Stochastic Answer Set Programming

A Research Program

Francisco Coelho

NOVA LINCS
High Performance Computing Chair
Departamento de Informática, Universidade de Évora

December 11, 2023

This is a join work with **Salvador Abreu**@DInf and **Bruno Dinis**@DMat.

In Short

- About **Machine Learning**:
  - Vector or matrix based models lack “structure”.
  - Large models don’t *explain* data.
- About **Logic Programs**:
  - Logic programs formalize knowledge.
  - Logic doesn’t *capture* uncertainty and is *fragile* to noise.
- **Probabilistic Logic Programs** extend formal knowledge with probabilities.
  - How to propagate probabilities through rules?

Goal: Combine Logic and Statistics.

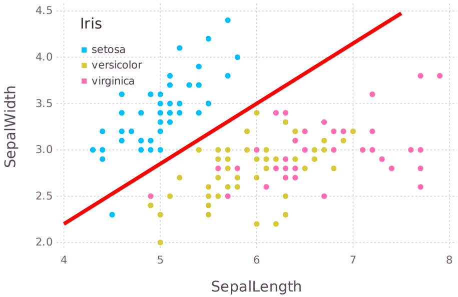
Machine Learning

- Standard Example — Iris Classification
- Assumptions of Machine Learning
- Where Machine Learning Fails

The Standard Example — Iris Classification

Learning Functions: The famous Iris database

- x_1 sepal length.
- x_2 sepal width.
- x_3 petal length.
- x_4 petal width.
- y species (one of *setosa*, *versicolor*, *virginica*).



- A *setosa* model: $-0.40 - 0.65x_1 + 1.00x_2 > 0.00$.
- A general **model template**:

$$f_\theta(\vec{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 > 0$$

Assumptions of Machine Learning

- Each instance is described in a **single row** by a **fixed set of features**

$$\begin{array}{cccc|c} \mathbf{x_1} & \mathbf{x_2} & \dots & \mathbf{x_n} & \mathbf{y} \\ \hline x_{11} & x_{21} & \dots & x_{n1} & y_1 \\ & & \vdots & & \\ x_{1m} & x_{2m} & \dots & x_{nm} & y_m \end{array}.$$

- Instances are **independent** of one another, **given the model**

$$y = f_\theta(\vec{x}).$$

- Parameters **minimize estimation error** *e.g.*

$$\hat{\theta} = \arg\min_\theta \sum_i \left\| y_i - f_\theta(\vec{x_i}) \right\|.$$

Failing Assumptions 1/2

student	course	grade
s_1	c_1	a
s_2	c_1	c
s_1	c_2	b
s_2	c_3	b
s_3	c_2	b
s_4	c_3	b
s_3	c_4	X
s_4	c_4	Y

Infer $X \succ_g Y$.

"*Proof.*"

$s_1 \succ_s s_2$ (rows 1 & 2)
so $c_2 \succ_c c_3$ (3 & 4).

Then $s_3 \succ_s s_4$ (5 & 6)
therefore $X \succ_g Y$ $\square$.

- Instances are not independent.
- Small dataset, so *no statistics*.

Failing Assumptions 2/2

Prompt: Given `f(machine) = e`, `f(learning) = g` and `f(algorithm) = m`, what is the value of `f(ksadbf)`?

input	output
machine	e
learning	g
algorithm	m
ksadbf	?

Bard: `f(ksadbf) = 11.` `We can see that the value of f(x) is simply the first letter of x minus 96.`

ChatGPT: `The information provided seems to define a function` f `where certain inputs are associated with specific outputs. Based on the given information:`

- $f(\texttt{machine}) = e$
- $f(\texttt{learning}) = g$
- $f(\texttt{algorithm}) = m$

`However, there is no information given about` $f(\texttt{ksadbf})$`. Without additional information or a specific rule that determines the output for` $f(\texttt{ksadbf})$`,` *`it is not possible to determine its value`*`. If there is a pattern or rule for how the outputs are assigned based on the inputs, you would need to provide that information for me to help determine the value of` $f(\texttt{ksadbf})$`.`

No help from state-of-the-art Natural Language Processing or LLMs.

