Fast Switch Naïve Bayes to Avoid Redundant Update for Concept Drift Learning

Anjin Liu
Centre for Artificial Intelligence,
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
Anjin.Liu@uts.edu.au

Guangquan Zhang
Centre for Artificial Intelligence,
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
Guangquan.Zhang@uts.edu.au

Kun Wang
Centre for Artificial Intelligence,
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
kun.wang-2@student.uts.edu.au

Jie Lu
Centre for Artificial Intelligence,
Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
Jie.Lu@uts.edu.au

***Abstract*—In data stream mining, concept drift may cause the predictions given by machine learning models become less accurate as time passes. Existing concept drift detection and adaptation methods are built based on a framework that is buffering new samples if a drift-warming level is triggered and retraining a new model if a drift-alarm level is triggered. However, these methods neglected the problem that the performance of a learning model could be more sensitive to the amount of training data rather than the concept drift. In other words, a retrained model built on very few data instances could be even worse than the old model trained before the drift. To elaborate and address this problem, we propose a fast switch Naïve Bayes model (fsNB) for concept drift detection and adaptation. The intuition is to apply the idea of following the leader in online learning. We manipulate a sliding and an incremental Naïve Bayes classifier, if the sliding one overwhelms the incremental one, the model reports a drift. The experimental evaluation shows the advantages of fsNB and demonstrates that retraining may not be the best options for a marginal drift.**

***Index Terms*—concept drift, data stream, machine learning**

I. Introduction

Concept drift refers to a data stream learning task in which the target variable may change over time. Algorithms designed to address concept drift have acknowledged the importance of balancing the short-term and long-term performance [1], [2]. Conventional machine learning models assume that the learning problem is well represented by the training data and has the same pattern as further analysis tasks, that is, the data is stationary. However, in real-world scenarios, this is assumption may not hold, especially for applications involving data stream analysis [3]–[5]. In industrial applications, data collected from manufacture and operation processes demonstrate an inherited non-stationary nature. For example, the measurement of a sensor could vary due to faulty, aging or changes in operation conditions [2]. The main challenge of concept drift is that such changes are unforeseeable and there is no guarantee that data received in the future have the same distribution as the past [6], [7].

Real drift and virtual drift are two major types of concept drift. Real drift occurs when there are changes in the class boundaries. These changes will make the current model obsolete. By contrast, virtual drift means changes in the marginal distribution, while the class boundaries are not affected [3]–[5]. In real-world scenarios, these two types of drift could occur at the same time. In this paper, we focus on real drift detection and adaptation with a supervised learning setting, that is, we assume the true label is available after prediction.

One issue remains unsolved in the literature for real drift handling is when and how to update the learning models. The most popular drift detection and adaptation framework is setting two levels of drift, which are drift-warning level $\alpha = 0.05$ and drift-alarm level $\alpha = 0.01$ [3], [8], [9]. In this case, the system will start buffering new data instances if warning level is reached and will use the buffered data to retrain a new model if a drift alarm is triggered. However, this strategy has a risk that there could be insufficient data for retraining a model. In other words, the retrained model may be underfitting. For example, as shown in Fig 1., if the drift alarm is triggered after six time points of the warning level is reached, there will be only six data instances for retraining. It is hardly to affirm that the new retrained model will be better than the old one.

If a drift adaptation cannot improve the prediction accuracy after a drift, even the drift detection result is correct, and we consider this as a redundant adaptation. We call the concept drift adaptation that reduces the model performance as negative drift adaptation. Existing drift detection algorithms neglect this issue. Hence there is a clear need for efficient methods that can help concept drift handling algorithms to decide on when to choose fine-tuning over retraining.

There are few algorithms build with paired learners that are capable of handling this issue. They compare the performance of a fine-tuned and a retrained model at every time point [1], [2], [10]. This strategy is useful and straightforward but

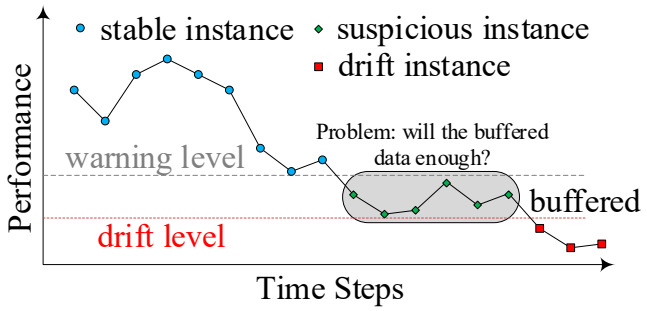

Fig. 1. A demonstration of existing real drift detection and adaptation framework. The problem is how to determine a drift adaptation is beneficial to the learning model but not reducing its overall performance.

not efficient enough. These methods have to retrain a model whenever a new data instance is available. If the instances arrive in sequence, they have to manually configure a sliding time window to store the data for retraining. This process has a very high computational cost and may easily be affected by noisy [2]. Besides, the learner switch threshold is also unclear. Most of the thresholds are chosen by grid search or expert experience [1], [2].

