Fast Anomaly Detection with Locality-Sensitive Hashing and Hyperparameter Auto Tuning

Jorge Meira, Carlos Eiras-Franco, Verónica Bolón-Canedo, Goreti Marreiros, Amparo Alonso-Betanzos

***Abstract*—This paper presents LSHAD, an anomaly detection (AD) method based on Locality Sensitive Hashing (LSH) which is able to deal with large-scale problems. The resulting algorithm is highly parallelizable and its implementation in Apache Spark further increases its ability to handle very large datasets. Moreover, the algorithm incorporates an automatic hyperparameter tuning mechanism that relieves users from costly tuning steps. Both the hyperparameter automation and its distributed properties make our method stand out from the rest, as these features are scarce in anomaly detection techniques. We report experiments comparing LSHAD with several state-of-the-art alternatives across a variety of datasets. The results show that our method obtains state-of-the-art performances in detecting anomalies, while being able to handle much larger datasets than its competitors. In addition, the experiments carried out evaluating the trade off between anomaly detection performance and scalability, show that our method offers significant advantages over competing methods.**

***Index Terms*—Anomaly Detection, Unsupervised Learning, AutoML, scalability, Big Data.**

I. Introduction

ANOMALY detection problems are present in numerous domains and research fields. These can be found in industrial machinery failure [1], [2], [3], network intrusion [4], [5], [6], [7], credit card fraud [8], [9], [10], medical and public health [11], [12], [13], image processing [14], [15] and others [16], [17], [18], [19], [20]. Anomalies are events that differ from the majority of the data substantially enough as to indicate that they have been generated from a different process. Its minority nature is problematic, hindering the use of supervised machine learning (ML) methods since it is difficult to find or build data with these labeled events. As a solution, unsupervised techniques can be applied to deal with this problem. These type of anomaly detectors can be trained on unlabeled data to model normal data, which allows to identify patterns that deviate from normality.

The literature records a wide variety of anomaly detection methods that can be categorized according to their approach [21]. One of such categories encompasses proximity-based algorithms, which are methods that can detect anomalies by measuring their proximity to normal data points. Elements distant from all others can be regarded as anomalies. This category can be further divided into two others: distance-based, which are methods that rank elements according to the distance to their neighbors, and density-based methods, being this latter category where our proposed method fits. The general idea of density-based methods is to compare the density around each data point with that of its local neighbors. The working assumption is that points located in low density regions have a high probability of being an anomaly, that is, the density around a normal point is similar to the density around its neighbors and considerably different from the density around an anomaly [22]. Our method is integrated in this category since its main characteristic is the random split of data into distinct density groups. LSHAD analyzes the density of each data point to infer an anomaly score. There are several density-based anomaly detection methods covered in the literature, such as Local Outlier Factor (LOF) [23], and some of its variations [24], [25], Local Outlier Correlation Integral (LOCI) [26], Local Outlier Probability (Loop) [27] and Adaptive kernel density-based [28].

During the last years, anomaly detection models are becoming more popular, in part due to the Big Data context, as datasets are being increasingly large, and unlabeled situations are most common. Data may come from different sources, connected devices such as cell phones, fleets of vehicles or industrial machinery. In these situations, an anomalous behaviour could be related to a machine in a factory that is on the verge of malfunctioning, or a member of a vehicle fleet that has experienced unusual environmental conditions. Performing anomaly detection in large amounts of data is a difficult task, as it requires considerable computational resources. One solution is the development and application of distributed anomaly detection methods.

When dealing with large amounts of data, the distributed paradigm allows parallel computations to be performed on the data. It is focused on distributing the data across different nodes, where each node operates on the data in parallel. The immutable nature of distributed operations, such as in the Apache Spark framework, helps attain consistencies in computations. For instance, let's assume that it is performed on operations such as sum of all n elements of a given dataset, and the time for a single operation is t time units. In the case of sequential execution in one processor, the time taken by the process will be $n * t$ time units as it sums up all the elements. However, if this job is executed using 4 processors, the time taken would be reduced to $(n/4) * t$ plus merging overhead time units. Thus, scalability is becoming a must for this type of algorithms, although at present only a few of them are able to cope with large datasets [29], [30], [31].

Another field that has emerged in recent years is the Automated Machine Learning (AutoML) [32], [33]. Almost every ML method has hyperparameters, and thus one of the

J. Meira, C. Eiras-Franco, V. Bolón-Canedo, A. Alanso-Betanzos work in CITIC, Universidade da Coruña, A Coruña 15071, Spain

J. Meira and G. Marreiros work in GECAD, Institute of Engineering Polytechnic of Porto (ISEP/IPP), Portugal.

First author email: j.a.meira@udc.es

main tasks that needs to be accomplished is the optimization of these hyperparameters so as to maximize the performance of the algorithms. AutoML aims at automatically set these hyperparameters to optimize performance. In this way, the human effort necessary for applying those methods is reduced, and a more rapid and simple solution is allowed.

In this work we propose a novel density-based method for anomaly detection based on the Locality Sensitive Hashing (LSH) technique. Our method was developed to address the mentioned problems regarding the difficulty of processing the large amounts of data generated day after day, and the scarcity of unsupervised methods focused on anomaly detection problems capable of automatically adjusting their hyperparameters. Therefore, the main contributions of our work are the following:

- The adaptation of the LSH technique for anomaly detection problems (LSHAD) in large datasets.
- The model incorporates a process for autotuning of its hyperparameters. This is a relevant issue, since the success of ML heavily relies on human machine learning experts who select, among others, the appropriate hyperparameters. This is a very complex and time-consuming task, even for experts. This issue becomes more critical when it has to be carried out by non-experts, which happens quite often. Therefore, there is a high demand for the so-called AutoML methods [34].
- The proposed method is developed in the Apache Spark framework using the MapReduce approach for distributed environments, thus rendering a distributed algorithm.

This method presents a quick and effective solution when the objective is to process large amounts of data in search of anomalies. The proposed method is able to obtain similar (in some cases better) performance in detecting anomalies compared to other methods. It also stands out for its fast configuration, as it does not need to tune hyperparameters, and for its ability to handle large amounts of data, as there are few available implementations of these methods in the literature with this capability.

The rest of this paper is organized as follows: Section II describes the LSH technique introduced by Indyk and Motwani [35] and applied to our algorithm; Section III reviews state-of-art methods used in anomaly detection; Section IV gives an explanation of LSH detailed functionalities and four different types of estimators are also proposed. Section V describes our algorithm, explaining as well the automatic hyperparameter tuning process implemented. An experiment is also presented to identify the best proposed estimator. Section VI describes the evaluation of our method and compares it with others algorithms in terms of anomaly detection and execution time performance. Finally, Section VII summarizes the main conclusions and describes our ideas for future work.

II. Background

LSH was first introduced by Indyk and Motwani [35], and its basic concept is to identify approximate nearest neighbors through the use of hash functions. This technique works with the basic principle that when two points in the feature space are close to each other, they are very likely to have the same hash function. Formally defined by Indyk and Motwani [35] as follows:

Definition 1: Given a space R^{dim}, and distances thresholds $r_1; r_2$, a family $H = \{h : R^{dim} \to U\}$ is called $(r_1; r_2; P_1; P_2)$-sensitive if for any two points $p; q \in R^{dim}$ it satisfies:

$$\text{if } \|p - q\| \leq r_1 \text{ then } P_H[h(q) = h(p)] \geq P_1,$$
$$\text{if } \|p - q\| > r_2 \text{ then } P_H[h(q) = h(p)] \leq P_2.$$

This first condition above states that nearby objects within distance r_1 will collide in the same bucket with high probability, whereas the second condition declares that distant objects are hashed to the same bucket with small probability. In order for a family H to be useful it has to satisfy $P_1 > P_2$ and $r_1 < r_2$.

A h hash function from H is generated by the concatenation of various L random projections, a user-specified parameter, (explained in section IV), $h = < proj_1; proj_2; :::; proj_L >$. As shown in Definition 1, the method is probabilistic, so the problem of false neighbor detection needs to be dealt with. A common practice to alleviate this, is making the hashes more specific by increasing L. However, when hashes are very specific, many points end up in different buckets than some of their neighbors. To help with this issue, T hashes are generated for each point. The impact of L and T in the performance of the proposed algorithm is studied in Section V. LSH allows to speed up the search for neighbors by requiring much less computational effort than the brute-force approach of measuring every possible pairwise distance. This technique and variants have already been successfully applied in a variety of practical scenarios such as computer vision [36], computational drug design [37], linguistics [38], biology [39], or web clustering [40].

