Received: 10 June 2020 | Revised: 28 August 2020 | Accepted: 19 September 2020
DOI: 10.1002/int.22296

RESEARCH ARTICLE

DSVD-autoencoder: A scalable distributed privacy-preserving method for one-class classification

Oscar Fontenla-Romero | **Beatriz Pérez-Sánchez** | **Bertha Guijarro-Berdiñas**

Department of Computer Science and Information Technologies, CITIC—Faculty of Informatics, Universidade da Coruña, A Coruña, Spain

Correspondence
Beatriz Pérez-Sánchez, Department of Computer Science and Information Technologies, CITIC—Faculty of Informatics, Universidade da Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain.
Email: beatriz.perezs@udc.es

Abstract
One-class classification has gained interest as a solution to certain kinds of problems typical in a wide variety of real environments like anomaly or novelty detection. Autoencoder is the type of neural network that has been widely applied in these one-class problems. In the Big Data era, new challenges have arisen, mainly related with the data volume. Another main concern derives from Privacy issues when data is distributed and cannot be shared among locations. These two conditions make many of the classic and brilliant methods not applicable. In this paper, we present distributed singular value decomposition (DSVD-autoencoder), a method for autoencoders that allows learning in distributed scenarios without sharing raw data. Additionally, to guarantee privacy, it is noniterative and hyperparameter-free, two interesting characteristics when dealing with Big Data. In comparison with the state of the art, results demonstrate that DSVD-autoencoder provides a highly competitive solution to deal with very large data sets by reducing training from several hours to seconds while maintaining good accuracy.

KEYWORDS
autoencoder, big data, distributed learning, neural network, one-class classification, privacy-preserving, singular value decomposition

1 | INTRODUCTION

In classic multiclass problems a classifier is trained using examples representing all available categories, which are known in advance, and when a new example arrives the goal is to classify it into one of those categories. However, other scenarios exist in which one class (representing the normal data or positive class) has to be distinguished from other classes with the additional difficulty that the positive class is well sampled in the training set, while the other classes (representing the abnormal data) are severely under-sampled or even nonexistent. The scarcity of abnormal examples can be due to several reasons, such as high costs to gather them, or the low frequency at which these kinds of abnormal events occur. This is a typical scenario in a wide variety of real environments and consequently this problem has gained a lot of attention through the years involving large data sets obtained from critical systems. These include, but are not limited to, the detection of medical diagnostic problems, fault detection in industrial machinery and robotics, intrusions in electronic security systems, video surveillance or document classification. To appropriately handle these types of situations the one-class classification paradigm would be more appropriate. This problem has received different denominations over the years as the already mentioned one-class classification[1] or single classification,[2] among others, arising from the different problems to which this paradigm has been applied such as outlier detection[3] or novelty detection.[4]

One of the most popular approaches for one-class classification, based on neural networks, is the autoencoder[5,6] which has been successfully employed in a wide variety of application fields. In previous years, different approaches have arisen to train autoencoders but the vast majority of them were oriented to improving classification accuracy over centralized small to medium size data sets. Nowadays, however, with the advent of new scenarios such as the Internet of Things, new challenges have arisen for the machine learning field. The more obvious challenge is related to scalability, as the data sets have grown enormously in size, thus demanding new fast efficient algorithms. Some other interesting challenges have also emerged related to certain external restrictions that those algorithms must accomplish as imposed by the conditions under which data is available for training. One of these conditions occurs when data is distributed and their privacy must be preserved between the different data providers sites. In many application domains, data might belong to several, perhaps competing, organizations that want to exchange knowledge and share the benefit of a machine-learning model that aggregates all their data but without the need of exchanging raw private data among them.[7,8] Moreover, new regulations are appearing, such as the EU General Data Protection Regulation[1], that establish the protection of natural persons with regard to the processing of personal data and on the free movement of such data. In this case, distributed learning is a promising solution. As it is also a natural way of scaling up learning algorithms, with the popularization of multicore processors and computer clusters, it has become an active and promising research line for big data learning.[7]

Nevertheless, most of the proposals are focused only on exploiting parallel processing to speed up learning or overcome memory problems, but they do not take data privacy into account. As a consequence, they work by moving data between locations during learning to obtain a final aggregate classifier. In this paper, we focus our attention on this specific context, in which distributed data is not allowed to be moved between locations, to propose a new privacy-preserving learning algorithm for autoencoders that works in a federated learning scenario. This method uses the singular value decomposition (SVD) and a cost function that measures the mean squared error before the output neurons' functions. The cornerstone of this method, called DSVD-autoencoder (distributed SVD autoencoder), is its mathematical formulation. On the one hand, it makes the optimization problem separable, thus allowing learning in parallel from each partition. On the other hand, it obtains the weights of the autoencoder by means of an analytical solution. As a result, we obtain a method that is (a) privacy-preserving, (b) distributed, (c) noniterative, and (d) hyperparameter-free, all these four characteristics being important advantages when dealing with large data sets and/or distributed data sets.

This paper is structured as follows. Section 2 contains a brief review of the main methods for one-class classification, providing a general overview of this study field. Section 3 describes certain technical issues as a necessary starting point to understand the proposed method. Section 4 describes our proposed learning method for autoencoders, DSVD-autoencoder. Section 5 analyzes the privacy preservation capacity of the proposed method in a distributed environment. Section 6 illustrates the behavior of the DSVD-autoencoder by means of a comparative study against some popular one-class classification approaches. Finally, conclusions are drawn in Section 7.

2 | RELATED WORK

Over the years a considerable number of methods have been proposed to solve the one-class classification problem. In the literature, several reviews can be found that analyze their suitability in different application fields such as mobile-masquerader detection,[9] speaker verification problem,[10] biometrics,[11] or credit-scoring systems,[12] as well as other studies that established taxonomies for one-class classification methods.[9,13,14] Most of the proposed methods are mainly focused on improving the classification accuracy and only a few of them take care of data privacy restrictions in distributed environments. One of the most interesting taxonomies for one-class classification methods was presented by Tax[15] who categorized them into three main approaches: density estimation methods, boundary methods and reconstruction methods.

Earlier research in this area was directed towards *density estimation* with parametric generative models that use a probabilistic approach to estimate the density of the positive class, assuming that low density areas in the training set have a low probability of containing positive data.[16] Examples of density methods are the Gaussian model,[17] the mixture of Gaussians,[18] Parzen density estimators,[19,20] or the local outlier factor (LOF).[21] More recently, Branch et al.[22] presented a distributed approach for wireless sensor networks where each node has a local data set with the aim of computing the set of the global top-k anomalies. The scheme is generic in that it is suitable for almost all density-based methods, except LOF. Later, Bai et al.[23] proposed the distributed local outlier factor computing method (DLC) for distributed density-based outlier computation but privacy of the data is not guaranteed since part of it must be shared between the nodes.

In a different line of research, *boundary* or discriminatory approaches optimize a closed border around the target data set and use a measure of distance to the estimated surface to detect anomalies. Most methods build the boundary around training data by using a hypersphere, a set of ellipsoids or a convex hull. One of the first approaches using hyperspheres was the k-centers algorithm.[24] Later on, several more popular methods were developed like support vector data description (SVDD)[25] or the one-class support vector machine (OC-SVM),[26] that has become one of the most successful tool for one-class classification. Subsequent extensions of SVDD and OC-SVM were proposed to improve their accuracy such as the Graph Embedded OC-SVM (GE-OC-SVM) and graph embedded SVDD (GE-SVDD)[27] or, more recently, the subspace-SVDD (S-SVDD).[28] The first approach based on deep learning is the deep-SVDD.[29] Alternatively, there are other approaches that employ a set of ellipsoids to fit the region of the data space,[30,31] or others that employ a family of convex hulls for one-class classification[32] like the well-known approximate polytope ensemble algorithm (APE).[33] Within the boundary approach, there can also be found ensembles-based methods like the one-class random forests (OCRF).[34] Lastly, for distributed environments the one-class classification (DOC-SVM)[35] was proposed as an extension of the OC-SVM. It considers several OC-SVM classifiers, each one trained using mostly their local data partition although some data are exchanged between them during learning to obtain a joint global model, therefore, exposing data-privacy.

