Learning Sets of Rules

- Sequential covering algorithms
- FOIL
- Inductive Logic Programming

Propositional vs. First-order Rules

**Propositional** (logic) rules do not contain any variables.

**First-order** (logic) rules can contain variables.

<table>
<thead>
<tr>
<th>Name1:</th>
<th>Chelsea</th>
<th>Name2:</th>
<th>Bill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mother1:</td>
<td>Hillary</td>
<td>Mother2:</td>
<td>Virginia</td>
</tr>
<tr>
<td>Father1:</td>
<td>Bill</td>
<td>Father2:</td>
<td>Bruno</td>
</tr>
<tr>
<td>Male1:</td>
<td>False</td>
<td>Male2:</td>
<td>True</td>
</tr>
<tr>
<td>Female1:</td>
<td>True</td>
<td>Female2:</td>
<td>False</td>
</tr>
</tbody>
</table>

\[ \text{Daughter}_{1,2} = \text{TRUE} \]
A propositional representation could only learn the rule:

\[
\text{IF } (\text{Father}_1 = \text{Bill}) \land (\text{Name}_2 = \text{Bill}) \land (\text{Female}_1 = \text{True}) \\
\text{THEN } \text{Daughter}_{1,2} = \text{TRUE}
\]

A first-order representation could learn the rule:

\[
\text{IF } \text{Father}(x, y) \land \text{Female}(y) \text{ THEN } \text{Daughter}(y, x)
\]

Slide CS478–3

Sequential Covering Algorithms

The basic algorithm:

1. Learn one rule
2. Remove the data it covers
3. Repeat

More specific version:

1. Learn one rule with high accuracy, any coverage
2. Remove positive examples covered by this rule
3. Repeat

Slide CS478–4
Generic Covering Algorithm

\[ \text{COVER}(\text{Target\_attr}, \text{Attrs}, \text{Examples}, \text{Threshold}) \]

- \( \text{Learned\_rules} \leftarrow \{\} \)
- \( \text{Rule} \leftarrow \text{LEARN-ONE-RULE}(\text{Target\_attr}, \text{Attrs}, \text{Examples}) \)
- \( \text{WHILE} \ \text{PERFORMANCE}(\text{Rule}, \text{Examples}) > \text{Threshold}, \ \text{DO} \)
  - \( \text{Learned\_rules} \leftarrow \text{Learned\_rules} + \text{Rule} \)
  - \( \text{Examples} \leftarrow \text{Examples} - \{\text{EXAMPLES CORRECTLY CLASSIFIED BY Rule}\} \)
  - \( \text{Rule} \leftarrow \text{LEARN-ONE-RULE}(\text{Target\_attr}, \text{Attrs}, \text{Examples}) \)
- \( \text{Learned\_rules} \leftarrow \text{SORT Learned\_rules ACCORD TO PERFORMANCE OVER Examples} \)
- \( \text{RETURN Learned\_rules} \)

Slide CS478–5

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>Ski?</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>Strong</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>

Slide CS478–6
LEARN-ONE-RULE($Target_{attr}$, $Attrs$, $Examples$)

- $Pos \leftarrow$ positive $Examples$; $Neg \leftarrow$ negative $Examples$
- If $Pos$
  - $NewRule \leftarrow$ most general rule possible; $NewRuleNeg \leftarrow Neg$
  - While $NewRuleNeg$
    1. $Candidate_{literals}(CLs) \leftarrow$ generate candidates
    2. $Best_{literal} \leftarrow \text{argmax}_{L \in CLs}$
      
      $\text{PERFORMANCE}(\text{Specialize}(NewRule, L))$
    3. add $Best_{literal}$ to $NewRule$ preconditions
    4. $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies
       $NewRule$ preconditions
- Return $NewRule$
## Common Performance Metrics

