Aims

This lecture will introduce you to theoretical and applied aspects of representing hypotheses for machine learning in first-order logic. Following it you should be able to:

- outline the key differences between propositional and first-order learning
- describe the problem of learning relations and some applications
- outline the problem of induction in terms of inverse deduction
- describe inverse resolution in propositional and first-order logic
- describe least general generalisation and $\theta$-subsumption
- reproduce the basic FOIL algorithm and its use of information gain

[Recommended reading: Mitchell, Chapter 10]
[Recommended exercises: 10.5 – 10.7 (10.8)]

Relevant programs

- Progol
  http://www.doc.ic.ac.uk/~shm/progol.html
- Aleph
  http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph
- FOIL
  http://www.rulequest.com/Personal/
- iProlog
  http://www.doc.ic.ac.uk/~shm/iprolog.html
- Golem
  http://www.doc.ic.ac.uk/~shm/golem.html

See also:
http://www-ai.ijs.si/~ilpnet2/systems/
**Representation in Propositional Logic**

Propositional variables: $P, Q, R, \ldots$

Negation: $\neg S, \neg T, \ldots$

Logical connectives: $\land, \lor, \leftarrow, \leftrightarrow$

Well-formed formulae: $P \lor Q, (\neg R \land S) \rightarrow T,$ etc.

Inference rules:

- **modus ponens**: Given $B$ and $A \leftarrow B$ infer $A$
- **modus tollens**: Given $\neg A$ and $A \leftarrow B$ infer $\neg B$

Enable **sound** or **valid** inference.

**Meaning in Propositional Logic**

Propositional variables stand for declarative sentences (properties):

- $P$ the paper is red
- $Q$ the solution is acid

Potentially useful inferences:

- $P \rightarrow Q$ If the paper is red then the solution is acid

Meaning of such formulae can be understood with a **truth table**:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P \rightarrow Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

**Representation in First-Order Predicate Logic**

We have a richer language for developing formulae:

- constant symbols: Fred, Jane, Copper, Manganese, …
- function symbols: Cons, Succ, …
- variable symbols: $x, y, z, \ldots$
- predicate symbols: Parent, Likes, Binds, …

We still have:

- Negation: $\neg \text{Likes}(\text{Bob}, \text{Footy}), \ldots$
- Logical connectives: $\land, \lor, \leftarrow, \leftrightarrow$

but we also have quantification:

- $\forall x \text{Likes}(x, \text{Fred})$
- $\exists y \text{Binds}(\text{Copper}, y)$

And we still have well-formed formulae and inference rules …

**Meaning in First-Order Logic**

Same basic idea as propositional logic, but more complicated.

Give meaning to first-order logic formulae by **interpretation** with respect to a given **domain** $D$ by associating:

- each constant symbol with some **element** of $D$
- each $n$-ary function symbol with some **function** from $D^n$ to $D$
- each $n$-ary predicate symbol with some **relation** in $D^n$

For variables, essentially consider associating all or some domain elements in the formula, depending on quantification.

**Interpretation is association of a formula with a truth-valued statement about the domain.**
Learning First Order Rules

Why do that?

- trees, rules so far have allowed only comparisons of a variable with a constant value (e.g., sky = sunny, temperature < 45)
- these are propositional representations – have same expressive power as propositional logic
- to express more powerful concepts, say involving relationships between example objects, propositional representations are insufficient, and we need a more expressive representation

E.g., to classify X depending on it’s relation R to another object Y

BUT in first order logic sets of rules can represent graph concepts such as:

\[
\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y) \\
\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, z) \land \text{Ancestor}(z, y)
\]

The declarative programming language Prolog is based on the Horn clause subset of first-order logic – a form of Logic Programming:

- Prolog is a general purpose programming language: logic programs are sets of first order rules
- “pure” Prolog is Turing complete, i.e., can simulate a Universal Turing machine (every computable function)
- learning in this representation is called Inductive Logic Programming (ILP)

