TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 04/09 pp160–173
Volume 22, Number 2, April 2017

Distributed and Weighted Extreme Learning Machine for Imbalanced Big Data Learning

Zhiqiong Wang, Junchang Xin*, Hongxu Yang, Shuo Tian, Ge Yu, Chenren Xu, and Yudong Yao

Abstract: The Extreme Learning Machine (ELM) and its variants are effective in many machine learning applications such as Imbalanced Learning (IL) or Big Data (BD) learning. However, they are unable to solve both imbalanced and large-volume data learning problems. This study addresses the IL problem in BD applications. The Distributed and Weighted ELM (DW-ELM) algorithm is proposed, which is based on the MapReduce framework. To confirm the feasibility of parallel computation, first, the fact that matrix multiplication operators are decomposable is illustrated. Then, to further improve the computational efficiency, an Improved DW-ELM algorithm (IDW-ELM) is developed using only one MapReduce job. The successful operations of the proposed DW-ELM and IDW-ELM algorithms are finally validated through experiments.

Key words: weighted Extreme Learning Machine (ELM); imbalanced big data; MapReduce framework; user-defined counter

1 Introduction

Machine Learning (ML), as a technique for effectively predicting and analyzing data[1], has been widely used for information retrieval, medical diagnosis, and natural language understanding. Many ML algorithms have been investigated, including the Extreme Learning Machine (ELM)[2–12], which has the least human intervention, high learning accuracy, and fast learning speed. However, raw data with imbalanced class distributions, which is known as the Imbalanced Learning (IL) problem, can be found almost everywhere[13], and the existing ML algorithms including ELM cannot resolve this problem effectively. Some methods, such as undersampling[14] and oversampling[15], can rebalance the data distributions, following which standard ML algorithms can be applied. Recently, Weighted ELM (WELM)[16] has been proposed to improve the ELM algorithm for the IL problem by assigning different weights to different samples/classes in the training dataset.

Although ML algorithms, including ELM and WELM, are effective for various applications, their implementation in Big Data (BD) applications still has many challenges[17] because the data volume to be stored and analyzed greatly exceeds the storage and computing capacity of a single machine. One feasible approach is to scale up the corresponding algorithms using parallel or distributed architectures[18]. Several distributed ELMs (e.g., PELM[19], ELM*[20], DK-ELM[21], E^{2}LM[22], and A-ELM*[23]) based on the

• Zhiqiong Wang and Shuo Tian are with the Sino-Dutch Biomedical & Information Engineering School, Northeastern University, Shenyang 110169, China. E-mail: wangzq@bmie.neu.edu.cn; dyhswdza@sina.com.
• Junchang Xin, Hongxu Yang, and Ge Yu are with the School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. E-mail: xinjunchang@cse.neu.edu.cn; 443577693@qq.com; yuge@cse.neu.edu.cn.
• Chenren Xu is with the School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China. E-mail: chenren@pku.edu.cn.
• Yudong Yao is with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Castle Point on Hudson Hoboken, NJ 07030, USA. E-mail: yyao@stevens.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-08-27; revised: 2017-01-14; accepted: 2017-01-18

MapReduce framework[24–28] have been proposed to improve the scalability of the ELM algorithm to resolve the problem of BD learning. However, these algorithms did not consider the IL problem.

A centralized WELM has been proposed to manage the imbalanced class distribution; however, it processes big datasets inefficiently. Some Distributed ELMs (DELMs) have been proposed to resolve the problem of BD learning, but they are powerless to imbalanced data. Therefore, in this study, considering both the large volume of data and imbalanced training samples, we proposed a Distributed and Weighted ELM (DW-ELM).

If the algorithm, which is similar to the DELM, is directly adopted, then two steps in the matrix multiplication need to be calculated when solving the matrices $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T}$, that is, two MapReduce jobs. The matrix $\boldsymbol{A}=\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}$ would be calculated in the first MapReduce job, and then matrix $\boldsymbol{AH}$ or $\boldsymbol{AT}$ would be calculated in the second MapReduce job. Here $\boldsymbol{A}$ is an $L \times N$ dimensional big matrix, and the amount of data transferred between the two jobs is very large; therefore, the training efficiency is very low. We skillfully apply the parameter $\boldsymbol{W}$ as a diagonal matrix; then, the DW-ELM algorithm is proposed such that matrices $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T}$ can be calculated efficiently in a single MapReduce job, which reduces the amount of data transmission effectively and improves the learning efficiency. Therefore, the DW-ELM algorithm is not a simple combination of DELM and WELM but solves a more complex problem using three matrix multiplications rather than two matrix multiplications of the DELM, which is the core contribution in this study.

This study addresses the issue of IL in BD applications. A DW-ELM based on WELM and the distributed MapReduce framework is proposed. We prove that the most expensive computational part in the WELM algorithm is decomposable and can be computed in parallel. The contributions of this study are as follows. First, we prove that the matrix multiplication operators in the centralized WELM algorithm are decomposable, which indicates that its scalability can be improved by MapReduce. Second, a DW-ELM algorithm based on the MapReduce framework is proposed to learn the imbalanced BD efficiently. Finally, an Improved DW-ELM (IDW-ELM) algorithm is proposed to further improve the computational performance.

The rest of the study is organized as follows. Section 2 briefly reviews the background for our work. The theoretical foundation and computational details of the proposed DW-ELM and IDW-ELM algorithms are introduced in Section 3. The experimental results to show the effectiveness of the proposed algorithms are reported in Section 4. Finally, Section 5 concludes this study.

2 Background

In this section, we describe the background for our work, which includes a brief overview of the WELM algorithm[16] and a detailed description of the distributed MapReduce framework[24–26].

2.1 Weighted ELM[16]

For N arbitrary distinct training samples $(\boldsymbol{x}_i, \boldsymbol{t}_i)$, where $\boldsymbol{x}_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^{\mathrm{T}} \in \mathbb{R}^n$ and $\boldsymbol{t}_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^{\mathrm{T}} \in \mathbb{R}^m$, the ELM can resolve the following learning problem:

$$\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{T} \tag{1}$$

where $\boldsymbol{\beta} = [\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_L]^{\mathrm{T}}$ is the output weight matrix between the hidden and output nodes, $\boldsymbol{T} = [\boldsymbol{t}_1, \cdots, \boldsymbol{t}_N]^{\mathrm{T}}$ are the target labels, $\boldsymbol{H} = [\boldsymbol{h}^{\mathrm{T}}(\boldsymbol{x}_1), \cdots, \boldsymbol{h}^{\mathrm{T}}(\boldsymbol{x}_N)]^{\mathrm{T}}$, where $\boldsymbol{h}(\boldsymbol{x}_i) = [g_1(\boldsymbol{x}_i), \cdots, g_L(\boldsymbol{x}_i)]$ are the hidden node outputs, and $g_j(\boldsymbol{x}_i)$ is the output of the j-th hidden node for input $\boldsymbol{x}_i$.

To maximize the marginal distance and minimize the weighted cumulative error with respect to each sample, an optimization problem is mathematically written as

$$\begin{aligned} &\text{Minimize}: \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\boldsymbol{W}\frac{1}{2}\sum_{i=1}^{N}\|\boldsymbol{\xi}_i\|^2 \\ &\text{Subject to}: \boldsymbol{h}(\boldsymbol{x}_i)\boldsymbol{\beta} = \boldsymbol{t}_i^{\mathrm{T}} - \boldsymbol{\xi}_i^{\mathrm{T}} \end{aligned} \tag{2}$$

where C is the regularization parameter to represent the trade-off between the minimized weighted cumulative error and maximized marginal distance. $\boldsymbol{\xi}_i$ is the training error of sample $\boldsymbol{x}_i$ caused by the difference of the desired output $\boldsymbol{t}_i$ and the actual output $\boldsymbol{h}(\boldsymbol{x}_i)\boldsymbol{\beta}$. Here $\boldsymbol{W}$ is an $N \times N$ diagonal matrix associated with every training sample $\boldsymbol{x}_i$ and

$$W_{ii} = 1/\#(\boldsymbol{t}_i) \tag{3}$$

or

$$W_{ii} = \begin{cases} 0.618/\#(\boldsymbol{t}_i), & \text{if } \#(\boldsymbol{t}_i) > \text{AVG}; \\ 1/\#(\boldsymbol{t}_i), & \text{if } \#(\boldsymbol{t}_i) \leqslant \text{AVG} \end{cases} \tag{4}$$

where $\#(\boldsymbol{t}_i)$ is the number of samples belonging to class $\boldsymbol{t}_i$ and AVG is the average number of samples per class.

