The logic of learning: logic and knowledge representation in machine learning

Peter A. Flach
Department of Computer Science
University of Bristol
www.cs.bris.ac.uk/~flach/
Overview of this talk

- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook
Overview of this talk

- A (very) quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook
Inductive concept learning

- **Given:** descriptions of instances and non-instances
- **Find:** a concept covering all instances and no non-instances

![Diagram showing concept learning](image)

- too general (covering non-instances)
- too specific (not covering instances)
- not yet refuted = Version Space
Given:

- positive examples $P$: facts to be entailed,
- negative examples $N$: facts not to be entailed,
- background knowledge $B$: a set of predicate definitions;

Find: a hypothesis $H$ (one or more predicate definitions) such that

- for every $p \in P$: $B \cup H \models p$ (completeness),
- for every $n \in N$: $B \cup H \not\models n$ (consistency).
ILP methods

- **top-down** (language-driven)
  - descend the generality ordering
    - start with short, general rule
  - specialise by
    - substituting variables
    - adding conditions

- **bottom-up** (data-driven)
  - climb the generality ordering
    - start with long, specific rule
  - generalise by
    - introducing variables
    - removing conditions
### Top-down induction: example

<table>
<thead>
<tr>
<th>example</th>
<th>action</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+p(b, [b])</code></td>
<td>add clause</td>
<td><code>p(X, Y)</code></td>
</tr>
<tr>
<td><code>-p(x, [])</code></td>
<td>specialise</td>
<td>`p(X, [V</td>
</tr>
<tr>
<td><code>-p(x, [a,b])</code></td>
<td>specialise</td>
<td>`p(X, [X</td>
</tr>
<tr>
<td><code>+p(b, [a,b])</code></td>
<td>add clause</td>
<td>`p(X, [X</td>
</tr>
</tbody>
</table>

LICS'01 workshop

The logic of learning
Bottom-up induction: example

- Treat positive examples + ground background facts as body

- Choose two examples as heads and anti-unify

$q([1, 2], [3, 4], [1, 2, 3, 4]):- q([1, 2], [3, 4], [1, 2, 3, 4]), q([a], [], [a]), q([], [], []), q([2], [3, 4], [2, 3, 4])$

$q([a], [], [a]):- q([1, 2], [3, 4], [1, 2, 3, 4]), q([a], [], [a]), q([], [], []), q([2], [3, 4], [2, 3, 4])$

$q([A|B], C, [A|D]):- q([1, 2], [3, 4], [1, 2, 3, 4]), q([A|B], C, [A|D]), q(W, C, X), q([S|B], [3, 4], [S, T, U|V]), q([R|G], K, [R|L]), q([a], [], [a]), q(Q, [], Q), q([P|K], K, [P|K]), q(N, K, O), q(M, [], M), q([], [], []), q(G, K, L), q([F|G], [3, 4], [F, H, I|J]), q([E|C], [E|C]), q(B, C, D), q([2], [3, 4], [2, 3, 4])$

- Generalise by removing literals until negative examples covered
A molecular compound is carcinogenic if:

1. it tests positive in the Salmonella assay; or
2. it tests positive for sex-linked recessive lethal mutation in Drosophila; or
3. it tests negative for chromosome aberration; or
4. it has a carbon in a six-membered aromatic ring with a partial charge of -0.13; or
5. it has a primary amine group and no secondary or tertiary amines; or
6. it has an aromatic (or resonant) hydrogen with partial charge ≥ 0.168; or
7. it has an hydroxy oxygen with a partial charge ≥ -0.616 and an aromatic (or resonant) hydrogen; or
8. it has a bromine; or
9. it has a tetrahedral carbon with a partial charge ≤ -0.144 and tests positive on Progol’s mutagenicity rules.
**ILP example: East-West trains**

1. TRAINS GOING EAST

1. 

2. 

3. 

4. 

5. 

2. TRAINS GOING WEST

1. 

2. 

3. 

4. 

