Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

An efficient parallel and distributed solution to nonconvex penalized linear SVMs*

Lei GUAN†1, Tao SUN†1, Lin-bo QIAO1, Zhi-hui YANG2,3, Dong-sheng LI†‡1, Ke-shi GE1, Xi-cheng LU1

[1]*College of Computer, National University of Defense Technology, Changsha 410073, China*

[2]*School of Computer Science, Fudan University, Shanghai 201203, China*

[3]*Shanghai Key Laboratory of Data Science, Shanghai 201203, China*

†E-mail: guanleics@gmail.com; nudtsuntao@163.com; dsli@nudt.edu.cn

Received Sept. 14, 2018; Revision accepted Dec. 8, 2018; Crosschecked Aug. 12, 2019; Published online Sept. 5, 2019

Abstract: Support vector machines (SVMs) have been recognized as a powerful tool to perform linear classification. When combined with the sparsity-inducing nonconvex penalty, SVMs can perform classification and variable selection simultaneously. However, the nonconvex penalized SVMs in general cannot be solved globally and efficiently due to their nondifferentiability, nonconvexity, and nonsmoothness. Existing solutions to the nonconvex penalized SVMs typically solve this problem in a serial fashion, which are unable to fully use the parallel computing power of modern multi-core machines. On the other hand, the fact that many real-world data are stored in a distributed manner urgently calls for a parallel and distributed solution to the nonconvex penalized SVMs. To circumvent this challenge, we propose an efficient alternating direction method of multipliers (ADMM) based algorithm that solves the nonconvex penalized SVMs in a parallel and distributed way. We design many useful techniques to decrease the computation and synchronization cost of the proposed parallel algorithm. The time complexity analysis demonstrates the low time complexity of the proposed parallel algorithm. Moreover, the convergence of the parallel algorithm is guaranteed. Experimental evaluations on four LIBSVM benchmark datasets demonstrate the efficiency of the proposed parallel algorithm.

Key words: Linear classification; Support vector machine (SVM); Nonconvex penalty; Alternating direction method of multipliers (ADMM); Parallel algorithm

https://doi.org/10.1631/FITEE.1800566 **CLC number:** TP391

1 Introduction

In the field of machine learning, the standard support vector machine (SVM) is a popular tool for performing linear classification. It is well known that some sparsity-inducing penalties (regularizers) can perform automatic variable selection when a loss function is added. Therefore, there is a natural tendency to combine the standard SVM and sparsity-inducing penalties to realize simultaneous linear classification and variable selection. Typically, these sparsity-inducing penalties consist of convex and nonconvex penalties. Since the nonconvex penalties outperform the convex ones with better statistics properties (Fan and Li, 2001), nonconvex penalized SVMs are more appealing than convex penalized SVMs with respect to robustness.

Formally, given a training set $S = (\boldsymbol{x}_i, y_i)_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, the nonconvex penalized linear SVMs minimize the following problem:

$$\min_{\{\boldsymbol{w},b\}} \ \frac{1}{n}\sum_{i=1}^{n}\left[1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b)\right]_+ + P_\lambda(\boldsymbol{w}), \quad (1)$$

‡ Corresponding author

* Project supported by the Major State Research Development Program, China (No. 2016YFB0201305)

ORCID: Dong-sheng LI, http://orcid.org/0000-0001-9743-2034

where $P_\lambda(\boldsymbol{w})$ is the nonconvex regularizer with the tuning parameter λ. Commonly used nonconvex penalties include the smoothly clipped absolute deviation (SCAD) penalty (Fan and Li, 2001), log penalty (Mazumder et al., 2011), minimax concave penalty (MCP) (Zhang CH, 2010), log-sum penalty (LSP) (Candès et al., 2008), and capped-ℓ_1 penalty (Zhang T, 2010). These nonconvex penalties are summarized in Table 1.

Table 1 Summarization of nonconvex regularizers

Name	$p_\lambda(w_j)$
LSP	$\lambda \log(1+\lvert w_j\rvert/\theta)$
SCAD	$\begin{cases} \lambda\lvert w_j\rvert, & \lvert w_j\rvert \le \lambda, \\ \dfrac{-w_j^2+2\theta\lambda\lvert w_j\rvert-\lambda^2}{2(\theta-1)}, & \lambda < \lvert w_j\rvert \le \theta\lambda, \\ \dfrac{(\theta+1)\lambda^2}{2}, & \lvert w_j\rvert > \theta\lambda. \end{cases}$
MCP	$\begin{cases} \lambda\lvert w_j\rvert - w_j^2/(2\theta), & \lvert w_j\rvert \le \theta\lambda, \\ \theta\lambda^2/2, & \lvert w_j\rvert > \theta\lambda. \end{cases}$
Capped-ℓ_1	$\lambda \min(\lvert w_j\rvert, \theta)$

$P_\lambda(\boldsymbol{w}) = \sum_{j=1}^{d} p_\lambda(w_j)$. Here, $\theta > 2$ for the SCAD regularizer and $\theta > 0$ for the other regularizers

Problem (1) is hard to solve due to the nondifferentiability of the hinge loss function as well as the nonconvexity and nonsmoothness of the nonconvex regularizer. To solve this problem efficiently, Guan et al. (2018) proposed an efficient algorithm by incorporating the framework of alternating direction method of multipliers (ADMM) (Boyd et al., 2011). The ADMM-based algorithm actually consists of two stages: pre-computation stage and iteration stage. The pre-computation stage takes a rather large proportion of the total running time due to the Cholesky factorization of a square matrix. The cost of performing Cholesky factorization increases quickly with the augment of data size. Specifically, when processing large-scale datasets, the pre-computation stage becomes the main burden in the algorithm. Moreover, the algorithm proposed by Guan et al. (2018) is single-threaded and unable to handle distributed-stored data or fully use the parallel computing power of modern computer systems.

To address the challenge, in this study, we propose an efficient parallel algorithm to the nonconvex penalized SVMs. The parallel algorithm is designed based on reasonably reformulating the optimization problem and incorporating the consensus ADMM procedure. When running with one processor, the parallel algorithm is reduced to the serial algorithm proposed by Guan et al. (2018). Experimental evaluations on four LIBSVM datasets demonstrate the fast execution speed and strong scalability of the proposed parallel algorithm. Compared with the serial algorithm, the parallel algorithm can attain a factor of over 15× speedup when evaluated on a multi-core computing node.

The contributions can be summarized as follows:

1. Reasonable derivations are performed to transform the nondifferentiable, nonconvex, and nonsmooth problem to an equivalent optimization objective that can be solved by applying the consensus ADMM procedure.

2. Compared with the serial algorithm proposed by Guan et al. (2018), our parallel algorithm handles the Cholesky factorization of a large matrix by splitting this large matrix into K ($K > 1$) small matrices and parallelly performing Cholesky factorization of each small matrix. Moreover, several techniques are adopted to further reduce the synchronization cost. The resulting parallel algorithm is quite efficient with low computation cost and only one synchronization operation per iteration.

3. Detailed convergence analysis demonstrates that the convergence of the proposed parallel algorithm is guaranteed.

4. Experimental evaluations show that our parallel algorithm can fully use the parallel computing power of modern machines and is well-suited for solving nonconvex penalized SVMs in the distributed environments.

2 Related work

Nonconvex optimization problems have been heavily studied in the literature. There are many studies focusing on nonconvex optimization problems (Gong et al., 2013; Laporte et al., 2014; Wang et al., 2015; Allen-Zhu and Hazan, 2016; Zhang SB et al., 2016). Nevertheless, many of them are unable to solve the nonconvex optimization problem. This is because many of them require the loss function be differentiable, which violates the nondifferentiability feature of the hinge loss function. Moreover, other research on nonconvex optimization such as Reddi et al. (2016) is inapplicable since the optimization problems on which they focused do not cover the optimization problem studied in this work. These methods share a common feature that the loss

function is possibly in a nonconvex form while the regularization term is convex but possibly nonsmooth.

Existing approaches to the nonconvex penalized hinge loss functions include those proposed by Zhang HH et al. (2006), Zhang T (2010), Liu et al. (2016), Zhang X et al. (2016), and Guan et al. (2018). These methods share a common feature that they are running serially and cannot fully use the parallel computing power of modern computer systems (Zhang et al., 2017). Besides that, many of them cover only a few of nonconvex regularizers.

There is much research concerned with the distributed nonconvex optimization problems (Razaviyayn et al., 2014; Daneshmand et al., 2015; di Lorenzo and Scutari, 2015; Facchinei et al., 2015; Hong et al., 2016; Sun et al., 2016). The optimization objectives of these existing methods share the same feature as mentioned before; i.e., the loss function is possibly in a nonconvex form while the regularization term is convex but possibly nonsmooth. Clearly, this optimization problem is far from the definition of nonconvex penalized SVMs in this study. Recently, Scutari et al. (2017) studied the parallel and distributed methods for nonconvex optimization problems. However, they focused on the minimization of a nonconvex smooth function subject to nonconvex smooth constraints, which does not accord with the nonconvex penalized hinge loss functions. To the best of our knowledge, this is the first work that investigates solving the nonconvex penalized hinge loss functions in a parallel and distributed fashion.

3 Parallel ADMM algorithm for nonconvex penalized SVMs

In this section, we derive the parallel algorithm based on the consensus ADMM. Then we present the implementation in detail followed by a discussion of the time complexity.

General notational conventions are described as follows: Uppercase (lowercase) bold letters are used for matrices (column vectors); "$\succeq$" denotes elementwise $\geqq$; "$< \cdot\ ,\ \cdot >$" represents the standard inner product in the Euclidean space; "$|\cdot|$" stands for the absolute value of a given real value. $\mathbf{1}_n$ ($\mathbf{0}_n$) is a column vector of all ones (zeros) of size n. $\boldsymbol{I}_m$ stands for an $m \times m$ identity matrix.

3.1 Parallel algorithm derivation

We first reformulate problem (1) into an equivalent form to handle the nondifferentiability. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^{\mathrm{T}} \in \mathbb{R}^{n\times d}$ be the feature matrix and $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^{\mathrm{T}} \in \mathbb{R}^n$ the target vector. Then problem (1) can be written more concisely in the following matrix form:

$$\begin{aligned} \min_{\{\boldsymbol{w},b,\boldsymbol{\xi}\}} \quad & \frac{1}{n}\mathbf{1}_n^{\mathrm{T}}\boldsymbol{\xi} + P_\lambda(\boldsymbol{w}) \\ \text{s.t.} \quad & \boldsymbol{Y}(\boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_n) \succeq \mathbf{1}_n - \boldsymbol{\xi}, \\ & \boldsymbol{\xi} \succeq \mathbf{0}_n, \end{aligned} \tag{2}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_n)^{\mathrm{T}}$ and $\boldsymbol{Y}$ is an $n\times n$ diagonal matrix with y_i on the i^{th} diagonal element, i.e., $\boldsymbol{Y} = \mathrm{diag}(\boldsymbol{y}_1,\ \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)$.

Note that by using variable splitting and introducing another slack variable $\boldsymbol{s}$, problem (2) can be converted to another equivalent constrained problem:

$$\begin{aligned} \min_{\{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{s},\boldsymbol{z}\}} \quad & \frac{1}{n}\mathbf{1}_n^{\mathrm{T}}\boldsymbol{\xi} + P_\lambda(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{w} = \boldsymbol{z}, \\ & \boldsymbol{Y}(\boldsymbol{X}\boldsymbol{w} + b\mathbf{1}_n) + \xi = \boldsymbol{s} + \mathbf{1}_n, \\ & \boldsymbol{\xi} \succeq \mathbf{0}_n, \boldsymbol{s} \succeq \mathbf{0}_n, \end{aligned} \tag{3}$$

where $\boldsymbol{z} = (z_1, z_2, \ldots, z_d)^{\mathrm{T}}$ and $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)^{\mathrm{T}}$.

