# An Effective Ensemble-based Framework for Outlier Detection in Evolving Data Streams

Asmaa F. Hassan, Sherif Barakat, Amira Rezk
Department of Information Systems
Faculty of Computers and Information
Mansoura University
Cairo, Egypt

***Abstract*—In the last few years, data streams have drawn lots of researchers' attention due to their various applications, such as healthcare monitoring systems, fraud and intrusion detection, the internet of things (IoT), and financial market applications. A data stream is an unbounded sequence of data continually generated over time and is prone to evolution. Outliers in streaming data are the elements that significantly deviate from the majority of elements and then have to be detected as they may be error values or events of interest. Detection of outliers is a challenging issue in streaming data and is one of the most crucial tasks in data stream mining. Existing outlier detection methods for static data are unsuitable for use in data stream settings due to the unique characteristics of streaming data such as unpredictability, uncertainty, high-dimensionality, and changes in data distribution. Thus, in this paper, a novel ensemble learning framework called Ensemble-based Streaming Outlier Detection (ESOD) is presented to perfectly detect outliers over streaming data using a sliding window technique that is updated in response to the incoming events from the data streaming environment to overcome the concept evolution nature of streaming data. The proposed framework has three phases, namely the training phase, testing/offline phase, and outlier detection/online phase. A detection weighted vote technique is used to determine the final decisions for potential outliers. In the extensive experimental study, which was conducted on 11 real-world benchmark datasets, the proposed framework was assessed using many accuracy metrics. The experiment results showed that the proposed framework beats many other state-of-the-art methods.**

***Keywords*—Outlier detection; data streams; data stream mining; ensemble learning; concept evolution***

## I. Introduction

Due to the recent advances in both software and hardware, many applications are generating streaming data, such as sensor networks, financial markets, real-time video monitoring, internet traffic, and medical data. The term "data stream" refers to a collection of temporally ordered, massive, usually arriving at a high rate, and potentially infinite data objects. A data stream can be formalized as $DS_t = \{x_{1,t}, x_{2,t}, x_{3,t}, ..., x_{N,t}\}$, where $x_{i,t}$ is the element number $i$ at time $t$, and it has a set of high-dimensionality attributes or features. Through its large volume, it is therefore difficult to fully store data streams in memory and scan them several times [1]. Data streams differ from static data where they have some unique characteristics such as *concept-evolution*, *concept-drift*, and *feature-evolution*. In particular, concept-evolution happens when new classes emerge in streams, concept-drift occurs when the distribution of data points shifts over time, and feature-evolution occurs when the feature set of data streams changes over time [2]. Data Stream Mining (DSM) is a new approach for extracting important information from data streams [3]. Hence, the traditional data mining techniques are not applicable to processing data streams because of the special characteristics of streaming data as shown in Table I.

Data streams, like traditional data, may have outliers, or data points that are considerably different from the bulk of data points [4]. They can be noise data points or interesting instances and have to be detected in many cases to achieve better performance and accuracy. Outlier detection is one of the most important data mining tasks for detecting unusual and anomalous data points or sequences hidden in a dataset [5]. However, outlier detection over data stream datasets completely differs from traditional data ones because it must be performed under only one pass, the available memory is limited, real-time response, and the concept-evolution nature of streaming data. In real life, outlier detection has a variety of essential applications, such as detecting credit card fraud; intrusion detection in computer networks or cybersecurity; system fault diagnosis in industry; early disease detection in the health care sector, etc. [6]. Outlier detection techniques may be based on one of the following learning methods: unsupervised, semi-supervised, supervised, or ensemble learning. The unsupervised learning technique does not need training data to build the model, while the semi-supervised learning technique combines a small collection of labelled data with a large dataset of unlabeled training data. In the supervised learning approach, it requires labelled training data availability [4]. On the other hand, the ensemble learning approach requires a group of multiple trained classifiers learning algorithms to detect outliers in order to improve the detection accuracy. In particular, a number of different base learners are used in an ensemble model, which is normally much stronger than all standalone base learners because it has the ability to improve the performance of weak learners [5]. For this purpose, this paper presents an effective outlier detection approach based on the ensemble learning technique where the iForestASD, decision tree, and Adaptive Random Forest (ARF) classifiers are used as the base learners to build an ensemble-based model over streaming data using the sliding window fashion with the objective of improving detection performance while decreasing detection time consumption. In the data streams environment, the sliding

window is a time-based streaming model which is generally used to effectively process the flow of streaming data through dividing these streams into several windows, as shown in Fig. 1, where each window has an equal pre-defined size *(w)* in time *(t)*. The size of a window may be specified in terms of time points or the number of recent objects. Hence, each window maintains only recent objects, while older ones are discarded, and all objects within the active sliding window have the same importance.

TABLE I. COMPARISON BETWEEN STREAMING DATA AND STATIC DATA

| Characteristic | Streaming Data | Static Data |
|---|---|---|
| ***Volume*** | Infinite | Finite |
| ***Type of Data*** | Heterogeneous | Homogeneous |
| ***Scanning Time*** | Single pass | Multiple passes |
| ***Data Processing*** | Real-time processing | Offline processing |
| ***Data Storage*** | Aggregated and summarized data only | Raw data |
| ***Concept*** | Evolving | Static |
| ***Type of Result*** | Approximate result | Accurate result |
| ***Temporal and Spatial Contexts*** | Important aspects | May be considered for certain applications |
| ***Space and Time Complexity*** | Strict | Not strict |

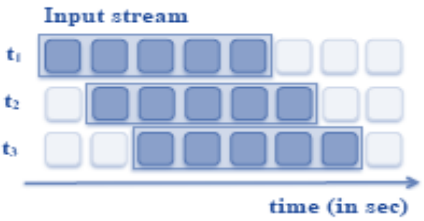



Fig. 1. The Sliding Window Time-based Model.

To detect outlier values, a combination of different base detectors; iForestASD, decision tree, and adaptive random forest algorithms; were chosen to develop an ensemble-based learning framework, which has never been explored before in the literature. The iForestASD model can be retrained more quickly than models trained with higher complexity like SVM. IForestASD allows you to retrain the model with a sliding window so that detection performance does not degrade even with concept evolution. While the decision tree technique does not need data normalization or scaling, and it has a shorter training period. Furthermore, random forest has high learning performance with good data visualization for high dimensional data and does not require hyper-parameter tuning or tree pruning. Moreover, it enables rapid detection model training even with limited computational resources. This is the main motivation for choosing this combination of base learners to construct our ensemble-based learning model for streaming outlier detection. In streaming data, the proposed framework (ESOD) attempts to detect both outliers and new class concepts. ESOD provides a more effective solution to this problem by integrating active learning in a supervised approach to detect any novel concepts and outliers in the streaming environment. The main research contributions and novelty are summarized in the following points:

- Proposing an effective Ensemble-based Streaming Outlier Detection (ESOD) framework for accurately detecting outliers in the context of streaming data adapting to the concept drift nature of streaming data.
- Building an ensemble-based model for the proposed framework using some heterogeneous machine learning algorithms as base learners, such as iForestASD, decision tree, and adaptive random forest, and then selecting the best combination.
- Using the sliding window time-based technique in the ESOD framework to process the flow of streaming data.
- Reducing computation time by allowing parameter tuning and comparing the outcomes of multiple algorithms in a single trial.
- Providing a detection weighted vote technique to report the potential outliers that outperforms the standalone base learners and state-of-the-art techniques in terms of accuracy, precision and other evaluation metrics.

The remainder of this paper is structured in the following way: related work on outlier detection techniques in streaming data is introduced section II. Section III explains the preliminaries of the used base learners. Section IV introduces the proposed framework. Section V discusses the experimental evaluations and the model development details. Finally, Section VI summarizes the proposed study and the next directions are given.

## II. RELATED WORK

Many attempts in the literature have been devoted to the outlier detection problem over streaming data. Nevertheless, many of these detection approaches work in batch mode, in which all data points are stored and many passes can be made over the data [7-9]. Several batch outlier detection algorithms have been adapted for data stream outlier detection, according to a review in [10]. These methods, while applicable to streaming data, are inefficient because they do not take into consideration the special features and characteristics of data streams. The outlier detection approaches in the streaming data context may be divided into three main categories, which are statistical-based, clustering-based, and classification-based methods [11, 12]. The first attempts at outlier detection were statistical-based approaches based on defining a model to represent the normal behavior of the data [13]. If an incoming data point does not fit the model or has an extremely low probability of fitting the model, it is termed an outlier. In [14], The authors proposed an algorithm called UKOF for top-n local outlier detection based on the kernel density estimation (KDE) model over large-scale high-volume data streams, in which they defined a KDE-based outlier factor (KOF) to measure the local outlierness score, as well as upper bounds of the KOF and an upper-bound-based pruning strategy to reduce the search space. Although this method had a low computational cost, it assumed a stable distribution of data, which is inappropriate in a data stream setting. In recent

research [15], a maximal weighted frequent-pattern-based outlier detection approach (MWFP-Outlier) was proposed to detect potential outliers from uncertain data streams. This approach had two phases, namely pattern mining phase and an outlier detection phase. The MWFP-Mine approach was proposed in the pattern mining phase to mine maximal weighted frequent patterns using the list structure, tree structure, and pruning strategies by fully considering the existential probabilities and weights for each pattern. During the outlier detection phase, four deviation indices were created to measure the degree of deviation, and then the top k ranked transactions were reported as potential outliers. It is unclear, however, how it dealt with outliers and fast processing.

