Neurocomputing 166 (2015) 68–83

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Learning concept-drifting data streams with random ensemble decision trees ☆

Peipei Li [a,*], Xindong Wu [a,b], Xuegang Hu [a], Hao Wang [a]

[a] Hefei University of Technology, Hefei, China, 230009
[b] University of Vermont, Vermont 05405, USA

ARTICLE INFO

Article history:
Received 2 August 2013
Received in revised form
15 February 2015
Accepted 11 April 2015
Communicated by Hongli Dong
Available online 22 April 2015

Keywords:
Data streams
Random decision tree
Concept drift
Noisy data

ABSTRACT

Few online classification algorithms based on traditional inductive ensembling, such as online bagging or boosting, focus on handling concept drifting data streams while performing well on noisy data. Motivated by this, an incremental algorithm based on Ensemble Decision Trees for Concept-drifting data streams (EDTC) is proposed in this paper. Three variants of *random feature selection* are introduced to implement split-tests and two thresholds specified in Hoeffding Bounds inequality are utilized to distinguish concept drifts from noisy data. Extensive studies on synthetic and real streaming databases demonstrate that our algorithm of EDTC performs very well compared to several known online algorithms based on single models and ensemble models. A conclusion is hence drawn that multiple solutions are provided for learning from concept drifting data streams under noise.

1. Introduction

Industry and business applications generate a tremendous amount of data streams, such as telephone call records and sensor network data. Meanwhile, with the development of Web Services technology, streaming data from the Internet span across a wide range of domains, including shopping transactions and Internet search requests. In contrast to the traditional data sources of data mining, these streaming data present new characteristics as being continuous, high-volume, open-ended and concept drifting. It is hence a challenging and interesting issue for us to learn from concept drifting data streams.

As an important branch of data mining, classification can be solved by abundant inductive learning models [1–3]. Decision trees, due to their advantages, are one of the most popular techniques. They are widely used as a basic model in ensemble classifiers. However, it is still a challenge to learn from data streams with these traditional inductive learning models or algorithms. Thus, new models or algorithms tailored towards mining from data streams, based on decision trees, have been proposed. These recent works can be roughly classified into two main categories: *incremental decision tree based* and *random decision tree based*. Incremental decision tree based approaches for data stream classification use informed split-tests in the generation of decision trees, while random decision tree [4] based approaches select the split nodes randomly.

On one hand, main works based on incremental decision trees for data stream classification are summarized below, such as a classical continuously changing data stream algorithm of CVFDT (Concept-adapting Very Fast Decision Tree learner) [5], a Streaming Ensemble Algorithm (SEA) [6], an Accuracy Weighted Ensemble (AWE) algorithm [7], a Weighted Aging Ensemble (WAE) algorithm [8], an Additive Expert ensemble algorithm of AddExp [9] developed from the incremental ensemble method of DWM (Dynamic Weighted Majority) [10], a VFDTc [11] model extended from the VFDT (Very Fast Decision Tree) algorithm [12], a boosting-like method [13], two variants of bagging with adaptive windowing and bagging based on adaptive-size Hoeffding tree [14], the perceptron classifier based learning algorithms [15,16] and other variants of decision tree based algorithms [17,18], an incrementally Optimized Very Fast Decision Tree (iOVFDT) [19] and an Evolutionary-Adapted Ensemble (EAE) method [20]. All of the above methods enable adapting to the concept drifting data streams, but they cannot perform well in the prediction accuracy especially when handling noisy data streams. This is because these classification methods require informed split tests. It will lead to be impacted from the noisy data more significantly.

On the other hand, main works based on random decision trees for data stream classification are summarized below, such as a random decision tree ensembling method [21] with cross-validation estimation, an online algorithm of Streaming Random Forests [22] extended from

☆A preliminary version of this paper has been accepted for publication in the Proceeding of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2011.

* Corresponding author.
E-mail addresses: peipeili.hfut@gmail.com (P. Li), xwu@cs.uvm.edu (X. Wu), jsjxhuxg@hfut.edu.cn (X. Hu), jsjxwangh@hfut.edu.cn (H. Wang).

http://dx.doi.org/10.1016/j.neucom.2015.04.024
0925-2312/

Random Forests [23], an algorithm using an entropy-based drift-detection technique for evolving data streams [24], an algorithm of Semi-Random Multiple decision Trees for Data Streams (SRMTDS) [25], an algorithm of Multiple Semi-Random decision Trees for concept-drifting data streams (MSRT) [26] and a Concept Drifting detection algorithm based on an ensemble model of Random Decision Trees (CDRDT) [27]. These methods mentioned above explore the efficiency and effectiveness in the classification of data streams using random split tests, which give us some insights for improving the efficiency and effectiveness in the classification of data streams with concept drifts and noisy data.

In this paper, we propose an incremental Ensemble algorithm based on random Decision Trees for Concept drifting data streams with noisy data called EDTC. This work is developed from our previous work in [28]. Our major contributions are as follows:

- We develop three variants of *random feature selection* to determine split-information in our algorithm, instead of informed split-tests in the aforementioned incremental decision tree based approaches. Because observations in [29,25] reveal that in comparison to the algorithms with informed split-tests in the heuristic methods, such as C4.5, bagging and boosting, the algorithms based on *random feature selection* are more effective and efficient even if in the case of noisy data.
- Contrary to most of the random decision tree based approaches mentioned above, we propose a new incremental ensemble algorithm based on random decision trees. A significant difference is that in our algorithm, each growing node in trees will not select split-features randomly until a real training instance arrives. It is conductive to avoid generating unnecessary branches. In addition, we know that the training of the ensemble classifiers can adapt to the incremental or gradual changes of data streams, but it only implicitly tells us that there are concept drifts in the data streams. To explicitly detect the types of concept drifts, in virtual of the Hoeffding Bounds inequality [30], we adopt a double-threshold-based mechanism of concept drifting detection for better tracking concept drifts and reducing the impact from noise. Instead of the global data distributions in all decision trees used in [27], we detect the concept drifts using the local data distributions in a decision tree. It is beneficial to improve the detection accuracy on the concept shifts (namely the sudden drifts) and the sampling changes [31].
- Contrary to our previous work in [28], we systematically analyze the generalization error of our EDTC algorithm impacted from the noisy data and the concept drifts. Meanwhile, we give more experiments conducted on the synthetic and real streaming data to show how our algorithm performs varying with the noisy data and the concept drifts. Extensive study shows that in comparison to the state-of-the-art online algorithms based on single and ensemble models mentioned above, our EDTC algorithm could efficiently and effectively tackle concept-drifting data streams with noisy data.

The rest of this paper is organized as follows. Section 2 reviews related work based on incremental decision trees and random decision trees. Our random ensemble random decision tree algorithm for concept drifting data streams is described in detail in Section 3. Section 4 provides the experimental study and Section 5 is the summary.

2. Related work

In this section, we first give the related works on the incremental decision tree based approaches for data stream classification, and then we give some works on random decision trees, and the random decision tree based approaches for data stream classification.

Incremental decision tree based approaches for data stream classification: We summarize main works based on incremental decision trees for data stream classification below. One classical continuously changing data stream algorithm of CVFDT (Concept-adapting Very Fast Decision Tree learner) [5] developed in 2001, using a fixed-size time window, constructs alternative sub-trees for each node to replace the old ones if concept drifts occur. Meanwhile, a Streaming Ensemble Algorithm (SEA) [6] addresses the concept drift of data streams, but it seems that the recovery time from concept drifts is demanded unnecessarily long. In 2003, an Accuracy Weighted Ensemble (AWE) algorithm was proposed for mining concept-drifting data streams using the more advanced weighted voting strategy [7]. A new algorithm called Weighted Aging Ensemble (WAE) [8] was developed from AWE in the following two modifications: (i) classifier weights depend on the individual classifier accuracies and the time they have been spending in the ensemble, (ii) individual classifiers are chosen to the ensemble on the basis of the non-pairwise diversity measure. An Additive Expert ensemble algorithm of AddExp [9] was developed from the incremental ensemble method of DWM (Dynamic Weighted Majority) [10] in 2005. It steadily updates the weights of experts and adds a new expert each time if misclassifying an instance. In 2006, a VFDTc model extended from the VFDT (Very Fast Decision Tree) algorithm [12] was designed for concept drifting data streams [11]. A boosting-like method [13] based on a classifier ensemble learned from data streams was presented in 2007 for quickly adapting to different kinds of concept drifts. Evidence shows that the advantages of boosting will be lost if there is non-trivial classification noise in the training test. In 2009, two new variants of Bagging with Adaptive Windowing and Bagging based on Adaptive-Size Hoeffding Tree [14] were developed for concept drifting data streams. In 2010 and 2011, the perceptron classifier based learning algorithms [15,16] and other variants of decision tree-based algorithms were proposed for evolving data streams [17,18,63]. In 2012, an incrementally Optimized Very Fast Decision Tree (iOVFDT) [19] was proposed for noisy data streams. In the following years, some evolutionary-adapted ensemble methods were proposed to process data streams characterized by the presence of recurring contexts [20,62]. All of the above methods are informed split-test based classification methods. They can adapt to the concept drifting data streams, but they cannot perform well in the prediction accuracy especially when handling noisy data streams. This is because the informed split-tests are more significantly impacted from the noisy data compared to the random selection in the split-tests.

Random decision trees: The concept of a *random decision forest* [32] was first proposed by Ho in 1995. It builds multiple trees in randomly selected subspaces of the feature space. In 1997, a *randomized tree* [33] was introduced by Amit. It generates feature subsets from all features randomly and uses the heuristic information entropy to select split-nodes. In 1998, Ho further designed a *random subspace* [34] technique to select random subsets of the available features in training individual classifiers of an ensemble. By using the bagging technique in tandem with random feature selection, Breiman proposed a model of *Random Forests* [23] in 2001. Contrary to the previous beliefs, a new model of *Random Decision Trees* [4] without any heuristics was designed by Fan in 2003. However, these algorithms cannot be applied in the handling of data streams due to the batch learning though they perform very well in the classification.

Random decision trees for data stream classification: New algorithms based on ensemble random decision trees subsequently have emerged for data stream handling. Main works are summarized below. A random decision tree ensembling method [21] with

cross-validation estimation was proposed by Fan in 2004. It could improve the accuracy of classification and adapt quickly to sudden drifts. In 2007, an algorithm of Streaming Random Forests [22] extended from *Random Forests* was presented by Abdulsalam et al. To further apply this algorithm to tackle concept drifts, an algorithm [24] using an entropy-based drift-detection technique was developed for evolving data streams. Meanwhile, an incremental algorithm of SRMTDS (Semi-Random Multiple Decision Trees for Data Streams) [25] was designed by Hu et al. It utilizes the inequality of Hoeffding Bounds to select cut-points at nodes with numerical attributes and introduces Naïve Bayes classifiers at leaves for higher predictive accuracies. To extend SRMTDS with the ability to deal with concept drifts, an algorithm of Multiple Semi-Random decision Trees for concept-drifting data streams called MSRT [26] and a Concept Drifting detection algorithm based on an ensemble model of Random Decision Trees called CDRDT [27] were further presented. Both of them adopt the double-threshold-based mechanism to detect concept drifts, but the difference lies in the evaluation method for error rates of classification, the values of thresholds and the strategy of adaptation to concept drifts. In the following years, many researchers have developed the streaming random decision trees and applied them into the real world applications, such as the spam email detection [35] and the video object tracking [36].

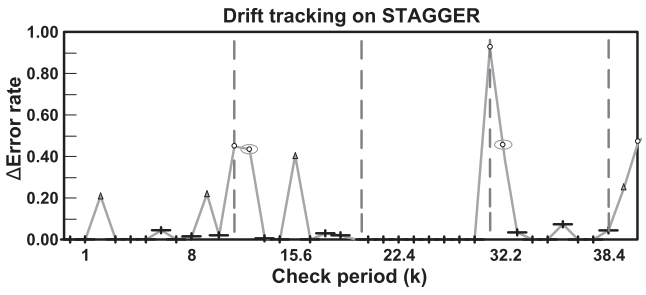

Fig. 1. An illustration to show three drifting cases. (For interpretation of the references to color in this figure, the reader is referred to the web version of this paper.)

