, $err_{ji} = 0$.

Definition 5 (Classification Error Rate of the Ensemble Model). *After classifying the test chunk D_{M+1}, the classification error rate of an ensemble model with M base classifiers can be calculated as:*

$$E\_err = \frac{\sum_{i=1}^{M} E\_err_i}{M}. \tag{3}$$

Definition 6 (Mean Square Error (MSE) of the Base Classifier). *Let MSE_i be mean square error of the i-th base classifier E_i, which can be calculated as:*

$$MSE_i = \frac{1}{n}\sum_{j=1}^{n}(E\_err_i)^2. \tag{4}$$

Definition 7 (Mean Square Error of a Base Classifier with Random Prediction). *According to Wang et al.*[19]*, on data chunk D_{M+1}, the probability that tuple x_j is classified as class label C_l is equal to the distribution of the class label C_l. We can calculate the mean square error of a base classifier with random prediction as:*

$$MSE_r = \sum_{c} p(c)(1-p(c))^2, \tag{5}$$

where $p(c)$ denotes the proportion of each class in the chunk D_{M+1} and $p(c) \in (0,1)$.

Definition 8 (Weight of the Base Classifier). *When every tuple in data chunk D_{M+1} is predicted by all base classifiers in the ensemble model, we can set weight for the i-th base classifier as:*

$$WE_i = MSE_r - MSE_i. \tag{6}$$

In our method, the first M training data chunks are inputted in base classifiers accordingly at the same time. Each base classifier is learned with the ELM technique, in which every instance of an uncertain tuple is classified and then the final class label can be predicted. Next, the distributed ELM algorithm proposed in this paper is discussed in detail.

3.2 DELM Algorithm

As mentioned earlier, in the ELM algorithm, the hidden layer output matrix $\boldsymbol{H}$, which is $N \times \tilde{N}$, is computed first. And then, the matrix $\boldsymbol{H}^+$, which is the Moore-Penrose generalized inverse of $\boldsymbol{H}$, is calculated. The computational process of the matrix $\boldsymbol{H}^+$ includes a series of matrix operations as follows: $\boldsymbol{H}^+ = (\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1} \cdot \boldsymbol{H}^{\mathrm{T}}$ or $\boldsymbol{H}^+ = \boldsymbol{H}^{\mathrm{T}} \cdot (\boldsymbol{H} \cdot \boldsymbol{H}^{\mathrm{T}})^{-1}$. As $N \gg \tilde{N}$, the matrix $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$ is far smaller than the matrix $\boldsymbol{H} \cdot \boldsymbol{H}^{\mathrm{T}}$. The previous formula is adopted because the computational cost is much less than the latter. Moreover, owing to $\boldsymbol{\beta} = (\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1} \cdot \boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$, the matrix $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ needs to be computed. The matrix $(\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1}$ or the product of $(\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1}$ and $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ is small and does not need to be calculated on the MapReduce framework.

He *et al.*[6] proposed a high effective PELM algorithm based on the MapReduce framework. As analyzed above, the two parts of the matrix operation are completed parallelly to compute $\boldsymbol{\beta}$ in the PELM algorithm: $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$. The former is the product of two matrices of $\tilde{N} \times N$ and $N \times \tilde{N}$. The latter is the product of two matrices of $\tilde{N} \times N$ and $N \times m$. Moreover, $(\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1}$ is the matrix of $\tilde{N} \times \tilde{N}$ and $(\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1} \cdot \boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ is the matrix of $\tilde{N} \times m$. Therefore, if the matrices $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ are calculated, the computational cost for computing $\boldsymbol{\beta}$ is very low.

The shortcoming of the PELM algorithm is that the amount of data transmission increases during the Shuffle stage. Specifically, the inner product of the rows of the left matrix and the columns of the right matrix are used to compute the product of two matrices. For example, while computing $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$, the number of *Reduce* job is $\tilde{N} \times \tilde{N}$ in an ideal state. Every *Reduce* job computes each element in the result matrix, namely, elements in the i-th row of the matrix $\boldsymbol{H}^{\mathrm{T}}$ and the j-th column of the matrix $\boldsymbol{H}$ are computed in every *Reduce* job, where $i = 1, 2, \cdots, \tilde{N}$ and $j = 1, 2, \cdots, \tilde{N}$. Thus, the Shuffle stage needs to propagate $\tilde{N} \times \tilde{N} \times N$ elements of the matrix $\boldsymbol{H}^{\mathrm{T}}$ and $\tilde{N} \times N \times \tilde{N}$ elements of the matrix $\boldsymbol{H}$ to the working nodes possessing the *Reduce* job. When N is large, the volume of the intermediate data is very large, which cannot be optimized by the Hadoop framework. In extreme moments, the scale of the input matrix is too large to store a row of the left matrix or a column of the right matrix in the memory of the *Reduce* nodes. This is the bottleneck issue which cannot be well solved yet in the PELM algorithm.

In this paper, based on the PELM algorithm, we propose the DELM algorithm in which a new partition method is raised to effectively induce the intermediate data. That is to say, instead of partitioning the matrix with rows and columns, blocks consisting of several rows or columns are adopted to divide the left and the right matrix.

