Streaming multi-layer ensemble selection using dynamic genetic algorithm.

LUONG, A.V., NGUYEN, T.T. and LIEW, A.W.-C.

2021

Streaming Multi-layer Ensemble Selection using Dynamic Genetic Algorithm

Anh Vu Luong
Griffith University
Gold Coast, Australia
vu.luong@griffithuni.edu.au

Tien Thanh Nguyen
Robert Gordon University
Aberdeen, Scotland, UK
t.nguyen11@rgu.ac.uk

Alan Wee-Chung Liew
Griffith University
Gold Coast, Australia
a.liew@griffith.edu.au

***Abstract*—In this study, we introduce a novel framework for non-stationary data stream classification problems by modifying the Genetic Algorithm to search for the optimal configuration of a streaming multi-layer ensemble. We aim to connect the two sub-fields of non-stationary stream classification and evolutionary dynamic optimization. First, we present Streaming Multi-layer Ensemble (SMiLE) - a novel classification algorithm for non-stationary data streams which comprises multiple layers of different classifiers. Second, we develop an ensemble selection method to obtain an optimal subset of classifiers for each layer of SMiLE. We formulate the selection process as a dynamic optimization problem and then solve it by adapting the Genetic Algorithm to the stream setting, generating a new classification framework called SMiLE_GA. Finally, we apply the proposed framework to address a real-world problem of insect stream classification, which relates to the automatic recognition of insects through optical sensors in real-time. The experiments showed that the proposed method achieves better prediction accuracy than several state-of-the-art benchmark algorithms for non-stationary data stream classification.**

***Index Terms*—Ensemble Method, Multi-layer Ensemble, Genetic Algorithm**

I. Introduction

In the era of big data, machine learning is becoming increasingly popular for analyzing complex data to save the cost and time of performing manual tasks. However, when dealing with real-world big data, traditional machine learning algorithms suffer from three major drawbacks: storing the whole dataset is infeasible; models fail to handle very high-speed data; changes in data distribution make models collapsed (*concept drift* [1]). Data can even be generated as a real-time stream in many applications including sensor networks, video streaming, and traffic monitor systems, which demands machine learning models to be updated continuously and rapidly. Naturally, data streams are potentially non-stationary because the process generating them may become different over time, leading to the concept drift issue. In particular, prediction models can get stuck in the concept of old data and never adapts readily to the new distribution. In such scenarios, online learning with an associated concept drift handling mechanism is one of the best schemes to adapt to distribution changes in data streams while maintaining good prediction performance. [2]–[4]

The field of optimization plays a crucial role in almost all machine learning algorithms. For example, Deep Neural Network (DNN), one of the most successful machine learning models, needs an optimization algorithm to search for its optimal weights. Gradient-based optimization methods like Stochastic Gradient Descent, Adam [5] are well-suited for optimizing DNN due to the feasibility to differentiate the loss function with respect to its weights. However, these optimization methods are not applicable for more complicated scenarios, for example when the loss function is not differentiable.

The dynamic nature of many real-world problems can affect their objective functions and constraints, corrupting the behaviors of traditional optimization methods. In the literature, optimization problems with their components changing over time are called *Dynamic Optimization Problems* (*DOPs*). Solving DOPs is particularly difficult due to the requirement to track changing optimal solution(s) over time. For complex problems like DOPs, Evolutionary Computation-based methods are an effective choice since their behaviors are inspired by biological evolution and self-organized populations operating in continuously changing environments.

In this paper, we propose a novel streaming classification framework by introducing a DOP solver that works in the data stream setting, which connects the two sub-fields and opens a new research direction for the machine learning community. Our contributions in this work are summarized as follows:

1) Streaming Multi-layer Ensemble: We introduce a cascade structure to combine different online learning algorithms into a multi-layer ensemble, which is able to learn incrementally from non-stationary data streams.
2) Ensemble selection for SMiLE: We propose a mechanism to make the Genetic Algorithm applicable to solve the SMiLE selection problem in a non-stationary stream setting.
3) Real-world application: We apply the proposed framework to address the insect stream classification problem. The goal is to recognize insects related to public health problems. The data streams in this problem was generated by using an optical sensor over time [6].
4) Experimental analysis: We compare the proposed methods with several state-of-the-art algorithms on the insect streaming data. The experiments show that the proposed method achieve higher prediction accuracy than the benchmark algorithms.

In the next section, we have some discussions on the background and related work (Section II), followed by the proposed methods (Section III), experimental setting (Section IV), and result and discussion (Section V). Finally, we draw some conclusions in Section VI.

II. Background and Related Work

A. Data stream learning

In the data stream setting (or online setting), learning models are expected to start making predictions at any time before obtaining the whole dataset since the stream of data may never end. Furthermore, they need to be incremental and fast due to the high-speed characteristic of data streams. Here we discuss two types of algorithms for data stream classification: single classifiers and ensemble systems.