PROLOG definitions for relational concepts

Some Prolog syntax:

- all predicate and constant names begin with a lower-case letter
  - predicate (relation) names, e.g. uncle, adjacent
  - constant names, e.g. fred, banana
- all variable names begin with an upper-case letter
  - X, Y, Head, Tail
- a predicate is specified by its name and arity (number of arguments), e.g.
  - male/1 means the predicate “male” with one argument
  - sister/2 means the predicate “sister of” with two arguments
Prolog definitions for relational concepts

- predicates are defined by sets of clauses, each with that predicate in its head
  - e.g. the recursive definition of ancestor/2
    
    ```prolog
    ancestor(X, Y) :- parent(X, Y).
    ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
    ```

- clause head, e.g. ancestor/2, is to the left of the `:-`

- clause body, e.g. parent(X, Z), ancestor(Z, Y), is to the right of the `:-`

- each instance of a relation name in a clause is called a literal
- a definite clause has exactly one literal in the clause head
- a Horn clause has at most one literal in the clause head

Prolog programs are sets of Horn clauses

Prolog is a form of logic programming (many approaches)

related to SQL, functional programming, ...

Induction as Inverted Deduction

Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition or subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation; ... it must be allowed that inductive investigations are of a far higher degree of difficulty and complexity than any questions of deduction... (W.S. Jevons, 1874)

A photograph of the Logic Piano invented by William Stanley Jevons. Photograph taken at the Sydney Powerhouse Museum on March 5, 2006. This item is part of the collection of the Museum of the History of Science, Oxford and was on loan to the Powerhouse Museum.  
Induction as Inverted Deduction

[From lecture on Concept Learning:] Induction is finding \( h \) such that

\[
(\forall(x_i, f(x_i)) \in D) \; B \land h \land x_i \vdash f(x_i)
\]

where

- \( x_i \) is \( i \)th training instance
- \( f(x_i) \) is the target function value for \( x_i \)
- \( B \) is other background knowledge

So let’s design inductive algorithm by inverting operators for automated deduction!

We have mechanical deductive operators \( F(A, B) = C \), where \( A \land B \models C \)

need inductive operators

\[
O(B, D) = h \text{ where } (\forall(x_i, f(x_i)) \in D) \; (B \land h \land x_i) \vdash f(x_i)
\]

Positives:

- Subsumes earlier idea of finding \( h \) that “fits” training data
- Domain theory \( B \) helps define meaning of “fit” the data

\[
B \land h \land x_i \vdash f(x_i)
\]

- Suggests algorithms that search \( H \) guided by \( B \)
Induction as Inverted Deduction

Negatives:

- Doesn’t allow for noisy data. Consider

\[(\forall x_i, f(x_i) \in D) \ (B \land h \land x_i) \models f(x_i)\]

- First order logic gives a huge hypothesis space \(H\)

\[\rightarrow\text{overfitting...}\]

\[\rightarrow\text{intractability of calculating all acceptable } h\text{'s}\]

Deduction: Resolution Rule

\[
\frac{P \lor L}{\neg L \lor R} \quad \frac{P}{\neg L \lor R}
\]

1. Given initial clauses \(C_1\) and \(C_2\), find a literal \(L\) that occurs in clause \(C_1\), but not in clause \(C\).

2. Form the resolvent \(C\) by including all literals from \(C_1\) and \(C_2\), except for \(L\) and \(-L\). More precisely, the set of literals occurring in the conclusion \(C\) is

\[C = (C_1 - \{L\}) \cup (C_2 - \{-L\})\]

where \(\cup\) denotes set union, and \(\neg\) denotes set difference.

