Lecture 6: Inductive Logic Programming
Cognitive Systems - Machine Learning

Part II: Special Aspects of Concept Learning

Sequential Covering, FOIL, Inverted Resolution, EBL

last change November 26, 2014
it is useful to learn the target function as a set of if-then-rules
- one of the most expressive and human readable representations
- e.g. decision trees

**Inductive Logic Programming (ILP):**
- rules are learned directly
- designed to learn *first-order rules* (i.e. including variables)
- *sequential covering* to incrementally grow the final set of rules

PROLOG programs are sets of first-order rules

⇒ a general-purpose method capable of learning such rule sets can be viewed as an algorithm for automatically inferring PROLOG programs
Outline

- Sequential Covering
  - General to Specific Beam Search
  - Learn One Rule
  - Performance Measures
  - Sequential vs. Simultaneous Covering
- First-Order Rules
- FOIL
  - Hypothesis Space
  - Generating Candidate Specializations
  - Example
  - FOIL Gain
- Induction as Inverted Deduction
  - Inverse Resolution
  - Generalization, $\theta$-Subsumption, Entailment
- Explanation-based Learning (EBL)
- Summary
Examples

- **Propositional Logic:**

  IF Humidity=normal AND Outlook=sunny
  THEN PlayTennis=yes

  IF Humidity=normal AND Temperature=mild AND Wind=weak
  THEN PlayTennis=yes

  playTennis :- humidity(normal), outlook(sunny).
  playTennis :- humidity(normal), temperature(mild), wind(weak).

- **First Order Logic:**

  IF Parent(x,y) THEN Ancestor(x,y)

  IF Parent(x,z) AND Ancestor(z,y) THEN Ancestor(x,y)

  ancestor(X,Y) :- parent(X,Y).
  ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
Sequential Covering

- **basic strategy**: learn one rule, remove the data it covers, then iterate this process
- one of the most widespread approaches to learn a disjunctive set of rules (each rule itself is conjunctive)
- subroutine **LEARN-ONE-RULE**
  - accepts a set of positive and negative examples as input and outputs a single rule that covers many of the positive and few of the negative examples
  - **high accuracy**: predictions should be correct
  - **low coverage**: not necessarily predictions for each example
- performs a greedy search without backtracking
  ⇒ no guarantee to find the smallest or best set of rules
Sequential Covering

Algorithm

SEQUENTIAL-COVERING( Target_attribute, Attributes, Examples, Threshold)

- \( \text{Learned Rules} \leftarrow \{\} \)
- \( \text{Rule} \leftarrow \)
  LEARN-ONE-RULE( Target_attribute, Attributes, Examples)
- While PERFORMANCE( Rule, Examples) > Threshold, Do
  - \( \text{Learned rules} \leftarrow \text{Learned rules} + \text{Rule} \)
  - \( \text{Examples} \leftarrow \text{Examples} - \{ \text{examples correctly classified by Rule} \} \)
  - \( \text{Rule} \leftarrow \text{LEARN-ONE-RULE}( \text{Target_attribute, Attributes, Examples} ) \)
- \( \text{Learned rules} \leftarrow \) sort \( \text{Learned rules} \) accord to PERFORMANCE over Examples
- return \( \text{Learned rules} \)
General to Specific Beam Search

**question**: How shall LEARN-ONE-RULE be designed to meet the needs of the sequential covering algorithm?

- organize the search through $H$ analogous to ID3
  - **but** follow only the most promising branch in the tree at each step
  - begin by considering the most general rule precondition (i.e. empty test)
  - then greedily add the attribute test that most improves rule performance over the training examples
- unlike to ID3, this implementation follows only a single descendant at each search step rather than growing a sub-tree that covers all possible values of the selected attribute
so far a local greedy search (analogous to hill-climbing) is employed

- danger of suboptimal results
- susceptible to the typical hill-climbing problems

⇒ extension to beam search

⇒ algorithm maintains a list of the $k$ best candidates at each step
⇒ at each step, descendants are generated for each of the $k$ candidates and the resulting set is again reduced to the $k$ most promising candidates
General to Specific Beam Search

- IF Wind=weak
  THEN PlayTennis=yes

- IF Wind=strong
  THEN PlayTennis=no

- IF Humidity=normal
  Wind=weak
  THEN PlayTennis=yes

- IF Humidity=normal
  Wind=strong
  THEN PlayTennis=yes

- IF Humidity=normal
  Outlook=sunny
  THEN PlayTennis=yes

- IF Humidity=high
  THEN PlayTennis=yes

- ...
LEARN-ONE-RULE\((Target\_attribute, Attributes, Examples, k)\)

Returns a single rule that covers some of the Examples. Conducts a general to specific greedy beam search for the best rule, guided by the PERFORMANCE metric.

