Information and Computation 301 (2024) 105209

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Neural-symbolic temporal decision trees for multivariate time series classification

Giovanni Pagliarini [a,b], Simone Scaboro [c], Giuseppe Serra [c], Guido Sciavicco [a], Ionel Eduard Stan [d,*]

[a] Department of Mathematics and Computer Science, University of Ferrara, Italy
[b] Department of Mathematical, Physical, and Computer Sciences, University of Parma, Italy
[c] Department of Mathematics, Computer Science and Physics, University of Udine, Italy
[d] Faculty of Engineering, Free University of Bozen-Bolzano, Italy

ARTICLE INFO

Article history:
Received 6 January 2024
Received in revised form 24 July 2024
Accepted 28 July 2024
Available online 2 August 2024

Keywords:
Time in artificial intelligence
Temporal logic and reasoning
Machine learning
Hybrid temporal decision trees

ABSTRACT

Multivariate time series classification is an ubiquitous and widely studied problem. Due to their strong generalization capability, neural networks are suitable for this problem, but their intrinsic black-box nature often limits their applicability. Temporal decision trees are a relevant alternative to neural networks for the same task regarding classification performances while attaining higher levels of transparency and interpretability. In this work, we approach the problem of hybridizing these two techniques, and present three independent, natural hybridization solutions to study if, and in what measure, both the ability of neural networks to capture complex temporal patterns and the transparency and flexibility of temporal decision trees can be leveraged. To this end, we provide initial experimental results for several tasks in a binary classification setting, showing that our proposed neural-symbolic hybridization schemata may be a step towards accurate and interpretable models.

1. Introduction

1.1. Overview

A *multivariate time series* is a collection of time-stamped tuples, each composed by the value of several attributes. The *classification* of (multivariate) time series is an active area of research across many scientific disciplines, such as air quality control and prediction in climate science, prices and rates of inflation analysis in economics, infectious diseases trends and spreading patterns modeling in medicine, pronunciation of word signs interpretation in linguistics, sensor-based predictive maintenance in engineering, among many others [1].

As it is valid for any other classification problem, symbolic and sub-symbolic techniques, two fundamental pillars of artificial intelligence, can address multivariate time series classification. *Symbolic learning* explicitly represents input data as a logical knowledge base using discrete symbols and performs rule-based logical reasoning. On the other hand, *sub-symbolic learning* learns patterns directly from continuous data and reasons implicitly using such patterns. Neural network-based models, a representative class of the sub-symbolic paradigm, excel at learning from unstructured (or non-tabular) data,

* Corresponding author.
E-mail address: ioneleduard.stan@unibz.it (I.E. Stan).

https://doi.org/10.1016/j.ic.2024.105209

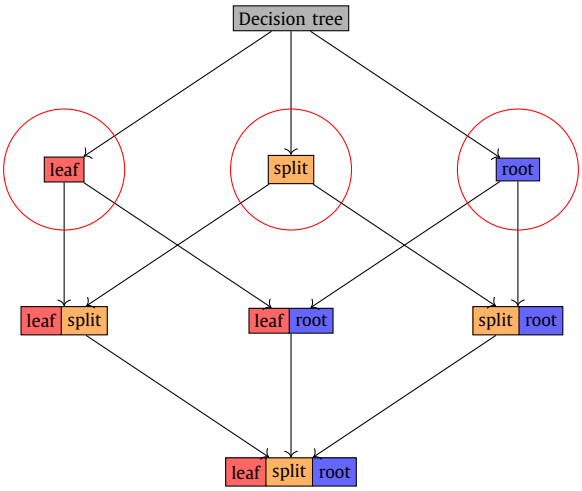


Fig. 1. Different types of temporal decision tree hybridization, partially ordered by the residual level of interpretability. This work focuses on three hybrids: leaf level, split level, and leaf level.

such as time series, images, audio, text, and graphs, among many others. The success of such models boils down to the exploitation of inductive biases in the data, such as spatial correlations in convolutional neural networks. However, neural network models could sometimes be more effective on structured (or tabular) data. Conversely, decision tree-based models, symbolic methods by nature, and their variants (e.g., random forests and gradient-boosted trees) are (still) dominant in the latter case [2,3].

Since the early days of artificial intelligence, experts have perpetuated a long-standing debate on which approach to prefer. Its roots lie in the fact that sub-symbolic learning methods tend to be more accurate than symbolic ones, which can better generalize the task at hand, despite requiring large amounts of data; in contrast, symbolic approaches can extract interpretable and explainable models (even from relatively small datasets), which are integrable with human expert knowledge. Interpretability is as crucial as accuracy, where the decisions made by artificial intelligence models directly impact human lives [4]. Indeed, the recent GDPR (General Data Protection Regulation) of the European Union highlights the need for interpretable/explainable automatic decision processes. As suggested in [5–7] (among others), one could explore hybrid approaches to understand if the strengths of both learning paradigms can be harnessed towards obtaining models with high levels of interpretability and good statistical performances.

A recent class of symbolic approaches for (multivariate) time series classification is *temporal decision trees (TDTs)*, initially proposed in [8]. Conventional decision trees (DTs) suffer in natively learning from input time series as they cannot deal with the temporal dimension. However, by trading some of the informative content of the original time series, DTs can learn from a *feature representation* of the time series, that is, compressing the time-dependent information into a more familiar tabular form (e.g., applying/finding the maximum value over each variable of time series before the learning phase). Instead, TDTs extend DTs in two directions. First, TDTs consider time series in their original format, assuming to make decisions on time series intervals. Second, TDTs perform reasoning in temporal terms by substituting DTs' propositional logic with more expressive yet still propositional interval temporal logic. TDTs harness traditional DT learning algorithms (e.g., CART [9]) with minor changes due to the resulting models' extended expressivity in logic terms. Application-wise, TDTs show promising results for time series classification in a variety of fields (e.g., see [8,10–13]). On the sub-symbolic side, recurrent neural networks (RNNs) are well-known models to learn from temporal data, including time series. RNNs can maintain information about the past to predict future inputs. Modern neural networks use gradient descent-based optimization algorithms during training. RNNs are no exception, but they suffer from the vanishing gradient problem: gradients become smaller and smaller as they propagate through the network, eventually disappearing. Gated recurrent units (GRUs) [14], specific types of RNNs, are a popular choice to address such an issue: a gating mechanism allows the network to selectively update its hidden state based on the current input and the previous hidden state. GRUs are suitable non-symbolic models to hybridize TDTs.

By drawing inspiration from the literature [15–17], we identify three different ways to inject neural computations into decision trees:

1. using a neural network at the leaf node of a decision tree to perform the final classification (*leaf hybridization*);
2. using a neural network as a feature extractor at the internal nodes of a decision tree to take neural split decisions (*split hybridization*); and
3. replacing the root node of a decision tree with a neural network that weighs the decisions taken by the (possibly more than two) child nodes of the root (*root hybridization*).

Note that these three hybridization techniques reflect different ways of blending neural processing with logical reasoning, deployable, at least in principle, between any neural network and decision tree model. Fig. 1 depicts how to combine the hybridization methods to convey neural-symbolic decision trees. It is essential to stress that such techniques belong to a more general schema for obtaining neural-symbolic models. However, we limit the scope of this article to single decision trees and neural networks. Thus, considering the different node types in decision trees, our proposed hybridization techniques are natural.

1.2. Motivation

Similarly to other machine learning tasks, multivariate classification of time series may require both sub-symbolic and symbolic reasoning. One illustrative example is the classification of audio recordings of coughing and breathing episodes aimed at diagnosing respiratory infections; typically, this problem is approached by classifying entire recordings using sub-symbolic or, less commonly, symbolic methods after suitable preprocessing. However, certain sounds might be better identified when examined individually rather than within the entire episode. For instance, specific respiratory sounds such as *rhonchi* (low-pitched breath sounds like snoring), *crackles* (high-pitched sounds similar to popping), *wheezing* (high-pitched whistling sounds from narrowed bronchial tubes), and *stridor* (harsh, vibratory sounds from narrowed upper airways) can be more accurately detected in isolation. In these cases, the adaptability and flexibility of a sub-symbolic learning model, like a neural network, are advantageous. On the other hand, understanding how these sounds occur with each other or within the entire breathing episode (which may be several seconds long) might be better suited to a symbolic model, such as TDTs. TDTs effectively capture the temporal logical relationships, such as the pattern of *repeated string sounds overlapped by rhonchi* that might indicate a specific infection.

A natural way to combine these learning techniques is to use a pre-trained sub-symbolic model to identify specific sounds and treat these identifications as features of an interval alongside other mathematical features like the average power at a particular frequency. A symbolic learning model can then utilize these features to identify logical rules, aligning with the split hybridization method described earlier. When a pre-trained model is unavailable, it can be trained 'on-the-fly.' In other situations, a different order of combining symbolic and sub-symbolic learning might be more effective. For example, subtle differences in how the same respiratory disease is present in different subjects might be easier to capture as a mathematical function rather than a logical rule. An initial sub-symbolic clustering step can facilitate subsequent rule extraction, representing root hybridization. Conversely, after identifying general rules to cluster subjects broadly, a final classification step based on a non-linear mathematical characterization can be handled by a neural network corresponding to leaf hybridization.

Combining two or all three hybridization techniques represents a logical next step. This paper examines the three basic combinations: split, root, and leaf hybridization. We provide learning algorithms for each technique and comprehensively compare them.

1.3. Experiments breakdown

In order to assess the viability of our solutions, we perform two batches of experiments, all concerning the task of diagnosing respiratory diseases. In the first batch, we consider the problem of diagnosing COVID-19 from audio samples of coughs and breaths. These tasks have been posed as binary, multivariate time series classification problems [18,19], for which both temporal decision trees [11] and neural networks [20–24] have proven their efficacy. The purpose of this experiment is purely that of assessing the qualities of each hybridization technique in the particular case of temporal decision trees, and not that of achieving the highest absolute performances, which is often the results of a very intensive hyperparametrization, complex feature extraction, and model stacking/bagging. More in particular, we shall focus on our ability of extracting logical rules for each of the classes, with and without neural component, in order to highlight the role and the effectiveness of the latter. In the second set of experiments we shall consider a different, but similar problem, that is, a multi-class respiratory disease dataset: the Respiratory Sound Database [25]. We take advantage from the previous experiment to establish the most effective hybridization technique and parametrization, and we focus specifically on extracting interpretable rules for each class.

1.4. Structure

The article is organized as follows: in Section 2 we review the most important contributions to combining the two techniques of learning decision trees and neural networks; in Section 3 we present basic notions on multivariate time series classification, temporal decision trees and neural network-based approaches to the problem; in Section 4 we present and formalize the three different hybridization methods discussed above; then, in Section 5 we perform the experimental comparison between the proposed methods, and we show several results in terms of extracted rules, before concluding.

2. Related work

Decision trees and neural networks are well-known alternatives for pattern recognition, and their strengths and weaknesses have been studied for over three decades [26,27]. Notoriously, decision trees favor the interpretability of their

decisions thanks to their hierarchical structure, which enables one to visualize and understand the relationship between input features and outputs. However, decision trees need help with continuous numerical variables as they work best when clear thresholds or boundaries exist between the outcomes, which is only sometimes the case in numerical domains. Conversely, neural networks challenge human understanding due to their complexity (e.g., many layers) and lack of interpretability (e.g., no explicit rules), frequently referred to as *black-box* models whose inner workings are hard to comprehend yet have a better generalization capability. In this section, we focus on recent literature that explores the integration of these two distinct models.

2.1. From decision trees to neural networks

In [28–30], the authors observe that learned decision trees can establish the topology of neural networks: exploit the structure of the decision trees to build simple neural networks with just two hidden layers (without considering the input and output layers) by mapping decision nodes or rules to neural units. As a result, such networks may exhibit redundant units and connections. To this end, the authors in [31] (which extends the work in [30]), propose a pruning-based compression technique. Decision trees can also initialize radial-basis function neural networks, where the decision boundaries (i.e., regions in the instance space) of the decision tree models determine neurons in the resulting networks, with each neuron being a basis function [32]. In this article, our hybridization techniques also exploit the structure of the decision trees, focusing on distinct types of tree nodes but not extracting neural networks as final models. We use, instead, neural networks to enhance the generalization ability of temporal decision trees to convey neural-symbolic ones.

2.2. From neural networks to decision trees

In [33], the authors note a limitation in decision tree induction algorithms: recursive splits are performed over fewer instances as the trees grow. To address such an issue, they propose querying trained neural networks acting as *oracles* for generating new samples to split on. In their article, they use m-of-n Boolean splits [34] (i.e., m out of n conditions must be satisfied), and such new instances enable more statistically significant tests, having more evidence. A similar work performs univariate splits [35]. Inspired by such a line of research, in [36], the authors propose training decision trees from inputs generated from oracles rather than directly from data. In [37], the researchers present a methodology that incorporates both discrete and continuous inputs and outputs, addressing some limitations of the above methods by incorporating a novel attribute significance analysis to perform splits. Instead, in [38], the authors extract decision trees from input data jointly with new randomly generated samples from ensembles of neural networks. In this paper, we also harness the possibility of querying oracles at different degrees of hybridization; as we shall see, for example, split-level hybridizations can query oracles to perform splits.

