Classifying one billion data with a new distributed SVM algorithm

Thanh-Nghi Do and François Poulet

Abstract—The new incremental, parallel and distributed Support Vector Machine (SVM) algorithm using linear or non-linear kernels proposed in this paper aims at classifying very large datasets on standard personal computers (PCs). SVM and kernel related methods have shown to build accurate models but the learning task usually needs a quadratic program so that the learning task for large datasets requires large memory capacity and long time. We extend a recent Least Squares SVM (LS-SVM) proposed by Suykens and Vandewalle for building incremental, parallel and distributed SVM algorithm. The new algorithm is very fast and can handle very large datasets in linear or non-linear classification tasks on PCs. An example of the effectiveness is given with the linear classification into two classes of one billion datapoints in 20-dimensional input space in some minutes on ten PCs (3 GHz Pentium IV, 512 MB RAM).

Index terms—Support vector machines, Least squares support vector machine, Incremental learning, Parallel and distributed algorithm, Massive data classification, Machine learning, Data mining.

I. INTRODUCTION

In recent years, the size of data stored in the world doubles every nine months [18], so the need to extract knowledge from very large databases is increasing [11]. Data mining deals with the challenge of large datasets to identify valid, novel, potentially useful and ultimately understandable patterns in data. It uses different algorithms for classification, regression, clustering and association.

We are interested in SVM learning algorithms proposed by Vapnik [26] because they have shown practical relevance for classification, regression and novelty detection. Successful applications of SVMs have been reported for various fields, for example in face identification, text categorization and bioinformatics [15]. The approach is systematic and properly motivated by statistical learning theory. SVMs are the most well known algorithms of a class using the idea of kernel substitution [5]. SVM and kernel related methods have shown to build accurate models, so they have become increasingly popular data mining tools. In spite of the prominent properties of SVM, they can not deal easily with very large datasets. SVM solutions are obtained from quadratic programs, so that the computational cost of an SVM approach is at least square of the number of training datapoints and the memory requirement makes SVM intractable. There is a need to scale up learning algorithms to handle massive datasets on PCs. The effective heuristics to improve SVM learning task are to divide the original quadratic program into series of small problems [2], [20], [21] and [4]. The incremental learning methods [24], [3], [13] can handle massive datasets by updating solutions in growing training set without loading the whole dataset in main memory. The parallel and distributed algorithms [22] use PC networks to improve the performance of learning task on large datasets. The active learning algorithms [25], [9] choose interested datapoint subset (active set) to construct models.

In this paper, we propose a new incremental, parallel and distributed SVM algorithm using linear or non-linear kernels to classify very large datasets on standard PCs. The recent LS-SVM classifier proposed by Suykens and Vandewalle [23] changes the inequality constraints to equalities in the optimization problem of the SVM standard algorithm, thus the training task requires the solution of a system of linear equations, so that LS-SVM is very fast to train. We extend LS-SVM to construct a new algorithm that is very fast to build incremental, parallel and distributed SVM for classification tasks. Our new algorithm can classify one billion datapoints in 20-dimensional input space into two classes in some minutes on ten machines (3 GHz Pentium IV, 512 MB RAM).

We briefly summarize the content of the paper now. In section 2, we introduce LS-SVM classifiers. In section 3, we describe how to build the incremental learning algorithm with the LS-SVM algorithm for classifying large datasets on one PC. In section 4, we present parallel and distributed versions of the incremental LS-SVM. We present numerical test results in section 5 before the conclusion and future work.

Some notations are used in this paper. All vectors are column vectors unless transposed to row vector by a superscript. The inner dot product of two vectors, \( x \) and \( y \) is denoted by \( x \cdot y \). The 2-norm of the vector \( x \) is denoted by \( ||x|| \). The matrix \( A[m \times n] \) is \( m \) datapoints in the \( n \)-dimensional real space \( \mathbb{R}^n \). The classes \( +1, -1 \) of \( m \) datapoints are denoted by

Dr. François Poulet is currently Associate Professor at ESIEA KDD Lab, 38 rue des Docteurs Calmette et Guérin, 53000-Laval, France, poulet@esia-ouest.fr.

