Time-Incremental Learning of Temporal Logic Classifiers Using Decision Trees

Erfan Aasi EAASI@BU.EDU
Boston University, Boston, MA 02215, USA

Mingyu Cai MIC221@LEHIGH.EDU
Lehigh University, Bethlehem, PA 18015, USA

Cristian-Ioan Vasile CVASILE@LEHIGH.EDU
Lehigh University, Bethlehem, PA 18015, USA

Calin Belta CBELTA@BU.EDU
Boston University, Boston, MA 02215, USA

Editors: N. Matni, M. Morari, G. J. Pappas

Abstract

Real-time and human-interpretable decision-making in autonomous systems is a significant but challenging task, which usually requires predictions of possible future events from limited data. While machine learning techniques have achieved promising results in this field, they lack interpretability and the ability to make online predictions for sequential behaviors. In this paper, we introduce a time-incremental learning framework to predict the labels of time-series signals that are received incrementally over time, referred to as prefix signals. These signals are being observed as they are generated, and their time lengths are shorter than their corresponding time horizons. We present a novel decision tree-based approach to learn a finite number of Signal Temporal Logic (STL) specifications from a given dataset and construct a predictor based on them. Each STL specification serves as a binary classifier of the time-series data and captures a specific part of the dataset's temporal properties over time. The predictor is built by assigning time-variant weights to the STL formulas, which represent their classification impacts. The weights are learned using neural networks to minimize the misclassification rate of classifying prefix signals with different time lengths. The predictor is then used to predict the labels of prefix signals by computing the weighted sum of their robustnesses with respect to the STL formulas. The effectiveness and classification performance of our algorithm is evaluated on urban-driving and naval-surveillance case studies.

Keywords: Machine Learning, Formal Methods, Decision Trees

1. Introduction

Real-time decision-making for robotic tasks is a challenging problem, which usually requires prediction of possible outcomes in an incremental manner, based on available partial signals over time. The accuracy of such incremental predictions determines the efficiency of mitigating the occurrence of undesired behaviors. For example, consider the naval surveillance scenario (Fig. 1) from Kong et al. (2016). A vessel with normal behavior approaches from the open sea and heads directly towards the harbor, while a vessel behaving anomalously either veers to the island and then heads to the harbor, or it approaches the other vessels in the passage between the peninsula and the island and then returns to the open sea.

Predicting the behavior label of the vessels on the fly is of interest for mitigating anomalous behaviors and / or minimizing casualties. Suppose the time horizon of the scenario is T. In this paper, we focus on a time-incremental learning framework, where a dataset $S = \{s^i, \ell^i\}_{i=1}^N$, consisting of N signals s^i with time length T and their corresponding labels ℓ^i are provided. For the example introduced above, the signals are the recorded trajectories of the vessels over time (e.g., $x(t)$ and $y(t)$) and the labels indicate normal and anomalous behaviors. Our goal is to develop a method to classify (predict) the real-time behavior of a vessel represented by a prefix signal $s[0{:}t], 1 \leq t \leq T$. Motivated by the interpretability and readability of temporal logics Clarke et al. (1986), we use Signal Temporal Logic (STL) Maler and Nickovic (2004) formulas as classifiers.

There is a rich literature on integrating formal methods with machine learning (ML) to express classifiers of time-series data as temporal logic formulas Bartocci et al. (2014), Mohammadinejad et al. (2020b), Bombara et al. (2016), Xu et al. (2019), Hoxha et al. (2018), Jha et al. (2019), Ketenci and Gol (2019), Jin et al. (2015), Neider and Gavran (2018), Aasi et al. (2022). Initial attempts in this area focused on finding optimal parameters for fixed formula structures Bakhirkin et al. (2018), Bartocci et al. (2015) Asarin et al. (2011), Hoxha et al. (2018), Jin et al. (2015). Learning both formula structures and their parameters has been addressed in supervised classification methods. The work in Kong et al. (2016) is based on lattice search techniques, while Bombara et al. (2016) uses decision tree algorithms. The other related approaches are in different inference frameworks Yan and Julius (2021), Vazquez-Chanlatte et al. (2017), Bombara and Belta (2017), Baharisangari et al. (2021), Yan et al. (2019), Mohammadinejad et al. (2020a), Linard and Tumova (2020). None of these existing works focus on predicting the label of prefix signals. An intuitive solution to this problem is to apply an offline supervised learning method to the given dataset, learn an STL formula, and construct a monitor Maler and Nickovic (2004). However, the Boolean or quantitative output of a monitor could be inconclusive for the prefix signals with time lengths shorter than the horizon of the learned STL formula.

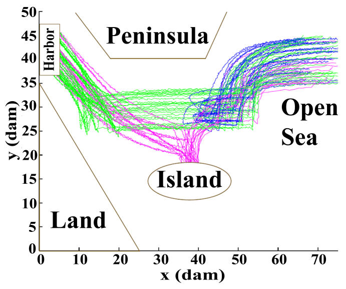

Figure 1: Schematic of the naval surveillance scenario Kong et al. (2016). Normal trajectories are shown in green, and the anomalous ones are in blue and magenta.

We propose a novel framework to predict the label of prefix signals with different time lengths. Our framework consists of three main parts: First, we analyze the signals and apply a heuristic method to find a finite number of time points along the horizon of the signals, called *decision times*, which are potentially informative for the classification of prefix signals into two classes. Then, for each decision time, we generate a decision tree-based classifier Breiman et al. (1984); Ripley (2007) and describe it by an STL specification. Each STL formula captures the temporal properties of the prefix signals with time horizons less than the corresponding decision time. Finally, we assign a time-variant weight distribution to the learned STL formulas using NNs, the weighted conjunction of the STL formulas is interpreted as a weighted STL (wSTL) formula Mehdipour et al. (2020), and the predictor is constructed based on that.