Contrary to the aforementioned algorithms based on random decision trees, our algorithm of EDTC presents three diverse characteristics: (i) Instead of generating all trees with fixed heights in advance [21,4,25,26], decision nodes are scaled up incrementally after seeing real training instances. (ii) A variety of versions of *random feature selection* are explored to solve the split-information in the growing of trees. (iii) A double-threshold-based drift-detection technique is utilized to distinguish concept drifts from noise. Instead of using the global data distribution in all decision trees as [27], we investigate the local data distribution in a decision tree to judge the types of concept drifts. It is beneficial to improve the detection accuracy on the concept shifts and the sampling changes [31].

Table 1
Competing algorithms.

Algorithm	Full name	Base classifier
Data Stream Classification Algorithms		
PegasosBag	Pegasos [51] based Bagging	SVMs
PerceptronBag	Perceptron based Bagging	Single perceptron classifier
RTBag	Random decision Tree based Bagging	Ensemble random Hoeffding trees
VFDTc [52]	Refined Very Fast Decision Tree Learner	Single Hoeffding decision tree
OzaBag/OzaBoost [53]	Online bagging/boosting	Naïve Bayes classifier
OCBoost [54]	Online Coordinate Boosting	Naïve Bayes classifier
SRMTDS [25]	Semi-Random Multi-decision Trees for Data Streams	Ensemble semi-random decision trees
Concept Drifting Data Stream Classification Algorithms		
CVFDT [5]	Concept-adapting Very Fast Decision Tree learner	Single Hoeffding decision tree
DDM [46]	Drift Detection Method	Naïve Bayes classifier
EDDM [55]	Early Drift Detection Method	Naïve Bayes classifier
MSRT [26]	Multiple Semi-Random decision Trees for data streams	Ensemble semi-random decision trees
Bag10-ASHT-W+R [14]	Online Bagging based on Adaptive Size Hoeffding Trees	Weighted Hoeffding decision trees
iOVFDT-adaptive [19]	error-adaptive Incrementally Optimized Very Fast Decision Tree	Hoeffding decision tree

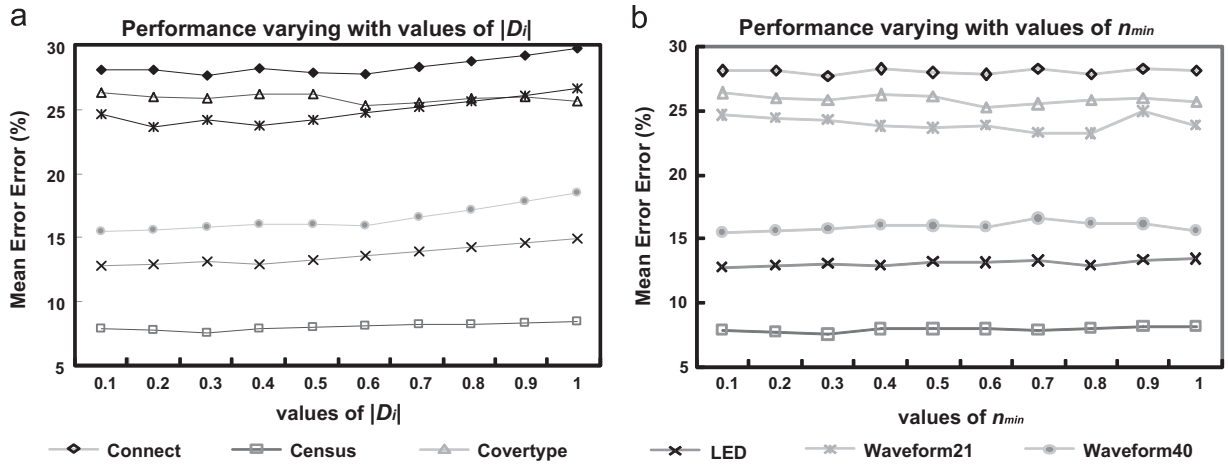

Fig. 2. Performance of EDT-RAN varying with values of n_{min} and D_i.

3. Ensemble random decision trees for concept-drifting data streams

We first formalize the problem of streaming data classification in this section. Let $E = R^d$ denote the domain of an instance space and Y denote the set of labels, where d is the dimensionality. Suppose a streaming data consists of the data chucks, namely $D = \{D_1, \ldots, D_i, \ldots\}$, each data chunk consists of a set of instances, namely $D_i = \{(x_j, y_j) | x_j \in R^d, y_j \in Y\}$. The goal of streaming data classification is to train a dynamic classifier $f : E_{\sum_{d_i}} \rightarrow Y$ that maps a feature vector to a set of labels and can adaptively adjust varying with the seen streaming data and the occurring concept drifts. Our EDTC algorithm to be presented here aims to handle this problem efficiently and effectively. Algorithm 1 shows the processing flow of EDTC. An ensemble of N-random decision trees is built using the following four components, including (i) *GenerateIncrTree*: It generates random decision trees incrementally in various strategies of *random feature selection*. (ii) *DetectDrifts*: if the total number of training instances arrived amounts to the value of a drifting check period-*DP*, distinguish concept drifts from noise and adapt to concept drifts. (iii) *JudgeMemory*: release the space overhead to avoid space overflow if the number of training instances arrived is up to the value of a memory check period-*MP*. (iv) Adopt a simple voting method in majority-class to classify testing data after training. Details of techniques are as follows.

3.1. Algorithm description

Incremental generation of a random decision tree:

Algorithm 2 shows the processing flow of *GenerateIncreTree*. To avoid growing sub-branches blindly, each node in trees is generated only if a real training instance arrives. Meanwhile, to obtain the maximum utility of each decision path in trees, the split-test mechanism in the growing of each random decision tree follows that discrete attributes should not be chosen again in a decision path while numerical attributes can be chosen multiple times. More specifically, a training instance (x_j, y_j) first traverses a tree from the root to an available node. Secondly, if it is a *growing* node, we select an available attribute (e.g., a_x, $1 \le x \le d$) randomly from the attribute set A as a split-attribute of the current node. According to the attribute type (numerical or discrete), we install the split-tests and generate the branches of children nodes as shown in Steps 8–18. If the height of decision tree grows up to the value of h, we mark the current node as a leaf and then update the statistical information of attribute values

Algorithm 1. EDTC (Ensemble random Decision Trees for Concept drifting data streams).

Input: Training set: $D = \{D_i | D_i = \{(x_j, y_j)\}(1 \le j \le |D_i|)\}$; Test set: $D' = \{(x_i', y_i')\}$ $(1 \le i \le |D'|)$; Attribute set: A=$\{a_1, \cdots, a_d\}$; Maximum height of the tree: h; Number of minimum split-examples: n_{min}; Number of trees: N; Memory check period: MP; Drifting check period: DP; Maximum memory consumption: MS; Size of sliding window: SW; Concept drifting types: F_{wd}.

Output: Prediction error on D';

1 Initialize a root for each decision tree T_k $(1 \le k \le N)$;
2 **for** *each training instance* $(x_j, y_j) \in D_i$ **do**
3 Load the current instance into a sliding window;
4 **if** *the size of sliding window* $> SW$ **then**
5 Forget the oldest instance in this window;
6 **end**
7 **for** *each tree* T_k $(1 \le k \le N)$ **do**
8 *GenerateIncreTree*$(T_k, (x_j, y_j), h, n_{min}, A)$;
9 **if** *the count of instances satisfies the check period of DP* **then**
10 $F_{wd} \leftarrow$ *DetectDrifts*(T_k, SW, DP);
11 **end**
12 **if** *the count of observed training instances at tree of* T_k *meets MP* **then**
13 *JudgeMemory*(MS);
14 **end**
15 **end**
16 **if** *the count of instances satisfies the check period of DP* **then**
17 Update the sizes of DP and SW regarding the worst case of concept drifts in F_{wd};
18 **end**
19 **end**
20 **for** *each testing instance* x_i' *in* D' **do**
21 Vote for the label of the testing instance x_i' in the majority class method;
22 **end**
23 return the prediction error on D';

relevant to all attributes in A. The above growing process will be repeated recursively until the end of training data is reached.

Random feature selection on numerical split-attributes: To solve the cut-point for each growing node with a numerical split-attribute, we explore three strategies based on *random feature selection* in this paper. The first one refers to the RHB method in [25], namely Randomization with the heuristic method in the inequality of Hoeffding Bounds. The second one specifies Randomization with the heuristic method of *Information Gain*, called RIG. The third one indicates RANdomization without any heuristic method, called RAN. Details are as follows.

- In the first strategy, we select a cut-point from the statistic attribute values of the current split-attribute (e.g., a_j) in the

Algorithm 2. GenerateIncreTree(T_k, (x_j, y_j), h, n_{min}, A).

```
Sort (x_j, y_j) into an available node;
if the depth of the current node in T_k is up to h then
    Label the current node as a leaf and Update the statistics at this leaf;
end
else
    if the passed current node of node_cur is a growing node without splitting then
        Select a split-attribute a_x ∈ A at node_cur randomly;
        if a_x is a numerical attribute then
            Mark the node as continuous and update the statistical information;
            if the statistical count of instances is up to n_min then
                Solve a cut-point for the current node by random feature selection;
                Generate two branches of children nodes according to < and ≥ the value
                of the cut point;
            end
        end
        if a_x is a discrete attribute then
            Generate a child by the attribute value of a_x in x_j;
            Set this child node to the current node of node_cur and go to Step 2;
        end
    end
end
```

Algorithm 3. DetectDrifts(T_k, SW, DP).

```
Initialize the drifting flag as f_wd;
for each node node_cur with the second highest level at the decision tree T_k do
    if the current node node_cur is unchecked then
        Compute the total error rate of all leaves as (Σ_{i=1} e_i)/DP;
        Let e_f set to e_s and e_s = (Σ_{i=1} e_i)/DP;
        if the number of detections at node_cur ≥ 2 then
            Update the value of f_wd regarding the relationships among Δe, e_l and e_h;
            while f_wd is a true concept drift and the height of node_cur > 2 do
                Copy node_cur as node_copy and set node_cur to its parent node back to Step 3;
            end
        end
        if node_copy has the flag of a true drifting then
            Reconstruct the sub-branch of node_copy using instances in the window of SW;
        end
    end
end
return the drifting type f_wd;
```

Table 2
Summary of the Friedman statistics F_r and the critical value in terms of each evaluation measure.

Performance of EDT and competing algorithms $k=11, N=7$			**Performance of EDTC and competing algorithms** $k=9, N=4$		
Evaluation measure	F_r	**Critical value** ($\alpha=0.05$)	**Evaluation measure**	F_r	**Critical value** ($\alpha=0.05$)
Error rate	9.6307		Error rate	5.6486	
Time overhead	3.5202	1.990	Time overhead	0.1236	2.360

evaluation function of Hoeffding Bounds inequality (denoted as $H(\cdot)$). The definition of Hoeffding Bounds inequality is given below. Consider a real-valued random variable r whose range is R. Suppose we have made n independent observations of this variable, and computed their mean $\bar{r}$, which shows that, with probability $1-\delta$, the true mean of the variable is at least $\bar{r}-\varepsilon$:

$$P(r \geq \bar{r}-\varepsilon) = 1-\delta, \varepsilon = \sqrt{R^2 \ln(1/\delta)/2n} \quad (1)$$

where R is set to $\log_2(M)$ (M refers to the number of class labels), and n indicates the number of instances required in a split-test (we use the value of n_{min} instead). According to Eq. (1), suppose $\Delta H = H(a_j^x) - H(a_j^y)$ refers to the difference between two candidate cut-points with the highest gain for attribute-a_j. For a given τ, if the condition of $\Delta H > \varepsilon$ or $\Delta H \leq \varepsilon < \tau$ holds, we will select the attribute value of xth cut-point with the highest gain as the final split-point, where τ is the threshold that we do not care about which classifier is more accurate, i.e., the case of ties.

- In the second strategy, we use the evaluation function of *Information Gain* only without the Hoeffding Bound constraints, that is, we will select the attribute value of xth cut-point with the highest gain as the final split-point if $\Delta H > 0$.
- However, in the third strategy, we evaluate the cut-point without any heuristic method. That is, we first randomly select a discretization interval index of attribute values for the current numerical split-attribute, and then specify the average value of the selected interval as the split-point. This processing is different from the split test on the numerical attribute involved in random decision tree [4], which randomly selects an attribute value for the current numerical attribute as the cut point.

