Original Article
International Journal of Fuzzy Logic and Intelligent Systems
Vol. 21, No. 2, June 2021, pp. 101-122
http://doi.org/10.5391/IJFIS.2021.21.2.101

ISSN(Print) 1598-2645
ISSN(Online) 2093-744X

Data Stream Classification Algorithms for Workload Orchestration in Vehicular Edge Computing: A Comparative Evaluation

Mutaz Al-Tarawneh
Computer Engineering Department, Mutah University, Karak, Jordan

Abstract

This paper reports on the use of online data stream classification algorithms to support workload orchestration in vehicular edge computing environments. These algorithms can be used to predict the ability of available computational nodes to successfully handle computational tasks generated from vehicular applications. Several online data stream classification algorithms have been evaluated based on synthetic datasets generated from simulated vehicular edge computing environments. In addition, a multi-criteria decision analysis technique was utilized to rank the different algorithms based on their performance metrics. The evaluation results demonstrate that the considered algorithms can handle online classification operations with various trade-offs and dominance relations with respect to their obtained performance. In addition, the utilized multi-criteria decision analysis technique can efficiently rank various algorithms and identify the most appropriate algorithms to augment workload orchestration. Furthermore, the evaluation results show that the leveraging bagging algorithm, with an extremely fast decision tree base estimator, is able to maintain marked online classification performance and persistent competitive ranking among its counterparts for all datasets. Hence, it can be considered a promising choice to reinforce workload orchestration in vehicular edge computing environments.

Keywords: Online classification, Data stream, Performance, Ranking

Received: Feb. 3, 2021
Revised : Mar. 28, 2021
Accepted: Apr. 26, 2021

Correspondence to: Mutaz Al-Tarawneh
(mutaz.altarawneh@mutah.edu.jo)
©The Korean Institute of Intelligent Systems

©This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc / 3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

In addition to the rapid development of artificial intelligence and data mining techniques, Internet of Things (IoT) technology has transformed traditional transportation systems into intelligent transportation systems (ITS) [1]. In the ITS paradigm, vehicles are equipped with various types of processing, sensing, detection, and communication devices, leading to more intelligent and connected vehicles. Therefore, a plethora of intelligent applications, such as autonomous driving, smart navigation, and infotainment, has been developed [2–4]. However, these applications are computationally intensive, with processing requirements that surpass the processing capacity of in-vehicle hardware resources. This discrepancy between the application requirements and the computing power of the in-vehicle hardware resources can hinder the development of ITS-centered applications [5]. To remedy this situation, cloud computing has been employed within vehicular networks, and applications have been allowed to utilize the powerful hardware resources of remote cloud data centers [6]. However, cloud

data centers may incur very long transmission delays that violate the extremely low latency requirements of some applications in vehicular networks. To overcome this issue, vehicular edge computing (VEC), which combines vehicular networks with multi-access edge computing (MEC), has been proposed [7]. In the VEC paradigm, several edge servers are deployed at the network edge, providing adequate computing power to handle the increasing processing requirements of in-vehicle applications.

Hence, VEC is considered a promising concept that provides a baseline for various applications to enhance the quality of service (QoS) based on computation offloading [8]. It can improve the user's quality of experience by offloading computational tasks to the edge servers, which are expected to handle the processing demand of vehicular applications. Thus, it can handle delay-sensitive vehicular applications [9,10]. The envisioned VEC scenarios encompass a dynamically changing process characterized by an evolving stream of offloading requests generated from a diverse set of applications with varying processing demands and delay requirements, an instantaneously changing network state, and an oscillatory utilization and capacity of the hardware resources of both edge and cloud servers [11]. In such a dynamic and heterogeneous environment, workload orchestration plays a vital role in handling the incoming offloading requests generated from different applications and deciding on the computational server to which each request should be offloaded, considering the application characteristics, network status, and utilization levels of both edge and cloud hardware resources. Apparently, as the number of vehicles and in-vehicle applications increases, the demand for the networking and hardware resources of the VEC system will also increase. The increased demand for these resources will ultimately lead to increased competition for resources, hindering the decision-making process involved in the operation of workload orchestration [12]. Consequently, predicting the status of hardware resources and their ability to successfully handle offloaded requests is a challenging yet inevitable process. In this regard, machine learning (ML) algorithms can provide efficient tools for making predictions in dynamically changing environments [13]. The ultimate goal of the prediction process is to help the workload orchestrator determine whether offloading an incoming request to a particular server will eventually succeed or fail. In other words, the prediction process associated with workload orchestration is a binary classification problem in which a class label is assigned to each incoming offloading request. The class label will be set to either success or failure. Whereas a success indicates that a particular computational server is predicted to successfully handle the offloaded request and deliver the execution results back to the request source, a failure indicates that the offloading process is expected to fail because of either vehicle mobility, network congestion, or unavailability of processing resources. In this context, the work in [13] is among the first efforts to propose an ML-based workload orchestrator for VEC systems. They used several classification algorithms and tested their ability to aid in the workload orchestration process. However, they used these algorithms in a batch-learning scheme. In this scheme, an ML model is built under the assumption that the entire dataset (i.e., offloading requests) is available in memory. However, batch learning schemes suffer from several limitations. First, the training phase may last for a long period and require a significant amount of processing and memory resources. Second, the performance of the trained model is sensitive to the training dataset size. Third, in the batch learning scheme, the training data are assumed to be static and do not change over time; once a model is trained, it cannot acquire knowledge or experience from new samples. In other words, when there is a change in the statistical properties of the model's input (i.e., concept drift), a new model must be constructed [14]. Consequently, as the offloading requests in VEC environments represent a continuous stream of data, online data stream classification algorithms are viable alternatives to offline (i.e., batch) classification algorithms in VEC-based workload orchestration for several reasons. First, online data stream classification algorithms are designed to handle unbounded streams of data and incrementally learn from incoming data; they must be continuously updated while making predictions when required [15]. Second, online data stream classification algorithms can be utilized in real-time applications that cannot be handled by classical (i.e., batch) learning algorithms [16]. Third, they are able to detect concept drift, allowing the trained models to continuously adapt and evolve with dynamically changing data streams [17,18]. Therefore, online data stream classification is perceived as a suitable tool for workload orchestration in VEC systems, where the environmental state and application behavior may change over time in an unpredictable manner. Many algorithms for data stream classification have been proposed [19–21]. Although some of these algorithms have been studied in multiple fields [16,22–25], the evaluation of their performance and the trade-offs involved in their performance metrics and memory costs have not been addressed for VEC environments. Hence, this work seeks to identify which online data stream classification algorithms may be suitable for VEC environments through rigorous comparative

evaluation. In addition, it aims to analyze the trade-offs associated with key performance metrics, such as the algorithm's accuracy, precision, recall, kappa statistic, and memory requirements [26]. Typically, the evaluation of online data stream classification algorithms involves multiple, typically conflicting criteria. Hence, it can be modeled as a multi-criteria decision analysis (MCDA) problem [27]. MCDA is a formal process that can be employed in decision-making by allowing identification of the alternatives, selection of the evaluation indicators (i.e., criteria), and aggregation of the obtained performance scores under the considered performance evaluation criteria [28,29]. In this context, the composite indicators (CIs) are a family of MCDA methods that can be used to assign scores to various alternatives and rank them appropriately [30–32]. Hence, this work employs a CI-based approach to rank various online data stream classification algorithms based on their obtained performance metrics. The experiments were carried out using several online data stream classification algorithms (Section 2) based on synthetically generated datasets. All tests were performed using a scikit-multiflow evaluation platform [33]. The main contributions of this study can be summarized as follows.

- Performing a rigorous evaluation of the performance of several online data stream classification algorithms on synthetic datasets generated from a simulated VEC environment.
- Applying a MCDA technique to rank various algorithms based on their observed performance trade-offs and dominance relations.
- Identifying the most suitable algorithm or set of algorithms to augment workload orchestration in VEC environments.

The remainder of this paper is organized as follows. Section 2 briefly explains the online data stream classification algorithms considered in this study. Section 3 describes the research methodology and utilizes research tools. Section 4 presents the results of the evaluation and trade-off analysis. Section 5 summarizes and concludes this paper.

2. Data Stream Classification

Traditionally, ML algorithms have been used to construct models based on static datasets. However, there is a growing need for mechanisms capable of handling vast volumes of data that arrive as streams. In this regard, new data samples can be obtained at any time, and storing these data samples is inappropriate. On the one hand, learning from continuous data streams requires the ML model to be created and continuously updated throughout the stream. On the other hand, it is important to address concept drift, in which the statistical properties of the incoming data change over time [34,35]. In addition, for VEC environments, the obtained ML model must be updated instantaneously, requiring algorithms that can achieve adequate accuracy levels under limited processing power and memory space [26]. In this work, several data stream classification algorithms, with distinct model construction mechanisms, complexity, and memory requirements, are considered. The algorithms are summarized as follows.

2.1 Bayes Learning Algorithms

In this category, the naive Bayes (NB) algorithm was used [20]. The NB algorithm performs Bayesian prediction under the assumption that all inputs, that is, features of the input data samples, are independent. The NB algorithm is a simple classification algorithm with low processing requirements. Given n different classes, the trained NB classifier predicts the class to which it belongs with a high probability for every incoming data sample.

2.2 Lazy Learning Algorithms

The most commonly used lazy data stream classification algorithms are the k-nearest neighbors classifier (kNN) [36], kNN classifier with adaptive windowing (kNN-ADWIN) change detector,, and self-adjusting memory coupled with the kNN classifier (SAM-kNN) [37,38]. In the data stream setting, the kNN algorithm operates by keeping track of a window with a fixed number of recently observed training data samples. When a new input data sample arrives, the kNN algorithm searches within the recently stored samples and finds the closest neighbors using a particular distance measure. Then, the class label of the incoming data sample can be assigned accordingly. The kNN-ADWIN classifier is an improvement over the regular kNN classifier because of its resistance to concept drift. It employs the ADWIN change detector to determine which previous data samples to keep and which ones to discard (i.e., forget), which in turn regulates the sample window size. In addition, the SAM-kNN is a refinement over the other two algorithms, in which a SAM model constructs an ensemble of models targeting either current or previous concepts. These dedicated models can be applied according to the requirements of the

current situation (i.e., concept). To accomplish this, the SAM model constructs both short-term (STM) and long-term (LTM) memories. Whereas the STM is built for the current concept, the LTM is used to store information about previous concepts. A cleaning process is then used to control the size of the STM while maintaining consistency between the LTM and STM.