According to the Karush-Kuhn-Tucker theorem[29], the following solutions for the WELM can be obtained:

$$\boldsymbol{\beta} = \left(\frac{\boldsymbol{I}}{\lambda} + \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}\right)^{-1}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T} \tag{5}$$

when $N \gg L$, or

$$\boldsymbol{\beta} = \boldsymbol{H}^{\mathrm{T}} \left(\frac{\boldsymbol{I}}{\lambda} + \boldsymbol{W}\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}} \right)^{-1} \boldsymbol{W}\boldsymbol{T} \tag{6}$$

when $N \ll L$.

2.2 MapReduce framework[24–26]

MapReduce is a simple and flexible parallel programming model initially proposed by Google for large scale data processing in a distributed computing environment[24–26], with Hadoop (http://hadoop.apache.org/) being one of its open source implementations. The typical procedure of a MapReduce job is illustrated in Fig. 1. First, the input to a MapReduce job starts as the dataset stored on the underlying distributed file system (e.g., Google File System (GFS)[30] and Hadoop Distributed File System (HDFS)[31]), which is split into several files across machines. Next, the MapReduce job is partitioned into many independent map tasks. Each map task processes a logical split of the input dataset. The map function takes a key-value pair (k_1, v_1) as input and generates intermediate key-value pairs $[(k_2, v_2)]$. Next, the intermediate data files from the already completed map tasks are fetched by the corresponding reduce task following the "pull" model (similarly, there are many independent reduce tasks). The intermediate data files from all the map tasks are sorted accordingly and passed to the corresponding reduce task. The reduce function combines all intermediate key-value pairs $(k_2, [v_2])$ with the same key into the output key-value pairs $[(k_3, v_3)]$. Finally, the output data from the reduce task is generally written back to the corresponding distributed file system.

3 Distributed and Weighted Extreme Learning Machine

In this section, we present the DW-ELM and IDW-ELM algorithms. To implement the parallel computation of

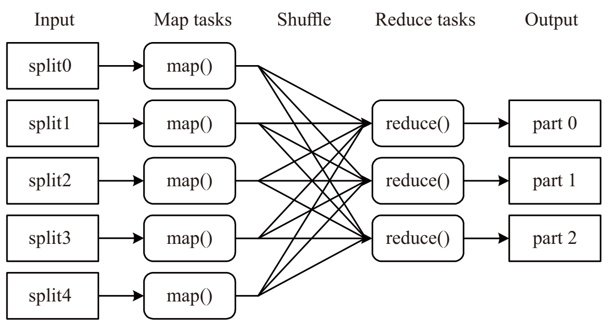


Fig. 1 Illustration of the MapReduce framework.

the WELM algorithm using the MapReduce framework, we first need to prove the matrix decomposability in the WELM algorithm.

3.1 Decomposability of the WELM algorithm

In imbalanced BD learning applications, the number of training records is much larger than the number of hidden nodes, that is, $N \gg L$. Therefore, the size of $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$ is much smaller than that of $\boldsymbol{W}\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}}$. Equation (5) is used to calculate the output weight vector $\boldsymbol{\beta}$ in WELM. We first analyze the properties of the WELM algorithm and formulate the matrix operations in the form of the MapReduce framework. In this way, we can extend the WELM algorithm for BD applications.

Let $\boldsymbol{U} = \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$ and $\boldsymbol{V} = \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T}$. Thus, we obtain

$$\boldsymbol{\beta} = \left(\frac{\boldsymbol{I}}{\lambda} + \boldsymbol{U} \right)^{-1} \boldsymbol{V} \tag{7}$$

Following the matrix operations, we have Eq. (8).

Then, we further obtain

$$u_{ij} = \sum_{k=1}^{N} W_{kk} \times g(\boldsymbol{w}_i \cdot \boldsymbol{x}_k + b_i) \times g(\boldsymbol{w}_j \cdot \boldsymbol{x}_k + b_j) \tag{9}$$

Similarly, following the matrix operations, we obtain Eq. (10).

Finally, we obtain Eq. (11).

$$v_{ij} = \sum_{k=1}^{N} W_{kk} \times g(\boldsymbol{w}_i \cdot \boldsymbol{x}_k + b_i) \times t_{kj} \tag{11}$$

According to Eq. (9), we know that the item u_{ij} in matrix $\boldsymbol{U}$ can be expressed by the summation of $W_{kk} \times g(\boldsymbol{w}_i \cdot \boldsymbol{x}_k + b_i) \times g(\boldsymbol{w}_j \cdot \boldsymbol{x}_k + b_j)$. Here W_{kk} is the weight of the training sample $(\boldsymbol{x}_k, \boldsymbol{t}_k)$, and $h_{ki} = g(\boldsymbol{w}_i \cdot \boldsymbol{x}_k + b_i)$ and $h_{kj} = g(\boldsymbol{w}_j \cdot \boldsymbol{x}_k + b_j)$ are the i-th and j-th elements in the k-th row $h(\boldsymbol{x}_k)$ of the hidden layer output matrix $\boldsymbol{H}$, respectively. Similarly, according to Eq. (11), we know that item v_{ij} in matrix $\boldsymbol{V}$ can be expressed by the summation of $W_{kk} \times g(\boldsymbol{w}_i \cdot \boldsymbol{x}_k + b_i) \times t_{kj}$. Here t_{kj} is the j-th element in the k-th row $\boldsymbol{t}_k$ of matrix $\boldsymbol{T}$ that is related to the training sample $(\boldsymbol{x}_k, \boldsymbol{t}_k)$.

The variables involved in the equations for the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ include W_{kk}, h_{ki}, h_{kj}, and t_{kj}. According to Eqs. (3) and (4), to calculate the corresponding weight W_{kk} related to the training sample $(\boldsymbol{x}_k, \boldsymbol{t}_k)$, we must first obtain the number $\#(\boldsymbol{t}_k)$ of training samples that belong to the same class as $\boldsymbol{t}_k$. The number of training samples in all classes can be easily calculated in one MapReduce job. At the same time, the remaining three variables, namely h_{ki}, h_{kj}, and t_{kj}, are

$$
\begin{aligned}
\boldsymbol{U} = {} & \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H} = \\
& \begin{bmatrix} h(\boldsymbol{x}_1) \\ h(\boldsymbol{x}_2) \\ \vdots \\ h(\boldsymbol{x}_N) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} W_{11} & 0 & \cdots & 0 \\ 0 & W_{22} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & W_{NN} \end{bmatrix} \begin{bmatrix} h(\boldsymbol{x}_1) \\ h(\boldsymbol{x}_2) \\ \vdots \\ h(\boldsymbol{x}_N) \end{bmatrix} = \\
& \begin{bmatrix} h(\boldsymbol{x}_1)^{\mathrm{T}} & h(\boldsymbol{x}_2)^{\mathrm{T}} & \cdots & h(\boldsymbol{x}_N)^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} W_{11} & 0 & \cdots & 0 \\ 0 & W_{22} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & W_{NN} \end{bmatrix} \begin{bmatrix} h(\boldsymbol{x}_1) \\ h(\boldsymbol{x}_2) \\ \vdots \\ h(\boldsymbol{x}_N) \end{bmatrix} = \\
& h(\boldsymbol{x}_1)^{\mathrm{T}} W_{11} h(\boldsymbol{x}_1) + h(\boldsymbol{x}_2)^{\mathrm{T}} W_{22} h(\boldsymbol{x}_2) + \cdots + h(\boldsymbol{x}_N)^{\mathrm{T}} W_{NN} h(\boldsymbol{x}_N) = \\
& \sum_{k=1}^{N} h(\boldsymbol{x}_k)^{\mathrm{T}} W_{kk} h(\boldsymbol{x}_k) = \\
& \sum_{k=1}^{N} \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_k + b_1) \\ g(\boldsymbol{w}_2 \cdot \boldsymbol{x}_k + b_2) \\ \vdots \\ g(\boldsymbol{w}_L \cdot \boldsymbol{x}_k + b_L) \end{bmatrix} W_{kk} \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_k + b_1) & g(\boldsymbol{w}_2 \cdot \boldsymbol{x}_k + b_2) & \cdots & g(\boldsymbol{w}_L \cdot \boldsymbol{x}_k + b_L) \end{bmatrix} = \\
& \sum_{k=1}^{N} W_{kk} \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_k + b_1) \\ g(\boldsymbol{w}_2 \cdot \boldsymbol{x}_k + b_2) \\ \vdots \\ g(\boldsymbol{w}_L \cdot \boldsymbol{x}_k + b_L) \end{bmatrix} \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_k + b_1) & g(\boldsymbol{w}_2 \cdot \boldsymbol{x}_k + b_2) & \cdots & g(\boldsymbol{w}_L \cdot \boldsymbol{x}_k + b_L) \end{bmatrix}
\end{aligned} \tag{8}
$$