In brief, because each strategy implies a certain random characteristic in the feature selection, we call them variants of *random feature selection*. In terms of the above processes, we can build an ensemble model of random decision trees with three different feature selection on numerical split-attributes.

Concept drifting detection: We now give the details of the double-threshold-based concept drifting detection method in our algorithm. Our concept drifting detection method called *Detect-Drifts* is still based on the time window idea [37]. However, it seeks to improve the performance in the tracking of concept drifts under noise. In this method, we define two thresholds as $e_l = c_1\varepsilon$ and $e_h = c_2\varepsilon$ (c_1, c_2 are constants, $c_1 < c_2$) in the inequality of Hoeffding Bounds. Our thought of these two thresholds is enlightened from the drifting detection in [38], but a significant difference depends on the introduction of the Hoeffding Bounds inequality. Also, it is very different from the method in [26] because we adopt different evaluation methods of error rates, different values of thresholds and different adjustment mechanism for concept drifts in our algorithm.

As shown in Algorithm 3, a bottom-up scanning in each decision tree starts on decision nodes with the second highest level every drifting check period. More specifically, for each available decision node with the same tree level, we first calculate the sum error rate of classification at its children nodes in Naïve Bayes at each drifting check period *DP*. We denote the estimation values in the current check period and the last check period as e_s and e_f respectively. Based on these two values, we compare their difference (i.e., $\Delta e = e_s - e_f$) with thresholds e_l and e_h to distinguish concept drifts from noise. This is because the change of error rates indicates the distribution change of class labels at children nodes, while the latter sufficiently represents the concept change hidden in the training data. Correspondingly, we could obtain three cases below:

- If satisfying $\Delta e \leq e_l$, it indicates that a *potential* concept drift is occurring.
- If satisfying $\Delta e \geq e_h$, a *true* concept drift is involved. In this case, sub-trees of the current decision tree will be reconstructed using instances in the sliding window.
- Otherwise, a *plausible* concept drift is considered due to the possible impact from noise. A similar detection process referred above is repeated till the root of the tree is reached.

Let us consider an example detected on the STAGGER [39] data set to show the different types of concept drifts. STAGGER is a standard database of concept-shifting data streams. In this example, we use the STAGGER generator in MOA [40] to generate the data set with 100 concept drifts, each concept drift has 10k-sized instances with 5%noise. Fig. 1 gives an illustration to show the cases of plausible/potential/true concept drifts. In this figure, '+' ○, Δ, indicate the detection point while the ellipse indicates the false alarm point. According to the relationships among e_l, Δe and e_h, we can get three cases of concept drifts from the observations. More specifically, the detection points marked in the symbol '+' are called as the potential concept drifts. The detection points marked in '○' with green font are called as the true concept drifts. And the others marked in 'Δ' are called as plausible concept drifts. According to the above detection method, we can avoid the false alarms by introducing the plausible concept drifts, because it is beneficial to reduce the noisy impact. However, we would not inform the arrival of a novel class until value of Δe is high enough between two check periods. Therefore, we will delay to report the real concept drifts (marked in dotted lines in this figure). Meanwhile, we will miss some real concept drifts, because we will probably mistake them as plausible concept drifts regarding the noisy data. More details of the STAGGER database and experimental results refer to Section 4.2.

In each concept drifting detection, to quickly adapt to concept drifts without sensitively reacting to noise, we will adjust the drifting check period (i.e., *DP*) and the size of sliding window (i.e., *SW*) dynamically regarding the worst drifting case in the whole of decision trees, namely the value in the set of drifting flags F_{wd}. More specifically, for a *potential* concept drift, we increase the values of *DP* and *SW* by a default value of *DP*. For a *true* drift (if there is a *true* drift in any one of decision trees), we reduce the value of *DP* to half an original length, and then shrink the value of *SW* by this updated value. However, for a *plausible* drift, we maintain the value of *DP* invariably and re-initialize the value of *SW*.

It is necessary to mention that according to the above concept drifting detection, the distribution change of class labels at children

nodes sufficiently represents the concept change hidden in the training data. In fact, the clustering approaches can also address the above point and be an effective means to identify concept change [41,42]. However, in this paper, we focus on the handling of completely labeled data streams without recurring concept drifts. To simplify the concept drifting detection, we use the change of error rates to represent the change of the distribution change of class labels at children nodes. This method is more efficient than that using clustering approach, but it is not more effective and earlier than the latter in the identifying the arrival of noisy and novel classes.

Handling of space overflow: With the continuously incoming training data, decision trees are growing, which probably leads to the space overflow. Thus, we provide an approach to space relief in *JudgeMemory*, which consists of three steps. Firstly, stop all undergoing splits of growing nodes and change them into leaves, then remove the storage space of information at the top-k nodes corresponding to their error rates of classification. Secondly, release the storage space of attribute values and class labels at decision nodes. Lastly, according to the simple pruning principle, cut off several sub-trees from bottom to top with the roots whose classification errors are more than 50%.

Majority voting mechanism for classification: Regarding the classification in our algorithm, we use the majority voting to classify each testing instance. More specifically, each testing instance $(x'_i, y'_i) \in D'$ first traverses the decision tree from the root to a corresponding leaf in each tree T_k. Second, we adopt the Naïve Bayes Classifier to predict the label of x'_i, namely the predicted label $y'_i = \max\{y'_i | sum_{NB}(y'_i) > sum_{NB}(y'_k), i \neq k, y'_i \in Y\}$. Third, we vote for the current instance in the majority class method. Finally, we can compute the classification error based on an accumulated sum of a loss function between the prediction value y'_i and the observed value y'_i, namely, $E = \sum_{i=1}^{|D'|} L(y'_i, y_i)$. The final error rate is given by $e_{D'} = E/|D'|$.

3.2. Algorithm analysis

In this subsection, we analyze our proposed algorithm in the following dimensions, such as the parameter estimation, the threshold in the concept drifting detection, the generalization error of our model without considering concept drift, and the generalization error of our model considering the concept drift and noise.

Parameter estimation in Hoeffding Bounds inequality: Considering the parameters in Hoeffding Bounds inequality, the smaller the value of ε, the more the predictive accuracy in the current tree, but it will lead to a larger value of δ, namely a lower confidence. Hence, an optimal solution should achieve a good trade-off between ε and δ. Meanwhile, our study demonstrates that the greater value of n_{min} indicates the smaller value of ε but the more computational complexity at the *numerical* nodes. However, an experimental conclusion [43] reveals that the model performs sufficiently well if the value of n_{min} is set to 200. For example, supposing the value of ε/R is 0.01, the value of δ is only 0.0393 in this case, i.e., the confidence is more than 96%. However, in this paper, we specify $n_{min} = 200$ and $\delta = 10^{-7}$ to ensure that the cut-point evaluated in the current model learned from finite data is close to the case learned from all streaming data (with the confidence more than 99%).

Threshold estimation in the detection of concept drifts: According to the principle of statistical quality control [44], the concept drift means the case of out-of-control in the production process. There is a representative assumption that the distribution of the quality characteristic is normal and 2-σ or 3-σ control limit is popular to use in [45–47]. With the influence from these work in our thinking, we assume that errors of classification follow a normal distribution and we also use the control limit of 2-σ or 3-σ to partition concept drifts and noise. Actually, our studies in different values varying from 0.1 to 3 with an interval of 0.1 also confirm that the values of $c_1 = 2$ and $c_2 = 3$ perform effectively in the concept drifting detection. More details refer to the 5th paragraph in Section 4.2.

Parameter estimation on h and N relevant to the classifier stability: The classifier stability of the random decision trees is relevant to two important parameters, namely the maximum height of decision tree-h and the number of trees-N. According to the empirical studies in [4], when we limit the random tree's depth to be $h = |A|/2$, i.e., the depth of tree is half of the number of attributes, we create the most diversity. It is beneficial to improve the accuracy of the random decision tree in the classification. Meanwhile, in terms of the analysis on the number of trees in [4,43,29], we know that it is in effect a random sample from a population of trees of a given depth (e.g., $|A|/2$), because each tree is constructed randomly. The averaging of multiple trees is to approximate the true mean of a large sample. When each attribute is binary, the population size is $2^{|A|/2}$. The random decision tree algorithm can be recast as a statistical sampling problem. The average probability is the sample mean using N trees. The problem is to decide if it is worthwhile to have larger samples, i.e., more random trees. According to the statistical conclusion, it is found that a random sample as small as $N=10$ gives very good result. When $N=30$, the sample's mean (i.e., average probability in our case) usually has small error with at least 99.7% confidence. Because of the "error-tolerance" property in optimal decision making, $N=10$ is probably sufficient for most applications. Because our algorithms are developed from random decision trees in [4], thus we can also get the similar conclusion. In addition, we use the p-value [48] returned by t-test to measure the accuracy of the ensemble random decision trees in the classification. Under the null hypothesis the average classification accuracy is not higher than 50% (the default probability for the data with the binary class labels), assuming that ρ is the p-value returned by the t-test for one mean and α is a given significance level (e.g., $\alpha=0.05$). We can get that the null hypothesis is rejected due to $\rho < \alpha$ regarding the experimental results on the testing data sets. That is, our algorithm can ensure the accuracy of the classification higher than the default probability at least given the significance level α. In the following, we also give the analysis on the generalization error of our algorithm in theorem.

Estimation on generalization error: Breiman pointed out that the upper bound of the generalization error-PE for a model of random tree ensembling [23] can be written as

$$PE \leq \overline{p}(1-s^2)/s^2 = \overline{p}(1/s^2 - 1) \quad (2)$$

It shows that the generalization error of ensemble classifiers is in direct proportion to the value of $\overline{p}$ (i.e., the correlation among trees) while it is proportional to the value of s (i.e., the strength of each individual tree) inversely. Evidence reveals that the evaluation bound on generalization error is applicable to ensemble classifiers including complete-random trees [49]. Therefore, it is also suitable for our random ensemble decision trees. The analysis is as follows. According to the definitions on three strategies of *random feature selection* mentioned above, all trees in our ensemble model are created with completely random selection on split-attributes, the correlation among trees is hence lower, namely ensuring a smaller value of $\overline{p}$. However, with respect to the value of s, it is non-deterministic due to the different randomization in each individual model. But we know that the higher predictive accuracy in a model infers the more likelihood that this model is optimal [29,25]. Therefore, to evaluate the strength of our model, it is sufficient to take into account of the probability that this model is optimal. More specifically, suppose a database has only one relevant attribute a_i and the remaining $d-1$-attributes are irrelevant (noisy), there are two extreme cases concluded below.

Case 1 (The *Best* Case): There are pure discrete-only attributes in the database and all of the discrete attributes are binary. Each decision path from the root to a leaf is completely independent.

Case 2 (The *Worst* Case): All attributes in the database are numerical. Each node in trees only generates two children branches at most. And the sampling mechanism with replacement is used in the generation of trees.

In both cases, probabilities of an attribute to appear in the L th level of the decision path are defined in the following equations:

$$Case\ 1: p_{max} = \frac{d-1}{d} \cdot \frac{d-2}{d-1} \cdot \cdots \cdot \frac{d-L}{d-L+1} \cdot \frac{1}{d-L} = \frac{1}{d} \tag{3}$$

$$Case\ 2: p_{min} = \frac{d-1}{d} \cdot \frac{d-1}{d} \cdot \cdots \cdot \frac{d-1}{d} \cdot \frac{1}{d} = \left[1 - \frac{1}{d}\right]^{L-1} \cdot \frac{1}{d} \tag{4}$$

Thus, we can get the probability that there is no one path to involve the only relevant attribute a_i in our ensemble model, namely the generalization error below:

$$PE \le (1-p)^{N \cdot 2^h - 1} (p_{min} \le p \le p_{max}) \tag{5}$$

Correspondingly, we can get the least probability for at least one path to involve the only relevant attribute a_i in Eq. (6) if and only if the value of p is equal to that of p_{min}:

$$LP = 1 - (1 - p_{min})^{N \cdot 2^h - 1} \tag{6}$$

In other words, our ensemble model is optimal with the confidence of LP at least. For instance, given $h = |A|/2$, $N = 10$ and $|A| = 100$, to ensure the value of LP up to 0.5, the number of decision paths should amount to 7 in a single decision tree. Meanwhile, to guarantee the probability of LP no less than 0.9, additional 17 decision paths are required in a single decision tree at most. These data show that it is actually not difficult to meet these constraints.