- Initialize \(Best\_hypothesis\) to the most general hypothesis \(\emptyset\)
- Initialize \(Candidate\_hypotheses\) to the set \(\{Best\_hypothesis\}\)
- While \(Candidate\_hypotheses\) is not empty, Do

1. **Generate the next more specific Candidate\_hypotheses**
   - \(New\_Candidate\_hypotheses\) ← new generated and specialized candidates

2. **Update Best\_hypotheses**
   - Select hypothesis \(h\) from \(New\_Candidate\_hypotheses\) with best PERFORMANCE over Examples
   - IF PERFORMANCE of \(h\) > PERFORMANCE of \(Best\_hypothesis\) THEN set \(h\) as new \(Best\_hypothesis\)

3. **Update Candidate\_hypotheses**
   - \(Candidate\_hypotheses\) ← the \(k\) best members of \(New\_Candidate\_hypotheses\)
LEARN-ONE-RULE cont.

- Return a rule of the form
  "IF Best\_hypothesis THEN prediction"
  where prediction is the most frequent value of Target\_attribute
  among those Examples that match Best\_hypothesis.
Example: Learn One Rule

- Learn *one* rule that covers a certain amount of positive examples
- with high accuracy
- remove the covered positive examples

Example

- (s1) Sky = sunny
- (a1) AirTemp = warm
- (h1) Humidity = normal
- (w1) Water = warm

- (s2) Sky = rainy
- (a2) AirTemp = cold
- (h2) Humidity = high
- (w2) Water = cool
Example cont.

- Assume $k = 4$
- Current most specific hypotheses: $s_1, s_2, a_1, a_2, h_1, h_2, w_1, w_2$
- Assume best possible hypotheses wrt performance $P$: $s_1, a_2, h_1, w_1$
- Generate new candidate hypotheses, e.g. by specializing $s_1$:
  - $s_1 \& s_1$ (duplicate)
  - $s_1 \& s_2$ (inconsistent)
  - $s_1 \& a_1$
  - $s_1 \& a_2$
  - $s_1 \& h_1$
  - ...

Performance Measures

- Relative Frequency (numbers of correctly classified examples by all examples)
  \[ \frac{n_c}{n} \]

- Entropy $S$ as set of examples that match precondition, $p_i$ proportion of examples from $S$ for which the target function takes the $i$-th value
  \[ -\text{Entropy}(S) = \sum_{i=1}^{c} p_i \log_2 p_i \]
Sequential vs. Simultaneous Covering

- **sequential covering:**
  - learn just one rule at a time, remove the covered examples and repeat the process on the remaining examples
  - many search steps, making independent decisions to select each precondition for each rule

- **simultaneous covering:**
  - ID3 learns the entire set of disjunctive rules simultaneously as part of a single search for a decision tree
  - fewer search steps, because each choice influences the preconditions of all rules

⇒ Choice depends on how much data is available
  - plentiful → sequential covering (more steps supported)
  - scarce → simultaneous covering (decision sharing effective)
Differences in Search

**generate-then-test:**
- search through all syntactically legal hypotheses
- generation of the successor hypotheses is only based on the syntax of the hypothesis representation
- training data is considered after generation to choose among the candidate hypotheses
- each training example is considered several times
  \[\Rightarrow\] impact of noisy data is minimized

**example driven:**
- individual training examples constrain the generation of hypotheses
- e.g. FIND-S, CANDIDATE ELIMINATION
  \[\Rightarrow\] search can easily be misled
Learning First-Order Rules

- propositional expressions do not contain variables and are therefore less expressive than first-order expressions.

- no general way to describe essential relations among the values of attributes.

