Propositionalisation as a way of understanding RDM and ILP

Peter A. Flach
University of Bristol, UK
www.cs.bris.ac.uk/~flach/
Objectives of this lecture

- To clarify the relation between RDM and attribute-value learning, also called propositional learning

- To clarify the role of representation in RDM, and the relation between different representation formalisms such as Prolog, Datalog, and databases

- To provide a feeling for the key steps in RDM algorithms, and where their complexity resides
A “toy” example

1. TRAINS GOING EAST
2. TRAINS GOING WEST

1. 
2. 
3. 
4. 
5. 

1. 
2. 
3. 
4. 
5.
### TRAIN_TABLE

<table>
<thead>
<tr>
<th>TRAIN</th>
<th>EAS TBOUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>TRUE</td>
</tr>
<tr>
<td>t2</td>
<td>TRUE</td>
</tr>
<tr>
<td>t6</td>
<td>FALSE</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### LOAD_TABLE

<table>
<thead>
<tr>
<th>LOAD</th>
<th>CAR</th>
<th>OBJECT</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>c1</td>
<td>circle</td>
<td>1</td>
</tr>
<tr>
<td>l2</td>
<td>c2</td>
<td>hexagon</td>
<td>1</td>
</tr>
<tr>
<td>l3</td>
<td>c3</td>
<td>triangle</td>
<td>1</td>
</tr>
<tr>
<td>l4</td>
<td>c4</td>
<td>rectangle</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### CAR_TABLE

<table>
<thead>
<tr>
<th>CAR</th>
<th>TRAIN</th>
<th>SHAPE</th>
<th>LENGTH</th>
<th>ROOF</th>
<th>WHEELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>t1</td>
<td>rectangle</td>
<td>short</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>c2</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>3</td>
</tr>
<tr>
<td>c3</td>
<td>t1</td>
<td>rectangle</td>
<td>short</td>
<td>peaked</td>
<td>2</td>
</tr>
<tr>
<td>c4</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example:
eastbound(t1).

Background knowledge:

hasCar(t1,c1). hasCar(t1,c2). hasCar(t1,c3). hasCar(t1,c4).
cshape(c1,rect). cshape(c2,rect). cshape(c3,rect). cshape(c4,rect).
clength(c1,short). clength(c2,long). clength(c3,short). clength(c4,long).
croof(c1,none). croof(c2,none). croof(c3,peak). croof(c4,none).
cwheels(c1,2). cwheels(c2,3). cwheels(c3,2). cwheels(c4,2).
hasLoad(c1,l1). hasLoad(c2,l2). hasLoad(c3,l3). hasLoad(c4,l4).
lnumber(l1,1). lnumber(l2,1). lnumber(l3,1). lnumber(l4,3).

Hypothesis:
eastbound(T):-hasCar(T,C),clength(C,short),
not croof(C,none).
Example:

\[ \text{eastbound}(t1). \]

Background knowledge:

- \text{hasCar}(t1, c1).
- \text{hasCar}(t1, c2).
- \text{hasCar}(t1, c3).
- \text{hasCar}(t1, c4).
- \text{cshape}(c1, \text{rect}).
- \text{cshape}(c2, \text{rect}).
- \text{cshape}(c3, \text{rect}).
- \text{cshape}(c4, \text{rect}).
- \text{clength}(c1, \text{short}).
- \text{clength}(c2, \text{long}).
- \text{clength}(c3, \text{short}).
- \text{clength}(c4, \text{long}).
- \text{croof}(c1, \text{none}).
- \text{croof}(c2, \text{none}).
- \text{croof}(c3, \text{peak}).
- \text{croof}(c4, \text{none}).
- \text{cwheels}(c1, 2).
- \text{cwheels}(c2, 3).
- \text{cwheels}(c3, 2).
- \text{cwheels}(c4, 2).
- \text{hasLoad}(c1, l1).
- \text{hasLoad}(c2, l2).
- \text{hasLoad}(c3, l3).
- \text{hasLoad}(c4, l4).
- \text{lshape}(l1, \text{circ}).
- \text{lshape}(l2, \text{hexa}).
- \text{lshape}(l3, \text{tria}).
- \text{lshape}(l4, \text{rect}).
- \text{lnumber}(l1, 1).
- \text{lnumber}(l2, 1).
- \text{lnumber}(l3, 1).
- \text{lnumber}(l4, 3).