Estimation on generalization error impacted from concept drifts and noisy data: Generalization error in Eq. (2) is obtained on the assumption that the distribution of training data is independent and identically distributed. However, it is hard to hold in the environment of concept drifting data streams. Thus, considering the impact from concept drifts, we give an infimum bound of generalization error for our random ensemble model. First of all, we divide the training data into small sequences $\{D_t\}$ as many concepts and maintain each sequence (concept) containing sufficient instances (e.g., no less than 30). In this paper, each sequence indicates a drifting check period. Therefore, we could obtain the generalization error on each sequential chunk in the following equation:

$$PE = P_{T_t, D_t}(P_{D_t}(V(T_t, D_t) = Y) - \max_{j \ne Y} P_{D_t}(V(T_t, D_t) = j) < 0) \tag{7}$$

where T_t specifies the current decision tree ensembling and each tree is generated or updated with the data chunk set $\{D_k, 1 \le k \le t\}$. V refers to the voting function, which takes votes in the method of majority-class after seeing the data chunk-D_t. In terms of Eq. (7), let $P_{D_t}^{T_t} = P_{D_t}(V(T_t, D_t) = Y) - \max_{j \ne Y} P_{D_t}(V(T_t, D_t) = j)$, the worst estimation on the generalization error can be written as Eq. (8), namely the maximum probability of $P_{D_t}^{T_t} < 0$. Details of proof in Eq. (8) refer to [27]:

$$PE^* \le \max(P_{(T_t, D_t)}(P_{D_t}^{T_t} < 0)), 1 \le t \le |D| \tag{8}$$

Because the inequations of $PE^* \ge P_{(T_t, D_t)}(P_{D_t}^{T_t} < 0)$ and $\max_{j \ne Y} P_{D_t}(V(T_t, D_t) = j) \le 1 - P_{D_t}(V(T_t, D_t) = Y)$ hold, we could hence transform Eq. (8) into the following equation:

$$PE^* \ge P_{(T_t, D_t)}(P_{D_t}(V(T_t, D_t) = Y) \le 0.5) \tag{9}$$

Meanwhile, in the above analysis of the probability of an optimal ensemble model, it is considered as an estimation of the classification ability. Thus, Eq. (9) is further written as Eq. (10), whose generalization error specifies the probability of $LP \le 0.5$:

$$PE^* \ge P_{(T_t, D_t)}(LP \le 0.5) \tag{10}$$

In Eq. (10), it explicitly reveals the following. On one hand, the higher the probability that an ensemble model is optimal, the lesser the generalization error. On the other hand, the generalization error is closely related to the values of $|A|$, N and h corresponding to the definition of LP. Therefore, for adaptation to concept drifts, it is feasible to adjust the heights of trees by pruning sub-branches.

In addition, according to the theorem on learning from noisy samples proposed in [50], if a sequence σ of m instances is drawn, where the instance size m satisfies

$$m \ge \frac{2}{\epsilon^2 (1-2\eta)^2} \ln(2N/\delta) \tag{11}$$

where ϵ is the worst-case classification error rate of a hypothesis, η is an upper bound on the noise rate, N is the number of hypotheses, and δ is the confidence, then a hypothesis H_i that minimizes disagreement with σ will have the PAC property, shown in Eq. (12) as follows:

$$Pr[d(H_i, H^*) \ge \epsilon] \le \delta \tag{12}$$

where $d(,)$ is the sum over the probability of elements from the symmetric difference between the two hypothesis sets H_i and H^* (the ground-truth). Let $c' = 2\mu \cdot \ln(2N/\delta)$, where μ makes Eq. (11) hold equality. Then Eq. (11) becomes

$$m = \frac{c'}{\epsilon^2 (1-2\eta)^2} \tag{13}$$

According to Eq. (12), δ is a constant, then the hypothesis H_i is more close to the ground-truth H^* when ϵ decreases along with the training process. Correspondingly, Eq. (13) could be expressed into Eq. (14) as follows:

$$\epsilon^2 = \frac{c'}{m(1-2\eta)^2} \tag{14}$$

According to the above analysis, we assume that the generalization error of our algorithm is linearly in proportion to the classification error (e.g., ϵ), namely we can represent it as $\epsilon = k \cdot PE^*$ (k is constant coefficient, $k > 0$). Thus, Eq. (14) could be expressed into Eq. (15) as follows:

$$PE^* = \frac{\sqrt{c'}}{k\sqrt{m}|1-2\eta|} \tag{15}$$

Eq. (15) shows that as the noisy rate η increases (let $\eta < 0.5$ aim to maintain $1 - 2\eta > 0$), the generalized error of our model is increasing, namely the classification error is increasing. This conclusion will be validated in experiments.[1]

4. Experiments

To validate the efficiency and effectiveness of our EDTC algorithm, we conduct large sets of experiments in two cases. Firstly, we ignore the mechanism of concept drifting detection in EDTC. In this case, EDTC enables handling data streams only, called the algorithm of EDT. We compare our EDT algorithm with several baseline methods for data streams in Section 4.1. Secondly, we compare our EDTC algorithm with several algorithms for concept drifting data streams in Section 4.2. All competing algorithms are summarized in Table 1.

[1] In our experiments, we only give the experimental results varying with the noise rate from 0% to 10%, because they have revealed the same conclusion.

Table 3
Prediction Results on UCI databases.

Noise (%)	Mean error rate ± variance (%)										
	VFDTc	SRMTDS	PegastronBag	PerceposBag	RTBag	EDT-RHB	EDT-RIG	EDT-RAN	OzaBag	OzaBoost	OCBoost
Connect-44k-22k-D-42											
0	33.89 ± 0.0	28.74 ± 1.01	33.89 ± 0.0	33.89 ± 0.0	33.19 ± 0.12	**22.24 ± 0.73**			26.82 ± 0.0	26.45 ± 0.0	36.92 ± 0.0
5	33.89 ± 0.0	32.01 ± 1.01	80.48 ± 0.0	33.89 ± 0.0	32.39 ± 0.23	**23.13 ± 0.75**			27.28 ± 0.0	28.74 ± 0.0	38.36 ± 0.0
10	33.89 ± 0.0	32.57 ± 0.92	80.48 ± 0.0	33.89 ± 0.0	32.59 ± 0.35	**25.49 ± 0.84**			28.58 ± 0.0	29.86 ± 0.0	39.52 ± 0.0
Census-190k-90k-CD-41											
0	18.33 ± 0.0	14.65 ± 0.30	47.13 ± 0.0	47.13 ± 0.0	10.40 ± 0.21	**7.03 ± 0.34**	7.56 ± 0.49	7.80 ± 0.66	14.69 ± 0.0	18.45 ± 0.0	23.35 ± 0.0
5	20.88 ± 0.0	15.92 ± 0.55	47.13 ± 0.0	47.13 ± 0.0	13.04 ± 0.28	**7.75 ± 0.61**	8.37 ± 0.47	8.53 ± 0.56	15.76 ± 0.0	20.31 ± 0.0	25.12 ± 0.0
10	24.17 ± 0.0	17.04 ± 1.01	47.13 ± 0.0	47.13 ± 0.0	14.79 ± 0.25	**8.16 ± 0.94**	9.09 ± 0.59	9.24 ± 0.61	17.87 ± 0.0	22.23 ± 0.0	27.78 ± 0.0
Covertype-382k-199k-CD-54											
0	27.38 ± 0.0	25.96 ± 1.30	51.23 ± 0.0	51.23 ± 0.0	29.84 ± 0.40	19.45 ± 0.94	19.87 ± 0.99	**18.82 ± 0.66**	37.75 ± 0.0	37.93 ± 0.0	57.46 ± 0.0
5	45.32 ± 0.0	42.43 ± 2.3	51.23 ± 0.0	51.23 ± 0.0	31.63 ± 0.67	23.25 ± 1.01	22.58 ± 0.97	**22.77 ± 0.68**	39.79 ± 0.0	39.75 ± 0.0	58.57 ± 0.0
10	50.58 ± 0.0	42.42 ± 2.53	51.23 ± 0.0	51.23 ± 0.0	31.52 ± 0.55	29.68 ± 1.21	28.13 ± 0.89	**27.67 ± 0.85**	40.49 ± 0.0	63.60 ± 0.0	67.3 ± 0.0
LED-500k-250k-D-24											
0	0 ± 0.0	0 ± 0.0	79.99 ± 0.0	0 ± 0.0	0 ± 0.0	0 ± 0.0			0 ± 0.0	0 ± 0.0	79.99 ± 0.0
5	20.62 ± 0.0	14.51 ± 0.45	79.99 ± 0.0	15.05 ± 0.0	13.29 ± 0.0	**12.83 ± 0.43**			13.41 ± 0.0	24.45 ± 0.0	79.99 ± 0.0
10	25.42 ± 0.0	16.83 ± 0.43	79.99 ± 0.0	22.89 ± 0.0	17.87 ± 0.0	**13.75 ± 0.62**			16.02 ± 0.0	42.25 ± 0.0	79.99 ± 0.0
Waveform21-500k-250k-C-21											
0	20.18 ± 0.0	18.21 ± 0.22	50.38 ± 0.0	14.92 ± 0.0	19.84 ± 0.0	13.78 ± 0.55	**13.29 ± 0.13**	13.67 ± 0.56	18.02 ± 0.0	19.07 ± 0.0	48.12 ± 0.0
5	24.32 ± 0.0	18.65 ± 0.35	57.97 ± 0.0	17.56 ± 0.0	21.78 ± 0.0	16.30 ± 0.34	15.82 ± 0.49	**15.57 ± 0.66**	18.59 ± 0.0	19.89 ± 0.0	48.20 ± 0.0
10	30.56 ± 0.0	19.09 ± 0.55	62.71 ± 0.0	19.65 ± 0.0	21.87 ± 0.21	17.64 ± 0.31	17.34 ± 0.40	**17.02 ± 0.59**	20.01 ± 0.0	21.94 ± 0.0	48.72 ± 0.0
Waveform40-500k-250k-C-40											
0	23.24 ± 0.0	14.78 ± 0.21	54.64 ± 0.0	15.29 ± 0.0	21.31 ± 0.0	15.34 ± 0.12	**14.50 ± 0.34**	15.01 ± 0.33	17.80 ± 0.0	17.60 ± 0.0	46.54 ± 0.0
5	25.64 ± 0.0	17.41 ± 0.55	55.92 ± 0.0	17.97 ± 0.0	21.85 ± 0.0	17.65 ± 0.28	**16.39 ± 0.40**	17.73 ± 0.35	19.08 ± 0.0	19.02 ± 0.0	47.36 ± 0.0
10	30.67 ± 0.0	24.37 ± 0.43	57.63 ± 0.0	20.31 ± 0.0	22.26 ± 0.10	18.24 ± 0.12	18.64 ± 0.19	**18.19 ± 0.28**	19.96 ± 0.0	19.86 ± 0.0	48.11 ± 0.0

Table 4
Time overheads on UCI databases.

Training + testing time (s)										
VFDTc	SRMTDS	PegasosBag	PerceptronBag	RTBag	EDT-RHB	EDT-RIG	EDT-RAN	OzaBag	OzaBoost	OCBoost
Connect-44k-22k-D-42										
3+3	3+3	2+1	3+1	3+1	3+4			2+3	8+1	9+3
Covertype-382k-199k-CD-54										
12+7	5+25	13+5	44+18	138+5	12+14	12+14	12+16	15+70	135+ 10	145+66
Census-190k-90k-CD-41										
9+5	20+15	10+8	10+8	200+50	15+6	16+6	16+6	8+13	27+9	32+12
LED-500k-250k-D-24										
1+8	1+20	8+5	53+21	37+5	10+14			71+1	127+27	160+98
Waveform21-500k-250k-C-21										
49+15	32+20	7+1	20+3	110+5	30+30	31+30	32+30	20+10	110+2	134 53
Waveform40-500k-250k-C-40										
89+24	35+8	18+16	35+16	600+17	33+18	33+19	33+20	30+19	208+14	240+89

All experiments are performed on a P4, 3.00 GHz PC with 1G main memory, running Windows XP Professional. Algorithms of SRMTDS, MSRT and ERDTC are implemented by us while others are from the open source of MOA [40] (Massive Online Analysis: a software environment for implementing algorithms and running experiments for online learning from data streams). In our algorithm, three strategies of *random feature selection* are designed for the solution to the cut-points of nodes with numerical split-attributes, they take no effect on the databases with discrete-only attributes. Thus, we only provide a copy of experimental values in the following experiments.