2.3 Tree-Based Learning Algorithms

Tree-based algorithms are widely used in data-stream classification applications. In this study, three main tree-based algorithms are used: the Hoeffding tree (HT) [39], Hoeffding adaptive tree (HAT) [40], and extremely fast decision tree (EFDT) [41]. The HT algorithm is an incremental, anytime decision tree induction algorithm that has the ability to learn from massive online data streams, assuming that the distribution generating the incoming data samples is static and does not change over time. It relies on the fact that a small set of input samples can often be sufficient to select an optimal splitting attribute. This idea is mathematically supported by the Hoeffding bound, which quantifies the number of input samples required to estimate some statistics within a predefined precision. A theoretically attractive feature of the HT algorithm that is not shared by other online decision tree algorithms is that it has profound performance guarantees. Relying on the Hoeffding bound, the output of an HT learner is asymptotically nearly identical to that of a batch-based learner using infinitely many input data samples. The HAT algorithm utilizes the ADWIN change detector to monitor the performance of different tree branches. The branches whose performance decreases are replaced with new branches if the new branches are more accurate. Furthermore, the EFDT classification algorithm builds a tree in an incremental manner. It selects and deploys a split once it is confident that the split is useful and subsequently revisits that split decision, replacing it if it then becomes evident that a more useful split is present. The EFDT is able to learn quickly from static distributions and ultimately learn the asymptotic batch tree if the distribution generating the input data samples is stationary.

2.4 Rule-Based Learning Algorithms

The very fast decision rule (VFDR) is an online data stream classification algorithm [42]. The learning process of the VFDR algorithm is similar to that of the HT algorithm, but instead utilizes a collection of rules instead of a tree. The ultimate goal of VFDR is to create a collection of rules that constitute a highly interpretable classifier. Each rule is represented as a combination of conditions based on attribute values and a structure for maintaining sufficient statistics. These statistics are used to determine the class predicted by the associated rule for incoming data samples.

2.5 Ensemble Learning Algorithms

In this study, several ensemble learning methods are evaluated. These algorithms include Oza bagging (OB) [43], Oza bagging with ADWIN change detector (OB-ADWIN) [43], leveraging bagging (LB) [44], online synthetic minority oversampling technique bagging (SMOTE-B) [45], online under-over bagging (UO-B) [45], adaptive random forest (ARF) [46], and the dynamic weighted majority classifier (DWMC) [47]. For instance, OB is an online ensemble learning algorithm that can be considered a refinement over traditional batch-based bagging to handle incremental learning. For traditional batch-based bagging, M classifiers are trained on M distinct datasets created by drawing N samples from an N-sized training dataset with replacement. In the online learning settings, as there is no training dataset, but rather a stream of input data samples, the drawing of input samples with replacements cannot be trivially performed. The online OB mimics the training phase by using each arriving input data sample to train the base estimator over k times, which is drawn by the binomial distribution. Because the input data stream can be assumed to be infinite, and given that the binomial distribution tends to a Poisson (λ =1) distribution with infinite samples, [43] found the process adopted by the online OB algorithm to be a good 'drawing with replacement'. OB-ADWIN is an improvement from the OB algorithm, where the ADWIN change detector is employed. In addition, the LB algorithm is based on the OB algorithm, in which a Poisson distribution is used to simulate the re-sampling process. The LB algorithm attempts to obtain better classification results by modifying the parameters of the Poisson distribution obtained from the binomial distribution when assuming an infinite input data stream. Hence, the LB algorithm changes the λ parameter of the Poisson distribution to six instead of one. The new value of λ would lead to greater diversity in the input space by attributing a different range of weights to the input data samples. To achieve further improvement over the OB algorithm, the LB uses output detection codes. In the detection codes, each class label is coded using an n-bit-long binary code, and each bit is associated with one of the n classifiers. As a new input data sample is analyzed, each classifier is trained on its associated bit. This assists the LB algorithm, to some extent, with error

correction.

The ARF algorithm is an adaptation of the traditional batch-based random forest algorithm applied to an online data stream scope. In ARF, multiple decision trees are generated, and the decision on how to classify each incoming data sample is made through a weighted voting system. In the voting process, the decision tree with the best performance (in terms of either accuracy or the kappa statistic) receives a more substantial weight in the voting process, that is, a higher priority in the classification decision. The DWMC algorithm relies on four mechanisms to handle concept drift: it first trains online base learners of the ensemble; it then assigns weights to the base learners depending on their performance, removes them based on their performance, and adds new learners based on the global performance of the whole ensemble.

In the UO-B and SMOTE-B algorithms, the basic idea is to use existing algorithms for online ensemble-based learning and combine them with batch-mode techniques for cost-sensitive bagging. Such a combination is useful for classifying imbalanced data streams in which the vast majority of incoming data samples belong to the same class. The UO-B and SMOTE-B algorithms represent an adaptation of the UnderOverBagging and SMTEBagging ensemble methods [48] for online data stream settings. Given an imbalanced dataset with N samples, where N^+ samples come from the minority class S^+ and N^- samples from the majority class S^-, batch bagging-based learning algorithms rely on under-sampling (underbagging) the majority class, over-sampling (overbagging) the minority class, or UnderOverBagging, which is a uniform combination of both underbagging and overbagging. Furthermore, the re-sampling rate (a) varies over the bagging iterations, boosting the diversity among the base learners. In SMOTEBagging, the majority class is sampled with replacement at a rate of 100% (i.e., N^- from the majority class is generated), while CN^+ samples from the minority class are generated for each base learner, for some $C > 1$, among which $a\%$ is generated by re-sampling, while the rest of the samples are obtained by the SMOTE technique [49]. In the online data stream settings, where no dataset is available, the sampling process is simulated by presenting each incoming data sample to the base model over k times, where k is sampled from a Poisson (λ) distribution. The value of the parameter λ is chosen in a manner that allows an online bagging scheme to mimic the behavior of its batch-based counterpart.

3. Research Tools and Methodology

3.1 VEC System Model

Figure 1 shows the system model assumed in this work. This model is based on the work proposed in [13,50], where a model and a simulation environment for a VEC system were proposed. In this model, in-vehicle applications are assumed to periodically generate offloading requests for some computational tasks. In this work, the workload orchestrator is assumed to offload the current request (i.e., task) to either: (A) the road side unit (RSU) via the wireless local area network (WLAN) that currently covers the offloading vehicle, (B) the cloud server through the RSU - using the wide area network (WAN), or (C) the cloud server via the cellular network (CN).

It is assumed that there are three main in-vehicle application types. These applications and their corresponding tasks are uniquely characterized by a set of parameters, as shown in Table 1. The usage percentage is the percentage of vehicles running each application, task input file size is the mean size of the input data required by the offloaded task, task output file size is the average size of the output result generated after executing the offloaded task, virtual machine (VM) utilization indicates the percentage of the processing power consumed (i.e., utilized) by the offloaded task, and the number of instructions per task represents the expected processing demand of the offloaded task. As shown in Table 1, the characteristics of different applications were chosen such that adequate diversification in their task offloading frequency, processing demand, and network bandwidth requirements is achieved. Such a diverse set of applications will

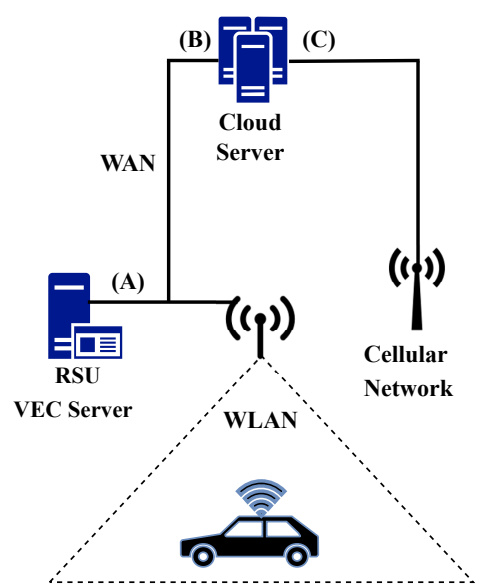

Figure 1. System model.

Table 1. In-vehicle application characteristics

	App-1	App-2	App-3
Usage percent	30	35	35
Task input file size (kB)	20	40	20
Task output file size (kB)	20	20	80
Task generation rate (per second)	0.3	0.2	0.07
Number of instructions per task ($\times 10^9$)	3	10	20
VM utilization on RSU (%)	6	20	40
VM Utilization on Cloud (%)	1.6	4	8

cause the simulated VEC environment to go through various states that cover a wide range of computational server loading and network bandwidth utilization levels.

3.2 Dataset Construction

To carry out a comparative evaluation between different data stream classification algorithms, synthetic datasets were generated based on the simulation tool of [13]. The datasets were created by simulating a VEC system with 2,400 vehicles and application characteristics, as shown in Table 1. In addition, the VEC simulation was accompanied by a hypothetical workload orchestrator that offloads the incoming requests according to Table 2. Each entry in Table 2 represents the probability that the orchestrator will make a particular decision for an incoming offloading request. As shown, these probabilities are application-dependent; for instance, there is a 0.60 probability of offloading tasks generated from App-1 to the VEC server, as compared with probabilities of 0.30 and 0.23 for App-2 and App-3, respectively. These offloading probabilities were set to mimic a relatively realistic situation that aligns with the application characteristics shown in Table 1. For instance, both App-2 and App-3 are more computationally intensive than App-1; they have a greater number of instructions per task. In addition, the average input/output file size is higher than that of App-1. Hence, they were assigned higher probabilities of being offloaded to the resource-enriched cloud server via the high-bandwidth WAN connection. Overall, the hypothetical orchestration behavior shown in Table 2 was observed to generate synthetic datasets with an adequate number of success and failure instances, leading to a more appropriate setting for evaluating the performance and discrimination ability of different online data stream classification algorithms. For each offloading request, a record-keeping process was performed. Record keeping involves recording the VEC environmental state when

Table 2. Hypothetical orchestration probabilities

	VEC server	Cloud via RSU	Cloud via CN
App-1	0.60	0.23	0.17
App-2	0.30	0.53	0.17
App-3	0.23	0.60	0.17

Table 3. VEC environment state variables per offloading decision

Variable	Decision
Number of instructions per task	All
Number of recently uploaded tasks	All
Input file size (kB)	All
Output file size (kB)	All
Edge server utilization (%)	VEC server
WLAN upload delay (ms)	VEC server
WLAN download delay (ms)	VEC server
WAN upload delay (ms)	Cloud via RSU
WAN download delay (ms)	Cloud via RSU
CN upload delay (ms)	Cloud via CN
CN download delay (ms)	Cloud via CN

the offloading decision is made, in addition to the offloading outcome. The environmental state variables depend on the decision made by the workload orchestrator, as shown in Table 3. The environmental state variables contain information about application characteristics related to the offloaded task, average utilization of the VMs instantiated on the VEC server, upload and download delays associated with network connections used for task offloading, and the number of tasks recently offloaded to the selected server.

