Learning Multi-Class Discriminative Patterns using Episode-Trees

Eng-Jon Ong*, Nicolas Pugeault†, Andrew Gilbert* and Richard Bowden*
*Centre for Vision, Speech and Signal Processing
University of Surrey, UK
Email: e.ong,a.gilbert,r.bowden@surrey.ac.uk
† College of Engineering, Mathematics and Physical Sciences,
University of Exeter, UK
Email: n.pugeault@exeter.ac.uk

***Abstract*—In this paper, we aim to tackle the problem of recognising temporal sequences in the context of a multi-class problem. In the past, the representation of sequential patterns was used for modelling discriminative temporal patterns for different classes. Here, we have improved on this by using the more general representation of *episodes*, of which sequential patterns are a special case. We then propose a novel tree structure called a MultI-Class Episode Tree (MICE-Tree) that allows one to simultaneously model a set of different episodes in an efficient manner whilst providing labels for them. A set of MICE-Trees are then combined together into a MICE-Forest that is learnt in a Boosting framework. The result is a strong classifier that utilises episodes for performing classification of temporal sequences. We also provide experimental evidence showing that the MICE-Trees allow for a more compact and efficient model compared to sequential patterns. Additionally, we demonstrate the accuracy and robustness of the proposed method in the presence of different levels of noise and class labels.**

***Keywords*–Data Mining, Classification, Episodes Patterns, Decision Trees**

Manuscript presented at the IARIA PATTERNS 2016 conference, 20-24th of March in Rome. (c) IARIA 2016.

I. Introduction

There are many problems in machine learning where the data consists of temporal sequence of events. In this work, we consider the problem where we have a collection of data streams that we want to label according to a large number of classes. Although there has been significant work concerned with mining frequently occurring patterns in data streams, few studies have focused on the different task of classifying such data streams. When solving this problem, we face several challenges: 1) a large number of classes may lead to bloated models, and large class confusion; 2) learning *discriminative* sequences rather than *frequently* occurring ones; 3) relevant sequences that contain discriminative power may be sparse in the data stream; and 4) the presence of ambiguities in the ordering of some parts of a discriminative sequence. In order to address these problems, this article presents a theoretical framework for learning of discriminative temporal sequences based on episodes [1] that are structured efficiently within a tree. We show how multiple episode-trees can be combined in a Boosted framework to yield an accurate and robust classifier.

There is a significant amount of prior research that investigate the discovery of *frequently occurring* temporal sequences, represented using sequential patterns. There are two main approaches for mining frequent sequential patterns: Apriori-based methods [2], [3] and pattern growth methods [4], [5]. An alternative to sequential patterns is the representation of so-called "episodes". Episodes are a more generic representation of temporal patterns proposed by Mannila et al.[1], [6] that allow the formalisation of ambiguity in the sequencing of some events in the pattern (whereas sequential patterns represent strict ordering). They also proposed algorithms for finding frequent episodes in data streams, in the limited case of either *serial* or *parallel* episodes. More recently, two independent groups have proposed algorithms for mining frequent episodes in the general case [7], [8], [9].

The present work addresses a different problem: our aim is the classification of data streams, and therefore the sequence of events are learnt for maximizing discrimination between the classes. Hence, a pattern may be discriminative for a single class amongst many without being frequent over the whole dataset. An early attempt at learning discriminative sequential patterns was proposed by Nowozin [10] and showed promising results for action recognition, yet the proposed approach is limited to binary discrimination. For problems containing more than two classes, this entails learning a collection of 1 versus 1 classifiers within a voting framework—this approach is clearly not scalable to problems with large number of classes. Recently, Ong et al. [11], [12] proposed a scalable approach to multi-class sequential pattern classification that makes use of boosted Sequential Pattern trees (SP-trees) for learning discriminative sequential patterns. One essential property of this approach is that the SP-trees allow for feature sharing between sequential patterns that describe different classes. To our knowledge these articles are the only attempts at learning discriminative sequential patterns.

This article extends SP-trees to the more general formalism provided by episodes. We will show that for certain specific patterns, episode trees will allow for a more compact encoding of discriminative temporal patterns. This article has four main contributions: first, we present a theoretic definition of a tree structure called MultI-Class Episode Trees (MICE-Trees) allowing for multiple classes to share sub-episodes; second, we propose an efficient algorithm for learning Boosted collections of such multi-class trees; third we demonstrate that MICE-Trees are similar to SP-Trees, but allow for a better and more compact representation of certain type of temporal sequences; and fourth, we show that the resulting classifiers can cope

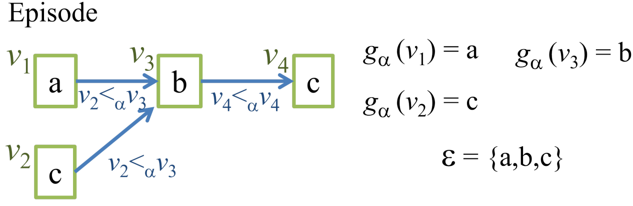

Figure 1. Illustration of a generic episode α, of the form $(a, c) \to (b) \to (c)$.

with a large amount of signal noise while still providing good classification accuracy.

The rest of the paper is organised as follows: section 1 will provide the theoretical framework of episodes and some important results; section 2 will present the MICE-Trees and provide proofs that paths in such a tree are effectively episodes; section 3 will describe an efficient algorithm for learning such MICE-Trees from a dataset of labelled streams; section 4 will present an approach for learning a Boosted forests of MICE-Trees; section 5 will present two synthetic dataset that are used to evaluate the performance and robustness of the proposed approach; and finally we will conclude in section 6.

II. Problem Statement

Given a vocabulary of events $\mathcal{E} = \{E_1, \ldots, E_{|\mathcal{E}|}\}$, we define a *data stream* $S \in \mathcal{S}_\mathcal{E}$ (where $\mathcal{S}_\mathcal{E}$ is the set of all possible data streams for a vocabulary $\mathcal{E}$), as a sequence of pairs

$$S = \left\langle (E_1, t_1), \ldots, (E_{|S|}, t_{|S|}) \right\rangle, \tag{1}$$

where (E_i, t_i) denotes the occurence of the transient event E_i at time t_i—where the time labels t_i are such that $t_i \leq t_j \; \forall i < j$.

