Markov Boundary Learning With Streaming Data for Supervised Classification

CHAOFAN LIU, SHUAI YANG, AND KUI YU
School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China
Corresponding author: Kui Yu (yukui@hfut.edu.cn)

This work was supported in part by the Natural Science Foundation of China under Grant 61876206.

ABSTRACT In this paper, we study the problem of Markov boundary (MB) learning with streaming data. A MB is a crucial concept in a Bayesian network (BN) and plays an important role in BN structure learning. In addition, in the supervised learning setting, the MB of a class attribute is optimal to feature selection for classification. Almost all existing MB learning algorithms focus on static data, but few efforts have been proposed to learning MBs with streaming data. In this paper, by linking dynamic AD-trees with streaming data, we proposed a new SDMB (streaming data-based MB) algorithm for learning MBs with streaming data. Specifically, given a target variable, SDMB employs a dynamic AD-tree to summarize the historical data, then the SDMB sequentially learns the MB of the target upon all available data by calculating independence tests using the dynamic AD-tree. In experiments, using the synthetic and real-world data sets, we evaluate the SDMB algorithm and compared it with the state-of-the-art online feature selection algorithms and data stream mining methods, and the experimental results validate the SDMB algorithm.

INDEX TERMS Markov boundary, Bayesian network, Streaming data, Supervised classification.

I. INTRODUCTION

The notion of a Markov boundary (MB) was coined by Pearl in a Bayesian network (BN) [21]. If a data distribution generating the data set can be faithfully represented by a BN, the MB of a node (in this paper, we will use feature, variable, and node interchangeably) is unique and consists of parents (direct causes), children (direct effects), and spouses (i.e., other parents of the node’s children) of the node in a BN. Figure 1 gives an example of a MB in the BN of lung cancer [14]. The MB of Lung cancer consists of Smoking and Genetics (parents), Coughing and Fatigue (children), and Allergy (spouse).

As can be seen in Figure 1, the MB of a node provides a complete picture of the local causal relationships between the node and the nodes in the MB. Thus learning MBs plays an essential role in local causal discovery [9], [38] and large-scale Bayesian network learning [8], [31]. For example, we can use the MB learning algorithms to find the MB of each variable in a data set for constructing a skeleton of a BN structure (i.e., undirected graph) to reduce the BN structure search spaces, then orient edges in the skeleton.

In addition, conditioning on the MB of a node, the node is independent of all other nodes. For example, given the MB of lung cancer, the lung cancer node is conditionally independent of all nodes except for the MB of lung cancer in Figure 1. Thus the MB is the theoretically optimal solution to feature selection [28], [37]. Thus learning the MB of a class variable is in fact a feature selection procedure [36], [39]. Then during the past decade, many algorithms have been proposed for learning the MB of a given feature without learning an entire BN structure. However, almost all MB learning algorithms were designed for tackling static data (i.e., in a batch fashion).

Streaming data is ubiquitous in many real-world applications, such as log files in web applications, clickstream records in E-commerce, stock data streams, and information...
from social networks. The volume of streaming data is continuously generated over time and thus may be infinite. Due to the limited memory capacity, streaming data should be processed incrementally without having access to all of the data. Then it is a challenge to summarize historical streaming data for incremental MB learning in one scan of streaming data. As new instances are continuously arriving, the distribution of streaming data may change over time. This may make a feature be a member of the MB of a target feature sometimes while not other times.

To tackle these problems, in this paper, our contributions are summarized as follows.

- We present a dynamic AD-tree to summarize historical streaming data. Then by linking the dynamic AD-tree with contingency tables, we describe the MakeCT algorithm to calculate dependence/independence relations between variables.
- Based on the dynamic AD-tree and the MakeCT algorithm, we propose a new SDMB (streaming data-based MB) algorithm to learn the MB of a given target variable in data streams. Specifically, as new data samples come, by linking the dynamic AD-tree with streaming data, SDMB first summarizes these new streaming data samples into the dynamic AD-tree, then by linking the updated dynamic AD-tree with contingency tables, SDMB calculates independence tests for learning the MB of a given target variable using the MakeCT algorithm.
- In the experiments, we have validated SDMB for standard MB learning with synthetic streaming data generated from benchmark Bayesian networks and for feature selection for classification with several state-of-the-art online feature selection algorithms and data stream classification methods (without feature selection) using synthetic and real-world data sets.

The paper is organized as follows. Section II reviews related work. Section III introduces the related notations and definitions. Section IV proposes the SDMB algorithm, and Section V gives experimental results. Section VI concludes the paper and ongoing work.

II. RELATED WORK

In the past decade, many algorithms have been proposed for learning a MB of a variable of interest from data. Margaritis and Thrun [17] proposed the GSMB algorithm, the first provably correct algorithm under the faithfulness assumption.

Several variants for improving the reliability of GSMB, such as IAMB [28], Inter-IAMB [30], and Fast-IAMB [35] were proposed. These algorithms learn PC (parents and children) and spouses of the variable simultaneously and do not distinguish PC from spouses. The algorithms require an exponential number of instances in relation to the size of the MB. Thus the quality of the MB found by these algorithms degrades greatly in practical settings with finite samples.

To tackle these drawbacks, HITON-MB [1], MMMB [29], PCMB [22], STMB [10], BAMB [15] and EEMB [32] were proposed, which take two separate steps to find the MB of a target variable: (1) discovering the PC of the target and then (2) finding its spouses. More details of existing MB learning algorithms can be found from the recent survey paper [36] and references therein.

However, all of those algorithms need to know all data samples in advance and cannot directly tackle streaming data. Thus in this paper, we will address the problem of learning MBs with streaming data.

Mining data streams has been well studied in the past decades [7], [23]. Many data stream mining algorithms have been designed for classification, and these algorithms can be roughly grouped into several categories: instance-based learning methods, decision trees learning methods, ensemble learning methods and clustering methods.

There are several commonly used methods for data stream classification, and each group has a few excellent algorithms. For example, the SAM [16], ML-SAM-kNN [24] and MLSAMPkNN [25] are typical instance-based learning methods. For ensemble learning methods, the most common approaches are Bagging and Boosting, which are firstly proposed by Oza and Russell [20], like OBA [19] and LB [4]. In many environment of data mining, ensemble learning methods can achieve the best performance, such as ARF [12], [13], RED-POS [11] and KUE [6]. And ERudesID2's [5] is a better rule-based classification method. In addition, the massive online analysis (MOA) is a better platform for streaming data mining that contains various of streaming data mining algorithms [2], [3].

III. NOTATIONS AND DEFINITIONS

In this section, we will introduce some basic definitions and notations frequently used in this paper.

A. MARKOV BOUNDARY AND BAYESIAN NETWORK

Let $V = \{V_1, ..., V_N\}$ contain $N$ observed features (nodes), $P$ be a discrete joint probability distribution over $V$, and $G$ represent a directed acyclic graph (DAG). We call the triplet $(V, G, P)$ a Bayesian network if $(V, G, P)$ satisfies the Markov condition: every variable is independent of any subset of its non-descendant variables conditioned on its parents in $G$ [21]. In a Bayesian network $< V, G, P >$, we assume that conditional independence relationships (or causal relationships) between variables can be represented by a DAG $G$ containing only directed edges ($\rightarrow$). For example, $V_i$ is a parent (cause) of $V_j$ only if $V_i \rightarrow V_j$.