The offloading outcome is set to success if the task associated with the offloaded request is offloaded successfully and its result is delivered to the request source. It is set to failure if the execution result is not delivered successfully to the requesting vehicle. The failure of the offloading process may be due to several reasons, such as high resource utilization of the selected server, unavailability of network bandwidth, or vehicle mobility. Ultimately, the records obtained during the simulation process were used to create three different datasets, with each dataset corresponding to one of the three possible offloading decisions. These datasets are denoted as the Edge, CloudRSU, and CloudCN datasets. Each dataset stores the environmental state variables associated with each offloading request, along with the offloading outcome. The stored entries appear in chronological order. Table 4 lists some sample entries

Table 4. Sample dataset entries

	Edge dataset		CloudRSU dataset		CloudCN dataset	
	Success	**Failure**	**Success**	**Failure**	**Success**	**Failure**
Number of instructions per task	3,057	3,178	9,531	19,242	3,196	9,598
Number of previously offloaded tasks	2	129	-	-	-	-
Number of offloaded tasks	-	-	1	184	1	92
Input file size	19	21	21	88	19	21
Output file size	18	21	42	22	20	36
Edge server utilization	25	20.9	-		-	-
WLAN upload delay	23.02	84.45	-		-	-
WLAN download delay	36.20	28.44	-		-	-
WAN upload delay	-	-	6.18	16.30	-	-
WAN download delay	-	-	12.07	55.65	-	-
CN upload delay	-	-	-	-	21.65	8
CN download delay	-	-	-	-	68.6	20.2

in the constructed datasets.

3.3 Evaluation Methodology

This section explains the main steps implemented to evaluate the performance of the considered data stream classification algorithms on the obtained datasets. Figure 2 depicts the evaluation procedure followed by the scikit-multiflow evaluation platform [33].

This evaluation procedure was performed for every data steam classification algorithm for each of the obtained datasets. As shown in Figure 2, the dataset is first loaded as a stream and then passed to the instantiated classification algorithm for online testing, incremental learning, and evaluation. In the evaluation platform, the instances contained in the obtained stream maintain their chronological order captured from the simulated VEC environment. In addition, each instance contains a copy of the dynamic state (i.e., application characteristics, designated server utilization, and network status) of the VEC environment upon making the orchestration decision. As a result, each tested algorithm in the evaluation platform is exposed to state variables that capture the state of the simulated VEC environment when making a classification decision. In this study, the classification algorithms were evaluated using a prequential evaluation method or the interleaved test-then-train method. The prequential evaluation method is designed specifically for online data stream settings, in the sense that each incoming input data sample (i.e., instance) serves two purposes, and those input

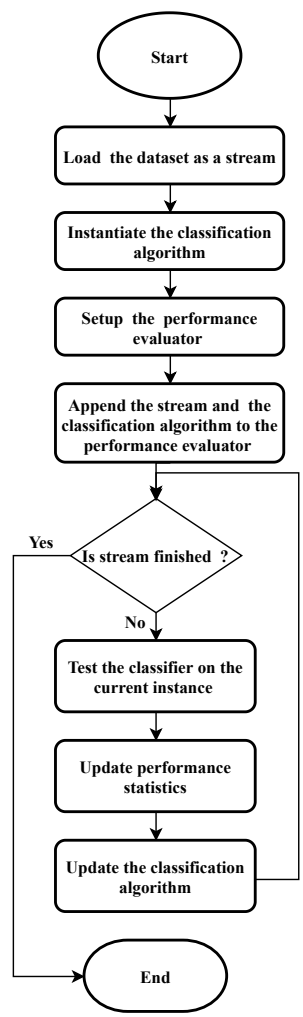

Figure 2. Evaluation flowchart.

instances are analyzed sequentially, in order of their arrival, and become immediately inaccessible. In this method, each incoming data sample is first used to test the classification algorithm (i.e., to make a prediction), after which the same input sample is used to train the classification algorithm. The considered performance metrics are incrementally updated after each observed instance, allowing a continuous update of the performance of each tested algorithm in addition to real-time tracking of its adaptability to unseen instances. Hence, the classification algorithm is always tested, and its performance metrics are updated on input data samples that have not yet been seen. The performance of the classification algorithms is quantified in terms of a number of widely used performance measures, including accuracy, precision, recall, F-score, kappa statistic, and memory size of the ML model obtained online. These measures are defined as follows:

- **Classification accuracy**: the percentage of correctly classified input samples.

$$Accuracy = \frac{TN + TP}{TP + FP + FN + TN}, \tag{1}$$

  where TP, TN, FP, and FN denote true positive, true negative, false positive, and false negative, respectively. TP is the number of correctly classified positive (i.e., successful) input instances. TN is the number of correctly classified negative instances (i.e., failure). FP is the number of positive instances that are misclassified as negative. FN is the number of negative instances that are misclassified as positive.

- **Precision:** measures the number of positive class predictions that actually belong to the positive class.

$$Precision = \frac{TP}{TP + FP}. \tag{2}$$

- **Recall:** quantifies the number of positive class predictions made out of all positive instances in the observed stream.

$$Recall = \frac{TP}{TP + FN}. \tag{3}$$

- **F-score:** is the harmonic mean of precision and recall.

$$F - score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{4}$$

- **Kappa statistic** (κ): is a robust classification accuracy measure that takes into account the probability of agreement by chance, indicating an improvement over a majority class classifier, which predicts all incoming samples fall in the majority class [51]. It plays a crucial role in evaluating classification accuracy, especially for imbalanced data streams.

$$\kappa = \frac{p_0 - p_c}{1 - p_c}, \tag{5}$$

  where p_0 is the prequential accuracy of the classifier and p_c is the probability that a chance classifier makes a correct prediction [52]. If $\kappa = 1$, the classification algorithm is always correct.

- **Model size**: is a measure of the memory space occupied by the classification model obtained throughout the prequential evaluation process.

Typically, the online classification of data streams does not have a single optimal solution (i.e., algorithm) that optimizes all of the involved performance metrics, but rather a plethora of possible solutions with noticeable trade-offs between different performance metrics. For such scenarios, the notion of optimality is captured through Pareto Optimalty, which considers an algorithm to be inferior or superior to another algorithm only if it is inferior in all metrics or superior in all metrics. The trade-offs in the algorithm selection process are captured by algorithms that are superior with respect to some metrics but inferior in other metrics. Such pairs of algorithms that are both superior and inferior with respect to certain metrics are called non-dominated and form the Pareto front of the algorithm performance optimization problem [53]. Hence, this work identifies, for each potential dataset, the set of non-dominated algorithms (SNDA) based on the obtained performance metrics. Having obtained the SNDA, a CI-based MCDA procedure is followed to assign scores and ranks to the various algorithms. The main steps followed by the CI-based procedure are shown in Figure 3. First, a set of non-dominated algorithms was used to construct a performance matrix (SNDA matrix).

The rows of the matrix represent the alternatives (i.e., algorithms), whereas the columns indicate the performance of these algorithms under each performance metric or indicator. Second, performance metrics are assigned weights that can be considered trade-off coefficients, meaning that they represent the decrease in indicator (x) that can be compensated for by another indicator (y). Third, the performance matrix was normalized using one of the eight possible normalization techniques. The

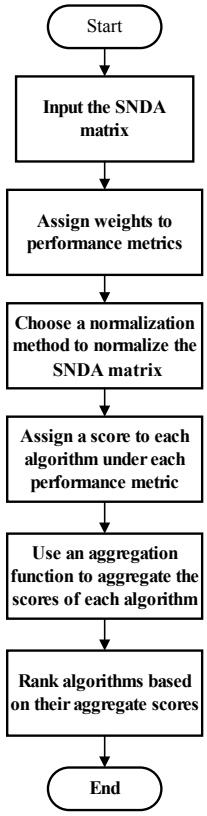

Figure 3. CI MCDA flowchart.

normalization techniques include the rank, percentile rank, standardized, min-max, logistic, categorical (-1,0,1), categorical (0.1,0.2,0.4,0.6,0.8,1), and target methods [29]. The outcome of the CI-based procedure can ultimately weigh or rank the SNDA and identify which algorithms are best suited for the classification problem in question. The normalization step helps in putting all performance indicators on the same scale and assigning a score to each alternative with respect to its counterparts. Fourth, the scores assigned after the normalized process are combined using a particular aggregation function to generate an index for each alternative (i.e., algorithm). These aggregation functions include additive, geometric, harmonic, minimum, and median functions [29]. Fifth, indices assigned to different algorithms are used to assign ranks to these algorithms. The ranks are in the range of 1 (best rank) to n (worst rank), where n is the number of elements in the SNDA. In this work, the steps outlined in the flowchart are performed for each possible combination of normalization and aggregation methods. Hence, an algorithm may be assigned different ranks under different combinations. To combine the results obtained under different combinations of normalization and aggregation, a rank frequency metric (rfm) is defined as shown in Eq. 6.

$$rfm_{A-i} = \frac{n_{rank_i}}{n_{combinations}} \times 100\%, \tag{6}$$

where rfm_{A-i} is the rank frequency metric of algorithm A, which shows the proportion of indices that rank algorithm A in the ith position, n_{rank_i} is the number of times the algorithm A is chosen for the ith rank, and $n_{combinations}$ is the number of possible combinations of normalization and aggregation under which algorithm A is ranked.

