Memetic Computing (2019) 11:305–315
https://doi.org/10.1007/s12293-018-0271-8

REGULAR RESEARCH PAPER

A unified distributed ELM framework with supervised, semi-supervised and unsupervised big data learning

Zhiqiong Wang[1] · Luxuan Qu[1] · Junchang Xin[2] · Hongxu Yang[2] · Xiaosong Gao[1]

Received: 6 November 2017 / Accepted: 10 July 2018 / Published online: 13 July 2018

Abstract
Extreme learning machine (ELM) as well as its variants have been widely used in many fields for its good generalization performance and fast learning speed. Though distributed ELM can sufficiently process large-scale labeled training data, the current technology is not able to process partial labeled or unlabeled training data. Therefore, we propose a new unified distributed ELM with supervised, semi-supervised and unsupervised learning based on MapReduce framework, called U-DELM. The U-DELM method can be used to overcome the existing distributed ELM framework's lack of ability to process partially labeled and unlabeled training data. We first compare the computation formulas of supervised, semi-supervised and unsupervised learning methods and found that the majority of expensive computations are decomposable. Next, MapReduce framework based U-DELM is proposed, which extracts three different matrices continued multiplications from the three computational formulas introduced above. After that, we transform the cumulative sums respectively to make them suitable for MapReduce. Then, the combination of the three computational formulas are used to solve the output weight in three different learning methods. Finally, by using benchmark and synthetic datasets, we are able to test and verify the efficiency and effectiveness of U-DELM on learning massive data. Results prove that U-DELM can achieve unified distribution on supervised, semi-supervised and unsupervised learning.

Keywords Distributed ELM · Supervised learning · Semi-supervised learning · Unsupervised learning · MapReduce

1 Introduction

Internet of Things, Cloud Computing and Mobile Internet have developed rapidly in recent years, which have led to an explosive growth in the amount of information. This is the era of Big Data [3]. Data acquisition and storage technologies have been developing so quickly that it is very common to obtain large quantities of unlabeled data. Human interventions are the prerequisites of gaining labeled data, which have two obvious disadvantages: time consuming and error-prone. If partial labeled data and unlabeled data can be used for training, it will not have these disadvantages. Currently, big data analysis only supports massive labeled data training, and has not been able to support partial labeled or unlabeled training data [7,19,22–24]. Therefore, we reviewed the centralized and distributed semi-supervised and unsupervised methods on training massive partial labeled or unlabeled data to find a solution.

In recent years, extreme learning machine (ELM) [9] has drawn wide attention among researchers for its good generalization performance, fast learning speed and minimum human intervention. Centralized learning methods which include weighted learning W-ELM [26], semi-supervised learning SS-ELM [8], and unsupervised learning US-ELM [8] have been widely used in biomedical science [18,20,25], voice and image recognition [5], industrial control [21], and geographic information [14], and have seen fruitful achievements. However, analyzing massive data is beyond the capability and computation ability of a single computer. Distributed computing framework which is applied on skyline query processing [11], scalable subgraph enumeration [12], and parallel Genetic Algorithms [4] shows good performance on overcoming big data processing issue. So distributed ELM algorithms have been proposed, including

✉ Junchang Xin
xinjunchang@mail.neu.edu.cn

1 Sino-Dutch Biomedical and Information Engineering School, Northeastern University, Shenyang, China

2 School of Computer Science and Engineering, Key Laboratory of Big Data Management and Analytics (Liaoning Province), Northeastern University, Shenyang, China

the parallel training method PELM [7] and ELM* [22], data increment, decrement and correctional method E^{2}LM [23], weighted learning method DW-ELM [19], network hidden layers increment, decrement adaptive method A-ELM* [24].

However, the current distributed ELMs can only support supervised learning on labeled training datasets and unsupervised learning [16] on unlabeled training datasets, but not semi-supervised learning. There is a void on massive partial labeled data training. We are here to expand the distributed ELM on semi-supervised and unsupervised learning with MapReduce framework. We have found a notable similarity among all three ELM learning methods, that is the output weight computational formulas originate from the same matrix multiplication outcome, but with different combinations. Therefore, we propose a unified distributed extreme learning machine for supervised, semi-supervised and unsupervised learning, which has filled the void in distributed ELM's processing of massive partial labeled or unlabeled datasets, enhanced the extendibility of traditional distributed ELMs and solved the classification, regression and clustering of big data. The three contributions of this paper are as follows.

1. The output weight computational formulas of supervised, semi-supervised and unsupervised learning methods had been compared. From these formulas, three different types of matrix multiplication were extracted. Then, the cumulative sums were rewritten separately to make them suitable for MapReduce.
2. A unified distributed extreme learning machine (U-DELM) based on MapReduce framework with supervised, semi-supervised and unsupervised learning, which can fill the void to help distributed ELM process massive partial labeled or unlabeled datasets.
3. Finally, by using benchmark and synthetic datasets, we were able to test and verify the efficiency and effectiveness of U-DELM on learning massive data. Results have proved that U-DELM can achieve unified distribution on supervised, semi-supervised and unsupervised learning.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the traditional supervised, semi-supervised, and unsupervised ELM. Section 3 introduces the theoretical principles and algorithm implementation on U-DELM. Section 4 shows the experimental results and verifies the performance of U-DELM. Finally, Sect. 5 summarizes the entire paper.

2 Background

In this section, we give a brief overview of the traditional supervised, semi-supervised, and unsupervised ELM, and then we introduce the MapReduce framework.

2.1 Extreme learning machine

ELM has been proposed for training single layer feedforward networks (SLFNs), which has very fast learning speed, good generalization performance and general approximation ability [1]. ELM first randomly generates the parameters of hidden layers, including the input weights and the hidden layer biases, then directly calculates the output weight by matrix operations, so that the structure of the entire network is determined. ELM has avoided the issues in traditional neural network that it has to change the network parameter iteratively during network structure confirmation, until it reaches convergence. Compared with traditional neural networks, ELM has extremely short training time and minimum training errors. All the supervised, semi-supervised and unsupervised ELMs can be divided into two stages: (1) random feature mapping; (2) output weights solving.

2.1.1 Supervised ELM

For any given training datasets with N samples $\{\mathbf{X}, \mathbf{T}\} = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^{N}$, the optimization goal of ELM is to minimize the training errors, and to minimize the 2-norm of the output weight matrix, which maximizes the separate margins of two different classes in ELM feature space [26]. As a result, the optimization problem of ELM for supervised learning can be described as,

$$\begin{aligned} &Minimize\colon \tfrac{1}{2}\|\beta\|^2 + \tfrac{1}{2}\sum_{i=1}^{N} \mathbf{C}_i \|\mathbf{e}_i\|^2 \\ &Subject\ to\colon h(\mathbf{x}_i)\,\beta = \mathbf{t}_i^T - \mathbf{e}_i^T, \quad i = 1, \ldots, N, \end{aligned} \tag{1}$$

where $\mathbf{C}_i$ is the penalty coefficient corresponding to the ith training point, $\mathbf{e}_i$ is the error vector with respect to the ith training point.

