A Decision Tree Approach to Data Classification using Signal Temporal Logic

Giuseppe Bombara
Boston University
Boston, MA, USA
gbombara@bu.edu

Cristian-Ioan Vasile
Boston University
Boston, MA, USA
cvasile@bu.edu

Francisco Penedo
Boston University
Boston, MA, USA
franp@bu.edu

Hirotoshi Yasuoka
DENSO CORPORATION
Kariya, Aichi, Japan
hirotoshi_yasuoka@denso.co.jp

Calin Belta
Boston University
Boston, MA, USA
cbelta@bu.edu

ABSTRACT

This paper introduces a framework for inference of timed temporal logic properties from data. The dataset is given as a finite set of pairs of finite-time system traces and labels, where the labels indicate whether the traces exhibit some desired behavior (e.g., a ship traveling along a safe route). We propose a decision-tree based approach for learning signal temporal logic classifiers. The method produces binary decision trees that represent the inferred formulae. Each node of the tree contains a test associated with the satisfaction of a simple formula, optimally tuned from a predefined finite set of *primitives*. Optimality is assessed using heuristic *impurity* measures, which capture how well the current primitive splits the data with respect to the traces' labels. We propose extensions of the usual impurity measures from machine learning literature to handle classification of system traces by leveraging upon the *robustness degree* concept. The proposed incremental construction procedure greatly improves the execution time and the accuracy compared to existing algorithms. We present two case studies that illustrate the usefulness and the computational advantages of the algorithms. The first is an anomaly detection problem in a maritime environment. The second is a fault detection problem in an automotive powertrain system.

CCS Concepts

•**Computing methodologies** → **Logical and relational learning;** *Classification and regression trees;* •**Theory of computation** → *Modal and temporal logics;*

Keywords

Signal Temporal Logic; Logic Inference; Decision Trees; Impurity Measure; Machine Learning; Anomaly Detection; Supervised Learning;

HSCC '16, April 12–14, 2016, Vienna, Austria.

DOI: http://dx.doi.org/10.1145/2883817.2883843

1. INTRODUCTION

Machine learning deals with the construction of algorithms that can learn from data. Such algorithms operate by building a classifier from examples, called training data, in order to make accurate predictions on new data [24]. One of the main problems in machine learning is the so called *two-class classification problem*. In this setting, the goal is to build a classifier that can distinguish objects belonging to one of two possible classes. This problem is of fundamental importance because its solution leads to solving the more general multi-class problem [24]. Furthermore, it can be directly used in the context of anomaly detection, where the objective is to find patterns in data that do not conform to the expected behavior. These non-conforming patterns are often referred to as *anomalies* or *negatives*, whereas the normal working conditions are usually referred to as *targets* or *positives*. Given the importance of this problem and its broad applicability, it has been the topic of several surveys [16, 6].

A specific formulation of the two-class problem is determined by several factors such as the nature of the input data, the availability of labels, as well as the constraints and requirements determined by the application domain [6]. In this paper, we deal with data in form of finite time series, called signals or traces, and we suppose that the labels of these traces are available. That is, the true class of each trace is known, either *positive* or *negative*, and this information is exploited during the classifier construction phase (*supervised learning*). We tackle the two-class classification problem by bringing together concepts and tools from formal methods and machine learning. Our thesis is that a *formal specification* of the normal working conditions can be gleaned directly from execution traces and expressed in the form of Signal Temporal Logic (STL) formulae, a specification language used in the field of formal methods to define the behavior of continuous systems [22]. The inferred formulae can then be applied directly as data classifiers for new traces. In this context, some work has been initially done to optimize the parameters of a formula for a given, fixed, formula structure [17, 1, 26]. Kong et. al. [20, 18] were the first to propose an algorithm to learn *both* the formula structure and its parameters from data and called this approach *temporal logic inference* (TLI). This approach, while retaining many qualities of traditional classifiers, presents several additional advantages. First, STL formulae have precise mean-

ing and allow for a rich specification of the normal behaviour that is easily *interpretable by humans*. Second, anomaly detection methods commonly applied to time series data are often model-based, i.e., they require a *good* model of the system running alongside the physical system [16]. Third, 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, TLI fits naturally as a step in the system's design workflow and its analysis and results can be employed in other phases.

In this paper, we propose a novel, decision-tree based framework for solving the two-class classification problem involving signals using STL formulae as data classifiers. We refer to it as *framework* because we are not just proposing a single algorithm but a *class* of algorithms. Every algorithm produces a binary decision tree which can be translated to an STL formula and used for classification purposes. Each node of a tree is associated with a simple formula, chosen from a finite set of primitives. Nodes are created by finding the best primitive, along with its optimal parameters, within a greedy growing procedure. The optimality at each step is assessed using *impurity* measures, which capture how well a primitive splits the signals in the training data. The impurity measures described in this paper are modified versions of the usual impurity measures to handle signals, and were obtained by exploiting the *robustness degree* concept [9]. Our novel framework presents several advantages. In particular, the proposed incremental construction procedure requires the optimization of a small and fixed number of primitives at each node. Moreover, the number of objects to be processed decreases at each iteration. These two features greatly improve the execution time and the accuracy compared to the algorithms proposed in [20, 18].

This paper is organized as follows. In Section 2 we briefly survey some previous research efforts related to learning temporal logic formulae. In Section 3, we review the definition of Signal Temporal Logic, and its parameterized version PSTL used in the rest of the paper. The classification problem is formally stated in Section 4, and our decision tree framework is presented in detail in Section 5. Two case studies are introduced in Section 6. In Section 7 we report and discuss the results obtained by applying our temporal logic inference algorithms. We conclude in Section 8 with a summary and an outlook to future research directions.

2. RELATED WORK

Most of the recent research on temporal logical inference has focused on mining only the values of parameters associated with a given temporal logic formula structure [1, 26, 17, 2]. That is, a designer provides a formula template such as "The engine speed settles below v m/s within τ second" and an optimization procedure finds values for v and τ. The given structure reflects the (substantial) domain knowledge of the designer on the system and its properties of interest to be queried. With this approach, it is not possible to acquire new knowledge about the system directly from data, since it requires the designer to be very specific about the form of system properties that are investigated.

In [18, 20], the authors proposed methods for inferring both the formula structure and its parameters from data. They defined a fragment of STL, called inference parametric signal temporal logic (iPSTL), and showed that this fragment admits a partial order among formulae (1) in the sense of language inclusion, and (2) with respect to the robustness degree. This implies that iPSTL formulae can be organized in an infinite directed acyclic graph (DAG) according to how general they are (for any valuation). This result enabled them to formulate the classification problem as an optimization problem, whose objective function involves the robustness degree, and solve it in two cyclic steps: first, optimize the formula structure by exploring the DAG, pruning and growing it, and then, optimize the formula parameters, for a fixed structure, using a nonlinear optimization algorithm. This approach presents two major limitations. First, the parameter optimization routine has high computational cost. This is mostly due to its nonlinear nature. Finding the optimal valuation becomes more and more challenging as the algorithm proceeds, because the dimension of the parameter space grows at each iteration. This leads to long execution times. On the contrary, in our algorithm the dimension of the parameter space is fixed. Second, the DAG is built using an ordering on the language accepted by PSTL formulae. This has adverse effects on the performance. In particular, even though changing the formula structure according to the DAG offers guarantees in terms of the language, it does not imply an improvement in terms of the misclassification rate, which is the metric of interest for a classification problem. In Sec. 7, we show through a case study that our approach is able to obtain 20 times better classification performance with respect to the results in [19].

