Probabilistic Logic Programming for Natural Language Processing

Fabrizio Riguzzi[1], Evelina Lamma[2], Marco Alberti[1], Elena Bellodi[2], Riccardo Zese[2], and Giuseppe Cota[2]

[1] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
{fabrizio.riguzzi,evelina.lamma,marco.alberti,
elena.bellodi,riccardo.zese,giuseppe.cota}@unife.it

Abstract. The ambition of Artificial Intelligence is to solve problems without human intervention. Often the problem description is given in human (natural) language. Therefore it is crucial to find an automatic way to understand a text written by a human. The research field concerned with the interactions between computers and natural languages is known under the name of Natural Language Processing (NLP), one of the most studied fields of Artificial Intelligence.
In this paper we show that Probabilistic Logic Programming (PLP) is a suitable approach for NLP in various scenarios. For this purpose we use `cplint` on SWISH, a web application for Probabilistic Logic Programming. `cplint` on SWISH allows users to perform inference and learning with the framework `cplint` using just a web browser, with the computation performed on the server.

Keywords: Probabilistic Logic Programming, Probabilistic Logical Inference, Natural Language Processing

1 Introduction

The ambition of Artificial Intelligence is to solve problems without human intervention. Often the problem description is given in human (natural) language. Therefore Natural Language Processing (NLP) is fundamental for problem solving.

In this paper we show that is possible to represent NLP models such as Probabilistic Context Free Grammars, Probabilistic Left Corner Grammars and Hidden Markov Models with Probabilistic Logic Programs.

Probabilistic Programming (PP) [5] has recently emerged as a useful tool for building complex probabilistic models and for performing inference and learning on them.

Probabilistic Logic Programming (PLP) [1] is PP based on Logic Programming that allows to model domains characterized by complex and uncertain relationships among domain entities.

Many systems have been proposed for reasoning with PLP. Even if they are freely available for download, using them usually requires a complex installation process and a steep learning curve. In order to mitigate these problems, we developed `cplint` on SWISH [8], a web application for reasoning on PLP with just a web browser: the algorithms run on a server and the users can post queries and see the results in their browser. The application is available at `http://cplint.lamping.unife.it`.

`cplint` on SWISH uses the reasoning algorithms included in the `cplint` suite, including exact and approximate inference and parameter and structure learning.

The paper is organized as follows. Section 2 provides an overview of the distribution semantics, Section 3 illustrates some examples of how to represent some of the most famous NLP models in PLPs. Finally, section 5 concludes the paper.

All the examples in the paper named as `<name>.pl` can be accessed online at `http://cplint.lamping.unife.it/example/inference/<name>.pl`.

2 Syntax and Semantics

The distribution semantics [11] is one of the most successful approaches for representing probabilistic information in Logic Programming and it is at the basis of many languages, such as Independent Choice Logic, PRISM, Logic Programs with Annotated Disjunctions (LPADs) and ProbLog.

We consider first the discrete version of probabilistic logic programming languages. In this version, each atom is associated with a Boolean random variable that can assume values true or false representing the its presence or absence in the program. The facts and rules of the program specify the dependences among the truth value of atoms and the main inference task is to compute the probability that a ground query is true, often conditioned on the truth of another ground goal, the evidence. All the languages following the distribution semantics allow the specification of alternatives either for facts and/or for clauses. We present here the syntax of LPADs because it is the most general [18].

An LPAD is a finite set of annotated disjunctive clauses of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i}$:- $b_{i1}, \ldots, b_{im_i}$. where $b_{i1}, \ldots, b_{im_i}$ are literals, $h_{i1}, \ldots h_{in_i}$ are atoms and $\Pi_{i1}, \ldots, \Pi_{in_i}$ are real numbers in the interval $[0, 1]$. This clause can be interpreted as "if $b_{i1}, \ldots, b_{im_i}$ is true, then h_{i1} is true with probability Π_{i1} or $\ldots$ or h_{in_i} is true with probability Π_{in_i}."

Given an LPAD P, the grounding $ground(P)$ is obtained by replacing variables with terms from the Herbrand universe in all possible ways. If P does not contain function symbols and P is finite, $ground(P)$ is finite as well.

$ground(P)$ is still an LPAD from which we can obtain a normal logic program by selecting a head atom for each ground clause. This normal program, called

"world", is assigned a probability by multiplying the probabilities of all the head atoms chosen. In this way we get a probability distribution over worlds from which we can define a probability distribution over the truth values of a ground atom: the probability of an atom q being true is the sum of the probabilities of the worlds where q is true, that can be checked because the worlds are normal programs that we assume have a two-valued well-founded model.

This semantics can be given also a sampling interpretation: the probability of a query q is the fraction of worlds, sampled from the distribution over worlds, where q is true. To sample from the distribution over worlds, you simply randomly select a head atom for each clause according to the probabilistic annotations. Note that you don't even need to sample a complete world: if the samples you have taken ensure the truth value of q is determined, you don't need to sample more clauses.