In this paper, we assume that, without concept drift, the residual of a learning model would be stable. As a result, a two-sample hypothesis test, such as Kolmogorov–Smirnov two-sample test (KS test), should be able to identify whether the residual of an incremental learner is statistically smaller than a non-incremental one's. According to the hypothesis test result, the learning model can decide to switch to a new retrained model or stay on the incrementally fine-tuned model. This is an improvement to the paired learners because we do not need to choose the replacement threshold manually. Also, the residual of the retrained model can be reused; therefore, no retraining process is required for new data instances, unless there is a concept drift.

The main contribution of this paper is as follows.

- A novel and efficient time window evaluation strategy is proposed so that the paired learner will not need to retrain at every time point.
- A simple but effective residual-based real drift detection algorithm is developed. The drift detection is aiming to reflect the difference in the performance between a fine-tuned model and a retrained model, but not the change in the data distribution.
- A follows by leader drift adaptation algorithm is developed. The drift adaptation can automatically choose the drift threshold and switch the leader model based on their performance.

II. Literature Review

In this section, we formally present the definition of concept drift. Followed by the state-of-the-art learner error-based drift detection and adaptation algorithms. At last, we introduce the methodology of Naïve Bayes classifier.

A. Real drift and virtual drift

Concept drift is defined as a phenomenon that the statistical properties of a target domain change over time in an arbitrary way [3]. These changes could be caused by hidden variables or some features that cannot be measured directly. More formally, concept drift is defined as: at time t a set of observations is given, denoted as $D_t = \{(X_i, y_i)\}_{i=1}^{n}$, where $X_i \in \mathbb{R}^d$ is the feature vector, d is the dimensionality, $y_i \in Y$ is the label, and the n denotes the number of instances arrived at time t. Given that D_t follows a joint distribution $p_t(X, y)$, concept drift is identified whenever there is a statistical significant difference between any two instances sets D_t, D_{t+1} that $p_t(X, y) \neq p_{t+1}(X, y)$. The joint distribution can be presented as $p(X, y) = p(y|X)\dot{p}(X)$. In considering problems that use X to infer y, concept drift is generally divided into virtual drift that $p(X)$ changed while $p(y|X)$ remains unchanged and real drift that $p(y|X)$ changed while $p(X)$ remains unchanged. In the real-world scenarios, most applications aspire to find the $p(y|X)$ change, because such changes directly affect the performance of the learning model [11].

According to a recent literature survey [5], virtual drift detection and adaptation algorithms are focusing on measuring and reducing the distribution discrepancy between two sample sets. For example, [6], [12], [13], they have a limited representation of the model's performance change. In other words, in some extreme cases that the data distribution drift significantly, but the decision boundary does not change, impulsive retraining may impair the overall performance. In contrast, real drift detection and adaption algorithms, such as EDDM [8], ADWIN [14], ECDD [15] and HDDM family [9] FW-DDM [16] are more productivity-oriented that only updating the models when their performance has significantly dropped.

Among the real-drift detection and adaptation algorithms, few research works are aiming at comparing the performance of a retrained model and a fine-tuned model [1], [2]. However, they have few parameters chosen empirically, such as the evaluation window size and the drift threshold [1], [2]. In general, we summarized the unsolved problems of these algorithms in Fig. 2., as follows: Problem 1: over frequent retraining; Problem 2: the evaluation window size is sensitive and will affect the overall performance; Problem 3: model switch threshold is uncertain.

