Neurocomputing 295 (2018) 17–28

Contents lists available at ScienceDirect

Neurocomputing

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

An effective pattern-based Bayesian classifier for evolving data stream

Jidong Yuan, Zhihai Wang*, Yange Sun, Wei Zhang, Jingjing Jiang

School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China

ARTICLE INFO

Article history:
Received 1 August 2017
Revised 7 December 2017
Accepted 2 January 2018
Available online 22 February 2018

Communicated by Dr. Haiqin Yang

Keywords:
Data stream
Frequent pattern
Bayesian
Lazy learning

ABSTRACT

One of the hot topics in graph-based machine learning is to build Bayesian classifier from large-scale dataset. An advanced approach to Bayesian classification is based on exploited patterns. However, traditional pattern-based Bayesian classifiers cannot adapt to the evolving data stream environment. For that, an effective Pattern-based Bayesian classifier for Data Stream (PBDS) is proposed. First, a data-driven lazy learning strategy is employed to discover local frequent patterns for each test record. Furthermore, we propose a summary data structure for compact representation of data, and to find patterns more efficiently for each class. Greedy search and minimum description length combined with Bayesian network are applied to evaluating extracted patterns. Experimental studies on real-world and synthetic data streams show that PBDS outperforms most state-of-the-art data stream classifiers.

1. Introduction

Classification based on patterns has attracted significant attention and research effort in recent years [1–7]. Pattern is a subset of data or a set of items, where an item refers to a pair of attribute-value. Frequent patterns (or itemsets) are generated w.r.t. minimum support. Since frequent pattern combines the set of single features non-linearly and indicates more underlying semantics of data [4], except associative classifier [7], it has also been successfully employed to approximate the joint probability of Bayesian classifier [1,2,8]. However, those previous pattern-based classifiers built on traditional datasets cannot adapt to the highly dynamic and complex data stream environment.

A data stream is an infinite sequence of records, generated in a non-stationary environment continuously. Building a pattern-based classifier on data streams differs from traditional pattern mining of static datasets in the following aspects: (1) Each record should be processed quickly in a limited time and memory consumption due to the characteristics of high speed and infinity. (2) Each record can only be examined once and only a small part of the datasets can be stored since of time and space constraints. Therefore the frequency and completeness of mined patterns in data stream cannot be promised. (3) When a classification request occurs, the algorithm needs to respond it in a timely manner and to avoid misclassification. (4) Concept drift may happen in streaming data, and the algorithm must be able to adapt it.

For mining patterns over data stream, state-of-the-art approaches mainly focus on discovering frequent pattern [9] or its variants, such as closed frequent pattern [10,11], maximal frequent pattern [12], high utility pattern [13] and even emerging pattern [14]. Most of the exiting classifiers for non-stationary data stream pay more attention to adaptive tree models [15,16], decision rules [17], ensemble algorithms [18–21] and *k*NN [22]. However, the above algorithms ignore the potential of incorporating pattern in graphical models.

As is well known, Bayesian classification is a graphical model based on Bayes theorem.[1] The major challenge of Bayesian classification lies in the computation of joint probability $P(\mathbf{x}, y)$. The simplest way to address this issue is the Naive Bayes model (as shown in Fig. 1 (*left*)). It assumes that all the attributes x_i are conditional independent given class y_i:

$$\begin{aligned} P(\mathbf{x}, y_i) &= P(x_1, x_2, \ldots, x_m, y_i) \\ &= P(y_i) \cdot P(x_1|y_i) \cdot P(x_2|y_i) \cdots P(x_m|y_i) \end{aligned} \quad (1)$$

Note that although Naive Bayes classifier could be applied on evolving data stream, it cannot adapt to concept drift directly. To alleviate the strong assumption, researchers proposed the conditional dependence model, *e.g.* Bayesian networks [23,24]. The joint probability of Bayesian network shown in Fig. 1 (*middle*) is:

$$\begin{aligned} P(\mathbf{x}, y_i) &= P(x_1, x_2, x_3, x_4, y_i) \\ &= P(y_i) \cdot P(x_1|y_i) \cdot P(x_2|x_1 y_i) \cdot P(x_3|x_1 y_i) \cdot P(x_4|x_3 y_i) \end{aligned} \quad (2)$$

Besides the low-order dependencies among variables, some higher order dependency models, *e.g.* frequent pattern based Bayesian

* Corresponding author.
E-mail addresses: yuanjd@bjtu.edu.cn (J. Yuan), zhhwang@bjtu.edu.cn (Z. Wang).

[1] $P(y_i|\mathbf{x}) = \frac{P(\mathbf{x}, y_i)}{P(\mathbf{x})} = \frac{P(y_i) \cdot P(\mathbf{x}|y_i)}{P(\mathbf{x})}$

https://doi.org/10.1016/j.neucom.2018.01.016
0925-2312/

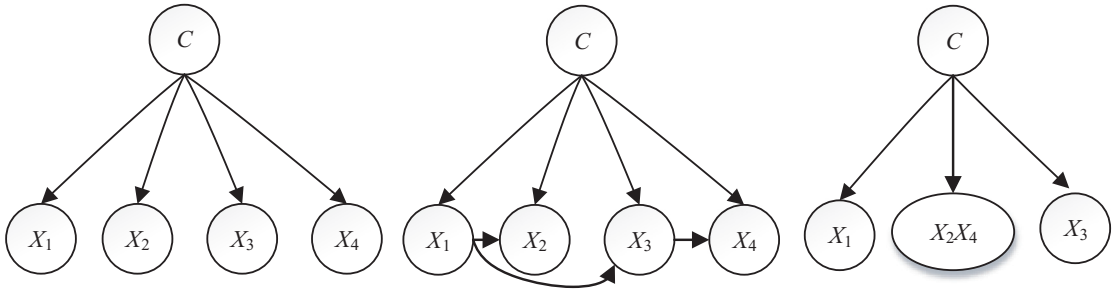


Fig. 1. (*Left*) The structure of Naive Bayes classifier. (*Middle*) The structure of Bayesian Network classifier. (*Right*) The structure of pattern-based Bayesian classifier.

[2,5,8], have also been proposed. Long and overlapped patterns are used to build Bayesian model previously [5,8], while independent pattern based Bayesian classifier presents its effectiveness and flexibility recently [2]. For pattern-based model, the joint probability is (as shown in Fig. 1 (*right*)):

$$\begin{aligned} P(\mathbf{x}, y_i) &= P(x_1,x_2,x_3,x_4,y_i) \\ &= P(y_i) \cdot P(x_1|y_i) \cdot P(x_2x_4|y_i) \cdot P(x_3|y_i) \end{aligned} \tag{3}$$

Previous works on streaming data mining either focus on finding interesting patterns efficiently [10–13,25] or creating classifiers effectively [15–17,19,20,22]. However, none of them combines pattern with classifier practically. Unlike these listed methods, our objective is to learn an efficient and effective **P**attern-based **B**ayesian classifier for **D**ata **S**tream (PBDS) that adapts to concept drift. Meanwhile, several challenges remain.

First, due to the space limitation, a window-based model for frequent pattern mining should be selected. To our knowledge, there are three basic stream processing models: landmark window model [9], sliding window model [11,12,25] and damped window model [26]. The sliding window considers a fixed number of stream records, compared with landmark window model, it detects changes in the properties of stream records faster (*e.g.* concept drift), and does not have to assign different weights to stream records (as damped window model), so the sliding window model is employed for efficient pattern mining in this paper.

Second, traditional pattern mining methods may generate excessive number of patterns that are useless in the classification, lacking of expressive power. In contrast, a lazy classifier can provide a more complete description for each record and avoid the extra computation adopted by eager algorithms. Since the computing of lazy classifier is performed on a demand driven basis, only the "useful" portion of the training data (in a sliding window) is mined for generating patterns applicable to the test instance, which increases the chance of achieving the most significant patterns that are useful for classifying the test case.

Third, unlike traditional methods that scan the whole database multiple times to get patterns, frequent pattern mining in a data stream can just check each instance for a single time. In order to store records adequately and mine frequent patterns rapidly, a compact representation of samples is crucial and helpful. Hence a simple but effective summary data structure for each class based on the sliding window model is proposed, which means the probability approximation for PBDS is separately tailored to each class.

Fourth, for a pattern-based Bayesian classifier, a set of long and not overlapped patterns that fully covers the given test case should be found, so it could be considered as a set covering problem essentially. As set covering problems are NP-hard in general, a heuristic pattern extraction mechanism is adopted, which is based on greedy search and the minimum description length (MDL) for the Bayesian classifier to reduce the generation of candidate itemsets, and to ensure the fitness between extracted patterns and original data records. It is also obvious that Naive Bayes is a special case of PBDS.

Table 1
Symbol table.

Symbol	Explanation
U	Data stream
x, T, T_i	A record
y, y_i	Class label of a record
C	Class attribute
X, X_i	Attribute of a record
Ω	Discrete domain of attribute X
$pa(X)$, $pa(X_i)$	parent node of attribute X or X_i
(X_i, x_i), x_i	Item
z	Itemset or pattern
B	Bayesian network
D, D_1, D_2, D_1'	Sliding window
N	Number of records in a sliding window
min_sup	Minimum (class) support
t	Time stamp
m	Length of a record
k	Number of parent nodes
r	Cardinality of attribute X
H, $H(\cdot\|\cdot)$	Entropy, conditional entropy
I	Mutual information
P, P_B, P_D	Probability distribution

Our contributions can be summarized as follows:

- A novel pattern-based Bayesian classifier for data stream is proposed;
- We propose an effective summary data structure for processing records over sliding windows;
- A lazy learning strategy is proposed to find local frequent patterns for each test record;
- A variant of MDL is formulated for pattern-based Bayesian classifier;
- Through an experimental study on real-world and synthetic datasets, the potential of proposed pattern-based Bayesian classifier is shown.

The remaining of this paper is organized as follows. Section 2 introduces preliminary concepts and notations. MDL for pattern-based Bayesian network is illustrated in Section 3. Section 4 describes our summary data structure, establishing the principle of extracting and updating local frequent patterns. Section 5 discusses the approach to estimate the Bayesian joint probabilities. In Section 6, an extensive experimental evaluation of the proposed approach is studied. Threat to works is discussed in Section 7. Finally, Section 8 concludes our work.

