Online learning for anomaly detection via subdivisible convex hulls

David Novoa-Paradela
CITIC, University of A Coruña
A Coruña, Spain
david.novoa@udc.es

Oscar Fontenla-Romero
CITIC, University of A Coruña
A Coruña, Spain
oscar.fontenla@udc.es

Bertha Guijarro-Berdiñas
CITIC, University of A Coruña
A Coruña, Spain
berta.guijarro@udc.es

***Abstract*—Due to the frequent use of anomaly detection systems in monitoring and the lack of methods capable of learning in real time, this research presents a new method that provides such online adaptability. The method developed is called OSHULL (Online and Subdivisible Distributed Scaled Convex Hull) and bases its operation on the properties of scaled convex hulls. It begins building a convex hull, using a minimum set of data, that is adapted and subdivided along time to accurately fit the boundary of the normal class data. The method has been evaluated and compared to several main algorithms of the field using some real and artificial data sets. As a consequence, an algorithm has been obtained with online learning ability and easily configurable, all without diminishing its effectiveness in relation to other batch state-of-the-art methods. Finally, its execution can be carried out in a distributed and parallel way, which is an interesting advantage in the treatment of big data sets.**

***Index Terms*—anomaly detection, convex hull, data streaming**

I. Introduction

Anomaly detection is a subarea of machine learning that deals with the development of methods for discriminating among what is considered normal and anomalous data. This makes the detection of anomalies, a priori, a problem of classification into two unique classes. However, since anomalies usually occur sporadically, normal data is the one that prevails in these problems, therefore requiring specific machine learning models which training phase is generally carried out using mostly even normal data. Their objective is to model the normal class boundaries as accurately as possible, so that the classification of new data can be carried out by checking whether they belong to the normal class or not. Because of this, anomaly detection is also known as One-Class classification.

This type of problems are quite frequent in real-world scenarios, such as predictive maintenance. Also, very often, anomaly detection systems are part of monitoring systems, for example, in the supervision of the operation of a car engine. In these problems the ability to learn in real time can be essential as there may not be a sufficient amount of data at the beginning of the learning process but over time. There are many use cases (medical, IT security, etc.) where this situation happens and the detection methods must be able to start making decisions as soon as possible with very little initial knowledge and adapt this knowledge as new data are available. This is called online learning. Despite its interest, there are a low number of such methods for anomaly detection. This paper presents the adaptation of an anomaly detection method [1] [2] based on convex hulls and random projections. The contribution is to allow the convex hulls to be dynamically changed in an online learning scenario and to represent non-convex regions as a union of several convex hulls. The algorithm will adapt the limits of the normal class every time it processes new data, without having to store all the data or retrain from scratch.

This work has been supported by Spanish Government's Secretaría de Estado de Investigación (Grant TIN2015-65069-C2-1-R), Xunta de Galicia (Grants ED431C 2018/34 and ED431G/01) and EU FEDER funds.

II. Related work

This section summarizes the main types of anomaly detection techniques available in the state-of-the-art.

A. Batch-learning anomaly detection methods

Regarding the methods that do not learn in real time [3] [4] we can distinguish:

- Probabilistic methods: their main objective is the estimation of the density function that generates the data. They assume that the training data is generated by a probability distribution function D, which can be estimated using those same data. This estimated distribution $\hat{D}$ usually represents a normality model so that normal data will be given in regions with high probabilities and anomalies in regions with lower probabilities. To model these density functions, parametric (based on Gaussian or regression models [5]) and non-parametric (based on histograms or kernels [6]) techniques are used.
- Distance-based methods: these methods are based on distance metrics to calculate the similarity between two data. They assume that normal data is given in the form of dense neighborhoods, while anomalies are far from such neighborhoods. The two most important methods are nearest neighbor [7] and clustering [8].
- Reconstruction-based methods: these methods can autonomously model the training data set and predict new data by calculating their reconstruction error defined as the distance between the value provided by the model and the target value, thus serving as a score. They can be classified into two groups, those based on neural networks [9] and those based on subspaces.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

- Domain-based methods: they generate a boundary based on the structure of training data that separates classes in a space. These classifiers become insensitive to the size and density of each class as the classification of new data is determined solely by its location with respect to the boundary. To carry out the separation, a hyperplane (or other parametric structure) is approached that separates the points of one class from the other in an optimal way, although it is usual to work with problems of more than two classes combining several hyperplanes [10].
- Methods based on information theory: They assume that anomalous data significantly alter the information contained in the normal set and, therefore, they process the content of data sets through measures such as entropy or relative entropy [11]. Metrics are calculated using the complete data set with and without including the point to classify. The results are compared with those produced using only the normal set so that, if there are large differences, we can assume that we are facing an anomaly.