In considering solving the optimization problem (3) with K processors, we first partition the feature matrix $\boldsymbol{X}$ and target vector $\boldsymbol{y}$ by rows:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \vdots \\ \boldsymbol{X}_K \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_K \end{bmatrix},$$

with $\boldsymbol{X}_i \in \mathbb{R}^{n_i\times d}$, $\boldsymbol{y}_i \in \mathbb{R}^{n_i}$, and $n = \sum_{i=1}^K n_i$. Let $\boldsymbol{Y}_i = \mathrm{diag}(y_{i1},\ y_{i2},\ \ldots,\ y_{in_i})$. Then $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$ represent the i^{th} block of data that will be handled by the i^{th} processor.

After that, we consider the case with a single global variable, with the objective and constraint terms split into K parts:

$$\begin{aligned} \min_{\{\boldsymbol{w}_i,b_i,\boldsymbol{z},\boldsymbol{\xi}_i,\boldsymbol{s}_i\}} \quad & \frac{1}{n}\textstyle\sum_{i=1}^K \mathbf{1}_{n_i}^{\mathrm{T}}\boldsymbol{\xi}_i + P_\lambda(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{w}_i = \boldsymbol{z}, \\ & \boldsymbol{Y}_i(\boldsymbol{X}_i\boldsymbol{w}_i + b_i\mathbf{1}_{n_i}) + \boldsymbol{\xi}_i = \boldsymbol{s}_i + \mathbf{1}_{n_i}, \\ & \boldsymbol{\xi}_i \succeq \mathbf{0}_{n_i}, \boldsymbol{s}_i \succeq \mathbf{0}_{n_i}, \quad i = 1, 2, \ldots, K, \end{aligned} \tag{4}$$

where $\boldsymbol{w}_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}$, $\boldsymbol{\xi}_i \in \mathbb{R}^{n_i}$, and $\boldsymbol{s}_i \in \mathbb{R}^{n_i}$ are local variables. Here, $\boldsymbol{z} \in \mathbb{R}^d$ is the common global variable shared by all the processors.

The surrogate Lagrangian function of problem (4) is

$$\begin{aligned}&\mathcal{L}_0(\boldsymbol{w}_i, b_i, \boldsymbol{z}, \boldsymbol{\xi}_i, \boldsymbol{s}_i, \boldsymbol{\gamma}_i, \boldsymbol{\tau}_i)\\ =\ & P_\lambda(\boldsymbol{z})+\Big[\frac{1}{n}\mathbf{1}_{n_i}^{\mathrm{T}}\boldsymbol{\xi}_i+<\boldsymbol{\gamma}_i,(\boldsymbol{w}_i-\boldsymbol{z})>\\ &+<\boldsymbol{\tau}_i,\boldsymbol{Y}_i(\boldsymbol{X}_i\boldsymbol{w}_i+b_i\mathbf{1}_{n_i})+\boldsymbol{\xi}_i-\boldsymbol{s}_i-\mathbf{1}_{n_i}>\Big],\end{aligned} \tag{5}$$

where $\boldsymbol{\gamma}_i \in \mathbb{R}^d$ and $\boldsymbol{\tau}_i \in \mathbb{R}^{n_i}$ are the dual variables corresponding to the first and second linear constraints of problem (4), respectively. Note that we call $\mathcal{L}_0(\boldsymbol{w}_i, b_i, \boldsymbol{z}, \boldsymbol{\xi}_i, \boldsymbol{s}_i, \boldsymbol{\gamma}_i, \boldsymbol{\tau}_i)$ the "surrogate Lagrangian function" since it does not involve the set of constraints $\{\boldsymbol{\xi}_i \succeq \mathbf{0}_{n_i}, \boldsymbol{s}_i \succeq \mathbf{0}_{n_i}\}$. These two simple bound constraints can be easily calculated by basic algebra computations and projections to the one-dimensional (1D) nonnegative set $[0, +\infty)$.

Let $\boldsymbol{H}_i = \boldsymbol{Y}_i\boldsymbol{X}_i$ and note that $\boldsymbol{y}_i = \boldsymbol{Y}_i\mathbf{1}_{n_i}$. Thus, we can write the scaled-form surrogate augmented Lagrangian function as

$$\begin{aligned}&\mathcal{L}(\boldsymbol{w}_i, b_i, \boldsymbol{z}, \boldsymbol{\xi}_i, \boldsymbol{s}_i, \boldsymbol{u}_i, \boldsymbol{v}_i)\\ =\ & \mathcal{L}_0(\boldsymbol{w}_i, b_i, \boldsymbol{z}, \boldsymbol{\xi}_i, \boldsymbol{s}_i, \boldsymbol{\gamma}_i, \boldsymbol{\tau}_i)+\frac{\rho_1}{2}||\boldsymbol{w}_i-\boldsymbol{z}||_2^2\\ &+\frac{\rho_2}{2}||\boldsymbol{H}_i\boldsymbol{w}_i+b_i\boldsymbol{y}_i+\boldsymbol{\xi}_i-\boldsymbol{s}_i-\mathbf{1}_{n_i}||_2^2\\ =\ & P_\lambda(\boldsymbol{z})+\Big[\frac{1}{n}\mathbf{1}_{n_i}^{\mathrm{T}}\boldsymbol{\xi}_i+\frac{\rho_1}{2}||\boldsymbol{w}_i-\boldsymbol{z}+\boldsymbol{u}_i||_2^2\\ &+\frac{\rho_2}{2}||\boldsymbol{H}_i\boldsymbol{w}_i+b_i\boldsymbol{y}_i+\boldsymbol{\xi}_i-\boldsymbol{s}_i-\mathbf{1}_{n_i}+\boldsymbol{v}_i||_2^2\Big]\\ &+\text{constant},\end{aligned} \tag{6}$$

where $\boldsymbol{u}_i = \boldsymbol{\gamma}_i/\rho_1$ and $\boldsymbol{v}_i = \boldsymbol{\tau}_i/\rho_2$ denote the scaled dual variables. The constants ρ_1 and ρ_2 are the penalty parameters satisfying $\rho_1 > 0$ and $\rho_2 > 0$.

Then the scaled-form ADMM procedure starts from an initial point $\left(\boldsymbol{w}_i^{(0)}, b_i^{(0)}, \boldsymbol{z}^{(0)}, \boldsymbol{\xi}_i^{(0)}, \boldsymbol{s}_i^{(0)}, \boldsymbol{u}_i^{(0)}, \boldsymbol{v}_i^{(0)}\right)$; for each iteration count $k = 0, 1, \ldots$, the resulting ADMM procedure can be expressed as follows:

$$\boldsymbol{w}_i^{(k+1)} = \arg\min_{\boldsymbol{w}_i}\ \mathcal{L}\left(\boldsymbol{w}_i, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right), \tag{7}$$

$$b_i^{(k+1)} = \arg\min_{b_i}\ \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right), \tag{8}$$

$$\boldsymbol{z}^{(k+1)} = \arg\min_{\boldsymbol{z}}\ \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right), \tag{9}$$

$$\boldsymbol{\xi}_i^{(k+1)} = \arg\min_{\boldsymbol{\xi}_i \succeq \mathbf{0}_{n_i}}\ \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right), \tag{10}$$

$$\boldsymbol{s}_i^{(k+1)} = \arg\min_{\boldsymbol{s}_i \succeq \mathbf{0}_{n_i}}\ \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right), \tag{11}$$

$$\boldsymbol{u}_i^{(k+1)} = \boldsymbol{u}_i^{(k)} + \left(\boldsymbol{w}_i^{(k+1)} - \boldsymbol{z}^{(k+1)}\right), \tag{12}$$

$$\boldsymbol{v}_i^{(k+1)} = \boldsymbol{v}_i^{(k)} + \left(\boldsymbol{\xi}_i^{(k+1)} - \boldsymbol{s}_i^{(k+1)} + \boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)} + b_i\boldsymbol{y}_i - \mathbf{1}_{n_i}\right). \tag{13}$$

In terms of optimizing problems (7) and (8), we can obtain their closed-form solutions via $\partial\mathcal{L}/\partial\boldsymbol{w}_i = \mathbf{0}_d$ and $\partial\mathcal{L}/\partial b_i = 0$, respectively; that is,

$$\boldsymbol{w}_i^{(k+1)} = (\rho_1\boldsymbol{I}_d+\rho_2\boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i)^{-1}\Big[\rho_1(\boldsymbol{z}^{(k)}-\boldsymbol{u}_i^{(k)}) +\rho_2\boldsymbol{H}_i^{\mathrm{T}}\left(\boldsymbol{s}_i^{(k)}+\mathbf{1}_{n_i}-\boldsymbol{\xi}_i^{(k)}-\boldsymbol{v}_i^{(k)}-b_i^{(k)}\boldsymbol{y}_i\right)\Big], \tag{14}$$

$$b_i^{(k+1)} = \frac{\boldsymbol{y}_i^{\mathrm{T}}(\boldsymbol{s}_i^{(k)} + \mathbf{1}_{n_i} - \boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)} - \boldsymbol{\xi}_i^{(k)} - \boldsymbol{v}_i^{(k)})}{\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{y}_i}, \tag{15}$$

where $\boldsymbol{I}_d$ is a $d \times d$ identity matrix and $\boldsymbol{I}_{n_i}$ is an $n_i \times n_i$ identity matrix.

Optimizing problem (9) is reduced to

$$\boldsymbol{z}^{(k+1)} = \arg\min_{\boldsymbol{z}}\ \frac{1}{2}\left\|\boldsymbol{z} - \left(\bar{\boldsymbol{w}}^{(k+1)} + \bar{\boldsymbol{u}}^{(k)}\right)\right\|_2^2 + \frac{1}{\rho_1 K}P_\lambda(\boldsymbol{z}), \tag{16}$$

where $\bar{\boldsymbol{w}}^{(k+1)} = \frac{1}{K}\sum_{i=1}^K \boldsymbol{w}_i^{(k+1)}$ and $\bar{\boldsymbol{u}}^{(k)} = \frac{1}{K}\sum_{i=1}^K \boldsymbol{u}_i^{(k)}$.

Observing $P_\lambda(\boldsymbol{z}) = \sum_{j=1}^d p_\lambda(z_j)$, we can obtain the solution to problem (16) by solving d independent univariate optimization problems. Let $\boldsymbol{\psi}_i^{(k+1)} = \boldsymbol{w}_i^{(k+1)} + \boldsymbol{u}_i^{(k)}$. Then it is clear that $\bar{\boldsymbol{\psi}}^{(k+1)} = \bar{\boldsymbol{w}}^{(k+1)} + \bar{\boldsymbol{u}}^{(k)}$, where $\bar{\boldsymbol{\psi}}^{(k+1)} = \frac{1}{K}\sum_{i=1}^K \boldsymbol{\psi}_i^{(k+1)}$. Therefore, optimizing problem (16) is equivalent to

$$z_j^{(k+1)} = \arg\min_{z_j}\ \frac{1}{2}\left(z_j - \bar{\psi}_j^{(k+1)}\right)^2 + \frac{1}{\rho_1 K}p_\lambda(z_j), \quad j = 1,\ 2,\ \ldots,\ d, \tag{17}$$

where subscript j denotes the j^{th} entry of the vector. The solution to problem (17) with many commonly used nonconvex penalties has been studied by Gong et al. (2013). We will give the closed-form solution to problem (17) for LSP, the SCAD penalty, MCP, and the capped-ℓ_1 penalty in Appendix A.

