

Symbolic Knowledge Extraction and Injection: Theory and Methods

(last built on: 2025-07-02)

Giovanni Ciatto and Matteo Magnini

Dipartimento di Informatica — Scienza e Ingegneria (DISI), Sede di Cesena,
Alma Mater Studiorum—Università di Bologna

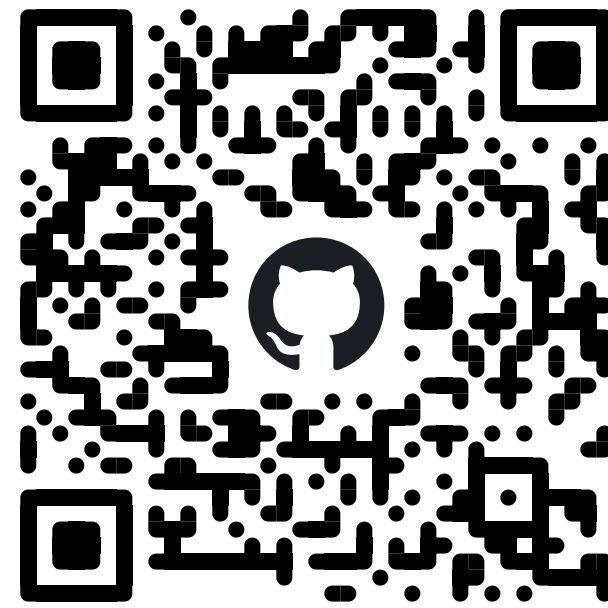
Mini-school @ WOA 2025, the 26th Workshop From Objects to Agents
2nd July 2025, Trento, Italy

This talk is partially supported by the “ENGINES — ENGINEering INtelligent Systems around intelligent agent technologies” project funded by the Italian MUR program “PRIN 2022” under grant number 20229ZXBZM.



Link to these slides

<https://MatteoMagnini.github.io/talk-2025-woa-nesy/>



 [printable version](#)

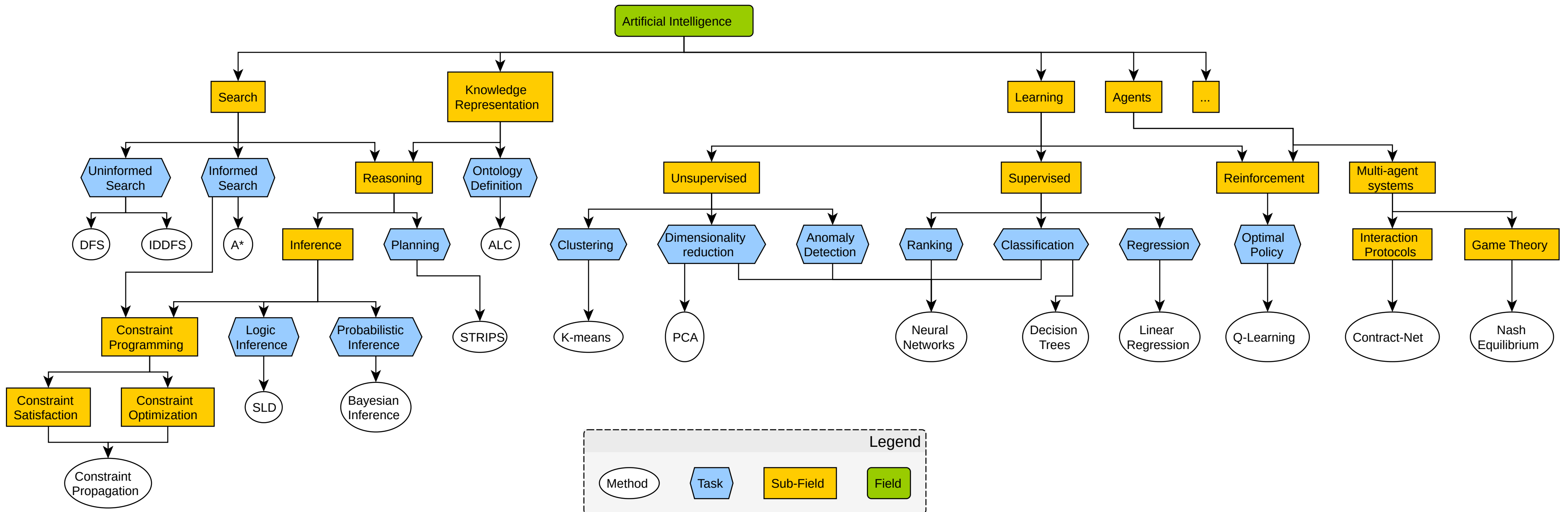


Background

Quick overview on symbolic vs. sub-symbolic AI



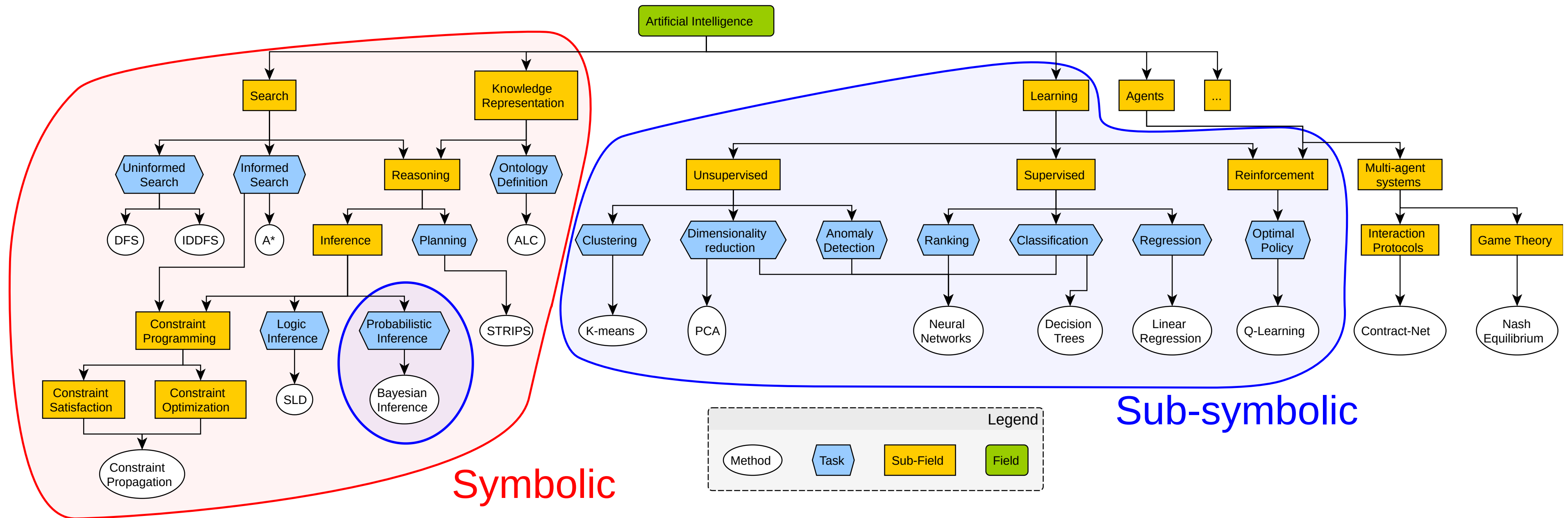
Overview on AI



- wide field of research, with many *sub-fields*
- each sub-field has its own relevant *tasks* (problems) ...
- ... and each task comes with many useful *methods* (algorithms)

Symbolic vs. Sub-symbolic AI

Two broad categories of AI approaches:



Why the wording “Symbolic” vs. “Sub-symbolic”? (pt. 1)

Local vs. Distributed Representations

Localist Representation						
	Bear	Tiger	Eagle	Fish	Turtle	Frog
Bear	●					
Tiger		●				
Eagle			●			
Gold Fish				●		
Turtle					●	
Frog						●

Distributed Representation						
	Walk	Swim	Fly	Egg	Claw	Wild
Bear	●	●	○	○	●	●
Tiger	●	●	○	○	●	●
Eagle	○	○	●	●	●	○
Gold Fish	○	●	○	●	○	○
Turtle	●	●	○	●	○	○
Frog	●	●	○	●	●	○

- **Local** \approx “symbolic”: each symbol has a clear, distinct meaning
 - e.g. **"bear"** is a symbol denoting a crisp category (either the animal is a bear or not)
- **Distributed** \approx “non-symbolic”: symbols do not have a clear meaning per se, but the whole representation does
 - e.g. **"swim"** is fuzzy capability: one animal may be (un)able to swim to some extent

Let’s say we need to represent N classes, how many columns would the tables have?

Why the wording “Symbolic” vs. “Sub-symbolic”? (pt. 2)

What is a “symbol” after all? Aren’t numbers symbols too?

According to [Tim van Gelder in 1990](#):

Symbolic representations of knowledge

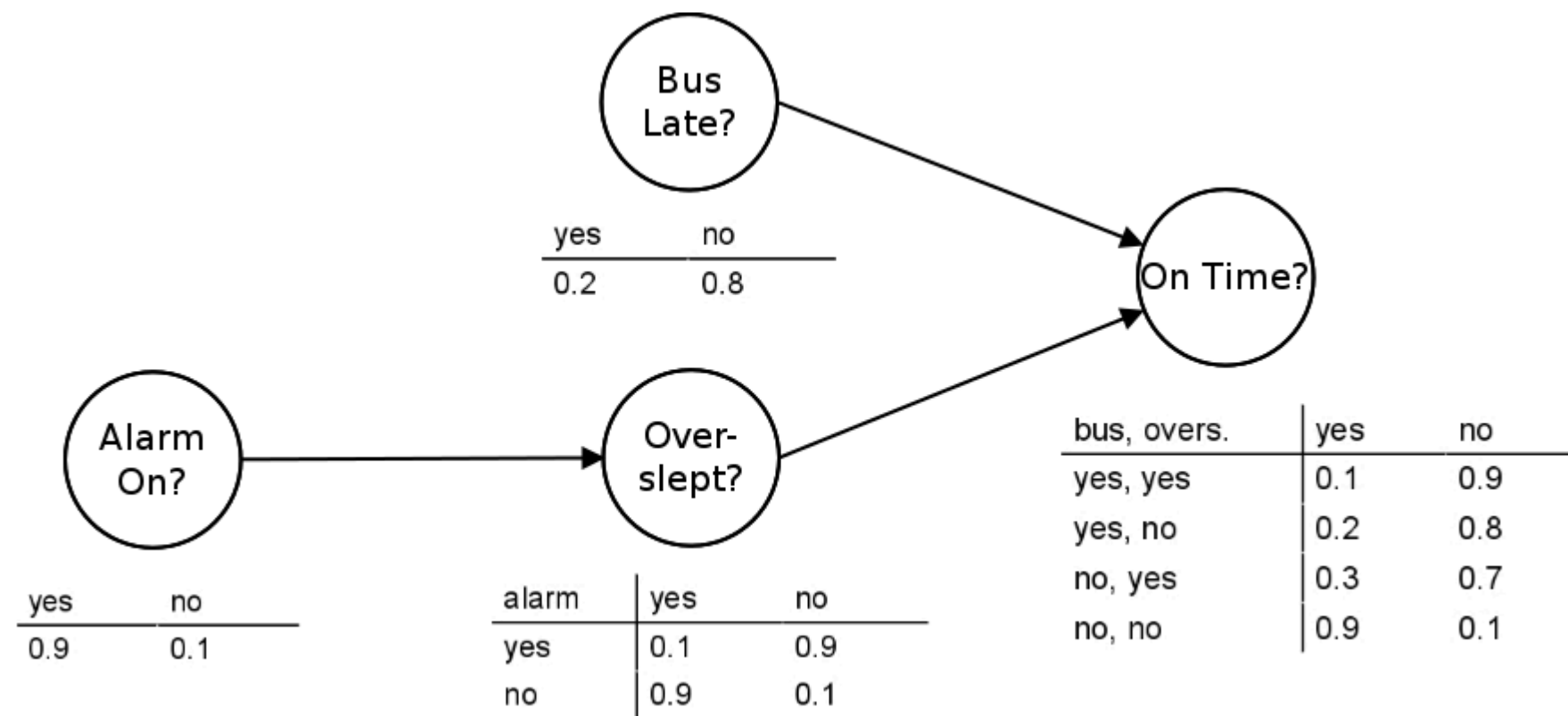
- involve a *set of symbols*
- which can be *combined* (e.g., concatenated) in (possibly) infinitely many ways,
- following precise *syntactical rules*,
- where both elementary symbols and any admissible combination of them can be *assigned with meaning*



Why “*Sub*-symbolic” instead of “Non-symbolic” or just “Numerical”?

- There exist approaches where symbols are combined with numbers, e.g.:

- **Probabilistic logic programming**: where logic statements are combined with probabilities
- **Fuzzy logic**: where logic statements are combined with degrees of truth
- **Bayesian networks**: a.k.a. graphical models, where nodes are symbols and edges are conditional dependencies with probabilities, e.g.



- These approaches are *not purely symbolic*, but they are *not purely numeric* either, so we call the overall category “**sub-symbolic**”

Examples of Symbolic AI (pt. 1)

- **Logic programming**: SLD resolution (e.g., Prolog)
- **Knowledge representation**: Semantic Web (e.g., OWL), Description Logics (e.g., ALC)
- **Automated reasoning**: Theorem proving, Model checking
- **Planning**: STRIPS, PDDL



Examples of Symbolic AI (pt. 2)

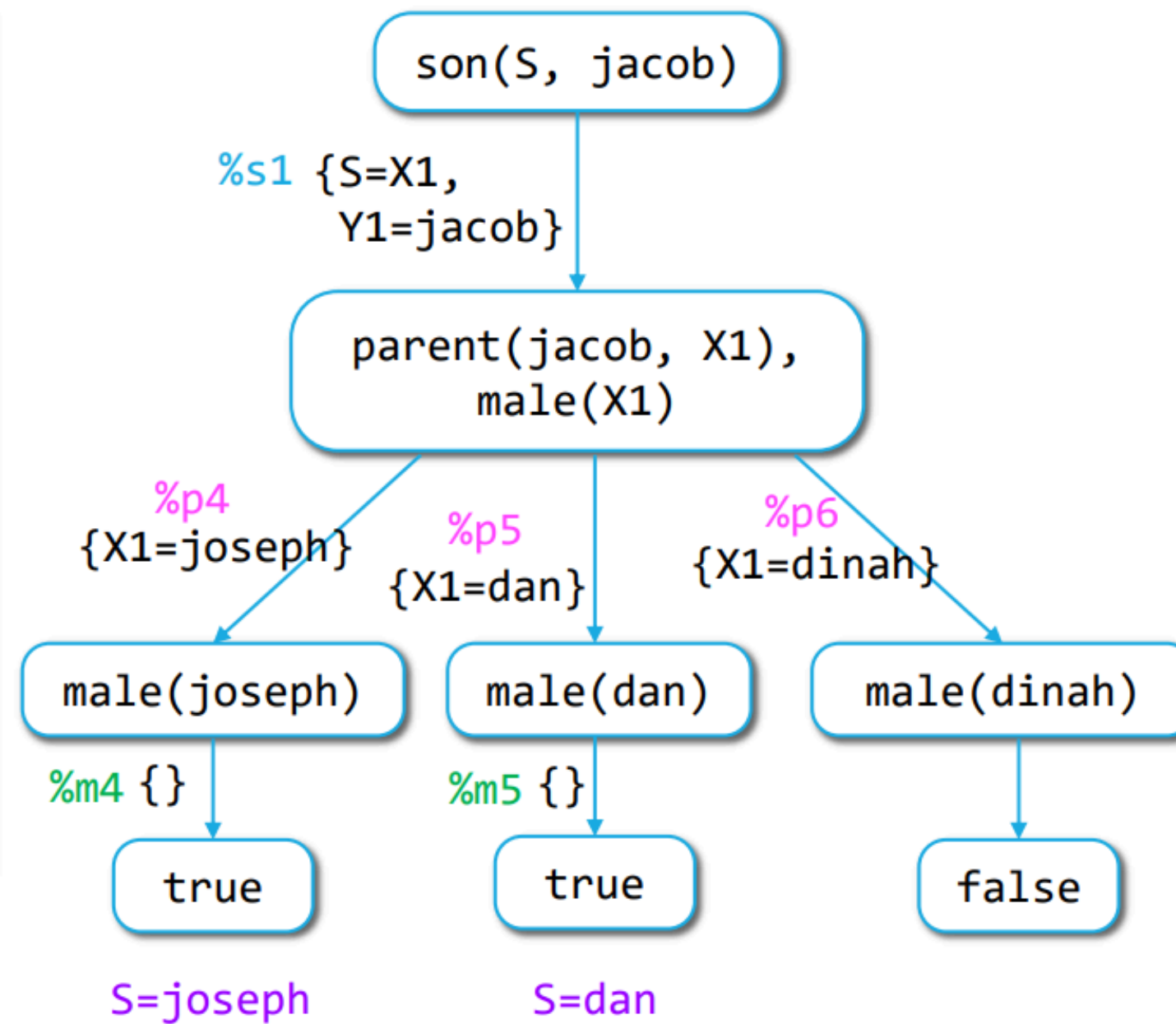
Logic programming with SLD resolution

```
parent(abraham, isaac). %p1
parent(isaac, jacob). %p2
parent(sarah, isaac). %p3
parent(jacob, joseph). %p4
parent(jacob, dan). %p5
parent(jacob, dinah). %p6

male(abraham). %m1
male(isaac). %m2
male(jacob). %m3
male(joseph). %m4
male(dan). %m5

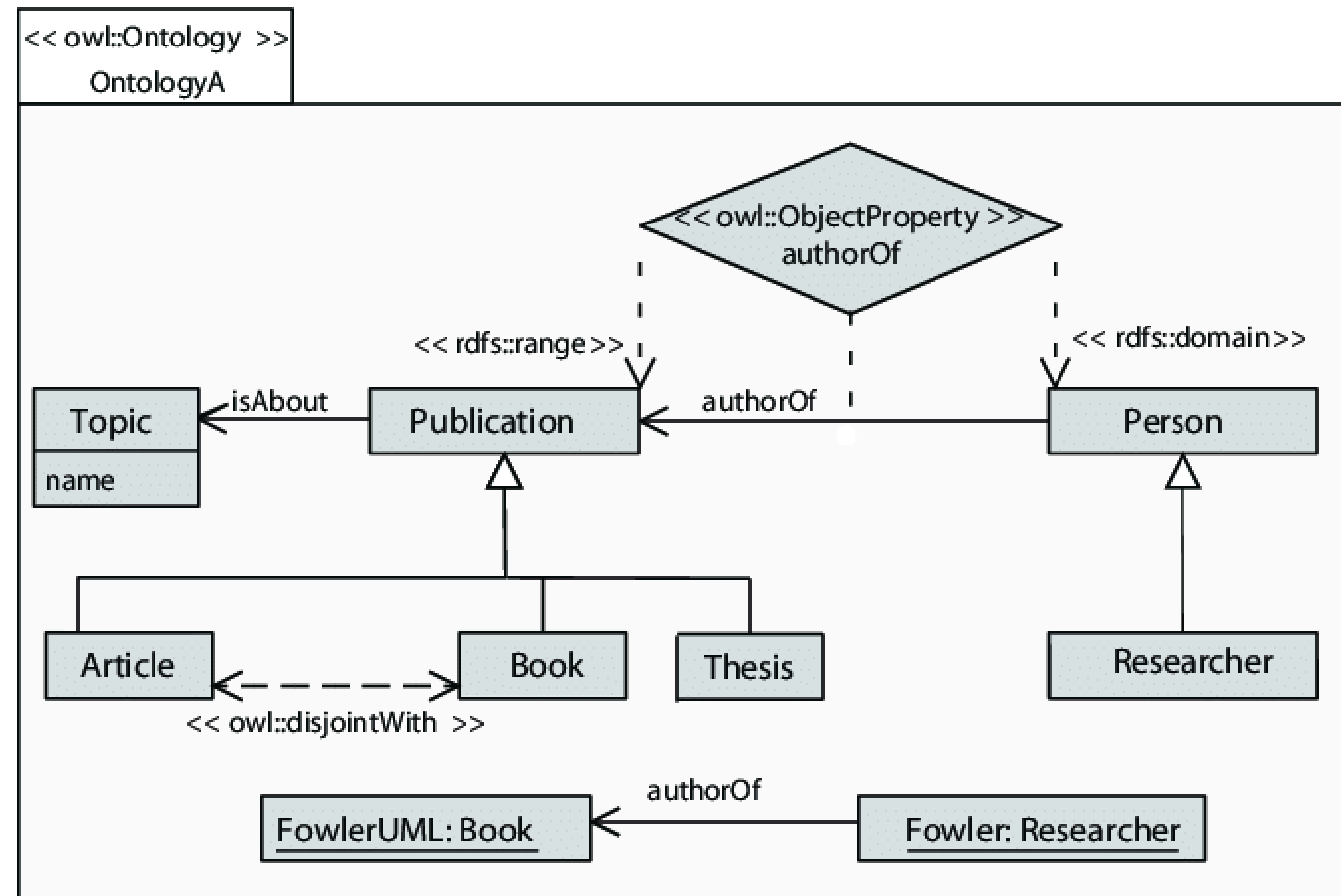
son(X,Y) :- parent(Y,X), %s1
            male(X).
```

