

Streaming Isolation Forest

Justin Jia Liu¹[0000–0001–6315–3854]^{*},
Guilherme Weigert Cassales¹[0000–0003–4029–2047],
Fei Tony Liu^{2,3}[0000–0001–9028–2774], Bernhard Pfahringer¹[0000–0002–3732–5787],
and Albert Bifet^{1,4}[0000–0002–8339–7773]

¹ AI Institute, University of Waikato, New Zealand
{justin.l,guilherme.cassales,bernhard,abifet}@waikato.ac.nz

² Artificial General Intelligence Pty Ltd

³ University of New South Wales
fei.t.liu@unsw.edu.au

⁴ LTCI, Télécom Paris, IP Paris, France

Abstract. Anomaly detection is crucial to identify unusual patterns in various domains. In particular, continuous and rapid flow creates distinct challenges within streaming data. This paper introduces the Streaming Isolation Forest (SiForest), a novel algorithm that uses isolation principles and reservoir sampling to align the model with current data distributions. SiForest efficiently detects anomalies with minimal computational and memory requirements and dynamically updates its model using a subtree regrowing strategy. Empirical evaluation on twenty-three benchmark datasets demonstrates that SiForest outperforms eight state-of-the-art algorithms in terms of AUC-ROC scores, achieving greater precision and adaptability.

Keywords: Anomaly detection · Data streams · Ensembles.

1 Introduction

Anomaly detection, also known as outlier detection, plays an essential role in various domains by identifying unusual patterns that do not conform to expected behaviour [3]. This technique is widely used in areas such as fraud detection, network security, fault detection in industrial systems, and health monitoring. With the increasing availability of large volumes of data, the importance of robust and efficient anomaly detection algorithms has grown significantly.

Traditional anomaly detection algorithms face challenges with rapid data streams [1]. Unlike batch processing, stream learning algorithms immediately process each real-time data point, making revisiting past data impractical. These algorithms must be accurate, efficient, and adaptable to concept drift.

This paper introduces Streaming Isolation Forest (SiForest), a novel anomaly detection algorithm for data streams. It employs the isolation principle [11] and leverages reservoir sampling [21] to simulate the uniform sampling in streaming

^{*} Corresponding author

contexts. The algorithm updates trees incrementally, ensuring that they match fully rebuilt trees distributionally, which maintains high accuracy and fast response without total reconstruction. The main contributions of this work are summarised as follows:

- We introduce an innovative anomaly detection algorithm that incorporates the isolation mechanism from iForest with reservoir sampling to facilitate rapid incremental model updates, eliminating the need for frequent full model updates. Consequently, this approach ensures high detection accuracy and quick adaptation to dynamic data streams.
- We show mathematically that the incrementally updated ensemble preserves an equivalent distribution as an ensemble fully reconstructed from the final snapshot of the reservoirs.
- We conducted a comprehensive empirical assessment on twenty-three benchmark datasets, employing SiForest along with eight data stream anomaly detection algorithms. This analysis reveals that SiForest outperforms the others in AUC-ROC performance.

2 Problem Formulation

Anomaly detection in streaming data aims to identify anomalies in a real-time data stream $S = \{x^{(1)}, \dots, x^{(i)}\}$, where $x \in \mathbb{R}^d$. The stream comprises predominantly normal examples $x_n \in S_n$ and a smaller subset of anomalies $x_a \in S_a$, such that $S = S_n \cup S_a$ with $|S_a| \ll |S_n|$. Each incoming data point x must be processed once to assign an anomaly score $\alpha(x)$, where anomalies are expected to score significantly higher ($\alpha(x_a) \gg \alpha(x_n)$).

The challenge lies in separating anomalies from normal data efficiently and accurately in a streaming context, without revisiting past data or requiring assumptions about data distributions.