B. Naïve Bayes Classifier

Naïve Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_i through x_d,

$$p(y|x_1, \ldots, x_d) = \frac{p(y)p(x_1, \ldots, x_d|y)}{P(x_1, \ldots, x_d)}.$$

Using the naive conditional independence assumption that

$$p(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d) = p(x_i|y)$$

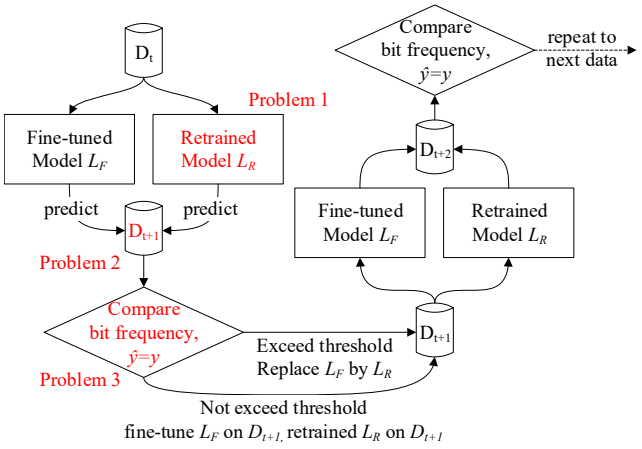

Fig. 2. A summary of paired learners for concept drift handling. The cost of frequent retraining is problem 1. How to set the window size for evaluation is problem 2. And how to set the drift threshold is problem 3.

for all i, this relationship is simplified to

$$p(y|x_1,\ldots,x_d) = \frac{p(y)\prod_{i=1}^{d} p(x_i|y))}{p(x_1,\ldots,x_d)}.$$

Since $p(x_1,\ldots,x_n)$ is constant given the input, we can use the following classification rule:

$$p(y|x_1,\ldots,x_d) \propto p(y)\prod_{i=1}^{d} p(x_i|y),$$

and we can use maximum a posterior estimation to estimate $p(y)$ and $p(x_i|y)$, i.e.,

$$\hat{y} = \arg\max_{y} p(y)\prod_{i=1}^{d} p(x_i|y).$$

The former is then the relative frequency of class y in the training set [17]. According to the "naive" assumption, Naïve Bayes classifier can be implemented in an incremental manner. For data stream learning tasks, we consider the incremental learning process as a model fine-tune process.

Another advantage of Naïve Bayes classifier is its low runtime complexity. The runtime complexity of Naïve Bayes and incremental Naïve Bayes is $\mathcal{O}(Nd)$, where N is the size of the training data. Low runtime complexity is essential for data stream learning [5], [18], [19].

III. Fast Switch Naïve Bayes for Learning with Concept Drift

In this section, we formally present the proposed fast switch Naïve Bayes model (fsNB) for learning under concept drift. The model includes a novel time window evaluation strategy, a residual-based drift detection algorithm, and a drift adaptation algorithm based on the Follow the Leader principle.

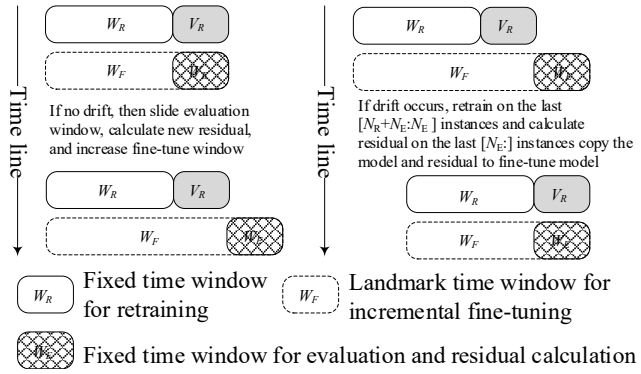

Fig. 3. The time window strategy of fsNB. There are three time windows for fsNB to implement drift detection and adaptation. The first window is a fixed window for model retraining, denotes as W_R. The second window is a landmark time window for model fine-tuning, denotes as W_F. The last one is a sliding window that stores the residuals of most recent instances, denotes as W_E. In the beginning, we build a model on W_R and evaluate it on W_E. The residual vector is buffered for drift detection, denotes as V_R. We initialize a fine-tuned model and its residual V_F by copying the first model and V_R. For a new data instance (X_t, y_t), the fine-tuned model will make the prediction and calculate the residual r_t. The r_t will be appended to V_F and the first element of V_F will be removed, which forms a sliding residual buffer.

A. The Time Window Strategy for Fast Switch Naïve Bayes

Time window strategy is a vital component for data stream learning [5]. It helps organizing training and testing data from data streams. According to an empirical study [16], different time window settings will result in totally different results. To illustrate how fsNB organized data instances in a data stream, we present the proposed time window strategy in Fig. 3. The fsNB maintains three time windows: retraining window W_R, fine-tuning window W_F and evaluation window W_E. The window size of them are denoted as N_R, N_F, and N_E respectfully. The time windows will not store the data. They are only to demonstrate how the models are trained and evaluated.