4. Results and Analysis

4.1 Edge Dataset Results

Table 5 summarizes the performance of the considered online data stream classification algorithms on a stream generated based on the Edge dataset, in terms of accuracy, precision, recall, F-score, kappa and the model size. The generated stream consisted of 25,000 samples with 17,333 and 7,667 success and failure instances, respectively. All algorithms were tested using their default set of parameters defined in the scikit-multiflow evaluation platform [33]. In addition, the ensemble-based algorithms were further tested on different sets of base estimators. For instance, the default base estimator for the LB algorithm is the kNN classifier.

However, it was also tested in other configurations, where its base estimator is set to either HT (LB+HT), HAT (LB+HAT), ARF (LB+ARF), EFDT (LB+EFDT), or VFDRC (LB+VFDRC). Similarly, the previous statement applies to all other ensemble-based algorithms. As shown, the considered algorithms exhibited different performance values under the metrics used. In addition, no single algorithm surpasses all other algorithms in terms of all the performance metrics. In this regard, the LB + ARF ensemble algorithm achieved the highest accuracy, precision, F-score, and kappa values, while the NB algorithm outperformed other algorithms in terms of recall and model size metrics. Figure 4 shows a box plot of the obtained performance values for each metric. It visualizes the degree of variation between the algorithms for each performance metric.

In general, the algorithms vary widely in terms of their kappa statistics. In addition, the considered algorithms achieved better precision, recall, and F-score compared with their accuracy values. Apparently, the class imbalance observed in the Edge dataset (i.e., stream) has a direct consequence on the performance of the classification algorithms; only those algorithms capable of handling class imbalance and concept drift achieved

Table 5. Performance comparison on the Edge dataset

Algorithm	Accuracy	Precision	Recall	F-score	Kappa	Model size (kB)
NB	0.6748	0.6763	**0.9857**	0.8022	0.0419	**12**
KNN	0.6875	0.7486	0.8022	0.7745	0.2675	502
kNNADWIN	0.6875	0.7487	0.8021	0.7745	0.2675	513
SAM-KNN	0.7468	0.7788	0.8681	0.8210	0.3927	981137
VFDRC	0.7199	0.7697	0.8296	0.7985	0.3413	81
HT	0.7393	0.7655	0.8798	0.8187	0.3627	145
HAT	0.7964	0.8252	0.8826	0.8529	0.5233	102
EFDT	0.7936	0.8371	0.8511	0.8440	0.5385	361
ARF	0.8991	0.9169	0.9333	0.9250	0.7707	1502
DWMC	0.7794	0.8149	0.8671	0.8402	0.4849	60
LB	0.7241	0.8047	0.762	0.7828	0.4054	4857
SMOTE-B	0.6727	0.8035	0.6604	0.7250	0.3302	60337
UO-B	0.7109	0.7123	0.9374	0.8095	0.2542	3453
OB	0.6872	0.7449	0.7946	0.7689	0.2864	4446
OB-ADWIN	0.6961	0.7542	0.7954	0.7743	0.3106	4440
LB+HT	0.9173	0.9302	0.9519	0.9409	0.8031	1069
LB+HAT	0.9304	0.9415	0.959	0.9502	0.8349	1037
LB+ARF	**0.9380**	**0.9552**	0.9552	**0.9552**	**0.8595**	24580
LB+EFDT	0.9326	0.9469	0.9562	0.9515	0.8409	2063
LB+VFDRC	0.9161	0.9305	0.9494	0.9399	0.8005	756
DWMC+HT	0.8412	0.871	0.9046	0.8875	0.6183	163
DWMC+HAT	0.853	0.884	0.9066	0.8952	0.6495	288
DWMC+ARF	0.9201	0.9393	0.9457	0.9425	0.8117	7581
DWMC+EFDT	0.87	0.8903	0.9264	0.9080	0.6871	444
DWMC+VFDRC	0.8309	0.8789	0.8765	0.8777	0.604	114
SMOTE-B + HT	0.7981	0.8791	0.8212	0.8492	0.5449	56153
SMOTE-B + HAT	0.8249	0.8951	0.8462	0.8700	0.6027	56311
SMOTE-B + ARF	0.8816	0.9247	0.9024	0.9134	0.7262	80722
SMOTE-B + EFDT	0.8565	0.9081	0.882	0.8949	0.6693	56513
SMOTE-B + FDRC	0.7893	0.8609	0.8295	0.8449	0.5166	56147
UO-B + HT	0.7872	0.7727	0.9787	0.8636	0.4018	371
UO-B + HAT	0.8231	0.8876	0.8524	0.8696	0.5949	56278
UO-B + ARF	0.88	0.9234	0.9014	0.9123	0.7227	81812
UO-B + EFDT	0.8118	0.7959	0.9792	0.8781	0.4844	926
UO-B + VFDRC	0.7973	0.7909	0.9612	0.8678	0.4505	418
OB + HT	0.7874	0.8116	0.9022	0.8545	0.4638	1661
OB + HAT	0.8448	0.8579	0.9297	0.8924	0.6158	746
OB + ARF	0.8406	0.8907	0.8774	0.8840	0.6295	11315
OB + EFDT	0.8151	0.8427	0.9011	0.8709	0.5467	3267
OB + VFDRC	0.779	0.7964	0.9145	0.8514	0.4286	872
OB-ADWIN+HT	0.83	0.851	0.9144	0.8816	0.5816	693
OB-ADWIN+HAT	0.8542	0.8676	0.9315	0.8984	0.6415	567

Table 5. Continued

Algorithm	Accuracy	Precision	Recall	F-score	Kappa	Model size (kB)
OB-ADWIN+ARF	0.902	0.9198	0.9404	0.9300	0.7667	15946
OB-ADWIN+EFDT	0.8604	0.8793	0.9254	0.9018	0.6615	816
OB-ADWIN+VFDRC	0.8221	0.8417	0.9151	0.8769	0.5585	556

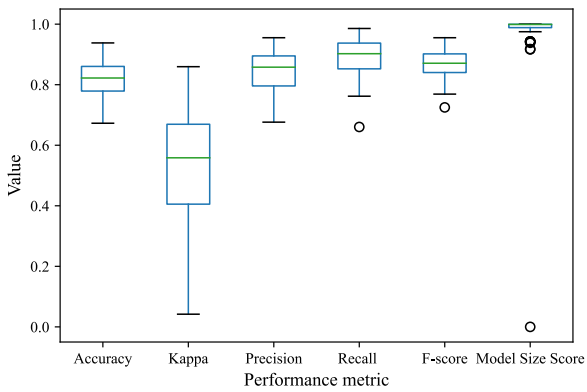

Figure 4. Box plot of performance metrics on the Edge dataset.

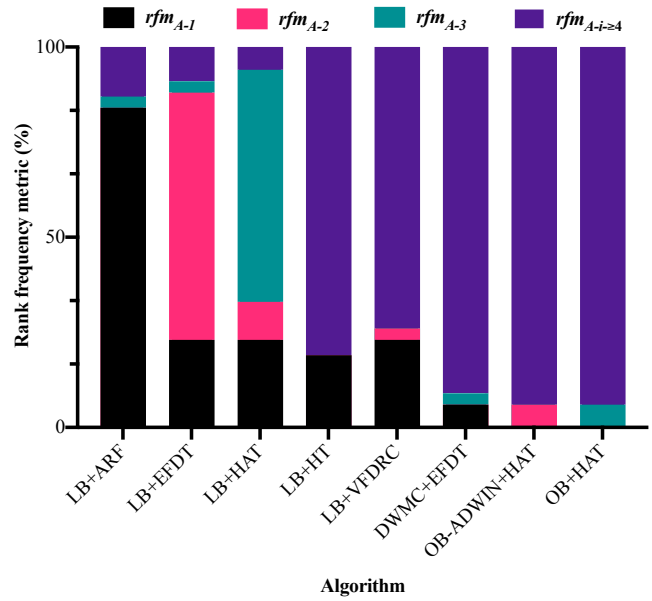

Figure 5. Algorithm ranking for the Edge dataset.

relatively higher performance compared with other algorithms. Specifically, the ARF algorithm and other ensemble algorithms whose base estimator is set to either HT, HAT, ARF, EFDT, or VFDRC have relatively better performance metrics compared with all other algorithms.

In Figure 4, the model size score (MSS) is computed according to Eq. 7.

$$MSS_i = 1 - \frac{MS_i - MS_{min}}{MS_{max} - MS_{min}}, \tag{7}$$

where MSS_i is the score assigned to algorithm i, MS_i is the model size under the ith algorithm, and MS_{min} and MS_{max} are the minimum and maximum model sizes, respectively. The formula given in Eq. 7 normalizes the model size and converts it into an ascending attribute that favors algorithms with lower model sizes. It also puts the model size on the same scale as the other performance metrics. As shown in Figure 4, the vast majority of classification algorithms achieved higher model size scores (i.e., low model sizes as compared with other algorithms). Nevertheless, there are few algorithms that have significantly higher model sizes (i.e., low model size scores).

Because there is no single algorithm that is superior in all performance metrics, but rather a set of algorithms in which some of them are superior in some metrics and inferior in other metrics, it is important to identify the SNDA. Table 6 shows the SNDA for the Edge dataset classification, along with their performance metrics.

