Probabilistic Logic Programming with cplint

Week 2, lecture 2: learning

Fabrizio Riguzzi

Parameter Learning

Definition

Given an LPAD $\mathcal{P}$ with unknown parameters and two sets $E^+ = \{e_1, \ldots, e_T\}$ and $E^- = \{e_{T+1}, \ldots, e_Q\}$ of ground atoms (positive and negative examples), find the value of the parameters $\mathbf{\Pi}$ of $\mathcal{P}$ that maximize the likelihood of the examples, i.e., solve

$$\arg\max_{\mathbf{\Pi}} P(E^+, \sim E^-) = \arg\max_{\mathbf{\Pi}} \prod_{t=1}^{T} P(e_t) \prod_{t=T+1}^{Q} P(\sim e_t).$$

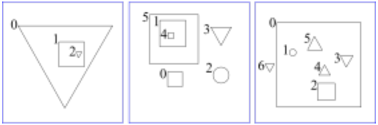
The predicates for the atoms in E^+ and E^- are called *target* because the objective is to be able to better predict the truth value of atoms for them.

Parameter Learning

- Typically, the LPAD $\mathcal{P}$ has two components:
  - a set of rules, annotated with parameters
  - a set of certain ground facts, representing background knowledge on individual cases of a specific world
- Useful to provide information on more than one world: a background knowledge and sets of positive and negative examples for each world
- Description of one world: *mega-interpretation* or *mega-example*
- Positive examples encoded as ground facts of the mega-interpretation and the negative examples as suitably annotated ground facts (such as $neg(a)$ for negative example a)
- The task then is maximizing the product of the likelihood of the examples for all mega-interpretations.

Example: Bongard Problems

- Introduced by the Russian scientist M. Bongard
- Pictures, some positive and some negative
- Problem: discriminate between the two classes.
- The pictures contain shapes with different properties, such as small, large, pointing down, ... and different relationships between them, such as inside, above, ...



Data

Each mega-examle encodes a single picture

```
begin(model(2)).
pos.
triangle(o5).
config(o5,up).
square(o4).
in(o4,o5).
circle(o3).
triangle(o2).
config(o2,up).
in(o2,o3).
triangle(o1).
config(o1,up).
end(model(2)).

begin(model(3)).
neg(pos).
circle(o4).
circle(o3).
in(o3,o4).
....
```

Program

Theory for parameter learning and background

```
pos:0.5 :-
  circle(A),
  in(B,A).
pos:0.5 :-
  circle(A),
  triangle(B).
```

The task is to tune the two parameters

Parameter Learning

- The random variables associated to clauses are unobserved in the data
- Relative frequency cannot be used
- An Expectation-Maximization algorithm must be used:
  - Expectation step: the distribution of the unseen variables in each instance is computed given the observed data
  - Maximization step: new parameters are computed from the distributions using relative frequency
  - End when likelihood does not improve anymore

EMBLEM

- EM over Bdds for probabilistic Logic programs Efficient Mining [Bellodi and Riguzzi IDA 2013]
- Input: an LPAD; logical interpretations (data); *target* predicate(s)
- All ground atoms in the interpretations for the target predicate(s) correspond to as many queries
- BDDs encode the explanations for each query
- Expectations computed with two passes over the BDDs

Input File

Preamble

```
:-use_module(library(slipcover)).
:- if(current_predicate(use_rendering/1)).
:- use_rendering(c3).
:- use_rendering(lpad).
:- endif.
:-sc.
:- set_sc(random_restarts_number,10).
:- set_sc(seed,seed(3020)).
:- set_sc(epsilon_em,0.001).
:- set_sc(epsilon_em_fraction,0.001).
:- set_sc(verbosity,1).
```

See `http://cplint.eu/help/help-cplint.html` for a list of options

Input File

Theory for parameter learning and background

```
bg([]).
in([
(pos:0.5 :-
  circle(A),
  in(B,A)),
(pos:0.5 :-
  circle(A),
  triangle(B))]).
```

