

# Classification of Time-Series Data Using Boosted Decision Trees

Erfan Aasi<sup>1</sup>, Cristian Ioan Vasile<sup>2</sup>, Mahroo Bahreinian<sup>1</sup>, and Calin Belta<sup>1</sup>

**Abstract**—Time-series data classification is central to the analysis and control of autonomous systems, such as robots and self-driving cars. Temporal logic-based learning algorithms have been proposed recently as classifiers of such data. However, current frameworks are either inaccurate for real-world applications, such as autonomous driving, or they generate long and complicated formulae that lack interpretability. To address these limitations, we introduce a novel learning method, called Boosted Concise Decision Trees (BCDTs), to generate binary classifiers that are represented as Signal Temporal Logic (STL) formulae. Our algorithm leverages an ensemble of Concise Decision Trees (CDTs) to improve the classification performance, where each CDT is a decision tree that is empowered by a set of techniques to generate simpler formulae and improve interpretability. The effectiveness and classification performance of our algorithm are evaluated on naval surveillance and urban-driving case studies.

## I. INTRODUCTION

To cope with the complexity of robotic tasks, machine learning (ML) techniques have been employed to capture their temporal and logical structure from time-series data. One of the main problems in ML is the two-class classification problem, where the goal is to build a classifier that distinguishes desired system behaviors from the undesired ones. Traditional ML algorithms focus on building such classifiers; however, they are often not easy to understand or they don't offer any insights about the system. Motivated by the readability and interpretability of temporal logic formulae [1], there has been great interest in applying formal methods to ML in recent years [2], [3], [4], [5], [6], [7], [8].

Signal Temporal Logic (STL) [9] is a specification language used to express temporal properties of real-valued signals. In this paper, we use STL to generate specifications of time-series system behaviors. Early methods for mining temporal properties from data mostly focus on parameter synthesis, given template formulae [2], [10], [11], [12]. These works require the designer to have a good understanding of the system properties. In [13], a general supervised learning framework that infers both the structure and the parameters of a formula is presented. The approach is based on lattice search and parameter synthesis, which makes it general, but inefficient. Using an efficient decision tree-based framework to learn STL formulae is explored in [14], [15], where the nodes of the tree contain simple formulae that are tuned optimally from a predefined set of primitives. In [16], the authors propose a systematic enumeration based method to

learn short, interpretable STL formulae. Other works about learning temporal logic formulae consider learning from positive examples only [3], clustering [4], active learning [17], and automata-based methods for untimed formulae [5], [6].

Most existing algorithms for learning STL formulae either do not achieve good classification performance for real-world applications, or do not provide any interpretability of the output formulae: they generate long and complicated specifications. To address these concerns, in this paper we introduce *Boosted Concise Decision Trees* (BCDTs) to learn STL formulae from labeled time-series data. To improve the classification accuracy of existing works, we use a boosting method to combine multiple models with weak classification power. The weak learning models are bounded-depth decision trees, called Concise Decision Trees (CDTs). Each CDT is a Decision Tree (DT) [18], empowered by a set of techniques called conciseness techniques, to generate simpler formulae and improve the interpretability of the final output. We also use a heuristic method in the BCDT algorithm to prune the ensemble of trees, which helps with the interpretability of the formulae. To relate STL and BCDTs, we establish a connection between boosted trees and weighted STL (wSTL) formulae [19], which have weights associated with Boolean and temporal operators. We show performance gains and improved interpretability of our method compared to literature, in naval surveillance and urban driving scenarios.

The main contributions of the paper are: (a) a novel inference algorithm based on boosted decision trees, which has better classification performance than related approaches, (b) a set of heuristic techniques to generate simple STL formulae from decision trees that improve interpretability, (c) two case studies in naval surveillance and urban-driving that highlight the classification performance and interpretability of our proposed learning algorithm.

## II. PRELIMINARIES

Let  $\mathbb{R}$ ,  $\mathbb{Z}$ ,  $\mathbb{Z}_{\geq 0}$  denote the sets of real, integer, and non-negative integer numbers, respectively. With a slight abuse of notation, given  $a, b \in \mathbb{Z}_{\geq 0}$  we use  $[a, b] = \{t \in \mathbb{Z}_{\geq 0} \mid a \leq t \leq b\}$ . The cardinality of a set is denoted by  $|\cdot|$ . A (discrete-time) signal  $s$  is a function  $s : [0, T] \rightarrow \mathbb{R}^n$  that maps each (discrete) time point  $t \in [0, T]$  to an  $n$ -dimensional vector of real values, where  $T \in \mathbb{Z}_{\geq 0}$ . Each component of  $s$  is denoted as  $s_j, j \in [1, n]$ .

Signal Temporal Logic (STL) was introduced in [9]. Informally, the STL formulae used in this paper are made of predicates  $\mu$  defined over components of real-valued signals in the form of  $\mu = s_j \sim \pi$ , where  $\pi \in \mathbb{R}$  is a threshold and  $\sim \in \{>, \leq\}$ , which are connected using Boolean operators, such as  $\neg, \wedge, \vee$ , and temporal operators, such as  $G_{[a,b]}$

\*This work was partially supported by the NSF under grants IIS-2024606 and IIS-1723995 at Boston University.

<sup>1</sup>Erfan Aasi, Calin Belta, and Mahroo Bahreinian are with Boston University, Boston, MA 02215, USA eaasi@bu.edu, cbelta@bu.edu, mahroobh@bu.edu

<sup>2</sup>Cristian Ioan Vasile is with Lehigh University, Bethlehem, PA 18015, USA cvasile@lehigh.edu

(always) and  $F_{[a,b]}$  (eventually). The semantics are defined over signals. For example, formula  $G_{[3,6]}s_3 \leq 1$  means that, for all times 3,4,5,6, component  $s_3$  of a signal  $s$  is less than or equal 1. STL has both qualitative and quantitative semantics. We use  $s \models \phi$  to denote Boolean satisfaction. The quantitative semantics is given by a robustness degree  $\rho(\phi, s)$  [20], which captures the degree of satisfaction of a formula  $\phi$  by a signal  $s$ . Positive robustness ( $\rho(\phi, s) \geq 0$ ) implies Boolean satisfaction  $s \models \phi$ , while negative robustness ( $\rho(\phi, s) < 0$ ) implies violation  $s \not\models \phi$ .

Weighted STL (wSTL) [19] is an extension of STL that has the same qualitative semantics as STL, but has weights associated with the Boolean and temporal operators, which modulate its robustness degree. In this paper, we restrict our attention to a fragment of wSTL with weights on conjunctions only. For example, the wSTL formula  $\phi_1 \wedge^\alpha \phi_2$ ,  $\alpha = (\alpha_1, \alpha_2) \in \mathbb{R}_{>0}^2$ , denotes that  $\phi_1$  and  $\phi_2$  must hold with priorities  $\alpha_1$  and  $\alpha_2$ . The priorities capture the satisfaction importance of their corresponding formulae.