Hypothesis:

\[ \text{eastbound}(T) :\neg \text{hasCar}(T, C), \text{clength}(C, \text{short}), \]
\[ \quad \text{not croof}(C, \text{none}). \]
Example:

eastbound(t1):-
    hasCar(t1,c1),cshape(c1,rect),clength(c1,short),croof(c1,none),cwheels(c1,2),
    hasLoad(c1,l1),lshape(l1,circ),lnumber(l1,1),
    hasCar(t1,c2),cshape(c2,rect),clength(c2,long),croof(c2,none),cwheels(c2,3),
    hasLoad(c2,l2),lshape(l2,hexa),lnumber(l2,1),
    hasCar(t1,c3),cshape(c3,rect),clength(c3,short),croof(c3,peak),cwheels(c3,2),
    hasLoad(c3,l3),lshape(l3,tria),lnumber(l3,1),
    hasCar(t1,c4),cshape(c4,rect),clength(c4,long),croof(c4,none),cwheels(c4,2),
    hasLoad(c4,l4),lshape(l4,rect),lnumber(l4,3).

Background knowledge: empty

Hypothesis:

eastbound(T):-hasCar(T,C),clength(C,short),
    not croof(C,none).
Example:

\[
eastbound(t1) :-
\]

\[
\text{hasCar}(t1,c1), \text{cshape}(c1,\text{rect}), \text{clength}(c1,\text{short}), \text{croof}(c1,\text{none}), \text{cwheels}(c1,2),
\]

\[
\text{hasLoad}(c1,l1), \text{lshape}(l1,\text{circ}), \text{lnumber}(l1,1),
\]

\[
\text{hasCar}(t1,c2), \text{cshape}(c2,\text{rect}), \text{clength}(c2,\text{long}), \text{croof}(c2,\text{none}), \text{cwheels}(c2,3),
\]

\[
\text{hasLoad}(c2,l2), \text{lshape}(l2,\text{hexa}), \text{lnumber}(l2,1),
\]

\[
\text{hasCar}(t1,c3), \text{cshape}(c3,\text{rect}), \text{clength}(c3,\text{short}), \text{croof}(c3,\text{peak}), \text{cwheels}(c3,2),
\]

\[
\text{hasLoad}(c3,l3), \text{lshape}(l3,\text{tria}), \text{lnumber}(l3,1),
\]

\[
\text{hasCar}(t1,c4), \text{cshape}(c4,\text{rect}), \text{clength}(c4,\text{long}), \text{croof}(c4,\text{none}), \text{cwheels}(c4,2),
\]

\[
\text{hasLoad}(c4,l4), \text{lshape}(l4,\text{rect}), \text{lnumber}(l4,3).
\]

Background knowledge: empty

Hypothesis:

\[
eastbound(T) :- \text{hasCar}(T,C), \text{clength}(C,\text{short}),
\]

\[
\text{not croof}(C,\text{none}).
\]
Example:

```
eastbound([car(rect, short, none, 2, load(circ, 1)),
            car(rect, long, none, 3, load(hexa, 1)),
            car(rect, short, peak, 2, load(tria, 1)),
            car(rect, long, none, 2, load(rect, 3))]).
```

Background knowledge: member/2, arg/3

Hypothesis:

```
eastbound(T) :- member(C, T), arg(2, C, short),
                 not arg(3, C, none).
```
Example:

```
eastbound([car(rect, short, none, 2, load(circ, 1)),
            car(rect, long, none, 3, load(hexa, 1)),
            car(rect, short, peak, 2, load(tria, 1)),
            car(rect, long, none, 2, load(rect, 3))]).
```

Background knowledge: member/2, arg/3

Hypothesis:

```
eastbound(T) :- member(C, T), arg(2, C, short),
               not arg(3, C, none).
```
## Prolog representations: summary

