JUSTC
Article
http://justc.ustc.edu.cn
Received: September 17, 2022; Accepted: November 15, 2022

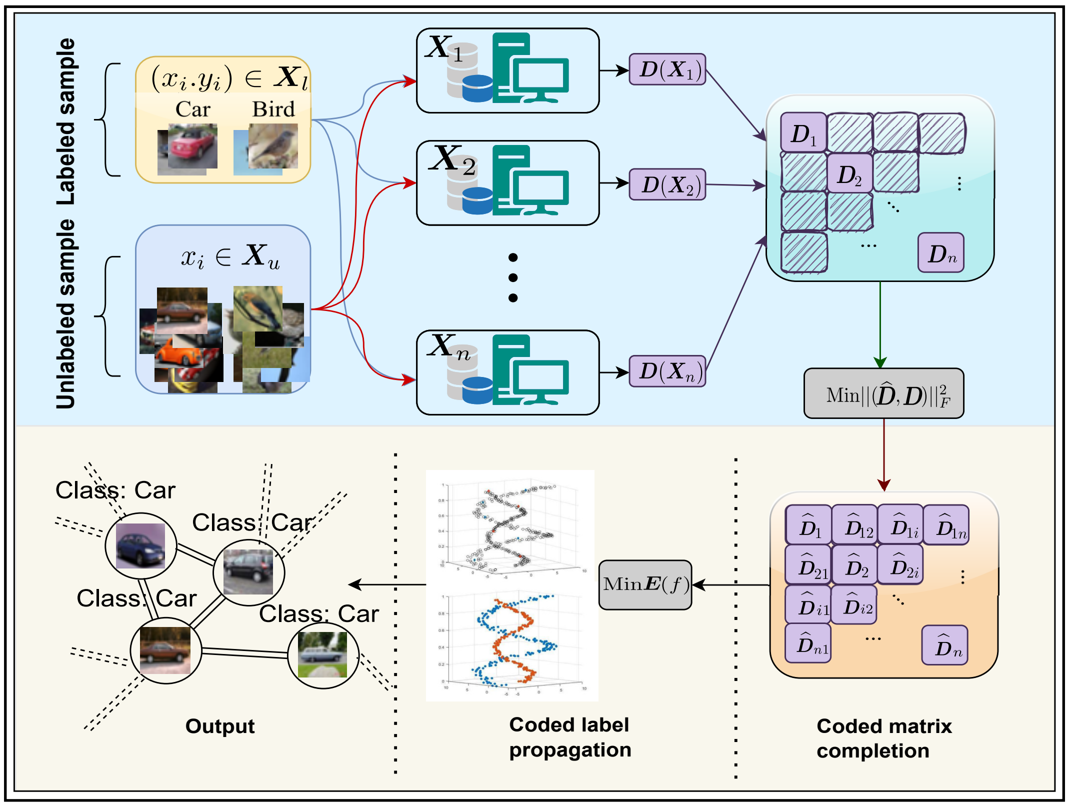
Coded computing for distributed graph-based semi-supervised learning

Siqi Tan, Li Chen ✉, and Weidong Wang

School of Information Science and Technology, University of Science of Technology of China, Hefei 230027, China

✉Correspondence: Li Chen, E-mail: chenli87@ustc.edu.cn

Graphical abstract


Labeled sample
Unlabeled sample
$(x_i.y_i) \in X_l$
Car
Bird
$x_i \in X_u$
X_1
X_2
X_n
$D(X_1)$
$D(X_2)$
$D(X_n)$
D_1
D_2
D_n
$\mathrm{Min}||(\widehat{D},D)||_F^2$
$\widehat{D}_1$ $\widehat{D}_{12}$ $\widehat{D}_{1i}$ $\widehat{D}_{1n}$
$\widehat{D}_{21}$ $\widehat{D}_2$ $\widehat{D}_{2i}$
$\widehat{D}_{i1}$ $\widehat{D}_{i2}$
$\widehat{D}_{n1}$ $\widehat{D}_n$
$\mathrm{Min}E(f)$
Class: Car
Class: Car
Class: Car
Class: Car
Output
Coded label propagation
Coded matrix completion

The proposed algorithm consists of two parts: coded matrix completion and coded label propagation.

Public summary

■ Existing distributed graph-based semi-supervised learning algorithms suffer from extensive communication and computation. The proposed scheme is more efficient with low complexity by providing a parallel and distributed solution for global Euclidean distance matrix completion.

■ The iteration time of distributed graph-based semi-supervised learning is defined by the slowest node, noted as the straggler node. A novel coded computation design for distributed graph-based semi-supervised learning is proposed to decrease the straggler effect.

■ We numerically verified the superiority of the proposed scheme on Alibaba cloud elastic compute service. In general, the simulation results have demonstrate that our proposed algorithm is efficient and straggler tolerant.

Article

http://justc.ustc.edu.cn

Received: September 17, 2022; Accepted: November 15, 2022

Coded computing for distributed graph-based semi-supervised learning

Siqi Tan, Li Chen ✉, and Weidong Wang

School of Information Science and Technology, University of Science of Technology of China, Hefei 230027, China

✉Correspondence: Li Chen, E-mail: chenli87@ustc.edu.cn

Cite This: *JUSTC*, **2023**, 53(4): 0401 (11pp)   Read Online

Abstract: Semi-supervised learning (SSL) has been applied to many practical applications over the past few years. Recently, distributed graph-based semi-supervised learning (DGSSL) has been shown to have good performance. Current DGSSL algorithms usually have the problems of inefficient graph construction and the straggler effect. This paper proposes a novel coded DGSSL (CDGSSL) to solve these problems. We first provide a novel parallel and distributed solution of matrix completion for efficient graph construction. Then, we develop the CDGSSL algorithm based on coding theory. Specifically, the proposed algorithm consists of two parts separately designed based on the maximum distance separable (MDS) code. In general, the proposed coded distributed algorithm is efficient and straggler tolerant. Moreover, we provide an optimal parameter design for the proposed algorithm. The results of the experiments on the Alibaba Cloud elastic compute service (ECS) demonstrate the superiority of the proposed algorithm.

Keywords: coded computation; distributed learning; matrix completion; maximum distance separable code; semi-supervised learning

CLC number: TP181   **Document code:** A

1 Introduction

Semi-supervised learning (SSL) has been of significant interest in machine learning and artificial intelligence over the past few years[1]. Compared with supervised learning (SL) and unsupervised learning (UL), SSL requires less human effort but gives higher accuracy[2]. SSL utilizes the information of both labeled and unlabeled data to obtain better performance than SL and UL. Thus, how discovering the input data distribution by exploiting unlabeled data is crucial in SSL[3].

The initial SSL algorithm was self-training in Ref. [4], which used the data set with all attributes as input data. Co-training was another SSL algorithm in Ref. [5], which assumed that the input data set consists of two subsets with different views. In general, three assumptions on the input data space are commonly used in SSL, smoothness, cluster, and manifold. Using these assumptions, various effective algorithms have been proposed[6–8]. Among these methods, graph-based SSL (GSSL) has attracted growing attention because it usually involves optimizing a convex objective and is both easily scalable and parallelizable. In GSSL, the original data structure was represented by a graph whose edges were measured by the similarity between data samples[9]. Then, the labeled information can be propagated to the unlabeled samples through the graph, which is noted as label propagation. Most GSSL methods address label propagation by solving a convex quadratic optimization problem for feature vectors. Some notable GSSL methods proposed in the literature include Gaussian field and harmonic function (GFHF)[10], local and global consistency[11], regularized Laplacian[12], and random walks[13].

The previously mentioned schemes are then applied to perform machine learning tasks, such as clustering, classification, and regression. Semi-supervised clustering aims to obtain better clusters than unsupervised clustering[14]. Semi-supervised classification (SSC) and semi-supervised regression (SSR) have been shown to outperform supervised classification and regression[15,16]. Although these algorithms can be applied to solve many problems in practice, most of them have focused on centralized learning. However, distributed protocols for SSL are urgently needed for multiple nodes, such as medical diagnostics[17], distributed music classification[18], and distributed multimedia classification[19].

To effectively utilize data separately stored over communication networks, several distributed SSL (DSSL) approaches have been designed. Shen et al.[20] proposed two distributed semi-supervised metric learning frameworks using diffusion cooperation and alternating direction multipliers (ADMM) strategies. Scardapane et al.[21] proposed a DSSL algorithm based on the in-network successive convex approximation (NEXT) framework. Due to the scalability and parallelizability of GSSL, some works have proposed distributed GSSL (DGSSL). DGSSL focuses on constructing an efficient graph and preserving data privacy in multiple agent cases. Fierimonte et al.[22] proposed the D-LapRLS algorithm with the kernel method and distributed average consensus (DAC) strategy. In addition, they proposed a distributed matrix completion algorithm based on the framework of diffusion adaptation to calculate the global Euclidean distance matrix (EDM). Güler et al.[23] made use of the privacy shield protocol multiparty computation (MPC) and homomorphic encryption (HE) to propose a DGSSL algorithm.