$$
\begin{aligned}
\boldsymbol{V} = {} & \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T} = \\
& \begin{bmatrix} h(\boldsymbol{x}_1) \\ h(\boldsymbol{x}_2) \\ \vdots \\ h(\boldsymbol{x}_N) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} W_{11} & 0 & \cdots & 0 \\ 0 & W_{22} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & W_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \vdots \\ \boldsymbol{t}_N \end{bmatrix} = \\
& \begin{bmatrix} h(\boldsymbol{x}_1)^{\mathrm{T}} & h(\boldsymbol{x}_2)^{\mathrm{T}} & \cdots & h(\boldsymbol{x}_N)^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} W_{11} & 0 & \cdots & 0 \\ 0 & W_{22} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & W_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \vdots \\ \boldsymbol{t}_N \end{bmatrix} = \\
& h(\boldsymbol{x}_1)^{\mathrm{T}} W_{11} \boldsymbol{t}_1 + h(\boldsymbol{x}_2)^{\mathrm{T}} W_{22} \boldsymbol{t}_2 + \cdots + h(\boldsymbol{x}_N)^{\mathrm{T}} W_{NN} \boldsymbol{t}_N = \\
& \sum_{i=1}^{N} h(\boldsymbol{x}_k)^{\mathrm{T}} W_{kk} \boldsymbol{t}_k = \\
& \sum_{k=1}^{N} \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_k + b_1) \\ g(\boldsymbol{w}_2 \cdot \boldsymbol{x}_k + b_2) \\ \vdots \\ g(\boldsymbol{w}_L \cdot \boldsymbol{x}_k + b_L) \end{bmatrix} W_{kk} \begin{bmatrix} t_{k1} & t_{k2} & \cdots & t_{km} \end{bmatrix} = \\
& \sum_{k=1}^{N} W_{kk} \begin{bmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}_k + b_1) \\ g(\boldsymbol{w}_2 \cdot \boldsymbol{x}_k + b_2) \\ \vdots \\ g(\boldsymbol{w}_L \cdot \boldsymbol{x}_k + b_L) \end{bmatrix} \begin{bmatrix} t_{k1} & t_{k2} & \cdots & t_{km} \end{bmatrix}
\end{aligned} \tag{10}
$$

only related to the training sample $(\boldsymbol{x}_k, \boldsymbol{t}_k)$ itself and not the other training samples; therefore, the calculation of the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices can be accomplished in another MapReduce job.

To summarize, the calculation process of the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices is decomposable; therefore, we can realize the parallel computation of these matrices using the MapReduce framework to break through the limitation of a single machine and improve the efficiency in which the WELM algorithm learns the imbalanced big training data.

3.2 DW-ELM algorithm

The process for the DW-ELM algorithm is shown in Algorithm 1. First, we randomly generate L pairs of hidden node parameters $(\boldsymbol{w}_i, b_i)$ (lines 1 and 2). Then, we use a MapReduce job to count the number of training samples contained in each class (line 3). Next, we use another MapReduce job to calculate the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices according to the input parameters and randomly generate the parameters (line 4). Finally, we solve the output weight vector $\boldsymbol{\beta}$ [according to Eq. (5) (line 5)].

We describe the specific processes of the two MapReduce jobs involved in the DW-ELM algorithm.

The process for the first MapReduce job of the DW-ELM algorithm is shown in Algorithm 2. The algorithm includes two classes, Mapper (lines 1–10) and Reducer (lines 11–16). Class Mapper contains three methods: Initialize (lines 2 and 3), Map (lines 4–7), and Close (lines 8–10), whereas Class Reducer only contains one method, namely Reduce (lines 12–16). In the Initialize method of the Mapper class, we initialize one array c, which is used to store the intermediate summation of the training samples contained in each class (line 3). In the Map method of the Mapper class, first, we analyze the training sample s and resolve the class to which sample s belongs (lines 5 and 6). Then, we adjust the corresponding value in the array c (line 7). In the Close method of the Mapper class, the intermediate summations stored in c are emitted by the mapper (lines 9 and 10). In the Reduce method of the Reducer class, first, we initialize a temporary variable sum (line 13). Next, we combine the intermediate summations of the different mappers that have the same Key and obtain the final summation of the corresponding element of the Key (lines 14 and 15). Finally, we store the results in a distributed file system (line 16).

The process for the second MapReduce job of the DW-ELM algorithm is shown in Algorithm 3. In the Initialize method of the Mapper class, we initialize two arrays, u and v, which are used to store the intermediate summations of the elements in the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices, respectively. In the Map method of the Mapper class, first, we initialize a local variable h (line 6). Then, we resolve the input training sample s and divide s into the training feature $\boldsymbol{x}$ and its corresponding training result $\boldsymbol{t}$ (line 7). Again, according to the training

Algorithm 1 DW-ELM

```
1 for $i = 1$ to $L$ do
2     Randomly generate hidden node parameters $(\boldsymbol{w}_i, b_i)$
3 Calculate all $\#(\boldsymbol{t}_k)$ using Algorithm 2
4 Calculate $\boldsymbol{U} = \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$, $\boldsymbol{V} = \boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T}$ using Algorithm 3
5 Calculate the output weight vector $\boldsymbol{\beta} = (\boldsymbol{I}/\lambda + \boldsymbol{U})^{-1}\boldsymbol{V}$
```

Algorithm 2 First MapReduce Job of the DW-ELM Algorithm

```
1 class Mapper
2     method Initialize()
3         $c$ = new AssociativeArray
4     method Map(sid id, sample $s$)
5         $\boldsymbol{t}$ = ParseT($s$)
6         num = Class($\boldsymbol{t}$)
7         $c$[num] = $c$[num] + 1
8     method Close()
9         for $i = 1$ to $c$.Length() do
10            context.write(cid $i$, count $c[i]$)
11 class Reducer
12    method Reduce(cid id, counts $[c_1, c_2, \ldots]$)
13        sum = 0
14        for all count $c \in [c_1, c_2, \ldots]$ do
15            sum = sum + $c$
16        context.write(cid id, count sum)
```

Algorithm 3 Second MapReduce Job of the DW-ELM Algorithm

```
1 class Mapper
2     method Initialize()
3         $u$ = new AssociativeArray
4         $v$ = new AssociativeArray
5     method Map(sid id, sample $s$)
6         $h$ = new AssociativeArray
7         $(\boldsymbol{x}, \boldsymbol{t})$ = ParseAll($s$)
8         $w$ = Weight(Counts[Class($\boldsymbol{t}$)])
9         for $i = 1$ to $L$ do
10            $h[i] = g(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i)$
11        for $i = 1$ to $L$ do
12            for $j = 1$ to $L$ do
13                $u[i,j] = u[i,j] + w \times h[i] \times h[j]$
14            for $j = 1$ to $m$ do
15                $v[i,j] = v[i,j] + w \times h[i] \times \boldsymbol{t}[j]$
16    method Close()
17        for $i = 1$ to $L$ do
18            for $j = 1$ to $L$ do
19                context.write(triple $('U', i, j)$, sum $u[i,j]$)
20            for $j = 1$ to $m$ do
21                context.write(triple $('V', i, j)$, sum $v[i,j]$)
22 class Reducer
23    method Reduce(triple $p$, sum $[s_1, s_2, \ldots]$)
24        $uv = 0$
25        for all sum $s \in [s_1, s_2, \ldots]$ do
26            $uv = uv + s$
27        context.write(triple $p$, sum $uv$)
```

result $\boldsymbol{t}$ and the result of Algorithm 2, we obtain the corresponding weight w of s (line 8). Next, we calculate the corresponding hidden layer output vector $h(\boldsymbol{x})$ (lines 9 and 10). Finally, we separately calculate the local summations of the elements in the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices and save the result to local variables u and v (lines 11–15). In the Close method of the Mapper class, the intermediate summations stored in u and v are emitted by the mapper (lines 17–21). In the Reduce method of the Reducer class, first, we initialize a temporary variable uv (line 24). Next, we combine the intermediate summations of the different mappers that have the same Key and obtain the final summation of the corresponding element of the Key (lines 25 and 26). Finally, we store the results in the distributed file system (line 27).