Finally, *reconstruction* methods involve training a regression model between inputs and outputs using only the positive data. They are based on the idea that, in case a negative sample is mapped using this model, the reconstruction error at the output will be higher than the one obtained with positive data. Some of the most important reconstruction methods are k-means clustering,[17] learning vector quantization,[36] self-organizing maps,[37] principal component analysis (PCA),[17] and autoassociative encoders.[5,38,39] Based on these initial studies, additional research have resulted in proposals to improve the robustness of some methods, such as the Robust PCA[40] and direct robust matrix factorization.[41] However, proposals based on purely distributed reconstruction methods are scarce. One of the few is the approach presented by O'Reilly et al.,[42] which is based on the use of PCA and the soft-margin minimum volume ellipse. This method, called distributed minimum volume elliptical-principal component analysis is formulated as a distributed optimization problem that allows local nodes to obtain the global solution by solving subproblems of a general optimization problem and passing information about the solution to the neighbors. Within the reconstructive approach, autoassociative encoders, or simply autoencoders or diabolo networks,[5,6,43] have been widely applied in the novelty and anomaly detection problem (see, e.g., the works of Thompson et al.,[44] Sanz et al.,[45] or Miranda et al.[46]). For autoencoders, only one distributed and privacy-preserving proposal called distributed one-class learning (DOCL)[47] has been proposed for the specific case of image filtering. This model, while respecting data privacy, is made up of n independent models whose parameters are not combined in any way to create a single final model that contains the aggregate knowledge of all the local data. In this study, we make a new distributed proposal for autoassociative encoders that preserves data-privacy.

3 | PRELIMINARIES

To facilitate understanding of the proposed method, this section summarizes some previous results that will serve as a basis. All these results take as a reference model a one-layer feedforward neural network (i.e., no hidden layers). In such context, the problem of learning the weights can be separated into several independent problems, as many as outputs neurons, since for each neuron their associated weights are only related to that particular output. Therefore, in what follows, we will consider only one output neuron to avoid a cumbersome mathematical explanation, although the extension to multiple output neurons can be obtained easily. A network such as this is characterized by its weight vector $\mathbf{w} \in \mathbb{R}^{m \times 1}$, where m is the number of inputs, including the bias, and the activation function of the output neuron $f : \mathbb{R} \rightarrow \mathbb{R}$. Given a training set defined by an input matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, n being the number of data instances, and by a vector of desired outputs $\mathbf{d} \in \mathbb{R}^{n \times 1}$, the output of the network $\mathbf{y} \in \mathbb{R}^{n \times 1}$ can be obtained as $\mathbf{y} = f(\mathbf{X}^T\mathbf{w})$.

Usually, function f is nonlinear and the optimal weights are typically estimated using iterative gradient descent methods[48] attempting to minimize the error between the network's output $f(\mathbf{X}^T\mathbf{w})$ and the desired output $\mathbf{d}$. As a consequence, local minima may appear at the surface of the cost function[49,50] that complicate obtaining the global optimum. However, Fontenla-Romero et al.[51] presented a method that allows obtaining the optimal weights by means of a noniterative analytical procedure, as briefly described below.

Contrary to what most iterative learning algorithms do, which minimize the error obtained at the output of the network, the method proposed[51] is based on minimizing the mean squared error calculated *before* the nonlinear activation function, that is, the error between $\mathbf{X}^T\mathbf{w}$ and $\bar{\mathbf{d}} = f^{-1}(\mathbf{d})$. This alternative function does not contain any local minima and, as was demonstrated by the authors, its global solution can be obtained by solving a system of linear equations defined by

$$\mathbf{A}\mathbf{w} = \mathbf{b}, \tag{1}$$

$\mathbf{A}$ and $\mathbf{b}$ being described in terms of the inputs and desired outputs in the training set as:

$$\begin{aligned} \mathbf{A} &= \mathbf{XFFX}^T, \\ \mathbf{b} &= \mathbf{XFF}\bar{\mathbf{d}}, \end{aligned} \tag{2}$$

where $\mathbf{F} = diag(f'(\bar{d}_1), f'(\bar{d}_2), ..., f'(\bar{d}_n))$ is a diagonal matrix formed by the derivative of the f function in the components of vector $\bar{\mathbf{d}}$.

This approach is efficient for problems where the number of data instances n is much greater than the number of inputs m, as the size of the system of linear equations in Equations (1) and (2) depends on m. However, in the opposite situation ($m \gg n$), the computation time required to obtain the solution increases significantly, given the computational burden of calculating matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ and its inverse $\mathbf{A}^{-1}$, necessary to solve $\mathbf{w}$ in Equation (1).

To overcome this problem, an alternative learning algorithm, called LANN-SVD, was further proposed.[52] To achieve efficiency whether $n \gg n$ or $m \gg n$ this approach transforms the system detailed in Equations (1) and (2) using the SVD of $\mathbf{XF}$, which enables us to obtain a low-rank approximation of $\mathbf{XF}$ as follows:

$$\mathbf{XF} = \mathbf{USV}^T. \tag{3}$$

Given that $\mathbf{XF} \in \mathbb{R}^{m \times n}$, it holds that $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{S} \in \mathbb{R}^{m \times n}$, and $\mathbf{V} \in \mathbb{R}^{n \times n}$, being matrices $\mathbf{U}$ and $\mathbf{V}$ orthogonal. Replacing $\mathbf{XF}$ in Equations (1) and (2) by its SVD factorization we get:

$$\mathbf{USV}^T\mathbf{VS}^T\mathbf{U}^T\mathbf{w} = \mathbf{USV}^T\mathbf{F\bar{d}}. \tag{4}$$

From this equation, and after some transformations,[53] the following expression to calculate the optimal weights is obtained:

$$\mathbf{w} \approx \mathbf{U}(\mathbf{SS}^T)^{\dagger}\mathbf{U}^T\mathbf{XFF\bar{d}}, \tag{5}$$

where † stands for the Moore-Penrose pseudoinverse.

One of the properties of **S** is that it is a diagonal matrix with r nonzero elements, r being the rank of the original matrix from which S is obtained. As a consequence, the SVD is computed in practice in an economy-size (reduced) form, where $\mathbf{U} \in \mathbb{R}^{m\times r}$, $\mathbf{S} \in \mathbb{R}^{r\times r}$, and $\mathbf{V} \in \mathbb{R}^{n\times r}$. This reduced SVD contains the same information as the standard SVD and can be calculated much more efficiently. Moreover, as the maximum possible rank of matrix **XF** is $r \leq \min(m, n)$ the calculation of its reduced SVD is equally efficiently whether $m \gg n$ or $n \gg m$. As a consequence, besides being noniterative, the computational complexity of LANN-SVD to calculate **w** relies on the smaller value between the number of instances n and the number of inputs m of the training data set.

4 | PRIVACY-PRESERVING TRAINING ALGORITHM FOR AUTOENCODERS

An autoencoder is a feedforward network, like the one shown in Figure 1, which learns how to map its inputs into its output nodes, through a narrower hidden layer, with the attempt to reconstruct the input. As the network has a narrow hidden layer, it is forced to compress redundancies in the input while retaining and differentiating nonredundant information. In this way, the network is able to reduce noise in data by mapping inputs into the space of the correlation model, and then the residuals of this mapping can be employed to detect novelties in future data points. It has been shown that autoencoders generate an input/middle layer mapping equivalent to PCA when linear activation functions are employed[38] in the neurons. However, if the activation functions are nonlinear, the mapping

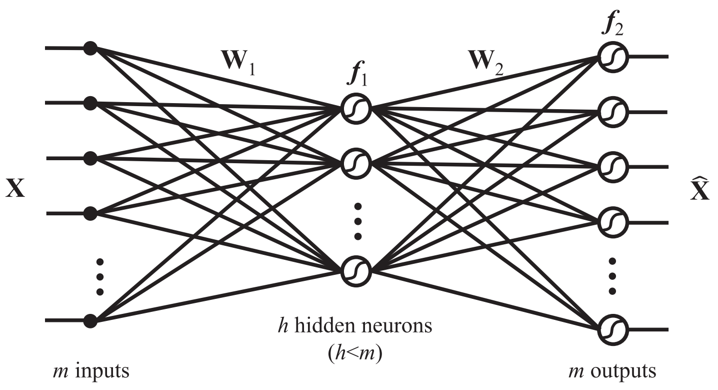

FIGURE 1 Architecture of an autoencoder neural network

has better characteristics than PCA.[54,55] Moreover, Hwang and Cho[56] analyzed the output characteristics of autoencoders and proved why nonlinearity in the hidden layer is necessary for novelty detection. This need of nonlinear functions brings, as a consequence, that the available methods to train autoencoders are of iterative nature, usually done by error back propagation using classical first-order iterative learning algorithms or fast second-order iterative methods such as scaled conjugate gradient descent. However, these methods need to have all the data stored in the same location, and therefore, they cannot be employed in a distributed scenario in which it is not possible to interchange data among different nodes for privacy reasons. As an alternative, in this section we present a non-iterative privacy-preserving learning method to train an autoencoder when the data is distributed among multiple locations.