As mentioned before, we have implemented the LSH technique in our anomaly detection algorithm. The goal is to apply LSH to rapidly retrieve neighbor counts to be used as a ranking score for detecting anomalies. We make the assumption that points with few neighbors are very likely to be considered as anomalous. An advantage of using this technique is its speed when processing data in high dimensional spaces. Combining this property with the distributed implementation of our method in Apache Spark makes our proposed algorithm highly efficient in terms of scalability.

III. Related Work

In this section we present a set of existing work related to anomaly detection and outlier detection algorithms, since their definitions are very similar [41]. Firstly, we describe the general mostly used and recent methods in this task, then we focus on density-based related methods, and lastly some LSH variants approaches are detailed.

Liu et al. [42] developed an algorithm named Isolation Forest, that works with binary trees. Each tree is created by partitioning the instances recursively, by randomly selecting a split value for a specific attribute. The path length of the tree is used as an anomaly score, where data points with shortest path lengths are considered anomalies.

One Class SVM (Support Vector Machine) [43], a variant of the classical SVM algorithm, is another method that can be

applied for anomaly detection problems. It relies on finding the smallest hypersphere containing all the training examples after being mapped by a kernel function. There are different ways to fit a SVM model, such as:

Training with data from different classes;
Training with data from unknown classes;
Training with data from one class.

As this is a One Class method, it means that all the data in the training set are represented by only one class. In anomaly detection problems it is usually employed for training the data that belongs to the non-anomalous class, as they are commonly available. Basically, this algorithm separates all the data points from the origin and maximizes the distance from the hypersphere to the origin. This results in a binary function that captures regions in the input space where the probability density of the data is high [43].

Deep learning methods are also widely used in anomaly detection problems [44], [45], [46]. The Autoencoder is the most common architecture employed in anomaly detection problems [44], [47], [48], [49]. This method is trained in order to make the output features the same or very similar to the input features [50]. Autoencoders are composed of two parts: the encoders layer(s) and the decoders layer(s). In the first part, the algorithm compresses the input into a latent-space representation, and in the second part it reconstructs the output from this representation. This method computes the anomaly ranking score through the reconstruction error metric which measures the difference between the input and output data.

A recent work was presented by Eiras-Franco et al. [29] proposing the ADMNC (Anomaly Detector for Mixed Numerical and Categorical inputs) algorithm, which targets data with both categorical and numerical variables. The model is trained through a maximum likelihood objective function optimized with stochastic gradient descent, and is prepared to deal with large amounts of data since the algorithm lends itself well to parallel computation as it was implemented on Apache Spark.

Concerning density-based methods, Breunig et al. [23] proposed the LOF algorithm which searches for anomalous data points by measuring the local deviation of a given point with respect to its neighbors. The basic concept of this method inspired other works such as LOCI [26], that aims at a Fast Outlier Detection using the Local Correlation Integral. It is an improved method that is able to identify not only outliers but also groups of outliers, providing an automatic cut-off to determine whether a point is or not an outlier. Its main drawback is its quadratic complexity, making it computationally expensive and thus being unable to deal with large datasets.

Regarding LSH methods, they have recently been applied to anomaly detection problems. Wang et al. [51] proposed an LSH framework for ranking points according to their likelihood of being an anomaly. In their approach, the data is first split in clusters and then a ranking of points is computed by building LSH tables. Thereafter each point is evaluated according to its rank to isolate a certain amount of anomalies. This ranking mechanism is based on the number of points that are hashed to the same bucket, assuming that points in buckets with few elements are likely to be anomalies. The authors argue that with their approach they could isolate the top anomalies very quickly, usually by scanning less than 3% of the dataset. Even though in their empirical study their method outperformed the rest, they only compared it with two other outlier detection methods.

The work from Pillutla et al. [52] presented an approach in which LSH is used to prune non-outlier data points according to their redundancy in the hash table. Their algorithm then processes the data using the pruned points, making this approach computationally less expensive. The authors developed a distributed system for their algorithm. They evaluated their method performance in outlier detection and in communication time, but they do not compare their method with other algorithms in this field.

Zhang et al. [53] proposed a density biased sampling approach using LSH for counting the neighbors and attaining a scalable density estimation. They also propose a parameter tuning rule, specific to outlier detection for LSH. The authors formally investigated density biased sampling for outlier detection. They argued that given the different importance to data points according to its density, as in density sampling, it will have a higher impact in the outlier detection performance compared to uniform sampling. The authors analyzed their argument with an empirical study in which they compared their approach to uniform sampling.

The works presented by Wang et al.[51], Pillutla et al.[52] and Zhang et al.[53] described in this section use LSH techniques for outlier detection problems. Despite the similarity with our method, we identified the following differences, thus highlighting our work:

Table I
CHARACTERISTICS OF DIFFERENT ANOMALY DETECTION ALGORITHMS

Methods	Auto-Hyperparameter	Distributed
One Class SVM	Yes[1]	No
LOF	Yes[1]	No
LOCI	No	No
Wang et al.[51] method	No	No
Wang et al.[51] method	No	No
Zhang et al.[53] method	No	No
PA-I	No	No
Autoencoder	No	No
IForest	No	No
ADMNC	No	Yes
LSHAD	Yes	Yes

Although the results by Wang et al.[51] show that their method is more scalable than others included in their study, the authors did not mention whether their method is capable of performing distributed computing (same for Pillutla et al.[52]). The implementation of our method in Apache Spark enables the use of data processing distributed among the various cores of the processor(s), allowing the users to deal with larger amounts of data than its competitors.

Our method is able to adjust its hyperparameters automatically, relieving the user of this time-consuming task and contributing to the field of AutoML.

[1]It has several hyperperameters, with only one being automatic tuned

We present a much broader experimental study by comparing our method with a wide range of datasets and different anomaly detection methods (see Section VI for the complete experiments).

Table I represents the differences of the methods described taking into account the auto-hyperparameter tuning characteristic and whether they can perform distributed computing.

IV. Proposed Method

In this section we describe the use of the LSH technique to identify anomalies. In order to understand the development of the automatic hyperparameter tuning mechanism implemented in our method, we also explain in this section each hyperparameter and its impact for the process of generating random projections and creating groups of neighbors. Finally, we present several density estimators for measuring the number of neighbors for each data point.

The main idea behind our method is to obtain an estimate of the density of the different input space regions quickly and inexpensively thanks to distributed computation. We achieve this by following the MapReduce approach implemented in Apache Spark [54]. Our proposed method leverages the LSH technique by applying hash functions to group the data points in buckets with their neighbors. Then, the number of neighbors of each bucket is used as to compute several evaluation metrics that will act as a score that ranks elements according to their level of anomaly. This process is described in Algorithm 1. First, a suitable set of hyperparameters is obtained using the tuning procedure described in Section V (Line 1). Then a hasher, consisting of L T hyperplanes is created and used to obtain the hashes for each element in the training dataset D. The number of elements corresponding to each hash is counted and used to compute an estimator (Line 3). Finally, the estimator values are used to establish a threshold under which a point will be considered an anomaly. The threshold is selected so that the number of elements that fall below it corresponds with the anomaly ratio of the training data, which must be provided by the user.

Algorithm 1: Pseudo-code for LSHAD. Training phase.

```
Input : D              Set of training points,
        anomalyRatio   Fraction of the dataset expected to be anomalous
Output: hasher         set of hyperplanes to obtain hashes,
        estPerHash     dictionary associating each hash with its estimator value,
        threshold      estimator value used to deem an element to be anomalous
1 L, T, w      ← tuneHyperparameters(D);
2 hasher       ← new HASHER(L, T, w);
3 estPerHash   ← hasher.HASHANDESTIMATEPERHASH(D);
4 threshold    ← COMPUTETHRESHOLD(estPerHash, anomalyRatio);
```

Algorithm 2: Pseudo-code for LSHAD. Detection phase.

```
Input : p              Test point,
        hasher         Hasher of the trained model,
        estPerHash     Estimator dictionary,
        threshold      Estimator threshold
Output: Boolean value indicating whether p is an anomaly
1 hashes   ← hasher.HASH(p);
2 estimator ← 0;
3 foreach h ∈ hashes do
4     estimator ← estimator + estPerHash[h];
5 end
Result: estimator < threshold
```

Once the model (consisting of an estimator value for each hash, a set of projection hyperplanes and a threshold value) is fitted to the training data, assessing whether a test point is an anomaly follows the process described in Algorithm 2. First, the hashes for p are computed (Line 1) and then the estimator values corresponding to the assigned hashes are accumulated. If the resulting value does not reach the threshold established by the learned model, then p is detected as an anomaly.

