Mining multi-dimensional concept-drifting data streams using Bayesian network classifiers

Hanen Borchani*[1], Pedro Larrañaga[1], João Gama[2], and Concha Bielza[1]

[1]Computational Intelligence Group, Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Boadilla del Monte, 28660 Madrid, Spain
[2]LIAAD-INESC Porto, Faculty of Economics, University of Porto, Rua Dr. Roberto Frias 378, 4200 Porto, Portugal

Abstract

In recent years, a plethora of approaches have been proposed to deal with the increasingly challenging task of mining concept-drifting data streams. However, most of these approaches can only be applied to uni-dimensional classification problems where each input instance has to be assigned to a single output class variable. The problem of mining multi-dimensional data streams, which includes multiple output class variables, is largely unexplored and only few streaming multi-dimensional approaches have been recently introduced. In this paper, we propose a novel adaptive method, named `Locally Adaptive-MB-MBC` (`LA-MB-MBC`), for mining streaming multi-dimensional data. To this end, we make use of multi-dimensional Bayesian network classifiers (MBCs) as models. Basically, `LA-MB-MBC` monitors the concept drift over time using the average log-likelihood score and the Page-Hinkley test. Then, if a concept drift is detected, `LA-MB-MBC` adapts the current MBC network locally around

*Corresponding author: Hanen Borchani. Tel.: +34 913363675; Fax: +34 913524819; E-mail: hanen.borchani@upm.es

each changed node. An experimental study carried out using synthetic multi-dimensional data streams shows the merits of the proposed method in terms of concept drift detection as well as classification performance.

Keywords Multi-dimensional Bayesian network classifiers, stream data mining, adaptive learning, concept drift.

1 Introduction

Nowadays, with the rapid growth of information technology, huge flows of records are generated and collected daily from a wide range of real-world applications, such as network monitoring, telecommunications data management, social networks, information filtering, fraud detection, etc. These flows are defined as *data streams*. Contrary to finite stationary databases, data streams are characterized by their concept-drifting aspect [37, 39], which means that the learned concepts and/or the underlying data distribution are not stable and may change over time. Moreover, data streams pose many challenges to computing systems due to limited memory resources (i.e., the stream can not be fully stored in memory), and time (i.e., the stream should be continuously processed and the learned classification model should be ready at any time to be used for prediction).

In recent years, the field of mining concept-drifting data streams has received an increasing attention and a plethora of approaches have been developed and deployed in several applications [1, 5, 11, 15, 17, 39]. All proposed approaches have a main objective consisting of coping with the concept drift and maintaining the classification model up-to-date along the continuous flows of data. They are usually composed of a detection method to monitor the concept drift and an adaptation method used for updating the classification model over time.

However, most of the work within this field has only been focused on mining uni-dimensional data streams where each input instance has to be assigned to a single output class variable. The problem of mining multi-dimensional data streams, where each instance has to be simultaneously associated with multiple output class variables,

remains largely unexplored and only few multi-dimensional streaming methods have been introduced [23, 30, 33, 40].

In this paper, we present a new method for mining multi-dimensional data streams based on multi-dimensional Bayesian network classifiers (MBCs). The so-called `Locally Adaptive-MB-MBC` (`LA-MB-MBC`) extends the stationary `MB-MBC` algorithm [6] to tackle the concept-drifting aspect of data streams. Basically, `LA-MB-MBC` monitors the concept drift over time using the average log-likelihood score and the Page-Hinkley test. Then, if a concept drift is detected, `LA-MB-MBC` adapts the current MBC network locally around each changed node. An experimental study carried out using synthetic multi-dimensional data streams shows the merits of the proposed adaptive method in terms of concept drift detection and classification performance.

The remainder of this paper is organized as follows. Section 2 briefly defines the multi-dimensional classification problem, then introduces multi-dimensional Bayesian network classifiers. Section 3 discusses the concept drift problem, and Section 4 reviews the related work on mining multi-dimensional data streams. Next, Section 5 introduces the proposed method for change detection and local MBC adaptation. Sections 6 and 7 cover the experimental study presenting the used data, the evaluation metrics, and a discussion on the obtained results. Finally, Section 8 rounds the paper off with some conclusions and future works.

2 Background

2.1 Multi-dimensional classification

In the traditional and more popular task of *uni-dimensional classification*, each instance in the data set is associated with a single class variable. However, in many real-world applications, more than one class variable may be required. That is, each instance in the data set has to be associated with a set of many different class variables at the same time. An example would be classifying movies at the online internet movie database (IMDb). In this case, a given movie may be classified simultaneously into three different categories, e.g. action, crime and drama. Additional examples

may include a patient suffering from multiple diseases, a text document belonging to several topics, a gene associated with multiple functional classes, etc.

Hence, the *multi-dimensional classification* problem can be viewed as an extension of the uni-dimensional classification problem where simultaneous prediction of a set of class variables is needed. Formally, it consists of finding a function f that predicts for each input instance given by a vector of m features $\mathbf{x} = (x_1, \ldots, x_m)$, a vector of d class values $\mathbf{c} = (c_1, \ldots, c_d)$, that is,

$$f : \Omega_{X_1} \times \ldots \times \Omega_{X_m} \longrightarrow \Omega_{C_1} \times \ldots \times \Omega_{C_d}$$
$$\mathbf{x} = (x_1, \ldots, x_m) \longmapsto \mathbf{c} = (c_1, \ldots, c_d)$$

where Ω_{X_i} and Ω_{C_j} denote the sample spaces of each feature variable X_i, for all $i \in \{1, \ldots, m\}$, and each class variable C_j, for all $j \in \{1, \ldots, d\}$, respectively. Note that, we consider that all class and feature variables are discrete random variables such that $|\Omega_{X_j}|$ and $|\Omega_{C_j}|$ are greater than 1.

When $|\Omega_{C_j}| = 2$ for all $j \in \{1, \ldots, d\}$, i.e., all class variables are binary, the multi-dimensional classification problem is known as a *multi-label classification* problem [25, 36, 42]. A multi-label classification problem can be easily modeled as a multi-dimensional classification problem where each label corresponds to a binary class variable. However, modeling a multi-dimensional classification problem, that possibly includes non-binary class variables, as a multi-label classification problem may require a transformation over the data set to meet multi-label framework requirements.

Since our proposed method is general and can be applied to classification problems where class variables are not necessarily binary, we opt to use, unless mentioned otherwise, the term multi-dimensional classification as a more general concept.

2.2 Multi-dimensional Bayesian network classifiers

A Bayesian network [22, 28] over a finite set $\mathbf{U} = \{X_1, \ldots, X_n\}$, $n \geq 1$, of discrete random variables is a pair $\mathcal{B} = (\mathcal{G}, \boldsymbol{\Theta})$. $\mathcal{G} = (V, A)$ is a directed acyclic graph (DAG) whose vertices V correspond to variables X_i and whose arcs A represent conditional

dependence relationships between triplets of variables. $\mathbf{\Theta}$ is a set of parameters such that each of its components $\theta_{x_i|\mathbf{pa}(x_i)} = P(x_i \mid \mathbf{pa}(x_i))$ represents the conditional probability of each possible value x_i of X_i given a set value $\mathbf{pa}(x_i)$ of $\mathbf{Pa}(X_i)$, where $\mathbf{Pa}(X_i)$ denotes the set of parents of X_i (nodes directed to X_i) in $\mathcal{G}$. The set of parameters $\mathbf{\Theta}$ is organized in tables, referred to as conditional probability tables (CPTs). $\mathcal{B}$ defines a joint probability distribution over $\mathbf{U}$ factorized according to structure $\mathcal{G}$ given by:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \mathbf{pa}(x_i)) \tag{1}$$

Two important definitions follow:

Definition 1 *Two sets of variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally independent given some set of variables $\boldsymbol{Z}$, denoted as $I(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{Z})$, iff $P(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z}) = P(\boldsymbol{X} \mid \boldsymbol{Z})$ for any assignment of values $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ of $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$, respectively, such that $P(\boldsymbol{Z} = \boldsymbol{z}) > 0$.*

Definition 2 *A Markov blanket of a variable X, denoted as $MB(X)$, is a minimal set of variables with the following property: $I(X, \boldsymbol{S} \mid MB(X))$ holds for every variable subset $\boldsymbol{S}$ with no variables in $MB(X) \cup X$.*

In other words, $MB(X)$ is a minimal set of variables conditioned by which X is conditionally independent of all the remaining variables. Under the faithfulness assumption, ensuring that all the conditional independencies in the data distribution are strictly those entailed by $\mathcal{G}$, $MB(X)$ consists of the union of the set of parents, children, and parents of children (i.e., spouses) of X [29]. For instance, as shown in Figure 1, $MB(X) = \{A, B, C, D, E\}$ which consists of the union of X parents $\{A, B\}$, its children $\{C, D\}$, and the parent of its child node D, i.e., $\{E\}$.

A multi-dimensional Bayesian networks classifier (MBC) is a Bayesian network specially designed to deal with the emerging problem of multi-dimensional classification.

Definition 3 *An MBC [38] is a Bayesian network $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ where the structure $\mathcal{G} = (V, A)$ has a restricted topology. The set of n vertices V is partitioned into two*

sets: $V_C = \{C_1, \ldots, C_d\}, d \geq 1$*, of class variables and* $V_X = \{X_1, \ldots, X_m\}, m \geq 1$*, of feature variables* $(d + m = n)$*. The set of arcs* A *is partitioned into three sets* A_C*,* A_X *and* A_{CX}*, such that:*

- $A_C \subseteq V_C \times V_C$ *is composed of the arcs between the class variables having a subgraph* $\mathcal{G}_C = (V_C, A_C)$ -class subgraph- *of* $\mathcal{G}$ *induced by* V_C*.*
- $A_X \subseteq V_X \times V_X$ *is composed of the arcs between the feature variables having a subgraph* $\mathcal{G}_X = (V_X, A_X)$ -feature subgraph- *of* $\mathcal{G}$ *induced by* V_X*.*
- $A_{CX} \subseteq V_C \times V_X$ *is composed of the arcs from the class variables to the feature variables having a subgraph* $\mathcal{G}_{CX} = (V, A_{CX})$ -bridge subgraph- *of* $\mathcal{G}$ *induced by* V *[4].*

Classification with an MBC under a 0-1 loss function is equivalent to solving the most probable explanation (MPE) problem, which consists of finding the most likely instantiation of the vector of class variables $\mathbf{c}^* = (c_1^*, \ldots, c_d^*)$ given an evidence about the input vector of feature variables $\mathbf{x} = (x_1, \ldots, x_m)$. Formally,

$$\mathbf{c}^* = (c_1^*, \ldots, c_d^*) = arg \max_{c_1, \ldots, c_d} p(C_1 = c_1, \ldots, C_d = c_d \mid \mathbf{x}) \tag{2}$$

Example 1 *An example of an MBC structure is shown in Figure 2. The class subgraph* $\mathcal{G}_C = (\{C_1, \ldots, C_4\}, A_C)$ *such that* A_C *consists of the two arcs between the class variables* C_1*,* C_2*, and* C_3*, the feature subgraph* $\mathcal{G}_X = (\{X_1, \ldots, X_8\}, A_X)$ *such that* A_X *contains the three arcs between the feature variables, and finally, the bridge subgraph* $\mathcal{G}_{CX} = (\{C_1, \ldots, C_4, X_1, \ldots, X_8\}, A_{CX})$ *such that* A_{CX} *is composed of the eight arcs from the class variables to the feature variables. As an MPE problem, we have*

$$\begin{aligned} \max_{c_1, \ldots, c_4} P(c_1, \ldots, c_4 \mid \mathbf{x}) = \max_{c_1, \ldots, c_4} & P(c_1 \mid c_2, c_3) P(c_2) P(c_3) P(c_4) \\ & \cdot P(x_1 \mid c_2, x_4) P(x_2 \mid c_1, c_2, x_5) P(x_3 \mid c_4) P(x_4 \mid c_1) \\ & \cdot P(x_5) P(x_6 \mid c_3) P(x_7 \mid c_4) P(x_8 \mid c_4, x_6) \end{aligned}$$

3 Concept drift

In uni-dimensional data streams, concept drift refers to the changes in the joint probability distribution $P(\mathbf{x}, c)$ which is the product of the class posterior distribution $P(c \mid \mathbf{x})$ and the feature distribution $P(\mathbf{x})$. Therefore, three types of concept drift can be distinguished [17, 37]: *conditional change* (also known as real concept drift) if a change occurs in $P(c \mid \mathbf{x})$; *feature change* (also known as virtual concept drift) if a change occurs in $P(\mathbf{x})$; and *dual change* if changes occur in both $P(c \mid \mathbf{x})$ and $P(\mathbf{x})$.

