Interpretable Classification of Time-Series Data using Efficient Enumerative Techniques

SARA MOHAMMADINEJAD, University of Southern California
JYOTIRMOY V. DESHMUKH, University of Southern California
ANIRUDDH G. PURANIC, University of Southern California
MARCELL VAZQUEZ-CHANLATTE, University of California, Berkeley
ALEXANDRE DONZÉ, Decyphir, Inc

Cyber-physical system applications such as autonomous vehicles, wearable devices, and avionic systems generate a large volume of time-series data. Designers often look for tools to help classify and categorize the data. Traditional machine learning techniques for time-series data offer several solutions to solve these problems; however, the artifacts trained by these algorithms often lack interpretability. On the other hand, temporal logic, such as Signal Temporal Logic (STL) have been successfully used in the formal methods community as specifications of time-series behaviors. In this work, we propose a new technique to automatically learn temporal logic formulas that are able to classify real-valued time-series data. Previous work on learning STL formulas from data either assumes a formula-template to be given by the user, or assumes some special fragment of STL that enables exploring the formula structure in a systematic fashion. In our technique, we relax these assumptions, and provide a way to systematically explore the space of all STL formulas. As the space of all STL formulas is very large, and contains many semantically equivalent formulas, we suggest a technique to heuristically prune the space of formulas considered. Finally, we illustrate our technique on various case studies from the automotive and transportation domains.

Additional Key Words and Phrases: Cyber-physical systems, time-series data, machine learning, formal methods, Signal Temporal Logic, interpretability

ACM Reference Format:
Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh G. Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. 2020. Interpretable Classification of Time-Series Data using Efficient Enumerative Techniques. 1, 1 (June 2020), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Cyber-physical systems (CPS) generate large amounts of data due to a proliferation of sensors responsible for monitoring various aspects of the system. Designers are typically interested in extracting high-level information from such data but due to its large volume, manual analysis is not feasible. Hence a crucial question is: *How do we automatically identify logical structure or relations within CPS data?* Machine learning (ML) algorithms may not be specialized to learn logical structure underlying time-series data[28], and typically require users to hand-create features of interest in the

Passed HSCC 2020 Repeatability Evaluation!
Authors' addresses: Sara Mohammadinejad, University of Southern California, saramoha@usc.edu; Jyotirmoy V. Deshmukh, University of Southern California, jdeshmuk@usc.edu; Aniruddh G. Puranic, University of Southern California, puranic@usc.edu; Marcell Vazquez-Chanlatte, University of California, Berkeley, marcell.vc@berkeley.edu; Alexandre Donzé, Decyphir, Inc
, alex@decyphir.com.

2020. Manuscript submitted to ACM

underlying time-series signals. These methods then try to learn discriminators over feature spaces to cluster or classify data. These feature spaces can often be quite large, and ML algorithms may choose a subset of these features in an *ad hoc* fashion. This results in an artifact (e.g., a discriminator or a clustering mechanism) that is not human-interpretable. In fact, ML techniques focus only on solving the classification problem and suggest no other comprehension of the target system [5].

Signal Temporal Logic (STL) is a popular formalism to express properties of time-series data in several application contexts, such as automotive systems [3, 14, 17], analog circuits [22], biology [26], robotics [31], etc. STL is a logic over Boolean and temporal combinations of signal predicates which allows human-interpretable specification of continuous system requirements. For instance, in the automotive domain, STL can be used to formulate properties such as "the car successfully stops before hitting an obstruction" [18].

There has been significant work in learning STL specifications from data. Some of this work has focused on supervised learning (where given labeled traces, an STL formula is learned to distinguish positively labeled traces from negatively labeled traces) [5, 9, 18, 24], or clustering (where signals are clustered together based on whether they satisfy similar STL formulas) [28, 29], or anomaly detection (where STL is utilized for recognizing the anomalous behavior of an embedded system) [16]. There are two main approaches in learning STL formulas from data: template-free learning and template-based learning. Template-free methods learn both the structure of the underlying STL formula and the formula itself. While these techniques have been proven effective for many diverse applications [5, 9, 16, 18, 24], they typically explore only a fragment of STL, and may produce long and complicated STL classifiers. Certain properties such as concurrent eventuality, repeated patterns and persistence may not be expressible in the chosen fragments for learning [18]. In template-based methods, the user provides a template parametric STL formula (PSTL formula) and the learning algorithm infers only the values of parameters from data [12, 13, 15, 28, 29]. Without a good understanding of the application domain, choosing the appropriate PSTL formula can be challenging. Furthermore, in template-based methods the users may provide very specific templates which may make it difficult to derive new knowledge from data[5].

Syntax-guided synthesis is a new paradigm in "learning from examples", where the user provides a grammar for expressions, and the learning algorithm tries to learn a concise expression that explains a given set of examples. In [27], systematic enumeration has been used to generate candidate solutions. For medium-sized benchmarks, the systematic enumeration algorithm, in spite of its simplicity, surprisingly outperforms several other learning approaches [1].

Inspired by the idea of learning expressions from grammars, in this paper, we consider the problem of learning STL based formulas to classify a given labeled time-series dataset (*with focus on the monotonic fragment of STL*). The key challenge in systematic enumeration for STL is that predicates over real-valued signals and time-bounds on temporal operators both involve real numbers. This means that even for a fixed length, there are an infinite number of STL formulas of that length. One solution is to apply the enumerative approach to PSTL, which uses parameters instead of numbers. The inference problem then tries to learn parameter values to separate labeled data into distinct classes. The parameter-valuation inference procedures are typically efficient, but over a large dataset, the cost for enumeration followed by parameter inference can add up. As a result, we explore an optimization which involves skipping formulas that are heuristically determined to be equivalent.

The work in [9] is highly related to the work presented in this paper. In this recent work, the authors discuss the problem of online monitoring and formula synthesis for a past fragment of STL. A key difference between [9] and this paper is the use of signature-based optimization and systematic exploration of the parameter space using a

multi-dimensional binary search method. It would be interesting to consider the extension of our method to the past fragment of STL and compare the relative merits of the two approaches in future work.

As a concrete application of enumerative search, we consider the problem of learning an STL-based classifier with a minimal misclassification rate for the given labeled dataset.

Our tool generates the shortest possible STL formula. In the ML community, there is a preference to classifiers with smaller descriptive complexity to comply with the Occam's Razor principle. Cf. [19] for results on shorter decision-trees being favorable. In the context of rule learning, shorter rules are considered more interpretable and are preferred by designers [11]. The main hypothesis is that shorter STL formulas can be translated to short plain English phrases, which makes them human-interpretable. This is in contrast to ML methods where a classifier may be a linear combination of a number of features such as statistical moments of the signal, frequency domain coefficients, or other ad hoc user-defined features.

To summarize, our key contributions are as follows:

- We extend the work in [12, 13, 15, 28, 29] by learning the structure or template of PSTL formulas automatically. The enumerative solver furthers the Occam's razor principle in learning (simplest explanations are preferred). Thus, it produces simpler STL formulas compared to existing template-free methods [5, 16, 18, 24].
- We introduce the notion of formula signature as a heuristic to prevent enumeration of equivalent formulas.
- We bridge formal methods and machine learning algorithms by employing STL, which is a language for formal specification of continuous and discrete system behaviors [21]. We use Boolean satisfaction of STL formulas as a formal measure for measuring the misclassification rate.
- We showcase our technique on real world data from several domains, including automotive and transportation domains.