The closed-form solution to problem (10) can be calculated through $\partial\mathcal{L}/\partial\boldsymbol{\xi}_i = \mathbf{0}_{n_i}$, followed by the projection to the 1D nonnegative set $[0,+\infty)$; that is,

$$\boldsymbol{\xi}_i^{(k+\frac{1}{2})} = \boldsymbol{s}_i^{(k)} + \mathbf{1}_{n_i} - \boldsymbol{v}_i^{(k)} - \boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)} - b_i^{(k+1)}\boldsymbol{y}_i - \frac{\mathbf{1}_{n_i}}{n\rho_2}, \tag{18}$$

$$\boldsymbol{\xi}_i^{(k+1)} = \left[\boldsymbol{\xi}_i^{k+\frac{1}{2}}\right]_+. \tag{19}$$

Similarly, the solution to problem (11) can be calculated via $\partial\mathcal{L}/\partial\boldsymbol{s}_i = \mathbf{0}_{n_i}$. Therefore, we can perform a two-step update as follows:

$$\boldsymbol{s}_i^{(k+\frac{1}{2})} = \boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)} + b_i^{(k+1)}\boldsymbol{y}_i + \boldsymbol{\xi}_i^{(k+1)} - \mathbf{1}_{n_i} + \boldsymbol{v}_i^{(k)}, \tag{20}$$

$$\boldsymbol{s}_i^{(k+1)} = \left[\boldsymbol{s}_i^{k+\frac{1}{2}}\right]_+. \tag{21}$$

3.2 Implementation details

3.2.1 Naive parallel ADMM algorithm

Based on the algorithm derivation in Section 3.1, we design a corresponding parallel algorithm that uses K processors to collaboratively solve problem (1) in a parallel way.

As shown in Algorithm 1, all the primal and dual variables along with the iteration count k are first initialized (line 1). Then each processor i loads the corresponding feature matrix $\boldsymbol{X}_i$ and label vector $\boldsymbol{y}_i$, and calculates the variable $\boldsymbol{H}_i$ (lines 2–4). Then each processor keeps iterating until the predefined stopping criterion is satisfied (lines 5–25). The order of the variable update is the same as that listed for problems (7)–(13).

Note that there are two AllReduce operations in each iteration. The first AllReduce operation is used to calculate the global sum of $\boldsymbol{w}_i^{(k+1)}$ and $\boldsymbol{u}_i^{(k)}$ over all processors (line 9). The reduced result across all processors is then used to perform the update of $\boldsymbol{z}$ (line 11). The second AllReduce operation is used to calculate the global sum of $\sum_{j=1}^{n_i}\xi_{ij}$ over all processors (line 19). The reduced result across all processors is used to calculate the objective value of problem (2) in each iteration.

Algorithm 1 Naive parallel ADMM algorithm

Require: training data $\mathcal{S}$, parameters $\rho_1>0$, $\rho_2>0$, λ, θ

1: Initialize K processors, along with $\boldsymbol{w}_i^{(0)}$, $b_i^{(0)}$, $\boldsymbol{z}^{(0)}$, $\boldsymbol{\xi}_i^{(0)}$, $\boldsymbol{s}_i^{(0)}$, $\boldsymbol{u}_i^{(0)}$, $\boldsymbol{v}_i^{(0)}$, and $k\leftarrow 0$
2: **for** $i\in\{1,2,\ldots,K\}$ in parallel over processors **do**
3: Load data $\boldsymbol{X}_i$ and $\boldsymbol{y}_i$
4: Calculate $\boldsymbol{H}_i = \boldsymbol{Y}_i\boldsymbol{X}_i$
5: **loop**
6: Update $\boldsymbol{w}_i^{(k+1)}$ according to Eq. (14)
7: Update $b_i^{(k+1)}$ according to Eq. (15)
8: Calculate $\boldsymbol{\psi}_i^{(k+1)} = \boldsymbol{w}_i^{(k+1)} + \boldsymbol{u}_i^{(k)}$
9: AllReduce $\boldsymbol{\psi}_i^{(k+1)}$ and obtain $\sum_{i=1}^K \boldsymbol{\psi}_i^{(k+1)}$
10: Calculate $\bar{\boldsymbol{\psi}}^{(k+1)} = \frac{1}{K}\sum_{i=1}^K \boldsymbol{\psi}_i^{(k+1)}$
11: Update $\boldsymbol{z}^{(k+1)}$ according to Eq. (16)
12: Update $\boldsymbol{\xi}_i^{(k+\frac{1}{2})}$ according to Eq. (18)
13: Update $\boldsymbol{\xi}_i^{(k+1)} = \max\left(\boldsymbol{\xi}_i^{(k+\frac{1}{2})}, \mathbf{0}_{n_i}\right)$
14: Update $\boldsymbol{s}_i^{(k+\frac{1}{2})}$ according to Eq. (20)
15: Update $\boldsymbol{s}_i^{(k+1)} = \max\left(\boldsymbol{s}_i^{(k+\frac{1}{2})}, \mathbf{0}_{n_i}\right)$
16: Update $\boldsymbol{u}_i^{(k+1)} = \boldsymbol{u}_i^{(k)} + \left(\boldsymbol{w}_i^{(k+1)} - \boldsymbol{z}^{(k+1)}\right)$
17: Update $\boldsymbol{v}_i^{(k+1)} = \boldsymbol{v}_i^{(k)} + \left(\boldsymbol{\xi}_i^{(k+1)} - \boldsymbol{s}_i^{(k+1)} + \boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)} + b_i^{(k+1)}\boldsymbol{y}_i - \mathbf{1}_{n_i}\right)$
18: Calculate $\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)} = \mathbf{1}_{n_i}^{\mathrm{T}}\boldsymbol{\xi}_i^{(k+1)}$
19: AllReduce $\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)}$, obtain $\sum_{i=1}^K\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)}$
20: $k\leftarrow k+1$
21: **if** the stopping criterion is satisfied **then**
22: Break
23: **end if**
24: **end loop**
25: **end for**

Ensure: $\boldsymbol{w}_0^*$ and b_0^*

3.2.2 Optimized parallel ADMM algorithm

Unfortunately, the naive parallel algorithm shown in Algorithm 1 is quite time-consuming due to the high computation cost and synchronization cost. First, note that performing the update of $\boldsymbol{w}_i$ requires calculating the inversion of a $d\times d$ matrix, where d stands for the dimension of the training data. Obviously, the computation cost is pretty high especially for high-dimensional data. Second, it is known that the AllReduce operation is actually a synchronous operation. There are two synchronization operations in each iteration (lines 9 and 19), which

hurt the performance of the parallel algorithm considerably.

To improve the performance of the naive parallel algorithm, we optimize Algorithm 1 with respect to decreasing the computation and synchronization cost. The optimized parallel algorithm is shown in Algorithm 2.

1. Optimization of computation

To reduce the computation cost, we adopt an efficient scheme similar to Guan et al. (2018). For each processor i, the $\boldsymbol{w}_i$-update is performed according to the amount of training data n_i and the dimension of data d.

Let $\rho = \rho_1/\rho_2$ and $\boldsymbol{f}_i^{(k)} = \rho\left(\boldsymbol{z}^{(k)} - \boldsymbol{u}_i^{(k)}\right) + \boldsymbol{H}_i^{\mathrm{T}}\left(\boldsymbol{s}_i^{(k)} + \boldsymbol{1}_{n_i} - \boldsymbol{\xi}_i^{(k)} - \boldsymbol{v}_i^{(k)} - b_i^{(k)}\boldsymbol{y}_i\right)$. Then we can reformulate problem (14) as

$$\boldsymbol{w}_i^{(k+1)} = (\rho\boldsymbol{I}_d + \boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i)^{-1}\boldsymbol{f}_i^{(k)}. \tag{22}$$

If d is larger than n_i in order, we can apply the Sherman-Morrison formula (Sherman and Morrison, 1950) to solve Eq. (22). Therefore, we have

$$\boldsymbol{w}_i^{(k+1)} = \frac{\boldsymbol{f}_i^{(k)}}{\rho} - \frac{(\boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{U}_i^{-1}(\boldsymbol{L}_i^{-1}(\boldsymbol{H}_i\boldsymbol{f}_i^{(k)}))))}{\rho^2}, \tag{23}$$

where $\boldsymbol{L}_i$ and $\boldsymbol{U}_i$ are the Cholesky factorization of an $n_i \times n_i$ positive definite matrix $\boldsymbol{I}_{n_i} + \dfrac{1}{\rho}\boldsymbol{H}_i\boldsymbol{H}_i^{\mathrm{T}}$, i.e., $\boldsymbol{I}_{n_i} + \dfrac{1}{\rho}\boldsymbol{H}_i\boldsymbol{H}_i^{\mathrm{T}} = \boldsymbol{L}_i\boldsymbol{U}_i$.

For the case $n_i \geq d$, we can observe that the matrix $\rho\boldsymbol{I}_d + \boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i$ is positive definite; hence, we can obtain the solution of $\boldsymbol{w}_i^{(k+1)}$ via

$$\boldsymbol{w}_i^{(k+1)} = \boldsymbol{U}_i^{-1}\left(\boldsymbol{L}_i^{-1}\boldsymbol{f}_i^{(k)}\right), \tag{24}$$

where $\boldsymbol{L}_i$ and $\boldsymbol{U}_i$ are the Cholesky factorization of a $d \times d$ matrix $\rho\boldsymbol{I}_d + \boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i$, i.e., $\rho\boldsymbol{I}_d + \boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i = \boldsymbol{L}_i\boldsymbol{U}_i$.

Therefore, optimizing problem (7) is reduced to

$$\boldsymbol{w}_i^{(k+1)} = \begin{cases} \boldsymbol{U}_i^{-1}(\boldsymbol{L}_i^{-1}\boldsymbol{f}_i^{(k)}), & \text{if } n_i \geq d, \\ \dfrac{\boldsymbol{f}_i^{(k)}}{\rho} - \dfrac{(\boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{U}_i^{-1}(\boldsymbol{L}_i^{-1}(\boldsymbol{H}_i\boldsymbol{f}_i^{(k)}))))}{\rho^2}, & \text{otherwise,} \end{cases} \tag{25}$$

where $\boldsymbol{L}_i$ and $\boldsymbol{U}_i$ are the Cholesky factorization of $\rho\boldsymbol{I}_d + \boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i$ if $n_i \geq d$, and the Cholesky factorization of $\boldsymbol{I}_{n_i} + \dfrac{1}{\rho}\boldsymbol{H}_i\boldsymbol{H}_i^{\mathrm{T}}$ otherwise.