To compute the conditional probability $P(q|e)$ of a query q given evidence e, you can use the definition of conditional probability, $P(q|e) = P(q, e)/P(e)$, and compute first the probability of q, e (the sum of probabilities of worlds where both q and e are true) and the probability of e and then divide the two.

If the program P contains function symbols, a more complex definition of the semantics is necessary, because $ground(P)$ is infinite, a world would be obtained by making an infinite number of choices and so its probability, the product of infinite numbers all smaller than one, would be 0. In this case you have to work with sets of worlds and use Kolmogorov's definition of probability space [7].

3 NLP Models

In NLP two common tasks are to check whether a sentence respects a grammar or to tag each word of a sentence with a part-of-speech (POS) tag. For NLP the grammars that are used in the theory of formal languages such as context free grammars or left corner grammars do not work well because the rules are too strict. Natural language is more flexible and is characterized by many exceptions to rules. To model natural language, probabilistic versions of the grammars above have been developed, such as Probabilistic Context Free Grammars or Probabilistic Left Corner Grammars. Similarly, for POS tagging, statistical tools such as Hidden Markov Models give good results.

3.1 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 $X \to Y_1, \ldots, Y_n$, where $X \in N$ and $Y_i \in (N \cup \Sigma)$.
2. A parameter θ for each rule $\alpha \to \beta \in R$. Therefore we have probabilistic rules of the form $\theta : \alpha \to \beta$

This kind of models can be represented by PLPs. For instance consider the program `pcfg.pl` (adapted from [15]) shown below. It represents the PCFG $\{0.2 : S \to aS, 0.2 : S \to bS, 0.3 : S \to a, 0.3 : S \to b\}$, where $\{S\}$ is N and $\{a, b\}$ is Σ.

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

In this example if we want to perform exact inference and we want to ask for the probability of the string "abaa", we have to submit to `cplint` on SWISH the query `prob(pcfg([a,b,a,a]),Prob).` and we obtain 0.0024 (In this case the string is not ambiguous so there exists only one derivation with probability $0.2 \cdot 0.2 \cdot 0.2 \cdot 0.3 = 0.0024$).

3.2 Probabilistic Left Corner Grammars

A Probabilistic Left Corner Grammar (PLCG) is a probabilistic version of left-corner grammar which uses the same set of rules as a PCFG. Whereas PCFGs assume top-down parsing, PLCGs are based on bottom-up parsing. PLCGs set probabilities not to expansion of non-terminals but to three elementary operations in bottom-up parsing, i.e. shift, attach and project. As a result they define a different class of distributions from PCFGs.

Programs for PLCGs look very different from those for PCFGs. The program `plcg.pl` (adapted from [14]) represent a PLCG that has as CFG rules $\{S \to SS, S \to a, S \to b\}$, where $\{S\}$ is N and $\{a, b\}$ is Σ

```
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).
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).
```

Besides exact inference, cplint offers also approximate inference by Monte Carlo sampling. If we want to know the probability that the string 'ab' is generated by the grammar, we have to write the query `mc_prob(plc([a,b]),P).` and `cplint` on SWISH will return ~ 0.031.

3.3 Hidden Markov Models

In a Hidden Markov Model (HMM) there are states, transitions between states, and symbols emitted by the states. Usually there are two kinds of probabilities in a HMM: transition probabilities, i.e. the probability of a transition from one state to another, and emission or output probabilities, i.e. the probability of a certain state emitting a certain symbol.

HMM can be used for POS tagging words can be considered as output symbols and a sentence the sequence of output symbols emitted by an HMM. In this case, the states are POS tags and the sequence of states that most probably gave the sentence as the sequence of output symbols can be considered as the POS tagging of the sentence.

In the program `hmmpos.pl` (adapted from `http://www.ling.gu.se/~lager/Spaghetti/spaghetti.html`, [4] and [13]) illustrated below, we have a simple HMM representation where the output probabilities are set to 1 (for every state there is only one possible output). In this model the states represent parts-of-speech, and the symbols emitted by the states are words (every POS emits only one word). The assumption is that a POS of a word depends only on the POS of the preceding word (or on the start state in case there is no preceding word).

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

```
hmm(S,O):-
  trans(start,Q0,[]),hmm(Q0,[],S0,O),reverse(S0,S).
hmm(Q,S0,S,[L|O]):-
  trans(Q,Q1,S0),
  out(L,Q,S0),
  hmm(Q1,[Q|S0],S,O).
hmm(_,S,S,[]).
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.
trans(v,det,_):0.36; trans(v,aux,_):0.01; trans(v,v,_):0.01;
  trans(v,n,_):0.26; trans(v,pron,_):0.36.
trans(n,det,_):0.01; trans(n,aux,_):0.25; trans(n,v,_):0.39;
  trans(n,n,_):0.34; trans(n,pron,_):0.01.
trans(pron,det,_):0.01; trans(pron,aux,_):0.45; trans(pron,v,_):0.52;
  trans(pron,n,_):0.01; trans(pron,pron,_):0.01.
out(a,det,_). out(can,aux,_). out(can,v,_). out(can,n,_). out(he,pron,_).
```

For instance we want to know the most frequent state sequence for the sentence "he can can a can". In this case it corresponds to the most frequent part-of-speech sequence for that sentence. By using the `cplint` syntax we query `mc_sample_arg( hmm(S,[he,can,can,a,can]),100,S,O).`, we should obtain the sequence [pron, aux, v, det, n].

