Hypothesis Formation

Given background knowledge B and positive examples $E^+ = e_1 \wedge e_2 \ldots$, negative examples E^- ILP systems are concerned with finding a hypothesis $H = D_1 \wedge \ldots$ that satisfies (note: $\cup$ and $\wedge$ used interchangeably)

Posterior Sufficiency. $B \wedge H \models E^+$ and
$B \wedge D_j \models e_1 \vee e_2 \vee \ldots$

Posterior Satisfiability. $B \wedge H \wedge E^- \not\models \Box$

Recall that if more than one H satisfies this, the one with highest posterior probability is chosen

The D_i can be found by examining clauses that "relatively subsume" at least one example

Single Example, Single Hypothesis Clause

What does it mean for clause D to "relatively subsume" example e

- Normal subsumption: $D \succeq e$ means $\exists\theta\ s.t.\ D\theta \subseteq e$. This also means $D\theta \models e$ or $\models (e \leftarrow D\theta)$

$$
\begin{array}{lc}
e: & gfather(henry, john) \leftarrow \\
B: & father(henry, jane) \leftarrow \\
 & father(henry, joe) \leftarrow \\
 & parent(jane, john) \leftarrow \\
 & parent(joe, robert) \leftarrow \\
D: & gfather(X, Y) \leftarrow father(X, Z), parent(Z, Y)
\end{array}
$$

- Note that for this B, D, e with $\theta = \{X/henry, Y/john, Z/jane\}$, $B \cup \{D\theta\} \models e$

- That is: $D \succeq_B e$ means $B \models (e \leftarrow D\theta)$ Clearly if $B = \emptyset$ normal subsumption between clauses results.

– Using the Deduction Theorem

$$\begin{aligned} B \models (e \leftarrow D\theta) &\equiv B \cup \{D\theta\} \models e \\ &\equiv B \cup \overline{e} \models \overline{D\theta} \\ &\equiv \{D\theta\} \models \overline{B \cup \overline{e}} \\ &\equiv\ \models (\overline{B \cup \overline{e}} \leftarrow D\theta) \end{aligned}$$

– That is, $D \succeq_B e$ means $D \succeq \overline{B \cup \overline{e}}$

– Recall that if $C_1 \succeq C_2$ then $C_1 \models C_2$. In fact, if $C_{1,2}$ are not self-recursive, then $C_1 \succeq C_2 \equiv C_1 \models C_2$

– Let $a_1 \wedge a_2 \ldots$ be the ground literals true in all models of $B \cup \overline{e}$. Then

$$B \cup \overline{e} \models a_1 \wedge a_2 \ldots$$
$$\overline{a_1 \wedge a_2 \wedge \ldots} \models \overline{B \cup \overline{e}}$$

– Let $\bot(B, e) = \overline{a_1 \wedge a_2 \wedge \ldots}$.

– if $D \succeq \bot(B, e)$ then $D \models \bot(B, e)$ and therefore $D \models \overline{B \cup \overline{e}}$.

– In fact, it can be shown that if D, e are not self-recursive and $D \succeq \bot(B, e)$ then $D \succeq \overline{B \cup \overline{e}}$ (that is, $D \succeq_B e$)

A Sufficient Implementation (given B, E)

1. $h_0 = B, i = 0, E^+ = \{e_1, \ldots, e_n\}$

2. repeat

   (a) increment i

   (b) Obtain the most specific clause $\bot(B, e_i)$

   (c) Find the clause D_i that: subsumes $\bot(B, e_i)$; and is consistent with the negative examples;

   (d) $h_i = h_{i-1} \cup \{D_i\}$

3. until $i > n$

4. return h_n

- $\perp(B, e_i)$ may be infinite
- May perform a lot of redundant computation ($D_i \in h_{i-1}$)
- Need not return in the hypothesis with maximum posterior probability

A "Greedy" Implementation (given B, E)

1. $h_0 = B, E_0^+ = E^+, i = 0$

2. repeat

   (a) increment i

   (b) Randomly choose a positive example e_i from E_{i-1}^+

   (c) Obtain the most specific clause $\bot(B, e_i)$

   (d) Find the clause D_i that: subsumes $\bot(B, e_i)$; and is consistent with the negative examples; and maximises $p(h_{i-1} \cup \{D_i\} | e_i^+ \cup E^-)$ where e_i^+ are the examples in E^+ made redundant by $h_{i-1} \cup \{D_i\}$