2 PRELIMINARIES

Definition 1 (Time-Series, Traces). *A trace x is a mapping from time domain T to value domain $\mathcal{D}$, $x : T \to \mathcal{D}$ where, $T \subseteq \mathbb{R}^{\geq 0}$, $\mathcal{D} \subseteq \mathbb{R}^n$ and $D \neq \emptyset$.*

Signal Temporal Logic (STL). Signal Temporal Logic [21] is used as a specification language for reasoning about properties of real-valued signals. The simplest properties or constraints can be expressed in the form of atomic predicates. An atomic predicate is formulated as $f(\mathbf{x}) \sim c$, where f is a function from $\mathcal{D}$ to $\mathbb{R}$, $\sim \in \{\geq, \leq, =\}$, and $c \in \mathbb{R}$. For instance, $x(t) \geq 2$ is an atomic predicate, where $f(x) = x(t)$, $\sim$ is $\geq$, and $c = 2$. Temporal properties include temporal operators such as **G** (always), **F** (eventually) and **U** (until). For example, $\mathbf{G}(x(t) > 2)$ means signal $x(t)$ is always greater than 2. Each temporal operator is indexed by an interval $I := (a, b) \mid (a, b] \mid [a, b) \mid [a, b]$, where $a, b \in T$. Every STL formula is written in the following form:

$$\varphi := \mathit{true} \mid f(\mathbf{x}) \sim c \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathbf{U}_I \, \varphi_2 \tag{1}$$

where $c \in \mathbb{R}$. **G** and **F** operators are special instances of **U** operator; $\mathbf{F}_I\varphi \triangleq \mathit{true}\,\mathbf{U}_I\,\varphi$, and $\mathbf{G}_I\varphi \triangleq \neg\mathbf{F}_I\neg\varphi$, and they are defined for formula simplification. The boolean semantics of an STL formula are defined formally as follows:

$$\begin{aligned}
(\mathbf{x}, t) &\models f(\mathbf{x}) \sim c \quad &\iff \quad & f(\mathbf{x}(t)) \sim c \text{ is true} \\
(\mathbf{x}, t) &\models \neg\varphi \quad &\iff \quad & (\mathbf{x}, t) \not\models \varphi \\
(\mathbf{x}, t) &\models \varphi_1 \wedge \varphi_2 \quad &\iff \quad & (\mathbf{x}, t) \models \varphi_1 \ \wedge \ (\mathbf{x}, t) \models \varphi_2 \\
(\mathbf{x}, t) &\models \varphi_1\,\mathbf{U}_I\,\varphi_2 \quad &\iff \quad & \exists t_1 \in t \oplus I : (\mathbf{x}, t_1) \models \varphi_2 \ \wedge \\
& & & \quad \forall t_2 \in [t, t_1) : (\mathbf{x}, t_2) \models \varphi_1,
\end{aligned}$$

where $\oplus$ denotes the Minkowski sum ($t \oplus [a, b] = [t + a, t + b]$). The signal x satisfies $f(x) > 0$ at time t (where $t \geq 0$) if $f(x(t)) > 0$. It satisfies $\varphi = \mathbf{G}_{[0,2)}(x > 0)$ if for all time $0 \leq t < 2, x(t) > 0$ and satisfies $\varphi = \mathbf{F}_{[0,1)}(x > 0)$ if exists t, such that $0 \leq t < 1$ and $x(t) > 0$. The signal x satisfies the formula $\varphi = (x > 0)\mathbf{U}_{[0,2]}(x < 1)$ if there exists some time t_1 where $0 \leq t_1 \leq 2$ and $x(t_1) < 1$, and for all time $t_2 \in [0, t_1), x(t_2) > 0$. We can create higher-level STL formulas by utilizing two or more of the operators. For instance, a signal x satisfies $\varphi = \mathbf{F}_{[0,1]}\mathbf{G}_{[0,2]}(x(t) > 1)$ iff there exists t_1 such that $0 \leq t_1 \leq 1$, and for all time $t_1 \leq t \leq t_1 + 2, x(t) > 1$.

In addition to the Boolean semantics, quantitative semantics of STL quantify the robustness degree of satisfaction by a particular trace[7, 8]. Intuitively, a STL with a large positive robustness is far from violation, and with large negative robustness is far from satisfaction. If the robustness is a small positive number, a very small perturbation might make it negative and lead to the violation of the property. Quantitative semantics of STL are formally defined as follow:

$$\begin{aligned}
&\rho(\mu, \mathbf{x}, t) = f(\mathbf{x}(t)) \\
&\rho(\neg\varphi, \mathbf{x}, t) = -\rho(\varphi, \mathbf{x}, t) \\
&\rho(\varphi_1 \ \wedge \ \varphi_2, \mathbf{x}, t) = \min(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t)) \\
&\rho(\varphi_1 \mathbf{U}_I \varphi_2, \mathbf{x}, t) = \sup_{t' \in t \oplus I} \min \begin{pmatrix} \rho(\varphi_2, \mathbf{x}, t'), \\ \inf_{t'' \in [t, t')} \rho(\varphi_1, \mathbf{x}, t'') \end{pmatrix}
\end{aligned}$$

For instance, consider the signal $x(t) = 11 - t$, the robustness of the formula $\mathbf{G}_{[0,10]}(x > 0)$ is the minimum of $x(t)$ over $[0, 10]$ (i.e. = 1), and the robustness of $F_{[0,5]}(x > 7)$ is the maximum of $x(t) - 7$ over $[0, 5]$ (i.e. = 4). Robustness approximates the signed distance between the signal and the set of signals satisfying (or violating) the formula.

Property "Always between time 0 and some unspecified time τ, the signal x is less than some value π" could be expressed using Parametric STL (PSTL) formula $\mathbf{G}_{[0,\tau]}(x < \pi)$, where the unspecified values τ and π are referred to as *parameters*.

Parametric Signal Temporal Logic (PSTL). PSTL [2] is an extension of STL where constants are replaced by parameters. The associated STL formula is obtained by assigning a value to each parameter variable using a valuation function. Let $\mathcal{P}$ be the set of parameters, V represent the domain set of the parameter variables $\mathcal{P}^V$, and T represent the time domain of the parameter variables $\mathcal{P}^T$. Then, $\mathcal{P}$ is the set containing the two disjoint sets $\mathcal{P}^V$ and $\mathcal{P}^T$, where at least one of the sets is non-empty. A valuation function ν maps a parameter to a value in its domain. A vector of parameter variables $\mathbf{p}$ is obtained by mapping parameter vectors $\mathbf{p}$ into tuples of respective values over V or T. Hence, we obtain the *parameter space* $\mathcal{D}_\mathcal{P} \subseteq V^{|\mathcal{P}^V|} \times T^{|\mathcal{P}^T|}$.

An STL formula is obtained by pairing a PSTL formula with a valuation function that assigns a value to each parameter variable. For example, consider the PSTL formula $\varphi(c, \tau) = \mathbf{F}_{[0,\tau]}(x > c)$ with parameters c and τ. The STL formula $\mathbf{F}_{[0,5]}(x > -2.3)$ which is an instance of φ is obtained with the valuation $\nu = \{\tau \mapsto 5, c \mapsto -2.3\}$.

Definition 2 (Monotonic PSTL). *A parameter p_i is said to be monotonically increasing or have positive polarity in a PSTL formula φ if condition* (2) *holds for all* $\mathbf{x}$*, and is said to be monotonically decreasing or have negative polarity if condition* (3) *holds for all* $\mathbf{x}$*, and monotonic if it is either monotonically increasing or decreasing [2].*

$$\nu(p_i) \leq \nu'(p_i) \Rightarrow \quad \left[\mathbf{x} \models \varphi(\nu(p_i)) \Rightarrow \mathbf{x} \models \varphi(\nu'(p_i))\right] \tag{2}$$

$$\nu(p_i) \geq \nu'(p_i) \Rightarrow \quad \left[\mathbf{x} \models \varphi(\nu(p_i)) \Rightarrow \mathbf{x} \models \varphi(\nu'(p_i))\right] \tag{3}$$

For example, in the formula $\mathbf{F}_{[0,\tau]}(x > c)$, polarity of τ is positive (the formula is monotonically increasing with respect to τ), and polarity of c is negative (the formula is monotonically decreasing with respect to c). If a trace satisfies $\mathbf{F}_{[0,1]}(x > 0)$ (there exists at least a time instance $t' \in [0,1]$ where $x(t') > 0$) , it will definitely satisfy $\mathbf{F}_{[0,2]}(x > 0)$ ($t' \in [0,2]$).