3 Related Work

Isolation Forest (iForest) [10] is a widely used anomaly detection algorithm known for its scalability and efficiency, which uses random partitions to isolate anomalies with linear-time complexity. However, iForest is static once it has been constructed, making it unsuitable for evolving data streams. To address this, iForestASD (IFASD) [4] adapts iForest using a sliding-window approach, updating the model only when the anomaly rate changes significantly. Although effective in controlled scenarios, its performance suffers from unforeseen rate changes.

Other algorithms designed for streaming contexts include Half-Space Trees (HST) [19], which use random splits and mass-based scoring. HST is built with the assumption that feature ranges are known in advance. Though sometimes effective, HST struggles with feature range determination in data streams. Similarly, xStream [12] and RS-Hash(RSH) [17] provide lightweight, ensemble-based

Algorithm 1 SiForest(S, w, k)**Input:** S - stream, w - reservoir size limit, k - number of trees**Output:** anomaly scores

```

1: collect the  $w$  samples  $D_s$  from  $S$ 
2:  $F \leftarrow \phi$ 
3:  $h \leftarrow \lceil \log_2 w \rceil$ 
4: for  $i = 1$  to  $k$  do
5:    $F \leftarrow F \cup \text{SiTree}(D_s, 0, h)$ 
6: end for
7: while get new data point  $x$  from data stream  $S$  do
8:    $a \leftarrow s(x, w)$  {report  $a$  as anomaly score for  $x$ }
9:   Update( $T, x$ ) for each SiTree  $T$  in SiForest  $F$ 
10: end while

```

approaches for high-dimensional streaming data, offering constant-time processing but often sacrificing accuracy in complex scenarios.

LODA [15] and SDOstream(SDOS) [8] employ ensemble techniques to detect anomalies. LODA uses sparse random projections with histograms for scoring. SDOstream introduces incremental updates and model ageing based on SDO [20], making it suitable for real-time applications.

Online Isolation Forest (oIF) [9] enhances the Isolation Forest by using dynamic multi-resolution histograms to model evolving data distributions. It processes data incrementally, adding new data and removing old data using a sliding window mechanism.

Robust Random Cut Forest (RRCF) [7] adapts iForest for streaming by dynamically adjusting tree structures to reflect changes in data. Although RRCF is effective with high-dimensional data, its biased dimension selection may be suboptimal in certain cases.

Notably, the SiForest and RRCF resemble each other as both maintain random binary trees in a streaming context. However, SiForest does not have the dimension bias found in the RRCF. It is discussed in Section 4.4.

4 Streaming Isolation Forest

SiForest initialises by collecting the first w samples from the data stream to construct k streaming isolation trees (SiTrees). As the data stream evolves, SiForest generates anomaly scores for the examples and dynamically maintains its model using reservoir sampling, guaranteeing that the trees reflect current data distributions without full reconstruction. The general SiForest pseudocode is presented in Algorithm 1. A detailed analysis of the scoring methodology, SiTree maintenance, and computational complexity is provided in the following subsections. We further examine how the bias dimension sampling in RRCF contrasts with SiForest, as detailed in Section 4.4.

Algorithm 2 SiTree(D_s, e, h)**Input:** D_s - input data, e - current tree height, h - height limit**Output:** a Streaming Isolation Tree

```

1: if  $e = h$  or  $|D_s| = 1$  then
2:   return Leaf( $D_s$ )
3: else
4:   let  $Q$  be a list of attributes in  $D_s$ 
5:   randomly select an attribute  $q \in Q$ 
6:   randomly select a split point  $p \in [min_q, max_q]$ 
7:    $D_{s,l} \leftarrow filter(D_s, q < p)$ 
8:    $D_{s,r} \leftarrow filter(D_s, q \geq p)$ 
9:   return Branch( $Left \leftarrow SiTree(D_{s,l}, e + 1, h)$ ,
10:                 $Right \leftarrow SiTree(D_{s,r}, e + 1, h)$ ,
11:                 $SplitAtt \leftarrow q$ ,
12:                 $SplitValue \leftarrow p$ )
13: end if