This time window strategy poses no additional runtime complexity to incremental Naïve Bayes classier. All the data will only be accessed once, except the most recent $[-(N_R+N_E):]$ data. Because these data will be used for creating a new retrained model and its residuals when concept drift occurs.

B. Drift Detection based on the Residual of Naïve Bayes Classifier

The underlying assumption here is that without a concept drift, residuals of a model on any two testing data batches should have the same distribution. Compare to using the model's accuracy or the frequency of correct predictions, using residuals contains more details about the model's performance change and could be an alternative way for drift detection.

The intuition of using residual for drift detection is to simplify data representation methods. We map high dimensional data to one-dimension by building a learning model, i.e., mapping the joint distribution $p(X, y)$ to residual, denoted as $f_{NB}(X, y) \rightarrow r$. Since real-drift only cares about the $p(y|X)$ drift, this mapping help simplify the drift detection problem.

According to the definition of Naïve Bayes classifier, we have the residual of a data instance $r(X_j, y_j)$ calculated by

$$r(\{x_{ji}\}_{i=1}^{d}, y_j) = 1 - p(y_j) \prod_{i=1}^{d} p(x_{ji}|y = y_j)$$

or shorten for r_j. And we have the residual vector of a batch of evaluation data as $V = \{r_j\}_{j=1}^{N_E}$, where N_E denotes the size of the evaluation window. For easy memory, we consider all batch data as a time window retrieved from a data stream, therefore, using W to denote the data batch. Drift detection is implemented by using the Kolmogorov-Smirnov two-sample test (KS test) to compare the residuals. Since we have a fixed retrained model and an incremental fine-tuned model, we could have two residual vectors on the evaluation data batch, denoted as V_F, V_R. We report a real drift if the KS test rejects the null hypothesis that $p(V_F) = p(V_R)$.

The empirical distribution function for N_E independent and identically distributed (i.i.d.) ordered residuals r_i is defined as

$$F_{N_E}(r) = \frac{1}{N_E} \sum_{i=1}^{N_E} I_{[-\infty,r]}(r_i)$$

where $I_{[-\infty,r]}(r_i)$ is the indicator function, equal to 1 if $r_i \leq r$ and equal to 0 otherwise.

The Kolmogorov–Smirnov statistic for the empirical distribution of residuals of retrained $F_R(r)$ and fine-tuned $F_F(r)$ models is

$$s_{KS} = \sup_r |F_R(r) - F_F(r)|.$$

Then for large samples, the null hypothesis is rejected at level α if

$$s_{KS} > c(\alpha) \sqrt{\frac{|V_R| + |V_F|}{|V_R||V_F|}},$$

where $|\cdot|$ denotes the cardinality of a set and the $c(\alpha)$ is calculated in general by

$$c(\alpha) = \sqrt{-\frac{1}{2} \ln \frac{\alpha}{2}}.$$

The two-sample test checks whether the two residual set come from the same distribution.

To determine whether the drift is improving or deteriorating the performance, we add a constrain before triggering the drift detection, i.e., the mean absolute value (MA) of the fine-tuned model residuals should be lower than the retrained one. This rule is presented as follow

$$\text{if } \mathrm{MA}(V_R) \leq \mathrm{MA}(V_F) \text{ then } \mathrm{KStest}(V_R, V_F).$$

This ensures the drift adaptation will only be performed when the retrained model significantly outperforms the fine-tuned one. The pseudocode of fsNB drift detection algorithm is shown in Alg. 1.

Algorithm 1: fsNB Drift Detection

Input : residual of retrained model, V_R
residual of fine-tuned model, V_F
drift significant level, $\alpha = 0.01$ as default
Output: drift indicator, $I_{drift} \in \{0, 1\}$

```
if MA(V_R) ≤ MA(V_F) then
    run KS-test and calculate the
     pValue=KStest(V_R,V_F);
    if pValue ≤ α then
        return 1;
return 0;
```

C. Drift Adaptation by Follow the Leader Principle

Concept drift adaptation or reaction is to update existing learning models according to the situation after a concept drift. If the drift severity or the impact is not significant, the model can incrementally be fine-tuned by new data instances. However, in some cases, the incremental fine-tune process is too slow for the drift recovery to keep the model meet its performance requirements. Therefore, a retraining process will be triggered. In this section, we propose a follow the leader drift adaptation algorithm for learning with concept drift.

