Learning Spatio-Temporal Specifications for Dynamical Systems

Suhail Alsalehi\(^1\)  
Erfan Aasi\(^2\)  
Ron Weiss\(^3\)  
Calin Belta\(^1,2\)

\(^{1}:\) Division of Systems Engineering, Boston University, Boston, MA, USA  
\(^{2}:\) Mechanical Engineering Department, Boston University, Boston, MA, USA  
\(^{3}:\) Biological Engineering Department, Massachusetts Institute of Technology, Cambridge, MA, USA

Editors: R. Firoozi, N. Mehr, E. Yel, R. Antonova, J. Bohg, M. Schwager, M. Kochenderfer

Abstract

Learning dynamical systems properties from data provides valuable insights that help us understand such systems and mitigate undesired outcomes. We propose a framework for learning spatio-temporal (ST) properties as formal logic specifications from data. We introduce Support-Vector Machine-Signal Temporal Logic (SVM-STL), an extension of Signal Temporal Logic (STL), capable of specifying spatial and temporal properties of a wide range of systems exhibiting time-varying spatial patterns. Our framework utilizes machine learning techniques to learn SVM-STL specifications from system executions given by sequences of spatial patterns. We present methods to deal with both labeled and unlabeled data. In addition, given system requirements in the form of SVM-STL specifications, we provide an approach for parameter synthesis to find parameters that maximize the satisfaction of such specifications. Our learning framework and parameter synthesis approach are showcased in an example of a reaction-diffusion system.

Keywords: Dynamical Systems, Inference and Parameter Synthesis, Temporal Logics

1. Introduction and Related Works

Many dynamical systems exhibit time-varying spatial behaviors. Smart cities, robotics swarms, and multicellular biological systems are just a few examples. With the increasing complexity of such systems, there is a need for formal ways to describe their spatial and temporal properties. To be deemed useful, these properties must be interpretable for humans and amenable to rigorous mathematical analysis. Two of the main challenges are inferring such properties from data (the inference problem) and synthesizing system input parameters such that certain properties are met (the parameter synthesis problem). This work explores both problems for dynamical systems.

Machine learning and formal logics are two fields with research efforts on the inference problem. Deep Neural Networks (DNN) have shown success in inferring properties from data (feature extraction). However, the inferred properties lack interpretability and can only be used by machine learning models for tasks such as classification. On the other hand, due to the expressivity and readability, formal logics are widely used for specifying spatial and temporal properties of dynamic systems. Inferring formal logic specifications from system executions has been explored in the literature, e.g., Asarin et al. (2011); Hoxha et al. (2018); Bombara et al. (2016); Vazquez-Chanlatte et al. (2017); Jha et al. (2019); Yan et al. (2019); Fan et al. (2020); Mohammadinejad et al. (2021); Xu et al. (2019); Baharisingh et al. (2021); Yan et al. (2019); Li et al. (2020).

Spatio-temporal (ST) logics are formal languages capable of specifying ST properties of dynamical systems Haghighi et al. (2015); Ma et al. (2020); Li et al. (2020); Bartocci et al. (2017);
Alsalehi et al. (2021); Yan et al. (2019). A common theme among these works is combining spatial logics with temporal logics to produce ST logics. For example, the authors of Mehdipour et al. (2018) nest spatial properties in STL formulae by defining predicates as geometric distances to hyperplanes of SVM classifiers, providing a qualitative valuation for spatial patterns. The literature on ST logics focuses on time-varying spatial patterns given by graphs, e.g. Bartocci et al. (2017), quadtrees Haghighi et al. (2015), abstractions of systems, e.g. Li et al. (2020) and so on. However, there are no works on specifications for time-varying spatial patterns given by images.

To address this concern, we introduce Support-Vector Machine-Signal Temporal Logic (SVM-STL), a logic capable of describing ST properties of dynamical systems. SVM-STL nest spatial properties of images into STL by defining machine learning-based predicates that automate feature extraction from ST trajectories (sequences of spatial patterns generated by executions of a system). SVM-STL is equipped with qualitative semantics that describes whether a trajectory satisfies a SVM-STL formula or not, as well as quantitative semantics, which quantifies the degree of satisfaction of a formula by a ST trajectory.

We provide a framework for learning SVM-STL formulae from ST trajectories by separating the learning of spatial properties from the learning of temporal properties. First, we ignore the temporal aspect of the data and utilize machine learning techniques to learn predicates that capture the spatial properties in the data. The predicates are a combination of SVM binary classifiers and a neural network model for automated feature extraction from images. Then, we utilize a decision tree-based algorithm Aasi et al. (2021) to learn SVM-STL formulae from ST trajectories. We also provide an unsupervised learning approach to learning SVM-STL formulae from unlabeled ST trajectories. To the best of our knowledge, this is the first framework for learning formal logics specifications from ST trajectories, where spatial patterns are given by images.

