Notes on the implementation of FAM

Nicos Angelopoulos

nicos.angelopoulos@sanger.ac.uk

Cancer Genome Project

Sanger Institute

overview

Stochastic logic programs

FAM: EM for SLPs

Pepl: An implementation of FAM

Examples

Parameter estimation in Stochastic Logic Programs

James Cussens

Machine Learning (2001)

44 (3): 245-271. doi:10.1023/A:1010924021315

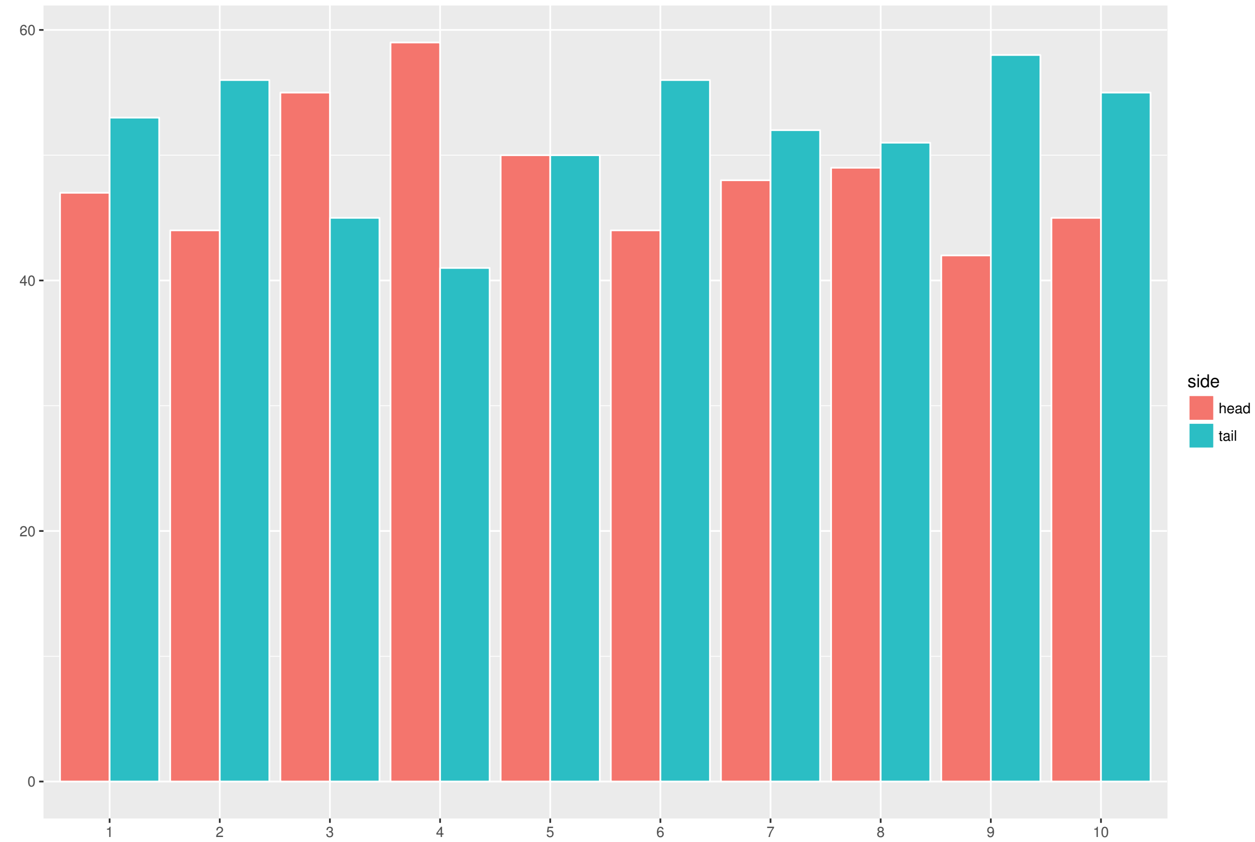
Stochastic Logic Programs (SLPs)

