2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Online Learning of Temporal Logic Formulae for Signal Classification

Giuseppe Bombara and Calin Belta

Abstract— **This paper introduces a method for online inference of temporal logic properties from data. Specifically, we tackle the online supervised learning problem. In this setting, the data is in form of a set of pairs of signals and labels and it becomes available over time. We propose an approach for efficiently processing the data *incrementally*. In particular, when a new instance is presented, the proposed method *updates* a binary tree that is linked with the inferred Signal Temporal Logic (STL) formula. This approach presents several benefits. Primarily, it allows the refinement of the current formula when more data is acquired. Moreover, the incremental construction offers insights on the trade-off between formula complexity and classification accuracy. We present two case studies to emphasize the characteristics of the proposed algorithm: 1) a fault classification problem in an automotive system and 2) an anomaly detection problem in the maritime environment.**

I. Introduction

In recent years, there has been a great interest in applying machine learning based techniques to the formal methods field [1]–[6]. In particular, the research efforts have focused on inferring formal descriptions of the behavior of a system from its execution traces. The system behaviors are specified using an appropriate temporal logic language, such as Signal Temporal Logic (STL) [7]. This approach, named *Temporal Logic Inference* (TLI) in [5], while retaining many qualities of the traditional classifiers, presents several additional advantages. In particular, classical machine learning methods are often over specific to the task. That is, they focus exclusively on solving the classification problem but offer no other insight on the system where they have been applied. On the contrary, temporal logic formulae have precise meaning and allow for a rich specification of the behavior of a system that is *interpretable* by human experts (knowledge discovery). Moreover, the formulae learned with this approach can be employed for monitoring the system.

In this research field, the initial work has focused on finding the optimal parameters for a formula when the formula structure is given (fixed) [1]–[3]. Later, few attempts have been made to tackle the so called *two-class classification problem* [4]–[6]. In this setting, the goal is to build a temporal logic formula that can distinguish traces belonging to one of two possible classes. It is worth stressing that, in this case, *both* the structure of the formula and its parameters have to be learned from data. The dataset is given as a finite set of pairs of system traces, also called *signals*, and *labels*. Each label indicates whether the respective trace exhibits some desired system behavior, e.g., the engine is working correctly.

In this paper, we turn our attention to the *online learning problem*. In this scenario, it is assumed that new data arrives over time and the inference system should be updated to accommodate it. This is in contrast with the classical (or offline) approach, implicitly pursued by all the previous works, where only a single batch of data is available at the beginning and no further data can be considered. The online learning scenario presents some major advantages. First, it provides a formula early during the data collection process and then is able to refine it progressively when more data becomes available. Second, it removes the usual separation between building phase and deployment phase of classifiers.

A trivial solution to the online learning problem would be, every time a new instance arrives, to use an offline learner from scratch on the whole data accumulated so far. Clearly, this is highly inefficient as any formula discovered, and any associated data structure, would be thrown away. Therefore, the focus of this research is to devise a method that *builds* and *updates* an STL formula in an efficient manner.

In [6], the authors established a connection between STL inference and decision trees, where each node of a tree contains a test associated with the satisfaction of a simple formula, optimally tuned from a predefined set of primitive formulae. Unfortunately, updating a decision tree when new data arrives is not a trivial task. Specifically, if the arrival of a new signal causes the change of the best primitive in a node, then that node and all its children should be pruned and reconstructed from scratch [8]. Recently, a different prospective to tackle this problem has emerged from the study of data streams [9]–[11]. The key insight is to create a new node only when we can be *reasonably* sure that the decision made for the node holds true for future data. Following this idea, in this paper, we propose an algorithm that creates a new node only when some conditions on the best overall primitive to pick are attained. This strategy completely avoids the inefficient reconstruction process associated with a wrong primitive selection and can offer some probabilistic guarantees on the formula structure returned.

On top of the benefits provided by the online paradigm, the incremental construction procedure that we propose provides some additional advantages. First, it can potentially handle very large datasets as it is necessary to store in memory only the signals belonging to the leaf that is currently processed. Second, we will show that the evolution of the classification accuracy, as new data is processed by the algorithm, provides useful insights on the trade-off between the complexity of the formula inferred and its accuracy. This information can

G. Bombara is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215, USA gbombara@bu.edu

C. Belta is with the Department of Mechanical Engineering, Boston University, Boston, MA 02215, USA cbelta@bu.edu

This work was supported by DENSO CORPORATION and by the Office of Naval Research grant N00014-14-1-0554.

be exploited to halt the learning process when a satisfactory solution has already been obtained and to avoid phenomena such as overfitting.

