Neurocomputing 74 (2011) 2532–2540

Contents lists available at ScienceDirect

Neurocomputing

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

A parallel incremental extreme SVM classifier

Qing He [a,*], Changying Du [a,b], Qun Wang [a,b], Fuzhen Zhuang [a,b], Zhongzhi Shi [a]

[a] Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[b] Graduate School of the Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Available online 14 May 2011

Keywords:
Parallel extreme SVM (PESVM)
MapReduce
Incremental extreme SVM (IESVM)
Parallel incremental extreme SVM (PIESVM)

ABSTRACT

The classification algorithm extreme SVM (ESVM) proposed recently has been proved to provide very good generalization performance in relatively short time, however, it is inappropriate to deal with large-scale data set due to the highly intensive computation. Thus we propose to implement an efficient parallel ESVM (PESVM) based on the current and powerful parallel programming framework MapReduce. Furthermore, we investigate that for some new coming training data, it is brutal for ESVM to always retrain a new model on all training data (including old and new coming data). Along this line, we develop an incremental learning algorithm for ESVM (IESVM), which can meet the requirement of online learning to update the existing model. Following that we also provide the parallel version of IESVM (PIESVM), which can solve both the large-scale problem and the online problem at the same time. The experimental results show that the proposed parallel algorithms not only can tackle large-scale data set, but also scale well in terms of the evaluation metrics of speedup, sizeup and scaleup. It is also worth to mention that PESVM, IESVM and PIESVM are much more efficient than ESVM, while the same solutions as ESVM are exactly obtained.

1. Introduction

Unlike those conventional iterative implementations, Huang et al. [1,2] proposed a new learning algorithm called extreme learning machine (ELM) for single-hidden layer feedforward neural networks (SLFNs), which randomly chooses input weights and hidden biases and analytically determines the output weights of SLFNs. The learning process of ELM for an SLFN includes two steps. First, the input vectors are mapped into the hidden layer output vectors through the hidden layer of a SLFN, with its input weights and hidden biases randomly generated. Second, a minimum norm least squares solution of the output weights is obtained by computing the generalized inverse of the hidden layer output matrix [1,2]. Though it has been studied that ELM can provide good generalization performance at an extremely fast learning speed [1,2], ELM still tends to be over-fitting at the second step according to the empirical risk minimization (ERM) principle [33–35].

To overcome the over-fitting problem in ELM, according to Vapnik's structure risk minimization (SRM) principle [34,35], Liu et al. [3] formulated a new nonlinear support vector machine (SVM), called extreme support vector machine (ESVM).[1] Later, Frénay and Verleysen [4] also performed a similar work on standard SVM. In ESVM, similar as the first step of ELM's learning process, a nonlinear map function is explicitly constructed by a random SLFN's hidden layer. Liu et al. [3] made a significant contribution. They show that the ELM learning approach can be applied to SVMs directly by simply replacing SVM kernels with random ELM kernels and better generalization can be achieved [5,3]. The ESVM classifier can also be regarded as a special form of regularization networks [6], which classifies the data points similar as proximal SVM (PSVM) [7], multisurface PSVM [8], and least squares SVM (LSSVM) [9]. As has been shown in [3], ESVM can produce better generalization performance than ELM almost all of the time and can run much faster than other nonlinear SVM algorithms with comparable accuracy.

It is observed that ESVM must do the multiplications of several matrices to obtain the solution, and the computation complexity depends on the size of the training data set. This leads to the inefficiency of ESVM when dealing with large-scale data. Furthermore, ESVM in its current form can not be used in online learning. Actually, many real life machine learning problems can be more naturally viewed as online rather than batch learning problems, the data is often collected continuously in time.

With the fast development of cloud computing, many researchers have proposed some parallel learning algorithms [10–13], such as the parallel implementation of ELM [10]. And

* Corresponding author at: Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. Tel.: +86 62600542.
E-mail addresses: heq@ics.ict.ac.cn (Q. He), 530751551@qq.com, ducy@ics.ict.ac.cn (C. Du).

[1] ESVM is essentially a linear PSVM [7] in the ELM feature space, which corresponds to a ELM with output weights trained by ridge regression (i.e. L2/Tikhonov-regularization on the output weights). So ESVM is not the same as standard SVM with ELM kernel [4].

0925-2312/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2010.11.036

lots of efforts have been devoted to the development of online/incremental learning[2] in recent years [14–23].

However, now more and more data processing problems are both large-scale and online, exploring effective and efficient algorithms that can solve both the large-scale problem and the online problem at the same time is of great significance. Although there have been many parallel and online learning algorithms, but most of them only focus on one problem, either large scale or online.

In this paper, we extend ESVM to an incremental learning algorithm IESVM, and develop the parallel implementations of ESVM and IESVM.

We note that Google has provided a PSVM: parallelizing support vector machine, but it does not support incremental learning [11]. And though several incremental SVM algorithms have been proposed [21–23], they are all serial algorithms and can not be parallelized easily, while our proposed IESVM can be parallel executed very easily.

We implement our PESVM based on MapReduce, which is a current and powerful parallel programming framework. The experiments show that PESVM scales well in terms of the evaluation metrics of speedup, sizeup and scaleup. By comparing ESVM with PESVM, IESVM and PIESVM, we observe that PESVM, IESVM and PIESVM can give exactly the same solutions as ESVM while saving much training time, which is shown in our experiments. The experiments also show that PIESVM has very fast incremental learning speed, which can be used to solve large-scale online learning problems efficiently.

The rest of the paper is organized as follows. We first give some preliminary knowledge in Section 2. Then Section 3 presents our parallel ESVM algorithm based on MapReduce. Section 4 gives our incremental ESVM classifier and its parallel implementation PIESVM. Then experimental results are shown in Section 5. Finally, we draw our conclusions in Section 6.

A word about our notations. All vectors will be column vectors unless transposed by a superscript$'$. The scalar product of two vectors x and y in n-dimensional space R^n will be denoted by $x'y$, and the 2-norm of a vector x is denoted by $\|x\|$. For a matrix $A\in R^{m\times n}$, A_i is the ith row of A which is a row vector R^n, while A_j is the jth column of A. A column vector of ones of arbitrary dimension will be denoted by e. The identity matrix of arbitrary dimension will be denoted by I.

2. Preliminary knowledge

2.1. Review of extreme SVM

To derive our parallel incremental ESVM classifier, we first give a brief description of the ESVM formulation [3].

