Learning Rules
Rule-Based Classification

Rule-Based Classifier
Using IF-THEN Rules for Classification

- Classify tuples by using a collection of "if…then…" rules.
  - Represent the knowledge in the form of IF-THEN rules

Rule: IF **(Condition)** THEN **Consequent**

- where
  - **Condition** is a conjunction of attributes (**rule antecedent** or **condition**)
  - **Consequent** is a class label (**rule consequent**)

- Examples of classification rules:

IF **(*age*=youth AND *student*=yes)** THEN ***buys_computer = yes***

IF **(*BloodType*=warm AND *LayEggs*=yes)** THEN ***class = bird***

Rule-Based Classifier: Rule Coverage and Accuracy

- A rule r **covers** an instance x if the attributes of the instance satisfy the condition of the rule.

Coverage of a rule:

- Fraction of tuples that satisfy the condition of a rule.

Accuracy of a rule:

- Fraction of tuples that satisfy the condition that also satisfy the consequent of a rule.

$\mathbf{n_{covers}}$: # of tuples covered by rule R

$\mathbf{n_{correct}}$: # of tuples correctly classified by rule R

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

$\mathbf{accuracy(R) = n_{correct} / n_{covers}}$

Rule-Based Classifier: Rule Coverage and Accuracy - Example

IF **(Status=Single)** THEN **Class=No**

Coverage = 4/10 = 40%

Accuracy = 2/4 = 50%

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	**Single**	125K	**No**
2	No	Married	100K	**No**
3	No	**Single**	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	**Single**	85K	**Yes**
9	No	Married	75K	**No**
10	No	**Single**	90K	**Yes**

Rule-Based Classifier - Example

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

R1: IF (Give Birth = no) AND (Can Fly = yes) THEN Class=Birds
R2: IF (Give Birth = no) AND (Live in Water = yes) THEN Class=Fishes
R3: IF (Give Birth = yes) AND (Blood Type = warm) THEN Class=Mammals
R4: IF (Give Birth = no) AND (Can Fly = no) THEN Class=Reptiles
R5: IF (Live in Water = sometimes) THEN Class=Amphibians

How a Rule-Based Classifier Works?

- **A rule-based classifier classifies a tuple based on the rule triggered by the tuple.**

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
lemur	warm	yes	no	no	?
turtle	cold	no	no	sometimes	?
dogfish shark	cold	yes	no	yes	?

R1: IF (Give Birth = no) AND (Can Fly = yes) THEN Class=Birds
R2: IF (Give Birth = no) AND (Live in Water = yes) THEN Class=Fishes
R3: IF (Give Birth = yes) AND (Blood Type = warm) THEN Class=Mammals
R4: IF (Give Birth = no) AND (Can Fly = no) THEN Class=Reptiles
R5: IF (Live in Water = sometimes) THEN Class=Amphibians

- A **lemur** triggers rule R3, so it is classified as a **mammal**
- A **turtle** triggers both R4 and R5
  - Since the classes predicted by the rules are contradictory (**reptiles** versus **amphibians**), their conflicting classes must be resolved.
- A **dogfish shark** triggers none of the rules
  - we need to ensure that the classifier can still make a reliable prediction even though a tuple is not covered by any rule.

Characteristics of Rule Sets

Mutually Exclusive Rules

- The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same tuple.
- Every tuple is covered by at most one rule in R.

Exhaustive Rules

- A rule set R has exhaustive coverage if there is a rule for each combination of attribute values.
- Every record is covered by at least one rule in R.

Characteristics of Rule Sets

Rules are not mutually exclusive:

- A tuple may trigger more than one rule
- ***Solution 1: Use ordered rule set***
  - The rules in a rule set are ordered in decreasing order of their priority (e.g., based on accuracy, coverage, or the order in which the rules are generated).
  - An ordered rule set is also known as a **decision list**.
  - When a tuple is presented, it is classified by the **highest-ranked rule** that covers the tuple.
- ***Solution 2: Use unordered rule set and a voting scheme***
  - A tuple triggers multiple rules and considers the consequent of each rule as a vote for a particular class.
  - The tuple is usually assigned to the class that receives the highest number of votes. In some cases, the vote may be weighted by the rule's accuracy

Characteristics of Rule Sets

Rules are not exhaustive

- A tuple may not trigger any rules
- ***Solution: Use a default class***
  - If the rule set is not exhaustive, then a default rule must be added to cover the remaining cases.

