Learning the Structure of Probabilistic Logic Programs

Elena Bellodi and Fabrizio Riguzzi

ENDIF – University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy
{elena.bellodi,fabrizio.riguzzi}@unife.it

1 Introduction

The ability to model both complex and uncertain relationships among entities is very important for learning accurate models of many domains. This originated a growing interest in the field of Probabilistic Inductive Logic Programming, which is based on languages that integrate logic programming and probability. Many of these languages are based on the distribution semantics [11], which underlies, e.g., PRISM, the Independent Choice Logic, Logic Programs with Annotated Disjunctions (LPADs) [12], ProbLog [3], CP-logic and others.

Recently approaches for learning the parameters of these languages have been proposed: LeProbLog [5] uses gradient descent while CoPrEM [7] and EMBLEM [2] use an Expectation Maximization approach in which the expectations are computed directly using Binary Decision Diagrams (BDD).

In this paper we present the algorithm SLIPCASE for "Structure LearnIng of ProbabilistiC logic progrAmS with Em over bdds". It performs a beam search in the space of LPADs using the log likelihood of the data as the guiding heuristics. To estimate the log likelihood of theory refinements it performs a limited number of Expectation Maximization iterations of EMBLEM.

2 Probabilistic Logic Programming

The distribution semantics [11] is one of the most interesting approaches to the integration of logic programming and probability. It was introduced for the PRISM language but is shared by many other languages. In this paper we will use LPADs for their general syntax. We review here the semantics for the case of no function symbols for the sake of simplicity.

Formally a *Logic Program with Annotated Disjunctions* [12] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} : -b_{i1}, \ldots, b_{im_i}$. In such a clause $h_{i1}, \ldots h_{in_i}$ are logical atoms and $b_{i1}, \ldots, b_{im_i}$ are logical literals, $\{\Pi_{i1}, \ldots, \Pi_{in_i}\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. $b_{i1}, \ldots, b_{im_i}$ is called the *body* and is indicated with $body(C_i)$. Note that if $n_i = 1$ and $\Pi_{i1} = 1$ the clause corresponds to a non-disjunctive clause. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$ the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does

not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

An *atomic choice* is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, n_i\}$. In practice $C_i\theta_j$ corresponds to a random variable X_{ij} and an atomic choice (C_i, θ_j, k) to an assignment $X_{ij} = k$. A set of atomic choices κ is *consistent* if only one head is selected for a ground clause. A *composite choice* κ is a consistent set of atomic choices. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$.

A *selection* σ is a composite choice that, for each clause $C_i\theta_j$ in $ground(T)$, contains an atomic choice (C_i, θ_j, k). A selection σ identifies a normal logic program w_σ defined as $w_\sigma = \{(h_{ik} \leftarrow body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. w_σ is called a *world* of T. Since selections are composite choices, we can assign a probability to worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$. We denote the set of all worlds of a program by W.

We consider only *sound* LPADs in which every possible world has a total well-founded model. We write $w_\sigma \models Q$ to mean that the query Q is true in the well-founded model of the program w_σ.

Let $P(W)$ be the distribution over worlds. The probability of a query Q given a world w is $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of a query Q is given by

$$P(Q) = \sum_{w\in W} P(Q, w) = \sum_{w\in W} P(Q|w)P(w) = \sum_{w\in W: w\models Q} P(w) \qquad (1)$$

Example 1. The following LPAD T encodes a very simple model of the development of an epidemic or pandemic:

$C_1 = epidemic : 0.6; pandemic : 0.3 : -flu(X), cold.$

$C_2 = cold : 0.7.$

$C_2 = flu(david).$

$C_3 = flu(robert).$

This program models the fact that if somebody has the flu and the climate is cold, there is the possibility that an epidemic or a pandemic arises.

It is often unfeasible to find all the instances where the query is true so inference algorithms find instead *explanations* for the query, i.e. composite choices such that the query is true in all the worlds whose selections are a superset of them. Explanations however, differently from possible worlds, are not necessarily mutually exclusive with respect to each other, so the probability of the query can not be computed by a summation as in (1). The explanations have first to be made disjoint so that a summation can be computed. To this purpose Binary Decision Diagrams are used.