B. Online-learning anomaly detection methods

There is no clear taxonomy for online anomaly detection algorithms, but a large part of them are adaptations of classic methods to add this ability. Some of these techniques are:

- osPCA [12] is a PCA based version that do not require to store the entire data matrix or covariance. By oversampling the training data and extracting the principal direction, osPCA allows to determine anomalies according to the variation of the resulting dominant eigenvector.
- KRLS [13] combines the principles of kernel machines and the popular Recursive Least Squares (RLS) algorithm to provide an efficient and non-parametric approach for performing online anomaly detection.
- ORUNADA [14] is an online version of UNADA [15] which applies incremental subspace clustering via a discrete time-sliding. This window updates in a near continuous way the feature space. Then, the partition of each subspace is upgraded using an incremental grid clustering algorithm that divides the space into units and place points into them, allowing also good scalability.

III. Background

This section summarizes the main ideas of the two methods taken as the basis for the development of OSHULL: the APE algorithm developed by Casale *et al*. [1], and its distributed version carried out by Diego Fernández *et al.* [2]. Both techniques base their operation in the use of convex hulls over random projections of the data in 2-dimensional spaces.

A convex hull (CH) is the smallest convex polyhedron that contains a set of data points. Figure 1 shows an example of a CH wrapping a two dimensional data set, in this case the CH is a polygon. The usefulness of this structure in anomaly detection problems is to calculate the polyhedron that encompasses the set of normal data, so that classifying a new data consists on checking whether it is located within the convex hull (normal data) or outside it (anomaly).

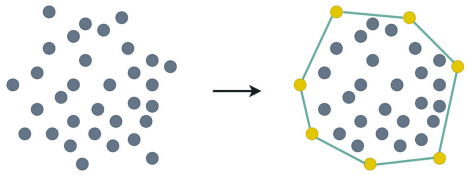

Fig. 1. Formation of a convex hull from a cloud of points.

Calculating the CH in high-dimensional spaces is, computationally, a very expensive task. Due to this, several anomaly detection methods choose to project the data on 2-dimensional spaces in which the calculation of CHs is simple [1]. This random projection technique is based on the idea that high-dimensional data spaces can be projected into a lower dimensional space without significantly losing the data structure [16] if multiple projections are used. The possibility to reduce the dimensionality of the problem without a high computational effort while preserving the data structure, allows to create very simple and powerful learning techniques. Building a CH in 2-dimensional spaces and check if a point falls inside are common tasks in geometric computing for which there are very efficient solutions [17]. Thus, the learning and classification phases consist on the following:

1) Learning phase: Given training data set (normal data) of dimension d, a number τ of random projections of the data onto a 2-dimensional subspace are made. First, τ random matrices are generated. Second, the training set is projected into the space generated by each of these projection matrices. Finally, the vertices of the CH are calculated in each projection, this being the aim of training. These vertices are projections of the original data, therefore, the algorithms only need to store the vertices that form the CHs and the projection matrices, discarding the rest of the training data.
2) Classification phase: To predict the class of a new data point, it is first projected using the τ projections generated during the training phase. For each projection, and given the set of vertices of the CH of that 2-D space, it is possible to check if the point is inside the corresponding polygon. A point will be classified as normal only if it is inside *all* CHs. In other case, the point will be considered anomalous. There are a lot of algorithms to perform this check in 2-D spaces [18]. A graphic example of this procedure is shown in Figure 2. The grey polyhedron represents the training data cloud in its original three dimensional space, the grey polygons represent the corresponding CHs calculated for each P-projection, and the orange dot a new datum to be classified. It will be classified as an anomaly since it falls out of CH in the the P_1 projection.

To avoid the effect of over-adjustment of the training data, reduced and expanded versions of the initial CH called Scaled Convex Hulls (SCH) are used [1]. The size of these scaled versions will be given by a λ parameter, and the new data

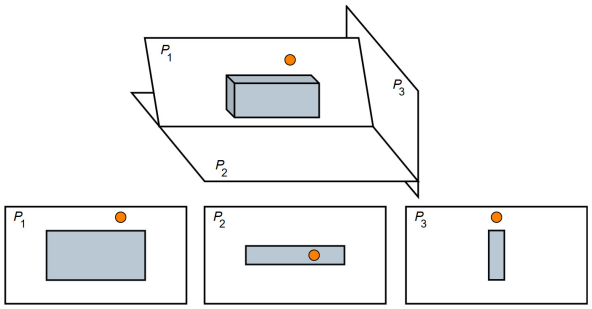

Fig. 2. Three projections of a three-dimensional data cloud and a new point to be classified.

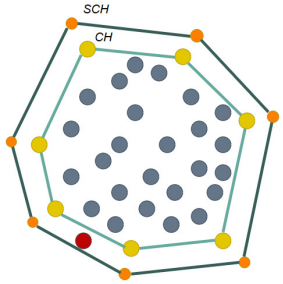

Fig. 3. Scaled convex hull (orange) obtained from an initial convex hull (yellow). The red dot will be correctly classified as normal data thanks to the use of the margin.

will be classified based on them. If $\lambda > 1$ the vertices will expanded and otherwise will be contracted.

In addition, D. Fernández et al. [2] proposed a distributed version of the APE algorithm of Casale et al. [1]. It allows data privacy preservation as the information that is exchanged among the distributed sites is not the data itself but their projections, and that projection process is not invertible. Our proposal is based on this version for being the most advanced.