The clustering-based methods are one of the common unsupervised methods used in outlier detection problems, which creates data clusters that represent the underlying data distribution [16]. Clustering-based methods divide the data into some clusters according to the similarity between the data points. Outliers are observations that are remote from clusters or clusters with significantly fewer data points. The clustering methods are categorized into distance-based and density-based. In 2019 [17], Tran et al. proposed three distance-based algorithms using the micro-clusters concept, and they claimed that the proposed algorithms had a reasonable processing time and a low space cost. Regarding the density-based methods, there are some state-of-the-art methods developed to detect outliers in data streams, such as [18–20]. Another notable attempt was presented in [21], a method called LiCS was introduced to detect outlier that classifies the samples using K-nearest neighbors of each node. Most recently, [22] proposed an incremental local density and cluster-based outlier factor method for detecting outliers in streaming data called iLDCBOF. The proposed method combined density-based spatial clustering of applications with noise (DBSCAN) and incremental versions of the local outlier factor (LOF), but it suffered from excessive computations. On the other hand, classification-based methods may either be incremental single model or ensemble classifier techniques, where the classification output is a function of the predictions from different classifiers. In addition, streaming ensemble-based methods have been developed to deal with the high dimensionality of data streams, and these techniques have great success in the outlier detection and prediction domains because of their accurate results and higher efficiency in both performance and resource consumption. Ensemble-based techniques integrate the results of many base models to create a more robust model that can detect outliers more effectively. Masud et al. [23] proposed a hyper outlier detection model for streaming data. They utilized the k-NN classifier first, followed by the SVM polynomial kernel classifier, and a data point is reported as an outlier if it is identified as an outlier by at least one of the classifiers. Then, to distinguish between novel and outlier data points, a neighborhood silhouette coefficient is applied. This approach addresses the novelty detection in multi-class underlying concepts, yet it requires all data chunks to be labelled to define the new concept. Wang et al. [24] proposed a model for detecting anomalies based on a matrix of uncompressed data against a matrix of compressed data, and this model utilized the original uncompressed data while considerably reducing computing costs. In [25], a new ensemble approach called RED-PSO was presented, which is suited to the drift notion of a data stream in the classification of non-stationary data streams. In [26], the authors proposed a sliding window-based ensemble approach for streaming outlier detection where a combination of clustering algorithms were utilized to construct clusters, which were later used in a one-class classification to identify outlier data. Togbe et al. [12] afforded iForestASD, i.e., an alternative isolation forest for streaming data implementation. They built the iForestASD on the scikit-multiflow framework; it is a machine learning framework for streaming data that is free and open source. In 2021 [27], they modified their implementation of iForestASD to address the idea of data stream drift, proposing three algorithms for drift detection: ADWIN, KSWIN, and finally extending KSWIN to deal with n-dimensional data streams. Another recent study was presented in [28], where the authors proposed an ensemble-based outlier detection framework for high-dimensional data named Average Selection and Ensemble of Candidates for Outlier Detection (ASEC-OD). The proposed framework selected the most effective base outlier detectors, which have the highest performance. To summarize, none of these approaches can detect outliers accurately and efficiently in streaming data environments while maintaining high performance rates. Our approach differs from the aforementioned outlier methods in two ways. First, the proposed framework integrates outlier detection and the concept evolution phenomenon of streaming data by considering concept evolution for new outlier or inlier concepts that appear in incoming streams. It can sequentially update the outlier detection model in the case of concept evolution, resulting in a more robust outlier detection system that is matched with a streaming scenario. Second, it takes advantage of each heterogeneous base classifier in building an ensemble model to achieve the highest rates of performance metrics.

## III. Base Learners

The outlier ensemble learning method will be very effective when heterogeneous base classifiers of diverse types are used. Thus, the distinct features of data can be identified or learned due to the variation between classifiers [29]. To boost efficiency, the proposed ensemble model constructs a set of models from three heterogeneous base learners. The chosen base learners (isolation forest, its variant iForestASD, decision tree, random forest, and its variant adaptive random forest) of the proposed framework are briefly introduced in this section.

### *A. Isolation Forest (iForest) and IForestASD*

Isolation Forest (iForest) is a famous tree-based approach to outlier detection to isolate outlier instances [30]. The random selection of an attribute iteratively creates an isolation tree *(itree)* until each data point is isolated. Isolation forest is based on the notion that it is simple to isolate an outlier but more complex to characterize an inlier data point. Accordingly, the isolation forest consists of all the isolation trees constructed from its training set subsampling. The outlier score $s(x, n)$, equation (1), is calculated by calculating the path length $h(x)$ from the root to an instance $x$ in an isolation tree constructed from $n$ data instances [31].

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (1)$$

where *E(h(x))*; which is defined by equation (2); is the average path length *h(x)* of *x* over an isolation trees collection, i.e., *itrees*.

$$E\big(h(x)\big) = \frac{\sum_{i=1}^{t} h_i(x)}{t} \quad (2)$$

where *c(n)* represents the average path length of the binary search tree's unsuccessful search process. Finally, equation (3) calculates *c(n)*, which is used to normalize an outlier score.

$$(n) = 2H(n-1) - (2(n-1)/n) \quad (3)$$

where *H(i)* is formulated by equation (4).

$$H(i) = \ln(i) + \gamma \text{ , } \gamma \text{ is the Euler's constant} \quad (4)$$

Therefore, *x* is considered as outlier data if the outlier score is close to 1, but it is considered as normal data if the score is less than 0.5.

On the other hand, the iForestASD [32] is an alternative version of the iForest method and is suitable for outlier detection over streaming data. The phrase "iForestASD" is an abbreviation for Isolation Forest Algorithm for Stream Data, where it uses the sliding window mechanism to process streaming data. In addition, iForestASD implemented the standard isolation forest technique to build the random forest. The IForestASD technique was developed to deal with the concept drift of the data stream by keeping a pre-defined anomaly rate threshold *(u)*. Fig. 2 displays the workflow of the iForestASD approach to updating the model. If the anomaly rate in the active window is higher than u, then concept drift has occurred. The iForestASD then deletes the current detector and constructs another detector using all the data in the active window. Otherwise, there was no concept drift. The trained anomaly detector was then would not change.

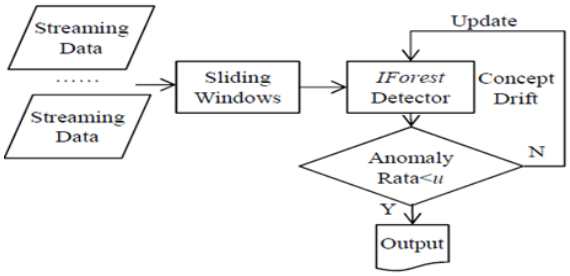



Fig. 2. The iForestASD Workflow. The Image is Reproduced from [32].

### B. Decision Tree (DT)

Decision trees are popular tree-based classification methods where each non-leaf node represents an attribute, whereas the edge links between its child nodes represent the attribute values and the leaf node represents a class label. In general, a decision tree is built in three main steps:

- Select the best features of the training dataset to build the root node.
- Split the training dataset into nodes using the Gini index.
- Loop until no further splitting can be done.

Information gain, gain ratio, and Gini index are used to partition the training dataset [33]. In this research, the Gini index is used as the DT splitting criterion, which is calculated by equation (5).

$$G(X) = 1 - \sum_{i=1}^{n} p_i^2 \quad (5)$$

where $p_i$ is the proportion of the $i^{th}$ class samples of the feature *X*.

### C. Random Forest and Adaptive Random Forest

Firstly, the classical random forest (RF) [34] method is a tree-based approach that can be used for both classification and prediction tasks. The prediction of a new data point is determined by aggregating the predictions of *n* trees in classification tasks. Random predictors are chosen to split a node in random forests; therefore, every tree is grown based on a different random data sample. After multiple models have been generated, the overall trees of the model contribute to a weighted decision or vote to produce an overall determination. Regarding randomness splitting of the data, RF has two main sources of randomness which are bootstrap aggregating or bagging and boosting. In bootstrap aggregating, each classifier is trained on random sampling with replacement from the original dataset. On the other hand, in boosting approach, instead of evaluating all features at each tree split, only a random set of samples is used to make the eventual decision. RFs can model high dimensional data and they can handle missing values.

Adaptive Random Forest (ARF) is one variant technique that adapts the standard random forest algorithm [35]. It produces decision trees by training them on resampled varieties of the original data and by randomly picking up a small number of features that can be inspected at each node for splitting. ARF offers data visualization of high dimensional data and it has a high accuracy and resistance to overfitting.

## IV. Proposed Ensemble-based Framework

In this section, a novel ensemble-based outlier detection framework for data streams called Ensemble-based Streaming Outlier Detection (ESOD) is introduced. The proposed framework utilizes three heterogeneous machine learning algorithms, which are iForestASD, decision tree (DT), and adaptive random forest (ARF) as the base learners to build the ensemble model for detecting outliers in streaming data based on a weighted voting technique. To enhance efficiency, the proposed framework employs various base learners to build an ensemble-based model. Ensemble-based Streaming Outlier Detection (ESOD) has three main phases, as shown in Fig. 3, namely the training phase, testing/offline phase, and outlier detection/online phase. These phases are detailed in depth in the subsections that follow.