Consider the autoencoder neural network with one hidden layer depicted in Figure 1, f_1 and f_2 being nonlinear activation functions for the hidden and output neurons. $\mathbf{W}_1 \in \mathbb{R}^{m\times h}$ and $\mathbf{W}_2 \in \mathbb{R}^{h\times m}$ are the weight matrices for the first and the second layer, respectively, where m is the number of input variables. Notice that, in this kind of network, the input and output layers have the same number of nodes (m), since the goal is to reconstruct the input data, while the number of hidden neurons (h) is always strictly lesser than the number of inputs.

Given a training set represented by an input matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, where n is the number of data samples, the outputs of the first layer $\mathbf{H} \in \mathbb{R}^{h\times n}$ can be calculated by the following equation:

$$\mathbf{H} = f_1\left(\mathbf{W}_1^T\mathbf{X}\right) \tag{6}$$

and the output of the network ($\hat{\mathbf{X}} \in \mathbb{R}^{m\times n}$) is given by:

$$\hat{\mathbf{X}} = f_2\left(\mathbf{W}_2^T\mathbf{H}\right). \tag{7}$$

Autoencoders are trained to minimize the reconstruction error between the input and the output defined usually by the following equation:

$$E(\mathbf{X}, \hat{\mathbf{X}}) = ||\mathbf{X} - \hat{\mathbf{X}}||_F^2, \tag{8}$$

where $||\cdot||_F$ is the Frobenius norm. In this case, the mean squared error loss function is more suitable than the cross-entropy loss since the aim is to reconstruct the input, as best as possible, at the output and, therefore, it is a regression problem and not a classification problem.

Due to their behavior, the weights $\mathbf{W}_1$ from the input to the hidden layer are known as the encoding weights, the weights $\mathbf{W}_2$ from the hidden to the output layer are known as the generative weights, and the activation values $\mathbf{H}$ of the hidden neurons provide a compact but approximate representation of the input vector.[6]

The method we propose to train an autoencoder when the data is distributed among multiple locations is carried out using a two-step procedure, one step for each one of the network's layers, as described below.

4.1 | Learning the first layer

In the first layer, the aim is to learn a vector space embedding of the input data extracting a meaningful but lower-dimensional representation of this data. The dimension of this new space is determined by the number of neurons (h) in the hidden layer. This can be accomplished by a low-rank matrix approximation, which is a minimization problem that tries to approximate a given matrix of data by another one (the optimization variable) subject to the constraint that the approximating matrix has reduced rank. According to the Eckart–Young–Mirsky theorem[57] the low-rank approximation with the h largest singular values is the solution with the least reconstruction error induced by any unitarily invariant norm, including the Frobenius norm. Based on this result, the rank-h SVD provides the optimal approximation to the original matrix among all rank-h matrices. Besides, every matrix is guaranteed to have a SVD and, due to its properties, this is a technique that has been widely used for data compression.[58]

Based on all of the above, we use the rank-h SVD of the input matrix $\mathbf{X}$ to obtain the weights of the first layer. As mentioned, the full SVD of $\mathbf{X}$ is a factorization of the matrix of the form:

$$\mathbf{X} = \mathbf{USV}^T, \tag{9}$$

where $\mathbf{S} \in \mathbb{R}^{m\times n}$ is a diagonal matrix with descending ordered nonnegative values on the diagonal that are the singular values of $\mathbf{X}$, whilst $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are orthogonal matrices which contain, respectively, the left and right singular vectors of $\mathbf{X}$.

In a low-rank approximation the best rank-h approximation of the original matrix $\mathbf{X}$ can be easily computed by calculating the full SVD of the matrix and then taking the first h columns of $\mathbf{U}$, truncating $\mathbf{S}$ to the first h diagonal elements, and taking the first h rows of $\mathbf{V}^T$. These new truncated matrices are denoted as $\mathbf{U}_h \in \mathbb{R}^{m\times h}$, $\mathbf{S}_h \in \mathbb{R}^{h\times h}$, and $\mathbf{V}_h^T \in \mathbb{R}^{h\times n}$, where $\mathbf{U}_h$ and $\mathbf{V}_h^T$ are, respectively, h-dimensional representations of rows (features) and columns (data samples) of $\mathbf{X}$. Hence, in our method the weights for the first layer, $\mathbf{W}_1 \in \mathbb{R}^{m\times h}$, are obtained using the $\mathbf{U}_h$ matrix from the optimal rank-h SVD for the input data ($\mathbf{X}$), as it contains the h-dimensional transformation of the input space ($\mathbb{R}^m \to \mathbb{R}^h$). A similar strategy of embedding hidden nodes by means of SVD can be found in Reference [59] for the extreme learning machine (ELM) in the context of classical classification problems.

However, when dealing with a distributed scenario a new problem arises as the data matrix ($\mathbf{X}$) is split into several nodes. Therefore, the SVD cannot be computed without compromising the privacy of the data as it is required to have all data stored in a single location. Nonetheless, the distributed calculation of the SVD can be performed if we make use of some results presented by Iwen et al.[60] They propose the partition of the original matrix $\mathbf{X}$, on which we want to apply the decomposition, into k block matrices, that is. $\mathbf{X} = [\mathbf{X}_1|\mathbf{X}_2|\cdots|\mathbf{X}_k]$, to, subsequently, compute in parallel the SVD decomposition of each of these blocks. The final SVD decomposition can be computed using, instead of $\mathbf{X}$, the concatenation of the k different local SVDs obtained from the distributed blocks. Algorithm 1 contains the steps for this DSVD of a matrix.

Algorithm 1 Distributed singular value decomposition (DSVD)

```
Inputs:
  $\mathbf{X} = [\mathbf{X}_1|\mathbf{X}_2|\cdots|\mathbf{X}_k] \in \mathbb{R}^{m\times n}$        ▷ Training data split in $k$ nodes
  $h$                                                   ▷ Rank of the SVD
Outputs:
  $\mathbf{U}_h$                          ▷ The $h$-first columns of $\mathbf{U}$ (left singular vectors)
  $\mathbf{S}_h$                          ▷ The $h$-first columns of $\mathbf{S}$ (singular values)
1: function DSVD ($\mathbf{X}, h$)
2:     for $i = 1, 2, \ldots, k$ do in parallel at node $i$
3:       $[\mathbf{U}_i, \mathbf{S}_i, \sim] = svd(\mathbf{X}_i)$;
4:       Send $\mathbf{U}_i\mathbf{S}_i$ to an arbitrary/central node
5:     end
6:     At the arbitrary/central node compute:
7:     $[\mathbf{U}, \mathbf{S}, \sim] = svd([(\mathbf{U}_1\mathbf{S}_1)|\cdots|(\mathbf{U}_k\mathbf{S}_k)])$;
8:     $\mathbf{U}_h = \mathbf{U}[:,1:h]$;                          ▷ Obtain the rank-$h$  $\mathbf{U}$
9:     $\mathbf{S}_h = \mathbf{S}[:,1:h]$;                          ▷ Obtain the rank-$h$  $\mathbf{S}$
10: end function
```

In addition, Figure 2 shows a diagram of the distributed computation of the SVD of a matrix.

