symmetry

MDPI

Article

Hyper-Heuristic Framework for Sequential Semi-Supervised Classification Based on Core Clustering

Ahmed Adnan [1,*], Abdullah Muhammed [1], Abdul Azim Abd Ghani [2], Azizol Abdullah [1] and Fahrul Hakim [1]

[1] Department of Communication Technology and Networks, Faculty of Computer Science and Information Technology, University Putra Malaysia, Serdang 43300, Malaysia; abdullah@upm.edu.my (A.M.); azizol@upm.edu.my (A.A.); fahrul@upm.edu.my (F.H.)

[2] Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, University Putra Malaysia, Serdang 43300, Malaysia; azim@upm.edu.my

* Correspondence: gs49566@student.upm.edu.my

Received: 11 May 2020; Accepted: 1 June 2020; Published: 4 August 2020

Abstract: Existing stream data learning models with limited labeling have many limitations, most importantly, algorithms that suffer from a limited capability to adapt to the evolving nature of data, which is called concept drift. Hence, the algorithm must overcome the problem of dynamic update in the internal parameters or countering the concept drift. However, using neural network-based semi-supervised stream data learning is not adequate due to the need for capturing quickly the changes in the distribution and characteristics of various classes of the data whilst avoiding the effect of the outdated stored knowledge in neural networks (NN). This article presents a prominent framework that integrates each of the NN, a meta-heuristic based on evolutionary genetic algorithm (GA) and a core online-offline clustering (Core). The framework trains the NN on previously labeled data and its knowledge is used to calculate the error of the core online-offline clustering block. The genetic optimization is responsible for selecting the best parameters of the core model to minimize the error. This integration aims to handle the concept drift. We designated this model as hyper-heuristic framework for semi-supervised classification or HH-F. Experimental results of the application of HH-F on real datasets prove the superiority of the proposed framework over the existing state-of-the art approaches used in the literature for sequential classification data with evolving nature.

Keywords: hyper-heuristic; extreme learning machine; genetic algorithm; online clustering; off-line clustering; evolving stream data; semi-supervised classification

1. Introduction

Most platforms continuously generate data in the modern era. Every moment, the deployment of the Internet of Things (IoT) [1], cloud-based systems [2], the usage of social platforms [3], and industrial processes [4] create a source of massive data. Some data can be stored to conduct analyses later, whereas others require prompt analysis. In stream data, the online analysis and classification of data are crucial in many conditions, such as in the case of non-availability of storage or when prompt reaction according to their content is needed. An example is the analysis of intrusion detection system (IDS) traffic to sense any possible attack in the data [5]. Another example is the analysis of body network data to identify any abnormal activity in the human data [6].

Several challenges occur in the learning of data stream. First, the data are numerous, with multivariate nature [7] and storing them for clustering or classification is not feasible. Thus, data reduction phases must be implemented before learning. Second, the probabilistic distribution

of learning algorithms for data stream require pre-defined parameters, a factor that results in changing performance when the data changes. Furthermore, data in real life are naturally evolving, whose handling requires a dynamic update in the internal parameters of the algorithm. Dealing with parameter updates is challenging. It requires a trial-and-error process that is not feasible in autonomous systems [8]. Thus, this study aims to incorporate artificial systems for this task.

Artificial intelligence (AI) is becoming increasingly useful in solving different types of data handling and processing problems. Examples of valuable AI algorithms include the neural networks (NN) with a powerful learning capability [9,10] and the heuristic-based approach useful for encoding experience knowledge [11]. Using AI for data classification has been proven effective, but a single AI approach suffers from limitations. Hence, integrating multiple AI blocks in one framework is useful to overcome the limitation of an individual AI block, such as integrating genetics algorithms (GA) [12] with other AI models. Using meta-heuristic searching types algorithm became crucial for stream data applications such as using Cuckoo search algorithm for electric load forecasting [13], or using integrated Cuckoo search and particle swarm optimization for natural terrain feature extraction [14], or hybrid chaotic immune algorithm for seasonal load demand forecasting [15].

The literature contains a wide range of sequential learning algorithms for stream data. Many researchers have adopted AI algorithms for this purpose. The usage of extreme learning machine for classifying them was done by the work of [16]. In their work, self-adaptive stream data classification framework (SASDC-Framework) was proposed. In SASDC-Framework, subsets of data were created to include the data that arrives at the same time. In addition, bisection method strategy and inverse distance weighted strategy were used to find out the best size of the subset. Obviously, the framework considers the incremental learning of the classifier as the countering part of concept drift. However, the incremental learning is not adequate when fast changes in data distribution occur. Also, other researchers have developed frameworks based on incremental learning. [17] built a learning framework for classifying streaming data. The framework is composed of a generative network, discriminant structure, and bridge. The first component used dynamic feature learning. The second component involved regularization of the network construction and facilitating parameter learning of the generative network. The third component required the integration of the network and structure by enabling the incorporation of supervision information. However, as it is declared by the authors of [17], their approach was not good at dealing with concept drift in streaming data.

To deal with the parameter changes, [18] used an inbuilt forgetting mechanism of NNs. The approach is based on active learning, which requires labels only for interesting examples that are crucial for appropriate model upgrading. The work demands the capability of controlling the speed of the forgetting mechanism. Unfortunately, the forgetting mechanism with lacking speed control is not useful for dealing effectively with parameter changes.

Other researchers have exploited the power of ensemble learning for sequential classification. [19] developed an ensemble-based sequential classification approach. Boosting the new batches of data was achieved by adding base learners according to its current accuracy. The approach is called iterative boosting streaming ensemble. This approach is satisfactory for learning the new behavior of data. However, ensemble learning is not powerful for handling the concept drift without a mechanism of aggregation that is aware of parameters changes. Hence, adding base learners according to its current accuracy is not optimal to handle the parameter changes.

Many researchers have dealt with concept drift by incorporating clustering algorithms in their sequential classification models in order to identify any emerging in the classes. In [20] the usage of the capability of fuzzy clustering for handling the concept drift was to serve Dynamic Incremental Semi-Supervised classification. Similarly, the work of [21] an incremental mixed-data clustering was used to serve in incremental semi-supervised flow network-based IDS. Moreover, [22] where Cauchy density is used to assign new samples to clusters or to create new clusters. However, it suffers from dependency on the threshold of assignment which is highly sensitivity to statistical distribution of clusters. In addition, it is important to exploit the clustering part from the perspective of overlooking

the outdated stored knowledge that effects tracking the dynamic changes without giving it absolute control on the decision of classification due to lacking essential gained knowledge. This concept was missed in the previous approaches that have used clustering for dealing with concept drift. In the work of [23], a new incremental grid density-based learning framework was proposed. The authors have focused on concept drift with limited labeling. Three techniques were used in the framework: grid density clustering, evolving ensemble of classifiers, uniform grid density sampling mechanism. The framework relies on the concept "classification upon clustering"; however, it does not handle the concept drift in the clustering part, and it considers that simple assisting of classification with clustering can overcome the concept drift.