As shown in Table 6, this set consists of 25 algorithms with various trade-offs between the performance metrics considered. To determine the most viable options (i.e., algorithms) for the edge data stream, the information shown in Table 6 was fed to the CI-based MCDA process to obtain the rank frequency metric (rfm) for each algorithm in the SNDA of the edge data stream. Figure 5 shows the rfm values associated with various algorithms from the SNDA of the edge data stream. The results are shown for those algorithms that have been ranked at least once in the first three ranks (i.e., algorithms with non-zero values for their rfm_{A-1}, rfm_{A-2}, or rfm_{A-3}). As depicted in Figure 5, the LB+ARF, LB+EFDT, and LB+HAT algorithms have higher values for rfm_{A-1}, rfm_{A-2}, and rfm_{A-3} than do their counterparts. In other words, they always compete for the first three ranks among the various combinations of normalization and aggregation methods. Apparently, the LB+ARF algorithm occupied the first rank for the majority of combinations and could be considered the most promising choice for the edge data stream classification problem. Conversely, the

Table 6. Non-dominated set of algorithms for the Edge dataset

Algorithm	Accuracy	Kappa	Precision	Recall	F-score	Model size score
NB	0.6748	0.0419	0.6763	0.9857	0.8022	1
kNN	0.6875	0.2675	0.7486	0.8022	0.7745	0.9995
kNN-ADWIN	0.6875	0.2675	0.7487	0.8021	0.7745	0.9995
SAMkNN	0.7468	0.3927	0.7788	0.8681	0.821	0
VFDRC	0.7199	0.3413	0.7697	0.8296	0.7985	0.9999
HT	0.7393	0.3627	0.7655	0.8798	0.8187	0.9999
HAT	0.7964	0.5233	0.8252	0.8826	0.8529	0.9999
EFDT	0.7936	0.5385	0.8371	0.8511	0.844	0.9996
ARF	0.8991	0.7707	0.9169	0.9333	0.925	0.9985
DWMC	0.7794	0.4849	0.8149	0.8671	0.8402	1
UO-B	0.7109	0.2542	0.7123	0.9374	0.8095	0.9965
LB+HT	0.9173	0.8031	0.9302	0.9519	0.9409	0.9989
LB+HAT	0.9304	0.8349	0.9415	0.959	0.9502	0.999
LB+ARF	0.938	0.8595	0.9552	0.9552	0.9552	0.975
LB+EFDT	0.9326	0.8409	0.9469	0.9562	0.9515	0.9979
LB+VFDRC	0.9161	0.8005	0.9305	0.9494	0.9399	0.9992
DWMC+HT	0.8412	0.6183	0.871	0.9046	0.8875	0.9998
DWMC+HAT	0.853	0.6495	0.884	0.9066	0.8952	0.9997
DWMC+EFDT	0.87	0.6871	0.8903	0.9264	0.908	0.9996
DWMC+VFDRC	0.8309	0.604	0.8789	0.8765	0.8777	0.9999
UB-O+HT	0.7872	0.4018	0.7727	0.9787	0.8636	0.9996
UB-O+EFDT	0.8118	0.4844	0.7959	0.9792	0.8781	0.9991
UB-O+VFDRC	0.7973	0.4505	0.7909	0.9612	0.8678	0.9996
OB+HAT	0.8448	0.6158	0.8579	0.9297	0.8924	0.9993
OB-ADWIN + HAT	0.8542	0.6415	0.8676	0.9315	0.8984	0.9994

LB+HT, LB+VFDRC, DWMC+EFDT, OB-ADWIN+HAT, and OB-HAT algorithms occupied ranks that were equal to or worse than the fourth rank for a large portion of normalization and aggregation combinations.

4.2 Cloud via RSU Dataset Results

This section presents the performance metrics of the various data stream classification algorithms on a stream generated based on the CloudRSU dataset. The generated stream consists of 25,000 samples with 17,961 and 7,039 success and failure instances, respectively. Table 7 shows the accuracy, precision, recall, F-score, kappa, and model size metrics for the different algorithms. The algorithms obtained different performance values under the considered metrics. Similar to the edge data stream, there is no single algorithm that surpasses other algorithms in terms of all performance metrics. In this regard, the LB+ARF algorithm achieved the highest accuracy, F-score, and kappa values, while the LB-EFDT achieved the highest precision value, while the UB-O and NB algorithms achieved the highest recall and model size values, respectively. Figure 6 shows a box plot of the obtained performance metrics for the various algorithms. The vast majority of algorithms have comparable performance metrics, with the kappa statistic having generally lower values compared with other metrics.

Although the majority of the considered algorithms, especially the ensemble-based ones, have comparable performance metrics, there is no single algorithm that surpasses all other algorithms with respect to the considered metrics. Hence, it is necessary to analyze the dominance relationship between these algorithms and identify a SNDA. Table 8 lists the SNDA for the stream generated from the CloudRSU dataset. This set con-

Table 7. Performance comparison on the CloudRSU dataset

Algorithm	Accuracy	Precision	Recall	F-score	Kappa	Model size (kB)
NB	0.9095	0.9334	0.9409	0.9371	0.7754	**11**
KNN	0.6980	0.7522	0.8634	0.8040	0.1599	463
kNN-ADWIN	0.6952	0.7530	0.8557	0.8011	0.1604	474
SAM-KNN	0.7525	0.7781	0.9162	0.8415	0.2934	980406
VFDRC	0.9087	0.9475	0.9239	0.9356	0.7791	54
HT	0.9267	0.9456	0.9525	0.9490	0.8182	152
HAT	0.9238	0.9347	0.9609	0.9476	0.8080	72
EFDT	0.9298	0.9575	0.9441	0.9508	0.8288	143
ARF	0.9522	0.9631	0.9705	0.9668	0.8814	1572
DWMC	0.9253	0.9418	0.9548	0.9483	0.8137	52
LB	0.7205	0.8062	0.8035	0.8048	0.3127	4467
SMOTE-B	0.6543	0.7811	0.7189	0.7487	0.1961	55935
UO-B	0.7264	0.7325	**0.9745**	0.8363	0.0958	3129
OB	0.6952	0.7505	0.8616	0.8022	0.1519	4456
OB-ADWIN	0.7145	0.7658	0.8671	0.8133	0.2164	3844
LB+HT	0.9467	0.9611	0.9660	0.9635	0.8702	1179
LB+HAT	0.9540	0.9690	0.9669	0.9679	0.8868	430
LB+ARF	**0.9614**	0.9726	0.9737	**0.9731**	**0.9049**	18477
LB+EFDT	0.9599	**0.9743**	0.9697	0.9720	0.9015	1467
LB+VFDRC	0.9488	0.9650	0.9636	0.9643	0.8739	448
DWMC+HT	0.9349	0.9498	0.9610	0.9554	0.8378	110
DWMC+HAT	0.9380	0.9498	0.9646	0.9571	0.8453	234
DWMC+ARF	0.9561	0.9674	0.9716	0.9695	0.8915	5115
DWMC+EFDT	0.9416	0.9552	0.9638	0.9595	0.8549	167
DWMC+VFDRC	0.9186	0.9548	0.9305	0.9425	0.8031	135
SMOTE-B+HT	0.9282	0.9458	0.9546	0.9502	0.8218	52406
SMOTE-B+HAT	0.9251	0.9433	0.9528	0.9480	0.8139	52585
SMOTE-B+ARF	0.9517	0.9665	0.9662	0.9663	0.8810	69819
SMOTE-B+EFDT	0.9392	0.9539	0.9618	0.9578	0.8492	52699
SMOTE-B+VFDRC	0.9151	0.9510	0.9295	0.9401	0.7941	52373
UO-B + HT	0.9269	0.9325	0.9681	0.9500	0.8142	388
UO-B + HAT	0.9288	0.9349	0.9680	0.9512	0.8194	554
UO-B + ARF	0.9446	0.9498	0.9742	0.9618	0.8607	15765
UO-B + EFDT	0.9320	0.9392	0.9679	0.9533	0.8282	517
UO-B + VFDRC	0.9257	0.9378	0.9600	0.9488	0.8133	322
OB + HT	0.9262	0.9348	0.9601	0.9473	0.8184	1322
OB + HAT	0.9252	0.9362	0.9612	0.9485	0.8118	647
OB + ARF	0.9613	0.9613	0.9700	0.9656	0.8770	17768
OB + EFDT	0.9416	0.9568	0.9619	0.9593	0.8553	1514
OB + VFDRC	0.9182	0.9478	0.9376	0.9427	0.7998	827
OB-ADWIN+HT	0.9247	0.9366	0.9600	0.9482	0.8107	399

Table 7. Continued

Algorithm	Accuracy	Precision	Recall	F-score	Kappa	Model size (kB)
OB-ADWIN+HAT	0.9246	0.9355	0.9611	0.9481	0.8100	648
OB-ADWIN+ARF	0.9516	0.9633	0.9695	0.9664	0.8802	19609
OB-ADWIN+EFDT	0.9374	0.9515	0.9618	0.9566	0.8444	645
OB-ADWIN+VFDRC	0.9241	0.9471	0.9471	0.9471	0.8128	312

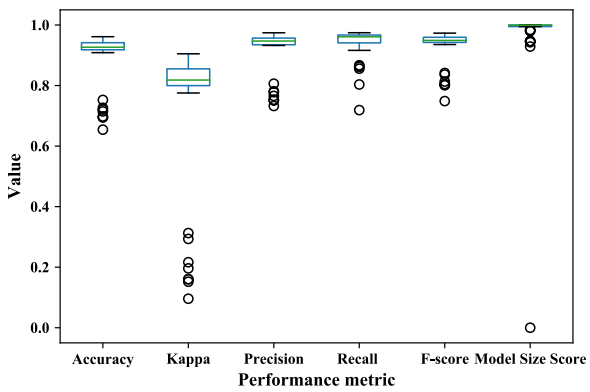

Figure 6. Box plot of performance metrics on the CloudRSU dataset.

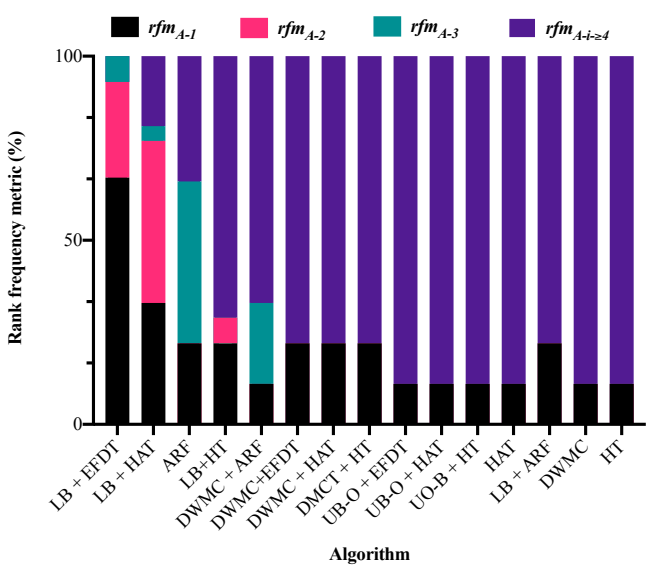

Figure 7. Algorithms ranking for the CloudRSU dataset.

tains 22 algorithms. This set was then used to obtain the rank performance metrics, as shown in Figure 7. The results are for those algorithms that have occupied one of the first three ranks at least once under all possible combinations of normalization and aggregation methods.