- **Literals**: We refer to atomic formulas also as atoms. Positive and negative atoms \((P, \neg P)\) are called positive/negative literals.
  
  e.g. \(\text{parent}(x, y)\) or \(\neg \text{parent}(x, y)\)...

- **Clauses**: A clause is a disjunction of positive and negative literals.
  
  e.g. \(\text{mother}_\text{of}(x, y) \lor \text{father}_\text{of}(z, y)\)
Learning First-Order Rules

Now we consider learning first-order rules (Horn Theories)

- a **Horn Clause** is a clause containing at most *one positive literal*

expression of the form:

\[ H \lor \neg L_1 \lor \neg L_2 \lor \cdots \lor \neg L_n \]

\[ \iff H \iff (L_1 \land L_2 \land \cdots \land L_n) \]

\[ \iff \text{IF } (L_1 \land L_2 \land \cdots \land L_n) \text{ THEN } H \]

- FOL terminology see *Intelligente Agenten*
natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE

outputs sets of first-order rules similar to Horn Clauses with two exceptions

1. **more restricted**, because literals are not permitted to contain function symbols
2. **more expressive**, because literals in the body can be negated

differences between FOIL and earlier algorithms:

- seeks only rules that predict when the target literal is *True*
- conducts a simple hill-climbing search instead of beam search
Algorithm

**FOIL** \((Target\_predicate, Predicates, Examples)\)

- \(Pos \leftarrow\) those \(Examples\) for which the \(Target\_predicate\) is \(True\)
- \(Neg \leftarrow\) those \(Examples\) for which the \(Target\_predicate\) is \(False\)
- \(Learned\_rules \leftarrow\) \(
\)
- while \(Pos\), Do
  - \(NewRule \leftarrow\) the rule that predicts \(Target\_predicate\) with no precondition
  - \(NewRuleNeg \leftarrow\) \(Neg\)
  - while \(NewRuleNeg\), Do
    - \(Candidate\_literals \leftarrow\) generate new literals for \(NewRule\), based on \(Predicates\)
    - \(Best\_literal \leftarrow\) \(\max_{L \in Candidate\_literals} \text{FoilGain}(L, NewRule)\)
    - add \(Best\_literal\) to preconditions of \(NewRule\)
    - \(NewRuleNeg \leftarrow\) subset of \(NewRuleNeg\) that satisfies \(NewRule\) preconditions
      - \(Learned\_rules \leftarrow\) \(Learned\_rules + NewRule\)
      - \(Pos \leftarrow\) \(Pos - \{\text{members of } Pos\text{ covered by } NewRule\}\)
  - Return \(Learned\_rules\)
**outer loop (set of rules):**
- specific-to-general search
- initially, there are no rules, so that each example will be classified negative (most specific)
- each new rule raises the number of examples classified as positive (more general)
- disjunctive connection of rules

**inner loop (preconditions for one rule):**
- general-to-specific search
- initially, there are no preconditions, so that each example satisfies the rule (most general)
- each new precondition raises the number of examples classified as negative (more specific)
- conjunctive connection of preconditions
Generating Candidate Specializations

- current rule:
  \[
P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \ldots L_n \text{ where}
  \]
  \[
  \Rightarrow L_1 \ldots L_n \text{ are the preconditions and}
  \]
  \[
  \Rightarrow P(x_1, x_2, \ldots, x_k) \text{ is the head of the rule}
  \]

- FOIL generates candidate specializations by considering new literals \(L_{n+1}\) that fit one of the following forms:
  - \(Q(v_1, \ldots, v_r)\) where \(Q \in Predicates\) and the \(v_i\) are new or already present variables (at least one \(v_i\) must already be present)
  - \(Equal(x_j, x_k)\) where \(x_j\) and \(x_k\) are already present in the rule
  - the negation of either of the above forms
## Training Data Example

<table>
<thead>
<tr>
<th>Examples</th>
<th>Background Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrandDaughter(Victor, Sharon)</td>
<td>Female(Sharon)</td>
</tr>
<tr>
<td>(\neg) GrandDaughter(Tom, Bob)</td>
<td>Father(Sharon, Bob)</td>
</tr>
<tr>
<td>(\neg) GrandDaughter(Victor, Victor)</td>
<td>Father(Tom, Bob)</td>
</tr>
<tr>
<td></td>
<td>Father(Bob, Victor)</td>
</tr>
</tbody>
</table>
Example