2.3. Hybrid neural-symbolic models

In [15], the researchers enhance the generalization ability of standard decision trees by training a small perceptron, a particular neural network, having one hidden layer with a single output at each decision node of the trees. During the induction phase, their method learns a nonlinear multivariate feature $f(\cdot)$ on the (subset of) training instances of a specific decision node, splitting such samples if $f(\cdot) < 0$ (since the output has *tanh* as the activation function). Such a method shows increased accuracy compared to the original tree learning algorithm and has reduced training time compared to a more significant multi-layer neural network that uses backpropagating to train. Similar works generate oblique decision trees (e.g., see [39]), while others use convolutional neural networks in decision nodes (e.g., see [40]). The idea of neural feature extraction falls under the split-level hybridization category, as we shall formally define in this article, limiting its scope to univariate splits. In [16], the authors propose a distinguishable methodology for inducing hybrid (neural-symbolic) decision trees. Initially, decision nodes perform splits only over categorical attributes, if any, steering qualitative reasoning, and subsequently, in leaf nodes, a neural network performs the final inference only over the numerical attributes, if any, ushering quantitative reasoning. Our work draws inspiration from such methodology to formalize the leaf-level hybridization technique, performing the final prediction. Finally, in [17], decision trees replace neural networks' final layer, and the method shows promising results in computer vision. Their approach motivates our proposed root-level hybridization.

3. Background

In this section, we provide the needed foundational knowledge to comprehend the scope of this work. We begin by explaining the multivariate time series concept and the corresponding classification challenges. Subsequently, we delve into the systematic formulation of interval temporal decision trees, after an introduction to interval temporal logic. The section culminates with an analysis of neural networks, which serve as the basis for our experiments.

3.1. Multivariate time series classification

Time series are observations interpreted over a linear order, that is, they are series of temporally ordered observations. Time series can be *univariate* or *multivariate*, depending on whether there is only one measurement or multiple measure-

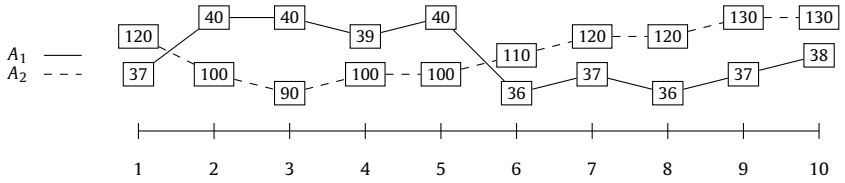


Fig. 2. Example of a multivariate time series with two attributes.

ments, and can be either numerical or categorical. Thus, a univariate time series is a single measurement evolving through time, while a multivariate time series are multiple measurements that evolve.

Formally, let $\mathfrak{D} = \langle\{1, 2, \ldots, N\}, \leq\rangle$ be a *finite linear order* of *size* N and $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a vector space with n dimensions called *attributes*. Then, a time series x is a signal drawn from the *space of* $\mathcal{A}$*-valued time series on* $\mathfrak{D}$ defined as:

$$\mathbb{X}(\mathfrak{D}, \mathcal{A}) = \{x : \mathfrak{D} \to \mathcal{A}\}.$$

Definition 1 *(Time series dataset).* Let $\mathbb{X}(\mathfrak{D}, \mathcal{A})$ the space of $\mathcal{A}$-valued time series on $\mathfrak{D}$. Then, a *time series dataset* is a finite set $\mathcal{X} = \{x_1, \ldots, x_m\} \subseteq \mathbb{X}(\mathfrak{D}, \mathcal{A})$ of m time series (also called *instances*).

The dataset $\mathcal{X}$ is called *labeled* if each instance has a label from a set of *labels* $\mathbb{Y}$ (also called *label space*), that is, $\mathcal{X} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, where $y_i \in \mathbb{Y}$, for all $1 \leq i \leq m$. When labels are categorical, they are generally referred to as *classes*. For a time series $x \in \mathbb{X}(\mathfrak{D}, \mathcal{A})$, let $x(w)[i]$ denote the value of the time series x at point w in the ith component, that is, the value of x at point w associated to A_i. Similarly, which is important in our context (as we shall see in the next subsection), let $x(w : v)[i]$ be the vector having exactly $v - w + 1$ values of A_i contained from point w to v (both included); note that, for a finite linear order $\mathfrak{D}$ of size N, $x(1 : N)[i]$ denotes the entire univariate time series represented by A_i. An example of multivariate time series is in Fig. 2, where $x(4)[1] = 39$ and $x(6 : 8)[2] = (110, 120, 120)$.

The *multivariate time series classification* problem involves extracting a classifier from a collection of labeled multivariate time series to recognize patterns that distinguish between classes. Existing multivariate time series classification methods are *feature-based* (see, e.g., [41]), *instance-based* (see, e.g., [42]) and *native* ones (see, e.g., [43]). Time series describe a variety of situations and their classification is an active area of research in many disciplines: air quality control and prediction in climate science, prices and rates of inflation in economics, infectious diseases trends and spreading patterns in medicine, pronunciation of word signs in linguistics, sensor recording of systems in aerospace engineering, among others [1].

3.2. Halpern and Shoham's interval temporal logic

Since time series represent continuous processes, it makes little sense to describe temporal patterns in terms of time points, and *intervals* are better suited. To this end, logical formalisms are needed to reason about time intervals.

While several different interval temporal logics have been proposed in the recent literature [44], *Halpern and Shoham's interval temporal logic* ($\mathcal{HS}$) [45] is certainly the formalism that received the most attention, being the most natural logic for time intervals. From a logical point of view, $\mathcal{HS}$ and its fragments have been studied on the most important classes of linearly ordered sets, from the class of all linear orders, to the classes of linear orders that can be built on classical sets such as $\mathbb{N}$ (natural numbers), $\mathbb{Q}$ (rational numbers) and $\mathbb{R}$ (real numbers) [45–47].

Let $\mathfrak{D}$ be a finite linearly ordered set of size N. An *interval* over $\mathfrak{D}$ is an ordered pair $[w, v]$ *starting* from w and *ending* in v (both included), where $w, v \in \mathfrak{D}$ and $w \leq v$. An interval is called *point interval* if $w = v$, and *strict interval* if $w < v$.[1] If we exclude the equality relation, there are twelve different binary ordering relations between two strict intervals on a linear order, often called *Allen's interval relations* [48]: the six relations $\mathcal{R}_{\mathsf{A}}$ (*adjacent to*), $\mathcal{R}_{\mathsf{L}}$ (*later than*), $\mathcal{R}_{\mathsf{B}}$ (*begins*), $\mathcal{R}_{\mathsf{E}}$ (*ends*), $\mathcal{R}_{\mathsf{D}}$ (*during*), $\mathcal{R}_{\mathsf{O}}$ (*overlaps*), depicted in Fig. 3, and their six *inverses*, that is, $\mathcal{R}_{\overline{\mathsf{X}}} = (\mathcal{R}_{\mathsf{X}})^{-1}$, for each $\mathsf{X} \in \{\mathsf{A}, \mathsf{L}, \mathsf{B}, \mathsf{E}, \mathsf{D}, \mathsf{O}\}$. Let $\mathbb{A}$ be the set of names of the twelve Allen's interval relations:

$$\mathbb{A} = \{\mathsf{A}, \mathsf{L}, \mathsf{B}, \mathsf{E}, \mathsf{D}, \mathsf{O}, \overline{\mathsf{A}}, \overline{\mathsf{L}}, \overline{\mathsf{B}}, \overline{\mathsf{E}}, \overline{\mathsf{D}}, \overline{\mathsf{O}}\}.$$

We associate an *existential modality* $\langle\mathsf{X}\rangle$ with each Allen's relation $\mathcal{R}_{\mathsf{X}}$. Let $\mathcal{P}$ be a set of *propositional letters*. $\mathcal{HS}$ formulas are generated by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\mathsf{X}\rangle\varphi,$$

where $p \in \mathcal{P}$ and $\mathsf{X} \in \mathbb{A}$. The remaining Boolean connectives are derived as usual, and we say $[\mathsf{X}]\varphi$ if and only if $\neg\langle\mathsf{X}\rangle\neg\varphi$, for each $\mathsf{X} \in \mathbb{A}$.

[1] Using both w and v as extrema of an interval intentionally clashes with the definition $x(w : v)$ for a time series $x \in \mathbb{X}(\mathfrak{D}, \mathcal{A})$.

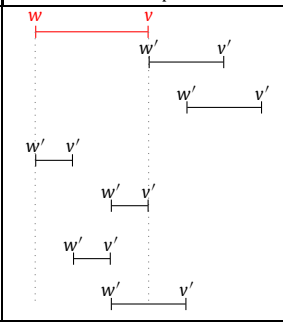
$\mathcal{HS}$ modality	Definition w.r.t. the interval structure			Example
$\langle A \rangle$	$[w,v]\mathcal{R}_A[w',v']$	iff	$v = w'$	
$\langle L \rangle$	$[w,v]\mathcal{R}_L[w',v']$	iff	$v < w'$	
$\langle B \rangle$	$[w,v]\mathcal{R}_B[w',v']$	iff	$w = w' \wedge v' < v$	
$\langle E \rangle$	$[w,v]\mathcal{R}_E[w',v']$	iff	$v = v' \wedge w < w'$	
$\langle D \rangle$	$[w,v]\mathcal{R}_D[w',v']$	iff	$w < w' \wedge v' < v$	
$\langle O \rangle$	$[w,v]\mathcal{R}_O[w',v']$	iff	$w < w' < v < v'$	

Fig. 3. Allen's interval relations and $\mathcal{HS}$ modalities.

The strict semantics of $\mathcal{HS}$ are given in terms of *timelines* (or, more commonly, *interval models*):

$$\mathfrak{K} = (\mathcal{W}, \{\mathcal{R}_X\}_{X \in \mathbb{A}}, V),$$

where $\mathcal{W}$ is the set of *strict intervals over* $\mathfrak{D}$, $\mathcal{R}_X$ are Allen's *interval relations*, and V is a *valuation function* $V : \mathcal{W} \to 2^{\mathcal{P}}$ which assigns to every interval $[w,v]$ the set of propositional letters $V([w,v]) \subseteq \mathcal{P}$ that are true on it.[2] For a timeline $\mathfrak{K}$, an interval $[w,v]$ (in that model) and a formula φ of $\mathcal{HS}$ (to be evaluated on that model), the *truth relation* $\mathfrak{K}, [w,v] \Vdash \varphi$ is defined by structural induction on the complexity of formulas:

$$\begin{array}{lll}
\mathfrak{K}, [w,v] \Vdash p & \text{iff} & p \in V([w,v]), \text{ for all } p \in \mathcal{P}; \\
\mathfrak{K}, [w,v] \Vdash \neg\psi & \text{iff} & \mathfrak{K}, [w,v] \nVdash \psi; \\
\mathfrak{K}, [w,v] \Vdash \psi_1 \vee \psi_2 & \text{iff} & \mathfrak{K}, [w,v] \Vdash \psi_1 \text{ or } \mathfrak{K}, [w,v] \Vdash \psi_2; \\
\mathfrak{K}, [w,v] \Vdash \langle X \rangle \psi & \text{iff} & \exists\, [w',v'] \text{ s.t. } [w,v]\mathcal{R}_X[w',v'] \text{ and } \mathfrak{K}, [w',v'] \Vdash \psi,
\end{array}$$

where $X \in \mathbb{A}$. Note that relations in $\mathbb{A}$ are mutually exclusive and jointly exhaustive with respect to $\mathcal{W}$; as such, exactly one relation holds for each pair of intervals in $\mathcal{W}$, and an existential *global operator* can be defined by disjunction of all modal operators:

$$\langle G \rangle \varphi = \varphi \vee \bigvee_{X} \langle X \rangle \varphi,$$

with $X \in \mathbb{A}$. The global operator allows to express global formulas (e.g., "there exists an interval *anywhere*..."), that is formulas of the kind:

$$\begin{array}{rcl}
\varphi^g & ::= & \neg\varphi^g \mid \varphi^g \vee \varphi^g \mid \langle G \rangle \varphi \\
\varphi & ::= & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle X \rangle \varphi \mid \langle G \rangle \varphi,
\end{array}$$

with $p \in \mathcal{P}$ and $X \in \mathbb{A}$. Given that any global formula φ^g either holds on all worlds of a timeline $\mathfrak{K}$, or it does not hold on any world of $\mathfrak{K}$, we can think of φ^g as a property that $\mathfrak{K}$ can have. As such, we write $\mathfrak{K} \Vdash \varphi^g$ when φ^g holds on all worlds, and $\mathfrak{K} \nVdash \varphi^g$ otherwise.

Interval temporal logics have been studied in the literature from a deductive standpoint. Satisfiability for $\mathcal{HS}$ is undecidable [45], and that various fragments have been considered in the literature to define fragments or variants of $\mathcal{HS}$ with better computational behavior. These include constraining the underlying temporal structure [49], restricting the set of modal operators [50,46], limiting the propositional power of the languages [51], and considering *coarser* interval temporal logics based on interval relations that describe a less precise relationship between intervals (similarly to what topological relations do) [52].

3.3. Interval temporal decision trees

In this subsection, we shall see how to extract knowledge (in terms of logical rules) from raw multivariate time series. Canonical decision trees (DTs), essentially based on propositional logic, are not able to learn directly from temporal structures, and linear temporal logic-like decision trees (such as, e.g., in [53,54]) learn point-based patterns, which, as we have observed, may not be right inductive bias choice for continuous processes. Interval temporal logic decision trees (TDTs), instead, learn $\mathcal{HS}$ properties by extending the classical learning schema of DTs, to capture relations among time intervals.

[2] In the standard literature, timelines are Kripke models, where the worlds are intervals.