Dr. Thanh-Nghi Do is Lecturer at the college of information technology, Can Tho university, 1 Ly Tu Trong street, Can Tho city, Vietnam, dnghti@cit.ctu.edu.vn.
the diagonal matrix \( D[m \times m] \) of \(-I, +I\). \( e \) is the column vector of \( I \). \( w, b \) are the coefficients and the scalar of the hyper-plane. \( z \) is the slack variable and \( C \) is a positive constant. \( I \) denotes the identity matrix.

II. LEAST SQUARES SUPPORT VECTOR MACHINE

Let us consider a linear binary classification task, as depicted in Figure 1, with \( m \) datapoints \( x_i \) (\( i=1..m \)) in the \( n \)-dimensional input space \( \mathbb{R}^n \), represented by the \([m \times n]\) matrix \( A \), having corresponding labels \( y_i = \pm 1 \), denoted by the \([m \times 1]\) diagonal matrix \( D \) of \( \pm I \). \( D[i,i] = I \) if \( x_i \) is in class \(+1\), \( D[i,i] = -I \) if \( x_i \) is in class \(-1\).

For this problem, the SVMs try to find the best separating plane, i.e. furthest from both class \(+1\) and class \(-1\). It can simply maximize the distance or margin between two parallel supporting planes for each class.

The plane \((w.x - b = +1)\) supports for class \(+1\) if all datapoints in class \(+1\) are on the right side of that plane or \((w.x - b \geq 1)\) if \( y_i = +1 \). The supporting plane \((w.x - b = -1)\) for class \(-1\) similarly requires \( w.x_i - b \leq -1 \) if \( y_i = -1 \). The constraints can be simplified to:

\[
D(A.w - e.b) \geq e
\]

(1)

where \( e \) will be the column vector of \( I \).

The margin between these supporting planes is \( 2/||w|| \) (where \( ||w|| \) is the 2-norm of the vector \( w \)).

In the linearly inseparable case, the constraints must be relaxed to insure that each datapoint is not on the wrong side of its supporting plane, so that a nonnegative slack variable is added to the left part of the constraints (1). Then, any point \( x_i \) falling on the wrong side of its supporting plane is considered as an error (having corresponding slack value \( z_i > 0 \)).

Therefore, a SVM algorithm has to simultaneously maximize the margin and minimize the error. The standard SVM formulation with a linear kernel is given by the following quadratic program (2):

\[
\begin{align*}
\min \Psi(w, b, z) &= (1/2) ||w||^2 + cz \\
\text{s.t.} \quad D(A.w - e.b) + z &\geq e
\end{align*}
\]

(2)

where the slack variable \( z \geq 0 \) and the constant \( c > 0 \) is used to tune errors and margin size.

The plane \((w, b)\) is obtained by the solution of the quadratic program (2). Then, the classification function of a new datapoint \( x \) based on the plane is: \( \text{predict}(x) = \text{sign}(w.x - b) \).

SVM can use some other classification functions, for example a polynomial function of degree \( d \), a RBF (Radial Basis Function) or a sigmoid function. To change from a linear to non-linear classifier, one must only substitute a kernel evaluation in (2) instead of the original dot product.

More details about SVM and others kernel-based learning methods can be found in [1] and [5].

SVM solutions are obtained from the quadratic program (2), so that the computational cost of an SVM approach is at least square of the number of training datapoints and the memory requirement makes SVM intractable. The LS-SVM proposed by Suykens and Vandewalle has used the equality instead of the inequality constraints in the optimization problem (2) with a least squares 2-norm error into the objective function \( \Psi \) as follow:

- minimizing the errors with \((c/2)||z||^2\)
- using the equality constraints \( D(A.w - e.b) + z = e \)

Thus substituting for \( z \) from the constraint in terms \( w \) and \( b \) into the objective function \( \Psi \) of the quadratic program (2), we get an unconstraint problem (3):

\[
\begin{align*}
\min \Psi(w, b) &= (1/2)||w||^2 + (c/2)||e - D(A.w - e.b)||^2 \\
\Psi(w, b) &= ce^T(-A.w + e.b + D.e) = 0
\end{align*}
\]