Definition 3 (Validity domain). *The validity domain for a given set of parameters $\mathcal{P}$ is the set of all valuations s.t. for the given set of traces X, each trace satisfies the STL formula obtained by instantiating the given PSTL formula with the parameter valuation. The boundary of the validity domain is the set of valuations where the robustness value of the given STL formula with respect to at least one trace in X is* 0.

Essentially, the validity domain boundary serves as a classifier to separate the set of traces satisfying the STL formula from the ones violating the formula.

Supervised Learning/Classification. Supervised Learning is an ML technique used for learning from labeled data set. Supervised classification problems are either binary (only two classes are involved) or multi-class classification (more than two classes are included). In this paper, we explore the problem of multi-class classification for time-series data and use boolean semantics of STL as a logical measure for misclassification rate (MCR). In general, MCR is computed as the number of falsely classified traces divided by the number of all traces. We then evaluate our method on real-world data, including automotive and transportation domains.

3 ENUMERATIVE LEARNING FOR STL

In this section, we introduce systematic PSTL enumeration for learning PSTL formula classifiers from time-series data, which the enumeration procedure is formalized in Algo. 1. From a grammar-based perspective a PSTL formula can be viewed as atomic formulas combined with unary or binary operators. For instance, PSTL formula $\mathbf{G}_{[0,\tau_1]}(x(t) > c_1) \wedge \mathbf{F}_{[0,\tau_2]}(x(t) < c_2)$ consists of binary operator $\wedge$, unary operators $\mathbf{G}$ and $\mathbf{F}$, and atomic predicates $x(t) > c_1$ and $x(t) < c_2$.

$$\begin{aligned} &\varphi := atom \mid unaryOp(\varphi) \mid binaryOp(\varphi, \varphi) \\ &unaryOp := \neg \mid \mathbf{F} \mid \mathbf{G} \\ &binaryOp := \vee \mid \wedge \mid \mathbf{U}_I \mid \Rightarrow \end{aligned} \tag{4}$$

Algo. 1 is algorithm with several nested iterations. The outermost loop iterates over the length of the formula, and in the first iteration of the loop, we basically enumerate formulas of length 1, or parameterized signal predicates using the *EnumAtoms* function. At end of the ℓ^{th} iteration of the algorithm, all formulas up to length ℓ are stored in a database *DB* that is an array of lists. Each array index corresponds to the formula length, and each of the lists stored at location ℓ is the list of all formulas of length ℓ. In each iteration corresponding to lengths greater than 1, the algorithm calls the function *ApplyUnaryOps*, and in all iterations for lengths greater than 2, the function also calls *ApplyBinaryOps*. When enumerating formulas of length ℓ, *ApplyUnaryOps* function iteratively applies each unary operator from the ordered list *unaryOps* to all formulas of length $\ell - 1$ to get a new formula. The *ApplyBinaryOps* iteratively applies each

binary operator from the ordered list *binaryOps* to a pair of formulas of lengths a and b, where $a, b \in [1, \ell - 2]$ and $a + b = \ell - 1$. We use *atomIt*, *unOpIt*, *binaryOpIt*, *argIt*, *lhsIt*, *rhsIt* as iterators (indices) on the lists *atoms*, *unaryOps*, *binaryOps*, $DB(\ell - 1)$, and *lhsArgs* and *rhsArgs* respectively.

For each PSTL formula φ generated by Algo. 1, we apply the procedure (*TryClassifier*) which is formalized in Algo. 2. If φ is a good classifier (small misclassification rate), all loops terminate and φ is returned. Otherwise, the procedure continues to generate new PSTL formulas. We explain Algo. 2 in Section 5. In order to avoid applying Algo. 2 (*TryClassifier*) on equivalent PSTL formulas, we use the idea of *formula signatures* to heuristically detect equivalent PSTL formulas. We explain this optimization in the next section.

4 SIGNATURE-BASED OPTIMIZATION

Enumerating logical classification, which is implemented in Algo. 2, for every PSTL formula is time-consuming. The problem with naïve enumeration approach for PSTL formulas is that many equivalent PSTL formulas are enumerated. Hence, we are interested in detecting equivalent PSTL formulas in order to decrease enumeration time. However, checking equivalence of PSTL formulas is in general undecidable [15]. Even if we restrict ourselves to a fragment of PSTL, equivalence checking is a computationally demanding task. Thus, we use signatures to avoid enumerating logically equivalent formulas. A *signature* is an approximation of a PSTL formula. Inspired by polynomial identity testing [25], we use the notion of signature to check the equivalence of two PSTL formulas. Let $X_n \subseteq X$ be a randomly chosen subset of X (the traces) of cardinality n. Let $\mathcal{D}_{\mathcal{P}^m} = \{\nu_1, \ldots, \nu_m\}$ be a finite subset of $\mathcal{D}_{\mathcal{P}}$ (the parameter space). The signature S of a formula φ maps φ to a matrix of real numbers, defined as below:

$$S_\varphi(i, j) = \rho(\varphi(\nu_j(\mathbf{p})), X_n(i), 0)$$

The $(i, j)^{th}$ element of the matrix represents the robustness of the i^{th} trace, $X_n(i)$ with respect to the j^{th} STL formula $\varphi(\nu_j(\mathbf{p}))$. For checking the satisfaction of a STL specification by a trace we use Breach [6], a toolbox for verification and parameter synthesis of hybrid systems. This procedure is implemented in *computeSignature* function in Algo. 2. *computeSignature* function in Alo. 2 is used to detect whether an enumerated PSTL formula φ is new or its equivalent has been enumerated. Consider two PSTL formulas: $\mathbf{F}_{[0,\tau_1]}(\mathbf{G}_{[0,\tau_2]}(\mathbf{x}(t) > c))$ and $\neg(\mathbf{G}_{[0,\tau_1]}(\mathbf{F}_{[0,\tau_2]}(\mathbf{x}(t) \leq c)))$.