Depending on the rate (also known as the extent or the speed) of change, concept drift can be also categorized into either *abrupt* or *gradual*. An abrupt concept drift occurs at a specific time point by suddenly switching from one concept to another. On the contrary, in a gradual concept drift, a new concept is slowly introduced over an extended time period. An additional categorization is based on whether the concept drift is *local* or *global*. A concept drift is said to be local when it only occurs in some regions of the instance space (sub-spaces), and global when it occurs in the whole instance space [12].

Several additional concept drift categorizations may be found in literature such as the one proposed by Minku et al. [26] characterizing concept drifts according to different additional criteria, namely, severity (severe if no instance maintains its target class in the new concept, or intersected otherwise), frequency (periodic or non-periodic) and predictability (predictable or random). Concept drifts may be also *reoccurring* if previously seen concepts reappear (generally at irregular time intervals) over time, or *novelties* when some new variables or some of their respective states appear or disappear over time [16].

The same definitions and categorizations of uni-dimensional concept drift can be applied in the context of multi-dimensional data streams. In fact, the feature change involving only a change in $P(\mathbf{x})$ is exactly the same; whereas, for the conditional change, we have now a vector of d class variables $\mathbf{C} = (C_1, \ldots, C_d)$ instead of a single class variable C, i.e., the conditional change may occur in the distribution $P(\mathbf{c} \mid \mathbf{x})$. Moreover, as previously, the change is called dual when both feature and conditional

changes occur together. Furthermore, the multi-dimensional concept drift can be also categorized into *abrupt* or *gradual* depending on the rate of change, and into *local* or *global* depending on whether it occurs in some regions of the instance space or in the whole instance space, respectively.

Consequently, the main differences between the uni-dimensional and the multi-dimensional concept drifts consist mainly of the changes that may occur in the distribution and the dependence relationships between the class variables, as well as the distribution and the dependence relationships between each class variable and the set of feature variables.

Besides these categorizations, and in the context of streaming multi-label classification, Read et al. [33] discuss that concept drift may also involve a change in the *label cardinality*, that is, a change in the average number of labels associated with each instance computed as $LCard = 1/N \sum_{l=1}^{N} \sum_{j=1}^{d} c_j^{(l)}$ with $c_j^{(l)} \in \{0, 1\}$, where N denotes the total number of instances and d the number of labels (or binary class variables).

In addition, Xioufis et al. [40] consider that a multi-label data stream contains separate multiple targets (concepts) and each concept is likely to exhibit independently its own drift pattern. This assumption allows to track the drift of each concept separately using for instance the binary relevance method [18]. In fact, binary relevance proceeds by decomposing the multi-label learning problem into d independent binary classification problems, such that each binary classification problem aims to predict a single label value. However, the main drawback of this assumption is the inability to deal with the correlations that concepts may have with each other and which may drift over time.

It is important to note that the different presented types of drift are not exhaustive and the categorizations discussed here are not mutually exclusive. In our case, we particularly deal with a *local concept drift* in multi-dimensional data streams. Moreover, as mentioned later in Section 6.1, we consider for the empirical study different rates for local concept drifts, i.e., either abrupt or gradual.

4 Related work

In this section, we review the existing related works. All have been developed under the streaming multi-label classification setting, and can be viewed as extension of stationary multi-label methods to concept-drifting data streams.

Qu et al. [30] propose an ensemble of improved binary relevance (`MBR`) taking into account the dependency among labels. The basic idea is to add each classified label vector as a new feature participating in the classification of the other related labels. To cope with concept drifts, Qu et al. use a dynamic classifier ensemble jointly with a weighted majority voting strategy. No drift detection method is employed in `MBR`. In fact, the ensemble keeps a fixed number K of base classifiers, and is updated continuously over time by adding new classifiers, trained on the recent data blocks, and discarding the oldest ones. Naive Bayes, C4.5 decision tree algorithm, and support vector machines (SVM) are used as different base classifiers to test the `MBR` method.

Xioufis et al. [40] tackle a special problem when dealing with multi-label data streams, namely *class imbalance*, i.e., the skewness in the distribution of positive and negative instances for all or some labels. In fact, each label in the stream may have more negative than positive instances, and some labels may have much more positive instances than others. To deal with this problem, the authors propose a multiple windows classifier (`MWC`) that maintains two windows of fixed size for each label: one for positive instances and one for negative ones. The size N_p of the positive windows is a parameter of the approach and the size N_n of the negative windows is determined using the formula $N_n = N_p/r$, where r is another parameter of the approach, called distribution ratio. r has the role of balancing the distribution of positive and negative instances in the union of the two windows. The authors assume an independent concept drift for each label, and use a binary relevance method [18] with k-nearest neighbors (kNN) as base classifier. No drift detection method is employed in `MWC`. Positive and negative windows of each label are updated continuously over time by including new incoming instances and removing older ones.

Moreover, Kong and Yu [23] propose also an ensemble-based method for multi-

label stream classification. The idea is to use an ensemble of multiple random decision trees [41] where tree nodes are built by means of random selected testing variables and spliting values. The so-called Streaming Multi-lAbel Random Trees (`SMART`) algorithm does not include a change detection method. In fact, to handle concept drifts in the stream, the authors simply use a fading function on each tree node to gradually reduce the influence of historical data over time. The fading function consists of assigning to each old instance with time stamp t_i a weight $w(t) = 2^{-(t-t_i)/\lambda}$, where t is the current time, and λ is a parameter of the approach, called fading factor, indicating the speed of the fading effects. The higher the value of λ, the slower the weight of each instance will decay.

Finally, Read et al. [33] present a framework for generating synthetic multi-label data streams along with a novel multi-label streaming classification ensemble method based on Hoeffding trees. Their method, named $\texttt{EaHT}_{PS}$, extends the single-label incremental Hoeffding tree (HT) classifier [10] by using a multi-label definition of entropy and by training multi-label pruned sets (PS) at each leaf node of the tree. To handle concept drifts, Read et al. use the ADWIN Bagging method [5] which consists of an online bagging method extended with an adaptive sliding window (ADWIN) as a change detector. When a concept drift is detected, the worst performing classifier of the ensemble of classifiers is replaced with a new classifier. Read et al. also introduce `BRa`, `EaBR`, `EaPS`, `HTa` methods, that extend respectively binary relevance (`BR`) [18], ensembles of `BR` (`EBR`) [32], ensembles of textttPS (`EPS`) [31], and multi-label Hoeffding trees (`HT`) [8] stationary methods by including ADWIN to detect the potential concept drifts.

The presented streaming multi-label methods are summarized in Table 1. Contrary to these methods, which are all based on a multi-label setting, requiring all the class variables to be binary, our proposed adaptive method has no constraints on the cardinalities of the class variables. Moreover, these methods either do not present any drift detection method (for instance, `MBR` [30], `MWC` [40] and `SMART` [23] approaches) or they use a drift detection method and keep updating an ensemble of classifiers over time by replacing the worst performing classifier with a new one when a drift

is detected (such as $\texttt{EaHT}_{PS}$ [33] using ADWIN algorithm as a change detector). In both cases, the concept drift cannot be detected locally, and the adaptation process is basically based on ensemble updating.

In our case, we only use a single model (i.e., MBC) and our proposed drift detection method performs locally: it is based on monitoring the average local log-likelihood of each node of the MBC network using the Page-Hinkley test. Being based on MBCs, our adaptive method presents also the merit of explicitly modeling the probabilistic dependence relationships among all variables through the graphical structure component.

5 Locally Adaptive-MB-MBC method

Before providing more details about the proposed approach, let us introduce the following notation. Let $\mathcal{D} = \{\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^s, \ldots\}$ denote a multi-dimensional data stream that arrives over time in batches, such that $\mathcal{D}^s = \{(\mathbf{x}^{(1)}, \mathbf{c}^{(1)}), \ldots, (\mathbf{x}^{(N^s)}, \mathbf{c}^{(N^s)})\}$ denotes the multi-dimensional batch stream received at step s, and containing N^s instances. For each instance in the stream, the input vector $\mathbf{x} = (x_1, \ldots, x_m)$ of m feature values is associated with an output vector $\mathbf{c} = (c_1, \ldots, c_d)$ of d class values. For the sake of simplicity, and regardless of being class or feature variable, we denote by V_i each variable in the MBC, $i = 1, \ldots, n$, such that n represents the total number of variables, i.e., $n = d + m$. Given an MBC learned from $\mathcal{D}^s$, denoted MBC^s, and a new coming batch stream $\mathcal{D}^{s+1}$, the adaptive learning problem consists of firstly detecting possible concept drifts, then, if required, updating the current MBC^s, as MBC^{s+1}, to best fit the new distribution of $\mathcal{D}^{s+1}$.

In what follows, we start by presenting the proposed drift detection method in Section 5.1. Next, we introduce the MBC adaptation method in Section 5.2.

5.1 Drift detection method

The objective here is to continuously process the batches of data streams and detect the local concept drift when it occurs. As mentioned before, this local concept drift

can also be either abrupt or gradual. Our proposed detection method is based on the average local log-likelihood score and the Page-Hinkley test, and is applied locally, i.e., to each variable in the MBC network.