The dealing with concept drift was not only exclusive to sequential classification approaches. Also, it was under researcher interests for handling the dynamic changes in stream data clustering models. Furthermore, bio-inspired approaches were used to handle such changes. [24] presented an online, bio-inspired approach to clustering dynamic data streams. The proposed ant colony stream clustering (ACSC) algorithm is a density-based clustering algorithm, whereby clusters are identified as high-density areas of the feature space separated by low-density areas. ACSC identifies clusters as groups of micro-clusters. The tumbling window model is used to read a stream, and rough clusters are incrementally formed during a single pass of a window. A stochastic method is employed to identify these rough clusters, and this process is shown to significantly speed up the algorithm with only a minor cost to performance than with a deterministic approach. The rough clusters are then refined using a method inspired by the observed sorting behavior of ants. Ants pick up and drop items according to their similarity with the surrounding items. Artificial ants sort clusters by probabilistically picking and dropping micro-clusters on the basis of local density and local similarity. Clusters are summarized using their constituent micro-clusters, and these summary statistics are stored offline. This approach suffers from lacking an assisting knowledge representation and storing structure. This was found by Ant inspired swarm optimization-based approach [25] for multi-density clustering. In this work, the concept drift was addressed by maintaining discovered clusters online and tracking their change using ant swarm optimization. However, this approach might not be capable of tracking fast changes of clusters. Other researchers have also employed optimization models to cluster categorical data streams [26]. In their model, a cluster validity function is proposed as the objective function to evaluate the effectiveness of the clustering model while each new input data subset is flowing. It simultaneously considers the certainty of the clustering model and the continuity with the last clustering model in the clustering process. An iterative optimization algorithm is proposed to identify the optimal solution of the objective function with some constraints regarding the coefficients of the objective function. Furthermore, the authors have strictly derived a detection index to drift concepts from the optimization model. They proposed a detection method that integrates the detection index and the optimization model to ascertain the evolution trend of cluster structures on a categorical data stream. The new method was claimed to effectively avoid ignoring the effect of the clustering validity on the detection result. However, the cluster validity objective function might not be generalizable to all types of data and clusters. Another clustering-based work that aimed at handling dynamic changes of clusters is the work of [27] where the algorithm is composed an online phase, which keeps summary information about evolving multi-density data stream in the form of core mini-clusters, and an offline phase that generates the final clusters using an adapted density-based clustering. The grid-based method is used as an outlier buffer to handle both noises and multi-density data and yet is used to reduce the merging time of clustering. This work can handle multi-density-based clusters; however, it is not handling concept drift effectively because of lacking a distinct block or model for detecting concept drift.

Overall, the most of previous literature on semi-supervised streaming data learning has used various techniques from AI to overcome the problematic issue of updating parameters of models based on various approaches some of them are evolutionary and other statistical incremental semi-supervised and heuristics. Accordingly, a novel AI framework for semi-supervised classification of online data streaming with providing better arrangement and interaction between various AI blocks is still a

research interest. Unlike previous works, this research proposes a hyper-heuristic framework based on three AI components to perform classification of streaming data by enabling parameter adaptation to overcome the parameter update and to avoid the concept drift. The three components consist of online sequential extreme learning machine (OSELM) [28], GA, and core online-offline clustering. The framework is designated as hyper-heuristic framework (HH-F). Each component has its role in assuring correct prediction or classification decision with an avoidance of concept drift. OSELM with its online learning capability, genetic with its evolutionary searching for the best parameters, and core online-offline clustering with its fast response to clusters changes all are integrated into one novel framework.

The remainder of this article is organized as follows. Section 2 establishes the methodology. Section 3 shows the experimental works and results. Section 4 provides the conclusion and future work.

2. Methodology

This section presents the developed methodology of HH-F. Section 2.1 presents the general framework. Then, Section 2.2 provides the dataset representation. Next, Section 2.3 describes the core online-offline clustering. Afterwards, Section 2.4 discusses the genetic optimization. Then, Section 2.5 provides the NN. The configuration is presented in Section 2.6. Section 2.7 presents the class prediction method. Next, Section 2.8 presents the big O analysis of computational cost. Section 2.9 list the evaluation metrics. Next, Section 2.10 provides the dataset description.

2.1. General Framework

Figure 1 presents the general framework of hyper-heuristic for stream data classification. Main blocks are considered in the system design including dataset generation, the GA, NN, configuration, core (online and offline phases), and class prediction. The input and output of each blocks is provided in Table 1. The proposed framework is considered hyper-heuristic because it uses genetics with the combination of machine learning technique to provide better clustering results for the core systems. The remaining blocks of the system are discussed below.

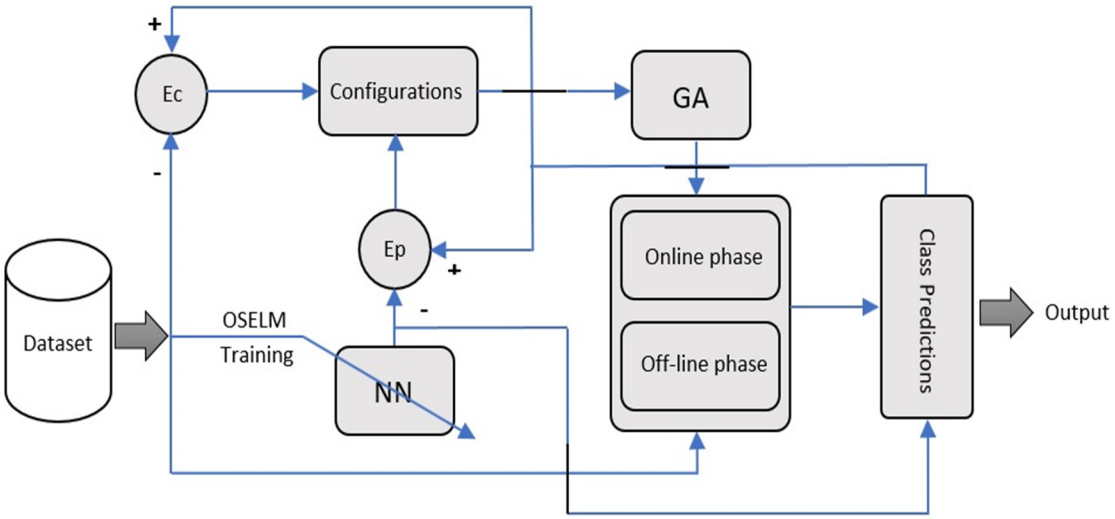

Figure 1. Framework of Stream Data Classification hyper-heuristic framework (HH-F).

The core of the framework is the direct clustering algorithm. In this article, we adopt the core of [27] because it performs clustering based on two phases, namely, online and offline. Using two phases aims for the clustering in the offline phase to use the result of data reduction that was done in the online phase. Another reason to adopt [27] is its hybrid nature which combines both grid and distance concepts in one streaming algorithm which makes it more memory and computationally efficient. As mentioned, the problem that arises when dealing with most clustering approaches is their

use of a static parameter setting, which creates a concern about the performance when dealing with evolving classes. We solve this problem in our framework by connecting the core clustering block with an optimization block.

Table 1. The framework blocks with their respective inputs and outputs.

Blocks	Input	Output
Neural Network (NN)	Data features.	Class assignment.
Genetic Algorithm (GA)	Error provided from block. Configuration.	Candidate solution for optimizing the error.
Configuration	Error of class prediction with respect to NN or error of class prediction with respect to ground truth.	One of the two types of errors.
Core clustering	Data features, genetic solution.	Cluster assignment.
Class prediction	Core Clustering results, NN predictions.	Classification result.

The optimization block requires an objective function that reflects the performance of the system. Thus, we use the error of class prediction as an updated objective function. The error is calculated as the difference between the predicted data and the ground truth in the previous chunks. However, constantly knowing the ground truth of the previous chunks is not feasible. Therefore, NN is trained whenever ground truth data are available and used to assist in the calculation of the objective function through its gained knowledge.

E_p, E_c denote the difference between the class prediction result in one side and the predicted NN or actual ground truth respectively. The actual ground truth is available partially according to the nature of applications, e.g., in IDS-based classification, the system will be able to identify the critical attacks after a period of time. This arrangement of the AI block in the framework is one of the novelties of this work. Other novelties include the design of the optimization, solution space, and the logic of the configuration block. The following subsections present the related details.

In order to elaborate the sequence of operation of the framework we present the general steps as follows:

1. Receive new chunk of the data for testing.
2. Use online-offline clustering block for assigning the records of the chunk to their respective clusters.
3. Use the NN block to predict the classes of the records of the chunk.
4. Use the class prediction block to use the prediction of the NN with the result for the online-offline clustering to perform class labeling of the clusters.
5. Find the error between the result of the class labeling of the clusters and the result of the NN as an objective function for the GA block that will keep modifying the parameters of the online-offline clustering block until reaching the minimum error.
6. Update the knowledge of NN when receiving labels of old records (ground truth).
7. Use the configuration error to switch between two types of errors: error with respect to NN E_p and error with respect to ground truth E_c.
8. If stream data is not finished go to step 1.