(3)

(4) and (5) can be rewritten by the linear equation system (6):

\[
[w_1 \; w_2 \; \ldots \; w_n \; b]^T = \left( \frac{1}{c} I^\circ + E^T E \right)^{-1} E^T D e
\]

(6)

where \( E = [A - e] \), \( I^\circ \) denotes the \((n+1)x(n+1)\) diagonal matrix whose \((n+1)th\) diagonal entry is zero and the other diagonal entries are \( I \).
The LS-SVM formulation (6) requires thus only one solution of linear equations of \((n+1)\) variables \((w_1, w_2, \ldots, w_m, b)\) instead of the quadratic program (3). If the dimensional input space is small enough (less than \(10^6\)), even if there are millions datapoints, the LS-SVM algorithm is able to classify them in some minutes on a PC.

<table>
<thead>
<tr>
<th>TABLE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>THE LINEAR LS-SVM ALGORITHM</strong></td>
</tr>
<tr>
<td>Input:</td>
</tr>
<tr>
<td>- training dataset represented by (A) and (D) matrices</td>
</tr>
<tr>
<td>- constant (c &gt; 0) for tuning errors and margin size</td>
</tr>
<tr>
<td>Training:</td>
</tr>
<tr>
<td>- create the matrix (E = [A, -c])</td>
</tr>
<tr>
<td>- solve the linear equation system (6)</td>
</tr>
<tr>
<td>- obtain the optimal plane ((w, b)): (w \cdot x - b)</td>
</tr>
<tr>
<td>Classification of a new datapoint (x) based on the plane is:</td>
</tr>
<tr>
<td>(f(x) = \text{sign}(w \cdot x - b))</td>
</tr>
</tbody>
</table>

The table 1 presents the linear LS-SVM algorithm. The numerical test results [23] have shown that this algorithm gives test correctness compared to standard SVM like LibSVM [4] but the LS-SVM is much faster than standard SVMs. An example of the effectiveness is given with the linear classification into two classes of one million datapoints. The algorithm can deal with non-linear classification tasks: in input of the algorithm, the training dataset represented by \(A[m \times n]\) is replaced by the kernel matrix \(K[m \times m]\), where \(K\) is a non linear kernel created by whole dataset \(A\) and the support vectors being \(A\) too, e.g.:

- A degree \(d\) polynomial kernel of two datapoints \(x_i, x_j\):
  \(K[i,j] = (x_i \cdot x_j + 1)^d\)
- A radial basis kernel of two datapoints \(x_i, x_j\):
  \(K[i,j] = \exp(-\gamma \| x_i - x_j \|^2)\)

The LS-SVM algorithm using the kernel matrix \(K[m \times m]\) requires very large memory size and execution time. Reduced support vector machine (RSVM) proposed by Lee and Mangasarian [17] creates rectangular \((m) \times (d)\) kernel matrix of size \((s < m)\) by sampling, the small random datapoints \(S\) being a representative sample of the entire dataset (and RSVM uses it as a set of support vectors). RSVM reduces the problem size and has a good classification accuracy compared to standard SVM algorithms.

### III. Incremental Algorithm of the LS-SVM

Although the LS-SVM algorithm is fast and efficient to classify large datasets, it needs to load whole dataset in the memory. With a large dataset e.g. one billion datapoints in 20 dimensional input, LS-SVM requires 80 GB RAM. Any machine learning algorithm has some difficulties to deal with the challenge of large datasets. Our investigation aims at scaling up the LS-SVM algorithm to mine very large datasets on PCs (Pentium IV, 512 MB RAM).

The incremental learning algorithms are a convenient way to handle very large datasets because they avoid loading the whole dataset in main memory: only subsets of the data are considered at any one time and update the solution in growing training set.