Consider the 2-class classification problem of classifying m points in n-dimensional real space R^n, represented by the $m\times n$ matrix A. A $m\times m$ diagonal matrix D with $+1$ or -1 along its diagonal specifies the membership of class $A+$ or class $A-$ of each point A_i. For this problem, the extreme support vector machine with a linear kernel, which has the same form as the linear PSVM [7], is given by the following quadratic program with parameter $v>0$ and linear equality constraint (y is the slack variable):

$$\min_{(w,r,y)\in R^{n+1+m}} \quad \frac{v}{2}\|y\|^2+\frac{1}{2}\left\|\begin{bmatrix}w\\ r\end{bmatrix}\right\|^2$$
$$\text{s.t.} \quad D(Aw-er)+y=e \tag{1}$$

which tries to find the proximal planes: $x'w-r=\pm 1$, where w,r are the orientation and the relative location to the origin respectively. These two planes are proximal to the points in class $A+$ and class $A-$ respectively, and the plane $x'w-r=0$, which is midway between the above two proximal planes, is chosen as the separating plane which acts as below:

$$x'w-r\begin{cases}>0, & \text{then} \quad x\in A+\\ <0, & \text{then} \quad x\in A-\\ =0, & \text{then} \quad x\in A+ \text{ or } x\in A-\end{cases} \tag{2}$$

The nonlinear ESVM classifier [3] is obtained by applying the above linear formulation in a feature space, which is introduced by a specially devised nonlinear mapping function. The mapping function $\Phi(\cdot): R^n\to R^{\tilde{n}}$, which maps the input vectors into the vectors in a feature space, can be constructed as follows:

$$\Phi(x)=G(Wx^1)$$
$$=\left(g\left(\sum_{j=1}^{n}W_{1j}x_j+W_{1(n+1)}\right),\ldots,g\left(\sum_{j=1}^{n}W_{\tilde{n}j}x_j+W_{\tilde{n}(n+1)}\right)\right)' \tag{3}$$

where $x\in R^n$ is the input vector, $x^1=[x'\ 1]'$, $W\in R^{\tilde{n}\times(n+1)}$ is a matrix whose elements are randomly generated, $\Phi(x)$ is the vector corresponding to x in the feature space, and the notation $G(\cdot)$ represents a map which takes a matrix Z with elements z_{ij} and returns another matrix of the same size with elements $g(z_{ij})$, where g is an activation function (typically the sigmoidal function). Note that x, W can be interpreted as the input vector and the input weights and hidden biases of an SLFN respectively, and $\Phi(x)$ is the hidden layer's output vector of x in ELM algorithm.

For the $m\times n$ matrix A, $\Phi(A)$ is defined as $\Phi(A)=[\Phi(A'_1),\ldots,\Phi(A'_m)]'$.

Then, the nonlinear ESVM can be formulated as the following quadratic program problem with a parameter $v>0$:

$$\min_{(w,r,y)\in R^{\tilde{n}+1+m}} \quad \frac{v}{2}\|y\|^2+\frac{1}{2}\left\|\begin{bmatrix}w\\ r\end{bmatrix}\right\|^2$$
$$\text{s.t.} \quad D(\Phi(A)w-er)+y=e \tag{4}$$

From the linear constraint of (4) we can get an explicit expression of y. Substituting y by its explicit expression in the objective function of (4), we can get the following unconstrained minimization problem:

$$\min_{(w,r)\in R^{\tilde{n}+1}} \frac{v}{2}\|D(\Phi(A)w-er)-e\|^2+\frac{1}{2}\left\|\begin{bmatrix}w\\ r\end{bmatrix}\right\|^2 \tag{5}$$

Setting the gradient with respect to w and r to zero and noting that $D^2=I$ gives the following necessary and sufficient optimality conditions for (5):

$$v\Phi(A)'(\Phi(A)w-er-De)+w=0$$
$$ve'(-\Phi(A)w+er+De)+r=0 \tag{6}$$

By solving the linear system of Eqs. (6) we can obtain the following simple expression for w and r in terms of problem data:

$$\begin{bmatrix}w\\ r\end{bmatrix}=\left(\frac{I}{v}+E'_\Phi E_\Phi\right)^{-1}E'_\Phi De \tag{7}$$

where $E_\Phi=[\Phi(A)\ -e]\in R^{m\times(\tilde{n}+1)}$.

Now for an unseen point x, the nonlinear classifier works as follows:

$$\Phi(x)'w-r\begin{cases}>0, & \text{then} \quad x\in A+\\ <0, & \text{then} \quad x\in A-\\ =0, & \text{then} \quad x\in A+ \text{ or } x\in A-\end{cases} \tag{8}$$

Since linear ESVM has a very similar form as nonlinear ESVM, the only difference between them is that nonlinear ESVM is modeled in a feature space introduced by an explicit mapping

[2] In this paper, we use the terms "incremental learning" and "online learning" interchangeably.

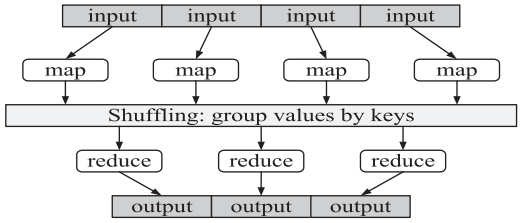


Fig. 1. Illustration of the MapReduce framework: the "map" is applied to all input records, which generates intermediate results that are aggregated by the "reduce".

function, in the following sections when we discuss the parallel algorithm and the incremental learning for ESVM we only consider its nonlinear case.

2.2. MapReduce

MapReduce [24–26], which is a current and powerful parallel programming framework, was developed within Google as a mechanism for the parallel processing of large amounts of raw data such as documents crawled from the web, web request logs, etc. As the framework shows in Fig. 1, map, written by a user of the MapReduce library, takes an input pair and produces a set of intermediate *key*/*value* pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function. The reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Hadoop [27] provides a free open source Java MapReduce implementation. Both Google and Hadoop provide MapReduce runtimes with fault tolerance and dynamic flexibility support.

A MapReduce application is executed in a parallel manner through two phases. In the first phase, all *map* functions can be executed independently with each other. In the second phase, each *reduce* function may depend on the outputs generated by any number of map operations. However, similar as map operations, all reduce operations can be executed independently.

From the perspective of dataflow, a map/reduce job consists of some independent map tasks and some independent reduce tasks. The map/reduce job usually splits the input data set into independent chunks which are processed by the map tasks in a completely parallel manner. The MapReduce framework sorts the outputs of the maps, which are then the input to the reduce tasks. Typically both the input and the output of the job are stored in a file system.

The MapReduce framework operates exclusively on $<key,value>$ pairs, that is, the framework views the input to the job as a set of $<key,value>$ pairs and produces a set of $<key,value>$ pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework. Several serialization systems exists; the default serialization mechanism requires keys and values to implement the Writable interface. Additionally, the key classes must facilitate sorting by the framework; a straightforward means to do so is for them to implement the WritableComparable interface.

