Fully Decentralized Semi-supervised Learning via Privacy-Preserving Matrix Completion

Roberto Fierimonte, *Student Member, IEEE* Simone Scardapane, *Student Member, IEEE*
Aurelio Uncini, *Member, IEEE* and Massimo Panella, *Member, IEEE*

***Abstract*—Distributed learning refers to the problem of inferring a function when the training data is distributed among different nodes. While significant work has been done in the contexts of supervised and unsupervised learning, the intermediate case of semi-supervised learning in the distributed setting has received less attention. In this paper, we propose an algorithm for this class of problems, by extending the framework of manifold regularization. The main component of the proposed algorithm consists of a fully distributed computation of the adjacency matrix of the training patterns. To this end, we propose a novel algorithm for low-rank distributed matrix completion, based on the framework of diffusion adaptation. Overall, the distributed semi-supervised algorithm is efficient, scalable, and it can preserve privacy by the inclusion of flexible privacy-preserving mechanisms for similarity computation. Experimental results and comparison on a wide range of standard semi-supervised benchmarks validate our proposal.**

***Index Terms*—Semi-supervised Learning; Distributed Learning; Privacy-preserving; Matrix Completion**

I. Introduction

THE field of distributed learning (DL) is concerned with solving a learning problem, whenever the training data is distributed among a network of interconnected nodes [1]–[3]. Typically, we require DL protocols that are able to scale efficiently to large networks, without reliance on a single coordinating node or broadcasting capabilities. Research on this field is vast, including algorithms for DL on wireless sensor networks (WSNs) [4], peer-to-peer (P2P) networks [5], [6], distributed databases [7], and so on. Currently, research has focused mostly on the supervised and unsupervised learning settings. Examples of the former are algorithms for distributed neural networks [2], [8], distributed support vector machines [9]–[11], and distributed LASSO [3], [12]. For the latter, we can cite strategies for distributed clustering [13], and distributed dictionary learning [14].

In this sense, many crucial sub-areas of machine learning remain to be extended to the fully distributed scenario. Among these, the DL setting could benefit strongly from the availability of distributed protocols for semi-supervised learning (SSL) [15]. In SSL, it is assumed that the labeled training data is supplemented by additional unlabeled data, which has to be suitably exploited in order to improve the test accuracy. Frequently, labeled data is scarce while unlabeled data is abundant, making the design of efficient SSL algorithms critical for good performance. Currently, state-of-the-art research on SSL is concerned on the single-agent (centralized) case, e.g. with the use of manifold regularization (MR) [16], [17], transductive learning [18], and several others. To the best of our knowledge, the case of SSL over multiple agents has been addressed only in very specific settings, such as localization over WSNs [19], while no algorithm is available for the general case. However, we argue that such an algorithm would be well suited for a wide range of applications. As an example, consider the case of medical diagnosis, with labeled and unlabeled data distributed over multiple clinical databases [20]. Other examples include distributed multimedia classification over peer-to-peer networks, distributed music classification [21], distributed sensor inference [22], [23], and so on. In all of them, labeled data at every agent is costly to obtain, while unlabeled data is plentiful.

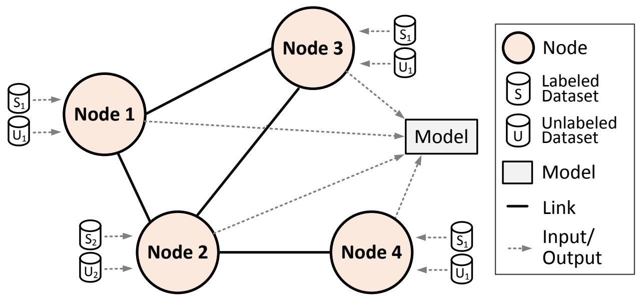


Fig. 1. Depiction of SSL over a network of agents. Each agent receives a labeled training dataset, together with an unlabeled one. The task is for all the nodes to converge to a single model, by exploiting all their local datasets.

The overall setting is summarized in Fig. 1, where each agent in a network receives two training datasets, one composed of labeled patterns and one composed of unlabeled patterns. Through local communication, all of them have to agree on the parameters of a single model, whose generalization performance should be comparable to having an (ideal) centralized agent collecting all the local datasets prior to training and solving the global optimization problem. By exploiting a fully distributed algorithm, as opposed to a centralized solution, it is possible to avoid communication bottlenecks on the network [8], preserve the privacy for sensible applications [24], and deploy the resulting algorithm even on largely unstructured networks, such as specific WSNs. We note that in DL, privacy has a very specific meaning. Particularly, we say

Authors are with the Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184, Rome. Email: roberto.fierimonte@gmail.com, {simone.scardapane,aurelio.uncini,massimo.panella}@uniroma1.it.
Corresponding author: M. Panella (email: massimo.panella@uniroma1.it).
Manuscript received July 11, 2015

that an algorithm is 'privacy-preserving' whenever an agent is not requested to communicate its training patterns, which in sensible applications can be considered as a breach of private information.

In this paper, we propose the first fully distributed algorithm for SSL over networks, satisfying the above requirements. In particular, we extend an algorithm belonging to the MR family, namely laplacian regularized least-square (LapRLS) [16]. MR algorithms, originated in the seminal works of [25] and [16], are based on the assumption that data often lie in a low-dimensional manifold $\mathcal{M}$ embedded in the higher-dimensional input space. When the structure of the manifold is unknown, it can be approximated well by a weighted graph where the vertexes are represented by the data points and the weights of the edges represent a measure of similarity between the points. In the MR framework, the classification function is obtained by solving an extension of the classical regularized optimization problem, with an additional regularization term, which incorporates information about the function's smoothness on the manifold.

The algorithm presented in this paper starts from the observation that, in the MR optimization problem, information is mostly encoded in a matrix $\mathbf{D}$ of pairwise distances between patterns. In fact, both the additional regularization term, and the kernel matrix (for any translation-invariant kernel function [26]) can be computed using the information about the distance between points. In the distributed setting, each agent can compute this matrix relatively only to its own training data, while information about the distance between points belonging to different agents are unknown. Obtaining this information would allow a very simple protocol for solving the overall optimization problem. As a consequence, we subdivide the training algorithm in two steps: a distributed protocol for computing $\mathbf{D}$, followed by a distributed strategy for solving the optimization problem.

For the former step, in the initial phase of the algorithm, we allow a small exchange of data patterns between agents. In this phase, privacy can be preserved with the inclusion of any privacy-preserving protocol for the computation of distances [24], [27]. For completeness, we describe the strategies that are used in our experiments in Section II-C. As a second step, we recover the rest of the global distance matrix $\mathbf{D}$ by building on previous works on Euclidean distance matrix (EDM) completion [28]–[31]. To this end, we consider two strategies. The first one is a simple modification of the state-of-the-art algorithm presented in [32], [33], which is based on a column-wise partitioning of $\mathbf{D}$ over the agents. In this paper, we modify it to take into account the specific nature of Euclidean distance matrices, by the incorporation of non-negativity and symmetry constraints. As a second strategy, we propose a novel algorithm for EDM completion, which is inspired to the framework of diffusion adaptation (DA) [34]. Our algorithm works by interleaving gradient descent steps with local interpolation of a suitable low-rank factorization of $\mathbf{D}$. While the first algorithm has a lower computational cost, we found that this comes at the cost of a worse performance, particularly when the sampling set of the matrix to complete is small. On the opposite, our algorithm exploits the particular structure of EDMs, at the cost of a possibly greater computational demanding. We discuss in more detail the advantages and disadvantages of the two approaches in Section III and in the experimental section.

As we stated before, once the matrix $\mathbf{D}$ is known, solving the rest of the optimization problem is trivial. In this paper we focus on the LapRLS algorithm, and we show that its distributed version can be solved using a single operation of sum over the network. This can be performed efficiently with the use of standard routines, such as the distributed consensus protocol [35], [36]. Due to space constraints, we leave the investigation of other MR algorithms (e.g. the laplacian SVM) to future works.

The rest of the paper is structured as follows: in Section II we introduce the theoretical tools upon which our algorithm is based. In particular, we detail the problem of SSL in the framework of MR in Section II-A, some notions of EDM completion in Section II-B, and two strategies for privacy-preserving similarity computation in Section II-C. In Section III we propose our algorithm to complete an EDM in a decentralized fashion. Then, Section IV details the proposed framework for distributed LapRLS. In Section V we present the results for both the distributed EDM completion and distributed LapRLS. Finally, Section VI concludes the paper with possible future developments.

Notation

In the rest of the paper, vectors are denoted by boldface lowercase letters, e.g. $\mathbf{a}$, while matrices are denoted by boldface uppercase letters, e.g. $\mathbf{A}$. All vectors are assumed to be column vectors unless otherwise specified. Symbol a_i denotes the ith element of vector $\mathbf{a}$, and A_{ij} the (i,j) entry of the matrix $\mathbf{A}$. The operator $\|\cdot\|_2$ is the standard L_2 norm on an Euclidean space. Finally, the notation $a[n]$ is used to denote dependence with respect to a time-instant n in an iterative procedure.

II. Preliminaries

In this section we introduce some concepts that are used in the development of our algorithm. We start by describing the basic setting of SSL in Section II-A. Then, we introduce the matrix completion problem and its application to the EDMs in Section II-B. As the last point, in Section II-C we report some results on privacy-preserving similarity computation.

A. Semi-supervised learning

In the SSL setting, the learning machine is provided with a set of l input/output labeled data $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and a set of u unlabeled data $U = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ [15]. In the following, inputs are assumed to be d-dimensional real vectors $\mathbf{x} \in X \subseteq \mathbb{R}^d$, while outputs are assumed to be scalars $y \in Y \subseteq \mathbb{R}$. The discussion can be extended straightforwardly to the case of a multi-dimensional output. It is assumed that labeled data is generated according to a joint probability distribution $\mathcal{P}_{X,Y}$, while unlabeled data are generated according to the marginal distribution $\mathcal{P}_X$ of $\mathcal{P}_{X,Y}$ [15].

