Distributed Weighted Extreme Learning Machine for Big Imbalanced Data Learning

Zhiqiong Wang, Junchang Xin, Shuo Tian and Ge Yu

Abstract Extreme Learning Machine (ELM) and its variants have been widely used in many big data learning applications where raw data with imbalanced class distribution can be easily found. Although there have been several works solving the machine learning and robust regression problems using MapReduce framework, they need multi-iterative computations. Therefore, in this paper, we propose a novel Distributed Weighted Extreme Learning Machine based on MapReduce framework, named DWELM, which can learn the big imbalanced training data efficiently. Firstly, after indepth analyzing the properties of centralized Weighted ELM (WELM), it can be found out that the matrix multiplication operators in WELM are decomposable. Next, a DWELM based on MapReduce framework can be developed, which can first calculate the matrix multiplications effectively using two MapReduce Jobs in parallel, and then calculate the corresponding output weight vector with centralized computing. Finally, we conduct extensive experiments on synthetic data to verify the effectiveness and efficiency of our proposed DWELM in learning big imbalanced training data with various experimental settings.

Keywords Weighted ELM · Big imbalanced data · MapReduce framework · In-mapper combining

Z. Wang (✉) · S. Tian
Sino-Dutch Biomedical & Information Engineering School, Northeastern University, Shenyang, China
e-mail: wangzq@bmie.neu.edu.cn

S. Tian
e-mail: xinjunchang@ise.neu.edu.cn

J. Xin · G. Yu
College of Information Science & Engineering, Northeastern University, Shenyang, China
e-mail: dyhswdza@sina.com

G. Yu
e-mail: yuge@ise.neu.edu.cn

J. Cao et al. (eds.), *Proceedings of ELM-2015 Volume 1*,
Proceedings in Adaptation, Learning and Optimization 6,
DOI 10.1007/978-3-319-28397-5_25

1 Introduction

With the proliferation of mobile devices, artificial intelligence, web analytics, social media, internet of things, location based services and other types of emerging technologies, the amount of data, and the rate at which it's being accumulated, is rising exponentially. For examples, Facebook users share 2.5 billion unique pieces of content, hit the "like" button 2.7 billion times and upload 300 million photos a day. Thus, the era of big data has arrived [1, 2].

Extreme Learning Machine (ELM) [3–8] has recently attracted increasing attention from more and more researchers due to the characteristics of excellent generalization performance, rapid training speed and little human intervene [9]. ELM and its variants have been extensively used in many fields, such as text classification, image recognition, handwritten character recognition, mobile object management and bioinformatics [10–21].

Recently, as important variants of ELM, some Distributed ELM (DELM) [22–25] have been proposed to resolve the problem of big data learning, and a centralized Weighted ELM (WELM) [26] has been proposed to deal with data with imbalanced class distribution. However, neither DELM nor WELM can cope with big imbalanced training data efficiently since they only consider one aspect of big imbalanced data, though raw data with imbalanced class distribution can be found in many big data learning applications [26]. Therefore, in this paper, a Distributed Weighted Extreme Learning Machine (DWELM) which combines the advantages of both DELM and WELM based on distributed MapReduce framework [27–29] is proposed, to improve the scalability of centralized WELM and make it learn the big imbalanced data efficiently. The contributions of this paper are as follows.

- We prove theoretically that the matrix multiplication operators in centralized WELM are decomposable.
- A novel Distributed Weighted Extreme Learning Machine based on MapReduce framework (DWELM) is proposed to learn big imbalanced data efficiently.
- Last but not least, our extensive experimental studies using synthetic data show that our proposed DWELM can learn big imbalanced data efficiently, which can fulfill the requirements of many real-world big data applications.

The rest of the paper is organized as follows. Section 2 briefly reviews the background for our work. The theoretical foundation and the computational details of the proposed DWELM are introduced in Sect. 3. The experimental results to show the effectiveness of the proposed approaches are reported in Sect. 4. Finally, Sect. 5 concludes this paper.