Given a collection of data streams $\mathcal{D} = \{(S_i, \lambda_i)\}_{i \in [1, |\mathcal{D}|}$, with associated class labels $\lambda_i \in \mathcal{L}$, the task of stream classification can be seen as the function

$$F : \mathcal{S}_\mathcal{E} \to \mathcal{L}, \tag{2}$$

that associates labels λ_i to data streams S_i. Specifically, this article proposes an approach for learning F from a set $\mathcal{D}$ of labelled streams using *discriminative episodes*.

A. Episodes

Following from Mannila et al. [6], we define an *episode* as

Definition 1. An episode is defined as $\alpha = (V_\alpha, <_\alpha, g_\alpha)$, where $V_\alpha = \left\{v_1, \ldots, v_{|v|}\right\}$ is a collection of vertices, $<_\alpha \subseteq V_\alpha \times V_\alpha$ is a strict partial order, and $g_. V_\alpha \to \mathcal{E}$ is a mapping between episode vertices and observed events.

where

Definition 2. The relation $<_\alpha$ is a *strict partial order*, iff. $\forall (a, b) \in <_\alpha$, we have:

$$a \neq b \tag{3}$$
$$(b, a) \notin <_\alpha \tag{4}$$
$$(b, c) \in <_\alpha \Rightarrow (a, c) \in <_\alpha \tag{5}$$

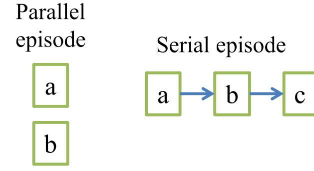

Figure 2. Illustration of a parallel episode (left) of the form (a, b) and of a serial episode (right), of the form $(a) \to (b) \to (c)$.

This is illustrated in Fig. 1. In this figure, each green box corresponds to an episode vertex v_i, and the letter in the box corresponds to the mapped event $E_j = g_\alpha(v_i)$, with an alphabet $\mathcal{E} = \{a, b, c\}$. The blue arrows joining the boxes represent the serial ordering between vertices enforced by the strict partial order $<_\alpha$.

Moreover, we define sequential patterns as special cases, coined *serial episodes*(see Fig. 2, right):

Theorem 1. *If $<_\alpha^+$ is a strict total order, then $\alpha = (V_\alpha, <_\alpha^+, g_\alpha)$ is a serial episode (i.e., a sequential pattern).*

Proof: If $<_\alpha^+$ is a strict total order, then $\forall a, b \in V_\alpha, a \neq b$, we have

$$(a, b) \in <_\alpha^+ \;\; \textbf{or} \;\; (b, a) \in <_\alpha^+,$$

hence there exists a sequence $(\beta_1, \ldots, \beta_{|V_\alpha|})$ such that

$$\forall i, j \in [1..|V_\alpha|], i < j \Rightarrow (v_{\beta_i}, v_{\beta_j}) \in <_\alpha \tag{6}$$

and therefore $(g_\alpha(v_{\beta_i}))_{i=1}^{|V_\alpha|}$ is a sequential pattern. ■

and conversely we define *parallel episodes* (illustrated in the left panel of Fig. 2):

Definition 3. If $<_\alpha = \emptyset$, then α is a *parallel episode*.

Finally we define a relation $\alpha \sqsubseteq S$ which states that an episode α occurs within a stream $S \in \mathcal{S}$:

Definition 4. Let α be an episode and $S \in \mathcal{S}$ be a stream, we define that $\alpha \sqsubseteq S$, iff. there exists a sequence $\beta_1, \ldots, \beta_{|V_\alpha|}$ such that $\forall i, j \in [1..|V_\alpha|], i < j \Rightarrow (v_{\beta_i}, v_{\beta_j}) \in <_\alpha$, and that $\exists \{t_i\}_{i=1}^{|V_\alpha|}, \left((g_\alpha(v_{\beta_i}), t_i)_{i=1}^{|V_\alpha|} \right) \subset S$.

This is illustrated in Fig. 3, where the red arrows indicate the occurrences of the episode α's vertices v_i, mapped to events $a, b, c \in \mathcal{E}$ in the stream S. Note that the sequence of the occurrence of events a and b for the episode vertices v_1 and v_2 do not matter (they form a parallel episode).

B. MultIClass Episode Trees (MICE-Trees)

This section presents a definition of the MICE-Trees and how paths to a MICE-Tree's leaves model different episodes α_k.

First, we define a MICE-Tree as $T = (N, L)$, where $N = \{n_i\}_{i=1}^{|N|}$ is the set of all tree nodes and $L = \{l_i\}_{i=1}^{|L|}$ the set of all links, such that $L \subseteq N \times N$—such a tree is illustrated in Fig. 4. In the following section we will define the nodes and links of the MICE-Trees and their specificity for the purpose of learning and encoding episodes.

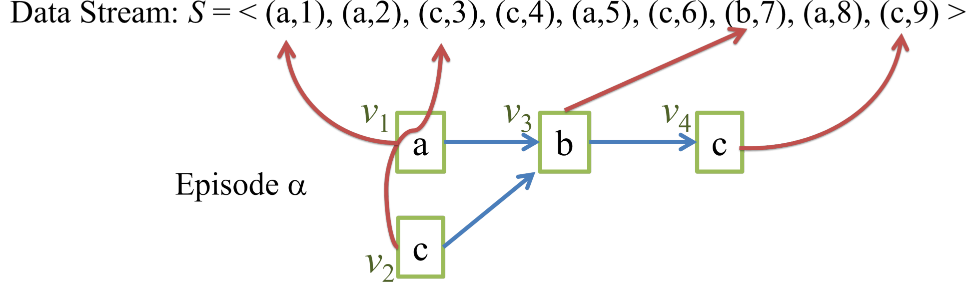

Figure 3. Illustration of how an episode α is matched in a data stream S, such as $\alpha \sqsubseteq S$.

1) Tree nodes $n \in N$: MICE-Tree nodes have the double purpose of storing episode information and the most likely class label given the data. Formally:

Definition 5. A MICE-Tree node $n \in N$ is defined as the tuple $n = (V_n, g_n, \lambda_n)$, where $\lambda_n \in \mathcal{L}$ is the most likely label at this node, V_n is a set of vertices and g_n is a mapping such that $\forall v_i \in V_n,\ \exists E_i \in \mathcal{E}$ such that $E_i = g_n(v_i)$ and $\{E_i\}_{i=1}^{|V_n|}$ is a set of (unordered) events.