The input and output types of a MapReduce job can be described as follows: (input) $<k_1,v_1>$ $\rightarrow$ map $\rightarrow$ $<k_2,v_2>$ $\rightarrow$ reduce $\rightarrow$ $<k_3,v_3>$ (output).

3. Parallelization of ESVM based on MapReduce

To make ESVM algorithm has a good scalability, and run faster when the input data set is very large, we propose a parallelization method for the ESVM classifier, and we implement it based on Hadoop's MapReduce framework.

3.1. Analysis of parallel extreme SVM

Our main task is to parallel execute the computation of (7).

First, for a given $A\in R^{m\times n}$, to compute $\Phi(A)=[\Phi(A'_1),\ldots,\Phi(A'_m)]'\in R^{m\times\tilde{n}}$, we can compute each $\Phi(A'_i)$, $i=1,\ldots,m$, in each parallel computing unit in principle. Hence, this part can be easily parallel executed.

As stated above, $E_\Phi=[\Phi(A)\ -e]\in R^{m\times(\tilde{n}+1)}$, so $E'_\Phi\in R^{(\tilde{n}+1)\times m}$, to calculate $E'_\Phi E_\Phi$, $E'_\Phi De$, we introduce the block multiplication of matrices. Note that D is a $m\times m$ diagonal matrix with ±1 labels along its diagonal denoting the class of each row of A, and e is a m dimensional column vector of ones, so De is a m dimensional column vector. Define $E_\Phi=[\alpha_1,\ldots,\alpha_m]'$, $De=[d_1,\ldots,d_m]'$, where α_i, $i=1,\ldots,m$, is a $\tilde{n}+1$ dimensional column vector, and d_i, $i=1,\ldots,m$, is the class label of point A_i. Then we have

$$E'_\Phi E_\Phi=[\alpha_1,\ldots,\alpha_m][\alpha_1,\ldots,\alpha_m]'=\sum_{i=1}^{m}\alpha_i\alpha'_i$$

$$E'_\Phi De=[\alpha_1,\ldots,\alpha_m][d_1,\ldots,d_m]'=\sum_{i=1}^{m}d_i\alpha_i \tag{9}$$

where $\alpha_i\alpha'_i$ is a $(\tilde{n}+1)\times(\tilde{n}+1)$ matrix, $d_i\alpha_i$ is a $\tilde{n}+1$ dimensional column vector.

In principle, we can compute each $\alpha_i\alpha'_i$ and $d_i\alpha_i$, $i=1,\ldots,m$, in each parallel computing unit (when the computation of these parallel computing units are finished, we sum up them by a procedure). Therefore, by introducing the block multiplication of matrices, the matrix multiplication process, whose computation complexity depends on the number of training data points, can be effectively parallel executed.

We know that $I/v+E'_\Phi E_\Phi$ is a $(\tilde{n}+1)\times(\tilde{n}+1)$ matrix, where $\tilde{n}$ can be typically very small (usually less than 200, as is shown in [3]) and is independent of the number of the training points m. To compute its inverse $(I/v+E'_\Phi E_\Phi)^{-1}$ (whose computation is of order $(\tilde{n}+1)^3$, which is very small compared with that of very large matrices' multiplication), we need not to introduce any parallelization method.

When we have gotten $E'_\Phi E_\Phi$, $E'_\Phi De$ and $(I/v+E'_\Phi E_\Phi)^{-1}$, where $(I/v+E'_\Phi E_\Phi)^{-1}$ is a $(\tilde{n}+1)\times(\tilde{n}+1)$ matrix, $E'_\Phi De$ is a $(\tilde{n}+1)$ dimensional column vector, to compute the solution $(I/v+E'_\Phi E_\Phi)^{-1}E'_\Phi De$ is very easy (whose computation is of order $(\tilde{n}+1)^2$), we serially execute it also.

Finally, when the testing data is large we can divide it into many parts, and test each part in each parallel computing unit.

We give an explicit statement of our parallel ESVM algorithm in Algorithm 1 now. To make the algorithm clear, we assume that we have enough parallel computing units, and assign each input data point to a parallel computing unit.

Algorithm 1. Parallel ESVM classifier (PESVM).

Given an input data set of m data points in R^n represented by the $m\times n$ matrix A and a diagonal matrix D of ±1 labels denoting the class of each row of A, we generate the parallel ESVM classifier as follows:

1. Execute the following procedure for point A_i on the ith parallel computing unit, where $i=1,\ldots,m$:
   (a) Compute $\Phi(A'_i)$, where $\Phi(\cdot)$ is defined as (3);
   (b) Define $\alpha_i=\begin{bmatrix}\Phi(A'_i)\\ -1\end{bmatrix}$ and $d_i=D_{ii}$;
   (c) Compute $\alpha_i\alpha'_i$ and $d_i\alpha_i$;
2. Sum up $\alpha_i\alpha'_i$ and $d_i\alpha_i$ respectively, where $i=1,\ldots,m$. And let $E'_\Phi E_\Phi=\sum_{i=1}^{m}\alpha_i\alpha'_i$, $E'_\Phi De=\sum_{i=1}^{m}d_i\alpha_i$.
3. Compute the inverse $(I/v+E'_\Phi E_\Phi)^{-1}$, for some positive value of v. (Typically v is chosen by means of a tuning set.)
4. Compute the solution $(I/v+E'_\Phi E_\Phi)^{-1}E'_\Phi De$.