2.2. Dataset Representation

The dataset is represented in Equation (1).

$$\Xi = \{X_i, Y_i\}, \text{ where, } i = 1, 2, \ldots, N \tag{1}$$

where N denotes the number of chunks, $X_i = \left\{x_1 \ldots x_j \ldots x_L\right\}$ denotes the input feature of the records in one chunk, output features can be denoted by $Y_i = \left\{y_1 \ldots y_j \ldots y_L\right\}$, $x_j = \left(x^1\, x^2 \ldots x^m\right)$ denotes the features in one record, and the output in one record is shown by $y_j = \left(y^j\right)$. Notably, the dataset is sequentially presented to the system. Moreover, whenever a chunk of data X_i is provided, the corresponding labels Y_i are not provided, such that the system must predict the classes of X_i. However, all the labels of the previous chunks are known at that point Y_j $j = 1, 2, \ldots, i-1$.

This is a general representation of any sequential classification problem. For example, it matches the representation of IDS data where we expected to present chunks to classifier for prediction at any moment. Next, whenever true labeling or knowledge of the nature of the records, i.e., attacks or normal, is known, it will be presented again to the system for knowledge update.

2.3. The Core (Online and Off-Line Phase)

The core consists of two phases, namely, online and the offline phases. We explain each of the two phases in the following sections.

2.3.1. Online Phase for Micro-Clusters Update

The role of the online phase is to generate micro-clusters that represents a summary of data distribution in the space. It is composed of two stages: grid projection and micro-clusters update. We explain each of them as follows.

(1) Grid: is the first stage of stream data reduction in the online phase. Grid represents the mapping of the data to a pre-defined space partitioning in the dimensions of the data. Square grids are typically the simplest. The grid granularity denotes the resolution of the grid partitioning in the different dimensions. To accommodate the evolving nature of data and consume less memory when the dimension of data is high, grid granularity can be used as part of the solution space in the meta-heuristic part in the previous framework. Grid granularity is denoted as g.

(2) Micro-clusters update: The input of the online micro-clusters phase involves streamed data after mapping to the grid. The output is the micro-cluster update. They are intermediate results of clustering before making the final cluster. A micro-cluster is generated from a grid using a procedure presented in Table 2. Assuming that the data are streamed with dimension m or any sample of the data $p_t^{(j)} = \left(x_t^1 x_t^2, \ldots . x_t^m\right)^j$, then, $p_t^{(j)}$ has a weight that is given by Equation (2).

$$w\left(t_c, p_t^{(j)}\right) = 2^{-\lambda(t_c - t)},\ \lambda > 0 \tag{2}$$

t_c denotes the current moment, and λ denotes the forgetting factor which controls the importance of the history in the clustering. The role of the weight model that is related to a declining exponential term with respect to time is to give higher weight for the most recent samples. We assume Grid g contains n points $p_{t1}^{(1)} p_{t2}^{(2)}, \ldots . p_{tn}^{(n)}$. First, we calculate the weight of the grid using Equation (3).

$$w_g(t_c) = \sum_{i=1}^{n} w\left(t_c, p_{t_i}^{(i)}\right) \tag{3}$$

The grid (or multiple nearby grids) that contains set of points are named as an outlier. However, after the weight of the grid reaches a certain threshold T_L it becomes a micro-cluster. Each micro cluster is composed of center and radius. The center of the micro-cluster is generated from the grid using Equation (4).

$$C_g(t_c) = \frac{\sum_{i=1}^{n} w\left(t_c, p_{ti}^{(i)}\right) p_{ti}^{(i)}}{w_g(t_c)} \tag{4}$$

Then, the radius of the micro-cluster is estimated using Equation (5).

$$r_g(t_c) = \frac{\sum_{i=1}^{n} w\left(t_c, p_{ti}^{(i)}\right) \text{distance}\left(p_{ti}^{(i)}, C_g(t_c)\right)}{w_g(t_c)} \tag{5}$$

Updating micro-clusters refers to either converting grids from outliers to micro-clusters or converting grids from micro-cluster to outliers. The decision of this update is made according to two thresholds, namely, the lower threshold T_L and upper threshold T_U. The grid density $w_g(t_c)$ will be compared with the two thresholds, and the conversion will be performed using Table 2.

Table 2. Pseudocode of Updating the Micro Clusters and Outliers in The Online Phase.

If ($w_g(t_c) < T_L$) then	// From equation 2
Remove (grid_list,g)	// List of all grids that contains micro clusters
Add (outliers_list,g)	// List of all grid that contains outliers
Endif	
If ($w_g(t_c) > T_u$) then	
Add (grid_list,g)	
Remove (outliers_list,g)	
Endif	

2.3.2. Offline Phase for Clustering

The input of the offline phase for cluster updates is the micro-clusters, and the output is the clustering results. The micro cluster must be applied before entering offline clustering. The condition is that the number of points of the micro-cluster should be lower than a pre-defined threshold named *MinPts*.

To update the clusters, a connected graph G will be created for each cluster. The nodes of the graph represent the micro-clusters, and the edges represent the connectivity decision between one micro-cluster and another. The decision to connect two micro-clusters is based on the Euclidian distance. Micro-cluster A is connected to micro-cluster B if and only if $d(G) - \sigma(G) < d(A, B) < d(G) + \sigma(G)$. $d(A, B)$ denotes the distance between micro-clusters A and B, $d(G)$ denotes the average distance between micro-clusters in the connected graph G, and $\sigma(G)$ denotes the average standard deviation of distances between the micro-clusters in the connected graph G.

2.4. The Genetic Algorithm (GA)

The meta-heuristic searching model aims to dynamically update the crucial parameters in the core clustering block. We select five parameters for this purpose. The solution structure contains the essential parameters of the core clustering algorithm. Thus, the solution space is represented as vector S in Equation (6).

$$S = (\ g, MinPts,\ \lambda, T_L,\ T_U) \tag{6}$$

where,

g denotes the grid granularity.
MinPts denotes the threshold of core mini cluster to use it for offline phase.
λ denotes the forgetting factor.
T_L denotes the threshold for converting a core mini cluster to an outlier.
T_U denotes the threshold for converting an outlier to core mini cluster.

GA will be used for the searching. For more effective searching, we propose our own operators for modified GA. The crossover operation is performed using Equations (7) and (8). We assume that we have two solutions S_A, S_B from the pool of solutions in certain generation.

$$S_A = (g_a, MinPts_a,\ \lambda_a, T_{La},\ T_{Ua}) \tag{7}$$

$$S_B = (g_b, MinPts_b,\ \lambda_b, T_{Lb},\ T_{Ub}) \tag{8}$$

For generating an offspring, a crossover operation between S_A and S_B is performed. In order to do the crossover operation, a random number r is generated between 0 and 1, and the crossover is done as represented in Equation (9).

$$S_A \oplus S_B = \begin{cases} (g_c, MinPts_a, \lambda_b T_{La}, T_{Ub}) & \text{if } r \leq 0.50 \\ (g_c, MinPts_b, \lambda_a, T_{Lb}, T_{Ua}) & \text{if } r \leq 1 \end{cases} \quad (9)$$

where, $g_c = I\left(\min(g_a, g_b), \max(g_a, g_b)\right)$

$I(x, y)$ represents a random integer between x and y. Thus, the mutation operator $M()$ is designed and can be shown in Equations (10) and (11).

$$S_A = (g_a, MinPts_a, \lambda_a, T_{La}, T_{Ua}) \quad (10)$$

$$M(S_A) = S'_A \quad (11)$$

where,
$S'_A = (g_a', MinPts_a', \lambda_a', T_{Lc}', T_{Uc}')$
$g_a' = L_g(g_a)$,
$\lambda_a' = L_\lambda(, \lambda_a)$
$MinPts_a' = L_{MinPts}(MinPts_a)$,
$T_{Lc}' = L_{T_2}(T_{La})$ and $T_{Uc}' = L_{T_3}(T_{Ua})$

$L_T(x)$ generates a random number around x with a maximum difference as T from x using uniform distribution. The parameter T is adjusted for each of the elements of the chromosome to define the strength of the mutation. The higher value of T is equivalent to stronger mutation while the lower value of T is equivalent to weaker mutation.

The last part in the genetic modification is applying a boosting for GA, i.e., narrowing down the initial searching zone, by operating the framework for pre-defined number of iterations based on the same boosting data that is used to boost the NN. This will narrow down the searching area to be close to the optimal values of parameters of core.