GrandDaughter(x,y) ←

Candidate additions to precondition:

\[ Equal(x,y), \text{Female}(x), \text{Female}(y), \text{Father}(x,y), \text{Father}(y,x), \text{Father}(x,z), \text{Father}(z,x), \text{Father}(z,y), \text{and the negations to these literals} \]

Assume greedy selection of Father(y,z):

\[ \text{GrandDaughter}(x,y) \leftarrow \text{Father}(y,z) \]

Candidate additions:

\[ \text{the ones from above and Female}(z), \text{Equal}(z,y), \text{Father}(z,w), \text{Father}(w,z), \text{and their negations} \]

Learned Rule:

\[ \text{GrandDaughter}(x,y) \leftarrow \text{Father}(y,z) \wedge \text{Father}(z,x) \wedge \text{Female}(y) \]
FOIL Gain

\[ \text{FoilGain}(L, R) \equiv t \left( \log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right) \]

with

- \( L \) as new literal introduced in rule \( R \) to gain new rule \( R' \)
- \( t \) as number of positive bindings of rule \( R \) which are still covered by \( R' \)
- \( p_1 \) as number of positive bindings of rule \( R' \) and \( n_1 \) as number of negative bindings
- \( p_0 \) as number of positive bindings of rule \( R \) and \( n_0 \) as number of negative bindings

Remark: Bindings are the number of instantiations of the variables by constants. A binding is positive if the instantiated rule covers a positive example.
Learning Recursive Rule Sets

- Extend FOIL such that the target predicate can also be included in the preconditions with the same restrictions to variables as before.

- Problem: rule sets that produce infinite recursions

- FOIL uses a generate-and-test strategy; alternatively recursive rule sets can be learned by analytical methods (see lecture inductive programming)

\[
\text{ancestor}(X,Y) :- \text{parent}(X,Y).
\]
\[
\text{ancestor}(X,Y) :- \text{parent}(X,Z), \text{ancestor}(Z,Y).
\]
**observation:** induction is just the inverse of deduction

in general, machine learning involves building theories that explain the observed data

Given some data \( D \) and some background knowledge \( B \), learning can be described as generating a hypothesis \( h \) that, together with \( B \), explains \( D \).

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

the above equation casts the learning problem in the framework of deductive inference and formal logic
features of inverted deduction:
- subsumes the common definition of learning as finding some general concept
- background knowledge allows a more rich definition of when a hypothesis $h$ is said to “fit” the data

practical difficulties:
- noisy data makes the logical framework completely lose the ability to distinguish between truth and falsehood
- search is intractable
- background knowledge often increases the complexity of $H$
resolution is a general method for automated deduction

complete and sound method for deductive inference

see Intelligente Agenten

Inverse Resolution Operator (propositional form):

1. Given initial clause $C_1$ and $C$, find a literal $L$ that occurs in $C_1$ but not in clause $C$.
2. Form the second clause $C_2$ by including the following literals

$$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$$

inverse resolution is not deterministic
Inverting Resolution

\[ C_1 : \text{PassExam} \lor \sim \text{KnowMaterial} \]

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

\[ C : \text{PassExam} \lor \sim \text{Study} \]
Inverse Resolution Operator (first-order form):

resolution rule:

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 $\neg L_2 \theta$. That is,

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

analytical derivation of the inverse resolution rule:

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

where $\theta = \theta_1 \theta_2$

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

where $L_2 = \neg L_1 \theta_1 \theta_2^{-1}$

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

Inverting Resolution

\[ D = \{ \text{GrandChild}(Bob, Shannon) \} \]

\[ B = \{ \text{Father}(Shannon, Tom), \text{Father}(Tom, Bob) \} \]
Remarks

- ILP based on restricted variants of inverted resolution was introduced mainly by Muggleton
- Inverse resolution leads to combinatorial explosion of candidate hypotheses
  - many possibilities to combine hypotheses with background-knowledge in order to generate more specific hypotheses

Other techniques:

- $\theta$-Subsumption (used by GOLEM)
  - replace terms by variables (intervened unification)
- inverse entailment (used by PROGOL)
  - generates just a single more specific hypothesis that entails the observed data
interesting to consider the relationship between the *more_general_than* relation and inverse entailment

*more_general_than*:

\[ h_j \geq^g h_k \iff (\forall x \in X)[h_k(x) \rightarrow h_j(x)]. \]

A hypothesis can also be expressed as \( c(x) \leftarrow h(x) \).

