Symbolic Learning and Rule Extraction with Sole.jl

Mauro Milella Alberto Paparella Riccardo Pasini Marco Perrotta

University of Ferrara, Department of Mathematics and Computer Science

Applied Computational Logic and Artificial Intelligence Laboratory

2 October 2025

Where to find these slides

Who are we?

Figure: Dept. of Mathematics and Computer Science, University of Ferrara.

Who are we?

Figure: Applied Computational Logic and Artificial intelligence Lab.

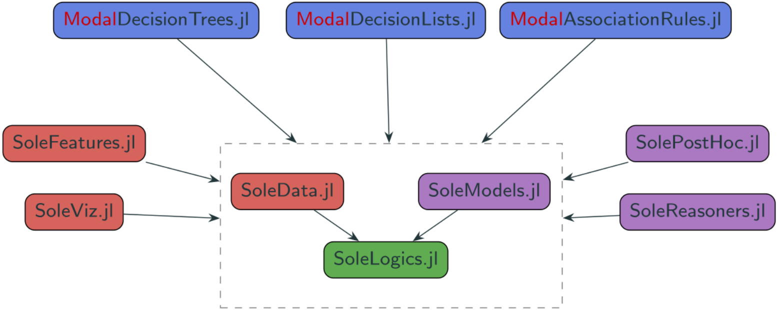


Figure: A general overview of the Sole.jl framework.

Figure: Third Millennium Symbolic Learning with Sole.jl, JuliaCon 2023.

Sole.jl at JuliaCon

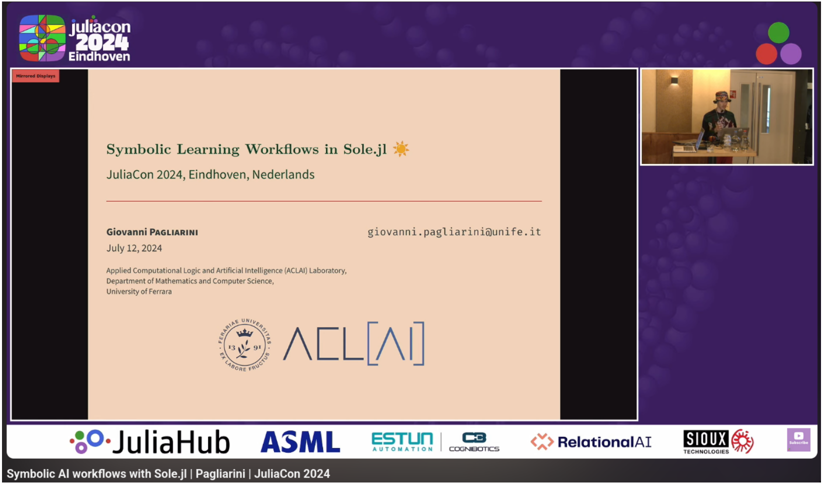


Figure: Symbolic AI workflows with Sole.jl, JuliaCon 2024.

Today's talk

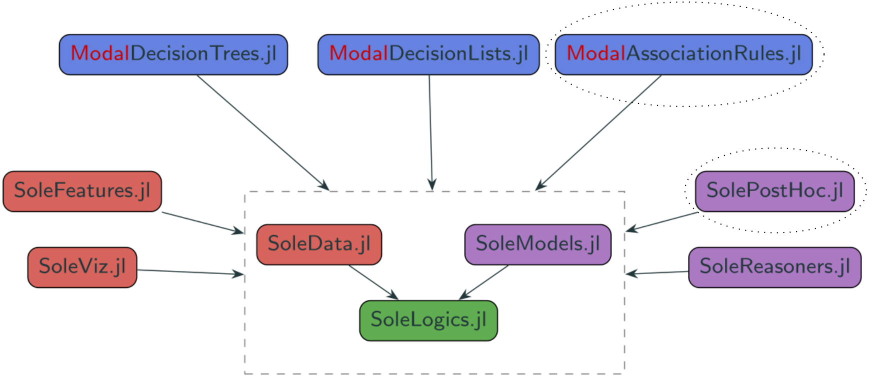


Figure: Today we will focus on ModalAssociationRules.jl and SolePostHoc.jl.