Single classifiers

Some batch learning methods are naturally incremental and fast, making them directly applicable to classify streaming data. The most noticeable method with a low computational cost is the well-known Naïve Bayes (NB) classifier. It performs instance-incremental prediction by making a naive assumption that all feature variables are mutually independent conditional on each class. However, this simple assumption is also the drawback of the NB method since it is generally invalid in many real-world scenarios. Other methods that can perform online learning by instinct are Perceptron and Stochastic Gradient Descent (SGD). Perceptron tries to linearly separate the data into different classes, while SGD is an incremental gradient-based optimization method for differentiable objective functions, especially convex loss functions such as log loss or hinge loss. Both SGD and Perceptron are very fast and cost-efficient, but they can only handle simple datasets for instance those with the linear separability property.

Another way to produce online classifiers is to 'streamify' batch learning algorithms. Decision Tree attracts the most attention in the literature due to its capability to retain high performance and theoretical support when porting to the stream environments. It is also a good base learner for many streaming ensembles with state-of-the-art prediction accuracy [2], [7], [8]. Very Fast Decision Tree (VFDT) [9] - also known as Hoeffding Tree - was the first successful adaption of Decision Tree to the data stream setting. To determine the best split attribute when building a tree, VFDT tries not to revisit old instances by waiting for new ones to arrive. An interesting characteristic of VFDT is that it asymptotically converges to a batch learning Decision Tree when having enough data. Hulton et al. introduced Concept-adapting Very Fast Decision Tree (CVFDT) [10] as an upgraded version of VFDT for non-stationary data streams. There are also many other variants of the Hoeffding Tree model in the literature, for example Extremely Fast Decision Tree (EFDT) [11], Random Hoeffding Tree (RHT) [12], Hoeffding Option Tree (HOT) [13], and Hoeffding Adaptive Tree (HAT) [14]. They all use the Hoeffding bound to check the condition for splits at each node.

Ensemble systems

Almost all the best-performing models for non-stationary data streams in terms of prediction accuracy are ensemble-based methods mainly because they can selectively exploit the advantages of various single classifiers at once. The most well-known ensemble-based system for data streams is the Online Bagging method, which was introduced by Oza [15]. It adapted the classical Bagging algorithm to the stream setting by employing the Poisson(1) distribution to simulate the bootstrap technique in an online manner. The author also proposed Online Boosting in his work, but it is less popular than Online Bagging due to the slower speed and the lower prediction accuracy. There are better variants of Boosting for non-stationary data streams, such as the Boosting-like Online Learning Ensemble (BOLE) [3] and the Online Smooth Boost (OSB) [16]. BOLE improved the performance of Online Boosting by weakening the condition for an expert to vote and making use of the Drift Detection Method (DDM) [17] to handle changes in data. In the OSB method, the definition of the online weak learner was redefined, and only smooth distributions were generated to avoid assigning too much weight to a single ensemble member (also known as ensemble expert). Recently, van Rijn et al. proposed the BLAST ensemble [18] which made use of the Online Performance Estimation framework to adaptively select a subset of best-performing base classifiers to form the voting panel. BLAST works well in practice when having a diverse set of different base classifiers. Another approach to handle streaming data is to use chunk-based ensembles. A well-known ensemble in this category is the Learn++.NSE [19], which generalized the Learn++ method [20] for non-stationary environments. Learn++.NSE exploits the ensemble error on a new data chunk to assign weights to the instances. Very recently, Montiel et al. introduced the Adaptive XGBoost (AXGB) ensemble system [4], a replica of the classical eXtreme Gradient Boosting (XGB) [21], for the stream setting. In this method, new ensemble members are generated from mini-batches of incoming instances. The learning process continues even when a fixed maximum size of the ensemble is reached thanks to the fact that the ensemble keeps updating to be adaptive to the latest concept of data.

B. Evolutionary computing algorithms for Dynamic Optimization Problems

Most optimization solvers in the literature were designed for problems with static fitness functions and constraints. However, in reality, these static assumptions may be invalid due to the dynamic characteristics of the environments where the problem is set up. In these cases, the objective functions and constraints can vary over time, making static optimization algorithms collapsed or even inapplicable. This dynamic context can be found everywhere in today's real-world applications including Social Networks, IoT Devices, Smart Homes, and Smart Traffic Monitoring Systems. Evolutionary Computing (EC) and Swarm Intelligence (SI) methods are

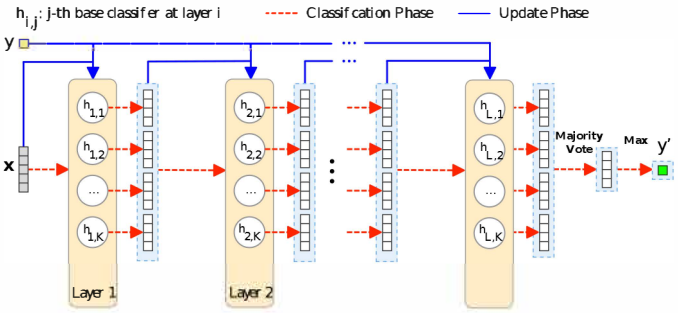

Fig. 1. Streaming Multi-layer Ensemble

especially useful for solving DOPs since they are developed based on real-life biological evolution and naturally self-organized populations which are always subjected to non-stationary phenomena. There are two main approaches on how to design EC and SI algorithms for DOPs regarding the way to generate population diversity: Active and Passive.