Without loss of generality, we enforce that the set of vertices in all tree nodes be strictly disjoint:

Definition 6. For any two nodes $n, n' \in N$, if $n \neq n'$ then $V_*(n) \cap V_*(n') = \emptyset$

Moreover, it follows from the definition that:

Property 1. *A tree node $n \in N$ models a (trivial) parallel episode $\alpha = (V_*(n), \emptyset, g_*(n))$.*

For convenience we define the following accessor functions for the properties of a node $n = (V_n, g_n, \lambda_n)$, namely: $V_*(n) = V_n$, $g_*(n) = g_n$ and $\lambda_*(n) = \lambda_n$.

2) Tree links $l \in L$: Now that we have defined the MICE-Tree nodes as encoding the parallel components of episodes, we will define how the tree links encode the serial constraints in the episode, and whether these are satisfied or not by data streams. Formally, we define a tree link as follows:

Definition 7. A MICE-Tree link $l \in L$ is defined as the tuple $l = (a, b, s)$ where $a, b \in N$ are nodes in the tree and $s \in \{+, -\}$ is the type of edge (positive or negative).

For convenience, we define the root of the tree as

Definition 8. We call *root node* of a tree $T = (N, L)$ the unique node $n^0 \in N$ that satisfies the condition $\nexists l \in L, n \in N, s \in \{+, -\}$ such that $l = (n, n^0, s)$.

and its leaves:

Definition 9. We say that a node $n^* \in N$ is a *leaf node* if $\nexists l \in L, n \in N, s \in \{+, -\}$ such that $l = (n^*, n, s)$.

Without loss of generality, we will define that leaf nodes are the only nodes in the tree containing no vertices, hence we have:

Property 2. *For any node $n \in N$, we have $V_*(n) = \emptyset$ iff. n is a leaf node.*

This property is ensured by the learning mechanism described in section 3.

In this work, we restrict ourselves to using binary trees, whereby every non-leaf tree-node ($n \in N$) will have exactly two child tree-nodes $n^+, n^- \in N$, where $(n, n^+, +) \in L$ and $(n, n^-, -) \in L$. Also, as for the nodes we define for links $l = (p_l, c_l, s_l)$ the accessor function $s_*(l) = s_l$.

Property 1 showed that MICE-Tree nodes model parallel episodes; conversely, we will now show that links can be interpreted as the *serial* components of episodes. As a first step, we need to show that two nodes connected by a link form a strict partial order (see definition 2).

Definition 10. We define a function φ between two sets of vertices V_a and V_b, such that $\varphi(V_a, V_b) = \{(x, y) : x \in V_a, y \in V_b\}$.

Theorem 2. *If $V_a \cap V_b = \emptyset$ then $\varphi(V_a, V_b)$ is a strict partial order.*

Proof: Let $(a, b) \in \varphi(V_a, V_b)$, then from definition 10, we have $a \in V_a$ and $b \in V_b$, and therefore $V_a \cap V_b = \emptyset$ implies: (1) $a \neq b$; (2) $b \notin V_a$ and therefore $\forall c \in V_b, (b, c) \notin \varphi(V_a, V_b)$ ■

From definitions 6, 7, 10 and theorem 2, we derive the following property for linked nodes:

Corollary 1. *A tree node $n \in N$ models a parallel episode $\alpha = (V_*(n), \emptyset, g_*(n))$, and a pair of nodes n and n' connected by a link l models the general episode α', such as:*

$$\alpha' = (V_*(n) \cup V_*(n'), \varphi(V_*(n), V_*(n')), g_*(n) \cup g_*(n')). \quad (7)$$

3) Tree Paths $P \subset T$: In the previous section we have presented the MICE-Tree nodes and edges, we will now define paths in the tree, and show that any path in a MICE-Tree encodes an episode.

First we will define a tree path:

Definition 11. A path P is a sequence of node-edge pairs $P = ((n_1, l_1), \ldots, (n_K, l_K))$, where K is the length of the path, and all the nodes N_i in the path are connected by their edges. Formally, $l_i = (n_i, n_{i+1}, s_i)$, $\forall i \in [1, K-1]$, and $l_K = (n_K, n^*, s_K)$, where n^* is a leaf node.

and leaf nodes:

Definition 12. We say that a node $n^* = (\emptyset, \emptyset, \lambda) \in N$ is a leaf node if $\nexists l \in L, n \in N, s \in \{+, -\}$ such that $l = (n^*, n, s)$.

Then we define the function ξ that generates an episode from any MICE-Tree path:

Definition 13. Let $P = ((n_1, l_1), \ldots, (n_{|P|}, l_{|P|}))$ be a path in tree T, and let $\psi(P) \subset P$ be the indices of positive links in the path: $\psi(P) = \{i : i \in [1, |P|], s_*(l_i) = +\}$. Then we define the function $\xi(P) = (V_P, <_P, g_P)$, such that

$$\begin{cases} V_P &= \bigcup_{i\in\psi(P)} V_*(n_i) \\ <_P &= \bigcup_{i\in\psi(P)} \bigcup_{j\in\{k\in S(P):k>i\}} \varphi(V_*(n_i), V_*(n_j)) \\ g_P &= \bigcup_{i\in\psi(P)} g_*(n_i) \end{cases} . \tag{8}$$

Lemma 1. *If P is a path in a MICE-Tree g_P is a map such that $g_P : V_P \to \mathcal{E}$.*

Proof: For all nodes n in the tree, we have $g_*(n)$ a map such that $g_*(n) : V_*(n) \to \mathcal{E}$, $\forall n \in N$ (definition 5) and $V_*(n_i) \cap V_*(n_j) = \emptyset$ $\forall n_i, n_j \in N, n_i \neq n_j$ (definition 6), therefore $\bigcup_{i\in\psi(P)} V_*(n_i)$ is a map. ■

Lemma 2. *If P is a path in a MICE-Tree $<_P$ is a strict partial order.*

Proof: From definition 10, $\varphi(V_*(n_i), V_*(n_j))$ is a strict partial order for any pair of nodes $n_i, n_j \in N, i \neq j$ (i), and definition 6 enforces that the sets of vertices of all tree nodes are disjoint $V_*(n_i) \cap V_*(n_j) = \emptyset$ if $n_i \neq n_j, \forall n_i, n_j \in N$ (ii), therefore we have:

From definition 2, $<_P$ is a strict partial order iff:

(1) $(a, b) \in <_P$ implies that $a \in V_*(n_i)$ and $b \in V_*(n_j)$ with $i \neq j$, and therefore because $V_*(n_i) \cap V_*(n_j) = \emptyset$ from (ii), we have $a \neq b$

(2) According to definition 13, $(a, b) \in <_P$ implies that $a \in V_*(n_i)$ and $b \in V_*(n_j)$ with $i < j$, and therefore because $V_*(n_i) \cap V_*(n_j) = \emptyset$ we have $(b, a) \notin <_P$.

