

**Boston University**

**OpenBU**

**<http://open.bu.edu>**

---

Boston University Theses & Dissertations

Boston University Theses & Dissertations

---

2020

# Learning temporal logic formulae from data

---

<https://hdl.handle.net/2144/39359>

*"Downloaded from OpenBU. Boston University's institutional repository."*

BOSTON UNIVERSITY  
COLLEGE OF ENGINEERING

Dissertation

**LEARNING TEMPORAL LOGIC FORMULAE FROM  
DATA**

by

**GIUSEPPE BOMBARA**

M.S., Università di Pisa, 2013

Submitted in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

2020

© 2020 by  
GIUSEPPE BOMBARA  
All rights reserved

## Approved by

First Reader

---

Calin A. Belta, Ph.D.  
Professor of Mechanical Engineering  
Professor of Systems Engineering  
Professor of Electrical and Computer Engineering

Second Reader

---

Douglas Densmore, Ph.D.  
Associate Professor of Electrical and Computer Engineering  
Associate Professor of Biomedical Engineering

Third Reader

---

Roberto Tron, Ph.D.  
Assistant Professor of Mechanical Engineering  
Assistant Professor of Systems Engineering

Fourth Reader

---

Wenchao Li, Ph.D.  
Assistant Professor of Electrical and Computer Engineering  
Assistant Professor of Systems Engineering

*In your actions, don't procrastinate.  
In your thoughts, don't wander.  
In your conversations, don't confuse.  
In your soul, don't be passive or aggressive.  
In your life, don't be all about business.*

Marcus Aurelius

## Acknowledgments

I would like to thank my family, my dad, Orazio, my mom Caterina, and my brother, Marco, for supporting me and loving me all these years. Most of which have been away from home (too many!).

A special acknowledgment goes to my advisor, Prof. Calin Belta, for the continuous support of my Ph.D. research, for his patience and motivation. His guidance helped me grow professionally and personally.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Tron, Prof. Li, and Prof. Densmore, for their insightful comments. I also would like to thank the many faculty members of the CISE area for the many fruitful discussions in these years and for all the amazing seminars that they organized.

Finally, I would like to express my sympathy and appreciation to the fellow members of the BU robotics lab. It has been an amazing journey with you guys!

This work would not have been possible without the aid of DENSO CORPORATION, the Office of Naval Research, and the National Science Foundation. I am grateful for their support.

Giuseppe Bombara

# LEARNING TEMPORAL LOGIC FORMULAE FROM DATA

GIUSEPPE BOMBARA

Boston University, College of Engineering, 2020

Major Professor: Calin A. Belta, Ph.D.

Professor of Mechanical Engineering

Professor of Systems Engineering

Professor of Electrical and Computer Engineering

## ABSTRACT

In recent years, there has been a great interest in applying machine learning-based techniques to the formal methods field. In this thesis, we focus on inferring high-level descriptions of a system from its execution traces. The system behaviors are formalized using an appropriate temporal logic language called *Signal Temporal Logic* (STL).

We approach the two-class *classification* problem first. Given a finite set of pairs of system traces and labels, where each label indicates whether the respective trace exhibits a system property, we devised a decision-tree based framework that outputs an STL formula that can distinguish the traces. Afterward, we tackle the *online learning* scenario. In this setting, it is assumed that new signals may arrive over time, and the previously inferred formula should be updated to accommodate the new data. Later, we propose a hierarchical clustering algorithm for producing formulae even when the input labels are not available (*unsupervised learning*). Finally, we extend this work to the multi-class case and drastically improve the execution time by introducing smooth cost functions.

The proposed framework, while retaining many qualities of traditional classifiers, presents some additional advantages. In particular, the produced formulae have a

precise meaning that is *interpretable* by users. They can be used to *acquire knowledge* about the system under analysis. The formulae usage is not restricted to signal classification or knowledge acquisition but they can be used in other phases of the system's operation, such as monitoring and control.

We present three case studies to illustrate the characteristics and effectiveness of the proposed algorithms: 1) a maritime surveillance problem; 2) an anomaly detection problem in an automotive powertrain subsystem; and 3) a fault classification problem in an automatic transmission.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related Work</b>	<b>4</b>
2.1	Parameter Mining . . . . .	4
2.2	Two-Class Supervised Learning . . . . .	5
2.3	Other Related Work . . . . .	6
<b>3</b>	<b>Preliminaries</b>	<b>8</b>
3.1	Signal Temporal Logic . . . . .	8
3.2	Quantitative Semantics for STL . . . . .	10
3.2.1	Smooth Robustness . . . . .	11
3.3	Parametric Signal Temporal Logic . . . . .	11
3.4	Signal Distances . . . . .	12
<b>4</b>	<b>Two-Class Classification</b>	<b>13</b>
4.1	Introduction and Problem Formulation . . . . .	13
4.2	STL formulae and Decision Trees . . . . .	13
4.3	PSTL primitives . . . . .	15
4.4	Impurity measures . . . . .	17
4.5	Parameterized learning algorithm . . . . .	19
4.6	Tree to STL formula . . . . .	20
4.7	Local node optimization . . . . .	21
4.8	Stop conditions, Pruning, and Formula Complexity . . . . .	22
4.9	Computational Complexity . . . . .	23
4.10	Case Study: Maritime Surveillance . . . . .	23

4.11	Maritime Surveillance: Two-Class Classification Results . . . . .	24
<b>5</b>	<b>Online Learning</b>	<b>29</b>
5.1	Introduction . . . . .	29
5.2	Online learning algorithm . . . . .	30
5.3	Primitive evaluation and node creation . . . . .	31
5.4	Maritime Surveillance: Online Learning Results . . . . .	34
5.5	Error evolution analysis . . . . .	35
5.6	Comparisons with the offline algorithm . . . . .	37
<b>6</b>	<b>Signal Clustering</b>	<b>38</b>
6.1	Introduction and Problem Formulation . . . . .	38
6.2	Hierarchical Clustering with STL . . . . .	39
6.3	Homogeneity measures . . . . .	40
6.4	Clustering Algorithm . . . . .	41
6.5	Clustering Evaluation . . . . .	43
6.6	Maritime Surveillance: Clustering Results . . . . .	44
6.7	Comparison with the supervised algorithm . . . . .	45
<b>7</b>	<b>Multi-Class Classification and Smooth Optimization</b>	<b>47</b>
7.1	Multi-Class Classification . . . . .	47
7.2	Primitive optimization procedure . . . . .	49
7.3	Smooth objective functions . . . . .	50
7.3.1	Smooth Impurity Measures for Multi-Class Classification . . .	50
7.3.2	Smooth Homogeneity Gain Measure for Clustering . . . . .	52
7.3.3	Smooth primitive optimization procedure . . . . .	53
7.4	Maritime Surveillance: Multi-Class and Smooth Optimization Results	53
<b>8</b>	<b>Case Studies</b>	<b>56</b>
8.1	Implementation and Validation . . . . .	57
8.2	Fuel Control System . . . . .	57

8.2.1	Model Description . . . . .	57
8.2.2	Modifications and fault injection . . . . .	58
8.2.3	Dataset generation . . . . .	58
8.2.4	Results . . . . .	60
8.3	Automatic Transmission . . . . .	65
8.3.1	Model Description . . . . .	65
8.3.2	Modifications and fault injection . . . . .	67
8.3.3	Dataset generation . . . . .	67
8.3.4	Results . . . . .	68
<b>9</b>	<b>Conclusion</b>	<b>72</b>
9.1	Comparisons and Discussion . . . . .	72
9.2	Summary of the thesis . . . . .	73
	<b>Bibliography</b>	<b>75</b>
	<b>Curriculum Vitae</b>	<b>80</b>

# List of Tables

4.1	Maritime Surveillance - Analysis of classification accuracy as a function of Impurity Measure ( $J$ ), Primitive Set ( $\mathcal{P}$ ) and Maximum Tree Depth ( $D$ ). . . . .	24
4.2	Maritime Surveillance - Instance selection for testing the extended impurity measures. . . . .	26
8.1	Fuel Control System - Analysis of classification accuracy as a function of Impurity Measure ( $J$ ), Primitive Set ( $\mathcal{P}$ ) and Maximum Tree Depth ( $D$ ). . . . .	61

# List of Figures

4.1	Binary tree with PSTL primitives at the internal nodes . . . . .	15
4.2	Maritime Surveillance - Example Trajectories . . . . .	25
4.3	Maritime Surveillance - Boundary of two PSTL primitives superimposed on a sample of the dataset . . . . .	28
5.1	Maritime Surveillance - MCR as function of the number of signal processed by the online algorithm . . . . .	35
6.1	Maritime Surveillance - Elbow Analysis for Clustering . . . . .	45
6.2	Maritime Surveillance - Clustering Results . . . . .	46
7.1	Maritime Surveillance - Multi-Class Classification Results . . . . .	54
8.1	Fuel Control - Overview of the model . . . . .	59
8.2	Fuel Control - Example Trajectories . . . . .	60
8.3	Fuel Control - Pruning Analysis . . . . .	62
8.4	Fuel Control - MCR as function of the number of signal processed by the online algorithm . . . . .	64
8.5	Automatic Transmission - Overview of the model . . . . .	66
8.6	Automatic Transmission - Example Trajectories . . . . .	68
8.7	Automatic Transmission - Clustering Results . . . . .	70
8.8	Automatic Transmission - Two qualitatively different normal scenarios	71

## List of Abbreviations

CTL	.....	Computation Tree Logic
DAG	.....	Directed Acyclic Graph
DTW	.....	Dynamic Time Warping
EGO	.....	Exhaust Gas Oxygen
iPSTL	.....	inference Parametric Signal Temporal Logic
LTL	.....	Linear Temporal Logic
MAP	.....	Manifold Absolute Pressure
MCR	.....	Misclassification Rate
PSTL	.....	Parametric Signal Temporal Logic
STD	.....	Standard Deviation
STL	.....	Signal Temporal Logic
TLI	.....	Temporal Logic Inference

# Notation

$s$	.....	signal
$T$	.....	tree
$L$	.....	leaf
$S$	.....	set of signals
$\theta$	.....	parameters for a formula
$\Theta$	.....	parameter space
$\psi$	.....	PSTL primitive
$\psi_\theta$	.....	PSTL primitive with fixed parameters
$\phi$	.....	STL formula
$r$	.....	robustness degree
$\rho$	.....	smooth robustness degree
$\mathcal{P}$	.....	set of PSTL primitives
$J$	.....	Impurity Measure
$IG$	.....	Information Gain
$MG$	.....	Misclassification Gain
$d$	.....	distance measure
$HG$	.....	Homogeneity Gain

## Chapter 1

# Introduction

In recent years, there has been a cross-fertilization between the fields of machine learning and formal methods. For example, formal verification techniques have been applied to provide guarantees on the behavior of machine learning components, such as neural networks (Liu et al., 2019).

In this thesis, we focus on *learning* high-level descriptions of a system from its execution traces. The system operation is described using Signal Temporal Logic (STL), a specification language used in the field of formal methods to define the behaviors of dynamical systems (Donzé and Maler, 2010). The inferred formulae can be employed directly for classification or, more generally, for monitoring and controlling the system.

This approach, while retaining many qualities of traditional classifiers, addresses some of the limitations of current formalisms. First, as opposed to most classifiers, STL formulae have precise meaning and allow for a rich specification of the behaviors that is *interpretable* by humans experts. Second, machine learning methods commonly applied to time series data are often model-based, i.e., they require a *good* model of the system under analysis (Isermann, 2006). Third, classical machine learning methods are often overly specific to the task. That is, they focus exclusively on solving the problem at hand but offer no other insight on the system where they have been applied (*knowledge discovery*).

We begin this thesis in Chapter 2, with a survey of the related research efforts, and in Chapter 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 assess the similarity between signals.

We focus on the so-called *two-class classification problem* first in Chapter 4. In this setting, our goal is to build a temporal logic formula that can distinguish traces belonging to one of two possible classes. The dataset is given as a finite set of pairs of system traces, also called *signals*, and *labels*. Each label indicates whether the respective trace exhibits some desired system behavior, e.g., an engine is working correctly (*supervised learning*). To construct a discriminating formula, we propose a novel, decision-tree based framework. We refer to it as *framework* because we are not just proposing a single algorithm but a *class* of algorithms. Every algorithm produces a binary decision tree which can be translated to an STL formula and used for classification purposes. In this approach, each node of the tree contains a test associated with the satisfaction of a simple formula, optimally tuned from a predefined set of primitive formulae. The optimality at each step is assessed using *impurity* measures, which capture how well a primitive splits the signals in the training data. The impurity measures described in this thesis are modified versions of the usual impurity measures to handle signals, and were obtained by exploiting the *robustness degree* concept (Donzé and Maler, 2010).

In Chapter 5, we turn our attention to the *online learning problem*. In this scenario, it is assumed that new data arrives over time and the inference system should be updated to accommodate it. This is in contrast with the classical (or offline) scenario, where only a single batch of data is available at the beginning and no further data can be considered. The online learning approach presents some major advantages. First, it provides a formula early during the signal collection process and then can refine it progressively when more signals become available. Second, it removes the usual separation between building phase and deployment phase of classifiers. The key insight we use to solve this problem *efficiently* is to create a new node in the decision tree only when we can be *reasonably* sure that the decision made holds for

future data (Domingos and Hulten, 2000). This is achieved through a probabilistic assessment among the possible options available for a node.

In Chapter 6, we deal with the *unsupervised clustering problem*. In this case, we assume 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 leverage the link between decision trees and STL formulae introduced in Chapter 4 and propose a hierarchical clustering algorithm for partitioning the input signals. Here, the optimality of the candidate primitives at each node is assessed using appropriately defined *homogeneity* measures, which capture how well a formula splits the signals with respect to the signals' *similarity*. The leaves are connected with the actual clusters and each leaf can be mapped to an STL formula that *describes* the respective cluster.

In Chapter 7, we first extend the supervised classification algorithm to deal with multi-class datasets. Later, we revisit the local node optimization problem underlying all the algorithms proposed in this thesis. We introduce new sets of impurity and homogeneity measures that are differentiable, allowing us to deploy a smooth optimization solver. This drastically improved the execution time of the whole framework.

In this thesis, we examine three case studies: 1) a maritime surveillance problem; 2) an anomaly detection problem in an automotive powertrain subsystem; and 3) a fault classification problem in an automatic transmission. The maritime surveillance case study is used as a running example throughout the thesis to explain the problems addressed and to test our proposed algorithms. The automotive case studies are presented and analyzed in Chapter 8.

We conclude in Chapter 9 with a critical review of the solutions proposed in this thesis and in other related works, along with a summary of the major contributions.

## Chapter 2

# Related Work

In this chapter, we focus mostly on research directly related to learning Signal Temporal Logic (STL) formulae from data, where two major areas are identified. Sec. 2.3 contains some references to related work in other logics.

### 2.1 Parameter Mining

The first area is concerned with finding the optimal parameters for a formula when a formula structure is given (Bartocci et al., 2015; Chen et al., 2016; Hoxha et al., 2017; Jha et al., 2017; Jin et al., 2015). That is, a designer provides a formula template such as “The engine speed settles below  $v$  m/s within  $\tau$  second” and an optimization procedure finds values for  $v$  and  $\tau$ . The given structure reflects the domain knowledge of the designer on the system and its properties of interest. This problem is called *parameter mining*, and the parameters for the formula are selected so that the resulting formula barely satisfies the input signals (Hoxha et al., 2017; Jha et al., 2017; Jin et al., 2015), or strongly satisfies them (Bartocci et al., 2015) (in the sense of the robustness degree). These approaches essentially differ in the way the underlying objective function is formulated and the optimization strategy employed. It is worth mentioning that in (Chen et al., 2016; Hoxha et al., 2017; Jin et al., 2015), the parameter optimization problem is cast within a more general active learning framework, where the original system is queried for new signals if deemed necessary.

## 2.2 Two-Class Supervised Learning

The second area tackles the *supervised two-class classification problem* and the goal is to construct a formula, *both* structure and parameters, that can *distinguish* between two sets of signals (Bartocci et al., 2014; Kong et al., 2017). In this setting, given a set of labeled traces, that is, whose class is known beforehand (e.g., either *normal* or *anomalous*), the goal is to build a formula that can *distinguish* between them. The approaches to solve this problem generally follow an iterative two-step procedure. In (Kong et al., 2017), the authors first defined a fragment of STL, called inference parametric signal temporal logic (iPSTL), and showed this fragment admits a partial order among formulae in the sense of language inclusion, and with respect to the robustness degree. This implies that iPSTL formulae can be organized in an infinite directed acyclic graph (DAG) capturing their ordering. This result is used to formulate the classification problem as an optimization problem, whose objective function involves the robustness degree, and solve it in two cyclic steps: 1) optimize the formula structure by exploring the DAG, pruning and growing it, and 2) optimize the formula parameters, for a fixed structure, using a nonlinear optimization algorithm. This approach presents two major limitations. First, the parameter optimization routine has a high computational cost. This is due to its nonlinear nature. Finding the optimal valuation becomes more and more challenging as the algorithm proceeds, because the dimension of the parameter space grows at each iteration. Second, the DAG is built using an ordering on the language accepted by iPSTL formulae. This has adverse effects on the performance. Specifically, the algorithm aims to optimize for the overall formula structure, i.e., for all valuations the structure should be good, which is too conservative. In particular, even though changing the formula structure according to the DAG offers guarantees in terms of the language, it does not imply an improvement in terms of the misclassification rate, the metric of interest for a classification problem. Bartocci et al. (2014); Bufo et al. (2014) also tackled the two-class problem. Their approach can be divided in two separate steps. First, they build two generative

models, one for each class. The models have to be in the form of stochastic systems and are used to compute the probability of satisfaction of a formula. Second, a discriminative formula is obtained by searching a formula that maximizes the odds of being true for the first model and false for the other model. As with other approaches, the formula structure and parameters are optimized separately. In particular, the formula structure is constructed through heuristics (Bartocci et al., 2014) or with a genetic algorithm (Bufo et al., 2014), whereas the parameter space is explored through statistical model checking. This approach presents some disadvantages. Primarily, it needs to build models of the system under analysis. This requires a domain expert and a certain amount of data for model parameter selection and model validation. The parameter optimization process, based on extensive simulation of the models, is expensive. Nenzi et al. (2018) proposed another approach based on genetic algorithms. Here, however, the structure construction uses STL primitives and the parameter mining applies a Gaussian Process optimization scheme which attempts to maximize the gap between robustness of normal traces and robustness of anomalous traces.

### 2.3 Other Related Work

Gol et al. (2014); Grosu et al. (2009) used a learning procedure for formulae defined in particular spatial superposition logics. These logics were developed for describing patterns in images without a time component. Specifically, Linear Spatial Superposition Logic (LSSL) was introduced in (Grosu et al., 2009), and the Tree Spatial Superposition Logic (TSSL) was introduced in (Gol et al., 2014).