ModalAssociationRules.jl

ModalAssociationRules.jl

What:

- find frequent patterns hidden in relational data;
- combine them to generate interesting rules.

ModalAssociationRules.jl

What:

- find frequent patterns hidden in relational data;
- combine them to generate interesting rules.

Why:

- describe non-trivial aspects of data.

ModalAssociationRules.jl

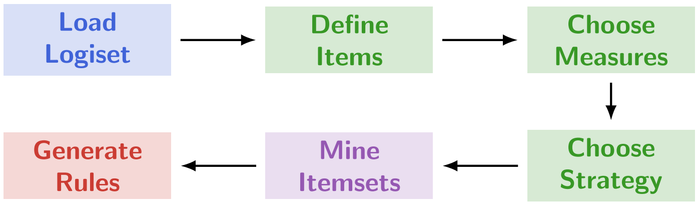
What:

- find frequent patterns hidden in relational data;
- combine them to generate interesting rules.

Why:

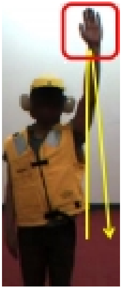
- describe non-trivial aspects of data.

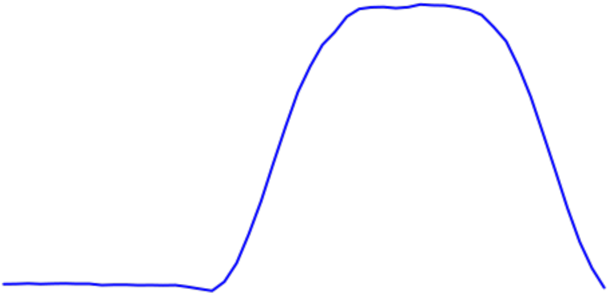
How:



ModalAssociationRules.jl

Let us mine association rules about a specific kind of body movement.