The simplest learning rule to try is to select the hypothesis that has the least loss over recent past time. This principle is called follow the leader, and is simply given at time t. The leader is selected by

$$\text{Leader} = \underset{f(x)\in\mathcal{H}}{\arg\min} \sum_{i=1}^{t} |f(x_i) - y_i|,$$

where the hypothesis set is $\{f_R(x), f_F(x)\}$, namely the retrained model and the fine-tuned model. In our case, we can simply substitute the residual as the loss measurement, and we have the leader selected by

$$\text{Leader} = \underset{f(x)\in\{f_R(x), f_F(x)\}}{\arg\min} \mathrm{MA}(V_R) - \mathrm{MA}(V_F),$$

This method can thus be looked as a greedy algorithm, because it does not consider the following situation. For a pair of learners, the one has the best performance on the most recent predictions will lead the model. In this case, the drift detection can be considered as a leader selection process. If there is a leader changed, which means the rank of their performance switched, the model reports a concept drift. The pseudocode of fsNB drift adaption algorithm is shown in Alg. 2 and the pseudocode of fsNB data stream learning algorithm is shown in Alg. 3, where the time windowing method is included.

IV. Experiment Evaluation

This section presents an implementation of the proposed algorithm in Section III. Six algorithms (3 baselines and 3 state-of-the-art algorithms) and five data sets were used for evaluating the performance of fsNB. Section IV-A introduce

Algorithm 2: fsNB Drift Adaptation

Input : Most recent $N_R + N_E$ data,
$\mathrm{W_{buff}} = \mathrm{Stream}[-(N_R + N_E) :]$
residual of retrained model, V_R
residual of fine-tuned model, V_F
drift indicator, I_{drift}
Output: leader learner

```
if I_drift = 1 then
    retrain model on
      f_R(x) = NaiveBayes(W_buff[: N_R]);
    calculate residual V_R = Y_[-N_E:] − f_R(X_[-N_E:]);
    copy f_F(x) ← f_R(x) and V_F ← V_R;
return f_F(x);
```

Algorithm 3: fsNB Data Stream Learning

Input : data stream $\{(X_i, y_i)\}_{i=1}^{t}$
drift significant level, $\alpha = 0.01$ as default
Retrain window size $N_R = 200$
Evaluation window size $N_E = 10$
Output: $\{\hat{y}\}_{i=1}^{t}$

```
initial retrain window W_R = (X_i, y_i)_{i=1}^{N_R};
initial evaluation window W_E = (X_j, y_j)_{j=1}^{N_E};
initial retrain model f_R(x) = NaiveBayes(W_R);
initial retrain residual V_R = y_j − f_R(X_j);
initial fine-tune model and its residual
  f_F(x) ← f_R(x), V_F ← V_R;
while stream has X_t for prediction do
    predict ŷ_t = f_F(X_t);
    calculate new residual r_t = y_t − proba(ŷ_t);
    slide fine-tune residual V_F[1 :] ← [V_F[1 :], r_t];
    slide evaluation window
      W_E[1 :] ← [W_E[1 :], (X_t, y_t)];
    detect drift I_drift = fsNBDetection(V_R, V_F, α);
    fsNBAdaptation([W_R, W_E], I_drift, V_F, V_R);
return {ŷ}_{i=1}^{t};
```

the configuration of the compared algorithms. Section IV-B presents the statistics of the data sets, and section IV-C summarised the evaluation results and discuss the findings. The dataset and the source code of this research are available online[1].

A. Experiment Configuration

To implement the fsNB algorithm, we applied the NaïveBayse module in the skmultiflow.bayes python package [20]. The evaluation metric is prequential accuracy [21].

$$\mathrm{preAcc}_t = \begin{cases} \mathrm{preAcc}_{ini}, & \text{if } t = t_{ini} \\ \mathrm{preAcc}_{t-1} + \frac{\mathrm{preAcc}_{ini} - \mathrm{preAcc}_{t-1}}{t - t_{ini} + 1}, & \text{otherwise} \end{cases}$$

Compared algorithms and their settings are shown as follows.

[1]https://github.com/Anjin-Liu/IJCNN-fsNB

TABLE I
SYNTHETIC DATA SET STATISTICS. THE RATIO INDICATES THE CLASS RATIO.

Data set	# Samples	# Features	# Class	Ratio
SEA	10,000	3	2	1:1
RTG	10,000	10	2	1:1
RBF	10,000	10	2	1:1
HYP	10,000	10	2	1:1
AGR	10,000	9	2	1:1

FixNB is a Naïve Bayes classifier trained at the first 200 data instances. It will then be used for testing the rest of the stream. FixNB is the first baseline method of our evaluation.

IncNB is an incremental Naïve Bayes classifier that continuously fine-tuned on new coming data. The learner will predict the new data first and then incrementally learn it. We consider IncNB as the second baseline methods.