2 Background

2.1 Weighted Extreme Learning Machine

ELM [3, 4] has been originally developed for single hidden-layer feedforward neural networks (SLFNs) and then extended to the "generalized" SLFNs where the hidden layer need not be neuron alike [5, 6]. ELM first randomly assigns the input weights and the hidden layer biases, and then analytically determines the output weights of SLFNs. It can achieve better generalization performance than other conventional learning algorithms at an extremely fast learning speed. Besides, ELM is less sensitive to user-specified parameters and can be deployed faster and more conveniently [7, 8]. Recently, a centralized Weighted ELM (WELM) [26] has been proposed to deal with data with imbalanced class distribution.

For N arbitrary distinct samples $(\mathbf{x}_j, \mathbf{t}_j)$, where $\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]^T \in \mathbb{R}^n$ and $\mathbf{t}_j = [t_{j1}, t_{j2}, \ldots, t_{jm}]^T \in \mathbb{R}^m$, standard SLFNs with L hidden nodes and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{L} \beta_i g_i(\mathbf{x}_j) = \sum_{i=1}^{L} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j \qquad (j = 1, 2, \ldots, N) \tag{1}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ is the weight vector connecting the ith hidden node and the input nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector connecting the ith hidden node and the output nodes, b_i is the threshold of the ith hidden node, and $\mathbf{o}_j = [o_{j1}, o_{j2}, \ldots, o_{jm}]^T$ is the jth output of the SLFNs [3].

The standard SLFNs with L hidden nodes and activation function $g(x)$ can approximate these N samples with zero error. It means $\sum_{j=1}^{L} ||\mathbf{o}_j - \mathbf{t}_j|| = 0$ and there exist β_i, $\mathbf{w}_i$ and b_i such that

$$\sum_{i=1}^{L} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j \qquad (j = 1, 2, \ldots, N) \tag{2}$$

The equation above can be expressed compactly as follows:

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

where $\mathbf{H}$ is called the hidden layer output matrix of the neural network and the ith column of $\mathbf{H}$ is the ith hidden node output with respect to inputs $\mathbf{x}_i, \mathbf{x}_2, \ldots, \mathbf{x}_N$.

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

$$\begin{aligned} &Minimize\ :\ \frac{1}{2}\|\beta\|^2 + C\mathbf{W}\frac{1}{2}\sum_{i=1}^{N}\left\|\xi_i\right\|^2 \\ &Subject\ to\ :\ \mathbf{h}(\mathbf{x}_i)\beta = \mathbf{t}_i^T - \xi_i^T \end{aligned} \tag{4}$$

where C is the regularization parameter to represent the trade-off between the minimization of weighted cumulative error and the maximization of the marginal distance. ξ_i, the training error of sample $\mathbf{x}_i$, is caused by the difference of the desired output $\mathbf{t}_i$ and the actual output $\mathbf{h}(\mathbf{x}_i)\beta$. $\mathbf{W}$ is a $N \times N$ diagonal matrix associated with every training sample $\mathbf{x}_i$, and

$$W_{ii} = 1/\#(\mathbf{t}_i) \tag{5}$$

or

$$W_{ii} = \begin{cases} 0.618/\#(\mathbf{t}_i) & \text{if } \#(\mathbf{t}_i) > AVG \\ 1/\#(\mathbf{t}_i) & \text{if } \#(\mathbf{t}_i) \leq AVG \end{cases} \tag{6}$$

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

According to Karush-Kuhn-Tucker (KKT) theorem [30], we have the following solutions for Weighted ELM (WELM):

$$\beta = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{W}\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{W}\mathbf{T} \tag{7}$$

when N is large or

$$\beta = \mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{W}\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{W}\mathbf{T} \tag{8}$$

when N is small.

