SLIPCOVER at Inductive Logic Programming Competition: Probabilistic Structure Learning by Searching the Clause Space

Elena Bellodi[1], Fabrizio Riguzzi[2], and Riccardo Zese[1]

[1] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy

Abstract. SLIPCOVER ("Structure LearnIng of Probabilistic logic programs by searChing OVER the clause space") performs structure and parameter learning of probabilistic logic programs. It searches the space of clauses storing all the promising ones, dividing them into clauses for target predicates (those we want to predict) and clauses for background predicates (the remaining ones), with a discriminative approach. Then it performs a greedy search in the space of theories, by trying to add each clause for a target predicate to the current theory. Finally, it adds all background clauses and performs parameter learning.

1 Structure Learning by means of SLIPCOVER

SLIPCOVER [2] learns a Logic Program with Annotated Disjunctions (LPAD) [4] by first identifying good candidate clauses and then searching for a theory guided by the Log-Likelihood (LL) of the data. It takes as input a set of mega-examples and an indication of which predicates are target, i.e., those for which we want to optimize the predictions of the final theory. The mega-examples must contain positive and negative examples for all predicates that may appear in the head of clauses, either target or non-target (background predicates). The mega-examples are sets of ground facts describing a portion of the domain of interest and must contain also negative facts for target predicates, expressed as *neg(atom)*.

The search over the space of clauses is performed according to a language bias expressed by means of mode declarations. Following [3], a mode declaration m is either a head declaration *modeh(r,s)* or a body declaration *modeb(r,s)*, where s is a a template for literals in the head or body of a clause and r is the recall.

2 The Algorithm in Short

SLIPCOVER's main function is shown in Algorithm 1: after the search in the space of clauses, it performs a greedy search in the space of theories.

The first phase aims at searching in the space of clauses in order to find a set of promising ones (in terms of LL), that will be employed in the subsequent greedy search phase. For each clause, its parameters are learned using the system EMBLEM [1,**?**] which also computes the LL. By starting from promising clauses, the greedy search is able to generate good final theories.

The second phase is a greedy search in the space of theories starting with an empty theory Th with the lowest value of LL. Then one target clause Cl at a time is added and the parameters of the new theory are learned by means of EMBLEM. If the LL is better than the current best, the clause is kept in the theory, otherwise it is discarded.

Finally, all the background clauses are added to the theory and parameter learning is performed.

Algorithm 1 Function SLIPCOVER

```
1: function SLIPCOVER(I, NInt, NS, NA, NI, NV, NB, NTC, NBC, ϵ, δ)
2:     IBs =InitialBeams(I, NInt, NS, NA)                          ▷ Clause search
3:     TC ← []
4:     BC ← []
5:     for all (PredSpec, Beam) ∈ IBs do
6:         Steps ← 1
7:         NewBeam ← []
8:         repeat
9:             while Beam is not empty do
10:                Remove the first couple ((Cl, Literals), LL) from Beam      ▷ Remove the first clause
11:                Refs ←ClauseRefinements((Cl, Literals), NV)  ▷ Find all refinements Refs of (Cl, Literals) with at most NV variables
12:                for all (Cl′, Literals′) ∈ Refs do
13:                    (LL″, {Cl″}) ←EMBLEM(I, {Cl′}, ϵ, δ)
14:                    NewBeam ←Insert((Cl″, Literals′), LL″, NewBeam, NB)
15:                    if Cl″ is range restricted then
16:                        if Cl″ has a target predicate in the head then
17:                            TC ←Insert((Cl″, Literals′), LL″, TC, NTC)
18:                        else
19:                            BC ←Insert((Cl″, Literals′), LL″, BC, NBC)
20:                        end if
21:                    end if
22:                end for
23:            end while
24:            Beam ← NewBeam
25:            Steps ← Steps + 1
26:        until Steps > NI
27:    end for
28:    Th ← ∅, ThLL ← −∞                                            ▷ Theory search
29:    repeat
30:        Remove the first couple (Cl, LL) from TC
31:        (LL′, Th′) ←EMBLEM(I, Th ∪ {Cl}, ϵ, δ)
32:        if LL′ > ThLL then
33:            Th ← Th′, ThLL ← LL′
34:        end if
35:    until TC is empty
36:    Th ← Th ⋃(Cl,LL)∈BC {Cl}
37:    (LL, Th) ←EMBLEM(I, Th, D, NEM, ϵ, δ)
38:    return Th
39: end function
```

3 SLIPCOVER for the Competition

In order to fit the competition's requirements we introduced two new features in SLIPCOVER.

The first regards the inclusion of negative literals in bottom clauses and so their use by the refinement operator. When this feature is selected, *modeb* declarations where all the arguments are input are used to generate negative literals for the inclusion in the bottom clauses. The requirement on arguments ensures that negative literals will be instantiated when called so that no floundering occurs.

The second feature allows the user to pose an upper limit on the number of explanations for queries, in order to limit the time spent by SLIPCOVER in testing candidate clauses and theories.

Finally, in order to perform non-observational predicate learning, we have implemented a simple meta-interpreter which performs abductive inference in order to obtain positive and negative examples for target predicates given the observed positive and negative examples.

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. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
4. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic Programs With Annotated Disjunctions. In: Demoen, B., Lifschitz, V. (eds.) Logic Programming: 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004. Proceedings. LNCS, vol. 3132, pp. 431–445. Springer Berlin Heidelberg, Berlin Heidelberg, Germany (2004)