Heterogeneous ensemble selection for evolving data streams.

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

2021

Heterogeneous Ensemble Selection for Evolving Data Streams

Anh Vu Luong[1], Tien Thanh Nguyen[2], Alan Wee-Chung Liew[1], Shilin Wang[3]

[1] School of Information and Communication Technology, Griffith University, Australia

[2] School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, Scotland, UK

[3] School of Electronic Information and Electrical Engineering, Shanghai Jiaotong University, China

Abstract: Ensemble learning has been widely applied to both batch data classification and streaming data classification. For the latter setting, most existing ensemble systems are homogenous, which means they are generated from only one type of learning model. In contrast, by combining several types of different learning models, a heterogeneous ensemble system can achieve greater diversity among its members, which helps to improve its performance. Although heterogeneous ensemble systems have achieved many successes in the batch classification setting, it is not trivial to extend them directly to the data stream setting. In this study, we propose a novel HEterogeneous Ensemble Selection (HEES) method, which dynamically selects an appropriate subset of base classifiers to predict data under the stream setting. We are inspired by the observation that a well-chosen subset of good base classifiers may outperform the whole ensemble system. Here, we define a good candidate as one that expresses not only high predictive performance but also high confidence in its prediction. Our selection process is thus divided into two sub-processes: accurate-candidate selection and confident-candidate selection. We define an accurate candidate in the stream context as a base classifier with high accuracy over the current concept, while a confident candidate as one with a confidence score higher than a certain threshold. In the first sub-process, we employ the *prequential accuracy* to estimate the performance of a base classifier at a specific time, while in the latter sub-process, we propose a new measure to quantify the predictive confidence and provide a method to learn the threshold incrementally. The final ensemble is formed by taking the intersection of the sets of confident classifiers and accurate classifiers. Experiments on a wide range of data streams show that the proposed method achieves competitive performance with lower running time in comparison to the state-of-the-art online ensemble methods.

Keywords: Data streams, Heterogeneous Ensembles, Ensemble Selection

1. Introduction

Traditional classification methods for batch data assume that the entire data can be stored in memory, and the processing time is unlimited. However, many datasets, including sensor network data, video streams,

event logs, and traffic monitoring data, are often generated dynamically in real-time in the form of data streams. There are some restrictions to be noted when designing a learning system for data streams [1]:

- be ready to make predictions on any sequentially arriving instances;
- expect an infinite sequence of instances processed under limited time and memory;
- the data can be statistically non-stationary (the appearance of concept drift); and
- each instance can only be examined once before it is discarded.

By combining the predictions of multiple base classifiers, an ensemble system often gives considerably better results than a single classifier. Ensemble methods have been the winners in many prestigious data mining competitions held by Netflix, KDD, and Kaggle. In addition to the accuracy improvement over a single classifier, an ensemble often provides greater stability in the result. The use of ensemble is often justified by the "No free lunch" theorem, which states that there is no universal best algorithm. In other words, instead of trying to find the best learner for a specific problem, we can gather several methods and combine them together to form a more robust algorithm. When constructing an ensemble system, the differences among the outputs of its individual members, also known as ensemble diversity, play a crucial role in promoting the ensemble performance. Also, ensemble methods are particularly useful for classifying evolving data streams as they allow us to selectively remove or add a set of base classifiers when the concept of data changes over time [2]. Commonly, there are two types of ensembles: Homogeneous Ensemble and Heterogeneous Ensemble. Most ensemble algorithms developed for data streams fall into the first category, in which all base classifiers are generated from one type of learning model, most frequently Hoeffding Trees [3]. In this kind of ensemble, the diversity among base classifiers is often obtained by modifying the characteristics of the base model or the input data. In contrast, the diversity of a heterogeneous ensemble is the result of its members being generated from radically different learning models. Although many heterogeneous ensemble systems have been developed for batch data classification, not many have been developed for the data stream setting.

Ensemble selection is one of the most widely studied topics in ensemble learning as a well-chosen subset of base classifiers often outperforms the whole ensemble system. Recently, many ensemble selection methods have been proposed for batch data [4–6]. Unfortunately, it is not trivial to extend these methods to the data stream setting, since all of them involve solving an optimization problem that requires going through the training set many times to find the best configuration for the ensemble.

In this work, we introduce HEES, a heterogeneous ensemble selection technique that automatically picks a competent subset of base classifiers to make predictions. Within the base classifier pool, we select those with top performance on recently arrived data as well as those with high prediction confidence. To do that,

we define two sub-processes: accurate-candidate selection and confident-candidate selection. An accurate candidate is a base classifier which exhibits high accuracy over recent data. Here, we use *prequential accuracy* [7,8] to estimate the performance of each base classifier on newly arrived samples. On the other hand, we define a confident candidate as a base classifier that has a higher confidence score than its *reliability threshold*. We propose a new measure to quantify the predictive confidence and provide a method to learn the reliability threshold under the stream setting. The selection procedure for each candidate is first formulated as a set of online optimization problems, in each of which an error function of the reliability threshold is minimized by applying the Stochastic Gradient Descent algorithm to update the parameter incrementally. We also provide a theoretical analysis of the convergence of these optimization problems.

Our contributions of this work are: (i) we propose a novel heterogeneous ensemble selection method that dynamically determines an appropriate subset of base classifiers to make predictions based on their confidence scores and their accuracy; (ii) we propose a measure that can estimate the confidence of a classifier's predictions; (iii) we provide a theoretical analysis of the convergence rate of the optimization problems that represent the confident-candidate selection sub-process; (iv) we perform extensive experiments to show that the proposed method is better than many well-known algorithms.

The paper is organized as follows. Section 2 presents related work. In section 3, we propose the novel heterogeneous ensemble selection method. Experimental studies are conducted in Section 4. Results and discussions are shown in section 5. Finally, we draw some conclusions in Section 6.

2. Background and Related work

When working with data streams, there are potentially infinitely many instances which come in one by one and need to be processed under resource constraints [1]. Moreover, the data can be statistically non-stationary, i.e., concept drift might appear over time. Classifiers for data streams, therefore, should have the ability to detect and adapt to concept drifts in order to achieve high predictive performance. The evolving data stream setting is the main motivation for the development of numerous online learning algorithms for multi-class classification [9–12] and multi-label classification [13–15]. In this study, we consider the data stream methods for multi-class classification.

Some classifiers for batch data can be modified to work under the stream setting, e.g., Naïve Bayes [16], k Nearest Neighbours [17], Perceptron [18], Stochastic Gradient Descent [19]. Naïve Bayes is a well-known instance-incremental classifier which performs Bayesian prediction while making the naïve assumption that all features are mutually independent conditional on each class. Its popularity is attributed to its simplicity and low computational cost. k Nearest Neighbours (kNN), also known as lazy learning classifier, makes

predictions by looking for k nearest training samples to the test sample and obtaining prediction based on the majority class among these k neighbors. In the data stream setting, kNN only keeps a sliding window of most recent instances with length w and finds k nearest neighbors in this window instead of the entire data. Perceptron is a linear classifier that performs classification based on a linear function that combines the feature vector with a set of weights. Stochastic gradient descent (SGD), also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function. SGD is a simple yet very efficient way to optimize convex loss functions, especially in the stream setting.

In the literature, most available ensemble systems for data stream are homogeneous, i.e., they comprise of several base classifiers generated from one type of learning algorithm. Most notably, Oza and Russell introduced Online Bagging and Online Boosting [20], adapted versions of Bagging and Boosting to the data stream setting. Online Bagging builds a set of M base classifiers, then incrementally trains each one with a different representation of the original data set, in which samples are given weights according to the Poisson(1) distribution. It has been shown that Online Bagging converges to the behavior of the traditional Bagging if the number of instances approaches infinity. Leverage Bagging [9] enhances Online Bagging by adding more randomization to the input and output of the ensemble members. Traditional Boosting for batch data ameliorates a weak learner by employing a set of base classifiers built sequentially–more specifically, each new base classifier is trained on a copy of the training set with more weight assigned to the instances that were frequently misclassified by the previous base classifier. Oza and Russel used the Poisson distribution to simulate Boosting's behavior in the online setting. From some studies in the literature [20–22], Online Bagging has outperformed Online Boosting. Recently, de Barros et al. proposed a Boosting-like Online Learning Ensemble (BOLE) [10], which improves the accuracy of Oza and Russel's Online Boosting by weakening the conditions for experts to vote and using the DDM [23] method to detect concept drift internally. Pham et al. [24] introduced a homogeneous ensemble algorithm based on Random Projection and Hoeffding Trees. This method applies the Random Projection technique to create a number of diverse instances for each instance and then forms an ensemble of several Hoeffiding Trees trained on these new instances. This method applies the Random Projection technique to create a number of diverse instances for each instance and then forms an ensemble of several Hoeffiding Trees trained on these new instances. One advantage of using Random Projection technique is that it can reduce the dimension of the data, giving the method the ability to deal with very high-dimensional data streams. Adaptive Random Forest (ARF) [12] aims to adapt the classical Random Forest to the data stream setting by employing the online bootstrap resampling, similar to Leveraging Bagging. To deal with concept drift, ARF uses two change detectors per base tree to detect warnings and drifts. In particular, when a warning is triggered, a background tree is created and updated without affecting the ensemble predictions. If the warning escalates

to a drift after a period of time, the background tree replaces the corresponding base tree in the ensemble. Although this mechanism helps ARF effectively handle concept drift, it slows down the method and increases the memory used. Recently, Gomes et al. introduced Streaming Random Patches (SRP) [36] , which resembles the classic Random Patches by combining the Random Subspace method and Online Bagging. SRP exploits the global subspace randomization (as in Random Subspace), while ARF takes advantage of local subspace randomization (as in Random Forest). In comparison to ARF, the SRP method is slightly better in terms of accuracy, but its runtime is higher.

Heterogeneous ensembles have been extensively studied for batch data [25–27]. However, only a small number of them were developed for the data stream setting, most remarkably HEFT-Stream [28] and BLAST [29]. In the HEFT-Stream method, an ensemble of Hoeffding Trees and Naïve Bayes learners is maintained, and when a sudden drift occurs, it adds a new classifier, whose type matches the current learner with the highest weight. This method focuses on tackling the feature drift problem, which is beyond the scope of this paper. The BLAST ensemble, on the other hand, used the Online Performance Estimation framework [11, 29], which measures the performance of base classifiers over the set of w most recent samples, to select k best base learners to classify a sample. Note that the fixed number of chosen base classifiers limits the flexibility of this model. The authors proposed to use $k = 1$, which means only one base classifier is selected to make predictions. This choice only works well when there exists a dominant candidate which outperforms all other base learners for a specific concept of data, which is not always the case in practice. Also, as BLAST does not take into account the confidence in predictions of each base classifier, it sometimes selects a classifier having ambiguous predictions, i.e., predictions are very closed to the decision boundary. Recently, Idrees et al. introduced Heterogeneous Dynamic Weighted Majority (HDWM) [37], which aims to switch between different types of base classifiers in an ensemble to promote the predictive performance of online learning. It utilizes the "seed" learners of different types to create ensemble diversity, avoiding the loss of diversity when base learners are removed from the ensemble due to concept drift. Although the method successfully reduced human effort to choose the base classifiers for an ensemble, its performance is not competitive to some state-of-the-art homogeneous ensembles.

Some ensemble selection methods have been proposed for data stream learning [33,11,38]. van Rijn et al. proposed an algorithm selection for data streams, which is the first effort to do meta-learning on data streams [33]. They utilized the sliding window technique to handle streaming data, aiming to predict what algorithm will perform best for the next window of data. For each window, they first construct a new meta-dataset based on data characteristics measured in the previous window and the meta-knowledge. An abstract meta-algorithm is then employed to determine which algorithm will be used to predict the next window of instances. One limitation of this method is how to determine the optimal value of the window size. Another

drawback is that it is a batch-incremental method, which is not applicable to high-speed instance-incremental data streams. The heterogeneous ensemble BLAST [29] we mentioned above can be considered as an ensemble selection method since it tries to select k best base classifiers to predict the next instance based on the estimated performance of these learners on the current window of data. Recently, Krawczyk et al. presented another ensemble selection method which allows the base classifiers to abstain from contributing to the final decision of the ensemble [38]. The confidence level of each member is monitored for each incoming instance and only learners with the confidence score exceed a certain threshold are selected. In this method, the authors used one threshold for all the members, and this threshold is updated based on the correct/incorrect decision of the whole ensemble. The use of only one threshold for all base learners, however, limits the flexibility of this method.

One problem when dealing with data stream is concept drift. Data streams that exhibit concept drifts are referred to as evolving or non-stationary data streams [2]. Since the nature of the data can change over time, classifiers for data stream setting should have the ability to detect and adapt when these changes occur. Some classifiers can naturally deal with concept drift, e.g., k Nearest Neighbour keeps a window of w most recent samples so that it avoids learning obsolete samples, and therefore focuses on the current concept. Another approach to tackle concept drift is to integrate the drift detectors into the classifiers. Examples of well-known drift detectors are Drift Detection Method (DDM) [23], Adaptive Sliding Window Algorithm (ADWIN) [30], and Early Drift Detection Method (EDDM) [31]. These are stand-alone methods that can detect concept drifts and can be combined with any online classifier. A common assumption used in these detectors is that if the distribution generating the data is stationary, the accuracy of the learning algorithm will increase or maintain when we train the classifier on more data. From this assumption, a significant drop in the accuracy rate is the evidence that the current classifier is out of date, and a new classifier should be constructed to replace this obsolete learner.

3. Proposed method

3.1. Problem formulation

Given a data stream S whose instances come as a sequence of data points $\{\boldsymbol{x}^{(t)}\}$ where $\boldsymbol{x}^{(t)}$ is a d-dimensional feature vector that arrives at time t. Assume that we can obtain the true label $y^{(t)} \in \mathcal{Y} = \{y_1, y_2, \dots, y_M\}$ of the data point $\boldsymbol{x}^{(t)}$ after some time, and it can be used for classifier training later on. These are common assumptions for classification problems in the data stream setting [2,9,20,22].

