Local-to-Global Support Vector Machines (LGSVMs)

Local-to-Global Support Vector Machines (LGSVMs)

F. Marchetti*, E. Perracchione**

*Dipartimento di Matematica "Tullio Levi-Civita", Università di Padova, Italy;
**Dipartimento di Scienze Matematiche "Giuseppe Luigi Lagrange", Politecnico di Torino, Italy

Abstract

For supervised classification tasks that involve a large number of instances, we propose and study a new efficient tool, namely the Local-to-Global Support Vector Machine (LGSVM) method. Its background *somehow* lies in the framework of approximation theory and of local kernel-based models, such as the Partition of Unity (PU) method. Indeed, even if the latter needs to be accurately tailored for classification tasks, such as allowing the use of the cosine semimetric for defining the patches, the LGSVM is a global method constructed by gluing together the local SVM contributions via compactly supported weights. When the number of instances grows, such a construction of a global classifier enables us to significantly reduce the usually high complexity cost of SVMs. This claim is supported by a theoretical analysis of the LGSVM and of its complexity as well as by extensive numerical experiments carried out by considering benchmark datasets.

Keywords: Local-to-Global Support Vector Machines, Partition of Unity, supervised classification, kernel models.

1. Introduction

Background. Support Vector Machines (SVMs) are one of the most popu-

Email addresses: francesco.marchetti@math.unipd.it (F. Marchetti*), emma.perracchione@polito.it (E. Perracchione**)

lar classification methods [28]. They take advantage of being easy to implement; however when the number of instances grows, both the complexity costs and the memory needs become prohibitive (refer e.g. to [26] for a general overview). The same problems occur in the context of scattered data approximation, and in this setting the so-called Partition of Unity (PU) scheme is nowadays a well-established and efficient kernel-based interpolation method. First introduced in the mid 1990s in [1], the PU scheme constructs a global approximant by means of the contributes of many local fits, which are glued together via the use of weight functions, i.e. compactly supported Radial Basis Functions (RBFs); refer e.g. to [35]. Such a scheme is also rather popular for researchers working on local collocation schemes for PDEs; refer e.g. to [23].

Related works. For classification tasks involving large datasets, several works deal with approximating the kernel matrix (with a particular focus on dot product and additive kernels [25]), as well as with reducing the number of support vectors [20] or with constructing decision tree-based decompositions [10]. Those approaches produce rather sparse global classifiers by projecting into lower dimensional spaces. Moreover, many *local* algorithms have been developed [16, 34, 39]. In this direction, most of recent research focuses on the parallel implementation of local classifiers [24] and on the integration of k-Nearest Neighbors (kNN) with SVM; see e.g. [4]. Such an idea has been further developed in other works, refer e.g. to [5, 38]. Doing locally in this way, the method turns out to be memory-efficient but a *local-to-global classifier* is not defined, meaning that a new example might be labelled without taking into account the contributions of neighboring classifiers. Indeed, the authors of such papers focus on how to select a reduced number of examples to construct local and disjoint classifiers. Some efforts in defining a *global* classifier starting from local ones can be found in [12, 15]. In the last paper, the authors mix together

the local contributions via the use of indicator functions. We further point out that the work outlined in [21] shares similar spirit with ours: the local models (constructed with kernels that are defined as the product of a local kernel times a global one) are weighted via a non-negative global gating function, which can be interpreted as the probability of picking a given classifier. The main differences with this paper and our idea are that: 1) we take compactly supported weights that form a partition of unity and this in turn allows to provide a clear theoretical framework; 2) our setting accomplishes the usage of any semi-metric, and it is not limited to the Euclidean norm.

Main contributions. Inspired by the PU scheme, we investigate what we call the Local-to-Global SVM (LGSVM) method. For a huge number of instances, the LGSVM allows us to construct a global classifier by solving many *small* SVM classification problems. As confirmed by the complexity analysis, this surely leads to a saving in terms of computational time. The basic idea, as in the standard PU scheme, is to distribute the initial set of examples into many subdomains or patches. While such patches are classically defined as balls in the Euclidean metric, whose centres form a grid on the domain $\Omega \subseteq \mathbb{R}^d$, here we allow the more general case of subdomains built via a given (semi-)metric $\mu : \Omega \times \Omega \longrightarrow \mathbb{R}$. This leads to the construction of μ-balls as patches. Precisely, we focus on the performances of the *cosine semi-metric* [31], which is based on the so-called *cosine similarity*, whose usage results in efficient LGSVM schemes (at least for instances characterized by *a few* features). Indeed, in high dimensions we should construct huge μ-grids of subdomains and both the memory requirements and the computational times would again grow consequently. To avoid this drawback, we propose an alternative approach where we give up the idea of covering the entire domain and we select the centres among the training elements.

Outline. The paper is organized as follows. In Section 2, we briefly review the classification problem via SVMs. Section 3 is devoted to the presentation of our method. The complexity analysis is carried out in Section 4, where we also sketched some implementation details. Finally, Section 5 deals with numerical tests and the last section with conclusions and future work.

2. Kernel-based classification with SVMs

In this section we briefly review the main concepts of SVMs in the framework of supervised machine learning for classification tasks. For further details, we refer the reader to [17, 29]. Let us consider the following dataset $Z_n = \{(\boldsymbol{x}_i, y_i),\ i = 1, \dots, n\}$, $\boldsymbol{x}_i \in \Omega \subseteq \mathbb{R}^d$ and $y_i \in \{-1, +1\}$, $i = 1, \dots, n$. The binary classification problem consists in finding a predictor, i.e. a decision function $s : \Omega \longrightarrow \{-1, +1\}$, that assigns appropriate labels, i.e. $\tilde{y}_i \in \{-1, +1\}$, to other unknown samples $\tilde{\boldsymbol{x}}_i$, $i = 1, \dots, m$. We might refer to $X_n = \{\boldsymbol{x}_i,\ i = 1, \dots, n\} \subset \Omega$ and to $T_m = \{\tilde{\boldsymbol{x}}_i,\ i = 1, \dots, m\} \subset \Omega$ as training and test sets, respectively. Even if in this presentation we restrict to the binary classification setting, any extension to multiple output classes is achievable (e.g. by means of the so-called *one-versus-rest* approach [6]). In the simplest case, the SVM algorithm provides a classifier that is fully characterized by a hyperplane. More precisely, the resulting decision function is of the form $s(\boldsymbol{x}) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$, where "$\cdot$" denotes the Euclidean scalar product, and where the vector of weights $\boldsymbol{w} \in \mathbb{R}^d$ and the bias $b \in \mathbb{R}$ are the solution of the following problem (see e.g. [29]):

$$\begin{cases} \min_{\boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}, \boldsymbol{\xi} \in \mathbb{R}^n} \dfrac{1}{2}\|\boldsymbol{w}\|_2^2 + C \sum_{i=1}^n \xi_i, \\ \text{s.t. } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \dots, n. \end{cases} \tag{1}$$

The so-called *slack variables* $\xi_i \in \mathbb{R}^+ = [0, +\infty)$, $i = 1, \dots, n$, $\boldsymbol{\xi} \coloneqq (\xi_1, \dots, \xi_n)$, model potential classification errors and $C \in \mathbb{R}^+$, which characterizes the so-

called *bounding box*, is a hyperparameter that controls the acceptance of such misclassifications in the training set X_n.

In order to construct non-linear classifiers, we consider kernels $\kappa : \Omega \times \Omega \longrightarrow \mathbb{R}$ that admit a Mercer's decomposition (see e.g. [17, Theorem 2.2]) as $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k\geq 0} \lambda_k \rho_k(\boldsymbol{x})\rho_k(\boldsymbol{y})$, $\boldsymbol{x}, \boldsymbol{y} \in \Omega$, where the series converges uniformly and absolutely and $\{\rho_k\}_{k\geq 0}$ is a countable set of eigenfunctions (with the associated eigenvalues $\{\lambda_k\}_{k\geq 0}$) of the operator $\mathcal{T} : L_2(\Omega) \longrightarrow L_2(\Omega)$, given by $\mathcal{T}[f](\boldsymbol{x}) = \int_\Omega \kappa(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y}) d\boldsymbol{y}$. We remark that a kernel admits a Mercer's decomposition if and only if it is a positive definite kernel. Moreover, the fact that a positive definite kernel can be written in terms of an inner product in the so-called *feature space* F, which is a Hilbert space, holds true. In other words,