The DW-ELM algorithm requires two MapReduce jobs, where job#1 is used to count the number of samples, $\#(t_k)$, of each t_k category in m categories; therefore, the total amount of data that job#1 needs to transmit in the Map phase is the total number of training samples multiplied by the number of Mapper. Assuming the number of Mappers is N_{Mapper}. Then, we can obtain

$$\text{cost}_{\text{basic1}} = \text{sizeof}\,(\text{Integer}) \cdot m \cdot N_{\text{Mapper}} \tag{12}$$

Job#2 is used to calculate the $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T}$ matrices. Therefore, the total amount of data to be transferred to job#2 is the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrix size multiplied by N_{Mapper} in the Map phase. Assuming that the number of hidden layer nodes is L, the number of categories is m, and the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices are $L \times L$ and $L \times m$, respectively, we further obtain

$$\text{cost}_{\text{basic2}} = \text{sizeof}\,(\text{float}) \cdot (L \cdot L + L \cdot m) \cdot N_{\text{Mapper}} \tag{13}$$

and

$$\begin{aligned}&\text{cost}_{\text{basic}} = \\ &\quad \text{cost}_{\text{basic1}} + \text{cost}_{\text{basic2}} = \\ &\quad \text{sizeof}\,(\text{Integer}) \cdot C \cdot N_{\text{Mapper}} + \\ &\quad \text{sizeof}\,(\text{float}) \cdot (L \cdot L + L \cdot m) \cdot N_{\text{Mapper}}\end{aligned} \tag{14}$$

3.3 IDW-ELM algorithm

The DW-ELM algorithm requires two MapReduce jobs to complete the training process. If we can merge the above two MapReduce jobs into a single MapReduce job, the performance of the DW-ELM algorithm can be significantly improved. In addition to providing several built-in counters, the MapReduce framework allows users to employ their own user-defined counters. Using a user-defined counter, users can transfer some statistical information between the Mapper and Reducer classes. The task of the first MapReduce job in the DW-ELM algorithm is to count the number of training samples contained in each class; however, we propose an IDW-ELM algorithm based on user-defined counters, using only one MapReduce job to complete the tasks. The framework is shown in Fig. 2.

The process for the MapReduce job of the IDW-ELM algorithm is shown in Algorithm 4. The Initialize method of the Mapper class is similar to Algorithm 3. In the Map method of the Mapper class, first, we initial a local variable h (line 6). Then, we resolve the training sample s, which means dividing s into the training feature $\boldsymbol{x}$ and its corresponding training result $\boldsymbol{t}$ (line 7). Again, we resolve the class that sample s belongs to according to the training result $\boldsymbol{t}$, and adjust the value of the corresponding user-defined counter (lines 8 and 9). Next, we calculate the hidden layer output vector $h(\boldsymbol{x})$ corresponding to $\boldsymbol{x}$ (lines 10 and 11). Finally, we separately calculate the local summations of the elements in the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices and save the result to local variables u and v (lines 12–16) (the results of different class need to be stored respectively, so u and v are 3-dimensional arrays.). In the Close method of the Mapper class, the intermediate summations stored in u and v are emitted by the mapper (lines 18–23). In the Reduce method of the Reducer class, first, we initialize a temporary variable uv (line 25). Next, we calculate the corresponding weight according to the statistical information stored in the user-defined counters to combine the intermediate summations of the different mappers that have the same Key and obtain the final summation of the corresponding element of the Key (lines 27–29). Finally, we store the results in the distributed file system (line 30).

With the IDW-ELM algorithm, only one MapReduce job is required using a user-defined counter to complete the calculation; local accumulation of u and v need to be computed separately in each classifier. The $\text{cost}_{\text{improved1}}$ variable is the amount of counter transmission. Hence, we can obtain

$$\begin{aligned}&\text{cost}_{\text{improved}} = \\ &\quad \text{cost}_{\text{improved1}} + \text{cost}_{\text{improved2}} = \\ &\quad \text{sizeof}\,(\text{Integer}) \cdot C \cdot N_{\text{Mapper}} + \\ &\quad \text{sizeof}\,(\text{float}) \cdot C \cdot (L \cdot L + L \cdot m) \cdot N_{\text{Mapper}}\end{aligned} \tag{15}$$

Assuming that the network bandwidth is B, the

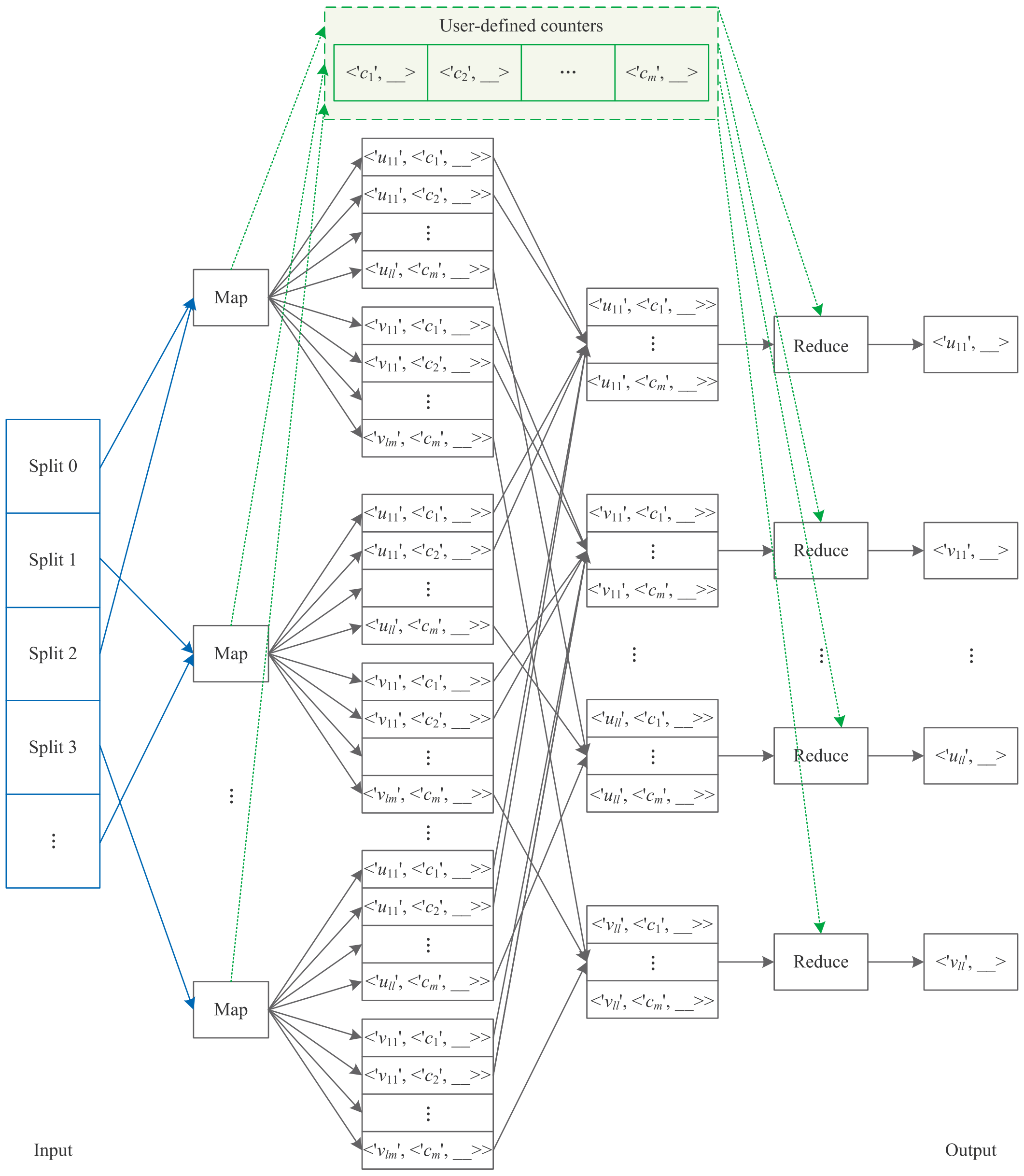

User-defined counters
<'c_1', _>
<'c_2', _>
<'c_m', _>
Split 0
Split 1
Split 2
Split 3
Map
Map
Map
<'u_{11}', <'c_1', _>>
<'u_{11}', <'c_2', _>>
<'u_{ll}', <'c_m', _>>
<'v_{11}', <'c_1', _>>
<'v_{11}', <'c_2', _>>
<'v_{lm}', <'c_m', _>>
<'u_{11}', <'c_m', _>>
<'v_{11}', <'c_m', _>>
<'u_{ll}', <'c_1', _>>
<'v_{ll}', <'c_1', _>>
<'v_{ll}', <'c_m', _>>
Reduce
<'u_{11}', _>
<'v_{11}', _>
<'u_{ll}', _>
<'v_{ll}', _>
Input
Output

Fig. 2 The framework of the IDW-ELM algorithm.

transmission times of the DW-ELM and IDW-ELM algorithms are, respectively,

$$T_{\text{basic}} = \text{cost}_{\text{basic}}/B = \text{sizeof (Integer)} \cdot C \cdot N_{\text{Mapper}}/B + \text{sizeof (float)} \cdot (L \cdot L + L \cdot m) \cdot N_{\text{Mapper}}/B \qquad (16)$$

and

$$T_{\text{improved}} = \text{cost}_{\text{improved}}/B = \text{sizeof (Integer)} \cdot C \cdot N_{\text{Mapper}}/B + \text{sizeof (float)} \cdot C \cdot (L \cdot L + L \cdot m) \cdot N_{\text{Mapper}}/B \qquad (17)$$

If the start-up time of the MapReduce job is T_{start} and if $T_{\text{improved}} - T_{\text{basic}} < T_{\text{start}}$, then the performance of the IDW-ELM algorithm is better than that of the DW-ELM algorithm; else $T_{\text{improved}} - T_{\text{basic}} > T_{\text{start}}$ and the performance of the IDW-ELM algorithm is lower than that of the DW-ELM algorithm. Through the analysis of the communication cost of the DW-ELM and IDW-ELM algorithms, we can see that when C is large, the communication cost of the IDW-ELM algorithm will be far greater than that of the DW-ELM algorithm. Under normal circumstances, the number of data classifications is usually small and the network bandwidth is large; therefore, the performance of the IDW-ELM algorithm in the experiment is better than that of the DW-ELM algorithm.

