Classification

Part 2

Dr. Sanjay Ranka
Professor
Computer and Information Science and Engineering
University of Florida, Gainesville

Overview

- Rule based Classifiers
- Nearest-neighbor Classifiers

Rule Based Classifiers

- Classify instances by using a collection of "if … then …" rules
- Rules are presented in Disjunctive Normal Form, $R = (r_1 \text{ v } r_2 \text{ v } \ldots r_k)$
- R is called *rule set*
- r_i's are called *classification rules*
- Each classification rule is of form
  - $r_i : (Condition_i) \rightarrow y$
    - Condition is a conjunction of attribute tests
    - y is the class label

Rule Based Classifiers

- r_i : $(Condition_i) \rightarrow y$
  - LHS of the rule is called *rule antecedent* or *pre-condition*
  - RHS is called the *rule consequent*
- If the attributes of an instance satisfy the pre-condition of a rule, then the instance is assigned to the class designated by the rule consequent
- Example
  - (Blood Type=Warm)∧ (Lay Eggs=Yes) → Birds
  - (Taxable Income < 50K) ∧ (Refund=Yes) → Cheat=No

Classifying Instances with Rules

- A rule r covers an instance x if the attributes of the instance satisfy the condition of the rule
- Rule:

  r : (Age < 35) $\wedge$ (Status = Married) $\rightarrow$ Cheat=No
- Instances:

  x_1 : (Age=29, Status=Married, Refund=No)

  x_2 : (Age=28, Status=Single, Refund=Yes)

  x_3 : (Age=38, Status=Divorced, Refund=No)
- Only x_1 is covered by the rule r

Rule Based Classifiers

- Rules may not be mutually exclusive
  - More than one rule may cover the same instance
- Strategies:
  - Strict enforcement of mutual exclusiveness
    - Avoid generating rules that have overlapping coverage with previously selected rules
  - Ordered rules
    - Rules are rank ordered according to their priority
  - Voting
    - Allow an instance to trigger multiple rules, and consider the consequent of each triggered rule as a vote for that particular class

Rule Based Classifiers

- Rules may not be exhaustive
- Strategy:
  - A *default rule* $r_d : ( ) \rightarrow y_d$ can be added
  - The default rule has an empty antecedent and is applicable when all other rules have failed
  - y_d is known as *default class* and is often assigned to the majority class

Example of Rule Based Classifier

Tid	Refund (categorical)	Marital Status (categorical)	Taxable Income (continuous)	Evade (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	Singl e	90K	Yes

- r_1: (Refund=No) & (Marital Status=Single) & (Taxable Income>80K) → Yes
- r_2: (Refund=No) & (Marital Status=Divorced) & (Taxable Income>80K) → Yes
- *default*: ( ) → No

Advantages of Rule Based Classifiers

- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

Basic Definitions

- Coverage of a rule:
  - Fraction of instances that satisfy the antecedent of a rule
- Accuracy of a rule:
  - Fraction of instances that satisfy both the antecedent and consequent of a rule

Tid	Refund	Marital Status	Taxable Income	Cheat
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

