Online Isolation Forest

Filippo Leveni [1] (✉) **Guilherme Weigert Cassales** [2] **Bernhard Pfahringer** [2] **Albert Bifet** [2] **Giacomo Boracchi** [1]

Abstract

The anomaly detection literature is abundant with offline methods, which require repeated access to data in memory, and impose impractical assumptions when applied to a streaming context. Existing online anomaly detection methods also generally fail to address these constraints, resorting to periodic retraining to adapt to the online context. We propose ONLINE-IFOREST, a novel method explicitly designed for streaming conditions that seamlessly tracks the data generating process as it evolves over time. Experimental validation on real-world datasets demonstrated that ONLINE-IFOREST is on par with online alternatives and closely rivals state-of-the-art offline anomaly detection techniques that undergo periodic retraining. Notably, ONLINE-IFOREST consistently outperforms all competitors in terms of efficiency, making it a promising solution in applications where fast identification of anomalies is of primary importance such as cybersecurity, fraud and fault detection.

1. Introduction

Anomaly detection deals with the problem of identifying data that do not conform to an expected behavior (Chandola et al., 2009). This task finds numerous applications ranging from the financial fraud (Ahmed et al., 2016; Dal Pozzolo et al., 2018) and intrusion (Bronte et al., 2016) detection, to health (Banaee et al., 2013) and quality monitoring (Stojanovic et al., 2016), to name a few examples. Usually, anomaly detection methods are trained offline with a static dataset and then used to classify new arriving instances. *Isolation Forest* (Liu et al., 2008; 2012) (IFOR) is perhaps the most popular offline anomaly detection solution, which is at the same time simple, easy to scale and shows strong performance on a variety of benchmarks.

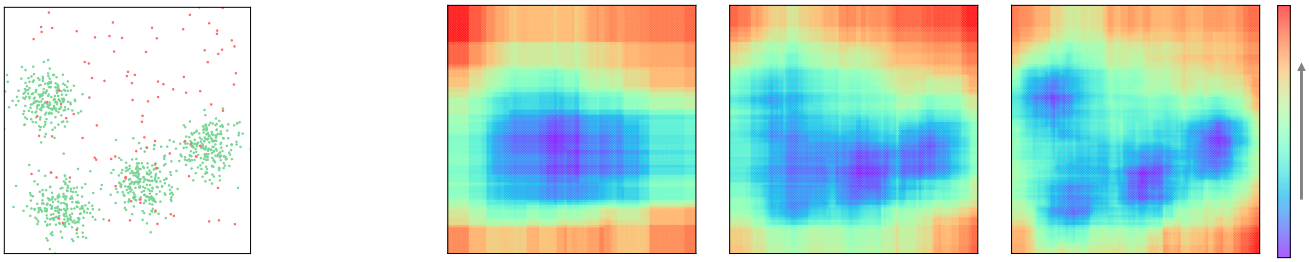
The popularity of IFOR can be acknowledged by the large number of subsequent works that build on top of it. The majority of these methods aim to improve the performance of IFOR by addressing the limitation of axis-parallel splits (Liu et al., 2010; Hariri et al., 2021; Lesouple et al., 2021; Xu et al., 2023), while others extend IFOR beyond the concept of point-anomaly, to identify functional-anomalies (Staerman et al., 2019) and structured-anomalies (Leveni et al., 2021; 2023). Despite their performance, IFOR and its variants are offline anomaly detection methods, and they are unable to operate in a streaming scenario where data comes in endless streams possibly following a dynamic nature. In the online context, offline anomaly detection solutions fail because the models would quickly become outdated and lose performance. The dynamic nature of online environments demands continuous adaptability, making offline approaches unsuitable for effective anomaly detection in the long run.

Online anomaly detection is far less explored compared to online classification (Gomes et al., 2017; 2019), and most of the solutions present in the literature are just online adaptations of offline anomaly detection methods carried out by periodic retraining. Although the retraining approach represents an improvement over the static counterpart, it leads to a substantial processing overhead. Furthermore, online anomaly detection algorithms have to cope with a potentially endless stream of data, strict memory limitations and the single-pass requirement, where it is not possible to store all data for later analysis. Thus, it is essential to have a fast and truly online anomaly detection approach.

We propose ONLINE-IFOREST, an anomaly detection method tailored for the streaming scenario, which can effectively handle the incremental nature of data streams. ONLINE-IFOREST models the data distribution via an ensemble of multi-resolution histograms endowed with a dynamic mechanism to learn new points and forget old ones. Each histogram collects the points count within its bins and, upon reaching a maximum height, a bin undergoes a split to increase the resolution of the histogram in the most populated regions of the space. Conversely, bins in sparsely populated

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy [2]Artificial Intelligence Institute, University of Waikato, Hamilton, New Zealand. Correspondence to: Filippo Leveni <filippo.leveni@polimi.it>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.



(a) Data stream $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t \in \mathbb{R}^d$.

(b) Anomaly scores s at different time instants t, from left to right.

Figure 1. ONLINE-IFOREST dynamically adapts to the data distribution of the stream and improves the anomaly scores estimate over time.

regions are eventually aggregated, thereby decreasing the histogram resolution in the corresponding regions of the space.

In Figure 1 we illustrate the online learning capabilities of our method with a toy example. Genuine data, depicted in green, are more densely distributed than anomalous one, represented in red. ONLINE-IFOREST processes points in Figure 1a one at a time (i.e., in a streaming fashion), and assigns an anomaly score to each of them. As the stream continues, ONLINE-IFOREST acquires more information about the data distribution and refines the estimate of the anomaly scores accordingly (Figure 1b).

Our experiments demonstrate that ONLINE-IFOREST features an exceptionally fast operational speed, addressing the high-speed demands of a streaming context, and achieves effectiveness on par with state-of-the-art online anomaly detection techniques. These properties make ONLINE-IFOREST a promising solution for streaming applications. The code of our method is publicly available at `https://github.com/ineveLoppiliF/Online-Isolation-Forest`.

2. Problem Formulation

We address the online anomaly detection problem in a virtually unlimited multivariate data stream $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t \in \mathbb{R}^d$, where $t \geq 1$. We assume that each $\boldsymbol{x}_i$ is a realization of an independently and identically distributed (i.i.d.) random variable having unknown distribution either $\boldsymbol{X}_i \sim \Phi_0$ or $\boldsymbol{X}_i \sim \Phi_1$, where Φ_0 is the distribution of genuine data and Φ_1 is the distribution of anomalous data. Given a point $\boldsymbol{x}_t$, the goal is to identify whether $\boldsymbol{X}_t \sim \Phi_0$ or $\boldsymbol{X}_t \sim \Phi_1$ for each time instant t.