Inverting Resolution

\[C_2 : \text{KnowMaterial} \lor \neg \text{Study}\]

\[C_1 : \text{PassExam} \land \text{KnowMaterial}\]

\[C : \text{PassExam} \land \neg \text{Study}\]

Inverting Resolution (Propositional)

1. Given initial clauses \(C_1\) and \(C\), find a literal \(L\) that occurs in clause \(C_1\), but not in clause \(C\).

2. Form the second clause \(C_2\) by including the following literals

\[C_2 = (C - (C_1 - \{L\})) \cup \{-L\}\]

3. Given initial clauses \(C_2\) and \(C\), find a literal \(-L\) that occurs in clause \(C_2\), but not in clause \(C\).

4. Form the second clause \(C_1\) by including the following literals

\[C_1 = (C - (C_2 - \{-L\}) \cup \{L\}\right)
Duce operators

<table>
<thead>
<tr>
<th>Op</th>
<th>Same Head</th>
<th>Different Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Identification</td>
<td>Absorption</td>
</tr>
<tr>
<td></td>
<td>$p \leftarrow A, B$</td>
<td>$p \leftarrow q, B$</td>
</tr>
<tr>
<td></td>
<td>$p \leftarrow A, q$</td>
<td>$q \leftarrow A$</td>
</tr>
<tr>
<td>W</td>
<td>Intra-construction</td>
<td>Inter-construction</td>
</tr>
<tr>
<td></td>
<td>$p \leftarrow A, B_1$</td>
<td>$p_1 \leftarrow w, B_1$</td>
</tr>
<tr>
<td></td>
<td>$w \leftarrow B_1$</td>
<td>$p_2 \leftarrow w, B_2$</td>
</tr>
<tr>
<td></td>
<td>$p \leftarrow A, w$</td>
<td>$w \leftarrow A$</td>
</tr>
</tbody>
</table>

Each operator is read as: pre-conditions on left, post-conditions on right.

First order resolution

First order resolution:

1. Find a literal $L_1$ from clause $C_1$, literal $L_2$ from clause $C_2$, and substitution $\theta$ such that $L_1\theta = \neg L_2\theta$

2. Form the resolvent $C$ by including all literals from $C_1\theta$ and $C_2\theta$, except for $L_1\theta$ and $\neg L_2\theta$. More precisely, the set of literals occurring in the conclusion $C$ is

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

Inverting First order resolution

Factor $\theta$

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2$$

$C_2$ should have no common literals with $C_1$

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2$$

By definition of resolution $L_2 = \neg L_1\theta_1\theta_2^{-1}$

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\}$$

Cigol

- **Father** (Tom, Bob)
- **GrandChild** (Bob, Shannon)
- **Father** (Shannon, Tom)
- **GrandChild** (Bob, Tom)

Graphical representation of Cigol relationships:
Subsumption and Generality

$\theta$-subsumption \( C \) $\theta$-subsumes \( D \) if there is a substitution \( \theta \) such that \( C\theta \subseteq D \).

\( C \) is at least as general as \( D \) \( (C \leq D) \) if \( C \) $\theta$-subsumes \( D \).

If \( C \) $\theta$-subsumes \( D \) then \( C \) logically entails \( D \) (but not the reverse).

$\theta$-subsumption is a partial order, thus generates a lattice in which any two clauses have a least-upper-bound and a greatest-lower-bound.

The least general generalisation (LGG) of two clauses is their least-upper-bound in the $\theta$-subsumption lattice.

LGG


- LGG of clauses is based on LGGs of literals (atoms)
- Lgg of literals is based on LGGs of terms, i.e. constants and variables
- LGG of two constants is a variable, i.e. a minimal generalisation

LGG of atoms

Two atoms are compatible if they have the same predicate symbol and arity (number of arguments)

- Lgg\((a, b)\) for different constants or functions with different function symbols is the variable \( X \)
- Lgg\((f(a_1, ..., a_n), f(b_1, ..., b_n))\) is \( f(\text{Lgg}(a_1, b_1), ..., \text{Lgg}(a_n, b_n)) \)
- Lgg\((Y_1, Y_2)\) for variables \( Y_1, Y_2 \) is the variable \( X \)

Note:

1. must ensure that the same variable appears everywhere its bound arguments do in the atom
2. must ensure introduced variables appear nowhere in the original atoms

LGG of clauses

The LGG of two clauses \( C_1 \) and \( C_2 \) is formed by taking the LGGs of each literal in \( C_1 \) with every literal in \( C_2 \).