5.
Example:

```
eastbound(t1).
```

Background knowledge:

```
car(t1,c1).
rectangle(c1).
short(c1).
open(c1).
two_wheels(c1).
load(c1,l1).
circle(l1).
one_load(l1).
```

```
car(t1,c2).
rectangle(c2).
short(c2).
open(c2).
three_wheels(c2).
load(c2,l2).
hexagon(l2).
one_load(l2).
```

```
car(t1,c3).
rectangle(c3).
short(c3).
open(c3).
two_wheels(c3).
load(c3,l3).
triangle(l3).
one_load(l3).
```

```
car(t1,c4).
rectangle(c4).
short(c4).
open(c4).
two_wheels(c4).
load(c4,l4).
rectangle(l4).
one_load(l4).
```

Hypothesis:

```
eastbound(T):-car(T,C),short(C),not open(C).
```
**Prolog representation (flattened)**

- **Example:**
  
  
  ```prolog
eastbound(t1).
```

- **Background knowledge:**
  
  ```prolog
car(t1,c1).
rectangle(c1).
short(c1).
oopen(c1).
two_wheels(c1).
load(c1,11).
circle(11).
one_load(11).
car(t1,c2).
rectangle(c2).
long(c2).
oopen(c2).
three_wheels(c2).
load(c2,12).
hexagon(12).
one_load(12).
car(t1,c3).
rectangle(c3).
short(c3).
peaked(c3).
two_wheels(c3).
load(c3,13).
triangle(13).
one_load(13).
car(t1,c4).
rectangle(c4).
long(c4).
oopen(c4).
two_wheels(c4).
load(c4,14).
rectangle(14).
```

- **Hypothesis:**
  
  ```prolog
eastbound(T) :- car(T,C), short(C), not open(C).
```
Prolog representation (terms)

Example:

\[
eastbound([c(\text{rectangle}, \text{short}, \text{open}, 2, l(\text{circle}, 1))),
\text{c}(\text{rectangle}, \text{long}, \text{open}, 3, l(\text{hexagon}, 1))),
\text{c}(\text{rectangle}, \text{short}, \text{peaked}, 2, l(\text{triangle}, 1))),
c(\text{rectangle}, \text{long}, \text{open}, 2, l(\text{rectangle}, 3))]).
\]

Background knowledge: \text{member/2, arg/3}

Hypothesis:

\[
eastbound(T) :\neg \text{member}(C, T), \text{arg}(2, C, \text{short}),
\neg \text{arg}(3, C, \text{open}).
\]
Prolog representation (terms)

Example:

```prolog
eastbound([c(rectangle, short, open, 2, l(circle, 1)),
c(rectangle, long, open, 3, l(hexagon, 1)),
c(rectangle, short, peaked, 2, l(triangle, 1)),
c(rectangle, long, open, 2, l(rectangle, 3))]).
```

Background knowledge: member/2, arg/3

Hypothesis:

```prolog
eastbound(T) :- member(C, T), arg(2, C, short),
                not arg(3, C, open).
```

LICS'01 workshop
Machine learning vs. ILP

- attribute-value concept learning
- multi-instance learning
- individual-centred representations
- Prolog program synthesis

LICS'01 workshop
The logic of learning
Overview of this talk

- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook
Knowledge Representation

- Entity-Relationship (ER) diagrams
- Relational Database
- Individual-Centred Representations
- Strongly typed language
- XML?
ER diagram for East-West trains

- **Train**
  - **Direction**
  - **Has**
    - **Car**
      - **Shape**
      - **Length**
      - **Roof**
      - **Wheels**
      - **Has**
        - **Load**
          - **Number**
          - **Object**

1-1 relationship:
- Train has 1 Car.
- Car has 1 Has Load.
- Load has 1 Number.
- Load has 1 Object.
A particular train

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

LICS'01 workshop  The logic of learning
Database representation

**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>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>

**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>open</td>
<td>2</td>
</tr>
<tr>
<td>c2</td>
<td>t1</td>
<td>rectangle</td>
<td>long</td>
<td>open</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>open</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<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 = 'rectangle' AND
CAR_TABLE.ROOF != 'open'

LICS’01 workshop
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 into sub-databases each describing a single individual

- Alternative: all information about a single individual packed together in a term
  - 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
Type signature:

data Shape  = Rectangle | Hexagon | …;  data Length = Long | Short;
data Roof   = Open | Peaked | …;  data Object = Circle | Hexagon | …;
type Wheels = Int;  type Load = (Object,Number);  type Number = Int
type Car    = (Shape,Length,Roof,Wheels,Load);  type Train = [Car];
eastbound::Train->Bool;

Example:
eastbound([(Rectangle,Short,Open,2,(Circle,1)), (Rectangle,Long,Open,3,(Hexagon,1)), (Rectangle,Short,Peaked,2,(Triangle,1)), (Rectangle,Long,Open,2,(Rectangle,3))]) = True

Hypothesis:
eastbound(t) = (exists \c -> member(c,t) &&
                LengthP(c)==Short && RoofP(c)!=Open)
Type signature:

data Shape = Rectangle | Hexagon | …; data Length = Long | Short;
data Roof = Open | Peaked | …; data Object = Circle | Hexagon | …;
type Wheels = Int; type Load = (Object,Number); type Number = Int
type Car = (Shape,Length,Roof,Wheels,Load); type Train = [Car];
eastbound::Train->Bool;