<table>
<thead>
<tr>
<th></th>
<th>instance description</th>
<th>examples separated</th>
<th>explicit structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>flattened</strong></td>
<td>in background knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ground clauses</strong></td>
<td>in example</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>ground terms</strong></td>
<td>in example</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
ER diagram for East-West trains

- **Train**
  - Direction

- **Car**
  - Shape
  - Length
  - Roof
  - Wheels

- **Load**
  - Number
  - Object

Relationships:
- 1:1 relationship between Train and Car
- M:1 relationship between Car and Shape
- 1:1 relationship between Car and Load
- 1:1 relationship between Car and Has

Has relationships are depicted as diamonds with arrows pointing towards each entity.
Each train is a structured object

- **train1**
  - Direction
  - Has
    - **car1**
      - Shape
      - Length
      - Roof
      - Wheels
      - Has
        - **load1**
          - Number
          - Object
  - Has
    - **car2**
      - Shape
      - Length
      - Roof
      - Wheels
      - Has
        - **load2**
          - Number
          - Object
  - Has
    - **car3**
      - Shape
      - Length
      - Roof
      - Wheels
      - Has
        - **load3**
          - Number
          - Object
Train-as-set database

LOAD_TABLE

<table>
<thead>
<tr>
<th>LOAD</th>
<th>CAR</th>
<th>OBJECT</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>c1</td>
<td>circle</td>
<td>1</td>
</tr>
<tr>
<td>l2</td>
<td>c2</td>
<td>hexagon</td>
<td>1</td>
</tr>
<tr>
<td>l3</td>
<td>c3</td>
<td>triangle</td>
<td>1</td>
</tr>
<tr>
<td>l4</td>
<td>c4</td>
<td>rectangle</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

CAR_TABLE

<table>
<thead>
<tr>
<th>CAR</th>
<th>TRAIN</th>
<th>SHAPE</th>
<th>LENGTH</th>
<th>ROOF</th>
<th>WHEELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>t1</td>
<td>rectangle</td>
<td>short</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>c2</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>3</td>
</tr>
<tr>
<td>c3</td>
<td>t1</td>
<td>rectangle</td>
<td>short</td>
<td>peaked</td>
<td>2</td>
</tr>
<tr>
<td>c4</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

TRAIN_TABLE

<table>
<thead>
<tr>
<th>TRAIN</th>
<th>DIRECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>EAST</td>
</tr>
<tr>
<td>t2</td>
<td>EAST</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>t6</td>
<td>WEST</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

SELECT DISTINCT TRAIN_TABLE.TRAIN FROM TRAIN_TABLE, CAR_TABLE WHERE TRAIN_TABLE.TRAIN = CAR_TABLE.TRAIN AND CAR_TABLE.SHAPE = 'short' AND CAR_TABLE.ROOF != 'none'
ER diagram for sequences

- Train
  - Direction
  - First
    - Shape
    - Length
    - Roof
    - Wheels
  - Load
    - Number
    - Object

- Car
  - Has
    - Next

- Has
  - 1

- First
  - 1

- Train
  - 1
Train-as-sequence database

**LOAD_TABLE**

<table>
<thead>
<tr>
<th>LOAD</th>
<th>CAR</th>
<th>OBJECT</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>c1</td>
<td>circle</td>
<td>1</td>
</tr>
<tr>
<td>l2</td>
<td>c2</td>
<td>hexagon</td>
<td>1</td>
</tr>
<tr>
<td>l3</td>
<td>c3</td>
<td>triangle</td>
<td>1</td>
</tr>
<tr>
<td>l4</td>
<td>c4</td>
<td>rectangle</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**TRAIN_TABLE**

<table>
<thead>
<tr>
<th>TRAIN</th>
<th>FIRST</th>
<th>DIRECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>c1</td>
<td>EAST</td>
</tr>
<tr>
<td>t2</td>
<td>c5</td>
<td>EAST</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>t6</td>
<td>c13</td>
<td>WEST</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