Once the model is trained, predictions can be made using the Algorithm 2. Checking a test point requires generating all its hash values with the hasher, and an estimator is calculated by using the precomputed counts in the model, which represent the properties of the training data distribution.

A. Hashing procedure

Although the LSH techniques that we use can take advantage of many LSH hash families, this particular implementation is based on the Euclidean distance, so we selected the classical hash function used in this scenario, formally computed by the following equation:

$$proj(x|\ ;\ ) = \left\lfloor \frac{x \quad + \quad}{w} \right\rfloor \tag{1}$$

The projection $proj(x|\ ;\ ) : R^d \to Z$ maps a d dimensional vector x, representing each data point, onto the set of integers, where $\ $ is a random vector drawn from a Gaussian distribution and $\ $ is a real number uniformly chosen from the interval $[0 : w]$. This scalar projection is then quantized into a set of hash bins, grouping in the same bin all elements that are close together in the original space. The user-specified hyperparameter w in Equation 1 represents the resolution of the said quantization. Figure 1 shows one such hash function that consists of a random projection in two dimensions with a specific w value.

A hash function is represented by such L random projections, defining the hash value (Equation 2):

$$H(x) = < proj_1(x|\ _1;\ _1); \ldots; proj_L(x|\ _L;\ _L) > \tag{2}$$

Where $proj_i(x|\ _i;\ _i)$, $1 \quad i \quad L$ (from Equation 2) is computed by Equation 1. After generating all the hash

Figure 1. Random projection in two dimensions

functions, the observations with the same hash values are grouped together.

Using the same notation from De nition 1 in Section II, in order for a family H to be useful it has to satisfy the condition that the probability of P_1 is much higher than P_2. Hash functions $H(x)$ will, in some cases, not ful l this condition, especially given that they are generated at random. To ensure that $P_1 > P_2$ while taking into account the probabilistic properties of $H(x)$, T hash tables are created, where a single hash table indicates the hash of each data point. As a result, every point x will receive a set of hashes $\{H_1(x); H_2(x) ::: H_T(x)\}$. When grouping elements that obtained the same hash, the method creates groups of elements that have a high probability of being close together. However, when increasing the value of parameters L and T, the computational complexity of the algorithm also increases, since more hashes need to be generated. Therefore it is necessary to identify suitable values for these parameters, such that accurate detection is obtained with as little computational effort as possible.

Figure 2 shows an example of a hash table $H(x)$ where the data points are on the left and the hash table on the right. Each row represents different hash values and the second column the collisions, that is, elements that share the same hash value.

Figure 2. Hash Table example

B. Anomaly level estimation

Once a suitable hasher has been found, the next goal is to estimate the density of the regions of the input space represented by each hash. Points hashed to low estimator buckets will be deemed as anomalous. We explored the use of 4 different of density estimators. Let D be the input dataset and $b_h = \{x \in D; H(x) = h\}$ the set of points that obtained hash h in one of the tables t. With these de nitions, we de ne the set of neighbors of point x as the set of elements in the dataset that share a hash with x across all T tables: $neigh(x) = \bigcup_{t=1}^{T} b_{H_t(x)}$.

Estimator A represents the number of points in the bucket:

$$E_A(h) = |b_h| \tag{3}$$

Estimator B is the average number of neighbors of the points contained in the bucket:

$$E_B(h) = \frac{\sum_{x \in b_h} neigh(x)}{|b_h|} \tag{4}$$

Estimator C represents the ratio between $E_A(x)$ and $E_B(x)$:

$$E_C(h) = \frac{E_A(x)}{E_B(x)} \tag{5}$$

Estimator D represents the sum of the inverse of the number of neighbors of all points in the bucket:

$$E_D(h) = \sum_{x \in b_h} \frac{1}{neigh(x)} \tag{6}$$

Further ahead in the experimental Section V we analyse all the proposed estimators evaluating their validity with the aim of identifying which one gives the best anomaly ranking score.

V. Hyperparameter Tuning and Experimentation

Progressive automation of Machine Learning, AutoML, has been a hot research topic for the past few years [32]. Applying traditional ML methods is time-consuming, resource-intensive, and challenging. One of the challenges of this topic is the manual hyperparameter tuning process, which besides of being very computationally expensive, also has a great in uence on the nal results of the algorithm.

Thus, in the case of our proposed algorithm, LSHAD, it might be dif cult for a non-expert user to tune hyperparameters in an environment where only unlabeled data is available. To overcome these challenges, we have carried out an implementation of an automatic hyperparameter tuning technique in our method. For this process we started by studying the behavior of each user-speci ed parameter, that is, the resolution of quantization bins w, the number of random projections L, and the number of tables T.

A. Datasets for Experiments

To analyse the hyperparameters and evaluate various anomaly detection methods, we selected several datasets widely used in the literature for classi cation tasks, but adapted for anomaly detection problems. The datasets, presented in Table II, were downloaded from the UCI Machine

Table II
DATASETS SELECTED FOR THE ANALYSIS ON HYPERPARAMETERS TUNING AND EVALUATION OF ANOMALY DETECTION

Synthetic Datasets	Samples	Features
2 banana clusters (2BC)	1,000	2
2 circular clusters (2CC)	1,000	2
2 point clouds with variance (2PV)	1,000	2
3 anisotropic clusters (3AC)	1,000	2
3 point clouds (3PC)	1,000	2
Real Datasets (Small)	-	-
Abalone 1-8 (Ab. 1-8)	4,177	11
Abalone 9-11 (Ab. 9-11)	4,177	11
Abalone 11-29 (Ab. 11-29)	4,177	11
Arrhythmia (Arrhyth)	420	278
German Credit (GC)	1,000	20
Heart	270	14
Pima Diabetes (Pima)	768	9
Breast Cancer (Breast)	683	10
Real Datasets (Medium)	-	-
CoverType (CT)	56,911	12
KDDCup99 (KDD99)	44,000	41
KDDCup99 (http) (KDD99h)	64,293	40
KDDCup99 (smtp) (KDD99s)	97,23	40
IDS 2012	42,301	27
IOT-23 sample (ID dataset: 1)	44550	18
Real Datasets (Large)	-	-
IOT-23 (ID: 1)	1,008,749	18
IOT-23 (ID: 3)	156,101	18
IOT-23 (ID: 7)	11,454,723	18
IOT-23 (ID: 9)	6,378,294	18
IOT-23 (ID: 17)	54,659,864	18
IOT-23 (ID: 33)	54,454,592	18
IOT-23 (ID: 35)	10,447,796	18
IOT-23 (ID: 36)	13,645,107	18
IOT-23 (ID: 39)	73,568,982	18
IOT-23 (ID: 43)	67,321,810	18
IOT-23 (ID: 48)	3,394,347	18
IOT-23 (ID: 49)	5,410,562	18
IOT-23 (ID: 52)	19,781,379	18
IOT-23 (ID: 60)	3,581,029	18

Learning [55], Zenodo[2] and Stratosphere Research Laboratory[3] Repository.