Example:
eastbound([(Rectangle, Short, Open, 2, (Circle, 1)),
(Rectangle, Long, Open, 3, (Hexagon, 1)),
(Rectangle, Short, Peaked, 2, (Triangle, 1)),
(Rectangle, Long, Open, 2, (Rectangle, 3))]) = True

Hypothesis:
eastbound(t) = (exists \(c \rightarrow\) member(c,t) &&
\(\text{LengthP}(c) == \text{Short} \&\& \text{RoofP}(c) != \text{Open}\))
Type signature:

```haskell
data Element = Br | C | Cl | F | H | I | N | O | S;

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

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

mutagenic :: Molecule -> Bool;
```
Examples:

\[
\text{mutagenic}(\text{True}, \text{False}, -1.246, 4.23, \\
\{(1, \text{C}, 22, -0.117), \\
(2, \text{C}, 22, -0.117), \\
..., \\
(26, \text{O}, 40, -0.388)\}, \\
\{\{1, 2\}, 7\}, \\
..., \\
\{\{24, 26\}, 2\})
\]

= True;

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

\[ \text{mutagenic}(m) = \]
\[ \text{ind1P}(m) == \text{True} \bigg| \bigg| \text{lumoP}(m) \leq -2.072 \bigg| \bigg| \]
\[ (\exists a \rightarrow a \text{ 'in' } \text{atomSetP}(m) \land \text{elementP}(a) == \text{C} \land \text{atomTypeP}(a) == 26 \land \text{chargeP}(a) == 0.115) \bigg| \bigg| \]
\[ (\exists b1 \ b2 \rightarrow b1 \text{ 'in' } \text{bondSetP}(m) \land b2 \text{ 'in' } \text{bondSetP}(m) \land \text{bondTypeP}(b1) == 1 \land \text{bondTypeP}(b2) == 2 \land \neg \text{disjoint}(\text{labelSetP}(b1), \text{labelSetP}(b2)) \bigg| \bigg| \]
\[ (\exists a \rightarrow a \text{ 'in' } \text{atomSetP}(m) \land \text{elementP}(a) == \text{C} \land \text{atomTypeP}(a) == 29 \land \]
\[ (\exists b1 \ b2 \rightarrow \]
\[ b1 \text{ 'in' } \text{bondSetP}(m) \land b2 \text{ 'in' } \text{bondSetP}(m) \land \text{bondTypeP}(b1) == 7 \land \text{bondTypeP}(b2) == 1 \land \text{labelP}(a) \text{ 'in' } \text{labelSetP}(b1) \land \]
\[ \neg \text{disjoint}(\text{labelSetP}(b1), \text{labelSetP}(b2))) \bigg| \bigg| \]
\[ \ldots; \]
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
Back to Prolog: what do we learn from all this?

- Structural predicates introduce local variables, utility predicates consume them.
- Interactions between local variables should not be broken up ===> features.
- Enhancement of existing transformation methods (e.g., LINUS) through feature construction.
**The key steps in 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 key steps in 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 $m$ features
  - usually simplified by adding one feature at a time

- **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 feature ‘has a short closed car’:

\[
\text{eastbound}(T) : \neg \text{car}(T,C), \text{short}(C), \neg \text{open}(C).
\]

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

\[
\text{eastbound}(T) : \neg \begin{align*}
\text{car}(T,C1), & \text{short}(C1), \\
\text{car}(T,C2), & \neg \text{open}(C2).
\end{align*}
\]
Propositionalising rules

Equivalently:

\[
\text{eastbound}(T) :\neg \text{hasShortCar}(T), \text{hasClosedCar}(T).
\]

\[
\text{hasShortCar}(T) :\neg \text{car}(T, C1), \text{short}(C1).
\]

\[
\text{hasClosedCar}(T) :\neg \text{car}(T, C2), \neg \text{open}(C2).
\]

Given a way to construct and select first-order features, body construction in ILP is semi-propositional

- head and all literals in body have the same global variable(s)
- corresponds to single table, one row per example
Prolog feature bias