?- son(S,jacob).



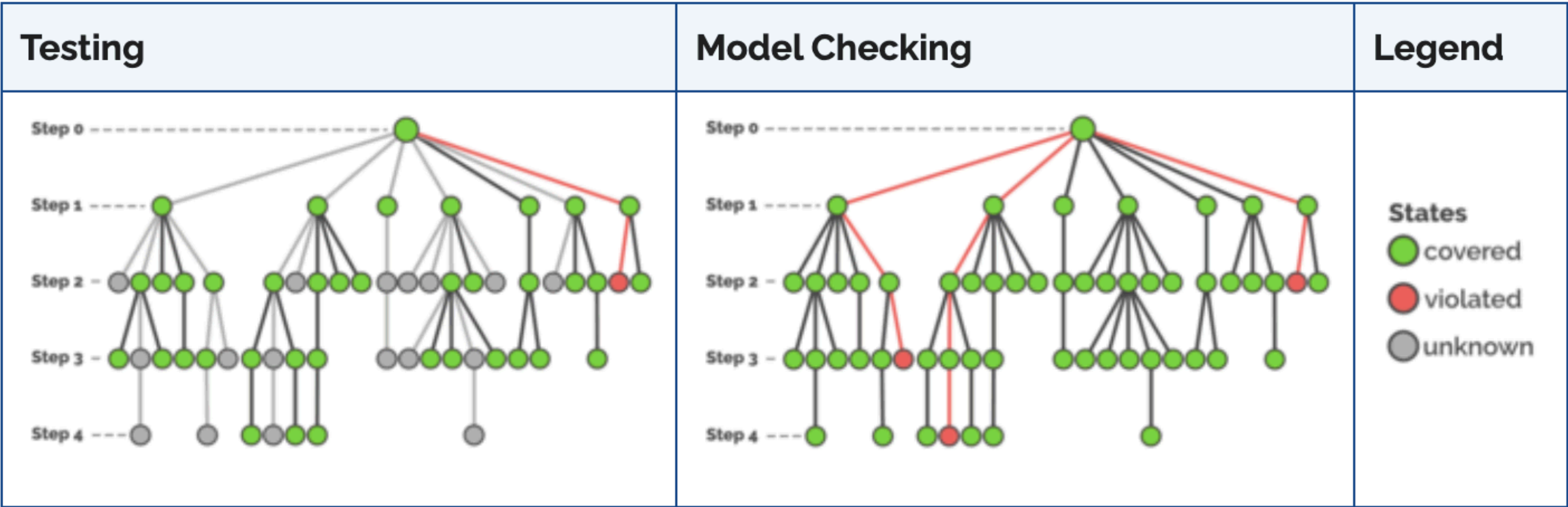
Examples of Symbolic AI (pt. 3)

Ontology definition in OWL



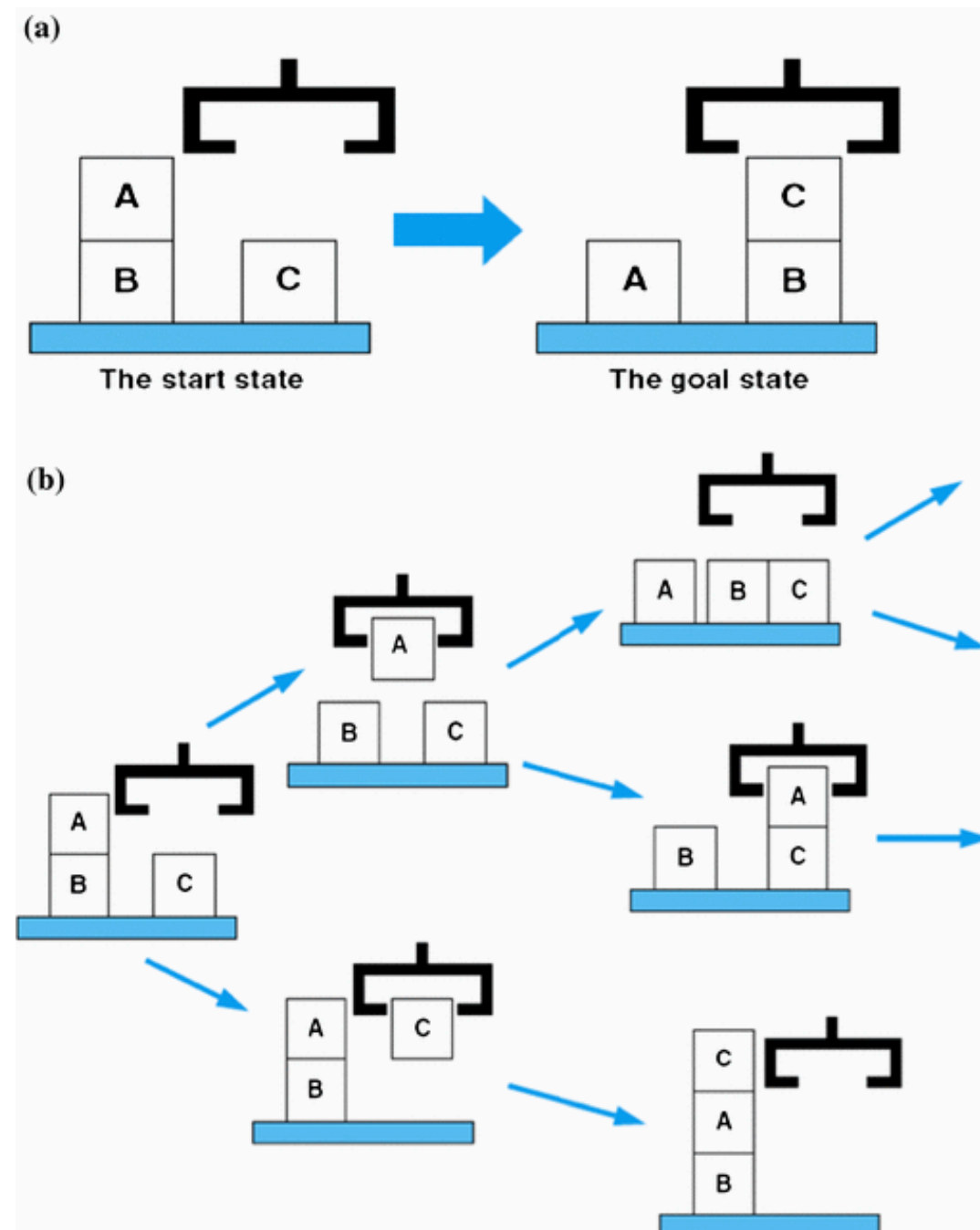
Examples of Symbolic AI (pt. 4)

Model-checking (as opposed to testing)



Examples of Symbolic AI (pt. 5)

Planning in STRIPS



Available actions

- **grab(X)**: grabs block **X** from the table
- **put(X)**: puts block **X** on the table
- **stack(X, Y)**: stacks block **X** on top of block **Y**
- **unstack(X, Y)**: un-stacks block **X** from block **Y**

What do these *symbolic* approaches have in common?

- **Structured representations:** knowledge (I/O data) is represented in a structured, formal way (e.g., logic formulas, ontologies)
- **Algorithmic manipulation of representations:** each approach relies on algorithms that manipulate these structured representations following exact rules
- **Crisp semantics:** the meaning of the representations is well-defined, and the algorithms produce exact results
 - representations are either *well-formed or not*, algorithms rely on rules which are either *applicable or not*
- **Model-driven:** algorithms may commonly work in zero- or few-shot settings, humans must commonly model and encode knowledge in the target structure
- **Clear computational complexity:** the decidability, complexity, and tractability of the algorithms are well understood



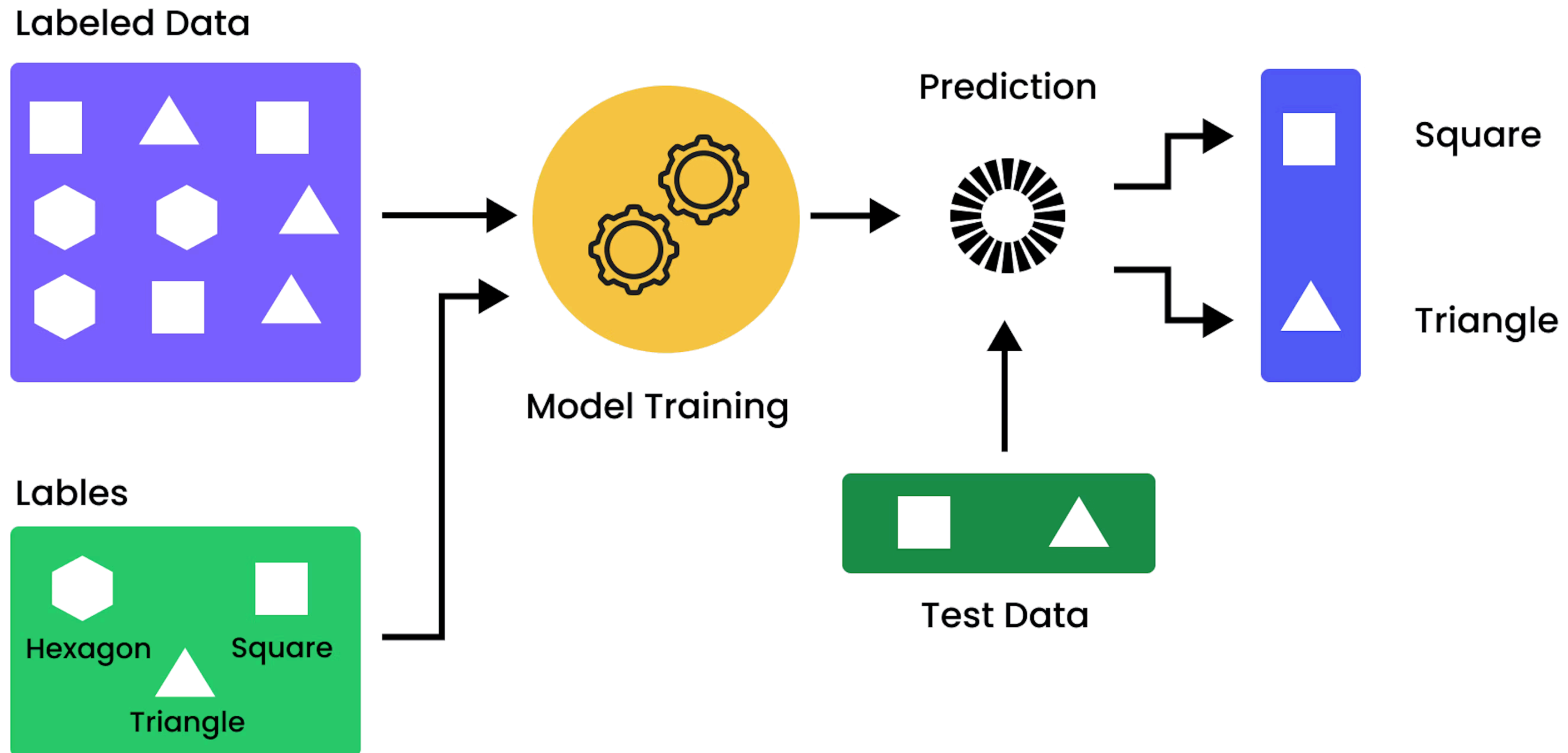
Examples of Sub-symbolic AI (pt. 1)

- **Machine learning**: supervised, unsupervised, and reinforcement learning
 - *Supervised* learning: fitting a discrete (classification) or a continuous function (regression) from examples
 - *Unsupervised* learning: clustering, dimensionality reduction
 - *Reinforcement* learning: learning a policy to maximize a reward signal, via simulation
- **Probabilistic reasoning**: Bayesian networks, Markov models, probabilistic logic programming



Examples of Sub-symbolic AI (pt. 2)

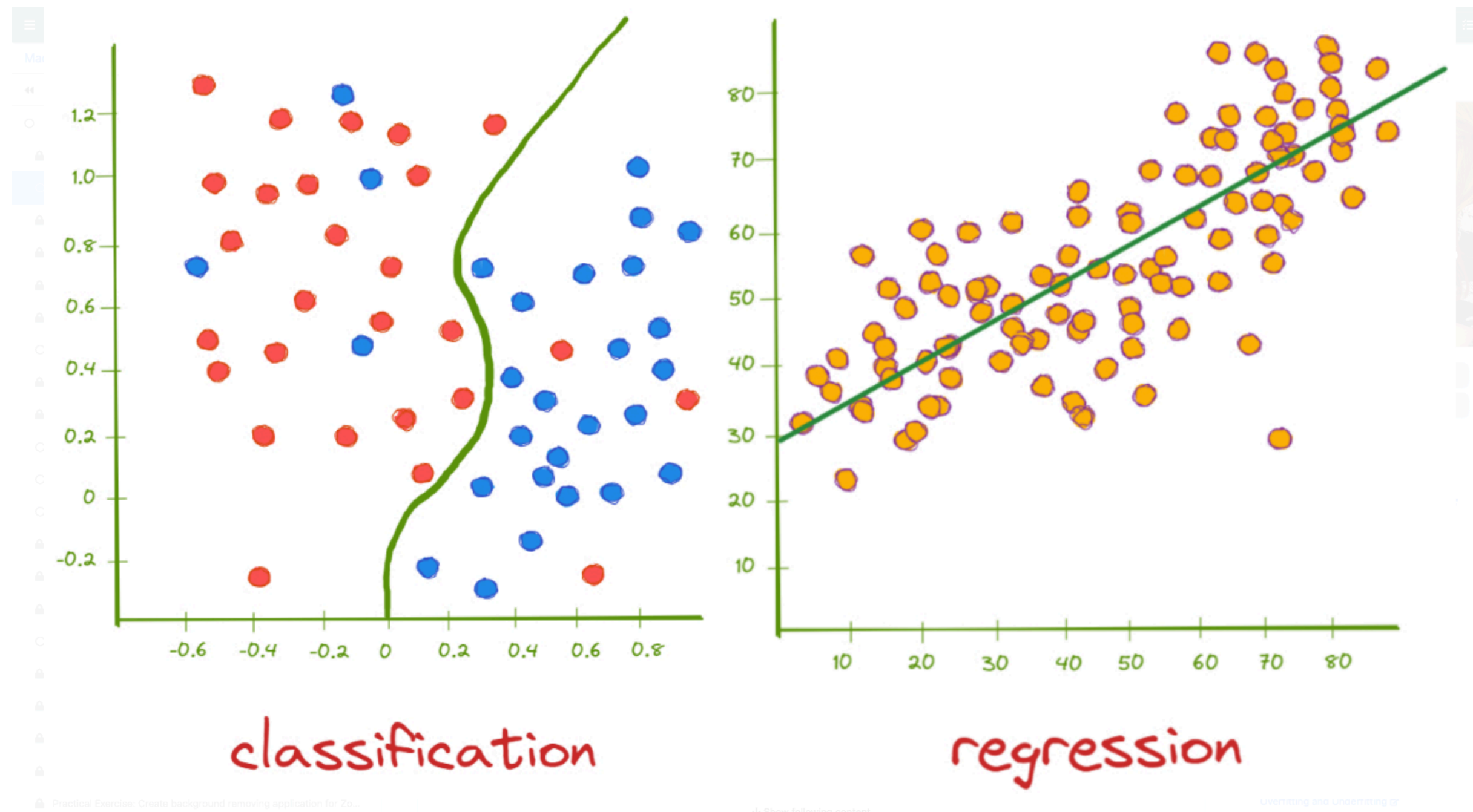
Supervised learning



Examples of Sub-symbolic AI (pt. 3)