Algorithm 4 The IDW-ELM Algorithm

```
class MAPPER
  method INITIALIZE()
    $u$ = new ASSOCIATIVEARRAY
    $v$ = new ASSOCIATIVEARRAY
  method MAP(sid id, sample $s$)
    $h$ = new ASSOCIATIVEARRAY
    $(\boldsymbol{x}, \boldsymbol{t})$ = ParseAll($s$)
    $c$ = Class($\boldsymbol{t}$)
    context.getCounter("Classes", "c"+$c$).increment(1)
    for $i = 1$ to $L$ do
      $h[i] = g(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i)$
    for $i = 1$ to $L$ do
      for $j = 1$ to $L$ do
        $u[c, i, j] = u[c, i, j] + h[i] \times h[j]$
      for $j = 1$ to $m$ do
        $v[c, i, j] = v[c, i, j] + h[i] \times \boldsymbol{t}[j]$
  method CLOSE()
    for $k = 1$ to $u$.Length() do
      for $i = 1$ to $L$ do
        for $j = 1$ to $L$ do
          context.write(triple $('U', i, j)$, value $(k, u[k, i, j])$)
        for $j = 1$ to $m$ do
          context.write(triple $('V', i, j)$, value $(k, v[k, i, j])$)
class REDUCER
  method REDUCE(triple $p$, values $[s_1, s_2, \ldots]$)
    $uv = 0$
    for all sum $s \in [s_1, s_2, \ldots]$ do
      count = context.getCounter("Classes", "c" + $s$.get-
              Class()).getValue()
      $uv = uv +$ Weight(count)$\times s$.getValue()
    context.write(triple $p$, sum $uv$)
```

4 Results and Performance Evaluation

In this section, the performances of our proposed DW-ELM and IDW-ELM algorithms are evaluated in detail by considering various experimental settings. We first describe the platform used in our experiments in Section 4.1. Then, we present and discuss the experimental results.

4.1 Experimental platform

All the experiments are run in a cluster with nine computers connected in a high-speed Gigabit network. Each computer has an Intel Quad Core 2.66 GHz CPU, 4 GB memory, and CentOS Linux 5.6 operating system. One computer is set as the master node and the others are set as the slave nodes. We use Hadoop version 0.20.2 and configure it to run up to 4 map tasks or 4 reduce tasks concurrently per node. Therefore, at any point in time, at most, 32 map tasks or 32 reduce tasks can run concurrently in our cluster.

In this experiment, the synthetic data that was randomly generated in the $[0, 1]^d$ data space are used to verify the efficiency and effectiveness of our proposed DW-ELM and IDW-ELM algorithms. The generation procedure of the experimental data is as follows. First, the data center of each class is generated randomly; second, the data of each class are generated according to the multivariate Gaussian distribution, where the former generated center point is used as the mean, whereas the reciprocal of the number of classes is the variance. Finally, the generated data of all classes are combined into the experimental data. The real-life datasets with yeast, glass, ecoli, pima, and colon are downloaded online. The imbalance ratio of the dataset can be as low as 0.0304, and the highest imbalance ratio occurs at 0.6667. However, both the aduit and banana datasets are from UCI, where the imbalance ratios are 0.3306 and 0.8605, respectively.

Table 1 summarizes the parameters used in our experimental evaluation. In each experiment, we vary a single parameter while setting all others to their default values. The imbalance ratio which quantitatively measures the imbalance degree of a dataset is defined as follows[16]:

$$\text{Imbalance ratio} = \text{Min}(\#(\boldsymbol{t}_i))/\text{Max}(\#(\boldsymbol{t}_i)) \tag{18}$$

The DW-ELM algorithm is a MapReduce-based distributed implementation of the original or centralized WELM algorithm, and it does not change the formulation in the WELM algorithm. Therefore, it does not have any impact on the classification accuracy. We evaluate the training time and speedup of the DW-ELM and IDW-ELM algorithms. The speedup achieved by an m-computer mega system is defined as

$$\text{Speedup}(m) = \frac{\text{Computing time on 1 computer}}{\text{Computing time on } m \text{ computers}} \tag{19}$$

Table 1 Experimental parameters.

Parameter	Range	Default
Number of nodes	1, 2, 3, 4, 5, 6, 7, 8	8
Dimensionality	10, 20, 30, 40, 50	50
Number of hidden nodes	100, 150, 200, 250, 300	200
Number of records	9×10^6, 12×10^6, 15×10^6, 18×10^6, 21×10^6	21×10^6
Number of classes	5, 10, 15, 20, 25	15
Imbalance ratio	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7	0.5

4.2 Experimental results

Table 2 describes the accuracy of the ELM*[20] and DW-ELM algorithms under the real-life datasets. From Table 2, the DW-ELM algorithm outperforms the ELM*[20] algorithm.

The next experiment verifies the DW-ELM algorithm training time and the acceleration ratio under the synthetic datasets. First, the impact of the number of slave nodes in the cluster on the running time of the ELM*[20], DW-ELM, and IDW-ELM algorithms is discussed. As shown in Fig. 3a, the training times for the ELM*[20], DW-ELM, and IDW-ELM algorithms decrease significantly with increasing the number of slave nodes in the cluster, and the training time of the ELM*[20] algorithm is always lower than that of the DW-ELM and IDW-ELM algorithms. Moreover, the IDW-ELM algorithm is always lower than that of the DW-ELM algorithm. The various speedup ratios, through changing the number of slave nodes, are shown in Fig. 3b, and the speedup ratio closely follows a linear growth trend. When the number of slave nodes increases, the number of map and reduce tasks that can be executed simultaneously also increases, which means the amount of work that can be completed simultaneously. Therefore, on the premise of the same amount of work, the training time for the ELM*[20], DW-ELM, and IDW-ELM algorithms decreases with increasing number of slave nodes.

Second, we investigate the impact of the training data dimensionality. As shown in Fig. 4a, with the increase of training data dimensionality, the training times of the ELM*[20], DW-ELM, and IDW-ELM algorithms all increase slightly, while the training time of the ELM*[20] algorithm is significantly lower than that of the DW-ELM and IDW-ELM algorithms. Also, the training time of the IDW-ELM algorithm is always lower than that of the DW-ELM algorithm. The variation trend of speedup ratio of three algorithms is shown in Fig. 4b, where the speedup ratios of three algorithms remain stable with the change of the data dimensionality, and the algorithm performance of the IDW-ELM algorithm is slightly better than that of the DW-ELM algorithm. Increasing the training data dimensionality leads to the running time for calculating the corresponding row h_k of the hidden layer output matrix $\boldsymbol{H}$ in Mapper to slightly increase; thus, the training times of the DW-ELM and IDW-ELM algorithms also slightly increase. The DW-ELM algorithm uses two MapReduce jobs to complete the calculation of the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices, but the IDW-ELM algorithm only uses one MapReduce jobs to complete the same calculation. Although, compared to the second MapReduce job of the DW-ELM algorithm, the amount of middle data transmission slightly increased, but its transmission time is far less than the time required by

Table 2 Performance results of the ELM*[20] and DW-ELM algorithms.