**CAR_TABLE**

<table>
<thead>
<tr>
<th>CAR</th>
<th>NEXT</th>
<th>SHAPE</th>
<th>LENGTH</th>
<th>ROOF</th>
<th>WHEELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>c2</td>
<td>rectangle</td>
<td>short</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>c2</td>
<td>c3</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>3</td>
</tr>
<tr>
<td>c3</td>
<td>c4</td>
<td>rectangle</td>
<td>short</td>
<td>peaked</td>
<td>2</td>
</tr>
<tr>
<td>c4</td>
<td>NIL</td>
<td>rectangle</td>
<td>long</td>
<td>none</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**SQL Query**

```
SELECT DISTINCT TRAIN_TABLE.TRAIN FROM TRAIN_TABLE, CAR_TABLE WHERE
TRAIN_TABLE.??? = CAR_TABLE.??? AND
CAR_TABLE.SHAPE = 'short' AND
CAR_TABLE.ROOF != 'none'
```
Individual-centred representations

- ER diagram is a tree (approximately)
  - root denotes individual
  - looking downwards from the root, only one-to-one or one-to-many relations are allowed
  - one-to-one cycles are allowed

- Database can be partitioned according to individual

- Strongly related to the term representation
  - tuples, lists, sets, multisets, trees, ...
Strongly typed languages

- Type signature specifies ‘data model’
  - similar to ER diagram

- Each example described by single statement

- Hypothesis construction guided by types
  - interaction between structural functions/predicates referring to subterms and utility predicates giving properties of subterms

- Example language: Escher™
  - functional logic programming
  - syntax close to Haskell
Strongly typed East-West trains

Type signature:
eastbound :: Train → Bool;

type Train = [Car]; type Car = (CShape, CLength, CRoof, CWheels, Load);
type Load = (LShape, LNumber);

data CShape = Rect | Hexa | ...; data CLength = Long | Short;
data CRoof = None | Peak | ...; data LShape = Circ | Hexa | ...;

type CWheels = Int; type LNumber = Int

Example:
eastbound([(Rect, Short, None, 2, (Circ, 1)),
             (Rect, Long, None, 3, (Hexa, 1)),
             (Rect, Short, Peak, 2, (Tria, 1)),
             (Rect, Long, None, 2, (Rect, 3))]) = True

Hypothesis:
eastbound(t) = (exists \ c -> member(c, t) &&
                 CLengthP(c) == Short && CRoofP(c) != None)
Strongly typed mutagenesis

Type signature:

```
mutagenic::Molecule->Bool;

type Molecule = (Ind1, IndA, Lumo, LogP, AtomSet, BondSet);
type AtomSet = {Atom};
type Atom = (AtomID, Element, AtomType, Charge);
type BondSet = {Bond};
type Bond = (AtomIDSet, BondType);
type AtomIDSet = {AtomID}

type Ind1 = Bool;    type AtomID = Int;
type IndA = Bool;    type AtomType = Int;
type Lumo = Float;   type Charge = Float;
type LogP = Float;   type BondType = Int;

data Element = Br | C | Cl | F | H | I | N | O | S;
```
Strongly typed mutagenesis

Examples:

```python
mutagenic(True, False, -1.246, 4.23,
    {(1, 'C', 22, -0.117),
     (2, 'C', 22, -0.117),
     ...
     (26, 'O', 40, -0.388)},
    {({1, 2}, 7),
     ...
     ({24, 26}, 2)})
= True;
```

NB. *Naming* of sub-terms cannot be avoided here, because molecules are graphs rather than trees
Strongly typed mutagenesis

Hypothesis:

mutagenic(m) =

\[ \text{ind1P}(m) == \text{True} \quad || \quad \text{lumoP}(m) <= -2.072 \quad || \]