Recently, [3, 5] also tackled the two-class classification problem for inferring temporal logic formulae. Their approach can be divided in two separate steps. First, they build two generative models, one for each class. The models have to be in the form of stochastic systems and are used to compute the probability of satisfaction of a formula. Second, a discriminative formula is obtained by searching a formula that maximizes the odds of being true for the first model and false for the other model. As with other approaches, the formula structure and parameters are optimized separately. In particular, the formula structure is constructed through heuristics [3] or with a genetic algorithm [5], whereas the parameter space is explored through statistical model checking. This approach present some disadvantages. Primarily, it needs to build models of the system under analysis. This requires a domain expert and a certain amount of data. We do not agree with the authors' statement that model-based methods require less data than direct methods. On the contrary, we believe that more or the same amount of data is needed for the model parameter selection and the model validation. Overall, in the case studies reported, a significant designer intervention was required to guide the procedure to obtain a satisfactory formula. As opposed, our method does not need a model of the system nor an expert to guide the learning process.

To conclude, [12, 11] used a learning procedure for formulae defined in particular spatial superposition logics. These logics were developed for describing patterns in images without a time component. In particular, every image is represented with a multi-resolution format using a fixed height quad-tree data structure (which should not be confused with a decision tree). In this representation, every node contains an attribute describing an area of the image. Nodes that appear at deeper levels provide information about smaller areas. A pattern in an image corresponds to a path [12]

or a combination of several paths [11] in the relative quad-tree. Therefore, to describe the patterns, the semantics of these spatial logics are defined over the paths of quad-trees. In these works, formulae are learned from a labeled set of paths [12] or a labeled set of quad-trees [11] by applying off-the-shelf rule-based learning algorithms to the attributes of the nodes.

3. SIGNAL TEMPORAL LOGIC

Let $\mathbb{R}$ be the set of real numbers. For $t \in \mathbb{R}$, we denote the interval $[t, \infty)$ by $\mathbb{R}_{\geq t}$. 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 $s \in \mathcal{S}$, 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, specifically $\mathcal{F} = \{f_i : \mathbb{R}^n \to \mathbb{R}, f_i(s) = s_i, i = \{1, \ldots, n\}\}$. The *suffix* at time $t \geq 0$ of a signal is denoted by $s[t] \in \mathcal{S}$ and it represents the signal s shifted forward in time by t time units, i.e., $s[t](\tau) = s(\tau + t)$ for all $\tau \in \mathbb{R}_{\geq 0}$.

The syntax of *Signal Temporal Logic* (STL) [22] is defined as follows:

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

where $\top$ is the Boolean *true* constant; $p_{f(x) \leq \mu}$ is a predicate over $\mathbb{R}^n$ defined by the function $f \in \mathcal{F}$ and $\mu \in \mathbb{R}$ of the form $p_{f(x) \leq \mu}(x) = f(x) \leq \mu$; $\neg$ and $\wedge$ are the Boolean operators negation and conjunction; and $\mathcal{U}_{[a,b)}$ is the bounded temporal operator *until*. We use $\bot$ to denote the Boolean *false* constant.

The semantics of STL is defined over signals in $\mathcal{S}$ recursively as follows [22]:

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

A signal $s \in \mathcal{S}$ is said to satisfy an STL formula ϕ if and only if $s[0] \models \phi$. We extend the type of allowed inequality predicates in STL to $s[t] \models p_{f(x) > \mu} \equiv s[t] \models \neg p_{f(x) \leq \mu}$. Thus, predicates are defined in this paper by a function $f \in \mathcal{F}$, a real number $\mu \in \mathbb{R}$ and an order relation $\sim \in \{\leq, >\}$. 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 $\mathbf{F}_{[a,b)}\phi \equiv \top \mathcal{U}_{[a,b)} \phi$ and $\mathbf{G}_{[a,b)}\phi \equiv \neg \mathbf{F}_{[a,b)} \neg\phi$, respectively.

In addition to Boolean semantics defined above, STL admits *quantitative semantics* [9, 10], which is formalized by the notion of *robustness degree*. The robustness degree of a signal $s \in \mathcal{S}$ with respect to an STL formula ϕ at time t is a function $r(s, \phi, t)$ and is recursively defined as

$$\begin{aligned}
r(s, \top, t) &= r_\top \\
r(s, p_{f(x) \leq \mu}, t) &= \mu - f(s(t)) \\
r(s, \neg\phi, t) &= -r(s, \phi, t) \\
r(s, \phi_1 \wedge \phi_2, t) &= \min\{r(s, \phi_1, t), r(s, \phi_2, t)\} \\
r(s, \phi_1 \mathcal{U}_{[a,b)} \phi_2, t) &= \\
& \sup_{t_u \in [t+a, t+b)} \Big\{ \min\big\{r(s, \phi_2, t_u), \inf_{t_1 \in [t, t_u)} \{r(s, \phi_1, t_1)\}\big\}\Big\}
\end{aligned}$$

where $b > a > 0$ and $r_\top \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a large constant representing the maximum value of the robustness. Note that a positive robustness degree $r(s, \phi, 0)$ of a signal s with respect to a formula ϕ implies that s satisfies ϕ (in Boolean semantics). In the following, we denote by $r(s, \phi)$ the robustness degree $r(s, \phi, 0)$ at time 0. Robustness can be extended to the derived predicates and operators as follows:

$$\begin{aligned}
r(s, p_{f(x) > \mu}, t) &= f(s(t)) - \mu \\
r(s, \phi_1 \vee \phi_2, t) &= \max\{r(s, \phi_1, t), r(s, \phi_2, t)\} \\
r(s, \mathbf{F}_{[a,b)}\phi, t) &= \sup_{t_f \in [t+a, t+b)} \{r(s, \phi, t_f)\} \\
r(s, \mathbf{G}_{[a,b)}\phi, t) &= \inf_{t_g \in [t+a, t+b)} \{r(s, \phi, t_g)\}
\end{aligned}$$

Moreover, the interpretation of robustness degree as a quantitative measure of satisfaction is justified by the following proposition from [8].

PROPOSITION 3.1. *Let $s \in \mathcal{S}$ be a signal and ϕ an STL formula such that $r(s, \phi) > 0$. All signals $s' \in \mathcal{S}$ such that $\|s - s'\|_\infty < r(s, \phi)$ satisfy the formula ϕ, i.e., $s' \models \phi$.*

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 in the time intervals associated with temporal operators and all the constants in the inequality predicates are replaced by free parameters. The two types of parameters are called *time* and *space* parameters, respectively. Specifically, let ψ be a PSTL formula and n_p and n_{TL} be the number of predicates and temporal operators contained in ψ, respectively. The parameter space of ψ is $\Theta = \Pi \times T$, where $\Pi \subseteq \mathbb{R}^{n_p}$ is set of all possible *space* parameters and $T = T_1 \times \ldots T_{n_{TL}}$ is the set of all *time* parameters, where $T_i = \{(a_i, b_i) \in \mathbb{R}^2_{\geq 0} \mid a_i \leq b_i\}$ for all $i \in \{1, \ldots, n_{TL}\}$. Conversely, if ψ is a PSTL formula, then every parameter assignment $\theta \in \Theta$ induces a corresponding STL formula ϕ_θ, where all the space and time parameters of ψ have been fixed according to θ. This assignment is also referred to as a valuation θ of ψ. For example, given $\psi = \mathbf{G}_{[a,b)}(s_1 \leq c)$ and $\theta = [2.5, 0, 1]$, we obtain the STL formula $\phi_\theta = \mathbf{G}_{[0,1)}(s_1 \leq 2.5)$.