II. Signal Temporal Logic

Let $\mathbb{R}_{\geq t}$ be the interval of real numbers $[t, \infty)$. We use $\mathcal{S} = \{s : \mathbb{R}_{\geq 0} \to \mathbb{R}^n\}$ with $n \in \mathbb{N}$ to denote the set of all continuous parameterized curves in the n-dimensional Euclidean space $\mathbb{R}^n$. In this paper, an element of $\mathcal{S}$ is called a *signal* and its parameter is interpreted as *time*. Given a signal s, the components of s are denoted by s_i, $i \in \{1, \ldots, n\}$. The set $\mathcal{F}$ contains the projection operators from a signal s to one of its components s_i, that is $\mathcal{F} = \{f_i : \mathbb{R}^n \to \mathbb{R}, f_i(s) = s_i\}$. The *suffix* at time $t \geq 0$ of a signal is denoted by $s[t]$ and it represents the signal s shifted forward in time by t units.

The syntax of *Signal Temporal Logic* (STL) is defined as:

$$\phi ::= \top \mid f(x) \sim \mu \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathbf{U}_{[a,b)} \phi_2$$

where $\top$ is the Boolean *true* constant ($\bot$ for *false*); $f(x) \sim \mu$ is a predicate over $\mathbb{R}^n$ defined by a function $f \in \mathcal{F}$, a real number $\mu \in \mathbb{R}$, and an order relation $\sim \in \{\leq, >\}$; $\neg$ and $\wedge$ are the Boolean operators negation and conjunction; and $\mathbf{U}_{[a,b)}$ is the bounded temporal operator *until*.

The semantics of STL is defined over signals in $\mathcal{S}$ as [7]:

$$\begin{aligned}
s[t] &\models \top && \Leftrightarrow \top \\
s[t] &\models f(x) \sim \mu && \Leftrightarrow f(s(t)) \sim \mu \\
s[t] &\models \neg\phi && \Leftrightarrow \neg(s[t] \models \phi) \\
s[t] &\models (\phi_1 \wedge \phi_2) && \Leftrightarrow (s[t] \models \phi_1) \wedge (s[t] \models \phi_2) \\
s[t] &\models (\phi_1 \mathbf{U}_{[a,b)} \phi_2) && \Leftrightarrow \exists t_u \in [t+a, t+b) \text{ s.t. } \big(s[t_u] \models \phi_2\big) \\
& && \quad \wedge \big(\forall t_1 \in [t, t_u)\ s[t_1] \models \phi_1\big)
\end{aligned}$$

A signal $s \in \mathcal{S}$ is said to satisfy an STL formula ϕ if and only if $s[0] \models \phi$. The other Boolean operations (i.e., disjunction, implication, equivalence) are defined in the usual way. Also, the temporal operators *eventually* and *globally* are defined as $\mathbf{F}_{[a,b)}\phi \equiv \top \mathbf{U}_{[a,b)} \phi$ and $\mathbf{G}_{[a,b)}\phi \equiv \neg\mathbf{F}_{[a,b)}\neg\phi$, respectively.

Parametric Signal Temporal Logic (PSTL) was introduced in [1] as an extension of STL where formulae are parameterized. A PSTL formula is similar to an STL formula, however all the time bounds associated with the temporal operators and all the constants in the inequality predicates are replaced by free parameters. If ψ is a PSTL formula, then every parameter assignment $\theta \in \Theta$ (where Θ is the parameter space of ψ) induces a corresponding STL formula $\phi = \psi(\theta)$, where the parameters of ψ have been fixed according to θ. This assignment is referred to as valuation θ of ψ. For example, given $\psi = \mathbf{F}_{[a,b)}(s > \pi)$ and $\theta = [1, 2.5, 3.7]$, the following STL formula is obtained $\psi(\theta) = \mathbf{F}_{[1,2.5)}(s > 3.7)$.