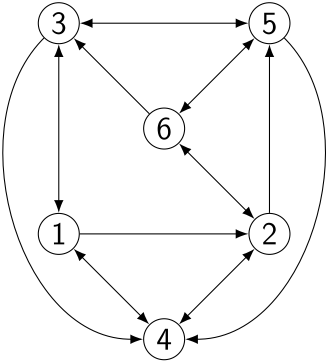
Where Machine Learning Fails

- No *interpretability*: large models ($\dim \theta \sim$ millions) are not understandable by humans.
- Need for *large number of training* examples.
- Unfit for *multiple relations*, *intersample dependencies* and *variable features*: The *Students, Courses and Results* and *Last Letter* examples.
- Poor *generalization*: A model trained in the iris dataset is useless in any other domain.
- No use of background knowledge.

Logic Programming

- An Example of Logic Programming.
- Inductive Logic Programming.
- Where ILP Fails.

An Example of Logic Programming




```
node(1..6).

edge(1,2). edge(2,4). edge(3,1).
edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4).
edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5).
edge(5,6). edge(6,5).

col(r). col(b). col(g).

1 { color(X,C) : col(C) } 1 :- node(X).
:- edge(X,Y), color(X,C), color(Y,C).

#show color/2.
```

```
color(2,b) color(1,g) color(4,r) color(3,b) color(5,g) color(6,r)
color(1,r) color(2,b) color(4,g) color(3,b) color(5,r) color(6,g)
color(1,r) color(2,g) color(4,b) color(3,g) color(5,r) color(6,b)
color(1,b) color(2,g) color(4,r) color(3,g) color(5,b) color(6,r)
color(2,r) color(1,g) color(4,b) color(3,r) color(5,g) color(6,b)
color(2,r) color(1,b) color(4,g) color(3,r) color(5,b) color(6,g)
```

Inductive Logic Programming

Learning Logic Programs from Examples.

Generate rules that...

- use **background knowledge**

$$\begin{array}{ll} parent(john, mary), & parent(david, steve), \\ parent(kathy, mary), & female(kathy), \\ male(john), & male(david). \end{array}$$

- to entail all the **positive examples**,
  $father(john, mary), father(david, steve),$
- but none of the **negative examples**.
  $father(kathy, mary), father(john, steve),$

A **solution** is

$$father(X, Y) \leftarrow parent(X, Y) \wedge male(X).$$

Where Logic Programming Fails

Meanwhile, in the **real world**, samples are *incomplete* and come with *noise*.

Logic inference is fragile: a mistake in the transcription of a fact is dramatic to the consequences:

- $parent(david, mary)$.
- $parent(jonh, mary)$.

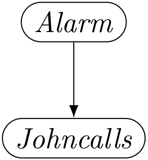
The statistic essence of machine learning provides robustness.

Probabilistic Logic Programming

- Define distributions from logic programs.
- Stochastic ASP: Specifying distributions.

Probabilistic Logic Programs (PLPs)

Logic programs **annotated** with probabilities.



$$\begin{aligned} alarm &: 0.00251, \\ johncalls &: 0.9 \leftarrow alarm, \\ johncalls &: 0.05 \leftarrow \neg alarm \end{aligned}$$

- $alarm : 0.00251$ is $alarm \vee \neg alarm$ plus $P(Alarm = true) = 0.00251$.
- $johncalls : 0.9 \leftarrow alarm$ is

$$P\left(Johncalls = true | Alarm = true\right) = 0.9$$

Any bayesian network can be represented by a PLP.

Distributions from Logic Programs

The program

$$\begin{aligned} alarm &: 0.00251, \\ johncalls &: 0.9 \leftarrow alarm, \\ johncalls &: 0.05 \leftarrow \neg alarm \end{aligned}$$

entails four possible models (or worlds):

model	probability
$alarm, johncalls$	0.002259
$alarm, \neg johncalls$	0.000251
$\neg alarm, johncalls$	0.049874
$\neg alarm, \neg johncalls$	0.947616

- **Models** are special sets of *literals* **entailed** from the program.
- Probabilities *propagate* from facts, through rules.

There's a Problem...

The program

$$alarm : 0.00251,$$
$$johncalls \vee marycalls \leftarrow alarm$$

entails three stable (*i.e.* minimal) models

model	probability
$alarm, johncalls$	x
$alarm, marycalls$	y
$\neg alarm$	0.99749

but **no single way to set** x, y.