Given $\boldsymbol{A} = \boldsymbol{H}^{\mathrm{T}} = (a_{ij})_{(\tilde{N} \times N)}$ and $\boldsymbol{B} = \boldsymbol{H} = (b_{ji})_{(N \times \tilde{N})}$, as $\tilde{N}$ is usually smaller than N, we just partition according to column and row with respect to the matrixes $\boldsymbol{A}$ and $\boldsymbol{B}$. There exist S blocks in which the size of the column is S_b.

For matrix $\boldsymbol{A}$, the submatrix is denoted as: $\boldsymbol{A}_t = (a_{ij})_{(\tilde{N}\times S_b)}, t \in \{1, 2, \cdots, S\}$, where $S = \left[\frac{N-1}{S_b}\right] + 1$. The number of the column in the last submatrix $\boldsymbol{A}_s$ is possible less than S_b.

For matrix $\boldsymbol{B}$, the submatrix is denoted as: $\boldsymbol{B}_t = (b_{ji})_{(S_b\times\tilde{N})}, t \in \{1, 2, \cdots, S\}$, where $S = \left[\frac{N-1}{S_b}\right] + 1$. As noted above, the number of the column in the last submatrix $\boldsymbol{B}_s$ is possible less than S_b.

The matrixes $\boldsymbol{A}$ and $\boldsymbol{B}$ are defined respectively as:

$$\boldsymbol{A} = (\boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_S), \quad \text{and} \quad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \\ \vdots \\ \boldsymbol{B}_s \end{pmatrix}. \tag{7}$$

Hence, the product of the two matrices is $\boldsymbol{C} = \boldsymbol{AB} = (\sum_{t=1}^{S} \boldsymbol{A}_t \boldsymbol{B}_t)_{\tilde{N}\times\tilde{N}}$.

As shown in (7), the product of every submatrix pair needs a *Reduce* work node. Namely, the block of $\boldsymbol{A}$ and the corresponding block of $\boldsymbol{B}$ are transferred to the same *Reduce* node. The most important advantage of the DELM algorithm is that it has better scalability. Meanwhile, the amount of the intermediate transmission data is far less than the PELM algorithm when the matrix is very large. Theoretically, although the matrix can be infinitely large, the product of two matrices is completed on clusters by adjusting the block size of the left and right matrices.

Because of calculating $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$, the process consists of two MapReduce stages. The first MapReduce stage partitions the input matrices $\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{A}$ and $\boldsymbol{H} = \boldsymbol{B}$ respectively and calculates the multiplication of the submatrices. The second one combines the intermediate result and calculates the final result. Specifically, this task is completed by two jobs. During job 1, in Map function, the matrix $\boldsymbol{A}$ is divided and denoted by $(key, value)$ pairs, where $key = \alpha$ and $\alpha = 1, 2, \cdots, S$; $value = (\text{“a”}, \boldsymbol{A}_\alpha)$. The process of the matrix $\boldsymbol{B}$ is similar to that of $\boldsymbol{A}$. In *Reduce* function, the product of the two submatrices is calculated, which is $\boldsymbol{C}_\alpha = \boldsymbol{A}_\alpha \cdot \boldsymbol{B}_\alpha$, and then the result $(key, value)$ pairs are stored in the Hadoop distributed file system (HDFS). During job 2, in Map function, the intermediate result matrix $\boldsymbol{C}_\alpha$ is accessed from the Hadoop and the key in $(key, value)$ pairs is set as the same value. In *Reduce* function, the sum of all $\boldsymbol{C}_\alpha$ is calculated and the final result is obtained.

Calculating the matrix $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ is similar to the process above. Algorithm 2 gives the pseudo-code of $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ and $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$. The input of Algorithm 2 is the left of matrix $\boldsymbol{A}$ and the right of matrix $\boldsymbol{B}$. The output is the result of matrix $\boldsymbol{C}'$, which is the product of $\boldsymbol{A}$ and $\boldsymbol{B}$. The algorithm is composed of two jobs. Lines 1∼6 are the Map stage in job 1, in which the matrixes $\boldsymbol{A}$ and $\boldsymbol{B}$ are partitioned into S blocks. And then, $(key, value)$ pairs are formalized, where key is the order number of the blocks and the value is the content of the corresponding blocks in $\boldsymbol{A}$ and $\boldsymbol{B}$. Lines 7∼10 are the *Reduce* stage in job 1, in which $\boldsymbol{C}_\alpha = \boldsymbol{A}_\alpha \cdot \boldsymbol{B}_\alpha$ is calculated. Lines 11∼13 are the Map stage in job 2, in which the same key value is set in $(key, value)$ pairs. Lines 14∼18 are the *Reduce* stage in job 2. All intermediate results of the submatrices are combined and summed. Finally, the final results of matrix $\boldsymbol{C}'$ are the output.