Let $\tau = (\mathcal{V}, \mathcal{E})$ be a *full directed binary tree* with *nodes* in $\mathcal{V}$ and *edges* in $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. We denote by $root(\tau)$ the *root of* τ, by $\mathcal{V}^\ell \subseteq \mathcal{V}$ the set of its *leaf nodes* (or, simply, *leaves*), and by $\mathcal{V}^\iota$ the set of its *internal nodes* (i.e., non-leaf nodes). We also denote nodes (either root, internal or leaf) by $\nu, \nu', \ldots, \nu_1, \nu_2, \ldots$ and leaves by $\ell, \ell', \ldots, \ell_1, \ell_2, \ldots$. Each non-leaf node ν of a tree τ has a *left* (resp. *right*) *child* $\swarrow(\nu)$ (resp., $\searrow(\nu)$), and each non-root node ν has a *parent* $\wr(\nu)$. For every $y \in \mathbb{Y}$, where $\mathbb{Y}$ is the label space, we denote by $leaves(y)$ the set of *leaves of* τ *labeled with* y. A *path* $\pi = \nu_0 \rightsquigarrow \nu_h$ *in* τ of *length* $h \geq 0$ between two nodes ν_0 and ν_h is a finite sequence of $h+1$ nodes such that $\nu_i = \wr(\nu_{i+1})$, for each $i = 0, \ldots, h-1$. For a path π and for a node ν in τ, π_ν denotes the unique path $root(\tau) \rightsquigarrow \nu$. Moreover, for a path π, the set of its *improper prefixes* is denoted by $prefix(\pi)$. Finally, a *branch of* τ is a path π_ℓ, for some $\ell \in V^\ell$.

Canonically, datasets are structured, meaning that, with respect to time series datasets, the attributes do *not* evolve through time. A structured dataset induces the following set of propositional letters:

$$\mathcal{P} = \{A \bowtie a \mid A \in \mathcal{A}, \bowtie \in \{<, \leq, =, \neq, \geq, >\}, \text{ and } a \in \mathbb{R}\}.$$

Observe that such a definition is natural for standard DTs. Indeed, DTs work by recursively testing scalar conditions on the values of input features (A); it is immediate to observe that the definition of induced propositional letters encodes such a concept.

A time series dataset generalizes to the temporal case a structured one, by postulating that finite timelines describe instances/time series in which attributes change value across the different intervals. The key idea of (interval) TDTs is that they naturally reason over intervals of temporal ordered structures, such as, but not limited to, time series, without any transformation of the input signals. Since reasoning is interval-based, we can guide the learning process by making decisions over the entire set of values within a certain strict interval. Thus, let $\mathcal{F} = \{f_1, \ldots, f_r\}$ be a set of *feature extraction functions*, where $f_i : \bigcup_{d=2}^{N} \mathbb{R}^d \to \mathbb{R}$, for each $i = 1, \ldots, r$. Each feature extraction function $f \in \mathcal{F}$ computes a real number given the vector of values for an attribute A within a certain strict interval whose length vary from 2 (the minimum size of an interval) to N (the maximum size of an interval, i.e., the size of the entire time series); with a slight abuse of notation, we denote the result of this computation by $f(A)$. A time series dataset induces the following propositional letters (that consider the interval-based reasoning):

$$\mathcal{P} = \{f(A) \bowtie a \mid f \in \mathcal{F}, A \in \mathcal{A}, \bowtie \in \{<, \leq, =, \neq, \geq, >\}, \text{ and } a \in \mathbb{R}\}.$$

Feature extraction functions should be, in principle, simple functions, ranging from the average, minimum or maximum value, to more complex yet interpretable functions (e.g., *Catch22* feature functions [55], which are studied ad-hoc for time series).

The following definition extends the one for splitting instances in DTs, which we use to define TDTs.

Definition 2 *(Decisions).* Let $\mathcal{P}$ be a set of propositional letters. Then, the set of *(temporal) split-decisions* (or, simply, *decisions*) is:

$$\Lambda = \{p, \neg p, \langle \mathsf{X} \rangle p, [\mathsf{X}] \neg p \mid p \in \mathcal{P} \text{ and } \mathsf{X} \in \mathbb{A}\}.$$

We partition the set Λ into the set of *existential* decisions $\Lambda^e = \{p, \langle \mathsf{X} \rangle p \mid p \in \mathcal{P} \text{ and } \mathsf{X} \in \mathbb{A}\}$ and the set of *universal* decisions $\Lambda^u = \Lambda \setminus \Lambda^e$. Moreover, decisions are said to be *propositional* if they are of the type p or $\neg p$. The *dual* of $p \in \Lambda^e$ (resp., $\langle \mathsf{X} \rangle p \in \Lambda^e$) is $\neg p \in \Lambda^u$ (resp., $[\mathsf{X}] \neg p \in \Lambda^u$), and vice versa; in general, we denote by $\overline{\lambda}$ the dual of $\lambda \in \Lambda$.

Definition 3 *(Interval temporal decision trees).* Let Λ be a set of decisions and $\mathbb{Y}$ a label space. Then, an *interval temporal decision tree (TDT)* is a structure:

$$\tau = (\mathcal{V}, \mathcal{E}, l, e),$$

where:

- $(\mathcal{V}, \mathcal{E})$ is a full directed binary tree,
- $l : \mathcal{V}^\ell \to \mathbb{Y}$ is a *leaf-labeling function* that assigns an element from $\mathbb{Y}$ to each leaf node in $\mathcal{V}^\ell$,
- $e : \mathcal{E} \to \Lambda$ is a *edge-labeling function* that assigns a decision from Λ to each edge in $\mathcal{E}$,

such that $e(\nu, \nu') = \neg e(\nu, \nu'')$, for all $(\nu, \nu'), (\nu, \nu'') \in \mathcal{E}$.

Note that a DT is a TDT where the decisions are propositional only. Fig. 4 illustrates an example of TDT.

We discuss how formulas of an interval temporal decision tree are built.

Definition 4 *(Node agreements).* Let τ be an interval temporal decision tree and let $\pi = \nu_0 \rightsquigarrow \nu_h$ be a path in τ. Given two nodes $\nu_i, \nu_j \in \pi$, with $i, j < h$, we say that they *agree*, denoted by $agr(\nu_i, \nu_j)$, if $\nu_{i+1} = \swarrow(\nu_i)$ and $\nu_{j+1} = \swarrow(\nu_j)$, and we say that they *disagree*, denoted by $dis(\nu_i, \nu_j)$, if $\nu_{i+1} = \swarrow(\nu_i)$ and $\nu_{j+1} = \searrow(\nu_j)$.

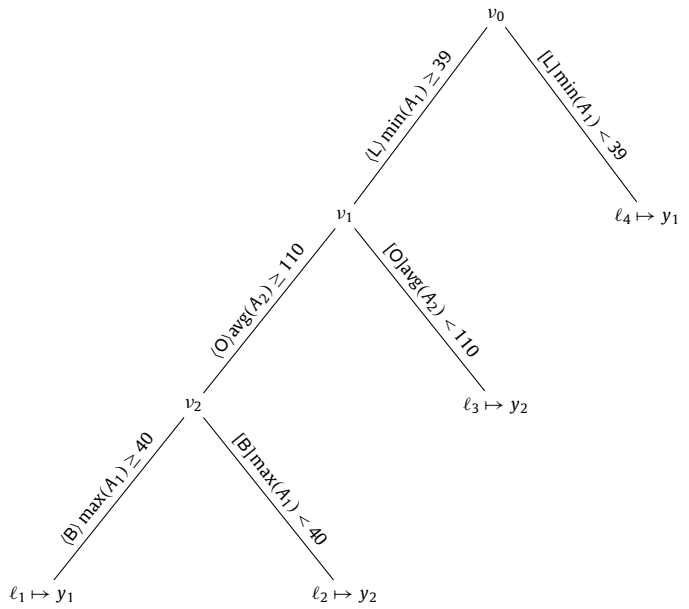


Fig. 4. Example of an interval temporal decision tree. Note that it holds, for example, that $root(\tau) = v_0$, $\swarrow(v_0) = v_1$, and $\swarrow(v_1) = v_2$.

Definition 5 *(Path-, leaf-, and class-formulas).* Let τ be an interval temporal decision tree. For each path $\pi = v_0 \rightsquigarrow v_h$ in τ, the *path-formula* φ_π is inductively defined as:

- if $h = 0$, then $\varphi_\pi = \top$;
- if $h = 1$, then $\varphi_\pi = e(v_0, v_1)$;
- if $h > 1$, let $\lambda = e(v_0, v_1)$ and $\pi' = v_1 \rightsquigarrow v_h$, then

$$\varphi_\pi = \begin{cases} (\lambda \wedge \varphi_{\pi'}) & \text{if } \lambda = p \text{ and } agr(v_0, v_1); \\ \langle \mathsf{X}\rangle (p \wedge \varphi_{\pi'}) & \text{if } \lambda = \langle \mathsf{X}\rangle p \text{ and } agr(v_0, v_1); \\ \lambda \wedge (\lambda \rightarrow \varphi_{\pi'}) & \text{if } \lambda = p \text{ and } dis(v_0, v_1); \\ \lambda \wedge [\mathsf{X}](p \rightarrow \varphi_{\pi'}) & \text{if } \lambda = \langle \mathsf{X}\rangle p \text{ and } dis(v_0, v_1); \\ (\lambda \wedge \varphi_{\pi'}) & \text{if } \lambda \in \Lambda^u. \end{cases}$$

Moreover, for a leaf ℓ, the *leaf-formula* φ_ℓ is defined as:

$$\varphi_\ell = \bigwedge_{\pi \in prefix(\pi_\ell)} \varphi_\pi,$$

and, for a class y, the *class-formula* φ_y is defined as:

$$\varphi_y = \bigvee_{\ell \in leaves(y)} \varphi_\ell.$$

For example, consider the TDT in Fig. 4. Then, the path-formula π_{ℓ_1} for ℓ_1 is:

$$\varphi_{\pi_{\ell_1}} = \langle \mathsf{L}\rangle \big(\min(A_1) \geq 39 \wedge \langle \mathsf{O}\rangle (\text{avg}(A_2) \geq 110 \wedge \langle \mathsf{B}\rangle \max(A_1) \geq 40) \big),$$

and the class formula φ_{y_1} for y_1 is:

$$\varphi_{y_1} = \Big(\overbrace{\langle \mathsf{L} \rangle \min(A_1) \geq 39}^{\varphi_{\pi_{\nu_1}}} \wedge \overbrace{\langle \mathsf{L} \rangle (\min(A_1) \geq 39 \wedge \overbrace{\langle \mathsf{O} \rangle \mathrm{avg}(A_2) \geq 110}^{\varphi_{\pi_{\nu_2}}})}^{\varphi_{\pi_{\ell_1}}} \wedge \overbrace{\langle \mathsf{L} \rangle \big(\min(A_1) \geq 39 \wedge \langle \mathsf{O} \rangle (\mathrm{avg}(A_2) \geq 110 \wedge \langle \mathsf{B} \rangle \max(A_1) \geq 40) \big)}^{\varphi_{\pi_{\ell_1}}} \Big) \vee \overbrace{[\mathsf{L}](\min(A_1) < 39)}^{\varphi_{\pi_{\ell_4}}}.$$

We shall define how $\mathcal{HS}$ formulas are interpreted on time series. To this end, we need to interpret $\mathcal{HS}$ formulas over time series. We do so by identifying time series as timelines, and by interpreting propositional formulas of the type $f(A_i) \bowtie a$ as:

$$x, [w, v] \Vdash f(A_i) \bowtie a \text{ iff } f(x(w:v)[i]) \bowtie a.$$

Now, we can exploit the truth relation for $\mathcal{HS}$ (see Subsection 3.2), where TDTs learn (and provide) $\mathcal{HS}$ formulas that must be checked on time series x (seen as timelines).

The classification of a single time series $x \in \mathbb{X}(\mathfrak{D}, \mathcal{A})$ starts at the root. Each series (that is, each timeline) has an *initial* dummy interval $[-1, 0] \notin \mathfrak{D}$ so that the only feasible operators at the beginning of the classification process are $\langle \mathsf{L} \rangle$ for the left branch and $[\mathsf{L}]$ for the right branch; in this way, we can describe (shift-invariant) patterns that appear anywhere in the series. For a time series x and formula φ, we use $x \Vdash \varphi$ to denote $x, [-1, 0] \Vdash \varphi$. Thus, a learned TDT classifies a time series as follows.

Definition 6 *(Run of interval temporal decision trees).* Let $\tau = (\mathcal{V}, \mathcal{E}, l, e)$ be an interval temporal decision tree, ν a node in τ, and x a time series. Then, the *run of* τ *on* x *from* ν, denoted by $\tau(x, \nu)$, is defined recursively as:

$$\tau(x, \nu) = \begin{cases} l(\nu) & \text{if } \nu \in V^{\ell}; \\ \tau(x, \swarrow(\nu)) & \text{if } x \Vdash \varphi_{\pi_{\swarrow(\nu)}}; \\ \tau(x, \searrow(\nu)) & \text{if } x \Vdash \varphi_{\pi_{\searrow(\nu)}}. \end{cases}$$

Finally, the *run of* τ *on* x, denoted by $\tau(x)$, is defined as $\tau(x, root(\tau))$.

Consider the time series x in Fig. 2 and the TDT τ in Fig. 4. Then, we have that:

$$\begin{aligned} \tau(x, root(\tau)) &= \tau(x, \nu_1) && (x \Vdash \varphi_{\nu_1}) \\ &= \tau(x, \nu_2) && (x \Vdash \varphi_{\nu_2}) \\ &= \tau(x, \ell_1) && (x \Vdash \varphi_{\ell_1}) \\ &= l(\ell_1) && (\ell_1 \in \mathcal{V}^{\ell}) \\ &= y_1. \end{aligned}$$

We can complete the logical interpretation of temporal decision trees by defining leaf- and class-rules, which model the knowledge that certain formulas are sufficient and necessary conditions for a given class.