However, most existing DGSSL designs usually suffer from the following two problems. First, constructing a graph with dispersedly stored data is a major problem for DGSSL. To solve this problem, existing DGSSL designs have provided several graph construction algorithms, mainly focusing on improving classification accuracy and protecting data privacy. These algorithms require extensive communication and computation, which calls for more efficient schemes with lower complexity. Second, most DGSSLs solve the optimization problem for feature vectors using iteration solutions. The iteration time is determined by the slowest node, noted as the straggler node. Thus, the algorithm execution time of the current state-of-art DGSSL schemes is limited by the straggler nodes. The corruption of any node may lead to the inability of iterations to occur, resulting in prolonged training time.

Motivated by the above observations, this paper proposes coded DGSSL (CDGSSL) to efficiently construct a graph and alleviate the effect of straggler nodes. In consideration of the classification performance, we adopt EDM completion for graph construction. Although EDM completion has been applied in the D-LapRLS algorithm, its necessary step requires computing the global EDM updates at each node and computing the new update with the computed mean. Such a solution has a high computational cost, growing as a polynomial with the global graph size. We provide a novel parallel and distributed approach for global EDM completion to reduce its computational cost. To address the straggler problem, we further develop a coded computation design for DGSSL by adopting the maximum distance separable (MDS) code. Note that although coded computation has been applied to matrix multiplication for solving the straggler problem[24–26], how to employ it in DGSSL is still unknown.

In the present work, we assume that a dataset is partitioned and arbitrarily distributed over different nodes. Every node can communicate with a central server. Our purpose is to use DGSSL for a binary classification problem without exchanging any data points. From the generalized formulation in Ref. [10], information is mostly encoded in a matrix of pairwise distances between data samples. In the initial phase of the algorithm, each node computes the local distance matrix with its data before sending it to the server. To complete the global distance matrix, we address the EDM completion problem using a gradient descent method similar to previous works[27–30]. Moreover, we also address the label propagation using the gradient descent method. Then, we provide a parallel and distributed solution to reduce the cost of computation per iteration. Each iteration requires the node to compute a partial gradient before sending the results back to the server. In addition, we further develop a coded DGSSL algorithm consisting of two-component algorithms, namely coded matrix completion and coded label propagation. Since the computation is mainly determined by matrix-vector or matrix-matrix multiplication, our algorithm works by encoding the computational matrix with MDS code. In this light, the server only requires partial results to decode. Furthermore, we offer the optimal parameter design of the proposed algorithm. The proposed scheme can be applied to travel mode identification in intelligent transportation systems (ITS), which can enable institutes to understand human behaviors and urban management better and plan[31]. The main contributions of this paper are summarized as follows.

(i) A novel distributed EDM completion algorithm for efficient graph construction. We adopt EDM completion to improve the classification performance of DGSSL. The solution in Ref. [22] has high computation costs because each node has to compute the global EDM update. We overcome this problem by providing a parallel and distributed solution for global EDM completion. Instead of computing the global EDM update, each node only needs to compute a partial gradient. Therefore, it reduces computational cost but with similar classification performance.

(ii) A coded computation design for DGSSL. According to the characteristic that coded computation declines the straggler effect, we provide a novel coded computation design for DGSSL in this work. To exploit the coded computation solutions, we adopt gradient descent methods so that the computation is concentrated on matrix-vector or matrix-matrix multiplication. The server only requires partial results to obtain the gradient by encoding the computational matrix with MDS code.

(iii) Optimal parameter design for the proposed algorithm. Coding parameter and matrix completion parameter are important for the algorithm execution time, classification performance, and straggler tolerance. We optimize coding parameters based on the expected overall runtime. We then provide the optimal matrix completion parameter design by introducing a novel measure to quantify how much accuracy is obtained for the matrix completion algorithm.

The rest of this paper is organized as follows. First, Section 2 introduces the system model for the DSSL and the theoretical tools upon which the proposed algorithm is based. In Section 3, we detail our proposed framework for CDGSSL. In particular, we introduce the matrix completion problem in Section 3.1, propose a DGSSL under a distributed computing framework in Section 3.2, and develop the CDGSSL algorithm in Section 3.3. In addition, we introduce the optimal parameter design in the algorithm in Section 4. After that, in Section 5, we give the numerical results of the proposed algorithm to illustrate its effectiveness. Finally, conclusions are drawn in Section 6.

In this paper, we use boldface lowercase letters to denote vectors, e.g., $\boldsymbol{a}$, while matrices are denoted by boldface uppercase letters, e.g., $\boldsymbol{A}$. We use $\mathbf{1}$ to denote vector $[1,1,\cdots,1]^{\mathrm{T}}$. All vectors are column vectors in this paper. The operator $\|\cdot\|_2$ is the standard L_2 norm on Euclidean space, and $\|\cdot\|_2$ stands for the Frobenius norm. $\mathbb{R}^{M\times N}$ is the set of real-valued $M\times N$ matrices. diag($\cdot$), sign($\cdot$), rank($\cdot$), and average($\cdot$) represent the vector formed by diagonal elements, the sign function, the rank operator, and the average value of matrix elements, respectively.

2 System model & preliminaries

2.1 System model

This paper considers a DGSSL scenario for a binary classification problem with N clients and a central server, where the clients can compute tasks and transmit information. The overall setup is shown in Fig. 1. For readability, we provide a list of notations used throughout the paper in Table 1. The client has a local data set $\boldsymbol{X}_i$ and a label vector $\boldsymbol{Y}_i$. $\boldsymbol{X}_i=[\boldsymbol{x}_{i1}^{\mathrm{T}};\ \boldsymbol{x}_{i2}^{\mathrm{T}};\ \cdots;\boldsymbol{x}_{iM_i}^{\mathrm{T}}]=\{\boldsymbol{S}_i,\boldsymbol{U}_i\}$, $\boldsymbol{X}_i\in\mathbb{R}^{M_i\times K}$, $i=1,2,\cdots,N$. $\boldsymbol{S}_i$ and $\boldsymbol{U}_i$ stand for labeled data and unlabeled data sets, respectively. $\boldsymbol{Y}_i=[y_{i1},\cdots,y_{iM_i}]^{\mathrm{T}}$, $\boldsymbol{Y}_i\in\mathbb{R}^{M_i\times 1}$. And $y_{ij}=\{+1,-1\}$ when $\mathbf{x}_{ij}\in \mathrm{S}_i$. Data set $\boldsymbol{X}$ is composed of a small amount of labeled data and a large amount of unlabeled data, that is, $\boldsymbol{X}=\{\boldsymbol{X}_l,\boldsymbol{X}_u\}$. $\boldsymbol{X}_l=\{\boldsymbol{x}_i\in\boldsymbol{S}|\boldsymbol{S}=\bigcup\boldsymbol{S}_j, j=1,\cdots,N\}_{i=1}^{l}$ is a labeled data set, $\boldsymbol{X}_u=\{\boldsymbol{x}_i\in\boldsymbol{U}\}_{i=1}^{u}$ is an unlabeled data set, and $l\ll u$, $l+u=\sum_{i=1}^{N}M_i$. The goal of this SSL task is to learn the predicted function to label the unlabeled data set $\boldsymbol{U}=[\boldsymbol{U}_1;\boldsymbol{U}_2;\cdots;\boldsymbol{U}_N]$ by exploiting the label dependency information for a binary classification problem.

2.2 Graph-based SSL

The proposed algorithm is based on GSSL algorithms, which are capable of processing large-scale data sets in practice and are accessible to parallelize[9]. GSSL setups can be divided into the following two steps.

Graph construction: Construct a graph with all the labeled and unlabeled data. Each data is a vertex of the graph, and the graph edges represent the similarity between samples.

Label propagation: Propagate label information to unlabeled data through graphs.

The centralized SSL works as follows. In GSSL algorithm, graph $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$ is based on the labeled data set $\{(\boldsymbol{x}_1,y_1),\cdots,(\boldsymbol{x}_l,y_l)\}$ and the unlabeled data set $\{(\boldsymbol{x}_{l+1},y_{l+1}),\cdots,(\boldsymbol{x}_{l+u},y_{l+u})\}$, where $\boldsymbol{V}$ is the node $\{\boldsymbol{x}_1,\cdots,\boldsymbol{x}_l,\cdots,\boldsymbol{x}_{l+u}\}$. Suppose there is a symmetric weight matrix $\boldsymbol{W}$ on the edge set $\boldsymbol{E}$ of the graph. A classic example of the weight matrix $\boldsymbol{W}$ is a Gaussian weight matrix with elements

$$(\boldsymbol{W})_{ij}=w_{ij}=\begin{cases}\exp\left(\dfrac{-\|\boldsymbol{x}_i-\boldsymbol{x}_j\|_2^2}{2\sigma^2}\right), i\neq j;\\ 0,\ \text{otherwise}.\end{cases} \tag{1}$$

