Information Sciences 690 (2025) 121587

Contents lists available at ScienceDirect

Information Sciences

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

Developing Big Data anomaly dynamic and static detection algorithms: AnomalyDSD spark package

Diego García-Gil [a,c,*], David López [b,c], Daniel Argüelles-Martino [d], Jacinto Carrasco [b,c], Ignacio Aguilera-Martos [b,c], Julián Luengo [b,c], Francisco Herrera [b,c]

[a] Department of Software Engineering, University of Granada, 18071, Granada, Spain
[b] Department of Computer Science and Artificial Intelligence, University of Granada, 18071, Granada, Spain
[c] Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Granada, 18071, Granada, Spain
[d] ArcelorMittal Global R&D, New Frontier, Digital Portfolio, Spain

ARTICLE INFO

Keywords:
Big Data
Anomaly detection
Outlier detection
Unsupervised learning

ABSTRACT

Background: Anomaly detection is the process of identifying observations that differ greatly from the majority of data. Unsupervised anomaly detection aims to find outliers in data that is not labeled, therefore, the anomalous instances are unknown. The exponential data generation has led to the era of Big Data. This scenario brings new challenges to classic anomaly detection problems due to the massive and unsupervised accumulation of data. Traditional methods are not able to cop up with computing and time requirements of Big Data problems.
Methods: In this paper, we propose four distributed algorithm designs for Big Data anomaly detection problems: HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD. They have been designed following the MapReduce distributed methodology in order to be capable of handling Big Data problems.
Results: These algorithms have been integrated into an Spark Package, focused on static and dynamic Big Data anomaly detection tasks, namely AnomalyDSD. Experiments using a real-world case of study have shown the performance and validity of the proposals for Big Data problems.
Conclusions: With this proposal, we have enabled the practitioner to efficiently and effectively detect anomalies in Big Data datasets, where the early detection of an anomaly can lead to a proper and timely decision.

1. Introduction

Anomaly detection refers to the challenge of finding observations that differs significantly from the remaining data. These unexpected patterns are typically referred as outliers or anomalies [10,17]. In most cases, data is generated by one or several processes that reflects the system behavior. When such system behaves wrongly, it produces anomalies or outliers. The identification of such anomalous observations has a crucial importance for identifying the system's erratic behavior [1,36,12]. Anomaly detection has a wide variety of domains, such as financial fraud detection [28], intrusion detection [30], sensor networks [32], industrial anoma-

* Corresponding author.
E-mail addresses: djgarcia@ugr.es (D. García-Gil), derwey@correo.ugr.es (D. López), daniel.arguelles@arcelormittal.com (D. Argüelles-Martino), jacintocc@decsai.ugr.es (J. Carrasco), nacheteam@ugr.es (I. Aguilera-Martos), julianlm@decsai.ugr.es (J. Luengo), herrera@decsai.ugr.es (F. Herrera).

https://doi.org/10.1016/j.ins.2024.121587
Received 28 February 2024; Received in revised form 20 October 2024; Accepted 21 October 2024

Available online 26 October 2024

lies [31], or health care [46]. For example, an anomalous behavior in an engine may signify that it is close to a breakdown. Detecting anomalies in engine sensors could prevent failure of those engines. Due to the wide variety of domains in which anomaly detection can be applied and the relevance of the benefits it brings, anomaly detection problem has an increasing importance nowadays. There are three different types of problems in anomaly detection [10]:

- Supervised anomaly detection: The dataset is labeled indicating which instances are normal and which are anomalous. A predictive model is built for separating normal and anomalous instances.
- Semi-supervised anomaly detection: The training set used does not have anomaly instances, only normal observations. The anomalous instances are provided in the test set.
- Unsupervised anomaly detection: The instances are not labeled. The anomalous instances are unknown and the algorithm should be able to detect them without previous knowledge.

Due to the automation in data acquisition and storage, most real-world anomaly detection problems belong to the unsupervised type. The unsupervised anomaly detection scenario is particularly challenging because machine learning approaches have no previous knowledge about the data [7]. Unsupervised anomaly detection algorithms score the data based on properties of the dataset only. This score represents the degree of "outlierness" of each instance. Then, using either a threshold or a fixed amount, anomalies are selected [34]. Unsupervised anomaly detection algorithms can be grouped in four categories [1]: nearest neighbors-based [6], clustering-based [26], statistical methods [40], and ensembles [48]. Recently, a toolbox for outlier detection was proposed, named PyOD [49]. This library provides a wide range of anomaly detection algorithms, including both well established methods and recent approaches.

The improvement of technologies as well as the birth of new ones such as smartphones, 5G communications, sensors, the internet cloud, virtual reality and smart home applications results in a huge volume of data being generated rapidly. The growth in the amount of data generated has led to the era of Big Data [45]. Big Data refers to high-volume, high-velocity, and high-variety data that can not be processed by conventional methods. It has created the necessity to develop specific methods for the different data that may arrive [35,23,41]. The automation in data acquisition, popularization of sensors, and lack of human supervision that characterizes Big Data has increased the need for efficient anomaly detection methods. The nature of Big Data makes it unable to be supervised and labeled by an expert in most cases. That problematic generates that the majority of real-world Big Data anomaly detection problems are unsupervised. Despite having very popular libraries such as PyOD [49] for normal-sized anomaly detection problems, in Big Data we can only find a handful of proposals devoted to specific domains of this problem [23,45,8,37].

In this paper, we tackle the anomaly detection problem by proposing four Big Data anomaly detection distributed designs: HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD. The first algorithm, HBOS_BD, is a distributed histogram-based anomaly detector. The second anomaly detector, LODA_BD, is a distributed histogram-based ensemble algorithm that performs random projections of the data. The third algorithm, LSCP_BD, is based on the use of different base anomaly detection methods and a novel distributed clustering-based partitioning strategy. The last algorithm, XGBOD_BD, is a distributed semi-supervised anomaly detection algorithm that employs unsupervised anomaly detection base detectors. These algorithms have been designed following the MapReduce distributed methodology in order to be capable of handling Big Data problems. They constitute the first suite of general anomaly detectors specifically tailored for unsupervised anomaly detection in Big Data scenarios. All four algorithms have been collected into a package focused on dealing with static and dynamic Big Data anomaly detection problems, named AnomalyDSD. This package has been implemented in Apache Spark and is available publicly in GitHub.[1]

We study the performance of the proposed algorithms in both a real-world case of study using sensor data provided by ArcelorMittal,[2] as well as in a collection of public normal-sized benchmark datasets for validating the proposals. The results obtained are evaluated using the Area Under the Receiver Operating Characteristic (ROC-AUC) metric. The computing times and scalability of the proposals in terms of size of the data, number of threads and workers are also studied. Achieved results show that the proposed algorithms are able to successfully and efficiently detect anomalies in Big Data problems.

The remainder of this paper is organized as follows: Section 2 presents the concept of unsupervised anomaly detection and the current state of anomaly detection in Big Data environments. Section 3 details the proposed distributed designs for static and dynamic Big Data anomaly detection. Section 4 contains the details of the AnomalyDSD package. Section 5 shows the experiments carried out to assess the performance of the proposals. Finally, Section 6 concludes the paper.

2. The unsupervised anomaly detection problem

In this section, we introduce the unsupervised anomaly detection problem in Section 2.1. We describe the actual situation in Big Data anomaly detection problem in Section 2.2.

For a deeper description of the anomaly detection problem and its challenges, refer to Appendix A. Similarly, Appendix B describes the MapReduce framework and the Apache Spark engine in detail.

[1] https://github.com/ari-dasci/S-AnomalyDSD.
[2] https://corporate.arcelormittal.com/.

2.1. Unsupervised anomaly detection

Unsupervised anomaly detection problem refers to particular type of problems in anomaly detection in which the instances of the dataset are unlabeled. This characteristic makes the process of finding anomalies more challenging since there is no ground-truth, nor the algorithms have prior knowledge of what an anomaly is. There are four different types of algorithms for the unsupervised anomaly detection problem [10].

- Nearest neighbors based: The outliers are determined by their distances or densities to their nearest neighbors/regions [6].
- Clustering based: The centroid is computed by a clustering algorithm and outliers are detected because they have a large distance to the dense areas [26].
- Statistical methods: Based on properties of the data solely, such as data dispersion or histograms [21,40].
- Ensemble methods: Different base anomaly detection methods and combination strategies are used to generate a more complex model [48,47].

Testing the effectiveness of an unsupervised anomaly detection algorithm is a very complex task since there is no ground-truth labeling of data. Most anomaly detection algorithms detect whether an observation is anomalous by applying a threshold to each score obtained by the algorithm. The difficulty of applying a threshold is that if the threshold is too high, it will generate a high number of false positives since it will catalog normal observations as anomalous. On the other hand, if the threshold is too small it will ignore some anomalous observations resulting in a higher number of false negatives. The metrics that are used to test the effectiveness of an algorithm are $precision$ (percentage of anomalies detected by the algorithm that are real anomalies) and $recall$ (percentage of ground-truth anomalies that have been detected as anomalies). Related to these metrics is the $ROC\ curve$ (Receiver Operating Characteristic curve), which represents in the X-axis the true positive rate ($recall$) and in the Y-axis the false positive rate (percentage of anomalies badly detected over the ground-truth normal observations) [1]. The most employed metric for evaluating the performance of unsupervised anomaly detection methods is the ROC-AUC (Area Under Curve) which is defined in [25] as: "*Given a ranking or scoring of a set of points in order of their propensity to be outliers (with higher ranks/scores indicating greater outlierness), the ROC-AUC is equal to the probability that a randomly selected outlier-inlier pair is ranked correctly (or scored in the correct order)*".

2.2. Big Data anomaly detection

With the explosion of data, popularization of sensors, and automation in data acquisition and storage, the anomaly detection problem has become a Big Data problem. Certain domains such as network intrusion detection or failures prevention needs to be addressed rapidly to avoid major issues. Moreover, with this exponential growth of the data, classical algorithms cannot process the data in a reasonable amount of time [35,23].