We assume that anomalous data are *"few"* and *"different"* (Liu et al., 2008) and express these assumptions in the following way: (i) *"few"* – the probability that a point $\boldsymbol{x}_i$ has been generated by the distribution Φ_1 of anomalous data is much lower than the probability that it has been generated by the distribution Φ_0 of genuine data, i.e., $P(\boldsymbol{X}_i \sim \Phi_1) \ll P(\boldsymbol{X}_i \sim \Phi_0)$, and (ii) *"different"* – the probability that a point $\boldsymbol{x}_i$ is closer to an anomalous point $\boldsymbol{x}_j$ rather than a genuine point $\boldsymbol{x}_k$ is low. As a consequence, our focus is on scenarios where anomalous data do not form dense and populous clusters.

Furthermore, we assume that we can store in memory only a finite and small subset $\boldsymbol{x}_{t-\omega}, \ldots, \boldsymbol{x}_t$ of size ω from the entire data stream at each time instant t, were ω is small enough. Additionally, we require the time interval between the acquisition of a sample $\boldsymbol{x}_t$ and its classification to be as small as possible.

3. Related Work

Among the wide literature on anomaly detection (Chandola et al., 2009), we focus on tree-based methods, as they achieve state-of-the-art performance at low computational and memory requirements. IFOR (Liu et al., 2008; 2012) introduced the concept of *"isolation"* as a criteria to categorize anomaly detection algorithms, and subsequent works showed that it is strongly related to the concepts of both distance and density (Zhang et al., 2017; Leveni et al., 2023).

The core component of IFOR is an ensemble of random trees constructed through an iterative branching process. Each individual tree is built by randomly selecting a data dimension and a split value within the bounding box containing data points in that dimension. Anomalous data are identified via an anomaly score computed on the basis of the average path length from the root node to the leaf node, under the assumption that anomalies are easier to isolate.

The most straightforward extension of IFOR to the streaming scenario is *Isolation Forest ASD* (Ding & Fei, 2013) (asdIFOR), which periodically trains from scratch a new IFOR ensemble on the most recent data and discards the old ensemble. The periodic retraining delays the adaptation to new data and, depending on how frequently it is performed, slows down the execution. In contrast, ONLINE-IFOREST seamlessly updates its internal structure at each new sample processed, enabling fast and truly online

anomaly detection. *Robust Random Cut Forest* (Guha et al., 2016) (RRCF) represents the first attempt to adapt IFOR to the streaming context. RRCF dynamically manages tree structures, and introduces a novel anomaly score based on the discrepancy in tree complexity when a data point is removed from the stream. The anomaly score grounds on the assumption that anomalies' impact on tree structures is more evident compared to genuine data points. However, tree modifications performed by RRCF tend to be resource-intensive compared to the fast bin splitting and aggregation of ONLINE-IFOREST. *LODA* (Pevnỳ, 2016) leverages on the Johnson–Lindenstrauss (Johnson & Lindenstrauss, 1984) lemma to project data onto 1-dimensional spaces and subsequently model data distributions via 1-dimensional histograms. LODA makes use of fixed resolution histograms, which make it ineffective in describing multi-modal and complex data arrangements. Conversely ONLINE-IFOREST histograms, thanks to their ability to increase resolution, successfully adapt to various data configurations. *Half Space Trees* (Tan et al., 2011) (HST), in contrast to LODA, employs an ensemble of d-dimensional multi resolution histograms. Specifically, HST builds an ensemble of complete binary trees by picking a random dimension and using the mid-point to bisect the space. Under the assumption that anomalous data points lie in sparse regions of the space, authors of HST propose a score based on node masses. Similarly to asdIFOR, HST suffers of periodic retraining. Additionally, the bins in HST histograms are generated without data, leading to high resolution in empty regions of the space and subsequent memory inefficiency. On the other hand, the data-dependent histograms of ONLINE-IFOREST allows for a more detailed description of data distribution in the most populous regions of the space.

4. Method

ONLINE-IFOREST, denoted as $\mathcal{F}$, is an ensemble of ONLINE-ITREES $\{T_1, \dots, T_\tau\}$ that we specifically designed to continuously and efficiently learn in streaming manner and adapt to the evolving data distribution inherent in streaming contexts. Each ONLINE-ITREE is a d-dimensional histogram that evolves its bins, both in terms of structure and the associated height, as it collects more information about the unknown distributions Φ_0 and Φ_1 over time. We rely on a sliding buffer $W = [\boldsymbol{x}_{t-\omega}, \dots, \boldsymbol{x}_t]$ containing the ω most recent points and, at each time instant t, we use the most and least recent points $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-\omega}$ from the buffer W to respectively expand and contract the tree accordingly.

ONLINE-IFOREST, detailed in Algorithm 1, initializes the buffer W as an empty list, and each tree $T \in \mathcal{F}$ is initially composed of the root node only. At each time step t, we get a new point $\boldsymbol{x}_t$ from the data stream (line 4), store it in the sliding buffer W, and use it to update every base