Algorithm 2. Parallel Block Matrix Multiplication Algorithm
Require: left matrix $\boldsymbol{A}$ and right matrix $\boldsymbol{B}$, number of block S, the size of column S_b in one block
Ensure: results matrix $\boldsymbol{C}'$

MapOne$(key, value)$
1: Divide matrix $\boldsymbol{A}$ into S submatrices;
2: Divide matrix $\boldsymbol{B}$ into S submatrices;
3: **for** $i = 1$ to S **do**
4: *emit*$(i,$ (“a”, $\boldsymbol{A}_i))$;
5: *emit*$(i,$ (“b”, $\boldsymbol{B}_i))$;
6: **end for**
ReduceOne$(key, value)$
7: **for** $i = 1$ to S **do**
8: Calculate $\boldsymbol{C}_i = \boldsymbol{A}_i \cdot \boldsymbol{B}_i$;
9: **end for**
10: *emit*$(i,$ (“c”, $\boldsymbol{C}_i))$;
MapTwo$(key, value)$
11: **for** $i = 1$ to S **do**
12: *emit*(samekey, (“c”, $\boldsymbol{C}_i)$);
13: **end for**
ReduceTwo$(key, value)$
14: Set $\boldsymbol{C}'$ as an empty matrix;
15: **for** $i = 1$ to S **do**
16: $\boldsymbol{C}' = \boldsymbol{C}' + \boldsymbol{C}_i$;
17: **end for**
18: Output results matrix $\boldsymbol{C}'$;

Algorithm 3 gives the pseudo-code of the DELM algorithm based on Algorithm 2. Lines 1∼4 establish the hidden layer output matrix $\boldsymbol{H}$. Line 5 calculates the matrix multiplication $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{T}$ and $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$ with parallel block matrix multiplication algorithm. Line 6 calculates the matrix $\boldsymbol{H}^{+}$, which is the Moore-Penrose generalized inverse of $\boldsymbol{H}$. And then, line 7 calculates the output weight $\boldsymbol{\beta}$.

Algorithm 3. Distributed Extreme Learning Machine (DELM) Algorithm
Require: training dataset: $N = \{(\boldsymbol{x}_i, \boldsymbol{t}_i) \mid \boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{t}_i \in \mathbb{R}^m, i = 1, 2, \cdots, N\}, \boldsymbol{x}_i = (x_{i1}, x_{i2}, \cdots, x_{in})^{\mathrm{T}} \in \mathbb{R}^n, \boldsymbol{t}_i = (t_{i1}, t_{i2}, \cdots, t_{im})^{\mathrm{T}} \in \mathbb{R}^m$; activation function: $g(x)$; hidden layer nodes: $\tilde{N}$
Ensure: output weight: $\boldsymbol{\beta}$

1: **for** $i = 1$ to $\tilde{N}$ **do**
2: Randomly assign input weight $\boldsymbol{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in})^{\mathrm{T}}$ and bias $\boldsymbol{b}_i, i = 1, 2, \cdots, \tilde{N}$;
3: **end for**
4: Calculate the hidden layer output matrix $\boldsymbol{H}$;
5: Calculate the matrix multiplication $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H}$ with Algorithm 2;
6: Calculate $\boldsymbol{H}^{+} = (\boldsymbol{H}^{\mathrm{T}} \cdot \boldsymbol{H})^{-1} \cdot \boldsymbol{H}^{\mathrm{T}}$, and $\boldsymbol{H}^{+}$ is the Moore-Penrose generalized inverse of $\boldsymbol{H}$;
7: Calculate the output weight $\boldsymbol{\beta} = \boldsymbol{H}^{+}\boldsymbol{T}$, $\boldsymbol{T} = (t_1, t_2, \ldots, t_N)^{\mathrm{T}}$;

3.3 WE-DELM Algorithm

In the context of the underlying uncertain data generating mechanism, the concept that we try to learn from the data, is constantly evolving. As one of the most important tasks of data mining, classifying uncertain streaming data is facing challenges such as dealing with uncertainty and concept drift effectively with distributed techniques. As we know, the ensemble classifier is a model to detect concept drift with high performance. Meanwhile, it is suitable for adopting distributed algorithms. Moreover, as the analysis above mentioned, the ELM algorithm, which can produce good generalization performance and has a fast learning speed, is feasible to train base classifiers. To the best of our knowledge, there is no literature of classifying uncertain streaming data with the parallel ensemble model based on ELM-MapReduce. In this paper, we propose the WE-DELM algorithm, in which distributed ELM is adopted, to efficiently address the problem of classifying large volume uncertain streaming data. Simultaneously, the WE-DELM algorithm uses weighted base classifiers and dynamically adjusts the weights of base classifiers to detect concept drifts.

The WE-DELM algorithm is composed of two steps. In the first step, for the beginning of the M training data chunks, M base classifiers in ensemble model are learned simultaneously with the DELM algorithm mentioned before. Meanwhile, the weight in each base classifier is initialized and adjusted. In the second step, the ensemble model is updated in light of the weight of each base classifier or the classification error rate of the ensemble model to adapt the evolving uncertain data stream. The WE model is shown in Fig.2.

3.3.1 Learning Base Classifiers and Setting the Weights

Concretely, an uncertain data stream D is divided into chunks $D_1, D_2, D_3, \cdots, D_M, \cdots$ and each chunk has n tuples. In the beginning of the uncertain data stream, there exist M training chunks. The M base classifiers are learned by the M training chunks with the DELM algorithm at the same time. And then, the weight of each base classifier is initialized as 1.

