REDUCTION OF ILP SEARCH SPACE WITH BOTTOM-UP PROPOSITIONALISATION

Hadeel Al-Negheimish and Alessandra Russo

Imperial College London
King Saud University

halnegheimish@ksu.edu.sa
MOTIVATION

• A typical learning task takes explicit language bias as input
• Specification of language bias requires human intuition:
  • Too constrained: Risks not finding a solution
  • Too inclusive: Might make search intractable
MOTIVATION

• A typical learning task takes explicit language bias as input
• Specification of language bias requires human intuition:
  • Too constrained: Risks not finding a solution
  • Too inclusive: Might make search intractable

How can we restrict the search space in a more automated way?
RELATED WORK

• Extracting meta-knowledge to restrict search e.g. [McCreath and Sharma, 1995]

• ILP frameworks without mode declarations
  • FOIL [Quinlan, 1990] and Meta-Interpretive Learning [Muggleton, 2015]

• Propositionalisation e.g. [Lavrač et al., 1991]

RELATED WORK

• Extracting meta-knowledge to restrict search e.g. [McCreath and Sharma, 1995]

• ILP frameworks without mode declarations
  • FOIL [Quinlan, 1990] and Meta-Interpretive Learning [Muggleton, 2015]

• Propositionalisation e.g. [Lavrač et al., 1991]

LEARNING TASK

Let $L_{BPILP}$ be a language and let $T$ be the learning task of our approach, such that $T = \langle B, E^+, E^- \rangle$ is defined in $L_{BPILP}$, where:

**B** Background knowledge, a stratified normal logic program, with no occurrence of the target predicate anywhere in the bodies of rules.

**E+** Positive examples, a set of ground atoms of the same predicate with no functor symbols.

**E-** Negative examples, also a set of ground atoms with no functor symbols of the same predicate in $E^+$.
LEARNING TASK

Let $\mathcal{L}_{\text{BPILP}}^*$ be an extended language defined as \{${\mathcal{L}_{\text{BPILP}}} \cup \varphi$\} where $\varphi$ is an invented predicate that does not appear anywhere in $\mathcal{L}_{\text{BPILP}}$. A solution $H$ is a (set of) normal clause(s) such that:

1. $H \in \mathcal{L}_{\text{BPILP}}^*$
2. $\forall e^+ \in E^+ : B \cup H \models e^+$
3. $\forall e^- \in E^- : B \cup H \not\models e^-$
OUR APPROACH

• We develop an algorithmic approach that interleaves incremental feature-construction and selecting useful features using:
  • **Bottom-up Propositionalisation**
  • **Extended Set Operations** - Takes into account generality

• It works on normal logic observational learning tasks
• **Background**

child(X,Y):= son(X,Y).
child(X,Y):= daughter(X,Y).
son(sam, sara).
daughter(alice, mary).
female(sara).
female(alice).
female(mary).
male(sam).

• **Positive Examples**

mother(sara, sam).
mother(mary, alice).

• **Negative Examples**

mother(alice, mary).
mother(sam, alice).
EXAMPLE

1. Get model(B):

\[
\begin{align*}
&\text{child}(\text{sam}, \text{sara}). & \quad & \text{female}(\text{sara}). \\
&\text{child}(\text{alice}, \text{mary}). & \quad & \text{female}(\text{alice}). \\
&\text{son}(\text{sam}, \text{sara}). & \quad & \text{female}(\text{mary}). \\
&\text{daughter}(\text{alice}, \text{mary}). & \quad & \text{male}(\text{sam}).
\end{align*}
\]

2. Find Target:

The head of the main clause of the hypothesis is the least general-generalisation of the positive examples

\[
lgg(\text{mother}(\text{sara}, \text{sam}), \text{mother}(\text{mary}, \text{alice})) = \text{mother}(X,Y)
\]
• With mother(X,Y) as target