Synthesizing parameters from spatio-temporal specifications has been the focus of several works in the literature, e.g. Haghighi et al. (2019); Liu et al. (2018); Bozkurt et al. (2020); Alsalehi et al. (2021). In this work, we introduce our approach to parameter synthesis for systems with spatial and temporal requirements. Our approach is unique in that we can learn requirements from executions with the desired behavior. The efficacy of the learning framework and the parameter synthesis approach are showcased in a case study of a reaction-diffusion system.

2. Preliminaries and Notations

Let \( \mathbb{R}, \mathbb{Z}, \mathbb{Z}_{\geq 0} \) be the set of real numbers, integers, and non-negative integers, respectively. Given \( a, b \in \mathbb{Z}_{\geq 0} \), with slight abuse of notation, we write \([a, b] = \{k \in \mathbb{Z}_{\geq 0} | a \leq k \leq b\} \). A signal \( s \) is a function \( s : T \rightarrow \mathbb{R}^n \) that maps each discrete time point \( k \in T = [0, T], T \in \mathbb{Z}_{\geq 0} \), to a n-dimensional real-valued vector \( s[k] \in \mathbb{R}^n \), \( n \in \mathbb{Z}_{\geq 0} \). A RGB image (image) is a phenotypical observation of a system at a fixed time point. RGB is an additive color model Hirsch (2005) in which the channels red, green, and blue are added together in various ways to reproduce a broad range of colors. An image is denoted by \( I \in \mathbb{R}^{L \times W \times C} \), where \( L, W, C \in \mathbb{Z}_{\geq 0} \) are the image length, width and channels. A spatio-temporal (ST) trajectory \( S \) is a function \( S : T \rightarrow \mathbb{R}^{L \times W \times C} \) that maps each discrete time point \( k \in T \), to an image \( S[k] \in \mathbb{R}^{L \times W \times C} \).

2.1. Convolutional Neural Network (CNN)

CNN is a deep learning algorithm commonly used for analyzing images. A simple CNN consists of one or more convolutional and fully-connected layers. Convolutional layers extract the high-level features from images, while fully connected layers learn classifiers, regression models, etc. Transfer learning Pan and Yang (2009), i.e., storing knowledge gained while solving one problem and applying it to a different but related problem, is often done to save time, computational power,
and/or when there are not enough examples to train a new model. A convolutional layers of a CNN that is trained on a broad set of classes (e.g., AlexNet Krizhevsky et al. (2012), VGG16 Simonyan and Zisserman (2014) and Inception Szegedy et al. (2015)) are good feature extractors Albashish et al. (2021); Shijie et al. (2017); Hertel et al. (2015); Mahdianpari et al. (2018). Next, we use $f_{cnn}$ to denote a pre-trained CNN (feature extractor), where $f_{cnn} : \mathbb{R}^{L \times W \times C} \to \mathbb{R}^m$, $m \in \mathbb{Z}_{\geq 0}$.

3. SVM-STL Specifications

We introduce SVM-STL, an extension of STL Maler and Nickovic (2004), capable of specifying spatial and temporal properties of dynamical systems such as “(eventually in the time interval [0,30] pattern 1 is observed for 10 time steps) AND (Never in time interval [0,30] pattern 2 is observed)”(see example in Fig. 2). The syntax of SVM-STL formulae is defined over $S$ as

$$\varphi := \top \mid \mu_j \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid F_{[a,b]} \phi \mid G_{[a,b]} \phi$$

where $\varphi, \varphi_1, \varphi_2$ are STL formulae, $\top$ is Boolean True; $\mu_j$ is a predicate of the form $\mu_j := h_j(S[k]) \sim r$, defined over components of ST trajectories $S$, where $h_j : \mathbb{R}^{L \times W \times C} \to \mathbb{R}$, $j = 1, \ldots, n$ is a predicate function, $\sim \in \{>, \leq\}$ and $r \in \mathbb{R}$ is a threshold; $\neg, \land, \lor$ are the logical operators negation, conjunction, and disjunction, respectively; and $F_{[a,b]} \phi, G_{[a,b]} \phi$ are the temporal operators eventually and always, with $a, b \in \mathbb{Z}_{\geq 0}$.

A predicate function $h_j$ is a classifier that maps images to real-values corresponding to how strong images belong to (spatial) class $j$. The mathematical structure of $h_j$ is introduced in Sec. 4.2.

SVM-STL is equipped with qualitative and quantitative semantics. The qualitative semantics describes satisfaction-violation of a SVM-STL formula $\varphi$ at time $k$ by a ST trajectory $S$ and we use $S[k] \models \varphi$ to denote (Boolean) satisfaction. The qualitative semantics is given recursively by:

\[
\begin{align*}
(S, k) \models \mu_j & \quad \Leftrightarrow h_j(S[k]) \sim r \\
(S, k) \models \neg \varphi & \quad \Leftrightarrow \neg((S, k) \models \varphi) \\
(S, k) \models \varphi_1 \land \varphi_2 & \quad \Leftrightarrow (S, k) \models \varphi_1 \land (S, k) \models \varphi_2 \\
(S, k) \models \varphi_1 \lor \varphi_2 & \quad \Leftrightarrow (S, k) \models \varphi_1 \lor (S, k) \models \varphi_2 \\
(S, k) \models F_{[a,b]} \varphi & \quad \Leftrightarrow \exists k' \in [k+a, k+b] \; \text{s.t.} \; (S, k') \models \varphi \\
(S, k) \models G_{[a,b]} \varphi & \quad \Leftrightarrow \forall k' \in [k+a, k+b], (S, k') \models \varphi
\end{align*}
\]