```

4.1 Anomaly Score

The anomaly score in SiForest follows the intuition that anomalies are more likely to be isolated at earlier levels of the tree when following the building procedure in Alg. 2. It is defined as $s(x, w) = 2^{-\sum_{T \in F} \frac{h_T(x)}{c(w)|F|}}$, where each factor of the sum represents a ratio between the travel path of instance x for a given tree T in the forest F and the expected path length of a binary tree constructed with w samples. This scoring function ensures an accurate scoring in the dense and sparse regions of the data [10]. The travel path is defined as $h_T(x) = |path(T, x)| + c(|leaf(T, x)|)$, which accounts for leaves in the height limit that have not yet been isolated. The expected path length is defined as $c(z) = 2H(z - 1) - (2(z - 1)/z)$, where z is an arbitrary number, and $H(i) = \ln(i) + 0.5772156649$ (Euler constant). Note that a correction factor is applied to the sum with the full size of the reservoir and to the $h_T(x)$ with the leaf mass.

4.2 Tree Maintenance

Initially, each SiTree is developed by selecting a feature at random and choosing a split value within the range of that feature. Leaf nodes are generated by dividing the feature values until either the examples are isolated or the maximum tree height is reached. Algorithm 2 provides a comprehensive description of the methodology employed in constructing SiTree. When a new example is introduced, the tree maintenance process begins. This updating procedure, detailed in Algorithm 3, involves replacing an example in the tree's reservoir and adjusting the tree structure appropriately. This section will detail the procedure.

The core procedure for constructing SiTree(Algorithm 2) on a set of examples D_s , represented as $T(D_s)$, can be formed through the following steps:

1. Select a dimension q at random with uniform probability

Algorithm 3 Update(T, x)**Input:** T - The SiTree, x - an example

```

1: let  $R$  be the reservoir of  $T$ 
2: let  $w$  be the reservoir size limit
3: let  $n$  be the number of examples that have been seen
4: uniformly select an index  $i \in [0, n]$ 
5: if  $i < w$  then
6:    $\hat{x} \leftarrow R[i]$ 
7:    $Remove(T, \hat{x})$  {Remove the old point  $\hat{x}$ }
8:    $Insert(T, x)$  {Insert the new point  $x$ }
9:    $R[i] \leftarrow x$  {Update the reservoir}
10: end if

```

Algorithm 4 Remove(T, x)**Input:** T - The SiTree, x - an example

```

1: for  $v$  in  $Path(x)$  on SiTree  $T$  do
2:   Let  $S_v$  be the set of points at node  $v$ 
3:   Let  $q$  be the splitting dimension at  $v$ 
4:   if  $v$  is a leaf then
5:     Remove  $x$  from the leaf
6:     if the leaf is empty then
7:       Let  $v$  be the parent of the leaf
8:        $v \leftarrow SiTree(S_v \setminus \{x\}, Height(v), HeightLimit(T))$ 
9:       return
10:    end if
11:   else if  $x_q = \min_q(S_v)$  or  $x_q = \max_q(S_v)$  then
12:      $v \leftarrow SiTree(S_v \setminus \{x\}, Height(v), HeightLimit(T))$ 
13:     return
14:   end if
15: end for

```

2. Choose $p \sim Uniform[\min_q(D_s), \max_q(D_s)]$
3. Define $D_{s,l} = \{x | x \in D_s, x_q < p\}$ and $D_{s,r} = D_s \setminus D_{s,l}$, then recursively apply the process on $D_{s,l}$ and $D_{s,r}$.

If we assume that the dimensionality is constant in the data, the distribution of the split point p depends only on the minimum and maximum values of the given dimension q of a set D_s . Thus, it is possible to maintain a SiTree when D_s is updated without fully rebuilding it. In other words, if we remove an example x from D_s and modify the tree structure accordingly, the updated tree $T' = Remove(T(D_s), x)$ comes from the same distribution as if it were created from $T(D_s \setminus \{x\})$, $P(T') = P(T(D_s \setminus \{x\}))$.

