Distributed Incremental wLPSVM Learning

Lei Zhu*, Tao Ban†, Kazushi Ikeda‡, Paul Pang*and Abdolhossein Sarrafzadeh*
*Unitec Institute of Technology, New Zealand
†National Institute of Information and Communications Technology, Japan
‡NARA Institute of Science and Technology, Japan

***Abstract*—Weighted linear proximal support vector machine (wLPSVM) is known as an efficient binary classification algorithm with good accuracy and class-imbalance robustness. In this work, original batch wLPSVM is facilitated with distributed incremental learning capability, which allows simultaneously learning from multiple streaming data sources that are geographically distributed. In our approach, incremental and distributed learning are solved as a merging problem at the same time. A new wLPSVM expression is derived. In the new expression, knowledge from samples are presented as a set of class-wised core matrices, and merging knowledge from two subsets of data can be simply accomplished by matrix addition. With the new expression, we are able to conduct incremental and distributed learning at the same time via merging knowledge from multiple incremental stages and multiple data sources.**

I. Introduction

Incremental learning and parallel learning has been extensively studied for decades. Incremental learning which updates the existing knowledge according to newly arrived training samples is suitable for dealing with the streaming problem. On the other hand, parallel learning which distributes computational costs among multiple cores or machines in a cluster, serves as a natural solution when learning task is large in scale. In the era of BigData, more and more learning tasks by nature are both large-scale and streaming, thus these applications call for algorithms that are capable of solving both streaming and large-scale problem at the same time. Although there have been many incremental and parallel learning algorithms, but most of them focus only on one aspect, either streaming or large-scale.

In this work, we update wLPSVM, an efficient batch binary classification algorithm with class-imbalance robustness, into an incremental parallel algorithm.

When take a closer look at the parallel learning, we can see there are different levels of parallelization, depending on how independently each node can execute its job. Early days studies [1], [2] mainly concentrate on shared-memory multiprocessor (SMP) systems. In these algorithms, intensive communication are required between slave nodes and master node, because the processes at slave nodes rely heavily on global information which maintained at the master node and slave nodes update global information frequently. Due to such constraint, the scalability for SMP parallelization is very limited. And clearly, SMP gives single entrance for training data.

Some recent parallelization works [3], [4] based on standard SVM are capable of distribute training load among multiple machines. However, due to the iterative nature of SVM optimization, for one set of data, slave nodes still need multiple rounds communication with the master node, and the communication overhead is positive correlated to the number of instance. Thus, these algorithms are suitable for clusters in which machines are interconnected with high speed links. And each cluster gives single entrance for training data.

The parallel learning with highest degree of freedom is known as distributed learning [5], in which each learning node can absorb and learn from local training data independently, with affordably low communication overhead between nodes, a global result can be obtain upon all local data learned. Algorithms in this type are capable of handling applications that has many geographically distributed data sources. As only learning result from local data rather than the raw data itself is transferred between nodes, privacy for some critical data source is also naturally preserved. Our approach and some others [6], [7] fall into this category. Furthermore, we can see that algorithm with higher parallelization level can be easily adopted into lower level scenario, such as distributed learning algorithm can be deployed onto SMP, but not vice versa.

wLPSVM is the weighted version of Linear Proximal Support Vector Machine (LPSVM). Due to its classification modeling, wLPSVM can be implemented by matrix computation in a few fixed steps. Thus it is much more efficient than classic SVM, whose solution requires iterative optimization and number of iteration is usually unknown before the system converge. However, initial wLPSVM is still a serial batch algorithm. In order to facilitate wLPSVM with distributed and incremental learning capability, we study the way of merging wLPSVM from different data.

In our approach, a new wLPSVM expression is derived. In the new expression, knowledge from samples are presented as a set of class-wised core matrices, and merging knowledge from two subsets of data can be simply accomplished by matrix addition. With the new expression, we are able to conduct incremental and distributed learning at the same time via merging knowledge from multiple incremental stages and multiple data sources. Moreover, because of the class-wised splitting, the class weight is represented as real value coefficient instead of the original weighting matrix form, thus the class weight can be updated during the incremental learning. The weight of classes can be real-time balanced against current class-imbalance.

In summary, our proposed algorithm allows: 1) wLPSVM model to be updated at any time according to instances being added and/or removed; 2) the weighting for both classes is

balanced in real-time; 3) each learning node only works with local data source, in case of multiple distributed data sources; and 4) the distributed incremental learning result is exactly equivalent to that of batch learning.

II. Linear Proximal SVM

In this section, we introduce Linear Proximal SVM (LPSVM) and its counter class-imbalance variant, wLPSVM, which is the base algorithm we working on.

A. Batch LPSVM

Let $\boldsymbol{\mathcal{S}} = \{(\boldsymbol{x}_1, y_i), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ be a given training dataset, $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n\end{bmatrix}^T \in \mathbb{R}^{n\times d}$ and $\boldsymbol{Y} = \begin{bmatrix}y_1 & y_2 & \cdots & y_n\end{bmatrix}^T \in \{+1, -1\}^{n\times 1}$ as its instance matrix and corresponding label vector, respectively. A classic SVM [8] learns a class separating plane

$$\boldsymbol{x}^T\boldsymbol{w} + b = 0, \tag{1}$$

which lies midway between two parallel bounding planes formulated as,

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{w} + b &= +1\\ \boldsymbol{x}^T\boldsymbol{w} + b &= -1.\end{aligned} \tag{2}$$

In practice, bounding planes (2) bound two classes often with some non-negative errors ξ_i

$$\begin{aligned}\boldsymbol{x}_i^T\boldsymbol{w} + b + \xi_i &\geq +1 \quad \text{for} \quad y_i = +1\\ \boldsymbol{x}_i^T\boldsymbol{w} + b - \xi_i &\leq -1 \quad \text{for} \quad y_i = -1.\end{aligned} \tag{3}$$

Here, the distance between these two planes equals to $\frac{2}{\|\boldsymbol{w}\|}$, which is called the "margin" in literature. The $\boldsymbol{w}$ and b in (1) and (2) are obtained by solving an optimization problem

$$\begin{aligned}\min \quad & \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{i=1}^{n}\xi_i\\ \text{s.t.} \quad & y_i(\boldsymbol{x}_i^T\boldsymbol{w} + b) + \xi_i \geq 1 \quad \xi_i \geq 0 \quad \forall i \in \{1, \cdots, n\},\end{aligned} \tag{4}$$

where $\frac{\|\boldsymbol{w}\|^2}{2}$ is for maximizing the margin, and $\sum_{i=1}^{n}\xi_i$ for minimizing the total training error. The regularization parameter C balances the importance of error and margin [9]. In practice, the dual problem of (4) is solved to obtain a SVM classifier [10], [11].