2.5. The Neural Network

The NN is embedded in the framework in order to exploit the information that can be extracted from the newly labeled data by storing its knowledge in the component of NN. Hence, this component will assist in providing a feedback in the case of any deviation in the prediction from the true gathered knowledge in order to correct the prediction.

In this framework, we adopt an extreme learning machine called OSELM [29,30]. This NN has an online variant that permits the updating of the knowledge of the NN according to the provided chunks. Given an activation function g and $\widetilde{N}$ hidden neurons, the learning procedure consists of two phases described below.

2.5.1. Boosting Phase Using the Initial Chunks

Given a small initial training set $\{X_0, Y_0\}$ to boost the learning algorithm first through the following boosting procedure.

1. Assign arbitrary input weight w_i and bias b_i based on a random variable with center u_i and standard deviation σ_i, $i = 1, \ldots, L$.
2. Calculate the initially hidden layer output matrix in Equation (12).

$$H_0 = [h_1, \ldots, H_L]^T \quad (12)$$

where, $h_i = [g(w_i.x_i + b_i), \ldots, g(w_L.x_i + b_L)]^T$ and, $i = 1, \ldots, L$.

3. Estimate the initial output weight by the Equation (13).

$$\beta^{(0)} = M_0 H_0^T Y_0 \tag{13}$$

where, $M_0 = (H_0^T Y_0)^{-1}$ and $Y_{0=}[y_{1,\ldots,}y_L]^T$ Set $K = 0$.

2.5.2. Sequential Learning Phase

For each further coming observation (X_i, Y_i), where, $X_i \in \ i = L+1, L+2, L+3, \ldots,$

1. Calculate the hidden layer output vector by using Equation (14).

$$h_{(k+1)} = [g(w_1.x_i + b_1), \ldots, g(w_L.x_i + b_L)]^T \tag{14}$$

2. Calculate the latest output weight $\beta^{(k+1)}$ based on Recursive Least Square (RLS) algorithm shown in Equations (15) and (16).

$$M_{k+1} = M_k - \frac{M_k h_k + 1h_k^T + 1M_k}{1 + h_{k+1}^T M_k h_{k+1}} \tag{15}$$

$$\beta^{(k+1)} = \beta^{(k)} + M_{k+1} h_{k+1} \left(y_i^T - h_{k+1}^T \beta^{(k)}\right) \tag{16}$$

Set $k = k + 1$

2.6. The Configuration

The configuration is used to activate the GA and enable the objective function that will be used. Two variants of the HH-F hyper-heuristic are available. In the first variant, the GA is run once in each iteration, and the difference between the class prediction and the trained NN will be used as an objective function that must be minimized. In the second variant, the GA will be run twice, and the objective function for the first and second instances will be the difference between the NN and the class prediction and between the ground truth and the class prediction, respectively.

2.6.1. Hyper-Heuristic Variant 1

Table 3 presents the pseudo-code of variant 1 (Hyper 1). The code contains two phases, namely, boosting and iterative phases. The latter is a combination of prediction and correction phases.

The GA and the OSELM are called in the boosting phase. In line 1, the initial parameters of the core are generated randomly. In line 2, the initial weights of the NN are generated on the basis of the boosting data. GA is then initialized. The corresponding objective function is defined by the difference between the class prediction and the ground truth, which is subsequently called for the numberOfIterations based on the defined MutationRate and CrossoverRate. Once the boosting phase is completed, N iterations of the iterative phase are performed, where N is equal to the number of chunks received by the stream data. Two stages are called in each phase. The first is the prediction that enables GA to run for numberOfIterations to change the parameters of the core based on the difference between the NN prediction and the class prediction (lines 7 and 8). Next, the updated parameters are assigned to the core, which performs a prediction that represents the output of the core (line 9). The second stage is the correction stage, which uses the recently known labels to train the supervised model again (line 11).

Table 3. Pseudocode of Hyper 1.

```
Data={xi,yi}, i = 1, 2, . . . , N                    // Online streaming data
SSBoundary                                           // Boundary for the random parameters
numberOfIterations
numberOfIterations
CrossoverRate
ObjectiveFunction = 'CoreProcess'
NN0                                                  // The initial NN
Output
Yp = {ypi}, i = 1, 2, . . . , N
Start
//Boosting phase
   1.   P0 = Random                  // Initialise random parameters for the core
   2.   NN0 = OSELM(x0,y0)
   3.   GA.Options = (SSBoundary,numberOfIterations, MutationRate,
        CrossoverRate, ObjectiveFunction, classPrediction(core(P0,X0))-y0)
   4.   P0 = GA.run;

//iterative phase
   5.   For i = 2 until N
   6.   //Prediction
   7.   GA.Options = (SSBoundary,numberOfIterations, MutationRate,
        CrossoverRate, ObjectiveFunction, classPrediction (core(P(i-1),xi))-NN(i-1,xi))
   8.   Pi = GA.run;
   9.   ypi = classPrediction (core(P(i),xi))
   10.  //Correction
   11.  NN(i) = OSELM(xi,yi)

End
```

2.6.2. Hyper-Heuristic Variant 2

Table 4 presents the pseudo-code of variant 2 (Hyper 2). Similar to Hyper 1, the procedure consists of the boosting and iterative phases, and the latter is a combination of prediction and correction phases.

The GA and the OSELM are called in the boosting phase. In lines 1 and 2, the initial parameters of the core are randomly generated, and the initial weights of the NN are generated on the basis of the boosting data, respectively. GA is then initialized, and the corresponding objective function is defined by the difference between the class prediction and the ground truth. The function is subsequently called for the numberOfIterations based on the defined MutationRate and CrossoverRate. The result comprises the initial parameters of core P0 (line 4). Once the boosting phase is completed, N iterations of the iterative phase are performed, where N is equal to the number of chunks received by the stream data. The difference between the two variants is that Hyper 2 contains a correction of the core parameters using GA based on the ground truth (lines 12 and 13). This step allows Hyper 2 to utilize the knowledge of previously known labels using GA, NN. While Hyper 1 utilizes the knowledge of previously known labels using only the NN component because the GA component works depend on NN when running in the prediction phase.

Table 4. Pseudocode of Hyper 2.

```
Data = {xi,yi}, i = 1, 2, . . . , N,                    // Online streaming data
SSBoundary                                              // Boundary for the random parameters
numberOfIterations
MutationRate
CrossoverRate
ObjectiveFunction = 'CoreProcess'
NN0                                                     // The initial NN
Output
Yp = {ypi}, i = 1, 2, . . . , N
Start
//Boosting phase
    1.  P0 = Random                 // Initialise random parameters for the core
    2.  NN0 = OSELM(x0,y0)
    3.  GA.Options = (SSBoundary,numberOfIterations, MutationRate,
        CrossoverRate,ObjectiveFunction, classPrediction (core(P0,X0))-y0)
    4.  P0 = GA.run;

//iterative phase
    5.  For i = 2 until N
    6.  //Prediction
    7.  GA.Options = (SSBoundary,numberOfIterations, MutationRate,
        CrossoverRate,ObjectiveFunction, classPrediction (core(P(i-1),xi))-NN(i-1,xi))
    8.  Pi = GA.run;
    9.  ypi = classPrediction (core(P(i),xi))
    10. //Correction
    11. NN(i) = OSELM(xi,yi)
    12. GA.Options = (SSBoundary,numberOfIterations, MutationRate,
        CrossoverRate,ObjectiveFunction, classPrediction (core(Pi,Xi))-yi)
    13. Pi = GA.run;

End
```

2.7. Class Prediction

The output of the core clustering is a set of distinct clusters. The clustering results are mapped into a classification using the labels of the points that are part of the cluster. Each point x in cluster S has a label y. First, a histogram for the number of points for each class inside the cluster is generated. Next, the peak of the histogram is selected as the class of the cluster. Any new predicted point belonging to this cluster will take the class of the cluster. This step is a separate block in the framework named class prediction.

In the case of partial labeling, only a few points might be labeled, which might result in the unsatisfactory performance of the class prediction. This issue is resolved using the NN output, which is a regular classifier that predicts each sample of the records. In the case of fully labeled data, the ground truth is used instead of NN predictions.