Once the DSVD of $\mathbf{X}$ is computed, the weights of the first layer are obtained using the left singular vectors: $\mathbf{W}_1 = \mathbf{U}_h$. Later on, these weights are broadcasted to every node i and, using their local data, they compute the outputs of the hidden layer as follows:

$$\mathbf{H}_i = f_1\left(\mathbf{W}_1^T\mathbf{X}_i\right);\ \forall\ i = 1, \ldots, k. \tag{10}$$

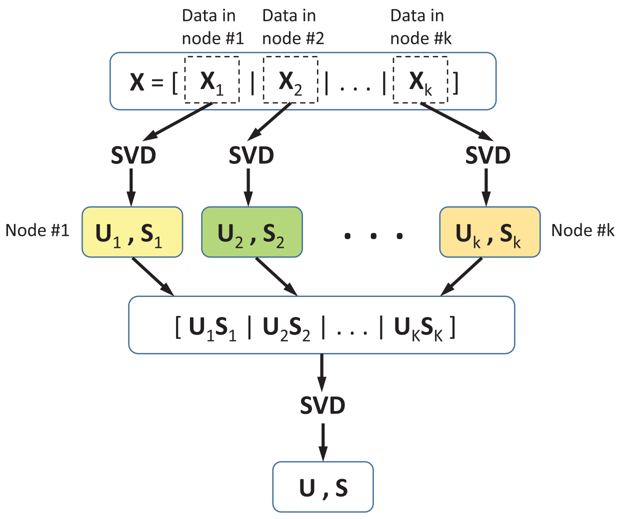

FIGURE 2 Distributed computation of the singular value decomposition of a matrix [Color figure can be viewed at wileyonlinelibrary.com]

It is important to point out that in all this process the information exchanged between nodes is always private since raw data are never sent but only encoded matrices ($\mathbf{U}_i\mathbf{S}_i$ and $\mathbf{W}_1$) from which the original data could not be deduced.

4.2 | Learning the second layer

The goal of the second layer of the network is to reconstruct the input from the low-dimensional representation provided by the output $\mathbf{H} = [\mathbf{H}_1|\cdots|\mathbf{H}_k]$ of the hidden layer computed in each node (see Equation 10). If a least-square loss function is considered, as the one previously presented in Equation (8), the optimal weights of this second layer $\mathbf{W}_2$ can be obtained by means of a minimization problem based on a sum-of-squares error function. In the case that the activation function of the output neurons is linear, the solution for this least-squares problem can be found exactly in a simple closed form using the pseudo-inverse of a matrix.[61] However, in the most general case, the activation functions are nonlinear and, therefore, this solution is no longer possible and the use of iterative algorithms is required. Although these kinds of methods are effective, a large number of steps might be needed to converge to the optimal value. This is not a very desirable behavior in problems with large data sets since they can involve large training times. To avoid this, we propose to apply the noniterative learning method LANN-SVD algorithm described in Section 3 to obtain the weights $\mathbf{W}_2$.

As mentioned, this supervised algorithm obtains near optimal weights for a one-layer neural network with non linear output functions, which corresponds to the second layer of the autoencoder. Moreover, this method allows separating the task of learning the weight matrix $\mathbf{W}_2$ into m-independent problems, as many as outputs, since for each output neuron its associated weights are only related to its inputs. Considering the jth output neuron, and taking into account that the desired outputs of the autoencoder are the $\mathbf{X}$ vector since the goal is to reconstruct the input at the output, the corresponding backpropagated desired output ($\bar{\mathbf{d}}$) needed in Equation (5) is obtained as follows:

$$\bar{\mathbf{d}} = f_2^{-1}(\mathbf{X}(j,:)^T)$$

where f_2^{-1} is the inverse of the output activation function and $\mathbf{X}(j,:)$ is the jth row of the matrix $\mathbf{X}$. In a distributed scenario the data ($\mathbf{X}$) is split in several locations or nodes, thus we propose to employ an equivalent equation obtained by using a partitioned vector from the local data ($\mathbf{X}_1, \ldots, \mathbf{X}_k$) as follows:

$$\bar{\mathbf{d}} = \begin{bmatrix} \bar{\mathbf{d}}_1 \\ \bar{\mathbf{d}}_2 \\ \vdots \\ \bar{\mathbf{d}}_k \end{bmatrix} = \begin{bmatrix} f_2^{-1}(\mathbf{X}_1(j,:)^T) \\ f_2^{-1}(\mathbf{X}_2(j,:)^T) \\ \vdots \\ f_2^{-1}(\mathbf{X}_k(j,:)^T) \end{bmatrix}$$

In addition, in Equation (5) the backpropagated desired output must be weighted by the derivative of the output activation function ($\mathbf{f}_d$) that can be calculated using the following equation:

$$\mathbf{f}_d = \begin{bmatrix} \mathbf{f}_{d1} \\ \mathbf{f}_{d2} \\ \vdots \\ \mathbf{f}_{dk} \end{bmatrix} = \begin{bmatrix} f_2'(\bar{\mathbf{d}}_1) \\ f_2'(\bar{\mathbf{d}}_2) \\ \vdots \\ f_2'(\bar{\mathbf{d}}_k) \end{bmatrix} = f_2'(\bar{\mathbf{d}}),$$

where f_2' is the derivative of the neural function at the output of the autoencoder. Once these values are obtained, using Equation (5) the optimal weights for the jth output neuron can be approximated by the following formula[52]:

$$\mathbf{W}_2(:,j) \approx \mathbf{U}(\mathbf{S}\mathbf{S}^T)^\dagger \mathbf{U}^T \mathbf{H}(\mathbf{f}_d.*\mathbf{f}_d.*\bar{\mathbf{d}}) \tag{11}$$

where $\mathbf{U}$ and $\mathbf{S}$ are matrices obtained by means of a SVD of the $\mathbf{H}*diag(\mathbf{f}_d)$ matrix and the .* notation is used for the Hadamard product. Notice that, for computational convenience, the diagonal matrix $\mathbf{F}$ in Equation (5) has been converted into a vector in Equation (11).

In this study, firstly we propose to employ the presented DVSD to compute the SVD of $\mathbf{H}*diag(\mathbf{f}_d)$. Note that to compute the DSVD of the $\mathbf{H}*diag(\mathbf{f}_d)$ matrix it is not necessary to send and store all the blocks of $\mathbf{H}$ (i.e., $\mathbf{H}_1$, ..., $\mathbf{H}_k$) and $\mathbf{f}_d$ together. In addition, we propose to use the following equivalent expression for Equation (11) using only the local information at each node by means of a block partitioned matrix product that involves only algebra on submatrices of the factors:

$$\mathbf{W}_2(:,j) \approx \mathbf{U}(\mathbf{S}\mathbf{S}^T)^\dagger \mathbf{U}^T [\mathbf{H}_1|\cdots|\mathbf{H}_k] \left(\begin{bmatrix} \mathbf{f}_{d1} \\ \mathbf{f}_{d2} \\ \vdots \\ \mathbf{f}_{dk} \end{bmatrix} .* \begin{bmatrix} \mathbf{f}_{d1} \\ \mathbf{f}_{d2} \\ \vdots \\ \mathbf{f}_{dk} \end{bmatrix} .* \begin{bmatrix} \bar{\mathbf{d}}_1 \\ \bar{\mathbf{d}}_2 \\ \vdots \\ \bar{\mathbf{d}}_k \end{bmatrix} \right) \tag{12}$$

or equivalently:

$$\mathbf{W}_2(:,j) \approx \mathbf{U}(\mathbf{S}\mathbf{S}^T)^\dagger \mathbf{U}^T [\mathbf{H}_1(\mathbf{f}_{d1}.*\mathbf{f}_{d1}.*\bar{\mathbf{d}}_1) + \cdots + \mathbf{H}_k(\mathbf{f}_{dk}.*\mathbf{f}_{dk}.*\bar{\mathbf{d}}_k)], \tag{13}$$

where the weights ($\mathbf{W}_2$) can be computed in a central node using the available matrices from the DSVD ($\mathbf{U}$ and $\mathbf{S}$) and the block-wise multiplications of $\mathbf{H}$, $\mathbf{f}_d$, and $\bar{\mathbf{d}}$ provided by each node.