LPSVM [12] models a binary classification as a regularized least square problem, which simplifies the above SVM optimization and results in an extremely efficient training algorithm. The optimization of LPSVM is given as,

$$\begin{aligned}\min \quad & \frac{C}{2}\|\boldsymbol{\xi}\|^2 + \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{w} + b^2)\\ \text{s.t.} \quad & \boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{e}b) + \boldsymbol{\xi} = \boldsymbol{e},\end{aligned} \tag{5}$$

where $\boldsymbol{\xi} \in \mathbb{R}^{n\times 1}$ refers to the vector of training errors, $\boldsymbol{D} = diag(\boldsymbol{Y}) \in \mathbb{R}^{n\times n}$ denotes a diagonal matrix of class labels, and $\boldsymbol{e} = [1, \ldots, 1]^T \in \mathbb{R}^{n\times 1}$. By (5), classic LPSVM seeks a class separating plane

$$\boldsymbol{x}^T\boldsymbol{w} - b = 0 \tag{6}$$

which lies midway between two parallel proximal planes

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{w} - b &= +1\\ \boldsymbol{x}^T\boldsymbol{w} - b &= -1.\end{aligned} \tag{7}$$

In contrast to classic SVM, LPSVM in (5) replaces the inequality constraint of (4) with an equality condition. As a result, the planes in (7) are not bounding planes anymore, but can be seen as "proximal" planes around which the instances of each class are clustered, as we have

$$\begin{aligned}\boldsymbol{x}_i^T\boldsymbol{w} - b + \xi_i &= +1 \quad \text{for} \quad y_i = +1\\ \boldsymbol{x}_i^T\boldsymbol{w} - b - \xi_i &= -1 \quad \text{for} \quad y_i = -1,\end{aligned} \tag{8}$$

and the error variable ξ_i is not necessary to be nonnegative. These planes (7) are pushed as far apart as possible by the term of $(\boldsymbol{w}^T\boldsymbol{w} + b^2)$ in the LPSVM optimization (5), and the total training error are minimized by the term of $\|\boldsymbol{\xi}\|^2$. To summarize from a geometrical perspective, both SVM and LPSVM learn a separating plane that lies in the midway of two parallel planes which are pushed as far apart as possible. Parallel planes in classic SVM case bound two classes, and in LPSVM these planes perform as "proximal" planes around which the instances of each class are clustered.

To solve (5), the equality constraint is substituted to the objective function, thus (5) is transformed to an unconstrained optimization problem,

$$\min \quad G = \frac{C}{2}\|\boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{e}b) - \boldsymbol{e}\|^2 + \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{w} + b^2). \tag{9}$$

Set the partial derivatives of G to 0, we have

$$\begin{aligned}\frac{\partial G}{\partial \boldsymbol{w}} &= C\boldsymbol{X}^T\boldsymbol{D}(\boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{e}b) - \boldsymbol{e}) + \boldsymbol{w}\\ &= C\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w} - C\boldsymbol{X}^T\boldsymbol{e}b - C\boldsymbol{X}^T\boldsymbol{D}\boldsymbol{e}\\ &= 0\\ \frac{\partial G}{\partial b} &= -C\boldsymbol{e}^T\boldsymbol{D}(\boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{e}b) - \boldsymbol{e}) + b\\ &= -C\boldsymbol{e}^T\boldsymbol{X}\boldsymbol{w} + C\boldsymbol{e}^T\boldsymbol{e}b + b + C\boldsymbol{e}^T\boldsymbol{D}\boldsymbol{e}\\ &= 0,\end{aligned} \tag{10}$$

where $\boldsymbol{D}\boldsymbol{D} = \boldsymbol{I}$ and $\boldsymbol{e}^T\boldsymbol{e} = \|\boldsymbol{e}\|^2 = n$.

Solving the linear system (10), the solution of LPSVM optimization (i.e., (5) is obtained,

$$\begin{aligned}\begin{bmatrix}\boldsymbol{w}\\ b\end{bmatrix} &= \begin{bmatrix}\boldsymbol{X}^T\boldsymbol{X} + \frac{\boldsymbol{I}}{C} & -\boldsymbol{X}^T\boldsymbol{e}\\ -\boldsymbol{e}^T\boldsymbol{X} & n + \frac{1}{C}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{X}^T\boldsymbol{D}\boldsymbol{e}\\ -\boldsymbol{e}^T\boldsymbol{D}\boldsymbol{e}\end{bmatrix}\\ &= \left[\frac{\boldsymbol{I}}{C} + \begin{bmatrix}\boldsymbol{X}^T\\ -\boldsymbol{e}^T\end{bmatrix}\begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}\right]^{-1}\begin{bmatrix}\boldsymbol{X}^T\boldsymbol{D}\boldsymbol{e}\\ -\boldsymbol{e}^T\boldsymbol{D}\boldsymbol{e}\end{bmatrix}.\end{aligned} \tag{11}$$

Let $\boldsymbol{E} = \begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}$ and $\boldsymbol{O} = \begin{bmatrix}\boldsymbol{w}\\ b\end{bmatrix}$, (11) can be formulated as

$$\boldsymbol{O} = (\frac{\boldsymbol{I}}{C} + \boldsymbol{E}^T\boldsymbol{E})^{-1}\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{e}. \tag{12}$$

To simplify (12), we use $\boldsymbol{M}$ and $\boldsymbol{v}$ to denote the matrix term $\frac{\boldsymbol{I}}{C} + \boldsymbol{E}^T\boldsymbol{E}$ and vector term $\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{e}$ in (12) respectively. Then, (12) can be rewritten as

$$\boldsymbol{O} = \boldsymbol{M}^{-1}\boldsymbol{v}, \tag{13}$$

and the LPSVM decision function is obtained

$$f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w} - b = \begin{bmatrix}\boldsymbol{x}^T & -1\end{bmatrix}\boldsymbol{O} \begin{cases} > 0 & \Rightarrow y = +1 \\ < 0 & \Rightarrow y = -1. \end{cases} \tag{14}$$

In summary, the algorithm of batch LPSVM can be stated as Algorithm 1.

Algorithm 1 Batch LPSVM Algorithm [12]

Input: Instance matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$; Class label vector $\boldsymbol{Y} \in \{+1, -1\}^{n\times 1}$; Regularization parameter C.
Output: $\boldsymbol{O} \in \mathbb{R}^{(d+1)\times 1}$.
1: Form identity matrix $\boldsymbol{I} \in \mathbb{R}^{(d+1)\times(d+1)}$ and column vector $\boldsymbol{e} = [1, \ldots, 1]^T \in \mathbb{R}^{n\times 1}$;
2: Expand the instance matrix $\boldsymbol{X}$ into $\boldsymbol{E}$, $\boldsymbol{E} = \begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}$;
3: Transform the class label vector $\boldsymbol{Y}$ into diagonal matrix $\boldsymbol{D}$, $\boldsymbol{D} = diag(\boldsymbol{Y})$;
4: Compute $\boldsymbol{M} = \frac{\boldsymbol{I}}{C} + \boldsymbol{E}^T\boldsymbol{E}$ and $\boldsymbol{v} = \boldsymbol{E}^T\boldsymbol{D}\boldsymbol{e}$;
5: Compute $\boldsymbol{O} = \boldsymbol{M}^{-1}\boldsymbol{v}$.