We illustrate the working of the *computeSignature* function using 4 randomly chosen subset of traces (i.e., n=4) from case studies and 5 random parameter samples (i.e., m=5) from time domain $T = [0, 3]$ and value domain $V = [0.8, 1.2]$. For both PSTL formulas the resulting signature is a 4×5 matix with exactly same elements (hence, the two PSTL formulas are equivalent):

$$S_\varphi = \begin{bmatrix} 0.8786 & 0.9294 & -0.4984 & 0.7571 & 0.9404 \\ 0.3317 & 0.2296 & -0.1853 & 0.9455 & 0.2218 \\ 0.4033 & 0.2785 & -0.5105 & 0.8429 & 0.2890 \\ 0.1742 & 0.1816 & -0.6257 & 0.1873 & 0.1738 \end{bmatrix}.$$

Remark. computeSignature *function in Algo. 2 only compares signature of formulas with the same parameters. For instance, formulas* $\mathbf{G}_{[0,\tau]}(\mathbf{x}(t) > c_1)$ *and* $\neg(\mathbf{F}_{[0,\tau]}(\mathbf{x}(t) \leq c_2))$ *are semantically equivalent; however,* computeSignature *function does not detect them as equivalent, and we are going to address this limitation in future work.*

Remark. *As the satisfiability of STL formulas is undecidable in general [15], checking if two arbitrary STL formulas are equivalent is also undecidable. Even if we restrict our attention to MITL formulas (where satisfiability is decidable),*

Algorithm 1: Formula enumeration algorithm

```
   Input: maxLength, atoms, unaryOps, binaryOps, DB
1  Init: ℓ ← 1;
2  while ℓ ≤ maxLength do
3      if ℓ = 1 then EnumAtoms() ;
4      else if ℓ = 2 then ApplyUnaryOps() ;
5      else ApplyUnaryOps(); ApplyBinaryOps();
6      ℓ ← ℓ + 1;
7  Function EnumAtoms():
8      atomIt← 1;
9      while atomIt≤ |atoms| do
10         φ ← get(atoms, atomIt);
11         TryClassifier(φ) ;                                   // Algorithm 2
12         add(DB, ℓ, φ);
13         atomIt← atomIt + 1;
14 Function ApplyUnaryOps():
15     argIt← 1, unOpIt← 1 ;
16     while unOpIt≤ |unaryOps| do
17         op ← get(unaryOps, unOpIt);
18         while argIt< |DB(ℓ − 1)| do
19             unaryArg ← get(DB(ℓ − 1), argIt);
20             φ ← op(unaryArg) ;
21             TryClassifier(φ);
22             Add(DB, ℓ, φ);
23             argIt← argIt + 1;
24         unOpIt← unOpIt + 1 ;
25 Function ApplyBinOps():
26     lhsIt, rhsIt, binaryOpIt← 1 ;
27     while binaryOpIt≤ |binaryOps| do
28         op ← get(binaryOps, binaryOpIt);
29         for i ← 1 to ℓ − 2 do
30             while lhsIt< |DB(i)| do
31                 while rhsIt< |DB(ℓ − i − 1)| do
32                     lhs ← get(DB(i), lhsIt) ;
33                     rhs ← get(DB(ℓ − i − 1), rhsIt);
34                     φ ← op(lhs, rhs) ;
35                     TryClassifier(φ);
36                     Add(DB, ℓ, φ);
37                     rhsIt ← rhsIt + 1 ;
38                 lhsIt ← lhsIt + 1;
39         binaryOpIt← binaryOpIt + 1 ;
```

checking equivalence is computationally expensive (EXPSPACE complete). There is recent new tool for checking satisfiability of MITL formulas [4]. To check the satisfiability of formulas of length less than 20, the tool takes about 10 seconds.

The problem at hand is more complicated than the above problems as we are checking the equivalence of PSTL formulas that requires an additional quantifier on the parameter space.

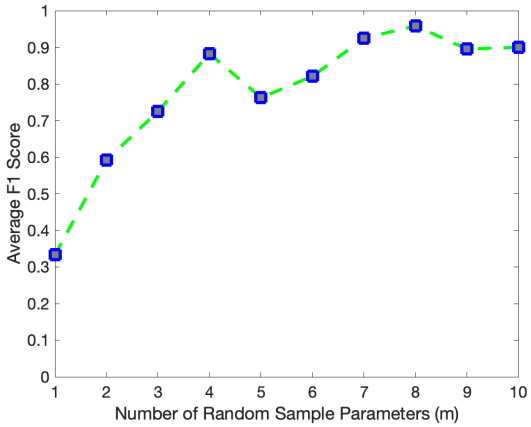

Fig. 1. Change of F_1 score with respect to the number of random sample parameters (m) for fixed number of random traces (n=10).

We have used two common performance parameters to evaluate signature-based optimization, precision and recall in addition to computation time. The precision is an important figure of merit to examine the performance of the technique in not detecting two different PSTL formulas as equivalent. The recall is the performance of the method in detection of all equivalent enumerated PSTL formulas.

$$Precision = \frac{|\{\varphi \mid \varphi \equiv \varphi' \wedge \varphi \equiv_{sign} \varphi'\}|}{|\{\varphi \mid \varphi \equiv \varphi' \wedge \varphi \equiv_{sign} \varphi'\}| + |\{\varphi \mid \varphi \not\equiv \varphi' \wedge \varphi \equiv_{sign} \varphi'\}|},$$

$$Recall = \frac{|\{\varphi \mid \varphi \equiv \varphi' \wedge \varphi \equiv_{sign} \varphi'\}|}{|\{\varphi \mid \varphi \equiv \varphi' \wedge \varphi \equiv_{sign} \varphi'\}| + |\{\varphi \mid \varphi \equiv \varphi' \wedge \varphi \not\equiv_{sign} \varphi'\}|},$$

$$F1\ score = 2.\ \frac{Precision\ .\ Recall}{Precision + Recall},$$

where $\varphi \equiv_{sign} \varphi'$ means $signature(\varphi) = signature(\varphi')$, and $\mathbf{F}_1$ *score* is the harmonic mean of the precision and recall. To evaluate the performance of our technique using precision and recall, we enumerated 500 formulas using our enumerative solver. For computing signatures, traces were chosen randomly from case studies. We added white Gaussian noise with $SNR = 2$ to traces to make them expressive and discriminate enough for detecting equivalent PSTL formulas. We also used 5 different random seeds and computed the average values of performance metrics. The result shows that recall is always equal to 1. This means signature can detect all equivalent PSTL formulas with the same parameters. However, the precision changes with respect to m and n. The average change of $\mathbf{F}_1$ *score* with respect to m for fix $n = 10$ is shown in Fig. 1. As figure shows, $\mathbf{F}_1$ *score* almost increases by increasing m. For $m = 5$ and $m = 9$ $\mathbf{F}_1$ *score* decreases which is due to the nature of added white Gaussian noise. Similarly, change of $\mathbf{F}_1$ *score* with respect to n for fixed $m = 10$ is illustrated is Fig. 2. The overall trend of the plot is increasing except for $n = 9$, where $\mathbf{F}_1$ *score* decreases and, the reason for that is the added white Gaussian noise to traces.

Using the idea of signatures reduced the computation time in targeted case studies; the summary of the results are shown in Table 1. As Table 1 shows, the computation time difference before and after optimization is noticeable, yet, it is at best 30%. The reason is that the enumerative solver produces simple PSTL classifiers in 5 case studies, and until reaching those PSTL classifiers, only a few formulas with equivalent signatures are enumerated. For producing complicated classifiers, the difference would be more noticeable. The run time of signature-based optimization increases

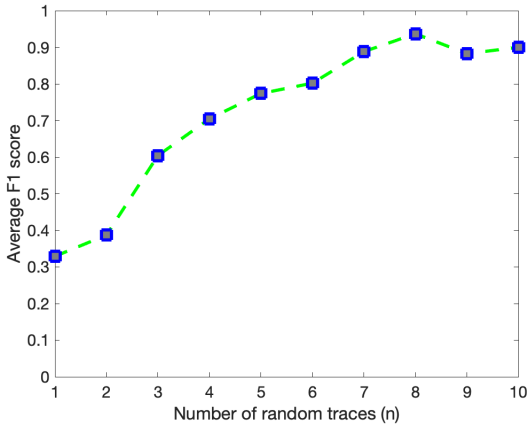

Fig. 2. Change of F_1 score with respect to the number of random traces (n) for fixed number of sample parameters (m=10).

Case	Before optimization (s)	After Optimization (s)
Maritime Surveillance:		
1*st* classifier	3903.02	2699.80
2*nd* classifier	53.78	48.73
3*rd* classifier	28.19	23.17
Linear system	44.25	39.05
Cruise control of train	35.84	32.31
PID controller	158.70	126.66

Table 1. Optimization using signature

with the number of m and n. In future work, we plan to exploit (obvious) syntactic structural equivalences to further prune the space of equivalent formulas. In general, this will require a canonical representation of an STL formula, which can be an expensive step.

