Neural Networks 80 (2016) 43–52

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

# Distributed semi-supervised support vector machines

Simone Scardapane[a,*], Roberto Fierimonte[a], Paolo Di Lorenzo[b], Massimo Panella[a], Aurelio Uncini[a]

[a] *Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184 Rome, Italy*

[b] *Department of Engineering, University of Perugia, Via G. Duranti 93, 06125, Perugia, Italy*

ARTICLE INFO

*Article history:*
Received 23 October 2015
Received in revised form 12 April 2016
Accepted 17 April 2016
Available online 27 April 2016

*Keywords:*
Semi-supervised learning
Support vector machine
Distributed learning
Networks

ABSTRACT

The semi-supervised support vector machine ($S^3VM$) is a well-known algorithm for performing semi-supervised inference under the large margin principle. In this paper, we are interested in the problem of training a $S^3VM$ when the labeled and unlabeled samples are distributed over a network of interconnected agents. In particular, the aim is to design a distributed training protocol over networks, where communication is restricted only to neighboring agents and no coordinating authority is present. Using a standard relaxation of the original $S^3VM$, we formulate the training problem as the distributed minimization of a non-convex social cost function. To find a (stationary) solution in a distributed manner, we employ two different strategies: (i) a distributed gradient descent algorithm; (ii) a recently developed framework for In-Network Nonconvex Optimization (NEXT), which is based on successive convexifications of the original problem, interleaved by state diffusion steps. Our experimental results show that the proposed distributed algorithms have comparable performance with respect to a centralized implementation, while highlighting the pros and cons of the proposed solutions. To the date, this is the first work that paves the way toward the broad field of distributed semi-supervised learning over networks.

## 1. Introduction

Semi-supervised learning (SSL) algorithms are a family of techniques for performing inference in the presence of both labeled and unlabeled data (Chapelle, Schölkopf, & Zien, 2006). Among them, in the binary classification setting the semi-supervised support vector machine ($S^3VM$) has attracted a large amount of attention over the last decades (Chapelle, Sindhwani, & Keerthi, 2008). The $S^3VM$ is based on the idea of minimizing the training error and maximizing the margin over both labeled and unlabeled data, whose labels are included as additional variables in the optimization problem. Since its first practical implementation in Joachims (1999), inspired by previous work on transductive learning by Vapnik (1998), numerous researchers have proposed alternative solutions for solving the resulting mixed integer optimization problem, including branch and bound algorithms (Chapelle, Sindhwani, & Keerthi, 2006), convex relaxations (Chapelle & Zien, 2005; Li, Tsang, & Kwok, 2013), convex–concave procedures (Fung & Mangasarian, 2001), and others. It has been applied to a wide variety of practical problems, such as text inference (Joachims, 1999), and it has given birth to numerous other algorithms, including semi-supervised least-square SVMs (Adankon, Cheriet, & Biem, 2009), and semi-supervised random vector functional-link networks (Scardapane, Comminiello, Scarpiniti, & Uncini, in press).

In this paper, we are interested in designing algorithms for solving the $S^3VM$ optimization problem, in the case where the training data is distributed across a network of interconnected agents (Scardapane, Wang, Panella, & Uncini, 2015). In the fully supervised case, this is a well-known scenario, which has been investigated extensively in multiple research fields, including peer-to-peer (P2P) (Ang, Gopalkrishnan, Hoi, & Ng, 2013) and sensor networks (Barbarossa, Sardellitti,& Di Lorenzo, 2014; Predd, Kulkarni, & Poor, 2006), robotic swarms, and many others. In all of these settings, the underlying network of agents is generally unstructured, and no centralized authority can coordinate the overall process. Thus, distributed training algorithms are designed based on simple local exchanges of information among

* Corresponding author. Tel.: +39 06 44585495; fax: +39 06 4873300.
*E-mail addresses:* simone.scardapane@uniroma1.it (S. Scardapane), robertofierimonte@gmail.com (R. Fierimonte), paolo.dilorenzo@unipg.it (P. Di Lorenzo), massimo.panella@uniroma1.it (M. Panella), aurelio.uncini@uniroma1.it (A. Uncini).

http://dx.doi.org/10.1016/j.neunet.2016.04.007
0893-6080/© 2016 Elsevier Ltd. All rights reserved.

neighboring agents. A large number of decentralized algorithms have been developed for training a supervised SVM in such a distributed scenario (Forero, Cano, & Giannakis, 2010; Lu, Roychowdhury, & Vandenberghe, 2008; Navia-Vázquez, Gutierrez-Gonzalez, Parrado-Hernández, & Navarro-Abellan, 2006), and they are briefly summarized in the next section.

To the best of our knowledge, however, there is a lack of distributed training algorithms for the SSL case over networks. Indeed, this problem has been addressed only for specific cases, such as localization in a WSN (Chen, Wang, Sun, & Shen, 2011). Nonetheless, as we argue in Fierimonte, Scardapane, Uncini, and Panella (submitted for publication), there is a large number of realistic applications where the agents can benefit from the inclusion of additional unlabeled data in the training process. As an example, consider a distributed medical application, where multiple clinical institutions possess similar databases, but privacy concerns do not allow them to share it with a centralized institution (Clifton, Kantarcioglu, Vaidya, Lin, & Zhu, 2002). In this case, labeled data is generally scarce, while each institution has access to a large amount of unlabeled samples. Using currently available distributed algorithms, however, would imply discarding these unlabeled databases, resulting in a possible loss of generalization accuracy.

To simplify our derivation, in this paper, we focus on the *linear* $S^3$VM formulation, whose decision boundary corresponds to an hyperplane in the input space. It is known that non-linear decision boundaries can be obtained with the use of kernel functions. In that case, however, the resulting SVM model is expressed in terms of all examples, which in a decentralized setting are distributed among the different agents. This is a notoriously complex problem (Predd et al., 2006), which in many contexts hinders the applicability of the resulting algorithms. In an alternative publication (Fierimonte et al., submitted for publication), we have explored the problem of training a semi-supervised Laplacian SVM using a distributed computation of the underlying kernel matrix. However, the resulting algorithm requires a large amount of computational and/or communication resources. The algorithms presented in this paper, instead, can be implemented even on agents with stringent requirements in terms of power, such as sensors in a WSN. At the same time, limiting ourselves to a linear decision boundary can be reasonable, as the linear $S^3$VM can perform well in a large range of settings, due to the scarcity of labeled data (Chapelle et al., 2008).

Specifically, starting from the smooth approximation to the original $S^3$VM presented in Chapelle and Zien (2005), we show that the distributed training problem can be formulated as the joint minimization of a sum of non-convex cost functions. This is a complex problem, which has been investigated only very recently in the distributed optimization literature (Bianchi & Jakubowicz, 2013; Di Lorenzo & Scutari, in press). In our case, we build on two different solutions. The first one is based on the idea of diffusion gradient descent (DGD) (Di Lorenzo & Sayed, 2013; Sayed, 2014a, 2014b), arisen from previous work in the context of distributed filtering applications (Lopes & Sayed, 2008). The main idea of DGD is to interleave gradient descent steps at every node, with local averaging of the estimates. The resulting algorithm leads to an extremely efficient implementation. Nevertheless, since it is a gradient-based algorithm exploiting only first order information of the objective function, it generally suffers from slow practical convergence speed, especially in the case of non-convex and large-scale optimization problems. Recently, it was showed in Di Lorenzo and Scutari (in press), Facchinei, Scutari, and Sagratella (2015) and Scutari, Facchinei, Song, Palomar, and Pang (2014) that exploiting the structure of nonconvex functions by replacing their linearization (i.e., their gradient) with a "better" approximant can enhance practical convergence speed. Thus, we propose a distributed algorithm based on the recently proposed In-Network Successive Convex Approximation (NEXT) framework (Di Lorenzo & Scutari, in press). The method hinges on successive convex approximation techniques while leveraging dynamic consensus (Zhu & Martínez, 2010) as a mechanism to distribute the computation among the agents as well as diffuse the needed information over the network. Both algorithms are provably convergent to stationary points of the non-convex optimization problem. Moreover, as shown in our experimental results, NEXT exhibits a faster practical convergence speed with respect to DGD, which is paid by a larger computation cost per iteration.

To summarize, our main contributions with respect to the current literature on distributed learning are two-fold. Firstly, to the best of our knowledge, this is the first work dealing explicitly with (fully) distributed implementations of semi-supervised routines and, more specifically, semi-supervised SVMs, paving the way to a large number of possible domains which can benefit from the availability of these techniques. Additionally, the present work is one of the first successful applications of optimization protocols explicitly designed for distributed *non-convex* costs, while the majority of works on distributed learning has focused on models giving rise to convex optimization problems.