Active approach

In the active approach, algorithms possess a mechanism to explicitly detect changes in the problem formulation. When a change is detected, the algorithm alters its original search behaviors to adapt to that change. Note that the performance of this approach highly depends on the efficiency of the associated change detector. The most notable work following the active approach is the introduction of hyper-mutation operator [22], which triggers an increase in the rate of mutation when a change occurs. Another successful work involves actively diversifying the pool of candidates by migrating individuals inside a subpopulation in a multi-population paradigm [23], [24]. It appears that the active approach faces the challenges of determining how much diversity needs to be magnified when a change is detected, making it difficult to solve the problem of data stream classification in which none or very little information is revealed at the beginning of the stream.

Passive approach

On the other hand, population diversity is continuously maintained over time during the search procedure in the passive approach. In detail, diversity in population/swarm is promoted by sacrificing search performance to prevent the optimization process to converge too quickly to the optimum and, therefore, avoid being stuck there when the solution becomes out of date. There are many proposed ways to accomplish this idea in the literature. For example, Simões and Costa introduce the transformation operator, which was inspired by the somatic hypermutation of B-cells [25]. When an individual performs the transformation operator, one gene segment is first randomly chosen from a random gene pool. Then, the selected segment substituted the gene located after a random transformation locus. In [26], robust optimization over time (ROOT) was proposed as a new approach to solve DOPs. This framework uses an adapted radial-basis-function to locally approximate the fitness, and an auto-aggressive model is employed to predict it. This method then searches for robust solutions by exploiting the information of local fitness approximation and prediction. Recently, Yazdani et al. followed the idea of ROOT to propose a multi-swarm Particle Swarm Optimization (PSO) algorithm for DOPs [27]. This method allows different swarms to track peaks and collect data about their search behaviors, which is then analyzed to determine the next robust solution. The average number of environments is maximized while the quality of solutions is kept acceptable. The passive approach can be easily applied to real-world problems such as stream classification since its performance is comparable to the active approach while there is no need to employ a change detector.

III. Proposed Method

A. Problem formulation

A data stream is defined as an infinite sequence of data points $X = \{x_1, x_2, ..., x_\infty\}$, in which x_k is a d-dimensional feature vector, with an associated sequence of class labels $Y = \{y_1, y_2, ..., y_\infty\}$, where $y_k \in \{l_1, l_2, ..., l_M\}$ is the true label of the sample x_k in X. A common assumption in the data stream literature is that the true label y_k of x_k is obtainable before the next data point x_{k+1} comes up.

Generally, there are two main approaches to process data streams: (1) use a single data instance $\{x_k, y_k\}$ at a time to update the classifier; (2) divide the incoming stream into equally sized chunks $\boldsymbol{C}_1, \boldsymbol{C}_2, ..., \boldsymbol{C}_\infty$ and then use all instances of a chunk to update the classifier.

A data stream is *stationary* if all its instances (excluding the outliers) are generated from the same distribution D. By contrast, a *non-stationary stream* includes concept drift [1] over time, or in other words, the underlying data distribution may change over time. Many types of concept drift are introduced in the literature, most notably *Abrupt Drift*, *Gradual Drift*, and *Incremental Drift*. If an abrupt drift occurs at a moment, the current data distribution is immediately substituted by a new distribution, which often severely damages the classification performance if the learning model fails to react in a timely manner. Meanwhile, gradual and incremental drifts happen over a longer period of time, making them more challenging to detect. The types of concept drift in evolving data streams are very similar to the types of changes in dynamic optimization problems [28]. Therefore, it is very natural to employ a DOPs solver to address the problem of non-stationary data stream classification.

B. Streaming Multi-layer Ensemble (SMiLE)

Inspired by the cascade structure of Multi-layer Perceptron (or Neural Networks), we proposed the Streaming Multi-layer Ensemble (SMiLE) for data stream classification. The main idea is to use the layer-by-layer processing of the features to perform representation learning. In particular, the output of a layer is considered as the input data for the next layer [29]. The proposed method is illustrated in Fig. 1.

Each layer of SMiLE is a heterogeneous ensemble containing various types of online classifiers. Table I shows the classifier list used in each layer. We choose this list of online classifiers based on their prediction accuracy and speed. Since

TABLE I
CLASSIFIER LIST AT EACH LAYER

Classifier	Type
Naïve Bayes (NB)	Bayesian
Perceptron (Perc)	Linear Model
SGD (hinge loss)	SVM
Hoeffding Tree (HT)	Decision Tree
Random Hoeffding Tree (RHT)	Random Tree
Hoeffding Option Tree (HOT)	Option Tree
Hoeffding Adaptive Tree (HAT)	Decision Tree

all the classifiers in the list are fast and incremental, each layer can be updated on the fly with a single instance. Also, all the layers are able to make predictions at any time.

When a true label y of a data point x is available, we can update the SMiLE as follows. First, x plays the role of the input vector for layer 1. In other words, K classifiers in layer 1 predict on x, giving K output probability vectors with the length of M (the number of classes). These vectors are then concatenated into a K×M-dimensional vector v_1, which is considered as the input vector for layer 2. Before considering layer 2, the instance $\{x, y\}$ is used to train each classifier in layer 1. Similarly to what we have done in layer 1, classifiers in layer 2 predict on v_1, yielding another $K \times M$-dimensional output vector before being updated using the instance $\{v_1, y\}$. The process continues until all classifiers of all layers of the SMiLE are updated by using a single instance $\{x, y\}$.