(3) let $(a, b) \in <_P$, and $(b, c) \in <_P$, from definition 7 we have $a \in V_*(n_i)$, $b \in V_*(n_j)$ and $c \in V_*(n_k)$ such that $i, j, k \in \psi(P)$, and from definition 13, $i < j < k$, and therefore $(a, c) \in <_P$. ■

Therefore, from lemmas 1 and 2 we can draw the following theorem:

Theorem 3. *If P is a path in a MICE-Tree T, then $\xi(P) = (V_P, <_P, g_P)$ is an episode.*

Proof: According to definition 1, the triplet $(V_P, <_P, g_P)$ defines an episode iff. (1) $g_P : V_P \to \mathcal{E}$, proved by lemma 1; and (2) $<_P$ is a strict partial order, proved by lemma 2. ■

An example of a MICE-Tree is then illustrated in Fig. 4. In this figure, each green square denote a non-leaf node $n \in N$, that code a parallel episode—the letters in the box indicate the events $E_i \in \{g_*(n)(v) : v \in V_*(n)\}$. The blue boxes denote leaf nodes n^* associated to a valid label $\lambda_*(n^*) \in \{c1, c2, c3\}$, and below are indicated the episodes that correspond to each leaf node's path. Finally, the orange boxes denote rejection leaf nodes where no assertion can be made on the episode's class and the blue arrows denote tree links $l \in L$.

C. Classifying Datastreams using MICE-Trees

In order to classify an input sequence S given a MICE Tree $T = (N, L)$, we define the following recursive function

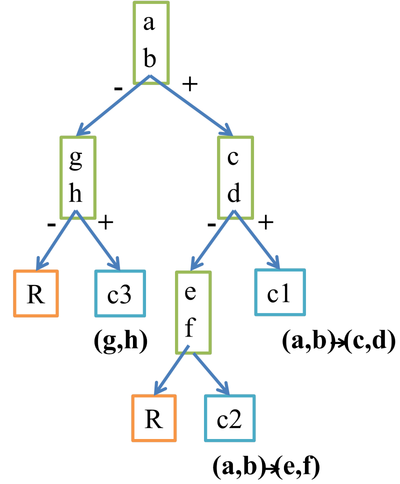

Figure 4. Example of a MICE-Tree.

$C_T : N \times S_\varepsilon \to \mathcal{L}$:

$$C_T(n, S) = \begin{cases} C_T(n^+, S) & \text{if} \quad \xi(P_n) \sqsubseteq S, V_n \neq \emptyset \\ C_T(n^-, S) & \text{if} \quad \xi(P_n) \not\sqsubseteq S, V_n \neq \emptyset \\ \lambda_n & \text{otherwise} \end{cases} \tag{9}$$

where the tree node n is the triplet, $n = (V_n, g_n, \lambda_n)$, and n^+, n^- are the positive and negative child-nodes of n respectively: $(n, n^+, +), (n, n^-, -) \in L$. It is now possible to define a multi-class classification function for labelling an input sequences given a MICE-Tree with root node r:

$$h_T(S) = C_T(r, S) \tag{10}$$

This is achieved in an computationally efficient manner using Algorithm 1.

Algorithm 1 MICE-Tree Classifier: **P** = $C_T(S)$

Input: Input datastream $\mathbf{S} =< (S_i, t_i) >_{i=1}^{|S|}$
Input: MICE-Tree $T = (N, L)$, root node r.
Output: Label of $S : \lambda \in \mathcal{L}$
Initialise current node to root node: $n^{cur} = r$
The contents of current node: $n^{cur} = (V_{cur}, g_{cur}, \lambda_{cur})$
Init start offset: $e = 1$
while n^{cur} is not leaf-node **do**
  For each $k \in [1, |V_{cur}|]$ get:
    $G_k = \{j : j \in [e, |S|], S_j = g_{cur}(v_k)\}$
  $Z = \bigcap_{k\in[1,|V_{cur}|]} [min(G_k), |S|]$
  if $Z = \emptyset$ **then**
    $n^{cur} = m$, such that $(n^{cur}, m, -1) \in L$
  else
    $n^{cur} = m$, such that $(n^{cur}, m, +1) \in L$
    $e = \min_{k\in[1,|V_{cur}|]}(\min(G_k))$
  end if
end while
Return λ_{cur}

III. Learning MICE Trees

In this section, we propose a novel method for constructing a MICE-Tree given a database of weighted and labelled data streams. Here, the construction of a MICE-Tree is achieved

in a greedy and recursive manner, whereby the multi-class training dataset is recursively partitioned into smaller and smaller subsets that are distributed across different nodes of the MICE-Tree. To achieve this, we theoretically show that a node in the MICE-Tree does indeed induce a binary partition on the training dataset in Section III-A. However, it is also important that there is a method for measuring the optimality of such a binary partition. To this end, a node-split criteria based on the Gini impurity measure is described in Section III-B. Finally, the MICE-Tree learning algorithm is described in Section III-C

A. MICE-Tree Node Binary Partition

For the purpose of learning, we are provided with a training data stream collection that we will denote as $\mathcal{D}$ (defined in Section II). Additionally, we are given a set of weights $W = (w_i)_{i=1}^{|\mathcal{D}|}$, where each example $(S_i, \lambda_i) \in \mathcal{D}$ is associated with the weight w_i.