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle_F, \quad \boldsymbol{x}, \boldsymbol{y} \in \Omega, \tag{2}$$

where $\Phi : \Omega \longrightarrow F$ is known as feature map. The feature map and space associated to a kernel are not unique. A possible choice is the one of taking the map $\Phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x})$, which is linked to the characterization of F as a reproducing kernel Hilbert space; see [17, 29] for further details. Within this context, given $\boldsymbol{x} \in \Omega$, a non-linear SVM classifier is defined by the following decision function:

$$s(\boldsymbol{x}) = \text{sign}(h(\boldsymbol{x})) \coloneqq \text{sign}(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}) + b), \tag{3}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are the solution of:

$$\begin{cases} \min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^n} \dfrac{1}{2}||\boldsymbol{w}||_2^2 + C\sum_{i=1}^n \xi_i, \\ \text{s.t. } y_i h(\boldsymbol{x}_i) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \end{cases} \tag{4}$$

with C and $\xi_i \in \mathbb{R}^+$, $i = 1, \ldots, n$. Actually, if F is a high-dimensional space, the minimization problem formulated in (4) is computationally inconvenient

or even infeasible. For instance, the widely-used *Gaussian* kernel $\kappa_G(\boldsymbol{x}, \boldsymbol{y}) \coloneqq e^{-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}$, where $\boldsymbol{x}, \boldsymbol{y} \in \Omega,\ \gamma > 0$, leads to an infinite dimensional feature space. Therefore, it is necessary to reformulate the optimization problem in its *dual form*:

$$\begin{cases} \min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \dfrac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \alpha_i\alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^n \alpha_i, \\ \text{s.t. } \sum_{i=1}^n \alpha_i y_i = 0, \\ 0 \le \alpha_i \le C, \quad i = 1, \dots, n, \end{cases} \tag{5}$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{R}^n$ is the vector of Lagrange multipliers. Here, by taking advantage of the so-called *kernel trick* (2), we evaluate the kernel function κ on the training elements and thus we avoid the calculation of inner products in the feature space. Moreover, the function h in (3) can be expressed as

$$h(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i y_i \kappa(\boldsymbol{x}, \boldsymbol{x}_i) + b, \quad \boldsymbol{x} \in \Omega. \tag{6}$$

We point out that the linear setting is included in this framework by considering the *linear* kernel $\kappa_L(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \boldsymbol{x} \cdot \boldsymbol{y}$.

3. Local-to-Global SVM (LGSVM)

We now present the LGSVM method. Despite it shows similarities with the PU scheme, which in approximation theory is a popular meshfree collation/interpolation approach, modifications and adaptations to the different context are necessary. As an example, while in the classical PU setting the metric used for decomposing the domain usually coincides with the Euclidean distance, here we allow the more general case of patches built via a given semi-metric $\mu : \Omega \times \Omega \longrightarrow \mathbb{R}$. Moreover, we have to introduce a different notion of domain covering, as explained below.

3.1. Admissible and quasi-admissible coverings

We first introduce the concept of *(quasi-)admissible* coverings.

Definition 1. *Let $\Omega \subset \mathbb{R}^d$ and let $\mu : \Omega \times \Omega \longrightarrow \mathbb{R}$ be a semi-metric (or semi-distance) and let Ω be connected and bounded in the μ semi-metric. Moreover, let $Z_n = \{(\boldsymbol{x}_i, y_i) \,:\, \boldsymbol{x}_i \in X_n,\ y_i = C_1, \dots, C_\ell,\ i = 1, \dots, n,\ \ell \in \mathbb{N},\ \ell \geq 2\} \subset \Omega$, be a dataset whose elements belong to one out of ℓ classes. Let us consider a collection of sets (or patches) $\mathcal{C}_p \coloneqq \{\Omega_1, \dots, \Omega_p\}$, where $\Omega_k \subset \mathbb{R}^d$ is open, connected and bounded with respect to μ for each $k = 1, \dots, p$. Then, $\mathcal{C}_p$ is an admissible covering for (Ω, X_n) if the following properties are satisfied.*

1. *The set $\mathcal{C}_p$ is a covering of Ω, i.e. $\Omega \subseteq \bigcup_{k=1}^{p} \Omega_k$ (covering assumption).*
2. *For each $k = 1, \dots, p$, there exists at least one point $\boldsymbol{x} \in \Omega$ such that $\boldsymbol{x} \in \Omega_k$ (partitioning assumption).*
3. *In each patch Ω_k, $k = 1, \dots, p$, at least two classes out of $C_1, \dots, C_\ell$ are represented by some elements of X_n (representation assumption).*

Moreover, $\mathcal{C}_p$ is a quasi-admissible covering for (Ω, X_n) if the representation assumption does not hold true.

Note that, the covering assumption in Definition 1 comes from the classical PU method (see e.g. [35]), while the representation assumption is introduced for accommodating the classification setting. Indeed, the following proposition is a direct consequence of the partitioning assumption of (quasi-)admissible coverings.

Proposition 1. *Let $\mathcal{C}_p = \{\Omega_1, \dots, \Omega_p\}$ be a (quasi-)admissible covering for (Ω, X_n). Then, if $p \geq 2$, for every patch Ω_i there exists a set Ω_j, $i, j = 1, \dots, p$, $i \neq j$, such that $\Omega_i \cap \Omega_j \neq \varnothing$.*

Proof. It is sufficient to prove the case $p = 2$. By virtue of the partitioning assumption in Definition 1, there exist $\boldsymbol{x}, \boldsymbol{y} \in \Omega$ such that $\boldsymbol{x} \in \Omega_1$ and $\boldsymbol{y} \in \Omega_2$. Then, letting $g : [0, 1] \longrightarrow \Omega$ be a continuous path from $\boldsymbol{x} = g(0)$ to $\boldsymbol{y} = g(1)$ in Ω, that exists since Ω is connected, there exists $\lambda^\star$ such that $\boldsymbol{\zeta} = g(\lambda^\star) \notin \Omega_1$ is on the boundary of Ω_1. If $\boldsymbol{\zeta} \notin \Omega_2$, then $\boldsymbol{\zeta} \notin \Omega$. Therefore, $\boldsymbol{\zeta} \in \Omega_2$ holds true. Moreover, since Ω_2 is open, there exists a neighborhood of $\boldsymbol{\zeta}$ that is contained in Ω_2. On the other hand, since $\boldsymbol{\zeta}$ is on the boundary of Ω_1, such a neighborhood necessarily contains elements that belong to Ω_1. Hence, $\Omega_1 \cap \Omega_2 \neq \varnothing$. □

The result in Proposition 1 indicates that every set of a (quasi-)admissible covering has to present some overlap with its neighboring patches. For our purposes, this is a desirable property. Indeed, the presence of overlaps is a key ingredient for *mixing* locally-defined classifiers, obtaining then a global method.

Along with a (quasi-)admissible covering, we need to introduce a family of compactly supported functions which form a partition of unity, see [35].

Definition 2. *A family of functions* $\{\omega_k\}_{k=1}^p$ *is called a partition of unity for a covering* $\mathcal{C}_p = \{\Omega_k\}_{k=1}^p$ *of* Ω *if it is a family of compactly supported, non-negative, continuous functions, with* $\operatorname{supp}(\omega_k) \subseteq \Omega_k$ *and such that* $\sum_{k=1}^p \omega_k(\boldsymbol{x}) = 1$, $\boldsymbol{x} \in \Omega$.

The Definition 2 is satisfied, for instance, by the Shepard's weights [30], i.e.

$$\omega_k(\boldsymbol{x}) = \frac{\bar{\omega}_k(\boldsymbol{x})}{\sum_{j=1}^{p} \bar{\omega}_j(\boldsymbol{x})}, \quad k = 1, \ldots, p, \tag{7}$$

where $\bar{\omega}_k$ are compactly supported functions, with support on Ω_k. The role played by such compactly supported functions is weighting and assembling the contributions of the local classifiers. Here and in our experiments, we take the Wendland's C^2 function $\bar{\omega}(\boldsymbol{x}, \boldsymbol{y}) = \left(1 - \bar{\gamma}\mu(\boldsymbol{x}, \boldsymbol{y})\right)_+^4 \left(4\bar{\gamma}\mu(\boldsymbol{x}, \boldsymbol{y}) + 1\right)$, $\boldsymbol{x}, \boldsymbol{y} \in \Omega$,

where $(\cdot)_+$ denotes the truncated power function and $1/\bar{\gamma} > 0$ is the support of the kernel. Note that, if μ coincides with the Euclidean distance, then the Wendland's weight results in a standard RBF.

3.2. The LGSVM classifier

For simplifying the presentation of the LGSVM algorithm, in what follows we first deal with the construction of such a method when an admissible covering for (Ω, X_n) is at the disposal. However, in many situations the distribution of the elements of X_n on Ω does not allow a straightforward construction of an admissible covering. In that case, as we will point out, the LGSVM classifier has to accomplish the case of quasi-admissible coverings too. In the following, without loosing generality, we restrict to the binary classification setting with $\ell = 2$, where $C_1 = \{y = -1\}$ and $C_2 = \{y = +1\}$. Indeed, the LGSVM method generalizes to the multiclass framework according to the standard SVM algorithm.

