Probabilistic Logic Programming for Natural Language Processing

Fabrizio Riguzzi, Evelina Lamma, Marco Alberti, **Elena Bellodi**, Riccardo Zese, Giuseppe Cota

Dipartimento di Matematica e Informatica
Dipartimento di Ingegneria
Università di Ferrara, Italy

[fabrizio.riguzzi,marco.alberti,elena.bellodi,riccardo.zese, giuseppe.cota,evelina.lamma]@unife.it

URANIA 2016

Outline

1. Probabilistic Logic Programming
2. Natural Language Processing
   - Probabilistic Context-Free Grammars
   - Probabilistic Left Corner Grammars
   - Hidden Markov Models
3. Conclusions and Future Work

Outline

1. Probabilistic Logic Programming
2. Natural Language Processing
   - Probabilistic Context-Free Grammars
   - Probabilistic Left Corner Grammars
   - Hidden Markov Models
3. Conclusions and Future Work

Idea

- Probabilistic Programming (PP) [Pfeiffer, 2016] has recently emerged as a useful tool for building complex probabilistic models and for performing inference and learning on them
- Probabilistic Logic Programming (PLP) is PP based on Logic Programming, that allows to model domains characterized by complex and uncertain relationships among domain entities
- Often a problem description is given in human (natural) language: the set of techniques developed to find automatic ways to understand a text goes under the name of Natural Language Processing (NLP)
- **We applied Probabilistic Logic Programming to NLP** in scenarios such as Probabilistic Context Free Grammars, Probabilistic Left Corner Grammars and Hidden Markov Models
- We used our web application for PLP called *cplint on SWISH*

Probabilistic Logic Programming (PLP) Languages under the Distribution Semantics

- A widespread approach proposed in Logic Programming is the Distribution Semantics [Sato, 1995]
- A probabilistic logic program defines a probability distribution over normal logic programs (called *possible worlds*)
- The distribution is extended to a joint distribution over worlds and interpretations (or queries) and the probability of a query is obtained from this distribution
- These languages differ in the way they define the distribution over logic programs
- Examples:
  - Stochastic Logic Programs [Dantsin, 1991]
  - Probabilistic Horn Abduction, Independent Choice Logic (ICL) [Poole 1993, 1997]
  - PRISM [Sato and Kameya, 1997]
  - **Logic Programs with Annotated Disjunctions** (LPADs)[Vennekens et al., 2004]
  - ProbLog [De Raedt et al., 2007]

Logic Programs with Annotated Disjunctions (LPADs)

- **Example**: encoding of the result of tossing a coin, on the base of the fact that it is biased or not

$$\begin{aligned}
C_1 &= \mathit{heads}(\mathit{Coin}) : 0.5; \mathit{tails}(\mathit{Coin}) : 0.5 \leftarrow \mathit{toss}(\mathit{Coin}), \neg \mathit{biased}(\mathit{Coin}).\\
C_2 &= \mathit{heads}(\mathit{Coin}) : 0.6; \mathit{tails}(\mathit{Coin}) : 0.4 \leftarrow \mathit{toss}(\mathit{Coin}), \mathit{biased}(\mathit{Coin}).\\
C_3 &= \mathit{fair}(\mathit{coin}) : 0.9; \mathit{biased}(\mathit{coin}) : 0.1.\\
C_4 &= \mathit{toss}(\mathit{coin}) : 1.
\end{aligned}$$

C_1: a fair coin lands on heads *or* on tails with probability 0.5
C_2: a biased coin lands on heads with probability 0.6 *or* on tails with 0.4
C_3: a certain coin *coin* has a probability of 0.9 of being fair and of 0.1 of being biased

C_4: *coin* is certainly tossed

- Distributions over the *head* of the formulas
- Worlds built by selecting *only one atom from the head of every grounding* of each rule $\rightarrow$ the LPAD has $2 \cdot 2 \cdot 2 = 8$ possible worlds.