In order to construct a MICE-Tree T, we firstly introduce a useful function $D : S_\varepsilon \to 2^N$ based on Eq. 9 that extracts the set of nodes a datastream example X passes through when it is classified by T (using Eq 9):

$$D_T(n, S) = n \cup \begin{cases} D_T(n^+, S) & \text{if} \quad \xi(P_n) \sqsubseteq S, V_n \neq \emptyset \\ D_T(n^-, S) & \text{if} \quad \xi(P_n) \not\sqsubseteq S, V_n \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \tag{11}$$

Both functions in D_T (Eq. 11 and C_T (Eq. 9) provides a deterministic mechanism for classifying an input datastream:

Remark 1. Let S be an input datastream, $T = (N, L)$ be a MICE-Tree and r its root node, then there exists only one unique path, whose nodes are $D_T(r, S)$, that will be used to classify it. That is, a sequence S will always go through the same path when C is used to classify it.

It can be observed that the properties of Eq. 11 will allow us to see that each node in a MICE-tree "captures" a subset of training datastreams from $\mathcal{D}$: Since each example training datastream can only "carve" a single unique path through a tree during classification (Rem. 1), then, only a subset of example datastreams will pass through a particular node in a tree. More specifically:

Definition 14. Let $T = (N, L)$ be a MICE-Tree, $\mathcal{D}$ be the labelled collection of datastreams. Let $n \in N$ be a node and $m \in N$ be its parent node, with P_m being the path from the root node to m. We then define the set of *captured example indices* of n as the following set: $\mathcal{I}_n = \{i \in [1, |\mathcal{D}|] : \xi(P_m) \sqsubseteq S_i, n \in D(r, S_i)\}$.

A consequence of the above definition is that a non-leaf node n will induce a binary partitioning of its captured example set $\mathcal{I}_n$ into a *positive partition*, $\mathcal{I}_n^+$, and *negative partition*, $\mathcal{I}_n^-$ where $\mathcal{I} = \mathcal{I}_n^+ \cup \mathcal{I}_n^-$ and:

$$\mathcal{I}_n^+ = \{i \in I : \xi(P_{n^+}) \sqsubseteq S_i\} \tag{12}$$

$$\mathcal{I}_n^- = \{i \in I : \xi(P_{n^+}) \not\sqsubseteq S_i\} \tag{13}$$

The partitioning induced by a node n is clearly dependent on its contents (i.e. V_n and g_n). Thus, for learning purposes, we define the following operation on a node:

Definition 15. Given a node $n = (V_n, g_n, \lambda_n)$ in a MICE-Tree, a new (episode vertex,event) pair (v', e') can be *added* into n, denoted as $n' = n + (v', e')$, where $e' \in \varepsilon$ and a v' is an episode-vertex. The new node is: $n' = (V_{n'}, g_{n'}, \lambda_{n'})$ with $V_{n'} = V_n \cup \{v'\}$, $g_{n'} = g_n \cup \{(v', e')\}$ and $\lambda_{n'} = \lambda_n$),

Importantly, adding (episode-vertex, event) pairs into any node has the effect of moving examples from its positive partition to the negative partition.

Theorem 4. *Let $n = (V_n, g_n, \lambda_n)$ be a node in a MICE-Tree, $\mathcal{I}_n$ the index set of examples captured by n and $\mathcal{I}_n^+$, $\mathcal{I}_n^-$, its induced partition respectively. Then, adding an (episode-vertex,event) pair, (v', e') to n will produce a new node, $n' = n + (v', e')$, with a potentially smaller positive partition, $\mathcal{I}_{n'}^+ \subset \mathcal{I}_n^+$ and potentially larger negative partition, $\mathcal{I}_{n'}^- \supset \mathcal{I}_n^-$.*

Proof: It is clear that the episode modelled by the path from the root node r to n is a subsequence that from r to n': $\xi(P_n) \sqsubseteq \xi(P_{n'})$. Then, by the Apriori rule, we have $\{i \in \mathcal{I}_n : \xi(P_{n'})\} \subset \{i \in \mathcal{I}_n : \xi(P_n)\}$ ■

Algorithm 2 GetBestSplit Algorithm

Input: Train Set: $\mathcal{S} = \{(\mathbf{S}_i, \lambda_i, w_i)\}_{i=1}^{|S|}$
Output: MICE-Tree Node, +ve partition, -ve partition, error
Let $\lambda_{\mathcal{S}}$ be the class with highest frequency in $\mathcal{S}$.
Initialise empty node: $n = (\emptyset, \emptyset, \lambda_{\mathcal{S}})$.
Initialise the event set: $E = \mathcal{E}$.
Initialise error: $\gamma_{best} = -1$
while $E \neq \emptyset$ **do**
  Let v' be an episode-vertex not present in any tree-node.
  Find $e_{best} \in E$ s.t. $n + (v', e_{best})$ minimises γ from
    Eq. 14 given weights w_i.
  if $\gamma_{best} < 0$ or $\gamma_{best} > \gamma$ **then**
    $n = n + (v', e_{best})$
    $E = E - \{e_{best}\}$ {Remove e_{best} from the set E}
  else
    break
  end if
end while
Let the path from the root node to n be P_n.
$\mathcal{S}_n^+ = \{(S, \lambda, w) \in \mathcal{S} : \xi(P_n) \sqsubseteq S\}$
$\mathcal{S}_n^- = \{(S, \lambda, w) \in \mathcal{S} : \xi(P_n) \not\sqsubseteq S\}$
Return $(n, \mathcal{S}_n^+, \mathcal{S}_n^-, \gamma_{best})$.

B. Node-Split Criteria

The final tool required for learning tree is a method of evaluating how "good" a node split is. This is achieved using an adapted Gini impurity that is popular in decision tree learning [13]. Here, we have changed the criteria to account for weighted training examples. Suppose we have found that non-leaf node n has caused a partition $\mathcal{I}^+$ and $\mathcal{I}^-$. Suppose the corresponding weights of these partitions are W'_+ and W'_- respectively. Similarly, let the corresponding labels be Y'_+ and Y'_- respectively. We define the total positive and negative partition weight coefficient as: $Z^+ = \sum_{i=1}^{|W'_+|} w'_{+,i} / \sum_{i=1}^{|W'|} w'_i$ and $Z^- = \sum_{i=1}^{|W'_-|} w'_{-,i} / \sum_{i=1}^{|W'|} w'_i$ respectively. Using both the weights and labels, it is possible to compute a normalised label histogram for each partition: F'_+ and F'_- respectively.