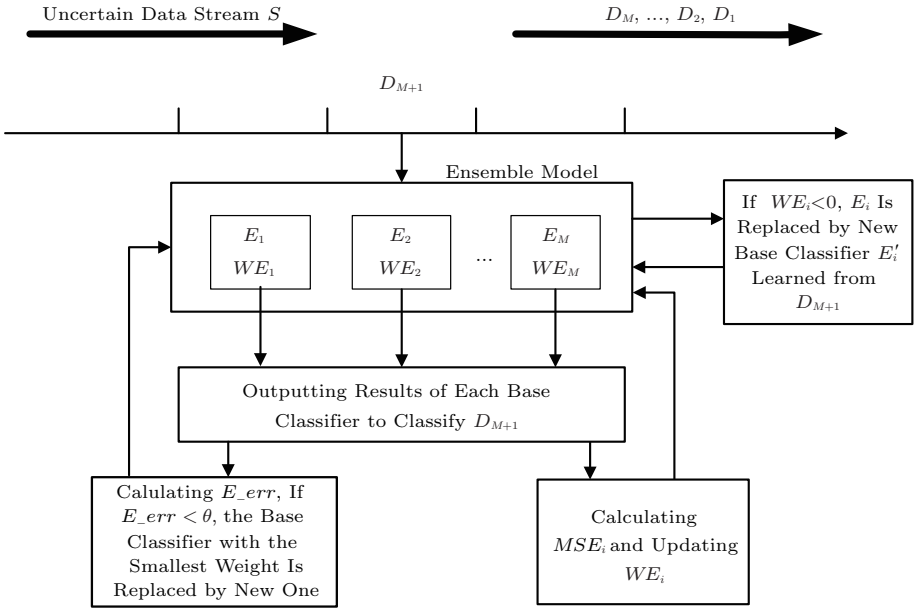


Fig.2. WE model.

While the test chunk D_{M+1} comes, the first step is that each tuple is classified by all base classifiers in the ensemble model. For every tuple x_j in test chunk D_{M+1}, its class label is gained as follows: when the k-th instance of x_j^k is inputted to the ensemble model, it can be classified by M base classifiers. The class labels of all instances of the tuple x_j are calculated by Definition 1. And then, the class label of the uncertain tuple x_j is obtained according to Definition 2 and Definition 3. Finally, all class labels of uncertain tuples in the test chunk D_{M+1} are gained.

Secondly, for the test chunk D_{M+1}, we calculate the classification error of each base classifier E_i for predicting tuple x_j, which is referred to as err_{ji}. err_{ji} denotes whether the tuple x_j is wrongly classified. Namely, if x_j is wrongly classified, $err_{ji} = 1$; otherwise, $err_{ji} = 0$. Afterwards, E_err_i, which is the classification error rate of the base classifier E_i on the test chunk, is calculated with (2).

Lastly, in light of the classification error rate of all base classifiers on test chunk D_{M+1} calculated above, we can compute the weight of all base classifiers. Namely, the smaller the E_err_i is, the more important role the E_i has. The weight WE_i of the i-th base classifier is adjusted as follows: we calculate MSE_i and MSE_r with (4) and (5) respectively. And then, the weight WE_i of the i-th base classifier is calculated with (6). The base classifier which has lower classification error rate is set as a higher weight.

3.3.2 Updating Base Classifiers

Due to characteristics such as large volume and time-changing generation, the underlying uncertain data stream may evolve for a long time. To detect concept drift and adopt the evolving uncertain data stream, in this paper, two methods are adapted.

Firstly, the base classifier which is not available to the present concept is deleted. That is, when $WE_i \leqslant 0$, the performance of the base classifier E_i is worse than the classifier randomly predicted. In this setting, the base classifier is dropped. The new base classifier E_i' is learned from the test chunk D_{M+1} and the weight is set for it. Secondly, if $WE_i > 0$ and the classification error rate E_err of ensemble model is bigger than threshold θ, it shows that the ensemble model is not suitable for the present concept of the evolving uncertain data stream. We need to drop the base classifier whose weight is the smallest in the ensemble model. After that, a new base classifier is trained on the up-to-date test chunk to adopt the changing uncertain data stream. Algorithm 4 gives the pseudo-code of the WE-DELM method.

Algorithm 4. Weighted Ensemble Classifier Based on Distributed Extreme Learning Machine (WE-DELM) Algorithm

Require:

D: uncertain data stream;

$E = \{E_1, E_2, \cdots, E_M\}$: ensemble classifier;

M: the number of the base classifiers;

n: the number of the tuples in each chunk;

θ: the threshold of error rate of ensemble classifier

Ensure: the classification results of test data

1: Training parallel M base classifiers on the first M training chunks $D_1, D_2, \cdots, D_M$ with DELM;
2: Set initial weights for each base classifier as 1;
3: **repeat**
4: Receive the up-to-date test chunk D_{M+1};
5: **for** $j = 1$ to n **do**
6: **for** $k = 1$ to m **do**
7: **for** $i = 1$ to M **do**
8: Classify x_j^k with E_i;
9: **end for**
10: **end for**
11: Calculate the predicted class label of the tuple x_j;
12: **end for**
13: **for** $i = 1$ to M **do**
14: Calculate the classification error rate E_err_i of the base classifier E_i;
15: Calculate MSE_i with (4) and MSE_r with (5);
16: Update the weight WE_i of E_i with (6);
17: **if** $(WE_i \leqslant 0)$ **then**
18: Drop the base classifier E_i;
19: Train new base classifier E_i' and set weight of E_i with the cross-validation method;
20: **end if**
21: **if** $(E\_err < \theta)$ **then**
22: Goto 27;
23: **end if**
24: Drop the base classifier E_i with the smallest WE_i;
25: Train new base classifier E_i' and set weight WE_i' with the cross-validation method;
26: **end for**
27: **until** uncertain data stream end;