4. PROBLEM FORMULATION

We wish to find an STL formula that separates traces produced by a system that exhibit some desired property, such as behaving normally, from other traces of the same system. Formally, 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 be an n-dimensional signal, $s^i : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, and let $l^i \in C$ be its label. We consider the following problem:

PROBLEM 4.1 (TWO-CLASS CLASSIFICATION). *Given a set of labeled signals* $\{(s^i, l^i)\}_{i=1}^N$*, where* $l^i = C_p$ *if* s^i *exhibits a desired behavior, and* $l^i = C_n$ *if* s^i *does not, find an STL formula* ϕ *such that the misclassification rate* $\mathrm{MCR}(\phi)$ *is minimized, where the misclassification rate is defined as:*

$$\mathrm{MCR}(\phi) := \frac{\left|\left\{s^i \;\middle|\; (s^i \models \phi \wedge l^i = C_n) \vee (s^i \not\models \phi \wedge l^i = C_p)\right\}\right|}{N}$$

In the above formula, $|\cdot|$ denotes the cardinality of a set, and $(s^i \models \phi \wedge l^i = C_n)$ represents a *false positive*, while $(s^i \not\models \phi \wedge l^i = C_p)$ represents a *false negative*.

5. LEARNING DECISION TREES

In our approach, the key insight to tackle Problem 4.1 is that it is possible to build a map between a fragment of STL and decision trees. Therefore, we can exploit the decision trees learning literature [24, 23, 4] to build a decision tree that classifies signals and then map the constructed tree to an STL formula.

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. In this paper, we focus on *binary* decision trees, where every non-terminal node splits the data into two children nodes and every leaf node predicts a label.

Unfortunately, the space of all possible decision trees for a given classification problem is very large, and it is known that the problem of learning the optimal decision tree is NP-complete, for various optimality criteria [14]. Therefore, most decision-tree learning algorithms are based on *greedy* approaches, where locally optimal decisions are taken at each node. These greedy growing algorithms can be stated in a simple recursive fashion, starting from the root node, and require three meta-parameters: the first is a list of possible ways to split the data; the second is a criterion to select the best split; and the third is a set of rules for stopping the algorithm.

Several learning algorithms can be created by selecting different meta-parameters. That is, once the meta-parameters have been fixed, a specific learning algorithm is *instantiated*. Since we are not just proposing a single algorithm but a class of algorithms, we refer to this approach as "decision tree learning framework for temporal logic inference". In the next sections, we explain in detail the parameterized algorithm and the choices we propose for the meta-parameters.

5.1 Parameterized learning algorithm

In Alg. 1 we present the parameterized procedure for inferring temporal logic formulae from data. The meta-parameters of Alg. 1 are: (1) a set of PSTL primitives $\mathcal{P}$; (2) an impurity measure J; and (3) a set of stopping criteria *stop*. The algorithm is recursive and takes as input arguments the formula to reach the current node ϕ^{path}, the set of data that reached that node S, and the current depth level h.

At the beginning, the stopping conditions are checked (line 1). If they are met, the algorithm returns a single leaf node marked with the label $c \in C$. The label c is chosen according to the best classification quality (line 2), using

Algorithm 1: Parameterized Decision Tree Construction – $buildTree(\cdot)$

Parameter: $\mathcal{P}$ – set of PSTL primitives
Parameter: J – impurity measure
Parameter: *stop* – set of stopping criteria
Input: ϕ^{path} – formula associated with current path
Input: $S = \{(s^i, l^i)_{i=1}^N\}$ – set of labeled signals
Input: h – the current depth level
Output: a (sub)-tree

1 **if** $stop(\phi^{path}, h, S)$ **then**
2 $\quad t \leftarrow leaf(\arg\max_{c \in C}\{p(S, c; \phi^{path})\})$
3 $\quad$ **return** t
4 $\phi^* = \arg\max_{\psi \in \mathcal{P}, \theta \in \Theta} J(S, partition(S, \phi_\theta \wedge \phi^{path}))$
5 $t \leftarrow non\_terminal(\phi^*)$
6 $S^*_\top, S^*_\bot \leftarrow partition(S, \phi^{path} \wedge \phi^*)$
7 $t.left \leftarrow buildTree(\phi^{path} \wedge \phi^*, S^*_\top, h+1)$
8 $t.right \leftarrow buildTree(\phi^{path} \wedge \neg\phi^*, S^*_\bot, h+1)$
9 **return** t

$p(S, c; \phi^{path})$ defined in Def. 5.4. If the stopping conditions are not met (line 4), the algorithm proceeds to find the optimal STL formula among all the valuations of PSTL formulae from the set of primitives $\mathcal{P}$ (details in Sec. 5.3). The cost function used in the optimization is the impurity measure J, which assesses the quality of the partition induced by PSTL primitives valuations. See Sec. 5.4 for details. At line 5, a new non-terminal node is created and associated with the optimal STL formula ϕ^*. Next, the partition induced by the formula $\phi^{path} \wedge \phi^*$ is computed (line 6). For each outcome of the split, the $buildTree()$ procedure is called recursively to construct the left and right subtrees (lines 7-8). The corresponding formula to reach a subtree and the corresponding data partition are passed. The depth level is increased by one.

The parameterized family of algorithms uses three procedures: (a) $leaf(c)$ creates a leaf node marked with the label $c \in C$, (b) $non\_terminal(\phi)$ creates a non-terminal node associated with the valuation of a PSTL primitive from $\mathcal{P}$, and (c) $partition(S, \phi)$ splits the set of signals S into satisfying and non-satisfying signals with respect to ϕ, i.e., $S_\top, S_\bot = partition(S, \phi)$, where $S_\top = \{(s^i, l^i) \in S \mid s^i \models \phi\}$ and $S_\bot = \{(s^i, l^i) \in S \mid s^i \not\models \phi\}$.

By fixing the meta-parameters ($\mathcal{P}$, J, *stop*), a particular algorithm is *instantiated*. For each possible instance, a decision tree is obtained by executing $buildTree(\top, S_{root}, 0)$ on the set of labeled signals S_{root}. Clearly, the returned tree depends on both the input data S_{root} and the particular instance chosen.