<table>
<thead>
<tr>
<th></th>
<th>(e_1) (sara, sam)</th>
<th>(e_2) (mary, alice)</th>
<th>(e_3) (alice, mary)</th>
<th>(e_4) (sam, alice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping (\mathcal{M}(e))</td>
<td>sara (\mapsto) X</td>
<td>mary (\mapsto) X</td>
<td>alice (\mapsto) X</td>
<td>sam (\mapsto) X</td>
</tr>
<tr>
<td></td>
<td>sam (\mapsto) Y</td>
<td>alice (\mapsto) Y</td>
<td>mary (\mapsto) Y</td>
<td>alice (\mapsto) Y</td>
</tr>
</tbody>
</table>

1. Find mapping of example terms to head vars
2. Find relevant atoms in model(B) by term linking
3. Generalise links wrt \(\mathcal{M}(e)\)
• With mother(X,Y) as target

1. Find mapping of example terms to head vars
2. Find relevant atoms in model(B) by term linking
3. Generalise links wrt \( M(e) \)

<table>
<thead>
<tr>
<th>Mapping ( M(e) )</th>
<th>( e_1 ) (sara, sam) +</th>
<th>( e_2 ) (mary, alice) +</th>
<th>( e_3 ) (alice, mary) -</th>
<th>( e_4 ) (sam, alice) -</th>
</tr>
</thead>
<tbody>
<tr>
<td>sara ↦ X</td>
<td>mary ↦ X</td>
<td>alice ↦ X</td>
<td>sam ↦ X</td>
<td></td>
</tr>
<tr>
<td>sam ↦ Y</td>
<td>alice ↦ Y</td>
<td>mary ↦ Y</td>
<td>alice ↦ Y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Links(e)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Features(e)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
With mother(X,Y) as target

<table>
<thead>
<tr>
<th>e₁ (sara, sam) ⊕</th>
<th>e₂ (mary, alice) ⊕</th>
<th>e₃ (alice, mary) ⊗</th>
<th>e₄ (sam, alice) ⊗</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping M(e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sara ↦ X</td>
<td>mary ↦ X</td>
<td>alice ↦ X</td>
<td>sam ↦ X</td>
</tr>
<tr>
<td>sam ↦ Y</td>
<td>alice ↦ Y</td>
<td>mary ↦ Y</td>
<td>alice ↦ Y</td>
</tr>
<tr>
<td>Links(e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features(e)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Find mapping of example terms to head vars
2. Find relevant atoms in model(B) by term linking
3. Generalise links wrt M(e)
With mother(X,Y) as target

Select Features:
\[\bigcap^* feats(e^+) \quad \forall e^+ \in E^+ \]
\[\bigcup feats(e^-) \setminus * \quad \bigcup feats(e^+)\]

<table>
<thead>
<tr>
<th>e₁</th>
<th>e₂</th>
<th>e₃</th>
<th>e₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>(sara, sam) ⊕</td>
<td>(mary, alice) ⊕</td>
<td>(alice, mary) ⊙</td>
<td>(sam, alice) ⊙</td>
</tr>
</tbody>
</table>

**Mapping M(e)**
- sara \(\mapsto\) X
- sam \(\mapsto\) Y
- mary \(\mapsto\) X
- alice \(\mapsto\) Y
- alice \(\mapsto\) X
- mary \(\mapsto\) Y
- sam \(\mapsto\) X
- alice \(\mapsto\) Y

**Links(e)**
- child(sam, sara).
- son(sam, sara).
- female(sara).
- male(sam).
- child(alice, mary).
- daughter(alice, mary).
- female(alice).
- female(mary).
- child(alice, mary).
- daughter(alice, mary).
- female(alice).
- female(mary).
- child(sam, sara).
- son(sam, sara).
- daughter(alice, mary).
- female(alice).
- male(sam).
- child(sam, sara).
- child(alice, mary).
- son(sam, sara).
- daughter(alice, mary).
- female(alice).
- female(mary).
- child(X, SK).
- child(Y, SK).
- son(X, SK).
- daughter(Y, SK).
- female(Y).
- male(X).