B. wLPSVM for Class Imbalance Learning

The class imbalance problem arises when samples of the class of interest are relatively rare as compared with the other class. The total training error $\|\boldsymbol{\xi}\|^2$ in (5) comes from two classes, the error thus can be represented as $\|\boldsymbol{\xi}\|^2 = \|\boldsymbol{\xi}_+\|^2 + \|\boldsymbol{\xi}_-\|^2$. In the presence of class imbalance, the LPSVM optimization (5) has $\|\boldsymbol{\xi}_+\|^2 << \|\boldsymbol{\xi}_-\|^2$. As a result, LPSVM shifts its positive class proximal plane away from the separating plane, which enlarges the margin at the price of an augmented $\|\boldsymbol{\xi}_+\|^2$. Consequently, the separating plane biases to the positive class, and results in the worse recognition of the positive class.

For LPSVM class imbalance learning, Fung et. al proposed a weighted LPSVM (wLPSVM) [13], [14], in which the classic LPSVM optimization (5) is revised to

$$\begin{array}{ll} \min & \frac{C}{2}\boldsymbol{\xi}^T\boldsymbol{N}\boldsymbol{\xi} + \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{w} + b^2) \\ \text{s.t.} & \boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{e}b) + \boldsymbol{\xi} = \boldsymbol{e}, \end{array} \tag{15}$$

where $\boldsymbol{N}$ is a diagonal weighting matrix with,

$$\boldsymbol{N}_{ii} = \begin{cases} \sigma_+ & \text{if } y_i = +1 \\ \sigma_- & \text{if } y_i = -1, \end{cases} \tag{16}$$

in which the class-wise weight σ is used, σ_+ for the positive class and σ_- for the negative class, to balance the impacts of two classes to the LPSVM separating plane. In practice, σ is determined by the number of samples for each class. For example in [15], σ_+ and σ_- are calculated as the ratio of the corresponding class size (i.e., l_- or l_+) to the size of the whole dataset,

$$\begin{aligned} \sigma_+ &= l_-/(l_+ + l_-) \\ \sigma_- &= l_+/(l_+ + l_-). \end{aligned} \tag{17}$$

By a similar approach as (9)-(12), (15) can be solved and the wLPSVM solution is obtained as,

$$\boldsymbol{O} = (\frac{\boldsymbol{I}}{C} + \boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E})^{-1}\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{N}\boldsymbol{e}. \tag{18}$$

Algorithm 2 presents the batch wLPSVM algorithm that implements (18).

Algorithm 2 Batch wLPSVM Algorithm [13]

Input: Instance matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$; Class label vector $\boldsymbol{Y} \in \{+1, -1\}^{n\times 1}$; Regularization parameter C.
Output: $\boldsymbol{O} \in \mathbb{R}^{(d+1)\times 1}$.
1: Calculate weights σ_+ and σ_- following weighting function such as (17);
2: Form weight matrix $\boldsymbol{N}$ following (16);
3: Expand the instance matrix $\boldsymbol{X}$ into $\boldsymbol{E}$, $\boldsymbol{E} = \begin{bmatrix}\boldsymbol{X} & -\boldsymbol{e}\end{bmatrix}$;
4: Transform the class label vector $\boldsymbol{Y}$ into diagonal matrix $\boldsymbol{D}$, $\boldsymbol{D} = diag(\boldsymbol{Y})$;
5: Compute $\boldsymbol{M} = \frac{\boldsymbol{I}}{C} + \boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E}$ and $\boldsymbol{v} = \boldsymbol{E}^T\boldsymbol{D}\boldsymbol{N}\boldsymbol{e}$;
6: Compute $\boldsymbol{O} = \boldsymbol{M}^{-1}\boldsymbol{v}$.

III. Proposed Distributed Incremental wLPSVM

In order to train a wLPSVM in incremental and distributed manner, we conduct distributed updating of $\boldsymbol{M}$ and $\boldsymbol{v}$, and then compute $\boldsymbol{O}$ as above. As we can see, the computational cost for computing $\boldsymbol{O}$ from $\boldsymbol{M}$ and $\boldsymbol{v}$ is a constant, if d is given. When the training data is huge and streaming, the major computational cost is spend on computing $\boldsymbol{M}$ and $\boldsymbol{v}$, as their complexity increases with the number of instances.

We consider incremental and distributed learning of $\boldsymbol{M}$ and $\boldsymbol{v}$ as one problem: reformulate these terms in an easily mergeable way. Take $\boldsymbol{M}$ for instance, if we can obtain $\boldsymbol{M}$ on data $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{S}}_a \cup \boldsymbol{\mathcal{S}}_b$ from simply merging $\boldsymbol{M}_a$ (on $\boldsymbol{\mathcal{S}}_a$) and $\boldsymbol{M}_b$ (on $\boldsymbol{\mathcal{S}}_b$) i.e., $\boldsymbol{M} = f(\boldsymbol{M}_a, \boldsymbol{M}_b)$, then both incremental and distributed learning are solved. For incremental learning, we let the system to compute $\boldsymbol{M}$ and $\boldsymbol{v}$ terms for newly incoming data and then merge them with existing terms; and for distributed learning, we let each node to calculate $\boldsymbol{M}$ and $\boldsymbol{v}$ terms from their local data, and then merge all local terms into a global model.

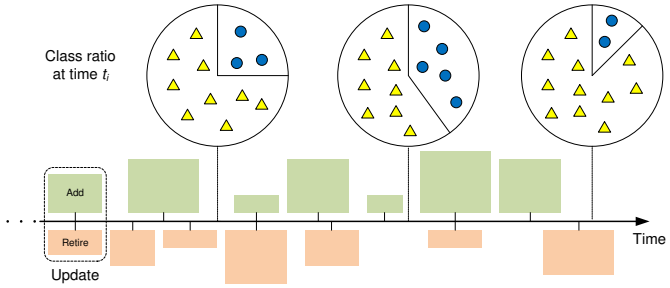

Fig. 1. Class imbalance variation in incremental learning

It is also worth notice that the weighting of two classes also need to be updated in incremental learning, as the class ratio may change over time during continuous data adding and removing, Figure 1 gives an demonstration. However, the weight for two classes in wLPSVM model is in presentation of a diagonal matrix $\boldsymbol{N}$, and this weighting matrix is also involved in multiplication with other matrices. This makes weighting update nearly impossible in the original wLPSVM form (18). Thus we need consider making weight update-able in our $\boldsymbol{M}$ and $\boldsymbol{v}$ reformulation. From now on, we introduce how to reformulate these terms and how to merge.

A. wLPSVM Reformulation

If leave weight $\boldsymbol{N}$ updating aside, i.e., consider σ_+ and σ_- as constant. The merging for $\boldsymbol{M}$ and $\boldsymbol{v}$ are straightforward.