4.1. Evaluations on streaming data without concept drift

In this part, synthetic data sets used in our experiments include the data sets of LED and Waveform, plus some additional medium-sized data sets from UCI. All of them are also simulated as experimental data in [52,11]. The data sets of LED/Waveform are generated in the LED/Waveform generator from MOA. The sizes of training sets of LED and Waveform are set to 500k while the magnitude of the testing set contains 250k. To validate the robustness to noise of our algorithm, we respectively add 5% and 10% label noise for all of the above data sets while maintaining the original data sets (namely with 0% noise data). In the following experiments, parameter values in EDT are set to $\delta = 10^{-7}$, $\tau = 0.05$, $h = |A|/2$, $MP = 500\text{k}$ (1k = 1000), $MS = 200$ M and $N = 10$ while parameter values in competing algorithms are used with the default parameter settings in the corresponding papers. All experimental results are averaged over 20 runs.

Before the performance evaluations of our algorithms, we first specify two important parameters, namely the size of the data chunk $|D_i|$ and the number of minimum split-examples n_{min}.

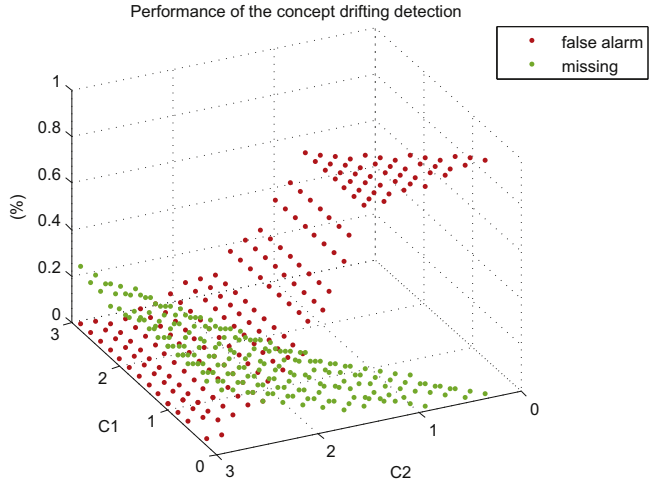

Fig. 3. Performance on *False alarm* and *missing* varying with values of c_1 and c_2.

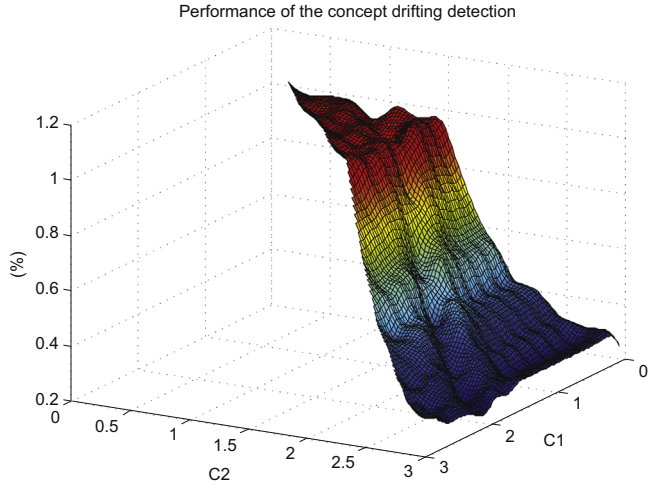

Fig. 4. Performance on the sum value of *False alarm* and *missing* varying with values of c_1 and c_2.

Fig. 2a reports the performance of EDT_RAN varying with values of $|D_i|$ from 0.1k to 1k (1k=1000) with a 0.1k variance each time. In this figure, we specify the value of n_{min} by an invariable value (e.g., 0.3k). From the experimental results, we can observe that as the value of D_i increases, the classification error is increasing. This is because the larger the size of D_i, the higher probability that the data chuck contains more class labels or novel labels, it indicates the lower classification accuracy predicted in the currently generated decision tree. Thus, the size of data chuck D_i should not be larger, an optimal value is set to $D_i = 0.2$k in the following experiments.

Fig. 2b reports the performance of EDT_RAN varying with values of n_{min} from 0.1k to 1k with a 0.1k variance each time. From the experimental results, we can see that as the value of n_{min} increases, the classification error presents in a light variance, which is limited to the bound of (0%, 3.54%). As we know that the larger value of n_{min} indicates the more computation cost when splitting and the fewer instances collected at leaves, which probably results in the reduction on predictive accuracies. Thus, it is better to select a smaller value of n_{min}. The vast majority of experiments show that a candidate optimal value of n_{min} is set to 0.2k. We can also get the above conclusion in EDT-RHB and EDT-RIG.

To conduct the performance analysis among all comparing algorithms systematically, we employ Friedman test [56] widely accepted as the favorable statistical test for comparisons of multiple algorithms over a number of data sets [57]. Given k comparing algorithms and N data sets, let r_i^j denote the rank of the jth algorithm on the ith data set. Let $R_j = (1/N)\sum_{i=1}^{N} r_i^j$ denote the average rank for the jth algorithm, under the null hypothesis, the following Friedman statistic F_r will be distributed according to the F-distribution with $k-1$ numerator degrees of freedom and $(k-1)(N-1)$ denominator degrees of freedom:

$$F_r = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}, \quad \text{s.t.} \quad \chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right]$$

We can get the Friedman statistics F_r using our EDT algorithms and other competing algorithms as shown in Table 2. And we can see that at significance level $\alpha = 0.05$, the null hypothesis of "equal" performance among the comparing algorithms is clearly rejected in terms of each evaluation metric. Consequently, we require proceeding with certain *post hoc test* to further analyze the relative performance among the comparing algorithms. As we are interested in whether the proposed approaches achieve competitive performance against other state-of-the-art approaches, we employ the *Bonferroni–Dunn test* [58] to serve the above purpose by treating our approaches as the control approaches. Here, the difference between the average ranks of our EDT algorithm and one comparing algorithm is compared with the following critical difference (CD): $CD = q_a\sqrt{k(k+1)/6N}$. For Bonferroni–Dunn test, we have CD=5.013 (k=11, N=7[2]) at significance level α=0.05. Accordingly, the performance between our EDT algorithm and one comparing algorithm is deemed to be significantly different if their average ranks over all data sets differ by at least one CD.

Fig. 5 illustrates the CD diagrams [56] on the evaluation measures of error rate and time overhead, where the average rank of each competing algorithm is marked among the axis, namely lower ranks to the right. In each subfigure, any competing algorithm whose average rank is within one CD to that of the best algorithm is interconnected with a thick line. Otherwise, any algorithm not connected with the best algorithm is considered to have significantly different performance between each other. From Fig. 5, we can see that our EDT-RIG algorithm is the control algorithm, and all EDT algorithms perform much better as compared with other competing algorithms on the evaluation measure of error rate, while they are comparable to the control algorithm (PegasosBag) on the evaluation measure of time overhead. Meanwhile, we can read more details of experimental results in the following paragraphs.

Predictive accuracy: In this set of experiments, we first compare the predictive ability in our EDT algorithms against baseline ones varying with the noisy data sets. A summary of the experimental results from UCI databases in Table 3 shows that with the increasing of the noisy rates, the performance of our algorithms is gradually degrading, but it still performs best as compared with all of baseline algorithms. The least improvement of predictive accuracy in our algorithm is 3% on average while the most improvement is up to 35%. The reasons are analyzed below. The performance of our algorithms impacted from the noisy data lie in the split-test at the continuous nodes and the classification using the Naïve Bayes classifier at leaves. As the noisy rates increase, the accuracy of the cut-points at the continuous nodes and the accuracy of the statistics relevant to the classes in the Naïve Bayes classifier will degrade, which lead to the lower prediction accuracy on the test data. However, it is still lighter compared to the impact

[2] The data sets involved here include six benchmark data sets with 5% noisy data and one real data set from Yahoo shopping data.

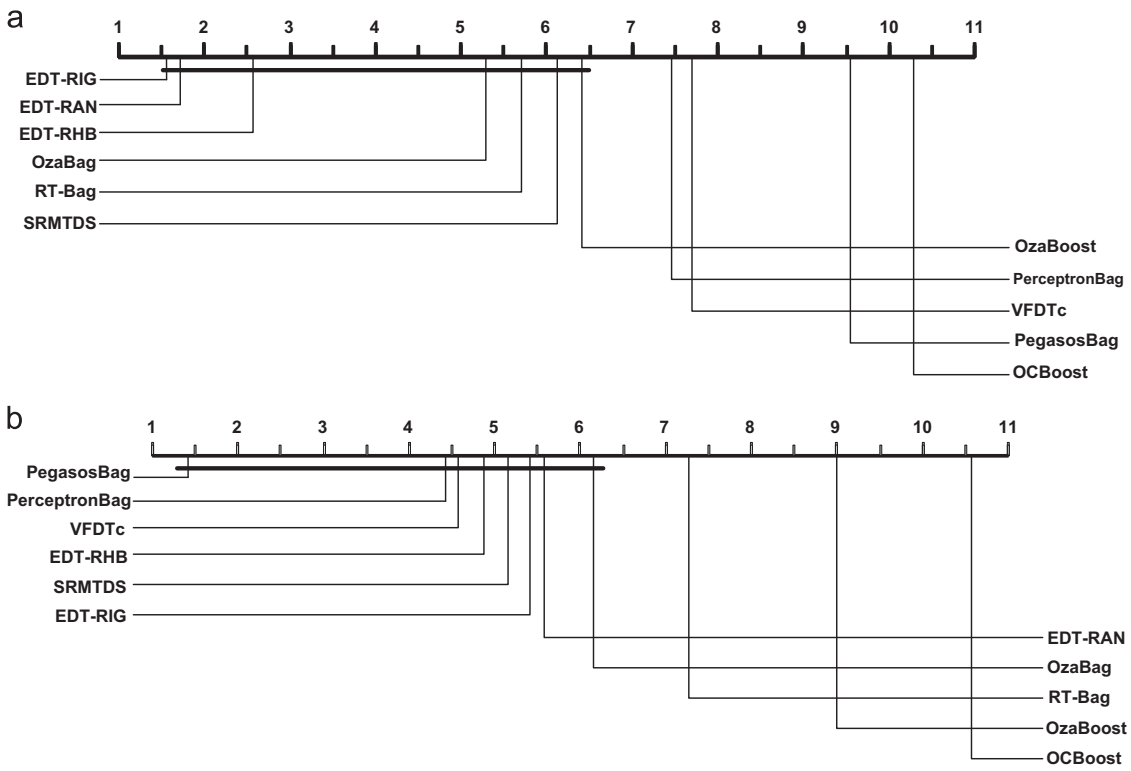

Fig. 5. Our EDT approaches against other competing algorithms with the *Bonferroni-Dunn test*. Algorithms not connected with our best approach in the CD diagram are considered to have significantly different performance from the control algorithm (significant level $\alpha=0.05$).