The case of admissible coverings

Let $\Omega \subset \mathbb{R}^d$, Z_n be a training dataset and let $\mathcal{C}_p$ be an admissible covering for (Ω, X_n) according to Definition 1. With some abuse of notation, in what follows we denote by $\Omega_k \coloneqq \Omega \cap \Omega_k$, $k = 1, \ldots, p$, the subdomains. First, in order to construct the LGSVM classifier, different SVMs are trained separately and locally on each subdomain. More precisely, let n_k be the number of training elements in Ω_k, $\boldsymbol{x}_i^{(k)} \in X_{n_k} = X_n \cap \Omega_k$ and $y_i^{(k)}$ the associated label, with $i = 1, \ldots, n_k$, we solve (5) on each subdomain Ω_k. We then obtain

$$h_k(\boldsymbol{x}) = \sum_{i=1}^{n_k} \alpha_k^i y_i^{(k)} \kappa(\boldsymbol{x}, \boldsymbol{x}_i^{(k)}) + b_k, \quad \boldsymbol{x} \in \Omega, \tag{8}$$

for each $k = 1, \ldots, p$; see (6). We point out that $\boldsymbol{\alpha}_k \coloneqq (\alpha_k^1, \ldots, \alpha_k^{n_k})$ and b_k are the vector of *local* Lagrange multipliers and the bias, respectively. Then, the

local-to-global step is performed by means of the PU structure (see Definition 2). Indeed, for $\boldsymbol{x} \in \Omega$, the *global* decision function of the LGSVM classifier is given by

$$s(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{k=1}^{p} h_k(\boldsymbol{x})\omega_k(\boldsymbol{x})\right), \tag{9}$$

where ω_k is the function defined as in (7) and h_k is the hyperplane computed in (8). We now argue about the fact that the global LGSVM classifier preserves the local conditions on the training set in the hypothesis in which misclassification errors do not occur, i.e. the *local* vector of slack variables $\boldsymbol{\xi}_k := (\xi_k^1, \ldots, \xi_k^{n_k}) \equiv \mathbf{0}$, $k = 1, \ldots, p$. This is almost an unrealistic assumption in the classification setting, but at the same time, the result tells us that the LGSVM scheme can effectively solve many subproblems carrying out the local information to the global classifier; refer to [35].

Proposition 2. *In the assumptions of this section, let us suppose that* $\boldsymbol{\xi}_k \equiv \mathbf{0}$, $k = 1, \ldots, p$. *Then, for* $\boldsymbol{x}_i \in X_n$, $s(\boldsymbol{x}_i) = y_i$.

Proof. Since the constraints for the primal problem are $y_i h_k(\boldsymbol{x}_i) \geq 1$, for $\boldsymbol{x}_i \in X_n$ (see (4)), we obtain $s(\boldsymbol{x}_i) = \operatorname{sign}\left(\sum_{k=1}^{p} h_k(\boldsymbol{x}_i)\omega_k(\boldsymbol{x}_i)\right) = y_i$, where the last equality follows from the fact that the weights are positive on their support and form a partition of unity. □

The above proposition provides the theoretical evidence of the fact that the LGSVM truly defines a global classifier whose accuracy depends on the local ones.

The case of quasi-admissible coverings

While the construction of quasi-admissible coverings is trivial, the representation assumption in Definition 1 may be too restrictive in many applications: indeed, the presence of one-class clusters in the data distribution is common.

A straightforward possibility consists in modifying a quasi-admissible covering in order to make it admissible. In view of that, one could adaptively increase the patch sizes until the resulting covering satisfies the representation assumption. However, this is not a proper solution for the LGSVM algorithm. Indeed, doing in this way excessively large subdomains may occur, thus undermining the main purpose of the LGSVM method. Therefore, we investigate a possible extension of the LGSVM setting to quasi-admissible coverings, which implies that we need to face the problem of managing the case of one-class subdomains, for which local SVM classifiers are not definable. Moreover, we also deal with empty patches, i.e. patches that do not contain any training element. Let us suppose that $\widetilde{p}$ patches have examples of both classes, i.e. they satisfy the representation assumption, where $\widetilde{p} \in \mathbb{N}$, $\widetilde{p} \leq p$. Moreover, we write $\widetilde{\Omega}_j$ in place of Ω_j if it satisfies such assumption. Then we define the union of such subdomains

$$\mathcal{M} \coloneqq \bigcup_j \widetilde{\Omega}_j, \tag{10}$$

and we refer to it as the *admissible subcovering*. The set $\mathcal{M}$ might consist of disjoint unions of overlapping sets, where local SVM models are defined. In other words, on the admissible subcovering we construct the LGSVM classifier as explained in the previous sections. On the other hand, if $\bar{\boldsymbol{x}} \in T_m$ is such that $\bar{\boldsymbol{x}} \in \Omega \setminus \mathcal{M}$, we face two possibilities:

1. The test element $\bar{\boldsymbol{x}}$ is contained in one-class patches. In this case, we assign $\bar{\boldsymbol{x}}$ to the class that characterizes the subdomains it belongs to. This is a natural choice that preserves the *local purposes* of the presented method.

2. The test element $\bar{\boldsymbol{x}}$ lies on empty subdomains. Here, we classify it by means of a *Nearest Centroid* (NC) classifier [32], which takes advantage of being *fast*.

We point out that other auxiliary classifiers might be taken into consideration and employed. The choice for NC classifier mainly relies on its simplicity and on low memory requirements. We now focus on the geometric properties of the subdomains.

3.3. Subdomains as μ-balls

In what follows, we provide a possible way to define (quasi-)admissible coverings. Let $r \in \mathbb{R}^+$. We consider coverings where each patch Ω_k is an open μ-ball, i.e. it is characterized by a centre $\hat{\boldsymbol{x}}_k \in \Omega$ and a radius r, i.e. $\Omega_k = \{\boldsymbol{x} \in \Omega \,:\, \mu(\hat{\boldsymbol{x}}_k, \boldsymbol{x}) \leq r\}$. As a consequence, the shape parameter of the Wendland C^2 function $\bar{\omega}$ is then set as $\bar{\gamma} = \frac{1}{r}$, so that the support of ω_k is contained in Ω_k. Among possible choices for the (semi-)metric, given $\boldsymbol{x}, \boldsymbol{y} \in \Omega$, we focus on the standard Euclidean metric $\mu_{\mathrm{E}}(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$, and on the *cosine semi-metric* which is briefly reviewed in what follows. For $\boldsymbol{x}, \boldsymbol{y} \in \Omega$, the *cosine semi-distance* is given by $\mu_{\mathrm{CS}}(\boldsymbol{x}, \boldsymbol{y}) = 1 - \mathrm{CS}(\boldsymbol{x}, \boldsymbol{y})$, where the so-called Cosine Similarity (CS) is defined as $\mathrm{CS}(\boldsymbol{x}, \boldsymbol{y}) = \dfrac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$. Since the CS represents the cosine of the angle between the two considered vectors, we observe that $0 \leq \mu_{\mathrm{CS}}(\boldsymbol{x}, \boldsymbol{y}) \leq 2$ for any $\boldsymbol{x}, \boldsymbol{y} \in \Omega$. With both (semi)-metrics the centres of the subdomains are constructed as μ-grids. We thus refer to the presented scheme as the *grid* approach. We provide further information concerning this construction in Section 4. In Figure 1, given a set of training instances, we give an illustrative example of subdomains constructed as μ-balls centered on μ-grids covering the domain $[-1, 1]^2$. Precisely, we compare the Euclidean metric and cosine semi-metric. We observe that constructing μ-grids as patch centres might be not effective for datasets having *many* features. Indeed, it is computationally demanding or even infeasible. In order to overcome this issue, in the following we provide an alternative construction of the patches.

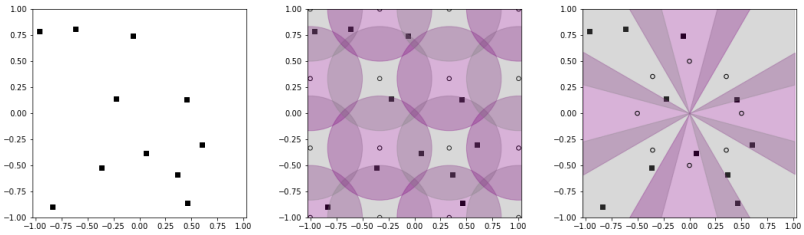

Figure 1: The grid approach. On the left, the dataset (black squares). In the center, we consider a grid of 16 centres (empty circles) and Euclidean balls with radius $r = 0.5$, while on the right we take 8 centres on a circumference and we decompose the domain by means of balls of radius $r = 1 - \cos(\pi/6)$ in the cosine semi-metric.