While there are some proposals in the literature for Big Data anomaly detection, most of them follow a similar clustering approach [23,45,4,43,33]. Other studies such as the proposed in [42], perform different steps for the selection of characteristics and subsequent data labeling, using classifiers such as SVM, Naïve Bayes or Random Forest. The method proposed in [37] follows a machine learning unsupervised approach for detecting anomalies in electricity consumption time-series domains. The method developed in [4], called *SSWLOFCC*, is based on composite clustering and big data technologies. In [8], authors perform an in-depth review focused on the application of metaheuristics and machine learning techniques for big data domains. Authors in [5] propose a deep learning methodology, consisting of Convolutional Neural Networks and Long Short-Term Memory networks for the real-time detection of network traffic anomalies within Big Data environments. In [4], authors conclude that to address the Big Data challenges, modernized machine learning algorithms need to be proposed.

Thus, algorithms are required for processing those large amounts of data in an efficient way. However, there are not general methods nor frameworks that helps in Big Data anomaly detection problems. Therefore, we aim to provide a distributed set of algorithm designs with different working mechanisms for Big Data anomaly detection problems that can solve a wide variety of anomaly detection tasks efficiently.

3. Big Data anomaly dynamic and static detection

In this section, we propose four Big Data anomaly detection distributed algorithms. These algorithms have been designed following the MapReduce paradigm, where all processes are performed in a distributed fashion.

In Section 3.1 we describe the Apache Spark primitives employed in the design of the proposals. The first algorithm is described in Section 3.2. It is based on drawing histograms of the features, named HBOS_BD. Section 3.3 describes an ensemble based on histograms calculation, namely LODA_BD. In Section 3.4 we describe an ensemble method based on the use of different base anomaly detection methods, LSCP_BD. Finally, Section 3.5 is devoted to XGBOD_BD, a semi-supervised anomaly detection algorithm that employs unsupervised anomaly detection base detectors.

3.1. Spark primitives

For the development of the algorithms, we have used some of the available Spark primitives in the Spark API. These primitives extend the MapReduce paradigm functionalities by allowing to perform more complex operations over the data. Here, we remark those employed in the implementation of our anomaly detection algorithms[3]:

- *map*: Performs an user-defined function over each element of a distributed set. A new distributed set is created after this transformation.
- *histogram*: Calculates a histogram of the distributed set using a specified number of buckets evenly spaced between the minimum and maximum of the dataset.
- *count*: Returns the number of elements of a distributed set.
- *sort*: Returns a sorted distributed set by ascending order.
- *join*: Merges two distributed datasets instance-wise and returns a new dataset.
- *mapPartitions*: Similarly to *map* function, performs an user-defined function over each partition of a distributed set. A new distributed set is created after this transformation.
- *repartitionByRange*: Returns a new distributed set partitioned by the given columns.
- *MulticlassMetrics*: Returns a set of metrics for classification tasks such as the precision and confusion matrix.
- *xgbClassifier*: Learns a distributed XGBoost [11] model.

These Spark primitives from Spark API are used in the following sections, where the four proposed anomaly detection algorithms are described.

3.2. Histogram-based outlier score for Big Data: HBOS_BD

Histogram-Based Outlier Score (HBOS) is an histogram-based anomaly detection method [21,2]. HBOS can process multivariate datasets and has a low time complexity. HBOS constructs an histogram for every feature of the data, each histogram is split into bins and the score for every instance (p) is computed based on the height of the bin where it is located. Equation (1) shows how this score is computed:

$$f(x) = \sum_{i=0}^{k} \log \frac{1}{hist_i(p)} \tag{1}$$

HBOS proposes two alternatives to process numerical features:

- Static: Computes a standard histogram using k equal-width bins. The height of the bins is an estimation based on the relative amount of samples inside each bin.
- Dynamic: The values are sorted and a fixed number of $\frac{N}{k}$ consecutive values are grouped into a single bin. k represents the number of bins and N the length of the data. The area of each bin represents the number of instances into each bin, which is the same for all bins. The width is determined by the first and the last value of the bin. Thus, the height of each bin is defined by Equation (2):

$$\frac{\frac{N}{k}}{last - first} \tag{2}$$

Despite being a very efficient method with a low time complexity, the iterative nature of HBOS makes it unsuitable for tackling Big Data problems. HBOS_BD is a distributed implementation using Spark primitives, such as distributed histogram calculation, which enables HBOS to be applied to large datasets rapidly. In HBOS_BD, both variants of the original HBOS method (static and dynamic) have been implemented, following the same behavior as the original ones.

Algorithm 1 describes HBOS_BD static anomaly detection process:

- The first step is the histogram calculation of the data. We iterate through each one-dimensional feature and draw an histogram with *n_bins* intervals on it, obtaining the limits and number of elements. Lastly, the histogram is normalized, dividing each value by the total sum of its values (lines 5-10).
- The next step is the scores calculation of the data using a distributed *map* function (lines 12-19). First, it is necessary to find out in which bin the feature of the selected instance is located. The location is computed using the following equation: $bin \leftarrow \frac{(feature - min) \cdot n\_bins}{max - min}$ where $feature$ is the value of the instance for that feature, max is the maximum value of the bin, min is the minimum value of the bin, and *n_bins* the number of bins (line 15). Once the bin is found, the score of a single instance is computed using Equation (1) (line 16), performing the summation at the end of the process (line 19).
- Finally, the resulting distributed set containing the anomaly score of each instance is returned (line 20).

[3] For a complete description of Spark's operations, please refer to Spark's API: http://spark.apache.org/docs/latest/api/scala/index.html.

Algorithm 1 HBOS_BD Static Algorithm.

```
1: Input: data the dataset in Dataset["features"] format
2: Input: n_bins the number of bins for the histograms
3: Output: an RDD[Double] with the scores of each instance of the dataset
4: limits, histograms ← ∅
5: for i ← 0 until n_attributes do
6:     hist ← histogram(select(data, i), n_bins)
7:     append(limits(i), getLimits(hist))
8:     hist_sum ← sum(getHistogram(hist))
9:     append(histograms(i), getHistogram(hist).map(l => l/hist_sum))
10: end for
11: scores ←
12: map instance ∈ data
13:     values ← ∅
14:     for i ← 0 until n_attributes do
15:         index ← computeIndex(instance(i), limits(i))
16:         values(i) ← computeScores(histogram(i), index)
17:     end for
18:     sum(values)/n_bins
19: end map
20: return(scores)
```

Algorithm 2 HBOS_BD Dynamic Algorithm.

```
1: Input: data the dataset in Dataset["features"] format
2: Input: n_bins the number of bins for the histograms
3: Output: an RDD[Double] with the scores of each instance of the dataset
4: limits, histograms ← ∅
5: inc ← count(data) / n_bins
6: for i ← 0 until n_attributes do
7:     sortedValues ← sort(select(data, i))
8:     limits(i)._1 ← select(sortedValues(0 until count(data) − inc by inc))
9:     limits(i)._2 ← select(sortedValues(inc until count(data) by inc))
10:    hist ← computeHistogram(selected(data, i), limits(i))
11:    hist_sum ← sum(getHistogram(hist))
12:    append(histograms(i), getHistogram(hist).map(l => l/hist_sum))
13: end for
14: scores ←
15: map instance ∈ data
16:     values ← ∅
17:     for i ← 0 until n_attributes do
18:         index ← computeIndexDynamic(instance(i), limits(i))
19:         values(i) ← computeScores(histogram(i), index)
20:     end for
21:     sum(values)/k
22: end map
23: return(scores)
```

Algorithm 2 describes HBOS_BD dynamic anomaly detection process:

- To compute the histogram, the data needs to be sorted first. The instances of interest are the ones that are at the bounds of each bin. The length of each bin is defined as: $\frac{N}{n\_bins}$ values, so the index of each feature can be computed previously (lines 8-9). Once we know the bounds of each bin, the height is computed using Equation (2) (lines 10-12).
- The next step is the scores calculation of the data using a distributed *map* function (lines 15-22). First, it is necessary to find out in which bin the feature of the selected instance is located. The location is computed comparing the value of the feature within the first and last value of each bin, in order to assess if the value is between those two (line 18). Once the bin is found, the score of a single instance is computed using Equation (1) (line 19), performing the summation at the end of the process (line 21).
- Finally, the resulting distributed set containing the anomaly score of each instance is returned (line 23).

The following are required as input parameters for both methods: the dataset (*data*), and the number of bins for the histograms (*n_bins*).

Regarding the computational complexity, original HBOS algorithms work in linear time $O(n)$ in case of fixed bin width, or in $O(nlog(n))$ using dynamic bin widths. In a distributed environment, the complexity is reduced by parallelizing the histogram construction. The distributed time complexity is $O\left(\frac{n\log n}{p} + c\right)$, where p is the number of parallel processors, and c is the communication overhead due to synchronization between nodes.

3.3. Lightweight on-line detector of anomalies for Big Data: LODA_BD

A collection of weak classifiers is known to be capable of producing a strong classifier. This is the premise of Lightweight On-line Detector of Anomalies (LODA) [40], in which a collection of very weak detectors can lead to a strong anomaly detector. LODA is based on the calculation of a collection of one-dimensional histograms, each one of them drawn from the projection of the original data onto a randomly generated vector. LODA's score, f(x), for a given sample, x, is the average of the logarithm of probabilities for each individual projection vector. Being p_i the probability estimated by the ith histogram, w_i the ith projection vector, and k the number of random projections, LODA's score can be denoted as shown in Equation (3):

$$f(x) = -\frac{1}{k}\sum_{i=1}^{k} \log p_i(x^T w_i) \tag{3}$$

In spite being a fast anomaly detector with a low time and space complexity, LODA is not adapted to the Big Data paradigm nor has a distributed approach, since it presents an iterative nature. The projection of the data onto a one-dimensional vector and calculation of the histogram is performed in an iterative fashion. This process is usually performed several hundreds of times, which, in a Big Data environment, will result in the scheduling and execution of several hundreds of tasks.