Input File

Data: two formats, models

```
begin(model(2)).
pos.
triangle(o5).
config(o5,up).
square(o4).
in(o4,o5).
circle(o3).
triangle(o2).
config(o2,up).
in(o2,o3).
triangle(o1).
config(o1,up).
end(model(2)).

begin(model(3)).
neg(pos).
circle(o4).
circle(o3).
in(o3,o4).
....
```

Input File

Data: two formats, keys (internal representation)

```
pos(2).
triangle(2,o5).
config(2,o5,up).
square(2,o4).
in(2,o4,o5).
circle(2,o3).
triangle(2,o2).
config(2,o2,up).
in(2,o2,o3).
triangle(2,o1).
config(2,o1,up).

neg(pos(3)).
circle(3,o4).
circle(3,o3).
in(3,o3,o4).
square(3,o2).
circle(3,o1).
in(3,o1,o2).
....
```

Input File

- Folds (a group of examples)
- Target predicates `output(<predicate>)`

```
fold(train,[2,3,5,...]).
fold(test,[490,491,494,...]).
output(pos/0).
```

Command

```
induce_par([train],P),
  test(P,[test],LL,AUCROC,ROC,AUCPR,PR).
```

```
http://cplint.eu/e/bongard.pl
```

Structure Learning for LPADs

- Given a set of interpretations (data)
- *Find the model and the parameters* that maximize the probability of the data (log-likelihood)
- SLIPCOVER: Structure LearnIng of Probabilistic logic program by searching OVER the clause space EMBLEM [Riguzzi & Bellodi TPLP 2015]
  1. Beam search in the space of clauses to find the promising ones
  2. Greedy search in the space of probabilistic programs guided by the LL of the data.
- *Parameter learning* by means of EMBLEM

SLIPCOVER

- Cycle on the set of predicates that can appear in the head of clauses, either target or background
- For each predicate, beam search in the space of clauses
- The initial set of beams is generated by building a set of *bottom clauses* as in Progol [Muggleton NGC 1995]
- Bottom clause: most specific clause covering an example

Language Bias

- Mode declarations as in Progol
- Syntax

```
modeh(RecallNumber,PredicateMode).
modeb(RecallNumber,PredicateMode).
```

- `RecallNumber` can be a number or *. Usually *. Maximum number of answers to queries to include in the bottom clause

Mode Declarations

- `PredicateMode` template of the form:

```
p(ModeType, ModeType,...)
```

- Some examples:

```
modeb(1,mem(+number,+list)).
modeb(1,dec(+integer,-integer)).
modeb(1,mult(+integer,+integer,-integer)).
modeb(1,plus(+integer,+integer,-integer)).
modeb(1,(+integer)=(#integer)).
modeb(*,has_car(+train,-car))
```

Mode Declarations

- `ModeType` can be:
  - Simple:
    - `+T` input variables of type `T`;
    - `-T` output variables of type `T`; or
    - `#T`, `-#T` constants of type `T`.
  - Structured: of the form `f(..)` where `f` is a function symbol and every argument can be either simple or structured. For example:

```
modeb(1,mem(+number,[+number|+list])).
```

Bottom Clause $\bot$

- Most specific clause covering an example *e*
- Form: $e \leftarrow B$
- *B*: set of ground literals that are true regarding the example *e*
- *B* obtained by considering the constants in *e* and querying the data for true atoms regarding these constants
- Values for output arguments are used as input arguments for other predicates
- A map from types to lists of constants is kept, it is enlarged with constants in the answers to the queries and the procedure is iterated a user-defined number of times
- `#T` arguments are instantiated in calls, `-#T` aren't and the values after the call are added to the list of constants

Bottom Clause $\perp$

- Example:

$e = father(john, mary)$
$B = \{parent(john, mary), parent(david, steve),$
$parent(kathy, mary), female(kathy), male(john), male(david)\}$
$modeh(father(+person, +person)).$
$modeb(parent(+person, -person)).$
$modeb(parent(-\#person, +person)).$
$modeb(male(+person)).\quad modeb(female(\#person)).$
$e \leftarrow B = father(john, mary) \leftarrow parent(john, mary), male(john),$
$parent(kathy, mary), female(kathy).$

Bottom Clause $\perp$

- The resulting ground clause $\perp$ is then processed by replacing each term in a + or - placemarker with a variable
- An input variable (+T) must appear as an output variable with the same type in a previous literal and a constant (#T or -#T) is not replaced by a variable.