3.4. An alternative approach for high dimensions

Despite being defined in any dimension d, the LGSVM based on μ-grids as patch centres is not totally suitable for *high* dimensions. Therefore, we propose a second scheme, namely the *sparse* approach, for determining the subdomains of the LGSVM classifier. Here, as in Subsection 3.3, we define the patches as μ-balls in a (semi-)metric space, but we give up the purpose of covering the entire domain, proceeding then as follows. Let $p \in \mathbb{N}$, $p \geq 1$ be the number of centres and $r > 0$ be a fixed μ-radius. In the sparse approach, we choose the centres among the training samples at the disposal. With respect to the grid approach, this strategy has the following advantages and drawbacks.

- Besides missing the covering requirement, it is not trivial to guarantee some *a priori* overlap among patches. In view of this, a preliminary tuning of the parameters p and r in a validation step is recommended.

- Since test instances may be *close to* some training data, we can expect to classify a large amount of test samples without taking patches in empty regions of the domain.

- Once the patches are determined, we proceed as in the grid approach, also making use of the auxiliary NC classifier if needed.

In Figure 2, given a set of training instances, we provide an illustrative example of subdomains construed as μ-balls via the sparse approach on $[-1,1]^2$. Precisely, we compare the Euclidean metric and cosine semi-metric. This example graphically shows the difference between the grid and the sparse approach. Moreover, we sum up the presented LGSVM method in Algorithm 1 and in the flow chart reported in Figure 3.

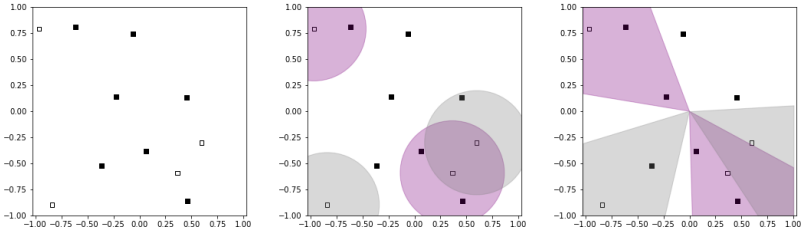

Figure 2: The sparse approach. On the left, we choose 4 centres (empty squares) among the dataset of Figure 1 (left). In the center, we use Euclidean balls with $r = 0.5$, while on the right we consider balls of radius $r = 1 - \cos(\pi/6)$ in the cosine semi-metric.

Algorithm 1 LGSVM classification

`INPUTS:` $\star$ The training dataset $Z_n = \{(\boldsymbol{x}_i, y_i),\ i = 1, \dots, n\} \subset \Omega \times \{-1, +1\}$;
$\star$ a set of μ-balls $\mathcal{C}_p = \{\Omega_1, \dots, \Omega_p\}$;
$\star$ the set $\mathcal{M}$ as in (10);
$\star$ an unknown test sample $\tilde{\boldsymbol{x}} \in \Omega$;
`Grid approach:` $\mathcal{C}_p$ is a quasi-admissible covering for (Ω, X_n);
`Sparse approach:` $\Omega_1, \dots, \Omega_p$ are centered at p distinct training elements respectively.
`OUTPUTS:` The predicted label $\tilde{y} \in \{-1, +1\}$ for $\tilde{\boldsymbol{x}}$.

`Case` $\tilde{\boldsymbol{x}} \in \mathcal{M}$`:` The element $\tilde{\boldsymbol{x}}$ is classified via $\tilde{y} = s(\tilde{\boldsymbol{x}}) = \text{sign}\left(\sum_{k=1}^{\tilde{p}} h_k(\tilde{\boldsymbol{x}})\omega_k(\tilde{\boldsymbol{x}})\right)$.

`Case` $\tilde{\boldsymbol{x}} \notin \mathcal{M}$`:` `if` $\tilde{\boldsymbol{x}}$ is contained in one-class balls only, then $\tilde{y} = y_i$.
`else` the label $\tilde{y}$ is assigned via the auxiliary classifier.

4. Complexity analysis and LGSVM implementation

For efficiently finding the set of data X_{n_k} lying on Ω_k, $k = 1, \dots, p$, we need to adopt a searching procedure. To this end, for PU interpolation and quasi-uniform nodes, the so called integer-based data structure has been introduced in

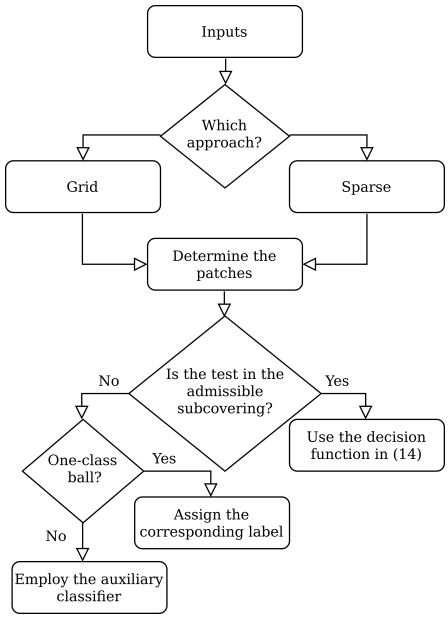

Figure 3: The flow chart corresponding to the LGSVM classification scheme as outlined in Algorithm 1.

[8, 9]. Here, since data might cluster, we consider the so-called Nearest Neighbor Search (NNS) [37], which is a widely-employed unsupervised learning strategy. In particular, such method takes advantage of the K-D Tree structure and the Ball Tree range search algorithms [2]. In the following, after investigating the complexity of LSGVMs, we also briefly discuss some implementation aspects.

4.1. Complexity analysis

The computational complexity required by our algorithm to distribute the m test points and the n training data among the subdomains using the K-D Tree structure and the associated Ball Tree range search is respectively given by (see e.g. [9]) $\mathcal{O}(d\,n\log n)+\mathcal{O}(d\,m\log m)$ and $\mathcal{O}(\log n)+\mathcal{O}(\log m)$. This step is not involved in the complexity cost of SVMs and this is the reason why the LGSVM is not suggested for a very few data. In that case some computational time is spent to organize data and it is not retrieved later when solving the

small SVM subproblems. Then, for each patch we need to solve the minimum dual problem. This is done via the SMO-type decomposition method: letting n_k the number of instances lying on the k-th patch, the algorithm requires a computational complexity of somewhere between $\mathcal{O}(n_k)$ to $\mathcal{O}(n_k^2)$ (refer to [13]). This needs to be done for all the p subdomains. Being usually $n_k \ll n$ for $k = 1, \dots, p$, this leads to a saving in term of computational complexity with respect to the classical SVM. Finally, for the prediction step, the complexity of the SVM algorithm is $\mathcal{O}(dn_{sv})$, where n_{sv} is the number of support vectors, see e.g. [11]. For the k-th patch, then the LGSVM requires a cost of $\mathcal{O}(dn_{sv_k})$. Again, usually we have $n_{sv_k} \ll n_{sv}$. Such observations on the costs of LGSVM and SVM are verified numerically by comparing the complexity times of both methods. In particular, the complexity of the LGSVM is then related to how many points lie on each patch and consequently to the radius and to the number of subdomains. Some considerations in this direction are provided in the next subsection.

4.2. Computational details

The LGSVM software, which is based upon the Python *scikit-learn* [27] scientific package, is available at `https://github.com/cesc14/LGSVM`, and we now briefly discuss some details concerning its implementation.