Parametric STL (PSTL) [2] is an extension of STL, where the endpoints  $a, b$  of the time intervals in the temporal operators and the thresholds  $\pi$  in the predicates are parameters. The set of all possible valuations of all parameters in a PSTL formula  $\psi$  is called the parameter space and is denoted by  $\Theta$ . A particular valuation is denoted by  $\theta \in \Theta$  and the corresponding formula by  $\psi_\theta$ .

### III. PROBLEM FORMULATION

#### A. Motivating Example

Consider the maritime surveillance scenario from [13], [15] (see Fig. 1(a)). The goal is to detect anomalous vessel behaviors by looking at their trajectories. A vessel behaving normally approaches from the open sea and heads directly towards the harbor, while a vessel with anomalous behaviors either veers to the island and then heads to the harbor, or it approaches other vessels in the passage between the peninsula and the island and then returns to the open sea.

In the scenario's dataset [15], the signals are represented as 2-dimensional trajectories with planar coordinates  $(x(t), y(t))$ . The labels indicate the type of a vessel's behavior (normal or anomalous). In Fig. 1(b) and 1(c), we show the  $x$  and  $y$  components of some signals, respectively, over time. For better visualization, we show the signals over a part of their time horizon. In Fig. 1(b), one of the areas that distinguishes between positive (normal) and negative (anomalous) signals is the area between lines  $L1$  and  $L2$ , over the time interval  $t \in [15, 25]$ . By using STL classifiers, formula  $\phi_1 = F_{[15,25]}(x > 40) \wedge F_{[15,25]}(x \leq 47)$ , or even a simpler formula  $\phi'_1 = F_{[15,25]}((x > 40) \wedge (x \leq 47))$ , can be used to distinguish positive and negative signals in this area. Similarly, in Fig. 1(c), we can describe the separation area between lines  $L3$  and  $L4$  by the STL formula  $\phi_2 = F_{[12,20]}(y > 26) \wedge F_{[12,20]}(y \leq 32)$ , or even a simpler formula  $\phi'_2 = F_{[12,20]}((y > 26) \wedge (y \leq 32))$ . Considering the common time interval between the separation areas in Fig. 1(b) and Fig. 1(c), a shorter and easier to read formula  $\phi_3 = F_{[15,20]}((40 < x \leq 47) \wedge (26 < y \leq 32))$  can be used to distinguish between positive and negative signals in the  $x$ - $y$  space. In this paper, we seek techniques to generate simple

formulae, such as  $\phi_3$ , to classify signals without losing the classification accuracy.

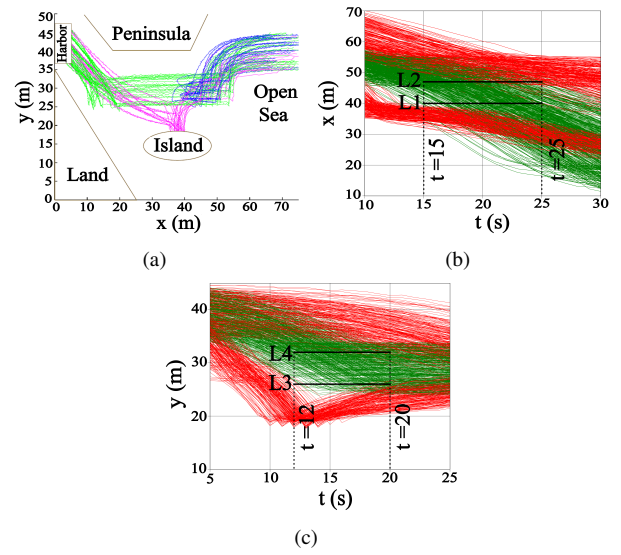


Fig. 1. (a) Naval surveillance scenario [13], where normal trajectories are shown in green, and anomalous signals are shown in blue and magenta, (b)  $x$  and (c)  $y$  components of naval trajectories. The green and red trajectories belong to the normal and anomalous behaviors, respectively.

#### B. Problem Statement

Let  $C = \{C_p, C_n\}$  be the set of possible (positive and negative) classes. We consider a labeled data set with  $N$  data samples as  $S = \{(s^i, \ell^i)\}_{i=1}^N$ , where  $s^i$  is the  $i^{\text{th}}$  signal and  $\ell^i \in C$  is its label.

**Problem 1:** Given a labeled data set  $S = \{(s^i, \ell^i)\}_{i=1}^N$ , find an STL formula  $\phi$  that minimizes the Misclassification Rate  $MCR(\phi)$  defined below:

$$2 \tag{1}$$

### IV. SOLUTION

We propose a solution to Pb. 1 based on BCDT method, presented in Sec. IV-A. BCDT grows multiple binary CDTs, inspired by AdaBoost [21] algorithm, where each CDT is a decision tree empowered by a set of conciseness techniques to generate simpler formulae. The construction method for a single CDT is explained in Sec. IV-B. We describe the meta parameters of the CDT method in Sec. IV-C, and in Sec. IV-D we explain the conciseness techniques and the connection with interpretability.

#### A. Boosted Concise Decision Trees Algorithm

The BCDT algorithm in Alg. 1 is inspired by the AdaBoost method [22]. AdaBoost combines weak classifiers with simple formulae, trained on weighted data samples. Weights of the data represent the difficulty of correct classification. After training a weak classifier, the weights of the correctly classified samples are decreased and weights of the misclassified samples are increased. The algorithm takes as input the labeled data set  $S$ , the number of learners (trees)  $K$ , and the weak learning model  $\mathcal{E}$ , which is the algorithm to construct CDTs (explained in Alg. 2). The CDTs are binary

decision trees, where formulae of the nodes are primitives (see Sec. IV-C) with general rectangular predicates  $\mu$  of the form  $As \leq b$ , with  $A = [\mathbb{I}_{n_1} \quad -\mathbb{I}_{n_2}]^T$ ,  $b \in \mathbb{R}^{n_1+n_2}$ ,  $\mathbb{I}_n$  as the  $n \times n$  identity matrix, and  $n_1, n_2 \in [0, n]$ .