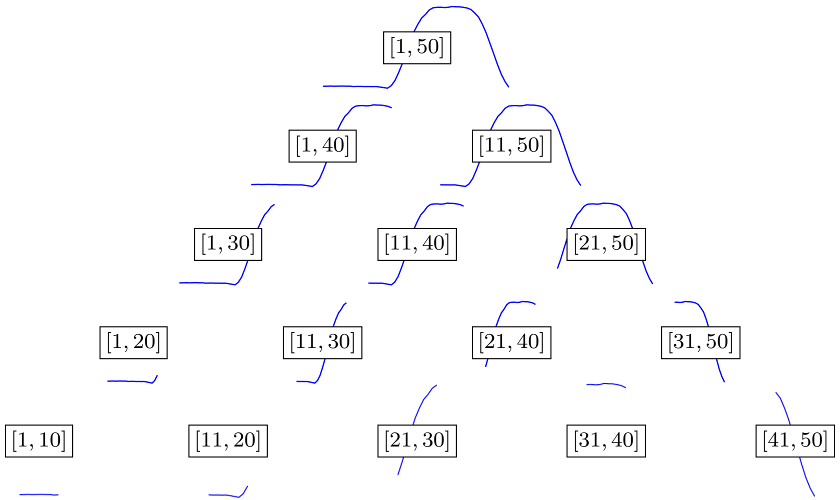




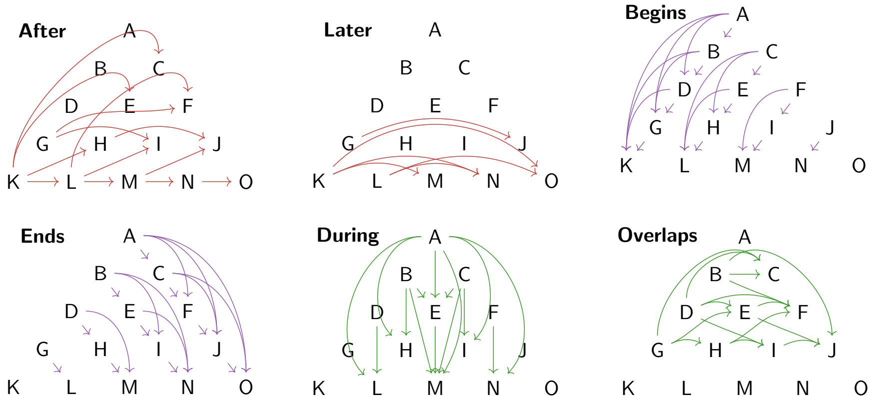
We *dissect* the signal into many intervals, called **worlds**.

A collection of worlds and the relations between them is called **logiset**.

ModalAssociationRules.jl - Logiset Instance



ModalAssociationRules.jl - Logiset Instance



We want to probe data with **items** that can be true or false on each world.

Load Logiset → **Define Items** → Choose Measures

↓

Generate Rules ← Mine Itemsets ← Choose Strategy

p $\coloneqq$ (distance(, mysignal) $\leq$, 1.0) $\triangleright$ Here

r $\coloneqq$ (distance(, mysignal) $\leq$, 3.5) $\triangleright$ After

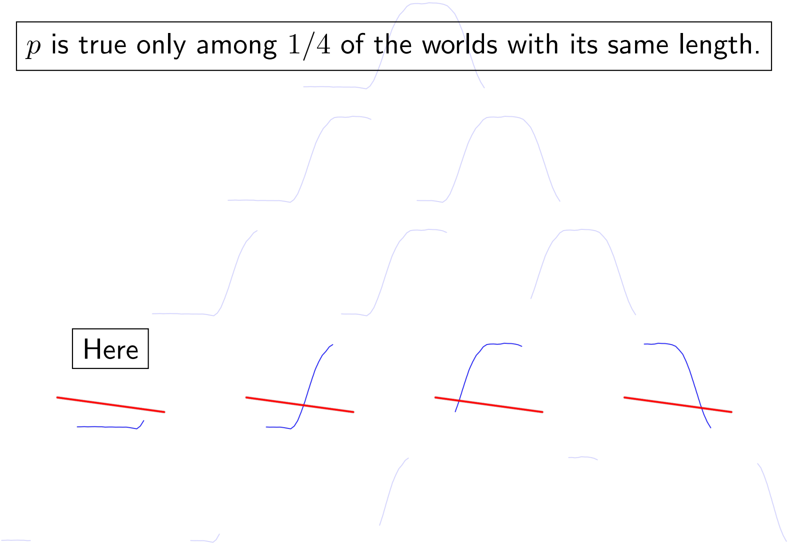
q $\coloneqq$ (distance(, mysignal), $\leq$, 1.7) $\triangleright$ After $\triangleright$ Begins

We need measures to establish if (a set of) items is interesting.

The **support** of an item is its probability to be true on a given world.

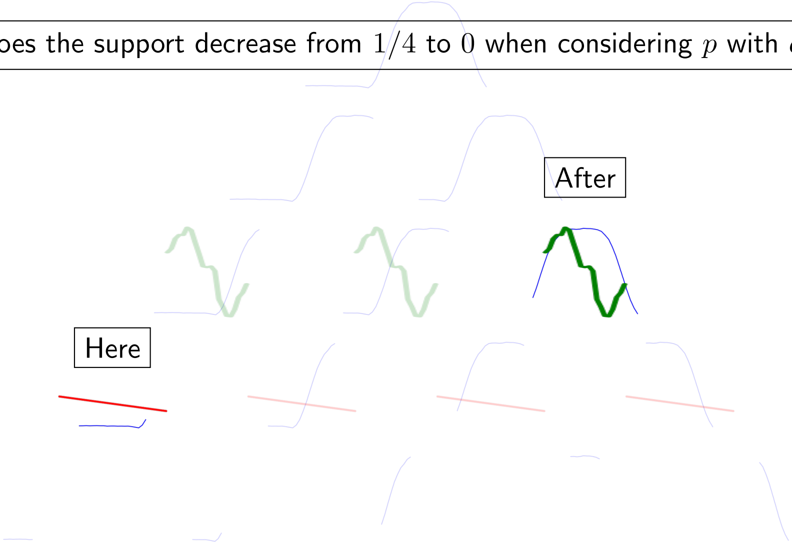
There are many measures (e.g., **confidence**) to assess whether two set of items P and Q might be arranged in an association rule $P \Rightarrow Q$.