**Entropy:** $S = \text{examples that match the rule’s preconditions.}$

$$-Entropy(S) \equiv \sum_{i=1}^{c} x_i \log_2 x_i$$

<table>
<thead>
<tr>
<th>Relative Frequency:</th>
<th>$\frac{n_c}{n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = # \text{examples the rule matches}$</td>
<td></td>
</tr>
<tr>
<td>$n_c = # \text{examples the rule matches and classifies correctly}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m estimate:</th>
<th>$\frac{n_c + mp}{n + m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p = \text{prior probability of the class assigned by the rule}$</td>
<td></td>
</tr>
<tr>
<td>$m = # \text{examples needed to override the prior}$</td>
<td></td>
</tr>
</tbody>
</table>

### Learn-One-Rule Search Space

- **general-to-specific search**
- searches for a rule with high accuracy, but possibly low coverage
- measure to select the “best” descendant; one whose covered examples have the lowest entropy
- greedy
- can extend to perform a *beam search*
Variants of Rule Learning Programs

- Sequential or simultaneous covering of data?
- General → specific, or specific → general?
- Generate-and-test, or example-driven?
- Whether and how to post-prune?
- What statistical evaluation function?

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

Learning First-Order Rules

- Inductive learning of first-order rules is often called
  inductive logic programming (ILP), because it can be
  used to learn PROLOG programs.

- ILP methods usually learn first-order Horn Clauses. A
  Horn clause is a disjunction of literals that has at most one
  positive literal (see book for details), such as:

  $$C \lor \neg X_1 \lor \ldots \lor \neg X_n$$

  which can conveniently be rewritten as:

  $$X_1 \land \ldots \land X_n \rightarrow C$$
FOIL(Target\_predicate, Predicates, Examples)

- Pos ← positive Examples; Neg ← negative Examples
- While Pos
  NewRule ← most general rule possible; NewRuleNeg ← Neg
  While NewRuleNeg
    1. Candidate\_literals(\(CLs\)) ← generate candidates
    2. Best\_literal ← \(\text{argmax}_{L \in CLs} \text{Foil\_Gain}(L, \text{NewRule})\)
    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

Given a rule:
\[ P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \ldots L_n \]

Candidate specializations can add a 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.

- \(\text{Equal}(x_j, x_k)\), where \(x_j\) and \(x_k\) are variables already present in the rule

- The negation of either of the above forms of literals
FOIL Gain Metric

Two Goals:
1. Decrease coverage of negative examples.
2. Maintain coverage of as many positive examples as possible.

\[ \text{FOIL Gain}(L, R) \equiv t \left[ \log_2 \left( \frac{P_{R+L}}{P_R + N_{R+L}} \right) - \log_2 \left( \frac{P_R}{P_R + N_R} \right) \right] \]

where

- \( L \) is a literal and \( R \) is a rule
- \( P_R \) is the number of positive bindings for \( R \)
- \( N_R \) is the number of negative bindings for \( R \)
- \( P_{R+L} \) is the number of positive bindings for \( R + L \)
- \( N_{R+L} \) is the number of negative bindings for \( R + L \)
- \( t \) is the number of positive bindings of \( R \) and \( R + L \)

Slide CS478-17

Learning Recursive Rules

- FOIL can learn recursive rules, such as:

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

- To learn recursive rules, the target predicate can be added to the list of candidate predicates used during rule learning.

- Special tricks are needed to avoid learning infinitely recursive rules.

Slide CS478-18
FOIL Example

\[
\begin{array}{cccccc}
0 & \rightarrow & 2 & \rightarrow & 3 & \rightarrow & 4 & \rightarrow & 5 \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
1 & & & & & & & & \\
\end{array}
\]

\[x \rightarrow y\] represents \(\text{LinkedTo}(x,y)\)

Instances:

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

Slide CS478–19

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 \(\text{CanReach}\))

Slide CS478–20
Summary

- Rule learning systems have achieved good results and have produced rules that perform at least as well as manually engineered rules.
- Rule learning approaches can consider one attribute value independent of the others.
- To deal with overfitting, rules can be post-pruned.
- To handle noise, the criteria for adding literals must be loosened up.
- But the search can become intractable if the space of literals gets too large.
- Hill-climbing search can get stuck on local maxima.
- Closed-world assumption required for negative examples.

Slide CS478–21