EMBLEM [2] applies the algorithm for performing EM over BDDs proposed in [8] to the problem of learning the parameters of an LPAD. EMBLEM takes as input a number of goals that represent the examples. For each goal it generates the BDD encoding its explanations. The typical input for EMBLEM will be a

set of interpretations, i.e., sets of ground facts, each describing a portion of the domain of interest. The user has to indicate which, among the predicates of the input facts. are target predicates: the facts for these predicates will then form the queries for which the BDDs are built. The predicates can be treated as closed--world or open-world. In the first case the body of clauses is resolved only with facts in the interpretation. In the second case, the body of clauses is resolved both with facts in the interpretation and with clauses in the theory. If the last option is set and the theory is cyclic, we use a depth bound on SLD-derivations to avoid going into infinite loops, as proposed by [6]. Then EMBLEM enters the EM cycle, in which the steps of expectation and maximization are repeated until the log-likelihood of the examples reaches a local maximum. Expectations are computed directly over BDDs using the algorithm of [8].

3 SLIPCASE

SLIPCASE learns an LPAD by starting from an initial theory and by performing a beam search in the space of refinements of the theory guided by the log likelihood of the data.

First the parameters of the initial theory are computed using EMBLEM and the theory is inserted in the beam (see Algorithm 1). Then an iteration is entered in which at each step the theory with the highest log likelihood is removed from the beam. Such a theory is the first of the beam since the theories are kept ordered.

Then SLIPCASE finds the set of refinements of the selected theory that are allowed by the language bias. `modeh` and `modeb` declarations in Progol style are used to this purpose. The refinements considered are: the addition of a literal to a clause, the removal of a literal from a clause, the addition of a clause with an empty body and the removal of a clause. The refinements must respect the input-output modes of the bias declarations and the resulting clauses must be connected.

For each refinement an estimate of the log likelihood of the data is computed by running the procedure BoundedEMBLEM (see Algorithm 2) that performs a limited number of Expectation-Maximization steps. BoundedEMBLEM differs from EMBLEM only in line 10, where it imposes that the iterations are at most $NMax$.

Once the log likelihood for each refinement is computed, the best theory found so far is possibly updated and each refinement is inserted in order in the beam. At the end the parameters of the best theory found so fare are computed with EMBLEM and the resulting theory is returned.

We decided to follow this approach rather than using a scoring function as proposed in [4] for Structural EM (SEM) because we found that using the log likelihood was giving better results with a limited additional cost. We think that is due to the fact that, while in SEM the number of incomplete or unseen variables is fixed, in SLIPCASE the revisions can introduce or remove unseen variables from the underlying Bayesian network.

Algorithm 1 Procedure SLIPCASE

```
1: function SLIPCASE(Th, MaxSteps, ε, δ, b, NMax)
2:     Build BDDs
3:     (LL, Th) =EMBLEM(Th, ε,δ)
4:     Beam = [(Th, LL)]
5:     BestLL = LL
6:     BestTh = Th
7:     Steps = 1
8:     repeat
9:         Remove the first couple (Th, LL) from Beam
10:        Find all refinements Ref of Th
11:        for all Th' in Ref do
12:            (LL'', Th'') =BoundedEMBLEM(Th', ε, δ, NMax)
13:            if LL'' > BestLL then
14:                Update BestLL, BestTh
15:            end if
16:            Insert (Th'', LL'') in Beam in order of LL''
17:            if size(Beam) > b then
18:                Remove the last element of Beam
19:            end if
20:        end for
21:        Steps = Steps + 1
22:    until Steps > MaxSteps or Beam is empty
23:    (LL, ThMax) =EMBLEM(BestTh, ε,δ)
24:    return ThMax
25: end function
```

Algorithm 2 Procedure BoundedEMBLEM

```
1: function BoundedEMBLEM(Theory, ε, δ, NMax)
2:     Build BDDs
3:     LL = −inf
4:     N = 0
5:     repeat
6:         LL_0 = LL
7:         LL = Expectation(BDDs)
8:         Maximization
9:         N = N + 1
10:    until LL − LL_0 < ε ∨ LL − LL_0 < −LL · δ ∨ N > NMax
11:    Update the parameters of Th
12:    return LL, Th
13: end function
```

4 Experiments

We implemented SLIPCASE Yap Prolog[1] and we tested it on two real world datasets: HIV[1] and UW-CSE[2]. We compared SLIPCASE with SEM-CP-logic [10] and with the algorithm for learning Markov Logic Networks using Structural Motifs (LSM) [9]. All experiments were performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM.

SLIPCASE offers the following options: putting a limit on the depth of derivations, necessary for problems that contain cyclic clauses; setting the number of iterations $NMax$ for BoundedEMBLEM; setting the size of the beam, the greatest number of variables in a learned rule (max_var) and of rules (max_rules) in the learned theory.

[1] `http://www.dcc.fc.up.pt/~vsc/Yap/`

[2] `http://alchemy.cs.washington.edu/data/uw-cse`

For all experiments with SLIPCASE we used a beam size of 5, max_var=5, max_rules=10 and $NMax = +\infty$ since we observed that EMBLEM usually converged quickly.

