Information Sciences 607 (2022) 1245–1264

Contents lists available at ScienceDirect

Information Sciences

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

Fast anomaly detection with locality-sensitive hashing and hyperparameter autotuning

Jorge Meira [a,b,*], Carlos Eiras-Franco [a], Verónica Bolón-Canedo [a], Goreti Marreiros [b], Amparo Alonso-Betanzos [a]

[a] CITIC, Universidade da Coruña, A Coruña 15071, Spain
[b] GECAD, Institute of Engineering Polytechnic of Porto (ISEP/IPP), Portugal

ARTICLE INFO

Article history:
Received 8 July 2021
Received in revised form 3 June 2022
Accepted 10 June 2022
Available online 16 June 2022

Keyword:
Anomaly detection
Unsupervised learning
AutoML
Scalability
Big data

ABSTRACT

This paper presents LSHAD, an anomaly detection (AD) method based on Locality Sensitive Hashing (LSH), capable of dealing with large-scale datasets. 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 so that users do not have to implement costly manual tuning. Our LSHAD method is novel as both hyperparameter automation and distributed properties are not usual in AD techniques. Our results for experiments with LSHAD across a variety of datasets point to state-of-the-art AD performance while handling much larger datasets than state-of-the-art alternatives. In addition, evaluation results for the tradeoff between AD performance and scalability show that our method offers significant advantages over competing methods.

1. Introduction

Anomaly detection (AD) problems are present in numerous domains and research fields, including industrial machinery failure [1,2], network intrusion [3,4], credit card fraud [5,6], medicine and public health [7,8], and image processing [9,10], among others [11,12]. Anomalies are events that differ sufficiently from most of the data to indicate that they have been generated from a different process. Their minority nature is problematic, as this hinders the use of supervised machine learning (ML) methods, given that it is difficult to find or build data with these labeled events. As a solution, unsupervised techniques can be used that can be trained to model normal data on unlabeled data, thereby enabling patterns that deviate from the normal to be detected.

The literature records a wide variety of AD methods that can be categorized according to approach [13]. Proximity-based algorithms detect anomalies by measuring their proximity to normal data points, such that elements distant from all others can be regarded as anomalies. This category includes distance-based methods, which rank elements according to their distance from neighbors, and density-based methods, which compare the density around a data point with that of local neighbors. Our proposed method is a density-based method. With density-based methods, the working assumption is that points located in low-density regions have a high probability of being anomalies: the density around a normal point is similar to the density around its neighbors, but is considerably different from the density around an anomaly [14]. Our method (described

* Corresponding author.
E-mail address: j.a.meira@udc.es (J. Meira).

https://doi.org/10.1016/j.ins.2022.06.035
0020-0255/

in detail in Section 4) corresponds in this category since its main characteristic is to randomly split data into different density groups and then analyze the density of each data point so as to infer an anomaly score. Several density-based anomaly detection methods have been described in the literature, including Local Outlier Factor (LOF) [15], and some of its variations [16,17], Local Outlier Correlation Integral (LOCI) [18] and Local Outlier Probability (Loop) [19].

AD models are becoming increasingly popular, partially due to increasingly large datasets in a Big Data context, and unlabeled data are increasingly common, mainly because sources vary greatly, e.g., connected devices such as cell phones, fleets of vehicles, or industrial machinery; anomalies, for instance, could derive from a machine on the verge of malfunctioning, or a vehicle that has experienced unusual environmental conditions. AD for large quantities of data is a difficult task, as it requires considerable computational resources. One solution is the development and application of distributed AD methods.

When dealing with large datasets, a distributed paradigm allowing for parallel computation distributes data across different nodes, with each node operating on the data in parallel. The immutable nature of distributed operations, such as in the Apache Spark framework, helps ensure consistencies in computations. To exemplify, assume that we have a task such as summing all n elements of a given dataset, and the time for a single operation is t units. In the case of sequential execution by a single processor, the summation time required will be $n * t$, but if execution is by 4 processors, time would be reduced to $(n/4) * t$ plus merging overhead in time units. Scalability is becoming a must for this type of task, although at present only a few algorithms are able to cope with large datasets [20,21].

Another field that has emerged in recent years is Automated ML (AutoML) [22]. Almost every ML method has hyperparameters, and thus a key task is optimization of these hyperparameters so as to maximize algorithm performance. AutoML automatically sets these hyperparameters to optimize performance, thereby reducing human effort and obtaining a more rapid and simple solution.

We describe a novel density-based method for AD, called LSHAD, based on the Locality Sensitive Hashing (LSH) technique. LSHAD was developed to address the above-described problems regarding the difficulty of processing very large datasets composed of data generated daily, and the lack of unsupervised methods for AD problems capable of automatically adjusting hyperparameters. Therefore, the main contributions of our work are the following:

- Adaptation of the LSH technique to AD in large datasets.
- Autotuning of hyperparameters. ML success heavily relies on humans to select appropriate hyperparameters, a very complex and time-consuming task that becomes even more critical when it has to be carried out by ML non-experts rather than experts, which happens quite often. There is therefore a great need for AutoML methods [22].
- A distributed algorithm, since development is in the Apache Spark framework using the MapReduce approach for distributed environments

Our method is rapid and effective when the objective is to process large quantities of data in search of anomalies. It achieves a similar (in some cases better) performance in AD compared to other methods. It also has the advantages over alternative methods that it is rapidly configured, as there is no need to tune hyperparameters, and can handle large datasets.

The rest of this paper is organized as follows: Section 2 describes the LSH technique developed by Indyk and Motwani [23] and applied to our algorithm; Section 3 reviews state-of-art methods used for AD; Section 4 explains LSH detailed functionalities and describes 4 different types of estimators. Section 5 describes our LSHAD algorithm, explains the automatic hyperparameter tuning process, and describes an experiment to identify the best estimator. Section 6 evaluates our method and compares it to other algorithms in terms of AD and execution time. Finally, Section 7 summarizes our main conclusions and describes ideas for future work.

2. Background

The basic concept underlying LSH, introduced by Indyk and Motwani [23], is to identify approximate nearest neighbors through the use of hash functions. The underlying principle that two points in the feature space that are close to each other are very likely to have the same hash function. LSH is formally defined by Indyk and Motwani [23] as follows:

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

- if $\|p - q\| \leqslant r_1$then $P_{\mathscr{H}}[h(q) = h(p)] \geqslant P_1$,
- if $\|p - q\| \geqslant r_2$ then $P_{\mathscr{H}}[h(q) = h(p)] \leqslant P_2$.

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

Generated from $\mathscr{H}$ is a h hash function by the concatenation of various L random projections (a user-specified parameter explained in Section 4), $h =< proj_1, proj_2, \ldots, proj_L >$. 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 make the hashes more specific by increasing L. However, if hashes are very specific, many points may end up in different buckets from their neighbors. Therefore, T hashes are generated for each point. The impact of L and T on algorithm performance is studied in Section 5. LSH speeds up the search for neighbors in requiring much less computational effort than the brute-force approach of measuring every possible pairwise distance. LSH and variants have already been successfully applied in practical scenarios such as computer vision [24], recommender systems [25], and linguistics [26].

In implementing the LSH technique in our AD algorithm, the goal is to rapidly retrieve neighbor counts to be used as a ranking score for AD. We assume that points with few neighbors are very likely to be anomalous. An advantage of using this technique is that it rapidly processes data in high-dimensional spaces, which, when combined with distributed implementation in Apache Spark, makes our LSHAD algorithm highly scalable.

3. Related Work

Below we describe existing work related to AD and outlier detection algorithms, with very similar definitions [27]. Firstly, we describe frequently used and recent general methods, before then focusing on density-based methods, and lastly some LSH variants.

Liu et al. [28] developed their Isolation Forest (IForest) algorithm that works with binary trees. Each tree is created by partitioning instances recursively and randomly selecting a split value for a specific attribute. Tree path length is used as an anomaly score, with data points with shorter path lengths considered anomalies.

One-Class Support Vector Machine (SVM) [29], a variant of the classical SVM algorithm, is another method that can be applied to AD problems. It relies on finding the smallest hypersphere containing all training examples after mapping by a kernel function. Different approaches to fitting a SVM model are training with data from different classes, training with data from unknown classes, and training with data from a single class. In the One-Class SVM method, all the data in the training set are represented by only one class. In AD problems the method is usually used to train data belonging to the non-anomalous class, as these data are commonly available. The algorithm separates all data points from the origin and maximizes the distance from the hypersphere to the origin, resulting in a binary function that captures regions in the input space where the data density probability is high [29].

Martínez-Rego et al. [30] proposed a modification of the One-class classification with passive-aggressive Kernel algorithm (PA-I) [54] combining it with a Bernoulli CUSUM chart to deal with stream change problems. With this adaptation the method is capable of accurately fitting the support of normal data in an online fashion. Thus, it can dynamically adapt to changes in data distribution.