2.8. Big O Analysis for Computation

This study is interested in the computational complexity of the sub-blocks of the supervised learning NN and the optimization GA because both must be called in each iteration to update the parameters of the core for countering the dynamic changes in the stream. Assume that the algorithm works for $GA(N_I, N_s)$ and $ELM\left(\widetilde{N}, N_k, m\right)$.

where,

N_I denotes the number of iterations.

N_s denotes the number of solutions.

N_k the size of the chunk.
$\widetilde{N}$ denotes the number of hidden neurons.
m denotes the number of features.

According to [31] the computational complexity of ELM has two stages: the first is the projection to hidden layer $O(N_k\widetilde{N}\,m + N_k\widetilde{N})$, and the second is the pseudo-inverse $O(N_k\widetilde{N}^2 + \widetilde{N}^3) + O(N_k\widetilde{N})$. Given that the GA complexity is represented as $O(N_I N_s)$, the total complexity is defined as $O(N_k\widetilde{N}\,m + N_k\widetilde{N}) + O(N_k\widetilde{N}^2 + \widetilde{N}^3) + O(N_k\widetilde{N}) + O(N_I N_s)$.

2.9. Evaluation Measurement

This section presents the evaluation measures used for quantifying the performance of the proposed HH-F and the comparison with state-of-the-art approaches. TP denotes true positive, TN denotes true negative, FP denotes false positive, and FN denotes false negative.

Accuracy

Accuracy represents the number of true predictions divided by all cases of prediction [32]. The formula is calculated in Equation (17).

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{17}$$

1. Precision (PPV)
   PVV represents the number of true positive predicted by the classifier divided by the number of all predicted positive records [33]. The formula is calculated in Equation (18).

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{18}$$

2. Recall (TPR)
   TPR represents the number of TP predicted by the classifier divided by the number of all tested positive records [34]. The formula is calculated in Equation (19).

$$TPR = \frac{TP}{TP + FN} \tag{19}$$

3. Specificity (TNR)
   TNR represents the number of TN predicted by the classifiers divided by the number of all tested negative records [35]. The formula is calculated in Equation (20).

$$TNR = \frac{TN}{N} \tag{20}$$

4. NPV
   This measure represents the negative predicted value calculated as the number of predicted TN value over the total number of negative values [36]. The formula is calculated in Equation (21).

$$NPV = \frac{TN}{TN + FN} \tag{21}$$

5. G-mean
   This measure is calculated on the basis of the precision and recall [33]. The formula is calculated in Equation (22).

$$G\_mean = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}} \tag{22}$$

6. F-measure
   F-measure is the harmonic mean of the precision and recall [33]. The formula is calculated in Equation (23).

$$F\_measure = \frac{2 \times precision \times recall}{precision + recall} \tag{23}$$

2.10. Dataset Description

This section provides a description of the datasets that will be used for evaluation. Three real data are selected for the evaluation, namely, KDD'99, NSL-KDD, and LandSat.

KDD'99

KDD'99 is the most recognized dataset for evaluating anomaly detection methods [37]. The dataset is constructed using the data captured in DARPA'98 IDS, including gigabytes of compressed raw (binary) TCP dump data on seven weeks of network traffic, and can be processed into approximately 5 million connection records of nearly 100 bytes each. Moreover, this dataset includes 2 million connection records. Each record contains 41 features and can be regarded as normal, DoS, U2R, R2L, or a probing attack.

1. NSL-KDD

Various studies have encountered issues affecting the performance in the KDD'99 dataset. [37] proposed a new dataset called NSL-KDD; here, they removed duplicated records in the training and testing data and updated the number of the selected difficult level groups as inversely proportional to the percentage of records in the original KDD dataset. The training dataset consists of 21 different attacks out of the 37 attacks present in the test dataset. The known attack types also occur in the training dataset and in novel attacks that are unavailable for the training data.

2. Landsat satellite

This dataset contains 4435 objects from remote-sensing satellite images. Each object represents a region, and each sub-region is provided by four intensity measurements taken at different wavelengths. The number of attributes is 36. This dataset is utilized by [27].

3. Experimental Evaluation

We constructed three types of evaluations to evaluate the proposed two approaches of the HH-F hyper-heuristic. First, we generated the classification measures for each approach based on the partial percentage of labeling for LandSat and NSL-KDD (25%, 50%, 75%, and 100%) to evaluate the performance of the approach based on the partially labeled stream data in the sequential learning. Second, we generated the classification measures of the fully sequential learning with full percentage of labeling for each of the two variants of HH-F hyper-heuristics and for two benchmarks, namely, Cauchy [22] and MuDi [27]. Lastly, we performed a statistical evaluation to verify the overall superiority of the proposed HH-F hyper-heuristic over the two benchmarks.

Table 5 presents the evaluation results of the performance of Hyper 1 and Hyper 2 in terms of partial labeling. For LandSat, the finding reveals that the accuracy increases with the percentages of labeling, which is a normal occurrence that opens an opportunity to update the knowledge of

HH-F. Similarly, the other measures (precision, recall, specificity, NPV, G-mean, and F-measure) demonstrated an increasing trend in LandSat with respect to the percentage of the labeled data in the sequential learning.

Table 5. The performance of Hyper 1 and Hyper 2 in terms of partial labeling for landsat.

LandSat	**25%**	**50%**	**75%**	**100%**	**Hyber1**
Accuracy	0.9154	0.9257	0.9523	0.9523	
Precision	0.7939	0.8276	0.9028	0.9102	
Recall	0.7463	0.7770	0.8568	0.8570	
Specificity	0.9451	0.9488	0.9693	0.9544	
NPV	0.9497	0.9579	0.9631	0.9632	
G-mean	0.7698	0.8019	0.8795	0.8832	
F-measure	0.7694	0.8015	0.8792	0.8828	
LandSat	**25%**	**50%**	**75%**	**100%**	**Hyber2**
Accuracy	0.9257	0.9458	0.9608	0.9786	
Precision	0.8276	0.8679	0.8951	0.9413	
Recall	0.7770	0.8374	0.8825	0.9357	
Specificity	0.9488	0.9581	0.9725	0.9894	
NPV	0.9579	0.9604	0.9721	0.9877	
G-mean	0.8019	0.8525	0.8888	0.9385	
F-measure	0.8015	0.8524	0.8888	0.9385	

This behavior is more evident in LandSat than in NSL-KDD (Table 6). The results in the LandSat showed that the accuracy reached the highest level at 75% in Hyper 1, which is interpreted as a saturated level of extracted knowledge (i.e., additional training is unnecessary for the model). On the other hand, we observe that in both datasets, the accuracy was high for hyper 1 and hyper 2 even with the smallest percentage of labeling, i.e., percentage of 25%. This is interpreted by the difference between the two datasets in terms of the knowledge that is carried by new records, i.e., in many datasets new incoming chunks do not carry newer knowledge which makes the training update not useful. Another observe is the differences in the values between the precision recalls, and other metrics which is interpreted by the differences in the TP, FP, TN, and FN values. Despite that, all the metrics indicate when they are high to better performance in the prediction.

Table 6. The performance of Hyper 1 and Hyper 2 in terms of partial labeling for NSL-KDD.

NSL-KDD	**25%**	**50%**	**75%**	**100%**	**Hyber1**
Accuracy	0.9918	0.9965	0.9969	0.9972	
Precision	0.9668	0.9828	0.9866	0.9720	
Recall	0.9072	0.9567	0.9676	0.9684	
Specificity	0.9910	0.9923	0.9937	0.9853	
NPV	0.9350	0.9692	0.9763	0.9888	
G-mean	0.9366	0.9697	0.9770	0.9702	
F-measure	0.9361	0.9696	0.9770	0.9702	
NSL-KDD	**25%**	**50%**	**75%**	**100%**	**Hyber2**
Accuracy	0.9884	0.9906	0.9952	0.9969	
Precision	0.9536	0.9548	0.9601	0.9649	
Recall	0.8777	0.9016	0.9496	0.9671	
Specificity	0.9833	0.9836	0.9836	0.9806	
NPV	0.9216	0.9385	0.9741	0.9869	
G-mean	0.9149	0.9278	0.9549	0.9660	
F-measure	0.9141	0.9274	0.9549	0.9660	

In the second evaluation, Hyper 1 and Hyper 2 are compared with MuDi and Cauchy on the three datasets using fully labeled data. Figure 2 shows the experimental results for the comparison on NSL-KDD. HH-F is found to be superior to the MuDi and Cauchy. Four variants of HH-F are used for the comparison: Hyper 1 with and without boosting and Hyper 2 with and without boosting, all of which are superior over the two benchmarks. Moreover, all variants exhibited similar performances in terms of accuracy. The boosting of GA evidently increased the accuracy of Hyper 1 and Hyper 2 relative to the variant that did not use boosting.

