Signal Clustering Using Temporal Logics

Giuseppe Bombara(✉) and Calin Belta

Boston University, Boston, USA
gbombara@bu.edu

Abstract. This paper introduces a new method for clustering signals using their temporal logic properties. Specifically, we propose a hierarchical clustering algorithm for efficiently processing a set of input signals. The input data is unlabeled, that is, no further information about properties of the signals are available to the learning algorithm other than the signals themselves. The algorithm produces a hierarchical structure where the internal nodes test some temporal properties of the data, and each terminal node contains a cluster (i.e., a group of similar signals). Each cluster can be mapped to a Signal Temporal Logic (STL) formula that describes its signals. The obtained formulae can be used directly for monitoring purposes but also, more generally, to acquire knowledge about the system under analysis. We present two case studies to illustrate the characteristics of our proposed algorithm. The first case study is related to a maritime surveillance problem, and the second is a fault classification problem in an automatic transmission system.

Keywords: Signal Temporal Logic · Specification mining · Clustering · Knowledge discovery · Formal methods · Unsupervised learning · Logic inference

1 Introduction

In recent years, there has been a great interest in applying machine learning based techniques to the formal methods field. In particular, some efforts have been made on inferring formal descriptions of the behaviors of a system from its execution traces [3,4,6,7,13,15–17]. The system behaviors can be described using an appropriate temporal logic, such as Signal Temporal Logic (STL) [19].

This approach, named *Temporal Logic Inference* (TLI) in [17], while retaining many qualities of traditional classifiers, presents several additional advantages. In particular, classical machine learning methods are often overly specific to the task. That is, they focus solely on solving the problem at hand and offer no other understanding about the system where they have been applied. On the contrary, temporal logic formulae have a precise meaning and allow for a rich specification of the behaviors of a system that is *interpretable* by human experts. In this research field, the initial work focused on finding the optimal parameters for a formula when the formula structure has been fixed [3,15,16]. Later, some attempts were made to tackle the *supervised two-class classification* *problem* [4,6,7,13,17], where the goal is to build a discriminative formula given a set of *labeled* traces (normal and anomalous).

In this paper, we turn our attention to the *unsupervised clustering problem*. In this scenario, it is assumed that only a set of *unlabeled* signals is available, that is, it is not known a priori if a signal exhibits a specific behavior or satisfies some property, and the end goal is to group *similar* signals together, in a so-called *cluster*, and to describe each cluster with a formula. We propose a hierarchical clustering approach for partitioning the input signals. Our algorithm produces a tree, where each internal node contains a formula that tests some temporal logic property of the data. The terminal nodes, or leaves, are connected with the actual clusters and each leaf can be mapped to an STL formula that *describes* the respective cluster.

The hierarchical approach provides several advantages. First, the number of signals to be processed at once decreases at each level of the hierarchy, and, more importantly, the number of clusters does *not* need to be known beforehand. We present a method for exploring the relationship between the number of clusters and the diversity in the data explained by the induced tree. We argue that this information provides useful insights on the trade-off between the complexity of the formulae inferred and the clustering effectiveness. Moreover, we will show that our unsupervised algorithm achieves classification performances on-par with the supervised algorithm in [6] and better than [17] in a case study. The formulae learned by our algorithm can be used as monitors directly during the deployment phase of a system. However, the implications of this research are even broader. The obtained formulae are in fact very useful for *acquiring knowledge* about the system under analysis. That is, they provide the designers a formal description of the possible behaviors of the system.

The paper is organized as follows. In Sect. 2, we briefly survey some previous research contributions regarding the learning of temporal logic formulae from data. In Sect. 3, we define the syntax and semantics of Signal Temporal Logic (STL), describe its parameterized variant PSTL, and review some common distance measures used to asses the similarity between signals. The unsupervised clustering problem of signals with STL formulae is discussed in Sect. 4 along with a motivating maritime surveillance case study. In Sect. 5, we describe in detail our hierarchical algorithm. We present an automotive case study in Sect. 6 and analyze the results obtained with our method in Sect. 7, along with some comparisons with related work. We conclude in Sect. 8 with a brief review of the work done and an outlook on future research directions.

2 Related Work

Attempts have been made in the recent years to learn temporal logic formulae from data. In this field, the initial work has focused on finding the optimal parameters for a formula when a formula template is given [3,15,16]. This problem is called *parameter mining*, and the parameters for the formula are selected so that the resulting formula barely satisfies the input signals [15,16], or strongly satisfies them [3] (in the sense of the robustness degree). These approaches essentially differ in the way the underlying optimization problem is formulated and solved.

Later, work was performed within the so-called *supervised two-class classification problem* [4,6,7,13,17]. In this setting, the goal is to build a temporal logic formula, *both* the formula structure and its parameters, that can distinguish signals belonging to one of two possible classes. The dataset is given as a finite set of pairs of signals and labels, where each label indicates whether the respective signal exhibits some desired system behavior or not. The approaches to solve this problem generally follow an iterative two-step procedure. In particular, [17] construct the formula structure by exploring a fragment of STL that admits a partial ordering, whereas the parameter optimization is performed using an SVM-like objective function. In [4,7], the formula structure is additively constructed by means of heuristics [7] or a genetic algorithm [4], while the parameter space is explored through a Statistical Model Checking approach on the likelihood ratio of two generative probabilistic models, which were previously fit to the available data. Finally, a decision-tree based approach to solve the two-class problem has been proposed in [6]. In this approach, a special binary tree is first constructed. Afterward, this tree is mapped to an STL formula that is used for classification.