Algorithm 3 MICE-Tree Learn Algorithm

Input: Training Set: $\{(\mathbf{S}_i, \lambda_i)\}_{i=1}^{|S|}$, $(w_i)_{i=1}^{|S|}$
Output: MICE-Tree $T = (N, L)$
Make Offseted Train Set: $\mathcal{S} = \{(\mathbf{S}_i, \lambda_i, w_i, o_i)\}_{i=1}^{|S|}$, $o_i = 1$
Queue element: $(Parent, TrainSubset, Depth, LinkType)$
Get optimal root node:
$(r, \mathcal{S}^+, \mathcal{S}^-, \epsilon) =$ GetBestSplit($\mathcal{S}$)
Initialise the queue:
$Q = \{(r, \mathcal{S}^+, 2, +), (R, \mathcal{S}^-, 2, -)\}$.
while $Q \neq \emptyset$ **do**
Remove last item of Q: $(n^{cur}, \mathcal{S}^{cur}, D^{cur}, s^{cur})$
if $|\mathcal{S}^{cur}| \leq \alpha$ OR $D^{cur} \geq \beta$ **then**
Get class (λ) with highest weighted freq. in $\mathcal{S}^{cur}$
$\lambda = -1$ if $\mathcal{S}^{cur} = \emptyset$.
$m = (\emptyset, \emptyset, \lambda)$
$N = N \cup \{m\}$ {Add new tree-node}
$L = L \cup \{(n^{cur}, m, s^{cur})\}$ {Link to new tree-node}
break
end if
Get optimal current node:
$(m, \mathcal{S}^+, \mathcal{S}^-, \epsilon) =$ GetBestSplit($\mathcal{S}^{cur}$)
$N = N \cup \{m\}$ {Add new tree-node}
$L = L \cup \{(n^{cur}, m, s^{cur})\}$ {Link to new tree-node}
if $\epsilon > 0$ **then**
$Q = Q \cup \{(m, \mathcal{S}^+, D^{cur} + 1, +), (m, \mathcal{S}^-, D^{cur} + 1), -\}$.
end if
end while
Return MICE-Tree: $T = (N, E)$

The node-split criteria is defined as:

$$\gamma = Z^+(1 - \sum_{i=1}^{C} f_{+,i}^2) + Z^-(1 - \sum_{i=1}^{C} f_{-,i}^2) \qquad (14)$$

C. MICE-Tree Learning Algorithm

The algorithm for learning the MICE-Tree is given in Algo. 3, where a MICE-Tree is constructed in a greedy and recursive manner. One key mechanism for constructing MICE-Trees is given in Algo. 2. Here, episode-vertices are sequentially added in a greedy fashion to the parent nodes of leaf nodes to maximally improve the splitting criteria (Eq. 14). This process terminates when the splitting criteria cannot be improved.

Algo. 3 then starts by constructing the root node using Algo 2. This induces a partitioning of the dataset into two training subsets (i.e. +ve and -ve partitions). Two new children tree nodes (+ve and -ve) are constructed and linked to the root node. Each training subset is then passed on to its respective child node. The algorithm then recursively applies Algo 2 to configure the contents of both child nodes.This recursive process is performed via a queue-based system until one of 3 termination criteria is reached: 1) maximum tree-depth β is reached; 2) training subset is smaller than minimum size α (set here as 1); 3) The training subset is "pure" (i.e. only belongs to a single class).

IV. BOOSTING MICE FORESTS

In this section, we describe the method for learning and combining the multiple MICE-trees in order to produce a robust and accurate classifier that generalises to unseen novel sequences in the presence of noise. To this end, we propose a novel machine learning method for learning strong classifiers based on MICE-Tree within the Multi-class AdaBoost framework [14]. A strong classifier outputs a class label based on the maximum votes cast by a number (S) of selected and weighted weak classifiers:

$$H(I) = \arg\max_{c} \sum_{i=1}^{S} \alpha_i \mathbb{I}(h_i(I) = c) \qquad (15)$$

In this paper, the weak classifiers h_i are the MICE-Tree classifiers defined in Section II-C as Eq. 10. Each weak classifier h_i is selected iteratively with respect to the following error:

$$\epsilon_i = \sum_{i=1}^{X} \mathbb{I}(h_i(X_i) \neq y_i) \qquad (16)$$

Typically, in order to determine the optimal weak classifier at each Boosting iteration, the common approach is to exhaustively consider the entire set of possible weak classifiers and finally select the best weak classifier (i.e. that with the lowest ϵ_i). Such an exhaustive search will not be possible due to the large search space of possible MICE-Trees. Thus, Algo. 3 is used for choosing the appropriate MICE-Tree weak classifier instead given a set of training datastreams with associated boosted weights.

The final MICE-Tree Boosting algorithm is detailed in Algo. 4. We have chosen to iteratively learn new MICE-Tree based on the multi-class AdaBoost method. However, we are not limited to this particular form of Boosting and it would be easy to integrate the MICE-Tree learning algorithm (Algo. 3) into other Boosting methods (e.g. GentleBoost, etc...).

Algorithm 4 MICE-Tree-Boost Algorithm

Initialise example weights: $\forall w_i \in W, w_i = 1/X$
for $t = 1, ..., M$ **do**
Select ($h_t = h^{\mathbf{T}_{best}}$) using Algo. 3
Obtain the classification error ϵ_t for h_t (Eq. 16)
Obtain the weight $\alpha_t = \ln \frac{1-\epsilon_{best}}{\epsilon_{best}} + \ln(C-1)$
Update weights: $w_i = w_i \exp(-\alpha_i [h_t(X_i) \neq y_i])$
Normalise weights: $\sum_{i=1}^{X} w_i = 1$
end for
Return the strong classifier:
$H(X) = \arg\max_c \sum_{i=1}^{M} \alpha_i \mathbb{I}(h_i(X) = c)$

V. EXPERIMENTAL EVALUATION

In this section we evaluate the proposed approach using two different sets of generated data streams, that allow us to control for properties of the encoded episodes. The first experiment, in section V-A was designed to illustrate the limitations of the more common Sequential pattern framework (as used by [10], [11]), and the advantages of the more generic episodes framework presented herein. The second experiment, in section V-B was designed to have a thorough evaluation of the MICE-Trees robustness to noise in the data streams.