p is true only among $1/4$ of the worlds with its same length.

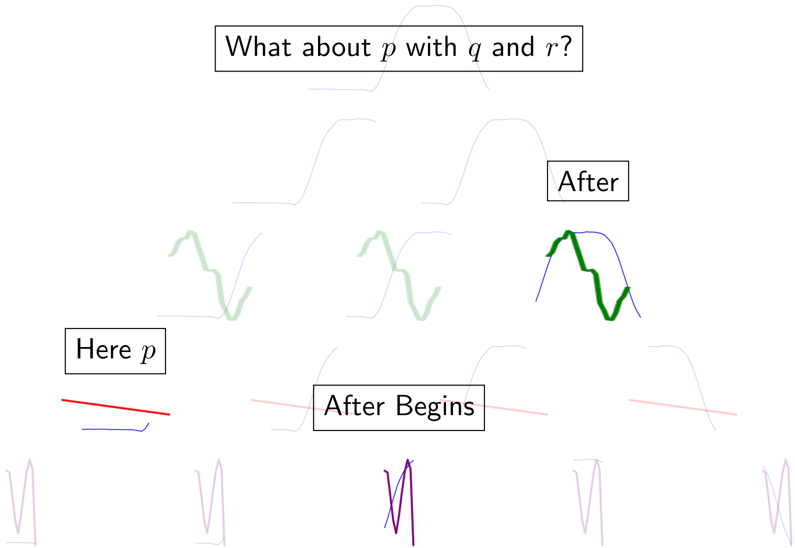


Support of $\{p, q\}$

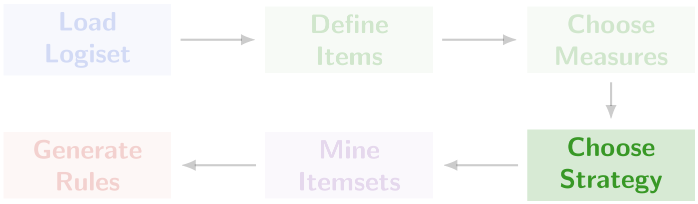
Does the support decrease from $1/4$ to 0 when considering p with q?



Support of $\{p, q, r\}$



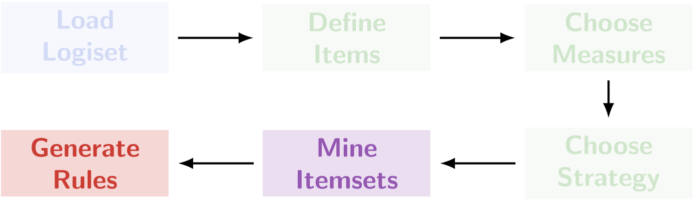
We need a **search strategy** for exploring the space of all the possible set of items ($\{p\}$, $\{q\}$, $\{r\}$, $\{s\}$, $\{p, q\}$, $\{p, r\}$, $\{p, q, r\}$, $\ldots$).



The simplest one, **Apriori**, performs a BFS over the search space.

ModalAssociationRules.jl - Mining and Rules Generation

We are ready to **mine** all the frequent set of items, and combine them into **association rules**.