3 Preliminaries

A temporal logic is a system of rules and symbols used for reasoning about propositions in terms of the flow of time. Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are the most commonly used temporal logics [8]. Signal Temporal Logic (STL) has emerged recently as a generalization of LTL, where time is continuous and the predicates can be defined over real values [19]. STL has found important applications in formal verification of hybrid systems where it is used to state and monitor requirements. In this section, we briefly review the syntax and the semantics of this logic.

Let $\mathbb{R}$ be the set of real numbers. For $t \in \mathbb{R}$, we denote the interval $[t, \infty)$ by $\mathbb{R}_{\geq t}$. We use $\mathcal{S} = \{s : \mathbb{R}_{\geq 0} \to \mathbb{R}^n\}$ with $n \in \mathbb{N}$ to denote the set of all continuous parameterized curves in the n-dimensional Euclidean space $\mathbb{R}^n$. In this paper, an element of $\mathcal{S}$ is called a *signal* and its parameter is interpreted as *time*. Given a signal s, the components of s are denoted by s_i, $i \in \{1, \ldots, n\}$. The set $\mathcal{F}$ contains the projection operators from a signal s to one of its components s_i, that is $\mathcal{F} = \{f_i : \mathbb{R}^n \to \mathbb{R}, f_i(s) = s_i, i = \{1, \ldots, n\}\}$.[1] The *suffix* at time $t \geq 0$ of a signal is denoted by $s[t] \in \mathcal{S}$, and it represents the signal s shifted forward in time by t time units, i.e., $s[t](\tau) = s(\tau + t)$ for all $\tau \in \mathbb{R}_{\geq 0}$.

The syntax of *Signal Temporal Logic* (STL) is defined as follows [19]:

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

where $\top$ is the Boolean *true* constant ($\bot$ for *false*); $f(x) \sim \mu$ is a predicate over $\mathbb{R}^n$ defined by a function $f \in \mathcal{F}$, a real number $\mu \in \mathbb{R}$, and an order relation

[1] A more general definition of the set $\mathcal{F}$ is used in [19].

$\sim \in \{\leq, >\}$; $\neg$ and $\wedge$ are the Boolean operators negation and conjunction; and $\mathbf{U}_{[a,b)}$ is the bounded temporal operator *until*.

The semantics of STL is defined over signals in $\mathcal{S}$ as [19]:

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

A signal $s \in \mathcal{S}$ is said to satisfy an STL formula ϕ if and only if $s[0] \models \phi$. Other Boolean operations, such as disjunction, implication, and equivalence, are defined in the usual way. The temporal operators *eventually* and *globally* are defined respectively as

$$\mathbf{F}_{[a,b)}\phi \equiv \top \mathbf{U}_{[a,b)}\phi\ , \qquad \mathbf{G}_{[a,b)}\phi \equiv \neg\mathbf{F}_{[a,b)}\neg\phi$$

In addition to the Boolean semantics defined above, some *quantitative semantics* have been proposed for STL [9,11]. These semantics are formalized through the introduction of a real valued function called *robustness*, which quantifies the *degree* of satisfaction of a signal with respect to a formula.

Parametric Signal Temporal Logic (PSTL) was introduced in [2] as an extension of STL where formulae are parameterized. A PSTL formula is similar to an STL formula, however all the time bounds in the time intervals associated with temporal operators and all the constants in the inequality predicates are replaced by free parameters. These two types of parameters are called *time* and *space* parameters, respectively. If ψ is a PSTL formula, then every parameter assignment $\theta \in \Theta$ (where Θ is the parameter space of ψ) induces a corresponding STL formula $\phi = \psi(\theta)$, where all the space and time parameters of ψ have been fixed according to θ. This assignment is also referred to as valuation θ of ψ. For example, given $\psi = \mathbf{F}_{[a,b)}(f_1(x) > \pi)$ and $\theta = [1.1, 2.3, 3.7]$, we obtain the STL formula $\psi(\theta) = \mathbf{F}_{[1.1,2.3)}(f_1(x) > 3.7)$.

Even though STL is defined using a dense-time semantics and natively supports predicates over reals, its monitoring algorithms work in practice with *sampled* signals and assume that the signals are piece-wise constant (or piece-wise linearly interpolated) [9]. The sampling rate does not have to be constant.

To measure the similarity between two signals, a *distance* function is generally used. Let $\bar{s}^1$ and $\bar{s}^2$ be two n-dimensional series of samples corresponding to the signals s^1 and s^2 in $\mathcal{S}$, respectively. The most straightforward distance function is the familiar Euclidean distance, extended to the case of multi-dimensional time-series. It is defined as:[2]

[2] If the original sampling times of s^1 and s^2 are not the same, the signals can be re-interpolated to obtain values for matching sampling times.

$$d^2(s^1, s^2) = \sum_{i=1}^{n} \sum_{t=t_0}^{t_f} \left(\bar{s}_i^1(t) - \bar{s}_i^2(t)\right)^2$$

where t_0 and t_f are the first and last sampling time, respectively.

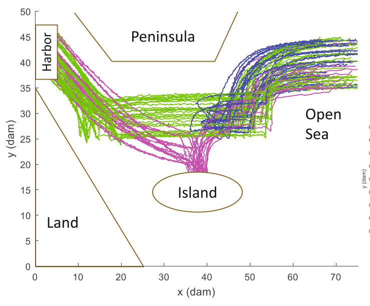
In the last decade, Dynamic Time Warping (DTW) has emerged as another popular distance measure for time series in various machine learning problems [20]. The core idea is to align two series by warping the time axis iteratively until an optimal alignment is found. There exist two possible extensions of DTW to the multi-dimensional signal case: (1) independent warping of each dimension (DTW_I), or (2) coordinated/dependent warping along all signal dimensions (DTW_D). Refer to [22] for a formal definition. While Euclidean distance is very sensitive to distortions in the time axis, DTW provides flexibility to match signals that are similar but locally out of phase [20]. Usually, a restriction on the amount of allowed warping is imposed as percentage of the signal length and is called the *warping factor*. The choice of the *right* distance function should reflect the kind of similarity measure the user is interested in and it is often application dependent.