III. Problem formulation

We want to find an STL formula that separates traces produced by a system that exhibit some desired property, such as behaving correctly, from other traces of the same system. The normal working conditions are referred to as *positives*, whereas the non-conforming patterns are referred to as *negatives*, or *anomalies*. Let $C = \{C_p, C_n\}$ be the set of classes, with C_p for the positive class and C_n for the negative class. Let $s^i \in \mathcal{S}$ be an n-dimensional signal, and let $l^i \in C$ be its label, we consider the following problem:

Problem 1 (Two-Class Classification): Given a dataset of labeled signals $S_{\mathrm{ds}} = \{(s^i, l^i)\}_{i=1}^N$, we want to find an STL formula ϕ^* such that the misclassification rate $\mathrm{MCR}(\phi, S_{\mathrm{ds}})$ is minimized, where the misclassification rate is defined as:

$$\mathrm{MCR}(\phi, S_{\mathrm{ds}}) := \frac{\left|\left\{s^i \mid (s^i \models \phi, l^i = C_n) \text{ or } (s^i \not\models \phi, l^i = C_p)\right\}\right|}{|S_{\mathrm{ds}}|}$$

In the above formula, $(s^i \models \phi,\ l^i = C_n)$ represents a *false positive*, while $(s^i \not\models \phi,\ l^i = C_p)$ represents a *false negative*. To solve Prob. 1, we assume that the dataset S_{ds} of labeled signals generated by the system becomes progressively available. This as opposed to all the previous approaches [4]–[6], where S_{ds} is supposed to be fully available from the beginning and no further data can be considered.

IV. STL formulae and Decision Trees

In this section, we briefly review the connection between STL formulae and decision trees that was introduced in [6]. A decision tree is a tree-structured sequence of questions about the data used to make predictions about the data's labels. In a tree, we define: the root as the initial node; the depth of a node as the length of the path from the root to that node; the parent of a node as the neighbor whose depth is one less; the children of a node as the neighbors whose depths are one more. A node with no children is called a leaf, all other nodes are called non-terminal nodes. We focus on *binary* trees, where every non-terminal node splits the data in two children nodes and every leaf predicts a class.

Most decision-tree learning methods in the machine learning literature share a common algorithmic structure [12]. In particular, each algorithm defines two core components: 1) a list of possible ways to split the data reaching a node; and 2) a criterion to select the best split. In [6], it was proposed to split the signals using a simple formula at each node, chosen from a finite set of PSTL formulae, called *primitives*, along with its optimal parameters. A binary tree with this structure can be mapped to an equivalent STL formula using a simple algorithm that recursively traverses the tree, starting from its root, and only keeps track of the paths reaching leaves associated with the positive class C_p. Specifically, at each node the formula is obtained by (1) conjunction of the node's formula with its left subtree's formula, (2) conjunction of the negation of the node's formula with its right subtree's formula, (3) disjunction of (1) and (2). Fig. 1 shows a decision tree and its corresponding STL formula.

A. PSTL primitives

Let $\mathcal{S}$ be the set of signals with values in $\mathbb{R}^n$, a possible set of primitives to split the signals in a node is defined as:

Definition 4.1 (First-Level Primitives):

$$\begin{aligned}
\mathcal{P}^1 = \big\{&\mathbf{F}_{[\tau_1,\tau_2)}(f_i(x) \sim \mu) \text{ or } \mathbf{G}_{[\tau_1,\tau_2)}(f_i(x) \sim \mu) \\
&\mid i \in \{1, \ldots, n\},\ \sim \in \{\leq, >\}\big\}
\end{aligned}$$

The parameters of $\mathcal{P}^1$ are (μ, τ_1, τ_2) and the space of parameters is $\Theta^1 = \{(a, b, c) \mid a \in \mathbb{R},\ b < c,\ b, c \in \mathbb{R}_{\geq 0}\}$.

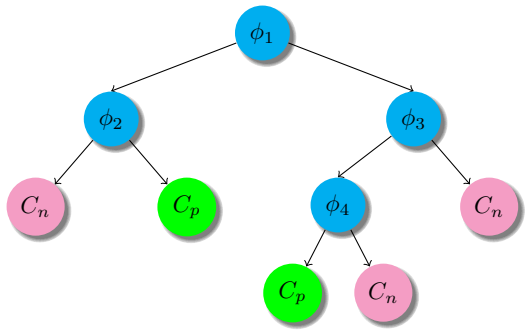

Fig. 1. Formula liked with this tree is $\phi = (\phi_1 \wedge \neg\phi_2) \vee (\neg\phi_1 \wedge (\phi_3 \wedge \phi_4))$.

The meaning of the primitives in $\mathcal{P}^1$ is straightforward. $\mathbf{F}_{[\tau_1,\tau_2)}(f_i(x) \sim \mu)$ and $\mathbf{G}_{[\tau_1,\tau_2)}(f_i(x) \sim \mu)$ are used to express that the predicate $f_i(x) \sim \mu$ must be true for at least one time instance or for all time instances in the interval $[\tau_1, \tau_2)$, respectively.