The classification phase of SMiLE is slightly simpler than the update process discussed above. Each layer consecutively makes predictions for the $K \times M$-dimensional output vector obtained from the previous layer, except for layer 1 which predicts for the raw feature vector x. The last layer L gives us K different M-dimensional output prediction vectors. The final prediction of SMiLE is achieved by aggregating these probability vectors using an ensemble combining method such as the Sum Rule or the Majority Vote Rule [30].

C. Ensemble selection for SMiLE

We introduce a novel selection method to improve the proposed SMiLE. It is inspired by the mechanism of the Drop Out method [31], a must-mentioning technique when talking about the success of Deep Learning [32]. The main idea is to drop a subset of neurons at each layer to reduce the complexity of the whole Neural Networks (Multi-layer Ensembles), avoiding the overfitting issue in many cases. Here, we selectively choose which classifiers to be used in each layer by formulating this process as a DOP and then solve it using the Genetic Algorithm. Our method is different from the original Drop Out method where a number of neurons are blindly dropped at random.

To formulate the selection process, we first introduce the concept of a SMiLE configuration. We employ a binary vector to represent the selection decision at the layer i: $S_i = [s_{i,1} \quad s_{i,2} \ldots \quad s_{i,K}]^T, s_{i,j} \in \{0, 1\}$, where $s_{i,j} = 1$ means the j-th classifier of layer i is selected, and $s_{i,j} = 0$ otherwise. Therefore, a selection configuration for SMiLE can be represented by a LK-length vector:

$$\boldsymbol{v} = [\underbrace{s_{1,1}s_{1,2}\ldots s_{1,K}}_{\text{layer 1}} \underbrace{s_{2,1}s_{2,2}\ldots s_{2,K}}_{\text{layer 2}} \ldots \underbrace{s_{L,1}s_{L,2}\ldots s_{L,K}}_{\text{layer L}}]$$

Each configuration now corresponds to a specific multi-layer ensemble which is simpler than the original one. We call this simpler multi-layer ensemble as a *refined ensemble*. For example, with $K = 3$, $L = 4$, $\boldsymbol{v} = [0\,0\,1 \;\; 1\,1\,0 \;\; 1\,1\,1 \;\; 1\,0\,1]$, number of classes $M = 4$, number of features $d = 5$, classifier list = Naïve Bayes (NB), Perceptron (Perc), Stochastic Gradient Descent (SGD), then the refined ensemble is shown in Fig. 2.

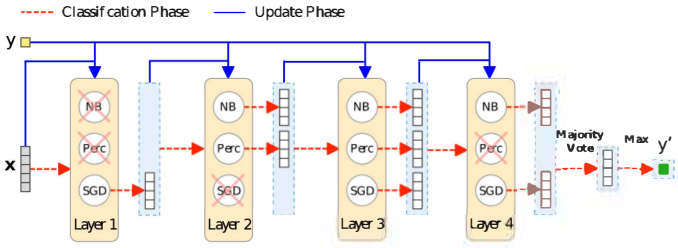

Fig. 2. An example of refined ensemble

The classification phase and update phase of a refined ensemble follow the same procedures of the original SMiLE which has been thoroughly discussed in sub-section III-B.

The fitness associated with each configuration is evaluated as follows. First, we use the chunk-based approach to store a chunk of N latest instances of the data stream. This chunk then will be used to calculate the fitness as follows:

- The first $N/2$ instances are used solely for updating the refined ensemble
- For the remaining $N/2$ instances, we use the interleaved-test-then-train method to evaluate the ensemble. Particularly, the ensemble makes prediction for an incoming data point to obtain a predicted label which is compared with the real label to compute the 0-1 loss. After that, the data point along with its true label is used for updating the ensemble. This is a very common technique to evaluate classifiers in data stream [2], [4], [33].

Lets consider a chunk of N instances $\boldsymbol{C}_i = [\{x_k, y_k\}, ..., \{x_{k+N-1}, y_{k+N-1}\}]$, and a configuration $\boldsymbol{v}$ with its corresponding refined ensemble E. Let $\{y'_{k+N/2}, ..., y'_{k+N-1}\}$ be the predictions of the ensemble E for $\{x_{k+N/2}, ..., x_{k+N-1}\}$. We take the accuracy of E on the last $N/2$ instances of $\boldsymbol{C}_i$ as the fitness score for the corresponding configuration:

$$fitness(\boldsymbol{v}) = \mathcal{L}_{0,1}(E) = \frac{1}{N/2} \sum_{i=k+N/2}^{k+N-1} \mathbb{I}[y'_i = y_i] \quad (1)$$

where $\mathbb{I}[.]$ is the indicator function which returns 1 if the condition is true and 0 otherwise. Note that the fitness function

Algorithm 1 Update phase of SMiLE_GA

Input: a data stream $S = C_1, C_2, ..., C_\infty$ (C_i is i-th chunk of data)

1: Initialize the population and the segment pool of GA
2: **for** each chunk C_k **do**
3: $iter = 0$
4: **while** $iter < max\_iter$ **do**
5: Evaluate population using equation (1)
6: Use Roulette Wheel to select 2 individuals using probabilities obtained from equation (2)
7: Transform selected individuals with probability of P_t
8: Mutate selected individuals with probability of P_m
9: $iter = iter + 1$
10: **end while**
11: Update the segment pool
12: Choose the refined ensemble corresponding to the best candidate to classify C_{k+1}
13: **end for**

changes over time due to the non-stationary characteristic of evolving data streams. Hence, we need a DOP solver instead of a static one to search for the best configuration of SMiLE. In particular, we used the Genetic Algorithm (GA) with the transformation operator introduced in [25], which is in charge of injecting diversity into the population. The proposed method SMiLE_GA is detailed in Algorithm 1.