When applying a heterogeneous ensemble of classifiers for data streams, we are given a set of Q online base classifiers $\boldsymbol{B} = \{B_q\}, q = 1, \dots, Q$. For a data point $\boldsymbol{x}^{(t)}$, each base classifier returns output in the form

of *Soft Label*, i.e., the output that $\boldsymbol{x}^{(t)}$ is assigned to y_m given by B_q: $P_q(y_m|\boldsymbol{x}^{(t)}) \in [0,1]$. The concatenated outputs of the Q base classifiers on a data point $\boldsymbol{x}^{(t)}$ is given by:

$$L(\boldsymbol{x}^{(t)}) = [\underbrace{P_1(y_1|\boldsymbol{x}^{(t)}) \dots P_1(y_M|\boldsymbol{x}^{(t)})}_{predictions\ of\ 1^{st} classifier} \dots \underbrace{P_Q(y_1|\boldsymbol{x}^{(t)}) \dots P_Q(y_M|\boldsymbol{x}^{(t)})}_{predictions\ of\ Q^{th}\ classifier}] \quad (1)$$

After obtaining $L(\boldsymbol{x}^{(t)})$, we aim to develop an ensemble selection method to find a subset of $\boldsymbol{B}$ that contains appropriate candidates to predict $\boldsymbol{x}^{(t)}$. This improves not only the predictive accuracy but also efficiency in the computation for classification of the ensemble.

3.2. Confident-candidate selection sub-process

When classifying a sample $\boldsymbol{x}^{(t)}$, the prediction of a classifier B_q is with high confident if the probability output for one class is much higher than that for the other classes, e.g., $P_q(y_1|\boldsymbol{x}^{(t)}) = 0.95$, $P_q(y_2|\boldsymbol{x}^{(t)}) = 0.025$, and $P_q(y_3|\boldsymbol{x}^{(t)}) = 0.025$ in a classification problem with three labels $\mathcal{Y} = \{y_1, y_2, y_3\}$. In contrast, if the predictions for all classes are similar for an instance $\boldsymbol{x}^{(t)}$, e.g., $P_q(y_1|\boldsymbol{x}^{(t)}) = 0.32$, $P_q(y_2|\boldsymbol{x}^{(t)}) = 0.34$, and $P_q(y_3|\boldsymbol{x}^{(t)}) = 0.34$, it is hard to assign $\boldsymbol{x}^{(t)}$ to a class. Based on this observation, we propose a new measure $e_q(\boldsymbol{x}^{(t)})$ to quantify the confidence of the outputs given by a base classifier B_q on a data point $\boldsymbol{x}^{(t)}$:

$$e_q(\boldsymbol{x}^{(t)}) = P_q(y_k|\boldsymbol{x}^{(t)}) - \frac{1}{M-1}\sum_{\substack{m=1 \\ m\neq k}}^{M} P_q(y_m|\boldsymbol{x}^{(t)}) \quad (2)$$

where $k = argmax_{m=1\dots M} P_q(y_m|\boldsymbol{x}^{(t)})$. Note that $e_q(\boldsymbol{x}^{(t)})$ is bounded in [0,1]. It is recognized that $e_q(\boldsymbol{x}^{(t)})$ is the difference between the maximum value among the predictions and the average of the other values. Therefore, the larger $e_q(\boldsymbol{x}^{(t)})$ is, the higher the confidence of the outputs of classifier B_q for $\boldsymbol{x}^{(t)}$ is, i.e., confidence is proportional to $e_q(\boldsymbol{x}^{(t)})$. The idea of subtracting the maximum probability to the average of the other posterior probabilities in Eq. (2) comes from a fuzzy classification system in [44]. For each base classifier, we use this formulation to represent a certain mar- gin of difference between the predicted probabilities, which is necessary before comparing the accuracy of a base classifier to the others.

When making predictions for a sample $\boldsymbol{x}^{(t)}$, we select base classifiers with high confidence scores since they allow us to make decisions easily in comparison to those with low confidence scores. We introduce the *reliability threshold* to specify whether a base classifier expresses enough confidence in its prediction according to its confidence score. In particular, we assign a reliability threshold θ_q to each base classifier B_q and define a *confident classifier* as one whose confidence score is higher than its reliability threshold. As different learning algorithms are used to generate the different base classifiers, aka different hypotheses,

in a heterogeneous ensemble system, different base classifiers should have different thresholds. The selection rule for each base classifier B_q can be formulated as follows:

$$\begin{cases} B_q \text{ is selected to classify } \boldsymbol{x}^{(t)}, & \text{if } e_q\left(\boldsymbol{x}^{(t)}\right) > \theta_q \\ B_q \text{ is rejected to classify } \boldsymbol{x}^{(t)}, & \text{otherwise} \end{cases} \tag{3}$$

Our task now is to determine the values of the reliability thresholds, which allows us to select base classifiers for the sample $\boldsymbol{x}^{(t)}$. To do this, we formulate the confident-candidate selection sub-process as a set of Q optimization problems, in each of which the objective is a function of θ_q, and then solve them to obtain the optimal values for $\theta_q (q = 1,2, \dots, Q)$.

Consider the selection problem $\mathcal{P}_q(\boldsymbol{x}^{(t)})$, our aim is to determine whether or not we should select the base classifier B_q for an instance $\boldsymbol{x}^{(t)}$. This can be viewed as a binary classification problem whose goal is to classify the base classifiers for an instance $\boldsymbol{x}^{(t)}$ into two classes: *selected (class 1)* and *rejected (class 0)*. Equation (3) specifies how we make predictions in the new classification problem $\mathcal{P}_q$. Particularly, we decide to select B_q for an instance $\boldsymbol{x}^{(t)}$ if it is confident enough about its prediction, or we predict $\hat{r}_q(\boldsymbol{x}^{(t)}) = 1$ if $e_q\left(\boldsymbol{x}^{(t)}\right) > \theta_q$, and vice versa. This is equivalent to $\hat{r}_q = \mathbb{I}[e_q\left(\boldsymbol{x}^{(t)}\right) > \theta_q]$, where $\mathbb{I}[\cdot]$ is the indicator function.

We now define the loss function for $\mathcal{P}_q(\boldsymbol{x}^{(t)})$ based on our prediction $\hat{r}_q(\boldsymbol{x}^{(t)})$ and the ground truth $r_q(\boldsymbol{x}^{(t)})$ of this problem. To do this, we first show how to obtain the true class label for $\mathcal{P}_q(\boldsymbol{x}^{(t)})$. Given an instance $\boldsymbol{x}^{(t)}$, the objective is to select a base classifier which correctly predicts $\boldsymbol{x}^{(t)}$, and reject a base classifier that misclassifies $\boldsymbol{x}^{(t)}$. Based on this observation, we can determine the ground truth $r_q(\boldsymbol{x}^{(t)})$ for the problem $\mathcal{P}_q(\boldsymbol{x}^{(t)})$ as follows. When the true label $y^{(t)}$ of $\boldsymbol{x}^{(t)}$ in the original classification problem is available, we know whether or not the classifier B_q makes a correct prediction for $\boldsymbol{x}^{(t)}$. If $\boldsymbol{x}^{(t)}$ is correctly classified by B_q, this sample will belong to the class *selected* (or $r_q(\boldsymbol{x}^{(t)}) = 1$) in the problem $\mathcal{P}_q(\boldsymbol{x}^{(t)})$, whereas if B_q misclassifies $\boldsymbol{x}^{(t)}$, then $r_q(\boldsymbol{x}^{(t)}) = 0$ (class *rejected*). The labeling process for $\mathcal{P}_q(\boldsymbol{x}^{(t)})$ can be summarized as follows:

$$\begin{cases} r_q\left(\boldsymbol{x}^{(t)}\right) = 1\,, & \text{if } B_q \text{ correctly classifies } \boldsymbol{x}^{(t)} \\ r_q\left(\boldsymbol{x}^{(t)}\right) = 0, & \text{if } B_q \text{ misclassifies } \boldsymbol{x}^{(t)} \end{cases} \tag{4}$$

Given the true label r_q and the prediction $\hat{r}_q$ for an instance $\boldsymbol{x}^{(t)}$ in the problem $\mathcal{P}_q$, we can apply the cross-entropy loss function as follows:

$$\mathcal{L}_q\left(\boldsymbol{x}^{(t)}, \theta_q\right) = \begin{cases} -r_q \log\left(s_q\right) - (1 - r_q)\log(1 - s_q), & \text{if } r_q \neq \hat{r}_q = \mathbb{I}[e_q\left(\boldsymbol{x}^{(t)}\right) > \theta_q] \\ 0, & otherwise \end{cases} \tag{5}$$

where $r_q = r_q\left(\boldsymbol{x}^{(t)}\right)$, $e_q = e_q(\boldsymbol{x}^{(t)})$, and $s_q = s_q\left(\boldsymbol{x}^{(t)}, \theta_q\right) = \text{sigmoid}\left(e_q\left(\boldsymbol{x}^{(t)}\right) - \theta_q\right)$.

Our goal is to find the optimal solution to the following optimization problem:

$$\min_{\theta_q \in \mathbb{R}} \quad \mathcal{L}_q\left(\boldsymbol{x}^{(t)}, \theta_q\right) \tag{6}$$

To solve this problem under the stream setting, where the reliability threshold of each base classifier needs to be updated on the fly based on newly arrived information, we apply the Stochastic Gradient Descent algorithm to incrementally adjust each threshold $\theta_q, q = 1, \ldots, Q$ after the true label of the instance $\boldsymbol{x}^{(t)}$ is revealed [41, 42]. In particular, whenever we make a wrong selection for B_q at time t, i.e., $r_q\left(\boldsymbol{x}^{(t)}\right) \neq \mathbb{I}[e_q > \theta_q^{(t)}]$, we update the corresponding threshold by the following rule:

$$\theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)} \frac{\partial \mathcal{L}_q}{\partial \theta_q} \tag{7}$$

where $\eta^{(t)}$ is the learning rate.

We simplify the notations $r_q = r_q\left(\boldsymbol{x}^{(t)}\right), e_q = e_q\left(\boldsymbol{x}^{(t)}\right), s_q = s_q(\boldsymbol{x}^{(t)}, \theta_q)$, and then apply the chain rule:

$$\frac{\partial \mathcal{L}_q}{\partial \theta_q} = \frac{\partial \mathcal{L}_q}{\partial s_q} \times \frac{\partial s_q}{\partial \theta_q} = \left[\frac{-r_q}{s_q} + \frac{1 - r_q}{1 - s_q}\right] \times s_q\left(1 - s_q\right)(-1)$$

$$= -r_q\left(s_q - 1\right) - \left(1 - r_q\right)s_q$$

$$= r_q - s_q$$

Thus, the final update rule is given by:

$$\begin{cases} \theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)}\left(r_q - s_q\right), & \text{if } r_q \neq \mathbb{I}\left[e_q > \theta_q^{(t)}\right] \\ \theta_q^{(t+1)} = \theta_q^{(t)}, & otherwise \end{cases} \tag{8}$$

We next provide the convergence analysis of this optimization problem. Specifically, we prove that if we use the update rule (8), the reliability threshold θ_q converges to the optimal solution θ_q^* which minimizes the loss $\mathcal{L}_q$, and the convergence rate is $O(1/\sqrt{t})$. The main proof is shown in Theorem 1, which is supported by three lemmas A1, A2, and A3 in the Appendix.

Theorem 1: Let $\theta_q^* = \underset{\theta_q \in \mathbb{R}}{\operatorname{argmin}} \quad \mathcal{L}_q\left(\boldsymbol{x}^{(t)}, \theta_q\right)$, and $\mathcal{L}_q^* = \mathcal{L}_q(\boldsymbol{x}^{(t)}, \theta_q^*)$. Choose $\eta^{(t)} = \frac{c}{\sqrt{t}}$ where c is a predefined positive number, then:

$$\mathcal{L}_q(\bar{\theta}_q) - \mathcal{L}_q^* \leq \frac{c^{-1}|1+2c|^2 + c\frac{2\sqrt{t}-1}{\sqrt{t}}}{2\sqrt{t}} \tag{9}$$

Where $\overline{\theta}_q = \frac{1}{t}(\theta_q^{(1)} + \theta_q^{(2)} + \cdots + \theta_q^{(t)})$

Proof.

Note that the cross-entropy loss function $\mathcal{L}_q$ is convex. Hence

$$\frac{\partial \mathcal{L}_q}{\partial \theta_q} \times (\theta_q^{(t)} - \theta_q^*) \geq \mathcal{L}_q - \mathcal{L}_q^* \tag{10}$$

On the other hand:

$$\mathbb{E}\left[\left|\theta_q^{(t+1)} - \theta_q^*\right|^2\right] = \mathbb{E}\left[\left|\theta_q^{(t)} - \eta^{(t)}\frac{\partial \mathcal{L}_q}{\partial \theta_q} - \theta_q^*\right|^2\right]$$

$$= \mathbb{E}[\left|\theta_q^{(t)} - \theta_q^*\right|^2] - 2\eta^{(t)}\mathbb{E}\left[\frac{\partial \mathcal{L}_q}{\partial \theta_q} \times (\theta_q^{(t)} - \theta_q^*)\right] + (\eta^{(t)})^2\mathbb{E}\left[\left|\frac{\partial \mathcal{L}_q}{\partial \theta_q}\right|^2\right]$$

$$\leq \mathbb{E}[\left|\theta_q^{(t)} - \theta_q^*\right|^2] - 2\eta^{(t)}\mathbb{E}\left[\frac{\partial \mathcal{L}_q}{\partial \theta_q} \times (\theta_q^{(t)} - \theta_q^*)\right] + (\eta^{(t)})^2$$

(Here we use the result from Lemma A1: $\mathbb{E}\left[\left|\frac{\partial \mathcal{L}_q}{\partial \theta_q}\right|^2\right] \in [0,1]$)

Combining with the inequality (10), we obtain:

$$\mathbb{E}\left[\left|\theta_q^{(t+1)} - \theta_q^*\right|^2\right] \leq \mathbb{E}[\left|\theta_q^{(t)} - \theta_q^*\right|^2] - 2\eta^{(t)}\left(\mathcal{L}_q - \mathcal{L}_q^*\right) + (\eta^{(t)})^2$$

$$\Rightarrow 2\left(\mathcal{L}_q - \mathcal{L}_q^*\right) \leq (\eta^{(t)})^{-1}\mathbb{E}\left[\left|\theta_q^{(t)} - \theta_q^*\right|^2\right] + \eta^{(t)} - (\eta^{(t)})^{-1}\mathbb{E}\left[\left|\theta_q^{(t+1)} - \theta_q^*\right|^2\right]$$

Calculating the sum of the inequality above from $i = 1,2,\ldots,t$ we have:

$$2\sum_{i=1}^{t}\left(\mathcal{L}_q - \mathcal{L}_q^*\right) \leq (\eta^{(1)})^{-1}\mathbb{E}\left[\left|\theta_q^{(1)} - \theta_q^*\right|^2\right] + \sum_{i=1}^{t}\eta^{(i)} + \sum_{i=2}^{t}((\eta^{(i)})^{-1} - (\eta^{(i-1)})^{-1})\mathbb{E}\left[\left|\theta_q^{(i)} - \theta_q^*\right|^2\right]$$

$$\leq \left(\eta^{(1)}\right)^{-1}|1+2c|^2 + \sum_{i=1}^{t}\eta^{(i)} + \sum_{i=2}^{t}\left((\eta^{(i)})^{-1} - (\eta^{(i-1)})^{-1}\right)|1+2c|^2$$

(Here we use the result from Lemma A3)

$$= (\eta^{(t)})^{-1}|1+2c|^2 + \sum_{i=1}^{t}\eta^{(i)}$$

$$= \frac{\sqrt{t}}{c}|1+2c|^2 + \sum_{i=1}^{t}\frac{c}{\sqrt{i}}$$

$$\leq \frac{\sqrt{t}}{c}|1+2c|^2 + c(2\sqrt{t}-1)$$

where the last inequality uses $\sum_{i=1}^{t}\frac{1}{\sqrt{i}} \leq 2\sqrt{t}-1$.