Suppose we have a very large dataset decomposed into small blocks of rows \(A_i, D_i\). The incremental algorithm of the LS-SVM can simply incrementally compute the solution of the linear equation system (6). More simply, let us consider a large dataset split into two blocks of rows \(A_1, D_2\) and \(A_2, D_2\):

\[
A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \quad \text{et} \quad e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}
\]

\[
E = [A - e] = \begin{bmatrix} A_1 - e_1 \\ A_2 - e_2 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix}
\]

We illustrate how to incrementally compute the solution of the linear equation system (6) as follow:

\[
E^T De = E_1^T D_1 e_1 + E_2^T D_2 e_2 \quad \text{(7)}
\]

\[
E^T E = \begin{bmatrix} E_1^T \\ E_2^T \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} E_1^T E_1 \\ E_2^T E_2 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} \quad \text{(8)}
\]

\[
E^T E = E_1^T E_1 + E_2^T E_2 \quad \text{(9)}
\]

\[
\begin{bmatrix} w_1 & w_2 & \ldots & w_n & b \end{bmatrix}^T = \left( \frac{1}{c} I^p + \sum_{i=1}^{m} E_i^T E_i \right)^{-1} \sum_{i=1}^{m} E_i^T D_i e_i \quad \text{(10)}
\]
From the formulas (7), (9) and (10), we can deduce the formula (11) of the incremental LS-SVM algorithm with a very large dataset generally decomposed into \( k \) small blocks of rows \( A_1, D_1, \ldots, A_k, D_k \):

\[
\begin{bmatrix}
  \mathbf{w}_1 & \mathbf{w}_2 & \ldots & \mathbf{w}_n
\end{bmatrix}^T = \frac{1}{c} \left( \mathbf{I}^n + \sum_{i=1}^{k} \mathbf{E}^T \mathbf{E}_i \right)^{-1} \sum_{i=1}^{k} \mathbf{E}^T \mathbf{D}_i \mathbf{e}_i \tag{11}
\]

### Table II

#### The Incremental LS-SVM Algorithm

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- training dataset represented by ( k ) blocks:</td>
</tr>
<tr>
<td>( A_1, D_1, \ldots, A_k, D_k )</td>
</tr>
<tr>
<td>- constant ( c &gt; 0 ) for tuning errors and margin size</td>
</tr>
<tr>
<td>Training:</td>
</tr>
<tr>
<td>- init: ( \mathbf{E}^T \mathbf{E} = 0, d = \mathbf{E}^T \mathbf{D} e = 0 )</td>
</tr>
<tr>
<td>- for ( i = 1 ) to ( k ) do</td>
</tr>
<tr>
<td>- load ( A_i ) and ( D_i )</td>
</tr>
<tr>
<td>- compute ( \mathbf{E}^T \mathbf{E} = \mathbf{E}^T \mathbf{E} + \mathbf{E}^T_i \mathbf{E}_i )</td>
</tr>
<tr>
<td>- compute ( d = d + d_i ) (where ( d_i = \mathbf{E}^T_i \mathbf{D} e_v ))</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>- solve the linear equation system (11)</td>
</tr>
<tr>
<td>- obtain the optimal plane ((\mathbf{w}, b): \mathbf{w}_i, \mathbf{w}_2, \ldots, \mathbf{w}_n, b))</td>
</tr>
</tbody>
</table>

Classification of a new datapoint \( x \) based on the plane is:

\[ f(x) = \text{sign}(\mathbf{w} \cdot x - b) \]

Consequently, the incremental LS-SVM algorithm presented in table 2 can handle massive datasets on a PC. The accuracy of the incremental algorithm is exactly the same as the original one. If the dimension of the input space is small enough (less than \( 10^4 \)), even if there are billions of datapoints, the incremental LS-SVM algorithm is able to classify them on a simple PC (Pentium IV, 512 MB RAM). The algorithm only needs to store a small \((n+1) \times (n+1)\) matrix and two \((n+1) \times 1\) vectors in memory between two successive steps. The numerical test has shown the incremental LS-SVM algorithm using a linear kernel can classify one billion datapoints in 20-dimensional input into two classes in 21 minutes and 10 seconds (except the time needed to read data from disk) on a PC (Pentium-IV 3 GHz, 512 MB RAM).