4 Motivating Example and Problem Description

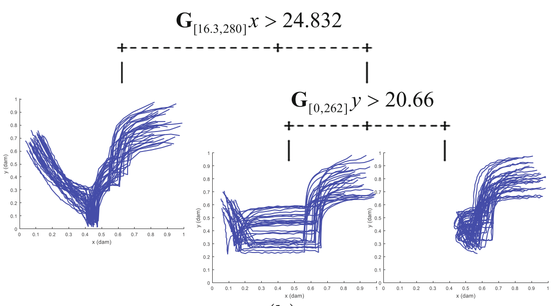
The motivating example and first case study is a maritime surveillance problem. It is assumed that the routes of the ships can be tracked in an area of interest. This surveillance problem, along with the related synthetic dataset, was initially proposed in [17], based on the scenarios described in [18]. Some example trajectories are show in Fig. 1a. In the first scenario, a vessel approaching from open sea heads directly towards the harbor. This behavior is considered normal. In the second scenario, a ship veers to the island first and then heads to the harbor. This behavior is compatible with human trafficking. In the third scenario, a boat tries to approach other vessels in the passage between the peninsula and the island and then veers back to the open sea. This scenario is compatible with terrorist activity. The signals in this dataset are represented as 2D trajectories with planar coordinates $[x(t), y(t)]$ and were generated using a Dubins' model with additive Gaussian noise [17]. The dataset is made of 2000 total trajectories, with 61 sample points per trace. There are 1000 normal trajectories, 500 human trafficking trajectories, and 500 terrorist activity trajectories.

In [6,17], a *supervised* learning problem based on this surveillance dataset was considered. Each training trajectory could be either normal, human trafficking, or terrorist activity, and its type was *known* beforehand. This information, along with the signals themselves, was exploited by the learning algorithm to construct a formula that distinguishes normal and anomalous behaviors.

The major shortcoming of supervised methods is their need of *labeled* examples during training. In the vast majority of real applications, only unlabeled data is available. For supervised algorithms to work, the raw trajectories have to be manually analyzed by a human expert (e.g. the port authority), who partitions and labels the data recorded. This generally is a time-consuming and error-prone



(a)


$\mathbf{G}_{[16.3,280]}x > 24.832$
$\mathbf{G}_{[0,262]}y > 20.66$
y (dam)
x (dam)

(b)

Fig. 1. (a) Maritime surveillance dataset. The vessels behaving normally are shown in green. The magenta and blue trajectories represent two types of anomalous routes: human trafficking and terrorist activity, respectively. (b) Tree induced for the maritime case study. The second cluster contains the normal trajectories. (Color figure online)

process. We wanted to develop an unsupervised algorithm that works directly on raw recorded trajectories to discover and represent the possible behaviors of the ships in the area. The algorithm should be able to partition the trajectories into separate groups and devise formulae to describe and discriminate them.

More formally, given a set of unlabeled signals, i.e., where the class (or type) of each signal is *not* known, the algorithm should: (1) partition the input data into separate groups, where each group contains only *similar* signals, and (2) associate each group of signals with an STL formula. In machine learning, the problem of grouping together similar objects is know as *clustering*. In our specific application, the objects are signals, and our objective is not just to partition similar signals into clusters but also to *describe* each cluster, that is, each type of behavior of the system, with an STL formula.

5 Hierarchical Clustering with STL

In literature, many algorithms have been proposed to solve clustering problems, such as K-means or DBSCAN [5]. Each approach has advantages and disadvantages, often depending on the particular application at hand. For our specific problem, we chose to design a divisive hierarchical clustering algorithm, which does not require the user to pre-assign the number of clusters beforehand (like K-means) and does not impose a probabilistic model on the data (like Gaussian Mixture Models). In general, these methods produce a hierarchy of clusters where the initial node contains the complete dataset and subgroups with more *similar* objects are present as one moves down the hierarchy. Our aim is to construct the hierarchy with a special structure so that every node can also be associated with a corresponding STL formula that represents it. To this end, we will expand the connection between STL formulae and trees as introduced in [6].

Given a set of unlabeled signals, our algorithm constructs a binary tree, where each internal node contains a simple formula that tests some temporal logic property of the signals. This formula is chosen from a set of formulae, called *primitives*, so that the signals in the resulting children nodes are more *homogeneous*, that is, they contain signals overall more *similar* to each other than their parent. The terminal nodes, or leaves, of the tree are connected with the final clusters, and each leaf is mapped to an STL formula that *describes* its respective cluster. Figure 1b shows a tree induced by our algorithm for the maritime surveillance case study. Two internal nodes partition the signals in three clusters.

This section is organized as follows. In Sect. 5.1, we first describe how binary trees can be connected with a fragment of STL by means of PSTL primitives in the nodes. In Sect. 5.2, we define some suitable *homogeneity measures* used during the optimization process to select the best primitive at each node. Finally, in Sect. 5.3 we describe our hierarchical clustering algorithm, and we conclude in Sect. 5.4 with some comments on the evaluation of its results and address the problem of choosing the appropriate number of clusters.

