Worksheet #2
April 23rd, 2020

Paper: Inference and learning in probabilistic logic programs using weighted boolean formulas

General questions

1. What is this paper about? Could you summarise its contribution in a paragraph?

   This paper addresses the problem of inference and learning parameters of probabilistic logic programs. It presents two contributions: (1) a suite of algorithms that can perform inference efficiently by converting the program, queries and evidence to a weighted boolean formula. The main advantage of this conversion is to reduce the problem of inference to well-studied methods, such as weighted model counting; (2) an algorithm, based on EM, for parameter estimation in the LFI setting, built on top of the inference algorithms.

2. How does this work differ from others mentioned in the paper?

   This is described in the end of Section 1: "The present paper is based on and integrates our previous papers (Fierenset al.2011; Gutmannet al.2011) in which inference and learning were studied andimplemented separately. Historically, the LFI approach, as detailed by Gutmannet al.(2010, 2011), was developed first and used Binary Decision Diagrams (BDDs). The use of BDDs for learning in an EM style is related to the approach by Ishihata et al.(2008), who developed an EM algorithm for propositional BDDs and suggested that their approach can be used to perform learning from entailment for PRISM programs. Fierens et al.(2011) later showed that an alternative approach to inference that is more general, efficient and principled can be realized using WMC and compilation of d-DNNFs rather than BDDs, as in the initial ProbLog implementation (Kimmig et al.2010). The present paper employs the approach by Fierens et al., also for learning from interpretations in an EM style and thus integrates the two earlier approaches. The resulting techniques are integrated in a novel implementation, called ProbLog2. While the first ProbLog implementation (Kimmiget al. 2010) was tightly integrated in the YAP Prolog engine and employed BDDs, ProbLog2 is much closer in spirit to some ASP systems than to Prolog and employs d-DNNFs and WMC."

3. Do the authors present experiments? What is the methodology used? Does it sound correct? Why?

   Yes, they do, in Section 9. Results seem sound. In order to prove that they are sound, we would have to be able to reproduce the experiments using the same methodology described here and with the same datasets and implementations.

4. What are the main results/findings/conclusions? Are the results useful/relevant? Why?

   This can be found in the abstract: "The results show that the inference algorithms improve upon the state of the art in probabilistic logic programming, and that it is indeed possible to learn the parameters of a probabilistic logic program from interpretations" and also in the conclusions, where a summary of the contributions is given. Note that they start the paper talking about two contribs and conclude talking about three...

Technical questions

1. What is the difference between "learning from interpretations" and "learning from entailment"? (It may be useful to consult this other paper

   The difference is subtle. Examples are given in page 190 of the paper mentioned above. In LFE, hypotheses are clausal theories, examples are clauses, and a hypothesis covers an example if the hypothesis logically entails the example. In LFI, hypotheses are clausal theories (as in LFE), but examples are Herbrand interpretations and an example is covered when it is a model for the hypothesis. In other words, proof goes from the examples to the hypothesis, contrary to what happens with LFE, where we start from the hypothesis to test the examples.

2. Apply steps (1) to (3) shown in pages 367 through 368 to the basic ProbLog example below. What is the resulting formula? Also apply

inference (page 375) to the given query using WMC (World Model Counting).

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :- influences(Y,X), smokes(Y).