The results shown in Figure 7 illustrate that the LB+EFDT, LB+HAT, and ARF algorithms dominate other algorithms in terms of the number of times they occupy one of the top three ranks. Evidently, the LB+EFDT algorithm ranks robustly within the first three ranks, with a high share of combinations assigning it to the first rank. Hence, it can be considered a suitable data stream classification algorithm for streams generated from the CloudRSU dataset.

4.3 Cloud via CN Dataset Results

This section presents the results obtained for the various algorithms on the data stream generated from the CloudCN dataset. Similar to the other two cases, the generated stream consisted of 25,000 instances. The number of success and failure instances in this stream were 16,188 and 8,812, respectively. Table 9 summarizes the performance metrics obtained using the studied algorithms. As shown, the algorithms vary in terms of their obtained performance values with no single algorithm dominating the others in all metrics. The LB+ARF, LB+EFDT, and NB algorithms achieved the highest accuracy, precision, recall, F-score, kappa, and model size, respectively.

Figure 8 shows the variability observed among the algorithms with respect to all performance metrics. As depicted, a large portion of the considered algorithms have very similar performance values. In addition, the values obtained under the kappa statistic are lower than those of the other metrics. The results of

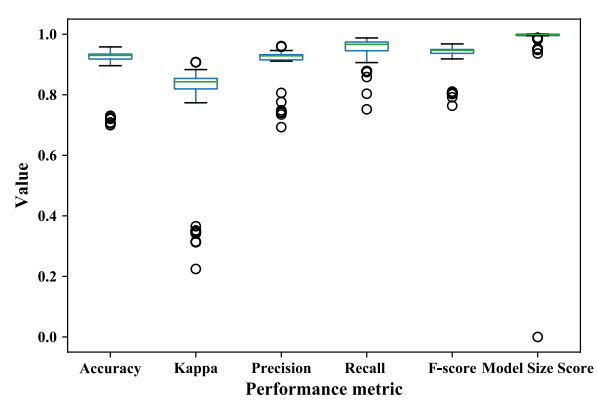

Figure 8. Box plot of performance metrics on the CloudCN dataset.

Table 8. Non-dominated set of algorithms for the CloudRSU dataset

Algorithm	Accuracy	Kappa	Precision	Recall	F-score	Model size score
NB	0.9095	0.7754	0.9334	0.9409	0.9371	1
VFDRC	0.9087	0.7791	0.9475	0.9239	0.9356	0.999956283
HT	0.9267	0.8182	0.9456	0.9525	0.9490	0.999855844
HAT	0.9238	0.808	0.9347	0.9609	0.9476	0.999937923
EFDT	0.9298	0.8288	0.9575	0.9441	0.9508	0.999865371
ARF	0.9522	0.8814	0.9631	0.9705	0.9668	0.998408233
DWMC	0.9253	0.8137	0.9418	0.9548	0.9483	0.999957844
UO-B	0.7264	0.0958	0.7325	0.9745	0.8363	0.996819955
LB + HT	0.9467	0.8702	0.9611	0.966	0.9635	0.998808409
LB + HAT	0.954	0.8868	0.969	0.9669	0.9679	0.999572642
LB + ARF	0.9614	0.9049	0.9726	0.9737	0.9731	0.981164191
LB + EFDT	0.9599	0.9015	0.9743	0.9697	0.9720	0.998514415
DMCT + HT	0.9349	0.8378	0.9498	0.961	0.9554	0.999899143
DWMC + HAT	0.938	0.8453	0.9498	0.9646	0.9571	0.999772316
DWMC + ARF	0.9561	0.8915	0.9674	0.9716	0.9695	0.994794179
DWMC + EFDT	0.9416	0.8549	0.9552	0.9638	0.9595	0.999840483
DWMC + VFDRC	0.9186	0.8031	0.9548	0.9305	0.9425	0.999873969
UO-B + HT	0.9269	0.8142	0.9325	0.9681	0.9500	0.999615492
UB-O + HAT	0.9288	0.8194	0.9349	0.968	0.9512	0.999446182
UB-O + ARF	0.9446	0.8607	0.9498	0.9742	0.9618	0.983930994
UB-O + EFDT	0.932	0.8282	0.9392	0.9679	0.9533	0.999483728
OB + ARF	0.9613	0.877	0.9613	0.97	0.9656	0.981887726

the algorithm dominance are shown in Table 10. As illustrated, the SNDA for the considered stream consists of 23 algorithms with various trade-offs between performance metrics.

Figure 9 shows the ranking results obtained by feeding the information given in Table 10 to the CI-based MCDA procedure. It considers only algorithms with non-zero values of rfm_{A-1}, rfm_{A-2}, or rfm_{A-3}. The results depicted in Figure 9 show that the LB+VFDRC, LB+HT, and LB+EFDT surpass all other algorithms with respect to their rank frequency metric. While the LB+EFDT ensemble-based algorithm is assigned to the first rank for a larger number of combinations as compared to the LB+VFDRC, the latter is never assigned to a rank worse than third under all combinations of normalization and aggregation. In addition, the LB+HT and LB+EFDT algorithms occupied ranking positions worse than or equal to the fourth rank for a noticeable portion of scoring and ranking combinations. Moreover, algorithms other than the best three algorithms were allocated the fourth or worse ranks for a relatively large number of combinations, hindering their potential applicability to the considered online data stream classification task.

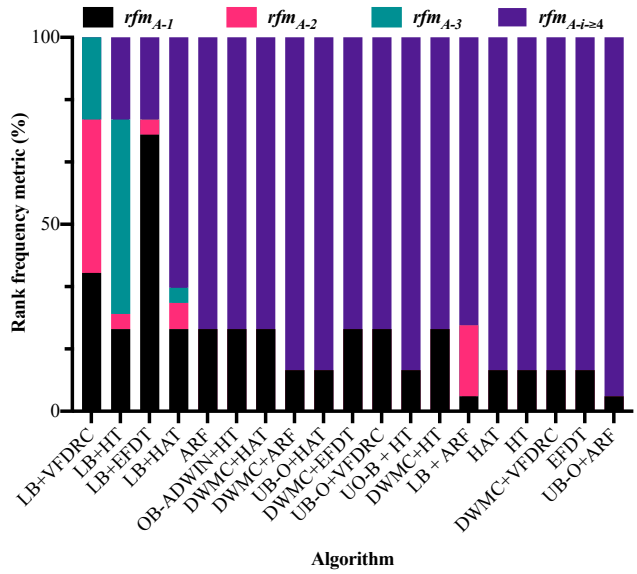

Figure 9. Algorithm ranking for the CloudCN dataset.

In summary, the results shown in Tables 6, 8, and 10 illustrate

Table 9. Performance comparison on the CloudCN dataset

Algorithm	Accuracy	Precision	Recall	F-score	Kappa	Model size (kB)
NB	0.8969	0.9156	0.9258	0.9207	0.7736	**11**
KNN	0.7236	0.7415	0.8784	0.8042	0.3456	445
kNN-ADWIN	0.7214	0.7408	0.8749	0.8023	0.3415	452
SAMKNN	0.7084	0.7349	0.8585	0.7919	0.3149	979425
VFDRC	0.8961	0.9310	0.9063	0.9185	0.7752	102
HT	0.9194	0.9176	0.9616	0.9391	0.8202	115
HAT	0.9224	0.9254	0.9570	0.9409	0.8279	86
EFDT	0.9203	0.9270	0.9515	0.9391	0.8238	156
ARF	0.9420	0.9381	0.9746	0.9560	0.8712	880
DWMC	0.9017	0.9111	0.9396	0.9251	0.7823	52
LB	0.7205	0.8062	0.8035	0.8048	0.3127	4467
SMOTE-B	0.6996	0.7761	0.7519	0.7638	0.3515	50811
UO-B	0.7051	0.6931	0.9755	0.8104	0.2246	4045
OB	0.7252	0.7434	0.8775	0.8049	0.3507	4465
OB-ADWIN	0.7306	0.7486	0.8777	0.8080	0.3656	3921
LB+HT	0.9462	0.9434	0.9753	0.9591	0.8807	711
LB+HAT	0.9455	0.9444	0.9730	0.9585	0.8793	704
LB+ARF	**0.9581**	0.9586	0.9774	**0.9679**	0.9077	11654
LB+EFDT	0.9581	**0.9609**	0.9749	0.9678	**0.9078**	2081
LB+VFDRC	0.9472	0.9462	0.9737	0.9598	0.8832	500
DWMC+HT	0.9288	0.9270	0.9659	0.9461	0.8416	198
DWMC+HAT	0.9330	0.9323	0.9666	0.9491	0.8513	360
DWMC+ARF	0.9449	0.9388	0.9786	0.9583	0.8775	4565
DWMC+EFDT	0.9323	0.9328	0.9648	0.9485	0.8499	417
DWMC+VFDRC	0.9232	0.9362	0.9455	0.9408	0.8313	161
SMOTE-B + HT	0.9263	0.9270	0.9617	0.9440	0.8363	47269
SMOTE-B +HAT	0.9336	0.9341	0.9653	0.9494	0.8528	47389
SMOTE-B +ARF	0.9402	0.9286	0.9709	0.9493	0.8673	62317
SMOTE-B +EFDT	0.9336	0.9324	0.9674	0.9496	0.8525	48022
SMOTE-B +FDRC	0.9273	0.9296	0.9603	0.9447	0.8389	47313
UO-B + HT	0.9269	0.9113	0.9824	0.9455	0.8348	281
UO-B + HAT	0.9308	0.9158	0.9833	0.9484	0.8439	656
UO-B + ARF	0.9306	0.9122	**0.9877**	0.9484	0.8429	16995
UO-B + EFDT	0.9300	0.9157	0.9821	0.9477	0.8422	668
UO-B + VFDRC	0.9301	0.9281	0.9667	0.9470	0.8445	346
OB + HT	0.9237	0.9153	0.9718	0.9427	0.8287	1138
OB + HAT	0.9329	0.9303	0.9688	0.9492	0.8508	753
OB + ARF	0.9375	0.9319	0.9745	0.9527	0.8606	14264
OB + FDT	0.9330	0.9301	0.9692	0.9492	0.8509	1980
OB + VFDRC	0.9179	0.9305	0.9433	0.9369	0.8194	823
OB-ADWIN+HT	0.9326	0.9267	0.9726	0.9491	0.8495	351