    DefaultRule: IF () THEN Class = ***defaultclass***
  - A default rule has an empty antecedent (TRUE) and is triggered when all other rules have failed.
  - ***defaultclass*** is known as the default class and is typically assigned to the majority class of training records not covered by the existing rules.

Rule Ordering Schemes

- Rule ordering can be implemented on a rule-by-rule basis or on a class-by-class basis.

Rule-Based Ordering Scheme

- Individual rules are ranked based on their quality.
  - Every tuple is classified by the "best" rule covering it.
  - Lower-ranked rules are much harder to interpret because they assume the negation of the rules preceding them.

Class-Based Ordering Scheme

- Rules that belong to the same class appear together in the rule set.
- The rules are then collectively sorted on the basis of their class information.
  - The relative ordering among the rules from the same class is not important; as long as one of the rules fires, the class will be assigned to the test tuple.
  - A high-quality rule to be overlooked in favor of an inferior rule that happens to predict the higher-ranked class.

Rule Ordering Schemes

Rule-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes) ⇒ Birds
(Body temperature=warm-blooded, Gives Birth=yes) ⇒ Mammals
(Body temperature=warm-blooded, Gives Birth=no) ⇒ Birds
(Aquatic Creature=semi) ⇒ Amphibians
(Skin Cover=scales, Aquatic Creature=no) ⇒ Reptiles
(Skin Cover=scales, Aquatic Creature=yes) ⇒ Fishes
(Skin Cover=none) ⇒ Amphibians

Class-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes) ⇒ Birds
(Body temperature=warm-blooded, Gives Birth=no) ⇒ Birds
(Body temperature=warm-blooded, Gives Birth=yes) ⇒ Mammals
(Aquatic Creature=semi) ⇒ Amphibians
(Skin Cover=none) ⇒ Amphibians
(Skin Cover=scales, Aquatic Creature=no) ⇒ Reptiles
(Skin Cover=scales, Aquatic Creature=yes ⇒ Fishes

Building Classification Rules

- To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a data set and the class label.

Direct Methods:

- Extract classification rules directly from data.
- Examples: RIPPER, CN2, …

Indirect Methods:

- Extract rules from other classification models (e.g. decision trees, etc).
- Examples: C4.5rules

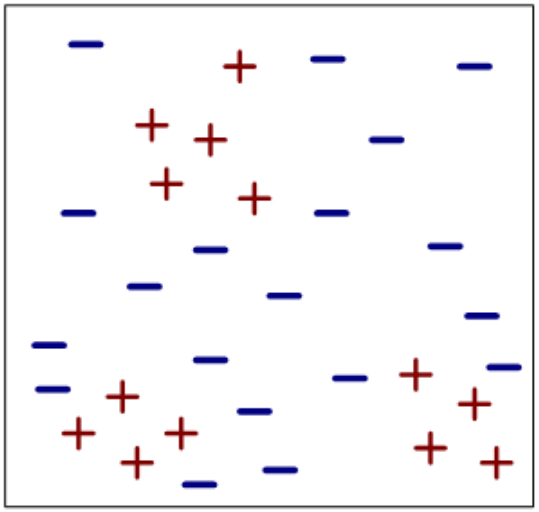
Direct Method: Sequential Covering

- **Sequential covering algorithm** extracts rules directly from training data
- Typical sequential covering algorithms: RIPPER, FOIL, CN2,
- Rules are learned sequentially, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes.

