Classifiers –II
(Rule Based Classifiers)

Using IF-THEN Rules for Classification

- Rules are good way for representing information or bits of knowledge (uses IF-THEN rules)

  R_1: IF age = youth AND student = yes THEN buys_computer = yes

  - **"IF Part" => Antecedent and "Then" =>consequent**
- Assessment of a rule: coverage and accuracy
  - n_{covers} = # of tuples covered by R_1
  - $n_{correct}$ = # of tuples correctly classified by R_1

  $coverage(R_1) = n_{covers} / |D|$ /* D: training data set */

  $accuracy(R_1) = n_{correct} / n_{covers}$

Rule Accuracy and Coverage

- Consider rule R1, which covers 2 of the 14 tuples.
- It can correctly classify both tuples.
  - coverage(R1) =2/14=14.28% and accuracy(R1) 2/2 =100%.
- Let X be tuple, if a rule R1 is statisfied by X, then rule is said to be tiggered.
- Eg:

  X=(age=youth,income=medium,student=yes,credit_rating=fair)
- If more than one rule are triggered, which class to specify?
- Solved using **conflict resolution strategy** to figure out which rules gets to fire and assign its class prediction to X.

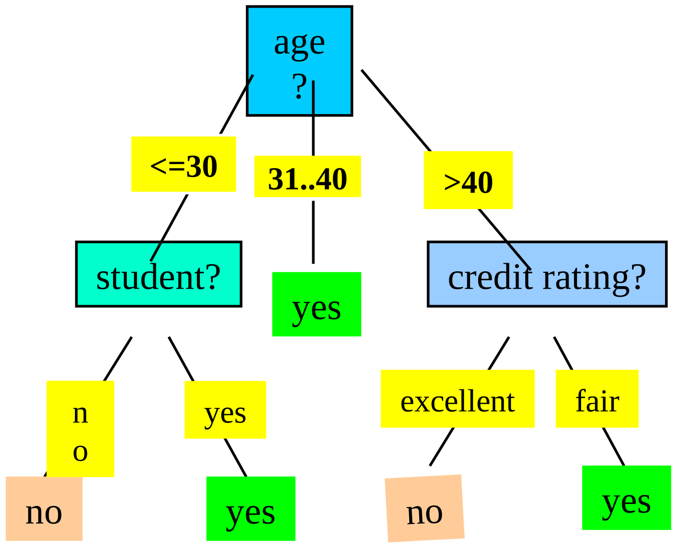
Using IF-THEN Rules for Classification

Resolution strategy: Based on size ordering or rule ordering(class-based, rule-based).

- **Size ordering:** Assign the highest priority to the rule with the most attribute tests
- **Class-based ordering:** The classes are sorted by decreasing order of prevalence (most frequent class) or misclassification cost per class.
- **Rule-based ordering (decision list):** Rules are organized into one long priority list, according to some measure of rule quality(accuracy,coverage,size attribute tests) or by experts
- Rule that appears in the list has the highest priority and it gets fired to get class prediction.
- When no rule is satisfied by X, then assign default rule to specify default class(majority class that were not covered by an rule) based on training set

Rule Extraction from a Decision Tree

- If decision trees become larger difficult to interpret.
- Using rule based classifier then rules are easier to understand than large trees.
- One rule is created for each path from the root to a leaf.
- Each splitting criterion along a give path is logically added to form rule antecedent and the leaf holds the class prediction.
- Rules are mutually exclusive(cannot have rule conflicts) and exhaustive (one rule for each possible attribute value)



Rule Extraction from a Decision Tree

- Example: Rule extraction from our buys_computer decision-tree

  IF age = young AND student = no THEN buys_computer = no

  IF age = young AND student = yes THEN buys_computer = yes

  IF age = mid-age THEN buys_computer = yes

  IF age = old AND credit_rating = excellent THEN buys_computer = no

  IF age = old AND credit_rating = fair THEN buys_computer = yes

Rule Extraction from a Decision Tree

- If the attribute tests may be irrelevant and redundant then the rules extracted can be difficult to follow.
- Some pruning is required.
- Prune the rule set which does not contribute to the overall accuracy.

Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data (without decision tree)
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Steps: Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - Add new rules to the rule-set
  - Repeat the process on the remaining tuples until termination condition, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold

Basic Sequential Covering Algorithm

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
Rule_set = {}; // initial set of rules learned is empty
for each class c do
    repeat
        Rule = Learn_One_Rule(D, Att_vals, c);
        remove tuples covered by Rule from D;
        Rule_set = Rule_set + Rule; // add new rule to rule set
    until terminating condition;
endfor
return Rule_Set;
```

Learn –One-Rule

- Learn_One_Rule procedure finds the “best” rule for the current class given the current set of training tuples.
- To learn a rule for the class
  - Start off with the most general rule with rule antecedent as empty
  - Adding new attributes by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
- Performs beam search to maintain k best candidates using **measures of rule quality**

Learn –One-Rule

- Rule R1 correctly classifies 38 of the 40 tuples it covers. Rule R2 covers only two tuples, which it correctly classifies.
- Their respective accuracies are 95% and 100%.
- R2 has greater accuracy than R1, but it is not the better rule because of its **small coverage.**
- Evaluate rule using other measures: “**entropy” , statistical test considers coverage, FOIL (First Order Inductive Learner) measured based on information gain.**

Sequential Covering Algorithm

while (enough target tuples left)

generate a rule

remove positive target tuples satisfying this rule

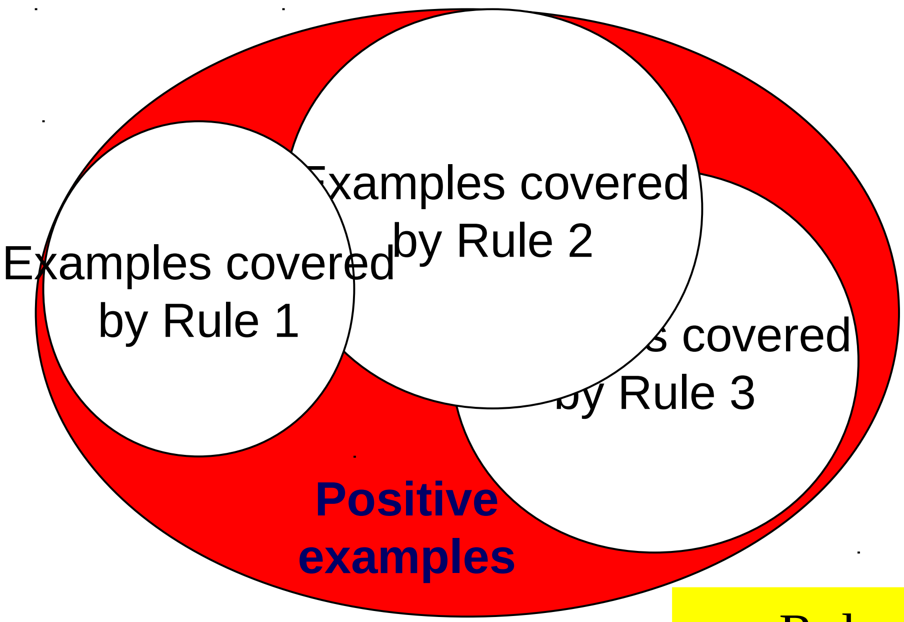
- To generate a rule

while(true)

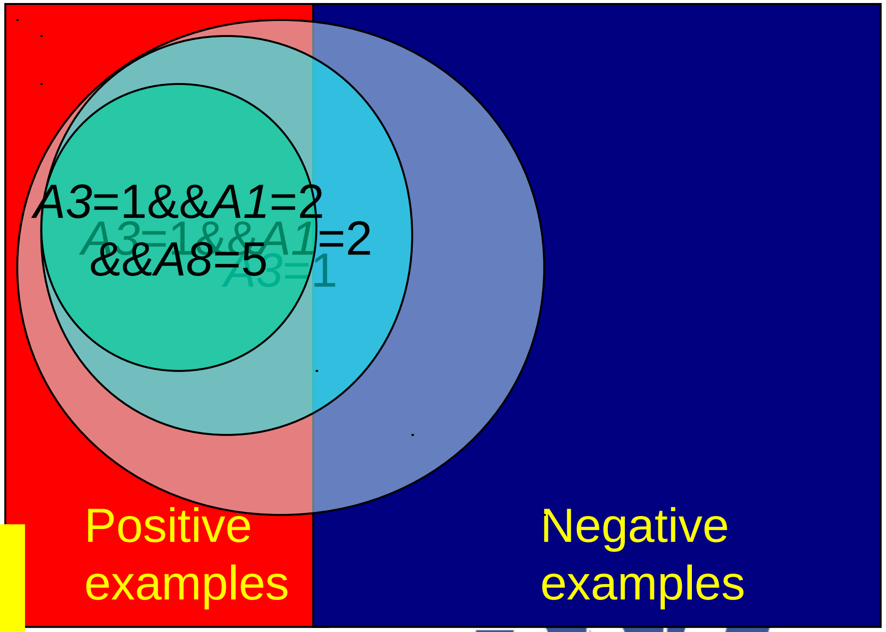
find the best predicate p

if foil-gain(p) > threshold then add p to current rule

else break



Rule Generation



How to Learn-One-Rule?

Foil-gain (in FOIL & RIPPER): assesses info_gain by extending condition

- favors rules that have high accuracy and cover many positive tuples

$$FOIL_{Gain} = pos' \times \left(\log 2 \frac{pos'}{pos' + neg'} - \log 2 \frac{pos}{pos + neg}\right)$$

- Pos/neg are # of positive/negative tuples covered by R