### A. The Training Phase

The ESOD training phase is used to prepare data and tune the hyper-parameters of each classifier for the finest outcome. It consists of a series of steps that must be completed in the following order:

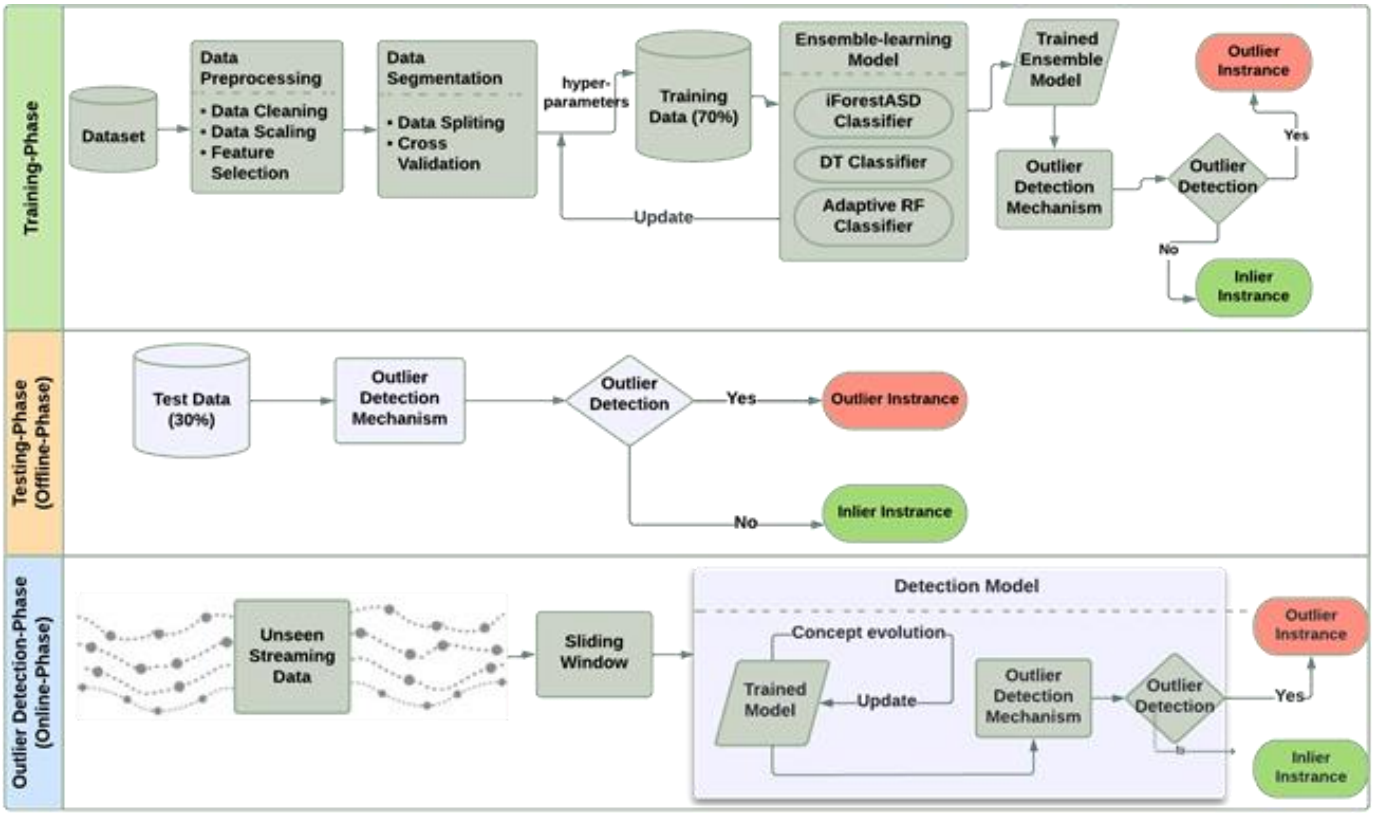



Fig. 3. The Ensemble-based Streaming Outlier Detection (ESOD) Block Diagram.

*1) Data preprocessing step:* Certain data preprocessing techniques are done in order to turn raw data streams into meaningful patterns. The preprocessing stage is a critical step for the proper representation of streaming data. Hence, many different preprocessing techniques are employed, such as the removal of missing values, min-max scalar and one-hot encoding, for a more efficient classification process. There are three further phases in the ESOD's preprocessing stage, which are the *data cleaning*, *data scaling*, and *feature selection* phases. In the *data cleaning* phase, missing data is first examined and eliminated if it exists. The *data scaling* phase then employs the one-hot encoding approach on nominal or non-binary categorical data, and the min-max scaler is employed to scale values into the range of 0-1. Finally, the *feature selection* phase, where the main goal is to extract the set of relevant and non-redundant features, makes the learning model more meaningful and quicker because not all features are important for the model's learning process. In this study, the Least Absolute Shrinkage and Selection Operator (LASSO) technique is used to choose the most important and correlated features from the feature space that have a strict reflection on the target.

*2) Data segmentation step:* In the data segmentation step, the preprocessed data *D* of n selected features is divided into two groups: training set $X_{train}$ with *m* data points of ratio 70% where $X_{train} \in D^{m \times n}$ and testing set $X_{test}$ with r data points of ration 30% where $X_{test} \in D^{r \times n}$. The testing set, in contrast to the training set, should be unlabeled. Then, the K-fold cross validation (CV) method is used, in which the training set is divided into *k* equal non-overlapping subsets, determined by random sampling or the bootstrap mechanism, and the model is tested on different subsets of the dataset. Fig. 4 depicts the K-fold cross validation process. If $k = 10$, for example, nine groups of the sample data are used to train the model and only one group is used to test the model at each fold, where the model parameters with the lowest mean squared error from the *k* training models are chosen for the final model. For each fold, the outlier detection models are trained with only data from the inlier class, so the model is trained *k* times. The k-fold cross validation approach is used in this study to choose the model hyper-parameters for setting each model as well as the configured models.

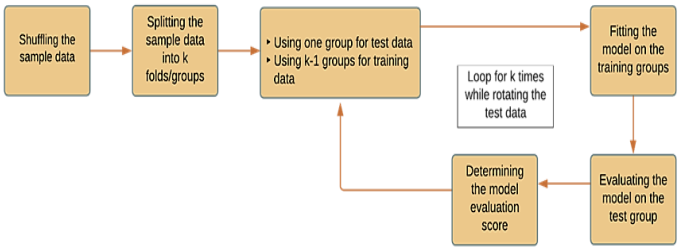



Fig. 4. The K-Fold Cross Validation Process.

*3) Ensemble-learning model step:* The heterogeneous base classifiers $B = \{\beta_1, \beta_2, \ldots, \beta_i\}$; iForestASD, decision tree, and ARF classifiers are built in this step, employing many hyper-parameter settings that are continually updated to

acquire the optimal parameters of each classifier. All three classifiers were trained using *K* random data samples ($s_1$, $s_2$,… $s_K$) and were then used to classify outliers of the test set $X_{test}$ in the testing phase. Specifically, the outlier detection approach ESOD uses similar insights as in the random forest approach, where all these base learners are considered as trees for the ensemble model. In more detail, the outlier detection final decision is made by the weighted voting of these trees where it is the class label that has the highest weighted vote of classifiers predicted. After constructing the base learning models, the ensemble learning model $\psi_{\beta1,\ \beta2,..,\ \beta i}$ tries to combine their outlier results to declare the final decision by estimating the weights of individual base learning models $\beta_1$, $\beta_2$, … , $\beta_i$. to aggregate their results with a weighted combination. The ESOD weighted voting approach involves assigning various weights to each classifier depending on its performance accuracy [36], calculated by equation (6), and voting on the classifiers based on this weight. After calculating the weight for each classifier, the most votes of classification results are selected as the ensemble's final detection result.

$$\omega_{\beta i} = \frac{A_{\beta i}}{\sum_i^3 A_i} \quad (6)$$

where $\omega_{\beta i}$ is the weight of classifier $\beta_i$, $A_{\beta i}$ is the classifier $\beta_i$ accuracy, and $A_i$ is the accuracy summation of the three base classifiers used in the ensemble model. The trained ensemble model has been constructed at this point and can classify a data point as an inlier or outlier.

### B. The Testing/Offline Phase

The test set was made up of the remaining 30% of the whole dataset. This test data is used as a final evaluation of an unseen dataset in the offline phase to ensure that the trained model was appropriately trained and predicts correctly. Hence, any unseen test sample *x* is classified into inlier or outlier instances using these trained base classifiers based on the above detection voting technique.

### C. The Outlier/Online Detection Phase

During the outlier detection phase, i.e., the online phase, a flow of fresh unseen data streams is generated and processed using the sliding-time window technique. As a result of the ever-changing nature of stream data, detection mechanism updates are frequently based on time intervals in the form of sliding windows, where current items are given higher priority during the detection phase than older ones. The trained model can classify the known classes on which it was trained, identify any novel classes, and finally update the model with the most recent data to include any detected outliers. The model, in particular, tries to classify incoming samples of streaming data into known classes, which are outliers or inliers, or novel patterns. Hence, it predicts class labels for instances that belong to known classes, and the novel patterns are identified as "*concept evolution*". Once the new class is detected, it is integrated into the model for detecting repeated patterns. Thus, the trained model is automatically updated to handle the concept evolution by comparing the mean difference of two adjacent sub-windows [37].

## V. Experimental Evaluation

This section presents the datasets, experimental setup, and parameter fine-tunning process followed by the assessment metrics and a brief discussion of the results.

### A. Dataset Description

This research uses 11 publicly available outlier benchmark datasets (http, Credit-card-fraud, smtp, Annthyroid, Thyroid, Cardio, Pima Diabetes, Breast-cancer, Arrhythmia, Heart Disease, Hepatitis) from the UC Irvine Machine Learning Repository [38] for the evaluation of the base learner classifiers, state-of-the-art methods, and proposed framework. Description of these datasets is summarized in Table II by name, number of points, dimensions, outlier samples, percent of outliers, outlier class, and application domain of the datasets.