Labelled Logic Programs

Labels

```
    1/2 : coin(head).
    1/2 : coin(tail).
?- coin(X).
```

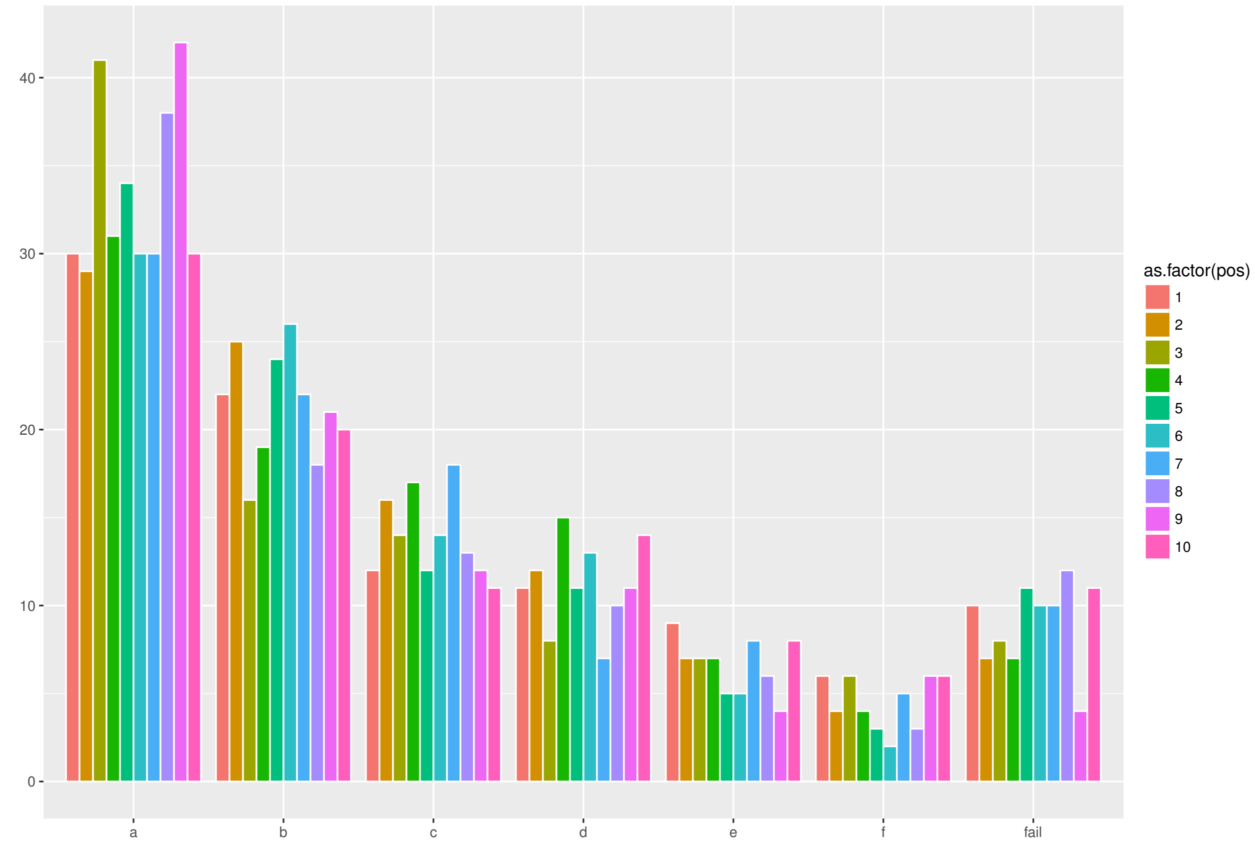
Sampling coins




```
1/2 : coin(head).
1/2 : coin(tail).

?- mlu_sample( coin(X), X, 100, 10, KVs ),
   mlu_plot( KVs ).
```

Recursive SLPs




```
1/3 : member3( H, [H|T] ).
2/3 : member3( El, [H|T] ) :-
                member3( El, T ).
```

recursive clauses- issues

There is no generic way to give uniform (or arbitrary distribution) for selecting a member from a list...

```
1/3 : member3( A, [A,_,_] ).
1/3 : member3( B, [_,B,_] ).
1/3 : member3( C, [_,_,C] ).
...
```

The name of the game:
parameter estimation

Given (a) an SLP, with a set of (b)initial parameters, (c) a query against this SLP, and (d) a dataset that connects observations/instantiantions of this query to frequencies we want to

find the set of parameters that maximise the likelihood of the data

FAM: EM for SLPs

Count frequencies with which each clause is involved in:

- unambiguous atom
- ambiguous atom
- failure derivation

$$\psi_{\lambda^{(h)}}[\nu_i \mid y] = \sum_{k=1}^{t-1} N_k \psi_{\lambda^{(h)}}[\nu_i \mid y_k] + N(Z_{\lambda^{(h)}}^{-1} - 1)\psi_{\lambda^{(h)}}[\nu_i \mid fail]$$

(1)