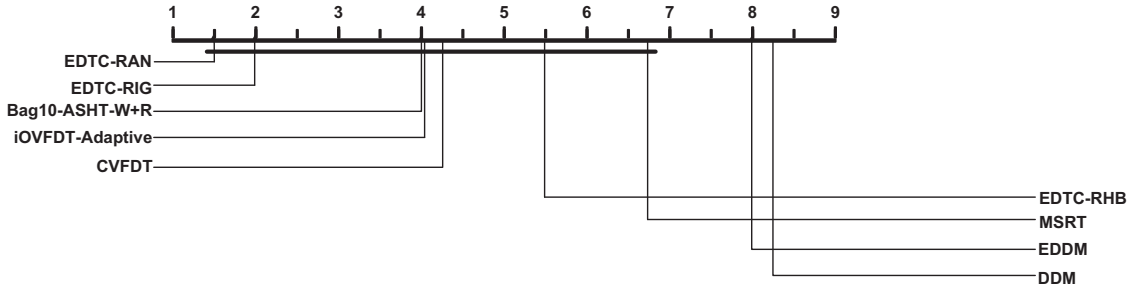

Fig. 6. Our EDTC approaches against other competing algorithms with the *Bonferroni-Dunn test*. Algorithms not connected with our best approach in the CD diagram are considered to have significantly different performance from the control algorithm (significant level $\alpha=0.05$).

of the noisy data on other baseline algorithms. This is because the Naïve Bayes classifier-based algorithms (such as OzaBag, OzaBoost and OCBoost), the SVM-based algorithm (such as PegasosBag), the perceptron-based algorithm (such as PerceptronBag) and the Hoeffding tree-based algorithms (such as VFDTc and RTBag) almost require the information of class distributions at each step in the whole modeling, if the class distributions have the noise, the impact will be more severe compared to our algorithms impacted from the noisy data locally. In fact, some algorithms such as PegasosBag and OCBoost even do not perform well on the data sets without noise. The reason is that PegasosBag is more suitable for data sets with numerical attributes and with linear data distributions between attributes and labels. OCBoost in [55] is more suitable for the data sets with binary classes. However, the data sets used in experiments do not follow these assumptions well.

Runtime overhead: Table 4 reports the experimental results on the runtime consumption. We can see the following: (i) There are no obvious variances of the total time overheads for three EDT algorithms. (ii) As compared with the baseline algorithms, on one hand, the testing time overheads in EDT consume heavier than baseline algorithms in general due to the Naïve Bayes classifier used in the classification. However, if the number of the base classifiers in the generated ensemble model is sufficiently large, the testing time overheads of baseline algorithms (such as RTBag, OzaBoost and OCBoost) will consume heavier than our algorithms. On the other hand, our algorithms are comparable to the algorithms of VFDTc, SRMTDS, PerceptronBag and OzaBag and they are faster than the algorithms of RTBag, OzaBoost and OCBoost except of the PegasosBag algorithm. The reasons are analyzed below. In general, the time overhead of a split-test on the discrete node is much lighter than that on the continuous node for the informed split-test, thus, on the data sets with purely discrete attributes or hybrid attributes, the training time overhead of VFDTc is lower than our algorithms, while on the data sets with purely numerical attributes, it consumes heavier. Regarding the SRMTDS algorithm, it is similar to EDT-RHB in the training, the training time overhead is hence similar. Regarding the algorithms of PerceptronBag, OzaBag and PegasosBag, the base classifiers of PerceptronBag and OzaBag are types of linear classifiers, while PegasosBag is a primal estimated sub-gradient solver for SVM whose time overhead does not depend directly on the size of the training set. Thus, their time

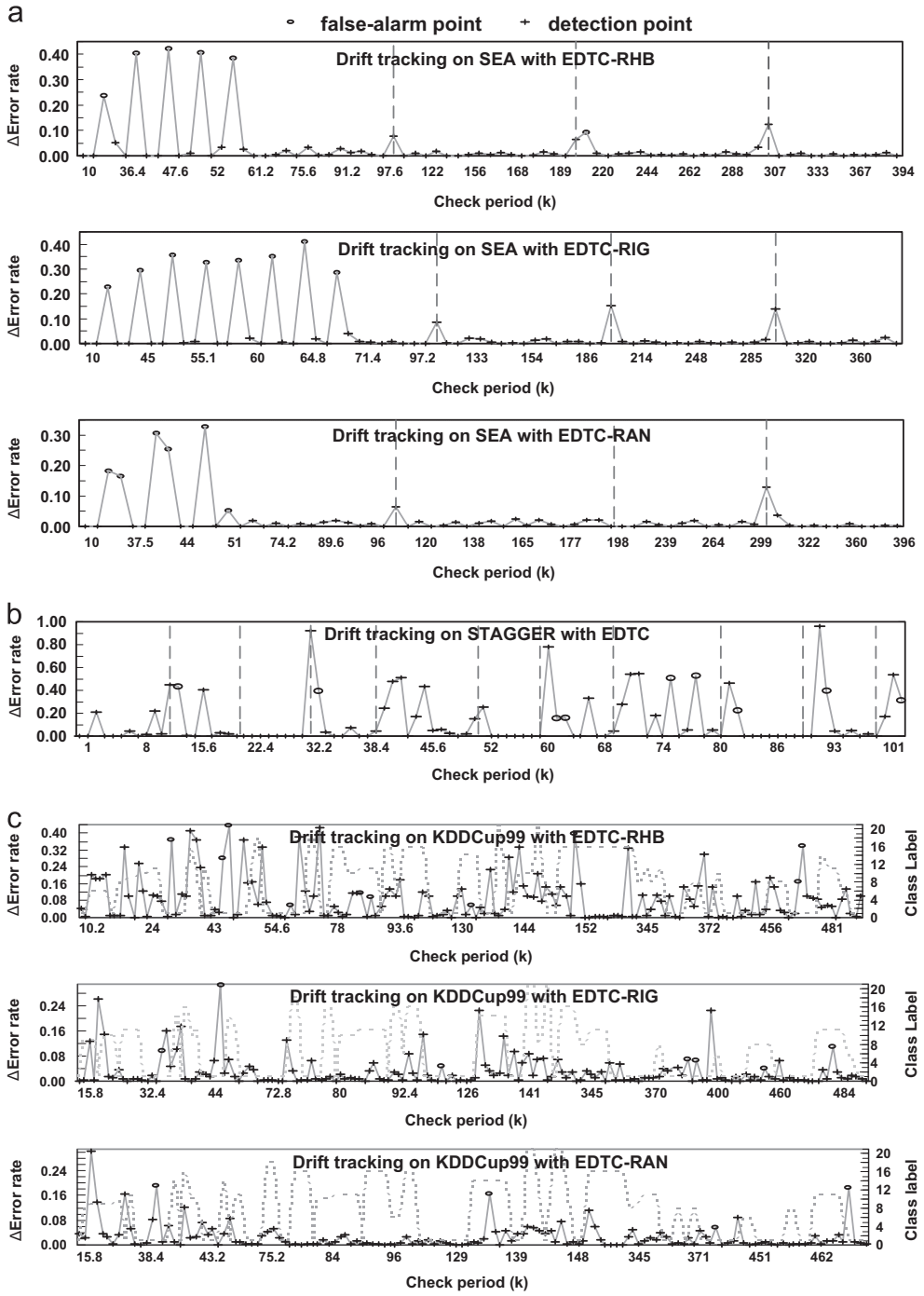

Fig. 7. Drift tracking on three benchmark database.

overheads perform as fast as even superior to our algorithm. Regarding the RTBag algorithm, it needs determining the cut-points from several randomly selected available attributes like random forests in [23], which consumes more time overheads in the split-tests compared to our algorithms with only once split-test. Regarding the boost-based algorithms, they require multiple iterations in the generation of models due to the weight adjusting, thus, they consume much more time overheads than our algorithms.

4.2. Evaluations on concept drifting data sets

In this section, we use both benchmark concept drifting databases and real streaming data to verify the effectiveness of EDTC in the concept drifting detection. Meanwhile, we also observe the performances of our algorithm on the predictive accuracy and the time overhead. Before giving the details of each performance, we first introduce the characteristics of benchmark databases and parameter settings below.

4.2.1. Benchmark databases

SEA: The database of SEA was first described in [6], which is a well-known data set of concept shift with numerical attributes only. It is composed of a three-dimensional feature space with two classes. All three features have values between 0 and 10 while only the first two features are relevant. Those points are partitioned into four chunks with different concepts. In each chunk, a data point

belongs to class 1 if $f_1+f_2 \le \theta$, where f_1 and f_2 represent the first two features and θ is a threshold between a pair of concepts. In this database, there are four thresholds of 8, 9, 7 and 9.5 to divide data chunks. We use the data generator of SEA in MOA to generate three data sets, where each test data set contains 100k-sized instances, and each training set contains four concepts and each concept includes 100k-sized instances. Meanwhile, the label noise of the data set is set to 0%, 5% and 10%.

STAGGER: STAGGER is a standard database of concept-shifting data streams [39] to test the abilities of inductive algorithms. This database consists of three attribute values: $color \in \{green, blue, red\}$, $shape \in \{triangle, circle, rectangle\}$ and $size \in \{small, medium, large\}$. There are three alternative underlying concepts, *A*: if $color = red \wedge = small, class = 1$; otherwise, $class = 0$; *B*: if $color = green \vee shape = circle$, $class = 1$; otherwise, $class = 0$; and *C*: if $size = medium \vee large$, $class = 1$; otherwise, $class = 0$. We use the database generator of STAGGER in MOA to generate three sets of training data and test data with 500k-sized instances. The training set includes 0.1k concepts and each concept contains 10k-sized instances. Meanwhile, the label noise of the data set is set to 0%, 5% and 10%.

KDDCup99: KDDCup99 is a database for network intrusion detection, containing 24-class labels and 41-attribute dimensions in total with 34-dimension of numerical attributes. We select this database because it has been simulated as streaming data with sampling change in [31]. In our experiments, this database contains 490k-sized instances with only 12 main class labels. The label noise is set to 0%, 5% and 10%.

Parameter settings: Some parameters in EDTC, such as ε, τ, n_{min} and N, are initialized with the default values mentioned above. For others used in the concept drifting detection, they are set to $c_2=3$, $c_1=2$, $DP=1$k and $SW=10$k respectively. However, the parameters in other related algorithms follow the default settings involved in the corresponding references. Before the performance evaluations of our algorithms, we first specify two important parameters in the concept drifting detection, the higher threshold e_h and the lower threshold e_l, namely the coefficients c_1 and c_2 (Because we set $e_l = c_1\varepsilon$ and $e_h = c_2\varepsilon$ using the Hoeffding Bounds inequality). Figs. 3 and 4 report the performance of EDT_RAN in the concept drifting detection varying with values of c_1 and c_2 from 0.1 to 3 with an interval of 0.1 ($c_2 = c_1 + 0.1$ for each point). We take the performance of EDT_RAN on two important measures of *False alarm* and *missing* as an example to show how to set the values of the coefficients of c_1 and c_2. We know that the smaller the values of *missing* and *False alarm*, the better the performance in the concept drifting. From the experimental results in Fig. 3, we can observe that as the values of c_1 and c_2 increase, EDT_RAN performs worse on the measure of *missing* while performs better on the measure of *False alarm*. To trade-off the performance of the concept drifting detection on these two measures, we find if $c_1 \ge 1$ and $c_2 \ge 1.5$, EDT_RAN performs well on both of measures as shown in Fig. 4 with the sum values of *False alarm* and *missing*. According to the

Table 5
Drifting detection statistics on concept drift databases.

Algorithm	Average number of detections	False alarm (%)	Missing	AvgExam (k)
SEA-400k with 4 concept drifts				
EDTC-RHB	77	6.49	0	7.000
EDTC-RIG	75	10.67	0	6.930
EDTC-RAN	69	7.25	1	6.700
STAGGER-1000k with 100 concept drifts				
EDTC	1019	7.36	14	1.877
KDDCup99-490k with 36 concept drifts				
EDTC-RHB	123	7.32	8	4.166
EDTC-RIG	123	4.87	8	4.968
EDTC-RAN	126	3.17	9	1.963

Table 6
Prediction results on concept drifting databases.