Regarding the UCI Machine Learning Repository datasets, we have employed a version of Abalone, where different classes of this dataset were considered as anomalies or non-anomalies. In this case for Ab. 1-8, the classes 1 to 8 are considered anomalies and the rest of the classes are non-anomalous. Regarding Ab. 9-11, we considered classes 9,10,11 as anomalies for Ab.9-11 where the other classes represents non-abnormal behavior. For Ab.11-29 the anomalous classes are classes 11 to 29. A sample of 20% of the CoverType dataset was also used where instances of class 2 were assumed to be normal, while instances of class 4 were selected as anomalies. We also selected other datasets from the same repository, such as German Credit, Arrhythmia, Pima diabetes, Breast cancer, Heart, three versions of the KDDCup99 dataset and finally a sample of IDS 2012, an update of the KDDcup99 solving some of its problems. For the German Credit dataset we consider the class representing people with a bad credit risk as anomalous. For health-related datasets, patients with the specific involved disease were considered to be an anomalous class. Regarding KDDCup99 we have employed the following versions: KDD99 (a sample of the full KDDCup99 with all cyberattacks), KDD99-SMTP (reduced KDDCup99 filtering only smtp connections), KDD99-HTTP (reduced KDDCup99 filtering only http connections). For the KDDCup versions and the IDS 2012 datasets, we considered any sort of intrusion as anomaly, as opposed to normal traffic.

From the Zenodo repository we have used a synthetic dataset of two-dimensional combinations of attributes of clusters of different shapes (see Figure 3).

Figure 3. Synthetic dataset shapes representing: 2 circular clusters (2CC), 2 banana clusters (2BC), 3 point clouds (3PC), 2 point clouds with variance (2PV), 3 anisotropic clusters (3AC).

Regarding Stratosphere Research Laboratory repository we used a new released dataset named as Aposemat IoT-23. This dataset contains malicious and benign network traffic of real IoT devices where we consider attacks as anomalies.

To analyse the behavior of the hyperparameters and test the proposed estimators 8 real datasets were used, namely, Ab.1-8, Arrhythmia, German Credit, CoverType, all KDDcupp99 versions and IDS 2012. Finally, to evaluate and compare the anomaly detection methods performance we have used all datasets as described in Section VI.

B. Hyperparameters Analysis

In this section we show the experiments carried out to analyse the behavior of each hyperparameter and its impact on the performance of our method. The purpose of this study is to identify a regular pattern or a correlation between the performance of our algorithm and the values of the hyperparameters to build an automatic tuning mechanism.

For all the experiments a five fold cross validation was performed, computing the average for each metric used. For this particular study, the datasets were changed in order to retain 1 % of anomalies. The purpose of this change is two-fold: (1) provide an accurate anomaly ratio to the algorithm and (2) ensure that normality is learned by keeping the number of anomalies low. In a real case it would not be possible to do this, and the user would have verify that the training dataset represents normality and would need to provide an estimation of the anomaly ratio for the training dataset.

In the first step of our analyses, we fixed a constant value for the parameter w, and tested the performance of the algorithm

[2]https://zenodo.org/
[3]https://www.stratosphereips.org/

using the metric Area under the Curve (AUC) for one of the proposed estimators for different values of L and T.

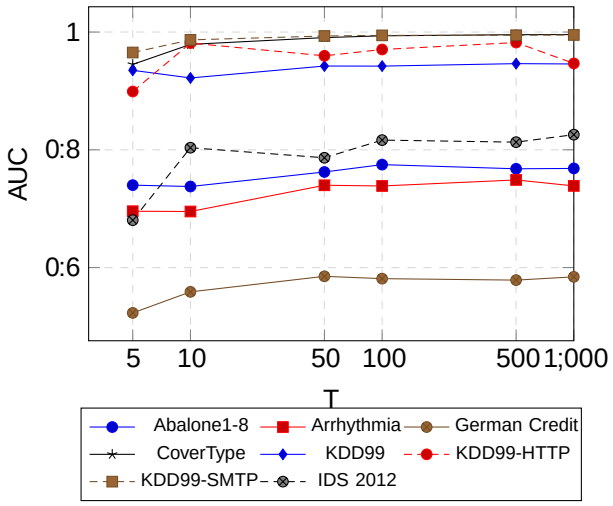

Figure 4. LSHAD Performance (AUC) varying the hyperparameter T, number of tables

Analysing the results obtained it was observed that by increasing the number of tables (T), the performance of LSHAD showed a very similar pattern regardless of the number of random projections, L, to be generated. For visualization purposes in Figure 4 we plot the LSHAD performance measured in AUC with two random projections, $L = 2$ (However we tested a wide range of values, from 1 to 128, for each different T values), for different values of Hash tables (from 5 to 1,000). As can be observed, the performance of LSHAD increases as the number of Hash Tables (T) increases, until it stagnates when reaching a certain AUC value. This behavior was already expected, as mentioned in Section II, repeating this random process several times will increase the likelihood that two similar data points will collide in the same bucket. As can be seen in Figure 4, for most datasets the LSHAD performance showed a significant increase at $T = 50$. For higher values the performance is maintained, or slightly increased and, even in some cases, the repetition created by a high number of tables leads to a decrease in performance.

In Figure 5, the model's AUC is shown versus different values of L (for a fixed T=50). It can be seen that different L values have a small impact on the performance of LSHAD for the CoverType[4] and all versions of KDD99 datasets. For Abalone1-8, there is a great decrease in performance with more than 8 random projections and for IDS 2012 a big increase in performance with more than 16 random projections. Regarding Arrhythmia, performance drops with more than 32 random projections. The difficulty of the German Credit dataset lead to poor results, so no conclussions could be extracted regarding the effect of L in this dataset. Taking into account the above, we fixed the parameter $L = 4$, since it is an acceptable value when balancing the trade-off between performance and computational resources. Shorter

[4]It has a similar performance to KDD99-SMTP but is not visible in the Figure 5 since it is behind KDD99-SMTP line.

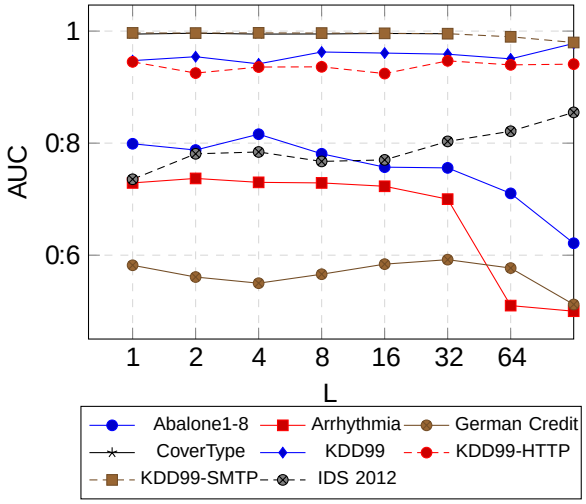

Figure 5. LSHAD Performance changing the hyperparameter L, number of random projections

hashes require less memory when saving the model and can be processed faster. The selected value can provide better processing speed without sacrificing too much performance.

In the next experimental step, we carried out the same test, but this time fixing the values of both parameters T and L. We define the parameter $T = 50$, because as noted in Figure 4, when T is greater than 50, performance improvement is not significant and we define $L = 4$ for the reasons mentioned before.

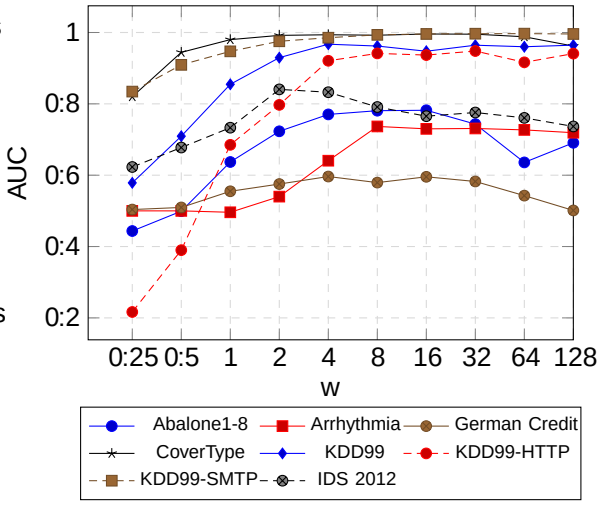

Figure 6. LSHAD Performance changing hyperparameter w

With these fixed values, we analysed the effect of w, the length of quantization bins. Figure 6 shows that, in those conditions, w has great impact in the detection accuracy. However, the optimal value for w depends on the characteristics of the dataset. Consequently, when $T = 50$ and $L = 4$, performance can be optimized by simply tweaking w. This simplifies the hyperparameter tuning process by reducing it to finding a suitable w value for the given dataset.