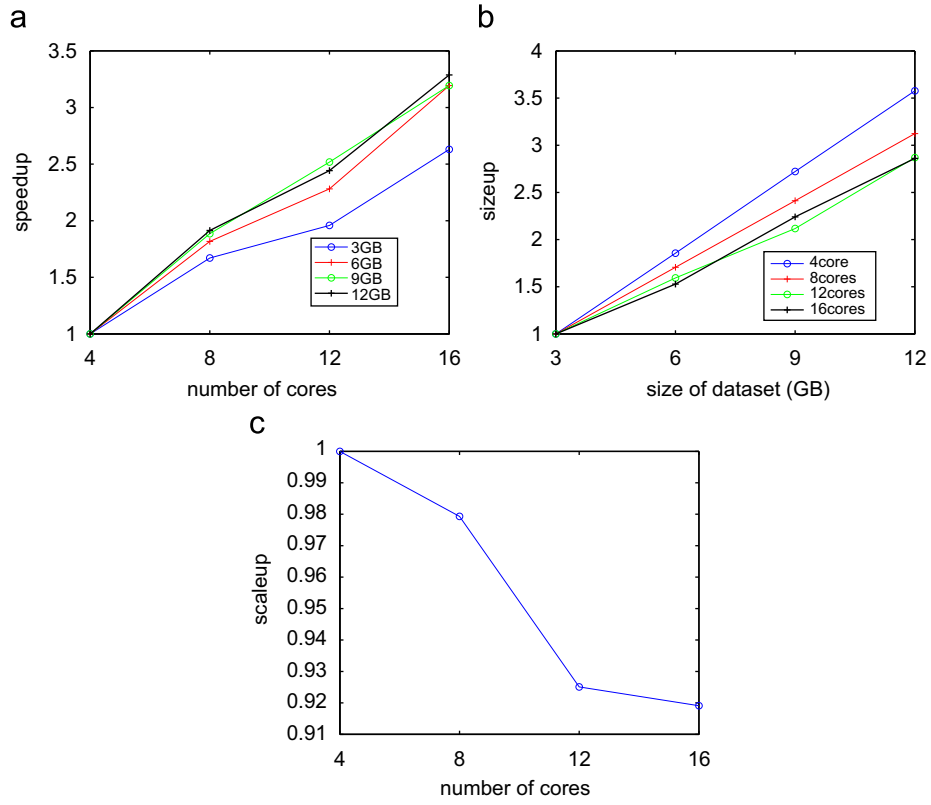


Fig. 2. Performance of the proposed parallel ESVM algorithm: (a) speedup, (b) sizeup, (c) scaleup.

Note that step 1 parallel executes the main computation in ESVM, which grows linearly in the number of the input training data points. So when the data set is larger, we may easily use more parallel computing units to preserve the fast speed of our parallel classifier. Further, this parallel algorithm gives exactly the same solution as a serial ESVM. The performance evaluation of the parallel algorithm with respect to speedup, scaleup and sizeup is shown in Fig. 2.

3.2. Implementation of parallel ESVM based on MapReduce

In this section we give the main procedures in parallel ESVM based on MapReduce, we will describe them in pseudocodes.

From the analysis above, we can see that parallel ESVM algorithm needs two map/reduce jobs. One for training (to be concrete, for getting $E'_\Phi E_\Phi$, $E'_\Phi De$) and the other for testing.

Both the two jobs' input data sets (i.e. training data set and testing data set) are stored on Hadoop distributed file system (HDFS) [28] as a sequence file of $<key,value>$ pairs, each of which represents a record in the data set. The *key* is the offset in bytes of this record to the start point of the data file, and the *value* is a string of the content of this record. The data set is split and globally broadcasted to all mappers.

In the training job, the map function first maps the input vector into a feature space by a random SLFN as (3), then constructs α_i and conducts the vector products of $\alpha_i\alpha_i'$ and $\alpha_i d_i$. In order to decrease the cost of network communication, we define two global variables[3] for each map task to accumulate each $\alpha_i\alpha_i'$ and $\alpha_i d_i$ within the same map task. We will collect these two global variables as the outputs in the close function of each map task. (The close function is executed when one map task is to be completed.)

[3] They are global just within a single map task. Since a node/computer may have several cores, each core executes a map/reduce task, they are not global on a node.

The pseudocode of the map function of the training job is shown in Algorithm 2. Note that Steps 5 and 7 accumulate each $\alpha_i\alpha_i'$ and $\alpha_i d_i$.

Algorithm 2. Train_map (*key*,*value*).

Input: Global variables *localmatrix* and *localvector* which are initialized with zeros, the offset *key*, the sample *value*
Output: Global variables *localmatrix* and *localvector* which have accumulated $\alpha_i\alpha_i'$ and $\alpha_i d_i$ respectively

1. Construct the sample *instance* and the class label d_i from *value*;
2. Compute $\Phi(instance)$, where $\Phi(\cdot)$ is defined as (3);
3. Construct α_i as $\alpha_i=[\Phi(instance)' \ -1]'$;
4. Compute $\alpha_i\alpha_i'$;
5. $localmatrix \leftarrow localmatrix+\alpha_i\alpha_i'$;
6. Compute $\alpha_i d_i$, where d_i is the class label of *instance*;
7. $localvector \leftarrow localvector+\alpha_i d_i$;

The pseudocode of the close function of each map task of the training job is shown in Algorithm 3. Note that Steps 1 and 5 output the intermediate data which will be used in the subsequent reduce procedures.

Algorithm 3. Train_map_close ().

Input: Global variables *localmatrix* and *localvector* which respectively represent the sum of local matrixes and local vectors within the same map task
Output: Global variables *localmatrix* and *localvector* which have accumulated $\alpha_i\alpha_i'$ and $\alpha_i d_i$ respectively $<key',value'>$ pairs, where key' is a number, $value'$ is a string comprises the values of a vector

1. for each $p=1,\ldots,$ the number of rows of *localmatrix* do
   - Take p as key';

• Construct $value'$ as a string, which comprises the values of *localmatrix*'s pth row;
• Emit $< key', value' >$ pair;
2. end for
3. Take the number of rows of *localmatrix*$+1$ as key';
4. Construct $value'$ as a string, which comprises the values of *localvector*;
5. Emit $< key', value' >$ pair;

Finally, the reduce function sums up the outputs generated from all of the map tasks. The pseudocode of reduce function of the training job is shown in Algorithm 4.

Algorithm 4. Train_reduce (key,V).

Input: key is the row number of *localmatrix* or the number of rows of *localmatrix*$+1$, V is the list of strings from different host, each of which comprises a *localmatrix*'s key-th row or a *localvector*
Output: $< key', value' >$ pairs, where key' is a row number, $value'$ is a string which comprises the values of a vector, which represents the key'-th row of $E'_\Phi E_\Phi$, or $E'_\Phi De$
1. Initialize one array *temparray* with zeros, to record the sum of the vectors in the list V;
2. while $V.hasNext()$ do
• Construct the vector *vector* from $V.next()$;
• $temparray \leftarrow temparray + vector$;
3. end while
4. Take key as key';
5. Construct $value'$ as a string, which comprises the values of *temparray*;
6. Emit $< key', value' >$ pair;

After the training job, we have got $E'_\Phi E_\Phi$ and $E'_\Phi De$, now to compute the solution of ESVM algorithm, all we need is getting the inverse $(I/v+E'_\Phi E_\Phi)^{-1}$ with some positive parameter v and conducting the multiplication $(I/v+E'_\Phi E_\Phi)^{-1}E'_\Phi De$, whose computing orders typically are very small, it is very easy to serially implement them.

When we have gotten the solution, the next job is testing.

In the testing job, the map function first maps the input vector into a feature space by a random SLFN as (3) also, then classifies the point according to Eq. (8). To employ the solution obtained in the training phase, we define two global variables for each map task of the testing job.

The pseudocode of the map function of the testing job is shown in Algorithm 5.

Algorithm 5. Test_map (key,$value$).