Both these processes can be constrained with custom **policies**.

```
X = MyDataset |> load |> scalarlogiset

# every item applies the distance between a chunk of data
# coming from a specific variable and a tensor
items = [
    (1, [your floats here]; distance=euclidean) |> Item
    (2, [...]) |> After |> Item
    (3, [...]) |> After |> Begins |> Item
]

# given a fact X, p(X) must be >= 0.1
itemsetmeasures = [(support, 0.1)]

rulemeasures = [ # given a rule X => Y...
    (gconfidence, 0.7) # we want prob(Y|X) >= 0.7
    (lift, 1.3) # X and Y must not be independent
]

# mine the interesting facts, then stream all the rules out
miner |> mine! |> generaterules!
```

SolePostHoc.jl

Overview

Many methods exist to **extract logical rules** from learning models. However, there is currently no unified framework that brings together all these known and modern algorithms under a single approach.

That's exactly what this package aims to provide!

Integrated Algorithms

SolePostHoc.jl integrates a wide range of algorithms for knowledge extraction, including:

- **Surrogate Trees** — *e.g.* Trepan, Refene, Batrees
- **Knowledge Distillation** — *e.g.* RuleCosi+
- **Rule Extraction** — *e.g.* Lumen, Intrees

Integrated Algorithms

SolePostHoc.jl integrates a wide range of algorithms for knowledge extraction, including:

- **Surrogate Trees** — *e.g.* Trepan, Refene, Batrees
- **Knowledge Distillation** — *e.g.* RuleCosi+
- **Rule Extraction** — *e.g.* Lumen, Intrees

Integrated Algorithms

SolePostHoc.jl integrates a wide range of algorithms for knowledge extraction, including:

- **Surrogate Trees** — *e.g.* Trepan, Refene, Batrees
- **Knowledge Distillation** — *e.g.* RuleCosi+
- **Rule Extraction** — *e.g.* Lumen, Intrees

Integrated Algorithms

SolePostHoc.jl integrates a wide range of algorithms for knowledge extraction, including:

- **Surrogate Trees** — *e.g.* Trepan, Refene, Batrees
- **Knowledge Distillation** — *e.g.* RuleCosi+
- **Rule Extraction** — *e.g.* Lumen, Intrees

Key Innovation

This package's primary purpose is to provide a **uniform interface** for knowledge extraction algorithms, enabling the **comparison** of different post-hoc interpretation methods while maintaining a **coherent and intuitive** user experience.

Practical Example

Consider a machine learning model trained on a generic dataset. For example, let us consider a **Random Forest Classifier** learned on the **Iris dataset** to classify 3 different species of flowers.

We are interested in extracting interpretable rules that explain the model's decision process. **SolePostHoc.jl** offers **two primary methods** for accomplishing this task.

Implementation Approaches

The first approach is to directly call the specific algorithm function. For example:

```
# Extract rules using the LUMEN algorithm directly
extracted_rules = lumen(model, args...)
```

```
# Extract rules using the Intrees algorithm directly
extracted_rules = intrees(model, x_train, y_train, args...)
```

Notice that this approach returns the output in the **original format** defined by the specific algorithm, which may differ significantly between different methods.

Implementation Approaches

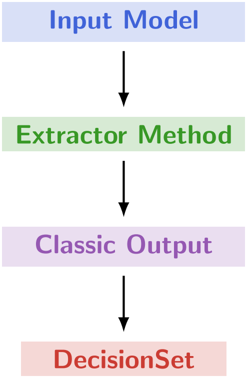
The second approach uses the unified interface through rule extractors:

```
# Extract rules using the unified interface
extractor = LumenRuleExtractor()
decision_set = modalextractrules(extractor, model, args...)
```

```
# Extract rules using the unified interface
extractor = IntreesRuleExtractor()
decision_set = modalextractrules(extractor, model, x_train,
                                          y_train, args...)
```

The key advantage of the second approach is that it not only executes the original algorithm (equivalent to calling `algo(...)` directly) but also converts the output into a **DecisionSet**.