Having described the training process of both layers, in Algorithm 2 we present all the steps of the proposed method in detail. Lines 2 to 13 contain all the necessary steps to obtain the optimal weights. Steps 14 to 18 are employed to determine a threshold on the reconstruction error to classify future data. If a negative (anomalous) sample is mapped using the trained model then the reconstruction error will be higher than the one obtained with a positive (normal) training sample. In this case, we have used a threshold based on the value of the *p*th percentile of the reconstruction error achieved over the training data. The *p*th percentile is the value below which a percentage *p* of the reconstruction errors calculated for the training set falls. Although we have opted for this option, other strategies could be employed to determine the threshold.

Finally, Algorithm 3 contains the steps to classify new data after training. As explained, the threshold is the parameter which establishes the upper limit in the reconstruction error for a data point to be considered as belonging to the positive class.

In the interest of reproducible research we have made the Matlab code available at https://github.com.

Algorithm 2 Training of the privacy-preserving DSVD-autoencoder

```
Inputs:
  $\mathbf{X} = [\mathbf{X}_1|\mathbf{X}_2|\cdots|\mathbf{X}_k] \in \mathbb{R}^{m \times n}$      ▷ Input data ($m$ inputs $\times n$ samples) split in $k$ nodes
  $f_1$      ▷ Activation function for the 1st layer
  $f_2$      ▷ Activation function (invertible and derivable) for the 2nd layer
  $h$      ▷ Number of hidden neurons ($h < m$)
  $p$      ▷ $P^{th}$ percentile for the decision threshold ($0 < p < 100$)
Outputs:
  $\mathbf{W}_1 \in \mathbb{R}^{m \times h}$      ▷ Optimal weights for the first layer
  $\mathbf{W}_2 \in \mathbb{R}^{h \times m}$      ▷ Optimal weights for the second layer
  $th_p \in \mathbb{R}$      ▷ Decision threshold for one-class classification
1: function DSVD-AUTOENCODER $f_1, f_2, h, p$
2:   $[\mathbf{U}_h, \sim, \sim] = DSVD(\mathbf{X}, h)$;      ▷ Rank-$h$ SVD (distributed)
3:   $\mathbf{W}_1 = \mathbf{U}_h$;      ▷ Optimal weights for the first layer
4:   Send $\mathbf{W}_1$ to the $k$ nodes.
5:   for $i = 1$ to $k$ do in parallel at node $i$
6:     $\mathbf{H}_i = f_1\left(\mathbf{W}_1^T\mathbf{X}_i\right)$      ▷ Outputs of the hidden layer for each node
7:   end      ▷ Note that $\mathbf{H} = [\mathbf{H}_1|\cdots|\mathbf{H}_k]$
8:   for $j = 1$ to $m$      ▷ One subproblem for each output neuron
9:     Compute in each node $i$: $\bar{\mathbf{d}}_i = f_2^{-1}(\mathbf{X}_i(j,:)^T)$; $\forall\ i = 1, \ldots, k$
10:     Compute in each node $i$: $\mathbf{f}_{di} = f'_2(\bar{\mathbf{d}}_i)$; $\forall\ i = 1, \ldots, k$
11:     $[\mathbf{U}, \mathbf{S}, \sim] = DSVD(\mathbf{H}^*diag(\mathbf{f}_d), fullrank)$;      ▷ Economy size DSVD
12:     Compute $\mathbf{W}_2(:,j)$ using Equation (13);
13:   end
14:   for $j = 1$ to $n$ do in the node to which the data $j$ belongs
15:     $\hat{\mathbf{X}}(:,j) = f_2\left(\mathbf{W}_2^T f_1\left(\mathbf{W}_1^T\mathbf{X}(:,j)\right)\right)$      ▷ Output of the neural network
16:     $\mathbf{e}(j) = ||\mathbf{X}(:,j) - \hat{\mathbf{X}}(:,j)||^2$      ▷ Reconstruction error for sample $j$
17:   end
18:   $th_p = percentile(\mathbf{e}, p)$      ▷ The threshold is the $p^{th}$ percentile of $\mathbf{e}$
19: end function
```

Algorithm 3 Classification stage for the DSVD-autoencoder

```
Inputs:
  $\mathbf{x} \in \mathbb{R}^{m\times 1}$                ▷ New sample to classify
  $\mathbf{W}_1, \mathbf{W}_2$                ▷ Optimal weights for the 1st and 2nd layers
  $th_p$                ▷ Decision threshold
Outputs:
  $Result \in \{Positive, Negative\}$                ▷ Classification
1: function CLASSIFICATION-BY-DSVD-AUTOENCODER $\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, th_p$
2:    $\mathbf{h} = f_1\left(\mathbf{W}_1^T\mathbf{x}\right)$                ▷ Output of the hidden layer
3:    $\hat{\mathbf{x}} = f_2\left(\mathbf{W}_2^T\mathbf{h}\right)$                ▷ Output of the network
4:    $e_{test} = \| \mathbf{x} - \hat{\mathbf{x}} \|^2$                ▷ Reconstruction error
5:    if ($e_{test} < = th_p$) then
6:      $Result = Positive$                ▷ $\mathbf{x}$ belongs to the positive class
7:    else
8:      $Result = Negative$                ▷ $\mathbf{x}$ does not belong to the positive class
9: end function
```

5 | PRIVACY THREAT MODEL

To analyze the privacy preservation capacity of the proposed method in a distributed environment, we consider two privacy threat scenarios which are based on the ideas presented by Shokri and Shmatikov,[62] Hitaj et al.,[63] and Zhao et al.[64] about direct and indirect leaks.

5.1 | Preventing direct leakage

A first threat scenario contemplates loss of privacy due to direct exposure of local data to any other participant in the distributed environment. This exposure is what happens in a conventional machine learning scenario since to carry out the training with all the data, each participant should send their local (private) data to a centralized node. This potentially confidential information could be leaked within the organization owning the centralized node or by external attackers who capture the information through the communications network.

In the proposed method, on one hand, participants do not reveal their data sets to anyone during the training process, thus ensuring strong privacy of their data. This fact can be observed by analyzing algorithms 1 and 2 in detail. First, by analyzing lines 2–4 of algorithm 1, it can be seen that each node computes a SVD using local data and then sends the product matrix ($\mathbf{U}_i\mathbf{S}_i$) to a central node. Since the matrix $\mathbf{V}_i$, which is part of the SVD factorization, is not calculated or sent, it is not possible to obtain the local data $\mathbf{X}_i$ from the $\mathbf{U}_i\mathbf{S}_i$ matrix using the expression of factorization, that is, $\mathbf{X}_i = \mathbf{U}_i\mathbf{S}_i\mathbf{V}_i^T$. Second, by analyzing lines 2–4 of algorithm 2, it can be seen that the weights of the first layer ($\mathbf{W}_1$) are calculated using aggregated information of the local SVDs, therefore, by sharing this information among the participants, no local data could be retrieved either. In addition, lines 5–7 perform only local computations and lines 8–13 perform the computation of the weights of the second layer (Equation 13) using the available matrices from the DSVD ($\mathbf{U}$ and $\mathbf{S}$) and the multiplications $\mathbf{H}_i{*}(\mathbf{f}_{di}.*\mathbf{f}_{di}.*\bar{\mathbf{d}}_i)$ provided by each i node.

From the result obtained by this product of matrices neither it is possible to obtain the local data ($\mathbf{X}_i$) with which it was generated.

On the other hand, as a global unique model is obtained after training, in the model operation phase carried out by algorithm 3, all participants can use this model locally and privately, without any communication with other participants and without revealing the new input data to be classified, nor the model's output, to anyone. Therefore, there is no data leakage in the operation phase either.

5.2 | Preventing indirect leakage

The second threat scenario contemplates loss of privacy due to indirect leakage through an adversary that pretends to be an honest participant in the distributed learning protocol but aims to extract training data from other participants via shared calculations/parameters of the neural network during the training process. Specifically, we consider the following scenario:

- It is assumed that k participants ($k \geq 2$) train a collective model following the proposed distributed training algorithm described in algorithms 1 and 2.
- The adversary works as one of the participants but its hidden objective is to infer the local training data (private) of the other participants.
- All participants agree in advance on a common learning objective and, thus, the adversary has knowledge about the structure of the neural network: number of inputs, outputs, and hidden neurons and also the type of the activation functions.
- The adversary follows the collaborative learning protocol, sending, and receiving the required calculations.
- The adversary is active since he tries to build some kind of model to learn or infer the training data employed by other participants.