Noise (%)	Mean error rate ± covariance (%)								
	CVFDT	MSRT	DDM	EDDM	EDTC-RHB	EDTC-RIG	EDTC-RAN	Bag10-ASHT–W+R	iOVFDT-Adaptive
SEA-400k-100k-C-3									
0	**8.53 ± 0.0**	12.45 ± 0.78	8.37 ± 0.0	8.36 ± 0.0	9.73 ± 1.41	10.27 ± 0.52	8.81 ± 0.41	9.61 ± 0.0	9.55 ± 0.0
5	12.36 ± 0.0	16.91 ± 0.99	14.44 ± 0.0	14.10 ± 0.0	14.09 ± 1.84	13.32 ± 0.92	**10.15 ± 0.63**	14.02 ± 0.0	13.47 ± 0.0
10	15.21 ± 0.0	18.65 ± 1.01	17.34 ± 0.0	17.32 ± 0.0	15.82 ± 1.75	**14.16 ± 0.87**	14.67 ± 0.54	17.62 ± 0.0	15.54 ± 0.0
KDDCup99-490k-310k-CD-41									
0	23.56 ± 0.0	35.26 ± 2.78	46.22 ± 0.0	46.25 ± 0.0	44.12 ± 1.13	9.12 ± 0.12	9.23 ± 0.14	**6.59 ± 0.0**	8.69 ± 0.0
5	26.28 ± 0.0	43.67 ± 2.63	75.81 ± 0.0	73.19 ± 0.0	73.13 ± 1.51	12.86 ± 0.32	13.12 ± 0.45	**12.37 ± 0.0**	13.57 ± 0.0
10	30.98 ± 0.0	50.31 ± 2.34	78.23 ± 0.0	81.41 ± 0.0	75.68 ± 1.78	**15.34 ± 0.31**	15.67 ± 0.62	17.89 ± 0.0	19.65 ± 0.0
STAGGER-1000k-500k-D-3									
0	22.12 ± 0.0	19.46 ± 0.35	23.68 ± 0.0	22.98 ± 0.0	**18.02 ± 1.15**			20.68 ± 0.0	20.13 ± 0.0
5	43.52 ± 0.0	44.12 ± 1.23	48.36 ± 0.0	47.36 ± 0.0	**32.11 ± 2.13**			44.23 ± 0.0	42.36 ± 0.0
10	61.24 ± 0.0	66.42 ± 0.35	63.58 ± 0.0	62.89 ± 0.0	**40.11 ± 1.98**			61.67 ± 0.0	61.01 ± 0.0

Table 7
Time overheads on concept drifting databases.

Data set	Training+testing time (s)								
	CVFDT	MSRT	DDM	EDDM	EDTC-RHB	EDTC-RIG	RAN	Bag10-ASHT-W+R	iOVFDT-adaptive
SEA-400k-100k-C-3	147+1	30+2	3+1	3+1	6+6	6+6	6+6	16+3	4+1
KDDCup99-490k-310k-CD-41	175+18	92+504	50+40	90+63	85+51	75+201	90+118	480+30	44+27
STAGGER-1000k-500k-D-3	21+6	7+3	5+2	5+2	7+6			20+2	5+3

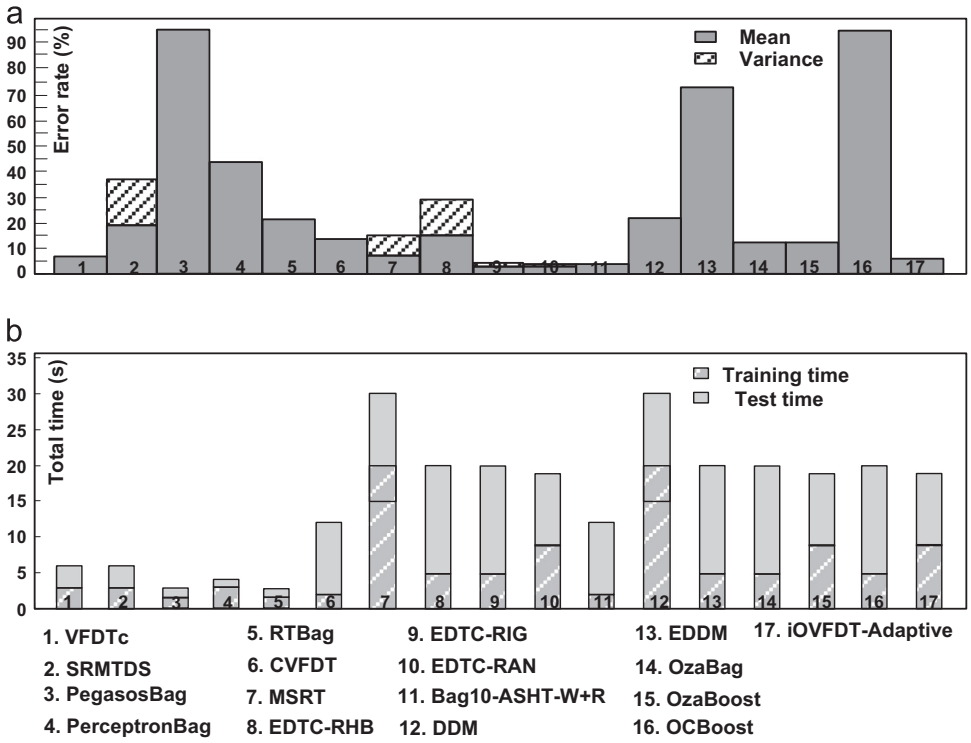

Fig. 8. Classification on the database of Yahoo shopping data.

2-σ or 3-σ control limit in the principle of statistical quality control [44], we select $c_1=2$ and $c_2=3$ as a set of optimal values used in the concept drifting detection.

Predictive accuracy: In this subsection, we first evaluate the predictive ability of EDTC by tracking concept drifts in the training data. Fig. 7a–c presents the cases of tracking concept drifts on three databases of SEA, STAGGER and KDDCup99.[3] To describe tracking curves clearly, Fig. 7b presents ten concepts of STAGGER only. From the observation of tracking curves, we can see that false alarms are prone to appear at the beginning of learning from training data. This is because there are larger fluctuations of error rates classified by a model learned from insufficient training data. In this case, concept drifts are probably considered falsely. In addition, Table 5 reports the statistical results of drifting detection, which show that our algorithm with three strategies of *random feature selection* could detect most of concept changes in a few instances after drift occurs. Meanwhile, according to the values of estimation metrics [59], including *False alarm* (probability that false alarms occur in the drifting detection), *Missing* (number of concepts missed) and *AvgExam* (average count of examples till detection), EDTC-RAN performs best.

On the other hand, we first get the Friedman statistics F_r using our EDTC algorithms and other competing algorithms as shown in Table 2. And we can see that at significance level $\alpha=0.05$, there are no significant variances of the performance in our EDTC algorithms against other competing algorithms regarding the evaluation measure of time overhead (due to $0.1236<2.360$). However, considering the evaluation measure of error rate, the null hypothesis of "equal" performance among the comparing algorithms is clearly rejected. Thus, we consequently proceed with certain *post hoc test* to further analyze the relative performance among the comparing algorithms. In a similar way, we have $CD=5.275$ ($k=9$, $N=4$[4]) at significance level $\alpha=0.05$ for Bonferroni–Dunn test. Meanwhile, Fig. 6 illustrates the CD diagrams on the evaluation measure of error rate. From Fig. 5, we can see that our EDTC-RAN algorithm is the control algorithm, and our algorithms of EDTC-RAN and EDTC-RIG perform much better compared to other competing algorithms, while our EDTC-RHB algorithm does not have significantly different performance compared to the control algorithm. More details of experimental results refer to the following paragraphs.

Meanwhile, Table 6 presents the predictive results on the test data. We can find that (i) if there are no noisy data, our EDTC algorithms are comparable to the baseline ones and sometimes perform a little worse on the predictive accuracy. However, as the noisy rates increase, the predictive accuracy of our algorithms will exceed the baseline ones and the advantage is more obvious. This indicates that the impact from noisy data in our algorithms is weaker than the baseline ones with informed-split tests. Our algorithms are more suitable for the streaming data classification with noisy data. (ii) It should be mentioned that it seems abnormal for EDTC-RHB with the error rate more than 80% on the KDDCup99 dataset. This is resulted from the fact that this database owns a heavily skewed distribution of class labels, namely a data set with the majority class label contains more than 90% instances from this database. Meanwhile, the condition of Hoeffding Bounds inequality in EDTC-RHB impedes the growing of trees. It is prone to generate tree stumps, which leads to a poor performance on the predictive ability.

Runtime overhead: In the other dimension, a set of experiments is conducted to evaluate the overheads of runtime in EDTC. Experimental results shown in Table 7 present that firstly, our EDTC algorithms cannot beat the algorithms of DDM, EDDM and iOVFDT-adaptive in the time overheads of training, but they are faster than the algorithms of

[3] We only give the concept drifting tracking curves on these databases with 5% label noise, because we can get the same conclusion from other sets of experimental results varying with the label noise from 0% to 10%.

[4] The data sets involved here include three benchmark data sets with 5% noisy data and one real data set from Yahoo shopping data.

MSRT, CVFDT and Bag10-ASHT-W+R. The reason is that in the training, the time overheads of our algorithms mainly lie in the split-test on the continuous nodes. This will consume more than DDM and EDDM with the models built on the simple statistics of data distributions. Secondly, in the time overheads of testing, they consume a little heavier compared to all baseline algorithms. It is more obvious on the databases of STAGGER and KDDCup99. This is because our algorithms use the Naïve Bayes classifier in the classification, which will consume more time overheads compared to the baseline algorithms using the majority-class method. Meanwhile, the Naïve Bayes classifier takes more time overhead on the databases with numerical attributes compared to that on the databases with discrete attributes.

Application on web shopping data: When tackling the real-world streaming data, it is hard to judge whether the current data streams carry a potential concept drift or not. Thus, a real data stream from Yahoo shopping databases is obtained via an interface of Yahoo web services [60] to verify the feasibility in our algorithm. This data set contains 113k-sized records and 22-dimension attributes with 16-dimension numerical ones. To mine the relation between the credibility of merchants and possible factors, we define the attribute of "*OverallRating*" as the class label and divide its values into five class labels. In our experiments, we randomly select 2/3 of total records as the training set and the remaining 1/3 as the test set corresponding to the original distribution of class labels. Experimental results shown in Fig. 8 reveal that EDTC-RHB performs worse than EDTC-RIG and EDTC-RAN. The reason is because the condition of Hoeffding Bounds inequality in EDTC-RHB impedes the growing of trees. It is prone to generate tree stumps, which lead to a poor performance on the predictive ability. However, EDTC-RIG and EDTC-RAN perform as well as Bag10-ASHT-W+R, iOVFDT-Adaptive and VFDTc on the predictive accuracy while all of them outperform remaining baseline algorithms very much. For instance, the lowest predictive accuracy and the highest one in EDTC-RIG are improved by 1.28% and 90.76% respectively. These data confirm that EDTC-RIG and EDTC-RAN are more suitable for handling real streaming data compared to EDTC-RHB. In addition, regarding the time overheads, our algorithms are not the fastest algorithms compared to all of 14 baseline algorithms, but the maximum consumption of time in our algorithms is also small (no more than 20 s). It is still faster in the handling of real data streams especially in the training.

5. Conclusion

In this paper, we have proposed an algorithm of random Ensemble Decision Trees for Concept drifting data streams called EDTC. Unlike other random decision trees, three variants of *random feature selection* are developed to determine the cut-points in the incremental growing of trees. Meanwhile, two thresholds defined in Hoeffding Bounds inequality are adopted in tandem with the dynamic adjustment of the drifting check period and the window size to track concept drifts. Experimental studies from synthetic and real world data streams present that regarding the predictive accuracy in the cases with noisy data and overheads of runtime, our algorithm of EDTC performs better than several state-of-the-art online algorithms.

In our future work, we will refine our algorithm using McDiarmid's Bound instead of Hoeffding's bound due to the following weaknesses [61]. First, only numerical data are applicable to Hoeffding's inequality. However, in the general case the data do not have to be numerical. Second, split measures, like information gain and Gini index, cannot be expressed as a sum of elements Y_i, where Y_i $(1 \le i \le N)$ are random variables with real values from a certain distribution. Third, they are using only the frequency of elements. Meanwhile, we will analyze some optimal solutions to further improve the system's computational efficiency especially in the test stage while maintaining its performance. In addition, we will extend the current work regarding how to adapt to gradual concept drifts and how to apply our models into the data streams with labeled data and unlabeled data, because they are challenging and interesting issues for our future work.