PairedLearner is the Paired Naïve Bayes Learners algorithm [1]. PairedLearner is the third baseline methods, because it has the same drift detection strategy as fsNB.

EDDM [8] is a upgraded version of DDM [22]. It has been widely used for concept drift detection and adaption evaluation. Since EDDM has the same mechanism of DDM, we only use EDDM for comparative analysis.

ADWIN [14] change detector is the most referenced change detector in the literature so far. It has combined with different drift adaptation strategy for data stream learning [23]. In this experiment, we only evaluate its power on drift detection. All the drift adaptation was implemented by retraining a new model.

HDDM family [9], including HDDM-A test and HDDM-W test, that was based on Hoeffding bound. They have shown reliable power on learner error-based drift detection [5].

All the algorithms were implemented by the MOA platform [19]. The experiments were conducted on a cluster node with 3.4GHz 8 cores CPU and 32GB RAM. The parameters were set as the default value as suggested by their authors.

B. The Data Sets

Each approach was tested on five synthetic data sets. The statistics about these data sets are summarized in Tables I. A brief description of each follows. The synthetic data sets were:

The **SEA** generator [24] produces data streams with three continuous attributes, $X = \{x_1, x_2, x_3\}$ and $x_1, x_2, x_3 \in [0, 10]$. An inequality determines the label of each data instance, $x_1 + x_2 \leq \theta$, where θ is a threshold to control the label boundary. The entire data stream was divided into four subsets with different data distributions ("Concepts") of equal size, and θ was 8, 9, 7, and 9.5, respectively. This evaluation method has been widely used in sudden drift detection and adaptation [16], [23], [25]. There were 10,000 data instances at a noise ratio of 10%.

The rotating **Hyp**erplane generator [26] produces data streams with ten continuous attributes, $X = \{x_1, \ldots, x_{10}\}$ and $x_1, \ldots, x_{10} \in [0, 1]$. The label boundary for classification was determined by $\Sigma_{i=1}^{d} w_i x_i \geq \theta$, where d is the number

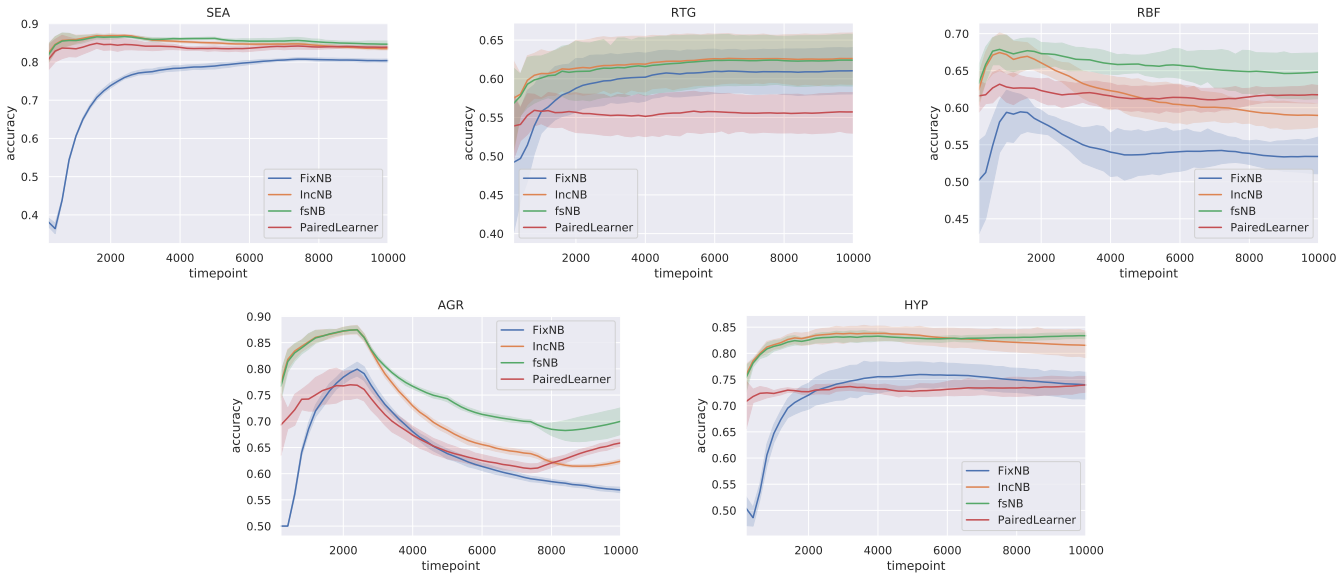

Fig. 4. A plot of the prequential accuracy and the confidence interval of the baseline and fsNB algorithm. It can be seen that fsNB outperform the baselines on all evaluated data sets except the RTG data set. This is because RTG data set does not contain any simulated drift.