2.2 MapReduce Framework

MapReduce is a simple and flexible parallel programming model initially proposed by Google for large scale data processing in a distributed computing environment [27–29], with one of its open source implementations Hadoop.[1] The typical procedure of a MR job is as follows: First, the input to a MR job starts as the dataset stored on the underlying distributed file system (e.g. GFS [31] and HDFS [32]), which is split into a number of files across machines. Next, the MR job is partitioned into many independent map tasks. Each map task processes a logical split of the input dataset. The map task reads the data and applies the user-defined map function on each record, and then buffers the resulting intermediate output. This intermediate data is sorted and partitioned for reduce phase, and written to the local disk of the machine executing the corresponding map task. After that, the intermediate data

[1] http://hadoop.apache.org/.

files from the already completed map tasks are fetched by the corresponding reduce task following the "pull" model (Similarly, the MR job is also partitioned into many independent reduce tasks). The intermediate data files from all the map tasks are sorted accordingly. Then, the sorted intermediate data is passed to the reduce task. The reduce task applies the user-defined reduce function to the intermediate data and generates the final output data. Finally, the output data from the reduce task is generally written back to the corresponding distributed file system.

3 Distributed Weighted Extreme Learning Machine

3.1 Preliminaries

In big imbalanced data learning applications, the number of training records is much larger than the dimensionality of the feature space, that is to say, $N \gg L$. According to $N \gg L$, the size of $\mathbf{H}^T\mathbf{WH}$ is much smaller than that of $\mathbf{WHH}^T$. Therefore, it is a better choice of using Eq. (7) to calculate the output weight vector β in WELM. Similar with ELM* [23], we analyze the properties of centralized WELM, and find the part that can be processed in parallel, and then transplant it into MapReduce framework. In this way, we can make WELM extend to the scale of big imbalance data efficiently. Let $\mathbf{U} = \mathbf{H}^T\mathbf{WH}$, $\mathbf{V} = \mathbf{H}^T\mathbf{WT}$, and we can get,

$$\beta = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{U}\right)^{-1}\mathbf{V} \tag{9}$$

According to the matrix multiplication operator, we have

$$\mathbf{U} = \mathbf{H}^T\mathbf{WH} = \sum_{k=1}^{N} h(\mathbf{x}_k)^T W_{kk} h(\mathbf{x}_k) \tag{10}$$

Then, we can further get,

$$u_{ij} = \sum_{k=1}^{N} W_{kk} \times g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) \times g(\mathbf{w}_j \cdot \mathbf{x}_k + b_j) \tag{11}$$

Similarly, according to the matrix multiplication operator, we also have

$$\mathbf{V} = \mathbf{H}^T\mathbf{WT} = \sum_{i=1}^{N} h(\mathbf{x}_k)^T W_{kk} \mathbf{t}_k \tag{12}$$

Then, we can further get,

$$v_{ij} = \sum_{k=1}^{N} W_{kk} \times g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) \times t_{kj} \tag{13}$$

According to Eq. (11), we know that the item u_{ij} in matrix $\mathbf{U}$ can be expressed by the summation of $W_{kk} \times g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) \times g(\mathbf{w}_j \cdot \mathbf{x}_k + b_j)$. Here, W_{kk} is the weight of training sample $(\mathbf{x}_k, \mathbf{t}_k)$, and $h_{ki} = g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i)$ and $h_{kj} = g(\mathbf{w}_j \cdot \mathbf{x}_k + b_j)$ are the ith and jth elements in the kth row $h(\mathbf{x}_k)$ of the hidden layer output matrix $\mathbf{H}$, respectively. Similarly, according to Eq. (13), we know that item v_{ij} in matrix $\mathbf{V}$ can be expressed by the summation of $W_{kk} \times g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) \times t_{kj}$. Here, W_{kk} is the weight of training sample $(\mathbf{x}_k, \mathbf{t}_k)$, $h_{ki} = g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i)$ is the ith element in the kth row $h(\mathbf{x}_k)$ of the hidden layer output matrix $\mathbf{H}$, and t_{kj} is the jth element in the kth row $\mathbf{t}_k$ of matrix $\mathbf{T}$ which related to $(\mathbf{x}_k, \mathbf{t}_k)$.