Every image is represented with a multi-resolution format using a fixed height quad-tree data structure (which should not be confused with a decision tree). In this representation, every node contains an attribute describing an area of the image. Nodes that appear at deeper levels provide information about smaller areas. A pattern in an image corresponds to a path (Grosu et al., 2009) or a combination of several paths (Gol et al., 2014) in the relative quad-tree. Therefore, to describe the patterns,

the semantics of these spatial logics are defined over the paths of quad-trees. In these works, formulae are learned from a labeled set of paths (Grosu et al., 2009) or a labeled set of quad-trees (Gol et al., 2014) by applying off-the-shelf rule-based learning algorithms to the attributes of the nodes.

Finally, there is a considerable amount of research on Boolean Logic (for an overview, see (Kearns and Vazirani, 1994)), and some recent efforts on learning Linear Temporal Logic formulae (Neider and Gavran, 2018; Shah et al., 2018).

## Chapter 3

# Preliminaries

### 3.1 Signal Temporal Logic

A temporal logic is a system of rules and symbols used for reasoning about propositions whose truth values change over time. Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are the most commonly used temporal logics (Clarke et al., 1999). Signal Temporal Logic (STL) has emerged as a generalization of LTL, where time is continuous and the predicates are defined over real values (Maler and Nickovic, 2004). STL has found significant 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  $\mathcal{S} = \{s : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n\}$  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 thesis, 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  $s_i$ ,  $i \in \{1, \dots, n\}$ . The set  $\mathcal{F}$  contains the projection operators from a signal  $s$  to one of its components  $s_i$ , that is  $\mathcal{F} = \{f_i : \mathbb{R}^n \rightarrow \mathbb{R}, f_i(s) = s_i, i \in \{1, \dots, n\}\}$ .<sup>1</sup> 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 (Maler and

---

<sup>1</sup>A more general definition of the set  $\mathcal{F}$  is used in (Maler and Nickovic, 2004).

Nickovic, 2004):

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

where  $\top$  is the Boolean *true* constant ( $\perp$  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  $\sim \in \{\leq, >\}$ ;  $\neg$  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 (Maler and Nickovic, 2004):

$$s[t] \models \top \quad \Leftrightarrow \quad \top \quad (3.1)$$

$$s[t] \models f(x) \sim \mu \quad \Leftrightarrow \quad f(s(t)) \sim \mu \quad (3.2)$$

$$s[t] \models \neg\phi \quad \Leftrightarrow \quad \neg(s[t] \models \phi) \quad (3.3)$$

$$s[t] \models (\phi_1 \wedge \phi_2) \quad \Leftrightarrow \quad (s[t] \models \phi_1) \wedge (s[t] \models \phi_2) \quad (3.4)$$

$$s[t] \models (\phi_1 \mathbf{U}_{[a,b]}\phi_2) \quad \Leftrightarrow \quad \exists t_u \in [t+a, t+b] \text{ s.t. } (s[t_u] \models \phi_2) \quad (3.5)$$

$$\wedge (\forall t_1 \in [t, t_u] \ s[t_1] \models \phi_1) \quad (3.6)$$

A signal  $s \in \mathcal{S}$  is said to satisfy an STL formula  $\phi$  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

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

**Remark 3.1.** Even though STL is defined using a dense-time semantics and natively supports predicates over reals, in practice its monitoring algorithms work with *sampled* data and assume signals to be piece-wise constant (or piece-wise linearly interpolated) (Donzé et al., 2013). The sampling rate does not have to be constant.

### 3.2 Quantitative Semantics for STL

Besides the Boolean semantics defined above, STL admits a *quantitative semantics* (Donzé and Maler, 2010; Fainekos and Pappas, 2009). This semantics is 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. Specifically, the robustness degree of signal  $s \in \mathcal{S}$  with respect to formula  $\phi$  at time  $t$  is a function  $r(s, \phi, t)$  recursively defined as

$$r(s, \top, t) = r_{\top} \quad (3.7)$$

$$r(s, f(x) \leq \mu, t) = \mu - f(s(t)) \quad (3.8)$$

$$r(s, \neg\phi, t) = -r(s, \phi, t) \quad (3.9)$$

$$r(s, \phi_1 \wedge \phi_2, t) = \min\{r(s, \phi_1, t), r(s, \phi_2, t)\} \quad (3.10)$$

$$r(s, \phi_1 \mathbf{U}_{[a,b]} \phi_2, t) = \quad (3.11)$$

$$\max_{t_u \in [t+a, t+b]} \left\{ \min \left\{ r(s, \phi_2, t_u), \min_{t_1 \in [t, t_u]} \{r(s, \phi_1, t_1)\} \right\} \right\} \quad (3.12)$$

$$r(s, \mathbf{F}_{[a,b]} \phi, t) = \max_{t_f \in [t+a, t+b]} \{r(s, \phi, t_f)\} \quad (3.13)$$

$$r(s, \mathbf{G}_{[a,b]} \phi, t) = \min_{t_g \in [t+a, t+b]} \{r(s, \phi, t_g)\} \quad (3.14)$$

where  $b > a > 0$  and  $r_{\top} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$  is a large constant representing the maximum value of the robustness. Note that a positive robustness degree  $r(s, \phi, 0)$  implies  $s$  satisfies  $\phi$  (in Boolean semantics). In the following, we denote with  $r(s, \phi)$  the robustness degree at time 0. Interpreting the robustness degree as a quantitative measure of satisfaction is justified by the following proposition (Donzé and Maler, 2010).

**Proposition 3.1.** *Let  $s \in \mathcal{S}$  be a signal and  $\phi$  an STL formula such that  $r(s, \phi) > 0$ . All signals  $s' \in \mathcal{S}$  such that  $\|s - s'\|_{\infty} < r(s, \phi)$  satisfy the formula  $\phi$ , i.e.,  $s' \models \phi$ .*

### 3.2.1 Smooth Robustness

The quantitative semantics and Prop. 3.1 established the robustness degree as more informative metric than the Boolean semantics while formulating optimization problems in various applications (Belta and Sadraddini, 2019). However, the definition in (3.7) involves the minimum and maximum functions. The presence of these functions makes the robustness degree discontinuous and hence hinders the possibility of using smooth optimization algorithms (such as gradient descent). For this reason, some scholars have introduced alternative definitions of the robustness degree by using various smooth approximations for the minimum and maximum (Jha et al., 2017; Li et al., 2018; Pant et al., 2017). In this thesis, when a smooth robustness is used, we employ the following differentiable approximation of the minimum and maximum functions:

$$\begin{aligned} \text{smoothmin}_\alpha(x_1, \dots, x_n) &= \frac{\sum_{i=1}^n x_i e^{-\alpha x_i}}{\sum_{i=1}^n e^{-\alpha x_i}} \\ \text{smoothmax}_\alpha(x_1, \dots, x_n) &= \frac{\sum_{i=1}^n x_i e^{\alpha x_i}}{\sum_{i=1}^n e^{\alpha x_i}} \end{aligned}$$

where  $\alpha \geq 0$  is a parameter that controls the smoothness of the approximation. That is, for  $\alpha = 0$ ,  $\text{smoothmin}_\alpha$  is equal to the arithmetic mean, while when  $\alpha \rightarrow \infty$ ,  $\text{smoothmin}_\alpha \rightarrow \min$ . To avoid ambiguities, we denote the resulting smooth approximation of the robustness function with  $\rho(\cdot)$ .

## 3.3 Parametric Signal Temporal Logic

*Parametric Signal Temporal Logic* (PSTL) was introduced in (Asarin et al., 2012) 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 the 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  $\psi$  is a PSTL formula, then every parameter assignment  $\theta \in \Theta$  (where  $\Theta$  is the parameter space of  $\psi$ ) induces a corresponding STL formula  $\phi = \psi(\theta)$ , where all

the space and time parameters of  $\psi$  have been fixed according to  $\theta$ . This assignment is also referred to as valuation  $\theta$  of  $\psi$ . 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  $\psi(\theta) = \mathbf{F}_{[1.1,2.3]}(f_1(x) > 3.7)$ .

### 3.4 Signal Distances

To measure the similarity between two signals, a *distance* function is 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:<sup>2</sup>

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

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 (Ratanamahatana and Keogh, 2004). The core idea is to align two series by warping the time axis iteratively until an optimal alignment is found. There are two possible extensions of DTW to the multi-dimensional signal case: 1) independent warping of each dimension (DTW<sub>I</sub>), or 2) coordinated/dependent warping along all signal dimensions (DTW<sub>D</sub>). Refer to (Shokoohi-Yekta et al., 2015) for a formal definition.

While Euclidean distance is sensitive to distortions in the time axis, DTW provides flexibility to match signals that are similar but locally out of phase (Ratanamahatana and Keogh, 2004). Usually, a restriction on the amount of allowed warping is imposed as a 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 is often application dependent.

---

<sup>2</sup>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.

## Chapter 4

# Two-Class Classification

### 4.1 Introduction and Problem Formulation

We want to find an STL formula that separates traces produced by a system that exhibit some desired property, such as behaving correctly, from other traces of the same system. The normal working conditions are often referred to as *targets*, or *positives*, whereas the non-conforming patterns are usually referred to as *anomalies*, or *negatives*. Let  $C = \{C_p, C_n\}$  be the set of classes, with  $C_p$  standing for the positive class and  $C_n$  for the negative class. Let  $s^i \in \mathcal{S}$  be an  $n$ -dimensional signal, and let  $l^i \in C$  be its label. We consider the following problem:

**Problem 1** (Two-Class Classification). Given a dataset of labeled signals  $S_{\text{ds}} = \{(s^i, l^i)\}_{i=1}^N$ , we want to find an STL formula  $\phi^*$  such that the misclassification rate  $\text{MCR}(\phi, S_{\text{ds}})$  is minimized, where the misclassification rate is defined as:

$$\text{MCR}(\phi, S_{\text{ds}}) := \frac{|\{s^i \mid (s^i \models \phi, l^i = C_n) \text{ or } (s^i \not\models \phi, l^i = C_p)\}|}{|S_{\text{ds}}|}$$

In the above formula,  $(s^i \models \phi, l^i = C_n)$  represents a *false positive*, while  $(s^i \not\models \phi, l^i = C_p)$  represents a *false negative*.

### 4.2 STL formulae and Decision Trees

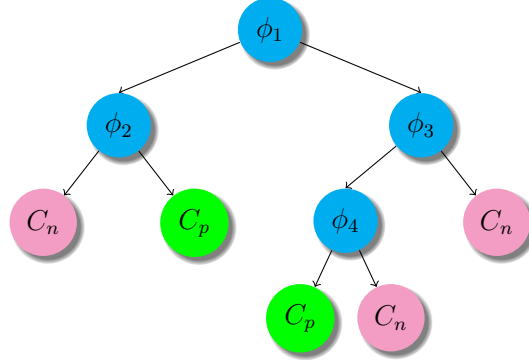
Our key insight to tackle Problem 1 is that it is possible to build a map between a fragment of STL and decision trees. Consequently, we can exploit and extend the decision tree learning literature (Breiman et al., 1984; Quinlan, 2014; Ripley, 1996)

to build a decision tree that classifies signals and map the constructed tree to an STL formula.

A decision tree is a tree-structured sequence of questions about the data used to make predictions about the data’s labels. 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, all other nodes are called non-terminal nodes. We focus on *binary* trees, where every non-terminal node splits the data in two children nodes and every leaf predicts a class.

Unfortunately, the space of all possible decision trees for a given classification problem is very large, and it is known that the problem of learning the optimal decision tree is NP-complete (Hyafil and Rivest, 1976). Most decision-tree learning algorithms are based on *greedy* approaches, where locally optimal decisions are taken at each node. These greedy induction algorithms can be stated in a simple recursive fashion, starting from the root node, and require two core components: 1) a list of possible ways to split the signals reaching a node; and 2) an optimality criterion to select the best split. Several learning algorithms can be created by selecting different components, called here *meta-parameters*. That is, once the user fixes the meta-parameters, a specific algorithm is *instantiated*. We propose to split the signals using a simple formula at each node, chosen from a finite set of PSTL formulae, called *primitives* (Sec. 4.3). The optimality of each candidate formula for a node is assessed using an appropriately defined *impurity measure*, which captures how well it splits the signals reaching that node (Sec. 4.4). Since we are not just proposing a single algorithm but a class of algorithms, we refer to this approach as “decision tree learning framework for temporal logic inference”.

The induction procedure is presented in detail in Sec. 4.5 and a resulting tree can be mapped to the equivalent STL formula using the simple algorithm described in Sec. 4.6. Figure 4-1 shows a tree and its corresponding STL formula. In Sec. 4.8 we



**Figure 4.1:** Binary tree with PSTL primitives at the internal nodes. The formula associated with the tree is  $\phi_{tree} = (\phi_1 \wedge \neg\phi_2) \vee (\neg\phi_1 \wedge (\phi_3 \wedge \phi_4))$  and can be obtained using Alg. 2.

discuss the link between depth of the tree and formula complexity. We define some termination conditions for the tree induction algorithm along with a post-completion pruning strategy. We conclude in Sec. 4.9 with an analysis of the computational complexity of the main algorithm.

### 4.3 PSTL primitives

To partition the data at each node, a finite list of possible splitting rules is considered (Ripley, 1996). We propose to use simple PSTL formulae, called *primitives*, to split the data. In particular, we define two types of primitives:

**Definition 4.1** (First-Level Primitives). Let  $\mathcal{S}$  be the set of signals with values in  $\mathbb{R}^n$ . We define the set of first-level primitives as follows:

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

The parameters for the PSTL formulae in  $\mathcal{P}^1$  are  $(\mu, \tau_1, \tau_2)$  and the respective space of parameters is  $\Theta^1 = \{(\mu, \tau_1, \tau_2) \mid \mu \in \mathbb{R}, \tau_1 < \tau_2, \tau_1, \tau_2 \in \mathbb{R}_{\geq 0}\}$ .

**Definition 4.2** (Second-Level Primitives). Let  $\mathcal{S}$  be the set of signals with values in  $\mathbb{R}^n$ . We define the set of second-level primitives as follows:

$$\mathcal{P}^2 = \left\{ \mathbf{G}_{[\tau_1, \tau_2]} \mathbf{F}_{[0, \tau_3]}(f_i(x) \sim \mu) \text{ or } \mathbf{F}_{[\tau_1, \tau_2]} \mathbf{G}_{[0, \tau_3]}(f_i(x) \sim \mu) \right. \\ \left. \mid i \in \{1, \dots, n\}, \sim \in \{\leq, >\} \right\}$$

The parameters for the PSTL formulae in  $\mathcal{P}^2$  are  $(\mu, \tau_1, \tau_2, \tau_3)$  and the respective space of parameters is  $\Theta^2 = \{(\mu, \tau_1, \tau_2, \tau_3) \mid \mu \in \mathbb{R}, \tau_1 < \tau_2, \tau_1, \tau_2, \tau_3 \in \mathbb{R}_{\geq 0}\}$ .

The meaning of first-level primitives is straightforward. The two primitives  $\mathbf{F}_{[\tau_1, \tau_2]}(f_i(x) \sim \mu)$  and  $\mathbf{G}_{[\tau_1, \tau_2]}(f_i(x) \sim \mu)$  are used to express that the predicate  $f_i(x) \sim \mu$  must be true for at least one time instance or for all time instances in the interval  $[\tau_1, \tau_2]$ , respectively. Similarly, the second-level primitives can be interpreted in natural language as: (a)  $\mathbf{F}_{[\tau_1, \tau_2]} \mathbf{G}_{[0, \tau_3]}(f_i(x) \sim \mu)$  specifies that “the predicate  $(f_i(x) \sim \mu)$  must hold true for  $\tau_3$  seconds and its start time is in the interval  $[\tau_1, \tau_2]$ ”; and (b)  $\mathbf{G}_{[\tau_1, \tau_2]} \mathbf{F}_{[0, \tau_3]}(f_i(x) \sim \mu)$  specifies that “at each time instance in the interval  $[\tau_1, \tau_2]$ , the predicate  $(f_i(x) \sim \mu)$  must be true within  $\tau_3$  time units”. Both first- and second-level primitives may be thought as specifications for bounded reachability and safety with varying degrees of flexibility.

**Remark 4.1.** It is important to stress that the proposed PSTL primitives are not the only possible ones. A user may define other primitives, either generic ones, like the first- and second- level primitives, or specific ones, guided by the particular nature of the learning problem at hand.

Given a set of primitives  $\mathcal{P}$ , we denote by  $\text{STL}_{\mathcal{P}}$  the STL fragment obtained by Boolean closure from  $\mathcal{P}$ .

**Definition 4.3** (Boolean Closure). Let  $\mathcal{P}$  be a finite set of PSTL formulae. The fragment of STL formulae induced by  $\mathcal{P}$  using Boolean closure is defined as:

$$\phi ::= \top \mid \varphi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

where  $\varphi$  is a valuation of a PSTL formula from  $\mathcal{P}$ .

$\text{STL}_{\mathcal{P}}$  is the fragment of STL that is mapped with decision trees. In other terms, each decision tree constructed with the set of primitives  $\mathcal{P}$  is mapped to an STL formula belonging to the  $\text{STL}_{\mathcal{P}}$  fragment.

#### 4.4 Impurity measures

In the previous section, we defined a list of ways to split the data using a set of primitives  $\mathcal{P}$ . It is also necessary to define a criterion to select which primitive best splits the data at each node. Intuitively, a good split leads to children that are *pure*, that is, they contain mostly objects belonging to the same class. This concept has been formalized in literature with *impurity measures* (Breiman et al., 1984; Quinlan, 2014), and the goal is to obtain children *purier* than their parents. In this section, we first state the canonical impurity measures and then we propose three modified measures, which are more suited to handle signals, using the robustness degree.

**Definition 4.4** (Impurity Measures). Let  $S$  be a finite set of signals and  $\phi$  an STL formula. The following partition weights are introduced to describe how the signals  $s^i$  are distributed according to their labels  $l^i$  and the formula  $\phi$ :

$$p_{\top} = \frac{|S_{\top}|}{|S|}, p_{\perp} = \frac{|S_{\perp}|}{|S|}, p_{C_p} = \frac{|S_{C_p}|}{|S|}, p_{C_n} = \frac{|S_{C_n}|}{|S|} \quad (4.1)$$

where  $S_{\top} = \{(s^i, l^i) \in S \mid s^i \models \phi\}$ ,  $S_{\perp} = \{(s^i, l^i) \in S \mid s^i \not\models \phi\}$ ,  $S_{C_p} = \{(s^i, l^i) \in S \mid l^i = C_p\}$ , and  $S_{C_n} = \{(s^i, l^i) \in S \mid l^i = C_n\}$ . In other words,  $p_{\top}$  and  $p_{\perp}$  represent the fraction of signals from  $S$  present in  $S_{\top}$  and  $S_{\perp}$ , respectively, whereas  $p_{C_p}$  and  $p_{C_n}$  represent the fraction of signals in  $S$  belonging to class  $C_p$  and  $C_n$ , respectively.

The (canonical) impurity measures are defined as (Breiman et al., 1984; Quinlan, 2014):

- *Information gain (IG)*

$$\begin{aligned}
 IG(S, \phi) &= H(S) - \sum_{\otimes \in \{\top, \perp\}} p_{\otimes} \cdot H(S_{\otimes}) \\
 H(S) &= -p_{C_p} \log p_{C_p} - p_{C_n} \log p_{C_n}
 \end{aligned} \tag{4.2}$$

- *Misclassification gain (MG)*

$$\begin{aligned}
 MG(S, \phi) &= MR(S) - \sum_{\otimes \in \{\top, \perp\}} p_{\otimes} \cdot MR(S_{\otimes}) \\
 MR(S) &= \min(p_{C_p}, p_{C_n})
 \end{aligned} \tag{4.3}$$

Intuitively, a positive value for one of the impurity measure, such as the Information gain  $IG(S, \phi)$ , means that we have *reduced the impurity* by splitting the set  $S$  with the formula  $\phi$  (or, equivalently, we have *gained purity*).