4.2.1. The grid approach

Given a training dataset X_n, since usually data are not quasi-uniformly distributed on the domain, estimating an *a priori* suitable number of centres is not straightforward. Therefore, we suggest to tune such hyperparameter via a validation procedure, although we provide a *safe* choice in the following. Concerning the Euclidean metric, since data can be normalized, without loosing generality we now fix the domain $\Omega = [0, 1]^d$. Then, the subdomains (balls) are

centered at a grid of p points. In our software package, the default value for the number of patches $p \in \mathbb{N}$ is given by

$$p \approx \left(\frac{n}{10}\right)^d. \tag{11}$$

Such a choice for p enables us to infer also on the radius. Indeed, focusing on the cases $d = 2, 3$, which are suitable settings for the grid approach, the covering property is guaranteed as long as the radius r is chosen as $r = \frac{\sqrt{d}\tau}{2(\sqrt[d]{p}-1)}$, where $\tau \in \mathbb{R}$, $\tau > 1$ is a hyperparameter that governs the overlap between the patches [9]. In the cosine semi-metric setting, we define the patch centres as points lying on a circumference and we standardise consistently the data at the disposal. Aside from such choice of the centres, we point out that more sophisticated solutions could be considered, e.g. taking quasi-uniformly distributed points with respect to the Hausdorff measure [14]. More precisely, here for $\boldsymbol{x} \in \Omega$ and $d \geq 2$, we consider the d-dimensional spherical coordinate system $\boldsymbol{x} = (\rho, \phi_1, \ldots, \phi_{d-1})$, where $\rho \in \mathbb{R}$, $\phi_1 \in [0, 2\pi)$, $\phi_i \in [0, \pi]$ for $i = 2, \ldots, d-1$.
Letting $m \in \mathbb{N}$, we take the angle $\theta \coloneqq \pi/m$. Therefore, the set of p centres, where p is $\mathcal{O}(m^{d-1})$, consists of the unique elements $\hat{\boldsymbol{x}}_{\boldsymbol{i}} = (a/2, i_1\theta, \ldots, i_{d-1}\theta)$, where the vector of indices $\boldsymbol{i} = (i_1, \ldots, i_{d-1})$ is such that $i_1 = 0, \ldots, 2m-1$, and $i_k = 0, \ldots, m$, for $k = 2, \ldots, d-1$. In the free-software package we fix as default choice $m \approx \left(\frac{n}{10d}\right) - 1$. Letting then $\widetilde{\theta} \coloneqq \tau\theta/2$, where the overlapping hyperparameter $\tau > 1$ is defined as in the Euclidean setting, focusing again on $d = 2, 3$ and working with rotation matrices, the covering property is guaranteed as long as $r = \cos\widetilde{\theta}$ for $d = 2$, and $r = \cos\widetilde{\theta}\big(\sin^2\widetilde{\theta} + \cos^3\widetilde{\theta}\big) + \sin\widetilde{\theta}\big(\cos^2\widetilde{\theta} - \sin\widetilde{\theta}\cos\widetilde{\theta}\big)$ for $d = 3$.

4.2.2. The sparse approach

For the sparse strategy, once the number of patches is fixed, we now deepen the choice of the centres among the training samples. First, we note that distributing the centres in all the output classes turns out to be desirable. In view of that, the number of centres assigned to a given class can be proportional to the number of training data representing such class with respect to the cardinality of the training dataset. This strategy is reasonable especially if the classes are *almost* balanced. On the opposite case, the centres might be evenly distributed among all classes. Then, once an output class is fixed, we suggest to select the centres uniformly random. This method has the main advantage of being fast and memory-saving. Furthermore, it turns out to be an effective choice, as numerically supported by our experiments.

Remark 1. *As an alternative to the random selection, one might choose the centres among the elements that are closer to data belonging to other classes. In this case, the regions that are not classified by the local SVM models are supposed to be non-critical, i.e. they do not show samples belonging to different classes. However depending on the data size, such a solution might results computationally expensive, because of the evaluation of distance matrices.*

As far as the ball radius is concerned, in the free-software package we fix $r = \frac{d}{np}$ as default choice. However, as usually done in machine learning literature, tuning such hyperparameter is recommended.

5. Numerical experiments

In the following tests, we focus on the comparison between standard SVMs and the discussed LGSVMs, showing how the proposed method effectively reduces the computational effort required by SVMs, while preserving the overall

classification performance. We consider both linear and Gaussian kernels. Moreover, for completeness, we show the results concerning the NC classifier. We take both toy and simulated datasets as well as popular benchmark test sets in machine learning literature. The hyperparameter C of the SVM algorithm and of the LGSVM local models is set to 1. As far as the the Gaussian kernel κ_G is concerned, we set the hyperparameter as $\gamma = \frac{1}{d\,\mathrm{var}(X_n)}$, where $\mathrm{var}(X_n)$ is the variance of the training set X_n and d denotes the data dimension. We point out that in the following tests we do not perform a fine tuning of the hyperparameters, since our aim is to present the overall behavior of the LGSVM classifier with respect to SVMs in different settings and by taking different parameters (see Section 5.5 for a discussion concerning the choice of the hyperparameters). In all the experiments, we evaluate the performances of the methods by means of the f_1-score [33], weighted with respect to the classes. We recall that the f_1-score is defined as the harmonic mean between precision and recall. More precisely, given the number of True Positive (TP), False Positive (FP) and False Negative (FN) cases, $\mathrm{f}_1\text{-score} = 2\,\frac{\mathrm{precision}\cdot\mathrm{recall}}{\mathrm{precision}+\mathrm{recall}}$, where $\mathrm{precision} = \dfrac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$ and $\mathrm{recall} = \dfrac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$. Note that, despite the large sizes of the considered datasets, the experiments have been carried out on a Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz processor, with 8GB RAM.

This section is organized as follows. In Subsection 5.1, we consider 2D classification tasks and we test the grid approach varying the cardinality of the dataset. In Subsection 5.2, we take a 3D dataset and we evaluate the performance of both the grid and the sparse approach. Moreover, in these subsections, we consider the Euclidean metric for the NC classifier. In Subsections 5.3 and 5.4, we increase the dimensionality of the considered data (from 48 to 2400, respectively). Therefore, for the LGSVM method we restrict to test the sparse approach with the cosine semi-metric, which is also employed by the NC clas-

sifier. Finally, in Subsection 5.5, we discuss some details concerning the key LGSVM hyperparameters.

5.1. Simulated 2D tests

In what follows, we compare SVMs with LGSVMs by considering three different simulated 2D datasets characterized by an increasing number of elements. More precisely, let $\boldsymbol{\mu}_1 \coloneqq (0,2)^\intercal,\ \boldsymbol{\mu}_2 \coloneqq (2,0)^\intercal,\ \boldsymbol{\mu}_3 \coloneqq (2,2)^\intercal$, be mean vectors and let

$$\Sigma \coloneqq \begin{pmatrix} 0.8 & 0.6 \\ 0.6 & 0.8 \end{pmatrix}, \tag{12}$$

be a covariance matrix. We consider then three classes $C_i,\ i = 1,2,3$, whose related data are from the multivariate Gaussian probability distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, respectively. More precisely, given $\boldsymbol{x} \in \mathbb{R}^2$, such distributions are characterized by the density functions $f_i(\boldsymbol{x}) = \frac{1}{2\pi\sqrt{\det(\Sigma)}} \exp\big(-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_i)^\intercal \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\big)$, for $i = 1,2,3$. Therefore, for $k \in \mathbb{N}$, $k \geq 1$, we can construct the labeled datasets $Z_k \coloneqq Z_k^1 \cup Z_k^2 \cup Z_k^3$, where $Z_k^i,\ i = 1,2,3$, is a labeled set that consists of k elements belonging to C_i, i.e. sampled from $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$. In Figure 4, we display an example of such datasets for $k = 10$. We point out that the cardinality of Z_k is then equal to $3k$.

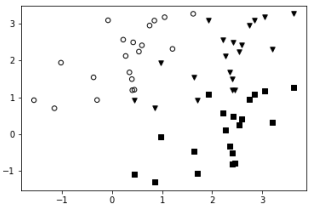

Figure 4: An example of the generated dataset Z_k. The three classes are displayed by means of empty circles (Z_{10}^1), triangles (Z_{10}^2) and squares (Z_{10}^3).

For $k = 5000, 50000, 500000$, we consider a three-class test by taking the 80% of Z_k as training set and the remaining 20% as test set, preserving the

balance between the three classes. Moreover, every dataset is standardized in a preprocessing step. In Table 1, we report the f_1-scores obtained by taking the NC and SVM classifiers, while in Table 2 we display the execution times (in seconds) for the whole process, i.e. training phase and classification test.

k	NC	SVM-lin.	SVM-Gauss.
5000	0.852	0.946	0.946
50000	0.848	0.941	0.942
500000	0.849	0.940	0.939

Table 1: The f_1-scores for the three class test obtained with NC and SVMs.

k	NC	SVM-lin.	SVM-Gauss.
5000	0.01	0.36	0.65
50000	0.02	30.34	60.15
500000	0.18	3630.13	13801.56

Table 2: The CPU times for the three class test obtained with NC and SVMs.