In the following, $V_i \perp \perp V_j | Z$ denotes that $V_i$ and $V_j$ are statistically independent given $Z \subseteq V \setminus \{V_i, V_j\}$ while $V_i \not\perp \not\perp V_j | Z$ represents that $V_i$ and $V_j$ are dependent given $Z$. The notation $V \setminus \{V_i, V_j\}$ means that all variables in $V$ excluding both $V_i$ and $V_j$.

**Definition 1 (Faithfulness [21]):** Given a BN $< V, G, P(V) >$, $G$ is faithful to $P(V)$ if and only if every conditional independence present in $P$ is entailed by $G$ and the Markov condition. $P(V)$ is faithful to $G$ if and only if there exists a DAG $G$ such that $G$ is faithful to $P(V)$. 
The faithfulness assumption establishes a relation between a probability distribution $P$ and its underlying DAG $G$. Under the assumption, we can use conditional independence tests, instead of d-separation in graph [21], to find all dependencies or independencies entailed with a Bayesian network.

Definition 2 (Markov boundary, MB [21]): Under the faithfulness assumption, in a Bayesian network, the MB of a variable $V_i$, noted as $MB(V_i)$, is unique and consists of parents, children, and spouses (other parents of the children) of $V_i$.

Given the MB of a node, the node is probabilistically independent of any other nodes outside of the MB, which is summarized in Theorem 1, and thus the MB is the theoretically optimal solution to feature selection [28], [37].

The faithfulness assumption establishes a relation between the MB of a node, the node is probabilistically independent of any other nodes outside of the MB, which is summarized in Theorem 1, and thus the MB is the theoretically optimal solution to feature selection [28], [37].

**Theorem 1:** [21] In a Bayesian network, for $\forall X \in V \setminus \{MB(V_i), V_i\}$, $X \perp\!\!\!\perp V_i|MB(V_i)$ holds.

**B. AD-TREE**

For every instance $x_i \in \mathbb{R}^d$, let $D$ denote the set $\{x_i | x_i \in \mathbb{R}^d, i \in N^+\}$ which represents an entire streaming data set. Assuming streaming data set $D$ comes in blocks at a time, that is, $D = D_1 \cup \cdots \cup D_{i-1} \cup D_i \cup \cdots$ where $D_i$ is the $i$th data block at time $i$ and has $R$ rows and $M$ attributes $\{V_1, \cdots, V_M\}$. Assuming $V_i$ have $n_i$ distinct values and $v_{ij}$ is a value of $V_i$ ($0 < j \leq n_i$). For $\forall V_i \in V$, $i \leq h, j \leq n_i$, and $m \leq n_h$, $(V_i = v_{ij}, V_h = v_{hm}, \cdots)$ is called a query. And the count of query, $C(V_i = v_{ij}, V_h = v_{hm}, \cdots)$, denotes the number of instances which all match this query in a data set.

An all-dimensions tree (AD-tree) is a data structure that can efficiently store all data information of a large data set [18]. An AD-tree consists of two types of nodes:

- AD-node: it represents a possible query of the data set, and store the count of the query.
- Vary-node: it is the specialization for each attribute $V_i$, all its child AD-nodes need to consider the value of $V_i$.

**FIGURE 2. A simple dataset and its static AD-tree.**

In Figure 2, the rectangle nodes represent the AD-nodes and the oval nodes denote the Vary-nodes. In $D_1$, $V_1$ and $V_2$ have 2 and 3 distinct values respectively, i.e., $n_1 = 2$ and $n_2 = 3$. ($v_1 = 1$); ($v_1 = 2$); ($v_1 = 1$, $v_2 = 2$) denote four independent queries. The count of the query $C(v_1 = 1)$ is 5 in $D_1$; i.e., $C(v_1 = 1, v_2 = 2) = 5$ (* stands for any values of $V_2$).

The AD-tree structure in Figure 2 saves all data information in the form of query for $D_1$. In the structure, the root node is an AD-node representing the query ($v_1 = *$, $v_2 = *$) (i.e., all instances) and the count is 8. Starting from the root node, the rectangle AD-nodes and the oval Vary-nodes unfold alternately layer by layer. For example, in Figure 2, by expanding with the variable $V_1$, due to $n_1 = 2$, the Vary-node $V_1$ has two child AD-nodes with constraints $v_1 = 1$ and $v_2 = 2$.

In AD-tree, a Vary-node may have multiple child AD-nodes that each represents a query. A most common value (MCV) node is the child AD-node with a maximum count of a query. For example, in Figure 2, the Vary-node $V_1$ has two child AD-nodes. The AD-node ($v_1 = 1$, $v_2 = *$) is the MCV node of $V_1$, since 1 is the most common value of $V_1$, i.e., $C(v_1 = 1) = 5 > C(v_1 = 2) = 3$ in $D_1$.

To save memory space, in two cases, a NULL node was presented to replace a AD-tree node or a subtree. One is an AD-node with the count of a query up to 0. The other is the subtree of a MCV node. The count values of queries of the subtree of a MCV node can be obtained exactly and will be introduced in next section. In the Figure 2, the nodes in the dotted box or the subtrees of these nodes are set NULL in the AD-tree structure.

To deal with dynamic data, based on the AD-tree introduced above, a dynamic AD-tree [26] was proposed and is able to update the count of each query as new data instances come. We will link MB learning with streaming data by the dynamic AD-tree in next section.

**IV. PROPOSED SDMB ALGORITHM**

Almost all exiting MB learning algorithms focus on dealing with static data and cannot tackle streaming data. In this section, using the dynamic AD-tree proposed in [26] to the gap between streaming data and MB learning, we propose the SDMB (Streaming Data-based MB) algorithm to learning the MB of a target variable of interest from streaming data.

**FIGURE 3. The flow chart of the SDMB algorithm.**

A. OVERVIEW OF SDMB

Given a streaming data $D = \{D_1 \cup \cdots \cup D_{i-1} \cup D_i \cup \cdots\}$ and assuming that $MB(T)_{i-1}$ is the MB of the target $T \in V$ and was obtained at time $i - 1$, Figure 3 gives the flow chart
of the SDMB algorithm, which summarizes the main idea of SDMB. When a data block \( D_i \) arrives, SDMB first updates the counts of each query in the current dynamic AD-tree, then it calculates \( MB_i \) using the updated AD-tree. In the MB updating step, to calculate \( MB(T)_j \), SDMB first adds new features to \( MB(T)_j \), then it removes old variables from \( MB(T)_j \) due to new data block coming.

**Algorithm 1 The SDMB Algorithm**

**Require:** \( D_i \), the target variable \( T \)

**Ensure:** \( MB(T)_j (j=1, \ldots, m) \) under current and historical streaming data

1. \( MB(T)_1 \leftarrow \emptyset \)
2. repeat
3. \(/^\text{Phase 1: make/update AD-tree}\)
4. \( \text{if } D_i \text{ then} \)
5. \( \text{makeADTree}(D_i) \)
6. \( \text{delete}(D_i) \)
7. \( \text{else} \)
8. \( \text{updateADTree}(D_i) \)
9. \( \text{delete}(D_i) \)
10. \( \text{end if} \)
11. \( /^\text{Phase 2: MB learning}\)
12. \( MB(T)_j \leftarrow MB(T)_{j-1} \)
13. \( \text{while } MB(T)_j \text{ has change do} \)
14. \( \text{find a candidate member}\)
15. \( S \leftarrow \{X | X \in \emptyset \} \)
16. \( \text{MakeCT}(S \cup T, \text{root-node}) \)
17. \( \text{for each attribute } X \in S \) do
18. \( \text{calculate } \text{dep}(X, T | MB(T)_j) \)
19. \( \text{end for} \)
20. \( A \leftarrow \max \text{dep}(X, T | MB(T)_j) \)
21. \( \text{remove false positives}\)
22. \( \text{MakeCT}(MB(T)_j \cup T, \text{root-node}) \)
23. \( \text{for each attribute } X \in MB(T)_j \) do
24. \( \text{if } X \perp T | MB(T)_j \) then
25. \( MB(T)_j \leftarrow MB(T)_j \backslash \{X\} \)
26. \( \text{end if} \)
27. \( \text{end for} \)
28. \( \text{end while} \)
29. \( \text{return } MB(T)_j \)
30. until \( \text{no more } D_i \text{ are coming} \)