Clauses form a subsumption lattice, with LGG as least upper bound and MGI (most general instance) as lower bound.

Lifts the concept learning lattice to a first-order logic representation.

Leads to relative LGGs with respect to background knowledge.
Subsumption lattice

RLGG - LGG relative to background knowledge

Example from Quinlan (1991)

Given two ground instances of target predicate \( Q/k, Q(c_1, c_2, \ldots, c_k) \) and \( Q(d_1, d_2, \ldots, d_k) \), plus other logical relations representing background knowledge that may be relevant to the target concept, the relative least general generalisation (rlgg) of these two instances is:

\[
Q(\text{lgg}(c_1, d_1), \text{lgg}(c_2, d_2), \ldots) \leftarrow \bigwedge \{\text{lgg}(r_1, r_2)\}
\]

for every pair \( r_1, r_2 \) of ground instances from each relation in the background knowledge.

RLGG Example

This figure depicts two scenes \( s_1 \) and \( s_2 \) and may be described by the predicates Scene/1, On/3, Left-of/2, Circle/1, Square/1 and Triangle/1.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Ground Instances (tuples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene</td>
<td>{&lt; s_1, a, b &gt;, &lt; s_2, f, e &gt;}</td>
</tr>
<tr>
<td>On</td>
<td>{&lt; s_1, b, c &gt;, &lt; s_2, d, e &gt;}</td>
</tr>
<tr>
<td>Left-of</td>
<td>{&lt; a &gt;, &lt; f &gt;}</td>
</tr>
<tr>
<td>Circle</td>
<td>{&lt; b &gt;, &lt; d &gt;}</td>
</tr>
<tr>
<td>Square</td>
<td>{&lt; c &gt;, &lt; e &gt;}</td>
</tr>
<tr>
<td>Triangle</td>
<td>{&lt; s_1 &gt;}</td>
</tr>
</tbody>
</table>
To compute RLGG of the two scenes generate the clause:

\[
\text{Scene}(\text{lgg}(s_1, s_2)) \leftarrow \\
\text{On}(\text{lgg}(s_1, s_2), \text{lgg}(a, f), \text{lgg}(b, c)), \\
\text{Left-of}(\text{lgg}(s_1, s_2), \text{lgg}(b, d), \text{lgg}(c, e)), \\
\text{Circle}(\text{lgg}(a, f)), \\
\text{Square}(\text{lgg}(b, d)), \\
\text{Triangle}(\text{lgg}(c, e))
\]

Compute LGGs to introduce variables into the final clause:

\[
\text{Scene}(A) \leftarrow \\
\text{On}(A, B, C), \\
\text{Left-of}(A, D, E), \\
\text{Circle}(D), \\
\text{Square}(D), \\
\text{Triangle}(E)
\]

### Refinement Operators

Propositional subsumption — clauses are sets of literals.

E.g., \(\text{flies} \leftarrow \text{bird}, \text{normal}\) can be represented as the set \{\text{flies}, \neg \text{bird}, \neg \text{normal}\}.

In a propositional representation, one clause is more general than the other if it contains a subset of its literals.

For first-order atoms, one atom \(a_1\) is more general than another \(a_2\) if there is a substitution \(\theta\) such that \(a_1\theta \subseteq a_2\).

A refinement operator takes one atom (clause) and produces another such that the first atom subsumes the second.

For first-order atoms, ideal refinement operators can be found (see tutorial notes).

