Sampling and probabilistic inference in D/Slps.

Nicos Angelopoulos

The Pirbright Institute
head of computational biology

https://www.pirbright.ac.uk/users/dr-nicos-angelopoulos
nicos.angelopoulos@pirbright.ac.uk

23.07.09

overview

- Introduction to Stochastic LP (SLP)
  and Distributional LP (DLP)
- Syntax
- Probabilistic inference and Sampling
- Relative expressivity

Background

SLPs - fixed arithmetic labels on clauses that act as probability. Failure adjusted maximisation (FAM) algorithm learns the labels from data.

DLPs - extension of SLPs, labels on clauses are computed at run time and can depend on ground arguments of goal. Have been used go define priors to affect Bayesian machine learning of model structures.

Implementations

Pepl Parameter Estimation in Prolog.

Bims Bayesian inference of model structure

Both implementations use similar techniques but different data structures and specifics.

Native Prolog implementations

The extended syntax is transformed to standard Prolog with extra arguments and run time code is injected in memory to guide SLD as to affect probabilistic reasoning.

Extra arguments include the labels and returning information such as the path and the status of success of a path.

Installation

SWI-Prolog

$?- \mathit{pack\_install}(\mathit{pepl}).$
$?- \mathit{pack\_install}(\mathit{bims}).$

`https://swi-prolog.org/packs/list/{pepl,bims}`

`https://stoics.org.uk/~nicos/sware/packs/pepl/pepl-2.3.tgz`

`https://github.com/nicos.angelopoulos/{pepl,bims}`

Current versions published recently and include the features described in this paper:

*Pepl v*2.3

*Bims v*3

Syntax - common part

$$
\begin{aligned}
0.5 :: \quad & \mathit{coin}(\mathit{head}). \\
0.5 :: \quad & \mathit{coin}(\mathit{tail}).
\end{aligned}
$$

$$
\begin{aligned}
1 :: \quad \mathit{doubles}(\mathit{Side}) \quad :- & \\
& \mathit{coin}(\mathit{Side}), \\
& \mathit{coin}(\mathit{Side}).
\end{aligned}
$$

Programs for *coin/1* and *doubles/1*.

Syntax for both *Slp* and *Dlp* are identical for these examples.

Syntax - *Dlp* only

Selecting an element uniformly from a list

```
:- pvars(umember(L,_E), [Len-length(L,Len)]).

1/X :: X :: umember( [H|_T], H ).
(1 - 1/X) :: X :: umember( [_H|T], El ) :-
            [X - 1] :: umember( T, El ).
```

pvars/2, allows the definition of probabilistic guards, that will connect the length of input list at run time defines label X.

Note that the program above is economical in that the recursive step passes the $X - 1$ as the length of the list so it doesn't have to be re-calculated.

Probabilistic inference and Sampling

Standard Prolog uses SLD (selective linear definite clause) resolution to refute queries (goals) against a logic program containing clauses.

With probabilistic clauses we can

1. each refutation has an associated probability value assigned to it, which is simply the product of probability labels of the clauses used in the refutation. Furthermore, any specific instantiation will have a total probability ascribed to it, which the sum of the products for all the refutation that derive it. (Probabilistic Inference)
2. Selective Stochastic Definite clause resolution (SSD): At each resolution step, choose which one to select from all matching clauses, in proportion to the relative value of the probabilistic labels. (Stochastic Sampling).

pepl - probabilistic inference

$$0.5 :: \quad coin(head).$$
$$0.5 :: \quad coin(tail).$$

$?-\ sload\_pe(coin).$
$?-\ scall(coin(Flip),Prb).$
$Flip \ = head,$
$Prb \ = 0.5\ ;$
$Flip \ = tail,$
$Prb \ = 0.5.$

A probabilistically "well" behaved program/query combination as sum of all derivation of *coin(Flip)* are equal to 1 (ie no loss of probability mass).

pepl - probabilistic inference - *scall/5*

$$scall(+Goal,\ +Eps, -Path, -Succ, -Prb).$$

Goal input query goal
Eps ϵ cut-off
Path indices of clauses used in path
Succ unbound if success, or fail otherwise.
Prb probability of path

pepl - probabilistic inference - sum over refutations

$$
\begin{aligned}
1 :: \quad & doubles(Side) \quad :- \\
& \quad coin(Side), \\
& \quad coin(Side).
\end{aligned}
$$

$?-\ scall\_sum(doubles(head),\ \ Prb).$

$Prb\ = 0.25.$

$?-\ scall\_sum(doubles(tail),\ \ Prb).$

$Prb\ = 0.25.$

$?-\ scall\_sum(doubles(Side),\ \ Prb).$

$Prb\ = 0.5.$

This query succeeds iff two consequtive *coin/1* tosses turn out the same result.