Datasets (imbalance ratio)	Testing result (%)	
	ELM*	DW-ELM
Yeast (0.0304)	80.14	94.22
Glass (0.1554)	93.99	95.47
Ecoli (0.1806)	91.09	93.81
Aduit (0.3306)	72.87	80.76
Pima (0.5350)	70.52	71.81
Colon (0.6667)	84.78	85.16
Banana (0.8605)	87.98	88.04

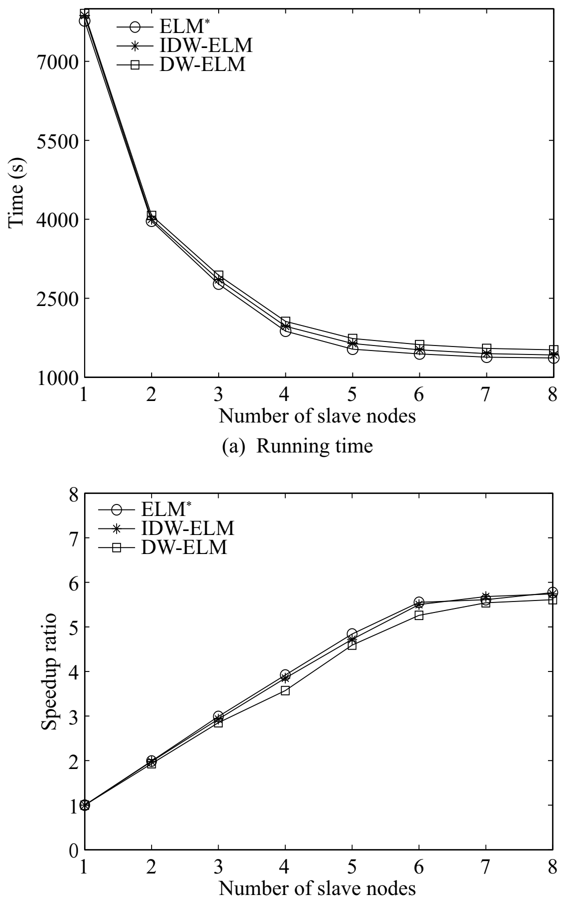
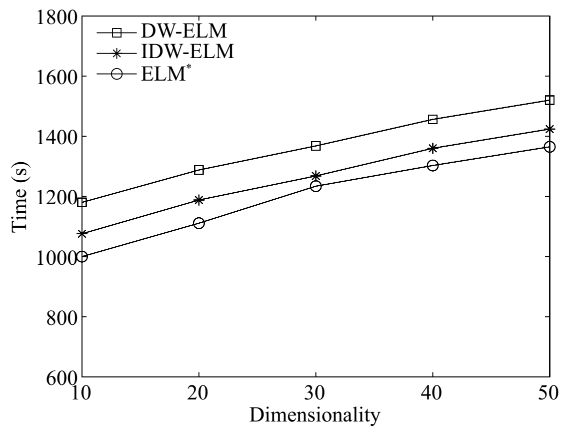
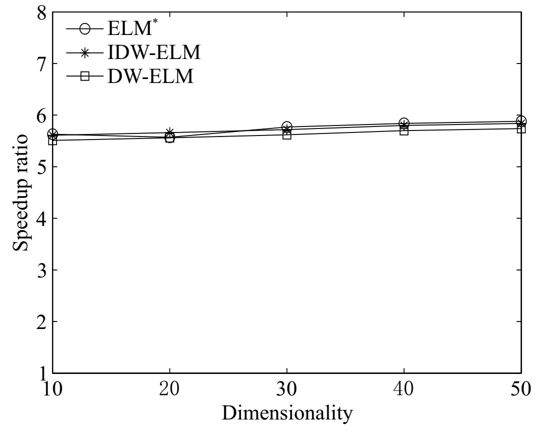


Fig. 3 The impact of the number of slave nodes on (a) time and (b) speedup ratio.

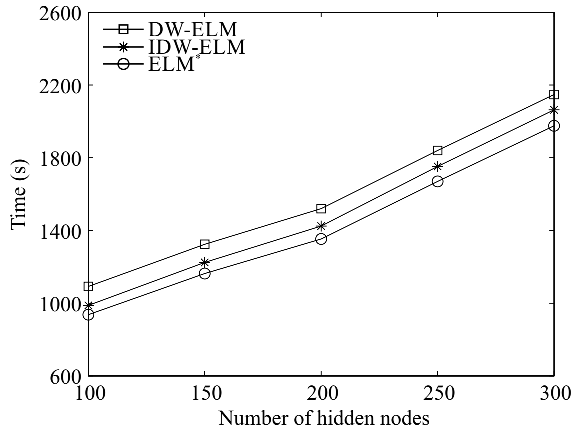


(a) Running time

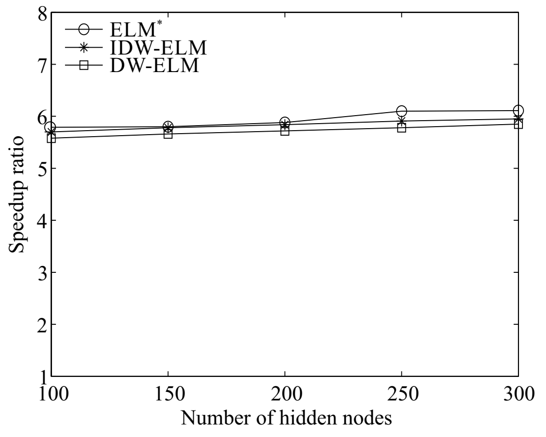


(b) Speedup ratio

Fig. 4 The impact of dimensionality on (a) time and (b) speedup ratio.



(a) Running time



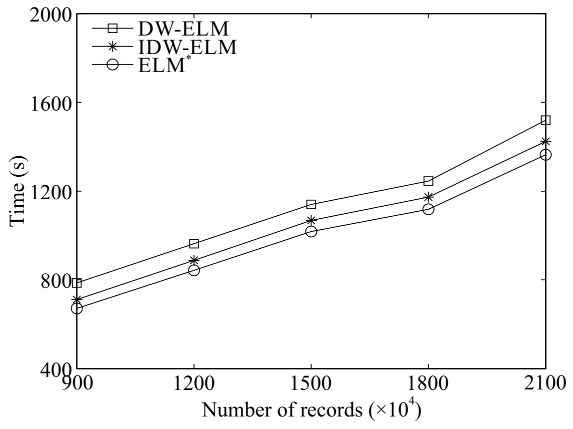
(b) Speedup ratio

Fig. 5 The impact of the number of hidden nodes on (a) time and (b) speedup ratio.

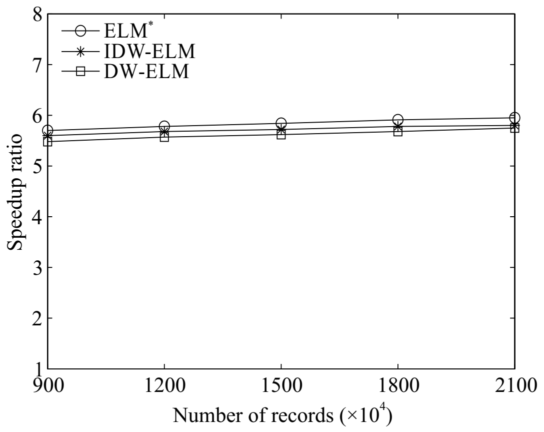
the first MapReduce job of the DW-ELM algorithm. Therefore, the training time of the IDW-ELM algorithm was significantly lower than that of the DW-ELM algorithm, illustrating the effectiveness of the proposed improved algorithm.

Again, we investigate the impact of the number of hidden nodes. As shown in Fig. 5a, by increasing the number of hidden nodes, the training times of the ELM*[20], DW-ELM, and IDW-ELM algorithms all increase, while the training time of the IDW-ELM algorithm is significantly lower than that of DW-ELM algorithm. In addition, the ELM*[20] algorithm is lower than that of IDW-ELM algorithm. The various of speedup ratios through changing the number of hidden nodes is shown in Fig. 5b, and the speedup ratio remains stable. Increasing the number of hidden nodes leads to the dimensionality of the hidden layer output matrix $\boldsymbol{H}$ to increase, and indirectly leads to the increase of the dimensionality of the intermediate $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices. This not only makes the computation time of the local accumulated sum of $\boldsymbol{U}$ and $\boldsymbol{V}$ increase, but also makes the transmission time of the intermediate results in the MapReduce job increase. Therefore, the training times for the DW-ELM and IDW-ELM algorithms increase with the number of hidden nodes. The DW-ELM algorithm uses two MapReduce jobs while the IDW-ELM algorithm uses only one MapReduce jobs to complete the calculation of the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices. Overall, the training time of the IDW-ELM algorithm is less than that of the DW-ELM algorithm, which verifies the validity of the improved algorithm.