In this paper, we consider one particular class of SSL algorithms belonging to the family of MR [16]. Practically, MR learning algorithms are based on three assumptions.

- Smoothness assumption: if two points $\mathbf{x}_1, \mathbf{x}_2 \in X$ are close in the intrinsic geometry of $\mathcal{P}_X$, then their conditional distributions $\mathcal{P}(y \mid \mathbf{x}_1)$ and $\mathcal{P}(y \mid \mathbf{x}_2)$ are similar.
- Cluster assumption: the decision boundary should lie in a low-density region of the input space X.
- Manifold assumption: the marginal distribution $\mathcal{P}_X$ is supported on a low-dimensional manifold $\mathcal{M}$ embedded in X.

Let $\mathcal{H}_K$ be a Reproducing Kernel Hilbert Space defined by the kernel function $K : X \times X \to \mathbb{R}$ with norm $\|f\|_K^2$, the approximation function for the SSL problem is estimated by solving:

$$f^* = \underset{f \in \mathcal{H}_K}{\operatorname{argmin}} \sum_{i=1}^{l} V(\mathbf{x}_i, y_i, f) + \gamma_A \|f\|_K^2 + \gamma_I \|f\|_I^2 \,, \quad (1)$$

where $V(\cdot,\cdot,\cdot)$ is a suitable loss function, $\|f\|_I^2$ is a penalty term that penalizes the structure of f with respect to the manifold and $\gamma_A, \gamma_I \geq 0$ are the regularization parameters. Usually, the structure of the manifold $\mathcal{M}$ is unknown and it must be estimated from both labeled and unlabeled data. In particular, we can define an adjacency matrix $\mathbf{W} \in \mathbb{R}^{l+u \times l+u}$, where each entry W_{ij} is a measure of similarity between patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ (see [16] for possible ways of constructing this matrix). Using this, the regularization term $\|f\|_I^2$ can be rewritten as [16]:

$$\|f\|_I^2 = f^{\mathrm{T}} \boldsymbol{\mathcal{L}} f \,, \quad (2)$$

where $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{l+u \times l+u}$ is the data adjacency graph Laplacian, given by $\boldsymbol{\mathcal{L}} = \mathbf{G} - \mathbf{W}$, with $\mathbf{G}$ a diagonal matrix with elements $G_{ii} = \sum_{j=1}^{N} W_{ij}$. Practically, the overall manifold $\mathcal{M}$ is approximated with an adjacency graph, which can be computed from both labeled and unlabeled data. In order to obtain better performances, usually a normalized Laplacian $\hat{\boldsymbol{\mathcal{L}}} = \mathbf{G}^{-1/2} \boldsymbol{\mathcal{L}} \mathbf{G}^{-1/2}$, or an iterated version $\hat{\boldsymbol{\mathcal{L}}}^q$, $q > 0$, is used [16]. An extension of the classical Representer Theorem proves that the function f^* is in the form of:

$$f^*(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) \,, \quad (3)$$

where $N = l + u$ and α_i are weight parameters. As we stated in the introduction, for simplicity in this paper we focus on a particular algorithm belonging to this framework, denoted as LapRLS. This is obtained by substituting Eq. (3) into problem (1) and setting a squared loss function:

$$V(\mathbf{x}_i, y_i, f) = \|y_i - f(\mathbf{x}_i)\|_2^2 \,. \quad (4)$$

Considering the dual optimization problem, by the optimality conditions the final parameters vector $\boldsymbol{\alpha}^* = [\alpha_1, \ldots, \alpha_N]^{\mathrm{T}}$ is easily obtained as:

$$\boldsymbol{\alpha}^* = \left(\mathbf{J}\mathbf{K} + \gamma_A \mathbf{I} + \gamma_I \boldsymbol{\mathcal{L}}\mathbf{K}\right)^{-1} \hat{\mathbf{y}} \,, \quad (5)$$

where $\hat{\mathbf{y}}$ is an N-dimensional vector with components:

$$\hat{y}_i = \begin{cases} y_i & \text{if } i \in \{1, \ldots, l\} \\ 0 & \text{if } i \in \{l+1, \ldots, l+u\} \end{cases} \,, \quad (6)$$

$\mathbf{J}$ is an $N \times N$ diagonal matrix with elements:

$$J_{ii} = \begin{cases} 1 & \text{if } i \in \{1, \ldots, l\} \\ 0 & \text{if } i \in \{l+1, \ldots, l+u\} \end{cases} \,, \quad (7)$$

and finally $\mathbf{K}$ is the $N \times N$ kernel matrix defined by $\{K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)\}$.

B. (Euclidean) matrix completion

The second notion that will be used in the proposed algorithm is the EDM completion problem [37]. A matrix completion problem is defined as the problem of recovering the missing entries of a matrix only from a set of known entries [29]. This problem has many practical applications, i.e. sensors localization, covariance estimation and customer recommendations, and it was largely investigated in the literature.

In this paper, we focus on completion of the square matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ containing the pairwise distances among the training patterns, i.e.:

$$D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \;\; \forall i, j = 1, \ldots, N \,. \quad (8)$$

$\mathbf{D}$ is called an Euclidean Distance Matrix (EDM). Clearly, Eq. (8) implies that $\mathbf{D}$ is symmetric and $D_{ii} = 0$ for all the elements on the main diagonal. It is possible to show that the rank r of $\mathbf{D}$ is upper bounded by $d + 2$, meaning that $\mathbf{D}$ is low-rank whenever $d \ll N$, which is common in all practical applications.

In the following, we suppose to have observed only a subset of entries of $\mathbf{D}$, in the form of a matrix $\hat{\mathbf{D}}$. More formally, there exists a matrix with binary entries $\boldsymbol{\Omega} \in [0,1]^{N \times N}$ such that:

$$\hat{\mathbf{D}} = \begin{cases} \hat{D}_{ij} = D_{ij} & \text{if } \Omega_{ij} = 1 \\ \hat{D}_{ij} = 0 & \text{otherwise} \end{cases} \,. \quad (9)$$

We wish to recover the original matrix $\mathbf{D}$ from $\hat{\mathbf{D}}$, i.e. we want to solve the following optimization problem:

$$\min_{\mathbf{D} \in \mathrm{EDM}(N)} \left\| \boldsymbol{\Omega} \circ \left(\hat{\mathbf{D}} - \mathbf{D} \right) \right\|_{\mathrm{F}}^2 \,, \quad (10)$$

where $\circ$ denotes the Hadamard product between two matrices, EDM(N) is the set of all EDMs of size N, and $\|\mathbf{A}\|_F$ is the Frobenius norm of matrix $\mathbf{A}$. It is possible to reformulate problem in Eq. (10) as a semidefinite problem by considering the Schoenberg mapping between EDMs and positive semidefinite matrices [37]:

$$\begin{aligned} \min_{\mathbf{D}} \quad & \left\| \boldsymbol{\Omega} \circ \left[\hat{\mathbf{D}} - \kappa(\mathbf{D}) \right] \right\|_{\mathrm{F}}^2 , \\ \text{s. t.} \quad & \mathbf{D} \succeq 0 \end{aligned} \quad (11)$$

where $\mathbf{D} \succeq 0$ means that $\mathbf{D}$ is positive semidefinite and:

$$\kappa(\mathbf{D}) = \mathrm{diag}(\mathbf{D})\mathbf{1}^{\mathrm{T}} + \mathbf{1}\mathrm{diag}(\mathbf{D})^{\mathrm{T}} - 2\mathbf{D} \,, \quad (12)$$

such that $\mathrm{diag}(\mathbf{D})$ extracts the main diagonal of $\mathbf{D}$ as a column vector. This observation motivated most of the initial research on EDM completion [37]. Recently, an alternative formulation was proposed in [31], which exploits the fact that every positive semidefinite matrix $\mathbf{D}$ with rank r admits a factorization $\{\mathbf{D} = \mathbf{V}\mathbf{V}^{\mathrm{T}}\}$, where $\mathbf{V} \in \mathbb{R}_{*}^{N \times r} = \{\mathbf{V} \in \mathbb{R}^{N \times r} : \det\left(\mathbf{V}^{\mathrm{T}}\mathbf{V}\right) \neq 0\}$. Using this factorization and assuming we know the rank of $\mathbf{D}$, problem (11) can be reformulated as:

$$\min_{\mathbf{V}\mathbf{V}^{\mathrm{T}} \in S_{+}(r,N)} \left\| \boldsymbol{\Omega} \circ \left[\hat{\mathbf{D}} - \kappa\left(\mathbf{V}\mathbf{V}^{\mathrm{T}}\right) \right] \right\|_{\mathrm{F}}^{2} , \tag{13}$$

where we have:

$$S_{+}(r,N) = \{\mathbf{U} \in \mathbb{R}^{N \times N} : \ \mathbf{U} = \mathbf{U}^{\mathrm{T}} \succeq 0, \ \mathrm{rank}\left(\mathbf{U}\right) = r\} . \tag{14}$$

C. *Privacy-preserving similarity computation*

As we stated in the introduction, a fundamental step in the algorithm presented in this paper is a distributed computation of similarity between two training patterns, i.e. a distributed computation of a particular entry of $\mathbf{D}$. If these patterns cannot be exchanged over the network, e.g. for privacy reasons, there is the need of implementing suitable protocols for privacy-preserving similarity computation. To show the applicability of the proposed approach, in our experimental simulations we make use of two state-of-the-art solutions to this problem. For completeness, we detail them here briefly.