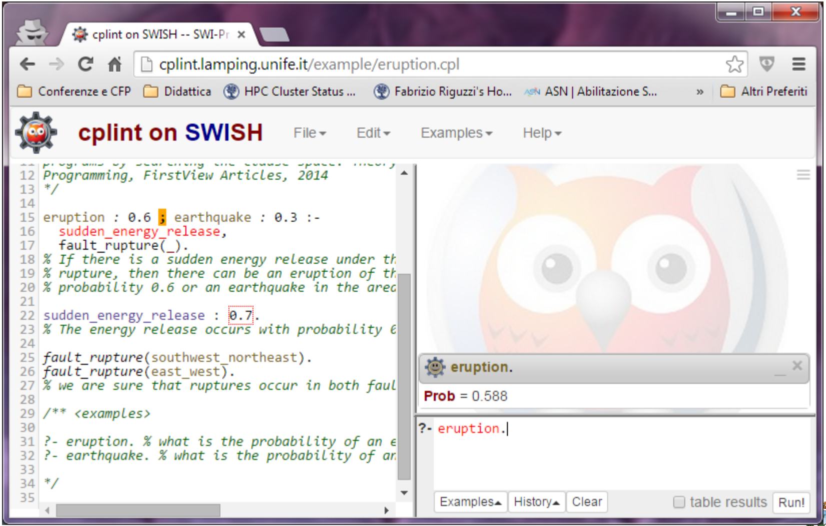
Reasoning Tasks

- Inference: computing the **probability of a query** given the model (the probabilistic logic program) and, possibly, some evidence
- Learning
  - Parameter learning: we know the structural part of the model (the logic formulas) but not the numeric part (*parameters or weights*, i.e. the probabilities) $\rightarrow$ **learning parameters from data**
  - Structure learning $\rightarrow$ we want to learn **both the structure and the parameters** of the model from data

cplint on SWISH

- Web application allowing the user to write Logic Programs with Annotated Disjunctions and performing inference or learning with just a web browser: http://cplint.lamping.unife.it
- cplint is a suite of programs for reasoning on LPADs
- SWISH is a web framework for logic programming based on some packages of SWI-Prolog
    - the Pengine library allows to create remote Prolog engines that evaluate the queries and return answers for them

Inference example in *cplint on SWISH*



Outline

1. Probabilistic Logic Programming

2. Natural Language Processing
   - Probabilistic Context-Free Grammars
   - Probabilistic Left Corner Grammars
   - Hidden Markov Models

3. Conclusions and Future Work

Probabilistic Context-Free Grammars

A Probabilistic Context-Free Grammar (PCFG) consists of:

1. A context-free grammar $G = (N, \Sigma, I, R)$ where
   - N is a finite set of non-terminal symbols,
   - Σ is a finite set of terminal symbols,
   - $I \in N$ is a distinguished start symbol,
   - R is a finite set of rules of the form $X \rightarrow Y_1, \ldots, Y_n$, where $X \in N$ and $Y_i \in (N \cup \Sigma)$
2. A parameter θ for each rule $\alpha \rightarrow \beta \in R$. **Therefore we have probabilistic rules of the form** $\theta : \alpha \rightarrow \beta$

Encoding of a PCFG in PLP

- PCFG = $\{0.2 : S \rightarrow aS, 0.2 : S \rightarrow bS, 0.3 : S \rightarrow a, 0.3 : S \rightarrow b\}$
  $\{S\} = N$, $\{a, b\} = \Sigma$

  ```
  pcfg(L):- pcfg(['S'],[],_Der,L,[]).
  ```

  $\rightarrow$ L is accepted if it can be derived from the start symbol S and an empty string of previous terminals.

  ```
  pcfg([A|R],Der0,Der,L0,L2):- rule(A,Der0,RHS),
                               pcfg(RHS,[rule(A,RHS)|Der0],Der1,L0,L1),
                               pcfg(R,Der1,Der,L1,L2).
  ```

