

Anomaly Detection for Data Streams Based on Isolation Forest Using Scikit-Multiflow

Maurras Ulbricht Togbe[1(✉)], Mariam Barry[2,3], Aliou Boly[4], Yousra Chabchoub[1], Raja Chiky[1], Jacob Montiel[5], and Vinh-Thuy Tran[2]

[1] ISEP, LISITE, Paris, France
{maurras.togbe,yousra.chabchoub,raja.chiky}@isep.fr
[2] BNP Paribas, IT Group, Division Data, Montreuil, France
{mariam.barry,vinh-thuy.tran}@bnpparibas.com
[3] Télécom Paris, LTCI, Institut Polytechnique de Paris, Palaiseau, France
[4] Université Cheikh Anta Diop de Dakar, Dakar, Senegal
aliou.boly@ucad.edu.sn
[5] Department of Computer Science, University of Waikato, Hamilton, New Zealand
jacob.montiel@waikato.ac.nz

Abstract. Detecting anomalies in streaming data is an important issue in a variety of real-word applications as it provides some critical information, e.g., Cyber security attacks, Fraud detection or others real-time applications. Different approaches have been designed in order to detect anomalies: statistics-based, isolation-based, clustering-based. In this paper, we present a quick survey of the existing anomaly detection methods for data streams. We focus on Isolation Forest (iForest), a state-of-the-art method for anomaly detection. We provide the implementation of IForestASD, a variant of iForest for data streams.

This implementation is built on top of scikit-multiflow, an open source machine learning framework for data streams. In fact, few anomalies detection methods are provided in the well-known data streams mining frameworks such as MOA or StreamDM. Hence, we extend scikit-multiflow providing an additional tool. We performed experiments on 3 real-world data sets to evaluate predictive performance and resource consumption (memory and time) of IForestASD and compare it with a well known and state-of-the-art anomaly detection algorithm for data streams called Half-Space Trees.

Keywords: Anomaly detection · Streaming · Scikit-multiflow · Survey

1 Introduction

Data streams mining is the era that deals with extracting relevant and meaningful patterns from data arriving in a continuous way. It is a challenging problem especially when applied to evolving data streams or is subject to big data

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12252, pp. 15–30, 2020.
https://doi.org/10.1007/978-3-030-58811-3_2

constraints where optimized storage and fast processing are required. When it comes to some streaming applications, such as network attacks, frauds, or failures warning from vital maintenance predictive tools, abnormal patterns (aka outliers) need to be detected as fast as possible to get insights for decision-making. Therefore, anomaly detection in data streams is a major task for some business applications which activities depend on continuous streams from real time events. Anomaly detection algorithms for data streams refer to methods able to handle a continuous and possibly infinite streams and at the same time, extract enough knowledge from the streams to compute anomaly score.

Many frameworks for Data stream mining have been proposed in the literature. Scikit-Multiflow [20] is the main open source machine learning framework for multi-output, multi-label and data streaming. Implemented in Python language, it includes various algorithms and methods for streams mining and in particular the popular Half-space Trees algorithm [24], a fast Anomaly Detection for Streaming Data. One of its motivations is to encourage researchers and industrials on data streams mining field to easily integrate and share their methods inside the framework and make their work easily accessible by the growing Python community.

Our contributions is two-fold: first, we provide a structured overview and categorization of anomaly detection methods for data streams along with a discussion about the main advantages and disadvantages of the approaches. Second, we implement in Scikit-MultiFlow framework an algorithm proposed by [11] (Ding & Fei, 2013), an isolation based anomaly detection (IForestASD) and highlight the simplicity of the framework to accelerate contributions. We performed experiments with 3 real data sets to evaluate predictive performance and resource consumption (memory and time) of IForestASD and compare it with a well known and state of art anomaly detection algorithm for data streams called Half-Space Trees. To our best knowledge there is no open source implementation of IForestASD, neither in Github neither in a streaming framework.

1.1 MOA: Massive Online Analysis

MOA [8] is the most popular open source framework for data stream mining in Java. It includes a collection of machine learning algorithms (classification, regression, clustering, outlier detection, concept drift detection and recommender systems) including data generators, tools for evaluation and an interface to visualize experiments results. MOA is related to Waikato Environment for Knowledge Analysis (WEKA) [28]. In the recent book [7], the authors covered the field of Online Learning from Sketches and Drift detection approaches to supervised and non supervised algorithms for Data Streams. Some practical examples with MOA are also provided. More information can be found in the MOA Manual[1].

The authors of [5] provided a comparative study of distributed tools for analyzing streaming Data, with a qualitative comparison between Apache Spark, Storm, Samza and Apache S4.

[1] https://moa.cms.waikato.ac.nz/documentation/.

Others Software Libraries and Frameworks. Multiple open-source software libraries use MOA to perform data stream analytics in their systems [7], including ADAMS, MEKA, and OpenML.