Unlike the previous example, here we have a probability mass loss of 0.5.

bims- probabilistic inference

```
?- dlp_call(doubles(Side)).
Side  = head ;
Side  = tail  ;
false.

?- dlp_call(doubles(Side),Path,Prb).
Side  = head,
Path  = [3:1,  1/0.5,  1:0.5],   Prb  = 0.25 ;
Side  = tail,
Path  = [3:1,  2/0.5,  2:0.5],   Prb  = 0.25 ;
false.

?- dlp_call_sum(doubles(Side),Sum).
Sum  = 0.5.
?- dlp_call_sum(doubles(head),Sum).
Sum  = 0.25.
```

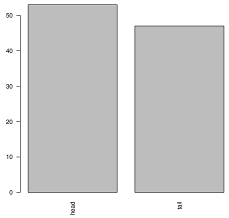
pepl- sampling

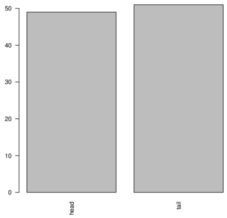
```
?- seed_pe. %                  sets  the  random seed
?- sample(coin(tail),0,Path,Succ,Prb).
Path  =[1],
Succ  =fail,
Prb  =0.5.

?- seed_pe.
?- sample(coin(head),0,Path,Succ,Prb).
Path  =[1],
Prb  =0.5

?- seed_pe.
?- sample(coin(Flip),0,Path,Succ,Prb).
Flip   =head,
Path  =[1],
Prb  =0.5.
```

bims- sampling

```
?- dlp_seed.

?- dlp_sample(doubles(Side)).
Side  = tail.

?- dlp_sample(doubles(Side)).
Side  = tail.

?- dlp_sample(doubles(Side)).
Side  = head.
```

sampling on variant length lists





Left, sampling 100 coin tosses with *Pepl* over *Slps*.

Right, sampling 100 coin tosses with *Bims*.

helper packs

```
?- lib(mlu),  lib(real),    lib(b_real)
?- seed_pe.
?- mlu_sample(sample(coin(Side)), 100, Side,  Freqs),
   Opts  =[interface(barplot),outputs([svg]),las=2],
   mlu_frequency_plot(Freqs, Opts).
Freqs  =[head-53, tail-47].
```

Real Real a c-language interface to *R*

b_real bio predicates for Real

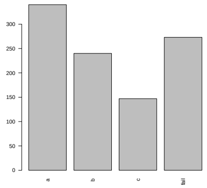
mlu machine learning utilities

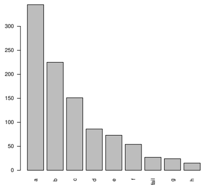
Expressivity - SLP - member selection

$$
\begin{array}{ll}
1/3 :: & \mathit{member3}(\ H, [H|T]\ ). \\
2/3 :: & \mathit{member3}(\ \mathit{Elem}, [\_H|T]\ ) :- \\
& \quad \mathit{member3}(\ \mathit{Elem}, T\ ).
\end{array}
$$

Slp program selecting uniformly from a 3 member list, but is unable to work on arbitrary length inputs.

Slp- sampling *member3/2*





Left:

```
?- Opts = [interface(barplot),outputs(pdf),las=2],
   mlu_sample(sample(member3(X,[a,b,c])), 1000, X, Freqs),
   mlu_frequency_plot(Freqs, Opts).
```

```
Freqs = [a-340, b-240, c-147, fail-273].
```

Right:

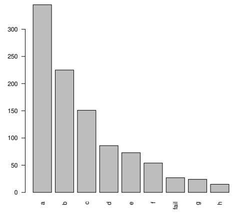
```
mlu_sample(sample(member3(X,[a,b,c,d,e,f,g,h])),1000,X,Freqs)
```

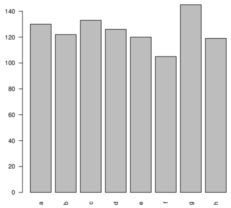
Dlp- sampling umember/2

```
:- pvars(umember(L,_E), [Len-length(L,Len)]).

1/X :: X :: umember( [H|_T], H ).
(1 - 1/X) :: X :: umember( [_H|T], El ) :-
        [X - 1] :: umember( T, El ).
```

```
?- List=[a,b,c,d,e,f,g,h],
   mlu_sample(dlp_sample(umember(List,X)),1000,X,Freqs),
   Opts =[interface(barplot),outputs(pdf)),las=2],
   mlu_frequency_plot(Freqs, Opts ).

Freqs =[a-130,b-122,c-133,d-126,e-120,f-105,g-145,h-119].
```

expressivity - plots comparison





Left, sampling 1000 draws for a single member from an eight member list using *Slp* predicate *member3/2*.

Right, sampling 1000 draws for a single member from the same list with *Dlp* predicate *umember/2*.

bottom line

We have enhanced two PLP packs with facilities to do high level sampling and probabilistic inference.

The ability to perform such tasks are both beneficial both to the libraries but also enable researchers in probabilistic logic programming to experiment with the two systems.

We demonstrated stochastic aspects emerging from sampling across *Slps* and *Dlps*.

Strong emphasis on LP aspects as both packs are implemented in Prolog and are easily installed via the *SWI-Prolog* pack installer.

Any . . .

. . . questions ?