---

**Algorithm 1** Boosted Concise Decision Trees (BCDT)

---

```

1: Input:  $S = \{(s^i, \ell^i)\}_{i=1}^N$ ,  $K$ ,  $\mathcal{E}$ 
2: Output: final classifier  $f_{BCDT}(\cdot)$ 
3: Initialize:  $\forall (s^i, \ell^i) \in S : D_1(s^i) = 1/|S|$ 
4: for  $k = 1, \dots, K$ :
5:   classifier  $f_{CDT}^k(\cdot) \leftarrow \mathcal{E}(S, D_k)$ 
6:    $\epsilon_k \leftarrow \sum_{(s^i, \ell^i) \in S} D_k(s^i) \cdot \mathbf{1}[\ell^i \neq f_{CDT}^k(s^i)]$ 
7:    $\alpha_k \leftarrow \begin{cases} \frac{1}{2} \ln(\frac{1}{\epsilon_k} - 1) & 0 < \epsilon_k \leq 1/2 \\ M & \epsilon_k = 0 \end{cases}$ 
8:    $D_{k+1}(s^i) \propto D_k(s^i) \exp(-\alpha_k \cdot \ell^i \cdot f_{CDT}^k(s^i))$ 
9:  $f_{BCDT}(\cdot) \leftarrow \begin{cases} \text{sign}(\sum_{k=1}^K \alpha_k \cdot f_{CDT}^k(\cdot)) & \alpha_k < M, \forall k \\ f_{CDT}^*(\cdot) & \text{otherwise} \end{cases}$ 
10: return  $f_{BCDT}(\cdot)$ 

```

---

In Alg. 1, initially all data samples are weighted equally (line 3). The algorithm iterates over the number of trees (line 4). At each iteration, the weak learning algorithm  $\mathcal{E}$  constructs a single CDT  $f_{CDT}^k(\cdot)$  based on data set  $S$  and current samples' weights  $D_k$  (line 5). Next, the misclassification error of the constructed tree  $\epsilon_k$  is computed (line 6). If the current tree has weak classification performance better than random guessing ( $0 < \epsilon_k \leq 1/2$ ), its weight is computed based on the original AdaBoost method, and if it has perfect classification performance and classifies all signals correctly ( $\epsilon_k = 0$ ), a big value  $M$  is assigned to its weight (line 7). At the end of each iteration, the samples' weights are updated and normalized (denoted by  $\propto$ ) based on the performance of the current tree, to focus on the misclassified signals in the next trees (line 8). To compute the final output of the algorithm, we use a heuristic method to prune the ensemble of trees, to generate simpler formulae and improve interpretability. Inspired by heuristic methods for pruning ensemble of decision trees in [18], [23], we compute the final output  $f_{BCDT}(\cdot)$  as (line 9): if the weights of all trees are less than  $M$ , the final output is computed as the weighted majority vote over all the CDTs (as in the AdaBoost method); otherwise, if there are one or more trees with weight  $M$ , the final output is computed by the tree with weight  $M$  that has the simplest STL formula, denoted by  $f_{CDT}^*(\cdot)$ . As a metric to compare the simplicity of formulae, the number of Boolean and temporal operators is considered. This pruning method helps with reducing the generalization error in the test phase and generating simpler formulae. We show its advantages with empirical results in Sec. V.

The final output  $f_{BCDT}(\cdot)$  assigns a label to each data sample. For simplicity, we abuse notation and consider  $C_p = 1$  and  $C_n = -1$ , such that  $f_{CDT}^k(\cdot) \in \{-1, 1\}$  for all  $k \in [1, K]$ . Note that one of the main assumptions in boosting methods is that each weak learner performs slightly better than random guessing (i.e., coin tossing). Therefore in Alg. 1, if any newly generated tree performs worse than random

guessing ( $\epsilon_k > 0.5$ ), we just discard it and generate another tree. An illustration of Alg. 1 is shown in Fig. 2.

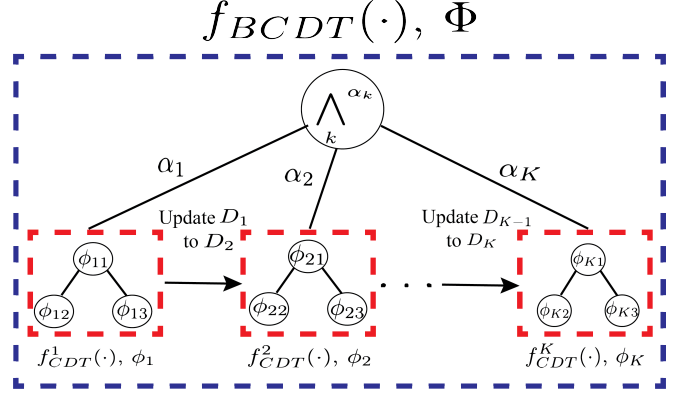


Fig. 2. Illustration of BCDT Alg. 1. The CDTs and their weights are used in construction of the final classifier  $f_{BCDT}(\cdot)$  in Alg. 1, and its corresponding formula  $\Phi$ . In this figure we have assumed  $\forall k \in [0, K] : \alpha_k < M$ .

We use the method from [15] to convert each CDT  $f_{CDT}^k(\cdot)$  to a corresponding STL formula  $\phi_k$ . The output of BCDT method is translated to a set of formulae and associated weights  $\{(\phi_k, \alpha_k)\}_{k=1}^K$ . The STL formula  $\Phi = \bigwedge_k \phi_k$  is the overall output formula; however, using wSTL [19] we express  $\Phi = \bigwedge_k^{\alpha_k} \phi_k$ , to capture the classification performance of each CDT.

### B. Construction of Concise Decision Tree

Decision Trees (DTs) [18], [24] are sequential decision models with hierarchical structures. In our algorithm, DTs operate on signals with the goal of predicting their labels. Inspired by [15], we present the Concise Decision Tree (CDT) method  $\mathcal{E}$  in Alg. 2, which extends the DT construction algorithm to CDTs, by applying conciseness techniques to generate simpler formulae (detailed in Sec. IV-D).

---

**Algorithm 2** Concise Decision Tree (CDT) method  $\mathcal{E}$

---

```

1: Meta-Parameters:  $\mathcal{P}, \mathcal{J}, stop$ 
2: Input:  $S, \phi^{path}, h, \phi_{parent}^c$ 
3: Output: sub-tree  $\mathcal{T}$ 
4: if  $stop(\phi^{path}, h, S)$  then
5:    $c = \mathcal{O}(S, \phi^{path}, \mathcal{P}, h)$ 
6:   return  $leaf(c)$ 
7:  $\mathcal{T} \leftarrow non\_terminal(\phi_{parent}^c)$ 
8:  $\phi^{new} = \phi^{path} \wedge \phi_{parent}^c$ 
9:  $S_{\top}, S_{\perp} \leftarrow partition(S, \phi^{new})$ 
10: for  $\otimes \in \{\top, \perp\}$  do
11:    $\phi_{\otimes}^c = \mathcal{O}(S_{\otimes}, \phi^{new}, \mathcal{P}, h+1)$ 
12:    $\phi_{\otimes}^{\otimes} = \mathcal{C}(\phi_{parent}^c, \phi_{\otimes}^c, S, \phi^{path}, h)$ 
13:   if  $\phi^{path} \wedge \phi_{\otimes}^{\otimes} \succeq_{\mathcal{J}} \phi^{new}$ :
14:     return  $\mathcal{E}(S, \phi^{path}, h, \phi_{\otimes}^{\otimes})$ 
15:  $\mathcal{T}.left \leftarrow \mathcal{E}(S_{\top}, \phi^{new}, h+1, \phi_{\top}^c)$ 
16:  $\mathcal{T}.right \leftarrow \mathcal{E}(S_{\perp}, \phi^{new}, h+1, \phi_{\perp}^c)$ 
17: return  $\mathcal{T}$ 

```

---

To limit the complexity of CDTs, we consider three meta-parameters in Alg. 2: (1) PSTL primitives  $\mathcal{P}$  capturing the

possible ways to split the data at each node, (2) impurity measures  $\mathcal{J}$  to select the best primitive at each node, and (3) stop conditions  $stop$  to limit the CDTs' growth. The meta-parameters are explained in details in Sec. IV-C.