(1)

Similar to Donzé and Maler (2010), the quantitative semantics is given by the real-valued robustness function $\rho(S, \varphi, k)$, which captures the degree of satisfaction of a formula $\varphi$ by a ST trajectory $S$. Specifically, a positive robustness score ($\rho(S, \varphi, k) \geq 0$) implies satisfaction $S \models \varphi$, while negative robustness ($\rho(S, \varphi, k) < 0$) implies violation. Given a formula $\varphi$ and a ST trajectory $S$, the robustness at time $k$ is recursively defined as follows:

\[
\begin{align*}
\rho(S, \mu_j, k) & = \begin{cases} h_j(S[k]) - r, & \text{if } \mu := h_j(S[k]) > r \\
r - h_j(S[k]), & \text{otherwise} \end{cases} \\
\rho(S, \neg \varphi, k) & = - \rho(S, \varphi, k) \\
\rho(S, \varphi_1 \land \varphi_2, k) & = \min(\rho(S, \varphi_1, k), \rho(S, \varphi_2, k)) \\
\rho(S, \varphi_1 \lor \varphi_2, k) & = \max(\rho(S, \varphi_1, k), \rho(S, \varphi_2, k)) \\
\rho(S, F_{[a,b]} \varphi, k) & = \max_{k' \in [k+a, k+b]} \rho(S, \varphi, k') \\
\rho(S, G_{[a,b]} \varphi, k) & = \min_{k' \in [k+a, k+b]} \rho(S, \varphi, k')
\end{align*}
\]
Inspired by Asarin et al. (2011), we define **Parametric SVM-STL (PSVM-STL)**, which is an extension of SVM-STL, where the time bounds $a, b$ of temporal operators and threshold $r$ of the predicate are parameters. The set of all possible valuations of all parameters in a PSVM-STL formula $\varphi$ is called the parameter space and is denoted by $\Theta$. A particular valuation of a PSVM-STL formula $\varphi$ at $\theta \in \Theta$ is denoted by $\varphi_\theta$.

**PSVM-STL primitives** are simple PSVM-STL formulae. We define the set of first-order primitives as $\mathcal{P} = \{ F_{[a, b]}(h_j(S[k])) \sim r), G_{[a, b]}(h_j(S[k])) \sim r) \}$, where $r \in \mathbb{R}$; $a, b, c \in \mathbb{Z}_{\geq 0}$; and $\sim \in \{ \leq, > \}$. The parameters of $\mathcal{P}$ are $(r, a, b)$.

**Weighted SVM-STL** (wSVM-STL) is another extension of SVM-STL (based on Mehdipour et al. (2020)) where robustness degree is modulated by the weights associated with the Boolean and temporal operators. In this paper, we focus on a fragment of wSVM-STL, with weights on conjunctions only, i.e., $\bigwedge_{i=1}^{w} \varphi_i$ and $\bigvee_{i=1}^{w} \varphi_i$. The weight $w = [w_1, ..., w_N]$ assigns a positive weight to each subformula $\varphi_i$. Here, weights capture importance/priorities of conjunctions/disjunctions.

### 4. Learning Spatio-temporal Properties

The **Inference Problem**: Given a set $S = \{S^{(i)}\}_{i=1}^{N_S}$ representing executions of a system (ST trajectories), learn ST properties of the system in the form of SVM-STL formulae.

We provide a framework to infer SVM-STL formulae from system executions in two stages. In the first stage, we construct predicate functions $h_1, ..., h_{n_I}$ that capture the spatial properties in the data. In the second stage, we learn SVM-STL formulae using a decision tree-based approach to capture the ST properties of the data. For learning from unlabeled images and trajectories, unsupervised learning techniques are utilized to cluster and label data. Overall, we divide the learning problem into four sub-problems addressed in subsequent sections: 1) clustering spatial data (images), 2) learning spatial properties, 3) clustering ST trajectories, 4) learning spatio-temporal specifications.

#### 4.1. Clustering Images

Given a set of data points, one can use unsupervised clustering algorithms to organize unlabeled data into $n$ similarity groups called clusters. Clustering requires a distance/similarity measure $d$, a criterion function $f_{\text{crit}}$ and an algorithm to optimize the criterion function. The choice of $n, d, f_{\text{crit}}$ and the clustering algorithm depends on the type of data and purpose of clustering. In this work, we use the k-means clustering algorithm which aims to partition data points into $n$ clusters in which each data point belongs to the nearest mean $\mu_i$ (cluster center) while minimizing the criterion function.