LODA_BD solves the previous issues by transforming the iterative projections of the data, onto a distributed matrix operation. A matrix is created containing all desired projections, and, through a distributed matrix multiplication with the original data, the projections are calculated at once. Then, LODA_BD calculates the different histograms on the projected data and calculates the anomaly score of each instance.

Algorithm 3 LODA_BD Algorithm.

```
1:  Input: data the dataset in Dataset["features"] format
2:  Input: n_bins the number of bins for the histograms
3:  Input: k the number of projections
4:  Output: an RDD[Double] with the scores of each instance of the dataset
5:  dataAsMatrix ← RowMatrix(data)
6:  projections ← createRandomProjections(size(data), k)
7:  projectedData ← multiply(dataAsMatrix, projections)
8:  limits, histograms ← Ø
9:  for i ← 0 until k do
10:     hist ← histogram(select(projectedData, i), n_bins)
11:     append(limits(i), getLimits(hist))
12:     hist_sum ← sum(getHistogram(hist))
13:     append(histograms(i), getHistogram(hist).map(l => l/hist_sum))
14: end for
15: scores ←
16: map instance ∈ projectedData
17:     values ← Ø
18:     for i ← 0 until k do
19:         values(i) ← calculateScores(limits(i), histograms(i))
20:     end for
21:     sum(values)/k
22: end map
23: return(scores)
```

Algorithm 3 describes LODA_BD anomaly detection process:

- First, the original data is transformed from Spark's Dataset format to a distributed matrix format (RowMatrix) (line 5).
- The random projection matrix is calculated in line 6. This matrix will be formed of values sampled from a Gaussian distribution of mean 0 and variance 1. It is composed of $\sqrt{d}$ randomly selected non-zero components (being d the number of features). The result is the local matrix $projections^{d \times k}$.
- Once the data and projection vectors are in matrix format, we perform a distributed matrix multiplication (line 7). This process multiplies each partition of the distributed data (in matrix format) by the local projection matrix.
- The next step is the histogram calculation of the projected data. We iterate through each one-dimensional feature and draw and histogram with *n_bins* intervals on it, obtaining the limits and number of elements. Lastly, the histogram is normalized, dividing each value by the total sum of its values (lines 9-14).
- In the final step, the anomaly score for each instance is calculated using a distributed *map* function (lines 15-22). For each element, we iterate through the k projections, and calculate the $\log p_i(projected\,Data)$ (line 19). Finally, all scores of each instance are normalized (line 21), and the resulting distributed set containing the anomaly score of each instance is returned.

The following are required as input parameters: the dataset (*data*), the number of bins for the histograms (*n_bins*), and the number of projections (*k*).

The computational complexity of LODA_BD can be calculated as follows: LODA_BD can learn in $O\left(\frac{nkd^{\frac{-1}{2}}}{p}+c\right)$, where n is the number of training samples, d is the dimensionality of the input space, k is the number of histograms, p is the number of parallel processors, and c is the communication overhead due to synchronization between nodes. The time complexity of the classification phase is $O\left(\frac{kd^{\frac{-1}{2}}}{p}+c\right)$.

3.4. Locally selective combination in parallel outlier ensembles for Big Data: LSCP_BD

Locally Selective Combination in Parallel Outlier Ensembles (LSCP) [48] is an ensemble method which addresses the issue of not having a ground-truth in unsupervised anomaly detection. It defines a pseudo ground-truth using a number of base detectors and selecting either the average or maximum of them. Then, it defines a local region around each instance, and uses the consensus of the nearest neighbors in randomly selected feature subspaces. The best performing base detectors in such local regions are selected and combined as the ensemble's decision.

Although ensembles have shown to be capable of achieving great performance and to adapt to Big Data environments effectively [18], methods with high computational complexity struggle in such domains. LSCP defines the local region of each instance by extracting the k-Nearest Neighbors of a number of groups of randomly selected features, and then taking the most selected examples by k-NN. This process involves the calculation of several k-NN methods. In Big Data domains, this will imply the calculation of billions of distances, which can be prohibitive in terms of computing time.

LSCP_BD alleviates this high computational complexity of the local region definition by calculating an approximation of such k-Nearest Neighbors. LSCP_BD takes advantage of the concept of partitioned and distributed data for creating partitions in which all instances will have a high degree of similarity. We substitute the use of k-NN for clustering as a faster method for neighborhood definition. In particular we employ Spark's distributed implementation of k-Means, and Bisecting k-Means for measuring the similarity among instances. This can result in large clusters of normal data, and small clusters of outliers or anomalies. We process those clusters individually, splitting big clusters and joining smaller clusters together. The result is a dataset in which each partition contains the local region of those instances.

Algorithm 4 LSCP_BD Algorithm.

```
Input: data the dataset in Dataset["features"] format
Input: n_base_detectors the number of base detectors
Input: pgt_strategy the strategy for the pseudo ground-truth generation ("avg", "max")
Input: clus_method clustering method for the local region calculation ("kmeans", "bisec")
Input: n_clus number of clusters for the local region calculation
Input: dcs percentage of base detectors selected for dynamic outlier ensemble selection
Output: an RDD[Double] with the scores of each instance of the dataset
detectors ← data
for i ← 0 until n_base_detectors do
    detectors ← join(detectors, learnBaseDetector(data))
end for
pseudo_gt ←
map score ∈ detectors
    if pgt_strategy = "avg" then append(score, avg(score)) end if
    if pgt_strategy = "max" then append(score, max(score)) end if
end map
gt_clustered ← clusterPartitioning(pseudo_gt, clus_method, n_clus)
scores ←
mapPartitions partition ∈ gt_clustered
    correlations ← ∅
    for i ← 0 until n_base_detectors do
        correlations(i) ← pearson(partition(i), partition(size) − 1)
    end for
    if dcs = 1 then
        max(partition(correlations))
    else
        mostCorrelated ← top(correlations, dcs)
        if pgt_strategy = "avg" then
            max(partition(mostCorrelated))
        else
            avg(partition(mostCorrelated))
        end if
    end if
end mapPartitions
return(scores)
```

In Algorithm 4 we describe the LSCP_BD anomaly detection method:

- LSCP_BD starts by learning a number of base detectors and joining their anomaly scores using Spark's distributed $join$ function (lines 8-11). These base detectors can be any distributed anomaly detection method.
- Then, the pseudo ground-truth for each example is established using a distributed *map* function on the joined base detector scores. The maximum (or average) anomaly score of each detector for each given instance will be selected as pseudo ground-truth, according to $pgt\_strategy$ (lines 12-16).
- Once pseudo ground-truth is established, the proposed $clusterPartitioning$ function is applied to the data with the selected clustering method ($clus\_method$) and number of clusters (n_clus). This process is described in depth in Algorithm 5. The result is a dataset with each partition containing a local region (line 17).
- Next, the model selection and combination process is applied to each partition using a distributed $mapPartitions$ function (lines 18-34). First, the correlation of each base detector with the pseudo ground-truth is calculated using the $pearson$ correlation (lines 20-23). Then, LSCP_BD will choose the selection and combination strategy attending to the percentage of base detectors selected for dynamic outlier ensemble selection (dcs). A dcs of 1 will imply to use all base detectors and to select the most correlated as the ensemble's decision (lines 24-25). If $dcs < 1$, the top dcs most correlated base detectors will be selected and, attending to $pgt\_strategy$, the average or maximum of them will be selected (Average-of-Maximum or Maximum-of-Average) (lines 26-34).
- Finally, the distributed set with the ensemble's decision for each instance is returned (line 35).

The following are required as input parameters: the dataset (*data*), the number of base detectors (*n_base_detectors*), the strategy for the pseudo ground-truth generation (*pgt_strategy*), the clustering method for the local region calculation (*clus_method*), the number of clusters for the local region calculation (*n_clus*), and the percentage of base detectors selected for dynamic outlier ensemble selection (*dcs*).

Algorithm 5 depicts the process of partitioning the data by similarity using a clustering method.

- This function starts by applying a distributed clustering method from Spark's MLlib (k-Means or Bisecting k-Means), with n_clus different clusters. The predicted data will have a cluster index assigned to each instance (lines 8-9).
- Then, the distributed data will be repartitioned according to those assigned clusters using Spark's $repartitionByRange$ function (line 10).
- The next step is to check each partition using a distributed $mapPartition$ function, for assessing if the number of instances is optimal ($size < max\_size \wedge size > min\_size$) (lines 12-21). If the partition's size is greater than the threshold (max_size), chunks of size max_size will be selected and assigned to a sub-cluster (e.g. cluster 1 will be divided in sub-clusters 1.1, 1.2, ...). On the other hand, partitions smaller than min_size threshold will be joined together into a new cluster.
- Finally, the resulting data with the new cluster indexes is repartitioned using $repartitionByRange$ function (line 22).

Algorithm 5 clusterPartitioning Function.

```
1: Input: data the dataset in Dataset["features"] format
2: Input: clus_method clustering method for the local region calculation ("kmeans", "bisec")
3: Input: n_clus number of clusters for the local region calculation
4: Input: max_size maximum number of elements per partition (default = 10,000)
5: Input: min_size minimum number of elements per partition (default = 1,000)
6: Output: an RDD[Double] with the scores of each instance of the dataset
7: clustering ← ∅
8: if clus_method = "kmeans" then clustering ← KMeans(data, n_clus) end if
9: if clus_method = "bisec" then clustering ← BisectingKMeans(data, n_clus) end if
10: repartitionedData ← repartitionByRange(clustering, "cluster")
11: balancedData ←
12: mapPartitions partition ∈ repartitionedData
13:     clusterSize ← size(partition)
14:     if clusterSize > max_size then
15:         for i ← 0 until clusterSize by max_size do
16:             partition(i).cluster ← partition(i).cluster + 0.1
17:         end for
18:     else if clusterSize < min_size then
19:         partition(i).cluster ← ∞
20:     end if
21: end mapPartitions
22: return(repartitionByRange(balancedData, "cluster"))
```

The following are required as input parameters: the dataset (*data*), the clustering method for the local region calculation (*clus_method*), the number of clusters for the local region calculation (*n_clus*), maximum number of elements per partition *max_size*, and minimum number of elements per partition *min_size*.