Deep learning is still a hot topic, with numerous applications and approaches described in the literature in fields such as computer vision [31], speech recognition [32], natural language processing [33,34], etc [35,36]. Deep-learning methods are also widely used in AD problems [37], especially the autoencoder architecture [38]. This method is trained in order to make output features the same or very similar to input features [39]. Autoencoders are composed of two parts: the encoding layer(s) compress(es) the input into a latent-space representation, and the decoding layer(s) reconstruct(s) the output from this representation. The anomaly ranking score is computed from the reconstruction error metric, which measures the difference between input and output data.

Eiras-Franco et al. [20] recently proposed the Anomaly Detector for Mixed Numerical and Categorical Inputs (ADMNC) algorithm, which, as the name indicates, targets data with both categorical and numerical variables. The model is trained through a maximum-likelihood objective function optimized with stochastic gradient descent. It is capable of dealing with large quantities of data, since implemented in Apache Spark, the algorithm lends itself well to parallel computation.

Concerning density-based methods, Breunig et al. [15] proposed the LOF algorithm, which searches for anomalous data points by measuring the local deviation of a given point from its neighbors. The same concept inspired other developments, such as LOCI [18], which aims at fast outlier detection using the local correlation integral. This improved method can identify not only outliers but also groups of outliers, providing an automatic cutoff to determine whether or not a point is an outlier. Its main drawback is its quadratic complexity, which makes it computationally expensive, and thus prohibitive for very large datasets.

LSH methods have recently been successfully applied to AD problems. Wang et al. [40] proposed an LSH framework for ranking points according to the likelihood that they are anomalous. The data is first split in clusters and then a ranking of points is computed by building LSH tables. Each point is next evaluated according to its rank to isolate a certain number of anomalies. This ranking mechanism is based on the number of points hashed to the same bucket on the assumption that points in buckets with few elements are likely to be anomalies. The authors reported that they could isolate the top anomalies very quickly, usually by scanning less than 3% of the dataset, and in their empirical study their method outperformed other AD methods, although the comparison was with just 2 other methods.

Pillutla et al. [41] presented an approach in which LSH is used to prune non-outlier data points according to their redundancy in a hash table. The algorithm then processes the data using the pruned points, which makes this approach computationally less costly. The authors developed a distributed system for their algorithm and evaluated their method in terms of AD and communication time, but did not compare their method with other algorithms.

Zhang et al. [42] proposed a density-biased sampling approach using LSH to count neighbors and obtain a scalable density estimate. They also proposed a parameter tuning rule, specific to AD for LSH. They formally investigated density-biased sampling for AD, suggesting that, given the different importance of data points according to density, this approach to sampling would have a higher impact on AD performance compared to uniform sampling, and conducting an empirical study to compare the approaches.

The works by Wang et al.[40], Pillutla et al.[41], and Zhang et al.[42] described in this section use LSH techniques for AD problems. We identified the following differences with our method:

- Although the results reported by Wang et al.[40] showed that their method is more scalable than others included in their study, they did not mention whether their method is capable of performing distributed computing (as was the case for Pillutla et al.[41]). Implementation of our method in Apache Spark enables distributed data processing across various processor cores, and thereby enabling larger datasets to be handled than handled by competitors.
- Our method adjusts hyperparameters automatically, relieving the user of this time-consuming task and contributing to the AutoML field.
- Our experimental study (described in detail below) is much broader, as we compare our method across a wide range of datasets and with different AD methods.

Table 1 summarizes the different methods, considering hyperparameter autotuning and distributed computing capabilities.

4. Proposed Method

Before we describe the use of the LSH technique for AD, we explain the automatic hyperparameter tuning mechanism implemented in our method and the impact of each hyperparameter on the process of generating random projections and creating groups of neighbors. We also describe several density estimators that measure 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 rapidly and inexpensively thanks to distributed computation using the MapReduce approach implemented in Apache Spark [43]. Our proposed method leverages the LSH technique by applying hash functions to group data points in buckets with their neighbors. The number of neighbors in each bucket is then used to compute several evaluation metrics that score and rank elements according to 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 5 (Line 1). Then a hasher, consisting of $L * T$ hyperplanes, is created 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 below which a point is considered an anomaly. The threshold is selected so that the number of elements that fall below it corresponds with the anomaly ratio for the training data, which is provided by the user.

Algorithm 1: Pseudocode for LSHAD. Training phase.

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

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, and assessing whether a test point p is an anomaly follows the process described in Algorithm 2. First, the hashes for p are computed (Line 1), then the estimator values corresponding to the assigned hashes are accumulated. If the resulting value fails to reach the threshold established by the learned model, then p is 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. An estimator is calculated using the precomputed counts in the model, which represent the properties of the training data distribution.

Table 1
Characteristics of different anomaly detection algorithms

Methods	Auto-Hyperparameter	Distributed
One Class SVM	Yes[1]	No
LOF	Yes[1]	No
LOCI	No	No
Pillutla et al. [41] method	No	No
Wang et al. [40]	No	No
Zhang et al. [42]	No	No
PA-I	No	No
Autoencoder	No	Yes
IForest	No	No
ADMNC	No	Yes
LSHAD	Yes	Yes

[1] It has several hyperparameters, with only one tuned automatically

Algorithm 2: Pseudocode for LSHAD. Detection phase.

Input : $p \leftarrow$ Test point,
$hasher \leftarrow$ Hasher of the trained model,
$estPerHash \leftarrow$ Estimator dictionary,
$threshold \leftarrow$ Estimator threshold

Output: Boolean value indicating whether p is an anomaly

$hashes \leftarrow hasher.\text{HASH}(p)$;
$estimator \leftarrow 0$;
foreach $h \in hashes$ **do**
  $estimator \leftarrow estimator + estPerHash[h]$;
end
Result: $estimator < threshold$

4.1. Hashing

Although the LSH techniques that we use can draw on many LSH hash families, since our implementation is based on the Euclidean distance, we selected the corresponding classical hash function, formally computed as follows:

$$proj(\boldsymbol{x}|\alpha,\beta) = \left\lfloor \frac{\boldsymbol{x}\cdot\alpha+\beta}{w} \right\rfloor \tag{1}$$

The projection $proj(x|\alpha,\beta) : \mathbb{R}^d \rightarrow \mathbb{Z}$maps a d dimensional vector $\boldsymbol{x}$, representing each data point, onto the set of integers, where α is a random vector drawn from a Gaussian distribution, and where β is a real number uniformly chosen from the interval $[0 : w]$. This scalar projection is then quantized into a set of hash buckets, grouping all elements that are close together in the original space in the same bucket. The user-specified hyperparameter w in Eq. 1 represents the resolution of the quantization. Fig. 1 shows one such hash function, consisting of a random projection in 2 dimensions with a specific w value.

A hash function, represented by such L random projections, defines the hash value (Eq. 2):

$$H(x) = < proj_1(x|\alpha_1,\beta_1), \ldots, proj_L(x|\alpha_L,\beta_L) > \tag{2}$$

Where $proj_i(x|\alpha_i,\beta_i), 1 \leqslant i \leqslant L$(from Eq. 2) is computed by Eq. 1. After all the hash functions are generated, observations with the same hash values are grouped together.

Using the same notation as used in Definition 1 in Section 2, in order for a family $\mathscr{H}$ to be useful it has to satisfy the condition that the probability of P_1 is much higher than that of P_2. Hash functions $H(x)$ will, in some cases, not fulfill this condition, especially as they are generated at random. To ensure that $P_1 > P_2$ while taking into account the probabilistic

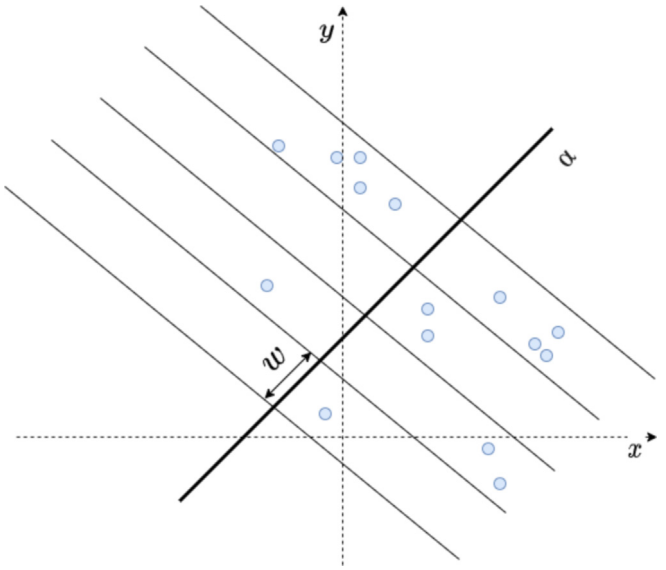

Fig. 1. Random projection in 2 dimensions. Axis $x, y \in \mathbb{Q}$and the blue dots are composed by random values from $\mathbb{Q}$.

properties of $H(x), T$ hash tables are created, each one indicating the hash of each data point. As a result, each point $\boldsymbol{x}$ will receive a set of hashes $\{H_1(x), H_2(x) \ldots H_T(x)\}$. When grouping elements with the same hash, the method creates groups of elements that have a high probability of being close together. However, increasing the values of parameters L and T also increases the computational complexity of the algorithm, since more hashes need to be generated. It is therefore necessary to identify suitable values for these parameters that trade off accurate AD against as little computational effort as possible.