5 LOGICAL CLASSIFICATION

Next, we explain how to learn a logical and interpretable classifier for binary classification of time-series data. Extension to multi-class classification is feasible using pairwise classifications (e.g. Maritime Surveillance case study in section 6). We assume that we are given two sets of traces X^0 (those with label 0) and X^1 (those with label 1). The main steps in Algo. 2 are as follows:

The function *isNew* checks whether the PSTL formula φ produced by Algo. 1 is heuristically equivalent to an existing formula. Internally, this is done by checking if the signature of the new formula is identical to the signature of an existing formula in the database of formulas, i.e. *DB*. In the next step, algorithm tries to obtain a point on the satisfaction boundary of the enumerated formula $\varphi(\mathbf{p})$ that results in a formula that serves as a good classifier. To explain this procedure further, we first recall the algorithm to approximate the satisfaction boundary of a given formula $\varphi(\mathbf{p})$.

Algorithm 2: Logical classification

Input: $\varphi, X^0, X^1, \delta, threshold, DB$

if $isNew(\varphi)$ **then**
  while $boundaryPrecision < \delta$ **do**
    // Selects a candidate point on satisfaction boundary
    $v^*(\mathbf{p}) \leftarrow pointOnBoundary(\varphi, X^1)$;
    // Replaces candidate point on satisfaction boundary in φ
    $\varphi^* \leftarrow \varphi(v^*(\mathbf{p}))$;
    // Compute Boolean satisfaction for traces with label 0 and 1
    $falsePos \leftarrow |\{x \mid x \in X^0 \wedge x \vDash \varphi^*|\}$;
    $trueNeg \leftarrow |\{x \mid x \in X^1 \wedge x \nvDash \varphi^*|\}$;
    $MCR \leftarrow (falsePos + trueNeg)/(|T^0| + |T^1|)$;
    if $MCR < threshold$ **then return** φ, MCR ;

Function $isNew(\varphi)$**:**
  $S_\varphi \leftarrow computeSignature(\varphi)$;
  // Compare signature with formula signatures in the database
  if $S_\varphi \in DB$ **then return** $false$;
  else
    // Add signature to database
    $Add(DB, \varphi, S_\varphi)$;
    return $true$;

Function $computeSignature(\varphi)$**:**
  // Randomly choose n traces
  $X_n \leftarrow selectRandom(X^0, X^1, n)$;
  // Randomly sample m parameter values from parameter space
  $\mathcal{D}_{\mathcal{P}^m} \leftarrow sampleRandom(\mathcal{D}_\mathcal{P}, m)$;
  for $j \leftarrow 1$ *to* m **do**
    $\varphi_m \leftarrow setParams(\varphi, \mathcal{D}_{\mathcal{P}^m}(j))$;
    for $i \leftarrow 1, n$ **do**
      $S_\varphi(i, j) \leftarrow \rho(\varphi_m, X_n(i), 0)$;

First, we recall from [29] that the satisfaction boundary of a formula $\varphi(\mathbf{p})$ with respect to a set of traces X is essentially the set of parameter valuations $v(\mathbf{p})$ where the robustness value of $\varphi(v(\mathbf{p}))$ for at least one trace is 0, i.e. the formula is marginally satisfied by at least one trace in X.

In general, computing the satisfaction boundary for arbitrary PSTL formulas is difficult; however, for formulas in the monotonic fragment of PSTL [29], there is an efficient procedure to approximate the satisfaction boundary [20]. The procedure in [20] recursively approximates the satisfaction boundary to an arbitrary precision by performing binary search on diagonals of sub-regions within the parameter space. The idea is that in an m-dimensional parameter space, a parameter valuation on the diagonal corresponds to a formula with zero robustness value. This point subdivides the parameter space into 2^m distinct regions: one where all valuations correspond to formulas that are valid over all traces, one where all valuations correspond to formulas that are invalid, and $2^m - 2$ regions where satisfaction is unknown. The algorithm then proceeds to search along the diagonals of these remaining regions. This approximation results in a

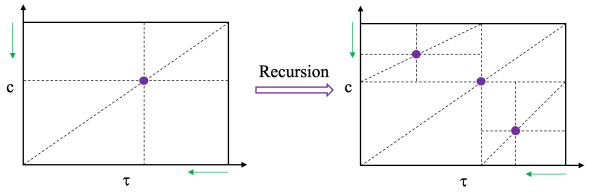

Fig. 3. Method to recursively approximate satisfaction boundary to arbitrary precision δ. Green arrows indicate the monotonicity direction (both decreasing).

series of overlapping axis-aligned hyper-rectangles guaranteed to include the satisfaction boundary [29]. More details of this procedure can be found in [20]. We visualize an instance of the method in Fig. 3.

We combine the procedure from [20] with a classification algorithm as follows. In each recursive iteration of the multi-dimensional binary search, the algorithm identifies a point on the satisfaction boundary of $\varphi(\mathbf{p})$ with respect to the 1-labeled traces, i.e. X^1. Let this point be denoted $v^*(\mathbf{p})$, and the resulting STL formula $\varphi(v^*(\mathbf{p}))$ be denoted as φ^* in short-hand. We then check the Boolean satisfaction of φ^* on the traces in X^0. The misclassification rate (*MCR*) is computed as follows:

$$MCR = \frac{|\{x \mid x \in X^0 \wedge x \vDash \varphi^*\}| + |\{x \mid x \in X^1 \wedge x \nvDash \varphi^*\}|}{|X^0| + |X^1|}.$$

If *MCR* is less than the specified threshold (threshold = 0.1 in our implementation), the algorithm terminates, and φ^* is returned as the binary STL classifier for traces X^0 and X^1. Otherwise, the algorithm goes to the next recursive computation of a boundary point (in another region of the parameter space). If the size of the parameter-space sub-region being searched (denoted by the variable *boundaryPrecision* in Algo. 2) exceeds a user-specified δ, the algorithm returns control to Algo. 1, which proceeds to enumerate the next PSTL formula for consideration.

Algo. 1 continues execution until a PSTL classifier φ with $MCR < threshold$ is found or the length of enumerated PSTL formula exceeds the *maxLength* defined by user, which means no PSTL classifier with $MCR < threshold$ can be found.

6 CASE STUDIES

To evaluate the new framework, we apply our method on various case studies from the automotive and transportation domains[1]. The results show that the employed technique has a number of advantages compared to previous methods. The software to reproduce the following results is available at : https://github.com/saramohammadinejad/learningSTL.