Algorithm 2 Optimized parallel ADMM algorithm

Require: training data $\mathcal{S}$, parameters $\rho_1 > 0$, $\rho_2 > 0$, λ, θ

Initialize K processors, along with $\boldsymbol{w}_i^{(0)}$, $b_i^{(0)}$, $\boldsymbol{z}^{(0)}$, $\boldsymbol{\xi}_i^{(0)}$, $\boldsymbol{s}_i^{(0)}$, $\boldsymbol{u}_i^{(0)}$, $\boldsymbol{v}_i^{(0)}$, and $k \leftarrow 0$
for $i \in \{1, 2, \ldots, K\}$ in parallel over processors **do**
  Load data $\boldsymbol{X}_i$ and $\boldsymbol{y}_i$
  Calculate $\boldsymbol{H}_i = \boldsymbol{Y}_i\boldsymbol{X}_i$ and $\rho = \rho_1/\rho_2$
  if $n_i \geq d$ **then**
    Form $\boldsymbol{C}_i = \rho\boldsymbol{I}_d + \boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i$
  else
    Form $\boldsymbol{C}_i = \boldsymbol{I}_{n_i} + \dfrac{1}{\rho}\boldsymbol{H}_i\boldsymbol{H}_i^{\mathrm{T}}$
  end if
  Calculate Cholesky factorization of $\boldsymbol{C}_i$ ($\boldsymbol{C}_i$=$\boldsymbol{L}_i\boldsymbol{U}_i$)
  loop
    Update $\boldsymbol{u}_i^{(k+1)} = \boldsymbol{u}_i^{(k)} + \left(\boldsymbol{w}_i^{(k)} - \boldsymbol{z}_i^{(k)}\right)$
    Calculate $\boldsymbol{f}_i^{(k)} = \rho\left(\boldsymbol{z}^{(k)} - \boldsymbol{u}_i^{(k+1)}\right) + \boldsymbol{H}_i^{\mathrm{T}}\left(\boldsymbol{s}_i^{(k)} + \boldsymbol{1}_{n_i} - \boldsymbol{\xi}_i^{(k)} - \boldsymbol{v}_i^{(k)} - b_i^{(k)}\boldsymbol{y}_i\right)$
    Update $\boldsymbol{w}_i^{(k+1)}$ according to Eq. (25)
    Update $b_i^{(k+1)}$ according to Eq. (15)
    Update $\boldsymbol{\xi}_i^{(k+\frac{1}{2})}$ according to Eq. (18)
    Update $\boldsymbol{\xi}_i^{(k+1)} = \max\left(\boldsymbol{\xi}_i^{(k+\frac{1}{2})}, \boldsymbol{0}_{n_i}\right)$
    Calculate $\boldsymbol{\psi}_i^{(k+1)} = \boldsymbol{w}_i^{(k+1)} + \boldsymbol{u}_i^{(k+1)}$ and $\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)} = \boldsymbol{1}_{n_i}^{\mathrm{T}}\boldsymbol{\xi}_i^{(k+1)}$
    Merge $\boldsymbol{\psi}_i^{(k+1)}$ and $\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)}$
    AllReduce $\boldsymbol{\psi}_i^{(k+1)}$ and $\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)}$
    Calculate $\bar{\boldsymbol{\psi}}^{(k+1)} = \dfrac{1}{K}\sum_{i=1}^{K}\boldsymbol{\psi}_i^{(k+1)}$
    Update $\boldsymbol{z}^{(k+1)}$ according to Eq. (16)
    Update $\boldsymbol{s}_i^{(k+\frac{1}{2})}$ according to Eq. (20)
    Update $\boldsymbol{s}_i^{(k+1)} = \max\left(\boldsymbol{s}_i^{(k+\frac{1}{2})}, \boldsymbol{0}_{n_i}\right)$
    Update $\boldsymbol{v}_i^{(k+1)} = \boldsymbol{v}_i^{(k)} + \left(\boldsymbol{\xi}_i^{(k+1)} - \boldsymbol{s}_i^{(k+1)} + \boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)} + b_i^{(k+1)}\boldsymbol{y}_i - \boldsymbol{1}_{n_i}\right)$
    $k \leftarrow k + 1$
    if the stopping criterion is satisfied **then**
      Break
    end if
  end loop
end for

Ensure: $\boldsymbol{w}_0^*$ and b_0^*

Note that for each processor i, $\boldsymbol{H}_i$ remains unchanged throughout the parallel algorithm, and ρ_1 and ρ_2 are two constants. Thus, we can carry out Cholesky factorization according to the values of d and n_i once, cache the factorization results, and then use the cached results in the subsequent steps. As shown in Algorithm 2, based on the values of n_i and

d, each processor first forms an intermediate variable $\boldsymbol{C}_i$, a $d\times d$ or $n_i\times n_i$ matrix, and then factors it (lines 2–9). In this way, each processor can use the cached Cholesky factorization results to perform the update of $\boldsymbol{w}_i$ (lines 13 and 14) in the whole iteration procedure.

2. Optimization of synchronization

As mentioned before, the naive parallel algorithm is not synchronously efficient because there are two AllReduce operations per iteration. We know that the AllReduce is actually a synchronous operation, which hurts the performance of the parallel algorithm considerably. Hence, we further optimize Algorithm 1 to decrease the performance damage caused by the two synchronous operations. The main ideas of synchronization optimization include altering the order of the variables update and merging the messages.

As mentioned before, in each iteration, we need to calculate $\sum_{i=1}^K \boldsymbol{\psi}_i^{(k+1)}$ and $\sum_{i=1}^K\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)}$, which can be implemented via two AllReduce operations. Note that $\sum_{i=1}^K \boldsymbol{\psi}_i^{(k+1)}$ is the global sum of $\boldsymbol{w}_i^{(k+1)}$ and $\boldsymbol{u}_i^{(k)}$ over all processors, and $\sum_{i=1}^K\sum_{j=1}^{n_i}\xi_{ij}^{(k+1)}$ is the global sum of $\sum_{j=1}^{n_i}\xi_j^{(k+1)}$ over all processors. Hence, we can merge $\boldsymbol{w}_i^{(k+1)}+\boldsymbol{u}_i^{(k)}$ and $\sum_{j=1}^{n_i}\xi_j^{(k+1)}$ into a single message, perform one AllReduce operation, and calculate their global sum over all processors simultaneously.

As shown in Algorithm 2, the $\boldsymbol{\xi}_i$-update is carried out ahead of the $\boldsymbol{z}$-update (lines 16 and 17). Then each processor calculates $\boldsymbol{\psi}_i^{(k+1)}$ and $\sum_{j=1}^{n_i}\xi_j^{(k+1)}$, merges them into one message, and then calculates their global sum over all processors via one AllReduce operation (lines 18–20). After that, $\bar{\boldsymbol{\psi}}^{(k+1)}=\frac{1}{K}\sum_{i=1}^K\boldsymbol{\psi}_i^{(k+1)}$ is carried out at line 21, followed by the $\boldsymbol{z}$-update at line 22. Note that in the optimized parallel algorithm, we alter the order of $\boldsymbol{u}_i$-update (line 12). Hence, in the subsequent variable update, the superscript of $\boldsymbol{u}_i$ should be $k+1$ instead of k. In fact, when running with one processor (thread), Algorithm 1 is reduced to the serial algorithm proposed by Guan et al. (2018) except that variable $\boldsymbol{u}_i$ is updated first in the order.

3.3 Time complexity analysis

In this subsection, we discuss the time complex of Algorithms 1 and 2. Here, we do not consider any special structure (e.g., sparsity) inside the matrix. We neglect the superscript of each variable for convenience. We also make two assumptions to do the time complexity analysis. First, each synchronization operation has the same time cost (denoted as T_{syn}). Second, each floating point operation (flop) takes a cost of one unit of time.

1. Time complexity of Algorithm 1

We first consider the computation complexity of processor i ($i=1,2,\dots,K$). Since $\boldsymbol{Y}_i\in\mathbb{R}^{n_i\times n_i}$ and $\boldsymbol{X}_i\in\mathbb{R}^{n_i\times d}$, line 4 can be performed at a cost of $O(dn_i^2)$ flops. In terms of the update of $\boldsymbol{w}_i$, we can first form $(\rho_1\boldsymbol{I}_d+\rho_2\boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i)^{-1}$ at a cost of $O(dn_i^2+d^3)$ flops. Following that, $\rho_1(\boldsymbol{z}-\boldsymbol{u}_i)+\rho_2\boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{s}_i+\boldsymbol{1}_{n_i}-\boldsymbol{\xi}_i-\boldsymbol{v}_i-b_i\boldsymbol{y}_i)$ can be calculated at a cost of $O(dn_i^2)$ flops. As a result, updating $\boldsymbol{w}_i$ takes $O(dn_i^2+d^3)$ flops. Similarly, we can easily find that lines 7, 8, and 10–18 take a total cost of $O(dn_i)$ flops, which is negligible compared with the cost of updating $\boldsymbol{w}_i$. Therefore, the average time complexity of Algorithm 1 is

$$\frac{1}{K}\sum_{i=1}^K(O(dn_i^2+d^3)+2\,T_{\mathrm{syn}})\cdot \mathrm{Ite}_i, \tag{26}$$

where Ite_i represents the iteration number of the i^{th} processor.

2. Time complexity of Algorithm 2

We first consider the computation cost of the pre-computation stage (lines 4–10) of processor i. It is clear that line 4 can be performed at a cost of $O(dn_i^2)$ flops. Then we can form the intermediate variable $\boldsymbol{C}$ with a cost of $O(d^2n_i)$ flops if $n_i\geq d$, and with a cost of $O(dn_i^2)$ flops otherwise. Note that the computation cost of line 10 is $O(d^3)$ flops if $n_i\geq d$, and $O(n_i^3)$ flops otherwise. If $n_i\geq d$, $O(d^2n_i)$+$O(d^3)$=$O(d^2n_i)$; otherwise, $O(dn_i^2)$+$O(n_i^3)$=$O(dn_i^2)$. As a result, performing the pre-computation stage takes a total computation cost of $O(d^2n_i)$ flops if $n_i\geq d$, and $O(dn_i^2)$ flops otherwise. Following that, we can calculate the cost of the iteration stage. In terms of line 14, if $n_i\geq d$, then $\boldsymbol{U}_i^{-1}(\boldsymbol{L}_i^{-1}\boldsymbol{f}_i)$ can be performed through two back-solve steps with a cost of $O(dn_i)$ flops. Otherwise, $\boldsymbol{H}_i^{\mathrm{T}}(\boldsymbol{U}_i^{-1}(\boldsymbol{L}_i^{-1}(\boldsymbol{H}_i\boldsymbol{f}_i)))$ can be performed through two matrix-vector multiplications and two back-solve steps with a cost of $O(dn_i)$ flops. As a result, it is clear that the cost of line 14 is $O(dn_i)$ flops. In addition, it is easy to see that lines 12–19, 21–29 take a total cost of $O(dn_i)$ flops per iteration. Finally, we can obtain the average time complexity

of Algorithm 2 as

$$\begin{cases} \dfrac{1}{K}\sum_{i=1}^{K} O(d^2 n_i) + (O(dn_i) + T_{\text{syn}}) \cdot \text{Ite}_i, \\ \qquad\qquad\qquad\qquad \text{if } n_i \geq d, \\ \dfrac{1}{K}\sum_{i=1}^{K} O(dn_i^2) + (O(dn_i) + T_{\text{syn}}) \cdot \text{Ite}_i, \\ \qquad\qquad\qquad\qquad \text{otherwise.} \end{cases} \tag{27}$$

Comparing Eq. (26) with Eq. (27), we can clearly see that the average time complexity of Algorithm 2 is much lower than that of Algorithm 1. Two main factors contribute to this conclusion. First, Algorithm 1 does computation-intensive calculation in each iteration. In contrast, Algorithm 2 performs only a computation-intensive operation at the precomputation stage, and the time cost of its iteration stage is pretty low, resulting in a much lower total computation cost. Second, the synchronization cost of Algorithm 2 is much lower than that of Algorithm 1 because Algorithm 2 needs fewer synchronization operations per iteration.