Following this scenario, previous works designed attack models based on inverse methods[65] or generative adversarial networks (GAN).[63] They have shown that most of the collaborative deep learning methods are vulnerable to attacks since it is possible to obtain private information (training data) of the participants. Specifically, the attack designed by Hitaj et al.[63] exploits the real-time nature of the learning process to train a GAN that generates prototypical samples of the targeted training set that was meant to be private. This is possible because the distributed learning procedures are usually based on iterative algorithms, with stochastic gradient sharing, that needs to run for several rounds before it is successful. However, the proposed method is robust against this type of threat for the following reasons:

- One of the fundamental premises for the attack based on generative models is that the collaborative learning method is iterative, so that it is possible to carry out the GAN training by means of the intermediate calculations obtained in each step. However, this premise is not fulfilled for the proposed method since it is not iterative. It allows obtaining the parameters in a single step, putting together the calculations of all participants, without performing a repetitive process until convergence is achieved. Therefore, it is not possible to train a GAN model to try to violate the privacy of the data in the proposed distributed algorithm.

- Unlike deep learning models, no gradients are shared. The information shared between the participants, in a single step, is the product of the $\mathbf{U}_i$ and $\mathbf{S}_i$ matrices of the SVD, to obtain the weights of the first layer, and the product $\mathbf{H}_i * (\mathbf{f}_{di}. * \mathbf{f}_{di}. * \bar{\mathbf{d}}_i)$ to obtain the weights of the second layer. As already mentioned in the previous subsection, using only $\mathbf{U}_i$ and $\mathbf{S}_i$ is not possible to obtain the local data $\mathbf{X}_i$ since the information of the matrix $\mathbf{V}_i$, that is necessary to complete the factorization $\mathbf{X}_i = \mathbf{U}_i\mathbf{S}_i\mathbf{V}_i^T$, is never shared between participants.

Therefore, it can be affirmed that the proposed method is robust against these types of privacy threats.

6 | EXPERIMENTAL ANALYSIS

To check whether the distributed and privacy-preserving features of the algorithm do not come with a degradation in performance, in this experimental study the DSVD-autoencoder is compared to some other well-known one-class methods available in the literature. Although they are not privacy-preserving algorithms they are the best options to demonstrate whether DSVD-autoencoder is competitive with respect to classification accuracy.

6.1 | Experimental conditions

For this experimental study several data sets were employed. As it is quite difficult to obtain public real-world data sets with labeled anomalies, it is quite common in this field to test anomaly detection using classification-oriented data sets that have to be previously re-purposed: whether the data sets correspond to binary or multiclass classification problems, one target class will be selected to be the normal (positive) class and the remaining ones as the anomalous (negative) class. To check the behavior and scalability of the proposed method we experimented on both small and large data sets. The data sets and their main characteristics are summarized in Table 1. All of them are available online in the same repository[66] with the exception of the Hearthstone set.[67] We employed only nonanomalous samples for training, as this is the usual situation in a real world environment. Afterwards, the trained models were tested using a mixture of anomalous and nonanomalous samples. The last column in Table 1 shows the ratio of anomalous samples in the test set for each data set.

To obtain reliable results, for small data sets 10 reruns of a 10-fold cross-validation were carried out, using 90% of the normal samples for training and keeping the remaining 10% of normal samples for testing. However, since the execution times considerably increase as the volume of data grows, for large data sets five repetitions of a holdout validation were carried out, using 70% of the normal samples for training and the remaining 30% for the test set. In any case, all the anomalous samples were added to the test sets resulting in the anomalous test sample rates shown in Table 1. In all cases, the area under the curve (AUC) was calculated as the performance measure.

TABLE 1 Characteristics of the data sets for the experimental study

Data set	#Samples	#Features	Positive class	Anomalous test samples rate
Arrhythmia	420	278	Normal	0.884
Breast	683	10	Benign	0.844
German credit	1000	20	Good	0.811
Car	1728	6	Unacc	0.959
Abalone	4177	10	1–8	0.836
MiniBooNE	130,064	50	Background	0.895
Covertype	286,048	12	Lodgepole pine	0.031
Susy	5,000,000	18	Background	0.738
Hearthstone	2,000,000	44	Win	0.766
Higgs	11,000,000	28	Signal	0.747

Note: Samples with missing values were omitted.

Regarding the algorithms, we have selected some of the most well-known one-class methods for the comparative study. More specifically,

- LOF[68] with Hamming Distance (LOF Hamming). This method assigns anomaly scores to the samples based on the distances among them within a local neighborhood.
- One-class Support Vector Machine[69] with radial basis functions as kernels (OC-SVM RBF).
- AUTO-NN trained with the classical iterative learning method based on gradient descent.[70]
- APE[33] algorithm. This method defines the boundary of the target class by means of the geometrical concept of Convex-Hull built on a n-dimensional space to which some of the input samples are projected.

Unlike the proposed method (DSVD-AUTO), most of the mentioned algorithms have several hyperparameters to tune. With the aim to obtain the best behavior of each algorithm a grid search was performed to obtain the optimal values for them. In the interest of reproducible research, the appendix includes the values of the hyperparameters employed for each data set.

All the experiments were carried out on an Intel Core i7-7700 processor (with 3.60 GHz clock speed, four cores and support for eight threads) and 16 Gb RAM memory.

6.2 | Results

Table 2 shows the mean AUC obtained by each method over the several simulations run for each of the data sets of small size (*SD*s from the mean where omitted being their order of magnitude of 10^{-2}). Absolute best results are boldfaced. Due to the high temporal demands required for the adjustment of hyperparameters and the training of the algorithms, for the comparison study over large data sets we only considered the method that showed the best general behavior over small data sets (i.e., OC-SVM) together with the regular learning method for autoencoders (i.e., AUTO-NN), since our proposal is an alternative algorithm for the

TABLE 2 Mean AUC obtained using the small data sets

Data set	LOF (Hamming)	OC-SVM (RBF)	AUTO-NN	APE	DSVD-AUTO
Arrhythmia	0.698	**0.748**	0.739	0.746	0.721
Breast	0.798	0.941	0.966	0.951	**0.967**
German credit	0.564	**0.645**	0.575	0.586	0.583
Car	0.257	**0.790**	0.711	0.718	0.719
Abalone	0.693	**0.812**	0.767	0.665	0.769
Mean	0.562	**0.781**	0.741	0.724	0.742

Note: Absolute best results are boldfaced.

Abbreviations: APE, approximate polytope ensemble; AUC, area under curve; AUTO, autoencoder; DSVD, distributed singular value decomposition; LOF, local outlier factor; RBF, radial basis function; OC-SVM, one-class support vector machine.

learning process of that model. These additional results are shown in Table 3. As can be observed, in general, the most effective training algorithm in terms of AUC is OC-SVM. However, the values obtained by DSVD-AUTO do not differ significantly from those of the OC-SVM, bearing in mind also that the proposed method allows distributed learning while OC-SVM does not. Besides, the proposed algorithm compared to the usual training method employed in autoencoders (AUTO-NN), based on gradient descent, shows better performance in all cases.

As we wanted to prove that DSVD-AUTO is it not only an effective method but also an efficient one, computational time required for training was also measured. Table 4 shows the results obtained when training with the large data sets using the best combination found for the hyperparameters. In this case, the proposal method was distributed over the eight available threads. As can be seen, the training speed of DSVD-AUTO is noticeable. For the most demanding cases, its running times can be counted in seconds while AUTO-NN needs hours and OC-SVM needs days or even a week to train. Moreover, the high training times of OC-SVM also makes it difficult to fine tune its hyperparameters for which not only an exhaustive adjustment like grid search but even a good heuristic search can be unfeasible. Therefore, as a summary, we can conclude that the proposed method offers a high training speed without implying accuracy loss when compared to the best state of the art algorithms for one-class classification.

TABLE 3 Mean AUC obtained using the large data sets

Data set	OC-SVM (RBF)	AUTO-NN	DSVD-AUTO
MiniBooNE	**0.678**	0.498	0.652
Covertype	0.979	0.725	**0.985**
Susy	**0.760**	0.636	0.748
Hearthstone	**0.572**	0.508	0.549
Higgs	**0.570**	0.501	0.545
Mean	**0.697**	0.567	0.678

Note: Absolute best results are boldfaced.

