# CS 561/571:Rule-based Classification

# Rule-Based Classifier

- Classify records by using a collection of "if…then…" rules
- **Rule**: $(Condition) \rightarrow y$
  - where
    - *Condition* is a conjunctions of attributes
    - *y* is the class label
  - *LHS*: rule antecedent or condition
  - *RHS*: rule consequent
  - Examples of classification rules:
    - (Blood Type=Warm) $\wedge$ (Lay Eggs=Yes) $\rightarrow$ Birds
    - (Taxable Income < 50K) $\wedge$ (Refund=Yes) $\rightarrow$ Evade=No

# 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: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

# Application of Rule-Based Classifier

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

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

| Name | Blood Type | Give Birth | Can Fly | Live in Water | Class |
|---|---|---|---|---|---|
| hawk | warm | no | yes | no | ? |
| grizzly bear | warm | yes | no | no | ? |

The rule R1 covers a hawk => Bird

The rule R3 covers the grizzly bear => Mammal

# Rule Coverage and Accuracy

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

| *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** |

**(Status=Single) → No**

**Coverage = 40%, Accuracy = 50%**

# How does Rule-based Classifier Work?

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

| 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 | ? |

A lemur triggers rule R3, so it is classified as a mammal

A turtle triggers both R4 and R5

A dogfish shark triggers none of the rules

# Characteristics of Rule-Based Classifier

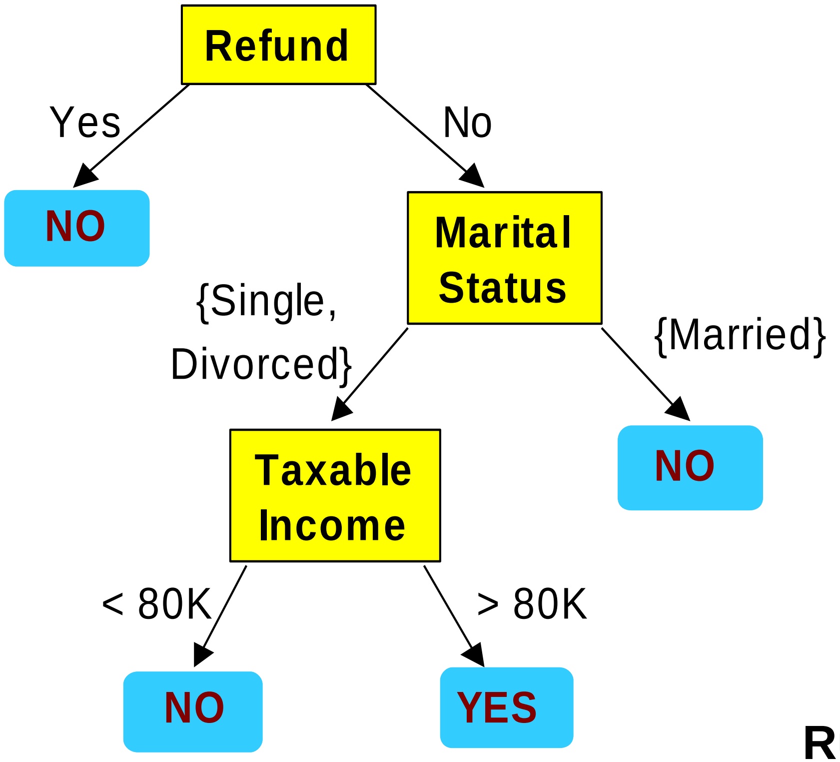
- Mutually exclusive rules
  - Rules are mutually exclusive if they are not triggered by the same record
  - Every record is covered by **at most** one rule

- Exhaustive rules
  - Classifier has exhaustive coverage if there is a rule for each combination of attribute values
  - Each record is covered by **at least** one rule