Supervised learning – Classification vs. Regression (1/2)

Data separation vs. curve fitting:



Examples of Sub-symbolic AI (pt. 4)

Supervised learning – Classification vs. Regression (2/2)

Focus on the target feature:

Classification Data

X ₁	X ₂	X ₃	X _p	Y
				cat
				dog
				cat
				cat

Categorical
"Labels"

Regression Data

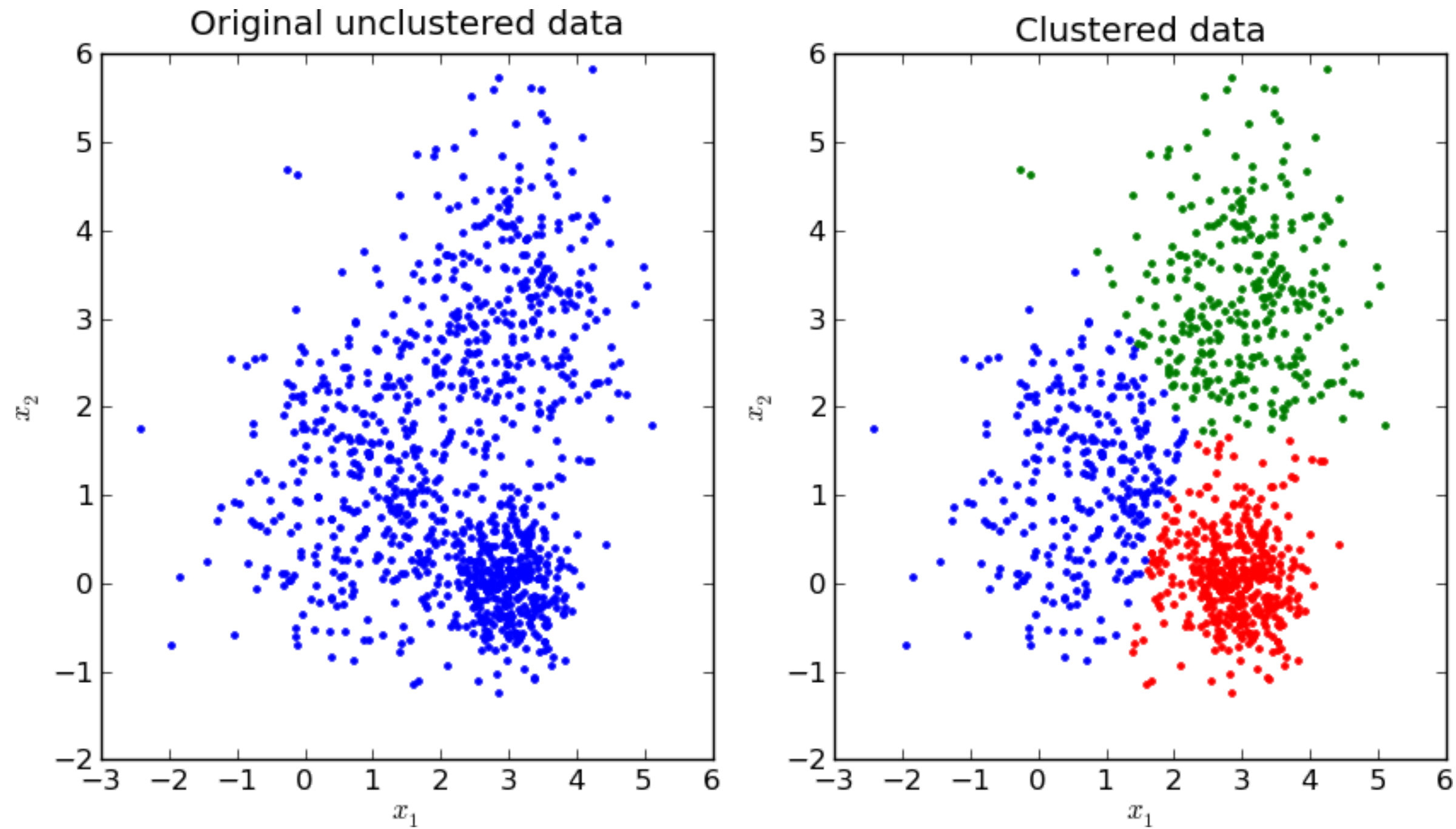
X ₁	X ₂	X ₃	X _p	Y
				5.2
				1.3
				23.0
				7.4

Numeric
Target



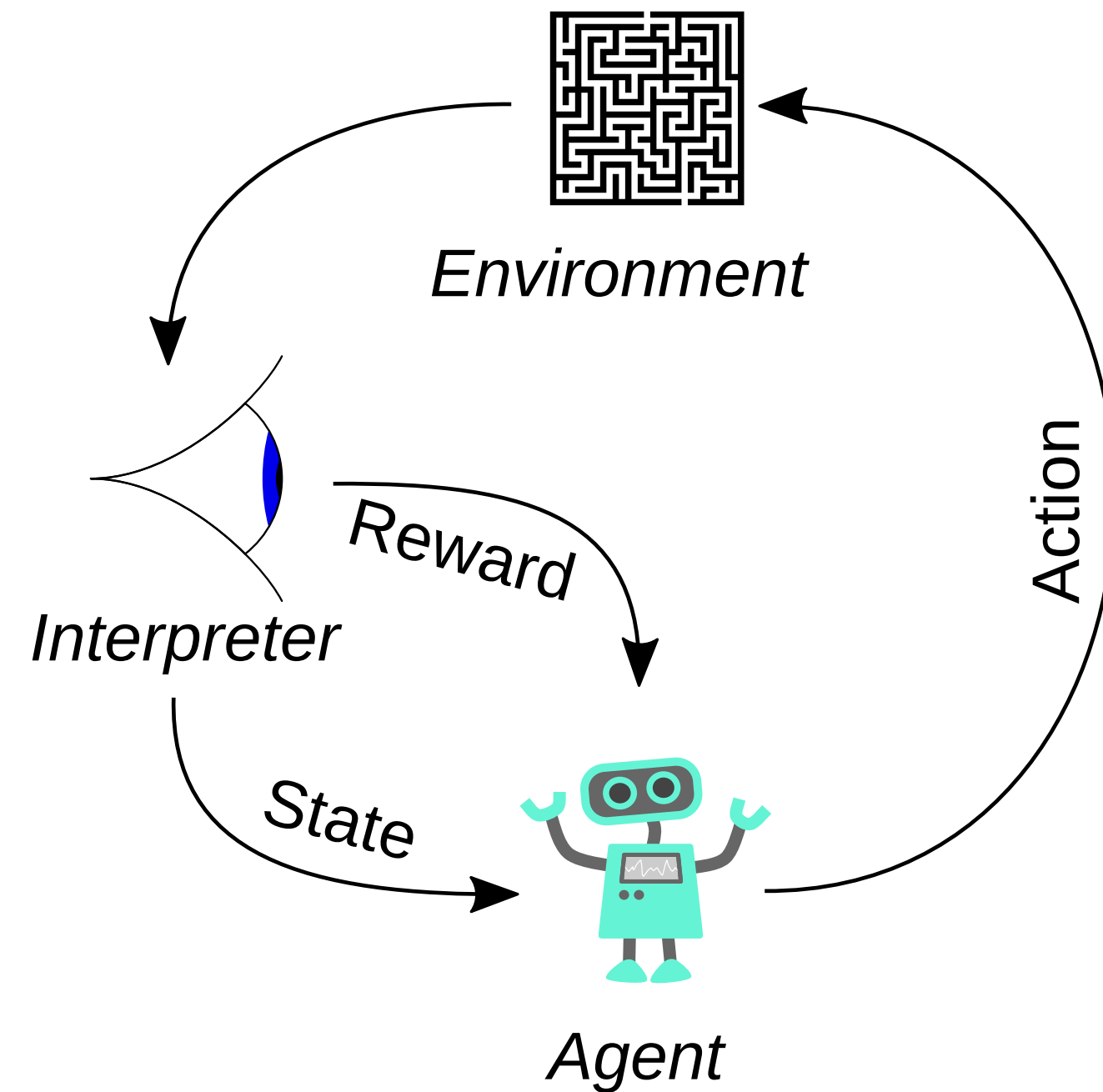
Examples of Sub-symbolic AI (pt. 5)

Unsupervised learning – Clustering



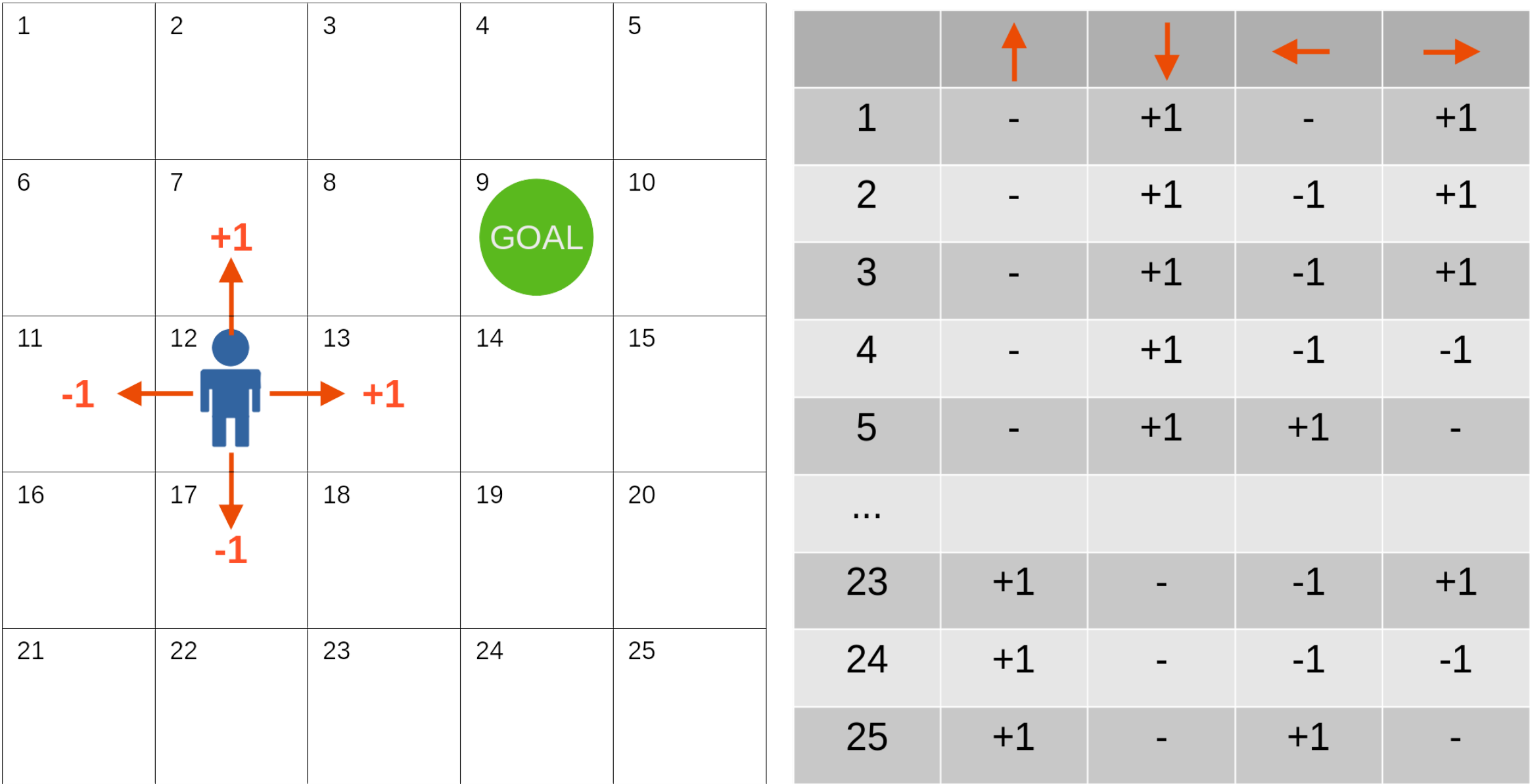
Examples of Sub-symbolic AI (pt. 6)

Unsupervised learning – Reinforcement learning (metaphor)



Examples of Sub-symbolic AI (pt. 7)

Reinforcement learning – Reinforcement learning (policy)



What do these *sub-symbolic* approaches have in common?

- **Numeric representations:** knowledge (I/O data) is represented in a less structured way, often as vectors/matrices/tensors of numbers
- **Differentiable manipulation of representations:** algorithms rely on mathematical operations involving these numeric representations, most-commonly undergoing some optimization process
 - e.g., sum, product, max, min, etc.
- **Fuzzy/continuous semantics:** representations are from continuous spaces, where similarities and distances are defined in a continuous way, and algorithms may yield fuzzy results
- **Data-driven + Usage vs. training:** algorithms are often trained on data, to be later re-used on other data
 - usage is commonly impractical or impossible without training
- **Unclear computational complexity:** strong reliance on greedy or time-limited optimization methods, lack of theoretical guarantees on the quality of the results



Long-standing dualism

Intuition vs. Reasoning

1. Esprit de *finesse* vs. Esprit de *géométrie* (Philosophy) — Blaise Pascal, 1669
2. *Cognitive* vs. *Behavioural* Psychology — B.F. Skinner, 1950s
3. *System 1* (fast, intuitive) vs. *System 2* (slow, rational) — Daniel Kahneman, 2011

Sub-symbolic AI

- Provides mechanisms emulating human-like *intuition*
- *Quick*, possibly *error-prone*, but often *effective*
- Requires *learning* from data
- Often *opaque*, hard to interpret or explain

Symbolic AI

- Provides mechanisms emulating human-like *reasoning*
- *Slow*, but *precise* and *verifiable*
- Requires symbolic *modeling* and *encoding* knowledge
- Often *transparent*, easier to interpret and explain



Need for integration

- the **NeSy community** has long recognized the *complementarity* among symbolic and sub-symbolic approaches...
- ... with a focus on **neural-networks** (*NN*) based sub-symbolic methods, as they are very *flexible*

Patterns of *integration* or *combination* (cf. **Bhuyan et al., 2024**)

1. **Symbolic Neuro-Symbolic**: symbols \rightarrow vectors \rightarrow NNs \rightarrow vectors \rightarrow symbols
2. **Symbolic[Neuro]**: symbolic module $\xrightarrow{\text{invokes}}$ NN \rightarrow output
3. **Neuro | Symbolic**: NN $\xrightarrow{\text{cooperates}}$ symbolic module $\xrightarrow{\text{cooperates}}$ NN \rightarrow ...
4. **Neuro-Symbolic \rightarrow Neuro**: symbolic knowledge $\xrightarrow{\text{influences}}$ NN
5. **Neuro_{Symbolic}**: symbolic knowledge $\xrightarrow{\text{constrains}}$ NN
6. **Neuro[Symbolic]**: symbolic module $\xrightarrow{\text{embedded in}}$ NN



Focus on two main approaches

(cf. [Ciatto et al., 2024](#))

- Symbolic Knowledge **Extraction** (*SKE*): extracting symbolic knowledge from sub-symbolic models
 - for the sake of *explainability* and *interpretability* in machine learning
- Symbolic Knowledge **Injection** (*SKI*): injecting symbolic knowledge into sub-symbolic models
 - for the sake of *trustworthiness* and *robustness* in machine learning