Others big data streams frameworks are SAMOA [6] and StreamDM. Apache SAMOA is an open source platform for mining big data streams. It is a framework that contains a programming abstraction for distributed streaming ML algorithms. StreamDM [9] is an open source data mining and machine learning library, designed on top of Spark Streaming, an extension of the core Spark API that enables scalable data streams processing.

1.2 Scikit-Multiflow: A Machine Learning Framework for Multi-output/Multi-label and Streaming Data

Scikit-multiflow [20] is an open-source framework in Python to implement algorithms and perform experiments in the field of machine learning on evolving data streams. Scikit-multiflow is inspired by the popular frameworks Scikit-learn[2], MOA and MEKA. One advantage of scikit-multiflow compared to the other frameworks is that it serves as a bridge between research communities that have flourished around the aforementioned popular frameworks, providing a common ground for researchers and practitioners.

Scikit-multiflow includes a collection of various algorithms and learning methods: From Rule and Trees based methods such as Hoeffding Anytime Tree or Extremely Fast Decision Tree [19] to ensemble methods such as Adaptive Random Forest classifier [13]. The list of existing algorithms can be found in the package map on the official webpage https://scikit-multiflow.github.io/scikit-multiflow/package_map.html.

Half-Space Trees is the only method for anomaly detection in the current scikit-multiflow release.

Half-Space Trees - Fast Anomaly Detection for Data Streams
Half-Spaces-Trees [24] - Fast Anomaly Detection for Data Streams - is the only anomaly detection algorithm currently implemented in scikit-multiflow. The authors of the algorithm shown that the training phase has constant amortized time complexity and constant memory requirement. When compared with a state-of-the-art method (Hoeffding Trees) its performs favorably in terms of detection accuracy and run-time performance. Thus, we expect HS-Trees to perform better than Isolation Forest ASD as well in term of speed, we will discuss the experiments results in Sect. 4. Compared to Isolation Tree, Half-Space Tree has a fixed depth for each tree in the ensemble and update policy consists of updating the mass profile using the nodes of trees.

1.3 Motivation of the Work

From the review of Big Data Streams Mining Frameworks, scikit-multiflow seems to be the most promising one as it implements the majority of well known stream

[2] https://scikit-learn.org/stable/scikit-learn.

learning methods using the very well known programming language Python with a growing community. And since it contains only one anomaly detection method, which is Half-Spaces-Trees [24], we thus decided to extend the framework by implementing an anomaly detection approach based on Isolation Forest algorithm proposed by Ding & Fei [11].

Hence, the motivations behind our implementation providing an isolation-based algorithm in Scikit-multiflow are the follow:

- **Isolation Forest** is a state-of-the-art algorithm for anomaly detection and the only ensemble method in scikit-learn's and widely used by the community. Also, it is a tree-based model easily suitable for online and incremental learning.
- **Scikit-multiflow** is the main streaming framework in Python which includes a variety of learning algorithms and streaming methods.

The paper is organized as follows: Sect. 1 introduces our work and its motivations. In Sect. 2, we provide a survey and classification of anomaly detection adapted to data streams. In Sect. 3, we focus on algorithms description: first batch Isolation Forest and its variant implemented for streaming setting (IForestASD). In Sect. 4 we present experimental evaluations, comparing IForestASD to Half-Space Trees and discuss the results. Finally, Sect. 5 concludes the work by providing future research directions.

2 Anomaly Detection in Data Streams: Survey

2.1 Survey

Anomaly Detection in Data Stream (ADiDS) presents many challenges due to the characteristics of this type of data. One important challenge is that data stream treatment has to be performed in a single pass to deal with memory limits and methods have to be applied in an online way. Thus, the several existing offline anomaly detection approaches such as statistical approach, clustering approach, etc. [1,10,16] are not adapted for data stream because they require many passes over dataset. They also need to have the entire dataset to be able to detect anomalies. We find in the literature some approaches that have been adapted or new designed methods for ADiDS. In [14], authors give a survey on outlier detection methods that can be applied to temporal data with a focus on data streams. They presented evolving prediction models, distance based approach and outlier detection in high dimensional data streams. [25,26] and [22] are all surveys about outlier detection in data stream context. In [26] authors present the issues of outlier detection in data stream like concept drift, uncertainty and arrival rate. A detailed study on time series and multidimensional streaming outlier detection methods is proposed in the book [1, Chap. 9]. It presents different approaches such as probabilistic-based, prediction-based and distance-based methods.

2.2 Approaches and Methods Classification

Generally, anomaly detection methods are based on the facts that anomalies are rare and have a different behavior compared to normal data. These characteristics are true for static datasets and also data streams. The most used anomalies detection approaches are statistics, clustering, nearest-neighbors that we will present below. We will focus on an approach based on isolation: Isolation Forest, proposed in 2008 by Liu *et al.* in [17].