The problem described above is a normalized least square problem. First, the constraint condition is substituted into the objective function, and then we solve the gradient of the objective function, afterwards, the gradient of the value is set as zero, and finally we get the output weight of the supervised ELM.

$$\beta = \left(\mathbf{H}^T\mathbf{C}\mathbf{H} + \mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{C}\mathbf{T} \tag{2}$$

when N is large or

$$\beta = \mathbf{H}^T\left(\mathbf{C}\mathbf{H}\mathbf{H}^T + \mathbf{I}\right)^{-1}\mathbf{C}\mathbf{T} \tag{3}$$

when N is small.

2.1.2 Semi-supervised ELM

The semi-supervised ELM learning originated from the Manifold Regularization Theory, which aims at maintaining the original ELM geometric properties of feature space for the training data in decision space [8]. In the semi-supervised training sets, there is a scarcity of labeled data and an abundance of unlabeled data. The labeled data as $\{\mathbf{X}_l, \mathbf{T}_l\} = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^{l}$, and unlabeled data as $\mathbf{X}_u = \{\mathbf{x}_i\}_{i=1}^{u}$, where l and u respectively represent the number of labeled data and unlabeled data. The formulation of semi-supervised ELM is given by,

$$\begin{aligned} Minimize\colon & \tfrac{1}{2}\|\beta\|^2 + \tfrac{1}{2}\sum_{i=1}^{l} \mathbf{C_i}\|\mathbf{e}_i\|^2 \\ & + \tfrac{\lambda}{2} Tr\left(\mathbf{F}^T\mathbf{L}\mathbf{F}\right) \\ Subject\ to\colon & h(\mathbf{x}_i)\,\beta = \mathbf{t}_i^T - \mathbf{e}_i^T, \quad i = 1,\ldots,l, \\ & f(\mathbf{x}_i) = h(\mathbf{x}_i)\,\beta, \quad i = 1,\ldots,l+u \end{aligned} \tag{4}$$

where $\mathbf{L} \in \mathbf{R}^{(l+u)\times(l+u)}$ is the Laplacian matrix built by both labeled and unlabeled data. $\mathbf{F} \in \mathbf{R}^{(l+u)\times m}$ is the ELM output with its ith row equal to $f(\mathbf{x}_i)$, and $Tr(\cdot)$ denotes the trace of a matrix based on manifold regularization framework. λ is a tradeoff parameter.

Similarly, the output weight of the semi-supervised ELM can be represented as,

$$\beta = \left(\mathbf{I}_L + \mathbf{H}^T\mathbf{C}\mathbf{H} + \lambda\mathbf{H}^T\mathbf{L}\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{C}\mathbf{T} \tag{5}$$

when N is large or

$$\beta = \mathbf{H}^T\left(\mathbf{I}_{l+u} + \mathbf{C}\mathbf{H}\mathbf{H}^T + \lambda\mathbf{L}\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{C}\mathbf{T} \tag{6}$$

when N is small.

2.1.3 Unsupervised ELM

The primary task of unsupervised learning is to map the training data from the input space into the ELM feature space, then cluster them in the new projection space using k-means algorithm [8]. In an unsupervised dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$, all training data are unlabeled, the purpose of training is to find the potential structure for the original data. The formulation of unsupervised ELM is given by.

$$\begin{aligned} Minimize\colon & \|\beta\|^2 + \lambda Tr\left(\beta^T\mathbf{H}^T\mathbf{L}\mathbf{H}\beta\right) \\ Subject\ to\colon & (\mathbf{H}\beta)^T\mathbf{H}\beta = \mathbf{I}_m. \end{aligned} \tag{7}$$

The above optimization problem is based on Rayleigh–Ritz theory which exists to resolve the eigenvalue and corresponding eigenvector in the following generalized eigenvalue problem.

$$\left(\mathbf{I}_L + \lambda\mathbf{H}^T\mathbf{L}\mathbf{H}\right)v = \gamma\mathbf{H}^T\mathbf{H}v. \tag{8}$$

First, we find $m+1$ generalized eigenvectors, which corresponded to $m+1$ smallest eigenvalues. After that, Laplacian eigenmaps algorithm has abandoned the first eigenvector, then the 2nd eigenvector to the $m+1$ eigenvectors are used for calculating the output weight, thus, the output weight is,

$$\beta = \left[\widetilde{v}_2, \widetilde{v}_3, \ldots, \widetilde{v}_{m+1}\right] \tag{9}$$

$\widetilde{v}_i = v_i \big/ \|\mathbf{H}v_i\|, \ i = 2,\ldots,m+1$ is the standard eigenvector. If the amount of training data is smaller than the amount of hidden neurons, the following generalized eigenvalue problem can be solved as

$$(\mathbf{I}_u + \lambda\mathbf{L}\mathbf{H}\mathbf{H}^T)u = \gamma\mathbf{H}\mathbf{H}^T u. \tag{10}$$

Similarly, the output weight is,

$$\beta = \mathbf{H}^T\left[\widetilde{u}_2, \widetilde{u}_3, \ldots, \widetilde{u}_{m+1}\right] \tag{11}$$

where $\widetilde{u}_i = u_i \big/ \left\|\mathbf{H}\mathbf{H}^T u_i\right\|, \ i = 2,\ldots,m+1.$

2.2 MapReduce framework

The MapReduce framework is first proposed by Google, which has been used for distributed computation [2]. So far, MapReduce is used to conduct big data issue in various research fields [6,15]. A significant open source implementation of MapReduce is Hadoop [10]. The main idea of MapReduce is to encourage users to only care about data processing and ignore the detailed problems in execution. MapReduce provides two original functions to execute the tasks of distributed computation, namely map and reduce. The map function takes key/value pairs (k_1, v_1) as input and creates temporary key/value pairs $[(k_2, v_2)]$, the reduce function receives key/value pairs $(k_2, [v_2])$, and merges them with the same keys, which creates the key/value pairs $[(k_3, v_3)]$ that contain the same keys as input. By using these two functions, researchers can accomplish the required tasks or programs. When MapReduce is in operation, it will automatically distribute and execute tasks with no additional human input. Thus, the complexity of parallel programming is greatly reduced. Non-professional programmers can also achieve their desired parallel operations easily by using the pointed map and reduce functions. MapReduce automatically takes care of task scheduling, data distributing, loading balancing, and has fault tolerance capability.

The execution procedure of MapReduce can be divided into the following phases: first, the input data is separated into several data blocks stored in the Hadoop Distributed File System (HDFS) [17] at the bottom layer. These data are input into one MapReduce task (Phase I: Input). Then, the

Table 1 Output weight of ELM

Variety	Output weight
Supervised ELM	$\beta = \left(\mathbf{H}^T\mathbf{CH} + \mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{CT}$
Semi-supervised ELM	$\beta = \left(\mathbf{I}_L + \mathbf{H}^T\mathbf{CH} + \lambda\mathbf{H}^T\mathbf{LH}\right)^{-1}\mathbf{H}^T\mathbf{CT}$
Unsupervised ELM	$\left(\mathbf{I}_L + \lambda\mathbf{H}^T\mathbf{LH}\right)v = \gamma\mathbf{H}^T\mathbf{H}v \Rightarrow \beta = \left[\widetilde{v}_2, \widetilde{v}_3, \ldots, \widetilde{v}_{m+1}\right]$

MapReduce tasks are divided into a certain amount of map tasks, each of them takes charge of a logic data block. Every map task reads the data block and processes according to the pointed map function given by the user, then outputs the result. The data in between is stored, segmented, and prepared for the reduce stage (Phase II: Map). Next, we use the data in between, to continue the corresponding storage (Phase III: Shuffle). Afterwards, the data is passed on to the reduce task to create the corresponding output (Phase IV: Reduce). Finally, the output is written back to the HDFS (Phase V: Output).