Algorithm 1: ONLINE-IFOREST

Input: ω window size, τ - number of trees, η - max leaf samples

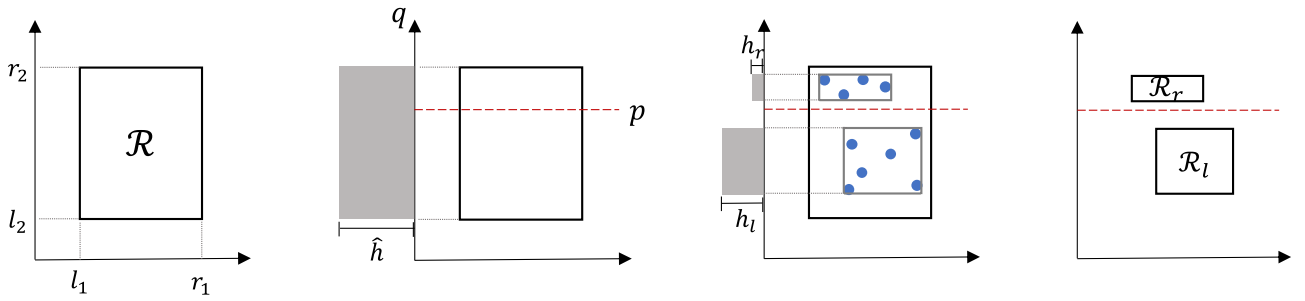
1 initialize W as empty list
2 initialize $\mathcal{F}$ as set of τ empty trees $\{T_1, \dots, T_\tau\}$
3 **while** *true* **do**
4 $\quad \boldsymbol{x}_t \leftarrow$ get point from stream
$\quad$ `/* Update forest */`
5 $\quad$ append $\boldsymbol{x}_t$ to W
6 $\quad$ **for** $i = 1$ to τ **do**
7 $\quad\quad$ `learn_point`$(\boldsymbol{x}_t, T_i.rootN, \eta, \mathtt{c}(\omega, \eta))$
8 $\quad$ **if** length of W greater than ω **then**
9 $\quad\quad \boldsymbol{x}_{t-\omega} \leftarrow$ pop oldest point from W
10 $\quad\quad$ **for** $i = 1$ to τ **do**
11 $\quad\quad\quad$ `forget_point`$(\boldsymbol{x}_{t-\omega}, T_i.rootN, \eta)$
$\quad$ `/* Score point */`
12 $\quad \mathcal{D} \leftarrow \emptyset$
13 $\quad$ **for** $i = 1$ to τ **do**
14 $\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup$ `point_depth`$(\boldsymbol{x}_t, T_i.rootN)$
15 $\quad s_t \leftarrow 2^{-\frac{E(\mathcal{D})}{\mathtt{c}(\omega,\eta)}}$

model T within our ensemble (lines 5-7). A tree T learns the new point $\boldsymbol{x}_t$ by updating each bin along the path from the root to the corresponding leaf, and potentially expanding its tree structure as described by the learning procedure in Algorithm 2. Subsequently, we remove the oldest point $\boldsymbol{x}_{t-\omega}$ from the buffer W and force every tree T to forget it (lines 8-11) via the forgetting procedure that entails updating the bins along the leaf path. The forgetting step, in contrast to the learning one, involves a potential contraction of the tree structure instead of an expansion, as outlined in Algorithm 3. We formalize and detail the learning and forgetting procedures in Section 4.1.

We compute the anomaly score $s_t \in [0, 1]$ for the point $\boldsymbol{x}_t$ (lines 12-15) according to the principles behind IFOR (Liu et al., 2008). In particular, anomalous points are easier to isolate than genuine ones, and therefore they are more likely to be separated early in the recursive splitting process of a random tree. Therefore, we determine the anomaly score by computing the depth of all the leaves where $\boldsymbol{x}_t$ falls in each tree $T \in \mathcal{F}$, and consequently mapping the depths to the corresponding s_t via the following normalization function

$$s_t \leftarrow 2^{-\frac{E(\mathcal{D})}{\mathtt{c}(\omega,\eta)}}, \tag{1}$$

where $E(\mathcal{D})$ is the average depth along all the τ trees of the ensemble, and $\mathtt{c}(\omega, \eta) = \log_2 \frac{\omega}{\eta}$ is an adjustment factor as a function of the window size ω and the number η of points required to perform a bin split. The adjustment factor represents the average depth of an ONLINE-ITREE and we derive it in Section 4.2. The computation of the leaf depth



(a) Bin support $\mathcal{R}$ and its boundaries $[l_i, r_i]$.

(b) Maximum bin height $\widehat{h}$ and split information q and p.

(c) Sampled points $\mathcal{X}$ and new bins height h_l and h_r.

(d) New bins support $\mathcal{R}_l$ and $\mathcal{R}_r$.

Figure 2. Split procedure of a leaf node $N = (h, \mathcal{R})$ in a tree T. (b) As soon as a node N reaches the maximum bin height $\widehat{h}$, we randomly select a dimension q and a split value p. (c) We randomly sample a set $\mathcal{X}$ of $\widehat{h}$ points from the support $\mathcal{R}$ and use them to initialize bins of newborn child nodes $N_l = (h_l, \mathcal{R}_l)$ and $N_r = (h_r, \mathcal{R}_r)$.

where $\boldsymbol{x}_t$ falls into is detailed in Algorithm 4.

4.1. ONLINE-ITREE

The fundamental component of our solution is the ONLINE-ITREE structure. Every ONLINE-ITREE is a d-dimensional histogram constructed by recursively splitting the input space $\mathbb{R}^d$ into bins, such that each bin stores the number of points that fell in the corresponding region of the space. We define ONLINE-ITREE as a dynamic collection of nodes $T = \{N_j\}_{j=1,\dots,m}$ that is continuously updated as new points are learned and old points are forgotten by the tree. We characterize the j-th node as $N_j = (h_j, \mathcal{R}_j)$, where h_j is the number of points that crossed it in their path to the leaf, that is the bin height, and $\mathcal{R}_j = \bigtimes_{i=1}^{d} [l_i, r_i]$ is the minimal d-dimensional hyperrectangle that encloses them, that is the support of the bin, where $\bigtimes$ denotes the Cartesian product. It is worth noting that h_j and $\mathcal{R}_j$ are sufficient for achieving an efficient online adaptation of IFOR.

When a new sample $\boldsymbol{x}_t$ is received from the data stream we run, independently on each ONLINE-ITREE, a learning procedure to update the tree. The learning procedure involves sending the incoming sample $\boldsymbol{x}_t$ to the corresponding leaf, and updating the heights h and supports $\mathcal{R}$ of all the bins along the path accordingly. When a leaf reaches the maximum height $\widehat{h}$, we split the corresponding bin in two according to the procedure illustrated in Figure 2 and described next. This is repeated until the window W gets full, then, together with the learning procedure for the new incoming sample $\boldsymbol{x}_t$, we include a forgetting procedure for the oldest sample $\boldsymbol{x}_{t-\omega}$ in W. The forgetting procedure might involve aggregating nearby bins in a single one as illustrated in Figure 3. The two fold updating mechanism enables ONLINE-ITREE to (i) incrementally learn when the stream starts and (ii) track possible evolution of the stream.