The rest of the paper is structured as follows. Section 2 goes briefly over previous works on distributed SVMs in the fully supervised case. Then, Section 3 introduces the $S^3$VM model together with the approximation presented in Chapelle and Zien (2005). In Section 4, we first formulate the distributed training problem for $S^3$VMs, and subsequently we derive our two proposed solutions. Then, Section 5 details an extensive set of experimental results and, finally, Section 6 concludes the paper.

*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 column vectors. Symbol $a_i$ denotes the $i$th 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. Other notation is introduced in the text when appropriate.

## 2. Related works

We start by briefly reviewing some works on distributed SVM algorithms in the fully supervised case. Similar overviews can be found in Scardapane, Wang, and Panella (in press, Section 2.1) and Wang and Zhou (2012). Initial works in this field were sparked by realizing that the set of support vectors represents an efficient way of 'compressing' data to be sent among the neighbors. In practice, this is complicated by the fact that each agent has no principled way of knowing whether a specific example is a support vector of the full problem. Thus, in Navia-Vázquez et al. (2006) the real set of support vectors is approximated by a specific set chosen *a priori*, whose weights are updated based on a least-square procedure. On the contrary, Lu et al. (2008) solve the problem considering the real set of support vectors, with assured convergence in a finite number of steps. Both approaches, however, are hindered by the necessity of sending the examples throughout the network on a Hamiltonian cycle. The most efficient procedure up-to-date is presented in Forero et al. (2010), where the problem is recast as multiple convex subproblems at every node, and solved with the use of the alternating direction method of multipliers (ADMM), an efficient procedure for distributed optimization of convex cost functions. Indeed, ADMM is among the most widely used methods for distributed learning of neural-like architectures, and it has been used successfully for linear models (Mateos, Bazerque, & Giannakis, 2010), random-vector functional links networks (Scardapane et al., 2015), recurrent neural networks (Scardapane, Wang et al., in press) and others (see also the survey in Boyd, Parikh, Chu, Peleato, & Eckstein, 2011).

It is worth noting that alternative approaches exploiting specific features of WSN and P2P networks were investigated in Ang et al. (2013), Beferull-Lozano and Tsakalides (2006) and Hensel and Dutta (2009). Additionally, for linear SVMs there has also been interest in applying other routines from the distributed optimization field. Among these, we can cite the random projection algorithm (Lee & Nedic, 2013), dual coordinate ascent (Jaggi et al., 2014), and the box-constrained QP (Lee & Roth, 2015).

## 3. Semi-supervised support vector machines

Let us consider the standard SSL problem, where we are interested in learning a binary classifier starting from $L$ labeled samples $(\mathbf{x}_i, y_i)_{i=1}^L$ and $U$ unlabeled samples $\left(\mathbf{x}'_i\right)_{i=1}^U$. Each input is a $d$-dimensional real vector $\mathbf{x}_i \in \mathbb{R}^d$, while each output can only take one of two possible values $y_i \in \{-1, +1\}$. While SVMs are specifically designed for binary problems, it is known that any multi-class problem can be transformed into a set of binary classification problems using standard techniques, e.g. one-versus-all or all-versus-all (Rifkin & Klautau, 2004). These techniques are also straightforward to apply in the distributed scenario, since the different binary problems can be solved in parallel (Graf, Cosatto, Bottou, Dourdanovic, & Vapnik, 2004).[1] The linear S$^3$VM optimization problem can be formulated as (Chapelle & Zien, 2005)

$$\min_{\mathbf{w},b,\hat{\mathbf{y}}} \frac{C_1}{2L}\sum_{i=1}^{L} l\left(y_i, f(\mathbf{x}_i)\right) + \frac{C_2}{2U}\sum_{i=1}^{U} l(\hat{y}_i, f\left(\mathbf{x}'_i\right)) + \frac{1}{2}\|\mathbf{w}\|_2^2, \quad (1)$$

where $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, $\hat{\mathbf{y}} \in \{-1, +1\}^U$ is a vector of unknown labels, $l(\cdot, \cdot)$ is a proper loss function, $C_1, C_2 > 0$ are coefficients weighting the relative importance of labeled and unlabeled samples, and $\|\mathbf{w}\|_2^2$ is the standard regularization term (Kurková, 2005). The main difference with respect to the classical SVM formulation is the inclusion of the unknown labels $\hat{\mathbf{y}}$ as variables of the optimization problem. This makes Problem (1) a mixed integer optimization problem, whose exact solution can be computed only for relatively small datasets, e.g. using standard branch-and-bound algorithms (Chapelle, Sindhwani et al., 2006). We note that, for $C_2 = 0$, we recover the standard SVM formulation. The most common choice for the loss function is the hinge loss, given by

$$l\left(y, f(\mathbf{x})\right) = \max\left(0, 1 - yf(\mathbf{x})\right)^p, \quad (2)$$

where $p \in \mathbb{N}$. In this paper, we use the choice $p = 2$, which leads to a smooth and convex function. Additionally, it is standard practice to introduce an additional constraint in the optimization problem, so that the resulting vector $\hat{\mathbf{y}}$ has a fixed proportion $r \in [0, 1]$ of positive labels:

$$\frac{1}{U}\sum_{i=1}^{U} \max\left(0, \hat{y}_i\right) = r. \quad (3)$$

This constraint helps achieve a balanced solution, especially when the ratio $r$ reflects the true proportion of positive labels in the underlying dataset.

[1] For the interested reader, a limited number of works considered distributed multiclass versions of the SVM, e.g. Lodi, Aanculef, and Sartori (2010). While they can represent an interesting point for further research, their applicability in SSL scenarios is currently very limited.

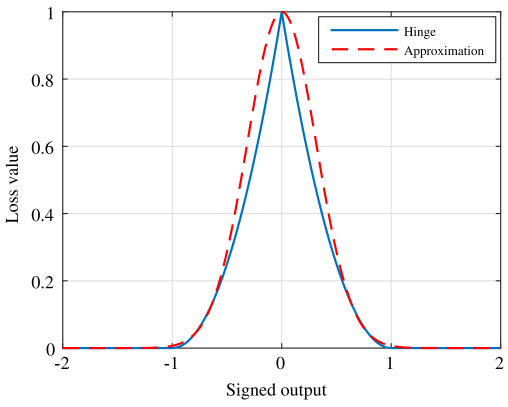


**Fig. 1.** For a fixed choice of $\mathbf{w}$ and $b$, $\max\left(0, 1 - \hat{y}_i f(\mathbf{x}'_i)\right)^2 = \max\left(0, 1 - |f(\mathbf{x}'_i)|\right)^2$. This is shown in blue for varying values of $f(\mathbf{x}'_i)$, while in dashed red we show the approximation given by $\exp\left\{-5f(\mathbf{x}'_i)^2\right\}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

A common way of solving Problem (1) stems from the fact that, for a fixed $\mathbf{w}$ and $b$, the optimal $\hat{\mathbf{y}}$ is given in closed form by

$$\hat{y}_i = \text{sign}(\mathbf{w}^T\mathbf{x}'_i + b), \quad i = 1, \dots, U.$$

Exploiting this fact, it is possible to devise a *continuous* approximation of the cost function in (1) (Chapelle et al., 2008). In particular, to obtain a smooth optimization problem solvable by standard first-order methods, Chapelle and Zien (2005) propose to replace the hinge loss over the unknown labels with the approximation given by $\exp\left\{-sf(\mathbf{x})^2\right\}$, $s > 0$. In the following, we choose in particular $s = 5$, as suggested by Chapelle et al. (2008). A visual example of the approximation is illustrated in Fig. 1. The resulting $\nabla$ S$^3$VM optimization problem writes as

$$\min_{\mathbf{w},b} \frac{C_1}{2L}\sum_{i=1}^{L} l\left(y_i, f(\mathbf{x}_i)\right) + \frac{C_2}{2U}\sum_{i=1}^{U} \exp\left\{-sf(\mathbf{x}'_i)^2\right\} + \frac{1}{2}\|\mathbf{w}\|_2^2. \quad (4)$$

Problem (4) does not incorporate the constraint in (3) yet. A possible way to handle the balancing constraint in (3) is a relaxation that uses the following linear approximation (Chapelle & Zien, 2005):

$$\frac{1}{U}\sum_{i=1}^{U} \mathbf{w}^T\mathbf{x}'_i + b = 2r - 1, \quad (5)$$

which can easily be enforced for a fixed $r$ by first translating the unlabeled points so that their mean is $\mathbf{0}$, and then fixing the offset $b$ as $b = 2r - 1$. The resulting problem can then be solved using standard first-order procedures.

## 4. Distributed learning for S$^3$VM

In this section, we first formulate a distributed optimization problem for a $\nabla$S$^3$VM over a network of agents in Section 4.1. Then, we present two alternative methods for solving the overall optimization problem in a fully decentralized fashion in Sections 4.2 and 4.3.