The time complexity of LSCP_BD depends on its different components. For the calculation, we will consider the use of Bisecting k-Means for the clustering step. As stated in [18], the computational complexity of Spark's distributed Bisecting k-Means is $O\left(\frac{n \cdot k \cdot d}{p} + c\right)$, where n is the number of data points, k is the number of clusters, d is the number of dimensions, p is the number of processors, and c is the communication overhead due to synchronization between nodes. Let B be the number of base detectors, each with

a distributed complexity $C_{b,dist}$, the complexity of the base detectors is $O\left(\frac{n \cdot C_{b,dist}}{p} + c\right)$. Lastly, the combination of scores can be calculated as $O\left(\frac{n \cdot B}{p} + c\right)$. Combining all these components, the total complexity in a distributed scenario with Bisecting k-Means is: $O\left(\frac{n \cdot k \cdot d}{p} + c\right) + O\left(\frac{n \cdot C_{b,dist}}{p} + c\right) + O\left(\frac{n \cdot B}{p} + c\right)$. Simplified, the overall complexity can be expressed as: $O\left(\frac{n}{p} \cdot (k \cdot d + C_{b,dist} + B) + c\right)$.

3.5. Extreme gradient boosting outlier detection for Big Data: XGBOD_BD

Extreme Gradient Boosting Outlier Detection (XGBOD) [47] is a recently proposed semi-supervised ensemble method for anomaly detection. This method combines the potential of both supervised and unsupervised approaches for anomaly detection. XGBOD uses a set of unsupervised anomaly detection algorithms to extract valuable representations of the structure of the data that increases the predictive capacities of a supervised classifier. The original set of features is expanded with a set of selected Transformed Outlier Scores (TOS), then, an XGBoost classifier is learned on the expanded data.

Despite being an efficient ensemble method, XGBOD makes use of several base unsupervised anomaly detection methods for selecting the different TOS. As we stated earlier, in environments with huge amounts of data, classic unsupervised methods are not capable of reaching the desired performance. Moreover, XGBOD applies the iterative classifier (XGBoost) to the expanded data. This results in XGBOD's inability of handling Big Data datasets.

XGBOD_BD tackles these issues by employing a distributed unsupervised anomaly detection method for TOS learning. It also makes use of Spark's distributed implementation of XGBoost. XGBOD_BD implements two different strategies for TOS selection: random and accuracy-based. The random strategy selects randomly and without replacement *n_selected_detectors* detectors. The accuracy strategy selects the top *n_selected_detectors* detectors based on the accuracy measure. For stating the accuracy of the different TOS, a threshold is applied. With this threshold, the TOS are labeled and the accuracy is calculated. Selected TOS are added to the original set of features, and a distributed XGBoost model is learned.

Algorithm 6 XGBOD_BD Algorithm.

```
1:  Input: data the dataset in Dataset["features", "label"] format
2:  Input: n_TOS the number of TOS
3:  Input: n_selected_TOS the number of selected TOS
4:  Input: TOS_strategy the strategy for the TOS selection ("rnd", "acc")
5:  Input: threshold the threshold for the accuracy strategy
6:  Output: an RDD[Double] with the label of each instance of the dataset
7:  TOS ← data
8:  for i ← 0 until n_TOS do
9:      TOS ← join(TOS, learnBaseDetector(data))
10: end for
11: selected_TOS ← ∅
12: if TOS_strategy = "rnd" then
13:     selected_TOS ← randomList(0, n_TOS, n_selected_TOS)
14: else
15:     labeled_TOS ←
16:     map (instance, scores) ∈ TOS
17:         for i ← 0 until n_TOS do
18:             if scores(i) >= threshold then (instance, 1) else (instance, 0)
19:         end for
20:     end map
21:     accuracy ← ∅
22:     for i ← 0 until n_TOS do
23:         accuracy(i) ← MulticlassMetrics(labeled_TOS, "accuracy")
24:     end for
25:     selected_TOS ← indices(sort(accuracy))
26: end if
27: features_combined ←
28: map (instance, scores) ∈ TOS
29:     select(selected_TOS, scores)
30: end map
31: predictions ← xgbClassifier(features_combined)
32: return(predictions)
```

In Algorithm 6 we describe the XGBOD_BD anomaly detection method:

- XGBOD_BD begins learning n_TOS and joining the anomaly scores using Spark's distributed *join* function (lines 7-10). These base unsupervised anomaly detectors can be any distributed anomaly detection method.
- Next, the TOS selection process is applied (lines 11-26). If the selected strategy is *random*, XGBOD_BD will pick $n\_selected\_TOS$ indices randomly and without replacement (lines 12-13). On the other hand, if the selected strategy is *accuracy* each TOS is labeled using a distributed *map* function and the provided *threshold* (lines 15-20). Then, the accuracy of each labeled TOS is

calculated using Spark's distributed metrics $MulticlassMetrics$, comparing the original label with the labeled TOS (lines 21-24). The $n\_selected\_TOS$ TOS indices with the higher accuracy are selected (line 25).
- Once the $n\_selected\_TOS$ TOS indices have been selected, the original features and the selected TOS are combined together (lines 27-30).
- Finally, a distributed XGBoost model is learned on the combined data, and its predictions are returned as the decision of the method.

The following are required as input parameters: the dataset (*data*), the number of TOS (*n_TOS*), the number of selected TOS (*n_selected_TOS*), the strategy for the TOS selection (*TOS_strategy*), and the threshold for the accuracy strategy (*threshold*).

The computational complexity of XGBOD_BD algorihtm can be reduced to the complexity of the XGBoost algorithm, and the learning of the different TOS. Similarly to LSCP_BD, the complexity of learning the different TOS is $O\left(\frac{n \cdot C_{b,dist}}{p} + c\right)$, being B be the number of base detectors, each with a distributed complexity $C_{b,dist}$, p is the number of processors, and c is the communication overhead due to synchronization between nodes. XGBoost learning complexity is $O\left(\frac{Kd||x||_0 \log n}{p} + c\right)$, with K the total number of trees, d be the maximum depth of the tree, and $||x||_0$ the number of non-missing entries in the training data. The prediction complexity of XGBoost is $O\left(\frac{Kd}{p} + c\right)$. Combining all these components, the total complexity in a distributed scenario is: $O\left(\frac{n \cdot C_{b,dist}}{p} + c\right) + O\left(\frac{Kd||x||_0 \log n}{p} + c\right) + O\left(\frac{Kd}{p} + c\right)$.

4. AnomalyDSD spark package

In this section, we present AnomalyDSD, the first anomaly detection package for static and dynamic Big Data. AnomalyDSD is composed of four distributed algorithm designs: HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD. This package has been implemented keeping in mind the easiness of use of Spark's philosophy. It has been developed under the Apache Spark version 3.0.1, taking advantage of all improvements and latest additions in terms of performance. All included algorithms have been programmed following the same criteria such as variable names and methods/functions, so that users can easily change between methods.

In order to use AnomalyDSD package it is necessary to first include the package in the Spark application by using:

```
> $SPARK_HOME/bin/spark-shell --packages ari-dasci:S-AnomalyDSD:1.0
```

Listing 1: Including AnomalyDSD in Spark application.

The next step is to import the AnomalyDSD package in order to be able to access to all its included methods:

```
import org.apache.spark.mllib.anomaly
```

Listing 2: Import AnomalyDSD package.

To run any of the algorithms developed, an instance of the class that corresponds to this algorithm must be created. All algorithms include a **fit** method that runs the algorithm (e.g. HBOS_BD(*parameters*).fit()). Each method includes a set of default values for its parameters, so it is not mandatory for the user to enter any parameters. The parameters are passed during the class instantiation, the **fit** method has no parameters. All methods have a common parameter which is the required dataset. This dataset must follow Spark's default type for the ML library. It must be a **Dataset[Row]** with a column named *features* containing all the features of the dataset in a **DenseVector** format. For the XGBOD algorithm, the dataset must also have a column of type **Double** called *label* that contains the labels of the instances.

The output of the algorithms is an **RDD[Double]** which contains a score for each feature. This RDD follows the same ordering of the input data. Listing 3 shows an example of loading a multi-column dataset, transform it to a **Dataset[Row]**, and running the HBOS_BD algorithm.

```
import org.apache.spark.mllib.anomaly
import org.apache.spark.ml.feature.VectorAssembler

val path = "file:///example/of/path/"

// Data loading
val raw_data: Dataset[Row] = spark.read.parquet(path)

// Data as Vector
val assembler: VectorAssembler = new VectorAssembler()
```

```
    .setInputCols(raw_data.columns)
    .setOutputCol("features")
    val full_dataset = assembler.transform(raw_data).select("features")

    // Run HBOS_BD algorithm
    val scores = new HBOS_BD(dataset = full_dataset, n_bins = 100, strategy = "static").fit()
```

Listing 3: Example of running HBOS_BD algorithm.

For further information about parameters refer to AnomalyDSD Spark Package website: https://github.com/ari-dasci/S-AnomalyDSD.

5. A real case of study using sensor data from an ArcelorMittal machine

This section describes the experimental details and analysis carried out to assess the performance of the proposed distributed anomaly detectors over a Big Data real case of study. In Section 5.1, we present the real case of study dataset. Section 5.2 details the experimental framework. We analyze the performance of the proposed algorithms in Section 5.3. Finally, Section 5.4 is devoted to the computing times and scalability of the proposals.

5.1. Description of ArcelorMittal machinery sensor data

Data from one of ArcelorMittal's machinery has been provided directly by the company. Due to the hostile environment where these assets work, they suffer breakdowns relatively often. Some of these failures can be fixed in place, but others keep the machinery stopped for several days. These breakdowns represent a serious delay in their production pipeline.

The data provided is composed by the sensor information of one of their production machines, as well as related information such as failures, contextual information, etc. Each of these sensors correspond to one of the features that composes the dataset. These variables, most of them of a real character, model different properties of the machinery.