Rule Extraction Process



The unified interface transforms any extractor's classic output into a standardized **DecisionSet** format for consistent interpretation and comparison across different algorithms.

DecisionSet Results

A **DecisionSet** is a vector of propositional logical rules in **Disjunctive Normal Form (DNF)**, with one rule per class/label.

Consider a trained model that classifies flower species.
Using **SolePostHoc.jl**, we might extract the following decision set:

```
Class "Iris-setosa":
    IF (SepalLengthCm < -0.5) AND (SepalWidthCm < 8.2)
    THEN predict "Iris-setosa"


Class "Iris-versicolor":
    IF (SepalLengthCm > 0.5) AND (SepalWidthCm < 3.25)
    THEN predict "Iris-versicolor"


Class "Iris-virginica":
    IF (PetalWidthCm > 2.0)
    THEN predict "Iris-virginica"
```

SoleXplorer.jl

Overview

SoleXplorer.jl: a simple, yet powerful, interactive machine learning framework.

- **Automated Setup**, designed to minimize user effort, with sensible defaults for all parameters;
- **Logics-based**, provides seamless integration with ModalAssociationRules and SolePostHoc packages;
- **Time-series Analysis** built-in support for temporal data, including dataset windowing and temporal feature extraction;
- **GUI-Ready**, designed for upcoming graphical user interface integration.

Overview

SoleXplorer.jl: a simple, yet powerful, interactive machine learning framework.

- **Automated Setup**, designed to minimize user effort, with sensible defaults for all parameters;
- **Logics-based**, provides seamless integration with ModalAssociationRules and SolePostHoc packages;
- **Time-series Analysis** built-in support for temporal data, including dataset windowing and temporal feature extraction;
- **GUI-Ready**, designed for upcoming graphical user interface integration.

Overview

SoleXplorer.jl: a simple, yet powerful, interactive machine learning framework.

- **Automated Setup**, designed to minimize user effort, with sensible defaults for all parameters;
- **Logics-based**, provides seamless integration with ModalAssociationRules and SolePostHoc packages;
- **Time-series Analysis** built-in support for temporal data, including dataset windowing and temporal feature extraction;
- **GUI-Ready**, designed for upcoming graphical user interface integration.

Overview

SoleXplorer.jl: a simple, yet powerful, interactive machine learning framework.

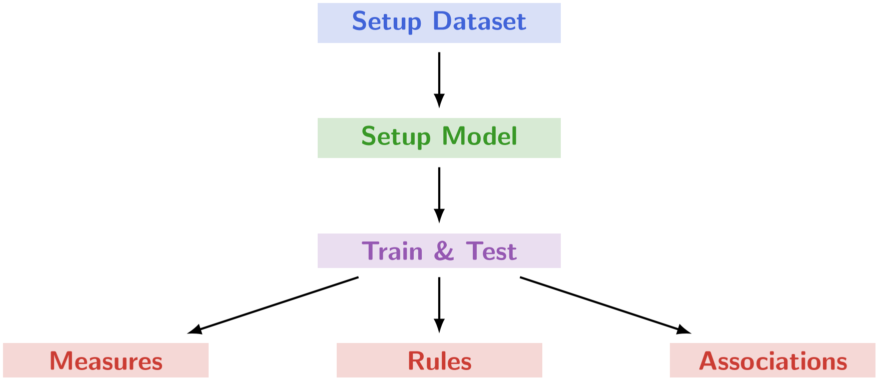
- **Automated Setup**, designed to minimize user effort, with sensible defaults for all parameters;
- **Logics-based**, provides seamless integration with ModalAssociationRules and SolePostHoc packages;
- **Time-series Analysis** built-in support for temporal data, including dataset windowing and temporal feature extraction;
- **GUI-Ready**, designed for upcoming graphical user interface integration.

Overview

SoleXplorer.jl: a simple, yet powerful, interactive machine learning framework.