SDMB first constructs an initial AD-tree \( ADT_1 \) using \( D_1 \) and calculate \( MB(T)_1 \) using \( ADT_1 \). When \( D_i \) arrives, SDMB summarizes \( D_i \) to \( ADT_{i-1} \) and obtains \( ADT_i \). Due to new data blocks coming, the queries in \( ADT_i \) changes. At Phase 2, to update \( MB(T)_{i-1} \), SDMB first makes \( MB(T)_{i-1} = MB(T)_{i-1} \), then checks whether new features can be added to \( MB(T)_i \) (Lines 13 to 21) and old features is able to be removed from \( MB(T)_i \) (Lines 22 to 27). Phase 1 and Phase 2 implement alternatively until no new data blocks come. Sections IV-B and IV-C below will give the detailed descriptions of the two phases.

**B. SUMMARIZING NEW COMING DATA BLOCKS**

At Phase 1 (Lines 4 to 10), we use the dynamic AD-tree proposed in [26] to sequentially summarize historical data blocks in one scan. Initially, the AD-tree \( ADT_1 \) is an empty tree. When the first data block \( D_1 \) comes, SDMB builds \( ADT_1 \) and summarizes all queries in \( D_1 \) to \( ADT_1 \). After that, \( ADT_1 \) will be updated once a new data block \( D_i \) arrives.

Based on Figure 2, Figure 4 gives an example of how to update an AD-tree as a new data block \( D_2 \) comes. The AD-tree (called \( ADT_1 \)) in Figure 2 is built from the data block \( D_1 \). When the data block \( D_2 \) comes, \( ADT_1 \) is updated as the AD-tree (called \( ADT_2 \)) as shown in Figure 4. The AD-tree updating focuses on AD-nodes. For example, the AD-nodes of the leftmost subtree of \( ADT_2 \) is updated as follows.

First, the A arrow shows that the count value of the root node is updated to 16 with 8 records in \( D_2 \) newly arrived.

In the order of depth first, we first update the child AD-nodes of the Vary-node \( V_i \) (arrow B), for these two AD-nodes, \( C(v_1 = 1, v_2 = *) = 5 \) and \( C(v_1 = 2, v_2 = *) = 3 \) are updated to 6 and 7 respectively. And a new AD-node is built because a new value of \( V_i \) (i.e., \( v_1 = 3 \)) generates in \( D_2 \). Meanwhile, AD-node with the MCV is updated to the AD-node in which \( C(v_1 = 2, v_2 = *) = 7 \) under the Vary-node \( V_i \), and the technical details of how to update the MCV node please see the work [26].
Finally, in arrow C, the nodes have restored instead of NULL under AD-node \{v_1 = 1, v_2 = \#\}, and as the new data block \(D_2\) arrives, the AD-nodes of the Vary-node \(V_2\) are updated, \(C(v_1 = 1, v_2 = 1) = 2\), \(C(v_1 = 1, v_2 = 1) = 2\), and \(C(v_1 = 1, v_2 = 3) = 2\). Then if several AD-nodes with the same maximum count values, we set the leftmost one as the MCV node.

Since the dynamic AD-tree can store all historical streaming data, this provides the basis for incremental MB learning from streaming data. In next section, we will discuss how to learn MBs using dynamic AD-trees.

C. UPDATE MB

When a new data block \(D_j\) arrives, the current dynamic AD-tree will be updated accordingly, such as the counts of the queries and the emerging new state value of \(V_1\) as shown in Figure 4. This leads to that the MB of \(T\) found using previous data streaming may change. Assuming that the \(MB(T)_{j-1}\) is the MB of \(T\) selected at the time before \(D_j\) comes, with new data block \(D_j\) coming, some variables in \(MB(T)_{j-1}\) may not be informative of \(T\), while the variables outside of \(MB(T)_{j-1}\) may become informative. Thus Phase 2 (Lines 11 to 29) updates the current MB using the dynamic AD-tree when a new data block comes. Specifically, let \(MB(T)_{j} = MB(T)_{j-1}\). In Algorithm 1, Phase 2.1 (Lines 13 to 21) first determines whether a new variable \(X\) outside of \(MB(T)_{j}\) (i.e., \(X \in V \setminus MB(T)\)) can be added to \(MB(T)_{j}\), i.e., whether \(X \not \perp \perp T|MB(T)\) holds. Then Phase 2.2 (Lines 22 to 27) checks each variable \(Y\) in \(MB(T)_{j}\) and will remove \(Y\) if \(Y\) is independent of \(T\) given \(MB(T)_{j}\) \(\setminus Y\), i.e., \(Y \not \perp \perp T|M\{MB(T)_{j}\} \setminus Y\).

Phase 2.1 (Lines 13 to 21). At Lines 13 to 21, for each variable \(X\) outside of \(MB(T)_{j}\), i.e., \(X \in S \subseteq V \setminus MB(T)\), given \(MB(T)_{j}\), SDMB calculates the association between \(X\) and \(T\) conditioning on \(MB(T)_{j}\). If \(X\) and \(T\) have the highest association among all variables in \(S\) and \(X \not \perp \perp T|MB(T)_{j}\) holds, \(X\) will be added to \(MB(T)_{j}\) (Line 18 to 21).

Phase 2.2 (Lines 22 to 27). After adding a new feature to \(MB(T)_{j}\), Lines 22 to 27 remove old features in \(MB(T)_{j}\). For each variable \(Y\) in \(MB(T)_{j}\), if \(Y \not \perp \perp T|MB(T)_{j}\) \(\setminus Y\) holds, \(Y\) will be removed from \(MB(T)_{j}\).

At Line 13 and Line 22, to calculate the independence/dependence of \(T\) and the other variables in \(V\), the \(G^2\) test [27] is employed to calculate \(dep(X, T|MB(T)_{j})\) (i.e., \(X \not \perp \perp T|MB(T)_{j}\) or \(X \perp \perp T|MB(T)_{j}\)). To calculate \(dep(X, T|MB(T)_{j})\), we use the \(G^2\) test. The statistic of the independence test \(V_i\) and \(V_j\) conditioning on \(V_k\) using the \(G^2\) test is written as follows.

\[
G^2 = 2 \sum_{a,b,c} S_{ijk}^{abc} \ln \left( \frac{S_{ijk}^{abc}}{S_{ijk}^{ac} S_{ikj}} \right)
\]

In Eq.(1), \(S_{ijk}^{abc}\) represents the number of \(v_i = a\) in a data set, \(S_{ijk}^{ab}\) represents the number of two attributes when \(v_i = a\) and \(v_j = b\) \((a \in n_i, b \in n_j)\), and \(S_{ijk}^{abc}\) represents the number of queries \(v_i = a, v_j = b\) and \(v_k = c\) in the data set. The \(G^2\) statistic is asymptotically distributed as chi-square (\(\chi^2\)) with appropriate degrees of freedom \(f\) that is calculated as \(f = (n_i - 1)(n_j - 1)n_k\) where \(n_i\) is the number of values that \(v_i\) takes.