5.2 Tree to STL formula

A decision tree obtained by an instantiation of Alg. 1 can be used directly for classification or converted to an equivalent STL formula using Alg. 2. The algorithm recursively traverses the subtree t given as input. At each node, the formula is obtained by (1) conjunction of the nodes'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). During the recursion process, Alg. 2 only keeps track of the paths reaching leaves associated with the positive class C_p. To produce the final

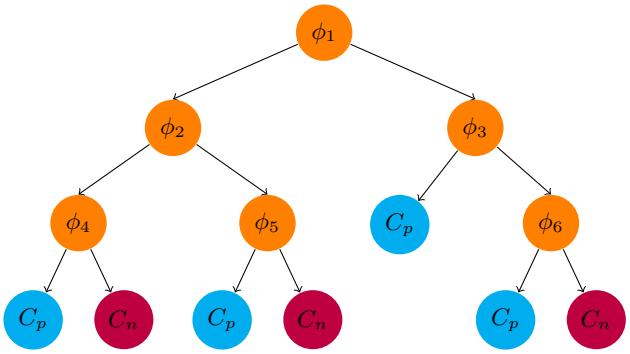

Figure 1: The formula associated with the tree is $\phi_{tree} = \Big(\phi_1 \wedge \big((\phi_2 \wedge \phi_4) \vee (\neg\phi_2 \wedge \phi_5)\big)\Big) \vee \Big(\neg\phi_1 \wedge \big(\phi_3 \vee (\neg\phi_3 \wedge \phi_6)\big)\Big)$ and can be obtained algorithmically using Alg. 2, where ϕ_i, $i \in \{1, \ldots, 6\}$ are valuations of primitive formulae from a set of PSTL formulae $\mathcal{P}$.

formula, the algorithm is executed starting from the root node, i.e., $Tree2STL(root)$. Fig. 1 shows a decision tree and its corresponding formula obtained by applying Alg. 2.

Algorithm 2: Tree to formula – $Tree2STL(\cdot)$

Input: t – node of a tree
Output: STL Formula

if *t is a leaf and class associated with t is C_p* **then**
  return $\top$
if *t is a leaf and class associated with t is C_n* **then**
  return $\bot$
$\phi_l = (t.\phi \wedge Tree2STL(t.left))$
$\phi_r = (\neg t.\phi \wedge Tree2STL(t.right))$
return $\phi_l \vee \phi_r$

5.3 PSTL primitives

To partition the data at each node, a finite list of possible splitting rules is usually considered [24]. We propose to use simple PSTL formulae, called *primitives*, to split the data. In particular, we define two types of primitives:

Definition 5.1 (First-Level Primitives). *Let $\mathcal{S}$ be the set of signals with values in $\mathbb{R}^n$, $n \geq 1$. We define the set of first-level primitives as follows:*

$$\mathcal{P}_1 = \big\{\mathbf{F}_{[\tau_1,\tau_2)}(x_i \sim \mu) \text{ or } \mathbf{G}_{[\tau_1,\tau_2)}(x_i \sim \mu) \mid i \in \{1, \ldots, n\},\ \sim \in \{\leq, >\}\big\}$$

The parameters of $\mathcal{P}_1$ are (μ, τ_1, τ_2) and the space of parameters is $\Theta_1 = \mathbb{R} \times \{(a, b) \mid a < b,\ a, b \in \mathbb{R}_{\geq 0}\}$.

Definition 5.2 (Second-Level Primitives). *Let $\mathcal{S}$ be the set of signals with values in $\mathbb{R}^n$, $n \geq 1$. We define the set of second-level primitives as follows:*

$$\mathcal{P}_2 = \big\{\mathbf{G}_{[\tau_1,\tau_2)}\mathbf{F}_{[0,\tau_3)}(x_i \sim \mu) \text{ or } \mathbf{F}_{[\tau_1,\tau_2)}\mathbf{G}_{[0,\tau_3)}(x_i \sim \mu) \mid i \in \{1, \ldots, n\},\ \sim \in \{\leq, >\}\big\}$$

The parameters of $\mathcal{P}_2$ are $(\mu, \tau_1, \tau_2, \tau_3)$ and the space of parameters is $\Theta_2 = \mathbb{R} \times \{(a, b) \mid a < b,\ a, b \in \mathbb{R}_{\geq 0}\} \times \mathbb{R}_{\geq 0}$.

The meaning of first-level primitives is straightforward. The two primitives $\mathbf{F}_{[\tau_1,\tau_2)}(x_i \sim \mu)$ and $\mathbf{G}_{[\tau_1,\tau_2)}(x_i \sim \mu)$ are used to express that the predicate $x_i \sim \mu$ must be true for at least one time instance or for all time instances in the interval $[\tau_1, \tau_2)$, respectively. Similarly, the second-level primitives can be interpreted in natural language as: (a) $\mathbf{F}_{[\tau_1,\tau_2)}\mathbf{G}_{[0,\tau_3)}(x_i \sim \mu)$ specifies that "the predicate $(x_i \sim \mu)$ of duration τ_3 must be performed and its start time must be in the interval $[\tau_1, \tau_2)$"; and (b) $\mathbf{G}_{[\tau_1,\tau_2)}\mathbf{F}_{[0,\tau_3)}(x_i \sim \mu)$ specifies that "at each time instance in the interval $[\tau_1, \tau_2)$, the predicate $(x_i \sim \mu)$ must be true within τ_3 time units". Both first- and second-level primitives may be thought as specifications for bounded reachability and safety with varying degrees of flexibility.

Given a set of primitives $\mathcal{P}$, we denote by $\text{STL}_\mathcal{P}$ the STL fragment obtained by Boolean closure from $\mathcal{P}$.

Definition 5.3 (Boolean Closure). *Let $\mathcal{P}$ be a finite set of PSTL formulae. The fragment of STL formulae induced by $\mathcal{P}$ using Boolean closure is defined as:*

$$\phi ::= \top \mid \varphi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

where φ is a valuation of a PSTL formula from $\mathcal{P}$.

$\text{STL}_\mathcal{P}$ is the fragment of STL that is mapped with decision trees. In other terms, each decision tree constructed with the set of primitives $\mathcal{P}$ is mapped to an STL formula belonging to the $\text{STL}_\mathcal{P}$ fragment.

Remark 1. Note that $\text{STL}_{\mathcal{P}_1} \subset \text{STL}_{\mathcal{P}_2}$, because $\mathbf{F}_{[\tau_1,\tau_2)}l \equiv \mathbf{F}_{[\tau_1,\tau_2)}\mathbf{G}_{[0,0^+)}l$ and similarly $\mathbf{G}_{[\tau_1,\tau_2)}l \equiv \mathbf{G}_{[\tau_1,\tau_2)}\mathbf{F}_{[0,0^+)}l$, where $l \equiv (x_i \sim \mu)$ is a linear inequality predicate and 0^+ represents the upper limit towards 0.

Remark 2. It is important to stress that the proposed PSTL primitives are not the only possible ones. A user may define other primitives, either generic ones, like the first- and second- level primitives, or specific ones, guided by the particular nature of the learning problem at hand.

5.4 Impurity measures

In the previous section, we defined a list of possible ways to split the data using a set of primitives $\mathcal{P}$. Now, it is 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 objects belonging to the same class. This concept has been formalized in literature with *impurity measures*, and the goal of the optimization algorithm is to obtain children purer that their parents. In this section, we first state the canonical impurity measures and then we propose three modified measures, which are more suited to handle signals, using the robustness degree.