Both require some basic understanding of how *supervised machine learning* works



Supervised Machine Learning 101

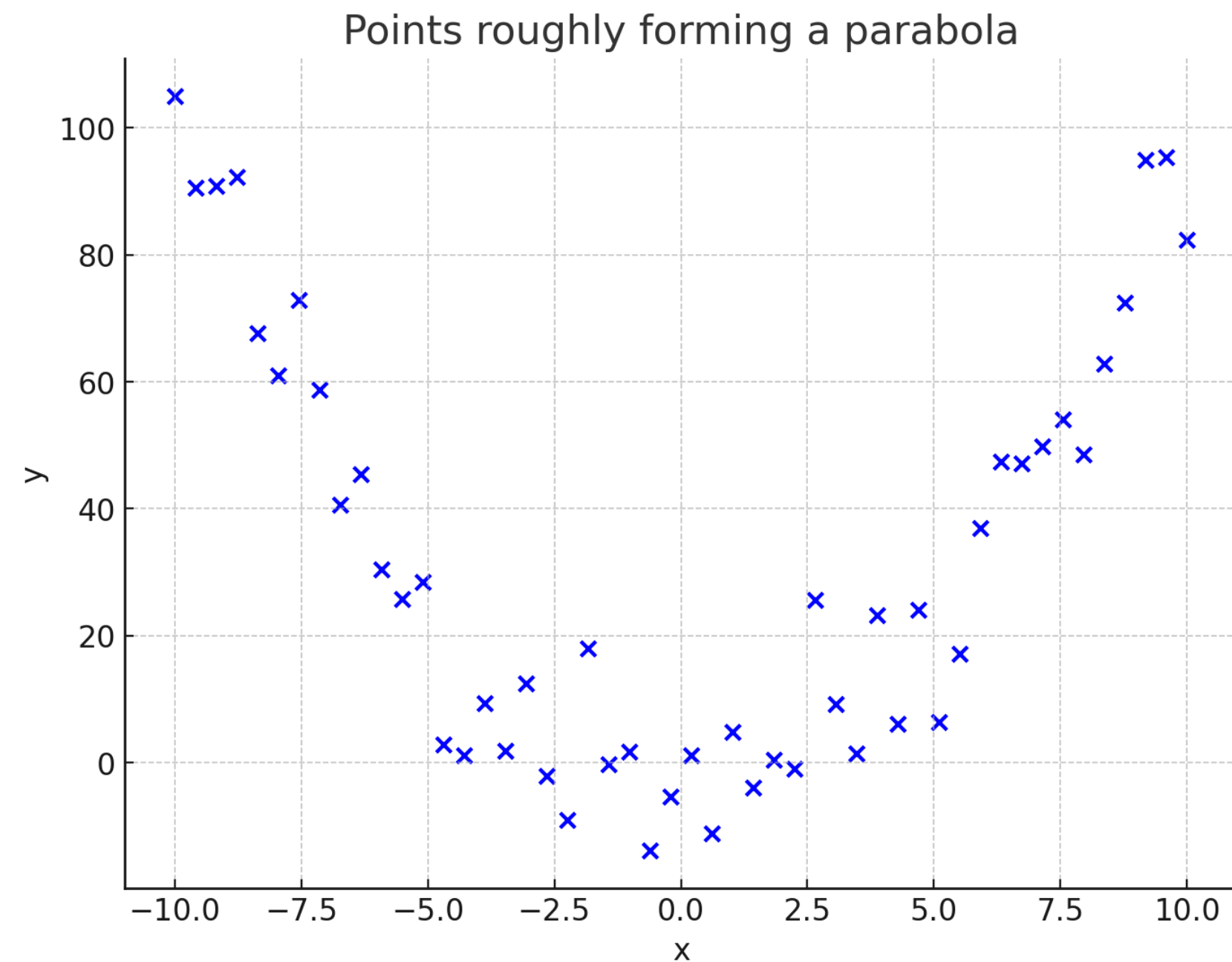
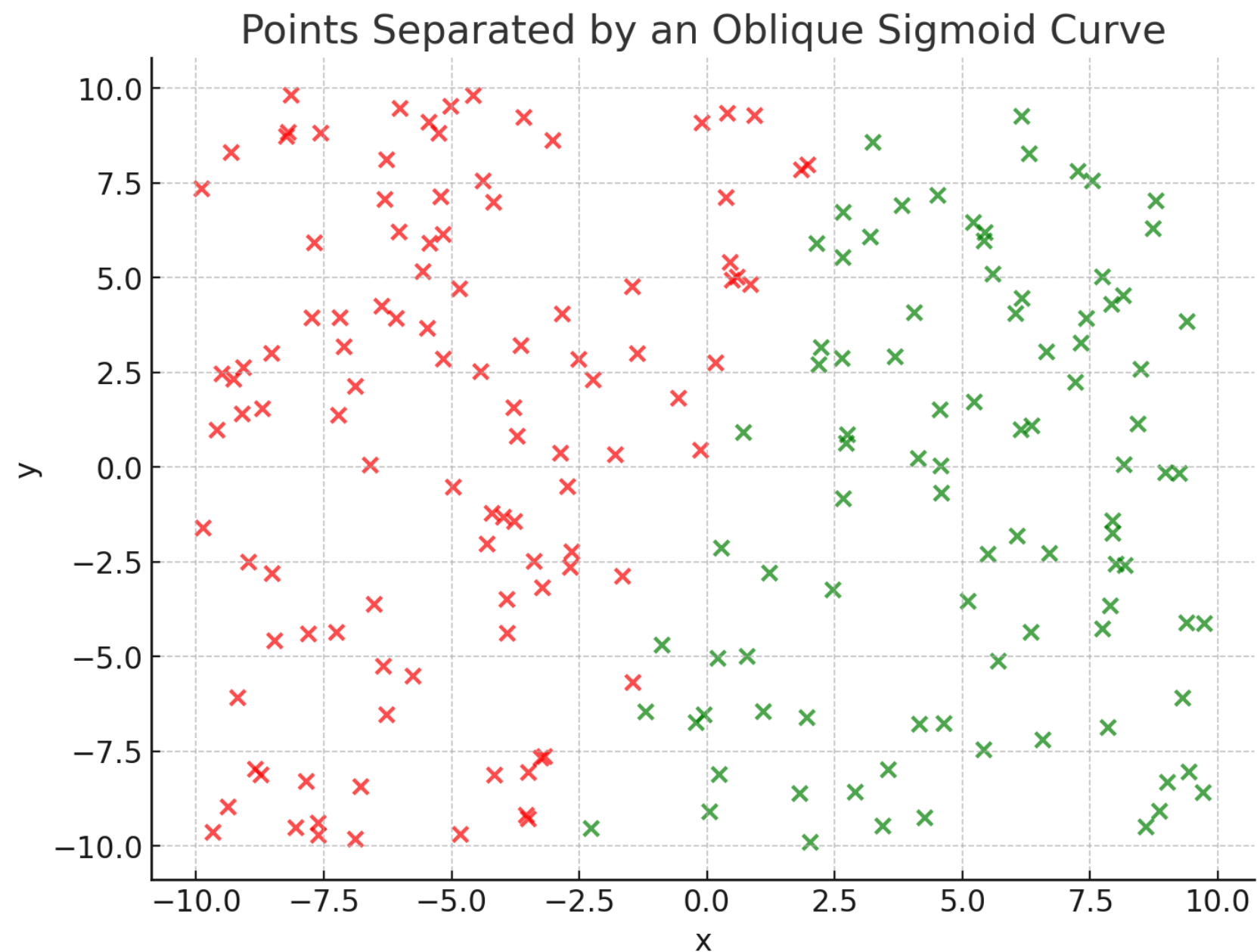


Supervised Machine Learning 101 (pt. 1)

1. Let's say you have a **dataset** of vectors in \mathbb{R}^n

2. Let's say the vectors are *labelled differently* (2 colors)

2. Let's say the vectors are *aligned* (along a curve)



3. Let's say you want to **separate** them

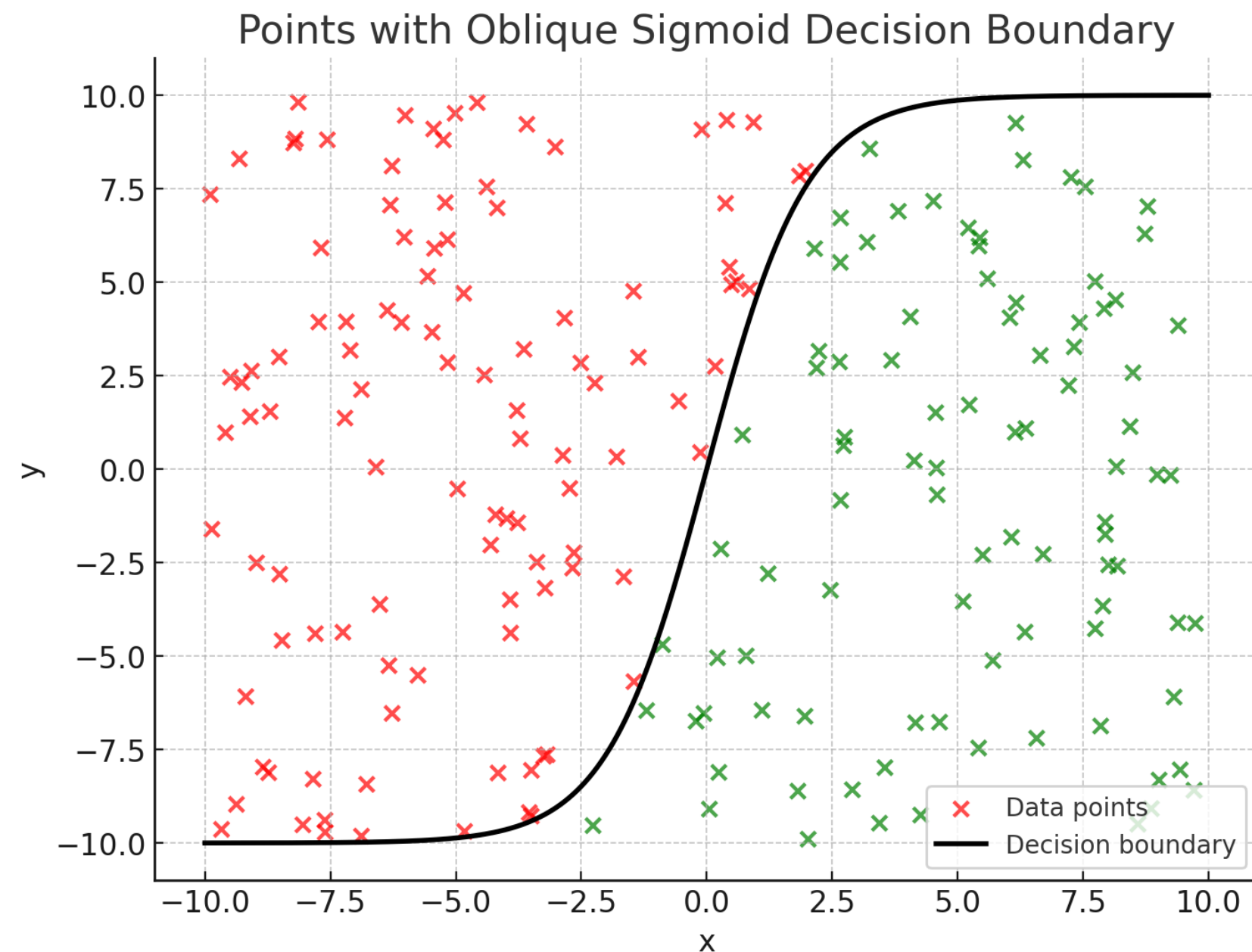
3. Let's say you want to **interpolate** them



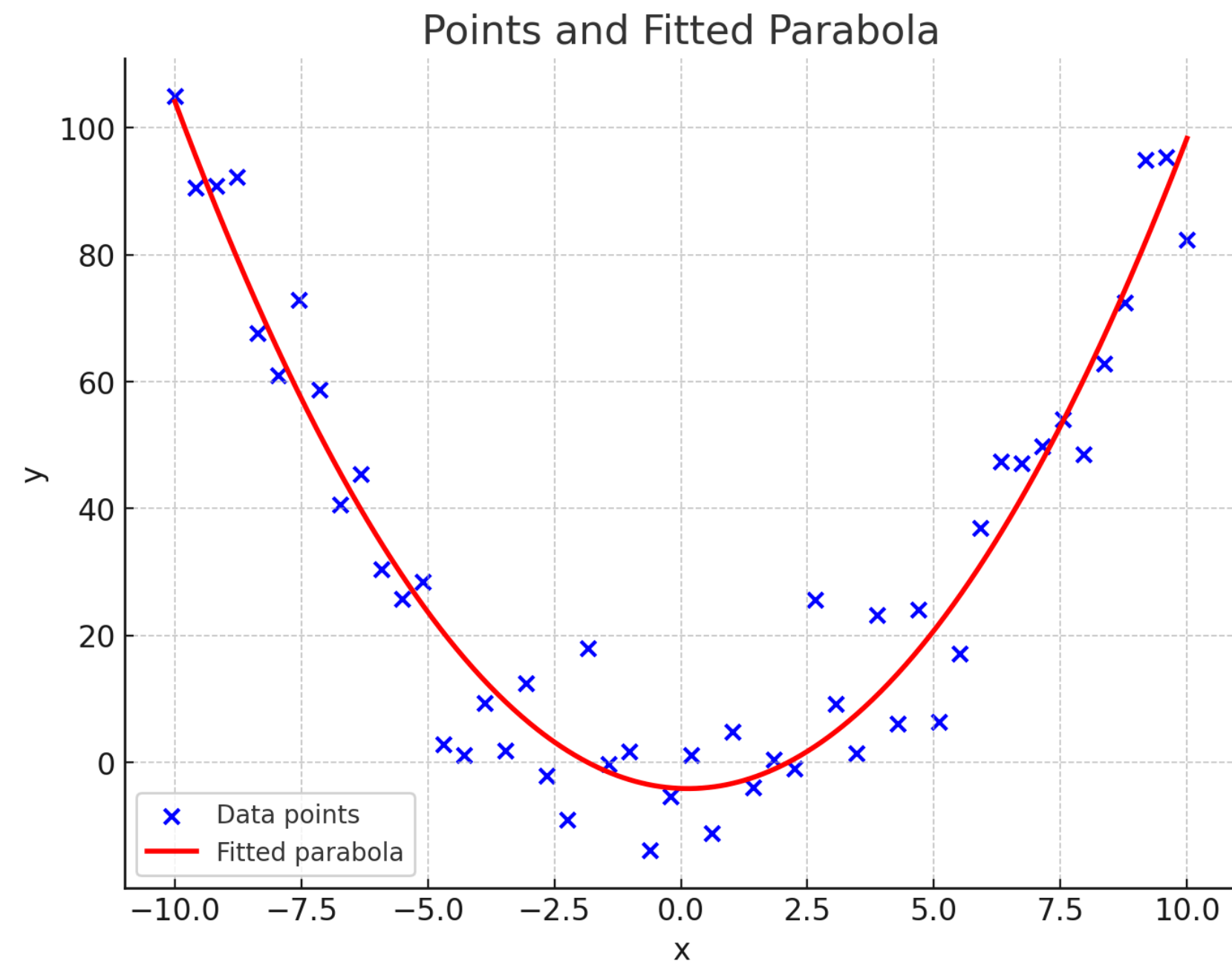
Supervised Machine Learning 101 (pt. 2)

4. Then, need to *approximate* the function f ...

4. ... **separating them** (the sigmoid here)



4. ... **interpolating them** (the parabola here)



5. The function f is the **decision boundary** (a.k.a. *classifier*)

5. The function f is the **regression curve** (a.k.a. *regressor*)

6. How to compute such a function f ?



Supervised Machine Learning 101 (pt. 3)

Let's formalize the problem (1/2)

1. Let's assume the data represented as a set of vectors in a **dataset** $D \subset \mathbb{R}^n$ s.t. $|D| = m$
 - e.g. $D = (x_1, y_1, c_1), (x_2, y_2, c_2), \dots, (x_m, y_m, c_m)$ for the classification dataset
 - e.g. $D = (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ for the regression dataset
2. Let \mathcal{X} (resp. \mathcal{Y}) be the *input* (resp. output or *target*) *space* of the data at hand
 - so $\exists X \in \mathcal{X}$ and $\exists Y \in \mathcal{Y}$ s.t. $(X, Y) \equiv D$
 - i.e. X is the *input data* and Y is the *target data* in D



Supervised Machine Learning 101 (pt. 4)

Let's formalize the problem (2/2)

3. Let $\mathcal{H} = \{f \mid f : \mathcal{X} \rightarrow \mathcal{Y}\}$ be the set of all possible functions mapping \mathcal{X} to \mathcal{Y}
 - i.e. the so-called **hypothesis space**
4. We need now to find **the best** $f^* \in \mathcal{H}$
 - best w.r.t to what?
5. We need to define a **loss function** \mathcal{L} that quantifies the *difference* between the *predicted output* $f(X)$ and the *true output* Y
 - for any possible $f \in \mathcal{H}$
6. So, our **learning problem** is as simple as a **search problem** in the hypothesis space:

$$f^* = \arg \min_{f \in \mathcal{H}} \mathcal{L}(f(X), Y)$$

7. Ok, but in *practice*, how do we do that?



Supervised Machine Learning 101 (pt. 5)

Let's solve the problem

1. How to *explore* the *hypothesis space*, depends on what *sorts of functions* it contains
2. There exist several sorts functions for which an **exploration algorithm** is known
 - e.g. $\mathcal{H} \equiv$ polynomials of degree 1 (i.e. *lines*: $f(\bar{x}) = \beta + \bar{\omega} \cdot \bar{x} = \beta + \omega_1 x_1 + \dots + \omega_n x_n$)
 - e.g. $\mathcal{H} \equiv$ polynomials of degree 2 (i.e. *parabolas*: $f(\bar{x}) = \beta + \omega_1 x_1 + \dots + \omega_n x_n + \omega_1^2 x_1^2 + \dots + \omega_n^2 x_n^2$)
 - e.g. $\mathcal{H} \equiv$ decision trees (i.e. *if-then-else* rules: $f(\bar{x}) = \text{if } x_1 \leq c_1 \text{ then } y_1 \text{ else if } x_2 \leq c_2 \text{ then } y_2 \dots$)
 - e.g. $\mathcal{H} \equiv k$ -nearest neighbors ($f(\bar{x}) = \text{class of (most of) the } k \text{ nearest neighbors of } \bar{x}$)
 - e.g. $\mathcal{H} \equiv$ neural networks (i.e. *NNs*: $f(\bar{x}) = \sigma(\beta + \bar{\omega} \cdot \bar{x})$ where σ is a non-linear activation function, e.g. sigmoid, ReLU, etc.)