Assuming \(S \subseteq V\) and let \(S = \{V_{(1)}, \ldots, V_{(n)}\}\) which has \(n\) features and \(V_{(i)}\) be the \(i\)th attribute in \(S\). \(S\) has a contingency table denoted by \(ct(V_{(1)} \ldots V_{(n)})\). For example, a contingency table about \(V_1\) is shown in Figure 5, and the contingency table denotes the counts of all queries about \(V_1\) (i.e., the values that \(V_1\) takes in \(D_3\)). The four contingency tables in Figure 5 contains the all possible queries in \(D_2\) and the count of each query. Thus, if we calculate the \(G^2\) statistic in Eq.(1) using \(D_3\), the values, such as \(S_{ij}^1\) and \(S_{ij}^2\), can be read directly from the contingency table as shown Figure 5.

![Figure 5. The example of contingency tables.](image)

However, for a static data set, to get the contingency tables for calculating the \(G^2\) statistic, we need to scan the entire data set to achieve the count of each query. For streaming data, all queries of historical streaming data are stored in a dynamic AD-tree and can be retrieved from the dynamic AD-tree. To calculate the \(G^2\) test efficient for streaming data, we link the dynamic AD-tree with the \(G^2\) test by contingency tables and describe the MakeCT function in Algorithm 2.

**Algorithm 2** The MakeCT Algorithm

**Require:** subset \(\{V_{(1)} \cdots V_{(n)}\}\) of \(\{V_1 \cdots V_M\}\), root-node

**Ensure:** \(ct(V_{(1)} \cdots V_{(n)})\), \(ADN\)

1. \(\text{MakeCT}(\{V_{(1)} \cdots V_{(n)}\}, ADN)\)
2. if \(\{V_{(1)} \cdots V_{(n)}\} = \emptyset\) then
3. \(\text{return ADN.count}\)
4. else then
5. \(\text{VN = Vary-node} V_{(1)} \) (the child-node of ADN)
6. \(\text{MCV = VN.MCV}\)
7. for \(k = 1, \ldots, n_{(1)}\) do
8. if \(k == \text{MCV}\) then
9. \(\text{continue}\)
10. else then
11. \(\text{ADN}_k = \text{the child-node of VN that} v_{(1)} = k\)
12. \(\text{CT}_k = \text{MakeCT}(\{V_{(2)} \cdots V_{(n)}\}, \text{ADN}_k)\)
13. end if
14. end for
15. \(\text{CT}_{\text{MCV}}\) is calculated by Eq.(2)
16. \(\text{return the concatenation of} CT_1, \cdots, CT_{n_{(1)}}\)
17. end if
The MakeCT function learns the contingency table from dynamic AD-tree. The \( c_t(V_{(i_1)}, \ldots, V_{(i_n)}) \) can be formed by combining the flowing conditional contingency tables \( c_t(V_{(i_2)}, \ldots, V_{(i_n)})|V_{(i_1)} = 1, \ldots, c_t(V_{(i_2)}, \ldots, V_{(i_n)})|V_{(i_1)} = n_{(i_1)}) \). For example, the \( c_t(V_1, V_2) \) can be composed of \( c_t(V_2|v_1 = 1) \) and \( c_t(V_2|v_1 = 2) \), as shown in Figure 6. We can use this concatenation recursively, \( c_t(V_2, \ldots, V_{(i_n)})|V_{(i_1)} = 1) = c_t(V_{(i_2)}, \ldots, V_{(i_n)})|V_{(i_1)} = 1, V_{(i_2)} = 1) + \ldots + c_t(V_{(i_3)}, \ldots, V_{(i_n)})|V_{(i_1)} = 1, V_{(i_2)} = n_{(i_2)}, \ldots, V_{(i_n)} = n_{(i_n)} \), until each \( V_{(i_k)} \) in the conditional part, such as \( c_t(V_{(i_k)} = 1, \ldots, v_{(i_n)} = 1) \), and it can be directly read from corresponding AD-node. This is the basic idea of Algorithm 2.

MakeCT traverses the each \( C(V_{(i)} = v_{(i)}, V_{(h)} = v_{(h)_{hm}}, \ldots) \) about \( \{V_{(i_1)}, \ldots, V_{(i_n)}\} \) in the tree structure, and the traversal start at the AD-node (ADN) as the root node of tree structure. Then, the traversal falls down according to one constraint to another where the constraint is from \( V_{(i_1)} \) to \( V_{(i_n)} \) in sequence.

At Lines 2 to 3, if the first argument is empty, it returns a one-element contingency table containing the count associated with the current AD-node. At Lines 5 to 14, \( V_{(i)} \), which is the child-node of ADN, has \( n_{(i_1)} \) distinct values, in the iteration over \( k \) from 1 to \( n_{(i_1)} \), it can not get the information directly for the MCV and do not to consider it temporarily (Lines 8 to 9); for the other AD-nodes under this current \( V_{(i_1)} \), MakeCT(\( (V_{(i_2)} \ldots V_{(i_n)}) \)) for each one recursively (Lines 11 to 12). At Line 15, the contingency table of \( CT_{MVC} \) can be calculated by a row-wise subtraction Eq.(2) [18].

\[
CT_{MVC} = \text{MakeCT}(V_{(i_2)} \ldots V_{(i_n)}, \text{ADN}) - \sum_{k \neq MVC} CT_k \quad (2)
\]

This is an example for Eq.(2). For a MCV node \((v_1 = 1, v_2 = *)\) in the Figure 4, its child nodes are NULL. We can calculate the count of a query \((v_1 = 1, v_2 = 1)\) as follows.

\[
C(v_1 = 1, v_2 = 1) = C(v_1 = 1) - [C(v_1 = 1, v_2 = 2) + C(v_1 = 1, v_2 = 3)] = 6 - (2 + 2) = 2 \quad (3)
\]

D. COMPLEXITY ANALYSIS

In this section, we analyze the time complexity of the proposed SDMB algorithm. Since the SDMB consists of two phases, in the following, we will give the time complexity of each phase.

In Phase 1: making/updating AD-tree. Each node needs to go through the corresponding instances when making or updating the AD-tree, the time complexity of Phase 1 is represented by the number of instances that need to be traversed. Without loss of generality, it is assumed that all attributes are binary, thus the memory cost of the sparse AD-tree shown above is \( 2^M \). For a data set with \( R \) rows, AD-tree has at most \([\log_2 R]\) levels. Then, let \( \binom{M}{k} \) denote the number of AD-nodes in \( k \) level, so the number of AD-nodes in a tree is \( \sum_{k=0}^{\log_2 R} R/2^k \binom{M}{k} \). Thus the time complexity of making a AD-tree is \( O(\sum_{k=0}^{\log_2 R} R/2^k \binom{M}{k}) \), that it has to traverse \( R2^{-k} \) instances for each AD-nodes [18]. Then, we update the AD-tree to save the coming data set \( D_t \) with \( R \) rows. During this process, the MCVs may change, thus the sub-ADtrees may need to be built. The worst case is that the MCV of each node has changed, so the time complexity of updating AD-tree is

\[
O(\sum_{i=0}^{R} \sum_{k=0}^{\log_2 R} R \binom{M}{k} + \sum_{i=0}^{M} \sum_{k=0}^{\log_2 R} R \binom{M}{k})
\]

In Phase 2: Learning MB. Almost all MB learning algorithms calculate the dependence/independence between two variables using conditional independence tests (CI tests) [32], we use the number of CI tests to represent the time complexity of the Phase 2. For finding a candidate member (Phase 2.1), it adds at most \(|M|\) features to \( MB(T) \). Since each addition will perform \(|M|\) CI tests, in the worst case, the time complexity of entire addition phase is \( O(|M|^2) \). Similarly for removing false positives (Phase 2.2), it takes at most \( O(|M|) \) CI tests. Thus, the total time complexity of Phase 2 is \( O(|M|^2) + O(|M|) = O(|M|^2) \).