From Jensen's inequality, we have:

$$\frac{1}{t}\sum_{i=1}^{t}\left(\mathcal{L}_q(\theta_q^{(i)}) - \mathcal{L}_q^*\right) \geq \mathcal{L}_q\left(\bar{\theta}_q\right) - \mathcal{L}_q^*$$

which indicates that:

$$\mathcal{L}_q\left(\bar{\theta}_q\right) - \mathcal{L}_q^* \leq \frac{c^{-1}\sqrt{t}|1+2c|^2 + c\left(2\sqrt{t}-1\right)}{2t} = \frac{c^{-1}|1+2c|^2 + c\dfrac{2\sqrt{t}-1}{\sqrt{t}}}{2\sqrt{t}}$$

In other words, $\mathcal{L}_q\left(\bar{\theta}_q\right)$ converges to $\mathcal{L}_q^*$ as t approaches $+\infty$, and the convergence rate is $O(1/\sqrt{t})$.

The confident-candidate selection sub-process is presented in Function 1 and Function 2. The Function 1, namely `C_Selection`, is employed to select confident base classifiers for a sample $\boldsymbol{x}^{(t)}$. First, we pass $\boldsymbol{x}^{(t)}$ through all base classifiers to obtain the concatenated outputs $L(\boldsymbol{x}^{(t)})$ (line 3) and then use it to calculate the confidence score of each base classifier (line 5). Finally, a comparison between this confidence score and the corresponding reliability threshold is drawn to decide whether or not should we select a base classifier (lines 6-7). The Function 2 (or `C_Update`) is used to update the reliability thresholds when the true label $y^{(t)}$ of the sample $\boldsymbol{x}^{(t)}$ is available. This function checks if a base classifier makes a wrong selection for $\boldsymbol{x}^{(t)}$ (line 5), then its reliability threshold will be updated by equation (8) (lines 6-7).

Function 1. Confident-candidate selection sub-process

Input: sample $\boldsymbol{x}^{(t)}$, $\boldsymbol{B} = \{B_q\}$, $\boldsymbol{\theta} = [\theta_1, \dots, \theta_Q]$

Output: $\boldsymbol{C}$ – the set of confident classifiers

Function `C_Selection`:
  $\boldsymbol{C} = \{\emptyset\}$
  Compute $L(\boldsymbol{x}^{(t)})$ by equation (1)
  For $q = 1,2, \dots, Q$:
    Compute $e_q(\boldsymbol{x}^{(t)})$ by equation (2)
    If $e_q(\boldsymbol{x}^{(t)}) > \theta_q^{(t)}$:
      $\boldsymbol{C} = \boldsymbol{C} \cup C_q$
    End If
  End For
  Return $\boldsymbol{C}$
End Function

Function 2. Update reliability thresholds $\boldsymbol{\theta} = [\theta_1, \dots, \theta_Q]$

Input: sample $\boldsymbol{x}^{(t)}$, label $y^{(t)}$, $\boldsymbol{B} = \{B_1, \dots, B_Q\}$
$\boldsymbol{\theta} = [\theta_1, \dots, \theta_Q]$ – the list of reliability thresholds

Output: $\boldsymbol{\theta} = [\theta_1, \dots, \theta_Q]$ – the list of updated reliability thresholds

Function `C_Update`:
  For $q = 1,2, \dots, Q$:
    Compute $e_q(\boldsymbol{x}^{(t)})$ and the prediction of B_q: $\hat{y}_q^{(t)} = \text{argmax}_{y_m \in \mathcal{Y}} P_q(y_m | \boldsymbol{x}^{(t)})$
    Compute $r_q(\boldsymbol{x}^{(t)}) = \mathbb{I}[\hat{y}_q^{(t)} = y^{(t)}]$
    If $r_q(\boldsymbol{x}^{(t)}) \neq \mathbb{I}\left[e_q(\boldsymbol{x}^{(t)}) > \theta_q^{(t)}\right]$:
      Compute $s_q(\boldsymbol{x}^{(t)}) = sigmoid\left(e_q(\boldsymbol{x}^{(t)}) - \theta_q^{(t)}\right)$
      Update $\theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)}(r_q(\boldsymbol{x}^{(t)}) - s_q(\boldsymbol{x}^{(t)}))$
    End If
  End For
  Return $\boldsymbol{\theta} = [\theta_1, \dots, \theta_Q]$
End Function

3.3. Accurate-candidate selection sub-process

In this sub-process, we use a performance measure to rank all the base classifiers and then choose the top K best-performing candidates. The traditional metrics for batch classification, e.g., accuracy or F1 score at a certain time, are not a reasonable choice in the data stream context [8]. Instead, metrics integrated with a *forgetting mechanism* are especially suitable for the stream setting, since they can estimate how well a classifier performs on recently arrived samples while ignoring the obsolete ones. In other words, the metrics used in this setting should express the performance of a classifier on the current concept. This observation inspires us to use the prequential accuracy metric [8]:

$$\rho^{(\alpha)}(t) = \frac{\sum_{k=1}^{t} \alpha^{t-k}(1-l_k)}{\sum_{k=1}^{t} \alpha^{t-k}} \tag{11}$$

Here, $\rho^{(\alpha)}(t)$ is the prequential accuracy at the t-th instance, l_k is the 0-1 loss at the k-th instance, and α $(0 < \alpha \leq 1)$ is the *fading factor* to realize the forgetting mechanism. The smaller the fading factor α is, the faster we forget the accuracy rates of old instances. One can notice that if $\alpha = 1$, the prequential accuracy metric becomes the standard accuracy metric commonly used in batch classification. In practice, the fading factor α is often chosen to be a number close to 1. From now on, we use ρ to denote $\rho^{(\alpha)}$ to keep the notation uncluttered.

Function 3. Update prequential accuracies $\boldsymbol{\rho} = [\rho_1, \dots, \rho_Q]$

Input: sample $\boldsymbol{x}^{(t)}$, label $y^{(t)}$, $\boldsymbol{B} = \{B_1, \dots, B_Q\}$ $\boldsymbol{\rho} = [\rho_1, \dots, \rho_Q]$ – the list of prequential accuracies
Output: $\boldsymbol{\rho}'$ – the sorted list of prequential accuracies $\boldsymbol{I} = [i_1, i_2, \dots, i_Q]$ – indices of $\boldsymbol{\rho}'$
Function A_Update: **For** $q = 1,2, \dots, Q$: Compute the prediction of B_q: $\hat{y}_q^{(t)} = \text{argmax}_{y_m \in y} P_q(y_m \mid \boldsymbol{x}^{(t)})$ Compute 0-1 loss: $l_q = \mathbb{I}[\hat{y}_q^{(t)} \neq y^{(t)}]$ Update the prequential p_q by equation (13) **End For** $\boldsymbol{p}' = sort(p) = [p_{i_1}, p_{i_2}, \dots, p_{i_Q}]$ $\boldsymbol{I} = [i_1, i_2, \dots, i_Q]$ Return $\boldsymbol{p}', \boldsymbol{I}$

To update the prequential accuracy incrementally, we will calculate $\rho(t+1)$ through $\rho(t)$ and the 0-1 loss at the $(t+1)$-th instance l_{t+1}. Based on the definition of the prequential accuracy, we have:

$$\rho(t+1) = \frac{\sum_{k=1}^{t+1} \alpha^{t+1-k}(1-l_k)}{\sum_{k=1}^{t+1} \alpha^{t+1-k}} = \frac{\sum_{k=1}^{t} \alpha^{t-k}(1-l_k)\times\alpha+(1-l_{t+1})}{\sum_{k=1}^{t} \alpha^{t-k}\times\alpha+1} \quad (12)$$

Let the denominator of $\rho(t)$ be $\mathrm{Z}(t) = \sum_{k=1}^{t} \alpha^{t-k}$, then the denominator of $\rho(t+1)$ can be calculated as $\mathrm{Z}(t+1) = \sum_{k=1}^{t} \alpha^{t-k} \times \alpha + 1 = \mathrm{Z}(t) \times \alpha + 1$. Substitute $Z(t+1)$ into equation (12), we have

$$\rho(t+1) = \frac{\rho(t)\times\mathrm{Z}(t)\times\alpha+(1-l_{t+1})}{\mathrm{Z}(t+1)} \quad (13)$$

When the true label of the instance $\boldsymbol{x}^{(t+1)}$ is available, we know the value of l_{t+1}, and we can also compute the value of $Z(t+1)$ through $Z(t)$; thus, we can now have access to the value $\rho(t+1)$ by using equation (13).

The update phase of the accurate-candidate selection sub-process is presented in Function 3 (`A_Update`). In this function, we apply equation (13) to update the prequential accuracy corresponding to each base classifier (lines 2-6) and then sort the list of updated prequential accuracies. Finally, the function returns this sorted list and its indices.

3.4. Proposed ensemble selection method

We have discussed how to select confident base classifiers in section 3.2 and how to select accurate base classifiers in section 3.3. After having these two sets, we can intersect them to obtain both high-accuracy and high-confidence base classifiers. To balance the contributions of the two sub-processes, we choose the number of high-accuracy candidates K to be the same as the number of base classifiers returned by the confident-candidate selection sub-process.

The dynamic selection mechanism gives our framework high flexibility to deal with changes in data. The number of base learners is automatically adapted to the new concept in the data without human intervention. If we use a fixed size of the selected set, we need to tune one more parameter, which is not practical in data stream learning.

The predictions of the base classifiers in the final set can either contribute equally to the final prediction or be weighted according to the prequential accuracy calculated in the accurate-candidate selection sub-process. We apply the latter strategy in our framework as some studies [10–12] suggested that using a weighted voting scheme often gives higher predictive accuracy than a normal voting scheme. Here, the prequential accuracy is used to weight the votes because it reflects the performance of a candidate on the current data concept.

To handle concept drift, we integrate the ADWIN Drift Detector [30] to our proposed framework. ADWIN detect changes by keeping recently arrived items in a variable-length window, whose maximal length

statistically satisfies the hypothesis "there has been no change in the average value inside the window". If this hypothesis is violated, a change is detected, and the window shrinks until the hypothesis holds. ADWIN is practically useful, owning to the fact that it is parameter- and assumption-free, which means that it automatically detects and adapts to the rate of concept drift. The only parameter in ADWIN is the confidence bound δ, a parameter inherent to all algorithms dealing with the random process, specifying how confident the algorithm is about its outputs. Also, by compressing the window using a variant of the exponential histogram technique, the resources used by ADWIN is highly efficient as it keeps a window of length W using only O(log W) memory and O(log W) processing time per item. In our framework, we use ADWIN to keep track of the accuracy of each base classifier. When a change is detected for a base classifier, we reset the learning process of this classifier. In this way, the base classifier completely forgets about the old data and starts learning the new concept.

The details of the proposed method are presented in Algorithm 1. First, some necessary variables ($\boldsymbol{B}, \boldsymbol{\rho}, \boldsymbol{I}, \boldsymbol{\theta}$) are initialized (lines 1-4). For each instance $\boldsymbol{x}^{(t)}$, we use Function 1 (`C_Selection`) to select a set $\boldsymbol{C}$ of confident candidates (lines 5-7). Based on the cardinality K of this set, we select a set $\boldsymbol{A}$ of the top K highest-accuracy candidates (lines 8-11). We then take the intersection of the two sets to obtain the final set $\boldsymbol{S}$ (line 12). Note that we select all the base classifiers to vote if $\boldsymbol{S}$ is empty (lines 13-15). The final step of the classification phase is to combine the votes of members in $\boldsymbol{S}$ to derive the final vote (line 16). Here we use the prequential accuracies to weight the votes as described above. When the true label $y^{(t)}$ of the sample $\boldsymbol{x}^{(t)}$ is available, we pass it through Function 2 (`C_Update`) and Function 3 (`A_Update`) to update the reliability thresholds $\boldsymbol{\theta}$ and the prequential accuracies $\boldsymbol{\rho}$, respectively (lines 17-18). Finally, ADWIN is used to detect changes for each base classifier (lines 19-25). If ADWIN triggers a change for a base classifier, we reset this classifier and then start its new learning process using the current instance $\boldsymbol{x}^{(t)}$ and its true label $y^{(t)}$.