First, let us formalise the proof for the remove operation in Algorithm 4. We can observe that when we rebuild a subtree at node v :

1. We use exactly the same random tree building process;
2. On exactly the same set of points ($S_v \setminus \{x\}$);
3. This is identical to how $T(D_s \setminus \{x\})$ would build that portion. Thus,

Theorem 1. *The completely constructed SiTree $T(D_s \setminus \{x\})$ comes from the same distribution as $T' = \text{Remove}(T(D_s), x)$, which has been modified through Algorithm 4 upon the removal of x , $P(T') = P(T(D_s \setminus \{x\}))$.*

Proof. Assume that $x_q^{(i)} \neq x_q^{(j)}$, $\forall x \in D_s$ and $i \neq j$. Let q be the splitting dimension at the root, we have two cases:

1. x is not the minimum or maximum in dimension q : The range $[\min, \max]$ remains unchanged, so the split distribution remains the same, $P(\text{root split}|T') = P(\text{root split}|T(D_s \setminus \{x\}))$.
2. x is the minimum or maximum in dimension q : We rebuild the entire tree by the removal operation; This is exactly $T(D_s \setminus \{x\})$.

Assume that this is true for all nodes up to depth d . For node v at depth $d+1$, let S_v be the points that reach v and q be the divisible dimension at v . We have three cases:

1. If $x \notin S_v$, the subtree at node v is distributionally identical in both T' and $T(D_s \setminus \{x\})$.
2. If $x \in S_v$ and x is not the minimum or maximum in dimension q , the range $[\min, \max]$ is unchanged, the split distribution is unchanged and the children of node v receive the same example distributions.
3. If $x \in S_v$ and x is the minimum or maximum in dimension q , with the removal operation to rebuild the subtree at node v , the rebuilt subtree comes from the equivalent distribution with the corresponding subtree in $T(D_s \setminus \{x\})$.

For any node v where rebuilding occurs, $P(\text{split}|T', v) = P(\text{split}|T(D_s \setminus \{x\}), v)$, because they use: 1. same point set ($S_v \setminus \{x\}$); 2. same random splitting process; 3. same recursive tree building process.

Therefore, T' and $T(D_s \setminus \{x\})$ are distributionally equivalent because:

- The Removal process handles affected dimension ranges via rebuilding
- Rebuilding uses the same random process as $T(D_s \setminus \{x\})$
- Unaffected portions of the tree remain identical
- Any rebuilt subtree follows the same distribution as $T(D_s \setminus \{x\})$ ■

Theorem 1 indicates that the deletion process produces a tree T' that is distributionally identical to $T(D_s \setminus \{x\})$, as the process accurately detects and reconstructs the parts of the tree when x is removed from the subset D_s . Similarly,

Theorem 2. *When an example x , where $x \notin D_s$, is inserted into a SiTree, the tree $T'' = \text{Insert}(T(D_s), x)$ can be adjusted using Algorithm 5. This results in the effect that T'' is effectively modified as if it were fully constructed from $T(D_s \cup \{x\})$ by $P(T'') = P(T(D_s \cup \{x\}))$.*

Due to constraints on page length, the proof of Theorem 2 will not be provided herein. Its demonstration can be conducted analogously to the case of

Algorithm 5 Insert(T, x)

Input: T - The SiTree, x - an example

```

1: Let  $S_v$  be the set of points at node  $v$ 
2: Let  $q$  be the splitting dimension at  $v$ 
3: for  $v$  in path( $x$ ) on SiTree  $T$  do
4:   if  $v$  is a leaf then
5:     if  $v$  is at tree height limit level then
6:       Add  $x$  to the leaf {Reach the tree height limit}
7:       return
8:     else
9:        $v \leftarrow SiTree(S_v \cup \{x\}, Height(v), HeightLimit(T))$ 
10:    return
11:   end if
12: else if  $x_q < \min_q(S_v)$  or  $x_q > \max_q(S_v)$  then
13:    $v \leftarrow SiTree(S_v \cup \{x\}, Height(v), HeightLimit(T))$ 
14:   return
15: end if
16: end for

```

Theorem 1 by verifying alterations in the selected feature range during the insertion of examples.