Still, when given an unlabeled dataset, the effect of w in

detection accuracy can not be directly observed, since no labels are available to measure performance. Thus, we extracted other indirect metrics that could provide us more information. The following unsupervised metrics were extracted in order to observe if there is a correlation between them and the performance of the algorithm:

Bucket Count (BC): number of buckets generated.

Average Bucket Size (ABS): Let $|D|$ be the cardinality of the input data and b each bucket size from B buckets generated, then:

$$ABS = \frac{\sum_{b \in B} \frac{b}{BC}}{|D|} \quad (7)$$

Average Bucket Distance (ABD): Average Euclidean distance of the first element in the bucket to its neighbors.

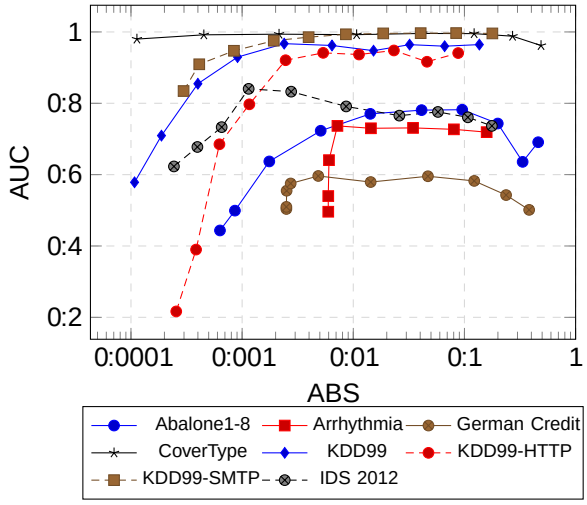

Figure 7. LSHAD Performance for different w values and displaying the ABS metric in horizontal axis

In order to assess the suitability of these metrics, we explored several w values for all datasets and plotted each metric vs. the AUC. No pattern was observed for the BC and ABD metrics, but ABS was found to contain useful information. For the majority of the tested datasets, the performance of LSHAD reached very high values when ABS was in the range $[0.05, 0.1]$, as can be observed in Figure 7. ABS can, therefore, be used to find a suitable value for w, since a w value that lands ABS in the $[0.05, 0.1]$ range will be likely to achieve good detection accuracy. Moreover, since the effect of w on ABS is known (a larger w increases ABS, while a smaller w decreases ABS), the search for a suitable w can be performed efficiently.

C. LSHAD with Auto Hyperparameter tuning

Algorithm 3 depicts our LSHAD model, which takes into account the ABS metric above. It begins by estimating a suitable value of w using a binary search.

To do so, a search interval must first be set. The inferior threshold is always set to 1 (line 2), while the upper threshold is found by doubling the w value, using it for hashing until it produces small enough buckets (line 3). That range is then explored using binary search (loop on line 7) to find a w that produces buckets with an average number of elements between 0.05 and 0.1 times the size of D. Once found, L, T and the retrieved w are reported as the tuned hyperparameters.

Algorithm 3: Pseudo-code for LSHAD: Hyperparameter training procedure.

```
Input  : D      Set of training points
Output: L; T; w      tuned hyperparameters
1  L    4, T    50;
2  wCandidate    1, avBucketSize    0,
     leftLimit    1, rightLimit    1;
3  while avBucketSize < 0.05 do
4  |   avBucketSize
   |     HashGroupAndCount(D; L; T; wCandidate);
5  |   wCandidate    wCandidate  2;
   end
6  rightLimit    wCandidate;
7  while avBucketSize < 0.05 or avBucketSize > 0.1
     do
8  |   wCandidate  b (leftLimit  + rightLimit )=2c;
9  |   avBucketSize
   |     HashGroupAndCount(D; L; T; wCandidate);
10 |   if avBucketSize < 0.05 then
11 |   |   leftLimit    wCandidate;
   |   end
12 |   if avBucketSize > 0.1 then
13 |   |   rightLimit    wCandidate;
   |   end
14 |   if leftLimit >  = rightLimit  then
15 |   |   break;
   |   end
   end
   Result: L; T; wCandidate
```

D. Estimators Experimentation

Regarding the proposed estimators in Section IV, which are $E_A(h)$; $E_B(h)$; $E_C(h)$; $E_D(h)$, we compared their AUC performance for each given dataset and evaluated them to find out which produced the best ranking score in the classification. We plotted a graph showing the AUC score for each estimator in Figure 8, where it can be seen that they all produced similar results with small variations in the AUC in the different datasets. In fact, using the statistical Nemenyi post-hoc test [56] with $= 0.05$, the scores achieved by the estimators could not be significantly differentiated (see Figure 9). Even though there is no statistical difference, we choose our algorithm to use the C estimator by default, since it was the one with the lowest critical difference value.

VI. Performance Evaluation

In this Section we evaluate LSHAD algorithm and compare it with other methods, both in terms of processing time and anomaly detection performance.

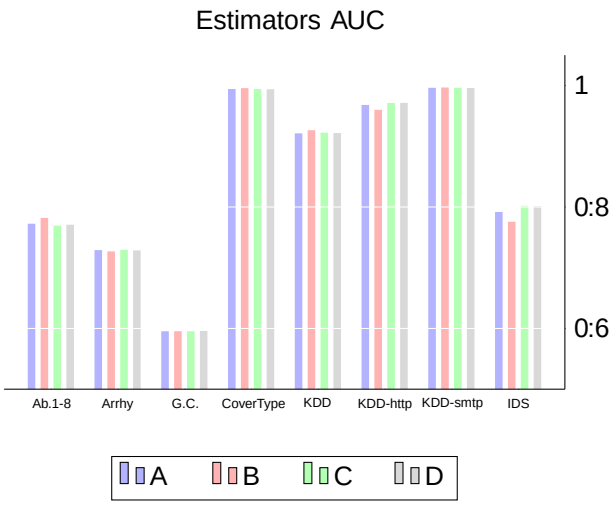

Figure 8. Proposed estimators AUC score

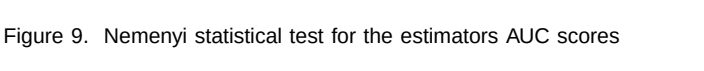

Figure 9. Nemenyi statistical test for the estimators AUC scores

A. Methods Evaluated and Datasets Used

To measure and compare the performance of our method against others, a variety of state-of-the-art algorithms, many of which were mentioned in Section III, have been selected. As LSHAD is a density-based method we considered the well known LOF and LOCI methods, using Euclidean (E), Jaccard (J) and Hamming (H) distances, for which we employed a Matlab implementation[5]. For the same category we also selected the approach by Zhang [52], as it also uses the LSH for scalable density estimation; in this case we tested their JAVA implementation[6] of four algorithms using their piecewise density biased sampling (PDBS), namely:

- 1 Sample(PDBS)- drawing one sample for all points to compute k-NN distance;
- Iterative(PDBS) - drawing one sample for each point to compute k-NN distance;
- Iterative+Ensemble(PDBS)- drawing multiple samples for each point to make ensembles for k-NN distance
- IForest(PDBS) - using the Isolation Forest detection method

A variety of other methods related to unsupervised anomaly detection were also selected for our evaluation: An auto-encoder implementation in Python using H2O framework [57], the One-Class SVM with Radial basis (SVM-R) and linear (SVM-L) kernel functions, for which we used the Matlab interface of LibSVM[8]; a distributed version of SVM that can handle large datasets (SDOC-SVM) [58]; an online One-Class classi cation with a passive-aggressive kernel (PA-I) [59], also built-in Matlab; and nally the ADMNC implemented in Scala Apache-Spark[9].

All these methods were tested with several datasets containing different compositions, in order to observe the algorithms behavior in a variety of scenarios. First we used a simple synthetic dataset suite[10] with only 3 dimensions and 1000 samples, where each synthetic dataset represents different shapes as described in Table II. The real datasets described in Section V were also used.

In our experiments we performed a 5-fold cross-validation, ltering for each dataset around 1% of the class anomaly samples, with the aim of simulating a real anomaly detection problem scenario, as we did in Section V to test the different hyperparameters. Note that to overcome computational dif - culties of some methods, we only used 2 folds instead of 5 for testing the medium datasets. In addition, as we use the AUC metric to evaluate our method, we ignore the threshold variable described in Section V-C and use the anomaly score provided by the estimator as the LSHAD output. For the experiments a MacBook-pro laptop with 8GB of RAM memory with a 2.9GHz Intel Dual-Core i5 processor was used.