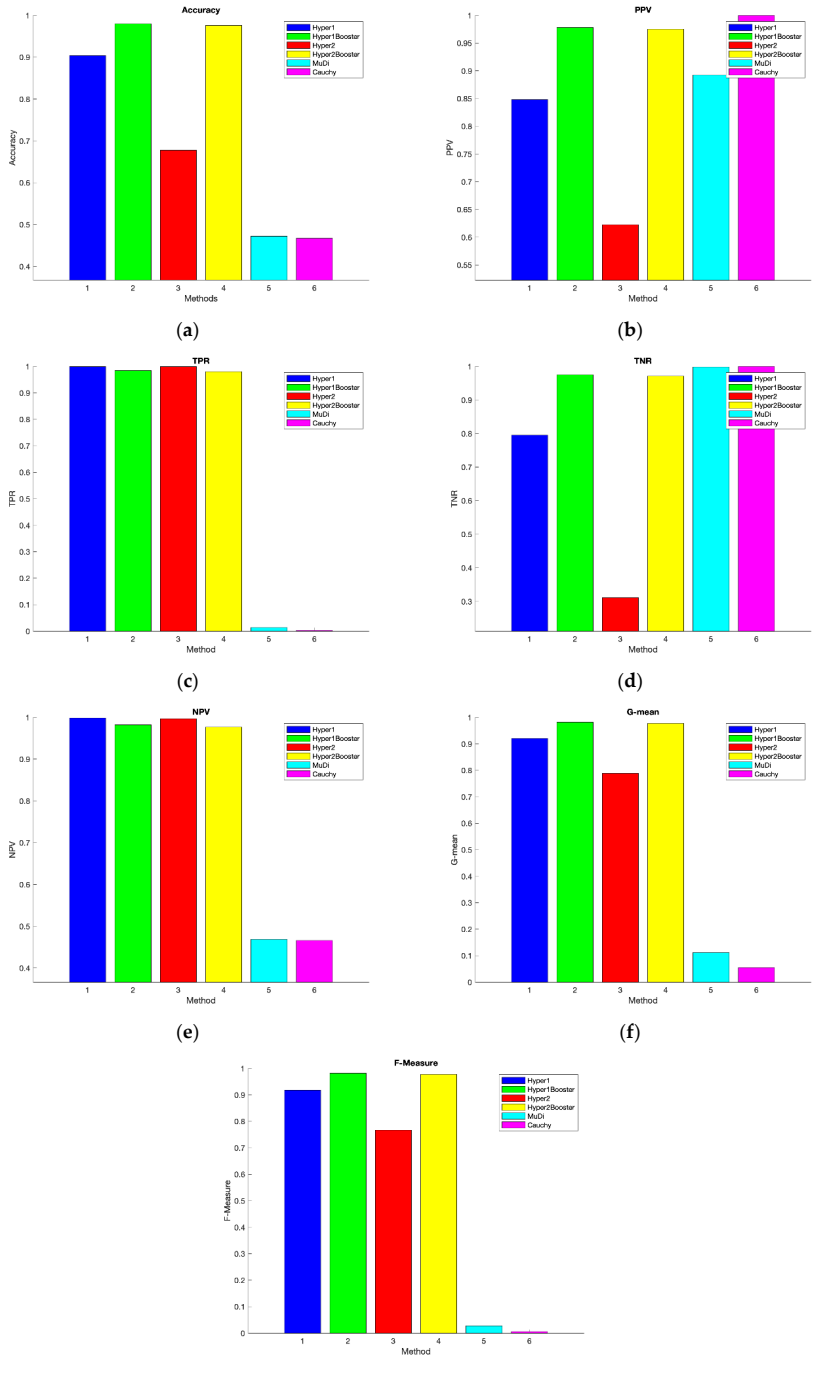

Figure 2. Classification measures hyper 1 and hyper 2 with/without boosting, MuDi and Cauchy for NSL-KDD, in terms of Accuracy (**a**), PPV (**b**), TPR (**c**), TNR (**d**), NPV (**e**), G-mean (**f**), F-Measured (**g**).

Figures 3 and 4 illustrate the classification measures of all variants of HH-F and the two benchmarks on KDD'99 and LandSat, respectively. The boosting failed the expectation of improving the accuracy of Hyper 1 over Hyper 2 (Figure 3). This phenomenon is interpreted by the higher amount of evolution in the KDD dataset than in the NSL-KDD. All variants of the HH-F outperformed MuDi and Cauchy. The accuracies of Hyper 1 and Hyper 2 are relatively similar for the majority of the experiments, but the accuracy of the latter is lower than the former unless boosting is performed to improve the performance.

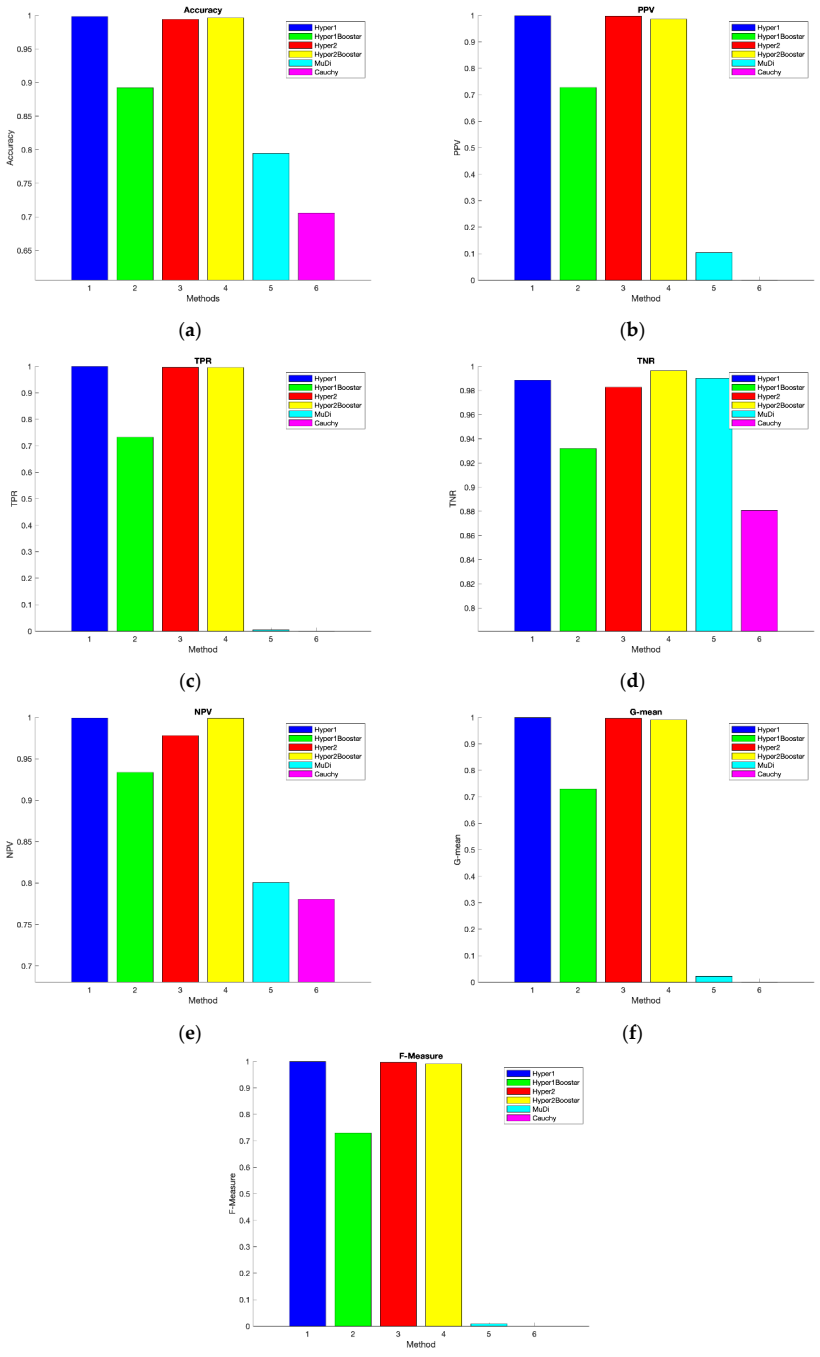

Figure 3. Classification measures hyper 1 and hyper 2 with/without boosting, MuDi and Cauchy for KDD99, in terms of Accuracy (**a**), PPV (**b**), TPR (**c**), TNR (**d**), NPV (**e**), G-mean (**f**), F-Measured (**g**).