We also propose an extended set of impurity measures that exploits the robustness degree. The idea behind these new impurity measures is to more explicitly take into consideration the classification quality of the signals with respect to a given formula. One natural choice to measure the classification quality for signals is the robustness degree as implied by Prop. 3.1.

**Definition 4.5** (Extended Impurity Measures). Consider the same setup as in Def. 4.4, and the same impurity measures. We redefine the partition weights as follows:

$$\begin{aligned}
 p_{\top}^{(r)} &= \frac{\sum_{s^i \in S_{\top}} r(s^i, \phi)}{\sum_{s^i \in S} |r(s^i, \phi)|} & p_{\perp}^{(r)} &= \frac{\sum_{s^i \in S_{\perp}} r(s^i, \phi)}{\sum_{s^i \in S} |r(s^i, \phi)|} \\
 p_{C_p}^{(r)} &= \frac{\sum_{s^i \in S_{C_p}} |r(s^i, \phi)|}{\sum_{s^i \in S} |r(s^i, \phi)|} & p_{C_n}^{(r)} &= \frac{\sum_{s^i \in S_{C_n}} |r(s^i, \phi)|}{\sum_{s^i \in S} |r(s^i, \phi)|}
 \end{aligned} \tag{4.4}$$

The extended impurity measures are similar to the classical ones, however, in this case each signal is given a weight proportional to its robustness. We will distinguish between the usual impurity measures and the extended ones by using the superscript

( $r$ ) (e.g.,  $IG^{(r)}$ ) for the extended impurity measures.

The following proposition ensures the extended impurity measures are well defined.

**Proposition 4.1.** *The intra-partition weights are bounded within 0 and 1 and sum to 1, i.e.,  $0 \leq p_{\top}, p_{\perp} \leq 1$  and  $p_{\top} + p_{\perp} = 1$ , in both definitions Def. 4.4 and Def. 4.5. The same invariant property is true for the inter-partition weights, i.e.,  $0 \leq p_{C_p}, p_{C_n} \leq 1$  and  $p_{C_p} + p_{C_n} = 1$ .*

## 4.5 Parameterized learning algorithm

In Alg. 1 we present the parameterized procedure for inferring temporal logic formulae from data. The *meta-parameters* of Alg. 1 are: (1) a set of PSTL primitives  $\mathcal{P}$ ; (2) an impurity measure  $J$ ; which were defined in the previous sections. A set of stopping conditions *stop* is also necessary for determining the algorithm termination. We discuss them in Sec. 4.8.

---

### Algorithm 1: Parameterized Decision Tree Construction – *buildTree*( $\cdot$ )

---

**Meta-Parameter:**  $\mathcal{P}$  – set of PSTL primitives  
**Meta-Parameter:**  $J$  – impurity measure  
**Parameter:** *stop* – set of stopping criteria  
**Input:**  $S = \{(s^i, l^i)_{i=1}^N\}$  – set of labeled signals  
**Input:**  $h$  – the current depth level  
**Output:** a (sub)-tree

```

1 if stop( $h, S$ ) then
2    $t \leftarrow \text{leaf}(\arg \max_{c \in C} \{p_c\})$ 
3   return  $t$ 
4  $\phi^* \leftarrow \arg \max_{\psi \in \mathcal{P}, \theta \in \Theta} J(S, \psi(\theta))$ 
5  $t \leftarrow \text{non\_terminal}(\phi^*)$ 
6  $S_{\top}^*, S_{\perp}^* \leftarrow \text{partition}(S, \phi^*)$ 
7  $t.\text{left} \leftarrow \text{buildTree}(S_{\top}^*, h + 1)$ 
8  $t.\text{right} \leftarrow \text{buildTree}(S_{\perp}^*, h + 1)$ 
9 return  $t$ 

```

---

Alg. 1 is recursive and takes as input arguments the set of data  $S$  that reaches the current node, and the current depth level  $h$ . At the beginning, the stopping conditions

are checked (line 1). If they are met, the algorithm returns a single leaf node marked with the label  $c \in C$ . The label  $c$  is chosen according to the majority vote over data reaching that leaf (line 2). If the stopping conditions are not met (line 4), the algorithm proceeds to find the optimal STL formula among all the valuations of PSTL formulae from the set of primitives  $\mathcal{P}$ . The cost function used in the optimization is the impurity measure  $J$ , which assesses the quality of the partition induced by PSTL primitives valuations. At line 5, a new non-terminal node is created and associated with the optimal STL formula  $\phi^*$ . Next, the partition induced by the formula  $\phi^*$  is computed (line 6). For each outcome of the split, the *buildTree()* procedure is called recursively to construct the left and right subtrees (lines 7-8). The corresponding data partition are passed. The depth level is increased by one.

The parameterized family of algorithms uses three procedures: (a) *leaf(c)* creates a leaf node marked with the label  $c \in C$ , (b) *non\_terminal( $\phi$ )* creates a non-terminal node associated with the valuation of a PSTL primitive from  $\mathcal{P}$ , and (c) *partition( $S, \phi$ )* splits the set of signals  $S$  into satisfying and non-satisfying signals with respect to  $\phi$ .

By fixing the meta-parameters  $(\mathcal{P}, J)$  and a set of stopping conditions (*stop*), a particular algorithm is *instantiated*. For each possible instance, a decision tree is obtained by executing *buildTree( $S_{ds}, 0$ )* on a batch set of labeled signals  $S_{ds}$ . Clearly, the returned tree depends on both the input data  $S_{ds}$  and the particular instance chosen.

## 4.6 Tree to STL formula

A decision tree obtained by an instantiation of Alg. 1 can be used directly for classification or converted to an equivalent STL formula using Alg. 2. This algorithm recursively traverses the tree, starting from the root, and only keeps track of the paths reaching leaves associated with the positive class  $C_p$ . At each node, the formula is obtained by (1) conjunction of the node's formula with its left subtree's formula, (2) conjunction of the negation of the node's formula with its right subtree's formula, (3)

---

**Algorithm 2:** Tree to formula –  $Tree2STL(\cdot)$ 


---

**Input:**  $t$  – node of a tree  
**Output:**  $\phi$  – STL Formula

```

1 if  $t$  is a leaf and class associated with  $t$  is  $C_p$  then
2   | return  $\top$ 
3 if  $t$  is a leaf and class associated with  $t$  is  $C_n$  then
4   | return  $\perp$ 
5  $\phi_l = (t.\phi \wedge Tree2STL(t.left))$ 
6  $\phi_r = (\neg t.\phi \wedge Tree2STL(t.right))$ 
7 return  $\phi_l \vee \phi_r$ 

```

---

disjunction of (1) and (2). Fig. 4-1 shows a simple tree and its corresponding formula obtained by applying Alg. 2.

## 4.7 Local node optimization

The cost function used in the local node optimization (line 4 of Alg. 1) is one of the impurity measures defined in the Sec. 4.4. The local node optimization strategy presents some advantages.

The optimization is performed over the chosen set of PSTL primitives  $\mathcal{P}$  and their valuations  $\Theta$ , Therefore, the optimization problem is always decomposed into  $|\mathcal{P}|$  problems over a fixed and small number of real-valued parameters. In other terms, the complexity of the optimization problem does *not* depend on the length of the overall formula. Refer to Sec. 7.2 for more details about the optimization algorithms employed.

The second advantage is due to the divide-and-conquer nature of Decision Trees. In Alg. 1), the signals are partitioned between the children of the currently processed node and, consequently, the optimization becomes easier as the depth of the tree increases (fewer data to process).

## 4.8 Stop conditions, Pruning, and Formula Complexity

Due to the recursive nature of decision trees, Alg. 1 can achieve perfect classification accuracy on the *training* data, if the maximum depth of the tree is unconstrained. This trivially occurs when we keep splitting the data until the current node contains only one signal (assuming there are no two exact signals with different labels).

This strategy is undesirable for several reasons. First, by fitting the training data perfectly, we are likely to model the noise contained in the data. In this scenario, the resulting formula performance on unseen signals will be poor for a lack of generalization ability (over-fitting). Inducing a deep tree will also lead to a longer execution time. Moreover, since in our approach there is a direct connection between depth of the tree and length of the corresponding STL formula (Sec. 4.6), a deeper tree will result in less interpretable formula.

To deal with these problems, two approaches are possible: 1) introduce more restrictive stopping conditions and 2) prune (merge back) unnecessary parts of the tree after its construction.

Several stopping criteria can be set for Alg. 1. For instance, stop if the vast majority of the signals belong to the same class, either positive or negative, e.g., stop if 99% of signals belong to the same class. Another common strategy is to stop if the algorithm has reached a certain, fixed, depth. These conditions also provide a faster termination of the algorithm.

For post-completion pruning, a popular method is cost-complexity pruning (Hastie et al., 2016). In this approach, a progressive sequence of shallower trees (corresponding to simpler formulae) is derived from the initial tree. Later, an independent set of data is used to pick the best tree in this sequence.

## 4.9 Computational Complexity

In this section, we provide a worst-case and average-case complexity analysis of Alg. 1 in terms of the complexity of the local optimization procedure (Alg. 1, line 4). This complexity analysis assumes that just the sufficient stopping conditions are set. Let  $C(N)$  and  $g(N)$  be the complexity of Alg. 1 and of the local optimization algorithm, respectively, where  $N$  is the number of signals to be processed by the algorithms. Trivially, we have  $g(N) = \Omega(N)$ , where  $\Omega(\cdot)$  is the asymptotic notation for lower bound (Cormen, 2009), because the algorithm must at least check the labels of all signals. The worst-case complexity of Alg. 1 is attained when at each node the optimal partition has size  $(1, N - 1)$ . In this case, the complexity satisfies the recurrence  $C(N) = C(N - 1) + C(1) + g(N)$ , which implies  $C(N) = \Theta(N + \sum_{k=2}^N g(k))$ , where  $\Theta(\cdot)$  is the two-sided asymptotic notation for complexity bound (Cormen, 2009). However, the worst-case scenario is not likely to occur in large datasets. Therefore, we consider the average case where at least a fraction  $\gamma \in (0, 1)$  of the signals are in one set of the partition. The recurrence relation becomes  $C(N) = C(\gamma N) + C((1 - \gamma)N) + g(N)$ , which implies the following complexity bound

$$C(N) = \Theta \left( N \cdot \left( 1 + \int_1^x \frac{g(u)}{u^2} \mathrm{d}u \right) \right)$$

obtained using the Akra-Bazzi method (Cormen, 2009). Finally, note that the hidden constants in the complexity bounds above depend on the cardinality of the set of primitives considered and the size of their parameterization.

## 4.10 Case Study: Maritime Surveillance

This case study emulates a port surveillance problem, where the goal is to detect suspicious vessels approaching the harbor from the open sea by looking at their trajectories. It is assumed that the routes of the ships can be tracked in an area of interest. Kong et al. (2017) proposed this problem based on the scenarios described

$(J, \mathcal{P})$	MCR ( $D = 1$ )	MCR ( $D = 3$ )	MCR ( $D = 5$ )
$(MG, \mathcal{P}^1)$	19.36% (0.46%)	1.31% (0.28%)	0.55% (0.33%)
$(MG, \mathcal{P}^2)$	19.24% (0.31%)	1.44% (0.18%)	0.45% (0.20%)
$(IG, \mathcal{P}^1)$	23.46% (1.42%)	0.81% (0.11%)	0.31% (0.46%)
$(IG, \mathcal{P}^2)$	21.17% (2.18%)	0.80% (0.18%)	0.07% (0.17%)

**Table 4.1:** Maritime Surveillance - Analysis of classification accuracy as a function of Impurity Measure ( $J$ ), Primitive Set ( $\mathcal{P}$ ) and Maximum Tree Depth ( $D$ ).

in (Kowalska and Peel, 2012).

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 veers back to the open sea. This scenario is compatible with terrorist activity.

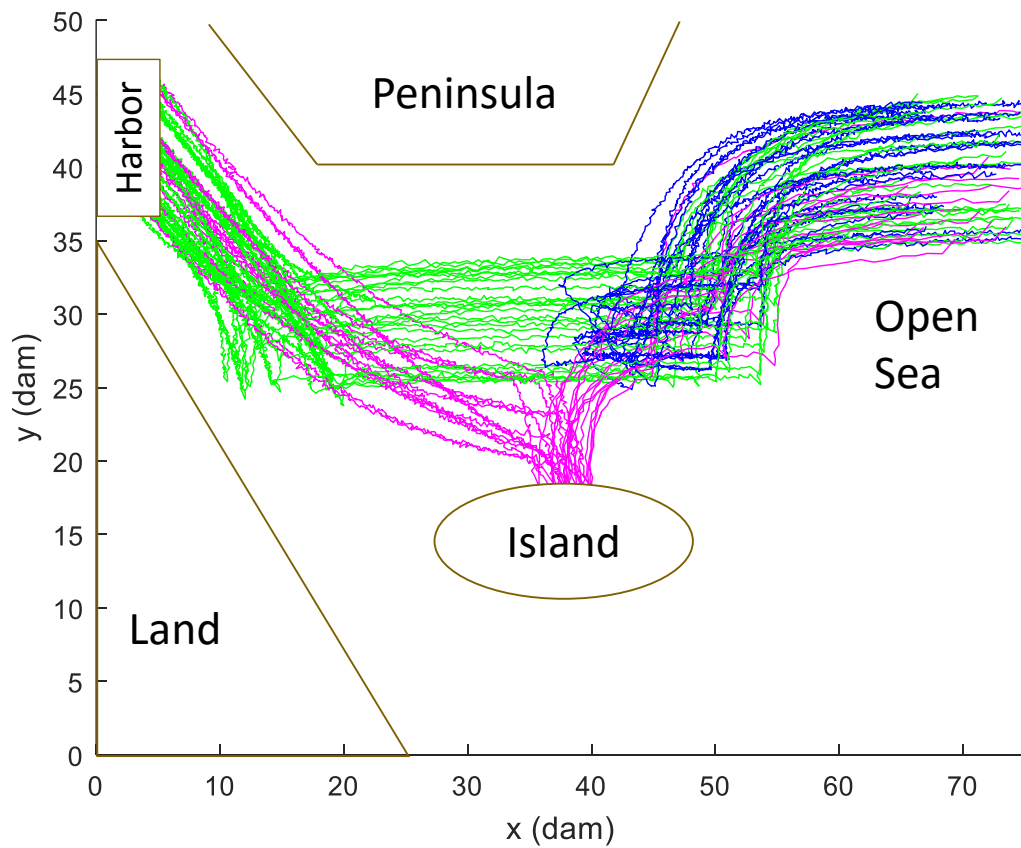
Fig. 4-2 shows some sample trajectories from this dataset. The signals are represented as 2D trajectories with planar coordinates  $[x(t), y(t)]$  and were generated using a Dubins' model with additive Gaussian noise. The dataset is composed of 2000 total trajectories, with 61 sample points per trace. There are 1000 normal trajectories, 500 human trafficking trajectories, and 500 terrorist activity trajectories.

We will use this case study as a running example throughout the thesis to explain the problems addressed and test our proposed algorithms.

## 4.11 Maritime Surveillance: Two-Class Classification Results

### Meta-Parameters Analysis

As discussed in Sec. 4.2, Alg. 1 represents a *family* of algorithms and several possible algorithms can be created depending on the chosen impurity measure and primitive set (the *meta-parameters*). There is a large number of possible combination, and we attempt to investigate at least some of them in Table 4.1 for the maritime surveillance



**Figure 4.2:** Maritime Surveillance - Example Trajectories: the vessels behaving normally are shown in green. The magenta and blue trajectories represent two types of anomalous paths (human trafficking and terrorism, respectively).

Instance	Primitives	Impurity	Stopping
$I_1$	$\mathcal{P}^1$	$MG_r$	Depth > 4
$I_2$	$\mathcal{P}^2$	$IG_r$	Depth > 3

**Table 4.2:** Maritime Surveillance - Instance selection for testing the extended impurity measures.

case study. For implementation and validation details, refer to Sec. 8.1. As mentioned in Sec. 4.8, all meta-parameter combinations can achieve a good accuracy if the maximum depth of the tree is unconstrained. Therefore, it is more insightful to study how their accuracy behaves as a function of the maximum depth allowed ( $D$ ) on the training data.

The table shows that information gain ( $IG$ ) has a small edge over the misclassification gain ( $MG$ ) as the depth of the tree increases. Especially for the naval surveillance case study,  $MG$  is better than  $IG$  at depth 1, however at depths 3 and 5,  $IG$  gains the lead. This phenomenon is known in literature and can be intuitively explained with the capability of  $IG$  of preparing the data for a better division later, sometimes at the expense of the first splits (Hastie et al., 2016).

The primitive sets  $\mathcal{P}^1$  and  $\mathcal{P}^2$  achieve very similar results. This indicates that the greater expressivity of  $\mathcal{P}^2$  is not needed to describe these case studies. The execution time for  $\mathcal{P}^2$  is higher than  $\mathcal{P}^1$  (about 5 times). Even though both sets have the same number of elements,  $\mathcal{P}^2$  involves a more complicated optimization problem. Specifically, primitives in  $\mathcal{P}^2$  have 4 free parameters, whereas primitives in  $\mathcal{P}^1$  have only 3 free parameters. Moreover, evaluating the primitives in  $\mathcal{P}^2$  is computationally more demanding (due to the presence of nested temporal operators).

The low variance present in all results indicates a strong consistency in the produced formulae during every cross-validation round.

### Extended Impurity Measures Results

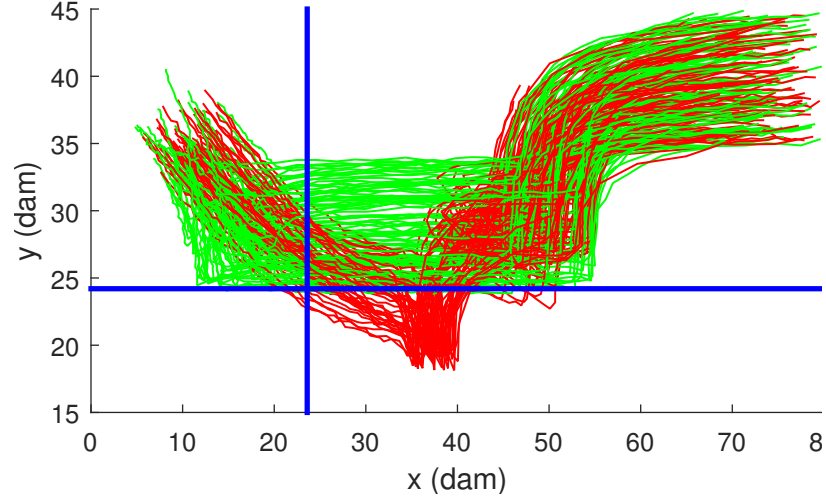
We tested two different instances of Alg. 1,  $I_1$  and  $I_2$ , defined by the choice of meta-parameters given in Table 4.2.