B. Comparing Anomaly detection Performance

For visualization purposes we split the tables according to the datasets structure. In Table III we show the AUC score results for the synthetic datasets, in Table IV the algorithms performance for small datasets with less than 5000 samples (Abalone's datasets, Arrhyth, GC, heart, Pima, Breast) and nally, Table V contains the performance results for medium datasets, with more than 5000 samples (CoverType, KDD-cup99's datasets, IDS 2012).

1) Testing synthetic datasets: First we carried out a comparison of the performance of our algorithm and the other algorithms described above, over a simple classi cation task, applying 5 datasets of different shapes with 2 dimensions each, represented in Figure 3. Observing the results presented in Table III, it is possible to verify that our LSHAD method obtains state-of-the-art results in the identi cation of the anomalies from the different data shapes, except for the 2 circular clusters. LSHAD obtains the highest performance for the 2 point clouds with variance shape, but the overall best performance was obtained by the much more exhaustive LOF and LOCI methods.

2) Testing Public datasets: Regarding the small datasets in Table IV, it can be seen that the AUC scores are very variable for all methods. Although our method did not obtain the best score in any dataset, the results are among the average state-of-the-art, and in some cases very near the best. The same happens for the medium datasets in Table V. In this table, LOF and LOCI methods were eliminated from the

[5]https://github.com/jeroenjanssens/lof-loci-occ

[6]From the LSH methods presented in section III et al. This is the only algorithm available to test

[7]https://bit.ly/2ugZQ0x

[8]https://www.csie.ntu.edu.tw/ cjlin/libsvm/

[9]https://github.com/eirasf/lsh-anomaly-detection

[10]https://zenodo.org/record/1171077#.XkE-HBP7TOR

Table III
AUC RESULTS OF SELECTED ALGORITHMS FOR SYNTHETIC DATASETS

	2BC	2CC	2PV	3AC	3PC
LOF (E)	82.78	78.40	78.40	90.20	96.90
LOF (H)	83.03	78.60	79.58	90.20	96.80
LOF (J)	83.10	78.62	79.73	90.10	96.80
LOCI (E)	80.46	76.20	75.61	88.12	94.80
LOCI (H)	80.63	78.20	76.57	88.64	95.20
LOCI (J)	80.11	75.40	76.65	87.81	96.87
SVM-L	50.13	52.00	53.40	56.80	62.80
SVM-R	72.36	56.54	81.30	83.00	91.80
DOC-SVM (RBF)	55.40	51.80	59.80	66.20	60.80
PA-I	55.77	58.00	56.60	64.20	70.80
ADMNC	53.80	59.29	70.26	85.32	82.70
Autoencoder	59.80	56.70	68.54	70.32	69.79
1 Samp(PDBS)	69.71	54.38	81.47	82.70	95.08
Ite (PDBS)	68.90	55.00	80.09	83.13	95.68
Ite+Ens(PDBS)	76.18	57.68	81.22	86.51	97.06
IForest(PDBS)	61.47	56.47	66.93	78.51	72.50
LSHAD	76.34	61.07	82.29	89.30	97.34

comparison study, as their quadratic complexity made them computationally very costly compared to the other methods, and therefore unable to manage large datasets. It was also not possible to test DOC-SVM, as its Matlab implementation failed when trying to split any large dataset, and thus being also unable to process them.

3) Statistical test evaluation: Finally, we also performed a statistical Nemenyi post-hoc test [56] with $= 0:05$ to check if the overall methods have a signi cant statistical difference between them. Figure 10 shows the algorithms sorted by score. The Nemenyi test divided the algorithms in 3 groups represented as the horizontal thick lines. As we can see, our method was placed in the group of algorithms with highest performance, among which there is no statistical difference.

Figure 10. Nemenyi statistical test for evaluating anomaly detection AUC scores methods

C. Testing Scalability

To test the scalability of the methods above, we used a synthetic dataset from the generator developed by Eiras-Franco et al. [29] available here[12], in which size was varied. We started with 100 samples and performed a ve-fold increase in each iteration. Since the methods are implemented on different platforms, we measured the relative execution time as the ratio of the time taken to process the rst dataset with 100 samples and the time taken to process a speci c dataset size for each algorithm. This allowed us to approximate the empirical time complexity of each method. For this test, the LOF and LOCI methods with Hamming distance, SVM with linear and radial basis, Autoencoder, DOC-SVM, PA-I, ADMNC, and IForest(PDBS) (the latter being the fastest of the four methods using the PDBS) were selected. Plotting the execution time results of each algorithm in Figure 11 we can observe that all the algorithms can process the data very fast for small sizes of 100 and 500 samples, except LOCI and LOF, due to their quadratic complexity. DOC-SVM was unable to process the synthetic data with more than 2500 samples due to its current implementation, although it is possible to visualize that it needs longer times to process the data for small samples compared to other algorithms. The execution time of PA-I starts to increase signi cantly for data sizes with more than 2500 data points, exceeding linear complexity. SVM-L, in turn, shows quadratic complexity. IForest(PDBS) recorded acceptable execution times for small datasets, but could not handle data with more than 62500 points. SVM-R behaved similarly, performing adequately up to 12500 data points, then showing a high increase in its process time, exceeding quadratic complexity.

ADMNC, Autoencoder and our LSHAD method achieved better execution time results than the rest of models when processing the larger datasets. Finally, LSHAD is the method with the lowest complexity when handling the largest amount of data with 1562500 samples.

An experiment was also carried out with the IoT-23 dataset since it has some large subsets in the order of 6GB, rounding 70,000,000 records [60]. Only the LHSAD and ADMNC algorithms were used in this experiment, since they are the only ones in the comparison suite capable of dealing with large datasets due to their distributed computing implementation approach. The algorithms were applied to each subset of the IoT-23 dataset and a 5-fold cross validation was performed. To do so, the resources of the Centre of Supercomputing of Galicia (CESGA) were used [44], employing 22 machines with 35GB of RAM and 22 cores each.

Table VI shows the LSHAD algorithm performing slightly better than ADMNC. It is important to note that the optimal values de ned for LHSAD hyperparameters tested on medium-sized datasets (Section V) also hold for large datasets, as performance results obtained are high. This renders our algorithm an adequate one when aiming at processing large dimensional datasets, with appropriate accuracy, as it is in the group of methods with best performance (as we have shown previously), while being its scalability the best among the methods tested.

D. Evaluate scalability vs anomaly detection performance

To evaluate the trade-off between scalability and anomaly detection for all the above algorithms we applied the Pareto Optimization method [61]. In multi-objective optimization, the Pareto front is de ned as the border between the region of

[11]This is a sample of the IoT-23 Subset with ID 1
[12]http://github.com/eirasf/ADMNC/

Table IV
AUC RESULTS OF SELECTED ALGORITHMS FOR REAL (SMALL) DATASETS

	Ab. 1-8	Ab. 9-11	Ab. 11-29	Arrhyth	GC	Heart	Breast	Pima
LOF (E)	69.36	60.29	59.27	66.70	58.47	61.22	60.21	68.38
LOF (H)	69.36	60.29	59.27	69.83	56.46	69.58	59.18	68.18
LOF (J)	69.36	60.29	59.27	70.10	56.81	65.28	60.17	68.23
LOCI (E)	85.24	67.56	71.55	67.35	59.17	86.42	99.51	73.48
LOCI (H)	85.26	68.56	71.55	71.41	57.09	72.25	99.37	69.87
LOCI (J)	85.15	68.74	71.59	71.44	56.63	85.39	99.40	72.75
SVM-L	79.44	61.40	76.70	67.94	56.97	85.16	99.50	59.77
SVM-R	81.21	67.56	74.48	74.79	64.52	81.14	97.76	67.10
DOC-SVM	55.61	57.48	55.02	65.30	54.19	53.83	74.97	67.12
PA-I	84.98	65.11	71.13	69.32	62.16	71.02	69.33	55.90
ADMNC	84.53	61.20	79.30	61.40	62.76	72.31	91.34	59.20
Autoencoder	82.23	58.34	67.76	79.54	64.00	83.20	97.90	67.10
1 Samp(PDBS)	70.85	52.72	68.82	73.06	53.70	68.00	98.60	70.16
Ite(PDBS)	70.07	50.75	68.78	71.93	54.10	59.67	98.62	68.90
Ite+Ens(PDBS)	73.80	50.97	73.31	72.60	54.50	62.67	98.30	72.10
IForest(PDBS)	84.61	55.60	70.66	72.10	55.60	53.83	91.63	53.98
LSHAD	77.24	53.82	67.13	72.22	58.23	79.95	98.58	71.13