Fig. 2 shows an example of a hash table $H(x)$, with data points on the left and the hash table on the right. The rows represents different hash values and the righthand column shows the collisions, which occur when data points share the same hash value.

4.2. Anomaly level estimation

Once a suitable hasher has been found, the next step 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 anomalous. We explored 4 different density estimators, as follows. Let D be the input dataset and let $b_h = \{x \in D, H(x) = h\}$ be the set of points with hash h in one of the tables t. We define the 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:

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

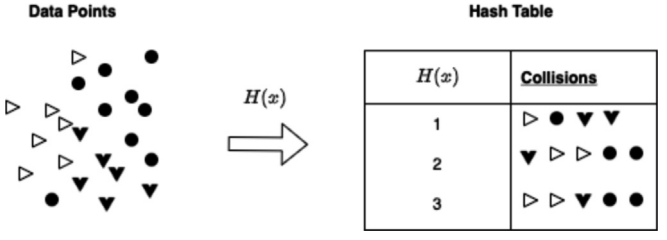

Fig. 2. Hash table example.

- 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:

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

In Section 5 we analyze the 4 estimators to determine which one gives the best anomaly ranking score.

4.3. LSHAD framework

LSHAD is implemented in the Apache Spark framework, designed for fast performance using RAM for caching and MapReduce for processing data. Parallel computation is enabled by the use of resilient distributed datasets (RDDs), an immutable partitioned collection of records with partitions that can be operated in parallel. Even though RDDs are immutable, they can be transformed into other RDDs using functions such as mapping, filtering, joining, groupBy, etc. The immutability ensures consistent computations since any changes in RDDs are permanent; the fact that data can be safely shared across various processes and threads enhances the computation process by caching RDD. Fig. 3 shows how LSHAD makes use of RDDs to compute tasks in parallel.

First, LSHAD splits data into train and test sets, and each set is transformed into an RDD in which data is partitioned according to a user-defined number of nodes/partitions that allowing task to run in parallel. In the training phase, LSHAD adjusts its hyperparameters in 2 iterative steps performed in parallel in the multiple partitions, namely, creating hashes, and retrieving specific measurements to search for the w size values that build optimal hash tables. Section 5 describes this process in detail.

In the testing phase, LSHAD calculates new hashes for each test point in each partition. It then accumulates the estimator values corresponding to the hashes assigned to all evaluated points. Finally, all estimator values are collected and compared to a threshold value to determine whether an observation is normal or anomalous.

5. Hyperparameter Tuning and Experimentation

AutoML, has been a hot research topic in recent years [22], as applying traditional ML methods is time-consuming, resource-intensive, and challenging. Manual hyperparameter tuning is challenging, as besides being very computationally expensive, hyperparameter tuning has a great influence on the final algorithm results.

Given the possible difficulties faced by a non-expert user in tuning hyperparameters when only unlabeled data is available, for LSHAD, we implemented automatic hyperparameter tuning, studying the behavior of each user-specified parameter, that is, the resolution of quantization buckets w, the number of random projections L, and the number of tables T.

5.1. Datasets

To analyze the hyperparameters and evaluate various AD methods, we selected several datasets widely used in the literature for classification tasks, but adapted for AD tasks. The datasets, presented in Table 2, were downloaded from the UCI Machine Learning [44], Zenodo[1] and Stratosphere Research Laboratory[2] Repository.

Regarding the UCI Machine Learning Repository datasets, we used a version of Abalone, which contains data on abalone shell characteristics that predict its age (number of rings of a cut shell). The idea is to observe whether an algorithm can identify differences in specific age ranges. Thus, for Ab. 1–8, Ab. 9–11, and Ab.11–29, classes considered anomalous are 1–8, 9–11, and 11–29, respectively, while all other classes are considered non-anomalous

Also used was a sample of 20% of the CoverType dataset, composed of cartographic variables that classify different types of forest cover. In this dataset, class 2 instances (Lodgepole Pine) were considered normal, while class 4 instances (Cottonwood/Willow) were considered anomalous.

Other datasets selected from the same repository were German Credit, Arrhythmia, Pima Diabetes, Breast Cancer, Heart, three versions of the KDDCup99 dataset, and finally, IDS 2012, an update of the KDDcup99 that solves some of its problems. For the German Credit dataset, representing people receiving bank loans classified as good or bad credit risks according to specific attributes, we considered bad credit risk as the anomalous class. For the Heart dataset, we considered patients with disease to be an anomalous class. Regarding KDDCup99, a well-known intrusion detection dataset, we used the following versions: KDD99 (a sample of the full KDDCup99 with all cyberattacks), KDD99-SMTP (reduced KDDCup99 filtering only

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

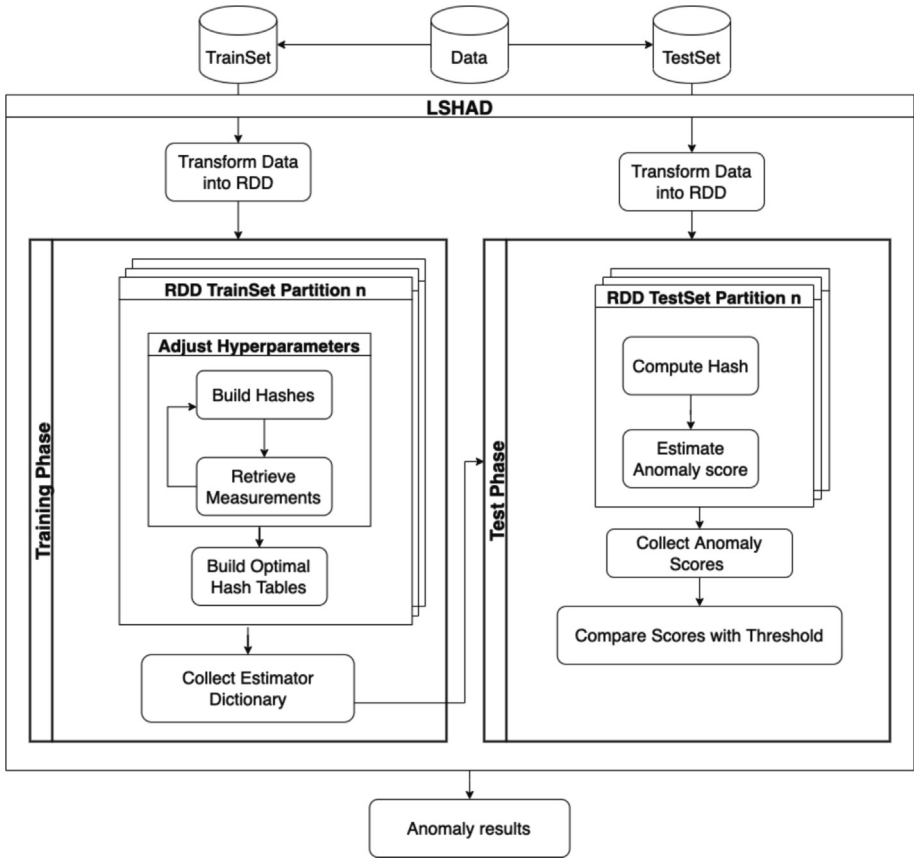

Fig. 3. LSHAD diagram.

smtp connections), and KDD99-HTTP (reduced KDDCup99 filtering only http connections). For the KDDCup versions and the IDS 2012 datasets, we considered any sort of intrusion as anomalous.

From the Zenodo repository we took a synthetic dataset of two-dimensional combinations of attributes of clusters of different shapes (see Fig. 4).

From the Stratosphere Research Laboratory repository we took a newly released dataset called Aposemat IoT-23, containing malicious and benign network traffic for real IoT devices, for which we consider attacks as anomalous.

To analyze hyperparameter behavior and test the estimators we used 8 real datasets, namely, Ab.1–8, Arrhythmia, German Credit, CoverType, all KDDCup99 versions, and IDS 2012, and to evaluate and compare the AD methods we used all datasets, as described in Section 6.