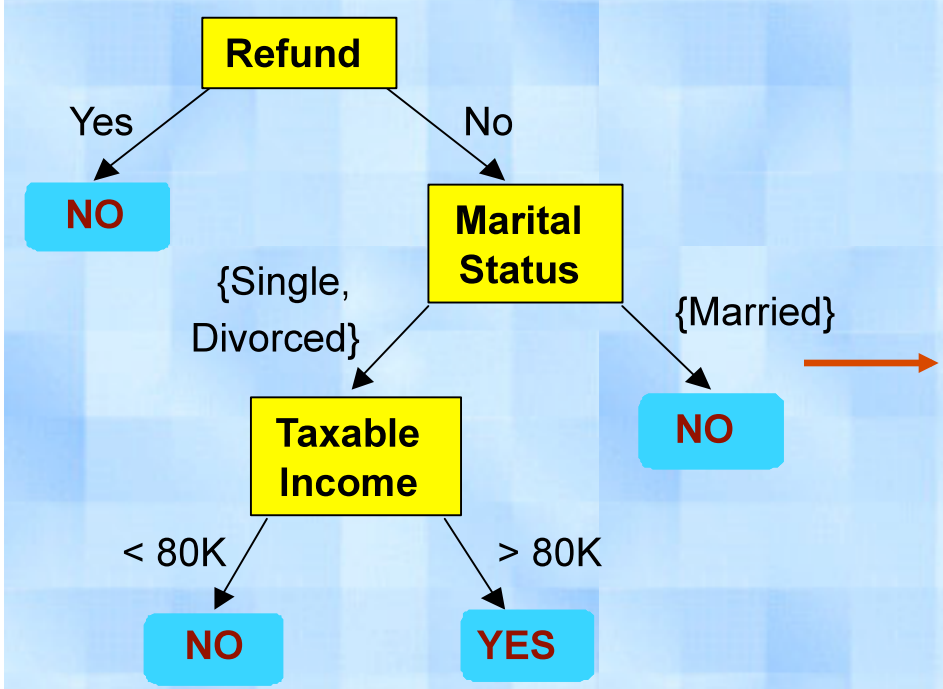
(Marital Status=Married) → No

Coverage = 40%, Accuracy = 100%

How to Build a Rule Based Classifier

- Generate an initial set of rules
  - Direct Method:
    - Extract rules directly from data
    - Examples: RIPPER, CN2, Holte's 1R
  - Indirect Method:
    - Extract rules from other classification methods (e.g. decision trees)
    - Example: C4.5 rules
- Rules are pruned and simplified
- Rules can be order to obtain a rule set *R*
- Rule set *R* can be further optimized

Indirect Method: From Decision Trees to Rules



Classification Rules

(Refund=Yes) ==> No

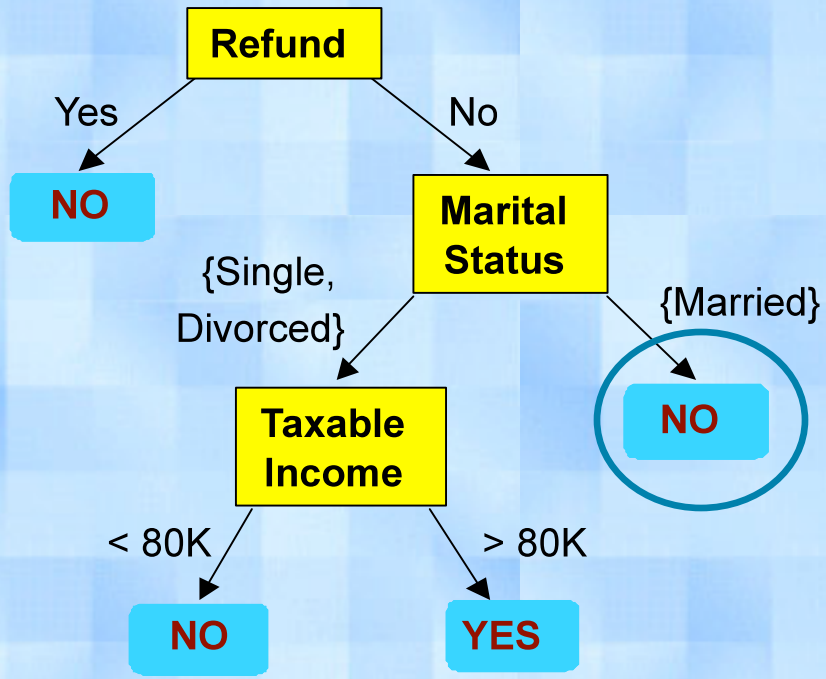
(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

- Rules are mutually exclusive and exhaustive
- Rule set contains as much information as the tree

Rules can be Simplified



Tid	Refund	Marital Status	Taxable Income	Cheat
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

- Initial Rule: (Refund=No) ∧ (Status=Married) → No
- Simplified Rule: (Status=Married) → No

Indirect Method: C4.5 rules

- Creating an initial set of rules
  - Extract rules from an un-pruned 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 r's has a lower pessimistic error rate
    - Repeat until we can no longer improve the generalization error

Indirect Method: C4.5 rules

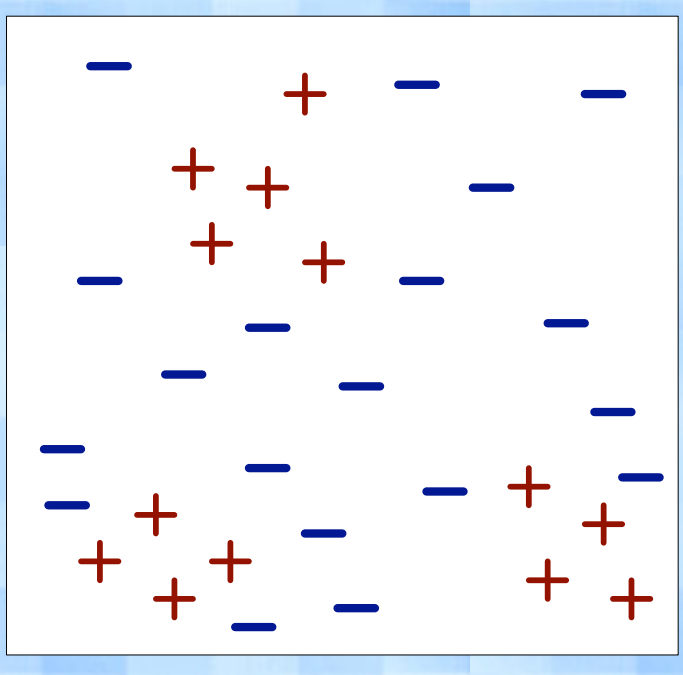
- Ordering the rules
  - Instead of ordering the rules, order subsets of rules
    - Each subset is a collection of rules with the same consequent (class)
    - Description length of each subset is computed, and the subsets are then ordered in the increasing order of the description length
      - Description length = L(exceptions) + $g \cdot L$(model)
      - g is a parameter that takes in to account the presence of redundant attributes in a rule set. Default value is 0.5