### IV. Parallel and Distributed Incremental LS-SVM Algorithm

The incremental SVM algorithm described above is very fast to train in most of the cases and can deal with very large datasets on PCs. However it only runs on one single machine. We have extended it to build a parallel and distributed version on a computer network using the remote procedure calls (RPC) mechanism and the thread concept. The parallel and distributed algorithm benefits from the PCs’ performance of a computer network. It speeds up data loading task and computational cost.

First, we distribute a large dataset \( A, D \) decomposed into small blocks of rows \( A_i, D_i \) on remote servers. The remote servers compute independently, incrementally the sums of \( \mathbf{E}^T_i \mathbf{E}_i \) and \( d_i = \mathbf{E}^T_i \mathbf{D} e_v \). Then a client machine will use these sums to solve the linear equation system (11).

The RPC protocol does not support asynchronous communication. A synchronous request-reply mechanism in RPC requires that the client and server are always available and functioning (i.e. the client or server is not blocked). The client can issue a request and must wait for the server’s response before continuing its own processing. Therefore, we have parallelized waiting on the client side with the set of threads. The parallel and distributed incremental LS-SVM algorithm is presented in figure 2.

The accuracy of the new algorithm is exactly the same as the original one.

### V. Results

The software program is written in C/C++ on PC Linux, we have also used the high performance linear algebra library, Lapack++ [10] to benefit by the high speed of computational matrix. Thus, the software program is able to deal with large datasets in linear and non-linear classification tasks. We focus on numerical tests with large datasets from ten thousands to one billion datapoints (c.f. table 3) generated by the RingNorm program [6]. It is a 20 dimensional, 2 class classification example. Each class is drawn from a multivariate normal distribution. Class 1 has mean equal to zero and covariance 4 times the identity. Class 2 (considered as \(-1\)) has unit covariance with mean = \(2/\sqrt{20}\). We use them to estimate the execution time for varying the block size, dataset size and the number of PCs. The execution time reported here has been evaluated on PCs (3GHz Pentium IV, 512 MB RAM running Linux Fedora Core 3). We have only measured the computational time without the time needed to read data from disk.

First, we have split the datasets into small blocks of rows and then distributed them on PCs. We have varied the block size for each incremental step from one thousand datapoints to ten millions datapoints (completely in main memory), then an increasing part of the block must be swapped on the secondary memory (on the hard disk) and finally the whole swap space is used. The highest block sizes is too large to fit in memory, the program is always swapping parts of the block between main memory and secondary memory and does not perform any more calculation.

Therefore we have finally split the datasets into small blocks of five thousands datapoints to reach good
performances on PCs (3GHz Pentium IV, 512 MB RAM).

With a non linear classification task of the RingNorm dataset, [9] has found that SVM algorithms need about 250 support vectors to non linearly classify this dataset. Then, we have also tried to tune the number of support vectors, we could obtain good results by only using 200 random datapoints being a representative sample of the entire dataset (as support vectors). The RBF kernel functions are constructed with the whole dataset and 200 random datapoints. The LS-SVM algorithm is able to linearly and non-linearly classify datasets in parallel and distributed incremental way on PCs.

An example of the effectiveness is given with one billion datapoints in 20-dimensional input space classified into two classes on ten PCs (3 GHz Pentium IV, 512 MB RAM) in 2 minutes 7 seconds using a linear kernel and 3 hours 8 minutes with a RBF kernel. The results obtained have demonstrated the effectiveness of the new algorithm to deal with very large datasets on PCs.

VI. EXTENSIONS

The LS-SVM algorithm described in the previous sections incrementally compute:

\[ E_1^T E_1, d_1 \]

\[ E_2^T E_2, d_2 \]

\[ E_3^T E_3, d_3 \]

\[ E_4^T E_4, d_4 \]

\[ E_5^T E_5, d_5 \]

\[ E_6^T E_6, d_6 \]

A1,D1 A2,D2 A3,D3 A4,D4 A5,D5 A6,D6

PC1 PC2 PC3 PC4 PC5 PC6

Fig. 2. The parallel and distributed incremental LS-SVM algorithm

The results presented in Table 4 have shown that the algorithm has linear dependences on the number of PCs, size of datasets and a second order of the number of dimensions (or a second order of the number of support vectors in a non linear kernel case). This is exactly the same as the theoretical algorithm complexity. Furthermore, the communication cost takes about one second when the dataset dimension is less than 100.

