The Distribution Semantics

Felix Quirin Weitkämper

Institut für Informatik der Ludwig-Maximilians-Universität München

June 6, 2023

Probabilistic Logic Programming

- Probabilistic logic programming is a key part of statistical relational artificial intelligence, with an extensive literature around both learning and inference.
- Within statistical relational artificial intelligence, probabilistic logic programming has distinct features:
  - The distribution semantics
  - Recursion and stratified negation taken from logic programming

Outline

The uniform random world

A uniform distribution over Ω_D^S assigns the same probability to each $\omega \in \Omega_D^S$.

We can obtain such a uniform distribution by tossing a fair coin for every ground atom $R(\vec{a})$, where $R \in S$ of arity n and $\vec{a} \in D^n$.

If the coin turns up heads, we set $R(\vec{a})$ to true; otherwise we set $R(\vec{a})$ to false.

Loaded coin flips

We can generate different distributions by throwing loaded coins.

If the coin turns up heads, we set $R(\vec{a})$ to true; otherwise we set $R(\vec{a})$ to false.

Example

Consider $S = \{\text{calls}/1, \text{alarm}\}$ and $D = \{\text{john}, \text{mary}\}$.
Then we could throw a coin with probability 0.1 of showing heads for each of calls(john) and calls(mary) and a coin with probability 0.2 of showing heads for alarm.

Independent distributions

Definition

Random worlds generated by throwing a (loaded) coin for every ground atom $R(\vec{a})$, where $R \in S$ of arity n and $\vec{a} \in D^n$, are called *independent distributions*.

They are characterised by the fact that the events $R(\vec{a})$ and $S(\vec{b})$ are unconditionally independent unless $R = S$ and $\vec{a} = \vec{b}$.

Probabilistic facts

Independent distributions can be specified by *probabilistic facts*.

Definition

A probabilistic fact consists of a probability α, a head atom H and a body B which is a conjunction of sort atoms, such that every variable in H occurs in at most one atom in B.
We write probabilistic facts as

$$\alpha :: H \leftarrow B.$$

Probabilistic facts

Example

Probabilistic facts can contain free variables, as in

$$\text{calls}(X) \leftarrow \text{person}(X).$$

They can also be ground, as in calls(mary).

Probabilistic facts

A finite set of probabilistic facts induces an independent distribution on each domain:

- For every probabilistic fact proving a ground atomic query $R(a_1, \ldots, a_n)$, toss a coin that shows heads with probability α.
- If any of these coins shows heads, then $R(a_1, \ldots, a_n)$ is set true.

Probabilistic facts

Example

Consider the probabilistic facts

$$0.1 :: \text{calls}(X) \leftarrow \text{person}(X).\ 0.2 :: \text{calls}(\text{mary}).\ 0.05 :: \text{burglary}$$

Then any person in the domain has an independent 10% chance of calling, but Mary has a 28% chance.
A burglary has a 5% chance of happening.

Symmetries

The independent distribution encoded by a set of probabilistic facts obeys a basic symmetry principle:

For every two domain individuals a and b that do not occur explicitly named in the head of a probabilistic fact, replacing a with b in an atom does not change its probability.

Example

In the example above, any person who is not Mary has the same chance of calling.

Probabilistic logic programming

Fundamental idea

A probabilistic logic program under the distribution sematics consists of

1. An independent distribution over basic facts
2. A logic program defining new predicates from those facts

In other words, the distribution semantics separates the logical from the probabilistic part of a PLP, with the probabilistic part reduced to fairly simple distributions.

Extensional and intensional signature

To use logic programs for defining new predicates, we need to distinguish between what we are defining and the terms in which we are defining them.

The logic programs that we consider will therefore have predicate symbols that do not occur in the head of any clause. They are sometimes called *lifted* logic programs.

Definition

Let Π be a logic program (i. e. a set of clauses). Then the *extensional signature* $\mathrm{Ext}(\Pi)$ of Π consists of all those relation symbols that only occur in the body of clauses. The signature of all relation symbols that occur in the head of any clause is called the *intensional signature* $\mathrm{Int}(\Pi)$.

Definitions

A lifted logic program defines its extensional predicates in terms of its intensional predicates.

Example

Consider the following logic program Π:
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
Then $\text{Ext}(\Pi) = \{\text{edge}/2\}$ and $\text{Int}(\Pi) = \{\text{path}/2\}$. Π defines the predicate path for any given set of edge relations.