The main contributions of the paper are as follows: (a) formulation of the time-incremental learning problem, where a dataset of signals and their labels is given and the goal is to develop a method to predict the label of prefix signals that are received incrementally over time; (b) interpretable classification solution to the time-incremental learning problem based on STL, decision trees, and neural networks; (c) study of the classification and prediction performance of the proposed framework on urban-driving and naval-surveillance case studies.

2. Preliminaries

Let $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{Z}_{\geq 0}$ represent the sets of real, integer, and non-negative integer numbers, respectively. Given $a, b \in \mathbb{Z}_{\geq 0}$, we abuse the notation and use $[a, b] = \{t \in \mathbb{Z}_{\geq 0} \mid a \leq t \leq b\}$. A discrete-time signal s with time horizon $T \in \mathbb{Z}_{\geq 0}$ is a function $s : [0, T] \to \mathbb{R}^n$ that maps each discrete time point

$t \in [0, T]$ to an n-dimensional vector $s(t)$ of real values. We denote the components of signal s as $s_j, j \in [1, n]$, and the prefix of s up to time point t by $s[0{:}t]$. Let $\ell \in C = \{C_p, C_n\}$ denote the label of a signal s, where C_p and C_n are the labels for the positive and negative classes, respectively. We consider a labeled dataset with N data samples as $S = \{s^i, \ell^i\}_{i=1}^N$, where s^i is the i^{th} signal and $\ell^i \in C$ is its corresponding label. A *prefix dataset* $S[0 : t_k]$ *with horizon* t_k, is the dataset consisting of prefix signals with horizon t_k and their labels, denoted by $\{s^i[0 : t_k], \ell^i\}_{i=1}^N$. The cardinality of a set is shown by $|\cdot|$, the empty set is denoted by $\varnothing$, and a vector of zeros is denoted by $\emptyset$ (the dimension should be clear from the context).

Signal Temporal Logic (STL): STL was introduced in Maler and Nickovic (2004) to handle real-valued, dense-time signals. Informally, the STL specifications used in this paper are made of predicates defined over signal components in the form of $s_j(t) \sim \pi$, where $\pi \in \mathbb{R}$ is threshold and $\sim \in \{\geq, <\}$, which are connected using Boolean operators, such as $\neg$ (*negation*), $\wedge$ (*conjunction*), $\vee$ (*disjunction*), and temporal operators, such as $G_{[a,b]}$ (*always*) and $F_{[a,b]}$ (*eventually*). The semantics of STL are defined over signals. For example, formula $\phi_1 = G_{[2,5]} s_1 < 4$ means that, for all times 2,3,4,5, component s_1 of a signal s is less than 4, while formula $\phi_2 = F_{[3,10]} s_2 \geq 6$ expresses that at some time between 3 and 10, $s_2(t)$ becomes larger than or equal to 6.

STL has both qualitative (Boolean) and quantitative semantics. We denote Boolean satisfaction of a formula ϕ at time t by $s(t) \models \phi$. For the quantitative semantics, the robustness degree Donzé and Maler (2010), Fainekos and Pappas (2009), denoted by $\rho(\phi, s, t)$, captures the degree of satisfaction of a formula ϕ at time t by a signal s. For simplicity of notation, we use $s \models \phi$ and $\rho(\phi, s)$ as short for $s(0) \models \phi$ and $\rho(\phi, s, 0)$, respectively. Boolean satisfaction $s \models \phi$ corresponds to non-negative robustness ($\rho(\phi, s) \geq 0$), while violation corresponds to negative robustness ($\rho(\phi, s) < 0$). The minimum amount of time required to decide the satisfaction of a STL formula ϕ is called its horizon, and is denoted by $hrz(\phi)$. For example, the horizons of the two example formulas ϕ_1 and ϕ_2 given above are 5 and 10, respectively.

Parametric STL (PSTL): PSTL Asarin et al. (2011) is an extension of STL, where the threshold π in the predicates and the endpoints a and b of the time intervals in the temporal operators are parameters. A specific valuation of a PSTL formula ψ under the parameter values $\theta \in \Theta$ is denoted by ψ_θ, where Θ is the set of all possible valuations of the parameters.

Weighted STL (wSTL): wSTL Mehdipour et al. (2020) is another extension of STL with the same qualitative semantics as STL, but its robustness degree is modulated by the weights associated with the Boolean and temporal operators. In this paper, we focus on a fragment of wSTL, with weights on conjunctions only.

3. Problem Statement

First, we provide some definitions used in the problem formulation.

Definition 1 (Predictor) *The predictor* $P_\Phi(s[0{:}t]) = \tilde{\ell}(t) \in C$ *is a function that maps prefix signal* $s[0{:}t]$ *to a label* $\tilde{\ell}(t)$*, which represents the satisfaction prediction of* $s[0{:}t]$ *at time t with respect to the STL formula* Φ*:* $P_\Phi(s[0{:}t]) = C_p$*, if* $s[0{:}t] \models \Phi$*; otherwise,* $P_\Phi(s[0{:}t]) = C_n$*.*

Definition 2 (Timepoint MisClassification Rate (TMCR)) *Let* $P_\Phi(s^i[0{:}t]) = \tilde{\ell}^i(t)$*. The Timepoint MisClassification Rate* $TMCR(P_\Phi, t)$ *of predictor* P_Φ *at time step t is defined as:*

$$\frac{1}{N} |\{s^i[0 : t] \mid (\tilde{\ell}^i(t) = C_p \wedge \ell^i = C_n) \vee (\tilde{\ell}^i(t) = C_n \wedge \ell^i = C_p)\}|$$

Definition 3 (Incremental MisClassification Rate (IMCR)) *The Incremental MisClassification Rate of predictor* P_Φ*, denoted by* $IMCR(P_\Phi)$*, is the vector of TMCR values over the time horizon* T*, i.e.,* $[TMCR(P_\Phi, 0), TMCR(P_\Phi, 1), ..., TMCR(P_\Phi, T)]$.