Definition 7 *(Leaf- and class-rules).* Let τ be an interval temporal decision tree. Then, for a leaf ℓ, the *leaf-rule* ρ_{ℓ} is defined as an object:

$$\varphi_{\ell} \Rightarrow l(\ell),$$

and, for a class y, the *class-rule* ρ_y is defined as an object:

$$\varphi_y \Leftrightarrow y.$$

Rules, as defined above, are written in a non-logical language because classes are not logical elements; however, in the context of decision trees, they behave like logical implications. Referring, once more, to the example tree in Fig. 4, the leaf-rule ρ_{ℓ_2} is:

$$(\underbrace{\langle \mathsf{L} \rangle p}_{\varphi_{\pi_{\nu_1}}} \wedge \underbrace{\langle \mathsf{L} \rangle (p \wedge \langle \mathsf{O} \rangle q)}_{\varphi_{\pi_{\nu_2}}} \wedge \underbrace{\langle \mathsf{L} \rangle (p \wedge \langle \mathsf{O} \rangle q \wedge [\mathsf{O}](q \rightarrow [\mathsf{B}] r))}_{\varphi_{\pi_{\ell_2}}}) \Rightarrow y_2,$$

where $p = \min(A_1) \geq 39, q = \text{avg}(A_2) \geq 110, r = \max(A_1) < 40$. In a sense, a leaf-rule provides a *sufficient* condition for a class. In the particular case of this example, the above leaf-rule can be simplified into:

$$\langle \mathsf{L} \rangle (p \wedge \langle \mathsf{O} \rangle q \wedge [\mathsf{O}](q \rightarrow [\mathsf{B}]r)) \Rightarrow y_2,$$

which can be easily interpreted in natural language: *IF there exists an interval where the minimum of attribute 1 is not less than 39, that is overlapping with at least another interval where the average of attribute 2 is not less than 110, and that all such intervals are prefixed by smaller intervals that all have the maximum of attribute 2 smaller than 40 THEN the instance belongs to* y_2. The class-rule ρ_{y_2} is:

$$\underbrace{\langle \mathsf{L} \rangle (p \wedge \langle \mathsf{O} \rangle q \wedge [\mathsf{O}](q \rightarrow [\mathsf{B}]r))}_{\varphi_{\ell_2}} \vee \underbrace{(\langle \mathsf{L} \rangle p \wedge [\mathsf{L}](p \wedge [\mathsf{O}]\neg q))}_{\varphi_{\ell_3}} \Leftrightarrow y_2,$$

which, in turn, can be interpreted as a *necessary and sufficient* condition for y_2. It is crucial to stress that simple feature extraction functions like the minimum and maximum values lead to more intuitive sentences/concepts; for this reason, the use of such functions enables more interpretable models, which, as we shall see, we use as the pure symbolic baseline in our experiments.

Given a dataset of m time series, any classification rule for a class can be evaluated in terms of precision and degree of applicability, by considering the number of instances m_ℓ that reached to the corresponding leaf, and the number of instances m_ℓ^c that the rule correctly classifies. Two widely used metrics in this context are the *confidence* and *support*, defined by:

$$\begin{aligned} \textit{confidence} &= \frac{m_\ell^c}{m_\ell} \\ \textit{support} &= \frac{m_\ell}{m}. \end{aligned}$$

Statistical trade-offs between these two metrics often emerge.

Extracting optimal DTs from a structured (or tabular) dataset in terms of the relation between their height and performance is an NP-hard problem [56], justifying sub-optimal procedures. As such, greedy recursive algorithms like ID3, C4.5, and CART are preferred [9,57]. Without loss of generality, the standard way of learning binary DTs for classification is straightforward: given a set of predefined, finite set of split conditions $\hat{\Lambda}$, starting from the root node that has the entire (labeled) dataset associated, select the locally optimal split decision, that partitions the dataset into two subsets, creating a split node, passing the two subsets to the children, and recursively call the same procedure on both children. This procedure propagates down the tree subsets containing progressively similar intra-node and dissimilar inter-node class values at any given level of the tree. Information-based metrics (e.g., Entropy and Gini-index) evaluate the (dis)similarity. The procedure continues until a stopping criterion applies based on the purity of a node (i.e., how is the distribution of class values), producing a leaf node. We deal with time series datasets in our context, and standard DTs are not immediately applicable, as they work only on structured datasets. In many data science and machine learning contexts, a way to deal with such an issue is the compress the temporal information of time series into a (coarser) feature representation, producing a structured dataset. There are at least two downsides to such a naive strategy. First, we lose the essential temporal dimension by shifting from N points to just one. Second, local patterns are lost because the resulting feature representation aims to summarize the (temporal) information globally. Accordingly, using DTs to learn from time series datasets might be the wrong inductive bias, motivating the design of TDTs.

Regarding the learning of TDTs, we exploit the CART algorithm to learn DTs for which a modern version in the Julia programming language is available [57]. TDTs belong to the broader class of *modal decision trees* [58], for which an implementation of a CART-like learning algorithm is available in the Julia language [58]. It is crucial to stress that the added complexity of inducing TDTs lies in the fact that such algorithms must make more expressive decisions due to the usage of $\mathcal{HS}$. Specifically, in DTs, decisions are simple proposition letters, while in TDTs, decisions also encompass interval-interval relations and feature extractions, but the general idea of greedy, recursive selection of split conditions remains. As such, the structure of the decisions is a hyperparameter (e.g., which interval-interval relations and feature extraction functions to use). Moreover, in line with DTs, other hyperparameters for TDTs are the information metric to evaluate the goodness of split conditions (e.g., Shannon Entropy [59]), and, as stopping criteria preventing the occurrence of little informative splits, the minimum information gain at the split nodes, the minimum number of instances at leaf nodes, and maximum information at leaf nodes. A formal presentation on learning TDTs is beyond the scope of this work, but the reader can refer to [11] for a formal presentation (and pseudocode). DTs are generalizable into sets of decision trees that operate by a majority voting policy. Coupling this scheme with bagging [60], an ensemble learning technique that ultimately reduces the variance of the models, in this case, sets of trees are often called *random forests* [61]. They tend to perform much better than single trees and are more popular. While they are considered to be on the verge between symbolic and sub-symbolic learning, their symbolic nature is still evident: sets of trees, similar to single trees, can be analyzed and discussed, and although the process of extracting rules is not as immediate as in single trees, it is still possible [62–64]. The generalization forest models are also straightforward for TDTs [10,11].

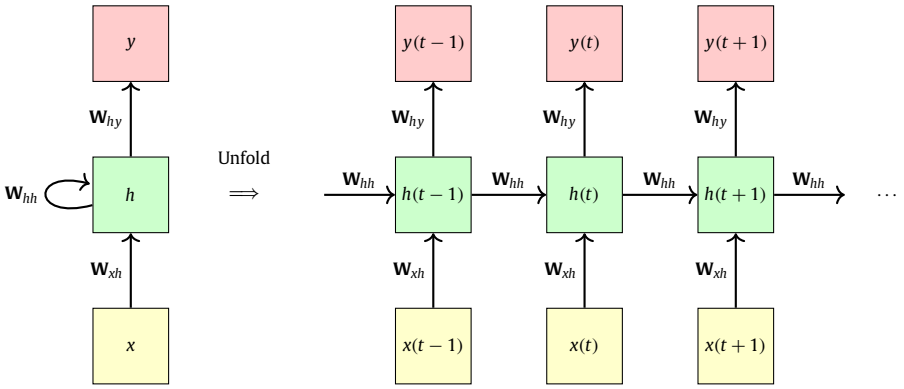


Fig. 5. Architecture of recurrent neural networks.

3.4. Neural networks

Neural networks are computational models composed of a graph of many simple computational cells, called neurons. A layer in a neural network is a set of neurons that computes its values using those from previous layers. Each a neuron calculates a value by applying a linear computation on the output values of neurons of the previous layer, followed by a non-linear activation function (e.g., *sigmoid* and *tanh*). The first layer holds the network's input, and the last layer produces the network's output. As such, a neural network is essentially a mathematical function υ that is the composition of many non-scalar, non-linear functions. Neural networks have proven highly effective at solving complex learning tasks in supervised and unsupervised settings.

Regarding temporal applications, recurrent neural networks (RNNs) are the go-to architecture, given their ability to process data sequentially and maintain information about the past to make predictions about future inputs. RNNs have hidden states $h(t)$ updated at each timestep t and passed to the next timestep as input, allowing the network to learn patterns across a time dimension. Gated recurrent units (GRUs) are popular RNN-like architectures with a gating mechanism that enables the networks to selectively update their hidden state based on the current input and previous hidden state. GRUs effectively address the vanishing gradient problem, where the gradients become smaller and smaller as they propagate through the networks, eventually disappearing, preventing the networks from learning long-term dependencies. Fig. 5 illustrates the general architecture of RNNs. In our context, we deal with time series $x = (x(1), x(2), \ldots, x(N))$ with N time points (i.e., the size of the linear order $\mathfrak{D}$), where $x(t)$ is a column vector of n values for each of the attributes $A_i \in \mathcal{A}$. RNNs calculate their hidden state at time t as follows:

$$h(t) = \phi(\mathbf{W}_{hh} h(t-1) + \mathbf{W}_{xh} x(t)),$$

where ϕ is a non-linear activation function, and $\mathbf{W}_{hh}$ and $\mathbf{W}_{xh}$ are hidden-to-hidden and input-to-hidden learnable weight matrices, respectively. Typically, the elements of $h(0)$ are all zeros. Moreover, RNNs calculate their output at time t as follows:

$$y(t) = softmax(\mathbf{W}_{hy} h(t)),$$

where *softmax* produces valid probabilities and $\mathbf{W}_{hy}$ is a hidden-to-output learnable weight matrix. It is crucial to stress that the output is optional in vanilla RNNs, and we draw inspiration from such an observation to present the hybridization techniques.

The typical approach when tackling a classification problem by neural network training involves structuring the network's last layer with as many neurons as the number of classes and training the network to output high values for neurons corresponding to the correct classes. To this end, the *softmax* function produces a probability distribution over the classes, followed by an *argmax* function to select the class with the highest probability. Fig. 6 schematically illustrates the classification neural network models. In this case, the neural network $\bar{\upsilon}$ returns the predicted class $\hat{y} \in \mathbb{Y}$ for an input time series $x \in \mathbb{X}(\mathfrak{D}, \mathcal{A})$, that is, $\bar{\upsilon}(x) = \hat{y}$. As we shall see, we use such architectures for the leaf-level hybridizations.

Feature extraction and clustering are also possible with neural networks in the unsupervised setting. As for feature extraction, the most common technique is autoencoding. *Autoencoders* are neural networks that extract significant feature representations from (unlabeled) data. This process is possible by training the networks to reproduce their input (i.e., learning the identify function) while introducing an information bottleneck (i.e., reducing the hidden states dimension). The neural networks are *encoder-decoder architectures*, where the encoder ends with a layer with a reasonably small number of neurons fed into the decoder as input. As such, the encoder compresses the input while the decoder takes such representation to reproduce the original information. For this work, we consider *sequence-to-sequence architectures* [65] as

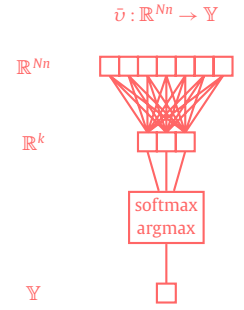


Fig. 6. Architecture for neural networks for the leaf-level hybrid.

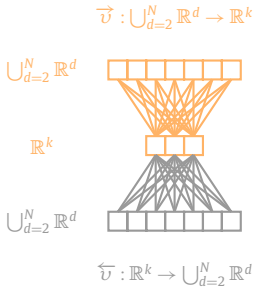


Fig. 7. Architecture for neural networks for the split-level hybrid.

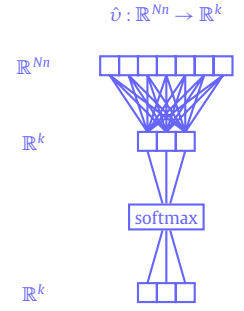


Fig. 8. Architecture for neural networks for the root-level hybrid.

autoencoders. In autoencoders, the encoder and decoder consist of several layers of neurons (typically decreasing in the encoder and increasing in the decoder). On the other hand, in sequence-to-sequence models, the sequential structure of RNNs preserves the input's time component, which we use for our purposes. Moreover, canonically, in sequence-to-sequence neural networks, the last hidden state $h(N)$, also called *context*, is taken by the decoder to produce its output.

Like autoencoders, in our setting, the sequence-to-sequence encoders approximate the identity function to represent the input, making it reproducible by the decoder. In essence, we use such encoders as feature extractors. Fig. 7 depicts the encoder-decoder architectures that we use for the split-level hybridizations. In this case, the encoder neural networks $\overrightarrow{\upsilon}$ compress their input $\bigcup_{d=2}^{N} \mathbb{R}^{d}$ (recall our definition of feature extraction functions in Subsection 3.3) into a lower dimensional representation $\mathbb{R}^{k}$, and the decoders $\overleftarrow{\upsilon}$ take such representation to reproduce the original signal.

Autoencoders also suit well for clustering. Recall that neural feature extractors output lower dimensional vector representations. Such representations can be the input to standard or specific clustering algorithms. For our purposes, instead, we use such representations to weigh the downstream predictions. Fig. 8 illustrates the architecture that we harness to define root-level hybridizations. In this case, the neural networks $\hat{\upsilon}$ produce lower dimensional representations $\mathbb{R}^{k}$, followed by the *softmax* function, producing k valid probabilities. As we shall see, one can virtually imagine that the root nodes of TDT root-level hybrids embed such architectures. Our definitions of TDTs have binary splits (i.e., have two children). Embedding such clustering-like architectures into root nodes produces k children, which, in turn, are TDTs. As such, each TDT predicts a class for a specific time series, and the hybrid root nodes weigh such predictions based on the probabilities. Therefore, the root-level hybrids measure the contribution of each of their k children for the final prediction.