3 Unified distributed ELM

Section 3.1 introduces the theory supporting distributed unification on the three ELM learning methods; Sect. 3.2 talks about the matrix multiplier parallel calculation method of $\mathbf{Q}$; Sect. 3.3 describes the implementation of the entire U-DELM method.

3.1 Preliminaries

All the three traditional ELM learning methods cannot avoid an important problem: when training data amount is larger than the capability of one single computer, there would be a challenge in calculating the output matrix $\mathbf{H}$ of the hidden layer and the output weight β. Therefore, it is very important to conduct parallel computation, and we have accomplished this task using the MapReduce framework. By doing so, while learning massive data, the training data amount is always much larger than the feature dimensions, which means $N \gg L$. Since $N \gg L$, for supervised ELM, the dimension of matrix $\mathbf{H}^T\mathbf{H}$ is way smaller than the dimension of matrix $\mathbf{HH}^T$. As a result, we can use Eq. (2) to calculate the output weight. Similarly, for semi-supervised or unsupervised ELM output weight calculation, $\mathbf{L}$ is a matrix of $N \times N$ dimension, so the matrix dimension $\mathbf{H}^T\mathbf{LH}$ is way smaller than $\mathbf{LHH}^T$. Therefore, Eqs. (5) and (8) can be used separately to calculate the output weight of semi-supervised and unsupervised ELM. Below, we use Table 1 to sum up the description above.

As it shows in the table, while computing the output weight β of the supervised, unsupervised and semi-supervised learning methods, the related matrix multiplication factors $\mathbf{U} = \mathbf{H}^T\mathbf{CH}$, $\mathbf{V} = \mathbf{H}^T\mathbf{CT}$, $\mathbf{P} = \mathbf{H}^T\mathbf{H}$ and $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$ are repeated in calculation. Therefore, we propose a separate distributed computation method using $\mathbf{U} = \mathbf{H}^T\mathbf{CH}$, $\mathbf{V} = \mathbf{H}^T\mathbf{CT}$, $\mathbf{P} = \mathbf{H}^T\mathbf{H}$ and $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$. We first detect whether the data is supervised, semi-supervised or unsupervised, then use the corresponding multiplication factors $\mathbf{U}$, $\mathbf{V}$, $\mathbf{P}$ and $\mathbf{Q}$ to compute the output weight β. Using these factors, we achieve the unified processing of supervised, unsupervised and semi-supervised learning.

Based on the above analysis, we have two discoveries:

– Through the entire computation progress of ELM's three learning methods, the most expensive computation is solving the output weight β of matrix multiplication operator $\mathbf{U} = \mathbf{H}^T\mathbf{CH}$, $\mathbf{V} = \mathbf{H}^T\mathbf{CT}$, $\mathbf{P} = \mathbf{H}^T\mathbf{H}$ and $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$. Since the matrix multiplication is decomposable, it forms the required condition for distributed computation.
– The three ELM learning methods can compute the output weight of different combinations of $\mathbf{U}$, $\mathbf{V}$, $\mathbf{P}$ and $\mathbf{Q}$, indicating that the three learning methods can select matrix multiplication operators according to their needs to achieve the unification.

Furthermore, references [19,22] respectively describes the decomposition of matrix $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{P}$. Matrix $\mathbf{U}$ and $\mathbf{V}$ can be decomposed according to reference [19], and matrix $\mathbf{P}$ can be decomposed according to reference [22].

Therefore, we can prove that $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$ is decomposable, and it can be transformed according to the laws of matrix multiplication.

$$\mathbf{Q} = \mathbf{H}^T\mathbf{LH} = \left(\mathbf{L}^T\mathbf{H}\right)^T\mathbf{H} = \mathbf{R}^T\mathbf{H}. \tag{12}$$

Since matrix $\mathbf{L}$ in Eq. (12) is not a diagonal matrix whereas $\mathbf{C}$ in matrix $\mathbf{U}$ is diagonal. Therefore, it is not possible to decompose the matrix $\mathbf{Q}$ by the proposed method in reference [19]. Equation (12) first calculates $\mathbf{L}^T\mathbf{H}$, then multiplies the result with matrix $\mathbf{H}$. The answer is defined as matrix $\mathbf{Q}$. Therefore, the computation of matrix $\mathbf{Q}$ can be separated as,

1. $\mathbf{R} = \mathbf{L}^T\mathbf{H}$
2. $\mathbf{Q} = \mathbf{R}^T\mathbf{H}$.

These two steps are both the transposition of the left matrix multiplied with the right matrix $\mathbf{H}$. Matrix $\mathbf{A}$ was used to represent the left matrix, each step can be abstracted as $\mathbf{A}^T\mathbf{H}$. As a result, the decomposition of matrix $\mathbf{Q}$ equaled to matrix $\mathbf{A}^T\mathbf{H}$. But $\mathbf{A}^T\mathbf{H}$ is not the transposition of matrix multiplied with itself. Thus, we cannot use the proposed method in reference [22] which calculates $\mathbf{H}^T\mathbf{H}$ to compute the matrix. Now that only one matrix is involved, we can clearly see it is easier to allocate data in MapReduce using $\mathbf{H}^T\mathbf{H}$ than $\mathbf{A}^T\mathbf{H}$. Therefore, the decomposition of matrix $\mathbf{A}^T\mathbf{H}$ is more difficult than $\mathbf{H}^T\mathbf{H}$ and we need to find a more suitable decomposition method to calculate matrix $\mathbf{A}^T\mathbf{H}$.