The variables involved in equations of matrices $\mathbf{U}$ and $\mathbf{V}$ include: W_{kk}, h_{ki}, h_{kj} and t_{kj}. According to Eqs. (5) and (6), to calculate the corresponding weight W_{kk} related to training sample $(\mathbf{x}_k, \mathbf{t}_k)$, we must first get the number $\#(\mathbf{t}_k)$ of training samples which belongs to the same class as $\mathbf{t}_k$. The numbers of training samples in all classes can be easily calculated in one MR job. At the same time, the remaining three variables h_{ki}, h_{kj} and t_{kj} only have relationship with training sample $(\mathbf{x}_k, \mathbf{t}_k)$ itself, and have nothing to do with the other training samples, so the calculation of matrices $\mathbf{U}$ and $\mathbf{V}$ can be done in another MR Job.

To sum up, the calculation process of matrices $\mathbf{U}$ and $\mathbf{V}$ is decomposable, therefore, similar to ELM* [23], we can realize the parallel computation of matrices $\mathbf{U}$ and $\mathbf{V}$ by using MapReduce framework, to break through the limitation of single machine, so as to improve the efficiency of which WELM learns big imbalanced training data.

3.2 DWELM

The process of DWELM is shown in Algorithm 1. Firstly, we randomly generate L pairs of hidden node parameters $(\mathbf{w}_i, b_i)$ (Lines 1–2). And then, using a MR Job to count the number of training samples contained in each class (Line 3). Next, using another MR Job to calculate matrices $\mathbf{U}$ and $\mathbf{V}$ according to the input parameters and randomly generate parameters (Line 4). Finally, we solve output weight vector β according to the Eq. 7 (Line 5).

Algorithm 1 DWELM

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

Here are the specific processes of two MR Jobs involved in DWELM:

The process of the 1st MR Job is shown in Algorithm 2. The algorithm includes two classes, Class Mapper (Lines 1–10) and Class Reducer (Lines 11–16). Class Mapper contains three methods, Initialize (Lines 2–3), Map (Lines 4–7) and Close (Line 8–10), while Class Reducer only contains one method, Reduce (Lines 12–16). In the Initialize method of Mapper, we initialize one array, c, which is used to store the intermediate summation of training samples contained in each class (Line 3). In the Map method of Mapper, firstly, we analyze the training sample s, and resolve the class which sample s belongs to (Lines 5–6). Then, adjust the corresponding value in the array c (Line 7). In the Close method of Mapper, the intermediate summations stored in c are emitted by the mapper (Lines 9–10). In the Reduce method of Reducer, firstly, we initialize a temporary variable *sum* (Line 13). And then, we combine the intermediate summations of different mappers which have the same Key, and furthermore, get the final summation of the corresponding element of the Key (Lines 14–15). Finally, we store the results into the distributed file system (Line 16).

Algorithm 2 The 1st MR Job of DWELM

```
class MAPPER
    method INITIALIZE()
        c = new ASSOCIATIVEARRAY
    method MAP(sid id, sample s)
        t =ParseT(s)
        num =Class(t)
        c[num] = c[num] + 1
    method CLOSE()
        for i = 1 to c.Length() do
            context.write(cid i, count c[i])
class REDUCER
    method REDUCE(cid id, counts [c_1, c_2, ... ])
        sum = 0
        for all count c ∈ [c_1, c_2, ... ] do
            sum = sum + c
        context.write(cid id, count sum)
```

The process of the 2nd MR Job is shown in Algorithm 3. The algorithm includes two classes, Class Mapper (Lines 1–21) and Class Reducer (Lines 22–27). Class Mapper contains three methods, Initialize (Lines 2–4), Map (Lines 5–15) and Close (Line 16–21), while Class Reducer only contains one method, Reduce (Lines 23–27). In the Initialize method of Mapper, we initialize two arrays, u and v, which are used to store the intermediate summations of the elements in matrices $\mathbf{U}$ and $\mathbf{V}$ respectively. In the Map method of Mapper, firstly, we initialize a local variable h (Line 6). Then, we resolve the input training sample s, dividing s into training feature $\mathbf{x}$ and its corresponding training result $\mathbf{t}$ (Line 7). Again, according to training result $\mathbf{t}$ and the result of Algorithm 2, we get the corresponding weight w of s (Line8). And then calculate the corresponding hidden layer output vector $h(\mathbf{x})$ (Lines 9–10). Finally, separately calculate local summations of the elements in matrices $\mathbf{U}$ and $\mathbf{V}$,