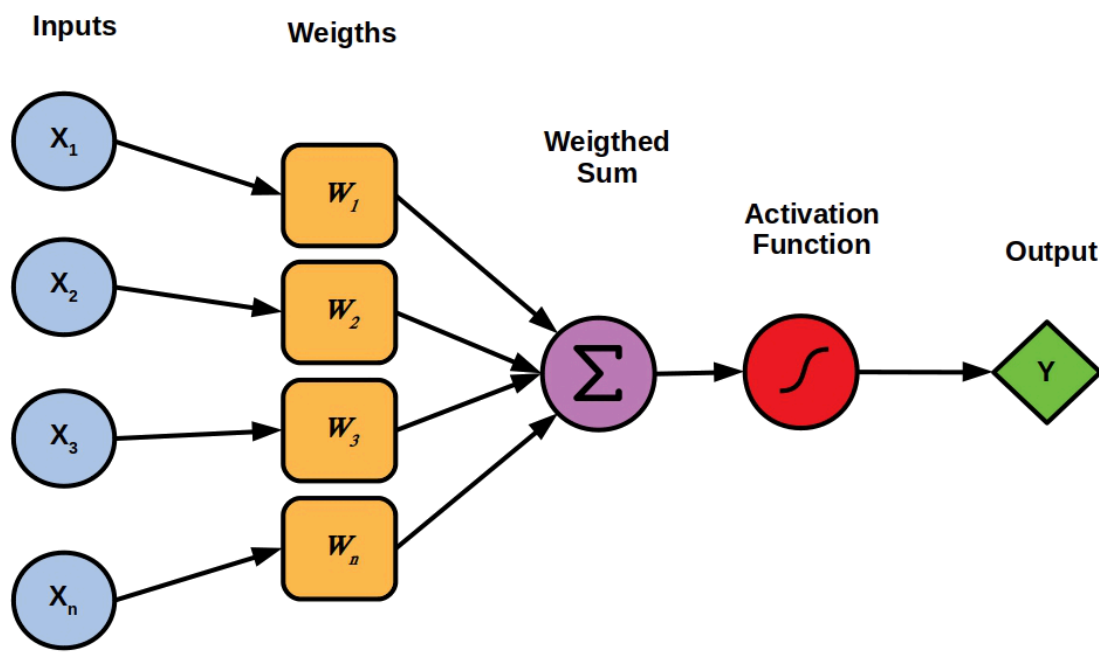
Remark: because of the “no free lunch” theorem, *no* single algorithm is the best in *the general case*

- i.e., each learning problem may be better addressed by a different learning algorithm

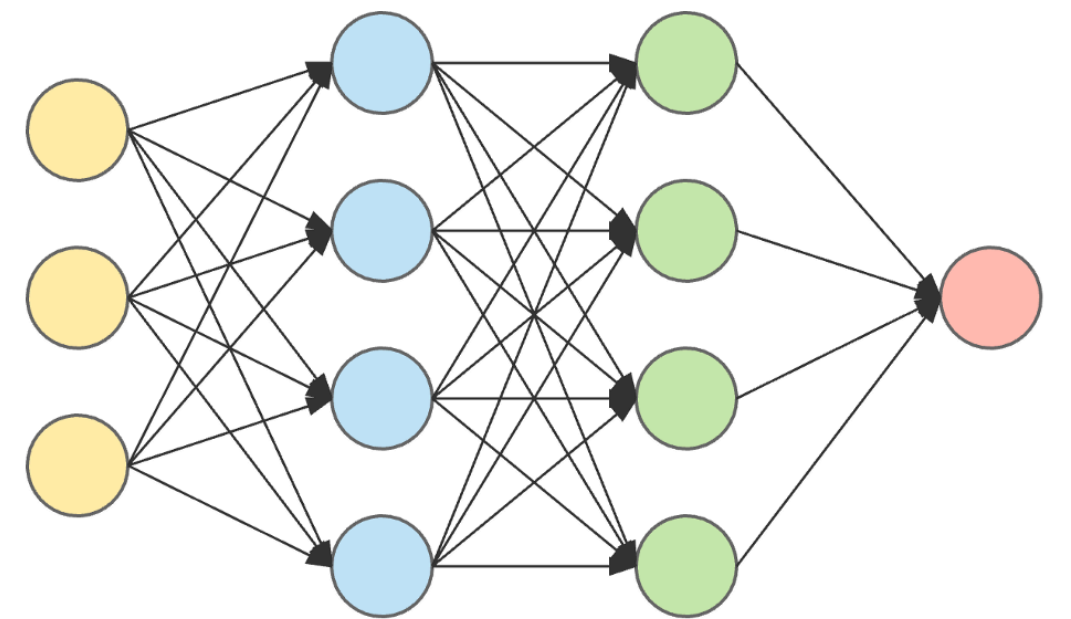


Supervised Machine Learning 101 (pt. 6)

Let's solve the problem with **neural networks**

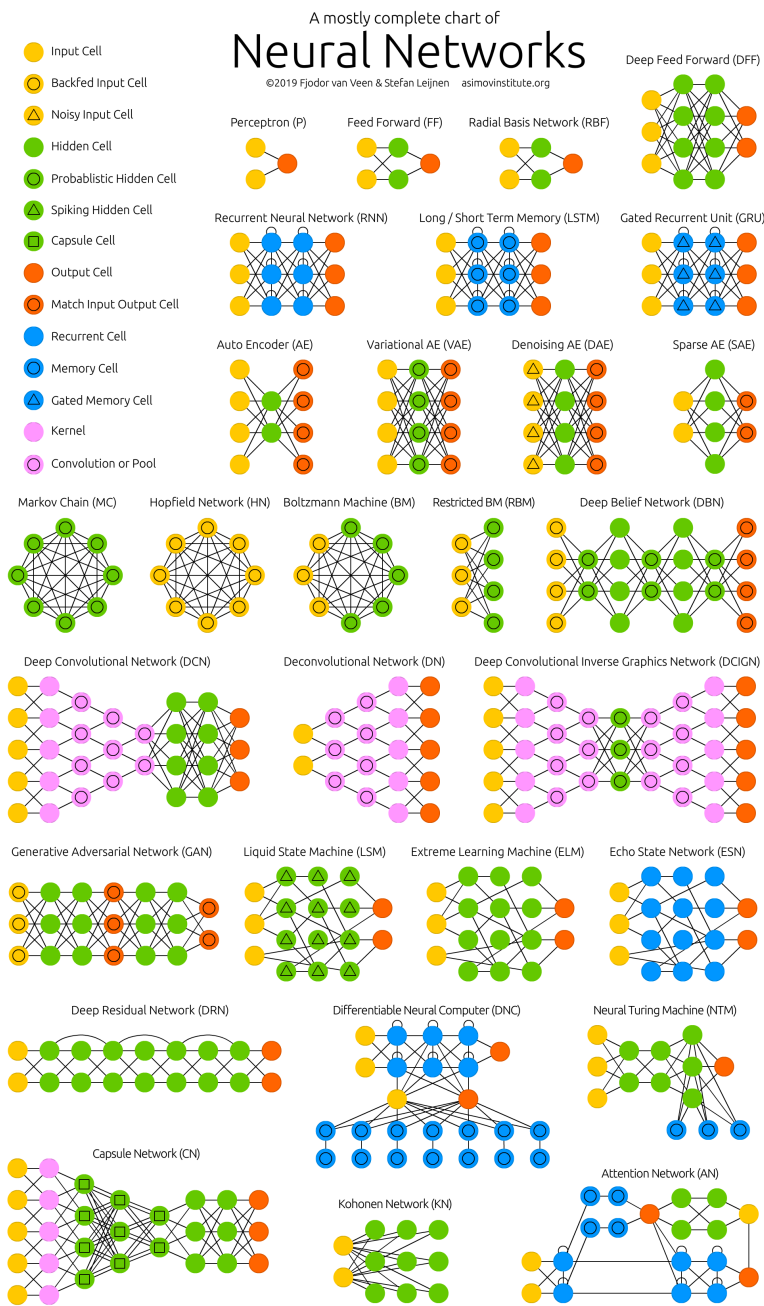


Single neuron



input layer hidden layer 1 hidden layer 2 output layer

(Feed-forward)
Neural network \equiv cascade of *layers*



Many admissible architectures



Supervised Machine Learning 101 (pt. 7)

Let's solve the problem with **gradient descent**



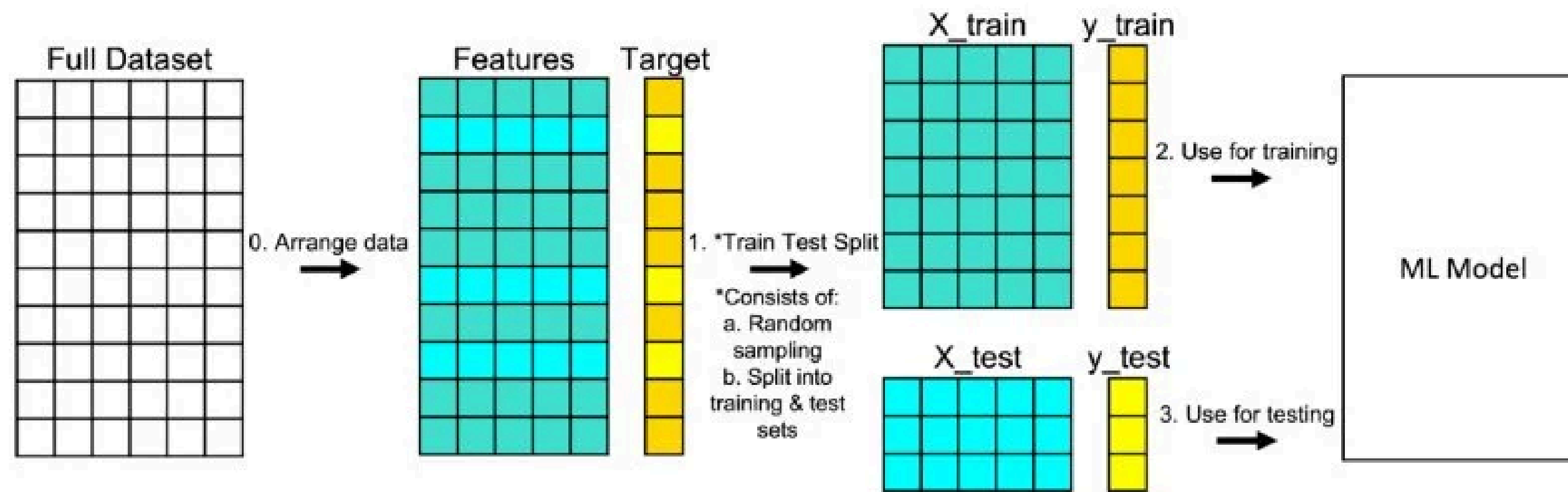
Supervised Machine Learning 101 (pt. 8)

How things may go wrong – *Under*fitting vs. *Over*fitting



Supervised Machine Learning 101 (pt. 9)

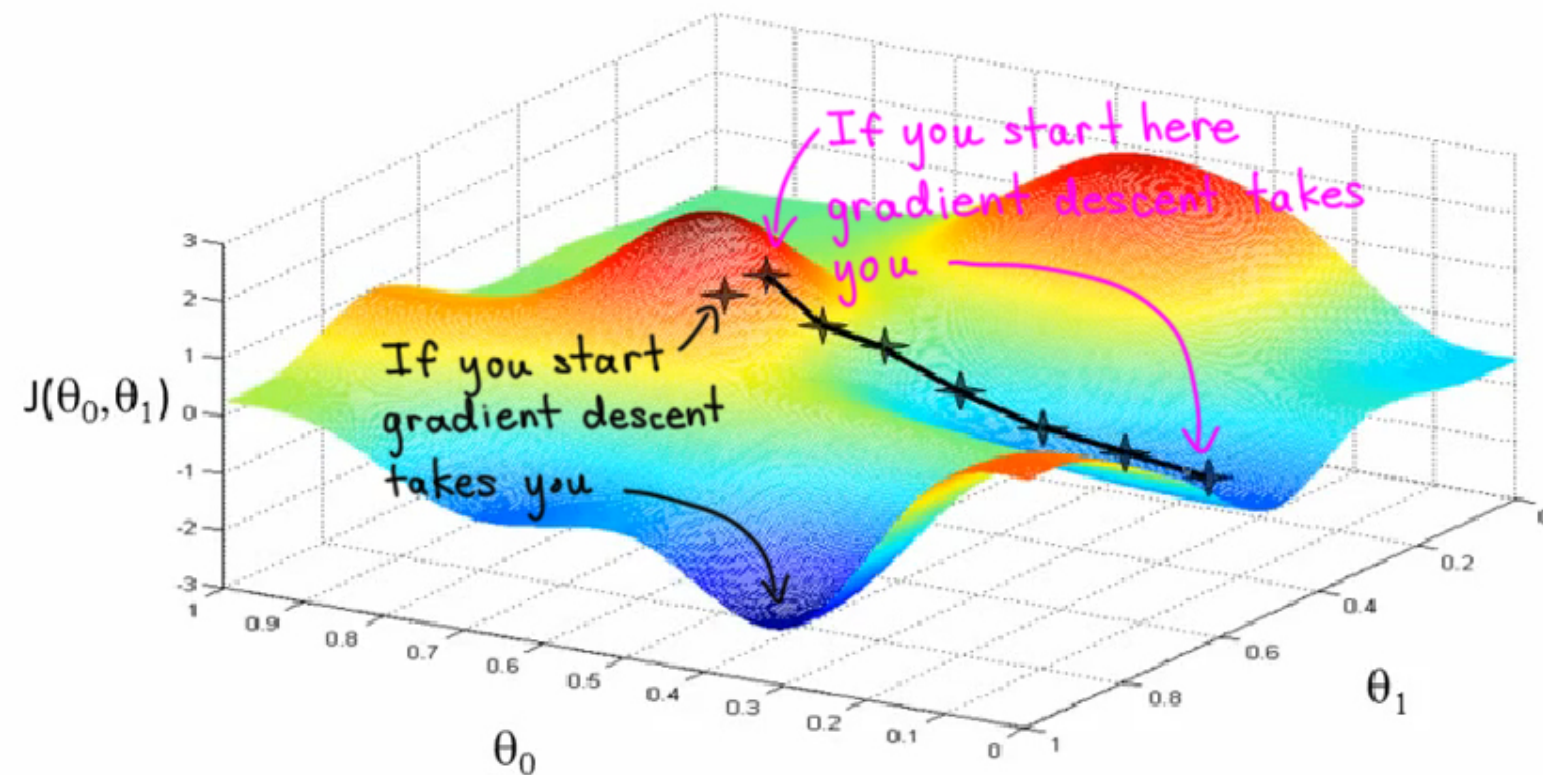
How to address: Test-Set Separation



Supervised Machine Learning 101 (pt. 10)

How things may go wrong – Training is **stochastic**

Depending on its start, training may yield **different results**:



Solution is **cross-validation**:

	Training Set (K-1)/K %			Validation Set 1/K %
Fold 1	Train	Train	Train	Validation
Fold 2	Train	Train	Validation	Train
...	Train	Validation	Train	Train
Fold K	Validation	Train	Train	Train

1. Train the **k different models** (same architecture) on k different subsets of the training set
2. **Average** the **results** of the k models
3. If average is good, the model is considered more **robust** and **reliable**
 - useful when **comparing** different **architectures** or hyper-parameters

Supervised Machine Learning 101 (pt. 11)

How things may go wrong – Data may not be numeric

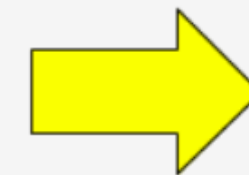
Solution: **change encoding**

workclass	
State-gov	0
Self-emp-not-inc	1
Private	2
Private	2
Private	2



(*Ordinal* encoding)

Color		Red	Yellow	Green
Red				
Red		1	0	0
Yellow		1	0	0
Green		0	1	0
Yellow		0	0	1