Then, we investigate the impact of the number of training records. As shown in Fig. 6a, with the increasing number of records, the training times of the ELM*[20], DW-ELM, and IDW-ELM algorithms all increase, obviously. While the training time of the IDW-ELM algorithm is significantly lower than that of the DW-ELM algorithm and the ELM*[20] algorithm is significantly lower than that of the IDW-ELM algorithm. The impact of increasing the number

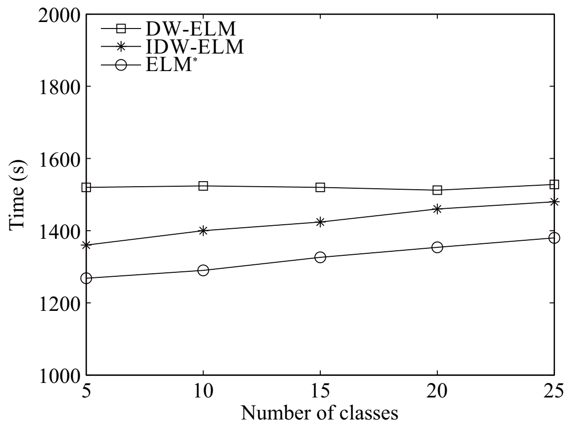


(a) Running time

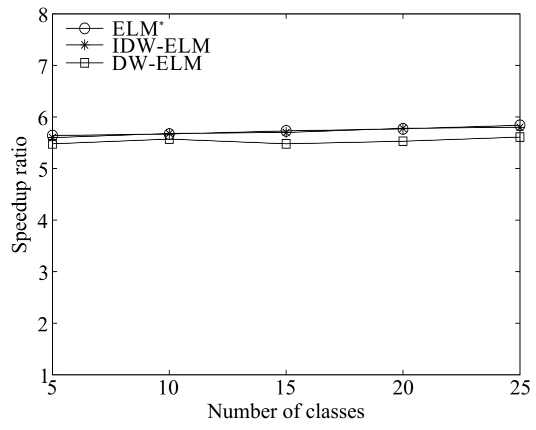


(b) Speedup ratio

Fig. 6 The impact of the number of records on (a) time and (b) speedup ratio.



(a) Running time



(b) Speedup ratio

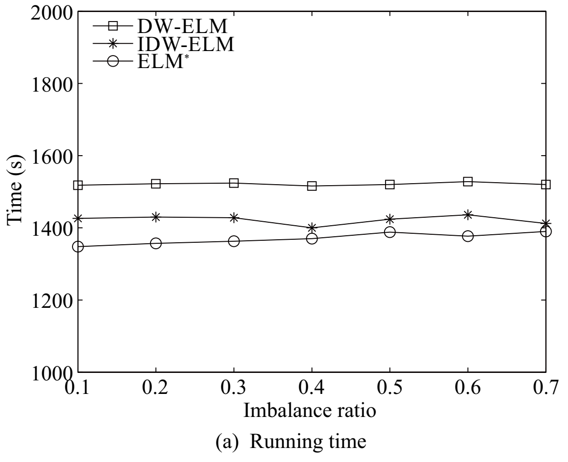
Fig. 7 The impact of the number of classes on (a) time and (b) speedup ratio.

of training records on the speedup ratio is shown in Fig. 6b. Although the speedup ratio does not achieve a linear change, but the speedup ratio of the ELM*[20] algorithm is slightly higher than that of the DW-ELM and IDW-ELM algorithms. Increasing the number of records means that the numbers of Mapper and Reducer which need to be launched increase. On the other hand, it increases the number of corresponding locally accumulated summation of $\boldsymbol{U}$ and $\boldsymbol{V}$ that need to be transmitted, leading to increased transmission time of the intermediate results. Therefore, the training times of the DW-ELM and IDW-ELM algorithms increase with the increasing number of training records. In addition, the training time of the IDW-ELM algorithm is less than that of the DW-ELM algorithm, which further verifies the validity of the improved algorithm.

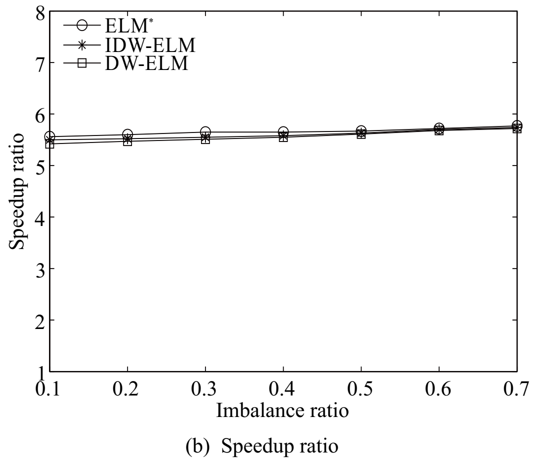
Next, we investigate the impact of the number of classes. As shown in Fig. 7a, along with the increasing number of classes, the training times of the ELM*[20] and DW-ELM algorithms are basically stable, while the training time of the IDW-ELM algorithm is slightly increased, and it is always less than the training time of the DW-ELM algorithm. The influence of changing the number of classes on speedup ratio is shown in Fig. 7b, and the speedup ratio remains stable. The number of classes increases, which only increases the number of statistical values in the first MapReduce job and subsequently the number of input values in the second MapReduce job of the DW-ELM algorithm, which has limited impact on the overall training time, so the training time is relatively stable. However, the increase in the number of classes makes the number of output intermediate results of the Mapper class in the IDW-ELM algorithm increase, and to a certain extent, this leads to the transmission time of the intermediate results to increase, so the training time increases slightly. In addition, the training time of the IDW-ELM algorithm is less than that of the DW-ELM algorithm, which means that although the effects of the improved algorithm decrease slightly with the increase in the number of classes, the optimization effect is still good.

Finally, we investigate the impact of imbalance ratio. As shown in Fig. 8a, with the increasing imbalance ratio, the training times of the ELM*[20], DW-ELM, and IDW-ELM algorithms are basically stable, and the training time of the IDW-ELM algorithm is always lower than that of the DW-ELM algorithm. Also, the training time of the ELM*[20] algorithm is lower than that of the IDW-ELM algorithm. The variation in speedup ratio of changing the imbalance ratio of the training records is shown in Fig. 8b, and the speedup ratio remains stable. Increasing the imbalance ratio did not produce any substantial effects on the calculation process of the MapReduce job, so the training time is relatively stable. Since the training time of the IDW-ELM algorithm is less than that of the DW-ELM algorithm, this further verifies the validity of the improved algorithm.

From the results of the six groups, the training time of the ELM*[20] algorithm is less than that of the DW-ELM algorithm in each group of experiments. The reason is because in the DW-ELM algorithm training process, the number of samples in each category needed to be first counted, that is, calculate the diagonal elements of the weight matrix $\boldsymbol{W}$ in the Map phase, then the $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{T}$ matrices were calculated in the MapReduce calculation; while only $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{T}$ matrices needed to be counted in the Map phase of the ELM*[20] algorithm. However, the G-mean value of the ELM*[20] algorithm in managing the unbalanced data was significantly lower than that of the DW-ELM algorithm, which apparently was not applicable to the imbalance BD learning.



(a) Running time



(b) Speedup ratio

Fig. 8 The impact of the imbalance ratio on (a) time and (b) speedup ratio.

5 Conclusion

Neither the WELM nor the DELM algorithms manage the imbalanced big training data efficiently since they only consider either the "big" or the "imbalanced" aspect of imbalanced big training data, and not both. In this study, we proposed a DW-ELM algorithm based on the MapReduce framework. The DW-ELM algorithm makes full use of the parallel computing ability of the MapReduce framework and realizes efficient learning of imbalanced BD. Specifically, through analyzing the characteristics of the WELM algorithm, we found that the matrix multiplication operators (i.e., $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{H}$ and $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{T}$) within the WELM algorithm are decomposable. Then, we transformed the corresponding matrix multiplication operators into summation forms, which suited the MapReduce framework well, and proposed a DW-ELM algorithm to calculate the matrix multiplications using two MapReduce jobs. Furthermore, an IDW-ELM algorithm, which only uses one MapReduce job to complete the tasks, was proposed to improve the performance of the DW-ELM algorithm. Finally, in the cluster environment, we performed a detailed validation of the performance of the DW-ELM and IDW-ELM algorithms with various experimental settings. The experimental results show that the DW-ELM and IDW-ELM algorithms can learn imbalanced BD efficiently.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China (Nos. 61402089, 61472069, and 61501101), the Fundamental Research Funds for the Central Universities (Nos. N161904001, N161602003, and N150408001), the Natural Science Foundation of Liaoning Province (No.

2015020553), the China Postdoctoral Science Foundation (No. 2016M591447), and the Postdoctoral Science Foundation of Northeastern University (No. 20160203).

References

[1] P. Langley, The changing science of machine learning, *Mach. Learn.*, vol. 82, no. 3, pp. 275–279, 2011.

[2] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, Extreme learning machine: Theory and applications, *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.

[3] G.-B. Huang, L. Chen, and C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, 2006.

[4] G.-B. Huang and L. Chen, Convex incremental extreme learning machine, *Neurocomputing*, vol. 70, nos. 16–18, pp. 3056–3062, 2007.

[5] G.-B. Huang and L. Chen, Enhanced random search based incremental extreme learning machine, *Neurocomputing*, vol. 71, nos. 16–18, pp. 3460–3468, 2008.

[6] G.-B. Huang, X. Ding, and H. Zhou, Optimization method based extreme learning machine for classification, *Neurocomputing*, vol. 74, nos. 1–3, pp. 155–163, 2010.

[7] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, Extreme learning machine for regression and multiclass classification, *IEEE Trans. Syst. Man Cybern. Part B-Cybern.*, vol. 42, no. 2, pp. 513–529, 2012.