This data covers a two year period, with almost 40 millions of observations, each with more than 100 variables. The objective is to detect these failures early enough to minimize the repair periods for these machines. The dataset is available publicly in GitHub.[4]

5.2. Experimental setup

The dataset has been preprocessed, scaling it to zero-one range. Six features have been removed from the total of 112, because they have a constant value.

Ensemble methods (LSCP_BD and XGBOD_BD) require a diverse base detector. For our experiments, we have use LODA_BD as the base detector since it has more diversity than HBOS_BD due to the use of random projections. Since XGBOD_BD is a semi-supervised method, the training data needs to be labeled. For this task, we selected an interval of [1, 2, 4, 6] hours before a failure and labeled it as class 1 (anomaly). The rest is labeled as class 0 (normal).

Performance is evaluated using the ROC-AUC metric. This metric has been widely employed in anomaly detection research [1,24,49,33]. This makes the results more easily comparable with other studies. Since we are in an unsupervised scenario, algorithms have been evaluated using a ground-truth that is unknown for them. The criteria followed for assessing the performance of the methods in terms of ROC-AUC was to detect anomalies in a fixed period of 1 to 48 hours before a failure. For calculating the ROC-AUC, we have employed the evaluation framework proposed in [7].

In Table 1 we can see the complete list of default parameters for the proposed distributed anomaly detection algorithms. Anomaly detection problems are very data-dependent. For this reason, a set of fixed parameters can hardly be extrapolated to another problem. In order to evaluate the performance of each of the proposed methods, the right combination of parameters must be found. For this task we have performed an hyper-parameter optimization using an hyper-parameter optimization framework named Optuna [3]. This framework allows the user to construct the parameter search space for the hyper-parameters dynamically. As objective value to optimize, we have employed the ROC-AUC value.

The experiments have been carried out in a cluster composed of 15 computing nodes and one master node. The computing nodes have the following hardware specs: 2 x Intel Xeon CPU E5-645, 6 cores per processor (12 threads), 2.40 GHz, 4 TB HDD, 64 GB RAM, and Infiniband 40 Gb/s. Regarding software, we have used the following configuration: Apache Hadoop 2.6.0, Apache Spark 3.0.1 with 360 cores (24 cores/node) and 780 GB RAM (52 GB/node) available.

5.3. Results and analysis

In this section, we present the results of HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD over the real-world case of study. These results have been evaluated using a ground-truth that is unknown for the algorithms. We have selected several fixed window sizes, ranging from 1 hour to 48 hours previous to a failure. Since we are in a unsupervised scenario, a low ROC-AUC value can be expected.

[4] Time-series industrial anomaly dataset https://github.com/ari-dasci/OD-TINA.

Table 1
Default parameter setting for the anomaly detectors.

Algorithm	Parameters
HBOS_BD	n_bins = 100, strategy = "static"
LODA_BD	n_bins = 100, k = 100
LSCP_BD	detector = "LODA_BD", n_base_detectors = 10, pgt_strategy = "avg", clus_method = "kmeans", n_clus = 19, max_size = 10,000, min_size = 1,000, dcs = 0.5
XGBOD_BD	detector = "LODA_BD", n_TOS = 10, n_selected_TOS = 5, TOS_strategy = "acc", threshold = 0.1

Table 2
ROC-AUC value for each AnomalyDSD algorithm and number of hours previous to a failure. The highest ROC-AUC value per hour is stressed in bold.

Previous hours	HBOS_BD		LODA_BD	LSCP_BD	XGBOD_BD
	Static	Dynamic			
1	0.6322	**0.6483**	0.6206	0.6367	0.5019
2	0.6352	**0.6571**	0.6194	0.6370	0.5166
3	0.6366	**0.6614**	0.6214	0.6382	0.5069
4	0.6411	**0.6635**	0.6267	0.6413	0.5040
5	0.6438	**0.6636**	0.6268	0.6406	0.5054
6	0.6482	**0.6707**	0.6287	0.6428	0.5053
12	0.6772	**0.6914**	0.6525	0.6672	0.5288
18	0.7010	**0.7159**	0.6607	0.6813	0.5140
24	0.7314	**0.7442**	0.7001	0.7224	0.5060
36	0.7772	**0.7869**	0.7275	0.7522	0.5519
48	0.8325	**0.8547**	0.8038	0.8291	0.5560

In Table 2 we show the results in terms of ROC-AUC value for the four methods that compose AnomalyDSD. As we can see, except XGBOD_BD, all methods are performing better than a random prediction, and their results are increasingly better as the number of previous hours to a failure is increased. As can be expected, the most restrictive scenario is to detect a failure just one hour before it happens. Since failures may be caused by continuous wear and tear over time, increasing the number of previous hours to a failure also improves the performance of our methods. The dynamic version of HBOS_BD is achieving the best results, followed by its static approach and LSCP_BD. LSCP_BD shows the perfect behavior of an ensemble method, being able to effectively improve the base detector performance (LODA_BD). XGBOD_BD is not being able to effectively detect anomalies in this particular problem. This can be explained by the labeling strategy adding noise to the dataset. In Fig. 1 we show the ROC-AUC value plots for the best result of each method.

In order to verify the performance of our proposal, we have compared HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD against the original versions of HBOS, LODA, LSCP and XGBOD algorithms. For this comparison, we have employed 3 normal-sized public anomaly detection benchmarks from ODDS,[5] namley Breast, Shuttle, and Satellite. We have also selected the two biggest numerical datasets available in [38,39], namely Donors (619,326 instances - 10 features) and Census (299,285 instances - 500 features). Table 3 gathers such results in terms of ROC-AUC metric.

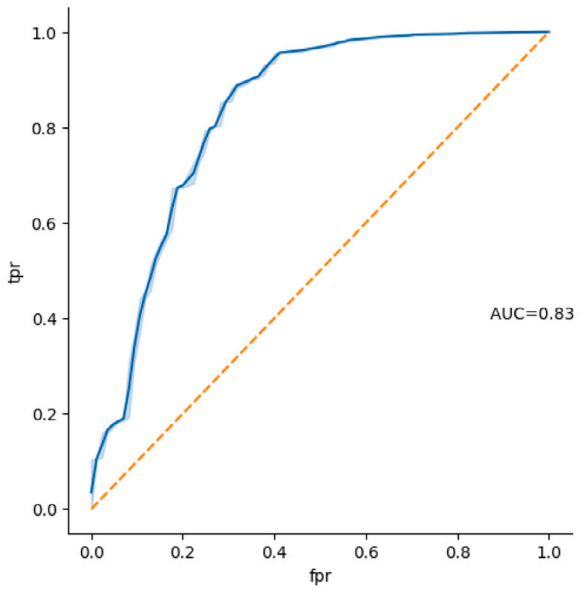
The results presented in Table 3 underscore the effectiveness of the proposed Big Data anomaly detection algorithms in handling large-scale datasets. For smaller datasets (Breast, Shuttle, and Satellite), the classic algorithms performed adequately, with XGBOD even achieving a perfect ROC-AUC score of 1.0 on the Shuttle dataset. The distributed designs of AnomalyDSD achieve similar performance in terms of ROC-AUC, showing that they are consistent with the behavior of the original algorithms. However, these traditional algorithms encountered significant challenges with larger datasets (Donors and Census), evidenced by incomplete results indicating that the algorithms failed to finish processing within a reasonable timeframe (less than 48 hours).

In stark contrast, AnomalyDSD algorithms demonstrated remarkable performance on these extensive datasets. LODA_BD achieved the highest ROC-AUC score in both Donors and Census dataset. These findings clearly illustrate the superiority of our Big Data adaptations, which not only matched but often exceeded the performance of traditional methods, thereby confirming their suitability and efficiency for Big Data anomaly detection tasks.

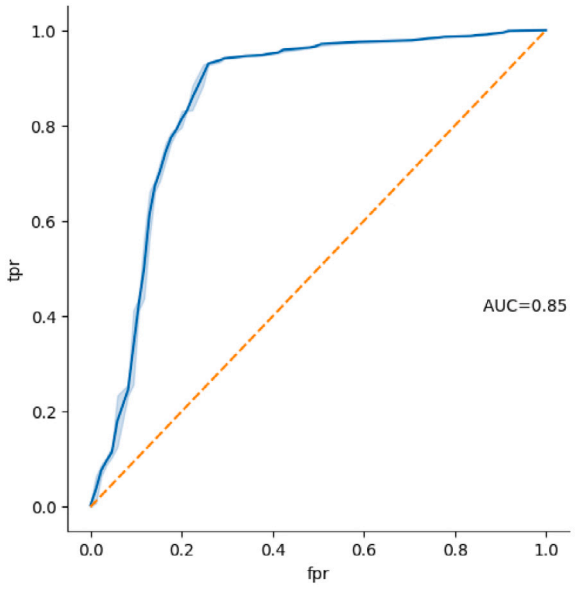
5.4. Efficiency analysis

In the previous section we have shown the suitability of HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD in terms of ROC-AUC value. In order to constitute a valid Big Data proposal, these proposals have to be efficient and scalable as well. In this section we present the computing times and scalability in terms of size of the data, number of threads and workers for the different methods that compose AnomalyDSD using ArcelorMittal's Big Data dataset. For this comparison, default parameters have been selected.

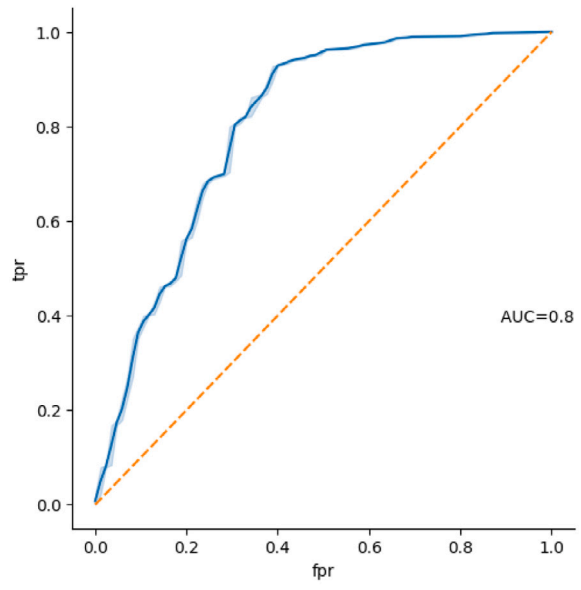
[5] ODDS Library: http://odds.cs.stonybrook.edu/.