4 Convergence analysis

In this section, we present a detailed convergence analysis of the proposed parallel algorithms. Note that the analysis of Algorithm 2 is very similar to that of Algorithm 1; therefore, we just consider Algorithm 1. To present the analysis, we first modify the method for updating $\boldsymbol{z}^{(k+1)}$ a little; that is,

$$\boldsymbol{z}^{(k+1)} = \arg\min_{\boldsymbol{z}} \ \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i{}^{(k)}\right) + \frac{\beta}{2}\|\boldsymbol{z} - \boldsymbol{z}^{(k)}\|^2, \tag{28}$$

where β is a small positive number. If $\beta = 0$, problem (28) is reduced to problem (9); if β is a very small value, problem (9) can be represented by problem (28) approximately. Our analysis follows the proof framework built in Wang et al. (2015), which has also been used in Sun et al. (2017a, 2017b, 2018a).

We need the following two assumptions:

Assumption 1 For any k and $i \in \{1, 2, \ldots, K\}$, $\boldsymbol{v}_i^{(k)} \in \text{Im}(\boldsymbol{y}_i)$.

Assumption 2 The augmented Lagrangian function $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{z}, \boldsymbol{\xi}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v})$ has a lower bound; that is, $\inf \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{z}, \boldsymbol{\xi}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v}) > -\infty$.

Now we introduce several definitions and properties needed in the analysis.

Definition 1 We say $f(x)$ is strongly convex with constant $\delta \geq 0$, if the function $f(x) - \delta\|x\|^2/2$ is also convex.

If a function is strongly convex, the fact naturally holds. Let x^* be a minimizer of f, which is strongly convex with constant δ. Then it holds that

$$f(x) - f(x^*) \geq \frac{\delta}{2}\|x - x^*\|^2.$$

To simplify the presentation, for any $i \in \{1, 2, \ldots, K\}$, we use

$$\boldsymbol{D}_i^{(k)} := (\boldsymbol{w}_i^{(k)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}).$$

Now, we are prepared to present the convergence analysis of our algorithm.

Lemma 1 Let $\{\boldsymbol{D}_i^{(k)}\}_{k=0,1,\ldots}$ be generated by our algorithm. Then

$$\begin{aligned} &\mathcal{L}(\boldsymbol{w}_i^{(k)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}) \\ \geq &\mathcal{L}(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k+1)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}) \\ &+ \frac{\nu}{2}\|\boldsymbol{D}_i^{(k+1)} - \boldsymbol{D}_i^{(k)}\|^2, \end{aligned} \tag{29}$$

where

$$\nu := \min\{\rho_1 + \rho_2 \sigma_{\min}(\boldsymbol{H}_i^{\mathrm{T}} \boldsymbol{H}_i), \rho_2, \rho_2\|\boldsymbol{y}_i\|^2, \beta\}.$$

The proof of Lemma 1 is given in Appendix B.

Lemma 2 If Assumption 1 holds, we have

$$\begin{aligned} \|\boldsymbol{v}_i^{(k+1)} - \boldsymbol{v}_i^{(k)}\|^2 \leq\ & c_1\|\boldsymbol{D}_i^{(k+1)} - \boldsymbol{D}_i^{(k)}\|^2 \\ &+ c_2\|\boldsymbol{D}_i^{(k+2)} - \boldsymbol{D}_i^{(k+1)}\|^2 \end{aligned} \tag{30}$$

and

$$\begin{aligned} \|\boldsymbol{u}_i^{(k+1)} - \boldsymbol{u}_i^{(k)}\|^2 \leq\ & c_3\|\boldsymbol{D}_i^{(k+1)} - \boldsymbol{D}_i^{(k)}\|^2 \\ &+ c_4\|\boldsymbol{D}_i^{(k+2)} - \boldsymbol{D}_i^{(k+1)}\|^2, \end{aligned} \tag{31}$$

where $c_1,\ c_2 > 0$ are polynomial compositions of $\|\boldsymbol{H}_i\|$ and $\|\boldsymbol{y}_i\|$ respectively, $c_3 = O\left(1/\rho_1^2\right)$, and $c_4 = O\left(1/\rho_1^2\right)$.

The proof of Lemma 2 is given in Appendix C.

Theorem 1 If Assumptions 1 and 2 hold, and

$$\frac{\nu}{2} > \rho_1 c_3 + \rho_2 c_1 + \rho_2 c_2 + \rho_1 c_4, \tag{32}$$

then for any fixed $i \in \{1, 2, \ldots, K\}$,

$$\lim_k \|\boldsymbol{D}_i^{(k+1)} - \boldsymbol{D}_i^{(k)}\| = 0.$$

The proof of Theorem 1 is given in Appendix D. For any $\boldsymbol{w}_i^*$ being the stationary point, there exists subsequence $\boldsymbol{w}_i^{k_j} \to \boldsymbol{w}_i^*$, and with Eq. (D5),

$\boldsymbol{w}_i^{k_j+1} \to \boldsymbol{w}_i^*$. If P is convex, we can see that $\boldsymbol{w}_i^*$ can minimize problem (3).

Now, we claim that inequality (32) can always be satisfied. This is because the parameters ρ_1 and ρ_2 are set by users. Noting that $c_3 = O\left(1/\rho_1^2\right)$ and $c_4 = O\left(1/\rho_1^2\right)$ (proved in Lemma 2), we have

$$\lim_{\rho_1 \to +\infty, \rho_2 \to 0} \rho_1 c_3 + \rho_2 c_1 + \rho_2 c_2 + \rho_1 c_4 = 0.$$

Thus, for any $\beta > 0$, we can choose a large enough ρ_1 and a small enough ρ_2 such that inequality (32) is satisfied.

5 Experimental evaluation

5.1 Experimental setup

All experiments were conducted on a cluster of four computing nodes interconnected with a Gigabit Ethernet. Each node was equipped with a Quad-Core Intel Xeon CPU E5-1620 (3.60 GHz) and 47 GB memory. The cluster could run up to 32 processors in total. Both Algorithms 1 and 2 were implemented in C++ with the Eigen library. Moreover, the message passing interface (MPI) implementation MPICH (version 3.0.2) was used for inter-processor communication.

We conducted experiments on four LIBSVM benchmark datasets, which are summarized in Table 2. We considered the SCAD penalized SVM in all of the experimental evaluations. We compared the proposed parallel algorithm (i.e., Algorithm 2) mainly with the naive parallel ADMM algorithm (i.e., Algorithm 1) and the serial ADMM algorithm proposed by Guan et al. (2018).

Table 2 Datasets used in the experiments*

Dataset	Number of training samples	Number of testing samples	Number of features	Sparsity (%)
a2a	2000	265	123	11.65
mushrooms	7300	824	112	18.75
news20	18 000	1996	1 355 191	0.03
rcv1	18 000	2242	47 236	0.16

* https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

In terms of parameter setting, we set $\theta = 3.7$ as suggested in Fan and Li (2001). Empirically, we set $\lambda = 2^{-6}$ for all the evaluations. Meanwhile, the tuning parameters ρ_1 and ρ_2 were selected by a grid search over $\{0.01, 0.1, 1, 1.5, 5, 10\}$. All primal and dual variables were initialized as zero vectors (zero scalar for b_i) to guarantee a fair comparison.

In our experiments, the termination condition of the parallel algorithm was designed by measuring the change of the objective value between consecutive iterations. To maintain consistency across the processors, we defined the relative change of the objective value as $\epsilon = \left|\left(\text{obj}^{(k+1)} - \text{obj}^{(k)}\right)/\text{obj}^{(k)}\right|$, where $\text{obj}^{(k)} = \frac{1}{n}\mathbf{1}^{\mathrm{T}}\boldsymbol{\xi}^{(k)} + P(\boldsymbol{z}^{(k)})$. Both Algorithms 1 and 2 were terminated when $\epsilon < 10^{-4}$ or the number of iterations exceeded 1000.

5.2 Comparison with the naive parallel algorithm

In the first experiment, we compared the optimized parallel algorithm with the naive parallel algorithm on the multi-core computing node. Different numbers of processors, including 1, 2, 4, and 8, were chosen for the evaluation of the performance of these two parallel algorithms. We conducted the evaluations on two small-scale datasets (a2a and mushrooms). Here, we did not conduct the evaluations on news20 and rcv1 because the running time of Algorithm 1 was unacceptably long on these two large-scale datasets.

In this part, we split the running time of both parallel algorithms into three parts: pre-computation time, iteration time, and synchronization time. These three components are described as follows:

1. Pre-computation time: This indicates the time spent on carrying out pre-computations ahead of the iteration procedure.

2. Iteration time: This indicates the total time spent on carrying out numerical calculations inside the iteration procedure.

3. Synchronization time: This indicates the total time spent on performing AllReduce operations.

Figs. 1a and 1b illustrate the performance breakdown results. In addition, Tables 3 and 4 summarize the performance comparison of Algorithms 1 and 2. In both tables, time indicates the running time in CPU seconds, and accuracy refers to the prediction accuracy on the test datasets. The following conclusions can be drawn based on the observation of the experimental results:

1. The running time of both Algorithms 1 and 2 tends to decrease as the number of processors

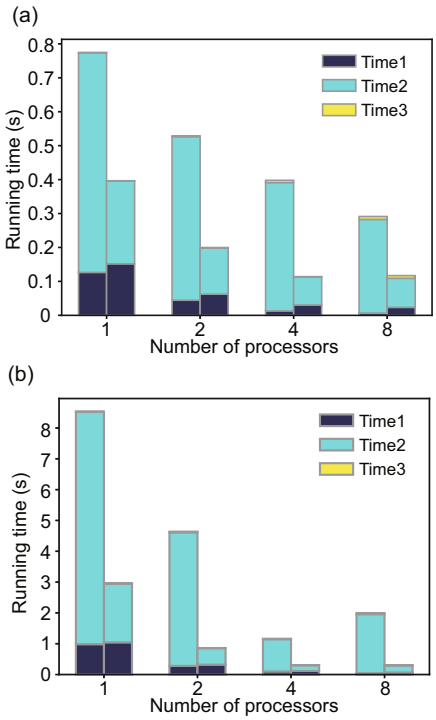

Fig. 1 Performance breakdown: (a) a2a; (b) mushrooms. Time1 denotes the pre-computation time; time2 refers to the iteration time; time3 stands for the synchronization time. All variables were measured in CPU seconds

increases. In comparison with Algorithm 1, Algorithm 2 has much shorter running time, indicating that the optimized parallel algorithm has a much lower time complexity. Besides, Algorithms 1 and 2 are comparable in terms of prediction accuracy.

2. For both evaluated datasets, the pre-computation time for Algorithms 1 and 2 is about the same. However, the iteration time of Algorithm 2 is much shorter, which makes it run much faster. On average, the speedup of Algorithm 2 over Algorithm 1 is 3.5× and 6.5× on a2a and mushrooms, respectively.

3. The experimental results demonstrate that the optimized parallel algorithm outperforms the naive parallel algorithm in terms of execution speed.

5.3 Comparison with the serial algorithm

In the second experiment, we compared the proposed parallel algorithm with the serial algorithm (Guan et al., 2018) on a single computing node to demonstrate that our parallel algorithm can fully use the parallel computing power of a multi-core computing node. As mentioned in Section 3.2.2, Algorithm 2 can be reduced to the serial algorithm proposed by Guan et al. (2018) in the case of with one processor. Therefore, we ran Algorithm 2 with one processor to simulate the performance of the serial algorithm. In addition, we set the number of processors to be eight to simulate the performance of our parallel algorithm on the multi-core computing node.