When a new chunk of data is available, we update the population max_iter times. In each iteration, we first recalculate the fitness of all candidates in the current population (line 5). This is done by using equation (1) for every candidate after updating them using the first half instances of the chunk. Next, two different individuals are selected from the population using the Roulette Wheel Selection technique with probabilities:

$$p_i = \frac{fitness_i}{\sum_{j=1}^{nPop} fitness_j} \quad (2)$$

where p_i and $fitness_i$ are the selection probability and fitness score of the i-th candidate, respectively, and $nPop$ is the size of the population used in GA (line 6). The two candidates are transformed with a probability of P_t (line 7). The transformation operator is as follows. First, a random segment is selected from the pool. Then, a position in the configuration is chosen at random. After that, the selected segment substitutes the genes located right after the chosen position (see Fig. 3) [25]. In the next stage, the two selected chromosomes are mutated with a probability of P_m (line 8). After evolving the population, we then update the segment pool as follows: 70% of the segments are taken from the individuals of the current population, while the remaining 30% are newly generated at random (line 11). The segment sizes are also generated randomly. At the final step, the ensemble corresponding to the highest-fitness chromosome is chosen to predict for instances in the next data chunk (line 12).

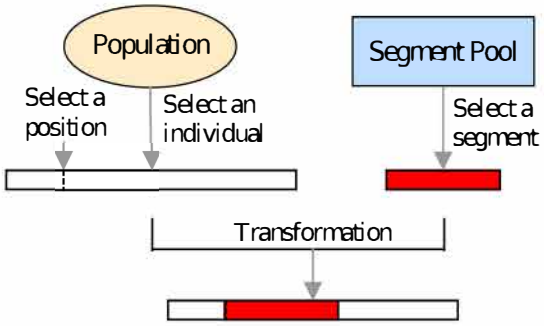

Fig. 3. The transformation operator

TABLE II
INSECT DATASETS USED IN THE EXPERIMENTS

Data	#instances	#classes	#features	Change point(s)
Abrupt	52,848	6	33	14,352; 19,500; 33,240; 38,682; 39,510
Incremental-gradual	24,150	6	33	14,028
Incremental	57,018	6	33	Throughout all the stream
Incremental-abrupt-reoccurring	79,986	6	33	26,568; 53,364
Incremental-reoccurring	79,986	6	33	26,568; 53,364
Out-of-control	905,145	24	33	Throughout all the stream

IV. EXPERIMENT SETTING

A. Datasets

We applied the proposed framework to solve the problem of insect stream classification. We used six datasets recently published by Souza et al. [6], which relates to the automatic recognition of disease-carrying insects through optical sensors in real-time. To collect data, a smart trap was used to capture selective species, especially those that are vectors of mosquito-borne diseases and agricultural pests. This trap frees all other species, alleviating the negative influence of the device on the ecological balance. Before the announcement of this data, the data stream literature was completely lacking knowledge of when and how data distributions change in real-world data streams. Souza et al. collected data in an artificial non-stationary environment for about three months to construct the insect stream datasets with concept drifts. Note that the true class labels were obtained by building different collector devices that many specimens of only one species was exhibited inside the collector. Many types of concept drift in data streams can be introduced by changing the temperature as follows:

- Abrupt: The first period of the stream was collected at a temperature of 30°C, then it suddenly changed to 20°C. After a period of time, the temperature increased back to around 35°C. Three other abrupt drifts similarly appeared until the stream ended.
- Incremental-gradual: The beginning instances were collected at around 37°C, and the temperature gradually decreased to 35°C. For a period after that, the temperature intercalates in the values of 35°C and 23°C until completely change to 23°C. Finally, the temperature gradually increased to 27°C.