More formally, the problem can be stated as follows. Given two training patterns $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$, belonging to different agents, we want to compute $\mathbf{x}_i^T \mathbf{x}_j$, without revealing the two patterns. Clearly, computing the inner product allows the computation of several other distance metrics, including the standard L_2 Euclidean norm. The first strategy that we investigate here is the random projection-based technique developed in [38]. Suppose that both agents agree on a projection matrix $\mathbf{R} \in \mathbb{R}^{m \times d}$, with $m < d$, such that each entry R_{ij} is independent and chosen from a normal distribution with mean zero and variance σ^2. We have the following lemma:

Lemma 1. *Given two input patterns $\mathbf{x}_i, \mathbf{x}_j$, and the respective projections:*

$$\mathbf{u}_i = \frac{1}{\sqrt{m}\sigma}\mathbf{R}\mathbf{x}_i, \textit{ and } \mathbf{u}_j = \frac{1}{\sqrt{m}\sigma}\mathbf{R}\mathbf{x}_j , \tag{15}$$

we have that:

$$\mathbb{E}\left\{\mathbf{u}_i^T \mathbf{u}_j\right\} = \mathbf{x}_i^T \mathbf{x}_j . \tag{16}$$

Proof. See [38, Lemma 5.2]. □

In light of Lemma 1, exchanging the projected patterns instead of the original ones allows to preserve, on average, the inner product. A thorough investigation on the privacy-preservation guarantees of this protocol can be found in [38]. Additionally, we can observe that this protocol provides a reduction on the communication requirements of the application, since it effectively reduces the dimensionality of the patterns to be exchanged by a factor m/d.

The second protocol that we investigate in our experimental section is a more general (nonlinear) transformation introduced in [39]. It is given by:

$$\mathbf{v} = \mathbf{b} + \mathbf{Q}\tanh\left(\mathbf{a} + \mathbf{C}\mathbf{x}\right) , \tag{17}$$

for a generic input pattern $\mathbf{x}$, where $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{Q} \in \mathbb{R}^{m \times t}$, $\mathbf{a} \in \mathbb{R}^t$, $\mathbf{C} \in \mathbb{R}^{t \times d}$ are matrices whose entries are drawn from normal distributions with mean zero and possibly different variances. As in the previous method, it is possible to show that the inner product is approximately preserved, provided that the input patterns are not "outliers" in a specific sense. See [39] for more details and an analysis of the privacy-preservation capabilities of this scheme. Again, choosing t and m allows to balance between a more accurate reconstruction and a reduction on the input dimensionality.

The field of privacy-preserving similarity computation, and more in general privacy-preserving data mining, is vast and with more methods introduced each year. Although we have chosen these two protocols due to their wide diffusion and simplicity, we stress that our algorithm does not depend specifically on any of them. We refer to [24], [27], [40] for more general investigations on this field.

III. Distributed Laplacian Estimation

In this section, we start by formulating a problem of distributed estimation of $\mathcal{L}$ in Section III-A. Then, we focus on two algorithms for its solution. The first is a modification of a state-of-the-art algorithm, described in Section III-B, while the second is a fully novel protocol which is based on the ideas of 'diffusion adaptation' [34] introduced in Section III-C.

A. *Formulation of the problem*

In the distributed Laplacian estimation problem, we suppose that both the labeled data and the unlabeled data are distributed through a network of L interconnected agents, as shown in Fig. 1. The connectivity within the agents is fixed and known a priori in the form of a matrix $\mathbf{C} \in \mathbb{R}^{L \times L}$, where $C_{ij} \neq 0$ if and only if agents i and j are connected, and the value of the element represents the weight that agent j assigns to the information originated from agent i. We denote with $\mathcal{N}_k$ the set of neighbors of the kth agent, i.e. the set of indexes i such that $C_{ki} \neq 0$. Possible choices of the connectivity weights for fixed and undirected topologies are discussed in [41].

Without loss of generality, we assume that data is organized as follows: the kth agent is provided with N_k patterns, such that $N = \sum_{k=1}^{L} N_k$. For each agent, the first l_k patterns are labeled: $S_k = \{(\mathbf{x}_{k,1}, y_{k,1}), \ldots, (\mathbf{x}_{k,l_k}, y_{k,l_k})\}$, while the last u_k are unlabeled: $U_k = \{\mathbf{x}_{k,l_k+1}, \ldots, \mathbf{x}_{k,l_k+u_k}\}$. The local data sets are non-overlapping, so we have $S = \cup_{k=1}^{L} S_k$ and $U = \cup_{k=1}^{L} U_k$.

Let $\boldsymbol{\mathcal{L}}_k \in \mathbb{R}^{N_k \times N_k}$, $k = 1 \ldots L$, be the Laplacian matrices computed by each agent using its own data; we are interested in estimating in a totally decentralized fashion the Laplacian matrix $\boldsymbol{\mathcal{L}}$ calculated with respect to all the N patterns. The local Laplacian matrices can be always expressed, rearranging the rows and the columns, as block matrices on the main diagonal of $\boldsymbol{\mathcal{L}}$:

$$\boldsymbol{\mathcal{L}} = \begin{bmatrix} \boldsymbol{\mathcal{L}}_1 & ? & ? \\ ? & \ddots & ? \\ ? & ? & \boldsymbol{\mathcal{L}}_L \end{bmatrix} \tag{18}$$

The same structure of (18) applies also to matrices $\mathbf{D}$ and $\mathbf{K}$, with $\mathbf{D}_k$ and $\mathbf{K}_k$ representing the distance matrix and kernel matrix computed over the local dataset. This particular structure implies that the sampling set is not random, and makes non-trivial the problem of completing $\boldsymbol{\mathcal{L}}$ solely from the knowledge of the local matrices. At the opposite, the idea of exchanging the entire local datasets between nodes is unfeasible because of the amount of data to share. Instead of completing in a distributed manner the global Laplacian matrix, in this paper we considered the alternative approach of computing the global EDM $\mathbf{D}$ first, and then using it to calculate the Laplacian. This approach has two advantages:

- We can exploit the structure of EDMs to design efficient algorithms.
- From the global EDM we can compute, in addition to the Laplacian, the kernel matrix $\mathbf{K}$ for all kernel functions K based on Euclidean distance (e.g. gaussian kernel) [26].

Based on these considerations, we propose a framework for the distributed estimation of $\boldsymbol{\mathcal{L}}$, which consists in five steps:

1) Patterns exchange: every agent exchanges a fraction p of the available input data (both labeled and unlabeled) with its neighbors. This step is necessary so that the agents can increase the number of known entries in their local matrices. In order to maximize the diffusion of the data within the network, this step is iterated $n_{\text{max}}^{(1)}$ times; at every iteration an increasing percentage of shared data is constituted by pattern received by the neighbors in previous iterations. A simple strategy to do this consists, at the iteration n, to choose $\frac{n_{\text{max}}-n+1}{n_{\text{max}}}p$ patterns from the local dataset, and $\frac{n-1}{n_{\text{max}}}p$ patterns received in the previous $n-1$ iterations. In order to preserve privacy, this step can include one of the privacy-preserving strategies showed in Section II-C.
2) Local EDM computation: each agent computes, using its original dataset and the data received from its neighbors, an incomplete approximation $\hat{\mathbf{D}}_k \in \mathbb{R}^{N\times N}$ of the real EDM matrix $\mathbf{D}$.
3) Entries exchange: the agents exchange a sample of their local EDMs $\hat{\mathbf{D}}_k$ with their neighbors. Again, this step is iterated $n_{\text{max}}^{(2)}$ times using the same rule of step 1.
4) Distributed EDM completion: the agents complete the estimate $\tilde{\mathbf{D}}$ of the global EDM using one of the distributed algorithms presented in the following sections.
5) Global Laplacian estimation: using $\tilde{\mathbf{D}}$ the agents compute the global Laplacian estimate $\tilde{\boldsymbol{\mathcal{L}}}$ and the Kernel matrix estimate $\tilde{\boldsymbol{K}}$.

B. Decentralized block estimation

As stated in the introduction, the first algorithm that we take into account for the decentralized completion of $\mathbf{D}$ is a modified version of the algorithm named *D-LMaFit* [32], [33]. To the best of our knowledge, this is the only existing algorithm for distributed matrix completion available in the literature.

Let $\hat{\mathbf{D}}$ be the incomplete global EDM matrix and denote with $\mathcal{I}$ the set of indexes corresponding to its known entries. In a centralized setting, without taking into account the structure of distance matrices, and assuming that the rank r is known, $\tilde{\mathbf{D}}$ can be completed by solving the problem:

$$\begin{aligned} \min_{\mathbf{A},\mathbf{B},\tilde{\mathbf{D}}} \quad & \left\|\mathbf{AB} - \tilde{\mathbf{D}}\right\|_{\text{F}}^2 \\ \text{s. t.} \quad & \tilde{D}_{ij} = \hat{D}_{ij},\ \forall (i,j) \in \mathcal{I} \end{aligned} , \tag{19}$$

where $\mathbf{A} \in \mathbb{R}^{N\times r}$, $\mathbf{B} \in \mathbb{R}^{r\times N}$ represent a suitable low-rank factorization of $\tilde{\mathbf{D}}$.