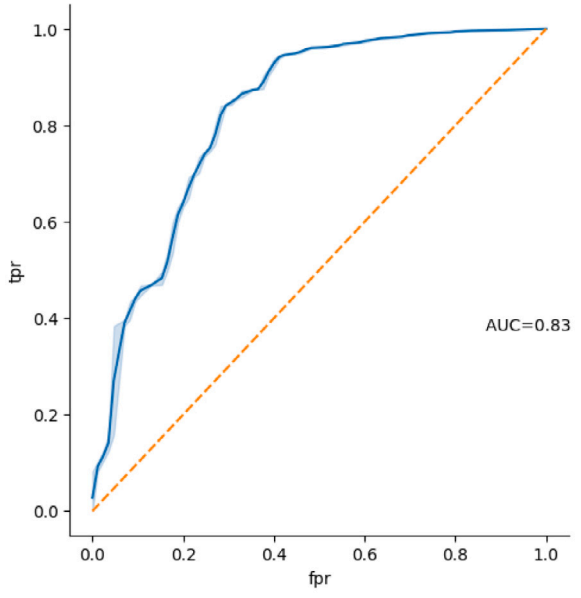
(a) HBOS_BD (static)



(b) HBOS_BD (dynamic)



(c) LODA_BD

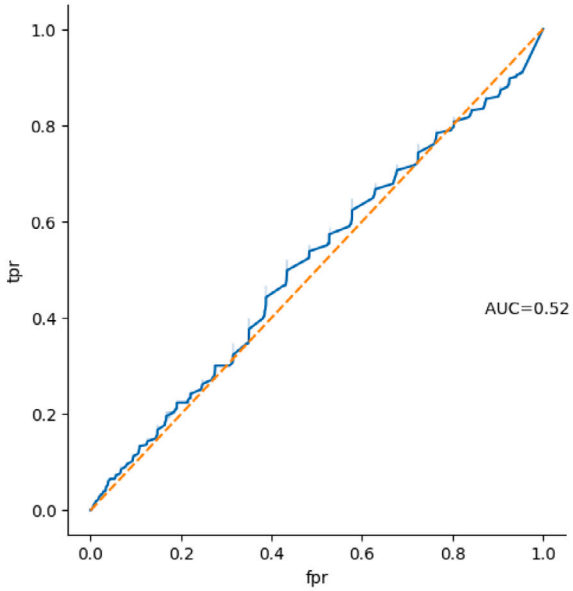


(d) LSCP_BD

Fig. 1. ROC-AUC values plot for HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD with their best results achieved.

In Table 4 we can see the runtimes for the four proposed methods. In order to show the scalability of AnomalyDSD algorithms, we also show the computing times using different samples of the dataset, ranging from 10% to a 100% of the original dataset size. As we can see, all methods are showing a linear time increase with the size of the data. LODA_BD is the best performing method, being able to process a Big Data dataset in less than 93 seconds, followed by the static version of HBOS_BD. Ensemble methods, as well as the dynamic version of HBOS_BD are expected to perform slowly since they are more computationally expensive methods.

For measuring the scalability of AnomalyDSD algorithms, we have run the different methods that compose AnomalyDSD using an increasing number of threads. In Table 5 we show the runtimes for HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD using 16, 32, 64, 128, and 256 threads from the total of 360 available threads. As can be appreciated, from 16 up to 64 threads the runtimes improve almost linearly, being 2x faster each increment of threads. From 64 threads onward the time gain starts decreasing. This can be explained by the difference in terms of physical cores and available threads (2 x cores).



(e) XGBOD_BD

Fig. 1. (*continued*)

Table 3
ROC-AUC value for classic HBOS, LODA, LSCP and XGBOD against each AnomalyDSD algorithm. The highest ROC-AUC value is stressed in bold.

Dataset			Breast	Shuttle	Satellite	Donors	Census
Classic	HBOS		0.9910	0.9855	0.7581	0.7187	0.6333
	LODA		0.9866	0.9920	0.6114	0.4889	0.4449
	LSCP		0.7845	0.5551	0.6106	0.7828	-
	XGBOD		0.9916	**1.0**	**0.9685**	-	-
Big Data	HBOS_BD	Static	0.9853	0.9922	0.8040	0.7364	0.6211
		Dynamic	0.9640	0.5807	0.7043	0.5103	0.5879
	LODA_BD		0.9879	0.9906	0.7291	**0.8261**	**0.6447**
	LSCP_BD		0.9831	0.9891	0.7420	0.8147	0.6128
	XGBOD_BD		**1.0**	**1.0**	0.9394	0.7547	0.6307

Table 4
Computing times with different sizes of data for HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD in seconds.

Algorithm Sample (%)	HBOS_BD		LODA_BD	LSCP_BD	XGBOD_BD
	Static	Dynamic			
10	76.54	1,358.49	47.42	508.82	2,033.77
20	81.65	1,773.09	52.98	613.75	2,098.80
30	97.53	2,290.66	63.45	752.22	2,191.52
40	111.13	2,661.60	68.25	962.91	2,226.92
50	120.37	2,854.53	74.12	1,086.29	2,353.04
60	144.39	3,133.64	72.32	1,151.85	2,454.87
70	161.58	3,457.43	76.68	1,269.82	2,598.17
80	175.49	3,712.69	85.12	1,340.68	2,680.75
90	187.66	4,074.33	87.17	1,455.42	2,787.53
Original	198.30	4,464.62	92.94	1,667.13	2,860.13

Additionally to the number of threads, we have measured the scalability in terms of available workers. In Table 6 we gather the results of HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD using an increasing number of workers. Similarly to the number of threads, the performance increases at a linear pace up to 8 workers. From 10 workers onward the time gain starts becoming less noticeable. Adding more workers not only brings benefits in terms of computing power, it also has the downside of increasing exponentially the number of communications between the workers.

Table 5
Computing times with different number of threads for HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD in seconds.

Threads	HBOS_BD		LODA_BD	LSCP_BD	XGBOD_BD
	Static	Dynamic			
16	836.96	23,896.53	679.13	8,212.83	14,195.39
32	474.50	12,694.02	337.39	4,249.22	7,072.06
64	317.72	7,614,48	201.01	2,397.62	3,883.32
128	222.52	5,073.10	123.03	1,601.19	2,779.84
256	198.76	4,451.82	105.80	1,332.67	2,663.07

Table 6
Computing times with different number of workers for HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD in seconds.

Workers	HBOS_BD		LODA_BD	LSCP_BD	XGBOD_BD
	Static	Dynamic			
4	383.60	13,418.41	273.28	4,864.79	6,525.12
6	315.04	9,229.28	200.83	2,325.94	4,705.57
8	265.03	7,222.37	157.12	2,060.46	3,366.33
10	223.89	6,113.41	138.32	1,651.43	3,175.52
12	208.80	5,370.53	117.99	1,448.69	3,064.11
14	202.80	4,741.01	109.98	1,350.99	2,960.13

5.5. Impact of parameters on ROC-AUC for AnomalyDSD

In this section, we analyze the impact of various parameters on the performance of the different algorithms of AnomalyDSD package, using the ROC-AUC as the performance metric.

HBOS_BD

- **Number of Bins**: A higher number of bins generally increases the resolution of the histogram, which can improve the ability to detect anomalies but may also introduce noise if the number of bins is too high. The best performing range is often between 10 to 30 bins.
- **Strategy**: The dynamic strategy often performs better as it adapts to the data distribution, providing a more nuanced detection capability compared to the static strategy. The dynamic strategy is generally preferred.

LODA_BD

- **Number of Bins**: Similar to HBOS_BD, the choice of the number of bins impacts the granularity of the detection. The best performing range is typically between 10 to 50 bins.
- **Number of Projections**: More projections generally provide better coverage of the data space, improving anomaly detection, but also increase computational complexity. The best performing range is usually between 50 to 200 projections.

LSCP_BD

- **Number of Base Detectors**: More base detectors generally improve detection accuracy by combining multiple perspectives but also increase computational load. The best performing range is typically between 10 to 20 base detectors.
- **Strategy for Pseudo Ground Truth Generation**: The *average* strategy tends to smooth out the influence of individual detectors, while the *maximum* strategy may highlight extreme values. The *average* strategy is often more reliable.
- **Clustering Method**: The choice between *k-Means* and *Bisecting k-Means* affects the local neighborhood calculation. *k-Means* is faster but may be less accurate than *Bisecting k-Means* in some cases. *Bisecting k-Means* often provides better results for more complex data structures.
- **Number of Clusters**: More clusters can capture finer details in the data but may also lead to overfitting if the number is too high. The best performing range is often between 5 to 15 clusters.

XGBOD_BD

- **Number of Base Detectors**: Similarly to LSCP_BD, the more base detectors, the better detection accuracy. The best performing range is typically between 10 to 15 base detectors.

- **Number of Selected Detectors**: Selecting the most adequate base detectors can have a positive impact in the performance, diminishing inaccurate detectors. A good value is 5 detectors.
- **Strategy for the Selection**: The *accuracy* strategy can perform well in certain scenarios, but the *random* strategy will ignore bias towards some detectors. The *random* strategy is often more reliable.
- **Threshold for the Accuracy Strategy**: This threshold only affects the *accuracy* strategy, the *random* strategy is not affected. A value of 0.1 is a good starting point.

Our analysis shows that parameter selection significantly impacts the performance of HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD algorithms. A careful balance is needed to optimize these parameters to achieve the best ROC-AUC results. The identified best performing ranges provide a useful starting point for parameter selection and tuning.