An example of the effectiveness is given with one billion datapoints in 20-dimensional input space classified into two classes on ten PCs (3 GHz Pentium IV, 512 MB RAM) in 2 minutes 7 seconds using a linear kernel and 3 hours 8 minutes with a RBF kernel. The results obtained have demonstrated the effectiveness of the new algorithm to deal with very large datasets on PCs.

### Table III

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nb. dim</th>
<th>Training set</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>10 000</td>
<td>1 000</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>50 000</td>
<td>5 000</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>100 000</td>
<td>10 000</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>500 000</td>
<td>50 000</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>1 000 000</td>
<td>100 000</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>5 000 000</td>
<td>500 000</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>10 000 000</td>
<td>1 000 000</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>50 000 000</td>
<td>5 000 000</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>100 000 000</td>
<td>10 000 000</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>500 000 000</td>
<td>50 000 000</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>1 000 000 000</td>
<td>100 000 000</td>
</tr>
</tbody>
</table>
can handle datasets with a very large number of datapoints and smaller number of dimensions. But some applications (like bioinformatic or text mining) require datasets with a very large number of dimensions and few training datapoints. Thus, the $(n+1)\times(n+1)$ matrix $E^TE$ is too large and the solution of the linear equation system of $(n+1)$ variables ($w$, $b$) has a high computational cost. To adapt the algorithm to this problem, we have applied the Sherman-Morrison-Woodbury formula [14] to the linear equation system (6), thus the solution of the new formula [8] depends on the inversion of the $(m)\times(m)$ matrix $EE^T$ instead of the $(n+1)\times(n+1)$ matrix $E^TE$. The cost of storage and computation depends on the number of training data. This formulation can handle datasets with very large number of dimensions and few training data.

We have constructed the column-incremental algorithm the same way as the row-incremental one. The data are split in blocks of columns $E_i$ and then we perform the incremental computation of $EE^T$. For each step, we only need to load the $(m)\times(blocksize)$ matrix $E_i$ for computing $EE^T$. Between two incremental steps, we need to store in memory the $(m)\times(m)$ matrix $EE^T$ although the order of the dimensional input space is very high.

With these two formulations of the linear incremental LS-SVM, we are able to train very large datasets (large either in number of training data or number of dimensions, but not yet both simultaneously). We have used them to classify bio-

<table>
<thead>
<tr>
<th>Linear classification</th>
<th>Non linear classification using RBF kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>Accuracy (%)</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>Accuracy (%)</td>
</tr>
<tr>
<td>Dataset 1</td>
<td>0.0012 78.10 %</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>0.006 76.68 %</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>0.011 76.52 %</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>0.056 76.51 %</td>
</tr>
<tr>
<td>Dataset 5</td>
<td>0.128 76.68 %</td>
</tr>
<tr>
<td>Dataset 6</td>
<td>0.828 76.68 %</td>
</tr>
<tr>
<td>Dataset 7</td>
<td>1.288 76.68 %</td>
</tr>
<tr>
<td>Dataset 8</td>
<td>6.37 76.68 %</td>
</tr>
<tr>
<td>Dataset 9</td>
<td>12.9 76.68 %</td>
</tr>
<tr>
<td>Dataset 10</td>
<td>63.2 76.68 %</td>
</tr>
<tr>
<td>Dataset 11</td>
<td>126.4 76.68 %</td>
</tr>
</tbody>
</table>

Table IV: The Classification Results with Execution Time Reported on 10 PCs

![Fig. 3. CPU time of the parallel and distributed incremental LS-SVM algorithm](image)

Woodbury formula [14] to the linear equation system (6), thus the solution of the new formula [8] depends on the inversion of the $(m)\times(m)$ matrix $EE^T$ instead of the $(n+1)\times(n+1)$ matrix $E^TE$. The cost of storage and computation depends on the number of training data. This formulation can handle datasets with very large number of dimensions and few training data.