- **Automated Setup**, designed to minimize user effort, with sensible defaults for all parameters;
- **Logics-based**, provides seamless integration with ModalAssociationRules and SolePostHoc packages;
- **Time-series Analysis** built-in support for temporal data, including dataset windowing and temporal feature extraction;
- **GUI-Ready**, designed for upcoming graphical user interface integration.

Pipeline



This is the complete pipeline of SoleXplorer.jl, encapsulated in a single function call: `symbolic_analysis()`.

Usage Example: Bronze

We have a dataset composed of a matrix (or dataframe) of measures, and a vector of labels, and we don't know if there's something interesting there, let's check it out...

```
using SoleXplorer, MLJ, JLD2

data_path = joinpath(@__DIR__, "respiratory_pneumonia.jld2")
data  = JLD2.load(data_path)
X = data["X"]
y = MLJ.CategoricalArray{String,1,UInt32}(data["y"])

model = symbolic_analysis(X, y, seed=123);
show_measures(model)
```

Usage Example: Silver

I would like to test various models to see which one suits for my experiment...

```
model = symbolic_analysis(X, y; model=ModalRandomForest(),
    seed=123);
show_measures(model)

model = symbolic_analysis(X, y; model=XGBoostClassifier(),
    seed=123);
show_measures(model)
```

Usage Example: Gold

Now it's time to tweak hyperparameters too find the best setting for the chosen model...

```
range = SoleXplorer.range(:max_depth; lower=1, upper=10)
model = symbolic_analysis(
    X, y;
    model=XGBoostClassifier(),
    seed=123,
    resampling=CV(nfolds=5, shuffle=true),
    tuning=GridTuning(resolution=5, resampling=CV(nfolds=5),
        range=range, measure=accuracy, repeats=5),
    measures=(accuracy, log_loss, confusion_matrix, kappa)
)
show_measures(model)
```

Usage Example: Platinum

It would be nice to dig into rule extraction...

```
range = (
    SoleXplorer.range(:max_depth; lower=1, upper=3),
    SoleXplorer.range(:num_round; lower=1, upper=10))
model = symbolic_analysis(
    X, y;
    model=XGBoostClassifier(),
    seed=123,
    tuning=AdaptiveTuning(range=range, resampling=CV(nfolds
        =5), measure=accuracy, repeats=10),
    extractor=LumenRuleExtractor()
)
```

Extracted Rules:

- `(V3 < 0.0084) && (V2 >= 0.0238) && (V4 >= 0.0031) -> healthy`
- `(V3 >= 0.0087) && (V5 < 0.0045) -> pneumonia`

Usage Example: Diamond

We've selected some rules that sound interesting. Finally, it would be nice to see if there are some associations among them...

```
manual_p = Atom(ScalarCondition(VariableMin(3), >=, 0.0087))
manual_q = Atom(ScalarCondition(VariableMin(5), <, 0.0045))
manual_r = Atom(ScalarCondition(VariableMax(4), <, 0.0031))

symbolic_analysis!(
    model,
    association=FPGrowth(
        Vector{Item}([manual_p, manual_q, manual_r]),
        [(gsupport, 0.1, 0.1)],
        [(gconfidence, 0.2, 0.2)],))
associations(model)
```

- `min[V3] >= 0.0087 => min[V5] < 0.0045`
- `min[V5] < 0.0045 => min[V3] >= 0.0087`
- `min[V5] < 0.0045 => max[V4] < 0.0031`
- `max[V4] < 0.0031 => min[V5] < 0.0045`

Conclusion

SoleXplorer.jl provides a complete pipeline for symbolic machine learning:

- **Measures**: various performance and interpretability metrics, such as accuracy, fidelity, and complexity.
- **Rules**: interpretable rules extracted from the model using `SolePostHoc.jl`.
- **Associations**: association rules mined from the dataset using `ModalAssociationRules.jl`.

Thank you for your attention!

Questions?