5.2. Hyperparameter analysis

To analyze the behavior of each hyperparameter and its impact on the performance of our method, we conducted an experimental study to identify regular patterns or correlations between the performance of the LSHAD algorithm and hyperparameter values in order to build an automatic tuning mechanism.

For all the experiments we used fivefold cross-validation, computing the average for each metric used. For this particular study, the datasets were modified in order to retain 1 % of anomalies. This was done to both provide an accurate anomaly rate to the algorithm, and to ensure that normality is learned by keeping the number of anomalies low. This would not be possible in real-life as the user would have verify that the training dataset represented normality and would need to provide an estimate of the anomaly rate for the training dataset.

We first fixed a constant value for the parameter w, and tested the performance of the algorithm using the metric area under the curve (AUC) for one of the estimators for different values of L and T.

Analysing the results, it was observed that by increasing the number of hash tables (T), LSHAD performance remained very similar irrespective of the number of random projections, L, to be generated. We tested values from 1 to 128 for L for each different T value; for visualization purposes, Fig. 5 shows a plot of LSHAD AUC performance measured for 2 random projections, $L = 2$, and for different hash table values T (from 5 to 1,000). As can be observed, LSHAD performance improves

Table 2
Datasets used to analyze hyperparameter tuning and anomaly detection evaluation.

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

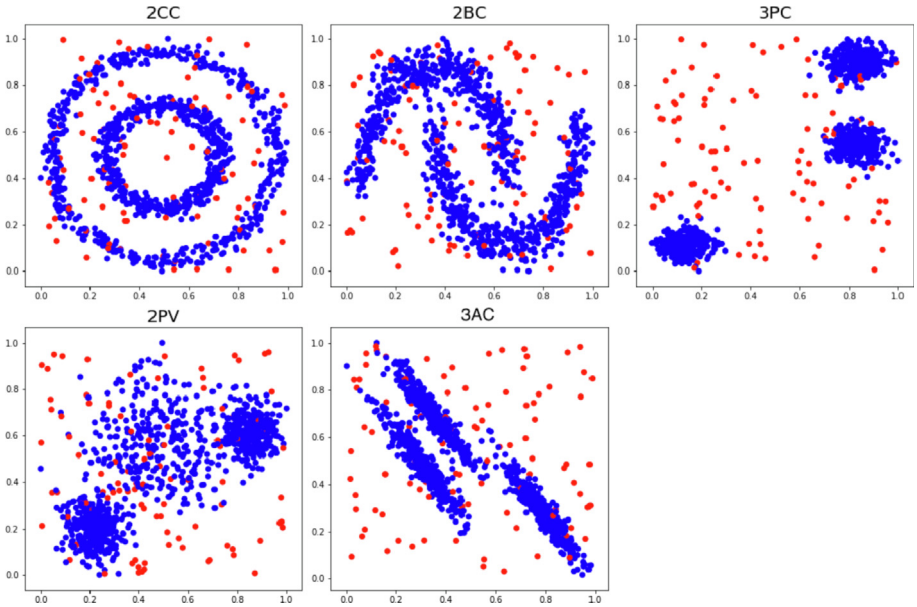

Fig. 4. Synthetic dataset of shapes representing 2 circular clusters (2CC), 2 banana clusters (2BC), 3 point clouds (3PC), 2 point clouds with variance (2PV), and 3 anisotropic clusters (3AC).

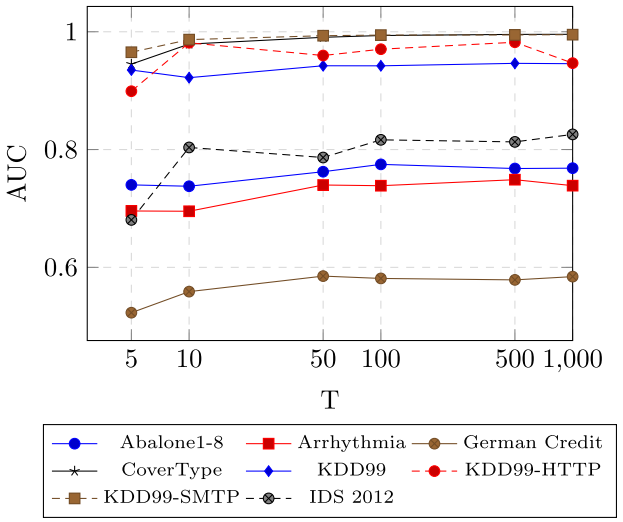

Fig. 5. LSHAD performance (AUC) varying the hyperparameter T, the number of hash tables.

as the number of hash table increases, until stagnating at a particular AUC value. This behavior was expected, as mentioned in Section 2, as repeating this random process several times will increase the likelihood that 2 similar data points will collide in the same bucket. As can be seen in Fig. 5, for most datasets the LSHAD performance approached optimum at $T = 50$; for higher values, performance was maintained or slightly improved, while in some cases, the repetition resulting from a high number of hash tables deteriorated performance.

Fig. 6 depicts the model AUC versus different values of L (for a fixed T = 50), showing that different L values have a small impact on LSHAD performance for the CoverType dataset[3] and all the KDD99 dataset versions. Performance deteriorated greatly for Abalone 1–8 with more than 8 random projections and for Arrhythmia, with more than 32 random projections, and improved greatly for IDS 2012 with more than 16 random projections. No conclusions could be drawn regarding the effect of L in the German Credit dataset; therefore, we fixed the parameter $L = 4$, as an acceptable value to trade off performance against computational cost. This is because shorter hashes require less memory in saving the model and so can be processed faster.

In the next experimental step, for the same test approach, we fixed the values of both the T and L parameters. We set $T = 50$, because, as observed from Fig. 5, performance improvement is not significant beyond that value, and we set $L = 4$ as the optimal tradeoff value described above.

Using these fixed values, we analyzed the effect of w, the length of the quantization buckets, with Fig. 7 showing that w has a great impact on AD accuracy, although its optimal value 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, which is merely a matter of finding a suitable w value for the given dataset.

For an unlabeled dataset, however, the effect of w on AD detection accuracy cannot be directly observed, since no labels are available to measure performance. To obtain more information, we thus extracted other indirect unsupervised metrics to observe if they were correlated with algorithm performance:

- Bucket count (BC): number of buckets generated.
- Average bucket size (ABS): Let $|D|$ be the cardinality of the input data and let b be each bucket size from B buckets generated. Hence:

$$ABS = \frac{\sum_{b \in B} b}{\frac{BC}{|D|}} \tag{7}$$

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

To assess the suitability of these metrics, we explored several w values for the datasets and plotted each metric versus the AUC. While no pattern was observed for the BC and ABD metrics, the ABS was found to contain useful information. For most of the tested datasets, LSHAD performance was much improved when ABS was in the range $[0.05, 0.1]$, as can be observed in Fig. 8. ABS can therefore be used to obtain 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 AD 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.

[3] Performance is similar to that for KDD99-SMTP but this is not visible in Fig. 6 since the corresponding line is behind the KDD99-SMTP line.

5.3. LSHAD with hyperparameter autotuning

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.

A search interval must first be set, for which the lower threshold is always set to 1 (line 2). The upper threshold is found by doubling the w value, and using it for hashing until small enough buckets result (line 3). That range is then explored using a binary search (loop on line 7) to find a value for 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: Pseudocode for LSHAD:. Hyperparameter training.

Input : $D \leftarrow$ Set of training points

Output: $L, T, w \leftarrow$ tuned hyperparameters

1 $L \leftarrow 4$, $T \leftarrow 50$;

2 $wCandidate \leftarrow 1$, $avBucketSize \leftarrow 0$, $leftLimit \leftarrow 1$, $rightLimit \leftarrow 1$;

3 **while** $avBucketSize < 0.05$ **do**

4 $\quad avBucketSize \leftarrow$ HASHGROUPANDCOUNT$(D, L, T, wCandidate)$;

5 $\quad wCandidate \leftarrow wCandidate * 2$;

end

6 $rightLimit \leftarrow wCandidate$;

7 **while** $avBucketSize < 0.05$ *or* $avBucketSize > 0.1$ **do**

8 $\quad wCandidate \leftarrow \lfloor (leftLimit + rightLimit)/2 \rfloor$;

9 $\quad avBucketSize \leftarrow$ HASHGROUPANDCOUNT$(D, L, T, wCandidate)$;

10 $\quad$ **if** $avBucketSize < 0.05$ **then**

11 $\quad\quad leftLimit \leftarrow wCandidate$;

$\quad$ **end**

12 $\quad$ **if** $avBucketSize > 0.1$ **then**

13 $\quad\quad rightLimit \leftarrow wCandidate$;

$\quad$ **end**

14 $\quad$ **if** $leftLimit >= rightLimit$ **then**

15 $\quad\quad$ break;