TABLE II
CLASSIFICATION ACCURACY OF THE STATE-OF-THE-ART ALGORITHMS (%). THE NUMBER IN THE BUCKET INDICATES THE RANK OF THE ALGORITHM ON THIS DATA SET. ALL THE RESULTS WERE SUMMARIZED BY RUNNING 15 COPIES OF THE DATA SETS, WHICH WERE GENERATED WITH A DIFFERENT RANDOM SEED. THE ALGORITHMS WITH UNDERLINE ARE THE BASELINE OF OUR METHOD. IT CAN BE SEEN THAT FSNB OUTPERFORMS BASELINES ON EVERY EVALUATED DATA SET.

Algorihtms	SEA	RTG	RBF	AGR	HYP	AvgRank
fsNB	85.32±1.20 (2)	64.09±4.48 (5)	66.29±3.58 (2)	69.06±3.16 (4)	**82.75±1.03 (1)**	**2.8**
HDDM-W test	84.49±0.79 (3)	63.39±4.92 (6)	**67.14±1.99 (1)**	74.53±0.89 (2)	82.58±1.75 (2)	**2.8**
HDDM-A test	84.48±0.72 (4)	64.26±4.73 (3)	65.66±2.57 (3)	**74.54±0.80 (1)**	82.29±2.30 (4)	3
EDDM	**85.36±0.62 (1)**	64.13±4.61 (4)	65.39±2.87 (4)	70.61±2.37 (3)	82.34±1.61 (3)	3
IncNB	83.62±0.36 (6)	**64.36±4.48 (1)**	60.42±2.62 (6)	62.09±0.61 (6)	80.27±3.17 (5)	4.8
PairedLearner	84.31±0.75 (5)	57.51±4.09 (8)	62.91±2.78 (5)	65.55±0.92 (5)	72.88±1.67 (7)	6
ADWIN	70.50±12.28 (8)	64.35±4.65 (2)	57.55±5.55 (7)	59.30±0.22 (7)	73.42±8.79 (6)	6
FixNB	80.56±0.60 (7)	62.70±4.58 (7)	53.24±3.63 (8)	56.67±0.60 (8)	71.69±6.29 (8)	7.6

of features related to drift, and w_i are weights that randomly initialize in the range of $[0, 1]$. Incrementally changing the threshold θ produces a rotating hyperplane label boundary, thereby generating incremental concept drifts. In this experiment we set $d = 2$, that is, only the first two features had incremental drifts. Again, there were 10,000 data instances, and the noise ratio was set to 10%.

The **AGR**AWAL [27] generator creates instances with six nominal and three continuous attributes. Ten functions are available to map instances into two classes. We used the first four functions in MOA to simulate four concepts of equal length. The same gradual drift configuration was applied to AGRg.

The Random Tree Generator (**RTG**) randomly builds a decision tree and randomly assigns a class label to each leaf node, after which the data is uniformly distributed to the leaf nodes. For this dataset, we applied the MOA default setting to create a non-drifting dataset.

The **RBF** generator creates data instances using a radial basis function. It creates centroids at random positions and associates them with a standard deviation value, a weight, and a class label. Incremental drifts are simulated by continuously moving the centroids. For the RBF incremental drift, 50/50 centroids are drifting.

C. Findings and Discussion

The evaluation results were calculated based on 15 runs of each dataset. The average accuracy and standard deviation of accuracy are given in Table II. The prequential accuracy of FixNB, IncNB, PairedLearner and fsNB are plot in Fig. 4.

As shown in Table II, the proposed fsNB and HDDM-W test have the best performance rank. Compare to the baseline algorithms, fsNB outperforms them on all evaluated data sets except the RTG data set, which indicates the improvement and the efficacy of fsNB. The reason that no algorithm can beat the IncNB on the RTG data set is that RTG has no simulated concept drift. Therefore, incremental learning is supposed to have the best performance. For the algorithm that has no false

alarms, which has no retraining process triggered, it should have the same result as incremental learning. Otherwise, any false alarms will result in an accuracy drop. From this point of view, the performance of the algorithms on RTG can be used to evaluate the false alarm rate. As shown in the result table, ADWIN change detector has the closest result to IncNB. This implies that ADWIN had almost no false alarm triggered. However, this at the cost of not sensitive to real drift.