IV. ONLINE AND SUBDIVISIBLE DISTRIBUTED SCALED CONVEX HULL

The main objective of the proposed method OSHULL is to carry out online one-class learning, so that starting from a minimum set of initial data points, it allows learning and adjusting the model with the arrival of new normal data. This section presents its theoretical foundations. The steps described below are applied independently to the convex hulls at each of the projections. Sections A and B correspond to the base algorithm developed by Casale et al. [1]. Our novel contribution can be found in sections C and later.

A. Calculation of the initial convex hulls

The first step of the algorithm is to carry out the projection of training data in 2-D spaces. For this aim, it is necessary to generate random matrices corresponding to the projections, considering that the more projections used, the more accurately the training set will be represented. Later on, a portion of the training set will be projected as determined by each of these projection matrices and the CHs will be calculated, one per projection. From this moment, the system is ready to readjust the CHs, as it will be seen in Section IV-C. Notice that only the projections of the data that correspond to CH vertices in each subspace need to be stored, and the rest of the projections as well as the original training data can be discarded.

B. Calculation of the scaled convex hulls

In some cases there may be normal data that is misclassified as false positives anomalies during training because they are located outside the limits of some of the convex hulls calculated during the initial formation phase. In order to avoid this behavior and to softly adjust the shape of the data, we have decided to use a $\lambda > 1$ expansion factor. This value allows to generate scaled convex hulls (SCH), whose purpose is to soften the classification of new data. In this work, the formula presented by Liu *et al.* [19] was used to calculate the SCH. Given the set of points $S \subseteq \mathbb{R}^d$, vertices of the CH are defined with respect to the center point $c = (1/|S|)\sum_i x_i, \forall x_i \in S$ and the expansion parameter $\lambda \in [0, +\infty)$ as:

$$v^{\lambda} : \{\lambda v + (1-\lambda)\, c \mid v \in CH\,(S)\} \tag{1}$$

The parameter λ specifies a constant contraction $(0 < \lambda < 1)$ or extension $(\lambda > 1)$ of the CH with respect to c. We will always use $\lambda > 1$ since we are not interested in reducing the CH. Figure 3 shows an example of a scaled convex hull.

C. Convex hull adjustment

To provide the method with the ability to learn in real time, the limits of CHs have to be readjusted as new data is available to the system. To do this, and at the same time guarantee some system stability, these limits will be expanded whenever a considerable number of data falls systematically and concentrated around the same area between the limits of the CH and the SCH. In this case, the behavior of the algorithm will be to extend the limits of the CH to cover, as far as possible, that area where normal data did not appear when building the initial CH. With this aim, we added a new stage to the training phase in which, after forming the initial convex hulls, they are further reshaped as explained below:

1) When the projection of new data point falls into the *margin* (space between the limits of the CH and the SCH), it is determined which of the vertices of the CH is the closest one according to the euclidean distance.
2) The position of this new datum in the projected space is stored in a list associated with this nearest vertex. Each vertex will have a list of data that has fallen close to it so that, when the length of this list exceeds an established threshold, the CH will expand by creating a new vertex.
3) The vertex expansion will consist in creating a new vertex calculated as the *centroid* of the data stored in the vertex's list that caused the expansion.
4) Simply adding the vertex to the hull can cause it to stop being convex. Instead, the CH is recalculated using as data the old vertices and the new one.

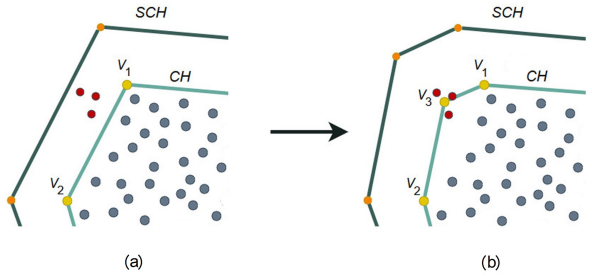

Fig. 4. Readjustment of a CH to cover new data falling consistently in its margin.

5) Once recalculated, if the new vertex becomes part of the convex hull, the SCH will be also accordingly modified using equation 1.

Figure 4 (a) contains an example of a CH that has received several data points in its margin, close to vertex V_1, while Figure 4 (b) represents the new CH after the expansion from V_1 and the generation of vertex V_3. It is important to remember that although in the figure the training data points are represented (blue dots) the algorithm only stores the vertices and their corresponding lists. Also, note that this readjustment process at no time involves projecting the available data and recalculating the CHs. On the contrary, this process will be carried out only once during the formation of the initial CHs since the projection matrices will always be the same and the modifications to a projection do not affect the others.

D. Region subdivision

Because representing data sets using a single CH produces poor results in data sets with non-convex geometric shapes, the OSHULL method implements an iterative process that subdivides CHs for a better fit to the shape of the data. In this way, starting from an initial convex hull, it can be recursively subdivided into as many convex hulls as necessary to properly approximate the shape of normal data. Therefore, the method may represent non-convex regions as the union of various convex regions. At each projection several convex hulls can coexist that will continue to be readjusted individually. Figure 5 shows an example of the subdivision of a CH to model more accurately the non-convex set of projected data. The following subsections describe the steps of this subdivision process.