Table 9. Continued

Algorithm	Accuracy	Precision	Recall	F-score	Kappa	Model size (kB)
OB-ADWIN+HAT	0.9344	0.9309	0.9705	0.9503	0.8540	703
OB-ADWIN+ARF	0.9383	0.9331	0.9743	0.9533	0.8625	12298
OB-ADWIN+EFDT	0.9345	0.9321	0.9693	0.9503	0.8545	689
OB-ADWIN+VFDRC	0.9291	0.9324	0.9598	0.9459	0.8431	452

Table 10. Non-dominated set of algorithms for the CloudCN dataset

Algorithm	Accuracy	Kappa	Precision	Recall	F-score	Model size score
NB	0.8969	0.7736	0.9156	0.9258	0.9207	1
VFDRC	0.8961	0.7752	0.931	0.9063	0.9185	0.999906689
HT	0.9194	0.8202	0.9176	0.9616	0.9391	0.999893732
HAT	0.9224	0.8279	0.9254	0.957	0.9409	0.999923127
EFDT	0.9203	0.8238	0.927	0.9515	0.9391	0.999851544
ARF	0.942	0.8712	0.9381	0.9746	0.9560	0.999113041
DWMC	0.9017	0.7823	0.9111	0.9396	0.9251	0.999957801
UB-O	0.7051	0.2246	0.6931	0.9755	0.8104	0.99588126
LB+HT	0.9462	0.8807	0.9434	0.9753	0.9591	0.999284837
LB+HAT	0.9455	0.8793	0.9444	0.973	0.9585	0.999292781
LB+ARF	0.9581	0.9077	0.9586	0.9774	0.9679	0.988112142
LB+EFDT	0.9581	0.9078	0.9609	0.9749	0.9678	0.997886041
LB+VFDRC	0.9472	0.8832	0.9462	0.9737	0.9598	0.999501079
DWMC+HT	0.9288	0.8416	0.927	0.9659	0.9461	0.999809508
DWMC+HAT	0.933	0.8513	0.9323	0.9666	0.9491	0.999644093
DWMC+ARF	0.9449	0.8775	0.9388	0.9786	0.9583	0.995349983
DWMC+EFDT	0.9323	0.8499	0.9328	0.9648	0.9485	0.999585037
DWMC+VFDRC	0.9232	0.8313	0.9362	0.9455	0.9408	0.999847256
UB-O+HT	0.9269	0.8348	0.9113	0.9824	0.9455	0.999723845
UB-O+HAT	0.9308	0.8439	0.9158	0.9833	0.9484	0.999341514
UB-O+ARF	0.9306	0.8429	0.9122	0.9877	0.9484	0.982658624
UB-O+VFDRC	0.9301	0.8445	0.9281	0.9667	0.9470	0.999657887
OB-ADWIN+HT	0.9326	0.8495	0.9267	0.9726	0.9490	0.999652466

the fact that for each classification task, there is a set of non-dominated algorithms, with comparable performance metrics that can be utilized in real-life applications. The ranking results in Figures 5, 7, and 9 provide an insight into the algorithms that are superior to other members in the corresponding non-dominated set, considering the trade-offs observed under all performance metrics. Although the results presented in Sections 4.1, 4.2, and 4.3 highlight the most suitable algorithms for each classification task, it is worth noting that the LB+EFDT (i.e., leveraging bagging algorithm, whose base estimator is set to the extremely fast decision tree) demonstrates consistent competence to occupy one of the top-ranking positions for each of the considered datasets because of its competitive performance metrics, in addition to its reasonably acceptable model size. This algorithm maintains an ensemble of n EFDT base classifiers, where n is set to 10 by default in the utilized evaluation platform [33]. To classify an incoming instance, every individual classifier will produce a prediction (i.e., a vote), and the final classification result is obtained by aggregating the individual predictions. Following Condorcet's jury theorem [54–56], there is theoretical proof that the error rate of an ensemble approaches 0, in the limit, provided that two conditions are satisfied. First,

individual base classifiers must perform better than random guessing. Second, individual classification models must exhibit diversity, that is, they should not produce correlated errors. In the case of the LB+EFDT algorithm, the EFDT algorithm is used as the base classification model. The EFDT is a refinement over the HT algorithm. Whereas the HT algorithm delays the selection of a split at a node until it is confident that it has determined the best split, the EFDT algorithm selects and deploys a split as soon as it is confident that the split is useful. In addition, the HT algorithm never revisits its splitting decision, while the EFDT algorithm revisits its splitting decision, replacing the split if it later becomes evident that a better split is available. Hence, the EFDT algorithm can learn rapidly from incoming instances with an inbuilt tolerance against concept drift, utilizing the most advantageous splitting decision recognized to date [41]. Consequently, the EFDT algorithm is considered a viable base classifier that meets the first condition implied by Condorcet's jury theorem.

The LB algorithm employs online bagging to train its base models (i.e., classifiers). In this regard, as each incoming classification instance is observed, online re-sampling is performed by presenting that instance to each model $k \sim$ Poisson (λ) times and updating each model accordingly. The value of k is regarded as the weight of the incoming instance. To increase online re-sampling, the LB algorithm uses a higher value of the λ parameter of the Poisson distribution, which is set to six by default. With this value of λ, the LB algorithm causes more randomness in the weights of the incoming instances. Hence, it achieves a higher input space diversity by assigning a different range of weights to each incoming instance. In addition, the LB algorithm leverages the bagging performance by adding randomization at the output of the ensemble using output codes. As shown in Section 2.5, the LB algorithm assigns to each possible class label a binary string of length n, where n is the number of base classifiers in the ensemble. Each base classifier learns one bit in a binary string. Unlike standard ensemble methods, the LB algorithm utilizes random output codes instead of deterministic ones. In other words, while the base classifiers in the standard methods predict the same function, using output codes allows each classifier in the LB ensemble to predict a different function [44]. This would minimize the influence of correlations between the base classifiers and, in turn, increase the diversity of the ensemble [57,58]. Hence, the introduction of randomization to the input and output of the ensemble classifiers yields, to some extent, a diversified ensemble satisfying the second condition imposed by Condorcet's jury theorem. Furthermore, the LB algorithm uses the ADWIN algorithm to deal with concept drift, with one ADWIN instance for each classifier in the ensemble. Each time a concept drift is detected, the worst classifier is reset. Hence, the LB algorithm always monitors the quality of its online learning process and adheres to the current class distribution of incoming classification instances. In general, the inherent diversity among its base classifiers compensates for the classification errors induced by any individual classifier. This can be clearly observed in Table 5, where a single EFDT classifier achieved an accuracy of 0.7936, and the accuracy of the LB+EFDT ensemble reached 0.9326. Altogether, the LB+EFDT algorithm demonstrated the ability to maintain a noticeable performance under all considered datasets and performance metrics. Hence, it could be considered a feasible choice for online data stream classification in real-life VEC environments.

5. Conclusion

VEC has recently become an integral part of intelligent transportation systems. In these systems, workload orchestration plays a crucial role in selecting the most appropriate computational node to execute tasks generated from vehicular applications. The continuous streams of tasks generated from vehicular applications pose significant challenges in data management, execution, and storage. Online ML algorithms can address continuous data streams and manage large data sizes. Hence, this paper has addressed the potential application of online data stream classification algorithms to support workload orchestration in VEC environments. These algorithms can be used to predict the ability of the available computational nodes to successfully handle a particular task. In this work, various online data stream classification algorithms were evaluated based on synthetic datasets generated from simulated VEC environments. In addition, a MCDA technique was employed to rank various algorithms based on the obtained performance metrics. The evaluation results show that the considered algorithms can handle online classification operations with noticeable trade-offs and dominance relations with respect to their obtained performance. In addition, the employed multi-criteria decision analysis technique demonstrated the ability to rank different algorithms and identify the most appropriate algorithms to handle online classification in VEC environments.

Future work will consider implementing a workload orchestration algorithm based on the findings of this study to show how online data stream classification can enhance orchestration

performance in dynamically changing VEC environments.

Conflict of Interest

No potential conflict of interest relevant to this article was reported.

References

[1] H. Guo, J. Liu, J. Ren, and Y. Zhang, "Intelligent task offloading in vehicular edge computing networks," *IEEE Wireless Communications*, vol. 27, no. 4, pp. 126-132, 2020. https://doi.org/10.1109/MWC.001.1900489

[2] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443-58469, 2020. https://doi.org/10.1109/ACCESS.2020.2983149

[3] J. Neil, L. Cosart, and G. Zampetti, "Precise timing for vehicle navigation in the smart city: an overview," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 54-59, 2020. https://doi.org/10.1109/MCOM.001.1900596

[4] D. Sabella, D. Brevi, E. Bonetto, A. Ranjan, A. Manzalini, and D. Salerno, "MEC-based infotainment services for smart roads in 5G environments," in *Proceedings of 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, Antwerp, Belgium, 2020, pp. 1-6. https://doi.org/10.1109/VTC2020-Spring48590.2020.9128807

[5] R. F. Atallah, C. M. Assi, and M. J. Khabbaz, "Scheduling the operation of a connected vehicular network using deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 5, pp. 1669-1682, 2019. https://doi.org/10.1109/TITS.2018.2832219

[6] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," in *Proceedings of 2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, 2018, pp. 1-7. https://doi.org/10.1109/ICC.2018.8422661

[7] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: a survey," *Mobile Networks and Applications*, 2020. https://doi.org/10.1007/s11036-020-01624-1

[8] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," in *Proceedings of 2017 IEEE International Conference on Communications (ICC)*, Paris, France, 2017, pp. 1-6. https://doi.org/10.1109/ICC.2017.7997360

[9] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697-1716, 2019. https://doi.org/10.1109/JPROC.2019.2915983

[10] Y. F. Payalan and M. A. Guvensan, "Towards nextgeneration vehicles featuring the vehicle intelligence," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 1, pp. 30-47, 2020. https://doi.org/10.1109/TITS.2019.2917866

[11] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 769-782, 2019. https://doi.org/10.1109/TNSM.2019.2901346

[12] Y. Wang, S. Wang, S. Zhang, and H. Cen, "An edge-assisted data distribution method for vehicular network services," *IEEE Access*, vol. 7, pp. 147713-147720, 2019. https://doi.org/10.1109/ACCESS.2019.2946484

[13] C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, "Machine learning-based workload orchestrator for vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2239-2251, 2021. https://doi.org/10.1109/TITS.2020.3024233

