Probabilistic Inductive Logic Programming on the Web

Fabrizio Riguzzi[1], Riccardo Zese[2], and Giuseppe Cota[2]

[1] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[fabrizio.riguzzi,riccardo.zese,giuseppe.cota]@unife.it

Abstract. Probabilistic Inductive Logic Programming (PILP) is gaining attention for its capability of modeling complex domains containing uncertain relationships among entities. Among PILP systems, `cplint` provides inference and learning algorithms competitive with the state of the art. Besides parameter learning, `cplint` provides one of the few structure learning algorithms for PLP, SLIPCOVER. Moreover, an online version was recently developed, `cplint` on SWISH, that allows users to experiment with the system using just a web browser. In this demo we illustrate `cplint` on SWISH concentrating on structure learning with SLIPCOVER. `cplint` on SWISH also includes many examples and a step-by-step tutorial.

1 Introduction

Probabilistic Inductive Logic Programming (PILP) [3] uses Probabilistic Logic Programming (PLP) for modeling in domain characterized by uncertain relationships among entities.

One of most successful approaches to PLP is based on the distribution semantics [8] where a probabilistic program defines a probability distribution over non probabilistic programs, called worlds. The probability of a query is simply the sum of the probability of the worlds where the query is true. Various languages follow the distribution semantics such as Probabilistic Logic Programs, Logic Programs with Annotated Disjunctions (LPADs), CP-logic and ProbLog.

Many systems for performing inference and learning with these languages have been proposed in the past 20 years. Among them, `cplint` provides an interesting mix of algorithms, including structure learning algorithms.

`cplint` on SWISH [6] is a web application for running `cplint` with just a web browser: the algorithms run on a server and the user can post queries and see the results in his browser. `cplint` on SWISH is available at `http://cplint.lamping.unife.it` and Figure 1 shows its interface.

`cplint` on SWISH is based on SWISH, a web framework for Logic Programming using features and packages of SWI-Prolog and its Pengines library. SWISH allows the user to write a Logic Program in a browser window and ask a query

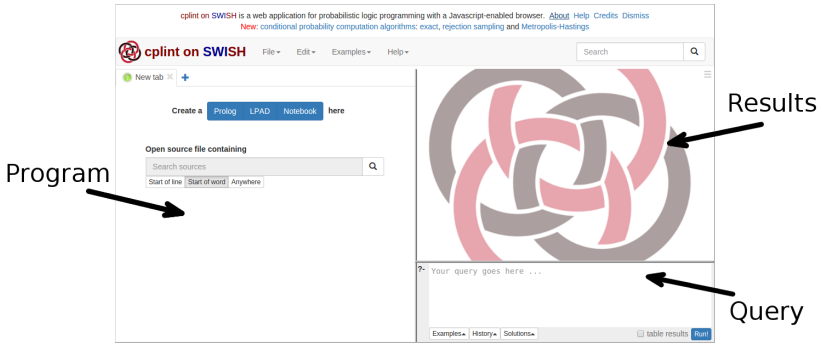

Fig. 1. `cplint` on SWISH interface.

over it. The query and the program are sent to a server using JavaScript. The server then builds a Pengine (Prolog Engine) that evaluates the query and returns answers for it to the user. Both the web server and the inference back-end are run entirely within SWI-Prolog.

`cplint` on SWISH uses the language of LPADs and includes two inference algorithms: PITA, that uses knowledge compilation, and MCINTYRE, that uses Monte Carlo sampling. For parameter learning EMBLEM [1] is available while SLIPCOVER [2] can be used for structure learning.

`cplint` on SWISH is similar to ProbLog2 [4] that has also an online version[3]. Problog2 offer inference and learning algorithms for ProbLog. In the online version, users are allowed to write programs and run algorithms on a server with a browser. ProbLog2 is written in Python, runs in an Python HTTP server and exploits the ACE editor[4] which is written in JavaScript.

The main difference between `cplint` on SWISH and ProbLog2 is that the first offers also structure learning. Moreover, `cplint` on SWISH uses a Prolog-only software stack. ProbLog2, instead, relies on several different technologies, including JavaScript, Python 3 and the DSHARP compiler. In particular, it writes intermediate files to disk in order to call external programs such as DSHARP while we work in main memory only.

With both `cplint` on SWISH and ProbLog2 users who want to experiment with PILP can do it without the need to install a system, a procedure which is often complex, error prone and limited mainly to the Linux platform. However, since it is impossible to predict the load of the server, the system is more targeted at development, while for production it is recommended to use the standalone version of `cplint`.

One of the main objectives of `cplint` on SWISH is to reach out to a wider audience and popularize PILP, similarly to what is done for the functional probabilistic language Church [5], which is equipped with the `webchurch` system for compiling Church programs into JavaScript. To try to achieve this goal, `cplint`

[3] `https://dtai.cs.kuleuven.be/problog/`
[4] `https://ace.c9.io/`

on SWISH includes various learning examples[5]: Machines (shown below), Registration, Bongard, Shop, Hidden Markov Model, Mutagenesis, University. Moreover, a complete online tutorial is available [7] at `http://ds.ing.unife.it/~gcota/plptutorial/`.