Let $I = \{I^{(i)}\}_{i=1}^{N_I}$ be a set of images. We consider the feature extractor $f_{\text{cn}} : \mathbb{R}^{W \times L \times C} \rightarrow \mathbb{R}$ and criterion function $f_{\text{crit}}$ given by $f_{\text{crit}}(I, \mu) = \sum_{\mu_j \in \mu} \sum_{I \in c_j} d_I(I, \mu_j)$, where $c_j = \{I^{(i)}|j = \arg \min_{j=1,...,n_I} d_I(I^{(i)}, \mu_j)\}$; $\mu = \{\mu_1, ..., \mu_{n_I}\}$, $\mu_1, ..., \mu_{n_I} \in \mathbb{R}^m$; and $d_I$ is the similarity measure given by $d_I(I, \mu_j) = ||f_{\text{cn}}(I) - \mu_j||_2^2$. We want to find the best set of cluster centers $\mu^{\text{best}}$ that minimize the objective function $f_{\text{crit}}^*$.

$$
\mu^{\text{best}} = \arg \min_{\mu} (f_{\text{crit}}(I, \mu));
$$

To find $\mu^{\text{best}}$, we use PSO Kennedy and Eberhart (1995) which optimizes the problem by iterative improvement of a candidate solution according to a criterion function. The PSO-based solution is summarized in Alg. 1, which starts by randomly initializing a set of $K$ particles with positions (parameters) $\pi_k = \{\pi_{k,j}\}$, where $\pi_{k,j} \in \mathbb{R}^m$, $k = 1, ..., K$, and $j = 1, ..., n_I$; and velocities $v_k = \{v_{k,j}\}$, where $v_{k,j} \in \mathbb{R}^m$. Each particle represents a candidate solution to (2). At each iteration, the criterion function is evaluated for $K$ sets of cluster centroids $\pi_k, k = 1, ..., K$. The position
of the \( k \)-th particle with the best set of centers so far is stored in the variable \( \pi_k^{\text{best}} \). Similarly, the position that performed best (lowest \( f_{\text{crit}} \)) among all particles so far is stored in the variable \( \pi^{\text{best}} \). At the end of each iteration, positions and velocities of particles are updated as follows:

\[
v_k \leftarrow Wv_k + \eta(0, r_p)(\pi_k^{\text{best}} - \pi_k) + \eta(0, r_g)(\pi^{\text{best}} - \pi_k) \tag{3}
\]
\[
\pi_k \leftarrow \pi_k + v_k \tag{4}
\]

where \( \eta(a, b) \) generates a random number from a uniform distribution in the interval \([a, b]\) and \( W, r_p, r_g \in \mathbb{R} \) are the PSO hyperparameters \((\text{Kennedy and Eberhart (1995)})\) that are specified by the user. The algorithm keeps iterating until a stopping condition \( \text{Stop} \) is met, e.g. after a certain robustness threshold or if \( \pi^{\text{best}} \) does not change significantly over the last \( z \) iterations. Once the stopping condition is met, each image \( S^{(i)} \) is given a label \( l^{(i)} \). Finally, we construct the new set \( \mathcal{I} = \{(I^{(i)}, l^{(i)})\}_{i=1}^{N_I} \), which consists of images \( I^{(i)} \) and their labels \( l^{(i)} \in \{1, \ldots, n_I\} \). Note that domain knowledge is incorporated to determine the number of clusters \( n_I \) and make decisions to merge or drop certain clusters.

**Algorithm 1** Clustering images using PSO

1. **Input:** \( \mathcal{I} = \{I^{(i)}\}_{i=1}^{N_I}, f_{\text{cnn}}, d_I, n_I, f_{\text{crit}} \text{ (hyperparameters): } W, r_p, r_g, K \)
2. **Initialize:** \( \pi_k^{\text{best}}, \pi^{\text{best}}, \nu_k, k = 1, \ldots, K \) \( \triangleright \) initialize particles
3. **while** \( \neg \text{Stop} \) \( \triangleright \) terminate if stopping condition is met
4.  
5.  
6.  
7.  
8.  
9.  
10. **Return:** \( \mathcal{I} = \{(I^{(i)}, l^{(i)})\}_{i=1}^{N_I} \)

### 4.2. Learning Spatial Properties

In this section, we construct the predicate functions \( h_1, \ldots, h_{n_I} \) that capture the spatial properties in the set of labeled images \( \mathcal{I} = \{(I^{(i)}, l^{(i)})\}_{i=1}^{N_I} \), where \( l^{(i)} \in \{1, \ldots, n_I\} \) is the label of image \( I^{(i)} \). As stated in Sec. 3, a predicate function \( h_j \) takes an image as an input and returns a real-value corresponding to how strongly the image belongs to the class \( j \). We propose a modified version of Support Vector Machine (SVM) as predicate functions.

SVM is a widely used binary classification algorithm, known for its simplicity, high speed, and accuracy in multi-dimensional spaces. Since SVM classifiers do not support tasks with more than two classes, we split the multi-class classification set of samples into multiple binary classification sets (One-vs-Rest) and fit a binary classification model on each. Specifically, we divide the set \( \mathcal{I} \) into \( n_I \) sets \( \mathcal{I}_1, \ldots, \mathcal{I}_{n_I} \) where \( \mathcal{I}_j = \{(I^{(i)}, l_B) | l_B = 1, \text{ if } l^{(i)} = j, \text{ and } -1 \text{ otherwise}\} \), where \( l^{(i)}_B \) is the Boolean label. Thus, problem above is reduced to \( n_I \) binary classification problems.