Definition 5.4 (Impurity Measures). *Let S be a finite set of signals, ϕ an STL formula and $S_\top, S_\bot = partition(S, \phi)$. The following partition weights are introduced to describe how the signals s^i are distributed according to their labels l^i and the formula ϕ:*

$$p_\top = \frac{|S_\top|}{|S|},\ p_\bot = \frac{|S_\bot|}{|S|},\ p(S, c; \phi) = \frac{\big|\{(s^i, l^i) \mid l^i = c\}\big|}{|S|} \tag{1}$$

Particularly, $p_\top$ and $p_\bot$ represent the fraction of signals from S present in $S_\top$ and $S_\bot$, respectively, and $p(S, c; \phi)$ represents the fraction of signals in S that belong to class $c \in C$.

The (canonical) impurity measures are defined as [4, 23]:
- Information gain (IG)

$$IG(S, \{S_\top, S_\bot\}) = H(S) - \sum_{\otimes \in \{\top, \bot\}} p_\otimes \cdot H(S_\otimes)$$
$$H(S) = -\sum_{c \in C} p(S, c; \phi) \log p(S, c; \phi) \quad (2)$$

- Gini gain (GG)

$$GG(S, \{S_\top, S_\bot\}) = Gini(S) - \sum_{\otimes \in \{\top, \bot\}} p_\otimes \cdot Gini(S_\otimes)$$
$$Gini(S) = \sum_{c \in C} p(S, c; \phi)\big(1 - p(S, c; \phi)\big) \quad (3)$$

- Misclassification gain (MG)

$$MG(S, \{S_\top, S_\bot\}) = MR(S) - \sum_{\otimes \in \{\top, \bot\}} p_\otimes \cdot MR(S_\otimes)$$
$$MR(S) = \min(p(S, C_p; \phi), p(S, C_n; \phi)) \quad (4)$$

We extend the impurity measures to account for the robustness degrees of the signals to be classified. These extensions are based on the intuition that, according to Prop. 3.1, the robustness degree can be used in the context of learning as a measure of the classification quality of a signal with respect to an STL formula.

Definition 5.5 (Extended Impurity Measures). *Consider the same setup as in Def. 5.4, and the same impurity measures, we redefine the partition weights as follows:*

$$p_\top = \frac{\sum_{s^i \in S_\top} r(s^i, \phi)}{\sum_{s^i \in S} |r(s^i, \phi)|} \qquad p_\bot = -\frac{\sum_{s^i \in S_\bot} r(s^i, \phi)}{\sum_{s^i \in S} |r(s^i, \phi)|}$$
$$p(S, c; \phi) = \frac{\sum_{s^i \in S_c} |r(s^i, \phi)|}{\sum_{s^i \in S} |r(s^i, \phi)|} \quad (5)$$

where $S_c = \{s^i \in S \mid l^i = c\}$.

We will distinguish between the usual impurity measures and the extended ones by using the subscript r (e.g., IG_r) for the extended impurity measures. The following proposition ensures that the extended impurity measures are well defined.

Proposition 5.1. *The intra-partition weights are bounded within* 0 *and* 1 *and sum to* 1*, i.e.,* $0 \leq p_\top, p_\bot \leq 1$ *and* $p_\top + p_\bot = 1$*, in both definitions Def. 5.4 and Def. 5.5. The same invariant property is true for the inter-partition weights, i.e.,* $0 \leq p(S, C_n; \phi), p(S, C_p; \phi) \leq 1$ *and* $\sum_{c \in C} p(S, c; \phi) = 1$.

Remark 3. The advantages of using the extended versions of the impurity measures over the canonical ones are most pertinent in the context of optimizing these over PSTL formulae. The robustness-based impurity functions are better behaved cost functions, because these are less flat over the space parameter than their frequency-based counterparts, i.e., the canonical measures are piecewise constant functions. Also, we argue that the use of robustness makes the computed classifiers better at generalizing, i.e., performance on unseen (test) data. The intuition is that the separation boundaries tend to be as far as possible from signals of the two classes in the sense of robustness. In this sense, the canonical measures are unable to distinguish between formulae which are barely satisfied by some signals from more robust ones. As future work, an empirical comparison of the robustness-based measures against the canonical ones will be performed.

Local optimization
The cost function used in the local node optimization (line 4 of Alg. 1) is one of the impurity measures defined in the previous section. The optimization is performed over the chosen set of PSTL primitives $\mathcal{P}$ and their valuations Θ. Therefore, the optimization problem is decomposed into $|\mathcal{P}|$ optimization problems over a fixed and small number of real-valued parameters. Consider signals of dimension n. In the case of $\mathcal{P}_1$, we have $4n$ optimization problems with 3 parameters each. On the other hand, for $\mathcal{P}_2$ we have $4n$ optimization problems with 4 parameters each.

The local optimization approach presents several advantages. In particular, the computation of the robustness values in the definition of the extended impurity measures (Def. 5.5) can be performed incrementally with respect to the tree data structure according to the following preposition.

Proposition 5.2. (Incremental Computation of Robustness) *At each step of the recursion of Alg. 1, the robustness of a signal* s^i *reaching the current node* n_c *can be computed as follows*

$$r(s^i, \phi^{tree}) = r(s^i, \phi^{path} \wedge \phi) = \min\{r(s^i, \phi^{path}), r(s^i, \phi)\} \quad (6)$$

where ϕ^{tree} *corresponds to the currently computed tree,* ϕ^{path} *corresponds to the branch of the tree from the root to the parent of* n_c*, and* ϕ *is a candidate valuation of a PSTL primitive for* n_c.

The first equality in Eq. (6) follows from the construction of the tree, because the robustness of a signal s^i reaching n_c is negative for any other branch of the tree not ending in n_c. The incremental computation can be achieved by taking advantage of the recursion in the second equality in Eq. (6).

Another very important advantage of the proposed approach is that at each iteration of Alg. 1, the data is partitioned between the children of the currently processed node. Thus, the local optimization problems become easier as the depth of the nodes increases.

The local optimization problems may be solved using any global non-linear optimization algorithm, such as Simulated Annealing [15] or Differential Evolution [25]. However, in order to use these numerical optimization algorithms, we need to define finite bounds for the parameters of the primitive formulae. These bounds may easily be inferred from data, but may also be application specific, if expert knowledge is available.

5.5 Stop conditions

Several stopping criteria can be set for Alg. 1. The most common strategy is to just split until the current node contains only signals from a single class or no signals. This strategy is very permissive, that is, it allows the algorithm to run for many iterations. However, it represents the sufficient conditions that guarantee the termination of the algorithm. Other more restrictive conditions are possible. For instance, stop if the vast majority of the signals belong to the same class, either positive or negative, e.g., stop if 99% of signals belong to the same class. Another common strategy is to stop if the algorithm has reached a certain, fixed,

depth. These conditions usually provide a faster termination of the algorithm. In general, a set of stopping criteria can be assembled by picking several stopping conditions, as long as the sufficient conditions for the termination of the algorithm are included.