The typical approach for learning neural networks involves initializing a network's with random weights and iteratively updating the weights at each layer based on the direction of the gradients. This basic optimization algorithm, called gra-

dient descent, has improved over the years by introducing more sophisticated techniques like Adam [66] and its variant AdamW [67]. These approaches help by incorporating adaptive learning rates and momentum, enabling faster convergence and effectively navigating complex loss landscapes. The learning rate guides the update of the weights, which value influences the speed and quality of convergence in the training process. During training, the model processes the training set many times (epochs), enabling gradual refinement and better generalization of the learned features and patterns. The model weights can be updated during each iteration at different moments, depending on the training strategy. A common strategy is the so-called *batch gradient descent*: it consists of processing the training set into non-overlapping subsets (or *batches*), and performing a unique update of the weights at each epoch, that is the average of the individual update given by each instance in the batch. The model processes the batch passing through the network, calculating the loss function, and updating the weights. The loss function depends on the task and guides the training process by quantifying the disparity between predicted and actual outcomes. Some examples of loss functions are the cross-entropy loss for classification tasks and MSE (mean squared error), RMSE (root mean squared error), or MAE (mean absolute error) for regression tasks.

4. Neural-symbolic hybrids

This section presents three schemes for injecting neural computation into any decision tree model. The three resulting models arise when considering injection into a single device of the original tree: leaf, split, or root nodes. Since the infusion only affects the inner workings of one of these mathematical objects, we name the three schemes *leaf hybridization*, *split hybridization* and *root hybridization*. We formalize the corresponding neuro-symbolic hybrid models for each scheme, adapting the presentation to the case of interval temporal decision trees (TDTs) and recurrent neural networks, and propose a learning algorithm accordingly. The proposed algorithms are based on the training algorithms for the underlying trees and networks and, thus, inherit their hyperparameters. Note that the schemes can be combined, giving rise to seven different hybrid models, which can, together with the pure decision tree, be partially ordered according to their residual level of interpretability, as illustrated in Fig. 1. In all cases, the resulting hybrid models and learning settings extend and generalize those of their pure symbolic and neural constituents.

4.1. Leaf hybridization

Similarly to [16], leaf hybridization consists of replacing the class labels at the leaves of the tree with neural networks, that are to be applied on any instance reaching the leaf, in order to obtain the final classification. This kind of *leaf-level hybrid* first performs a clustering step by means of interpretable, symbolic computation, and then delegates the final classification to some neural network that is, in principle, more specific to that cluster. A few modifications to the leaf-labeling function, and the operational semantics of the temporal tree suffice to formalize this model.

Definition 8 *(Leaf-level interval temporal decision tree hybrids).* Let Λ be a set of decisions, $\mathbb{Y}$ a label space, and $\mathcal{N}$ a set of neural network classifiers of the type $\bar{\upsilon} : \mathbb{R}^{Nn} \to \mathbb{Y}$. Then, a *leaf-level (interval) temporal decision tree hybrid* is a structure:

$$\tau_{leaf} = (\mathcal{V}, \mathcal{E}, l, e),$$

where:

- $(\mathcal{V}, \mathcal{E})$ is a full directed binary tree,
- $l : \mathcal{V}^{\ell} \to \mathcal{N}$ is a *leaf-labeling function* that assigns a neural network classifier to each leaf node in $\mathcal{V}^{\ell}$,
- $e : \mathcal{E} \to \Lambda$ is a *edge-labeling function* that assigns a decision from Λ to each edge in $\mathcal{E}$,

such that $e(\nu, \nu') = \neg e(\nu, \nu'')$, for all $(\nu, \nu'), (\nu, \nu'') \in \mathcal{E}$.

Definition 9 *(Run of leaf-level temporal decision tree hybrids).* Let $\tau_{leaf} = (\mathcal{V}, \mathcal{E}, l, e)$ be a leaf-level temporal decision tree hybrid, ν a node in τ, and x a time series. Then, the *run of τ on x from ν*, denoted by $\tau(x, \nu)$, is defined recursively as:

$$\tau(x, \nu) = \begin{cases} l(\ell)(x) & \text{if } \nu \in V^{\ell}; \\ \tau(x, \swarrow(\nu)) & \text{if } x \Vdash \varphi_{\pi_{\swarrow(\nu)}}; \\ \tau(x, \searrow(\nu)) & \text{if } x \Vdash \varphi_{\pi_{\searrow(\nu)}}, \end{cases}$$

Finally, the *run of τ on x*, denoted by $\tau(x)$, is defined as $\tau(x, root(\tau))$.

While the definitions of leaf- and class-formulas (Definition 5) remain the same, leaf-rules (Definition 7) have neural classifiers as consequents, which slightly changes the semantics: a leaf-formula $\varphi_{\ell} \Rightarrow \bar{\upsilon}_{\ell}$, has the semantics *IF instance x satisfies the leaf-formula φ_{ℓ} THEN it belongs to $\bar{\upsilon}_{\ell}(x)$*. An advantage of this computational model is that the antecedents of the rules are still logical formulas, and thus, some form of interpretation of the knowledge is still possible. The antecedents, in

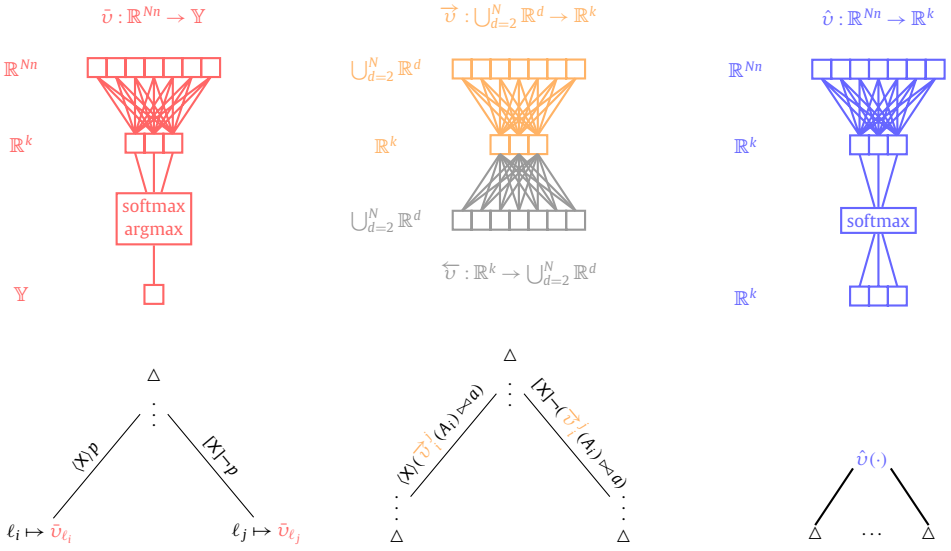


Fig. 9. Neural-symbolic interval temporal decision trees: leaf-level hybrid (left), split-level hybrid (middle), root-level hybrid (right).

fact, still induce a partitioning of the instance space, which can be inspected and enhanced with expert knowledge. On the other hand, with such a model there is no natural way of deriving class-rules.

In principle, the proposed learning algorithm for this hybridization scheme requires training a pure decision tree, as well as a neural network classifier for each leaf, using the instances that reached the leaf itself. To this end, we make the following observations. Regarding the training dataset, the sub-datasets traversing the tree to the leaves are exponentially smaller; therefore, simply training each network on the set of instances that reached the leaf may result in a weaker model. Since such an algorithm must ensure that a high enough number of time series is used to train each neural network, we use a strict pre-pruning policy. Leveraging the recursive structure of the tree, we propose a layered training algorithm, based on *hierarchical fine-tuning*. At the beginning, a master network is trained, associated to the root node of the tree, and two copies of it are associated to the child nodes; then, in a top-down fashion, the network associated to each node of the tree is fine-tuned with the set of instances that reached that node, and two copies of it are propagated (and associated) to the child nodes. In the end, each leaf node ℓ holds a copy of the original network that has been fine-tuned many times, each time becoming more and more specific to the current partition of the instance space; the leaf, then, is labeled with the network itself. Hyperparameters for these models are those used by the underlying decision tree and neural network classifier learning algorithms, as well as the hyperparameters used for the fine-tuning, since their values may differ from those used for the training of the master network. This hybrid collapses to a pure tree when the neural architecture in use collapses to a simple constant model (i.e., a model that only outputs the majority class); moreover, it collapses to a neural network classifier when the tree consists of a single leaf. Fig. 9 (left) shows the schema for such hybrids.

4.2. Split hybridization

Similarly to [15], we design a split-level hybrid, that splits instances based on neural feature extractors. Recall the definition of feature extraction functions from Subsection 3.3, that is, functions of type $f : \bigcup_{d=2}^{N} \mathbb{R}^d \to \mathbb{R}$ acting on specific attributes A_i of the time series. This hybridization strategy requires training n single-attribute encoders $\overrightarrow{v}_1, \ldots, \overrightarrow{v}_n$ before decision tree training via autoencoding, with $\overrightarrow{v}_i : \bigcup_{d=2}^{N} \mathbb{R}^d \to \mathbb{R}^k$, for each $1 \leq i \leq n$. Recall that such encoders learn vector representations of arbitrary size k, and we can consider each vector value as a different scalar feature. Therefore, we obtain the following neural propositional letters:

$$\mathcal{P}_{neuro} \quad = \quad \{\overrightarrow{v}_i^{\,j}(A_i) \bowtie a \mid i \in [1,n],\, j \in [1,k],\, \bowtie \in \{<, \leq, =, \neq, \geq, >\},\, a \in \mathbb{R}\},$$

where $\overrightarrow{v}_i^{\,j}(A_i)$ is the value in the j-th component of the feature vector $\overrightarrow{v}_i(A_i)$.

It is immediate, now, to define neural decisions, essentially by replacing $\mathcal{P}$ with $\mathcal{P}_{neuro}$ in Definition 2.

Definition 10 *(Neural decisions).* Let $\mathcal{P}_{neuro}$ be a set of neural propositional letters. Then, the set of *(temporal) neural split-decisions* (or, simply, *neural decisions*):

$$\Lambda_{neuro} = \{p, \neg p, \langle \mathsf{X} \rangle p, [\mathsf{X}] \neg p \mid p \in \mathcal{P}_{neuro}, \mathsf{X} \in \mathbb{A}\}.$$

These decisions are usable in conjunction with standard propositional decisions.

Hybrid split-level TDTs are pure TDTs having a set of non-neural propositional letters and a set of neural propositional letters, as follows.

Definition 11 *(Split-level interval temporal decision tree hybrids).* Let Λ be a set of decisions, Λ_{neuro} a set of neural decisions, and $\mathbb{Y}$ a label space. Then, a *split-level (interval) temporal decision tree hybrid* is a structure:

$$\tau_{split} = (\mathcal{V}, \mathcal{E}, l, e),$$

where:

- $(\mathcal{V}, \mathcal{E})$ is a full directed binary tree,
- $l : \mathcal{V}^{\ell} \to \mathbb{Y}$ is a *leaf-labeling function* that assigns an element from $\mathbb{Y}$ to each leaf node in $\mathcal{V}^{\ell}$,
- $e : \mathcal{E} \to \Lambda_{neuro} \cup \Lambda$ is a *edge-labeling function* that assigns a (neural or non-neural) decision to each edge in $\mathcal{E}$,

such that $e(v, v') = \neg e(v, v'')$, for all $(v, v'), (v, v'') \in \mathcal{E}$.

Note that the remaining definitions for the pure case also apply to this model; furthermore, this hybrid collapses to the pure tree when $\Lambda_{neuro} = \emptyset$.

The proposed learning algorithm in this case involves, first, training n neural feature extractors, and then, using them to learn split-level TDT hybrids. Each feature extractor is trained via autoencoding on a specific attribute. Since each feature extractor is applied on any sub-interval of the series (which is, itself, a series), the n (single-attribute) datasets used for training the encoders are augmented with all of their sub-intervals; in this way, the models are trained to extract feature vectors from any sub-interval. A schema for such hybrids is shown in Fig. 9 (middle).

4.3. Root hybridization

A natural counterpart of the leaf hybridization requires performing some neural computation first, followed by some interpretable one. Elaborating on this idea, we can design a hybrid that performs a clustering step by means of uninterpretable computation, and delegates the final classification to one of many decision trees that are, in principle, more specific to the kind of instance at hand (similarly to [17]); we name this model *root-level hybrid*. As in the case of split hybridization, the neural clustering relies on a neural feature extractor that outputs a vector representation of size k. Since, as before, each of these values is a real number capturing a specific (but unknown) characteristic of the input instance (e.g., in the case of a time series, the level of irregularity), the feature vector can be used to label the instance as belonging to a cluster. In principle, a standard clustering algorithm can perform such a task. However, to avoid relying on non-neural techniques, we propose a different clustering policy: we weight the outputs of the k decision trees by applying the *softmax* function to the feature vector (of length k). Note that weighting the outputs of all trees, instead of delegating the computation to a single one of them (e.g., the one corresponding to the *maximum* of the softmaxed values) breaks the symmetry with the leaf-level hybrid, where, instead, only one of the neural networks is responsible for the final outcome. However, differently from a decision tree, where the input information travels compactly along a single path, the information in a neural network flows through the whole model resulting in a dense output vector, capturing unknown features of the input. With this in mind, to limit the final classification to a single model (focusing on one of these single characteristics) is an arbitrary choice, and may hamper the performance. Furthermore, while the hard clustering approach trains each tree on an average of m/k instances, this approach trains each tree on all m instances, leveraging the full extent of the data available.

We now define the hybrid root-level TDTs and their run.