Alg. 2 is recursive, and takes as input (1) the set of labeled signals  $S$  at the current node, referred to as *parent* node, (2) the path formula  $\phi^{path}$  from the root to the parent node, (3) the depth  $h$  from the root to the node, and (4) the candidate formula  $\phi_{parent}^c$  for the node. At the beginning, the stop conditions  $stop$  are checked (line 4). If they are satisfied (lines 5-6), a single leaf is returned that is marked with label  $c$ , according to the primitive optimization method in Alg. 3. Otherwise, a non-terminal node is created that is associated with the candidate formula  $\phi_{parent}^c$  (line 7). The formula  $\phi^{new}$  is the updated path formula from the root, considering the candidate primitive  $\phi_{parent}^c$  of the parent node (line 8). Next, the data set  $S$  is partitioned according to the new formula (line 9), where  $S_{\top}$  and  $S_{\perp}$  are the set of signals that satisfy and violate  $\phi^{new}$ , respectively.

Following the structure of the tree, first for the left child of the node ( $\otimes = \top$ ) and then for the right child ( $\otimes = \perp$ ), we follow these steps (line 10): first, the candidate primitive for the child  $\phi_{\otimes}^c$  is computed from the set  $\mathcal{P}$  (line 11). Then, by applying the conciseness method  $\mathcal{C}$  (explained in Sec. IV-D) on the combination of parent's candidate formula  $\phi_{parent}^c$  and the child's candidate primitive  $\phi_{\otimes}^c$ , we find a new formula  $\phi^{\otimes}$  (line 12) as a new candidate for the parent node. In line 13, the notation  $\succeq^{\mathcal{J}}$  is used to compare two formulae based on the impurity measure  $\mathcal{J}$ . If the impurity reduction of the new candidate formula  $\phi^{\otimes}$  is more than the previous candidate  $\phi_{parent}^c$ , the algorithm is repeated for the parent node, with  $\phi_{parent}^c$  replaced by  $\phi^{\otimes}$  (line 14). Note that the decision tree method in [15] is based on the idea of incremental impurity reduction at each node of the tree. Following the same idea, we argue that by applying the conciseness techniques at each node, if the impurity reduction of the new candidate formula is better than the previous one, the new candidate leads to a stronger classifier with a simpler specification. Finally, when there is no more possibility of applying the conciseness method on the parent node, we continue the construction of the tree for the left and right children (lines 15-16) and the sub-tree for the parent is returned (line 17).

---

### Algorithm 3 Parameterized Primitive Optimization $\mathcal{O}$

---

```

1: Meta-Parameters:  $\mathcal{J}, stop$ 
2: Input:  $S, \phi^{path}, prim, h$ 
3: Output: optimal primitive  $\phi^*$ 
4: if  $stop(\phi^{path}, h, S)$  then
5:    $\phi^* \leftarrow \arg \max_{c \in C} \{p(S, c; \phi^{path})\}$ 
6: else
7:    $\phi^* = \arg \max_{\substack{\psi \in prim, \theta \in \Theta}} \mathcal{J}(S, partition(S, \phi^{path} \wedge \psi_{\theta}))$ 
8: return  $\phi^*$ 

```

---

The parameterized primitive optimization method  $\mathcal{O}$ , presented in Alg. 3, finds the best primitive with optimal evaluation, from the input primitive set  $prim$ . This method has similar meta parameters as Alg. 2 and takes as input (1) the set of labeled signals  $S$  at the current node, (2) the

path formula  $\phi^{path}$  from the root to the current node, (3) a set of input primitives  $prim$ , and (4) the depth  $h$  from the root to the node. If the stop conditions are satisfied (line 4), a label  $c^* \in C$  is computed (line 5) according to the best classification quality, using the partition weight  $p(S, c; \phi^{path})$  of the impurity measure (see Sec. IV-C.2); otherwise, the best primitive from the input primitive set  $prim$  is computed by solving an optimization method based on the impurity measure  $\mathcal{J}$ .

### C. Meta Parameters

1) *PSTL primitives  $\mathcal{P}$* : The splitting rules at each node are simple PSTL formulae, called *primitives* [15]. Here we use *first-order primitives*  $\mathcal{P}_1$ :  $G_{[t_0, t_1]}(s_j \sim \pi), F_{[t_0, t_1]}(s_j \sim \pi)$ , where the decision parameters are  $(t_0, t_1, \pi)$ .

2) *Impurity measure  $\mathcal{J}$* : We use the Misclassification Gain (MG) impurity measure [18] as a criterion to select the best primitive at each node. Given a finite set of signals  $S$ , an STL formula  $\phi$ , and the subsets of  $S$  that are partitioned based on satisfaction of  $\phi$  as  $S_{\top}, S_{\perp} = partition(S, \phi)$ , we have  $MG(S, \{S_{\top}, S_{\perp}\}) = MR(S) - \sum_{\otimes \in \{\top, \perp\}} p_{\otimes} MR(S_{\otimes})$ , where  $MR(S) = \min(p(S, C_p; \phi), p(S, C_n; \phi))$ , and the  $p$  parameters are partition weights computed based on signals' labels and satisfaction of  $\phi$ . Here, we extend the robustness-based impurity measures in [15] to account for the sample weights  $D_k$  from the BCDT in Alg. 1. The boosted impurity measures are defined by the partition weights below

$$\begin{aligned}
p_{\otimes} &= \frac{\sum_{(s^i, \ell^i) \in S_{\otimes}} D_k(s^i) \cdot \rho(\phi, s^i)}{\sum_{(s^i, \ell^i) \in S} D_k(s^i) \cdot |\rho(\phi, s^i)|}, \quad \otimes \in \{\top, \perp\} \\
p(S, c; \phi) &= \frac{\sum_{(s^i, \ell^i) \in S, \ell^i = c} D_k(s^i) \cdot |\rho(\phi, s^i)|}{\sum_{(s^i, \ell^i) \in S} D_k(s^i) \cdot |\rho(\phi, s^i)|}
\end{aligned} \tag{2}$$

This formulation also works for other types of impurity measures, such as information and Gini gains [25].

3) *Stop Conditions*: There are multiple stopping conditions that can be considered for terminating Alg. 2. We stop the growth of trees either when they reach a given depth, or when  $\lambda$  percent of the signals belong to the same class. In our implementations, we set  $\lambda = 95\%$ .