4 Related Work

PRISM [12] is one of the most influential systems for the application of PLP to NLP. PRISM uses a symbolic-statistical modeling language that follows the distribution semantics that differs from ICL/LPADs/ProbLog because the programs are required to satisfy some constraints, namely that clauses have mutually exclusive bodies and literals in the body of clauses are mutually independent. This restriction is satisfied by all the models presented in this paper and in fact all of them are inspired by models for the PRISM system. However, the restriction severely limits the expressiveness and may prove too constraining for more complex models.

In [6] the authors build a Bayesian network corresponding to the distribution of the parse trees induced by a given PCFG. The Bayesian network can be used to handle partial sequences, not handled by standard approaches. Contextual information regarding the sentences and other evidence can also be exploited by defining the probability values of the conditional probability tables as functions of the position of the subsequence within the whole terminal string, the length of the subsequence and the position of the term in the sequence.

Other approaches for parameter learning exploit counting for PLCGs starting from a parse tree [2] or define a parser which exploits probabilistic best-first

parsing methods and use beam-search [10] to ensure termination of the learning process. An EM algorithm is exploited in [16,17] to learn the parameters from unbracketed sentences.

Applying (probabilistic) inductive logic programming (ILP) to NLP was investigated in [3] where the authors present an ILP system able to learn syntactic and semantic parsers that are then used to answer natural language database queries by mapping them into Prolog. However the system cannot handle uncertain information.

5 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. All the proposed examples are available in `cplint` on SWISH, a web application for PLP.

A complete online tutorial of `cplint` on SWISH [9] is available at `http://ds.ing.unife.it/~gcota/plptutorial/`.

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.

Acknowledgement This work was supported by the "GNCS-INdAM".

References

1. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Mach. Learn. 100(1), 5–47 (2015)
2. Manning, C.D., Carpenter, B.: Probabilistic parsing using left corner language models. CoRR cmp-lg/9711003 (1997)
3. Mooney, R.J.: Inductive logic programming for natural language processing. In: Muggleton, S. (ed.) ILP-96. LNCS, vol. 1314, pp. 3–22. Springer (1996)
4. Nivre, J.: Logic Programming Tools for Probabilistic Part-of-Speech Tagging. Master thesis, School of Mathematics and Systems Engineering, Växjö University (October 2000), `http://stp.lingfil.uu.se/~nivre/docs/thesis3.pdf`
5. Pfeffer, A.: Practical Probabilistic Programming. Manning Publications (2016)
6. Pynadath, D.V., Wellman, M.P.: Generalized queries on probabilistic context-free grammars. IEEE Trans. Pattern Anal. Mach. Intell. 20(1), 65–77 (1998)
7. Riguzzi, F.: The distribution semantics for normal programs with function symbols. Int. J. Approx. Reason. 77, 1 – 19 (2016), `http://dx.doi.org/10.1016/j.ijar.2016.05.005`
8. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Probabilistic logic programming on the web. Software Pract. and Exper. 46(10), 1381–1396 (October 2016), `http://dx.doi.org/10.1002/spe.2386`
9. Riguzzi, F., Cota, G.: Probabilistic logic programming tutorial. The Association for Logic Programming Newsletter 29(1), 1–1 (March/April 2016), `http://www.cs.nmsu.edu/ALP/2016/03/probabilistic-logic-programming-tutorial/`

10. Roark, B., Johnson, M.: Efficient probabilistic top-down and left-corner parsing. CoRR cs.CL/0008017 (2000)
11. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP-95. pp. 715–729. MIT Press, Cambridge, Massachusetts (1995)
12. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: IJCAI 97. vol. 97, pp. 1330–1339 (1997)
13. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. J. Artif. Intell. Res. 15, 391–454 (2001)
14. Sato, T., Kameya, Y., Kurihara, K.: Variational bayes via propositionalized probability computation in prism. Ann. Math. Artif. Intell. 54(1-3), 135–158 (2008)
15. Sato, T., Kubota, K.: Viterbi training in prism. Theor. Pract. Log. Prog. 15(02), 147–168 (2015)
16. Uytsel, D.H.V., Compernolle, D.V.: Language modeling with probabilistic left corner parsing. Computer Speech & Language 19(2), 171–204 (2005)
17. Van Uytsel, D.H., Van Compernolle, D., Wambacq, P.: Maximum-likelihood training of the plcg-based language model. In: ASRU-2001. pp. 210–213. IEEE (2001)
18. 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), http://dx.doi.org/10.1007/978-3-540-27775-0_30