- 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

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

```
mutagenic(M,false):-not(has_atom(M,A),atom_type(A,21)),
    logP(M,L),L>1.99,L<5.64.
mutagenic(M,false):-not(has_atom(M,A),atom_type(A,195)),
    logP(M,L),L>1.81.
mutagenic(M,false):-logP(M,L),L>5.64,L<6.36.
mutagenic(M,false):-lumo(M,Lu),Lu>-0.77.
mutagenic(M,true):-has_atom(M,A),atom_type(A,21),
    logP(M,L),L>5.64,L<6.36.
mutagenic(M,true):-lumo(M,Lu),Lu>-0.95,
    logP(M,L),L<2.21.
```
Feature construction: summary

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

- Feature construction is a discovery task
  - use of discovery systems such as Warmr, Tertius or Midos
  - alternative: use a relevancy filter
Overview of this talk

- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook
I write $E |< H$ for ‘$H$ is a possible inductive hypothesis given evidence $E$’

- like deduction: from input to output
- unlike deduction: possibly unsound

What are sensible properties of $|< \,$?

What are possible material definitions of $|< \,$?
General induction postulates

(I1) If $\alpha |< \beta$ and $\alpha \land \beta \rightarrow \gamma$, then $\alpha \land \gamma |< \beta$.

(I2) If $\alpha |< \beta$ and $\alpha \land \beta \rightarrow \gamma$, then $\alpha \land \neg \gamma |\not< \beta$.

(\(I2'\)) If $\beta \rightarrow \neg \alpha$, then $\alpha |\not< \beta$.

(I3) If $\alpha |< \beta$ and $\alpha \land \beta \rightarrow \gamma$, then $\alpha |< \beta \land \gamma$.

(I4) If $\alpha |< \beta$, then $\alpha |< \alpha$.

(I5) If $\alpha |< \beta$, then $\beta |< \beta$.

(I6) If $\alpha |< \beta$ and $\beta \leftrightarrow \gamma$, then $\alpha |< \gamma$.

(I7) If $\alpha |< \gamma$ and $\alpha \leftrightarrow \beta$, then $\beta |< \gamma$. 
Explanatory induction

- \( E \triangleright H \) is interpreted as ‘evidence \( E \) is explained by hypothesis \( H \)’
  - induction as reverse deduction

- Close link with abduction
  - Peirce: ‘if \( A \) were true, \( C \) would be a matter of course’

- Depends on notion of explanation
(E1) If $\alpha \mid< \beta, \neg\gamma \rightarrow \beta$ and $\gamma \mid< \gamma$, then $\alpha \mid< \gamma$.

(E2) If $\gamma \mid< \gamma$ and $\neg\alpha \mid< \gamma$, then $\alpha \mid< \alpha$.

(E3) If $\alpha \mid< \beta \wedge \gamma$, then $\beta \rightarrow \alpha \mid< \gamma$.

(E4) If $\alpha \mid< \gamma$ and $\beta \mid< \gamma$, then $\alpha \wedge \beta \mid< \gamma$.

(E5) If $\alpha \mid< \gamma$ and $\neg\neg\alpha \rightarrow \beta$, then $\beta \mid< \gamma$. 

Explanatory induction postulates
Let $|\sim$ be an explanation mechanism, and define the explanatory power of a formula $\alpha$ as $C_{\sim} = \{ \gamma \mid \alpha \mid \sim \gamma \}$.

The explanatory consequence relation $\langle$ based on $|\sim$ is defined as

$$\alpha \langle \beta \quad \text{iff} \quad C_{\sim}(\alpha) \subseteq C_{\sim}(\beta) \subseteq L$$

(E1–5) are sound and complete if $|\sim = |=$.
Confirmatory induction

- $E |< H$ is interpreted as ‘evidence $E$ confirms hypothesis $H$’

- A kind of closed-world reasoning
  - ‘assume that everything you haven’t seen behaves like something you have seen’
  - closely related to non-monotonic reasoning
Confirmatory induction postulates

(C1) If $\alpha \prec \beta$ and $\beta \rightarrow \gamma$, then $\alpha \prec \gamma$.

(C2) If $\alpha \prec \alpha$ and $\alpha \not\prec \neg \beta$, then $\beta \prec \beta$.

(C3) If $\alpha \prec \beta$ and $\alpha \prec \gamma$, then $\alpha \prec \beta \land \gamma$.

(C4) If $\alpha \prec \gamma$ and $\beta \prec \gamma$, then $\alpha \lor \beta \prec \gamma$.

(C5) If $\alpha \prec \beta$ and $\alpha \prec \gamma$, then $\alpha \land \gamma \prec \beta$. 
Let \( \text{Reg} \) be a function constructing a set of regular models from observations.

