Neurocomputing 149 (2015) 464–471

Contents lists available at ScienceDirect

Neurocomputing

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

Elastic extreme learning machine for big data classification

Junchang Xin [a,*], Zhiqiong Wang [b], Luxuan Qu [b], Guoren Wang [a]

[a] College of Information Science & Engineering, Northeastern University, China
[b] Sino-Dutch Biomedical & Information Engineering School, Northeastern University, China

ARTICLE INFO

Article history:
Received 22 August 2013
Received in revised form
25 September 2013
Accepted 30 September 2013
Available online 10 September 2014

Keywords:
Extreme learning machine
Big data classification
Incremental learning
Decremental learning
Correctional learning

ABSTRACT

Extreme Learning Machine (ELM) and its variants have been widely used for many applications due to its fast convergence and good generalization performance. Though the distributed ELM* based on MapReduce framework can handle very large scale training dataset in big data applications, how to cope with its rapidly updating is still a challenging task. Therefore, in this paper, a novel Elastic Extreme Learning Machine based on MapReduce framework, named Elastic ELM (E^2LM), is proposed to cover the shortage of ELM* whose learning ability is weak to the updated large-scale training dataset. Firstly, after analyzing the property of ELM* adequately, it can be found out that its most computation-expensive part, matrix multiplication, can be incrementally, decrementally and correctionally calculated. Next, the Elastic ELM based on MapReduce framework is developed, which first calculates the intermediate matrix multiplications of the updated training data subset, and then update the matrix multiplications by modifying the old matrix multiplications with the intermediate ones. Then, the corresponding new output weight vector can be obtained with centralized computing using the update the matrix multiplications. Therefore, the efficient learning of rapidly updated massive training dataset can be realized effectively. Finally, we conduct extensive experiments on synthetic data to verify the effectiveness and efficiency of our proposed E^2LM in learning massive rapidly updated training dataset with various experimental settings.

1. Introduction

Today, we are surrounded by data. People upload videos, take pictures on their mobile phones, text friends, update their Facebook status, leave comments around the web, and so forth. Machines, too, are generating and keeping more and more data [1]. It is difficult to estimate how much total data are stored as electronically all over the world. For example: Facebook stored almost 10 billion pictures, it is about 1PB; New York Stock Exchange will produce 1TB exchange data; The Internet Archive store the data about 2PB, and it is increasing at least in speed of 20TB data per month [2]. Thus, the era of Big Data [3] has arrived. Big Data is known for its 4Vs [4]: V1 (Volume), it involves a great volume of data; V2 (Variety), the data cannot be structured into regular database tables; V3 (Velocity), the data is produced with great velocity and must be captured and processed rapidly; V4 (Value), high commercial value but low density value, meaning that sometimes there is a very big volume of data to process before finding valuable needed information.

Due to the characteristics of excellent generalization performance, little human intervene, and rapid training speed, Extreme Learning Machine (ELM) [5–10] has recently attracted more and more researchers' attention [11]. Because of its advantage, ELM can be applied in many fields and displayed a significant result [12–26]. As a variant of ELM, distributed ELM (i.e. PELM [27] and ELM* [28]) based on MapReduce [29–31] can resolve the V1 (Volume) problem of Big Data. However, it is quite common in big data classifications that some new training data arrived, some old training data expired and some error training data corrected. For example, a large number of patients all over the world go to hospital every day. Some patients' conditions are clear and consistent, and they got doctors' diagnosis; other patients, whose conditions have been changed from the last diagnosis, need to be further examined by the doctors again; the others' conditions are re-confirmed after the consultation from doctors. All the above situations can result in updates of the training dataset on Computer-Aided Diagnosis (CAD).

A simple and direct way is to retraining the ELM* [28] using the whole training dataset. Obviously, this kind of method is time-consuming, since the training dataset is very large. Therefore, it is essential to improve the ELM* [28] algorithm, in order to support the functionalities such as incremental learning, decremental learning, and correctional learning. There is a great overlap between the old training

* Corresponding author.
E-mail address: xinjunchang@ise.neu.edu.cn (J. Xin).

http://dx.doi.org/10.1016/j.neucom.2013.09.075
0925-2312/

dataset and the new one, if the overlapped information can be used, the retraining cost will be reduced greatly. Therefore, in this paper, an Elastic ELM (E^2LM) is proposed to improve ELM* [28] with the abilities of incremental learning, decremental learning, and correctional learning. The contributions of this paper can be summarized as follows.

(1) We prove theoretically that the most expensive computation part in ELM* can be incrementally, decrementally and correctionally calculated, which indicate that ELM* can be extended to support incremental learning, decremental learning, and correctional learning.
(2) A novel Elastic Extreme Learning Machine (E^2LM) based on MapReduce framework is proposed, which can enhance the training performance of ELM* for handling the rapid updated massive training dataset.
(3) Last but not least, our extensive experimental studies using synthetic data show that our proposed E^2LM can learn the rapid updated massive training dataset effectively, which can fulfill the requirements of many big data classification applications.