TABLE II. Description of the Benchmark Datasets used in the Performance Evaluation

| Dataset | Samples (m) | Features (n) | Outliers (%) | Outlier Class | Application Domain |
|---|---|---|---|---|---|
| **http** | 567,479 | 22 | 2,211 (0.39%) | Attack | Intrusion detection |
| **Credit-card-fraud** | 284,807 | 30 | 492 (0.17%) | Fraudulent | Fraud detection |
| **smtp** | 95,156 | 22 | 29 (0.03%) | Attack | Intrusion detection |
| **Annthyroid** | 7,200 | 6 | 534 (7.42%) | Hyperfunction and Subnormal functioning | Disease detection |
| **Thyroid** | 3,772 | 6 | 93 (2.47%) | Hyperfunction and Subnormal functioning | Disease detection |
| **Cardio** | 1,831 | 21 | 176 (9.61%) | Pathologic | Disease detection |
| **Pima Diabetes** | 768 | 9 | 193 (3.98%) | Diabetes | Disease detection |
| **Breast-cancer** | 683 | 30 | 239 (34.99%) | Malignant | Disease detection |
| **Arrhythmia** | 452 | 279 | 66 (14.60%) | The smallest classes (3, 4, 5, 7, 8, 9, 14, 15) | Disease detection |
| **Heart Disease** | 297 | 13 | 137 (46%) | Class 0 | Disease detection |
| **Hepatitis** | 155 | 19 | 32 (20.65%) | Die | Disease detection |

TABLE III. HYPER-PARAMETER SETTINGS FOR DIFFERENT BASE LEARNER CLASSIFIERS

| Classifier | Hyper-parameters Values |
|---|---|
| **iForestASD** | n_estimators = 100<br>contamination = 0.2<br>bootstrap = True<br>n_jobs = -1<br>random_state = 42 |
| **Decision Tree (DT)** | criterion = gini<br>n_estimators = 100<br>Min_Samples_Split = 2<br>Splitter = best<br>max_features = log2<br>max_depth = 6 |
| **Adaptive Random Forest (ARF)** | n_models = 100<br>max_features = sqrt<br>aggregation_method = mean<br>lambda_value = 6<br>drift_detector = ADWIN<br>warning_detector = ADWIN<br>δw (warning threshold) = 0.01<br>δd (drift threshold) = 0.001 |
| **K-nearest neighbors (K-NN)** | n_neighbors = 7<br>weights = distance<br>metric = Euclidean |
| **Logistic Regression (LR)** | Penalty = l2<br>solver = saga<br>dual = True<br>tol = 0.01<br>random_state = 42<br>max_iter = 1000 |
| **SVM** | Regularization parameter = 1<br>kernel = sigmoid<br>gamma = scale<br>shrinking = True<br>coef0 = 0.01 |
| **GaussianNB (GNB)** | priors = none<br>var_smoothing = 0.001 |
| **Artificial Neural Network (ANN)** | hidden_layer = 4<br>activation = relu<br>optimizer = adam<br>batch_size = 32<br>loss = binary_crossentropy<br>epochs = 100 |
| **Gradient Boosting (GB)** | learning_rate = 0.1 n_estimators=100<br>min_samples_split = 500<br>min_samples_leaf = 50<br>max_depth = 5<br>max_features = sqrt<br>subsample = 0.8 |
| **Local Outlier Factor (LOF)** | n_neighbors = 7<br>algorithm = ball_tree<br>leaf_size = 30<br>metric = minkowski<br>p = 2 |

### *B. Experimental Setup*

All the developed models are implemented in the open-source Python programming language using the Anaconda distribution API. The main libraries and packages used include pandas, NumPy, SciPy, matplotlib, and seaborn. Furthermore, the scikit-learn library [39] is the most widely used open-source Python machine learning package. Scikit-learn structures various machine learning tasks such as regression, classification, and clustering algorithms and contains random forest, decision tree, K-NN classifiers, and more. Finally, to develop algorithms and conduct experiments, we used the Scikit-Multiflow framework [40], a Python-based open-source machine learning framework for evolving data streaming.

### *C. Hyper-parameter Tuning*

The model hyper-parameters were tuned through a series of loops that iterated through changes in each relevant parameter of the given model based on the Grid Search strategy, and the model performance was assessed using stratified 10-fold cross-validation setting. Table III displays the optimal hyper-parameter values for each classifier used in the evaluation process after executing the hyper-parameter

fine-tuning method. In addition, the model performance is assessed using several ranges of sliding window size settings because of the relevance of this parameter in terms of computation time and memory use. For small datasets (Pima Diabetes, Breast-cancer, Arrhythmia, Heart Disease, Hepatitis), we used the sliding window of sizes w = 100, 120, 140, 160, 180, and 200. Whereas w = 100, 200, 300, 400, 500, 600, and 700 are the window sizes chosen for huge datasets (http, Credit-card-fraud, smtp, Annthyroid, Thyroid, Cardio).

### *D. Performance Metrics*

To assess the effectiveness of the proposed framework, the average scores of ten separate trials were used to construct the most popular performance metrics in the outlier detection area, which are accuracy, precision, sensitivity or recall, and F1-score measures. All these metrics are calculated by the following equations, equation (7) – (10). For the binary classification, there are only four possible outcomes, which are True Negative (TN), False Negative (FN), False Positive (FP) and True Positive (TP).

$$\text{Accuracy} = (TP+TN)/(TP+TN+FP+FN) * 100 \quad (7)$$

$$Recall = TP/(TP + FN) * 100 \quad (8)$$

$$Precision = TP / (TP + FP) * 100 \quad (9)$$

$$F1= 2*(\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall}) \quad (10)$$

### *E. Results and Discussion*

The proposed framework is evaluated by comparing results with its standalone base learners and other some predictive machine learning techniques such as Logistic Regression (LR), SVM, GaussianNB, K-NN, Artificial Neural Network (ANN), Gradient Boosting, and Local Outlier Factor (LOF). In addition, ESOD is compared with many state-of-the-art methods found in [14], [15], [21], [22], [26], [27], and [28]. We individually show the performance of the comparisons on the aforementioned 11 datasets using the accuracy, precision, recall, and F1-score metrics. The performance results are the average of 30 trials of independent experiments.

In the first experiments set, the efficiency of the proposed framework is evaluated using the accuracy, precision, and recall metrics with all baselines on the datasets. Table IV compares performance results of all base learners with the proposed model. From the results, it is clear that the linear regression, K-NN, SVM, and LOF perform worse than other algorithms in most cases. Although the base learners, i.e., iForestASD, decision tree, and adaptive random forest classifiers, of the proposed model independently perform very well, but the proposed framework performs better than these three standalone base learners. On the other hand, the experimental results reveal that the ESOD performs very well on the large-scale datasets such as http, credit and smtp datasets, beside its superior performance over other small datasets. For instance, ESOD achieves 98.65% and 99.64% of precision score on http and heart disease datasets, respectively. Another notable issue is that ESOD gives the highest recall or outlier detection rate, where it gets 99.78%, 99.05%, 99.66%, and 99.54% rates on http, credit, smtp, and hepatitis datasets, respectively.

Fig. 5 plots the performance comparison between different machine learning algorithms, and ESOD method in terms of F1-score. It should be noted that the K-NN, SVM, and LOF methods exhibited very low performance rates compared to ESOD and the others. On the http dataset, which is the biggest dataset, the K-NN, SVM and LOF models exhibit poor recall rates of 77.90%, 70.78%, and 61.58%, respectively, demonstrating their inability to learn enough samples to accurately classify outlier data, while ESOD achieves 99.21% on the same dataset. Furthermore, all the K-NN, SVM and LOF models get the lowest F1 rates on the Breast-cancer dataset, which has the largest outlier ratio, with 77.34%, 75.25%, and 69.59%, respectively, while ESOD can achieve the highest F1-score of 89.18%. In general, the proposed model has the best F1-score rates on all datasets.

The next set of experiments is performed to compare the proposed framework with some state-of-the-art methods for streaming outlier detection presented in [14], [15], [21], [22], [26], [27] and [28] and the results are shown in Fig 6. In more detail, from Fig. 6(a)-(k), the performance of the proposed model (ESOD) is better than its competitive methods on all benchmark datasets, where it achieves higher rates than others. For instance, ESOD has attained 97.37% accuracy, 98.65% precision, a recall of 99.78% and finally F1-score of 99.21% on the http dataset, Fig. 6(a). In contrast, other methods have achieved lower rates, which are 71.87%, 70.63%, 73.22%, 88.03%, 90.16%, 93.81%, and 84.32% of the recall measure for the methods UKOF [14], MWFP-Outlier [15], LiCS [21], iLDCBOF [22], Method in [26], Method in [27], and ASEC-OD [28], respectively. One thing is notable here is that the MWFP-Outlier and LiCS methods have the lowest performance rates on all datasets in most cases. However, the model in [27] has a slightly high rates like our proposed model, where it gains 93.81%, 91.75%, 91.83%, 91.75%, 92.43%, 91.31%, 81.82%, 87.33%, 90.27%, 93.50%, 90.70% of recall on http, credit, smtp, Annthyroid, thyroid, cardio, Pima, breast-cancer, arrhythmia, heart, and hepatitis datasets, respectively. Overall, the precisions of the seven state-of-the-art methods are also lower than the ESOD precision (61.86% - 93.52%). One can observe that the performance rates of our proposed framework is more consistent with increasing number of features on the dataset as compare to the other seven state-of-the-art methods on all datasets. On Arrhythmia dataset, which has 279 features, LiCS performs slightly better than iLDCBOF, Method in [26] and ASEC-OD. Meanwhile, UKOF obtains the lowest rates on breast-cancer dataset, which has the highest outlier ratio, but method in [26] approximately performs as good as ASEC-OD [28].