### 4.1. Formulation of the problem

For the rest of this paper, we assume that labeled and unlabeled training samples are not available on a single processor. Instead, they are distributed over a network of $N$ agents. In particular, we assume that the $k$th node has access to $L_k$ labeled samples, and $U_k$ unlabeled ones, such that $\sum_{k=1}^N L_k = L$ and $\sum_{k=1}^N U_k = U$.

The network of the agents is modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, N\}$ is the vertex (i.e., agent) set, and $\mathcal{E}$ is the set of edges. The neighborhood of agent $k$ (excluding node $k$) is defined as $\mathcal{N}_k = \{t \mid (t,k) \in \mathcal{E}\}$; it sets the communication pattern between single-hop neighbors, i.e. agents in $\mathcal{N}_k$ can communicate with node $k$. We introduce the weights $C_{kt}$ matching the graph $\mathcal{G}$, i.e. $C_{kt} > 0$ if $t \in \mathcal{N}_k$ or $t = k$. We also define the matrix $\mathbf{C} \triangleq (C_{kt})_{k,t=1}^N$. Specific ways of choosing these weights will be discussed in the following. We make the following assumption on the network connectivity.

**Assumption 1.** The graph $\mathcal{G}$ is strongly connected. Furthermore, the weight matrix $\mathbf{C}$ satisfies $\mathbf{C}\,\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T\mathbf{C} = \mathbf{1}^T$.

Given the connectivity pattern among agents, we are interested in devising distributed solutions in the following setting: (i) agents know their local functions and data only; and (ii) only inter-node communications between single-hop neighbors are possible. This setting is particularly important in all the applications where data are naturally distributed over the network, and sharing local information with a central processor is either unfeasible or not economical/efficient, owing to the large size of the network and volume of data, time-varying network topology, energy constraints, and/or privacy issues.

Following the rationale introduced in Section 3, the distributed $\nabla$ S$^3$VM problem can be cast as

$$\min_{\mathbf{w}} \sum_{k=1}^N l_k(\mathbf{w}) + \sum_{k=1}^N g_k(\mathbf{w}) + r(\mathbf{w}), \tag{6}$$

where we have defined the following shorthands:

$$l_k(\mathbf{w}) = \frac{C_1}{2L}\sum_{i=1}^{L_k} l\left(y_{k,i}, f(\mathbf{x}_{k,i})\right), \tag{7}$$

$$g_k(\mathbf{w}) = \frac{C_2}{2U}\sum_{i=1}^{U_k} \exp\left\{-sf(\mathbf{x}'_{k,i})^2\right\}, \tag{8}$$

$$r(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 . \tag{9}$$

In the previous equations, we use the double subscript $(k,i)$ to denote the $i$th sample available at the $k$th node, and we assume that the bias $b$ has been fixed *a priori* using the strategy detailed in the previous section. In a distributed setting, this requires that each agent knows the mean of all unlabeled points given by $\frac{1}{U}\sum_{i=1}^U \mathbf{x}'_i$. This can easily be achieved, before starting the training process, with a number of different in-network algorithms. For example, the agents can compute the average using a standard consensus procedure (Barbarossa et al., 2014), push-sum protocols (Hensel & Dutta, 2009) in a P2P network, or a number of alternative techniques.

### 4.2. Solution 1: distributed gradient descent

The first solution is based on the DGD procedure (Sayed, 2014b), i.e. an algorithm for solving general distributed unconstrained optimization problems of the form:

$$\min_{\mathbf{w}} \sum_{k=1}^N J_k(\mathbf{w}), \tag{10}$$

where $J_k(\cdot)$ is the local (smooth) cost function of the $k$th agent. Denoting by $\mathbf{w}_k[n]$ the local estimate of the $k$th node at time $n$, the DGD method proceeds iteratively as

$$\psi_k = \mathbf{w}_k[n] - \alpha[n]\nabla J_k(\mathbf{w}), \tag{11}$$

$$\mathbf{w}_k[n+1] = \sum_{t=1}^N C_{kt}\psi_t, \tag{12}$$

**Algorithm 1** Distributed $\nabla$S$^3$VM using a diffusion gradient descent procedure.

```
Input: Regularization factors $C_1, C_2$, maximum number of iterations $T$.
1: Initialization:
2:     $\mathbf{w}_k[0] = \mathbf{0}, k = 1, \ldots, N$.
3: for $n$ from 0 to $T$ do
4:     for $k$ from 1 to $N$ do in parallel
5:         Compute auxiliary variable $\psi_k$ using (13).
6:         Combine estimates as $\mathbf{w}_k[n+1] = \sum_{t=1}^N C_{kt}\psi_t$.
7:     end for
8: end for
```

where $\alpha[n]$ is a (possibly time-varying) step-size sequence, $\nabla(\cdot)$ denotes the gradient operator, and $C_{kt}$ are the connectivity weights of the network. This method can be seen as the combination of local descent steps, see (11), followed by variable exchanges and averaging of information among neighbors, see (12).

Distributed gradient algorithms as in (11)–(12) are largely used in the fields of distributed optimization (Nedić & Ozdaglar, 2009; Nedić, Ozdaglar, & Parrilo, 2010; Tsitsiklis, Bertsekas, & Athans, 1986), distributed stochastic optimization (Chen & Sayed, 2012, 2013), and adaptive filtering (Cattivelli, Lopes, & Sayed, 2008; Lopes & Sayed, 2008), among others. All the previous art on DGD focused on the solution of convex versions of problem (10). In our case, the $g_k(\mathbf{w})$ are non-convex, and the analysis in the aforementioned papers cannot be used. However, convergence of a similar family of algorithms in the case of non-convex (smooth) cost functions has been recently studied in Bianchi and Jakubowicz (2013).

Customizing the DGD method in (11)–(12) to Problem (6), we obtain the following local update for the $k$th agent:

$$\psi_k = \mathbf{w}_k[n] - \alpha_k[n] \times \left(\nabla l_k(\mathbf{w}_k[n]) + \nabla g_k(\mathbf{w}_k[n]) + \frac{1}{N}\nabla r(\mathbf{w}_k[n])\right). \tag{13}$$

Note that we have included a factor $\frac{1}{N}$ in (13) in order to be consistent with the formulation in (10). Defining the margin $m_{k,i} = y_{k,i}f(\mathbf{x}_{k,i})$, we can easily show that

$$\nabla l_k(\mathbf{w}) = -\frac{C_1}{L}\sum_{i=1}^{L_k}\mathbb{I}(1-m_{k,i})\cdot m_{k,i}\left(1-m_{k,i}\right), \tag{14}$$

$$\nabla g_k(\mathbf{w}) = -s\frac{C_2}{U}\sum_{i=1}^{U_k}\exp\left\{-sf(\mathbf{x}'_{k,i})^2\right\}f(\mathbf{x}'_{k,i})\mathbf{x}'_{k,i}, \tag{15}$$

$$\nabla r(\mathbf{w}) = \mathbf{w}, \tag{16}$$

where $\mathbb{I}(\cdot)$ is the indicator function defined for a generic scalar $o \in \mathbb{R}$ as

$$\mathbb{I}(o) = \begin{cases} 1 & \text{if } o \le 0 \\ 0 & \text{otherwise.} \end{cases}$$

The overall algorithm is summarized in Algorithm 1. Its convergence properties are illustrated in following theorem.

**Theorem 1.** *Let $\{\mathbf{w}_k[n]\}_n$ be the sequences generated by Algorithm* 1, *and let $\bar{\mathbf{w}}[n] = \frac{1}{N}\sum_{k=1}^N \mathbf{w}_k[n]$ be its average across the agents. Under Assumption* 1, *let us select the step-size sequence $\{\alpha[n]\}_n$ such that* (i) *$\alpha[n] \in (0,1]$, for all $n$,* (ii) *$\sum_{n=0}^{\infty}\alpha[n] = \infty$; and* (iii) *$\sum_{n=0}^{\infty}\alpha[n]^2 < \infty$. Then, if the sequence $\{\bar{\mathbf{w}}[n]\}_n$ is bounded,*[2]

[2] Note that this condition is not restrictive in practical implementations. Indeed, one can always limit the behavior of the algorithm using a finite (but arbitrarily large) box constraint that guarantees the boundness of the sequence $\{\bar{\mathbf{w}}[n]\}_n$, and thus the convergence of the method.