and save the result to local variables u and v (Lines 11–15). In the Close method of Mapper, the intermediate summations stored in u and v are emitted by the mapper (Lines 17–21). In the Reduce method of Reducer, firstly, we initialize a temporary variable uv (Line 24). And then, we combine the intermediate summations which have the same Key, and furthermore, get the final summation of the corresponding element of the Key (Lines 25–26). Finally, we store the results into the distributed file system (Line 27).

Algorithm 3 The 2nd MR Job of DWELM

```
class MAPPER
    method INITIALIZE()
        u = new ASSOCIATIVEARRAY
        v = new ASSOCIATIVEARRAY
    method MAP(sid id, sample s)
        h = new ASSOCIATIVEARRAY
        (x, t) =ParseAll(s)
        w =Weight(Counts[Class(t)])
        for i = 1 to L do
            h[i] = g(w_i · x + b_i)
        for i = 1 to L do
            for j = 1 to L do
                u[i,j] = u[i,j] + w × h[i] × h[j]
            for j = 1 to m do
                v[i,j] = v[i,j] + w × h[i] × t[j]
    method CLOSE()
        for i = 1 to L do
            for j = 1 to L do
                context.write(triple ('U', i, j), sum u[i,j])
            for j = 1 to m do
                context.write(triple ('V', i, j), sum v[i,j])
class REDUCER
    method REDUCE(triple p, sum [s_1, s_2, ... ])
        uv = 0
        for all sum s ∈ [s_1, s_2, ... ] do
            uv = uv + s
        context.write(triple p, sum uv)
```

4 Performance Evaluation

4.1 Experimental Platform

All the experiments are running on a cluster with 9 computers which are 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. One computer is set as the Master node

Table 1 Experimental parameters

Parameter	Range and default
Dimensionality (D)	10, 20, 30, 40, **50**
Number of hidden nodes (N_h)	100, 150, **200**, 250, 300
Number of records (N_r)	3M(1.4G), 4M(1.86G), **5M**(2.3G), 6M(2.8G), 7M(3.27G)
Number of classes (N_c)	5, 10, **15**, 20, 25
Imbalance ratio (R)	0.3, 0.4, **0.5**, 0.6, 0.7
Number of nodes (N_n)	1, 2, 3, 4, 5, 6, 7, **8**

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.

Because DWELM is MapReduce-based implementation of centralized WELM, and it does not change any formula in WELM, so it does not have any effect on the classification accuracy rate. In addition, the other learning algorithms of MapReduce solutions such as SVM needs many iterations to obtain the final results. Our DWELM only use two MapReduce job to gain the results. So, the performance of two MapReduce jobs is obviously optimal to several MapReduce computations. Even though we compare the SVM and DWELM, the results of our DWELM are better than SVM. Therefore, we only evaluate the training time of DWELM in the experiments. Table 1 summarizes the parameters used in our experimental evaluation, along with their ranges and default values shown in bold. In each experiment, we vary a single parameter, while setting the remainders to their default values. The imbalance ratio which quantitatively measure the imbalance degree of a dataset is defined as $\text{Min}(\#(\mathbf{t}_i))/\text{Max}(\#(\mathbf{t}_i))$ [26].

4.2 Experimental Results

Firstly, we investigate the influence of the training data dimensionality. As shown in Fig. 1, with the increase of training data dimensionality, the training time of DWELM increase slightly. Increase of training data dimensionality leads to the running time for calculating the corresponding row h_k of hidden layer output matrix $\mathbf{H}$ in Mapper slightly increases, then leads to the training time of DWELM slightly increases.