Using basic matrix block computation, we can easily know that

Lemma 1: Let $\begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix} = \begin{bmatrix}\boldsymbol{X}_a & \boldsymbol{Y}_a \\ \boldsymbol{X}_b & \boldsymbol{Y}_b\end{bmatrix}$, $\boldsymbol{E} = \begin{bmatrix}\boldsymbol{E}_a \\ \boldsymbol{E}_b\end{bmatrix} = \begin{bmatrix}\boldsymbol{X}_a & -\boldsymbol{e} \\ \boldsymbol{X}_b & -\boldsymbol{e}\end{bmatrix}$, $\boldsymbol{D} = \begin{bmatrix}\boldsymbol{D}_a & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}_b\end{bmatrix}$ and $\boldsymbol{N} = \begin{bmatrix}\boldsymbol{N}_a & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{N}_b\end{bmatrix}$. Then,

$$\begin{aligned}
&\boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E} = \boldsymbol{E}_a^T\boldsymbol{N}_a\boldsymbol{E}_a + \boldsymbol{E}_b^T\boldsymbol{N}_b\boldsymbol{E}_b \\
&\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{N}\boldsymbol{e} = \boldsymbol{E}_a^T\boldsymbol{D}_a\boldsymbol{N}_a\boldsymbol{e} + \boldsymbol{E}_b^T\boldsymbol{D}_b\boldsymbol{N}_b\boldsymbol{e} \\
&\boldsymbol{E}^T\boldsymbol{E} = \boldsymbol{E}_a^T\boldsymbol{E}_a + \boldsymbol{E}_b^T\boldsymbol{E}_b \\
&\boldsymbol{E}^T\boldsymbol{e} = \boldsymbol{E}_a^T\boldsymbol{e} + \boldsymbol{E}_b^T\boldsymbol{e}.
\end{aligned} \tag{19}$$

The first two formulas actually merge $\boldsymbol{M}_a$, $\boldsymbol{M}_b$ into $\boldsymbol{M}$ and $\boldsymbol{v}_a$, $\boldsymbol{v}_b$ into $\boldsymbol{v}$, in condition that $\boldsymbol{\mathcal{S}}_a$ and $\boldsymbol{\mathcal{S}}_b$ has the same class ratio, i.e., $\boldsymbol{N}_a$, $\boldsymbol{N}_b$ and $\boldsymbol{N}$ consist the same σ_+ and σ_-. From now on, the question remains is how to update σ_+ and σ_-.

To make σ_+ and σ_- easy to change (update), we consider extract them from matrix form $\boldsymbol{N}$, and turn them into real number coefficient. By applying Lemma 1 repeatedly, we can see that any left hand term in (20) can be expressed as a summation of all instance-wised terms. Thus all these terms are independent to the ordering of instances, we have

Lemma 2: Let $\boldsymbol{\mathcal{S}}$ be a dataset with n samples, i.e., $\boldsymbol{\mathcal{S}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2) \ldots, (\boldsymbol{x}_n, y_n)\}$. $\eta = \begin{bmatrix}1 & 2 & \cdots & i & \cdots & j & \cdots & n\end{bmatrix}$ identifies a sequence of data samples. Applying η to the dataset, we form instance matrix $\boldsymbol{X}^{\boldsymbol{\eta}} = \begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_i & \cdots & \boldsymbol{x}_j & \cdots & \boldsymbol{x}_n\end{bmatrix}^T$ and label vector $\boldsymbol{Y}^{\boldsymbol{\eta}} = \begin{bmatrix}y_1 & y_2 & \cdots & y_i & \cdots & y_j & \cdots & y_n\end{bmatrix}^T$, and corresponding matrices $\boldsymbol{E}^{\boldsymbol{\eta}}$, $\boldsymbol{D}^{\eta}$ and $\boldsymbol{N}^{\boldsymbol{\eta}}$. Given another sample sequence $\eta' = \begin{bmatrix}1 & 2 & \cdots & j & \cdots & i & \cdots & n\end{bmatrix}$ and corresponding $\boldsymbol{E}^{\boldsymbol{\eta'}}$, $\boldsymbol{D}^{\boldsymbol{\eta'}}$ and $\boldsymbol{N}^{\boldsymbol{\eta'}}$. Then,

$$\begin{aligned}
&\boldsymbol{E}^{\eta T}\boldsymbol{N}^{\boldsymbol{\eta}}\boldsymbol{E}^{\boldsymbol{\eta}} = \boldsymbol{E}^{\boldsymbol{\eta'}^T}\boldsymbol{N}^{\boldsymbol{\eta'}}\boldsymbol{E}^{\boldsymbol{\eta'}} = \textstyle\sum_{i=1}^n \boldsymbol{E}_i^T\boldsymbol{N}_i\boldsymbol{E}_i \\
&\boldsymbol{E}^{\eta T}\boldsymbol{D}^{\boldsymbol{\eta}}\boldsymbol{N}^{\boldsymbol{\eta}}\boldsymbol{e} = \boldsymbol{E}^{\boldsymbol{\eta'}^T}\boldsymbol{D}^{\boldsymbol{\eta'}}\boldsymbol{N}^{\boldsymbol{\eta'}}\boldsymbol{e} = \textstyle\sum_{i=1}^n \boldsymbol{E}_i^T\boldsymbol{D}_i\boldsymbol{N}_i\boldsymbol{e} \\
&\boldsymbol{E}^{\eta T}\boldsymbol{E}^{\boldsymbol{\eta}} = \boldsymbol{E}^{\boldsymbol{\eta'}^T}\boldsymbol{E}^{\boldsymbol{\eta'}} = \textstyle\sum_{i=1}^n \boldsymbol{E}_i^T\boldsymbol{E}_i \\
&\boldsymbol{E}^{\eta T}\boldsymbol{e} = \boldsymbol{E}^{\boldsymbol{\eta'}^T}\boldsymbol{e} = \textstyle\sum_{i=1}^n \boldsymbol{E}_i^T\boldsymbol{e}
\end{aligned} \tag{20}$$

Since we have this sample ordering independency, we can re-order samples in wLPSVM and then split them as in (1), such that subset $\mathcal{S}_a$ and $\mathcal{S}_b$ consist only samples from positive and negative class respectively. Then we have a reformulated wLPSVM model, transforming the weighting matrix $\boldsymbol{N}$ into two simple weight coefficients σ_+ and σ_-.