In view of these results we can conclude that:

- HBOS_BD, LODA_BD, and LSCP_BD have been able to effectively detect anomalies in a real-world case of study, as well as in normal-sized benchmark datasets. XGBOD_BD has not been able to achieve good results in the real-world dataset due to its particularities, but it achieved remarkable results in the normal-sized benchmark datasets, showing its superior performance in certain scenarios.
- The dynamic version of HBOS_BD has established as the best performing for the real-world dataset in terms of ROC-AUC value, while the static version achieves excellent results in the normal-sized benchmark datasets.
- LODA_BD has shown to be the most efficient method in terms of computing time, being able to process a Big Data dataset in less than 93 seconds.
- In our cluster, AnomalyDSD algorithms scale linearly up to 8 workers and 64 threads. From there on, the time reduction scaling becomes less noticeable.
- As we could expect, ensemble methods (LSCP_BD and XGBOD_BD) and the dynamic version of HBOS_BD are the most computationally expensive methods, but they are able to achieve better performance than their base algorithms.
- Parameter selection has a significant impact in the performance of anomaly detection methods. An optimization strategy per problem is recommended.

6. Conclusions

This work presents the first suitable distributed designs for Big Data anomaly detection in the form of a package, called AnomalyDSD, where the size of available data and high dimensional problems, pose new challenges to traditional anomaly detection methods. We have presented four different distributed algorithms following the MapReduce paradigm, namely HBOS_BD, LODA_BD, LSCP_BD, and XGBOD_BD.

Experimental results using a real-world case of study have shown the validity of our proposals. All methods have been able to effectively tackle a Big Data dataset in a timely manner, achieving excellent ROC-AUC values. These results have been validated by an additional experiment using a benchmark of normal-sized datasets for comparing AnomalyDSD algorithms against the classic versions of HBOS, LODA, LSCP and XGBOD.

Anomaly detection is a crucial problem for many different real-world applications. With this proposal, we have enabled the practitioner to efficiently and effectively detect anomalies in Big Data datasets, where the early detection of an anomaly can lead to a proper and timely decision for both corporations and academia.

Future directions and integration with other methods To further enhance the capabilities of AnomalyDSD, it is valuable to explore how other types of anomaly detection methods can be integrated and cooperate with the proposed framework:

- Nearest Neighbors Based Methods: Integrating these with AnomalyDSD can be beneficial, especially in hybrid approaches where initial outlier scores from AnomalyDSD can be refined using nearest neighbors analysis. This combination can improve the precision of outlier detection in dense datasets where anomalies might be close to normal instances.
- Clustering Based Methods: These methods can complement AnomalyDSD by providing a different perspective on the data structure. For example, the initial results from AnomalyDSD can be clustered, and further analysis can be performed on these clusters to identify outliers within specific regions, thereby enhancing the detection of collective anomalies.
- Statistical Methods: AnomalyDSD can be used to identify potential outliers, which can then be statistically tested to confirm their anomalous nature. This layered approach can reduce false positives and improve overall detection accuracy [34].
- Ensemble Methods: AnomalyDSD itself includes ensemble-based designs like LSCP_BD and XGBOD_BD, which demonstrate the effectiveness of this approach. Other ensemble methods can be integrated with AnomalyDSD to further enhance its performance. For instance, different ensemble techniques can be applied to the outputs of AnomalyDSD algorithms, or AnomalyDSD can serve as one of the components in a larger ensemble framework.

By incorporating these methods, AnomalyDSD can become even more versatile, capable of tackling a wider range of anomaly detection challenges in Big Data environments. The integration of diverse approaches allows for a more comprehensive analysis, leveraging the unique strengths of each method to achieve superior performance.

These future directions highlight the potential for AnomalyDSD to evolve and adapt, ensuring it remains a powerful tool for anomaly detection in increasingly complex Big Data environments.

CRediT authorship contribution statement

Diego García-Gil: Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **David López:** Writing – original draft, Validation, Software. **Daniel Argüelles-Martino:** Validation, Supervision. **Jacinto Carrasco:** Writing – original draft, Supervision, Investigation. **Ignacio Aguilera-Martos:** Writing – original draft, Validation, Resources. **Julián Luengo:** Writing – original draft, Validation, Supervision, Conceptualization. **Francisco Herrera:** Writing – original draft, Validation, Supervision, Investigation, Formal analysis, Conceptualization.

Ethical approval

This article does not contain any studies with human participants performed by any of the authors.

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.

Acknowledgements

This research results from the Strategic Project IAFER-Cib (C074/23), as a result of the collaboration agreement signed between the National Institute of Cybersecurity (INCIBE) and the University of Granada. This initiative is carried out within the framework of the Recovery, Transformation and Resilience Plan funds, financed by the European Union (Next Generation).

Appendix A. Anomaly detection

An anomaly is an observation that is substantially different from the others. Fig. A.2 shows a graphical representation of anomalies in a two-dimensional data set. The clusters C1 and C2 are composed of normal observations since the majority of points correspond to those regions. Observations O1, O2 and the cluster C3 are located in regions that are far away from C1 and C2, therefore, they are considered as anomalies [10].

There are three main types of anomalies [22]:

- Point anomaly: The anomalous instances are isolated. In Fig. A.2, O1 and O2 are point anomalies. This is the most common scenario in anomaly detection.
- Collective anomaly: The anomaly is a combination of various anomalous instances. For example, detecting an intrusion in a network system may involve detecting multiple connection attempts.
- Contextual anomaly: The anomaly instance seems a standard instance because has not abnormal values, but inside a certain context, that value can be an anomaly. For example, if we measure the occupation of a bus in a range of 0% to 100% during a day. The 50% occupation seems to be completely normal, but if that value is given at 08:00 a.m. when people are going to work or school, the estimated occupation should be much higher.

Anomalies should not be confused with noise, even though they are related. Noise has the same behavior described in Fig. A.2, but, contrary to anomalies, noise has no interest to the data analyst. Anomalies are valuable information that must be detected, extracted and analyzed. Noise damages the quality of the data and those observations must be either fixed or removed [35,19,20,14].

As we have mentioned before, anomaly detection refers to the problem of finding patterns that differs significantly from standard observations. Anomaly detection is used in a wide variety of domains such as credit-card fraud detection [28], intrusion detection [30], sensor networks [32], industrial anomalies [31], health care [15], and much more [22]. Due to the variety of domains that involve the anomaly detection problem, and the presence of more and more sensors in all domains, it is gaining an increasing attention nowadays.

The output of an anomaly detection algorithm can be of two types [10]:

- Scores: The algorithm returns an anomaly score for each instance in the test dataset, indicating the instances that are most likely to be anomalous. A strategy must be defined to choose which scores are considered anomalies.
- Labels: The algorithm returns a binary label indicating which instances are anomalies and which are normal.

We can find many different anomaly detection techniques in the literature [10,1], depending on the approach they use they can classified in:

- Extreme-value analysis: It is the most basic form of anomaly detection. It is based on the analysis of 1-dimensional data. These methods assume that a value is anomalous if it is either too large or to small.

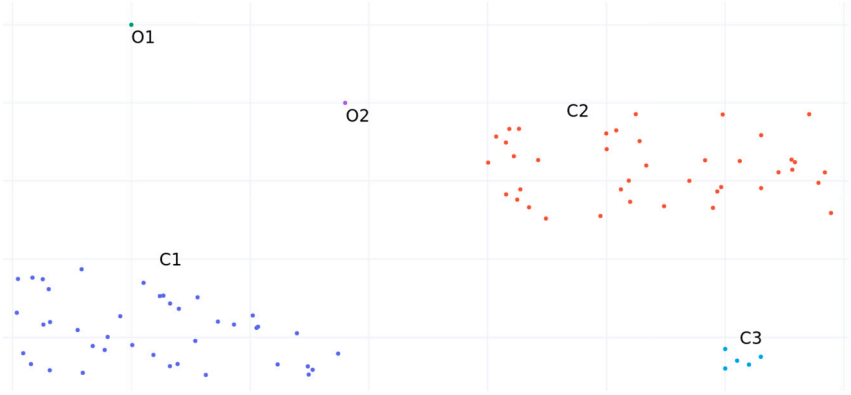


Fig. A.2. Illustration of anomalies in a two-dimensional data set.

- Probabilistic and Statistical models: The data is modeled as a probability distribution. The instances found in higher probability regions are the normal ones and anomalous instances are in lower probability regions [21,40].
- Classification-based: These techniques use a labeled training dataset to build a model and afterwards, the test dataset is classified using the learned model assigning a label to each instance. As examples of these techniques we can find: Deep Learning-based [9], Bayesian Networks-based [44], Support Vector Machines-based [16] and Rule-based [27] methods.
- Nearest Neighbors-based: They assume that the instances within a high-density neighborhood are normal instances, and instances away from those neighborhoods would be anomalies. Two main approaches exist within these techniques: Distance-based methods and Density-based models [6].
- Clustering-based: Assumes that the instances inside a cluster are normal. On the other hand, the instances that are outside a cluster are anomalous [26]. The main difference with respect to nearest neighbors techniques is that clustering methods evaluate each instance according to the cluster it belongs to, while nearest neighbors techniques analyze each instance in its local neighborhood.
- Ensembles: Consists on the combination of multiple diverse base learning algorithms to obtain a global model that improves the base detectors [40,48,47].

There are plenty of applications in the domain of anomaly detection. The intrusion detection consists in the detection of anomalous activity in a computer network [30]. This problem is characterized by the large amount of information flow that can lead to a high number of false alarms rate. The fraud detection refers to find unusual movements related to crimes in commercial applications such as credit cards, phone companies, banks, etc [28]. Those unusual movements are related to identity thief or theft attempts by a consumer. Other applications not related to frauds such as medical health consist in the detection of anomalies in patient measurements, which can be produced by some disease of the patient, instrumentation errors or recording errors [15]. Also, anomaly detection can be applied in the world of the industry, detecting unexpected behavior of the engines in a assembly line, engine sensor instrumentation errors or some damage in structures [31]. Other applications of interest are image processing [9], anomaly detection in text data or anomaly detection in sensor network which refers to detection of anomalies in the data collection that may signify intrusions or errors in the sensors [32].

Appendix B. Big Data & MapReduce

MapReduce is the most popular and widely used paradigm for Big Data processing nowadays. The MapReduce paradigm can process and generate large amounts of data in a distributed and efficient fashion, also it minimizes disk access and network use [13].

