Probabilistic Answer Set Programming

A Research Draft

Francisco Coelho

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

May 23, 2022

In short. . . . . . a word wall. I'm sorry.

- **Machine Learning** has important limitations:
  - The *one table, conditionally independent rows* assumption.
  - *Background knowledge* is hard to include.
  - *Training* requires "large" amounts of data.
  - *Models* are hard do interpret.
- **Inductive Logic Programming** is based on first order logic — solves all the problems above but is sensible to *noise*.
- **Distribution Semantics** defines the probability of a proposition from probabilities of the (marginally independent) facts.
- **Answer Set Programs** resets the common syntax and semantic of logic programs; A "program" defines *stable models*, not a computation neither a variable substitution.

~~Goals~~ Wish list

Extend distribution semantics to answer sets

- Within a theoretical framework.
- Computationally applicable to “real world” scenarios.
- Easy to include background knowledge.
- Perform common tasks such as *marg, mle, map, etc.*
- Learn program “parameters” and “structure” from *noisy samples* — possibly using *templates*.
- Related to Bayesian Networks, HMMs, *etc.*

The seed on an idea

We want to define the **joint distribution** of the stable models.

1. A **boolean random variable** can be described by a disjunction $a; \neg a$.
2. This ASP program has two stable models: a and $\neg a$.
3. A program with n such facts $a_i; \neg a_i$ has 2^n stable models, the distinct combinations of those choices.
4. **If each a_i has probability p_i then the probability of a stable model W would be**

$$P(W) = \prod_{a_i \in W} p_i \prod_{\neg a_i \in W} (1 - p_i).$$

The seed on an idea

We want to define the **joint distribution** of the stable models.

1. A **boolean random variable** can be described by a disjunction $a; \neg a$.
2. This ASP program has two stable models: a and $\neg a$.
3. A program with n such facts $a_i; \neg a_i$ has 2^n stable models, the distinct combinations of those choices.
4. **If each a_i has probability p_i then the probability of a stable model W would be**

$$P(W) = \prod_{a_i \in W} p_i \prod_{\neg a_i \in W} (1 - p_i).$$

But this is wrong.

Even assuming that those facts are marginally independent — which we will do.

Problem 1: Disjuntive Clauses

The ASP program with probabilistic facts

$$b \vee \neg b$$
$$h_1 \vee h_2 \leftarrow b$$

has **three** stable models: $\{\neg b\}, \{b, h_1\}$ and $\{b, h_2\}$.

How to assign a probability to each model?

Problem 1: Disjuntive Clauses

The ASP program with probabilistic facts

$$b \vee \neg b$$
$$h_1 \vee h_2 \leftarrow b$$

has **three** stable models: $\{\neg b\}, \{b, h_1\}$ and $\{b, h_2\}$.

How to assign a probability to each model?

Possible approaches:

1. Pre-assign a **conditional distribution of the head**:

$$P(h_1, h_2 | b).$$

2. Bayesian learn from **observations**:

$$P(h_1, h_2 | b, z) \propto P(b, z | h_1, h_2) P(h_1, h_2).$$

3. Start with the former as **prior** and **update** with the latter.

Questions to address

- How to **match** an observation z with a clause case h, b?
- How do observations **update** the probabilities?
- Why match observations with clauses and **not with stable models**?
- Is this just **bayesian networking**?
- How to frame this in a **sound theoretic setting**?
- Is this enough to compute the **joint distribution of the atoms**?

Questions to address

- How to **match** an observation z with a clause case h, b?
- How do observations **update** the probabilities?
- Why match observations with clauses and **not with stable models**?
- Is this just **bayesian networking**?
- How to frame this in a **sound theoretic setting**?
- Is this enough to compute the **joint distribution of the atoms**?

Counters

Instead of setting and updating probabilities, we associate **counters** to disjunctive clauses and their cases.

Bayesian updates: Matching observations

- An observation is a subset of the literals from a program[1].
- A consistent observation has no subset $\{p, \neg p\}$.
- A **consistent** observation z is relevant for the clause $h \leftarrow b$ if $b \subseteq z$.
- A disjunctive clause

$$h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$$

  has n cases: $\{h_i, b_1, \ldots, b_m\}, i = 1 : n$.
- The **consistent** observation z matches the case $\{h, b_*\}$ if $\{h, b_*\} \subseteq z$.

The above definitions also apply to **facts** *i.e.* clauses with an empty body and **constraints** *i.e.* clauses with no head.

[1] The set of atoms, a, of the program and their classic negations, $\neg a$.

Bayesian updates: Clauses Update

A consistent observation **relevant** for a clause $h_1 \vee \cdots \vee h_n \leftarrow b$ should:

- Increase the *probability of any matched case*.
- Decrease the *probability of any unmatched case*.

Bayesian updates: Clauses Update

A consistent observation **relevant** for a clause $h_1 \vee \cdots \vee h_n \leftarrow b$ should:

- Increase the *probability of any matched case*.
- Decrease the *probability of any unmatched case*.

Update algorithm

1. Associate three **counters**, r, u, n, to each clause $h \leftarrow b$.
2. Associate a **counter**, m_i, to each case h_i, b of each clause.
3. **Initial** values result from *prior* knowledge.
4. Each *consistent* observation **increments**:
   - The r counters of relevant clauses.
   - The u counters of unmatched relevant clauses.
   - The n counters of not relevant clauses.
   - The m_i counters of matched cases h_i, b.
   - Clause counters must verify $r \leq u + \sum_i m_i$.