Statistics-Based: Statistics-based approaches generally establish a model that characterizes the normal behavior based on the dataset. The new incoming data which don't fit the model or have very low probability to fit the model are considered as abnormal. Some methods give a score for the data based on the deviation degree from the model [29]. Statistics-based methods can be parametric in which case they need to have a prior knowledge about the dataset distribution. They can be non parametric where they learn from the given dataset to deduce the underlying distribution. In the context of data stream, such prior knowledge is not always available.

Clustering-Based and Nearest-Neighbors-Based: Clustering and nearest neighbors approaches are based on the proximity between observations. The methods in this category are based either on the distance (distance-based) or the density (density-based). Clustering methods divide the dataset in different clusters according to the similarity between the observations. The most distant cluster or the cluster which has the smallest density can be considered as an anomaly cluster [2,4]. Nearest-neighbors methods determine the neighbors of one observation by computing the distance between all the observations in the dataset. The observation which is far from its k nearest-neighbors can be considered as an anomaly [3]. It is also characterized as the observation which has the most less neighbors in a radius r (a fixed parameter) [21]. These approaches need to compute the distance or the density between all the observations in the dataset or they need to have some prior knowledge about the dataset. So they can suffer of a high CPU, time and memory consumption or a lack of information.

Isolation-Based: Introduced by [17], the principle of the isolation-based approach is to isolate abnormal observations from the dataset. Anomalies data are supposed to be very different from normal ones. They are also supposed to represent a very small proportion of the whole dataset. Thus, they are likely to be quickly isolated. Some isolation-based methods are presented in Sect. 3. Isolation based methods are different from others statistics, clustering or nearest-neighbors approaches because they don't compute a distance or a density from the dataset. Therefore, they have a lower complexity and are more scalable. They don't suffer from the problem of CPU, memory or time consumption. Thus, isolation based methods are adapted to the data stream context.

The Table 1 summarizes advantages and disadvantages of the existing approaches for ADiDS.

There are many methods adapted or designed for ADiDS in the literature. They usually use the data stream concept of window to compute anomaly [22].

Table 1. Comparison of ADiDS approaches.

Approaches	Advantages	Disadvantages
Statistics-based	Non-parametric methods are adapted to data stream context	– Parametric methods are difficult to apply to data stream – Non-parametric methods can only be used for low dimensional data stream
Nearest-neighbors	Distance-based methods – Adapted for global anomalies detection	Distance-based methods – Are not adapted for non-homogeneous densities – Have high computational cost for high dimensional data stream
	Density-based methods – Adapted for local anomalies detection – More efficient than distance-based methods	Density-based methods – Have high complexity – Are not effective for high dimensional data stream
Clustering-based	Adapted for clusters identification	Not optimized for individual anomaly identification
Isolation-based	– Have less CPU, time and memory consumption – Are efficient for anomaly detection	– Performance depend a lot on the window and model update policy choices – Hard to adapt for categorical data

The Fig. 1 presents a classification of some anomaly detection methods for each category that exist in the literature and applicable to data stream.

A recent work [27] on anomaly detection approaches has shown that isolation forest algorithm has good performance compared to other methods. Thus we decided to implement an adaptation of isolation forest in a data streams mining framework that will be detailed in the next sections.

3 Isolation Forest and IForestASD for Streaming

The first method proposed in isolation-based category is Isolation forest (IForest) [17,18]. One of IForest's limits is that it was designed for static dataset and not for data stream. In [11], authors propose an improvement of IForest to adapt it to data stream context using sliding windows. The proposed method is named Isolation Forest Algorithm for Stream Data (IForestASD).

There exists other improvements and adaptations of Isolation Forest such as Extended Isolation Forest [15] or Functional Isolation Forest [23] but they are designed for batch settings and not adapted for data streaming.

In this section we will present the IForest algorithm then we will explained the implemented algorithm IForestASD.