Abbreviations: AUC, area under curve; AUTO, autoencoder; DSVD, distributed singular value decomposition; RBF, radial basis function; OC-SVM, one-class support vector machine.

TABLE 4 Computational time (s) employed by the methods for training

Data set	OC-SVM (RBF)	AUTO-NN	DSVD-AUTO
MiniBooNE	26.29	483.41	0.04
Covertype	1.38×10^3	10.20	0.04
Susy	1.93×10^5	1.76×10^4	0.50
Hearthstone	3.16×10^4	460.03	0.07
Higgs	6.23×10^5	1.08×10^4	1.47

Note: Results correspond to the mean time over five runs.

Abbreviations: AUTO, autoencoder; DSVD, distributed singular value decomposition; RBF, radial basis function; OC-SVM, one-class support vector machine.

7 | CONCLUSIONS

New machine learning techniques and tools are continuously appearing to deal with the increasing amount of available data. In this paper, we have presented the DSVD-autoencoder, a new distributed learning algorithm for autoencoder neural networks that allows handling large-scale data sets with data privacy restrictions. Several experiments have been performed over the domain of one-class classification and the effectiveness of the method has been demonstrated compared to some of the reference methods in the literature. The main advantages of the proposed DSVD-autoencoder are as follows:

- It avoids convergence problems as it is a noniterative training procedure, thus speeding up the training stage.
- It is hyperparameter-free, a highly desirable characteristic for large-scale learning environments, in which tuning a model can be a very time consuming task.
- It is highly scalable as the minimization problem to obtain the optimal weights of the second layer can be split into several subproblems (one for each output unit) that can be solved independently in a parallel manner.
- It is a privacy-preserving method and, therefore, adequate when data must be preserved between the different data providers sites.
- In comparison with the state of the art, it is a highly competitive solution to deal with very large data sets by reducing training from several hours to seconds while maintaining good accuracy.
- It is a lightweight model that could be also useful for a network of low capacity devices like those found in Internet of Things scenarios.

ACKNOWLEDGMENTS

This study has been supported by the grant *Machine Learning on the Edge—Ayudas Fundación BBVA a Equipos de Investigación Científica 2019*. It has also been possible thanks to the support received by the National Plan for Scientific and Technical Research and Innovation of the Spanish Government (Grant no. TIN2015-65069-C2-1-R and PID2019-109238GB-C2), and by the Xunta de Galicia (Grant no. ED431C 2018/34) with the European Union ERDF funds. CITIC, as a Research Center of the University System of Galicia, is funded by Consellería de Educación, Universidade e Formación Profesional of the Xunta de Galicia through the European Regional Development Fund (ERDF) and the Secretaría Xeral de Universidades (Ref. ED431G 2019/01).

ENDNOTES

[1] https://ec.europa.eu/info/law/law-topic/data-protection_en.

REFERENCES

1. Moya MR, Koch MW, Hostetler LD. One-class classifier networks for target recognition applications. In: *Proceedings of the World Congress on Neural Networks*; 1993:797-801.
2. Minter TC. Single-class classification. In: *Proceedings of the Symposium on Machine Processing of Remotely Sensed Data*; 1995:2A12-2A15.
3. Chandola V, Banerjeea A, Kumar V. Anomaly detection: a survey. *Acm Comput Surv*. 2009;41(3):1-72.
4. Bishop C. Novelty detection and neural network validation. *IEEE Proc Vision Image Signal Process*. 1994; 141:217-222.
5. Japkowicz N, Myers C, Gluck M. A novelty detection approach to classification. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*; 1995:518-523.
6. Schwenk H. The diabolo classifier. *Neural Comput*. 1998;10(8):2175-2200.
7. Peteiro-Barral D, Guijarro-Berdiñas B. A survey of methods for distributed machine learning. *Prog Artif Intell*. 2013;2:1-11.
8. Ohrimenko O, Schuster F, Fournet C, et al. Oblivious multi-party machine learning on trusted processors. In: *Proceedings of the 25th USENIX Security Symposium*; 2016:619-636.
9. Mazhelis O. One-class classifiers: a review and analysis of suitability in the context of mobile masquerader detection. *S Afr Comput J*. 2006;36:29-48.
10. Brew A, Grimaldi M, Cunningham P. An evaluation of one-class classification techniques for speaker verification. *Mach Learn*. 2007;27(4):295-307.
11. Bergamini C, Koerich AL, Sabourin R. Combining different biometric traits with one-class classification. *Signal Process*. 2009;89(11):2117-2127.
12. Kennedy K, Namee BM, Delany SJ. Credit scoring: solving the low default portfolio problem using one-class classification. In: *Proceedings of the Irish Conference on Artificial Intelligence and Cognitive Science*; 2009:168-177.
13. Khan SS, Madden MG. A survey of recent trends in one class classification. In: *Artificial Intelligence and Cognitive Science (AICS'09)*; 2010:188-197.
14. Khan SS, Madden MG. One-class classification: taxonomy of study and review of techniques. *Knowl Eng Rev*. 2014;29(3):345-374.
15. Tax DMJ. *One-class Classification* [PhD thesis]. Delft University of Technology; 2001.
16. Tarassenko L, Hayton P, Brady M. Novelty detection for the identification of masses in mammograms. In: *Proceedings of the IEEE Conference on Artificial Neural Networks*; 1995:442-447.
17. Bishop C. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press; 1995.
18. Duda R, Hart P. *Pattern Classification and Scene Analysis*. New York, NY: John Wiley & Sons; 1973.
19. Parzen E. On estimation of a probability density function and mode. *Ann Math Stat*. 1962;33(3):1095-1076.
20. Tax DMJ, Duin RPW. Combining one-class classifiers. In: *Proceedings of the Second International Workshop on Multiple Classifier Systems*; 2001:299-308.
21. Breunig MM, Kriegel HP, Ng RT, Sander J. LOF: identifying density-based local outliers. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*; 2000:93-104.
22. Branch J, Szymanski B, Giannella C, Wolff R, Kargupta H. In-network outlier detection in wireless sensor networks. In: *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'06)*; 2006:51.
23. Bai M, Wang X, Xin J, Wang G. An efficient algorithm for distributed density-based outlier detection on big data. *Neurocomputing*. 2016;181:19-28.
24. Ypma A, Duin R. Support objects for domain approximation. In: *Proceedings of the International Conference on Artificial Neural Networks (ICANN'98)*; 1998:719-724.
25. Tax DMJ, Duin RPW. Support vector data description. *Mach Learn*. 2004;54:45-66.
26. Schölkopf B, Williamson RC, Smola AJ, Shawe-Taylor J, Platt JC. Support vector method for novelty detection. In: *Advances in Neural Information Processing Systems 12 (NIPS)*; 2000:582-588.
27. Mygdalis V, Iosifidis A, Tefas A, Pitas I. Graph embedded one-class classifiers for media data classification. *Pattern Recognit*. 2016;60(C):585-595.