Definition 12 *(Root-level interval temporal decision tree hybrids).* Let Λ be a set of decisions and $\mathbb{Y}$ a label space. Then, a *root-level (interval) temporal decision tree hybrid* is a structure:

$$\tau_{root} = (\hat{\upsilon}, \{\tau_1, \ldots, \tau_k\}),$$

where:

- $\hat{\upsilon}$ is a neural network-based clustering model,
- $\tau_1, \ldots, \tau_k$ are temporal decision trees on Λ and $\mathbb{Y}$.

Definition 13 *(Run of root-level temporal decision tree hybrids).* Let $\tau_{root} = (\hat{\upsilon}, \{\tau_1, \ldots, \tau_k\})$ be a root-level interval temporal decision tree hybrid, and x a time series. Then, the *run of* τ_{root} *on* x, denoted by $\tau_{root}(x)$, is defined as:

$$\tau_{root}(x) = \operatorname*{argmax}_{y} \{\Sigma_{\tau_i(x)=y} \hat{\upsilon}^i(x)\}.$$

The proposed learning algorithm starts by training a single encoder $\overrightarrow{v} : \mathbb{R}^{Nn} \to \mathbb{R}^k$ via autoencoding on the whole dataset; the neural root is, then, a network defined by $\hat{v}(x) = softmax(\overrightarrow{v}(x))$ Then, k decision trees are trained on the same dataset, but with different weights for each instance; more specifically, when training the i-th decision tree, each time series x is weighted by $\hat{v}^i(x)$, that is, the value of the i-th component of the vector $\hat{v}(x)$. This hybrid collapses to the pure case when $k = 1$. The schema for such hybrids is shown in Fig. 9 (right).

5. Experimental results

To assess the performances of hybrid temporal decision trees, we carried out several experiments in two batches. In the first set, we considered the task of diagnosing COVID-19 from audio samples of coughs and breaths. The tasks have been posed as binary, multivariate time series classification problems [18,19], and both temporal decision trees [11] and neural networks [20–24] have proven their efficacy. As a by-product of these experiments, we learned which hybrid model seems to work best; we used this information to organize a second set of experiments, with a similar dataset for diagnosing various respiratory diseases. The dataset is the Respiratory Sound Database [25], which contains audio recordings of respiratory cycles from healthy and non-healthy subjects. In this second case, we limited ourselves to consider the best hybrid combination.

5.1. Experiment 1: choosing the best hybrid model

5.1.1. Data

The data used for these experiments was originally crowdsourced by researchers at the University of Cambridge [18], and includes 9986 labeled audio samples recorded by 6613 volunteers. Most of the cough and breath recordings contained several episodes, thus a semi-automated segmentation of the audio samples was performed, deriving a pool of audio samples containing single coughs and single breaths. Following the original paper, from this pool, three binary classification tasks were designed, resulting in three cough classification datasets (*C1*, *C2*, *C3*), and three breath classification datasets (*B1*, *B2*, *B3*). The tasks were:

1. to distinguish between declared positive subjects, and negative ones that have a clean medical history, have never smoked, and have no symptoms;
2. to distinguish between declared positive subjects with cough as symptom, and negative ones that have a clean medical history, have never smoked, and have cough as a symptom;
3. to distinguish between declared positive subjects with cough as symptom, and negative ones that have asthma, have never smoked, and have cough as a symptom.

To counteract the small amount of negative instances for tasks C2, B2, C3, and B3, audio augmentation methods were used in both.

Different techniques were used to clean and normalize each audio, including *noise gate*, *peak normalization*, *silence removal*, *pitch normalization*. Each audio sample was, then, processed using a variant of the *Mel-Frequency Cepstral Coefficients* (*MFCC*) [68], a widespread technique for extracting *spectral* representations of sounds, facilitating their interpretation in terms of audio frequencies. Our MFCC variant produces, for each sample, a multivariate time series representation where the $n = 15$ attributes describe the power of the different sound frequencies. Finally, moving average filters were applied, reducing the number of time points and the computational load; a moving average of length 10 and step 7 was used for cough tasks, and one of length 75 and step 60 was used for breath tasks. A more detailed description of both the data and the data processing steps can be found in [11]. Note that, in order to preserve interpretability of the results, the data processing steps, here, slightly differ from those of the original work; as a result, the six resulting datasets only serve for an internal performance comparison, as well as for benchmarking the hybrid models.

5.1.2. Experimental setting

For each classification task, the experiments were conducted in a randomized cross-validation setting with 10 repetitions, where the validation sets are roughly 4 times larger than the training sets (i.e., a standard 80%-20% train-validation splitting policy), and both the training and validation sets *(1)* are balanced with respect to the two classes *(2)* are such that if one has an instance, the other never has its augmented versions (thus preventing data leakage) *(3)* never exceed a total of 200 instances. Table 1 reports some specifications for the six datasets, including the length of the time series (N), the number of positive and negative instances, and the number of instances in the training and validation sets at each repetition. Five models were compared: a (pure) temporal decision tree, a neural network classifier, and a leaf-, a split-, and a root-level hybrid; for a fair comparison, the classifier and the hybrids are based on the same neural technology, and, where possible, on the same architecture. After an exploratory analysis in which different tree-pruning conditions were tested, Shannon Entropy as information metric and a minimum entropy gain of 0.01 were fixed for the pure tree and the trees in the three hybrids. Additionally, a minimum number of instances at the leaf nodes of 2 was chosen for the pure tree and for the split-level hybrid, of 8 for the leaf-level hybrid, and of 16 for the root-level hybrid. For all tree models, except for the split-level hybrid, where both neural and non-neural features were used, the feature extraction functions are fixed to be

Table 1
Specifications for the six binary classification tasks from the dataset in use [18]. For each dataset, the length of the series in the original dataset (N_{orig}), the length of the series after the moving average filter (N), the number of positive and negative instances is shown, together with the total number of instances, and the number of training and validation instances used for cross-validation. Recall that $n = 15$ for all datasets.

	Task	N_{orig}	N	# pos	# neg	# tot	# train/val
Cough	C1	59	7	202	223	425	158/40
	C2	59	7	53	187	240	84/22
	C3	51	6	53	55	108	64/16
Breath	B1	604	9	570	618	1188	160/40
	B2	366	5	182	482	664	158/40
	B3	361	5	182	77	259	80/20

Table 2
Search space for the prior hyperparameter search the three architectures used in the leaf-, split-, and root-level hybrid models. The optimal hyperparametrization found by random search is marked in bold, and its average validation accuracy/MAE is reported. For the sequence-to-sequence models the measure is computed on the normalized values.

	CLS (leaf-level)	S2S (split-level)	S2S (root-level)
Hidden size	32, **64**, 128, 256	64, **128**, 256, 512	32, 64, **128**, 256
Dropout rate	**0.5**, 0.6, 0.7	**0.5**, 0.6, 0.7	**0.5**, 0.6, 0.7
# Epochs	100, **200**	**100**, 200	100, **200**
Batch size	16, 32, **64**, 128, 256	32, 64, **128**, 256	32, **64**, 128, 256
Patience	20, **50**, 100	20, **50**, 100	20, 50, **100**
Code size	–	1, **2**, 4, 8	2, **4**, 8, 16
Learning rate	10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}	10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}	10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}
Val. ACC (%)	63.0	–	–
Val. MAE	–	0.028	0.2956

$\mathcal{F} = \{\min, \max\}$, and the first split is forced to use a decision of the kind $\langle \mathsf{G} \rangle$, inducing the learning of global leaf- and class-formulas.

A GRU-based network classifier (referred to as *CLS*) was used for the leaf-level hybrid, and GRU-based sequence-to-sequence networks (referred to as *S2Ss*) [69,70] were used as feature extractors for the root-level and split-level hybrid. The CLS architecture consists of three GRU layers, plus the final linear layer for producing a classification output. The S2S architecture is an autoencoder composed of three GRU layers for both the encoder and the decoder. In all cases, each GRU layer is followed by a dropout layer [71], for counteracting overfitting. All architectures were trained using batch gradient descent, optimizing the cross-entropy and the mean average error (*MAE*) for the cases of classifiers and autoencoders, respectively. Note that, given that the architecture of the underlying networks is comparable, the three hybrids are, in principle, comparable.

Neural networks were trained via the AdamW [67] algorithm. For each of the three hybrids, a prior hyperparameter search and selection was performed using the dataset for C1. This was done via a randomized, balanced cross-validation with 5 repetitions, after selecting a balanced test set (60 instances) for validating the final hyperparametrization. For each repetition, the models were trained and validated on balanced sets of 293 and 72 instances, respectively; the validation sets were used for the early-stopping condition, which stops the training when the validation loss does not decrease for a specified number of epochs (this hyperparameter, in neural network terminology, is referred to as *patience*). Other parameters included in the search were the *learning rate*, *dropout rate*, and the *hidden size* of the hidden layers of each GRU layer. For S2S models, the size of the hidden state (*code size*) was also included in the search. Ultimately, each of the three best hyperparametrizations was found by 50 tries of random search, optimizing the average validation accuracy (*ACC*) for the case of CLS models and MAE for the case of S2S models. Table 2 reports, for the three cases, the parameter space, the best hyperparametrization across the 50 tries, and its average accuracy or MAE. For the case of the leaf-level hybrid, a similar search was conducted on the hyperparametrizations for the hierarchical fine-tuning policy; several leaf-level hybrid models were trained with different hyperparametrizations and, ultimately, a batch size equal to 8, a number of fine-tuning epochs equal to 20, a learning rate equal to 0.0001, and no early-stopping condition were fixed. Since leaf hybridization generalizes neural network classifiers, the same hyperparametrization found for the leaf-level hybrid was used for the neural network classifier.

All experiments were done using open-source Julia packages: Flux.jl [72] for training neural networks, ModalDecisionTrees.jl [73] for training decision trees, and Hyperopt.jl [74] for the hyperparameter search.

5.1.3. Results and discussion

Table 3 gives an overview of the cross-validation results for the five models. The results are given in terms of average and standard deviations across the 10 repetitions of a measure of performance and a measure of symbolic complexity of

Table 3
Average cross-validation accuracy (percentage points) and number of leaves (i.e. classification rules) for the models in comparison. The table reports the means and standard deviations over 10 repetitions and, for each task, the average accuracies of the hybrid models that perform better than the neural classifier (CLS) and the pure temporal decision tree (TDT) are shown in bold and underlined, respectively. For the case of the root-level hybrid, the number of rules reported is the total number of rules in the underlying trees.

Model			C1		C2		C3		B1		B2		B3		Average	
			ACC	$\mathcal{L}$	ACC	$\mathcal{L}$	ACC	$\mathcal{L}$	ACC	$\mathcal{L}$	ACC	$\mathcal{L}$	ACC	$\mathcal{L}$	ACC	$\mathcal{L}$
	TDT	avg	66.5	10.0	92.7	2.1	93.8	2.9	64.5	15.6	81.0	7.4	87.5	4.0	81.0	7.0
		std	8.6	4.2	7.2	0.3	7.8	1.2	7.3	6.1	8.1	1.9	8.6	0.7	7.9	2.4
Hybrids	Leaf-level	avg	55.0	4.3	55.0	2.0	71.2	2.4	53.6	5.3	56.5	6.0	64.5	3.1	59.3	3.8
		std	3.3	1.3	5.1	0.0	11.4	0.8	6.2	2.2	3.8	1.9	7.4	1.1	6.2	1.2
	Split-level	avg	**66.5**	9.0	**89.9**	3.6	**94.8**	2.7	**64.4**	10.8	**81.3**	4.5	**90.0**	4.6	**81.2**	5.9
		std	7.4	4.4	10.8	2.1	5.3	0.8	6.9	8.2	9.2	2.6	5.8	0.8	7.6	3.2
	Root-level	avg	**69.1**	44.9	**91.6**	5.4	**93.8**	22.0	**68.6**	55.2	**80.5**	16.0	**84.6**	7.0	**81.4**	25.1
		std	9.4	7.0	7.7	1.8	9.0	3.2	10.2	13.7	7.9	2.8	7.4	2.8	8.6	5.2
	CLS	avg	63.5	–	66.1	–	85.3	–	56.6	–	67.5	–	75.8	–	69.1	–
		std	7.8	–	5.9	–	4.2	–	3.7	–	4.4	–	7.6	–	5.6	–

the model. The performance is evaluated via classification accuracy (ACC), while the complexity is evaluated via the number of classification rules (which is equivalent to the number of tree leaves), denoted by $\mathcal{L}$.

Both with cough and breath samples, all models scored a higher average accuracy for task 2 and task 3, than task 1, which seems to enclose the hardest problem. Furthermore, the accuracy for each cough task, with respect to the corresponding breath task, is higher by 2 to 11.7 percentage points. Across all tasks, the leaf-level hybrid is the one that performs worst (average accuracy 59.3%), followed by the pure neural network classifier (average accuracy 69.1%). The three remaining models, namely, the pure temporal tree, the split-level hybrid and the root-level hybrid have similar accuracies (81.0%–81.4% on average across the six tasks). In all cases, both hybrids outperform the neural classifier, and in six out of twelve cases they yield a higher average accuracy with respect to the temporal decision tree; the remaining cases are evenly distributed across the six tasks, with the exception of C2, where the pure temporal tree have higher accuracy than all other models. Of the two hybrids, the root-level hybrid appears to lead to larger improvements, bringing the average accuracy from 66.5% to 69.1% for C1, and from 64.5% to 68.6% for B1. Note how for the six tasks the pure tree outperforms the neural network classifier, with accuracy gaps ranging from 3.0 (C1) to 26.6 (C2) percentage points. In principle, this is likely to put hybrid models at a disadvantage, when compared to the pure tree, but, overall, the results show that in five out of eighteen cases, hybridizing a tree with a weaker neural technology still brings an improvement.