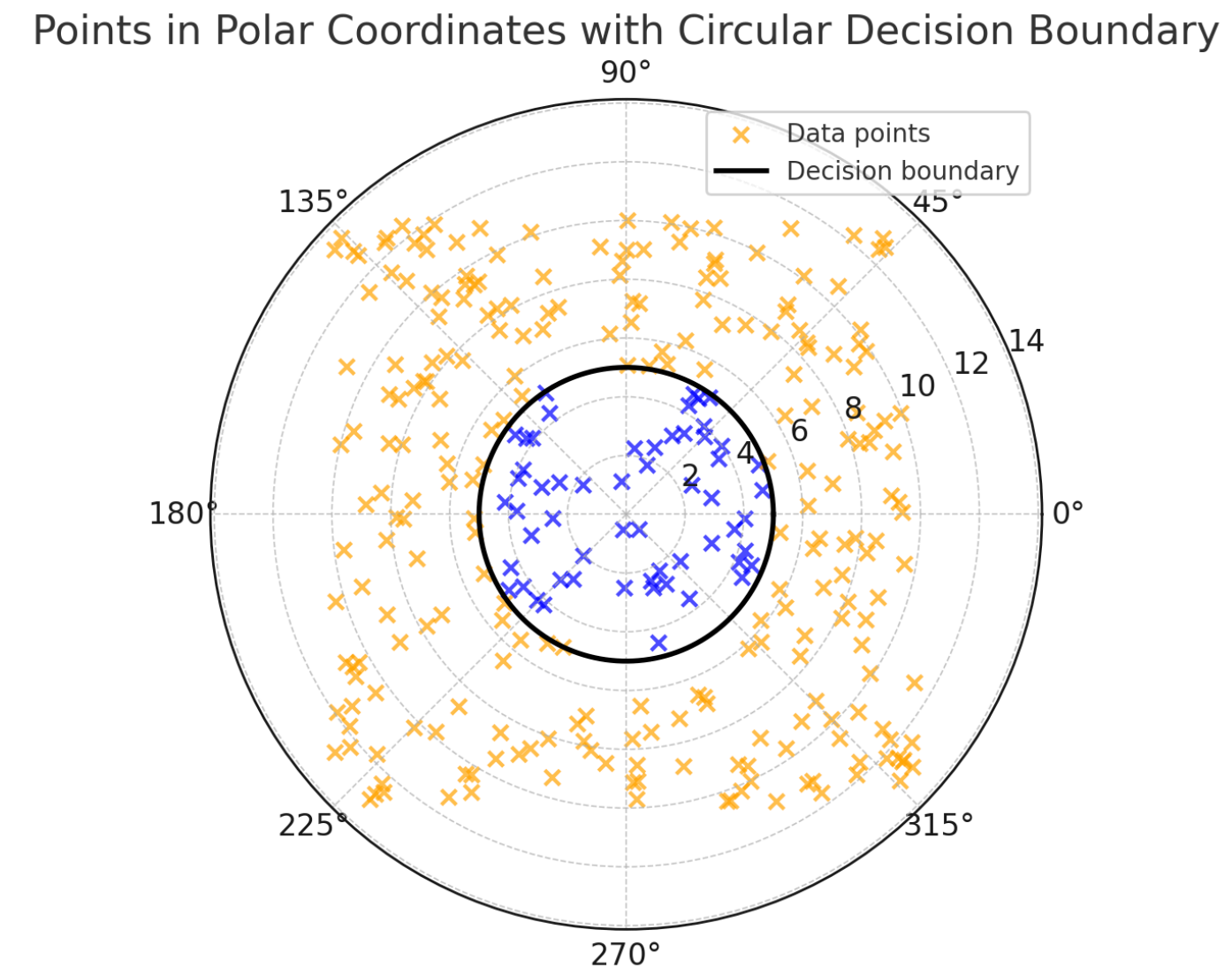
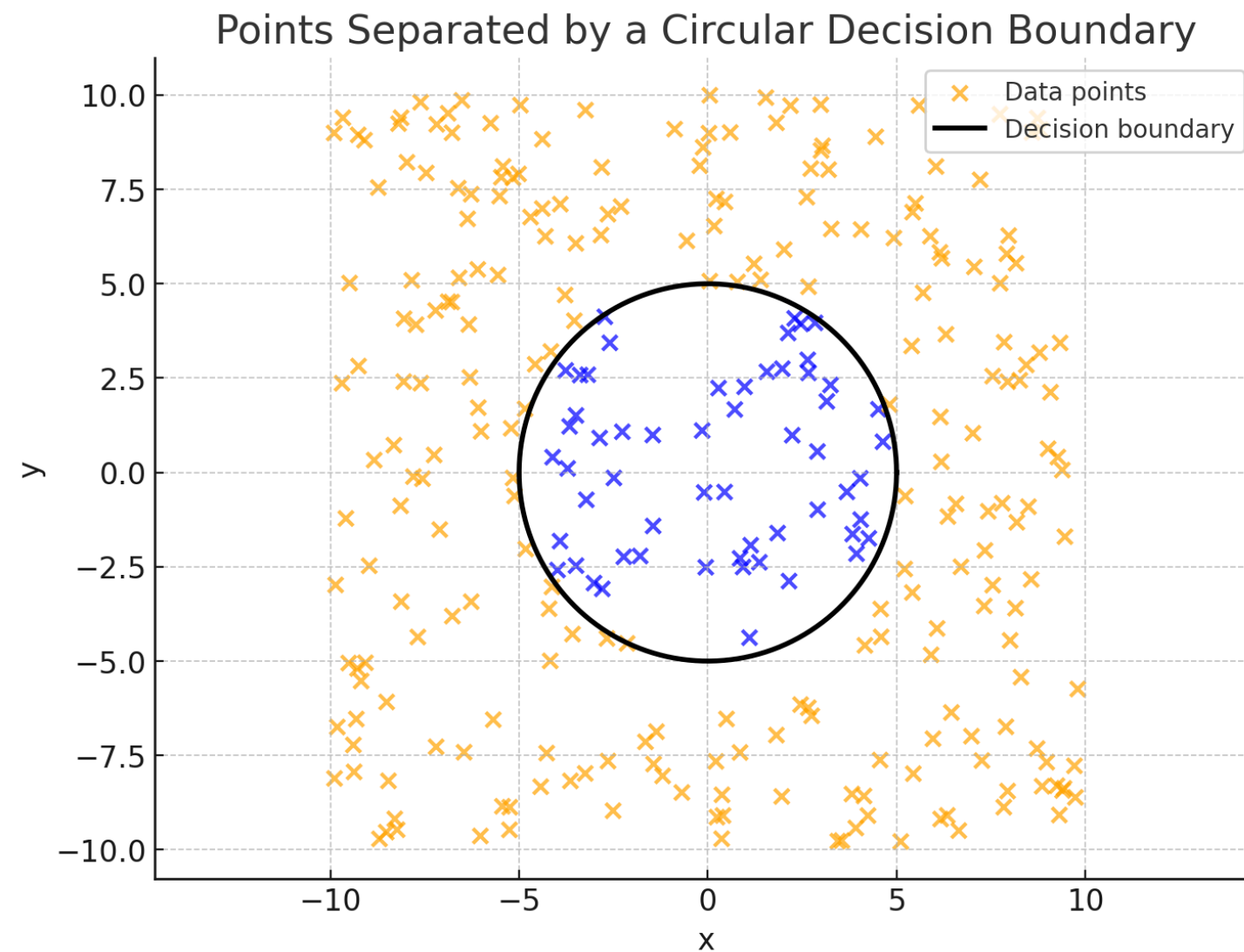
(*One-hot* encoding)



Supervised Machine Learning 101 (pt. 12)

How things may go wrong – Data may *not* be *separable* by a proper function...

Solution: *feature engineering* (map data into a different space)



Non-linearly-separable data ...



... may be separable in *another space*
($\rho = \sqrt{x^2 + y^2}, \theta = \arctan(y/x)$)

Symbolic Knowledge Extraction (SKE)

How to extract symbolic knowledge from sub-symbolic predictors



Definition and Motivation (pt. 1)

any *algorithmic procedure* accepting trained sub-symbolic predictors as input and producing *symbolic* knowledge as output, so that the extracted knowledge reflects the behaviour of the predictor with high *fidelity*.

Definition and Motivation (pt. 2)

- **Explainable AI (XAI)**: SKE methods are often used to provide explanations for the decisions made by sub-symbolic predictors, making them more interpretable and understandable to humans (a.k.a. *post-hoc explainability*)
 - *local explanations*: explanations for individual predictions
 - *global explanations*: explanations for the overall behaviour of the predictor
- **Knowledge discovery**: SKE methods can help discover patterns and relationships in the data that may not be immediately apparent, thus providing insights into the underlying processes
- **Model compression**: SKE methods can simplify complex sub-symbolic models by extracting symbolic rules that approximate their behaviour, thus reducing the model's size and complexity



Explainability vs Interpretability

They are *not* synonyms in spite of the fact that they are often used interchangeably!

Explanation

- elicits *relevant aspects* of objects (to ease their interpretation)
- it is an operation that transform poorly interpretable objects into *more interpretable* ones
- search of a *surrogate* interpretable model

Interpretation

- binds objects with *meaning* (what the human mind does)
- it is *subjective*
- it does not need to be measurable, only *comparisons*



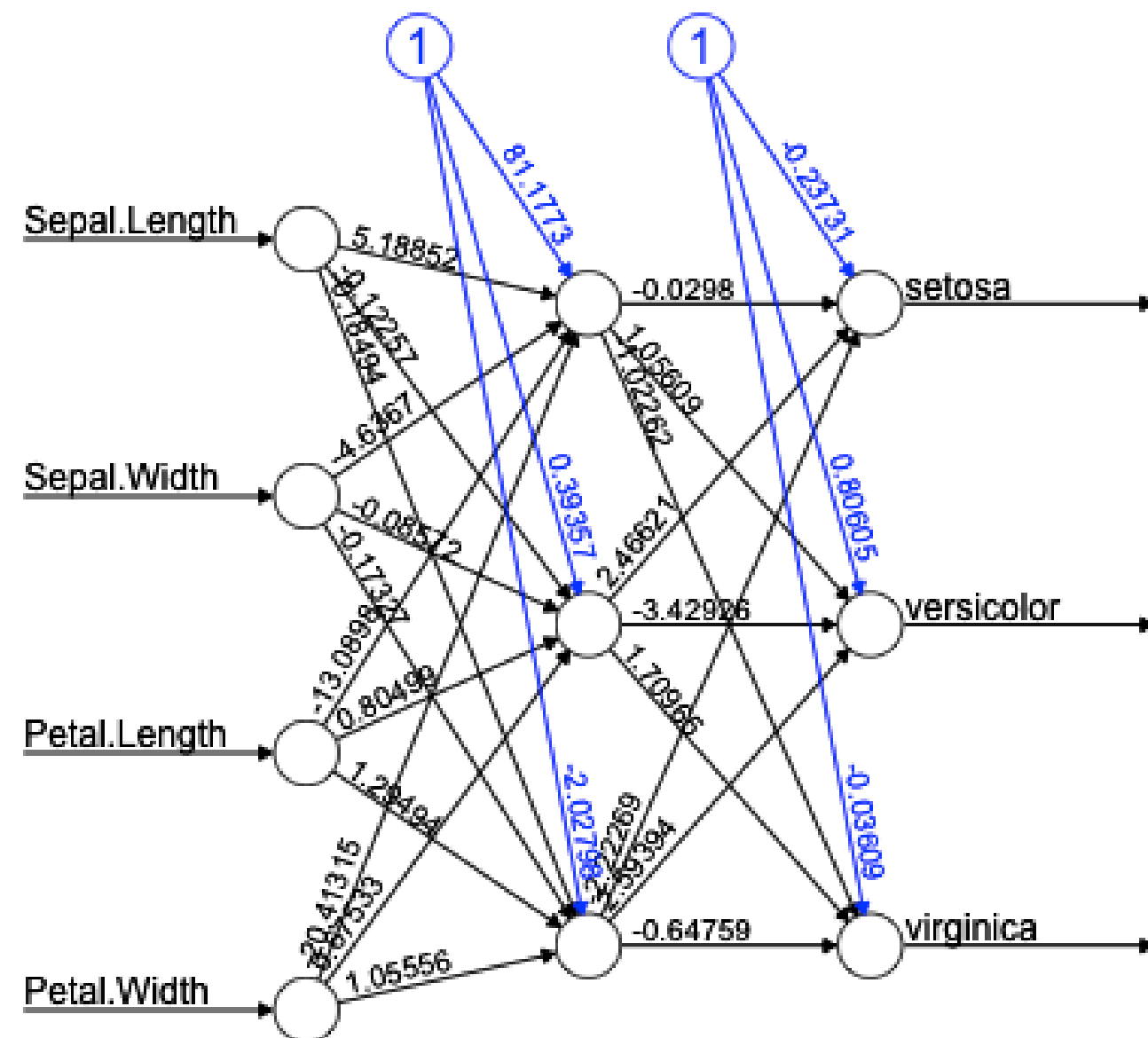
Concepts

Main entities and how to extract symbolic knowledge from sub-symbolic predictors



Entities

Sub-symbolic predictor



Error: 0.346668 Steps: 26926

Symbolic knowledge

Logic Rule

Class = setosa \leftarrow PetalWidth ≤ 1.0

Class = versicolor \leftarrow PetalLength $> 4.9 \wedge$
SepalWidth $\in [2.9, 3.2]$

Class = versicolor \leftarrow PetalWidth > 1.6

Class = virginica \leftarrow SepalWidth ≤ 2.9

Class = virginica \leftarrow SepalLength $\in [5.4, 6.3]$

Class = virginica \leftarrow PetalWidth $\in [1.0, 1.6]$

How SKE works

Decompositional SKE

if the method *needs* to inspect (even partially) the internal parameters of the underlying black-box predictor, e.g., neuron biases or connection weights for NNs, or support vectors for SVMs

Pedagogical SKE

if the algorithm *does not need* to take into account any internal parameter, but it can extract symbolic knowledge by only relying on the predictor's outputs.



Overview

SKE methods: theory and practice

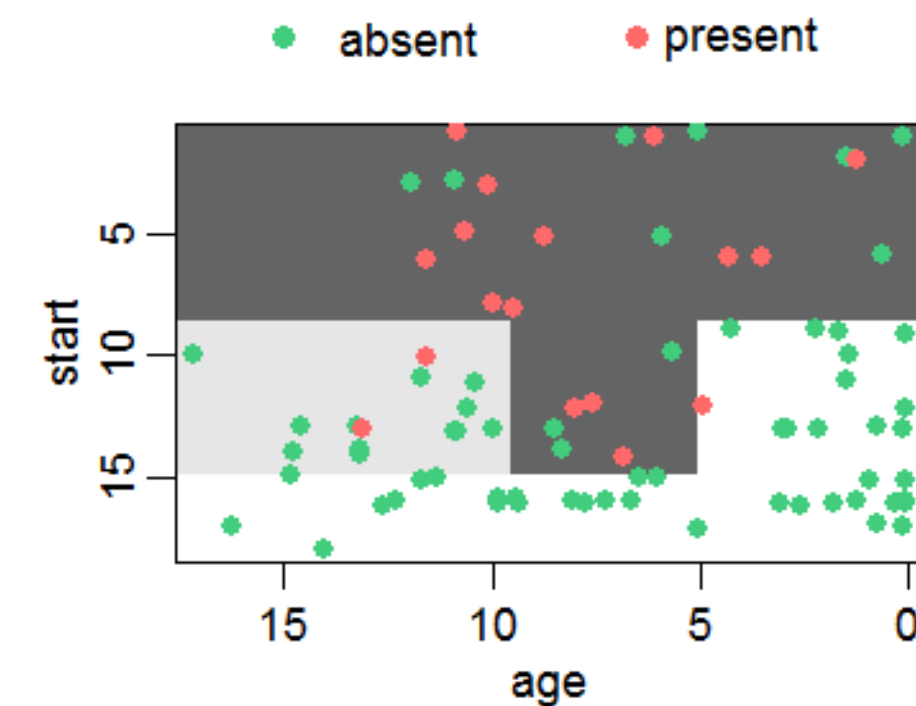
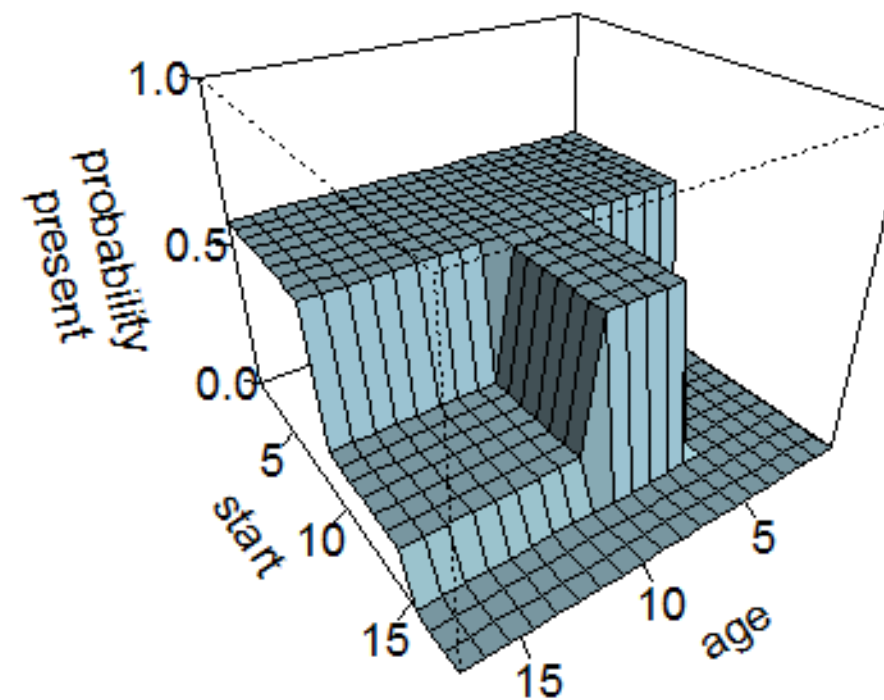
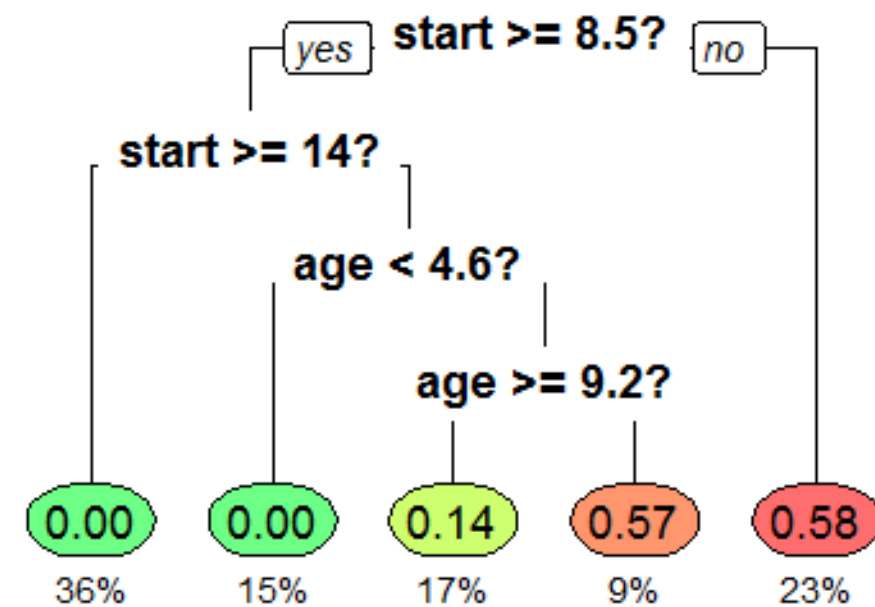


CART (pt. 1)

Classification and regression trees (cf. [Breiman et al., 1984](#))



CART (pt. 2)



An example decision tree estimating the probability of kyphosis after spinal surgery, given the *age* of the patient and the vertebra at which surgery was *start* ed (rf. [wiki:dt-learning](https://www.wiki-dt-learning.com/)). Notice that all decision trees subtend a partition of the input space, and that those trees themselves provide intelligible representations of *how* predictions are attained.