28. Sohrab F, Raitoharju J, Gabbouj M, Iosifidis A. Subspace support vector data description. In: *Proceedings of the 24th International Conference on Pattern Recognition (ICPR)*; 2018:722-727.
29. Ruff L, Vandermeulen R, Görnitz N, et al. Deep one-class classification. In: *Proceedings of the 35th International Conference on Machine Learning*; 2018:4393-4402.
30. Dolia AN, Bie TD, Harris CJ, Shawe-Taylor J, Titterington D. The minimum volume covering ellipsoid estimation in kernel-defined feature spaces. *Mach Learn*. 2006;4212:630-637.
31. Martínez-Rego D, Castillo E, Fontenla-Romero O, Alonso-Betanzos A. A minimum volume covering approach with a set of ellipsoids. *IEEE Trans Pattern Anal Mach Intell*. 2013;35(12):2997-3009.
32. Casale P, Pujol O, Radeva P. Approximate convex hulls family for one-class classification. In: *Proceedings of the 10th International Workshop on Multiple Classifier Systems*; 2011:106-115.
33. Casale P, Pierluigi P, Radeva P. Approximate polytope ensemble for one-class classification. *Pattern Recognit*. 2014;47(2):854-864.
34. Désir C, Bernard S, Petitjean C, Heutte L. A new random forest method for one-class classification. In: 7626 of *Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR 2012). Lecture Notes in Computer Science*; 2012:282-290.
35. Castillo E, Peteiro-Barral D, Berdiñas BG, Fontenla-Romero O. Distributed one-class support vector machine. *Int J Neural Syst*. 2015;25(7):1550029.
36. Carpenter G, Grossberg S, Rosen D. ART 2-A: an adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks*. 1991;4(4):493-504.
37. Kohonen T. *Self-organizing maps*. Berlin, Heidelberg: Springer-Verlag; 1995.
38. Baldi P, Hornik K. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*. 1989;2(1):53-58.
39. Hertz J, Krogh A, Palmer R. *Introduction to the theory of neural computation*. Boca Raton, FL: Addison Wesley Publishing Company; 1991.
40. Candès EJ, Li X, Ma Y, Wright J. Robust principal component analysis? *J ACM*. 2011;58(3).
41. Xiong L, Chen X, Schneider J. Direct robust matrix factorization for anomaly detection. In: *Proceedings of the IEEE 11th International Conference on Data Mining (ICDM'11)*; 2011:844-853.
42. O'Reilly C, Gluhak A, Imran MA. Distributed anomaly detection using minimum volume elliptical principal component analysis. *IEEE Trans Knowl Data Eng*. 2016;28:2320-2333.
43. Kramer MA. Autoassociative neural networks. *Comput Chem Eng*. 1992;16(4):313-328.
44. Thompson BB, Marks RJ, Choi JJ, El-Sharkawi MA, Huang MY, Bunje C. Implicit learning in autoencoder novelty assessment. In: *Procedings of the International Joint Conference on Neural Networks—volume 3 (IJCNN'02)*; 2002:2878-2883.
45. Sanz J, Perera R, Huerta C. Fault diagnosis of rotating machinery based on auto-associative neural networks and wavelet transforms. *J. Sound Vibration*. 2007;302(4-5):981-999.
46. Miranda V, Castro ARG, Lima S. Diagnosing faults in power transformers with autoassociative neural networks and mean shift. *IEEE Trans Power Del*. 2012;27(3):1350-1357.
47. Shahin Shamsabadi A, Haddadi H, Cavallaro A. Distributed one-class learning. In: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*; 2018:4123-4127.
48. Battiti R. First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*. 1992;4(2):141-166.
49. Sontag E, Sussmann HJ. Backpropagation can give rise to spurious local minima even for networks without hidden layers. *Complex Syst*. 1989;3:91-106.
50. Budinich M, Milotti E. Geometrical interpretation of the back-propagation algorithm for the perceptron. *Phys A*. 1992;185:369-377.
51. Fontenla-Romero O, Guijarro-Berdiñas B, Pérez-Sánchez B, Alonso-Betanzos A. A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recognit*. 2010;43(5):1984-1992.
52. Fontenla-Romero O, Pérez-Sánchez B, Guijarro-Berdiñas B. LANN-SVD: a non-iterative SVD-based learning algorithm for one-layer neural networks. *IEEE Trans Neural Netw Learn Syst*. 2018;29(8):3900-3905.
53. Fontenla-Romero O, Guijarro-Berdiñas B, Pérez-Sánchez B, Martínez-Rego D, Rego-Fernández D. A fast learning algorithm for high dimensional problems: an application to microarrays. In: *Proceedings of the European Symposium on the Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*; 2016:283-288.

54. Japkowicz N, Hanson SJ, Gluck MA. Nonlinear autoassociation is not equivalent to PCA. *Neural Comput*. 2000;12(3):531-545.
55. Sakurada M, Yairi T. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In: *Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA'14)*; 2014:4-11.
56. Hwang B, Cho S. Characteristics of auto-associative MLP as a novelty detector. In: *Proceedings of the International Joint Conference on Neural Networks—volume 5 (IJCNN'99)*; 1999:3086-3091.
57. Eckart C, Young G. The approximation of one matrix by another of lower rank. *Psychometrika*. 1936;1(3): 211-218.
58. Fowler ML, Chen M, Johnson JA, Zhou Z. Data compression using SVD and Fisher information for radar emitter location. *Signal Process*. 2010;90(7):2190-2202.
59. Deng WY, Bai Z, Huang GB, Zheng QH. A fast SVD-hidden-nodes based extreme learning machine for large-scale data analytics. *Neural Networks*. 2016;77:14-28.
60. Iwen MA, Ong BW. A distributed and incremental SVD algorithm for agglomerative data analysis on large networks. *SIMAX*. 2016;37:1699-1718.
61. Bishop CM. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. New York, NY: Springer-Verlag; 2006.
62. Shokri R, Shmatikov V. Privacy-preserving deep learning. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Association for Computing Machinery; 2015:1310-1321.
63. Hitaj B, Ateniese G, Perez-Cruz F. Deep models under the gan: information leakage from collaborative deep learning. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery; 2017:603-618.
64. Zhao Q, Zhao C, Cui S, Jing S, Chen Z. PrivateDL: Privacy-preserving collaborative deep learning against leakage from gradient sharing. *Int J Intelligent Syst*. 2020;35(8):1262-1279.
65. Fredrikson M, Jha S, Ristenpart T. Model inversion attacks that exploit confidence information and basic countermeasures. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. Association for Computing Machinery; 2015:1322-1333.
66. Dua D, Graff C. UCI machine learning repository. Irvine, CA: University of California, School of Information and Computer Science, 2019. http://archive.ics.uci.edu/ml
67. Knowledge Pit. Silver Bullet Solutions. Hearthstone dataset; 2017. https://knowledgepit.fedcsis.org/contest/view.php?id=120
68. Breunig M, Kriegel H, Ng R, Sander J. Lof: Identifying density-based local outliers. *SIGMOD Rec*. 2000;29:93-104.
69. Lee G, Scott CD. The one class support vector machine solution path. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'07)*; 2007:521-524.
70. Schwenk H. The diavolo classifier. *Neural Comput*. 1998;10(8):2175-2200.
71. Eiras-Franco C, Martínez-Rego D, Guijarro-Berdiñas B, Alonso-Betanzos A, Bahamonde A. Large scale anomaly detection in mixed numerical and categorical input spaces. *Inf Sci*. 2019;487:115-127.

How to cite this article: Fontenla-Romero O, Pérez-Sánchez B, Guijarro-Berdiñas B. DSVD-autoencoder: A scalable distributed privacy-preserving method for one-class classification. *Int J Intell Syst*. 2021;36:177-199. https://doi.org/10.1002/int.22296

APPENDIX: A HYPERPARAMETERS USED FOR TRAINING

This appendix contains the values of the best hyperparameters found for each method and data set in the experiments carried out in this study (Tables A1 and A2). For the AUTO-NN and DSVD-Auto, in all cases, the transfer function employed for the first layer was the logarithmic function.

TABLE A1 Values of the hyperparameters of employed methods for small data sets

Data set	LOF hamming		AUTO-NN			DSVD-AUTO			APE			
	K	P	#Hiddens	f_2	Percentile	#Hiddens	f_2	Percentile	α	λ	#Dimension	#Projections
Arrhythmia	10*	0.01*	23	Linear	95	12	Linear	99	0.20	0.89	2	100
Breast	7	0.20	1	Log-sigmoid	95	1	Log-sigmoid	95	0.20	0.25	2	100
German Credit	10*	0.01*	2	Linear	99	7	Linear	99	0.00	0.30	2	100
Car	10	0.20	4	Linear	75	5	Log-sigmoid	75	0.00	0.98	2	100
Abalone	10*	0.01*	1	Linear	99	1	Log-sigmoid	99	0.00	0.34	2	100

Abbreviations: APE, approximate polytope ensemble; AUTO, autoencoder; DSVD, distributed singular value decomposition; LOF, local outlier factor.

*The values were obtained from Reference [71].

TABLE A2 Values of the hyperparameters of DSVD-autoencoder and AUTO-NN employed for largest data sets

	AUTO-NN and DSVD-AUTO		
Data set	**#Hiddens**	f_2	**Percentile**
MiniBooNE	2	Linear	99
Covertype	1	Linear	99
Susy	10	Linear	99
Hearthstone	2	Log-sigmoid	99
Higgs	13	Linear	99

Abbreviations: AUTO, autoencoder; DSVD, distributed singular value decomposition.