Next, we consider a set \( \mathcal{I}_j = \{(I^{(i)}, l^{(i)}_B)\}_{i=1}^{N_I} \) of \( N_I \) training samples, where \( I^{(i)} \) is the \( i \)-th observation and \( l^{(i)}_B \in \{-1, 1\} \) is its label. We want to finds a decision boundary of the form \( \omega_j^{(i)} f_{\text{cnn}}(I) + b_j = 0 \) that maximizes the geometric margin between support vectors. Formally, the
optimalization problem can be written as (Gunn et al. (1998)):

$$
\min_{\omega_j} ||\omega_j|| \text{, subject to } l_i(\omega_j^T f_{cnn}(I^{(i)}) + b_j) \geq 1, \quad i = 1, \ldots, N_I.
$$

(5)

This is a convex optimization problem that can be solved using a gradient-based method. The resulting SVM classifier is given by $y_j(I) = \text{sign}(\omega_j^T f_{cnn}(I) + b_j), \text{ with } y_j : \mathbb{R}^{W \times L \times C} \rightarrow \{-1, 1\}$.

We define the predicate function as the signed Euclidean distance from images to the decision boundary of the learned classifiers. Specifically, the predicate function $h_j$ is given by

$$
\begin{align*}
  h_j(I^{(i)}) &= \frac{\omega_j^T f_{cnn}(I^{(i)}) + b_j}{||\omega_j||},
\end{align*}
$$

(6)

where if $h_j(I) > 0$ then $l_j^{(i)} = 1$, otherwise $l_j^{(i)} = -1$.

With slight abuse of notation, we define the operator $h : S \rightarrow s$, which maps ST trajectories $S : \mathbb{T} \rightarrow \mathbb{R}^{E \times W \times C}$ to spatio-temporal (ST) signals $s : \mathbb{T} \rightarrow \mathbb{R}^n$, a signal-like representation.

4.3. Clustering ST trajectories

Let $S = \{S(i)\}_{i=1}^{N_S}$ be a set of ST trajectories and consider the operator $h$. We consider the criterion function $f_{\text{crit}}$ given by $f_{\text{crit}}(S, \mu) = \sum_{\mu_j \in \mu} \sum_{S \in c_j} d_{\text{dtw}}(h(S), \mu_j)$, where $c_j = \{S^{(i)}| j = \text{arg min}_{j=1, \ldots, n_s} (d_{\text{dtw}}(h(S^{(i)}), \mu_j))\}$ and $d_{\text{dtw}} : s \times s \rightarrow \mathbb{R}$ is the dynamic time warping distance - a similarity measure between two one-dimensional temporal sequences Müller (2007). For $n$ dimensional sequences, we rearrange sequences into long one-dimensional sequences.

We follow a similar approach to clustering images (Sec. 4.1) to find the best set of cluster centers $\mu^{\text{best}} = \{\mu_1, \ldots, \mu_{n_I}\}$, where $\mu_1, \ldots, \mu_{n_I} \in \mathbb{R}^{m \times T}$ that minimize the objective function $f_{\text{crit}}$:

$$
\mu^{\text{best}} = \text{arg min}_{\mu} (f_{\text{crit}}(S, \mu)).
$$

(7)

We employ PSO to find the best set of cluster centers that minimizes the objective function $f_{\text{crit}}$. With appropriate changes, we use Alg.1 to construct a new set $S' = \{(S^{(i)}, l^{(i)}_S)\}_{i=1}^{N_S}$ that consists of ST trajectories $S^{(i)}$ and their corresponding labels $l^{(i)}_S \in \{1, \ldots, N_S\}$. A label signifies that a trajectory belongs to the class of that label.

4.4. Learning Spatio-temporal Specifications

Having the labeled ST trajectories, we want to learn ST properties in the form of SVM-STL formulae. Without loss of generality, we will look at a two-class (binary) classification problem. Let $C = \{1, -1\}$ be the set of positive and negative classes. We consider a labeled dataset with $N_s$ samples as $S' = \{(S^{(i)}, l^{(i)}_S)\}_{i=1}^{N_S}$, where $S^{(i)}$ is the $i^{th}$ trajectory and $l^{(i)}_S \in C$ is its corresponding label. Using the predicate functions $h_1, \ldots, h_{n_I}$ and the operator $h : S \rightarrow s$, we map ST trajectories in $S'$ into ST signals, to produce a new dataset $S = \{(s^{(i)}, l^{(i)}_S)\}_{i=1}^{N_S}$.