Algorithm 2: ONLINE-ITREE – learn point

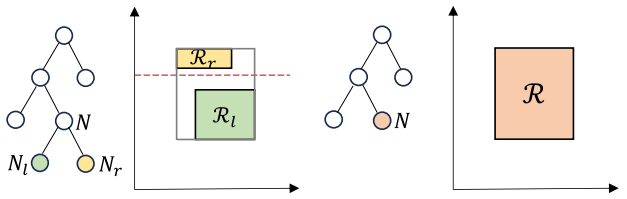
Input: $\boldsymbol{x}$ - input data, N - a tree node, η - max leaf samples, δ - depth limit

Function `learn_point` $(\boldsymbol{x}, N, \eta, \delta)$ **:**
  update bin height h and support $\mathcal{R}$
  if N is a leaf **then**
    if $h \geq \eta\, 2^k$ **and** $k < \delta$ **then**
      $q \leftarrow$ sample from $\mathcal{U}_{\{1,\dots,d\}}$
      $p \leftarrow$ sample from $\mathcal{U}_{[l_q, r_q]}$
      $\mathcal{X} \leftarrow$ sample from $\mathcal{U}_{\mathcal{R}}$
      partition $\mathcal{X}$ into $\mathcal{X}_l, \mathcal{X}_r$
      compute h_l, h_r and $\mathcal{R}_l, \mathcal{R}_r$
      initialize child nodes N_l, N_r
    else
      if $x_q < p$ **then**
        `learn_point`$(\boldsymbol{x}, N_l, \eta, \delta)$
      else
        `learn_point`$(\boldsymbol{x}, N_r, \eta, \delta)$

ONLINE-ITREE bins associated to more populated regions of the space undergo frequent splits, whereas bins associated with sparsely populated regions undergo less frequent splits. Since each split increases the depth of the tree's leaf nodes, we can distinguish between anomalous and genuine points based on the depth k of the leaf nodes they fall into. In Figure 1b we see that an ensemble of ONLINE-ITREEs assigns different anomaly scores to regions with high and low population.

LEARNING PROCEDURE

Every time we feed a point $\boldsymbol{x}_t$ to the tree via the learning procedure, we update both the height h and the support $\mathcal{R}$



(a) Before (b) After

Figure 3. Following the forgetting procedure, support $\mathcal{R}$ of node N is the minimal hyperrectangle that encloses supports $\mathcal{R}_l, \mathcal{R}_r$ of child nodes N_l, N_r.

Algorithm 3: ONLINE-ITREE – forget point

Input: $\boldsymbol{x}$ - input data, N - a tree node, η - max leaf samples

1 **Function** `forget_point` $(\boldsymbol{x}, N, \eta)$ **:**
2 decrease bin height h
3 **if** N is NOT a leaf **then**
4 **if** $h < \eta\, 2^k$ **then**
5 update bin support $\mathcal{R}$ from $\mathcal{R}_l, \mathcal{R}_r$
6 forget split q, p and child nodes N_l, N_r
7 **else**
8 **if** $x_q < p$ **then**
9 `forget_point`$(\boldsymbol{x}, N_l, \eta)$
10 **else**
11 `forget_point`$(\boldsymbol{x}, N_r, \eta)$

of all the nodes crossed by $\boldsymbol{x}_t$ along the path from the root to the leaf (line 2 of Algorithm 2). When the bin height h of a leaf node N reaches a maximum value $\widehat{h}$, we increase the resolution of the histogram in the corresponding region of the space by splitting the associated bin in two, and generating two child nodes N_l and N_r.

We define the maximum height of a bin as $\widehat{h} = \eta\, 2^k$, where η is a user-defined parameter, and k is the depth of the corresponding node $N \in T$. Since the maximum height $\widehat{h}$ grows exponentially as a function of the depth k of the considered node, we increase the number of points required to perform a split in deeper nodes of the tree. This design choice results in compact trees, leading to a substantial efficiency gain. We perform the split procedure only if the depth k of the leaf node N is less than a maximum value $\delta = \log_2 \frac{\omega}{\eta}$, as a function of the window size ω and the number η of points required to split histogram bins. Trees with limited depth had already been discussed in IFOR as a solution to the swamping and masking effects (Murphy, 1951) in anomaly detection.

The split procedure, invoked when the bin height h reaches

Algorithm 4: ONLINE-ITREE – point depth

Input: $\boldsymbol{x}$ - input data, N - a tree node, η - max leaf samples
Output: depth of point $\boldsymbol{x}$

1 **Function** `point_depth` $(\boldsymbol{x}, N)$ **:**
2 **if** N is a leaf **then**
3 **return** $k + \mathrm{c}(h, \eta)$
4 **else**
5 **if** $x_q < p$ **then**
6 **return** `point_depth`$(\boldsymbol{x}, N_l)$
7 **else**
8 **return** `point_depth`$(\boldsymbol{x}, N_r)$

the maximum value $\widehat{h}$, is as follows. First, we randomly sample a split dimension $q \in \{1, \ldots, d\}$ and split value $p \in [l_q, r_q]$ from the random variables $Q \sim \mathcal{U}_{\{1,\ldots,d\}}$ and $P \sim \mathcal{U}_{[l_q, r_q]}$ respectively, where $\mathcal{U}$ denotes the uniform distribution (lines 5-6 of Algorithm 2). Second, we sample a set $\mathcal{X}$ of $\widehat{h}$ points from the support $\mathcal{R} = \bigtimes_{i=1}^{d} [l_i, r_i]$ such that each element $\boldsymbol{x} \in \mathcal{X}$ is distributed according to $\boldsymbol{X} \sim \mathcal{U}_{\mathcal{R}}$ (line 7). The uniform sampling of $\mathcal{X}$ grounds on the approximation of the data distribution by a piece wise uniform distribution represented by the union of histogram leaves' bins. Finally, we partition the elements of $\mathcal{X}$ into $\mathcal{X}_l = \{\boldsymbol{x} \in \mathcal{X} | x_q < p\}$ and $\mathcal{X}_r = \{\boldsymbol{x} \in \mathcal{X} | x_q \geq p\}$, and use them to initialize heights h_l, h_r and supports $\mathcal{R}_l$, $\mathcal{R}_r$ of the newborn left and right child nodes N_l and N_r respectively (lines 8-10). For illustration purposes, we depicted the split procedure in Figure 2 when $d = 2$.

FORGETTING PROCEDURE

In contrast to the learning procedure, which involves creating new nodes and thereby enhancing the histogram resolution in that area, in the forgetting procedure we aggregate nodes and merge the associated bins, ultimately reducing the number of bins and, hence, the histogram resolution in the corresponding region of the space. Specifically, every time we feed a point $\boldsymbol{x}_{t-\omega}$ to the tree via the forgetting procedure, we decrease the bin height h of all the nodes N crossed by it along the path from the root to the leaf (line 2 of Algorithm 3). When the height h of an internal node (i.e., of a node that experienced a split) drops below the threshold $\widehat{h}$, we forget the split in N by merging its two child nodes N_l and N_r. The forget procedure (illustrated in Figure 3) consists in first updating the bin support $\mathcal{R}$ of the node N as the minimal hyperrectangle that encloses bin supports $\mathcal{R}_l$ and $\mathcal{R}_r$ (line 5 of Algorithm 3) of N_l and N_r respectively. Then split information q, p and child nodes N_l, N_r are discarded (line 6).