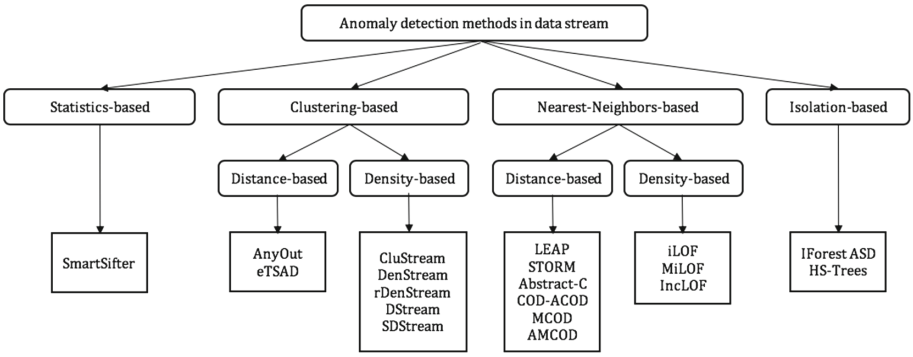

Fig. 1. Classification of data stream anomaly detection methods

3.1 Isolation Forest Method

Isolation forest (IForest) is an isolation-based method which isolates observations by splitting dataset. Anomaly data have two properties which have been exploited by isolation-based methods: They have a behavior which is different from normal data and they are rare in the dataset. With those properties, a relatively small number of partitions is needed to isolate anomalies as shown in Fig. 2. IForest is based on a forest of random distinct itrees. Every tree has to decide if a considered observation is an anomaly or not. If a random forest of itrees globally consider an observation as an anomaly, then it is very likely to represent an anomaly. IForest is composed of two phases: the first one is the training phase which is the construction of the forest of random itrees and the second one is the scoring phase where IForest gives an anomaly score for every observation in the dataset.

In the training phase, all the t random and independents itrees of the forest are built. Using the sample of ψ randomly selected data, every internal node of the binary itree is split in two other nodes (left and right). To split one node, IForest randomly chooses one attribute: d from the m data attributes. Then it randomly chooses a split value v between the min and the max value of d in the considered node. IForest splits internal nodes until a complete data isolation or reaching a maximal tree depth called max_depth which is equal to $ceilling(log_2(\psi))$. After the forest training phase, the scoring phase can begin. In this phase, every new observation x has to pass through all the t itrees to get its path length $h(x)$ [17]. The anomaly score of x is computed with this formula: $s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$ where $E(h(x)) = \frac{\sum_{i=1}^{t} h_i(x)}{t}$ is the average path length of x over t itrees and $c(n)$ is the average path length of unsuccessful search in Binary Search Tree (BST). $c(n) = 2H(n-1) - (2(n-1)/n)$ with $H(i) = ln(i) + \gamma$ (γ is Euler's constant). Finally, if $s(x,n)$ is close to 1, x is considered as an anomaly. If $s(x,n)$ is less than 0.5, x is considered as a normal data.

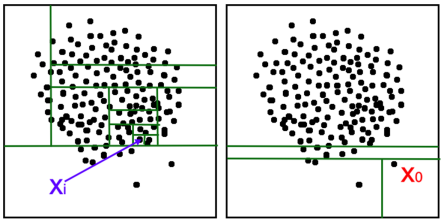

Fig. 2. Normal data X_i needs 11 split steps to be isolated. X_0 has been isolated very quickly in 3 steps.

3.2 IForestASD: Isolation Forest Algorithm for Stream Data Method

Isolation Forest is an efficient method for anomaly detection with relatively low complexity, CPU and time consumption. It requires all the data in the beginning to build t random samples. It also needs many passes over the dataset to build all the random forest. So, it is not adapted to data stream context. In [11], authors have proposed the so called IForestASD method which is an improvement of IForest for ADiDS. IForestASD uses sliding window to deal with streaming data. On the current complete window, IForestASD executes the standard IForest method to get the random forest. This is the IForest detector model for IForestASD. IForestASD method can also deal with concept drift in the data stream by maintaining one input desired anomaly rate (u). If the anomaly rate in the considered window is upper than u, then a concept drift occurred so IForestASD deletes its current IForest detector and builts another one using all the data in the considered window. Figure 3 represents the workflow used in IForestASD to update the model. In Sect. 4 we present our experiments results of this algorithm in scikit-multiflow framework using incremental fit and prediction methods.

4 Experimental Evaluation

In this section, we present the methodology followed in our tests and discuss the results. We compare and discuss the impact of the data window size and the number of estimators in the performance of both Half-Space Tree and IForestASD in terms of F_1 score, training and testing time, and model size.

Performance is measured following the prequential (test-then-train method) evaluation strategy designed specifically for stream settings, in the sense that each sample serves two purposes (test then train), and that samples are analyzed sequentially, in the order of arrival, and become immediately inaccessible. This approach assures that the model is tested on new samples that have not been used in the training yet. All the results and performance metrics reported in this paper correspond to the output of scikit-multiflow from prequential evaluator[3].

[3] https://scikit-multiflow.github.io/scikit-multiflow/documentation.html.

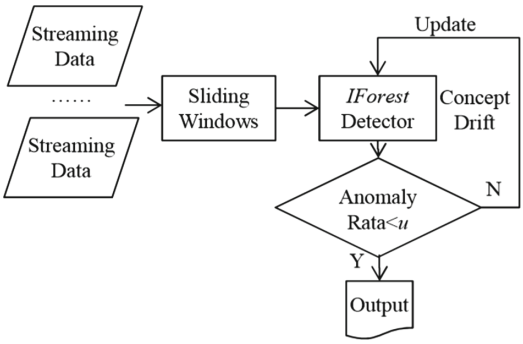

Fig. 3. Isolation Forest ASD algorithm workflow for Drift Detection implemented in Scikit-multiflow. Image extracted from the original paper by [Ding & Fei, 2013] [11]

4.1 Search Space, Data Sets and Metrics

Data Sets Description
To prove the effectiveness of our implementation of IForestASD, we benchmark it against the Half-Space Trees method on a variety of configurations (12 for each dataset). We used some well known public datasets [12] which have been used in the IForestASD paper. Their characteristics are resumed in Table 2.