5.6 Complexity

In this section, we provide a worst-case and average-case complexity analysis of Alg. 1 in terms of the complexity of the local optimization procedure (Alg. 1, line 4). This complexity analysis assumes that just the sufficient stopping conditions are set. Let $C(N)$ and $g(N)$ be the complexity of Alg. 1 and of the local optimization algorithm, respectively, where N is the number of signals to be processed by the algorithms. Trivially, we have $g(N) = \Omega(N)$, where $\Omega(\cdot)$ is the asymptotic notation for lower bound [7], because the algorithm must at least check the labels of all signals. The worst-case complexity of Alg. 1 is attained when at each node the optimal partition has size $(1, N-1)$. In this case, the complexity satisfies the recurrence $C(N) = C(N-1) + C(1) + g(N)$, which implies $C(N) = \Theta(N + \sum_{k=2}^{N} g(k))$, where $\Theta(\cdot)$ is the two-sided asymptotic notation for complexity bound [7]. However, the worst case scenario is not likely to occur in large datasets. Therefore, we consider the average case where at least a fraction $\gamma \in (0, 1)$ of the signals are in one set of the partition. The recurrence relation becomes $C(N) = C(\gamma N) + C((1-\gamma)N) + g(N)$, which implies the following complexity bound

$$C(N) = \Theta\left(N \cdot \left(1 + \int_1^x \frac{g(u)}{u^2} \mathrm{d}\, u\right)\right)$$

obtained using the Akra-Bazzi method [7]. Finally, note that the hidden constants in the complexity bounds above depend on the cardinality of the set of primitives considered and the size of their parameterization.

6. CASE STUDIES

In this section, we present two case studies that illustrate the usefulness and the computational advantages of the algorithms. The first is an anomalous trajectory detection problem in a maritime environment. The second is a fault detection problem in an automotive powertrain system. The automotive application is particularly appealing because the systems involved are getting more and more sophisticated. In a modern vehicle, several highly complex dynamical systems are interconnected and the methods present in literature may fail to cope with this complexity.

6.1 Maritime surveillance

This synthetic dataset emulates a maritime surveillance problem, where the goal is to detect suspicious vessels approaching the harbor from sea by looking at their trajectories. It was developed in [19], based on the scenarios described in [21], for evaluating their inference algorithms.

The trajectories are represented with planar coordinates $x(t)$ and $y(t)$ and were generated using a Dubins' vehicle model with additive Gaussian noise. Three types of scenarios, one normal and two anomalous, were considered. In the normal scenario, a vessel approaching from sea heads directly towards the harbor. In the first anomalous scenario, a ship veers to the island and heads to the harbor next. This scenario is compatible with human trafficking. In the second

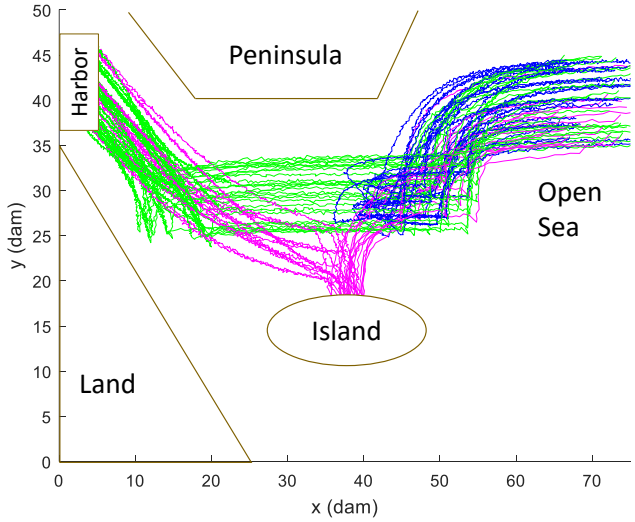

Figure 2: Naval surveillance dataset [19]. The vessels behaving normally are shown in green. The magenta and blue trajectories represent two types of anomalous paths.

anomalous scenario, a boat tries to approach other vessels in the passage between the peninsula and the island and then veers back to the open sea. This scenario is compatible with terrorist activity. Some sample traces are shown in Fig. 2. The dataset is composed of 2000 total traces, with 61 sample points per trace. There are 1000 normal traces and 1000 anomalous.

6.2 Fuel control system

We investigate a fuel control system for a gasoline engine. A model for this system is provided as built-in example in Simulink and we modified it for our purposes. This model was initially used for Bayesian statistical model checking [27] and has been recently proposed as benchmark for the hybrid systems community [13]. We selected this model because it includes all the complexities of real world industrial models, but is still quick to simulate, i.e., it is easy to obtain a large number of traces.

The key quantity in the model 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 goal of the control system is to keep it close to the "ideal" stoichiometric value for the combustion process. For this system, the target air-fuel ratio is 14.6, as it provides a good compromise between power, fuel economy, and emissions. 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 are related directly to the inputs, the engine speed and the throttle angle. The remaining two sensors provide crucial 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. The EGO value is related to the air-to-fuel ratio, whereas the MAP value is related to the air mass rate. The Simulink diagram is made of several subsystems with different kinds of blocks, both continuous and discrete,

Instance	Primitives	Impurity	Stopping
I_1	$\mathcal{P}_1$	MG_r	Majority class rate >0.975, Depth >4
I_2	$\mathcal{P}_2$	IG_r	Depth >3

Table 1: Algorithm meta-parameters. Refer to Sec. 5 for details.

among which there are look-up tables and a hybrid automaton. Due to these characteristics, this model can exhibit a rich and diverse number of output traces, thus making it an interesting candidate for our investigation.

The base model, that is, the one included in Simulink, includes a very basic fault detection scheme and fault injection mechanism. The fault detection scheme is a simple threshold crossing test (within a Stateflow chart), and is only able to detect single off range values. For avoiding the overlap of two anomaly detection schemes, the built-in one has been removed. In the base model, the faults are injected by simply reporting an incorrect and fixed value for a sensor's reading. Moreover, these faults are always present from the beginning of the simulation. We replaced this simple fault injection mechanism with a more sophisticated unit. The new subsystem is capable of inducing faults in both the EGO and MAP sensors with a *random* arrival time and with a *random* value. Specifically, the faults can manifest at anytime during the execution (uniformly at random) and the readings of the sensors affected are offset by a value that *varies* at every execution. Finally, independent Gaussian noise signals, with zero mean and variance $\sigma^2 = 0.01$, have been added at the output of the sensors.

For the fuel control system, 1200 total simulations were performed. In all cases, the throttle command provides a periodic triangular input, and the engine speed is kept constant at 300 rad/sec (2865 RPM). The simulation time is 60 seconds. In details, we obtained: 600 traces where the system was working normally; 200 traces with a fault in the EGO sensor; 200 traces with a fault in the MAP sensor; 200 traces with faults in both sensors. For every trace, we collected 200 samples of the EGO and MAP sensors' readings. Some sample traces are shown in Fig. 3. The average simulation time to obtain a single trace was roughly 1 second.