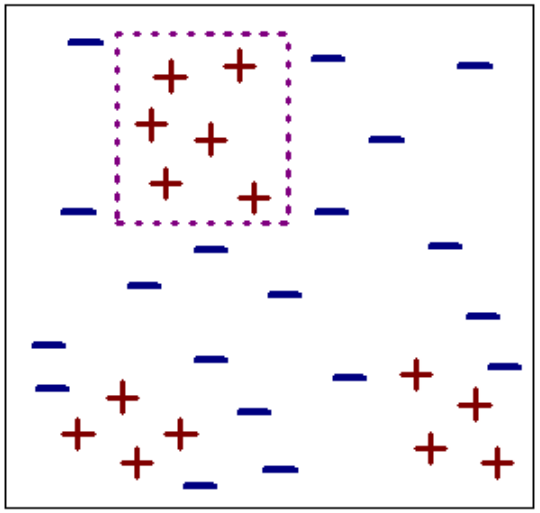
Sequential Covering Algorithm

1: Let E be the training records and A be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
2: Let Y_o be an ordered set of classes $\{y_1, y_2, \ldots, y_k\}$.
3: Let $R = \{\ \}$ be the initial rule list.
4: **for** each class $y \in Y_o - \{y_k\}$ **do**
5: **while** stopping condition is not met **do**
6: $r \leftarrow$ Learn-One-Rule (E, A, y).
7: Remove training records from E that are covered by r.
8: Add r to the bottom of the rule list: $R \longrightarrow R \vee r$.
9: **end while**
10: **end for**
11: Insert the default rule, $\{\} \longrightarrow y_k$, to the bottom of the rule list R.