5.1.1 The average local log-likelihood score

The likelihood measures the probability of a data set $\mathcal{D}^s$ given the current multi-dimensional Bayesian network classifier. For convenience in the calculations, the logarithm of the likelihood is usually used:

$$\begin{aligned} LL^s &= log\ P(D^s \mid \boldsymbol{\theta}^s) \\ &= log \prod_{l=1}^{N^s} \prod_{i=1}^{n} P(v_i^{(l)} \mid \mathbf{pa}(v_i)^{(l)}, \boldsymbol{\theta}^s) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} log(\theta_{ijk}^s)^{N_{ijk}^s} \end{aligned} \tag{3}$$

where $v_i^{(l)}, \mathbf{pa}(v_i)^{(l)}$ are respectively the values of variable V_i and its parent set $\mathbf{Pa}(V_i)$ in the l^{th} instance in $\mathcal{D}^s$. r_i denotes the number of possible states of V_i, and q_i denotes the number of possible configurations that the parent set $\mathbf{Pa}(V_i)$ can take. N_{ijk}^s is the number of instances in $\mathcal{D}^s$ where variable V_i takes its k^{th} value and $\mathbf{Pa}(V_i)$ takes its j^{th} configuration.

We consider then the average log-likelihood score in $\mathcal{D}^s$, which is equal to the original log-likelihood score LL^s divided by the total number of instances N^s. This in fact will allow us to compare the likelihood of an MBC network based on different batch streams that may present different numbers of instances. Hence, using the maximum likelihood estimation for the parameters, $\hat{\theta}_{ijk}^s = \frac{N_{ijk}^s}{N_{ij}^s}$ where $N_{ij}^s = \sum_{k=1}^{r_i} N_{ijk}^s$ for every $i, \ldots, n$, the average log-likelihood can be expressed as follows:

$$\overline{LL}^s = \sum_{i=1}^{n} \frac{1}{N^s} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk}^s log \frac{N_{ijk}^s}{N_{ij}^s} \tag{4}$$

Finally, since the change should be monitored on each variable, we use *the average*

local log-likelihood of each variable V_i in the network expressed as:

$$\overline{ll}_i^s = \frac{1}{N^s} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk}^s log \frac{N_{ijk}^s}{N_{ij}^s} \tag{5}$$

5.1.2 Change point detection

In recent years, several change detection methods have been proposed to determine the point at which the concept drift occurs. As pointed out in [16], these methods can be categorized into four groups: i) methods based on sequential analysis such as the sequential probability ratio test; ii) methods based on control charts or statistical process control; iii) methods based on monitoring distributions on two different time-windows such as the ADWIN algorithm; and iv) contextual methods such as the splice system. More details about these methods and their references can be found in [16], Section 3.2.

In this work, In order to detect the change point, we make use of the Page-Hinkley (PH) test [20, 27]. The PH test is a sequential analysis technique commonly used for change detection in signal processing, and has been proven to be appropriate for detecting concept drifts in data streams [34].

In particular, we apply the PH test in order to determine whether a sequence of average local log-likelihood values of a variable V_i can be attributed to a single statistical law (null hypothesis); or it demonstrates a change in the statistical law underlying these values (change point). Let $\overline{ll}_i^1, ..., \overline{ll}_i^s$, denote the average local log-likelihood values for variavle V_i computed with Equation (5) using the first batch stream $\mathcal{D}^1$ till the last received one $\mathcal{D}^s$, respectively. To test the above hypothesis, the PH test considers first a cumulative variable CUM_i^s, defined as the *cumulated difference* between the obtained average local log-likelihood values and their mean till the current moment (i.e., the last batch $\mathcal{D}^s$):

$$CUM_i^s = \sum_{t=1}^{s} (\overline{ll}_i^t - mean_{\overline{ll}_i^t} - \delta) \tag{6}$$

where $mean_{\overline{ll}_i^t} = \frac{1}{t}\sum_{h=1}^{t} \overline{ll}_i^h$ denotes the mean of $\overline{ll}_i^1, ..., \overline{ll}_i^t$ values, and δ is a positive *tolerance parameter* corresponding to the magnitude of changes which are allowed. The maximum value MAX_i^s of variable CUM_i^t for $t = 1, ..., s$, is then computed:

$$MAX_i^s = max\Big\{CUM_i^t, t = 1, ..., s\Big\} \tag{7}$$

Next, the PH value is computed as the difference between MAX_i^s and CUM_i^s:

$$PH_i^s = MAX_i^s - CUM_i^s \tag{8}$$

When this difference is greater than a given threshold λ (i.e., $PH_i^s > \lambda$), the null hypothesis is rejected and the PH test alarms a change, otherwise, no change is signaled. Specifically, depending on the result of this test, two states can be distinguished:

- If $PH_i^s \leq \lambda$ then there is no concept drift: the distribution of the average local log-likelihood values is stable. The new batch $\mathcal{D}^s$ is deemed to come from the same distribution as the previous data set of instances.
- If $PH_i^s > \lambda$ then a concept drift is considered to have occurred: the distribution of the average local log-likelihood values is drifting. The new batch $\mathcal{D}^s$ is deemed to come from a different distribution than the previous data set of instances.

The threshold λ is a parameter allowing to control the rate of false alarms. In general, small λ values may increase the number of false alarms, whereas higher λ values may lead to a fewer false alarms but may rise at the same time the risk of missing some concept drifts.

Note that, the PH test is designed here to detect decreases in the log-likelihood, since an increase in the log-likelihood score informs that the current MBC network still fits well the new data and thus no adaptation is required. In our case, each local PH test value, PH_i^s, allows us to check if a drift occurs or not at each considered variable V_i. This in fact will locally specify where (i.e., for which set of variables) the concept drift occurs. Afterwards, the challenge is to locally update the MBC structure,

i.e., update only the parts that are in conflict with the the new coming batch stream without re-learning the whole MBC from scratch.

5.2 Local MBC adaptation

The objective here is to locally update the MBC network over time, so that if a concept drift occurs, only the changed parts in the current MBC are re-learned from the new coming batch stream and not the whole network. This presents two main challenges: First, how to locally detect the changes, and second how to update the current MBC.

To deal with these challenges, we propose the `Locally Adaptive-MB-MBC` method, outlined by Algorithm 1. Given the current network MBC^s, the new coming batch stream D^{s+1}, and the PH test parameters δ and λ, the local change detection firstly computes the average log-likelihood $\overline{ll}_i^{s+1}$ of each variable V_i using the new coming batch stream D^{s+1} (step 4), then computes the corresponding value PH_i^{s+1} (step 5). Next, if this PH_i^{s+1} value is higher than λ, then variable V_i is added to the set of nodes to be changed (steps 6 to 8). Subsequently, whenever the resulting set of $ChangedNodes$ is not empty, i.e., a drift is detected, then the $UpdateMBC$ function, outlined by Algorithm 2, is invoked to locally update the current MBC^s network (step 11); otherwise, we conclude that no drift is detected and the MBC network is kept unchanged (step 13).

Before introducing the $UpdateMBC$ algorithm, note that since the local log-likelihood computes the probability of each variable V_i given the set of its parents in the MBC structure, then a detected change for a variable V_i informs that the set of parents of the variable V_i has changed due to either the removal of some existing parents or the inclusion of new parents:

- The removal of an existing parent means that this parent was strongly relevant to V_i given D^s, and becomes either weakly relevant or irrelevant to V_i given D^{s+1}. In other words, this parent was a member of the parent set, or more broadly a member of the parents-children set of V_i, but with respect to D^{s+1}, it does not pertain to the parents-children set of V_i.

Algorithm 1 `Locally Adaptive-MB-MBC`

1. *Input*: Current MBC^s, new multi-dimensional data stream D^{s+1}, δ, λ
2. $ChangedNodes = \emptyset$
3. **for** every variable V_i **do**
4. Compute the average local log-likelihood $\overline{ll}_i^{s+1}$ using Equation (5)
5. Compute the local PH test, PH_i^{s+1}
6. **if** $PH_i^{s+1} > \lambda$ **then**
7. $ChangedNodes \leftarrow ChangedNodes \cup \{V_i\}$
8. **end if**
9. **end for**
10. **if** $ChangedNodes \neq \emptyset$ **then**
11. $MBC^{s+1} \leftarrow UpdateMBC(ChangedNodes, MBC^s, D^{s+1}, PC^s, MB^s)$
12. **else**
13. $MBC^{s+1} \leftarrow MBC^s$, i.e., no drift is detected
14. **end if**
15. **return** MBC^{s+1}

- The inclusion of a new parent means that this parent was either weakly relevant or irrelevant to V_i given D^s, and becomes strongly relevant to V_i given D^{s+1}. In other words, this parent was not a member of the parents-children set of V_i, but with respect to D^{s+1}, it should be added as a new member of the parents-children set of V_i.

Recall that, variables are defined to be *strongly relevant* if they contain information about V_i not found in all other remaining variables. That is, the strongly relevant variables are the members of the Markov blanket of V_i, and thereby, all the members in the parents-children set of V_i are also strongly relevant to V_i. On the other hand, variables are said to be *weakly relevant* if they are informative but redundant, i.e., they consist of all the variables with an undirected path to V_i which are not themselves members of the Markov blanket nor the parents-children set of V_i. Finally, variables are defined as *irrelevant* if they are not informative, and in this case, they consist of variables with no undirected path to V_i [21, 2].

Therefore, the intuition behind $UpdateMBC$ algorithm, is basically to firstly learn with D^{s+1} the new parents-children set of each changed node using the HITON-PC algorithm [2, 3], determine the sets of its old and new adjacent nodes, and then locally update the MBC structure.

$UpdateMBC$ is outlined by Algorithm 2. It takes as input the set of changed nodes, the current network MBC^s, the new coming batch stream D^{s+1}, the parents-children sets of all variables PC^s, and the Markov blanket sets of all class variables MB^s. For each variable V_i in the set of changed nodes, $UpdateMBC$ initially learns from D^{s+1} the new parents-children set of V_i, $PC(V_i)^{s+1}$, using HITON-PC algorithm (step 3). Then, it determines the set of its old adjacent nodes, i.e., $\left\{PC(V_i)^s \setminus PC(V_i)^{s+1}\right\}$ (step 4). The variables included in this set are variables that pertained to $PC(V_i)^s$ but do not pertain anymore to $PC(V_i)^{s+1}$, which means that they represent the set of variables that were strongly relevant to V_i and have become either weakly relevant or irrelevant to V_i. In this case, for each variable $OldAdj$ belonging to this set, the arc between it and V_i is removed from MBC^{s+1} (step 5), then, the parents-children and Markov blanket sets are updated accordingly. Specifically, the following rules are performed:

- Remove V_i from the parents-children set of $OldAdj$ (step 6): since the arc between V_i and $OldAdj$ was removed, V_i does not pertain anymore to the parents-children set of $OldAdj$.
- If the old adjacent node $OldAdj$ is a class variable, then update its Markov blanket $MB(OldAdj)^{s+1}$ by removing from it the changed node V_i and its parents that do not belong to the parents-children set $PC(OldAdj)^{s+1}$ of $OldAdj$ (steps 7 to 9).
- If the changed node V_i is a class variable, then update its Markov blanket $MB(V_i)^{s+1}$ by removing from it the old adjacent node $OldAdj$ and its parents that do not belong to the parents-children set of V_i, $PC(V_i)^{s+1}$ (steps 10 to 12).
- Update the Markov blanket of each class variable that belongs to the parent set of V_i, without being a parent nor a child of $OldAdj$, by removing from it the old adjacent node $OldAdj$ (steps 13 to 15).