Maritime Surveillance. We first consider the Maritime surveillance problem presented in [5] to compare our inference technique with their Decision Tree tool (DTL4STL) [5]. The synthetic data set employed in [5] is 2-dimensional and consists of one normal and two types of anomalous trajectories. In order to make a fair comparison, we used the exact data set from [5], as illustrated in Fig. 4. We used 100 traces for training and, 100 traces were reserved for testing. Our enumerative solver could learn three STL formulas to classify three kinds of trajectories. The STL formula learned to classify normal traces (green traces) from two kinds of anomalous traces (red and blue traces) is $(y(t) \geq 19.7398)\mathbf{U}_{[0,49.2247]}(x(t) \leq 24.8640)$ with training MCR = 0.01 and testing MCR = 0.05. The total training time is 2699.80 seconds (with signature) and 3903.02 seconds (without signature). Hence, signature-based optimization

[1]We run the experiments on an Intel Core-i7 Macbook Pro with 2.7 GHz processors and 16 GB RAM. While there are platform differences between our experiments and [5] and [16], our results give comparable or favorable runtimes.

could improve the training time for above 30 %. The STL formula learned to classify red traces from the others is $\mathbf{F}_{[0,10.9071]}(y[t] \leq 28.8430)$ (training MCR = 0.01, testing MCR = 0.02, training time = 48.73 seconds(with signature) and 53.78 seconds (without signature)). Finally, our enumerative solver could classify blue traces from the others with training and testing MCRs equal to 0 and 0.02, respectively. The learned STL formula is $G_{[0,60]}(x[t] \geq 36.3260)$ with training time of 23.17 seconds (with signature) and 28.19 seconds (without signature). The simplest STL formula learned by DTL4STL [5] to classify green traces from the others is:

$$\varphi = (\varphi_1 \wedge (\neg\varphi_2 \vee (\varphi_2 \wedge \neg\varphi_3))) \vee (\neg\varphi_1 \wedge (\varphi_4 \wedge \varphi_5))$$
$$\varphi_1 = \mathbf{G}_{[199.70,297.27)}(\mathbf{F}_{[0.00,0.05)}(x[t] \leq 23.60)$$
$$\varphi_2 = \mathbf{G}_{[4.47,16.64)}(\mathbf{F}_{[0.00,198.73)}(y[t] \leq 24.20)$$
$$\varphi_3 = \mathbf{G}_{[34.40,52.89)}(\mathbf{F}_{[0.00,61.74)}(y[t] \leq 19.62)$$
$$\varphi_4 = \mathbf{G}_{[30.96,37.88)}(\mathbf{F}_{[0.00,250.37)}(x[t] \leq 36.60)$$
$$\varphi_5 = \mathbf{G}_{[62.76,253.23)}(\mathbf{F}_{[0.00,41.07)}(y[t] \leq 29.90)$$

with the average misclassification rate of 0.007. This STL formula is long and complicated compared to the STL formula φ_{green} learned by our framework. Thus, [5] achieves better MCR, at the price of a highly uninterpretable formula. Our technique considers the space of all PSTL formulas in increasing order of complexity which results in simple and interpretable STL classifiers. However, DTL4STL tool [5] is only restricted to eventually and globally as PSTL templates which is the reason for generating long and complicated STL classifiers.

STL classifiers for higher-dimensional signals will be more complicated and hence less interpretable. In future work, we will consider dimensionality reduction techniques like PCA or ICA to first identify the important dimensions. An inherent advantage of systematic enumeration is that we encounter formulas in increasing order of complexity, thereby learning from as few features as possible. For instance, in this case study, two of learned formulas only use 1 out of 2 available features.

Linear System. We now compare our technique with another supervised learning technique, and for a fair comparison, we use the same models as [16]. In [16], the authors use the following dynamic system model to benchmark the performance of their supervised learning procedure.

$$\dot{x} = \begin{cases} 0.03 \cdot x + w, & attack = 0 \text{ or normal} \\ -0.03 \cdot x + w, & attack = 1 \text{ or anomaly} \end{cases} \tag{5}$$

Here, $y(t) = x(t)$ is the observation, and $w(t)$ is a white noise with 0.04 variance. The system was modeled in Simulink and interfaced with Breach [6] to simulate the data. We generated 100 time-series traces of the system for the two different system modes, resulting in a total of 200 time-series traces. Fig. 5 shows the results of simulation. Green traces represent normal behaviors (absence of attack), and red traces represent the behavior of the system under attack. 50% of the data was split for training (50 normal and 50 anomaly), and the remaining data was reserved for testing. The enumerative solver was trained and tested on this data to extract an STL formula. The dashed blue line in the figure shows the threshold (=0.9736) of the learned formula: $\mathbf{G}_{[0,3]}(x(t) \geq 0.9736)$. This is a simpler formula compared to the one obtained in [16], which is $\mathbf{F}_{[0,3.0)}(\mathbf{G}_{[0.5,2.0)}(y > 0.9634))$. Our procedure takes 39.05s (with signature-based

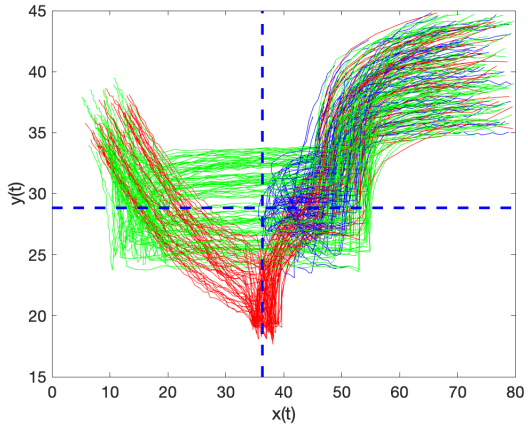

Fig. 4. Naval surveillance data set [5] (Green traces: normal trajectories, red and blue traces: two kinds of anomalous trajectories and dash lines: the thresholds of the STL learned by enumerative solver (= 36.3260, 28.8430)).

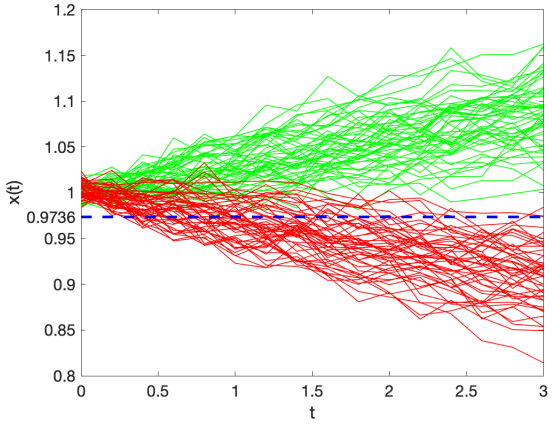

Fig. 5. Simulation results of the linear system (Green traces: normal operation of the system, red traces: anomalous behavior of the system and dash line: the threshold of the STL learned by enumerative solver (= 0.9736)).

pruning) and 44.25s (without signatures) with training $MCR = 0$ and testing $MCR = 0.02$, while the MCR for training and test is 0 and 0.01 in [16]. Hence, We learn a simpler formula with comparable accuracy.

Cruise Control of Train. We also benchmark an example for cruise control in a train from [16]. The system consists of a 3-car train, where each car has its own electronically-controlled pneumatic (ECP) braking mechanism. The velocity of the entire train is modeled as a single system. Hence, there are a total of 4 models (1 for velocity + 3 for braking in each car). The train system is constructed as a hybrid system/automata model, having continuous and discrete transitions for the velocity system, dynamics shown in Figure 6. Similarly, the ECP braking system of each car is modeled as a hybrid system, with its dynamics shown in Figure 7.

The definitions of the parameters used in the above train cruise control models are as follows:

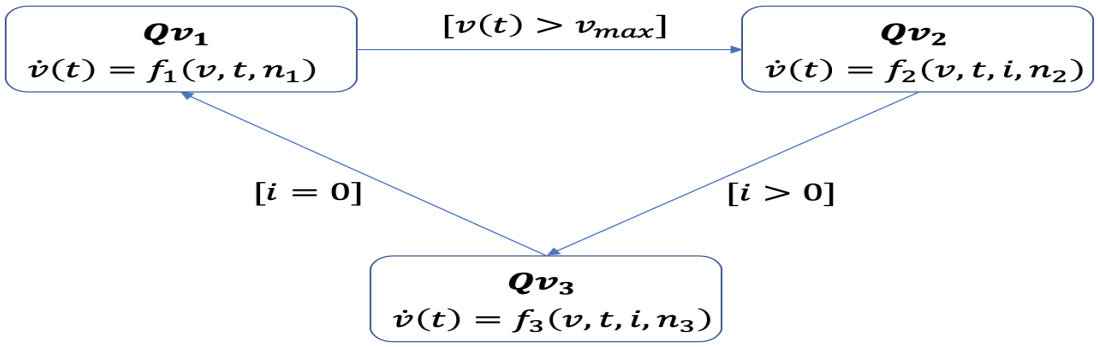

Fig. 6. Train velocity system.