We are now ready to formulate the main problem that we consider in this paper:

Problem 1 *Given a labeled data set $S = \{(s^i, \ell^i)\}_{i=1}^N$, find an STL formula Φ and its corresponding predictor P_Φ such that $IMCR(P_\Phi)$ is minimized.*

4. Solution

Our approach to Pb. 1 is illustrated in Fig. 2. It consists of three main components: (i) Signal Analysis, described in Sec. 4.1, applies a heuristic signal analysis method on the given dataset to find a finite number of potentially informative timepoints, referred as decision times and denoted by the set $\mathcal{T} = \{t_k\}_{k=1}^K$; (ii) Classifier Learning, described in Sec. 4.2, generates an STL formula for each decision time t_k in $\mathcal{T}$, using a decision tree method. This part learns both the structure of the STL formula and its parameters. The set of generated STL formulas is denoted by $F = \{\phi_k\}_{k=1}^K$; (iii) Classifier Evaluation, explained in Sec. 4.3, assigns a time-dependent weight distribution to the STL formulas in F, based on NNs. The weights capture the prediction performance of each STL formula over time. The output of this part is the wSTL formula $\Phi = \bigwedge_k^{\omega_k(t)} \phi_k$ and a predictor P_Φ based on that.

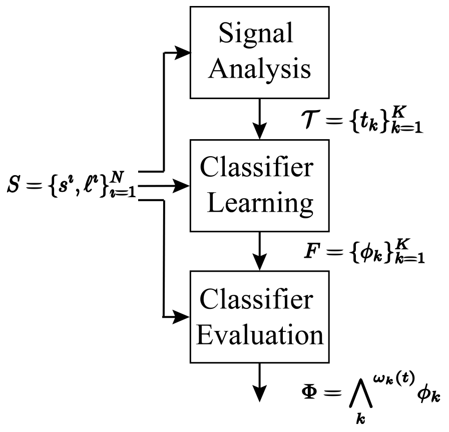
Signal Analysis
$\mathcal{T} = \{t_k\}_{k=1}^K$
$S = \{s^i, \ell^i\}_{i=1}^N$
Classifier Learning
$F = \{\phi_k\}_{k=1}^K$
Classifier Evaluation
$\Phi = \bigwedge_k^{\omega_k(t)} \phi_k$

Figure 2: Schematic of our framework.

4.1. Signal Analysis

Given the dataset S, the role of the Signal Analysis part is to analyze the signals of the dataset over time and find a finite number of timepoints, referred to as decision times. The decision times, denoted by t_k, are the timepoints that are potentially informative for classifying the prefix dataset $S[0 : t_k]$ into two classes, and they are considered as candidate timepoints for generating classifiers. Here we propose a metric to formulate an optimization problem to find the decision times.

A commonly used distance function for the case of multi-dimensional time-series data is the Euclidean distance Bombara and Belta (2017). For two signals s^1 and s^2, the Euclidean distance is defined as $d^2(s^1, s^2) = \sum_{j=1}^n \sum_{t=0}^T (s_j^1(t) - s_j^2(t))^2$. In this work, we extend it to define the *time-dependent Euclidean distance* as $d^2(s^1, s^2, t) = \sum_{j=1}^n (s_j^1(t) - s_j^2(t))^2$, $\forall t \in \{0, ..., T\}$. In particular, given the dataset S, the set of signals with positive labels are indexed by $h \in \{1, ..., N_p\}$ and with negative labels by $g \in \{1, ..., N_n\}$, respectively, with $N_p + N_n = N$. Thus, we can formulate the *positive-negative distance* in the dataset S as $d_{pn}^2(t) = \sum_{j=1}^n \sum_{h=1}^{N_p} \sum_{g=1}^{N_n} (s_j^h(t) - s_j^g(t))^2$, $\forall t \in \{0, ..., T\}$.

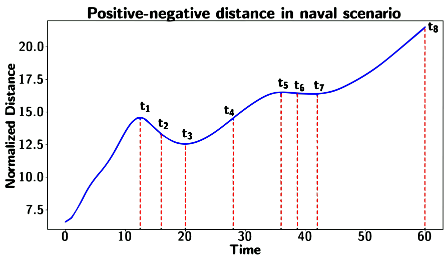

Figure 3: Positive-negative distance in the naval scenario.

To find proper time steps as the decision times, we use d_{pn}^2 as an optimization metric to evaluate the separation between the positive and negative labeled prefix signals. We compute the decision times as the informative timepoints where the first- or second-order discrete derivatives of d_{pn}^2 are zero. The intuition is that at these timepoints, the positive and negative signals are locally at the furthest or the closest distance from each other (first-order derivatives), or they are the switching timepoints for the evolution of the positive-negative distance over time (second-order derivatives). We also consider the horizon T of signals as a decision time. The set of decision times is denoted by $\mathcal{T} = \{t_k\}_{k=1}^{K}$, where $\forall k \in \{1, ..., K\} : t_k \in \mathbb{Z}_{\geq 0}$, $1 \leq t_k \leq T$.

Note that a trivial solution to Pb. 1 is to generate classifiers at every timepoint along the horizon of the signals, which is obviously inefficient and computationally expensive. We compare the efficiency and the prediction accuracy of this trivial solution with our method in Sec. 5.