   (e) $h_i = h_{i-1} \cup \{D_i\}$

   (f) $E_i^+ = E_{i-1}^+ \backslash e_i^+$

3. until $E_i^+ = \emptyset$

4. return h_i

– $\perp(B, e_i)$ may be infinite

– Need not return in the hypothesis with maximum posterior probability

Finding $\bot$: an example

B:

gfather(X,Y) $\leftarrow$ father(X,Z), parent(Z,Y)
father(henry,jane) $\leftarrow$
mother(jane,john) $\leftarrow$
mother(jane,alice) $\leftarrow$

e_i**:**

gfather(henry,john) $\leftarrow$

Conjunction of ground atoms provable from $B \cup \overline{e_i}$:

$\neg$parent(jane,john) $\wedge$
father(henry,jane) $\wedge$
mother(jane,john) $\wedge$
mother(jane,alice) $\wedge$
$\neg$gfather(henry,john)

$\bot(B, e_i)$**:**

gfather(henry,john) $\vee$ parent(jane,john) $\leftarrow$
father(henry,jane),
mother(jane,john),
mother(jane,alice)

D_i**:**

parent(X,Y) $\leftarrow$ mother(X,Y)

Ways of obtaining a finite $\bot$: depth-bounded mode language

Finding a clause D_i that subsumes $\bot(B, e_i)$ is hampered by the fact that $\bot(B, e_i)$ may be infinite!

Use constrained subset of definite clauses to construct finite most-specific clauses

Mode declarations

modeh(,gfather(+person,-person))*

modeh(,parent(+person,-person))*

modeb(,father(+person,-person))*

modeb(,parent(+person,-person))*

modeb(,mother(+person,-person))*

Definite mode language

Let $C : h \leftarrow b_1, \ldots, b_n$ be a definite clause with an ordering over literals. Let M be a set of mode declarations. C is in the definite mode language $\mathcal{L}(M)$ iff

1. h is the atom of a *modeh* declaration in M with every place-marker of $+type$ and $-type$ replaced with variables, and every place marker of $\#type$ replaced by a ground term.

2. Every atom b_i in body of C is an atom in a *modeb* declaration in M with $+, -, \#$ places being replaced as above.

3. Every variable of $+type$ in b_i is either of $+type$ in h or or $-type$ in a b_j $(1 \leq j < i)$

Given a set of mode declarations M it is always possible to decide if a clause C is in $\mathcal{L}(M)$

Depth of variables. Let C be a definite clause, v be a variable in an atom in C, and U_v all other variables in body atoms of C that contain v

$$d(v) = \begin{cases} 0 & \text{if } v \text{ in head of } C \\ (\max_{u \in U_v} d(u)) + 1 & \text{otherwise} \end{cases}$$

$C : gfather(X, Y) \leftarrow father(X, Z), parent(Z, Y)$

Then $d(X) = d(Y) = 0$, $d(Z) = 1$

Depth bounded definite mode language

Let C be a definite clause with an ordering over literals. Let M be a set of mode declarations. C is in the depth-bounded definite mode language $\mathcal{L}_d(M)$ iff all variables in C have depth at most d

The clause for $gfather/2$ earlier is in $\mathcal{L}_2(M)$

For every $\perp(B, e_i)$ it is the case that

- There is a $\perp_d(B, e_i)$ in $\mathcal{L}_d(M)$ s.t. $\perp_d(B, e_i) \succeq \perp(B, e_i)$

- $\perp_d(B, e_i)$ is finite

- If $C \succeq \perp_d(B, e_i)$ then $C \succeq \perp(B, e_i)$

Finding $\bot_i$: an example

$\bot(B,\ e_i)$:

gfather(henry,john) $\vee$ parent(jane,john) $\leftarrow$
father(henry,jane),
mother(jane,john),
mother(jane,alice)

modes:

modeh(,parent(+person,-person))*
modeb(,mother(+person,-person))*
modeb(,father(+person,-person))*

$\bot_0(B, e_i)$:

parent(X,Y) $\leftarrow$

$\bot_1(B, e_i)$:

parent(X,Y) $\leftarrow$
mother(X,Y),
mother(X,Z)

Revised "Greedy" Implementation (given B, E, d)

1. $h_0 = B, E_0^+ = E^+, i = 0$

2. repeat

   (a) increment i

   (b) Randomly choose a positive example e_i from E_{i-1}^+

   (c) Obtain the most specific clause $\bot_d(B, e_i)$

   (d) Find the clause D_i that: subsumes $\bot(B, e_i)$; and is consistent with the negative examples; and maximises $p(h_{i-1} \cup \{D_i\} | e_i^+ \cup E^-)$ where e_i^+ are the examples in E^+ made redundant by $h_{i-1} \cup \{D_i\}$