1) Identification of non-convex areas, support points: The proposed methodology for subdividing a CH begins by establishing a metric that measures how well a convex hull represents the normal data. A convex hull is composed of vertices (data projections) that determine edges between them. When a CH envelops a non-convex data set, empty regions in which normal data does not exist are generated within the convex hull (see Figure 5). These regions are those that the subdivision process should try to remove to avoid classification failures (false negatives).

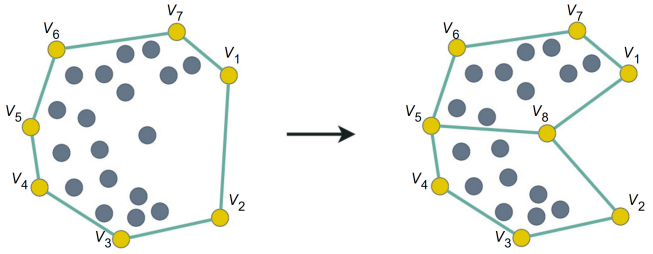

Fig. 5. Convex hull that must be subdivided to better capture the boundary of a data set with half moon morphology.

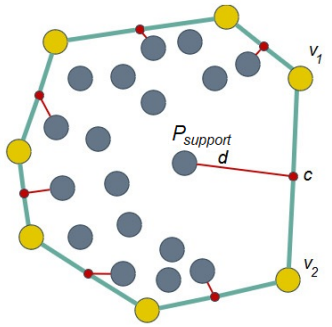

Fig. 6. Convex hull and distances to support points (red lines).

Since the edges of the CH closest to these empty regions do not have nearby data points, we propose to locate these regions using a *support point* associated to each edge. This support point is defined as the data point inside the convex hull closest, in the Euclidean sense, to the midpoint of that edge. In Figure 6, vertices V_1 and V_2 form an edge with a midpoint in c. Therefore, the support point of this edge will be the data inside the CH closer to c, in this case the point $P_{support}$. Calculating the distances to the support points of every edge (red lines in Figure 6), it can be seen how the edges that best represent normal data have closer support points than those that are poorly adjusted. In our example, the distance d of the edge connecting V_1 and V_2 is much longer than the others, which indicates that the area around c is a good place to start subdividing the CH. During online training, each time a datum falls inside a convex hull, the distance of the new datum to all edges is calculated. If, for a given edge, this distance is shorter than the distance to its current support point, this point will be substitute by the new datum. Notice that, a support point can be shared by some edges, which is a common situation at the beginning of training or after a subdivision.

2) Edge selection to divide the convex hull: This step must be able to decide when a distance from an edge to its support point is large enough to subdivide a CH. Each projection can generate convex hulls of different sizes, therefore, the minimum distance of an edge to its support point cannot be a constant chosen by the user. To automate its calculation, the algorithm uses the interquartile range (IQR) of the distances between all edges and their support points. This allows to automatically calculate and adapt the threshold that separates

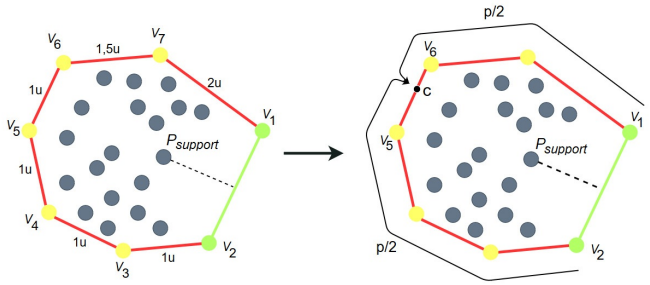

Fig. 7. The figure on the left shows a CH before being divided. The length of each edge is represented in units u, so that the perimeter p totals $7.5u$. The figure on the right shows the point c, located at a distance $p/2$ from V_1 and V_2. The vertex closest to c will be chosen as the pivot of the subdivision.

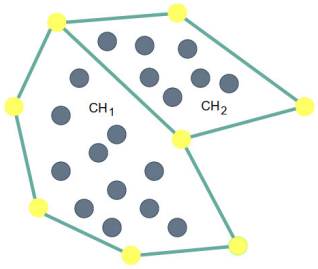

Fig. 8. Two convex hulls created from a subdivision.

the normal distances from the atypical and extremely atypical. The algorithm will rely on this extremely atypical threshold to decide when the distance of an edge to its support point is large enough to subdivide the CH based on that edge.

3) Determination of the forms of the new regions: Once the need of division is detected, our approach tries to reduce the number of convex decision regions by dividing the CH into two convex hulls of similar size (area). The idea of the procedure is to find a *pivot vertex* V_i of the CH that, together with the vertices of the selected edge and its support point $P_{support}$, will form the new edges that divide the CH into two convex hulls of similar areas. Figure 7 illustrates this process in which the (V_1, V_2) edge is selected for division. To find this *pivot vertex* V_i, we calculate the length p of the perimeter of the CH excluding the (V_1, V_2) edge (red edges in Figure 7) and the point c that is at a distance $p/2$ from the two vertices of the edge selected for division. Finally, the pivot vertex to make the subdivision will be V_6 as it is the closest to that midpoint c. Figure 8 shows the two resulting CHs after subdivision.