Table 1. Average and worst case complexity of ONLINE-IFOREST.

COMPLEXITY	AVERAGE CASE	WORST CASE
TIME	$O(n\,\tau\,\log_2\frac{\omega}{\eta})$	$O(n\,\tau\,\log_2\frac{w}{\eta})$
SPACE	$O(\tau\sqrt{\frac{\omega}{\eta}}+\omega)$	$O(\tau\frac{\omega}{\eta}+\omega)$

4.2. Complexity Analysis

The computational complexity is a crucial aspect in the online context, where data streams must be processed at high speed with low memory requirements. Time and space complexities of ONLINE-IFOREST are closely tied to the depth of the ONLINE-ITREEs within the ensemble. Therefore, we first derive ONLINE-ITREE depth in both the average and worst case scenarios, and then express time and space complexities of ONLINE-IFOREST as functions of these depths (Table 1).

Average case To determine the average depth $\bar{k}$ of an ONLINE-ITREE, we first note that a perfectly balanced binary tree constructed with ω points has exactly $\frac{\omega}{2^k}$ points at each node at depth k (Knuth, 2023). Since a node at depth k requires $\eta\,2^k$ points to undergo a split in ONLINE-ITREE, we can state that depth k exists if and only if

$$\frac{\omega}{2^{k-1}} \geq \eta\,2^{k-1}, \tag{2}$$

i.e., if there were enough points at depth $k-1$ to perform the split. Making explicit the inequality with respect to k we have

$$k \leq \frac{1}{2}\,\log_2\frac{\omega}{\eta}+1, \tag{3}$$

from which it follows that the average depth of an ONLINE-ITREE is

$$\bar{k} = \lfloor\frac{1}{2}\,\log_2\frac{\omega}{\eta}+1\rfloor. \tag{4}$$

Hence, the average *time* complexity of ONLINE-IFOREST (i.e., the computational complexity of traversing each tree from the root to a leaf), is $O(n\,\tau\,\log_2\frac{\omega}{\eta})$, where τ it the number ONLINE-ITREEs in the ensemble and n is the number of samples in the data stream. We express the average *space* complexity of ONLINE-IFOREST (i.e., the amount of memory space required) as a function of the number of nodes in an ONLINE-ITREE, that is in turn tied to the average depth $\bar{k}$, plus the buffer size ω. Specifically, we note that the number of nodes in a perfectly balanced binary tree with depth k is $2^{k+1}-1$ (Knuth, 2023). Therefore, the average space complexity of ONLINE-IFOREST is $O(\tau\,2^{\frac{1}{2}\log_2\frac{\omega}{\eta}}+\omega) = O(\tau\sqrt{\frac{\omega}{\eta}}+\omega)$, and it is independent from the number n of samples in the data stream.

Worst case We note that a binary tree degenerated into a linked list, constructed with ω points, has $\omega-k$ points at depth k. Therefore, similarly to the average case, depth k exists if and only if

$$\omega-(k-1) \geq \eta\,2^{k-1}. \tag{5}$$

By making the depth k explicit, and placing an upper bound on it, we have

$$k \leq \log_2\frac{w+1-k}{\eta}+1 \leq \log_2\frac{w+1}{\eta}+1 \tag{6}$$

from which it follows that the worst case depth of an ONLINE-ITREE is

$$\tilde{k} \leq \lfloor\log_2\frac{w+1}{\eta}+1\rfloor. \tag{7}$$

Thus, the worst case *time* and *space* complexities of ONLINE-IFOREST are $O(n\tau\log_2\frac{w}{\eta})$ and $O(\tau 2^{\log_2\frac{w+1}{\eta}}+\omega) = O(\tau\frac{\omega}{\eta}+\omega)$ respectively.

4.3. Adaptation Speed vs. Modeling Accuracy

The length ω of the sliding buffer W plays a crucial role in controlling the trade-off between adaptation speed and modeling accuracy in ONLINE-IFOREST. Specifically, adopting a small ω allows ONLINE-IFOREST to quickly adapt to changes, but it results in a coarse modeling of the underlying data distribution. To this regard, we can observe Figure 1b, illustrating the anomaly scores as ONLINE-IFOREST processes an increasing number of points. Moving from left to right, the anomaly scores describe the learned data distribution after processing 100, 300 and 1000 points, and this is equivalent to what ONLINE-IFOREST would learn over sliding buffers of corresponding lengths. Notably, after processing 100 points, the anomaly scores are coarse, whereas they become more fine-grained after 1000 points.

5. Experiments

In this section we first compare the performance of ONLINE-IFOREST and state-of-the-art methods on a large anomaly detection benchmark where anomalous and genuine distributions Φ_0 and Φ_1 are stationary. Then, we resort to a dataset exhibiting concept drift to assess their capability to adapt to distribution changes.

5.1. Datasets

Stationary We run our experiments on the eight largest datasets used in (Liu et al., 2008; 2012) (Http, Smtp (Yamanishi et al., 2004), Annthyroid, Forest Cover Type, Satellite, Shuttle (Asunction & Newman, 2007), Mammography and Mulcross (Rocke & Woodruff, 1996)), two datasets from Kaggle competitions (Donors and Fraud (Pang et al.,

Table 2. Stationary datasets properties.

DATASET	n	d	% OF ANOMALIES
DONORS	619326	10	5.90
HTTP	567497	3	0.40
FORESTCOVER	286048	10	0.90
FRAUD	284807	29	0.17
MULCROSS	262144	4	10.00
SMTP	95156	3	0.03
SHUTTLE	49097	9	7.00
MAMMOGRAPHY	11183	6	2.00
NYC_TAXI_SHINGLE	10273	48	5.20
ANNTHYROID	6832	6	7.00
SATELLITE	6435	36	32.00

2019)), and the shingled version of NYC Taxicab dataset used in (Guha et al., 2016). We chose these datasets as they contain real data where the genuine and anomalous distributions Φ_0 and Φ_1 are unknown, and contain labels about anomalous data to perform performance evaluation. Information on the cardinality n, dimensionality d, and % of anomalies for the datasets is outlined in Table 2.