Algorithm 1. Proposed ensemble selection method

Input: sample $\boldsymbol{x}^{(t)}$, label $y^{(t)}$, $\boldsymbol{B} = \{B_1, \dots, B_q\}$

1. Initialize base models B_q for all $q \in \{1,2, \dots, Q\}$
2. Initialize the list of prequential accuracies $\boldsymbol{\rho}$: $\rho_q = 0$ for all $q \in \{1,2, \dots, Q\}$
3. Initialize the indices of $\boldsymbol{\rho}$: $\boldsymbol{I} = [1, \dots, Q]$
4. Initialize the list of reliability thresholds $\boldsymbol{\theta}$: $\theta_q = 1$ for all $q \in \{1,2, \dots, Q\}$
5. **For** all samples $\boldsymbol{x}^{(t)}$:
   (Classify the sample $\boldsymbol{x}^{(t)}$)
6. $\boldsymbol{C} =$ `C_Selection`$(\boldsymbol{x}^{(t)}, \boldsymbol{B}, \boldsymbol{\theta})$
7. $K = |\boldsymbol{C}|$

8. $\boldsymbol{A} = \{\emptyset\}$ *($\boldsymbol{A}$ is the set of K highest-accuracy candidates)*
9. **For** $q = 1,2,\dots,K$:
10. $\boldsymbol{A} = \boldsymbol{A} \cup B_{i_q}$
11. **End For**
12. $\boldsymbol{S} = \boldsymbol{C} \cap \boldsymbol{A}$
13. **If** $\boldsymbol{S} = \emptyset$**:**
14. $\boldsymbol{S} = \boldsymbol{B}$
15. **End If**
16. $\hat{y}^{(t)} = \boldsymbol{S}.predict(\boldsymbol{x}^{(t)})$ *(Combine the votes of members in $\boldsymbol{S}$)*

(Update the ensemble when the true label $y^{(t)}$ of $\boldsymbol{x}^{(t)}$ is revealed)

17. $\boldsymbol{\theta} = \texttt{C\_Update}(\boldsymbol{x}^{(t)}, y^{(t)}, \boldsymbol{B}, \boldsymbol{\theta})$
18. $\boldsymbol{\rho}, \boldsymbol{I} = \texttt{A\_Update}(\boldsymbol{x}^{(t)}, y^{(t)}, \boldsymbol{B}, \boldsymbol{\rho})$
19. **For** $q = 1,2,\dots,Q$:
20. **If** ADWIN detects change for B_q:
21. Reset the base classifier B_q
22. Update the base classifier B_q with $\left(\boldsymbol{x}^{(t)}, y^{(t)}\right)$
23. **End If**
24. **End For**
25. **End For**

Table 1. Data streams used in the experiments

Dataset name	# of observations	# of classes	# of dimension
20_newsgroups	399,940	2	1000
Electricity	45,312	2	8
KDDCup99_full	4,898,431	23	41
Adult	48,842	2	14
Vehicle	98,528	2	100
Cod-rna	488,565	2	8
IMDB-F.drama	120,919	2	1001
CovType	581,012	7	54
Agrawal	1,000,000	2	9
AssetNegotiation-F2	1,000,000	2	5
AssetNegotiation-F3	1,000,000	2	5
AssetNegotiation-F4	1,000,000	2	5
BNG_tic-tac-toe	39,366	2	9
BNG_vote	131,072	2	16
BNG_trains	1,000,000	2	32
BNG_soybean	1,000,000	19	35
BNG_mushroom	1,000,000	2	22
BNG_kr-vs-kp_small	1,000,000	2	36
BNG_segment	1,000,000	7	19
BNG_ionosphere	1,000,000	2	34
BNG_credit-g	1,000,000	2	20
BNG_SPECT	1,000,000	2	22
BNG_solar-flare	1,000,000	6	12

BNG_page-blocks	295,245	5	10
BNG_lymph	1,000,000	4	18
BNG_labor	1,000,000	2	16
BNG_hepatitis	1,000,000	2	19
BNG_heart-statlog	1,000,000	2	13
BNG_heart-c	1,000,000	5	13
BNG_dermatology	1,000,000	6	34
BNG_credit-a	1,000,000	2	15
BNG_spambase	1,000,000	2	57
BNG_optdigits	1,000,000	10	64
BNG_anneal	1,000,000	6	38
BNG_wine	1,000,000	3	13
BNG_waveform-5000	1,000,000	3	40
BNG_sonar	1,000,000	2	60
BNG_satimage	1,000,000	6	36
BNG_cmc	55,296	3	9
Hyperplane_10_1E-4	1,000,000	5	10
Hyperplane_10_1E-3	1,000,000	5	10
LED_50000	1,000,000	10	24
RandomRBF_50_1E-4	1,000,000	5	10
RandomRBF_50_1E-3	1,000,000	5	10
RandomRBF_0_0	1,000,000	5	10
SEA_50000	1,000,000	2	3
SEA_50	1,000,000	2	3
Stagger3	1,000,000	2	3
Stagger2	1,000,000	2	3
Stagger1	1,000,000	2	3

3.5. Time complexity analysis

Suppose $\mathcal{O}(U), \mathcal{O}(V)$ are the average time complexities of the base classifiers' classification process and update process, respectively. The complexity of the learning process of the proposed method on a sample $(\boldsymbol{x}^{(t)}, y^{(t)})$ is $\mathcal{O}(\max(QU, Q\log Q + QV + Q\log W))$ in which $\mathcal{O}(QU)$ is the time complexity of the classification process, and $\mathcal{O}(QlogQ + QV + Q\log W)$ is the time complexity of the update process. In detail, $\mathcal{O}(QU)$ and $\mathcal{O}(QV)$ are the time complexities of all base classifiers making predictions and updates, respectively, while $\mathcal{O}(QlogQ)$ is for sorting the list of prequential accuracies $\boldsymbol{\rho}$ in the `A_Update` function. Finally, $\mathcal{O}(Q\log W)$ is the time complexity of using ADWIN to detect changes for all base classifiers.

In practice, $\mathcal{O}(logW)$ and $\mathcal{O}(logQ)$ are2 often much smaller than $\mathcal{O}(V)$, so the running time of the update process depends mostly on the time of all base classifiers performing updates. We can see later in the experiments that the CPU running time of the proposed method is statistically comparable to the standard heterogeneous ensemble with no selection method.

4. Experimental studies

4.1. Datasets

We conducted experiments on 50 data streams that are available in OpenML[1] to evaluate the performance of our proposed method. They are commonly used in the data stream literature [9,29,32,33], comprising of both real-world data streams and synthetically generated data streams. Table 1 shows the details of these datasets, and their descriptions can be found in the Supplementary Material.

4.2. Experimental setup

Two common evaluation procedures are often used for the stream setting: (1) *Holdout method* evaluates the learning model on a single holdout set at regular time intervals. This approach is applicable when the dataset has been well divided into train and test sets. However, it cannot provide reliable estimates for non-stationary streams when using a fixed test set. (2) *Prequential method* first predicts the class label for each arrived instance and then uses the ground truth to update the learning model.

For all experiments in this paper, we use the prequential evaluation method since not only is it more suitable for evolving data streams than the holdout method, but it is also the most widely used procedure to evaluate the performance of a classifier under the data stream setting. There are two main steps in this method:

- Predict the class label: At time t, we use our current model to predict the class label $\hat{y}^{(t)}$ for the sample $\boldsymbol{x}^{(t)}$.
- Update the model: if the true class label $y^{(t)}$ of the sample $\boldsymbol{x}^{(t)}$ is revealed, the current model is updated using $(\boldsymbol{x}^{(t)}, y^{(t)})$.

One common assumption in the prequential evaluation method is that the true class label $y^{(t)}$ of the sample $\boldsymbol{x}^{(t)}$ is available before the next sample $\boldsymbol{x}^{(t+1)}$ arrives, so we can update our model with $(\boldsymbol{x}^{(t)}, y^{(t)})$ immediately after predicting $\boldsymbol{x}^{(t)}$. However, this assumption does not hold for many real-world applications in which labels are not instantly obtained after making predictions. Gomes et al. [12] introduce the *delayed setting* to simulate this scenario. In this study, we use the prequential evaluation with two different settings:

- Immediate prequential evaluation: each true label is instantly revealed to the model before the next instance arrives.
- Delayed prequential evaluation: with delay d, the true label of the current sample is presented after the model makes d more predictions from the arrival time of this sample. The delay d is set to 1,000 in our experiments.

We use CPU running time, memory usage, and classification performance to evaluate the experiments. The CPU running time is the summation of training and testing time measured in seconds. To quantify the

[1] https://www.openml.org/

memory consumed by the learning algorithms, we use the RAM-Hours [34], i.e., one GB of RAM deployed for 1 hour corresponds to one RAM-Hour. We use the predictive accuracy to access the classification performance since this is the most frequently used metric in both batch and stream data [39,40]. We report all these measures for both immediate setting and delayed setting.

To statistically compare the results of different stream classification methods, we apply the Friedman test [35] to analyze the difference between each evaluation of multiple methods on multiple datasets. Specifically, the Friedman test checks the null hypothesis that "all methods perform equally". If the null hypothesis is rejected, the Nemenyi post-hoc test is then conducted to compare all pairwise combinations of the methods on multiple datasets. For these tests, we set the significance level to 0.05.

4.3. Benchmark algorithms

To demonstrate the performance of our proposed method for evolving data streams, we compare it with well-known adaptive ensemble algorithms. First, the baseline Majority Vote Ensemble, which combines the votes of all its members to make predictions, is used to show the improvement obtained by HEES. Here, the ADWIN Drift Detector is integrated into the baseline to make it adapted to evolving data streams. We call this ensemble MajorityVoteAdwin. Second, we compare HEES to the state-of-the-art heterogeneous ensemble BLAST. Two versions of BLAST were introduced in [11]: BLAST with Fading Factor (BLAST (FF)) and BLAST with Window (BLAST (Window)). Since the author has shown that the BLAST (FF) works better than BLAST (Window), we only used BLAST (FF) in our experiments. The Fading Factor and the number of active classifiers are set to 0.999 and 1, respectively, as suggested in the original paper. We use the same pool of base classifiers for BLAST, MajorityVoteAdwin, and HEES, which is discussed later in section 5.1.

Furthermore, HEES is compared to some well-known homogeneous ensemble methods, including Bagging and Boosting variants. Bagging inspired methods are represented by Online Bagging with ADWIN (OzaBagAdwin) [22], while Boosting variants being used are Online Boosting with ADWIN (OzaBoostAdwin) and Boosting-like Online Learning Ensemble (BOLE) [10].

Finally, we compare HEES with one single method chosen from the pool of base classifiers shown in Table 2 to demonstrate the advantage of an ensemble method over a single base classifier. Here, we pick the Extremely Fast Decision Trees (EFDT)[43] since it is the best performing single classifier in our experiments (see the Nemenyi test result in Figure S1 in the Supplementary Material).

All the base classifiers and benchmark ensembles are available in the MOA[2] library version 2019.05, and their parameters are set to the defaults in this library if not mentioned.

Table 2. Base classifiers used in the experiments

<table>
<tr><th>Classifier</th><th>Type</th><th colspan="4">Heterogeneous Ensemble</th></tr>
<tr><td>1. SGD (hinge loss)</td><td>SVM</td><td rowspan="4">$HEES_4$</td><td rowspan="6">$HEES_6$</td><td rowspan="8">$HEES_8$</td><td rowspan="10">$HEES_{10}$</td></tr>
<tr><td>2. Hoeffding Tree</td><td>Decision Tree</td></tr>
<tr><td>3. Random Hoeffding Tree</td><td>Decision Tree</td></tr>
<tr><td>4. Perceptron</td><td>Linear Model</td></tr>
<tr><td>5. Naïve Bayes</td><td>Bayesian</td><td rowspan="2"></td></tr>
<tr><td>6. Extremely Fast Decision Tree</td><td>Decision Tree</td></tr>
<tr><td>7. Hoeffding Option Tree</td><td>Option Tree</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>8. Hoeffding Adaptive Tree</td><td>Decision Tree</td></tr>
<tr><td>9. k Nearest Neighbors</td><td>Lazy</td><td colspan="3" rowspan="2"></td></tr>
<tr><td>10. Rule Classifier</td><td>Decision Rules</td></tr>
</table>