Remark 4.1: It is important to stress that the set of primitives $\mathcal{P}^1$ is not the only possible one. A user may define other primitives, for instance generic primitives using nested operators (such as $\mathbf{G}_{[\tau_1,\tau_2)}\mathbf{F}_{[0,\tau_3)}(f_i(x) \sim \mu)$), or specific ones, guided by the particular nature of the problem. We do not investigate other primitives here. However, the proposed method works without modifications using any other set.

B. Impurity measures

Along with a set of primitives, it is also necessary to define a criterion to select which primitive best splits the data at each node. Intuitively, a good split leads to children that are *homogeneous*, that is, they contain mostly signals belonging to the same class. This concept has been formalized in literature with *impurity measures* [12], and the goal is to obtain children *purer* than their parent. In [6], some impurity measures were extended to explicitly handle signals and STL formulae. In this paper, we consider only the misclassification gain.

Definition 4.2 (Misclassification Gain MG): Let S be a set of labeled signals and ϕ an STL formula, we define the following partition weights to describe how the signals s^i are distributed according to their labels l^i and the formula ϕ

$$p_\top = \frac{|S_\top|}{|S|}, \quad p_\bot = \frac{|S_\bot|}{|S|}, \quad p_c = \frac{|S_c|}{|S|} \tag{1}$$

where $S_\top = \{(s^i, l^i) \in S \mid s^i \models \phi\}$, $S_\bot = \{(s^i, l^i) \in S \mid s^i \not\models \phi\}$, and $S_c = \{(s^i, l^i) \in S \mid l^i = c\}$.
The misclassification gain is defined as:

$$\begin{aligned} MG(S,\phi) &= MR(S,\phi) - \sum_{\otimes \in \{\top,\bot\}} p_\otimes \cdot MR(S_\otimes, \phi) \\ MR(S,\phi) &= \min(p_{C_p}, p_{C_n}) \end{aligned} \tag{2}$$

Intuitively, a positive value of $MG(S,\phi)$ means that we have *reduced the impurity* by splitting the set S with the formula ϕ (or, equivalently, we have *gained purity*).

Algorithm 1: Online Decision Tree Construction

Input: (s, l) – a new labeled signal
Data: T – a tree

1 **if** T *does not exist* **then**
2 $\quad T \leftarrow emptyLeaf()$
3 $L, c, \phi^{path} \leftarrow locateLeaf(s, l)$
4 **if** $l \neq c$ **then**
5 $\quad updateLeaf(L, \phi^{path})$

V. Online Learning

In the batch algorithm proposed in [6], a *greedy* recursive procedure is followed to construct the tree, with all the data available S_{ds}, starting from the root. The data is partitioned as new nodes are created using locally optimal decisions on the signals reaching each node. The decision on which primitive to pick and which parameters to use is made by optimizing the impurity measure (2) on the set of primitives $\mathcal{P}^1$ and its space of parameters Θ^1 (see Def. 4.1).

As discussed in [8], updating a decision tree when new data arrives is not an easy task. Specifically, if the arrival of a new signal causes the change of the best primitive in a node, in the sense of the impurity measure, then that node and all its children should be pruned and reconstructed. A different prospective to deal with this problem has emerged from the research on data streams. Instead of creating a node immediately, based on the data currently available, the key idea is to defer its creation only when we can be *reasonably* sure that the decision made will hold in the future [9].

A. Online learning algorithm

In Alg. 1 we report the online procedure for inferring temporal logic formulae using high-level pseudocode. The algorithm can be executed whenever new signals are available. Alg. 1 operates on a data structure T representing the tree and takes as input a new labelled signal (s, l) to be processed. At the beginning, the algorithm checks if the tree exists (line 1). If it does not, it creates a tree with a single leaf at the root (line 2). When a tree exists, the new labelled signal (s, l) is sorted through the tree to the leaf L where it belongs and the label $c \in C$ of this leaf is examined (line 3). The label c associated with a leaf is simply chosen using the majority vote on the labels of the signals falling in that leaf. If the new signal is misclassified (line 4), that is $l \neq c$, the procedure $updateLeaf()$ is invoked on leaf L (line 5).