Non-stationary We use the INSECTS dataset (Souza et al., 2020) previously used for change detection (Frittoli et al., 2023; Stucchi et al., 2023a;b). INSECTS contains feature vectors ($d = 33$) describing the wing-beat frequency of six (annotated) species of flying insects. This dataset contains 5 real changes caused by temperature modifications that affect the insects' flying behavior. For our purposes, we selected the two most populous classes as genuine ('ae-aegypti-female', 'cx-quinq-male'), and the least populous as the anomalous one ('ae-albopictus-male'). This results in a total of $n = 212514$ points with 5.50% of anomalies.

5.2. Competing methods and methodology

In our experiments we compared ONLINE-IFOREST (oIFOR) to state-of-the-art methods in the online anomaly detection literature described in Section 3. In particular, we compared to *iForestASD* (asdIFOR), *Half Space Trees* (HST), *Robust Random Cut Forest* (RRCF) and *LODA* (LODA) using their *PySAD* (Yilmaz & Kozat, 2020) implementation.

For comparison purposes, we set the number of trees $\tau = 32$ for all the algorithms, and considered the number of random cuts in LODA equivalent to the number of trees. We set window size $\omega = 2048$ for both oIFOR and asdIFOR, and used the default value $\omega = 250$ for HST. We set the subsampling size used to build trees in asdIFOR to the default value $\psi = 256$, while the number of bins for each random projection in LODA to $b = 100$. The trees maximum depth δ depends on the subsamping size ψ in asdIFOR, on the window size ω and number η of points required to split histogram bins in oIFOR, while it is fixed to the default value $\delta = 15$ in HST. The parameters configuration for all the algorithms is illustrated in Table 3.

Table 3. Algorithms execution parameters.

ALGORITHM	τ	ω	η	ψ	δ	b
OIFOR	32	2048	32	–	$\log_2 \frac{\omega}{\eta}$	–
ASDIFOR	32	2048	–	256	$\log_2 \psi$	–
HST	32	250	–	–	15	–
RRCF	32	–	–	256	–	–
LODA	32	–	–	–	–	100

Each algorithm was executed 30 times on both stationary and non-stationary datasets. We randomly shuffled every stationary dataset before each execution, then used the same shuffled version to test all the algorithms. Processing each data point individually is prohibitive in terms of time due to the data stream size. To solve this problem we divided every dataset in batches of 100 points each and passed one chunk at a time to the algorithms in an online manner.

We use the ROC AUC and execution time (in seconds) to evaluate the effectiveness and efficiency of the considered algorithms, respectively. In addition, we employ the critical difference diagram for both metrics to synthesize the results across multiple executions on datasets with diverse characteristics.

5.3. Results

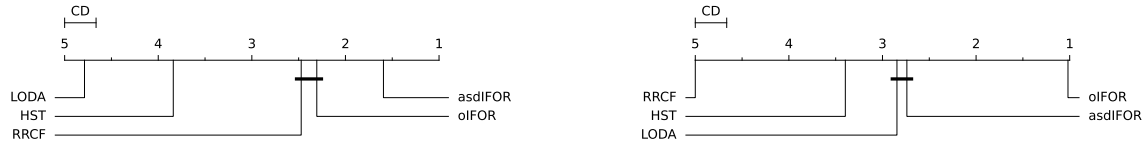
ANOMALY DETECTION IN STATIONARY DATA STREAMS

In Table 4 we show the median ROC AUC for each algorithm after processing each dataset, as well as the median total execution time for each algorithm to process the entire data stream at hand. The last two rows represent the median value and the mean rank among the total 330 executions. We highlight the best row-wise result in bold.

oIFOR exhibits, by far, the lowest time complexity with respect to all the competitors at hand. In particular, when we compare oIFOR to the second best method (asdIFOR), we can notice that oIFOR execution time is less than half in 4 out of 5 biggest datasets, and it is reduced by more than an order of magnitude in 4 out of 6 smallest ones. This result is confirmed by the critical difference diagram presented in Figure 4b, which shows that oIFOR is statistically better than all the others, while LODA and asdIFOR are statistically equivalent. The statistical analysis has been conducted for 5 populations (one for each algorithm) with 330 paired total execution times. Critical difference diagrams are based on the post-hoc Nemenyi test, and differences between populations are significant when the difference of their mean rank is greater than the critical distance $CD = 0.336$. Mean

Table 4. ROC AUCs and total execution times.

	AUC (↑)					TIME (↓)				
	OIFOR	ASDIFOR	HST	RRCF	LODA	OIFOR	ASDIFOR	HST	RRCF	LODA
DONORS	**0.795**	0.769	0.715	0.637	0.554	**252.36**	551.85	2145.85	4924.46	2111.09
HTTP	0.998	**0.999**	0.992	0.996	0.632	**179.36**	509.40	2016.00	8367.16	2017.85
FORESTCOVER	0.887	0.861	0.722	**0.917**	0.500	**107.65**	197.82	1045.39	2997.86	1009.92
FRAUD	0.936	0.946	0.910	**0.951**	0.722	**100.09**	285.69	973.91	4936.03	931.93
MULCROSS	**0.995**	0.952	0.011	0.800	0.506	**90.33**	270.79	936.01	3244.96	848.12
SMTP	0.861	**0.905**	0.851	0.894	0.731	**29.95**	142.65	325.77	1273.98	254.69
SHUTTLE	0.992	**0.996**	0.981	0.957	0.528	**16.35**	108.28	167.48	770.61	130.61
MAMMOGRAPHY	0.854	**0.855**	0.831	0.824	0.622	**3.32**	80.01	37.92	118.22	29.55
NYC_TAXI_SHINGLE	0.572	0.709	0.342	**0.725**	0.499	**8.03**	83.15	36.70	151.67	36.82
ANNTHYROID	0.685	**0.810**	0.636	0.740	0.589	**2.00**	77.06	24.26	93.40	19.79
SATELLITE	0.651	**0.709**	0.531	0.662	0.501	**3.74**	78.90	21.78	93.77	17.55
MEDIAN	**0.866**	0.863	0.739	0.832	0.541	**29.95**	142.57	323.29	1274.45	254.64
MEAN RANK	2.167	**1.583**	3.917	2.500	4.833	**1.000**	2.667	3.500	5.000	2.833



(a) ROC AUCs

(b) Times

Figure 4. Critical difference diagram for ROC AUCs and total execution times.