TABLE IV. EVALUATION OF ACCURACY, PRECISION AND RECALL RATES ON DIFFERENT INDIVIDUAL MODELS VS. ESOD FRAMEWORK. THE BEST AVERAGE SCORES PER EACH DATASET (COLUMN) AND MODEL (ROW) ARE SHOWN IN BOLD

| | **http** | **Credit** | **smtp** | **Annthyroid** | **Thyroid** | **Cardio** | **Pima** | **Breast-cancer** | **Arrhythmia** | **Heart Disease** | **Hepatitis** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Accuracy*** | | | | | | | | | | | |
| **iForestASD** | **95.68** | 93.05 | 95.17 | 94.15 | 95.66 | 94.33 | 86.62 | 74.89 | 83.64 | **88.60** | 87.54 |
| **DT** | 92.24 | 91.22 | **92.36** | 91.06 | 91.50 | 90.74 | 86.51 | 75.31 | 84.39 | 86.71 | 88.34 |
| **ARF** | 93.66 | 92.05 | 92.70 | 92.98 | 93.84 | **94.50** | 85.65 | 74.29 | 83.44 | 86.57 | 86.53 |
| **LR** | 71.07 | 85.15 | 85.66 | 85.70 | 86.01 | **86.30** | 84.28 | 70.99 | 83.16 | 85.72 | 86.11 |
| **K-NN** | 80.08 | 80.11 | 80.34 | 81.00 | **81.37** | 81.22 | 80.76 | 69.60 | 80.23 | 80.77 | 80.31 |
| **SVM** | 70.48 | 70.50 | 71.00 | 69.83 | 70.00 | 70.53 | 67.54 | 62.07 | 71.41 | 70.87 | **73.04** |
| **GNB** | 92.00 | **92.23** | 91.89 | 90.08 | 90.69 | 91.22 | 84.70 | 73.60 | 83.00 | 83.18 | **90.60** |
| **ANN** | 85.14 | 86.43 | 86.45 | 85.91 | 85.07 | 86.18 | 86.00 | 74.13 | 83.46 | **86.52** | 83.48 |
| **GB** | 90.06 | 90.20 | 89.04 | **92.00** | 91.25 | 90.24 | 85.16 | 73.36 | 80.33 | 85.15 | 86.02 |
| **LOF** | 75.09 | 76.00 | 75.35 | 76.48 | 74.08 | 76.63 | 75.62 | **77.09** | 76.32 | 76.36 | 76.08 |
| **ESOD** | **97.37** | **94.05** | **95.67** | **95.32** | **95.70** | **96.23** | **88.18** | **79.92** | **85.45** | 87.44 | 90.58 |
| ***Precision*** | | | | | | | | | | | |
| **iForestASD** | 97.20 | 95.18 | 93.03 | 95.21 | 97.43 | 94.64 | 85.97 | 84.24 | 94.28 | **98.70** | 96.35 |
| **DT** | **94.90** | 91.57 | 91.53 | 93.90 | 94.42 | 94.54 | 84.08 | 83.81 | 92.21 | 93.29 | 92.80 |
| **ARF** | 96.26 | 92.71 | **97.36** | 94.46 | 95.55 | 95.10 | 84.76 | 85.94 | 92.42 | 97.26 | 93.98 |
| **LR** | 80.54 | 82.50 | 82.45 | 83.67 | 81.65 | 88.92 | 81.36 | 72.67 | 88.22 | 91.24 | **92.80** |
| **K-NN** | 70.20 | 71.56 | 72.19 | 72.77 | 74.37 | 75.58 | 79.31 | 79.71 | 76.47 | 88.96 | **89.34** |
| **SVM** | 70.19 | 70.60 | 70.66 | 71.06 | 71.80 | 72.71 | 73.20 | 73.71 | **74.02** | 72.96 | 71.57 |
| **GNB** | 78.42 | 76.66 | 75.60 | 74.30 | 78.69 | 79.24 | 81.21 | **86.62** | 84.78 | 82.54 | 82.66 |
| **ANN** | 90.04 | 89.90 | 91.26 | 90.65 | 91.15 | **93.28** | 83.38 | 80.75 | 91.02 | 91.33 | 92.14 |
| **GB** | 89.16 | 90.91 | 91.51 | **92.91** | 90.72 | 92.73 | 83.88 | 82.41 | 90.78 | 92.17 | 90.49 |
| **LOF** | 51.25 | 51.36 | 51.48 | 51.79 | **64.81** | 53.33 | 54.40 | 62.06 | 58.55 | 63.93 | 62.12 |
| **ESOD** | **98.65** | **96.70** | **97.92** | **96.33** | **97.46** | **98.30** | **87.57** | **88.59** | **95.06** | **99.64** | **96.54** |
| ***Recall*** | | | | | | | | | | | |
| **iForestASD** | **98.68** | 98.09 | 94.83 | 95.44 | 94.26 | 92.30 | 89.65 | 86.23 | 95.43 | 97.55 | 98.63 |
| **DT** | 93.62 | 94.48 | 93.72 | 94.30 | 93.51 | 91.85 | 88.23 | 83.79 | 91.87 | 94.11 | **96.05** |
| **ARF** | **98.16** | 96.02 | 94.35 | 95.24 | 94.13 | 92.22 | 86.75 | 85.75 | 94.07 | 96.65 | 94.85 |
| **LR** | 80.25 | 80.51 | 81.76 | 81.78 | 81.80 | 81.94 | 81.88 | 82.33 | 82.43 | **83.60** | 83.54 |
| **K-NN** | 87.50 | 86.15 | 86.29 | **88.57** | 82.27 | 86.99 | 86.46 | 75.11 | 83.19 | 82.78 | 85.39 |
| **SVM** | 71.37 | 73.76 | 75.08 | 75.27 | 75.41 | 75.82 | 76.65 | 76.86 | **78.11** | 77.91 | 77.90 |
| **GNB** | 82.10 | 82.44 | 84.91 | 85.22 | 85.82 | 85.77 | 83.84 | 85.44 | 86.06 | 89.26 | **89.83** |
| **ANN** | 91.94 | 91.52 | **93.25** | 90.99 | 91.08 | 91.71 | 86.14 | 85.71 | 92.07 | 92.40 | 90.30 |
| **GB** | 81.81 | 82.04 | 86.74 | 82.46 | **89.60** | 85.26 | 75.06 | 82.23 | 83.94 | 88.31 | 87.39 |
| **LOF** | 77.14 | 78.55 | 78.16 | 76.99 | 76.55 | 78.71 | 71.09 | **79.19** | 73.42 | 76.77 | 72.80 |
| **ESOD** | **99.78** | **99.05** | **99.66** | **98.17** | **97.14** | **95.20** | **90.92** | **89.78** | **96.86** | **98.27** | **99.54** |