5.1 STL Formulae and Decision Trees

In a tree, we define: the root as the initial node; the depth of a node as the length of the path from the root to that node; the parent of a node as the neighbor whose depth is one less; the children of a node as the neighbors whose depths are one more. A node with no children is called a leaf, and all other nodes are called non-terminal nodes. We focus on *binary* trees, where every non-terminal node has exactly two children and every leaf contains a cluster.

In [6], it was proposed to split the signals reaching a node using a simple formula, chosen from a finite set of PSTL template formulae, called *primitives.* Trivially, the signals satisfying the node's primitive are routed to the left child while the signals violating the node's primitive are routed to the right child. A tree-structured sequence of questions such as this is referred to as *decision tree.* In this work, we use the following set of primitives.

Definition 1 (PSTL Primitives). *Let $\mathcal{S}$ be the set of signals with values in $\mathbb{R}^n$, we define*

$$\mathcal{P} = \left\{ \mathbf{F}_{[\tau_1,\tau_2]}(f_i(x) \sim \mu) \ \textit{or}\ \mathbf{G}_{[\tau_1,\tau_2]}(f_i(x) \sim \mu) \mid i \in \{1,\dots,n\},\ \sim \in \{\le, >\} \right\}$$

The parameters for the PSTL formulae in $\mathcal{P}$ are $(\mu, \tau_1, \tau_2]$ and the space of parameters is $\Theta = \{(a,b,c) \mid a \in \mathbb{R},\ b < c,\ b, c \in \mathbb{R}_{\ge 0}\}$.

The primitive $\mathbf{F}_{[\tau_1,\tau_2]}(f_i(x) \sim \mu)$ is used to express that the predicate $f_i(x) \sim \mu$ must be true for at least one time instance in the interval $[\tau_1, \tau_2]$, while the primitive $\mathbf{G}_{[\tau_1,\tau_2]}(f_i(x) \sim \mu)$ expresses that $f_i(x) \sim \mu$ must be true for all time in the interval.

Every leaf of a tree with this structure can be mapped to an equivalent STL formula that describes the signals falling in that leaf. Starting from a leaf of interest and backtracking to the root of the tree, the STL formula can be recursively obtained by (1) conjunction with the parent node's primitive if the current node is its left child; or (2) conjunction with the negation of the parent node's primitive if the current node is its right child. For example, in the naval surveillance case study, the formula corresponding to the cluster that contains the normal trajectories is (Fig. 1b)

$$\phi_{norm} = \mathbf{F}_{[16.3,280]}(x \leq 24.832) \wedge \mathbf{G}_{[0,262]}(y > 20.66) \tag{1}$$

Notice also the insight we can gain from the plain English translation of this formula: "The normal ship x coordinate is eventually below 24.83 in the time interval $[16.3, 280]$", i.e., it eventually approaches the port, and "the normal ship y coordinate is always above 20.66", i.e., it never approaches the island.

Remark 1. It is important to stress that the set of primitives $\mathcal{P}$ in Eq. 1 is not the only possible set. A user may define other primitives, for instance generic primitives using nested temporal operators (such as $\mathbf{G}_{[\tau_1,\tau_2]}\mathbf{F}_{[0,\tau_3]}(f_i(x) \sim \mu)$), or specific ones, guided by the particular nature of the learning problem at hand. For space restrictions, we do not investigate other primitives in this paper. However, the proposed algorithm works without modifications using other sets of primitives. The fragment of STL that is mapped with decision trees corresponds to the Boolean closure of the valuations from $\mathcal{P}$ [6].

5.2 Homogeneity Measures

The previous section describes how an STL formula can be associated to each leaf of a tree that contains primitives of $\mathcal{P}$ in its nodes. It is also necessary to define a criterion with which to select the primitive that best splits the data at each node. Our goal is to divide the signals so that the resulting two groups are more *homogeneous*. This means that each child produced contains signals more *similar* with each other and more different from the signals in the other child. To formalize this concept, we need to define (1) a measure of homogeneity for a set of signals and (2) a measure of the increase in homogeneity obtained by splitting the signals using a certain formula.

Definition 2 (Inertia-based Homogeneity Measure I). *Let S be a finite set of signals in $\mathcal{S}$, we define*

$$I(S) = \frac{1}{2}\frac{1}{|S|^2}\sum_{i=1}^{|S|}\sum_{j=1}^{|S|} d^2(s^i, s^j)$$

where $d(\cdot,\cdot)$ is a suitable distance function between two signals, such as the Euclidean Distance or Dynamic Time Warping (See Sect. 3).

$I(S)$ is the average squared distance of the signals in set S and, intuitively, a set is homogeneous when this quantity is low, i.e., the signals are close to each other.

Remark 2. As the name suggests, the Inertia-based homogeneity measure in Definition 2 recalls the *moment of inertia* concept in physics. When the distance function $d(\cdot, \cdot)$ is Euclidean, this measure is also related to the *variance* concept [1,21]. For brevity, we only present one homogeneity measure. However, others are possible, for instance by exploiting the various *linkage criteria* (i.e., single-linkage, complete-linkage, etc.) from the agglomerative clustering literature [1,5].

Definition 3 (Homogeneity Gain HG). *Let S be a finite set of signals and ϕ an STL formula, the Homogeneity Gain is defined as*

$$HG(S, \phi) = I(S) - \left[\frac{|S_\top|}{|S|} \cdot I(S_\top) + \frac{|S_\bot|}{|S|} \cdot I(S_\bot)\right] \tag{2}$$

where $S_\top = \{s^i \in S \mid s^i \models \phi\}$ and $S_\bot = \{s^i \in S \mid s^i \not\models \phi\}$ are the subsets of signals from S satisfying and not satisfying the formula ϕ, respectively.