2 EMBLEM and SLIPCOVER

EMBLEM and SLIPCOVER in `cplint` on SWISH take as input a program divided in five parts: (1) a preamble where all the parameters (such as the maximum number of iterations and the verbosity level) are set, (2) a background knowledge that contains information valid for all interpretations, (3) an initial LPAD if there is one, (4) a language bias for guiding the learning phase. In particular, for EMBLEM it contains the declarations of the input and output predicates. Basically, input predicates are those for which we do not want to learn parameters, while we want to learn parameters for output predicates. SLIPCOVER uses also bias declaration in the style of Progol and Aleph: atoms for `modeh/2` specify the literals that can appear in the head of clauses while atoms for `modeb/2` specify the atoms that can appear in the body of clauses. Moreover, SLIPCOVER requires the use of the `determination/2` predicate, as in Aleph, for specifying which predicates can appear in the body of clauses.

Using the machine dataset of the ACE data mining system[6] as a running example[7] we have:

```
modeh(*,class(sendback)).
modeb(*,not_replaceable(-comp)).
modeb(*,replaceable(-comp)).
determination(class/1,replaceable/1).
determination(class/1,not_replaceable/1).
```

Finally, (5) the last part contains example interpretations. Here, we can use two different representations shown below, *models* or *keys*, as in ACE. The first specifies an example interpretation as a list of Prolog facts surrounded by `begin(model(<name>))` and `end(model(<name>))`. In the latter the facts can be directly listed using the first argument as the example name.

```
begin(model(1)).
class(sendback).
neg(class(fix)).
worn(engine).
end(model(1)).
```

```
class(1,sendback).
neg(1,class(fix)).
worn(1,engine).
```

We can also define folds and which examples are included so that learning can be performed by just asking the query `induce(<folds>,P).` The learned program will be returned in variable `P`.

`cplint` on SWISH has also facilities for testing the learned models: predicate `test/7` takes as input the learned program and the testing folds and returns the

[5] `http://cplint.lamping.unife.it/example/learning/learning_examples.swinb`
[6] `https://dtai.cs.kuleuven.be/ACE/`
[7] `http://cplint.lamping.unife.it/example/learning/mach.pl`

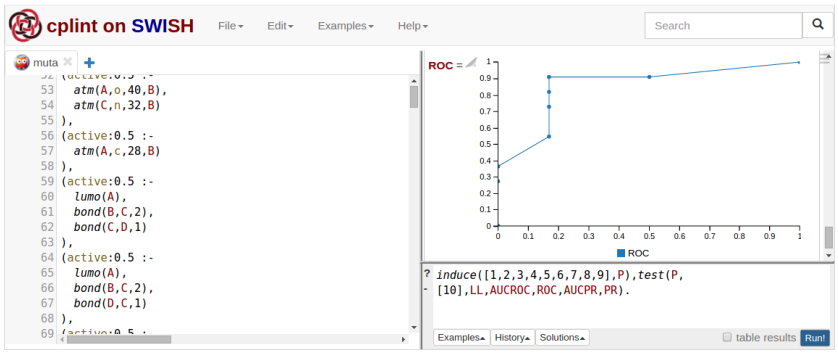

Fig. 2. ROC curve for the mutagenesis dataset.

log-likelihood, the areas under the precision-recall and receiver operating characteristics curves together with the set of points forming the curves themselves. These set of points can also be drawn on the screen as the respective curves, as shown in Figure 2. `cplint` on SWISH offers also a separate AUC calculator[8] that takes as input the list of examples together with their sign and probability.

Acknowledgement This work was supported by the "GNCS-INdAM".

References

1. Bellodi, E., Riguzzi, F.: Expectation Maximization over Binary Decision Diagrams for probabilistic logic programs. Intell. Data Anal. 17(2), 343–363 (2013)
2. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theor. Pract. Log. Prog. 15(2), 169–212 (2015)
3. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming - Theory and Applications, LNCS, vol. 4911. Springer (2008)
4. Fierens, D., den Broeck, G.V., Renkens, J., Shterionov, D.S., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theor. Pract. Log. Prog. 15(3), 358–401 (2015)
5. Goodman, N.D., Tenenbaum, J.B.: Probabilistic models of cognition, `http://probmods.org`
6. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Probabilistic logic programming on the web. Software Pract. and Exper. (2015)
7. Riguzzi, F., Cota, G.: Probabilistic logic programming tutorial. The Association for Logic Programming Newsletter 29(1) (March/April 2016), `http://www.cs.nmsu.edu/ALP/2016/03/probabilistic-logic-programming-tutorial/`
8. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP-95. pp. 715–729. MIT Press, Cambridge, Massachusetts (1995)

[8] `http://cplint.lamping.unife.it/example/learning/exauc.pl`