In extending problem (19) to a decentralized setting, the algorithm presented in [32] considers a column-wise partitioning of $\hat{\mathbf{D}}$ over the agents. For simplicity of notation, we suppose here that this partitioning is such that the kth agent stores only the columns corresponding to its local dataset. Thus, the block partitioning has the form $\hat{\mathbf{D}} = \left[\hat{\mathbf{D}}_1, \dots, \hat{\mathbf{D}}_L\right]$, where $\hat{\mathbf{D}}_k \in \mathbb{R}^{N\times N_k}$ is the block of the matrix held by the kth agent, and $\mathcal{I}_k$ is the set of indexes of known entries of $\hat{\mathbf{D}}_k$. The same block partition applies also to matrices $\{\mathbf{B} = [\mathbf{B}_1, \dots, \mathbf{B}_L]\}$, with $\mathbf{B}_k \in \mathbb{R}^{r\times N_k}$, and $\tilde{\mathbf{D}} = \left[\tilde{\mathbf{D}}_1, \dots, \tilde{\mathbf{D}}_L\right]$, with $\tilde{\mathbf{D}}_k \in \mathbb{R}^{N\times N_k}$. The matrix $\mathbf{A}$ cannot be partitioned, but each agent stores a local copy $\mathbf{A}_k$ to use in computations. The *D-LMaFit* algorithm consists in an alternation of matrix factorizations and inexact average consensus, formalized in the following steps:

1) Initialization: For each agent, the matrices $\mathbf{A}_k[0]$ and $\mathbf{B}_k[0]$ are initialized as random matrices of appropriate dimensions. Matrix $\tilde{\mathbf{D}}_k[0]$ is initialized as $\tilde{\mathbf{D}}_k[0] = \hat{\mathbf{D}}_k$.
2) Update of $\mathbf{A}$: At time n, the kth agent updates its local copy of the matrix $\mathbf{A}$. If $n = 0$, the updating rule is:
$$\mathbf{A}_k[1] = \sum_{i=1}^{L} C_{ki}\mathbf{A}_i[0] - \alpha\left(\mathbf{A}_k[0] - \tilde{\mathbf{D}}_k[0]\mathbf{B}_k^{\text{T}}[0]\right), \tag{20}$$
where α is a suitable positive step-size. If $n > 0$, the updating rule is given in Eq. (B1) at the bottom of the page. In Eq. (B1), $\tilde{\mathbf{C}}$ is a mixing matrix that satisfies some properties [33]. A suitable choice is $\tilde{\mathbf{C}} = (1/2)(\mathbf{I} + \mathbf{C})$.
3) Update of $\mathbf{B}$ and $\tilde{\mathbf{D}}$: At the nth iteration, agent k updates matrices $\mathbf{B}_k$ and $\tilde{\mathbf{D}}_k$ according to:

$$\mathbf{A}_k[n+1] = \mathbf{A}_k[n] - \sum_{i=1}^{L}\left(C_{ki}\mathbf{A}_i[n] - \tilde{C}_{ki}\mathbf{A}_i[n-1]\right) - \alpha\left(\mathbf{A}_k[n] - \mathbf{A}_k[n-1] - \tilde{\mathbf{D}}_k[n]\mathbf{B}_k^{\text{T}}[n] + \tilde{\mathbf{D}}_k[n-1]\mathbf{B}_k^{\text{T}}[n-1]\right) \tag{B1}$$

$$\mathbf{B}_k\left[n+1\right] = \mathbf{A}_k^{\dagger}[n+1]\mathbf{A}_k^{\mathrm{T}}\left[n+1\right]\tilde{\mathbf{D}}_k\left[n\right] \tag{21}$$

$$\tilde{\mathbf{D}}_k\left[n+1\right] = \mathbf{A}_k\left[n+1\right]\mathbf{B}_k\left[n+1\right] + \mathrm{P}_{\mathcal{I}_k}\left(\hat{\mathbf{D}}_k - \mathbf{A}_k\left[n+1\right]\mathbf{B}_k\left[n+1\right]\right) \tag{22}$$

where $\mathbf{A}_k^{\dagger}[n+1]$ is the Moore-Penrose inverse of $\mathbf{A}_k[n+1]$, and $P_{\mathcal{I}}\left(\mathbf{M}\right) : \mathbb{R}^{n\times m} \rightarrow \mathbb{R}^{n\times m}$ is a projection operator defined by:

$$P_{\mathcal{I}}\left(\mathbf{M}\right)_{ij} = \begin{cases} M_{ij} & \text{if } (i,j)\in\mathcal{I} \\ 0 & \text{otherwise} \end{cases} . \tag{23}$$

The algorithm stops when the maximum number of iterations $n_{\max}^{\text{EDM}}$ is reached.

As we stated, *D-LMaFit* is not specifically designed for EDM completion. Consequently, it has some important limitations in our context. In particular, the resulting matrix $\tilde{\mathbf{D}}$ can have negative entries and could be non-symmetric; moreover, it is distributed across the nodes and so, if an agent want to access to the complete matrix, it has to collect the local matrices $\tilde{\mathbf{D}}$ through all the network. In order to at least satisfy the constraint that $\tilde{\mathbf{D}}$ be an appropriate EDM, we introduce the following modifications into the original algorithm:

- The updating equation for $\tilde{\mathbf{D}}_k$ is modified by setting to 0 all the negative entries. This projection operator is a standard approach in non-negative matrix factorization to enforce non-negativity constraints [42].
- When all the agents gathered the complete matrix $\tilde{\mathbf{D}}$, this is symmetrized as $\tilde{\mathbf{D}} = \frac{\tilde{\mathbf{D}}+\tilde{\mathbf{D}}^{\mathrm{T}}}{2}$.

C. Diffusion gradient descent

The second algorithm for distributed EDM completion proposed in this paper exploits the low-rank factorization $\mathbf{D} = \kappa(\mathbf{V}\mathbf{V}^{\mathrm{T}})$ showed in Section II-B. In particular, we consider the general framework of DA [34], which is a family of methods for jointly optimizing a sum of cost functions over a group of agents. The optimization is performed by interleaving local gradient descent steps with 'diffusion' steps, where neighboring estimates are fused at every node. DA has been used for a wide range of distributed optimization tasks, including distributed filtering [43], dictionary learning [14], and control of connectivity over graphs [44], among others.

To begin with, we can observe that the objective function in Eq. (13) can be approximated locally by:

$$J_k(\mathbf{V}) = \left\|\mathbf{\Omega}_k \circ \left[\hat{\mathbf{D}}_k - \kappa\left(\mathbf{V}\mathbf{V}^{\mathrm{T}}\right)\right]\right\|_{\mathrm{F}}^2 \quad k=1,\ldots,L\ , \tag{24}$$

where $\mathbf{\Omega}_k$ is the local auxiliary matrix associated with $\hat{\mathbf{D}}_k$. Hence, we can exploit a DA algorithm to minimize the joint cost function given by $\tilde{J}(\mathbf{V}) = \sum_{k=1}^{L} J_k(\mathbf{V})$. The diffusion gradient descent for the distributed completion of an EDM is defined by an alternation of updating and diffusion equations in the form of:

1) Initialization: All the agents initialize the local matrices $\mathbf{V}_k$ as random $N\times r$ matrices.
2) Update of $\mathbf{V}$: At time n, the kth agent updates the local matrix $\mathbf{V}_k$ using a gradient descent step with respect to its local cost function:

$$\tilde{\mathbf{V}}_k\left[n+1\right] = \mathbf{V}_k[n] - \eta_k[n]\nabla_{\mathbf{V}_k}J_k(\mathbf{V})\,. \tag{25}$$

where $\eta_k\left[n\right]$ is a positive step-size. It is straightforward to show that the gradient of the cost function is given by:

$$\nabla_{\mathbf{V}_k}J_k(\mathbf{V}) = \kappa^*\Big\{\mathbf{\Omega}_k\circ \circ\left(\kappa\left(\mathbf{V}_k\left[n\right]\mathbf{V}_k^{\mathrm{T}}\left[n\right]\right) - \hat{\mathbf{D}}_k\right)\Big\}\mathbf{V}_k\left[n\right]\ , \tag{26}$$

where $\kappa^*(\mathbf{A}) = 2\left[\operatorname{diag}\left(\mathbf{A}1\right) - \mathbf{A}\right)$ is the adjoint operator of κ.
3) Diffusion: The updated matrices are combined according to the mixing weights $\mathbf{C}$:

$$\mathbf{V}_k\left[n+1\right] = \sum_{i=1}^{L} C_{ki}\tilde{\mathbf{V}}_i\left[n+1\right]. \tag{27}$$

For a rationale of this approach, and an analysis of its convergence behavior in the case of convex cost functions, we refer to any introductory publication on DA [34], [45], [46]. Convergence of a similar family of algorithms in the case of non-convex cost functions is instead derived in [47]. Compared with the state-of-the-art decentralized block algorithm presented in the previous section, the diffusion-based approach has two main advantages. First, it is able to take into account naturally the properties of EDM matrices. Secondly, at every step each node has a complete estimate of the overall matrix, instead of a single column-wise block. Thus, there is no need of gathering the overall matrix at the end of the optimization process.

IV. Distributed Semi-supervised Manifold Regularization

In this section, we consider the more general distributed SSL setting, as illustrated in Fig. 1. We suppose that the agents in the network have performed a distributed matrix completion step, using either the algorithm in Section III-B or the one in Section III-C, so that the estimates $\tilde{\mathbf{D}}$, $\tilde{\boldsymbol{\mathcal{L}}}$ and $\tilde{\mathbf{K}}$ are globally known. For the kth agent, we denote with $\hat{\mathbf{y}}_k$ the N_k dimensional vector with elements:

$$\hat{y}_{k,i} = \begin{cases} y_{k,i} & \text{if } i\in\{1,\ldots,l_k\} \\ 0 & \text{if } i\in\{l_k+1,\ldots,l_k+u_k\} \end{cases} , \tag{28}$$

and $\hat{\mathbf{J}}_k$ the $N_k\times N$ matrix defined by $\hat{\mathbf{J}}_k = \begin{bmatrix}\overline{\mathbf{0}}_k & \mathbf{\Lambda}_k & \underline{\mathbf{0}}_k\end{bmatrix}$, where $\mathbf{\Lambda}_k$ is a $N_k\times N_k$ diagonal matrix with elements:

$$\Lambda_{k,ii} = \begin{cases} 1 & \text{if } i\in\{1,\ldots,l_k\} \\ 0 & \text{if } i\in\{l_k+1,\ldots,l_k+u_k\} \end{cases} , \tag{29}$$

$\overline{\mathbf{0}}_k$ is a $N_k\times\sum_{j<k}N_j$ null matrix and $\underline{\mathbf{0}}_k$ is a $N_k\times\sum_{j>k}N_j$ null matrix. Using this notation, the optimization problem of LapRLS can be reformulated in distributed form as:

$$\min_{\boldsymbol{\alpha}} \sum_{k=1}^{L} \|\hat{\mathbf{y}}_k - \hat{\mathbf{J}}_k\tilde{\mathbf{K}}\boldsymbol{\alpha}\|_2^2 + \gamma_A\boldsymbol{\alpha}^{\mathrm{T}}\tilde{\mathbf{K}}\boldsymbol{\alpha} + \gamma_I\boldsymbol{\alpha}^{\mathrm{T}}\tilde{\mathbf{K}}\tilde{\boldsymbol{\mathcal{L}}}\tilde{\mathbf{K}}\boldsymbol{\alpha}\ . \tag{30}$$