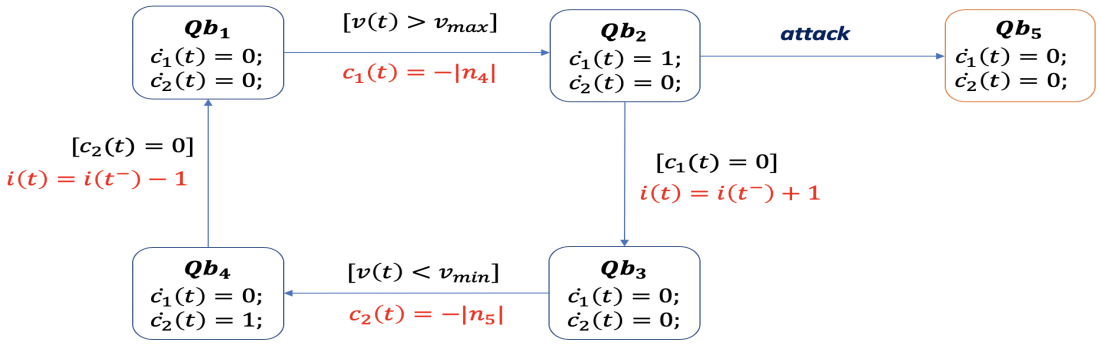

Fig. 7. ECP braking subsystem of each train car.

- t represents the time instance.
- $v(t)$ is the velocity at time instant.
- c_1 and c_2 are clock variables to maintain a counter.
- i is the number of brakes engaged, and is an integer in the interval $[0, 3]$.
- v_{max} and v_{min} are the upper and lower velocity limits, respectively. In our experiments, we use $v_{max} = 28.5m/s$, $v_{min} = 20m/s$.
- Qv_1, Qv_2 and Qv_3 represent the different velocity states. Qb_1, Qb_2, Qb_3, Qb_4 and Qb_5 are the braking states. Qb_5 state is entered when there is an attack on the system, in which case, the number of brakes engaged is 0.
- $n_1 \sim N(0, 1)$, $n_2 \sim N(0, 0.1)$, $n_3 \sim N(0, 0.3)$, $n_4 \sim N(0, 3)$ and $n_5 \sim N(0, 3)$ are the Gaussian noise variables.

The observations/readings of the train velocity is assumed to be noisy. The cruising speed of the train is set to $25m/s$ and the train oscillates about this speed by $\pm 2.5m/s$. Under normal conditions, the train maintains the cruising speed and applies its brakes when the speed exceeds a threshold/upper limit. In an anomalous situation (or attack), all the brakes fail to engage and hence the train fails to maintain its speed within the desired/set limits. The velocity parameters were set to be in the interval $[0, 30]m/s$. For this system, we generated data from 200 simulations (100 for normal and

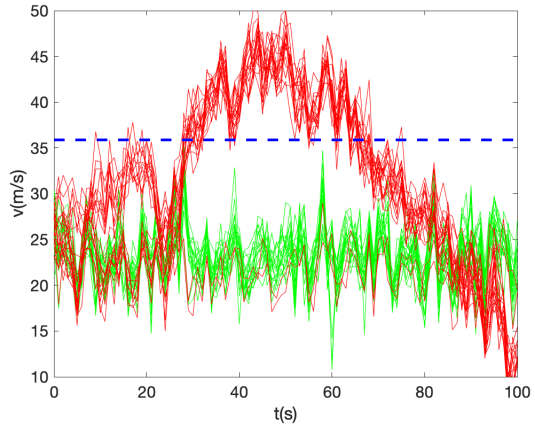

Fig. 8. Simulation results of cruise control of the train (Green traces: normal operation, red traces: anomalous behavior and dash line: the threshold of the STL learned by enumerative solver (= 35.8816)).

100 for anomaly behavior). The time-series data corresponding to the simulations is shown in Fig. 8, in which the green traces represent normal behaviors, while red traces represent anomalies. Since Breach [6] could possibly choose a low initial velocity during an anomaly situation, a very low percentage of the traces tend to exhibit normal behaviors, as seen from the figure. Similar to our previous approaches, the data was split 50% (50 normal and 50 anomaly traces) for training and the rest for testing.

We applied our solver to extract the STL formula: $\mathbf{G}_{[0,100]}(x(t) < 35.8816)$, where the threshold learned by our solver is 35.8816 (shown by the dashed blue line in the figure). The *MCR*s for training and testing are 0.05 and 0.02 respectively. The STL formula obtained by our approach is simpler compared to the one extracted in [16], which is: $\mathbf{F}_{[0,100)}(\mathbf{F}_{[10,69)}(y < 24.9) \wedge \mathbf{F}_{[13.9,44.2)}(y > 17.66)))$ with $MCR = 0$. The execution time of our approach is 32.31s (with signatures) and 35.84s (without signatures).

PID controller. Given a deterministic system $S(t)$ and a specification ϕ, the goal is to mine the subset of initial states or inputs that ensure satisfaction of an STL formula φ on the output of the model. In other words, we want to know what STL formulas do the inputs satisfy that guarantee that the outputs will be desirable. In our experiments, we try to learn the STL formulas that separates good inputs from bad inputs. For our experiment, we consider a PID controller for a damped second-order continuous system. The desired output of this system is to observe that the output has settled or is oscillating. We use 31 input traces with periods: 10, 12, 14, ..., 70 and their negations. The traces are shown in Fig. 9. For oscillating outputs, the STL formula learned by the solver is: $\mathbf{G}_{[0,368]}(\mathbf{F}_{[0,21]}(x[t] \geq 1))$. It means for periods below $21 \times 2 = 42$, the output does not settle. This is to be expected, as this period is less than the settling time of the system. The time required for learning is 126.66s (with signatures) and is 158.70s (without signatures) with both training and testing $MCR = 0$.

7 RELATED WORKS

There has been considerable recent work on learning STL formulas from data for various applications such as supervised learning [5, 9, 18, 24], clustering [28, 29], or anomaly detection [16].

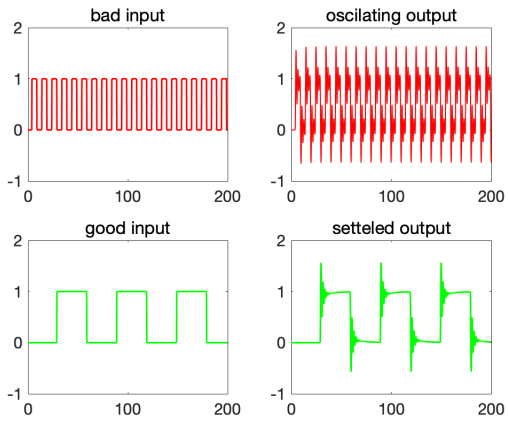

Fig. 9. PID controller (red: bad input leads to undesirable output, green: good input leads to desirable output).