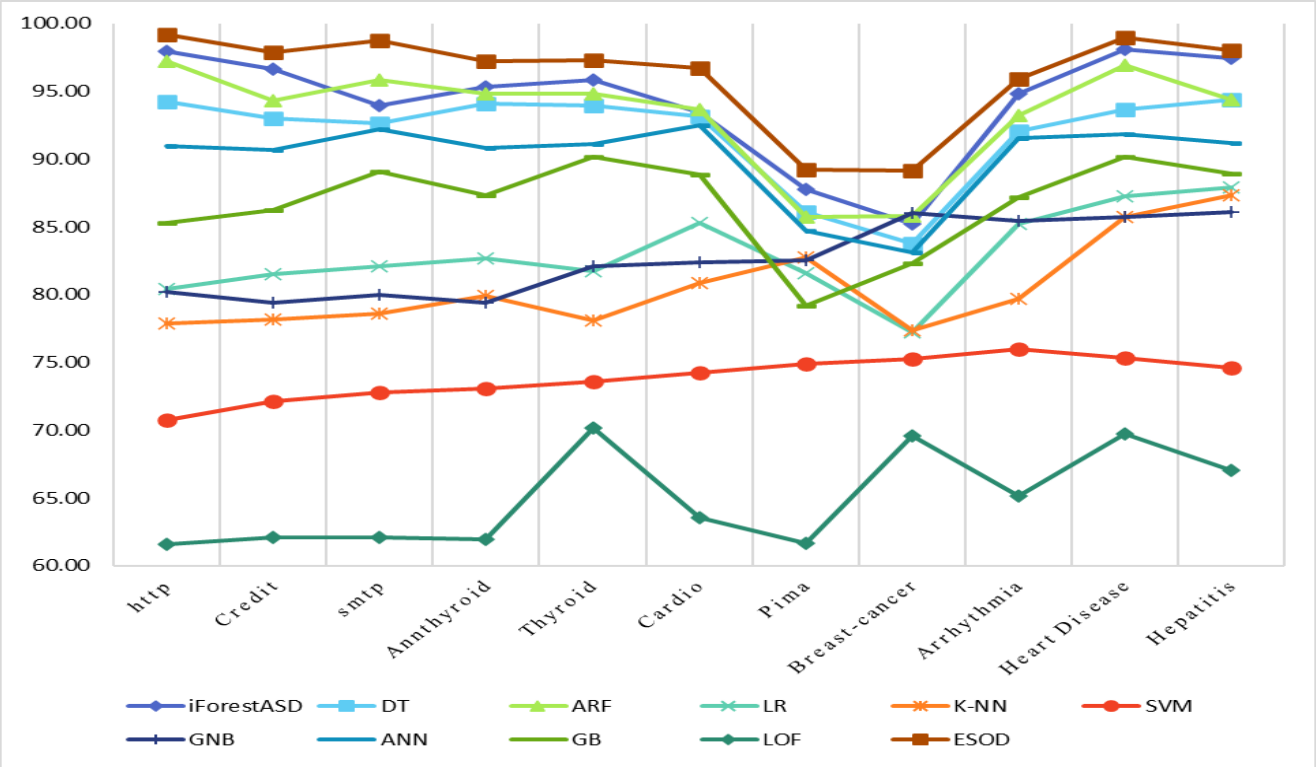
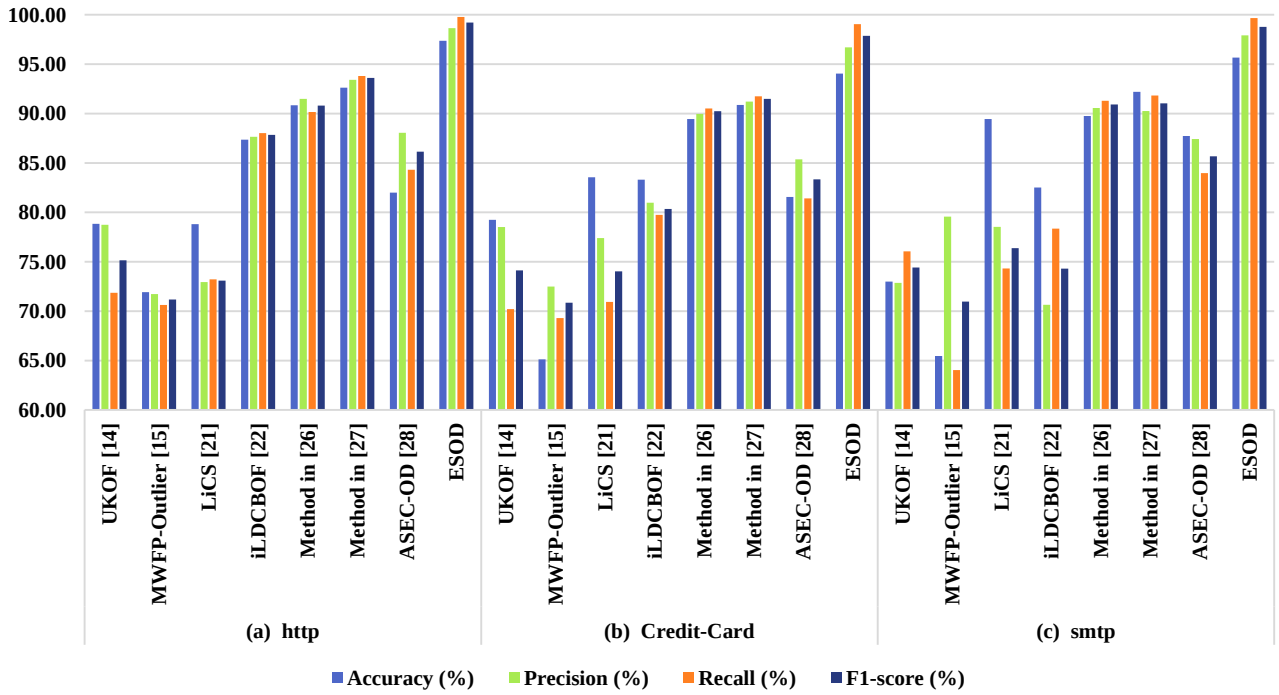



Fig. 5. F1-score Rates of different Algorithms on different Datasets.



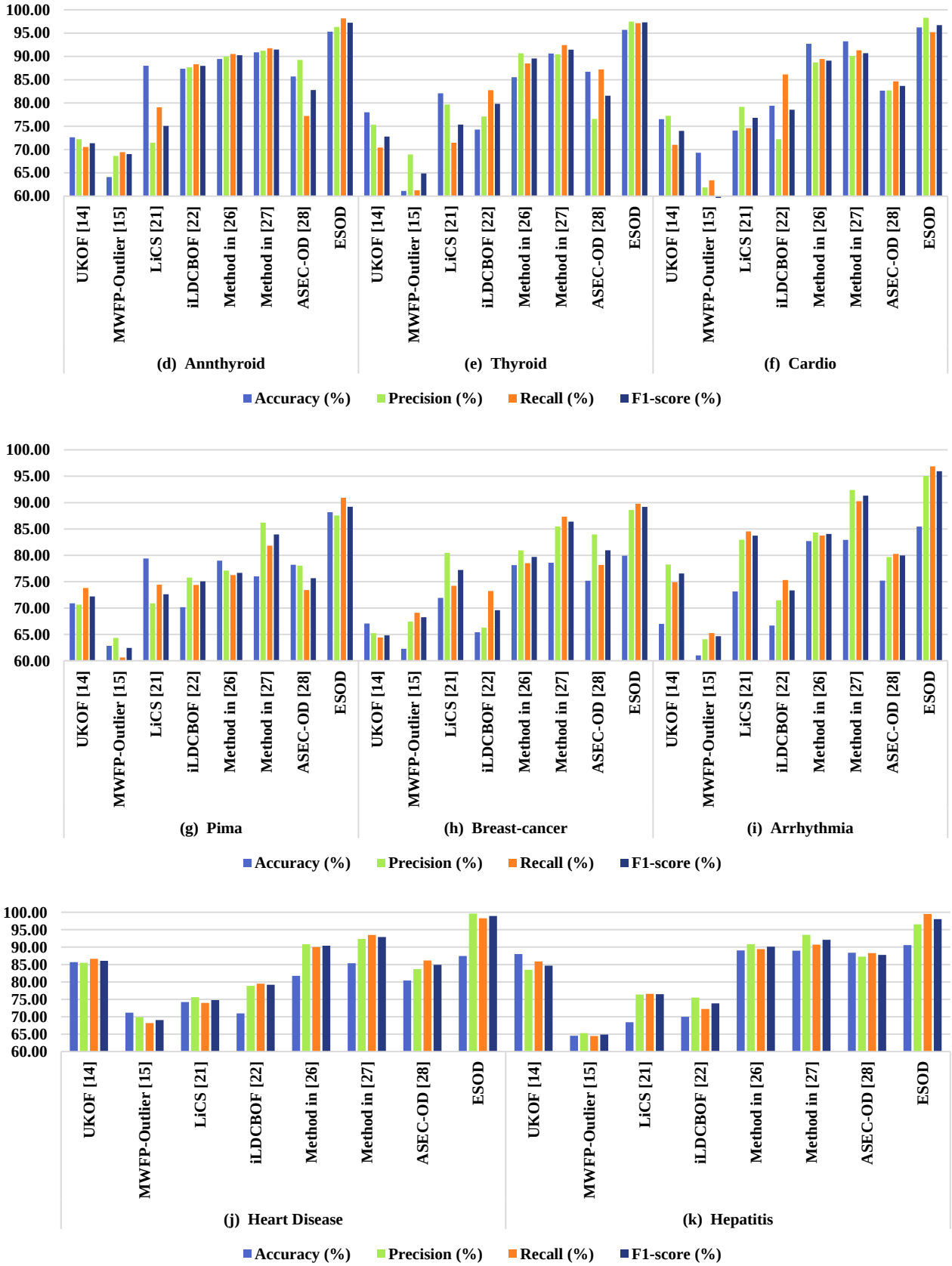



Fig. 6. Performance Comparison between State-of-the-Art Methods and the Proposed ESOD Model.

Another experiment was conducted to assess the impact of changing the window size of streaming data on the proposed framework. Table V compares the performance evaluation metrics when the window sizes were varied *(w)*. When the window size is increased to $w = 500$ for the http, credit, smtp, Annthyroid, thyroid, and cardio datasets, while $w = 180$ for the Pima, breast-cancer, arrhythmia, heart, and hepatitis datasets, the rates have improved as the F1-score grows for all datasets. However, when the window size is greater than 500 for large datasets and greater than 180 for small datasets, the F1-score falls dramatically. Furthermore, the model's performance varies significantly depending on the dataset, with ESOD earning a F1-score over 99% for the http dataset, which is the largest dataset. As a result, we expect ESOD to perform better on datasets with a high scale data.

The final experiments set is performed to evaluate the average execution time of ESOD against the other methods. Fig. 7 shows that on the http dataset, for instance, ESOD performs with an execution time of 27.46 milliseconds compared to the LiCS, which takes about 41.56 milliseconds while the iLDCBOF method takes 37.89 milliseconds. Furthermore, LiCS takes the longest time among all the investigated methods, and the proposed method executes the http dataset at almost half the execution time of LiCS. On the other hand, comparison of execution time within the Hepatitis dataset, for instance, displays that ESOD has the shortest execution time of 11.40 milliseconds as compared to the others. In addition, within the Pima dataset, the result indicates that ESOD has an execution time of 19.64 milliseconds as against UKOF which has resulted in 26.47 milliseconds of execution time. In general, the results demonstrate that the proposed method is significantly faster than all other methods in every tested case on all datasets.