### D. Conciseness

We propose the conciseness method  $\mathcal{C}$ , presented in Alg. 4, to improve the simplicity and interpretability of STL formulae. This algorithm takes as inputs the candidate primitive  $\phi_{parent}^c$  for the parent node, the candidate primitive for its child (either left or right child)  $\phi_{\otimes}^c, \otimes \in \{\top, \perp\}$ , the set of signals  $S$ , path formula  $\phi^{path}$ , and depth  $h$  of the parent node. The output of the algorithm is a new candidate primitive for the parent node, denoted by  $\phi_{new}^c$ .

First, the method constructs a new PSTL primitive for the parent node, denoted by  $\phi_{parent}$ , by combining the candidate primitives of the parent and the child nodes (line 3), where the combination operator is denoted by  $\Join$ . This is done by considering the possible ways to combine two candidate primitives, which we propose two heuristic techniques for it. Then, the optimal valuation of the new PSTL primitive is



computed by using the optimization method  $\mathcal{O}$  and the path formula  $\phi^{path}$  (line 4).

---

**Algorithm 4** Conciseness Method  $\mathcal{C}$

---

1: **Input:**  $\phi_{parent}^c, \phi_{\otimes}^c, S, \phi^{path}, h$   
2: **Output:** new candidate primitive  $\phi_{new}^c$   
3:  $\phi_{parent} = \phi_{parent}^c \sqcup \phi_{\otimes}^c$   
4:  $\phi_{new}^c = \mathcal{O}(S, \phi^{path}, \phi_{parent}, h)$   
5: **return**  $\phi_{new}^c$

---

The heuristic techniques to combine two primitives and generate shorter PSTL formulae are as following:

1) *Combination of Always operators:* If the candidate primitives of the parent and child nodes are as  $\phi_{parent}^c = G_{[t_0, t_1]}(\mu_{parent})$  and  $\phi_{\otimes}^c = G_{[t_2, t_3]}(\mu_{child})$ , respectively, we construct a new PSTL primitive  $\phi_{parent} = G_{[t_4, t_5]}((\mu_{parent}) \wedge (\mu_{child}))$  for their combination. For example, given  $\phi_{parent}^c = G_{[t_0, t_1]}((s_1 > \pi_1) \wedge (s_2 \leq \pi_2))$  and  $\phi_{\otimes}^c = G_{[t_2, t_3]}(s_2 > \pi_3)$ , the combined PSTL primitive is  $\phi_{parent} = G_{[t_4, t_5]}((s_1 > \pi_1) \wedge (\pi_3 < s_2 \leq \pi_2))$ .

2) *Combination of Eventually operators:* Similar to the combination of always operators, if the candidate primitives of the parent and child nodes are as  $\phi_{parent}^c = F_{[t_0, t_1]}(\mu_{parent})$  and  $\phi_{\otimes}^c = F_{[t_2, t_3]}(\mu_{child})$ , respectively, we construct a new PSTL primitive as  $\phi_{parent} = F_{[t_4, t_5]}((\mu_{parent}) \wedge (\mu_{child}))$ .

In Fig. 3, we provide an example of how Alg. 4 works.

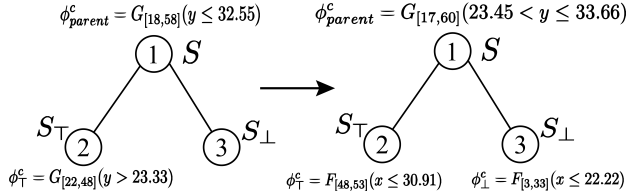


Fig. 3. Example of applying the conciseness method  $\mathcal{C}$  for the naval surveillance data set. On the left, the candidate primitive for the parent node 1 is  $\phi_{parent}^c = G_{[18,58]}(y \leq 32.55)$ , and after partitioning the signals, the candidate primitive for the left child (node 2) is computed as  $\phi_T^c = G_{[22,48]}(y > 23.33)$ . The conciseness method  $\mathcal{C}$  constructs a new candidate PSTL primitive  $\phi_{parent} = G_{[t_0, t_1]}(\pi_0 < y \leq \pi_1)$  for their combination, where its optimal valuation according to the primitive optimization method  $\mathcal{O}$  is  $\phi_T^c = G_{[17,60]}(23.45 < y \leq 33.66)$ . Due to the higher impurity reduction of  $\phi_T^c$  compared to  $\phi_{parent}^c$ , the new candidate  $\phi_T^c$  is chosen as  $\phi_{parent}^c$  for the parent node 1 on the right. The partitioning of the signals and the candidate primitives for the left and right children (nodes 2 and 3) are recomputed according to the new candidate primitive  $\phi_{parent}^c$ . According to the conciseness technique  $\mathcal{C}$ , there is no more possibility of combining the candidate primitives of the parent node and its children, because the temporal operators of the parent and its children nodes are different. Therefore, the procedure of CDT construction is followed up from the left and right children.

**Remark 1:** There are multiple ways of combining the primitives to improve interpretability. For example, given the candidate primitive the parent node as  $\phi_{parent}^c = G_{[t_0, t_1]}(\mu_{parent})$ , and the candidate primitive of its child as  $\phi_{\otimes}^c = F_{[t_2, t_3]}(\mu_{child})$ , we can construct a new PSTL primitive for the parent node as  $\phi_{parent} = F_{[t_4, t_5]}((\mu_{parent}) \cup_{[0, t_6]}(\mu_{child}))$ . We will explore the other ways of combining the primitives in the future works.

**Remark 2:** Note that our heuristic method combines two primitives whenever their combination improves the impurity

measure and classification performance. This leads to a larger set of primitives for constructing the trees, and it is more efficient compared to the naive approach of investigating all possible combinations of primitives.

### E. Complexity Analysis

We denote the lower and the two-sided asymptotic bounds for the complexity of the overall algorithm by  $\Omega(\cdot)$  and  $\Theta(\cdot)$ , respectively. The complexity of the BCDT algorithm (Alg. 1) is equivalent to the complexity of the AdaBoost method with  $K$  trees  $O(K \cdot C_{\mathcal{E}}(N))$ , where  $C_{\mathcal{E}}(N)$  is the complexity of constructing a CDT by Alg. 2, and  $N$  is the number of signals to be processed. Let  $C_{\mathcal{O}}(N)$  be the complexity of the optimization method in Alg. 3. Clearly we have  $C_{\mathcal{O}}(N) = \Omega(N)$ , because the method must at least check the labels of all signals [26]. The worst-case complexity of Alg. 2 is obtained when at each node the optimal partition has size  $(1, N - 1)$ , and we run the conciseness method (Alg. 4) for each child of the node, which leads to  $2C_{\mathcal{O}}(N)$ . Using the recursive nature of decision trees, the complexity analysis of [14] and the Akra-Bazzi method [26], for the worst-case and average-case complexity of  $C_{\mathcal{E}}(N)$  we have  $\Theta(N + 4 \sum_{k=2}^N C_{\mathcal{O}}(k))$  and  $\Theta(N \cdot (1 + 2 \int_1^N \frac{C_{\mathcal{O}}(u)}{u^2} du))$ , respectively.