A. Exp 1: Comparison with SP-trees

The first experiments evaluates the special case where classes cannot be discriminated by purely serial episodes, and therefore form a special challenge for SPs and should benefit from the greater generality of the episode model.

In order to assess this, we generated a specifically designed dataset where all serial episodes are ambiguous. We achieved this by generating 10 classes $A, \dots, J$, each characterised by a unique episode of the form: $\alpha^\lambda = (E_1^\lambda, E_2^\lambda) \rightarrow (E_3^\lambda, E_4^\lambda)$, such that there are no two class episodes that share any event. Then we generated exhaustively a collection of streams that satisfy each class λ, and injected noise in the resulting streams by sampling from all possible sequence of events present in other classes $\lambda' \neq \lambda$ that do not satisfy any class episode. For example, some streams of class A perturbated with events from class B would be:

$$\begin{array}{l} S_1 = \langle (E_1^A,1),(E_2^A,2),(E_1^B,3),(E_2^B,4),(E_4^B,5),(E_3^A,6),(E_4^A,7)\rangle \\ S_2 = \langle (E_2^A,1),(E_1^A,2),(E_1^B,3),(E_4^B,4),(E_3^B,5),(E_3^A,6),(E_4^A,7)\rangle \\ S_3 = \langle (E_1^A,1),(E_2^A,2),(E_2^B,3),(E_3^B,4),(E_4^B,5),(E_4^A,6),(E_3^A,7)\rangle \\ \vdots \end{array} \quad (17)$$

In this case, the E_i^B elements (shown in blue in Eq. 17), injected in the middle, will contain all *serial* episodes that define the episode α_B—hence only the *generic* episodes are unique to the classes we generated this way.

We then learnt a collection of 100 SP-Trees and 100 MICE-Trees from this data, using 500 training samples out of a collection of 2,880 streams and testing on the rest. The results, for different values of maximal tree depth are shown in Fig. 5. There, we can see that SP-Trees require significantly mode complex models, in terms of both number and depth of trees, to match the performance of MICE-Trees, confirming our assertion that the episode framework allow for a more efficient and compact encoding of certain classes of temporal patterns.

B. Exp 2: Robustness to noise

The aim of this second experiment was to assess the robustness of the MICE-Trees classification for a large number of classes (we used 100 classes) denoted by temporal patterns corrupted by increasing amounts of noise.

1) Episode generation: In order to generate the dataset we first defined a vocabulary $\mathcal{E}$ of 26 symbols. Then for each of the classes we generated a signature episode by the following steps: first, we generated a single sequence of events $(E_1, \dots, E_N) \in \mathcal{E}^N$, where $N = 18$ is the number of events in the sequence; second, in order to transform this purely serial episode in a generic one, we permuted randomly a number of contiguous pairs of symbols. Hence if the original sequence created was

$$(E_1) \rightarrow (E_2) \rightarrow (E_3) \rightarrow (E_4) \rightarrow (E_5)$$

then after permutation of the leftmost (E_1, E_2) and rightmost (E_4, E_5) adjacent pairs of events, we obtain the alternative sequence

$$(E_2) \rightarrow (E_1) \rightarrow (E_1) \rightarrow (E_4) \rightarrow (E_3)$$

and therefore this class is best described by the episode

$$(E_1, E_2) \rightarrow (E_3) \rightarrow (E_4, E_5).$$

We use the same procedure to randomly generate unique episodes for each of the 100 classes.

2) Episode corruption with temporal noise: All the patterns generated at this point are perfect occurences of the class' episode: there are no spurious events in the data stream. In order to be able to control the proportion of spurious events in the training and testing data streams, we propose modelled the pattern generation as a Markov process. The process starts at the beginning of the pattern and from there has a chance α to move onto the next event in the noise-free pattern. Conversely, the process has a $1 - \alpha$ chance to generate a noisy event E^ϵ. Hence, a purity parameter of $\alpha = 1$ will generate a noise-free sequence, and a purity parameter of $\alpha = 0$ would generate an infinite sequence of randomised events. Moreover, the length of the generated pattern increases with diminishing values of α. This Markov process is illustrated in Fig. 6 for a simple noise-free sequence.

Moreover, and in order to make the injected noise more structured, spurious event E_k^ϵ are not generated using a uniform distribution, but using a noise generator that keeps a memory of the previous noisy event generated, and produces the next one using an a priori, randomly generated, Markov transition matrix encoding inhomogeneous transition probabilities $p(E_k^\epsilon | E_{k-1}^\epsilon)$—this noise generation approach ensures that the injected noise will be more likely to feature structure and repeating patterns.

The usage of the α parameter and Markov chain allows the pattern to be heavily corrupted, Figure 7 shows the corruption of a pattern with noise for decreasing levels of α.

Note that the pattern length increases quickly with noise; for values of α lower than 0.5 the proportion of noise elements is greater that the original pattern.

3) Results: Figure 8 shows the performance of the MICE-Trees classification for 100 classes and values of α varying between 20% and 100%. The full red line shows the classification error and the dashed blue line shows the Signal to Noise Ratio (SNR) for this value of α. This graph shows that the MICE-Trees yield excellent classification results, providing near-perfect classification for values of α above 75%. For values of α below 50%, the classification performance drops sharply, illustrating the fact that diminishing SNR values make the learning extremely challenging.

Overall, this shows that the MICE-Trees can provide excellent classification performance event in presence of large amount of spurious events and can handle a large number of classes.

VI. Conclusions

In this work, we presented a theoretic definition of a tree structure called MultI-Class Episode Trees (MICE-trees) allowing for multiple classes to share sub-episodes whilst providing the ability to classify datastreams. We then proposed an efficient algorithm for learning Boosted collections of such multi-class episode trees. The performance of the proposed model was then evaluated using two sets of experiments. The first demonstrated that MICE-Trees allow for a better and more compact representation of certain type of temporal sequences when compared to sequential patterns. We also show that the resulting classifiers can cope with a large amount of signal noise while still providing good classification accuracy.