The remainder of the paper is organized as follows. Section 2 briefly introduces traditional ELM and distributed ELM*. The theoretical foundation and the computational details of the proposed E^2LM approach are introduced in Section 3. The experimental results are reported in Section 4 to show the effectiveness and efficiency of our E^2LM approach. Finally, we conclude this paper in Section 5.

2. Background

In this section, we describe the background for our work, which includes a brief overview of ELM and ELM*.

2.1. ELM

ELM [5,6] 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 [7,8]. 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 a extremely fast learning speed. Besides, ELM is less sensitive to user-specified parameters and can be deployed faster and more conveniently [9,10].

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

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

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, $\boldsymbol{\beta}_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ 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 vector of the SLFNs [5].

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 $\boldsymbol{\beta}_i$, $\mathbf{w}_i$ and b_i such that

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

The equation above can be expressed compactly as follows:

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

where $\mathbf{H}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_L, b_1, b_2, \ldots, b_L, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L)$

$$= [h_{ij}] = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & g(\mathbf{w}_2 \cdot \mathbf{x}_1 + b_2) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ g(\mathbf{w}_1 \cdot \mathbf{x}_2 + b_1) & g(\mathbf{w}_2 \cdot \mathbf{x}_2 + b_2) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_2 + b_L) \\ \vdots & \vdots & \vdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & g(\mathbf{w}_2 \cdot \mathbf{x}_N + b_2) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{bmatrix}_{N \times L} \tag{4}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1m} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{L1} & \beta_{L2} & \cdots & \beta_{Lm} \end{bmatrix}_{L \times m} \quad \text{and}$$

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1m} \\ t_{21} & t_{22} & \ldots & t_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ t_{N1} & t_{N2} & \ldots & t_{Nm} \end{bmatrix}_{N \times m} \tag{5}$$

$\mathbf{H}$ 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 $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. The smallest norm least-squares solution of the above linear system is

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T} \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized the inverse of matrix $\mathbf{H}$. Then the output function of ELM can be modeled as follows.

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\mathbf{H}^{\dagger}\mathbf{T} \tag{7}$$

2.2. ELM*

The orthogonal projection method can be efficiently used in ELM [10]: $\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ if $\mathbf{H}^T\mathbf{H}$ is nonsingular or $\mathbf{H}^{\dagger} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}$ if $\mathbf{H}\mathbf{H}^T$ is nonsingular. According to the ridge regression theory, it suggests that a positive value $1/\lambda$ is added to the diagonal of $\mathbf{H}^T\mathbf{H}$ or $\mathbf{H}\mathbf{H}^T$ in the calculation of the output weights β, and the resultant solution is stable and tends to have better generalization performance [10]. Moreover, in big data applications, we can easily have $N \gg L$. Thus, the size of $\mathbf{H}^T\mathbf{H}$ is much smaller than that of $\mathbf{H}\mathbf{H}^T$, we can get

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

and the corresponding output function of ELM is

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{9}$$

Let $\mathbf{U} = \mathbf{H}^T\mathbf{H}$, $\mathbf{V} = \mathbf{H}^T\mathbf{T}$, and we can get,

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{U}\right)^{-1}\mathbf{V} \tag{10}$$

According to Eq. (4), we have $h_{ij} = g(\mathbf{w}_j \cdot \mathbf{x}_i + b_j)$, and $h_{ij}^T = h_{ji} = g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i)$. Thus, we can further have,

$$u_{ij} = \sum_{k=1}^{N} h_{ik}^T h_{kj} = \sum_{k=1}^{N} h_{ki} h_{kj} = \sum_{k=1}^{N} g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) g(\mathbf{w}_j \cdot \mathbf{x}_k + b_j) \tag{11}$$

$$v_{ij} = \sum_{k=1}^{N} h_{ik}^T t_{kj} = \sum_{k=1}^{N} h_{ki} t_{kj} = \sum_{k=1}^{N} g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) t_{kj} \tag{12}$$

According to Eqs. (11) and (12), the process of calculating $\mathbf{U}$ and $\mathbf{V}$ are both decomposable. Thus, we can use MapReduce framework to speedup the computation of output weight vector β. The process of calculating matrices $\mathbf{U}$ and $\mathbf{V}$ based on MapReduce framework is shown in Algorithm 1 [28].

The algorithm has two classes, Class Mapper (Lines 1–20) and Class Reducer (Lines 21–26). Class Mapper contains three methods, Initialize (Lines 2–4), Map (Lines 5–14) and Close (Lines 15–20), while Class Reducer only contains one method, Reduce (Lines 22–26).