## V. CASE STUDIES

We demonstrate the effectiveness and computational advantages of our method with two case studies. The first is the naval surveillance scenario from Sec. III-A. The second is an urban-driving scenario, implemented in the simulator CARLA [27]. We use Particle Swarm Optimization (PSO) method [28] for solving the optimization problems in Alg. 3. The parameters of the PSO method are tuned empirically. We use  $M = 100$  in our implementations. We run the case studies on a 3.70 GHz processor with 16 GB RAM.

### A. Naval Surveillance

We compare our inference algorithm with the methods from [15] (the DTL4STL tool) and [16]. The dataset is composed of 2000 signals, with 1000 normal and 1000 anomalous trajectories. Each signal has 61 timepoints (see Fig. 4(b) for some example trajectories). We test our algorithm with 5-fold cross validation and maximum depth = 3 for the trees (as in [15]). The results are provided in Table. I for different number of decision trees  $K$  in Alg. 1; TR-M(%) and TR-S(%) are the mean and standard deviation of the MCR in the training phase, respectively; TE-M(%) and TE-S(%) are the mean and standard deviation of the MCR in the test phase; R is the runtime, and CT is the number of times that by applying the conciseness method  $\mathcal{C}$  during the construction of CDTs, a simpler formula is found.

In Fig. 4(a), the classification performance of our framework is represented, with respect to different number of decision trees  $K$ . From this figure and Table. I it is clear that the best classification performance, over both training and test phases, is obtained with  $K = 3$ , where we find a set of concise trees that are able to classify all signals correctly in the test phase. Note that adding to the number of trees increases the complexity of the framework and leads

to capturing finer details of the dataset, which has the risk of overfitting, as the TE-M increases for  $K > 3$  (see Fig. 4(a)).

TABLE I

$K$	TR-M (%)	TR-S (%)	TE-M (%)	TE-S (%)	R	CT
1	0.36	0.35	0.95	0.97	11m 8s	4
2	0.34	0.21	0.55	0.33	30m 47s	14
3	0.01	0.02	0.00	0.00	33m 16s	10
4	0.05	0.10	0.10	0.12	61m 33s	29
5	0.01	0.02	0.10	0.20	81m 52s	33
6	0.00	0.00	0.05	0.10	85m 55s	38

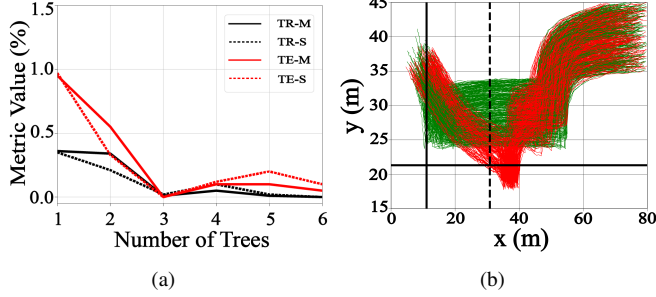


Fig. 4. (a) Classification performance of our framework for naval surveillance scenario, with different number of decision trees  $K$ . The best performance is obtained with  $K = 3$ , (b) Examples of trajectories from the naval surveillance case study. The green and red trajectories belong to normal and anomalous behaviors, respectively. For formula  $\Phi_1^{Naval}$ , the thresholds of the always and eventually operators are shown by solid and dashed black lines, respectively.

In the following, the learned formulae with  $K = 3$  are presented over all the folds. At each fold  $f$ , first the initial wSTL formula  $\Phi_f$  learned by Alg. 1 is presented, which is the weighted conjunction of three STL formulae; then by applying the heuristic technique from Alg. 1 for trees with weight  $M$ , the final output  $\Phi_f^{Naval}$  is presented:

- Fold 1:  $\Phi_1 = \phi_{11}^M \wedge \phi_{12}^{2.71} \wedge \phi_{13}^{2.88}$   
 $\Rightarrow \Phi_1^{Naval} = \phi_{11} = \phi_{111} \wedge \phi_{112}$   
 $\phi_{111} = F_{[28,53]}(x \leq 30.85)$   
 $\phi_{112} = G_{[2,26]}((y > 21.31) \wedge (x > 11.10))$
- Fold 2:  $\Phi_2 = \phi_{21}^{2.59} \wedge \phi_{22}^{0.80} \wedge \phi_{23}^M$   
 $\Rightarrow \Phi_2^{Naval} = \phi_{23} = \phi_{231} \wedge \phi_{232}$   
 $\phi_{231} = G_{[8,32]}(y > 23.15), \phi_{232} = F_{[20,55]}(x \leq 33.57)$
- Fold 3:  $\Phi_3 = \phi_{31}^M \wedge \phi_{32}^{2.49} \wedge \phi_{33}^{2.64}$   
 $\Rightarrow \Phi_3^{Naval} = \phi_{31} = \phi_{311} \wedge \phi_{312}$   
 $\phi_{311} = F_{[57,60]}(x \leq 33.91), \phi_{312} = G_{[7,51]}(y > 21.36)$
- Fold 4:  $\Phi_4 = \phi_{41}^{2.65} \wedge \phi_{42}^M \wedge \phi_{43}^M$   
 $\Rightarrow \Phi_4^{Naval} = \phi_{43} = \phi_{431} \wedge \phi_{432}$   
 $\phi_{431} = F_{[52,55]}(x \leq 35.34)$   
 $\phi_{432} = G_{[0,26]}((y > 22.20) \wedge (x > 11.73))$
- Fold 5:  $\Phi_5 = \phi_{51}^{2.53} \wedge \phi_{52}^M \wedge \phi_{53}^{1.83}$   
 $\Rightarrow \Phi_5^{Naval} = \phi_{52} = \phi_{521} \wedge \phi_{522}$   
 $\phi_{521} = G_{[9,44]}(y > 23.43), \phi_{522} = F_{[57,59]}(x \leq 33.10)$

Note that there are some similarities between the inferred formulae in different folds; for example, the structures of the formulas  $\phi_{112}$  and  $\phi_{432}$  are the same and their thresholds and

time bounds are really close to each other. Also it is worth to mention that in the fourth fold, although both formulae  $\phi_{42}$  and  $\phi_{43}$  have weight  $M$ , formula  $\phi_{43}$  is chosen over  $\phi_{42}$  because  $\phi_{43}$  has less number of Boolean and temporal operators (see Sec. IV-A). The output formulae of each fold are simple and easy to interpret. For example, from the plain English translation of formula  $\Phi_1^{Naval}$ , the behavior of normal vessels is interpreted as: "Normal vessel's  $x$  and  $y$  coordinates are bigger than 11.10 and 21.31m, respectively, over the time interval  $[2, 26]$ , and their  $x$  coordinate gets less than or equal to 30.85m, at some timepoint in the time interval  $[28, 53]$ ". The thresholds of the formula  $\Phi_1^{Naval}$  are shown in Fig. 4(b).

