
بسم الله الرحمن الرحیم

Institute for Research in Fundamental Sciences
Institute for Studies in Theoretical Physics and Mathematics (IPM)

School of Computer Science

Ph.D Thesis

Verification of Multi-Agent Learning Systems based on Epistemic Logic

By: Amirhoshang Hoseinpour Dehkordi

Supervisors:
Dr. Ali Movaghar
Dr. Majid Alizadeh

Winter 2023

وزارت علوم، تحقیقات و فناوری
پژوهشگاه دانش های بنیادی
(مرکز تحقیقات فیزیک نظری و ریاضیات)

باسمه تعالی

نام دانشجو : امیرهوشنگ حسین پور دهکردی
شماره: ۵۱/۸۰/۶۸۷۵
تاریخ : ۱۴۰۱/۱۲/۷
پیوست : دارد

فرم شماره ۲ - آموزشی

گزارش دفاع از رساله دکتری

به : معاون پژوهشی و تحصیلات تکمیلی
از : پژوهشکده علوم کامپیوتر

نام و نام‌خانوادگی دانشجو : امیرهوشنگ حسین پور دهکردی شماره دانشجویی : **۹۵۱۵۱۱۱۱** دوره دکتری : علوم کامپیوتر
تعداد کل واحد گذرانده : ۱۸ معدل کل : ۱۷/۲۵ تاریخ دفاع از رساله : ۱۴۰۱/۱۲/۰۳

* نتیجه ارزیابی رساله :
_ قبول با درجهٔ : (عالی ☑ بسیار خوب ☐ خوب ☐)
_ غیر قابل قبول

نام و نام خانوادگی و سمت اعضای هیات داوران رساله :

ردیف	نام و نام خانوادگی	سمت / دانشگاه *ضروری است نام دانشگاه اعضای هیات داوران عنوان گردد*	امضاء
۱	دکتر مجید علیزاده	استاد راهنما / دانشگاه تهران	
۲	دکتر علی موقر رحیم‌آبادی	استاد راهنما / دانشگاه صنعتی شریف	
۳	دکتر محمد گنج تابش	داور خارجی / دانشگاه تهران	
۴	دکتر محمد ایزدی	داور خارجی / دانشگاه صنعتی شریف	
۵	دکتر حمید بیگی	داور خارجی / دانشگاه صنعتی شریف	
۶	دکتر مسعود پورمهدیان	داور داخلی / پژوهشگاه دانش‌های بنیادی	
۷	دکتر نظام رهبانی	رئیس جلسه، نماینده معاون پژوهشی / پژوهشگاه دانش های بنیادی	
۸	دکتر احمد خونساری	از طرف رئیس پژوهشکده / دانشگاه تهران	

نام و امضاء رئیس پژوهشکده
پژمان لطفی کامران
مهر پژوهشکده

پژوهشگاه دانش‌های بنیادی (مرکز تحقیقات فیزیک نظری و ریاضیات)
پژوهشکده علوم کامپیوتر

"فرم باید بدون خدشه (خط خوردگی ، لاک گرفتگی ، دست نویس) باشد"

Acknowledgements

To my father,
Your support, sacrifices, and inspiration made my persistence throughout my academic pursuit.
I always admire you.

To my mother,
Your boundless love has shaped my character and given profound meaning to my life.
I always love you.

To my beloved wife,
Your belief in my potential, understanding, and helps have been a constant source of motivation.
You are the source of hope in my life.

I also dedicate this thesis in loving memory of my dear sister, Leila,
Who taught me and encouraged me in the early stages of my career.
You forever reside within my heart.

By acknowledging my father, mother, and wife, and dedicating this thesis to the memory of my sister, I am profoundly grateful for the impact each of you has had on both my academic and personal growth.

Thank you.
Amirhoshang Hoseinpour Dehkordi

Abstract

Artificial Intelligence (AI) algorithms play an important role in today's life. Data availability has made the Machine Learning (ML) algorithms noticed the most. Classification is the most common task ML algorithms do. These algorithms are mainly developed based on statistical approaches. There are often several classification approaches for a given problem, none of which is usually dominating, and each one delivers applicable information. Thus, employing ensemble learning techniques can enhance the classification system; however, interpreting this information is challenging. To address this challenge, we recommend interpreting information using public announcement logic (PAL), an Epistemic Logic (EL) extension. We suggest the verification of developed systems through multi-classifier modeling. In fact, we introduce MASKS (Multi-Agent System Knowledge Sharing) as a PAL model to verify properties in classifiers. MASKS bridges logical verification methods to widely applied classifiers. The language of MASKS is based on EL, which simplifies information interpretation. We present a tool to apply the MASKS verification method on arbitrary classifiers. Here, a pointwise verification is applied, in which property satisfaction is evaluated for a single input instead of the whole system. Thus, a property set is fed to classifiers as input to verify a property for the system. Using MASKS, we also suggest a uniform information framework, which investigates information gathered by a group of classifiers and information provided by other sources. When classifiers agree on an answer, the input verifies the property for the system of classifiers. The approach could be applied to single-frame data (SFD) entries. A real-world application based on MASKS was also developed to validate the findings.

Additionally, we introduce the Linear Temporal Public Announcement Logic (LTPAL) model by combining PAL with Linear Temporal Logic (LTL) to extend the model's expressiveness to interpret the model data-stream (DS) inputs. Usually, semi-continued DSs are a sequence of SFDs in which each frame is correlated to adjacent frames. This characteristic allows us to define actions in these types of inputs. To cover semi-continued DSs, we extend PAL to LTPAL. In LTPAL, a property can be defined for consecutive inputs, which leads us to define a verification method for a DS over multiple classifiers. The language of LTPAL allows us to define actions and investigate their occurrences. It is also possible to identify classifiers that infer that some action has occurred. An LTPAL tool is also developed and presented to use the verification approach in applications.

Keywords: Multi-Agent Systems, Formal Verification, Epistemic Logic, Temporal Logic

Contents

List of Tables

List of Figures

List of Acronyms

AI	artificial intelligence.
ANN	artificial neural network.
CAD	computer-aided detection.
CKC	classifier knowledge calculator.
DNN	deep neural network.
DS	data-stream.
EL	epistemic logic.
GPU	graphical processing unit.
GUI	graphical user interface.
IOU	intersection over union.
KBS	knowledge-based system.
LP	learning phase.
LS	learning system.
LTL	linear temporal logic.
LTPAL	linear temporal public announcement logic.
MALS	multi-agent learning system.
MAS	multi-agent system.
MASKA	multi-agent system knowledge aggregator.
MASKS	multi-agent system knowledge Sharing.
ML	machine learning.
MPPE	most probable path extraction.
MSE	mean squared error.
NLP	natural language processing.
NN	neural network.
NNF	negation normal form.
PAL	public announcement logic.
PALS	public announcement logic satisfaction.
QC	Quality Control.
ReLU	rectified linear unit.
RL	reinforcement learning.

RTS	real-time system.
SFD	single-frame data.
SFKE	single-frame knowledge extraction.
TEMS	temporal satisfaction.
TSTS	time series transition system.
UV	ultra-violet.
YOLO	You Only Look Once: Unified, Real-Time Object Detection.

Chapter 1

Introduction

Multi-agent learning system (MALS) is a sub-field of learning systems (LS) that focuses on studying multiple LS behaviors in a shared environment. In these systems, each agent does its tasks anonymously. Here, agents do their tasks using the information gathered in a learning process. For each input, the aggregated result would be represented by collecting the results of all agents. The algorithm of interpreting all information and the way of modeling defines MALSs. MALS could empower an LS either by increasing accuracy or reducing errors [1].

Nowadays, LSs are extending life quality in human beings. It made them spread in most applications, including: healthcare, self-driving cars, and industrial quality control (QC). Although applying LS in this era reduces errors compared to humans, each failure in these systems could cause serious consequences. Because of the statistical nature of LSs, one major drawback of this approach is that errors cannot be interpreted. In other words, no one could predict when or why an error would happen. To overcome this problem, scientists started to verify the system's properties to ensure that the system works well in specific conditions. Thus, verification has a pivotal role in LSs.

1.1 Learning Systems

An LS is a system with the ability to learn without being explicitly programmed [2]. It is commonly referred to as a system that operates after a learning phase (LP). These systems are designed to learn from experiences during the LP and improve performance with the experiences [3]. To begin this process, parameters would be randomly assigned . During the LP, training data tune the parameters of the LS. After the LP, LS is ready to solve the targeted problem. Depending on the nature of the problem, LSs are generally classified into three types:

- **Supervised learning**: These problems are presented with inputs and desired outputs. Supervised learning can be categorized into classification and regression based on the continuity of outputs. The most common of which are classification problems. The classification problems aim to assign a label from a predefined set to an input. In contrast, regressions assign a numeric value to an input.

- **Unsupervised learning**: No output is provided in LP for unsupervised learning algorithms. The main goal of unsupervised learning algorithms is to determine the structure of inputs.

- **Reinforcement learning (RL)**: With a predefined goal, agents interact with a dynamic environment to maximize it. The environment rewards agents based on actions and evaluates each agent's goal.

In general, among these LSs, the information provided by supervised learning algorithms could be undoubtedly evaluated. The purpose of unsupervised learning algorithms is to find the best structure, and RLs are to find the best strategy to maximize the goal function. Although there are metrics to evaluate these approaches, correct answers are not provided in both of them. One of the limitations of these systems is that they do not have a specific label or answer for each input. For this reason, the benchmarks generally express the improvement in their efficiency.

1.2 Verification of learning systems

The term "formal verification" refers to the process of checking the satisfaction of a property using a mathematical approach for the system. There are many mathematical approaches available for verifying systems, including: finite-state machines, transition systems, Petri nets, timed automata, hybrid automata, process algebra, epistemic logic, and temporal logic [4]. Historically, the first verification methods have focused on model checking, in which the whole possible states of the machine will be explored. The method has been applied to finite-state models. Model-checking algorithms were memory and time-consuming in systems with enormous states. Therefore, more practical approaches have been announced in which specific properties were defined. In order to identify the verification, a mathematical model has to be selected, and the model's satisfaction with the predefined properties was evaluated. The size of systems was expanded by employing LSs, such as Deep Neural Networks (DNNs), and such verification approaches could not be applied to real-world applications. For example, the method developed by Pulina *et. al.,* [5] in 2010 was limited to modeling a Neural Network (NN) with a maximum of six neurons, while Alexnet, which was entered in ILSVRC-2012 competition, includes about $\sim$650,000 neurons [6]. To overcome the problem, Huang *et. al.,* utilized pointwise verification to verify a property for an input of a system. Pointwise verification was successfully applied to verify larger DNNs [7].

1.3 Organization of the thesis

The main issue addressed in this thesis is to propose a verification of MALS based on epistemic logic (EL). The overall structure of the study takes the form of six chapters, including an introduction, preliminaries, multi-classifier's verification approach, verifying data streams, tools, and conclusion and future work. Chapter 2 provides problem definition and preliminaries. This chapter begins

by presenting classifiers and their formal definitions in section 2.2. Then, the basics of modal logic will be presented in section 2.3. In this thesis, epistemic logic and temporal logic, which are families of modal logic, will be involved. Epistemic logic will be introduced in section 2.3.1, and temporal logic will be introduced in section 2.3.2. This chapter will also give insight into the Kripke model and transition system created in chapters 3 and 4. Chapter 3 is concerned with the verification of multi-agent classifiers. This chapter begins with creating a public announcement logic (PAL) to interpret the aggregation of the distributed knowledge in section 3.2. A Multi-Agent Systems' Knowledge-Sharing algorithm (MASKS) will be developed to verify predefined properties in section 3.3. We will apply this model to the Fashion-MNIST, MNIST, and Fruit-360 datasets, in section 3.4. Next, in chapter 4, a combination of PAL and Linear Temporal Logic (LTL), namely Linear Temporal Public Announcement Logic (LTPAL), is created to define properties in a formal transition system. It will be done to extract knowledge in classifiers with data stream (DS) inputs. Then, a verification method is proposed in section 4.4.1. Next, a method of correction in DS classifiers based on the developed transition system is suggested in section 4.5. Chapter 5 presents the developed tools for MASKS and LTPAL. Finally, chapter 6 concludes the thesis and provides insights for future researches.

Chapter 2

Preliminaries

The vast usage of classifiers in real-world and critical applications made the appearance of the errors unsafe. In this study, we will show the power of Modal logic to verify classifiers. This study gives new ideas about the application of modal logic in the verification of classifiers. First, we will apply epistemic logic to verify single-framed classifiers. This model will develop a model to analyze information gained and communicate among multiple classifiers. Then, we will define a verification approach. After that, we will suggest temporal logic to interpret data stream inputs. Finally, a new model is proposed combining epistemic and temporal logic models to define suitable properties and verify the property for data stream inputs.

2.1 Introduction

One of the most successful approaches to verifying computation models is Modal logic. Modal logic was initially developed to investigate the truth in different conditions. For example, a formula may be true in the system's state but false in another state [8]. Modal logic contains various worlds (states) connected with relations; among them, different formulas would be

satisfied [9]. These worlds simulate different states of a system. Besides, in modal logic, different types of relations can be defined to specify the best model for the system. Here, one can define arbitrary modalities to create a modal logic for a specific application. Here, one can define arbitrary modalities to create a modal logic for a specific application. Epistemic logic and temporal logic are families of modal logic developed for specific applications. Epistemic logic and temporal logic are families of modal logic developed for specific applications. For example, in epistemic logic, relation $w_v R_i w_u$ between two worlds, w_v and w_u shows that the i-th agent could not distinguish these two worlds [10]; in temporal logic, relation $w_v R_i w_u$ simulates that w_u occurs right after w_v [11]. In other words, epistemic/temporal logic reasoning about how information/time could change the assertion of formulas. In the verification, these formulas could play the role of verifying property. In the temporal logic proposed paper, Pnueli argued that this logic could be helpful formalism to verify the correctness of computer programs [12]. Previously, a model was developed by Dehkordi*et. al.,* to verify properties for classifiers, in which the analysis of the input data and external information in a multi-agent scenario was considered [13]. The proposed method works fine for single-framed input data; however, it is unable to be used appropriately for data streams, i.e., video streams. In this note, a formal transition system model called Linear Temporal Public Announcement Logic (LTPAL) to extract knowledge in a classification process is suggested [14]. In semi-continued data streams, more progress is also possible by finding the similarity of vicinity frames, as the information changes in such data streams are considered minor [15]. The process of data understanding for single-frame data is divided into several steps. The first and foremost one is object recognition from the input data, for which many solutions are often with great performances [16]. The next step is knowledge extraction to understand the information acquired in the initial step (see [17]), where objects are transformed into symbols. This step allows knowledge to be shared within a multi-agent system

via a modal logic approach [13]. Subsequently, the given rules, which are fed into the system, should be interpreted. It can be achieved by employing predefined protocols (i.e., "cat is an animal" or "bat and pigeon could not have appeared in a single data frame" as predefined protocols). The final step is to satisfy the specified formula, employing a collected set of knowledge.

The epistemic and temporal logic operators are appropriate for describing systems' information and time-varying behavior. The undeniable role of modal logic in older computer systems persuades us to find out how modal logic could be investigated in verifying one of the recently most applied processes, classification. Classification nowadays is considerably applied in real-life and critical applications. Thus, it is evident that the verification of classifiers is required these days. In this article, we tried to provide some existing verification methods and give ideas for applying epistemic and temporal logic in the verification of classifies. To do this, first, we provide the preliminaries of *classifiers* in section 2.2 and *modal logic* in section 2.3, then ideas are presented in section 2.4.

2.1.1 Problem Definition

Classification is a process of assigning a label (from a label set) to an input (a vector). The solution comes with uncertainties in most classification approaches (i.e., neural networks). There are approaches in which multiple answers would be provided if the answer does not meet the minimum requirements (i.e., low probability or near boundary answers). Using the collection of answers, we could create a knowledge set. The knowledge set shows what are the possible answers using applied classifiers. This chapter will explain how to develop a model to extract knowledge in a classification process. Then, a developed model to consider classifiers capturing single-framed data will be introduced. Next, an extension to consider the data streams will be presented. Then, we will define some ideas of "verification" in data streams. All of these steps

will be based on a multi-agent system. Thus, it needs to define an intelligent agent, a classifier. There are many definitions for intelligent agents, one of the most common of all, which is also applied in this study, is to develop human-like classification and reasoning from input data by its experiences (such experiences could be achieved in a learning phase). Note that a set of agents (classifiers) also could refer to a group of agents (classifiers), so the phrases set and the group could be used interchangeably.

2.2 Classifiers

Classification algorithms are widely used in many real-world applications. Fraud detection in banking, object detectors and recognizers in automated cars, abnormally detectors in medical diagnosis, and fault detectors in aerospace tech are samples of the application of such algorithms. The primary purpose of the classifier is to find the most suitable label from a set of labels for the input data. In other words, classification is partitioning an input vector space into several classes. We notice that inputs are vectors in Euclidean n-space with a defined arbitrary distance metric. A classifier aims to do the classification process. Currently, the most commonly used classifiers are developed by machine learning algorithms. The first step to creating a machine learning classifier is building a model. Logistic Regression, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbour (KNN), and Artificial Neural Networks (ANN) are samples of machine learning classifier algorithms. Then, these algorithms train in a *training phase* in which a *training dataset* will feed to create machine learning classifiers. The training dataset consists of sample inputs with the correct label. Here, the classifier expects to find a relation between inputs and corresponding labels. In other words, to define the aim of the training phase, let $X = \{x_1, \dots x_n\}$ be a training dataset, $Y = \{y_1, \dots, y_n\}$ corresponding provided labels, and $C = \{c_1, \dots, c_k\}$ set

of labels, in which for all i, $y_i \in C$, and $X_{c_i} = \{x_j \mid y_j = c_i\}$. The training phase aims to discover the dependencies between X_{c_i} for all i [18]. Thus, the quality of the model's architecture and training dataset play the most crucial role. In the following, we discuss ANNs as a classifier, one of the most used machine learning methods.

As mentioned before, the inputs of a classifier are in the form of a vector of dimension n_0. In ANNs, the output is demonstrated by a vector e_i of dimension n_{-1}, in which the value of the $i-th$ element is the largest (the *softmax* function does the process of choosing the maximum value). Let $C = \{c_1, \ldots, c_{n_{-1}}\}$ be the set of labels; the representation of e_i in the output shows that c_i is the label the ANN provides for the input. Converting the input vector to the output vector is done by designing a fully connected and directed k-partite graph. The first partition of the graph is called the input, and the last is the output layer. These two layers are mandatory for ANNs, but adding layers between them is also possible; in this case, we call it a Deep Neural Network (DNN) 2.1. Each layer could be connected to any other layers with weighted edges. When an input is fed into the input, the process begins. Inductively, the value of the next layer is calculated by applying a nonlinear function on the sum of the value provided by each input edge. Each input edge's value multiplies the source node's value with the edge's weight. This process continued until the last layer's weights were calculated. Here, the weights should be defined in the training process. The training process in ANN initiates with random weights. The training dataset would be fed into the ANN one by one. ANN provides an answer for each input in the last layer, and we know the correct answer (it is provided in the training dataset). After that, the weights would be updated according to the difference between the correct and provided answer, using the gradient descent method. After the process, the trained ANN presents the function in which the input vector space can be partitioned into a certain number of classes. Generally, small changes in the training dataset cause minor changes in the classification algorithm. Although the sound and complete training

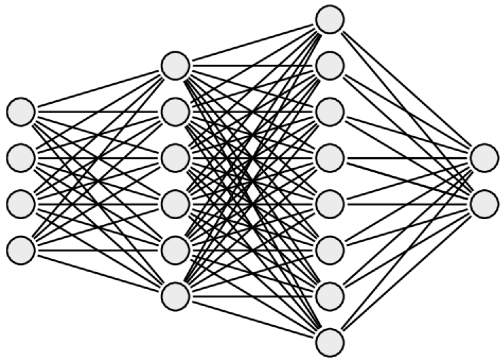


Figure 2.1: An example of DNN with an input vector of size 4, output 2, and two hidden layers of sizes 6 and 8.

dataset are impossible to collect in real-world applications, an approximately accurate classifier could be expected with a suitable training dataset.

More specifically, a feed-forward ANN is a tuple $N = (L, T, \Phi)$ where $L = \{L_k \mid k \in \{0, \cdots, n\}\}$, $n \geq 1$ is a set of layers, which contains nodes, each called neurons; $T \subseteq L \times L$ is the edge's weights of the n-partite graph, and $\Phi = \{\phi_k \mid k \in \{0, \cdots, n\}\}$ is a set of activation functions $\phi_k : R \to R$, where $0 < k \leq n$ and $D_{L_k} \subseteq \mathbb{R}^{n_k}$ are the dimensions of k-th layer. An ANN is called a deep neural network with at least three layers. The size of L_k is shown by n_k. Moreover, L_0 and L_n are called the input and output layers, respectively. For a single input x, $[x]_G$ is the class label of the ANN $G \in \mathbb{G}$, which contains x. Whereas for a set of input points X, the class label is $[X]_G = \bigcup_{y \in X} [y]_G$. In this chapter, and for clarification purposes, we drop the index G and write $[x]$ and $[X]$ wherever it can be understood from the context. *Hidden layers* are layers that are not input or output. The process is carried out inductively as follows: First, an input data x_{L_0} is fed into the input layer. Next, with i-th layer data, the data of the $i + 1$-th layer is calculated by $x_{L_{i+1}} = \phi_{i+1}(\Sigma_{j \leq i} x_{L_j} \times T_{L_j, L_{i+1}})$, in which T_{L_j, L_i} is the adjacency matrix of layers L_j and L_i. Then, the output class can be decided by the output layer's value (the most common way is to use

the maximum argument value as an output class).

2.3 Modal logic

Unanimously, the definition of modal logic is connected to incorporating modalities such as *necessity*, *possibility*, and *contingency* concepts. The modalities elevate modal logic to evaluate "what would be", "what would have been", "what is possible", "what is believed", and other closely related concepts. These characteristics of modal logic make it suitable to express more complex scenarios such as natural language processing (i.e., expressions of time, action, change, causality, information, knowledge, belief, obligation, permission, and far beyond) [19].

To be formally specific, the modal logic family is assumed to have the "Necessitation Rule" and "Distribution Axiom", which is called **K** (after Saul Kripke).

- Necessitation Rule: If A is a theorem of **K**, then so is $\Box A$,
- Distribution Axiom: $\Box(A \to B) \to (\Box A \to \Box B)$.

Here, the symbol $\to$ is used for "if ... then" and "$\Box$ for "it is necessary that". The *necessitation Rule* indicates that "any theorem in this logic is necessary", and the *Distribution Axiom* says that "if it is necessary that if A then B, then if necessarily A, then necessarily B". The operator $\diamond$ is for the term "it is possible that"; it can be defined by letting $\diamond A = \neg\Box\neg A$ in which $\neg$ uses for "not".

- (M): $\Box A \to A$,
- (4): $\Box A \to \Box\Box A$,
- (5): $\diamond A \to \Box \diamond A$,
- (B): $A \to \Box \diamond A$.

The axiom (M) indicates that whatever is necessary would be the case. A modal logic **M** could be driven by adding (M) to **K** (**M**, also known as **T**). Many logicians believe that M is too weak to correctly formalize the logic of necessity and possibility. The axioms (4) and (5) would add iteration or repetition to modal operators. **S4** is the system that results from adding (4) to **M**, and **S5** is **M** plus (5). In **S4**, any number of boxes (diamonds) could be replaced by one, which means the term "A is necessarily necessary ... necessary" is equivalent to "A is necessary. The same goes for diamonds (possible). The system **S5** indicates a stronger term in which the last one could be replaced in any sequence of modal operators (box or diamond). In other words, $\bigcirc\bigcirc\ldots\Box\varphi = \Box\varphi$ and $\bigcirc\bigcirc\cdots\diamond\varphi = \diamond\varphi$ in which $\bigcirc$ could be either $\diamond$ or $\Box$. The system **S5** could also be formulated by adding (B) to **S4**, in which (B) demonstrates that "if A is the case, then A is necessarily possible" [20].

The idea of modal logic leads us to define more modalities and create a family of modal logic. For example, the "necessary" modal operators K and G would lead to the definition of "epistemic logic" and "temporal logic" respectively.

Definition 2.3.1 (Language of modal logic)**.** The syntax of the language modal logic is as follows in BNF:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi.$$

Here, the intended meaning of the formula $\Box\varphi$ is "φ is necessary". To define the semantics of modal logic's language, we will define the Kripke model, which provides a model for modal logic (formal semantics for non-classical logic systems) for reasoning about modal relations such as knowledge (Fagin et al. [21]). First, we define the Kripke model for a single-agent system.

Definition 2.3.2 (Kripke frame)**.** Let W be a set (namely worlds), and R be a binary relation (namely accessibility relation) on W; a Kripke frame is a pair $\langle W, R\rangle$ [22].

Definition 2.3.3 (Kripke model)**.** Let $\langle W, R\rangle$ be a Kripke frame and $Prop$ be a set of propositional letters; a Kripke model $\mathcal{M}$ is a tuple (W, R, V), in which an evaluation function $V : W \longrightarrow 2^{Prop}$ maps each world to the set propositions that are true at that world.

A formula φ is valid in a model $\mathcal{M} = (W, R, V)$ when φ is true in every world $w \in W$.

In a multi-agent system with n agents, we have multiple relations $R_1, \ldots, R_n$. Here, the frame is a tuple $\langle W, R_1, \ldots, R_n\rangle$, and the Kripke model is a tuple $\mathcal{M} = (W, R_1, \ldots, R_n, V)$. The remaining concepts stay similar.

- $\mathcal{M}, w \models p \quad$ iff $\quad p \in V(w)$,
- $\mathcal{M}, w \models \neg\phi \quad$ iff $\quad \mathcal{M}, w \nvDash \varphi$,
- $\mathcal{M}, w \models \phi \wedge \psi \quad$ iff $\quad \mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$,
- $\mathcal{M}, w \models \Box\phi \quad$ iff $\quad \forall v \in R_i(w), \mathcal{M}, v \models \phi$.

In the following sections, we will discuss *epistemic logic* and *temporal logic*.

2.3.1 Epistemic logic

Epistemic logic is in the family of modal logic in which knowledge is interpreted. The "necessary" modal operator in epistemic logic is K, in which $K_a\varphi$ reads as "agent a knows φ". In this context, the formula $K_a\varphi$ asserts φ is true in all worlds that agent a considers epistemically related to its current information. In other words, in this context, when an agent knows a formula, there is no possible scenario in the agent's perspective in which the formula is interpreted as false. It means that with the knowledge that the agent collects, the formula could be interpreted as true, and the agent could ensure it. The formula $\neg K_a\neg\varphi$ could be read as "agent a does not know $\neg\varphi$". It shows

that the agent could not reason that the formula was false. It means the formula is possible in a scenario. In the following, the formal definition and the syntax of epistemic logic are provided.

Definition 2.3.4 (Language of epistemic logic)**.** Let $Ag = \{1, \ldots, n\}$ be a finite set of agents. The syntax of the language epistemic logic is as follows in BNF:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi.$$

In this definition, p is an atomic formula. The meaning of $\neg\varphi$ is the negation of the formula φ, and $\varphi \wedge \psi$ is φ and ψ, and $K_i\varphi$ is "agent i knows φ". The Kripke model of an epistemic logic model is a tuple $\mathcal{M} = (W, R_1, \ldots, R_n, V)$, where W is a set of worlds, $R_i \subseteq W \times W$ is an equivalent relation between worlds for each agent, and $V : W \longrightarrow 2^{Prop}$ is the evaluation function, which determines "which atomic formula is true in that world". The semantics of epistemic logic formulas are as follows:

- $\mathcal{M}, w \models p$ iff $p \in V(w)$,
- $\mathcal{M}, w \models \neg\phi$ iff $\mathcal{M}, w \nvDash \varphi$,
- $\mathcal{M}, w \models \phi \wedge \psi$ iff $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$,
- $\mathcal{M}, w \models K_i\phi$ iff $\forall v \in R_i(w), \mathcal{M}, v \models \phi$.

In other words, for a model $\mathcal{M} = (W, R_1, \ldots, R_n, V)$, interpretation of epistemic formula $K_i\varphi$ in a world w would be made by checking the satisfaction of the formula in all worlds related to w with R_i.

One of the exciting parts of using the multi-agent system is aggregating all gathered knowledge together. For this purpose, a notation of *distributed knowledge* is developed to capture knowledge

distributed within a group of agents. We will add *distributed knowledge* operator to epistemic logic syntax by adding $D_A, A \subseteq Ag$. Where Ag is a group of agents, the intended meaning of the formula $D_A\varphi$ would be "φ is a distributed knowledge in the group A". Another notation suggested for a group of agents is to find out whether everybody knows a formula or not, and it would be shown by E_A. The intended meaning of the formula $E_A\varphi$ would be "everybody in group A knows φ".

- $\mathcal{M}, w \models D_A\phi$ iff $\forall v \in R_A(w), \mathcal{M}, v \models \phi$, where $R_A := \bigcap_{i \in A} R_i$.
- $\mathcal{M}, w \models E_A\phi$ iff $\forall v \in R_A(w), \mathcal{M}, v \models \phi$, where $R_A := \bigcup_{i \in A} R_i$.

Public Announcement Logic

Public Announcement Logic (PAL) aims to study knowledge and public communication. Here, we will demonstrate the PAL's extension of the epistemic logic. Let us start with the "Muddy Children Puzzle" example:

Example 2.3.5 (Muddy Children Puzzle)**.** The father called his three children, playing in the mud. When they came to father, some had mud on their forehead. The father asked them to sit around a table to see the other's forehead. No one can see his or her forehead, and no one can examine his or her forehead. Father said: "At least one of you has a muddy forehead; if anybody knows whether his or her forehead is dirty, stand up". No child stood up. Father repeated the quote. Some of them stood up. Father repeated. All of them stood up. How many children had muddy foreheads?

This is where PAL can be applied to find the answer.

Definition 2.3.6 (Language of PAL)**.** Let $Ag = \{1, \ldots, n\}$ be a finite set of agents. The syntax of the language PAL is as follows in BNF:

$$\phi ::= p \mid \neg\phi \mid (\phi \wedge \phi) \mid K_i\phi \mid D_A\phi \mid [\phi]\phi.$$

The formula, $[\phi]\phi$, is added to predefined epistemic logic. The formula $[\psi]\phi$ reads as "after a correct announcement of ψ, ϕ will hold".

- $\mathcal{M}, w \models [\psi]\phi$ iff $\mathcal{M}, w \models \psi$ implies $\mathcal{M}^{\psi}, w \models \phi$.

Where $\mathcal{M}^{\psi} := (W^{\psi}, R_1^{\psi}, \ldots, R_n^{\psi}, V^{\psi})$ with

$W^{\psi} := \{w \in W; \mathcal{M}, w \models \psi\}$,

$R_j^{\psi} := R_j \cap (W^{\psi} \times W^{\psi})$ for all $j \in \{1, \ldots, n\}$ and

$V^{\psi}(w) := V(w)$ for all $w \in W^{\psi}$.

The announcement operator will make the Kripke model looks like a dynamic model. More precisely, the satisfaction of $\mathcal{M}, w \vDash [\psi]\phi$ is equal to the satisfaction of $\mathcal{M}^{\psi}, w \vDash \phi$. Here, the dynamics of the model could be observed.

Now, we could find the answer to the muddy children. Let $Ag = \{1, 2, 3\}$ be the set of agents, which are three children here. Let p_i represent the ith child's forehead is muddy. The model has eight worlds $W = \{w_{000}, w_{100}, w_{010}, w_{001}, w_{110}, w_{101}, w_{011}, w_{111}\}$, the size of the power set of $Prop = \{p_1, p_2, p_3\}$. In the initial model, any child has no information about his/her forehead but knows about others. Thus, we can define relations as follows:

$$R_1 = \{(w_{0ij}, w_{0ij}), (w_{0ij}, w_{1ij}), (w_{1ij}, w_{0ij}), (w_{1ij}, w_{1ij}) \mid i, j \in \{0, 1\}\},$$

$$R_2 = \{(w_{i0j}, w_{i0j}), (w_{i0j}, w_{i1j}), (w_{i1j}, w_{i0j}), (w_{i1j}, w_{i1j}) \mid i, j \in \{0, 1\}\},$$

$$R_3 = \{(w_{ij0}, w_{ij0}), (w_{ij0}, w_{ij1}), (w_{ij1}, w_{ij0}), (w_{ij1}, w_{ij1}) \mid i, j \in \{0, 1\}\}.$$

For any world, the evaluation function is defined as follows:

$$V(w_{i_1 i_2 i_3}) = \{p_i \mid i_j = 1, j \in \{1, 2, 3\}\}.$$

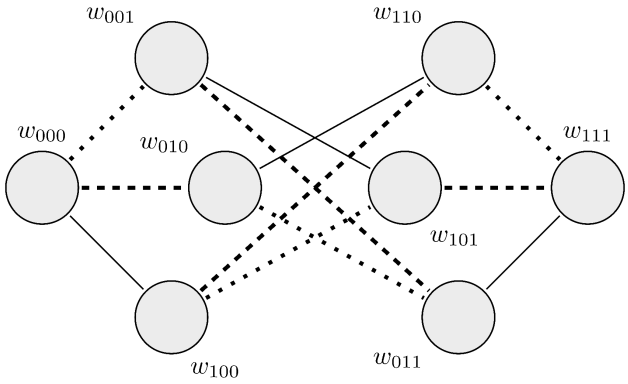


Figure 2.2: The muddy children problem's initial Kripke model.