(a) [convergence]*: all its limit points are stationary solutions of problem* (6)*;* (b) [consensus]*: all the sequences* $\mathbf{w}_k[n]$ *asymptotically agree, i.e.* $\lim_{n\to+\infty} \|\mathbf{w}_k[n] - \bar{\mathbf{w}}[n]\|_2 = 0$*,* $k = 1, \ldots, N$*.*

**Proof.** See Bianchi and Jakubowicz (2013). □

### *4.3. Solution 2: in-network successive convex approximations*

The DGD algorithm is extremely efficient to implement, however, as we discussed in Section 1, its convergence is often sub-optimal due to two main reasons. First, the update in (13) considers only first order information and does not take into account the fact that the local cost function has some hidden convexity (since it is composed by the sum of a convex term plus a non-convex term) that one can properly exploit. Second, each agent $k$ obtains information on the cost functions $J_t(\cdot)$, $t \neq k$, only in a very indirect way through the averaging step in (11). In this section, we use a recent framework for in-network non-convex optimization from Di Lorenzo and Scutari (in press), which exploits the structure of nonconvex functions by replacing their linearization (i.e., their gradient) with a "better" approximant, thus typically resulting in enhanced practical convergence speed. In this section we customize the NEXT algorithm from Di Lorenzo and Scutari (in press) to our case, and we refer to the original paper for more details.

The main idea of NEXT is to parallelize the problem in (6) such that, at each agent, the original (global) non-convex cost function is replaced with a strongly convex surrogate that preserves the first order conditions (Di Lorenzo & Scutari, in press). To this aim, we associate to agent $k$ the surrogate $F_k(\mathbf{w}; \mathbf{w}_k[n])$, which is obtained by (i) keeping unaltered the local convex function $l_k(\mathbf{w})$ and the regularization function $r(\mathbf{w})$; (ii) linearizing the local non-convex cost $g_k(\mathbf{w})$ and all the other (non-convex and unknown) terms around the current local iterate $\mathbf{w}_k[n]$. As a result, the surrogate at node $k$ takes the form:

$$F_k(\mathbf{w}; \mathbf{w}_k[n]) = l_k(\mathbf{w}) + \tilde{g}_k(\mathbf{w}; \mathbf{w}_k[n]) + r(\mathbf{w}) + \boldsymbol{\pi}_k(\mathbf{w}_k[n])^T (\mathbf{w} - \mathbf{w}_k[n]), \tag{17}$$

where

$$\tilde{g}_k(\mathbf{w}; \mathbf{w}_k[n]) = g_k(\mathbf{w}_k[n]) + \nabla g_k^T(\mathbf{w}_k[n]) (\mathbf{w} - \mathbf{w}_k[n]), \tag{18}$$

and $\boldsymbol{\pi}_k(\mathbf{w}_k[n])$ is defined as

$$\boldsymbol{\pi}_k(\mathbf{w}_k[n]) = \sum_{t\neq k} \nabla h_t(\mathbf{w}_k[n]), \tag{19}$$

with $\nabla h_k(\cdot) = \nabla l_k(\cdot) + \nabla g_k(\cdot)$. Clearly, the information in (19) related to the knowledge of the other cost functions is not available at node $k$. To cope with this issue, the NEXT approach consists in replacing $\boldsymbol{\pi}_k(\mathbf{w}_k[n])$ in (17) with a local estimate $\tilde{\boldsymbol{\pi}}_k[n]$ that asymptotically converges to $\boldsymbol{\pi}_k(\mathbf{w}_k[n])$, thus considering the local approximated surrogate $\tilde{F}(\mathbf{w}; \mathbf{w}_k[n], \tilde{\boldsymbol{\pi}}_k[n])$ given by

$$\tilde{F}_k(\mathbf{w}; \mathbf{w}_k[n], \tilde{\boldsymbol{\pi}}_k[n]) = l_k(\mathbf{w}) + \tilde{g}_k(\mathbf{w}; \mathbf{w}_k[n]) + r(\mathbf{w}) + \tilde{\boldsymbol{\pi}}_k[n]^T (\mathbf{w} - \mathbf{w}_k[n]). \tag{20}$$

In the first phase of the algorithm, each agent solves a convex optimization problem involving the surrogate function in (20), thus obtaining a new estimate $\tilde{\mathbf{w}}_k[n]$. Then, an auxiliary variable $\mathbf{z}_k[n]$ is computed as a convex combination of the current estimate $\mathbf{w}_k[n]$ and the new $\tilde{\mathbf{w}}_k[n]$, as

$$\mathbf{z}_k[n] = \mathbf{w}_k[n] + \alpha[n] \left(\tilde{\mathbf{w}}_k[n] - \mathbf{w}_k[n]\right), \tag{21}$$

where $\alpha[n]$ is a possibly time-varying step-size sequence. This concludes the optimization phase of NEXT. The consensus phase of NEXT consists of two main steps. First, to achieve asymptotic agreement among the estimates at different nodes, each agent updates its local estimate combining the auxiliary variables from the neighborhood, i.e., for all $k$,

$$\mathbf{w}_k[n+1] = \sum_{t=1}^{N} C_{kt} \mathbf{z}_t[n]. \tag{22}$$

This is similar to the diffusion step in (12). Second, the update of the local estimate $\tilde{\boldsymbol{\pi}}_k[n]$ in (20) is computed in two steps: (i) an auxiliary variable $\mathbf{v}_k[n]$ is updated through a dynamic consensus (Zhu & Martínez, 2010) step as

$$\mathbf{v}_k[n+1] = \sum_{t=1}^{N} C_{kt} \mathbf{v}_t[n] + \Big(\nabla h_k(\mathbf{w}_k[n+1]) - \nabla h_k(\mathbf{w}_k[n])\Big); \tag{23}$$

(ii) the variable $\tilde{\boldsymbol{\pi}}_k[n]$ is updated as

$$\tilde{\boldsymbol{\pi}}_k[n+1] = N\mathbf{v}_k[n+1] - \nabla h_k(\mathbf{w}_k[n+1]). \tag{24}$$

The steps of the NEXT algorithm for Problem (6) are described in Algorithm 2. Its convergence properties are described by a Theorem completely similar to Theorem 1, and the details on the proof can be found in Di Lorenzo and Scutari (in press).

**Algorithm 2** Distributed $\nabla$S$^3$VM using the In-Network Convex Approximation framework.

```
Input: Regularization factors $C_1, C_2$, maximum number of iterations $T$.
 1: Initialization:
 2:     $\mathbf{w}_k[0] = \mathbf{0}, \quad k = 1, \ldots, N$.
 3:     $\mathbf{v}_k[0] = \nabla h_k(\mathbf{w}_k[0]), \quad k = 1, \ldots, N$.
 4:     $\tilde{\boldsymbol{\pi}}_k[0] = (N-1)\mathbf{v}_k[0], \quad k = 1, \ldots, N$.
 5: for $n$ from 0 to $T$ do
 6:     for $k$ from 1 to $N$ do in parallel
 7:         Solve the local optimization problem:
                $\tilde{\mathbf{w}}_k[n] = \arg\min \tilde{F}_k(\mathbf{w}; \mathbf{w}_k[n], \tilde{\boldsymbol{\pi}}_k[n])$.
 8:         Compute $\mathbf{z}_k[n]$ using (21).
 9:     end for
10:     for $k$ from 1 to $N$ do in parallel
11:         Perform consensus step in (22).
12:         Update auxiliary variable using (23).
13:         Set $\tilde{\boldsymbol{\pi}}_k[n+1]$ as (24).
14:     end for
15: end for
```

## 5. Experimental results

### *5.1. Experimental setup*

We tested the proposed distributed algorithms on three semi-supervised learning benchmarks, whose overview is given in Table 1. For more details on the datasets see Chapelle, Schölkopf et al. (2006) for the first two, and Mierswa and Morik (2005) for GARAGEBAND.

- **G50C** is an artificial dataset, wherein labels correspond to Gaussians in a 50-dimensional input space. The dataset is designed such that the Bayes error is 5%.
- **PCMAC** corresponds to distinguishing among pc-related and mac-related texts in the 20 Newsgroup dataset.[3]
- **GARAGEBAND** is a music classification dataset, whose input is given by 49 features extracted according to the procedure

[3] http://qwone.com/jason/20Newsgroups/.

**Table 1**
Description of the datasets. The fourth and fifth columns denote the size of the training and unlabeled datasets, respectively.

| Name | Features | Instances | $L$ | $U$ | Ref. |
|---|---|---|---|---|---|
| G50C | 50 | 550 | 40 | 455 | (Chapelle, Schölkopf et al., 2006) |
| PCMAC | 7511 | 1940 | 40 | 1700 | (Chapelle, Schölkopf et al., 2006) |
| GARAGEBAND | 44 | 790 | 40 | 670 | (Mierswa & Morik, 2005) |

**Table 2**
Optimal values of the parameters used in the experiments. In the first group are reported the values of the regularization coefficients for the three models, averaged over the 150 repetitions. In the following groups are reported the values of the initial step-size and of the diminishing factor for C-∇S3VM, DG-∇S3VM and NEXT-∇S3VM, respectively.

| Dataset | $C_1$ | $C_2$ | $\alpha_0^{C}$ | $\delta^{C}$ | $\alpha_0^{DG}$ | $\delta^{DG}$ | $\alpha_0^{NEXT}$ | $\delta^{NEXT}$ |
|---|---|---|---|---|---|---|---|---|
| G50C | 1 | 1 | 1 | 0.55 | 1 | 0.55 | 0.6 | 0.8 |
| PCMAC | 100 | 100 | 0.1 | 0.55 | 1 | 0.9 | 0.5 | 0.8 |
| GARAGEBAND | 2 | 5 | 0.09 | 0.8 | 0.1 | 0.1 | 0.05 | 0.55 |

detailed in Mierswa and Morik (2005). The original dataset comprises 9 different musical genres. In order to obtain a binary classification task, we select the two most prominent ones, namely ‘rock’ and ‘pop’, and discard the rest of the dataset.