Lines 1∼2 initialize the ensemble model. That is, based on the beginning of the M training data chunks, M base classifiers are trained simultaneously on the Hadoop platform with the DELM algorithm. Meanwhile, the weight of each base classifier is set as 1. After building the ensemble model, it can be used to classify the up-to-date test chunk D_{M+1} having n tuples. Specially, for every uncertain tuple x_j with m instances in the test chunk, each instance is classified by M base

classifiers respectively and then the predicted class label of the uncertain tuple x_j is obtained according to Definition 3 (lines 5∼12). Then we calculate the classification error rate of all M base classifiers. Based on this, MSE_i can be calculated and then the weight WE_i of the base classifiers can be changed by (6) (lines 13∼16). Lines 17∼25 show the process in which we update the ensemble model to track the time-changing uncertain data stream. On the one hand, if $WE_i \leqslant 0$, it denotes that the performance of the base E_i is worse than that predicted randomly. It is necessary to drop the base E_i and build a new base classifier E_i' on test chunk D_{M+1}. The weight of E_i' can be set with the cross-validation method. On the other hand, in the light of the class results predicted by the ensemble model, we can detect the concept drift. E_err, which is the classification error rate of the ensemble model, is calculated with (3). If E_err is smaller than threshold θ, it means that there is not concept drift. The program goes directly to line 27 to check whether the uncertain data stream ends. Otherwise, if E_err is bigger than threshold θ, we consider that the concept drift arises. In this situation, the ensemble model needs to be updated, namely, the base classifier with the smallest weight is dropped. Meanwhile, a new base classifier E_i' is learned from the test chunk D_{M+1}, whose weight is set with the cross-validation method.

4 Experiments

In this section, the performance of the proposed WE-DELM learning algorithm is demonstrated and compared with popular algorithms like the conventional back propagation (BP) algorithm and support vector machines (SVMs) algorithm. The experiment under the distributed environment is carried out in clusters with eight nodes, in which a node is used as the main node and seven nodes are used as data nodes. The hadoop cluster adopts the version 0.20.2 and its OS adopts Linux 5.6. Each node has an Intel® Core™ 2 Quad Q8400 2.66 GHz CPU and 4 GB main memory. The single machine experiment is run in an environment with an Intel® Core™ i5-2450M @2.50 GHz CPU and 4 GB main memory.

4.1 Dataset

In real applications, there is no uncertain dataset that is available for the data model proposed in this paper. In our experiment, the dataset is generated by artificially added probability in a real dataset or synthetic dataset to simulate the uncertain streaming data model. We will introduce different datasets in detail as follows.

In this paper, we adopt real-world dataset KDDcup99, and the popular static datasets Iris and Spambase to demonstrate the performance of our method.

• *KDDcup99.* The KDDcup99 dataset is built to estimate the performance of the intrusion detection system. In this dataset, there are 42 dimensions in each instance, which has 39 class labels. Different class labels represent different attack types. The original dataset consists of almost 5 000 000 instances.

• *Iris.* The Iris dataset, which is referred to as the Iris flower dataset with 150 instances, is a static dataset used for classifying. We can predict which class labels the Iris flower belongs to in light of four attributes like sepal length, sepal width, petal length, and petal width.

• *Spambase.* The Spambase data is used to detect whether an email is spam. The Spambase dataset contains two class labels, 57 dimensions and 4 601 instances.

4.2 Experimental Analysis

Firstly, we set parameters on different algorithms, such as BP, SVM, ELM, PELM and WE-DELM, as shown in Table 2.

Table 2. Parameters of Algorithms

	Number of Hidden Layer Nodes	Activation Function	Number of Tuples in a Chunk
BP	300	–	300
SVM	–	–	–
ELM	300	–	–
PELM	300	–	–
WE-DELM	300	–	10 000

4.2.1 Comparison of Accuracy Rate and Efficiency

Because the size of the Iris dataset is very small, we duplicate the dataset by 40 000 copies so that the dataset contains six million instances. And then, we compare test time and accuracy rate of the five algorithms on the Iris dataset copied as above. The results are shown in Table 3.

In Table 3, for the Iris dataset with six million instances, we show that the test time of the BP or SVM algorithm is very long, and the test time of the traditional centralized ELM algorithm is more than one

hour. However, the PELM or WE-DELM algorithm just needs a few minutes to test all data, which has higher efficiency. Because the dimension of the Iris dataset is lower and the total amount of the data is not great, the intermediate data quantity is not very large. This is the reason why PELM has better parallel performance.

Table 3. Test Time and Accuracy Rate on the Iris Dataset

	Test Time (s)	Accuracy Rate (%)
BP	–	95.26
SVM	–	97.35
ELM	4 653	96.92
PELM	487	97.35
WE-DELM	502	96.85

In the same manner, we duplicate the Spambase dataset of 4 000 copies so that the dataset has 18.4 million instances. The comparison results of accuracy rate and test time are given in Table 4.