The initial model is depicted in fig 2.2.

When the father says, "At least one of you has a muddy forehead", it means the world w_{000} is not the case (see fig 2.3). After that, when he said, "if anybody knows whether his or her forehead is dirty, stand up", and no one stands up, it means w_{100}, w_{010}, and w_{001} are not the actual worlds because if it was, by removing w_{000}, children number 1,2, and 3 could distinguish whether w_{100}, w_{010}, and w_{001} (respectively) are the actual worlds. When the father repeats, some of them stand up. It means the world w_{111} is impossible. If it was, no one could be ensured and stood up because if the actual world was w_{111}, the first child could not distinguish it from w_{011}, the second child could not distinguish it with w_{101}, and the third child could not distinguish it with w_{110}. Therefore, the answer to the question "How many children had muddy foreheads?" would be "two".

2.3.2 Temporal logic

Temporal logic is a logic for reasoning about time and temporal information. The model is represented by $\mathcal{T} = \langle T, \prec \rangle$, where T is a non-empty set of time instants and $\prec$ is a binary relation. The temporal precedence relation is usually a strict partial ordering, an irreflexive, transitive, and asymmetric relation (Sometimes, it is assumed to be reflexive and added

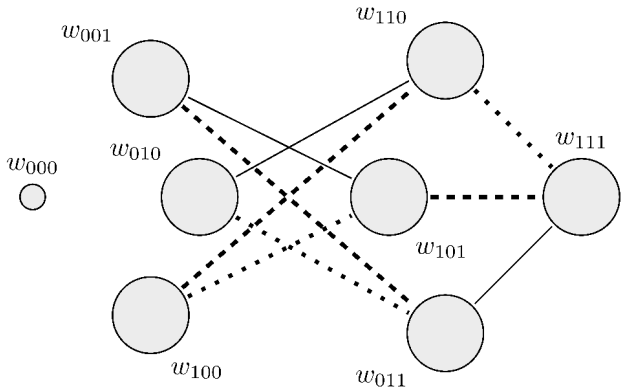


Figure 2.3: After the first announcement, the children know that w_{000} is impossible.

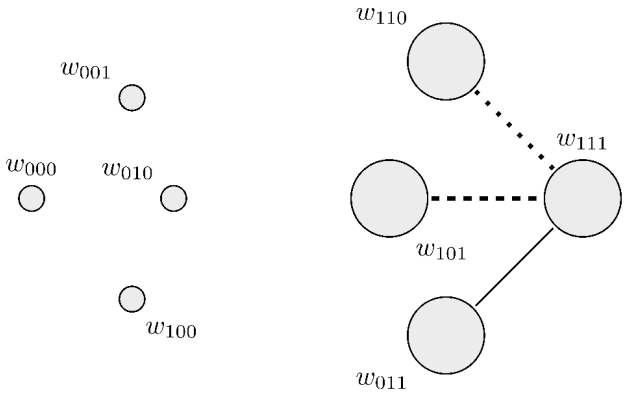


Figure 2.4: After the children are notified that no one stands up, they know that w_{100}, w_{010}, and w_{100} are impossible.

antisymmetry). Some temporal logic possible properties $\mathcal{T} = \langle T, \prec \rangle$, which are expressed by first-order syntax represented below:

- reflexivity: $\forall x(x \prec x)$;
- irreflexivity: $\forall x \neg(x \prec x)$;
- transitivity: $\forall x \forall y \forall z(x \prec y \wedge y \prec z \rightarrow x \prec z)$;
- asymmetry: $\forall x \forall y \neg(x \prec y \wedge y \prec x)$;
- anti-symmetry: $\forall x \forall y(x \prec y \wedge y \prec x \rightarrow x = y)$;
- linearity (trichotomy, connectedness): $\forall x \forall y(x = y \vee x \prec y \vee y \prec x)$;
- forward-linearity: $\forall x \forall y \forall z(z \prec x \wedge z \prec y \rightarrow (x = y \vee x \prec y \vee y \prec x))$;
- backward-linearity: $\forall x \forall y \forall z(x \prec z \wedge y \prec z \rightarrow (x = y \vee x \prec y \vee y \prec x))$;
- beginning: $\exists x \neg \exists y(y \prec x)$;
- end: $\exists x \neg \exists y(x \prec y)$;
- no beginning: $\forall x \exists y(y \prec x)$;
- no end (unboundedness): $\forall x \exists y(x \prec y)$;

To date, the main applications of temporal logic include formalism and reasoning of philosophical problems about time, the formalism of natural language expression, which are about time, encoding timed knowledge in artificial intelligence, and a tool for specification, analysis, and verification of computer programs.

These are all centered around the problem of "What will be necessary or possible in the future?". One fundamental distinction in temporal logic is its linearity, which could be linear or non-linear(tree-like representation). In linear temporal logic, the flow of time is depicted as a line, but in non-linear temporal logic (also known as branching into multiple possible futures), the past is fixed and allows multiple possible futures. Prior introduced temporal logic (also known as basic Tense logic), and the main idea was to use temporal logic in natural language [23, 24]. The basic language of Prior's Tense logic is the extension of propositional logic with two modal operators:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid P\varphi \mid F\varphi$$

. The intended meaning of $P\varphi$ is "It has sometimes been the case that φ" and $F\varphi$ means "It will sometimes be the case that φ" [25]. Here, operators $H\varphi = \neg P\neg\varphi$, $G\varphi = \neg F\neg\varphi$, $A\varphi = H\varphi \wedge \varphi \wedge G\varphi$, and $E\varphi = P\varphi \vee \varphi \vee F\varphi$ could be defined, which means:

- $H\varphi$: "It has always been the case that φ",
- $G\varphi$: "It will always be the case that φ",
- $A\varphi$: "always φ",
- $E\varphi$: "sometimes φ".

A temporal model for temporal logic is a triple $\mathcal{M} = \langle T, \prec, V\rangle$ where $\mathcal{T} = \langle T, \prec\rangle$ is the frame, and V is a valuation function. The truth of formula φ, which is denoted by $\mathcal{M}, w \vDash \varphi$, is defined inductively as follows:

- $\mathcal{M}, w \vDash p$ iff $w \in V(p)$;
- $\mathcal{M}, w \vDash \neg\varphi$ iff $\mathcal{M}, w \nvDash \varphi$;

- $\mathcal{M}, w \vDash \varphi \wedge \psi$ iff $\mathcal{M}, w \vDash \varphi \wedge \mathcal{M}, w \vDash \psi$;
- $\mathcal{M}, w \vDash P\varphi$ iff $\mathcal{M}, w' \vDash \varphi$ for some time instant w' such that $w' \prec w$;
- $\mathcal{M}, w \vDash F\varphi$ iff $\mathcal{M}, w' \vDash \varphi$ for some time instant w' such that $w \prec w'$;

In the following section, we will discuss Linear Temporal logic, which is widely applied in computer science.

Linear time Temporal Logic (LTL)

A very natural class of temporal logic is linear temporal logic [12, 26]. After the invention of temporal logic, several linear extensions have been developed. This section will discuss an extension of temporal logic with *Next* and *Until* operators. After adding these operators, Linear-time Temporal Logic (LTL) will be developed.

Here, the operator X ("neXt") and U ("Until") with the following semantics could be added.
$\mathcal{M}, w \vDash X\varphi$ iff $\mathcal{M}, s(w) \vDash \varphi$,
$\mathcal{M}, w \vDash \varphi U\psi$ iff $\mathcal{M}, s \vDash \psi$ for some s such that $w \prec s$ and $\mathcal{M}, u \vDash \varphi$ for every u such that $w \prec u \prec s$.

Here, $X\varphi$ is true when φ is true in the immediate successor. The intended meaning of $X\varphi$ is "φ will be the case in the immediate next state" [25]. The operator X satisfies the K-axiom ($X(\varphi \to \psi) \to (X\varphi \to X\psi)$) and the functionality axiom ($X\neg\varphi \to \neg X\varphi$). The binary temporal operator $\varphi U\psi$ is true when φ is true in the next states *Until* ψ becomes true. The intended meaning of $\varphi U\psi$ is "φ will be the case until a time when ψ is the case" [25]. By adding *neXt* and *Until* operators and considering that there is no past operator in the LTL, the language of LTL in BNF would be:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid X\varphi \mid \varphi U\varphi.$$

2.4 Verification of Classifiers

Nowadays, data availability causes learning classification algorithms to get noticed and mature fast. There are lots of classification methods available. Most of them are based on statistical approaches. Although are fast and accurate, they still have errors that are hard to be interpreted. The wide usage of classifiers in real and critical applications made their error critical and sometimes even hazardous. Thus, scientists noticed the interpretation and verification of classifiers. Unfortunately, the earlier verification methods do not work here due to the enormous input domain and the model's size of most classifiers. These problems require verification and interpretation methods that are executed within a reasonable time.

To do this, using the characteristics of statistical classifiers could help to improve the performance. Usually, statistical classifiers try to partition the vector space into some bounded space and assign a label to each of them. Hence, statistical classifiers give answers with probabilities. It means the probabilities are lower near partitions' boundaries, and more errors will happen when input is located near boundaries. For example, suppose the image is located near the partition boundaries in an image classifier. In that case, the output label might move from one partition to another by adding a small amount of noise. Epistemically, the noise would not change the representative class of the image, but mathematically, it would change the value of the image's vector value. Hence, when the image is located near the partition's boundaries, small changes could change the partition, which could change the label. This scenario depicted errors caused by noises (adversarial examples). Scenarios other than noises could cause such errors. For example, assume that we took a picture of an object, the camera angle changed, and then we took another picture of the same object. These two images epistemically contain the same object but might be located in different partitions with different labels. Many other manipulations could cause such errors. Considering these, Huang *et. al.,* [7] use pointwise verification for Deep Neural Network

(DNN) classifiers. To do this, instead of verifying properties on a model, they verified properties for an input point of the model. As a property set, they fed sets of neighborhood and manipulation (for each layer of the DNN model) of input into the model. If the model assigns the same labels to all inputs, the input is counted as verified. Huang *et. al.,* showed that errors would be reduced significantly.

2.4.1 Epistemic Logic and Classifiers

In most classification problems, no dominating method exists. Some classifiers made better suggestions in specific cases. This section will show how interpreting answers reduces errors using multiple classifiers. Initially, using the method developed by Huang *et. al.,* a classifier provides a set of answers by inputting the input property set. If the set size equals one, all of the property set's elements are classified in the same class. Thus, all agreed on the answer, and it was verified. Suppose the set size was greater than one, the answer would not be verified, but the verified answer would be among the set (because of the property definition). By applying multiple classifiers, multiple sets of answers would be created. Assume that, in a Kripke model $\mathcal{M} = (W, R_1, \ldots, R_n, V)$, each world (state) w_i demonstrates a possible output label $V(w_i) = c_i$ of the classifier. The interpretation in the epistemic logic would be that the i-th classifier could not distinguish between different possible worlds. For example, possible worlds for the i-th ANN classifier would be $\{w_i \mid V(w_i) \in [X]_G\}$. By calculating the intersection of possible worlds each classifier provides, $W_D = \bigcap_i W_i$ would be the distributed knowledge of these classifiers. Now, if the cardinality of W_D equals one, the input would be verified for the set of classifiers, considering the predefined properties. It is trivial that all classifiers agree to the provided answer, and it satisfies all predefined properties in them. Dehkordi *et. al.,* [13] showed that using multiple classifiers would significantly reduce the error comparing the method developed by Huang *et.*

al., [7]. This kind of interpretation of knowledge could be applied in critical systems because, although it took much time to get the answer, error reduction is significant and is more important in such problems.

In other scenarios, the power of epistemic logic could be utilized. Besides, the epistemic logic model could be used to collect information from different sources. For example, assume that some image classifiers detect day/night and bat/pigeon and some outer information that states bats would not appear in days. Thus, we would interpret that the situation with day and bat together would not be valid. In the epistemic model, one can find which classifier was faulty if an error happened. It leads us to create a monitoring system over classifiers. If the agents disagree on a single label, the system of classifiers will provide a set of labels. It means all classifiers agreed that the answer is among the set. The developed system could be applied as an assistant system in this condition. For example, in computer-aided detection (CAD), a computer is applied to generate output as an assisting tool for a clinician detecting potential abnormalities, i.e., on diagnostic radiology exams. The developed system could be applied as an assistant to bring detected objects to the doctor's attention.

2.4.2 Temporal Logic and Classifiers

As epistemic logic is a potent tool for interpreting information of single framed data, temporal logic is compelling for analysis stream of data to detect actions. For example, only objects could be classified in an image, but actions could also be determined in a video. Temporal logic could help when an ordered stream of data is provided. More specifically, in a data stream, one can define an action using an LTL formula (i.e., $\varphi = \varphi_1 \wedge X\varphi_2 \wedge XX\varphi_3$ defines an action in which φ_1 happened, then in the next step φ_2, and finally in the next φ_3). The satisfaction of such a formula means that the action (φ) happened. Here, the data stream is assumed to be a stream of single-

framed data; hence, we had an epistemic model for verifying single-framed data. Combining the temporal and epistemic logic leads us to define a property for an input. They led us to apply the pointwise verification using the predefined properties. To do this, we create a transition system consisting of each frame's Kripke models. Each frame has a Kripke model with a set of worlds W_i (considering the predefined property); we put all frames' Kripke models in a hierarchical order. Then we connect every two consecutive models with temporal relations. It makes a k-partite graph in which W_i is connected to W_{i-1} and W_{i+1}. In the transition system, $\Pi_i \mid W_i \mid$ unique execution temporal paths would be created; these are all possible scenarios of objects that appeared in the data stream. Now, by checking the satisfaction of LTL formulas (which are the properties to be verified) over all paths, we can check whether the model for specific input is verified or not. Here, we could also check whether a formula is possible in an execution path or not (in case some paths satisfy the properties). As we said in section 2.4.1, a possible execution path could be applied in assistant systems (i.e., CAD).

2.5 Conclusion

Modal logic is always known as a potent tool for verification. This chapter offered modal logic as a verifier of classifiers and an assistant system. First, we showed how epistemic logic could be applied to verify single-framed classifiers. It was done by defining a Kripke model constructed using the information of applied classifiers. Then, we defined the application of temporal logic to detect actions in data stream inputs. Finally, a combination of epistemic and temporal logic models is exhibited to verify properties for data stream inputs. In the logic, properties could be created in the form of LTL. These models are applicable when the cost of fault classification is high, such as in critical cases. The application of the developed model as an assistant was also described.

Chapter 3

Multi-classifier's verification approach

In this chapter, a new ensemble approach for classifiers is introduced. A verification method for better error elimination is developed by integrating of multiple classifiers. A multi-agent system comprised of multiple classifiers is designed to verify the satisfaction of the safety property. In order to examine the reasoning concerning the aggregation of distributed knowledge, a logical model has been proposed. A Multi-Agent Systems' Knowledge-Sharing algorithm (MASKS) has been formulated and developed, to verify predefined properties. As a rigorous evaluation, we applied this model to the Fashion-MNIST, MNIST, and Fruit-360 datasets, where it reduced the error rate to approximately one-tenth of the individual classifiers.

3.1 Introduction

Nowadays, classifiers are a mandatory part of most real-world AI applications. Among these, and in particular, are safety-critical systems, in which failures may result in fatal consequences. The most common types of classifiers are based on statistical methods employed to improve the performance of certain tasks (i.e., Artificial Neural Networks (ANNs)). Such an AI-assisted

statistical process of decision-making is based on the acquired information. This complicates the interpretation of the cause underlying the decisions made. As for performance measures, two issues are material and often cause vulnerabilities. First, are architectural flaws and deficiencies, and second, having limited sets for training the classifiers. Besides architectural flaws and limited datasets, the system may even be confronted with prearranged noisy inputs, which have been inputted with malignant intentions (i.e., adversarial examples [27] and [28]). For example, in image processing cases, adversarial inputs tend to locate themselves towards neighborhoods (which, considering predefined norms, are images located near the vicinity). Furthermore, manipulations (which, in human perception, are considered visually similar), such as camera angle changes and image skewing [7]. In this case, it seems the property proposed to describe the classifier's knowledge should incorporate possible neighborhoods and manipulations to verify its correct performance. Here, and considering formal verification, theoretical understanding regarding the correctness of our formula (property) gains importance. So, verification methods have been applied more widely to classifiers [29]. As a basis, one can reference the safety verification on ANNs (as classifiers) in *Multi-Layer Perceptrons Neural Networks*, in which linear computation constraints are abstracted to "Boolean combinations of linear arithmetic constraints", Pulina *et. al.,* [5]. Advancing in time, environmental modeling, formal specification, system modeling, computational engines, and correct-by-construction design were further considered by Sanjit *et. al.,* [30]. Building on the studies above, a unified framework was created to provide safe and reliable integration for human-robot systems, D. Sadigh *et. al.,* [31]. In addition, and regarding safety-critical verification of ANNs (as classifiers), Scheibler *et. al.,* applied a bounded model checking method. In this work, the formulas introduced by the verification method were solved via iSAT3 (an SMT-solver) and special deductions [32]. Following their research, improvements were made by Katz *et. al.,* by taking into consideration more general properties.

This allowed them to verify networks using simplex methods including piece-wise linear ReLU activation functions [33]. An approach to studying the robustness of multiple classifiers with a focus on ANNs is developed by Gross *et. al.,* "to generate optimal results for medium-sized neural networks" [34]. Albeit, none of these methods could verify larger networks, such as Alexnet, which includes about ∼650,000 ReLU nodes [6]. All of these methods put effort into the verification of the model, so the time complexity was based on the size of the model. Therefore, and due to the growing rate of size of such models, these models could not be applied to verify the latest models.

Pushing onward, and as another perspective, point-wise robustness (which could be applied for each layer of ANNs -as classifiers) was introduced and inserted into the verification method by Huang *et. al.,* [7]. Their algorithm exhaustively searched the neighborhood of a network's inputs with reasonable complexity and in an acceptable time frame. This model appears well optimized and more general. Their research showed impressive results by adequately defining neighborhoods and manipulating them as properties (they could eliminate misclassifications). However, defining good neighborhoods and manipulation in general for classifiers has difficulties. Besides, it is not designed to verify safety-critical cases regarding a collaborative system of classifiers because it cannot reason about the knowledge generated with multiple classifiers.

In this chapter, a new ensemble approach for classifiers is introduced. Unlike statistical ensemble approaches, this method can determine the source of knowledge in a deterministic manner. Moreover, we will examine the following: "Could this multi-agent scenario helps us to reduce error in the case of improper neighborhoods and manipulations?". Next, all possible "manipulations" are collected according to the predefined properties. Consequently, all inputs in the test set, plus all manipulations, would be fed into all classifiers as inputs. Each set of classifiers would output a set of classes.

The structure of the chapter is as follows. In section 3.2, the developed model, based on Epistemic Logic and a multi-agent system, is presented. In section 3.3, collaboration algorithms are introduced. In section 3.4, two types of examples explaining the details of the proposed model are provided. Finally, section 3.5 concludes the chapter.

3.2 Logical model for classifiers

In this section, we introduce a novel interpretation of Dynamic Epistemic Logic that suits the formal description of the dynamics of the knowledge of classifiers as the agents in a Multi-Agent System.

3.2.1 Safety in Classifier Verification

Formal verification is the mathematical process of ensuring that a model fulfills some properties in an environment [35]. Pulina *et. al.,*, [5] defined a verification method for classifiers using *safety* as the property to be satisfied by the classifiers. *Safety* is defined as a consistent classification result due to minor input changes.

Let us illustrate the property of *safety* with a sample example. Classification is a process in which an input space is partitioned into several output classes. Assume we have an image classifier that accepts 100×100 matrices representing grayscale input images, and the classifier will verify if an image has a face. So, the input space is a 10000-dimensional vector space, and the output space would consist of two classes: "images with a face" and "images without a face". Note that by changing the value of a single pixel of an image, the input image *changes*, but human perception of an image does not alter by such minor changes. On the other hand, consider the objects that are recognized as being the same through various camera angles, or at night or day, etc.; the human mind perceives them as the same object, whereas they may be completely different as an input point

for the classifier. To verify *safety*, these minor changes shall not lead to inconsistent classification results.

To overcome these difficulties, Huang *et. al.,*, [7], introduced the concepts of *neighborhood* and *manipulation* sets of an input image.

Definition 3.2.1. Let $X \subseteq \mathbb{R}^n$ be an input domain and $x_0 \in X$, then the ϵ-*neighborhood* of x_0 is the set:

$$\eta_\epsilon(x_0) = \{x \in X \mid d(x_0, x) < \epsilon\},$$

where d is an arbitrary distance metric.

Definition 3.2.2. Let $X \subseteq \mathbb{R}^n$ be an input domain and $x_0 \in X$, then the $\mathcal{F}$-*manipulation* of x_0 is the set:

$$\mu_\mathcal{F}(x_0) = \{x \in X \mid \mathcal{F}(x_0, x) = 1\},$$

where $\mathcal{F} : \mathbb{R}^n \times \mathbb{R}^n \to \{0, 1\}$ as an arbitrary function.

Whenever ϵ and $\mathcal{F}$ are clear from the context, we will drop the subscripts in η and μ and will call them, neighborhood and manipulation set.

Remark 3.2.3. The verification method goals are developed to satisfy specific predefined properties. However, the classification generally aims to reduce errors. The verification methods could help classifiers become self-aware if properties are defined correctly. Nevertheless, defining such properties is difficult in many cases. As we can see, Huang *et. al.,*, have presupposed an arbitrary but fixed measure of distance ϵ for neighborhood and characteristic function of manipulation set $\mathcal{F}$, but defining the ϵ for neighborhood and the function $\mathcal{F}$ in manipulation is not trivial. Examples 3.4.2, 3.4.3, and 3.4.4 will show how multi-classifiers could reduce the misclassification caused by improper safety properties.

Let G be a classifier. For an input x, $[x]_G$ is the label of the output class of x in G. For a set of input points X, $[X]_G$, which is defined as $\bigcup_{y \in X}[y]_G$, is the label of the output class of X in G.

The concepts of *safety* and robustness are defined by Huang *et. al.,* [7]. We explain these within the literal context of this chapter in the definitions below.

Definition 3.2.4. Let G be a classifier. We say that G is safe for input point x_0 exactly when for all input point x, $x \in \eta(x_0) \cup \mu(x_0)$ implies that $[x]_G = [x_0]_G$.

Definition 3.2.5. Let G be a classifier and x be an input point. x is called *robust* exactly when $[\eta(x) \cup \mu(x)]_G = [x]_G$.

Definition 3.2.6. A property $\rho(x)$ in a given classifier G is *verified* exactly when $\mid [\rho(x)]_G \mid = 1$. In which "$\mid A \mid$" the set A size.

For classifier G and input point x_0, if every point in the $\eta(x_0) \cup \mu(x_0)$ is classified into the same class, the input point x_0 would be a *robust* point and the safety property would be verified for x_0.

Definition 3.2.7. A property $\rho(x)$ in a given set of classifiers $A_G = \{G_1, \ldots, G_n\}$ is *verified* exactly when $\mid \bigcap_{G \in A_G}[\rho(x)]_G \mid = 1$.

3.2.2 Dynamic Epistemic Logic and Public Announcement Logic

Epistemic Logic is a modal logic concerned with knowledge and reasoning about knowledge. The basic modal operator of Epistemic Logic, written as K, is read as "It is known that,". Dynamic Epistemic Logic is introduced to model the epistemic interactions of the agents when dealing with more than one agent.

Here we review the basics of Dynamic Epistemic Logic (DEL) to be able to reason regarding agents' knowledge. DEL is a logical framework designed to deal with the dynamics of knowledge

of agents in multi-agent systems by adding dynamic modalities to Epistemic Logic. Hence, the agent's actions can alter the facts of the possible worlds. For more details, see [36–40]. The simplest version of Dynamic Epistemic Logic is Public Announcement Logic (PAL), which is an extension of multi-agent Epistemic Logic, where dynamic operators are employed to model the epistemic consequences of announcements to the entire group of agents (see [17, 41, 42]).

Definition 3.2.8 (Language of PAL)**.** Let Φ be a set of propositional variables and $Ag = \{1, \ldots, n\}$ be a finite set of agents. The epistemic language is defined inductively through the following grammar, in BNF:

$$\phi ::= \top \mid p \mid \neg\phi \mid (\phi \wedge \phi) \mid \mathsf{K}_i\phi \mid \mathsf{D}_A\phi \mid [\phi]\phi.$$

where $p \in \Phi$, $i \in Ag$ and $A \subseteq Ag$.

The intended meaning of $\mathsf{K}_i\phi$ is "agent i knows ϕ". $\mathsf{D}_A\phi$ is the distributed knowledge of ϕ among agents in A, i.e., if the agents pulled their knowledge altogether, they would know that ϕ holds. The formula $[\psi]\phi$ means that after a truthful announcement of ψ, ϕ holds.

An inference system for PAL can be found in [43].

For semantics, we will use the Kripke models. A Kripke model for Epistemic Logic is tuples $\mathcal{M} = (W, R_1, \cdots, R_n, V)$, where $W = \{w_0, w_1, \cdots, w_k\}$ is a set of worlds or states, $R_i \subseteq W \times W$ is the *equivalence* relation for every agent i, and $V : W \to 2^{\Phi}$ is the evaluation function. The fact that R_i is an equivalence relation means that the agent i cannot tell states w and w' apart, when wR_iw'.

As usual, the truth of complex formulas is defined inductively:

Definition 3.2.9. Let $\mathcal{M} = (W, R_1, \ldots, R_n, V)$ be model for PAL. We say that ϕ is true in $w \in W$, written $\mathcal{M}, w \models \phi$, when:

- $\mathcal{M}, w \models p \quad \text{iff} \quad p \in V(w)$;
- $\mathcal{M}, w \models \neg\phi \quad \text{iff} \quad \mathcal{M}, w \not\models \phi$;
- $\mathcal{M}, w \models \phi \wedge \psi \quad \text{iff} \quad \mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$;
- $\mathcal{M}, w \models \mathsf{K}_i\phi \quad \text{iff} \quad \forall v \in R_i(w),\ \mathcal{M}, v \models \phi$, where $R_i(w) = \{w' \mid wR_iw'\}$;
- $\mathcal{M}, w \models \mathsf{D}_A\phi \quad \text{iff} \quad \forall v \in R_{\mathsf{D}_A}(w),\ \mathcal{M}, v \models \phi$, where $R_{\mathsf{D}_A} := \bigcap_{i \in A} R_i$;
- $\mathcal{M}, w \models [\psi]\phi \quad \text{iff} \quad \mathcal{M}, w \models \psi$ implies $\mathcal{M}^\psi, w \models \phi$,

where $\mathcal{M}^\psi$ is the updated Kripke model $\mathcal{M}$ by the announcement ψ in which $W^{\mathcal{M}^\psi} := \{w \in W \mid (\mathcal{M}, w) \models \psi\}$, the relation $R_i^{\mathcal{M}^\psi} := R_i \cap (W^{\mathcal{M}^\psi} \times W^{\mathcal{M}^\psi})$. Moreover, the valuation $V^{\mathcal{M}^\psi}$ restricts the valuation V to $W^{\mathcal{M}^\psi}$ [44].

Here, we concentrate on the interplay between knowledge and epistemic action, which only modifies the agents' knowledge while leaving the facts unchanged. So, $[\psi]$ is an operator that takes us to a new model consisting only of those worlds where ψ has been rendered as true. Therefore, after the announcement of ψ, no agent considers worlds where ψ was false. Hence, we should evaluate formulas in the sub-model $\mathcal{M}^\psi$.

Note that the operator D_A can be interpreted as a necessity operator of the relation on R_{D_A}.

Considering all such Kripke models, the set of all valid formulas obtained from these semantics is known as modal logic $\mathbf{S5}$, see [45].

3.2.3 Classifiers and Dynamic Epistemic Logics

Let us consider a classifier G, for which we will verify *safety*. Here we use *safety* (as the property) to simplify the demonstration, but any property could be defined arbitrarily. Let x be an input point.

Suppose the ϵ and $\mathcal{F}$ for defining neighborhood and manipulation sets are given. As defined in definition 3.2.6, the *safety* of x is verified in G, exactly when x is robust for G. Given that input point x is not robust, and some points in its neighborhood and manipulation set are classified into different classes. These classes can be considered alternative outputs. One can reduce these cases by employing multiple classifiers within a knowledge sharing multi-agent system.

Suppose we have a group $\mathbb{G}$ of classifiers that share knowledge about an input point. In case the intersection of these output classes (possible knowledge) is a single output class, such a system can verify properties by applying the shared knowledge. Note that the *safety* of x is verified in $\mathbb{G}$, exactly when $\bigcap_{G\in\mathbb{G}}[x]_G$ is a singleton, cf. definition 3.2.7.

To formalize sharing of knowledge by classifiers, we introduce The DEL- model $\mathfrak{M}_{\mathbb{G}} = (W, R_1, \ldots, R_n, V)$ of PAL as follows.

Suppose $\mathbb{G} = \{G_1, \ldots, G_n\}$ is a finite set of classifiers, X is the set of input points, and C is the set of all output classes. The set of propositional variables Φ will be interpreted by $X \times C$, i.e., each propositional variable is interpreted by a pair (x, c), where x is an input point and c is an output class. Each classifier will represent one agent in PAL.

The set of worlds W is all possible output results for the input values plus an extra world $\bar{c}$ i.e.,

$$W \coloneqq \{c \in C \mid \exists x \in X\ \exists G \in \mathbb{G} \text{ such that } c \in [\eta(x) \cup \mu(x)]_G\} \cup \{\bar{c}\}.$$

We add $\bar{c}$ to the set of worlds to find out "which class appears as an output of any agent?". We also notice that, since the classification of distinct input points is unrelated, we can consider a fixed input point x and create the model based on that. The final model will be the disjoint union of the models for each input.

So, suppose that x is a given fixed input point, then for any given $c \in C$, we define $R_i^x(c)$ and $V^x(c)$

abbreviate $R_i(c)$ and $V(c)$, respectively, as follows:

$$R_i(c) \coloneqq \begin{cases} \{c\} & c \notin [\eta(x) \cup \mu(x)]_{G_i} \\ \{c' \mid c' \in [\eta(x) \cup \mu(x)]_{G_i}\} \cup \{\bar{c}\} & c \in [\eta(x) \cup \mu(x)]_{G_i} \end{cases}$$

$$V(c) \coloneqq \{(x, c) \mid \exists G \in \mathbb{G} \text{ such that } c \in [\eta(x) \cup \mu(x)]_G\}.$$

$R_i(\bar{c})$ and $V(\bar{c})$ are defined as follows:

$$R_i(\bar{c}) \coloneqq \{c' \mid c' \in [\eta(x) \cup \mu(x)]_{G_i}\} \cup \{\bar{c}\},$$

$$V(\bar{c}) \coloneqq \{(x, c) \mid \exists G \in \mathbb{G} \text{ such that } c \in [\eta(x) \cup \mu(x)]_G\}.$$

Note that since the above definitions are given for a single fixed input x, then for any given $c \in C$, we have $V(c) = \{(x, c)\}$ or $V(c) = \varnothing$.

Now we can show that a point is robust for a classifier if and only if its interpretation is valid in $\mathfrak{M}_{\mathbb{G}}$.

Theorem 3.2.10. *Suppose that $\mathbb{G} = \{G\}$ is a single-classifier system, x is an input point and c is an output class of G. Then*

$$[\eta(x) \cup \mu(x)]_G = \{c\} \textit{ if and only if } \mathfrak{M}_{\mathbb{G}}, c \models \mathsf{K}p_c, \textit{ where } p_c \textit{ is interpreted by } (x, c).$$

Proof. Only-if part. Suppose $\mathfrak{M}_{\mathbb{G}}, c \models \mathsf{K}p_c$, where p_c is interpreted by (x, c). As we have only one classifier and c is an output class, we have $c \in [\eta(x) \cup \mu(x)]_G$. Suppose $c' \in [\eta(x) \cup \mu(x)]_G$. So, $c' \in R(c)$ and by hypothesis, we have $\mathfrak{M}_{\mathbb{G}}, c' \models p_c$, which means that $(x, c') \in V(c)$, which implies that $c = c'$.