TABLE I
PSEUDOCODE OF THE PROPOSED DISTRIBUTED SSL ALGORITHM, AT THE kTH AGENT.

```
Input: Labeled $S_k$ and unlabeled $U_k$ training data, number of nodes $L$ (global), regularization parameters $\gamma_A$, $\gamma_I$ (global)
Output: Optimal vector $\boldsymbol{\alpha}_k^*$
 1: for $n = 1$ to $n_{\text{max}}^1$ do
 2:     Select a set of input patterns and share them with the neighbors $\mathcal{N}_k$, using a privacy-preserving transformation if needed.
 3:     Receive patterns from the neighbors.
 4: end for
 5: Compute the incomplete EDM matrix $\hat{\mathbf{D}}_k$.
 6: for $n = 1$ to $n_{\text{max}}^2$ do
 7:     Select a set of entries from $\hat{\mathbf{D}}_k$ and share them with the neighbors.
 8:     Receive entries from the neighbors.
 9:     Update $\hat{\mathbf{D}}_k$ with the entries received.
10: end for
11: Complete the matrix $\tilde{\mathbf{D}}$ using the algorithm presented in Sec. III-B or in Sec. III-C.
12: Compute the Laplacian matrix $\tilde{\mathcal{L}}$ and the kernel matrix $\tilde{\mathbf{K}}$ using $\tilde{\mathbf{D}}$.
13: Compute the sum $\hat{\mathbf{J}}_{\text{tot}}$ over the network using the DAC protocol.
14: return $\boldsymbol{\alpha}_k^*$ according to Eq. (32).
```

Denoting with $\hat{\mathbf{J}}_{\text{tot}} = \sum_{k=1}^{L} \hat{\mathbf{J}}_k^{\text{T}} \hat{\mathbf{J}}_k$ and $\hat{\mathbf{y}}_{\text{tot}} = \sum_{k=1}^{L} \hat{\mathbf{J}}_k^{\text{T}} \hat{\mathbf{y}}_k$, we can derive the expression for the optimal weights vector $\boldsymbol{\alpha}^*$:

$$\boldsymbol{\alpha}^* = \left(\hat{\mathbf{J}}_{\text{tot}} \tilde{\mathbf{K}} + \gamma_A \mathbf{I} + \gamma_I \tilde{\mathcal{L}} \tilde{\mathbf{K}} \right)^{-1} \hat{\mathbf{y}}_{\text{tot}} \,. \tag{31}$$

The particular structure of $\boldsymbol{\alpha}^*$ implies that the distributed solution can be decomposed as $\boldsymbol{\alpha}^* = \sum_{k=1}^{L} \boldsymbol{\alpha}_k^*$, where:

$$\boldsymbol{\alpha}_k^* = \left(\hat{\mathbf{J}}_{\text{tot}} \tilde{\mathbf{K}} + \gamma_A \mathbf{I} + \gamma_I \tilde{\mathcal{L}} \tilde{\mathbf{K}} \right)^{-1} \hat{\mathbf{J}}_k^{\text{T}} \hat{\mathbf{y}}_k \,. \tag{32}$$

To compute the local solution $\boldsymbol{\alpha}_k^*$, the kth agent requires only the knowledge of matrix $\hat{\mathbf{J}}_{\text{tot}}$, which can be computed with a distributed sum over the network. This is a primitive operation in most families of networks, including WSNs and P2P networks. In the WSN case, in particular, this is typically achieved with the use of a distributed average consensus protocol (DAC) [35], [36], [41]. DAC, or simply consensus, is a totally distributed iterative protocol designed to compute the average of a series of measurements within a network. Basically, this is an iterative process which, at every time instant n, updates the current local estimate $\boldsymbol{\Gamma}_k[n-1]$ of $\hat{\mathbf{J}}_{\text{tot}}$ as:

$$\boldsymbol{\Gamma}_k[n] = \sum_{i=1}^{L} C_{ki} \boldsymbol{\Gamma}_i[n-1] \,. \tag{33}$$

Convergence of Eq. (33) is guaranteed for a wide range of choices of $\mathbf{C}$. Clearly, the sum can be obtained by post-multiplying the final estimate by L. For more information, we refer to [35], [36], [41]. The DAC protocol will be used in the experimental section.

TABLE II
DESCRIPTION OF THE DATASETS.

Name	Features	Size	N. Classes	|TR|	|TST|	|U|
2Moons	2	400	2	14	200	186
BCI	117	400	2	14	100	286
G50C	50	550	2	50	186	314
COIL20	1024	1440	20	40	400	1000
COIL2	1024	1440	2	40	400	1000

Overall, the distributed LapRLS algorithm can be summarized in five main steps:

1) Distributed Laplacian estimation: this step corresponds to the process illustrated in Sec. III. It includes the patterns exchange (with the inclusion of a privacy-preserving strategy, if needed) and the points exchange procedures, the distributed EDM completion, and the computation of $\tilde{\mathcal{L}}$ and $\tilde{\mathbf{K}}$.
2) Global sum of $\hat{\mathbf{J}}_{\text{tot}}$: in this step the local matrices $\hat{\mathbf{J}}_k^{\text{T}} \hat{\mathbf{J}}_k$ are summed up using the DAC protocol.
3) Local training: using the matrix $\hat{\mathbf{J}}_{\text{tot}}$ computed in the previous step, each agent calculate its local solution, given by:
$$\boldsymbol{\alpha}_k^* = \left(\hat{\mathbf{J}}_{\text{tot}} \tilde{\mathbf{K}} + \gamma_A \mathbf{I} + \gamma_I \tilde{\mathcal{L}} \tilde{\mathbf{K}} \right)^{-1} \hat{\mathbf{J}}_k^{\text{T}} \hat{\mathbf{y}}_k \,. \tag{34}$$
4) Global sum of $\boldsymbol{\alpha}^*$: in this step, using the DAC protocol, the local vectors $\boldsymbol{\alpha}_k^*$ are summed up to compute the global weights vector.
5) Output estimation: when a new unlabeled pattern $\mathbf{x}$ is available to the network, each agent can initialize a partial output as:
$$f_k(\mathbf{x}) = \sum_{i=1}^{N_k} K(\mathbf{x}, \mathbf{x}_{k,i}) \beta_{k,i}^* \,, \tag{35}$$
where $\boldsymbol{\beta}_k^*$ is a N_k-dimensional vector containing the entries of $\boldsymbol{\alpha}^*$ corresponding to the patterns belonging to the kth agent. The global output is then computed as:
$$f(\mathbf{x}) = \sum_{k=1}^{L} f_k(\mathbf{x}) \,, \tag{36}$$
which can be obtained efficiently with the use of the DAC protocol.

A pseudocode of the algorithm, from the point of view of a single agent, is provided in Table I.

V. EXPERIMENTAL RESULTS

A. Experiments setup

We tested the performance of our proposed algorithm over five publicly available datasets. In order to get comparable results with state-of-the-art SSL algorithms, the datasets were chosen among a variety of benchmarks for SSL. A schematic overview of their characteristics is given in Tab. II. The datasets are both synthetic (G50C and 2Moons) and taken from real-world examples (BCI and COIL). G50C is a binary classification task, whose inputs are constructed from overlapping 50-dimensional Gaussians [17]. 2Moons is another artificial dataset, where inputs are arranged in two opposite 'crescent moon' shapes, with some overlap [17]. COIL is an image recognition task comprising 100 objects taken from different angles [48]. BCI, instead, is a medical task involving electroencephalography recordings from 400 independent trials [49]. For further information about the datasets, we refer to [16] for 2Moons, to [15], [49] for BCI, and to [17] for the rest of the datasets. Additional information can be found in [15, Chapter 21]. The COIL dataset is used in two different versions, one with 2 classes (COIL2) and a harder version with 20 classes (COIL20). In all the cases, input variables are normalized between -1 and 1 before the experiments.

In our experimental setup we considered a 7-nodes network, whose topology is kept fixed for all the experiments. The topology is generated according to the so-called "Erdős-Rényi model" [50], such that each pair of agents is connected with a probability c. In particular, in our implementation we set $c = 0.5$, while we choose the weights matrix $\mathbf{C}$ using the so-called 'max-degree' strategy [41]:

$$C_{kj} = \begin{cases} \frac{1}{d+1} & \text{if } k \in \mathcal{N}_j \backslash k \\ 1 - \frac{d_k}{d+1} & \text{if } k = j \\ 0 & \text{otherwise} \end{cases} . \tag{37}$$

This choice ensures both convergence of the DAC protocol [41] and it satisfies the requirements of the DA framework [34]. All the experiments are repeated 25 times, to average possible outliers results due to the randomness in the processes of exchange and in the initialization of the matrices in the EDM completion algorithms. Following a standard SSL setting, at every run data are randomly shuffled and then partitioned in a labeled training set TR, a test set TST, and an unlabeled set U, whose cardinalities are reported in Tab. II. In order to simulate a distributed scenario, both the labeled and unlabeled training sets are then partitioned uniformly across the nodes. Since in this paper we are not interested in the analysis of the communication process over a real network, we implement a serial version of the code to perform the simulations. All the experiments are performed using MATLAB R2014a on an Intel i7-3820 @3.6 GHz and 32 GB of memory. Open-source MATLAB code for repeating the simulations is available on the web.[1]