For G50C and GARAGEBAND, input variables are normalized between $-1$ and 1. The experimental results are computed over a 10-fold cross-validation, and all the experiments are repeated 15 times. For each repetition, the training folds are partitioned in one labeled and one unlabeled datasets, according to the proportions given in Table 1. Results are then averaged over the 150 repetitions.

We have implemented the proposed algorithms in an open-source MATLAB toolbox.[4] Since in our implementation we are not concerned with the analysis of communication over a realistic network, we implement a serial version of the code to perform the simulations, in which the network is simulated artificially. We compare the following models:

- **LIN-SVM**: this is a fully supervised SVM with a linear kernel, trained only on the labeled data. The model is trained using the LIBSVM library (Chang & Lin, 2011).
- **RBF-SVM**: similar to before, but a RBF kernel is used instead. The parameter for the kernel is set according to the internal heuristic of LIBSVM.
- **C-∇S3VM**: this is a centralized ∇ S$^3$VM trained on both the labeled and the unlabeled data using a gradient descent procedure.
- **DG-∇S3VM**: in this case, training data (both labeled and unlabeled) is distributed evenly across the network, and the distributed model is trained using the diffusion gradient algorithm detailed in Section 4.2.
- **NEXT-∇S3VM**: data is distributed over the network as before, but the model is trained through the use of the NEXT framework, as detailed in Section 4.3. The internal optimization problem in (20) is solved using a standard gradient descent procedure.

For C-∇S3VM, DG-∇S3VM and NEXT-∇S3VM we set $s = 5$ and a maximum number of iterations $T = 500$. In order to obtain a fair comparison between the algorithms, we also introduce a stopping criterion, i.e. the algorithms terminate when the norm of the gradient of the global cost function in (6) at the current iteration is less than $10^{-5}$. Clearly, this is only for comparison purposes, and a truly distributed implementation would require a more sophisticated mechanism, which however goes outside the scope of the present paper. The same value for the threshold is set for the gradient descent algorithm used within the NEXT framework to optimize the local surrogate function in (20). In this case, we let the gradient run for a maximum of $T = 50$ iterations. We note that, in general, we do not need to solve the internal optimization problem to optimal accuracy, as convergence of NEXT is guaranteed as long as the problems are solved with increasing accuracy for every iteration (Di Lorenzo & Scutari, in press).

We searched the values of $C_1$ and $C_2$ by executing a 5-fold cross-validation in the interval $\{10^{-5}, 10^{-4}, \dots, 10^{3}\}$ using C-∇S3VM as in Chapelle and Zien (2005). The values of these parameters are then shared with DG-∇S3VM and NEXT-∇S3VM. For all the models, included NEXT's internal gradient descent algorithm, the step-size $\alpha$ is chosen using a decreasing strategy given by

$$\alpha[n] = \frac{\alpha_0}{(n+1)^{\delta}}, \tag{25}$$

where $\alpha_0, \delta > 0$ are set by the user. In particular, this step-size sequence satisfies the convergence conditions for both the DGD algorithm and NEXT, see Theorem 1. After preliminary tests, we selected for every model the values of $\alpha_0$ and $\delta$ that guarantee the fastest convergence. The optimal values of the parameters are shown in Table 2.

The network topologies are undirected and generated according to the so-called ‘Erdős–Rényi model’ (Newman, 2010), such that every edge has a 25% probability of appearing. The only constraint is that the network is connected. The topologies are generated at the beginning of the experiments and kept fixed during all the repetitions.

We choose the weight matrix $\mathbf{C}$ using the *Metropolis–Hastings* strategy (Lopes & Sayed, 2008):

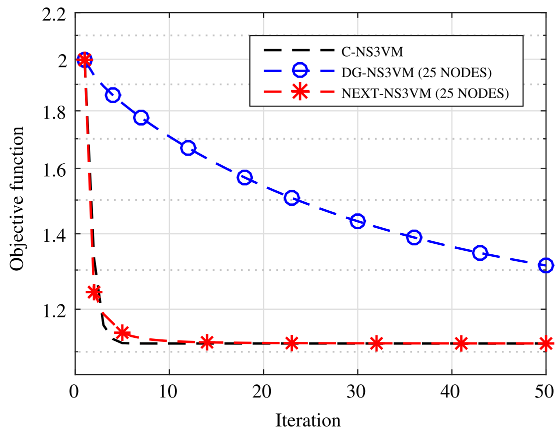
$$C_{kj} = \begin{cases} \dfrac{1}{\max\{d_k, d_j\}+1} & k \neq j,\ \{k,j\} \in \mathcal{E} \\ 1 - \displaystyle\sum_{j\in\mathcal{N}_k} \dfrac{1}{\max\{d_k, d_j\}+1} & k = j \\ 0 & k \neq j,\ \{k,j\} \notin \mathcal{E} \end{cases} \tag{26}$$

where $d_k$ is the degree of node $k$ and $\mathcal{N}_k$ is the set of nodes' indexes directly connected to node $k$. This choice of the weight matrix satisfies the convergence conditions for both the distributed approaches, see Assumption 1.

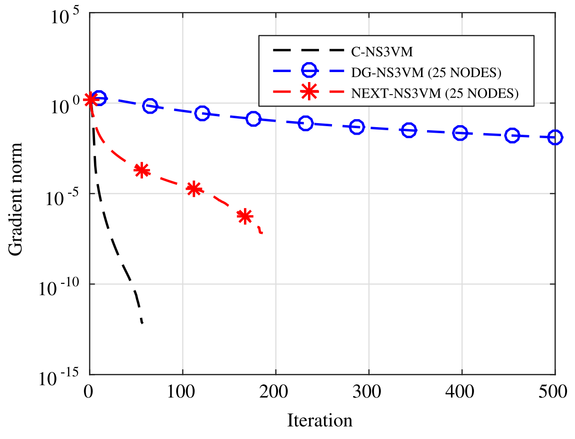
## *5.2. Results and discussion*

The first set of experiments consists in analyzing the performance of C-∇S3VM, when compared to a linear SVM and RBF-SVM trained only on the labeled data. While these results are well known in the semi-supervised literature, they allow us to quantitatively evaluate the performance of C-∇S3VM, in order to provide a coherent benchmark for the successive comparisons. Results of this experiment are shown in Table 3.

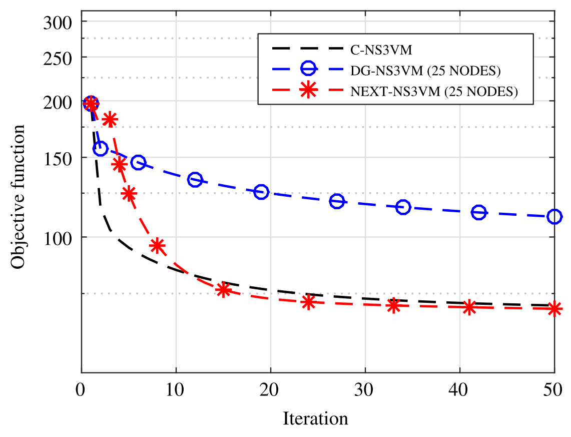
[4] https://bitbucket.org/robertofierimonte/code-distributed-s3vm.



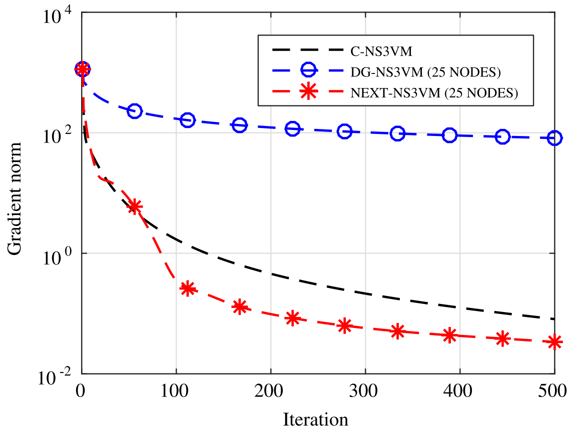
(a) Objective function (G50C).