$\perp = \mathit{father}(X, Y) \leftarrow$
$\mathit{parent}(X, Y), \mathit{male}(X), \mathit{parent}(\mathit{kathy}, Y), \mathit{female}(\mathit{kathy}).$

Determination

```
determination(pred1/n1,pred2/n2).
```

- indicates that `pred2/n2` can appear in the body of clauses for predicate `pred1/n1`
- As in Progol

SLIPCOVER

- The initial beam associated with predicate P/Ar of h will contain the clause with the empty body $h : 0.5.$ for each bottom clause $h :- b_1, \ldots, b_m$ In each iteration of the cycle over predicates, it performs a beam search in the space of clauses for the predicate.
- The beam contains couples $(Cl, LIterals)$ where $Literals = \{b_1, \ldots, b_m\}$
- For each clause Cl of the form $Head :- Body$, the refinements are computed by adding a literal from *Literals* to the body.
- Each refinement is evaluated in terms of LL by using EMBLEM
- and added in order of LL to the lists *TC* (target predicates) or *BC* (non-target predicates)

SLIPCOVER

- After the clause search phase, SLIPCOVER performs a greedy search in the space of theories:
  - it starts with an empty theory and adds a target clause at a time from the list *TC*.
  - After each addition, it runs EMBLEM and computes the LL of the data as the score of the resulting theory.
  - If the score is better than the current best, the clause is kept in the theory, otherwise it is discarded.
- Finally, SLIPCOVER adds all the clauses in *BC* to the theory and performs parameter learning on the resulting theory.

Example Input File for Bongard

Preamble

```
:-use_module(library(slipcover)).
:- if(current_predicate(use_rendering/1)).
:- use_rendering(c3).
:- use_rendering(lpad).
:- endif.
:-sc.
:- set_sc(megaex_bottom,20).
:- set_sc(max_iter,3).
:- set_sc(max_iter_structure,10).
:- set_sc(maxdepth_var,4).
:- set_sc(verbosity,1).
```

See `http://cplint.eu/help/help-cplint.html` for a list of options

Input File

Background

```
bg([]).
```

Input File

Data:

```
begin(model(2)).
pos.
triangle(o5).
config(o5,up).
square(o4).
in(o4,o5).
circle(o3).
triangle(o2).
config(o2,up).
in(o2,o3).
triangle(o1).
config(o1,up).
end(model(2)).

begin(model(3)).
neg(pos).
circle(o4).
circle(o3).
in(o3,o4).
....
```

Input File

- Folds
- Target predicates `output(<predicate>)`
- Input predicates are those whose atoms you are not interested in predicting

  ```
  input_cw(<predicate>/<arity>).
  ```

  True atoms are those in the interpretations and those derivable from them using the background knowledge
- Open world input predicates are declared with

  ```
  input(<predicate>/<arity>).
  ```

  the facts in the interpretations, the background clauses and the clauses of the input program are used to derive atoms

Input File

```
fold(train,[2,3,5,...]).
fold(test,[490,491,494,...]).
output(pos/0).
input_cw(triangle/1).
input_cw(square/1).
input_cw(circle/1).
input_cw(in/2).
input_cw(config/2).
```

Input File

Language bias

```
determination(pos/0,triangle/1).
determination(pos/0,square/1).
determination(pos/0,circle/1).
determination(pos/0,in/2).
determination(pos/0,config/2).
modeh(*,pos).
modeb(*,triangle(-obj)).
modeb(*,square(-obj)).
modeb(*,circle(-obj)).
modeb(*,in(+obj,-obj)).
modeb(*,in(-obj,+obj)).
modeb(*,config(+obj,-#dir)).
```

Input File

Search bias

```
lookahead(logp(B),[(B=_C)]).
```

- When trying to add atom `logp(B)`, add instead the conjunction `logp(B),B=_C`

Command

- Structure learning

```
induce([train],P),
  test(P,[test],LL,AUCROC,ROC,AUCPR,PR).
```

```
http://cplint.eu/e/bongard.pl
```