The decomposition of matrix $\mathbf{A}^T\mathbf{H}$ is shown in Eq. (13).

$$\begin{aligned}
\mathbf{A}^T\mathbf{H} &= \begin{bmatrix} a(\mathbf{x}_1) \\ a(\mathbf{x}_2) \\ \vdots \\ a(\mathbf{x}_N) \end{bmatrix}^T \begin{bmatrix} h(\mathbf{x}_1) \\ h(\mathbf{x}_2) \\ \vdots \\ h(\mathbf{x}_N) \end{bmatrix} \\
&= \begin{bmatrix} a(\mathbf{x}_1)^T & a(\mathbf{x}_2)^T & \cdots & a(\mathbf{x}_N)^T \end{bmatrix} \begin{bmatrix} h(\mathbf{x}_1) \\ h(\mathbf{x}_2) \\ \vdots \\ h(\mathbf{x}_N) \end{bmatrix} \\
&= a(\mathbf{x}_1)^T h(\mathbf{x}_1) + a(\mathbf{x}_2)^T h(\mathbf{x}_2) \\
&\quad + \cdots + a(\mathbf{x}_N)^T h(\mathbf{x}_N) \\
&= \sum_{k=1}^{N} a(\mathbf{x}_k)^T h(\mathbf{x}_k) \\
&= \sum_{k=1}^{N} \begin{bmatrix} a_{k1} \\ a_{k2} \\ \vdots \\ a_{kN} \end{bmatrix} \\
&\quad \times \begin{bmatrix} g(\mathbf{w}_1\mathbf{x}_k + b_1) & g(\mathbf{w}_2\mathbf{x}_k + b_2) & \cdots & g(\mathbf{w}_L\mathbf{x}_k + b_L) \end{bmatrix} \\
&= \sum_{k=1}^{N} a(\mathbf{x}_{kj})\, g(\mathbf{w}_i\mathbf{x}_k + b_i). \qquad (13)
\end{aligned}$$

According to Eq. (13),

$$\mathbf{Q} = \sum_{k=1}^{N}\left[\left(\sum_{k=1}^{N} l(\mathbf{x}_k)^T h(\mathbf{x}_k)\right)^T h(\mathbf{x}_k)\right]. \qquad (14)$$

Furthermore,

$$\mathbf{Q}_{ij} = \sum_{k=1}^{N}\left[\left(\sum_{k=1}^{N} l(\mathbf{x}_{kj})\, g(\mathbf{w}_i\mathbf{x}_k + b_i)\right) g(\mathbf{w}_i\mathbf{x}_k + b_i)\right]. \qquad (15)$$

According to the equations above, every element $l(\mathbf{x}_{kj})$ of the kth row in matrix $\mathbf{L}$ is the kth neighbor which correlates with training data $\mathbf{x}_k$. $g(\mathbf{w}_i\mathbf{x}_k + b_i)$ referred as the ith element of kth row in hidden layer output matrix $\mathbf{H}$, which is computed by input training data $\mathbf{x}_k$. Both matrix correlate with the same group of input training data $\mathbf{x}_k$, and have no relation with other groups of training data.

3.2 Distributed matrix multiplication on MapReduce

From the analysis above, the computation of matrix $\mathbf{Q}$ can be achieved through MapReduce framework. Here is a detailed description of accomplishing distributed computation in matrix $\mathbf{Q}$.

3.2.1 Distributed computing of Q

The distributed computation of matrix $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$ can be accomplished using four rounds of MapReduce, which is shown in Algorithm 1.

Algorithm 1 MR jobs of **Q**

```
1 class L&H
2   method map(rowID, values)
3     if filename is fileL
4       context.write(rowID, L#values)
5     else if filename is fileH
6       context.write(rowID, H#values)
7   method reduce(rowID, values)
8     for all values do
9       context.write(rowID, L&H)
10 class $L^T H$
11   method map(rowID, L&H)
12     (l, h) =Parse(L&H)
13     $(j, v)$ =Parse(l)
14     for $i = 0$ to l.size() do
15       context.write($j, v_i \cdot \mathbf{h}$)
16   method reduce(columnID, list[values])
17     sum=0
18     for all value $\in$ list[values] do
19       sum=sum+value
20       context.write(columnID, sum)
21 class $L^T H$&H
22   $\mathbf{R} = \mathbf{L}^T\mathbf{H}$
23   method map(rowID, values)
24     if filename is fileR
25       context.write(rowID, R#values)
26     else if filename is fileH
27       context.write(rowID, H#values)
28   method reduce(rowID, values)
29     for all values do
30       context.write(rowID, R&H)
31 class $R^T H$
32   method map(rowID, R&H)
33     (r, h) =Parse(R&H)
34     for $i = 0$ to $r$.size() do
35       context.write ($i, r_i \cdot \mathbf{h}$)
36   method reduce($i$, list[values])
37     sum=0
38     for all value $\in$ list[values] do
39       sum=sum+value
40       context.write($i$, sum)
```

Algorithm 1 describes the four jobs of MapReduce, among which job1 (lines 1–9) and job3 (lines 21–30) are mainly used for operation joint, job2 (lines 10–20) and job4 (lines 31–40) are mainly used for matrix multiplication operations. The algorithm logic of job1 and job3 are exactly the same. There are two component methods, namely map method and reduce method. Map method can read both fileL and fileH at the same time. The line numbers in the two files are used as keys and the data of the line are used as values. In order to distinguish the key/value pairs from different sources, every piece of data is marked with a label tag, if tag=L, then it came from fileL; if tag=H, then it came from fileH. The major task for map stage is to tag data from different files. Reduce method receives the same value list from file fileL and fileH, then for the same key, the Cartesian product of the data from fileL and fileH are calculated, which means real connection operation is conducted in the reduce stage. The purpose of the job is to ensure the input training data $\mathbf{x}_k$ and l_k are related and h_k can be distributed in the same map.

Job2 mainly calculates the multiplication of $\mathbf{L}^T$ with $\mathbf{H}$. Both matrix $\mathbf{L}$ and $\mathbf{H}$ are stored on HDFS by row, the column of matrix $\mathbf{L}$ is the row of matrix $\mathbf{L}^T$. Thus the matrix exterior product method can be used to multiply the columns of the left matrix with the rows of the right matrix, which forms a partial matrix sharing the same rows and column as the outcome matrix. Finally, the partial matrixes are added up together to form the final matrix multiplication result. The matrix exterior product method can reduce the amount of transmit data on the shuffle stage of job compared with matrix inner product method. The algorithm contains two methods. Map method first connects the outcome of matrices $\mathbf{L}$ and $\mathbf{H}$ which we utilize in the reduce stage of job1. It has been resolved as one line l_k of Laplacian matrix $\mathbf{L}$ and one line h_k of hidden layer output matrix $\mathbf{H}$ (line 12). Since matrix $\mathbf{L}$ is not only sparse matrix but also symmetric matrix, the non-zero elements of l_k are separated as the pairs of (j, v). Among them, j is the column number where the matrix elements are located; v is the value of the element (line 13). Finally, the value of $v_i \times h_k$ corresponding to each j is calculated as the value output of map method, line number j being the key output of map method (line 15). In reduce method, different Mappers are combined with same key value and thus the final cumulated the sum of the key value corresponding element.

The map method of job3 receives the output from the reduce stage of job2 and the initial input matrix $\mathbf{H}$, which conducts the connection operation according to the method of job1. Job4 mainly multiplies the transposition of $\mathbf{L}^T\mathbf{H}$ output matrix and matrix $\mathbf{H}$. The algorithm logic are similar to job2, when the four rounds of MapReduce tasks are finished, the final result is stored in HDFS.

3.2.2 Improved distributed computing of Q

To reduce the computation and transmit cost of the entire computation, job2 and job3 in the Algorithm 1 are combined together, which forms the improved MapReduce algorithm. By using these 3 jobs we can compute the matrix $\mathbf{Q} = \mathbf{H}^T\mathbf{L}\mathbf{H}$. Algorithm 2 execution is as follows.