Although the use of the perimeter of the CH does not provide maximum precision, this operation is easy to calculate because the positions of the vertices are known at all times, and the results it produces are sufficient for the proper functioning of the algorithm. Other possible options, such as calculating the area in irregular polygons and for all possible combinations of vertices, imply operations computationally more expensive, not so adequate for a real time learning algorithm.

Finally, the process of subdivision of a convex hull for a given projection can be summarized as:

1) Calculate the extremely atypical threshold by using the distances between edges and support points for all convex hulls in that projection. Including all convex hulls in the calculation allows to relate them.
2) Locate the edge of the CH whose distance to its support point is higher than this threshold and it is the greatest for that projection.
3) Choose the pivot vertex dividing the CH into two regions of similar sizes from the vertices of the selected edge according to the perimeter.
4) Generate two new convex hulls using the support point, the pivot vertex, and the vertices of the selected edge.
5) The edges of these two new convex hulls should select their support points by reusing the CH support points available before the division.

E. Freezing process

In the same way that a CH must be subdivided if it has edges enclosing empty spaces with distances exceeding the atypical threshold, a CH with all its edges at low distances, and therefore well adjusted, must be maintained until the end of training and prevent it from being unnecessarily divided. This process is called *freezing*. A CH will freeze if all its edges are at normal distances and it has not been subdivided for a given number of iterations. The higher this number, the higher the opportunities for data to fall inside a CH and, therefore, to reduce the distance from the edges to their support points. A frozen CH cannot be subdivided again, but it will continue to be readjusted with data dropping in its margin.

The threshold used as the maximum distance to freeze a CH must be a value in the range of normal distances, so the method uses again the interquartile range. In this case, the third quartile is taken as the threshold. Thus, if the distance of all edges of a CH to their support points is less than $Q3$ and it has not been subdivided during a minimum number of iterations, the region will freeze.

F. Pruning process

After several subdivisions, some convex hulls can cover regions completely empty of normal data. Also, small convex hulls can be found that overlap the margins of other adjacent convex hulls. To get rid of them, a periodic pruning process is performed that eliminates those convex hulls that have not received data inside during a given time interval, as it is assumed that they do not represent normal data.

With this aim, when a CH is created, a variable is associated that counts the points that fall inside it. This counter will increase during training any time a data falls only and exclusively within the CH, i.e., if a data falls in two or more convex hulls simultaneously, none of them will increase its counter. With this behavior, if one CH is included in a greater CH, the small one eventually will disappear.

G. Pseudocode

Algorithm 1 contains the pseudocode for the OSHULL *training phase*. In line 8, information about a projection is

stored (CH vertices and its center). In line 9, the model is completed by storing the information of the new projection created (the projection matrix, the vertices of both the SCH and CH, and the CH center). In line 14, Algorithm 2 is called, which is responsible for carrying out the expansion, subdivision, freezing and pruning in the CHs of the t projection. In Algorithm 2, line 6 determines if the new data falls inside the CH, in the margin or outside it. In line 9, for each CH where the point has fallen into the margin, we check which vertex is the closest and add the point to its associated list. In line 10, we record the fall in the i CH. In line 13, if the new point has only fallen into a CH, and the vertex in question exceeds the minimum number of data nearby, the CH is expanded.

Algorithm 1 OSHULL training phase

Inputs: $S \in R^d$, training set to form the initial CH; $D \in R^d$, training set for adapting the CH boundaries; τ, number of projections; λ, scaling parameter; $minIt$, minimum number of iterations to subdivide and freeze; $minDNear$, number of data that must fall near a vertex to expand it;

Output: M, learnt model composed of τ projection matrices and their respective CH;

1: $M = \varnothing$
2: **for** $t = 1..\tau$ **do** ▷ Generate projections and initial CHs
3: $P_t \sim N(0,1)$ ▷ Random projection matrix $[2 \times d]$
4: $\bar{S}_t : \{P_t x \,|\, x \in S\}$ ▷ Project the data
5: $\{v\}_t = CH(\bar{S}_t)$ ▷ Vertices of the CH
6: $\bar{c} = getCenter(\bar{S}_t)$ ▷ Center of the CH
7: $\{v\}_t^{\lambda} : \{\lambda v_i + (1-\lambda)\bar{c} \,|\, v_i \in \{v\}_t\}$ ▷ Calculate SCH
8: $H_t = storeCH(\{v\}_t, \bar{c})$ ▷ Store CH info
9: $M = M \cup \left(P_t, \{v\}_t^{\lambda}, H_t\right)$
10: **end for**
11: **for** $i = 1..len(D)$ **do** ▷ Readjust the model with D
12: **for** $t = 1..\tau$ **do** ▷ For each projection
13: $\bar{x}_i : \{P_t x_i \,|\, x_i \in D\}$ ▷ Project the new data point
14: $M = readjustCH(M, t, \bar{x}_i, minIt, minDNear)$
15: **end for**
16: **end for**

Algorithm 3 contains the *classification phase* that is used after training. In line 10, if the new point is out of all CHs in one projection, it will be classified as an anomaly and is not necessary to continue checking the remaining projections.