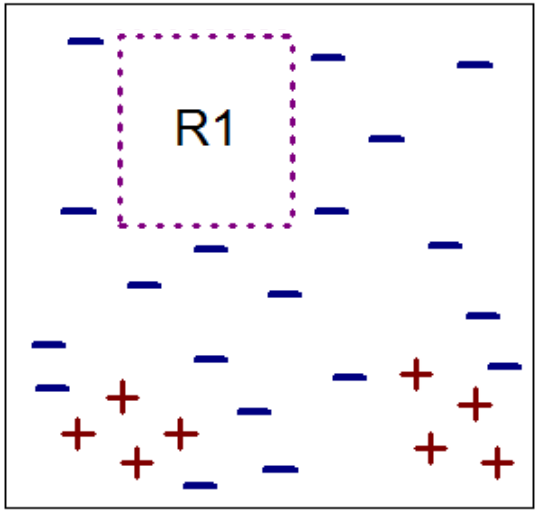
Sequential Covering - Example



original data set



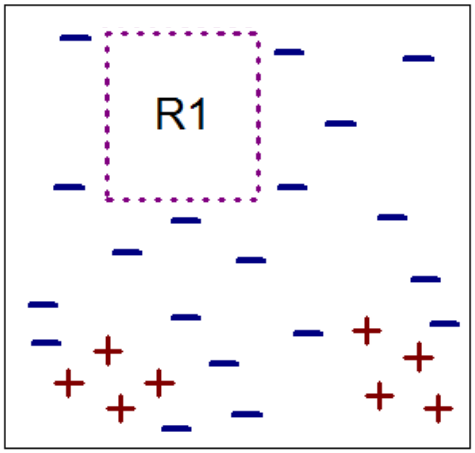
tuples covered by R1

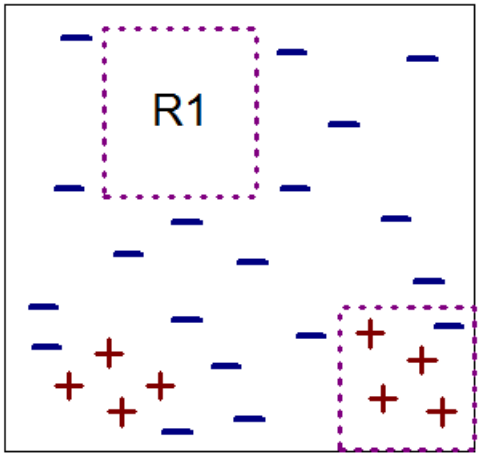


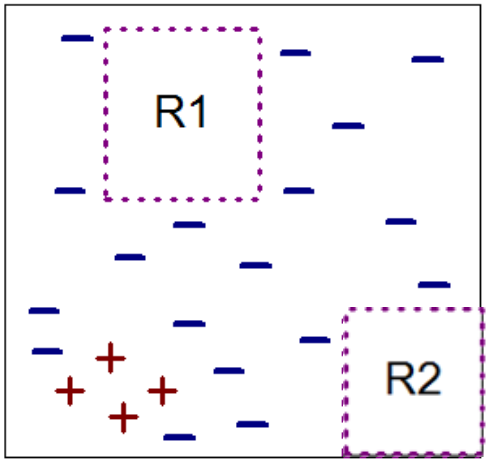
remove tuples covered by R1

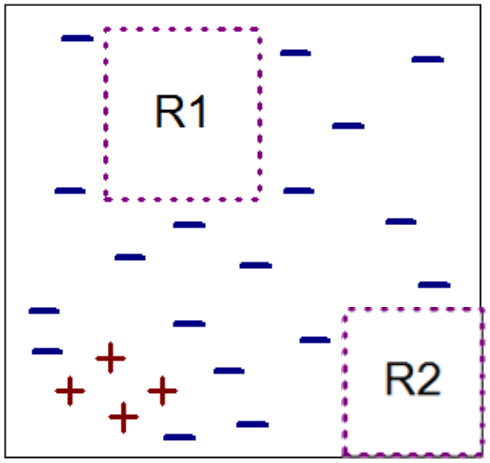
- Find the first rule R1 and remove the tuples covered by R1 from the training data set.
- Add the rule to the rule list.

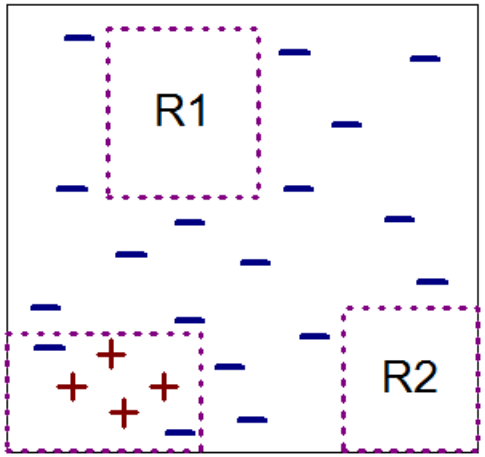
Sequential Covering - Example

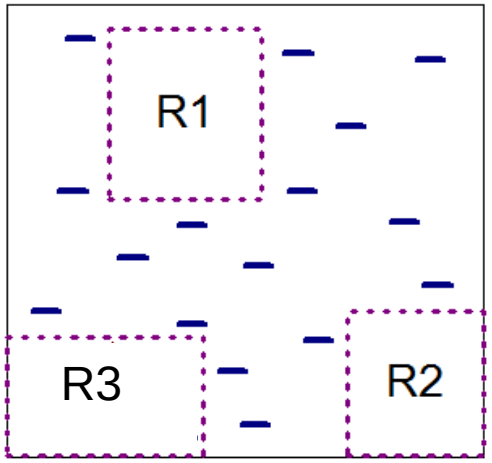












Sequential Covering
How to Learn-One-Rule?

- The objective of Learn-One-Rule function is to extract a classification rule that covers many of the positive examples and none (or very few) of the negative examples in the training set.
- Finding an optimal rule is computationally expensive given the exponential size of the search space.
- Learn-One-Rule function addresses the exponential search problem by growing the rules in a greedy fashion.
  - It generates an initial rule r and keeps refining the rule until a certain stopping criterion is met.
  - The rule is then pruned to improve its generalization error.

Learn-One-Rule: Rule-Growing Strategy

- Two common ***rule-growing strategies***: **general-to-specific** or **specific-to-general**.