If-part. Suppose $[\eta(x) \cup \mu(x)]_G = \{c\}$. Because c is an output of the classifier, $c \in W$, and as

it is the only output of classifier G on input x, we have $R(c) = \{c\}$. So, it suffices to show that $\mathfrak{M}_{\mathbb{G}}, c \models p_c$. As $\{(x, c)\} \in V(c)$, the statement is proved. □

Theorem 3.2.11. *Suppose that $\mathbb{G} = \{G_1, \ldots, G_n\}$, is a multi-classifier system such that $\bigcap_{G\in\mathbb{G}}[\eta(x) \cup \mu(x)]_G \neq \emptyset$, x is an input point and c is an output class for classifiers in $\mathbb{G}$, then*

$$\bigcap_{G\in\mathbb{G}}[\eta(x) \cup \mu(x)]_G = \{c\} \text{ if and only if } \mathfrak{M}_{\mathbb{G}}, \bar{c} \vDash \mathsf{D}_{\mathbb{G}} p_c, \text{ where } p_c \text{ is interpreted by } (x, c).$$

Proof. Only-if part. Suppose that $\mathfrak{M}_{\mathbb{G}}, \bar{c} \vDash \mathsf{D}_{\mathbb{G}} p_c$, then for all c' in $R_{D_{\mathbb{G}}}(\bar{c})$ we have $\mathfrak{M}_{\mathbb{G}}, c' \vDash p_c$. On the other hand, by the assumption, there is an element d in $\bigcap_{G\in\mathbb{G}}[\eta(x) \cup \mu(x)]_G$. It is enough to show that $d = c$. We have $d \in \bigcap_{G\in\mathbb{G}} R_{G_i}(\bar{c})$. Since $\mathfrak{M}_{\mathbb{G}}, d \vDash p_c$, then $V(d) = \{(x, c)\}$ which implies that $d = c$.

If-part. Suppose $\bigcap_{G\in\mathbb{G}}[\eta(x) \cup \mu(x)]_G = \{c\}$. Because c is an output of the classifiers, $c \in W$, and $c \in [\eta(x) \cup \mu(x)]_G$, for all $G \in \mathbb{G}$. On the other hand, as it is the only common output of all the classifiers, for any other output class, there exists a classifier G that does not have it as output. So, $R_{\mathsf{D}_{\mathbb{G}}}(c) = \{c, \bar{c}\}$. Therefore, it suffices to show that $\mathfrak{M}_{\mathbb{G}}, \bar{c} \models p_c$. As $\{(x, c)\} \in V(c) \cap V(\bar{c})$, the statement is proved. □

It is noteworthy that for each input point x_0, the possible worlds shall represent all possible epistemic states (i.e., they show all possible subsets of the output class set). These possible worlds can be filtered through the neighborhood and manipulation set $\mu(x_0) \cup \eta(x_0)$. Let C be the set of all output classes. If $\mid \mu(x_0) \cup \eta(x_0) \mid \geq \mid C \mid$, the number of all possible worlds shall be $2^{|C|}$ (generally, $\mid \mu(x_0) \cup \eta(x_0) \mid \gg \mid C \mid$ can be assumed). Therefore, the satisfaction of an atomic formula (x_0, c_i) in a state w means that world c_i has appeared as an output class of the classifier G_k, for some point of $\mu(x_0) \cup \eta(x_0)$. For instance, if c_i, c_j, and c_k appear in the output classes of G_k, for all members of $\mu(x_0) \cup \eta(x_0)$, then the *actual world* [40] can satisfy (x_0, c_i), (x_0, c_j), and (x_0, c_k).

3.2.4 2-Multi-classifiers Systems

In the DEL, the interpretation of external knowledge is also considered feasible. This means that in a MAS, besides the internally distributed knowledge, outer knowledge sharing can be investigated to reduce the possible worlds. For example, in an optical character recognition (OCR) problem limited to numbers, knowledge of the existence of a circle in the shape of the input image could reduce to four possible worlds (0, 6, 8, and 9, which have at least a loop in their shape). Another benefit of considering external knowledge is dividing a problem into smaller parts, solving more simplistic problems using various MAS, and sharing knowledge to conquer the problem. For this kind of divide-and-conquer algorithm in the MAS scenario, primarily, the problem should be broken down into simpler parts, and the rule of division should be collected (here, rules are the restriction applied to the process of combination of the divisions), where each division can be solved with a MAS of classifiers. Secondly, all MASs should publicly announce their remaining possible worlds as external knowledge. Next, considering the rules, impossible combinations should be avoided to reduce the number of possible worlds. Finally, when one and only one possible world exists, it can be verified deterministically. For example, assume two classifiers that are supposed to classify an input image, one of which classifies the background and another that does the same for the foreground. Next, suppose that the first classifier has identified the background as a city street, and the second is uncertain regarding the foreground between the existence of an elephant or a car. Using external knowledge that implies, "elephants do not wander on city streets", the only possible world for the foreground would be: there is a *car* on the street.

Formally, Assume that

$$\mathcal{M}_k = (W_k, R_{k,1}, \cdots R_{k,n_k}, V_k),$$

where $1 \leq k \leq n$, are given. The Kripke model:

$$\mathcal{M} = (W, R_1, \cdots, R_n, V)$$

The model that knowledge shares between classifiers is defined through the following components:

- $W = W_1 \times \cdots \times W_n$,
- $(w_{i_1}, \cdots, w_{i_n}) R_k (w_{j_1}, \cdots, w_{j_n})$, for $1 \leq k \leq n$, exactly when for all $l \neq k$ we have $w_{i_l} = w_{j_l}$ and there exists $1 \leq m \leq n_k$ such that $w_{i_k} R_{k,m} w_{j_k}$,
- $V(w_1, \cdots, w_n) = (V_1(w_1), \cdots, V_n(w_n))$.

3.3 Classifier's Collaboration

Suppose that a group of trusted classifiers work together towards achieving a more aware system (where trusted classifiers share the entire owned knowledge correctly). Algorithm 1 would take a classifier, an input point, and a neighborhood and manipulation function. This algorithm calculates all possible worlds for the classifier according to the input and its neighborhoods and manipulations. As mentioned before, knowledge will be generated using this function. In other words, first of all, the knowledge extraction process from one agent-input is developed, i.e., we collect all output classes of inputs in $\eta(x)$ related to the given classifier. In this algorithm, a classifier is a function $\mathcal{N}(x)$ for input x, and the output result of the function represents the output class of x. Consequently, $\mathcal{K}$ represents the knowledge of $\mathcal{N}$ with the definitions mentioned above. As a result, the output represents whether the investigated classifiers are robust for the input x. Furthermore, it includes the possible answers of the set $\eta(x)$ from the agent's perspective.

After obtaining the knowledge of each agent (i.e., ANN -as classifiers), algorithm 2 has been developed to aggregate all the knowledge of these agents. This knowledge also demonstrates the

Algorithm 1 The Classifier Knowledge Calculator (CKC) function shall calculate the knowledge produced by a classifier for an input point, using the neighborhood function and manipulation function

Let $\mathcal{N}(x) = c$ be the function of the considered a classifier and c be the result class

function CKC($\mathcal{N}, x_0, \eta, \mu$)
    ▷ $\mathcal{N}, x_0, \eta, \mu$ are a classifier, an input point, neighborhood function, and manipulation function respectively
    $\mathcal{K} \leftarrow \emptyset$
    for all $x \in \eta(x_0) \cup \mu(x_0)$ **do**
        $c \leftarrow \mathcal{N}(x)$ ▷ c represents the respective possible world
        if $c \notin \mathcal{K}$ **then**
            Add c to $\mathcal{K}$ set
    if $\mid \mathcal{K} \mid = 1$ **then**
        return 1, $\mathcal{K}$
    return 0, $\mathcal{K}$

possible worlds from the perspective of each agent. Herein, by determining the intersected knowledge of all classifiers in the MAS, the result of verification, and the aggregated knowledge from the group of agents can be measured. As an output, if the intersection results in one class, the input is considered verified. Otherwise, if more than one class exists in the intersection result, the input point cannot be verified. Although, the set of possible classes represents the possible verified outputs for the MAS. Finally, if an empty set returns as an output, inconsistency emerges in the MAS's agent knowledge, and the input cannot be verified for that particular case.

Algorithm 3 would aggregate all classifiers' knowledge and external knowledge. Here, $\mathcal{K}_{\mathcal{S}}$ shows the remaining possible worlds in which verification formulas can be satisfied if the cardinality of the remaining possible worlds equals one. In other words, $\mid \mathcal{K}_{\mathcal{S}} \mid$ represents the number of words connected to $\bar{w}$ with relations R_i, $i \in \mathcal{N}_{\mathcal{S}}$.

Algorithm 2 The MAS Knowledge Aggregator (MASKA) function will aggregate the knowledge produced by a group of classifiers for each input point using a neighborhood function and manipulation function.

function MASKA($\mathcal{N_S}, x_0, \eta, \mu$)
    ▷ $\mathcal{N_S}, x_0, \eta, \mu$ are a group of classifiers, an input point, neighborhood function, and manipulation function respectively
    $\mathcal{K_S} \leftarrow set\ of\ all\ output\ classes$
    for all $\mathcal{N} \in \mathcal{N_S}$ **do**
        $is\_Robust, \mathcal{K} \leftarrow$ CKC $(\mathcal{N}, x_0, \eta, \mu)$
        $\mathcal{K_S} \leftarrow \mathcal{K_S} \cap \mathcal{K}$
        if $\mathcal{K_S} = \emptyset$ **then**
            return 0, $\emptyset$
    if $\mid \mathcal{K_S} \mid = 1$ **then**
        return 1, $\mathcal{K_S}$
    return 0, $\mathcal{K_S}$

3.4 Examples

Reducing the error rate of classifiers is crucial in critical scenarios. In our method, a MAS of classifiers has been offered for this exact purpose. In the sequel, we will provide two types of examples explaining the details of our proposed model. In the first example, we demonstrate exactly how the knowledge set can be defined in the context of Dynamic Epistemic Logic. In the other examples, we demonstrate the usability of our model in real-world practical applications. To achieve this, we select three datasets to examine the effects. The selected datasets are Modified-Fashion-MNIST [46] (unbalanced 4 output classes), MNIST [47] (balanced 10 output classes), and Fruits-360 [48] (unbalanced 131 output class). A tool is also developed to apply the method in arbitrary classifiers and datasets.

Example 3.4.1. (Digit Recognition)

In this example, we will develop a scenario to clarify the approach mentioned in previous chapters. Herein, the input will lie in the domain of images. One of the digits 0 to 9 would appear

Algorithm 3 The MAS Knowledge Sharing (MASKS) function shall aggregate the knowledge that is produced by an external system

function MASKS($\mathcal{N}_\mathcal{S}, x_0, \eta, \mu$)
▷ $\mathcal{N}_\mathcal{S}, x_0, \eta, \mu$ are a group of classifiers, an input point, neighborhood function, and manipulation function respectively
    $\mathcal{K}_\mathcal{S} \leftarrow \emptyset$
    for all $\mathcal{N} \in \mathcal{N}_\mathcal{S}$ **do**
        is_Robust, $\mathcal{K} \leftarrow$CKC($\mathcal{N}, x_0, \eta, \mu$)
        $\mathcal{K}_\mathcal{S} \leftarrow \mathcal{K}_\mathcal{S} \cap \mathcal{K}$
        if $\mathcal{K}_\mathcal{S} = \emptyset$ **then**
            return 0, $\emptyset$
    if $\mid \mathcal{K}_\mathcal{S} \mid = 1$ **then**
        ▷ Check whether the knowledge can verify the input, or if more knowledge is needed
        return 1, $\mathcal{K}_\mathcal{S}$
    for all $\mathcal{M} \in$ All knowledge sources **do**
        is_Robust, $\mathcal{K} \leftarrow$ Announced knowledge
    ▷ The external knowledge must be written in the DEL formula, where possible worlds are ones in which the formula is satisfied.
        $\mathcal{K}_\mathcal{S} \leftarrow \mathcal{K}_\mathcal{S} \cap \mathcal{K}$
        if $\mathcal{K}_\mathcal{S} = \emptyset$ **then**
            return 0, $\emptyset$
    if $\mid \mathcal{K}_\mathcal{S} \mid = 1$ **then**
        return 1, $\mathcal{K}_\mathcal{S}$
    return 0, $\mathcal{K}_\mathcal{S}$

in the image. The model has eleven worlds $W = \{w_0, \cdots, w_9\} \cup \bar{w}$. Each world w_i represents the existence of one digit and $\bar{w}$, where $V_\Phi(w_i) = \{(x, i)\}$. The world $\bar{w}$ could not be a possible answer because more than one digit is true in it. This world was added to check the satisfaction of the verification formula (as mentioned in previous sections) $V_\Phi(\bar{w}) = \{(x, 0), \ldots, (x, 9)\}$. For a fixed input of x, we denote the atomic formula (x, i) by p_i. Let the figure of the digit "0" be an input image, and let a multi-agent system A with three given agents $A = \{A_0, A_1, A_2\}$ as classifiers. Assume that A_0 shows "0", "6", "8", and "9" are possible answers for the input x_0. Figure 3.1 depicts the possible worlds and relations of the model for A_0. The same input for A_1 results "0", "2", "4", "6", and "8" are possible answers. The intersection of these two agents' possible answers will be w_0, w_6, and w_8, fig 3.2. The last agent, A_2, possible results are w_0, w_2, and w_9. Thus it ignores the world w_6 and w_8 as possible results. So, the model verifies the properties for the answer that the input case is "0" 3.3. We have $\mathfrak{M}_\mathbb{G}, \bar{w} \vDash \mathsf{D}_A p_0$.

Example 3.4.2. (Modified Fashion MNIST: The unbalanced-4-output-class example)

The Fashion-MNIST is a dataset of clothes images that contains a training set of 60,000 examples and a test set of 10,000. Each image is a 28 by 28 matrix, associated with a label from ten classes. In this research, we modified the ten output classes of Fashion-MNIST ("T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", and "Ankle boot") to four ("Top cover", "Bottom cover", "Shoes", and "Bag" in which "Top cover" contains "T-shirt/top", "Pullover", "Dress", "Coat", and "Shirt"; "Bottom cover" contains "Trouser"; "Shoes" contains "Sandal", "Sneaker", and "Ankle boot"; and "Bag" contains "Bag";) in order to reach an unbalanced dataset with four output classes. Here, the neighborhood and manipulation set (property to verify) is a set of 100 random salt-and-pepper noises (noises are created before the classification process and fixed for all inputs and classifiers). Although creating a more meaningful neighborhood and manipulation set could boost our results, we decided to select it

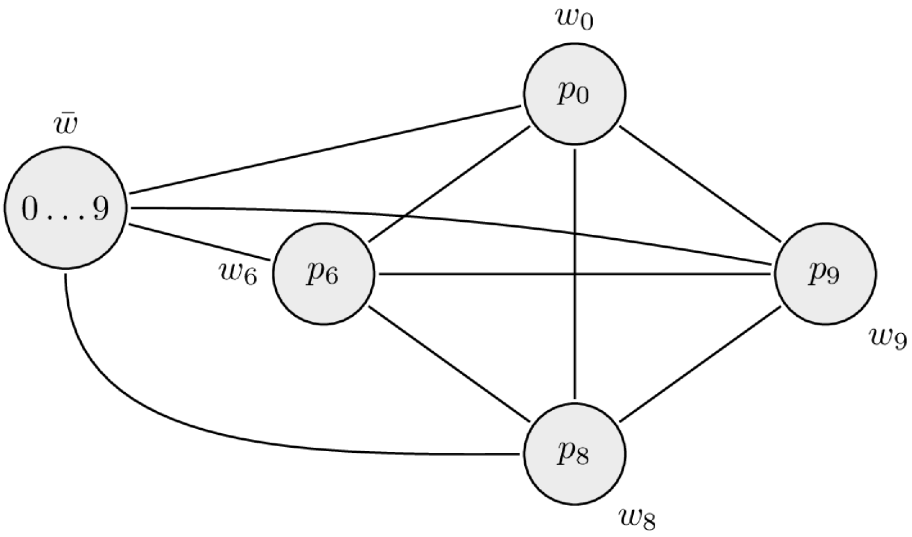


Figure 3.1: The Epistemic model, considering the classifier A_0's knowledge.
$W^{(p_0 \vee p_6 \vee p_8 \vee p_9)} = \{w_0, w_6, w_8, w_9, \bar{w}\}$;
$R_{A_0} = \{(w_0, w_0), (w_6, w_6), (w_8, w_8), (w_0, w_8), (w_8, w_0), (w_0, w_9), (w_9, w_0),$
$(w_6, w_8), (w_8, w_6), (w_6, w_9), (w_9, w_6), (w_8, w_9), (w_9, w_8), (\bar{w}, \bar{w}), (\bar{w}, w_0),$
$(w_0, \bar{w}), (\bar{w}, w_6), (w_6, \bar{w}), (\bar{w}, w_8), (w_8, \bar{w}), (\bar{w}, w_9), (w_9, \bar{w})\}$;

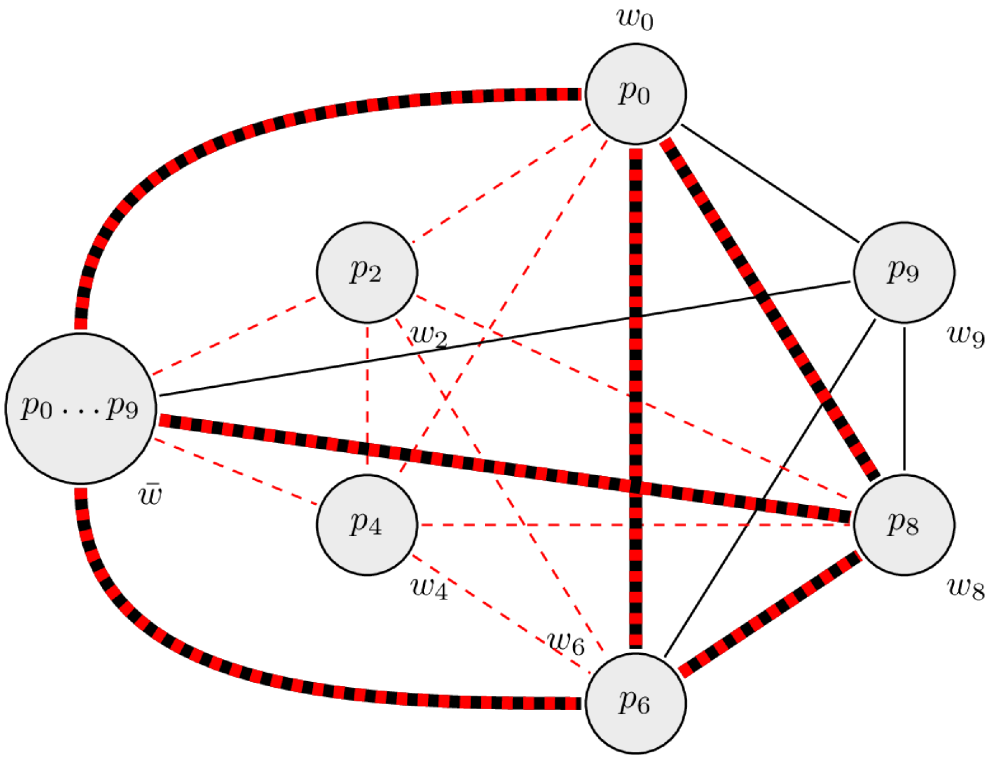


Figure 3.2: The updated model, after A_1's announcement. The red-black relations are intersected relations of the model, the possible worlds after the announcement will be $w_0,\ w_6,\ and\ w_8$

$W^{(p_0 \vee p_6 \vee p_8 \vee p_9) \wedge (p_0 \vee p_2 \vee p_4 \vee p_6 \vee p_8)} = \{w_0, w_6, w_8, \bar{w}\}$;

$R_{A_0} \cap R_{A_1} = \{(w_0, w_0), (w_6, w_6), (w_8, w_8), (w_0, w_6), (w_6, w_0), (w_0, w_8), (w_8, w_0),$
$(w_6, w_8), (w_8, w_6), (\bar{w}, \bar{w}), (\bar{w}, w_0), (w_0, \bar{w}), (\bar{w}, w_6), (w_6, \bar{w}), (w_8, \bar{w}), (\bar{w}, w_8)\}$;

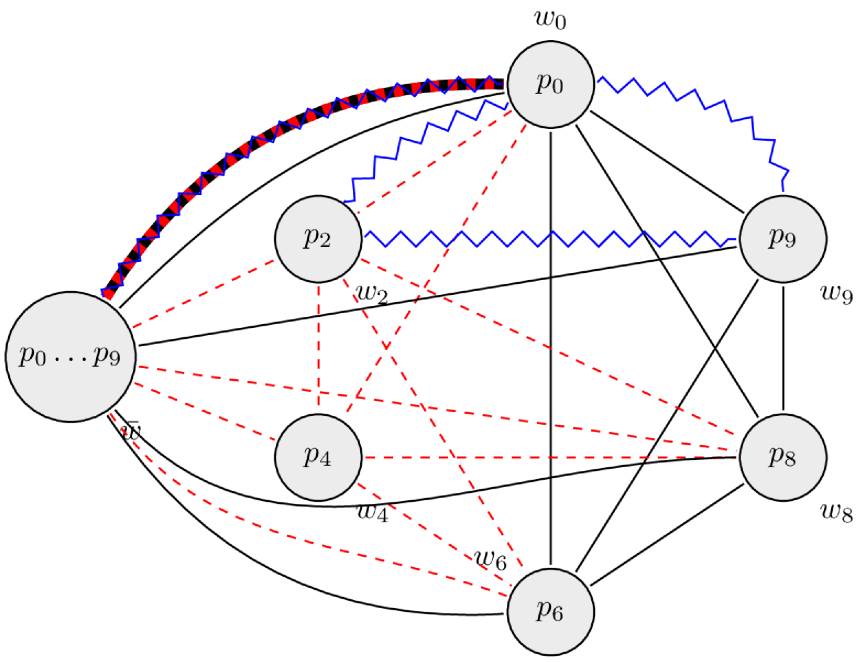


Figure 3.3: The updated model, after A_2's announcement. The red-black with blue zigzag relation is intersected relation of the model, w_0 would be the possible world after the announcement
$W^{(p_0 \vee p_6 \vee p_8 \vee p_9) \wedge (p_0 \vee p_2 \vee p_4 \vee p_6 \vee p_8) \wedge (p_0 \vee p_2 \vee p_9)} = \{w_0, \bar{w}\}$;
$R_{A_0} \cap R_{A_1} \cap R_{A_2} = \{(w_0, w_0), (\bar{w}, \bar{w}), (\bar{w}, w_0), (w_0, \bar{w})\}$;

randomly to simulate errors in selecting neighborhood and manipulation for real applications. In this example, all classifiers' accuracy is about 99 percent. This example could examine our method for data sets with few output classes. We execute the MASKS for single to 1000 classifiers (see table 3.1). Considering the neighborhood and manipulation set, the number of wrongs verified answers for a single classifier is 89, which means 89 out of 10000 cases are robust (or verified), but the presented answer is incorrect. These wrong verified answers occur when the classifier cannot correctly classify the input image and cannot find the correct output for the input's neighborhood and manipulations. The value of wrong verified answers is close to the classifier's error value for the original input (without neighborhoods and manipulations). Thus, in this case, the neighborhood and manipulation set did not select ideally because all neighborhoods and manipulation for the classifier resulted in a wrong answer. As mentioned before, choosing a neighborhood and manipulation set is very complex. Here, wrong verified answers could be reduced by increasing the number of classifiers (with the same neighborhood and manipulation set). In this example, increasing the number of classifiers could reduce the wrong verified answers caused mainly by improper neighborhood and manipulation set selection. As we can see, the methods reduce the errors by about 80.5 percent. The detail of results for 1 to 1000 classifiers are represented in fig 3.4.

Example 3.4.3. (MNIST: The unbalanced-10-output-class example)

The MNIST is a dataset of handwriting digit images that contains a training set of 60,000 examples and a test set of 10,000. Each image is a 28 by 28 matrix, associated with a label from ten classes. Here, the neighborhood and manipulation set (property to verify) is a set of 100 noisy salt-and-pepper images (noises are created before the classification process and fixed for all inputs and classifiers); and the classifiers' accuracy is about 99%. We execute the MASKS for a single to 608 agents (see table 3.1). In this example, the wrong verified answers for a single classifier are

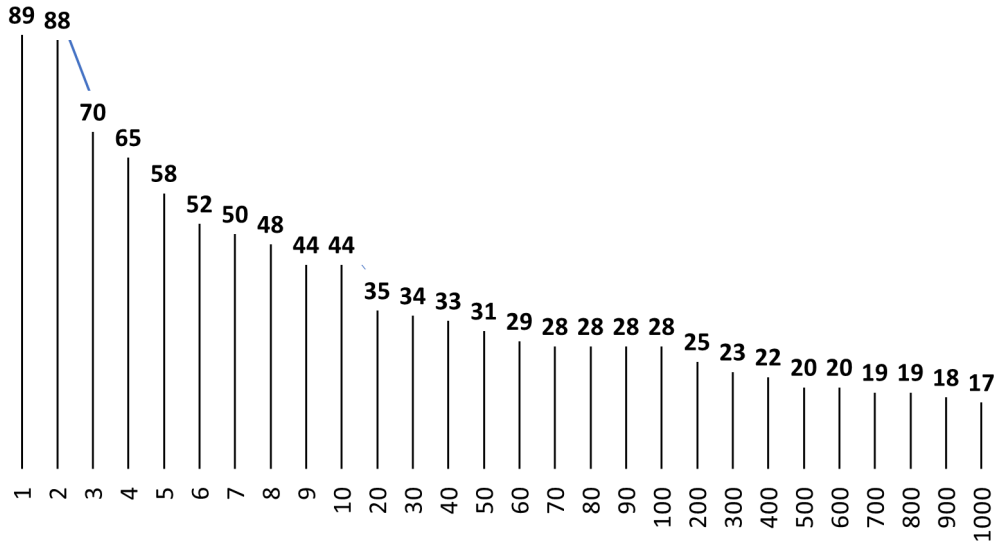


Figure 3.4: Fashion-MNIST: wrong verified answers for various numbers of classifiers.

less than in the previous example with the same accuracy. This example shows that the random noises work better, but it is not ideal. Perhaps, increasing the number of output classes is a cause. In this example, increasing the number of classifiers could reduce wrong verified answers faster. As we can see here, the method reduces errors by about 95.75 percent with $\sim$ 700 classifiers. In this example, increasing the number of classifiers could reduce the wrong verified answers caused by the improper selection of neighborhood and manipulation set and classifier's accuracy. The detailed results for 1 to 608 agents are represented in fig 3.5.

Example 3.4.4. (Fruit-360: The unbalanced-131-output-class example)

Fruits-360 is a dataset of fruits and vegetable images; the set contains 67692 training and 22688 test images (100 by 100) of 131 fruit and vegetable label classes. Here, the neighborhood and manipulation set is a set of 50 noisy Salt-and-pepper images; and the classifiers' accuracy is about 97 percent. We execute the MASKS for single to 74 classifiers (see table 3.1). In this example, the

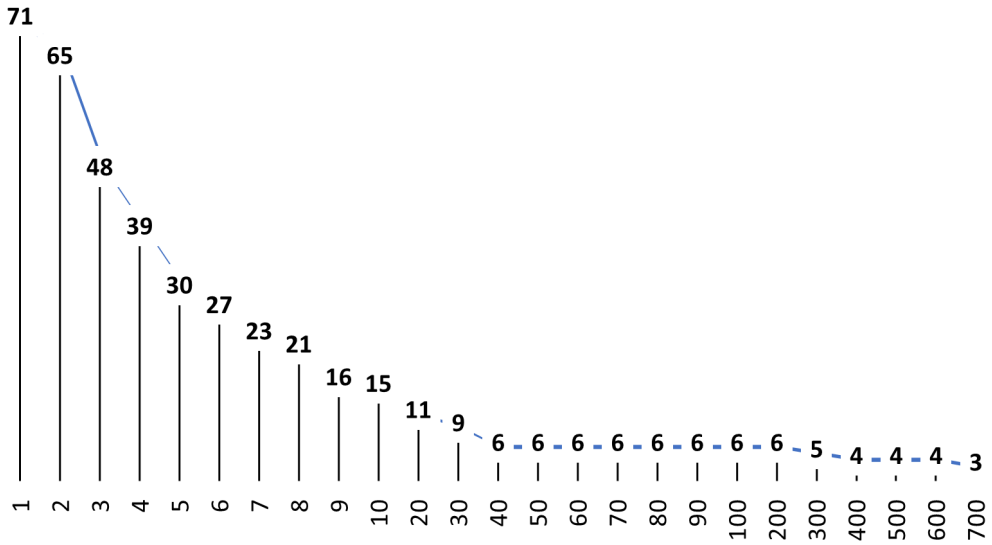


Figure 3.5: MNIST: wrong verified answers for various numbers of classifiers.

percentage of wrong verified answers of verified cases for a single classifier is even less than in the previous example. For a single classifier, wrong verified cases are about thirds of error cases (errors of original inputs without neighborhoods and manipulations). It shows that the random noises are working better in this example. As we can see here, the number of wrong verified cases for two classifiers is more than a single classifier; this means two classifiers are agreed for more wrong verified answers. Thus, it seems the impact of accuracy is significant. The wrong verified cases reduce by increasing classifiers. As we can see here, all errors disappeared in verified cases. The detailed results for 1 to 74 agents are represented in fig 3.6.

Remark 3.4.5. (Conclusion of examples)

In these examples, a MAS of ANNs (as classifiers) has been developed to classify these datasets. As mentioned before, the input picture, the manipulation, and the neighborhood set will be fed into classifiers. If a single output class is represented as output, this input is verified (for the manipulation

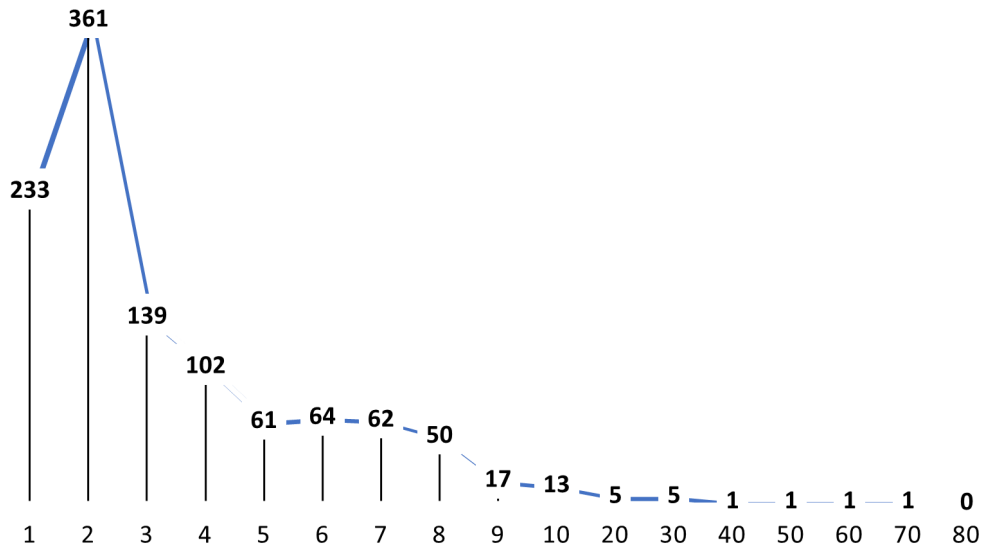


Figure 3.6: Fruit-360: wrong verified answers for various numbers of classifiers.

and the neighborhood set). Otherwise, the algorithm does not represent any output for it. As is shown in figs 3.4, 3.5, and 3.6, for a system with one classifier, the number of verified input cases that are wrongly classified is much less than in the multi-classifiers variant. These examples with one classifier are the same as the method developed in [7], in which neighborhood and manipulations are applied just in input (not in deeper layers). Compared with a single classifier, it appears that an increased number of classifiers could reduce errors. Even though the number of correctly predicted cases also decreases with the increasing number of classifiers, fig 3.7, 3.8, and 3.9 are ratios of "wrong verified cases" by "correct verified cases", which shows that the trend of decreasing the "correct verified cases" is not as steep as the slope of the "wrong verified cases". Note that the model will decide on verified cases; thus, the predicted cases are verified ones. Other cases (unverified ones) did not count as "correct predicted cases" or "wrong predicted cases". Here, the unverified cases could be categorized as high-risk inputs. These inputs are the ones in which classifiers in the

developed MAS do not show a good confidence level regarding the classification.