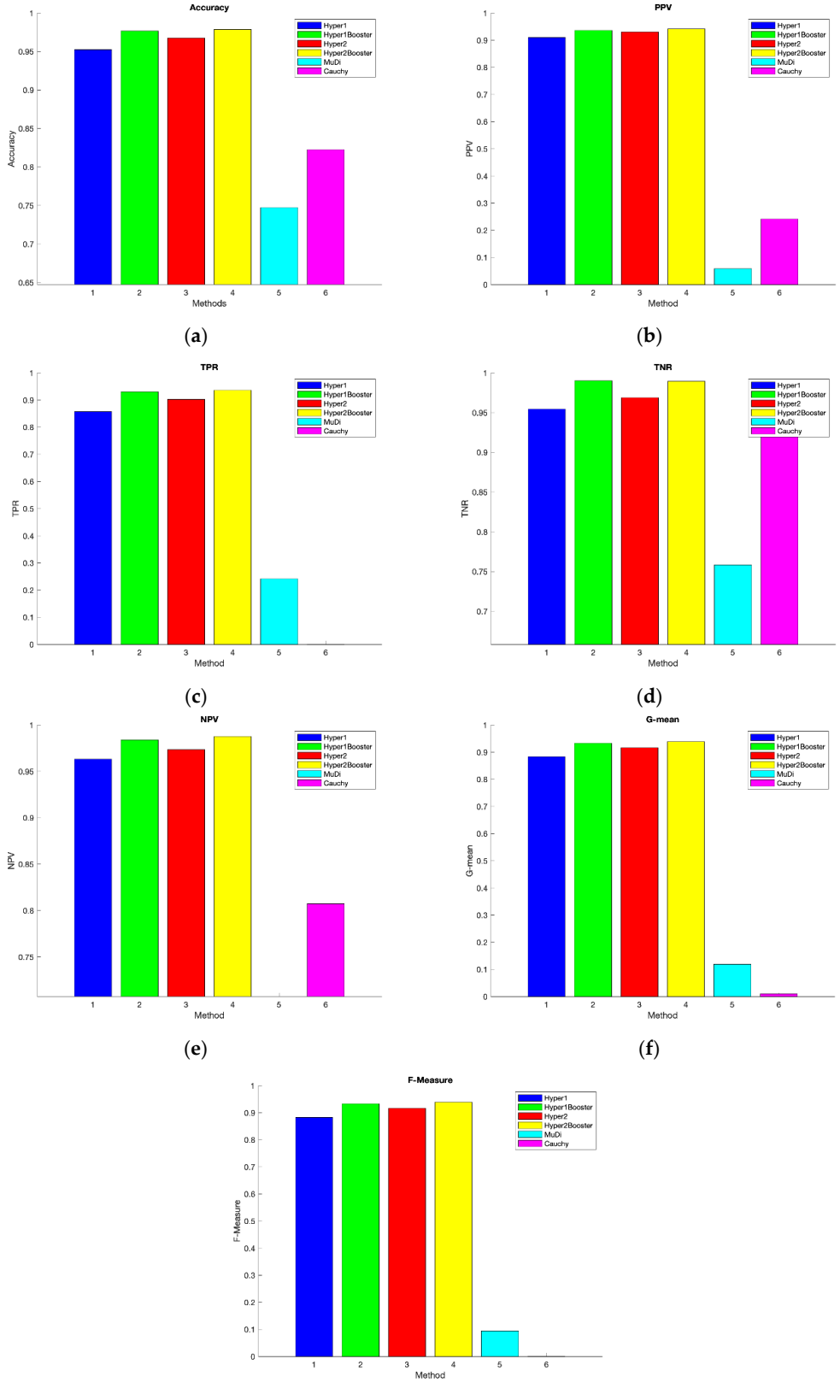

Figure 4. Classification measures hyper 1 and hyper 2 with/without boosting, MuDi and Cauchy for LandSat, in terms of Accuracy (**a**), PPV (**b**), TPR (**c**), TNR (**d**), NPV (**e**), G-mean (**f**), F-Measured (**g**).

Contrary to the case of (Figure 3), the boosting has succeeded in of improving the accuracy of Hyper 1 over Hyper 2 (Figure 4). This phenomenon is interpreted by the less amount of evolution in the LandSat dataset. All variants of the HH-F outperformed MuDi and Cauchy. This is interpreted by the nature of HH-F in terms of embedding a tracking block, i.e., the genetic integrated with the configuration, to update dynamically the parameters of core clustering which assures an ability of avoidance of concept drift impact. On the other side, similar to the majority of existing IDS algorithms, Cauchy and MuDi lack such a handling of concept drift impact.

The last part of the evaluation is the statistical analysis. Table 7 lists the t-values after comparing the HH-F variants and the two benchmarks in terms of accuracy. Hyper 1 and Hyper 2 demonstrated a statistical significance with respect to the superiority of HH-F over Cauchy and MuDi, which statistically proves that the former is superior over the two benchmarks.

Table 7. T-test values for the comparison of the accuracies of the HH-F variants with MuDi and Cauchy.

T-Test	Hyper 1	Hyper 2
Cauchy	1.59896×10^{-25}	1.1281×10^{-128}
MuDi	7.64165×10^{-16}	7.94039×10^{-16}

4. Conclusions and Future Work

This study proposed a hyper-heuristic semi-supervised framework on the basis of genetic and online sequential extreme learning machines for sequential data classification called HH-F. This framework is a combination of core online–offline clustering, an NN with one hidden layer, and genetic optimization with configuration and class prediction blocks. The integration of these blocks serves in an overall goal which is dealing with the concept drift and updating the parameters of the model dynamically. NN serves as the knowledge storage component of the model of assigning stream samples to various classes, online-offline clustering serves as the fast reacting block of the changes in the clusters shape, density and distribution and the evolutionary meta-heuristic genetic serves as the tracking component of the changes of the internal parameters of the online-offline clustering by leveraging the knowledge of the ground truth and the gained knowledge within NN. More specifically, the framework trains the NN on previously labeled data and its knowledge is used to calculate the error of the core online–offline clustering block. The genetic optimization is responsible for selecting the best parameters of the core model to counter the concept drift and handle the evolving nature of the classes. Four variants of the proposed framework were created. The first variant called Hyper 1 uses only GA in the prediction phase, whereas the second variant called Hyper 2 uses GA in the prediction and correction phases. Both variants were repeated twice with and without genetic boosting (i.e., the other two variants).

The evaluation was performed on three online datasets, namely NSL-KDD, KDD-99, and LandSat. For NSL-KDD and LandSat, the performance of partially labeled data was assessed. The result showed how the performance of Hyper 1 and Hyper 2 increased with the increase in the percentage of labeled data. Then, the performances of all variants were compared with those of Cauchy and MuDi. The variants outperformed the two benchmarks, and Hyper 1 and Hyper 2, with and without genetic boosting exhibited slight performance variations. Boosting exerts a positive influence on accuracy, except when high data evolution is involved. Future work will incorporate a block in the framework to address the imbalance issue of some data classes in the stream data and test the performance in real-world applications, such as minimizing false alarms in IDS.

