Learning Sets of Rules

Learning Rules

If (Color = Red) and (Shape = round) then Class is A
If (Color = Blue) and (Size = large) then Class is B
If (Shape = square) then Class is A

- Natural and intuitive hypotheses
  - Comprehensibility - Easy to understand?

Learning Rules

If (Color = Red) and (Shape = round) then Class is A
If (Color = Blue) and (Size = large) then Class is B
If (Shape = square) then Class is A

- Natural and intuitive hypotheses
  - Comprehensibility - Easy to understand?
- Ordered (Prioritized) rules - default at the bottom, common but not so easy to comprehend
- Unordered rules
  - Theoretically easier to understand, except must
  - Force consistency, or
  - Create a separate unordered list for each output class and use a tie-break scheme when multiple lists are matched

Sequential Covering Algorithms

- There are a number of rule learning algorithms based on different variations of sequential covering
  - CN2, AQx, etc.

1. Find a "good" rule for the current training set
2. Delete covered instances (or those covered correctly) from the training set
3. Go back to 1 until the training set is empty or until no more "good" rules can be found

Finding "Good" Rules

- The large majority of instances covered by the rule infer the same output class
- Rule covers as many instances as possible (general vs specific rules)
- Rule covers enough instances (statistically significant)
- Example rules and approaches?
- How to find good rules efficiently? - General to specific search is common
- Continuous features - some type of ranges/discretization

Common Rule "Goodness" Approaches

- Relative frequency: n_c/n
- m-estimate of accuracy (better when n is small): $\frac{n_c + mp}{n + m}$

  where p is the prior probability of a random instance having the output class of the proposed rule, penalizes rules with small n, Laplacian common: $(n_c+1)/(n+|C|)$ (i.e. $m = 1/p_c$)
- Entropy - Favors rules which cover a large number of examples from a single class, and few from others $\sum_{i=1}^{|C|} -p_i \log_2 p_i$
  - Entropy can be better than relative frequency
  - Improves consequent rule induction. R1:(.7,.1,.1,.1) R2 (.7,0.3,0) - entropy selects R2 which makes for better subsequent specializations during later rule growth
  - Empirically, rules of low entropy have higher significance than relative frequency, but Laplacian often better than entropy

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*

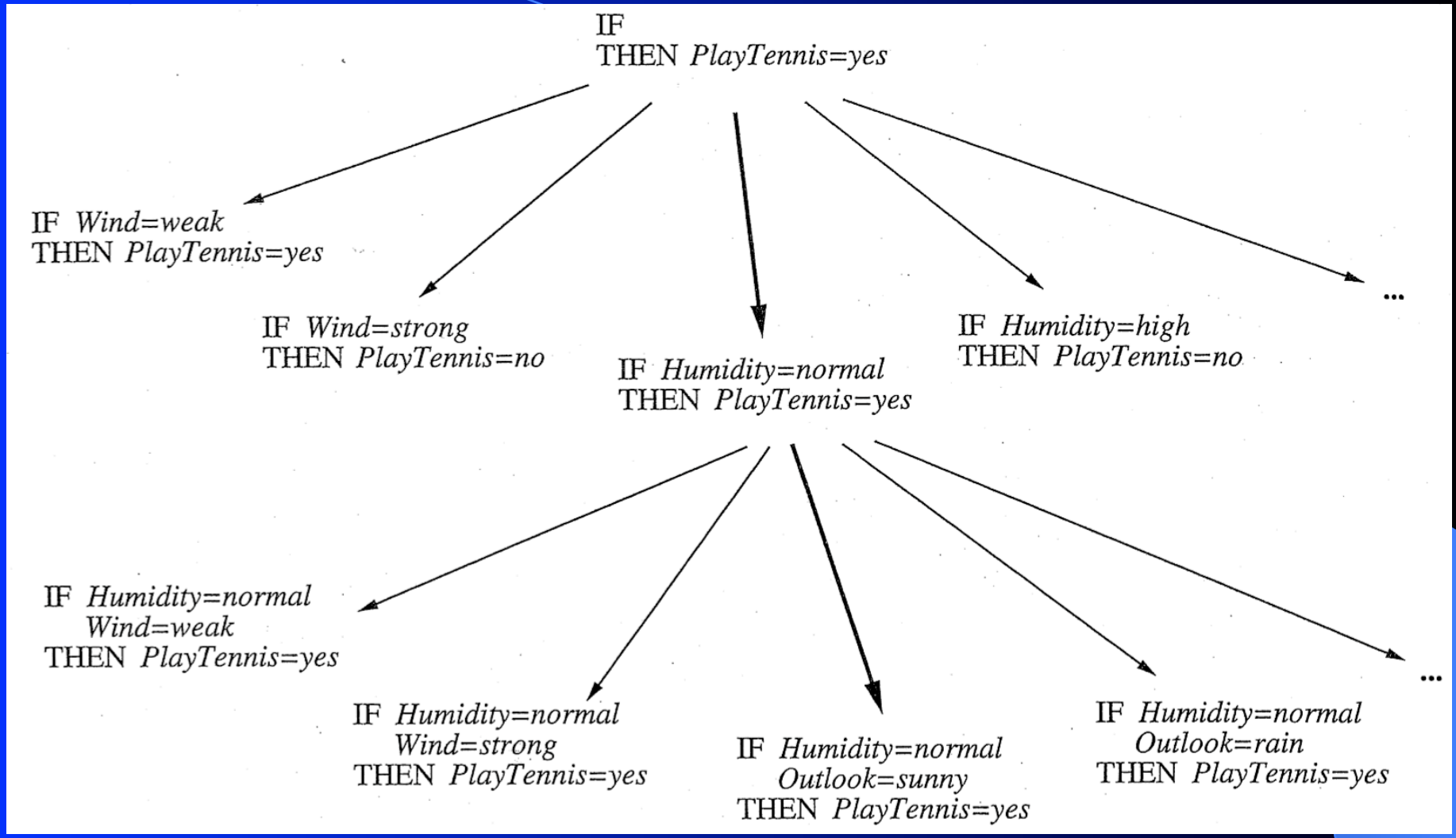
LEARN-ONE-RULE(*Target_attribute, Attributes, Examples, k*)