TABLE V. PERFORMANCE EVALUATION OF ESOD WHEN VARYING THE WINDOW SIZE W. THE BEST AVERAGE SCORES PER EACH DATASET ARE SHOWN IN BOLD AND HIGHLIGHTED

| Dataset | Window Size (w) | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| **http** | 100 | 95.63 | 95.34 | 95.88 | 95.61 |
| | 200 | 95.72 | 95.72 | 96.12 | 95.92 |
| | 300 | 95.89 | 95.94 | 96.69 | 96.31 |
| | 400 | 96.71 | 96.06 | 96.64 | 96.35 |
| | **500** | **97.37** | **98.65** | **99.78** | **99.21** |
| | 600 | 97.11 | 97.41 | 97.83 | 97.62 |
| | 700 | 97.00 | 96.88 | 97.20 | 97.04 |
| **Credit** | 100 | 89.69 | 95.19 | 94.69 | 94.94 |
| | 200 | 90.61 | 95.27 | 94.21 | 94.74 |
| | 300 | 91.49 | 96.21 | 97.70 | 96.95 |
| | 400 | 92.99 | 96.57 | 97.82 | 97.19 |
| | **500** | **94.05** | **96.70** | **99.05** | **97.86** |
| | 600 | 93.79 | 96.66 | 98.42 | 97.53 |
| | 700 | 92.014 | 95.83 | 96.39 | 96.11 |
| **smtp** | 100 | 92.38 | 94.47 | 94.46 | 94.46 |
| | 200 | 93.47 | 95.35 | 95.24 | 95.29 |
| | 300 | 94.06 | 96.37 | 95.26 | 95.81 |
| | 400 | 95.36 | 96.78 | 98.00 | 97.39 |
| | **500** | **95.67** | **97.92** | **99.66** | **98.78** |
| | 600 | 95.28 | 97.50 | 99.58 | 98.53 |
| | 700 | 94.22 | 95.50 | 98.70 | 97.07 |
| **Annthyroid** | 100 | 92.34 | 92.65 | 92.04 | 92.34 |
| | 200 | 92.87 | 92.88 | 92.66 | 92.77 |
| | 300 | 93.91 | 93.97 | 92.85 | 93.41 |
| | 400 | 94.57 | 94.62 | 94.25 | 94.43 |
| | **500** | **95.32** | **96.33** | **98.17** | **97.24** |
| | 600 | 95.30 | 95.33 | 97.14 | 96.23 |

| | | | | | |
|---|---|---|---|---|---|
| | 700 | 95.04 | 95.21 | 96.44 | 95.82 |
| **Thyroid** | 100 | 92.44 | 95.98 | 94.08 | 95.02 |
| | 200 | 92.78 | 96.03 | 96.00 | 96.01 |
| | 300 | 93.65 | 96.14 | 96.20 | 96.17 |
| | 400 | 94.40 | 96.60 | 97.01 | 96.80 |
| | **500** | **95.70** | **97.46** | **97.14** | **97.30** |
| | 600 | 95.51 | 97.00 | 96.45 | 96.72 |
| | 700 | 94.43 | 97.00 | 96.19 | 96.59 |
| **Cardio** | 100 | 90.83 | 96.85 | 91.40 | 94.05 |
| | 200 | 91.40 | 96.93 | 92.25 | 94.53 |
| | 300 | 93.70 | 97.38 | 93.83 | 95.57 |
| | 400 | 93.63 | 98.02 | 93.89 | 95.91 |
| | **500** | **96.23** | **98.30** | **95.20** | **96.73** |
| | 600 | 94.88 | 97.15 | 95.03 | 96.08 |
| | 700 | 94.72 | 97.04 | 94.73 | 95.87 |
| **Pima** | 100 | 84.55 | 83.51 | 84.36 | 83.93 |
| | 120 | 84.69 | 84.35 | 83.66 | 84.00 |
| | 140 | 85.74 | 85.17 | 84.88 | 85.02 |
| | 160 | 87.01 | 86.58 | 86.9 | 86.74 |
| | **180** | **88.18** | **87.57** | **90.92** | **89.21** |
| | 200 | 88.04 | 85 | 86.81 | 85.90 |
| **Breast-cancer** | 100 | 74.44 | 75.22 | 75.61 | 75.41 |
| | 120 | 75.11 | 75.43 | 76.33 | 75.88 |
| | 140 | 76.53 | 76.7 | 74.94 | 75.81 |
| | 160 | 77.31 | 76.68 | 75.66 | 76.17 |
| | **180** | **79.92** | **88.59** | **89.78** | **89.18** |
| | 200 | 77.12 | 79.04 | 75.46 | 77.21 |
| **Arrhythmia** | 100 | 81.66 | 92.48 | 91.23 | 91.85 |
| | 120 | 81.56 | 92.89 | 92.94 | 92.91 |
| | 140 | 82.13 | 93.69 | 93.98 | 93.83 |
| | 160 | 84.78 | 94.44 | 95.48 | 94.96 |
| | **180** | **85.45** | **95.06** | **96.86** | **95.95** |
| | 200 | 83.02 | 94.01 | 93.66 | 93.83 |
| **Heart Disease** | 100 | 82.41 | 92.32 | 94.66 | 93.48 |
| | 120 | 83.07 | 93.14 | 96.01 | 94.55 |
| | 140 | 84.53 | 95.61 | 96.74 | 96.17 |
| | 160 | 86.18 | 97.85 | 97.37 | 97.61 |
| | **180** | **87.44** | **99.64** | **98.27** | **98.95** |
| | 200 | 87.30 | 98.10 | 98.12 | 98.11 |
| **Hepatitis** | 100 | 84.42 | 92.77 | 95.59 | 94.16 |
| | 120 | 85.61 | 92.81 | 95.62 | 94.19 |
| | 140 | 89.09 | 94.32 | 97.20 | 95.74 |
| | 160 | 89.35 | 95.05 | 98.20 | 96.60 |
| | **180** | **90.58** | **96.54** | **99.54** | **98.02** |
| | 200 | 86.27 | 96.29 | 97.86 | 97.07 |

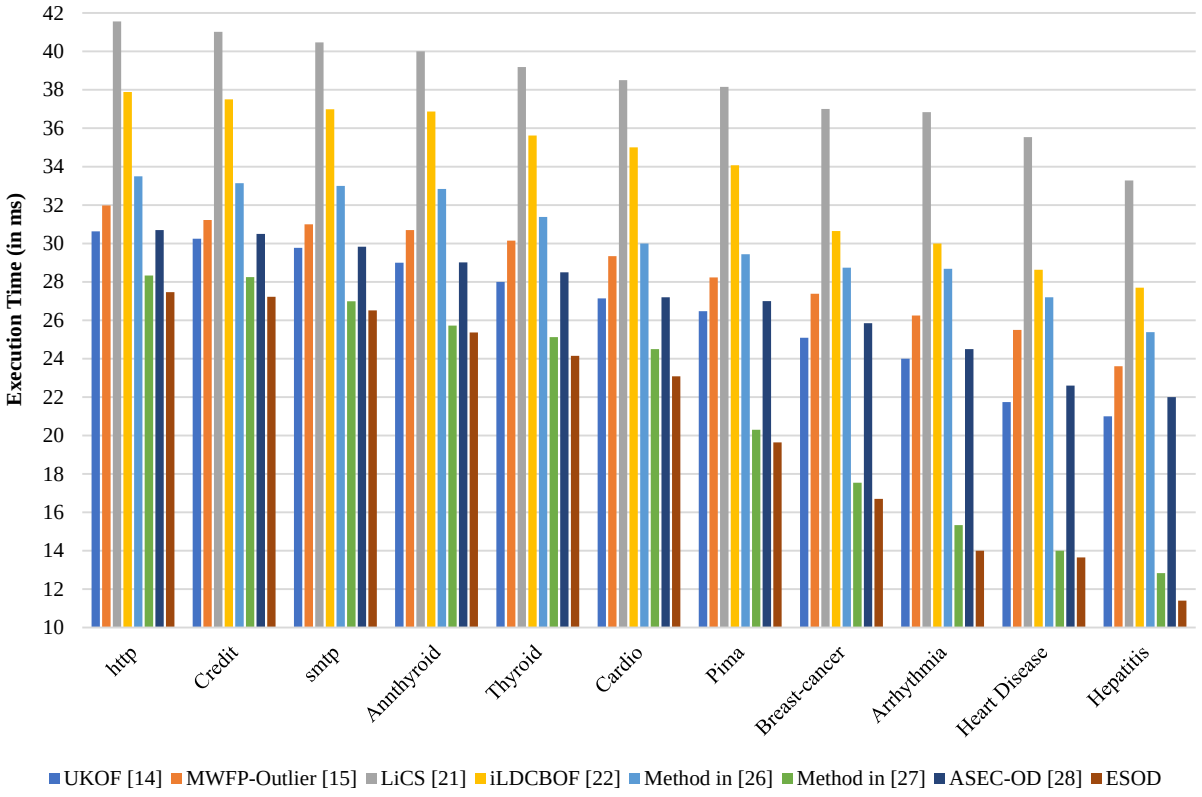



Fig. 7. Execution Time Evaluation for ESOD Compared with different Methods on different Datasets.