Experimental Setup
The hyper-parameters setting we used to perform various experiments to compare the two models HSTrees and IForestASD is the following:

Table 2. Datasets and hyper-parameters set up

	Shuttle	Forest-cover	SMTP
Attributes, number of features	9	10	3
u - Drift rate = anomaly rate	7.15%	0.96%	0.03%
Number of samples	49,097	286,048	95,156
W - Window size range tested	[50, 100, 500, 1000]		
T - Number of trees in ensemble	[30, 50, 100]		

The anomaly threshold was set to 0.5 and number of observations (samples) was limited to 10 000 for both Datasets for each experiment. For the parameter u, we set the value to the anomalies rate in the datasets as did the authors of IForestASD algorithm [11]. This value is used to detect drift and reset the anomaly detection model as described in Fig. 3.

The source code to replicate experiments can be found in the Github at https://github.com/MariamBARRY/skmultiflow_IForestASD.

Evaluation Metrics
In the results table, we reported 3 metrics: F1, Running Time (training + testing) ratio (IForestASD/HSTrees) and Model Size ratio (HSTrees/IForestASD).

F1 Metric. It corresponds to the harmonic mean of precision and recall and is computed as follows: $f_1 = 2 \times (precision \times recall)/(precision + recall)$. F1 is a popular metric if there is an imbalanced class distribution and we search for a balance between precision and recall. This is the case of abnormal data which is less represented than normal data.

Running Time Ratio (IForestASD Coefficient Ratio) - IRa. Since Half-Space Trees (HST) is always faster than IForestASD (IFA), for both training and testing time, by an order of 400 for the worst case, we reported the ratio between running time (training and testing) of the two models: IFA over HST. For example for the Forest-Cover data, IFA is 3 times slower than HST for $W = 50, T \in \{30, 50, 100\}$, and for $W = 1000, T = 100$ the running time ratio is 391 in the Table 3 meaning that IFA is 391 slower than HST. Absolute values of running time (in seconds) and their evolution over hyper-parameters variation in reported in Fig. 5.

Model Size Ratio (HSTrees Coefficient Ratio) - HRa. In the opposite of the running time, when we consider the model size, we observe that IFA always used less memory then HST. So, to compare these two methods, we compute the ratio of the higher value (HST) over the smaller value (IFA) to get the HST coefficient. Figure 4 reports the impact of parameters choice on resources used (model size absolute value in Kilo-bytes) for both HSTrees and the evolution of resources used for the 3 datasets when window size and number of trees varies. Interpretation are provided below.

4.2 Results Discussion

Table 3 reports the results obtained for each hyper-parameter combination (window size and number of trees) for both datasets.

We observed that, based on the F1 score, IForestASD performs better than HS-Trees for both Shuttle and SMTP datasets, no matter the windows size or the number of trees. Three further points are observed:

First, we noticed that the number of trees has not a significant impact on the prediction performance for all the datasets. However, when the window size increases (varying from 50, 100, 500 to 1000), the score is better as F1 score increases for all data sets. Second, while for HS-Trees, F1 improves for window size >= 500, it decreases for IForestASD. This can be explained by the fact that unlike HS-Trees, IForestASD deletes its current anomaly detector model (isolation forest) and builds another one using all the data in the current window when the anomaly rate in the window is upper than u. Thirdly, there is a large difference between the performance of the two models depending on the dataset, especially for Shuttle which has a larger anomaly rate of 7%, IForestASD outperforms HSTrees with 80% F1. Therefore we assume that IForestASD performs better on data set with relatively high anomaly rate.

Table 3. Comparison between HS-Trees (HST) and IForestASD (IFA) - We fixed the window size w (50, 100, 50, 1000) and varied the number of trees T for each dataset Forest Cover, Shuttle and SMTP (36 configurations) - HRa and IRa represent HSTRees ratio and IFA ratio described in Sect. 4.1