Subsequently, $UpdateMBC$ determines the set of the new adjacent nodes of the changed node V_i, denoted as $\left\{PC(V_i)^{s+1} \setminus PC(V_i)^s\right\}$ (step 17). The variables in-

Algorithm 2 $UpdateMBC(ChangedNodes, MBC^s, D^{s+1}, PC^s, MB^s)$

1. *Initialization:* $MBC^{s+1} \leftarrow MBC^s$; $PC^{s+1} \leftarrow PC^s$; $MB^{s+1} \leftarrow MB^s$
2. **for** every variable $V_i \in ChangedNodes$ **do**
3. Learn $PC(V_i)^{s+1} \leftarrow$ HITON-PC(V_i)
   #Determine the set of the old adjacent nodes of the changed node V_i
4. **for** every variable $OldAdj \in \left\{PC(V_i)^s \setminus PC(V_i)^{s+1}\right\}$ **do**
5. Remove the arc between $OldAdj$ and V_i from MBC^{s+1}
6. $PC(OldAdj)^{s+1} \leftarrow PC(OldAdj)^{s+1} \setminus \{V_i\}$
7. **if** $OldAdj \in V_C$ **then**
8. $MB(OldAdj)^{s+1} \leftarrow MB(OldAdj)^{s+1} \setminus \left\{V_i \cup \{\mathbf{Pa}(V_i)^{s+1} \setminus PC(OldAdj)^{s+1}\}\right\}$
9. **end if**
10. **if** $V_i \in V_C$ **then**
11. $MB(V_i)^{s+1} \leftarrow MB(V_i)^{s+1} \setminus \left\{OldAdj \cup \{\mathbf{Pa}(OldAdj)^{s+1} \setminus PC(V_i)^{s+1}\}\right\}$
12. **end if**
13. **for** every class $H \in \left\{\mathbf{Pa}(V_i)^{s+1} \setminus PC(OldAdj)^{s+1}\right\}$ **do**
14. $MB(H)^{s+1} \leftarrow MB(H)^{s+1} \setminus \{OldAdj\}$
15. **end for**
16. **end for**
    #Determine the set of the new adjacent nodes of the changed node V_i
17. **for** every variable $NewAdj \in \left\{PC(V_i)^{s+1} \setminus PC(V_i)^s\right\}$ **do**
18. Insert an arc from $NewAdj$ to V_i in MBC^{s+1}
19. $PC(NewAdj)^{s+1} \leftarrow PC(NewAdj)^{s+1} \cup \{V_i\}$
20. **if** $NewAdj \in V_C$ **then**
21. $MB(NewAdj)^{s+1} \leftarrow MB(NewAdj)^{s+1} \cup \{V_i \cup \mathbf{Pa}(V_i)^{s+1}\}$
22. **end if**
23. **if** $V_i \in V_C$ **then**
24. $MB(V_i)^{s+1} \leftarrow MB(V_i)^{s+1} \cup \{NewAdj \cup \mathbf{Pa}(NewAdj)^{s+1}\}$
25. **end if**
26. **for** every class $H \in \left\{\mathbf{Pa}(V_i)^{s+1} \setminus \{NewAdj \cup PC(NewAdj)^{s+1}\}\right\}$ **do**
27. $MB(H)^{s+1} \leftarrow MB(H)^{s+1} \cup \{NewAdj\}$
28. **end for**
29. **end for**
30. **end for**
31. Learn from D^{s+1} new CPTs for nodes that have got a new parent set in MBC^{s+1}
32. **return** MBC^{s+1}; PC^{s+1}; MB^{s+1}

cluded in this set are variables that belong to $PC(V_i)^{s+1}$ but they were not previously in $PC(V_i)^s$, which means that they represent the set of variables that were weakly relevant or irrelevant to V_i and become strongly relevant to V_i. Hence, new depen-

dence relationships should be inserted between those variables and V_i verifying at each insertion that no cycles are introduced. In this case, a new arc is inserted from each new adjacent node $NewAdj$ to V_i (step 18), then the parents-children and Markov blanket sets are updated accordingly. The following rules are performed:

- Add V_i to the parents-children set of $NewAdj$ (step 19): since an arc was inserted between V_i and $NewAdj$, V_i becomes a member of the parents-children set of $NewAdj$.
- If the new adjacent node $NewAdj$ is a class variable, then update its Markov blanket $MB(NewAdj)^{s+1}$ by adding to it the changed node V_i as well as its parent set $\mathbf{Pa}(V_i)$ (steps 20 to 22).
- If the changed node V_i is a class, then update its Markov blanket $MB(V_i)^{s+1}$ by adding to it $NewAdj$ and its parent set $\mathbf{Pa}(NewAdj)$ (steps 23 to 25).
- Update the Markov blanket of each class variable that belongs to the parent set of V_i, without being a parent nor a child $NewAdj$, by adding to it the new adjacent node $NewAdj$ (steps 26 to 28).

Finally, new conditional probability tables (CPTs) are learnt from D^{s+1} for all the nodes that have got a new parent set in MBC^{s+1} (step 31), and then the updated MBC network MBC^{s+1}, the sets PC^{s+1} and MB^{s+1} are returned in step 32.

Note here that, all variables that belong to both $PC(V_i)^s$ and $PC(V_i)^{s+1}$ of a changed node V_i do not trigger any kind of change. In fact, these variables were strongly relevant to V_i and are still strongly relevant to V_i, so that the dependence relationships between them and V_i remain the same. Moreover, the order of processing the changed nodes does not affect the final result, that is, independently of the order, the updated MBC network MBC^{s+1} and the sets PC^{s+1} and MB^{s+1} will be the same by the end of the $UpdateMBC$ algorithm. This is guaranteed because the identification of the old and new adjacent nodes is performed independently for each changed node, and thereby, it is not affected by the order nor by the results of other nodes. The updating process of PC and MB sets is also ensured via simple operations such as removing or adding variables, and hence, the order of variable removal or addition will

not affect the final sets.

Example 2 *To illustrate the* `Locally Adaptive-MB-MBC` *algorithm, let us first reconsider the structure shown in Figure 2 as an example of an* MBC^s *structure learnt from a batch stream* D^s *using the* `MB-MBC` *algorithm [6]. Then, let assume that we receive afterwards a new batch stream* D^{s+1} *generated from the* MBC^{s+1} *structure shown in Figure 3. Given both* MBC^s *and* D^{s+1}*, the* `Locally Adaptive-MB-MBC` *algorithm starts by computing the average log-likelihood and the PH test for each variable in* MBC^s*. A change should be signaled for variables* C_1*,* C_4*,* X_2*, and* X_5 *by Algorithm 1, i.e.,* $ChangedNodes = \{C_1, C_4, X_2, X_5\}$*. Then, the MBC network should be locally updated via the* $UpdateMBC$ *algorithm (Algorithm 2).*

The $UpdateMBC$ *algorithm updates the local structure around each changed node, then updates accordingly the parents-children and Markov blanket sets. Note that* $UpdateMBC$ *takes as input the current network* MBC^s*, the set of* $ChangedNodes$*, the new coming batch stream* D^{s+1}*, as well as the current parents-children sets of all the variables* PC^s*, and the current Markov blankets sets of all the class variables* MB^s*, all represented in Table 2.*

In what follows, we present a trace of $UpdateMBC$ *algorithm for each variable in the* $ChangedNodes$ *set:*

- *The changed node* C_1 *(see Figure 4): Firstly, we determine the new parents-children set of* C_1 *given* D^{s+1} *using the HITON-PC algorithm (i.e., step 3 in Algorithm 2). We assume that HITON-PC detects the new parents-children set of* C_1 *correctly, so we should have* $PC(C_1)^{s+1} = \{C_2, X_2, X_4\}$*. Next, we determine the set of old and new adjacent nodes for* C_1*.*

  - For the old adjacent nodes, the steps 5 to 15 in Algorithm 2 would be performed. In this case, we have $PC(C_1)^s \setminus PC(C_1)^{s+1} = \{C_3\}$*, which means that* C_1 *has only* C_3 *as an old adjacent node. Thus, we start by removing the arc between* C_1 *and* C_3 *(step 5); update the parents-children set of* C_3 *as follows:* $PC(C_3)^{s+1} = PC(C_3)^{s+1} \setminus \{C_1\} = \{X_6\}$ *(step 6); then, since* C_3 *belongs to* V_C*, we proceed by updating also the Markov blanket of* C_3 *as follows:* $MB(C_3)^{s+1} = MB(C_3)^{s+1} \setminus$

$\left\{C_1 \cup \{\mathbf{Pa}(C_1)^{s+1} \setminus PC(C_3)^{s+1}\}\right\}$. *As it can be seen, we have* $\mathbf{Pa}(C_1)^{s+1} \setminus PC(C_3)^{s+1} = \{C_2\}$, *hence,* C_2 *should be removed from the Markov blanket of* C_3, *which results finally in:* $MB(C_3)^{s+1} = MB(C_3)^{s+1} \setminus \{C_1, C_2\} = \{X_6\}$ *(steps 7 to 9).*

Moreover, since C_1 *belongs to* V_C, *we update as well the Markov blanket of* C_1, *i.e.,* $MB(C_1)^{s+1} = MB(C_1)^{s+1} \setminus \left\{C_3 \cup \{\mathbf{Pa}(C_3)^{s+1} \setminus PC(C_1)^{s+1}\}\right\} = \{C_2, X_2, X_4, X_5\}$ *(steps 10 to 12).*

Finally, we update the Markov blanket set of each class parent of C_1 *(steps 13 to 15). In our case, we have only* C_2 *as parent of* C_1, *which does not pertain to* $PC(C_3)$, *thus* C_3 *should be removed from the Markov blanket of* C_2, *that is,* $MB(C_2)^{s+1} = MB(C_2)^{s+1} \setminus \{C_3\} = \{C_1, X_1, X_2, X_4, X_5\}$.

- For the new adjacent nodes, we have $PC(C_1)^{s+1} \setminus PC(C_1)^{s} = \emptyset$. *Thus, no new dependence relationships must be added for* C_1.

- *The changed node* C_4 *(see Figure 5): The first step is to determine the new parents-children set of* C_4 *given* D^{s+1} *and using the HITON-PC algorithm. As previously, we assume that HITON-PC detects the new parents-children set of* C_4 *correctly, so we should have* $PC(C_4)^{s+1} = \{C_3, X_3, X_7, X_8\}$.

  - Next, we determine the set of old adjacent nodes, which in our case is empty, i.e, $PC(C_4)^{s} \setminus PC(C_4)^{s+1} = \emptyset$.