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$$b \vee \neg b \quad \text{counters: } 7, 2; 12, 3, 0$$

$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 6, 2, 5$$

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$$b \vee \neg b \quad \text{counters: } 7, 2; 12, 3, 0$$

$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 6, 2, 5$$

Counters of $b \vee \neg b$

0 observations where not relevant (because the body is $\top$);
There where 12 relevant observations;
Of those, b was matched by 7, $\neg b$ by 2 and 3 observations matched neither ($\models \sim b, \sim\neg b$).

Counters of $h_1 \vee h_2 \leftarrow b$

There where $11 = 6 + 5$ observations, 6 relevant to this clause;
From these, 4 matched h_1, 3 matched h_2 and 2 matched no case.

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$$b \vee \neg b \quad \text{counters: } 7, 2; 12, 3, 0$$

$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 6, 2, 5$$

What can be computed?

- $P(\neg b) = \frac{2}{12}$ because $\neg b$ matched 2 of 12 relevant observations.
- $P(h_1|b) = \frac{4}{6}$ because h_1 matched 4 of 6 relevant observations.
- $P(b)$ can't be computed without further information. *E.g.* supposing that **observations are independent** then

$$P(b) = \frac{7+6}{12+0+6+5}.$$

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$b \vee \neg b$	counters: $7, 2; 12, 3, 0$
$h_1 \vee h_2 \leftarrow b$	counters: $4, 3; 6, 2, 5$

Note...

Counters are local to clauses and, for distinct clauses, may result from distinct sources. *E.g. the relevant counter of* $h_1 \vee h_2 \leftarrow b$ *and the match counter of* b *in* $b \vee \neg b$*.*

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$$b \vee \neg b \quad \text{counters: } 7, 2; 12, 3, 0$$

$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 6, 2, 5$$

Note...

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$b \vee \neg b$	counters: $7, 2; 12, 3, 0$
$h_1 \vee h_2 \leftarrow b$	counters: $4, 3; 6, 2, 5$

Note...

Some observations may have neither b nor $\neg b$:

$$P(b) + P(\neg b) < 1.$$

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$$b \vee \neg b \quad \text{counters: } 7, 2; 12, 3, 0$$
$$h_1 \vee h_2 \leftarrow b \quad \text{counters: } 4, 3; 6, 2, 5$$

Note...

Since h_1 and h_2 are not independent,

$$\sum_m P(m) \approx 1.02 > 1.$$

Updates and counters: An example

Given the following ASP program with **annotated counters**,

$$b \lor \neg b \quad \text{counters: } 7, 2; 12, 3, 0$$
$$h_1 \lor h_2 \leftarrow b \quad \text{counters: } 4, 3; 6, 2, 5$$

Note...

What is missing to compute the joint distribution of the program's atoms

$$P(H_1, H_2, B)?$$

Shortcomming 2: Default Negation

- How to deal with rules with $\sim a$ parts?
- Should missing elements on observations be replaced with $\sim a$ atoms?

1 Development

2 Conclusions

Background Material

Machine Learning

Models are numeric functions: $y \approx f_\theta(x),\ \theta_i, x_j, y \in \mathbf{R}$.

- Amazing achievements.
- Noise tolerant.
- (as of today) Huge enterprise funding .

but

- (essentially) Academically solved.
- Models trained from “large” amounts of samples.
- Hard to add background knowledge.
- Models are hard to interpret.
- Single table, independent rows assumption.

Inductive Logic Programming

Models are logic program: $p_\theta(x, y),\ \theta_i, x_j, y \in \mathcal{A}$.

- Amazing achievements, at scale.
- Models trained from "small" amounts of samples.
- Compact, readable models.
- Background knowledge is easy to incorporate and edit.

but

- as of today, Little enterprise commitment.
- as of today, Mostly academic interest.
- Noise sensitive.

Distribution Semantics

Assigns probability to (marginally independent) facts and derives probability of ground propositions.
Let F be set of facts, $S \subseteq F$, R a set of definite clauses and p a proposition:

$$P_F(S) = \prod_{f \in S} P(f) \prod_{f \notin S} \big(1 - P(f)\big)$$

$$P(W) = \sum_{S \subseteq F:\ W = M(S \cup R)} P_F(S)$$

$$P(p) = \sum_{S:\ S \cup R\ \vdash\ p} P_F(S) = \sum_{W:\ p \in W} P(W)$$

- Amazing achievements, at scale.
- Lots of tools and research.
- The best of both “worlds”?

Answer Set Programming

A "program" defines stable models *i.e.* minimal sets of derived ground atoms[2].

- Pure declarative language, unlike Prolog.
- Uses *generate & test* methods instead of proofs .
- Uses both default $\sim p$ and classical negation $\neg p$[3]
- Clauses can be disjunctive $a; b \leftarrow c, d$.

---

[2] Alternative ~~fact~~ definition: X is a stable model of P if $X = \mathrm{Cn}(P^X)$.

[3] Classic negation $\neg a$ in ASP results from replacing the occurrences of $\neg a$ by a new atom $a_\neg$ and adding the restriction $\leftarrow a_\neg, a$.