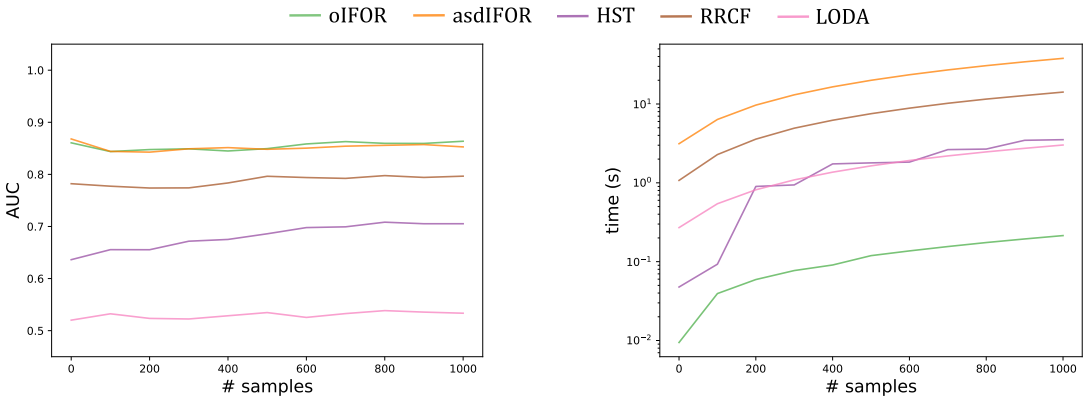
ranks and critical diagrams have been generated via *Autorank* (Herbold, 2020) library. In Figure 6 we sorted the median total execution times listed in Table 4 by dataset size, and we can appreciate the linear trend exhibited by all the algorithms.

Table 4 shows that oIFOR, asdIFOR and RRCF exhibit the best detection performance over different datasets. While oIFOR exhibits a slightly higher overall median ROC AUC, the Nemenyi test highlights that asdIFOR is the most effective algorithm, and that oIFOR and RRCF are statistically equivalent (Figure 4a). The exceptionally low performance of HST on the Mulcross dataset in Table 4 is due to the fact that the size of anomaly clusters is large and that anomaly clusters have an equal or higher density compared to genuine ones in that dataset. This scenario, combined with the high default maximum depth value δ, makes this situation particularly difficult for HST to handle, as it is based on the opposite assumptions.

Learning in the early stages of a data stream In addition to the conventional evaluation of online anomaly detection algorithms based on their final anomaly detection capability, we highlight the initial learning speed demonstrated by various methods. We focused to the early stages of the data stream, aiming to comprehend how the various methods learn in a critical phase such as the initial one, and analyzed their performance within the first 1000 samples of the stream. Solid curves in Figure 5a show the median ROC AUC of each algorithm over all the 30 executions and the 11 datasets, for a total of 330 runs. We computed the ROC AUCs at each time instant $\dot{t}$ using the scores from $t = 1$ to $t = \dot{t}$. All the algorithms show a fast learning speed, since within the first 1000 samples all of them get very close to the final median performance showed in Table 4. The corresponding efficiency is shown in Figure 5b, where all the algorithms exhibit similar trends, with oIFOR being the fastest to adapt.

ANOMALY DETECTION IN NON-STATIONARY DATA STREAMS

In Figure 7 we show the median ROC AUC of each algorithm over 30 executions on the INSECTS dataset. We are interested in investigating the instantaneous anomaly detection performance of all the algorithms, therefore we computed the ROC AUCs at each time instant $\dot{t}$ using the scores within a window of size 5000 centered in $\dot{t}$, i.e., from $t = \dot{t} - 2500$ to $t = \dot{t} + 2500$. The choice of 5000 for window size was made to guarantee that the resulting curves exhibit a satisfactory degree of smoothness. Vertical dotted lines represent the time instants when the change in distri-



(a) ROC AUCs (b) Times

Figure 5. Evolution of the median ROC AUCs and total execution times within the first 1000 samples of the stream.

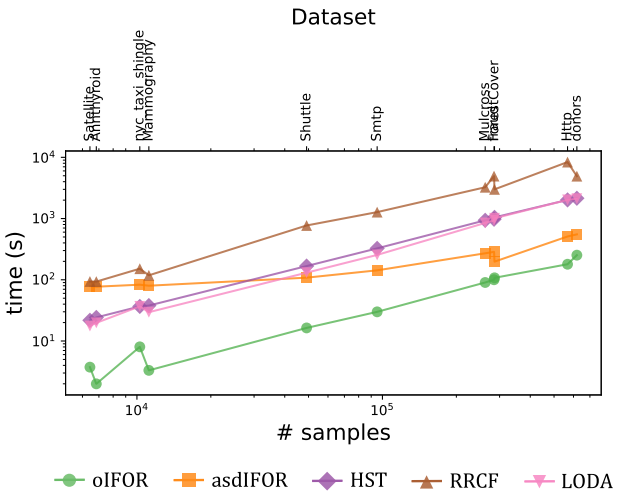


Figure 6. Total execution times ordered by dataset size.

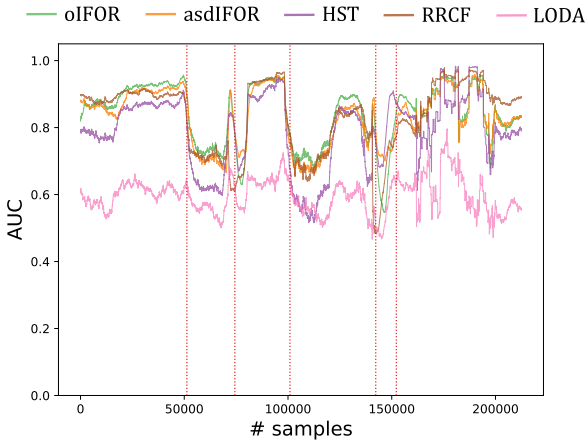


Figure 7. The performance of online anomaly detection methods is significantly influenced by changes in data distributions Φ_0 and Φ_1.

butions occur. Figure 7 shows that all the tested algorithms are affected in a similar way by sudden changes in the genuine and anomalous distributions Φ_0 and Φ_1, and we cannot identify a method that consistently maintains performance after a change. Although LODA is less affected by changes compared to the others, the overall low performance indicates that LODA struggles in learning the underlying distributions. The execution times of the algorithms are not influenced by distribution changes, and ONLINE-IFOREST remains the fastest option.

6. Conclusion and Future Works

In this work we presented ONLINE-IFOREST, an anomaly detection algorithm specifically designed for the streaming scenario. ONLINE-IFOREST is an ensemble of histograms that dynamically adapt to the data distribution keeping only statistics about data points. Thanks to a sliding window, ONLINE-IFOREST is able to selectively forget old data points and update histograms accordingly. Extensive experiments showed that ONLINE-IFOREST features an extremely fast processing and learning speed while maintaining effectiveness comparable to that of state-of-the-art methods. The intuitive operational approach, coupled with its high speed, positions ONLINE-IFOREST as a good candidate for addressing real-world streaming anomaly detection challenges.