Example 1 *In Fig. 3, the positive-negative distance is depicted over time for the naval surveillance case study (Sec. 1). The Signal Analysis part finds 8 decision times $\mathcal{T} = \{t_k\}_{k=1}^{8}$, where t_1, t_3, t_5 and t_7 are the timepoints that the first-order discrete derivatives are zero, t_2, t_4 and t_6 are the zero second-order discrete derivatives, and t_8 is the horizon T of the signals.*

4.2. Classifier Learning

The next component, Classifier Learning, takes as input the set of decision times $\mathcal{T}$ from the Signal Analysis part, in addition to the dataset S. Classifier Learning is responsible for generating classifiers at each decision time t_k on the prefix dataset $S[0 : t_k]$. To provide interpretable specifications for the classifiers and inspired by Bombara et al. (2016), we use the decision tree method $\mathcal{E}$ in Alg. 1 to construct the classifiers. For each decision time $t_k \in \mathcal{T}$, Alg. 1 is used to grow a decision tree $tree_k$ and a corresponding STL formula ϕ_k. The algorithm has three meta parameters: 1) PSTL primitives $\mathcal{P}$: the splitting rules at each node are simple PSTL formulae, called primitives Bombara et al. (2016). Here we use the first-order primitives described as $\mathcal{P}_1 = \{G_{[t_0,t_1]}(s_j \sim \pi), F_{[t_0,t_1]}(s_j \sim \pi)\}$, 2) impurity measure $\mathcal{J}$: we use the extended misclassification gain impurity measure from Bombara et al. (2016) as a criterion to select the best primitive at each node, and 3) the stopping conditions $stop$: we stop the growth of the trees when they reach a given depth.

Alg. 1 is recursive and takes as input (1) the prefix dataset $S[0{:}t_k] = \{s^i[0{:}t_k], \ell^i\}_{i=1}^{N}$, (2) the path formula to reach the current node ϕ^{path}, and (3) the current depth level h. At the beginning, the stopping conditions are checked (line 4), and if they are satisfied, a single leaf that is assigned label $c^* \in C$ is returned (lines 5-6), where $p(S[0{:}t_k], c; \phi^{path})$ captures the classification quality based on the impurity measure $\mathcal{J}$. If the stopping conditions are not satisfied, an optimal STL formula among all the possible valuations of the first-order primitives is found (line 7), denoted by ϕ^*, and assigned to a new non-terminal node in the tree (line 8). Next, the prefix dataset $S[0{:}t_k]$ is partitioned according to the optimal formula into satisfying and violating prefix datasets $S_\top[0{:}t_k]$ and $S_\perp[0{:}t_k]$, respectively (line 9), and the construction of the tree continues on the left and right subtrees of the current node (lines 10-11). Each decision tree $tree_k$ is translated to a corresponding STL formula ϕ_k, using the translation method in Bombara et al. (2016). Output of the Classifier Learning is the set of STL formulas $F = \{\phi_k\}_{k=1}^{K}$, $\forall \phi_k \in F : hrz(\phi_k) \leq t_k$.

Example 2 ((Cont.)) *For the naval surveillance scenario, the output of the Signal Analysis block is $\mathcal{T} = \{t_k\}_{k=1}^{8}$. Classifier Learning generates a classifier for each decision time t_k, trained on the prefix dataset $S[0{:}t_k]$, and the output of this part is $F = \{\phi_k\}_{k=1}^{8}$. For example, with 3-fold cross validation and maximum depth = 2 for the trees, the STL formulas learned for the third and fourth decision times are: (i) $t_3 = 20 : \phi_3 = (\phi_{31} \wedge \phi_{32}) \vee (\neg\phi_{31} \wedge \phi_{33})$, $\phi_{31} = G_{[11,16]}(y > 23.33)$, $\phi_{32} = F_{[15,18]}(y \leq 33.83)$, $\phi_{33} = G_{[4,13]}(x > 42.69)$; and (ii) $t_4 = 28 : \phi_4 = (\phi_{41} \wedge \phi_{42}) \vee (\neg\phi_{41} \wedge \phi_{43})$, $\phi_{41} = G_{[7,24]}(y > 20.88), \phi_{42} = F_{[7,27]}(x \leq 44.71)$, $\phi_{43} = G_{[7,27]}(x > 29.02)$. Note that $hrz(\phi_3) = 18 \leq t_3$ and $hrz(\phi_4) = 27 \leq t_4$. The trajectories of the vessels over time and the learned formulas are shown in Fig. 4.*

Algorithm 1 Decision Tree Construction Method $\mathcal{E}$

```
1:  Meta-Parameters: PSTL primitives $\mathcal{P}$, impurity measure $\mathcal{J}$, stopping conditions $stop$
2:  Input: prefix dataset $S[0:t_k]$, path formula $\phi^{path}$, current depth level $h$
3:  Output: sub-tree $tree_k$
4:  if $stop(\phi^{path}, h, S[0:t_k])$ then
5:      $c^* = \text{argmax}_{c \in C}\{p(S[0:t_k], c; \phi^{path})\}$
6:      return $leaf(c^*)$
7:  $\phi^* = \text{argmax}_{\psi \in \mathcal{P}, \theta \in \Theta} \mathcal{J}(S[0:t_k], partition(S[0:t_k], \phi_\theta \wedge \phi^{path}))$
8:  $tree_k \leftarrow non\_terminal(\phi^*)$
9:  $S_\top[0:t_k], S_\bot[0:t_k] \leftarrow partition(S[0:t_k], \phi^{path} \wedge \phi^*)$
10: $tree_k.left \leftarrow \mathcal{E}(S_\top[0:t_k], \phi^{path} \wedge \phi^*, h+1)$
11: $tree_k.right \leftarrow \mathcal{E}(S_\bot[0:t_k], \phi^{path} \wedge \neg\phi^*, h+1)$
12: return $tree_k$
```