Reservoir sampling approximates random sampling in batch learning, allowing us to maintain a uniform random sample in real-time. Therefore, with the Algorithm 3, a SiTree can be preserved dynamically in evolving streaming data with appropriate removal and insertion methods.

4.3 Computational Complexity

The primary operations within the main loop of Algorithm 1 include two key operations: scoring the examples (line 8) and updating the model (line 9). In the case of scoring, each example crosses from the root of the tree to a leaf, which amounts to $O(\log_2 w)$ complexity with the height limit $\log_2 w$, where w represents the size of the reservoir. The construction of a tree has a worst-case time complexity of $O(w \log_2 w)$. However, the time complexity associated with updating the model is dependent on the probability of altering the range of the selected feature for each branch. In the worst-case scenario, where reconstructing the entire tree is necessary, the complexity could increase to $O(w \log_2 w)$. In contrast, the optimal scenario involves simply traversing the example to the leaf node, resulting in a complexity of $\Omega(\log_2 w)$.

In terms of space, SiForest maintains k reservoirs, each of size w , leading to a total space complexity of $O(kw)$. These fixed parameters ensure efficient memory usage, making SiForest suitable for high-throughput streaming applications.

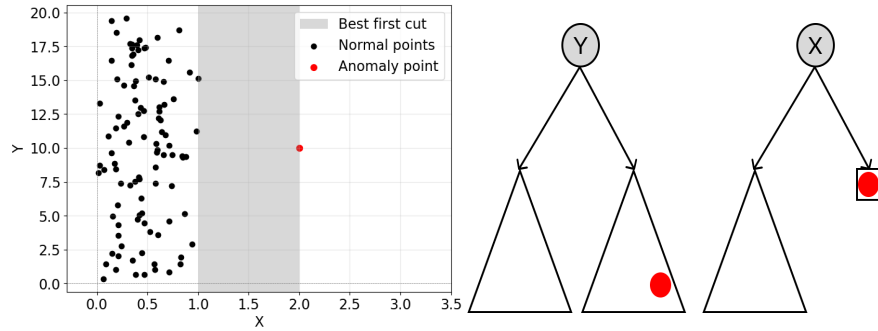


Fig. 1: RRCF is biased to construct the Y-root tree, whereas SiForest tends to form the X-root tree when constructing a tree with normal data points ranging from (0,0) to (1.0, 20.0); an anomaly position is at (2.0,10.0).

4.4 SiForest and RRCF

RRCF constructs trees by randomly selecting dimensions in proportion to their range, followed by a uniform split within the chosen dimension. This approach assumes that dimensions with smaller ranges contain primarily noise, reducing the likelihood of selecting less relevant features [7] and improving the effectiveness of RRCF for high-dimensional data. However, this design has a drawback: when small-range dimensions hold crucial information, they are less likely to be chosen.

Figure 1 illustrates this problem. The left image shows 100 normal examples clustered within a narrow region of the X-axis, whereas a red anomaly appears isolated. In RRCF, the probability of making a vertical cut in the shaded area at the root node depends on two factors: (1) the likelihood of selecting the X-axis, which is proportional to the feature ranges. This likelihood is calculated as $\frac{\text{range}(X)}{\text{range}(X) + \text{range}(Y)}$, which is $\frac{2}{2+20}$; and (2) the probability of placing the split within the shaded region, which is uniform at $\frac{1}{2}$. The combined probability of these two events is $\frac{2}{22} \times \frac{1}{2} = \frac{1}{22}$, which means that the tree is more likely to be rooted along the Y-axis. As a result, range-based partitioning tends to favour wider dimensions, often pushing anomalies deeper into the tree and reducing their anomaly scores.

In contrast, SiForest resolves this issue with an unbiased isolation method of choosing the dimension, offering a $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$ chance for the initial cut to isolate the anomaly, as shown in the right tree.