B. Distributed Laplacian estimation

In this section we compare the performance of the two strategies for distributed EDM completion illustrated in Section III. We analyze the matrix completion error, together with the overall computational time for the two strategies. Given an estimate $\tilde{\mathbf{D}}$ of $\mathbf{D}$, we define the matrix completion error as:

$$E(\tilde{\mathbf{D}}) = \frac{\left\| \tilde{\mathbf{D}} - \mathbf{D} \right\|_F}{\left\| \mathbf{D} \right\|_F} . \tag{38}$$

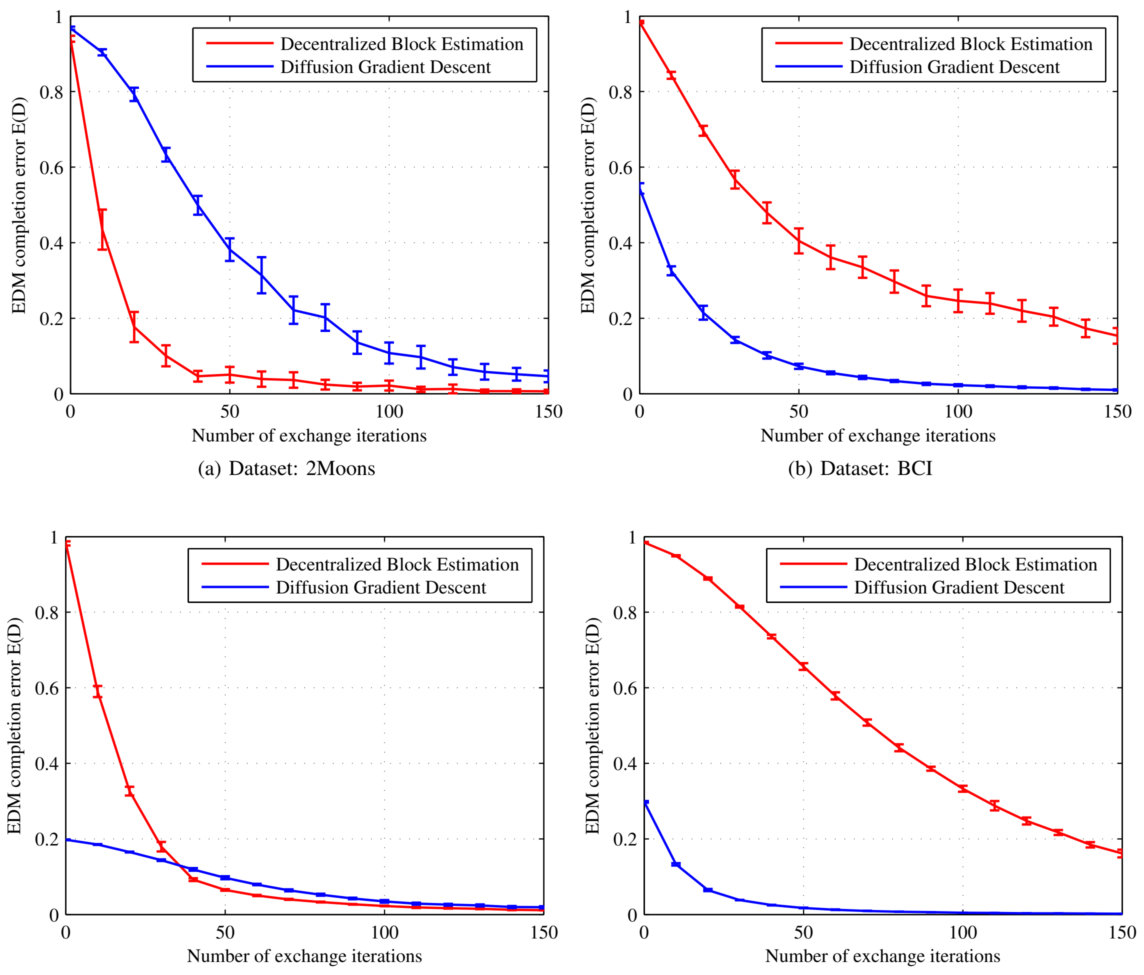
[1]https://bitbucket.org/robertofierimonte/distributed-semisupervised-code

The first set of experiments consists in comparing the completion error and the time required by the two algorithms, for different sizes of the sampling set of $\mathbf{D}$. In our context, the size of the sampling set depends only on the amount of data that are exchanged before the algorithm runs. To this end, we consider the completion error when varying the number of iterations for both the patterns exchange and the entries exchange steps, while keeping fixed the exchange fraction p, as defined in Section III-A. In particular, for all the datasets we varied the maximum number of iterations $n_{\max}^{(1)}$ and $n_{\max}^{(2)}$ from 0 to 150, by steps of 10. Results of this experiment are presented in Fig. 2. The solid red and the solid blue lines show the performance of Decentralized Block Estimation and Diffusion Grandient Descent, respectively. Since the value of the completion error only depends on the input $\mathbf{x}$, the results for datasets COIL20 and COIL2 are reported together.

The values for the patterns exchange fraction p1 and the entries exchange fraction p2 are chosen to balance the communication overhead and the size of the sampling set of $\mathbf{D}$. For both the algorithms, we set the maximum number of iterations $n_{\max}^{\text{EDM}}$to 1500, and we used a fixed step-size strategy. In particular for the Decentralized Block Estimation we set $\alpha = 0.4$, as suggested in [33], while for the Diffused Gradient Descent, the optimal values for η are chosen singularly for each dataset by searching in the interval 10^j, $j \in \{-10, \dots, -3\}$. These parameters, together with the values for p1 and p2, are reported in Tab. III, and are used in all the experiments.

First of all, we see that without a prior exchange of patterns, the EDM completion error is generally large. Although this is to be expected, it underlines the importance of having a data-exchange phase in the beginning of the algorithm. Secondly, we can see that with the solely exception of the 2Moons dataset (see Fig. 2a), the novel Diffused Gradient Descent algorithm achieves better performance when compared to the Decentralized Block Estimation, in particular when relatively few information is exchanged before the completion process. For three out of four datasets, as the number of the exchange iterations increases, the diffusion strategy is able to converge rapidly to the real EDM $\mathbf{D}$, while the performance is poorer for the block partitioning strategy, resulting for datasets BCI and COIL in a completion error of 19% even for high quantity of information exchanged (see Fig. 2b and Fig. 2d). Despite a small variability in the experimental results, we see that the error obtained after a small number of data exchanges is generally acceptable (i.e. under 0.1) in all cases. Still, the block estimation strategy can be a better choice for a small number of features (such is the case of the 2Moons dataset), since the combination of the Schoenberg mapping with the low-rank factorization may result in degraded performance.

When considering the time required by the two algorithms, which is shown in Fig. 3, we observe that the block partition strategy requires for datasets 2Moons and G50C less than half the time required by the diffused strategy, while, as the number of the features increases, the diffusion strategy tends to be less computational expensive. In fact, the time required by both strategies is nearly the same for the dataset BCI, while for COIL the diffusion strategy is 1.2 times faster. We remark that the Decentralized Block Estimation requires an



(a) Dataset: 2Moons

(b) Dataset: BCI

(c) Dataset: G50C

(d) Datasets: COIL20\COIL2

Fig. 2. Average EDM completion error of the two strategies on the considered datasets, when varying the number of iterations for the patterns and entries exchange protocols. The vertical bars represent the standard deviation from the average.

additional step for all the agents to gather the columns-wise blocks through the network, which has not been taken into account in calculating the computational time.

C. *Distributed semi-supervised manifold regularization*

The second experiment analyzes the performance of the distributed algorithm when compared to a centralized learning strategy and to a local learning strategy. We compare the following algorithms:

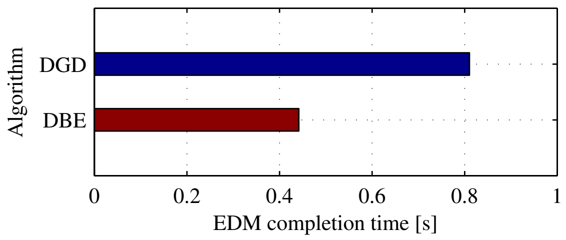
- **Centr-LapRLS**: this is the algorithm depicted in Sec. II-A. It is equivalent to a single agent collecting all the training data.
- **Local-LapRLS**: in the local setting, the training set is distributed across the agents and every agent trains a LapRLS on its own dataset, without any communication with other agents. The error is averaged throughout the nodes.
- **Distr-LapRLS**: as before, the training set is distributed within the network, but the agents converge to a centralized solution using the strategy detailed in Sec. IV. In this experiment, the EDM completion is achieved by the Diffusion Gradient Descent algorithm.

For all the algorithms, we build the Laplacian and the kernel matrices according to the method detailed in [17], using the parameters reported in Tab. III. In particular the parameters for datasets G50C and COIL come from [17], while those for 2Moons and BCI come from [16] and [15], respectively. Lower values for the exchange iterations in datasets 2Moons and BCI are chosen to balance the higher values for the exchange fractions.

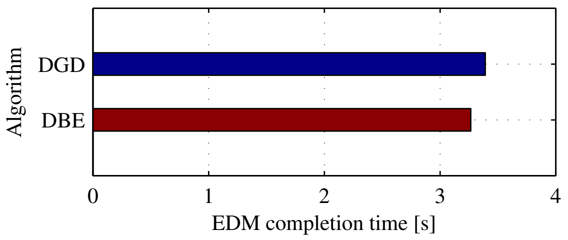
The classification error and the computational time for the three models over the five datasets are reported in Table IV. Results of the proposed algorithm, Distr-LapRLS, are highlighted in bold. We can see that Distr-LapRLS is generally able to match the same performance of the Centr-LapRLS, both in mean and variance, except for a small decrease in the G50C dataset. The performance of Local-LapRLS is noticeably worse than the other two algorithms, since the local models are built on considerably smaller training sets. The final classification accuracy varies from one dataset to the other, however, the results are generally in line with the

TABLE III
VALUES FOR THE PARAMETERS USED IN THE SIMULATIONS. THE VALUES IN THE FIRST GROUP ARE USED IN THE DISTRIBUTED PROTOCOLS AND IN THE DIFFUSION GRADIENT DESCENT ALGORITHM. THOSE IN THE SECOND GROUP ARE USED TO BUILD THE LAPLACIAN AND KERNEL MATRICES. IN THE THIRD GROUP ARE REPORTED THE PARAMETERS USED IN THE PRIVACY-PRESERVING TRANSFORMATIONS.