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*
     - *All_constraints* ← the set of all constraints of the form $(a = v)$, where a is a member of *Attributes*, and v is a value of a that occurs in the current set of *Examples*
     - *New_candidate_hypotheses* ←
       for each h in *Candidate_hypotheses*,
       for each c in *All_constraints*,
       - create a specialization of h by adding the constraint c
     - Remove from *New_candidate_hypotheses* any hypotheses that are duplicates, inconsistent, or not maximally specific
  2. *Update Best_hypothesis*
     - For all h in *New_candidate_hypotheses* do
       - If (PERFORMANCE(h, *Examples*, *Target_attribute*)
         > PERFORMANCE(*Best_hypothesis*, *Examples*, *Target_attribute*))
         Then *Best_hypothesis* ← h
  3. *Update Candidate_hypotheses*
     - *Candidate_hypotheses* ← the k best members of *New_candidate_hypotheses*, according to the PERFORMANCE measure.
- 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*.

PERFORMANCE(h, *Examples*, *Target_attribute*)

- *h_examples* ← the subset of *Examples* that match h
- return $-Entropy(h\_examples)$, where entropy is with respect to *Target_attribute*


IF
THEN PlayTennis=yes
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
...

Learning First Order Rules

- Inductive Logic Programming
- Propositional vs. first order rules
  - 1st order allows variables in rules
  - If Color of object1 = *x* and Color of object 2 = *x* then Class is A
  - More expressive
- FOIL - Uses a sequential covering approach from general to specific which allows the addition of literals with variables

RIPPER

```
procedure IREP(Pos,Neg)
begin
    Ruleset := ∅
    while Pos≠ ∅ do
        /* grow and prune a new rule */
        split (Pos,Neg) into (GrowPos,GrowNeg)
          and (PrunePos,PruneNeg)
        Rule := GrowRule(GrowPos,GrowNeg)
        Rule := PruneRule(Rule,PrunePos,PruneNeg)
        if the error rate of Rule on
          (PrunePos,PruneNeg) exceeds 50% then
            return Ruleset
        else
            add Rule to Ruleset
            remove examples covered by Rule
              from (Pos,Neg)
        endif
    endwhile
    return Ruleset
end
```

Figure 1: The IREP algorithm

Table 3. The CN2 induction algorithm.

```
Let E be a set of classified examples.
Let SELECTORS be the set of all possible selectors.

Procedure CN2(E)
  Let RULE_LIST be the empty list.
  Repeat until BEST_CPX is nil or E is empty:
     Let BEST_CPX be Find_Best_Complex(E).
     If BEST_CPX is not nil,
     Then let E' be the examples covered by BEST_CPX.
          Remove from E the examples E' covered by BEST_CPX.
          Let C be the most common class of examples in E'.
          Add the rule 'If BEST_CPX then the class is C'
              to the end of RULE_LIST.
  Return RULE_LIST.

Procedure Find_Best_Complex(E)
  Let STAR be the set containing the empty complex.
  Let BEST_CPX be nil.
  While STAR is not empty,
     Specialize all complexes in STAR as follows:
     Let NEWSTAR be the set {x ∧ y|x ∈ STAR, y ∈ SELECTORS}.
     Remove all complexes in NEWSTAR that are either in STAR (i.e.,
        the unspecialized ones) or null (e.g., big = y ∧ big = n).
     For every complex C_i in NEWSTAR:
          If C_i is statistically significant and better than
             BEST_CPX by user-defined criteria when tested on E,
          Then replace the current value of BEST_CPX by C_i.
     Repeat until size of NEWSTAR ≤ user-defined maximum:
        Remove the worst complex from NEWSTAR.
     Let STAR be NEWSTAR.
  Return BEST_CPX.
```

Insert Rules at Top or Bottom

- Typically would like focused exception rules higher and more general rules lower in the list
- Typical (CN2): Delete all instances covered by a rule during learning
  - Putting new rule on the bottom (i.e. early learned rules stay on top) makes sense since this rule is rated good only after removing all instances covered by previous rules, (i.e. instances which can get by the earlier rules).
  - Still should get exceptions up top and general rules lower in the list because exceptions will achieve a higher score and thus be added first (assuming statistical significance) than a general rule which has to cover more cases. Even though *E* keeps getting diminished there should still be enough data to support reasonable general rules later (in fact the general rules should get increasing scores after true exceptions are removed).
    - Highest scoring rules: Somewhat specific, high accuracy, sufficient coverage
    - Medium scoring rules: General and specific with reasonable accuracy and coverage
    - Low scoring rules: Specific with low coverage, and general with low accuracy

Rule Order - Continued

- If delete only correct instances covered by a rule
  - Putting new rule on the the top (i.e. first learned rule stays on bottom) could make sense because we could learn exception rules for those instances not covered by general rules at the bottom
  - This only works if the rule placed at the bottom is truly more general than the later rules (i.e. many novel instances will slide past the more exceptional rules and get covered by the general rules at the bottom)
- Sort after: (Mitchell) Proceed with care because rules were learned based on specific subsets of the training set
- Other variations possible, but many could be problematic because there are an exponential number of possible orderings
- Also can do unordered lists with tie-breaking mechanisms