*The classifiers are in descending order based on their computational speed

[2] https://moa.cms.waikato.ac.nz/

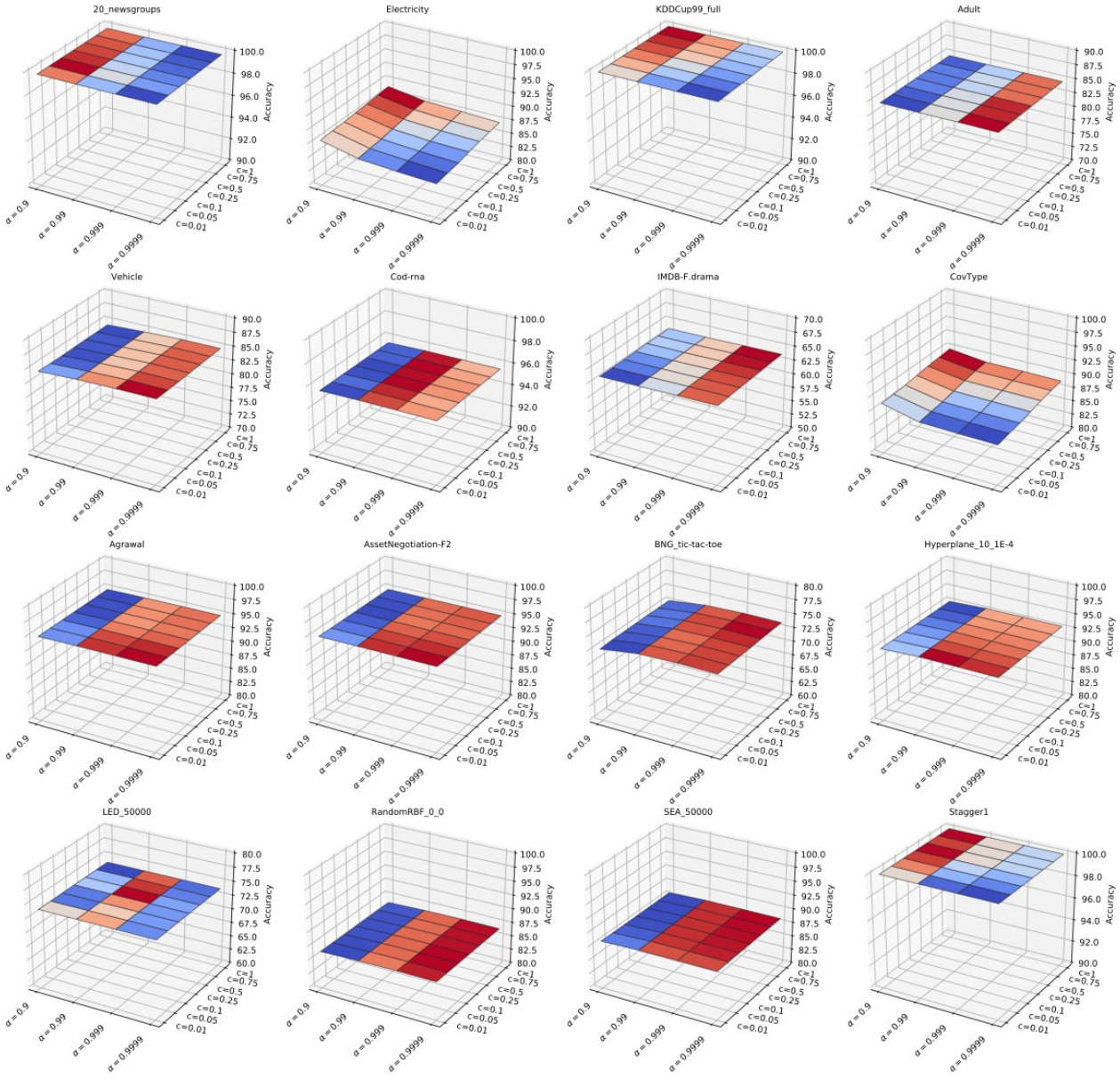


Figure 1. Accuracy (immediate) of HEES with different values of α and c on 16 data streams**.**

5. Results and discussions

5.1. Influence of parameters

a. Fading factor and learning rate

Our proposed method has only two parameters apart from those of ADWIN and of the base classifiers, corresponding to the two selection sub-processes. The first parameter is the constant c in the learning rate

$\eta = \frac{c}{\sqrt{t}}$ used to select the confident base classifiers. The second one is the fading factor α used to control the speed of forgetting of the prequential accuracy in the accurate-candidate selection sub-process.

In Figure 1, we present 3D plots of 16 data sets to illustrate the impact of using different values of c and α. The x-axis shows 4 different values (0.9, 0.99, 0.999, 0.9999) for α, while the y-axis shows 7 different values (0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1) for c. For each pair (α, c), the accuracy of the corresponding model is shown in the z-axis. We do not report the CPU running time and memory for these variants because the two parameters α and c do not affect the time and memory used by HEES. From this figure, it is observed that all the surfaces plotted are almost flat for all data streams, with only two exceptions: Electricity and CovType. In both datasets, HEES obtains the best performance when $c = 1$ and $\alpha = 0.9$. However, the accuracy gap of HEES using other values of c and α is not significant (less than 3%). This observation suggests that the impact of α and c on the proposed method's accuracy is relatively insignificant, or HEES is very robust to different values of its parameters. As a result, our method is easily applicable to many real-world problems since it does not require much effort to tune parameters.

The minor effect of the two parameters in our framework can be explained as follows:

- In section 3.2, we used the parameter c to control the learning rate $\eta^{(t)} = \frac{c}{\sqrt{t}}$ where t is the number of instances seen so far. When t increases, the value of the learning rate $\eta^{(t)}$ becomes smaller; thus, the impact of the learning rate on the thresholds will get smaller. In other words, when we see more incoming instances, the effect of the parameter c on the confident-candidate selection sub-process will become less. Therefore, this parameter has insignificant impact on the performance of the proposed method on almost all datasets, except some small datasets like Electricity, BNG_tic-tac-tor and Covtype.
- In the accurate-candidate selection sub-process (section 3.3), it is clear that if we change the value of the Fading Factor α, the prequential accuracy of each base classifier will be affected. However, this parameter would *not affect the relative rankings* of the base classifiers in terms of prequential accuracy. Since we use these rankings to select base classifiers in the accurate-candidate selection sub-process, the parameter α only has a negligible effect on this selection sub-process. In addition, the mechanism of this sub-process is similar to the BLAST method [11], whose authors also claimed that the Fading Factor α has a small impact on its performance.

In subsequent experiments, we use $c = 0.5$ and $\alpha = 0.999$.

We also compare our method HEES to its variants with fixed reliability thresholds to show the benefit of the threshold estimation module in the confident-candidate selection sub-process. Specifically, we use 5

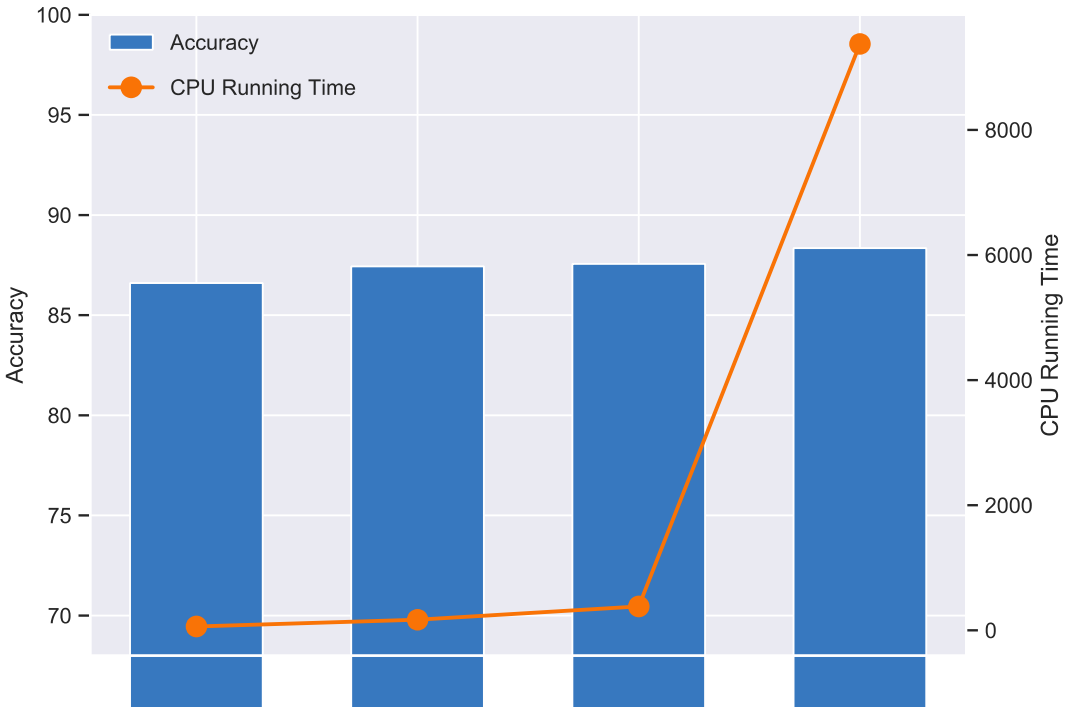


Figure 2. Average accuracy and CPU running time of four variants of HEES based on 50 data streams.

different values for the fixed thresholds: 0.5, 0.6, 0.7, 0.8, 0.9. The Nemenyi test (see Figure S2 in the Supplementary Material) shows that our proposed version of HEES significantly outperforms all its fixed-threshold variants, demonstrating the advantage of dynamically learning the thresholds.

b. Number of base classifiers

We conduct an experiment using 10 different classifiers taken from the MOA library to see the impact of different sets of base classifiers on the accuracy and running time of the proposed method. Unlike homogeneous ensembles, which often need hundreds of base classifiers to obtain good performance, heterogeneous ensembles require only a small number of base classifiers [11,28,37] to achieve a comparable result with a much faster runtime, a key factor in data stream learning because of the sheer volume and high speed of incoming instances. When constructing the base classifier pool for our experiments, we select 10 very popular learning algorithms for data streams. These methods have been used frequently in the data stream literature, especially in some state-of-the-art ensemble systems [11,28,33,37] . Table 2 presents the details of the 10 chosen classifiers. As mentioned before, heterogeneous ensembles use diverse learning algorithms to promote system diversity. Base classifiers of highly diverse learning algorithms usually produce different predictions so that an ensemble of them can improve the classification performance. Here, the 10 learning methods use significantly different approaches to train the base

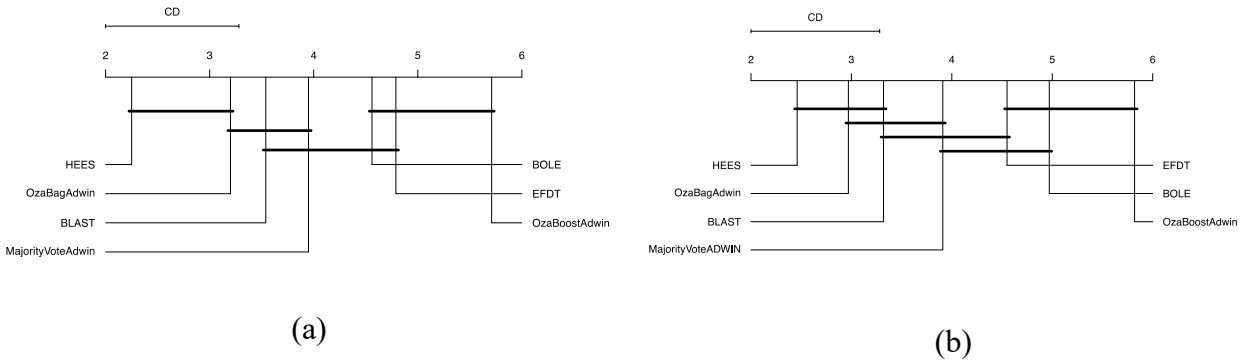
classifiers, and this ensures the generation of diverse outputs.Table 2 also shows four variants of HEES, each of which uses a different set of base classifiers. Particularly, HEES_4, HEES_6, HEES_8, and HEES_{10} use 4, 6, 8, and 10 base classifiers, respectively. The comparison between these 4 variants is designed to illustrate how the different number of base classifiers affects the performance and the runtime of HEES. Figure 2 shows the result of this comparison. Here, we measure the accuracy (immediate setting) and CPU running time of each variant on each dataset and then report the average accuracy and runtime over 50 datasets. We observe a trend where increasing the number of base classifiers results in classification performance improvements. This is attributable to the fact that when having a larger pool of base classifiers, more good candidates can be selected to contribute to the final prediction. However, the runtime also goes up when we increase the number of base classifiers. Based on Figure 2, HEES_8 can be considered the most effective one since it offers a good trade-off between predictive performance and runtime. In particular, HEES_8 has higher accuracy and reasonable runtime in comparison with HEES_6 and HEES_4, while it is much faster than HEES_{10} with only a small trade-off for accuracy. The inclusion of kNN and Rule Classifier makes HEES_{10} extremely slow (hence the jump in running time as seen in Fig. 2 for HEES_{10}). Experiments in [11] show that the runtimes of kNN and Rule Classifier are very high in comparison to other online classifiers, making them not suitable for high-speed data streams.

Further experiments in this work are based on the HEES_8 and referred to solely as HEES. To make a fair comparison, we also use the 8 base classifiers for the BLAST and MajorityVoteAdwin ensemble.

In addition, we also design experiments to investigate the behavior of the proposed method with homogeneous base classifiers. Specifically, we integrate our selection algorithm to Online Bagging ensemble of 100 Hoeffding Trees. We observe that our framework does not work well with Online Bagging because of the following reason. Since all the base classifiers are generated from the Hoeffding Trees algorithm, they behave similarly in our selection process. In the confident-candidate selection sub-process, a plot of the values of the thresholds associated with the first ten base classifiers over time on the Electricity and Covtype datasets (see Figure S3 in Supplementary Material) shows that the first ten base classifiers behave similarly. Likewise, in the accurate-candidate selection sub-process, all the base classifiers also have similar prequential accuracies over time, as shown in Figure S4 in Supplementary Material. The similarity in the behavior of all base classifiers makes the whole selection process inefficient because almost all the time, either all the base classifiers are selected or none of them are selected.

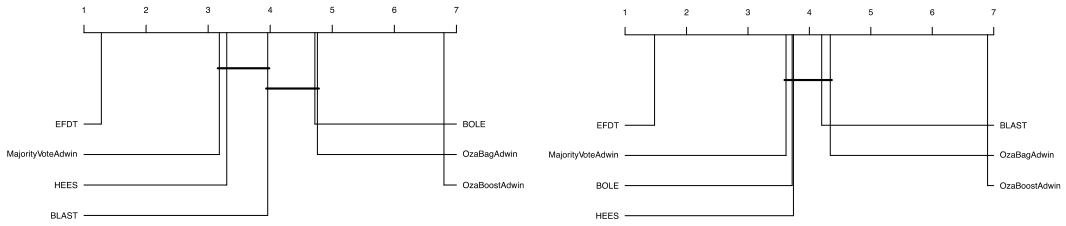
5.2. Accuracy comparison

Tables 3 and 4 show the accuracy of all algorithms under the immediate setting and the delayed setting, respectively. The P-values computed based on these rankings with Friedman test are 7.8e-16 for the immediate setting and 2.2e-16 for the delayed setting; therefore, we rejected the null hypotheses that all methods perform equally. We then conduct the Nemenyi significance tests, and the results are illustrated in Figure 3. The figure shows that the proposed method is significantly better than MajorityVoteAdwin, BOLE, EFDT, and OzaBoostAdwin in both settings. Moreover, HEES statistically outperforms BLAST in the immediate setting. Meanwhile, there is no statistical difference in the pairwise comparison between the proposed method and the remaining benchmark algorithms.