For the  $I_2$  instance, we obtained a mean misclassification rate of 0.7% with a standard deviation of 0.8% and a run time of about 5 minutes per split. A sample formula learned in one of the cross-validation splits is:

$$\begin{aligned}
\phi^{I_2} &= (\phi_1^{I_2} \wedge (\neg\phi_2^{I_2} \vee (\phi_2^{I_2} \wedge \neg\phi_3^{I_2}))) \\
&\quad \vee (\neg\phi_1^{I_2} \wedge (\phi_4^{I_2} \wedge \phi_5^{I_2})) \\
\phi_1^{I_2} &= \mathbf{G}_{[199.70,297.27]} \mathbf{F}_{[0.00,0.05]}(x \leq 23.60) \\
\phi_2^{I_2} &= \mathbf{G}_{[4.47,16.64]} \mathbf{F}_{[0.00,198.73]}(y \leq 24.20) \\
\phi_3^{I_2} &= \mathbf{G}_{[34.40,52.89]} \mathbf{F}_{[0.00,61.74]}(y \leq 19.62) \\
\phi_4^{I_2} &= \mathbf{G}_{[30.96,37.88]} \mathbf{F}_{[0.00,250.37]}(x \leq 36.60) \\
\phi_5^{I_2} &= \mathbf{G}_{[62.76,253.23]} \mathbf{F}_{[0.00,41.07]}(y \leq 29.90)
\end{aligned} \tag{4.5}$$

We can see in Fig. 4.3 how the thresholds for  $\phi_1$  and  $\phi_2$  capture the key features of the data set. Notice also the insight we can gain from their plain English translation: “Normal vessels’  $x$  coordinate is below 23.6 during the last 100 seconds, i.e., they approach and remain at the port”, and “normal vessels’  $y$  coordinate never go below 24.2, i.e., they don’t approach the island”. It is worth mentioning the second term of the outer disjunction in  $\phi^{I_2}$ , as it highlights a feature of the data set difficult to spot on the figures: some normal vessels don’t reach the port (inspecting the data set, some normal traces stop right after crossing the passage). As usual when employing decision trees, deeper formulae focus on finer details of the data set.

In the case of  $I_1$ , we obtained a mean misclassification rate of 0.4% and a standard deviation of 0.29%. The run time is about 1 minute per split. A sample formula



**Figure 4-3:** Maritime Surveillance - Boundary of two PSTL primitives superimposed on a sample of the dataset. Normal trajectories are green and anomalous trajectories are red. We show in blue the boundaries of  $\phi_1^{I_2}$  and  $\phi_2^{I_2}$  of Eq. (4.5).

learned in one of the splits is:

$$\begin{aligned}
 \phi^{I_1} &= (\phi_1^{I_1} \wedge \phi_2^{I_1}) \vee (\neg \phi_1^{I_1} \wedge ((\phi_3^{I_1} \wedge (\phi_4^{I_1} \wedge \phi_5^{I_1})) \vee (\neg \phi_3^{I_1} \wedge \phi_6^{I_1}))) \\
 \phi_1^{I_1} &= \mathbf{G}_{[224,280]}(x \leq 18) & \phi_2^{I_1} &= \mathbf{G}_{[14.2,125]}(y > 22) \\
 \phi_3^{I_1} &= \mathbf{F}_{[109,277]}(y > 30.6) & \phi_4^{I_1} &= \mathbf{G}_{[279,293]}(x \leq 19.2) \\
 \phi_5^{I_1} &= \mathbf{F}_{[77.8,107]}(x > 41) & \phi_6^{I_1} &= \mathbf{F}_{[258,283]}(x \leq 29.5)
 \end{aligned} \tag{4.6}$$

Note the similarity between the subformulae  $\phi_1^{I_2}$  and  $\phi_1^{I_1}$ , or between  $\phi_2^{I_2}$  and  $\phi_2^{I_1}$  in Eq. (4.5) and Eq. (4.6), respectively.

## Chapter 5

# Online Learning

### 5.1 Introduction

In the batch algorithm proposed in Sec. 4.5, a *greedy* recursive procedure is followed to construct the tree, with all the data available  $S_{ds}$ , starting from the root. The data is partitioned as new nodes are created using locally optimal decisions on the signals reaching each node. The decision on which primitive to pick and which parameters to use is made by optimizing an impurity measure  $J$  on a set of primitives  $\mathcal{P}$  and its space of parameter  $\Theta$ .

In this chapter, we tackle the *online learning problem*. Here, new signals may arrive over time and the inference system should be updated to accommodate it. A trivial solution to the online problem would be, every time a new instance arrives, to use the offline learner of Sec. 4.5 from scratch on the whole data accumulated so far. Clearly, this is highly inefficient as any formula discovered, and any associated data structure, would be thrown away. Therefore, the focus of this chapter is to devise a method that *builds* and *updates* an STL formula used for data classification in an efficient manner.

As discussed in (Utgoff et al., 1997), updating a decision tree when new data arrives is not an easy task. Specifically, if the arrival of a new signal causes the change of the best primitive in a node, in the sense of the impurity measure, then that node and all its children should be pruned and reconstructed. A different perspective to deal with this problem has emerged from the study of *data streams* (i.e., a data source that generates ordered sequence of instances, usually at a high rate (Domingos and Hulten,

2000; Jin and Agrawal, 2003)). Instead of creating a node immediately, based on the data currently available, the key idea is to *defer* its creation until we can be *reasonably* sure that the decision made will hold in the future (Domingos and Hulten, 2000). In particular, if an infinite amount of data was available, we would (theoretically) be able to pick the best formula to use at each node. With a finite amount of data, it is not possible to be sure about which formula is the overall best, however a decision on the primitive to pick can be made using probabilistic arguments (Domingos and Hulten, 2000; Jin and Agrawal, 2003; Rutkowski et al., 2015).

We present the online algorithm for constructing the tree in Sec. 5.2, and we describe the details of the decision process behind the creation of a new non-terminal node in Sec. 5.3.

## 5.2 Online learning algorithm

In Alg. 3 we report the online procedure for inferring temporal logic formulae using high-level pseudocode. The algorithm can be executed whenever new signals are available. Alg. 3 operates on a data structure  $T$  representing the tree and takes as input a new labelled signal  $(s, l)$  to be processed. At the beginning, the algorithm checks if the tree exists (line 1). If it does not, it creates a tree with a single leaf at the root (line 2). When a tree exists, the new labelled signal  $(s, l)$  is sorted through the tree to the leaf  $L$  where it belongs and the label  $c \in C$  of this leaf is examined (line 3). The label  $c$  associated with a leaf corresponds simply to the majority of the labels of the signals falling in that leaf. If the new signal is misclassified (line 4), that is  $l \neq c$ , the procedure *updateLeaf()* is invoked on leaf  $L$  (line 5).

The procedure *updateLeaf()*, reported in Alg. 4, operates on a single leaf of the tree and performs three major steps. First, it finds the optimal parameters for each primitive in the set  $\tilde{\mathcal{P}} \subseteq \mathcal{P}$  according to the impurity measure  $J$  (line 1). Second, it evaluates the status of the leaf to decide if it should be kept as a leaf or if a new non-terminal node can be created in its place (line 2). This part is discussed in the

---

**Algorithm 3:** Online Decision Tree Construction
 

---

**Input:**  $(s, l)$  – a new labeled signal  
**Data:**  $T$  – a tree  
**1** **if**  $T$  *does not exist* **then**  
**2**    $T \leftarrow \text{emptyLeaf}()$   
**3**  $L, c \leftarrow \text{locateLeaf}(s, l)$   
**4** **if**  $l \neq c$  **then**  
**5**    $\text{updateLeaf}(L)$

---

next section. Third, if the conditions are met (line 3), the leaf is transformed into a non-terminal node and it is associated with the optimal formula  $\phi_{\text{bst1}}$  (line 4). Two empty leaves are initially added as children of this new node (line 5-6). Finally, the signals  $S$  are partitioned according to  $\phi_{\text{bst1}}$  (line 7), and for each outcome of the split the corresponding partition is passed to the appropriate leaf (lines 8-9).

Algs. 3 and 4 use several functions: a)  $\text{emptyLeaf}()$  creates a leaf with no signals in it and initializes the set of primitives to analyze  $\tilde{\mathcal{P}}$  to  $\mathcal{P}$ ; b)  $\text{locateLeaf}(s, l)$  locates and stores a signal  $s$  with label  $l$  in the leaf  $L$  where it belongs according to the decision tree. c)  $\text{evalLeafStatus}()$  tests the status of the candidate primitives in the leaf under analysis and checks the conditions to create a new node; d)  $\text{storeInLeaf}(L, S)$  stores the signals  $S$  in leaf  $L$ .

**Remark 5.1.** Alg. 4 operates on a single leaf of the tree at the time and only the signals belonging to that specific leaf are required to be in memory. Therefore, this approach can potentially handle large datasets. Alternately, if all accumulated signals can be stored in memory, the addition of new signals can be parallelized over different leaves.

### 5.3 Primitive evaluation and node creation

Hypothetically, if an infinite set of signals  $S_\infty$  was available at a leaf, we would be able to pick the *best* formula to split the signals, both in terms of primitive

---

**Algorithm 4:** Update a Leaf – *updateLeaf*( $\cdot$ )

---

**Meta-Parameter:**  $\mathcal{P}$  – set of PSTL primitives  
**Meta-Parameter:**  $J$  – impurity measure  
**Parameter:**  $\delta$  – confidence threshold  
**Parameter:**  $N_{\max}$  – maximum number of signals  
**Input:**  $L$  – a leaf of tree  $T$   
**Data:**  $S$  – set of signals contained in leaf  $L$   
**Data:**  $\tilde{\mathcal{P}}$  – set of candidate primitives for leaf  $L$

- 1  $\theta_i \leftarrow \arg \max_{\theta \in \Theta} J(S, \psi_i(\theta_i)), \forall \psi_i \in \tilde{\mathcal{P}}$
- 2  $\tilde{\mathcal{P}}, \phi_{\text{bst1}}, \text{createNode} \leftarrow \text{evalLeafStatus}(S, \tilde{\mathcal{P}}, \{\theta_i\}_1^{|\tilde{\mathcal{P}}|})$
- 3 **if**  $\text{createNode} == \text{True}$  **then**
- 4      $N \leftarrow \text{non\_terminal}(\phi_{\text{bst1}})$
- 5      $N.\text{left} \leftarrow \text{emptyLeaf}()$
- 6      $N.\text{right} \leftarrow \text{emptyLeaf}()$
- 7      $S_{\top}, S_{\perp} \leftarrow \text{partition}(S, \phi_{\text{bst1}})$
- 8      $\text{storeInLeaf}(N.\text{left}, S_{\top})$
- 9      $\text{storeInLeaf}(N.\text{right}, S_{\perp})$

---

and its parameters, with respect to the impurity measure (4.3). Assume that  $\phi_{\text{bst1}}$  ( $= \psi_{\text{bst1}}(\theta_{\text{bst1}})$ ) is this formula, corresponding to the primitive  $\psi_{\text{bst1}}$  with optimal valuation  $\theta_{\text{bst1}}$ . Assume also that  $\phi_{\text{bst2}}$  is the best formula obtainable with any other primitive, say  $\psi_{\text{bst2}}$ , we would obviously have:

$$J(S_{\infty}, \phi_{\text{bst1}}) - J(S_{\infty}, \phi_{\text{bst2}}) > 0 \quad (5.1)$$

That is, primitive  $\psi_{\text{bst1}}$  provides an overall higher impurity reduction (purity gain) than  $\psi_{\text{bst2}}$ . With a finite amount of data, it is not possible to be sure about which formula is the best. However, some probabilistic guarantees on the best overall primitive to pick can be obtained using the finite set of signals  $S$  collected so far in the leaf. Following the idea initially proposed in (Domingos and Hulten, 2000), a bound is derived on the difference of purity gains (using just the signals  $S$  available), such that,

$$\text{if} \quad J(S, \phi_{\text{bst1}}) - J(S, \phi_{\text{bst2}}) > \epsilon(S, \delta) \quad (5.2)$$

$$\text{then } \Pr(\Delta J(S_\infty, \phi_{\text{bst1}}, \phi_{\text{bst2}}) > 0) \geq 1 - \delta \quad (5.3)$$

In other words, if, on the  $S$  signals available, the difference between the purity gain of the best formula  $\phi_{\text{bst1}}$ , obtained with the best primitive  $\psi_{\text{bst1}}$ , and the purity gain of the formula  $\phi_{\text{bst2}}$ , obtained with the second best primitive  $\psi_{\text{bst2}}$ , is greater than a certain  $\epsilon$  then, with probability greater than  $1 - \delta$ ,  $\psi_{\text{bst1}}$  is indeed better than  $\psi_{\text{bst2}}$  (as if we had access to infinite signals  $S_\infty$ ). Moreover, if (5.2) holds and since there are  $|\mathcal{P}|$  primitives, we have that  $\psi_{\text{bst1}}$  is the *best* primitive with probability  $(1 - \delta)^{(|\mathcal{P}|-1)}$ .

In literature (Domingos and Hulten, 2000; Jin and Agrawal, 2003; Rutkowski et al., 2015), several approaches have been pursued to obtain a value for  $\epsilon$  in (5.2) in order to guarantee (5.3). They vary on the impurity measure used and on the concentration inequality (Hoeffding, McDiarmid, etc) or probabilistic approximation employed. In this thesis, we investigate only the Misclassification Gain ( $J = MG$ ) and use the bound recently derived in (Rutkowski et al., 2015) using a Gaussian Approximation approach: <sup>1</sup>

$$\epsilon_{MG}(S, \delta) = z_{1-\delta} \frac{1}{\sqrt{2|S|}} \quad (5.4)$$

where  $z_{1-\delta}$  is  $(1 - \delta)$ -th quantile of the normal distribution. In general,  $\epsilon$  depends on the confidence threshold  $\delta$  and on the cardinality of  $S$ .  $\epsilon$  grows as  $\delta$  approaches 0 (i.e., we want more confidence) and becomes smaller as the number of signals acquired increases (i.e., we collected more evidence).

**Remark 5.2.** It may happen that two primitives are almost equally good in terms of impurity reduction. In this scenario, a large number of signals would be required to assess the best one. To avoid a long decision time, the split of a leaf can be forced when more than  $N_{\text{max}}$  signals have been collected (tie breaking). Even in this scenario, the probabilistic bound is useful to speed up the computation because it can be employed

---

<sup>1</sup>In this approach, each gain  $MG(S_\infty, \phi_i)$  is a fixed unknown and the respective  $MG(S, \phi_i)$  is its empirical estimator using  $S$  signals. The behavior of  $MG(S, \phi_i)$  is characterized as a binomial random variable function of  $|S|$  i.i.d signals. Later, it is approximated with a Gaussian r.v.

to progressively eliminate all the non-promising primitives from further analysis, that is, the primitives that compared with the current best satisfy the condition in (5.2).

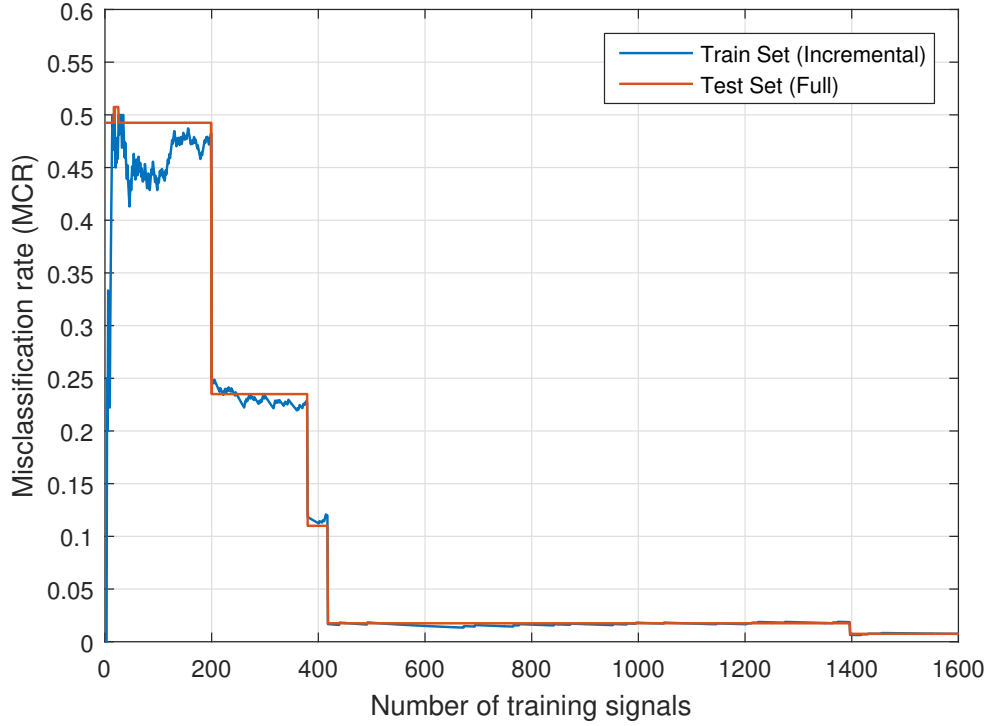
The arguments previously discussed are used in the implementation of the function *evalLeafStatus()* for Alg. 4. This function takes as input arguments the set of signals  $S$  collected so far in the leaf, the current set of candidate PSTL primitives  $\tilde{\mathcal{P}} \subseteq \mathcal{P}$ , and the optimal parameters for each primitive in  $\tilde{\mathcal{P}}$ , that is  $\{\theta_i\}_1^{|\tilde{\mathcal{P}}|}$ . It returns the updated set of PSTL primitives to consider in the future  $\tilde{\mathcal{P}}$ , the best splitting formula  $\phi_{\text{bst1}}$ , and a Boolean *createNode* that indicates whether the leaf should become a new non-terminal node. *evalLeafStatus()* performs three major actions. First, it finds the best primitive, that is, the one associated with the highest purity gain. Second, it removes all the non-promising primitives from set  $\tilde{\mathcal{P}}$  by checking them against the best primitive using (5.2). Third, it sets *createNode* = *True* if only one primitive is left in set  $\tilde{\mathcal{P}}$ , or if the number of signals in the leaf has exceeded the maximum  $|S| > N_{\text{max}}$ .

The specific values of the probability confidence threshold  $\delta$  in (5.3) and of the maximum number of signals  $N_{\text{max}}$  are parameters of Alg. 4 and can be decided by the user.

## 5.4 Maritime Surveillance: Online Learning Results

We used again the cross-validation scheme described in Sec. 8.1. In this case, however, the training signals are presented to Alg. 3 one at the time. We used the misclassification gain (*MG*) as impurity measure and  $\mathcal{P}^1$  as primitive set.