Intuitively, a positive value of $HG(S, \phi)$ means that by splitting the set S with the formula ϕ, we obtain two sets $S_\top$ and $S_\bot$ with a *reduced overall diversity* or, equivalently, we *gained homogeneity.* The homogeneity gain in Definition 3 is connected to the so-called *Ward's criterion* in the clustering literature [1,21].

The defined homogeneity gain guides the primitive selection and parameter optimization process. In particular, given a set of primitives $\mathcal{P}$ and a set of signals S, we select the primitive $\psi^* \in \mathcal{P}$, and its optimal parameters $\theta^* \in \Theta$, so that the resulting STL formula $\psi^*(\theta^*)$ maximizes the homogeneity gain:

$$\psi^*, \theta^* = \arg\max_{\psi \in \mathcal{P}, \theta \in \Theta} HG(S, \psi(\theta)) \tag{3}$$

This problem is decomposed into $|\mathcal{P}|$ optimization problems over a small number of real-valued parameters ($|\Theta|$), which can be solved using any global non-linear optimization algorithm.

Remark 3. To solve the parameter optimization problem in Eq. (3) we used Simulated Annealing with satisfactory results. However, we feel there is room for improvement. In the future, we plan to investigate alternative optimization routines, such as Differential Evolution, and try to exploit the *monotonic* property of some PSTL formulae to speed up the optimization process [2,16].

5.3 Clustering Algorithm

In Algorithm 1 we show our parameterized clustering procedure with a high-level object oriented notation. The meta-parameters of Algorithm 1 are: (1) a set of PSTL primitives $\mathcal{P}$; (2) a set of stopping criteria *stop*; and (3) three measure functions $d()$, $I()$, $HG()$ (distance, homogeneity, and homogeneity gain, respectively) described in the previous sections. The algorithm is *iterative* and takes as input argument a set of unlabeled training signals S_{tr}. At the beginning, an empty tree is created. This tree has a single leaf, which is also the root, that

Algorithm 1. Hierarchical Clustering Algorithm - HClustSTL()

```
Parameter: $\mathcal{P}$ – set of PSTL primitives
Parameter: stop – set of stopping criteria
Parameter: $d$, $I$, $HG$ – distance, homogeneity, and homogeneity gain measures
Input: $S_{tr}$ – set of training signals
Output: $T$ – a Tree
1 $T \leftarrow$ createEmptyTree($S_{tr}$)
2 while stop()==false do
3     $L \leftarrow T$.selectDivLeaf()
4     $\psi^*, \theta^* \leftarrow \arg\max_{\psi \in \mathcal{P}, \theta \in \Theta} HG(L.S, \psi(\theta))$
5     $T$.nonTerminal($L$, $\psi^*(\theta^*)$)
6     $S^*_\top, S^*_\bot \leftarrow$ partition($L.S, \psi^*(\theta^*)$)
7     $T$.createLeftLeaf($L$, $S^*_\top$)
8     $T$.createRightLeaf($L$, $S^*_\bot$)
9 return $T$
```

contains all the input signals (line 1). At each iteration, the least homogeneous leaf is selected for further processing (line 3). This is the leaf L that contains the most diverse signals $L.S$ according to the homogeneity measure $I()$. The algorithm proceeds to find the optimal STL formula, among all the valuations of PSTL formulae from the set of primitives $\mathcal{P}$, to split the signals in L (line 4). The goal here is to get more homogeneous children, and the optimality is assessed using the homogeneity gain measure $HG()$ according to Eqs. (2)–(3). Next, the leaf L is converted to a non-terminal node, and it is associated with the formula $\psi^*(\theta^*)$ (line 5). The induced partition of the signals $S^*_\top, S^*_\bot$ is computed (line 6), and for each outcome of the split, a corresponding child leaf is created (lines 7–8). The stopping conditions are checked at every iteration (line 2), and the constructed tree T is returned when they are met. Several stopping criteria can be set for Algorithm 1. The most common strategy is to stop if the tree has reached a certain prespecified depth. Another strategy is to stop when the leaves' diversity is below a certain threshold. As we will discuss in the next section, it is generally good practice to use permissive stopping conditions and then assess the quality of the induced hierarchy using other tools.

Remark 4. We implemented and tested Algorithm 1 using MATLAB. The only computationally expensive step of the algorithm is the optimization in line 4. This problem becomes easier as the depth of the tree increases because fewer signals need to be processed. Aside from the optimization routine itself (see Remark 3), the computation of the objective function $HG(S, \psi(\theta))$ implicitly requires the construction of the partition $S_\top, S_\bot$ induced by $\psi(\theta)$ and the computation of the distances among the signals in S. To speed up the execution, we precompute and store a *distance matrix* that contains the distances, taken pairwise, of the signals in the dataset S_{tr}. Moreover, since the distance computations involve multi-dimensional signals, normalizing the dataset before the execution of the algorithm has proven to be useful.

5.4 Clustering Evaluation

Evaluating the quality of clustering results is a difficult task [1]. If independent labeled data is available, it is possible to assess how similar the returned clusters are to the predetermined classes, treating the latter as a *gold standard* during the evaluation. Several performance metrics used for classification tasks can be adapted for this purpose, such as the misclassification rate (MCR). However, there is no unanimous agreement that this is an adequate way of assessing the performance of a clustering algorithm [12]. Specifically, manually assigned labels only represent *one* possible partitioning of the dataset, which does not imply that there are no other, equal or more meaningful, partitions. Moreover, in general, labeled data is not available at all, and clustering algorithms should be interpreted more as *exploratory* and *knowledge discovery* tools [1,21] that assist the human user who is the ultimate (and subjective) judge. In this context, clustering is a trial and error process and may involve several attempts. Our algorithm, with explicit support for meta-parameters, e.g., different distance measures, is well suited for this task.