2. Definition and notation

In this section, some relevant definitions of this study will be introduced. Table 1 summarizes the notation of this paper, we expand on the definitions below.

Definition 1. Item and itemset

Let X_i be an attribute that describes a data feature. The discrete domain of X_i is Ω. An item (X_i, x_i) assigns value $x_i \in \Omega$ to attribute X_i. Usually an item (X_i, x_i) is represented by a single character x_i. An itemset (or a pattern) $\mathbf{z} = \{x_1, x_2, ..., x_k\}$ is a set of k items, which is also called k-itemset.

Definition 2. Data stream

A data stream is an infinite sequence of training records: $\mathbf{U} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, ...\}$, where $\mathbf{x}_t$ is the most recent record arriving at time stamp t. A record $\mathbf{x} = \{x_1, x_2, ..., x_m, y\}$ is represented by a set of m items, where y is the class label of $\mathbf{x}$.

Definition 3. Sliding window

Sliding window $D = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ contains N recent records of data stream $\mathbf{U}$, where N is the size of sliding window. The sliding window model is one of the most popular stream processing models for efficient mining of frequent patterns, whose target is the fixed number of recent records.

Definition 4. Support and frequent pattern

The support of itemset $\mathbf{z}$, $sup(\mathbf{z})$, is the proportion of records containing itemset $\mathbf{z}$ to the total records. If $sup(\mathbf{z}) \geq \min\_sup$, $\mathbf{z}$ is called as frequent itemset or frequent pattern, min_sup represents the minimum threshold of support value. The support of itemset in a sliding window is defined as:

$$sup(\mathbf{z}) = \frac{|\mathbf{z}|}{N} \tag{4}$$

where $|\mathbf{z}|$ is the number of records matching itemset $\mathbf{z}$ in current window.

Definition 5. Class support

In order to mine frequent patterns for building lazy Bayesian classifier, we should find frequent patterns for each possible class ($sup_{y_i}(\mathbf{z}) \geq \min\_sup$). The class support of itemset $\mathbf{z}$ for each class in sliding window could be represented as:

$$sup(\mathbf{z}) = \sum\nolimits_{\forall y_i \in C} sup_{y_i}(\mathbf{z}) = \sum\nolimits_{\forall y_i \in C} \frac{|\mathbf{z}|_{y_i}}{N} \tag{5}$$

Definition 6. Minimum description length

The MDL is closely related to Bayesian classifiers [23,27]. It is used to select optimal model and to avoid over-fitting. Let B be a Bayesian network built on sliding window D, The MDL scoring function of B given a training dataset (or a sliding window) D, is written as:

$$MDL(B|D) = DL(B) + DL(D|B) \tag{6}$$

where $DL(B)$ represents the length of the description of the model B, and $DL(D|B)$ the length of the description of the data D when encoded with the help of the model B. According to MDL, the best model of a given dataset is the one that minimizes the sum of the encoding lengths of the data $DL(D|B)$ and $DL(B)$ the model itself.

3. MDL for pattern-based Bayesian classifier

This section will discuss in greater details about MDL for pattern-based Bayesian classifier. MDL offers us a reasonable method to trade-off between fitness and complexity of the learned model. In order to learn effective patterns for Bayesian classifier, not only the length and support information of patterns are adopted, MDL is also employed as a selection criteria for pattern discovery (details in Section 4).

For Bayesian classifier, it minimizes the sum of the encoding model $DL(B)$ and the encoding data given model $DL(D|B)$ (as shown in Eq. (6)). For representing the Bayesian classifier B, we should describe the parents of each node and the conditional probability table combined with each node. Suppose for each node X_i, there are k_i parents for it, then we need $k_i log(m)$ bits to list its parents. Let r_i $(1 \leq i \leq m)$ be the cardinality of X_i, $pa(X_i)$ the parents of X_i, the number of conditional probabilities for node X_i is $r_i \times \prod_{j \in pa(X_i)} r_j$. Since all conditional probabilities for one node sum to 1, and the encoding length for each conditional probability is $logN/2$, the total description length for model B will be:

$$DL(B) = \sum_{i=1}^{m} \left(k_i \log m + \frac{\log N}{2} \times (r_i - 1) \times \prod_{j \in pa(X_i)} r_j \right) \tag{7}$$

For the description length of data D given model B, it is encoded by log likelihood method as follow:

$$\begin{aligned} DL(D|B) &= -\sum_{i=1}^{N} \log P_B(\mathbf{x}_i) \\ &= -N \sum_{j=1}^{m} \sum_{X_j, pa(X_j)} P_D(X_j, pa(X_j)) \log P_B(X_j|pa(X_j)) \end{aligned} \tag{8}$$

where P_B is the joint probability distribution over model B (as shown in Eq. (9)) and P_D the true probability distribution.

$$P_B(X_1, ..., X_m) = \prod_{i=1}^{m} P_B(X_i|pa(X_i)) \tag{9}$$

It is obvious that $DL(D|B)$ is minimized when

$$P_D(X_j|pa(X_j)) = P_B(X_j|pa(X_j)) \tag{10}$$

Applying Eq. (10) to Eq. (8), $DL(D|B)$ is written as

$$DL(D|B) = N \sum_{i=1}^{m} H(X_i|pa(X_i)) \tag{11}$$

According to the information theory, the conditional entropy is

$$H(X|pa(X)) = -\sum_{X, pa(X)} P_D(x, pa(x)) \log P_D(x|pa(x)) \tag{12}$$

It is well known that the mutual information of X, $pa(X)$ is

$$I(X; pa(X)) = \sum_{X, pa(X)} P_D(x, pa(x)) \log \frac{P_D(x, pa(x))}{P_D(x) P_D(pa(x))} \tag{13}$$

Therefore it is easy to proof that

$$H(X|pa(X)) = H(X) - I(X; pa(X)) \tag{14}$$

After applying Eq. (14) to Eq. (11), we get

$$DL(D|B) = N \sum_{i=1}^{m} H(X_i) - N \sum_{i=1}^{m} I(X_i; pa(X_i)) \tag{15}$$

Finally the MDL for a Bayesian model is

$$\begin{aligned} MDL(B|D) &= DL(B) + DL(D|B) \\ &= \sum_{i=1}^{m} \left(k_i \log m + \frac{\log N}{2} \times (r_i - 1) \times \prod_{j \in pa(X_i)} r_j \right) \\ &\quad + N \sum_{i=1}^{m} H(X_i) - N \sum_{i=1}^{m} I(X_i; pa(X_i)) \end{aligned} \tag{16}$$

For our pattern-based Bayesian classifier, if the dependency among items of pattern are not considered, the parent node will be the only class attribute. In addition, a lazy method is used to find patterns for each possible class label ($r_j = 1$), so the mutual information $I(X_i; pa(X_i))$ will be 0 because the value of parent node is constant. The MDL for our pattern-based Bayesian model is

$$MDL_{PBDS}(B|D) = \sum_{i=1}^{m} \frac{\log N}{2} \times (r_i - 1) + N \sum_{i=1}^{m} H(X_i) \tag{17}$$

where m represents the number of patterns, r_i the cardinality of pattern X_i, N the number of records for one special class.

Table 2
Running example.

Record ID	Attribute X_1	Attribute X_2	Attribute X_3	Attribute X_4	Class C
T_1	a_1	b_1	c_1	d_2	y_2
T_2	a_1	b_1	c_1	d_1	y_2
T_3	a_2	b_1	c_1	d_2	y_1
T_4	a_3	b_2	c_1	d_2	y_1
T_5	a_3	b_1	c_2	d_2	y_1
T_6	a_3	b_1	c_2	d_1	y_2
T_7	a_2	b_1	c_2	d_1	y_1
T_8	a_1	b_2	c_1	d_2	y_2
T_{test}	a_1	b_1	c_2	d_2	?

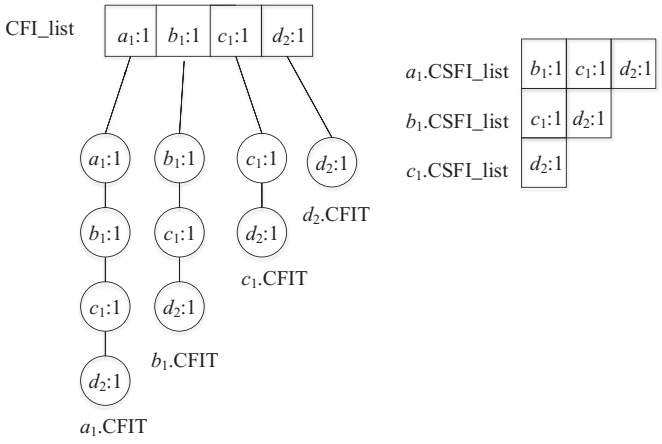


Fig. 2. CFI-forest construction after processing the first record T_1 of class y_2.

4. Frequent pattern mining for data stream

In this section, we describe the proposed lazy frequent pattern mining strategy by using a sliding window for Bayesian classifier. The construction of Candidate Frequent Itemsets forest (CFI-forest) that only scan the data stream once is introduced at first, then details about how to select patterns based on support and MDL_{PBDS} for characterizing each test record will be given.

4.1. Construction of CFI-forest

In order to construct CFI-forest, for a basic sliding window $D = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, all the records are read and split to $|C|$ folds according to their class label y_i, $|C|$ is the number of classes in D. At the same time, CFI-forest CFI_i for each class fold, $\mathbf{CFI} = \{CFI_1, CFI_2, ..., CFI_{|C|}\}$ are built and updated.

A running example better explaining our algorithm is shown in Table 2. Suppose the current sliding window D_1 contains four records, $D_1 = \{T_1, T_2, T_3, T_4\}$, where each $T = \{X_1, X_2, X_3, X_4, C\}$. The CFI-forest structure consists of three parts: a list of Candidate Frequent Items (CFI_list), a list of Candidate Frequent Item Trees (CFIT) and a list of Candidate Suffix Frequent Items (CSFI_list). Accordingly we split records into two folds, $\{T_3, T_4\} \in y_1$ and $\{T_1, T_2\} \in y_2$ at first step. Here we take the fold $\{T_1, T_2\} \in y_2$ as an example to illustrate the construction of CFI-forest structure CFI_2.