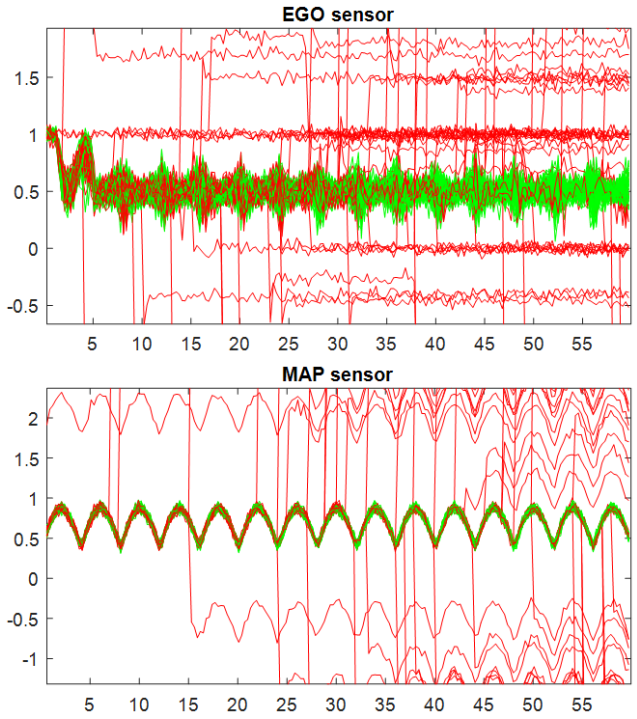

Figure 3: Fuel Control Dataset. Normal traces are shown in green, anomalous traces are shown in red.

7. IMPLEMENTATION AND RESULTS

We implemented and tested two different instances of Alg.1, I_1 and I_2, defined by the choice of meta-parameters given in Table 1. In the case of I_1, the implementation was done in MATLAB using standard libraries, employing the simulated annealing optimization method [15], and run on a 3.5 GHz processor with 16 GB RAM. As for I_2, we used the SciPy library for Python, solving the optimization problem with its implementation of the differential evolution algorithm [25], and we tested it on similar hardware. [1]

7.1 Maritime surveillance

We tested the I_2 instance using a non stratified 10-fold cross-validation with a random permutation of the data set, obtaining a mean misclassification rate of 0.007 with a standard deviation of 0.008 and a run time of about 4 hours per split. A sample formula learned in one of the cross-validation splits is:

$$
\begin{aligned}
\phi^{I_2} &= (\phi_1^{I_2} \wedge (\neg\phi_2^{I_2} \vee (\phi_2^{I_2} \wedge \neg\phi_3^{I_2}))) \\
&\qquad \vee (\neg\phi_1^{I_2} \wedge (\phi_4^{I_2} \wedge \phi_5^{I_2})) \\
\phi_1^{I_2} &= \mathbf{G}_{[199.70,297.27)}\mathbf{F}_{[0.00,0.05)}(x \leq 23.60) \\
\phi_2^{I_2} &= \mathbf{G}_{[4.47,16.64)}\mathbf{F}_{[0.00,198.73)}(y \leq 24.20) \\
\phi_3^{I_2} &= \mathbf{G}_{[34.40,52.89)}\mathbf{F}_{[0.00,61.74)}(y \leq 19.62) \\
\phi_4^{I_2} &= \mathbf{G}_{[30.96,37.88)}\mathbf{F}_{[0.00,250.37)}(x \leq 36.60) \\
\phi_5^{I_2} &= \mathbf{G}_{[62.76,253.23)}\mathbf{F}_{[0.00,41.07)}(y \leq 29.90)
\end{aligned}
\tag{7}
$$

We can see in Fig. 4 how the thresholds for ϕ_1 and ϕ_2 capture the key features of the data set. Notice also the insight we can gain from their plain English translation: "Normal vessels' x coordinate is below 23.6 during the last 100 seconds, i.e., they approach and remain at the port", and "normal vessels' y coordinate never go below 24.2, i.e., they don't approach the island". It is worth mentioning the second term of the outer disjunction in ϕ^{I_2}, as it highlights a feature of the data set difficult to spot on the figures: some normal vessels don't reach the port (inspecting the data set, some normal traces stop right after crossing the passage). As usual when employing decision trees, deeper formulae focus on finer details of the data set.

In the case of I_1, we tested it using a 5-fold cross-validation, obtaining a mean misclassification rate of 0.0040 and a standard deviation of 0.0029. The run time is about 16 minutes

[1]The software is available at http://hyness.bu.edu/Software.html

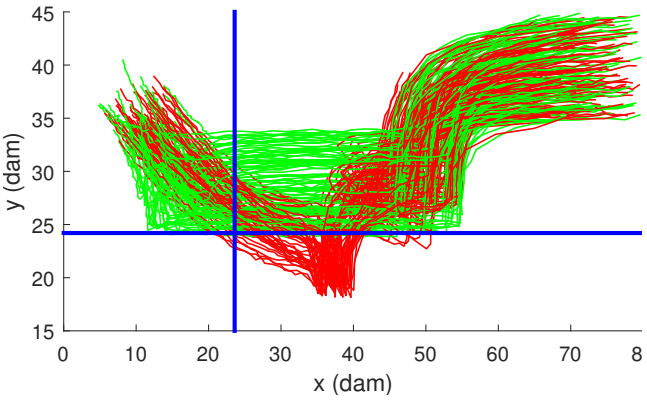

Figure 4: Sample of the naval surveillance dataset. Normal trajectories are green and anomalous trajectories are red. We show in blue the boundaries of $\phi_1^{I_2}$ and $\phi_2^{I_2}$ of Eq. (7).

per split. A sample formula learned in one of the splits is:

$$\begin{aligned}
\phi^{I_1} &= (\phi_1^{I_1} \wedge \phi_2^{I_1}) \vee (\neg\phi_1^{I_1} \wedge ((\phi_3^{I_1} \wedge (\phi_4^{I_1} \wedge \phi_5^{I_1})) \vee (\neg\phi_3^{I_1} \wedge \phi_6^{I_1})))\\
\phi_1^{I_1} &= \mathbf{G}_{[224,280)}(x \leq 18) \qquad \phi_2^{I_1} = \mathbf{G}_{[14.2,125)}(y > 22)\\
\phi_3^{I_1} &= \mathbf{F}_{[109,277)}(y > 30.6) \qquad \phi_4^{I_1} = \mathbf{G}_{[279,293)}(x \leq 19.2)\\
\phi_5^{I_1} &= \mathbf{F}_{[77.8,107)}x > 41) \qquad \phi_6^{I_1} = \mathbf{F}_{[258,283)}(x \leq 29.5)
\end{aligned} \tag{8}$$

This dataset was also used in [19]. Unfortunately, it is not possible to make a formal comparison between the formulae learned by our approach and the ones in [19]. This is due to the fact that iPSTL, defined in [19], and $\text{STL}_{\mathcal{P}_1}$ (or $\text{STL}_{\mathcal{P}_2}$) do not represent the same STL fragment. However, it is always possible to make a comparison in terms of sheer classification performance. In the comparison, it is clear that we improve the misclassification rate by a factor of 20 while spending a similar amount of execution time.

7.2 Fuel control

In this scenario, we tested both instances using the EGO and MAP sensors' readings (variables x_1 and x_2). We performed a similar cross-validation for I_2, resulting in a mean misclassification rate of 0.054 with a standard deviation of 0.025 and a run time of about 15 hours per split. A sample formula, obtained from one of the cross-validation splits, is:

$$\begin{aligned}
\phi^{I_2} &= \neg\phi_1^{I_2} \wedge \phi_2^{I_2} \wedge \phi_3^{I_2}\\
\phi_1^{I_2} &= \mathbf{F}_{[1.85,58.70)}\mathbf{G}_{[0.00,0.57)}(x_1 \leq 0.13)\\
\phi_2^{I_2} &= \mathbf{G}_{[11.35,59.55)}\mathbf{F}_{[0.00,0.03)}(x_1 \leq 0.99)\\
\phi_3^{I_2} &= \mathbf{G}_{[1.65,58.89)}\mathbf{F}_{[0.00,0.44)}(x_2 \leq 0.90)
\end{aligned} \tag{9}$$

Notice in this case how the resulting subformulae are equivalent to first-level primitives, suggesting that $\mathcal{P}_2$ is an overly complicated set of primitives.