The procedure $updateLeaf()$, reported in Alg. 2, operates on a single leaf of the tree and performs three major steps. First, it finds the optimal parameters for each primitive in the set $\tilde{\mathcal{P}} \subseteq \mathcal{P}^1$ according to the misclassification gain (line 1). This optimization can be performed using any nonlinear solver. Second, it evaluates the status of the leaf to decide if it should be kept as a leaf or if a new non-terminal node can be created in its place (line 2). This part is discussed in the next section. Third, if the conditions are met (line 3), the leaf is transformed into a non-terminal node and it is associated

Algorithm 2: Update a Leaf – $updateLeaf(\cdot)$

Input: L – a leaf of tree T
Input: ϕ^{path} – formula associated with path to leaf L
Data: S – set of signals contained in leaf L
Data: $\tilde{\mathcal{P}}$ – set of candidate primitives for leaf L

1 $\theta_i \leftarrow \arg\max_{\theta\in\Theta} MG(S, \phi^{path} \wedge \psi_i(\theta)),\ \forall \psi_i \in \tilde{\mathcal{P}}$
2 $\tilde{\mathcal{P}}, \phi_{\text{bst1}}, createNode \leftarrow evalLeafStatus(S, \tilde{\mathcal{P}}, \{\theta_i\}_1^{|\tilde{\mathcal{P}}|})$
3 **if** $createNode == True$ **then**
4 $\quad N \leftarrow non\_terminal(\phi_{\text{bst1}})$
5 $\quad N.left \leftarrow emptyLeaf()$
6 $\quad N.right \leftarrow emptyLeaf()$
7 $\quad S_\top, S_\bot \leftarrow partition(S,\ \phi_{\text{bst1}})$
8 $\quad storeInLeaf(N.left, S_\top)$
9 $\quad storeInLeaf(N.right, S_\bot)$

with the optimal formula ϕ_{bst1} (line 4). Two empty leaves are initially added as children of this new node (line 5-6). Finally, the signals S are partitioned according to ϕ_{bst1} (line 7), and for each outcome of the split the corresponding partition is passed to the appropriate leaf (lines 8-9).

Algs. 1 and 2 use several functions: a) $emptyLeaf()$ creates a leaf with no signals in it and initializes the set of primitives to analyze $\tilde{\mathcal{P}}$ to $\mathcal{P}^1$; b) $locateLeaf(s, l)$ locates and stores a signal s with label l in the leaf L where it belongs according to the decision tree. c) $evalLeafStatus()$ evaluates the status of the candidate primitives in the leaf under analysis and checks the conditions to create a new node; d) $non\_terminal(\phi)$ creates a non-terminal node associated with the formula ϕ; e) $partition(S, \phi)$ splits the set of signals S into satisfying and non-satisfying signals with respect to ϕ; and f) $storeInLeaf(L, S)$ stores the signals S in leaf L.

Remark 5.1: Alg. 2 operates on a single leaf of the tree at the time and only the signals belonging to that specific leaf are required to be in memory. Therefore, this approach can potentially handle large datasets. Alternately, if all accumulated signals can be stored in memory, the addition of new signals can be parallelized over different leaves.

B. Primitive evaluation and node creation

Hypothetically, if an infinite set of signals S_∞ was available at a leaf, we would be able to pick the *best* formula to split the signals, both in terms of primitive and its parameters, with respect to the impurity measure (2). Assume that ϕ_{bst1} $(= \psi_{\text{bst1}}(\theta_{\text{bst1}}))$ is this formula, corresponding to the primitive ψ_{bst1} with optimal valuation θ_{bst1}. Assume also that ϕ_{bst2} is the best formula obtainable with any other primitive, say ψ_{bst2}, we would obviously have:

$$MG(S_\infty, \phi_{\text{bst1}}) - MG(S_\infty, \phi_{\text{bst2}}) > 0 \tag{3}$$

That is, primitive ψ_{bst1} provides an overall higher impurity reduction (purity gain) than ψ_{bst2}. With a finite amount of data, it is not possible to be sure about which formula is the best. However, some probabilistic guarantees on the best overall primitive to pick can be obtained using the finite set of signals S collected so far in the leaf. Following the idea initially proposed in [9], a bound is derived on the difference of purity gains (using just the signals S available), such that,

$$\text{if} \qquad MG(S, \phi_{\text{bst1}}) - MG(S, \phi_{\text{bst2}}) > \epsilon(S, \delta) \tag{4}$$

$$\text{then} \qquad \Pr(\Delta MG(S_\infty, \phi_{\text{bst1}}, \phi_{\text{bst2}}) > 0) \geq 1 - \delta \tag{5}$$

In other terms, if, on the S signals available, the difference between the misclassification gain of the best formula ϕ_{bst1}, obtained with the best primitive ψ_{bst1}, and the misclassification gain of the formula ϕ_{bst2}, obtained with the second best primitive ψ_{bst2}, is greater than a certain ϵ then, with probability greater than $1-\delta$, ψ_{bst1} is indeed better then ψ_{bst2} (as if we had access to infinite signals S_∞). Moreover, if (4) holds and since there are $|\mathcal{P}^1|$ primitives, we have that ψ_{bst1} is the *best* primitive with probability $(1-\delta)^{(|\mathcal{P}^1|-1)}$.