The total time complexity of the SDMB algorithm is the sum of two phases. It can be seen that the building-time is shorter than the updating-time, and the time of Phase 2 is generally much shorter than Phase 1. The specific time records can be found in the Section V-D below.

E. TRACING SDMB

In this section, we use the network of lung-cancer in Figure 1 to trace the execution of SDMB as shown in Figure 7. Lung-Cancer in black is the target feature and the \( MB(L) \) is \( \{A, C, S, F, G\} \) in blue.

As \( D_t \) comes, we update the current AD-tree and let \( MB(L)_i = MB(L)_{i-1} \). Then we add/remove the features to/from \( MB(L)_i \). The implementing details are described as follows.
FIGURE 7. MB learning for lung-cancer Bayesian network.

1) When $D_1$ comes, SDMB builds the AD-tree using $D_1$ and assuming that SDMB achieves $MB(L)_1 = \{P, G, A, Y\}$.

2) As $D_{i-1}$ arrives and assuming that $MB(L)_{i-1} = \{P, G, C, Y, F\}$ is achieved using previous data $D_1 \cup \cdots \cup D_{i-1}$. At this time, $C$ and $F$ are added to $MB(L)_{i-1}$, but $A$ is removed compared to $MB(L)_1$.

3) As $D_i$ arrives, SDMB first updates the AD-tree. Then the process of SDMB adds/removes features to/from $MB(L)_i$ are described as follows.

1). Let $MB(L)_i = MB(L)_{i-1}$ first.
2). Add $A$ (i.e., +) to $MB(L)_i$ ($A$ is added again) and do not remove any nodes (i.e., -).
3). Add new feature $S$ (i.e., +) and remove old feature $P$ and $Y$ (i.e., -).
4). The procedure is terminated until no features can be added/removed from $MB(L)_i$ and it works until new data blocks come.

V. EXPERIMENTS

In this section, we evaluate the performance of the SDMB algorithm using synthetic and real-world data sets and report the experimental results in Sections V-A, V-B, V-C and V-B.

We compare SDMB with four online feature selection algorithms, including OFS [33], SOFS [34], B-AMD and B-ARDA [40], and four data stream mining classification methods, including OBA [19], LB [4], ARF [12] and KUE [6]. We evaluate the MB selected by SDMB against these comparison algorithms using prediction accuracy.

A. EXPERIMENTS OF SDMB WITH SYNTHETIC STREAMING DATA

Since in the benchmark Bayesian networks we know the MB of each variable, and thus we can observe how the performance of the SDMB algorithm changes as new data blocks come. Then we generate four data sets using corresponding benchmark Bayesian networks as shown in Table 1.

Each data set has 100,000 data samples, we split each one into 100 data blocks and each data block has 1000 data samples. Assuming that data blocks come one by one, we sequentially evaluate the SDMB algorithm using the following metrics. For each network, we find the MBs of all variables and report the average following metrics of all variables.

- Precision. The number of true positives in the output (i.e. the variables in the output belonging to the true MB of a target variable) divided by the number of variables in the output of a MB learning algorithm.

- Recall. The number of true positives in the output divided by the number of variables included in the true MB of a target variable.

- $F_1$. $F_1 = \frac{2 \times (\text{precision} \times \text{recall})}{\text{precision} + \text{recall}}$.

Since no algorithms are proposed to learn MBs with streaming data so far. To validate SDMB using synthetic data, we assume that SDMB works in a batch fashion, called SDMB-b. Assuming that data blocks arrive one by one, SDMB learns MBs upon a data block arriving, whereas SDMB-b assumes that all data blocks are known in advance and calculates MBs using the whole of these data blocks.

Figure 8 shows that as data blocks sequentially arrive, the values of Precision, Recall and F1 of SDMB fluctuate over time using the four Bayesian networks. These fluctuations illustrate that as new data blocks come over time, the distribution of previous data may change over time. Thus as a new data block comes, SDMB may add new features or remove the old features to/from the MB set currently selected.

In Table 2, using the Alarm network, we compare SDMB with SDMB-b. We do not report the results of SDMB and SDMB-b using the other three networks since the
performance of SDMB is also the same as that of SDMB-b. In Table 2, in the row of 30000 data records, the results of Precision, Recall and F1 of SDMB are achieved by assuming that the data block sequentially arrives with 10000, 10000, and 10000 data records respectively, while those of SDMB-b are got using all 30000 data records. The meaning of remaining rows in Table 2 are the same as that of the row of 30000 data records. In Table 2, we can see that the results of SDMB-b are the same as SDMB exactly. This states that SDMB can deal with streaming data without degrading the performance compared to its batch version. We also can see that the dynamic AD-tree is able to exactly summarize all historical data information without any losses. The findings in Table 2 also give us a new way to deal with the data set with very large data samples, that is, considering the large-scale data set as streaming data and using dynamic AD-trees to summarize data information.

### B. COMPARISON WITH FOUR ONLINE LEARNING ALGORITHM

In this section, we compare SDMB with four state-of-the-art online feature selection algorithms: OFS, SOFS, B-AMD and B-ARDA for classification using four real-world data sets (Chess, Mushroom, Connect-4 and Poker) and four synthetic data sets (Alarm, Child, Insurance, and Water) as shown in Table 3. And in Alarm, Child, Insurance, and Water, we respectively selected 4th, 20th, 25th and 31th node as the class attribute. Each data set has a roughly uniform ratio of positive and negative classes and no missing values.

### TABLE 3. Benchmark datasets and real-world datasets in our experiments.

| Dataset     | \#features | \#train | \#test |
|-------------|------------|---------|--------|
| Chess       | 36         | 3,600   | 196    |
| Mushroom    | 22         | 5,500   | 144    |
| Connect-4   | 42         | 62,000  | 5,557  |
| Poker       | 10         | 820,000 | 9,201  |
| Alarm       | 36         | 95,000  | 5,000  |
| Child       | 19         | 95,000  | 5,000  |
| Insurance   | 26         | 95,000  | 5,000  |
| Water       | 31         | 95,000  | 5,000  |

OFS, SOFS, B-AMD and B-ARDA need to specify the number of selected features in advance. Thus we vary the number of selected features from 1 to 15 and report the corresponding classification accuracy as shown in Figures 9 to 15.

### FIGURE 8. F1, Recall, and Precision of SDMB as data blocks arrive over time.
And we select the best classification accuracy of each algorithm as its accuracy for comparing with SDMB. SDMB does not need to specify the number of selected features in advance. We use three classifiers, KNN, NB, and SVM, for evaluating the MB selected by SDMB.

In the experiments, according to the case that streaming data is continuously coming, we fixed the training and testing data samples as shown in Table 3. For the Mushroom and Chess data sets, we divide the training data into data blocks and each data block has 500 data samples. As for the remaining six data sets, we partition their training data sets into data blocks and each data block has 1000 data samples.