		Forest cover				Shuttle				SMTP			
		F1		Time	Size	F1		Time	Size	F1		Time	Size
W	T	HST	IFA	IRa	HRa	HST	IFA	IRa	HRa	HST	IFA	IRa	HRa
50	30	**0.36**	0.22	**3**	702	0.13	**0.64**	**3**	817	0	**0.34**	**3**	926
50	50	**0.36**	0.23	**3**	694	0.13	**0.64**	**3**	854	0	**0.34**	**3**	940
50	100	**0.36**	0.22	**3**	737	0.13	**0.64**	**3**	950	0	**0.34**	**3**	986
100	30	**0.39**	0.30	12	367	0.14	**0.71**	12	461	0	**0.36**	9	596
100	50	**0.39**	0.29	12	404	0.13	**0.72**	14	505	0	**0.36**	9	734
100	100	**0.39**	0.30	12	416	0.13	**0.71**	16	551	0	**0.36**	9	808
500	30	0.39	**0.49**	158	108	0.14	**0.80**	519	28	0	**0.39**	106	288
500	50	0.39	**0.47**	152	129	0.13	**0.80**	393	36	0	**0.40**	111	372
500	100	0.39	**0.47**	156	140	0.15	**0.80**	363	92	0	**0.40**	111	434
1000	30	**0.54**	0.40	372	**63**	0.17	**0.78**	1757	**21**	0	**0.39**	264	**127**
1000	50	**0.54**	0.40	387	**74**	0.14	**0.78**	1175	**28**	0	**0.39**	272	**155**
1000	100	**0.55**	0.40	391	**86**	0.14	**0.77**	1678	**23**	0	**0.39**	272	**176**

In complement with predictive performance, two important notions in data streaming applications are time and memory usage of models. Here we analyze the impact of hyper-parameters (windows size W and number of trees T) on the amount of resources used by each model and compare them.

Models Memory Consumption Comparison
Regarding the model size evolution from Fig. 4, where model size of both models are represented in 2 Y axis (IForestASD in right), we can highlight 3 points:

- IForestASD used less memory than HSTrees (≈20 times less), this is explained by the fact that with IForestASD, update policy consists on discarding completely old model when drift anomaly rate in sliding window $> u$ (updates by replacement) while HSTrees continuously updates the model at each observations
- For HSTrees, Window size W has no impact on the model size, only the number of trees T increases the memory used (barplots). This is consistent with the HSTrees update policy which consists on updating statistics in Tree nodes with a fixed ensemble size.
- For IForestASD, both window size w and number of trees T have a positive impact on model size, for both 3 datasets (in 3 linesplot). This is due to the fact that IForestASD uses all the instances in the window (W) to build the ensemble with T trees.

When it comes to some critical streaming applications, such as a predictive model for Network security intrusion, a fast model but less accurate is often

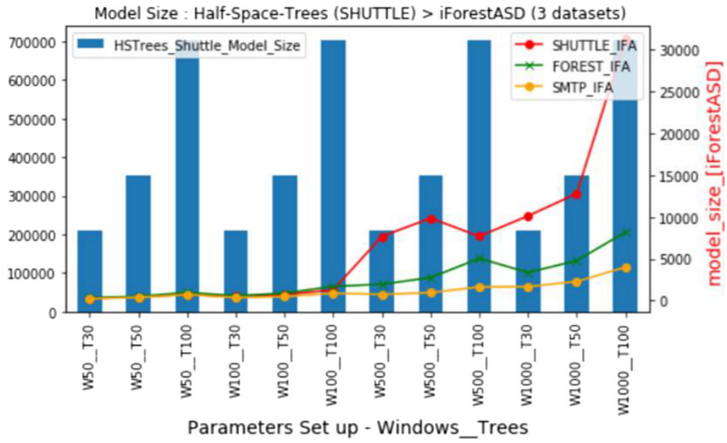

Fig. 4. Model-size metric (kilo-bytes): we fixed the window size (50, 100, 500, 100) and varied the number of Trees (30, 50, 100). For each setting we run HSTrees (barplot - Left Y axis) with Shuttle and IForestASD (lines - Right Y axis) for each dataset - 36 measures - and plot lines (right Y axis). HST is 20x bigger than IFA

preferred over a slow and more accurate model. Indeed, having a high rate of false positive anomalies to investigate is preferred than missing critical true positive anomalies (attack). Therefore, we aim to analyze below the behavior of the models in terms of training and testing time.

We can observe from Fig. 5 that IForestASD is faster with a small window size while Half-Space Trees is faster with bigger window size. The number of estimators increases the running time of both of them. This is consistent with the time complexity of the two base estimators: isolation Tree and Half-Space Tree. Indeed, the worse-case time complexity for training an iForest is $O(t\psi_1^2)$ [18] while for the streaming HS-Trees, the (average-case) amortized time complexity for n streaming points is $O(t(h + \psi_2))$ in the worst-case [11], where t is the number of trees in the ensemble, ψ_1 is the subsampling size of the iForest, ψ_2 the window size in HS-Trees and h the maximum depth (level) of a tree.

Furthermore, the testing time of IForestASD – in the right axis in red line – can be 100x longer than HSTrees testing time (40,000 vs 400) which is a constraint for critical applications that need anomalies scoring quickly. The difference is significant between the two models because in IForestASD, each instance in the window is tested across each of the isolation trees to compute the anomaly score, thus the exponential increase regarding hyper-parameters.