[14] J. S. Rojas, A. Pekar, A. Rendon, and J. C. Corrales, "Smart user consumption profiling: Incremental learning-based OTT service degradation," *IEEE Access*, vol. 8, pp. 207426-207442, 2020. https://doi.org/10.1109/ACCESS.2020.3037971

[15] D. Nallaperuma, R. Nawaratne, T. Bandaragoda, A. Adikari, S. Nguyen, T. Kempitiya, D. De Silva, D. Alahakoon, and D. Pothuhera, "Online incremental machine learning platform for big data-driven smart traffic management," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4679-4690, 2019. https://doi.org/10.1109/TITS.2019.2924883

[16] U. Adhikari, T. H. Morris, and S. Pan, "Applying Hoeffding adaptive trees for real-time cyber-power event and intrusion classification," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4049-4060, 2018. https://doi.org/10.1109/TSG.2017.2647778

[17] X. Li, Y. Zhou, Z. Jin, P. Yu, and S. Zhou, "A classification and novel class detection algorithm for concept drift data stream based on the cohesiveness and separation index of Mahalanobis distance," *Journal of Electrical and Computer Engineering*, vol. 2020, article. 4027423, 2020. https://doi.org/10.1155/2020/4027423

[18] L. Rutkowski, M. Jaworski, and P. Duda, "Basic concepts of data stream mining," in *Stream Data Mining: Algorithms and Their Probabilistic Properties*. Cham, Switzerland: Springer, 2020, pp. 13-33. https://doi.org/10.1007/978-3-030-13962-9_2

[19] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *Proceedings of the 24th European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016.

[20] Q. Yang, Y. Gu, and D. Wu, "Survey of incremental learning," in *Proceedings of 2019 Chinese Control And Decision Conference (CCDC)*, Nanchang, China, 2019, pp. 399-404. https://doi.org/10.1109/CCDC.2019.8832774

[21] K. K. Wankhade, S. S. Dongre, and K. C. Jondhale, "Data stream classification: a review," *Iran Journal of Computer Science*, vol. 3, pp. 239-260, 2020. https://doi.org/10.1007/s42044-020-00061-3

[22] Z. El Mrabet, D. F. Selvaraj, and P. Ranganathan, "Adaptive Hoeffding tree with transfer learning for streaming synchrophasor data sets," in *Proceedings of 2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, 2019, pp. 5697-5704. https://doi.org/10.1109/BigData47090.2019.9005720

[23] C. Nixon, M. Sedky, and M. Hassan, "Practical application of machine learning based online intrusion detection to internet of things networks," in *Proceedings of 2019 IEEE Global Conference on Internet of Things (GCIoT)*, Dubai, UAE, 2019, pp. 1-5. https://doi.org/10.1109/GCIoT47977.2019.9058410

[24] V. G. T. Da Costa, E. J. Santana, J. F. Lopes, and S. Barbon, "Evaluating the four-way performance trade-off for stream classification," in *Green, Pervasive, and Cloud Computing*. Cham, Switzerland: Springer, 2019, pp. 3-17. https://doi.org/10.1007/978-3-030-19223-5_1

[25] J. F. Lopes, E. J. Santana, V. G. T. da Costa, B. B. Zarpelao, and S. B. Junior, "Evaluating the four-way performance trade-off for data stream classification in edge computing," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 1013-1025, 2020. https://doi.org/10.1109/TNSM.2020.2983921

[26] A. Bifet, R. Gavalda, G. Holmes, and B. Pfahringer, *Machine Learning for Data Streams: With Practical Examples in MOA*. Cambridge, MA: MIT Press, 2018.

[27] S. Greco, J. Figueira, and M. Ehrgott, *Multiple Criteria Decision Analysis*. New York, NY: Springer, 2016.

[28] M. Cinelli, M. Kadzinski, M. Gonzalez, and R. Slowinski, "How to support the application of multiple criteria decision analysis? Let us start with a comprehensive taxonomy," *Omega*, vol. 96, article no. 102261, 2020. https://doi.org/10.1016/j.omega.2020.102261

[29] M. Cinelli, M. Spada, W. Kim, Y. Zhang, and P. Burgherr, "MCDA Index Tool: an interactive software to develop indices and rankings," *Environment Systems and Decisions*, vol. 41, no. 1, pp. 82-109, 2021. https://doi.org/10.1007/s10669-020-09784-x

[30] L. Diaz-Balteiro, J. Gonzalez-Pachon, and C. Romero, "Measuring systems sustainability with multi-criteria methods: a critical review," *European Journal of Operational Research*, vol. 258, no. 2, pp. 607-616, 2017. https://doi.org/10.1016/j.ejor.2016.08.075

[31] S. El Gibari, T. Gomez, and F. Ruiz, "Building composite indicators using multicriteria methods: a review," *Journal of Business Economics*, vol. 89, no. 1, pp. 1-24, 2019. https://doi.org/10.1007/s11573-018-0902-z

[32] S. Greco, A. Ishizaka, M. Tasiou, and G. Torrisi, "On the methodological framework of composite indices: a review of the issues of weighting, aggregation, and robustness," *Social Indicators Research*, vol. 141, no. 1, pp. 61-94, 2019. https://doi.org/10.1007/s11205-017-1832-9

[33] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: a multi-output streaming framework," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2915-2914, 2018.

[34] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964-994, 2016. https://doi.org/10.1007/s10618-015-0448-4

[35] J. Demsar and Z. Bosnic, "Detecting concept drift in data streams using model explanation," *Expert Systems with Applications*, vol. 92, pp. 546-559, 2018. https://doi.org/10.1016/j.eswa.2017.10.003

[36] K. Forster, S. Monteleone, A. Calatroni, D. Roggen, and G. Troster, "Incremental kNN classifier exploiting correct-error teacher for activity recognition," in *Proceedings of 2010 9th International Conference on Machine Learning and Applications*, Washington, DC, 2010, pp. 445-450. https://doi.org/10.1109/ICMLA.2010.72

[37] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, Minneapolis, MN, 2007, pp. 443-448. https://doi.org/10.1137/1.9781611972771.42

[38] V. Losing, B. Hammer, and H. Wersing, "KNN classifier with self adjusting memory for heterogeneous concept drift," in *Proceedings of 2016 IEEE 16th International Conference on Data Mining (ICDM)*, Barcelona, Spain, 2016, pp. 291-300. https://doi.org/10.1109/ICDM.2016.0040

[39] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, 2001, pp. 97-106. https://doi.org/10.1145/502512.502529

[40] A. Bifet and R. Gavalda, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis*. Heidelberg, Germany: Springer, 2009, pp. 249-260. https://doi.org/10.1007/978-3-642-03915-7_22

[41] C. Manapragada, G. I. Webb, and M. Salehi, "Extremely fast decision tree," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, London, UK, 2018, pp. 1953-1962. https://doi.org/10.1145/3219819.3220005

[42] P. Kosina and J. Gama, "Very fast decision rules for classification in data streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 168-202, 2015. https://doi.org/10.1007/s10618-013-0340-z

[43] N. C. Oza, "Online bagging and boosting," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Waikoloa, HI, 2001, pp. 2340-2345. https://doi.org/10.1109/ICSMC.2005.1571498

[44] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Machine Learning and Knowledge Discovery in Databases*. Heidelberg, Germany: Springer, 2010, pp. 135-150. https://doi.org/10.1007/978-3-642-15880-3_15

[45] B. Wang and J. Pineau, "Online bagging and boosting for imbalanced data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no.12, pp. 3353-3366, 2016. https://doi.org/10.1109/TKDE.2016.2609424

[46] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9, pp. 1469-1495, 2017. https://doi.org/10.1007/s10994-017-5642-8

[47] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: an ensemble method for drifting concepts," *he Journal of Machine Learning Research*, vol. 8, pp. 2755-2790, 2007.

[48] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *Proceedings of 2009 IEEE Symposium on Computational Intelligence and Data Mining*, Nashville, TN, 2009, pp. 324-331. https://doi.org/10.1109/CIDM.2009.4938667

[49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002. https://doi.org/10.1613/jair.953

[50] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: an environment for performance evaluation of edge computing systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, article no. e3493, 2018. https://doi.org/10.1002/ett.3493

[51] T. Vasiloudis, F. Beligianni, and G. De Francisci Morales, "BoostVHT: boosting distributed streaming decision trees,"

in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, Singapore, 2017, pp. 899-908. https://doi.org/10.1145/3132847.3132974

[52] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Sydney, Australia, 2015, pp. 59-68. https://doi.org/10.1145/2783258.2783372

[53] H. Ishibuchi, H. Masuda, and Y. Nojima, "Selecting a small number of non-dominated solutions to be presented to the decision maker," in *Proceedings of 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, San Diego, CA, 2014, pp. 3816-3821. https://doi.org/10.1109/SMC.2014.6974525

[54] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993-1001, 1990. https://doi.org/10.1109/34.58871

[55] K. K. Ladha, "Condorcet's jury theorem in light of de Finetti's theorem," *Social Choice and Welfare*, vol. 10, no. 1, pp. 69-85, 1993. https://doi.org/10.1007/BF00187434

[56] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren, "The online performance estimation framework: heterogeneous ensemble learning for data streams," *Machine Learning*, vol. 107, no. 1, pp. 149-176, 2018. https://doi.org/10.1007/s10994-017-5686-9

[57] Y. Lv, S. Peng, Y. Yuan, C. Wang, P. Yin, J. Liu, and C. Wang, "A classifier using online bagging ensemble method for big data stream learning," *Tsinghua Science and Technology*, vol. 24, no. 4, pp. 379-388, 2019. https://doi.org/10.26599/TST.2018.9010119

[58] M. Kolarik, M. Sarnovsky, and J. Paralic, "Diversity in ensemble model for classification of data streams with concept drift," in *Proceedings of 2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Herl'any, Slovakia, 2021, pp. 000355-000360. https://doi.org/10.1109/SAMI50585.2021.9378625

Mutaz Al-Tarawneh is an associate professor in computer engineering at Mutah University, Jordan. He obtained his Ph.D. in Computer Engineering from Southern Illinois University Carbondale, USA, in 2010. His current research interests include cloud computing, machine learning, and IoT systems.
E-mail: mutaz.altarawneh@mutah.edu.jo