$\quad$ **end**

end

Result: $L, T, wCandidate$

5.4. Estimator experiments

Regarding the proposed estimators in Section 4, namely, $E_A(h), E_B(h), E_C(h), E_D(h)$, we compared their AUC performance for each given dataset to determine which produced the best ranking score for classification. Fig. 9 depicts a graph showing the AUC score for each estimator, showing that they all produced similar results with small variations in the AUC for different

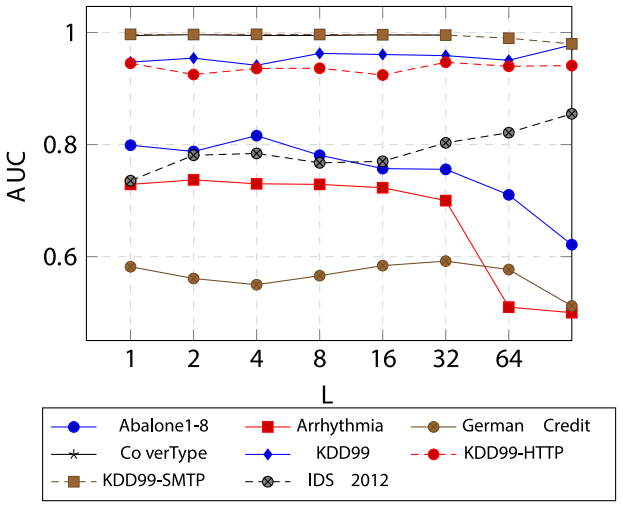

Fig. 6. LSHAD performance changing the hyperparameter L, the number of random projections.

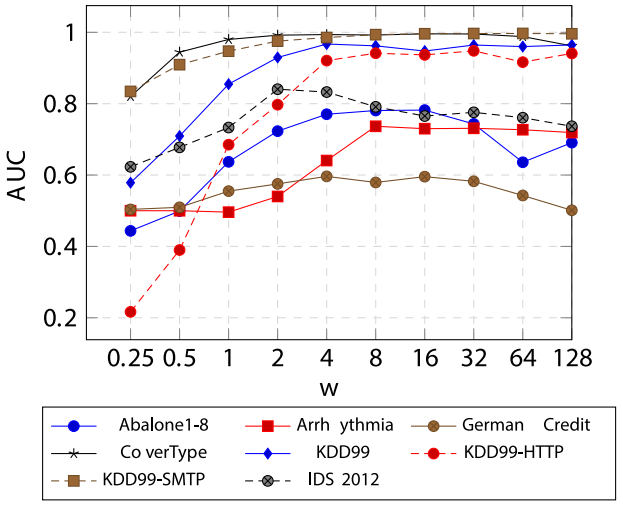

Fig. 7. LSHAD performance changing the hyperparameter w, the quantization bucket length.

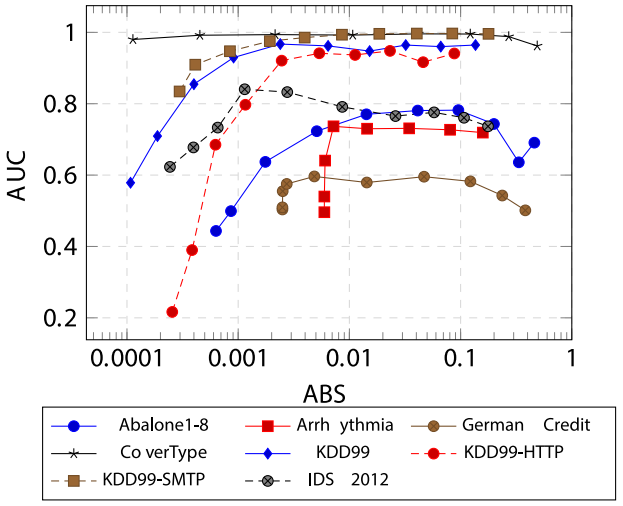

Fig. 8. LSHAD performance for different w values, with the ABS metric on the horizontal axis.

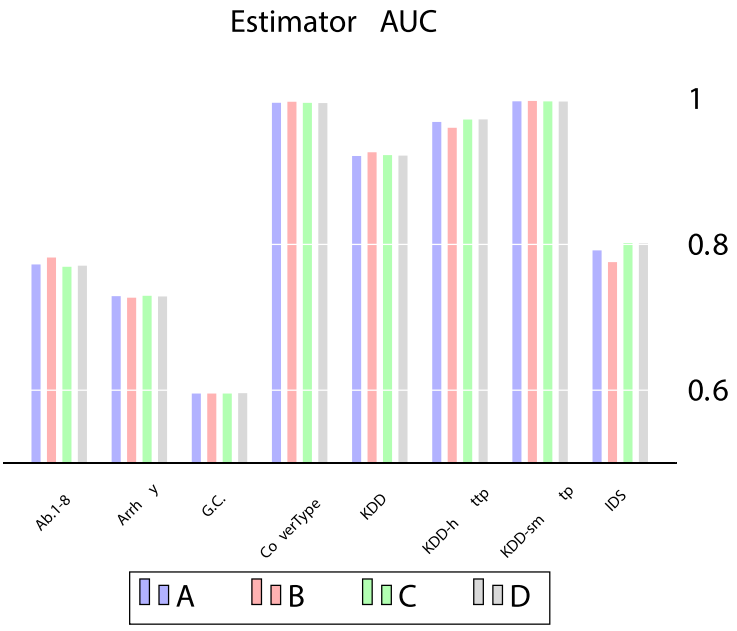

Fig. 9. AUC scores for the different estimators.

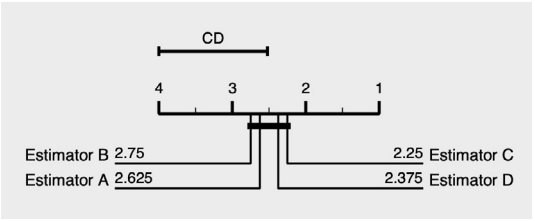

Fig. 10. Nemenyi statistical test for the estimator AUC scores.

datasets. In fact, the statistical Nemenyi post hoc test [45] with $\alpha = 0.05$ could not significantly differentiate the estimator scores (see Fig. 10). We chose our algorithm to use the C estimator by default as it was the estimator with the lowest critical difference value.

6. Performance Evaluation

Below we evaluate the LSHAD algorithm compared to other methods in terms of processing time and AD performance.

6.1. Methods and datasets

To measure and compare the performance of our method against other methods, a variety of state-of-the-art algorithms (many referred to in Section 3) were selected. As LSHAD is a density-based method, we first selected the well-known LOF and LOCI methods, and used Euclidean (E), Jaccard (J) and Hamming (H) distances, for which we employed a Matlab implementation[4]. From the same category we also selected the approach by Zhang et al.[5]. [41], as it also uses LSH for scalable density estimation; in this case we tested their JAVA implementation[6] of 4 algorithms using their piecewise density-biased sampling (PDBS), namely:

- **1 Sample PDBS** (1 Samp PDBS)- drawing one sample for all points to compute the k-NN distance
- **Iterative PDBS)** (Ite PDBS)- drawing one sample for each point to compute the k-NN distance

[4] https://github.com/jeroenjanssens/lof-loci-occ
[5] From the LSH methods presented in Section 3, this is the only algorithm available to test
[6] https://bit.ly/2ugZQ0x

Table 3
Selected algorithm AUC results for 5 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**

- **Iterative + Ensemble PDBS)** (Ite + Ens PDBS)- drawing multiple samples for each point to make ensembles for the k-NN distance
- **Isolation Forest PDBS)** (IForest PDBS)- using the IForest detection method.

Other methods related to unsupervised AD were also selected for our evaluation: the Autoencoder implementation in Python using the Elephas[7] framework, an extension of Keras that allows distributed deep-learning models to be run at scale with Spark; the One-Class SVM with radial basis (SVM-R) and linear (SVM-L) kernel functions, for which we used the Matlab LibSVM interface[8]; a distributed version of SVM that can handle large datasets (DOC-SVM) [46]; an online one-class classifier with a passive-aggressive kernel (PA-I) [30], also built-in Matlab; and finally, ADMNC implemented in Scala-Apache Spark[9].

All the methods were tested with several datasets with different compositions in order to observe algorithm behavior in a variety of scenarios. We first used a simple synthetic dataset[10] suite with just 3 dimensions and 1000 sample representing different shapes as described in Table 2. We also used the real datasets described in Section 5.