Table 4. Test Time and Accuracy Rate on the Spambase Dataset

	Test Time (s)	Accuracy Rate (%)
BP	–	92.31
SVM	–	91.33
ELM	–	92.42
PELM	2 898	93.15
WE-DELM	1 292	92.45

Table 4 shows that the test time of BP, SVM and the traditional centralized ELM algorithm is very long for the Spambase dataset with 18.4 million instances. However, the test time of PELM and WE-DELM algorithm is very short. The intermediate data quantity is very large as the number of the dimension is 57. As a result, WE-DELM has better parallel performance than the PELM algorithm.

In Table 3 and Table 4, the results show that the accuracy rate between these algorithms is close and there is not much difference. In the setting of large amount of uncertain streaming data with high dimension, the accuracy rate of DELM algorithm, which is proposed in this paper, is close to that of other algorithms. Meanwhile, the efficiency of the WE-DELM algorithm is much better than that of other algorithms.

4.2.2 Comparison of Speedup

In this subsection, speedup is adopted to measure the combination property of a distributed algorithm. Speedup is the ratio of the running time of a task running on a single processor system to that on a parallel processor system. It can be expressed by: $Speedup(m) = \frac{T_1}{T_m}$, where T_1 is the time that the distributed algorithm runs on one node and T_m is the time that the distributed algorithm runs on m nodes. We duplicate the Spambase dataset as different copies according to the requirement of this experiment. The result of speedup between the DELM and the PELM algorithm is given in Fig.3.

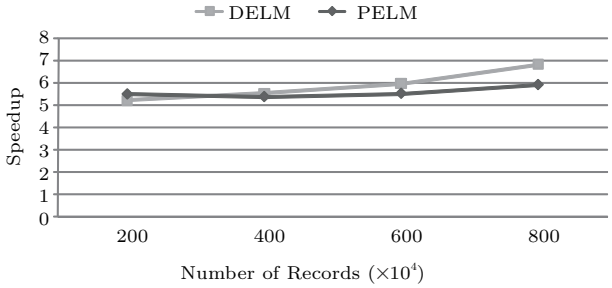


Fig.3. Speedup experiment on the Spambase dataset.

In Fig.3, it shows that the speedup of the DELM algorithm is better than that of the PELM algorithm with increasing data volume. The reason is that DELM algorithm based on matrix partition can reduce the amount of data transmission during the Shuffle stage. Hence the DELM algorithm has better efficiency and scalability. In theory, the DELM algorithm can be used for training data with infinite size.

The test time of two algorithms with the different numbers of examples is shown in Fig.4. By increasing the dataset size, we can see that the test time of the DELM algorithm is less than that of the PELM algorithm. Moreover, due to the increase of the scale of the matrix, the difference of test time becomes bigger and bigger. The main reason is that DELM is accomplished on a partition matrix. A better partition method can greatly reduce the intermediate data in the Shuffle stage and maintain high concurrency granularity. Therefore, DELM proposed in this paper has better efficiency.

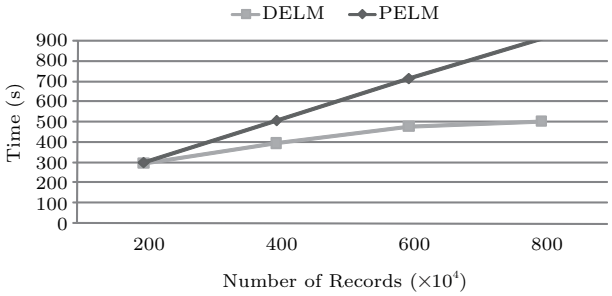


Fig.4. Test time experiment on the Spambase dataset.

4.2.3 Influence of the Matrix Partition Strategy in DELM

In this subsection, we study the impact of the matrix partition strategy in DELM on efficiency. We duplicate 1 000 copies of the Spambase dataset so that the dataset contains 4.6 million instances. Seven partition methods are adopted, in which every block is composed of 1 000, 5 000, 10 000, 20 000, 50 000, 75 000 and 100 000 tuples respectively. The result is shown in Fig.5.

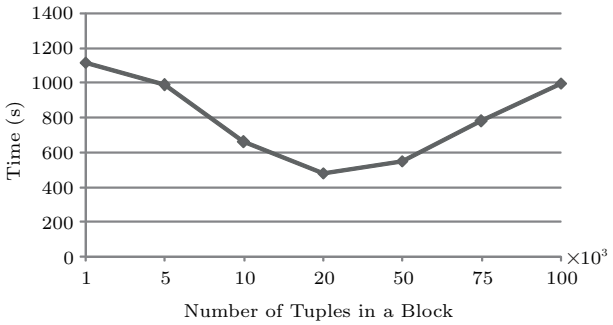


Fig.5. Relation between the partition strategy and the running time.

From Fig.5, when the size of the block is small, the analysis result shows that running time is very long. By increasing the block size, running time is gradually shorter. While each block contains 20 000 records, the running time is the shortest. And then, the running time increases. If the block size is small, the amount of the intermediate transmission data is large. As the block size increases, efficiency is better and better owing to the decrease of the intermediate transmission data. But the trend is not absolute. When the block size gets to a certain extent, the performance of the algorithm will degrade seriously. The smaller the amount of the intermediate transmission data is, the better performance the algorithm has. Hence, it is necessary to control the amount of intermediate transmission data in the Shuffle stage.