The confirmatory consequence relation \( \prec \) based on \( \text{Reg} \) is defined as:

\[ \alpha \prec \beta \iff \emptyset \subset \text{Reg}(\alpha) \subseteq \beta \]

(C1–5) are sound and complete if \( \text{Reg}(\alpha) \) are the most preferred models of \( \alpha \).
Overview of this talk

- A quick overview of ILP
- Knowledge representation
  - individual-centred representations
- Learning as inference
  - inductive consequence relations
- Conclusions and outlook
First-order representations in...

- ...probabilistic models
  - Koller’s probabilistic relational models
  - first-order Bayesian classification with 1BC
  - towards first-order Bayesian networks

- ...support vector machines
  - kernels on sequences
  - a kernel on Escher terms

- ...neural networks
  - recurrent NN for Escher terms
The naive Bayes classifier

Bayesian classifier:

\[
\arg \max_c P(c \mid d) = \arg \max_c \frac{P(d \mid c)P(c)}{P(d)} = \arg \max_c P(d \mid c)P(c)
\]

Naive Bayes assumption (propositional case):

\[
= \arg \max_c P(c) \prod_i P(A_i = a_i \mid c)
\]
Naive Bayes net

Class

Individual

A1  A2  A3

Class

A1  A2  A3
Towards first-order Bayes nets

Class

Molecule

AtomSet

Atom

Lumo

LogP

Has

Contains

Element

AtomType

Charge

Class

LUMO

LogP
Support vector machines

- Wide margin classifier
  - support vectors are the datapoints closest to the separating hyperplane

- Kernel: (implicit) transformation to feature space
  - to deal with problems that are not linearly separable in input space
  - feature space is often high-dimensional
Linear classifiers construct a hyperplane separating the input points.

**Decision rule**

\[ h(x) = \text{sgn}(\langle w \cdot x \rangle + b) \]

**Hypothesis**

\[ w = \sum_{i} \alpha_i y_i x_i \]

**Equivalently**

\[ h(x) = \text{sgn}\left(\sum_{i} \alpha_i y_i \langle x_i \cdot x \rangle + b\right) \]

where \( \alpha_i \) represent hypothesis in dual co-ordinates.
Learning in feature space:

\[ h(x) = \text{sgn} \left( \sum_i \alpha_i y_i \langle \phi(x_i) \cdot \phi(x) \rangle + b \right) \]

A kernel calculates the inner product directly in input space:

\[ K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle \]

This measures the similarity between \( x \) and \( z \) in terms of features \( \phi \)
Let $x$ and $z$ be terms of type $T$. We define $K_T(x,z)$ recursively as follows:

- If $T = T_1 \times \ldots \times T_n$ is a tuple type, $x = (x_1,\ldots,x_n)$ and $z = (z_1,\ldots,z_n)$, then $K_T(x,z) = K_{T_1}(x_1,z_1) + \ldots + K_{T_n}(x_n,z_n)$.

- If $T = \{T'\}$ is a set type, $x = \{x_1,\ldots,x_n\}$ and $z = \{z_1,\ldots,z_m\}$, then $K_T(x,z) = K_{T'}(x_1,z_1) + \ldots + K_{T'}(x_1,z_m) + K_{T'}(x_2,z_1) + \ldots + K_{T'}(x_2,z_m) + \ldots + K_{T'}(x_n,z_m)$.

- If $x = f(x_1,\ldots,x_n)$ and $z = f(z_1,\ldots,z_n)$ where $f$ is a data constructor of type $T_1 \to \ldots \to T_n \to T$, then $K_T(x,z) = 1 + K_{T_1}(x_1,z_1) + \ldots + K_{T_n}(x_n,z_n)$; if $x$ and $z$ have different data constructors then $K_T(x,z) = 0$. 
Recurrent neural networks

- Consist of a recurrent or folding part that is unfolded to encode a given input tree, followed by a traditional feed-forward network.
- Folding part trained by backpropagation through structure.
- Generalises naturally to terms.
Recurrent NN for Escher terms

T x List Int
T' x T' -> T' x T' -> T
Int -> List Int -> List Int
T' x T'

f((a,b),(c,d))

[(f((a,b),(c,d)), [4,21,42])

LICS'01 workshop
The logic of learning
Concluding remarks

- Data models and knowledge representation are integral parts of any approach to learning, modelling and reasoning.

- Individual-centred representation are natural in classification and provide better understanding of the relation with propositional approaches.

- There is still much to explore in upgrading existing propositional approaches with richer knowledge representation.
Acknowledgements

Joint work with

- Nicolas Lachiche
- John Lloyd
- Christophe Giraud-Carrier
- Nada Lavrac
- Thomas Gaertner
- Elias Gyftodimos