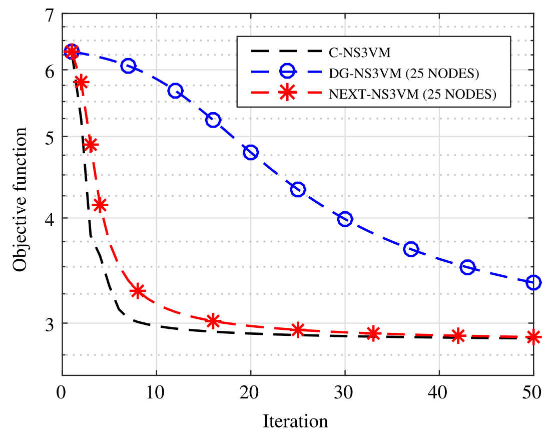
(b) Gradient norm (G50C).



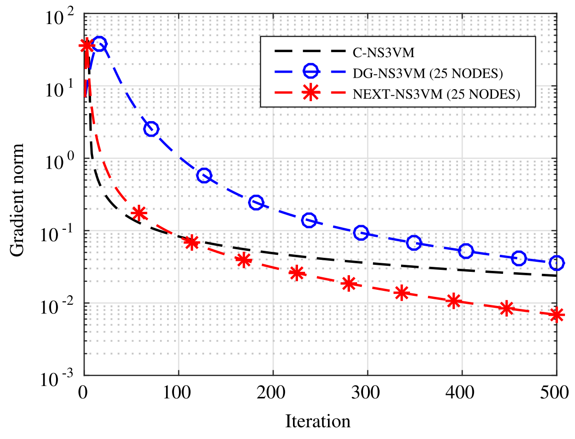
(c) Objective function (PCMAC).



(d) Gradient norm (PCMAC).



(e) Objective function (GARAGEBAND).



(f) Gradient norm (GARAGEBAND).

**Fig. 2.** Convergence behavior of DG-∇S3VM and NEXT-∇S3VM, compared to C-∇S3VM. The panels on the left show the evolution of the global cost function, while the panels on the right show the evolution of the squared norm of the gradient.

We can see that, for all the datasets, *C*-∇S3VM outperforms standard SVMs trained only on labeled data, with a reduction of the classification error ranging from 2.37% on GARAGEBAND to 15.22% on PCMAC. Clearly, the training time required by C-∇S3VM is higher than the time required by a standard SVM, due to the larger number of training data, and to the use of the gradient descent algorithm. Another important aspect to be considered is that, with the only exception of G50C, the RBF-SVM fails in matching the performance of the SVM with linear kernel due to higher complexity of the model in relationship to the amount of training data.

Next, we investigate the convergence behavior of DG-∇S3VM and NEXT-∇S3VM, compared to the centralized implementation. In particular, we test the algorithm on randomly generated networks of $N = 25$ nodes. Results are presented in Fig. 2. Particularly, panels on the left show the evolution of the global cost function in (6), while panels on the right show the evolution of the squared norm of the gradient. For readability, the graphs use a logarithmic scale on the *y*-axis, while on the left we only show the first 50 iterations of the optimization procedure. The results are similar for all three datasets: NEXT-∇S3VM is able to converge faster (up to one/two orders of magnitude) than

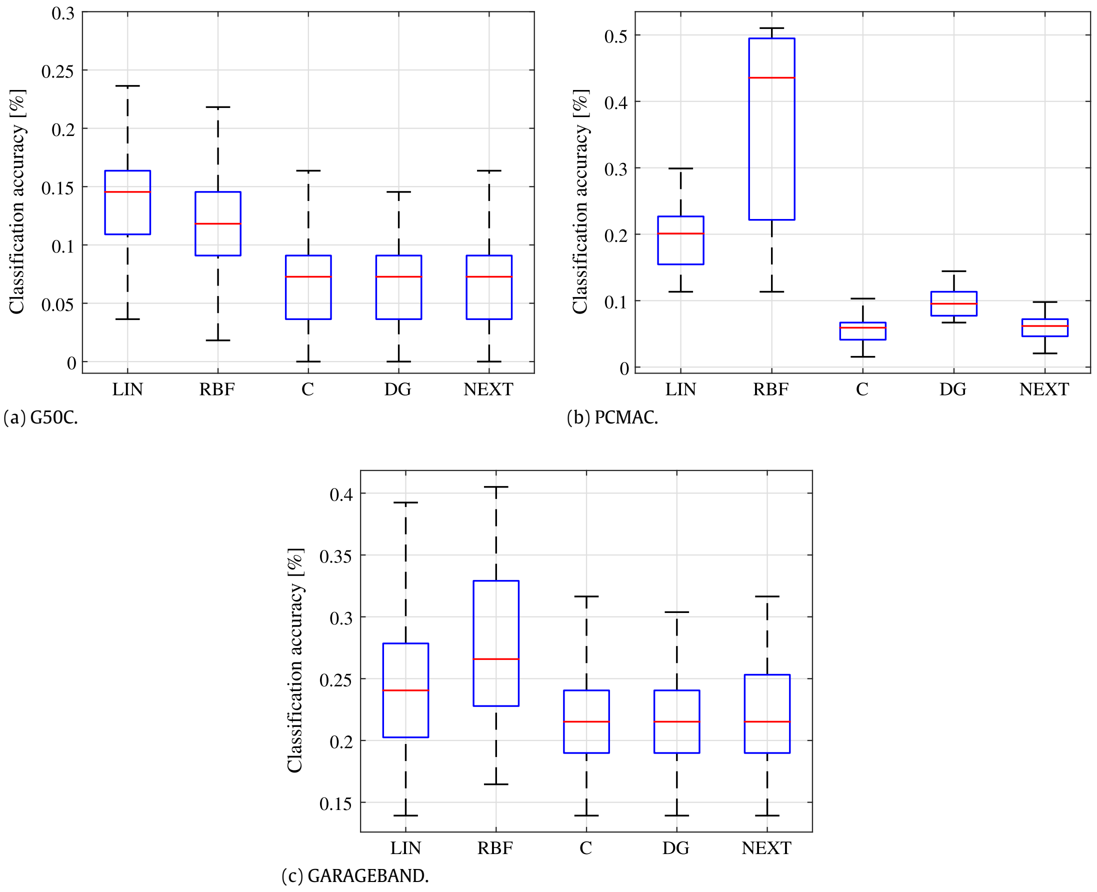


**Fig. 3.** Box plots for the classification accuracy of the 5 algorithms, in the case $N = 25$. The central line is the median, the edges are the 25th and 75th percentiles, and the whiskers extend to the most extreme data points. For readability, the names of the algorithms have been abbreviated to LIN (LIN-SVM), RBF (RBF-SVM), C (C-∇S3VM), DG (DG-∇S3VM) and NEXT (NEXT-∇S3VM).

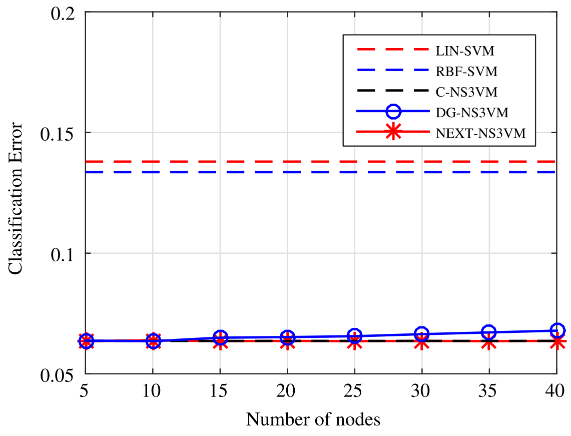
**Table 3**
Average value for classification error and computational time for the centralized algorithms.

| Dataset | Algorithm | Error (%) | Time (s) |
|---|---|---|---|
| G50C | LIN-SVM | 13.79 | 0.0008 |
| | RBF-SVM | 13.36 | 0.0005 |
| | **C-∇S3VM** | **6.36** | **0.024** |
| PCMAC | LIN-SVM | 21.32 | 0.0035 |
| | RBF-SVM | 36.68 | 0.0032 |
| | **C-∇S3VM** | **6.10** | **35.12** |
| GARAGEBAND | LIN-SVM | 23.87 | 0.0010 |
| | RBF-SVM | 27.92 | 0.0007 |
| | **C-∇S3VM** | **21.50** | **0.2872** |