Proposition 1: Given a wLPSVM model (18) over dataset $\boldsymbol{S}$, and let $\boldsymbol{M}_+ = \boldsymbol{E}_+^T\boldsymbol{E}_+$, $\boldsymbol{M}_- = \boldsymbol{E}_-^T\boldsymbol{E}_-$, $\boldsymbol{v}_+ = \boldsymbol{E}_+^T\boldsymbol{e}$ and $\boldsymbol{v}_- = \boldsymbol{E}_-^T\boldsymbol{e}$, then the wLPSVM model (18) can be reformulated as,

$$\boldsymbol{O} = (\frac{\boldsymbol{I}}{C} + \sigma_+\boldsymbol{M}_+ + \sigma_-\boldsymbol{M}_-)^{-1}(\sigma_+\boldsymbol{v}_+ - \sigma_-\boldsymbol{v}_-). \tag{21}$$

Proof:

As $\boldsymbol{\mathcal{S}}$ is a 2-class dataset decomposable into $\boldsymbol{\mathcal{S}}_+$ and $\boldsymbol{\mathcal{S}}_-$, we apply Lemma 2 and Lemma 1 to the term $\boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E}$ in (18), and have

$$\boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E} = \boldsymbol{E}_+^T\boldsymbol{N}_+\boldsymbol{E}_+ + \boldsymbol{E}_-^T\boldsymbol{N}_-\boldsymbol{E}_-. \tag{22}$$

As $\boldsymbol{N}_+ = \sigma_+\boldsymbol{I}$ and $\boldsymbol{N}_- = \sigma_-\boldsymbol{I}$, term $\boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E}$ can be further written as,

$$\boldsymbol{E}^T\boldsymbol{N}\boldsymbol{E} = \sigma_+\boldsymbol{E}_+^T\boldsymbol{E}_+ + \sigma_-\boldsymbol{E}_-^T\boldsymbol{E}_-. \tag{23}$$

Similarly, applying Lemma 2 and Lemma 1 to the term $\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{N}\boldsymbol{e}$ in (18), we have

$$\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{N}\boldsymbol{e} = \boldsymbol{E}_+^T\boldsymbol{D}_+\boldsymbol{N}_+\boldsymbol{e} + \boldsymbol{E}_-^T\boldsymbol{D}_-\boldsymbol{N}_-\boldsymbol{e}. \tag{24}$$

As $\boldsymbol{D}_+ = \boldsymbol{I}$ and $\boldsymbol{D}_- = -\boldsymbol{I}$, also $\boldsymbol{N}_+ = \sigma_+\boldsymbol{I}$ and $\boldsymbol{N}_- = \sigma_-\boldsymbol{I}$, we have

$$\boldsymbol{E}^T\boldsymbol{D}\boldsymbol{N}\boldsymbol{e} = \sigma_+\boldsymbol{E}_+^T\boldsymbol{e} - \sigma_-\boldsymbol{E}_-^T\boldsymbol{e}. \tag{25}$$

Substituting (23) and (25) into (18), and replacing $\boldsymbol{E}_+^T\boldsymbol{E}_+$, $\boldsymbol{E}_-^T\boldsymbol{E}_-$, $\boldsymbol{E}_+^T\boldsymbol{e}$ and $\boldsymbol{E}_-^T\boldsymbol{e}$ with $\boldsymbol{M}_+$, $\boldsymbol{M}_-$, $\boldsymbol{v}_+$ and $\boldsymbol{v}_-$ respectively, we obtain

$$\boldsymbol{O} = (\frac{\boldsymbol{I}}{C} + \sigma_+\boldsymbol{M}_+ + \sigma_-\boldsymbol{M}_-)^{-1}(\sigma_+\boldsymbol{v}_+ - \sigma_-\boldsymbol{v}_-). \tag{26}$$

□

Recall Lemma 1 we can see that $\boldsymbol{M}_+$, $\boldsymbol{M}_-$, $\boldsymbol{v}_+$ and $\boldsymbol{v}_-$ are can be easily obtained by merged corresponding terms from subsets of training data.

B. Incremental wLPSVM

Given an initial wLPSVM model (18) on dataset $\boldsymbol{\mathcal{S}}$. If $\boldsymbol{\mathcal{S}}_{\boldsymbol{l}}$ denotes the set of data that remains after retiring $\boldsymbol{\mathcal{S}}_{\boldsymbol{r}}$, and $\boldsymbol{\mathcal{S}'}$ denotes the updated dataset after the addition and retirement, then we have

$$\begin{aligned}
\boldsymbol{\mathcal{S}} &= \boldsymbol{\mathcal{S}}_l \cup \boldsymbol{\mathcal{S}}_r \\
\boldsymbol{\mathcal{S}}' &= \boldsymbol{\mathcal{S}}_l \cup \boldsymbol{\mathcal{S}}_a.
\end{aligned} \tag{27}$$

Then, the incremental learning of wLPSVM is to compute a updated wLPSVM model $\boldsymbol{O'}$ based on $\boldsymbol{\mathcal{S}}_a$, $\boldsymbol{\mathcal{S}}_r$, and a trained wLPSVM model (i.e., (21)) on the current dataset $\boldsymbol{\mathcal{S}}$.

By (21), a batch wLPSVM on the updated dataset $\boldsymbol{\mathcal{S}'}$ can be written as,

$$\boldsymbol{O}' = (\frac{\boldsymbol{I}}{C} + \sigma'_+\boldsymbol{M}'_+ + \sigma'_-\boldsymbol{M}'_-)^{-1}(\sigma'_+\boldsymbol{v}'_+ - \sigma'_-\boldsymbol{v}'_-). \tag{28}$$

In (28), the weighting σ'_+ and σ'_- can be simply updated by,

$$\begin{aligned}
\sigma'_+ &= l'_-/(l'_+ + l'_-) \\
\sigma'_- &= l'_+/(l'_+ + l'_-),
\end{aligned} \tag{29}$$

in which

$$\begin{aligned}
l'_- &= l_- - l_{r-} + l_{a-} \\
l'_+ &= l_+ - l_{r+} + l_{a+},
\end{aligned} \tag{30}$$

And then, we consider the updating of $\boldsymbol{M}_+$, $\boldsymbol{M}_-$, $\boldsymbol{v}_+$ and $\boldsymbol{v}_-$.

As seen in 27, $\boldsymbol{\mathcal{S}}_+$ is a union of $\boldsymbol{\mathcal{S}}_{l+}$ and $\boldsymbol{\mathcal{S}}_{r+}$, $\boldsymbol{\mathcal{S}}'_+$ is a union of $\boldsymbol{\mathcal{S}}_{l+}$ and $\boldsymbol{\mathcal{S}}_{a+}$, we can apply Lemma 2 and Lemma 1 to decompose $\boldsymbol{M}_+$ and $\boldsymbol{M}'_+$, and have

$$\boldsymbol{M}_+ = \boldsymbol{E}_+^T\boldsymbol{E}_+ = \boldsymbol{E}_{l+}^T\boldsymbol{E}_{l+} + \boldsymbol{E}_{r+}^T\boldsymbol{E}_{r+} \tag{31}$$

$$\boldsymbol{M}'_+ = \boldsymbol{E}_+^{'T}\boldsymbol{E}'_+ = \boldsymbol{E}_{l+}^T\boldsymbol{E}_{l+} + \boldsymbol{E}_{a+}^T\boldsymbol{E}_{a+} \tag{32}$$

(32) minus (31), we obtain the updating of $\boldsymbol{M}_+$ as,

$$\boldsymbol{M}'_+ = \boldsymbol{M}_+ - \boldsymbol{E}_{r+}^T\boldsymbol{E}_{r+} + \boldsymbol{E}_{a+}^T\boldsymbol{E}_{a+}. \tag{33}$$

Here, $\boldsymbol{E}_{r+}^T\boldsymbol{E}_{r+}$ and $\boldsymbol{E}_{a+}^T\boldsymbol{E}_{a+}$ can be computed from new coming data, and $\boldsymbol{M}_+$ is a part of existing model. By an analogous process, we can update $\boldsymbol{M}_-$,$\boldsymbol{v}_+$ and $\boldsymbol{v}_-$ respectively as,

$$\begin{aligned}\boldsymbol{M}'_- &= \boldsymbol{M}_- - \boldsymbol{E}_{r-}^T\boldsymbol{E}_{r-} + \boldsymbol{E}_{a-}^T\boldsymbol{E}_{a-}\\ \boldsymbol{v}'_+ &= \boldsymbol{v}_+ - \boldsymbol{E}_{r+}^T\boldsymbol{e} + \boldsymbol{E}_{a+}^T\boldsymbol{e}\\ \boldsymbol{v}'_- &= \boldsymbol{v}_- - \boldsymbol{E}_{r-}^T\boldsymbol{e} + \boldsymbol{E}_{a-}^T\boldsymbol{e}.\end{aligned} \tag{34}$$

With all components ready, we can now calculate $\boldsymbol{O}'$ using (28). The pseudo-code of the proposed incremental wLPSVM is given in Algorithm 3 below.