As shown in Fig. 2, for the first record $T_1 = \{a_1, b_1, c_1, d_2\}$, a set of sub-records $\{a_1, b_1, c_1, d_2\}$, $\{b_1, c_1, d_2\}$, $\{c_1, d_2\}$, $\{d_2\}$ are generated and inserted to x_i.CFIT separately. CFI_list includes all the items and their corresponding support number, while x_i.CSFI_list gives the suffix items and their support number of x_i (note that item d_2 has no suffix item). Similarly, T_2 is processed and added on the CFI_2 (as shown in Fig. 3). After sliding the window, the previous D_1 is updated to $D_1' = \{T_2, T_3, T_4, T_5\}$. Now the distribution of records is also changed to $\{T_3, T_4, T_5\} \in y_1$ and $\{T_2\} \in y_2$, so we need

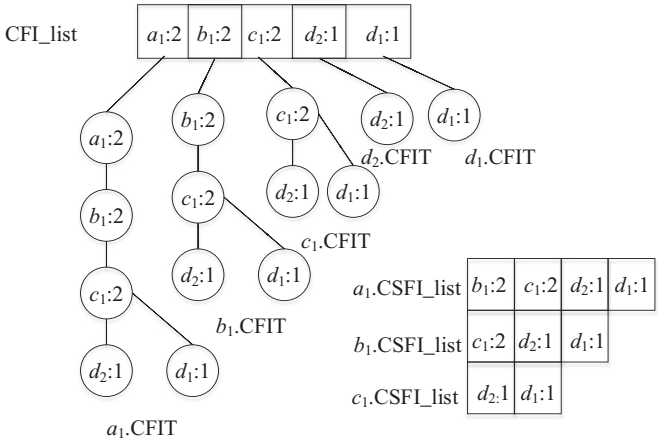


Fig. 3. CFI-forest construction after processing the second record T_2 of class y_2.

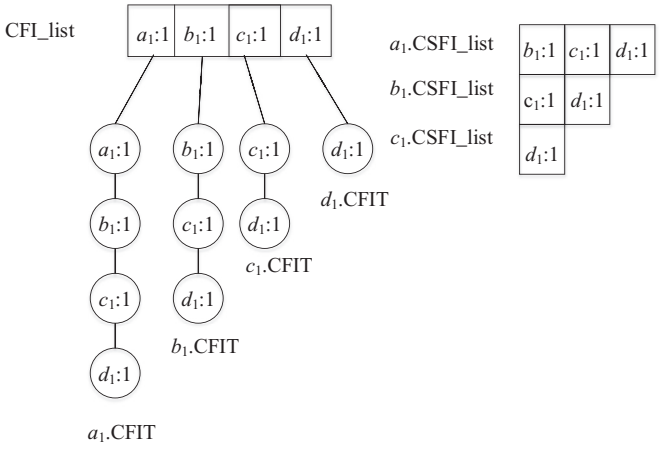


Fig. 4. CFI-forest construction of class y_2 after sliding the window to D'.

to remove $T_1 = \{a_1, b_1, c_1, d_2\}$ from CFI_2, the latest CFI_2 is shown in Fig. 4.

4.2. Lazy pattern mining

The previous section mainly illustrates the process of creating and updating CFI-forest for each possible class. Here we will describe how to find effective patterns for Bayesian classifier. This problem is not straightforward and some issues need to be addressed. For example, which pattern or set of patterns should be selected for classifying? Conversely, what is the criterion for selecting pattern or a set of patterns? If the length and support of the selected patterns are different from each other, which one(s) should be chosen?

Definition 7. Orders for patterns

Given two frequent patterns $\mathbf{z}_1$ and $\mathbf{z}_2$, $\mathbf{z}_1 \succ \mathbf{z}_2$ (also called $\mathbf{z}_1$ precedes $\mathbf{z}_2$ or $\mathbf{z}_1$ has a higher precedence than $\mathbf{z}_2$) if

1. The length of $\mathbf{z}_1$ is longer than that of $\mathbf{z}_2$, or
2. Their length are the same, but the support of $\mathbf{z}_1$ is greater than that of $\mathbf{z}_2$, or
3. Both the length and supports of $\mathbf{z}_1$ and $\mathbf{z}_2$ are the same, but the MDL_{PBDS} of $\mathbf{z}_1$ based Bayesian is shorter than the $\mathbf{z}_2$ based one.

Our aim is to find patterns that cover the test record lazily, since set covering is a NP-hard problem, a heuristic pattern selection method is adopted to find patterns greedily. The main strategy of pattern discovery corresponds to the following steps:

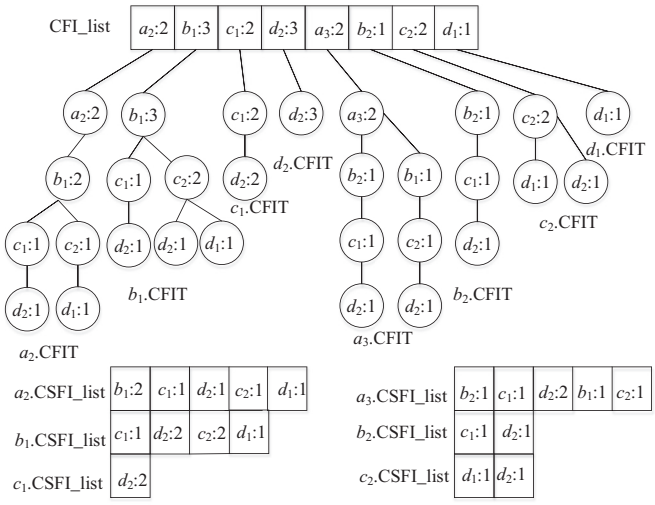


Fig. 5. CFI-forest of class y_1 under sliding window D_2.

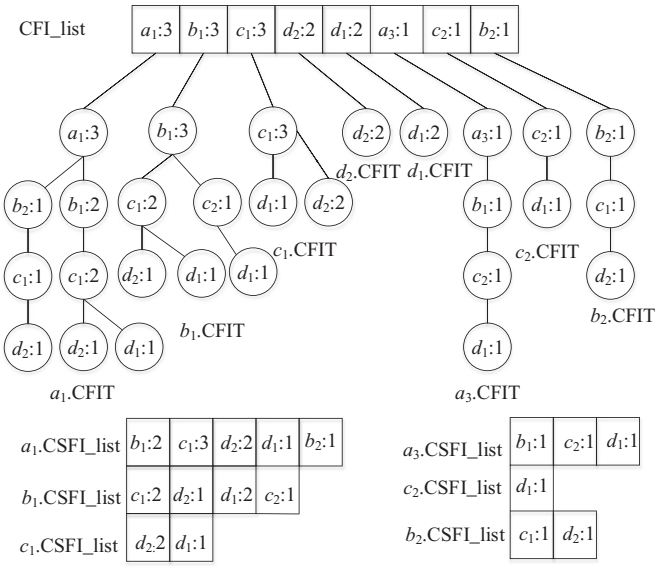


Fig. 6. CFI-forest of class y_2 under sliding window D_2.

- Step 1: Each CFI_i tries to find the first node x'_1 of $T_{test} = \{x'_1, x'_2, ..., x'_m\}$ in CFI_list. If found and its support larger than *min_sup*, go to Step 2. Or it means x'_1 is not frequent given class y_i. Then, it moves to the next item x'_2 and carries out Step 1 until one item x'_i of T_{test} have be founded. If there is no entry with the same item of T_{test} in the current CFI_list, the probability that T_{test} belongs to y_i is set to 0;
- Step 2: Find the x'_i.CSFI_list of frequent item x'_i. For suffix items of x'_i, looking for the longest candidate frequent itemset $\mathbf{z}' = \{x_i, x_{i+1}, x_{i+2}, ...\}$ according to x'_i.CSFI_list;
- Step 3: Check whether the discovered longest pattern is really frequent or not by searching x'_i.CFIT. If true, $\mathbf{z}'$ is saved. Or x'_i.CFIT is traversed to check sub-patterns of $\mathbf{z}'$, selecting best pattern under rules of Def. 7;
- Step 4: Best pattern and infrequent items are deleted from T_{test}, repeating steps 1–4 iteratively until all items in record T_{test} have been tested.

In summary, our goal is to find the longest disjointed patterns, by using *min_sup* and MDL_{PBDS} to break ties. When there is no frequent pattern discovered by our method, PBDS declines to the strong conditional independence assumption model NB. Note, we cannot ensure that each item or pattern occurs together with a given class. In other words, sometimes the discovered patterns of CFI_i are not enough to cover test record since some items are infrequent. Hence it arise the zero frequency count or non-frequency problem. For Bayesian classifier, multiplied by zero probability will ignore the contributions of other patterns, so Laplacian smoothing is used to address this issue. An example will be given to elaborate the selection strategy.

Suppose the current sliding window $D_2 = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$, the CFI-forest for each class has been constructed and been shown in Figs. 5 and 6 separately. For a test record $T_{test} = \{a_1, b_1, c_2, d_2\}$, the minimum class support *min_sup* is set to 0.5 (so the minimum support number for each class is 4 * 0.5 = 2). Considering the CFI_1 of y_1, the first item a_1 is not exist in the CFI_list, so a_1 is infrequent given class y_1. Then we move to the next item b_1, it is clear that b_1 is frequent since the support number 3 is larger than 2 (Step 1). After we search b_1.CSFI_list, the longest candidate frequent pattern is $\{b_1: 2, c_2: 2, d_2: 2\}$ (Step 2). When traversing b_1.CFIT, it is easy to find that the support number of $\{b_1, c_2, d_2\}$ is 1, so we continue to check the subsets of $\{b_1, c_2, d_2\}$. $\{b_1: 2, d_2: 2\}$ and $\{b_1: 2, c_2: 2\}$ are the current longest pattern with same length and support, so our MDL_{PBDS} strategy is used to select the better one (Step 3). Assume $\{b_1: 2, c_2: 2\}$ is selected, we remove it and the infrequent item a_1 from T_{test}, the remaining item d_2 is also checked accordingly. In this example, the final patterns for y_1 are $\{a_1, b_1c_2, d_2\}$, and $\{a_1d_2, b_1, c_2\}$ for y_2.