### From Propositional to First-order Representations

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
From Propositional to First-order Representations

Use a single relation (the target relation):

\[
\text{play_tennis}(\text{Day}, \text{Outlook}, \text{Temperature}, \text{Humidity}, \text{Wind}, \text{PlayTennis}).
\]

Training data:

\[
\text{play_tennis}(d1, \text{sunny}, \text{hot}, \text{high}, \text{weak}, \text{no}).
\]

... 

Hypothesis (complete and correct for examples):

\[
\text{play_tennis}(\text{Day}, \text{overcast}, \text{Temperature}, \text{Humidity}, \text{Wind}, \text{yes}).
\]

\[
\text{play_tennis}(\text{Day}, \text{rain}, \text{Temperature}, \text{Humidity}, \text{weak}, \text{yes}).
\]

\[
\text{play_tennis}(\text{Day}, \text{sunny}, \text{Temperature}, \text{normal}, \text{Wind}, \text{yes}).
\]

From Propositional to First-order Representations

Multiple relations define the target w.r.t. background knowledge:

\[
\text{play_tennis}(\text{Day}, \text{PlayTennis}).
\]

Training data:

\[
\text{play_tennis}(d1, \text{no}).
\]

\[
\text{outlook}(d1, \text{sunny}). \quad \text{temperature}(d1, \text{hot}).
\]

\[
\text{humidity}(d1, \text{high}). \quad \text{wind}(d1, \text{weak}).
\]

... 

Hypothesis (complete and correct for examples):

\[
\text{play_tennis}(\text{Day}, \text{yes}) \leftarrow \text{outlook}(\text{Day}, \text{overcast}).
\]

\[
\text{play_tennis}(\text{Day}, \text{yes}) \leftarrow \text{outlook}(\text{Day}, \text{rain}), \text{wind}(\text{Day}, \text{weak}).
\]

\[
\text{play_tennis}(\text{Day}, \text{yes}) \leftarrow \text{outlook}(\text{Day}, \text{sunny}), \text{humidity}(\text{Day}, \text{normal}).
\]

Michalski's Trains

Declare types:

\[
\text{train}(\text{east1}). \quad \text{train}(\text{east2}). \quad \text{train}(\text{east3}). \ldots
\]

\[
\text{train}(\text{west6}). \quad \text{train}(\text{west7}). \quad \text{train}(\text{west8}). \ldots
\]

\[
\text{car}(\text{car}_{11}). \quad \text{car}(\text{car}_{12}). \quad \text{car}(\text{car}_{13}). \quad \text{car}(\text{car}_{14}).
\]

\[
\text{car}(\text{car}_{21}). \quad \text{car}(\text{car}_{22}). \quad \text{car}(\text{car}_{23}).
\]

... 

\[
\text{shape}(\text{hexagon}). \quad \text{shape}(\text{rectangle}). \quad \text{shape}(\text{triangle}). \ldots
\]

...
Michalski's Trains: background knowledge

Define all the trains:

\% eastbound train 1
has_car(east1, car_11). long(car_11).
wheels(car_11,2). load(car_11, rectangle, 3).

has_car(east1, car_12). short(car_12).
wheels(car_12,2). shape(car_12, rectangle).

\% eastbound train 2
has_car(east2, car_21). short(car_21).
open_car(car_21). load(car_21, triangle, 1).

... 

Michalski's Trains: foreground examples

Positive examples:
- eastbound(east1).
- eastbound(east2).
- eastbound(east3).
- eastbound(east4).
- eastbound(east5).

Negative examples:
- eastbound(west6).
- eastbound(west7).
- eastbound(west8).
- eastbound(west9).
- eastbound(west10).

Logically, the negative examples are instances (here the trains west1, west2, etc.) for which the target predicate (here eastbound/1) is false.

Michalski's Trains: hypothesis

Learned using Aleph in SWI Prolog:

[clauses constructed] [70]
[search time] [0.01]
[best clause]
eastbound(A) :-
  has_car(A, B), short(B), closed(B).
[pos cover = 5 neg cover = 0] [pos-neg] [5]
true.