Direct Method: Sequential Covering

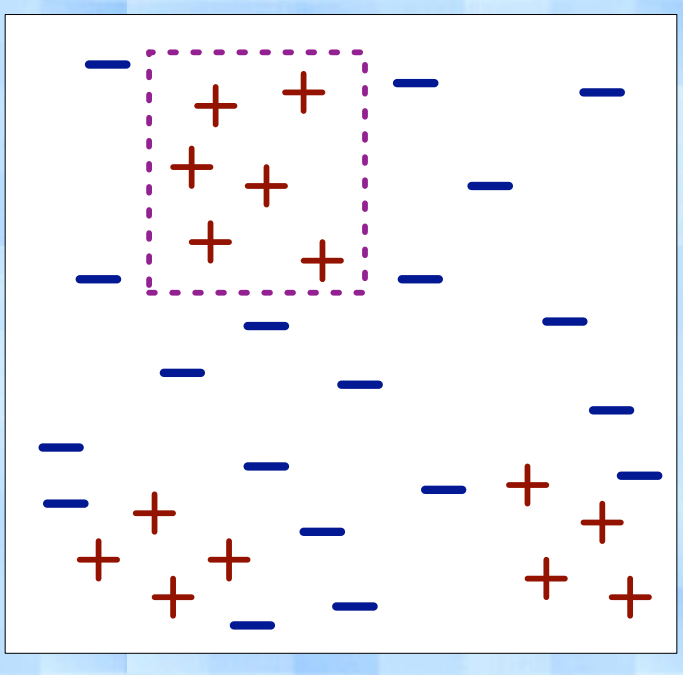
- Sequential Covering Algorithm (E: training examples, A: set of attributes)
  1. Let R = { } be the initial rule set
  2. While stopping criteria is not met
     1. $r \leftarrow$ Learn-One-Rule (E, A)
     2. Remove instances from E that are covered by r
     3. Add r to rule set: $R = R \vee r$

- Example of stopping criteria: Stop when all instances belong to same class or all attributes have same value