   (e) $h_i = h_{i-1} \cup \{D_i\}$

   (f) $E_i^+ = E_{i-1}^+ \backslash e_i^+$

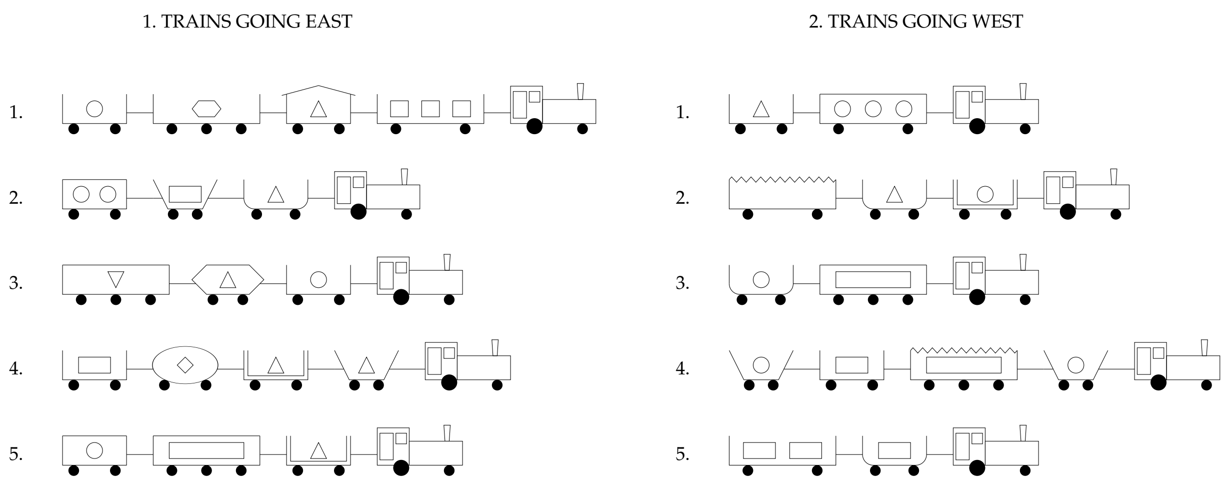
3. until $E_i^+ = \emptyset$

4. return h_i

– Need not return in the hypothesis with maximum posterior probability

Question. How should the implementation be modified so that it returns the hypothesis with maximum posterior probability?

An example: trainspotting



Trainspotting: Modes

```
:- modeh(1,eastbound(+train)).
:- modeb(1,short(+car)).
:- modeb(1,closed(+car)).
:- modeb(1,long(+car)).
:- modeb(1,open_car(+car)).
:- modeb(1,double(+car)).
:- modeb(1,jagged(+car)).
:- modeb(1,shape(+car,#shape)).
:- modeb(1,load(+car,#shape,#int)).
:- modeb(1,wheels(+car,#int)).
:- modeb(*,has_car(+train,-car)).
```

Trainspotting: Examples & Background

Positive	Negative
eastbound(east1).	eastbound(west6).
eastbound(east2).	eastbound(west7).
eastbound(east3).	eastbound(west8).
eastbound(east4).	eastbound(west9).
eastbound(east5).	eastbound(west10).

```
% type definitions
car(car_11).  car(car_12). ...
car(car_21).  car(car_22).  ...
...

shape(elipse).  shape(hexagon). ...
...

% eastbound train 1
has_car(east1,car_11).  has_car(east1,car_12). ...
shape(car_11,rectangle). shape(car_12,rectangle). ...
open_car(car_11). closed(car_12).
long(car_11).  short(car_12). ...
...

% westbound train 6
has_car(west6,car_61).  has_car(west6,car_62). ...
long(car_61).  short(car_62).
shape(car_61,rectangle).  shape(car_62,rectangle).
...
```

Trainspotting: Search

```
eastbound(A) :-
   has_car(A,B).
[5/5]
eastbound(A) :-
   has_car(A,B), short(B).
[5/5]
eastbound(A) :-
   has_car(A,B), open_car(B).
[5/5]
eastbound(A) :-
   has_car(A,B), shape(B,rectangle).
[5/5]

...

[theory]

[Rule 1] [Pos cover = 5 Neg cover = 0]

eastbound(A) :-
       has_car(A,B), short(B), closed(B).

[pos-neg] [5]
```