5. Pattern-based Bayesian classification

In this section, we present the learning algorithm for PBDS, illustrating how patterns are deployed for Bayesian classifier.

As shown in Algorithms 1 and 2, before a classification request occurs, if the number of records contained in the sliding window D is smaller than N, record $\mathbf{x}$ is add to the window D, and the corresponding CFI_i is updated (Algorithm 2 Lines 2–5). Or the sliding window D moves forward, CFI_i deletes the oldest $\mathbf{x}_{old}$ at first, then the new record is added to CFI_i (Algorithm 2 Lines 7–12); When a test record comes, for each possible class y_i, it finds patterns on CFI_i to cover the test record first (Algorithm 1 Lines 1–7). Then the joint probability $P(\mathbf{x}, y_i)$ is calculated (Algorithm 1 Line 8); The predicted class label is returned as the maximum one among all the joint probability $P(\mathbf{x}, y_i)$ (Algorithm 1 Line 10). The classification accuracy is reported when all the records are processed.

For simplicity, we continue to elaborate our example talked above in Section 4.2. After we found the patterns for y_1 and y_2 under window D_2, the joint probabilities are

Algorithm 1 PBDS($\mathbf{x}_{test}$, **CFI**, *min_sup*).

Input: $\mathbf{x}_{test}$, test record to be classified;**CFI**, a list of CFI-forests;*min_sup*, minimum class support
Output: predicted class label y_{test}

```
1: for each CFI_i of CFI do
2:     finalPattern = φ
3:     repeat
4:         bestPattern = selectPattern(x_test, CFI_i, min_sup)
5:         x_test = x_test − bestPattern
6:         finalPattern = finalPattern ∪ bestPattern;
7:     until x_test = φ
8:     P(x, y_i) = P(y_i) ∏_{z∈finalPattern} P(z|y_i) {z is the selected pattern}
9: end for
10: return y_test = max(P(x, y_i))
```

Algorithm 2 CFI-forest(**U**, *N*).

Input: **U**, data stream;*N*, length of a sliding window *D*;
Output: a list of CFI_i for each class y_i

```
for each x of U do
    if |D| < N then
        |D| = |D| + 1
        D = D ∪ x {add x to sliding window D}
        CFIBuilding(x, CFI_i)
    else
        |D| = |D| − 1
        D = D − x_old {delete the oldest record x_old from sliding window D}
        deleteFromCFI(x_old, CFI_i);
        |D| = |D| + 1
        D = D ∪ x
        CFIBuilding(x, CFI_i)
    end if
end for
```

$P(T_{test}, y_1) = P(a_1,b_1,c_2,d_2,y_1) = P(y_1) \cdot P(a_1|y_1) \cdot P(b_1c_2|y_1) \cdot P(d_2|y_1)$

$P(T_{test}, y_2) = P(a_1,b_1,c_2,d_2,y_2) = P(y_2) \cdot P(a_1b_1|y_2) \cdot P(c_2|y_2) \cdot P(d_2|y_2)$

While the item a_1 does not exist in the records of y_1, we use Laplacian smoothing to estimate the probability of $P(a_1|y_1)$: $P(a_1|y_1) = \frac{1}{|C|+|X_1|}$. For fairness, the conditional probabilities of selected frequent patterns are $P(b_1c_2|y_1) = \frac{|b_1c_2y_1|+1}{|C|+|X_2X_3|}$ and $P(d_2|y_1) = \frac{|d_2y_1|+1}{|C|+|X_4|}$, where $|b_1c_2y_1|$ represents the number of records that match pattern $b_1c_2y_1$. A similar technique is applied on $P(T_{test}, y_2)$. Finally, we pick the larger approximation and the class label of T_{test} is predicted to y_2.

6. Experiment and evaluation

This section will describe first the datasets retained to conduct our experiments, prior to introduce scalability analysis, different data stream classification methods compared, and the case studies.

6.1. Datasets

In order to evaluate the performance of our PBDS approach on Massive Online Analysis (MOA) platform [28], synthetic and real datasets that have relatively large records (≥ 10000) are selected.[2] Their characteristics are shown in Table 3. Since continuous attribute values cannot be directly employed in classification by means of patterns, here we are only interested in the case where all the variables are discrete. The entropy-based discretization approach is performed as a preprocessing step for continuous variables [29].

6.1.1. Synthetic datasets

Compared with real datasets, synthetic datasets are easier to reproduce, and more importantly, it is straightforward to learn the ground of their truth. *e.g.*, we can notice where exactly concept drift happens, what the type of drift is, and the best classification result achievable on each concept. The synthetic datasets shown in Table 3 contain three types of concept drift: sudden, gradual, and mixed.

[2] Datasets Chess, Connect-4, EEG, MAGIC, PokerHand and CoverType are downloaded from UCI Machine Learning Repository http://archive.ics.uci.edu/ml/; Others are generated by the classical data generators separately via MOA.

HyperPlane is a two-class dataset that models a rotating hyper plane in a *d*-dimensional space. It is represented by a set of points *x* that satisfy $\sum_{i=1}^{d} w_i x_i = w_0$, where x_i is the *i*th coordinate of *x*. Instances for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled positive, while instances that $\sum_{i=1}^{d} w_i x_i < w_0$ are negative. A stream containing 1,000,000 records with gradual drifts by the modification weight w_i changing by 0.001 with each instance, adding 5% of noise, is generated.

The goal of LED or LED_{drift} dataset is to predict the digit displayed on a seven-segment LED display. We set the LED generator to create two datasets, each containing 1,000,000 records and 10% of noise. The first dataset (LED) contains no drift, described by 7 relevant features, while the second dataset (LED_{drift}) is produced by 24 binary attributes, 17 of which are irrelevant. In addition, sudden and gradual concept drifts are simulated by interchanging relevant attributes of LED_{drift}.

The SEA dataset consists of three attributes, where only the first two are related to its class label. This dataset is divided into four blocks with different concepts. In each block, the class attribute is learned according to a formula $f_1 + f_2 \leq \theta$, where f_1 and f_2 represent the first two attributes and θ is a threshold value. The thresholds are 9, 8, 7, and 9.5 separately. We generated a dataset containing 1,000,000 records with sudden drifts occurring every 250,000 instances and involving 10% of class noise.

In order to evaluate our model from different aspects, datasets RBF and STAGGER that contain neither concept drift nor noise are generated independently. Both consist of 1,000,000 records. The RBF generator reproduces a random radial basis function stream, while the STAGGER concepts are Boolean functions of three attributes encoding objects: size, shape, and color.

6.1.2. Real datasets

For real-world datasets, it is not easy to notice exactly when a drift starts to occur, which type of drift is present, or even if there really is a drift. Therefore, the selected real datasets serve to compare the algorithms in a real-life situation rather than a concrete concept drift scenario.

Datasets Chess, Connect-4 and Poker Hand are all donated from the area of games. The Chess dataset is an end-game database for White King and Rook against Black King. It contains six attributes and one class variable, the goal is to find the optimal depth-of-win for White in 0–16 moves, otherwise drawn. Connect-4 contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced. The outcome class is the game theoretical value for the first player. The purpose of Poker Hand dataset is to predict poker hands. It consists of 1,000,000 instances, each record represents a hand consisting of five cards drawn from a standard deck of 52. Each card is described by two attributes (suit and rank), for a total of 10 predictive attributes.

EEG Eye State dataset consists of 14 EEG values and a value indicating the eye state. MAGIC Gamma Telescope dataset is generated to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. The Cover Type dataset includes the forest cover type for cells of 30×30 meters produced from US Forest Service Region 2 Resource Information System. It contains 581,012 instances, which are defined by 53 cartographic variables that depict one of seven possible forest cover types. The aim is to predict the forest cover type based on cartographic variables.

6.2. Scalability

In this experiment, we initially studied the execution time and memory usage of PBDS when data volume increases. It is straight-

Table 3
Dataset characteristics.

Data	Records	Attribute			Items	Classes	Noise (%)	Drift type
		Continuous	Nominal	Discretized				
HyperPlane	1000000	10	0	10	120	2	5	Gradual
LED	1000000	0	7	7	14	10	10	None
LED_{drift}	1000000	0	24	24	48	10	10	Mixed
SEA	1000000	3	0	3	71	2	10	Sudden
RBF	1000000	10	0	10	682	2	0	None
STAGGER	1000000	0	3	3	9	2	0	None
Chess	28056	0	6	6	41	18	–	Unknown
Connect-4	67557	0	42	42	126	3	–	Unknown
PokerHand	1000000	0	10	10	85	10	–	Unknown
EEG	14980	14	0	14	82	2	–	Unknown
MAGIC	19020	10	0	10	79	2	–	Unknown
CoverType	581012	10	44	54	377	7	–	Unknown