Table V
AUC RESULTS OF SELECTED ALGORITHMS FOR REAL (MEDIUM) DATASETS

	CT	KDD99	KDD99h	KDD99s	IDS	IOT-23[11]
Autoencoder	98.95	99.13	99.99	99.69	80.44	88.05
1 Samp(PDBS)	96.59	93.29	59.90	99.68	53.64	93.83
Ite(PDBS)	95.39	84.96	59.90	99.69	54.45	93.67
Ite+Ens(PDBS)	98.88	90.93	59.40	99.69	55.29	93.60
IForest (PDBS)	99.50	96.67	94.81	99.73	92.99	93.70
PA-I	99.49	98.90	99.50	95.92	96.50	73.55
SVM-R	99.53	95.35	99.91	99.32	61.61	73.96
SVM-L	95.01	69.37	99.95	99.51	80.66	77.01
ADMNC	57.94	94.05	91.62	88.26	56.75	93.29
LSHAD	99.66	97.74	99.44	99.85	87.32	93.92

Table VI
AUC% RESULTS OF LSHAD AND ADMNC FOR IOT-23 DATASETS

ID DATASET	LSHAD		ADMNC	
1	89.60	0.87	91.95	1.87
3	99.53	0.12	95.45	0.61
7	99.94	0.02	99.68	0.43
9	99.97	2.42	64.99	15.72
17	71.76	21.05	97.22	1.06
33	76.42	5.08	83.31	18.26
35	98.48	1.38	99.84	0.06
36	99.77	0.29	99.36	1.21
39	97.42	0.21	76.98	2.80
43	91.29	3.23	99.99	0.0005
48	99.78	0.19	99.55	0.78
49	99.52	0.15	99.37	0.30
52	94.19	3.55	99.61	0.57
60	99.65	0.17	99.80	0.15
Avg. AUC	94.09		93.36	

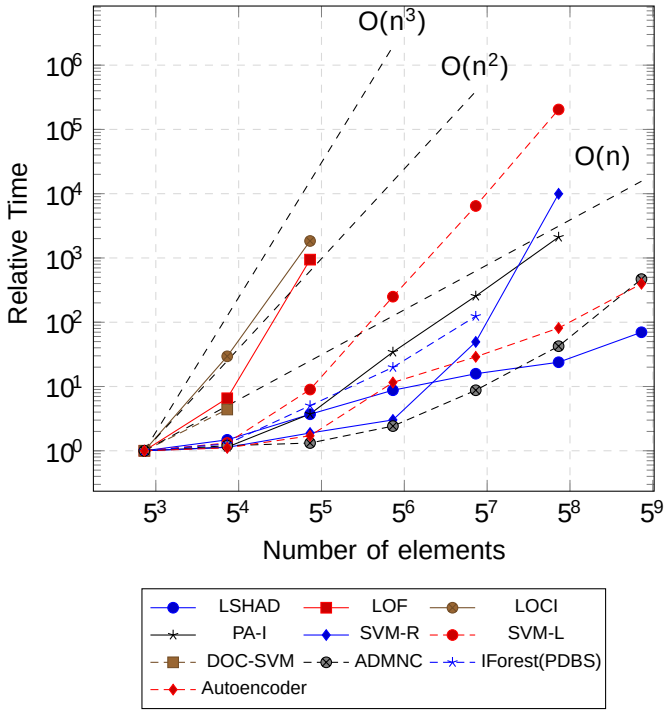

Figure 11. Execution time of each algorithm increasing the size samples of the Synthetic dataset. Axis are represented using logarithmic scale

feasible points (not strictly dominated by any other), for which all constraints are satis ed, and the region of unfeasible points (dominated by others).

We plotted in Figure 12 all the algorithms used in our study, in order to maximize the average AUC metric of all datasets (X axis) and maximize processing speed (Y axis). To compute the time complexity we used the number of samples of the largest dataset that each algorithm could handle and the time it took to process that particular dataset. Thus, it was applied $\frac{\log(t)}{\log(n)}$. Observing Figure 12 we can see that our

method, LOCI, and SVM-R are in the Pareto front, although it should be noted that LOCI and SVM-R were unable to process the largest dataset.

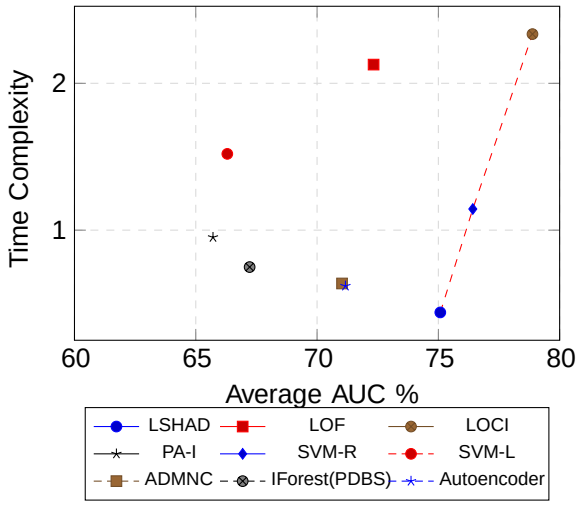

Figure 12. Pareto front of a multi-objective optimization problem based on the mean performance anomaly detection of all datasets (higher is better) vs time complexity (smaller is better)

In summary, and after all the experimentation regarding accuracy and scalability, LSHAD has demonstrated to be among the best methods of the state-of-the-art, with the additional contribution of hyperparameter autotuning, thus facilitating its use.

VII. Conclusions and Future Work

This paper presents LSHAD, a novel distributed anomaly detection method for large-scale problems. We used the LSH technique as the basis of our algorithm, in order to obtain an anomaly detection model that could handle large-scale datasets. The LSH properties allow us to find groups of similar data points. We present an approach that leverages LSH to estimate the density of the input space regions, which is, in turn, used to estimate the probability of a data point being an anomaly. Our algorithm is implemented in the Apache-Spark framework and tailored for distributed environments, being able to process large datasets due to its scalability properties. An important advantage of our method is its AutoML feature, in which an automatic hyperparameter tuning mechanism has been implemented as a solution for reducing the computational resources and time consumption of the manual hyperparameter tuning process. We described and analysed each hyperparameter and proposed an automatic tuning method.

We also empirically evaluated our algorithm and compared its anomaly detection and scalability performances with state-of-art methods in a variety of datasets.

Our study showed that our method achieves good results, comparable to the best available methods, in detecting anomalies for both synthetic and real datasets, while in terms of scalability, performs better than other methods, specially with large dataset sizes. In summary, our work presents the following contributions:

1) A novel method for anomaly detection based on LSH, that is accurate and scalable. It obtains accuracy results in par with state-of-the art methods, while displaying scalability beyond that of any of its competitors.
2) The model has been developed so as to manage distributed scenarios. As it was developed using the Apache Spark framework, the method can separate the data processing operations across multiple clusters.
3) We have provided the algorithm with a method that automates the time-consuming and error-prone process of tuning hyperparameters. This feature not only improves efficiency, but also makes the algorithm available to non-expert users in the field of ML, a scarce feature nowadays in most anomaly detection models

As future work we plan to continue our research on the capabilities of LSHAD with the intention of implementing an online version of this algorithm for dealing with data streams. Our parallelized implementation in Apache-spark framework is available[13] for anyone to improve or use in the anomaly detection field.

Acknowledgment

This research has been financially supported in part by the Spanish Ministerio de Economía y Competitividad (research project PID2019-109238GB-C2), and by the Xunta de Galicia (Grants ED431C 2018/34 and ED431G 2019/01) with the European Union ERDF funds. CITIC, as Research Center accredited by Galician University System, is funded by "Consellería de Cultura, Educación e Universidades from Xunta de Galicia", supported in an 80% through ERDF Funds, ERDF Operational Programme Galicia 2014-2020, and the remaining 20% by "Secretaría Xeral de Universidades" (Grant ED431G 2019/01).This work was also supported by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within the Project UIDB/00760/2020.