Some *Probabilistic Logic Programs* define more than one joint distribution.

. . . and an Oportunity

Some *PLP*s define more than one joint distribution.

- There is **no single probability assignment** from the facts stable models: $x, y \in [0, 1]$.
- But any assignment is bound by Kolmogorov's axioms, and **forms equations** such as:

$$x + y = P\left(alarm\right).$$

- Existing **data can be used to estimate the unknowns** in those equations.

Stable Models, Events and Probabilities

What are we talking about?

- A logic program has **atoms** (and **literals**) and **rules**:

$$male(john), \neg parent(kathy, mary),$$
$$father(X, Y) \leftarrow parent(X, Y) \wedge male(X).$$

- A **stable model** is a **minimal** model that contains:
  - program's *facts*: $parent(john, mary),\ male(john)$.
  - consequences, by the *rules*: $father(john, mary)$.
- Some programs have more than one model:

Logic Program	**Stable Models**
$a \vee \neg a, b \vee c \leftarrow a$	$\{\neg a\}, \{a, b\}, \{a, c\}$

How to propagate probability from annotated facts to other *events*?

Logic Programs and Probabilities

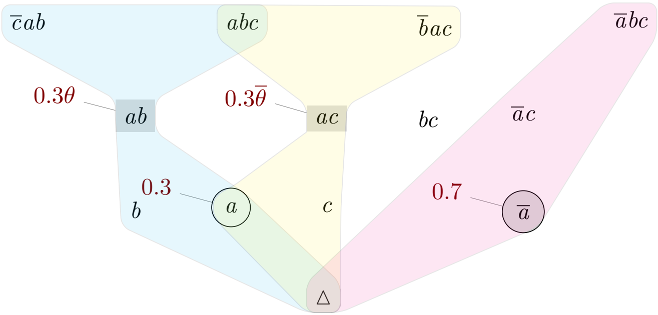
- Consider the literals of a logic program

$$L = \{a_1, \ldots a_n, \neg a_1, \ldots \neg a_n\}.$$

- Any model of that program is a (consistent) subset of L.
- Let $\Omega = \mathbf{P}(L)$, *i.e.* an event e is a subset of L, $e \subseteq L$.
    - Setting a probability for some events seems straightforward: $P(\neg alarm) = 0.997483558$.
    - For others, not so much:
        - $P(alarm, johncalls)$, $P(johncalls, marycalls, alarm)$, $P(marycalls)$?
        - $P(alarm, \neg alarm)$, $P(\neg marycalls)$?

How to propagate probability from *facts* to *consequences* and other *events*?

Classes of Events

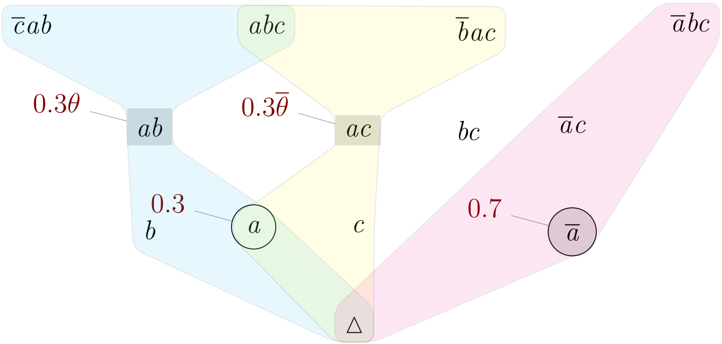


$$a : 0.3$$
$$b \vee c \leftarrow a$$

$$\overline{a} = \{\neg a\}, ab = \{a, b\}, ac = \{a, c\}$$

- Define equivalence classes for all events, based on $\subseteq, \supseteq$ relations with the **stable models**.
- This example shows 6 out of $2^3 + 1$ classes.

Probabilities for all Events



1. Set weights in the stable models (shaded nodes), using parameters when needed: $\mu\left(\overline{a}\right) = 0.7; \mu\left(ab\right) = 0.3\theta; \mu\left(ac\right) = 0.3\left(1 - \theta\right)$
2. Assume that the stable models are disjoint events.
3. Define weight of an event as the sum of the weights of the related stable models.
4. Normalize weights to get a (probability) distribution.