In Table 3, we present the results obtained via LGSVMs. More precisely, here we take the grid approach with overlapping hyperparameter $\tau = 1.1$. Moreover, concerning the number of centres, for both the Euclidean and the cosine settings, we take $p = 16$ ($k = 5000$), $p = 100$ ($k = 50000$) and $p = 400$ ($k = 500000$). As for NC and SVMs, in Table 4 we display the execution times measured in seconds.

k	LGSVM-lin. (Euclidean)	LGSVM-Gauss. (Euclidean)	LGSVM-lin. (cosine)	LGSVM-Gauss. (cosine)
5000	0.946	0.947	0.946	0.943
50000	0.941	0.941	0.941	0.941
500000	0.939	0.939	0.939	0.939

Table 3: The f_1-scores for the three class test obtained with LGSVM.

k	LGSVM-lin. (Euclidean)	LGSVM-Gauss. (Euclidean)	LGSVM-lin. (cosine)	LGSVM-Gauss. (cosine)
5000	0.89	1.16	0.17	0.24
50000	16.13	34.59	1.62	2.29
500000	327.21	754.70	34.70	52.54

Table 4: The CPU times for the three class test obtained with LGSVM.

We observe that the LGSVM method is capable to overall preserve the classification scores obtained by standard SVMs, while significantly reducing the execution times. Especially when taking the cosine semi-metric for determining the subdomains, we managed to significantly reduce the computational time, even by a factor of about 262 in the case $k = 500000$. The effectiveness and efficiency of the proposed LGSVMs are evident *at a glance* in Figure 5 (left), where we display the best performance achieved by the different methods.

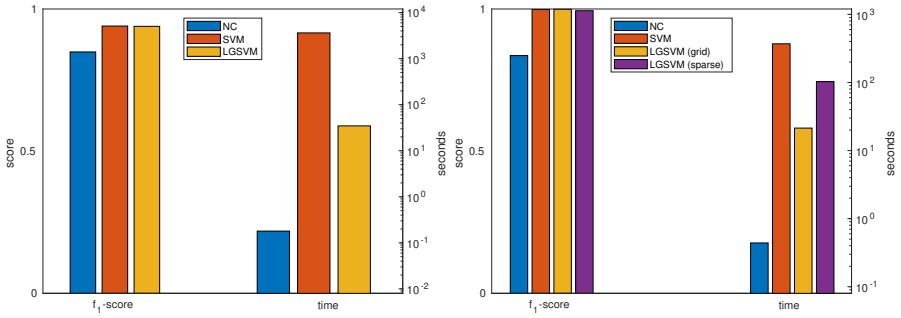

Figure 5: Left: simulated 2D tests ($k = 500000$). We display the cases of linear SVM and LGSVM with the cosine semi-metric. Right: skin segmentation dataset. We display the cases of Gaussian SVM and LGSVM computed with the grid approach ($p = 146$), and the sparse setting ($p = 80$ and $r = 0.1$). In both cases we take the the cosine semi-metric.

5.2. The Skin Segmentation dataset

We consider the *Skin Segmentation* dataset [3], which consists of 245057 samples out of which 50859 are skin samples and 194198 are non-skin samples. Each element is characterized by three features, that describe RGB colours levels. The task is to predict whether an element is a true skin sample or not. In this case, we standardize and then divide the whole dataset into 10 folds, preserving the proportions between the classes. Then, we iteratively consider 9 folds as training set and we take the remaining fold as test set. Similarly to the previous simulated cases, we evaluate the mean of the f_1-score obtained in the ten classification tasks, as well as the execution time concerning the

whole procedure, i.e. all the ten experiments. In Table 5 we report the results concerning NC and SVMs.

	f_1-score	CPU-time (s)
NC	0.836	0.44
SVM-lin.	0.931	3702.30
SVM-Gauss.	0.998	369.91

Table 5: Skin Segmentation dataset: the f_1-scores and CPU times obtained with NC and SVMs.

Here, we test LGSVMs by considering both the grid and sparse approaches.

5.2.1. Grid approach

In Table 6, we report the results obtained via LGSVMs taking the grid approach ($\tau = 1.1$) in the Euclidean metric case, while in Table 7 we consider the case of cosine semi-metric. In both settings, we take different numbers of centers and we display the percentage of test samples classified by the auxiliary NC algorithm. As in the previous experiments with simulated datasets, the LGSVM classifier is able to reduce the computational time with respect to SVMs, preserving its accuracy in terms of f_1-scores.

	p	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	125	0.00	0.995	99.49
	343	0.00	0.996	31.57
	1000	0.01	0.997	40.75
LGSVM-Gauss.	125	0.00	0.999	45.67
	343	0.00	0.999	31.91
	1000	0.01	0.999	41.74

Table 6: Skin segmentation dataset: the f_1-scores and the CPU times obtained by LGSVMs via the grid approach and using the Euclidean metric.

5.2.2. Sparse approach

In Table 8, we report the results obtained via LGSVMs adopting the sparse approach in the Euclidean metric case, while in Table 9 we consider the case of the cosine semi-metric. In this settings, we take different numbers of centers

	p	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	62	0.00	0.996	26.08
	146	0.00	0.997	20.51
	366	0.00	0.998	28.24
LGSVM-Gauss.	62	0.00	0.999	24.06
	146	0.00	0.999	21.38
	366	0.00	0.999	29.52

Table 7: Skin segmentation dataset: the f_1-scores and the CPU times obtained by LGSVMs via the grid approach and using the cosine semi-metric.

and different values for the radius of the patches. Moreover, we display again the percentage of test samples classified by the auxiliary NC scheme.

	p	r	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	40	0.1	22.80	0.917	12.09
		0.2	8.63	0.953	63.34
		0.4	1.61	0.970	2756.05
	80	0.1	15.85	0.932	16.96
		0.2	5.34	0.969	92.00
		0.4	0.82	0.978	4632.51
	160	0.1	10.76	0.950	27.34
		0.2	3.42	0.979	164.97
		0.4	0.28	0.981	9145.64
LGSVM-Gauss.	40	0.1	22.80	0.918	11.29
		0.2	8.63	0.959	36.44
		0.4	1.61	0.990	625.59
	80	0.1	15.85	0.933	18.05
		0.2	5.34	0.976	67.55
		0.4	0.82	0.995	1132.44
	160	0.1	10.76	0.952	30.05
		0.2	3.42	0.986	125.28
		0.4	0.28	0.998	2355.86

Table 8: Skin segmentation dataset: the f_1-scores and the CPU times obtained by LGSVMs via the sparse approach and using the Euclidean metric.

As far as the outcome obtained via the sparse approach is concerned, we observe that in this classification test (with $d = 3$) it is generally outperformed by the grid approach in terms of scores and computational times, as also observable in Figure 5 (right) where we choose a favourable parameters setting for each classifier. Moreover, these experiments indicate that, depending on the considered dataset and task, the trade-off between the number of centres and the value of the radius determines satisfying scores and fast execution times.

	p	r	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	20	0.1	10.59	0.964	49.49
		0.2	5.17	0.979	226.37
		0.4	0.90	0.977	1304.68
	40	0.1	6.09	0.976	84.92
		0.2	2.19	0.989	379.00
		0.4	0.48	0.979	2261.96
	80	0.1	2.98	0.991	159.33
		0.2	1.31	0.991	775.72
		0.4	0.26	0.980	4298.70
LGSVM-Gauss.	20	0.1	10.59	0.968	30.94
		0.2	5.17	0.987	76.18
		0.4	0.90	0.998	209.51
	40	0.1	6.09	0.979	51.99
		0.2	2.19	0.996	133.10
		0.4	0.48	0.998	394.52
	80	0.1	2.98	0.994	102.85
		0.2	1.31	0.997	261.00
		0.4	0.26	0.999	775.41

Table 9: Skin segmentation dataset: the f_1-scores and the CPU times obtained by LGSVMs via the sparse approach and using the cosine semi-metric.

5.3. The Adult dataset

Here we consider the *Adult* dataset [22], where the task is to predict whether a person makes more than 50,000 dollars per year. We perform the following preprocessing steps.

1. We remove the samples containing missing values.

2. We drop the feature *education*, which is highly correlated to another feature *education-num*.

3. The possible values in the *native-country* column are restricted to 1 in place of *United-States* and 0 for all the other countries.