Finally, a distributed version of the algorithm is easily done by distributing the operations corresponding to each of the generated 2-D spaces (i.e., projections) through a network of independent computational nodes. Each node is assigned a projection or a group of projections on which to carry out the learning phase (Algorithms 1 and 2). After that, a new data will be classified taking into account the aggregated classifications over the different nodes (Algorithm 3).

V. RESULTS

In this section, several experiments are presented to show the behaviour of the proposed method. To have a reference

Algorithm 2 Procedure used during training to readjust the convex hulls of a projection: expansion, subdivision, freezing and pruning.

Input: M, learnt model; t, projection to be deal; $\bar{x}$, new projected point in the subspace t; $minIt$, minimum number of iterations to subdivide and freeze; $minDNear$ number of data that must fall near a vertex to expand it;

Output: M, updated learnt model;

1: **procedure** READJUSTCH
2: $CHlist : \{CH | CH \in M_t\}$ ▷ List of CHs from M in t
3: $InfoList : \{H \,|\, H \in M_t\}$ ▷ Info about each CH in t
4: $cExpList = []$ ▷ Candidates Expand List
5: **for** $i = 1..len(CHlist)$ **do** ▷ For each t-convex hull
6: $Pos = determineFallPlace(\bar{x}_t, CHlist_i)$
7: **if** $Pos = INSIDE$ **then** $updateSuppPts(InfoList_i)$
8: **else if** $Pos = MARGIN$ **then**
9: $InfoList_i = updateNearPts(InfoList_i)$
10: $cExpList = recordCH(cExpList, i)$
11: **end if**
12: **end for**
13: **if** $((len(cExpList) == 1)$ and $(InfoList_i[numberDataNear] \geq minDNear))$ **then**
14: $i = cExpList[0]$ ▷ The index of the CH is i
15: $CHlist_i = expandConvex(CHlist_i, InfoList_i)$
16: **end if**
17: $\mu = calculateIQR(CHlist, InfoList)$ ▷ Calc. IQR
18: $CHlist = freeze(CHlist, InfoList, minIt, \mu)$
19: $CHlist = subdivide(CHlist, InfoList, minIt, \mu)$
20: $CHlist = delete(CHlist, InfoList, minIt, \mu)$
21: $M = updateModel(M, CHlist, InfoList)$
22: **end procedure**

Algorithm 3 OSHULL classification phase

Input: $x \in R^d$ datum to be classified; M, learnt model;
Output: $Result \in \{NORMAL, ANOMALY\}$

1: **for** $t = 1..\tau$ **do** ▷ For each projection
2: $CHlist : \{CH \,|\, CH \in M_t\}$ ▷ CHs in t projection
3: $\bar{x}_t = P_t x$ ▷ Project the new point
4: $Result = ANOMALY$
5: **for** $i = 1..len(CHlist)$ **do** ▷ For each convex hull
6: **if** $\bar{x}_t \in CHlist_i$ **then**
7: $Result = NORMAL$
8: **end if**
9: **end for**
10: **if** $Result == ANOMALY$ **then**
11: $Break$
12: **end if**
13: **end for**

point, we have also included in this experimental study several of the well-known machine learning algorithms for anomaly detection: One-Class Support Vector Machine (O-SVM) [20], Robust Covariance (RC) [21], Isolation Forest (IF) [22] and Local Outlier Factor (LOF) [23]. Remember that none of these classical algorithms work online. In addition, we decided to include in the comparative analysis a previous version of our algorithm (O-DSCH) that do not have the ability to subdivide the convex hull in order to check the improvement obtained with this new characteristic.

To evaluate the algorithms, several data sets were employed of two types: five artificial and three real ones. The artificially generated data sets allow us to evaluate the ability of the method to adapt to certain geometric shapes, such as half-moons or quasi-spheres separated in space. All are three-dimensional. In addition, real data sets have been used to evaluate the algorithm against higher dimensional data sets, corresponding to detection of microcalcifications in mammograms [1] (6 features), detection of fraud in credit cards [24] (30 features) and detection of failures in data storage systems [25] (19 features). In the case of real data sets, to assess the performance of each algorithm, data were partitioned into training and test sets using a 5-fold cross validation. Instead, for the artificial data sets, we created five versions of each one, divided into train and test sets, preserving the shape but varying aspects such as the size and distribution of data in space. In both cases, the partition into train and test sets was carried out using only data from normal class. Subsequently, to each test set, 1% of anomaly data points were added to check the performance using data of both types. The final results for every data set were obtained as the average of the five folds. In all cases the input variables have been previously normalized. The value of the lambda parameter has been estimated for each test using a grid search. To estimate the performance of the methods during the experiments, we used standard metrics for classification problems. Considering the anomalous class as the positive one these statistical metrics are based on the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) as follows:

- Recall: measures the probability of classifying an anomalous data as such. It is defined as:

$$Recall = TP/(TP + FN)$$

- Specificity: measures the probability of classifying a normal data as such. It is defined as:

$$Specificity = TN/(TN + FP)$$

- Accuracy: measure the probability of classifying a data correctly, whether it is positive or negative. It is defined as:

$$Accuracy = (TP + TN)/(VP + TN + FP + FN)$$

- Similarity: relates accuracy (A) and recall (R), allowing the results to be compared more directly. It is defined as:

$$Similarity = 1 - \left(\sqrt{(1-A)^2 + (1-R)^2}/\sqrt{2}\right)$$

TABLE I
AVERAGE TEST RESULTS (%) ± STANDARD DEVIATION FOR THE 5 ARTIFICIAL DATA SETS.