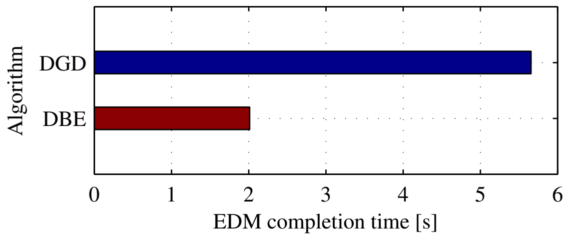
Dataset	p1 [%]	$n_{\max}^{(1)}$	p2 [%]	$n_{\max}^{(2)}$	η	γ_A	γ_I	nn	σ_K	q	t	σ_a	σ_b	σ_Q	σ_C
2Moons	3.5	100	3.5	100	10^{-3}	2^{-5}	4	6	0.035	1	–	–	–	–	–
BCI	2.5	100	2.5	100	10^{-6}	10^{-6}	1	5	1	2	10^4	0	0	1	10^{-6}
G50C	2	150	2.5	150	10^{-6}	10^{-6}	10^{-2}	50	17.5	5	$2 \cdot 10^4$	0	1	1	$1.1 \cdot 10^{-6}$
COIL	2	150	2.5	150	10^{-7}	10^{-6}	1	2	0.6	1	10^3	0	0	1	10^{-6}



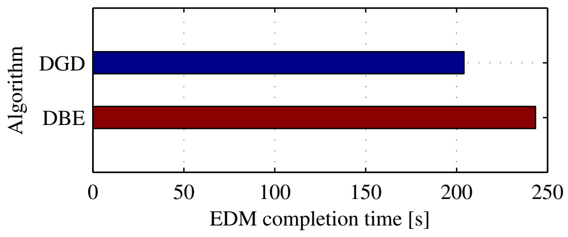
(a) Dataset: 2Moons



(b) Dataset: BCI



(c) Dataset: G50C



(d) Datasets: COIL20\COIL2

Fig. 3. Average EDM completion time required by the two strategies on the considered datasets. DGD and DBE are the acronyms for Decentralized Block Estimation and Diffusion Gradient Descent respectively.

current state-of-the-art in SSL [15], [17].

The computational time required by the distributed algorithm is given by the sum of the time required by both the exchange protocols, the distributed Laplacian estimation, the DAC protocol, and the matrix inversion in (32). When comparing the results with the values for EDM completion time obtained in the previous experiment, we notice that the order of magnitude of the time required by Distr-LapRLS is given by the time necessary to complete the distance matrix. The final column of Table IV also highlight the main limitation of our current approach, namely, the time requested from very large datasets can be prohibitive in certain low-power applications. Partly, this is due to a simple implementation of the two optimization strategies, both of which can be improved by more advanced choices of their internal step-sizes, and other techniques. Some of these future directions are discussed in the conclusive section.

D. Privacy preservation

As a final experiment, we include in our algorithm the two privacy-preserving strategies presented in Sec. II-C. In particular, we analyze the evolution of the classification error when varying the ratio m/d from 0.1 to 0.95, i.e. when varying the dimensionality m of the transformed patterns. In this experiment we do not consider the 2Moons dataset, because of its limited number of features.

Since the value of σ in the linear random projection has no influence on the error of the transformed patterns, we set $\sigma = 1$ for all the datasets. As for the nonlinear transformation, the values for the parameters are searched inside a grid and then optimized locally. Possible values for t are searched in 10^i, $i = \{1, \ldots, 5\}$, while values for the variances are searched in 10^j, $j = \{-6, \ldots, 6\}$. The optimal values for the datasets are reported in the third group of Table III.

Results of the experiment are presented in Fig. 4. The classification error for the linear random projection and nonlinear transformation are shown with solid red and dashed blue lines, respectively. In addition, the mean value for Distr-LapRLS (together with its confidence interval) is reported as a baseline, shown with a dashed black line.

By observing the results, we can see that when compared to Distr-LapRLS, the privacy-preserving strategies show different behaviors depending on the dataset. In particular, for dataset BCI, the error is nearly the same of Distr-LapRLS, while it is slightly lower for COIL2 and COIL20, and somewhat higher for G50C, where it shows a decreasing trend. For all the datasets, we see that the error achieved using the privacy-preserving strategies remains inside the limits of Distr-LapRLS

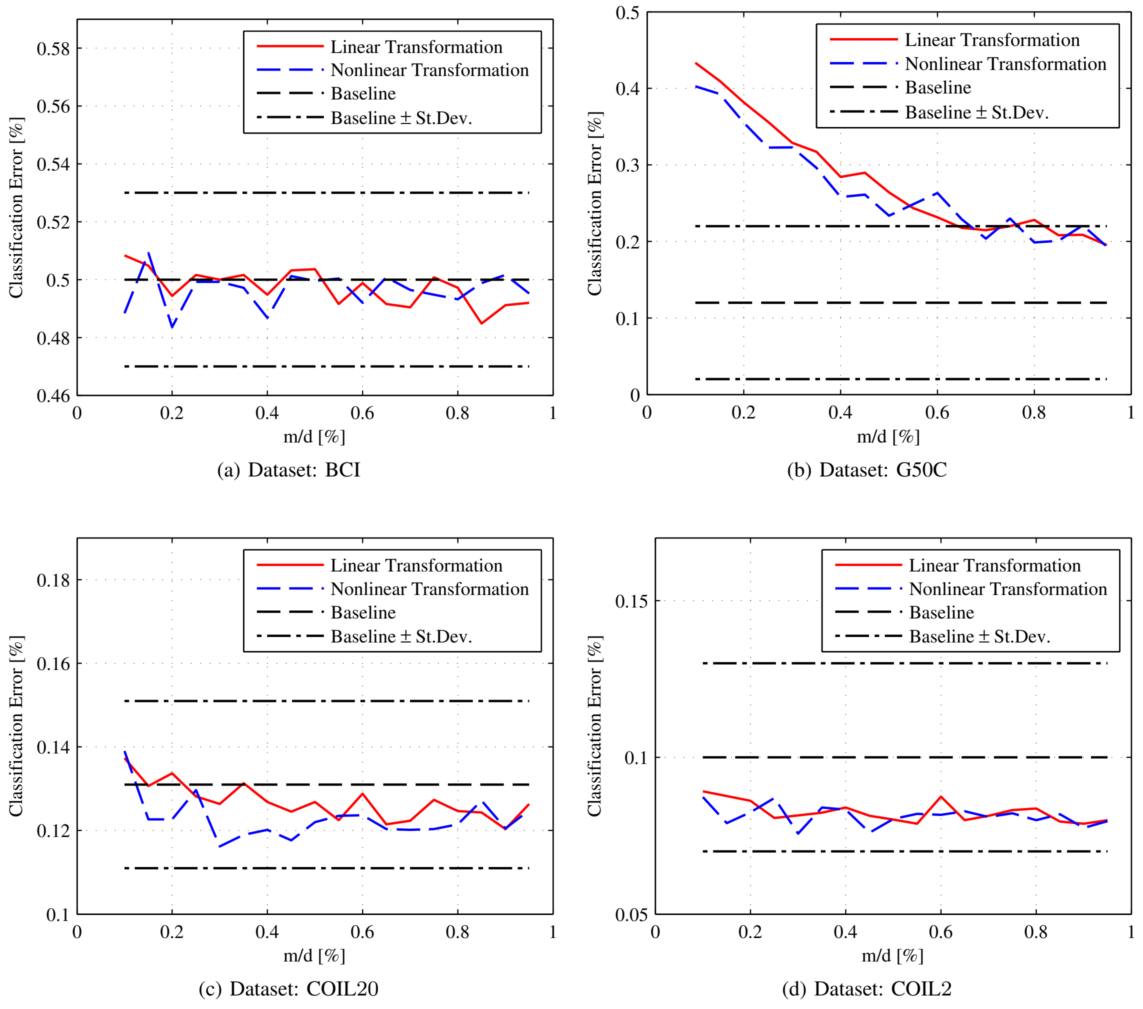


Fig. 4. Average classification error of the privacy-preserving transformation on the considered datasets when varying the ratio m/d.

error's confidence interval, denoting how the variability introduced does not have significant influence on the algorithm's performance.

We notice that in most cases, we can obtain a comparable or even better performance with respect to the privacy-free algorithm, with significantly fewer features, leading to a reduction of the information exchanged and therefore of the overall computational requirements. For all the datasets, both the transformations present a non-smooth trend, caused by the heuristic nature of these methods. Moreover, the error is very similar between the strategies, suggesting that the use of a nonlinear transformation, potentially safer than a linear one, does not influence the performance.

VI. Conclusions

In this paper, we have proposed a totally decentralized algorithm for semi-supervised learning in the framework of MR. The core of our proposal is constituted by a distributed protocol designed to compute the Laplacian matrix. Our experimental results show that, in most cases, the performance of the novel diffusion adaptation-based algorithm for distributed EDM completion overcome those of the state-of-the-art column-wise partitioning strategy. Secondly, experiments show that the distributed LapRLS is competitive with an ideal centralized LapRLS model trained on the overall dataset, i.e., an agent solving the global optimization problem of the algorithm in a centralized fashion.

Although we have focused here on a particular algorithm belonging to MR, namely Lap-RLS, the framework is easily applicable to additional ones, including the laplacian Support Vector Machine (LapSVM) [16], and others. Moreover, extensions beyond MR are possible, i.e. to all the methods that encode information in the form of a matrix of pairwise distances, such as spectral dimensionality reduction, spectral clustering, and so on. In the case of kernels that directly depend on the dot product between patterns (e.g. the polynomial one), particular care must be taken in designing appropriate privacy-preserving protocols for distributed margin computation [51], an aspect which is left to future investigations.