For the maritime surveillance case study, we obtained a mean MCR of 1.45% (STD 0.88%). The mean runtime (to process *all* the signals in the training set) was about 140 seconds per cross-validation round. A sample formula, obtained in one of the



**Figure 5.1:** Maritime Surveillance - MCR as function of the number of signal processed by the online algorithm. In blue, evolution of MCR computed on training signals (as seen by the algorithm). In red, evolution of MCR computed on an independent test set.

rounds after processing 1600 signals, is:

$$\phi = (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge (\phi_3 \vee (\neg\phi_3 \wedge \phi_4))) \quad (5.5)$$

$$\phi_1 = \mathbf{G}_{[211,295]}(x \leq 18.6) \quad \phi_2 = \mathbf{G}_{[69.7,158]}(y > 23.7)$$

$$\phi_3 = \mathbf{G}_{[97.9,299]}(y \leq 32.4) \quad \phi_4 = \mathbf{F}_{[61,178]}(x \leq 22.2)$$

## 5.5 Error evolution analysis

It is interesting to analyze how Alg. 3 performs as signals are incrementally processed. Fig. 5.1 displays how two error rates evolve as training signals from the maritime dataset are presented to the algorithm. The first, in blue, is the misclassification rate on the set of *training* signals seen *so far* by the algorithm, while the second, in red, is

the misclassification rate respect an independent (fixed) *test* set. As expected, the functions are flat for ample intervals with jumps at the points where the algorithm creates a new node in the tree. At these points, the corresponding formula becomes more complex (longer) and, generally, the misclassification rate decreases.

The evolution of the errors on the training and test data provides valuable information. For instance, if the training error decreases while the test error remains stable or increases, the formula inferred is getting overly specific to the training data and will not generalize well on unseen data (overfitting). From Fig. 5-1, it is also clear that after a certain number of signals has been processed, adding more signals, while still increasing the complexity of the formula, does not improve the classification accuracy significantly. This information can be exploited to stop early the learning process (that is, *before* the whole dataset is processed) and focus on a reasonably accurate and more interpretable formula. For example in the maritime surveillance case study, by looking at Fig. 5-1, if we stop after 420 signals have been processed (after around 30 seconds), we obtain a simpler formula (compare with (5.5)):

$$\begin{aligned}\phi &= (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \phi_3) & (5.6) \\ \phi_1 &= \mathbf{G}_{[211,295]}(x \leq 18.6) & \phi_2 = \mathbf{G}_{[69.7,158]}(y > 23.7) \\ \phi_3 &= \mathbf{G}_{[97.9,299]}(y \leq 32.4)\end{aligned}$$

with an associated MCR of 2.5%. This is a form of *online evaluation* of a formula's performance and complexity as opposed to the post-completion pruning procedure described in Sec. 4.8. Notice also the insight we can gain from the English translation of (5.6): “Normal vessels’  $x$  coordinate is below 18.6 during the last 80 minutes, i.e., they approach and remain at the port”, and “normal vessels’  $y$  coordinate never go below 23.7 in the time interval between 69.7 and 158 minutes, i.e., they do not approach the island” (refer to Fig. 4-2).

## 5.6 Comparisons with the offline algorithm

Given the same meta-parameters, Alg. 1 and Alg. 3 are able to produce similar formulae. However, comparing them is not straightforward due to the different problem setting they address. Specifically, Alg. 1 is an offline algorithm, that is, it processes the whole dataset in a single batch and produce a formula, whereas the Alg. 3 processes the dataset one signal at a time and produces a new formula after each addition. In terms of classification performance, the accuracy of the online algorithm is on par with the results of the offline algorithm. However, to elaborate the *whole* maritime surveillance dataset, Alg. 3 is around 2.5 times slower than Alg. 1. In this regard, it is worth noting that the signal addition time of online algorithm is not constant. For the maritime dataset, it goes from less than 1 second to around 5 seconds, depending on whether the algorithm decides to 1) do nothing, 2) only reoptimize the primitives' parameters, 3) create a new node. Moreover, as mentioned in Sec. 5.5, it is often *not* necessary to process the whole dataset available.

## Chapter 6

# Signal Clustering

### 6.1 Introduction and Problem Formulation

In the previous chapters, *supervised* learning problems were considered. For the maritime surveillance case study, 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 algorithms 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 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 known 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.

## 6.2 Hierarchical Clustering with STL

In literature, many algorithms have been proposed to solve clustering problems, such as K-means or DBSCAN (Bishop, 2006). 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 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 binary trees introduced in Sec. 4.2.

In this unsupervised scenario, the primitives should be chosen 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 similarity can be assessed using appropriate distance measures between signals.

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. Fig. 6-2b shows a tree induced by our algorithm for the maritime surveillance case study. Two internal nodes partition the signals in three clusters.

This chapter is organized as follows. In Sec. 6.3, we define suitable *homogeneity measures* used during the optimization process to select the best primitive at each node. In Sec. 6.4, we describe our hierarchical clustering algorithm. We conclude in

Sec. 6.5 with some comments on the evaluation of its results and address the problem of choosing the appropriate number of clusters.

### 6.3 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 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 6.1** (Inertia-based Homogeneity Measure  $I$ ). Let  $S$  be a finite set of signals in  $\mathcal{S}$ , we define

$$I(S) = \frac{1}{2} \frac{1}{|S|^2} \sum_{s^i \in S} \sum_{s^j \in S} d^2(s^i, s^j) \quad (6.1)$$

where  $d(\cdot, \cdot)$  is a suitable distance function between two signals, such as the Euclidean Distance or Dynamic Time Warping (See Sec. 3.1).

$I(S)$  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 6.1.** As the name suggests, the Inertia-based homogeneity measure in Def. 6.1 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 (Aggarwal and Reddy, 2013; Ripley, 1996). 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 (Aggarwal and Reddy, 2013; Bishop, 2006).

**Definition 6.2** (Homogeneity Gain  $HG$ ). Let  $S$  be a finite set of unlabeled signals and  $\phi$  an STL formula, the Homogeneity Gain is defined as

$$HG(S, \phi) = I(S) - \left[ \frac{|S_{\top}|}{|S|} \cdot I(S_{\top}) + \frac{|S_{\perp}|}{|S|} \cdot I(S_{\perp}) \right] \quad (6.2)$$

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

Intuitively, a positive value of  $HG(S, \phi)$  means that by splitting the set  $S$  with the formula  $\phi$ , we obtain two sets  $S_{\top}$  and  $S_{\perp}$  with a *reduced overall diversity* or, equivalently, we *gained homogeneity*. The homogeneity gain in Def. 6.2 is connected to the so-called *Ward's criterion* in the clustering literature (Aggarwal and Reddy, 2013; Ripley, 1996).

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 the resulting STL formula  $\psi^*(\theta^*)$  maximizes the homogeneity gain:

$$\psi^*, \theta^* = \arg \max_{\psi \in \mathcal{P}, \theta \in \Theta} HG(S, \psi(\theta)) \quad (6.3)$$

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. Refer to Sec. 7.2 for more details about the optimization algorithms employed.

## 6.4 Clustering Algorithm

In Alg. 5 we show our parameterized clustering procedure with a high-level object oriented notation. The meta-parameters of Alg. 5 are: 1) a set of PSTL primitives  $\mathcal{P}$ ; 2) three measure functions  $d()$ ,  $I()$ ,  $HG()$  (distance, homogeneity, and homogeneity gain, respectively). 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 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. (6.2)-(6.3). 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  $S_{\top}^*, S_{\perp}^*$  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 Alg. 5. 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 6.2.** We implemented and tested Alg. 5 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, the computation of the objective function  $HG(S, \psi(\theta))$  implicitly requires the construction of the partition  $S_{\top}, S_{\perp}$  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* containing 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 executing the algorithm has proven to be useful.

---

**Algorithm 5:** Hierarchical Clustering Algorithm - *HClustSTL()*


---

**Meta-Parameter:**  $\mathcal{P}$  – set of PSTL primitives  
**Meta-Parameter:**  $d, I, HG$  – distance, homogeneity, and homogeneity gain  
**Parameter:**  $stop$  – set of stopping criteria  
**Input:**  $S_{tr}$  – set of training signals  
**Output:**  $T$  – a Tree

```

1  $T \leftarrow createEmptyTree(S_{tr})$ 
2 while  $stop() \neq false$  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    $S_{\top}^*, S_{\perp}^* \leftarrow partition(L.S, \psi^*(\theta^*))$ 
7    $T.createLeftLeaf(L, S_{\top}^*)$ 
8    $T.createRightLeaf(L, S_{\perp}^*)$ 
9 return  $T$ 

```

---

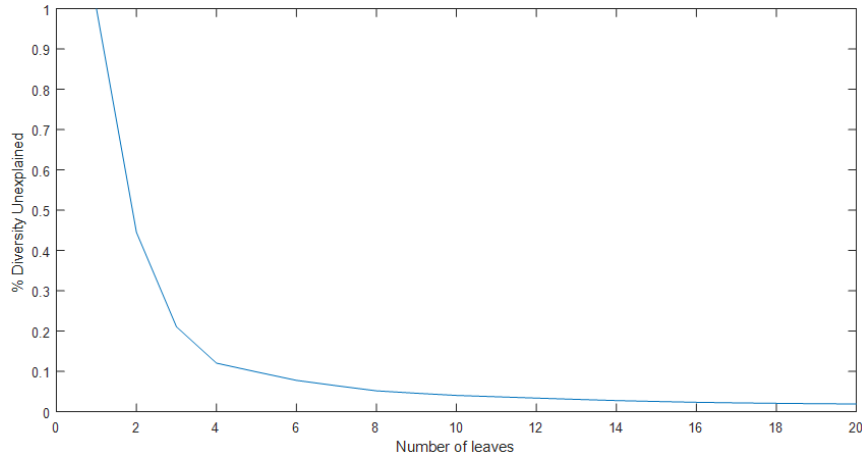
## 6.5 Clustering Evaluation

Evaluating the quality of clustering results is a difficult task (Aggarwal and Reddy, 2013). 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 (Färber et al., 2010). 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. In general, labeled data is not available at all, and clustering algorithms should be interpreted more as *exploratory* and *knowledge discovery* tools (Aggarwal and Reddy, 2013; Ripley, 1996) 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 (Aggarwal and Reddy, 2013). 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 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.

## 6.6 Maritime Surveillance: Clustering Results

For the maritime surveillance case study, we obtained excellent results with the classical Euclidean norm. Alg. 5 was set to run until a tree of depth 4 was constructed. The elbow analysis in Fig. 6.1 shows that the last major drop in unexplained diversity happens when there are three leaves, and this corresponds to the tree shown in Fig. 6.2b. In this case, each cluster can be mapped with one of the original scenarios of the dataset.



**Figure 6.1:** Maritime Surveillance - Elbow Analysis for Clustering

The formula corresponding to the cluster that contains the normal trajectories is:

$$\phi_{norm} = \mathbf{F}_{[16.3,280]}(x \leq 24.832) \wedge \mathbf{G}_{[0,262]}(y > 20.66) \quad (6.4)$$

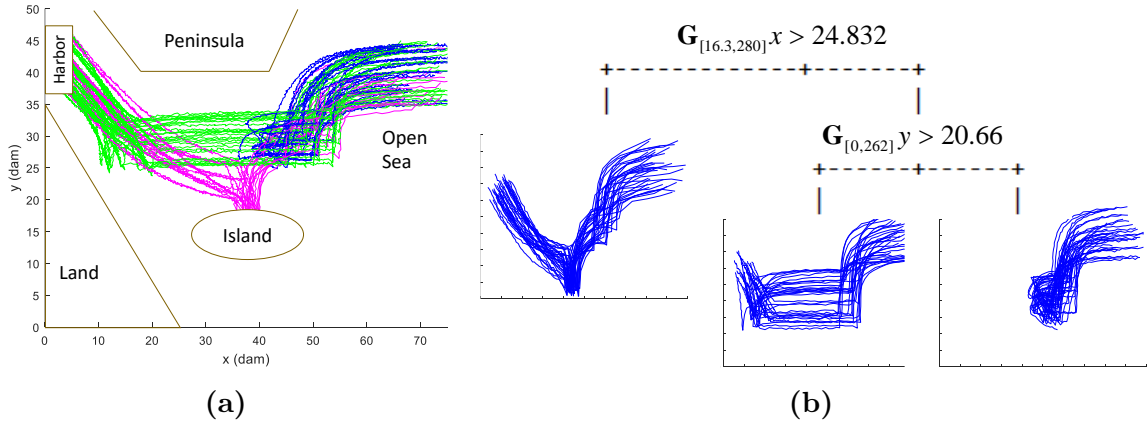
Since labeled data is available for this dataset, we tested the performance of the learned formula in a classification task. Using an independent test set, the formulae achieved an average misclassification rate (rate of incorrect decisions) of 0.79%.

Notice 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.

## 6.7 Comparison with the supervised algorithm

Even though supervised and unsupervised problems are quite different in nature, we can still try to make some comparisons since Alg. 5 was able to recover the same class structure of the original dataset for the naval surveillance case study (as shown in Fig. 6.2).

In terms of classification performance, the formula in Eq. (6.4) achieves a similar accuracy as the one obtained by Alg. 1 (using the same primitive set). This result is



**Figure 6.2:** Maritime Surveillance - Clustering Results. (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.

quite impressive considering that Alg. 5 does *not* use labeled examples and yet it is able to achieve similar accuracy.

For the naval case study, the execution time of the unsupervised algorithm was 283 seconds, which is roughly three times slower with respect to the supervised algorithm. This difference 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.

## Chapter 7

# Multi-Class Classification and Smooth Optimization

### 7.1 Multi-Class Classification

In Chapter 4, we presented an approach for solving the two-class classification problem and producing STL formulae. It is straightforward to extend it to work with multi-class datasets. In this scenario, a signal can belong to one of *multiple* classes ( $C = \{C_i\}_1^N$ ), instead of just two. For example, in the maritime surveillance case study, we can explicitly consider three classes ( $N = 3$ ): normal behavior, human trafficking, and piracy, as opposed to normal and anomalous behaviors only ( $N = 2$ ).

The main tree induction algorithm (Alg. 1) remains unchanged, however in this case it produces a tree where each leaf is linked to one of the possible  $N$  classes ( $C_i \in C$ ). A formula that describes the signals classified as belonging to class  $C_i$  is obtained by using Alg. 6 (a simple generalization of Alg. 2). Alg. 6 recursively traverses the tree, starting from the root, and only keeps track of the paths reaching leaves associated with class  $C_i$ .

The only major modification required is to extend the definition of the impurity measures to account for the presence of multiple classes.

**Definition 7.1** (Multi-Class Information Gain Impurity Measure). Let  $S$  be a finite set of signals and  $\phi$  an STL formula. The following partition weights are introduced to describe how the signals  $s^i$  are distributed according to their labels  $l^i \in C$  and the

---

**Algorithm 6:** Tree to formula – *Tree2STL*( $\cdot$ )

---

**Input:**  $C_i$  – target class  
**Input:**  $t$  – node of a tree  
**Output:**  $\phi$  – STL Formula

- 1 **if**  $t$  is a leaf and class associated with  $t$  is  $C_i$  **then**
- 2   | **return**  $\top$
- 3 **if**  $t$  is a leaf and class associated with  $t$  is  $C_i$  **then**
- 4   | **return**  $\perp$
- 5  $\phi_l = (t.\phi \wedge \text{Tree2STL}(t.\text{left}))$
- 6  $\phi_r = (\neg t.\phi \wedge \text{Tree2STL}(t.\text{right}))$
- 7 **return**  $\phi_l \vee \phi_r$

---

formula  $\phi$ :

$$p_{\top} = \frac{|S_{\top}|}{|S_{\top}| + |S_{\perp}|}, \quad p_{\perp} = \frac{|S_{\perp}|}{|S_{\top}| + |S_{\perp}|}, \quad p_c = \frac{|S_c|}{|S|}, \quad p_{\top c} = \frac{|S_{\top c}|}{|S_{\top}|}, \quad p_{\perp c} = \frac{|S_{\perp c}|}{|S_{\perp}|}, \quad (7.1)$$

where ( $I(\cdot)$  is the indicator function):

$$\begin{aligned}
 |S_{\top}| &= \sum_{(s^i, l^i) \in S} I(s^i \models \phi) & |S_{\perp}| &= \sum_{(s^i, l^i) \in S} I(s^i \models \neg\phi) \\
 |S_c| &= \sum_{\substack{(s^i, l^i) \in S \\ l^i = c}} 1 & & (7.2) \\
 |S_{\top c}| &= \sum_{\substack{(s^i, l^i) \in S \\ l^i = c}} I(s^i \models \phi) & |S_{\perp c}| &= \sum_{\substack{(s^i, l^i) \in S \\ l^i = c}} I(s^i \models \neg\phi)
 \end{aligned}$$

In other words,  $p_{\top}$  and  $p_{\perp}$  represent the fraction of signals from  $S$  present in  $S_{\top}$  and  $S_{\perp}$ , respectively, and  $p_c$ ,  $p_{\top c}$ , and  $p_{\perp c}$  are the fraction of signals belonging to class  $c \in C$  in  $S$ ,  $S_{\top}$ , and  $S_{\perp}$ , respectively.

The Information Gain (IG) is defined as:

$$\begin{aligned}
 IG(S, \phi) &= H(S) - H(S|\phi) \\
 H(S) &= - \sum_{c \in C} p_c \log p_c \\
 H(S|\phi) &= - \sum_{c \in C} p_{\top c} \log p_{\top c} - \sum_{c \in C} p_{\perp c} \log p_{\perp c}
 \end{aligned} \tag{7.3}$$

**Remark 7.1.** Def. 7.1 is equivalent to the one commonly present in literature (Bishop, 2006). However, we chose the form in Def. 7.1 to highlight the presence of the indicator function and clarify the modifications we introduce later.

## 7.2 Primitive optimization procedure

A local node optimization problem is at the core of all algorithms proposed in the previous chapters (Line 4 of Alg. 1, Line 1 of Alg. 4, and Line 4 of Alg. 5). For a given primitive, the goal is to find its best parameters against an objective function (impurity reduction for classification problems, and homogeneity gain for clustering problems).

These optimization problems may be solved using any global *non-smooth* optimization algorithm. Initially, we used Simulated Annealing (Ingber, 1996) and Differential Evolution (Storn and Price, 1997). However, we found that Particle Swarm Optimization (Shi and Eberhart, 1998) delivers superior performance. This has led to an average speedup of 5x with respect to our first results (Bombara et al., 2016).

To use any of these numerical optimization algorithms, we need to define finite bounds for the parameters of the primitive formulae. These bounds may easily be inferred from data, but may also be application-specific if expert knowledge is available.

### 7.3 Smooth objective functions

As mentioned in the previous section, this is a non-smooth optimization problem. Discontinuities are caused by the (hard) partitioning behavior of the objective functions themselves. An indicator function is embedded in the impurity reduction (Def. 7.1) and homogeneity gain (Def. 6.2) functions. Once a primitive and its parameters are fixed, a signal in a node either satisfies or not satisfies the resulting formula and will be routed to the left or to the right child, respectively.

As the optimization algorithm explores the parameter space of a PSTL primitive, the objective function *jumps* from one value to another once a signal is re-routed from the left child to the right child (or vice-versa). This implies that the gradient of the objective function is generally flat with impulses where the function changes value. For this reason, smooth solvers, i.e. solvers that exploit the information provided by the gradient, cannot be employed.