Method	Recall	Specificity	Accuracy	Similarity
OSHULL	**95.4± 1.3**	94.4± 1.9	94.4± 1.7	94.7± 1.5
O-DSCH	73.6± 7.1	98.8± 0.9	98.5± 0.9	81.3± 4.0
O-SVM	94.7± 1.5	97.4± 1.3	97.4± 1.1	**95.7± 1.3**
RC	93.5± 1.8	**99.5± 0.4**	**99.5± 0.4**	95.3± 1.4
IF	87.1± 2.6	98.2± 0.9	98.1± 1.4	90.7±2.0
LOF	93.8± 1.7	99.5± 0.5	99.4± 0.3	95.6± 1.2

TABLE II
AVERAGE TEST RESULTS (%) ± STANDARD DEVIATION FOR THE MICROCALCIFICATION DATA SET.

Method	Recall	Specificity	Accuracy	Similarity
OSHULL	70.8± 4.2	78.4± 2.3	77.9± 3.3	74.1± 3.2
O-DSCH	21.9± 24.6	**99.8± 0.2**	**94.3± 2.0**	44.6± 12.3
O-SVM	60.2± 8.4	90.5± 2.1	88.4± 2.7	70.7± 3.5
RC	72.7 ± 4.0	89.8± 2.5	88.6± 1.9	79.1± 2.9
IF	87.3± 2.8	75.7± 3.2	76.5± 2.3	**81.1± 2.4**
LOF	**88.5± 1.9**	74.7± 3.2	75.7± 2.6	80.9 ± 3.2

In the case of the OSHULL algorithm, to assess its behavior in an online learning environment, only half of the training data was used to create the initial convex hulls and the second half was used to adapt them in real time, sample to sample. However, OSHULL has shown similar results with partitions ranging between 30-50% for this first phase, and 70-50% for the second. The balance lies in having a representative initial data set, while having a sufficiently large amount of data to carry out the adjustment.

Table I summarize the average test results obtained using the artificial data sets. To model these sets, OSHULL used 50 projections. Tables II, III and IV show the results obtained on the real data sets using for the OSHULL, respectively, 125, 200 and 50 projections. Best results are boldfaced. Finally, as a summary, Table V contains the average test similarity obtained by the algorithms for the eight data sets. In addition,

TABLE III
AVERAGE TEST RESULTS (%) ± STANDARD DEVIATION FOR THE CREDIT CARD FRAUD DATA SET.

Method	Recall	Specificity	Accuracy	Similarity
OSHULL	80.4± 4.4	96.2± 1.4	95.9± 2.7	85.8± 1.9
O-DSCH	32.4± 17.0	**99.9± 0.3**	**98.8± 1.5**	52.2± 11.4
O-SVM	87.4± 2.1	95.9± 1.3	95.7± 1.8	**90.5 ± 2.8**
RC	89.9± 1.5	89.9± 1.0	89.9± 1.1	89.9± 2.3
IF	**91.3± 2.7**	84.2±2.6	84.4± 3.1	87.3± 3.9
LOF	89.6± 1.9	89.0± 2.2	89.1± 3.0	89.3± 3.7

TABLE IV
AVERAGE TEST RESULTS (%) ± STANDARD DEVIATION FOR THE FAILURES IN DATA STORAGE SYSTEMS DATA SET.

Method	Recall	Specificity	Accuracy	Similarity
OSHULL	84.1± 2.3	**82.4± 1.1**	82.4± 1.2	83.2± 1.7
O-DSCH	58.1± 3.0	99.4± 0.4	**98.9± 1.1**	70.3± 2.1
O-SVM	**93.1± 1.5**	79.6± 2.5	79.7± 2.1	84.8 ± 1.7
RC	71.9± 1.7	67.7± 1.3	67.7± 1.6	69.7± 1.6
IF	62.1± 1.4	56.5± 1.9	56.6± 2.5	59.2± 2.0
LOF	91.3± 1.3	80.4± 1.0	80.5± 1.2	**84.9± 1.4**

TABLE V
AVERAGE SIMILARITY (%) ± STANDARD DEVIATIONS AND AVERAGE RANKING POSITION IN THE DIFFERENT TESTS.

	Avg. similarity	Avg. ranking position
LOF	**91.6± 5.9**	**2.4**
O-SVM	90.6± 6.6	2.6
RC	89.4± 8.8	2.6
OSHULL	89.3± 7.8	3.6
IF	85.1± 6.3	4.1
O-DSCH	71.7± 19.1	5.6

a ranking is shown as the average position occupied by the algorithms considering all test results.