?-

Learning First Order Rules

- to learn logic programs we can adopt propositional rule learning methods
- the target relation is clause head, e.g. ancestor/2
  - think of this as the consequent
- the clause body is constructed using predicates from background knowledge
  - think of this as the antecedent
- unlike propositional rules first order rules can have
  - variables
  - tests on more than one variable at a time
  - recursion
- learning is set up as a search through the hypothesis space of first order rules
Example: First Order Rule for Classifying Web Pages

[Slattery, 1997]

course(A) ←
  has-word(A, instructor),
  not has-word(A, good),
  link-from(A, B),
  has-word(B, assign),
  not link-from(B, C)

Train: 31/31, Test: 31/34

Can learn graph-type representations.

Specializing Rules in FOIL

Learning rule: \( P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \ldots L_n \)
Candidate specializations:

- \( Q(v_1, \ldots, v_r) \), where at least one of the \( v_i \) in the created literal must already exist as a variable in the rule.
- \( Equal(x_j, x_k) \), where \( x_j \) and \( x_k \) are variables already present in the rule.
- The negation of either of the above forms of literals

Completeness and Consistency (Correctness)

\( \mathcal{H} \): complete, consistent
\[ \text{covers}(\mathcal{H}, \mathcal{E}) \]

\( \mathcal{H} \): incomplete, consistent
\[ \text{covers}(\mathcal{H}, \mathcal{E}) \]
Completeness and Consistency (Correctness)

$H$: complete, inconsistent

$H$: incomplete, inconsistent

Variable Bindings

- A substitution replaces variables by terms
- Substitution $\theta$ applied to literal $L$ is written $L\theta$
- If $\theta = \{x/z, y/z\}$ and $L = P(x, y)$ then $L\theta = P(3, z)$

FOIL bindings are substitutions mapping each variable to a constant:

$$\text{GrandDaughter}(x, y) \leftarrow$$

With 4 constants in our examples we have 16 possible bindings:

$$\{x/Victor, y/Sharon\}, \{x/Victor, y/Boh\}, \ldots$$

With 1 positive example of GrandDaughter, other 15 bindings are negative:

$$\text{GrandDaughter}(Victor, Sharon)$$

Information Gain in FOIL

$$\text{Foil Gain}(L, R) \equiv t \left( \log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

Where

- $L$ is the candidate literal to add to rule $R$
- $p_0 = \text{number of positive bindings of } R$
- $n_0 = \text{number of negative bindings of } R$
- $p_1 = \text{number of positive bindings of } R + L$
- $n_1 = \text{number of negative bindings of } R + L$
- $t$ is the number of positive bindings of $R$ also covered by $R + L$

Note

- $- \log_2 \frac{p_0}{p_0 + n_0}$ is minimum number of bits to identify an arbitrary positive binding among the bindings of $R$
- $- \log_2 \frac{p_1}{p_1 + n_1}$ is minimum number of bits to identify an arbitrary positive binding among the bindings of $R + L$
- $\text{Foil Gain}(L, R)$ measures the reduction due to $L$ in the total number of bits needed to encode the classification of all positive bindings of $R$
Learning with FOIL

Target Predicate: ancestor

Examples:

1. New clause: ancestor(X,Y) :-.
   Best antecedent: parent(X,Y) Gain: 31.02
   Learned clause: ancestor(X,Y) :- parent(X,Y).

2. New clause: ancestor(X,Y) :-.
   Best antecedent: parent(Z,Y) Gain: 13.65
   Best antecedent: ancestor(X,Z) Gain: 27.86
   Learned clause: ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).

Definition: ancestor(X,Y) :- parent(X,Y).
   ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).

Completeness and Correctness

Instances:
- pairs of nodes, e.g. (1, 5), with graph described by literals LinkedTo(0,1), ~LinkedTo(0,8) etc.