In literature [9]–[11], several approaches have been pursued to obtain a value for ϵ in (4) in order to guarantee (5). They vary on the impurity measure investigated and on the concentration inequality (Hoeffding, McDiarmid, etc) or probabilistic approximation used. In this paper, we employ the bound recently derived in [11]:

$$\epsilon(S, \delta) = z_{1-\delta} \frac{1}{\sqrt{2|S|}} \tag{6}$$

where $z_{1-\delta}$ is $(1-\delta)$-th quantile of the normal distribution. In general, ϵ depends on the confidence threshold δ and on the cardinality of S. ϵ grows as δ approaches 0 (i.e., we want more confidence) and becomes smaller as the number of signals acquired increases (i.e., we collected more evidence).

Remark 5.2: It may happen that two primitives are almost equally good in terms of impurity reduction. In this scenario, a large number of signals would be required to assess the best one. To avoid a long decision time, the split of a leaf can be forced when more than $N_{\max}$ signals have been collected (tie breaking). Even in this scenario, the probabilistic bound is useful to speed up the computation because it can be employed to progressively eliminate all the non-promising primitives from further analysis, that is, the primitives that compared with the current best satisfy the condition in (4).

The arguments previously discussed are used in the implementation of the function $evalLeafStatus()$ for Alg. 2. This function takes as input arguments the set of signals S collected so far in the leaf, the current set of candidate PSTL primitives $\tilde{\mathcal{P}} \subseteq \mathcal{P}^1$, and the optimal parameters for each primitive in $\tilde{\mathcal{P}}$, that is $\{\theta_i\}_1^{|\tilde{\mathcal{P}}|}$. It returns the updated set of PSTL primitives to consider in the future $\tilde{\mathcal{P}}$, the best splitting formula ϕ_{bst1}, and a boolean $createNode$ that indicates whether the leaf should become a new non-terminal node. $evalLeafStatus()$ performs three major actions. First, it finds the best primitive, that is, the one associated with the highest misclassification gain. Second, it removes all the non-promising primitives from set $\tilde{\mathcal{P}}$ by checking them against the best primitive using (4). Third, it sets $createNode = True$ if only one primitive is left in set $\tilde{\mathcal{P}}$, or if the number of signals in the leaf has exceeded the maximum $|S| > N_{\max}$.

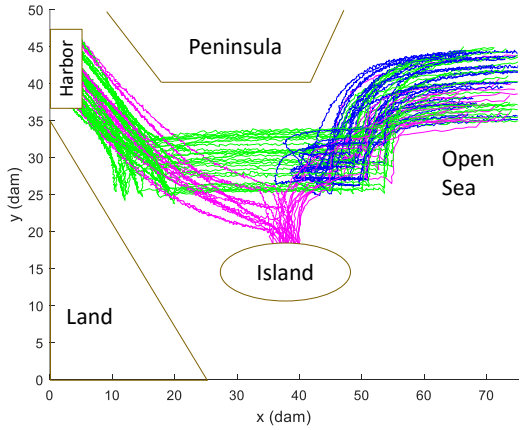

Fig. 2. Naval Surveillance dataset. The vessels behaving normally are shown in green. The magenta and blue trajectories represent two types of anomalous paths (human trafficking and terrorist activity, respectively).

The specific values of the probability confidence threshold δ in (5) and of the maximum number of signals $N_{\max}$ are meta-parameters of Alg. 2 and can be decided by the user.

Remark 5.3: Further considerations are possible during the leaf evaluation. For example, even if a good primitive has been found, a node creation can be deferred when the leaf contains mostly signals belonging to the same class. These considerations will be the subject of future work.

VI. Case Studies

We present two case studies to illustrate the effectiveness of the proposed algorithm and to analyze its behaviour. The first case study is a maritime surveillance problem, which was initially proposed in [5]. The second case study is a fault detection problem in an automotive powertrain system. This dataset was obtained by modifying a built-in Simulink model for the purposes of our investigation.

We implemented the algorithms described in Sec. V using MATLAB. The built-in Particle Swarm solver was used for the parameter optimization problem. To assess the performance of the proposed method we used a 5-fold cross-validation scheme [12]. One round of cross-validation entails partitioning the dataset into two subsets, performing the training on one subset, and the evaluation of the error on the other (testing). The training signals are presented to Alg. 1 one at the time. The results for each round are averaged to obtain a single estimate of the misclassification rate (MCR) and its standard deviation (STD). We ran our experiments on a PC with an Intel 5930K CPU and 16 GB RAM.