Algorithm 3 The Proposed Incremental wLPSVM Algorithm

Input: initial wLPSVM model: $\{\boldsymbol{O}, \boldsymbol{M}_+, \boldsymbol{M}_-, \boldsymbol{v}_+, \boldsymbol{v}_-, l_+, l_-, C\}$; $\boldsymbol{X}_r$ and $\boldsymbol{Y}_r$ to retire; $\boldsymbol{X}_a$ and $\boldsymbol{Y}_a$ to add; and regularization parameter C.
Output: updated wLPSVM model: $\{\boldsymbol{O}', \boldsymbol{M}'_+, \boldsymbol{M}'_-, \boldsymbol{v}'_+, \boldsymbol{v}'_-, l'_+, l'_-, C\}$.

```
if input model is empty then
    Generate E_{a+} and E_{a-} from X_a and Y_a
    Compute M_+ = E_{a+}^T E_{a+}, M_- = E_{a-}^T E_{a-}, v_+ = E_{a+}^T e and v_- = E_{a-}^T e;
    Compute σ_+ and σ_- according to (17);
    Compute O according to (21);
else
    Generate E_{a+}, E_{a-}, E_{r+} and E_{r-} from X_a, Y_a, X_r and Y_r;
    Compute M'_+, M'_-, v_+, v_- according to (33) and (34)
    Compute σ'_+ and σ'_-, according to (29) and (30);
    Compute O' according to (28).
end if
```

C. Distributed Incremental wLPSVM

Let's first look at the distributed wLPSVM learning problem alone. Assume we have t distributed nodes $n_1, \ldots, n_t$ and each of them have their local training data $\boldsymbol{\mathcal{S}}_1, \ldots, \boldsymbol{\mathcal{S}}_t$ respectively. The distributed learning of global $\boldsymbol{O}$ on $\boldsymbol{\mathcal{S}} = \boldsymbol{\mathcal{S}}_1 \cup \ldots \cup \boldsymbol{\mathcal{S}}_t$, is actually a problem of merging all local models $m_i = \{\boldsymbol{M}_{i+}, \boldsymbol{M}_{i-}, \boldsymbol{v}_{i+}, \boldsymbol{v}_{i-}, l_{i+}, l_{i-}\}$. From Lemma 1, we can easily know the merging rules as

$$\begin{aligned}\boldsymbol{M}_+ &= \textstyle\sum_{i=1}^t \boldsymbol{M}_{i+}\\ \boldsymbol{M}_- &= \textstyle\sum_{i=1}^t \boldsymbol{M}_{i-}\\ \boldsymbol{v}_+ &= \textstyle\sum_{i=1}^t \boldsymbol{v}_{i+}\\ \boldsymbol{v}_- &= \textstyle\sum_{i=1}^t \boldsymbol{v}_{i-}.\end{aligned} \tag{35}$$

For the number of samples, we have $l_+ = \sum_{i=1}^t l_{i+}$ and $l_- = \sum_{i=1}^t l_{i-}$. Using (17) we can calculate σ_+ and σ_-. Eventually, we obtain global $\boldsymbol{O}$ computed as (21).

When we consider distributed and incremental learning all together, of course we can obtain $\boldsymbol{E}_r^T\boldsymbol{E}_r$, $\boldsymbol{E}_a^T\boldsymbol{E}_a$, $\boldsymbol{E}_r^T\boldsymbol{e}$ and $\boldsymbol{E}_a^T\boldsymbol{e}$ by merging the corresponding local terms as in (35), and then compute $\boldsymbol{O}'$ via (33), (34) and (28). But in this sense, each slave node will have to transmit local terms about both sample adding and removal, and these terms will do a subtraction as in (33) and (34) after merging. To reduce communication cost, we conduct this subtraction at slave nodes.

For the i-th slave node n_i, given $\boldsymbol{X}_{ir}$ and $\boldsymbol{Y}_{ir}$ to be retired, $\boldsymbol{X}_{ia}$ and $\boldsymbol{Y}_{ia}$ to be added, it generates the following information from local change. The change of instance number for both class are computed as

$$\begin{aligned}l_{ic+} &= l_{ia+} - l_{ir+}\\ l_{ic-} &= l_{ia-} - l_{ir-}.\end{aligned} \tag{36}$$

The changes to terms $\boldsymbol{M}_+$, $\boldsymbol{M}_-$, $\boldsymbol{v}_+$ and $\boldsymbol{v}_-$ respectively as

$$\begin{aligned}\boldsymbol{M}_{ic+} &= -\boldsymbol{E}_{ir+}^T\boldsymbol{E}_{ir+} + \boldsymbol{E}_{ia+}^T\boldsymbol{E}_{ia+}\\ \boldsymbol{M}_{ic-} &= -\boldsymbol{E}_{ir-}^T\boldsymbol{E}_{ir-} + \boldsymbol{E}_{ia-}^T\boldsymbol{E}_{ia-}\\ \boldsymbol{v}_{ic+} &= -\boldsymbol{E}_{ir+}^T\boldsymbol{e} + \boldsymbol{E}_{ia+}^T\boldsymbol{e}\\ \boldsymbol{v}_{ic-} &= -\boldsymbol{E}_{ir-}^T\boldsymbol{e} + \boldsymbol{E}_{ia-}^T\boldsymbol{e}.\end{aligned} \tag{37}$$

Then n_i submits its local change m_i towards the master node, m_i is in form of $\{l_{ic+}, l_{ic-}, \boldsymbol{M}_{ic+}, \boldsymbol{M}_{ic-}, \boldsymbol{v}_{ic+}, \boldsymbol{v}_{ic-}\}$.