V. Conclusion and Future Study

In this paper, we proposed a fast switch Naïve Bayes algorithm for concept detection and adaptation. Unlike conventional concept drift detection and adaptation algorithms using warning and drift level to control the adaptation process, we directly switch to the most suitable classifier after detecting a significant performance difference. The proposed method can balance the fine-tuned and retrained models based on the situation. It consists of a novel time window evaluation strategy, implemented a residual-based two-sample test and developed a drift adaptation algorithm. Compared to the baselines, fast switch Naïve Bayes has less retraining process (target Problem 1) and has a drift threshold dynamically selected according to evaluation window size (target Problem 2, 3). The empirical study demonstrated the advantages of our algorithm and showed its potential for further improvements.

In the future work, a diversity generation method could be developed for ensemble the proposed method, which may further boost the overall learning performance. We will also continue to investigate the influence of different drift types on the model's residuals and try to understand the impact of the concept drift on the fine-tuned and retrained model. At last, a good practice would be extending the proposed framework to learning models other than Naïve Bayes classifier.

Acknowledgment

The work presented in this paper was supported by the Australian Research Council (ARC) under Discovery Project DP190101733.

References

[1] S. H. Bach and M. Maloof, "Paired learners for concept drift," in *Proceedings of the Tenth IEEE International Conference on Data Mining*, 2008, pp. 23–32.

[2] Y. Xu, R. Xu, W. Yan, and P. Ardis, "Concept drift learning with alternating learners," in *Proceedings of the 2017 International Joint Conference on Neural Networks*, 2017, pp. 2104–2111.

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–37, 2014.

[4] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in non-stationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, pp. 12–25, 2015.

[5] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[6] A. Liu, J. Lu, F. Liu, and G. Zhang, "Accumulating regional density dissimilarity for concept drift detection in data streams," *Pattern Recognition*, vol. 76, pp. 256–272, 2018.

[7] S. Pan, K. Wu, Y. Zhang, and X. Li, "Classifier ensemble for uncertain data stream classification," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2010, pp. 488–495.

[8] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early drift detection method," in *Proceedings of the Fourth International Workshop on Knowledge Discovery from Data Streams*, vol. 6, 2006, pp. 77–86.

[9] I. Frias-Blanco, J. d. Campo-Avila, G. Ramos-Jimenes, R. Morales-Bueno, A. Ortiz-Diaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2015.

[10] J. Shao, Z. Ahmadi, and S. Kramer, "Prototype-based learning on concept-drifting data streams," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 412–421.

[11] A. Liu, G. Zhang, and J. Lu, "Region drift disagreement-based diverse instances weighting ensemble for concept drift adaptation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. Early Access, doi:10.1109/TNNLS.2020.2978523, pp. 1–16, 2020.

[12] A. Liu, Y. Song, G. Zhang, and J. Lu, "Regional concept drift detection and density synchronized drift adaptation," in *Proceedings of the Twenty-sixth International Joint Conference on Artificial Intelligence*, 2017, pp. 2280–2286.

[13] A. Liu, G. Zhang, and J. Lu, "Concept drift detection via equal intensity k-means space partitioning," *IEEE Transactions on Cybernetics*, vol. Early Access, doi:10.1109/TCYB.2020.2983962, pp. 1–14, 2020.

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

[15] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognition Letters*, vol. 33, no. 2, pp. 191–198, 2012.

[16] A. Liu, G. Zhang, and J. Lu, "Fuzzy time windowing for gradual concept drift adaptation," in *Proceedings of the Twenty-sixth IEEE International Conference on Fuzzy Systems*. IEEE, 2017.

[17] H. Zhang, "The optimality of naive bayes," *AA*, vol. 1, no. 2, p. 3, 2004.

[18] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–36, 2017.

[19] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, 2017.

[20] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *Journal of Machine Learning Research*, vol. 19, no. 72, pp. 1–5, 2018. [Online]. Available: http://jmlr.org/papers/v19/18-251.html

[21] L. L. Minku and X. Yao, "Ddd a new ensemble approach for dealing with concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619–633, 2012.

[22] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proceedings of the Seventeenth Brazilian Symposium on Artificial Intelligence*, vol. 3171. Springer, 2004, pp. 286–295.

[23] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, 2017.

[24] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the Seventh ACM International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 377–382.

[25] S. Xu and J. Wang, "Dynamic extreme learning machine for data stream classification," *Neurocomputing*, vol. 238, pp. 433–449, 2017.

[26] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the Seventh ACM International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 97–106.

[27] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.