Input: the offset key, the sample $value$, global variables w and r which is solution
Output: $< key', value' >$ pairs, where key' is the flag of correct classification or wrong classification and $value'$ is a string which comprises the sample information and the classification result
1. Construct the sample *instance* and its class label *classlabel* from $value$;
2. Compute $\Phi(instance)$ where $\Phi(\cdot)$ is defined as (3);
3. Compute $\Phi(instance)'w-r$;
4. if $\Phi(instance)'w-r \geq 0$ then
$predictclass = A+$;
else
$predictclass = A-$;
5. if $classlabel == predictclass$ then
$key' = correct$;
else
$key' = wrong$;
6. Construct $value'$ as a string, which comprises $value$ and $predictclass$;
7. Emit $< key', value' >$ pair;

The reduce function of the testing job is not necessary if just the classification result is wanted, and the outputs of map function will be written to the distributed file system directly. If we also want to get the testing accuracy we can implement it easily via a reduce function, here we omit it.

Now, we give a description on how the *map*/*reduce* procedures fit together and how the dataflow is when calling these procedures.

As we introduced in Section 2, the input and output types of a MapReduce job can be summarized as follows: (input) $< k_1, v_1 >$ $\rightarrow$ map $\rightarrow$ $< k_2, v_2 >$ $\rightarrow$ reduce $\rightarrow$ $< k_3, v_3 >$ (output).

In the training job of PESVM, the input is a vector, k_1 is its offset in the input sequence file of $< key, value >$ pairs, v_1 is the vector. It is processed by the map function just as in Algorithm 2, and the outputs of map is some new $< k_2, v_2 >$ pairs, where k_2 is a row number, v_2 is a string comprises of the corresponding row vector. These $< k_2, v_2 >$ pairs of different maps will be first sorted by the map/reduce framework according to their keys, then they are input to the reduce function. As it is shown in Algorithm 4, the reduce function sums up these pairs according to their keys, and outputs the final results as $< k_3, v_3 >$.

4. Incremental ESVM and its parallel implementation

4.1. Incremental learning algorithm for ESVM

Now we describe an incremental learning algorithm for ESVM which is capable of adding new data to generate an appropriately altered classifier. Assume that the current classifier is based on an input data set $A^1 \in R^{m^1 \times n}$, and a corresponding $m^1 \times m^1$ diagonal matrix D^1 of ± 1, and the classifier has the following form:

$$\begin{bmatrix} w \\ r \end{bmatrix} = \left(\frac{I}{v} + (E^1_\Phi)' E^1_\Phi \right)^{-1} (E^1_\Phi)' D^1 e \tag{10}$$

where $E^1_\Phi = [\Phi(A^1) \ -e] \in R^{m^1 \times (\tilde{n}+1)}$.

Suppose that a new set of training data points represented by the new matrix[4] $A^2 \in R^{m^2 \times n}$ needs to be added, for which we correspondingly have $E^2_\Phi = [\Phi(A^2) \ -e] \in R^{m^2 \times (\tilde{n}+1)}$, and an $m^2 \times m^2$ diagonal matrix D^2 of ± 1. The classifier (10) can be updated to reflect the addition of the new training dada as follows:

$$\begin{aligned} \begin{bmatrix} w \\ r \end{bmatrix} &= \left(\frac{I}{v} + [(E^1_\Phi)' \ (E^2_\Phi)'] \begin{bmatrix} E^1_\Phi \\ E^2_\Phi \end{bmatrix} \right)^{-1} [(E^1_\Phi)' \ (E^2_\Phi)'] \begin{bmatrix} D^1 O \\ O D^2 \end{bmatrix} e \\ &= \left(\frac{I}{v} + (E^1_\Phi)' E^1_\Phi + (E^2_\Phi)' E^2_\Phi \right)^{-1} ((E^1_\Phi)' D^1 e + (E^2_\Phi)' D^2 e) \end{aligned} \tag{11}$$

For the derivation of the classifier in (10), $(E^1_\Phi)' E^1_\Phi$ and $(E^1_\Phi)' D^1 e$ have been computed before, so to obtain the new solution we only need to compute $(E^2_\Phi)' E^2_\Phi$ and $(E^2_\Phi)' D^2 e$ now, and then add them to $(E^1_\Phi)' E^1_\Phi$ and $(E^1_\Phi)' D^1 e$ respectively. After that, all we need is computing the inverse of a $(\tilde{n}+1) \times (\tilde{n}+1)$ matrix and the

[4] Here m^2 is the number of new data points, it does not represent $m*m$. When appears in the sequel, it has the same meaning.

Table 1
The comparison between IESVM and ESVM.

Training set	Samples	Training time (s)	
		ESVM	IESVM
First training	100000	13.213	13.112
Incremental 1	50000	18.845	6.658
Incremental 2	25000	22.215	3.398
Incremental 3	12500	23.603	1.728
Incremental 4	12500	25.84	1.722

product of this inverse and a $(\tilde{n}+1)$ dimensional vector ($\tilde{n}$ typically is very small, usually less than 200 [3,1]).

We summarize our incremental ESVM algorithm in Algorithm 6.

Note that this incremental learning algorithm gives exactly the same solution as an iterative ESVM and the parallel ESVM. And all we need to store is the random matrix $W \in R^{\tilde{n}\times(n+1)}$, relatively small $(\tilde{n}+1)\times(\tilde{n}+1)$ matrix $(E_\Phi^1)'E_\Phi^1$ and the $(\tilde{n}+1)$ dimensional vector $(E_\Phi^1)'D^1e$. Since it takes $2\tilde{n}(n+1)m^2$ operations to compute $\Phi(A^2)$, $2(\tilde{n}+1)^2m^2$ operations to compute $(E_\Phi^2)'E_\Phi^2$ and $2(\tilde{n}+1)m^2$ operations to compute $(E_\Phi^2)'D^2e$, the amount of computation of the incremental algorithm only grows linearly in the number of new data points. The comparison between IESVM and ESVM is shown in Table 1.

Algorithm 6. Incremental ESVM classifier (IESVM).

Given a current classifier that is based on an input data set of m^1 data points in R^n represented by the $m^1\times n$ matrix A^1 and a diagonal matrix D of ±1 labels denoting the class of each row of A^1, we generate an incremental nonlinear classifier by adding new data represented by a new matrix $A^2\in R^{m^2\times n}$ and a corresponding diagonal matrix $D^2\in R^{m^2\times m^2}$ of ±1 as follows:

1. Compute $\Phi(A^2)\in R^{m^2\times\tilde{n}}$ for the new data, which needs the random generated matrix W which is stored in the memory.
2. Define $E_\Phi^2=[\Phi(A^2)\ -e]\in R^{m^2\times(\tilde{n}+1)}$.
3. Compute $(E_\Phi^2)'E_\Phi^2$ and $(E_\Phi^2)'D^2e$.
4. Add the products obtained in step 3 to $(E_\Phi^1)'E_\Phi^1$ and $(E_\Phi^1)'D^1e$ respectively, which are stored in the memory.
5. Compute the inversion of a $(\tilde{n}+1)\times(\tilde{n}+1)$ matrix and the product of this inversion and a $(\tilde{n}+1)$ dimensional vector.
6. The new solution is given by (11).