TABLE III
ACCURACY RESULTS OF SMILE, SMILE_GA AND BASE CLASSIFIERS

Dataset	SMiLE_GA	SMiLE	NB	Perc	SGD	HT	RHT	HOP	HAT
Abrupt	**70.8882 (1)**	64.3033 (3)	50.7342 (7)	67.7888 (2)	16.6667 (9)	53.8317 (6)	49.3377 (8)	55.9491 (5)	61.5028 (4)
Incremental-gradual	**70.8033 (1)**	60.1656 (6)	53.6273 (7)	70.1946 (2)	16.6708 (9)	60.6087 (4.5)	46.3354 (8)	60.6087 (4.5)	61.7019 (3)
Incremental	59.0480 (2)	53.7567 (4)	47.4201 (7)	**59.3444 (1)**	16.6667 (9)	52.1572 (5.5)	43.9914 (8)	52.1572 (5.5)	54.0058 (3)
Incremental-abrupt-reoccurring	71.5950 (2)	69.1296 (3)	58.5390 (6)	**72.9315 (1)**	16.7229 (9)	57.7326 (7)	51.0527 (8)	59.6292 (5)	64.0750 (4)
Incremental-reoccurring	**74.4630 (1)**	70.1910 (2)	48.7560 (7)	68.3370 (3)	16.7254 (9)	53.3818 (6)	46.8107 (8)	54.7083 (5)	66.6517 (4)
Out-of-control	**67.0174 (1)**	64.0208 (3)	45.9915 (8)	66.7527 (2)	14.1356 (9)	56.2187 (4)	54.5515 (6)	56.1770 (5)	49.2031 (7)
Average of accuracy	**68.9692**	63.5945	50.8447	67.5582	16.2647	55.6551	48.6799	56.5383	59.5234
Average of ranking	**1.3333**	3.5000	7.0000	1.8333	9.0000	5.5000	7.6667	5.0000	4.1667

- Incremental: the stream of instances was collected while incrementally increasing the temperature from 20°C to 40°C.
- Incremental-abrupt-reoccurring: There were three consecutive cycles of incremental changes of temperature from 20°C to 24°C.
- Incremental-reoccurring: There were three consecutive cycles of incremental changes where the temperature first gradually increased from 20°C to 40°C, then slowly decreased from 40°C to 20°C, and finally rose incrementally from 20°C back to 40°C.
- Out-of-control: there was no pattern in the changes of the temperature. This dataset is drift-free since all instances were collected in uniformly random order and each example was sampled uniformly at a time during the stream.

The details of the datasets are summarized in Table II.

B. Benchmark algorithms and parameters

The benchmark algorithms used in our experiments were the Online Bagging (OB) [15], Online Smooth Boost (OSB) [16], Adaptive XGBoost (AXGB) [4], and Learn++.NSE (LNSE) [19]. Note that OB and OSB are two instance-incremental ensemble systems, while AXGB and LNSE are two chunk-incremental ensemble systems. Since AXGB was developed only for binary classification problems, we wrapped it with a one-vs-all classifier when dealing with multi-class data streams. We compared these algorithms with our proposed methods SMiLE and SMiLE_GA.

Regarding the hyper-parameters, we used default values reported in the original papers if not stated here. The ensemble size of the benchmark algorithms was set to 30. The numbers of layers of SMiLE and SMiLE_GA were set to 4 when comparing to other benchmark algorithms. Each layer contained 7 different learning algorithms. We used the Majority Vote Rule for combing the outputs of the last layer. The chunk size was set to 500. The parameter for the GA module is as follows: the transformation probability was set to $P_t = 0.75$; the mutation rate was set to $P_m = 0.05$; the population size and the segment pool size were both set to 30.

We compared the proposed methods to the benchmark algorithms concerning prediction accuracy, which is one of the most common metrics used in data stream evaluation [2], [3], [8], [33]. To evaluate the prediction performance, we used the interleaved-test-then-train strategy (also known as the prequential evaluation method), in which an instance is first used for testing and then for training.

V. RESULT AND DISCUSSION

A. Proposed methods vs. baselines

The accuracy results of SMiLE, SMiLE_GA, and 7 base learners on 6 datasets are reported in Table III. Overall, the proposed method SMiLE_GA achieved the best prediction accuracy results. It ranks first on 4 datasets and ranks second on the remaining 2 datasets, demonstrating the considerable improvement of the proposed method in comparison to its base classifiers. We can see that the SGD performed extremely badly, which obtained only about 16% of accuracy on all datasets. This deteriorates the performance of SMiLE, making its accuracy even worse than the single learner Perceptron. Fortunately, by using the ensemble selection module, SMiLE_GA can refuse to use SGD whenever this classifier exhibits harmful impacts to the performance of the whole framework. In comparison to SMiLE, the upgraded version SMILE_GA performs better on all datasets, especially on the Incremental-gradual and Abrupt datasets, where there are huge accuracy gaps of around 10% and 6% respectively between SMiLE_GA and SMiLE. This observation demonstrates the benefit of using the ensemble selection module.

B. Proposed methods vs. benchmark algorithms

Table IV shows the accuracy results of SMiLE, SMiLE_GA, and 4 benchmark methods. For all datasets, we can see that the proposed method SMiLE_GA achieved the best overall results with an average accuracy of 68.9692% and an average ranking of 1.1667, followed by its counterpart SMiLE method. The SMiLE_GA framework ranked first on 5/6 datasets and ranked second on the remaining dataset, demonstrating the effectiveness of our method in comparison to available streaming learning algorithms in the literature. Especially on the dataset with abrupt concept drifts, the SMiLE method left a large gap of 6.5% accuracy compared to the second-best method. By contrast, the worst-performing method was the OSB with an average accuracy of 57.5764% and an average ranking of 4.6667. This poor performance can be attributed to the fact that the OSB method did not consider a strategy to deal with changes in the distribution of data. Meanwhile, AXGB, LNSE, and OB obtained average performance in our experiments. From Table IV, we can see a general view of the prediction performance of the algorithms. However, when