  $\rightarrow$ if there is a rule for A (i.e. it is a non-terminal), expand A using the rule and continue with the rest of the list.

  ```
  pcfg([A|R],Der0,Der,[A|L1],L2):- \+ rule(A,_,_),
                                   pcfg(R,Der0,Der,L1,L2).
  ```

  $\rightarrow$ if A is a terminal, move it to the output string.

  ```
  pcfg([],Der,Der,L,L).
  rule('S',Der,[a,'S']):0.2; rule('S',Der,[b,'S']):0.2;
  rule('S',Der,[a]):0.3; rule('S',Der,[b]):0.3.
  ```

  $\rightarrow$ encodes the rules of the grammar.

Inference on a PCFG in cplint on SWISH

- What is the probability that the string abaa belongs to the language?
- Submit to *cplint on SWISH* (http://cplint.lamping.unife.it/example/inference/pcfg.pl) the query `?-prob(pcfg([a,b,a,a]),Prob).`
- $Prob = 0.0024$

Probabilistic Left Corner Grammars (PLCG)

PLCGs set probabilities not during the expansion of non-terminals but during 3 elementary operations in bottom-up parsing, i.e. shift, attach and project. As a result they define a different class of distributions than PCFGs.

- Given the rules

  ```
  S->SS
  S->a
  S->b
  ```

  where $\{S\} = N$ and $\{a, b\} = \Sigma$
- and the LPAD

  ```
  plc(Ws) :- g_call(['S'],Ws,[],[],_Der).
  g_call([],L,L,Der,Der).
  g_call([G|R], [G|L],L2,Der0,Der) :- % shift
    terminal(G),
    g_call(R,L,L2,Der0,Der).
  g_call([G|R], [Wd|L] ,L2,Der0,Der) :-
    \+ terminal(G), first(G,Der0,Wd),
    lc_call(G,Wd,L,L1,[first(G,Wd)|Der0],Der1),
    g_call(R,L1,L2,Der1,Der).
  ```

Probabilistic Left Corner Grammars (PLCG)

```
lc_call(G,B,L,L1,Der0,Der) :- % attach
  lc(G,B,Der0,rule(G, [B|RHS2])),
  attach_or_project(G,Der0,attach),
  g_call(RHS2,L,L1,[lc(G,B,rule(G, [B|RHS2])),attach|Der0],Der).
lc_call(G,B,L,L2,Der0,Der) :- % project
  lc(G,B,Der0,rule(A, [B|RHS2])),
  attach_or_project(G,Der0,project),
  g_call(RHS2,L,L1,[lc(G,B,rule(A, [B|RHS2])),project|Der0],Der1),
  lc_call(G,A,L1,L2,Der1,Der).
lc_call(G,B,L,L2,Der0,Der) :- \+ lc(G,B,Der0,rule(G,[B|_])),
                             lc(G,B,Der0,rule(A, [B|RHS2])),
    g_call(RHS2,L,L1,[lc(G,B,rule(A, [B|RHS2]))|Der0],Der1),
                                lc_call(G,A,L1,L2,Der1,Der).
attach_or_project(A,Der,Op) :- lc(A,A,Der,_), attach(A,Der,Op).
attach_or_project(A,Der,attach) :- \+ lc(A,A,Der,_).
lc('S','S',_Der,rule('S',['S','S'])).
lc('S',a,_Der,rule('S',[a])).
lc('S',b,_Der,rule('S',[b])).
first('S',Der,a):0.5; first('S',Der,b):0.5.
attach('S',Der,attach):0.5; attach('S',Der,project):0.5.
terminal(a). terminal(b).
```

- the probability (with approximate inference by Monte Carlo sampling) that the string ab is generated by the grammar can be computed with the query `?-mc_prob(plc([a,b]),P).` in *cplint on SWISH*
- $P \sim 0.031$