  - Then, the set of new adjacent nodes which is equal to $PC(C_4)^{s+1} \setminus PC(C_4)^{s} = \{C_3\}$. *Consequently, we insert an arc from* C_3 *to* C_4 *(step 18), we update* $PC(C_3)^{s+1} = PC(C_3)^{s+1} \cup \{C_4\} = \{C_4, X_6\}$ *(step 19), and* $MB(C_3)^{s+1} = MB(C_3)^{s+1} \cup \{C_4 \cup \mathbf{Pa}(C_4)^{s+1}\} = \{C_4, X_6\}$ *(step 20 to 22). Similarly, update the Markov blanket set* $MB(C_4)^{s+1} = MB(C_4)^{s+1} \cup \{C_3 \cup \mathbf{Pa}(C_3)^{s+1}\} = \{C_3, X_3, X_7, X_8, X_6\}$ *(steps 23 to 25).* C_4 *has no more parents except* C_3, *so steps 26-28 in the* $UpdateMBC$ *algorithm are not applied in this case.*

- *The changed node* X_2 *(see Figure 6): As previously, the first step is to determine the new parents-children set of* X_2 *given* D^{s+1} *and using the HITON-PC algorithm. Assuming that HITON-PC detects the new parents-children set of*

X_2 correctly, we should have $PC(X_2)^{s+1} = \{C_1, X_7\}$.

- Next, given that $PC(X_2)^s = \{C_1, C_2, X_5\}$, the set of old adjacent nodes is determined as $PC(X_2)^s \setminus PC(X_2)^{s+1} = \{C_2, X_5\}$.

For the first old adjacent node C_2, we remove the arc between C_2 and X_2, we update $PC(C_2)^{s+1} = PC(C_2)^{s+1} \setminus \{X_2\} = \{C_1, X_1\}$, and we update $MB(C_2)^{s+1} = MB(C_2)^{s+1} \setminus \{X_2 \cup \{\mathbf{Pa}(X_2)^{s+1} \setminus PC(C_2)^{s+1}\}\}$. Here X_2 has two parents namely C_1 and X_5 (in fact X_5 is not removed yet from the set of parents of X_2 because we start by processing the old adjacent variable C_2), and since C_1 pertains to $PC(C_2)^{s+1}$, the only variables to be removed from $MB(C_2)^{s+1}$ are then X_2 and X_5, i.e., $MB(C_2)^{s+1} = \{C_1, X_1, X_4\}$.

For the second old adjacent node X_5, we remove the arc between X_5 and X_2, we update $PC(X_5)^{s+1} = PC(X_5)^{s+1} \setminus \{X_2\} = \emptyset$, then update the Markov blanket set for every class variable of X_2 that does not pertain to $PC(X_5)^{s+1}$. In our case, X_2 has only C_1 as a class parent (because both C_2 and X_5 have been already removed), so its Markov blanket is modified as follows $MB(C_1)^{s+1} = MB(C_1)^{s+1} \setminus \{X_5\} = \{C_2, X_2, X_4\}$.

- For the new adjacent nodes, we have $PC(X_2)^{s+1} \setminus PC(X_2)^s = \{X_7\}$. Thus, we insert an arc from X_7 to X_2, update $PC(X_7)^{s+1} = PC(X_7)^{s+1} \cup \{X_2\} = \{C_4, X_2\}$, then update the Markov blanket set for every class variable of X_2 that does not pertain to $PC(X_7)^{s+1}$. In our case, X_2 has only C_1 as a class parent, which is different from X_7 and not pertaining to $PC(X_7)$, so its Markov blanket is modified as follows $MB(C_1)^{s+1} = MB(C_1)^{s+1} \cup \{X_7\} = \{C_2, X_2, X_4, X_7\}$.

- *The changed node X_5 (see Figure 7): The first step is to determine the new parents-children set of X_5 given D^{s+1} and using the HITON-PC algorithm. Assuming that HITON-PC detects the new parents-children set of X_5 correctly, we obtain $PC(X_5)^{s+1} = \{C_3\}$.*

  - Then, given that $PC(X_5)^s = \{X_2\}$, we determine first the set of old adjacent nodes $PC(X_5)^s \setminus PC(X_5)^{s+1} = \{X_2\}$. Since the changed variable X_2 has been processed before the changed node X_5, we can see that the arc between these two variables has been already removed during the previous phase. More-

over, X_5 has been already removed from $PC(X_2)^{s+1}$, so there is no change for $PC(X_2)^{s+1} = \{C_1, X_7\}$. X_5 at this step has no class parents, so steps 13-15 in the $UpdateMBC$ algorithm are not applied in this case.

- For the new adjacent nodes, we have $PC(X_5)^{s+1} \setminus PC(X_5)^s = \{C_3\}$. Thus, we insert an arc from C_3 to X_5, update $PC(C_3)^{s+1} = PC(C_3)^{s+1} \cup \{X_5\} = \{C_4, X_5, X_6\}$, and update its Markov blanket set $MB(C_3)^{s+1} = MB(C_3)^{s+1} \cup \{X_5\} = \{C_4, X_5, X_6\}$. X_5 is not a class variable and has no more class parents except C_3, so no more changes have to be considered.

Note finally that, the changes performed on the local structure of each changed node lead as well to the changes of the PC and MB sets of some adjacent nodes such as, in our case, those of variables C_2, C_3 and X_7. However, some other variables do not present any change and their PC sets are kept the same, namely, X_1, X_3, X_4, X_6, and X_8. In addition, the order of processing the changed variables affects the order of the execution of some operations, however it does not affect the final result.

6 Experimental design

6.1 Data sets

We will use the following data streams:

- Synthetic multi-dimensional data streams: We randomly generated a sequence of five MBC networks, such that the first MBC network is randomly defined on a set of $d = 5$ class variables and $m = 10$ feature variables. Then, each subsequent MBC network is obtained by randomly changing the dependence relationships around a percentage p of nodes with respect to the preceding MBC network in the sequence. Depending on parameter p, we set three different configurations to test different rates of concept drift:
  - Configuration 1: no concept drift ($p = 0\%$). In this case, the same MBC network is used to sample the total number of instances in the sequence.

This aims to generate a stationary data stream and allows us to verify the resilience of the proposed algorithm to false alarms.

- Configuration 2: gradual concept drift ($p = 20\%$). The percentage of changed nodes between each consecutive MBC networks is equal to $p = 20\%$. For each selected changed node, its parent set is modified by removing the existing parents and randomly adding new ones. For the parameters, new CPTs are randomly generated for the set of changed nodes presenting new parent sets, whereas the CPTs of the non-changed nodes are kept the same as the preceding MBC.
- Configuration 3: abrupt concept drift ($p = 50\%$). Similar to configuration 2, but we fixed the percentage of changed nodes between each consecutive MBC networks to $p = 50\%$.

Afterwards, for each configuration, 5 000 instances are randomly sampled from each MBC network in the sequence, using the probabilistic logic sampling method [19], then concatenated to form a data stream of 25 000 instances.

- SynT-drift data stream provided by Read et al. [33]: In order to compare our approach against existing multi-label stream classification methods (see Section 4), namely, `BRa`, `EaBR`, `EaHT`$_{PS}$, `EaPS`, `HTa`, `MBR`, and `MWC`, we test our proposed adaptive methods on SynT-drift.

  SynT-drift is a multi-label synthetic data stream including 1 000 000 instances with $d = 8$ binary class variables and $m = 30$ binary feature variables. It is sampled using the random tree generator proposed by Domingos and Hulten [10], that constructs a decision tree by choosing attributes at random to split, and assigning a random class label to each leaf. Once the tree is built, new examples are generated by assigning uniformly random values to attributes which then determine the class label via the tree.

  Read et al. [33] included three concept drifts in SynT-drift of varying type, magnitude and extent. In the first drift, they changed only 10% of label dependencies. In the second drift, the underlying concept changes and more labels

are associated on average with each instance (i.e., the label cardinality $LCard$ changes from 1.8 to 3.0), and in the third drift, 20% of label dependencies change.

6.2 Evaluation metrics

The synthetic data streams are processed by windows of instances, and the *prequential* setting [9, 14] is used to evaluate the predictive performance of the MBC network on each window. In this setting, each incoming window is used for testing the MBC network before it is used for training, in such a way that the MBC network is always tested on instances that have not been seen before. We used the following metrics in order to assess the performance of the proposed adaptive method:

- **Mean accuracy** over the d class variables. It is defined as a class-based measure where the accuracy is calculated separately for each class variable, then averaged across all the class variables:

$$Acc_m = \frac{1}{d} \sum_{i=1}^{d} \frac{1}{N^s} \sum_{l=1}^{N^s} \delta(\hat{c}_{li}, c_{li}) \tag{9}$$

  where N^s is the size of the testing data set, $\hat{c}_{li}$ denotes the C_i class value predicted by the multi-dimensional classifier for sample l, and c_{li} denotes its corresponding true value. $\delta(\hat{c}_{li}, c_{li}) = 1$ if the predicted and true class values are equal, i.e., $\hat{c}_{li} = c_{li}$, and $\delta(\hat{c}_{li}, c_{li}) = 0$ otherwise.

- **Global accuracy** over the d-dimensional class variable (also known as *exact match* [33]). It is considered as an instance-based measure where the accuracy is calculated separately for each instance in the testing data set, then averaged across all the instances:

$$Acc_g = \frac{1}{N^s} \sum_{l=1}^{N^s} \delta(\hat{\mathbf{c}}_l, \mathbf{c}_l) \tag{10}$$

  In this more strict case, the (d-dimensional) vector of predicted classes $\hat{\mathbf{c}}_l$ is com-

pared to the vector of true classes $\mathbf{c}_l$, so that we have $\delta(\hat{\mathbf{c}}_l, \mathbf{c}_l) = 1$ if both vectors are equal in all their components, i.e., $\hat{\mathbf{c}}_l = \mathbf{c}_l$, and $\delta(\hat{\mathbf{c}}_l, \mathbf{c}_l) = 0$ otherwise.

Note that for experiments on SynT-drift data stream, the KLDiv and SHD evaluation are omitted since we do not have an original MBC network for this data. Moreover, as reported in [33], we compute the subset accuracy instead of the mean accuracy:

- **Subset accuracy**: This is an instance-based measure defined as a trade-off between the mean accuracy (which tends to be overly lenient) and the global accuracy (which tends to be overly strict). It alleviates the very strict global accuracy measure by taking into account the partial correctness of the predicted class values and is computed as:

$$Acc_{subset} = \frac{1}{N} \sum_{l=1}^{N} \frac{|\hat{\mathbf{c}}_l \cap \mathbf{c}_l|}{|\hat{\mathbf{c}}_l \cup \mathbf{c}_l|} \tag{11}$$

- **Kullback-Leibler Divergence (KLDiv)** [24]: It measures the divergence between the learned MBC networks and the original ones. The lower the KLDiv values, the better the quality of the learning algorithm.

- **Structural Hamming Distance (SHD)** [35]: It compares the structure of the learned and the original MBC networks, and is defined as the number of operations required to make two completed partially DAGs (CPDAGs) match. The operations are add or delete an undirected edge, and add, delete, or reverse the orientation of an edge. Each of these operations is penalized with the same strength by increasing the SHD by 1. In our case, since all learned and original MBCs are DAGs, we build first the CPDAGs of both learned and original MBC DAGs using the DAG-to-CPDAG algorithm [7], then we compute the SHD metric. The lower the resulting SHD value is, the better the algorithm performed.

- **Running time**: It reports the cumulative learning plus testing times in seconds.