For the binary classification problem, a real-valued function $f:\boldsymbol{V}\to\mathbb{R}$ is assumed on the graph, and the classification rule corresponding to the problem is $y_i=\operatorname{sign}(f(\boldsymbol{x}_i))$, $y_i\in\{-1,+1\}$. Constraint f as $f(\boldsymbol{x}_i)=y_i, i=1,\cdots,l$, on the labeled data. Intuitively, we expect unlabeled data near the labeled data in the graph to have similar labels. Define a quadratic energy function[10]

$$E(f)=\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}w_{ij}(f(\boldsymbol{x}_i)-f(\boldsymbol{x}_j))^2=f^{\mathrm{T}}\Delta f, \tag{2}$$

where $\boldsymbol{f}=(f_l^{\mathrm{T}}f_u^{\mathrm{T}})^{\mathrm{T}}$, $\boldsymbol{f}_l=(f(\boldsymbol{x}_1);\cdots;f(\boldsymbol{x}_l))$ and $\boldsymbol{f}_u=(f(\boldsymbol{x}_{l+1});\cdots;f(\boldsymbol{x}_{l+u}))$. Define a diagonal matrix $\boldsymbol{H}:=\operatorname{diag}(h_1,h_2,\cdots,h_{l+u})$, whose diagonal element $h_i=\sum_{j=1}^{l+u}w_{ij}$ is the sum of the ith row elements of the matrix $\boldsymbol{W}$. The combinatorial Laplace is given by $\Delta:=\boldsymbol{H}-\boldsymbol{W}$. The real-valued function can be expressed as $f=\arg\min_{f|f_l}E(f)$ when the smallest energy function is a harmonic function as it satisfies the $\Delta f=0$. Function harmony means that the values of f in unlabeled data are the average of the data near it as

$$f(\boldsymbol{x}_j)=\frac{1}{h_j}\sum_{i}^{m}w_{ij}f(\boldsymbol{x}_i). \tag{3}$$

This is consistent with the smoothness of the graph. Thus, label information can be propagated to unlabeled data through the graph by solving the problem

$$\min_{f_u}E(f). \tag{4}$$

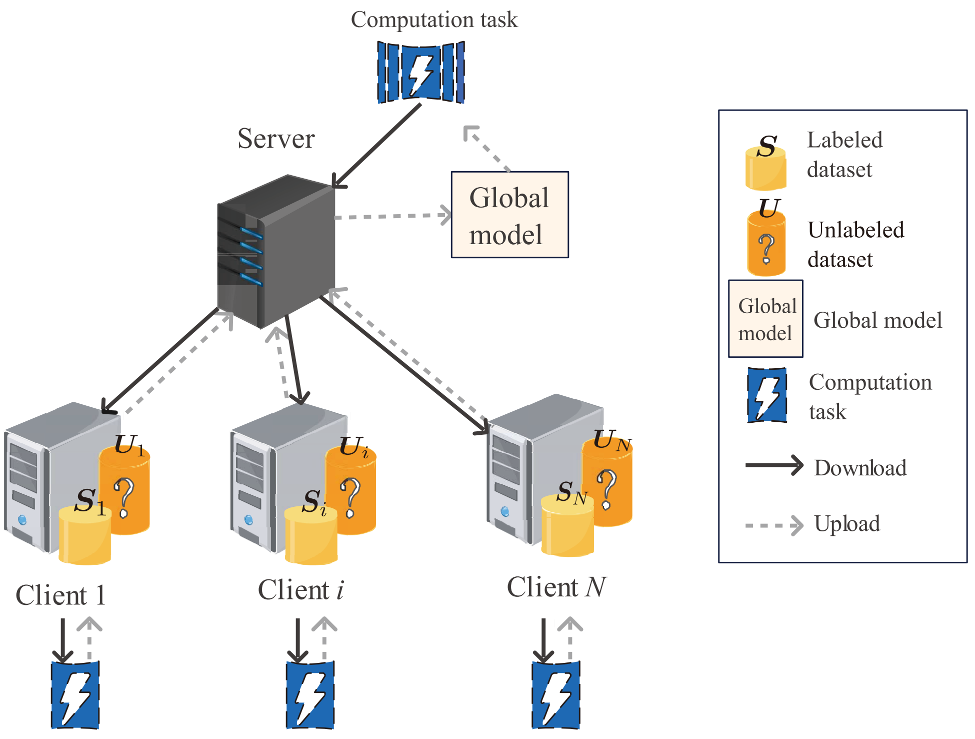


Fig. 1. Illustration of the distributed SSL setup with N client and 1 server. Each client owns a labeled data set $\boldsymbol{S}_i$ and an unlabeled data set $\boldsymbol{U}_i$. The server collects information from clients to run a distributed SSL task to estimate all the unknown labels and return the unknown label to clients.

Table 1. Notations used throughout this paper.

Data	$\boldsymbol{x}_i$
Data label	y_i
Local data set	$\boldsymbol{X}_i = \{\boldsymbol{x}_{ij}, j = 1, \cdots, M_i\}$
Local data label	$\boldsymbol{Y}_i = \{\boldsymbol{y}_{ij}, j = 1, \cdots, M_i\}$
Global data set	$\boldsymbol{X} = \boldsymbol{X}_1 \cup \cdots \cup \boldsymbol{X}_N$
Global data label	$\boldsymbol{Y} = \boldsymbol{Y}_1 \cup \cdots \cup \boldsymbol{Y}_N$
Subscript l	Labeled
Subscript u	Unlabeled
Unlabeled data set	$\boldsymbol{X}_u$
Labeled data set	$\mathbf{X}_l$
Local unlabeled data set	$\boldsymbol{U}_i = \boldsymbol{X}_u \cap \boldsymbol{X}_i$
Local labeled data set	$\boldsymbol{S}_i = \boldsymbol{X}_l \cap \boldsymbol{X}_i$
Weight matrix	$\boldsymbol{W}$
Classifier	$f : \boldsymbol{V} \to \mathbb{R}$
Euclidean distance matrix	$\boldsymbol{D}$
Diagonal matrix	$\boldsymbol{H}$
Combinatorial Laplacian	Δ

To explicitly calculate the results with matrix operations, the weight matrix $\boldsymbol{W}$ and a diagonal matrix $\boldsymbol{H}$ are divided into 4 blocks starting from the first l rows and l columns. Then (4) can be reformulated as

$$\begin{aligned}\min E(f) =& f^{\mathrm{T}}(\boldsymbol{H}-\boldsymbol{W})f = \\ & f_l^{\mathrm{T}}(\boldsymbol{H}_{ll}-\boldsymbol{W}_{ll})f_l + f_u^{\mathrm{T}}(\boldsymbol{H}_{uu}-\boldsymbol{W}_{uu})f_u - \\ & 2f_u^{\mathrm{T}}\boldsymbol{W}_{ul}f_l,\end{aligned} \tag{5}$$

where we have

$$\boldsymbol{W} = \begin{bmatrix}\boldsymbol{W}_{ll} & \boldsymbol{W}_{lu}\\ \boldsymbol{W}_{ul} & \boldsymbol{W}_{uu}\end{bmatrix}, \boldsymbol{H} = \begin{bmatrix}\boldsymbol{H}_{ll} & \boldsymbol{0}_{lu}\\ \boldsymbol{0}_{ul} & \boldsymbol{H}_{uu}\end{bmatrix}. \tag{6}$$

The explicit expression for the optimal solution to Eq. (5) is

$$f_u = (\boldsymbol{H}_{uu} - \boldsymbol{W}_{uu})^{-1}\boldsymbol{W}_{ul}f_l. \tag{7}$$

Considering the high complexity of the matrix inversion in (7), the centralized SSL algorithms are not suitable for large-scale data sets. Thus, we focus on a particular algorithm belonging to distributed computing framework to apply DGSSL to large-scale data sets in this paper.

3 The proposed coded DGSSL framework

3.1 Matrix completion

In the distributed SSL system described in Fig. 1, assume that the N clients only connect to the central server. Note that two items of the same data stored in different clients are treated as two different samples in this paper. The ith client has M_i data samples, and $\sum M_i := \sum_{i=1}^{N} M_i$ stands for the numbers of global data samples. Let $\boldsymbol{W}_i \in \mathbb{R}^{M_i \times M_i}$ be the local weight matrix computed by each client using its data. To complete a distributed SSL algorithm using the whole data set, the global weight matrix is in the form of

$$\boldsymbol{W} = \begin{bmatrix}\boldsymbol{W}_1 & ? & ?\\ ? & \ddots & ?\\ ? & ? & \boldsymbol{W}_N\end{bmatrix}. \tag{8}$$

As far as accuracy is concerned, the missing parts of the matrix $\boldsymbol{W}$ is a drawback to the distributed SSL algorithm. Since the diagonal matrix $\boldsymbol{H}$ can be calculated from the weight matrix, the distributed SSL problem in (7) is solved using the weight matrix in (8). In SSL, the graph information is concentrated in the weight matrix $\boldsymbol{W}$, which can be calculated using the Euclidean distance matrix $\boldsymbol{D}$[27]. Since the algorithm is classified based on the smoothness principle of the graph[9], the incomplete weight matrix in (8) may cause a problem, that is, the node only has smoothness on the graph composed of its client's data set. However, all nodes need to be smooth on the graph of the large data sets when joined together. As a result, the accuracy of the entire SSL algorithm is not enough to reach the level we expect. To address this problem mentioned above, we adopt a matrix completion algorithm for completing the weight matrix.

In this paper, instead of completing the weight matrix, we consider completing the global EDM and then using it to calculate the weight matrix. The Laplacian[12] and other kernel matrices for all kernel functions based on Euclidean distance can be obtained from the global EDM besides the Gaussian weight matrix.