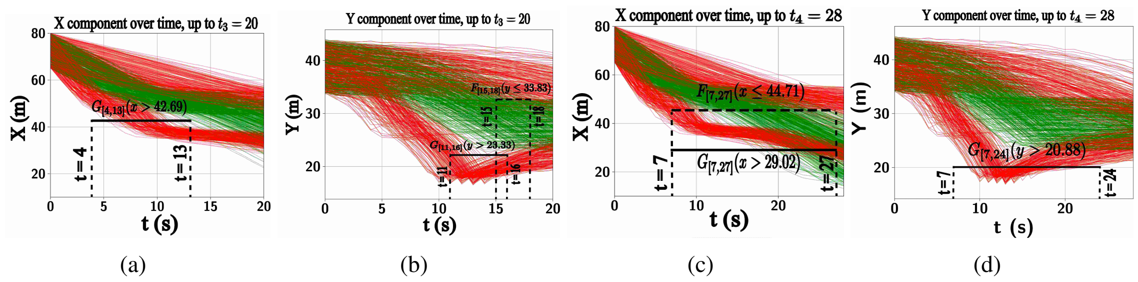

Figure 4: (a), (b) x and y components of naval trajectories over time, up to the decision time $t_3 = 20$, (c) and (d) the same components, up to decision time $t_4 = 28$, respectively. The learned formulas ϕ_3 and ϕ_4 are shown, where the thresholds of the always and eventually operators are shown by solid and dashed lines, respectively.

4.3. Classifier Evaluation

The final component of our framework, Classifier Evaluation, takes as input the given dataset S and the set of generated STL formulas $F = \{\phi_k\}_{k=1}^{K}$. Classifier Evaluation assigns a non-negative, time-variant weight distribution to the formulas, denoted by $\omega(t) = \{\omega_k(t)\}_{k=1}^{K}$, which adjusts the classification performance of the formulas for classifying prefix signals with different time lengths. In Alg. 2, we present a method to find the weights of the formulas $\omega(t)$ over time. With slight abuse of notation, we denote the column vector of the weights at time t by $\omega(t)$, which is a $K \times 1$ array. In Alg. 2, we desire to find the $K \times T$ dimensional matrix $\Omega = [\omega(0), \omega(1), ..., \omega(T)]$ that includes the vectors of the weights over all timepoints along the horizon of the signals.

First, we introduce some notations: at each time step t, the subset of the formulas in F that have a horizon less than or equal t is denoted by $F_t^{\leq}$, and the rest of the formulas that have higher horizon are denoted by $F_t^{>}$. Note that $F_t^{\leq} \cap F_t^{>} = \varnothing$ and $F_t^{\leq} \cup F_t^{>} = F$. The reason for this partitioning is that at each time step t and the prefix signal $s^i[0{:}t]$, the formulas in $F_t^{\leq}$ are able to predict a label for $s^i[0{:}t]$, based on their satisfaction or violation with respect to the prefix signal. However, the set of formulas in $F_t^{>}$ may not be conclusive to predict a label and they need more time instances of the prefix signal. It is clear that at $t = 0$, $F_0^{\leq} = \varnothing$ and $F_0^{>} = F$, and at $t = T$, $F_T^{\leq} = F$ and $F_T^{>} = \varnothing$.

Alg. 2 takes as input the dataset S and the set of STL formulas F. The subsets $F_0^{\leq}$ and $F_0^{>}$ and the weight vector $\omega(0)$ are initialized by $\varnothing, F$, and the zero vector $\emptyset$, respectively (line 3). For each time step t along the horizon of signals (line 4), the subsets of formulas $F_t^{\leq}$ and $F_t^{>}$ are computed by the $partition\_formulas$ function (line 5). This function compares the horizons of the formulas in F with the current time step t, and partitions them into $F_t^{\leq}$ and $F_t^{>}$. If there is no update in the subset $F_t^{\leq}$ compared to the previous time step (line 6), the same weight vector from the previous time step is used for the current time (line 7). If the set $F_t^{\leq}$ is updated (line 8), first, we construct the prefix dataset $S[0:t]$ (line 9). Then, the robustness of the prefix signals in $S[0{:}t]$ are computed with respect to the formulas in $F_t^{\leq}$, by the $compute\_robustness$ function, and stored in the robustness matrix $R_t^{\leq}$ (line 10). The dimensions of the $R_t^{\leq}$ are $N \times |F_t^{\leq}|$, where the i^{th} row contains the robustness of prefix signal $s^i[0{:}t]$ with respect to the formulas in $F_t^{\leq}$. The robustness matrix $R_t^{\leq}$ and the labels of the signals $\{\ell^i\}_{i=1}^N$ are used to learn the weights of the formulas in $F_t^{\leq}$ at time t, denoted by the vector $\omega^{\leq}(t)$ (line 11), and the weights of the formulas in $F_t^{>}$ are set to zero, denoted by $\omega^{>}(t)$ (line 12). The function $learn\_weights$ constructs a Neural Network (NN) to learn the weights of the formulas in $F_t^{\leq}$. Using NNs to learn the weights of wSTL formulas has been explored previously in Yan and Julius (2021). Inspired by Cuturi and Blondel (2017), the differentiable loss function of our designed NN measures the difference between the predicted label of the prefix signal $s^i[0:t]$, and its actual label ℓ^i from the dataset S. Finally, the transposed weight vector $\omega(t)$ is added as a column vector (line 13), to the weight matrix Ω.

Algorithm 2 Learning the weights of the STL formulas

```
1:  Input: dataset $S = \{s^i, \ell^i\}_{i=1}^N$, set of STL formulas $F = \{\phi_k\}_{k=1}^K$
2:  Output: matrix of the weights $\Omega$
3:  Initialize: $F_0^{\leq} \leftarrow \varnothing$, $F_0^{>} \leftarrow F$, $\omega(0) \leftarrow \emptyset$
4:  For $t = 1, ..., T$ :
5:      $F_t^{\leq}$, $F_t^{>} \leftarrow partition\_formulas(F, t)$
6:      if $F_t^{\leq} = F_{t-1}^{\leq}$ then
7:          $\omega(t) = \omega(t-1)$
8:      Otherwise
9:          $S[0:t] = \{s^i[0:t], \ell^i\}_{i=1}^N$
10:         $\mathrm{R}_t^{\leq} \leftarrow compute\_robustness(F_t^{\leq}, S[0:t])$
11:         $\omega^{\leq}(t) \leftarrow learn\_weights(R_t^{\leq}, \{\ell\}_{i=1}^N)$
12:         $\omega^{>}(t) = \emptyset$
13:         $\omega(t) = [\omega^{\leq}(t), \omega^{>}(t)]^{\top}$
14: return $\Omega = [\omega(0), \omega(1), ..., \omega(T)]$
```