As can be seen, when we increase the number of classifiers, the error reduction is significant. Therefore, the method is well suitable for classifying critical cases (i.e., medical). In such cases, multiple classifiers could be applied to solve a classification problem with a small error ratio for verified cases. Unverified cases could be classified by a more trustworthy classifier (i.e., doctors in medical cases). Moreover, as can be seen, defining a suitable property is complex and inaccurate. Thus, in these examples, we did not define a very complex property (random noise as neighborhood and manipulation set) to show that even with an inaccurate property, MASKS could reduce errors.

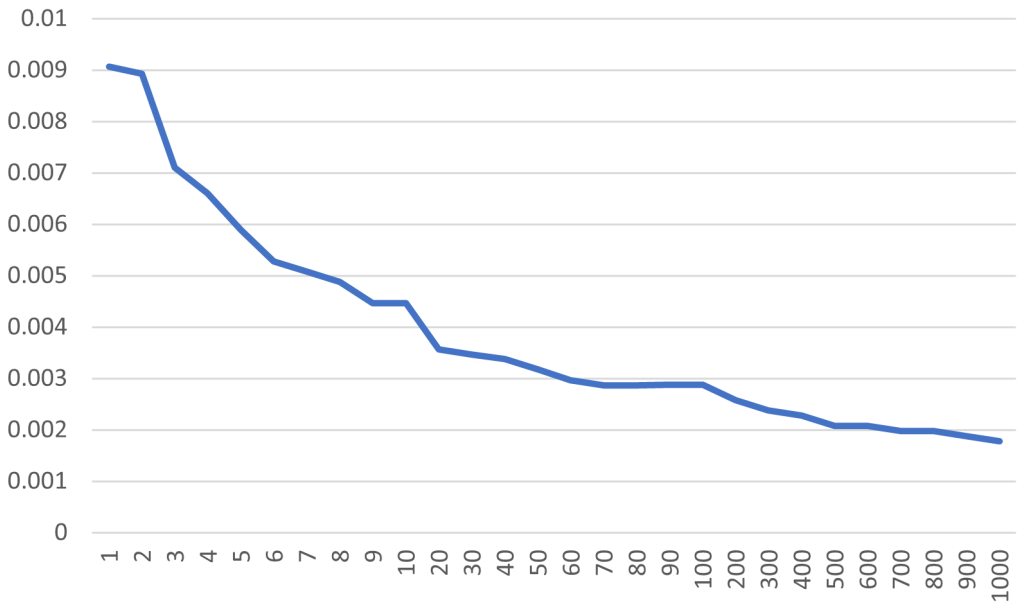


Figure 3.7: Fashion-MNIST: "wrong verified cases"/"correct verified cases" rate for various number of classifiers.

Remark 3.4.6. (Difficulties) It is known that the verification method goals are developed to satisfy specific predefined properties. However, the classification generally aims to reduce errors. The verification methods could help classifiers to be self-aware if properties are defined correctly.

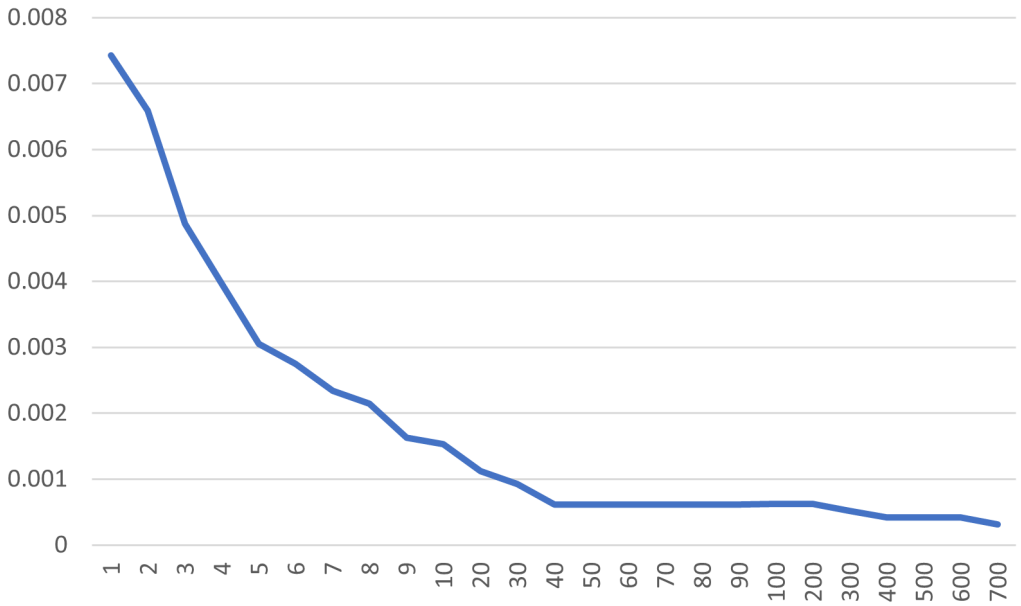


Figure 3.8: MNIST: "wrong verified cases"/"correct verified cases" rate for various number of classifiers.

Nevertheless, defining such properties is difficult in many cases. For example, safety properties are applied in many image classification methods, but defining the ϵ for neighborhood and the function f in manipulation is not trivial. Examples 3.4.2,3.4.3, and 3.4.4 will show how multi-classifiers could reduce the misclassification caused by improper safety properties. Accordingly, our developed systems of classifiers may have miscalculation problems, if faced with the following scenarios:

1. **Wrongly verifying input points:** This occurs when an input point has been considered robust in a wrong class (i.e., the input point and all members of the knowledge set lie in the wrong output class). This failure may happen when the inaccuracy of the partitioning algorithm of the classifiers is more than the broadness of the neighborhood and the manipulation set. The root cause of this can be in applying low classification accuracy, selecting an inadequately

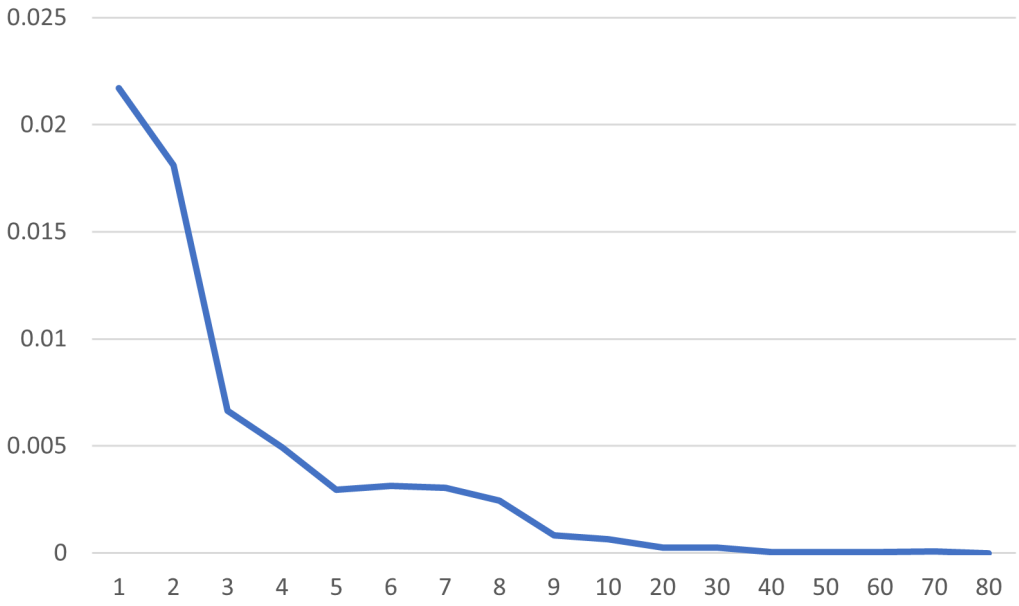


Figure 3.9: Fruit-360: "wrong verified cases"/"correct verified cases" rate for a various number of classifiers.

sized set for the neighborhood and manipulation functions, or it can result from an insufficient number of classifiers employed in the MAS.

Note: If the neighborhood and manipulation set contains the input point, this problem could be considered ensemble learning. The probability of such error depends on every single classifier of the MAS (see Dietterich *et. al.,* [49]).

2. **Correct answers not verified:** Sometimes, the MAS does not verify input points correctly classified with the classifiers. In this scenario, the input point should be located near the partitioning boundaries of all the classifiers of the MAS. In other words, all classifiers wrongly classified similar inputs (elements of the neighborhood and manipulation set) into the same class. In this case, all classifiers present at least two classes as output classes. A subset of classes can be collected by examining every intersection of output classes for

classifiers' outputs. This subset is considered common among outputs for each classifier. Although the input point cannot be verified for a single result, it is acknowledged that the verified element resides in a subset of the represented results.

Remark 3.4.7. (Comparing with a voting system)

Although the primary purpose of MASKS is to verify property for a multi-classifier, we developed a "major voting" system to explain how our method can significantly reduce error [50]. Here, classifiers are the same as those applied in examples 3.4.2, 3.4.3, and 3.4.4. Figs 3.10, 3.11, and 3.12 show that a voting system with the same classifiers is compared with the developed MASKS algorithm. In the Major Voting system, unverified cases have more than one class with major votes; others are counted as verified. As can be seen, MASKS's error reduction is more significant. It can show that interpreting the knowledge of multiple classifiers in a classification problem with distributed knowledge seems to have better results than major voting in these cases (i.e., in table 3.1, for the Fashion-MNIST dataset, for 1000 classifiers, the number of errors in the MASKS is 4 times less than in the voting system).

3.5 Conclusion and Future Work

Classification is one of the primary tasks of Artificial Intelligence (AI). Nowadays, many classifiers are applied in critical applications, in which errors would cause serious impacts. This chapter aims to develop a multi-agent verification method to reduce errors by integrating multiple classifiers. Here, it is shown that a multi-agent scenario could perform better in order to reduce errors. To do this, primarily, a property has been defined to present the knowledge of the classifiers. Next, a multi-agent system was designed to investigate these multiple classifiers. Then, a Dynamic Epistemic Logic-based method was developed for reasoning about the aggregation of distributed knowledge.

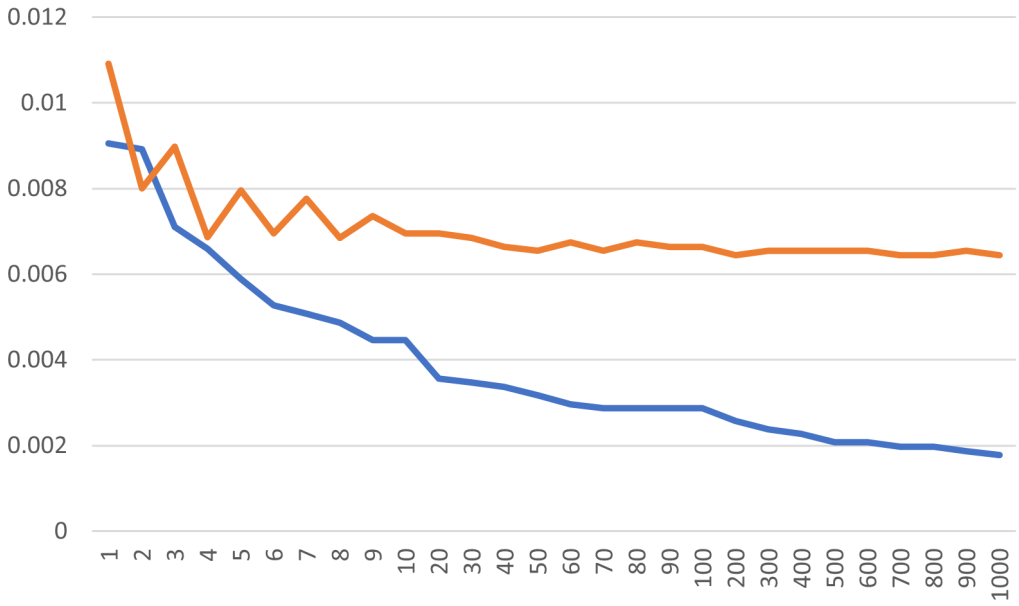


Figure 3.10: Fashion-MNIST: Comparing wrong verified cases of various numbers of classifiers for MASKS and Voting.

This knowledge was acquired through separate classifiers and external information sources. Finally, it was shown that aggregated knowledge might lead to refining output results. As a result, the model could verify the model for a specific input, if the knowledge of the entire system satisfies its correctness. In other words, the system could distinguish robust answers. To conclude, a multi-agent system for the knowledge sharing (MASKS) algorithm has been proposed for the model above. This proposed method was applied to the Fashion-MNIST, MNIST, and Fruit-360 datasets. As a result, the error rate of the entire system dropped significantly.

Looking into the future, we aim to develop an approach that can model time-series classifiers (i.e., for *recurrent neural networks* or *reinforcement learning*) for real-time verifying approaches. Moreover, we will develop a tool to verify any multi-agent system's inputs and check whether an input point can be verified in the system. This tool should be able to manipulate knowledge sharing

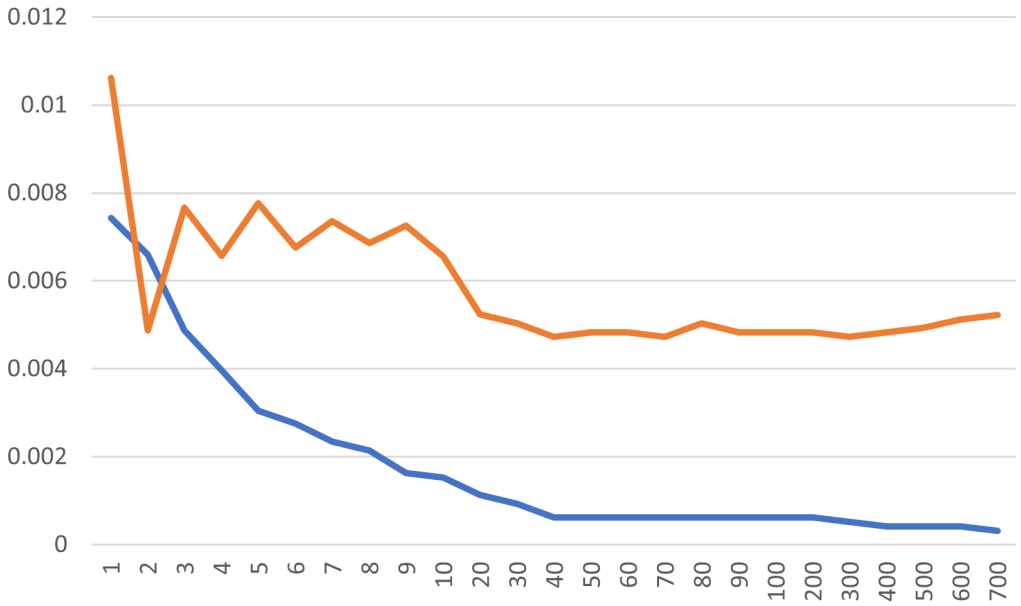


Figure 3.11: MNIST: Comparing wrong verified cases of various numbers of classifiers for MASKS and Voting.

in trusted or untrusted networks. To enhance the performance of this tool, Fuzzy Logic could be applied to avoid state space explosion for classifiers, especially where more than two output classes exist.

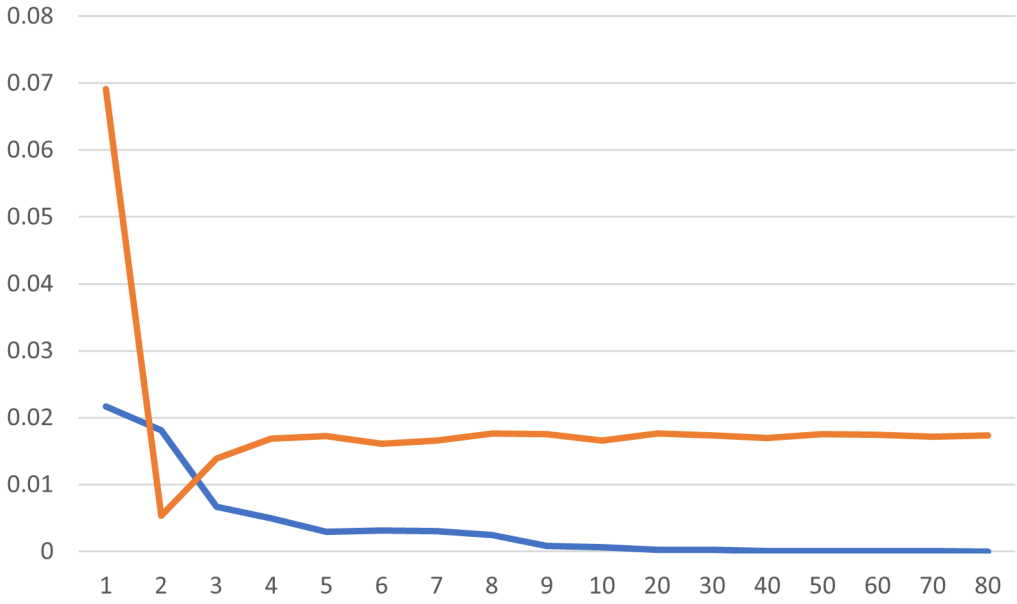


Figure 3.12: Fruit-360: Comparing wrong verified cases of various numbers of classifiers for MASKS and Voting.

Dataset	number of classifiers	verified		unverified	Method
		correct	Wrong		
Fashion-MNIST	1	9821	89	90	MASKS
Fashion-MNIST	1000	9588	17	395	MASKS
Fashion-MNIST	1	9892	108	0	Major Voting
Fashion-MNIST	1000	9936	64	0	Major Voting
MNIST	1	9556	71	373	MASKS
MNIST	608	9519	3	478	MASKS
MNIST	1	9895	105	0	Major Voting
MNIST	608	9949	51	0	Major Voting
Fruit-360	1	10730	233	11725	MASKS
Fruit-360	74	15850	0	6838	MASKS
Fruit-360	1	21221	1467	0	Major Voting
Fruit-360	74	22288	386	14	Major Voting

Table 3.1: Comparing MASKS with Major Voting method for single and multi-classifiers scenarios.

Chapter 4

Verifying data-streams

In this chapter, a formal transition system model is presented called Linear Temporal Public Announcement Logic (LTPAL) to extract knowledge in a classification process. The model combines Public Announcement Logic (PAL) and Linear Temporal Logic (LTL). For this purpose, first, an epistemic logic model is created to capture information gathered by classifiers in single-framed data input. Next, using LTL, classifiers are considered for data stream inputs. Then, a verification method is proposed for such data streams. Finally, we formalize natural language properties in LTPAL with a video-stream object detection sample.

4.1 Introduction

Nowadays, classification is inseparable from real-world applications. The presence of errors increases the necessity for better interpretation and comprehension of such algorithms. Due to the probabilistic nature of most of these approaches, the reasoning about results remains complicated.

Reasoning is one of the most studied and accepted dimensions of AI [51], [52], [53], [54]). Furthermore, it inspires researchers to target Visual data understanding, question answering,

epidemic broadcasting protocols, and natural language queries [55], [56]. For instance, a question-answering (QA) reasoning system was developed by Bauer *et al.*, who introduced an algorithm to select the best paths of commonsense knowledge and get the whole inference required for QA [57]. The method works fine for specific applications. However, it provides no general solution for QA problems. Based on the systematic analysis of popular knowledge resources and the knowledge-integration methods, Ma *et al.* developed a modeling solution [58]. The method, called "non-extractive commonsense QA", employs ConceptNet (Speer *et al.* [59]) and the most recently introduced ATOMIC (Sap *et al.* [60]). More presently, based on human intelligence, the "CoLlision Events for Video REpresentation and Reasoning" method (CLEVRER) was developed by Yi and coworkers [61], which is a reasoning system for video streams. In this method, unlike former studies, "causal structures" are also taken into account. Moreover, this model can answer four main varieties of questions: *descriptive* (e.g., "what color"), *explanatory* (e.g., "what's responsible for"), *predictive* (e.g., "what will happen next"), and *counterfactual* (e.g., "what if"). Here, the solution for extracting features from the video frames is ResNet-50, a well-known classifier [62].

Thanks to the characteristics of the reasoning systems and the power of temporal logic in formalizing natural language, the model developed by Fong *et al.* deals with reasoning systems [63]. From another perspective, Metric Temporal Logic (MTL), an extension of Linear Temporal Logic (LTL), was examined to handle the stochastic state information [64]. Concerning typical AI-based problems, the employment of modal logic would be attractive. Recently, Dehkordi *et al.* developed an epistemic-based model for the extraction and aggregation of knowledge from classifiers [13]. The model works perfectly for single-framed data. However, it addresses no data stream inputs. Dehkordi *et al.* [13], developed Algorithms 4 and 5 for knowledge-sharing modeling in a multi-classifier scenario. In a network of trusted classifiers

(which share their knowledge correctly), the process of knowledge extraction from one input for an agent is developed in Algorithm 4. In other words, all output classes of the input, presented in the neighborhood set of $\eta_\epsilon(x)$ (here referred to as $\eta(x)$), were collected. $\eta(x)$ is a set of inputs in ϵ vicinity (based on distance function $d(x_0, x)$). Here, the classifier function $\mathcal{N}(x)$ yields the respecting output classes by $\eta(x)$ input. Its knowledge contains the set of output classes of $\mathcal{N}(x)$ which is represented by $\mathcal{K}$ (as the knowledge set of the classifier). The output is robust if the size of the set of output classes of $\mathcal{N}(x)$ is one. Therefore, from the agent's perspective, the output concludes the robustness of x. An agent's perspective is knowledge, which is accessible by that agent. Let $\mathbf{N}$ be the time complexity of each classifier; the time complexity of Algorithm 4 would be $O(\mathbf{N}\times \mid \eta(x_0) \cup \mu(x_0) \mid \times \mid W \mid)$, where "$\mid A \mid$" is the cardinality of the set A.

Algorithm 4 The Classifier Knowledge Calculator (CKC) function will calculate the knowledge produced by a classifier for an input point, using the neighborhood function and manipulation function.

Let $\mathcal{N}(x) = c$ be the function of the considered classifier and c be the result class

1: **function** CKC($\mathcal{N}, x_0, \eta, \mu$)
2: ▷ $\mathcal{N}, x_0, \eta, \mu$ are a classifier, an input point, neighborhood function, and manipulation function, respectively
3: $\mathcal{K} \leftarrow \emptyset$
4: **for all** $x \in \eta(x_0) \cup \mu(x_0)$ **do**
5: $c \leftarrow \mathcal{N}(x)$ ▷ c represents the respective possible world
6: **if** $c \notin \mathcal{K}$ **then**
7: Add c to $\mathcal{K}$ set
8: **if** $\mid \mathcal{K} \mid = 1$ **then**
9: **return** 1, $\mathcal{K}$
10: **return** 0, $\mathcal{K}$

In Algorithm 5, all captured knowledge is aggregated, and the reasoning is applied. After finding the intersection, if it contains just one class, the input is verified. Here, the MAS's knowledge is inconsistent if the output is an empty set. In the case of more than one output class, agents are not capable of concluding the answer. However, they know the answer is among these classes. The time

complexity of $\mathcal{K}_{\mathcal{S}} \leftarrow \mathcal{K}_{\mathcal{S}} \cap \mathcal{K}$ is $O(\mid W \mid)$; thus, Algorithm 5 would run in $O(\mathbf{N}\times \mid \eta(x_0) \cup \mu(x_0) \mid \times \mid W \mid \times \mid \mathcal{N}_{\mathcal{S}} \mid)$. This algorithm models the single-framed data knowledge capturing in a multi-agent scenario.

Algorithm 5 The MAS Knowledge Sharing (MASKS) function will reason about the knowledge produced by classifiers.

function MASKS($\mathcal{N}_{\mathcal{S}}, x_0, \eta, \mu$)
▷ $\mathcal{N}_{\mathcal{S}}, x_0, \eta, \mu$ are a group of classifiers, an input point, neighborhood function, and manipulation function, respectively
  $\mathcal{K}_{\mathcal{S}} \leftarrow set\ of\ all\ output\ classes$
  for all $\mathcal{N} \in \mathcal{N}_{\mathcal{S}}$ **do**
    $is_Robust, \mathcal{K} \leftarrow$CKC($\mathcal{N}, x_0, \eta, \mu$)
    $\mathcal{K}_{\mathcal{S}} \leftarrow \mathcal{K}_{\mathcal{S}} \cap \mathcal{K}$
    if $\mathcal{K}_{\mathcal{S}} = \emptyset$ **then**
      return 0, $\emptyset$
  if $\mid \mathcal{K}_{\mathcal{S}} \mid = 1$ **then**
    ▷ Check whether the knowledge can verify the input or if more knowledge is needed
    return 1, $\mathcal{K}_{\mathcal{S}}$
  return 0, $\mathcal{K}_{\mathcal{S}}$

This chapter aims to extend the expressiveness of the model developed by Dehkordi *et al.* [13] to present data streams. Moreover, a flexible reasoning system is suggested to render the knowledge achieved by classifiers. Here, the model is applied to the existing classifiers to translate the information attained from the results. Furthermore, the ability of the model to handle multi-knowledge flow in a multi-classifier scenario is assessed. For this purpose, various collaboration scenarios of classifiers (single agents or a group of agents) are studied. For single-framed data, our method verifies the properties (formulas) using the definition of "verification" mentioned in Dehkordi *et al.* [13]. Certain questions (in human language) are translated to the developed temporal formula (as properties) to check the adaptation and applicability of the proposed model. By checking the satisfiability of the temporal formula, the "verification of data streams" is presented.

The main contributions of this paper to the development of a reasoning model on top of a pre-existed intelligent system are summarized as follows:

1. We collect all the classifier's provided answers by Algorithm 4.

2. TThen, using the method of Dehkordi *et al.* [13], we construct a multi-agent epistemic logic Kripke model to find all possible outcomes of multi-classifiers, Algorithm 5. Here, we can investigate the verification for single-frame data.

3. By letting each data stream frame as a single-frame input, we can create a Kripke model for each frame. Then, a transition system can be created by placing the Kripke models in hierarchical order. Here, we develop a logic for the transition system by combining temporal logic (Gerth *et al.* [65]) with the epistemic logic model. It leads us to extract all possible knowledge sequences represented in data streams.

4. Finally, the situation of a formula (equivalently a property) is determined for data streams.

The structure of the chapter is as follows. In section 4.2, the formal verification of classifiers is defined. A PAL model is defined in section 4.3 to capture the information from single-framed data. Finally, in section 4.4, an extension of the model (called "Linear Temporal Public Announcement Logic", LTPAL) is defined by combining PAL with LTL. LTPAL is used to reason the captured knowledge from data streams by multi-agent systems. To create a more convenient approach for defining the verification's property, a procedure is suggested to formalize natural language in LTPAL -which can be served as an application.

4.2 Formal verification of Classifiers

Generally, formal verification is a mathematical process to ensure that a model fulfills some property function in an environment [35]. Thus, to define the verification of classifiers, we should ensure that the defined property function holds for the classifier. The concept of creating a set using an indicator function was defined by Yang *et al.* [66]. Here, we define the property set as follows:

Definition 4.2.1. *A property set ρ for elements of a set X defines with an indicator function $f_\rho : X \to \{true, false\}$, in which $\rho(X) = \{x \in X \mid f_\rho(x) = true\}$.*

A single-object detector classifier is a classifier detecting a single object from an input. In some critical cases, single-object classifiers provide a set of answers because the confidence level of the classification process does not satisfy the minimum requirements. For example, in the YOLO, the classifier threshold can be defined; and the outputs are a set of answers with scores over the threshold (Redmon et al. [16]). Another study created a set of inputs by discovering a neighborhood and manipulation set for the inputs. In this study, the outcome was all the outputs of the classifier for the neighborhood and manipulation set (Huang et al. [7]). In this study, the property set is denoted by $\rho(X)$; and the set of answers provided by classifier G for input x considering the property ρ is represented by the notation $[\rho(x)]_G$. By definition, a single-object classifier should provide a single output class. The concept of the verification of property in classifiers was defined by Dehkordi *et al.* [13]. We explain these within the literal context of this paper in the definitions as follows:

Definition 4.2.2. *A property $\rho(x)$ for a single-object detector classifier G is called* verified *exactly when* $\mid [\rho(x)]_G \mid = 1$.

Definition 4.2.3. *A property $\rho(x)$ for a set of single-object detector classifiers $A_G = \{G_1, \ldots, G_n\}$ is called* verified *exactly when* $\mid \bigcap_{G \in A_G} [\rho(x)]_G \mid = 1$.

In a multi-agent scenario, a property for single-object detector classifiers is verified when multiple classifiers agree about a single answer. For more detail about these prerequisites, see [13].

Note that we can assume a multi-object classifier as multiple single-object detector classifiers and the definitions are identical for each single-object detector classifier. Here, we use the term classifier instead of the single-object detector classifier for simplicity.

We define the verification for data streams by defining the properties in the form of LTPAL formulas in section 4.4.

4.3 Public Announcement Logic

In this section, we introduce a logical model for interpreting single-framed data. The model is based on PAL's extension to the epistemic logic [13]. Here, we call this extension as PAL. It is applied to extract knowledge from single-framed data (i.e., an image).

Let us first introduce the syntax and semantics of the logic.

Definition 4.3.1 (Language of PAL)**.** *Let $Ag = \{1, \ldots, n\}$ be a finite set of agents. The syntax of the language PAL is as follows, in BNF:*

$$\phi ::= p \mid \neg\phi \mid (\phi \wedge \phi) \mid K_i\phi \mid D_A\phi \mid [\phi]\phi.$$

where p is a propositional variable (atomic formula) is a pair of forms (x, c), in which x denotes the input data, c represents the target class, and $A \subseteq Ag$ is a subset of agents. We also notice that $K_i\phi$ read as "i-th classifier knows ϕ" ($i \in$ set of all classifiers). In other words, the i-th classifier is assured about the truth value of ϕ. Then, $D_A\phi$ read as "ϕ is a distributed knowledge in group A of classifiers". The formula $D_A\phi$ holds exactly when, aggregation of knowledge of

agents (equivalently $\cap_{i\in A}R_i$) in the group A, satisfies ϕ (if $A = \{i\}$ then K_i is equal to D_A). Finally, the formula $[\psi]\phi$ reads as "after a correct announcement of ψ, ϕ is".

Suppose that $\mathbb{G} = \{G_1, \ldots, G_n\}$ is a finite set of classifiers, X is the set of input points, and C is the set of all output classes. A PAL Kripke model is a tuple $M = (W, R_1, \ldots, R_n, V)$, where W is a set of worlds (hereabouts, the set W represents all possible output results of the input data),

$$W := \{c \in C \mid \exists x \in X,\ \exists G \in \mathbb{G},\ \text{such that},\ c \in [\rho(x)]_G\} \cup \{\bar{c}\}.$$

$R_i \subseteq W \times W$ is an equivalent relation between worlds for each classifier G_i in $\mathbb{G}$.

$$R_i(c) := \begin{cases} \{c\} & c \notin [\rho(x)]_{G_i} \\ \{c' \mid c' \in [\rho(x)]_{G_i}\} \cup \{\bar{c}\} & c \in [\rho(x)]_{G_i} \end{cases}$$

$R_i(\bar{c})$ are defined as follows:

$$R_i(\bar{c}) := \{c' \mid c' \in [\rho(x)]_{G_i}\} \cup \{\bar{c}\}.$$

We add $\bar{c}$ to the set of worlds to find out the "class appearing as an output of any agent". Since the classification of distinct input points is unrelated, we can consider a fixed input point x and create the model consequently. The final model will be the disjoint union of the models for each input. The intended meaning of the cR_ic' relation is that the i-th classifier cannot epistemically distinguish c and c'. Finally, $V : W \longrightarrow 2^{Prop}$ is the evaluation function specifying that the knowledge is represented in any world, where $Prop$ is the set of all atoms. $V(c)$ and $V(\bar{c})$ are defined as follows:

$$V(c) := \{(x, c) \mid \exists G \in \mathbb{G},\ \text{such that},\ c \in [\rho(x)]_G\},$$

$$V(\bar{c}) := \{(x, c) \mid \exists G \in \mathbb{G}, \text{ such that, } c \in [\rho(x)]_G\}.$$

We extend the evaluation function V to all formulas as follows:

- $\mathcal{M}, c \models p$ iff $p \in V(c)$,
- $\mathcal{M}, c \models \neg\phi$ iff $\mathcal{M}, c \not\models \varphi$,
- $\mathcal{M}, c \models \phi \wedge \psi$ iff $\mathcal{M}, c \models \phi$ and $\mathcal{M}, c \models \psi$,
- $\mathcal{M}, c \models K_i\phi$ iff $\forall v \in R_i(c), \mathcal{M}, v \models \phi$,
- $\mathcal{M}, c \models D_A\phi$ iff $\forall v \in R_A(c), \mathcal{M}, v \models \phi$, where $R_A := \bigcap_{i \in A} R_i$,
- $\mathcal{M}, c \models [\psi]\phi$ iff $\mathcal{M}, c \models \psi$ implies $\mathcal{M}^\psi, c \models \phi$.