In [15], using first-order primitives and maximum tree depth of 3, the authors get a MCR with mean 1.3% and standard deviation 0.28% for this data set. To provide a fair comparison, we ran the algorithm from [15] on the same computer that we used for our algorithm and for the same data set. We obtained a MCR with mean 1.5% and standard deviation 0.5% in the test phase, with total runtime of 33 seconds. An example formula learned in one of the folds using the method from [15] is:

$$\begin{aligned}
 &(\phi_1 \wedge \phi_2) \vee (\neg \phi_1 \wedge ((\phi_3 \wedge \phi_4) \vee (\neg \phi_3 \wedge \phi_5))) \\
 &\phi_1 = G_{[39,60]}(x \leq 19.5), \quad \phi_2 = F_{[11,38]}(x > 41.2) \\
 &\phi_3 = G_{[20,59]}(y < 32.3), \quad \phi_4 = G_{[59,60]}(x \leq 39.2) \\
 &\phi_5 = G_{[20,53]}(y > 29.7)
 \end{aligned}$$

Compared to [15], our algorithm obtains a better classification performance, in addition to simpler and more interpretable formulae, at the cost of higher runtime due to the boosting and conciseness techniques. In [16], the authors obtain a MCR with mean 5% in test phase and total runtime of 45 minutes and the formula learned in their work is  $(y \geq 19.74)U_{[0,9.84]}(x \leq 24.86)$ . From the interpretability view, both the formulae learned by our algorithm and by [16] are simple and easy to interpret and both methods have roughly similar runtime, but our algorithm has noticeably better classification performance.

## B. Urban Driving

Consider an autonomous vehicle (referred to as *ego*) driving in an urban environment shown in Fig. 5. The scenario also contains a pedestrian and another car, which is assumed to be driven by a "reasonable" human who obeys traffic laws. Ego and the other car are in different, adjacent lanes, moving in the same direction. The cars move uphill in the  $y-z$  plane of the coordinate frame, towards positive  $y$  and  $z$  directions, with no lateral movement in the  $x$  direction. The accelerations of the cars are constant, and smaller for ego.

The positions and accelerations of the cars are initialized such the other car is always ahead of ego. The vehicles are headed towards an intersection without any traffic lights. There is an unmarked cross-walk at the end of the road before the intersection. When the pedestrian crosses the street, the other car brakes to stop before the intersection. If the pedestrian does not cross, the other car keeps moving without decreasing its velocity. Ego does not have a clear line-of-sight to the pedestrian crossing at the intersection,

because of the other car and the uphill shape of the road. The goal is to develop a method allowing ego to infer whether a pedestrian is crossing the street by observing the behavior (e.g., relative position and velocity over time) of the other car.

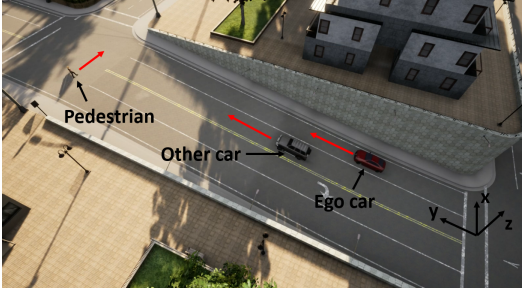


Fig. 5. Urban driving scenario implemented in the simulator CARLA [27]

The simulation of this scenario ends whenever ego gets closer than  $8m$  to the intersection. We assume that labeled behaviors (relative distances and velocities) are available, where the labels indicate whether a pedestrian is crossing or not. We collected 300 signals with 500 uniform time-samples per trace, where 150 were with and 150 without pedestrians crossing the street (see Fig. 6(b) and 6(c)). The dataset is available in [29]. We evaluate our algorithm with 5-fold cross-validation and maximum depth = 2 for the trees. The results are shown in Table II for different values of  $K$ .

TABLE II

$K$	TR-M (%)	TR-S (%)	TE-M (%)	TE-S (%)	R	CT
1	0.00	0.00	1.00	1.33	7m 10s	2
2	0.00	0.00	0.67	0.82	9m 57s	2
3	0.00	0.00	0.33	0.66	14m 52s	1
4	0.00	0.00	0.00	0.00	24m 40s	3
5	0.00	0.00	1.00	1.33	24m 49s	3
6	0.00	0.00	0.33	0.66	32m 52s	3

The classification performance of our framework is shown in Fig. 6(a), for different number of decision trees  $K$ , and the best performance is obtained with  $K = 4$ . In the following the learned formulae with  $K = 4$  are presented, by following the same notation as naval surveillance scenario in Sec. V-A:

- Fold 1:  $\Phi_1 = \phi_{11}^M \wedge \phi_{12}^{2.74} \wedge \phi_{13}^M \wedge \phi_{14}^M$   
 $\Rightarrow \Phi_1^{Urban} = \phi_{13} = \phi_{131} \wedge \phi_{132}$   
 $\phi_{131} = F_{[463,499]}(y \leq 8.78)$ ,  $\phi_{132} = G_{[477,481]}(v_y > 8.01)$
- Fold 2:  $\Phi_2 = \phi_{21}^{2.65} \wedge \phi_{22}^M \wedge \phi_{23}^M \wedge \phi_{24}^M$   
 $\Rightarrow \Phi_2^{Urban} = \phi_{23} = \phi_{231} \wedge \phi_{232}$   
 $\phi_{231} = F_{[463,488]}(z \leq 1.40)$ ,  $\phi_{232} = G_{[488,493]}(v_z > 1.19)$
- Fold 3:  $\Phi_3 = \phi_{31}^M \wedge \phi_{32}^M \wedge \phi_{33}^M \wedge \phi_{34}^M$   
 $\Rightarrow \Phi_3^{Urban} = \phi_{32}$   
 $\phi_{32} = F_{[370,485]}((y \leq 14.01) \wedge (v_y > 7.45))$
- Fold 4:  $\Phi_4 = \phi_{41}^{3.03} \wedge \phi_{42}^M \wedge \phi_{43}^M \wedge \phi_{44}^M$   
 $\Rightarrow \Phi_4^{Urban} = \phi_{44} = \phi_{441} \wedge \phi_{442}$   
 $\phi_{441} = F_{[474,486]}(y \leq 9.47)$   
 $\phi_{442} = G_{[476,496]}((v_z > 1.03) \wedge (y > 2.65))$
- Fold 5:  $\Phi_5 = \phi_{51}^M \wedge \phi_{52}^M \wedge \phi_{53}^{2.47} \wedge \phi_{54}^M$

$$\Rightarrow \Phi_5^{Urban} = \phi_{51}$$

$$\phi_{51} = F_{[384,469]}((y \leq 49.80) \wedge (v_y > 9.13))$$

Notice that the main objective of this scenario is to infer whether a pedestrian is crossing the street, based on the behavior of the other car when it gets close to the intersection. Hence, we expect the desired specifications to be short and they reason over the signals at time intervals close to the end of the simulation. This conforms to the time intervals of our inferred formulae and the fact that at each fold, we have trees with perfect classification in training phase (weight  $M$ ). The output formulae of our method are simple and easy to understand. For example,  $\Phi_3^{Urban}$  states that there is a pedestrian crossing the street, if "at some timepoint in the time interval  $[370, 485]$ , the vehicles get closer than  $14.01m$  in the  $y$ -direction, and the  $y$  component of ego's velocity gets bigger than the corresponding component of other car by  $7.45m/s$ ". This simply means that the other car is stopped at the intersection, because a pedestrian is crossing it, and ego is getting close to the other car; therefore, in the  $y$ -direction, the relative distance gets smaller and the velocity of ego gets bigger than the other car. The thresholds of formula  $\Phi_3^{Urban}$  are shown in Fig. 6(b) and 6(c).