Secondly, we investigate the influence of the number of hidden nodes. As shown in Fig. 2, with the increase of the number of hidden nodes, the training time of DWELM increases. Increasing of the number of hidden nodes leads to an increase of the dimensionality of hidden layer output matrix $\mathbf{H}$, and indirectly leads to the increase of the dimensionality of the intermediate matrices $\mathbf{U}$ and $\mathbf{V}$. This not only

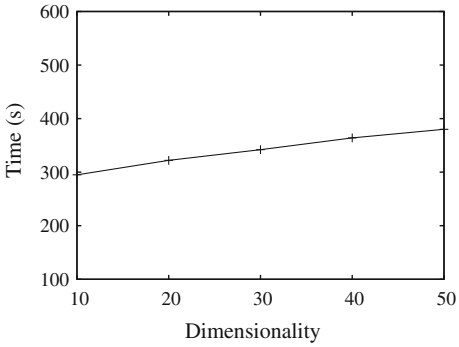

Fig. 1 The influence of D

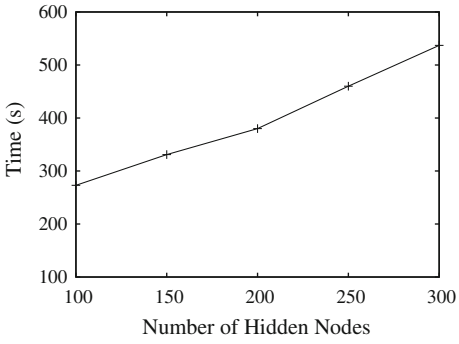

Fig. 2 The influence of N_h

makes the computation time of the local accumulated sum of **U** and **V** increase, but also makes the transmission time of intermediate results in MR Job increase. Therefore, the training time of DWELM increases with the number of hidden nodes.

Again, we investigate the influence of the number of training records. As shown in Fig. 3, with the increase of the number of records, the training time of DWELM increases obviously. Increasing of the number of records means that the number that MR Job needs to deal with increases, leading to the amount of Mapper and Reducer

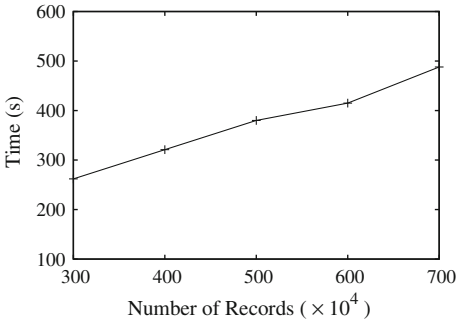

Fig. 3 The influence of N_r

which need to be launched increase. On the other hand, it increases the number of corresponding local accumulated sum of **U** and **V** which need to be transmitted, leading to the transmission time of intermediate results increases. Therefore, the training time of DWELM increases with the increasing of the number of training records.

Then, we investigate the influence of the number of classes. As shown in Fig. 4, along with the increase of the number of classes, the training time of DWELM is basically stable. The number of classes increases, which only increases the number of statistical values in the 1st MR Job and the number of input values in the 2nd MR Job of DWELM, which has limited impact on the overall training time, so the training time is relatively stable.

Next, we investigate the influence of imbalance ratio. As shown in Fig. 5, with the increase of imbalance ratio, the training time of DWELM is basically stable. Increasing of imbalance ratio did not produce any substantial effects on the calculation process of MR Job, so the training time is relatively stable.

Finally, we discuss the influence of the number of working slave nodes in the Cluster. As shown in Fig. 6, with the number of slave nodes increasing, the training time of DWELM decreased significantly. Increasing of number of slave nodes implies that increasing of the amount of Mapper/Reducers that be launched at the same time, it also means that the work can be completed in unit time increasing.

