Knowledge in Learning

- Sequential covering algorithms
- FOIL
- Induction as inverse of deduction
- Inductive Logic Programming
Learning Disjunctive Sets of Rules

Method 1: Learn decision tree, convert to rules

Method 2: Sequential covering algorithm:
1. *Learn one rule* with high accuracy, any coverage
2. Remove positive examples covered by this rule
3. Repeat
Sequential Covering Algorithm

\textbf{SEQUENTIAL-COVERING}(Target\_attribute, Attributes, Examples, Threshold)

- Learned\_rules $\leftarrow \{\}$
- \textbf{Rule} $\leftarrow$ \textbf{LEARN-ONE-RULE}(Target\_attribute, Attributes, Examples)
- \textbf{while} \textbf{PERFORMANCE}(Rule, Examples) $>$ Threshold, \textbf{do}
  - Learned\_rules $\leftarrow$ Learned\_rules $+$ Rule
  - Examples $\leftarrow$ Examples $-$ \{examples correctly classified by Rule\}
  - Rule $\leftarrow$ \textbf{LEARN-ONE-RULE}(Target\_attribute, Attributes, Examples)
- Learned\_rules $\leftarrow$ sort Learned\_rules according to \textbf{PERFORMANCE} over Examples
- return Learned\_rules
Learn-One-Rule

IF Wind=weak
THEN PlayTennis=yes

IF Wind=strong
THEN PlayTennis=no

IF Humidity=normal
THEN PlayTennis=yes

IF Humidity=high
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=normal
Outlook=rain
THEN PlayTennis=yes
LEARN-ONE-RULE

- $Pos \leftarrow$ positive Examples
- $Neg \leftarrow$ negative Examples
- while $Pos$, do
  
  Learn a NewRule
  - $NewRule \leftarrow$ most general rule possible
  - $NewRuleNeg \leftarrow Neg$
  - while $NewRuleNeg$, do
    
    Add a new literal to specialize $NewRule$
    1. $Candidate\_literals \leftarrow$ generate candidates
    2. $Best\_literal \leftarrow \arg\max_{L \in Candidate\_literals} Performance(SpecializeRule(NewRule, L))$
    3. add $Best\_literal$ to $NewRule$ preconditions
    4. $NewRuleNeg \leftarrow$ 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 $Learned\_rules$
Learning First Order Rules

Why do that?

• Can learn sets of rules such as

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

• General purpose programming language PROLOG: programs are sets of such rules
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
FOIL

- First-Order Induction of Logic (FOIL)
- Learns Horn clauses without functions
- Allows negated literals in rule body
- Sequential covering algorithm
  - Greedy, hill-climbing approach
  - Seeks only rules for predicting True
- Each new rule generalizes overall concept ($S \rightarrow G$)
- Each added conjunct specializes rule ($G \rightarrow S$)
FOIL(Target Predicate, Predicates, Examples)

- Pos ← positive Examples
- Neg ← negative Examples
- while Pos, do
  
  Learn a NewRule
  – NewRule ← most general rule possible
  – NewRuleNeg ← Neg
  – while NewRuleNeg, do
    
    Add a new literal to specialize NewRule
    1. Candidate_literals ← generate candidates
    2. Best_literal ←
      \[ \text{argmax}_{L \in \text{Candidate_literals}} \text{Foil}\_\text{Gain}(L, \text{NewRule}) \]
    3. add Best_literal to NewRule preconditions
    4. NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
    – Learned_rules ← Learned_rules + NewRule
    – Pos ← Pos – \{members of Pos covered by NewRule\}

- Return Learned_rules
Specializing Rules in FOIL

Learning rule: $P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \ldots L_n$
Candidate specializations add new literal of form:

- $Q(v_1, \ldots, v_r)$, 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.
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 \( R \)
- \( p_0 \) = number of positive bindings of \( R \)
- \( n_0 \) = number of negative bindings of \( R \)
- \( p_1 \) = number of positive bindings of \( R + L \)
- \( n_1 \) = 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 optimal number of bits to indicate the class of a positive binding covered by \( R \)
FOIL Example

Instances:
- Pairs of nodes, e.g. \( \langle 1, 5 \rangle \), with graph described by literals \( \text{LinkedTo}(0, 1) \), \( \neg \text{LinkedTo}(0, 8) \) etc.

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

Hypothesis space:
- Each \( h \in H \) is a set of horn clauses using predicates \( \text{LinkedTo} \) (and \( \text{CanReach} \))

Learned rules:
- \( \text{CanReach}(x, y) \leftarrow \text{LinkedTo}(x, y) \)
- \( \text{CanReach}(x, y) \leftarrow \text{LinkedTo}(x, z) \) \& \( \text{CanReach}(z, y) \)
Induction as Inverted Deduction

Induction is finding $h$ such that

$$(\forall \langle x_i, f(x_i) \rangle \in D) \ B \land h \land x_i \vdash f(x_i)$$

where

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

Idea: Design inductive algorithm by inverting operators for automated deduction.
Deduction: Resolution Rule

\[
\begin{align*}
P & \lor L \\
\neg L & \lor R \\
P & \lor R
\end{align*}
\]

1. Given initial clauses \(C_1\) and \(C_2\), find a literal \(L\) from clause \(C_1\) such that \(\neg L\) occurs in clause \(C_2\).

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

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

where \(\cup\) denotes set union, and “−” denotes set difference.
Inverting Resolution
**Progol**

**Progol**: Reduce comb. explosion by generating the most specific acceptable $h$

1. User specifies $H$ by stating predicates, functions, and forms of arguments allowed for each

2. **Progol** uses sequential covering algorithm. For each $\langle x_i, f(x_i) \rangle$
   - 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
Summary

- Sequential (set) covering
- Inductive Logic Programming (ILP)
  - FOIL
  - Inverse resolution