7 Experimental results

For the first set of experiments, carried out using 15 variables (5 class variables with 3 possible values each, and 10 binary feature variables), we used the probabilistic logic sampling method [19] to randomly sample five different data streams for each configuration (i.e., for each p=0%, p=20% and p=50%). Each generated data stream includes a total number of 25 000 instances.

For the sake of comparison, we consider the `Globally Adaptive-MB-MBC` (`GA-MB-MBC`) which is also based on the `MB-MBC` algorithm [6] but differs from `LA-MB-MBC` by dealing globally with concept drift, that is, it learns the whole MBC^s network from scratch whenever a change is detected. Specifically, `GA-MB-MBC`, outlined in Algorithm 3, takes as input the current network MBC^s, the new coming batch stream D^{s+1}, and the PH test parameters δ and λ. It starts by computing the average global log-likelihood $\overline{LL}^{s+1}$ using Equation (4) (step 2), and the PH test value PH^{s+1} for the whole MBC network (step 3). Next, if PH^{s+1} is higher than λ, then a new network MBC^{s+1} is learned from D^{s+1} using the `MB-MBC` algorithm (step 5). Otherwise, i.e., $PH^s \leq \lambda$, the MBC network is kept unchanged (step 7).

Algorithm 3 `Globally Adaptive-MB-MBC`

1. *Input*: Current MBC^s, new multi-dimensional data stream D^{s+1}, δ, λ
2. Compute the global average log-likelihood $\overline{LL}^{s+1}$ using Equation (4)
3. Compute PH^{s+1}
4. **if** $PH^{s+1} > \lambda$ **then**
5. Learn a new network MBC^{s+1} from D^{s+1} using the `MB-MBC` algorithm.
6. **else**
7. $MBC^{s+1} \leftarrow MBC^s$, i.e., no drift is detected
8. **end if**
9. **return** MBC^{s+1}

We applied both `LA-MB-MBC` and `GA-MB-MBC` using three different values of λ, namely $\lambda = 1, 5, 10$, and four different block sizes, namely $block = 400, 700, 1000, 2000$. This in fact allows us to study the sensitivity of both algorithms with respect to the input parameter λ and the block size, respectively. Tables 3, 4 and 5 show the estimated performance results as mean values and standard deviations for each metric and each

method over the five randomly generated data streams. The best result for each metric is written in bold.

In Table 3, presenting the results with $p = 0\%$ (i.e., stationary data streams), we can first notice the very low sensitivity of both algorithms with respect to λ values. In fact, even if the best result for the mean accuracy is obtained with `GA-MB-MBC` with $\lambda = 1$ and block $= 1000$, and the best result for the global accuracy is obtained with `GA-MB-MBC` with $\lambda = 1$ and block $= 1000$ or block $= 2000$, both algorithms `LA-MB-MBC` and `GA-MB-MBC` present similar predictive performance for the remaining λ values (i.e., $\lambda = 5$ and $\lambda = 10$). Moreover, regarding the block size, as expected the best results are obtained with block $= 1000$ and block $= 2000$ instances; however, using blocks of 400 instances results in the worst results since having only a small number of instances may affect the quality of the learned MBCs. `LA-MB-MBC` and `GA-MB-MBC` show similar results as well for SHD, KLDiv, and running time. In order to study whether the differences in each metric performance are statistically significant or not, we performed a statistical comparison using the Friedman test followed by the Tukey-Kramer post-hoc test with a significance level $\alpha = 0.05$. We represent the values that are statistically significantly better with the symbol $\bullet$ in Table 3. For all the comparisons, it turns out that only three values were statistically significantly better, which let us state once again that the performance of both algorithms is quite similar.

In Table 4, presenting the experimental results with a drift rate $p = 20\%$, `LA-MB-MBC` is performing the best with $\lambda = 1$ and block $= 1000$ for the mean accuracy, global accuracy and KLDiv. However, the best SHD result is obtained with `GA-MB-MBC` with $\lambda = 1$ and block $= 700$. In addition, contrary to results in Table 3 ($p = 0\%$), we can observe that under a higher drift rate ($p = 20\%$), both algorithms become more sensitive to the λ value. For both `LA-MB-MBC` and `GA-MB-MBC` algorithms, the best accuracies are obtained with $\lambda = 1$ and block $= 1000$, and as long as λ increases, all the performance measures deteriorate. In fact, using higher λ values, some concept drifts cannot be detected and consequently the model cannot be updated correctly; which may affect its performance over time. In this case, we can see that `GA-MB-MBC` is more sensitive since missing the detection of a drift affects the whole MBC network. In ad-

dition, as previously, using small blocks leads to worse results than the ones obtained with blocks 1000 or 2000. Finally, we can see that there are only few statistically significant differences (represented by •) resulting from a statistical comparison of both algorithms using the Friedman test followed by the Tukey-Kramer post-hoc test with a significance level $\alpha = 0.05$, i.e., both algorithms show similar predictive performance.

Table 5 shows the experimental results with a drift rate $p = 50\%$. We may observe here that the best accuracies were obtained with `GA-MB-MBC` with $\lambda = 5$ and block $= 1000$. In general, we can conclude here that, in all λ values, `GA-MB-MBC` outperforms `LA-MB-MBC` in SHD and mean and global accuracies, however, `LA-MB-MBC` presents slightly better KLDiv values than `GA-MB-MBC`. The better performance of `GA-MB-MBC` compared to `LA-MB-MBC` can be explained by the fact that having 50% of drift affects a larger instance space (i.e., it can be viewed as a global change), and consequently it might be better to re-build all the MBC network rather than updating it locally.

In addition, we plot in Figures 8, 9 and 10 the mean and global accuracy curves for `LA-MB-MBC` and `GA-MB-MBC` algorithms, with $\lambda = 1$, block $= 1000$, and p equal to 0%, 20% and 50%, respectively. For each curve, the X-axis represents the block number, and the Y-axis represents the classification accuracy. Note that we limited this part to $\lambda = 1$, block $= 1000$, as similar conclusions are reached with the remaining configurations including different λ and block size values.

In Figure 8, we can observe that `LA-MB-MBC` and `GA-MB-MBC` curves are almost superposed showing the similar performance of both algorithms, as well as their resilience to false alarms.

In Figures 9 and 10, we can first notice that both algorithms perform well in detecting the change at blocks 5, 10, 15 and 20. With $p = 20\%$ (Figure 9) the change is more gradual, whereas, in Figure 10 with $p = 50\%$, the change is abrupt and more important fluctuations in predictive performance are present. We can also see in Figure 9 that `LA-MB-MBC` usually outperforms `GA-MB-MBC` in updating the MBC network and recuperating more quickly its performance. Nevertheless, with higher drift rate, i.e., $p = 50\%$, `GA-MB-MBC` presents a slightly better performance than `LA-MB-MBC`.

In the second set of experiments with SynT-drift data streams, we compare `LA-MB-MBC`

and GA-MB-MBC algorithms against seven multi-label classification methods. Five of them, i.e., BRa, EaBR, EaHT$_{PS}$, EaPS and HTa were proposed by [33], whereas MBR and MWC were proposed respectively in [30] and [40]. Similarly, as [33], we divide the stream into 20 windows and we report the average of subset and global accuracies across the data windows, as well as the cumulative running time in seconds. Note that both LA-MB-MBC and GA-MB-MBC are performed using $\lambda = 1$. The obtained results are reported in Table 6.

For the subset accuracy, GA-MB-MBC performs better than LA-MB-MBC, and also better than any other method except MBR and EaHT$_{PS}$. For the global accuracy, LA-MB-MBC performs better than all remaining methods except HTa. Although not the best, LA-MB-MBC presents a good performance even though SynT-drift data set is generated based on tree models, and as expected, methods based on Hoeffding trees (i.e., EaHT$_{PS}$ and HTa) provide the best accuracy results. Nevertheless, the main shortcoming of our LA-MB-MBC algorithm is the running time which is slower than all remaining methods, and this is mainly due to the testing part that involves the computation of the most probable explanation.

8 Conclusion

In this paper, we have presented a new method for mining multi-dimensional data streams, namely, LA-MB-MBC. Basically, LA-MB-MBC proceeds locally at the level of each node in the MBC network, that is, it monitors the average local log-likelihood of each node over time, then, whenever a concept drift is detected, it learns a new local structure for each changed node.

Experimental results on synthetic data streams including different rates of change were promising. In comparison against the GA-MB-MBC algorithm, LA-MB-MBC is shown to be resilient to false alarms, and also efficient in detecting the change points and adapting the MBC networks especially when there is a small percentage of change. Moreover, both considered methods show similar predictive performance and exhibit competitive accuracy results when compared with existing multi-label classification

methods.

In the future, it will be interesting to investigate the use of different exact or approximate inference methods in order to alleviate the computational burden when calculating the most probable explanation.

Acknowledgements

This work has been financially supported by projects TIN2010-20900-C04-04, Cajal Blue Brain, and by the European Commission (Grant MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data; ICT-2013-612944). The authors would like to thank Jesse Read for kindly providing the SynT-drift data stream.

References

[1] C.C. Aggarwal, *Data Streams: Models and Algorithms*, Springer, 2007.

[2] 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, *Journal of Machine Learning Research* **11** (2010), 171–234.

[3] 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 II: Analysis and extensions, *Journal of Machine Learning Research* **11** (2010), 235–284.

[4] C. Bielza, G. Li and P. Larrañaga, Multi-dimensional classification with Bayesian networks, *International Journal of Approximate Reasoning* **52**(6) (2011), 705–727.

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

[6] H. Borchani, C. Bielza, P. Martínez-Martín and P. Larrañaga, Markov blanket-based approach for learning multi-dimensional Bayesian network classifiers: An application to predict the European Quality of Life-5 Dimensions (EQ-5D) from the 39-item Parkinson's Disease Questionnaire (PDQ-39), *Journal of Biomedical Informatics* **45** (2012), 1175–1184.

[7] D.M. Chickering, Learning equivalence classes of Bayesian-network structures, *Journal of Machine Learning Research* **2** (2002), 445–498.

[8] A. Clare and R.D. King, Knowledge discovery in multi-label phenotype data, *in: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, 2001, pp. 42–53.

[9] A.P. Dawid, Statistical theory: The prequential approach, *Journal of the Royal Statistical Society:: Series A* **147**(2) (1984), 278–292.

[10] P. Domingos and G. Hulten, Mining high-speed data streams, *in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 71–80.

[11] M.M. Gaber, Advances in data stream mining, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **2**(1) (2012), 79–85.

[12] J. Gama and G. Castillo, Learning with local drift detection, *in: Proceedings of the 2nd International Conference on Advanced Data Mining and Applications*, 2006, pp. 42–55.

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

[14] J. Gama, R. Sebastião and P.P. Rodrigues, On evaluating stream learning algorithms, *Machine Learning* **90**(3) (2013), 317–346.

[15] J. Gama, *Knowledge Discovery from Data Streams*, Data Mining and Knowledge Discovery Series. Chapman & Hall/CRC, 2010.

[16] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy and A. Bouchachia. A survey on concept drift adaptation, *ACM Computing Surveys* **46**(4) (2014), 44:1–44:37.