The output of the Classifier Evaluation is considered as the weighted conjunction of the STL formulas in F, denoted by the wSTL formula $\Phi = \bigwedge_k^{\omega_k(t)} \phi_k$. The final output of our framework is considered as the predictor P_Φ, which predicts the label of a prefix signal $s^i[0:t]$ as:

$$P_\Phi(s^i[0:t]) = \tilde{\ell}^i(t) = sign\,\Big(\sum_{k=1}^{K} \omega_k(t)\,.\,\rho(\phi_k,\, s^i[0:t])\Big). \tag{1}$$

Note that our proposed predictor computes the robustness of the prefix signal as the weighted sum of the robustnesses of each STL formula in Φ, which is different from monitoring the wSTL formula Φ and computing its robustness by the methods in Mehdipour et al. (2020); Yan and Julius (2021). The time-dependent nature of the weights adjusts the predictor, based on the classification performance of the STL formulas and the time length of the prefix signals. In Sec. 5, we emphasize the importance of the weight distributions of the STL formulas, by showing the performance of the predictor, with and without considering the weight distributions.

Example 3 ((Cont.)) *We apply the Classifier Evaluation part in Alg. 2 to the naval scenario example over the time interval* $[20, 28]$. *Note that* $t = 20$ *and* $t = 28$ *are the horizons of formulas* ϕ_3 *and* ϕ_4 *from Example 2, respectively. At* $t = 20$, *the set of formulas with horizon less than or equal* 20 *is* $F_{20}^{\leq} = \{\phi_1, \phi_2, \phi_3\}$, *and the set of formulas with longer horizon is* $F_{20}^{>} = \{\phi_4, ...\phi_8\}$. *For the prefix signals* $s^i[0 : 20]$, *the weighted sum of their robustnesses with respect to formulas in* $F_{20}^{\leq}$ *are used to predict their labels* $\hat{\ell}^i(20)$. *An NN is constructed to find the optimal weights* $\omega^{\leq}(20) = [\omega_1(20), \omega_2(20), \omega_3(20)]$ *for the formulas in* $F_{20}^{\leq}$, *to minimize the misclassification rate of the prefix signals. For the formulas in* $F_{20}^{>}$, *their weight vector* $\omega^{>}(20) = [\omega_4(20), ..., \omega_8(20)]$ *is set to zero, and we have* $\omega(20) = [\omega^{\leq}(20), \omega^{>}(20)]^\top$.

For the timepoints $t = 21$ *to* $t = 27$, *there is no update in the set* $F_t^{\leq}$, *i.e.,* $F_t^{\leq} = F_{t-1}^{\leq}$, *and the robustness of the prefix signals with respect to formulas in* $F_t^{\leq}$ *are the same as* $t = 20$. *Therefore, we have:* $\forall 21 \leq t \leq 27 : \omega(t) = \omega(t = 20)$. *At* $t = 28$, *the formula* ϕ_4 *is included in the set* $F_t^{\leq}$. *Since* $F_t^{\leq} \neq F_{t-1}^{\leq}$, *a new NN is constructed to compute the weights* $\omega^{\leq}(t) = [\omega_1^{\leq}(t), ..., \omega_4^{\leq}(t)]$, *with respect to prefix signals* $s^i[0 : 28]$, *and* $\omega^{>}(t) = [\omega_5(t), ..., \omega_8(t)]$ *is set to zero. The same procedure is followed for all other timepoints along the signals horizon.*

5. Case Studies

We demonstrate the usefulness and classification performance of our approach with two case studies, i.e., the naval surveillance scenario in Sec. 1 and an urban driving scenario implemented in the simulator CARLA Dosovitskiy et al. (2017). We compare our framework with two baselines:
(i) Uniform-weights: The main purpose of the uniform-weights baseline is to emphasize on the importance of considering time-variant weight distributions in the predictor function. In this baseline, we follow the Signal Analysis and Classifier Learning parts exactly as in our framework, i.e., $\mathcal{T}_u = \mathcal{T} = \{t_k\}_{k=1}^{K}$ and $F_u = F = \{\phi_k\}_{k=1}^{K}$, but we consider a uniform distribution for the weights of the formulas at all decision times and we skip the Classifier Evaluation part. Therefore, $\forall k \in \{1, ..., K\}$ and $\forall t \in \{1, ..., T\}$, we have $\omega_{k,u}(t) = 1$ and the final output of uniform-weights baseline is considered as $\Phi_u = \bigwedge_{k=1}^{K} \phi_k$;
(ii) All-times: In this baseline, we show the significance of choosing a finite number of decision times. Instead of choosing a finite number of decision times, we generate classifiers at all timepoints along the horizon T of signals. Hence, the set of decision times in the all-times baseline is $\mathcal{T}_a = \{1, 2, ..., T\}$ and we skip the signal analysis part. For each decision time $t_{k,a} \in \mathcal{T}_a$ and prefix dataset $S[0 : t_k]$, we use the Classifier Learning and Classifier Evaluation parts to learn an STL formula $\phi_{k,a}$ and weight distribution $\omega_{k,a}(t)$. The final output of the all-times baseline is considered as $\Phi_a = \bigwedge_{k=1}^{T}{}^{\omega_{k,a}(t)} \phi_{k,a}$.