The Euclidean distance matrix completion problem is formulated as follows. Consider a Euclidean distance matrix $\boldsymbol{D} \in \mathbb{R}^{\sum M_i \times \sum M_i}$, that is

$$(\boldsymbol{D})_{ij} = \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2, \tag{9}$$

where $\boldsymbol{D}$ is symmetrical with $\boldsymbol{D}_{ii} = 0$. The rank of the matrix $\boldsymbol{D}$ has an upper bound of $K+2$, which means $\boldsymbol{D}$ is of low rank when $K \ll \sum M_i$. The graph is used to represent the structure of the original data in the graph-based SSL algorithm. Moreover, an approximate recovery distance matrix can be a good representation of the internal structure of the original data. Thus, we consider the approximate distance matrix completion approach[28] when recovering the distance matrix from the partial distance matrix $\widehat{\boldsymbol{D}}$ by solving the optimization problem

$$\min_{\boldsymbol{D} \in \mathrm{EDM}(\sum M_i)} \|P_{\boldsymbol{\Omega}}(\widehat{\boldsymbol{D}} - \boldsymbol{D})\|_F^2, \tag{10}$$

where $\mathrm{EDM}(\sum M_i)$ represents all sets of Euclidean distance matrices of size $\sum M_i$, $\boldsymbol{\Omega}$ is a subset of indexes that indicates that there is a binding relationship for the pair of data. $P_{\boldsymbol{\Omega}}(\cdot)$ is an orthogonal projection operator, which means $[P_{\boldsymbol{\Omega}}(\boldsymbol{D}_{ij})]_{ij} = \boldsymbol{D}_{ij}$ when $(i, j) \in \boldsymbol{\Omega}$, and $[P_{\boldsymbol{\Omega}}(\boldsymbol{D}_{ij})]_{ij} = 0$, otherwise.

According to the Schoenberg map between the Euclidean distance matrix and semi-regularly deterministic matrix, the problem in (10) is reformulated as a semi-regular definite programming problem

$$\begin{aligned}&\min_{\boldsymbol{D}} \|P_{\Omega}(\widehat{\boldsymbol{D}} - \chi(\boldsymbol{D}))\|_F^2 \\ &\text{s.t. } \boldsymbol{D} \geqslant 0,\end{aligned} \tag{11}$$

where the linear operator $\chi(\boldsymbol{D}) = \mathrm{diag}(\boldsymbol{D})\mathbf{1}^{\mathrm{T}} + \mathbf{1}\,\mathrm{diag}(\boldsymbol{D})^{\mathrm{T}} - 2\boldsymbol{D}$.

A semi-positive definite matrix of rank r can be decomposed into $\{\boldsymbol{D} = \boldsymbol{V}\boldsymbol{V}^{\mathrm{T}}\}$. Thus the optimization problem becomes

$$\min_{\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}}\in S_+(r,\sum M_i)} \|P_\Omega\left(\widehat{\boldsymbol{D}}-\chi(\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}})\right)\|_F^2, \tag{12}$$

where we have

$$S_+\left(r,\sum M_i\right) = \{\boldsymbol{U}\in\mathbb{R}^{\Sigma M_i\times\Sigma M_i} : \boldsymbol{U} = \boldsymbol{U}^{\mathrm{T}} \succcurlyeq 0, \mathrm{rank}(\boldsymbol{U}) = r\}.$$

In matrix completion, solving the optimization problem has been studied for a single node. However, all algorithms that solve this problem for a single node have high computational overhead and therefore do not apply to large-scale data sets. Thus, we consider extending the optimization problem to a distributed computing system.

3.2 DGSSL under distributed computing framework

We propose a novel DGSSL algorithm under distributed computing framework, which consists of three main stages.

Local EDM computation: Each client computes $\boldsymbol{D}_i$ using its data set and sends it to the central server. The central server has a pre-distance matrix $\widehat{\boldsymbol{D}}$, that is

$$\widehat{\boldsymbol{D}} := \boldsymbol{D}_1\oplus\boldsymbol{D}_2\oplus\cdots\oplus\boldsymbol{D}_N = \oplus_{i=1}^N \boldsymbol{D}_i, \tag{13}$$

where $\oplus$ represents the straight sum of the matrix, $\boldsymbol{D}_i\in\mathbb{R}^{M_i\times M_i}$, $\widehat{\boldsymbol{D}}\in\mathbb{R}^{\Sigma M_i\times\Sigma M_i}$.

Matrix completion: Iteratively solve the EDM completion problem in (12). The iterative solution to (12) is given by

$$\nabla_V F(\boldsymbol{V}) = \chi^*\left\{P_\Omega(\chi(\boldsymbol{V}[n]\boldsymbol{V}^{\mathrm{T}}[n]) - \widehat{\boldsymbol{D}})\right\}\boldsymbol{V}[n], \tag{14}$$

$$\boldsymbol{V}[n+1] = \boldsymbol{V}[n] - \beta\nabla_V F(\boldsymbol{V}), \tag{15}$$

where $\chi^*(\boldsymbol{A}) = 2\,[\mathrm{diag}(\boldsymbol{A})\boldsymbol{1}^{\mathrm{T}} - \boldsymbol{A}]$.

Label propagation: Iteratively solve the optimal problem for f_u. The iterative solution to (5) is in the form of

$$\begin{aligned} f_u^{t+1} &= f_u^t - \alpha\nabla E(f) = \\ &f_u^t - 2\alpha(\boldsymbol{H}_{uu} - \boldsymbol{W}_{uu})f_u^t - 2\alpha\boldsymbol{W}_{ul}f_l. \end{aligned} \tag{16}$$

Concerning the distributed system, the algorithm can be completed with storage nodes executing computation tasks assigned by the server. The amount of computation for each iteration in (15) and (16) is mainly on matrix-vector or matrix-matrix multiplication. Thus, the partial sum can be calculated on different nodes, and then the results are obtained by adding all the partial sums on the central node. In this way, the proposed algorithm is based on the distributed computing framework. The distributed computation setting can be seen in Fig. 2.

Nevertheless, the system may be significantly affected by the noise of straggler nodes and system failures when multiplying the vectors or matrix of the distributed computing matrix. In each iteration, the central node has to wait for all nodes to return their results before it can recover the final result. Thus, the straggler node in the system determines the time of the algorithm. In addition, if a node fails in the system, it may lead to the failure of the entire algorithm. Given the above situations, we further develop a coded distributed SSL by adopting coding theory to solve the above problems.

3.3 Coded DGSSL

We propose a coded distributed SSL algorithm based on the MDS code to make it robust to stragglers or system failure. In a distributed setting, each client can only calculate its distance matrix based on local training data, and the distance information between data at different clients is unknown. An approximate distance matrix is obtained by performing distributed matrix completion of the diagonal distance matrix of blocks received and restored by the central server. Therefore, the proposed algorithm consists of two-component algorithms.

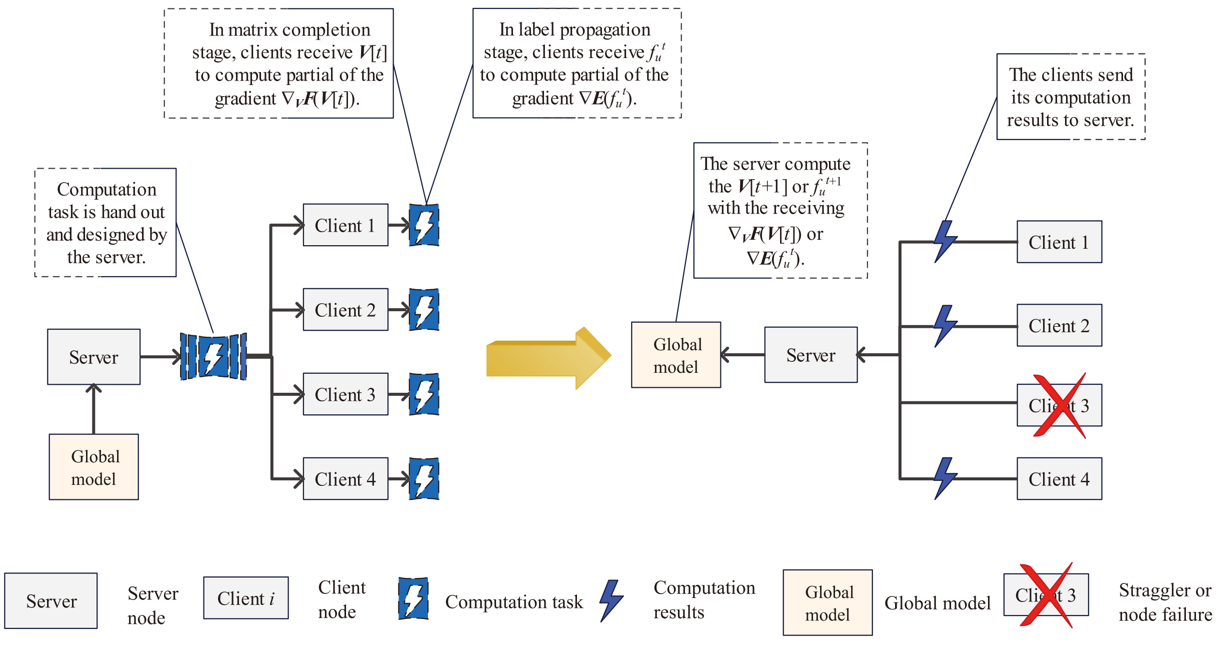

In matrix completion stage, clients receive $V[t]$ to compute partial of the gradient $\nabla_V F(V[t])$.
In label propagation stage, clients receive f_u^t to compute partial of the gradient $\nabla E(f_u^t)$.
Computation task is hand out and designed by the server.
Server
Global model
Client 1
Client 2
Client 3
Client 4
The server compute the $V[t+1]$ or f_u^{t+1} with the receiving $\nabla_V F(V[t])$ or $\nabla E(f_u^t)$.
The clients send its computation results to server.
Global model
Server
Client 1
Client 2
Client 3
Client 4
Server
Server node
Client i
Client node
Computation task
Computation results
Global model
Global model
Client 3
Straggler or node failure

Fig. 2. Illustration of distributed computation for distributed SSL.