[8] G.-B. Huang, D. Wang, and Y. Lan, Extreme learning machines: A survey, *Int. J. Mach. Learn. Cybern.*, vol. 2, no. 2, pp. 107–122, 2011.

[9] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, Evolutionary extreme learning machine, *Pattern Recognit.*, vol. 38, no. 10, pp. 1759–1763, 2005.

[10] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, and N. Sundararajan, On-line sequential extreme learning machine, in *Proc. of the IASTED Int. Conf. on Computational Intelligence*, Calgary, Canada, 2005, pp. 232–237.

[11] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, A fast and accurate on-line sequential learning algorithm for feedforward networks, *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, 2006.

[12] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, On-line sequential fuzzy extreme learning machine for function approximation and classification problems, *IEEE Trans. Syst. Man Cybern. Part B-Cybern.*, vol. 39, no. 4, pp. 1067–1072, 2009.

[13] H. He and E. A. Garcia, Learning from imbalanced data, *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.

[14] X. Liu, J. Wu, and Z. Zhou, Exploratory undersampling for class-imbalance learning, *IEEE Trans. Syst. Man Cybern. Part B-Cybern.*, vol. 39, no.2, pp. 539–550, 2006.

[15] H. Han, W.-Y. Wang, and B.-H. Mao, Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning, in *Proc. of Int. Conf. on Intelligent Computing*, Hefei, China, 2005, pp. 878–887.

[16] W. Zong, G.-B. Huang, and Y. Chen, Weighted extreme learning machine for imbalance learning, *Neurocomputing*, vol. 101, no. 3, pp. 229–242, 2013.

[17] M. Chen, S. Mao, and Y. Liu, Big data: A survey, *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[18] J. Chen, Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, and X. Zhou, Big data challenge: A data management perspective, *Front.. Comput. Sci.*, vol. 7, no. 2, pp. 157–164, 2013.

[19] Q. He, T. Shang, F. Zhuang, and Z. Shi, Parallel extreme learning machine for regression based on mapreduce, *Neurocomputing*, vol. 102, no. 2, pp. 52–58, 2013.

[20] J. Xin, Z. Wang, C. Chen, L. Ding, G. Wang, and Y. Zhao, ELM*: Distributed extreme learning machine with mapreduce, *World Wide Web*, vol. 17, no. 5, pp. 1189–1204, 2014.

[21] X. Bi, X. Zhao, G. Wang, P. Zhang, and C. Wang, Distributed extreme learning machine with kernels based on mapreduce, *Neurocomputing*, vol. 149, no. 1, pp. 456–463, 2015.

[22] J. Xin, Z. Wang, L. Qu, and G. Wang, Elastic extreme learning machine for big data classification, *Neurocomputing*, vol. 149, no. 1, pp. 464–471, 2015.

[23] J. Xin, Z. Wang, L. Qu, G. Yu, and Y. Kang, A-ELM*: Adaptive distributed extreme learning machine with MapReduce, *Neurocomputing*, vol. 174, no. 1, pp. 368–374, 2016.

[24] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, in *Proc. Symposium on Operating System Design and Implementation*, San Francisco, CA, USA, 2004, pp. 137–150.

[25] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[26] J. Dean and S. Ghemawat, MapReduce: A flexible data processing tool, *Commun. ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[27] C.-T. Chu, S.-K. Kim, Y.-A. Lin, Y.-Y. Yu, G. Bradski, A.-Y. Ng, and K. Olukotun, Map-reduce for machine learning on multicore, in *Proc. 20th Annual Conf. on Neural Information Processing Systems*, Vancouver, Canada, 2007, pp. 281–288.

[28] X. Meng and M. W. Mahoney, Robust regression on mapreduce, in *Proc. 30th Int. Conf. on Machine Learning*, Atlanta, GA, USA, 2013, pp. 888–896.

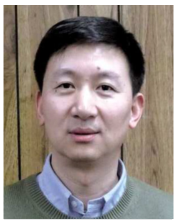
[29] R. Fletcher, Constrained optimization, in *Practical Methods of Optimization*. N. J. Hoboken, Ed. John Wiley & Sons Ltd, 1981, pp. 127–416.

[30] S. Ghemawat, H. Gobioff, and S.-T. Leung, The google file system, in *Proc. 19th ACM Symposium on Operating Systems Principles*, New York, UK, USA, 2003, pp. 29–43.

[31] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The hadoop distributed file system, in *Proc. 26th IEEE Symposium on Mass Storage Systems and Technologies*, Incline Village, NV, USA, 2010, pp. 1–10.

Zhiqiong Wang received the MSc degree and PhD degree in computer software and theory from the Northeastern University, China, in 2008 and 2014, respectively. She visited the National University of Singapore in 2010 and the Chinese University of Hong Kong in 2013 as the academic visitor. Currently, she is an associate professor with the Sino-Dutch Biomedical and Information Engineering School of Northeastern University. She served as a PI or co-PI for more than ten national research grants from NSFC, the Natural Science Foundation of Liaoning Province, etc. She has published more than 30 papers. Her main research interests are biomedical, biological data processing, cloud computing, and machine learning.

Junchang Xin received the BSc, MSc, and PhD degrees in computer science and technology from the Northeastern University, China, in 2002, 2005, and 2008, respectively. He visited National University of Singapore as Post-Doctoral Visitor (April 2010 – April 2011). He is currently an associate professor with the School of Computer Science and Engineering, Northeastern University, China. He served as a PI or co-PI for more than ten national research grants from NSFC, the 863 Program, Project 908 under the State Oceanic Administration (SOA), etc. He has published more than 60 research papers. His research interests include big data, uncertain data, and bioinformatics.

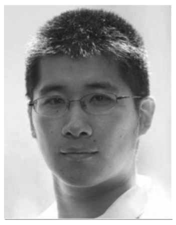


Yudong Yao received the BEng and MEng degrees from Nanjing University of Posts and Telecommunications, China, in 1982 and 1985, respectively, and the PhD degree from Southeast University, China, in 1988, all in electrical engineering. From 1989 and 1990, he was at Carleton University, Ottawa, Canada, as a research associate working on mobile radio communications. From 1990 to 1994, he was with Spar Aerospace Ltd., Montreal, Canada, where he was involved in research on satellite communications. From 1994 to 2000, he was with Qualcomm Inc., San Diego, CA, where he participated in research and development in wireless code-division multipleaccess (CDMA) systems. He has been with Stevens Institute of Technology, Hoboken, New Jersey, since 2000 and is currently a professor and department director of electrical and computer engineering. Dr. Yao was an associate editor of *IEEE Communications Letters* and *IEEE Transactions on Vehicular Technology*, and an editor for *IEEE Transactions on Wireless Communications*. He served as a PI or co-PI for multiple national research grants from NSF. He holds one Chinese patent and twelve U.S. patents and published more than 100 research papers. His research interests include wireless communications and networks, spread spectrum and CDMA, antenna arrays and beamforming, cognitive and software defined radio (CSDR), and digital signal processing for wireless systems. He is a NAI fellow, IEEE fellow, and IEEE ComSoc distinguished lecturer.

Hongxu Yang received the bachelor degree from Dalian Jiaotong University in 2014. Currently, she is a master student of the School of Computer Science and Engineering, Northeastern University, China. Her main research interests are big data management, machine learning, and bioinformatics.

Shuo Tian received the bachelor degree from Northeast Forestry University in 2013. Currently, she is a master student of the Sino-Dutch Biomedical and Information Engineering School of Northeastern University. Her main research interests are machine learning, digital image processing, and bioinformatics.

Ge Yu received the BE and ME degrees in computer science from Northeastern University of China in 1982 and 1986, respectively, PhD degree in computer science from Kyushu University of Japan in 1996. He has been a professor with Northeastern University of China since 1996. He served as a PI or co-PI for more than twenty national research grants from NSFC, the 863 Program, the 973 Program, etc. He has published more than 200 research papers. His research interesting includes database theory and technology, distributed and parallel systems, big data and cloud computing, cyber-physical systems. He is a CCF fellow, a member of IEEE and ACM.



Chenren Xu received the BE degree from Shanghai University in 2008, MS degree in applied mathematical statistics and PhD degree in electrical and computer engineering from the Rutgers University in 2014. He was a postdoctoral research fellow in the School of Computer Science, Carnegie Mellon University, from 2014 to 2015. He is currently an assistant professor in the School of Electronics Engineering and Computer Science and also a member of the Center for Energy-efficient Computing and Applications (CECA), Peking University. He was the receipt of Best Poster Award of ACM SenSys in 2011, Gold Medal of Samsung Best Paper Award Competition and Best Paper Award Nominee Award of ACM UbiComp in 2014. His current research interests are in wireless networks, mobile systems, and IoT health.