In Table 4, the best results are highlighted in bold face, we can see that SDMB is significantly better than the other
C. Liu et al.: MB Learning With Streaming Data for Supervised Classification

TABLE 4. Classification accuracy of SDMB and its online feature selection rivals.

| Data Sets   | SDMB KNN | SDMB NB | SDMB SVM | OFS    | SOFS   | B-AMD  | B-ARDA |
|-------------|----------|---------|----------|--------|--------|--------|--------|
| Chess       | 0.9643   | 0.9490  | 0.9541   | 0.8112 | 0.7148 | 0.8163 | 0.8724 |
| Mushroom    | 1        | 0.9931  | 0.9931   | 0.8632 | 0.7868 | 0.7854 | 0.8403 |
| Connect-4   | 0.6578   | 0.6896  | 0.6624   | 0.6487 | 0.6487 | 0.6487 | 0.6487 |
| Poker       | 0.7172   | 0.5877  |          |        |        |        |        |
| Alarm       | 0.9342   | 0.9342  | 0.9342   | 0.8024 | 0.8024 | 0.8024 | 0.8096 |
| Child       | 0.6858   | 0.6996  | 0.7286   | 0.6872 | 0.6934 | 0.6872 | 0.6954 |
| Insurance   | 0.9318   | 0.9206  | 0.9320   | 0.7843 | 0.7924 | 0.8163 | 0.9108 |
| Water       | 0.6190   | 0.8620  | 0.8620   | 0.5603 | 0.5589 | 0.6019 | 0.8620 |

FIGURE 15. Classification accuracy by varying the number of features on Water.

four online feature selection algorithms on both synthetic and the real-world datasets. This indicates that SDMB can find the most predictive features even with the distribution shifts. From Figures 9 to 15, we can see that OFS, SOFS, B-AMD do not change much as the number of selected features increases, while B-ARDA fluctuates more than OFS, SOFS, and B-AMD. This may possibly explain that OFS, SOFS, and B-AMD are not able to tackle distribution shifts, while B-ARDA is better than OFS, SOFS, and B-AMD. Thus, among OFS, SOFS, B-AMD and B-ARDA, B-ARDA is the best.

Moreover, compared to OFS, SOFS, B-AMD and B-ARDA, SDMB does not determine the number of selected features in advance. From Figures 9 to 15, we can see that it is not easy to determine the number of selected features for B-ARDA, especially with streaming data.

For the synthetic data sets generated from Alarm, Insurance and Water, we found that the MB learnt by SDMB is completely correct, therefore SDMB achieves the highest classification accuracy. For Child, there are 4 nodes in the true MB, we found them with the other 4 non-MB nodes, and the prediction accuracy on this dataset is relatively low. SDMB and B-ARDA get the best prediction accuracy (86.20%) as they only select one feature on dataset Water in Figure 15.

SDMB performs well in real datasets, especially on Mushroom. It selects 3 features (odor, spore-print-color and population) and they work very well. The prediction accuracy of the four comparison algorithms (except for B-ARDA) increases and tends to be stable with the increasing number of selected features on Chess. For Connect-4, the performances of all the algorithms are similar and not high and tends to be stable as the number of features is up to four.

C. EXPERIMENTS ON DATA STREAM MINING

In this section, we evaluate the SDMB algorithm on data stream mining, and report the experimental results in the following two subsections. We first compare SDMB with four state-of-the-art data stream mining classification methods, then, we conduct experiments on concept drift to evaluate explicitly adaptation of SDMB to various types of concept drift in the stream.

1) COMPARE WITH FOUR DATA STREAM MINING CLASSIFICATION METHODS

In this subsection, we compare SDMB with four state-of-the-art data stream mining classification methods: OBA, LB, ARF and KUE. We also compare them by classification accuracy on real-world and synthetic data sets. The details of these data sets is shown in Table 3. For four synthetic data sets, we selected the 4-th, 20-th, 25-th, and 31-th nodes as the class attribute.

We execute the SDMB as in Section V-B, and execute the OBA, LB, ARF and KUE classification methods within the MOA framework and the classifiers parameters are default. For the Mushroom and Chess data sets, training is performed every 500 data samples, and for other data sets is 1000.

In Table 5, the best results are highlighted in bold, we can observe that SDMB is basically superior to the other four data stream mining classification methods on the synthetic data sets, because these four data sets are mainly used as benchmark data sets for MB learning. For Child, SDMB learns 4 real MBs and 4 other non-MB nodes, the prediction accuracy on this data set is relatively low. The classification effect of SDMB on Chess and Mushroom is slightly better than the data stream classification methods, but it is worse on Connect-4 and Poker. This may possibly explain that SDMB is not able to tackle distribution drift as well as OBA, LB, ARF and KUE. And among OBA, LB, ARF and KUE, KUE is the best.

2) EXPERIMENTS OF SDMB ON CONCEPT DRIFT

In this subsection, we execute SDMB on three data sets and record the MBs of the target node. These three data sets...
TABLE 5. Classification accuracy of SDMB and its data stream mining rivals.

| Data Sets   | SDMB  | KNN  | OBA  | LB   | ARF  | KUE  |
|-------------|-------|------|------|------|------|------|
|             | K      | NB   | SVM  | K    | NB   | SVM  |
| Chess       | 0.9643 | 0.9490 | 0.9541 | 0.9160 | 0.9000 | 0.9120 | 0.9160 |
| Mushroom    | 1      | 0.9951 | 0.9951 | 0.5389 | 0.5878 | 0.8389 | 0.5800 |
| Connect-4   | 0.6879 | 0.6896 | 0.6824 | 0.6793 | 0.7021 | 0.6749 | 0.7217 |
| Alarm       | 0.9342 | 0.9542 | 0.9342 | 0.8967 | 0.8971 | 0.8908 | 0.8959 |
| Child       | 0.6858 | 0.6996 | 0.7286 | 0.9203 | 0.9224 | 0.9217 | 0.9188 |
| Insurance   | 0.9318 | 0.9206 | 0.9320 | 0.7600 | 0.7601 | 0.7592 | 0.7594 |
| Water       | 0.6190 | 0.8620 | 0.8620 | 0.5064 | 0.5243 | 0.5132 | 0.4896 |

TABLE 6. MBs of SDMB learning on three types of concept drifts in the data stream.

| Data Set       | MB1 | MB2 | MB3 | MB4 | MB5 | MB6 | MB7 | MB8 | MB9 | MB10 |
|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| No drift       | 1.46| 1.567| 1.346| 1.346| 1.346| 1.456| 1.346| 1.346| 1.346| 1.346 |
| Gradual drift  | 1.46| 1.567| 1.346| 1.346| 1.567| 1.456| 1.346| 1.346| 1.346| 1.346 |
| Abrupt drift   | 1.46| 1.567| 1.346| 1.346| 1.567| 1.456| 1.346| 1.346| 1.346| 1.346 |

TABLE 7. Runtime (in Seconds) of SDMB and its rivals.