TABLE IV
ACCURACY RESULTS OF SMILE, SMILE_GA AND BENCHMARK ALGORITHMS

Dataset	SMiLE_GA	SMiLE	AXGB	OSB	LNSE	OB
Abrupt	**70.8882 (1.0)**	64.3033 (2.0)	58.8100 (4.0)	55.8299 (6.0)	61.5104 (3.0)	57.5008 (5.0)
Incremental-gradual	**70.8033 (1.0)**	60.1656 (5.0)	56.8500 (6.0)	60.4017 (4.0)	61.6398 (3.0)	62.2774 (2.0)
Incremental	59.0480 (2.0)	53.7567 (6.0)	**61.7500 (1.0)**	55.6544 (4.0)	56.2314 (3.0)	54.6985 (5.0)
Incremental-abrupt-reoccurring	**71.5950 (1.0)**	69.1296 (2.0)	57.8900 (5.0)	60.1105 (4.0)	55.5822 (6.0)	60.9494 (3.0)
Incremental-reoccurring	**74.4630 (1.0)**	70.1910 (2.0)	60.3300 (4.0)	55.1697 (5.0)	60.6994 (3.0)	55.1672 (6.0)
Out-of-control	**67.0174 (1.0)**	64.0208 (3.0)	66.8400 (2.0)	58.2923 (5.0)	52.1297 (6.0)	59.3976 (4.0)
Average of accuracy	**68.9692**	63.5945	60.4117	57.5764	57.9655	58.3318
Average of ranking	**1.1667**	3.3333	3.6667	4.6667	4.0000	4.1667

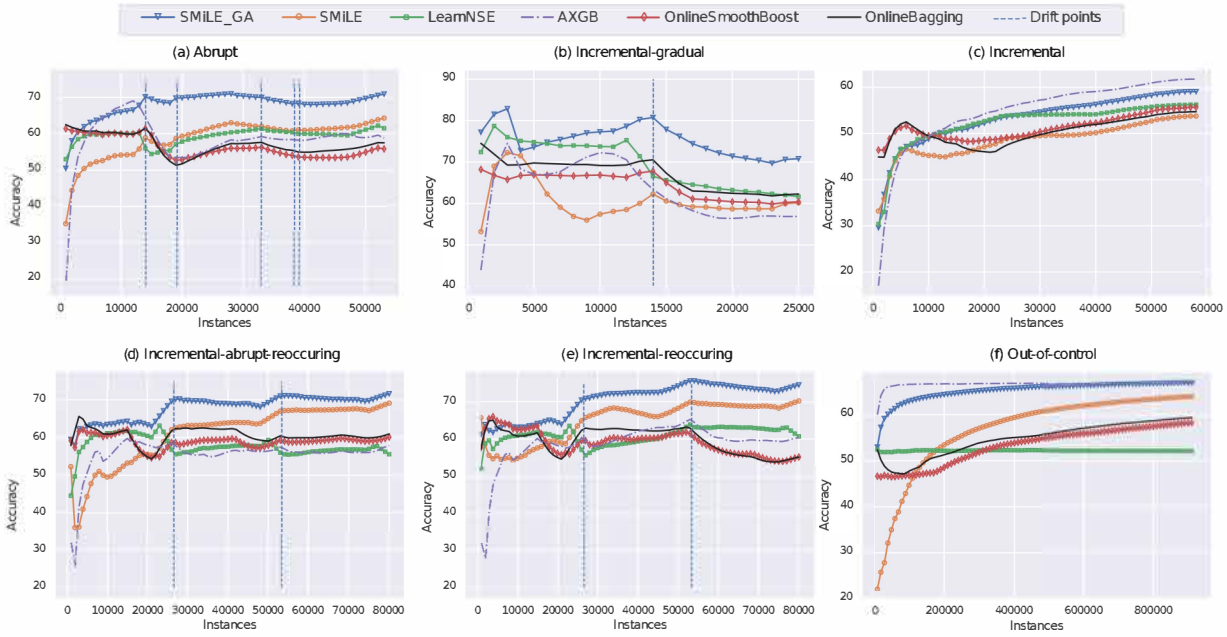

Fig. 4. Accuracy results over time

dealing with data streams, we are often interested in observing these performances over incoming instances, which helps us to better understand how changes happen during the stream. Therefore, we present Fig. 4 to show an individual evaluation for each dataset.

Fig. 4a shows the accuracy results over time for the Abrupt data. The accuracies of all algorithms tended to decrease when concept drift occurred, except for the second one. OB, AXGB, and OSB endured the most significant drop in performance when changes occurred. By contrast, the performance of our proposed method SMiLE_GA was very stable during the stream. It recovered the prediction accuracy very quickly every time a drift happens, leading to its best accuracy at the end of the stream.

Fig. 4b illustrates the results over time for the Incremental-gradual data stream. In this figure, there were significant decreases in the performances of all methods right after the gradual drift occurred. As in the description of this dataset, there were two different concepts presented in this period, making it difficult for all algorithms to detect and react to this change. The proposed method SMiLE continues to perform best almost all over the stream.

Fig. 4c shows the accuracy results over time for the Incremental dataset. Due to the slow rate of the incremental changes, all algorithms tended to perform stably with a gradual increase in the prediction accuracy. In this dataset, AXGB exhibited the highest accuracy after 10000 instances arrived until the end of the stream.