FAM: the algorithm

0. Let h = 0, and $\lambda^{(0)}$ such that $Z_{\lambda^{(0)}} > 0$.
1. For each parameterised clause C_i compute $\psi_{\lambda^{(h)}}[\nu_i \mid y]$ using (1) (*ML-Eq.8*).
2. For each parameterised clause C_i let $S_i^{(h)}$ be the sum of the expected counts $\psi_\lambda^{(h)}[\nu_{i'} \mid y]$ for all the clauses $C_{i'}^+$ such that $C_{i'}^+$ shares the predicate symbol as C_i.
3. For each parameterised clause C_i, if $S_i^{(h)} = 0$ then $l_i^{h+1} = l_i^{(h)}$ otherwise

$$l_i^{(h+1)} = \frac{\psi_\lambda^{(h)}[\nu_i \mid y]}{S_i^{(h)}}$$

4. $h \leftarrow h + 1$ and go to 1 unless $\lambda(h+1)$ has converged

Pepl

Pepl

is a Prolog library implementing FAM for SLPs

- current version $(2.0.6)$
- comes with a few canned examples
- 3 ways of calculating the scores
- easy installation

installation

SWI-Prolog

```
?- pack_install( pepl ).
```

Yap (6.3)

Need to download and untar the sources from

`http://stoics.org.uk/~nicos/sware/pepl`

usage

load with

```
?- use_module( library(pepl) ).
```

test

```
?- [main], main.
```

counting

- exact
- sampling
- stored

bloodtype example

```
bloodtype(a):- genotype(a,a).
bloodtype(a):- genotype(a,o).
bloodtype(a):- genotype(o,a).
bloodtype(b):- genotype(b,b).
bloodtype(b):- genotype(b,o).
bloodtype(b):- genotype(o,b).
bloodtype(o):- genotype(o,o).
bloodtype(ab):- genotype(a,b).
bloodtype(ab):- genotype(b,a).
genotype(X,Y):- gene(X),gene(Y).
1/3 :: gene( a ).
1/3 :: gene( b ).
1/3 :: gene( o ).
```

bloodtype example

```
main_exact :-
      fam( [
             goal(bloodtype(_A)),
             slp( '../slp/prism_bt' ),
             data([bloodtype(a)-4,bloodtype(b)-2,
                    bloodtype(o)-3,bloodtype(ab)-1 ]),
             count(exact), termin([iter(15)])
        ] ).
```

convergance

```
Initial parameters.
 11:0.3333333333333331483,12:0.3333333333333331483
 13:0.3333333333333331483
log_likelihood(-14.68742486079359)
Iteration(1).
 11:0.3166666666666665186,12:0.1833333333333334814
 13:0.5000000000000000000
log_likelihood(-12.867527895731104)
Iteration(2).
 11:0.2981012658227847889l,12:0.1654929577464788748O
 13:0.5364057764307362807
log_likelihood(-12.80273557775792)
Iteration(3).
 11:0.2934894563330276984,12:0.1633644799262847779
 13:0.5431460637406874125
log_likelihood(-12.800558696834383)
Iteration(4).
 11:0.2925414369601421760,12:0.1630727496624192474
 13:0.5443858133774386320
log_likelihood(-12.80048249699677 9)
```

Stochastic pallidromic grammar

```
'0.3' :: s --> [a], s, [a].
'0.2' :: s --> [b], s, [b].
'0.1' :: s --> [a],[a].
'0.4' :: s --> [b],[b].
```

learning grammar parameters

```
main_exact :-
    main_gen( Results ),
    pepl:list_frequency( Results, FreqRes ),
    keysort( FreqRes, SortRes ),
    pepl:dbg_ls_pepl( sorted_results, SortRes ),
    fam( [
              goal(phrase(s,_A,[])),
              data(SortRes),
              prior(uniform),
              eps(1.0e-4),
              count(exact),
              termin([iter(6)])
        ] ).
```

learning grammar parameters

```
Initial parameters.
  1:0.25000000000000000000,2:0.25000000000000000000
  3:0.25000000000000000000,4:0.25000000000000000000
log_likelihood(-2424.895790866377)
Iteration(1).
  1:0.32014022443378076233,2:0.21563474429112902686
  3:0.09652566424557013081,4:0.36769936702951994123
log_likelihood(-2215.2751506968234)
Iteration(2).
  1:0.33512440321377318098,2:0.21188328541353049217
  3:0.09439155913279248522,4:0.35860075223990400817
log_likelihood(-2215.7881878681233)
Iteration(3).
  1:0.34303435608830179504,2:0.21443932071504043235
  3:0.09170113806671904844,4:0.35082518512993871029
log_likelihood(-2217.720234860738)
Iteration(4).
  1:0.34824909414362670290,2:0.21768013996541976662
  3:0.09012170293373986119,4:0.34394906295721355827
log_likelihood(-2219.6961257280636)
Iteration(5).
  1:0.35112403332959135627,2:0.22277056303243245039
```

bottom-line

FAM is a robust, specialised EM
that is relevant to PLP in general

Restrictive expressivity of SLP recursive calls,
but this makes it suitable for estimating parameters

Pepl is an easy to install implementation of FAM for SLPs