| Data Sets | SDMB  | OFS  | SOFS | B-AMD | B-ARDA | OBA | LB | ARF | KUE  |
|-----------|-------|------|------|-------|--------|-----|----|-----|------|
| Chess     | 1.82  | 0.12 | 0.22 | 0.19  | 0.25   | 0.22| 0.09| 0.39| 0.20 |
| Mushroom  | 1.16  | 0.05 | 0.13 | 0.14  | 0.20   | 0.20| 0.17| 0.38| 0.64 |
| Connect-4 | 111.39 | 19.28 | 1.06 | 1.88 | 2.51 | 2.36 | 2.00 | 2.33 | 1.80 | 1.95 |
| Poker     | 215.1 | 5.62 | -    | -    | -      | -   | 13.86| 15.14| 15.05 | 2.41 |
| Alarm     | 255.21| 0.45 | 2.51 | 2.85 | 4.18   | 4.04| 4.64| 14.47| 13.48| 4.45 |
| Child     | 113.93| 36.00| 2.64 | 2.61 | 3.68   | 3.76| 3.14| 11.45| 11.62| 2.60 |
| Insurance | 542.73| 2.30 | 2.23 | 2.25 | 3.33   | 3.17| 3.44| 9.55 | 14.06| 2.25 |
| Water     | 226.7 | 0.11 | 2.15 | 2.30 | 3.26   | 3.28| 15.14| 14.00| 17.05 | 5.00 |

are generated by RandomTreeGenerator in MOA, including no drift, gradual drift and abrupt drift data stream with 10,000 instances, respectively. For the gradual drift, there is a drift centered around instance 5,000 with a window of 2,000 instances; and there is a drift centered around instance 5,000 with a window of 1 instance for the abrupt drift.

In Table 6, we report the MBs of SDMB learning on three types of concept drift in the stream of which $D_i$ sequentially arrives with 1000 data instances. In Table 6, 1 represents that the 1-th attribute is an MB learned by SDMB, and the meaning of remaining numbers in Table 6 are the same as number 1.

In Table 6, the different MBs that SDMB learned in each $D_i$ on three data sets are highlighted in bold. It can be seen that only on $D_6$, that is, after the arrival of the 6000-th–7000-th instances, the MBs that SDMB learned on the data sets with concept drift are different from the data set with no drift. Since the drift centered around instance 5,000, it may indicate that SDMB can reflect the concept drift, but it is not very sensitive to the concept drift. On the gradual drift and burst drift data sets, the MBs learned by SDMB is exactly the same, it means that although SDMB can not distinguish the types of various concept drift, it can adapt them. Because AD-tree saves all the data information, each MB learning is carried out under all historical data, thus SDMB can tackle the impact of concept drift, and it is not very sensitive to drift accordingly.

D. RUNNING TIME

In this section, we report the running time of SDMB and its comparison algorithms on real-world and synthetic data sets. In Table 7, we show the running time of SDMB and its rivals. For SDMB, we recorded the total time of Phase 1: make/update AD-tree (denoted as MK) and the total time of Phase 2: MB learning (denoted as MB). Since four online feature selection algorithms need to specify the number of selected features, there will be multiple running times, therefore we show the time of the experiment which get the best classification accuracy. For the four data stream mining classification methods, we record the elapsed time displayed on the MOA platform.

The fastest time are highlighted in bold face in Table 7, we can see that the total time of SDMB is much longer than its rivals on each data set. SDMB spends the majority of time on making/updating AD-tree. And the time of MB learning of SDMB is roughly similar to the other algorithms. The dynamic AD-tree structure can completely save the streaming data information in a relatively fixed memory, but to be honest, it needs to pay more time.

OFS is the fastest on most data sets of these four online feature selection algorithms, and KUE also performs best in terms of time among these four data stream mining methods. Moreover, KUE and OFS are comparable in terms of running speed.

In Table 8, we show the specific running time of the two phases in SDMB. We select the alarm and child data sets as...
examples, and we record the time of two phases (MK/MB) of $D_1$ sequentially arrives with 10000 data records. The MK-time of $D_1$ is less than each subsequent MK-time, and each MK-time after $D_1$ is similar. It means that the time for making AD-tree is shorter than the updating, and each updating time is about the same. Then, the time for MB learning is much less than making/updating AD-tree, once again, it indicates that the time of SDMB is mainly spent on making/updating AD-tree. This also verifies the analysis of the time complexity described above.

### VI. CONCLUSION AND ONGOING WORK

In this work, to learn the MB of a given target with streaming data, we proposed the SDMB algorithm. SDMB employs dynamic AD-trees to process and summarize streaming data, and sequentially learns the MB of the target upon new data samples coming using the dynamic AD-trees. The experimental results show that SDMB can tackle MB learning with streaming data, it outperforms the state-of-the-art online feature selection methods for feature selection, and it performs well as the state-of-the-art data stream classification methods on real-word data sets. Since SDMB needs to reconsider the discarded features at each time when new data samples arrive, it needs to summarize these discarded variable information to dynamic AD-trees. This results in that the dynamic AD-trees are very memory expensive and only can store no more than 50 variables. Thus SDMB is not able to deal with high-dimensional streaming data. Our ongoing work is developing new algorithms with new data structures to tackle streaming data with large number of features for both local causal structure learning and online feature selection.

### REFERENCES

[1] C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos, “Local causal and Markov blanket induction for causal discovery and feature selection for classification part I: Algorithms and empirical evaluation,” J. Mach. Learn. Res., vol. 11, pp. 171–234, Jan. 2010.

[2] A. Bifet, Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams (Frontiers in Artificial Intelligence and Applications), vol. 207. Amsterdam, The Netherlands: IOS Press, 2010.

[3] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive online analysis,” J. Mach. Learn. Res., vol. 11, pp. 1601–1604, May 2010.

[4] A. Bifet, G. Holmes, and B. Pfahringer, “Leveraging bagging for evolving data streams,” in Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases (ECML PKDD), in Lecture Notes in Computer Science, vol. 6321, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds. Barcelona, Spain: Springer, Sep. 2010, pp. 135–150.

[5] A. Cano and B. Krawczyk, “Evolving rule-based classifiers with genetic programming on GPUs for drifting data streams,” Pattern Recognit., vol. 87, pp. 248–268, Mar. 2019.

[6] A. Cano and B. Krawczyk, “Kappa updated ensemble for drifting data stream mining,” Mach. Learn., vol. 109, no. 1, pp. 175–218, Jan. 2020.

[7] M. M. Gaber, “Advances in data stream mining,” WIREs Data Mining Knowl. Discovery, vol. 2, no. 1, pp. 79–85, Jan. 2012.

[8] T. Gao, K. Budnis, and M. Campbell, “Local-to-global Bayesian network structure learning,” in Proc. 35th Int. Conf. Mach. Learn., vol. 70, 2017, pp. 1193–1202.

[9] T. Gao and Q. Ji, “Local causal discovery of direct causes and effects,” in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 2512–2520.

[10] T. Gao and Q. Ji, “Efficient Markov blanket discovery and its application,” IEEE Trans. Cybern., vol. 47, no. 5, pp. 1169–1179, May 2017.

[11] H. Ghomeshi, M. M. Gaber, and Y. Kovalchuk, “A non-canonical hybrid Metaheuristic approach to adaptive data stream classification,” Future Intell. Comput. Syst. Technol., vol. 10, no. 5, pp. 1–25, Nov. 2019.

[12] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfahringer, G. Holmes, and T. Abdessalem, “Adaptive random forests for evolving data stream classification,” Mach. Learn., vol. 106, nos. 9–10, pp. 1469–1495, Oct. 2017.

[13] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfahringer, G. Holmes, and T. Abdessalem, “Correction to: Adaptive random forests for evolving data stream classification,” Mach. Learn., vol. 108, no. 10, pp. 1877–1878, Oct. 2019.

[14] I. Guyon and C. Aliferis, “Causal feature selection,” in Computational Methods of Feature Selection. London, U.K.: Chapman & Hall, 2007, pp. 75–97.