4. All the categorical features are *one-hot* encoded. All the features are finally standardized.

The resulting elements are then described by a number of 48 numerical features. The training dataset consists of 30162 samples, where 7508 people make more than 50,000 dollars per year, while 22654 do not. The test set consists of 15060

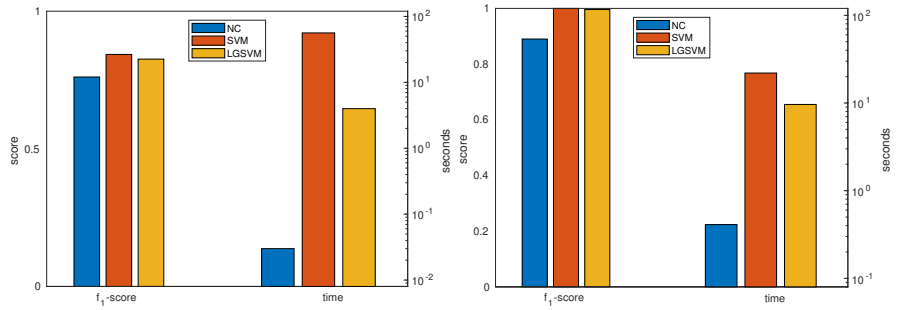

Figure 6: Left: adult dataset. We display the cases of Gaussian SVM and linear LGSVM with parameters $p = 100$, $r = 0.4$. Right: swarm dataset, Task 3. We display the cases of linear SVM and LGSVM with parameters $p = 2000$, $r = 0.4$.

elements, where 3700 people make more than 50,000 dollars per year, while 11360 do not. In Table 10 we report the results obtained via NC and SVMs, while in Table 11 we display the results achieved by LGSVMs. We observe that a proper choice of the hyperparameters determines a fast algorithm, which may speed up standard SVMs keeping comparable classification results, as we also show in Figure 6 (left).

	f_1-score	CPU-time (s)
NC	0.761	0.03
SVM-lin.	0.843	70.26
SVM-Gauss.	0.843	56.23

Table 10: Adult dataset: the f_1-scores and the CPU times obtained with NC and SVMs.

5.4. The Swarm Behaviour dataset

As last example, we take the *Swarm Behaviour* dataset available at `https://archive.ics.uci.edu/ml/datasets/Swarm+Behaviour`. This dataset was constructed by means of a survey carried out by the University of New South Wales, Australia. In the questionnaire, some short videos simulating small particles are shown to each participant, who is then asked if the particles are *flocking*, *aligned* and *grouped*; for further details, we directly refer to the survey webpage `https://unsw-swarm-survey.netlify.app`. The dataset consists of

	p	r	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	100	0.2	65.65	0.802	1.92
		0.4	27.52	0.826	3.99
		0.6	3.08	0.839	17.67
	500	0.2	29.32	0.822	6.36
		0.4	1.30	0.835	14.26
		0.6	0.12	0.840	82.93
	1000	0.2	17.24	0.827	11.25
		0.4	0.26	0.835	28.89
		0.6	0.12	0.840	161.62
LGSVM-Gauss.	100	0.2	65.65	0.801	2.46
		0.4	27.52	0.820	4.58
		0.6	3.08	0.833	18.09
	500	0.2	29.32	0.820	4.58
		0.4	1.30	0.831	17.01
		0.6	0.12	0.835	85.65
	1000	0.2	17.24	0.823	13.79
		0.4	0.26	0.831	31.69
		0.6	0.12	0.835	158.12

Table 11: Adult dataset: the f_1-scores and the CPU times obtained by LGSVMs via the sparse approach and using the cosine semi-metric.

24016 elements, which are described by 2400 numerical features extracted from a corresponding video. The dataset is standardized. Then, according to the setting provided by the authors, we consider three different binary classification tasks.

Task 1: the positive class is assigned if the particles are flocking. In this case, 12008 samples are labeled as positive.

Task 2: the positive class is assigned if the particles are grouped. Here, 9006 samples are labeled as positive.

Task 3: the positive class is assigned if the particles are aligned. In this case, 7505 samples are labeled as positive.

For each classification problem, we randomly divide the dataset into a training (90%) and test (10%) set, preserving the proportions between the classes. In Table 12 we report the results obtained for each task via NC and SVMs. The results achieved by LGSVMs are reported in Tables 13, 14 and 15. As in the previous subsection, LGSVMs are capable of leading to a saving in terms of computational time while preserving the classification performance of SVMs

(see also Figure 6, right).

		f_1-score	CPU-time (s)
Task 1	NC	0.756	0.46
	SVM-lin.	1.000	31.38
	SVM-Gauss.	0.999	205.13
Task 2	NC	0.840	0.31
	SVM-lin.	1.000	34.26
	SVM-Gauss.	1.000	167.36
Task 3	NC	0.890	0.41
	SVM-lin.	1.000	21.97
	SVM-Gauss.	1.000	35.56

Table 12: Swarm dataset: the f_1-scores and the CPU times obtained with NC and SVMs.

	p	r	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	1000	0.2	23.19	0.935	6.21
		0.4	5.25	0.985	6.11
		0.6	1.96	0.997	439.97
	2000	0.2	12.49	0.968	8.77
		0.4	2.33	0.994	9.35
		0.6	0.62	0.998	847.37
	4000	0.2	4.29	0.992	14.25
		0.4	0.75	0.999	18.81
		0.6	0.17	1.000	1734.40
LGSVM-Gauss.	1000	0.2	23.19	0.935	6.47
		0.4	5.25	0.985	6.30
		0.6	1.96	0.997	1082.82
	2000	0.2	12.49	0.968	10.22
		0.4	2.33	0.994	9.64
		0.6	0.62	0.998	2058.36
	4000	0.2	4.29	0.992	14.66
		0.4	0.75	0.999	19.65
		0.6	0.17	1.000	4016.18

Table 13: Swarm dataset, Task 1: the f_1-scores and the CPU times obtained by LGSVMs via the sparse approach and using the cosine semi-metric.

5.5. On the setting of LGSVM hyperparameters

Here, in view of the experiments we carried out, we further comment on the parameters involved in the proposed LGSVMs. In both the grid and the sparse approach, the number p of centres of the μ-balls, as well as the related radii r, need to be set. First, we notice that the hyperparameter tuning is more delicate in the case of the sparse approach rather than in the grid one. This is reasonable, since the grid approach leads to an effective domain decomposition

	p	r	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	1000	0.2	23.27	0.967	5.97
		0.4	5.12	0.997	19.09
		0.6	1.75	1.000	416.38
	2000	0.2	10.37	0.988	8.77
		0.4	1.92	0.999	33.27
		0.6	0.50	1.000	856.42
	4000	0.2	3.58	0.997	15.34
		0.4	0.46	0.999	64.99
		0.6	0.04	1.000	1764.66
LGSVM-Gauss.	1000	0.2	23.27	0.967	6.18
		0.4	5.12	0.997	30.31
		0.6	1.75	1.000	968.82
	2000	0.2	10.37	0.988	8.49
		0.4	1.92	0.999	52.66
		0.6	0.50	1.000	1872.86
	4000	0.2	3.58	0.997	14.49
		0.4	0.46	0.999	108.59
		0.6	0.04	1.000	4161.53

Table 14: Swarm dataset, Task 2: the f_1-scores and the CPU times obtained by LGSVMs via the sparse approach and using the cosine semi-metric.

that can be controlled through the overlapping parameter τ, while in the sparse case the tuning of the values of p and r is more influenced by the structure of the considered dataset. Moreover, we recommend to start by testing a severely limited number of centres and a small radius, and then to increase their values monitoring the percentage of samples classified by the auxiliary NC classifier, as well as the number of one-class μ-balls. This is an efficient way to design a proper validation range of values for each hyperparameter, since at the beginning the majority of the instances is handled by the NC algorithm, which is extremely fast. As a further insight, we observe that the CPU time required by LGSVMs is very often more sensitive to changes in the parameter r rather than p. Indeed, on the one hand by increasing p we enlarge the number of local problems to deal with, and it is likely that the size of each local problem is not significantly affected. On the other hand, by increasing r the number of local problems is not modified, but the size of each local problem is very often remarkably amplified, and the algorithm slows down consequently. Therefore, this intuition should also be taken into account in pursuing a *clever* exploration of the hyperparameters

	p	r	% aux.	f_1-score	CPU-time (s)
LGSVM-lin.	1000	0.2	22.98	0.967	6.25
		0.4	6.49	0.988	6.98
		0.6	1.79	0.994	444.35
	2000	0.2	10.99	0.988	8.88
		0.4	2.21	0.996	9.63
		0.6	0.37	0.999	939.91
	4000	0.2	4.12	0.996	14.43
		0.4	0.46	0.999	16.03
		0.6	0.08	1.000	1791.69
LGSVM-Gauss.	1000	0.2	22.98	0.967	6.19
		0.4	6.49	0.988	6.89
		0.6	1.79	0.994	1054.80
	2000	0.2	10.99	0.988	8.57
		0.4	2.21	0.996	9.65
		0.6	0.37	0.999	2059.68
	4000	0.2	4.12	0.996	14.01
		0.4	0.46	0.999	18.31
		0.6	0.08	1.000	3939.46

Table 15: Swarm dataset, Task 3: the f_1-scores and the CPU times obtained by LGSVMs via the sparse approach and using the cosine semi-metric.

space. Finally, besides classical cross validation schemes, we remark that the tuning of the hyperparameters, as well as the choice of the kernel, may benefit of more sophisticated validation strategies; concerning this topic, we refer e.g. to [17, §14.3] and to [18, 19].