(exists \( a \rightarrow \) a 'in' \( \text{atomSetP}(m) \) && \( \text{elementP}(a)==\text{C} \) &&
\( \text{atomTypeP}(a)==26 \) && \( \text{chargeP}(a)==0.115 \) ||
(exists \( b1 \ b2 \rightarrow \) b1 'in' \( \text{bondSetP}(m) \) && b2 'in' \( \text{bondSetP}(m) \) &&
\( \text{bondTypeP}(b1)==1 \) && \( \text{bondTypeP}(b2)==2 \) &&
not disjoint(\( \text{atomIDSetP}(b1) \),\( \text{atomIDSetP}(b2) \)) ||
(exists \( a \rightarrow \) a 'in' \( \text{atomSetP}(m) \) &&
\( \text{elementP}(a)==\text{C} \) && \( \text{atomTypeP}(a)==29 \) &&
(exists \( b1 \ b2 \rightarrow \)
\( b1 \) 'in' \( \text{bondSetP}(m) \) && b2 'in' \( \text{bondSetP}(m) \) &&
\( \text{bondTypeP}(b1)==7 \) && \( \text{bondTypeP}(b2)==1 \) &&
\( \text{atomIDP}(a) \) 'in' \( \text{atomIDSetP}(b1) \) &&
not disjoint(\( \text{atomIDSetP}(b1) \),\( \text{atomIDSetP}(b2) \)))) ||

...;
Complexity of classification problems

- Simplest case: single table with primary key
  - *attribute-value* or *propositional* learning
  - example corresponds to tuple of constants

- Next: single table without primary key
  - *multi-instance* problem
  - example corresponds to set of tuples of constants

- Complexity resides in many-to-one foreign keys
  - *non-determinate* variables
  - lists, sets, multisets
Propositionalisation with LINUS

<table>
<thead>
<tr>
<th>Training examples</th>
<th>Background knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>daughter(sue,eve). (+)</td>
<td>parent(eve,sue). female(ann).</td>
</tr>
<tr>
<td>daughter(ann,pat). (+)</td>
<td>parent(ann,tom). female(sue).</td>
</tr>
<tr>
<td>daughter(tom,ann). (-)</td>
<td>parent(pat,ann). female(eve).</td>
</tr>
<tr>
<td>daughter(eve,ann). (-)</td>
<td>parent(tom,sue).</td>
</tr>
</tbody>
</table>

Transformation to attribute-value form:

<table>
<thead>
<tr>
<th>Class</th>
<th>Variables</th>
<th>Propositional features</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊕</td>
<td>sue</td>
<td>eve</td>
</tr>
<tr>
<td>⊕</td>
<td>ann</td>
<td>pat</td>
</tr>
<tr>
<td>⊖</td>
<td>tom</td>
<td>ann</td>
</tr>
<tr>
<td>⊖</td>
<td>eve</td>
<td>ann</td>
</tr>
</tbody>
</table>
Propositionalisation with LINUS

Result of attribute-value learning:

\[
\text{Class} = \oplus \quad \text{if} \ (\text{female}(X) = \text{true}) \land (\text{parent}(Y,X) = \text{true})
\]

Transformation to program clause form:

\[
\text{daughter}(X,Y) \leftarrow \text{female}(X), \text{parent}(Y,X)
\]

<table>
<thead>
<tr>
<th>Class</th>
<th>Variables</th>
<th>Propositional features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(X)</td>
<td>(Y)</td>
</tr>
<tr>
<td>\oplus</td>
<td>sue</td>
<td>eve</td>
</tr>
<tr>
<td>\oplus</td>
<td>ann</td>
<td>pat</td>
</tr>
<tr>
<td>\ominus</td>
<td>tom</td>
<td>ann</td>
</tr>
<tr>
<td>\ominus</td>
<td>eve</td>
<td>ann</td>
</tr>
</tbody>
</table>
DINUS: dealing with determinate variables

\[ \text{grandmother} = g, \quad \text{father} = f, \quad \text{mother} = m \]

\[ f(X,A) \quad X - \text{old, } A - \text{new} \quad \text{not determinate}! \]

\[ A - \text{old, } X - \text{new} \quad \text{determinate}! \]

<table>
<thead>
<tr>
<th>( g(X,Y) )</th>
<th>Variables</th>
<th>New variables</th>
<th>Propositional features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( X )</td>
<td>( Y )</td>
<td>( f(U, X) )</td>
</tr>
<tr>
<td>Class</td>
<td>( X )</td>
<td>( Y )</td>
<td>( U )</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>ann</td>
<td>bob</td>
<td>pat</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>ann</td>
<td>sue</td>
<td>pat</td>
</tr>
<tr>
<td>( \ominus )</td>
<td>bob</td>
<td>sue</td>
<td>tom</td>
</tr>
<tr>
<td>( \ominus )</td>
<td>tom</td>
<td>bob</td>
<td>zak</td>
</tr>
</tbody>
</table>
Traditional view of rule learning