3.3.1 Coded matrix completion

Calculate the distance matrix $\boldsymbol{D}$ using a coded distributed matrix completion algorithm. The optimization problem is to extend (12) into a distributed computing system and ensure fault tolerance for stragglers. We observe that the amount of computation of the iterative process is mainly for the matrix-matrix and matrix-vector multiplication in (14) and (15). Thus, we employ coding techniques to make the matrix-matrix and matrix-vector multiplication operations fault-tolerant to system noise in distributed computing. The problem in (14) can be reformulated as

$$\nabla_V F(\boldsymbol{V}) = \boldsymbol{W}\boldsymbol{V}[n] - \boldsymbol{T}\boldsymbol{V}[n], \tag{17}$$

where we have

$$\boldsymbol{T} := \chi^*(P_\Omega(\widehat{\boldsymbol{D}})), \boldsymbol{W} := \chi^*\{P_\Omega(\chi(\boldsymbol{V}[n]\boldsymbol{V}^{\mathrm{T}}[n]))\}. \tag{18}$$

Here we use (N,k_1), (N,k_2) MDS code, whose encoding matrices are $\boldsymbol{S}_1 \in \mathbb{R}^{\sum_{i=1}^N l_i \times \sum M_i}$, $\boldsymbol{S}_2 \in \mathbb{R}^{r \times \sum_{i=1}^N s_i}$, respectively. The ith node receives the encoding matrix block $\boldsymbol{T}_i \in \mathbb{R}^{l_i \times \sum M_i}$, $\boldsymbol{S}_{2i} \in \mathbb{R}^{r \times s_i}$ sent by the central node, where $\boldsymbol{T}_i$ is part of the row in the code matrix $\widehat{\boldsymbol{T}} = \boldsymbol{S}_1\boldsymbol{T}$ and $\boldsymbol{S}_{2i}$ is partial columns in $\boldsymbol{S}_2$. At the beginning of each iteration, the central node multicasts $\boldsymbol{V}[n]$. Each node then calculates the matrix-matrix multiplication and sends the results to the central node. The central node starts decoding the results and prepares for the next iteration after receiving $\max\{k_1,k_2\}$ results.

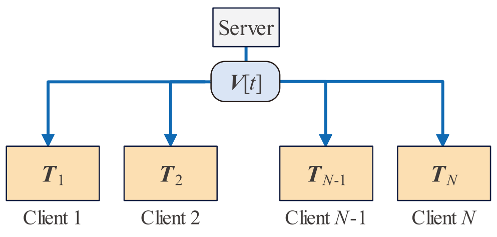
A single iteration of coded distributed matrix completion is shown in Fig. 3. The pseudocode of coded EDM completion is summarized in Algorithm 1, where the maximum number of iterations is denoted as L.

Algorithm 1: Coded matrix completion

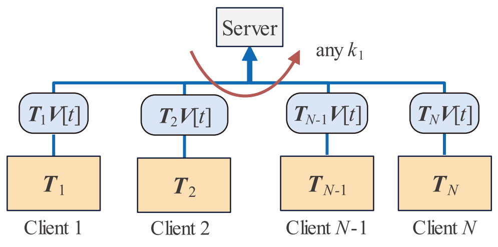
```
Input: Incomplete global EDM matrix $\widehat{D}$, coding matrix $S_1, S_2$
Output: Optimal EDM matrix $D$
1 Compute $T := \chi^*(P_\Omega(\widehat{D})), \widehat{T} = S_1 T$;
2 Send $T_i, S_{2i}$ to the $i$th client;
3 for $t = 1 : L$ do
4     Multicast $V[t-1]$ to all the clients;
5     for client $i \in [N]$ do
6         Compute $G[t-1]$ via (18);
7         Compute $P_i = T_i V[t-1], Q_i = G[t-1] S_{2i}$;
8         Send $P_i, Q_i$ to the server;
9         $P_{\text{list}} = [], Q_{\text{list}} = [], C_{\text{rec}} = []$;
10         while: rank$(C_{\text{rec}}) < \max\{k_1, k_2\}$ do
11             on Receiving message $P_j, Q_j$ from client $j$;
12             $C_{\text{rec}} \leftarrow [C_{\text{rec}}, C_j], P_{\text{list}} \leftarrow [P_{\text{list}}, P_j], Q_{\text{list}} = [Q_{\text{list}}, Q_j]$;
13             dec$(C_{\text{rec}}, P_{\text{list}})$, dec$(C_{\text{rec}}, Q_{\text{list}})$;
14     Compute $\nabla_V F(V[t-1])$ via (14);
15     Compute $\mathbf{V}[t]$ via (15);
16 Return $D = \chi\left(V[L]V[L]^{\mathrm{T}}\right)$
```

3.3.2 Coded label propagation

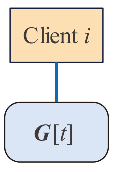
Optimize the distribution strategy for the label propagation algorithm and make it fault-tolerant to straggler nodes. We adopt coding technology to increase the fault tolerance of the algorithm to straggler as well. Define $\boldsymbol{A} := 2(\boldsymbol{H}_{uu} - \boldsymbol{W}_{uu})$. We adopt (N,k_3) MDS code. For $\boldsymbol{A}$, its encoding matrix is $\boldsymbol{S}_3 \in \mathbb{R}^{\sum_{i=1}^N q_i \times u}$, where $\sum_{i=1}^N q_i = u$. The ith node receives the encoding matrix block $\boldsymbol{A}_i \in \mathbb{R}^{q_i \times u}$ sent by the central node. $\boldsymbol{A}_i$ is part of the rows in the code matrix $\widehat{\boldsymbol{A}} = \boldsymbol{S}_3\boldsymbol{A}$. At the beginning of each iteration, the central node multicasts $\boldsymbol{f}_u^{t-1}$. Each



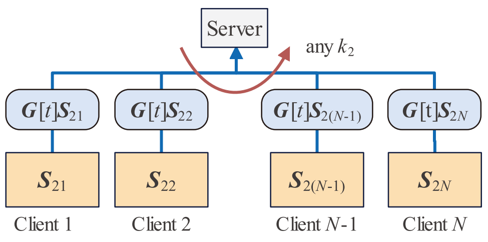
(a) In the beginning of the $(t+1)$th iteration, the server multicast $V[t]$ to cilents.



(b) Server waits for the earliest responding k_1 clients results.



(c) Clients compute $\boldsymbol{G}[t]$ by the receiving $V[t]$.



(d) Server waits for the earliest responding k_2 clients results.

Fig. 3. Single iteration of coded distributed matrix completion.

node then calculates the matrix-vector multiplication and sends the results to the central node. The central node starts decoding the results and prepares for the next iteration after receiving any k_3 task results.

A single iteration of coded label propagation is shown in Fig. 4. The pseudocode of the entire coded label propagation algorithm is summarized in Algorithm 2, where the largest number of iterations in the label propagation is denoted as T.

Algorithm 2: Coded label propagation

```
Input: Distance matrix $\widehat{D}$, label $Y = Y_1 \cup \cdots \cup Y_N$
Output: Optimal unlabeled label $Y_u$
1 Compute $W$ via (1) using $\widehat{D}$;
2 Compute $A = 2(H_{uu} - W_{uu})$, $\widehat{A} = S_3 A$;
3 Send $A_i$ to the $i$th client;
4 for $t = 1 : T$ do
5     Multicast $f_u^{t-1}$ to all the clients;
6     $y_{\text{list}} = [], C_{\text{rec}} = []$;
7     for client $i \in [N]$ do
8         Compute $y_i = A_i f_u^{t-1}$;
9         Send $y_i$ to the server;
10          while: rank$(C_{\text{rec}}) < k_3$ do
11              on Receiving message $y_j$ from client $j$;
12              $C_{\text{rec}} \leftarrow [C_{\text{rec}}, C_j], y_{\text{list}} \leftarrow [y_{\text{list}}, y_j]$;
13          dec$(C_{\text{rec}}, y_{\text{list}})$;
14      Compute $f_u^t$ via (16);
15 Return $Y_i = \text{sign}(f(x_i))$
```

4 Optimal design for coded distributed SSL

From the previous discussion, it can be seen that our purpose is to overcome the drawback of long computation time and make the algorithm straggler tolerant. Coding parameter k^* and matrix completion parameter r^* are key to the proposed algorithm.

In the coding algorithm, the parameter k stands for the redundancy of the algorithm. If k is too small, the run time of each task becomes too large that the overall runtime of the distributed coded algorithm may eventually increase. On the other hand, if k is too large, the degree of redundancy of the algorithm may not be enough to mitigate the impact of possible stragglers in the system.