(a) (b)

Figure 3. The result of Nemenyi test based on accuracy. **a.** Immediate setting. **b.** Delayed setting



(a) (b)

Figure 4. The result of Nemenyi test based on **(a)** CPU Running Time and **(b)** RAM-Hour

The proposed method obtains the best average rankings for both settings (2.25 for the immediate setting and 2.46 for the delayed setting), followed by OzaBagAdwin (3.20 for the immediate setting and 2.97 for the delayed setting). The OzaBoostAdwin has the lowest rankings in both settings (5.71 for the immediate setting and 5.82 for the delayed setting), even worse than the single classifier EFDT. There is not much variation with respect to ranking changes while comparing the results between the immediate and delayed settings. The only change is that EFDT swaps rankings with BOLE, suggesting that this Boosting inspired method is not suitable for delayed training.

Under the immediate setting, the proposed method ranks first in terms of accuracy in 13 cases (26%), ranks second in 18 cases (36%). HEES only performs poorly on the BNG_anneal dataset (rank 5th), but the differences between the accuracies of HEES and the benchmark algorithms are not very significant. In comparison with the state-of-the-art heterogeneous ensemble BLAST, our proposed method wins on 39 datasets (78%), performs equally on 1 dataset (2%), and loses on 10 datasets (20%).

For the delayed setting, there are 10 cases (20%) in which the proposed method performs best, and 19 cases (38%) where it ranks second. HEES once again performs poorly on the BNG_anneal dataset (rank 5th). In this setting, HEES also has low rankings on the Covtype dataset (rank 5th) and Electricity dataset (rank 5th). However, HEES still has the best rankings on average like in the immediate setting, suggesting that the proposed method works well in both settings.

5.3. Resources comparison

The CPU running times and the RAM-Hour scores of the benchmark algorithms and the proposed method are presented in Table S1 and S2 (Supplementary Material). We only report the runtime and memory for the immediate setting since the relative rankings of all methods remain the same in the delayed setting. We once again conduct the Friedman tests to compare the CPU running times and the RAM-Hour scores of multiple methods on multiple datasets. In this case, the P-values computed by the tests are less than 2.2e-16 for both cases. Thus, we reject the null hypotheses that all methods perform equally in terms of the consumed resources (time and memory). We then use the Nemenyi post-hoc test for all pairwise comparisons among the 7 methods. The result of this test is illustrated in Figure 4.

In terms of CPU running time, the single classifier EFDT obtains the best average rankings (rank value 1.28), followed by the baseline MajorityVoteAdwin (rank value 3.18) and the proposed method HEES (rank value 3.30). The Nemenyi test shows that EFDT requires significantly less running time in comparison to all other methods. This makes sense, as each ensemble contains multiple single classifiers; thus, its runtime is at least the sum of all base classifiers' runtime. Another observation here is that heterogeneous ensemble seems to run faster than homogeneous ensembles. This is due to the fact that a homogeneous ensemble often includes many more base classifiers than a heterogeneous ensemble. Another reason is that slow base classifiers like *k*NN and Rule Classifier are excluded in the 3 heterogeneous ensembles in our experiments because of the time-accuracy trade-off discussed in section 5.1b. Based on the Nemenyi result, the proposed method is significantly faster than OzaBagAdwin and OzaBoostAdwin, while there is no statistical difference in the pairwise comparison between the proposed method and MajorityVoteAdwin and BLAST. Meanwhile, the OzaBoostAdwin is the slowest method (rank value 6.80) in the experiment, which suggests that it is not suitable for high-speed data streams.

Among all methods, the single classifier EFDT consumes the least memory per hour (rank value 1.48), while OzaBoostAdwin requires the most memory overhead per hour (rank value 6.90). There is no statistical difference in the pairwise comparison among the remaining methods. The relative rankings do not vary much while comparing the results between the runtime and the RAM-Hour score. The only change is that BOLE improves its ranking (rank value 3.72) to surpass HEES (rank value 3.74) and BLAST (rank value 4.2).

5.4. Discussions

The OzaBoostAdwin method performs very poorly in our experiments. It ranks lowest in both performance and resource comparisons. It is not surprising as experiments in many studies [20–22] have shown that Online Boosting performs much worse than Online Bagging. We recommend using other Boosting variants like BOLE instead of the original Online Boosting, especially for high-speed data streams.

The EFDT method is the best base classifier in our experiments. Being a single classifier, it obviously runs much faster and consumes less memory than the other ensemble methods. However, in terms of accuracy, it performs worse than the others except for OzaBoostAdwin (both settings) and BOLE (delayed setting). In addition, EFDT obtains significantly lower rankings than the proposed method in both settings according to the Nemenyi test.

The BOLE ensemble improves Oza and Russell's Online Boosting by weakening the requirements to allow the individual members to vote and employing the DDM method to handle concept drift. Our experiment shows that it achieves higher accuracy in comparison to OzaBoostAdwin. Also, this method is fairly memory efficient. However, it still ranks lower than other ensemble methods in terms of accuracy and runtime.

The MajorityVoteAdwin ensemble is obtained by integrating the ADWIN drift detector to the MajorityVote ensemble, which uses all its base classifiers to vote for each arrived sample. This method rank 4rd in both immediate and delayed setting. We use the MajorityVoteAdwin ensemble as a baseline to our algorithm, aiming to show that a well-chosen subset of base classifiers can outperform the whole ensemble system, and the experiments show that our proposed method obtains better average ranking than this baseline.

BLAST is a heterogeneous ensemble that uses the Online Performance Estimation framework to rank the base classifiers and then select the k best base learners to make predictions. This method is quite similar to the accurate-candidate selection sub-process in our method. However, the number of chosen base classifiers in BLAST is fixed (the authors suggested to use $k = 1$) before the stream starts, and it does not take into

account the confidence in the predictions of base classifiers. Consequently, the flexibility of BLAST is restricted, and it can only achieve average performance in our experiments. In comparison to the heterogeneous ensemble MajorityVoteAdwin in terms of accuracy, BLAST has slightly higher rankings under both immediate setting and delayed setting. By contrast, its rankings is lower than the proposed method and the homogeneous ensemble OzaBagAdwin. Futhermore, the resources consumed by BLAST are more efficient than the three homogeneous ensembles OzaBagAdwin, OzaBoostAdwin, and BOLE.

The homogeneous ensemble OzaBagAdwin is built based on the Online Bagging and the drift detector ADWIN. This ensemble obtained good predictive performance in many studies [11,21]. This is also shown in our experiments, as it ranks 2nd in both the immediate setting and the delayed setting with regard to accuracy. However, its CPU running time and RAM-Hour score are very high compared to the three heterogeneous ensembles, as discussed in section 5.3. For high-speed data streams, one would prefer to use a heterogeneous ensemble as it can obtain comparable accuracy with significantly less runtime and memory, two very important factors in online learning.

The proposed method ranks first in the Nemenyi post hoc test in terms of accuracy for both the immediate setting and the delayed setting. Our method is more flexible than both BLAST and MajorityVoteAdwin as it can dynamically adjust the number of active classifiers selected to vote for the arriving samples. While BLAST always chooses one best classifier according to the Online Performance Estimation Framework to predict samples, and the MajorityVoteAdwin always uses the votes of all base classifiers, the proposed method automatically selects a subset of base classifiers with variable size to make predictions. Moreover, while the BLAST method only considers the recent performance of base classifiers, the proposed method uses both the recent accuracy scores and the confidence in the predictions of the base classifiers to perform the selection process. Consequently, the proposed method obtains better average rankings for both settings in terms of accuracy, and it statistically beats MajorityVoteAdwin (both settings) and BLAST (immediate setting). Also, the time and memory resources consumed by the proposed method is more efficient than BLAST and all the homogeneous ensembles.

6. Conclusions

In this study, we proposed a novel heterogeneous ensemble selection method for evolving data streams by selecting high-accuracy and high-confidence base classifiers. The selection process is divided into two sub-processes: confident-candidate selection and accurate-candidate selection. When classifying a data sample, each sub-process returns a set of base classifiers, and the intersection of these sets is used to make the final vote. On the one hand, the confident-candidate selection is performed by introducing a reliability threshold for each base classifier and following the rule that a classifier is allowed to vote if its confidence score

exceeds this threshold. We introduced a measure to quantify the confidence in the prediction of each candidate. By using this measure, we model the confident-candidate selection sub-process as a classification problem in which the reliability thresholds act as parameters, and then we apply the Stochastic Gradient Descent algorithm to optimize these parameters in an online manner. On the other hand, in the accurate-candidate selection sub-process, we employ the prequential accuracy to estimate the recent performance of each base classifier and then use this score to rank the candidates. A set of base classifiers with the highest rankings are returned by this sub-process. The size of this set is chosen to be equal to the size of the set returned by the confident-candidate selection sub-process. Since the number of base classifiers chosen to vote can vary for each arrived sample, our selection method is very flexible in comparison to other heterogeneous ensembles, e.g., BLAST or MajorityVoteAdwin.

We compare HEES against state of the art ensembles over 50 data streams from both real-world applications and synthetic data generators. We use two different settings to evaluate the performance of each ensemble: immediate prequential setting and delayed prequential setting. The experiments show that not only does the proposed method achieved high predictive performance in both settings, but it also uses very little time and memory resources, two crucially important factors in the data stream context. Furthermore, we evaluated how the parameters affect the performance of the proposed method. There are two parameters that control the performance of HEES: the constant c in the learning rate $\eta = \frac{c}{\sqrt{t}}$ controlling the confident-candidate selection sub-process and the fading factor α controlling the accurate-candidate selection sub-process. We experimentally demonstrated that the proposed method is very robust to these parameters, making it highly applicable to many different data streams.

In future work, a plausible approach to improve HEES is to speed up the convergence in the confident-candidate selection sub-process by using a second-order algorithm for online convex optimization, such as the Stochastic Quasi-Newton Method. We are also interested in investigating how to develop an active learning strategy for our framework to tackle various real-world problems, including delayed streams where labels are not readily available. Another possibility is to implement a parallel version of HEES to resolve the issue of high computation cost in many big data streams.

Table 3. Accuracy – immediate setting

Dataset name	OzaBagAdwin	OzaBoostAdwin	BOLE	BLAST	EFDT	MajorityVoteAdwin	HEES
20_newsgroups	99.5497 (5)	79.9415 (7)	99.6800 (2)	99.5727 (4)	98.3428 (6)	99.6602 (3)	99.7520 (1)
Electricity	84.3485 (6)	88.7579 (2)	90.8280 (1)	85.1209 (4)	80.6431 (7)	84.4986 (5)	87.0101 (3)
KDDCup99_full	99.9470 (4)	30.0149 (7)	99.9803 (1)	99.9630 (3)	99.9053 (6)	99.9450 (5)	99.9690 (2)
Adult	84.7058 (1)	81.0245 (7)	82.2120 (6)	84.3147 (4)	83.8930 (5)	84.5788 (3)	84.6915 (2)
Vehicle	84.2096 (4)	79.9357 (7)	81.3718 (6)	84.4846 (2)	82.8709 (5)	84.2542 (3)	84.8510 (1)
Cod-rna	95.5766 (1)	95.2189 (4)	94.8686 (6)	95.0698 (5)	94.7104 (7)	95.4323 (3)	95.5676 (2)
IMDB-F.drama	63.0794 (5)	49.0212 (7)	55.7828 (6)	63.6128 (3)	63.2175 (4)	63.7758 (1)	63.7236 (2)
Covtype	84.7394 (5)	92.4809 (2)	92.6015 (1)	87.3579 (4)	84.6678 (7)	84.6709 (6)	88.4433 (3)
Agrawal	94.9480 (3)	91.3200 (7)	92.3598 (5)	94.9616 (2)	94.9662 (1)	91.8785 (6)	94.9203 (4)
AssetNegotiation-F2	94.8739 (4)	92.4566 (7)	93.1513 (6)	94.8781 (2.5)	94.8781 (2.5)	94.8582 (5)	94.8793 (1)
AssetNegotiation-F3	94.8030 (4)	92.0472 (7)	92.4183 (6)	94.8060 (2)	94.8034 (3)	94.4257 (5)	94.8070 (1)
AssetNegotiation-F4	94.7081 (4)	91.9490 (7)	93.2271 (6)	94.7128 (1)	94.7103 (3)	94.6968 (5)	94.7117 (2)
BNG_tic-tac-toe	74.0639 (1)	71.4728 (7)	73.3882 (6)	73.6016 (4)	73.7438 (2)	73.5051 (5)	73.7413 (3)
BNG_vote	96.5324 (2)	96.1342 (6)	95.6924 (7)	96.4874 (3)	96.4645 (4)	96.4256 (5)	96.5538 (1)
BNG_trains	94.2126 (3)	90.5996 (7)	94.5210 (1)	93.9019 (4)	93.8651 (5)	93.4647 (6)	94.3701 (2)
BNG_soybean	90.2813 (1)	88.9955 (6)	90.0764 (5)	90.1310 (4)	86.7306 (7)	90.2248 (2)	90.2079 (3)
BNG_mushroom	98.9510 (2)	98.7342 (7)	98.7764 (6)	98.9140 (4)	98.9246 (3)	98.8255 (5)	98.9634 (1)
BNG_kr-vs-kp_small	95.2423 (3)	93.8942 (6)	93.4371 (7)	95.2270 (4)	95.2751 (2)	94.9549 (5)	95.2980 (1)
BNG_segment	86.4928 (5)	84.4816 (7)	86.1050 (6)	86.7665 (2)	86.8029 (1)	86.5986 (4)	86.7249 (3)
BNG_ionosphere	95.2660 (2)	94.5796 (6)	96.3940 (1)	94.6508 (5)	94.5173 (7)	94.7751 (4)	95.2053 (3)
BNG_credit-g	78.2546 (1)	70.7416 (7)	76.5613 (6)	77.6404 (5)	77.6606 (4)	77.7167 (3)	78.1084 (2)
BNG_SPECT	84.9970 (1)	80.6309 (7)	82.0841 (6)	84.8693 (3)	84.7103 (5)	84.8150 (4)	84.9402 (2)
BNG_solar-flare	77.6641 (4)	71.3557 (7)	76.0827 (6)	77.6443 (5)	77.6665 (3)	77.6787 (2)	77.6800 (1)
BNG_page-blocks	90.4195 (5)	90.6088 (3)	89.8224 (7)	90.6393 (2)	90.6606 (1)	90.1804 (6)	90.5336 (4)
BNG_lymph	90.7837 (2)	88.2649 (7)	88.6689 (6)	90.4359 (4)	90.4381 (3)	90.4139 (5)	90.7974 (1)
BNG_labor	94.6272 (1)	93.9364 (7)	94.3634 (4)	94.2506 (5)	94.0453 (6)	94.4011 (3)	94.5580 (2)
BNG_hepatitis	92.0154 (1)	87.9413 (7)	90.1394 (6)	91.4911 (4)	91.3487 (5)	91.7023 (3)	91.7896 (2)
BNG_heart-statlog	88.7972 (1)	85.2580 (7)	87.5640 (6)	88.0921 (4)	88.0540 (5)	88.4014 (3)	88.5871 (2)
BNG_heart-c	88.2121 (3)	85.7231 (7)	87.5648 (6)	88.1843 (4)	88.2790 (2)	88.1045 (5)	88.2904 (1)
BNG_dermatology	98.5767 (3)	97.5513 (6)	97.9874 (5)	98.6070 (1)	97.1862 (7)	98.5645 (4)	98.6012 (2)
BNG_credit-a	88.3241 (2)	85.6891 (7)	86.5707 (6)	87.9507 (4)	87.9261 (5)	88.2832 (3)	88.3288 (1)
BNG_spambase	66.5615 (1)	59.8777 (6)	56.8125 (7)	66.4993 (3)	66.4295 (5)	66.4370 (4)	66.5413 (2)
BNG_optdigits	90.6541 (6)	93.4566 (1)	92.1487 (2)	91.7469 (3)	84.7050 (7)	90.7869 (5)	91.6997 (4)
BNG_anneal	94.8361 (6)	95.3469 (2)	95.7623 (1)	95.0877 (4)	95.1228 (3)	94.4914 (7)	94.9965 (5)
BNG_wine	93.9554 (1)	92.8401 (6)	92.4778 (7)	93.4094 (4)	93.2879 (5)	93.5994 (3)	93.8587 (2)
BNG_waveform-5000	88.6755 (1)	82.2002 (7)	86.2666 (6)	87.3046 (4)	86.7596 (5)	88.4340 (2)	88.3767 (3)
BNG_sonar	82.3031 (1)	74.5457 (7)	80.2281 (5)	80.7370 (4)	79.9985 (6)	81.4782 (3)	81.8965 (2)

