Inference and Learning with Model Uncertainty in Probabilistic Logic Programs *

Victor Verreet[1] Vincent Derkinderen[1] Pedro Zuidberg Dos Martires[1] Luc De Raedt[1,2]
[1]Department of Computer Science, KU Leuven, Belgium
[1]Leuven.AI - KU Leuven Institute for AI, Belgium
[2]Center for Applied Autonomous Systems, Örebro University, Sweden
{victor.verreet,vincent.derkinderen,pedro.zudo,luc.deraedt}@kuleuven.be

An issue that has received limited attention in probabilistic logic programming (PLP) is the modeling of so-called *epistemic uncertainty*, the uncertainty about the model itself. Accurately quantifying this uncertainty is paramount to robust inference, learning and ultimately decision making. We introduce BetaProbLog, a PLP language that can model epistemic uncertainty. BetaProbLog has sound semantics, an effective inference algorithm that combines Monte Carlo techniques with knowledge compilation, and a parameter learning algorithm.

1 Introduction

Uncertainty is either aleatoric or epistemic in nature [3]. The former is the intrinsic uncertainty that a probabilistic model brings that cannot be reduced with further observations of the world. For example, a coin toss outcome is intrinsically uncertain. In contrast, epistemic uncertainty stems from the lack of knowledge about the true model and can be reduced with more observations. Explicitly modeling both uncertainty types allows for gauging the uncertainty of learned probabilistic models and reasoning over the robustness of subsequent predictions. Risky decisions can be avoided when a learner has uncertain results. It also allows for probabilistic inference even when the probabilistic model is not exactly known.

We introduce BetaProbLog, an extension of the probabilistic logic programming (PLP) language ProbLog. BetaProbLog has a well-defined semantics rooted in probability theory. With so-called *second-order queries*, BetaProbLog calculates expectation values, offering a vast range of query types. We provide an easy to interpret Monte Carlo inference algorithm for BetaProbLog based on knowledge compilation combined with parallelized tensor operations. This leads to an inference algorithm that is more than an order of magnitude faster than the one proposed by [1]. In addition, we tackle parameter learning with epistemic uncertainty. To our knowledge, learning in second-order networks has so far been limited to networks of only two nodes [4]. We empirically evaluate our work on probabilistic inference tasks in second-order Bayesian networks, digit classification, and by performing parameter learning in the presence of epistemic uncertainty. The details of this can be found in our full paper [5].

2 BetaProbLog

BetaProbLog is an extension of ProbLog, a probabilistic logic language based on Prolog [2]. A BetaProbLog program consists of three disjoint sets $\mathscr{F}$, $\mathscr{B}$, and $\mathscr{R}$:

*This is an extended abstract of a paper originally published at the 36th AAAI Conference on Artificial Intelligence (AAAI22): https://www.aaai.org/AAAI22Papers/AAAI-3743.VerreetV.pdf

Submitted to:
ICLP 2022

1. a set of probabilistic facts $\mathscr{F}$. Given a probabilistic fact p_f::f, the atom f has probability p_f of being true in a world of the program.
2. a set of beta facts $\mathscr{B}$. A fact beta(α_f,β_f)::f defines a random variable $X_f \sim \text{Beta}(\alpha_f, \beta_f)$ representing the probability of the atom f.
3. a set of logic rules $\mathscr{R}$ of the form h :- b_1,...,b_n making up a stratified normal logic program.

An example BetaProbLog program is shown in Example 1. Note that in the absence of beta facts ($\mathscr{B} = \emptyset$) BetaProbLog reduces to ProbLog. A unique stable model $\Pi(\mathscr{H})$ of a BetaProbLog program results from assigning a truth value to each probabilistic atom $\mathscr{H} \subseteq \mathscr{F} \cup \mathscr{B}$, and applying $\mathscr{R}$ to derive the truth value of the derived atoms. The probability of query atom q is itself a random variable X_q, defined as $X_q = \sum_{\mathscr{H} \subseteq \mathscr{F} \cup \mathscr{B}, q \in \Pi(\mathscr{H})} \prod_{f \in \mathscr{H}} w(f) \prod_{f \in (\mathscr{F} \cup \mathscr{B}) \setminus \mathscr{H}} w(\neg f)$ with $w(f) = p_f$ if f is a non-negated fact and $w(f) = X_f$ if it is a non-negated beta fact, and in case of negation $w(\neg f) = 1 - w(f)$.

Alongside a BetaProbLog program, the user specifies a second-order query $Q = (q, g)$, with g a function $g : [0,1] \to \mathbb{R}$ operating on the probability X_q of atom q being satisfied. The answer $A(Q)$ to the query Q is the expected value, denoted with $\mathbb{E}$, of $g(X_q)$: $A(Q) = \mathbb{E}_{p(X)}[g(X_q)] = \int g(X_q) \left(\prod_{i=1}^{|\mathscr{B}|} p(X_{f_i}) \right) \mathrm{d}X$.

Example 1 *A BetaProbLog program with two beta facts, one probabilistic fact and two clauses.*

```
1  beta(40, 160)::burglary.
2  beta(10, 90)::earthquake.
3  0.8::alarm_on.
4
5  alarm :- alarm_on, burglary.
6  alarm :- alarm_on, earthquake.
```

To compute the probability $P(X_{alarm} < 0.3)$, we can use the second order query $Q = (alarm, g)$ with an indicator $g(X_q) = [1 \text{ if } X_q < 0.3 \text{ else } 0]$.