We desire to learn an SVM-STL specification $\varphi$, based on the SVM-STL primitives $\mathcal{P}$, such that the misclassification rate $\text{MCR}(\varphi)$ defined below is minimized:

$$
\text{MCR}(\varphi) := \frac{|\{s^{(i)}|(s^{(i)} = \varphi \land l^{(i)}_S = -1) \lor (s^{(i)} \not= \varphi \land l^{(i)}_S = 1)\}|}{N_S}; \quad i = 1, \ldots, N_s
$$

(8)

Motivated by the AdaBoost method Shalev-Shwartz and Ben-David (2014) and the boosted concise decision tree method in Aasi et al. (2021), we use a Boosted Decision Tree (BDT) method.
to learn the SVM-STL formulas, explained in Alg. 2. AdaBoost algorithm combines weak classifiers with simple formulae, trained on weighted data samples, where the weights of the data represent the difficulty of correct classification. After training a weak classifier, the weights of the correctly classified samples are decreased and weights of the misclassified samples are increased. In Aasi et al. (2021), they proposed a boosted method, empowered by a set of conciseness techniques, to generate short and simple formulas with promising classification performance (conciseness was not considered in our work).

The BDT method in Alg. 2 takes as input the labeled dataset $S$, the number of decision trees to grow $K$, and the decision tree construction method $E$ as the weak learning algorithm. An uniform distribution $D_1(.)$ is assigned to the signals as an initial weight distribution (line 2). The algorithm iterates over the number of trees (line 3) and at each iteration $k$, the weak learning method $E$ constructs a decision tree $f_{DT}^k(.)$. The decision trees are constructed using first-order SVM-STL primitives $P$ and the misclassification gain impurity measure Breiman et al. (1984). Next, the misclassification error $\epsilon_k$ of the constructed tree is computed (line 5), and a weight $\alpha_k$ is computed for the tree (line 6). The weights of the trees capture contribution to computing the labels of the signals.

At the end of each iteration, the data weights are updated and normalized (denoted by $\propto$), to focus on the misclassified signals in the next trees (line 7). The final classifier $f_{BDT}^K(.)$ is constructed as the weighted sum of the decision trees (line 8). The decision tree construction method $E$ is detailed in Aasi et al. (2021).

**Algorithm 2 Boosted Decision Trees (BDT)**

1: **Input:** dataset $S = \{(s^{(i)},l^{(i)})\}_{i=1}^{N_S}$, number of decision trees $K$, weak learning method $E$
2: **Initialize:** $\forall (s^{(i)},l^{(i)}) \in S : D_1(s^{(i)}) = 1/N_S$
3: for $k = 1, \ldots, K$:
4: $E(S,D_k) \Rightarrow$ classifier $f_{DT}^k(.)$
5: $\epsilon_k \leftarrow \sum_{(s^{(i)},l^{(i)}) \in S} D_k(s^{(i)}) \cdot 1[l^{(i)} \neq f_{DT}^k(s^{(i)})]$
6: $\alpha_k = \frac{1}{2} \ln \left( \frac{1}{\epsilon_k} - 1 \right)$
7: $D_{k+1}(s^{(i)}) \propto D_k(s^{(i)}) \exp (-\alpha_k \cdot l^{(i)} \cdot f_{DT}^k(s^{(i)}))$
8: $f_{BDT}^K(.) = \text{sign} \left( \sum_{k=1}^{K} \alpha_k \cdot f_{DT}^k(.) \right)$
9: **Return:** $f_{BDT}^K(.)$ \hspace{1cm} ◦ final classifier

**Remark 1 (Applicability to other logics)** The framework presented above can be generalized to learn formulae of any ST logic that is made of STL over spatial classifiers, whenever there is a way to learn correct (a spatial classifier is correct if positive values indicate image belongs to the spatial class, and negative values indicate image does not belong to the spatial class) and interpretable spatial classifiers. Specifically, the method will work for any predicate functions $h_j : \mathbb{D} \rightarrow \mathbb{R}$, $j = 1, \ldots, n_I$ over the spatial domain $\mathbb{D}$, where $n_I$ is the number of spatial classes in the data. For example, the Tree Spatial Superposition Logic (TSSL) Gol et al. (2014) is a spatial logic equipped with quantitative semantics (robustness) that is interpretable, real-valued, and correct. TSSL learns specifications from states of networked systems with spatial domain $\mathbb{D} = \mathbb{R}^{m \times m}$. Thus, one can learn $n_I$ (one-vs-rest) spatial classifiers (TSSL formulae), and use the robustness functions as predicate functions $h_j := \rho_j : \mathbb{D} \rightarrow \mathbb{R}$. In this case, our framework can be applied with minimal modifications to learn ST properties in the form of TSSL-STL formulae.
5. Parameter Synthesis

Assume that certain desired ST properties were captured while inferring SVM-STL specifications from trajectories. One might want to find the set of system parameters (inputs) such that the executions from the system satisfy the desired specifications. In the following, we provide our approach to parameter synthesis from SVM-STL specifications Alsalehi et al. (2021); Haghighi et al. (2015).