BNG_satimage	84.0677 (4)	84.5127 (1)	83.5110 (5)	82.7787 (6)	82.6513 (7)	84.4883 (2)	84.3843 (3)
BNG_cmc	52.9062 (7)	54.0093 (1)	53.3890 (2)	53.1539 (4)	53.0147 (6)	53.0997 (5)	53.2353 (3)
Hyperplane_10_1E-4	90.0683 (4)	80.5298 (7)	87.8308 (6)	93.8233 (1)	88.9152 (5)	92.2226 (3)	92.9739 (2)
Hyperplane_10_1E-3	88.3785 (4)	74.7480 (7)	86.7038 (5)	91.8482 (1)	81.5866 (6)	89.8474 (3)	90.9733 (2)
LED_50000	73.9247 (2)	65.5604 (7)	73.9381 (1)	73.7455 (5)	69.8808 (6)	73.8899 (4)	73.8981 (3)
RandomRBF_50_1E-4	59.3582 (5)	62.3468 (4)	62.6365 (3)	64.1603 (2)	51.2695 (7)	58.6528 (6)	65.5343 (1)
RandomRBF_50_1E-3	51.9859 (1)	38.5476 (6)	50.4046 (2)	46.0909 (5)	31.8711 (7)	47.8790 (4)	48.8570 (3)
RandomRBF_0_0	87.6844 (2)	87.3722 (3)	88.5122 (1)	86.0249 (6)	86.0732 (5)	85.1414 (7)	86.5998 (4)
SEA_50000	88.3789 (4)	83.6650 (7)	88.2097 (5)	88.4505 (2)	84.8242 (6)	88.4936 (1)	88.4440 (3)
SEA_50	88.5785 (5)	83.3069 (7)	88.5975 (4)	88.9968 (3)	84.9588 (6)	89.2349 (1)	89.1715 (2)
Stagger3	99.9996 (4)	99.9992 (7)	99.9996 (4)	99.9996 (4)	99.9996 (4)	99.9996 (4)	99.9997 (1)
Stagger2	99.9988 (6)	99.9990 (3.5)	99.9993 (1)	99.9990 (3.5)	99.9985 (7)	99.9990 (3.5)	99.9990 (3.5)
Stagger1	99.9984 (7)	99.9996 (1)	99.9986 (6)	99.9989 (5)	99.9990 (3)	99.9990 (3)	99.9990 (3)
Average Ranking	3.2	5.71	4.56	3.54	4.79	3.95	**2.25**
Mean ± Std	87.2303 ±11.6737	82.6725 ±15.4537	86.4342 ±12.3553	87.2439 ±11.8804	85.6651 ±13.5718	86.9977 ±11.9088	**87.5609 ±11.6018**

Table 4. Accuracy – delayed setting

Dataset name	**OzaBagAdwin**	**OzaBoostAdwin**	**BOLE**	**BLAST**	**EFDT**	**MajorityVoteAdwin**	**HEES**
20_newsgroups	94.0024 (3)	75.3615 (7)	92.8483 (5)	94.5370 (1)	94.0583 (2)	92.7764 (6)	93.2664 (4)
Electricity	74.2530 (1)	72.3822 (3)	61.6041 (7)	71.5359 (4)	73.8604 (2)	66.1175 (6)	70.5385 (5)
KDDCup99_full	98.8968 (3)	29.0903 (7)	98.5026 (5)	99.2838 (2)	99.5231 (1)	98.4609 (6)	98.7061 (4)
Adult	84.6892 (1)	81.0209 (7)	81.9468 (6)	84.2649 (4)	83.9325 (5)	84.4969 (3)	84.6390 (2)
Vehicle	84.2312 (4)	79.9442 (7)	81.2577 (6)	84.5808 (2)	82.9434 (5)	84.2866 (3)	84.8741 (1)
Cod-rna	95.5700 (1)	95.2193 (4)	94.8438 (6)	95.0940 (5)	94.7029 (7)	95.4441 (3)	95.5672 (2)
IMDB-F.drama	62.7649 (5)	48.2134 (7)	54.5468 (6)	63.4603 (2)	63.1343 (4)	63.6596 (1)	63.4320 (3)
Covtype	80.4006 (2)	81.5064 (1)	54.0266 (7)	78.8272 (3)	77.9029 (4)	72.1413 (6)	73.1937 (5)
Agrawal	94.9468 (3)	91.3159 (7)	92.3709 (5)	94.9687 (2)	94.9701 (1)	91.8614 (6)	94.9293 (4)
AssetNegotiation-F2	94.8744 (4)	92.4504 (7)	93.1672 (6)	94.8792 (2)	94.8787 (3)	94.8604 (5)	94.8793 (1)
AssetNegotiation-F3	94.8018 (4)	92.0303 (7)	92.4089 (6)	94.8063 (2)	94.8027 (3)	94.4233 (5)	94.8068 (1)
AssetNegotiation-F4	94.7073 (4)	91.9579 (7)	93.2181 (6)	94.7124 (1)	94.7090 (3)	94.6958 (5)	94.7104 (2)
BNG_tic-tac-toe	74.0343 (1)	71.6989 (7)	73.2185 (6)	73.5912 (4)	73.5730 (5)	73.6459 (3)	73.6955 (2)
BNG_vote	96.5265 (1)	96.0691 (6)	95.6878 (7)	96.4773 (3)	96.4681 (4)	96.4066 (5)	96.5227 (2)
BNG_trains	94.2060 (3)	90.6086 (7)	94.5054 (1)	93.9393 (4)	93.8739 (5)	93.4570 (6)	94.3626 (2)
BNG_soybean	90.2778 (1)	89.0070 (6)	90.0951 (5)	90.1387 (4)	86.7397 (7)	90.2223 (2)	90.1985 (3)
BNG_mushroom	98.9510 (2)	98.7383 (7)	98.7793 (6)	98.9097 (4)	98.9209 (3)	98.8265 (5)	98.9643 (1)

BNG_kr-vs-kp_small	95.2408 (3)	93.9135 (6)	93.4673 (7)	95.2263 (4)	95.2812 (2)	94.9629 (5)	95.3045 (1)
BNG_segment	86.4942 (5)	84.4770 (7)	86.1200 (6)	86.7662 (2)	86.8013 (1)	86.6062 (4)	86.7239 (3)
BNG_ionosphere	95.2576 (2)	94.5783 (6)	96.3846 (1)	94.6505 (5)	94.5309 (7)	94.7835 (4)	95.2042 (3)
BNG_credit-g	78.2472 (1)	70.7168 (7)	76.5469 (6)	77.6535 (5)	77.6549 (4)	77.7169 (3)	78.1063 (2)
BNG_SPECT	84.6349 (5)	80.6923 (7)	82.0728 (6)	84.8444 (2)	84.7109 (4)	84.8102 (3)	84.9361 (1)
BNG_solar-flare	77.6624 (4)	71.3787 (7)	76.0667 (6)	77.6321 (5)	77.6638 (3)	77.6760 (1)	77.6752 (2)
BNG_page-blocks	90.4226 (5)	90.6330 (3)	89.8183 (7)	90.6439 (2)	90.6547 (1)	90.1783 (6)	90.5477 (4)
BNG_lymph	90.7838 (2)	88.2864 (7)	88.6525 (6)	90.4597 (3)	90.4298 (4)	90.4061 (5)	90.7993 (1)
BNG_labor	94.6284 (1)	93.9155 (7)	94.3555 (4)	94.2445 (5)	94.0563 (6)	94.4182 (3)	94.5579 (2)
BNG_hepatitis	92.0113 (1)	87.9235 (7)	90.1304 (6)	91.4978 (4)	91.3501 (5)	91.7042 (3)	91.7921 (2)
BNG_heart-statlog	88.8021 (1)	85.2636 (7)	87.4879 (6)	88.0866 (4)	88.0364 (5)	88.3980 (3)	88.5815 (2)
BNG_heart-c	88.2146 (3)	85.7466 (7)	87.5824 (6)	88.1988 (4)	88.2778 (2)	88.1208 (5)	88.3000 (1)
BNG_dermatology	98.5791 (3)	97.5428 (6)	97.9845 (5)	98.6057 (1)	97.1843 (7)	98.5581 (4)	98.6025 (2)
BNG_credit-a	88.3285 (1)	85.6916 (7)	86.6195 (6)	87.9363 (4)	87.9112 (5)	88.2561 (3)	88.3240 (2)
BNG_spambase	66.5681 (1)	59.9605 (6)	56.8422 (7)	66.4979 (3)	66.4323 (5)	66.4526 (4)	66.5488 (2)
BNG_optdigits	90.6578 (6)	93.4634 (1)	92.1441 (2)	91.7622 (3)	84.6937 (7)	90.7848 (5)	91.7193 (4)
BNG_anneal	94.8381 (6)	95.3412 (2)	95.7514 (1)	95.0783 (4)	95.1166 (3)	94.4916 (7)	95.0006 (5)
BNG_wine	93.9617 (1)	92.8295 (6)	92.4734 (7)	93.4021 (4)	93.2927 (5)	93.5857 (3)	93.8612 (2)
BNG_waveform-5000	88.6676 (1)	82.2098 (7)	86.2675 (6)	87.3264 (4)	86.7580 (5)	88.4303 (2)	88.3649 (3)
BNG_sonar	82.2996 (1)	74.5703 (7)	80.2744 (5)	80.7476 (4)	79.9782 (6)	81.4602 (3)	81.8719 (2)
BNG_satimage	84.0674 (4)	84.5183 (1)	83.5056 (5)	82.7361 (6)	82.6445 (7)	84.4900 (2)	84.3956 (3)
BNG_cmc	52.9266 (7)	54.0703 (1)	53.3723 (2)	53.2894 (4)	52.9947 (6)	53.1126 (5)	53.3630 (3)
Hyperplane_10_1E-4	90.0614 (4)	80.4910 (7)	87.7981 (6)	93.7854 (1)	88.9075 (5)	92.2184 (3)	92.9585 (2)
Hyperplane_10_1E-3	88.0007 (4)	74.6396 (7)	86.3944 (5)	91.1087 (1)	81.3541 (6)	89.3756 (3)	90.3773 (2)
LED_50000	73.9154 (2)	65.4797 (7)	73.9232 (1)	73.7171 (5)	69.7891 (6)	73.8633 (4)	73.8654 (3)
RandomRBF_50_1E-4	57.4138 (4)	55.8905 (6)	60.3925 (3)	60.4054 (2)	46.8032 (7)	56.7975 (5)	63.2767 (1)
RandomRBF_50_1E-3	29.7050 (2)	25.7519 (7)	31.2965 (1)	27.7276 (5)	27.6066 (6)	28.4070 (4)	28.6005 (3)
RandomRBF_0_0	87.6849 (2)	87.3710 (3)	88.4915 (1)	86.0423 (6)	86.0851 (5)	85.1371 (7)	86.5920 (4)
SEA_50000	88.3518 (4)	83.6253 (7)	88.1943 (5)	88.4316 (2)	84.7963 (6)	88.4752 (1)	88.4314 (3)
SEA_50	88.5362 (5)	83.2699 (7)	88.5684 (4)	88.9584 (3)	84.9324 (6)	89.1859 (1)	89.1362 (2)
Stagger3	99.9996 (5)	99.9993 (7)	99.9997 (2.5)	99.9997 (2.5)	99.9996 (5)	99.9996 (5)	99.9998 (1)
Stagger2	99.9990 (5)	99.9988 (6)	99.9992 (4)	99.9993 (3)	99.9987 (7)	99.9994 (1.5)	99.9994 (1.5)
Stagger1	99.9985 (4.5)	99.9996 (1)	99.9984 (7)	99.9985 (4.5)	99.9985 (4.5)	99.9988 (2)	99.9985 (4.5)
Average Ranking	2.97	5.82	4.97	3.32	4.55	3.91	**2.46**
Mean ± Std	86.3019 ± 13.4755	81.6171 ± 16.5411	84.4401 ± 14.7469	86.2293 ± 13.6211	85.1144 ± 14.2148	85.7729 ± 13.8250	**86.3021 ± 13.5163**

Appendix

Lemma A1: $\mathbb{E}\left[\left|\frac{\partial \mathcal{L}_q}{\partial \theta_q}\right|^2\right]$ is bounded in [0, 1].