Definitions

Formally speaking, a *definition* of I in E is a function taking a possible E-world as input and returning a possible I-world.

Example

The input of the logic program Π from the last slide is a directed graph G, and it returns a new binary relation path on the nodes of G which is the transitive relation of the edge relation in G.
If the input graph G is given by

$$\text{edge}(a, b).\ \text{edge}(b, c).\ \text{edge}(c, a).$$

then $\Pi(G)$ is the graph with relation path on the nodes $\{a, b, c\}$ for which $\text{path}(x, y)$ holds for all $x, y \in \{a, b, c\}$.

Definitions and distributions

- Assume that we are given two signatures E and I, a random E-world μ and a definition Π of I in E. Then we can derive a random I-world by simply applying the definition:

Definition

Let $\Pi(\mu)$ be the random I-world such that

$$\Pi(\mu)(\omega) = \mu(\Pi^{-1}(\omega))$$

for every possible I-world ω.

Definitions and distributions

Example

Let μ be the uniform distribution on edge-graphs with domain D and let Π be our running example. Then for any transitive path-graph ω, $\Pi(\mu)(\omega)$ is the fraction of directed graphs on node set D whose transitive closure is ω. If ω is intransitive, then $\Pi(\mu)(\omega) = 0$.

Probabilistic logic programs

- We can now introduce our general framework for probabilistic logic programs:

Definition

A *probabilistic logic program* in signature I consists of...

1. A finite set of probabilistic facts in a signature E.
2. A logic program defining I in E.
3. A domain D, expressed by sort facts.

A *lifted* probabilistic logic program consists only of points 1 and 2.

Probabilistic logic programs

Example

We could extend the transitive closure program above into a lifted probabilistic logic program by including a single sort node which lists nodes, and then adding the probabilistic fact $0.7 :: \text{edge}(X, Y) \leftarrow \text{node}(X), \text{node}(Y)$.
A domain could then be any set of nodes, such as
node(a). node(b). node(c).

Probabilistic clauses

Even the simple examples from last lecture do not seem to fit into our framework

Example

0.7::burglary.
0.2::earthquake.
0.9::alarm :- burglary.
0.8::alarm :- earthquake.

Probabilistic clauses

All clauses are annotated with probabilities, not just probabilistic facts!
We can convert these general *probabilistic clauses* into a probabilistic fact and a logic programming clause:

Definition

A *probabilistic clause* C is of the form $\alpha :: H \leftarrow B$, where $\alpha \in [0, 1]$, H is an atom and B is a conjunction of literals, but not necessarily of sort atoms.
Let $X_1, \ldots, X_n$ be the variables in C and let B' be the sort atoms in B.
Then C is syntactic sugar for

$$\alpha :: u_C(X_1 \ldots, X_n) \leftarrow B'.$$
$$H \leftarrow B, u_C.$$

Probabilistic clauses

Definition

A *probabilistic clause* C is of the form $\alpha :: H \leftarrow B$, where $\alpha \in [0, 1]$, H is an atom and B is a conjunction of literals, but not necessarily of sort atoms.
Let $X_1, \ldots, X_n$ be the variables in C and let B' be the sort atoms in B.
Then C is syntactic sugar for

$$\alpha :: u_C(X_1 \ldots, X_n) \leftarrow B'.$$
$$H \leftarrow B, u_C.$$

Probabilistic clauses

The example program from above is translated to the following PLP:

Example

```
0.7::burglary.
0.2::earthquake.
0.9::u_1.
alarm :- burglary, u_1.
0.8::u_2.
alarm :- earthquake, u_2.
```

Combining two clauses

In the distribution semantics, two probabilistic facts are combined as independent coin tosses:

If a ground atomic query is proven by two probabilistic facts with probabilities α and β, then its unconditional probability is given by $\alpha + \beta - \alpha\beta$.

This formula can be generalised inductively to more than two probabilistic facts.

Combining two clauses

By the definitions earlier, the same is true for two or more probabilistic clauses.

Every probabilistic clause is realised with a separate probabilistic fact.

Therefore, if there is a burglary *and* an earthquake, then the chance of an alarm is given by the combination of the independent u_1 and u_2.

Indeterminate clauses

The same combination rule is employed for indeterminate clauses, that is, probabilistic clauses with free variables in the body that are not in the head.