Example of Sequential Covering

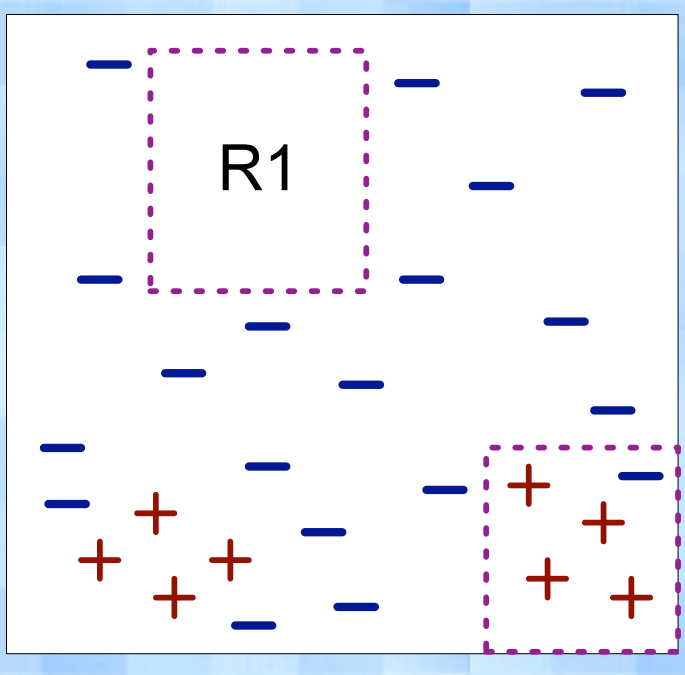


(i) Original Data

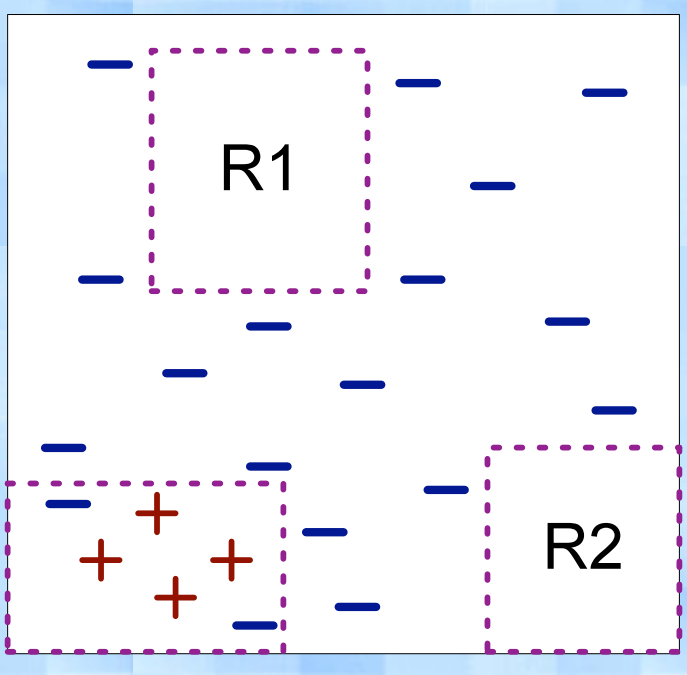


(ii) Step 1

Example of Sequential Covering



(iii) Step 2



(iv) Step 3

Learn One Rule

- The objective of this function is to extract the best rule that covers the current set of training instances
  - What is the strategy used for rule growing
  - What is the evaluation criteria used for rule growing
  - What is the stopping criteria for rule growing
  - What is the pruning criteria for generalizing the rule

Learn One Rule

- Rule Growing Strategy
  - General-to-specific approach
    - It is initially assumed that the best rule is the empty rule, r: $\{\} \rightarrow y$, where y is the majority class of the instances
    - Iteratively add new conjuncts to the LHS of the rule until the stopping criterion is met
  - Specific-to-general approach
    - A positive instance is chosen as the initial seed for a rule
    - The function keeps refining this rule by generalizing the conjuncts until the stopping criterion is met

Learn One Rule

- Rule Evaluation and Stopping Criteria
  - Evaluate rules using rule evaluation metric
    - Accuracy
    - Coverage
    - Entropy
    - Laplace
    - M-estimate
  - A typical condition for terminating the rule growing process is to compare the evaluation metric of the previous candidate rule to the newly grown rule

Learn One Rule

- Rule Pruning
  - Each extracted rule can be pruned to improve their ability to generalize beyond the training instances
  - Pruning can be done by removing one of the conjuncts of the rule and then testing it against a validation set
- Instance Elimination
  - Instance elimination prevents the same rule from being generated again
  - Positive instances must be removed after each rule is extracted
  - Some rule based classifiers keep negative instances, while some remove them prior to generating next rule

Direct Method: Sequential Covering

- Use a general-to-specific search strategy
- Greedy approach
- Unlike decision tree (which uses simultaneous covering), it does not explore all possible paths
  - Search only the current best path
  - Beam search: maintain *k* of the best paths
- At each step,
  - Decision tree chooses among several alternative attributes for splitting
  - Sequential covering chooses among alternative attribute-value pairs

Direct Method: RIPPER

- For 2-class problem, choose one of the classes as positive class, and the other as negative class
  - Learn rules for positive class
  - Negative class will be default class
- For multi-class problem
  - Order the classes according to increasing class prevalence (fraction of instances that belong to a particular class)
  - Learn the rule set for smallest class first, treat the rest as negative class
  - Repeat with next smallest class as positive class

Foil's Information Gain

- Compares the performance of a rule before and after adding a new conjunct
- Suppose the rule r: $A \rightarrow y$ covers p_0 positive and n_0 negative instances
- After adding a new conjunct B, the rule r': $A \wedge B \rightarrow y$ covers p_1 positive and n_1 negative instances
- Foil's information gain is defined as

$$= t \cdot [\log_2(p_1/(p_1 + n_1)) - \log_2(p_0/(p_0 + n_0))]$$

where t is the number of positive instances covered by both r and r'

Direct Method: RIPPER

- Growing a rule:
  - Start from empty rule
  - Add conjuncts as long as they improve Foil's information gain
  - Stop when rule no longer covers negative examples
  - Prune the rule immediately using incremental reduced error pruning
  - Measure for pruning: $v = (p - n) / (p + n)$
    - p: number of positive examples covered by the rule in the validation set
    - n: number of negative examples covered by the rule in the validation set
  - Pruning method: delete any final sequence of conditions that maximizes v

Direct Method: RIPPER

- Building a Rule Set:
  - Use sequential covering algorithm
    - Finds the best rule that covers the current set of positive examples
    - Eliminate both positive and negative examples covered by the rule
  - Each time a rule is added to the rule set, compute the description length
    - Stop adding new rules when the new description length is *d* bits longer than the smallest description length obtained so far. *d* is often chosen as 64 bits