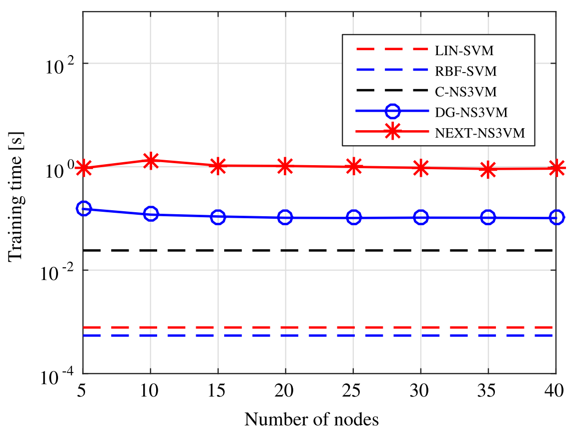
DG-∇S3VM, thanks to a better exploitation of the structure of the objective function. Indeed, both NEXT-∇S3VM and the centralized implementation are able to converge to a stationary point in a relatively small number of iterations, as shown by the panels on the left. The same can be seen from the gradient norm evolution, shown on the right panels, where the fast convergence of NEXT-∇S3VM is even more pronounced. Similar insights can be obtained by the analysis of the box plots in Fig. 3, where we also compare with the results of LIN-SVM and RBF-SVM obtained previously. The same conclusions are also obtained with a rigorous statistical test. In particular, the corrected Friedman test (see Demšar, 2006, Section 3.2.2 for details and a discussion on its significance when comparing classifiers over multiple datasets) finds significant differences among the results of the 5 algorithms, with a $p = 0.05$ confidence value (in particular, the corrected statistic has a value of 20.5 which is greater than the critical value 3.84). To analyze which pair of algorithms present significant differences, we run a set of post-hoc Nemenyi tests with the same confidence interval (as also suggested in Demšar, 2006), revealing differences between LIN-SVM, and both C-∇S3VM and NEXT-∇S3VM, and similarly for RBF-SVM.

As a final experiment, we investigate the scalability of the distributed algorithms, by analyzing the training time and the test error of DG-∇S3VM and NEXT-∇S3VM when varying the number of nodes in the network from $N = 5$ to $N = 40$ by steps of 5. Results of this experiment are shown in Fig. 4. The three panels on the left show the evolution of the classification error, while the three panels on the right show the evolution of the training time. Results of LIN-SVM, RBF-SVM and C-∇S3VM are shown with dashed lines for comparison. It is possible to see that NEXT-∇S3VM can track efficiently the centralized solution in all settings, regardless of the size of the network, while DG-∇S3VM is not able to properly converge (in the required number of iterations) for larger networks on PCMAC. With respect to training time, results are more varied. Generally speaking, NEXT-∇S3VM requires in average more training time than DG-∇S3VM. However, for large datasets (PCMAC and GARAGEBAND) both algorithms are comparable in training time with the centralized solution and, more notably, the training time generally decreases for bigger networks.

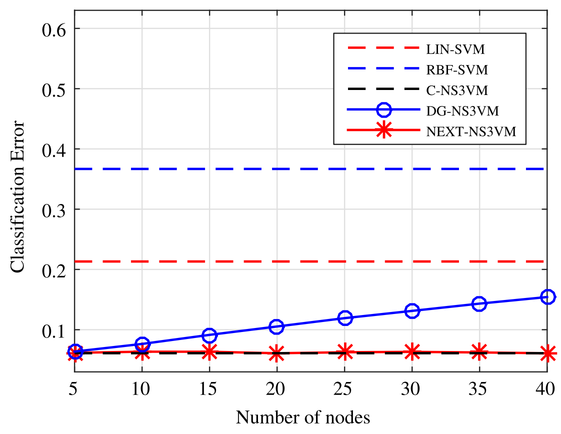
It is worth mentioning here that the results presented in this paper strongly depend on our selection of the step-size sequences, and the specific surrogate function in (20). In the former case, it is known that the convergence speed of any gradient descent procedure can be accelerated by considering a proper adaptable step-size criterion. Along a similar reasoning, the training time of NEXT-∇S3VM can in principle be decreased by loosening the precision to which the internal surrogate function is optimized,



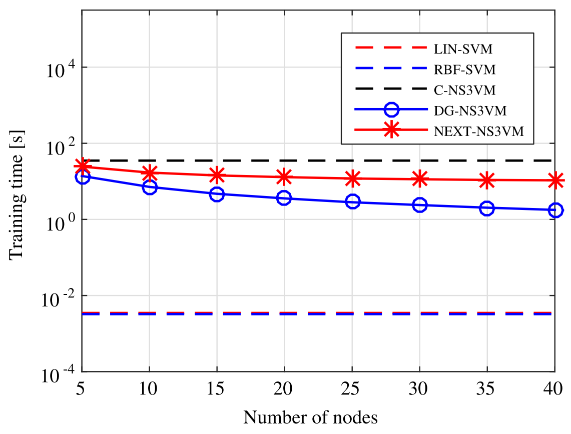
(a) Classification error (G50C).



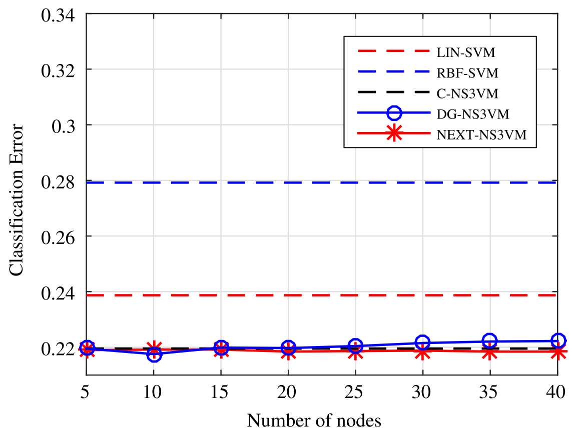
(b) Training time (G50C).



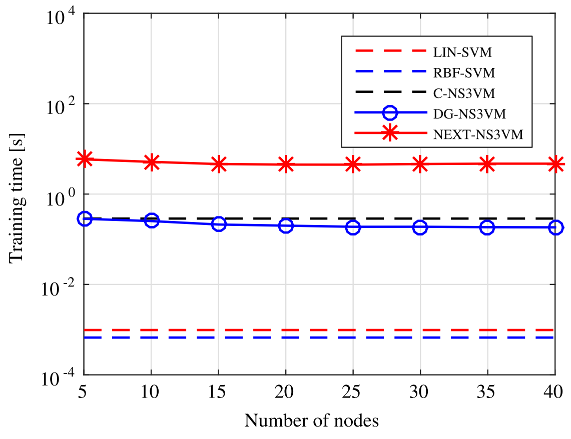
(c) Classification error (PCMAC).



(d) Training time (PCMAC).



(e) Classification error (GARAGEBAND).



(f) Training time (GARAGEBAND).

**Fig. 4.** Training time and test error of GD-∇S3VM and NEXT-∇S3VM when varying the number of nodes in the network from $N = 5$ to $N = 40$. Results for LIN-SVM, RBF-SVM and C-∇S3VM are shown with dashed lines for comparison.

due to the convergence properties of NEXT already mentioned above. Finally, we can also envision a different choice of surrogate function for NEXT-∇S3VM, in order to achieve a different trade-off between training time and speed of convergence. As an example, we can replace the hinge loss $l_k(\mathbf{w})$ with its first-order linearization $\tilde{l}_k(\mathbf{w})$, similarly to (18). In this case, the resulting optimization problem would have a closed form solution, resulting in a faster training time per iteration (at the cost of more iterations required for convergence).

Overall, the experimental results suggest that both algorithms can be efficient tools for training a ∇S3VM in a distributed setting, wherein NEXT-∇S3VM is able to converge extremely faster, at the expense of a larger training time. Thus, the choice of a specific algorithm will depend on the applicative domain, and on the amount of computational resources (and size of the training dataset) available to each agent.

## 6. Conclusions

In this paper, we have investigated the problem of learning a semi-supervised support vector machine, when training data is distributed over a network of interconnected agents. Particularly, we have leveraged over recent advances on distributed non-convex optimization, in order to provide two flexible mechanisms with a different balance in computational requirements and speed of convergence. Overall, our work is one of the first steps toward semi-supervised distributed learning which, as we stated in Section 1, has a large number of practical applications in real-world networks.

In this sense, a natural extension would be to consider different semi-supervised techniques to be extended to the distributed setting, particularly among those developed for the $S^3$VM (Chapelle et al., 2008), including the possibility of having more general regularizers instead of the $L_2$ norm, such as the $L_1$ norm to encourage sparsity (Gnecco & Sanguineti, 2010; Kurková, 2005). A second line of research involves relaxing some of the assumptions we made in this work, particularly in terms of synchronous updates (requiring a mechanism for correctly synchronizing the agents in the network), and fixed connectivity of the underlying graph. Indeed, NEXT was originally designed for time-varying connectivities (Di Lorenzo & Scutari, in press), while there has been multiple efforts recently in order to design asynchronous gradient updates over networks (Zhao & Sayed, 2015). Finally, as discussed in the previous section, we can customize the proposed algorithms with adaptive criteria for the internal solvers, or different choices of the surrogate cost function in NEXT-$\nabla$S3VM. We are planning to consider all these aspects in our future works.