We have constructed the column-incremental algorithm the same way as the row-incremental one. The data are split in blocks of columns $E_i$ and then we perform the incremental computation of $EE^T$. For each step, we only need to load the medical datasets with interesting results concerning the learning time and classification accuracy [7]. The parallel and distributed version of the column-incremental LS-SVM algorithm is also implemented.

For mining massive datasets with simultaneously large number (at least $10^5$) of datapoints and attributes, there are at least two problems to solve: the learning time increases dramatically with the training data size and the memory requirement increases according to data size. Although the LS-SVM algorithm and its incremental versions can in fact
retire and add new training data efficiently these algorithms need to store and invert a matrix with size \((m x m)\) (or \((n+1)x(n+1))\). This requires too much main memory and very high computational time. To scale LS-SVM to large datasets, we have applied the boosting approach to the LS-SVM algorithm. This solution brings out two advantages. The first one is to be able to solve the scaling problem and the second one is the preservation of the classification accuracy. More details about boosting can be found in [12] or on the web site www.boosting.org. We briefly explain the mechanism of boosting of LS-SVM. In years 90’ boosting technique introduced by Freund and co-workers is a general method for improving the accuracy of any given learning algorithm. The boosting algorithm calls repeatedly a given weak or base learning algorithm \(k\) times so that each boosting step concentrates mostly on the errors produced by the previous step. For achieving this goal, we need to maintain a distribution weights over the training points. Initially, all weights are set equally and at each boosting step the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. The final hypothesis is a weighted majority vote of \(k\) weak hypotheses. Alternately, we consider the LS-SVM algorithm as a weak algorithm thus at each boosting step we can sample a subset of the training set according to the distribution weights over the training examples. Note that LS-SVM only classifies the subset (less than the original training set). The subset size is in opposite proportion to the number of boosting steps. Row-incremental or column-incremental LS-SVM can be adapted to solve large sizes of subset. Boosting of LS-SVM has shown performances concerning the training time, the memory requirements and the classification accuracy [8].

VII. CONCLUSION AND FUTURE WORK

We have presented a new parallel and distributed incremental SVM algorithm being able to deal with very large datasets in linear and non-linear classification tasks on PCs. We have extended the recent LS-SVM algorithm proposed by Suyken and Vandewalle [23] to build incremental, parallel and distributed SVM. The accuracy of the new algorithm is exactly the same as the original one but its complexity is linearly dependent on the number of machines, size of datasets and a second order of the number of dimensions. The algorithm also requires to store a \((n+1)x(n+1))\) matrix and two \((n+1)x1)\) vectors in memory (where \(n\) is the number of dimensions or the number of support vectors in a non linear kernel case).

We focus on numerical tests with large datasets generated by the RingNorm program. Our new algorithm can classify one billion datapoints in 20-dimensional input space into two classes on ten PCs (3 GHz Pentium IV, 512 MB RAM, Linux Fedora Core 3) in 2 minutes 7 seconds using a linear kernel and 3 hours 8 minutes with a RBF kernel. A new version of the algorithm is also implemented by XML-RPC [16] to allow the parallel and distributed operations over any XML-capable transport protocol, typically over HTTP. The software program could then be distributed on different kind of machines, for example on a set of various remote PCs, Unix stations or any other computer reachable via the web.

We have used the Sherman-Morrison-Woodbury formula [14] to adapt the LS-SVM to process datasets with a very large number of dimensions. We have extended this idea by applying boosting to LS-SVM for mining massive datasets with simultaneously very large number of datapoints and dimensions.

In general, a complex non-linear classification task needs large number of support vectors. This requires large number of random datapoints from entire dataset for creating the rectangular kernel matrix in input. The algorithm must work on large number of dimensions and thus it is intractable. A forthcoming improvement will be to combine our method with other machine learning algorithms to construct another approach that can deal with a complex non-linear classification task.

REFERENCES