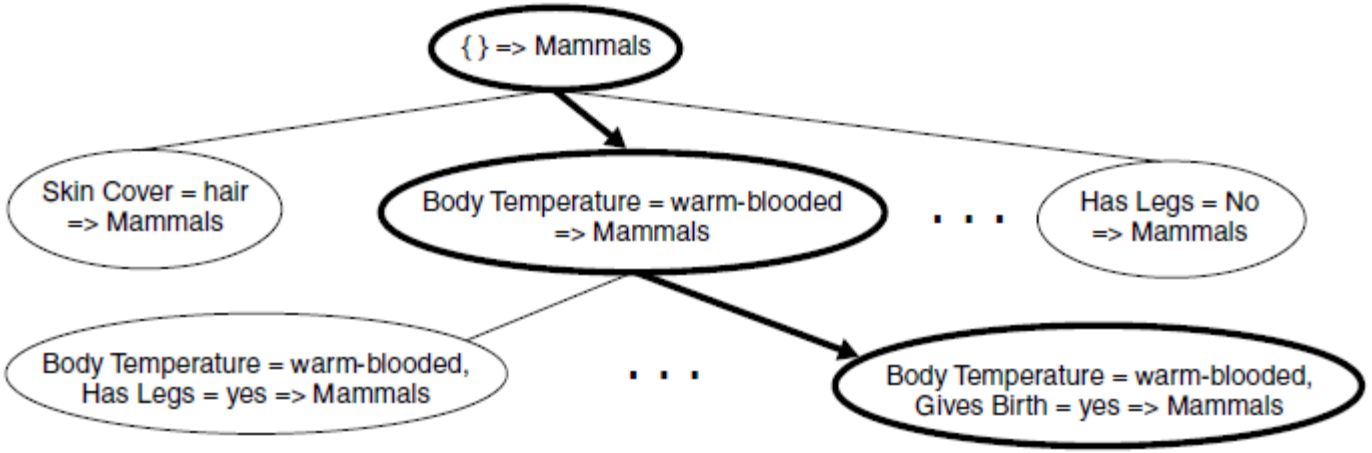
General-to-specific:

- An initial rule r : {} → y is created, where the left-hand side is an empty set and the right-hand side contains the target class.
  - The rule has poor quality because it covers all the examples in the training set.
- New conjuncts are subsequently added to improve the rule's quality.
  - Algorithm then explores all possible candidates and greedily chooses the next conjunct.
  - Process continues until the stopping criterion is met (e.g., when the added conjunct does not improve the quality of the rule).

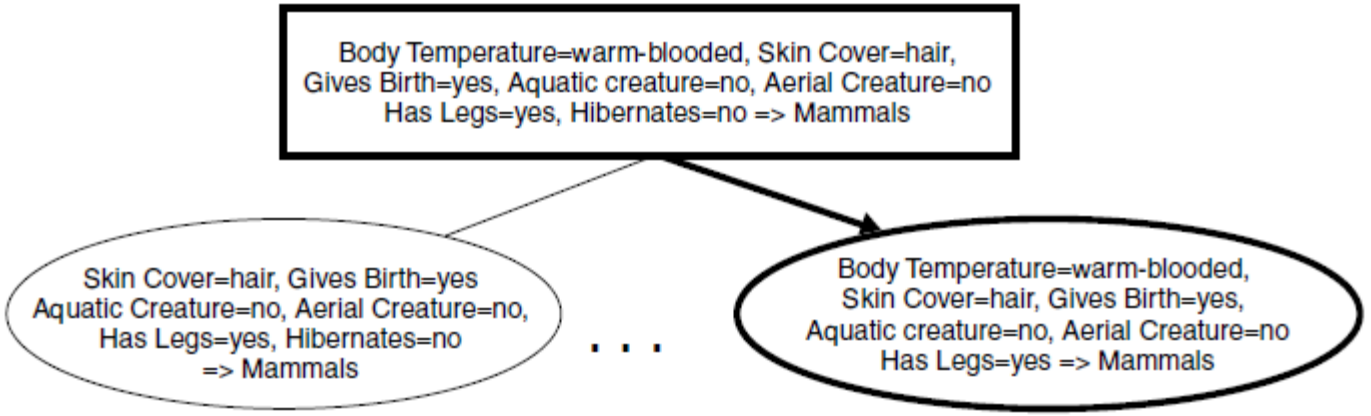
Specific-to-general

- One of the positive examples is randomly chosen as the initial seed for the rule-growing process.
- During the refinement step, the rule is generalized by removing one of its conjuncts so that it can cover more positive examples.