A. Naval Surveillance

This dataset emulates a maritime surveillance problem, where the goal is to detect suspicious vessels approaching the harbor from sea by looking at their trajectories [5].

The trajectories are represented with planar coordinates $[x(t), y(t)]$ and were generated using a Dubins' model with additive Gaussian noise. Three types of scenarios were considered: one normal and two anomalous. In the normal scenario, a vessel approaching from sea heads directly towards the harbor. In the first anomalous scenario, compatible with human trafficking, a ship veers to the island and heads to the harbor next. In the second anomalous scenario, compatible with terrorist activity, a boat tries to approach other vessels in the passage between the peninsula and the island and then veers back to the open sea.

The dataset is made of 2000 total traces, with 61 sample points per trace. There are 1000 normal traces and 1000 anomalous. Some sample traces are shown in Fig. 2. Using the cross-validation procedure, we obtained a mean MCR of 1.45% (STD 0.88%). The mean runtime (to process *all* the signals in the training set) was about 140 seconds per cross-validation round. A sample formula, obtained in one of the rounds after processing 1600 signals, is:

$$\phi = (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge (\phi_3 \vee (\neg\phi_3 \wedge \phi_4))) \tag{7}$$
$$\phi_1 = \mathbf{G}_{[211,295)}(x \leq 18.6) \qquad \phi_2 = \mathbf{G}_{[69.7,158)}(y > 23.7)$$
$$\phi_3 = \mathbf{G}_{[97.9,299)}(y \leq 32.4) \qquad \phi_4 = \mathbf{F}_{[61,178)}(x \leq 22.2)$$

B. Fuel Control System

We also investigated a fuel control system for a gasoline engine. The key quantity in the system is the *air-to-fuel ratio*, that is, the ratio between the mass of air and the mass of fuel in the combustion process. The system has one main output, the air-to-fuel ratio, one control variable, the fuel rate, and two inputs, the engine speed and the throttle command. The system estimates the correct fuel rate to achieve the target stoichiometric ratio by taking into account four sensor readings. Two sensors are related directly to the inputs: the engine speed and the throttle angle. The remaining two sensors provide feedback information: the EGO sensor reports the amount of residual oxygen present in the exhaust gas, and the MAP sensor reports the (intake) manifold absolute pressure.

Faults were artificially injected in both the EGO and MAP sensors with a *random* arrival time and with a *random* value. A total of 1200 simulations were performed, obtaining 600 normal traces and 600 anomalous, that is, with at least one faulty sensor. For every trace, we collected 200 samples from the EGO and MAP sensors. Using the cross-validation procedure, we obtained a mean MCR of 2.17% (STD 1.26%). The mean runtime was 115 seconds per round. A sample formula obtained during one round is reported in (9).

VII. Discussion

The maritime surveillance case study was also investigated in [5], with their batch SVM-based TLI approach, and in [6], using a batch decision tree algorithm. Therefore, some comparisons can be made. In terms of classification performance, the accuracy of the online algorithm presented here outperforms the results obtained in [5], improving the misclassification rate by a factor of 20, and is on par with the results of the offline algorithm in [6]. The execution time is not easily comparable because the problem setting is different. Specifically, both [5] and [6] present offline algorithms, that is, they process the whole dataset in a single batch and produce a formula, whereas the algorithm

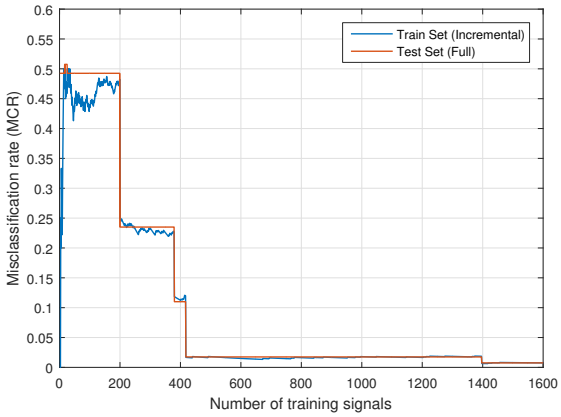

Fig. 3. MCR as function of the number of signal processed (Naval dataset). In blue, evolution of MCR computed on training signals (as seen by the algorithm). In red, evolution of MCR computed on an independent test set.

proposed here processes the dataset one signal at a time and produces a new formula after each addition. To elaborate the *whole* naval surveillance dataset, Alg. 1 is around 6 times faster than [5], and is roughly 2.5 times slower than the batch decision-tree algorithm in [6].