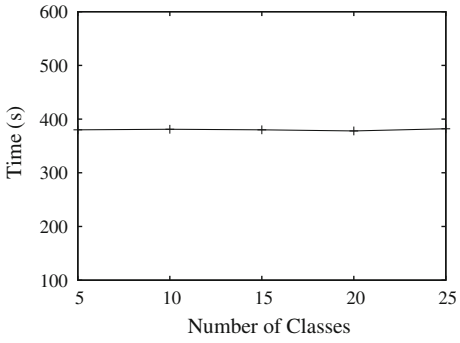

Fig. 4 The influence of N_c

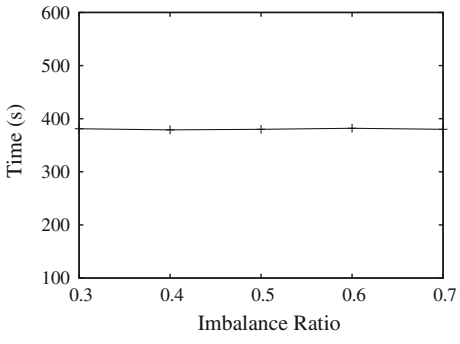

Fig. 5 The influence of R

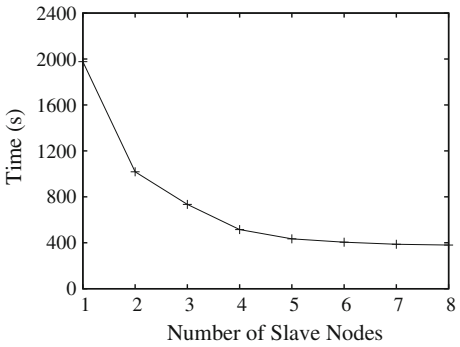

Fig. 6 The influence of N_n

Therefore, in the premise of constant total workload, the training time of DWELM decreases.

In summary, no matter how the experimental parameters change, DWELM can always deal with large-scale data (millions of data) effectively and rapidly (several minutes). At the same time, DWELM has better scalability, through the expansion of the hardware platform, they can easily handle billions and even hundreds of billion of the big imbalanced training data, thereby improve the processing efficiency of big data learning applications significantly.

5 Conclusions

Neither WELM nor DELM can cope with big imbalanced training data efficiently since they only consider either "big" or "imbalanced" aspect of big imbalanced training data. In this paper, we combine the advantages of WELM and DELM, and propose a Distributed Weighted Extreme Learning Machine based on MapReduce framework (DWELM). Specifically, through analyzing the characters of centralized WELM, we found that the matrix multiplication operators (i.e. $\mathbf{H}^T\mathbf{WH}$ and $\mathbf{H}^T\mathbf{WT}$) in WELM are decomposable. Then, we transform the corresponding matrix multiplication operators into summation forms, which suit MapReduce framework well, and propose a DWELM which calculates the matrix multiplications using two MapReduce Jobs. Finally, in the Cluster environment, we use synthetic data to do a detailed validation of the performance of DWELM with various experimental settings. The experimental results show that DWELM can learn big imbalanced training data efficiently.

Acknowledgments This research was partially supported by the National Natural Science Foundation of China under Grant Nos. 61402089 and 61472069; the 863 Program under Grant No. 2012AA011004, and the Fundamental Research Funds for the Central Universities under Grant Nos. N141904001 and N130404014.