## VI. Conclusion and Future Work

In this study, we propose a novel sliding window ensemble-based framework for detecting outliers in data streams called an Ensemble-base Streaming Outlier Detection (ESOD). To improve outlier detection decisions, the proposed framework was built with three machine learning algorithms as base learners: iForestASD, decision tree, and adaptive random forest (ARF) based on a weighted voting detection technique. Furthermore, ESOD considered the concept evolution nature of streaming data. Extensive empirical evaluations on various real datasets demonstrate the performance of our framework in comparison to that of existing algorithms in the literature. The results showed that the proposed framework beat existing algorithms in terms of outlier detection rate, as well as overall performance. In the future, we will look into how to adapt the framework to the feature-evolution nature of data streams. Another option is to use more optimization techniques and different feature selection methods. Aside from improving execution performance, we are also interested in implementing various time-based techniques that may lead to improved detection and accuracy rates.

## References

[1] M. Hahsler, M. Bolaños, and J. Forrest, “Introduction to stream: An extensible framework for data stream clustering research with R,” J. Stat. Softw., vol. 76, no. 1, 2017, doi: 10.18637/jss.v076.i14.

[2] S. Ramírez-Gallego, B. Krawczyk, S. García, Michał Wozniak, and F. Herrera, “A survey on data preprocessing for data stream mining: Current status and future directions,” Neurocomputing, vol. 239, pp. 39–57, 2017, doi: 10.1016/j.neucom.2017.01.078.

[3] M. M. Gaber, “Advances in data stream mining,” Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 2, no. 1, pp. 79–85, 2012, doi: 10.1002/widm.52.

[4] V. Chandola and V. KUMAR, “Anomaly Detection: A Survey,” ACM Comput. Surv., no. September, pp. 1–72, 2009.

[5] C. C. Aggarwal, Outlier Analysis, vol. 24, no. 2. 2016.

[6] T. Kim and C. H. Park, “Anomaly pattern detection for streaming data,” Expert Syst. Appl., vol. 149, p. 113252, Jul. 2020, doi: 10.1016/j.eswa.2020.113252.

[7] M. Sakr, W. Atwa, and A. Keshk, “Sub-Grid Partitioning Algorithm for Distributed Outlier Detection on Big Data,” 2018 13th Int. Conf. Comput. Eng. Syst., vol. IEEE, pp. 252–257, 2018.

[8] N. Paulauskas and A. Baskys, “Application of Histogram-Based Outlier Scores to Detect Computer Network Anomalies,” electronics, vol. 8, pp. 1–8, 2019.

[9] M. E. Silva and I. Pereira, “Bayesian Outlier Detection in Non - Gaussian Autoregressive Time Series Bayesian outlier detection in non-Gaussian AutoRegressive time series ∗,” J. Time Ser. Anal., no. December, 2018, doi: 10.1111/jtsa.12439.

[10] C. H. Park, “Outlier and anomaly pattern detection on data streams,” J. Supercomput., vol. 75, no. 9, pp. 6118–6128, 2019, doi: 10.1007/s11227-018-2674-1.

[11] M. Sakr, W. Atwa, and A. Keshk, “Parallel outlier detection in real time data streams,” Inf. Sci. Lett., vol. 217, no. 3, pp. 211–217, 2020.

[12] M. Togbe et al., “Anomaly Detection for Data Streams Based on Isolation Forest using Scikit-multiflow,” 2020.

[13] S. Mishra and M. Chawla, “A Comparative Study of Local Outlier Factor Algorithms for Outliers Detection,” in Emerging Technologies in Data Mining and Information Security, Advances in Intelligent Systems and Computing, 2019, pp. 347–356. doi: 10.1007/978-981-13-1498-8.

[14] F. Liu, Y. Yu, P. Song, Y. Fan, and X. Tong, "Scalable KDE-based top-n local outlier detection over large-scale data streams," Knowledge-Based Syst., vol. 204, p. 106186, 2020, doi: 10.1016/j.knosys.2020.106186.

[15] S. Cai et al., "MWFP-outlier: Maximal weighted frequent-pattern-based approach for detecting outliers from uncertain weighted data streams," Inf. Sci. (Ny)., vol. 591, pp. 195–225, 2022, doi: 10.1016/j.ins.2022.01.028.

[16] A. Zubaroğlu and V. Atalay, "Data stream clustering: a review," Artif. Intell. Rev., 2020, doi: 10.1007/s10462-020-09874-x.

[17] L. Tran, L. Fan, and C. Shahabi, "Fast distance-based outlier detection in data streams based on micro-clusters," ACM Int. Conf. Proceeding Ser., pp. 162–169, 2019, doi: 10.1145/3368926.3369667.

[18] A. Hassan, H. Mokhtar, and O. Hegazy, "A heuristic approach for sensor network outlier detection," Int J Res Rev Wirel Sens Netw, vol. 1, no. 4, pp. 66–72, 2011.

[19] C. HewaNadungodage, Y. Xia, and J. Lee, "Gpu-accelerated outlier detection for continuous data streams," in the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016, pp. 1133–1142.

[20] K. Yu, W. Shi, N. Santoro, and X. Ma, "Real-time outlier detection over streaming data," in 2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Internet of People and Smart City Innovation, SmartWorld/UIC/ATC/SCALCOM/IOP/SCI, 2019, pp. 125–132. doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00063.

[21] F. Z. Benjelloun, A. Oussous, A. Bennani, S. Belfkih, and A. Ait Lahcen, "Improving outliers detection in data streams using LiCS and voting," J. King Saud Univ. - Comput. Inf. Sci., vol. 33, no. 10, pp. 1177–1185, 2021, doi: 10.1016/j.jksuci.2019.08.003.

[22] A. Degirmenci and O. Karal, "Efficient density and cluster based incremental outlier detection in data streams," Inf. Sci. (Ny)., vol. 607, pp. 901–920, 2022, doi: 10.1016/j.ins.2022.06.013.

[23] M. M. Masud, J. Gao, J. Han, L. Khan, and B. M. Thuraising-ham, "Classification and adaptive novel class detection of feature-evolving data streams," IEEE Trans. Knowl. Data Eng., vol. 25, 2013.

[24] W. Wang, D. Wang, S. Jiang, S. Qin, and L. Xue, "Anomaly detection in big data with separable compressive sensing," in the 2015 international conference on communications, signal processing, and systems. Springer, 2016, pp. 589–94.

[25] H. Ghomeshi, M. M. Gaber, and Y. Kovalchuk, "A non-canonical hybrid metaheuristic approach to adaptive data stream classification," Futur. Gener. Comput. Syst., vol. 102, pp. 127–139, 2020, doi: 10.1016/j.future.2019.07.067.

[26] N. Iftikhar, T. Baattrup-Andersen, F. E. Nordbjerg, and K. Jeppesen, "Outlier Detection in Sensor Data using Ensemble Learning," Procedia Comput. Sci., vol. 176, pp. 1160–1169, 2020, doi: 10.1016/j.procs.2020.09.112.

[27] M. Togbe et al., "Anomalies Detection Using Isolation in Concept-Drifting Data Streams," computers, pp. 1–21, 2021, [Online]. Available: https://doi.org/10.3390/ computers10010013.

[28] N. Jayanthi, B. Vijaya Babu, and N. Sambasiva Rao, "An Ensemble Framework Based Outlier Detection System in High Dimensional Data," Mater. Today Proc., vol. 7, no. 4, pp. 1162–1175, Feb. 2021, doi: 10.1016/j.matpr.2020.11.491.

[29] S. Rayana, W. Zhong, and L. Akoglu, "Sequential ensemble learning for outlier detection: A bias-variance perspective," Proc. - IEEE Int. Conf. Data Mining, ICDM, pp. 1167–1172, 2017, doi: 10.1109/ICDM.2016.117.

[30] F. Liu, K. Ting, and Z. Zhou, "Isolation forest," in In 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413–422.

[31] P. Karczmarek, A. Kiersztyn, W. Pedrycz, and E. Al, "K-Means-based isolation forest," Knowledge-Based Syst., vol. 195, p. 105659, 2020, doi: 10.1016/j.knosys.2020.105659.

[32] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window," IFAC Proc. Vol., vol. 46, no. 20, pp. 12–17, 2013, [Online]. Available: https://doi.org/https://doi.org/10.3182/20130902-3-CN-3020.00044.

[33] T. Thomas, A. P. Vijayaraghavan, and S. Emmanuel, "Applications of Decision Trees," in Machine Learning Approaches in Cyber Security Analytics, Singapore: Springer Singapore, 2020, pp. 157–184. doi: 10.1007/978-981-15-1706-8_9.

[34] L. Breiman, "Random forests," Mach Learn, vol. 45, no. 1, pp. 5–32, 2001.

[35] H. M. Gomes et al., "Adaptive random forests for evolving data stream classification," Mach. Learn., pp. 1–27, 2017, [Online]. Available: https://doi.org/10.1007/s10994-017-5642-8.

[36] V. C. Osamor and A. F. Okezie, "Enhancing the weighted voting ensemble algorithm for tuberculosis predictive diagnosis," Sci. Rep., vol. 11, no. 1, pp. 1–11, 2021, doi: 10.1038/s41598-021-94347-6.

[37] L. Yang and A. Shami, "A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams," IEEE Internet Things Mag., vol. 4, no. 2, pp. 96–101, Jun. 2021, doi: 10.1109/IOTM.0001.2100012.

[38] D. Dua and C. Gra, "UCI machine learning repository," 2017. http://archive.ics.uci.edu/ml.

[39] F. Pedregosa et al., "Scikit- learn: Machine learning in Python," 12:2825–2830, Mach Learn Res, 2011. https://scikit-learn.org.

[40] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-Multiflow: A Multi-output Streaming Framework," J. Mach. Learn. Res., vol. 19, no. 72, pp. 1–5, 2018, [Online]. Available: http://jmlr.org/papers/v19/18-251.html.