Probabilities for all Events

	$[\![e]\!]$	$\#[e]_\sim$	$\mu\big([e]_\sim\big)$	$\mu(e)$	$\mathrm{P}(E = e)$	$\mathrm{P}\big(E \in [e]_\sim\big)$
	$\bot$	37	0	0	0	0
$\square$	$\Diamond$	9	0	0	0	0
	$\overline{a}$	9	$\frac{7}{10}$	$\frac{7}{90}$	$\frac{7}{207}$	$\frac{7}{23}$
	ab	3	$\frac{3}{10}\theta$	$\frac{1}{10}\theta$	$\frac{1}{23}\theta$	$\frac{3}{23}\theta$
	ac	3	$\frac{3}{10}\overline{\theta}$	$\frac{1}{10}\overline{\theta}$	$\frac{1}{23}\overline{\theta}$	$\frac{3}{23}\overline{\theta}$
	$\overline{a}$, ab	0	$\frac{7+3\theta}{10}$	0	0	0
	$\overline{a}$, ac	0	$\frac{7+3\overline{\theta}}{10}$	0	0	0
	ab, ac	2	$\frac{3}{10}$	$\frac{3}{20}$	$\frac{3}{46}$	$\frac{3}{23}$
	$\overline{a}$, ab, ac	1	1	1	$\frac{10}{23}$	$\frac{10}{23}$
		64		$Z = \frac{23}{10}$		

Estimating the Parameters

A sample can be used to estimate the parameters θ, by minimizing

$$\mathrm{err}(\theta) := \sum_{e \in \mathcal{E}} \big(\mathrm{P}(E = e \mid \Theta = \theta) - \mathrm{P}(S = e)\big)^2 .$$

where

- $\mathcal{E}$ is the set of all events,
- $\mathrm{P}(E \mid \Theta)$ the **model+parameters** based distribution,
- $\mathrm{P}(S)$ is the **empiric** distribution from the given sample.

Behind Parameter Estimation

So, we can derive a distribution $\mathrm{P}\left(E \;\middle|\; \Theta = \hat{\theta}\right)$ from a program P and a sample S.

- The sample defines an empiric distribution $\mathrm{P}(S)$...
- ...that is used to estimate θ in $\mathrm{P}(E \mid \Theta)$...
- ...and score the program P w.r.t. that sample using, *e.g.* the $\mathrm{err}()$ function.

Back to Inductive Logic Programming

Recall the *Learning Logic Programs from Examples* setting:

- Given **positive** and **negative** examples, and **background knowledge**. . .
- find a **program**. . .
  - . . . using the facts and relations from the **BK**. . .
  - . . . such that **all the PE** and **none the NE** examples are entailed.

Given a sample of events, and a set of programs, the score of those programs (w.r.t. the sample) can be used in evolutionary algorithms while searching for better solutions.

In Conclusion

- **Machine Learning** has limitations.
- As does **Inductive Logic Programming**.
- But, distributions can be defined by **Stochastic Logic Programs**.

Distributions can be defined by **Stochastic Logic Programs**. Here we:

1. Look at the program's **stable models** and
2. Use them to partition the **events** and then
3. Using annotated probabilities, define:
    1. a finite **measure**. . .
    2. that, normalized, is a **distribution** on all events.

Distributions can be defined by **Stochastic Logic Programs**.

- These distributions might have some **parameters**, due to indeterminism in the program.
- A **sample** can be used to estimate those parameters...
- ...and **score** programs concurring to describe it.
- This score a key ingredient in **evolutionary algorithms**.

*...and a step towards the induction of stochastic logic programs using **data** and **background knowledge**.*

Future Work

Induction of Stochastic Logic (ASP) Programs.

1. **Meta-programming:** formal rules for rule generation.
2. **Generation**, **Combination** and **Mutation** operators.
3. **Complexity.**
4. **Applications.**
5. **Profit.**

Thank You!

Questions?

References

- Gary Marcus, *Deep Learning: A Critical Appraisal*, 2018.
- François Chollet, *On the Measure of Intelligence*, 2019.
- Bengio *et al.*, *A Meta-Transfer Objective for Learning to Disentangle Causal Mechanisms*, 2019.
- Cropper *et al.*, *Turning 30: New Ideas in Inductive Logic Programming*, 2020.
- Fabrizio Riguzzi, *Foundations of Probabilistic Logic Programming* , 2023.