We use Particle Swarm Optimization (PSO) Kennedy and Eberhart (1995) to solve the optimization problems in Alg. 1 and the PyTorch Paszke et al. (2019) library for modeling the NNs. The parameters of the PSO and the NN used in Sec. 4.3, are tuned empirically. We use the same hyperparameter initialization for our method and the two baselines. In both scenarios, we evaluate our framework with a maximum depth of 2 for the decision trees, and with 3-fold cross-validation. The execution times are measured based on the system's clock. All computations are done in Python 2, on an Ubuntu 18.04 system with an Intel Core i7 @3.70 GHz processor and 16 GB RAM.

5.1. Naval scenario

Consider the naval surveillance scenario from Kong et al. (2016) shown in Fig. 1. The dataset consists of 2000 signals, with 1000 normal and 1000 anomalous trajectories. The signals are the 2-

dimensional trajectories with planar coordinates $(x(t), y(t))$ and 61 timepoints. The labels indicate the type of the vessel's behavior (normal or anomalous).

Results: The Signal Analysis component of our framework finds 8 decision times (see Fig. 3), which are: $\mathcal{T}_{Naval} = \{t_k\}_{k=1}^{8} = \{12, 16, 20, 28, 36, 38, 41, 60\}$. Examples of formulas learned by the Classifier Learning part are presented in Sec. 2. The final wSTL formula learned by our framework is $\Phi_{Naval} = \bigwedge_{k=1}^{8}{}^{\omega_k(t)} \phi_k$, where $\omega_k(t)$ is the time-dependent weight distribution learned by the Classifier Evaluation part. The total execution time for our method is $739s$.

The uniform-weights baseline has the same structure and formulas as our framework, but with uniform weight distribution for the formulas: $\Phi_{Naval,u} = \bigwedge_{k=1}^{8} \phi_k$. The total runtime of the uniform-weights baseline is $727s$. The difference between the runtime of this baseline and our approach is due to running the Classifier Evaluation part for our method, which takes about $12s$. For the all-times baseline, the set of decision times consists of all timepoints along the signals horizon: $\mathcal{T}_a = \{1, 2, ..., 60\}$. For each decision time, we learn an STL formula and then assign a weight distribution, which leads to the formula $\Phi_{Naval,a} = \bigwedge_{k=1}^{60}{}^{\omega_{k,a}(t)} \phi_{k,a}$. The runtime of the all-time baseline is $9747s$, which is noticeably larger than our framework, as it learns 60 STL formulas and computes their corresponding weight distributions over time.

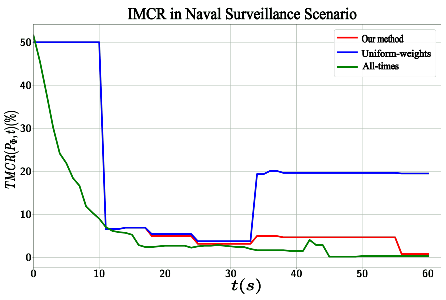

Figure 5: IMCR comparison of our framework with the baseline methods, in the naval surveillance scenario.

The incremental misclassification rate (IMCR) (see Sec. 3) of our framework, compared with the two baselines, is shown in Fig. 5. The IMCR of our approach is better than the uniform-weights baseline at all timepoints, which shows the significance of considering the time-dependent weights for the formulas that adjust their classification power over time. For the all-times baseline, its runtime ($9747s$) and memory consumption (learning 60 STL formulas and computing their weight distributions) are noticeably bigger than our method (with a runtime of $739s$ and learning only 8 STL formulas and their weight distributions). Therefore, as we expected, the IMCR of all-time baseline is generally better than our approach, at the cost of very large execution time and memory consumption.

5.2. Urban-driving scenario

Consider the urban-driving scenario in Fig. 6. It consists of an autonomous vehicle, referred as *ego*, a pedestrian, and another car driven by a human driver. Ego and the other car are in different, adjacent lanes, moving in the same direction on an uphill road toward an intersection with no traffic light. The vehicles are moving by applying constant accelerations, which are smaller for ego, and their positions are initialized such that the other car is always ahead of ego. There is an unmarked cross-walk at the end, and a pothole in the middle of the ramp-shaped road.

There are two possible types (labels) for the behavior of the other car: aggressive or safe. An aggressive driver keeps the same acceleration while moving in its lane, while a safe driver brakes slightly when he reaches the pothole, and applies a full-brake to stop before the intersection if the pedestrian crosses the street. Predicting the behavior label of the human driver in another car is valuable, especially in the case that ego does not have a clear line-of-sight to the pedestrian. We implement this scenario in the simulator CARLA. The simulation ends whenever ego gets closer than $2m$ to its goal point. We assume ego is able to estimate the relative position and velocity of the other car. The dataset of the scenario consists of 300 signals with 477 uniform time-samples per trace. 150 of the signals are for an aggressive driver and 150 are for a safe driver. The signals are 4-dimensional, which are the relative position and velocity of the other car in y and z axes.

Results: The Signal Analysis part finds 11 decision times in the dataset of this scenario, which are: $\mathcal{T}_{Urban} = \{t_k\}_{k=1}^{11} = \{97, 131, 166, 186, 240, 289, 334, 394, 420, 440, 476\}$. An example of the formula learned by the Classifier Learning part for the decision time $t_2 = 131$ is: $\phi_2 = (\phi_{21} \wedge \phi_{22}) \vee (\neg\phi_{21} \wedge \phi_{23})$, where $\phi_{21} = F_{[122,130]}(v_y \leq 1.11)$, $\phi_{22} = F_{[108,124]}(y \leq 12.58)$, and $\phi_{23} = F_{[110,121]}(y \leq 6.81)$. For example, ϕ_{21} states that at some timepoint between 122 to 130, the relative velocity of the other car in the y-axis gets less than or equal to 1.11 m/s. Note that $hrz(\phi_2) = 130 \leq t_2$. The final wSTL formula learned by our approach is $\Phi_{Urban} = \bigwedge_{k=1}^{11} {}^{\omega_k(t)} \phi_k$, where the weight distribution $\omega_k(t)$ is obtained through the Classifier Evaluation part. The total runtime of our framework for this case study is $812s$.