Where $\mathcal{M}^\psi := (W^\psi, R_1^\psi, \ldots, R_n^\psi, V^\psi)$ with

$W^\psi := \{c \in W; \mathcal{M}, c \models \psi\}$,

$R_j^\psi := R_j \cap (W^\psi \times W^\psi)$ for all $j \in \{1, \ldots, n\}$, and

$V^\psi(c) := V(c)$ for all $w \in W^\psi$.

For any given model $\mathcal{M}$ and any given formula ψ, Algorithm 6 simply calculates the model $\mathcal{M}^\psi$.

Now we can show that a point is verified for classifiers exactly when its interpretation is valid in $\mathcal{M}$.

Theorem 4.3.2. *[13] Suppose that $\mathbb{G} = \{G\}$ is a classifier, x is an input and c is an output class of G. Then:*

$$[\rho(x)]_G = \{c\} \textit{ exactly when } \mathcal{M}, c \models Kp_c, \textit{ where } (x, c) \textit{ denoted by } p_c.$$

Proof. Only-if part. Suppose $\mathcal{M}, c \models Kp_c$, where p_c is interpreted by (x, c). As there is only one classifier and c is an output class, we have $c \in [\eta(x) \cup \mu(x)]_G$. Suppose $c' \in [\eta(x) \cup \mu(x)]_G$. So $c' \in R(c)$ and we have $\mathcal{M}, c' \models p_c$ by hypothesis, which means that $(x, c') \in V(c)$, thus $c = c'$.
If-part. Suppose $[\eta(x) \cup \mu(x)]_G = \{c\}$. Because c is an output of the classifier, $c \in W$, and as it is the only output of classifier G on input x, we have $R(c) = \{c\}$. Thus, it suffices to show that $\mathcal{M}, c \models p_c$. As $\{(x, c)\} \in V(c)$, the statement is proved. □

Theorem 4.3.3. *[13] Suppose that $\mathbb{G} = \{G_1, \ldots, G_n\}$, is a multi-classifier system such that $\bigcap_{G\in\mathbb{G}}[\rho(x)]_G \neq \emptyset$, x is an input and c is an output class for classifiers in $\mathbb{G}$, then:*

$$\bigcap_{G\in\mathbb{G}}[\rho(x)]_G = \{c\} \textit{ exactly when } \mathcal{M}, \bar{c} \vDash D_{\mathbb{G}}p_c, \textit{ where } (x, c) \textit{ denoted by } p_c.$$

Proof. Only-if part. Suppose that $\mathcal{M}, \bar{c} \vDash D_{\mathbb{G}}p_c$, then for all c' in $R_{D_{\mathbb{G}}}(\bar{c})$ we have $\mathcal{M}, c' \vDash p_c$. On the other hand, by the assumption there is an element d in $\bigcap_{G\in\mathbb{G}}[\eta(x)\cup\mu(x)]_G$. It is enough to show that $d = c$. We have $d \in \bigcap_{G\in\mathbb{G}} R_{G_i}(\bar{c})$. Since $\mathcal{M}, d \vDash p_c$, then $V(d) = \{(x, c)\}$ which implies that $d = c$.
If-part. Suppose $\bigcap_{G\in\mathbb{G}}[\eta(x) \cup \mu(x)]_G = \{c\}$. Because c is an output of the classifiers, $c \in W$, and $c \in [\eta(x) \cup \mu(x)]_G$, for all $G \in \mathbb{G}$. On the other hand as it is the only common output of all the classifiers, for any other output class, there exist a classifier G, that does not have it as output. So $R_{D_{\mathbb{G}}}(c) = \{c, \bar{c}\}$. Therefore it suffices to show that $\mathcal{M}, \bar{c} \models p_c$. As $\{(x, c)\} \in V(c) \cap V(\bar{c})$, the statement is proved. □

In the following theorem, Algorithm 7 was developed to investigate the correctness of PAL formulas. Let $DePth_\varphi$ and $DeNop_\varphi$ be the number of non-path $(\neg, \wedge)$ and path operators $(K_i, D_A, [\ ])$ in formula φ; the time complexity of Algorithm 7 would be $O(\mid W \mid^{DePth_\varphi} \times DeNop_\varphi)$. Consequently, the time complexity of Algorithm 6 is $O(\mid W \mid \times \mid W \mid^{DePth_\varphi} \times DeNop_\varphi)$.

Algorithm 6 The Single-Framed Knowledge Extraction (SFKE) function will reduce the model using the knowledge produced by classifiers.

Let $\mathcal{M}$ as a Kripke model and φ as a PAL formula

1: **function** SFKE($\mathcal{M}, \varphi$)
2: ▷ V could be driven from the model $\mathcal{M}$
3: **for all** possible world $c \in \mathcal{M}$ **do**
4: **if** not PALS($\varphi, \mathcal{M}, c$) **then**
5: Remove c from the model $\mathcal{M}$.
6: Remove all relations cR_jc' and $c'R_jc$ from the model $\mathcal{M}$.
7: **return** $\mathcal{M}$

Theorem 4.3.4. *Let $\mathcal{M} = (W, R_1, \cdots, R_n, V)$ as a Kripke model, $c \in W$ and ϕ a PAL formula. Then, $\mathcal{M}, c \vDash \phi$ if and only if PALS(ϕ,$\mathcal{M}$, c).*

Proof. We only prove the if direction; the other direction is proved similarly. The proof is obtained by induction on the complexity of ϕ. Suppose that $\mathcal{M}, c \vDash \phi$, and $\phi \in$PAL.

- If ϕ is an atomic formula, then the *if condition* at line 3 would be satisfied, so line 4 would be executed. Here, we will check whether ϕ is in $V(c)$ or not. By definition we have $\mathcal{M}, c \models p$ if and only if $p \in V(c)$.

- If $\phi \equiv \neg\phi_1$, then, in this case, the *if condition* at line 7 would be satisfied, so line 8 would be executed. At this line we return the negation of PALS(ϕ_1,$\mathcal{M}$, c). Then, by induction PALS(ϕ_1,$\mathcal{M}$, c) if $\mathcal{M}, c \vDash \phi_1$.

- If $\phi \equiv \phi_1 \wedge \phi_2$, then, the *if condition* at line 10 would be satisfied, and line 11 would be executed. At this line we return the conjunction of PALS(ϕ_1,$\mathcal{M}$, c) and PALS(ϕ_2,$\mathcal{M}$, c). By induction of PALS(ϕ_1,$\mathcal{M}$, c), if $\mathcal{M}, c \vDash \phi_1$ and PALS(ϕ_2,$\mathcal{M}$, c), then $\mathcal{M}, c \vDash \phi_2$.

- If $\phi \equiv K_i\phi_1$, then, the *if condition* at line 13 would be satisfied, and line 14 would be executed. In this line, for every $c' \in R_i(c)$, PALS(ϕ_1,$\mathcal{M}$, c') would be calculated; *true* would be returned

if and only if they all are satisfied. By induction, we have PALS(ϕ_1,$\mathcal{M}$, c') if $\mathcal{M}, c' \vDash \phi_1$. By definition, we have $\forall c' \in R_i(c), \mathcal{M}, c' \models \phi_1$ if $\mathcal{M}, c \models K_i\phi_1$. Therefore, for all $c' \in R_i(c)$, PALS($\phi_1, \mathcal{M}, c'$) if $\mathcal{M}, c \models K_i\phi_1$. Consequently, PALS($\phi, \mathcal{M}, c$) if $\mathcal{M}, c \models \phi$.

- If $\phi \equiv D_A\phi_1$, then, the *if condition* at line 16 would be satisfied, and lines 17 and 18 would be executed. In line 17, first, R_A would be calculated the same as R_{D_A} in the definition. Then, for every $c' \in R_A(c)$, PALS(ϕ_1,$\mathcal{M}$, c') would be calculated; *true* would be returned if and only if they all are satisfied. By induction, we have PALS(ϕ_1,$\mathcal{M}$, c') if $\mathcal{M}, c' \vDash \phi_1$. By definition, we have $\forall c' \in R_{D_A}(c), \mathcal{M}, c' \models \phi_1$ if $\mathcal{M}, c \models D_A\phi_1$. Therefore, for all $c' \in R_i(c)$, PALS($\phi_1, \mathcal{M}, c'$) if $\mathcal{M}, c \models D_A\phi_1$. Consequently, PALS($\phi, \mathcal{M}, c$) if $\mathcal{M}, c \models \phi$.

- If $\phi \equiv [\phi_2]\phi_1$, then, the *if condition* at line 20 would be satisfied, and lines 21 to 25 would be executed. In line 21, first, we check if PALS($\phi_2, \mathcal{M}, c$) is false. Based on the definition, $\mathcal{M}, c \models \phi_2$ implies that $\mathcal{M}^{\phi_2}, c \models \phi_1$ would be *true*. Because, by induction we had PALS(ϕ_2,$\mathcal{M}$, c) if $\mathcal{M}, c \vDash \phi_2$. Otherwise, we could calculate $\mathcal{M}^{\phi_2}$ with the function SFKE($\mathcal{M}, \phi_2$), Algorithm 6. SFKE returns a modification of the Kripke model $\mathcal{M}$, in which all worlds c, $\mathcal{M}, c \nvDash \phi_2$ are removed. Therefore, SFKE($\mathcal{M}, \phi_2$)$\equiv$ $\mathcal{M}^{\phi_2}$. By induction, we have PALS(ϕ_2,$\mathcal{M}$, c) if $\mathcal{M}, c \vDash \phi_2$ and PALS(ϕ_1,$\mathcal{M}^{\phi_2}$, c) if $\mathcal{M}^{\phi_2}, c \vDash \phi_1$. Based on the definition, we have $\mathcal{M}, c \models \phi_2$ implies $\mathcal{M}^{\phi_2}, c \models \phi_1$ if $\mathcal{M}, c \models [\phi_2]\phi_1$. Consequently, PALS($\phi, \mathcal{M}, c$) if $\mathcal{M}, c \models \phi$.

□

Algorithm 7 The PAL Satisfaction function (PALS) will investigate the satisfaction of PAL formulas.

Let ϕ be the PAL formula, $\mathcal{M}$ the Kripke model, and c a world

function PALS($\phi, \mathcal{M}, c$)
▷ R_i and V could be driven from the model $\mathcal{M}$
    if ϕ is an atomic formula **then**
        return $\phi \in V(c)$
▷ This will be true if $\phi \in V(c)$ and false otherwise
    if ϕ is in form of $\neg\psi$ **then**
        return $\neg$ PALS($\psi, \mathcal{M}, c$)
    if ϕ is in form of $\psi_1 \wedge \psi_2$ **then**
        return PALS($\psi_1, \mathcal{M}, c$)$\wedge$PALS($\psi_2, \mathcal{M}, c$)
    if ϕ is in form of $K_i\psi$ **then**
        return $\wedge_{c'}$PALS($\psi, \mathcal{M}, c'$);$\forall c' \in R_i(c)$
    if ϕ is in form of $D_A\psi$ **then**
        $R_A := \bigcap_{i \in A} R_i$
        return $\wedge_{c'}$PALS($\psi, \mathcal{M}, c'$);$\forall c' \in R_A(c)$
    if ϕ is in form of $[\psi_2]\psi_1$ **then**
        if $\neg$ PALS($\psi_2, \mathcal{M}, c$) **then**
            return true
        else
            return PALS(ψ_1, SFKE($\mathcal{M}, \psi_2$), c)
    return false

4.4 Linear Temporal Public Announcement Logic

In this section, an LTL extension of PAL is introduced to extract the knowledge of data stream inputs. It employs knowledge collected by classifiers in single-frame inputs and puts them in a hierarchical order.

Definition 4.4.1 (Language of LTPAL). *We introduce an LTL extension of PAL by the following grammar in BNF:*

$$\phi ::= p \mid \neg\phi \mid (\phi \wedge \phi) \mid K_j\phi \mid D_A\phi \mid [\phi]\phi$$

$$\Phi ::= \phi \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid X\Phi \mid [\Phi U \Phi]$$

Here, the temporal operators are $X\Phi$ (in the ne$X$t data frame Φ must be true) and $[\Phi U \Psi]$ (Ψ must remain true Until Φ becomes true).

To define the Kripke semantic of this logic, assume that $M_0 = (\{c_{0_0}\}, R_{0_0}, \ldots, R_{0_k}, V_0), M_1 = (W_1, R_{1_0}, \ldots, R_{1_k}, V_1), \ldots, M_{n-1} = (W_{n-1}, R_{(n-1)_0}, \ldots, R_{(n-1)_k}, V_{n-1})$, *and* $M_n = (\{c_{n_0}\}, R_{n_0}, \ldots, R_{n_k}, V_n)$ are PAL models, where W_i's are mutually disjoint sets, and $V_0(c_{0_0}) = V_n(c_{n_0}) = \emptyset$.

We build a new model $\mathcal{TS} = (S, R, s^0, s^{-1}, \rightarrow, L)$, known as a transition system, in which $S = \bigcup_{i=0}^{n} W_i$ is the set of states, $R = \{R_{i_j} \mid 0 \leq i \leq n,\ 0 \leq j \leq k\}$, $s^0 = c_{0_0}$ and $s^{-1} = c_{n_0}$ are initial and final states, $\rightarrow^k = W_k \times W_{k+1},\ 0 \leq i < n$ are transition relations, $\rightarrow = \bigcup_{0 \leq k < n} \rightarrow^k$, $(c_{ki}, c_{(k+1)j}) \in \rightarrow^k$ is denoted by $\rightarrow^k_{ij}$, and the labeling function $L : S \longrightarrow 2^{Prop}$ which assigning propositional letters to states is defined by $L(c_{i_j}) = V_i(c_{i_j})$, for each $c_{i_j} \in W_i$. To extend the labeling function to all LTPAL formulas, we first fix the following notations.

An executive path $\pi^{i\ldots j}_{\mathcal{TS}}$ in the model $\mathcal{TS}$ is a sequence of worlds $c_i c_{i+1} \ldots c_j$ where $c_i \rightarrow c_{i+1} \rightarrow \cdots \rightarrow c_j$ for $c_k \in W_k,\ i \leq k \leq j$. An executive path $\pi^{i\ldots j}_{\mathcal{TS}}$ is called total and denoted by $\pi_{\mathcal{TS}}$, if $c_i = s^0$ and $c_j = s^{-1}$. The set of all total executive paths will be denoted by $\Pi_{\mathcal{TS}}$. Next, for $\phi \in$ PAL and $\Phi, \Phi_1, \Phi_2 \in$ LTPAL, we define:

- $\mathcal{TS}, \pi^{i\ldots j}_{\mathcal{TS}} \vDash \phi$ iff $M_i, c_i \vDash \phi$,
- $\mathcal{TS}, \pi^{i\ldots j}_{\mathcal{TS}} \vDash X\Phi$ iff $\mathcal{TS}, \pi^{i+1\ldots j}_{\mathcal{TS}} \vDash \Phi$,
- $\mathcal{TS}, \pi^{i\ldots j}_{\mathcal{TS}} \vDash \Phi_1 U \Phi_2$ iff there exists m, $i \leq m \leq j$, $\mathcal{TS}, \pi^{m\ldots j}_{\mathcal{TS}} \vDash \Phi_2$, and for all k, $i \leq k < m$, we have $\mathcal{TS}, \pi^{i\ldots k}_{\mathcal{TS}} \vDash \Phi_1$.

In the first clause above, we consider the transition system TS over PAL formulas simply as a Kripke model. Moreover, Algorithm 8 is developed to create a transition system from PAL models. We

remark that in Algorithm 8, to have the same beginning and end in calculating the paths, two extra models are added, line 3-12 and line 23-29. Note that these two models do not affect the satisfaction of the input formula.

Algorithm 8 The Time-Series Transition System (TSTS) function will create the transition system by the given input information.

Let $\mathcal{C}$ as the set of classifiers, X the time-series input data of size k ($X = [x_1, \ldots, x_k]$), and ρ the predefined property set. We also use the MASKS function developed in [13].

1: **function** TSTS($\mathcal{C}, X, \rho$)
2: $k := size(X)$
3: $S := \{c_0\}$
4: $W := \{c_0\}$
5: $R_1, \ldots, R_n := \{(c_0, c_0)\}$
6: $s^0 := c_0$
7: $s^{-1} := c_0$
8: $\rightarrow := \emptyset$
9: $V(c_0) := \emptyset$
10: $L(c_0) := \emptyset$
11: $\mathcal{M} = (W, R_1, \ldots, R_n, V)$
12: $\mathcal{TS} := (S, R_1, \ldots, R_n, s^0, s^{-1}, \rightarrow, L)$
13: **for all** x_0 in X **do**
14: $_, \mathcal{M}' :=$MASKS$(\mathcal{C}, x_0, \rho)$
15: ▷ Kripke model $\mathcal{M}' = (W', R'_1, \ldots, R'_n, V')$
16: $S := S \cup W'$
17: $\forall_i R_i := R_i \cup R'_i$
18: **for all** $c \in W$ and $c' \in W'$ **do**
19: $\rightarrow :=\rightarrow \cup \{(x, c')\}$
20: $\mathcal{M} := \mathcal{M}'$
21: $W := \{c_{k+1}\}$
22: $S := S \cup W$
23: $\forall_i R_i := R_i \cup \{(c_{k+1}, c_{k+1})\}$
24: $s^{-1} := c_{k+1}$
25: **for all** $c \in W$ and $c' \in W'$ **do**
26: $\rightarrow :=\rightarrow \cup \{(x, c')\}$
27: **return** $\mathcal{TS}$

Temporal formulas can be investigated by the function TEMS developed in Algorithm 9.

Here, the first and last Kripke models should be excluded. In other words, path $\pi_{TS}^{i...j}$ in function $TEMS(\Phi, \mathcal{TS}, \pi_{TS}^{i...j})$ should not start or end in the first and last Kripke models of the transition system. For example, for a transition system $\mathcal{TS}$, the investigation of formula Φ over a total path $\pi_{TS}^{0...n}$ would be TEMS($\Phi, \mathcal{TS}, \pi_{TS}^{1...n-1}$). Let n as the number of Kripke models in the transition system, DeN_Φ as the number of operators $\neg$, $\wedge$, and X in the formula Φ; and DeU_Φ be the number of operators U in formula Φ, the time complexity of Algorithm 9 would be $O(n^{DeU_\varphi} \times DeN_\varphi)$.

Other temporal operators could be driven from the two operators of *next* ($X\Phi$), and *until* ($U\Phi$) in the following way:

- $F\Phi \equiv (\top U \Phi)$
- $(\Phi R \Psi) \equiv \neg(\neg\Phi U \neg\Psi)$
- $(\Phi W \Psi) \equiv (\Psi R (\Phi \vee \Psi))$
- $G\Phi \equiv (\bot R \Phi)$

The intended meaning of *future* ($F\Phi$) is "eventually Φ becomes true", *global* ($G\Phi$) is "Φ must remain true forever", *release* ($\Phi R \Psi$) is "Ψ remains true until and including when Φ becomes true, if Φ never becomes true, Ψ always remains true", and *weak until* ($\Phi W \Psi$) is "Φ has to remain true at least until Ψ; if Ψ never holds, Φ must always remain true" [67], [68].

Theorem 4.4.2. *Let $\mathcal{TS} = (S, R_1, \ldots, R_n, s^0, s^{-1}, \rightarrow, L)$ be a Transition System and Φ an LTPAL formula. Then for $\mathcal{TS}, \pi_{TS}^{i...j} \in \Pi$, we have $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi$ iff TEMS(Φ,$\mathcal{TS}$, $\pi_{TS}^{i...j}$).*

Proof. We only prove the if direction; the other direction is proved similarly. The proof is by induction on the complexity of Φ.

For any path formula in which the path length is lower than zero, lines 3 and 4 would be executed; and *false* will be returned. For any PAL formula, lines 6 to 8 would be executed. Similar to the

Algorithm 9 The TEMporal Satisfaction function (TEMS) will investigate the satisfaction of LTPAL formulas.

Let Φ as the LTL formula, $\mathcal{TS}$ the transition system, and c a world

1: **function** TEMS($\Phi, \mathcal{TS}, \pi_{TS}^{i...j}$)
2: $\triangleright \rightarrow$ could be driven from the model $\mathcal{TS}$
3: **if** $i > j$ **then**
4: **return** false
5: **if** Φ is a PAL formula **then**
6: $\mathcal{M} :=$ the i-th Kripke model of $\mathcal{TS}$
7: **return** PALS($\Phi, \mathcal{M}, c_i$)
8: $\triangleright$ let c_i as first world in path $\pi_{TS}^{i...j}$
9: **if** Φ is in form of $\neg\Psi$ **then**
10: **return** $\neg$ TEMS($\Psi, \mathcal{TS}, \pi_{TS}^{i...j}$)
11: **if** Φ is in form of $\Psi_1 \wedge \Psi_2$ **then**
12: **return** TEMS($\Psi_1, \mathcal{TS}, \pi_{TS}^{i...j}$)$\wedge$TEMS($\Psi_2, \mathcal{TS}, \pi_{TS}^{i...j}$)
13: **if** Φ is in form of $X\Psi$ **then**
14: **return** TEMS($\Psi, \mathcal{TS}, \pi_{TS}^{i+1...j}$)
15: **if** Φ is in form of $\Psi_1 U \Psi_2$ **then**
16: **if** TEMS($\Psi_2, \mathcal{TS}, \pi_{TS}^{i...j}$) **then**
17: **return** true
18: **while** TEMS($\Psi_1, \mathcal{TS}, \pi_{TS}^{i...j}$) **do**
19: $i := i + 1$
20: **if** TEMS($\Psi_2, \mathcal{TS}, \pi_{TS}^{i...j}$) **then**
21: **return** true
22: **return** false

semantics of LTPAL, we will check whether Φ is in $PALS(\Phi, \mathcal{M}, c)$ or not. Here, $\mathcal{M}$ is the i-th Kripke model of the Transition system $\mathcal{TS}$.

Suppose that $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi$, and $\Phi \in$LTPAL is not a PAL formula. Thus, Φ would be in the form of $\neg\Phi_1, \Phi_1 \wedge \Phi_2, X\phi_1$, or $\Phi_1 U \Phi_2$.

- If $\Phi \equiv \neg\Phi_1$, then the *if condition* at line 11 would be satisfied, and line 12 would be executed. At this line, we return the negation of TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$). Therefore, by induction, TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi_1$. Consequently, TEMS($\Phi$,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if

$\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi$.

- If $\Phi \equiv \Phi_1 \wedge \Phi_2$ then, the *if condition* at line 14 would be satisfied, and line 15 would be executed. At this line we return the conjunction of TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) and TEMS(Φ_2,$\mathcal{TS}$, $\pi_{TS}^{i...j}$). Therefore, by induction, TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi_1$ and TEMS(Φ_2,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi_2$. Consequently, TEMS($\Phi$,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi$.

- If $\Phi \equiv X\Phi_1$, then the *if condition* at line 17 would be satisfied, and line 18 would be executed. In this line, TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i+1...j}$) would be calculated; *true* would be returned if and only if for the next state of the path $\pi_{TS}^{i...j}$, which is $\pi_{TS}^{i+1...j}$, Φ_1 is true. By induction, we have TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i+1...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i+1...j} \vDash \Phi_1$. By definition, we have $\mathcal{TS}, \pi_{\mathcal{TS}}^{i+1...j} \vDash \Phi$ if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash X\Phi$. Consequently, TEMS($\Phi$,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi$.

- If $\Phi \equiv \Phi_1 U \Phi_2$, then the *if condition* at line 20 would be satisfied, and lines 21 to 29 would be executed. In line 21, first, we check if TEMS(Φ_2,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) is true. By definition, we have: $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi_1 U \Phi_2$ if there exists m, $i \leq m \leq j$, $\mathcal{TS}, \pi_{\mathcal{TS}}^{m...j} \vDash \Phi_2$, and for all k, $i \leq k < m$, we have $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...k} \vDash \Phi_1$. Hence, if $m = i$, TEMS(Φ,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) would be true. Otherwise, by definition TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) must be true over the path while satisfying TEMS(Φ_2,$\mathcal{TS}$, $\pi_{TS}^{i...j}$). If TEMS(Φ_2,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) is never satisfied while TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) is true, and false should be returned (line 31). By induction we have TEMS(Φ_1,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...k} \vDash \Phi_1$ and TEMS(Φ_2,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...k} \vDash \Phi_2$. Consequently, TEMS($\Phi$,$\mathcal{TS}$, $\pi_{TS}^{i...j}$) if $\mathcal{TS}, \pi_{\mathcal{TS}}^{i...j} \vDash \Phi$.

□

We conclude the paper by giving a formal definition of verification, possibility, and missing information in a transition system. Thus, the following notations are introduced first.

Notation 1. *Let* $\psi, \phi \in$ *PAL,* $\Phi, \Phi_1, \Phi_2 \in$ *LTPAL,* $\Phi'_1, \Phi'_2 \in$ *LTPAL/PAL,* $O_1 = \{\wedge, \vee\}, O_2 = \{U, R\}, O = O_1 \cup O_2$*,* $f(\wedge) = \vee, f(\vee) = \wedge$*,* $f(U) = R, f(R) = U$*. For* $\Omega \in \{D_A, \neg D_A \neg, K_i, \neg K_i \neg, [\psi]D_A, [\psi]\neg D_A \neg, [\psi]K_i, [\psi]\neg K_i \neg\}$ *we define* $\Omega\Phi$ *as follows:*

$$\Omega\Phi = \begin{cases} \Omega\Phi_1 & \textit{iff } \Phi = \Phi_1, \\ X\Omega\Phi_1 & \textit{iff } \Phi = X\Phi_1, \\ X\Omega\neg\Phi_1 & \textit{iff } \Phi = \neg X\Phi_1, \\ \Omega\Phi'_1 O \Omega\Phi'_2 & \textit{iff } \Phi = \Phi'_1 O \Phi'_2, \\ \Omega(\neg\Phi'_1 f(O_1) \neg\Phi'_2) & \textit{iff } \Phi = \neg(\Phi'_1 O_1 \Phi'_2), \\ \Omega(\neg\Phi_1 f(O_2) \neg\Phi_2) & \textit{iff } \Phi = \neg(\Phi_1 O_2 \Phi_2). \end{cases} \tag{4.1}$$

Now, for any transition system $\mathcal{TS}$ and $\Phi \in$ LTPAL, we define verification of Φ in $\mathcal{TS}$ formula as follows:

1. Φ is *verified* in $\mathcal{TS}$ for the group A of classifiers exactly when, we have $\mathcal{TS}, \pi_{\overline{\mathcal{TS}}} \vDash D_A\Phi$.
2. Φ is *possible* in $\mathcal{TS}$ for the group A of classifiers exactly when, we have $\mathcal{TS}, \pi_{\overline{\mathcal{TS}}} \vDash \neg D_A \neg\Phi$.
3. A PAL formula ψ is called *verified-missing* information in $\mathcal{TS}$ for a formula Φ in group A of classifiers exactly when, we have $\mathcal{TS}, \pi_{\overline{\mathcal{TS}}} \nvDash D_A\Phi$ and $TS, \pi_{\overline{\mathcal{TS}}} \vDash [\psi](D_A\Phi)$.
4. A PAL formula ψ is called *possible-missing* information in $\mathcal{TS}$ for a formula Φ in group A of classifiers exactly when, we have $\mathcal{TS}, \pi_{\overline{\mathcal{TS}}} \nvDash \neg D_A\neg\Phi$ and $TS, \pi_{\overline{\mathcal{TS}}} \vDash [\psi](\neg D_A\neg\Phi)$.

Example. Example. Assume a one-bed medical operating room, which is monitored by a camera. An operation is carried out on three animals which may be dogs, wolves, or foxes, denoted by d, w, and f. During these operations, someone revealed that the sterilization system

did not work; we know that dogs have the flu. It could spread from dogs to wolves and from wolves to foxes if they go to the operating room after each other. No one knows the order of the operation. Thus, they fed three respective camera images $X = \{x_1, x_2, x_3\}$ of operations into two classifiers $A = \{1, 2\}$ to find out whether the fox contracted the flu or not. The first classifier doubts fox and dog for the first image, wolf and dog for the second, and fox and wolf for the third. The second classifier is uncertain about the wolf and dog for the first image, about the fox and wolf for the second, and about the wolf and dog for the third. We will apply our model to this scenario. Let (x_i, d), (x_i, w), and (x_i, f) as atomic formulas (d_i, w_i and f_i in brief) demonstrating the existences of dog, wolf, and fox for the i-th image, respectively. By Algorithm 3, Kripke models $\mathcal{M}_i = (W_i, R_{i,1}, R_{i,2}, V_i)$, for $i \in \{1, 2, 3\}$, are shown in Figure 4.1, where $V_i(c) = \{c\}$, for $c \in \{d_i, w_i, f_i\}$ and $V_i(\bar{c}_i) = \{d_i, w_i, f_i\}$.

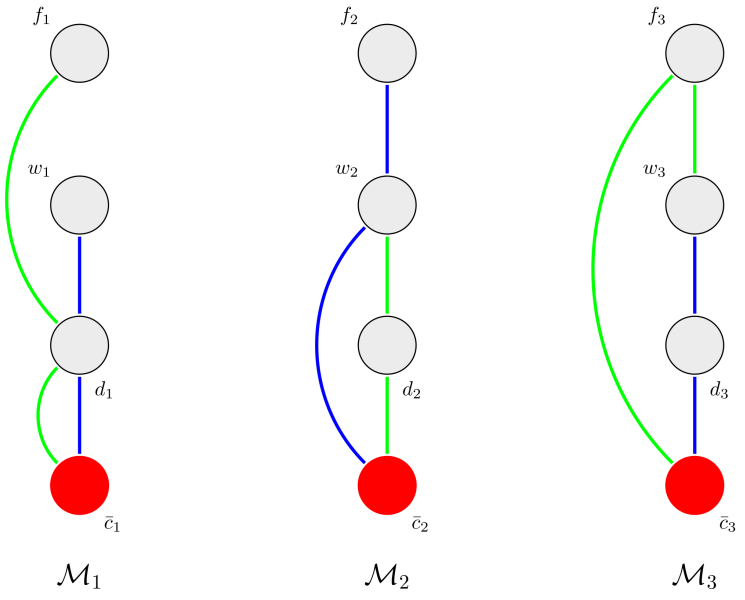


Figure 4.1: Kripke models $\mathcal{M}_i = (W_i, R_{i,1}, R_{i,2}, V_i)$; $R_{i,1}$ and $R_{i,2}$ are green and blue lines, respectively.

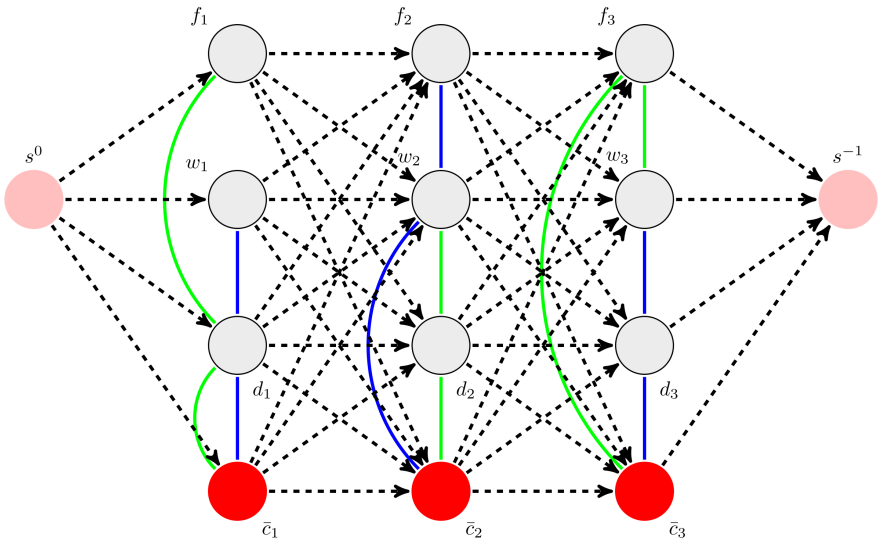


Figure 4.2: The transition system $\mathcal{TS} = (S, R, s^0, s^{-1}, \rightarrow, L)$.