In the matrix completion algorithm, the distance matrix is restored by its low-rank decomposition matrix, where the rank r of the low-rank matrix is a parameter that needs to be preset. As a result, the parameter r represents the computational matrix size. If r is too small, the accuracy of the distance matrix finally recovered may be too small to meet the requirements. On the other hand, if r is too large, the running time of each task during the iteration becomes too large.

4.1 Design of the optimal coding parameter k^*

We first consider the overall runtime of an uncoded distributed algorithm, assuming that the runtime of each task is the same and independent of each other. Then, we use T^i_{uncoded} to represent the runtime of the ith client for a computational task (the distribution is different in different computing scenarios). Thus, the runtime of an uncoded distributed algorithm can be given by

$$T_{\text{uncoded}} = \max\{T^1_{\text{uncoded}}, \cdots, T^N_{\text{uncoded}}\}. \tag{19}$$

Then, we consider the overall runtime for a coded distributed algorithm. T^i_{coded} represent the runtime of the ith client in a coded distributed algorithm. For a (N,k) MDS code, the decodable index set is a collection of any k indexes, i.e. $\mathcal{L} = \{i | i \in \{1,2,\cdots,n\}\}$, where the size of the set is denoted by $|\mathcal{L}| = k$. Therefore, the runtime of a coded distributed algorithm is

$$T_{\text{MDS-coded}} = \min_{\mathcal{L}} \max_{j\in\mathcal{L}} T^j_{\text{coded}}. \tag{20}$$

If the distribution of running time is subject to F, $\Pr(T_0 \leqslant t) = F(t)$. The design can be optimized based on the expected overall runtime if the overall runtime distribution of the entire coded distributed algorithm is calculated given the run time distribution and code parameters[24].

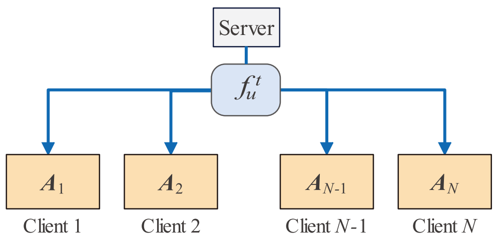
Assuming that the distribution of run times follows a shifted exponential distribution. The model is driven by Ref. [26], which the authors use to model file queries and latencies for cloud storage systems. The distribution running time is given by

$$\Pr(T_0 \leqslant t) = 1 - \mathrm{e}^{-\mu(t-1)}, t \geqslant 1, \tag{21}$$

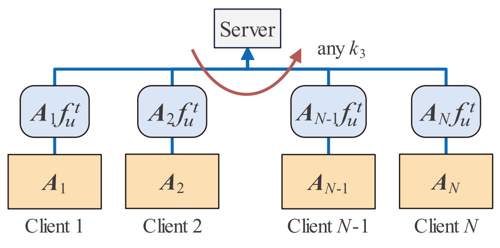
where μ is called a straggling parameter.

The average runtime of any distributed algorithm is bounded by $1/N$ in a distributed computing cluster with N worker nodes. Thus, the average operation time of uncoded and coded distributed algorithms can be expressed as

$$\mathrm{E}[T_{\text{uncoded}}] = \frac{1}{N}\left(1 + \frac{1}{\mu}\log N\right) = \Theta\left(\frac{\log N}{N}\right), \tag{22}$$



(a) In the beginning of the $(t+1)$th iteration, the server multicast f_u^t to clients.



(b) Server waits for the earliest responding k_3 clients results.

Fig. 4. Single iteration of coded distributed label propagation.

$$\mathrm{E}\left[T_{\text{MDS-coded}}\right] = \Theta\left(\frac{1}{N}\right). \tag{23}$$

Therefore, we can design the optimal k^* to achieve the goal with the empirical distribution of the run time known by solving

$$k^* = \left[1 + \frac{1}{W_{-1}\left(-\mathrm{e}^{-\mu-1}\right)}\right]N, \tag{24}$$

where $W_{-1}(\cdot)$ is the lower branch of lambert's function, and $W_{-1}(x)$ is the solution of $te^x = x$ $(t \leqslant -1)$.

4.2 Fitting initial value of matrix completion parameter r^*

In the matrix completion algorithm, the time required for each worker node in a single iteration to complete the matrix-matrix multiplication is $\Theta(M^2r)$. Moreover, the time required for the server to broadcast the matrix elements to the N worker nodes is $\Theta(Mr)$. Since the distance matrix rank has an upper bound $K+2$, the ratio of r to its upper bound is

$$\rho := \frac{r}{K+2}. \tag{25}$$

Therefore, the relationship between rank ratio ρ and computation time is linear.

The distance matrix represents the topology inside the original data set in the SSL algorithm. The approximate distance matrix completion algorithm can also be well-achieved by algorithm accuracy because the recovered approximate distance matrix can represent the topology inside the data set. Assuming that given the initial distance matrix $\boldsymbol{D}$, and the recovered approximate distance matrix $\tilde{\boldsymbol{D}}$, we define the average of the two matrix elements as $\mu := \text{average}(\boldsymbol{D})$, $\nu := \text{average}(\tilde{\boldsymbol{D}})$. Then we define the accuracy of matrix completion as

$$\gamma := 1 - \frac{\left\|\boldsymbol{D}/\mu - \tilde{\boldsymbol{D}}/\nu\right\|_F}{\left\|\tilde{\boldsymbol{D}}/\nu\right\|_F}. \tag{26}$$

Thus, we can select an appropriate r to perform the semi-supervised learning algorithm if we know the amount of computation of the worker node per unit time and the relationship between precision and rank ratio ρ in the matrix completion algorithm.

5 Simulation & discussion

We conduct simulations to evaluate the performance of the CDGSSL framework. Our experiment is performed on Alibaba Cloud elastic compute service (ECS) using two different working instance types: n4.small and n4.large. Set n4.large as the central server and the clients are all ECS instances of n4.small. We set the simulation parameters of the component algorithms as follows unless specified otherwise. For the coded matrix completion algorithm, we set $\beta = 1.0\mathrm{E}-5$, and the completion algorithm is done with $L = 10$ iterations. Moreover, for the coded label propagation algorithm, we set $\alpha = 1.0\mathrm{E}-3$ and the training is done in $T = 100$ iterations. The clients are set by $N = 10$ n4.small. In our experiments, we use MNIST and CIFAR10 data sets for the image classification task. Each training sample in MNIST is a 28×28 image of handwritten numbers, converted into a vector of size $K = 784$, with pixel values from $\{0,\cdots,255\}$. Samples in CIFAR10 are 32×32 images, converted into vectors of $K = 3072$, with pixel values from $\{0,\cdots,255\}$.

In Fig. 5, we measure 1500 times the round-trip time on the server and plot a complementary CDF. Note that the round-trip time consists of computation and communication time for one iteration of the matrix-vector multiplication. We obtain an empirical distribution of task run time to observe the frequency of straggler nodes in our test bench and to design the optimal k^*. The average round-trip time is 1.12 ms, and the 96th percentile delay is 1.51ms. The 96th percentile delay means that about 4 out of 100 computational tasks would be 1.5 times longer than the average task. We assume that a client has a 4% probability of becoming a straggler node. Thus, running an uncoded distributed algorithm with 10 clients has a 33% probability of straggler nodes. Moreover, the emergence of straggler nodes would cause the algorithm to be 1.5 times longer. Therefore, it is necessary to slow down the impact of this straggler node through coding. If we design a $k = 9$ coding algorithm, the probability of an uncoded distributed algorithm being affected by a straggler node with ten clients declines from 33% to 5.8%. If we design a $k = 8$ coding algorithm, the probability of an uncoded distributed algorithm being affected by a straggler node with ten clients declines from 33% to 0.62%.

The curve with the accuracy and rank ratio of the matrix completion distance algorithm ρ is shown in Fig. 6. The main purpose of obtaining this curve is to get the optimal rank ratio ρ^*. We randomly generate 3000 training data ranging from 0 to 25. The training data is divided into groups by their dimension d. Moreover, we calculate its block diagonal distance matrix and complete it by the matrix distance completion algorithm. We can observe that the accuracy increases rapidly to relatively flat as the rank ratio reaches 0.4. This indicates that the increase in calculation time may somewhat be much more costly for increasing accuracy. We repeatedly measure the time required for matrix multiplication in the matrix completion algorithm on the client n4.small when $r = 1$. The average time is $t = 8.064$ ms. Assuming we expect each increase in time at 0.5 s, the increase in accuracy is not less than 2%. Then according to the accuracy curve, we can

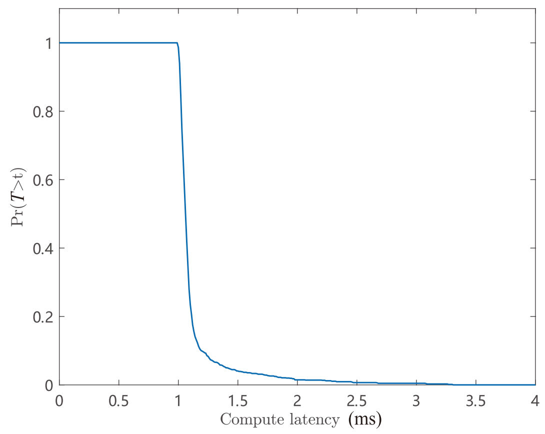


Fig. 5. The round-trip time of a task once between the central server and the client on ECS instance.

obtain the optimal rank ratio ρ^*=0.258.