Our results highlight that a trade-off must be made between resources consumption, processing time and predictive performance (F1 for anomalies) to get acceptable results. In the context of anomaly detection (predictive performance), we can conclude that in terms of F1 score, IForestASD is better than HS-Trees. However, in the context of data streaming applications running time is an important factor and faster models are preferred. In this sense, Half-Space Trees is a

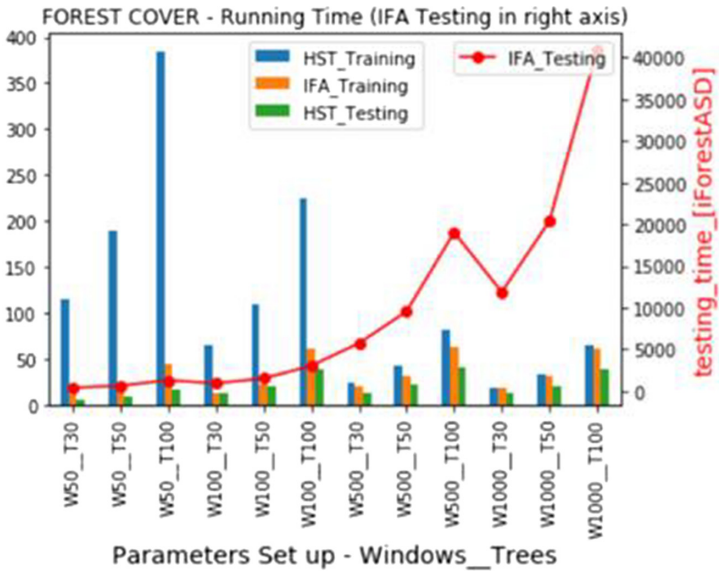

Fig. 5. Running time metric (secondes): forest-cover dataset - we fixed the window size (50, 100, 500, 100) and vary the number of Trees (30, 50, 100) to compare HST and IForestASD running time. IForestASD testing time is represented in another scale on the right Y axis and plot in red line. (Color figure online)

good option due to its lower processing time in the opposite to IForestASD which is exponential with respect to the windows size.

4.3 Guidelines for Setup Depending on Application Requirements

The number of experiments (36 configurations - 3 nb-estimators × 4 window-size × 3 datasets) used to obtain previous findings and discussions provides some insights about good grids of parameters and which model to use in which context.

Since models performance can vary a lot from one choice to another, depending on the characteristics of the data and hyper-parameters setting, we provide some guidelines for deciding between these anomaly detection methods. Thus, based on our experiments, depending on specific application requirements and resources constraints (running time and model size), one can note:

- **Window w for F1**: If the priority is F1 performance for anomaly detection, then setting W close to 500 gives better results with IForestASD.
- **Time & Memory**: If a fast model and especially a fast scoring time is needed, HSTrees should be the privileged option as it is still the state-of-the-art among fast streaming models.
- **IForestASD set up**: When fast model is not a prioritized requirement, IFA gives better results with low running time when using a smaller windows

(lower than 100) and low model size with number of estimators T between 30 and 50.
- **HSTrees set up**: When a fast model is required, HSTress with $w = 1000$ and number of $T = 30$ lead to the best results with optimal resources cost.

We have seen that the best model can be either HSTrees or IForestASD depending on the dataset and application requirements, thus we recommend testing models with dataset sample using scikit-multiflow framework to identify the best parameters for a each application.

5 Conclusion and Future Work

We presented a structured and comprehensive overview of anomaly detection algorithms for streaming data and categorized them by anomaly based approaches: statistical, nearest-neighbors, clustering and isolation based. For each category, we highlighted the advantages and disadvantages of the different approaches. From this study, Isolation forest seems to be an accurate method but not adapted to data streams [27]. We thus proposed an implementation of an anomaly detection approach based on isolation forest for streaming data using sliding window (IForestASD algorithm) in Scikit-multiflow, a machine learning framework for multi-output/multi-label and stream data. The motivation behind this work is that there exist very few anomalies detection methods in streaming frameworks reviewed, and the contribution on scikit-multiflow can help companies, researchers and the growing python community exploit methods and collaborate for the challenging area of Anomaly Detection in Streaming context.

Future Research Direction
This work is the first step of research project about streaming methods in scikit-multiflow which involves multiple stakeholders (co-authors).

The IForestASD approach requires a parameter for drift detection that is manually fixed. As a perspective, one can focus on optimizing the approach to be more efficient for drift detection in streaming data or use existing methods in scikit-multiflow such as ADWIN to automatically adapt the sliding window size. One major improvement of this work would be to update properly the anomaly detection model by taking into account previous anomaly detectors and observations instead of discarding them completely. This can be done using adaptive learning approaches.

Another future work is to implement in scikit-multiflow a novel anomaly or isolation based methods to reduce the running time complexity of original Isolation Forest approach. Implementing a method to justify and interpret anomalies for data streams using trees based algorithms is a promising work as well as in many industries models recommendations need to be justified or understood by the final users to be implemented in production environments. These improvements could then be implemented on top of scikit-multiflow framework in order to provide new open source methods for anomaly detection in streaming data.