5 Experiments

This section presents experiments to evaluate our proposed method, detailing the setup, datasets, and evaluation metrics. We also analyse the results and compare our method with eight state-of-the-art algorithms.

Table 1: Overview of the 23 benchmark datasets.

Dataset	#Examples	#Features	#Anomalies	%Anomalies
ALOI [5]	49,534	27	1,508	3.04
backdoor [13]	95,329	196	2,329	2.44
campaign [14]	41,188	62	4,640	11.27
celeba [14]	202,599	39	4,547	2.24
census [14]	299,285	500	18,568	6.20
cover [2]	286,048	10	2,747	0.96
donors [14]	619,326	10	36,710	5.93
fraud [14]	284,807	29	492	0.17
http [16]	567,498	3	2,211	0.39
insects [9]	212,514	33	11,701	5.51
Friday [18]	191,033	78	1,966	1.03
magic.gamma [5]	19,020	10	6,688	35.16
mammography [22]	11,183	6	260	2.32
mulcross [16]	262,144	4	26,114	10.00
NYC taxi [7]	10,320	1	536	5.19
NYC taxi shingle [7]	10,273	48	536	5.22
shuttle [16]	49,097	9	3,511	7.15
skin [5]	245,057	3	50,859	20.75
smtp [16]	95,156	3	30	0.03
Thursday-Infiltration [18]	288,602	78	36	0.01
Thursday-WebAttacks [18]	170,366	78	2,180	1.28
Tuesday-workingHours [18]	445,909	78	13,835	3.10
Wednesday-workingHours [18]	692,703	78	252,672	36.48

5.1 Dataset

We carried out our experiments on a range of real-world datasets, including a synthetic dataset from OpenML called Mulcross. The chosen datasets contain a minimum of 10k samples, as our algorithm is built to manage large-scale data in practical applications. Table 1 summarises the characteristics of the data set, including size, features, anomaly counts, and references.

5.2 Setup

The experiments were conducted on an Intel[®] Core[™] i5-10500 3.10GHz CPU with 32 GB RAM running Ubuntu 22.04.4 LTS. SiForest was implemented in Python with CapyMOA [6] and compared with eight other streaming anomaly detection algorithms: RRCF; HST, IFASD and LODA in PySAD [23]; xStream, RSH, SDOS; and oIF. Each experiment was repeated ten times, and performance was measured using the AUC-ROC across benchmark datasets. The results were averaged and rounded to three decimals.

Models were compared with similar parameters: RRCF had 100 trees of size 256; HST used a window size of 256, eight height levels, and 100 trees; IFASD

had a tree size of 256, a 2048 sliding window, and 100 trees; LODA had 100 projectors and bins; xStream used 100 projections with Half-Space Chains, a depth of eight, and a window size of 256; RSH involved 100 detectors with four hash functions, in a range of 10000; SDOS utilized a window size of 256, 100 elements, and six nearest observers; oIF used 100 trees, with a maximum sample of 32 and a sliding window size of 2048; SiForest deployed 100 trees, each with a reservoir size of 256.

It is important to note that for all our experiments, we preserve the original sequence of examples within each dataset. Retaining the dataset in its initial order usually reveals two primary attributes within the data stream: the distribution of the data can change over time, and the density of anomalies can fluctuate significantly between different subsequences. However, if we randomise the sequence of dataset examples, these two features will be removed, thereby simplifying the detection of anomalies.

5.3 Results

In this section, we present results for SiForest compared with eight other state-of-the-art anomaly detection algorithms for data streams. Given that the original sequence may exhibit potential concept drift and a high concentration of anomalies at certain points, this allows us to evaluate the performance of the algorithms under such conditions.