It has two phases: the map and the reduce phase. First, the master node partitions the data and distributes it across the cluster. The map function applies a transformation operation to the local key-value pairs of each computing node. In other words, each computing node process a certain part of the algorithm for a certain subset of the data. After the map phase, all pairs with the same key are redistributed. When all pairs with the same key are in the same computing node, the reduce phase starts. The reduce phase is a summary operation that joins all the results from the different map phases into the final values.

Apache Spark [29] is an open-source framework for Big Data, focused on speed, ease of use and sophisticated analytics. Spark allows to persist the data in memory for consecutive or iterative processing, improving the performance greatly. Spark is built on top of Resilient Distributed Datasets (RDDs), which are unsorted and immutable by nature. They allow programmers to persist them in memory, and they are tracked using a lineage so that each partition can be recomputed in case of failure. RDDs support two types of operations: transformations, which are lazily evaluated and produce a new RDD, and actions, which triggers all previous transformations and return a value.

Data availability

Data will be made available on request.

References

[1] C.C. Aggarwal, Outlier Analysis, 2nd ed., Springer Publishing Company, Incorporated, 2016.
[2] I. Aguilera-Martos, M. García-Barzana, D. García-Gil, J. Carrasco, D. López, J. Luengo, F. Herrera, Multi-step histogram based outlier scores for unsupervised anomaly detection: arcelormittal engineering dataset case of study, Neurocomputing 544 (2023) 126228.
[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: a next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.
[4] R.A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, M.A. Amanullah, I. Abaker Targio Hashem, E. Ahmed, M. Imran, Clustering-based real-time anomaly detection—a breakthrough in big data technologies, Trans. Emerg. Telecommun. Technol. 33 (2022) e3647.
[5] T. Arjunan, Real-time detection of network traffic anomalies in big data environments using deep learning models, Int. J. Res. Appl. Sci. Eng. Technol. 12 (2024) 10–22214.
[6] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, Lof: identifying density-based local outliers, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 93–104.
[7] J. Carrasco, D. López, I. Aguilera-Martos, D. García-Gil, I. Markova, M. Garcia-Barzana, M. Arias-Rodil, J. Luengo, F. Herrera, Anomaly detection in predictive maintenance: a new evaluation framework for temporal unsupervised anomaly detection algorithms, Neurocomputing 462 (2021) 440–452.
[8] C. Cavallaro, V. Cutello, M. Pavone, F. Zito, Discovering anomalies in big data: a review focused on the application of metaheuristics and machine learning techniques, Front. Big Data 6 (2023) 1179625.
[9] R. Chalapathy, S. Chawla, Deep learning for anomaly detection: a survey, arXiv preprint arXiv:1901.03407, 2019.
[10] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. (CSUR) 41 (2009) 1–58.
[11] T. Chen, C. Guestrin, Xgboost: a scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '16, ACM, New York, NY, USA, 2016, pp. 785–794.
[12] Z. Chen, Z. Li, X. Chen, X. Chen, H. Fan, R. Hu, Rectifying inaccurate unsupervised learning for robust time series anomaly detection, Inf. Sci. (2024) 120222.
[13] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: OSDI'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation, USENIX Association, 2004.
[14] W. Dong, M. Woźniak, J. Wu, W. Li, Z. Bai, Denoising aggregation of graph neural networks by using principal component analysis, IEEE Trans. Ind. Inform. 19 (2022) 2385–2394.
[15] R.K. Dwivedi, R. Kumar, R. Buyya, A novel machine learning-based approach for outlier detection in smart healthcare sensor clouds, Int. J. Healthc. Inf. Syst. Inf. (IJHISI) 16 (2021) 1–26.
[16] S.M. Erfani, S. Rajasegarar, S. Karunasekera, C. Leckie, High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning, Pattern Recognit. 58 (2016) 121–134.
[17] L. Erhan, M. Ndubuaku, M. Di Mauro, W. Song, M. Chen, G. Fortino, O. Bagdasar, A. Liotta, Smart anomaly detection in sensor systems: a multi-perspective review, Inf. Fusion 67 (2021) 64–79.
[18] D. García-Gil, S. García, N. Xiong, F. Herrera, Smart data driven decision trees ensemble methodology for imbalanced big data, Cogn. Comput. (2024) 1–17.
[19] D. García-Gil, J. Luengo, S. García, F. Herrera, Enabling smart data: noise filtering in big data classification, Inf. Sci. 479 (2019) 135–152.
[20] D. García-Gil, F. Luque-Sánchez, J. Luengo, S. García, F. Herrera, From big to smart data: iterative ensemble filter for noise filtering in big data classification, Int. J. Intell. Syst. 34 (2019) 3260–3274.
[21] M. Goldstein, A. Dengel, Histogram-based outlier score (hbos): a fast unsupervised anomaly detection algorithm, in: KI-2012: Poster and Demo Track, 2012, pp. 59–63.
[22] M. Goldstein, S. Uchida, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, PLoS ONE 11 (2016) e0152173.
[23] R.A.A. Habeeb, F. Nasaruddin, A. Gani, I.A.T. Hashem, E. Ahmed, M. Imran, Real-time big data processing for anomaly detection: a survey, Int. J. Inf. Manag. 45 (2019) 289–307.
[24] S. Han, X. Hu, H. Huang, M. Jiang, Y. Zhao, Adbench: anomaly detection benchmark, Adv. Neural Inf. Process. Syst. 35 (2022) 32142–32159.
[25] J.A. Hanley, B.J. McNeil, The meaning and use of the area under a receiver operating characteristic (roc) curve, Radiology 143 (1982) 29–36.
[26] Z. He, X. Xu, S. Deng, Discovering cluster-based local outliers, Pattern Recognit. Lett. 24 (2003) 1641–1650.
[27] S. Hela, B. Amel, R. Badran, Early anomaly detection in smart home: a causal association rule-based approach, Artif. Intell. Med. 91 (2018) 57–71.
[28] W. Hilal, S.A. Gadsden, J. Yawney, Financial fraud: a review of anomaly detection techniques and recent advances, Expert Syst. Appl. 193 (2022) 116429.
[29] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, Learning Spark: Lightning-Fast Big Data Analysis, O'Reilly Media, Inc., 2015.
[30] I.F. Kilincer, F. Ertam, A. Sengur, Machine learning methods for cyber security intrusion detection: datasets and comparative study, Comput. Netw. 188 (2021) 107840.
[31] B. Kim, M.A. Alawami, E. Kim, S. Oh, J. Park, H. Kim, A comparative study of time series anomaly detection models for industrial control systems, Sensors 23 (2023) 1310.
[32] I. Kraljevski, F. Duckhorn, C. Tschöpe, M. Wolff, Machine learning for anomaly assessment in sensor networks for ndt in aerospace, IEEE Sens. J. 21 (2021) 11000–11008.
[33] M.T.R. Laskar, J.X. Huang, V. Smetana, C. Stewart, K. Pouw, A. An, S. Chan, L. Liu, Extending isolation forest for anomaly detection in big data via k-means, ACM Trans. Cyber-Phys. Syst. 5 (2021).
[34] D. López, I. Aguilera-Martos, M. García-Barzana, F. Herrera, D. García-Gil, J. Luengo, Fusing anomaly detection with false positive mitigation methodology for predictive maintenance under multivariate time series, Inf. Fusion 100 (2023) 101957.
[35] J. Luengo, D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, Big Data Preprocessing - Enabling Smart Data, Springer, 2020.
[36] A.B. Nassif, M.A. Talib, Q. Nasir, F.M. Dakalbab, Machine learning for anomaly detection: a systematic review, IEEE Access 9 (2021) 78658–78700.
[37] S.-V. Oprea, A. Bâra, F.C. Puican, I.C. Radu, Anomaly detection with machine learning algorithms and big data in electricity consumption, Sustainability 13 (2021).
[38] G. Pang, C. Shen, L. Cao, A.V.D. Hengel, Deep learning for anomaly detection: a review, ACM Comput. Surv. (CSUR) 54 (2021) 1–38.
[39] G. Pang, C. Shen, A. van den Hengel, Deep anomaly detection with deviation networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 353–362.
[40] T. Pevnỳ, Loda: lightweight on-line detector of anomalies, Mach. Learn. 102 (2016) 275–304.
[41] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera, Big data: tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce, Inf. Fusion 42 (2018) 51–61.
[42] M.M. Rathore, A. Ahmad, A. Paul, Real time intrusion detection system for ultra-high-speed big data environments, J. Supercomput. 72 (2016) 3489–3510.
[43] L. Rettig, M. Khayati, P. Cudré-Mauroux, M. Piórkowski, Online anomaly detection over big data streams, in: Applied Data Science, Springer, 2019, pp. 289–312.
[44] E. Roberts, B.A. Bassett, M. Lochner, Bayesian anomaly detection and classification for noisy data, in: A. Abraham, P. Siarry, K. Ma, A. Kaklauskas (Eds.), Intelligent Systems Design and Applications, Springer International Publishing, Cham, 2021, pp. 426–435.
[45] S. Thudumu, P. Branch, J. Jin, J.j. Singh, A comprehensive survey of anomaly detection techniques for high dimensional big data, J. Big Data 7 (2020) 1–30.
[46] M. Woźniak, M. Wieczorek, J. Siłka, Bilstm deep neural network model for imbalanced medical data of iot systems, Future Gener. Comput. Syst. 141 (2023) 489–499.

[47] Y. Zhao, M.K. Hryniewicki, Xgbod: improving supervised outlier detection with unsupervised representation learning, in: In 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, 2018, pp. 1–8.
[48] Y. Zhao, Z. Nasrullah, M.K. Hryniewicki, Z. Li, LSCP: locally selective combination in parallel outlier ensembles, in: Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019, SIAM, Calgary, Canada, 2019, pp. 585–593.
[49] Y. Zhao, Z. Nasrullah, Z. Li, PyOD: a Python toolbox for scalable outlier detection, J. Mach. Learn. Res. 20 (2019) 1–7.