Acknowledgements. We would like to thank Albert BIFET from the University of Waikato and Télécom Paris for his insightful discussions about Big Data Streams mining, Adrien CHESNAUD and Zhansaya SAILAUBEKOVA for their contributions on the code, and Fabrice LE DEIT from BNP Paribas IT Group for supporting the project.

References

1. Aggarwal, C.C.: Outlier Analysis, 2nd edn. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-47578-3
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases-Volume 29, pp. 81–92. VLDB Endowment (2003)
3. Angiulli, F., Fassetti, F.: Detecting distance-based outliers in streams of data. In: Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, pp. 811–820. ACM (2007)
4. Assent, I., Kranen, P., Baldauf, C., Seidl, T.: AnyOut: anytime outlier detection on streaming data. In: Lee, S., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA 2012. LNCS, vol. 7238, pp. 228–242. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29038-1_18
5. Behera, R.K., Das, S., Jena, M., Rath, S.K., Sahoo, B.: A comparative study of distributed tools for analyzing streaming data. In: 2017 International Conference on Information Technology (ICIT), pp. 79–84 (2017)
6. Bifet, A., Morales, G.D.F.: Big data stream learning with SAMOA. In: 2014 IEEE International Conference on Data Mining Workshop, pp. 1199–1202, December 2014. https://doi.org/10.1109/ICDMW.2014.24
7. 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/
8. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010). http://portal.acm.org/citation.cfm?id=1859903
9. Bifet, A., Maniu, S., Qian, J., Tian, G., He, C., Fan, W.: StreamDM: advanced data mining in spark streaming. In: International Conference on Data Mining Workshops (ICDMW). IEEE, November 2015. https://doi.org/10.1109/ICDMW.2015.140. https://hal.inria.fr/hal-01270606
10. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. (CSUR) **41**(3), 15 (2009)
11. Ding, Z., Fei, M.: An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proc. Vol. **46**(20), 12–17 (2013). https://doi.org/10.3182/20130902-3-CN-3020.00044
12. Dua, D., Graff, C.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
13. Gomes, H.M., et al.: Adaptive random forests for evolving data stream classification. Mach. Learn. **106**(9), 1469–1495 (2017). https://doi.org/10.1007/s10994-017-5642-8
14. Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: a survey. IEEE Trans. Knowl. Data Eng. **26**(9), 2250–2267 (2014)
15. Hariri, S., Kind, M.C., Brunner, R.J.: Extended isolation forest. arXiv preprint arXiv:1811.02141 (2018)

16. Hodge, V., Austin, J.: A survey of outlier detection methodologies. Artif. Intell. Rev. **22**(2), 85–126 (2004)
17. 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)
18. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection. ACM Trans. Knowl. Discov. Data (TKDD) **6**(1), 3 (2012)
19. Manapragada, C., Webb, G.I., Salehi, M.: Extremely fast decision tree. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018. Association for Computing Machinery (2018). https://doi.org/10.1145/3219819.3220005
20. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-Multiflow: a multi-output streaming framework. J. Mach. Learn. Res. **19**(72), 1–5 (2018). http://jmlr.org/papers/v19/18-251.html
21. Pokrajac, D., Lazarevic, A., Latecki, L.J.: Incremental local outlier detection for data streams. In: 2007 IEEE Symposium on Computational Intelligence and Data Mining, pp. 504–515. IEEE (2007)
22. Salehi, M., Rashidi, L.: A survey on anomaly detection in evolving data: [with application to forest fire risk prediction]. ACM SIGKDD Explor. Newslett. **20**(1), 13–23 (2018)
23. Staerman, G., Mozharovskyi, P., Clémençon, S., d'Alché Buc, F.: Functional isolation forest (2019)
24. Tan, S.C., Ting, K.M., Liu, F.T.: Fast anomaly detection for streaming data. In: IJCAI (2011)
25. Tellis, V.M., D'Souza, D.J.: Detecting anomalies in data stream using efficient techniques: a review. In: 2018 International Conference ICCPCCT. IEEE (2018)
26. Thakkar, P., Vala, J., Prajapati, V.: Survey on outlier detection in data stream. Int. J. Comput. Appl. **136**, 13–16 (2016)
27. Togbe, M.U., Chabchoub, Y., Boly, A., Chiky, R.: Etude comparative des méthodes de détection d'anomalies. Revue des Nouvelles Technologies de l'Information Extraction et Gestion des Connaissances, RNTI-E-36, pp. 109–120 (2020)
28. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools and Techniques, 4th edn. Morgan Kaufmann, Amsterdam (2017)
29. Yamanishi, K., Takeuchi, J.I., Williams, G., Milne, P.: On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. Data Min. Knowl. Disc. **8**(3), 275–300 (2004)