Algorithm 2 Improved MR jobs of **Q**

```
1  class L^T H&H
2    method map(rowID, L&H)
3      (l, h) =Parse(L&H)
4      (j, v) =Parse(l)
5      for i = 0 to l.size() do
6        context.write(j, R#v_i · h)
7        context.write(rowID, H#h)
8    method reduce(columnID, list[values])
9      sum=0
10     if value is startwith R
11       for all value ∈ list[values] do
12         sum=sum+value
13         linkR.add(sum)
14     else if value is startwith H
15       linkH.add(value)
16     for (r : linkR)
17       for (h : linkH)
18         context.write(columnID, r&h)
```

In Algorithm 2, there are three jobs of MapReduce. The job1 and job3 are consistent with job1 and job4 in Algorithm 1, the key difference of the two algorithms is in job2 of the Algorithm 2. In map stage of job2, we first receive the outputs from job1 reduce stage, which is the joint result of matrix $\mathbf{L}$ and $\mathbf{H}$. Then came resolving, multiplication, tagging and exporting of the output (lines 3–7). In the reduce stage, if the data came from matrix $\mathbf{H}^T\mathbf{L}$, then sum up is conducted first, and then the results are put into linked list linkR (lines 10–13). If the data is from matrix $\mathbf{H}$, then the data are put directly into linked list linkH (lines 14, 15). Afterwards, the two linked list were combined together and the combination result is passed into the next job task as the output of reduce stage.

3.3 A unified distributed ELM framework

Unified distributed ELM framework with supervised, semi-supervised and unsupervised learning is described in Algorithm 3. First, randomly generate L pairs of hidden layer nodes parameters $(\mathbf{w}_i, b_i)$ (lines 1, 2); then the training data undergoes processing and is distributed into their own categories. $\mathbf{X}_1 = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^N$ is labeled training dataset, $\mathbf{X}_2 = \{x_i\}_{i=1}^N$ is unlabeled training dataset. If the training dataset $\mathbf{X}_1$ was nonempty set and $\mathbf{X}_2$ is empty set, which indicates that the training data are all labeled, then calculate matrix $\mathbf{U}$ and $\mathbf{V}$. Equation (2) is used to calculate output weight (lines

3–6). If the training dataset $\mathbf{X}_1$ and $\mathbf{X}_2$ are all nonempty set, which indicates that some of the input training data has labels the others are still unlabeled, then first separately calculate matrix $\mathbf{U}$, $\mathbf{V}$ and use Algorithm 1 to calculate matrix $\mathbf{Q}$. After that, Eq. (5) calculates the output weight (lines 7–10). If the training dataset $\mathbf{X}_1$ is an empty set and $\mathbf{X}_2$ is nonempty set, which indicates that all training data are unlabeled, then calculate matrix $\mathbf{P}$ and use Algorithm 1 to calculate matrix $\mathbf{Q}$ separately. The result is put into Eq. (8), to get the eigenvector and the output weight (lines 11–14). Since the most expensive computation is accomplished using the MapReduce framework, the result is a relative small matrix. As a result, the computation of output weight β on line 6, 10 and 14 can be completed in the centralized environment. The unified distributed ELM framework with supervised, semi-supervised and unsupervised learning is finished as the training process is finished.

Algorithm 3 U-DELM

```
1 Input: Training data $\mathbf{X}_1 = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^N$, $\mathbf{X}_2 = \{x_i\}_{i=1}^N$
2 Output: Output weight vector $\beta$
3  for $i = 1$ to $L$ do
4    Randomly generate hidden node parameters $(\mathbf{w}_i, b_i)$
5    if $\mathbf{X}_1 \neq \emptyset$ then
6      if $\mathbf{X}_2 = \emptyset$ then
7        calculate $\mathbf{U} = \mathbf{H}^T\mathbf{CH}$, $\mathbf{V} = \mathbf{H}^T\mathbf{CT}$
8        calculate the output weight vector
         $\beta = (\mathbf{U} + \mathbf{I})^{-1}\mathbf{V}$
9      else
10       calculate $\mathbf{U} = \mathbf{H}^T\mathbf{CH}$, $\mathbf{V} = \mathbf{H}^T\mathbf{CT}$
11       calculate $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$ using Algorithm 1
12       calculate the output weight vector
         $\beta = (\mathbf{I}_L + \mathbf{U} + \lambda\mathbf{Q})^{-1}\mathbf{V}$
13   else
14     calculate $\mathbf{P} = \mathbf{H}^T\mathbf{H}$
15     calculate $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$ using Algorithm 1
16     substitute $(\mathbf{I}_L + \lambda\mathbf{Q})\, v = \gamma\mathbf{P}v$
       calculate $\beta = [\widetilde{v}_2, \widetilde{v}_3, \ldots, \widetilde{v}_{m+1}]$
```

4 Results

In this section, the performance of U-DELM is evaluated. First, we introduce the platform used in the experiments, then, we give an analysis and evaluation of the experiment results. Section 4.1 shows the experimental platform and Sect. 4.2 shows the experimental results.

4.1 Experimental platform

The experiments conducted in the paper were all based on Hadoop cluster server which was connected by 9 computers with up to Gigabit speeds. The configuration of the cluster server was Intel Quad Core 2.66 GHz CPU, 4 GB memory with CentOS Linux 5.6 system. Among them, one computer was set as the Master node, the others as Slave nodes. The cloud computation environment was built on Hadoop1.0.4. The semi-supervised and unsupervised benchmark data used in the experiments came from the experimental database from reference [8]. The big data experiment section used synthetic data and the experimental platform described above to test the performance of U-DELM in processing big data.

There are three explanations for experiment set up they are as follows,

- Since the successful learning of supervised data has already been proven on ELM* [22], this paper only focuses on the test and verification of U-DELM algorithm based on semi-supervised and unsupervised datasets. The MapReduce framework for matrix $\mathbf{Q}$ computation which uses Algorithm 1 is called U-DELM-Naive, while using Algorithm 2 for matrix $\mathbf{Q}$ computation is called U-DELM-Improved. Thus, U-DELM-Naive and U-DELM-Improved will be used as names in the following experiments.
- Both U-DELM and traditional ELM algorithms have used the same model for calculating the output weight. The only difference is U-DELM uses MapReduce framework for computation of matrix $\mathbf{U}$, $\mathbf{V}$, $\mathbf{P}$ and $\mathbf{Q}$, while traditional ELM algorithms only uses single machine environment for computation. As a result, when all relative parameters of U-DELM and traditional ELM algorithms remained the same, same output weight is received. In other words, U-DELM will not change the traditional ELM training results, and it will not influence the accuracy of classifier and clustering. As a result, it is not necessary to compare the accuracy between the U-DELM method and traditional ELM algorithms.
- Since the distributed computation efficiency of matrix $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{P}$ has been proven in [19,22], the experiment only evaluated the computation efficiency and speedup ratio of matrix $\mathbf{Q} = \mathbf{H}^T\mathbf{LH}$. Here, matrix $\mathbf{L}$ originated from k nearest neighbor algorithm of training data [13], matrix $\mathbf{H}$ is the output of training data through ELM feature space.