To overcome this problem, we introduce smooth versions of the objective functions that exploit the robustness degree. The idea behind these new objective functions is to more explicitly take into consideration the classification quality of the signals with respect to a given formula. One natural choice to assess the classification quality for signals is the robustness degree as implied by Prop. 3.1. Therefore, we substitute the indicator function  $I(\cdot)$ , guided by the Boolean satisfaction of a formula, with a sigmoid function  $\varsigma_\alpha(\rho) = \frac{1}{1+e^{-\alpha\rho}}$ , guided by the smooth robustness degree  $\rho(s, \phi)$ , a *quantitative* satisfaction measure as defined in Sec. 3.2.1.  $\alpha > 0$  is parameter that regulates the level of smoothing for the sigmoid. When  $\alpha \rightarrow \infty$ , the sigmoid converges to the indicator function.

#### 7.3.1 Smooth Impurity Measures for Multi-Class Classification

**Definition 7.2** (Smooth Multi-Class Impurity Measures). Let  $S$  be a finite set of signals and  $\phi$  an STL formula. The following partition weights are introduced to describe how the signals  $s^i$  are distributed according to their labels  $l^i$  and the formula

$\phi$ :

$$p_{\top} = \frac{|S_{\top}|}{|S_{\top}| + |S_{\perp}|}, \quad p_{\perp} = \frac{|S_{\perp}|}{|S_{\top}| + |S_{\perp}|}, \quad p_c = \frac{|S_c|}{|S|}, \quad p_{\top c} = \frac{|S_{\top c}|}{|S_{\top}|}, \quad p_{\perp c} = \frac{|S_{\perp c}|}{|S_{\perp}|}, \quad (7.4)$$

where:

$$\begin{aligned} |S_{\top}| &= \sum_{(s^i, l^i) \in S} \varsigma(\rho(s^i, \phi)) & |S_{\perp}| &= \sum_{(s^i, l^i) \in S} \varsigma(\rho(s^i, \neg\phi)) \\ |S_c| &= \sum_{\substack{(s^i, l^i) \in S \\ l^i = c}} 1 & & (7.5) \\ |S_{\top c}| &= \sum_{\substack{(s^i, l^i) \in S \\ l^i = c}} \varsigma(\rho(s^i, \phi)) & |S_{\perp c}| &= \sum_{\substack{(s^i, l^i) \in S \\ l^i = c}} \varsigma(\rho(s^i, \neg\phi)) \end{aligned}$$

The Smooth Information Gain (IG) is defined as:

$$\begin{aligned} IG(S, \phi) &= H(S) - H(S|\phi) \\ H(S) &= - \sum_{c \in C} p_c \log p_c \\ H(S|\phi) &= - \sum_{c \in C} p_{\top c} \log p_{\top c} - \sum_{c \in C} p_{\perp c} \log p_{\perp c} \end{aligned} \quad (7.6)$$

The signals' sums in Eq. (7.5) are *weighted*. Each signal is counted according to its distance to the satisfaction boundary of the formula  $\phi$ , as determined by the sigmoid and the robustness degree. Intuitively, if a signal strongly satisfies (or strongly violates) the formula, it is wholly counted as going to the left child (or to the right child), like in Eq. (7.2). However, if a signal is close to the boundary, then its contribution is proportionally split among the two children (depending on how close and which side of the boundary it is located).

The following proposition ensures the smooth impurity measures are well defined.

**Proposition 7.1.** *The intra-partition weights are bounded within 0 and 1 and sum to 1, i.e.,  $0 \leq p_{\top}, p_{\perp} \leq 1$  and  $p_{\top} + p_{\perp} = 1$ , in both definitions Def. 7.1 and Def. 7.2. The same invariant property is true for the inter-partition weights, i.e.,  $0 \leq p_c, p_{\perp c}, p_{\top c} \leq 1$ ,*

$\forall c \in C$ , and  $\sum_{c \in C} p_c = 1$ ,  $\sum_{c \in C} p_{\perp c} = 1$ ,  $\sum_{c \in C} p_{\top c} = 1$ .

We will distinguish between the classical information gain and the smooth one using the superscript  $(s)$ , e.g.,  $IG^{(s)}$  for the smooth information gain.

### 7.3.2 Smooth Homogeneity Gain Measure for Clustering

**Definition 7.3** (Smooth Homogeneity Gain  $HG$ ). Let  $S$  be a finite set of unlabeled signals and  $\phi$  an STL formula, the Homogeneity Gain is defined as

$$HG(S, \phi) = I(S) - I(S|\phi) \quad (7.7)$$

$$I(S) = \frac{1}{2} \frac{1}{|S|^2} \sum_{s^i \in S} \sum_{s^j \in S} d^2(s^i, s^j) \quad (7.8)$$

$$\begin{aligned} I(S|\phi) = & p_{\top} \frac{1}{2} \frac{1}{|S_{\top}|^2} \sum_{s^i \in S} \sum_{s^j \in S} \varsigma(\rho(s^i, \phi)) \varsigma(\rho(s^j, \phi)) d^2(s^i, s^j) + \\ & + p_{\perp} \frac{1}{2} \frac{1}{|S_{\perp}|^2} \sum_{s^i \in S} \sum_{s^j \in S} \varsigma(\rho(s^i, \neg\phi)) \varsigma(\rho(s^j, \neg\phi)) d^2(s^i, s^j) \end{aligned} \quad (7.9)$$

where:

$$\begin{aligned} p_{\top} &= \frac{|S_{\top}|}{|S_{\top}| + |S_{\perp}|} & p_{\perp} &= \frac{|S_{\perp}|}{|S_{\top}| + |S_{\perp}|} \\ |S| &= |S_{\top}| + |S_{\perp}| & & (7.10) \\ |S_{\top}| &= \sum_{s^i \in S} \varsigma(\rho(s^i, \phi)) & |S_{\perp}| &= \sum_{s^i \in S} \varsigma(\rho(s^i, \neg\phi)) \end{aligned}$$

In Eq. (7.9), every pairwise distance  $d(s^i, s^j)$  is weighted according to each signal's distance to the satisfaction boundary of the formula  $\phi$ , as determined by the sigmoid and the robustness degree. Intuitively, if a signal strongly satisfies (or strongly violates) the formula, it contributes entirely towards the left child cluster (or the right child cluster). However, if a signal is close to the boundary, its contribution is proportionally split among the two clusters (depending on how close and which side of the boundary

it is located).

We will distinguish between the classical homogeneity gain and the smooth one using the superscript  $(s)$ , e.g.,  $HG^{(s)}$  for the smooth homogeneity measure.

### 7.3.3 Smooth primitive optimization procedure

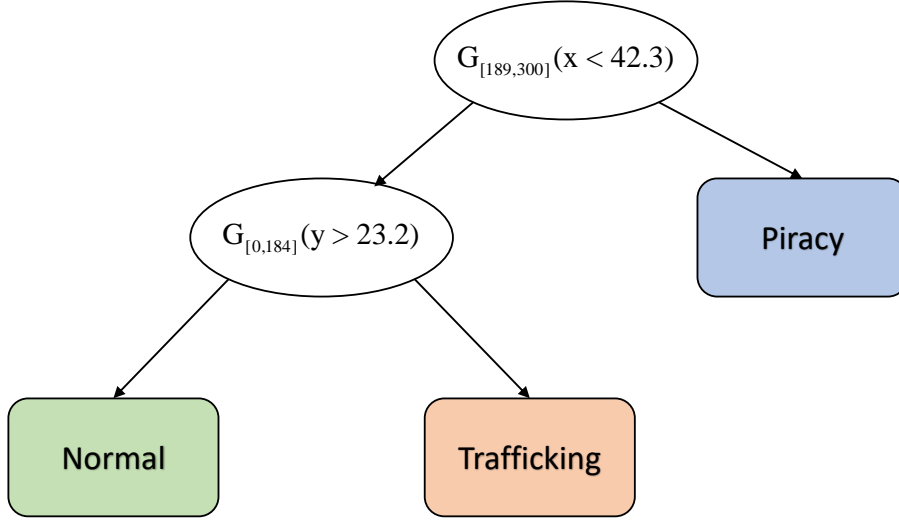
The canonical measures, defined in Def. 7.1 and Def. 6.2, are based on hard partitions of the signals and produce piecewise constant functions with respect to the primitive’s parameters. On the contrary, the smooth objective functions introduced in Def. 7.2 and Def. 7.3 change gradually as the parameters of a primitive change. This allows us to use optimization algorithms that exploit the gradient of the objective function to guide the exploration of the parameter space.

We chose to employ the Sequential Quadratic Programming (SQP) method. SQP represents the state of the art in nonlinear programming methods (Nocedal and Wright, 2006). SQP constructs a sequence of optimization problems, each of which optimizes a quadratic approximation of the objective function subject to linearized constraints. Since the optimization problem is non-convex, we combine SQP with a global Scatter Search (Ugray et al., 2007) to explore multiple basins of attraction.

**Remark 7.2.** The sigmoid function is used to produce a *soft* count of the signals based on their distance to the decision boundary. The robustness degree is used to quantify this distance. Since the robustness degree is sensitive to the scale of the signals, it is important to normalize the dataset before executing any learning algorithm.

## 7.4 Maritime Surveillance: Multi-Class and Smooth Optimization Results

We investigate Alg. 1 with the multi-class information gain of Def. 7.1 and the  $\mathcal{P}^1$  primitive set. Using the usual cross-validation scheme described in Sec. 8.1, we obtained a mean MCR of 0.25% (STD 0.78%) with a mean runtime of 45 seconds per cross-validation round.



**Figure 7.1:** Maritime Surveillance - A tree induced by the multi-class learning algorithm.

A tree induced in one of the cross-validation rounds is shown in Fig. 7.1. There are three leaves and each one of them is associated with one of the original classes (normal behavior, piracy, human trafficking). The corresponding formulae are:

$$\begin{aligned}
 \phi_{\text{Normal}} &= \mathbf{G}_{[189,300]}(x < 42.3) \wedge \mathbf{G}_{[0,184]}(y > 23.2) \\
 \phi_{\text{Piracy}} &= \mathbf{F}_{[189,300]}(x \geq 42.3) \\
 \phi_{\text{Trafficking}} &= \mathbf{G}_{[189,300]}(x < 42.3) \wedge \mathbf{F}_{[0,184]}(y \leq 23.2)
 \end{aligned} \tag{7.11}$$

Likewise, with the  $\mathcal{P}^2$  primitive set, we achieved a mean MCR of 0.20% (STD 0.61%). The mean runtime was 400 seconds per cross-validation round. These results indicate that in the multi-class case Alg. 1 is slightly more accurate and faster than the two-class case (refer to Sec. 4.11). Intuitively, this can be explained by recognizing that a multi-class dataset provides more information to the learning algorithm, which aids the partitioning process.

We obtain similar results, in terms of average MCR, using the smooth impurity measure in Def. 7.2 and the new optimization procedure described in Sec. 7.3.3. However, the execution time improves by 6x for both the  $\mathcal{P}^1$  primitives and the  $\mathcal{P}^2$

primitives.

## Chapter 8

### Case Studies

In this chapter, we present two case studies from the automotive field to illustrate the effectiveness of the proposed algorithms and to analyze their behavior. The automotive applications are particularly appealing because the systems involved are becoming more and more sophisticated. In a modern vehicle, several complex dynamical systems are interconnected and the methods present in literature may fail to cope with this complexity. In particular, anomaly detection methods derived from control theory lack proper handling of the hybrid components, and most classical machine learning methods lack a dynamical part.

We investigate two fundamental powertrain subsystems. The first is the *fuel combustion* subsystem, and the second is the *automatic transmission* subsystem. Good quality models of these systems are present as built-in examples in Simulink (The MathWorks Inc., 2017). They were developed using the experimental work of Crossley and Cook (1991).

We selected these models because they include all the complexities of real world industrial models, but are still quick to simulate, i.e., it is easy to obtain a large number of sample trajectories. These models were proposed as benchmarks for the Hybrid Systems community by Hoxha et al. (2014) and variations of the base schemes have been used as examples in recent formal methods papers (Fainekos et al., 2012; Yang et al., 2012; Zhao et al., 2003; Zuliani et al., 2013).

## 8.1 Implementation and Validation

We implemented and tested the algorithms described in the previous chapters using MATLAB.<sup>1</sup> To assess the performance of the supervised algorithms (Alg. 1 and Alg. 3, respectively), we use a five-fold cross-validation scheme. One round of cross-validation entails partitioning the whole dataset into two complementary subsets, performing the training on one subset, and the evaluation of the error on the other (testing). The results for each round are averaged to obtain a single estimate of the misclassification rate (MCR), its standard deviation (STD), and the average execution time. This procedure gives confidence on how well the formulae obtained will generalize to independent datasets and highlights problems such as overfitting (Ripley, 1996). We ran our experiments on a Windows PC, with an Intel 5930K CPU and 16 GB ram.

## 8.2 Fuel Control System

### 8.2.1 Model Description

We investigate a fuel control system for a gasoline engine. The key quantity in this system is the *air-to-fuel ratio*, that is, the ratio between the mass of air and the mass of fuel in the combustion process. The goal of the control system is to keep it close to the “ideal” stoichiometric value for the combustion process. For this system, the target air-fuel ratio is 14.6, as it provides a good compromise between power, fuel economy, and emissions. An overview of the Simulink model is shown in Fig. 8-1 (The MathWorks Inc., 2017).

The system has one main output, the air-to-fuel ratio, one control variable, the fuel rate, and two inputs, the engine speed and the throttle command. The system estimates the correct fuel rate to achieve the target stoichiometric ratio by taking into account four sensor readings. Two are related directly to the inputs: the engine speed and the throttle angle. The remaining two sensors provide crucial feedback

---

<sup>1</sup>The LoTuS Toolbox (Learning Temporal Specifications, formerly DTL2STL) is available at <http://sites.bu.edu/hyness/lotus/>

information: the EGO sensor reports the amount of residual oxygen present in the exhaust gas, and the MAP sensor reports the (intake) manifold absolute pressure. EGO is related to the air-to-fuel ratio, whereas MAP is related to the air mass rate.

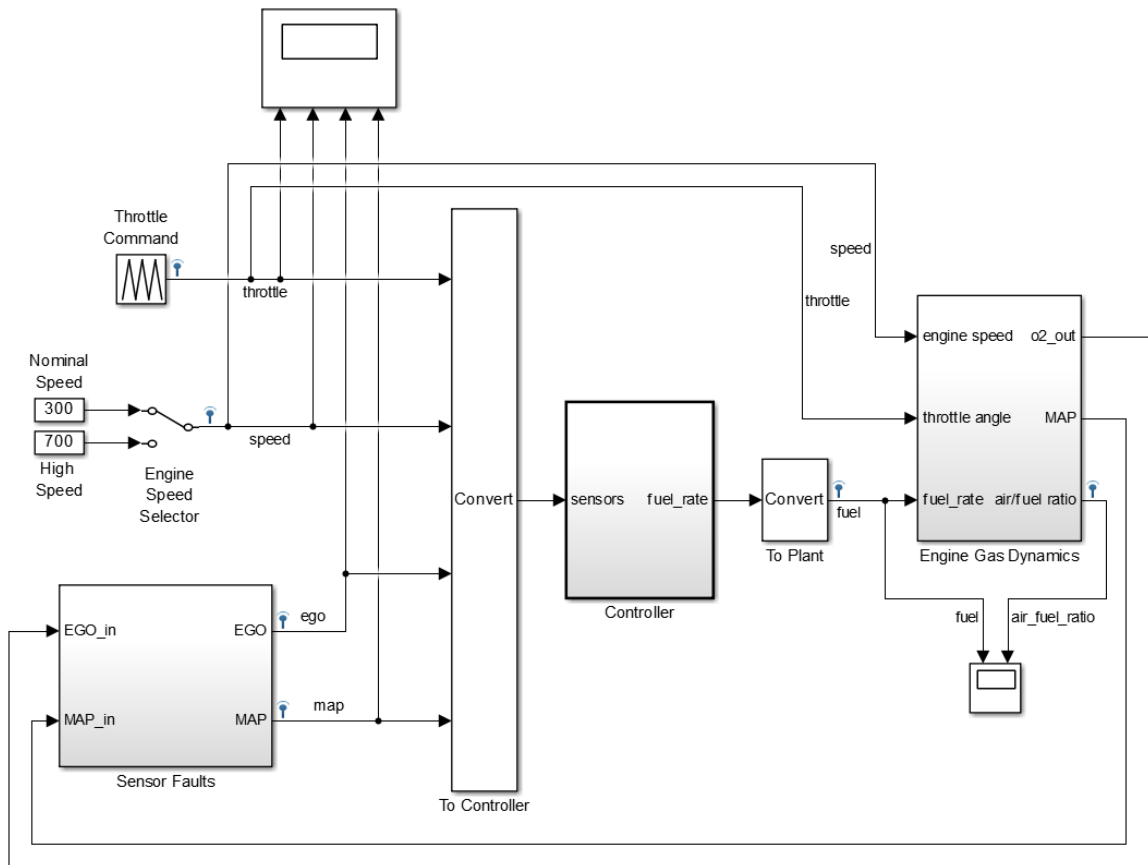
This system (Fig. 8.1) exhibits both discrete and continuous dynamics that are described with nonlinear and linear differential equations, a Stateflow control logic, and several lookup tables. Overall, this model provides a representative summary of the important features of hybrid systems.

### 8.2.2 Modifications and fault injection

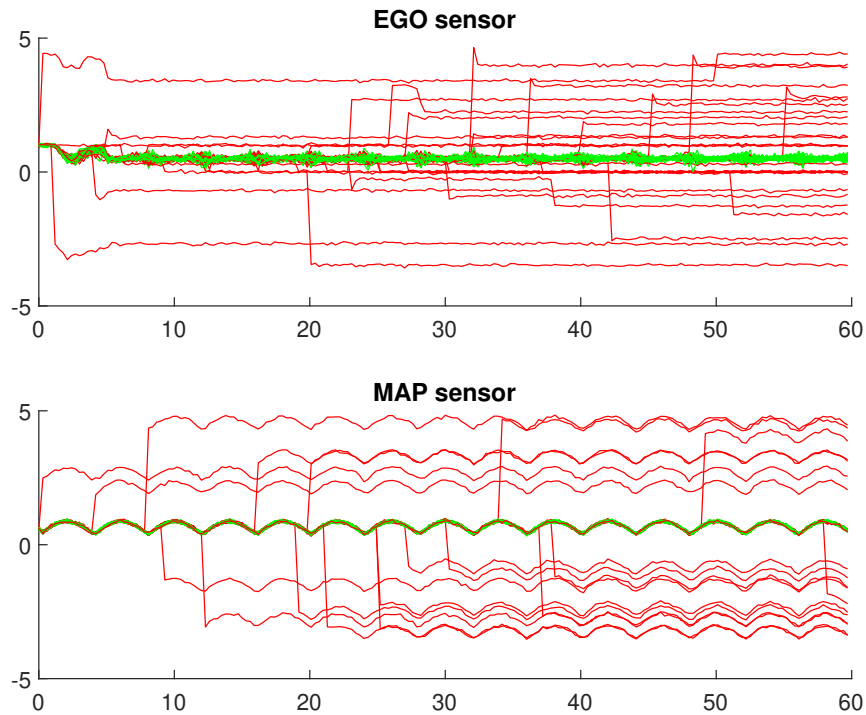
The base model includes a trivial fault detection scheme and fault injection mechanism. The fault detection scheme is a simple threshold crossing test, i.e., it is only able to detect single off range values, and is built within the Stateflow control logic. To avoid the overlap of two anomaly detection schemes, the built-in one has been removed. In the base model, the faults are injected by simply reporting an incorrect and fixed value for a sensor's reading. These faults are always present from the beginning of the simulation. We substituted this fault injection mechanism with a more sophisticated unit. The new subsystem can induce faults in both the EGO and MAP sensors with a *random* arrival time and with a *random* additive offset value. Specifically, the faults can manifest at anytime during the execution (uniformly at random), and the readings of the the sensors affected are offset by a value that *varies* at every execution. Finally, independent Gaussian noise signals, with zero mean and variance  $\sigma^2 = 0.01$ , have been added at the output of the sensors.