For the master node, who has a current wLPSVM model $\{\boldsymbol{O}, \boldsymbol{M}_+, \boldsymbol{M}_-, \boldsymbol{v}_+, \boldsymbol{v}_-, l_+, l_-, C\}$ maintained, when it receives all updates m_i from slave nodes $n_1, \ldots, n_t$, it updates $\boldsymbol{M}$ and $\boldsymbol{v}$ terms as

$$\begin{aligned}\boldsymbol{M}'_+ &= \boldsymbol{M}_+ + \textstyle\sum_{i=1}^t \boldsymbol{M}_{ic+}\\ \boldsymbol{M}'_- &= \boldsymbol{M}_- + \textstyle\sum_{i=1}^t \boldsymbol{M}_{ic-}\\ \boldsymbol{v}'_+ &= \boldsymbol{v}_+ + \textstyle\sum_{i=1}^t \boldsymbol{v}_{ic+}\\ \boldsymbol{v}'_- &= \boldsymbol{v}_- + \textstyle\sum_{i=1}^t \boldsymbol{v}_{ic-}.\end{aligned} \tag{38}$$

The number of instance is updated as

$$\begin{aligned}l'_+ &= l_+ + \textstyle\sum_{i=1}^t l_{ic+}\\ l'_- &= l_- + \textstyle\sum_{i=1}^t l_{ic-}.\end{aligned} \tag{39}$$

Since we know

$$\begin{aligned}&\textstyle\sum_{i=1}^t \boldsymbol{M}_{ic+} = -\sum_{i=1}^t \boldsymbol{E}_{ir+}^T\boldsymbol{E}_{ir+} + \sum_{i=1}^t \boldsymbol{E}_{ia+}^T\boldsymbol{E}_{ia+}\\ &= -\boldsymbol{E}_{r+}^T\boldsymbol{E}_{r+} + \boldsymbol{E}_{a+}^T\boldsymbol{E}_{a+}\\ &\textstyle\sum_{i=1}^t \boldsymbol{v}_{ic+} = -\sum_{i=1}^t \boldsymbol{E}_{ir+}^T\boldsymbol{e} + \sum_{i=1}^t \boldsymbol{E}_{ia+}^T\boldsymbol{e}\\ &= -\boldsymbol{E}_{r+}^T\boldsymbol{e} + \boldsymbol{E}_{a+}^T\boldsymbol{e},\end{aligned} \tag{40}$$

and so for the negative class side, updating (38) is actually equivalent to (33) and (34). Using (29) to have class weight computed, the master node can eventually compute the updated classifier $\boldsymbol{O}'$ via (28).

The slave and master node pseudo-code for the proposed distributed incremental wLPSVM are given respectively as below.

Algorithm 4 The Proposed Distributed Incremental wLPSVM Algorithm at Slave Node

Input: $\boldsymbol{X}_{ir}$ and $\boldsymbol{Y}_{ir}$ to be retired, $\boldsymbol{X}_{ia}$ and $\boldsymbol{Y}_{ia}$ to be added.
Output: message $m_i = \{l_{ic+}, l_{ic-}, \boldsymbol{M}_{ic+}, \boldsymbol{M}_{ic-}, \boldsymbol{v}_{ic+}, \boldsymbol{v}_{ic-}\}$.

```
Generate E_{ia+}, E_{ia-}, E_{ir+}, E_{ir-}, l_{ia+}, l_{ia-}, l_{ir+} and l_{ir-} from X_{ia}, Y_{ia}, X_{ir} and Y_{ir};
Compute l_{ic+} and l_{ic-} as (36);
Compute M_{ic+}, M_{ic-}, v_{ic+} and v_{ic-} as (37);
```

Algorithm 5 The Proposed Distributed Incremental wLPSVM Algorithm at Master Node

Input: initial wLPSVM model $\{\boldsymbol{O}, \boldsymbol{M}_+, \boldsymbol{M}_-, \boldsymbol{v}_+, \boldsymbol{v}_-, l_+, l_-, C\}$, messages m_i from all slave nodes.
Output: updated wLPSVM model $\{\boldsymbol{O}', \boldsymbol{M}'_+, \boldsymbol{M}'_-, \boldsymbol{v}'_+, \boldsymbol{v}'_-, l'_+, l'_-, C\}$.

```
Update M_+, M_-, v_+, v_- as (38);
Update l_+, l_- as (39);
Compute σ'_+ and σ'_- using (29);
Compute O' as (28).
```

IV. Experiments

Since the good classification capability and the robustness to class-imbalance of wLPSVM has been shown in [13], [14], and proposed distributed incremental wLPSVM theoretically gives the same learning result, here our experiments here focus on result equivalence demonstration and efficiency evaluation.

We implement our algorithm using Matlab Parallel Environment. All experiments are conducted on a laptop with 4-core 2.4GHz CPU and 8 GB memory. The data used are all downloaded from the public UCI repository [16].

A. Learning Result Equivalence to Batch wLPSVM

Although we have shown the theoretical equivalence between proposed algorithms and the original batch algorithm, here we further verify this equivalence using numerical experiment. We compare the learning result (classifier) from incremental, distributed and distributed incremental learning against that from batch learning. Since the classifier learned here is a vector, we use 2-norm of the vector difference as measurement. Here we set the incremental learning as two-step (i.e., learn 50% of the data and then the rest 50%), and we conduct the distributed learning using two nodes.

TABLE I
Learning result difference against batch wLPSVM.

Dataset	# Instance	# Feature	Inc	Dist	DistInc
Heart	303	75	0	0	0
BreastWisconsin	569	32	0	0	0
AnonymousWeb	37,711	294	0	0	0
CoverType	581,012	54	0	0	0

Here we measure the learning result differences on four datasets with various sizes and dimensions. For datasets that are originally multi-class problem, we transform them into binary one-against-rest data. The characteristic of each dataset and corresponding learning result differences are reported in Table I.

Not surprisingly, the differences for all four cases are all zero, i.e., the classifiers learning from proposed incremental, distributed and distributed incremental wLPSVM learning algorithms are all exactly equals to that from batch learning. The theoretical equivalence is verified by numerical experiment.

B. Parallelization Effectiveness

To evaluate the parallelization effectiveness, i.e., how well the training is accelerated using more learning nodes, we first fix the training data size and measure the training time variant with respect to the number of cores used. Here, we also interested in how data size affects such effectiveness. Thus we conduct above comparison on different proportion of data.

The data we used here is binary CoverType, which has 581k instances. We let the number of cores varies from 1 to 4, and the data proportion varies from 12.5% to 100%. We conduct training at each setup for 10 rounds, and the average training time are shown in Table II

First look at the effect of speedup, let the training speed using 1-core to be the basis, we calculate how much times faster when using more than one cores to process the same amount of data. The speedup curves at different data size are plotted in Figure 2.

TABLE II
Average training time at different data size and number of cores.