An important related issue is deciding the number of clusters. One of the major advantages of hierarchical clustering algorithms is that the number of clusters does not have to be *prefixed* before the execution of the algorithm. Given a constructed hierarchy of clusters, some methods have been proposed in literature to help the user decide the appropriate number of clusters without requiring explicit data labeling [1]. We used the so-called *elbow* method. The core idea is to explore the relationship between the number of clusters and the remaining diversity in the clusters. The latter is quantified with the *percentage of unexplained diversity* defined as the ratio between the weighted sum of inhomogeneity in the clusters and the initial inhomogeneity of the whole dataset. The percentage of unexplained diversity is plotted against the number of clusters. Generally, the first splits of the tree explain a lot of diversity in the data, but at some point the marginal decrease will drop, giving an angle in the graph (hence the elbow name). This information can be exploited to stop the algorithm, if a satisfactory solution has been found, or to prune the tree in order to remove unnecessary clusters. Moreover, since there is a direct connection in our method between depth of induced tree, number of clusters, and length of the formulae representing the clusters, this analysis provides insights on the trade-off between formulae complexity and clustering effectiveness.

For the maritime surveillance case study, we obtained excellent results with the classical Euclidean norm. Algorithm 1 was set to run until a tree of depth 4 was constructed. The elbow analysis in Fig. 2 shows that the last major drop in unexplained diversity happens when there are three leaves, and this corresponds to the tree shown in Fig. 1b. In this case, each cluster can be mapped with one of the original scenarios of the dataset.

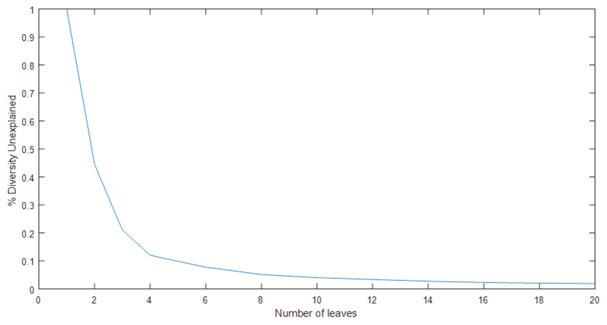


Fig. 2. Maritime surveillance case study - elbow analysis

6 Automotive Case Study

The second case study is a fault classification problem in an automatic transmission system. We constructed this dataset for the purposes of this investigation by modifying a built-in Simulink model [23]. We selected this model because it includes all the complexities of real world industrial models, such as continuous and discrete blocks, look-up tables, and Stateflow charts. Moreover, this model was proposed as a benchmark for the Hybrid Systems community in [14] and variations of the base scheme have been used as case studies in some recent formal methods papers [10,16,24].

In this section, we briefly describe the base model and our modifications. Consider the closed-loop vehicle model with four speeds and an automatic transmission controller of [23]. The transmission subsystem allows the internal combustion engine to run at appropriate rotational speeds in order to provide a range of speeds and torques necessary for the vehicle to move and maneuver effectively. The goal of the controller is to automatically change gear, freeing the driver from having to shift gears manually. The system has two inputs: the throttle opening and the brake. The throttle can take any value between 0 (fully closed) and 100 (fully open). The brake can also take values between 0 and 100, and it is used to model the variable load on the engine, such as the vehicle going uphill or downhill. The system has two continuous and one discrete state variables. The continuous state variables are the engine speed (in rpm) and the vehicle speed (in mph). The discrete state variable is the gear, which takes an integer value between 1 and 4. The engine speed and the vehicle speed are also used as outputs of the system.

For the simulation, we consider the vehicle starting from rest and subsequently performing a surpass maneuver. Specifically, the vehicle starts with zero speed, engine at 1000 RPM, and with throttle at 50%. The surpass maneuver begins at a random time, between 15 and 25 s, when the driver steps the throttle to 90%. The overall simulation time is 50 s. Noise has also been injected in the system. In particular, independent Gaussian random noise was added to the engine speed, the transmission speed, and the vehicle's speed sensor.

In this case study, three types of faults have been simulated. The first type of fault models a malfunction in the vehicle's speed sensor. The fault can manifest at anytime during the execution and the readings of the sensor are substituted with a random (erratic) value between 0 and 100 mph. The second and third types of faults are obtained by tampering with the gear shifting logic, which is implemented with a Stateflow chart in the Simulink model. In the second type of fault, the Stateflow chart is modified so that the vehicle is unable to engage the fourth gear. For the third type of fault, the Stateflow chart is modified so that the automatic transmission switches directly from second gear to fourth gear, and vice versa, by skipping the third gear altogether.

We performed 1500 total simulations with different settings. In detail, we obtained: 750 traces where the system was working normally; 250 traces with a fault in the vehicle's speed sensor (Type 1 fault); 250 traces with an *unable to engage the fourth gear* fault (Type 2 fault); 250 traces with a *skip the third gear* fault (Type 3 fault). For every trace, we collected the system's output values, that is, the speed of the engine (variable x_1) and the speed of the vehicle (variable x_2). Some sample traces are shown in Fig. 3a.