In our experiments we performed fivefold cross-validation, filtering around 1% of the class anomaly samples for each dataset with the aim of simulating a real AD scenario, as done in Section 5 to test the different hyperparameters. Note that, to overcome computational difficulties for some methods, we only used 2 folds rather than 5 folds in testing the medium datasets. In addition, as we use the AUC metric for evaluation, we ignore the threshold variable described in Section 5.3 and use the anomaly score provided by the estimator as the LSHAD output. Used for the experiments was a MacBook-Pro laptop with 8 GB of RAM memory and a 2.9 GHz Intel Dual-Core i5 processor.

6.2. AD performance comparison

For visualization purposes we split the tables according to datasets structure. Table 3 shows AUC results for the synthetic datasets, Table 4 shows algorithm performance results for small datasets with fewer than 5000 samples (Abalone, Arrhythmia, German Credit, Heart, Pima Diabetes, Breast Cancer) and finally, Table 5 shows algorithm performance results for medium datasets, with more than 5000 samples (CoverType, KDDCup99 datasets, IDS 2012).

6.2.1. Synthetic datasets

We first made a comparison of the algorithms for a simple classification task, applying 5 datasets of different shapes with 2 dimensions each, represented in Fig. 4. Table 3 shows that our LSHAD method obtains state-of-the-art results in AD for the different data shapes, except for the 2 circular clusters. While LSHAD obtains the best performance for the 2 point clouds with variance, overall the best performance is achieved by the much more exhaustive LOF and LOCI methods.

6.2.2. Real datasets

For the small datasets, from Table 4 it can be seen that AUC scores are very variable. Although LSHAD did not obtain the best score in any dataset, its results are average state-of-the-art, and in some cases close to the best. For the medium datasets, Table 5 reports a similar outcome. Note that LOF and LOCI were excluded from the comparison, as their quadratic com-

[7] http://maxpumperla.com/elephas/
[8] https://www.csie.ntu.edu.tw/ cjlin/libsvm/
[9] https://github.com/eirasf/ADMNC
[10] https://zenodo.org/record/1171077#.XkE-HBP7TOR
[11] http://github.com/eirasf/ADMNC/

Table 4
Selected algorithm AUC results for small real 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 5
Selected algorithm AUC results for medium real datasets.

	CT	KDD99	KDD99h	KDD99s	IDS	IOT-23[1]
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**

[1] **This is a sample of the IoT-23 Subset with ID 1**

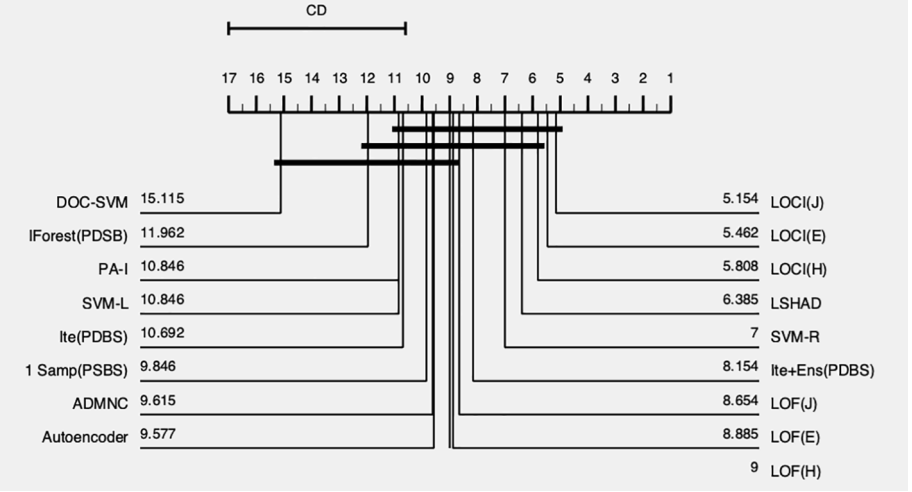

Fig. 11. Nemenyi statistical test to evaluate AUC scores for AD methods.

plexity made them computationally excessively costly in managing large datasets, nor was it possible to test DOC-SVM, as its Matlab implementation failed in trying to split large datasets.

6.2.3. Statistical test evaluation

We ran a statistical Nemenyi post hoc test [45] with $\alpha = 0.05$ to check for any significant statistical difference between methods. In Fig. 11, which shows the algorithms sorted by score, it can be observed that the Nemenyi test divided the algo-

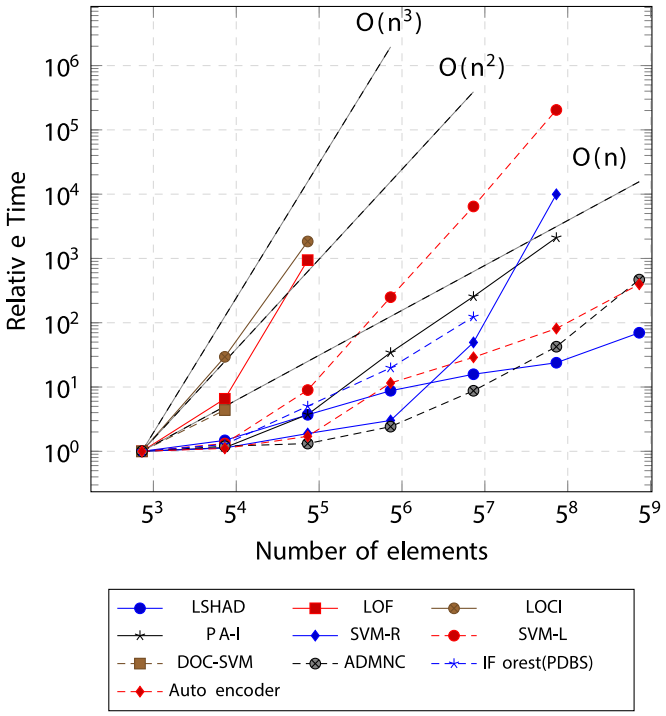

Fig. 12. Execution time of each algorithm increasing the size samples of the Synthetic dataset. Axis are represented using logarithmic scale.

rithms in 3 groups, represented by horizontal thick lines. LSHAD was placed in the group of algorithms with the best performance, for which there is no statistical difference.

6.3. Scalability testing

To test method scalability, we used a synthetic dataset from the generator developed by Eiras-Franco et al. [20] [11]. Varying size, we started with 100 samples and increased the sample 5 times for each iteration. Since the methods are implemented on different platforms, we measured relative algorithm execution time as the ratio between the processing time for the first dataset with 100 samples and the processing time for each other specific dataset size. This allowed us to approximate the empirical time complexity of each method. Selected for this test were the LOF and LOCI methods with Hamming distance, SVM-L, SVM-R, Autoencoder, DOC-SVM, PA-I, ADMNC, and IForest(PDBS) (as the fastest of the 4 PDBS methods). Fig. 12 depicts execution time results of each algorithm, showing that all the algorithms process the data very rapidly for small datasets (100 and 500 samples), except LOCI and LOF (given their quadratic complexity). DOC-SVM was unable to process datasets with more than 2500 samples due to its current implementation, and needed more time to process the small datasets compared to the other

Table 6
AUC results for LSHAD, ADMNC, and Autoencoder for IoT-23 datasets.

ID DATASET	LSHAD	ADMNC	Autoencoder
1	89.60 ±0.87	**91.95 ± 1.87**	62.58 ± 0.0038
3	**99.53 ± 0.12**	95.45 ± 0.61	96.80 ± 0.0011
7	99.94 ± 0.02	99.68 ± 0.43	**99.71 ± 0.00023**
9	**99.97 ± 2.42**	64.99 ± 15.72	99.89 ±8.96$e-9$
17	71.76 ± 21.05	97.22 ± 1.06	**99.99** ±6.79$e-5$
33	76.42 ± 5.08	**83.31 ± 18.26**	51.81 ± 0.017
35	98.48 ± 1.38	**99.84 ± 0.06**	95.21 ± 0.017
36	99.77 ± 0.29	99.36 ± 1.21	**99.99** ±8.99$e-8$
39	97.42 ± 0.21	76.98 ± 2.80	**99.99 ± 4.54e-5**
43	91.29 ± 3.23	**99.99 ± 0.0005**	59.72 ±0.038
48	**99.78 ± 0.19**	99.55 ± 0.78	99.58 ±9.02$e-6$
49	**99.52 ± 0.15**	99.37 ± 0.30	99.27 ±132$e-5$
52	94.19 ± 3.55	99.61 ± 0.57	**99.99** ±1.71$e-7$
60	99.65 ± 0.17	99.80 ± 0.15	**99.99** ±7.18$e-6$
Avg. AUC	**94.09**	93.36	92.82

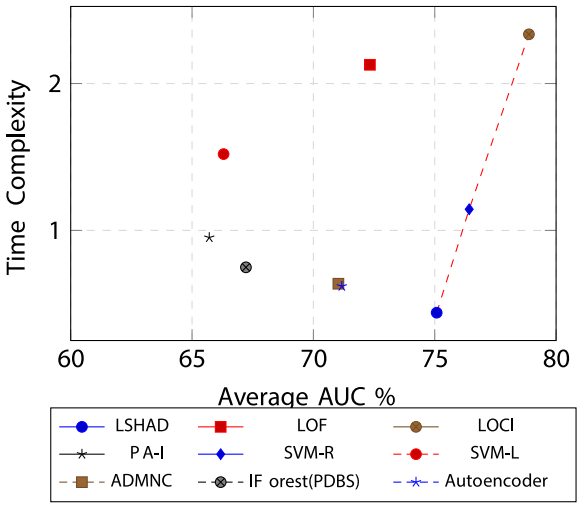

Fig. 13. Pareto front of a multi-objective optimization problem based on mean AD performance for all datasets (higher is better) versus time complexity (smaller is better).

algorithms; PA-I execution time started to increase significantly for datasets with more than 2500 samples, exceeding linear complexity; SVM-L exhibits quadratic complexity; IForest(PDBS), although showing acceptable execution times for small datasets, could not handle datasets of more than 62500 samples; and SVM-R performed adequately up to 12500 samples, then slowed down considerably, exceeding quadratic complexity.