Table 2 summarizes the parameters used in our experimental evaluation. In each experiment, we vary a single parameter while setting all others to their default values. And the speedup achieved by an m computer mega system is defined as,

$$\text{speedup (m)} = \frac{\text{computing time on 1 computer}}{\text{computing time on m computers}} \tag{16}$$

Table 2 Experimental Parameters

Parameter	Range	Default
Dimensionality	100, 200, 300, 400, 500	500
Number of hidden nodes	100, 200, 300, 400, 500	300
Number of samples ($\times 10^6$)	3, 4, 5, 6, 7	5
Number of slave nodes	1, 2, 3, 4, 5, 6, 7, 8	8

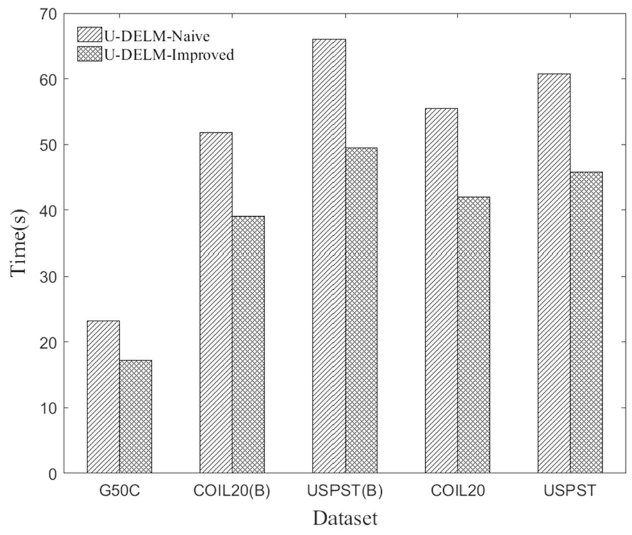

Fig. 1 Training time of SS-ELM benchmark datasets

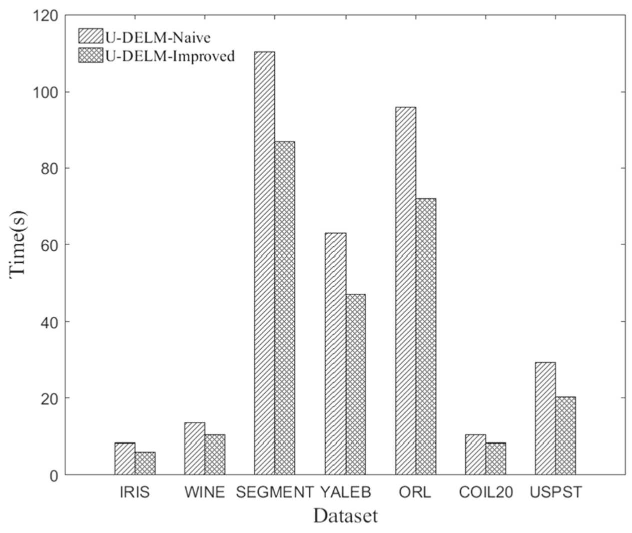

Fig. 2 Training time of US-ELM benchmark datasets

4.2 Experimental results

There were two type of datasets in the experiment. The performance of uniformed centralized and distributed ELM was examined with both of the benchmark datasets. Synthetic data was used to test the performance of U-DELM-Naive and U-DELM-Improved in processing bigdata. The experimental training time under uniformed semi-supervised ELM and unsupervised ELM benchmark datasets are shown respectively in Figs. 1 and 2. We did not show the speedup results because we only used one computer in the experiment. The experiment of uniformed semi-supervised ELM and unsupervised ELM which is centralized ELM is defined as unified ELM in the following paragraph.

Figure 1 shows 5 groups of experiments with the semi-supervised ELM benchmark datasets. As we can see in Fig. 1 the training time of U-DELM-Improved is always shorter than U-DELM-Naive. The execution time of unified ELM under SS-ELM benchmark database is around 0–2 s [8], while for U-DELM the result is around 17–66 s. Similarly, under unsupervised ELM benchmark datasets in Fig. 2, there were 7 groups of experiments conducted to examine the training time of U-DELM-Improved and U-DELM-Naive. The training time of U-DELM-Improved is also shorter than U-DELM-Naive. The execution time of unified ELM under SS-ELM benchmark datasets is around 0–4 s [8], while the results for U-DELM is around 6–110 s.

Comparing the development environment of unified ELM, which utilizes MATLAB against the development environments of unified distributed ELM, which utilizes JAVA. MATLAB is more suitable for computing matrix multiplication than JAVA. More importantly, U-DELM requires the communication and transformation of data between various nodes in the cluster while unified ELM do not. Thus, in the experiment of small data sample experiments, the time for communication and transformation of data between nodes is way longer than the time for just computation. Therefore, the training time of U-DELM is higher than unified ELM in training small sets of data.

From the results of benchmark datasets experiments, U-DELM does not have an obvious advantage. However, when the amount of data is enlarged to a much larger scale, the unified supervised, semi-supervised and unsupervised ELM will have to face the reality of overextended computation time, or overloaded data which is already beyond the capabilities of the unified ELM. The computation ability of U-DELM would not be affected under a larger dataset scale, And as the data amount increased, the performance of U-DELM will gradually turn into an obvious advantage. The experiment below summarizes the performance of U-DELM under larger dataset.