The resulting transition system $\mathcal{TS} = (S, R, s^0, s^{-1}, \rightarrow, L)$ of $\mathcal{M}_i$'s, by Algorithm 8, is depicted in Figure 4.2, where $L = V_1 \cup V_2 \cup V_3$ and $L(s^0) = L(s^{-1}) = \emptyset$.

Next, we would like to check whether any fox contracted the flu or not, which can be expressed in LTPAL by $\Phi = d_1 \wedge Xw_2 \wedge XXf_3$. By Algorithm 9, $TS, \pi_{\overline{\mathcal{TS}}} \nvDash D_A\Phi$, which means that the system does not ensure that a fox contracts the flu (it is not verified). Also, $TS, \pi_{\overline{\mathcal{TS}}} \nvDash \neg D_A\neg\Phi$ means that a fox did not contract the flu (it is impossible).

Next, we check the impact of the collaboration of agents and announcements. For the former, assume that, in the beginning, wolves also had the flu which can be expressed by $\Phi' = F(w_1 \wedge Xf_2) \vee F(w_2 \wedge Xf_3)$. Then, $TS, \pi_{\overline{\mathcal{TS}}} \nvDash \neg D_A\neg\Phi'$, which means that from group A's perspective, contracting the flu is impossible for a fox. Nevertheless, no single agent knows that the flu is impossible for a fox. Since, $\mathcal{TS}, \pi_{\overline{\mathcal{TS}}} \vDash \neg D_{A_i}\neg\Phi'$, for $A_i = \{i\}$. For the latter, assume that someone announced the second operation was done on a wolf. In this case, we have

$\mathcal{TS}, \pi_{\overline{\mathcal{TS}}} \not\models [w_2]\neg D_{A_2}\neg\Phi'$. It means that the second agent could ensure that the flu is impossible for a fox when w_2 is announced.

4.4.1 Verification of LTPAL in application

A straightforward interpretation of human language in modal logic (especially in LTL) is one of the most critical strengths of such logic (see [69], [70], and [71]). This kind of interpretation could help robots react to human orders [72]. This ability of the LTL made its verification the most crucial.

In order to convert the question into a formula, we first need to extract all possible outputs to LTL formulas. Then, we will investigate the satisfaction of each answer. The developed model would lead us to a PAL modification on such LTL formulas. The creative way of LTPAL made it more convenient to translate. To explain that, let $\Phi(p_1, \ldots, p_\sigma)$ be an LTL-translated formula, and $\phi_i,\ 1 \leq i \leq \sigma$ is a PAL formula. Then, $\Phi(\phi_1, \ldots, \phi_\sigma)$ is obtained from ϕ by substituting each p_i with ϕ_i for all $1 \leq i \leq \sigma$. For instance, for $\Phi(p_1, p_2) = G(p_1 U X p_2)$, $\phi_1 = \neg D_A \neg p_1$, and $\phi_2 = K_i p_2$, we have $\Phi(\phi_1, \phi_2) = G((\neg D_A \neg p_1) U (X(K_i p_2)))$. The formula $\Phi(p_1, p_2)$ means that "p_1 should always be a possible answer until in the world, which is right after, p_2 is a robust knowledge for the i-th agent". For the transition system $\mathcal{TS}$ and the investigated execution path $\pi_{\mathcal{TS}}$, the satisfaction of a formula could be determined. By checking the satisfaction of the formula over all paths, one could interpret whether the formula is verified (satisfies over all paths) or possible (satisfies in a path). It guides the questioner to find out, "how reliable is this answer?". It would lead us to design a more proper system for critical applications. Moreover, by capturing missing knowledge, it could be caught which knowledge is mandatory for the system that is not discovered by classifiers and which part of the system could be manipulated to obtain the missing information. We will show that for every LTPAL formula, there exists a negation normal form (NNF); it would permit us to investigate answers or their negation.

Lemma 4.4.3. *For each formula $\phi \in \Phi/PAL$, there exists an equivalent formula $\phi' \in \Phi$ (called $NNF(\phi)$) in which $\neg$ is not a main operator.*

By induction, the formula ϕ is in the form of:

- $\phi \equiv \neg(\neg\varphi)$, *by the definition* $\neg(\neg\varphi) \equiv \varphi$, *so, let* $\phi' \equiv \varphi$.
- $\phi \equiv \neg(X\varphi)$, *by the definition* $\neg(X\varphi) \equiv X(\neg\varphi)$, *so, let* $\phi' \equiv X(\neg\varphi)$.
- $\phi \equiv \neg(\varphi \wedge \psi)$, *by the definition* $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$, *so, let* $\phi' \equiv \neg\varphi \vee \neg\psi$.
- $\phi \equiv \neg(\varphi \vee \psi)$, *by the definition* $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$, *so, let* $\phi' \equiv \neg\varphi \wedge \neg\psi$.
- $\phi \equiv \neg(\varphi U \psi)$, *by the definition* $\neg(\varphi U \psi) \equiv \neg\varphi R \neg\psi$, *so, let* $\phi' \equiv \neg\varphi R \neg\psi$.
- $\phi \equiv \neg(\varphi R \psi)$, *by the definition* $\neg(\varphi R \psi) \equiv \neg\varphi U \neg\psi$, *so, let* $\phi' \equiv \neg\varphi U \neg\psi$.

Example. (Representation of a Video) Assume a medical *clean-room* in an Operating theater monitored by a camera. The recorded video will be fed into a set of classifiers A, and the transition system $\mathcal{TS}$ will be created by algorithm 8. The room will be cleaned with an air conditioner and an ultra-violet (UV) LED. As it is known that the UV LED would damage people, it must be turned off while a person is in the room. On the contrary, the air conditioner should work during a person's appearance in that room. It also should be stopped in an empty room to save electricity. Therefore, the first LTPAL formula (here referred to as protocol) is "shut down UV LED while any person monitored", and the second one is "shout down the air conditioner when no one is in the room". The classifiers should answer the vital question of "How robust are the protocols?".

Let p be "at least one person exists", q be "the UV LED is on", and r be "air conditioner is on". We have the following translations:

- $G(p \Rightarrow X\neg q)$: which is the translation of "by observation of human, the UV LED should be shut down".

- $TS, \pi_{\overline{\mathcal{TS}}} \vDash D_A G(p \Rightarrow X(\neg q))$: it means that the property would be verified within group A.
- $TS, \pi_{\overline{\mathcal{TS}}} \vDash \neg D_A \neg G(p \Rightarrow X(\neg q))$: it means that there is a scenario in which the formula holds (this property is possible within group A).

Therefore if $TS, \pi_{\overline{\mathcal{TS}}} \vDash D_A G(p \Rightarrow X(\neg q))$ holds, we will ensure that the system enforces UV LED to be turned off when the human exists.

4.5 Finding the most probable path in semi-continuous data streams

Traversing all paths in the developed transition system could be very time-consuming, according to the size of PAL models. It would be useful for offline classification, but computations should be reduced in real-time applications. It could be reduced considering this characteristic for the semi-continuous data streams, in which each frame strongly correlated with the next and previous frames. For instance, video stream image frames are usually strongly correlated with the next and previous image frames [73]. In other words, it could be assumed that the video stream is semi-continuous data, and the overall changes between frames would be minimal. Using this property of data streams together with recent tag similarity algorithms, like ones developed in natural language processing (NLP), the most probable path would be determined in the following manner:

- Determine the LTPAL model as we discussed in section 4.4.
- Calculate the probability of each transition relation $\rightarrow_{ij}^{k}$. This value is the probability of converting tags in world c_{ki} to the tags in world $c_{(k+1)j}$.
- Traverse the transition system to find the most probable path.

- Check the satisfaction of desired formulas in the most probable path.

As mentioned before, the set of paths $\Pi_{\mathcal{TS}}$ in the transition system $\mathcal{TS}$ are created by worlds of $n+1$ models $M_0, \ldots, M_n$, in which transition relations are between two successive worlds of models M_i and $M_{(i+1)}$. First, we extract word tags of true atomic formulas in each world to find the similarities and differences between the two worlds. Then we will find the probability of each transition relation (i.e., by NLP approaches) by finding the tags' similarities in both worlds. Here, tags are classes' labels of satisfied atomic formulas in that world. For example, assume $V(c_1) = \{p_{1,1}, \ldots, p_{1,i}\}$ and $V(c_2) = \{p_{2,1}, \ldots, p_{2,j}\}$, and let $p_{i',j'} = (x, c_{i',j'})$, then $c_{i',j'}$ represents the tag of atomic formula $p_{i',j'}$. Therefore, $C_1 = \{c_{1,1}, \ldots, c_{1,i}\}$ are corresponding tags of c_1, and $C_2 = \{c_{2,1}, \ldots, c_{2,j}\}$ are corresponding tags of c_2. Herein, the problem of finding tag similarities is finding similarity scores between C_1 and C_2. Several NLP-based methods determine the similarity score of these two sets of tags (see Oghbaie *et. al.,* [74], Yang *et. al.,* [75], Pawar *et. al.,* [76], and Mueller *et. al.,* [77]). After finding the similarity score of each transition relation, the most probable path would be found. This approach could be executed in time complexity $O(|\rightarrow|)$ ($|\rightarrow|$ represents the number of transition relations) by following a greedy algorithm. Note that the complexity of finding the similarity score is not considered because it differs due to various approaches with varied complexities. To introduce the approach, we assume the transition system was made of $n+1$ PAL models $\mathcal{M}_i = (W_i, R_{i_1}, \ldots, R_{i_n}, V), 0 \leq i \leq n$. Let $max\pi_{\mathcal{TS}}^{0\ldots(i-1)} \rightarrow c_{i_j}$ be the best-found path of length $i+1$, starting at the initial state s^0 to the world c_{i_j}. Moreover, let $\mathcal{P}_{\rightarrow_{j,k}^{i}}$ be the represented probability score between worlds c_{i_j} and $c_{(i+1)_k}$ (which is calculated similarity between tags in worlds c_{i_j} and $c_{(i+1)_k}$ by the NLP algorithm). The possibility score of the most probable path from s^0 to $c_{(i+1)_k}$ is $\mathcal{P}_{max\pi_{\mathcal{TS}}^{0\ldots i} \rightarrow c_{(i+1)_k}} = max_j(\mathcal{P}_{max\pi_{\mathcal{TS}}^{0\ldots(i-1)} \rightarrow c_{i_j}} \times \mathcal{P}_{\rightarrow_i^{j,k}})$, and the path would be $max\pi_{\mathcal{TS}}^{0\ldots i} \rightarrow c_{(i+1)_k} \equiv max\pi_{\mathcal{TS}}^{0\ldots(i-1)} \rightarrow c_{i_j} \rightarrow c_{(i+1)_k}$. The best path would be turned into one by continuing this approach to the state s^{-1}, and the most probable path would be found.

The most probable path could be involved in many real-world applications. For example, a frequent mistake in video object detection is misclassifications in some frames while tracking an object. The misclassifications might be happened because of the extraordinary situation (i.e., angle, low light, or partially overlapped object) of the object. Employing this algorithm would lead to getting rid of partial misclassifications. Algorithm 10 is developed to find the most probable path in the transition system.

Algorithm 10 The Most Probable Path Extraction (MPPE) function shall collect the most probable path of a data stream. The similarity score of tags in the 3rd line could be found with arbitrary NLP algorithms.

Let $\mathcal{TS}$ as a transition system

1: **function** MPPE($\mathcal{TS}$)
2: **for all** $(c_i, c_{i+1}) \in \rightarrow$ **do**
3: $\mathcal{P}_{(c_i,c_{i+1})} :=$ similarity score of tags in c_i and c_{i+1}
4: **for** $index \in \{1, \ldots, length(\pi_{\mathcal{TS}})\}$ **do**
5: **for all** $c_i \in W_i$ **do**
6: $maxP_{c_i} := 0$
7: $max\pi_{c_i} := \emptyset$
8: **for all** $\pi_{\mathcal{TS}}^{0\ldots i} = max\pi_{\mathcal{TS}}^{0\ldots i-1} \rightarrow c_i$ **do**
9: **if** $\mathcal{P}_{\pi_{\mathcal{TS}}^{0\ldots i}} > maxP_{c_i}$ **then**
10: $maxP_{c_i} := \mathcal{P}_{\pi_{\mathcal{TS}}^{0\ldots i}}$
11: $max\pi_{c_i} := \pi_{\mathcal{TS}}^{0\ldots i}$
12: **return** $max\pi_{s^{-1}}$

4.6 Conclusions

In this chapter, an approach was designed to formalize acquired knowledge by any classification. Next, by introducing LTPAL, an extension of PAL, the flow of knowledge in data streams was modeled. This model presents acquired knowledge in a transition system structure. It can examine the satisfaction of properties (LTPAL formulas) to solve classifiers' verification problems.

Consequently, we investigated the reliability of the answers. As an application, we explained a scenario in which "natural language translations to LTPAL" should be verified. Moreover, this model can answer the question: "which missing knowledge could lead the system to the correct answer?". Thus, the developed model was suitable for interpreting and verifying multi-classifiers that classify the data streams. We extended the model to consider the probabilities. Then, we found the most probable scenario for each data stream to assist classifiers.

Chapter 5

Tools

5.1 MASKS' Tool

MASKS is a tool to verify multi-classifier with predefined properties. Using MASKS, we check the satisfaction of the properties for all classifiers. In other words, if the property φ is satisfied using the knowledge of all agents, it would be the verified formula (property). We use the PAL formula $D_A\varphi$ to collect distributed knowledge of classifiers. Here, for more convenience, we developed python codes in three steps to run the tool. The first two steps create classifiers and validate inputs over them. Self-created ones could replace the first two steps.

The first one is "`0-create_model.py`", in which classifiers would be created. After defining the target dataset, the input data would be collected using the "load_input_data" function. This function's output would contain a train dataset set, and stored in the "train_data" variable. In this python file, the train data would be applied to train classifiers (i.e., ANNs). The architecture of the classifiers could be defined in the "define_model" function in "`build_model.py`". The number of output classes ("no_classes") and the input shape ("input_shape") should be determined. The

output would be multiple classifiers. These classifiers would be stored in the "Models" folder. The "model_no" variable could determine the number of classifiers

In order to run "`0-create_model.py`", the following inputs are required:

- The number of output classes: was stored in the "no_classes" folder,
- The dimension of input images: was stored in the "input_shape" folder,
- The number of classifiers to be created: was stored in the "model_no" folder,
- Train and validation dataset: were stored in "train_data" and "validation_data" folders (validation is optional),
- The classifier architecture: was defined in the "define_model" folder in the "`build_model.py`" file,
- If you use "`load_input_data.py`" for loading image files, the "train_df_path" should be set as the train file path.

Next, using the stored model in the "Models" folder, the tool could execute "`1-Eval_model.py`" to evaluate test inputs and their predefined properties. The set of neighborhoods and manipulations would be defined in the "NeighMan" *python class* (here is a set of noise on the input image). The output of "`1-Eval_model.py`" would be the results of inputs, neighborhoods, and manipulations for each classifier in a *NumPy* file in the "Results" folder.

In order to run "`1-Eval_model.py`", the following inputs are required:

- Image dimension: was stored in "img_width", "img_height", and "img_num_channels" folders,
- The input images: were stored in "input_test" folder,

- Outputs of test inputs (and their neighborhoods and manipulations) for each classifier: were stored *numpy* array in files "Results" folder,
- The number of neighborhoods and manipulations: was stored in "no_classes" folder,
- Ordered correct labels of inputs: were stored in "target_test" folder,
- The "project_path" and "data_dir" should be set as the project path and test file path.

Then, using the output results in the "Results" folder, the tool can execute "`2-MASKS.py`". The correct output labels would be provided with the function "load_labels". After execution of this python code, the following results for one to the number of classifiers would be provided:

1. "agent_counter": number of agents,
2. "correct_answer": correct verified answers,
3. "wrong_answer": wrong verified answer,
4. "conflict_answer": unverified answers because of conflicting,
5. "correct_assist": unverified answers because more than one output class is provided, but the correct answer is in the provided output classes,
6. "wrong_assist": unverified answers because more than one output class are provided, but the correct answer is not in the provided output classes.

For further investigation, the result of these multi-agent systems, every input would be stored in the "Agents" folder.

In order to run "`2-MASKS.py`", the following inputs are required:

- Outputs of test inputs (and their neighborhoods and manipulations) for each classifier: would be stored *NumPy* array in the files in the "Results" folder,
- The number of output classes: was stored in the "no_classes" folder,
- The number of neighborhoods and manipulations: was stored in the "nei_man_no" folder,
- Ordered correct labels of inputs: were stored in the "target_test" folder.

The tool and Modified Fashion MNIST, MNIST, and Fruit-360 can be found on `https://github.com/iuwa/MASKS` (examples are `FashionMNIST-MASKS.zip`, `MNIST-MASKS.zip`, and `Fruit-360-MASKS.zip`).

5.2 LTPAL Tool

This section describes the developed tool, which is done in the Python programming language. This tool aims at creating an LTPAL model for data stream inputs. To do this, first, we collect all knowledge gained from classifiers for specific input. This knowledge is "predicted output classes" collected by group A of classifiers. As mentioned, the knowledge is provided from a data stream so that classifiers would provide output classes for each data frame of the stream. Here, a Kripke model would be developed based on the provided knowledge for output classes of each data frame. Consequently, possible worlds of the Kripke model would be calculated. Thus far, for each data frame of the data stream, a Kripke model was provided. Using these Kripke models and adding them in sequential order, we could create a transition system as we described in section 4.4. Now, LTPAL formulas could be investigated over the paths of the transition system. Due to the theory provided in section 4.4, the condition of an LTPAL formula Φ (verified, possible, verified-missing, or possible-missing) over path $\pi_{\overline{\mathcal{TS}}}$ of the transition system is investigated.

- The LTPAL formula Φ is *verified* for group A of classifiers exactly when $D_A\Phi$ satisfies over path $\pi_{\overline{\mathcal{TS}}}$(defined in section 4.4.1).
- The LTPAL formula Φ is *possible* for group A of classifiers exactly when $\neg D_A \neg \Phi$ satisfies over path $\pi_{\overline{\mathcal{TS}}}$.

Next, we labeled each transition relation with a probability to develop the method of finding the most probable path in the transition system described in section 4.5. Then the most probable path would be found. In the following, we will describe the structure of the developed tool in detail.

5.2.1 Atomic Formula

As it is assumed, the classifiers have a specified output domain. It means that for a group of classifiers A, the aggregated knowledge would be a subset of C. The output domain defines all possible outcomes of classifiers for every input. Here, a pair (x, c) for each class $c \in C$ will be stored as the atomic formula. We have created a *python-class* namely "AtomicFormula" with the variables "inputData" and "name", which are referred to the specific data frame, and the name of the output class. Here, we have created an instance for each pair of input x and output class c.

5.2.2 Kripke Model

In this section, we will describe the development of the Kripke model in the tool. It should be considered that a PAL Kripke model is a tuple $M = (W, R_1, \ldots, R_n, V)$. Therefore, we developed a *python-class* called "KripkeModel", including W, $R_i; 1 \leq i \leq n$, and V. To create a proper *python-class* instance, the following steps should be taken. First, all output classes are collected for a specific input data frame. Then, the intersection of the classes provided by classifiers is calculated

(see algorithm 3). The cardinality of such output classes is the number of possible worlds in this step. Here, the initial Kripke model is created as follows:

- Let the cardinality of output classes as n,
- $W = \{c_1, ..., c_n, \bar{c}\}$,
- $R = \{(i, j) \mid i, j \in \{c_1, \dots c_n, \bar{c}\}\}$,
- $V(c_i) = \{(x, c_i)\}$, $V(\bar{c}) = \{(x, c_i) \mid i \in \{1, \dots n\}\}$, (x, c_i) is from the "AtomicFormula" class.

After creating the Kripke model $\mathcal{M}$, the PAL formula φ could be investigated for each world. Herein, for all provided PAL formulas and all worlds $c_i \in W$, if $\mathcal{M}, c_i \not\models \varphi$, the world is considered impossible and removed from the Kripke model $\mathcal{M}$. In this process, we also remove all relations with one end in c_i and the evaluation function $V(c_i)$. Therefore, the created Kripke model is reduced to $\mathcal{M}^{\varphi}$.

5.2.3 Transition System

In this step, a list of Kripke models was provided, each related to a data frame. Therefore, to create the transitions system, first, we created a *python-class*, named "TransitionSystem". It contains variables S, R, s^0, s^{-1}, $\rightarrow$, and L. This *python-class* contains a *python-methods* "add_kripke" that gets a "KripkeModel" object as input and appends it to the end of the "TransitionSystem". The transition system is created by calling this *python-method* for all Kripke models. The next step is to investigate the LTPAL formula φ in this transition system.

Get Probabilities

As mentioned in section 4.5, finding the most probable path has two main benefits.

- We could investigate the satisfaction of input formulas in one and the most probable path instead of all paths (the number of all paths is a multiplication of the number of worlds in each Kripke model).

- By finding the most probable path, we could find the most probable output classes for each data frame in a data stream input, which causes a correction for classifiers' output results.

The first step for finding the most probable path is to label each $\rightarrow$ with a probability. Thus, the "arrowProbability" variable was added to the "TransisionSystem" *python-class*. For each arrow $c_i \rightarrow c_j$, the probability refers to changing satisfied atomic formulas in $V(c_i)$ to ones in $V(c_j)$. Finding such probabilities is dependent on the input data. For instance, if we want to find the probability of $c_i \rightarrow c_j$ in a video-stream data, in which $V(c_i) = \{$AtomicFormula('cat'), AtomicFormula('dog')$\}$ and $V(c_j) = \{$AtomicFormula('cat'), AtomicFormula('chair'), AtomicFormula('tree')$\}$, we should find the probability of converting the set $\{$'cat', 'dog'$\}$ into the set $\{$'cat', 'chair', 'tree'$\}$ The probabilities are determined regarding the corresponding training set. Therefore, we developed a general NLP "GetProb" *python-module* to find these probabilities. Any arbitrary functions could be replaced. Then we created a bipartite fully connected graph in the developed "get_prob(set1, set2)" *python-method*. In each part, we have words in one input set and empty words in the number of another set. Here, in each part, we have |set1|+|set2| nodes. Each edge from one word in "set1" to a word in "set2" represents the probability of converting objects. Each edge from "set1" to an empty string shows the probability of removing such an object in the data stream (i.e., the probability of getting out of the image frame in a video stream). Each edge from an empty string to a word in "set2" represents the appearing such object in the following data frame. The spaCy NLP tool was applied in order to find the similarity between two words [78]. We applied "Max weight matching" after creating such a graph, which was done by the *networkx* toolbar [79]. This algorithm finds the maximum

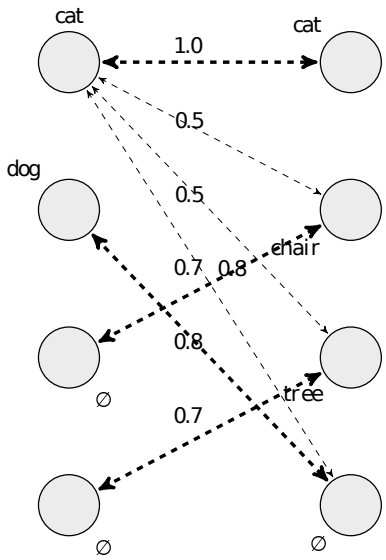


Figure 5.1: Finding probabilities between two worlds. Bold dashed arrows denote matched pairs, and the probability of changing the object "cat" into other objects is denoted by tiny dashed arrows. As can be seen, the best match for "cat" in the first data frame is the "cat" in the second one. Other objects' pairs are defined in the same way.

value of all converting pairs (see fig 5.1). The value $e^{max\ weight\ matching}$ represents the maximum probability of converting "set" into "set2". In general, it is not the best answer for finding the probability because the probabilities could be different for various problems. Finally, each "arrowProbability" would be labeled for each $\rightarrow$ in the transition system after finding these probabilities. Then using algorithm 10, the most probable path would be found.

5.2.4 Extended Data: how to use the LTPAL tool

We developed the LTPAL tool to verify the properties of multiple classifiers. Here, properties appear in the form of LTPAL formulas, which should be investigated in data streams (i.e., videos).

Input

As stated, the tool could apply to an ensemble of classifiers. Therefore, one should collect suitable classifiers. Then, each input (and its neighborhoods) should be fed into the classifiers. The output is a list of detected objects for each data frame, so there is a list of lists (nested list) of objects for each data stream. The input should consist of the following:

- Lexical ontology to define sub-features of all classes (for example "cat" is "animal"),
- LTPAL formulas (properties),
- PAL formulas (arbitrary constraints),
- Classifiers' output class domain (C),
- Number of frames,
- Classifiers' ids,
- Predictions for each frame.

We provide sample inputs on the tool's webpage ("`classifiersPredictions.json`" and "`subsetDict.json`" files) for more information.

Output

The expected output is a log file, including:

- Kripke model of each data frame,
- All execution paths,
- Evaluation of each property over path $\pi_{\overline{\mathcal{TS}}}$,

- Define whether each formula is a verified answer or a possible answer,
- The transition system, with probabilities of each arrow,
- The most probable path.

A sample output file is provided on the tool's webpage ("`output_log_file.log`" and "`result.json`" files).

Execution

After the configuring the input files, "`MASKS.py`" is executed using the `Python3`.

The file "subsetDict.json" contains all parental relations between output classes. Sample "subsetDict.json":

```
{
  "cat": [ "animal", "thing", "mammal" ],
  "dog": [ "animal", "thing", "mammal" ],
  "chair": [ "thing" ],
  "mammal": [ "animal", "thing" ],
  "animal": [ "thing" ]
}
```

The file "`classifiersPredictions.json`" contains all LTPAL and PAL formulas, output classes, number of frames, classifiers' ids, and each frame's extracted information for each classifier. Sample "`classifiersPredictions.json`":

```
{
  "formulas_LTPAL": [ "X_iU_i(cat,chair)", "X_i&(K_i~&(cat,dog)
   ,cat)", "X_ichair" , "X_idog" ],
```

```
  "formulas_PAL": [ "&(K_i~&(cat,dog),cat)" ],
  "allClasses": [ "cat", "dog", "chair" ],
  "number_of_frames": 2,
  "classifiers_ids": [ "1", "2" ],
  "1": [
    [{"name": "cat"},{"name": "dog"},{"name": "chair"} ],
    [{"name": "cat"},{"name": "dog"} ]
  ],
  "2": [
    [{"name": "cat"},{"name": "chair"} ],
    [{"name": "cat"} ]
  ]
}
```

The file "`result.json`" is created after the execution, which contains the created transition system. Sample "`result.json`":

```
{
    "S": [[[0, 0]], [[1, 2], [1, 3]], [[2, 1]], [[3, -1]]],
    "R": [[[[0, 0], [0, 0]]], [[[1, 2], [1, 2]], [[1, 2], [1, 3
  ]], [[1, 3], [1, 2]], [[1, 3], [1, 3]]], [[[2, 1], [2, 1]]],
    [[[3, -1], [3, -1]]]],
    "s_0": 0,
    "s_1": -1,
    "Arrow": [[[0, 0], [1, 2]], [[0, 0], [1, 3]], [[1, 2], [2,
```

```
   1]], [[1, 3], [2, 1]], [[2, 1], [3, -1]]],
7    "L": {"0_0": [], "1_2": [{"name": "cat"}], "1_3": [{"name":
     "chair"}]}, {"name": "cat"}], "2_1": [{"name": "cat"}], "3_
   -1": []}
8 }
```

The execution would also provide a file containing information extracted from the inputs. Each frame provides each classifier's outputs, the shared information, and the Kripke model for that frame. Then, all execution paths are provided. Next, for each input formula, the tool provides the state of the formula (i.e., verified/possible). Consequently, the transition system is provided. Sample "output_log_file.log":

```
1 MASKS started ---------------------------------->
2 ____Creating Krikpke Model for frame: 0________
3 __________classifiers_prediction_____________
4 [[{'name': 'cat'}, {'name': 'dog'}, {'name': 'chair'}], [{'name
     ': 'cat'}, {'name': 'chair'}]]
5 __________arrayOfOutputClasses_____________
6 [[(cat), (dog), (chair)], [(cat), (chair)]]
7 ______Intersected arrayOfOutputClasses________
8 [[(cat), (chair)], [(cat), (chair)]]
9 The input formula: &(K_i~&(cat,dog),cat) removed list of worlds
     : [1] with names [['chair']] in the kripke Model of frame: 0
10 __________kripke_____________ for frame number: 0
11 (
```

```
W=[2, 3],
R=[(2, 2), (2, 3), (3, 2), (3, 3)],
V={2: [{"name": "cat"}],
   3: [{"name": "chair"}, {"name": "cat"}]})
_______Creating Krikpke Model for frame: 1_________
__________classifiers_prediction_____________
[[{'name': 'cat'},{'name': 'dog'}],[{'name': 'cat'}]]
__________arrayOfOutputClasses_____________
[[(cat), (dog)], [(cat)]]
______Intersected arrayOfOutputClasses________
[[(cat)], [(cat)]]
no world removed by PAL formula for frame number: 1
__________kripke_____________ for frame number: 1
(
W=[1],
R=[(1, 1)],
V={1: [{"name": "cat"}]})
__________Paths_____________
[[(0, 0), (1, 2), (2, 1), (3, -1)], [(0, 0), (1, 3), (2, 1), (3
   , -1)]]
__________Validating Formulas_____________
The input formula: X_iU_i(cat,chair) is --verified--
The input formula: X_iU_i(cat,chair) is --possible--
The input formula: X_i&(K_i~&(cat,dog),cat) is --verified--
```

```
35 The input formula: X_i&(K_i~&(cat,dog),cat) is --possible--
36 The input formula: X_ichair is --possible--
37 The Transition System model is: (
38 S=[[(0, 0)], [(1, 2), (1, 3)], [(2, 1)], [(3, -1)]],
39 R=[[((0, 0), (0, 0))], [((1, 2), (1, 2)), ((1, 2), (1, 3)), ((1
      , 3), (1, 2)), ((1, 3), (1, 3))], [((2, 1), (2, 1))], [((3,
      -1), (3, -1))]],
40 s_0=0,
41 s_1=-1,
42 Arrow=[((0, 0), (1, 2)), ((0, 0), (1, 3)), ((1, 2), (2, 1)), ((
      1, 3), (2, 1)), ((2, 1), (3, -1))],
43 L={(0, 0): [], (1, 2): [{"name": "cat"}], (1, 3): [{"name": "
      chair"}, {"name": "cat"}], (2, 1): [{"name": "cat"}], (3, -1
      ): []})
44 MASKS ended <-----------------------------------
```

The tool is available at https://github.com/iuwa/LTPAL.

Chapter 6

Conclusion and future work

This study sets out to investigate how multi-agent learning systems can be verified by defining epistemic logic. We did this by defining safety properties for classifiers; one can define arbitrary properties based on the classification problem. Using these properties, we created property sets as inputs. Classifiers processed these sets of inputs and provided sets of output labels. Then, by defining public announcement logic, we analyzed the output labels and evaluated the properties' verification for single-object classifiers. Then, we extend the approach to analyze multi-object classifiers' information. A considerable improvement was observed by applying the approach to three standard datasets: Fashion-MNIST, MNIST, and Fruit-360. An analysis of failures is provided. We also provide algorithms. Although data stream inputs can be assumed as a series of independent input data frames, we showed that considering the relation between input frames could enhance the analysis in many cases. First, we defined Linear Temporal Public Announcement Logic (LTPAL) to consider the relations over time. It is a combination of PAL and temporal logic. PAL was applied to interpret single frames, and the timed transition between frames was analyzed with temporal logic. The resulting model is a transition system created by

Kripke models interconnected with transition relations. We extended the definition of verification for the transition system to consider a sequence of frames. Additionally, we can verify LTPAL formulas as properties of multi-classifiers in the defined verification. After defining the verification of classifiers for data stream entries, an application of the LTPAL model was presented. Considering the semi-continuous characteristic of data streams, we found the most probable sequence of labels for a data stream input. It is done by finding the probability of labels changing in consecutive frames assigned to each transition relation.

Future Work

We developed MASKS and LTPAL tools to verify classifiers based on predefined properties. The following modifications will be done to improve the tools:

- As mentioned before, the LTPAL tool is an extension of MASKS; therefore, we will define a unified input-output language to unify the two.
- In addition, in the current MASKS tool, the definition of properties is limited to *neighborhood* and *manipulations*, and they are hard-coded in the tool. We will develop an approach to input any arbitrary properties from user.
- The developed model investigates the knowledge represented by classifiers after the process ends. In some cases, the process will never end. Thus, the reasoning system should be applied while the system is running. Applying such a real-time reasoning system enables us to capture new and more complex information about objects without shutting down the classifiers or changing them. We will develop the LTPAL tool as a real-time system (RTS) to yield whether the LTPAL formula is satisfied.