On the other hand, when the size of the block is larger, the amount of calculation increases and the concurrency of the algorithm reduces. Moreover, it leads to longer running time for the job because the amount of calculation of single node is too much. Under this situation, there exists the risk that the job cannot run.

4.2.4 Comparison of WE-DELM, DCE and UCVFDT Under Uncertain Data Stream Environment

In this subsection, we demonstrate the effectiveness of WE-DELM on the KDDcup99 dataset.

For UCVFDT, we set the parameters as follows: $\delta = 0.000\,1, f = 20\,000, \zeta = 0.000\,5, w = 100\,000, n_{\min} = 300$. For the DCE algorithm, the parameters are set as: $EnsembleSize = 25, K_{nei} = 5, ChunkSize = 5\,000$. For WE-DELM, we set the parameters as: $M = 5, n = 5\,000, \theta = 70\%$. The experimental results of the three algorithms about accuracy rate and training time are shown in Fig.6 and Fig.7 respectively.

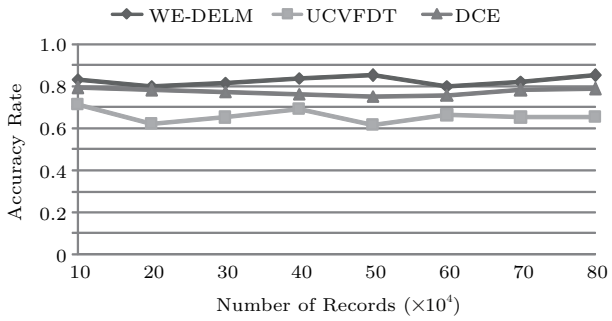


Fig.6. Accuracy rate.

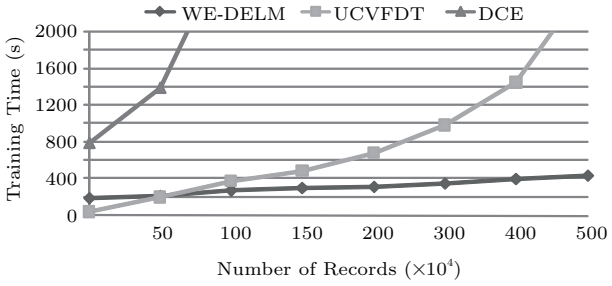


Fig.7. Training time.

From Fig.6, we can see that the difference of accuracy rate between WE-DELM and DCE is very small. However, the accuracy rate of the UCVFDT algorithm is especially low as the ensemble model has better performance than the single classifier model in accuracy rate for adopting concept drift.

From Fig.7, it is clear that the fluctuation of running time of UCVFDT or DCE is big with increasing the amount of data. Especially to DCE, when the number of the tuples in the dataset is more than one million, the training time is so long. The reason is that DCE belongs to the ensemble model and the training time of each base classifier is not fast. Thus, DCE cannot deal with an infinite data stream. UCVFDT has high efficiency for small size datasets, even better than WE-DELM. One cause is that the WE-DELM algorithm learned with distributed method cannot reduce the training time for small datasets. Another cause is

that data transmission and activating task add extra time. Overall, with the increased amount of data, the efficiency of UCVFDT degrades clearly. The training time of WE-DELM proposed in this paper slowly increases. In light of this, WE-DELM is more suitable for an infinite uncertain data stream with high speed.

5 Conclusions

The ELM algorithm has recently attracted more attention and it has been used successfully for a wide variety of classification problems. In the meantime, the need to analyze increasingly large (often streaming) data has renewed interest in parallel computing technologies. Uncertain streaming data, an important type of big data, is particularly challenging in this context. In order to address the challenges like uncertainty and concept drift with high efficiency in this paper, we proposed the DELM algorithm used for large matrix operation to handle big dataset on the Hadoop platform. And then, we proposed the WE-DELM algorithm, in which each base classifier in the ensemble model is trained in parallel with DELM, to classify an uncertain data stream with concept drift. Experiments on both synthetic and real-world datasets were conducted to validate our proposed methods. Our experimental results show that the proposed algorithms have better performance in terms of efficiency, accuracy and speedup.

Acknowledgement The authors would like to thank anonymous reviewers and editors for their valuable comments.

References

[1] Babcock B, Babu S, Datar M *et al.* Models and issues in data stream systems. In *Proc. the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, June 2002, pp.1-16.

[2] Tran T T, Peng L, Li B *et al.* PODS: A new model and processing algorithms for uncertain data streams. In *Proc. the 2010 ACM SIGMOD International Conference on Management of Data*, June 2010, pp.159-170.

[3] Cao K Y, Wang G R, Han D H *et al.* Continuous outlier monitoring on uncertain data streams. *Journal of Computer Science and Technology*, 2014, 29(3): 436-448.

[4] Zhao L, Yang Y Y, Zhou X. Continuous probabilistic subspace skyline query processing using grid projections. *Journal of Computer Science and Technology*, 2014, 29(2): 332-344.