**Features(e)**
- child(Y, X).
- son(Y, X).
- female(X).
- male(Y).
- child(Y, X).
- daughter(Y, X).
- female(Y).
- female(X).
- child(X, Y).
- daughter(X, Y).
- female(X).
- female(Y).
- child(X, Y).
- daughter(X, Y).
- female(X).
- female(Y).
- child(X, SK).
- child(Y, SK).
- son(X, SK).
- daughter(Y, SK).
- female(Y).
- male(X).
Saturated Hypothesis:
mother(X,Y):- child (Y,X), female(X), not child(X,Y),
not daughter(X,Y), not male(X).

Reduced Hypothesis:
mother(X,Y):- child (Y,X), female(X).
SUMMARY

• Bottom-up Propositionalisation (*feature construction*)
  • Get mapping $M$
  • Find links via terms
  • Generalise atoms in Links wrt $M$

• Extended Set Operations (*feature selection*)
  • If a simple solution is not found, process is repeated to expand skolem variables or invent predicates
THEORETICAL PROPERTIES

- Stratification of B U H
- Soundness
- Feature minimality
- Existence of solution
evaluate

- Animals Dataset
  - Divided to 4 tasks, each with one of the class constants
  - Resulting Hypothesis:

\[
\text{class}(A, \text{mammal}) :- \text{has\_milk}(A).
\text{class}(A, \text{fish}) :- \text{has\_gills}(A).
\text{class}(A, \text{bird}) :- \text{has\_covering}(A, \text{feathers}).
\text{class}(A, \text{reptile}) :- \text{has\_covering}(A, \text{scales}),
\text{not has\_gills}(A).
\]
EVALUATION

- **Animals Dataset**
  - Divided to 4 tasks, each with one of the class constants
  - Resulting Hypothesis:

```prolog
class(A, mammal):- has_milk(A).
class(A, fish):- has_gills(A).
class(A, bird):- has_covering(A, feathers).
class(A, reptile):- has_covering(A, scales), not has_gills(A).
```
• Eastbound Trains

- 1. TRAINS GOING EAST
- 2. TRAINS GOING WEST

- with eastbound as $E^+$
  
  \[
  \text{eastbound}(A) :\neg \ \text{closed}(V1), \ \text{shape}(V1,V2), \ \text{car}(V1), \ \text{wheels}(V1,V3), \ \text{load}(V1,V4,V5), \ \text{short}(V1), \ \text{has\_car}(A,V1).
  \]

- with westbound as $E^+$
  
  \[
  \text{westbound}(A) :\neg \ \text{inv}(A).
  \]
  \[
  \text{inv}(A) :\neg \ \text{closed}(V1), \ \text{shape}(V1,V2), \ \text{car}(V1), \ \text{wheels}(V1,V3), \ \text{load}(V1,V4,V5), \ \text{short}(V1), \ \text{has\_car}(A,V1).
  \]
• **Eastbound Trains**

1. TRAINS GOING EAST

2. TRAINS GOING WEST

• with eastbound as $E^+$

```
eastbound(A):- closed(V1), shape(V1,V2), car(V1),
wheels(V1,V3), load(V1,V4,V5), short(V1), has_car(A,V1).
```

• with westbound as $E^+$

```
westbound(A):- not inv(A).
inv(A):- closed(V1), shape(V1,V2), car(V1),
wheels(V1,V3), load(V1,V4,V5), short(V1), has_car(A,V1).
```
CONCLUSION

• We have introduced a method for bottom-up propositionalisation and used it for algorithmic learning without explicit language bias
CONCLUSION

• We have introduced a method for bottom-up propositionalisation and used it for algorithmic learning without language bias

Strengths
• It learns normal logic programs
• Supports predicate invention
CONCLUSION