Target function:
- CanReach(x,y) true iff directed path from x to y

Hypothesis space:
- Each $h \in H$ is a set of Horn clauses using predicates LinkedTo (and CanReach)
FOIL as a propositional learner

- target predicate is usual form of class value and attribute values
  - Class1(V1, V2, . . . , Vm), Class2(V1, V2, . . . , Vm), . . .
- literals restricted to those in typical propositional learners
  - Vj = const, Vj > num, Vj ≤ num
- plus extended set
  - Vj = Vj, Vj ≥ Vj
- FOIL results vs C4.5
  - accuracy competitive, especially with extended literal set
  - FOIL required longer computation
  - C4.5 more compact, i.e. better pruning

FOIL learns ProLOG programs from examples

- from I. Bratko’s book “ProLOG Programming for Artificial Intelligence”
- introductory list programming problems
- training sets by randomly sampling from universe of 3 and 4 element lists
- FOIL learned most predicates completely and correctly
  - some predicates learned in limited
  - some learned in more complex form than in book
  - most learned in few seconds, some much longer

Determinate Literals

- adding a new literal Q(X, Y) where Y is the unique value for X
- this will result in zero gain!
- FOIL gives a small positive gain to literals introducing a new variable
- BUT there may be many such literals
Refining clause $A \leftarrow L_1, L_2, \ldots, L_{m-1}$

- a new literal $L_m$ is determinate if
  - $L_m$ introduces new variable(s)
  - there is exactly one extension of each positive tuple that satisfies $L_m$
  - there is no more than one extension of each negative tuple that satisfies $L_m$

So $L_m$ preserves all positive tuples and does not increase the set of bindings.

At each step in specializing the current clause, unless FOIL finds a literal with close to the maximum possible gain, it adds all determinate literals to the clause, and iterates. This “lookahead” helps to overcome greedy search myopia without blowing up the search space. The clause is post-pruned to remove redundant literals.

Identifying document components

- Problem: learn rules to locate logical components of documents
- documents have varying numbers of components
- relationships (e.g. alignment) between pairs of components
- inherently relational task
- target relations to identify sender, receiver, date, reference, logo.

- background knowledge
  - 20 single page documents
  - 244 components
  - 57 relations specifying
    * component type (text or picture)
    * position on page
    * alignment with other components
- test set error from 0% to 4%
Text applications of first-order logic in learning

Q: when to use first-order logic in machine learning?
A: when relations are important.

Learning information extraction rules

What is information extraction? Fill a pre-defined template from a given text.
Partial approach to finding meaning of documents.
Given: examples of texts and filled templates
Learn: rules for filling template slots based on text

Representation for text

Example: text categorization, i.e. assign a document to one of a finite set of categories.

Propositional learners:
- use a "bag-of-words", often with frequency-based measures
- disregards word order, e.g. equivalence of
  That's true, I did not do it
  That's not true, I did do it

First-order learners: word-order predicates in background knowledge

has_word(Doc, Word, Pos)
Pos1 < Pos2

Sample Job Posting

Subject: US-TN-SOFTWARE PROGRAMMER
Date: 17 Nov 1996 17:37:29 GMT
Organization: Reference.Com Posting Service
Message-ID: <56nigp$mrs@bilbo.reference.com>
SOFTWARE PROGRAMMER
Position available for Software Programmer experienced in generating software for PC-Based Voice Mail systems. Experienced in C Programming. Must be familiar with communicating with and controlling voice cards; prefers Dialogic, however, experience with others such as Rhetoric and Natural Microsystems is okay. Prefer 5 years or more experience with PC Based Voice Mail, but will consider as little as 2 years. Need to find a Senior level person who can come on board and pick up code with very little training. Present Operating System is DOS. May go to OS-2 or UNIX in future.
Please reply to:
Kim Anderson
AdNET
(901) 458-2888 fax
kimander@memphisonline.com
Example job posting