Note that when calculating the mean value of X_q, by choosing $g(X_q) = X_q$, the second-order query reduces to a regular ProbLog inference problem. Higher moments of the distribution can be calculated with $g(X_q) = X_q^k$ for the kth moment. This subsequently allows the calculation of other statistics such as the variance or skewness of the distribution.

3 Inference and Learning

Both ProbLog and BetaProbLog perform probabilistic inference by solving a weighted model counting task, converting the queried program into a compiled circuit compactly representing X_q (Figure 1). The integral of the second order query is hard to solve in general but can be approximated by Monte Carlo sampling. Instead of sampling directly from $X_f \sim \text{Beta}(\alpha_f, \beta_f)$ we consider $U_f \sim \text{Uniform}(0,1)$ and apply a reparametrization function r to such that $r(U_f, \alpha_f, \beta_f) \sim \text{Beta}(\alpha_f, \beta_f)$. This allows us to calculate gradients with respect to the distribution parameters, which enables learning.

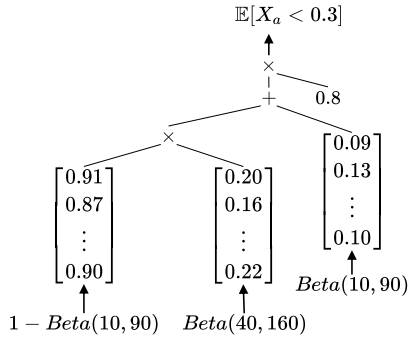

Figure 1: Circuit example with samples

BetaProbLog performs parameter learning by minimizing a loss function. The total loss $\mathscr{L}$ is a sum of loss functions L applied to a second-order query, which is compared to a target value. More concretely, given a BetaProbLog program with a set of probabilistic or distributional parameters θ, a dataset $\mathscr{D}$ of tuples (d,t) with d a partially observed model of the program and t a target value of $A(Q_d, \theta)$, the total loss is $\mathscr{L}(\theta) = \sum_{(d,t) \in \mathscr{D}} L(A(Q_d, \theta), t)$ where we explicitly write the dependency of $A(\cdot)$ on θ as we want to learn these parameters.

4 Experiments

We summarize the results of our experiments below, details are available in the full paper [5].

Question 1: Can BetaProbLog capture the uncertainty with respect to the true model? Following [1]'s experiment, we analyse how well the spread between the inferred and the actual probability is captured. The results were positive, even when the inference algorithm was limited to few samples.

Question 2: Does the inference computation time scale? We investigate BetaProbLog's run time on several translated Bayesian networks (BN). Results show our approach scales very well. For example Hepar2, a BN with 10^3 parameters, took 1.5s to execute. Due to the parallelization the evaluation time remained relatively constant regardless of using 10^1 or 10^4 samples.

Question 3: Can BetaProbLog recover parameters from target probabilities? This learning task aims to recover the network's missing beta parameters such that the given queries match their target probability. For this we utilise BetaProbLog's parameter learning approach, minimizing the error between the inferred probability of q and the target probability t with the loss function $(A(Q,\theta)-t)^2$ where $Q=(q,I)$ and I the identity function. Given enough data, BetaProbLog recovers the parameters.

5 Conclusion

Modeling epistemic uncertainty allows for reasoning even when the underlying probabilistic model is not exactly known. We extended the PLP language ProbLog with sound semantics to support epistemic uncertainty through beta distributions. These semantics are implemented in the language BetaProbLog for which we also provide an inference and learning algorithm.

Acknowledgements

VV received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. VD is supported by an SB PhD fellowship at the Research Foundation – Flanders [1SA5520N]. PZD has received support from the KU Leuven Special Research Fund. LDR is also funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant No [694980]) and the Wallenberg AI, Autonomous Systems and Software Program. The authors thank Angelika Kimmig for insightful discussions.

References

[1] F. Cerutti, L. M. Kaplan, A. Kimmig & M. Sensoy (2019): *Probabilistic Logic Programming with Beta-Distributed Random Variables*. In: *AAAI*, AAAI Press, pp. 7769–7776, doi:10.1609/aaai.v33i01.33017769.

[2] D. Fierens, G. Van den Broeck, J. Renkens, D. Sht. Shterionov, B. Gutmann, I. Thon, G. Janssens & L. De Raedt (2015): *Inference and learning in probabilistic logic programs using weighted Boolean formulas*. *Theory Pract. Log. Program.* 15(3), pp. 358–401, doi:10.1017/S1471068414000076.

[3] E. Hüllermeier & W. Waegeman (2021): *Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods*. *Mach. Learn.* 110(3), pp. 457–506, doi:10.1007/s10994-021-05946-3.

[4] L. Kaplan, F. Cerutti, M. Şensoy & K. Vijay Mishra (2020): *Second-Order Learning and Inference using Incomplete Data for Uncertain Bayesian Networks: A Two Node Example*. In: *FUSION*, IEEE, pp. 1–8, doi:10.23919/FUSION45008.2020.9190472.

[5] V. Verreet, V. Derkinderen, P. Zuidberg Dos Martires & L. De Raedt (2022): *Inference and Learning with Model Uncertainty in Probabilistic Logic Programs*. In: *AAAI*, AAAI Press.