[17] J. Gao, B. Ding, W. Fan, J. Han and P.S. Yu, Classifying data streams with skewed class distributions and concept drifts, *IEEE Internet Computing* **12**(6) (2008), 37–49.

[18] S. Godbole and S. Sarawagi, Discriminative methods for multi-labeled classification, *in: Proceedings of the 8th Pacific Asia Conference on Knowledge Discovery and Data Mining*, 2004, pp. 22–30.

[19] M. Henrion, Propagating uncertainty in Bayesian networks by probabilistic logic sampling, *in: Proceedings of the 4th Conference on the Uncertainty in Artificial Intelligence*, 1988, 149–163.

[20] D. Hinkley, Inference about the change-point from cumulative sum tests, *Biometrika* **58**(3) (1971), 509–523.

[21] R. Kohavi and G.H. John, Wrappers for feature subset selection, Artificial Intelligence **97**(1-2) (1997), 273–324.

[22] D. Koller and N. Friedman, *Probabilistic Graphical Models. Principles and Techniques*, The MIT Press, Cambridge, MA, 2009.

[23] X. Kong and P.S. Yu, An ensemble-based approach to fast classification of multi-label data streams, in: Proceedings of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2011, pp. 95–104.

[24] S. Kullback and R.A. Leibler, On information and sufficiency, *Annals of Mathematical Statistics* **22**(1) (1951), 79–86.

[25] G. Madjarov, D. Kocevb, D. Gjorgjevikja and S. Džeroskib, An extensive experimental comparison of methods for multi-label learning, *Pattern Recognition*, **45**(9) (2012), 705–727.

[26] L. Minku, A. White and X. Yao, The impact of diversity on on-line ensemble learning in the presence of concept drift, *IEEE Transactions on Knowledge and Data Engineering* **22**(5) (2010), 730–742.

[27] E. Page, Continuous inspection schemes, *Biometrika* **41**(1954), 100–115.

[28] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, 1988.

[29] J. Pearl and T.S. Verma, Equivalence and synthesis of causal models, *in: Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence*, 1990, 220–227.

[30] W. Qu, Y. Zhang, J. Zhu and Q. Qiu, Mining multi-label concept-drifting data streams using dynamic classifier ensemble, *in: Proceedings of the 1st Asian Conference on Machine Learning*, 2009, 308–321.

[31] J. Read, B. Pfahringer and G. Holmes, Multi-label classification using ensembles of pruned sets, *in: Proceedings of the 8th IEEE International Conference on Data Mining*, 2008, 995–1000.

[32] J. Read, B. Pfahringer, G. Holmes and E. Frank, Classifier chains for multi-label classification, *Machine Learning* **85**(3) (2011), 333–359.

[33] J. Read, A. Bifet, G. Holmes and B. Pfahringer, Scalable and efficient multi-label classification for evolving data streams, *Machine Learning* **88**(1) (2012), 243–272.

[34] R. Sebastião and J. Gama, A study on change detection methods, *in: Proceedings of the 14th Portuguese Conference on Artificial Intelligence*, 2009, 353–364.

[35] I. Tsamardinos, L.E. Brown and C.F. Aliferis, The max-min hill-climbing Bayesian network structure learning algorithm, *Machine Learning* **65**(1) (2006), 31–78.

[36] G. Tsoumakas and I. Katakis, Multi-label classification: An overview, *International Journal of Data Warehousing and Mining* **3**(3) (2007), 1–13.

[37] A. Tsymbal, The problem of concept drift: Definitions and related work, *Technical Report TCD-CS-2004-15*, Department of Computer Science, Trinity College Dublin, Ireland, 2004.

[38] L.C. van der Gaag and P.R. de Waal, Muti-dimensional Bayesian network classifiers, *in: Proceedings of the 3rd European Workshop on Probabilistic Graphical Models*, 2006, pp. 107–114.

[39] G. Widmer and M. Kubat, Learning in the presence of concept drift and hidden contexts, *Machine Learning* **23**(1) (1996), 69–101.

[40] E.S. Xioufis, M. Spiliopoulou, G. Tsoumakas and I. Vlahavas, Dealing with concept drift and class imbalance in multi-label stream classification, *in: Proceedings*

of the 22nd International Joint Conference on Artificial Intelligence, 2011, 1583–1588.

[41] X. Zhang, Q. Yuan, S. Zhao, W. Fan, W. Zheng and Z. Wang, Multi-label classification without the multi-label cost, *in: Proceedings of the 10th SIAM International Conference on Data Mining*, 2010, 778–789.

[42] M. Zhang and Z. Zhou, A review on multi-label learning algorithms, *IEEE Transactions on Knowledge and Data Engineering* **26**(8) (2014), 1819-1837.

Table 1: Summary of streaming multi-label classification methods.

Reference	**Method**	**Base classifier**	**Adaptation strategy**
Qu et al. [30]	Ensemble of improved binary relevance (`MBR`)	Naive Bayes, C4.5, SVM	Evolving ensemble. No detection
Xioufis et al. [40]	Multiple windows classifier (`MWC`)	kNN	Two windows of fixed size for each label. No detection
Kong and Yu [23]	Streaming multi-label random trees (`SMART`)	Random tree	Fading function. No detection
Read et al. [33]	Ensemble of multi-label Hoeffding trees with PS at the leaves ($\texttt{EaHT}_{PS}$), as well as `BRa`, `EaBR`, `EaPS`, and `HTa` methods	Hoeffding tree	Evolving ensemble. Detection using the ADWIN algorithm

Table 2: PC^s and MB^s sets for the MBC structure shown in Figure 2.

PC^s	MB^s
$PC(C_1)^s = \{C_2, C_3, X_2, X_4\}$	$MB(C_1)^s = \{C_2, C_3, X_2, X_4, X_5\}$
$PC(C_2)^s = \{C_1, X_1, X_2\}$	$MB(C_2)^s = \{C_1, C_3, X_1, X_2, X_4, X_5\}$
$PC(C_3)^s = \{C_1, X_6\}$	$MB(C_3)^s = \{C_1, C_2, X_6\}$
$PC(C_4)^s = \{X_3, X_7, X_8\}$	$MB(C_4)^s = \{X_3, X_7, X_8, X_6\}$
$PC(X_1)^s = \{C_2, X_4\}$	
$PC(X_2)^s = \{C_1, C_2, X_5\}$	
$PC(X_3)^s = \{C_4\}$	
$PC(X_4)^s = \{C_1, X_1\}$	
$PC(X_5)^s = \{X_2\}$	
$PC(X_6)^s = \{C_3, X_8\}$	
$PC(X_7)^s = \{C_4\}$	
$PC(X_8)^s = \{C_4, X_6\}$	

Table 3: Experimental results (mean ± std. dev.) over synthetic data with $p = 0\%$. Symbol • represents statistically significantly better values.

	LA-MB-MBC	GA-MB-MBC
$\lambda = 1$, block = 400		
Mean accuracy	0.513 ± 0.041	0.524 ± 0.037
Global accuracy	0.066 ± 0.020	0.074 ± 0.012
SHD	23.926 ± 5.798	19.432 ± 4.731•
KLDiv	1.017 ± 0.295	0.836 ± 0.127
Running time	$\mathbf{1182.466 \pm 61.454}$	1208.019 ± 168.250
$\lambda = 1$, block = 700		
Mean accuracy	0.529 ± 0.034	0.530 ± 0.036
Global accuracy	0.076 ± 0.020	0.081 ± 0.014
SHD	$\mathbf{18.577 \pm 6.017}$	19.411 ± 6.368
KLDiv	0.738 ± 0.163	0.642 ± 0.137
Running time	1256.061 ± 68.628	1226.992 ± 145.795
$\lambda = 1$, block = 1000		
Mean accuracy	0.532 ± 0.033	$\mathbf{0.538 \pm 0.032}$
Global accuracy	0.081 ± 0.011	$\mathbf{0.088 \pm 0.011}$•
SHD	20.576 ± 6.730	22.272 ± 7.752
KLDiv	0.614 ± 0.172	0.561 ± 0.123
Running time	1311.709 ± 189.934	1369.942 ± 236.678
$\lambda = 1$, block = 2000		
Mean accuracy	0.535 ± 0.030	0.536 ± 0.030
Global accuracy	0.085 ± 0.008	$\mathbf{0.088 \pm 0.011}$
SHD	21.400 ± 8.118	22.900 ± 7.736
KLDiv	0.544 ± 0.242	$\mathbf{0.465 \pm 0.183}$
Running time	1294.821 ± 173.761	1437.549 ± 163.353
$\lambda = 5$, block = 1000		
Mean accuracy	0.533 ± 0.034	0.535 ± 0.035
Global accuracy	0.082 ± 0.011	0.086 ± 0.012
SHD	20.000 ± 5.712	21.688 ± 8.381
KLDiv	0.632 ± 0.176	0.584 ± 0.124
Running time	1286.319 ± 203.174	1327.425 ± 284.478
$\lambda = 10$, block = 1000		
Mean accuracy	0.533 ± 0.035	0.533 ± 0.034
Global accuracy	0.081 ± 0.011	0.084 ± 0.013
SHD	20.232 ± 5.137	22.352 ± 7.603
KLDiv	0.689 ± 0.108	0.581 ± 0.103•
Running time	1300.462 ± 252.984	1346.379 ± 251.647

Table 4: Experimental results (mean ± std. dev.) over synthetic data with $p = 20\%$. Symbol • represents statistically significantly better values.

	LA-MB-MBC	GA-MB-MBC
$\lambda = 1$, block = 400		
Mean accuracy	0.499 ± 0.029	0.504 ± 0.027
Global accuracy	0.060 ± 0.007	0.064 ± 0.009
SHD	26.445 ± 7.620	20.835 ± 3.797
KLDiv	1.006 ± 0.212•	1.184 ± 0.192
Running time	1382.035 ± 178.733	$\mathbf{1226.243 \pm 75.250}$•
$\lambda = 1$, block = 700		
Mean accuracy	0.511 ± 0.031	0.499 ± 0.023
Global accuracy	0.070 ± 0.007	0.064 ± 0.004
SHD	23.949 ± 8.748	$\mathbf{20.760 \pm 6.287}$
KLDiv	0.827 ± 0.140	1.090 ± 0.279
Running time	1439.025 ± 167.930	1278.331 ± 139.580•
$\lambda = 1$, block = 1000		
Mean accuracy	$\mathbf{0.519 \pm 0.025}$	0.510 ± 0.011
Global accuracy	$\mathbf{0.073 \pm 0.008}$	0.070 ± 0.010
SHD	23.976 ± 8.371	22.776 ± 7.278
KLDiv	$\mathbf{0.657 \pm 0.137}$	0.871 ± 0.340
Running time	1478.655 ± 223.565	1420.196 ± 215.475
$\lambda = 1$, block = 2000		
Mean accuracy	0.503 ± 0.018	0.504 ± 0.008
Global accuracy	0.068 ± 0.005	0.068 ± 0.010
SHD	29.500 ± 9.278	24.217 ± 8.36•
KLDiv	0.722 ± 0.224	0.860 ± 0.196
Running time	1663.217 ± 101.465	1499.978 ± 160.857
$\lambda = 5$, block = 1000		
Mean accuracy	0.513 ± 0.031	0.495 ± 0.025
Global accuracy	0.070 ± 0.012	0.064 ± 0.006
SHD	25.872 ± 7.503	23.624 ± 7.400
KLDiv	0.820 ± 0.203	1.150 ± 0.366
Running time	1401.360 ± 207.668	1370.247 ± 210.343
$\lambda = 10$, block = 1000		
Mean accuracy	0.506 ± 0.034	0.490 ± 0.027
Global accuracy	0.067 ± 0.011	0.063 ± 0.006
SHD	27.408 ± 7.043	23.928 ± 6.861•
KLDiv	0.981 ± 0.295	1.170 ± 0.328
Running time	1417.753 ± 235.996	1396.702 ± 192.921

Table 5: Experimental results (mean ± std. dev.) over synthetic data with $p = 50\%$. Symbol • represents statistically significantly better values.