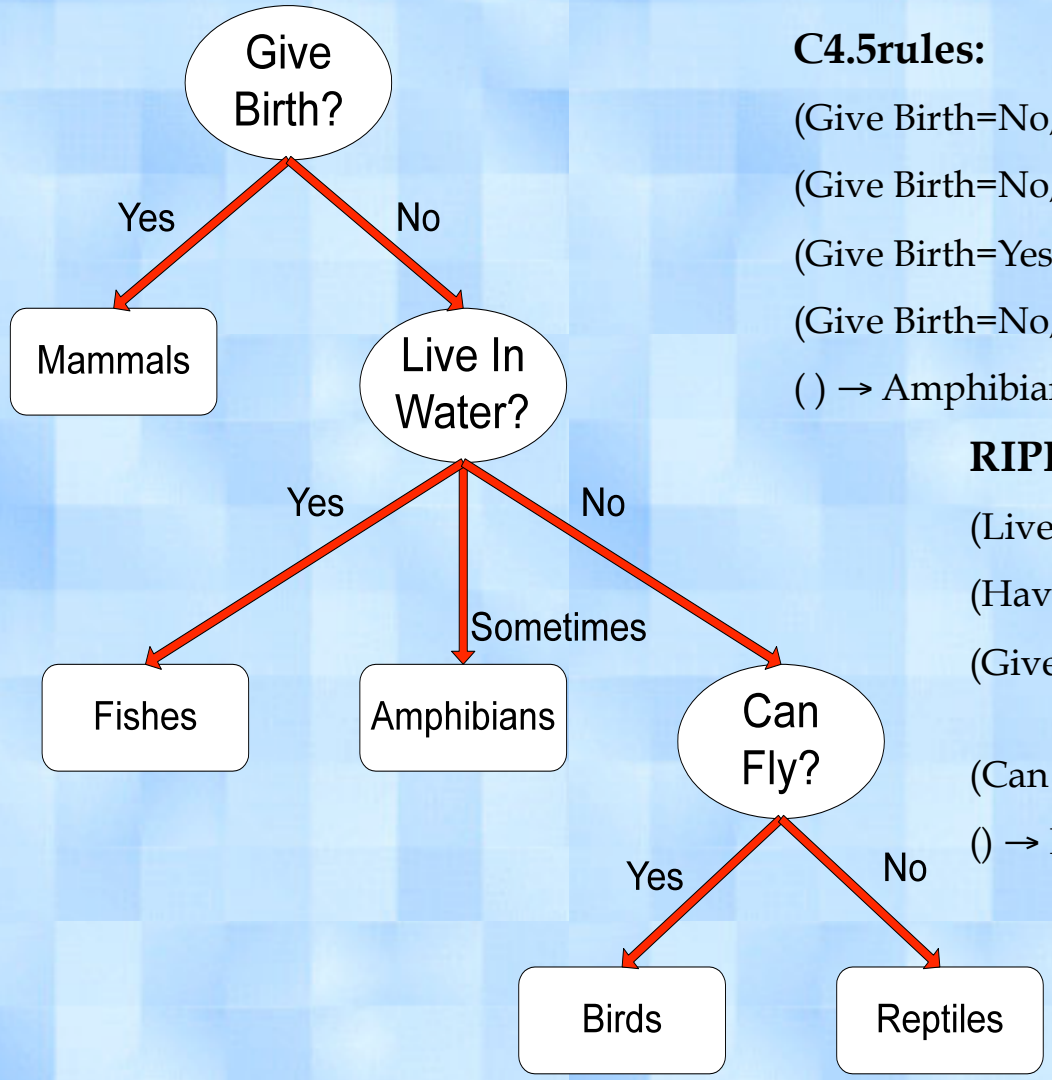
Direct Method: RIPPER

- Optimize the rule set:
  - For each rule *r* in the rule set ***R***
    - Consider 2 alternative rules:
      - Replacement rule (*r**): grow new rule from scratch
      - Revised rule (*r′*): add conjuncts to extend the rule *r*
    - Compare the rule set for *r* against the rule set for *r** and *r′*
    - Choose rule set that minimizes MDL principle
  - Repeat rule generation and rule optimization for the remaining positive examples

Example

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

C4.5 rules versus RIPPER



C4.5rules:

(Give Birth=No, Can Fly=Yes) → Birds

(Give Birth=No, Live in Water=Yes) → Fishes

(Give Birth=Yes) → Mammals

(Give Birth=No, Can Fly=No, Live in Water=No) → Reptiles

( ) → Amphibians

RIPPER:

(Live in Water=Yes) → Fishes

(Have Legs=No) → Reptiles

(Give Birth=No, Can Fly=No, Live In Water=No) → Reptiles

(Can Fly=Yes,Give Birth=No) → Birds

() → Mammals

C4.5rules versus RIPPER

C4.5rules:

		PREDICTED CLASS				
		Amphibians	Fishes	Reptiles	Birds	Mammals
ACTUAL	Amphibians	2	0	0	0	0
CLASS	Fishes	0	2	0	0	1
	Reptiles	1	0	3	0	0
	Birds	1	0	0	3	0
	Mammals	0	0	1	0	6

RIPPER:

		PREDICTED CLASS				
		Amphibians	Fishes	Reptiles	Birds	Mammals
ACTUAL	Amphibians	0	0	0	0	2
CLASS	Fishes	0	3	0	0	0
	Reptiles	0	0	3	0	1
	Birds	0	0	1	2	1
	Mammals	0	2	1	0	4

Eager Learners

- So far we have learnt that classification involves
  - An *inductive step* for constructing classification models from data
  - A *deductive step* for applying the derived model to previously unseen instances
- For decision tree induction and rule based classifiers, the models are constructed immediately after the training set is provided
- Such techniques are known as *eager learners* because they intend to learn the model as soon as possible, once the training data is available

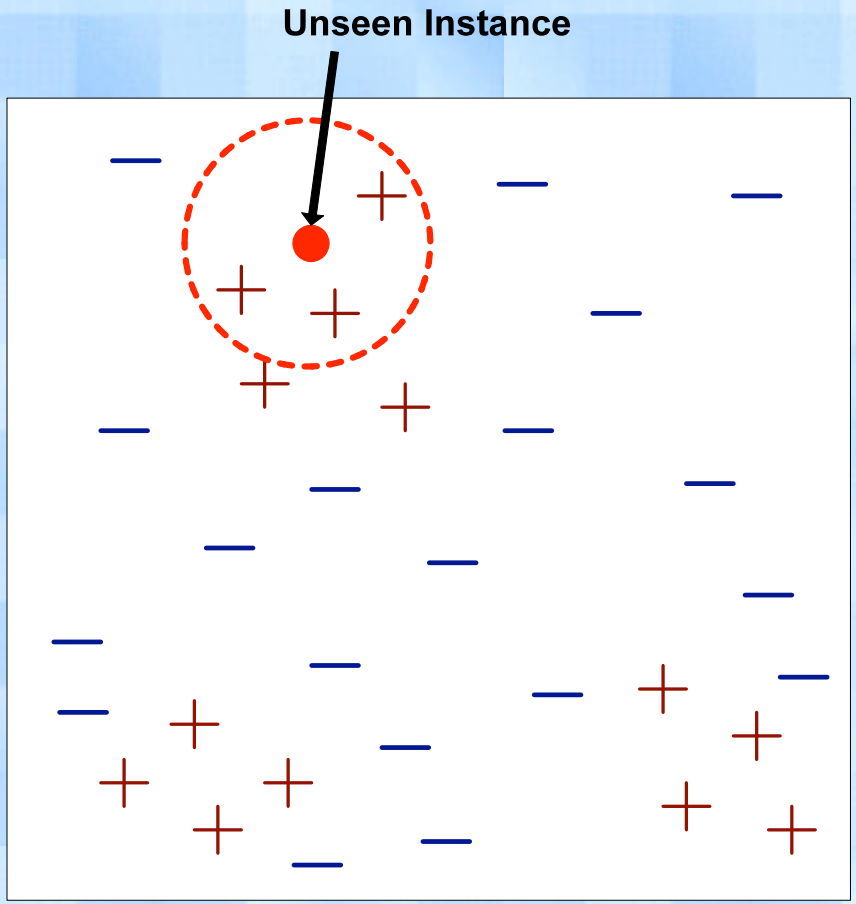
Lazy Learners

- An opposite strategy would be to delay the process of generalizing the training data until it is needed to classify the unseen instances
- Techniques that employ such strategy are known as *lazy learners*
- An example of lazy learner is the *Rote Classifier*, which memorizes the entire training data and perform classification only if the attributes of a test instance matches one of the training instances exactly