query(smokes(carl)).
```

- Step 1: Relevant Ground Program (RGP)
  In the grounded program below, all grounds that are commented are not generated because SLD-resolution fails for those branches of the derivation tree. The last commented grounding is not relevant for the query.

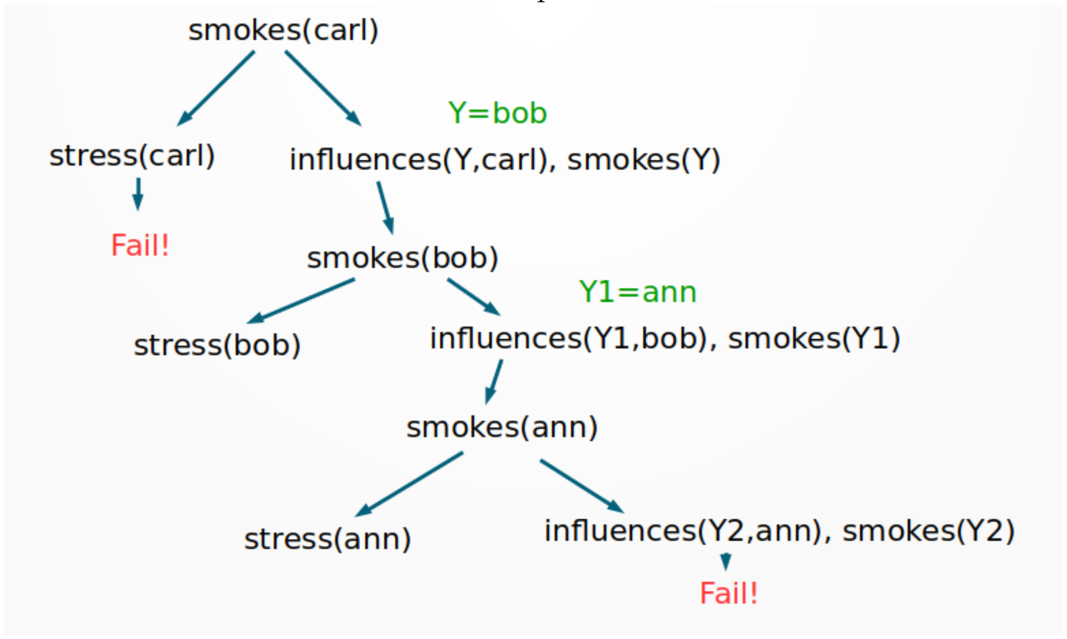
  ```
  % probabilistic facts are already ground...
  0.8::stress(ann).
  0.4::stress(bob).
  0.6::influences(ann,bob).
  0.2::influences(bob,carl).

  % grounding rules
  % smokes(carl) :- stress(carl).
  smokes(bob) :- stress(bob).
  smokes(ann) :- stress(ann).

  % smokes(carl) :- influences(ann,carl), smokes(ann).
  smokes(carl) :- influences(bob,carl), smokes(bob).
  % smokes(ann) :- influences(carl,ann), smokes(carl).
  % smokes(ann) :- influences(bob,ann), smokes(bob).
  % smokes(bob) :- influences(carl,bob), smokes(carl).
  % smokes(bob) :- influences(ann,bob), smokes(ann).

  query(smokes(carl)).
  ```

hbtp



smokes(carl)
stress(carl)
Fail!
Y=bob
influences(Y,carl), smokes(Y)
smokes(bob)
stress(bob)
Y1=ann
influences(Y1,bob), smokes(Y1)
smokes(ann)
stress(ann)
influences(Y2,ann), smokes(Y2)
Fail!

The grounding can be depicted as the SLD-tree shown in Figure 2.

- Step 2: converting to propositional formula
  $smokes(bob) \leftrightarrow stress(bob) \wedge smokes(ann) \leftrightarrow stress(ann) \wedge smokes(carl) \leftrightarrow influences(bob, carl) \wedge smokes(bob)$
- Step 3: weighted boolean formula
  $smokes(bob) \leftrightarrow stress(bob)(weight : 0.4) \wedge smokes(ann) \leftrightarrow stress(ann)(weight : 0.8) \wedge smokes(carl) \leftrightarrow influences(bob, carl)(weight : 0.2) \wedge smokes(bob)(weight : 0.4)$

3. Why the conversion of cyclic rules to a boolean formula is more complicated?

   Because the conversion algorithm needs to handle the cycles, which is a non-trivial problem.

4. Can you use one of the algorithms mentioned in page 372 to the program below (mentioned in page 366)? Also apply inference (page 375) to the given query using WMC (World Model Counting).

```
0.6::edge(1,2).
0.1::edge(1,3).
0.4::edge(2,5).
0.3::edge(2,6).
0.3::edge(3,4).
0.8::edge(4,5).
0.2::edge(5,6).

path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

query(path(1,6)).
```

The mechanism employed here to avoid cycles is tabling. Otherwise, the method is the same as used in the previous question to generate the Relevant Ground Program (RGP). In tabling, the algorithm keeps track of call patterns storing them in an efficient data structure. Whenever a pattern already stored is invoked again to build the RGP, the computation for that stops. Eventually, that path will have a branch with a solution or a pending solution (loop). If the pattern was never seen before, it is stored.

5. Using the example found here, explain the steps to learn the parameters of this program, according to Section 7 (page 381).

   Same as the one given in the previous question, but taking care of the cycles when generating the RGP. The probability of a path that leads to a cycle tends to zero.