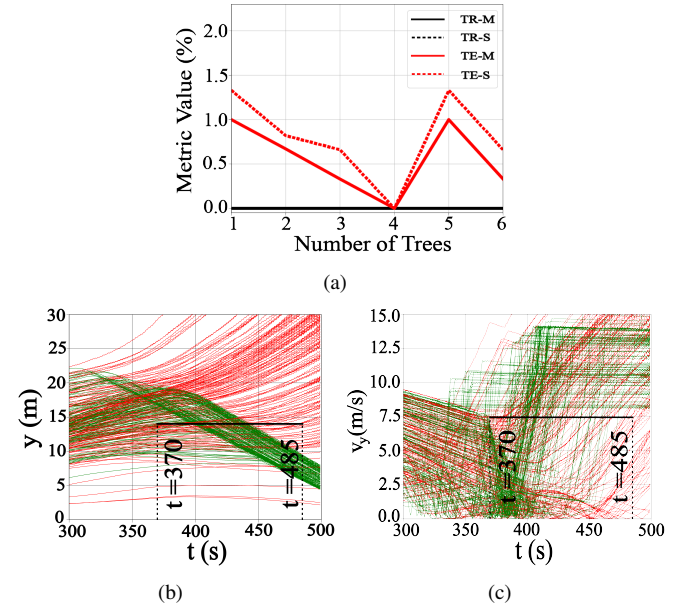


Fig. 6. (a) Our framework's classification performance for urban driving scenario, with different number of decision trees  $K$ . The best performance is obtained with  $K = 4$ , (b) and (c) the  $y$  component of relative distance and relative velocity between ego and the other vehicle, respectively, over time. The green and red signals belong to the cases when there is a pedestrian crossing the street and when there is no pedestrian crossing, respectively. The thresholds and time bound of the formula  $\Phi_3^{Urban}$  are shown by solid black lines.

To provide a fair comparison, we evaluate the performance of the algorithm from [15] on the same data set and on the same computer that is used for the algorithm developed in this paper. For the algorithm in [15], with first-order primitives, 5-fold cross validation and maximum depth of 2 for the trees, we obtained a mean MCR of 1% with standard deviation 1.5% in the test phase, with total runtime of 7.72 seconds. An example formula learned in one of the folds using the method from [15] is  $F_{[474,499]}(z <$

$1.2) \wedge F_{[0,499]}(v_y > 8.97)$ . The results show that our inferred formulae either have the same structure or are simpler than the formulae inferred by [15]. Moreover, our method achieves better classification performance than the algorithm in [15], at the cost of higher execution time.

## VI. CONCLUSION

In this paper, we propose a novel method for two-class classification of time-series data. Our algorithm grows an ensemble of decision trees that are empowered by conciseness techniques, to improve the interpretability of the formulae. The classification and interpretability advantages of our algorithm are evaluated on naval surveillance and urban-driving case studies, and are compared with two algorithms from literature. In future works, we will investigate alternate ways of achieving a tradeoff between formula conciseness and MCR performance, with faster execution time. Moreover, we will consider the STL inference from signals with heterogeneous time lengths.

## REFERENCES

- [1] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.
- [2] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *International Conference on Runtime Verification*. Springer, 2011, pp. 147–160.
- [3] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar, "Telex: learning signal temporal logic from positive examples using tightness metric," *Formal Methods in System Design*, vol. 54, no. 3, pp. 364–387, 2019.
- [4] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia, "Logical clustering and learning for time-series data," in *International Conference on Computer Aided Verification*, 2017, pp. 305–325.
- [5] D. Neider and I. Gavran, "Learning linear temporal properties," in *Formal Methods in Computer Aided Design*. IEEE, 2018, pp. 1–10.
- [6] Z. Xu, M. Ornik, A. A. Julius, and U. Topcu, "Information-guided temporal logic inference with prior knowledge," in *American Control Conference*, 2019, pp. 1891–1897.
- [7] A. Ketenci and E. A. Gol, "Synthesis of monitoring rules via data mining," in *American Control Conference*, 2019, pp. 1684–1689.
- [8] R. Yan and A. Julius, "Neural network for weighted signal temporal logic," *arXiv preprint arXiv:2104.05435*, 2021.
- [9] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.
- [10] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, 2015.
- [11] 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*, vol. 20, no. 1, pp. 79–93, 2018.
- [12] A. Bakhirkin, T. Ferrère, and O. Maler, "Efficient parametric identification for stl," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, 2018, pp. 177–186.
- [13] 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, 2016.
- [14] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A decision tree approach to data classification using signal temporal logic," in *Hybrid Systems: Computation and Control*, 2016, pp. 1–10.
- [15] G. Bombara and C. Belta, "Offline and online learning of signal temporal logic formulae using decision trees," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 3, pp. 1–23, 2021.
- [16] S. Mohammadinejad, J. V. Deshmukh, A. G. Puranic, M. Vazquez-Chanlatte, and A. Donzé, "Interpretable classification of time-series data using efficient enumerative techniques," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, 2020, pp. 1–10.
- [17] A. Linard and J. Tumova, "Active learning of signal temporal logic specifications," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 779–785.
- [18] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.
- [19] N. Mehdipour, C.-I. Vasile, and C. Belta, "Specifying user preferences using weighted signal temporal logic," *IEEE Control Systems Letters*, 2020.
- [20] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.
- [21] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.
- [22] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [23] V. Y. Kulkarni and P. K. Sinha, "Pruning of random forest classifiers: A survey and future directions," in *2012 International Conference on Data Science & Engineering (ICDSE)*. IEEE, 2012, pp. 64–68.
- [24] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 2007.
- [25] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 35, no. 4, pp. 476–487, 2005.
- [26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
- [27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *preprint arXiv:1711.03938*, 2017.
- [28] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.
- [29] E. Aasi, "Unmarked crosswalk carla scenario," [https://github.com/erfanaasi/unmarked\\_crosswalk\\_carla\\_scenario](https://github.com/erfanaasi/unmarked_crosswalk_carla_scenario), GitHub repository, 2022, [Online; accessed 1-March-2022].