Probabilistic Inductive Logic Programming

Fabrizio Riguzzi
Dipartimento di Matematica e Informatica
Università di Ferrara
fabrizio.riguzzi@unife.it

Abstract

The field of Probabilistic Logic Programming (PLP) has seen significant advances in the last 20 years, with many proposals for languages that combine probability with logic programming. Since the start, the problem of learning probabilistic logic programs has been the focus of much attention and a special issue of Theory and Practice of Logic Programming on Probability, Logic, and Learning has just appeared online. Learning PLP represents a whole subfield of Inductive Logic Programming (ILP). In Probabilistic ILP (PILP) two problems are considered: learning the parameters of a program given the structure (the rules) and learning the structure and the parameters at the same time. Usually structure learning systems use parameter learning as a subroutine. In this article we present the field of PILP and discuss the main results.

1 Introduction

Probabilistic Logic Programming (PLP) started in the early nineties with seminal works such as those of Dantsin [6], Ng and Subrahmanian [17], Poole [19] and Sato [23].

Since then the field has steadily developed and many proposals for the integration of logic programming and probability have appeared. These proposals can be grouped in two classes: those that use a variant of the distribution semantics [23] and those that follow a Knowledge Base Model Construction approach [27, 1]. The distribution semantics underlines many languages such as Probabilistic Logic Programs [6], Probabilistic Horn Abduction [19], Independent Choice Logic [18], PRISM [23], pD [12], Logic Programs with Annotated Disjunctions [26], P-log [5], ProbLog [8] and CP-logic [25]. While the number of languages is large, all these approaches share a common approach so that there are transformation with linear complexity that can translate one language in another. Under the distribution semantics, a probabilistic logic program defines a probability distribution over normal logic programs (termed *worlds*). The probability of a ground query Q is then obtained from the joint distribution of the query and the worlds: it is the sum of the probability of worlds where the query is true.

The languages following a KBMC approach include CLP(BN) [22], Bayesian Logic Programs [15] and the Prolog Factor Language [13]. In this languages, a program is a template for generating a ground graphical model, be it a Bayesian network or a Markov network.

Learning probabilistic logic programs has been considered from the start: [23] already presented an algorithm for learning the parameters of programs under the distribution semantics. This is the first problem that was considered in the domain of learning and was the only one until recently, when works regarding the induction of the structure (the rules) and the parameters at the same time started to appear. The whole field was called Probabilistic Inductive Logic Programming (PILP) in [20].

In the following we provide an overview of PILP by concentrating on languages under the distribution semantics.

2 Language

We illustrate the distribution semantics with ProbLog [8], the language with the simplest syntax. A ProbLog program consists of a set of rules (certain) and a set of *probabilistic facts* of the form

$$p_i :: A_i.$$

where $p_i \in [0, 1]$ and A_i is an atom, meaning that each ground instantiation $A_i\theta$ of A_i is true with probability p_i and false with probability $1 - p_i$. From a ProbLog program we obtain normal programs called worlds by including the set C of rules and a subset L of the (ground) probabilistic facts. Each world is obtained by selecting or rejecting each grounding of each probabilistic fact. The probability of a world is given by the product of a factor p_i for each grounding of a probabilistic fact $p_i :: A_i$ included in the world and of a factor $1-p_i$ for each grounding of a probabilistic fact not included in the world. The probability of a ground atom (query) is then the sum of the probabilities of the worlds where the query is true.

Example 1 *The following program encodes the fact that a person sneezes if he has the flu and this is the active cause of sneezing, or if he has hay fever and hay fever is the active cause for sneezing:*

$$\begin{array}{l} sneezing(X) \text{ :- } flu(X), fluSneezing(X). \\ sneezing(X) \text{ :- } hayFever(X), hayFeverSneezing(X). \\ flu(bob). \\ hayFever(bob). \\ 0.7 :: fluSneezing(X). \\ 0.8 :: hayFeverSneezing(X). \end{array}$$

This program has 4 worlds, $sneezing(bob)$ is true in 3 of them and its probability is $0.7 \times 0.8 + 0.3 \times 0.8 + 0.7 \times 0.2 = 0.94$.