	LA-MB-MBC	GA-MB-MBC
$\lambda = 1$, block = 400		
Mean accuracy	0.479 ± 0.016	0.474 ± 0.028
Global accuracy	0.057 ± 0.007	0.058 ± 0.015
SHD	24.690 ± 4.951	20.565 ± 3.384
KLDiv	1.096 ± 0.234	1.423 ± 0.259
Running time	1343.131 ± 142.840	$\mathbf{1218.967 \pm 93.073}$•
$\lambda = 1$, block = 700		
Mean accuracy	0.476 ± 0.013•	0.473 ± 0.032
Global accuracy	0.059 ± 0.008	0.061 ± 0.016
SHD	26.720 ± 4.219	21.389 ± 3.189•
KLDiv	1.074 ± 0.194	1.404 ± 0.463
Running time	1439.041 ± 185.292	1304.240 ± 131.041
$\lambda = 1$, block = 1000		
Mean accuracy	0.479 ± 0.023	0.484 ± 0.016
Global accuracy	0.064 ± 0.010	0.067 ± 0.012•
SHD	26.240 ± 3.156	21.000 ± 2.930•
KLDiv	$\mathbf{0.871 \pm 0.250}$	1.165 ± 0.570
Running time	1559.923 ± 176.562	1420.462 ± 119.949
$\lambda = 1$, block = 2000		
Mean accuracy	0.465 ± 0.015	0.468 ± 0.012
Global accuracy	0.058 ± 0.005	0.061 ± 0.005
SHD	30.183 ± 5.275	23.050 ± 4.975•
KLDiv	0.921 ± 0.103	0.995 ± 0.304
Running time	1673.916 ± 116.459	1569.938 ± 120.646
$\lambda = 5$, block = 1000		
Mean accuracy	0.473 ± 0.022	$\mathbf{0.490 \pm 0.005}$
Global accuracy	0.057 ± 0.004	$\mathbf{0.071 \pm 0.007}$•
SHD	25.752 ± 3.600	$\mathbf{20.376 \pm 3.215}$•
KLDiv	1.004 ± 0.219	1.011 ± 0.464
Running time	1491.192 ± 199.937	1409.244 ± 126.923
$\lambda = 10$, block = 1000		
Mean accuracy	0.468 ± 0.020	0.489 ± 0.007
Global accuracy	0.054 ± 0.007	0.070 ± 0.008
SHD	28.224 ± 5.012	20.736 ± 3.380•
KLDiv	1.144 ± 0.239	1.053 ± 0.398
Running time	1407.993 ± 248.196	1403.722 ± 133.458

Table 6: Experimental results over SynT-drift data.

	Global accuracy	Subset accuracy	Running time
BRa	0.018	0.196	62
EaBR	0.015	0.195	375
EaHT$_{PS}$	0.026	**0.221**	34
EaPS	0.030	0.184	628
HTa	**0.046**	0.164	**14**
MBR	0.020	0.199	678
MWC	0.014	0.159	1869
LA-MB-MBC	0.040	0.173	3714
GA-MB-MBC	0.038	0.198	3097

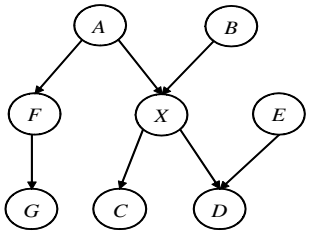

Figure 1: The Markov blanket of X denoted $MB(X)$ consists of the union of its parents $\{A, B\}$, its children $\{C, D\}$, and the parent $\{E\}$ of its child D

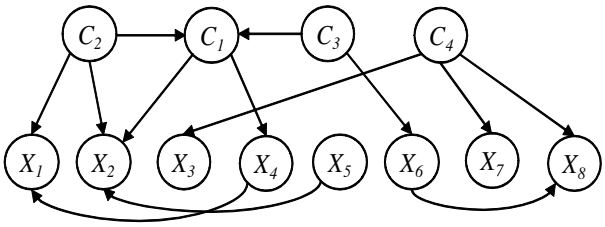

Figure 2: An example of an MBC structure

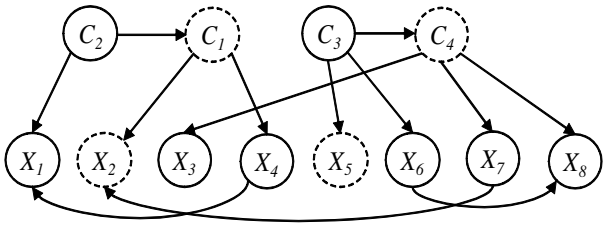

Figure 3: Example of an MBC structure including structural changes in comparison with the initial MBC structure in Figure 2. Nodes C_1, C_4, X_2, and X_5, represented in dashed line, are characterized as changed nodes

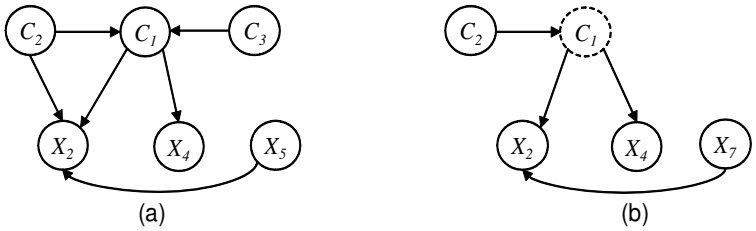

Figure 4: Markov blanket of node C_1 (a) before and (b) after change

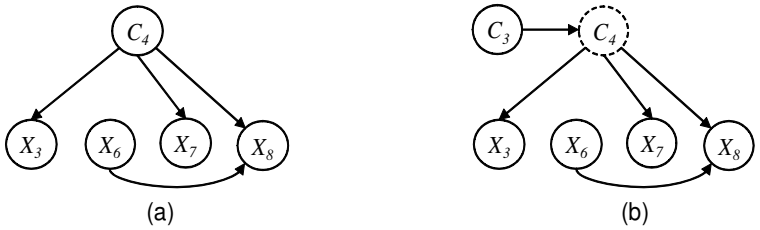

Figure 5: Markov blanket of node C_4 (a) before and (b) after change

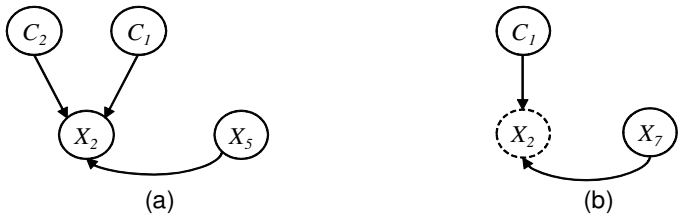

Figure 6: Parents-children set of node X_2 (a) before and (b) after change

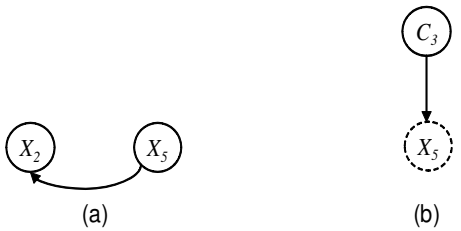

Figure 7: Parents-children set of node X_5 (a) before and (b) after change

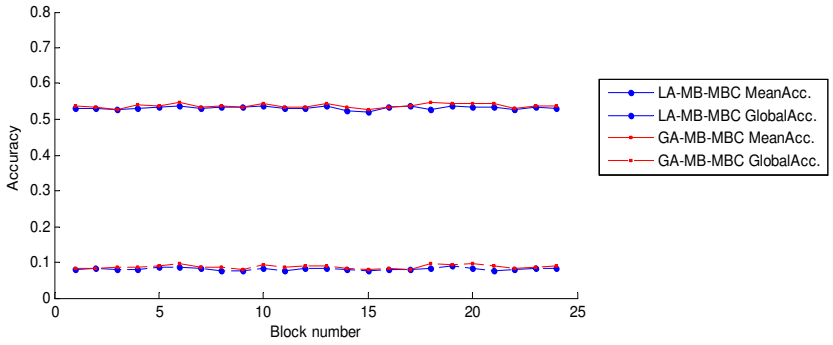

Figure 8: Classification results with the drift rate $p = 0\%$ and $\lambda = 1$

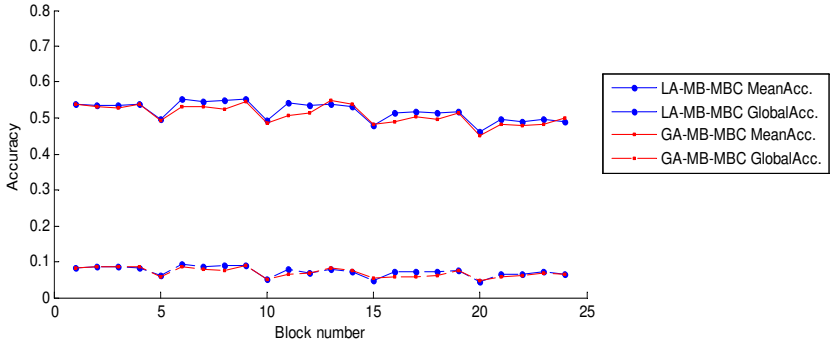

Figure 9: Classification results with the drift rate $p = 20\%$ and $\lambda = 1$

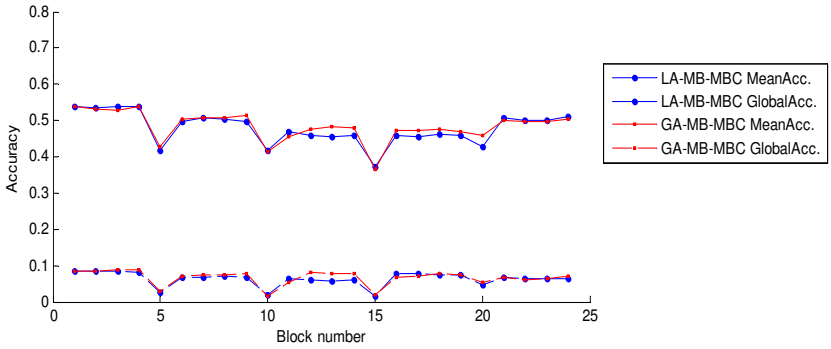

Figure 10: Classification results with the drift rate $p = 50\%$ and $\lambda = 1$