CART (pt. 3)

1. generate a *synthetic* dataset by using the predictions of the sub-symbolic predictor
2. *train* a decision tree on the synthetic dataset
3. compute the *fidelity* and repeat step 2 until satisfied
4. [optional] rewrite the tree as a set of symbolic *rules*



Practical example of SKE

The Adult dataset (cf. [Becker Barry and Kohavi Ronny, 1996](#))



Adult classification task

The Adult dataset contains the records (48,842) of individuals based on census data (this dataset is also known as Census Income). The dataset has many features (14) related to the individuals' demographics, such as age, education, and occupation. The target feature is whether the individual earns more than \$50,000 per year.

Examples of Adult records

age	workclass	education	...	hours-per-week	native-country	income
39	State-gov	Bachelors	...	40	United-States	<=50K
50	Self-emp-not-inc	Bachelors	...	13	United-States	<=50K
38	Private	HS-grad	...	40	United-States	<=50K
53	Private	11th	...	40	United-States	<=50K
28	Private	Bachelors	...	40	Cuba	<=50K
37	Private	Masters	...	40	United-States	<=50K
49	Private	9th	...	16	Jamaica	<=50K
52	Self-emp-not-inc	HS-grad	...	45	United-States	>50K
31	Private	Masters	...	50	United-States	>50K
42	Private	Bachelors	...	40	United-States	>50K



What we will do

1. Download the Adult dataset and preprocess it
2. Train a sub-symbolic predictor – a *neural network* – on the Adult dataset
3. Use a *pedagogical SKE method* – CART – to extract symbolic knowledge from the trained neural network
4. Visualise the extracted symbolic knowledge as a *decision tree* and as a set of *rules*

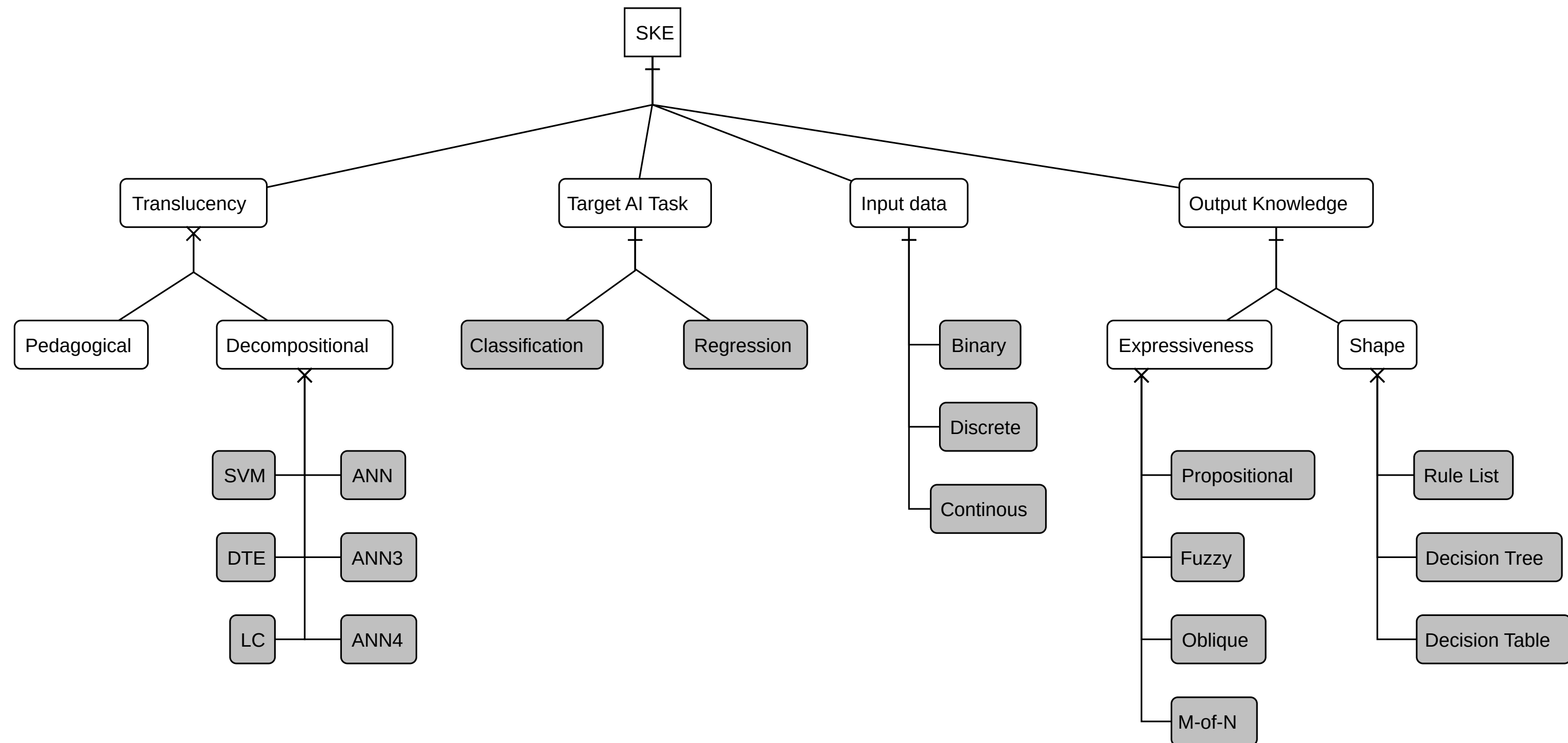


Jump to the code

GitHub repository at github.com/MatteoMagnini/demo-2025-woa-nesy/blob/master/notebook/extraction.ipynb



Taxonomy of SKE methods (pt. 1)



Taxonomy of SKE methods (pt. 2)

Target AI task

- *classification*
 $f : \mathcal{X} \subseteq \mathbb{R}^n \rightarrow \mathcal{Y} \text{ s.t. } |\mathcal{Y}| = k$
- *regression*
 $f : \mathcal{X} \subseteq \mathbb{R}^n \rightarrow \mathcal{Y} \subseteq \mathbb{R}^m$

Input data

- *binary*
 $\mathcal{X} \equiv 0, 1^n$
- *discrete*
 $\mathcal{X} \in x_1, \dots, x_{n^n}$
- *continuous*
 $\mathcal{X} \subseteq \mathbb{R}^n$

Taxonomy of SKE methods (pt. 3)

Shape

- *rule list*, ordered sequences of if-then-else rules
- *decision tree*, hierarchical set of if-then-else rules involving a comparison among a variable and a constant
- *decision table*, 2D tables summarising decisions for each possible assignment of the input variables

Expressiveness

- *propositional*, boolean statements + logic connectives, including arithmetic comparisons among variables and constants
- *fuzzy*, hierarchical set of if-then-else rules involving a comparison among a variable and a constant
- *oblique*, boolean statements + logic connectives + arithmetic comparisons
- *M-of-N*, any of the above + statements of the form “at least k of the following statements are true”



Discussion

Notable remarks

- discretisation of the input space
- discretisation of the output space
- features should have semantic meaning
- rules constitutes global explanations

Limitations

- tabular data as input, no images
- high dimensional datasets could lead to poorly readable rules
- high variable input spaces could do the same



Future research activities

- target images or highly dimensional data in general
- target reinforcement learning (when based on NN)
- target unsupervised learning
- design and prototype your own extraction algorithm



Symbolic Knowledge Injection (SKI)

How to inject symbolic knowledge into sub-symbolic predictors



Definition and Motivation (pt. 1)

Any algorithmic procedure affecting how sub-symbolic predictors draw their inferences in such a way that predictions are either *computed* as a function of, or *made consistent* with, some given symbolic knowledge.



Definition and Motivation (pt. 2)

- **Improve predictive performance:** by injecting symbolic knowledge, we can
 - *guide* the learning process in order to *penalise* inconsistencies with the symbolic knowledge, or
 - *structure* the model's architecture to *mimic* the symbolic knowledge
- **Enhance interpretability:** with SKI we can make predictors that are
 - interpretable by *transparent box design*, as they are built to mimic symbolic knowledge
 - interpretable using *symbols as constraints*, as they are built to respect symbolic knowledge
- **Robustness to data degradation:** symbolic knowledge can help sub-symbolic models maintain performance even in the presence of noisy or scarcity of data
- **Enhance fairness:** by incorporating symbolic knowledge about fairness constraints, we can ensure that sub-symbolic models make decisions that align with ethical considerations
- **And more:** SKI can simplify the predictor's architecture, in particular it can reduce the number of weights in a neural network, thus improving its efficiency and reducing the risk of overfitting



Concepts

Main entities and how to inject symbolic knowledge into sub-symbolic predictors

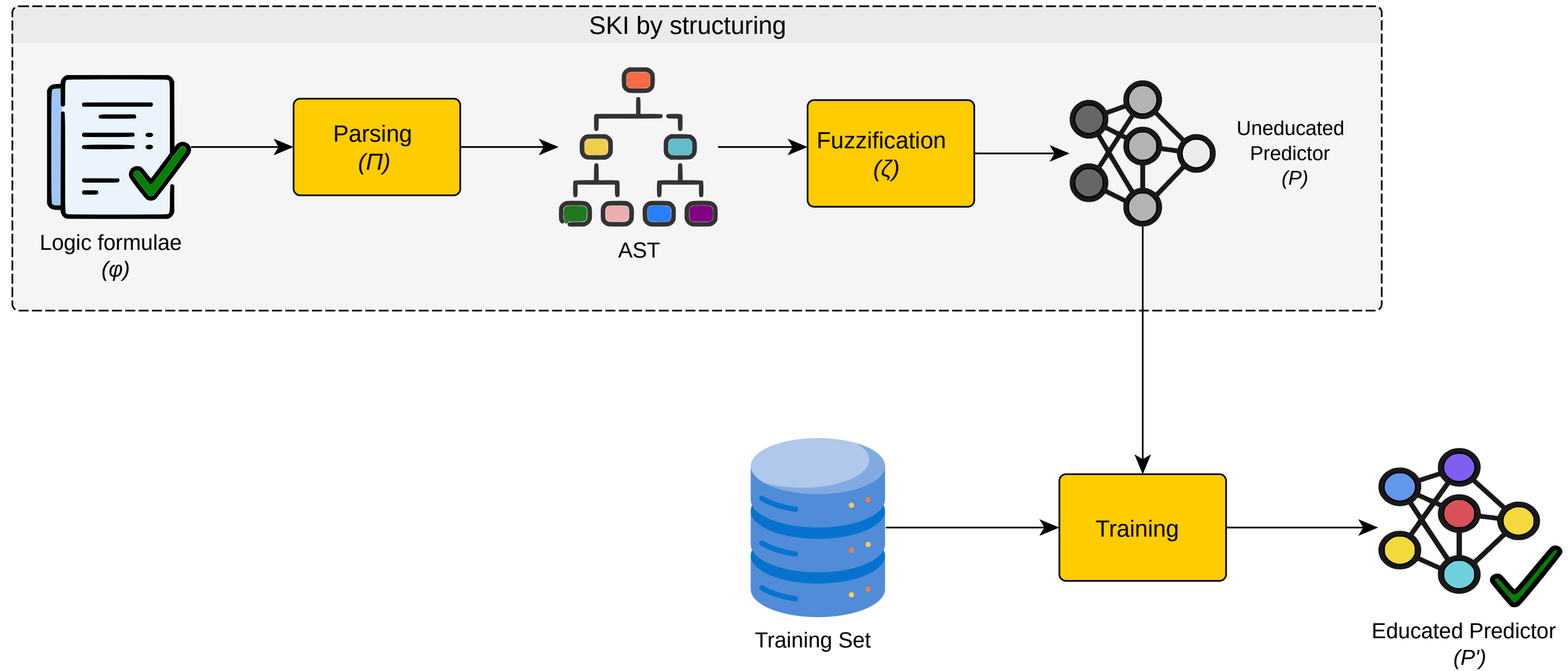


Entities

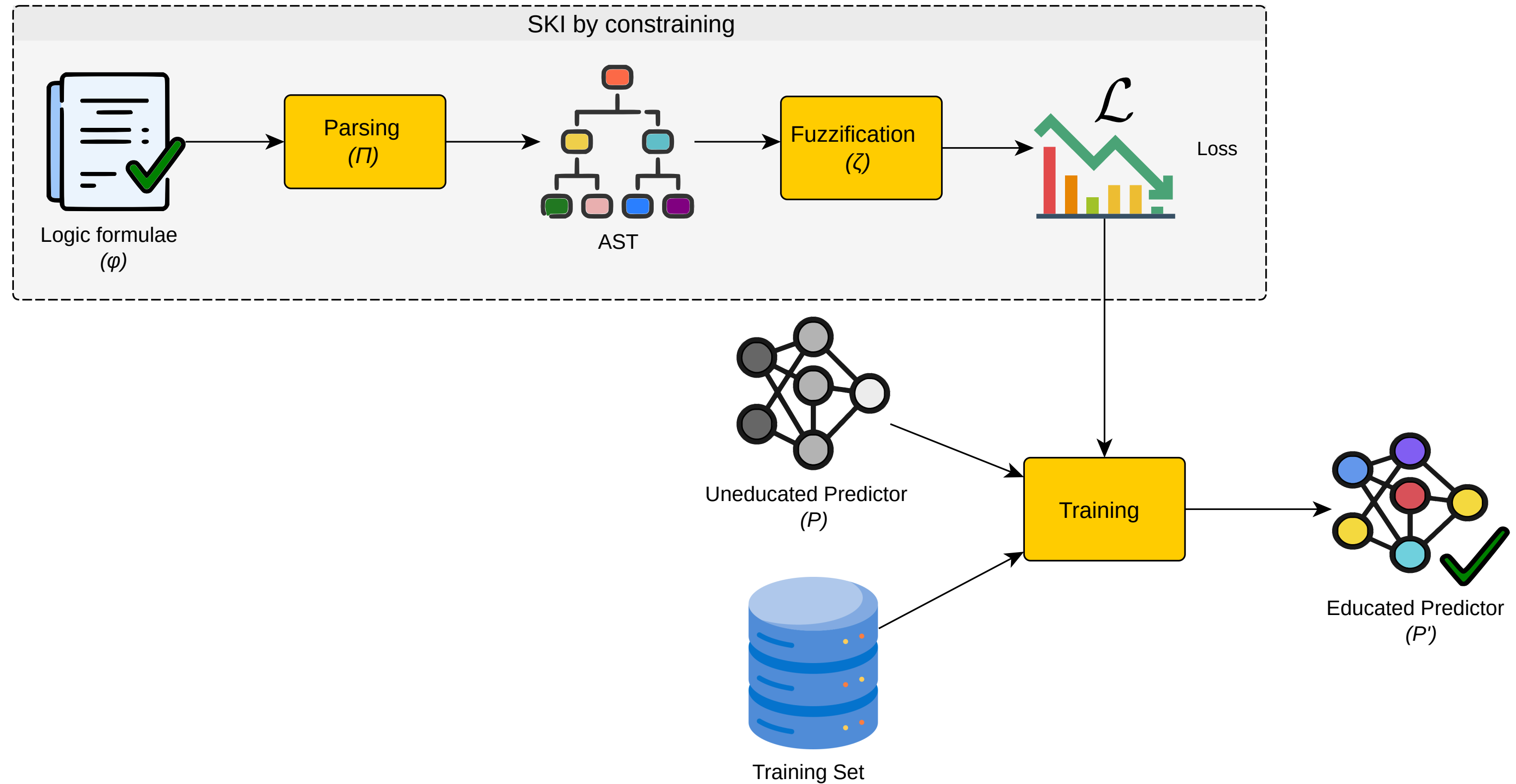
- **Predictor**: a sub-symbolic model that makes predictions based on input data, usually a neural network
- **Symbolic knowledge**: structured, formal knowledge that can be represented in a symbolic form. The most common forms of symbolic knowledge are
 - *Propositional logic*, simple rules with if-then structure
 - *Datalog*, a subset of first-order logic with no function symbols, only constants and variables
- **Fuzzification**: the process of converting symbolic knowledge into a form that can be used by sub-symbolic predictors, e.g. by assigning degrees of truth to symbolic statements
- **Injector**: the main component that injects symbolic knowledge into the predictor, by modifying its architecture, its training process or by other means