\( \theta - \text{subsumption} \):
Consider two clauses \( C_j \) and \( C_k \), both of the form \( H \lor L_1 \lor \ldots \lor L_n \), where \( H \) is a positive literal and the \( L_i \) are arbitrary literals. Clause \( C_j \) is said to *\( \theta - \text{subsume} \)* clause \( C_k \) iff \( (\exists \theta)[C_j \theta \subseteq C_k] \).

*entailment*:
Consider two clauses \( C_j \) and \( C_k \). Clause \( C_j \) is said to *entail* clause \( C_k \) (written \( C_j \vdash C_k \)) iff \( C_k \) follows deductively from \( C_j \).
if \( h_1 \geq_g h_2 \) then \( C_1 : c(x) \leftarrow h_1(x) \) \( \theta \)-subsumes \( C_2 : c(x) \leftarrow h_2(x) \)

furthermore, \( \theta \)-subsumption can hold even when the clauses have different heads

\[
\begin{align*}
A : & \text{Mother}(x, y) \leftarrow \text{Father}(x, z) \land \text{Spouse}(z, y) \\
B : & \text{Mother}(x, L) \leftarrow \text{Father}(x, B) \land \text{Spouse}(B, y) \land \text{Female}(x)
\end{align*}
\]

\( A\theta \subseteq B \) if we choose \( \theta = \{y/L, z/B\} \)

\( \theta \)-subsumption is a special case of entailment

\[
\begin{align*}
A : & \text{Elephant}(\text{father}_\text{of}(x)) \leftarrow \text{Elephant}(x) \\
B : & \text{Elephant}(\text{father}_\text{of}(\text{father}_\text{of}(y))) \leftarrow \text{Elephant}(y)
\end{align*}
\]

\( A \models B \), but \( \neg \exists \theta[A\theta \subseteq B] \)
Generalization is a special case of $\theta$-Subsumption.

$\theta$-Subsumption is a special case of entailment.

In its most general form, inverse entailment produces intractable searches.

$\theta$-Subsumption provides a convenient notion that lies midway between generalization and entailment!
Explanation-Based Learning

- Using prior knowledge and deductive reasoning to augment information given by trainings examples
- explanation-based learning (EBL)
  e.g., Mitchell et al., 1986
- Different approaches: background-knowledge is or is not complete and correct
- **Reach more accuracy with less examples!**
- For example, applied in inductive programming (Dialogs, Igor)
- Related to learning on the knowledge level and analogy-based learning
Inductive Learning

Given:
- Instance space $X$
- Hypothesis space $H$
- Training examples $D$ of some target function $f$

$$D = \{\langle x_1, f(x_1) \rangle, \ldots \langle x_n, f(x_n) \rangle \}$$

Determine:
- A hypothesis from $H$ consistent with training examples $D$. 
Analytical Learning

Given:
- Instance space $X$
- Hypothesis space $H$
- Training examples $D$ of some target function $f$.

$$D = \{\langle x_1, f(x_1) \rangle, \ldots \langle x_n, f(x_n) \rangle \}$$

- Domain theory $B$ for explaining training examples

Determine:
- A hypothesis from $H$ consistent with both the training examples $D$ and domain theory $B$.

We say
- $B$ “explains” $\langle x, f(x) \rangle$ if $x + B \vdash f(x)$
- $B$ is “consistent with” $h$ if $B \nvdash \neg h$
SafeToStack(x,y) Learning Problem