- **Hypothesis construction**: find a set of $n$ rules
  - usually simplified by $n$ separate rule constructions

- **Rule construction**: find a pair (Head, Body)
  - e.g. select class and construct body

- **Body construction**: find a set of $m$ literals
  - usually simplified by adding one literal at a time
The role of feature construction

- **Hypothesis construction**: find a set of $n$ rules

- **Rule construction**: find a pair (Head, Body)

- **Body construction**: find a set of $m$ features

- **Feature construction**: find a set of $k$ literals
  - e.g. interesting subgroup, frequent itemset
  - discovery task rather than classification task
Features concern interactions of local variables

The following rule has one boolean feature ‘has a short closed car’:

\[ \text{eastbound}(T) : - \text{hasCar}(T,C), \]
\[ \text{clength}(C, \text{short}), \text{not croof}(C, \text{none}). \]

The following rule has two boolean features ‘has a short car’ and ‘has a closed car’:

\[ \text{eastbound}(T) : - \]
\[ \text{hasCar}(T,C1), \text{clength}(C1, \text{short}), \]
\[ \text{hasCar}(T,C2), \text{not croof}(C2, \text{none}). \]
Propositionalising rules

Equivalently:

\[
\text{eastbound}(T) : \neg \text{hasShortCar}(T), \text{hasClosedCar}(T).
\]
\[
\text{hasShortCar}(T) : \neg \text{hasCar}(T, C_1), \text{clength}(C_1, \text{short}).
\]
\[
\text{hasClosedCar}(T) : \neg \text{hasCar}(T, C_2), \neg \text{croof}(C_2, \text{none}).
\]

Given a way to construct and select first-order features, body construction in ILP is \textit{semi-propositional}:

- Head and body literals have the same global variable(s)
- Corresponds to single table, one row per example
Prolog feature bias in 1BC

- Flattened representation, but derived from strongly-typed term representation
  - one free global variable
  - each (binary) structural predicate introduces a new existential local variable and uses either global variable or local variable introduced by other structural predicate
  - utility predicates only use variables
  - all variables are used

- NB. features can be non-boolean
  - if all structural predicates are one-to-one
Example: mutagenesis with LINUS

42 regression-unfriendly molecules
57 first-order features with one utility literal
LINUS using CN2: 83% (leave-one-out)

mutagenic(M,false):-\neg \ (\text{has\_atom}(M,A),\text{atom\_type}(A,21)),
\neg \ \logP(M,L),L>1.99,L<5.64.
mutagenic(M,false):-\neg \ (\text{has\_atom}(M,A),\text{atom\_type}(A,195)),
\neg \ \logP(M,L),L>1.99,L<5.64.

mutagenic(M,false):-lumo(M,Lu),Lu>-0.77.
mutagenic(M,true):-lumo(M,Lu),Lu>-0.77.
mutagenic(M,true):-\neg \ \logP(M,L),L>6.36.

mutagenic(M,true):-\neg \ \logP(M,L),L>2.21.
Feature construction: summary

- All the expressiveness of RDM & ILP is in the features
  - body construction is essentially propositional
  - every RDM system does constructive induction

- Feature construction is a discovery task
  - scope for use of discovery systems such as Warmr, Tertius or Midos
  - alternative: exhaustive feature construction followed by feature selection using a relevancy filter
Concluding remarks

- Relational data mining: more data mining than logic program synthesis
  - Individual-centred representations
  - Choice of representation formalisms
  - Input format more important than output format
  - Data modelling – e.g. object-oriented data mining

- Joint work with Nicolas Lachiche, Nada Lavra, John Lloyd, and others
Further reading


- See also the ILPnet2 on-line library at http://www.cs.bris.ac.uk/~ILPnet2/Library/