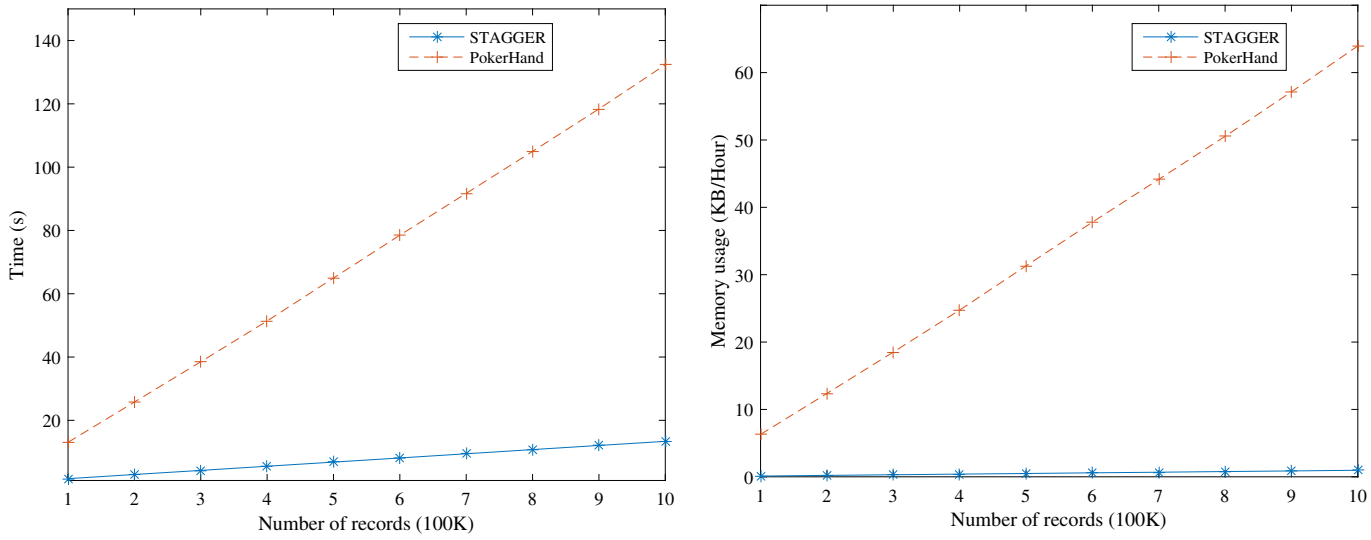


Fig. 7. Linear scalability of PBDS algorithm ($min\_sup = 50\%$, $N = 100$).

forward to find that in Fig. 7, the execution time and memory usage grows linearly on both relatively sparse data (STAGGER with 2 classes and 9 items) and comparatively dense data (PokerHand with 10 classes and 85 items). The default value of *min_sup* is set to 50%, *N* is 100. Compared with STAGGER, PokerHand consumes more resources since it contains more items and class values, and our lazy model needs to build compressed tree for each possible class.

In addition, for the proposed PBDS, there are two factors that strongly determine its efficiency and effectiveness, the value of minimum support *min_sup* for each class and the size of sliding window *N*, we vary each one independently below.

As shown in Fig. 8 (*left*), with the increase of support value, the accuracies of STAGGER and PokerHand vary in the beginning, then they remain stable as lacking of frequent patterns that satisfy *min_sup*. Similar situation appears on the time consumptions shown in Fig. 8 (*right*), when support value is relatively small, there are more items that satisfy the threshold, so it needs more time to select the best pattern for each record.

Fig. 9 illustrates the accuracy and execution time of PBDS when *N* varies from 1 to 1000. Although the classification accuracy of PBDS increases with some fluctuation, it remains constant gradually as window size *N* growing (as shown in Fig. 9 (*left*)). Fig. 9 (*right*) also exhibits that the execution time of PBDS climbs steadily as more instances need to be scanned when *N* rises. In summary, there is a trade-off between classification accuracy and execution time for our PBDS when *N* or *min_sup* shift.

6.3. Comparative study of classifiers

The performance of proposed method was evaluated in terms of accuracy, time consumption, and memory usage, as shown in Tables 4–6 separately (the best results for each dataset are indicated in bold). Experiments designed to check its potentiality of handling different types of concept drifts are also discussed in this section.

6.3.1. Classification accuracy

Classification accuracy measures the ability of a classifier to correctly predict the class label of unknown record. Here we compare PBDS with a list of classifiers that belong to specific categories, such as Bayesian classifier, rule-based classifier, instance based classifier, tree model and ensemble method. In particular, we consider the MOA default settings of Nave Bayes (NB), Naive Bayes Multinomial (NBM) [30], *k*NN, *k*NNwithPAW (PAW) [22], RuleClassifierNBayes (RCNB) [17], HoeffdingTree (HT) [15], and Accuracy Weighted Ensemble classifier [21](AWE). For the proposed PBDS, the *min_sup* that decides the quantity of patterns, and the size of sliding window *N* are learned by a standard 5-fold cross validation on a 10% sampling of the original dataset, where $min\_sup \in \{0, 0.05, 0.1 : 0.1 : 1.0\}$, and $N \in \{1, 10, 100 : 100 : 1000\}$, the learned parameters are also presented in the last column of Table 4. The presequential scheme that interleaves training and test records is used for evaluating above selected classification algorithms.

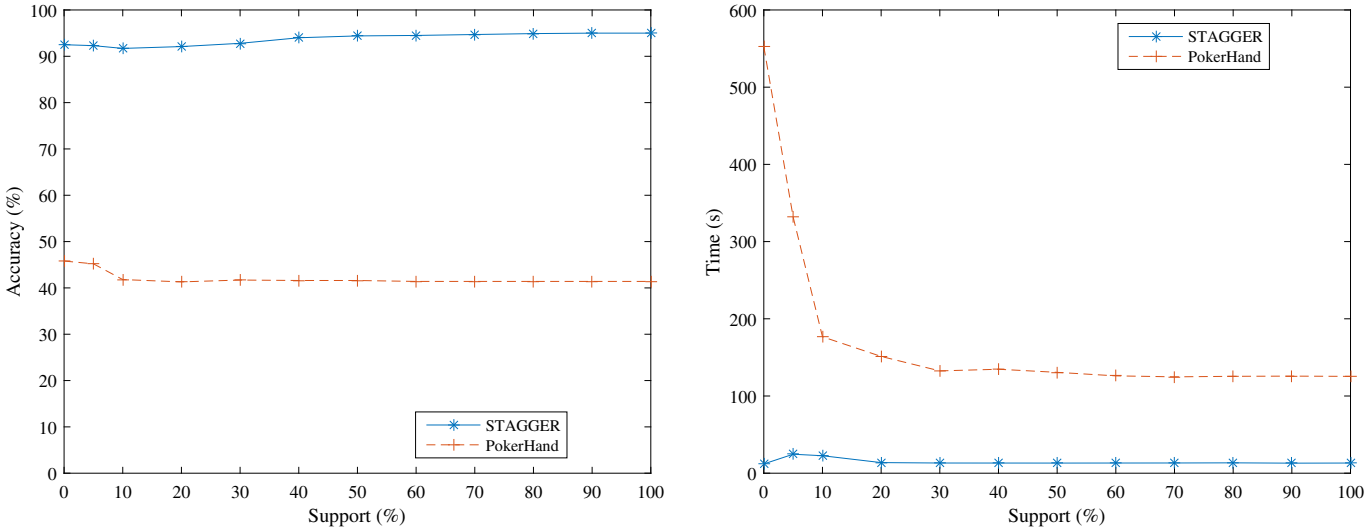


Fig. 8. Accuracy and execution time of PBDS when *min_sup* varies ($N = 100$).

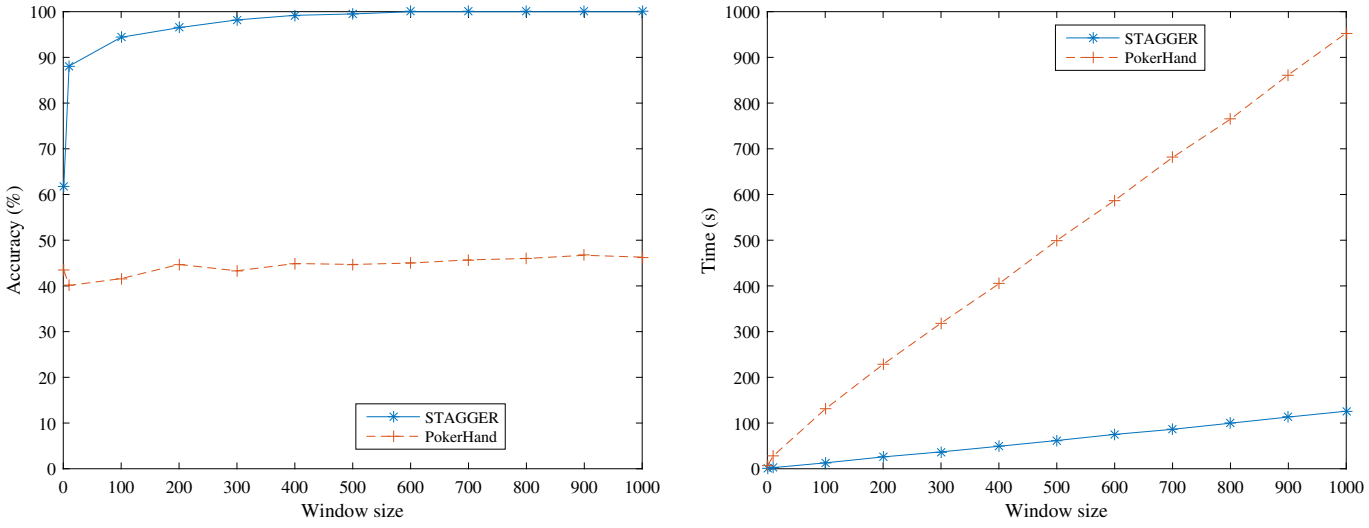


Fig. 9. Accuracy and execution time of PBDS when N varies ($min\_sup = 50\%$).

Table 4
Accuracy comparison with state-of-the-art classifiers (%).