The problem of computing the probability of queries is called inference. Solving it by computing all the worlds and then identifying those that entail the query is impractical as the number of possible worlds is exponential in the number of ground probabilistic facts. Usually inference is performed by resorting to knowledge compilation [7]: the program and the query are compiled into a language that allows an efficient computation of the probability. The first approach for performing inference [8] required finding a covering set of explanations for the query. An explanation is a minimal set of probabilistic facts that is sufficient

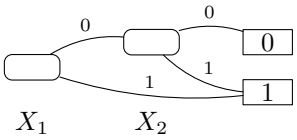
for entailing the query and a covering set of explanations is a set that contains all possible explanations for the query. From the set of explanations, a Boolean formula in Disjunctive Normal Form (DNF) can be obtained, where each probabilistic fact is associated to a Boolean variable, an explanation is the conjunction of the facts it contains and the whole formula is the disjunction of the formulas for the different explanations.

In Example 1, if we associated Boolean variable X_1 to $fluSneezing(bob)$ and X_2 to $hayFeverSneezing(bob)$ the Boolean formula that encodes the set of explanations is $X_1 \vee X_2$.

Computing the probability of a Boolean formula in DNF is an NP-hard problem. With knowledge compilation, we transform the formula so that the computation of the probability is polynomial in the size of the obtained representation. The hardness is transferred to the compilation phase but this approach works well in practice because considerable effort has been devoted to the development of efficient compilers.

A target language for knowledge compilation is that of Binary Decision Diagrams (BDDs). They are rooted graphs with one level for each Boolean variable. A node n in a BDD has two children: one corresponding to the 1 value of the variable associated with the level of n and one corresponding the 0 value of the variable. The leaves store either 0 or 1.

A BDD for the function $X_1 \vee X_2$ is shown below



From a BDD we can compute the probability of the query with a dynamic programming algorithm that is linear in the size of the BDD.

Another target language is that of deterministic, decomposable negation normal form (d-DNNF): a d-DNNF is a rooted directed acyclic graph in which each leaf node is labeled with a literal and each internal node is labeled with a conjunction or disjunction. The graphs must moreover satisfy a number of restrictions. d-DNNF has been used in [11] for performing inference with Problog: again, once the d-DNNF has been built, computing the probability is linear in the size of the graph.

3 Learning

The problem that PILP aims at solving can be expressed as

- Given
  - a background knowledge as a probabilistic logic program B
  - a set of positive and negative examples E^+ and E^-
  - a language bias $\mathcal{L}$
- Find

– a probabilistic logic program P such that the probability of positive examples according to $P \cup B$ is maximized and the probability of negative examples is minimized.

This problem has two variants: parameter learning and structure learning. In the first, we are given the structure (the rules) of P and we just want to infer the parameters of P, while in the second we want to infer both the structure and the parameters of P.

Parameter learning for languages following the distribution semantics has been performed by using the Expectation Maximization (EM) algorithm or by gradient descent. The EM algorithm is used to estimate the probability of models containing random variables that are not observed in the data. It consists of a cycle in which the steps of expectation and maximization are repeatedly performed. In the expectation step, the distribution of the hidden variables is computed according to the current values of the parameters, while in the maximization step the new values of the parameters are computed. Examples of EM algorithms are PRISM [24], LFI-Problog [11] and EMBLEM [3]. The latter two use knowledge compilation for computing the distribution of the hidden variables. RIB [21] is a system for parameter learning that uses a special EM algorithm called information bottleneck that was shown to avoid some local maxima of EM.

Gradient descent methods compute the gradient of the target function and iteratively modify the parameters moving in the direction of the gradient. An example of these methods is LeProbLog [14] that uses a dynamic programming algorithm for computing the gradient exploiting BDDs.

Structure learning is yet relatively unexplored. One of the first works [9] presented an algorithm for performing theory compression on ProbLog programs. Theory compression means removing as many clauses as possible from the theory in order to maximize the probability. No new clause can be added to the theory.

SEM-CP-logic [16] learns parameters and structure of ground CP-logic programs. It performs learning by considering the Bayesian networks equivalent to CP-logic programs and by applying techniques for learning Bayesian networks.