It is interesting to analyze how Alg. 1 performs as signals are incrementally processed. Fig. 3 displays how two error rates evolve as training signals from the naval dataset are presented to the algorithm. The first, in blue, is the misclassification rate on the set of *training* signals seen *so far* by the algorithm, while the second, in red, is the misclassification rate respect an independent (fixed) *test* set. As expected, the functions are flat for ample intervals with jumps at the points where the algorithm creates a new node in the tree. At these points, the corresponding formula becomes more complex (longer) and, generally, the misclassification rate decreases.

The evolution of the errors on the training and test data provides valuable information. For instance, if the training error decreases while the test error remains stable or increases, the formula inferred is getting overly specific to the training data and will not generalize well on unseen data (overfitting). From Fig. 3, it also clear that after a certain number of signals has been processed, adding more signals, while still increasing the complexity of the formula, does not improve the classification accuracy significantly. This information can be exploited to stop early the learning process (that is, *before* the whole dataset is processed) and focus on a reasonably accurate and more interpretable formula. For example in the naval case study, by looking at Fig. 3, if we stop after 420 signals have been processed (after around 30 seconds), we obtain a simpler formula (compare with (7)):

$$\phi = (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \phi_3) \tag{8}$$
$$\phi_1 = \mathbf{G}_{[211,295)}(x \leq 18.6) \qquad \phi_2 = \mathbf{G}_{[69.7,158)}(y > 23.7)$$
$$\phi_3 = \mathbf{G}_{[97.9,299)}(y \leq 32.4)$$

with an associated MCR of 2.5%. Notice also the insight we can gain from the English translation of (8): "Normal vessels' x coordinate is below 18.6 during the last 80 minutes, i.e., they approach and remain at the port", and "normal vessels' y coordinate never go below 23.7 in the time interval between 69.7 and 158 minutes, i.e., they do not approach the island".

Likewise, for the automotive case study, by stopping the learning process after 522 signals, we get the formula (x_1 corresponds to the EGO sensor):

$$\phi = \mathbf{F}_{[59.4,59.7)}(x_1 > .394) \wedge \mathbf{G}_{[24.2,59.5)}(x_1 \leq .896) \tag{9}$$

with an associated MCR of 2.9%.

VIII. Conclusion

We presented a method for learning signal classifiers in form of STL formulae incrementally from data. We exploited the connection between decision trees and STL formulae. In our algorithm, a new node is created only when some conditions on the overall best primitive to pick are attained. This strategy avoids any pruning and can handle large datasets. Finally, we argued that the evolution of the misclassification rate provides information on the trade-off between formula complexity and its accuracy.

References

[1] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *Runtime Verification*. Springer, 2012, pp. 147–160.

[2] B. Hoxha, A. Dokhanchi, and G. Fainekos, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *International Journal on Software Tools for Technology Transfer*, pp. 1–15, Feb. 2017.

[3] X. Jin, A. Donzé, J. Deshmukh, and S. A. Seshia, "Mining Requirements from Closed-Loop Control Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2015.

[4] S. Bufo, E. Bartocci, G. Sanguinetti, M. Borelli, U. Lucangelo, and L. Bortolussi, "Temporal Logic Based Monitoring of Assisted Ventilation in Intensive Care Patients," in *Leveraging Applications of Formal Methods, Verification and Validation*, ser. Lecture Notes in Computer Science. Springer, Oct. 2014, no. 8803, pp. 391–403.

[5] Z. Kong, A. Jones, and C. Belta, "Temporal Logics for Learning and Detection of Anomalous Behavior," *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1210–1222, Mar. 2017.

[6] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A Decision Tree Approach to Data Classification Using Signal Temporal Logic," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '16. New York, NY, USA: ACM, 2016, pp. 1–10.

[7] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic over Real-Valued Signals," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, no. 6246, pp. 92–106.

[8] P. E. Utgoff, N. C. Berkman, and J. A. Clouse, "Decision Tree Induction Based on Efficient Tree Restructuring," *Machine Learning*, vol. 29, no. 1, pp. 5–44, Oct. 1997.

[9] P. Domingos and G. Hulten, "Mining High-speed Data Streams," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 71–80.

[10] R. Jin and G. Agrawal, "Efficient Decision Tree Construction on Streaming Data," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 571–576.

[11] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "A New Method for Data Stream Mining Based on the Misclassification Error," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 5, pp. 1048–1059, May 2015.

[12] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge university press, 1996.