Data	NB	NBM	RCNB	*k*NN	PAW	HT	AWE	PBDS [*N*, *min_sup*]
HyperPlane	**88.50 (1)**	79.70 (6)	85.50 (4)	77.50 (7)	75.80 (8)	86.80 (3)	80.50 (5)	87.90 [100, 0.05] (2)
LED	71.90 (7)	70.50 (8)	73.10 (2)	72.60 (3)	72.20 (4)	72.00 (5.5)	72.00 (5.5)	**73.40 [1000, 0.05] (1)**
LED_{drift}	28.80 (5)	27.80 (6)	33.40 (2)	23.10 (8)	24.80 (7)	**33.60 (1)**	29.70 (4)	30.80 [1000, 0.90] (3)
SEA	81.00 (5)	65.10 (8)	84.90 (2)	80.70 (6.5)	80.70 (6.5)	84.60 (3)	84.50 (4)	**85.70 [200, 0.10] (1)**
RBF	73.90 (5.5)	68.70 (8)	75.10 (3)	74.70 (4)	76.50 (2)	**85.40 (1)**	73.30 (7)	73.90 [1000, 0.05] (5.5)
STAGGER	**100.00 (4)**	93.33 (8)	**100.00 (4)**	**100.00 (4)**	**100.00 (4)**	**100.00 (4)**	**100.00 (4)**	**100.00 [600, 0.50] (4)**
Chess	5.00 (7)	2.10 (8)	40.60 (4)	87.70 (2)	73.10 (3)	22.30 (5)	5.60 (6)	**99.80 [100, 0.05] (1)**
Connect-4	48.10 (8)	53.40 (7)	64.50 (3)	**66.50 (1)**	65.40 (2)	61.50 (4)	59.50 (6)	60.20 [100, 1.00] (5)
PokerHand	46.20 (6.5)	46.20 (6.5)	62.50 (2)	49.40 (5)	49.60 (4)	**79.70 (1)**	44.60 (8)	58.30 [200, 0.30] (3)
EEG	83.60 (6)	82.30 (7)	90.40 (4)	91.20 (2)	91.10 (3)	79.60 (8)	87.00 (5)	**92.90 [10, 0.05] (1)**
MAGIC	47.50 (8)	68.50 (7)	**100.00 (3.5)**	**100.00 (3.5)**	**100.00 (3.5)**	**100.00 (3.5)**	**100.00 (3.5)**	**100.00 [10, 0.05] (3.5)**
CoverType	80.10 (6)	64.80 (8)	96.90 (4)	**98.00 (1)**	97.60 (2)	71.80 (7)	88.10 (5)	97.50 [1000, 0.05] (3)
Average	5.75	7.29	3.13	3.92	4.08	3.83	5.25	**2.75**

As shown in Table 4, PBDS outperforms other classical methods, especially on datasets LED, SEA, Chess and EEG. The average rank of PBDS on 12 synthetic and real data streams is 2.75, it wins all other classifiers. The critical difference diagram shown in Fig. 10 tells that our pattern-based Bayesian classifier is significantly better than the traditional Naive Bayes based model NBM. In addition, it is notable that the accuracy of PBDS on dataset Chess is 99.80%, while NB (or NBM) just classify 5.00% (or 2.10%) of instances accurately.

6.3.2. Time and memory usage

In terms of classification time, as shown in Table 5, it is straightforward that NB or NBM is one of the most efficient classifiers, ranking 1.25 and 2.00 separately. Compared with NB or NBM,

Table 5
Time comparison with state-of-the-art classifiers (s).

Data	NB	NBM	RCNB	*k*NN	PAW	HT	AWE	PBDS
HyperPlane	4.69 (2)	**4.63 (1)**	502.33 (7)	345.53 (6)	547.09 (8)	14.67 (3)	250.27 (5)	122.07 (4)
LED	**3.73 (1)**	6.38 (3)	864.58 (8)	270.03 (6)	420.08 (7)	6.33 (2)	120.41 (4)	218.69 (5)
LED_{drift}	**9.41 (1)**	15.23 (2)	620.97 (5)	1198.89 (7)	1574.38 (8)	11.59 (3)	227.19 (4)	1058.93 (6)
SEA	**1.86 (1)**	4.80 (3)	579.13 (8)	105.98 (6)	191.84 (7)	3.91 (2)	56.91 (4)	71.43 (5)
RBF	5.59 (2)	**4.84 (1)**	177.63 (4)	435.05 (6)	674.53 (7)	16.39 (3)	209.03 (5)	1816.89 (8)
STAGGER	**1.61 (1)**	1.95 (3)	314.06 (8)	129.39(6)	217.53 (7)	1.83 (2)	23.02 (4)	65.43 (5)
Chess	**0.27 (1)**	0.39 (2)	3.33 (6)	6.69 (7)	10.53 (8)	0.50 (3)	2.61 (4)	3.06 (5)
Connect-4	**1.20 (1)**	1.38 (2)	4.41 (4)	130.02 (6)	176.67 (7)	1.94 (3)	19.70 (5)	289.34 (8)
PokerHand	**5.42 (1)**	7.77 (2)	233.94 (6)	534.72 (7)	757.53 (8)	10.14 (3)	147.77 (4)	186.39 (5)
EEG	0.23 (2)	**0.22 (1)**	0.45 (4)	6.73 (7)	9.78 (8)	0.44 (3)	1.92 (5)	6.71 (6)
MAGIC	**0.23 (1)**	0.25 (2)	0.36 (3)	6.97 (7)	9.89 (8)	0.41 (4)	1.78 (5)	6.19 (6)
CoverType	**13.89 (1)**	17.81 (2)	62.77 (4)	958.78 (6)	1226.58 (7)	25.64 (3)	943.64 (5)	2121.18 (8)
Average	**1.25**	2.00	5.58	6.42	7.50	2.83	4.50	5.92

Table 6
Memory usage comparison with state-of-the-art classifiers (KB/h).

Data	NB	NBM	RCNB	*k*NN	PAW	HT	AWE	PBDS
HyperPlane	0.019 (2)	**0.015 (1)**	260.583 (8)	66.008 (4)	149.858 (7)	67.012 (5)	82.706 (6)	3.316 (3)
LED	**0.009 (1)**	0.010 (2)	36.497 (5)	42.290 (7)	94.340 (8)	0.341 (3)	11.058 (4)	38.692 (6)
LED_{drift}	0.071 (2)	**0.063 (1)**	56.618 (5)	473.934 (7)	894.272 (8)	1.226 (3)	42.572 (4)	174.814 (6)
SEA	0.003 (2)	**0.002 (1)**	4947.741 (8)	12.011 (6)	31.215 (7)	1.051 (3)	4.080 (5)	1.721 (4)
RBF	0.089 (2)	**0.061 (1)**	99.730 (5)	87.134 (4)	190.549 (7)	163.168 (6)	35.668 (3)	439.353 (8)
STAGGER	**0.001 (1.5)**	**0.001 (1.5)**	0.880 (4)	14.556 (7)	35.127 (8)	0.002 (3)	1.391 (6)	0.947 (5)
Chess	**0.001 (1.5)**	**0.001 (1.5)**	0.029 (4)	0.977 (7)	2.191 (8)	0.009 (3)	0.207 (6)	0.033 (5)
Connect-4	0.011 (2)	**0.009 (1)**	0.209 (4)	77.171 (6)	148.864 (7)	0.161 (3)	4.577 (5)	415.236 (8)
PokerHand	0.034 (2)	**0.028 (1)**	12.630 (5)	102.161 (7)	207.466 (8)	2.295 (3)	17.065 (6)	3.077 (4)
EEG	**0.001 (1.5)**	**0.001 (1.5)**	0.005 (3)	1.689 (7)	3.477 (8)	0.008 (4)	0.410 (6)	0.075 (5)
MAGIC	**0.001 (1.5)**	**0.001 (1.5)**	0.002 (3)	1.321 (5)	2.627 (8)	0.004 (4)	0.138 (6.5)	0.138 (6.5)
CoverType	0.275 (2)	**0.195 (1)**	6.050 (3)	763.954 (5)	1410.557 (6)	81.086 (4)	1505.897 (7)	2123.484 (8)
Average	1.75	**1.25**	4.75	6.00	7.50	3.67	5.38	5.71

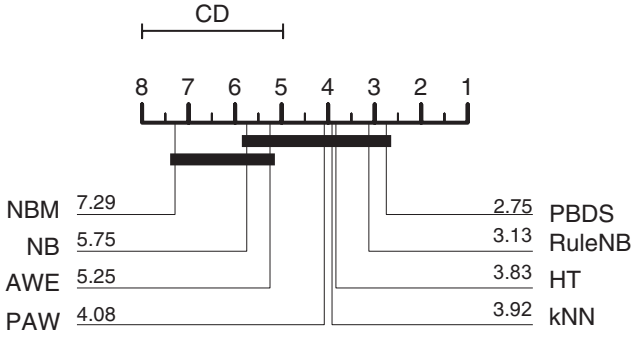


Fig. 10. Critical difference diagram for the algorithms.

PBDS needs to update CFI-forest, searching local frequent patterns for each record, so it is no doubt that PBDS is slower than NB or NBM. However, in comparison with instance based lazy classifiers (*e.g.* *k*NN and PAW), PBDS is faster when items of the datasets are relatively small (*e.g.* HyperPlane, LED, LED_{drift} and so on).

Similarly, simple but not so accurate classifiers (*e.g.* NB and NBM) achieved minimal memory consumption, as shown in Table 6. In most cases, especially when the number of items or class attributes are not large, the memory usage of PBDS is relatively less than instance based classifiers (*e.g.* *k*NN and PAW), ensemble method (*e.g.* AWE), and even RCNB (*e.g.* HyperPlane, SEA and PokerHand). This is partly because our method employs a compressed tree structure to store each record efficiently.

From Figs. 7–9 and Tables 3–6, it is clear that there is a trade-off between resources (including execution time and memory usage) and accuracy for PBDS, and the parameter selection partly depends on the number of classes or items of dataset. When class values or items are relatively dense, our experiment results prefer larger window size *N* for better accuracy, correspondingly, the time consumption and memory usage are relatively higher. In addition, our lazy PBDS outperforms other lazy classifiers (like *k*NN and PAW) on resource and accuracy comparisons. We will also show that PBDS is more robust to concept drifts compared with other classifiers in next section.

6.3.3. Case studies

This section will demonstrate that our pattern-based model is useful in dealing with several different types of concept drifts. For that, box plots that enable us to study the distributional characteristics of a group of accuracies, as well as the level of their accurateness, are employed to exhibit advantages of our algorithm.

As shown in Fig. 11 (*left*), the horizontal axis indicates the number of records processed in unit of 50*K*, and the vertical axis shows the classification accuracies of different algorithms on the HyperPlane dataset. It is well known that gradual concept drifts exist in HyperPlane, so there is no wonder that the accurate rate of each algorithm varies over the period. However, PBDS is apparently the most stable and accurate one, which could be observed more clearly from the box plot of accuracies shown in Fig. 11 (*right*). This can be explained by the fact that our pattern-based model is appropriate for gradual changes, and captures local patterns well.