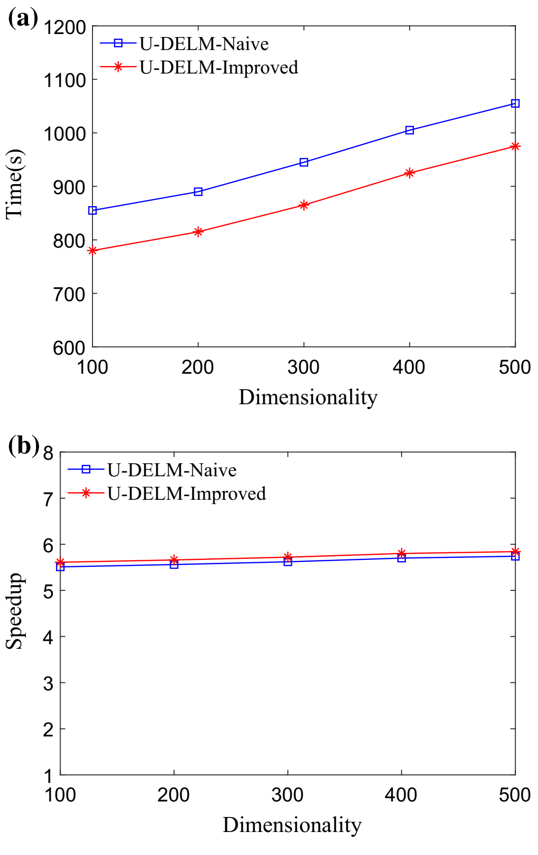

Fig. 3 The influence of dimensionality: **a** running time, **b** speedup

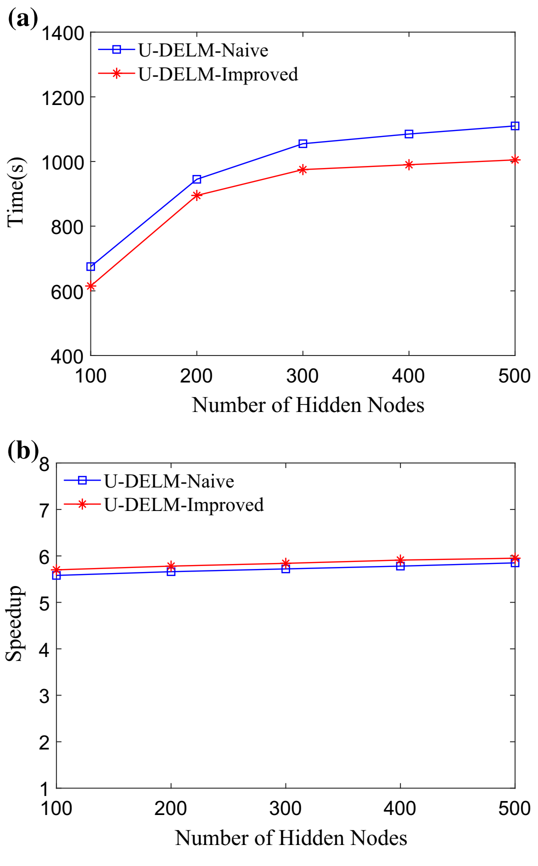

Fig. 4 The influence of number of hidden nodes: **a** running time, **b** speedup

First, we studied the training data dimensionality's influence on the running time of U-DELM shown in Fig. 3. As it is shown in Fig. 3a, as data dimensionality increases, the running time of U-DELM-Naive and U-DELM-Improved grows. We can also see U-DELM-Improved has a better performance than U-DELM-Naive. As data dimensionality changes, the speedup tendency of these two methods is given in Fig. 3b. Both methods have a stable speedup, the performance of U-DELM-Improved is slightly better than U-DELM-Naive.

As data dimensionality grows, there are more elements in the corresponding h_k line of hidden layer output matrix $\mathbf{H}$. The dimensionality of matrix $\mathbf{L}$ also grows, which increases the amount of computation needed for matrix multiplication of $\mathbf{H}^T\mathbf{LH}$. Due to these factors, the running time also increases. There were 4 MapReduce jobs in U-DELM-Naive, but only 3 jobs were used in U-DELM-Improved, which indicates that the data transmission time between machines has been shortened and the performance has been improved. Therefore, U-DELM-Improved has a better performance than U-DELM-Naive. As for speedup, theoretically speaking, a system of m computers equals to m speedup. But in reality, the increase of computers would only add additional communication cost between computers, thus it is very hard for the system to gain linear speedup.

Second, we studied the number of hidden nodes and their influence on the running time of U-DELM in Fig. 4. As it shown in Fig. 4a, as the number of hidden nodes increases, the running time of U-DELM-Naive and U-DELM-Improved grows. The running time of U-DELM-Improved is shorter than U-DELM-Naive. As the number of hidden nodes changes, the speedup stays constant as shown in Fig. 4b.

As the number of hidden nodes grows, the dimensionality of hidden layer output matrix $\mathbf{H}$ also increased. While the dimension of matrix $\mathbf{L}$ remained unchanged which causes the dimension of matrix $\mathbf{H}^T\mathbf{LH}$ to increase. Thus, the amount of in between results in MapReduce also increases, thus increasing the data transmission time inside the cluster server. As a result, as the number of hidden nodes increases, the running time also increases. While calculating matrix $\mathbf{H}^T\mathbf{LH}$, U-DELM-Improved saves one round of MapReduce job than U-DELM-Naive, thus has shorter running time.

Next, we studied the number of training records, influence on the running time of U-DELM in Fig. 5. As the number of training records increases, the increased running time is shown in Fig. 5a. U-DELM-Improved has a better per-

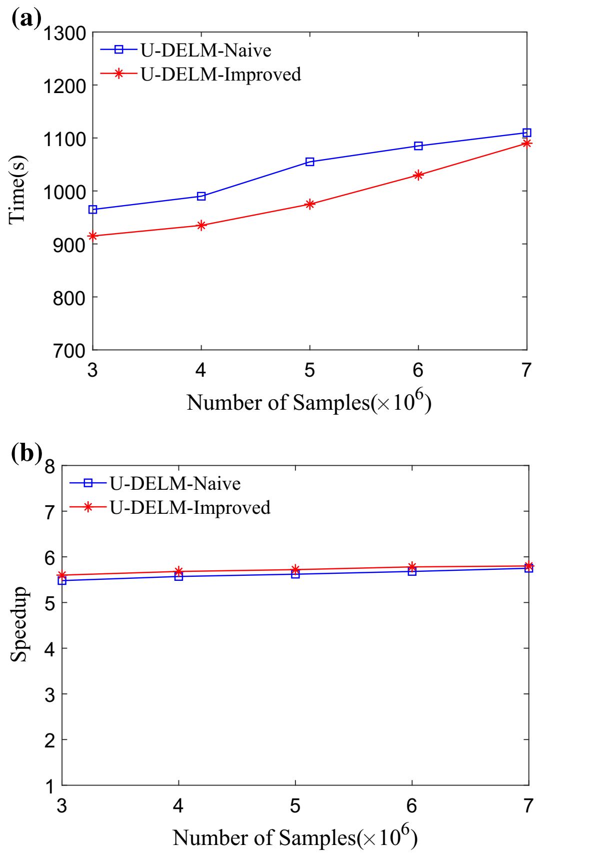

Fig. 5 The influence of number of samples: **a** running time, **b** speedup

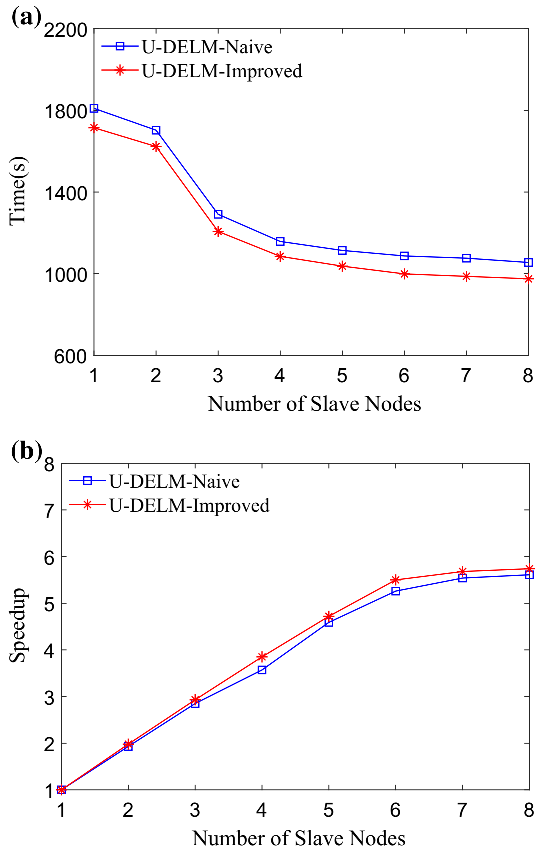

Fig. 6 The influence of number of slave nodes: **a** running time, **b** speedup

formance than U-DELM-Naive. As the number of training records changes, the speedup of these two methods is shown in Fig. 5b. Though the change was not linear, the speedup of U-DELM-Improved was slightly better than U-DELM-Naive.