Acknowledgments

This work is supported in part by the Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT) of the Ministry of Education, China, under Grant IRT13059, National 863 Program of China under Grant 2012AA011005, the National 973 Program of China under Grant 2013CB329604, the Natural Science Foundation of China under Grants 61273292, 61273297, 61229301, 61070131, 61305063, the Doctoral Fund of Ministry of Education of China under Grant 2013JYBS0632, the Postdoctoral Science Foundation of Hefei University of Technology, Hefei, China, under Grant 2013HGBH0025, the China Postdoctoral Science Foundation under Grant 2014M551801, Industrial Science and Technology Pillar Program of Changzhou, Jiangsu, China, under Grant CE20120026, the Anhui Province Key Laboratory Open Foundation of Affective Computing & Advanced Intelligent Machine under Grant ACAIM150101 and the US National Science Foundation (NSF) under Grant CCF-0905337.

References

[1] G. Johannes, G. Venkatesh, R. Raghu, L. Wei-Yin, BOAT—optimistic decision tree construction, in: Proceedings of SIGMOD'99, 1999, pp. 169–180.
[2] Q.J. Ross, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
[3] J. Shafer, R. Agrawal, M. Mehta, Sprint: a scalable parallel classifier for data mining, in: Proceedings of VLDB'96, 1996, pp. 544–555.
[4] W. Fan, H. Wang, P.S. Yu, S. Ma, Is random model better? On its accuracy and efficiency, in: Proceedings of ICDM'03, 2003, pp. 51–58.
[5] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of KDD'01, 2001, pp. 97–106.
[6] W.N. Street, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of KDD'01, 2001, pp. 377–382.
[7] H. Wang, W. Fan, P.S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of KDD'03, 2003, pp. 226–235.
[8] M. Woźniak, A. Kasprzak, P. Cal, Weighted aging classifier ensemble for the incremental drifted data streams, in: Proceedings of Flexible Query Answering Systems (FQAS'13), 2013, pp. 579–588.
[9] J.Z. Kolter, M.A. Maloof, Using additive expert ensembles to cope with concept drift, in: Proceedings of ICML'05, 2005, pp. 449–456.
[10] K.J. Zico, M.A. Maloof, Dynamic weighted majority: an ensemble method for drifting concepts, J. Mach. Learn. Res. 8 (2007) 2755–2790.
[11] J. Gama, R. Fernandes, R. Rocha, Decision trees for mining data streams, Intell. Data Anal. 10 (2006) 23–45.
[12] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of KDD'00, 2000, pp. 71–80.
[13] M. Scholz, R. Klinkenberg, Boosting classifiers for drifting concepts, Intell. Data Anal. 11 (2007) 3–28.
[14] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavaldà, New ensemble methods for evolving data streams, in: Proceedings of KDD'09, 2009, pp. 139–148.
[15] A. Bifet, G. Holmes, B. Pfahringer, E. Frank, Fast perceptron decision tree learning from evolving data streams, in: Proceedings of PAKDD'10, 2010, pp. 299–310.
[16] N.G. Pavlidis, D.K. Tasoulis, N.M. Adams, D.J. Hand, λ-perceptron: an adaptive classifier for data streams, Pattern Recognit. 44 (2011) 78–96.
[17] J. Gama, P. Kosina, Learning decision rules from data streams, in: Proceedings of IJCAI'11, 2011, pp. 1255–1260.
[18] P. Zhang, B.J. Gao, X. Zhu, L. Guo, Enabling fast lazy learning for data streams, in: Proceedings of ICDM'11, 2011, pp. 932–941.
[19] H. Yang, S. Fong, Incrementally optimized decision tree for noisy big data, in: Proceedings of BigMine Workshop of ACM SIGKDD, 2012, pp. 36–44.
[20] K. Jackowski, Fixed-size ensemble classifier system evolutionarily adapted to a recurring context with an unlimited pool of classifiers, Pattern Analysis and Applications (2013) 1–16. http://dx.doi.org/10.1007/s10044-013-0318-x.
[21] W. Fan, Streamminer: a classifier ensemble-based engine to mine concept-drifting data streams, in: Proceedings of VLDB'04, 2004, pp. 1257–1260.
[22] H. Abdulsalam, D.B. Skillicorn, P. Martin, Streaming random forests, in: Proceedings of IDEAS'07, 2007, pp. 225–232.
[23] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.

[24] H. Abdulsalam, D.B. Skillicorn, P. Martin, Classifying evolving data streams using dynamic streaming random forests, in: Proceedings of DEXA'08, 2008, pp. 643–651.
[25] X. Hu, P. Li, X. Wu, G. Wu, A semi-random multiple decision-tree algorithm for mining data streams, J. Comput. Sci. Technol. 22 (2007) 711–724.
[26] P. Li, X. Hu, X. Wu, Mining concept-drifting data streams with multiple semi-random decision trees, in: Proceedings of ADMA'08, 2008, pp. 733–740.
[27] P. Li, X. Wu, X. Hu, Q. Liang, Y. Gao, A random decision tree ensemble for mining concept drifts from noisy data streams, Appl. Artif. Intell. 24 (2010) 680–710.
[28] P. Li, X. Wu, Q. Liang, X. Hu, Y. Zhang, Random ensemble decision trees for learning concept-drifting data streams, in: Proceedings of PAKDD'11, 2011, pp. 313–325.
[29] W. Fan, On the optimality of probability estimation by random decision trees, in: Proceedings of AAAI'04, 2004, pp. 336–341.
[30] W. Hoeffding, Probability inequalities for sums of bounded random variables, J. Am. Stat. Assoc. 58 (1963) 13–30.
[31] Y. Yang, X. Wu, X. Zhu, Combining proactive and reactive predictions for data streams, in: Proceedings of KDD'05, 2005, pp. 710–715.
[32] T.K. Ho, Random decision forests, in: Proceedings of ICDAR'95, 1995, pp. 278–282.
[33] Y. Amit, D. Geman, Shape quantization and recognition with randomized trees, Neural Comput. 9 (1997) 1545–1588.
[34] T.K. Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (1998) 832–844.
[35] H. Abdulsalam, D.B. Skillicorn, P. Martin, Classification using streaming random forests, IEEE Trans. Knowl. Data Eng. 23 (2011) 22–36.
[36] A. Wang, G. Wan, Z. Cheng, S. Li, An incremental extremely random forest classifier for online learning and tracking, in: Proceedings of ICIP'09, 2009, pp. 1433–1436.
[37] G. Widmer, M. Kubat, Learning flexible concepts from streams of examples: Flora2, in: Proceedings of ECAI'92, 1992, pp. 463–467.
[38] J.a. Gama, G. Castillo, Learning with local drift detection, in: Proceedings of ADMA'06, 2006, pp. 42–55.
[39] J.C. Schlimmer, R.H. Granger Jr., Incremental learning from noisy data, Mach. Learn. 1 (1986) 317–354.
[40] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, Mach. Learn. Res. 11 (2010) 1601–1604.
[41] X. Wu, P. Li, X. Hu, Learning from concept drifting data streams with unlabeled data, Neurocomputing 92 (2012) 145–155.
[42] P. Li, X. Wu, X. Hu, Mining recurring concept drifts with limited labeled streaming data, ACM Trans. Intell. Syst. Technol. 3 (2012) 29–1-32.
[43] P. Li, Q. Liang, X. Wu, X. Hu, Parameter estimation in semi-random decision tree ensembling on streaming data, in: Proceedings of PAKDD'09, 2009, pp. 376–388.
[44] D.C. Montgomery, Introduction to Statistical Quality Control, John Wiley & Sons Inc., New York, 2004.
[45] G. Castillo, J. Gama, P. Medas, Adaptation to drifting concepts, in: Proceedings of EPAI03, 2003, p. 279–293.
[46] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Proceedings of SBIA Brazilian Symposium on Artificial Intelligence, 2004, pp. 286–295.
[47] J. Gama, P. Medas, P. Rodrigues, Learning decision trees from dynamic data streams, in: Proceedings of SAC'05, 2005, pp. 573–577.
[48] L. Dümbgen, P-values for classification, Electron. J. Stat. 2 (2008) 468–493.
[49] F.T. Liu, K.M. Ting, W. Fan, Maximizing tree diversity by building complete-random decision trees, in: Proceedings of PAKDD'05, 2005, pp. 605–610.
[50] D. Angluin, P. Laird, Learning from noisy examples, Mach. Learn. 2 (1988) 343–370.
[51] S. Shalev-Shwartz, Y. Singer, N. Srebro, Pegasos: primal estimated sub-gradient solver for SVM, in: Proceedings of ICML'07, 2007, pp. 807–814.
[52] J. Gama, R. Rocha, P. Medas, Accurate decision trees for mining high-speed data streams, in: Proceedings of KDD'03, 2003, pp. 523–528.
[53] N.C. Oza, S. Russell, Online bagging and boosting, in: Artificial Intelligence and Statistics 2001, Morgan Kaufmann, 2001, pp. 105–112.
[54] R. Pelossof, I. Vovsha, M. Jones, C. Rudin, Online coordinate boosting, in: Proceedings of ICCV Workshops, 2009, pp. 1354–1361.
[55] M. Baena-Garća, J. del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavaldà, R. Morales-Bueno, Early drift detection method, in: Proceedings of the Fourth International Workshop on Knowledge Discovery from Data Streams, 2006.
[56] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[57] M. Zhang, L. Wu, Lift: Multi-label learning with label-specific features, IEEE Trans. Pattern Anal. Mach. Intell. 37 (2015) 107–120.
[58] J. Demšar, Multiple comparisons among means, J. Am. Stat. Assoc. 56 (1961) 52–64.
[59] J. Gama, R. Sebastião, P.P. Rodrigues, Issues in evaluation of stream learning algorithms, in: Proceedings of KDD'09, 2009, pp. 329–338.
[60] Yahoo! Shopping Web Services, ⟨http://developer.yahoo.com/everything.html⟩.
[61] L. Rutkowski, L. Pietruczuk, P. Duda, M. Jaworski, Decision trees for mining data streams based on the McDiarmid's bound, IEEE Trans. Knowl. Data Eng. 25 (2013) 1272–1279.
[62] J. Gama, P. Kosina, Recurrent concepts in data streams classification, Knowledge and Information Systems 40 (3) (2014) 489–507.
[63] M. Wozniak, A. Hybrid, Decision Tree Training Method using Data Streams, Knowledge and Information Systems 29 (2) (2011) 335–347.

Peipei Li is currently a Post-Doctoral Fellow at Hefei University of Technology, China. She received her B.S., M.S., and Ph.D. degrees from Hefei University of Technology, in 2005, 2008, and 2013 respectively. She was a Research Fellow at Singapore Management University from 2008 to 2009. She was a Student Intern at Microsoft Research Asia between August 2011 and December 2012. Her research interests are in data mining and knowledge engineering.

Xindong Wu is a Yangtze River Scholar in the School of Computer Science and Information Engineering at the Hefei University of Technology (China), a Professor of Computer Science at the University of Vermont (USA), and a Fellow of the IEEE and the AAAS. He holds a Ph.D. in Artificial Intelligence from the University of Edinburgh, Britain. He is the Founder and current Steering Committee Chair of the IEEE International Conference on Data Mining and the Founder and current Editor-in-Chief of Knowledge and Information Systems, and an Editor-in-Chief of the Springer Book Series on Advanced Information and Knowledge Processing (AI&KP). He was the Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering from 2005 to 2008. His research interests include data mining, Big Data analytics, knowledge-based systems, and Web information exploration. He has published over 300 refereed papers in these areas in various journals and conferences, including IEEE TPAMI, TKDE, ACM TOIS, DMKD, KAIS, IJCAI, AAAI, ICML, KDD, ICDM, and WWW, as well as 36 books and conference proceedings.

Xuegang Hu is a Professor at the School of Computer Science and Information Engineering, Hefei University of Technology, China. He received his B.S. degree from the Department of Mathematics at Shandong University, China, and his M.S. and Ph.D. degrees in Computer Science from the Hefei University of Technology, China. He is engaged in research in data mining and knowledge engineering.

Hao Wang is a Professor of the School of Computer Science and Information Engineering, Hefei University of Technology, China. He received his B.S. degree from the Department of Computer Science and Engineering at Shanghai Jiao Tong University, China, and his M.S. and Ph.D. degrees in Computer Science from the Hefei University of Technology. His interests are in Artificial Intelligence and Robotics and Knowledge Engineering.