Fig. 12 (*left*) shows the evolving of classification accuracy on the SEA dataset. When a sudden concept drift appears, the accurate rates of all tested algorithms show a dip downward, except for our method. In addition, PBDS maintains a higher, more stable accuracy, preserving the smallest descending. This point of view is also established by the box plot shown in Fig. 12 (*right*), it suggests that PBDS not only has the best accuracy, but also holds a lower variance as data evolves. This might be caused by the pattern-based model which could adapt to or even capture sudden concept drifts instantly, building classifiers lazily in a timely manner to cope with

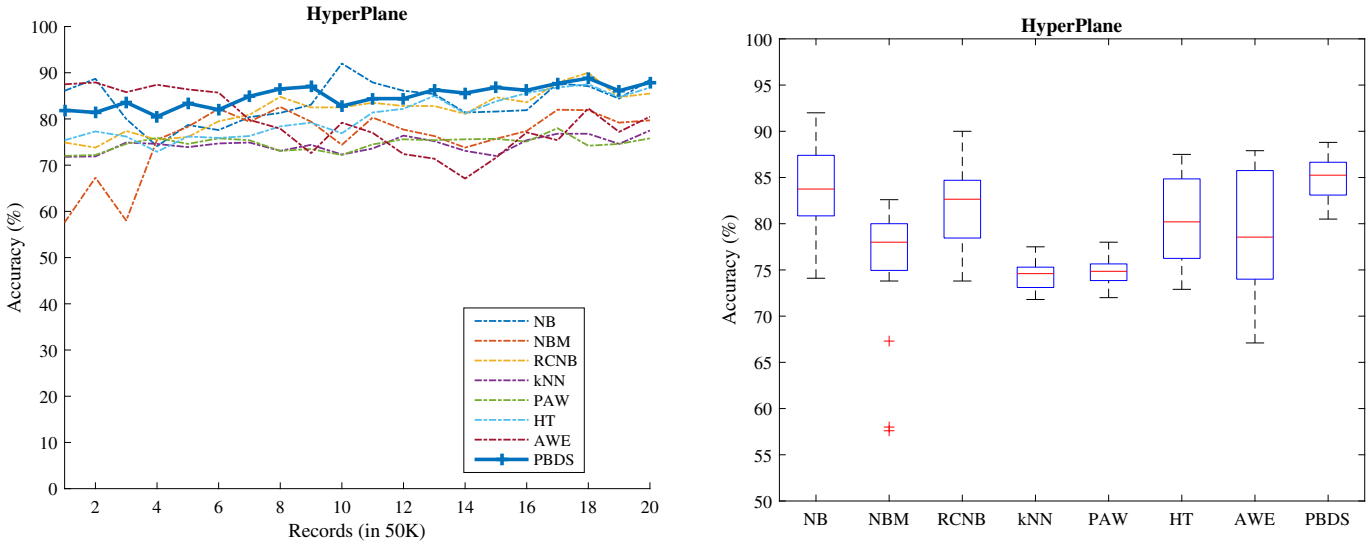


Fig. 11. Classification accuracy on the HyperPlane dataset.

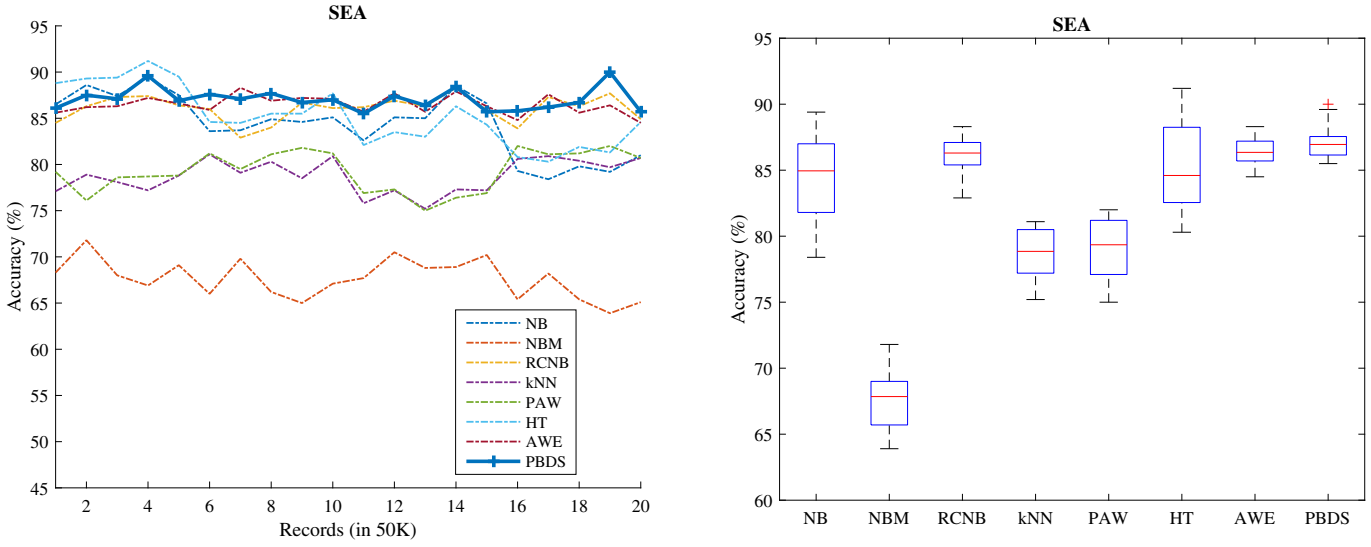


Fig. 12. Classification accuracy on the SEA dataset.

this type of drift. Since the dataset contains 10% of the noise, it also illustrates that the proposed method is more suitable for the noise environment.

Fig. 13 presents the classification accuracy on the LED_{drift} dataset, which is designed to verify the performance of different algorithms on mixed drifts. This dataset consists of a complex shift by combining two gradually drifting streams. After 500*K* records, the target distribution is instantly switched from a concept to another. It is obvious that all tested algorithms maintain a high and stable accuracy when the data was relatively steady. When a sudden concept drift occurs around 500*K*, accuracy of each method declines dramatically (as shown in Fig. 13 (*left*)). Although RCNB and HT are better than PBDS in terms of the final accuracy, PBDS gets a higher median value (as shown in Fig. 13 (*right*)). The reason may lie in the fact that our lazy sliding window-based model can track various kinds of drifts immediately, so it could update the pattern-based classifier in real-time.

Since the concept of real-world stream is uncertain and unpredictable, it is more reasonable to employ it on verifying the performance of our proposed algorithm. Fig. 14 (*left*) illustrates the accuracy changes on the CoverType. It is clear that the curves of all algorithms involve varying degrees of volatility, indicating that concept drift may exist in the dataset. Obviously, the pattern of the whole distribution of accuracies shown in Fig. 14 (*right*) demonstrates that our method is the most stable and accurate one, which also presents that PBDS is more robust to real-world environment.

In a nutshell, our approach performs better than other algorithms in the following three aspects: (1) it is more accurate than classical classifiers, and significantly better than Naive Bayes model (*e.g.* NBM); (2) it is more suitable for scenarios with different types of drift; (3) when the number of items or class attributes are relatively small, our algorithm is more efficient than other instance based approaches in terms of classification time and memory usage.

7. Threat to works

As discussed above, although PBDS is effective on concept drift data stream, outperforms most state-of-the-art classifiers on accuracy, there are some limitations. First, how to explore numerical-attributes-based patterns for classification directly is a big challenge for pattern discovery. Since only nominal data/attribute holds

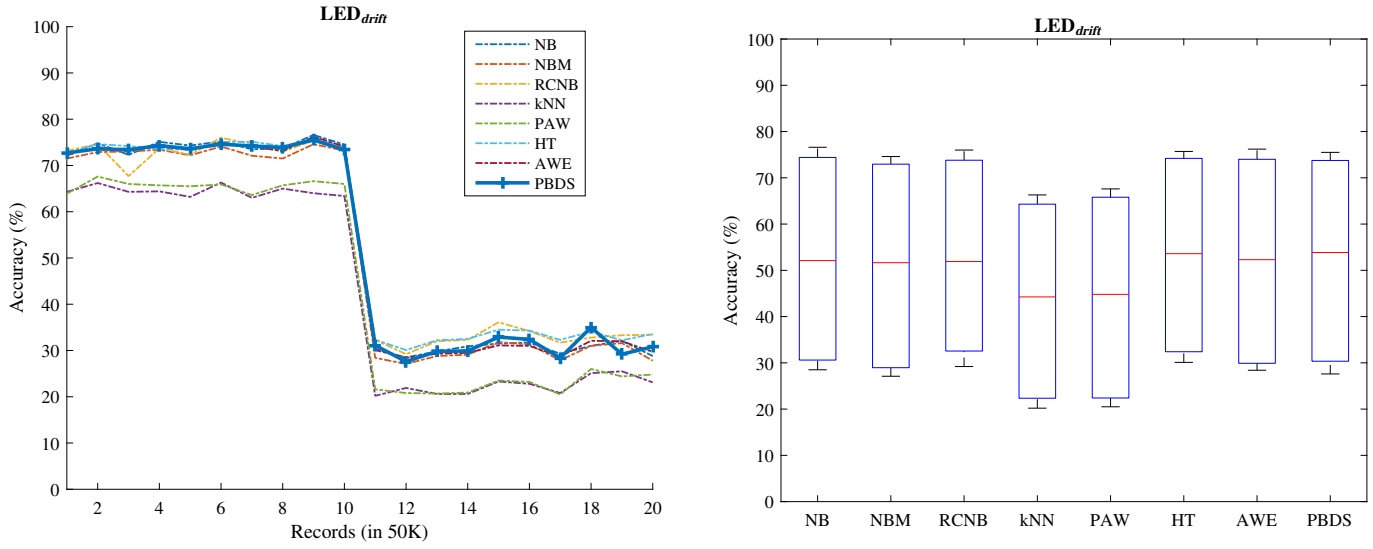


Fig. 13. Classification accuracy on the LED$_{drift}$ dataset.