SLIPCASE [2] learns Logic Programs with Annotated Disjunctions by iteratively refining theories and learning the parameters of each theory with EMBLEM. This is possible as parameter learning is usually fast. SLIPCOVER [4] (in the TPLP special issue) is an evolution of SLIPCASE that uses bottom clauses to guide the refinement process, thus reducing the number of revisions and exploring more effectively the search space. Moreover, SLIPCOVER separates the search for promising clauses from that of the theory: the space of clauses is explored with beam search, while the space of theories is searched greedily.

ProbFOIL [10] combines the rule learner FOIL with ProbLog. Logical rules are learned from probabilistic data in the sense that both the examples themselves and their classifications can be probabilistic. In this setting the parameters (the probability values) are fixed and the structure (the rules) are to be learned.

4 Conclusions

While this article does not aim at being a complete account of the activity in the field of PLP, we hope to have given an introduction that highlights the important results already achieved. There are many avenues for future research: improving the efficiency of inference, that is a basic component of learning systems, and developing faster and more accurate learning systems. These challenges must be taken up for PLP systems to increase their adoption.

References

[1] Fahiem Bacchus. Using first-order probability logic for the construction of bayesian networks. In *UAI 1993*, pages 219–226. Morgan Kaufmann, 1993.

[2] Elena Bellodi and Fabrizio Riguzzi. Learning the structure of probabilistic logic programs. In *ILP 2011*, volume 7207 of *LNCS*, pages 61–75. Springer, 2011.

[3] Elena Bellodi and Fabrizio Riguzzi. Expectation Maximization over binary decision diagrams for probabilistic logic programs. *Intelligent Data Analysis*, 17(2):343–363, 2013.

[4] Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, FirstView Articles(CoRR abs/1309.2080), 2014.

[5] C.Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(1):57–144, 2009.

[6] Evgeny Dantsin. Probabilistic logic programs and their semantics. In *Russian Conference on Logic Programming*, volume 592 of *LNCS*, pages 152–164. Springer, 1991.

[7] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intellingence Research*, 17:229–264, 2002.

[8] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI 2007*, pages 2462–2467, 2007.

[9] Luc De Raedt, Kristian Kersting, Angelika Kimmig, Kate Revoredo, and Hannu Toivonen. Compressing probabilistic Prolog programs. *Machine Learning*, 70:151–168, 2008.

[10] Luc De Raedt and Ingo Thon. Probabilistic rule learning. In *ILP 2010*, volume 6489 of *LNCS*, pages 47–58. Springer, 2010.

[11] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *CoRR*, abs/1304.6810, 2013.

[12] Norbert Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[13] Tiago Gomes and Vítor Santos Costa. Evaluating inference algorithms for the prolog factor language. In *ILP 2012*, volume 7842 of *LNCS*, pages 74–85. Springer, 2012.

[14] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter learning in probabilistic databases: A least squares approach. In *ECML/PKDD 2008*, volume 5211 of *LNCS*, pages 473–488. Springer, 2008.

[15] Kristian Kersting and Luc De Raedt. Towards combining inductive logic programming with bayesian networks. In *ILP 2001*, volume 2157 of *LNCS*, pages 118–131. Springer, 2001.

[16] W. Meert, J. Struyf, and H. Blockeel. Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. *Fundumanta Informaticae*, 89(1):131–160, 2008.

[17] Raymond Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and computation*, 101(2):150–201, 1992.

[18] D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[19] David Poole. Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. *New Generation Computing*, 11(3):377–400, 1993.

[20] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *ALT 2004*, volume 3244 of *LNCS*, pages 19–36. Springer, 2004.

[21] Fabrizio Riguzzi and Nicola Di Mauro. Applying the information bottleneck to statistical relational learning. *Machine Learning*, 86(1):89–114, 2012.

[22] Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. Clp(bn): Constraint logic programming for probabilistic knowledge. In *UAI 2003*, pages 517–524. Morgan Kaufmann, 2003.

[23] T. Sato. A statistical learning method for logic programs with distribution semantics. In *ICLP 1995*, pages 715–729, 1995.

[24] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[25] J. Vennekens, Marc Denecker, and Maurice Bruynooghe. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009.

[26] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *International Conference on Logic Programming*, volume 3131 of *LNCS*, pages 195–209. Springer, 2004.

[27] Michael P Wellman, John S Breese, and Robert P Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(01):35–53, 1992.