Learning Disjunctive Sets of Rules

- Method 1. Learn decision tree, convert to rules

- Method 2. Sequential covering algorithm

  i) Learn one rule with high accuracy, any coverage

  ii) Remove positive examples covered by this rule

  iii) Repeat

Sequential Covering Algorithm

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

- *Learned_rules* ← {}
- *Rule* ← LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)
- while PERFORMANCE(*Rule*, *Examples*) > *Threshold*, do
    - *Learned_rules* ← *Learned_rules* + *Rule*
    - *Examples* ← *Examples* − {examples correctly classified by *Rule*}
    - *Rule* ← LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)
- *Learned_rules* ← sort *Learned_rules* accord to PERFORMANCE over *Examples*
- return *Learned_rules*

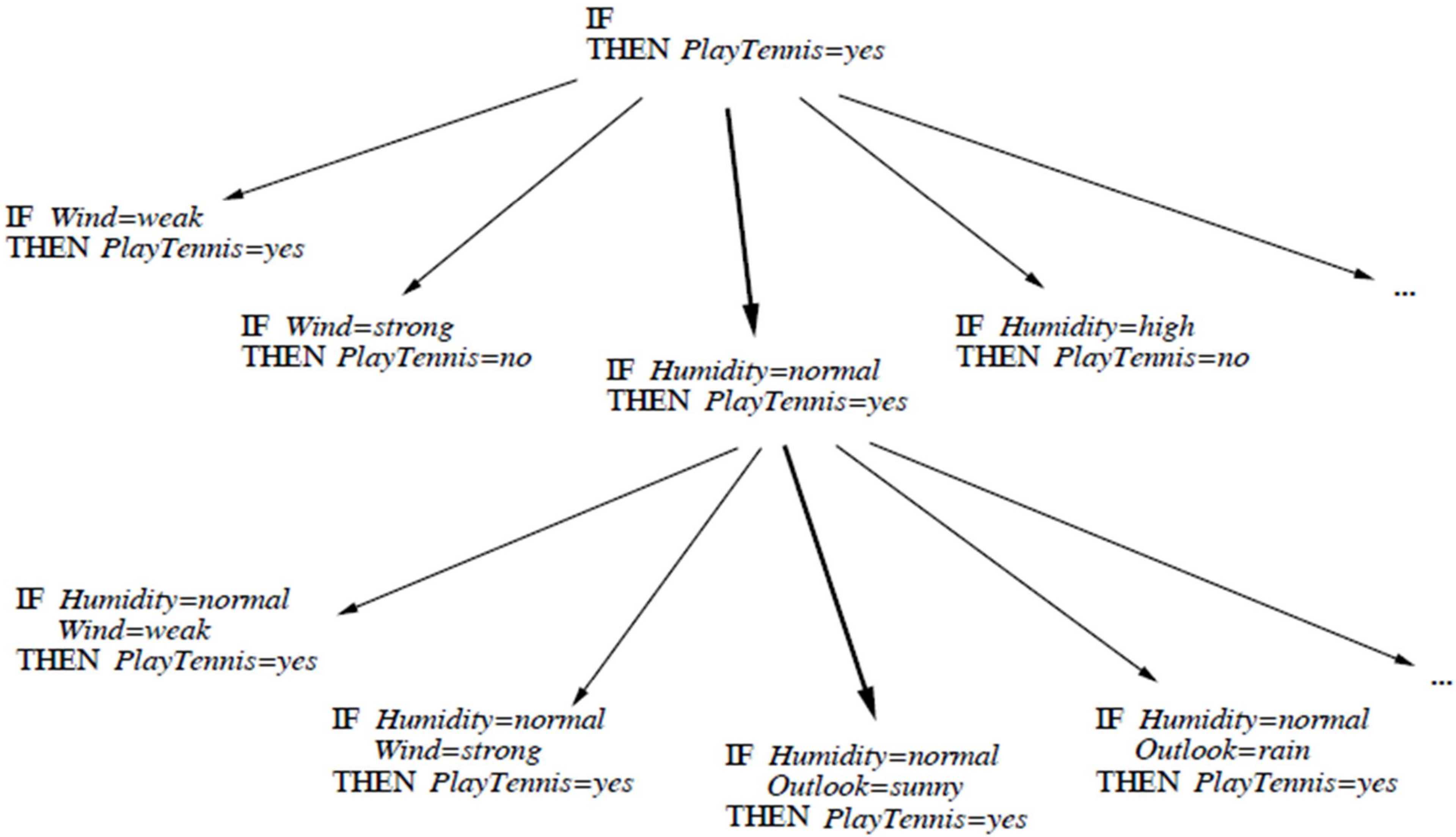
Sequential Covering Algorithm

- The sequential covering algorithm for learning a disjunctive set of rules.
- LEARN-ONE-RULE return a single rule that covers at least some of the Examples.
- PERFORMANCE is a user-provided subroutine to evaluate rule quality.
- The covering algorithm learns rules until it can no longer learn a rule whose performance is above the given Threshold.

LEARN-ONE-RULE

- The search for rule preconditions as LEARN-ONE-RULE proceeds from general to specific.
- At each step, the preconditions of the best rule are specialized in all possible ways.
- Rule postconditions are determined by the examples found to satisfy the preconditions.

LEARN-ONE-RULE



LEARN-ONE-RULE

Pos ← positive *Examples*

Neg ← negative *Examples*

while *Pos* is not empty **do**

Learn a NewRule

- NewRule ← most general rule possible
- NewRuleNeg ← Neg
- **while** NewRuleNeg is not empty **do**

  Add a new literal to specialize NewRule

  - Candidate literals ← generate candidates
  - Best literal ← argmax $_{L \in \text{Candidate literals}}$ Performance(SpecializeRule(NewRule, L))
  - add Best literal to NewRule preconditions
  - NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
- Learned rules ← Learned rules + NewRule
- Pos ← Pos – { members of Pos covered by NewRule }

Return Learned rules

Performance in LEARN-ONE-RULE

- ***Relative frequency***
  - Let **n** denote the number of examples the rule matches and let **nc** denote the number of these that it classifies correctly.
  - The relative frequency estimate of rule performance is **nc / n**
- ***Entropy***
  - Let S be the set of examples that match the rule preconditions.
  - Entropy measures the uniformity of the target function values for this set of examples.
  - We take the negative of the entropy so that better rules will have higher scores.

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

  - where c is the number of distinct values the target function may take on, p_i is the proportion of examples from S for which the target function takes on the i^{th} value.

Learning First Order Rules

- The problem is that propositional representations offer no general way to describe the essential relations among the values of the attributes.
- In contrast, a program using first-order representations could learn the following general rule:

IF Father(y, x) **and** Female(y), **THEN** Daughter(x, y)

- where x and y are variables that can be bound to any person.