Table 3 Performance comparison on the a2a dataset

Number of processors	Method	Time (s)	Accuracy (%)
1	Algorithm 1	0.65	84.52
	Algorithm 2	0.24	84.53
2	Algorithm 1	0.48	84.72
	Algorithm 2	0.14	84.53
4	Algorithm 1	0.38	84.15
	Algorithm 2	0.08	84.25
8	Algorithm 1	0.28	83.92
	Algorithm 2	0.09	83.92

Table 4 Performance comparison on the mushrooms dataset

Number of processors	Method	Time (s)	Accuracy (%)
1	Algorithm 1	7.54	100
	Algorithm 2	1.91	100
2	Algorithm 1	4.36	100
	Algorithm 2	0.54	100
4	Algorithm 1	1.06	100
	Algorithm 2	0.19	100
8	Algorithm 1	1.95	100
	Algorithm 2	0.23	100

Figs. 2a and 3a illustrate the objective function values in terms of CPU time. Figs. 2b and 3b show the prediction accuracy in terms of CPU time. In all the figures, the solid lines stand for the parallel algorithm with eight processors, while the dashed lines refer to the parallel algorithm with one processor (i.e., the serial algorithm). Table 5 summarizes the experimental results. From the experimental results shown above, one can draw the following conclusions:

1. On the two large-scale datasets evaluated, the parallel algorithm runs much faster and needs much shorter running time than the serial version. Compared with the serial algorithm, our parallel algorithm attains an amazing performance improvement, with a factor of over 25× speedup on the news20 dataset and a factor of over 134× speedup on the rcv1 dataset. This demonstrates the efficiency of our parallel algorithm especially in using the parallel computing power of the multi-core cluster node.

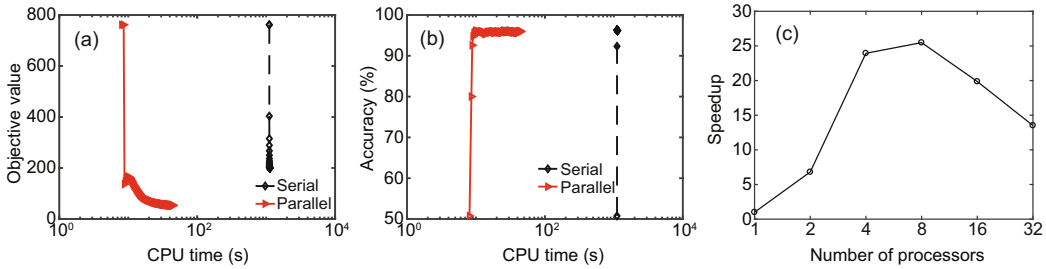

Fig. 2 Evaluation results on the news20 dataset: (a) objective; (b) accuracy; (c) speedup

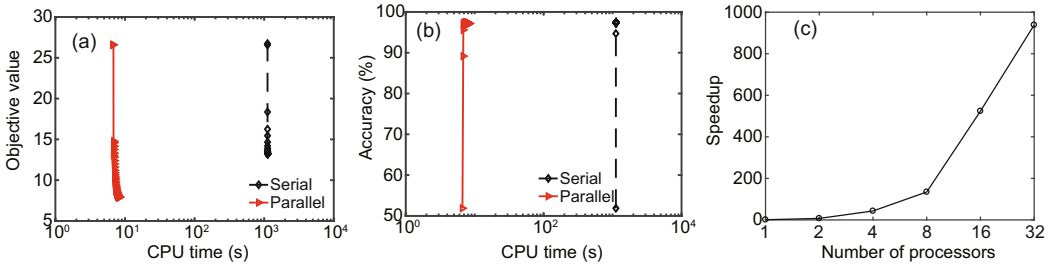

Fig. 3 Evaluation results on the rcv1 dataset: (a) objective; (b) accuracy; (c) speedup

Table 5 Comparison of the parallel algorithm with the serial version on a single computing node

Dataset	Method	Number of iterations	Time (s)	Accuracy (%)
news20	Serial	37	1129.17	96.14
	Parallel	145	44.31	95.99
rcv1	Serial	27	1126.10	97.37
	Parallel	56	8.37	97.34

2. Both the objective values of the serial and parallel algorithms start to go down at nonzero time points. This phenomenon validates the fact that both the parallel and serial algorithms need to do pre-computation before the iteration starts. However, the curves for the serial and parallel algorithms start to decline at different time points. Notably, the serial algorithm lags behind the parallel algorithm considerably with respect to the starting point. This demonstrates that the serial algorithm spends much more time on the pre-computation stage.

3. The parallel and serial algorithms have very comparable prediction accuracy, despite that the parallel algorithm requires a greater number of iterations to converge.

5.4 Evaluation of the optimized parallel algorithm

In the third experiment, we evaluated the scalability of the proposed parallel algorithm by running Algorithm 2 with 1, 2, 4, 8, 16, and 32 processors. Moreover, the main factors affecting the performance of the parallel algorithm were investigated by discussing the breakdown results.

5.4.1 Scalability

In this experiment, we varied the number of processors from 1 to 32, and plotted the speedup of our parallel algorithm. We measured the speedup according to the following criterion:

$$\text{speedup} = \frac{\text{running time with one processor}}{\text{running time with } p \text{ processors}}.$$

In Figs. 2c and 3c, we plotted the speedup of the parallel algorithm on the news20 and rcv1 datasets, respectively. These two figures show the strong scalability of our parallel algorithm. On the other hand, we see that the parallel algorithm performed better on the rcv1 dataset. When evaluated on the news20 dataset, the speedup curve rose rapidly as the number of processors increased from 1 to 8, followed by a sudden drop after the number of processors exceeded 8. In contrast, the speedup curve on the rcv1 dataset showed a continuous increase. Meanwhile, we see that the parallel algorithm attained a much higher speedup on the rcv1 dataset. We will discuss the main factors affecting the performance of the parallel algorithm in Section 5.4.2.

5.4.2 Breakdown results

To precisely analyze our parallel algorithm, in this subsection, we present the breakdown results for running time with different numbers of processors.

Figs. 4a and 4b present the breakdown results for the running time with different numbers of processors. Table 6 summarizes the corresponding results. Based on the observation of the breakdown results, we can draw the following conclusions:

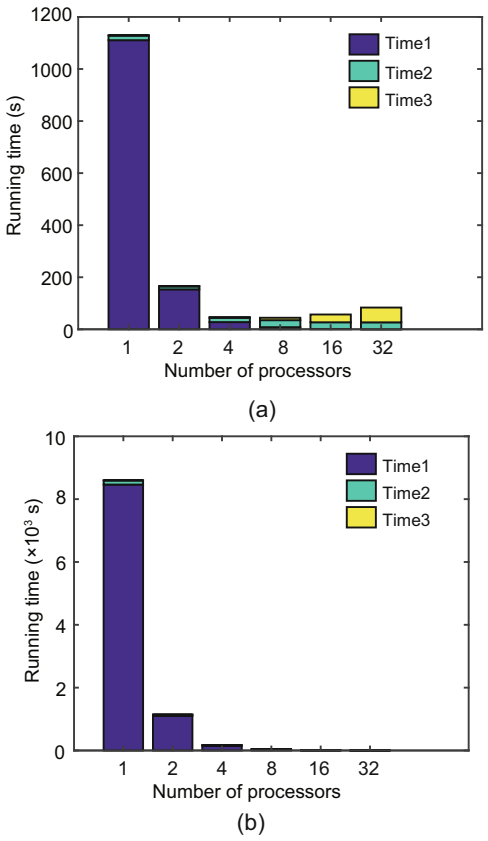

Fig. 4 Breakdown results for the running time: (a) news20; (b) rcv1

1. The parallel algorithm can always converge within 200 iterations on the evaluated datasets. Running with more processors tends to achieve a shorter running time.

2. When the number of processors is relatively small, the running time of the parallel algorithm is dominated by the pre-computation time. Specifically, when running with one processor, the pre-computation time has an overwhelming percentage. This in part reflects the inefficiency of the Eigen library in performing large-scale Cholesky factorization. More importantly, it reveals that the main burden of the serial algorithm lies in the pre-computation stage, in which the Cholesky factorization of an $n \times n$ or a $d \times d$ matrix needs to be performed.

3. The pre-computation time drops sharply as the number of processors increases. The parallel algorithm could greatly decrease the pre-computation cost by dividing the feature matrix into K parts and performing the Cholesky factorization of K $n_i \times n_i$ ($\sum_{i=1}^{K} n_i = n$) or $d \times d$ matrices in parallel. When scaling to a large number of processors, the pre-computation time occupies only a small percentage of the total running time. Thus, the parallel algorithm enjoys much shorter execution time.

4. As the number of processors increases, the iteration and synchronization time comes to dominate the running time. On the other hand, we see that the synchronization time on the news20 dataset is much

Table 6 Breakdown results for the running time

Dataset	Number of processors	Number of iterations	Time1 (s)	Time2 (s)	Time3 (s)	Time (s)
news20	1	37	1110.66	18.13	0.38	1129.17
	2	55	153.39	11.35	1.07	165.81
	4	112	27.64	15.66	3.85	47.16
	8	145	8.14	26.89	9.28	44.31
	16	157	1.36	25.58	29.90	56.83
	32	175	0.30	26.39	56.84	83.53
rcv1	1	27	1116.52	9.57	0.01	1126.10
	2	32	149.18	3.07	0.15	152.40
	4	48	24.48	1.57	0.09	26.14
	8	56	6.67	1.46	0.25	8.37
	16	59	1.10	0.59	0.46	2.15
	32	60	0.12	0.36	0.71	1.20

Time1: pre-computation time; time2: iteration time; time3: synchronization time; time: total running time. Time is measured in CPU seconds

more than that on the rcv1 dataset. This is because the dimension of the news20 dataset is much larger than that of the rcv1 dataset. A larger dimension requires more communication time and thus leads to a higher synchronization cost.

6 Conclusions and future work

To enable large-scale optimization of the nonconvex penalized SVMs, we have proposed, analyzed, and evaluated an efficient algorithm in a parallel and distributed environment. Our parallel algorithm was designed based on the consensus ADMM. We adopted many efficient schemes to decrease the computation and synchronization costs of the proposed parallel algorithm. Moreover, the convergence of the algorithm has been guaranteed. Experimental evaluations on four LIBSVM datasets demonstrated the efficiency of the proposed parallel algorithm. Our parallel algorithm was well-suited for handling nonconvex penalized SVMs in a parallel and distributed fashion.

The possible future work includes employing the heavy-ball techniques, which have been deeply studied in both convex and nonconvex settings (Ochs et al., 2014; Sun et al., 2018b), to accelerate our proposed algorithms.

Compliance with ethics guidelines

Lei GUAN, Tao SUN, Lin-bo QIAO, Zhi-hui YANG, Dong-sheng LI, Ke-shi GE, and Xi-cheng LU declare that they have no conflict of interest.

References

Allen-Zhu Z, Hazan E, 2016. Variance reduction for faster non-convex optimization. Proc 33rd Int Conf on Machine Learning, p.699-707.

Boyd S, Parikh N, Chu E, et al., 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found Trends Mach Learn*, 3(1):1-122. https://doi.org/10.1561/2200000016

Candès EJ, Wakin MB, Boyd SP, 2008. Enhancing sparsity by reweighted ℓ_1 minimization. *J Fourier Anal Appl*, 14(5-6):877-905. https://doi.org/10.1007/s00041-008-9045-x

Daneshmand A, Facchinei F, Kungurtsev V, et al., 2015. Hybrid random/deterministic parallel algorithms for convex and nonconvex big data optimization. *IEEE Trans Signal Process*, 63(15):3914-3929. https://doi.org/10.1109/TSP.2015.2436357

di Lorenzo P, Scutari G, 2015. Distributed nonconvex optimization over networks. IEEE 6th Int Workshop on Computational Advances in Multi-sensor Adaptive Processing, p.229-232. https://doi.org/10.1109/CAMSAP.2015.7383778

Facchinei F, Scutari G, Sagratella S, 2015. Parallel selective algorithms for nonconvex big data optimization. *IEEE Trans Signal Process*, 63(7):1874-1889. https://doi.org/10.1109/TSP.2015.2399858

Fan JQ, Li RZ, 2001. Variable selection via nonconcave penalized likelihood and its oracle properties. *J Am Stat Assoc*, 96(456):1348-1360. https://doi.org/10.1198/016214501753382273

Gong PH, Zhang CS, Lu ZS, et al., 2013. A general iterative shrinkage and thresholding algorithm for non-convex regularized optimization problems. Proc Int Conf on Machine Learning, p.37-45.