Hidden Markov Models (HMM)

- Hidden Markov Models for part-of-speech tagging: words can be considered as output symbols and a sentence the sequence of output symbols emitted by an HMM
- **States** represent **parts of speech** and the **symbols** emitted by the states are **words**
- The assumption is that a word depends probabilistically on just its own part of speech (i.e. its tag) which in turn depends on the part of speech of the preceding word (or on the start state in case there is no preceding word)
- Two kinds of probabilities:
  - *transition* probabilities: from one state to another
  - *output* probabilities: 1 in our program (for every state there is only one possible output)

Encoding of HMM in PLP

```
hmm(O):-hmm(_,O).
```

→ O is an output sequence if there is a state sequence S such that hmm(S,O) holds.

```
hmm(S,O):- trans(start,Q0,[]),hmm(Q0,[],S0,O),reverse(S0,S).
```

→ O is an output sequence and S a state sequence if the chain starts at state start and ends generating state sequence S and output sequence O.

```
hmm(Q,S0,S,[L|O]):-trans(Q,Q1,S0),out(L,Q,S0),hmm(Q1,[Q|S0],S,O).
```

→ an HMM in state Q goes in state Q1, emits the word L and continues the chain.

```
hmm(_,S,S,[]).
```

→ an HMM in any state terminates the sequence without emitting any symbol.

```
trans(start,det,_):0.30; trans(start,aux,_):0.20; trans(start,v,_):0.10;
  trans(start,n,_):0.10; trans(start,pron,_):0.30.
trans(det,det,_):0.20; trans(det,aux,_):0.01; trans(det,v,_):0.01;
  trans(det,n,_):0.77; trans(det,pron,_):0.01.
trans(aux,det,_):0.18; trans(aux,aux,_):0.10; trans(aux,v,_):0.50;
  trans(aux,n,_):0.01; trans(aux,pron,_):0.21.
```

```
out(a,det,_). out(can,aux,_). out(can,v,_).
out(can,n,_). out(he,pron,_).
```

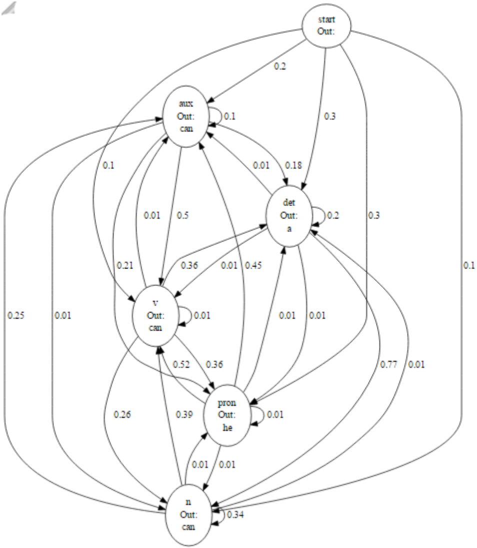
Inference on a HMM in cplint on SWISH

- Which is the most frequent state sequence for the sentence he can can a can?
- It corresponds to the most frequent part-of-speech tagging for that sentence, that should be [pron, aux, v, det, n]
- Submit to *cplint on SWISH* (http://cplint.lamping.unife.it/example/inference/hmmpos.pl) the query

  `mc_sample_arg(hmm(S,[he,can,can,a,can]),100,S,O).`

Inference on a HMM in cplint on SWISH

G =



Conclusions and Future Work

- Conclusions
  - PCFGs, PLCGs and HMMs are some of the most widely used models in NLP. In this paper we show that is possible to represent these models with Probabilistic Logic Programs
- Future Work
  - We are currently considering a version of probabilistic Definite Clause Grammars, where the probability distribution is defined on the possible non-terminals with the same expansion, rather than on the possible expansions of a non-terminal. This extension could be mapped naturally on LPADs, and could be applied to probabilistic parsing of ambiguous grammars