Given:
- Instances: pairs of physical objects
- Hypotheses: Sets of Horn clause rules, e.g.,
  \[ \text{SafeToStack}(x, y) \leftarrow \text{Volume}(x, vx) \land \text{Type}(y, \text{Box}) \]
- Training Examples: typical example is
  \[ \text{SafeToStack}(\text{Obj1}, \text{Obj2}) \]
  \[ \begin{align*}
  \text{On}(\text{Obj1}, \text{Obj2}) & \quad \text{Owner}(\text{Obj1}, \text{Fred}) \\
  \text{Type}(\text{Obj1}, \text{Box}) & \quad \text{Owner}(\text{Obj2}, \text{Louise}) \\
  \text{Type}(\text{Obj2}, \text{Endtable}) & \quad \text{Density}(\text{Obj1}, 0.3) \\
  \text{Color}(\text{Obj1}, \text{Red}) & \quad \text{Material}(\text{Obj1}, \text{Cardbd})
  \end{align*} \]
- Domain Theory:
  \[ \begin{align*}
  \text{SafeToStack}(x, y) & \leftarrow \neg \text{Fragile}(y) \\
  \text{SafeToStack}(x, y) & \leftarrow \text{Lighter}(x, y) \\
  \text{Lighter}(x, y) & \leftarrow \text{Wt}(x, wx) \land \text{Wt}(y, wy) \land \text{Less}(wx, wy)
  \end{align*} \]
**SafeToStack**(x,y) cont.

**Determine:**

- A hypothesis from $H$ consistent with training examples and domain theory.
Prolog-EBG

Prolog-EBG(*TargetConcept*, *Examples*, *DomainTheory*)

- **LearnedRules** ← {}  
- **Pos** ← the positive examples from *Examples*

  for each *PositiveExample* in *Pos* that is not covered by *LearnedRules*, do

  1. **Explain**:  
     - "Explanation" ← an explanation (proof) in terms of *DomainTheory* that *PositiveExample* satisfies *TargetConcept*

  2. **Analyze**:  
     - "SufficientConditions" ← the most general set of features of *PositiveExample* that satisfy *TargetConcept* according to "Explanation".

  (for each *PositiveExample* in *Pos* that is not covered by *LearnedRules*, do)

  3. **Refine**:  
     - "LearnedRules" ← "LearnedRules" + "NewHornClause", where "NewHornClause" is of the form

           \[ \text{TargetConcept} \leftarrow \text{SufficientConditions} \]

- **Return** *LearnedRules*
An Explanation

Explanation:

\[
\begin{align*}
\text{SafeToStack}(\text{Obj1, Obj2}) \\
\text{Lighter}(\text{Obj1, Obj2}) \\
\text{Weight}(\text{Obj1, 0.6}) & \quad \text{Weight}(\text{Obj2, 5}) \\
\text{Volume}(\text{Obj1, 2}) & \quad \text{Density}(\text{Obj1, 0.3}) & \quad \text{Equal}(0.6, 2^{0.3}) & \quad \text{LessThan}(0.6, 5) & \quad \text{Type}(\text{Obj2, Endtable})
\end{align*}
\]

Training Example:
Summary

- In ILP sets of first-order rules can be learned directly
- Hypothesis language of Horn clauses is more expressive than feature vectors (allowing variables, representing relations)
- Suitable for learning over structured data (meshes, chemical structures, graph-representations in general)
- Not only applicable for learning classifiers but also for learning general (recursive) programs (inductive programming)
- Sequential covering algorithms learn just one rule at a time and perform many search steps
Summary

- **FOIL** is a sequential covering algorithm
  - a specific-to-general search is performed to form the result set
  - a general-to-specific search is performed to form each new rule
- Induction can be viewed as the inverse of deduction
- While Foil is a generate-and-test algorithm, approaches based on inverse resolution (**Golem**, **Prolog**) are example-driven
- ILP can be naturally combined with deductive inference: Explanation-based learning allows for enriching training examples by inferences drawn from a theory.
## Learning Terminology

### ILP

<table>
<thead>
<tr>
<th>Supervised Learning</th>
<th>unsupervised learning</th>
</tr>
</thead>
</table>

### Approaches:

<table>
<thead>
<tr>
<th>Concept / Classification</th>
<th>Policy Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbolic</td>
<td>statistical / neuronal network</td>
</tr>
<tr>
<td>inductive (ILP)</td>
<td>analytical (EBL)</td>
</tr>
</tbody>
</table>

### Learning Strategy:

| Data:                     | learning from examples |
| Target Values:            | structured, symbolic concept (true/false) |

Ute Schmid (CogSys, WIAI)