Currently, the main limit of our algorithm is the computation time required by the distributed algorithm for completing the Laplacian matrix. This is due to a basic implementation of the two optimization algorithms. In this sense, in future works we intend to improve the distributed algorithm to achieve better

TABLE IV
AVERAGE VALUES FOR CLASSIFICATION ERROR AND COMPUTATIONAL TIME, TOGETHER WITH STANDARD DEVIATION, FOR THE THREE ALGORITHMS. RESULTS FOR THE PROPOSED ALGORITHM ARE HIGHLIGHTED IN BOLD.

Dataset	Algorithm	Error [%]	Time [s]
2Moons	Centr-LapRLS	0.008 ± 0.001	0.006 ± 0.015
	Distr-LapRLS	$\mathbf{0.01 \pm 0.03}$	$\mathbf{0.875 \pm 0.030}$
	Local-LapRLS	0.41 ± 0.28	0.000 ± 0.000
BCI	Centr-LapRLS	0.49 ± 0.04	0.021 ± 0.012
	Distr-LapRLS	$\mathbf{0.49 \pm 0.05}$	$\mathbf{3.396 \pm 0.028}$
	Local-LapRLS	0.54 ± 0.14	0.001 ± 0.000
G50C	Centr-LapRLS	0.07 ± 0.02	0.101 ± 0.017
	Distr-LapRLS	$\mathbf{0.12 \pm 0.10}$	$\mathbf{5.764 \pm 0.066}$
	Local-LapRLS	0.45 ± 0.06	0.001 ± 0.000
COIL20	Centr-LapRLS	0.13 ± 0.02	1.565 ± 0.019
	Distr-LapRLS	$\mathbf{0.13 \pm 0.02}$	$\mathbf{195.933 \pm 2.176}$
	Local-LapRLS	0.78 ± 0.07	0.056 ± 0.001
COIL2	Centr-LapRLS	0.10 ± 0.03	1.556 ± 0.028
	Distr-LapRLS	$\mathbf{0.10 \pm 0.03}$	$\mathbf{191.478 \pm 0.864}$
	Local-LapRLS	0.43 ± 0.12	0.055 ± 0.000

computational performance. Examples of possible modifications include adaptive strategies for the choice of the step-size, as well as early stopping protocols.

REFERENCES

[1] R. Wolff, K. Bhaduri, and H. Karguta, "A generic local algorithm for mining data streams in large distributed systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, pp. 465–478, 2009.

[2] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed learning for Random Vector Functional-Link networks," *Information Sciences*, vol. 301, pp. 271–284, 2015.

[3] P. Di Lorenzo and A. H. Sayed, "Sparse distributed learning based on diffusion adaptation," *IEEE Transactions on Signal Processing*, vol. 61, no. 6, pp. 1419–1433, 2013.

[4] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Processing Magazine*, pp. 56–69, 2007.

[5] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, "Distributed data mining in peer-to-peer networks," *IEEE Internet Computing*, vol. 10, no. 4, pp. 18–26, 2006.

[6] H. H. Ang, V. Gopalkrishnan, S. C. Hoi, and W. K. Ng, "Classification in P2P networks with cascade support vector machines," *ACM Transactions on Knowledge Discovery from Data*, vol. 7, no. 4, p. 20, 2013.

[7] A. Lazarevic and Z. Obradovic, "The distributed boosting algorithm," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 311–316.

[8] L. Georgopoulos and M. Hasler, "Distributed machine learning in networks by consensus," *Neurocomputing*, vol. 124, pp. 2–12, Jan. 2014.

[9] A. Navia-Vázquez, D. Gutierrez-Gonzalez, E. Parrado-Hernández, and J. J. Navarro-Abellan, "Distributed support vector machines," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1091–1097, 2006.

[10] Y. Lu, V. Roychowdhury, and L. Vandenberghe, "Distributed parallel support vector machines in strongly connected networks," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1167–1178, 2008.

[11] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *Journal of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[12] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5262–5276, 2010.

[13] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 593–599.

[14] J. Chen, Z. J. Towfic, and A. H. Sayed, "Dictionary Learning Over Distributed Models," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 1001–1016, Feb 2015.

[15] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised learning*. The MIT Press, 2006.

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

[17] S. Melacci and M. Belkin, "Laplacian support vector machines trained in the primal," *Journal of Machine Learning Research*, vol. 12, pp. 1149–1184, 2011.

[18] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Optimization techniques for semi-supervised support vector machines," *Journal of Machine Learning Research*, vol. 9, pp. 203–233, 2008.

[19] J. Chen, C. Wang, Y. Sun, and X. S. Shen, "Semi-supervised Laplacian regularized least squares algorithm for localization in wireless sensor networks," *Computer Networks*, vol. 55, no. 10, pp. 2481–2491, 2011.

[20] M. Torii, K. Wagholikar, and H. Liu, "Using machine learning for concept extraction on clinical documents from multiple data sources," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 580–587, 2011.

[21] S. Scardapane, R. Fierimonte, D. Wang, M. Panella, and A. Uncini, "Distributed music classification using random vector functional-link nets," in *2015 International Joint Conference on Neural Networks (IJCNN'15)*. IEEE, 2015, pp. 1–8.

[22] O. Obst, "Distributed fault detection in sensor networks using a recurrent neural network," *Neural Processing Letters*, vol. 40, no. 3, pp. 261–273, 2014.

[23] S. Scardapane, W. Dianhui, and M. Panella, "A decentralized training algorithm for echo state networks in distributed big data applications," *Neural Networks*, 2015, under press.

[24] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis, "State-of-the-art in privacy preserving data mining," *ACM Sigmod Record*, vol. 33, no. 1, pp. 50–57, 2004.

[25] M. Belkin and P. Niyogi, "Semi-supervised learning on Riemannian manifolds," *Machine learning*, vol. 56, no. 1-3, pp. 209–239, 2004.

[26] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The annals of statistics*, pp. 1171–1220, 2008.

[27] B. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Computing Surveys (CSUR)*, vol. 42, no. 4, p. 14, 2010.

[28] S. Al-Homidan and H. Wolkowicz, "Approximate and exact completion problems for Euclidean distance matrices using semidefinite programming," *Linear algebra and its applications*, vol. 406, pp. 109–141, 2005.

[29] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[30] E. J. Candes and Y. Plan, "Matrix completion with noise," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 925–936, 2010.

[31] B. Mishra, G. Meyer, and R. Sepulchre, "Low-rank optimization for distance matrix completion," in *2011 50th IEEE conference on Decision and control and European control conference (CDC-ECC'11)*. IEEE, 2011, pp. 4455–4460.

[32] Q. Ling, Y. Xu, W. Yin, and Z. Wen, "Decentralized low-rank matrix completion," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'12)*. IEEE, 2012, pp. 2925–2928.

[33] A. Y. Lin and Q. Ling, "Decentralized and privacy-preserving low-rank matrix completion," *Journal of the Operations Research Society of China*, vol. 3, no. 2, pp. 189–205, 2015.

[34] A. H. Sayed, "Adaptive networks," *Proceedings of the IEEE*, vol. 102, no. 4, pp. 460–497, 2014.

[35] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[36] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Distributed detection and estimation in wireless sensor networks," in *E-Reference Signal Processing*, R. Chellapa and S. Theodoridis, Eds. Elsevier, 2013, pp. 329–408.

[37] A. Y. Alfakih, A. Khandani, and H. Wolkowicz, "Solving Euclidean distance matrix completion problems via semidefinite programming," *Computational optimization and applications*, vol. 12, no. 1-3, pp. 13–30, 1999.

[38] K. Liu, H. Kargupta, and J. Ryan, "Random projection-based multiplicative data perturbation for privacy preserving distributed data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 92–106, 2006.

[39] K. Bhaduri, M. D. Stefanski, and A. N. Srivastava, "Privacy-preserving outlier detection through random nonlinear data distortion," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 41, no. 1, pp. 260–272, 2011.

[40] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, "Tools for privacy preserving distributed data mining," *ACM SiGKDD Explorations Newsletter*, vol. 4, no. 2, pp. 28–34, 2002.

[41] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[42] C.-B. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural computation*, vol. 19, no. 10, pp. 2756–2779, 2007.

[43] C. G. Lopes and A. H. Sayed, "Diffusion least-mean squares over adaptive networks: Formulation and performance analysis," *IEEE Transactions on Signal Processing*, vol. 56, no. 7, pp. 3122–3136, 2008.

[44] P. Di Lorenzo and S. Barbarossa, "Distributed estimation and control of algebraic connectivity over random graphs," *IEEE Transactions on Signal Processing*, vol. 62, no. 21, pp. 5615–5628, 2014.

[45] X. Zhao and A. H. Sayed, "Asynchronous adaptation and learning over networks - part i: Modeling and stability analysis," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 811–826, Feb 2015.

[46] A. H. Sayed, "Adaptation, learning, and optimization over networks," *Foundations and Trends® in Machine Learning*, vol. 7, no. 4-5, pp. 311–801, 2014.

[47] P. Bianchi and J. Jakubowicz, "Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization," *IEEE Transactions on Automatic Control*, vol. 58, no. 2, pp. 391–405, 2013.

[48] S. A. Nene, S. K. Nayar, H. Murase *et al.*, "Columbia object image library (coil-20)," Technical Report CUCS-005-96, Tech. Rep., 1996.

[49] T. N. Lal, M. Schröder, T. Hinterberger, J. Weston, M. Bogdan, N. Birbaumer, and B. Schölkopf, "Support vector channel selection in bci," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 1003–1010, 2004.

[50] M. Newman, *Networks: an introduction*. Oxford University Press, 2010.

[51] Q. Shi, C. Shen, R. Hill, and A. Hengel, "Is margin preserved after random projection?" in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*. ACM, 2012, pp. 591–598.