Subject: US-TN-SOFTWARE PROGRAMMER
Date: 17 Nov 1996 17:37:29 GMT
Organization: Reference.Com Posting Service
Message-ID: \56nigp$mrs@bilbo.reference.com

SOFTWARE PROGRAMMER

Position available for Software Programmer experienced in generating software for PC-Based Voice Mail systems. Experienced in C Programming. Must be familiar with communicating with and controlling voice cards; preferable Dialogic, however, experience with others such as Rhetorix and Natural Microsystems is okay. Prefer 5 years or more experience with PC Based Voice Mail, but will consider as little as 2 years. Need to find a Senior level person who can come on board and pick up code with very little training. Present Operating System is DOS. May go to OS-2 or UNIX in future.

Please reply to:
Kim Anderson
AdNET
(901) 458-2888 fax
kimander@memphisonline.com

Example filled template

title: SOFTWARE PROGRAMMER
salary:
company:
recruiter:
state: TN
city:
country: US
language: C
platform: PC | DOS | OS-2 | UNIX
application:
area: Voice Mail
req_years_experience: 2
desired_years_experience: 5
req_degree:
desired_degree:
post_date: 17 Nov 1996

A learning method for Information Extraction

Rapier (Califf and Mooney, 2002) is an ILP-based approach which learns information extraction rules based on regular expression-type patterns

Pre-Filler Patterns: what must match before filler
Filler Patterns: what the filler pattern is
Post-Filler Patterns: what must match after filler

Algorithm uses a combined bottom-up (specific-to-general) and top-down (general-to-specific) approach to generalise rules.

syntactic analysis: Brill part-of-speech tagger
semantic analysis: WordNet (Miller, 1993)
**Progol**

**Progol**: Reduce combinatorial explosion by generating most specific acceptable $h$ as lower bound on search space

1. User specifies $H$ by stating predicates, functions, and forms of arguments allowed for each
2. Progol uses sequential covering algorithm.
   For each $(x_i, f(x_i))$
   - Find most specific hypothesis $h_i$ s.t. $B \land h_i \land x_i \vdash f(x_i)$
     - actually, considers only $k$-step entailment
3. Conduct general-to-specific search bounded by specific hypothesis $h_i$, choosing hypothesis with minimum description length

**Protein structure**

fold('Four-helical up-and-down bundle', P) :-
  helix(P, H1),
  length(H1, hi),
  position(P, H1, Pos),
  interval(1 <= Pos <= 3),
  adjacent(P, H1, H2),
  helix(P, H2).

"The protein P has fold class 'Four-helical up-and-down bundle' if it contains a long helix H1 at a a secondary structure position between 1 and 3 and H1 is followed by a second helix H2".

**Protein structure classification**

- Protein structure largely driven by careful inspection of experimental data by human experts
- Rapid production of protein structures from structural-genomics projects
- Machine-learning strategy that automatically determines structural principles describing 45 classes of fold
- Rules learnt were both statistically significant and meaningful to protein experts

A. Cootes, S.H. Muggleton, and M.J.E. Sternberg
**Immunoglobulin:**

Has antiparallel sheets B and C; B has 3 strands, topology 123; C has 4 strands, topology 2134.

**TIM barrel:**

Has between 5 and 9 helices; Has a parallel sheet of 8 strands.

**SH3:**

Has an antiparallel sheet. C and D are the 1st and 4th strands in the sheet B respectively. C and D are the end strands of B and are 4.360 (± 2.18) angstroms apart. D contains a proline in the c-terminal end.

**Closed-loop Learning**

**Summary**

- can be viewed as an extended approach to rule learning
- BUT: much more ...
- learning in a general-purpose programming language
- use of rich background knowledge
- incorporate arbitrary program elements into clauses (rules)
- background knowledge can grow as a result of learning
- control search with declarative bias
- learning probabilistic logic programs