We observe 100 random images from the MNIST data set and their labels after the CDGSSL task in Fig. 7. From the data set, we extract images of the numbers 0 and 1 and randomly assign $M_i = 1305$ images to each client. 40 randomly selected samples (rate=0.03) are labeled within each data set. According to the previous discussion of the optimal design of two essential parameters, we set up $r^* = 200$ for the matrix completion algorithm and $k = 9$ for the MDS code. We visually observe that the proposed CDGSSL algorithm is feasible, and its accuracy increases as the rank r increases.

In addition, we compare the proposed CDGSSL framework with the following three baseline schemes:

(Ⅰ) Local semi-supervised learning (LSSL). Each client only uses its local data set to perform the learning task of the label propagation algorithm. The client does not share information or communicate with other nodes in this scenario. Moreover, all the calculations are performed locally, which has the best privacy.

(Ⅱ) Centralized semi-supervised learning (CSSL). All data sets are concentrated on one node in the centralized semi-supervised learning scenario. Thus, the distance matrix can be computed by that node. It implements a label propagation algorithm on distance matrices without matrix completion or learning unknown labels. This baseline gives the best accuracy in graph-based learning scenarios.

(Ⅲ) Privacy-preserving semi-supervised learning (PSSL) in Ref. [22], the D-LapRLS algorithm. The clients collaborate to complete the distance matrix. Then the clients execute the label propagation algorithm through the global distance matrix. The D-LapRLS algorithm performs the best in DGSSL algorithms as far as accuracy is concerned.

Table 2 presents a comparison of the average classification accuracy of the CDGSSL framework and two baseline protocols. We conduct the simulations over two data sets including MNIST and CIFAR10. For the CDGSSL algorithm, we consider the average accuracy in three cases. Moreover, we set $k = 9$ for the CDGSSL algorithm. We can see that the

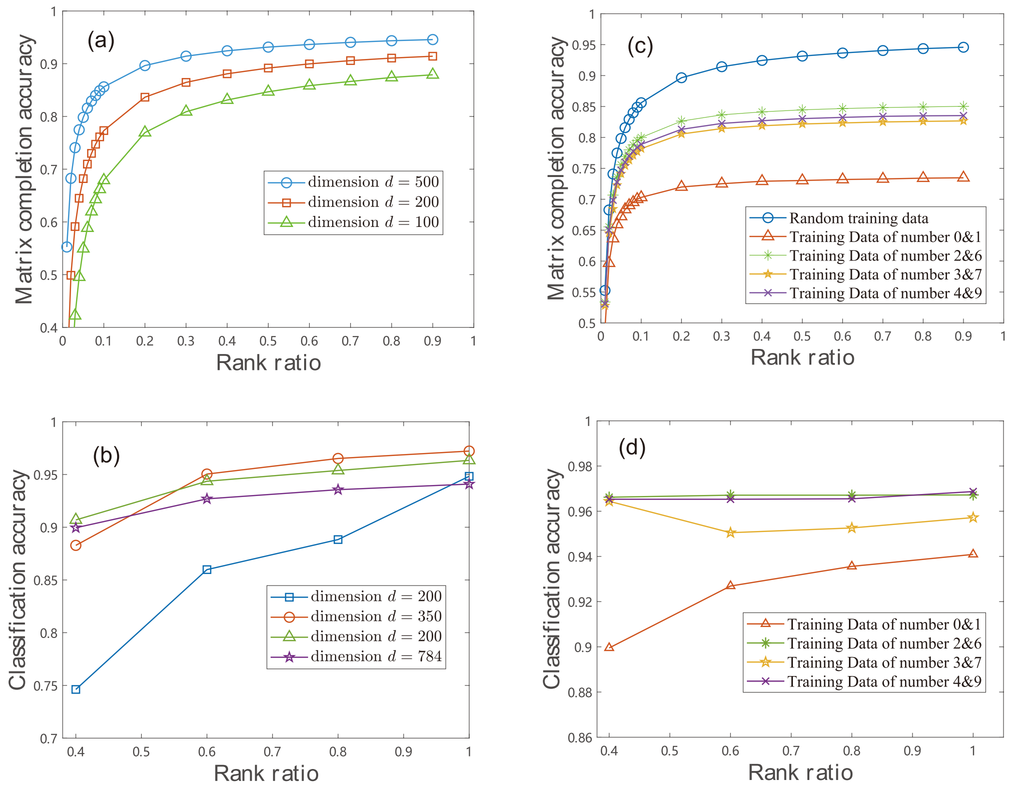
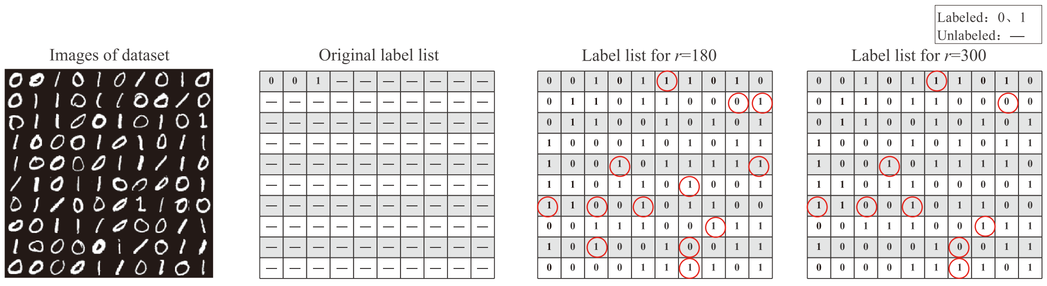


Fig. 6. Matrix completion accuracy and classification accuracy of with rank ratio $\rho = r/K+2$.


Labeled: 0、1
Unlabeled: —
Images of dataset

Original label list

0	0	1	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—

Label list for r=180

0	0	1	0	1	1	1	0	1	0
0	1	1	0	1	1	0	0	0	1
0	1	1	0	0	1	0	1	0	1
1	0	0	0	1	0	1	0	1	1
1	0	0	1	0	1	1	1	1	1
1	1	0	1	1	0	1	0	0	1
1	1	0	0	1	0	1	1	0	0
0	0	1	1	1	0	0	1	1	1
1	0	1	0	0	1	0	0	1	1
0	0	0	0	1	1	1	1	0	1

Label list for r=300

0	0	1	0	1	1	1	0	1	0
0	1	1	0	1	1	0	0	0	0
0	1	1	0	0	1	0	1	0	1
1	0	0	0	1	0	1	0	1	1
1	0	0	1	0	1	1	1	1	0
1	1	0	1	1	0	0	0	0	1
1	1	0	0	1	0	1	1	0	0
0	0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	1	1
0	0	0	0	1	1	1	1	0	1

Fig. 7. Illustration of the random 100 images from the MNIST data set and their label after the CDGSSL task, as the rank r increased. The highlighted label is the incorrect label after the training process.

Table 2. Average accuracy of two baseline protocols and different CDGSSL protocols with the rising rank r.

Dataset	Protocol(s)	Labeled&Unlabeled data	α	β	Rank(r)	Accuracy(%)
MNIST	CSSL	200&12665	1.0E – 03	–	–	95.99
	CDGSSL	200&12665	1.0E – 03	1.0E – 05	200	90.90
	CDGSSL	200&12665	1.0E – 03	1.0E – 05	150	89.73
	CDGSSL	200&12665	1.0E – 03	1.0E – 05	50	62.29
	LSSL	40&2533	1.0E – 03	–	–	60.63
CIFAR10	CSSL	400&10000	1.0E – 04	–	–	72.14
	CDGSSL	400&10000	1.0E – 04	1.0E – 05	3000	70.39
	CDGSSL	400&10000	1.0E – 04	1.0E – 05	2000	67.45
	CDGSSL	400&10000	1.0E – 04	1.0E – 05	1000	62.76
	LSSL	80&2000	1.0E – 04	–	–	52.60

accuracy increases as the rank r increases. On the other hand, as discussed in Section 4, the computational complexity increases as the rank r increases. The optimal design of rank r^* represents the trade-off between the accuracy and computational complexity provided by our framework.

Table 3 compares the CDGSSL algorithm with three baseline schemes. We compare the accuracy and computational complexity overhead of four protocols. We can observe that the computational complexity of CDGSSL is much lower than that of PSSL in both matrix completion and label propagation, while the accuracy maintains a high level. The comparison indicates that the proposed algorithm reduces the computational complexity under the distributed computing framework and can apply to large-scale data sets.

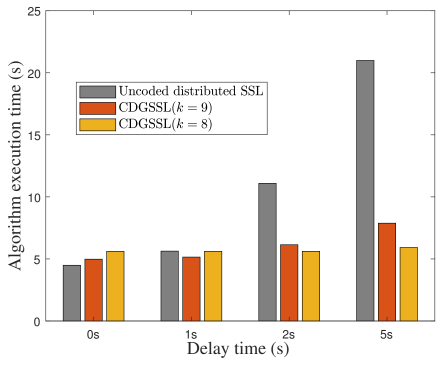
The average execution time of the CDGSSL algorithm and the uncoded algorithm (in Section 3.2) is presented in Fig. 8. In this experiment, we artificially add latency to each iteration, setting a 4% probability that each client would become a straggler node. The delay function is added using time.sleep() function when the client is selected to be a straggler node. We set $r = 100$ for the CDGSSL algorithm. The (Ⅱ) and (Ⅲ) schemes are under different k for the coding scheme design. As expected, as the latency increases, the effect of straggler on the uncoded baseline scheme is significant. In the meantime, the impact of straggler on the coding scheme is negligible, although the coding scheme would bring additional computational overhead. This additional overhead is negligible compared to the impact of the straggler node. Thus the coding scheme can slow down the impact of the straggler nodes present in this system on the algorithm.