[5] Zhou A Y, Jin C Q, Wang G R *et al.* A survey on the management of uncertain data. *Chinese Journal of Computers*, 2009, 32(1): 1-16. (in Chinese)

[6] He Q, Shang T, Zhuang F *et al.* Parallel extreme learning machine for regression based on MapReduce. *Neurocomputing*, 2013, 102: 52-58.

[7] Aggarwal C C, Yu P S. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2009, 21(5): 609-623.

[8] Masud M M, Gao J, Khan L *et al.* A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Proc. the 8th IEEE International Conference on Data Mining*, December 2008, pp.929-934.

[9] Xu W, Qin Z, Chang Y. A framework for classifying uncertain and evolving data streams. *Information Technology Journal*, 2011, 10(10): 1926-1933.

[10] Domingos P, Hulten G. Mining high-speed data streams. In *Proc. the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2000, pp.71-80.

[11] Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In *Proc. the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2001, pp.97-106.

[12] Gama J, Rocha R, Medas P. Accurate decision trees for mining high-speed data streams. In *Proc. the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2003, pp.523-528.

[13] Liu J, Li X, Zhong W. Ambiguous decision trees for mining concept-drifting data streams. *Pattern Recognition Letters*, 2009, 30(15): 1347-1355.

[14] Gama J, Kosina P. Learning decision rules from data streams. In *Proc. the 22nd International Joint Conference on Artificial Intelligence*, July 2011, pp.1255-1260.

[15] Kosina P, Gama J. Handling time changing data with adaptive very fast decision rules. In *Machine Learning and Knowledge Discovery in Databases*, Flach P, Bie T, Cristianini N (eds.), Springer, 2012, pp.827-842.

[16] Frias-Blanco I, del Campo-Avila J, Ramos Jimenez G *et al.* Online and nonparametric drift detection methods based on Hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 2014, 27(3): 810-823.

[17] Street W N, Kim Y. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proc. the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2001, pp.377-382.

[18] Stanley K O. Learning concept drift with a committee of decision trees. Technical Report, UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA, 2003.

[19] Wang H, Fan W, Yu P S *et al.* Mining concept-drifting data streams using ensemble classifiers. In *Proc. the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2003, pp.226-235.

[20] Nishida K, Yamauchi K, Omori T. ACE: Adaptive classifiers-ensemble system for concept-drifting environments. In *Proc. the 6th Int. Workshop on Multiple Classifier Systems*, June 2005, pp.176-185.

[21] Li P, Wu X, Hu X *et al.* A random decision tree ensemble for mining concept drifts from noisy data streams. *Applied Artificial Intelligence*, 2010, 24(7): 680-710.

[22] Ye Y, Wu Q, Huang J Z *et al.* Stratified sampling for feature subspace selection in random forests for high dimensional data. *Pattern Recognition*, 2013, 46(3): 769-787.

[23] Liang C, Zhang Y, Song Q. Decision tree for dynamic and uncertain data streams. In *Proc. the 2nd Asian Conference on Machine Learning*, November 2010, pp.209-224.

[24] Qin B, Xia Y, Li F. DTU: A decision tree for uncertain data. In *Proc. the 13th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining*, April 2009, pp.4-15.

[25] Pan S, Wu K, Zhang Y *et al.* Classifier ensemble for uncertain data stream classification. In *Proc. the 14th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining*, June 2010, pp.488-495.

[26] Jenhani I, Amor N B, Elouedi Z. Decision trees as possibilistic classifiers. *International Journal of Approximate Reasoning*, 2008, 48(3): 784-807.

[27] Liu B, Xiao Y, Cao L *et al.* One-class-based uncertain data stream learning. In *Proc. the 11th SIAM International Conference on Data Mining*, April 2011, pp.992-1003.

[28] Cao K, Wang G, Han D *et al.* Classification of uncertain data streams based on extreme learning machine. *Cognitive Computation*, 2015, 7(1): 150-160.

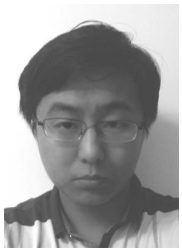
[29] Huang G B, Wang D H, Lan Y. Extreme learning machines: A survey. *International Journal of Machine Learning and Cybernetics*, 2011, 2(2): 107-122.

[30] Huang G B, Babri H A. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks*, 1998, 9(1): 224-229.

[31] Huang G B, Zhu Q Y, Siew C K. Extreme learning machine: Theory and applications. *Neurocomputing*, 2006, 70(1/2/3): 489-501.

[32] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 2008, 51(1): 107-113.

Dong-Hong Han received her M.S. and Ph.D. degrees in computer science and technology from Northeastern University, Shenyang, in 2002 and 2007, respectively. Currently, she is an associate professor in the College of Information Science and Engineering, Northeastern University, Shenyang. Her research interests include data stream management, data mining, and uncertain data management.



Xin Zhang is a postgraduate in the College of Information Science and Engineering, Northeastern University. He received his B.S. degree in computer science and technology from the College of Information Science and Engineering, Tianjin University of Finance and Economics, Tianjin, in 2011. His research interests are data stream management, uncertain data management, and data mining.

Guo-Ren Wang received his B.S., M.S. and Ph.D. degrees in computer science and technology from Northeastern University, Shenyang, in 1988, 1991 and 1996, respectively. Currently, he is a professor in the College of Information Science and Engineering, Northeastern University. His research interests are XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management.