Furthermore, this incremental process allows us to handle arbitrarily large data sets, by successively adding new data [29], which can be very useful when the training data can not be obtained in one time (In online learning, constantly there is new incoming data[5]) or the training data is too large to store in memory.

We observe that IESVM is very similar with incremental proximal SVM (IPSVM) [29,3], and the incremental mechanisms of them are identical. But IPSVM is only a linear classifier, the classification ability of which may be restricted [29,7], while IESVM is a nonlinear classifier with very good generalization performance [3]. The linear form of IESVM can be exactly the same as IPSVM.

4.2. The parallel implementation of IESVM

In above we have given an incremental algorithm and a parallel algorithm for ESVM, which are very useful when the training data can not be obtained in one time or the number of input data points is very large. By incorporating them we can get a parallel incremental ESVM classifier, which can further improve the training speed of the classifier.

We give an explicit statement of our parallel incremental ESVM algorithm in Algorithm 7.

Note that the only difference between parallel ESVM classifier and this parallel incremental ESVM classifier is that the latter one adds a Step 3 to the parallel ESVM algorithm. Moreover, the computation of Step 3 is very small, which is of order $(\tilde{n}+1)^2$, and which is independent of the number of data points. So the performances of these two parallel algorithms are very similar.

Algorithm 7. Parallel incremental ESVM classifier (PIESVM).

Given a current classifier that is based on an input data set of m^1 data points in R^n represented by the $m^1\times n$ matrix A^1 and a diagonal matrix $D$of ±1 labels denoting the class of each row of A^1, we generate a parallel incremental nonlinear classifier by adding new data represented by a new matrix $A^2\in R^{m^2\times n}$ and a corresponding diagonal matrix $D^2\in R^{m^2\times m^2}$ of ±1 as follows:

1. Execute the following procedure for point A_i in A^2 on the i-th parallel computing unit, where $i=1,\ldots,m^2$:
   (a) Compute $\Phi(A_i')$;
   (b) Define $\alpha_i=\begin{bmatrix}\Phi(A_i')\\ -1\end{bmatrix}$, and $d_i=D_{ii}^2$;
   (c) Compute $\alpha_i\alpha_i'$ and $d_i\alpha_i$;
2. Sum up $\alpha_i\alpha_i'$ and $d_i\alpha_i$ respectively, where $i=1,\ldots,m^2$.And let $(E_\Phi^2)'E_\Phi^2=\sum_{i=1}^{m^2}\alpha_i\alpha_i'$, $(E_\Phi^2)'D^2e=\sum_{i=1}^{m^2}d_i\alpha_i$
3. Add the products obtained in step 2 to $(E_\Phi^1)'E_\Phi^1$ and $(E_\Phi^1)'D^1e$ respectively, which are stored in the memory.
4. Compute the inversion of a $(\tilde{n}+1)\times(\tilde{n}+1)$ matrix and the product of this inversion and a $(\tilde{n}+1)$ dimensional vector.
5. The new solution is given by (11).

We observe that PIESVM can have very fast parallel incremental learning speed while giving exactly the same solution as ESVM. The comparison among ESVM, PESVM, IESVM, and PIESVM are given in Experiments section.

Now, large-scale online learning is very popular, which requires the learning machine has a fast incremental learning speed, so PIESVM has a very good application prospective.

5. Experiments

Since the high classification accuracy of ESVM has been shown in [3], and PESVM, IESVM and PIESVM can give exactly the same solutions as ESVM, so our experiments only focus on evaluating their efficiencies.[6]

5.1. Experimental setup

Data sets: The data set Australian from the public UCI repository [30] is adopted in the experiments. Since the size of this data

[5] In fact, we can also implement the "decremental" learning easily in our learning scheme. Since in some applications, a part of the old data may be out of date when the new training data come, forgetting the oldest instances is also of sense. To be concrete, consider the case where we want to forget the oldest $1/L$ part of the old data when we add in equal quantity new data. Instead of only storing one matrix $(E_\Phi^1)'E_\Phi^1$ and one vector $(E_\Phi^1)'D^1e$ each time in the IESVM algorithm, here we need to store L matrices $(E_\Phi^1)'E_\Phi^1$, $(E_\Phi^2)'E_\Phi^2,\ldots,(E_\Phi^L)'E_\Phi^L$ and L vectors $(E_\Phi^1)'D^1e$, $(E_\Phi^2)'D^2e,\ldots,(E_\Phi^L)'D^Le$ each time, where $(E_\Phi^1)'E_\Phi^1$ and $(E_\Phi^1)'D^1e$ record the oldest $1/L$ part of data. When the new data $(E_\Phi^N)'E_\Phi^N$ and $(E_\Phi^N)'D^Ne$ come, we let $(E_\Phi^1)'E_\Phi^1=(E_\Phi^2)'E_\Phi^2$, $(E_\Phi^2)'E_\Phi^2=(E_\Phi^3)'E_\Phi^3,\ldots,(E_\Phi^L)'E_\Phi^L=(E_\Phi^N)'E_\Phi^N$ and $(E_\Phi^1)'D^1e=(E_\Phi^2)'D^2e$, $(E_\Phi^2)'D^2e=(E_\Phi^3)'D^3e,\ldots,(E_\Phi^L)'D^Le=(E_\Phi^N)'D^Ne$, and add them together respectively to get the new solution as in Algorithm 6.

[6] The source codes of our implementation are available at: http://www.intsci.ac.cn/users/duchangying/papers/PIESVM_Codes.zip.

set is small and we only focus on the efficiency evaluation of our algorithms, we construct the large-scale data sets by duplicating the data to satisfy the experimental requirement.

Experimental environment: We run our parallel experiments on a computing cluster of four computers, each of which has four 2.8 GHz cores and 4 GB memory. Hadoop version 0.17.0 and Java 1.5.0_14 are installed in CentOS system as the MapReduce environment.

All the serial experiments are conducted on a single core laptop with 2.2 GHz CPU and 2 GB memory. Java 1.6.0 is installed in Windows 7 system as the experiment environment.

Evaluation metrics: We evaluate the performance of our proposed parallel algorithms PESVM and PIESVM in terms of speedup, sizeup and scaleup [31,32]. Scaleup evaluates the ability of the algorithm to grow both the system and the data set size. Scaleup is defined as the ability of an m-times larger system to perform an m-times larger job in the same run-time as the original system. Obviously, the higher the scaleup curve is (which indicates that the communication time is small), the better performance an algorithm demonstrates.