6 Conclusions

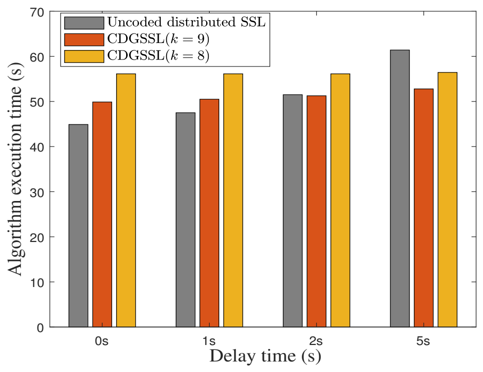
We have proposed the CDGSSL for binary classification problems in this work. The proposed algorithm has overcome the drawbacks of inefficient graph construction and straggler problem. We first provide a parallel and distributed solution of matrix completion for efficient graph construction. Then, we proposed CDGSSL based on MDS code to further tackle the straggler problem. Moreover, we have provided optimal parameters designed to improve the performance of

Table 3. Comparison of storage, matrix completion, label propagation, and accuracy overhead of: CSSL, LSSL, PSSL($r = 200$), CDGSSL($r^* = 200$).

Protocol	Storage	Matrix completion	Label propagation	Accuracy(%)
CSSL	$\Theta(MN)$	0	$\Theta(NM^2)$	95.99
LSSL	$\Theta(M)$	0	$\Theta(M^2)$	60.63
PSSL	$\Theta(M)$	$\Theta((NM)^2 r)$	$\Theta((NM)^2)$	93.28
CDGSSL	$\Theta(M)$	$\Theta\left(\frac{NM^2 r}{\log(N)}\right)$	$\Theta\left(\frac{NM^2}{\log(N)}\right)$	90.90



(a) Label Propagation



(b) Matrix Completion

Fig. 8. Comparison of average running time under different delay conditions overhead of uncoded algorithm and MDS-coded algorithm with $k = 9$ and $k = 8$.

CDGSSL. Finally, numerical experiments on Alibaba Cloud ECS have been made with the MNIST dataset. Simulation results have demonstrated the effectiveness of our proposed algorithm in alleviating the straggler effect by up to 33%, and attaining lower computational complexity, compared with existing methods.

Acknowledgements

This work was supported by the National Key R&D Program of China (2021YFB2900302).

Conflict of interest

The authors declare that they have no conflict of interest.

Biographies

Siqi Tan received the B.E. degree in Electronic Information Engineering from the University of Science and Technology of China (USTC) in 2020. Now she is currently pursuing the M.E. degree at the Department of Electronic Engineering and Information Science, USTC. Her research interests include wireless communications and coded distributed computation.

Li Chen received the B.E. degree in Electrical and Information Engineering from the Harbin Institute of Technology in 2009, and the Ph.D. degree in Electrical Engineering from the University of Science and Technology of China (USTC) in 2014. He is currently an Associate Professor at the Department of Electronic Engineering and Information Science, USTC. His research interests include integrated computation and communication, integrated sensing, and communication.

References

[1] van Engelen J E, Hoos H H. A survey on semi-supervised learning. *Machine Learning,* **2020**, *109*: 373–440.

[2] Ang J C, Mirzal A, Haron H, et al. Supervised, unsupervised, and semi-supervised feature selection: A review on gene selection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* **2016**, *13* (5): 971–989.

[3] Zhu X, Goldberg A B. Introduction to Semi-Supervised Learning. Cham, Switzerland: Springer, **2009**.

[4] Scudder H. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory,* **1965**, *11* (3): 363–371.

[5] Blum A, Mitchell T. Combining labeled and unlabeled data with co-training. In: COLT' 98: Proceedings of the Eleventh Annual Conference on Computational Learning Theory. New York: ACM, **1998**: 92–100.

[6] Belkin M, Niyogi P, Sindhwani V. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research,* **2006**, *7*: 2399–2434.

[7] Belkin M, Niyogi P. Semi-supervised learning on Riemannian manifolds. *Machine Learning,* **2004**, *56*: 209–239.

[8] Chapelle O, Schölkopf B, Zien A, Transductive support vector machines. In: Semi-Supervised Learning. Cambridge: MIT Press. **2006**, 105–117.

[9] Chong Y, Ding Y, Yan Q, et al. Graph-based semi-supervised learning: A review. *Neurocomputing,* **2020**, *408* (30): 216–230.

[10] Zhu X, Ghahramani Z, Lafferty J. Semi-supervised learning using Gaussian fields and harmonic functions. In: ICML'03: Proceedings of the Twentieth International Conference on International Conference on Machine Learning. Washington, DC: AAAI Press, **2003**: 912–919.

[11] Zhou D, Bousquet O, Lal T N, et al. Learning with local and global consistency. In: NIPS'03: Proceedings of the 16th International Conference on Neural Information Processing Systems. New York: ACM, **2003**: 321–328.

[12] Chen J, Wang C, Sun Y, et al. Semi-supervised Laplacian regularized least squares algorithm for localization in wireless sensor networks. *Computer Networks,* **2011**, *55* (10): 2481–2491.

[13] Szummer M, Jaakkola T. Partially labeled classification with Markov random walks. In: NIPS'01: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic. Cambridge: MIT Press, **2001**: 945–952.

[14] Grira N, Crucianu M, Boujemaa N. Active semi-supervised fuzzy clustering for image database categorization. In: MIR '05: Proceedings of the 7th ACM SIGMM International Workshop on Multimedia Information Retrieval. New York: ACM, **2005**: 9–16.

[15] Chapelle O, Zien A. Semi-supervised classification by low density separation. In: AISTATS 2005–Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics. Stuttgart, Germany: Max-Planck-Gesellschaft, **2005**: 57–64.

[16] Kostopoulos G, Karlos S, Kotsiantis S, et al. Semi-supervised regression: A recent review. *Journal of Intelligent & Fuzzy Systems,* **2018**, *35* (2): 1483–1500.

[17] Torii M, Wagholikar K, Liu H. Using machine learning for concept extraction on clinical documents from multiple data sources. *Journal of the American Medical Informatics Association,* **2011**, *18*: 580–587 .

[18] Scardapane S, Fierimonte R, Wang D, et al. Distributed music classification using random vector functional-link nets. In: 2015 International Joint Conference on Neural Networks (IJCNN). Killarney, Ireland: IEEE, **2015**: 1–8.

[19] Shih T K, Distributed multimedia databases In: Shih T K, editor. Distributed Multimedia Databases: Techniques and Applications. Hershey, PA: IGI Global, **2002**: 2–12.

[20] Shen P, Du X, Li C. Distributed semi-supervised metric learning. *IEEE Access,* **2016**, *4*: 8558–8571.

[21] Scardapane S, Fierimonte R, Di Lorenzo P, et al. Distributed semi-supervised support vector machines. *Neural Networks,* **2016**, *80*: 43–52 .

[22] Fierimonte R, Scardapane S, Uncini A, et al. Fully decentralized semi-supervised learning via privacy-preserving matrix completion. *IEEE Transactions on Neural Networks and Learning Systems,* **2017**, *28*: 2699–2711.

[23] Gan H, Li Z, Wu W, et al. Safety-aware graph-based semi-supervised learning. *Expert Systems With Applications,* **2018**, *107*: 243–254.

[24] Lee K, Lam M, Pedarsani R, et al. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory,* **2018**, *64* (3): 1514–1529.

[25] Chen L, Han K, Du Y, et al. Block-division-based wireless coded computation. *IEEE Wireless Communications Letters,* **2022**, *11* (2): 283–287.

[26] Agarwal A, Duchi J C. Distributed delayed stochastic optimization. In: 2012 IEEE 51st IEEE Conference on Decision and Control (CDC). Maui, USA: IEEE, **2012**: 5451–5452.

[27] Alfakih A Y, Khandani A K, Wolkowicz H. Solving euclidean distance matrix completion problems via semidefinite programming. *Computational Optimization and Applications,* **1999**, *12*: 13–30.

[28] Al-Homidan S, Wolkowicz H. Approximate and exact completion problems for Euclidean distance matrices using semidefinite programming. *Linear Algebra and Its Applications,* **2005**, *406*: 109–141.

[29] Liu W, Chen L, Zhang W. Decentralized federated learning: Balancing communication and computing costs. *IEEE Transactions on Signal and Information Processing Over Networks,* **2022**, *8*: 131–143.

[30] Liu W, Chen L, Chen Y, et al. Accelerating federated learning via momentum gradient descent. *IEEE Transactions on Parallel and Distributed Systems,* **2020**, *31* (8): 1754–1766.

[31] Wang Z, Du Y, Wei K, et al. Vision, application scenarios, and key technology trends for 6G mobile communications. *Science China Information Sciences,* **2022**, *65*: 151301.