Fig. 4d illustrates the accuracy results for the Incremental-abrupt-reoccurring dataset. There were small decreases in the accuracy of all methods whenever a change occurred, except for the case of SMiLE, whose performance kept increasing even when concept drifts happened. However, the SMiLE_GA method obtained higher accuracy than its counterpart SMiLE even with its decline in accuracy when changes appeared.

Fig. 4e shows the result results for the Incremental-reoccurring dataset. The overall trend was very similar to what had previously been shown in Fig. 4d for the Incremental-abrupt-reoccurring dataset, except for that in the current figure, the falls in the prediction accuracies of all methods were more significant after the final change occurred, especially for OSB and OB methods, those did not have a module to deal with concept drifts.

Fig. 4f shows the results for the Out-of-control data.

Even though this dataset presents undefined changes in type and number, all methods obtained stable performances over time with gradual increases in the accumulated accuracies. SMiLE_GA and AXGB were the best-performing methods in this dataset. Although SMiLE_GA obtains lower accuracy at the first period of the stream, its accuracy increased more quickly and surpassed the accuracy of AXGB at the end of the stream. Learn++.NSE seemed to perform poorly on this data, showing a minor downward trend in its accuracy during the data stream.

VI. Conclusions

In this work, we have proposed the Streaming Multi-layer Ensemble (SMiLE) for data stream classification. It arranges its base classifiers in a cascade structure and performs layer-by-layer processing of the feature vector before utilizing an ensemble combining method to form the final prediction. Furthermore, we introduced a novel data stream classification framework called SMiLE_GA, which modifies the Genetic Algorithm to solve the dynamic optimization problem representing the process of ensemble selection for multiple layer ensemble. We applied the proposed methods to solve a real-world problem of insect stream classification. Our experiments showed that the SMiLE_GA greatly improved the prediction accuracy of its base classifiers and surpassed the performance of many benchmark algorithms.

References

[1] 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.

[2] 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.

[3] R. S. M. de Barros, S. G. T. de Carvalho Santos, and P. M. G. Júnior, "A boosting-like online learning ensemble," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 1871–1878.

[4] J. Montiel, R. Mitchell, E. Frank, B. Pfahringer, T. Abdessalem, and A. Bifet, "Adaptive xgboost for evolving data streams," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[6] V. M. Souza, D. M. dos Reis, A. G. Maletzke, and G. E. Batista, "Challenges in benchmarking stream learning algorithms with real-world data," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1805–1858, 2020.

[7] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2010, pp. 135–150.

[8] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 240–249.

[9] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

[10] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 97–106.

[11] 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*, 2018, pp. 1953–1962.

[12] A. Bifet, E. Frank, G. Holmes, and B. Pfahringer, "Ensembles of restricted hoeffding trees," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 3, no. 2, pp. 1–20, 2012.

[13] B. Pfahringer, G. Holmes, and R. Kirkby, "New options for hoeffding trees," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2007, pp. 90–99.

[14] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 249–260.

[15] N. C. Oza, "Online bagging and boosting," in *2005 IEEE international conference on systems, man and cybernetics*, vol. 3. Ieee, 2005, pp. 2340–2345.

[16] S.-T. Chen, H.-T. Lin, and C.-J. Lu, "An online boosting algorithm with theoretical justifications," *arXiv preprint arXiv:1206.6422*, 2012.

[17] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286–295.

[18] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren, "Having a blast: Meta-learning and heterogeneous ensembles for data streams," in *2015 ieee international conference on data mining*. IEEE, 2015, pp. 1003–1008.

[19] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[20] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.

[21] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[22] H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," Naval Research Lab Washington DC, Tech. Rep., 1990.

[23] C.-K. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp. 103–127, 2008.

[24] H. Ma, S. Shen, M. Yu, Z. Yang, M. Fei, and H. Zhou, "Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey," *Swarm and evolutionary computation*, vol. 44, pp. 365–387, 2019.

[25] A. Simões and E. Costa, "On biologically inspired genetic operators: Transformation in the standard genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001), San Francisco, USA*, 2001.

[26] X. Yu, Y. Jin, K. Tang, and X. Yao, "Robust optimization over time—a new perspective on dynamic optimization problems," in *IEEE Congress on evolutionary computation*. IEEE, 2010, pp. 1–6.

[27] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang, "A new multi-swarm particle swarm optimization for robust optimization over time," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 99–109.

[28] M. Mavrovouniotis, C. Li, and S. Yang, "A survey of swarm intelligence for dynamic optimization: Algorithms and applications," *Swarm and Evolutionary Computation*, vol. 33, pp. 1–17, 2017.

[29] T. T. Nguyen, N. Van Pham, M. T. Dang, A. V. Luong, J. McCall, and A. W. C. Liew, "Multi-layer heterogeneous ensemble with classifier and feature selection," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 725–733.

[30] T. T. Nguyen, X. C. Pham, A. W.-C. Liew, and W. Pedrycz, "Aggregation of classifiers: a justifiable information granularity approach," *IEEE transactions on cybernetics*, vol. 49, no. 6, pp. 2168–2177, 2018.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[33] J. Gama, R. Sebastiao, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine learning*, vol. 90, no. 3, pp. 317–346, 2013.