Author Contributions: Conceptualization, A.A. (Ahmed Adnan), A.M., A.A.A.G., A.A. (Azizol Abdullah), and F.H.; validation, A.A. (Ahmed Adnan), and A.M.; Designed and performed the experiments, A.A. (Ahmed Adnan).; writing—original draft preparation, A.A.(Ahmed Adnan).; supervision, A.M., A.A.A.G., A.A. (Azizol Abdullah), and F.H.; funding acquisition, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, F.; Deng, P.; Wan, J.; Zhang, D.; Vasilakos, A.V.; Rong, X. Data mining for the internet of things: Literature review and challenges. *Int. J. Distrib. Sens. Netw.* **2015**, *2015*. [CrossRef]
2. Abaker, I.; Hashem, T.; Yaqoob, I.; Badrul, N.; Mokhtar, S.; Gani, A.; Ullah, S. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.* **2015**, *47*, 98–115.
3. Bello-orgaz, G.; Jung, J.J.; Camacho, D. Social big data: Recent achievements and new challenges. *Inf. Fusion* **2016**, *28*, 45–59. [CrossRef] [PubMed]
4. Lee, J.; Ardakani, H.D.; Yang, S.; Bagheri, B. Industrial big data analytics and cyber-physical systems for future maintenance & service innovation. *Procedia CIRP* **2015**, *38*, 3–7.
5. Moustafa, N.; Creech, G.; Slay, J. Big data analytics for intrusion detection system: Statistical decision-making using finite dirichlet mixture models. In *Data Analytics and Decision Support for Cybersecurity*; Springer: Cham, Switzerland, 2017.
6. Chen, M.; Ma, Y.; Song, J.; Bin, C.L. Smart clothing: Connecting human with clouds and big data for sustainable health monitoring. *Mob. Netw. Appl.* **2016**, *21*, 825–845. [CrossRef]
7. Goldstein, M.; Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS ONE* **2016**, *11*, e0152173. [CrossRef] [PubMed]
8. Lughofer, E.; Sayed-mouchaweh, M. Autonomous data stream clustering implementing split-and-merge concepts—Towards a plug-and-play approach. *Inf. Sci.* **2015**, *304*, 54–79. [CrossRef]
9. Pool, J.; Dally, W.J. Learning Both Weights and Connections for Efficient Neural Networks. Advances in Neural Information Processing Systems. 2015, pp. 1135–1143. Available online: https://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf (accessed on 18 June 2020).
10. Kang, M.; Kang, J. Intrusion detection system using deep neural network for in-vehicle network security. *PLoS ONE* **2016**, *11*, e0155781. [CrossRef] [PubMed]
11. Maitland, E. Decision making and uncertainty: The role of heuristics and experience in assessing a politically hazardous environment. *Strateg. Manag. J.* **2014**, *36*, 1554–1578. [CrossRef]
12. Metiaf, A.L.I.; Hong, W.U.Q.; Aljeroudi, Y. Searching with direction awareness: Multi-objective genetic algorithm based on angle quantization and crowding distance moga-aqcd. *IEEE Access* **2018**, *7*, 10196–10207. [CrossRef]
13. Zhang, Z.; Hong, W.C.; Li, J. Electric load forecasting by hybrid self-recurrent support vector regression model with variational mode decomposition and improved cuckoo search algorithm. *IEEE Access* **2020**, *8*, 14642–14658. [CrossRef]
14. Kundra, H.; Sadawarti, H. Hybrid algorithm of Cuckoo Search and Particle Swarm Optimization. *Res. J. Inf. Technol.* **2015**, *7*, 58–69.
15. Hong, W.C.; Dong, Y.; Lai, C.Y.; Chen, L.Y.; Wei, S.Y. SVR with hybrid chaotic immune algorithm for seasonal load demand forecasting. *Energies* **2011**, *4*, 960–977. [CrossRef]
16. Deng, S.; Wang, B.; Huang, S.; Yue, C.; Zhou, J.; Wang, G. Self-adaptive framework for efficient stream data classification on storm. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *50*, 123–136. [CrossRef]
17. Li, Y.; Wang, Y.; Liu, Q.; Bi, C.; Jiang, X.; Sun, S. Incremental semi-supervised learning on streaming data. *Pattern Recognit.* **2019**, *88*, 383–396. [CrossRef]
18. Ksieniewicz, P.; Woźniak, M.; Cyganek, B.; Kasprzak, A.; Walkowiak, K. Data stream classification using active learned neural networks. *Neurocomputing* **2019**, *353*, 74–82. [CrossRef]
19. Junior, B.; do Carmo Nicoletti, M. An iterative boosting-based ensemble for streaming data classification. *Inf. Fusion* **2019**, *45*, 66–78. [CrossRef]
20. Casalino, G.; Castellano, G.; Mencar, C. Data stream classification by dynamic incremental semi-supervised fuzzy clustering. *Int. J. Artif. Intell. Tools* **2019**, *28*, 1–26. [CrossRef]
21. Noorbehbahani, F.; Fanian, A.; Mousavi, R.; Hasannejad, H. An incremental intrusion detection system using a new semi-supervised stream classification method. *Int. J. Commun. Syst.* **2017**, *30*, 1–26. [CrossRef]
22. Skrjanc, I.; Ozawa, S.; Ban, T.; Dov, D. Large-scale cyber attacks monitoring using Evolving Cauchy Possibilistic Clustering. *Appl. Soft Comput.* **2018**, *62*, 592–601. [CrossRef]

23. Sethi, T.S.; Kantardzic, M.; Hu, H. A grid density based framework for classifying streaming data in the presence of concept drift. *J. Intell. Inf. Syst.* **2016**, *46*, 179–211. [CrossRef]
24. Fahy, C.; Yang, S.; Gongora, M. Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams. *IEEE Trans. Cybern.* **2019**, *49*, 2215–2228. [CrossRef]
25. Fahy, C.; Yang, S. Finding and Tracking Multi-Density Clusters in Online Dynamic Data Streams. *IEEE Trans. Big Data* **2019**. [CrossRef]
26. Bai, L.; Cheng, X.; Liang, J.; Shen, H. An optimization model for clustering categorical data streams with drifting concepts. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 2871–2883. [CrossRef]
27. Amini, A.; Saboohi, H.; Herawan, T.; Wah, T.Y. MuDi-Stream: A multi density clustering algorithm for evolving data stream. *J. Netw. Comput. Appl.* **2016**, *59*, 370–385. [CrossRef]
28. Huang, G.; Liang, N.; Rong, H.; Saratchandran, P.; Sundararajan, N. On-Line Sequential Extreme Learning Machine Review of Extreme Learning Ma- Proposed Online Sequential Ex- treme Learning Machine. *Comput. Intell.* **2005**, *2005*, 232–237.
29. Abbas, M.; Albadr, A.; Tiun, S. Extreme learning machine: A review. *Int. J. Appl. Eng. Res.* **2017**, *12*, 4610–4623.
30. Huang, G.; Huang, G.-B.; Song, S.; You, K. Trends in extreme learning machines: A review. *Neural Netw.* **2015**, *61*, 32–48. [CrossRef]
31. Akusok, A.; Björk, K.; Miche, Y.; Lendasse, A. High-performance extreme learning machines: A complete toolbox for big data applications. *IEEE Access* **2015**, *3*, 1011–1025. [CrossRef]
32. Brownfield, B.; Lemos, T.; Kalivas, J.H. Consensus classification using non-optimized classifiers. *Anal. Chem.* **2018**, *90*, 4429–4437. [CrossRef]
33. Hong, X.; Chen, S.; Harris, C.J. A kernel-based two-class classifier for imbalanced data sets. *IEEE Trans. Neural Netw.* **2007**, *18*, 28–41. [CrossRef] [PubMed]
34. Joshi, M.V. On Evaluating Performance of Classifiers for Rare Classes. In Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM, Maebashi City, Japan, 9–12 December 2002; pp. 641–644.
35. Lan, Y.; Wang, Q.; Cole, J.R.; Rosen, G.L. Using the RDP classifier to predict taxonomic novelty and reduce the search space for finding novel organisms. *PLoS ONE* **2012**, *7*, e32491. [CrossRef] [PubMed]
36. Seliya, N.; Khoshgoftaar, T.M.; Van Hulse, J. A Study on the Relationships of Classifier Performance Metrics. In Proceedings of the 2009 21st IEEE International Conference on Tools with Artificial Intelligence ICTAI, Newark, NJ, USA, 2–4 November 2009; pp. 59–66.
37. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A Detailed Analysis of the KDD CUP 99 Data Set. In Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–6. [CrossRef]