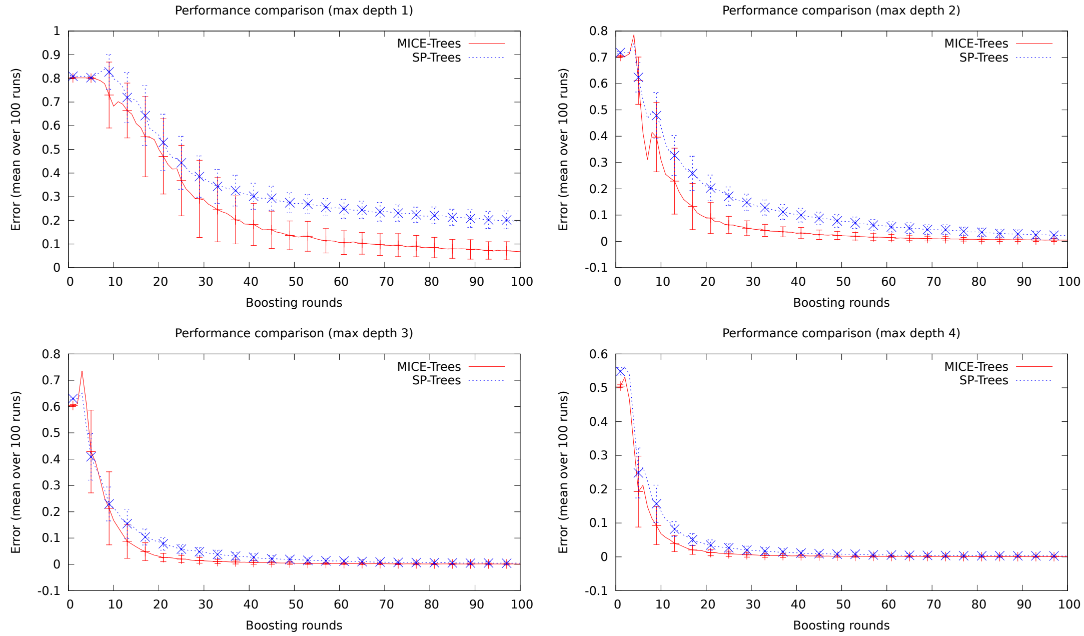

Figure 5. Performance of the classification for the dataset A (10 classes). The graphs show the mean classification error over 100 randomised tests, for rounds of Boosting from 1 to 100 and for maximal tree depths from 1 to 4. The error bars show the standard deviations for all curves.

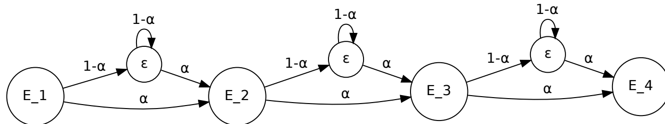

Figure 6. Illustration of the noise injection process in a noise-free sequence (E_1, E_2, E_3, E_4). The generation is processed as a Markov process where the chance to insert spurious events ε (randomly generated) is dependent on a purity parameter α.

α = 1.0 {A W W Q W V W M B Q E U X K X F B U}
α = 0.8 {A W W Q S W V W P M B Q E U X K X F B U S}
α = 0.5 {A U W A C W Q E P W V H I E W I S M B Q E D U L G A X K O X F I S B K A U E H P}
α = 0.3 {P X J A A W K I W Q V Q T W F J V B Q O F L W M M G B Q U U A H C G M K X H J E U U X K X F S B W M U}

Figure 7. A example of the pattern corruption for increasing α

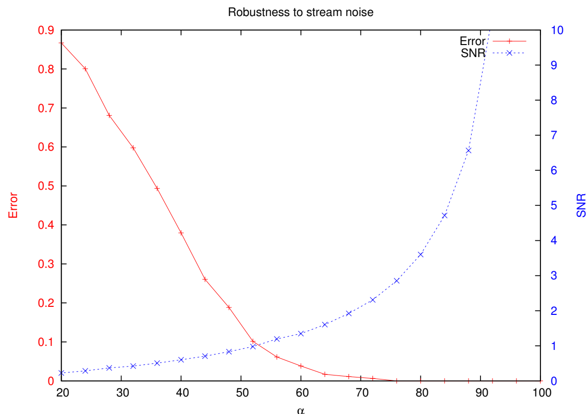

Figure 8. This figure illustrates the robustness of the MICE-Trees to noise injected in the data streams.

References

[1] H. Mannila, H. Toivonen, and A. Verkamo, "Discovering frequent episodes in sequences," in Proc. of Int. Conf. on Knowledge Discovery and Data Mining (KDD'95), 1995, pp. 210–215.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns: Generalizations and performance improvements," in Proc. of 5th International Conference on Extending Database Technology, 1996.

[3] M. Zaki, "Spade:an efficient algorithm for mining frequent sequences," Machine Learning, 2001.

[4] J. Han, J. Pei, and Y. Yin, "Mining frequent pattern without candidate generation," in Proc. of International Conference on Management of Data (SIGMOD), 2000.

[5] J. Han, J. Pei, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," in Proc of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2000.

[6] H. Mannila, H. Toivonen, and A. Verkamo, "Discovery of frequent episodes in event sequences," Data Mining and Knowledge Discovery, vol. 1, 1997, pp. 259–289.

[7] N. Tatti and B. Cule, "Mining closed strict episodes," in Proc. of the Int. Conf. on Data Mining (ICDM'2010), 2010.

[8] ——, "Mining closed strict episodes," Data Mining and Knowledge Discovery, vol. 25, 2012, pp. 34–66.

[9] A. Achar, S. Laxman, R. Viswanathan, and P. Sastry, "Discovering injective episodes with general partial orders," Data Mining and Knowledge Discovery, vol. 25, 2012, pp. 67–108.

[10] S. Nowozin, "Discriminative subsequence mining for action classification," in IEEE Int. Conf. in Computer Vision (ICCV'2007), 2007.

[11] E. Ong, H. Cooper, N. Pugeault, and R. Bowden, "Sign language recognition using sequential pattern trees," in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'2012), 2012.

[12] E. Ong, O. Koller, N. Pugeault, and R. Bowden, "Sign spotting using hierarchical sequential patterns with temporal intervals," in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'2014), 2014.

[13] L. Brieman, J. Friedman, R. Olshen, and C. Stone, Classification and regression trees. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[14] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class adaboost," Statistics and Its Interface, vol. 2, 2009, pp. 349–360.