5.2. Experimental results

5.2.1. Performance of PESVM

Performance experiments of PESVM are run on the cluster of four computers. We duplicate the data set Australian to get a large-scale data set up to 12 GB (totally about 148 millions of data points, two thirds of which are used as training set, other one third used as test set).

To measure the speedup, we fix the size of data set and increase the core number of computers (e.g., from one computer to four computers in this setting, and the number of cores varied from 4 to 16.). Also we investigate different cases of data size, such as the sizes of data sets are 3, 6, 9 and 12 GB. All the results are shown in Fig. 2(a). We can find that PESVM has a very good speedup performance. The maximal value can reach about 3.3 when we use four computers. Also with the size of the data set increases, the speedup performs better. The reason is that the time to process small data set is not dominant compared with the time consumed by communication and task arrangement. When the size of data set increase, the time of intensive computation becomes dominant, so the speedup increases. Therefore, PESVM algorithm can process large data sets efficiently.

Sizeup measures how much longer it takes on a given system, when the data set size is m-times larger than the original data set. For the measure of sizeup, we fix the number of cores to 4, 8, 12, and 16 respectively, and increase the size of data set from 3 to 12 GB for each core number. Fig. 2(b) shows the results, and the higher value of the sizeup indicates the longer time the system takes when the size of data set increases. When we increase the size of data set from 3 to 6 GB, the sizeup of a four cores (1 node) system is nearly 2 while it is only about 1.5 for the 16 cores (four nodes) system. This is because the communication time of the four cores system is smaller than that of the 16 cores system, and this communication time would not increase much in PESVM when the data set is doubled. So PESVM has a good sizeup performance.

Scaleup validates how well the PESVM algorithm handles larger data sets when more cores of computers are available. We simultaneously increase the number of computers and the size of data set, thus we can obtain four values of scaleup for the following combinations (four cores, 3 GB), (eight cores, 6 GB), (12 cores, 9 GB) and (16 cores, 12 GB). The higher scaleup value is, better the performance we get. The results in Fig. 2(c) show that the values of scaleup are all higher than 0.91, which indicates PESVM algorithm scales well.

5.2.2. Performance of IESVM

In this section we compare IESVM with ESVM only in term of the training time since both of them are serial algorithms. The experiments are conducted on the single core laptop introduced in experimental setup. We choose the first 500 samples of data set Australian and duplicate it to construct a large data set T, which contains 200 000 of data points.

We divide T into five parts, they respectively contain $T/2$, $T/4$, $T/8$, $T/16$, $T/16$ data points. And we assume that at first we only have $T/2$ training data points, the rest four parts of training data are incrementally added into the training process. When the new training data comes, ESVM must retrain the model over all obtained training data to get an "updated" classifier, while IESVM can update the model only using the new incoming data and the existing model. Thus this incremental learning manner only involves some simply computation, which can dramatically save the training time. The training time of these two algorithms are listed in Table 1. These results are the average of five runs, which show that IESVM obviously has a speed advantage over ESVM.

5.2.3. Comparison of our proposed algorithms

As mentioned before, the proposed methods PESVM, IESVM and PIESVM can give exactly the same solutions as ESVM. Actually, IESVM is the extended version of ESVM to deal with online learning scenario, and PESVM and PIESVM are the parallel versions of ESVM and IESVM respectively. So here we also study their learning speeds when training the classification models.

The parallel experiments of PESVM and PIESVM are conducted on a four cores computer of the computing cluster introduced in experimental setup, and the serial ones of ESVM and IESVM are conducted on the one core laptop. We also choose the first 500 samples of data set Australian and duplicate it to construct a large data set M, which contains 20 million of data points.

We divide M into five parts, they respectively contain $M/2$, $M/4$, $M/8$, $M/16$, $M/16$ data points. The experimental setting here is similar as Section 5.2.2. The results of training time listed in Table 2 demonstrate that PIESVM has very fast incremental learning speed, e.g., to learn the last incremental $M/16$ data points, ESVM needs to train the whole obtained data set M, which needs 2597.078 s, while PIESVM only needs 44.083 s.[7] When the training data is larger, we can simply use more computers to preserve the fast speed of PIESVM, and the advantage of PIESVM over ESVM will be more obvious, so PIESVM can be used to solve large-scale online learning problems effectively and efficiently.

6. Conclusions

In this paper, based on MapReduce, a parallelization method for the effective classification algorithm ESVM is proposed. Also an incremental ESVM algorithm is introduced. By incorporating incremental ESVM and parallel ESVM, a powerful parallel incremental extreme SVM classifier is provided. The experimental results show that these parallel algorithms not only can tackle large-scale data set, but also scale well in terms of the evaluation metrics of speedup, sizeup and scaleup. And the incremental ESVM classifier can add new data very easily, it has an obvious speed advantage over ESVM. Moreover, PESVM, IESVM and PIESVM can give exactly the same solutions as ESVM while saving much training time. The experiments also show that PIESVM has very fast incremental learning speed, which can be used to solve both the large-scale problem and the online learning problem effectively and efficiently. Since PIESVM

[7] Although the 2.20 GHz laptop may be a little slower than the 2.8 GHz computer, the results still can verify the advantage of the proposed methods.

Table 2
The comparison among ESVM, PESVM, IESVM, and PIESVM.

Training set	Samples	Training time (s)			
		ESVM	IESVM	PESVM	PIESVM
First training	10 million	1296.841	1297.296	177.703	178.915
Incremental 1	5 million	1944.219	649.761	253.064	97.744
Incremental 2	2.5 million	2270.025	326.354	290.04	59.287
Incremental 3	1.25 million	2434.52	165.472	308.882	43.218
Incremental 4	1.25 million	2597.078	165.523	330.339	44.083

has a very good application prospective, we intend to explore multicategory PIESVM in our future work.

Acknowledgments

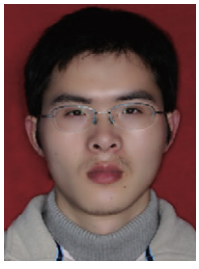
This work is supported by the National Natural Science Foundation of China (No. 60933004, 60975039, 61035003, 60903141, 61072085), National Basic Research Priorities Programme (No. 2007CB311004) and National Science and Technology Support Plan (No. 2006BAC08B06).