[15] Z. Ling, K. Yu, H. Wang, L. Liu, W. Ding, and X. Wu, “BAMB: A ranked Markov blanket discovery approach to feature selection,” ACM Trans. Intell. Syst. Technol., vol. 10, no. 5, pp. 1–25, Nov. 2019.

[16] V. Losing, B. Hammer, and H. Wersing, “Tackling heterogeneous concept drift with the self-adjusting memory (SAM),” Knowl. Inf. Syst., vol. 54, no. 1, pp. 171–201, Jan. 2018.

[17] D. Margaritis and S. Thrun, “Bayesian network induction via local neighborhoods,” in Proc. Adv. Neural Inf. Process. Syst., 2000, pp. 505–511.

[18] A. Moore and M. S. Lee, “Cached sufficient statistics for efficient machine learning with large datasets,” J. Artif. Intell. Res., vol. 8, pp. 67–91, Mar. 1998.

[19] N. C. Oza, “Online bagging and boosting,” in Proc. IEEE Int. Conf. Syst., Man Cybern., Waikoloa, HI, USA, Oct. 2005, pp. 2340–2345.

[20] N. C. Oza and S. J. Russell, “Online bagging and boosting,” in Proc. 8th Int. Workshop Artif. Intell. Statist. (AISTATS), T. S. Richardson and T. S. Jaakkola, Eds. Ed. Key West, FL, USA: Society for Artificial Intelligence and Statistics, Jan. 2001, pp. 105–112.

[21] I. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Amsterdam, The Netherlands: Elsevier, 2014.

[22] J. M. Peña, R. Nilsson, J. Björkgegen, and J. Tegmér, “Towards scalable and data efficient learning of Markov boundaries,” Int. J. Approx. Reasoning, vol. 45, no. 2, pp. 211–232, Jul. 2007.

[23] B. R. Prasad and S. Agarwal, “Stream data mining: Platforms, algorithms, performance evaluators and research trends,” Int. J. Database Theory Appl., vol. 9, no. 9, pp. 201–218, Sep. 2016.

[24] M. Roseberry and A. Cano, “Multi-label kNN classifier with self adjusting memory for drifting data streams,” in Proc. 2nd Int. Workshop Learn. Unbalanced Domains, Theory Appl., 2018, pp. 23–37.

[25] M. Roseberry, B. Krawczyk, and A. Cano, “Multi-label punitive kNN with self-adjusting memory for drifting data streams,” ACM Trans. Knowl. Discov. Data. vol. 13, no. 6, pp. 60:1–60:31, 2019.

[26] M. Roseberry and A. W. Moore, “Sequential update of AD-trees,” in Proc. 23rd Int. Conf. Mach. Learn. (ICML), Pittsburgh, PA, USA, Jun. 2006, pp. 769–776.

[27] P. Spirtes, C. Glymour, and R. Scheines, Causation, Prediction, and Search (Adaptive Computation and Machine Learning), 2nd ed. Cambridge, MA, USA: MIT Press, 2000.

| Data Sets | Operation | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $D_8$ | $D_9$ | $D_{10}$ | $D_{11}$ | $D_{12}$ | $D_{13}$ | $D_{14}$ | $D_{15}$ | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alarm | MK | 502 | 2113 | 1945 | 2627 | 2221 | 1999 | 2428 | 2351 | 2336 | 2526 | 2544 | 2632 | 2182 | 2572 | 2235 | ... |
| Child | MK | 242 | 901 | 986 | 988 | 931 | 919 | 980 | 888 | 781 | 1352 | 1417 | 1359 | 865 | 966 | 787 | ... |
| Alarm | MB | 2 | 2 | 2 | 2 | 3 | 7 | 5 | 7 | 7 | 3 | 3 | 3 | 3 | 4 | ... |
| Child | MB | 5 | 8 | 18 | 18 | 20 | 35 | 31 | 30 | 57 | 61 | 62 | 67 | 71 | 71 | 30 | ... |

**TABLE 8. Runtime (in Milliseconds) of the two phases in SDMB.**
[29] I. Tsamardinos, C. F. Aliferis, and A. Statnikov, “Time and sample efficient discovery of Markov blankets and direct causal relations,” in Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), 2003, pp. 673–678.

[30] I. Tsamardinos, C. F. Aliferis, A. R. Statnikov, and E. Statnikov, “Algorithms for large scale Markov blanket discovery,” in Proc. FLAIRS Conf., vol. 2, 2003, pp. 376–380.

[31] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, “The max-min hill-climbing Bayesian network structure learning algorithm,” Mach. Learn., vol. 65, no. 1, pp. 31–78, Oct. 2006.

[32] H. Wang, Z. Ling, K. Yu, and X. Wu, “Towards efficient and effective discovery of Markov blankets for feature selection,” Inf. Sci., vol. 509, pp. 227–242, Jan. 2020.

[33] J. Wang, P. Zhao, S. C. H. Hoi, and R. Jin, “Online feature selection and its applications,” IEEE Trans. Knowl. Data Eng., vol. 26, no. 3, pp. 698–710, Mar. 2014.

[34] Y. Wu, S. C. H. Hoi, T. Mei, and N. Yu, “Large-scale online feature selection for ultra-high dimensional sparse data,” ACM Trans. Knowl. Discovery Data, vol. 11, no. 4, pp. 48:1–48:22, 2017.

[35] S. Yaramakala and D. Margaritis, “Speculative Markov blanket discovery for optimal feature selection,” in Proc. 5th IEEE Int. Conf. Data Mining (ICDM), 2005, p. 4.

[36] K. Yu, X. Guo, L. Liu, J. Li, H. Wang, Z. Ling, and X. Wu, “Causality-based feature selection: Methods and evaluations,” 2019, arXiv:1911.07147. [Online]. Available: http://arxiv.org/abs/1911.07147

[37] K. Yu, L. Liu, and J. Li, “A unified view of causal and non-causal feature selection,” 2018, arXiv:1802.05844. [Online]. Available: http://arxiv.org/abs/1802.05844

[38] K. Yu, L. Liu, and H. Chen, “Mining Markov blankets without causal sufficiency,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 12, pp. 6333–6347, Dec. 2018.

[39] K. Yu, L. Liu, J. Li, and T. D. Le, “Multi-source causal feature selection,” IEEE Trans. Pattern Anal. Mach. Intell., early access, Mar. 29, 2019, doi: 10.1109/TPAMI.2019.2908373.

[40] T. Zhai, H. Wang, F. Koriche, and Y. Gao, “Online feature selection by adaptive sub-gradient methods,” in Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases (ECML PKDD), Dublin, Ireland, Sep. 2018, pp. 430–446.

**CHAOFAN LIU** received the B.S. degree in computer network from the Hebei University of Technology, in 2018. He is currently pursuing the M.S. degree with the Hefei University of Technology, Hefei, China. His research interests include data mining and artificial intelligence.

**SHUAI YANG** received the B.S. and M.S. degrees in computer science from the Hefei University of Technology, Hefei, China, in 2016 and 2019, respectively, where he is currently pursuing the Ph.D. degree. His research interests include data mining, transfer learning, and causal discovery.

**KUI YU** received the Ph.D. degree in computer science from the Hefei University of Technology, Hefei, China, in 2013. From 2015 to 2018, he was a Research Fellow with the University of South Australia, Australia. From 2013 to 2015, he was a Postdoctoral Fellow with the School of Computing Science, Simon Fraser University, Canada. He is currently a Professor with the Hefei University of Technology. His main research interests include data mining and causal discovery.