Regarding I_1, using a 5-fold cross-validation, we obtained a mean misclassification rate of 0.0350 and a standard deviation of 0.0176. The run time is about 18 minutes per split.

A sample formula learned in one of the splits is:

$$\begin{aligned}
\phi_1 &= (\phi_1^{I_1} \wedge (\phi_2^{I_1} \wedge (\phi_3^{I_1} \wedge \phi_4^{I_1}))) \vee (\neg\phi_1^{I_1} \wedge (\phi_5^{I_1} \wedge (\phi_6^{I_1} \wedge \phi_7^{I_1})))\\
\phi_1^{I_1} &= \mathbf{F}_{[22,58.4)}(x_2 > 0.932) \quad \phi_2^{I_1} = \mathbf{G}_{[29.3,59.6)}(x_2 < 0.994)\\
\phi_3^{I_1} &= \mathbf{G}_{[56,58.3)}(x_1 > 0.0979) \quad \phi_4^{I_1} = \mathbf{G}_{[49.9,55.3)}(x_1 < 0.863)\\
\phi_5^{I_1} &= \mathbf{G}_{[39.5,58.4)}(x_2 > 0.193) \quad \phi_6^{I_1} = \mathbf{F}_{[59.4,59.7)}(x_1 > 0.25)\\
\phi_7^{I_1} &= \mathbf{G}_{[2.52,53.3)}(x_1 < 1.05)
\end{aligned} \tag{10}$$

In both case studies, the execution time of I_2 is higher then I_1. This occurs because the instance I_2 involves a more complicated optimization problem. Specifically, I_2 uses primitives from $\mathcal{P}_2$ with 4 free parameters, whereas I_1 uses primitives with only 3 free parameters.

8. CONCLUSION

In this paper, we presented an inference framework of timed temporal logic properties from time series data. The framework defines customizable decision-tree algorithms that output Signal Temporal Logic (STL) formulae as classifiers. This work is in line with recent interest in Temporal Logic Inference (TLI) and is motivated by the need to construct classifiers which provide good performance and can be interpreted over specific application domains. The proposed algorithms are model-free and are suitable for inferring properties from time series data for problems such as anomaly detection, monitoring, and application domains as diverse as the automotive industry and maritime port security.

The proposed framework describes decision-tree learning algorithms which may be customized by providing three components: (a) a set of primitive properties of interest; (b) an impurity measure which captures the node's homogeneity; and (c) stopping conditions for the algorithm. The performance advantage of the proposed procedures is due to the incremental nature of growing STL formulae represented as trees. Moreover, the problem of finding optimal primitives becomes easier as a procedure grows a tree. This follows from the fact that a node's optimization problem has always a fixed number of parameters and the data is partitioned between the two children of the node. Another contribution of the paper is the definition of extended versions of the classical impurity measures such that these take into account the robustness degrees of signals. We argue that the extended versions of the impurity measures increase the generalization capability of the resulting formulae.

In the paper, we test two possible instances of the framework (form a possibly very large set of choices) on two case studies in the maritime security and automotive fields. We show that the algorithms are able to capture relevant timed properties in both cases. The quality of the computed STL formulae is assessed using the misclassification rate averaged over multiple test folds.

Future work includes extending the proposed framework to online mode, where traces are provided incrementally, instead of a single batch of signals available from the beginning of the learning procedure. We plan to perform a comprehensive comparative study of the framework for multiple choices of primitive formulae sets and impurity measures, tested on case studies of varying complexity. Future work will also focus on improving the local optimization procedures, which will boost the overall performance of the framework.

9. ACKNOWLEDGMENTS

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

10. REFERENCES

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

[2] E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25, July 2015.

[3] E. Bartocci, L. Bortolussi, and G. Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Formal Modeling and Analysis of Timed Systems*, pages 23–37. Springer, 2014.

[4] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[5] 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*, number 8803 in Lecture Notes in Computer Science, pages 391–403. Springer, Oct. 2014.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Comput Surv*, 41(3):15:1–15:58, July 2009.

[7] T. H. Cormen. *Introduction to Algorithms*. MIT Press, third edition, July 2009.

[8] A. Donzé, T. Ferrere, and O. Maler. Efficient robust monitoring for STL. In *Computer Aided Verification*, pages 264–279. Springer, 2013.

[9] A. Donzé and O. Maler. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In K. Chatterjee and T. A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, number 6246 in Lecture Notes in Computer Science, pages 92–106. Springer Berlin Heidelberg, 2010.

[10] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, Sept. 2009.

[11] E. A. Gol, E. Bartocci, and C. Belta. A formal methods approach to pattern synthesis in reaction diffusion systems. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 108–113. IEEE, 2014.

[12] R. Grosu, S. A. Smolka, F. Corradini, A. Wasilewska, E. Entcheva, and E. Bartocci. Learning and detecting emergent behavior in networks of cardiac myocytes. *Commun. ACM*, 52(3):97–105, 2009.

[13] B. Hoxha, H. Abbas, and G. Fainekos. Benchmarks for temporal logic requirements for automotive systems. *Proc Appl. Verification Contin. Hybrid Syst.*, 2014.

[14] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

[15] L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control Cybern.*, 25:33–54, 1996.

[16] R. Isermann. *Fault-diagnosis systems*. Springer, 2006.

[17] X. Jin, A. Donzé, J. Deshmukh, and S. A. Seshia. Mining Requirements from Closed-Loop Control Models. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, PP(99):1–1, 2015.

[18] A. Jones, Z. Kong, and C. Belta. Anomaly detection in cyber-physical systems: A formal methods approach. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 848–853. IEEE, 2014.

[19] Z. Kong, A. Jones, and C. Belta. Temporal Logics for Learning and Detection of Anomalous Behaviors. *IEEE Trans. Autom. Control*, 2016. inpress.

[20] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta. Temporal Logic Inference for Classification and Prediction from Data. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, pages 273–282, New York, NY, USA, 2014. ACM.

[21] K. Kowalska and L. Peel. Maritime anomaly detection using Gaussian Process active learning. In *2012 15th International Conference on Information Fusion (FUSION)*, pages 1164–1171, July 2012.

[22] O. Maler and D. Nickovic. Monitoring Temporal Properties of Continuous Signals. In Y. Lakhnech and S. Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, number 3253 in Lecture Notes in Computer Science, pages 152–166. Springer Berlin Heidelberg, 2004.

[23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, June 2014.

[24] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge university press, 1996.

[25] R. Storn and K. Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, Dec. 1997.

[26] H. Yang, B. Hoxha, and G. Fainekos. Querying Parametric Temporal Logic Properties on Embedded Systems. In *Testing Software and Systems*, number 7641 in Lecture Notes in Computer Science, pages 136–151. Springer, 2012.

[27] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to Stateflow/Simulink verification. *Form Methods Syst Des*, 43(2):338–367, Aug. 2013.