Proportion	1 core	2 core	3 core	4 core
100%	18.33s	10.36s	8.52s	7.90s
75%	14.02s	8.00s	6.57s	6.07s
50%	9.31s	5.48s	4.40s	4.05s
25%	4.92s	2.94s	2.35s	2.15s

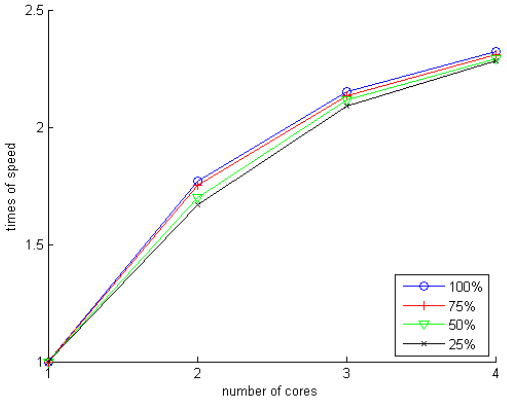

Fig. 2. Speedup against number of cores

As we can see here, with the increase of number of cores, the training speed increases. The parallelization of proposed algorithm truly accelerates training by investing more computational nodes. However, this increase is not linear (i.e., n-times faster when use n-cores) as we expected, and the marginal benefit for adding one more core decreases with the number of cores already used. This can be explained by two facts: 1) in theory, parallelized $\boldsymbol{M}$ and $\boldsymbol{v}$ computing can achieve linear speedup, but the computation at master node has fixed cost, thus proposed algorithm can only achieve near linear speedup at large dataset; and 2) Matlab parallel environment need extra time to coordinate multiple nodes, and this extra cost increases with the number of node in action, thus the more node used the more speedup effect is neutralize.

Regrading the data size, we can see here the speedup performs better with the increase of data size. The reason is that the time to process small dataset is not dominant compared with the time consumed by nodes communication and task arrangement. When the size of dataset increase,the time of intensive computation becomes dominant, so the speed up increases. Therefore, proposed algorithm can process large datasets efficiently.

Next we look at the aspect of sizeup, which measures how much times longer the given system takes to learn when the dataset is m-times larger. The sizeup curves at different number of cores are given in Figure 3.

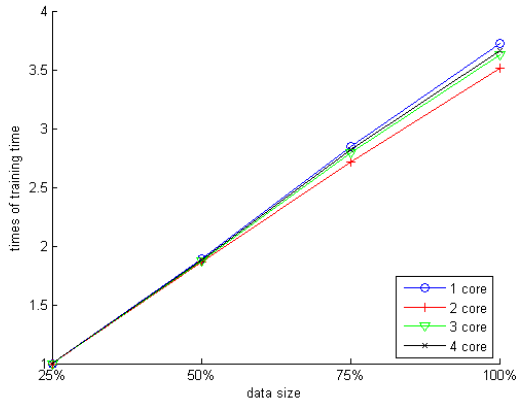

Fig. 3. Training time against data size

As we can see here, the training time grows linearly with the data size, regardless the number of cores used, and the slope is less than one. This means when the data is m-times larger, proposed algorithm only takes less than m-times time to learn, which also indicates the effectiveness of proposed algorithm on learning large datasets.

C. Incremental Effectiveness

To evaluate the effectiveness of proposed incremental learning, we progressively expand the training set by adding more new instances and measure the training time cost by batch, parallel, incremental and parallel incremental wLPSVM respectively. For all parallel algorithms, we set the number of cores used as four, and the result is shown in Table III below.

TABLE III
TRAINING TIME FOR DIFFERENT ALGORITHMS

# new Instances	batch	Incremental	Parallel	Parallel Inc.
30 k	4.36s	4.36s	1.97s	1.97s
15 k	6.54s	2.20s	2.95s	1.01s
7.5 k	7.62s	1.09s	3.44s	0.49s
3.75 k	8.13s	0.56s	3.70s	0.25s

As we can see here, the training time for proposed incremental and parallel incremental wLPSVM decreases as the new data becomes smaller, and the training time for algorithms without incremental capability grows continuously. This is because proposed incremental algorithms only learn from the newly introduced data thus the training time is determined by the size of new data. On the other hand, non-incremental variants have to learn from all data thus the training time is determined by the size of accumulated data. When we compare the parallel algorithms with the non-parallel ones, we can see that the training time is clearly reduced by parallelization. We can easily conclude that the combination of parallel and incremental wLPSVM learning is more suitable for solving big data learning problems.

V. CONCLUSION AND FUTURE WORK

In this work, we update wLPSVM into an distributed incremental algorithm, which capable of simultaneously learning from multiple streaming data sources that are geographically distributed. The incremental and distributed learning of wLPSVM in solve at the same time as a merging problem. We derive a new wLPSVM expression, in which knowledge from samples are presented as a set of class-wised core matrices, and merging knowledge from two subsets of data can be simply accomplished by matrix computation. With the help of such merging, we are able to conduct incremental and distributed learning at the same time via merging knowledge from multiple incremental stages and multiple data sources.

In the experiments, the effectiveness of proposed algorithm on learning big datasets is roughly evaluated based on our Matlab prototype. Due to the limitations of Matlab and our experimental environment, we can not fully showcase proposed algorithm, such as the capability of learning from multiple data sources. In the future, we will implement our algorithm in MapReduce. That will also makes it easier to adopt our algorithm in real world applications.

REFERENCES

[1] M. J. Zaki, C.-T. Ho, and R. Agrawal, "Parallel classification for data mining on shared-memory multiprocessors," in *Data Engineering, 1999. Proceedings., 15th International Conference on*. IEEE, 1999, pp. 198–205.

[2] T. Rohlfing and C. R. Maurer, "Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees," *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 1, pp. 16–25, 2003.

[3] K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, H. Cui, and E. Y. Chang, "Parallelizing support vector machines on distributed computers," in *Advances in Neural Information Processing Systems*, 2008, pp. 257–264.

[4] D. Pechyony, L. Shen, and R. Jones, "Solving large scale linear svm with distributed block minimization," in *NIPS workshop on Big Learning*, 2011.

[5] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1663–1707, 2010.

[6] Q. He, C. Du, Q. Wang, F. Zhuang, and Z. Shi, "A parallel incremental extreme svm classifier," *Neurocomputing*, vol. 74, no. 16, pp. 2532–2540, 2011.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[8] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Berlin, Germany: Springer-Verlag, 1995.

[9] T. Yamasaki and K. Ikeda, "Incremental svms and their geometrical analyses," in *Neural Networks and Brain, 2005. ICNN B '05. International Conference on*, vol. 3, oct. 2005, pp. 1734 –1738.

[10] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," *Neural Networks, IEEE Transactions on*, vol. 21, no. 7, pp. 1048 –1059, july 2010.

[11] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *NIPS'00*, 2000, pp. 409–415.

[12] G. Fung and O. L. Mangasarian, "Proximal support vector machine classifiers," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, San Francisco, California, 2001, pp. 77–86.

[13] G. M. Fung and O. L. Mangasarian, "Multicategory proximal support vector machine classifiers," *Mach. Learn.*, vol. 59, pp. 77–97, 2005.

[14] D. Zhuang, B. Zhang, Q. Yang, J. Yan, Z. Chen, and Y. Chen, "Efficient text classification by weighted proximal svm," in *Data Mining, Fifth IEEE International Conference on*, nov. 2005, p. 8 pp.

[15] X. Tao and H. Ji, "A modified psvm and its application to unbalanced data classification," in *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 1, aug. 2007, pp. 488 –490.

[16] M. Lichman, “UCI machine learning repository,” 2013. [Online]. Available: http://archive.ics.uci.edu/ml