6. Conclusions and future work

In this paper, we presented and discussed the LGSVM classifier, which is a global method constructed upon multiple local SVMs classification tasks that are glued together via compactly-supported weight functions. The proposed method turns out to be a valid alternative to standard SVMs. Indeed, especially in low dimensions, LGSVMs improve the standard setting in terms of execution times, without loosing the overall classification capability. The LGSVM classifier is promising and further aspects need to be investigated. For example, future work concerns theoretical studies about how the number of centres and the radius relate to the distribution of the training dataset in the domain. Moreover, different kernel bases might be used (see e.g. [7]). Finally, the pre-

sented method can be further speeded up by reducing via greedy methods [36] the training instances or by implementing parallel computing.

Acknowledgements

This research has been accomplished within the Italian Network on Approximation (RITA), the thematic group on "Approximation Theory and Applications" (TAA) of the Italian Mathematical Union (UMI) and partially funded by the GNCS-INδAM. We sincerely thank the reviewers for helping us to significantly improve the paper.

References

[1] I. Babuška, J.M. Melenk, *The partition of unity method*, Int. J. Numer. Meth. Eng. **40** (1997), 727–758.

[2] J. L. Bentley, *Multidimensional binary search trees used for associative searching*, Communications of the ACM (1975).

[3] R. Bhatt, A. Dhall, *Skin Segmentation Dataset*, UCI Machine Learning Repository.

[4] E. Blanzieri, F. Melgani, *An adaptive SVM nearest neighbor classifier for remotely sensed imagery.* In: IEEE Int. Conf. on Geoscience and Remote Sensing Symposium, 2006, 3931–3934.

[5] L. Bottou, V. Vapnik, *Local learning algorithms*, Neural Comput. **4** (1992), 888–900.

[6] E.J. Bredensteiner, K.P. Bennett, *Multicategory classification by Support Vector Machines*, Comput. Optim. Appl. **12** (1999), 53–79.

[7] C. Campi, F. Marchetti, E. Perracchione, *Learning via variably scaled kernels*, Adv. Comput. Math. **47** (2021), 51.

[8] R. Cavoretto, A. De Rossi, *A trivariate interpolation algorithm using a cube-partition searching procedure*, SIAM J. Sci. Comput. **37** (2015), A1891–A1908.

[9] R. Cavoretto, A. De Rossi, E. Perracchione, *Optimal selection of local approximants in RBF-PU interpolation*, J. Sci. Comput. **74** (2018), 1–22.

[10] F. Chang, C.Y. Guo, X.R. Lin, C.J. Lu, *Tree Decomposition for Large-Scale SVM Problems*, J. Mach. Learn. Res. **11** (2010), 2935–2972.

[11] M. Claesen, F. De Smet, J.A.K. Suykens, B. De Moor, *Fast prediction with SVM models containing RBF Kernels*, 2014, `https://arxiv.org/pdf/1403.0736.pdf`.

[12] R. Collobert, S. Bengio, Y. Bengio, *A parallel mixture of SVMs for very large scale problems*, In: NIPS, 633–640, 2001.

[13] K.L. Du, M.N.S Swamy, *Neural Networks and Statistical Learning*, Springer, London, 2010.

[14] S. De Marchi, G. Elefante, *Quasi-Monte Carlo integration on manifolds with mapped low-discrepancy points and greedy minimal Riesz s-energy points*, Appl. Numer. Math. **127** (2018), 110–124 .

[15] O. Dekel, O. Shamir, *There's a hole in my data space: Piecewise predictors for heterogeneous learning problems*, J. Mach. Learn. Res. **22** (2012), 291–298.

[16] S. Ding, X. Zhang, Y. An, Y. Xue, *Weighted linear loss multiple birth support vector machine based on information granulation for multi-class classification*, Pattern Recognit. **67** (2017), 32-46.

[17] G.E. Fasshauer, M.J. McCourt, *Kernel-based Approximation Methods Using* Matlab, World Scientific, Singapore, 2015.

[18] Y.N. Guo, X. Zhang, D.W. Gong, Z. Zhang, J.J. Yang, *Novel interactive preference-based multiobjective evolutionary optimization for Bolt Supporting Networks*, IEEE Trans. Evol. Comput. **24**(4) (2020), 750–764.

[19] J.J. Ji, Y.N. Guo, X.Z. Gao, D.W. Gong, Y.P. Wang, *Q-learning-based hyperheuristic evolutionary elgorithm for dynamic task allocation of Crowdsensing*, IEEE Trans Cybern. (2021), 1–14.

[20] T. Joachims, C.N.J. Yu, *Sparse kernel SVMs via cutting-plane training*, Mach. Learn. **76** (2009), 179–193.

[21] C. Jose, P. Goyal, P. Aggrwal, M. Varma, *Local deep kernel learning for efficient non-linear SVM prediction*, In: Proceedings of the 30th International Conference on Machine Learning, 2013, 486–494.

[22] R. Kohavi, *Scaling up the accuracy of Naive-Bayes classifiers: a Decision-Tree Hybrid*, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (1996).

[23] E. Larsson, V. Shcherbakov, A. Heryudono, *A least squares radial basis function partition of unity method for solving PDES*, SIAM J. Sci. Comput. **39** (2017), A2538–A2563.

[24] Y. Lu, V. Roychowdhury, *Parallel randomized sampling for support vector machine (SVM) and support vector regression (SVR)*, Knowl. Inf. Syst. **14** (2008), 233–247.

[25] S. Maji, A.C. Berg, J. Malik, *Efficient classification for additive kernel SVMs*, IEEE PAMI, **35(1)**, 2013.

[26] A.K. Menon, *Large-scale support vector machines: Algorithms and theory*, technical report, University of California San Diego (2009).

[27] F. Pedregosa et al., *Scikit-learn: Machine Learning in Python*, J. Mach. Learn. Res. **12** (2011), 2825–2830.

[28] B. Schölkopf, A.J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, USA, 2002.

[29] J. Shawe-Taylor, N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

[30] D. Shepard, *A two-dimensional interpolation function for irregularly spaced data*, in: Proceedings of 23-rd National Conference, Brandon/Systems Press, Princeton, 1968, 517–524.

[31] S. Sohangir, D. Wang, *Improved sqrt-cosine similarity measurement*, J. Big Data **4** (2017), 25.

[32] R. Tibshirani, T. Hastie, B. Narasimhan, G. Chu, *Diagnosis of multiple cancer types by shrunken centroids of gene expression*. Proceedings of the National Academy of Sciences of the United States of America **99**(10) (2002), 6567-6572.

[33] C.J. Van Rijsbergen, *Information Retrieval*, Butterworth-Heinemann (1979).

[34] X. Wang, F. Chung, S. Wang, *On minimum class locality preserving variance support vector machine*, Pattern Recognit. **43**(8) (2010), 2753-2762.

[35] H. Wendland, *Fast evaluation of radial basis functions: Methods based on partition of unity*, in: C.K. Chui et al. (Eds.), Approximation Theory X: Wavelets, Splines, and Applications, Vanderbilt Univ. Press, Nashville, 2002, 473–483.

[36] T. Wenzel, B. Haasdonk, G. Santin, *A novel class of stabilized greedy kernel approximation algorithms: Convergence, stability and uniform point distribution*, J. Approx. Theory **262**, 105508.

[37] P.N. Yianilos, *Data structures and algorithms for nearest neighbor search in general metric spaces*, In Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms (SODA '93), SIAM, USA (1993), 311–321.

[38] H. Zhang, A.C. Berg, M. Maire, L. Malik, *SVM-KNN: Discriminative nearest neighbor classification for visual category recognition*, In: Proc. of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognit. vol. 2, 2006, 2126–2136.

[39] H. Zhang, L. Cao, S. Gao, *A locality correlation preserving support vector machine*, Pattern Recognit. **47**(9) (2014), 3168-3178.

Francesco Marchetti got his Ph.D. at the University of Padova in 2021. He is research fellow at the Department of Mathematics "Tullio Levi-Civita" of the University of Padova. His research interests are in approximation theory and kernel methods for approximation and classification.

Emma Perracchione got her Ph.D. at the University of Torino in 2017. She received the National Scientific Abilitation as associate Professor in 2020. Currently she is research assistant at the Dipartimento di Scienze Matematiche "G.L. Lagrange" of the Politecnico di Torino. She published more than forty papers mainly on kernel methods for approximation and classification.