### 8.2.3 Dataset generation

For the fuel control system, 1200 total simulations were performed. In all cases, the throttle command provides a periodic triangular input, and the engine speed is kept constant at 300 rad/sec (2865 RPM). The simulation time is 60 seconds. In details, we obtained:



**Figure 8.1:** Fuel Control - Overview of the model (modified from (The MathWorks Inc., 2017))



**Figure 8.2:** Fuel Control - Example Trajectories: normal in green, anomalous in red

- 600 traces where the system was working normally;
- 200 traces with a fault in the EGO sensor;
- 200 traces with a fault in the MAP sensor;
- 200 traces with faults in both sensors.

For every trace, we collected 200 samples of the EGO and MAP sensors' readings. Fig. 8.2 shows some sample traces.

## 8.2.4 Results

### Two-Class Classification: Meta-Parameters Analysis

Table 8.1 shows how Alg. 1 behaves, in terms of the accuracy achieved on training data, as a function of the chosen *meta-parameters* (i.e., impurity measure and primitive set) for various depths ( $D$ ) of the induced tree.

$(J, \mathcal{P})$	MCR ( $D = 1$ )	MCR ( $D = 3$ )	MCR ( $D = 5$ )
$(MG, \mathcal{P}^1)$	23.31% (0.41%)	1.06% (0.32%)	0.35% (0.16%)
$(MG, \mathcal{P}^2)$	23.29% (0.59%)	1.02% (0.17%)	0.60% (0.23%)
$(IG, \mathcal{P}^1)$	23.50% (0.41%)	0.96% (0.39%)	0.31% (0.31%)
$(IG, \mathcal{P}^2)$	23.33% (0.56%)	1.02% (0.31%)	0.46% (0.19%)

**Table 8.1:** Fuel Control System - Analysis of classification accuracy as a function of Impurity Measure ( $J$ ), Primitive Set ( $\mathcal{P}$ ) and Maximum Tree Depth ( $D$ ).

### Extended Impurity Measures Results

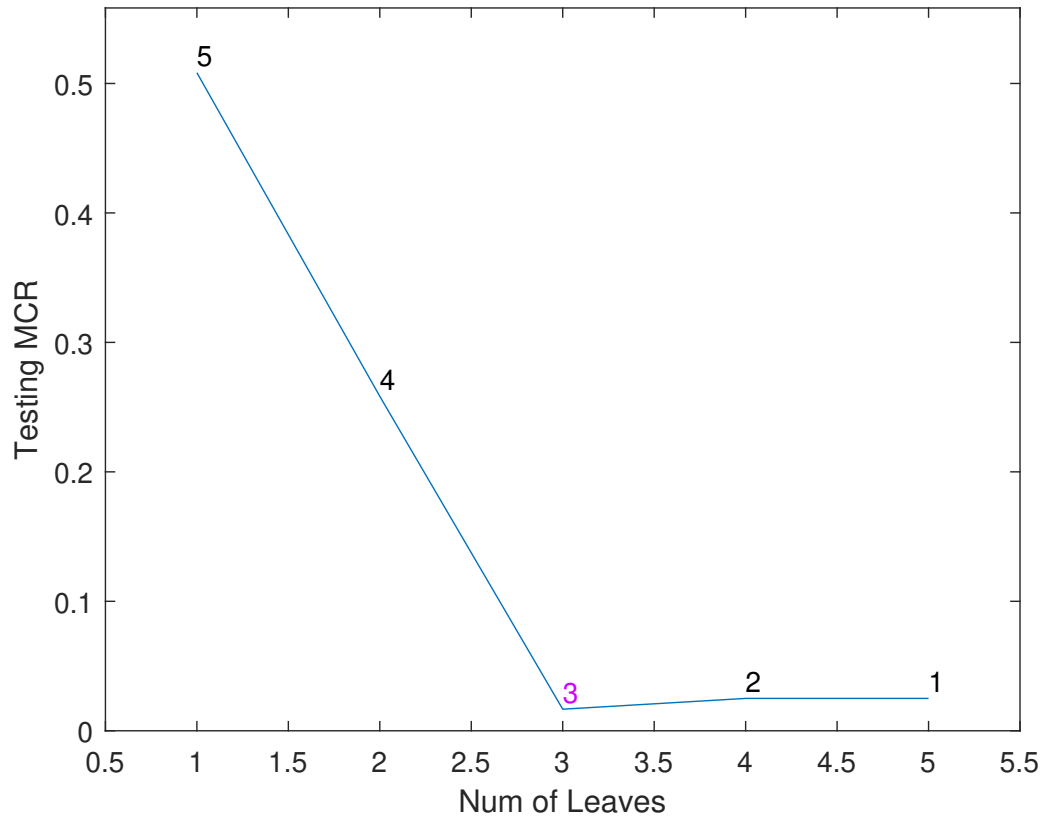
For the set of primitives  $\mathcal{P}^2$  and impurity measure  $IG^{(r)}$ , we obtained a mean misclassification rate of 5.4% with a standard deviation of 2.5% and a run time of about 9 minutes per split. A sample formula, obtained from one of the cross-validation splits, is ( $x_1$  and  $x_2$  correspond to the EGO and MAP sensors, respectively):

$$\begin{aligned}
\phi &= \neg\phi_1 \wedge \phi_2 \wedge \phi_3 \\
\phi_1 &= \mathbf{F}_{[1.85,58.70]} \mathbf{G}_{[0.00,0.57]}(x_1 \leq 0.13) \\
\phi_2 &= \mathbf{G}_{[11.35,59.55]} \mathbf{F}_{[0.00,0.03]}(x_1 \leq 0.99) \\
\phi_3 &= \mathbf{G}_{[1.65,58.89]} \mathbf{F}_{[0.00,0.44]}(x_2 \leq 0.90)
\end{aligned} \tag{8.1}$$

Notice in this case how the resulting sub-formulae are equivalent to first-level primitives, suggesting that  $\mathcal{P}^2$  is an overly complicated set of primitives.

When using the set of primitives  $\mathcal{P}^1$  with impurity measure  $MG^{(r)}$ , we achieved a mean misclassification rate of 3.5% and a standard deviation of 1.76%. The run time is about 1.5 minutes per split. A sample formula learned in one of the splits is:

$$\begin{aligned}
\phi &= (\phi_1 \wedge (\phi_2 \wedge (\phi_3 \wedge \phi_4))) \vee (\neg\phi_1 \wedge (\phi_5 \wedge (\phi_6 \wedge \phi_7))) \\
\phi_1 &= \mathbf{F}_{[22,58.4]}(x_2 > 0.932) \quad \phi_2 = \mathbf{G}_{[29.3,59.6]}(x_2 < 0.994) \\
\phi_3 &= \mathbf{G}_{[56,58.3]}(x_1 > 0.0979) \quad \phi_4 = \mathbf{G}_{[49.9,55.3]}(x_1 < 0.863) \\
\phi_5 &= \mathbf{G}_{[39.5,58.4]}(x_2 > 0.193) \quad \phi_6 = \mathbf{F}_{[59.4,59.7]}(x_1 > 0.25) \\
\phi_7 &= \mathbf{G}_{[2.52,53.3]}(x_1 < 1.05)
\end{aligned} \tag{8.2}$$



**Figure 8·3:** Fuel Control - Pruning Analysis - Test MCR as a function of the number of leaves in the tree sequence produced by the cost-complexity pruning algorithm.

### Offline Post-Completion Pruning

Following the discussion in Sec. 4.8, it is often advisable to induce a deeper tree from the training data and then use independent data to determine the right size.

For the automotive case study, if we use the impurity function  $IG$  with primitive set  $\mathcal{P}^1$  and terminate the algorithm only when a maximum depth of 5 has been reached,

we obtain the following formula ( $x_1$  and  $x_2$  stand for EGO and MAP, respectively):

$$\phi = \phi_1 \wedge (\phi_2 \wedge ((\phi_3 \wedge \phi_4) \vee \phi_5)) \quad (8.3)$$

$$\phi_1 = \mathbf{G}_{[47.7,59.7]}(x_1 > .285) \quad \phi_2 = \mathbf{G}_{[8.45,59.5]}(x_1 < .937)$$

$$\phi_3 = \mathbf{G}_{[46.8,59.7]}(x_2 > .392) \quad \phi_4 = \mathbf{F}_{[37,39.9]}(x_1 < .358)$$

$$\phi_5 = \mathbf{F}_{[46.8,59.7]}(x_1 < .392) \quad (8.4)$$

with an associated MCR of 0.42% on the training data and 2.50% on the testing data (execution time 30 secs). Fig. 8-3 shows the testing error for the sequence of trees constructed by the post-completion pruning procedure. By selecting the third tree in the sequence, we simplify it to:

$$\phi = \mathbf{G}_{[47.7,59.7]}(x_1 > .285) \wedge \mathbf{G}_{[8.45,59.5]}(x_1 < .937)$$

with an associated MCR of 2.50% on the training data and 1.67% on the testing data. This formula establishes tight thresholds for the EGO sensor values. Therefore, we not only obtained a more interpretable formula but also one that is performing better on unseen data.

### Online Learning Results

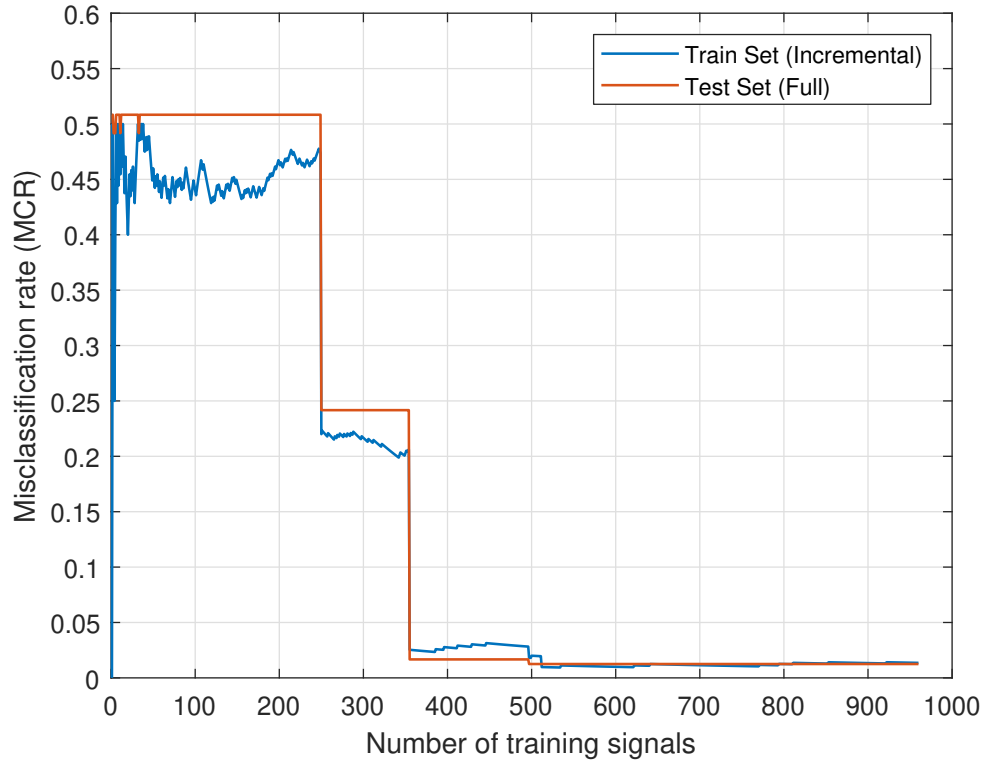
We used the misclassification gain ( $MG$ ) as impurity measure and the  $\mathcal{P}^1$  as primitive set. We obtained a mean misclassification rate of 2.67% with a standard deviation of 0.86%. The mean runtime was 90 seconds per round.

A sample formula obtained at the end of one of the cross-validation round is reported:

$$\phi = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \quad (8.5)$$

$$\phi_1 = \mathbf{G}_{[6.62,59.4]}(x_1 > 0.060) \quad \phi_2 = \mathbf{G}_{[8.59,59.7]}(x_1 < 0.921)$$

$$\phi_3 = \mathbf{F}_{[55.4,59.5]}(x_1 > 0.556) \quad \phi_4 = \mathbf{F}_{[39.2,50.2]}(x_2 < 0.412)$$



**Figure 8-4:** Fuel Control - MCR as function of the number of signal processed by the online algorithm. In blue, evolution of MCR computed on training signals (as seen by the algorithm). In red, evolution of MCR computed on an independent test set.

with an associated MCR of 1.25%. Fig. 8-4 shows the evolution of training and test error on this cross-validation round. By stopping the learning process after 354 signals, we get the simpler formula:

$$\phi = \mathbf{G}_{[6.62,59.4]}(x_1 > 0.060) \wedge \mathbf{G}_{[8.59,59.7]}(x_1 < 0.921) \quad (8.6)$$

with an associated MCR of 1.67%.

## 8.3 Automatic Transmission

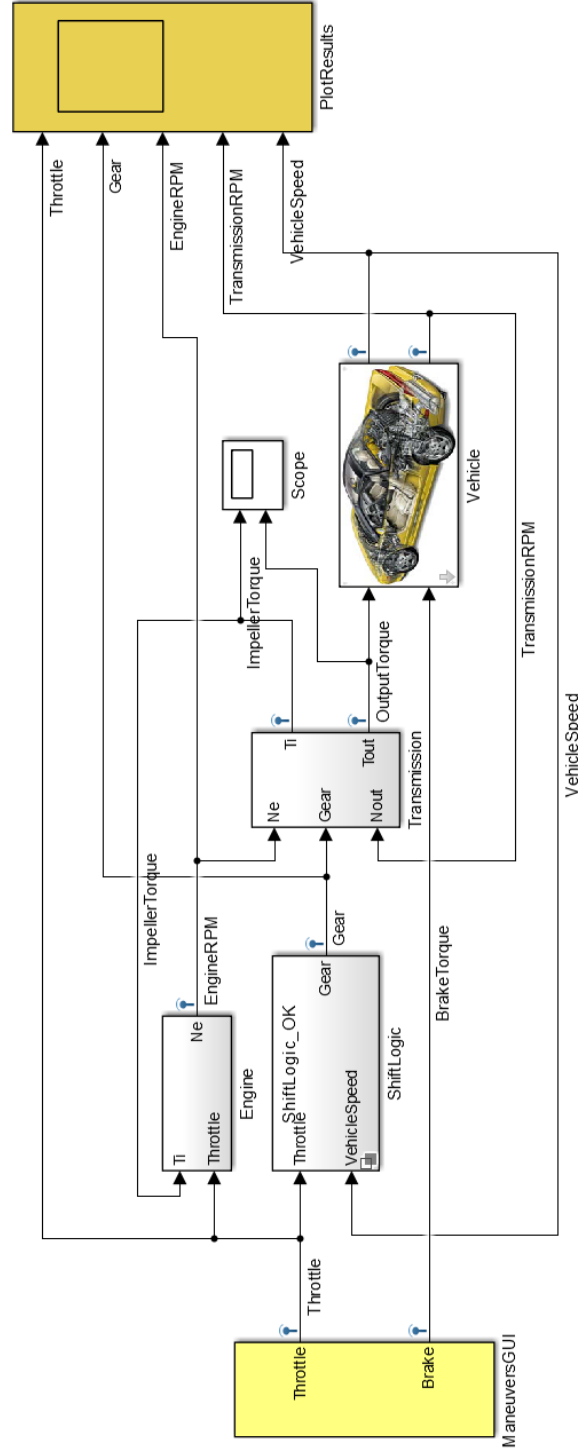
### 8.3.1 Model Description

As second case study, we consider a closed-loop vehicle model with four speeds and an automatic transmission controller. The transmission system allows the internal combustion engine to run at appropriate rotational speed 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 as the vehicle moves, freeing the driver from having to shift gears manually. Although this Simulink model is not an industrial-class model, it has all necessary components to represent the real system (The MathWorks Inc., 2017). The model (see Figure 8-5 for an overview) is composed by four main subsystems: the engine, the transmission, the shift logic, and the vehicle dynamics.

The system has two inputs: throttle opening and brake. The throttle, at each point in time, can take any value between 0 (fully closed) and 100 (fully open). The brake can also take values between 0 and 100 and is used to model the variable load on the engine, such as the vehicle going uphill or downhill.

The system has two continuous state variables and one discrete. 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.

Overall, this model contains different kinds of blocks, both continuous and discrete, among which there are several look-up tables. The gear selection logic is implemented with a Stateflow chart using two concurrently executing Finite State Machines (FSMs). Because of these characteristics, this system exhibits a rich and diverse number of output trajectories and make this model an interesting candidate for our investigation.



**Figure 8.5:** Automatic Transmission - Overview of the model (modified from (The MathWorks Inc., 2017))

### 8.3.2 Modifications and fault injection

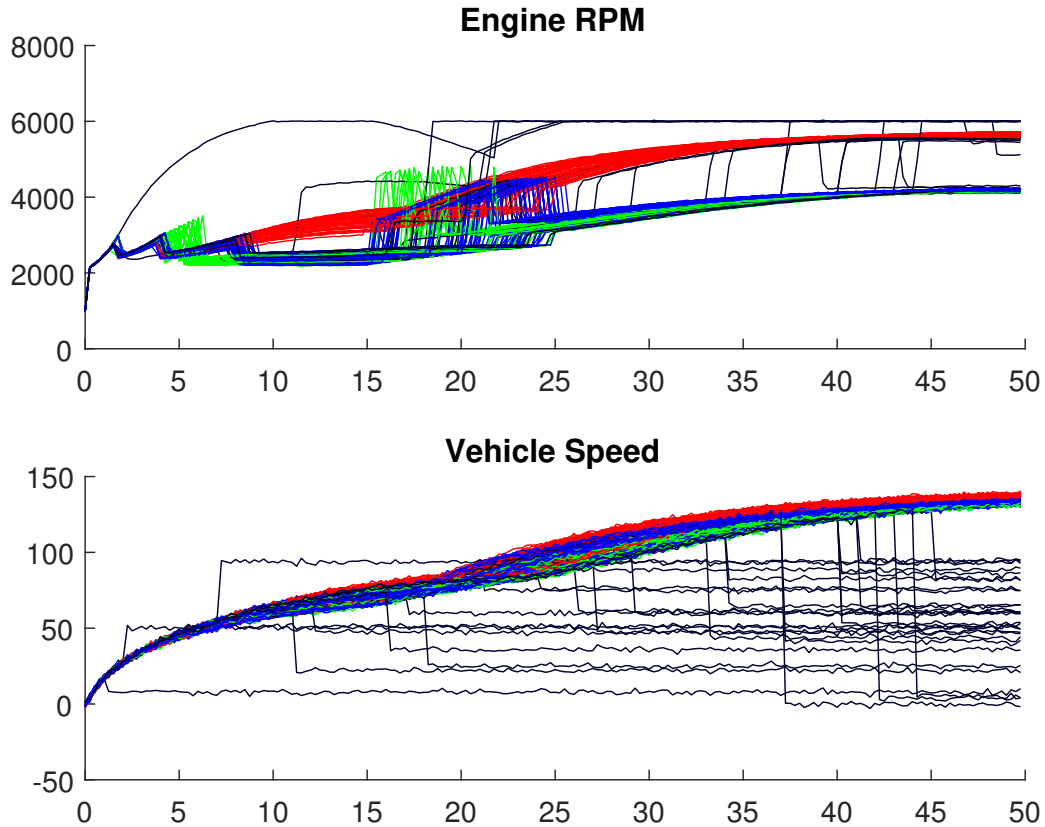
For this case study, we consider the vehicle starting from rest and subsequently performing a surpass maneuver. Specifically, the vehicle starts with zero speed, engine at 1000 RPM, and with throttle at 50%. The surpass maneuver begins at a random time, between 15 and 25 seconds, when the driver steps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

### 8.3.3 Dataset generation

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).