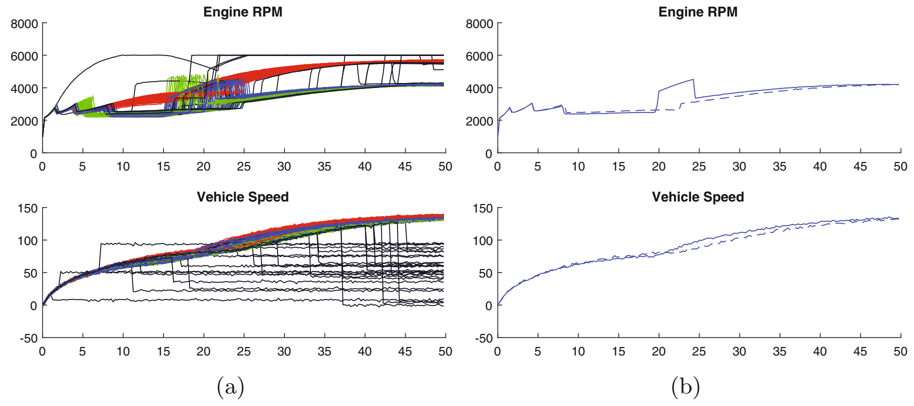


Fig. 3. Automatic transmission - (a) Example trajectories: Normal in blue, anomalous in black (Type 1-`sensor fault`), red (Type 2-`no 4th`), and green (Type 3-`skip 3rd`). (b) Two qualitatively different normal scenarios: (1) shift sequence 4th-3rd-4th with a solid line, and (2) stay in 4th gear with a dashed line. (Color figure online)

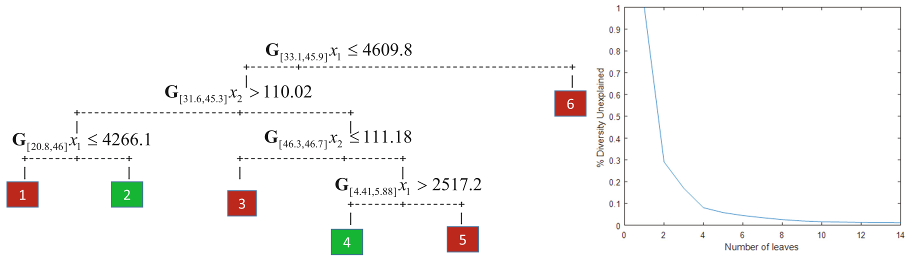
7 Results

For the automotive case study, we executed Algorithm 1 with the aim of reconstructing the different types of faults and the normal conditions present in the dataset. We obtained the best results using the *dependent* multi-dimensional dynamic time-warping distance (DTW$_D$) with 0.2 warping factor. This is due to the fact that the components of each signal come from the same system and share the same clock. Moreover, the signals in the dataset can be shifted and distorted in time due to the varying input throttle and the noise injected. Algorithm 1 was set to run until a tree of depth 4 was constructed. The elbow analysis in this case shows that there is not much gain after six clusters have been created (Fig. 4b). This corresponds to the tree shown in Fig. 4a.

Type 1 and Type 2 faults are mapped with a cluster each. Type 3 faults are mapped with two clusters, and there are also two separate clusters containing normal signals. This happens because, when the surpassing maneuver starts, two scenarios are possible. In the first scenario, the transmission downshifts from the fourth gear to the third gear and the engine jumps from about 2500 RPM to about 4000 RPM. The engine torque is thus increased and so is the mechanical advantage of the transmission. With continued heavy throttle, the vehicle accelerates and eventually shifts back to the forth gear (Fig. 3b - solid line). In the second scenario, the automatic transmission deems the gear change not necessary, according to the shift schedule, and the vehicle stays in the forth gear for the whole time (Fig. 3b - dashed line). Using a simple disjunction, we can obtain a single formula that overall describes the normal conditions (it encompasses the signals contained in both clusters):

$$\phi_{norm} = \begin{array}{l} \mathbf{G}_{[33.1,45.9]} x_1 \leq 4609.8 \wedge ((\mathbf{G}_{[31.6,45.3]} x_2 > 110.02 \wedge \mathbf{F}_{[20.8,46]} x_1 > 4266.1) \\ \vee (\mathbf{F}_{[31.6,45.3]} x_2 \leq 110.02 \wedge (\mathbf{F}_{[46.3,46.7]} x_2 > 111.18 \wedge \mathbf{G}_{[4.41,5.88]} x_1 > 2517.2))) \end{array}$$

Since labeled data is available for this case study, we tested the classification performance of this formula using an independent test set and achieved a misclassification rate of 0.031.


$\mathbf{G}_{[33.1,45.9]} x_1 \leq 4609.8$
$\mathbf{G}_{[31.6,45.3]} x_2 > 110.02$
$\mathbf{G}_{[20.8,46]} x_1 \leq 4266.1$
$\mathbf{G}_{[46.3,46.7]} x_2 \leq 111.18$
$\mathbf{G}_{[4.41,5.88]} x_1 > 2517.2$
1
2
3
4
5
6
% Diversity Unexplained
Number of leaves

(a) Cluster Tree - Leaves 2 and 4 contain normal signals (b) Elbow Analysis

Fig. 4. Automatic transmission results

We return to the maritime surveillance case study for some final remarks. This case study was also investigated in [17], with their SVM-based approach, and in [6], using a decision-tree based algorithm. Both these works tackle the *supervised two-class classification* problem. Even though the problem settings are quite different, we can still try to make some comparisons since our algorithm was able to recover the same class structure of the original dataset (as shown in

Fig. 1). In terms of classification performance, the formula in Eq. (1) achieves a misclassification rate of 0.0079 on an independent test set. This outperforms the results obtained in [17], improving the misclassification rate by a factor of 20, and is on par with the results of the supervised algorithm in [6]. This result is quite impressive considering that our algorithm does *not* use labeled examples and yet it is able to achieve similar accuracy. For the naval case study, the execution time of Algorithm 1 was 283 s, which is still far better than [17] but roughly three times slower with respect to [6].[3] The difference with [6] is due to two main factors: (1) the computation of distances, which are not needed in supervised algorithms, and (2) the greater depth at which the tree is initially induced, since labels are not available to guide the algorithm termination. Moreover, it must be noted that the exploratory nature of clustering generally requires more than one execution.