Consider a system $S$ that exhibits time-varying spatial patterns depending on $p$ design parameters $\pi \in \Pi \subset \mathbb{R}^p$ (see Sec. 6 for an example). The ST trajectory generated by parameters $\pi$ is denoted by $S_\pi$. Consider also some ST property given as a SVM-STL formula $\varphi$. We want to find parameters $\pi^*$ such that the specification given by $\varphi$ is maximally satisfied, i.e.

$$\pi^* = \arg \max_{\pi \in \Pi} (\rho(\varphi, S_\pi, 0))$$

(9)

Note that the objective function is, in general, not differentiable. Heuristic optimization algorithms such as genetic algorithms, particle swarm optimization (PSO), or simulated annealing can be used to solve the optimization problem. Our PSO-based solution to (9) is summarized in Algorithm 3.

Algorithm 3 Parameter Synthesis using PSO

1: Input: $\varphi, \pi_0, S, Stop, (\text{hyperparameters : } W, r_p, r_g, K)$
2: initialize $[\pi_k, v_k], k = 1, \ldots, K$ \hfill \triangleright \text{initialize particle positions and velocities}$
3: while $\neg \text{Stop}$ \hfill \triangleright \text{Terminate if stopping condition is met}$
4: for $k := 1, \ldots, K$ do
5: $S_{\pi_k} \leftarrow S(\pi_k)$; \hfill \triangleright \text{Generate trajectories}$
6: $\pi_k^\text{best} \leftarrow \arg \max_{\pi = \pi_k, \pi_k^\text{best}} (\rho(\varphi, S_\pi))$
7: $[\pi_k, v_k] \leftarrow \text{update according to (3) and (4)}$
8: $\pi^\text{best} \leftarrow \arg \max_{\pi = \pi^\text{best}, \pi_k^\text{best}[k = 1, \ldots, K]} (\rho(\varphi, S_\pi))$
9: return $\pi^* \leftarrow \pi^\text{best}$

6. Experiments

In this section, we showcase our proposed framework for (1) learning ST specifications for a reaction-diffusion system and (2) synthesis of parameters for the same system to produce a desired behavior. The algorithms are implemented on a PC with a Core i7 CPU @350GHz. For clustering, we used built-in MATLAB functions ($kmeans$). For optimization, we used a custom Particle Swarm Optimization (PSO). The decision tree-based SVM-STL learning algorithm is implemented in Python 3 on an Ubuntu 18.04 system with a Core i7 @3.7GHz and 16GB RAM.

We consider a $32 \times 32$ reaction-diffusion system $S_{RD}$ Turing (1990) which describes the concentration change in space and time of two species. There are two types of changes 1) local reactions in which species are transformed into each other and 2) diffusion which causes species to spread out over a surface in space. The concentrations of the species evolve according to
Spatio-Temporal Specifications: Learning and Synthesis

\[
\frac{dx_{i,j}^1}{dt} = D_1(\mu_{i,j}^1 - x_{i,j}^1) + R_1x_{i,j}^1x_{j,i}^1 - x_{i,j}^1 + R_2 \\
\frac{dx_{i,j}^2}{dt} = D_2(\mu_{i,j}^2 - x_{i,j}^2) + R_3x_{i,j}^1 - x_{i,j}^2 + R_4
\]

where \( x_{i,j}^1 \) and \( x_{i,j}^2 \) are the concentrations of the two species at location \((i,j)\), \( \mu_{i,j}^1 \) and \( \mu_{i,j}^2 \) are the inputs to location \((i,j)\) from neighboring locations, \( D_1, D_2 \) are the diffusion coefficients, and \( R_1 = 1, R_2 = -12, R_3 = -1, R_4 = 16 \) are the parameters defining local dynamics for the species. The training set was generated using diffusion coefficients \( D_1, D_2 \) from the set \( P_1 \times P_2 \) where \( P_1, P_2 = \{0.1, 0.2, \ldots, 9.9\} \).

Learning ST properties from system executions: We consider a set \( S = \{S^{(i)}\}_{i=1}^{N_S} \), where \( S^{(i)} : T \rightarrow \mathbb{R}^{32 \times 32} \) are the ST trajectories, with \( T = [0, 60] \) and \( N_S = 25000 \).

To cluster the images in the set \( S \), we use the approach detailed in Sec. 4.1. We start by removing temporal dependency and create a new set \( I = \{I^{(i)}\}_{i=1}^{N_I} \), where \( N_I = N_S \times T = 25000 \times 60 = 1500000 \). We consider the feature extractor \( f_{cnn} \) based on the VGG16 architecture pre-trained on the ImageNet dataset and the distance measure \( d_I \). The number of classes is tuned empirically as \( n = 6 \). Sample images from the spatial classes are shown in Fig. 1a. Next, we follow the approach presented in Sec. 4.2 to learn the predicate functions \( h_1, \ldots, h_6 \) from the labeled set of images \( I \). A graphical representation is shown in Fig. 1b.

![Figure 1](image-url)

Figure 1: a) Sample images from spatial classes 1, 2, 6; b) ST trajectories from classes 1, 2, 3, color coded by green, yellow, and red, respectively.