In [18], a fragment of PSTL (rPSTL or reactive parametric signal temporal logic) is defined to capture causal relationships from data. However, there are some temporal properties namely, *concurrent eventuality* and *nested always eventually* that cannot be described directly in rPSTL. In [16], the authors extend [18] by using a fragment of rPSTL, inference parametric STL (iPSTL), that does not require a causal structure. In this work, classical ML algorithms (one-class support vector machines) are applied for unsupervised learning problem. In [5], a decision tree based method is employed to learn STL formulas, which creates a map between a restricted fragment of STL and a binary decision tree in order to build a STL classifier. While this seminal work has advanced work in the intersection of formal methods and machine learning, one disadvantage of these approaches has been that they lead to long formulas which can become an issue for interpretability.

The work by Nenzi et al. [24] uses genetic algorithms to learn the STL formula structure, where longer formulas are obtained in a new generation by combining formulas from a previous generation. While this procedure is able to learn interesting formulas, a key difference is that it does not guarantee that the shortest formula will be learned, and does not check for equivalent formulas (beyond what is encoded by the fitness function of the GP algorithm). In the area of learning programs using genetic programming, grammars are used for delineating the search space and avoiding unproductive search in infeasible regions [10, 23, 30].

The closest work to our approach appears in [9], where the authors propose a similar systematic enumeration of all parametric formulas. In this recent work, the authors discuss the problem of online monitoring and formula synthesis for a past fragment of STL. However, the proposed work in [9] does not use signature-based optimization to prevent enumerating repeated formulas; this is one of the key contributions of our paper. Furthermore, [9] assumes a past-only fragment of STL, while our technique allows future operators. Finally, [9] does not provide details on how parameter valuations are explored, except for language indicating a grid-based parameter space exploration. We use a systematic procedure to do this: we compute the validity domain boundary of the enumerated PSTL formula in an on-the-fly and on-demand fashion. I.e., we use a multi-dimensional binary search procedure to estimate the validity domain boundary; however, we do not have to compute the boundary to a given precision. As we explore parameter valuations on the validity domain boundary, our procedure terminates when it encounters sufficiently low misclassfication rate for a given valuation. This allows us a more systematic and efficient exploration of the parameter space.

TeLEx [12] is a novel technique that addresses the problem of learning STL formulas from just positive example traces. A novel technique is proposed to automatically learn the structure of the STL formula by incrementally constructing more complex formula guided by the robustness metric of subformula. However, in spite of systematic enumeration technique, this method does not guarantee to learn the simplest STL formulas.

In template-based techniques, a fixed PSTL template is provided by the user, and the techniques only learn the values of parameters associated with the PSTL. In [28], a total ordering on parameter space of PSTL specifications is utilized as feature vectors for learning logical specifications. Unfortunately, recognizing the best total ordering is not straightforward for users. In [29], the authors eliminate this additional burden on the user by suggesting a method that maps timed traces to a surface in the parameter space of the formula, and then employing these curves as features. In [15], the input to the algorithm is a requirement template expressed in PSTL, where the traces are actively generated from a model of the system.

Our proposed technique, which uses systematic enumeration, can produce smaller formulas which may be more human-interpretable, and with higher accuracy($\geq$ 95% in all investigated case studies).

8 CONCLUSION

We proposed a new technique for multi-class classification of time-series data using Signal Temporal Logic formulas. The key idea is to combine an algorithm for systematic enumeration of PSTL formulas with an algorithm for estimation of the satisfaction boundary of the enumerated PSTL formula. We also investigate an optimization using formula signatures to avoid enumerating equivalent PSTL formulas. We then illustrate this technique with a number of case studies on real world data from different domains. The results show that the enumerative solver has a number of advantages. As future work, we will extend this approach to unsupervised, semi-supervised, and active learning. We will also investigate other optimization techniques to make the enumerative solver faster and more memory-efficient.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers and the paper shepherd who meticulously read our paper and gave us valuable feedback. We would also like to thank Dr. Abhishek Udupa for early discussions on signature-based pruning in enumerative learning for STL. The authors gratefully acknowledge the support of the National Science Foundation under the FMitF grant CCF-1837131.

REFERENCES

[1] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*. IEEE, 1–8.

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

[3] Ayca Balkan, Paulo Tabuada, Jyotirmoy V Deshmukh, Xiaoqing Jin, and James Kapinski. 2018. Underminer: A framework for automatically identifying nonconverging behaviors in black-box system models. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 1 (2018), 20.

[4] Marcello Maria Bersani, Matteo Rossi, and Pierluigi San Pietro. 2013. Deciding the satisfiability of MITL specifications. *arXiv preprint arXiv:1307.4469* (2013).

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

[6] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 167–170.

[7] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*. 92–106.

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

[9] Ebru Aydin Gol. 2018. Efficient online monitoring and formula synthesis with past stl. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE, 916–921.

[10] Paul Holmes and Peter J Barclay. 1996. Functional languages on linear chromosomes. In *Proceedings of the 1st annual conference on genetic programming*. 427–427.

[11] Hisao Ishibuchi and Takashi Yamamoto. 2003. Interpretability issues in fuzzy genetics-based machine learning for linguistic modelling. In *Modelling with Words*. Springer, 209–228.

[12] Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. 2017. Telex: Passive STL learning using only positive examples. In *International Conference on Runtime Verification*. Springer, 208–224.

[13] Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. 2019. TeLEx: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design* (2019), 1–24.

[14] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. 2014. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*. ACM, 253–262.

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

[16] Austin Jones, Zhaodan Kong, and Calin Belta. 2014. Anomaly detection in cyber-physical systems: A formal methods approach. In *53rd IEEE Conference on Decision and Control*. IEEE, 848–853.

[17] James Kapinski, Xiaoqing Jin, Jyotirmoy Deshmukh, Alexandre Donze, Tomoya Yamaguchi, Hisahiro Ito, Tomoyuki Kaga, Shunsuke Kobuna, and Sanjit Seshia. 2016. *ST-Lib: a library for specifying and classifying model behaviors*. Technical Report. SAE Technical Paper.

[18] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. 2014. Temporal logic inference for classification and prediction from data. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*. ACM, 273–282.

[19] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. 2016. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 1675–1684.

[20] Oded Maler. 2017. Learning Monotone Partitions of Partially-Ordered Domains (Work in Progress). (2017).

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

[22] Oded Maler and Dejan Ničković. 2013. Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer* 15, 3 (2013), 247–268.

[23] Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael OâĂŹneill. 2010. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines* 11, 3-4 (2010), 365–396.

[24] Laura Nenzi, Simone Silvetti, Ezio Bartocci, and Luca Bortolussi. 2018. A robust genetic algorithm for learning temporal specifications from data. In *International Conference on Quantitative Evaluation of Systems*. Springer, 323–338.

[25] Ran Raz and Amir Shpilka. 2005. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity* 14, 1 (2005), 1–19.

[26] Szymon Stoma, Alexandre Donzé, François Bertaux, Oded Maler, and Gregory Batt. 2013. STL-based analysis of TRAIL-induced apoptosis challenges the notion of type I/type II cell line classification. *PLoS computational biology* 9, 5 (2013), e1003056.

[27] Abhishek Udupa, Arun Raghavan, Jyotirmoy V Deshmukh, Sela Mador-Haim, Milo MK Martin, and Rajeev Alur. 2013. TRANSIT: specifying protocols with concolic snippets. *ACM SIGPLAN Notices* 48, 6 (2013), 287–296.

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

[29] Marcell Vazquez-Chanlatte, Shromona Ghosh, Jyotirmoy V Deshmukh, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2018. Time-Series Learning Using Monotonic Logical Properties. In *International Conference on Runtime Verification*. Springer, 389–405.

[30] Peter A Whigham et al. 1995. Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: from theory to real-world applications*, Vol. 16. 33–41.

[31] Zhe Xu, Calin Belta, and Agung Julius. 2015. Temporal logic inference with prior information: An application to robot arm movements. *IFAC-PapersOnLine* 48, 27 (2015), 141–146.