References

[1] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of International Joint Conference on Neural Networks (IJCNN2004), vol. 2, Budapest, Hungary, 25–29 July 2004, pp. 985–990.
[2] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[3] Q. Liu, Q. He, Z. Shi, Extreme support vector machine classifier, Lecture Notes in Computer Science, vol. 5012, 2008, pp. 222–233.
[4] B. Frénay, M. Verleysen, Using SVMs with randomised feature spaces: an extreme learning approach, in: Proceedings of the 18th European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, 28–30 April 2010, pp. 315–320.
[5] G.-B. Huang, X.-J. Ding, H.-M. Zhou, Optimization method based extreme learning machine for classification, Neurocomputing 74 (12) (2010) 155–163.
[6] T. Evgeniou, M. Pontil, T. Poggio, Regularization networks and support vector machines, in: Advances in Large Margin Classifiers, 2000, pp. 171–203.
[7] G. Fung, O.L. Mangasarian, Proximal support vector machine classifiers, in: Proceedings of ACM SIGKDD 2001, ACM Press, 2001, pp. 77–86.
[8] O.L. Mangasarian, E.W. Wild, Multisurface proximal support vector machine classification via generalized eigenvalues, IEEE Transaction on Pattern Analysis and Machine Intelligence 28 (1) (2006) 69–74.
[9] J.A.K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Processing Letters 9 (3) (1999) 293–300.
[10] M.V. Heeswijk, Y. Miche, E. Oja, A. Lendasse, Solving large regression problems using an ensemble of GPU-accelerated ELMs, in: Proceedings of the 18th European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, 28–30 April 2010, pp. 309–314.
[11] E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, H. Cui, Parallelizing support vector machines on distributed computers, in: J. Platt, D. Koller, Y. Singer, S. Roweis (Eds.), NIPS 20, MIT Press, 2007, pp. 257–264.
[12] P.K. Chan, S.J. Stolfo, Toward parallel and distributed learning by meta-learning, in: AAAI Workshop in Knowledge Discovery in Databases, 1993, pp. 227–240.
[13] C. Kruengkrai, C. Jaruskulchai, A parallel learning algorithm for text classification, in: Proceedings of ACM SIGKDD 2002, ACM Press, August 2002, pp. 201–206.
[14] J. Kivinen, A.J. Smola, R.C. Williamson, Online learning with kernels, IEEE Transactions on Signal Processing 52 (8) (2002) 2165–2176.
[15] Y. Lan, Y.-C. Soh, G.-B. Huang, Constructive hidden nodes selection of extreme learning machine for regression, Neurocomputing 73 (16–18) (2010) 3191–3199.
[16] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Transactions on Neural Networks 17 (4) (2006) 879–892.
[17] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate on-line sequential learning algorithm for feedforward networks, IEEE Transactions on Neural Networks 17 (6) (2006) 1411–1423.
[18] G. Feng, G.-B. Huang, Q. Lin, R. Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, IEEE Transactions on Neural Networks 20 (8) (2009) 1352–1357.
[19] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16–18) (2007) 3056–3062.
[20] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16–18) (2008) 3460–3468.
[21] L. Ralaivola, F. d'Alché-Buc, Incremental support vector machine learning: a local approach, Lecture Notes in Computer Science, vol. 2130, 2001, pp. 322–329.
[22] S. Rüping, Incremental learning with support vector machines, in: N. Cercone, T. Lin, X. Wu (Eds.), Proceedings of the 2001 IEEE International Conference on Data Mining2001, pp. 641–642.
[23] N.A. Syed, H. Liu, K.K. Sung, Incremental learning with support vector machines, in: Proceedings of the Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999.
[24] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large cluster, Communications of the ACM 51 (1) (2008) 107–113.
[25] R. Lammel, Google's MapReduce programming model-revisited, Science of Computer Programming 70 (2008) 1–30.
[26] C.-T. Chu, S.K. Kim, Y.-A. Lin, Y. Yu, G.R. Bradski, A.Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, in: B. Scholkopf, J.C. Platt, T. Hoffman (Eds.), NIPS 19, MIT Press, 2006, pp. 281–288.
[27] Hadoop: Open source implementation of MapReduce 〈http://hadoop.apache.org〉.
[28] D. Borthakur, The Hadoop distributed file system: architecture and design, 2007.
[29] G. Fung, O.L. Mangasarian, Incremental support vector machine classification, in: Proceedings of the Second SIAM International Conference on Data Mining, Arlington, Virginia, USA, 2002, pp. 247–260.
[30] P.M. Murphy, D.W. Aha, UCI repository of machine learning databases, 1992.
[31] X. Xu, J. Jager, H.P. Kriegel, A fast parallel clustering algorithm for large spatial databases, Data Mining and Knowledge Discovery 3 (1999) 263–290.
[32] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, Evaluating MapReduce for multi-core and multiprocessor systems, in: Proceedings of the 13th International Symposium on High-Performance Computer Architecture (HPCA), Phoenix, AZ, 2007.
[33] V.N. Vapnik, Estimation of Dependencies Based on Empirical Data, Springer, Berlin, 1982.
[34] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.
[35] V.N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.

Qing He is a Professor in the Institute of Computing Technology, Chinese Academy of Sciences (CAS), and he is a Professor at the Graduate University of Chinese (GUCAS). He received the B.S degree from Hebei Normal University, Shijiazhang, PR China, in 1985, and the M.S. degree from Zhengzhou University, Zhengzhou, PR China, in 1987, both in mathematics. He received the Ph.D. degree in 2000 from Beijing Normal University in fuzzy mathematics and artificial intelligence, Beijing, PR China. Since 1987–1997, he has been with Hebei University of Science and Technology. He is currently a doctoral tutor at the Institute of Computing and Technology, CAS. His interests include data mining, machine learning, classification, fuzzy clustering.

Changying Du is an M.S. candidate student in the Institute of Computing Technology, Chinese Academy of Sciences (CAS). He received the B.S degree from the Department of Mathematics, Central South University, Changsha, PR China, in 2008. His research interests include machine learning, data mining, distributed classification and clustering, natural language processing and information retrieval.



Qun Wang is an M.S. candidate student in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include machine learning, data mining, distributed classification and clustering, natural language processing and information retrieval.

Fuzhen Zhuang is a Ph.D candidate student in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include machine learning, data mining, distributed classification and clustering, natural language processing. He has published several papers in some prestigious refereed journals and conference proceedings, such as IEEE Transactions on Knowledge and Data Engineering, ACM CIKM, SIAM SDM.

Zhongzhi Shi is a Professor in the Institute of Computing Technology, CAS, leading the Research Group of Intelligent Science. His research interests include intelligence science, multi-agent systems, semantic Web, machine learning and neural computing. He won a 2nd-Grade National Award at Science and Technology Progress of China in 2002, two 2nd-Grade Awards at Science and Technology Progress of the Chinese Academy of Sciences in 1998 and 2001, respectively. He is a senior member of IEEE, member of AAAI and ACM, Chair for the WG 12.2 of IFIP. He serves as Vice President for Chinese Association of Artificial Intelligence.