Of the three hybridization schemes, the leaf-level hybrid is the one that appear to consistently cause a degradation in performances; this could be due to the small size of the datasets used for initial training and/or fine-tuning, and an underlying difficulty of calibrating the parameters for the initial training and those for the fine-tuning stages. Note that a small size of the datasets probably does not affect as much the performances of the other two hybrids. In fact, they do not require a fine-tuning stage, and, in particular, the split-level hybrid augments the number of instances by as much as $N \cdot (N-1)/2$ by considering all sub-intervals. Leaf hybridization is also the one of the three schemes with stronger similarities with methods in the existing literature [16]; however, these methods use, at the leaves, models that require less training data than GRUs (e.g., MLP or linear regression). This suggests that a network with fewer GRU layers, or with a simpler RNN technology could yield better results. Finally, the fact that split hybridization performs better than the other schemes could be due to the nature of temporal decision trees, that allow the exploration of every interval of every instance, as well as the learning algorithm allowing training on a larger dataset.

As for the symbolic complexity of the models, leaf-level hybridization provides models that are in five out of six cases smaller than the purely symbolic counterpart; split-level hybridization provides models that are in four out of six cases smaller than their purely symbolic counterpart. As the root-level hybrids have four trees, they always yield larger models in terms of number of leaves with respect to temporal trees; however, in four out of six cases the average number of leaves across the four trees is smaller than that of temporal trees and, in fact, root-level hybrids appear to hold the smallest trees. This trend is consistent with the values chosen for the minimum number of instances at the tree leaves, which, recall, are $8, 2, 16, 2$ for the leaf-level hybrid, split-level hybrid, root-level hybrid, and pure tree, respectively.

Compared with the leaf- and root-level hybridization schemes, the split-level hybridization scheme appears to capture a good trade-off between performance gain and symbolic complexity. To further investigate this trade-off, we now compare rules extracted via the purely symbolic approach, and split-level hybridization scheme. Recall from Table 1, that tasks C1, B1, and B2 are the ones with the highest number of instances, and that their (balanced) validation sets count 20 instances for each of the two classes *pos* and *neg*, respectively. We, first, compared trees trained on these three tasks, and assessed how neural features would only appear at lower levels of the trees.

Table 4 reports a few pairs of purely symbolic and neuro-symbolic rules extracted from trees trained on the three tasks, along with their validation confidences and supports. Each pair shows an example where, under the same conditions, the learning of a pure tree incurred in stopping criteria, whereas the learning of a split-level hybrid was able to further refine a rule using neural features from autoencoders. As a result, compared with their purely symbolic counterparts, hybrid rules have higher confidences, but lower supports. With the selected rules, the degradation in terms of support ranges between

Table 4
Pairs of purely symbolic and split-level hybrid rules extracted from trees trained on tasks C1, B1, and B2. For each rule, confidence (*conf*) and support (*supp*) are shown.

Task	Rule			*conf*	*supp*
C1	$[G](min(A_7) < 1.06 \times 10^4) \wedge \langle G \rangle (min(A_6) \geq 3.92 \times 10^2)$	$\Rightarrow$	*neg*	0.56	0.40
	$[G](min(A_7) < 1.06 \times 10^4) \wedge \langle G \rangle (min(A_6) \geq 3.92 \times 10^2) \wedge$ $[G](min(A_6) \geq 3.92 \times 10^2 \rightarrow (\overrightarrow{v}_{10}(A_{10}) < 3.72 \times 10^{-4} \wedge$ $\langle O \rangle (max(A_9) \geq 4.22 \times 10^4)))$	$\Rightarrow$	*neg*	0.75	0.30
C1	$[G](min(A_9) < 8.35 \times 10^4)$	$\Rightarrow$	*neg*	0.79	0.48
	$[G](min(A_9) < 8.35 \times 10^4) \wedge [G](\overrightarrow{v}_7(A_7) \geq -5.67 \times 10^{-5}) \wedge$ $\langle G \rangle (min(A_7) \geq 2.54 \times 10^3)$	$\Rightarrow$	*neg*	1.00	0.30
B1	$[G](min(A_8) < 5.40 \times 10^3)$	$\Rightarrow$	*neg*	0.62	0.60
	$[G](min(A_8) < 5.40 \times 10^3 \wedge \overrightarrow{v}_{13}(A_{13}) \geq 3.54 \times 10^{-5} \wedge$ $min(A_{11}) < 8.72 \times 10^4)$	$\Rightarrow$	*neg*	0.73	0.38
B1	$\langle G \rangle (min(A_8) \geq 1.88 \times 10^3 \wedge min(A_4) < 1.05 \times 10^3)$	$\Rightarrow$	*pos*	0.64	0.62
	$\langle G \rangle (min(A_8) \geq 1.88 \times 10^3 \wedge min(A_4) < 1.05 \times 10^3) \wedge$ $[G](\overrightarrow{v}_4(A_4) \geq -6.04 \times 10^{-5}) \wedge$ $[G]((min(A_8) \geq 1.88 \times 10^3 \wedge min(A_4) < 1.05 \times 10^3) \rightarrow$ $[L](\overrightarrow{v}_{14}(A_{14}) < 5.62 \times 10^{-5}) \wedge$ $[\overline{D}](\overrightarrow{v}_{15}(A_{15}) < -6.30 \times 10^{-5}))$	$\Rightarrow$	*pos*	0.89	0.22
B2	$[G](min(A_9) < 2.87 \times 10^4) \wedge$ $\langle G \rangle (min(A_4) \geq 1.66 \wedge min(A_{11}) \geq 3.52 \times 10^4)$	$\Rightarrow$	*neg*	0.88	0.20
	$[G](min(A_9) < 2.87 \times 10^4) \wedge$ $\langle G \rangle (min(A_4) \geq 1.66 \wedge min(A_{11}) \geq 3.52 \times 10^4) \wedge$ $[G]((min(A_4) \geq 1.66 \wedge min(A_{11}) \geq 3.52 \times 10^4) \rightarrow$ $\overrightarrow{v}_3(A_3) \geq 2.81 \times 10^{-4})$	$\Rightarrow$	*neg*	1.00	0.18

Table 5
Split-level hybrid rules extracted from trees trained on the eight tasks of Respiratory Sound Database. For each rule, confidence (*conf*) and support (*supp*) are shown.

Task	Rule			*conf*	*supp*
CO/P	$\langle G \rangle (min(A_9) \geq 6.18 \times 10^2)$	$\Rightarrow$	*neg*	0.95	0.48
	$[G](min(A_9) < 6.18 \times 10^2 \wedge min(A_5) < 2.48 \times 10^2)$	$\Rightarrow$	*pos*	0.91	0.28
CO/T	$\langle G \rangle (min(A_6) \geq 2.62 \times 10^2 \wedge \langle L \rangle \overrightarrow{v}_{14}(A_{14}) \geq -2.59 \times 10^{-2})$	$\Rightarrow$	*neg*	0.91	0.42
	$[G](min(A_6) < 2.62 \times 10^2 \wedge min(A_{13}) < 3.42 \times 10^4 \wedge \overrightarrow{v}_3(A_3) \geq 5.84 \times 10^{-3})$	$\Rightarrow$	*pos*	0.92	0.46
UR/P	$\langle G \rangle (min(A_{11}) \geq 7.96 \times 10^1 \wedge min(A_{15}) < 8.17 \times 10^2 \wedge \overrightarrow{v}_3(A_3) \geq 1.04 \times 10^{-1})$	$\Rightarrow$	*neg*	0.83	0.43
	$\langle G \rangle (min(A_{11}) \geq 7.96 \times 10^1) \wedge$			0.75	0.43
	$[G](min(A_{11}) < 7.96 \times 10^1 \rightarrow (min(A_{15}) \geq 8.17 \times 10^2 \wedge \langle G \rangle \overrightarrow{v}_5(A_5) \geq 3.30 \times 10^{-2}))$	$\Rightarrow$	*pos*		
UR/T	$\langle G \rangle (min(A_1) \geq 3.04 \times 10^2 \wedge \overrightarrow{v}_1(A_1) < 1.52 \times 10^{-2})$	$\Rightarrow$	*pos*	0.83	0.67
	$[G](min(A_1) < 3.04 \times 10^2 \wedge \overrightarrow{v}_{13}(A_{13}) \geq 2.55 \times 10^{-2}) \wedge \langle G \rangle min(A_9) \geq 5.34 \times 10^3$	$\Rightarrow$	*neg*	1.00	0.33
BR/P	$\langle G \rangle (min(A_{10}) \geq 3.14 \times 10^2)$	$\Rightarrow$	*neg*	1.00	0.50
	$[G](min(A_{10}) < 3.14 \times 10^2)$	$\Rightarrow$	*pos*	1.00	0.50
BR/T	$\langle G \rangle (min(A_5) \geq 3.42 \times 10^2)$	$\Rightarrow$	*neg*	1.00	0.50
	$[G](min(A_5) < 3.42 \times 10^2)$	$\Rightarrow$	*pos*	1.00	0.50
PN/P	$\langle G \rangle (min(A_3) \geq 3.80 \times 10^2)$	$\Rightarrow$	*pos*	1.00	0.18
	$[G](min(A_3) < 3.80 \times 10^2 \wedge \overrightarrow{v}_8(A_8) \geq 4.48 \times 10^{-3} \wedge min(A_1) < 5.79 \times 10^2) \wedge$			0.77	0.34
	$\langle G \rangle (min(A_3) \geq 2.15 \times 10^2 \wedge \langle O \rangle max(A_5) < 7.08 \times 10^1)$	$\Rightarrow$	*pos*		
	$[G](min(A_3) < 3.80 \times 10^2 \wedge \overrightarrow{v}_8(A_8) \geq 4.48 \times 10^{-3} \wedge min(A_1) < 5.79 \times 10^2) \wedge$			0.82	0.29
	$\langle G \rangle (min(A_3) \geq 2.15 \times 10^2) \wedge [G](min(A_3) \geq 2.15 \times 10^2 \rightarrow [O](max(A_5) \geq 7.08 \times 10^1))$	$\Rightarrow$	*neg*		
PN/T	$\langle G \rangle (min(A_{12}) \geq 1.74 \times 10^3)$	$\Rightarrow$	*yes*	0.80	0.38
	$[G](min(A_{12}) < 1.74 \times 10^3 \wedge \overrightarrow{v}_2(A_2) \geq 1.20 \times 10^{-3}) \wedge \langle G \rangle (min(A_{14}) \geq 1.81 \times 10^2)$	$\Rightarrow$	*neg*	0.77	0.50

2 (B2) and 40 (B1, second pair) percentage points, with an average of 18.40 percentage points. In turn, the gain in terms of confidence ranges between 11 (B2, first pair) and 25 (B2, second pair) percentage points, and with an average of 20 percentage points. One possible reason for this is that by learning neuro-symbolic rules we have the chance of exploring the solution space in more detail before reaching a stopping condition.

5.2. Experiment 2: rule extraction for respiratory diseases

As a final set of experiments, we tested the same approach with the same parametrization on a different, similar dataset for diagnosing various respiratory diseases. Taking advantage from the previous experiment, we focused on split hybridization, and we limited ourselves to rule extraction.

5.2.1. Data and experimental setting

The dataset is the Respiratory Sound Database [25], which contains audio recordings of respiratory cycles from healthy and non-healthy subjects. Similarly to before, we designed a set of eight binary classification tasks by varying the respiratory disease of interest in: COPD (*CO*), URTI (*UR*), Bronchiectasis (*BR*), and Pneumonia (*PN*); and the position of the audio sensor in: Trachea (*T*) and Posterior (*P*). Using the provided labels, we segmented the recordings into recordings containing single respiratory cycles; the number of instances in the resulting eight datasets ranges between 52 and 200, with an average of 124.75. We trained trees using the same randomized, balanced cross-validation setting, but this time with 4 repetitions; as a result, the test sets count a number of instances between 10 and 40, with an average of 25.

5.2.2. Results

The purpose of this experiment is not that of obtaining classification models with the best possible absolute performances. The obtained models, however, are relatively competitive: the resulting average accuracies were 68.3% for UR, 94.0% for BR, 87.5% for CO, and 82.6% for PN. On the other hand, several useful rules could be extracted from the learned models; they are reported in Table 5.

6. Conclusions

In this paper we have presented a method for multivariate time series classification that combines the high generalization capacity of trained neural networks with the symbolic nature of temporal decision trees. Our experiments showed promising results, and allowed us to draw some initial conclusions. The hybridization between temporal decision trees and neural networks seems quite natural and elegant, by considering the different type of nodes in decision trees. The best results seem to arise when the hybridization scheme blends the symbolic and neural computation in the root and split/internal nodes. Moreover, the root-level hybridization technique is similar to random forests, where the individual decisions are weighted by a final decision making policy; in our case, such decision policy is governed by a neural network at the root node of a (temporal) decision tree, whereas in random forests, the policy is at the leaf nodes. As a final commentary, this work could represent a first step towards using temporal decision trees for explaining or approximating recurrent neural network classifiers.

CRediT authorship contribution statement

Giovanni Pagliarini: Writing – original draft, Validation, Investigation, Data curation. **Simone Scaboro:** Investigation, Data curation. **Giuseppe Serra:** Supervision. **Guido Sciavicco:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation. **Ionel Eduard Stan:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ionel Eduard Stan reports financial support was provided by Francesco Severi National Institute of Higher Mathematics National Group of Scientific Calculations. Ionel Eduard Stan reports financial support was provided by University of Ferrara. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been partially funded by the FIRD project *Modal Geometric Symbolic Learning* (University of Ferrara), the Italian Ministry of University and Research through PNRR - M4C2 - Investimento 1.3 (Decreto Direttoriale MUR n. 341 del 15/03/2022), Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 8 "Pervasive AI", funded by the European Union under the NextGeneration EU programme", and the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (INDAM-GNCS) project *Symbolic and Numerical Analysis of Cyberphysical Systems*, CUP code E53C22001930001. Guido Sciavicco and Ionel Eduard Stan are GNCS-INdAM members.