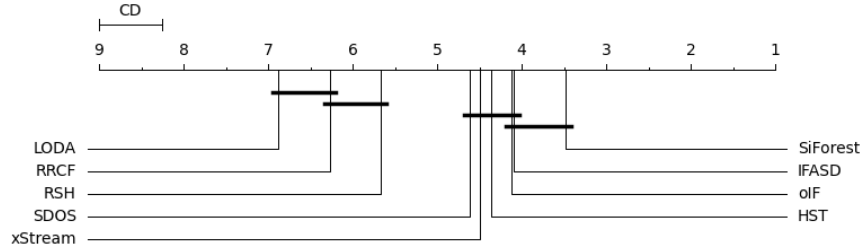


Fig. 2: Critical difference diagram for AUC-ROC rankings on the datasets with original sequences. $p = 0.05$.

The performance results in Table 2 indicate that our SiForest algorithm outperforms the other algorithms in terms of both the average AUC score and the ranking within the original order. We also evaluated the algorithms by ranking them for each run and showing the results using a critical difference diagram (Figure 2). The result indicates that SiForest is in the same group as IFASD and oIF, with a noticeable leading gap and significantly better than the other algorithms. We identified why certain algorithms don't perform well on specific datasets. Unfortunately, we can't share the insights here due to space constraints.

Table 2: AUC-ROC performance of SiForest and baseline algorithms on 23 datasets with examples in their original sequence. The highest scores are bolded for each dataset, highlighting SiForest’s superior ranking.

dataset	SiForest	HST	RRCF	IFASD	LODA	xStream	RSH	SDOS	oIF
ALOI	0.583	0.787	0.579	0.994	0.463	0.332	0.693	0.704	0.635
backdoor	0.808	0.567	0.736	0.614	0.204	0.769	0.611	0.731	0.657
campaign	0.612	0.693	0.602	0.693	0.623	0.640	0.718	0.592	0.667
celeba	0.669	0.741	0.616	0.691	0.578	0.673	0.727	0.578	0.756
census	0.555	0.601	0.547	0.593	0.512	0.592	0.600	0.490	0.609
cover	0.552	0.463	0.573	0.665	0.517	0.587	0.418	0.512	0.861
donors	0.717	0.738	0.606	0.772	0.334	0.764	0.745	0.798	0.804
fraud	0.948	0.913	0.861	0.942	0.562	0.949	0.951	0.947	0.949
Friday	0.624	0.574	0.589	0.598	0.528	0.312	0.413	0.598	0.460
http	0.999	0.313	0.241	0.950	0.290	0.320	0.075	0.080	0.238
insects	0.758	0.765	0.604	0.722	0.448	0.752	0.736	0.705	0.718
magic	0.664	0.806	0.488	0.412	0.381	0.720	0.746	0.776	0.742
mammo	0.846	0.805	0.744	0.721	0.263	0.877	0.840	0.845	0.844
mulcross	0.955	0.324	0.725	0.952	0.025	0.949	0.179	0.009	0.988
NYC	0.635	0.617	0.489	0.634	0.520	0.408	0.495	0.565	0.513
NYC-S	0.830	0.424	0.555	0.708	0.500	0.525	0.500	0.790	0.472
shuttle	0.997	0.978	0.775	0.956	0.470	0.989	0.874	0.996	0.975
skin	0.482	0.104	0.365	0.699	0.543	0.093	0.183	0.347	0.306
smtpt	0.888	0.837	0.804	0.889	0.659	0.816	0.835	0.851	0.878
Thu-Inf	0.950	0.907	0.843	0.913	0.307	0.941	0.931	0.941	0.948
Thu-WA	0.630	0.542	0.606	0.618	0.843	0.727	0.194	0.733	0.451
Tue	0.651	0.644	0.453	0.392	0.535	0.666	0.332	0.546	0.622
Wed	0.543	0.794	0.431	0.572	0.495	0.769	0.361	0.588	0.555
Avg AUC(↑)	0.735	0.649	0.601	0.726	0.461	0.660	0.572	0.640	0.680
Avg Rank(↓)	3.474	4.365	6.261	4.091	6.878	4.487	5.670	4.609	4.122