As future work we aim to remove the sliding window W while retaining the forgetting capabilities of ONLINE-IFOREST. Additionally, we seek to automate the selection of the number η of points required to split histogram bins.

Acknowledgements

This work was supported by the *"PNRR-PE-AI FAIR"* and the *"AI for Sustainable Port-city logistics (PNNR Grant P2022FLLPY)"* projects, both funded by the NextGeneration EU program.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ahmed, M., Mahmood, A. N., and Islam, M. R. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

Asunction, A. and Newman, D. The uci machine learning repository. https://archive.ics.uci.edu, 2007.

Banaee, H., Ahmed, M. U., and Loutfi, A. Data mining for wearable sensors in health monitoring systems: A review of recent trends and challenges. *Sensors*, 13(12): 17472–17500, 2013.

Bronte, R., Shahriar, H., and Haddad, H. Information theoretic anomaly detection framework for web application. In *Computer Software and Applications Conference (COMPSAC)*, volume 2, pp. 394–399. Institute of Electrical and Electronics Engineers (IEEE), 2016.

Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., and Bontempi, G. Credit card fraud detection: A realistic modeling and a novel learning strategy. *Transactions on Neural Networks and Learning Systems*, 29(8):3784–3797, 2018.

Ding, Z. and Fei, M. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *International Federation of Automatic Control (IFAC)*, 46(20):12–17, 2013.

Frittoli, L., Carrera, D., and Boracchi, G. Nonparametric and online change detection in multivariate datastreams using quanttree. *Transactions on Knowledge and Data Engineering (TKDE)*, 35(8):8328–8342, 2023.

Gomes, H. M., Barddal, J. P., Enembreck, F., and Bifet, A. A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, 50(2):1–36, 2017.

Gomes, H. M., Read, J., Bifet, A., Barddal, J. P., and Gama, J. Machine learning for streaming data: State of the art, challenges, and opportunities. *International Conference on Knowledge Discovery and Data Mining (SIGKDD) Explorations Newsletter*, 21(2):6–22, 2019.

Guha, S., Mishra, N., Roy, G., and Schrijvers, O. Robust random cut forest based anomaly detection on streams. In *International Conference on Machine Learning (ICML)*, volume 48, pp. 2712–2721. Proceedings of Machine Learning Research (PMLR), 2016.

Hariri, S., Kind, M. C., and Brunner, R. J. Extended isolation forest. *Transactions on Knowledge and Data Engineering (TKDE)*, 33(04):1479–1489, 2021.

Herbold, S. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software (JOSS)*, 5(48):2173, 2020.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space. In *Conference on Modern Analysis and Probability*, volume 26, pp. 189–206. American Mathematical Society, 1984.

Knuth, D. E. *The Art of Computer Programming*, volume 1-4B Boxed set. Addison-Wesley Professional, 2023.

Lesouple, J., Baudoin, C., Spigai, M., and Tourneret, J.-Y. Generalized isolation forest for anomaly detection. *Pattern Recognition Letters*, 149:109–119, 2021.

Leveni, F., Magri, L., Boracchi, G., and Alippi, C. Pif: Anomaly detection via preference embedding. In *International Conference on Pattern Recognition (ICPR)*, pp. 8077–8084. Institute of Electrical and Electronics Engineers (IEEE), 2021.

Leveni, F., Magri, L., Alippi, C., and Boracchi, G. Hashing for structure-based anomaly detection. In *International Conference on Image Analysis and Processing (ICIAP)*, pp. 25–36. Springer Nature, 2023.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *International Conference on Data Mining (ICDM)*, pp. 413–422. Institute of Electrical and Electronics Engineers (IEEE), 2008.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. On detecting clustered anomalies using sciforest. In *European Conference on Machine Learning (ECML)*, pp. 274–290. Springer Nature, 2010.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation-based anomaly detection. *Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

Murphy, R. B. *On Tests for Outlying Observations*. Princeton University Press, 1951.

Pang, G., Shen, C., and van den Hengel, A. Deep anomaly detection with deviation networks. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 353–362. Association for Computing Machinery (ACM), 2019.

Pevnỳ, T. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

Rocke, D. M. and Woodruff, D. L. Identification of outliers in multivariate data. *Journal of the American Statistical Association (JASA)*, 91(435):1047–1061, 1996.

Souza, V. M., dos Reis, D. M., Maletzke, A. G., and Batista, G. E. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, 34(6):1805–1858, 2020.

Staerman, G., Mozharovskyi, P., Clémençon, S., and d'Alché Buc, F. Functional isolation forest. In *Asian Conference on Machine Learning (ACML)*, volume 101, pp. 332–347. Proceedings of Machine Learning Research (PMLR), 2019.

Stojanovic, L., Dinic, M., Stojanovic, N., and Stojadinovic, A. Big-data-driven anomaly detection in industry (4.0): An approach and a case study. In *International Conference on Big Data (ICBD)*, pp. 1647–1652. Institute of Electrical and Electronics Engineers (IEEE), 2016.

Stucchi, D., Magri, L., Carrera, D., and Boracchi, G. Multimodal batch-wise change detection. *Transactions on Neural Networks and Learning Systems*, 34(10):6783–6797, 2023a.

Stucchi, D., Rizzo, P., Folloni, N., and Boracchi, G. Kernel quanttree. In *International Conference on Machine Learning (ICML)*, volume 202, pp. 32677–32697. Proceedings of Machine Learning Research (PMLR), 2023b.

Tan, S. C., Ting, K. M., and Liu, T. F. Fast anomaly detection for streaming data. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pp. 1511–1516. Association for the Advancement of Artificial Intelligence (AAAI), 2011.

Xu, H., Pang, G., Wang, Y., and Wang, Y. Deep isolation forest for anomaly detection. *Transactions on Knowledge and Data Engineering (TKDE)*, 35(12):12591–12604, 2023.

Yamanishi, K., Takeuchi, J.-i., Williams, G., and Milne, P. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.

Yilmaz, S. F. and Kozat, S. S. Pysad: A streaming anomaly detection framework in python. *arXiv preprint arXiv:2009.02572*, 2020.

Zhang, X., Dou, W., He, Q., Zhou, R., Leckie, C., Kotagiri, R., and Salcic, Z. Lshiforest: A generic framework for fast tree isolation based ensemble anomaly analysis. In *International Conference on Data Engineering (ICDE)*, pp. 983–994. Institute of Electrical and Electronics Engineers (IEEE), 2017.