Nearest-neighbor Classifiers

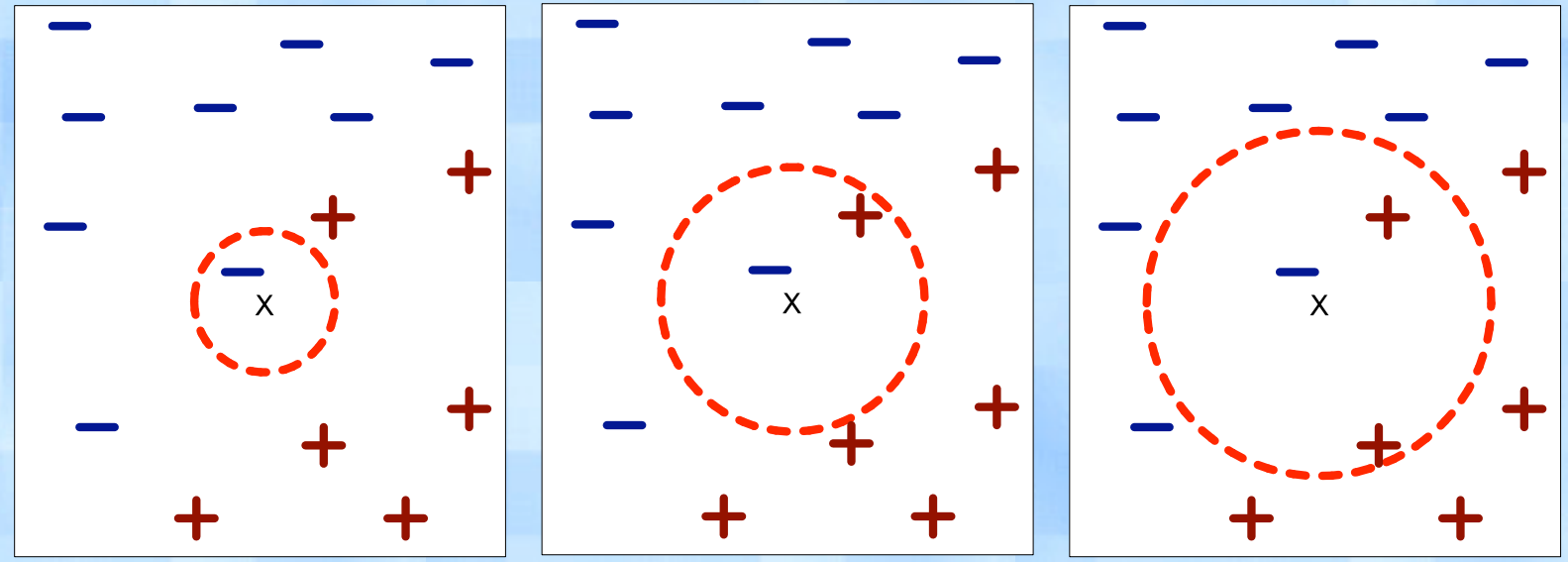
- One way to make the *"Rote Classifier"* approach more flexible is to find all training instances that are *relatively similar* to the test instance. They are called *nearest neighbors* of the test instance
- The test instance can then be classified according to the class label of its neighbors
- *"If it walks like a duck, quacks like a duck, and looks like a duck, then it's probably a duck"*

Nearest Neighbor Classifiers



- For data set with continuous attributes
- Requires
  - Set of sorted instances
  - Distance metric to compute distance between instances
  - Value of *k*, the number of nearest neighbors to retrieve
- For classification
  - Retrieve the *k* nearest neighbors
  - Use class labels of the nearest neighbors to determine the class label of the unseen instance (e.g. by taking majority vote)

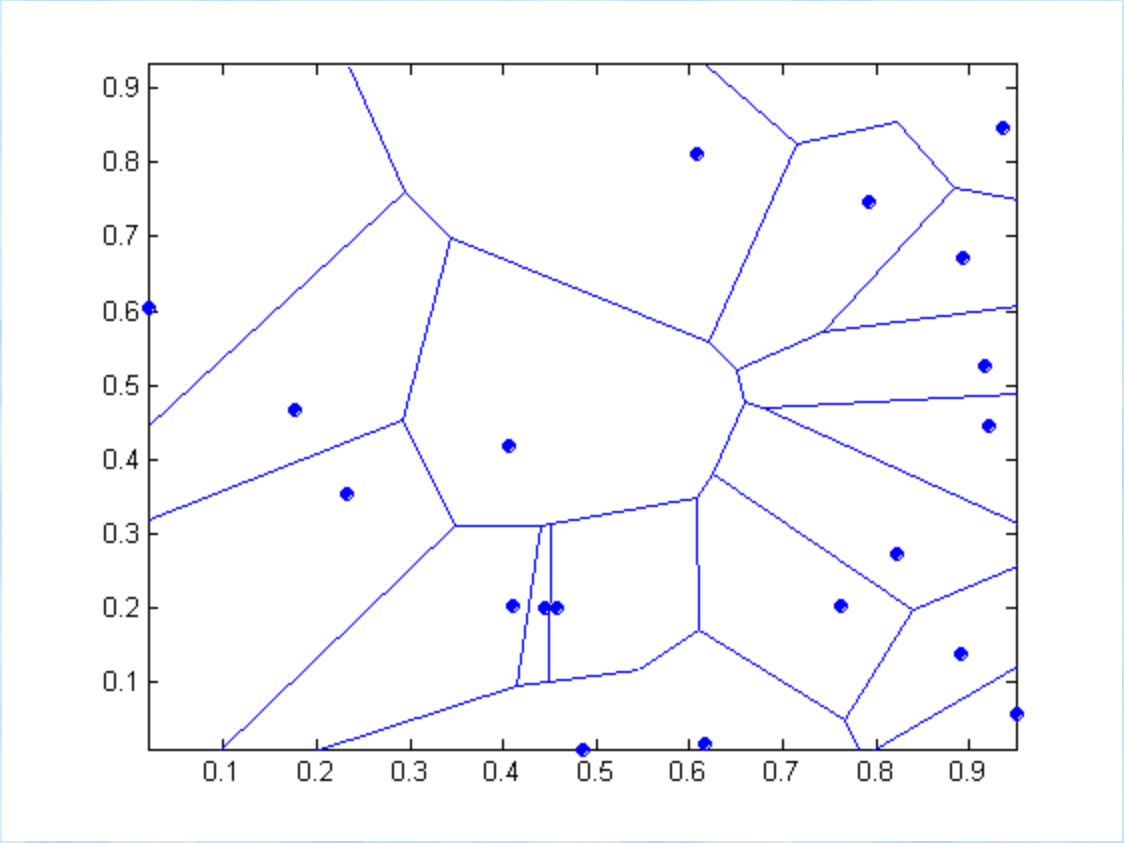
Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