8 Conclusion

We introduced a novel method for learning signal classifiers in the form of STL formulae from raw data. Specifically, we tackled the clustering problem. In this challenging scenario, no labels are available for the signals, and the end goal is to cluster similar signals together and represent each cluster with an STL formula.

We exploited the connection between binary trees and STL formulae introduced in [6] and constructed an hierarchical clustering algorithm that partitions signals using PSTL primitives to test simple temporal logic properties of the data. The best primitives, along with their parameters, are chosen by optimizing some appropriately defined *homogeneity* measures, which capture how well a formula splits the signals according to their similarity. Finally, each leaf of the tree is associated with a cluster and can be mapped to an STL formula that describes it. *Both* the formula structure and its parameters are derived from the input data.

This work is in line with the recent interest in learning temporal specification from data and is motivated by the need to construct formulae that provide good clustering performance while being interpretable. This allows the designers to acquire some knowledge over the derived clusters and the specific application domain. Moreover, the inferred formulae can be used in other contexts, such as system monitoring. The proposed algorithm has been tested on two case studies in the maritime surveillance and automotive fields. We showed that the algorithm is able to capture relevant characteristics of the signals in both cases and achieves solid classification accuracy.

In the future, we plan to investigate other homogeneity measures, especially some measures derived from information theory, and then perform a comparative study of the behavior of the proposed algorithm for each measure. Future work also includes improving the local optimization procedure, which will speed up the overall performance of the algorithm.

[3] We ran our experiments on a Windows PC with an Intel 5920K CPU.

Acknowledgments. This work was partially supported by DENSO CORPORATION and by the Office of Naval Research under grant N00014-14-1-0554. The authors would like to acknowledge Hirotoshi Yasuoka (DENSO CORPORATION) and Rachael Ivison (Boston University) for providing valuable feedback during this research. We also thank the anonymous reviewers for their comments.

References

1. Aggarwal, C.C., Reddy, C.K.: Data Clustering: Algorithms and Applications Chapman & Hall/CRC, 1st edn. CRC Press, Boca Raton (2013)
2. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric Identification of Temporal Properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29860-8_12
3. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: System design of stochastic models using robustness of temporal properties. Theoret. Comput. Sci. **587**, 3–25 (2015)
4. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Cham (2014). doi:10.1007/978-3-319-10512-3_3
5. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics. Springer, New York (2006)
6. Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., Belta, C.: A decision tree approach to data classification using signal temporal logic. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, pp. 1–10. ACM, New York (2016)
7. Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., Bortolussi, L.: Temporal logic based monitoring of assisted ventilation in intensive care patients. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8803, pp. 391–403. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45231-8_30
8. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
9. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15297-9_9
10. Fainekos, G., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of automotive control applications using S-TaLiRo. In: American Control Conference (ACC), vol. 2012, pp. 3567–3572, June 2012
11. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoret. Comput. Sci. **410**(42), 4262–4291 (2009)
12. Färber, I., Günnemann, S., Kriegel, H.P., Kröger, P., Müller, E., Schubert, E., Seidl, T., Zimek, A.: On using class-labels in evaluation of clusterings. In: MultiClust: 1st International Workshop on Discovering, Summarizing and Using Multiple Clusterings Held in Conjunction with KDD 2010 (2010)
13. Grosu, R., Smolka, S.A., Corradini, F., Wasilewska, A., Entcheva, E., Bartocci, E.: Learning and detecting emergent behavior in networks of cardiac myocytes. Commun. ACM **52**(3), 97–105 (2009)
14. Hoxha, B., Abbas, H., Fainekos, G.: Benchmarks for temporal logic requirements for automotive systems. In: Proceedings of Applied Verification for Continuous and Hybrid Systems (2014)

15. Hoxha, B., Dokhanchi, A., Fainekos, G.: Mining parametric temporal logic properties in model-based design for cyber-physical systems. Int. J. Softw. Tools Technol. Transfer, 1–15 (2017)
16. Jin, X., Donzé, A., Deshmukh, J., Seshia, S.A.: Mining requirements from closed-loop control models. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **PP**(99), 1 (2015)
17. Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., Belta, C.: Temporal logic inference for classification and prediction from data. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, HSCC 2014, pp. 273–282. ACM, New York (2014)
18. Kowalska, K., Peel, L.: Maritime anomaly detection using Gaussian Process active learning. In: 2012 15th International Conference on Information Fusion (FUSION), pp. 1164–1171, July 2012
19. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30206-3_12
20. Ratanamahatana, C.A., Keogh, E.: Everything you know about dynamic time warping is wrong. In: Third Workshop on Mining Temporal and Sequential Data. Citeseer (2004)
21. Ripley, B.D.: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge (1996)
22. Shokoohi-Yekta, M., Wang, J., Keogh, E.: On the non-trivial generalization of dynamic time warping to the multi-dimensional case. In: Proceedings of the 2015 SIAM International Conference on Data Mining, Proceedings, Society for Industrial and Applied Mathematics, pp. 289–297, June 2015
23. The MathWorks Inc: MATLAB and Simulink R2017a. Natick, Massachusetts (2017)
24. Zhao, Q., Krogh, B.H., Hubbard, P.: Generating test inputs for embedded control systems. IEEE Control Syst. **23**(4), 49–57 (2003)