• We have introduced a method for bottom-up propositionalisation and used it for algorithmic learning without language bias

**Strengths**
- It learns normal logic programs
- Supports predicate invention

**Weaknesses**
- Observational learning only
- Works best on single-clause hypothesis
- Does not work well with recursive definitions
• Let $x, y$ be head variables

• Let $A = \{p(x, x), p(SK, y), q(x)\}$
  and $B = \{p(x, x), p(x, y), q(y)\}$

• $A \cap^* B =$
• Let $x, y$ be head variables

• Let $A = \{p(x, x), p(SK, y), q(x)\}$ and $B = \{p(x, x), p(x, y), q(y)\}$

• $A \cap^* B = \{p(x, x), ..\}$ since $p(x, x) \models p(x, x)$
Let $x, y$ be head variables

Let $A = \{p(x, x), p(SK, y), q(x)\}$ and $B = \{p(x, x), p(x, y), q(y)\}$

$A \cap^* B = \{p(x, x), p(SK, y)\}$ since $p(SK, y) \models p(x, y)$
Let $x, y$ be head variables

Let $A = \{p(x, x), p(SK, y), q(x)\}$ and $B = \{p(x, x), p(x, y), q(y)\}$

$A \cap^* B = \{p(x, x), p(SK, y)\}$

$q(x) \not\equiv q(y)$

Since they are head variables they are treated as constants and cannot be unified to another value
Recursive Definitions

• At its current implementation, this approach does not work very well with recursive definitions.

• This has to do with checking feature coverage; the target predicate is treated the same as other features.
Recursive Definitions

• Learnable Recursive Theory

• \( \text{vp}(X,Y) :- \text{vp}(X,Z), \text{modif}(Z,Y) \)
Recursive Definitions

- Unlearnable Recursive Theory

- \texttt{ancestor}(X,Y) :- \texttt{parent}(X,Z), \texttt{ancestor}(Z,Y)

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
\textbf{B} & \textbf{E}^+ & \textbf{E}^- \\
\hline
\texttt{parent}(a,b). & \texttt{ancestor}(a,c). & \texttt{ancestor}(d,a). \texttt{ancestor}(f,e). \\
\texttt{parent}(b,c). & \texttt{ancestor}(a,d). & \\
\texttt{parent}(b,f). & \texttt{ancestor}(b,e). & \\
\texttt{parent}(f,g). & \texttt{ancestor}(a,e). & \\
\texttt{parent}(c,d). & & \\
\texttt{parent}(d,e). & & \\
\texttt{ancestor}(X,Y) :- & & \\
\texttt{parent}(X,Y). & & \\
\hline
\end{tabular}
\end{center}
Multiple Mappings

- Let target be $p(X,Y)$
- Let $e = p(bob, bob)$
- $\mathcal{M}(e) = \{ bob \mapsto \{X,Y\}\}$

**How do we generalise links in this case?**

A feature is generated for each mapping.
Multiple Mappings

- Let target be \( p(X,Y) \)
- Let \( e = p(bob, bob) \)
- \( M(e) = \{ bob \mapsto \{X,Y\}\} \)

If \( q(bob) \) is in \( \text{Links}(e) \) then both \( q(X) \) and \( q(Y) \) are generalisations with respect to \( M(e) \)
Reduction In Feature Templates

• Let $t = \|\text{head vars}\|$.

• Any atom can be generalised in $(t+1)^n$ ways where $n$ is the arity of the predicate.
Reduction In Feature Templates

• Example:

Let \( t = 2 \) (target = \( p(X,Y) \))

let \( r/2 \) be a predicate, it can be generalised as

- \( r(X,X) \)
- \( r(X,Y) \)
- \( r(X, SK) \)
- \( r(Y,X) \)
- \( r(Y,Y) \)
- \( r(Y, SK) \)
- \( r(SK, X) \)
- \( r(SK, Y) \)
- \( r(SK, SK) \)