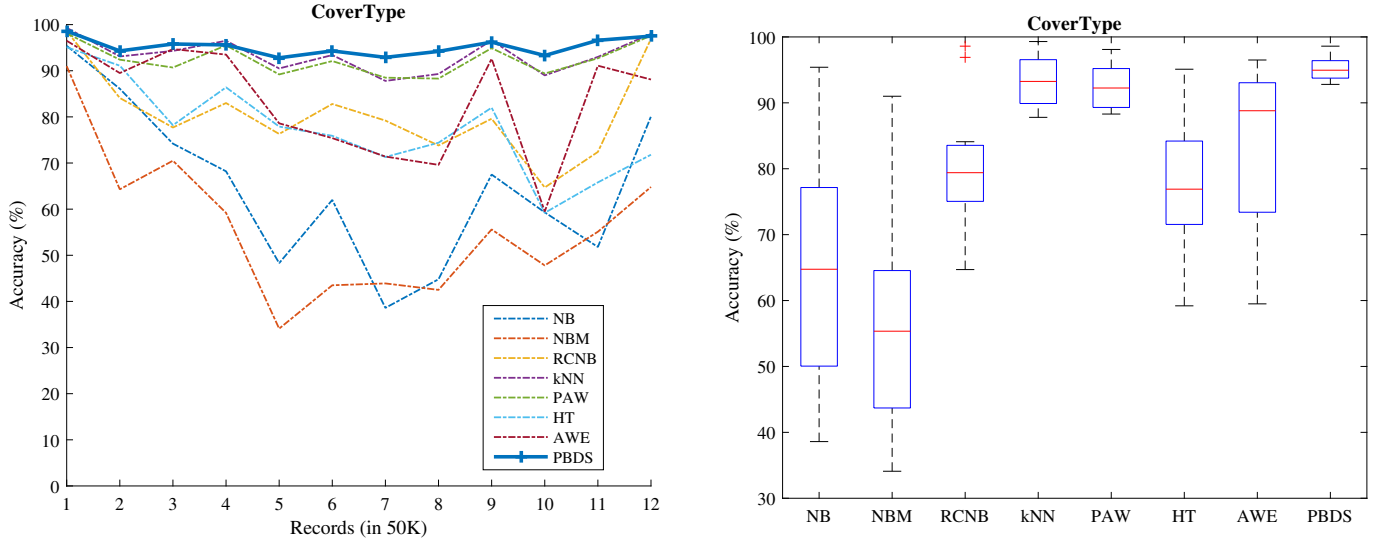


Fig. 14. Classification accuracy on the CoverType dataset.

for frequent pattern mining, and the process of discretization may lose important information for discriminative classification. Second, PBDS is still time-consuming though a compressed tree structure is proposed. More advanced and efficient data storage system should be considered in the future. Third, the parameter setting (*e.g.* the size of sliding window N) affects the final result notably, another future direction of our work could involve investigating how to create faster and more accurate pattern-based Bayesian classifier using adaptive sliding windows.

8. Conclusion

In this paper, we propose PBDS, an effective pattern-based Bayesian classifier for evolving data stream. PBDS exploits a lazy learning strategy to find local frequent patterns when a classification request occurs. A sliding window based tree structure is presented to store and process streaming data. To ensure the quality of extracted patterns, MDL-based principle is used for pattern selection. The experimental results on real and synthetic datasets show the potential of the proposed approach.

Acknowledgments

This work is supported by National Natural Science Foundation of China (nos. 61702030, 61672086, 61771058) , Beijing Natural Science Foundation (no. 4182052) and the Fundamental Research Funds for the Central Universities (no. 2016RC048).

References

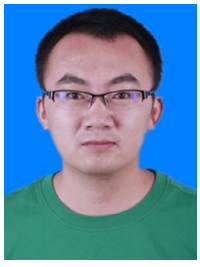
[1] E. Baralis, L. Cagliero, Rib: a robust itemset-based Bayesian approach to classification, Knowl. Based Syst. 71 (2014) 366–375.
[2] E. Baralis, L. Cagliero, P. Garza, Enbay: a novel pattern-based Bayesian classifier, IEEE Trans. Knowl. Data Eng. 25 (12) (2013) 2780–2795.
[3] B. Bringmann, S. Nijssen, A. Zimmermann, Pattern-based classification: a unifying perspective, arXiv: 1111.6191 (2011).
[4] H. Cheng, X. Yan, J. Han, C.-W. Hsu, Discriminative frequent pattern analysis for effective classification, in: 2007 IEEE 23rd International Conference on Data Engineering, IEEE, 2007, pp. 716–725.
[5] H. Fan, K. Ramamohanarao, A Bayesian approach to use emerging patterns for classification, in: Proceedings of the 14th Australasian Database Conference, vol. 17, Australian Computer Society, Inc., 2003, pp. 39–48.
[6] K. Ramamohanarao, H. Fan, Patterns based classifiers, World Wide Web 10 (1) (2007) 71–83.

[7] J. Yuan, Z. Wang, M. Han, Y. Sun, A lazy associative classifier for time series, Intell. Data Anal. 19 (5) (2015) 983–1002.
[8] D. Meretakis, B. Wüthrich, Extending Naive Bayes classifiers using long itemsets, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 1999, pp. 165–174.
[9] H.-F. Li, M.-K. Shan, S.-Y. Lee, Dsm-fi: an efficient algorithm for mining frequent itemsets in data streams, Knowl. Inf. Syst. 17 (1) (2008) 79–97.
[10] N. Jiang, L. Gruenwald, Cfi-stream: mining closed frequent itemsets in data streams, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 592–597.
[11] F. Nori, M. Deypir, M.H. Sadreddini, A sliding window based algorithm for frequent closed itemset mining over data streams, J. Syst. Softw. 86 (3) (2013) 615–623.
[12] G. Lee, U. Yun, K.H. Ryu, Sliding window based weighted maximal frequent pattern mining over data streams, Expert Syst. Applic. 41 (2) (2014) 694–708.
[13] M. Zihayat, A. An, Mining top-k high utility patterns over data streams, Inf. Sci. 285 (2014) 138–161.
[14] M. García-Borroto, J.F. Martínez-Trinidad, J.A. Carrasco-Ochoa, A survey of emerging patterns for supervised classification, Artif. Intell. Rev. 42 (4) (2014) 705–721.
[15] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 97–106.
[16] E. Ikonomovska, J. Gama, S. Džeroski, Learning model trees from evolving data streams, Data Mining Knowl. Discov. 23 (1) (2011) 128–168.
[17] J. Gama, P. Kosina, et al., Learning decision rules from data streams, in: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, 2011, p. 1255.
[18] H.M. Gomes, J.P. Barddal, F. Enembreck, A. Bifet, A survey on ensemble learning for data stream classification, ACM Comput. Surv. (CSUR) 50 (2) (2017) 23.
[19] W.N. Street, Y. Kim, A streaming ensemble algorithm (sea) for large-scale classification, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 377–382.
[20] Y. Sun, Z. Wang, H. Liu, C. Du, J. Yuan, Online ensemble using adaptive windowing for data streams with concept drift, Int. J. Distrib. Sensor Netw. 12 (5) (2016) 1–9 4218973.
[21] H. Wang, W. Fan, P.S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2003, pp. 226–235.
[22] A. Bifet, B. Pfahringer, J. Read, G. Holmes, Efficient data stream classification via probabilistic adaptive windows, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, 2013, pp. 801–806.
[23] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (2–3) (1997) 131–163.
[24] G. Webb, J. Boughton, Z. Wang, Not so Naive Bayes: aggregating one-dependence estimators, Mach. Learn. 58 (1) (2005) 5–24.
[25] Z. Farzanyar, M. Kangavari, N. Cercone, Max-fism: mining (recently) maximal frequent itemsets over data streams using the sliding window model, Comput. Math. Applic. 64 (6) (2012) 1706–1718.
[26] B.-E. Shie, S.Y. Philip, V.S. Tseng, Efficient algorithms for mining maximal high utility itemsets from data streams with different models, Expert Syst. Applic. 39 (17) (2012) 12947–12960.
[27] W. Lam, F. Bacchus, Learning Bayesian belief networks: an approach based on the MDL principle, Comput. Intell. 10 (3) (1994) 269–293.
[28] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, Moa: massive online analysis, J. Mach. Learn. Res. 11 (May) (2010) 1601–1604.
[29] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1993, pp. 1022–1027.
[30] A. McCallum, K. Nigam, et al., A comparison of event models for Naive Bayes text classification, in: AAAI-98 Workshop on Learning for Text Categorization, vol. 752, Citeseer, 1998, pp. 41–48.

Jidong Yuan received the B.S. degree in Computer Science and Technology from Northeastern University, Shenyang, China, in 2010. He received the M.S. degree and Ph.D. degree in Computer Science and Technology from Beijing Jiaotong University, Beijing, China, in 2012 and 2016, respectively. He is currently a lecturer in the School of Computer and Information Technology, Beijing Jiaotong University. His research interests include data mining and pattern recognition.

Zhihai Wang received the Ph.D. degree in Computer Application from Hefei University of Technology in 1998. He is currently a professor at the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. His research interests include data mining and artificial intelligence.

Yange Sun received the B.S. degree from the Xinyang Normal University, Xinyang, in 2004 and the M.S. degree from the Central China Normal University, Wuhan, in 2007, both in Computer Science. She is currently a Ph.D. candidate in the School of Computer and Information Technology, Beijing Jiaotong University. Her research interests include data mining and machine learning.



Wei Zhang received the B.S. degree in Mathematics and Applied Mathematics from China University of Mining and Technology, Xuzhou, China, in 2011 and the M.S. degree in Computational Mathematics from Guilin University of Electronic Technology, Guilin, China, in 2015. He is currently a Ph.D. candidate in School of Computer and Information Technology, Beijing Jiaotong University. His research interests include machine learning and data mining.

Jingjing Jiang received the B.S. degree in Computer Science and Technology from Anqing Normal University, Anqing, China, in 2014 and the M.S. degree in Computer Science and Technology, Beijing Jiaotong University, Beijing, China, in 2017. Her research interests include data mining and machine learning.