## References

Adankon, M. M., Cheriet, M., & Biem, A. (2009). Semisupervised least squares support vector machine. *IEEE Transactions on Neural Networks*, *20*(12), 1858–1870.

Ang, H. H., Gopalkrishnan, V., Hoi, S. C., & Ng, W. K. (2013). Classification in P2P networks with cascade support vector machines. *ACM Transactions on Knowledge Discovery from Data*, *7*(4), 20.

Barbarossa, S., Sardellitti, S., & Di Lorenzo, P. (2014). Distributed detection and estimation in wireless sensor networks. In R. Chellappa, & S. Theodoridis (Eds.), *Communications and radar signal processing*: *Vol. 2. Academic press library in signal processing* (pp. 329–408).

Bianchi, P., & Jakubowicz, J. (2013). Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization. *IEEE Transactions on Automatic Control*, *58*(2), 391–405.

Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, *3*(1), 1–122.

Cattivelli, F. S., Lopes, C. G., & Sayed, A. H. (2008). Diffusion recursive least-squares for distributed estimation over adaptive networks. *IEEE Transactions on Signal Processing*, *56*(5), 1865–1877.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*, 27:1–27:27.

Chapelle, O., Schölkopf, B., & Zien, A. (2006). *Semi-supervised learning*. Cambridge: MIT Press.

Chapelle, O., Sindhwani, V., & Keerthi, S. S. (2006). Branch and bound for semi-supervised support vector machines. In *Advances in neural information processing systems* (pp. 217–224).

Chapelle, O., Sindhwani, V., & Keerthi, S. (2008). Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, *9*, 203–233.

Chapelle, O., & Zien, A. (2005). Semi-supervised classification by low density separation. In *Proceedings of the tenth international workshop on artificial intelligence and statistics. Vol. 1* (pp. 57–64).

Chen, J., & Sayed, A. H. (2012). Diffusion adaptation strategies for distributed optimization and learning over networks. *IEEE Transactions on Signal Processing*, *60*(8), 4289–4305.

Chen, J., & Sayed, A. H. (2013). Distributed pareto optimization via diffusion strategies. *IEEE Journal of Selected Topics in Signal Processing*, *7*(2), 205–220.

Chen, J., Wang, C., Sun, Y., & Shen, X. S. (2011). Semi-supervised Laplacian regularized least squares algorithm for localization in wireless sensor networks. *Computer Networks*, *55*(10), 2481–2491.

Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., & Zhu, M. Y. (2002). Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations Newsletter*, *4*(2), 28–34.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Di Lorenzo, P., & Sayed, A. H. (2013). Sparse distributed learning based on diffusion adaptation. *IEEE Transactions on Signal Processing*, *61*(6), 1419–1433.

Di Lorenzo, P., & Scutari, G. (2016). NEXT: In-network nonconvex optimization. *IEEE Transactions on Signal and Information Processing over Networks*, in press.

Facchinei, F., Scutari, G., & Sagratella, S. (2015). Parallel selective algorithms for nonconvex big data optimization. *IEEE Transactions on Signal Processing*, *63*(7), 1874–1889.

Fierimonte, R., Scardapane, S., Uncini, A., & Panella, M. (2015). Fully decentralized semi-supervised learning via privacy-preserving matrix completion. *IEEE Transactions on Neural Networks and Learning Systems*, submitted for publication.

Flouri, K., Beferull-Lozano, B., & Tsakalides, P. (2006). Training a SVM-based classifier in distributed sensor networks. In *Proceedings of 14nd European signal processing conference. EUSIPCO'06* (pp. 1–5).

Forero, P. A., Cano, A., & Giannakis, G. B. (2010). Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, *11*, 1663–1707.

Fung, G., & Mangasarian, O. L. (2001). Semi-superyised support vector machines for unlabeled data classification. *Optimization Methods and Software*, *15*(1), 29–44.

Gnecco, G., & Sanguineti, M. (2010). Regularization techniques and suboptimal solutions to optimization problems in learning from data. *Neural Computation*, *22*(3), 793–829.

Graf, H. P., Cosatto, E., Bottou, L., Dourdanovic, I., & Vapnik, V. (2004). Parallel support vector machines: The cascade SVM. In *Advances in neural information processing systems* (pp. 521–528).

Hensel, C., & Dutta, H. (2009). GADGET SVM: a gossip-based sub-gradient SVM solver. In *Proceedings of the 2009 international conference on machine learning. ICML'2009*.

Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., & Jordan, M. I. (2014). Communication-efficient distributed dual coordinate ascent. In *Advances in neural information processing systems* (pp. 3068–3076).

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the 1999 international conference on machine learning. ICML'99* (pp. 200–209).

Kurková, V. (2005). Neural network learning as an inverse problem. *Logic Journal of IGPL*, *13*(5), 551–559.

Lee, S., & Nedic, A. (2013). Distributed random projection algorithm for convex optimization. *IEEE Journal of Selected Topics in Signal Processing*, *7*(2), 221–229.

Lee, C.-P., & Roth, D. (2015). Distributed box-constrained quadratic optimization for dual linear SVM. In *Proceedings of the 32nd international conference on machine learning. ICML'15. ICML*.

Li, Y.-f., Tsang, I. W., & Kwok, J. T. (2013). Convex and scalable weakly labeled SVMs. *Journal of Machine Learning Research*, *14*, 2151–2188.

Lodi, S., Aanculef, R., & Sartori, C. (2010). Single-pass distributed learning of multi-class SVMs using core-sets. In *Proceedings of the 2010 SIAM international conference on data mining* (pp. 257–268).

Lopes, C. G., & Sayed, A. H. (2008). Diffusion least-mean squares over adaptive networks: Formulation and performance analysis. *IEEE Transactions on Signal Processing*, *56*(7), 3122–3136.

Lu, Y., Roychowdhury, V., & Vandenberghe, L. (2008). Distributed parallel support vector machines in strongly connected networks. *IEEE Transactions on Neural Networks*, *19*(7), 1167–1178.

Mateos, G., Bazerque, J. A., & Giannakis, G. B. (2010). Distributed sparse linear regression. *IEEE Transactions on Signal Processing*, *58*(10), 5262–5276.

Mierswa, I., & Morik, K. (2005). Automatic feature extraction for classifying audio data. *Machine Learning*, *58*(2–3), 127–149.

Navia-Vázquez, A., Gutierrez-Gonzalez, D., Parrado-Hernández, E., & Navarro-Abellan, J. J. (2006). Distributed support vector machines. *IEEE Transactions on Neural Networks*, *17*(4), 1091–1097.

Nedić, A., & Ozdaglar, A. (2009). Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, *54*(1), 48–61.

Nedić, A., Ozdaglar, A., Parrilo, P., et al. (2010). Constrained consensus and optimization in multi-agent networks. *IEEE Transactions on Automatic Control*, *55*(4), 922–938.

Newman, M. (2010). *Networks: an introduction*. Oxford University Press.

Predd, J. B., Kulkarni, S. R., & Poor, H. V. (2006). Distributed learning in wireless sensor networks. *IEEE Signal Processing Magazine*, *23*(4), 56–69.

Rifkin, R., & Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, *5*, 101–141.

Sayed, A. H. (2014a). Adaptation, learning, and optimization over networks. *Foundations and Trends in Machine Learning*, *7*(4–5), 311–801.

Sayed, A. H. (2014b). Adaptive networks. *Proceedings of the IEEE*, *102*(4), 460–497.

Scardapane, S., Comminiello, D., Scarpiniti, M., & Uncini, A. (2015). A semi-supervised random vector functional-link network based on the transductive framework. *Information Sciences*, in press.

Scardapane, S., Wang, D., & Panella, M. (2015). A decentralized training algorithm for echo state networks in distributed big data applications. *Neural Networks*, in press.

Scardapane, S., Wang, D., Panella, M., & Uncini, A. (2015). Distributed learning for random vector functional-link networks. *Information Sciences*, *301*, 271–284.

Scutari, G., Facchinei, F., Song, P., Palomar, D. P., & Pang, J.-S. (2014). Decomposition by partial linearization: Parallel optimization of multi-agent systems. *IEEE Transactions on Signal Processing*, *62*(3), 641–656.

Tsitsiklis, J. N., Bertsekas, D. P., Athans, M., et al. (1986). Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, *31*(9), 803–812.

Vapnik, V. (1998). *Statistical learning theory, vol. 1*. New York: Wiley.

Wang, D., & Zhou, Y. (2012). Distributed support vector machines: An overview. In *Proceedings of the 2012 24th Chinese control and decision conference*, CCDC'12. (pp. 3897–3901). IEEE.

Zhao, X., & Sayed, A. H. (2015). Asynchronous adaptation and learning over networks part I: Modeling and stability analysis. *IEEE Transactions on Signal Processing*, *63*(4), 811–826.

Zhu, M., & Martínez, S. (2010). Discrete-time dynamic average consensus. *Automatica*, *46*(2), 322–329.