References

[1] A.P. Ruiz, M. Flynn, J. Large, M. Middlehurst, A.J. Bagnall, The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Min. Knowl. Discov. 35 (2) (2021) 401–449.

[2] R. Shwartz-Ziv, A. Armon, Tabular data: deep learning is not all you need, Inf. Fusion 81 (2022) 84–90.
[3] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data?, Adv. Neural Inf. Process. Syst. 35 (2022) 507–520.
[4] B. Goodman, S. Flaxman, European Union regulations on algorithmic decision-making and a "right to explanation", AI Mag. 38 (3) (2017) 50–57.
[5] A.S. d'Avila Garcez, M. Gori, L.C. Lamb, L. Serafini, M. Spranger, S.N. Tran, Neural-symbolic computing: an effective methodology for principled integration of machine learning and reasoning, J. Appl. Log. 6 (4) (2019) 611–632.
[6] A.S. d'Avila Garcez, L.C. Lamb, D.M. Gabbay, Neural-Symbolic Cognitive Reasoning, Cognitive Technologies, Springer, 2009.
[7] M. Minsky, Logical versus analogical or symbolic versus connectionist or neat versus scruffy, AI Mag. 12 (2) (1991) 34–51.
[8] G. Sciavicco, I.E. Stan, Knowledge extraction with interval temporal logic decision trees, in: Proc. of the 27th International Symposium on Temporal Representation and Reasoning (TIME), in: LIPIcs, vol. 178, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 9.
[9] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth Publishing Company, 1984.
[10] F. Manzella, G. Pagliarini, G. Sciavicco, I.E. Stan, Interval temporal random forests with an application to COVID-19 diagnosis, in: Proc. of the 28th International Symposium on Temporal Representation and Reasoning (TIME), in: LIPIcs, vol. 206, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 7.
[11] F. Manzella, G. Pagliarini, G. Sciavicco, I.E. Stan, The voice of COVID-19: breath and cough recording classification with temporal decision trees and random forests, Artif. Intell. Med. 137 (2023) 102486.
[12] M. Coccagna, F. Manzella, S. Mazzacane, G. Pagliarini, G. Sciavicco, Statistical and symbolic neuroaesthetics rules extraction from EEG signals, in: Proc. of the 9th International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC), in: Lecture Notes in Computer Science, vol. 13258, Springer, 2022, pp. 536–546.
[13] G. Bechini, E. Losi, L. Manservigi, G. Pagliarini, G. Sciavicco, I.E. Stan, M. Venturini, Statistical rule extraction for gas turbine trip prediction, J. Eng. Gas Turbines Power 145 (2023) 1–10.
[14] K. Cho, B. van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: encoder–decoder approaches, in: Proc. of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST), Association for Computational Linguistics, 2014, pp. 103–111.
[15] H. Guo, S.B. Gelfand, Classification trees with neural network feature extraction, IEEE Trans. Neural Netw. 3 (6) (1992) 923–933.
[16] Z. Zhou, Z. Chen, Hybrid decision tree, Knowl.-Based Syst. 15 (8) (2002) 515–528.
[17] A. Wan, L. Dunlap, D. Ho, J. Yin, S. Lee, S. Petryk, S.A. Bargal, J.E. Gonzalez, NBDT: neural-backed decision tree, in: Proc. of the 9th International Conference on Learning Representations (ICLR), 2021, pp. 1–12.
[18] C. Brown, J. Chauhan, A. Grammenos, J. Han, A. Hasthanasombat, D. Spathis, T. Xia, P. Cicuta, C. Mascolo, Exploring automatic diagnosis of COVID-19 from crowdsourced respiratory sound data, in: Proc. of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2020, pp. 3474–3484.
[19] J. Shuja, E. Alanazi, W. Alasmary, A. Alashaikh, COVID-19 open source data sets: a comprehensive survey, Appl. Intell. 51 (3) (2021) 1296–1325.
[20] A. Hassan, I. Shahin, M.B. Alsabek, COVID-19 detection system using recurrent neural networks, in: Proc. of the 2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI), 2020, pp. 1–5.
[21] M. Pahar, M. Klopper, R. Warren, T. Niesler, COVID-19 cough classification using machine learning and global smartphone recordings, Comput. Biol. Med. 135 (2021) 104572.
[22] G. Deshpande, B.W. Schuller, The DiCOVA 2021 challenge – an encoder-decoder approach for COVID-19 recognition from coughing audio, in: Proc. of the 22nd Annual Conference of the International Speech Communication Association (INTERSPEECH), 2021, pp. 931–935.
[23] M. Alkhodari, A.H. Khandoker, Detection of COVID-19 in smartphone-based breathing recordings: a pre-screening deep learning tool, PLoS ONE 17 (1) (2022) 1–25.
[24] A.B. Nassif, I. Shahin, M. Bader, A. Hassan, N. Werghi, COVID-19 detection systems using deep-learning algorithms based on speech and image data, Mathematics 10 (4) (2022) 564.
[25] B.M. Rocha, D. Filos, L. Mendes, I. Vogiatzis, E. Perantoni, E. Kaimakamis, P. Natsiavas, A. Oliveira, C. Jácome, A. Marques, A respiratory sound database for the development of automated classification, in: Proc. of the 3rd Precision Medicine Powered by pHealth and Connected Health (ICBHI), Springer, 2018, pp. 33–37.
[26] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawai, R. Marks, A performance comparison of trained multilayer perceptrons and trained classification trees, in: Proc. of the IEEE International Conference on Systems, Man and Cybernetics (SMC), vol. 78, 1990, pp. 1614–1619.
[27] J.W. Shavlik, R.J. Mooney, G.G. Towell, Symbolic and neural learning algorithms: an experimental comparison, Mach. Learn. 6 (1991) 111–143.
[28] I.K. Sethi, Entropy nets: from decision trees to neural networks, Proc. IEEE 78 (10) (1990) 1605–1613.
[29] R.P. Brent, Fast training algorithms for multilayer neural nets, IEEE Trans. Neural Netw. 2 (3) (1991) 346–354.
[30] I. Ivanova, M. Kubat, Initialization of neural networks by means of decision trees, Knowl.-Based Syst. 8 (6) (1995) 333–344.
[31] R. Setiono, W.K. Leow, On mapping decision trees and neural networks, Knowl.-Based Syst. 12 (3) (1999) 95–99.
[32] M. Kubat, Decision trees can initialize radial-basis function networks, IEEE Trans. Neural Netw. 9 (5) (1998) 813–821.
[33] M.W. Craven, J.W. Shavlik, Extracting tree-structured representations of trained networks, in: Proc. of the 8th Advances in Neural Information Processing Systems (NIPS), 1995, pp. 24–30.
[34] G.G. Towell, J.W. Shavlik, Extracting refined rules from knowledge-based neural networks, Mach. Learn. 13 (1993) 71–101.
[35] D. Dancey, D. McLean, Z. Bandar, Decision tree extraction from trained neural networks, in: Proc. of the 7th International Florida Artificial Intelligence Research Society Conference (FLAIRS), 2004, pp. 515–519.
[36] R. Krishnan, G. Sivakumar, P. Bhattacharya, Extracting decision trees from trained neural networks, Pattern Recognit. 32 (12) (1999) 1999–2009.
[37] G.P.J. Schmitz, C. Aldrich, F.S. Gouws, ANN-DT: an algorithm for extraction of decision trees from artificial neural networks, IEEE Trans. Neural Netw. 10 (6) (1999) 1392–1401.
[38] Z. Zhou, Y. Jiang, NeC4.5: neural ensemble based C4.5, IEEE Trans. Knowl. Data Eng. 16 (6) (2004) 770–773.
[39] R. Setiono, H. Liu, A connectionist approach to generating oblique decision trees, IEEE Trans. Syst. Man Cybern., Part B, Cybern. 29 (3) (1999) 440–444.
[40] V.N. Murthy, V. Singh, T. Chen, R. Manmatha, D. Comaniciu, Deep decision network for multi-class image classification, in: Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2240–2248.
[41] M. Längkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, Pattern Recognit. Lett. 42 (2014) 11–24.
[42] A.J. Bagnall, J. Lines, A. Bostrom, J. Large, E.J. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Min. Knowl. Discov. 31 (3) (2017) 606–660.
[43] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P. Muller, Deep learning for time series classification: a review, Data Min. Knowl. Discov. 33 (4) (2019) 917–963.
[44] V. Goranko, A. Montanari, G. Sciavicco, A road map of interval temporal logics and duration calculi, J. Appl. Non-Class. Log. 14 (1–2) (2004) 9–54.
[45] J.Y. Halpern, Y. Shoham, A propositional modal logic of time intervals, J. ACM 38 (4) (1991) 935–962.
[46] D. Bresolin, D. Della Monica, A. Montanari, P. Sala, G. Sciavicco, Interval temporal logics over strongly discrete linear orders: expressiveness and complexity, Theor. Comput. Sci. 560 (2014) 269–291.

[47] D. Bresolin, D.D. Monica, A. Montanari, P. Sala, G. Sciavicco, Decidability and complexity of the fragments of the modal logic of Allen's relations over the rationals, Inf. Comput. 266 (2019) 97–125.
[48] J. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.
[49] A. Montanari, G. Sciavicco, N. Vitacolonna, Decidability of interval temporal logics over split-frames via granularity, in: Proc. of the 8th European Conference on Logics in Artificial Intelligence (ECAI), in: Lecture Notes in Computer Science, vol. 2424, Springer, 2002, pp. 259–270.
[50] L. Aceto, D. Della Monica, V. Goranko, A. Ingólfsdóttir, A. Montanari, G. Sciavicco, A complete classification of the expressiveness of interval logics of Allen's relations: the general and the dense cases, Acta Inform. 53 (3) (2016) 207–246.
[51] D. Bresolin, A. Kurucz, E. Muñoz-Velasco, V. Ryzhikov, G. Sciavicco, M. Zakharyaschev, Horn fragments of the Halpern-Shoham interval temporal logic, ACM Trans. Comput. Log. 18 (3) (2017) 22:1–22:39.
[52] E. Muñoz-Velasco, M. Pelegrín-García, P. Sala, G. Sciavicco, I.E. Stan, On coarser interval temporal logics, Artif. Intell. 266 (2019) 1–26.
[53] G. Bombara, C.I. Vasile, F. Penedo, H. Yasuoka, C. Belta, A decision tree approach to data classification using signal temporal logic, in: A. Abate, G.E. Fainekos (Eds.), Proc. of the 19th International Conference on Hybrid Systems: Computation and Control, ACM, 2016, pp. 1–10.
[54] D. Neider, I. Gavran, Learning linear temporal properties, in: N.S. Bjørner, A. Gurfinkel (Eds.), Proc. of Formal Methods in Computer Aided Design (FMCAD), IEEE, 2018, pp. 1–10.
[55] C. Lubba, S. Sethi, P. Knaute, S. Schultz, B. Fulcher, N. Jones, catch22: CAnonical Time-series CHaracteristics - selected through highly comparative time-series analysis, Data Min. Knowl. Discov. 33 (6) (2019) 1821–1852.
[56] L. Hyafil, R.L. Rivest, Constructing optimal binary decision trees is NP-complete, Inf. Process. Lett. 5 (1) (1976) 15–17.
[57] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.
[58] D. Della Monica, G. Pagliarini, G. Sciavicco, I.E. Stan, Decision trees with a modal flavor, in: Proc. of the 21st International Conference of the Italian Association for Artificial Intelligence (AIxIA), in: Lecture Notes in Computer Science, vol. 13796, Springer, 2022, pp. 47–59.
[59] C.E. Shannon, A mathematical theory of communication, Bell Syst. Tech. J. 27 (3) (1948) 379–423.
[60] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[61] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[62] J.H. Friedman, B.E. Popescu, Predictive learning via rule ensembles, Ann. Appl. Stat. 2 (3) (2008).
[63] N. Meinshausen, Node harvest, Ann. Appl. Stat. 4 (4) (2010).
[64] H. Deng, Interpreting tree ensembles with inTrees, Int. J. Data Sci. Anal. 7 (2019) 277–289.
[65] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Proc. of the 27th Advances in Neural Information Processing Systems (NIPS), 2014, pp. 3104–3112.
[66] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), Proc. of the 3rd International Conference on Learning Representations (ICLR), 2015, pp. 1–15.
[67] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: Proc. of the 7th International Conference on Learning Representations, ICLR 2019, 2019, pp. 1–8.
[68] S.B. Davis, P. Mermelstein, Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences, IEEE Trans. Acoust. Speech Signal Process. 28 (4) (1980) 357–366.
[69] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724–1734.
[70] I. Sutskever, O. Vinyals, Q. Le, Sequence to sequence learning with neural networks, in: Proc. of the 27th Annual Conference on Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.
[71] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (56) (2014) 1929–1958.
[72] M. Innes, Flux: elegant machine learning with Julia, J. Open Sour. Softw. (2018).
[73] G. Pagliarini, F. Manzella, G. Sciavicco, I.E. Stan, ModalDecisionTrees.jl: interpretable models for native time-series & image classification, https://github.com/aclai-lab/ModalDecisionTrees.jl, 2023.
[74] F. Bagge Carlson, Hyperopt. jl: hyperparameter optimization in Julia, https://github.com/bensadeghi/DecisionTree.jl, 2018.