- Pos'/neg' are # of positive/negative tuples covered by R' (new condition)

- Rule pruning based on an independent set of test tuples

  If FOIL_Prune is higher for the pruned version of R, prune R

$$FOIL_{Prune}(R) = \frac{pos - neg}{pos + neg}$$

Classification using Frequent Patterns

- We can examine how association rules generated are used for classification

Associative classification methods are- Classification Based on Associations (CBA), Classification Based on Multiple Association Rules (CMAR)

- General steps:
  - **Mine the data for frequent item sets**, that is, find commonly occurring attribute–value pairs in the data.
  - Analyze the frequent item sets to **generate association rules** per class, which satisfy **confidence and support criteria.**
  - Organize the rules to form a rule-based classifier.

CBA: Classification Based on Associations

- Mine high-confidence, high-support class association rules
- LHS: conjunctions of attribute-value pairs; RHS: class labels
- p_1 ^ p_2 … ^ p_l "$A_{class\text{-}label}$ = C" (confidence, support)
  - Rank rules in descending order of confidence and support
- Classification: Apply the first rule that matches a test case; o.w. apply the default rule
- Effectiveness: Often found more accurate than some traditional classification methods, such as C4.5
- Why? — Exploring high confident associations among multiple attributes may overcome some constraints introduced by some classifiers that consider only one attribute at a time

CMAR: Classification Based on Multiple Association Rules

- CMAR ADOPTS VARIANT fp-growth algorithm to find the complete set of rules satisfying minimum confidence and support
- The enchanced FP-Tree maintains class labels among the tuples satisfying frequent itemset.
- Prune rules based on confidence, correlation and database coverage.

CMAR: Classification Based on Multiple Association Rules

- Classification based on generated/pruned rules
  - If only one rule satisfies tuple X, assign the class label of the rule
  - If a rule set S satisfies X
    - Divide S into groups according to class labels
    - Use a weighted χ^2 measure to find the strongest group of rules, based on the statistical correlation of rules within a group
    - Assign X the class label of the strongest group
- CMAR improves model construction efficiency and classification accuracy