- Due to the nature of most classifiers, the output domain is limited to predefined classes. Thus, in time series classifiers, defined actions are limited too. To be more formal, assuming objects are a language's syntax, then actions will be a formula of that language. Although it is possible to define the syntax for each action, it is essential to be able to define a huge number of actions. In the future, we will describe a knowledge-based system (KBS) that could capture actions without setting them as output labels of classifiers.

- In order to create a KBS, it is required to capture the whole knowledge produced by classifiers. A part of this knowledge can be captured with the method presented in section 4.3, but the mentioned approach would not consider all knowledge produced by classifiers. For example, in YOLO (You Only Look Once: Unified, Real-Time Object Detection), which is an image object detection tool, in addition to the name of the detected object, some other specifications (x, y, w, h, and IOU) are also reported. In this classifier, pair (x, y) as a coordinate represents the center of the box that contains the respective object, pair (w, h) is the width and height, and Intersection Over Union (IOU) is the confidence of the prediction. These specifications could vary for different classifiers and applications. We will develop a KBS to capture arbitrary properties to cover all the specifications. As we defined in section 4.3, atomic formulas are in the form of (x, c), in which x is the input and c is the class that appeared in the input. In order to add any arbitrary properties provided by the classifier, the definition of the atomic formula will be changed to tuple (x, c, p), in which p is a tuple of provided properties. Each of these properties can hold a value or $\emptyset$ if the value is not provided. Using the definition of atomic formula, we have $(x, c, p) \vDash (x, c)$ for all p, which means if class c is detected in an input with a classifier, provided properties will not ignore the appearance. Following that, we also have:

$$(x, c, p) \vDash (x, c, p'),\ iff\ \forall_{i \in p, i' \in p'}\ if\ i \neq i'\ then\ i' \subseteq i$$

This leads us to $(x, c) \equiv (x, c, p),\ iff\ p = (\emptyset, \emptyset, \ldots, \emptyset)$. In order to create the Kripke model, the evaluation of each atomic formula in each world must be defined.

- Current developed tools get inputs and pass outputs through commands. We will design a graphical user interface (GUI) to make the tools more accessible.

Finally, a method will be developed to verify classifiers as learning systems. In the future, we will extend the tool to define the verification of other learning systems. In other learning systems, fuzzy logic can play the primary verification role because of the continuity of outputs. We will examine the application of fuzzy logic in verifying learning systems.

Bibliography

[1] S. Albrecht and P. Stone, "Multiagent learning: foundations and recent trends," in *Tutorial at IJCAI-17 conference*, vol. 223, 2017.

[2] A. L. Samuel, "Machine learning," *The Technology Review*, vol. 62, no. 1, pp. 42–45, 1959.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] T. Kropf, *Introduction to formal hardware verification*. Springer Science & Business Media, 1999.

[5] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings* (T. Touili, B. Cook, and P. B. Jackson, eds.), vol. 6174 of *Lecture Notes in Computer Science*, pp. 243–257, Springer, 2010.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[7] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*, pp. 3–29, Springer, 2017.

[8] E. A. Emerson, "Temporal and modal logic," in *Formal Models and Semantics*, pp. 995–1072, Elsevier, 1990.

[9] P. Blackburn, M. De Rijke, and Y. Venema, *Modal logic: graph. Darst*, vol. 53. Cambridge University Press, 2001.

[10] R. Rendsvig and J. Symons, "Epistemic logic," 2019.

[11] P. Øhrstrøm and P. Hasle, *Temporal logic: From ancient ideas to artificial intelligence*, vol. 57. Springer Science & Business Media, 2007.

[12] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, ieee, 1977.

[13] A. H. Dehkordi, M. Alizadeh, and A. Movaghar, "Meet masks: A novel multi-classifier's verification approach," 2020.

[14] A. H. Dehkordi, M. Alizadeh, and A. Movaghar, "Linear temporal public announcement logic: a new perspective for reasoning about the knowledge of multi-classifiers," 2020.

[15] S. Irani and N. Venkatasubramanian, "Semi-continuous transmission for cluster-based video servers," in *Third IEEE International Conference on Cluster Computing (CLUSTER'01)*, pp. 303–303, IEEE Computer Society, 2001.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[17] J. Plaza, "Logics of public communications," *Synthese*, vol. 158, no. 2, pp. 165–179, 2007.

[18] L. Wang, "Research and implementation of machine learning classifier based on KNN," *IOP Conference Series: Materials Science and Engineering*, vol. 677, p. 052038, dec 2019.

[19] J. Van Benthem, "Modal logic: A contemporary view," *Internet Encyclopedia of Philosophy, URL: http://www. iep. utm. edu/modal-lo*, 2016.

[20] J. Garson, "Modal Logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, Summer 2021 ed., 2021.

[21] R. Fagin, J. Y. Halpern, Y. Moses, and M. Vardi, *Reasoning about knowledge*. MIT press, 2004.

[22] O. Gasquet, A. Herzig, B. Said, and F. Schwarzentruber, *Kripke's worlds: An introduction to modal logics via tableaux*. Springer Science & Business Media, 2013.

[23] A. Prior, *Time and Modality*. Oxford: Oxford University Press, 1957.

[24] A. Prior, *Past, Present and Future*. Oxford: Oxford University Press, 1967.

[25] V. Goranko and A. Rumberg, "Temporal Logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, Summer 2022 ed., 2022.

[26] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, "On the temporal analysis of fairness," in *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 163–173, 1980.

[27] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402, Springer, 2013.

[28] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[29] J. Törnblom and S. Nadjm-Tehrani, "Formal verification of input-output mappings of tree ensembles," *Science of Computer Programming*, vol. 194, p. 102450, 2020.

[30] S. A. Seshia and D. Sadigh, "Towards verified artificial intelligence," *CoRR*, vol. abs/1606.08514, 2016.

[31] D. Sadigh, *Safe and Interactive Autonomy: Control, Learning, and Verification*. PhD thesis, UC Berkeley, 2017.

[32] K. Scheibler, L. Winterer, R. Wimmer, and B. Becker, "Towards verification of artificial neural networks," in *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, MBMV 2015, Chemnitz, Germany, March 3-4, 2015*. (U. Heinkel, D. Kriesten, and M. Rößler, eds.), pp. 30–40, Sächsische Landesbibliothek, 2015.

[33] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[34] D. Gross, N. Jansen, G. A. Pérez, and S. Raaijmakers, "Robustness verification for classifier ensembles," in *International Symposium on Automated Technology for Verification and Analysis*, pp. 271–287, Springer, 2020.

[35] P. Bjesse, "What is formal verification?," *ACM SIGDA Newsletter*, vol. 35, no. 24, pp. 1–es, 2005.

[36] J. Van Benthem, J. Van Eijck, and B. Kooi, "Logics of communication and change," *Information and computation*, vol. 204, no. 11, pp. 1620–1662, 2006.

[37] P. BALBIANI, A. BALTAG, H. V. DITMARSCH, A. HERZIG, T. HOSHI, and T. DE LIMA, "'knowable' as 'known after an announcement'," *The Review of Symbolic Logic*, vol. 1, no. 3, p. 305–334, 2008.

[38] R. Rendsvig and J. Symons, "Epistemic logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, summer 2019 ed., 2019.

[39] A. Baltag and B. Renne, "Dynamic epistemic logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, winter 2016 ed., 2016.

[40] C. Menzel, "Possible worlds," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, winter 2017 ed., 2017.

[41] J. Plaza, "Logics of public announcements," in *Proceedings 4th International Symposium on Methodologies for Intelligent Systems*, 1989.

[42] H. P. Van Ditmarsch, "Comments to 'logics of public communications'," *Synthese*, vol. 158, no. 2, pp. 181–187, 2007.

[43] Y. Wang and Q. Cao, "On axiomatizations of public announcement logic," *Synthese*, vol. 190, no. 1, pp. 103–134, 2013.

[44] Y. N. Wáng and T. Ågotnes, "Public announcement logic with distributed knowledge," in *International Workshop on Logic, Rationality and Interaction*, pp. 328–341, Springer, 2011.

[45] H. Van Ditmarsch, W. van Der Hoek, and B. Kooi, *Dynamic epistemic logic*, vol. 337. Springer Science & Business Media, 2007.

[46] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[47] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[48] H. Mureşan and M. Oltean, "Fruit recognition from images using deep learning," *arXiv preprint arXiv:1712.00580*, 2017.

[49] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, pp. 1–15, Springer, 2000.

[50] G. Auda, M. Kamel, and H. Raafat, "Voting schemes for cooperative neural network classifiers," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 3, pp. 1240–1243, IEEE, 1995.

[51] P. H. Winston, "Artificial intelligence 3rd edition," *Addison-Wesley, Reading, MA*, vol. 34, pp. 167–339, 1992.

[52] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2002.

[53] N. J. Nilsson, *The quest for artificial intelligence*. Cambridge University Press, 2009.

[54] R. P. Goldman and E. Charniak, "Probabilistic text understanding," *Statistics and Computing*, vol. 2, no. 2, pp. 105–114, 1992.

[55] J. Gao, C. Sun, Z. Yang, and R. Nevatia, "Tall: Temporal activity localization via language query," in *Proceedings of the IEEE international conference on computer vision*, pp. 5267–5275, 2017.

[56] L. Anne Hendricks, O. Wang, E. Shechtman, J. Sivic, T. Darrell, and B. Russell, "Localizing moments in video with natural language," in *Proceedings of the IEEE international conference on computer vision*, pp. 5803–5812, 2017.

[57] L. Bauer, Y. Wang, and M. Bansal, "Commonsense for generative multi-hop question answering tasks," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 4220–4230, Association for Computational Linguistics, Oct.-Nov. 2018.

[58] K. Ma, J. Francis, Q. Lu, E. Nyberg, and A. Oltramari, "Towards generalizable neuro-symbolic systems for commonsense question answering," *arXiv preprint arXiv:1910.14087*, 2019.

[59] R. Speer and J. Lowry-Duda, "Conceptnet at semeval-2017 task 2: Extending word embeddings with multilingual relational knowledge," *arXiv preprint arXiv:1704.03560*, 2017.

[60] M. Sap, R. Le Bras, E. Allaway, C. Bhagavatula, N. Lourie, H. Rashkin, B. Roof, N. A. Smith, and Y. Choi, "Atomic: An atlas of machine commonsense for if-then reasoning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3027–3035, 2019.

[61] K. Yi, C. Gan, Y. Li, P. Kohli, J. Wu, A. Torralba, and J. B. Tenenbaum, "Clevrer: Collision events for video representation and reasoning," *arXiv preprint arXiv:1910.01442*, 2019.

[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[63] B. Fong, A. Speranzon, and D. I. Spivak, "Temporal landscapes: A graphical temporal logic for reasoning," *arXiv preprint arXiv:1904.01081*, 2019.

[64] D. de Leng and F. Heintz, "Approximate stream reasoning with metric temporal logic under uncertainty," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 2760–2767, 2019.

[65] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *International Conference on Protocol Specification, Testing and Verification*, pp. 3–18, Springer, 1995.

[66] P. Yang, Z. Zhang, B. B. Zhou, and A. Y. Zomaya, "Sample subset optimization for classifying imbalanced biological data," in *Advances in Knowledge Discovery and Data Mining* (J. Z. Huang, L. Cao, and J. Srivastava, eds.), (Berlin, Heidelberg), pp. 333–344, Springer Berlin Heidelberg, 2011.

[67] A. Hossain and F. Laroussinie, "From quantified ctl to qbf," *arXiv preprint arXiv:1906.10005*, 2019.

[68] M. Kwiatkowska, A. Lomuscio, and H. Qu, "Parallel model checking for temporal epistemic logic," in *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pp. 543–548, 2010.

[69] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[70] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution," in *2009 IEEE International Conference on Robotics and Automation*, pp. 4163–4168, IEEE, 2009.

[71] R. Nelken and N. Francez, "Automatic translation of natural language system specifications into temporal logic," in *International Conference on Computer Aided Verification*, pp. 360–371, Springer, 1996.

[72] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1988–1993, IEEE, 2010.

[73] J. H. Bappy, S. Paul, E. Tuncel, and A. K. Roy-Chowdhury, "Exploiting typicality for selecting informative and anomalous samples in videos," *IEEE Transactions on Image Processing*, vol. 28, no. 10, pp. 5214–5226, 2019.

[74] M. Oghbaie and M. M. Zanjireh, "Pairwise document similarity measure based on present term set," *Journal of Big Data*, vol. 5, no. 1, p. 52, 2018.

[75] W. Yang, W. Lu, and V. W. Zheng, "A simple regularization-based algorithm for learning cross-domain word embeddings," *arXiv preprint arXiv:1902.00184*, 2019.

[76] A. Pawar and V. Mago, "Calculating the similarity between words and sentences using a lexical database and corpus statistics," *arXiv preprint arXiv:1802.05667*, 2018.

[77] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *thirtieth AAAI conference on artificial intelligence*, 2016.

[78] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spacy: Industrial-strength natural language processing in python," 2020.

[79] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

پژوهشگاه دانش های بنیادی

(مرکز تحقیقات فیزیك نظري و رياضيات)

پژوهشکده علوم کامپیوتر

پایان‌نامه دکتری

درست یابی سیستم های یادگیری چند عاملی بر پایه منطق شناختی

امیرهوشنگ حسین پور دهکردی

اساتید راهنما:

علی موقر

مجید علیزاده

زمستان ۱۴۰۱

چکیده

سیستم یادگیری چند عاملی [1] زیر شاخه ای از سیستم های یادگیری است، که بر مطالعه رفتارهای چند سیستم یادگیری در یک محیط مشترک تمرکز دارد. در این سیستم ها هر عامل وظایف خود را به صورت مستقل انجام می دهد. در اینجا، عوامل وظایف خود را با استفاده از اطلاعات جمع آوری شده در یک فرآیند یادگیری انجام می دهند. برای هر ورودی، نتایج تجمیع شده همه عامل ها نمایش داده می شود. این سیستم یادگیری چند عاملی همه اطلاعات و نحوه مدل سازی سیستم های یادگیری چند عاملی را مدل می کند. سیستم یادگیری چند عاملی می تواند یک سیستم یادگیری را با افزایش دقت یا کاهش خطاها تقویت کند.

امروزه سیستم های یادگیری کیفیت زندگی انسان را افزایش داده اند. و این باعث شده است در اکثر برنامه ها گسترش پیدا کنند. به عنوان مثال، ما می توانیم سیستم های یادگیری را در پزشکی، خودروهای خودران و سیستم های کنترل کیفیت صنعتی ببینیم. اگرچه استفاده از سیستم های یادگیری خطاها را در مقایسه با انسان کاهش می دهد، اما هر خطا در این سیستم ها می تواند عواقب جدی ایجاد کند. به دلیل ماهیت آماری سیستم های یادگیری، خطاها قابل تفسیر نیستند. به عبارت دیگر، هیچ‌کس نمی‌تواند پیش‌بینی کند که چه زمانی یا چرا یک خطا رخ می‌دهد. برای حل این مشکل، دانشمندان شروع به درستی‌یابی ویژگی های یک سیستم کردند تا مطمئن شوند که سیستم در شرایط خاص به خوبی کار می کند. استفاده گسترده از سیستم های یادگیری درستی‌یابی سیستم های یادگیری را ضروری کرده است.

سرفصل های چکیده پایان نامه

این پایان نامه درستی‌یابی سیستم یادگیری چند عاملی بر اساس منطق شناختی را ارائه می کند. در اینجا، در فصل ۱.۲ مقدماتی بر سیستم های یاد گیری و در فصل ۲.۲ درست یابی آن‌ها ارائه می‌شود. پس از آن در مورد طبقه بندی کننده ها [2] در ۳.۲ و سپس، مبانی منطق موجهات در ۴.۲ ارائه می‌شود. در این پایان‌نامه، منطق شناختی و منطق زمانی که از خانواده‌های منطق وجهی هستند، استفاده خواهند شد. منطق شناختی در بخش ۱.۴.۲ معرفی خواهد شد، و منطق زمانی در بخش ۲.۴.۲ معرفی خواهد شد. در فصل ۳ ، مدلی برای درستی‌یابی ایجاد خواهد شد. طبقه بندی کننده های چند عاملی در این فصل ابتدا یک منطق اعلام عمومی [3] برای تفسیر انباشت دانش توزیع شده ایجاد می کنند. یک الگوریتم اشتراک

[1] Multi-Agent Learning System
[2] classifier
[3] Public Announcement Logic

دانش سیستم های چند عاملی [۴] برای تأیید ویژگی های از پیش تعریف شده در بخش ۲.۳ توسعه خواهد یافت. سپس، فصل ۳.۳ ترکیبی از منطق اعلام عمومی و منطق زمانی خطی [۵] ، یعنی منطق خطی اعلام عمومی موقت [۶] ، را برای تعریف ویژگی ها در یک سیستم انتقال ارائه می‌کند. این کار برای استخراج دانش در طبقه‌بندی‌کننده‌ها با ورودی‌های جریان داده [۷] انجام خواهد شد.

[۴] MASKS
[۵] LTL
[۶] LTPAL
[۷] Data Stream

فصل ۱

مقدمات

۱.۱ سیستم های یادگیری

سیستم یادگیری سیستمی با توانایی یادگیری بدون توسعه برنامه است. سیستم یادگیری معمولاً به سیستمی گفته می شود که پس از یک مرحله یادگیری [۱] عمل می کند. این سیستم ها برای یادگیری از تجربیات مرحله یادگیری و بهبود عملکرد با استفاده از این تجربیات طراحی شده اند. یک سیستم یادگیری در حالت اولیه تصادفی طراحی می شود و با پارامترهای اختصاص داده شده مرحله یادگیری آغاز می شود. در طول مرحله یادگیری، داده های آموزشی پارامترهای سیستم های یادگیری را تنظیم می کنند. پس از مرحله یادگیری، سیستم یادگیری آماده است تا مشکل مورد نظر را حل کند. سیستم های یادگیری بسته به ماهیت مسأله به سه دسته تقسیم می شوند:

- **یادگیری نظارت‌شده** [۲] **(یادگیری با داده‌های برچسب‌دار):** در این مسائل داده ها با ورودی و خروجی ارائه می شوند. یادگیری نظارت‌شده بر اساس پیوستگی خروجی ها به طبقه بندی و رگرسیون تقسیم می شود. مسائل طبقه‌بندی، با هدف تخصیص یک برچسب از یک مجموعه از پیش تعریف‌شده به یک ورودی طراحی می شوند. در مقابل، رگرسیون ها یک مقدار عددی را به یک ورودی اختصاص می دهند.

- **یادگیری خودران** [۳] **(خودسازمانده):** هیچ خروجی برای الگوریتم های یادگیری خودسازمانده ارائه نمی شود. در روش خودسازمانده ساختار ورودی ها پیدا می شود.

- **یادگیری تقویتی** [۴] **:** با یک هدف از پیش تعریف شده آغاز می شود، عامل ها با یک محیط پویا تعامل دارند تا میزان تابع هدف را به حداکثر برسانند. محیط بر اساس اقدامات به عامل ها پاداش می دهد و هدف هر عامل را ارزیابی می کند.

---

[۱] Learning Phase
[۲] Supervised Learning
[۳] Unsupervised Learning
[۴] Reinforce Learning

به طور کلی، در بین این سیستم های یادگیری، اطلاعات ارائه شده توسط الگوریتم های یادگیری نظارت شده به سادگی قابل ارزیابی است. الگوریتم های یادگیری خودسازمانده سعی می کنند بهترین ساختار را پیدا کنند و الگوریتم های یادگیری تقویتی سعی می کنند بهترین استراتژی را برای به حداکثر رساندن تابع هدف بیابند. اگرچه معیارهایی برای ارزیابی الگوریتم های یادگیری خودسازمانده و یادگیری تقویتی وجود دارد، اما روش های سر راست برای آنها ارائه نشده است. واضح ترین دلیل آن عدم وجود برچسب یا جواب مشخص برای هر ورودی است. به همین دلیل عموما معیار ها میزان بهبود کارایی آنها را بیان می کنند.

۲.۱ درستی‌یابی سیستم های یادگیری

در سیستم های کامپیوتری، درستی‌یابی صوری، فرآیند بررسی ارضا یک ویژگی [۵] با استفاده از یک روش ریاضی است. از متداول‌ترین رویکردهای ریاضی برای درستی‌یابی صوری، ماشین‌های حالت محدود [۶] ، سیستم‌های انتقال [۷] ، شبکه‌های پتری [۸] ، اتوماتای زمان‌دار [۹] ، اتوماتای ترکیبی [۱۰] ، جبر فرآیند [۱۱] ، منطق شناختی [۱۲] و منطق زمانی [۱۳] را می توان نام برد. یکی از اولین رویکردهای درستی‌یابی صوری، وارسی مدل است که در آن کل حالت های ممکن ماشین بررسی می شود. این روش می تواند برای مدل های حالت محدود اعمال شود. الگوریتم‌های وارسی مدل در سیستم‌هایی با حالت‌های بسیار زیاد، حافظه و زمان‌بر خواهند بود. بنابراین رویکردهای کاربردی تری ارائه شد که در آن ویژگی های خاصی درستی‌یابی می شد. در این روش ها بوسیله یک مدل ریاضی، ارضی این ویژگی ها در مدل مورد ارزیابی قرار می گرفت. با ظهور سیستم های یادگیری، مانند شبکه های عصبی عمیق [۱۴] ، اندازه سیستم ها افزایش یافت و چنین رویکردهای درستی‌یابی صوری را نمی شد برای مشکلات دنیای واقعی اعمال کرد. به عنوان مثال، روش توسعه یافته توسط پولینا و همکاران، در سال ۲۰۱۰ میلادی به مدل سازی یک شبکه عصبی [۱۵] با حداکثر شش نورون محدود می شد، در حالی که الکس نت [۱۶] که در مسابقه ILSVRC-2012 شرکت کرده بود، شامل حدود ۶۵۰۰۰۰ نورون بود. برای غلبه بر این مشکل، هوانگ و همکاران، از درستی‌یابی نقطه ای برای درستی‌یابی یک ویژگی برای یک ورودی سیستم استفاده کردند. درستی‌یابی نقطه‌ای با موفقیت برای تأیید شبکه های عصبی عمیق بزرگتر اعمال شد.

---

[۵] property
[۶] Finite state machine
[۷] Transition system
[۸] Petri nets
[۹] Timed automata
[۱۰] Hybrid automata
[۱۱] Process algebra
[۱۲] Epistemic logic
[۱۳] Temporal logic
[۱۴] Deep Neural Network
[۱۵] Neural Network
[۱۶] Alexnet

۳.۱ طبقه بندی کننده ها

الگوریتم های طبقه بندی به طور گسترده در بسیاری از برنامه های کاربردی استفاده می شود. تشخیص تقلب در بانکداری، آشکارسازها و تشخیص‌دهنده‌های شی در خودروهای خودکار، آشکارسازهای تشخیص پزشکی، و آشکارسازهای خطا در فناوری هوافضا نمونه‌هایی از کاربرد این الگوریتم‌ها هستند. هدف اصلی طبقه بندی کننده یافتن مناسب ترین برچسب از مجموعه ای از برچسب ها برای داده های ورودی است. به عبارت دیگر، طبقه بندی، تقسیم یک فضای برداری ورودی به چندین کلاس است. ورودی‌ها بردارهایی در فضای n بعدی اقلیدسی با متریک فاصله دلخواه هستند. هدف طبقه بندی کننده انجام فرآیند طبقه بندی است. در حال حاضر، متداول‌ترین طبقه‌بندی‌کننده‌ها توسط الگوریتم‌های یادگیری ماشین پیاده سازی می‌شوند. اولین گام برای ایجاد یک طبقه‌بندی کننده بر پایه یادگیری ماشین، انتخاب یک مدل است. رگرسیون لجستیک [۱۷] ، درخت تصمیم [۱۸] ، ماشین بردار پشتیبان [۱۹] ، K ـ نزدیکترین همسایه [۲۰] و شبکه های عصبی مصنوعی [۲۱] نمونه هایی از الگوریتم های طبقه بندی کننده بر پایه یادگیری ماشین هستند. این الگوریتم‌ها در مرحله آموزشی آموزش می‌بینند که در آن یک مجموعه داده آموزشی برای ایجاد طبقه‌بندی‌کننده‌های بر پایه یادگیری ماشین به ورودی داده می‌شود. مجموعه داده آموزشی شامل ورودی های نمونه با برچسب صحیح است. در اینجا، طبقه بندی کننده انتظار دارد که رابطه ای بین ورودی ها و برچسب های مربوطه پیدا کند. به عبارت دیگر، برای تعریف مرحله آموزش، $X = \{x_1, \dots x_n\}$ یک مجموعه داده، $Y = \{y_1, \dots, y_n\}$ برچسب های ارائه شده مربوطه و $C = \{c_1, \dots, c_k\}$ مجموعه ای از برچسب ها، که در آن برای همه i ، $y_i \in C$ و $X_{c_i} = \{x_j \mid y_j = c_i\}$ است فرض کنید. هدف مرحله آموزش کشف وابستگی‌های بین X_{c_i} برای همه i ها است. بنابراین، کیفیت مجموعه داده‌های آموزشی و معماری مدل مهم‌ترین نقش را ایفا می‌کنند. در ادامه، شبکه های عصبی مصنوعی ها را به عنوان یکی از پرکاربردترین روش های طبقه بندی کننده بر پایه یادگیری ماشین، مورد بحث قرار می دهیم.

همانطور که قبلا ذکر شد، ورودی های یک طبقه بندی کننده به صورت بردار با بعد n_0 هستند. در شبکه های عصبی مصنوعی، خروجی با یک بردار e_i با بعد n_{-1} نشان داده می شود، که در آن مقدار عنصر i ام بزرگترین است (تابع $softmax$ فرآیند انتخاب حداکثر مقدار را انجام می دهد. فرض کنید $C = \{c_1, \dots, c_{n_{-1}}\}$ مجموعه برچسب‌ها باشد، برای ورودی e_i ، c_i برچسبی است که شبکه های عصبی مصنوعی برای ورودی در خروجی ارائه می کنند. تبدیل بردار ورودی به بردار خروجی با طراحی یک گراف k بخشی کاملا متصل و جهت دار انجام می شود. اولین بخش گراف نمودار ورودی و آخرین بخش لایه خروجی است. این دو لایه برای شبکه های عصبی مصنوعی اجباری هستند، اما افزودن لایه ها بین آنها نیز امکان پذیر است. در این مورد، ما آن را شبکه عصبی عمیق [۲۲] می نامیم. هر لایه می تواند به هر لایه دیگر با یال های وزن دار متصل شود. هنگامی که یک ورودی وارد می شود، فرآیند شروع می شود. به صورت استقرایی، مقدار لایه بعدی با اعمال یک تابع غیرخطی بر روی مجموع مقدار ارائه شده توسط هر یال ورودی محاسبه می شود. مقدار هر یال ورودی، مقدار گره منبع را با وزن یال ضرب می کند. این روند تا محاسبه وزن آخرین لایه ادامه خواهد یافت. در اینجا وزن ها باید در فرایند یادگیری

[۱۷] Logistic regression
[۱۸] Decision Tree
[۱۹] Support vector machine (SVM)
[۲۰] K-Nearest Neighborhood
[۲۱] Artificial Neural Networks
[۲۲] Deep Neural Network

تعریف شوند. فرآیند یادگیری در شبکه عصبی مصنوعی با وزن های تصادفی آغاز می شود. مجموعه داده آموزشی را یک به یک به شبکه های عصبی مصنوعی وارد می کنیم. شبکه های عصبی مصنوعی برای هر ورودی در آخرین لایه پاسخی ارائه می دهد و ما پاسخ صحیح را می دانیم (در مجموعه داده آموزشی ارائه شده است). پس از آن، وزن ها با توجه به تفاوت بین پاسخ صحیح و ارائه شده، با استفاده از روش نزول گرادیان [۲۳] به روز می شوند. پس از انجام فرآیند، شبکه های عصبی مصنوعی آموزش دیده تابعی را ارائه می دهد که در آن فضای برداری ورودی را می توان به تعداد معینی از کلاس ها تقسیم کرد. به طور کلی، تغییرات کوچک در مجموعه داده آموزشی باعث تغییرات جزئی در الگوریتم طبقه بندی می شود. اگرچه جمع‌آوری مجموعه داده‌های آموزشی کامل و جامع در برنامه‌های کاربردی واقعی غیرممکن است، یک طبقه‌بندی کننده تقریباً دقیق را با یک مجموعه داده آموزشی مناسب می‌توان انتظار داشت.

۴.۱ منطق وجهی

تعریف منطق وجهی به مفاهیم *نیاز*، *امکان* و *اقتضا* مرتبط است. منطق وجهی را برای ارزیابی "آنچه خواهد بود"، "آنچه بوده است"، "چه چیزی ممکن است"، "آنچه باور می‌شود" و دیگر مفاهیم نزدیک استفاده می کنند. این ویژگی‌های منطق وجهی، آن را برای بیان سناریوهای پیچیده‌تر مانند پردازش زبان طبیعی (یعنی بیان زمان، عمل، تغییر، علیت، اطلاعات، دانش، باور، الزام و جواز) مناسب می‌سازد. به طور صوری، خانواده منطق وجهی دارای "قاعده ضرورت" و "اصول توزیع" است که **K** (از نام Saul Kripke) نامیده می‌شود.

- قاعده ضرورت: اگر A قضیه ای از K باشد، آنگاه $\Box A$ نیز درست است.
- اصول توزیع: $\Box(A \to B) \to (\Box A \to \Box B)$.

در اینجا، نماد $\to$ برای "اگر ... آنگاه" و " $\Box$ " برای "ضروری است که" استفاده می شود. قاعده *ضرورت* نشان می‌دهد که "هر قضیه‌ای در این منطق ضروری است" و اصول *توزیع* می‌گوید که "اگر لازم است که A آنگاه B ، پس اگر لزوماً A ، سپس لزوماً B ". عملگر $\Diamond$ برای بیان "این امکان وجود دارد" بکار می رود. می توان آن را با فرض $\Diamond A = \neg\Box\neg A$ تعریف کرد که در آن $\neg$ برای "نقیض" استفاده می شود.

- $\Box A \to A$ (M)،
- $\Box A \to \Box\Box A$:(۴)،
- $\Diamond A \to \Box \Diamond A$:(۵)،
- $A \to \Box \Diamond A$ (B).

ایده منطق وجهی، ما را به تعریف وجوه بیشتر و ایجاد خانواده ای از منطق وجهی سوق می دهد. برای مثال، عملگرهای وجهی "ضروری" K و G به ترتیب منجر به تعریف "منطق شناختی" و "منطق زمانی" می‌شوند.

[۲۳] Gradient decent

نحو [۲۴] منطق وجهی به شرح زیر است:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi.$$

در اینجا، معنای مورد نظر فرمول $\Box\varphi$ " φ ضروری است" می باشد. برای تعریف معناشناسی [۲۵] زبان منطق وجهی مدل کریپکی [۲۶] را تعریف می‌کنیم، که مدلی برای منطق وجهی (معناشناسی رسمی برای سیستم‌های منطق غیرکلاسیک) است و برای استدلال در مورد روابط وجهی بکار می رود. ابتدا مدل کریپکی را برای سیستم تک عاملی تعریف می کنیم.

W را یک مجموعه از دنیا ها و R یک رابطه باینری (دسترسی) برای W در نظر بگیرید. یک فریم [۲۷] کریپکی یک جفت $\langle W, R\rangle$ است.

تعریف ۱.۴.۱ (مدل کریپکی)**.** فرض کنید $\langle W, R\rangle$ یک فریم کریپکی باشد و $Prop$ مجموعه‌ای از اتمی ها باشد. یک مدل کریپکی $\mathcal{M}$ یک سه تایی (W, R, V) است، که در آن تابع ارزیابی $V : W \longrightarrow ۲^{Prop}$ هر جهان را به مجموعه گزاره‌هایی که صادق هستند نگاشت می‌کند. در این جهان فرمول φ در مدل $\mathcal{M} = (W, R, V)$ زمانی معتبر است که φ در همه جهان های $w \in W$ صادق باشد.

- $\mathcal{M}, w \models p$ اگر و فقط اگر $p \in V(w)$،
- $\mathcal{M}, w \models \neg\phi$ اگر و فقط اگر $\mathcal{M}, w \not\models \varphi$،
- $\mathcal{M}, w \models \phi \wedge \psi$ اگر و فقط اگر $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$،
- $\mathcal{M}, w \models \Box\phi$ اگر و فقط اگر $\forall v \in R_i(w), \mathcal{M}, v \models \phi$.