- The *k*-nearest neighbors of an instance *x* are defined as the data points having the *k* smallest distances to *x*

1-nearest Neighbor



- If $k = 1$, we can illustrate the decision boundary of each class by using a *Voronoi diagram*

Distance Metric

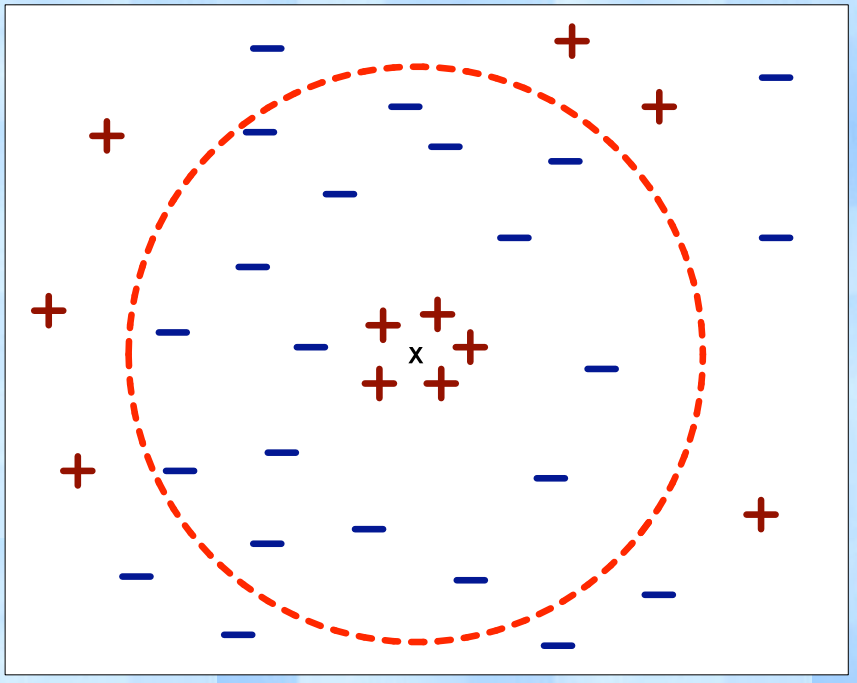
- Distance metric is required to compute the distance between two instances
- A nearest neighbor classifier represents each instance as a data point embedded in a *d*-dimensional space, where *d* is the number of continuous attributes
- Euclidean Distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Weighted Distance
  - Weight factor, $w = 1 / d^2$
  - Weight the vote according to the distance

Choosing the value of *k*

- If *k* is too small, classifier is sensitive to noise points
- If *k* is too large
  - Computationally intensive
  - Neighborhood may include points from other classes



Nearest Neighbor Classifiers

- Problems with Euclidean distance
  - High dimensional data
    - Curse of dimensionality
  - Can produce counter intuitive results (e.g. text document classification)
    - Solution: Normalization

1 1 1 1 1 1 1 1 1 1 1 0		1 0 0 0 0 0 0 0 0 0 0 0
	vs.	
0 1 1 1 1 1 1 1 1 1 1 1		0 0 0 0 0 0 0 0 0 0 0 1

Euclidean distance between pairs of un-normalized vectors

Nearest Neighbor Classifiers

- Other issues
  - Nearest neighbor classifiers are lazy learners
    - It does not build models explicitly
    - Classifying instances is expensive
  - Scaling of attributes
    - Person:
      - (Height in meters, Weight in pounds, Class)
      - Height may vary from 1.5 m to 1.85 m
      - Weight may vary from 90 lbs to 250 lbs
      - Distance measure could be dominated by difference in weights