References

1. Chen, M., Mao, S., Liu, Y.: Big data: a survey. Mob. Netw. Appl. **19**(2), 171–209 (2014)
2. Chen, J., Chen, Y., Du, X., Li, C., Lu, J., Zhao, S., Zhou, X.: Big data challenge: a data management perspective. Front. Comput. Sci. **7**(2), 157–164 (2013)
3. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1–3), 489–501 (2006)
4. Huang, G.-B., Chen, L., Siew, C.-K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Netw. **17**(4), 879–892 (2006)
5. Huang, G.-B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing **70**(16–18), 3056–3062 (2007)
6. Huang, G.-B., Chen, L.: Enhanced random search based incremental extreme learning machine. Neurocomputing **71**(16–18), 3460–3468 (2008)
7. Huang, G.-B., Ding, X., Zhou, H.: Optimization method based extreme learning machine for classification. Neurocomputing **74**(1–3), 155–163 (2010)
8. Huang, G.-B., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. IEEE Trans. Syst. Man Cybern. Part B: Cybern. **42**(2), 513–529 (2012)
9. Huang, G.-B., Wang, D.H., Lan, Y.: Extreme learning machines: a survey. Int. J. Mach. Learn. Cybern. **2**(2), 107–122 (2011)
10. Zhang, R., Huang, G.-B., Sundararajan, N., Saratchandran, P.: Multi-category classification using an extreme learning machine for microarray gene expression cancer diagnosis. IEEE/ACM Trans. Comput. Biol. Bioinf. **4**(3), 485–495 (2007)
11. Zhu, Q.-Y., Qin, A.K., Suganthan, P.N., Huang, G.-B.: Evolutionary extreme learning machine. Pattern Recogn. **38**(10), 1759–1763 (2005)
12. Wang, G., Zhao, Y., Wang, D.: A protein secondary structure prediction framework based on the extreme learning machine. Neurocomputing **72**(1–3), 262–268 (2008)
13. Zhao, X., Wang, G., Bi, X., Gong, P., Zhao, Y.: XML document classification based on ELM. Neurocomputing **74**(16), 2444–2451 (2011)
14. Sun, Y., Yuan, Y., Wang, G.: An OS-ELM based distributed ensemble classification framework in P2P networks. Neurocomputing **74**(16), 2438–2443 (2011)
15. Wang, B., Wang, G., Li, J., Wang, B.: Update strategy based on region classification using elm for mobile object index. Soft Comput. **16**(9), 1607–1615 (2012)
16. Huang, G.-B., Liang, N.-Y., Rong, H.-J., Saratchandran, P., Sundararajan, N.: On-line sequential extreme learning machine. In: Proceedings of CI, pp. 232–237 (2005)
17. Liang, N.-Y., Huang, G.-B., Saratchandran, P., Sundararajan, N.: A fast and accurate on-line sequential learning algorithm for feedforward networks. IEEE Trans. Neural Netw. **17**(6), 1411–1423 (2006)
18. Rong, H.-J., Huang, G.-B., Sundararajan, N., Saratchandran, P.: On-line sequential fuzzy extreme learning machine for function approximation and classification problems. IEEE Trans. Syst. Man Cybern.: Part B **39**(4), 1067–1072 (2009)
19. Wang, X., Shao, Q., Miao, Q., Zhai, J.: Architecture selection for networks trained with extreme learning machine using localized generalization error model. Neurocomputing **102**(1), 3–9 (2013)
20. Zhai, J., Xu, H., Wang, X.: Dynamic ensemble extreme learning machine based on sample entropy. Soft Comput. **16**(9), 1493–1502 (2012)
21. Wang, Z., Yu, G., Kang, Y., Zhao, Y., Qu, Q.: Breast tumor detection in digital mammography based on extreme learning machine. Neurocomputing **128**(3), 175–184 (2014)
22. He, Q., Shang, T., Zhuang, F., Shi, Z.: Parallel extreme learning machine for regression based on mapreduce. Neurocomputing **102**(2), 52–58 (2013)
23. Xin, J., Wang, Z., Chen, C., Ding, L., Wang, G., Zhao, Y.: ELM*: Distributed extreme learning machine with mapreduce. World Wide Web **17**(5), 1189–1204 (2014)

24. Bi, X., Zhao, X., Wang, G., Zhang, P., Wang, C.: Distributed extreme learning machine with kernels based on mapreduce. Neurocomputing **149**(1), 456–463 (2015)
25. Xin, J., Wang, Z., Qu, L., Wang, G.: Elastic extreme learning machine for big data classification. Neurocomputing **149**(1), 464–471 (2015)
26. Zong, W., Huang, G.-B., Chen, Y.: Weighted extreme learning machine for imbalance learning. Neurocomputing **101**(3), 229–242 (2013)
27. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of OSDI, pp. 137–150 (2004)
28. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
29. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. Commun. ACM **53**(1), 72–77 (2010)
30. Fletcher, R.: Practical Methods of Optimization, Volume 2: Constrained Optimization. Wiley, Hoboken (1981)
31. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. In: Proceedings of SOSP, pp. 29–43 (2003)
32. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: Proceedings of MSST, pp. 1–10 (2010)