Guan L, Qiao LB, Li DS, et al., 2018. An efficient ADMM-based algorithm to nonconvex penalized support vector machines. Proc 18th IEEE Int Conf on Data Mining, p.1209-1216. https://doi.org/10.1109/ICDMW.2018.00173

Hong MY, Luo ZQ, Razaviyayn M, 2016. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM J Optim*, 26(1):337-364. https://doi.org/10.1137/140990309

Laporte L, Flamary R, Canu S, et al., 2014. Nonconvex regularizations for feature selection in ranking with sparse SVM. *IEEE Trans Neur Netw Learn Syst*, 25(6):1118-1130. https://doi.org/10.1109/TNNLS.2013.2286696

Liu HC, Yao T, Li RZ, 2016. Global solutions to folded concave penalized nonconvex learning. *Ann Stat*, 44(2):629-659. https://doi.org/10.1214/15-AOS1380

Mazumder R, Friedman JH, Hastie T, 2011. SparseNet: coordinate descent with nonconvex penalties. *J Am Stat Assoc*, 106(495):1125-1138. https://doi.org/10.1198/jasa.2011.tm09738

Ochs P, Chen YJ, Brox T, et al., 2014. iPiano: inertial proximal algorithm for nonconvex optimization. *SIAM J Imag Sci*, 7(2):1388-1419. https://doi.org/10.1137/130942954

Razaviyayn M, Hong MY, Luo ZQ, et al., 2014. Parallel successive convex approximation for nonsmooth nonconvex optimization. Proc 27th Int Conf on Neural Information Processing Systems, p.1440-1448.

Reddi SJ, Sra S, Poczos B, et al., 2016. Proximal stochastic methods for nonsmooth nonconvex finite-sum optimization. Proc 30th Conf on Neural Information Processing Systems, p.1145-1153.

Scutari G, Facchinei F, Lampariello L, 2017. Parallel and distributed methods for constrained nonconvex optimization—Part I: theory. *IEEE Trans Signal Process*, 65(8):1929-1944. https://doi.org/10.1109/TSP.2016.2637317

Sherman J, Morrison WJ, 1950. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann Math Stat*, 21(1):124-127. https://doi.org/10.1214/aoms/1177729893

Sun T, Jiang H, Cheng LZ, et al., 2017a. A convergence frame for inexact nonconvex and nonsmooth algorithms and its applications to several iterations. https://arxiv.org/abs/1709.04072

Sun T, Jiang H, Cheng LZ, 2017b. Iteratively linearized reweighted alternating direction method of multipliers for a class of nonconvex problems. https://arxiv.org/abs/1709.00483

Sun T, Yin PH, Cheng LZ, et al., 2018a. Alternating direction method of multipliers with difference of convex functions. *Adv Comput Math*, 44(3):723-744. https://doi.org/10.1007/s10444-017-9559-3

Sun T, Yin PH, Li DS, et al., 2018b. Non-ergodic convergence analysis of heavy-ball algorithms. https://arxiv.org/abs/1811.01777v2

Sun Y, Scutari G, Palomar D, 2016. Distributed non-convex multiagent optimization over time-varying networks. Proc 50th Asilomar Conf on Signals, Systems and Computers, p.788-794. https://doi.org/10.1109/ACSSC.2016.7869154

Wang Y, Yin WT, Zeng JS, 2015. Global convergence of ADMM in nonconvex nonsmooth optimization. https://arxiv.org/abs/1511.06324

Zhang CH, 2010. Nearly unbiased variable selection under minimax concave penalty. *Ann Stat*, 38(2):894-942. https://doi.org/10.1214/09-AOS729

Zhang HH, Ahn J, Lin XD, et al., 2006. Gene selection using support vector machines with non-convex penalty. *Bioinformatics*, 22(1):88-95. https://doi.org/10.1093/bioinformatics/bti736

Zhang SB, Qian H, Gong XJ, 2016. An alternating proximal splitting method with global convergence for nonconvex structured sparsity optimization. Proc 30th AAAI Conf on Artificial Intelligence, p.2330-2336.

Zhang T, 2010. Analysis of multi-stage convex relaxation for sparse regularization. *J Mach Learn Res*, 11:1081-1107.

Zhang X, Wu YC, Wang L, et al., 2016. Variable selection for support vector machines in moderately high dimensions. *J R Stat Soc Ser B*, 78(1):53-76. https://doi.org/10.1111/rssb.12100

Zhang YM, Li DS, Guo CX, et al., 2017. CubicRing: exploiting network proximity for distributed in-memory key-value store. *IEEE/ACM Trans Netw*, 25(4):2040-2053. https://doi.org/10.1109/TNET.2017.2669215

Appendix A: Closed-form solution of z-update

We present the closed-form solution to problem (17) for LSP, SCAD penalty, MCP, and capped-ℓ_1 penalty. All results can be directly obtained by applying the conclusions drawn in Gong et al. (2013). Here, $\bar{\psi}_i^{(k)}$ indicates the i^{th} entry of variable $\bar{\boldsymbol{\psi}}$ in iteration k.

1. LSP: $z_j^{(k+1)}=\operatorname{sign}\left(\bar{\psi}_j^{(k+1)}x\right)$, where $x=\arg\min_{z_j\in\mathcal{C}}\ \frac{1}{2}\left(z_j-|\bar{\psi}_j^{(k+1)}|\right)^2+\frac{\lambda}{\rho_1 K}\log(1+z_j/\theta)$ and $\mathcal{C}$ is a set composed of three elements or one element.

If $(\rho_1K)^2\left(|\bar{\psi}_j^{(k+1)}|-\theta\right)^2-4\rho_1K\left(\lambda-\rho_1K|\bar{\psi}_j^{(k+1)}|\theta\right)\geq 0$,

$$\mathcal{C}=\left\{0,\left[\left(\rho_1K\left(|\bar{\psi}_j^{(k+1)}|-\theta\right)+\sqrt{(\rho_1K)^2\left(|\bar{\psi}_j^{(k+1)}|-\theta\right)^2-4\rho_1K\left(\lambda-\rho_1K|\bar{\psi}_j^{(k+1)}|\theta\right)}\right)\Big/(2\rho_1K)\right]_+,\right.$$
$$\left.\left[\left(\rho_1K\left(|\bar{\psi}_j^{(k+1)}|-\theta\right)-\sqrt{(\rho_1K)^2\left(|\bar{\psi}_j^{(k+1)}|-\theta\right)^2-4\rho_1K\left(\lambda-\rho_1K|\bar{\psi}_j^{(k+1)}|\theta\right)}\right)\Big/(2\rho_1K)\right]_+\right\};$$

otherwise, $\mathcal{C}=\{0\}$.

2. SCAD: Consider $\theta>2$ and let

$$\begin{cases} x_1=\operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right)\min\left(\lambda,\max(0,|\bar{\psi}_j^{(k+1)}|-\lambda/(\rho_1K))\right)\ \text{ s.t. }\ |z_j^{(k)}|\leq\lambda,\\ x_2=\operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right)\min\left(\theta\lambda,\max\left(\lambda,\dfrac{\rho_1K|\bar{\psi}_j^{(k+1)}|(\theta-1)-\theta\lambda}{\rho_1K(\theta-2)}\right)\right)\ \text{ s.t. }\ \lambda<|z_j^{(k)}|\leq\theta\lambda,\\ x_3=\operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right)\max\left(\theta\lambda,|\bar{\psi}_j^{(k+1)}|\right)\ \text{ s.t. }\ |z_j^{(k)}|>\theta\lambda.\end{cases}$$

Thus, we have $z_j^{(k+1)}=\arg\min_m h_j(m)$ s.t. $m\in\{x_1,x_2,x_3\}$, where $h_j(m)=\frac{1}{2}\left(m-\bar{\psi}_j^{(k+1)}\right)^2+\frac{1}{\rho_1K}p_\lambda(m)$ and $p_\lambda(m)$ refers to the SCAD penalty in Table 1.

3. MCP: Let $x_1=\operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right)m$ and $x_2=\operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right)\max\left(\theta\lambda,|\bar{\psi}_j^{(k+1)}|\right)$, where $m=\arg\min_{z_j\in\mathcal{C}}\frac{1}{2}\left(z_j-|\bar{\psi}_j^{(k+1)}|\right)^2+\frac{\lambda}{\rho_1K}z_j-\frac{z_j^2}{2\theta}$. Here, $\mathcal{C}=\left\{0,\theta\lambda,\min\left(\theta\lambda,\max\left(0,\frac{\theta(\rho_1K|\bar{\psi}_j^{(k+1)}|-\lambda)}{\rho_1K(\theta-1)}\right)\right)\right\}$ if $\theta-1\neq 0$, and $\mathcal{C}=\{0,\theta\lambda\}$ otherwise. Then if $h_j(x_1)\leq h_j(x_2)$, we have $z_j^{(k+1)}=x_1$; otherwise, $z_j^{(k+1)}=x_2$. Here, $h_j(m)=\frac{1}{2}\left(m-\bar{\psi}_j^{(k+1)}\right)^2+\frac{1}{\rho_1K}p_\lambda(m)$ and $p_\lambda(m)$ refers to the MCP regularizer in Table 1.

4. Capped-ℓ_1: Let

$$\begin{cases} x_1 = \operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right) \max\left(\theta, |\bar{\psi}_j^{(k+1)}|\right) \ \text{ s.t. } \ |z_j^{(k)}| \geq \theta, \\ x_2 = \operatorname{sign}\left(\bar{\psi}_j^{(k+1)}\right) \min\left(\theta, \max\left(0, |\bar{\psi}_j^{(k+1)}| - \lambda/(\rho_1 K)\right)\right) \ \text{ s.t. } \ |z_j^{(k)}| \leq \theta. \end{cases}$$

Then if $h_j(x_1) \leq h_j(x_2)$, we have $z_j^{(k+1)} = x_1$; otherwise, $z_j^{(k+1)} = x_2$. Here, $h_j(m) = \frac{1}{2}\left(m - \bar{\psi}_j^{(k+1)}\right)^2 + \frac{1}{\rho_1 K} p_\lambda(m)$ and $p_\lambda(m)$ indicates the capped-ℓ_1 regularizer in Table 1.