Proof.

Since $r_q \in \{0,1\}$, and $0 \leq s_q \leq 1$, we have:

$$-1 \leq \frac{\partial \mathcal{L}_q}{\partial \theta_q} = r_q - s_q \leq 1$$

$$\Rightarrow 0 \leq \left|\frac{\partial \mathcal{L}_q}{\partial \theta_q}\right|^2 \leq 1$$

$$\Rightarrow 0 \leq \mathbb{E}\left[\left|\frac{\partial \mathcal{L}_q}{\partial \theta_q}\right|^2\right] \leq 1 \blacksquare$$

Lemma A2: If we choose $\theta_q^1 \in [0,1]$ and $\eta^{(t)} = \frac{c}{\sqrt{t}}$ where c is predefined positive number, then $\theta_q^{(t+1)}$ is bounded in $[-\eta^{(t)}, 1 + \eta^{(t)}]$ $\forall t = 1,2, \ldots$

Proof.

For $t = 2$, we have $r_q(\boldsymbol{x}^{(1)}) \in \{0,1\}$, and $0 < s_q(\boldsymbol{x}^{(t)}) < 1$. Therefore

$$-\eta^{(1)} \leq \theta_q^{(2)} = \theta_q^{(1)} - \eta^{(1)}(r_q(\boldsymbol{x}^{(1)}) - s_q(\boldsymbol{x}^{(1)})) \leq 1 + \eta^{(1)}.$$

Assume the statement holds for t. We will show that it also holds for $t + 1$.

Denoting $r_q = r_q(\boldsymbol{x}^{(t)}), e_q = e_q(\boldsymbol{x}^{(t)}), s_q = s_q(\boldsymbol{x}^{(t)}, \theta_q)$, we consider three cases:

Case 1: $1 < \theta_q^{(t)} \leq 1 + \eta^{(t-1)}$

Note that $0 \leq e_q \leq 1$, hence $e_q < \theta_q^{(t)}$, this means $s_q < 0.5$ and $C_q$is not selected. We only update the threshold if $r_q = 1$. In this case:

$$\theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)}(r_q - s_q) \leq \theta_q^{(t)} - \eta^{(t)}(1 - 0.5)$$

$$\leq 1 + \eta^{(t-1)} - 0.5\eta^{(t)} = 1 + \frac{c}{\sqrt{t-1}} - \frac{0.5c}{\sqrt{t}}$$

$$= 1 + \eta^{(t)} + \frac{c}{\sqrt{t-1}} - \frac{1.5c}{\sqrt{t}} < 1 + \eta^{(t)}$$

Where the last inequality uses: $\frac{c}{\sqrt{t-1}} - \frac{1.5c}{\sqrt{t}} < 0 \ \ \forall t \geq 2$

Case 2: $0 \leq \theta_q^{(t)} \leq 1$

- If $r_q = 0$, we update the threshold if $\mathbb{I}\left[e_q > \theta_q^{(t)}\right] = 1$. In this case, $s_q > 0.5$, then

  $\theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)}(r_q - s_q)$

  $\leq \theta_q^{(t)} - \eta^{(t)}(0 - 1) \leq 1 + \eta^{(t)}$

  and $\theta_q^{(t+1)} \geq \theta_q^{(t)} - \eta^{(t)}(0 - 0.5) \geq 0$

- If $r_q = 1$, we update the threshold if $\mathbb{I}\left[e_q > \theta_q^{(t)}\right] = 0$. In this case, $s_q \leq 0.5$, then

  $\theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)}(r_q - s_q)$

  $\leq \theta_q^{(t)} - \eta^{(t)}(1 - 0.5) \leq 1$

  and $\theta_q^{(t+1)} \geq \theta_q^{(t)} - \eta^{(t)}(1 - 0) \geq -\eta^{(t)}$

Therefore $-\eta^{(t)} \leq \theta_q^{(t+1)} \leq 1 + \eta^{(t)}$

Case 3: $-\eta^{(t)} \leq \theta_q^{(t)} < 0$

We have $0 \leq e_q \leq 1$, hence $e_q > \theta_q^{(t)}$, this means that $s_q > 0.5$ and C_q is selected. We only update the threshold if $r_q = 0$. In this case:

$\theta_q^{(t+1)} = \theta_q^{(t)} - \eta^{(t)}(r_q - s_q) \geq \theta_q^{(t)} - \eta^{(t)}(0 - 0.5)$

$\geq -\eta^{(t-1)} + 0.5\eta^{(t)} > -\eta^{(t)}$.

By induction, we have $-\eta^{(t)} \leq \theta_q^{(t+1)} \leq 1 + \eta^{(t)}\ \forall t \geq 1$■

Lemma A3: Suppose $\theta_q^* = \underset{\theta_q \in \mathbb{R}}{\operatorname{argmin}}\ \mathcal{L}_q(\boldsymbol{x}^{(t)}, \theta_q)$. Choose $\eta^{(t)} = \frac{c}{\sqrt{t}}$ where c is a predefined positive number. We have $\mathbb{E}\left(\left|\theta_q^{(t)} - \theta_q^*\right|^2\right) \leq |1 + 2c|^2\ \forall t \geq 1$.

Proof.

If $\theta_q > 1$, we have $e_q \leq 1 < \theta_q$, so C_q is not selected. In this case, we make wrong selection if $r_q = 1$, hence the loss $\mathcal{L}_q(\boldsymbol{x}^{(t)}, \theta_q) = -\log(\text{sigmoid}(e_q - \theta_q))$. We can see that the loss $\mathcal{L}_q$ get larger if θ_q increases.

Similarly, if $\theta_q < 0$, the loss $\mathcal{L}_q$ get larger if θ_q decreases.

Therefore, if there is a $\theta_q^* \in \mathbb{R}$ that minimizes $\mathcal{L}_q\left(\boldsymbol{x}^{(t)}, \theta_q\right)$, it will be bounded in $[-\epsilon, 1+\epsilon]$, where ϵ is an arbitrary positive number. Choose $\epsilon = c$, we obtain $\theta_q^* \in [-c, 1+c]$.

Now note that $\eta^{(t)} \leq c \; \forall t \geq 1$, combining with the Lemma 2, we have $\theta_q^{(t)} \in [-c, 1+c] \; \forall t \geq 1$. Then

$$-1 - 2c \leq \theta_q^{(t)} - \theta_q^* \leq 1 + 2c$$

$$\Rightarrow \left|\theta_q^{(t)} - \theta_q^*\right|^2 \leq |1 + 2c|^2$$

$$\Rightarrow \mathbb{E}\left(\left|\theta_q^{(t)} - \theta_q^*\right|^2\right) \leq |1 + 2c|^2 \; \forall t \geq 1 \blacksquare$$

References

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

[2] H.M. Gomes, J.P. Barddal, F. Enembreck, A. Bifet, A survey on ensemble learning for data stream classification, ACM Computing Surveys (CSUR). 50 (2017) 23.

[3] P. Domingos, G. Hulten, Mining high-speed data streams, in: Kdd, 2000: p. 4.

[4] R.M.O. Cruz, D.V.R. Oliveira, G.D.C. Cavalcanti, R. Sabourin, FIRE-DES++: Enhanced online pruning of base classifiers for dynamic ensemble selection, Pattern Recognition. 85 (2019) 149–160.

[5] R.M.O. Cruz, R. Sabourin, G.D.C. Cavalcanti, T.I. Ren, META-DES: A dynamic ensemble selection framework using meta-learning, Pattern Recognition. 48 (2015) 1925–1935.

[6] B. Krawczyk, M. Galar, M. Woźniak, H. Bustince, F. Herrera, Dynamic ensemble selection for multi-class classification with one-class classifiers, Pattern Recognition. 83 (2018) 34–51.

[7] J. Gama, R. Sebastião, P.P. Rodrigues, Issues in evaluation of stream learning algorithms, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009: pp. 329–338.

[8] J. Gama, R. Sebastião, P.P. Rodrigues, On evaluating stream learning algorithms, Machine Learning. 90 (2013) 317–346.

[9] A. Bifet, G. Holmes, 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.

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

[11] J.N. van Rijn, G. Holmes, B. Pfahringer, J. Vanschoren, The online performance estimation framework: heterogeneous ensemble learning for data streams, Machine Learning. 107 (2018) 149–176.

[12] H.M. Gomes, A. Bifet, J. Read, J.P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem, Adaptive random forests for evolving data stream classification, Machine Learning. 106 (2017) 1469–1495.

[13] T.T. Nguyen, M.T. Dang, A.V. Luong, A.W.C. Liew, T. Liang, J. McCall, Multi-label classification via incremental clustering on an evolving data stream, Pattern Recognition. 95 (2019) 96-113.

[14] T.T. Nguyen, T.T.T. Nguyen, A.V. Luong, Q.V.H. Nguyen, A.W.C. Liew, B. Stantic, Multi-label classification via label correlation and first order feature dependance in a data stream, Pattern Recognition. 90 (2019) 35-51.

[15] A. Osojnik, P. Panov, S. Džeroski, Multi-label classification via multi-target regression on data streams, Machine Learning. 106 (2017) 745–770.

[16] G.H. John, P. Langley, Estimating continuous distributions in Bayesian classifiers, in: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1995: pp. 338–345.

[17] J. Beringer, E. Hüllermeier, Efficient instance-based learning on data streams, Intelligent Data Analysis. 11 (2007) 627–650.

[18] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychological Review. 65 (1958) 386.

[19] L. Bottou, Stochastic learning, in: Summer School on Machine Learning, Springer, 2003: pp. 146–168.

[20] N.C. Oza, Online bagging and boosting, in: 2005 IEEE International Conference on Systems, Man and Cybernetics, SMC 2005: pp. 2340–2345.

[21] N.C. Oza, S. Russell, Experimental comparisons of online and batch versions of bagging and boosting, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001: pp. 359–364.

[22] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavaldà, New ensemble methods for evolving data streams, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009: pp. 139–148.

[23] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Brazilian Symposium on Artificial Intelligence, Springer, 2004: pp. 286–295.

[24] X.C. Pham, M.T. Dang, S.V. Dinh, S. Hoang, T.T. Nguyen, A.W.C. Liew, Learning from data stream based on random projection and Hoeffding tree classifier, in: 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA), IEEE, 2017: pp. 1–8.

[25] Y. Xia, C. Liu, B. Da, F. Xie, A novel heterogeneous ensemble credit scoring model based on bstacking approach, Expert Systems with Applications. 93 (2018) 182–199.

[26] T.T. Nguyen, M.P. Nguyen, X.C. Pham, A.W.C. Liew, Heterogeneous classifier ensemble with fuzzy rule-based meta learner, Information Sciences. 422 (2018) 144–160.

[27] J.-F. Connolly, E. Granger, R. Sabourin, Evolution of heterogeneous ensembles through dynamic particle swarm optimization for video-based face recognition, Pattern Recognition. 45 (2012) 2460–2477.

[28] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, L. Wan, Heterogeneous ensemble for feature drifts in data streams, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2012: pp. 1–12.

[29] J.N. van Rijn, G. Holmes, B. Pfahringer, 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.

[30] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, SIAM, 2007: pp. 443–448.

[31] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, R. Morales-Bueno, Early drift detection method, in: Fourth International Workshop on Knowledge Discovery from Data Streams, 2006: pp. 77–86.

[32] J. Read, A. Bifet, B. Pfahringer, G. Holmes, Batch-incremental versus instance-incremental learning in dynamic and evolving data, in: International Symposium on Intelligent Data Analysis, Springer, 2012: pp. 313–323.

[33] J.N. van Rijn, G. Holmes, B. Pfahringer, J. Vanschoren, Algorithm selection on data streams, in: International Conference on Discovery Science, Springer, 2014: pp. 325–336.

[34] A. Bifet, G. Holmes, B. Pfahringer, E. Frank, Fast perceptron decision tree learning from evolving data streams, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2010: pp. 299–310.

[35] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research. 7 (2006) 1–30.

[36] H.M. Gomes, J. Read, A. Bifet, Streaming Random Patches for Evolving Data Stream Classification, in: 2019 IEEE 19th Int. Conf. Data Min., 2019.

[37] M.M. Idrees, L.L. Minku, F. Stahl, A. Badii, A heterogeneous online learning ensemble for non-stationary environments, Knowledge-Based Syst. 188 (2020) 104983.

[38] B. Krawczyk, A. Cano, Online ensemble learning with abstaining classifiers for drifting and noisy data streams. Applied Soft Computing, (2018) 68, 677-692.

[39] A. Tharwat, Classification assessment methods. *Applied Computing and Informatics*, in press, (2018).

[40] A. Bifet , R. Gavaldà, G. Holmes , B. Pfahringer , Machine learning for data streams: with practical examples in MOA, MIT Press, 2018.

[41] D.P. Bertsekas , Incremental gradient, subgradient, and proximal methods for convex optimization: A survey, Optim. Mach. Learn. 1-38 (2011) 3 2010 . Bertsekas, D. P. (2011). Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. Optimization for Machine Learning, 2010(1-38), 3.

[42] E. Hazan , Introduction to online convex optimization, Found. Trend Optim. 2 (3–4) (2016) 157–325.

[43] C. Manapragada , G.I. Webb , M. Salehi , Extremely fast decision tree, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1953–1962.

[44] T. Murata , H. Ishibuchi , T. Nakashima , M. Gen , Fuzzy partition and input selection by genetic algorithms for designing fuzzy rule-based classification systems, in: International Conference on Evolutionary Programming, Springer, Berlin, Heidelberg, 1998, pp. 407–416.