Given the set \( S = \{S^{(i)}\}_{i=1}^{N_S} \), we use the approach detailed in Sec. 4.3 to cluster ST trajectories into \( n_S = 3 \) classes. The outcome of the clustering process is the labeled dataset \( \tilde{S} = \{(s^{(i)}, l^{(i)}_S)\}_{i=1}^{N_S} \) that is used to learn the SVM-STL specifications.

We solve 3 different two-class classification problems (see Sec. 4.4) to learn an SVM-STL formula for each class in \( S \). The performance metrics are summarized in Tab. 6. As an example formula, the learned specification for the ST class 3 in one of the folds is \( \varphi_3 = \varphi_{31} \land \varphi_{32} \land \varphi_{33} \), where \( \varphi_{31}, \varphi_{32}, \varphi_{33} \) correspond to decision trees 1, 2, 3. The weights (superscript) indicate that the first decision tree (\( \varphi_{31} \)) has the highest contribution in predicting a label for a given signal.
\( \varphi_{31} = (G_{[19,49]} h_1 \leq -4.0 \land G_{[36,47]} h_2 > 3.1) \lor (-G_{[19,49]} h_1 \leq -4.0 \land F_{[38,49]} h_2 > 4.2) \)

\( \varphi_{32} = (G_{[17,48]} h_1 \leq -4.0 \land F_{[22,44]} h_2 > 3.1) \lor (-G_{[17,48]} h_1 \leq -4.0 \land F_{[22,48]} h_2 > 4.2) \)

\( \varphi_{33} = (G_{[33,49]} h_1 \leq -4.1 \land G_{[7,44]} h_2 > 2.8) \lor (-G_{[33,49]} h_1 \leq -4.1 \land G_{[31,44]} h_2 > 4.1) \) (12)

Table 1: Performance summary. Tree depth = 2, Num. of decision trees = 3, K-fold = 2

<table>
<thead>
<tr>
<th>Specification</th>
<th>Avg. accuracy (%) training/testing</th>
<th>Std. dev. (%) training/testing</th>
<th>Run time (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi_1 )</td>
<td>99.99 / 99.68</td>
<td>0.00 / 0.00</td>
<td>2.1</td>
</tr>
<tr>
<td>( \varphi_2 )</td>
<td>93.84 / 88.11</td>
<td>6.82 / 15.8</td>
<td>1.8</td>
</tr>
<tr>
<td>( \varphi_3 )</td>
<td>99.48 / 99.24</td>
<td>0.26 / 0.28</td>
<td>2.0</td>
</tr>
</tbody>
</table>

The learned formulae are intuitive and easy to interpret. E.g. \( G_{[19,49]} h_1 \leq -4.0 \land G_{[36,47]} h_2 > 3.1 \) from \( \varphi_{31} \), is translated to as "always in time interval [19,49] the spatial class 1 is not observed AND in time interval [36,47] spatial class 2 is observed" (see Fig. 1a). The thresholds for predicates \( h_1 \leq -4.0 \) and \( h_2 > 3.1 \) show how strong the spatial classes 1, 2 are met (see red lines in Fig. 1b)

**Parameter Synthesis:** we applied the parameter synthesis approach presented in Alg. 3 to find a pair of diffusion coefficients that maximize the degree of satisfaction, with respect to the requirements given by the SVM-STL formula: \( \psi = F_{[0,30]} G_{[0,60]} h_5(S) > 0 \land G_{[0,60]} h_4(S) < 0 \)

Using Alg. 3 with with \( K = 100, W = 0.6, r_p = 1.5, r_g = 2.5 \) and a max number of iterations 20; we found parameters \( D_1 = 3.9, D_2 = 30 \) that result in satisfying the formula \( \psi \). PSO found a ST trajectory \( S \) (see Fig. 2) the satisfies \( \psi \) in ~2.5 minutes with a robustness score \( \rho(S, \psi, 0) = 0.11 \). The results illustrate the capability of SVM-STL to specify a wide range of spatial and temporal requirements for dynamical systems, and synthesize parameters to meet them.

![Figure 2](image)

**Figure 2:** Sample trajectory \( S \models \psi = F_{[0,30]} G_{[0,60]} h_5(S) > 0 \land G_{[0,60]} h_4(S) < 0 \), where \( h_5 \) corresponds to class 5 large spots and \( h_4 \) corresponds to Class 4 small spots, using parameters \( D_1 = 3.9 \) and \( D_2 = 30 \)

7. Conclusions and Future Work

This work investigated the problems of learning spatio-temporal logic formulas from system executions. We introduced SVM-STL, an extension of Signal Temporal Logic that allows for specifying spatial properties. Our framework can learn SVM-STL formulas from labeled as well as unlabeled data. We also presented a method for parameter synthesis for dynamical systems from such specifications. The learning and synthesis frameworks were showcased for a reaction-diffusion system. In future research, we will explore 1) improving expressivity by learning for a wider range of PSVM-STL primitives, 2) learning specifications using end-to-end CNN models as predicates and, 3) alternative approaches for clustering high dimensional spatio-temporal signals.
8. Acknowledgments

This work was supported by NIH under Grant R01 EB030946 and NSF under Grant OIA-2020983.

References


Spatio-Temporal Specifications: Learning and Synthesis