Example

The clause

$$0.2 :: \text{ill}(X) \leftarrow \text{contact}(X, Y), \text{infectious}(Y).$$

defines an independent 20% coin toss for every infectious contact.

Annotated disjunctions

All that we have discussed so far concerns *Boolean* relations, which may or may not hold individually.

Commonly, one wants to model choices between mutually exclusive possibilities

For instance, a die will roll any one of 1, 2, 3, 4, 5 and 6 but not more than one number at a time.

Annotated disjunctions

This is expressed by an *annotated disjunction*:

$$\frac{1}{6} :: \text{one(X)}; \frac{1}{6} :: \text{two(X)}; \frac{1}{6} :: \text{three(X)};$$
$$\frac{1}{6} :: four(X); \frac{1}{6} :: five(X); \frac{1}{6} :: six(X).$$

Semantically, the annotated disjunction means that each event will hold with the stated probability, and not more than one can hold.

This only makes sense if the annotations sum up to less than or equal to 1.

Annotated disjunctions

Annotated disjunctions can be expressed in the distribution semantics:

$$\frac{1}{6} :: \text{one}(X).$$

$$\frac{1}{5} :: \text{two}(X) \leftarrow \neg\text{one}(X).$$

$$\frac{1}{4} :: \text{three}(X) \leftarrow \neg\text{one}(X), \neg\text{two}(X).$$

$$\frac{1}{3} :: \text{four}(X) \leftarrow \neg\text{one}(X), \neg\text{two}(X), \neg\text{three}(X).$$

$$\frac{1}{2} :: \text{five}(X) \leftarrow \neg\text{one}(X), \neg\text{two}(X), \neg\text{three}(X), \neg\text{four}(X).$$

$$\text{six}(X) \leftarrow \neg\text{one}(X), \neg\text{two}(X), \neg\text{three}(X), \neg\text{four}(X), \neg\text{five}(X).$$

Stochastic Logic Programs

The distribution semantics is not the only possible semantics for probabilistic logic programs.

Stochastic logic programs give meaning directly to probabilistic clauses, without deconstructing them.

Stochastic Logic Programs

In a stochastic logic program, the probability is given to the resolution proof of the Prolog trace.

Definition

A *stochastic logic program* is a finite set of probabilistic clauses without negation such that the probabilities of all clauses with the same head sum to one.

For this brief outlook, we limit ourselves to *ground* SLP. Otherwise, one has to ensure that the probabilities of all clauses whose head unifies with a given query always sums to one.

Stochastic Logic Programs

Let φ be a query atom and Σ a stochastic logic program.

Then when choosing which head to unify φ with, the stochastic logic program chooses any rule with probability α, where α is the annotation of that rule.

Then the probability of φ to be true is the probability of the proof terminating successfully rather than eventually reaching fail.

Example

We can model an alarm-burglary scenario as follows:

Example

0.7::burglary :- true.
0.3::burglary :- fail.
0.2::earthquake :- true.
0.8::earthquake :- fail.
0.7 :: alarm :- (burglary;earthquake).
0.1 :: alarm :- burglary.
0.1 :: alarm :- burglary, earthquake.
0.1 :: alarm :- fail.
Note that one has to make all different scenarios explicit.

No aggregation

The lack of implicit aggregation can sometimes be counterintuitive:

Example

```
0.5 :: goal :- fact_1.
0.5 :: goal :- fact_2.
1 :: fact_1 :- true.
1 :: fact_2 :- true.
```

- In the distribution semantics, both facts would have a 50% chance of leading to goal. Overall, this would imply a 75% chance of goal being true.
- In SLP, either branch of the proof of goal would certainly lead to success, so goal is true with probability 1.

No aggregation

- In the distribution semantics, both facts would have a 50% chance of leading to goal. Overall, this would imply a 75% chance of goal being true.
- In SLP, either branch of the proof of goal would certainly lead to success, so goal is true with probability 1.

No memoisation

There is also no sharing of subgoals between branches:

Example

```
1 :: goal :- left, right.´
1 :: left :- fact.
1 :: right :- fact.
0.5 :: fact :- true.
0.5 :: fact :- fail.
```

No memoisation

- In the distribution semantics, fact would have a 50% chance of being true. If true, it leads to both left and right being true and then goal is also true.
- In SLP, the proof of goal will branch into proofs of left and right, and then both of them will have an independent 50% chance of ending in fact. Therefore, there is an overall probability of 25% that goal is true.