6 Conclusion

This paper introduced Streaming Isolation Forest(SiForest), a novel anomaly detection algorithm for streaming data. Using dynamic tree maintenance with reservoir sampling, SiForest efficiently updates its model to reflect evolving data distributions. We provide theoretical proof that it is possible to incrementally construct an tree using a single-pass approach and make it distributionally equivalent to a tree built with access to the full data. In practice, SiForest surpasses eight state-of-the-art streaming anomaly detection algorithms, demonstrating superior accuracy and flexibility, even when anomalies are unevenly distributed within streams. Future work could explore alternative sampling strategies and enhancements to address concept drift to improve performance in dynamic real-world environments.

Acknowledgments. TAI AO programme CONT-64517-SSIFDS-UOW funded by MBIE.

References

1. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Machine Learning for Data Streams with Practical Examples in MOA. MIT Press (2018), <https://moa.cms.waikato.ac.nz/book/>
2. Blackard, J.A., Dean, D.J.: Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture* **24**(3), 131–151 (1999)
3. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. *ACM computing surveys (CSUR)* **41**(3), 1–58 (2009)
4. Ding, Z., Fei, M.: An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *IFAC* **46**(20), 12–17 (2013)
5. Emmott, A., Das, S., Dietterich, T., Fern, A., Wong, W.K.: A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158* (2015)
6. Gomes, H.M., Lee, A., Gunasekara, N., Sun, Y., Cassales, G.W., Liu, J.J., et al.: CapyMOA: Efficient machine learning for data streams in python (2025), <https://arxiv.org/abs/2502.07432>
7. Guha, S., Mishra, N., Roy, G., Schrijvers, O.: Robust random cut forest based anomaly detection on streams. In: *ICML*. pp. 2712–2721. PMLR (2016)
8. Hartl, A., Iglesias, F., Zseby, T.: Sdostream: Low-density models for streaming outlier detection. In: *ESANN*. pp. 661–666 (2020)
9. Leveni, F., Weigert Cassales, G., Pfahringer, B., Bifet, A., Boracchi, G.: Online isolation forest. In: *ICML*. PMLR (2024)
10. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: 2008 eighth ieee international conference on data mining. pp. 413–422. IEEE (2008)
11. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* **6**(1), 1–39 (2012)
12. Manzoor, E., Lamba, H., Akoglu, L.: xstream: Outlier detection in feature-evolving data streams. In: 24th ACM SIGKDD. pp. 1963–1972 (2018)
13. Moustafa, N., Slay, J.: Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: 2015 MilCIS. IEEE (2015)
14. Pang, G., Shen, C., Van Den Hengel, A.: Deep anomaly detection with deviation networks. In: 25th ACM SIGKDD. pp. 353–362 (2019)
15. Pevný, T.: Loda: Lightweight on-line detector of anomalies. *Machine Learning* **102**, 275–304 (2016)
16. Rayana, S.: Odds library (2016), <https://odds.cs.stonybrook.edu>
17. Sathe, S., Aggarwal, C.C.: Subspace outlier detection in linear time with randomized hashing. In: 16th ICDM. pp. 459–468. IEEE (2016)
18. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A., et al.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **1** (2018)
19. Tan, S.C., Ting, K.M., Liu, T.F.: Fast anomaly detection for streaming data. In: 22nd international joint conference on artificial intelligence. Citeseer (2011)
20. Vázquez, F.I., Zseby, T., Zimek, A.: Outlier detection based on low density models. In: 2018 ICDMW. pp. 970–979. IEEE (2018)
21. Vitter, J.S.: Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* **11**(1), 37–57 (1985)
22. Woods, K.S., Doss, C.C., Bowyer, K.W., Solka, J.L., Priebe, C.E., Kegelmeyer Jr, W.P.: Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. *IJPRAI* **7**(06), 1417–1436 (1993)
23. Yilmaz, S.F., Kozat, S.S.: Pysad: A streaming anomaly detection framework in python. *arXiv preprint arXiv:2009.02572* (2020)