**Figure 8-6:** Automatic Transmission - Example Trajectories: normal in blue, anomalous in black (Type 1-sensor fault), red (Type 2-no 4th gear), and green (Type 3-skip 3rd gear).

For every trace, we collected the system’s output values, that is, the speed of the engine and the speed of the vehicle. Some sample traces are shown in Fig. 8-6.

### 8.3.4 Results

#### Clustering Results

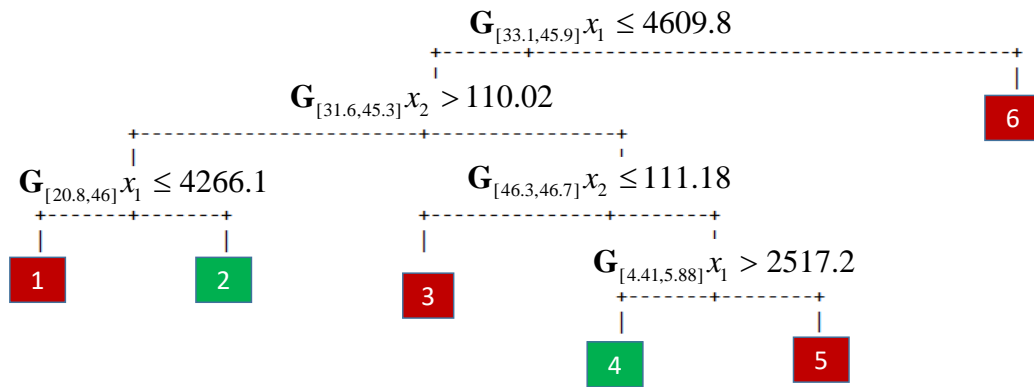
For this automotive case study, we executed Alg. 5 with the aim of reconstructing the different types of faults and the normal conditions present in the dataset. We obtained the best results using the *dependent* multi-dimensional dynamic time-warping distance ( $DTW_D$ ) with 0.2 warping factor. This 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. Alg. 5 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. 8·7b). This corresponds to the tree shown in Fig. 8·7a.

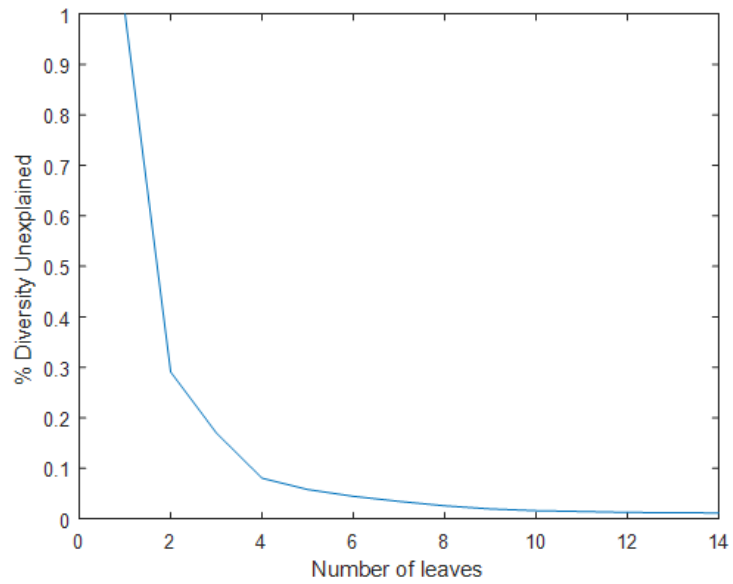
Type 1 and Type 2 faults are mapped with a cluster each. Type 3 faults are mapped with two clusters, and there are also two separate clusters containing normal signals. This happens because, when the surpassing maneuver starts, two scenarios are possible. In the first scenario, the transmission downshifts 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 fourth gear (Fig. 8·8 - 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 fourth gear for the whole time (Fig. 8·8 - 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):

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

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 3.1%.

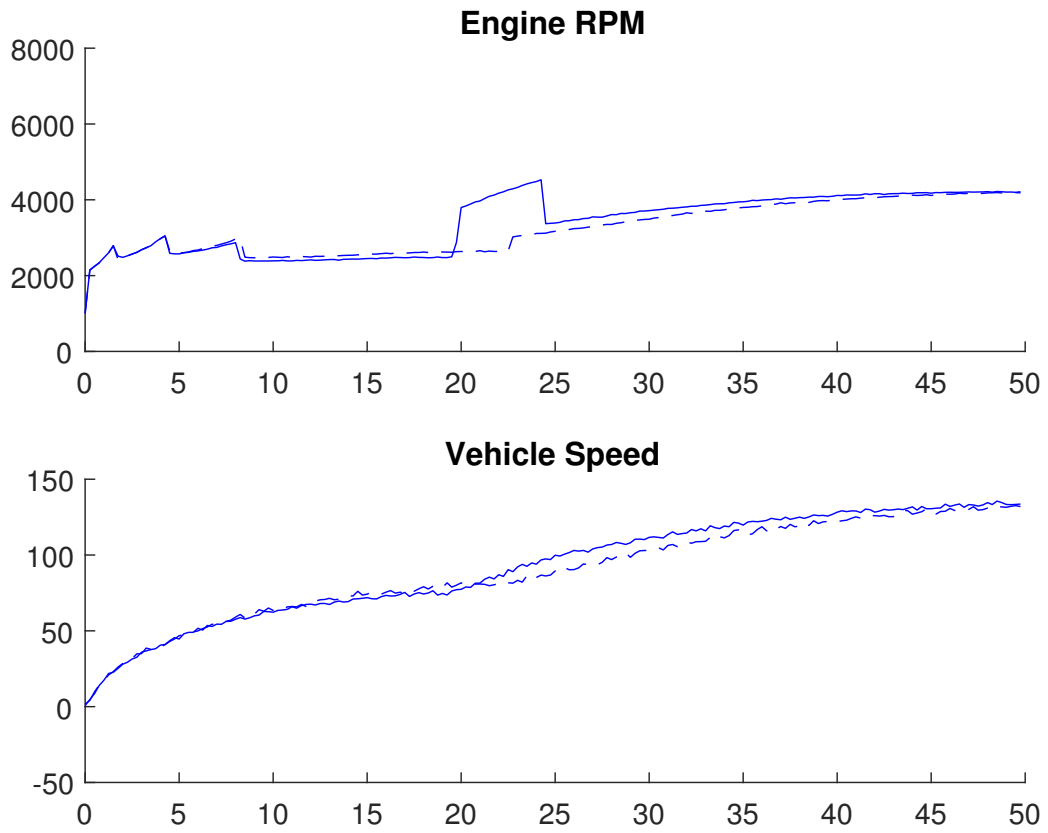


(a) Cluster Tree - Leaves 2 and 4 contain normal signals



(b) Elbow Analysis

**Figure 8·7:** Automatic Transmission - Clustering Results



**Figure 8.8:** Automatic Transmission - 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.

## Chapter 9

# Conclusion

### 9.1 Comparisons and Discussion

The maritime surveillance case study was also investigated in (Kong et al., 2017), with their offline SVM-based TLI approach. Unfortunately, it is not easy to make a formal comparison between the formulae learned by our approach and the ones in (Kong et al., 2017). This is due to the fact that iPSTL, defined in (Kong et al., 2017), and  $STL_{\mathcal{P}}$  do not represent the same STL fragment. Nevertheless, it is always possible to make a judgement in terms of sheer classification performance and execution time. In comparison, we improve the misclassification rate by a factor of 20 while being around 100 times faster.

The performance advantage of our proposed algorithms is due to the “divide and conquer” nature of growing STL formulae represented as trees. For the offline algorithms, the problem of finding optimal primitives becomes progressively easier as the tree is constructed. This follows from the fact that a node’s optimization problem has always a fixed number of parameters and the data is partitioned between the two children of the node. This is particularly important because our practical experience with parameter mining problems and the results reported by Jha et al. (2017) suggest that this problem does not scale well with the number of parameters. For the online algorithm, a new node is created only when some conditions on the overall best primitive to pick are attained. This strategy avoids any pruning and can handle large datasets. Another major speed-up was achieved with the introduction of differentiable versions of the objective functions in the local node optimization

problem.

As opposed to all the other works present in literature, this thesis investigates the trade-off between formulae complexity and their effectiveness in solving the task at hand. For the offline supervised case, we employed a post-completion pruning procedure to avoid over-fitting and produce more interpretable formulae. In the online case, we argued that the evolution of the misclassification rate (as signals are processed by the algorithm) provides information that can be exploited to interrupt the data collection process if a satisfactory solution has already been found. In the clustering case, the formulae complexity is directly tied to the number of clusters. The right number of clusters is not known beforehand but can be deduced using visual tools like the elbow method.

## 9.2 Summary of the thesis

We presented an inference framework of timed temporal logic properties from time-series data. This work is in line with a recent interest in Temporal Logic Inference (TLI) and is motivated by the need to construct classifiers which provide good performance and can be interpreted over specific application domains.

The framework defines customizable algorithms that output Signal Temporal Logic (STL) formulae. Three induction algorithms are presented to solve the following problems:

1. offline supervised learning (two class and multi-class);
2. online supervised learning (signals incrementally available);
3. unsupervised clustering (unlabeled signals).

These algorithms may be customized by providing: (a) a set of primitive properties of interest; (b) an impurity, or inhomogeneity, measure which captures the signals' class distribution, or signals' similarity, within a node, respectively.

Another contribution of this thesis is the introduction of new impurity and inhomogeneity measures that take into account the robustness degrees of signals. These new measures are differentiable with respect to the formula's parameters and enable us to use a smooth optimization solver, greatly improving the execution time of all algorithms.

The proposed framework has been tested on three case studies in the maritime surveillance and automotive fields. We showed that it is able to capture relevant characteristics of the signals in all cases and achieves solid classification accuracy. We also provide visual tools to explore the trade-off between formula complexity and its effectiveness.

The algorithms and tools described in this thesis have been collected in a software toolbox named LoTuS (Learning Temporal Specifications), which has been made available online.<sup>1</sup>

---

<sup>1</sup>LoTuS Toolbox - <http://sites.bu.edu/hyness/lotus/>

# Bibliography

- Aggarwal, C. C. and Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition.
- Asarin, E., Donzé, A., Maler, O., and Nickovic, D. (2012). Parametric identification of temporal properties. In *Runtime Verification*, pages 147–160. Springer.
- Bartocci, E., Bortolussi, L., Nenzi, L., and Sanguinetti, G. (2015). System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25.
- Bartocci, E., Bortolussi, L., and Sanguinetti, G. (2014). Data-driven statistical learning of temporal logic properties. In *Formal Modeling and Analysis of Timed Systems*, pages 23–37. Springer.
- Belta, C. and Sadraddini, S. (2019). Formal Methods for Control Synthesis: An Optimization Perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):115–140.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag New York, Inc.
- Bombara, G., Vasile, C.-I., Penedo, F., Yasuoka, H., and Belta, C. (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, HSCC '16*, pages 1–10, New York, NY, USA. ACM.
- Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. CRC press.
- Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., and Bortolussi, L. (2014). Temporal Logic Based Monitoring of Assisted Ventilation in Intensive Care Patients. In *Leveraging Applications of Formal Methods, Verification and Validation*, number 8803 in Lecture Notes in Computer Science, pages 391–403. Springer.
- Chen, G., Sabato, Z., and Kong, Z. (2016). Active learning based requirement mining for cyber-physical systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4586–4593.
- Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

- Cormen, T. H. (2009). *Introduction to Algorithms*. MIT Press, third edition.
- Crossley, P. and Cook, J. (1991). A nonlinear engine model for drivetrain system development. In *International Conference on Control 1991*, pages 921–925 vol.2.
- Domingos, P. and Hulten, G. (2000). Mining High-speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA. ACM.
- Donzé, A., Ferrere, T., and Maler, O. (2013). Efficient robust monitoring for STL. In *Computer Aided Verification*, pages 264–279. Springer.
- Donzé, A. and Maler, O. (2010). Robust Satisfaction of Temporal Logic over Real-Valued Signals. In Chatterjee, K. and Henzinger, T. A., editors, *Formal Modeling and Analysis of Timed Systems*, number 6246 in Lecture Notes in Computer Science, pages 92–106. Springer Berlin Heidelberg.
- Fainekos, G., Sankaranarayanan, S., Ueda, K., and Yazarel, H. (2012). Verification of automotive control applications using S-TaLiRo. In *American Control Conference (ACC), 2012*, pages 3567–3572.
- Fainekos, G. E. and Pappas, G. J. (2009). Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291.
- Färber, I., Günnemann, S., Kriegel, H.-P., Kröger, P., Müller, E., Schubert, E., Seidl, T., and Zimek, A. (2010). 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*.
- Gol, E. A., Bartocci, E., and Belta, C. (2014). A formal methods approach to pattern synthesis in reaction diffusion systems. In *2014 IEEE 53rd Annual Conference on Decision and Control (CDC)*, pages 108–113. IEEE.
- Grosu, R., Smolka, S. A., Corradini, F., Wasilewska, A., Entcheva, E., and Bartocci, E. (2009). Learning and detecting emergent behavior in networks of cardiac myocytes. *Communications of the ACM*, 52(3):97–105.
- Hastie, T., Tibshirani, R., and Friedman, J. (2016). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, 2nd edition.
- Hoxha, B., Abbas, H., and Fainekos, G. (2014). Benchmarks for temporal logic requirements for automotive systems. *Proc. of Applied Verification for Continuous and Hybrid Systems*.

- Hoxha, B., Dokhanchi, A., and Fainekos, G. (2017). Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, pages 1–15.
- Hyafil, L. and Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17.
- Ingber, L. (1996). Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25:33–54.
- Isermann, R. (2006). *Fault-Diagnosis Systems*. Springer.
- Jha, S., Tiwari, A., Seshia, S. A., Sahai, T., and Shankar, N. (2017). TeLEx: Passive STL Learning Using Only Positive Examples. In *Runtime Verification*, Lecture Notes in Computer Science, pages 208–224. Springer, Cham.
- Jin, R. and Agrawal, G. (2003). Efficient Decision Tree Construction on Streaming Data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 571–576, New York, NY, USA. ACM.
- Jin, X., Donzé, A., Deshmukh, J., and Seshia, S. A. (2015). Mining Requirements from Closed-Loop Control Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99):1–1.
- Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Mass.
- Kong, Z., Jones, A., and Belta, C. (2017). Temporal Logics for Learning and Detection of Anomalous Behavior. *IEEE Transactions on Automatic Control*, 62(3):1210–1222.
- Kowalska, K. and Peel, L. (2012). Maritime anomaly detection using Gaussian Process active learning. In *2012 15th International Conference on Information Fusion (FUSION)*, pages 1164–1171.
- Li, X., Ma, Y., and Belta, C. (2018). A Policy Search Method For Temporal Logic Specified Reinforcement Learning Tasks. In *2018 Annual American Control Conference (ACC)*, pages 240–245.
- Liu, C., Arnon, T., Lazarus, C., Barrett, C., and Kochenderfer, M. J. (2019). Algorithms for Verifying Deep Neural Networks. *arXiv:1903.06758 [cs, stat]*.
- Maler, O. and Nickovic, D. (2004). Monitoring Temporal Properties of Continuous Signals. In Lakhnech, Y. and Yovine, S., editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, number 3253 in Lecture Notes in Computer Science, pages 152–166. Springer Berlin Heidelberg.

- Neider, D. and Gavran, I. (2018). Learning Linear Temporal Properties. <https://arxiv.org/abs/1806.03953>.
- Nenzi, L., Silveti, S., Bartocci, E., and Bortolussi, L. (2018). A Robust Genetic Algorithm for Learning Temporal Specifications from Data. In McIver, A. and Horvath, A., editors, *Quantitative Evaluation of Systems*, Lecture Notes in Computer Science, pages 323–338. Springer International Publishing.
- Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, 2 edition.
- Pant, Y. V., Abbas, H., and Mangharam, R. (2017). Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1235–1240.
- Quinlan, J. R. (2014). *C4.5: Programs for Machine Learning*. Elsevier.
- Ratanamahatana, C. A. and Keogh, E. (2004). Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer.
- Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge university press.
- Rutkowski, L., Jaworski, M., Pietruczuk, L., and Duda, P. (2015). A New Method for Data Stream Mining Based on the Misclassification Error. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1048–1059.
- Shah, A., Kamath, P., Shah, J. A., and Li, S. (2018). Bayesian Inference of Temporal Task Specifications from Demonstrations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 3807–3816. Curran Associates, Inc.
- Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73.
- Shokoohi-Yekta, M., Wang, J., and Keogh, E. (2015). 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, pages 289–297. Society for Industrial and Applied Mathematics.
- Storn, R. and Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359.

- The MathWorks Inc. (2017). *MATLAB and Simulink R2017a*. Natick, Massachusetts, USA.
- Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., and Martí, R. (2007). Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *INFORMS Journal on Computing*, 19(3):328–340.
- Utgoff, P. E., Berkman, N. C., and Clouse, J. A. (1997). Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning*, 29(1):5–44.
- Yang, H., Hoxha, B., and Fainekos, G. (2012). Querying Parametric Temporal Logic Properties on Embedded Systems. In *Testing Software and Systems*, number 7641 in Lecture Notes in Computer Science, pages 136–151. Springer.
- Zhao, Q., Krogh, B. H., and Hubbard, P. (2003). Generating test inputs for embedded control systems. *IEEE Control Systems*, 23(4):49–57.
- Zuliani, P., Platzer, A., and Clarke, E. M. (2013). Bayesian statistical model checking with application to Stateflow/Simulink verification. *Formal Methods in System Design*, 43(2):338–367.

## Curriculum Vitae