Learn-One-Rule: Rule-Growing Strategy - Example



General-to-specific



Specific-to-general

Learn-One-Rule: Rule Evaluation

- An **evaluation metric** is needed to determine which conjunct should be added (or removed) during the rule-growing process.
  - **Accuracy** is an obvious choice because it explicitly measures the fraction of training examples classified correctly by the rule.
  - A potential limitation of accuracy is that it does not take into account the ***rule's coverage***.
    - Training data set has: 60 positive examples, 100 negative examples
    - R1: covers 50 positive examples and 5 negative examples
    - R2: covers 2 positive examples and no negative examples.
    - Although the accuracy of R2 (100%) is higher than the accuracy of R1 (90.9%), R1 is a better rule because of the coverage of the rule.
- We need evaluation metrics take into account the ***rule's coverage:***
  - **Foil's Information Gain,** and other metrics.

Learn-One-Rule: Rule Evaluation

- **Foil's Information Gain** is a ***rule-quality measure*** which considers both coverage and accuracy.

R0: {A} $\Rightarrow$ class (initial rule)

R1: {A and B} $\Rightarrow$ class (rule after adding conjunct B)

$$\textbf{Foil's Information Gain(R0, R1)} = \mathbf{p1} \times \left(\mathbf{log_2} \frac{\mathbf{p1}}{\mathbf{p1+n1}} - \mathbf{log_2} \frac{\mathbf{p0}}{\mathbf{p0+n0}} \right)$$

where

p0: number of positive instances covered by R0

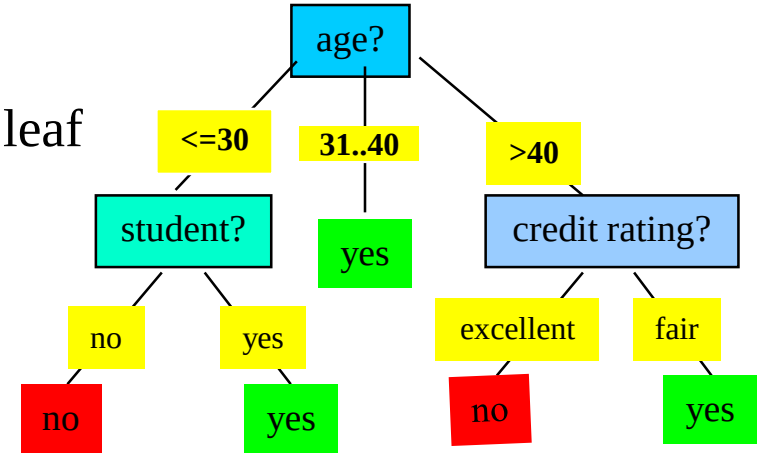
n0: number of negative instances covered by R0

p1: number of positive instances covered by R1

n1: number of negative instances covered by R1

Indirect Method for Rule Extraction
Rule Extraction from a Decision Tree

- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive



- 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*

Indirect Method for Rule Extraction
C4.5 Rules

- Extract rules from an unpruned decision tree
- For each rule, r: A $\rightarrow$ y,
  - Consider an alternative rule r′: A′ $\rightarrow$ y where A′ is obtained by removing one of the conjuncts in A
  - Compare the pessimistic error rate for r against all r's
  - Prune if one of the alternative rules has lower pessimistic error rate
  - Repeat until we can no longer improve generalization error
- Instead of ordering the rules, order subsets of rules (class ordering)
  - Each subset is a collection of rules with the same rule consequent (class)

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.

- Inductive Logic Programming deals with learning of first order rules.