The cplint Probabilistic Logic Programming System

Elena Bellodi,
Evelina Lamma
Department of Engineering
University of Ferrara
Ferrara, Italy
{blllne2, lme}@unife.it

Riccardo Zese,
Giuseppe Cota
Department of Engineering
University of Ferrara
Ferrara, Italy
{zsercr,ctogpp}@unife.it

Fabrizio Riguzzi
Department of Mathematics and
Computer Science
University of Ferrara
Ferrara, Italy
rzf@unife.it

Probabilistic Logic Programming (PLP) [1] is a probabilistic programming approach that uses logic programming as the underlying language. With respect to probabilistic programming based on imperative or object-oriented languages, PLP offers the additional features of declarativity and ease of knowledge representation, making it a valid tool for Statistical Relational Learning and, more generally, Statistical Relational Artificial Intelligence. In fact, *PLP is very useful for modeling domains with complex and uncertain relationships among entities, through logic and probability theory respectively.* This poster would like to present our system `cplint` and our web application `cplint on SWISH` [2] for performing inference and learning in such domains by means of Probabilistic Logic Programming.

Sato's distribution semantics [3] emerged as one of the most prominent for giving a meaning to PLP, being adopted by many languages among which Logic Programs with Annotated Disjunctions (LPADs) [4]. A probabilistic logic program without function symbols defines a distribution over normal, non-probabilistic logic programs called worlds and the probability of a query is obtained from the joint distribution of the query and the worlds by marginalization. The distribution over worlds is defined by encoding choices in clauses. For example, in LPADs, the clause heads are disjunctions and each disjunct is annotated with a probability, as in the following program that models information on the laboratory examinations of hepatitis B and C infected patients (http://www.cs.sfu.ca/~oschulte/jbn/dataset.html). Here the goal is to predict the hepatitis type of a patient, so the "target head" in the clauses is *type(patient,type)* where *type* can be type b or type c:

```
type(A,type_c):0.770:-b_rel11(B,A),fibros(B,C),
b_rel13(D,A).
type(A,type_b):0.403:-b_rel11(B,A), fibros(B,C).
```

The clauses' body contains literals relative to results of biopsy, information on interferon therapy, results of out-hospital examinations, results of in-hospital examinations. The first clause states that a patient A has a chance of having type c hepatitis of 77% if the body holds (the atoms are true in the knowledge base).

Continuous random variables have been added to the language [5] by means of clauses of the form `a : Density ←Body,` where *Density* is a special atom identifying a probability density on variable *Var* of atom *a* and *Body* (optional) is a regular clause body. For example, consider a factory with two machines a and b. Each machine produces a widget with a continuous feature distributed as a Gaussian with a given mean and variance. The widget then is processed by a third machine that adds a random quantity to the feature distributed as a Gaussian:

```
widget(X) :- machine(M),st(M,Z),pt(Y),{X=:=Y+Z}.
machine(a):0.3;machine(b):0.7.
st(a,X): gaussian(X, 2.0, 1.0).
st(b,X): gaussian(X, 3.0, 1.0).
pt(X): gaussian(X, 0.5, 1.5).
```

One can compute what is the distribution of the feature, or the distribution of the feature given that the widget was produced by machine a, by taking S samples with rejection sampling and drawing a histogram with NB bins, and many other queries.

`cplint` is a suite of algorithms for performing (exact and approximate) inference and learning with LPADs including programs that encode continuous random variables. It can be used as a local application in one of the Prolog compilers Yap, XSB or SWI-Prolog, or as a web application, `cplint on SWISH` (http://cplint.eu). It has extensive graphical capabilities, either using R or C3.js. For example, it can draw ROC and PR curves.

`cplint on SWISH` manages a lot of application domains: reasoning about actions, random walks, marketing, natural language, biology, genetics, model checking, medicine, games, social networks, filtering, Bayesian estimation, regression, tools diagnosis models, bitcoin protocols, but can be applied to any complex real world domain characterized by imprecise knowledge.

REFERENCES

[1] F. Riguzzi, Foundations of Probabilistic Logic Programming, River Publishers, 2018.

[2] F. Riguzzi, E. Bellodi, E. Lamma; R. Zese,; G. Cota, 2016. Probabilistic logic programming on the web. In SOFTWARE, PRACTICE AND EXPERIENCE, vol. 46 (10), pp.1381-1396.

[3] T. Sato. 1995. A Statistical Learning Method for Logic Programs with Distribution Semantics. In Logic Programming, Proc. of the Twelfth Int. Conference on Logic Programming, 1995, Leon Sterling (Ed.). MIT Press, Cambridge, Massachusetts, 715–729.

[4] J. Vennekens, S. Verbaeten, and M. Bruynooghe. 2004. Logic Programs With Annotated Disjunctions. In 24th Int. Conference on Logic Programming (ICLP 2004) (Lecture Notes in Computer Science), Vol. 3131. Springer, Berlin Heidelberg, 431–445.

[5] S. Michels, A. Hommersom, P. J. F. Lucas, and M. Velikova. 2015. A new probabilistic constraint logic programming language based on a generalised distribution semantics. Artif. Intell. 228 (2015), 1–44.