Appendix B: Proof of Lemma 1

Proof Note that $\mathcal{L}\left(\boldsymbol{w}_i, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right)$ is strongly convex with $\rho_1 + \rho_1\sigma_{\min}(\boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i)$ with respect to $\boldsymbol{w}_i$. Thus, the minimization of $\mathcal{L}\left(\boldsymbol{w}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right)$ directly yields

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{w}_i^{(k)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq & \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \\ & + \frac{\rho_1 + \rho_2\sigma_{\min}(\boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i)}{2}\left\|\boldsymbol{w}_i^{(k+1)} - \boldsymbol{w}_i^{(k)}\right\|^2. \end{aligned} \tag{B1}$$

Similarly, we can obtain the following inequalities:

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq & \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \\ & + \frac{\rho_2\|\boldsymbol{y}_i\|^2}{2}\left\|b_i^{(k+1)} - b_i^{(k)}\right\|^2, \end{aligned} \tag{B2}$$

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq & \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \\ & + \frac{\rho_2}{2}\left\|\boldsymbol{\xi}_i^{(k+1)} - \boldsymbol{\xi}_i^{(k)}\right\|^2, \end{aligned} \tag{B3}$$

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq & \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k+1)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \\ & + \frac{\rho_2}{2}\left\|\boldsymbol{s}_i^{(k+1)} - \boldsymbol{s}_i^{(k)}\right\|^2. \end{aligned} \tag{B4}$$

Note that $\boldsymbol{z}^{(k+1)}$ is the minimizer of

$$\mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) + \frac{\beta\|\boldsymbol{z} - \boldsymbol{z}^{(k)}\|^2}{2}$$

with respect to $\boldsymbol{z}$, which means

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq & \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k+1)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \\ & + \frac{\beta\left\|\boldsymbol{z}^{(k+1)} - \boldsymbol{z}^{(k)}\right\|^2}{2}. \end{aligned} \tag{B5}$$

Summing inequalities (B1)–(B5) yields

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{w}_i^{(k)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq & \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k+1)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \\ & + \frac{\nu}{2}\left\|\boldsymbol{D}_i^{(k+1)} - \boldsymbol{D}_i^{(k)}\right\|^2, \end{aligned} \tag{B6}$$

where

$$\nu := \min\left\{\rho_1 + \rho_2\sigma_{\min}(\boldsymbol{H}_i^{\mathrm{T}}\boldsymbol{H}_i), \rho_2, \rho_2\|\boldsymbol{y}_i\|^2, \beta\right\}.$$

Appendix C: Proof of Lemma 2

Proof The optimization condition for updating $\boldsymbol{w}_i^{(k+1)}$ gives

$$\nabla|_{\boldsymbol{w}=\boldsymbol{w}_i^{(k+1)}}\mathcal{L}\left(\boldsymbol{w},b_i^{(k)},\boldsymbol{z}^{(k)},\boldsymbol{\xi}_i^{(k)},\boldsymbol{s}_i^{(k)},\boldsymbol{u}_i^{(k)},\boldsymbol{v}_i^{(k)}\right)=\boldsymbol{0}_d. \tag{C1}$$

That is also

$$\rho_1\left(\boldsymbol{w}_i^{(k+1)}-\boldsymbol{z}^{(k)}+\boldsymbol{u}_i^{(k)}\right)+\rho_1\boldsymbol{H}_i^{\mathrm{T}}\left(\boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)}+b_i^{(k)}\boldsymbol{y}_i+\boldsymbol{\xi}_i^{(k)}-\boldsymbol{s}_i^{(k)}-\boldsymbol{1}_{n_i}+\boldsymbol{v}_i^{(k)}\right)=\boldsymbol{0}_d. \tag{C2}$$

On the other hand, the optimization condition for updating $b^{(k+1)}$ gives

$$\nabla|_{b=b_i^{(k+1)}}\mathcal{L}\left(\boldsymbol{w}_i^{(k+1)},b,\boldsymbol{z}_i^{(k)},\boldsymbol{\xi}_i^{(k)},\boldsymbol{s}_i^{(k)},\boldsymbol{u}_i^{(k)},\boldsymbol{v}_i^{(k)}\right)=0. \tag{C3}$$

This can be represented as

$$\boldsymbol{y}_i^{\mathrm{T}}\left(\boldsymbol{H}_i\boldsymbol{w}_i^{(k+1)}+b_i^{(k+1)}\boldsymbol{y}_i+\boldsymbol{\xi}_i^{(k)}-\boldsymbol{s}_i^{(k)}-\boldsymbol{1}_{n_i}+\boldsymbol{v}_i^{(k)}\right)=0. \tag{C4}$$

In Eq. (C4), letting $k=k+1$, we have

$$\boldsymbol{y}_i^{\mathrm{T}}\left(\boldsymbol{H}_i\boldsymbol{w}_i^{(k+2)}+b_i^{(k+2)}\boldsymbol{y}_i+\boldsymbol{\xi}_i^{(k+1)}-\boldsymbol{s}_i^{(k+1)}-\boldsymbol{1}_{n_i}+\boldsymbol{v}_i^{(k+1)}\right)=0. \tag{C5}$$

Subtraction of Eqs. (C4) and (C5) gives

$$\begin{aligned}\left\|\boldsymbol{y}_i^{\mathrm{T}}(\boldsymbol{v}_i^{(k+1)}-\boldsymbol{v}_i^{(k)})\right\|\leq&\|\boldsymbol{H}_i\|\|\boldsymbol{y}_i\|\left\|\boldsymbol{w}_i^{(k+2)}-\boldsymbol{w}_i^{(k+1)}\right\|+\|\boldsymbol{y}_i\|\left\|b_i^{(k+2)}-b_i^{(k+1)}\right\|+\|\boldsymbol{y}_i\|\left\|\boldsymbol{\xi}_i^{(k+1)}-\boldsymbol{\xi}_i^{(k)}\right\|\\&+\|\boldsymbol{y}_i\|\left\|\boldsymbol{s}_i^{(k+1)}-\boldsymbol{s}_i^{(k)}\right\|.\end{aligned} \tag{C6}$$

With Assumption 1, we obtain

$$\left\|\boldsymbol{v}_i^{(k+1)}-\boldsymbol{v}_i^{(k)}\right\|^2\leq c_1\left\|\boldsymbol{D}_i^{(k+1)}-\boldsymbol{D}_i^{(k)}\right\|^2+c_2\left\|\boldsymbol{D}_i^{(k+2)}-\boldsymbol{D}_i^{(k+1)}\right\|^2, \tag{C7}$$

where $c_1,c_2>0$ are polynomial compositions of $\|\boldsymbol{H}_i\|$ and $\|\boldsymbol{y}_i\|$, respectively. In Eq. (C2), letting $k=k+1$, we have

$$\rho_1\left(\boldsymbol{w}_i^{(k+2)}-\boldsymbol{z}^{(k+1)}+\boldsymbol{u}_i^{(k+1)}\right)+\rho_1\boldsymbol{H}^{\mathrm{T}}\left(\boldsymbol{H}_i\boldsymbol{w}_i^{(k+2)}+b_i^{(k+1)}\boldsymbol{y}_i+\boldsymbol{\xi}_i^{(k+1)}-\boldsymbol{s}_i^{(k+1)}-\boldsymbol{1}_{n_i}+\boldsymbol{v}_i^{(k+1)}\right)=\boldsymbol{0}_d. \tag{C8}$$

Similarly, subtraction of Eqs. (C2) and (C8) tells us

$$\left\|\boldsymbol{u}_i^{(k+1)}-\boldsymbol{u}_i^{(k)}\right\|^2\leq\hat{c}_3\left\|\boldsymbol{D}_i^{(k+1)}-\boldsymbol{D}_i^{(k)}\right\|^2+\hat{c}_4\left\|\boldsymbol{v}_i^{(k+1)}-\boldsymbol{v}_i^{(k)}\right\|^2, \tag{C9}$$

where $\hat{c}_3,\hat{c}_4>0$ are polynomial compositions of ρ_1, $\|\boldsymbol{H}\|$, and $\|\boldsymbol{y}_i\|$, $\hat{c}_3=O(1/\rho_1^2)$, and $\hat{c}_4=O(1/\rho_1^2)$. Substituting inequality (C9) into inequality (30), we then obtain

$$\left\|\boldsymbol{u}_i^{(k+1)}-\boldsymbol{u}_i^{(k)}\right\|^2\leq c_3\left\|\boldsymbol{D}_i^{(k+1)}-\boldsymbol{D}_i^{(k)}\right\|^2+c_4\left\|\boldsymbol{D}_i^{(k+2)}-\boldsymbol{D}_i^{(k+1)}\right\|^2, \tag{C10}$$

where $c_3=O\left(1/\rho_1^2\right)$ and $c_4=O\left(1/\rho_1^2\right)$.

Appendix D: Proof of Theorem 1

Proof With direct calculations, we can derive

$$\begin{aligned}&\mathcal{L}\left(\boldsymbol{w}_i^{(k)},b_i^{(k)},\boldsymbol{z}^{(k)},\boldsymbol{\xi}_i^{(k)},\boldsymbol{s}_i^{(k)},\boldsymbol{u}_i^{(k)},\boldsymbol{v}_i^{(k)}\right)\geq\mathcal{L}\left(\boldsymbol{w}_i^{(k+1)},b_i^{(k+1)},\boldsymbol{z}^{(k+1)},\boldsymbol{\xi}_i^{(k+1)},\boldsymbol{s}_i^{(k+1)},\boldsymbol{u}_i^{(k+1)},\boldsymbol{v}_i^{(k+1)}\right)\\&+\frac{\nu}{2}\left\|\boldsymbol{D}_i^{(k+1)}-\boldsymbol{D}_i^{(k)}\right\|^2-\rho_1\left\|\boldsymbol{u}_i^{(k+1)}-\boldsymbol{u}_i^{(k)}\right\|^2-\rho_2\left\|\boldsymbol{v}_i^{(k+1)}-\boldsymbol{v}_i^{(k)}\right\|^2.\end{aligned} \tag{D1}$$

Substituting inequalities (30) and (31) into inequality (D1), we have

$$\mathcal{L}\left(\boldsymbol{w}_i^{(k)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) \geq \mathcal{L}\left(\boldsymbol{w}_i^{(k+1)}, b_i^{(k+1)}, \boldsymbol{z}^{(k+1)}, \boldsymbol{\xi}_i^{(k+1)}, \boldsymbol{s}_i^{(k+1)}, \boldsymbol{u}_i^{(k+1)}, \boldsymbol{v}_i^{(k+1)}\right)$$
$$+\left(\frac{\nu}{2}-\rho_1 c_3-\rho_2 c_1\right)\left\|\boldsymbol{D}^{(k+1)}-\boldsymbol{D}^{(k)}\right\|^2-\left(\rho_2 c_2+\rho_1 c_4\right)\left\|\boldsymbol{D}^{(k+2)}-\boldsymbol{D}^{(k+1)}\right\|^2. \tag{D2}$$

Denote $a_k := \mathcal{L}\left(\boldsymbol{w}_i^{(k)}, b_i^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{\xi}_i^{(k)}, \boldsymbol{s}_i^{(k)}, \boldsymbol{u}_i^{(k)}, \boldsymbol{v}_i^{(k)}\right) + \left(\frac{\nu}{2}-\rho_1 c_3-\rho_2 c_1\right)\left\|\boldsymbol{D}_i^{(k+1)}-\boldsymbol{D}_i^{(k)}\right\|^2$. Then we can see that inequality (D2) actually indicates

$$\left[\frac{\nu}{2}-\left(\rho_1 c_3+\rho_2 c_1+\rho_2 c_2+\rho_1 c_4\right)\right]\left\|\boldsymbol{D}^{(k+2)}-\boldsymbol{D}^{(k+1)}\right\|^2 \leq a_k-a_{k+1}. \tag{D3}$$

With Assumption 2, $\inf_k\{a_k\} > -\infty$, and then $\sum_k(a_k-a_{k+1}) < +\infty$. Thus, we obtain

$$\sum_k\left[\frac{\nu}{2}-\left(\rho_1 c_3+\rho_2 c_1+\rho_2 c_2+\rho_1 c_4\right)\right]\left\|\boldsymbol{D}_i^{(k+2)}-\boldsymbol{D}_i^{(k+1)}\right\|^2 < +\infty. \tag{D4}$$

This means

$$\lim_k\left\|\boldsymbol{D}_i^{(k+1)}-\boldsymbol{D}_i^{(k)}\right\| = 0. \tag{D5}$$