For the larger datasets, ADMNC, Autoencoder, and LSHAD achieved the best execution times, while LSHAD showed the lowest complexity when handling the largest amount of data (1562500 samples).

An experiment was also carried out with the IoT-23 dataset since it has some large subsets in the order of 7 GB, rounding 70,000,000 records [47]. Only the LHSAD, ADMNC, and Autoencoder algorithms were used, given the evidence that they could deal with large datasets, given their distributed approach. The algorithms were applied to each IoT-23 dataset subset and fivefold cross-validation was performed. The resources of the Centre of Supercomputing of Galicia (CESGA) [44] were used, consisting of 22 machines with 35 GB of RAM and 22 cores each.

Table 6 shows that the overall average AUC for LSHAD was slightly better than for ADMNC and Autoencoder. However, the 3 algorithms outperformed each other in specific scenarios. Results were similar, at around 99% AUC, for subsets 3, 7, 35, 36, 48, 49, 52, and 60, while differences occurred with the remaining subsets: for subsets 1, 33, and 43: LSHAD and ADMNC outperformed Autoencoder, for subset 9 and 39, LSHAD and Autoencoder outperformed ADMNC; and for subset 17, ADMNC and Autoencoder outperformed LSHAD. LSHAD therefore produced similar or better results than ADMNC or Autoencoder for all subsets except subset 17.

While LSHAD achieved the best average AUC, slightly better (1%) than its competitors, overall the three methods did an excellent AD job for this dataset. Autoencoder had the lowest average AUC, but only performed poorly with 3 datasets (1, 33, 43); its higher standard deviation on those datasets indicates difficulty in adjusting the parameters. While the reasons are difficult to ascertain, due to the lack of transparency and interpretation of this method (it operates like a black box), we can deduce possible cause. First, dataset 33 is unbalanced, as only 2.54% represents benign data. This quantity of normal activity may not be sufficiently representative, causing the Autoencoder to generate noise when reconstructing its input. Moreover, for dataset 33 (Kenjiro attack type capture), data distribution may be noisy, as performance of both LHSAD and ADMNC with this dataset was also poorer relative to their results for the other datasets. Second, while datasets 1 and 43 have balanced classes, the problem may lie in a loss of important information in the compression phase, as autoencoders are lossy [48] in the degradation that occurs in compression. The density-based methods using the hashing (LSHAD) and Gaussian mixture model (ADMNC) techniques function better for the specific distributions in these datasets. Comparing LSHAD with ADMNC, ADMNC slightly outperformed LSHAD in several datasets. Nonetheless, the weakest performance of LHSAD was an impressive AUC of 71%.

Note that the optimal values defined for LHSAD hyperparameters tested on medium datasets (Section 5) also hold for large datasets, as indicated by the high performance results. This would suggest that LSHAD is suitable for processing large-dimension datasets, with acceptable accuracy rates, as it is among the best performing algorithms and also is among the most scalable methods.

6.4. Scalability versus AD performance

We used the Pareto optimization method [49] to evaluate the tradeoff between scalability and AD for the algorithms. In multi-objective optimization, the Pareto front is defined as the border between the region of feasible points (not strictly dominated by any other) for which all constraints are satisfied and the region of unfeasible points (dominated by others).

Fig. 13 plots all the algorithms used in our study, maximizing the average AUC (*X* axis) and minimizing processing speed (*Y* axis). To compute t time complexity we used the number of samples of the largest dataset n that each algorithm was capable of handling and the processing time t required, that is, $\frac{log(t)}{log(n)}$. Fig. 13 shows that LSHAD, LOCI, and SVM-R are on the Pareto front, although note that LOCI and SVM-R were unable to process the largest datasets.

In summary, in our experiments for accuracy and scalability, LSHAD is demonstrated to be among the best state-of-the-art methods, and has the additional advantage of hyperparameter autotuning.

7. Conclusions and Future Work

LSHAD is a novel algorithm based on the LSH technique, developed in order to obtain an AD model that could handle large-scale datasets. We leverage LSH, which enables groups of similar data points to be detected, to estimate the density of the input space regions, which is used, in turn, to estimate the probability of a data point being an anomaly. Our algorithm, implemented in the Apache Spark framework, is tailored for distributed environments and so is capable of processing large datasets due to its scalability properties. An important advantage of our method is its AutoML feature, which implements automatic hyperparameter tuning, and thereby reduces computational resource needs and the time required for manual hyperparameter tuning.

The LSHAD algorithm was compared for AD and scalability performances with state-of-art methods in a variety of datasets. Our empirical study demonstrates that LSHAD is comparable to the best available methods in achieving satisfactory AD results for both synthetic and real datasets, and performs better than other methods in terms of scalability, especially with very large datasets. In summary, our contributions are as follows:

1. We propose a novel AD method based on LSH that obtains accuracy results on a par with state-of-the art methods and scalability results that outperform those of any of its competitors.
2. The model manages distributed scenarios, as it was developed using the Apache Spark framework and so can distribute data processing across multiple clusters.
3. The model automates the time-consuming and error-prone hyperparameter tuning process, which not only improves efficiency, but also makes the algorithm available to non-expert users in the ML field, currently not a feature of most AD models.

As future work we plan to continue researching LSHAD capabilities with the intention of implementing an online version of this algorithm for dealing with data streams. Our parallelized implementation in the Apache Spark framework is available[12] for further improvement or for use in the AD field.

CRediT authorship contribution statement

Jorge Meira: Investigation, Conceptualization, Software, Writing - original draft, Writing - review & editing. **Carlos Eiras-Franco:** Software, Validation, Writing - review & editing. **Verónica Bolón-Canedo:** Supervision, Writing - review & editing. **Goreti Marreiros:** Supervision, Writing - review & editing. **Amparo Alonso-Betanzos:** Supervision, Writing - review & editing.

Data availability

We have shared in the manuscript the links to all the methods and data used in our work.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been financially supported in part by the Spanish Ministerio de Economía y Competitividad (project PID-2019-109238GB-C22) and by the Xunta de Galicia (grants ED431C 2018/34 and ED431G 2019/01) through European Union ERDF funds. CITIC, as a research center accredited by the Galician University System, is funded by the Consellería de Cultura, Educación e Universidades of the Xunta de Galicia, supported 80% through ERDF Funds (ERDF Operational Programme Galicia 2014–2020) and 20% by the Secretaría Xeral de Universidades (Grant ED431G 2019/01).This work was also

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

supported by National Funds through the Portuguese FCT - Fundação para a Ciência e a Tecnologia (projects UIDB/00760/2020 and UIDP/00760/2020).