The HIV dataset records mutations in HIV's reverse transcriptase gene in patients that are treated with the drug zidovudine. It contains 364 examples, each containing facts for six classical zidovudine mutations. The goal is to discover causal relations between the occurrence of mutations in the virus, so all the predicates were set as target. The input initial theory was composed of six probabilistic clauses of the form $target\_mutation : 0.2$. The language bias allows each atom to appear in the head and in the body. We used a five-fold cross-validation approach, by considering a single fold as the grouping of 72 or 73 examples.

We ran SLIPCASE with a depth bound equal to three and obtained a final structure with 6 rules for each fold. For SEM-CP-logic, we tested the theory obtained by it that is reported in [1] over each of the five folds. For LSM, we used the generative training algorithm to learn weights, because all the predicates were considered as target, with the option `-queryEvidence` (to mean that all the groundings of the query predicates not in the database are assumed false evidence), and the MC-SAT algorithm for inference over the test fold, by specifying all the six mutations as query atoms. Table 1 shows the AUCPR and AUCROC averaged over the five folds for the algorithms.

The UW-CSE dataset contains information about the Computer Science department of the University of Washington, and is split into five mega-examples, each containing facts for a particular research area. The goal is to predict the `advisedBy/2` predicate, namely the fact that a person is advised by another person: this was our target predicate. The input theory for SLIPCASE was composed by two clauses of the form $advisedby(X, Y) : 0.5$.. We used a five-fold cross-validation approach. We ran SLIPCASE with no depth bound. For LSM, we used the preconditioned rescaled conjugate gradient discriminative training algorithm to learn weights, by specifying `advisedby/2` as the only non-evidence predicate plus the option `-queryEvidence`, and the MC-SAT algorithm for inference over the test fold, by specifying `advisedby/2` as the query predicate. Table 1 shows the average AUCPR and AUCROC for both the algorithms[3], while table 2 shows the learning times in hours for both datasets.

As the tables show, SLIPCASE was able to achieve higher AUCPR and AUCROC with respect to LSM on both datasets and with respect to SEM-CP-logic on the HIV dataset. These results were also achieved using similar or smaller computation time with respect to LSM.

References

1. Beerenwinkel, N., Rahnenführer, J., Däumer, M., Hoffmann, D., Kaiser, R., Selbig, J., Lengauer, T.: Learning multiple evolutionary pathways from cross-sectional

[3] On the first fold of UW-CSE LSM gave a segmentation error so the averages are computed over the remaining four folds.

Table 1. Results of the experiments on all datasets in terms of the average of the Area Under the PR Curve and under the ROC Curve.

Dataset	AUCPR			AUCROC		
	Slipcase	LSM	SEM-CP-logic	Slipcase	LSM	SEM-CP-logic
HIV	0.623	0.381	0.579	0.733	0.652	0.721
UW-CSE	1	0.0117	-	1	0.518	-

Table 2. Execution time in hours of the experiments on all datasets.

Dataset	Time(h)	
	Slipcase	LSM
HIV	0.010	0.003
UW-CSE	0.040	2.520

data. Journal of Computational Biology 12(6), 584–598 (2005)

2. Bellodi, E., Riguzzi, F.: Expectation Maximization over binary decision diagrams for probabilistic logic programs. Intel. Data Anal. (to appear)
3. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence. pp. 2462–2467. AAAI Press (2007)
4. Friedman, N.: The Bayesian structural EM algorithm. In: Conference on Uncertainty in Artificial Intelligence. pp. 129–138. Morgan Kaufmann (1998)
5. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: European Conference on Machine Learning and Knowledge Discovery in Databases. LNCS, vol. 5211, pp. 473–488. Springer (2008)
6. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.: Parameter estimation in ProbLog from annotated queries. Tech. Rep. CW 583, KU Leuven (2010)
7. Gutmann, B., Thon, I., De Raedt, L.: Learning the parameters of probabilistic logic programs from interpretations. Tech. Rep. CW 584, KU Leuven (2010)
8. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. In: Late Breaking Papers of the International Conference on Inductive Logic Programming. pp. 44–49 (2008)
9. Kok, S., Domingos, P.: Learning markov logic networks using structural motifs. pp. 551–558. Omnipress (2010)
10. Meert, W., Struyf, J., Blockeel, H.: Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. Fundamenta Informaticae 89(1), 131–160 (2008)
11. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
12. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: International Conference on Logic Programming. LNCS, vol. 3131, pp. 195–209. Springer (2004)