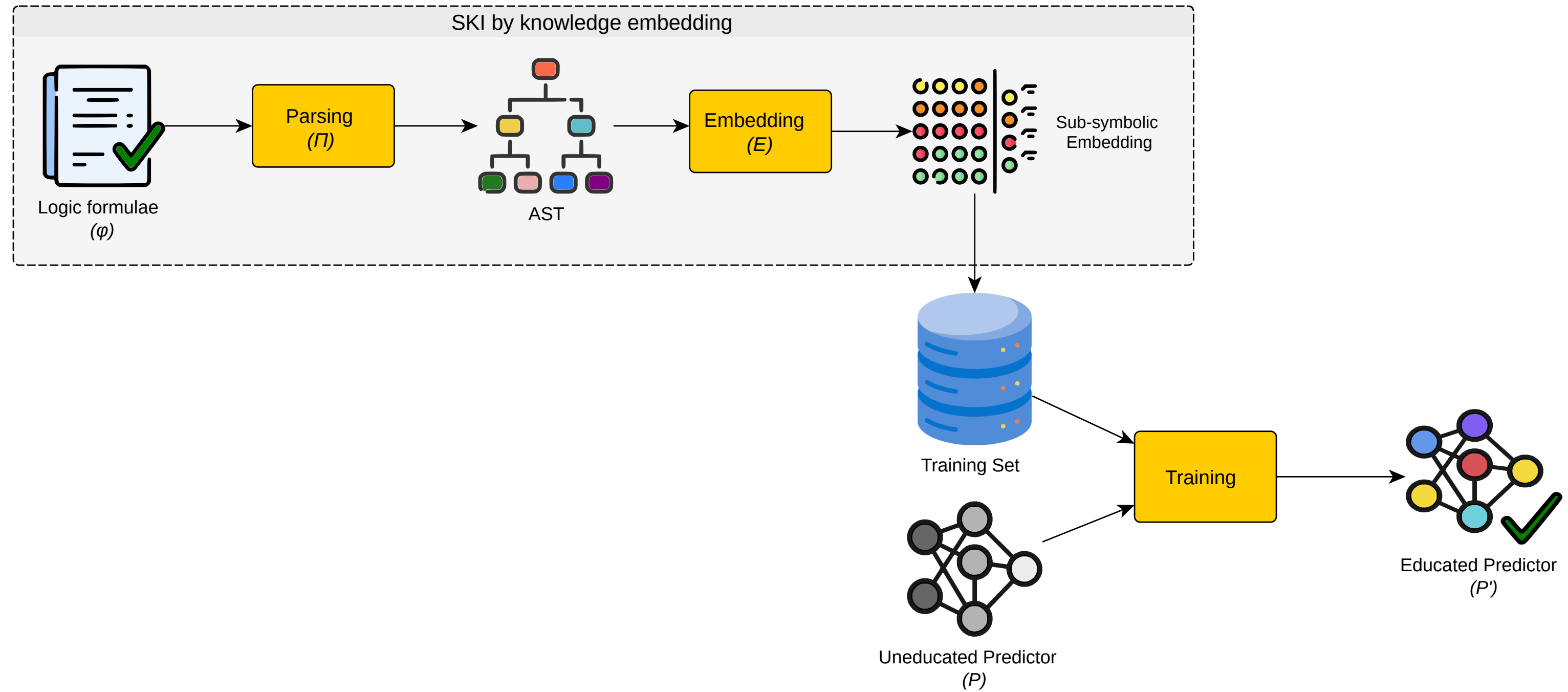
Structuring



Constraining



Embedding



Overview

SKI methods: theory and practice



Knowledge Injection via Network Structuring (KINS)

(ref. [Magnini et al., 2023](#))



Fuzzification

Formula	C. interpretation	Formula	C. interpretation
$[[\neg\phi]]$	$\eta(1 - [[\phi]])$	$[[\phi \leq \psi]]$	$\eta(1 + [[\psi]] - [[\phi]])$
$[[\phi \wedge \psi]]$	$\eta(\min([[\phi]], [[\psi]]))$	$[[class(\bar{X}, y_i) \leftarrow \psi]]$	$[[\psi]]^*$
$[[\phi \vee \psi]]$	$\eta(\max([[\phi]], [[\psi]]))$	$[[\text{expr}(\bar{X})]]$	$\text{expr}([[\bar{X}]])$
$[[\phi = \psi]]$	$\eta([[\neg(\phi \neq \psi)]])$	$[[\text{true}]]$	1
$[[\phi \neq \psi]]$	$\eta([[\phi]] - [[\psi]])$	$[[\text{false}]]$	0
$[[\phi > \psi]]$	$\eta(\max(0, \frac{1}{2} + [[\phi]] - [[\psi]]))$	$[[X]]$	x
$[[\phi \geq \psi]]$	$\eta(1 + [[\phi]] - [[\psi]])$	$[[k]]$	k
$[[\phi < \psi]]$	$\eta(\max(0, \frac{1}{2} + [[\psi]] - [[\phi]]))$	$[[p(\bar{X})]]^{**}$	$[[\psi_1 \vee \dots \vee \psi_k]]$

* encodes the value for the i^{th} output ** assuming p is defined by k clauses of the form:

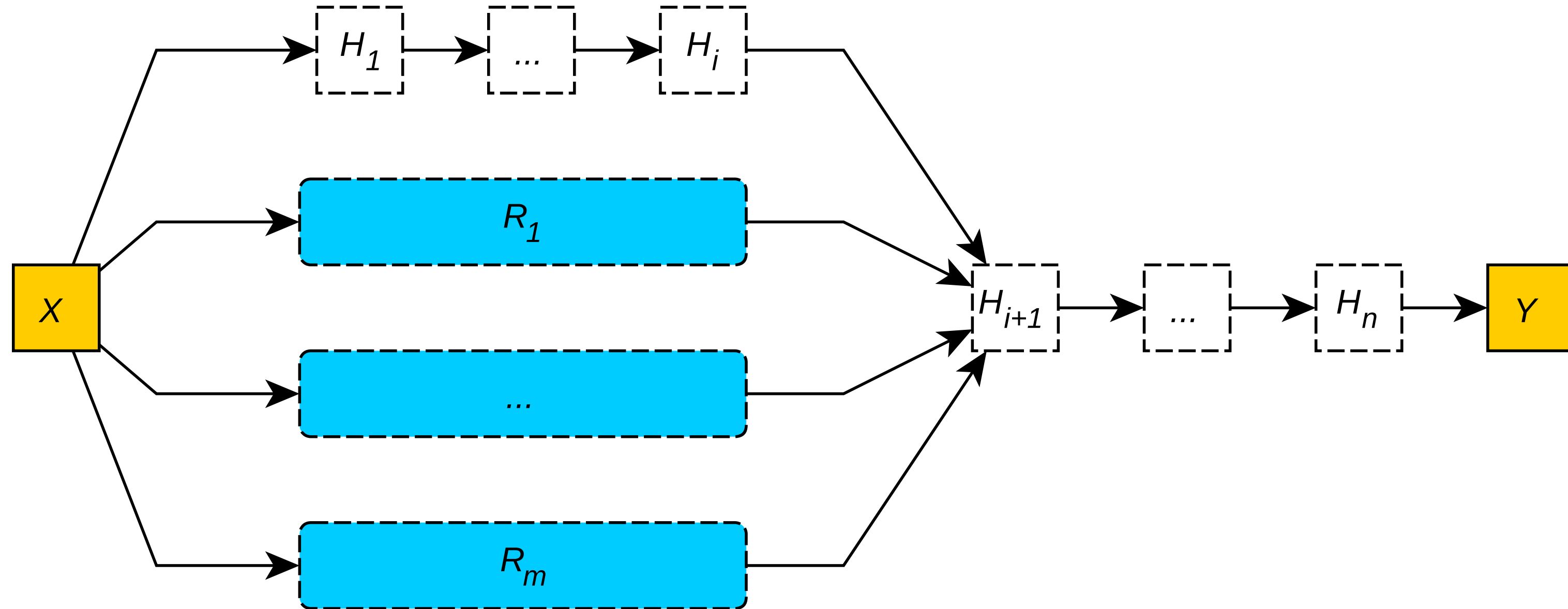
$$p(\bar{X}) \leftarrow \psi_1, \dots, p(\bar{X}) \leftarrow \psi_k$$



Injector (pt.1)



Injector (pt. 2)



Knowledge Injection via Lambda Layer (KILL)

(ref. [Magnini et al., 2022](#))



Fuzzification

Formula	C. interpretation	Formula	C. interpretation
$[[\neg \phi]]$	$\eta(1 - [[\phi]])$	$[[\phi \leq \psi]]$	$\eta([[\phi]] - [[\psi]])$
$[[\phi \wedge \psi]]$	$\eta(\max([[\phi]], [[\psi]]))$	$[\text{class}(\bar{X}, y_i) \leftarrow \psi]$	$[[\psi]]^*$
$[[\phi \vee \psi]]$	$\eta(\min([[\phi]], [[\psi]]))$	$[\text{expr}(\bar{X})]$	$\text{expr}([[\bar{X}]])$
$[[\phi = \psi]]$	$\eta([[\phi]] - [[\psi]])$	$[\text{true}]$	0
$[[\phi \neq \psi]]$	$[[\neg(\phi = \psi)]]$	$[\text{false}]$	1
$[[\phi > \psi]]$	$\eta(\frac{1}{2} - [[\phi]] + [[\psi]])$	$[[X]]$	x
$[[\phi \geq \psi]]$	$\eta([[\psi]] - [[\phi]])$	$[[k]]$	k
$[[\phi < \psi]]$	$\eta(\frac{1}{2} + [[\phi]] - [[\psi]])$	$[p(\bar{X})]**$	$[[\psi_1 \vee \dots \vee \psi_k]]$

* encodes the penalty for the i^{th} neuron ** assuming predicate p is defined by k clauses of the form:

$$p(\bar{X}) \leftarrow \psi_1, \dots, p(\bar{X}) \leftarrow \psi_k$$



Injector (pt.1)

Cost function: whenever the neural network wrongly predicts a class and violates the prior knowledge a cost proportional to the violation is added. In this way the output of the network differs more from the expected one and this affects the back propagation step.

$$Y' = f(Y, \text{cost})$$

$$f = Y \times (1 + \text{cost})$$

$$\text{cost}(X, Y) = \eta(p(X) - (1 - Y)) \quad (1 - Y \text{ because } 0 \text{ means true})$$



Injector (pt. 2)



Practical example of SKI

The poker hand data set (PHDS) (cf. [Cattral Robert and Oppacher Franz, 2002](#))



PHDS classification task

- Each record represents one poker hand
- 5 cards identified by 2 values: suit and rank
- Classes: 10
- Training set: 25,010
- Test set: 1,000,000

id	S1	R1	S2	R2	S3	R3	S4	R4	S5	R5	class
1	1	10	1	11	1	13	1	12	1	1	9
2	2	11	2	13	2	10	2	12	2	1	9
3	3	12	3	11	3	13	3	10	3	1	9
4	4	10	4	11	4	1	4	13	4	12	9
5	4	1	4	13	4	12	4	11	4	10	9
6	1	2	1	4	1	5	1	3	1	6	8
7	1	9	1	12	1	10	1	11	1	13	8
8	2	1	2	2	2	3	2	4	2	5	8
9	3	5	3	6	3	9	3	7	3	8	8
10	4	1	4	4	4	2	4	3	4	5	8
11	1	1	2	1	3	9	1	5	2	3	1
12	2	6	2	1	4	13	2	4	4	9	0
13	1	10	4	6	1	2	1	1	3	8	0
14	2	13	2	1	4	4	1	5	2	11	0
15	3	8	4	12	3	9	4	2	3	2	1



Logic rules to inject (pt. 1)

Class	Logic Formulation
Pair	$\text{class}(R_1, \dots, S_5, \text{pair}) \leftarrow \text{pair}(R_1, \dots, S_5)$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_1 = R_2$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_1 = R_3$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_1 = R_4$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_1 = R_5$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_2 = R_3$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_2 = R_4$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_2 = R_5$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_3 = R_4$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_3 = R_5$ $\text{pair}(R_1, \dots, S_5) \leftarrow R_4 = R_5$



Logic rules to inject (pt. 2)

Class	Logic Formulation
Two Pairs	$\text{class}(R_1, \dots, S_5, \text{two}) \leftarrow \text{two}(R_1, \dots, S_5)$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_3 = R_4$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_2 = R_4$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_2 = R_3$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_3 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_3 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_5 \wedge R_2 = R_3$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_4 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_2 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_5 \wedge R_2 = R_4$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_4 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_3 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_1 = R_5 \wedge R_3 = R_4$ $\text{two}(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_4 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_2 = R_4 \wedge R_3 = R_5$ $\text{two}(R_1, \dots, S_5) \leftarrow R_2 = R_5 \wedge R_3 = R_4$



Logic rules to inject (pt. 3)

Class	Logic Formulation
Three of a Kind	$\text{class}(R_1, \dots, S_5, \text{three}) \leftarrow \text{three}(R_1, \dots, S_5)$ $\text{three}(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_3$ $\text{three}(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_4$ $\text{three}(R_1, \dots, S_5) \leftarrow R_1 = R_2 \wedge R_1 = R_5$ $\text{three}(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_4$ $\text{three}(R_1, \dots, S_5) \leftarrow R_1 = R_3 \wedge R_1 = R_5$ $\text{three}(R_1, \dots, S_5) \leftarrow R_1 = R_4 \wedge R_1 = R_5$ $\text{three}(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_4$ $\text{three}(R_1, \dots, S_5) \leftarrow R_2 = R_3 \wedge R_2 = R_5$ $\text{three}(R_1, \dots, S_5) \leftarrow R_2 = R_4 \wedge R_2 = R_5$ $\text{three}(R_1, \dots, S_5) \leftarrow R_3 = R_4 \wedge R_3 = R_5$
Flush	$\text{class}(R_1, \dots, S_5, \text{flush}) \leftarrow \text{flush}(R_1, \dots, S_5)$ $\text{flush}(R_1, \dots, S_5) \leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5$



What we will do

1. Download the PHDS dataset and preprocess it
2. Define the *symbolic knowledge* to inject
3. Train a sub-symbolic predictor – a *neural network* – on the PHDS dataset
4. Train a second neural network with the symbolic knowledge injected
5. We will inject the knowledge in the *loss function*
6. Visualise and compare the results of the two predictors

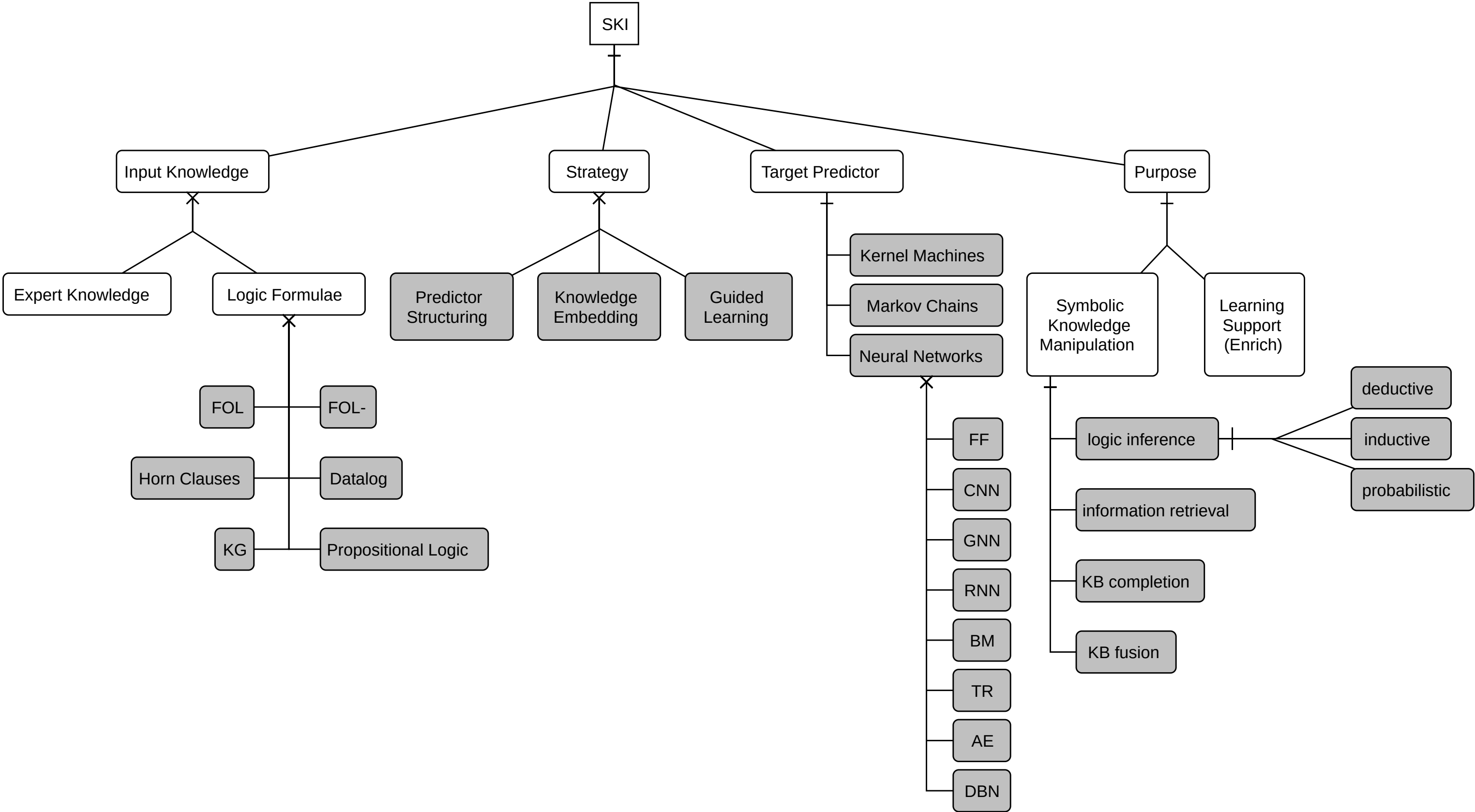


Jump to the code!

GitHub repository at github.com/MatteoMagnini/demo-2025-woa-nesy/blob/master/notebook/injection.ipynb



Taxonomy of SKI methods (pt. 1)



Taxonomy of SKI methods (pt. 2)

- **input knowledge**: how is the knowledge to-be-injected represented?
 - commonly, some sub-set of first-order logic (FOL)
- **target predictor**: which predictors can knowledge be injected into?
 - mostly, neural networks
- **strategy**: how does injection actually work?
 - *guided learning*: the input knowledge is used to *guide the training* process
 - *structuring*: the *internal* composition of the predictor is *(re-)structured* to reflect the input knowledge
 - *embedding*: the input knowledge is *converted* into numeric array form
- **purpose**: why is knowledge injected in the first place?
 - *knowledge manipulation*: improve / extend / reason about symbol knowledge—subsymbolically
 - *learning support*: improve the sub-symbolic predictor (e.g. speed, size, etc.)



Discussion

Notable remarks

- Knowledge should express relations about input-output pairs
- embedding implies *extensional* representation of knowledge
- guided learning and structuring support *intensional* knowledge
- propositional knowledge implies binarising the I/O space

Limitations

- Recursive data structures are natively not supported
- extensional representation cost storage
- guided learning works poorly with lacking data



Future research activities

- *foundational*
address recursion
- *practical*
find a language that is a good balance between expressiveness and ease of use
- *target*
apply to large language models



Let's keep in touch!

The talk is over, I hope you enjoyed it



 Unibo email

 GitHub

 LinkedIn

 Research group