References

[1] B. Bai, Z. Guo, C. Zhou, W. Zhang, J. Zhang, Application of adaptive reliability importance sampling-based extended domain PSO on single mode failure in reliability engineering, Information Sciences 546 (2021) 42–59.
[2] R.M. Souza, E.G. Nascimento, U.A. Miranda, W.J. Silva, H.A. Lepikson, Deep learning for diagnosis and classification of faults in industrial rotating machinery, Computers & Industrial Engineering 153 (2021) 107060.
[3] X. Kan, Y. Fan, Z. Fang, L. Cao, N.N. Xiong, D. Yang, X. Li, A novel IoT network intrusion detection approach based on adaptive particle swarm optimization convolutional neural network, Information Sciences 568 (2021) 147–162.
[4] X. Li, Z. Hu, M. Xu, Y. Wang, J. Ma, Transfer learning based intrusion detection scheme for Internet of vehicles, Information Sciences 547 (2021) 119–135.
[5] F. Carcillo, Y.-A. Le Borgne, O. Caelen, Y. Kessaci, F. Oblé, G. Bontempi, Combining unsupervised and supervised learning in credit card fraud detection, Information sciences 557 (2021) 317–331.
[6] X. Zhang, Y. Han, W. Xu, Q. Wang, HOBA: A novel feature engineering methodology for credit card fraud detection with a deep learning architecture, Information Sciences 557 (2021) 302–316.
[7] M. Hammad, R.N. Kandala, A. Abdelatey, M. Abdar, M. Zomorodi-Moghadam, R. San Tan, U.R. Acharya, J. Pławiak, R. Tadeusiewicz, V. Makarenkov, et al, Automated detection of shockable ECG signals: a review, Information Sciences 571 (2021) 580–604.
[8] P. Feng, J. Fu, Z. Ge, H. Wang, Y. Zhou, B. Zhou, Z. Wang, Unsupervised semantic-aware adaptive feature fusion network for arrhythmia detection, Information Sciences 582 (2022) 509–528.
[9] H. Fanta, Z. Shao, L. Ma, SiTGRU: single-tunnelled gated recurrent unit for abnormality detection, Information Sciences 524 (2020) 15–32.
[10] P. Mishra, C. Piciarelli, G.L. Foresti, A neural network for image anomaly detection with deep pyramidal representations and dynamic routing, International Journal of Neural Systems 30 (10) (2020) 2050060.
[11] P. Tang, W. Qiu, Z. Huang, S. Chen, M. Yan, H. Lian, Z. Li, Anomaly detection in electronic invoice systems based on machine learning, Information Sciences 535 (2020) 172–186.
[12] S. Kandanaarachchi, Unsupervised anomaly detection ensembles using item response theory, Information Sciences 587 (2022) 142–163.
[13] V. Chandola, Anomaly Detection: A Survey, Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications 41 (3) (2009) 71–97, https://doi.org/10.1016/B978-0-12-398537-8.00004-3.
[14] H.-P. Kriegel, P. Kröger, A. Zimek, Outlier detection techniques, Tutorial at KDD 10..
[15] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, LOF: identifying density-based local outliers, in: ACM sigmod record, vol. 29, ACM, 93–104, 2000..
[16] W. Jin, A.K.H. Tung, 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, ACM Press, New York, New York, USA, 293–298, ISBN 158113391X, 2001, DOI: 10.1145/502512.502554, http://portal.acm.org/citation.cfm?doid=502512.502554..
[17] J. Tang, Z. Chen, A.W. Fu, D.W. Cheung, Capabilities of outlier detection schemes in large datasets, framework and methodologies, Knowledge and Information Systems 11 (1) (2006) 45–84, ISSN 0219–1377, DOI: 10.1007/s10115-005-0233-6, http://link.springer.com/10.1007/s10115-005-0233-6..
[18] S. Papadimitriou, H. Kitagawa, P.B. Gibbons, C. Faloutsos, Loci: Fast outlier detection using the local correlation integral, in: Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405), IEEE, 315–326, 2003..
[19] H.-P. Kriegel, P. Kröger, E. Schubert, A. Zimek, LoOP: local outlier probabilities, in: Proceedings of the 18th ACM conference on Information and knowledge management, ACM, 2009, pp. 1649–1652.
[20] C. Eiras-Franco, D. Martínez-Rego, B. Guijarro-Berdiñas, A. Alonso-Betanzos, A. Bahamonde, Large scale anomaly detection in mixed numerical and categorical input spaces, Information Sciences 487 (2019) 115–127.
[21] C. Eiras-Franco, B. Guijarro-Berdiñas, A. Alonso-Betanzos, A. Bahamonde, A scalable decision-tree-based method to explain interactions in dyadic data, Decision Support Systems 127 (2019) 113141.
[22] M. Bahri, F. Salutari, A. Putina, M. Sozio, AutoML: state of the art with a focus on anomaly detection, challenges, and research directions, International Journal of Data Science and Analytics (2022) 1–14.
[23] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM, 604–613, 1998..
[24] Z. Li, J. Tang, L. Zhang, J. Yang, Weakly-supervised semantic guided hashing for social image retrieval, International Journal of Computer Vision 128 (8) (2020) 2265–2278.
[25] X. Chi, C. Yan, H. Wang, W. Rafique, L. Qi, Amplified locality-sensitive hashing-based recommender systems with privacy protection, Concurrency and Computation: Practice and Experience (2020) e5681..
[26] M.A. Abdulhayoglu, B. Thijs, Use of locality sensitive hashing (LSH) algorithm to match Web of Science and Scopus, Scientometrics 116 (2) (2018) 1229–1245.
[27] A. Smiti, A critical overview of outlier detection methods, Computer Science Review 38 (2020) 100306.
[28] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, ACM Transactions on Knowledge Discovery from Data (TKDD) 6 (1) (2012) 3.
[29] B. Schölkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, R.C. Williamson, Estimating the support of a high-dimensional distribution, Neural computation 13 (7) (2001) 1443–1471.
[30] D. Martínez-Rego, D. Fernández-Francos, O. Fontenla-Romero, A. Alonso-Betanzos, Stream change detection via passive-aggressive classification and Bernoulli CUSUM, Information Sciences 305 (2015) 130–145.
[31] A. Bouguettaya, H. Zarzour, A.M. Taberkit, A. Kechida, A review on early wildfire detection from unmanned aerial vehicles using deep learning-based computer vision algorithms, Signal Processing 190 (2022) 108309.
[32] T.J. Park, N. Kanda, D. Dimitriadis, K.J. Han, S. Watanabe, S. Narayanan, A review of speaker diarization: Recent advances with deep learning, Computer Speech & Language 72 (2022) 101317.
[33] Y. Han, Y. Lang, M. Cheng, Z. Geng, G. Chen, T. Xia, DTaxa: An actor–critic for automatic taxonomy induction, Engineering Applications of Artificial Intelligence 106 (2021) 104501.
[34] Z. Geng, Y. Zhang, Y. Han, Joint entity and relation extraction model based on rich semantics, Neurocomputing 429 (2021) 132–140.
[35] W. Hong, E.J. Hwang, J.H. Lee, J. Park, J.M. Goo, C.M. Park, Deep Learning for Detecting Pneumothorax on Chest Radiographs after Needle Biopsy: Clinical Implementation, Radiology 211706 (2022).
[36] X. Hu, Y. Han, Z. Geng, A novel matrix completion model based on the multi-layer perceptron integrating kernel regularization, IEEE Access 9 (2021) 67042–67050.
[37] R. Chalapathy, S. Chawla, Deep learning for anomaly detection: A survey, arXiv preprint arXiv:1901.03407..
[38] T. Cemgil, S. Ghaisas, K. Dvijotham, S. Gowal, P. Kohli, The Autoencoding Variational Autoencoder, Advances in Neural Information Processing Systems 33 (2020) 15077–15087.
[39] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, in: TensorFlow: Concepts, tools, and techniques to build intelligent systems, O'Reilly Media, 2019.
[40] Y. Wang, S. Parthasarathy, S. Tatikonda, Locality Sensitive Outlier Detection: A ranking driven approach, in: 2011 IEEE 27th International Conference on Data Engineering, IEEE, 410–421, ISBN 978-1-4244-8959-6, 2011, DOI: 10.1109/ICDE.2011.5767852, http://ieeexplore.ieee.org/document/5767852/..

[41] M.R. Pillutla, N. Raval, P. Bansal, K. Srinathan, C. Jawahar, LSH based outlier detection and its application in distributed setting, in: Proceedings of the 20th ACM international conference on Information and knowledge management, ACM, 2011, pp. 2289–2292.
[42] X. Zhang, M. Salehi, C. Leckie, Y. Luo, Q. He, R. Zhou, R. Kotagiri, Density biased sampling with locality sensitive hashing for outlier detection, in: International Conference on Web Information Systems Engineering, Springer, 269–284, 2018..
[43] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), 15–28, 2012..
[44] D. Dua, E. Karra Taniskidou, UCI Machine Learning Repository [http://archive. ics. uci. edu/ml]. Irvine, CA: University of California, School of Information and Computer Science..
[45] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (Jan) (2006) 1–30.
[46] E. Castillo, D. Peteiro-Barral, B.G. Berdiñas, O. Fontenla-Romero, Distributed one-class support vector machine, International Journal of Neural Systems 25 (07) (2015) 1550029.
[47] A. Parmisano, S. Garcia, M.J. Erquiaga, Stratosphere Laboratory. Aposemat IoT-23. A labeled dataset with malicious and benign IoT network traffic., https://www.stratosphereips.org/datasets-iot23, 2020..
[48] X. Chen, D.P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, P. Abbeel, Variational lossy autoencoder, arXiv preprint arXiv:1611.02731..
[49] J. Teich, Pareto-front exploration with uncertain objectives, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 314–328, 2001..