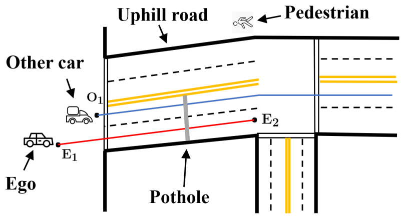

Figure 6: Schematic of the urban-driving scenario. The initial points of ego and other car are denoted by E_1 and O_1 and the goal point for ego is denoted by E_2.

The uniform-weights baseline has the same formulas with uniform weight distribution: $\Phi_{Naval,u} = \bigwedge_{k=1}^{11} \phi_k$, and its total runtime is $805s$. For the all-times baseline, the set of decision times is: $\mathcal{T}_a = \{1, 2, ..., 477\}$, and correspondingly the final wSTL formula is $\Phi_{Naval,a} = \bigwedge_{k=1}^{477} {}^{\omega_{k,a}(t)} \phi_{k,a}$. The runtime of the all-time baseline is $33158s$, which is exponentially larger than our framework, as it learns 477 STL formulas and their weight distributions over time.

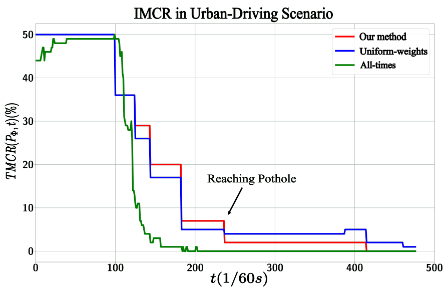

Figure 7: IMCR comparison of our framework with the baseline methods in the urban-driving scenario.

The IMCR comparison of our approach with the baseline methods is shown in Fig. 7. As we expected, the classification accuracy of our approach is better than the uniform-weights baseline over the time horizon of the scenario. For the all-times baseline, its IMCR is generally better than our framework, at the cost of exponentially larger execution time ($33158s$) and memory consumption (learning 477 STL formulas and their weight distributions), which makes it inefficient for large datasets with long time horizons.

6. Conclusion

In this paper, we considered the problem of predicting the labels of prefix signals over time, given a dataset of labeled signals. Our proposed framework combines temporal logic and neural networks to construct a predictor based on STL formulas for classifying the prefix signals. The effectiveness of our method was evaluated in an urban driving and a naval surveillance scenario. In future work, we will explore advanced signal analysis techniques and formulate optimization problems to find the decision times. We will also investigate the performance of other machine learning methods, such as recurrent neural networks and transformers, for time-incremental learning frameworks.

Acknowledgments

This work was partially supported by the NSF under grant IIS-2024606 at Boston University.

References

Erfan Aasi, Cristian Ioan Vasile, Mahroo Bahreinian, and Calin Belta. Classification of time-series data using boosted decision trees. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1263–1268. IEEE, 2022.

Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *International Conference on Runtime Verification*, pages 147–160. Springer, 2011.

Nasim Baharisangari, Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu, and Zhe Xu. Uncertainty-aware signal temporal logic. *arXiv preprint arXiv:2105.11545*, 2021.

Alexey Bakhirkin, Thomas Ferrère, and Oded Maler. Efficient parametric identification for stl. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 177–186, 2018.

Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. Data-driven statistical learning of temporal logic properties. In *International conference on formal modeling and analysis of timed systems*, pages 23–37. Springer, 2014.

Ezio Bartocci, Luca Bortolussi, Laura Nenzi, and Guido Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25, 2015.

Giuseppe Bombara and Calin Belta. Signal clustering using temporal logics. In *International Conference on Runtime Verification*, pages 121–137. Springer, 2017.

Giuseppe Bombara, Cristian-Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2016.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. In *International Conference on Machine Learning*, pages 894–903. PMLR, 2017.

Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *preprint arXiv:1711.03938*, 2017.

Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20(1):79–93, 2018.

Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. Telex: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design*, 54(3):364–387, 2019.

Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, 2015.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

Ahmet Ketenci and Ebru Aydin Gol. Synthesis of monitoring rules via data mining. In *American Control Conference*, pages 1684–1689, 2019.

Zhaodan Kong, Austin Jones, and Calin Belta. Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3):1210–1222, 2016.

Alexis Linard and Jana Tumova. Active learning of signal temporal logic specifications. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 779–785. IEEE, 2020.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

Noushin Mehdipour, Cristian-Ioan Vasile, and Calin Belta. Specifying user preferences using weighted signal temporal logic. *IEEE Control Systems Letters*, 2020.

Sara Mohammadinejad, Jyotirmoy V Deshmukh, and Aniruddh G Puranic. Mining environment assumptions for cyber-physical system models. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 87–97. IEEE, 2020a.

Sara Mohammadinejad, Jyotirmoy V Deshmukh, Aniruddh G Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. Interpretable classification of time-series data using efficient enumerative techniques. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2020b.

Daniel Neider and Ivan Gavran. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10. IEEE, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

Marcell Vazquez-Chanlatte, Jyotirmoy V Deshmukh, Xiaoqing Jin, and Sanjit A Seshia. Logical clustering and learning for time-series data. In *International Conference on Computer Aided Verification*, pages 305–325. Springer, 2017.

Zhe Xu, Melkior Ornik, A Agung Julius, and Ufuk Topcu. Information-guided temporal logic inference with prior knowledge. In *2019 American control conference (ACC)*, pages 1891–1897. IEEE, 2019.

Ruixuan Yan and Agung Julius. Neural network for weighted signal temporal logic. *arXiv preprint arXiv:2104.05435*, 2021.

Ruixuan Yan, Zhe Xu, and Agung Julius. Swarm signal temporal logic inference for swarm behavior analysis. *IEEE Robotics and Automation Letters*, 4(3):3021–3028, 2019.