۱.۴.۱ منطق شناختی

منطق شناختی از خانواده منطق وجهی است که معرفت در آن تفسیر می شود. عملگر مودال "ضروری" در منطق شناختی K است، که در آن $K_a\varphi$ به صورت "عامل a می داند φ" خوانده می شود. در این زمینه، فرمول $K_a\varphi$ بیان می کند که φ در تمام دنیاهایی که عامل a از نظر معرفتی مرتبط با اطلاعات فعلی خود می داند صادق است. به عبارت دیگر، زمانی که یک عامل یک فرمول را می داند، هیچ سناریویی در دیدگاه عامل وجود ندارد که در آن، فرمول به عنوان نادرست تفسیر شود. به این معنی که با دانشی که عامل جمع‌آوری می‌کند، فرمول را می‌توان درست تفسیر کرد و عامل می‌تواند آن را تضمین کند. فرمول $\neg K_a\neg\varphi$ را می توان به صورت "عامل a $\neg\varphi$ را نمی داند" تعبیر کرد. این نشان می دهد که عامل نمی تواند نادرستی فرمول را استدلال کند. یعنی سناریو ای وجود دارد که فرمول در آن امکان پذیر است. در ادامه، تعریف صوری و نحو منطق شناختی ارائه شده است.

---

[۲۴] Syntax
[۲۵] Semantic
[۲۶] Kripke
[۲۷] Frame

تعریف ۲.۴.۱ (زبان منطق شناختی)**.** $Ag = \{۱, \ldots, n\}$ را مجموعه محدودی از عوامل قرار دهید. نحو منطق معرفتی به شرح زیر خواهد بود:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi.$$

در این تعریف، p یک فرمول اتمی است. معنای $\neg\varphi$ نقیض فرمول φ است و $\varphi \wedge \psi$ معادل φ "و" ψ است و $K_i\varphi$ "عامل i ام φ را می داند" است. مدل کریپکی یک مدل منطق شناختی یک سه تایی $\mathcal{M} = (W, R_۱, \ldots, R_n, V)$ است که W مجموعه ای از دنیاها است، $R_i \subseteq W \times W$ یک رابطه هم‌ارزی بین دنیاها برای هر عامل است، و $V : W \longrightarrow ۲^{Prop}$ تابع ارزیابی است که تعیین می‌کند "کدام فرمول اتمی در آن جهان صادق است". معناشناسی فرمول های منطق شناختی به شرح زیر است:

- $\mathcal{M}, w \models p$ اگر و فقط اگر $p \in V(w)$،
- $\mathcal{M}, w \models \neg\phi$ اگر و فقط اگر $\mathcal{M}, w \not\models \varphi$،
- $\mathcal{M}, w \models \phi \wedge \psi$ اگر و فقط اگر $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$،
- $\mathcal{M}, w \models K_i\phi$ اگر و فقط اگر $\forall v \in R_i(w), \mathcal{M}, v \models \phi$.

به عبارت دیگر، برای یک مدل $\mathcal{M} = (W, R_۱, \ldots, R_n, V)$ ، تفسیر فرمول شناختی $K_i\varphi$ در یک جهان w با بررسی ارضی فرمول در تمام دنیاهای مرتبط با w بوسیله R_i بررسی می شود.

یکی از بخش‌های هیجان‌انگیز استفاده از سیستم چند عاملی، تجمیع دانش جمع‌آوری‌شده است. برای این منظور، نمادی از *دانش توزیع‌شده* برای جمع‌آوری دانش توزیع شده در یک گروه از عوامل ایجاد می‌شود. عملگر دانش توزیع‌شده را با $D_A, A \subseteq Ag$ نشان می دهیم. Ag را گروهی از عوامل در نظر بگیرید، معنای فرمول $D_A\varphi$ " φ یک دانش توزیع شده در گروه A از عوامل است" خواهد بود. نماد دیگری که برای گروهی از عوامل پیشنهاد می شود این است که آیا همه یک فرمول را می دانند یا نه، و با E_A نشان داده می شود. معنای فرمول $E_A\varphi$ "همه در گروه A ، φ را می دانند" خواهد بود.

- $\mathcal{M}, w \models D_A\phi$ اگر و فقط اگر $\forall v \in R_A(w), \mathcal{M}, v \models \phi$، وقتی $R_A := \bigcap_{i \in A} R_i$.
- $\mathcal{M}, w \models E_A\phi$ اگر و فقط اگر $\forall v \in R_A(w), \mathcal{M}, v \models \phi$، وقتی $R_A := \bigcup_{i \in A} R_i$.

۱.۱.۴.۱ منطق اعلان عمومی

منطق اعلان عمومی [۲۸] با هدف مطالعه دانش و ارتباطات بین عامل ها است. در اینجا، گسترش منطق شناختی، یعنی منطق اعلان عمومی را نشان خواهیم داد. با مثال "معمای کودکان گل آلود" شروع می کنیم:

پازل کودکان گل آلود پدر سه فرزندش را که در گل و لای بازی می کردند صدا می زند. وقتی نزد پدر می آیند، پیشانی برخی از آنها گِل آلود بود. پدر از آنها می خواهد که دور یک میز بنشینند تا پیشانی

[۲۸] Public announcement logic (PAL)

دیگران را ببینند. هیچ کس نمی تواند پیشانی خودش را ببیند. پدر می گوید: "حداقل یکی از شما پیشانی گل آلود دارد. اگر کسی می داند که آیا پیشانی او کثیف است یا خیر، برخیزد." هیچ کودکی بلند نمی شود. پدر این نقل قول را تکرار می کند. برخی از آنها بر می خیزند. پدر تکرار می کند و همه آنها می ایستند. چند کودک پیشانی گل آلود داشتند؟

اینجاست که می توان از منطق اعلان عمومی برای یافتن پاسخ استفاده کرد.

تعریف ۳.۴.۱ (زبان منطق اعلان عمومی). $Ag = \{۱, \ldots, n\}$ را مجموعه محدودی از عوامل در نظر بگیرید. نحو زبان منطق اعلان عمومی به شرح زیر خواهد بود:

$$\phi ::= p \mid \neg\phi \mid (\phi \wedge \phi) \mid K_i\phi \mid D_A\phi \mid [\phi]\phi.$$

فرمول $[\phi]\phi$ به منطق شناختی تعریف شده اضافه شده است. فرمول $[\psi]\phi$ به صورت "پس از اعلان صحیح ψ ، ϕ ارضی خواهد شد" خوانده می شود.

- $\mathcal{M}, w \models [\psi]\phi$ اگر و فقط اگر $\mathcal{M}, w \models \psi$ برقراری $\mathcal{M}^\psi, w \models \phi$ را نتیجه دهد.

وقتی $\mathcal{M}^\psi := (W^\psi, R_۱^\psi, \ldots, R_n^\psi, V^\psi)$ با جهان های $W^\psi := \{w \in W; \mathcal{M}, w \models \psi\}$ و روابط $R_j^\psi := R_j \cap (W^\psi \times W^\psi)$ و برای همه $j \in \{۱, \ldots, n\}$ و $V^\psi(w) := V(w)$ all for $w \in W^\psi$ است.

اپراتور اعلان، مدل کریپکی را یک مدل پویا می کند. به طور دقیق تر، برقراری $\mathcal{M}, w \vDash [\psi]\phi$ معادل ارضی $\mathcal{M}^\psi, w \vDash \phi$ است. در اینجا، پویایی مدل را می توان مشاهده کرد.

حالا می توانیم جواب بچه های گل آلود را پیدا کنیم. $Ag = \{۱, ۲, ۳\}$ را مجموعه ای از عوامل در نظر بگیرید، که در اینجا سه فرزند هستند. p_i نشان دهنده گل آلوده بودن i امین فرزند باشد. مدل دارای هشت جهان $W = \{w_{...}, w_{۱..}, w_{.۱.}, w_{..۱}, w_{۱۱.}, w_{۱.۱}, w_{.۱۱}, w_{۱۱۱}\}$ خواهد بود، که اندازه مجموعه توانی [۲۹] $Prop = \{p_۱, p_۲, p_۳\}$ است. در مدل اولیه، هیچ کودکی اطلاعاتی در مورد پیشانی خود ندارد اما از دیگران اطلاع دارد. بنابراین می توانیم روابط را به صورت زیر تعریف کنیم:

$$R_۱ = \{(w_{.ij}, w_{.ij}), (w_{.ij}, w_{۱ij}), (w_{۱ij}, w_{.ij}), (w_{۱ij}, w_{۱ij}) \mid i, j \in \{۰, ۱\}\},$$

$$R_۲ = \{(w_{i.j}, w_{i.j}), (w_{i.j}, w_{i۱j}), (w_{i۱j}, w_{i.j}), (w_{i۱j}, w_{i۱j}) \mid i, j \in \{۰, ۱\}\},$$

$$R_۳ = \{(w_{ij.}, w_{ij.}), (w_{ij.}, w_{ij۱}), (w_{ij۱}, w_{ij.}), (w_{ij۱}, w_{ij۱}) \mid i, j \in \{۰, ۱\}\}.$$

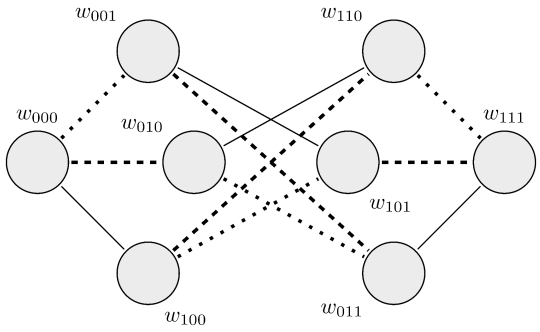
برای هر جهانی، تابع ارزیابی به صورت زیر تعریف می شود:

$$V(w_{i۱i۲i۳}) = \{p_i \mid i_j = ۱, j \in \{۱, ۲, ۳\}\}.$$

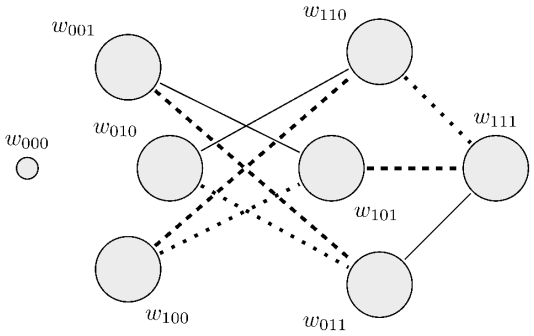
مدل اولیه در شکل ۱.۲ نشان داده شده است.

وقتی پدر می‌گوید: "حداقل یکی از شما پیشانی گلی دارد"، به این معنی است که جهان $w_{...}$ ممکن نیست (شکل ۲.۲ را ببینید). بعد از آن، وقتی می گوید: "اگر کسی می‌داند پیشانی‌اش کثیف است، برخیز" و هیچ‌کس بلند نمی‌شود، یعنی $w_{۱..}$ ، $w_{.۱.}$ و $w_{..۱}$ جهان ممکن نیستند زیرا اگر چنین بود، با

[۲۹] Power set



شکل ۱.۱: مدل کریپکی اولیه مسأله کودکان گل آلود.



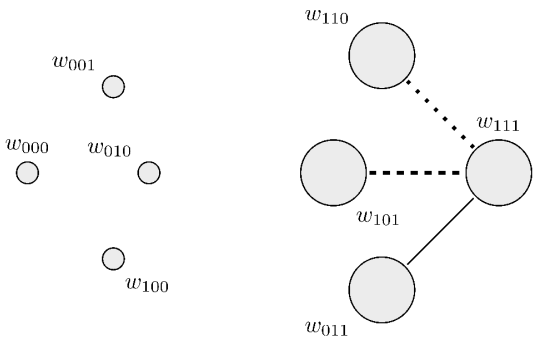
شکل ۲.۱: پس از اولین اعلان، کودکان می دانند که $w_{\dots}$ غیرممکن است.

حذف $w_{\dots}$، فرزندان شماره ۲،۱ و ۳ می‌توانستند تشخیص دهند که $w_{۱\dots}$ ، $w_{\cdot۱\cdot}$ و $w_{\dots۱}$ (به ترتیب) جهان های واقع هستند. وقتی پدر تکرار می کند، برخی از آنها بلند می شوند. این بدان معناست که جهان $w_{۱۱۱}$ غیرممکن است. اگر اینطور بود، هیچ کس نمی توانست مطمئن شود و بایستد، زیرا اگر دنیای واقعی $w_{۱۱۱}$ بود، فرزند اول نمی توانست آن را از $w_{\cdot۱۱}$ تشخیص دهد، فرزند دوم نمی توانست آن را با $w_{۱\cdot۱}$ تشخیص دهد و فرزند سوم نتوانست آن را با $w_{۱۱\cdot}$ تشخیص دهد. بنابراین، پاسخ به سؤال "چند کودک پیشانی گل آلود داشتند؟"، "دو" خواهد بود.

۲.۴.۱ منطق زمانی

منطق زمانی منطقی برای استدلال در مورد زمان است. مدل با $\mathcal{T} = \langle T, \prec \rangle$ نشان داده می‌شود، که T مجموعه‌ای ناتهی از لحظه‌های زمانی و $\prec$ یک رابطه دوتایی است. رابطه تقدم زمانی معمولاً یک نظم جزئی دقیق [۳۰] ، یک رابطه غیر انعکاسی [۳۱] ، متعدی (ترایایی) [۳۲] و نامتقارن [۳۳] است (گاهی اوقات، انعکاسی [۳۴] و پادتقارنی [۳۵] فرض می شوند). برخی از ویژگی های ممکن منطق زمانی $\mathcal{T} = \langle T, \prec \rangle$ ، که

[۳۰] Strict partial ordering
[۳۱] Irreflexive
[۳۲] Transitive
[۳۳] Asymmetric
[۳۴] Reflexive
[۳۵] Antisymmetry



شکل ۳.۱: بعد از اینکه به کودکان اطلاع داده شد و هیچکس نمی‌ایستد، آنها متوجه می شوند که $w_{۱..}$ ، $w_{.۱.}$ و $w_{۱..}$ غیرممکن است.

با نحو مرتبه اول نشان داده شده در زیر بیان می‌شوند:

- بازتابی : $\forall x(x \prec x)$؛
- غیر انعکاسی : $\forall x\neg(x \prec x)$؛
- ترایایی : $\forall x\forall y\forall z(x \prec y \wedge y \prec z \rightarrow x \prec z)$؛
- نامتقارن : $\forall x\forall y\neg(x \prec y \wedge y \prec x)$؛
- پاد تقارنی: $\forall x\forall y(x \prec y \wedge y \prec x \rightarrow x = y)$؛
- خطی [۳۶] (تریکوتومی [۳۷] ، اتصال [۳۸]): $\forall x\forall y(x = y \vee x \prec y \vee y \prec x)$؛
- رو به جلو خطی [۳۹] : $\forall x\forall y\forall z(z \prec x \wedge z \prec y \rightarrow (x = y \vee x \prec y \vee y \prec x))$؛
- رو به عقب خطی [۴۰] : $\forall x\forall y\forall z(x \prec z \wedge y \prec z \rightarrow (x = y \vee x \prec y \vee y \prec x))$؛
- آغاز [۴۱] : $\exists x\neg\exists y(y \prec x)$؛
- پایان [۴۲] : $\exists x\neg\exists y(x \prec y)$؛
- بی آغاز [۴۳] : $\forall x\exists y(y \prec x)$؛
- بی پایان [۴۴] (نامحدود [۴۵]): $\forall x\exists y(x \prec y)$؛

[۳۶] Linearity
[۳۷] Trichotomy
[۳۸] Connectedness
[۳۹] Forward-linearity
[۴۰] Backward-linearity
[۴۱] Beginning
[۴۲] end
[۴۳] no start
[۴۴] No end
[۴۵] Unboundedness

تا به امروز، کاربردهای اصلی منطق زمانی شامل فرمالیسم و استدلال مسائل فلسفی در مورد زمان، فرمالیسم بیان زبان طبیعی، کدگذاری [۴۶] دانش زمان بندی شده در هوش مصنوعی، و ابزاری برای مشخص کردن، تجزیه و تحلیل و درست یابی برنامه های رایانه ای بوده است.

اینها همه حول محور این مشکل "چه چیزی در آینده ضروری یا ممکن خواهد بود؟" هستند. منطق زمانی می تواند خطی یا غیرخطی (درخت مانند) باشد. در منطق زمانی خطی، جریان زمان به صورت یک مسیر نشان داده می‌شود، اما در منطق زمانی غیرخطی (همچنین با عنوان انشعاب به آینده‌های ممکن چندگانه شناخته می‌شود)، گذشته ثابت است و آینده‌های ممکن متعدد هستند. زبان اصلی *منطق زمان قبلی* [۴۷] ، بسط منطق گزاره ای با دو عملگر وجهی است:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid P\varphi \mid F\varphi$$

. معنای $P\varphi$ به صورت "گاهی اوقات اینطور بوده است که φ " و $F\varphi$ به صورت "گاهی اوقات اینطور خواهد بود که φ " خواهند بود. در اینجا عملگرهای $H\varphi = \neg P\neg\varphi$، $G\varphi = \neg F\neg\varphi$ ، $A\varphi = H\varphi \wedge \varphi \wedge G\varphi$ و $E\varphi = P\varphi \vee \varphi \vee F\varphi$ را می توان بصورت زیر تعریف کرد:

- $H\varphi$: " همیشه اینطور بوده است که φ "،
- $G\varphi$: " همیشه اینطور خواهد بود که φ "،
- $A\varphi$: "همیشه φ "،
- $E\varphi$: "گاهی اوقات φ ".

یک مدل برای منطق زمانی، سه تایی $\mathcal{M} = \langle T, \prec, V\rangle$ است که $\mathcal{T} = \langle T, \prec\rangle$ یک فریم است و V یک تابع ارزش گذاری است. تصدیق فرمول φ با $\mathcal{M}, w \vDash \varphi$ نشان داده می شود و به صورت استقرایی به صورت زیر تعریف می شود:

- $\mathcal{M}, w \vDash p$ iff $w \in V(p)$;
- $\mathcal{M}, w \vDash \neg\varphi$ iff $\mathcal{M}, w \nvDash \varphi$;
- $\mathcal{M}, w \vDash \varphi \wedge \psi$ iff $\mathcal{M}, w \vDash \varphi \wedge \mathcal{M}, w \vDash \psi$;
- $\mathcal{M}, w \vDash P\varphi$ iff $\mathcal{M}, w' \vDash \varphi$ for some time instant w' such that $w' \prec w$;
- $\mathcal{M}, w \vDash F\varphi$ iff $\mathcal{M}, w' \vDash \varphi$ for some time instant w' such that $w \prec w'$;

در بخش بعدی به بحث منطق زمانی خطی [۴۸] می پردازیم که به طور گسترده در علوم کامپیوتر کاربرد دارد.

[۴۶] Encoding
[۴۷] Prior's Tense logic
[۴۸] Linear temporal logic

۵.۱ منطق زمانی خطی

یک کلاس بسیار طبیعی از منطق زمانی، منطق زمانی خطی است. این بخش در مورد منطق زمانی با عملگرهای "بعدی" و "تا" بحث می‌کنیم. پس از افزودن این عملگرها، منطق زمانی خطی را می توان توسعه داد. با افزودن عملگرهای *neXt* و *Until* و با توجه به اینکه هیچ عملگر قبلی در منطق زمانی خطی وجود ندارد، زبان منطق زمانی خطی به این صورت خواهد بود:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid X\varphi \mid \varphi U \varphi.$$

در اینجا، عملگرهای X ("بعدی") و U ("تا") با معنای زیر می‌توانند اضافه شوند.

$\mathcal{M}, w \vDash X\varphi$ iff $\mathcal{M}, s(w) \vDash \varphi$,
$\mathcal{M}, w \vDash \varphi U \psi$ iff $\mathcal{M}, s \vDash \psi$ for some s such that $w \prec s$ and $\mathcal{M}, u \vDash \varphi$ for every u such that $w \prec u \prec s$.

در اینجا، $X\varphi$ زمانی درست است که φ در جهان بلافاصله بعدی درست باشد. معنای $X\varphi$ "در جهان بلافاصله بعدی φ خواهد بود" است. عملگر X اصل K ($X(\varphi \to \psi) \to (X\varphi \to X\psi)$) و اصل کارکرد [۴۹] ($X\neg\varphi \to \neg X\varphi$) را برآورده می‌کند. عملگر زمانی دوتایی $\varphi U \psi$ زمانی درست است که φ در جهان های بعدی *تا زمانی که* ψ درست شود، صادق باشد. معنای $\varphi U \psi$ " φ تا زمانی که ψ درست شود، صادق باشد" است.

[۴۹] Functionality

فصل ۲

درست یابی طبقه بندی کننده ها

۱.۲ تایید طبقه بندی کننده ها

امروزه، در دسترس بودن داده ها باعث می شود که الگوریتم های طبقه بندی بر پایه یادگیری به سرعت مورد توجه قرار گیرند و به بلوغ برسند. روش های طبقه بندی زیادی وجود دارد که اکثر آنها بر اساس رویکردهای آماری هستند. اگرچه این الگوریتم ها سریع و دقیق هستند، اما خطاهایی هم دارند که تفسیر آنها سخت است. استفاده گسترده از طبقه‌بندی‌کننده‌ها در کاربردهای واقعی و حیاتی، خطای آن‌ها را بحرانی و گاهی اوقات خطرناک می‌کند. بنابراین، تفسیر و درست یابی طبقه‌بندی‌کننده‌ها مورد توجه واقع شدند. متأسفانه، روش های درست یابی سنتی به دلیل دامنه ورودی و اندازه مدل اکثر طبقه بندی کننده ها، کارایی ندارند. این مسائل نیازمند روش های درست یابی و تفسیر است که در زمان معقول اجرا شوند.

برای انجام این کار، استفاده از ویژگی های طبقه بندی کننده های آماری می تواند به بهبود عملکرد درست یابی کننده ها کمک کند. معمولاً طبقه‌بندی‌کننده‌های آماری سعی می‌کنند یک فضای برداری را به فضاها افراض کنند و به هر یک از آنها یک برچسب اختصاص دهند. از این رو، طبقه بندی کننده های آماری پاسخ هایی را با احتمال می دهند. این بدان معناست که احتمالات در نزدیکی مرزهای فضا های افراض شده کمتر است و زمانی که ورودی نزدیک مرزها قرار گیرد، خطاهای بیشتری رخ می دهد. برای مثال، فرض کنید تصویر در نزدیکی مرزهای تقسیم در یک طبقه‌بندی کننده تصویر قرار دارد. در این حالت، برچسب خروجی ممکن است با اضافه کردن مقدار کمی نویز از یک دسته به دسته دیگر منتقل شود. از نظر معرفتی، نویز کلاس نماینده تصویر را تغییر نمی‌دهد، اما از نظر ریاضی، مقدار برداری تصویر را تغییر می‌دهد. بنابراین، هنگامی که تصویر در نزدیکی مرزهای تقسیم قرار می گیرد، تغییرات کوچک می تواند فضای افراض شده را تغییر دهد، که می تواند برچسب را تغییر دهد. این سناریو خطاهای ناشی از نویزها (نمونه های متخاصم [۱]) را به تصویر می کشد. سناریوهایی غیر از نویزها می توانند باعث چنین خطاهایی شوند. به عنوان مثال، فرض کنید از یک شی عکس می گیریم، زاویه دوربین را تغییر می دهیم و مجدداً یک عکس دیگر از همان شی می گیریم. این دو تصویر از نظر معرفتی حاوی یک شی هستند، اما ممکن است در پارتیشن‌های مختلف با برچسب‌های متفاوت قرار بگیرند. بسیاری از دستکاری های

[۱] Adversarial examples

[۲] دیگر می تواند باعث چنین خطاهایی شود. با در نظر گرفتن این موارد، هوانگ و همکاران از درست یابی نقطه‌ای [۳] برای طبقه‌بندی‌کننده‌های شبکه عصبی عمیق استفاده می‌کند. برای انجام این کار، به جای درست یابی ویژگی های یک مدل، آنها ویژگی های یک نقطه ورودی برای مدل را درست یابی کردند. به عنوان یک مجموعه ویژگی، آنها مجموعه هایی از همسایگی [۴] و دستکاری (برای هر لایه از مدل شبکه عصبی عمیق) ورودی را به مدل وارد می کردند. اگر مدل برچسب‌های یکسانی را به همه ورودی‌ها اختصاص می داد، ورودی به‌عنوان درست یابی شده در نظر گرفته می‌شود. هوانگ و همکاران نشان داد که با این روش، خطاها به میزان قابل توجهی کاهش می یابد.

۲.۲ منطق شناختی و طبقه بندی کننده ها

در اکثر مسائل طبقه بندی، هیچ روش غالبی وجود ندارد که بهترین نتیجه را ارائه دهد. برخی از طبقه بندی کننده ها در موارد خاص پیشنهادات بهتری ارائه می دهند. این بخش نشان می دهد که چگونه تفسیر پاسخ ها با استفاده از چند طبقه بندی کننده، خطاها را کاهش می دهد. در ابتدا، با استفاده از روش توسعه یافته توسط هوانگ و همکاران، یک طبقه بندی کننده مجموعه ای از پاسخ ها را با وارد کردن مجموعه ویژگی های ورودی ارائه می دهد. اگر اندازه مجموعه پاسخ برابر با یک باشد، تمام عناصر مجموعه ویژگی در یک کلاس طبقه بندی می شوند. بنابراین، همه در مورد پاسخ توافق دارند و آن را درست یابی کردند. فرض کنید اندازه مجموعه بزرگتر از یک باشد، پاسخ درست یابی نمی شود، اما پاسخ درست یابی شده در بین مجموعه ارائه شده قرار می گیرد. با اعمال چند طبقه بندی کننده، مجموعه های متعددی از پاسخ ها ایجاد می شود. فرض کنید، در مدل کریپکی $\mathcal{M} = (W, R_1, \ldots, R_n, V)$ ، هر جهان (حالت) w_i یک برچسب خروجی ممکن را نشان می‌دهد $V(w_i) = c_i$. تفسیر رابطه R_i برای هر طبقه بندی کننده در منطق شناختی این است که طبقه‌بندی‌کننده i نمی‌تواند بین دو جهان‌های تمایز قائل شود. به عنوان مثال، جهان‌های ممکن برای امین-i طبقه‌بندی‌کننده $W = \{w_i \mid V(w_i) \in [X]_G\}$ خواهد بود. با محاسبه اشتراک جهان های ممکن که هر طبقه بندی کننده ارائه می دهد، $W_D = \bigcap_i W_i$ بیانگر دانش توزیع شده این طبقه بندی کننده ها خواهد بود. حال، اگر اندازه W_D برابر با یک باشد، ورودی برای مجموعه طبقه‌بندی‌کننده‌ها با در نظر گرفتن ویژگی‌های از پیش تعریف‌شده درست یابی می‌شود. اینکه همه طبقه‌بندی‌کننده‌ها با پاسخ ارائه‌شده موافقت می‌کنند، بدیهی است و همه ویژگی‌های از پیش تعریف‌شده در آنها را برآورده می‌شود. دهکردی و همکاران نشان داد که استفاده از طبقه‌بندی‌کننده‌های متعدد به طور قابل توجهی خطای مقایسه روش توسعه‌یافته توسط هوانگ و همکاران را کاهش می‌دهد. . این نوع تفسیر دانش را می‌توان در سیستم‌های بحرانی به کار برد، زیرا اگرچه زمان زیادی برای دریافت پاسخ صرف می شود، کاهش خطا در چنین مسائلی اهمیت بیشتری دارد.

در سناریوهای دیگری می توان از قدرت منطق شناختی استفاده کرد. همچنین می توان از مدل منطق شناختی برای جمع آوری اطلاعات از منابع مختلف استفاده کرد. به عنوان مثال، فرض کنید طبقه‌بندی‌کننده‌ای تصویر روز/شب و خفاش/کبوتر را شناسایی می‌کنند و می دانیم که خفاش‌ها در روز ظاهر نمی‌شوند. بنابراین می توان تفسیر کرد که وضعیت روز و خفاش با هم اتفاق نخواهد افتاد. در مدل شناختی، در صورت وقوع خطا، می توان متوجه شد که کدام طبقه بندی کننده معیوب بوده است و

[۲] Manipulation
[۳] Pointwise verification
[۴] Neighborhood

این ما را به ایجاد یک سیستم نظارت بر طبقه بندی کننده ها هدایت می کند. اگر عوامل سیستم طبقه بندی کننده مجموعه ای از برچسب ها را ارائه دهند، همه طبقه بندی کننده ها توافق کردند که پاسخ در میان مجموعه است. سیستم توسعه یافته می تواند به عنوان یک سیستم دستیار در این شرایط استفاده شود. به عنوان مثال، در تشخیص به کمک کامپیوتر [۵] ، یک کامپیوتر برای تولید خروجی به عنوان ابزار کمکی برای تشخیص پزشک استفاده می‌شود. سیستم توسعه یافته می تواند به عنوان یک دستیار برای کاهش تردیدهای تشخیص های بالینی استفاده شود.

۳.۲ منطق زمانی و طبقه بندی کننده ها

همانطور که منطق شناختی ابزاری قوی برای تفسیر اطلاعات داده‌های تکی [۶] است، منطق زمانی برای تجزیه و تحلیل جریان داده‌ها [۷] برای شناسایی اقدامات بکار می آید. به طور خاص، در یک جریان داده، می‌توان با استفاده از فرمول منطق زمانی خطی یک عمل را تعریف کرد (مثلاً $\varphi = \varphi_1 \wedge X\varphi_2 \wedge XX\varphi_3$ عملی را تعریف می‌کند که در آن φ_1 اتفاق افتاده است، سپس در مرحله بعدی φ_2 و در نهایت φ_3). برقراری چنین فرمولی به این معنی است که عمل (φ) اتفاق افتاده است. در اینجا، جریان داده به عنوان جریانی از داده های تکی فرض می شود. از این رو، ما یک مدل معرفتی برای درست یابی داده های تکی داشتیم. ترکیب منطق زمانی و معرفتی ما را به تعریف یک ویژگی برای ورودی جریان داده سوق می دهد. درست یابی نقطه ای را با استفاده از ویژگی های از پیش تعریف شده می توانیم اعمال کنیم. برای انجام این کار، یک سیستم انتقال متشکل از مدل‌های کریپکی هر فریم ایجاد می‌کنیم. هر فریم دارای یک مدل کریپکی با مجموعه ای از دنیاها W_i (با در نظر گرفتن ویژگی از پیش تعریف شده) است. ما تمام مدل های کریپکی فریم ها را به ترتیب قرار می دهیم. سپس تمام جهان های هر دو مدل متوالی را با روابط زمانی به هم وصل می کنیم. این کار یک گراف بخشی-k [۸] می سازد که در آن W_i به W_{i-1} و W_{i+1} متصل است. در سیستم انتقال، مسیرهای زمانی اجرایی منحصر به فرد $\Pi_i \mid W_i \mid$ ایجاد می شود. اینها همه سناریوهای ممکن از اشیاء هستند که در جریان داده طبقه بندی می شوند. اکنون، با بررسی ارضی فرمول های منطق زمانی خطی (که ویژگی هایی هستند که باید درست یابی شوند) در تمام مسیرها، می توانیم بررسی کنیم که آیا مدل برای ورودی خاص تأیید شده است یا خیر. در اینجا، ما همچنین می‌توانیم بررسی کنیم که آیا یک فرمول در یک مسیر اجرا امکان‌پذیر است یا خیر (در صورتی که برخی از مسیرها ویژگی‌ها را برآورده کنند). همانطور که در بخش ۲.۳ گفتیم، یک مسیر اجرایی ممکن است در سیستم‌های دستیار (به عنوان مثال، CAD) بکار گرفته شود.

نتیجه گیری

منطق وجهی به عنوان یک ابزار قوی برای درست یابی شناخته می شود. این تحقیق منطق وجهی را به عنوان درست یابی کننده طبقه بندی کننده ها و یک سیستم دستیار ارائه می کند. ابتدا، نشان دادیم که چگونه منطق شناختی را می توان برای درست یابی طبقه بندی کننده های داده های تکی به کار برد.

[۵] Computer aided diagnosis
[۶] Data frame
[۷] Data stream
[۸] k-partite

این کار با تعریف یک مدل کریپکی ساخته شده با استفاده از اطلاعات طبقه‌بندی‌کننده‌ها انجام شد. سپس، کاربرد منطق زمانی را برای تشخیص ورودی‌های جریان داده تعریف کردیم. در نهایت، ترکیبی از مدل‌های منطق شناختی و زمانی برای درست یابی ویژگی‌های ورودی‌های جریان داده ارائه دادیم. اینجا، ویژگی ها را می توان به شکل منطق زمانی خطی ایجاد کرد. این مدل‌ها مواقعی قابل استفاده هستند که هزینه طبقه‌بندی خطا بالا باشد. کاربرد مدل توسعه یافته به عنوان دستیار نیز شرح داده شد.