Learning Sets of Rules

- Paths in Decision Tree can be seen as rules that imply a label.
- Can extract a set of rules from DT.
- Instead can learn set of rules directly.
- How? Most common are the Sequential covering algorithms.

Sequential Covering Algorithms

- Repeat:
  - Learn one rule
  - Remove examples explained by rule from training set
- Until all examples explained

Sequential Covering

- To learn one rule: pick label/conclusion and grow rule for it
- To grow rule: repeatedly add condition to the rule. How?
- As in DT: evaluate all possible additions by scoring them using a "gain formula".
- What is an appropriate criterion?

Refinements and Variants

- Beam search instead of greedy search
- Learning rules for all classes vs. using a default class (e.g. just for + in +/- case).
- How to pick label to learn rule for?
- How to handle multi-class data?
- Grow rule to cover specific example helps focus the search
- Pruning

Pruning

- REP is good for accuracy but expensive
- IREP/RIPPER retain acc but faster
  - Prune tail of rule – not every subset
  - Replace rule with newly learned rule in new context (different data b/c other examples removed by rules that were learned later)
  - Or keep original rule
Relational Rules

- \( p(x) \) a shorthand for parent\((x)\)
- \( gp(x) \) a shorthand for grand-parent\((x)\)
- \( a(x) \) a shorthand for ancestor\((x)\)

\[
\begin{align*}
p(x, y) \ p(y, z) & \rightarrow gp(x, z) \\
p(x, y) & \rightarrow a(x, y) \\
a(y, z) & \rightarrow a(x, z)
\end{align*}
\]

- Need a new setting for learning

---

Inductive Logic Programming

- "Standard Setting":
- Given \( B \) (background knowledge)
  \( E \) (set of examples)
- Find: \( H \) (set of rules)
- That covers examples and generalizes well.

\[
\begin{align*}
B = \{ e(1,2); e(2,3); e(3,4); e(4,2); e(3,5) \}
\end{align*}
\]

- Positive examples:
  - \( p(1,2), p(1,3), p(1,5), p(3,4), p(3,5), p(4,5) \)

- Negative examples:
  - \( P(3,1), p(4,1), p(5,2), p(5,4) \)

- Learn set of rules. How?

---

FOIL Algorithm

\[
\begin{align*}
& x & y & z \\
& 1 & 2 & \rightarrow 1 2 2 \\
& 1 & 3 & \rightarrow 1 3 2 \\
& 1 & 5 & \rightarrow 1 5 2 \\
& 3 & 4 & \rightarrow 3 4 4 \text{ and } 3 4 5 \\
& 3 & 5 & \rightarrow 3 5 4 \text{ and } 3 5 5 \\
& 4 & 5 & \rightarrow 4 5 2 \\
& \text{-----} \\
& 3 & 1 & \rightarrow 3 1 4 \text{ and } 3 1 5 \\
& 4 & 1 & \rightarrow 4 1 2 \\
& 5 & 2 & \rightarrow x \\
& 5 & 4 & \rightarrow x
\end{align*}
\]

- Proposed rule extension
  \( p(x,y) \leftarrow e(x,z) \)

- Before: 6+ 4-
- After: 8+ 3-
FOIL Algorithm

• Gain criterion
\[
#p \cdot \left( \log \frac{oldpos + oldneg}{oldpos} - \log \frac{newpos + newneg}{newpos} \right)
\]

#p is the number of oldpos that are covered by new

6 \cdot \left( \log \frac{10}{6} - \log \frac{11}{8} \right)

Least General Generalization

• Slightly different setting
  - [e(1,2), e(2,3), e(2,4) \rightarrow p(1,3)]
  - [e(1,2), e(2,3), e(2,4) \rightarrow p(1,4)]
  - LGG is the least general rule that covers both rules
  - \[e(1,2), e(2,3), e(2,4), e(A,B), e(A,C), e(D,E), e(2,F), e(D,G), e(2,H) \rightarrow p(1,F)]\]

• Calculating LGG

LGG based algorithm (roughly)

• Repeat
  - Pick pos example
  - Repeat: combine with other pos example as long as no neg examples covered
  - Output rule
• Until all covered

Prolog

• New Idea
  \[B \land H \models E \]
  \[B \land \neg E \models \neg H\]

• Find all literals implied by \[B \land \neg E\]
• Call this \[\neg \neg H\] -- must be more specific than \[\neg H\]
• Negate to get \[\neg \neg H = \lor c_i\]
• Hyp must satisfy: \[H \leq \lor c_i\]
• Alg: search for \(H\) satisfying this cond

Learning Rules and ILP

• Sequential covering algorithms inspired by DT learning
• Relational Rules raise more challenges
• FOIL extends sequential covering
• LGG: uses logical structure to capture generalization
• Progol: “inverse entailment” helps identify generalization