References

[1] S. E. Guttormsson, R. Marks, M. El-Sharkawi, and I. Kerszenbaum, "Elliptical novelty grouping for on-line short-turn detection of excited running rotors," IEEE Transactions on Energy Conversion, vol. 14, no. 1, pp. 16–22, 1999.
[2] R. Fujimaki, T. Yairi, and K. Machida, "An approach to spacecraft anomaly detection problem using kernel feature space," in Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005, pp. 401–410.
[3] O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Verstockt, R. Van de Walle, and S. Van Hoecke, "Convolutional neural network based fault detection for rotating machinery," Journal of Sound and Vibration, vol. 377, pp. 331–345, 2016.
[4] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles, "Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification," in Proc. IEEE Workshop on Information Assurance and Security, 2001, pp. 85–90.
[5] D. Barbará, J. Couto, S. Jajodia, and N. Wu, "Adam: a testbed for exploring the use of data mining in intrusion detection," ACM Sigmod Record, vol. 30, no. 4, pp. 15–24, 2001.
[6] K. Sequeira and M. Zaki, "Admit: anomaly-based data mining for intrusions," in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002, pp. 386–395.

[13] https://github.com/eirasf/lsh-anomaly-detection

[7] K. Labib and R. Vemuri, "Nsom: A real-time network-based intrusion detection system using self-organizing maps," *Networks and Security*, pp. 1–6, 2002.

[8] P. K. Chan, W. Fan, A. L. Prodromidis, and S. J. Stolfo, "Distributed data mining in credit card fraud detection," *IEEE Intelligent systems*, no. 6, pp. 67–74, 1999.

[9] S. Ghosh and D. L. Reilly, "Credit card fraud detection with a neural-network," in *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, vol. 3. IEEE, 1994, pp. 621–630.

[10] A. Srivastava, A. Kundu, S. Sural, and A. Majumdar, "Credit card fraud detection using hidden markov model," *IEEE Transactions on dependable and secure computing*, vol. 5, no. 1, pp. 37–48, 2008.

[11] B. Rolinski, H. Küster, B. Ugele, R. Gruber, and K. Horn, "Total bilirubin measurement by photometry on a blood gas analyzer: potential for use in neonatal testing at the point of care," *Clinical chemistry*, vol. 47, no. 10, pp. 1845–1847, 2001.

[12] W.-K. Wong, A. W. Moore, G. F. Cooper, and M. M. Wagner, "Bayesian network anomaly pattern detection for disease outbreaks," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 808–815.

[13] R. G. Stafford, J. Beutel *et al.*, "Application of neural networks as an aid in medical diagnosis and general anomaly detection," Jul. 19 1994, uS Patent 5,331,550.

[14] M. Augusteijn and B. Folkert, "Neural network classification and novelty detection," *International Journal of Remote Sensing*, vol. 23, no. 14, pp. 2891–2902, 2002.

[15] S. Singh and M. Markou, "An approach to novelty detection applied to the classification of image regions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 4, pp. 396–407, 2004.

[16] V. Emamian, M. Kaveh, and A. H. Tewfik, "Robust clustering of acoustic emission signals using the kohonen network," in *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*, vol. 6. IEEE, 2000, pp. 3891–3894.

[17] P. Crook, G. Hayes *et al.*, "A robot implementation of a biologically inspired method for novelty detection," in *Proceedings of the Towards Intelligent Mobile Robots Conference*, 2001.

[18] P. Protopapas, J. Giammarco, L. Faccioli, M. Struble, R. Dave, and C. Alcock, "Finding outlier light curves in catalogues of periodic variable stars," *Monthly Notices of the Royal Astronomical Society*, vol. 369, no. 2, pp. 677–696, 2006.

[19] T. Kudo, T. Morita, T. Matsuda, and T. Takine, "Pca-based robust anomaly detection using periodic traffic behavior," in *2013 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2013, pp. 1330–1334.

[20] J. Meira, R. Andrade, I. Praça, J. Carneiro, V. Bolón-Canedo, A. Alonso-Betanzos, and G. Marreiros, "Performance evaluation of unsupervised techniques in cyber-attack anomaly detection," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–13, 2019.

[21] V. Chandola, "Anomaly Detection: A Survey," *Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications*, vol. 41, no. 3, pp. 71–97, 2009.

[22] H.-P. Kriegel, P. Kröger, and A. Zimek, "Outlier detection techniques," *Tutorial at KDD*, vol. 10, 2010.

[23] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[24] W. Jin, A. K. H. Tung, and J. Han, "Mining top-n local outliers in large databases," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*. New York, New York, USA: ACM Press, 2001, pp. 293–298. [Online]. Available: http://portal.acm.org/citation.cfm?doid=502512.502554

[25] J. Tang, Z. Chen, A. W. Fu, and D. W. Cheung, "Capabilities of outlier detection schemes in large datasets, framework and methodologies," *Knowledge and Information Systems*, vol. 11, no. 1, pp. 45–84, dec 2006. [Online]. Available: http://link.springer.com/10.1007/s10115-005-0233-6

[26] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "Loci: Fast outlier detection using the local correlation integral," in *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*. IEEE, 2003, pp. 315–326.

[27] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Loop: local outlier probabilities," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1649–1652.

[28] L. Zhang, J. Lin, and R. Karim, "Adaptive kernel density-based anomaly detection for nonlinear systems," *Knowledge-Based Systems*, vol. 139, pp. 50–63, 2018.

[29] C. Eiras-Franco, D. Martínez-Rego, B. Guijarro-Berdiñas, A. Alonso-Betanzos, and A. Bahamonde, "Large scale anomaly detection in mixed numerical and categorical input spaces," *Information Sciences*, vol. 487, pp. 115–127, 2019.

[30] C. Eiras-Franco, B. Guijarro-Berdiñas, A. Alonso-Betanzos, and A. Bahamonde, "A scalable decision-tree-based method to explain interactions in dyadic data," *Decision Support Systems*, vol. 127, p. 113141, 2019.

[31] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[32] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*. Springer, 2019.

[33] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in neural information processing systems*, 2015, pp. 2962–2970.

[34] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Practical automated machine learning for the automl challenge 2018," in *International Workshop on Automatic Machine Learning at ICML*, 2018, pp. 1189–1232.

[35] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[36] S. Baluja and M. Covell, "Audio fingerprinting: Combining computer vision & data stream processing," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 2. IEEE, 2007, pp. II–213.

[37] D. Dutta, R. Guha, P. C. Jurs, and T. Chen, "Scalable partitioning and exploration of chemical spaces using geometric hashing," *Journal of chemical information and modeling*, vol. 46, no. 1, pp. 321–333, 2006.

[38] D. Ravichandran, P. Pantel, and E. Hovy, "Randomized algorithms and nlp: Using locality sensitive hash functions for high speed noun clustering," in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, 2005, pp. 622–629.

[39] J. Buhler and M. Tompa, "Finding motifs using random projections," *Journal of computational biology*, vol. 9, no. 2, pp. 225–242, 2002.

[40] T. Haveliwala, A. Gionis, and P. Indyk, "Scalable techniques for clustering the web," 2000.

[41] C. C. Aggarwal, "An introduction to outlier analysis," in *Outlier analysis*. Springer, 2017, pp. 1–34.

[42] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, p. 3, 2012.

[43] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[44] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[45] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*. PMLR, 2018, pp. 4393–4402.

[46] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," *arXiv preprint arXiv:1906.02694*, 2019.

[47] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM, 2014, p. 4.

[48] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, 2015.

[49] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 665–674.

[50] N. Japkowicz, "Concept-learning in the absence of counter-examples: an autoassociation-based approach to classification," 1999.

[51] Y. Wang, S. Parthasarathy, and S. Tatikonda, "Locality Sensitive Outlier Detection: A ranking driven approach," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, apr 2011, pp. 410–421. [Online]. Available: http://ieeexplore.ieee.org/document/5767852/

[52] M. R. Pillutla, N. Raval, P. Bansal, K. Srinathan, and C. Jawahar, "Lsh based outlier detection and its application in distributed setting," in