We can conclude that, thanks to the capacity of subdivision of the convex hulls, the performance of the base algorithm has been improved, as reflected in the average similarity, because it is possible to more accurately represent the boundaries of non-convex regions of data. In addition, compared to the classical non online algorithms, although OSHULL occupies the fourth position, its average similarity is only about 2.3 points from the best. Therefore, we can state that we have been able to obtain an algorithm with the ability to learn in real time without implying an important decrease in its performance in the detection of anomalies.

VI. CONCLUSION

The OSHULL algorithm fulfills the objectives established in order to obtain an anomaly detection method with on-line learning capability without losing significant performance compared to several classic non online anomaly detection methods. Due to its subdivision capacity, it is positioned as an alternative for non-convex problems. OSHULL is easily configurable through its parameters and it is not necessary to know a priori the percentage of anomalous data in the data set. It models are relatively light, it is not necessary to store all the data used for the creation or adaptation of the convex hulls. The model only needs the vertices of each convex hull to decide if a new one is within the limits of the normal class. Although the computational cost has been increased with respect to the base algorithm, this can be compensated by running OSHULL in a distributed and parallel way.

REFERENCES

[1] P. Casale, O. Pujol, and P. Radeva, "Approximate polytope ensemble for one-class classification," *Pattern Recognit.*, vol. 47, no. 2, pp. 854 – 864, 2014.

[2] D. Fernández-Francos, O. Fontenla-Romero, and A. Alonso-Betanzos, "One-class convex hull-based algorithm for classification in distributed environments," *IEEE Trans. Syst. Man Cybern.: Systems*, pp. 1–11, 2018.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[4] S. S. Khan and M. G. Madden, "One-class classification: Taxonomy of study and review of techniques," *CoRR*, vol. abs/1312.0049, 2013. [Online]. Available: http://arxiv.org/abs/1312.0049

[5] G. Thatte, U. Mitra, and J. Heidemann, "Parametric methods for anomaly detection in aggregate traffic (extended version)," *IEEE/ACM Trans. Netw.*, vol. 19, 2011.

[6] Dit-Yan Yeung and C. Chow, "Parzen-window network intrusion detectors," in *Object recognition supported by user interaction for service robots*, vol. 4, 2002, pp. 385–388 vol.4.

[7] P. Soucy and G. W. Mineau, "A simple KNN algorithm for text categorization," in *IEEE Int. Conf. on Data Mining*, 2001, pp. 647–648.

[8] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, "DBSCAN: Past, present and future," in *5th IEEE Int. Conf. on the Applications of Digital Information and Web Technologies*, 2014, pp. 232–238.

[9] G. Williams, R. Baxter, Hongxing He, S. Hawkins, and Lifang Gu, "A comparative study of rnn for outlier detection in data mining," in *IEEE Int. Conf. on Data Mining*, 2002, pp. 709–712.

[10] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. New York, NY, USA: Cambridge University Press, 2000.

[11] E. Keogh, J. Lin, and A. Fu, "Hot sax: efficiently finding the most unusual time series subsequence," in *5th IEEE Int. Conf. on Data Mining (ICDM'05)*, 2005, pp. 1–8.

[12] Y. Lee, Y. Yeh, and Y. F. Wang, "Anomaly detection via online oversampling principal component analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1460–1470, 2013.

[13] T. Ahmed, M. Coates, and A. Lakhina, "Multivariate online anomaly detection using kernel recursive least squares," in *26th IEEE Int. Conf. on Computer Communications*, 2007, pp. 625–633.

[14] J. Dromard, G. Roudière, and P. Owezarski, "Online and scalable unsupervised network anomaly detection method," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 34–47, 2017.

[15] P. Casas, J. Mazel, and P. Owezarski, "UNADA: Unsupervised network anomaly detection using sub-space outliers ranking," in *NETWORKING 2011*. Springer-Verlag, 2011, pp. 40–51.

[16] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporary mathematics*, vol. 26, no. 189-206, pp. 1 – 1, 1984.

[17] C. B. Barber, D. P. Dobkin, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996.

[18] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Berlin, Heidelberg: Springer-Verlag, 1985.

[19] Z. Liu, J. G. Liu, C. Pan, and G. Wang, "A novel geometric approach to binary classification based on scaled convex hulls," *IEEE Transactions on Neural Networks*, vol. 20, no. 7, pp. 1215–1220, 2009.

[20] G. Ratsch, S. Mika, B. Scholkopf, and K. Muller, "Constructing boosting algorithms from SVMs: an application to one-class classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 9, pp. 1184–1199, 2002.

[21] D. Cournapeau. (2019) Robust covariance api web. https://scikit-learn.org/stable/index.html, Revised at 20/07/2019.

[22] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[23] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *ACM Sigmoid International Conference on management of data*. ACM, 2000, pp. 93–104.

[24] K. Mezhoud. (2019) Credit card fraud detection. https://www.kaggle.com/dileep070/anomaly-detection/, Revised at 11/08/2019.

[25] C. Commons. (2019) High storage system data for energy optimization. https://www.kaggle.com/inIT-OWL/high-storage-system-data-for-energy-optimization, Revised at 11/11/2019.