As the number of training samples grow, the lines of matrix **H** and matrix **L** increased as well, which increases the amount of computation needed in the matrix multiplication of $\mathbf{H}^T\mathbf{LH}$. Thus, the amount of in between results in MapReduce increases and the running time also increases. Due to these factors, the training time also increases. U-DELM-Improved saves one round of MapReduce job than U-DELM-Naive, which decreases the data transmission time in between clusters, so U-DELM-Improved has a better performance than U-DELM-Naive.

Finally, there was a discussion in the number of working slave nodes in the cluster and their influence on the running time of U-DELM in Fig. 6. As it is shown in Fig. 6a, as the number of slave nodes increased, the running time decreases, so U-DELM-Improved has a better performance than U-DELM-Naive. In Fig. 6b, we can see the change in speedup when the number of slave nodes increases. The trend of speedup is close to linear growth.

As the number of slave nodes grows, the ability of conducting map and reduce tasks rapidly increases, thus enhancing the ability of MapReduce in conducting parallel computation and improving computation efficiency. As a result, the training time sharply decreases as the number of slave nodes increases.

To summarize, no matter the change to the experimental parameters, both U-DELM-Naive and U-DELM-Improved are able to complete the learning of massive data in a short period of time. And both U-DELM methods have good scalability, thus can effectively deal with the massive data learning problem and have a wide range of practical application.

5 Conclusions

In order to overcome the disadvantage that distributed ELM is unable to deal with massive partial labeled or unlabeled training data, a unified distributed ELM with supervised, semi-supervised and unsupervised learning method U-DELM is proposed. The U-DELM can support massive dataset supervised, semi-supervised and unsupervised learning at the same time. By using comparison and analysis of the characters of the solution for the output weight formula, we found the most expensive computations of the calculation process were caused by the different combination of same matrix multiplication outcome. First, we rewrote the matrix multiplication into a cumulative sum that was suitable for MapReduce. Next, by combining the results in between, we were able to reduce the number of jobs in the calculation process, which shortened the transition time in between and improved the training efficiency of U-DELM. Finally, the test results support that U-DELM can be used effectively in training massive datasets for supervised, semi-supervised and unsupervised learning.

Acknowledgements This research was partially supported by the following foundations: the National Natural Science Foundation of China under Grant Nos. 61472069, 61402089, and U1401256. The Fundamental Research Funds for the Central Universities under Grant Nos. N161602003, N171607010, N161904001, and N160601001. The Natural Science Foundation of Liaoning Province under Grant No. 2015020553.

References

1. Cheng X, Liu H, Xu X, Sun F (2017) Denoising deep extreme learning machine for sparse representation. Memet Comput 9(3):199–212
2. Dean J, Ghemawat S (2010) MapReduce: a flexible data processing tool. Commun ACM 53(1):72–77
3. Elsayed S, Sarker R (2016) Differential evolution framework for big data optimization. Memet Comput 8(1):17–33
4. Ferrucci F, Salza P, Sarro F (2017) Using hadoop MapReduce for parallel genetic algorithms: a comparison of the global, grid and island models. Evol Comput 1:421–446
5. Han M, Yang X, Jiang E (2016) An extreme learning machine based on cellular automata of edge detection for remote sensing images. Neurocomputing 198:27–34
6. Hashem IAT, Anuar NB, Gani A, Yaqoob I, Xia F, Khan SU (2016) MapReduce: review and open challenges. Scientometrics 109(1):389–422
7. He Q, Shang T, Zhuang F, Shi Z (2013) Parallel extreme learning machine for regression based on MapReduce. Neurocomputing 102:52–58
8. Huang G, Song S, Gupta J, Wu C (2014) Semi-supervised and unsupervised extreme learning machines. IEEE Trans Cybern 44(12):2405–2417
9. Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1–3):489–501
10. Huang S, Wang B, Chen Y, Wang G, Yu G (2017) An efficient parallel method for batched OS-ELM training using MapReduce. Memet Comput 9(3):183–197
11. Koh JL, Chen CC, Chan CY, Chen ALP (2017) MapReduce skyline query processing with partitioning and distributed dominance tests. Inf Sci 375:114–137
12. Lai L, Qin L, Lin X, Chang L (2017) Scalable subgraph enumeration in MapReduce: a cost-oriented approach. VLDB J 26(3):421–446
13. Lu W, Shen Y, Chen S, Ooi BC (2012) Efficient processing of k nearest neighbor joins using MapReduce. Proc VLDB Endow 5(10):1016–1027
14. Lu X, Zou H, Zhou H, Xie L, Huang GB (2016) Robust extreme learning machine with its application to indoor positioning. IEEE Trans Cybern 46(1):194–205
15. Park Y, Min JK, Shim K (2017) Efficient processing of skyline queries using MapReduce. IEEE Trans Knowl Data Eng 29(5):1031–1044
16. Rizk Y, Awad M (2015) On the distributed implementation of unsupervised extreme learning machines for big data. Proc Comput Sci 53(1):167–174
17. Shvachko K, Kuang H, Radia S, Chansler R (2010) The hadoop distributed file system. In: Proceedings of the 26th IEEE symposium on mass storage systems and technologies (MSST 2010). Incline Village, pp 1–10
18. Wang Z, Qu Q, Yu G, Kang Y (2016) Breast tumor detection in double views mammography based on extreme learning machine. Neural Comput Appl 27(1):227–240
19. Wang Z, Xin J, Yang H, Tian S, Yu G, Xu C, Yao Y (2017) Distributed and weighted extreme learning machine for imbalanced big data learning. Tsinghua Sci Technol 22(2):160–173
20. Wang Z, Yu G, Kang Y, Zhao Y, Qu Q (2014) Breast tumor detection in digital mammography based on extreme learning machine. Neurocomputing 128:175–184
21. Wong KI, Vong CM, Wong PK, Luo J (2015) Sparse Bayesian extreme learning machine and its application to biofuel engine performance prediction. Neurocomputing 149(Part A):397–404
22. Xin J, Wang Z, Chen C, Ding L, Wang G, Zhao Y (2013) ELM*: distributed extreme learning machine with MapReduce. World Wide Web 17(5):1189–1204
23. Xin J, Wang Z, Qu L, Wang G (2015) Elastic extreme learning machine for big data classification. Neurocomputing 149(Part A):464–471
24. Xin J, Wang Z, Qu L, Yu G, Kang Y (2016) A-ELM*: adaptive distributed extreme learning machine with MapReduce. Neurocomputing 174(Part A):368–374
25. Zhao Y, Wang G, Yin Y, Li Y, Wang Z (2016) Improving ELM-based microarray data classification by diversified sequence features selection. Neural Comput Appl 27(1):155–166
26. Zong W, Huang GB, Chen Y (2013) Weighted extreme learning machine for imbalance learning. Neurocomputing 101:229–242