# Example

| *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** |

# From Decision Trees To Rules




**Classification Rules**

(Refund=Yes) ==> No

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

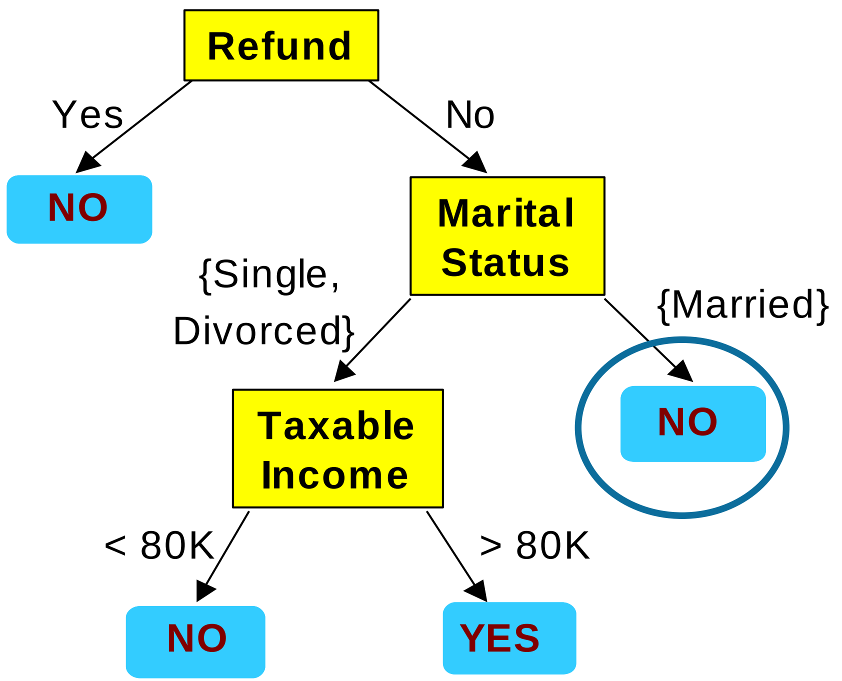
(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(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) $\wedge$ (Status=Married) $\rightarrow$ No

**Simplified Rule:** (Status=Married) $\rightarrow$ No

# Effect of Rule Simplification

- Rules are no longer mutually exclusive
  - A record may trigger more than one rule
  - Solution?
    - Ordered rule set
    - Unordered rule set – use voting schemes
- Ordered rule set
  - Arrange according to the decreasing order of priority
    - Coverage, accuracy etc.
    - Order in which rules are generated
  - for a test record
    - Classify with the highest ranked rule that covers the record

# Effect of Rule Simplification

- Unordered rule set – use voting schemes
  - Allow a test record to trigger multiple classification rules
  - Consider the consequent of each rule as a vote for a particular class
  - Votes are tallied to determine the class label of the test record
  - Record assigned the class having the highest number of votes
- Advantage
  - Less cost associated with model building (*don’t have to keep the rules in sorted order*)
- Disadvantage
  - Classifying a test record is expensive as it must be compared with all the preconditions

# Other issues

- Rules are no longer exhaustive
  - A record may not trigger any rules
  - Solution?
    - Use a default class

      $r_d$: ()-$\rightarrow y_d$

# Ordered Rule Set

- Rules are rank ordered according to their priority
  - An ordered rule set is known as a decision list
- When a test record is presented to the classifier
  - It is assigned to the class label of the highest ranked rule it has triggered
  - If none of the rules fired, it is assigned to the default class

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

| Name | Blood Type | Give Birth | Can Fly | Live in Water | Class |
|---|---|---|---|---|---|
| turtle | cold | no | no | sometimes | ? |

# Rule Ordering Schemes

- Rule-based ordering
  – Individual rules are ranked based on their quality
- Class-based ordering
  – Rules that belong to the same class appear together

**Rule-based Ordering**

(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

**Class-based Ordering**

(Refund=Yes) ==> No

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

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

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

# Building Classification Rules

- Direct Method:
  - Extract rules directly from data
  - e.g.: RIPPER, CN2, Holte’s 1R

- Indirect Method:
  - Extract rules from other classification models (e.g. decision trees, neural networks, etc.)
  - e.g: C4.5 rules

# Direct Method: Sequential Covering

- Algorithm extracts rules one class at a time for the datasets
  - Class sequence depends on a number of factors:

    class prevalence (# records that belong to a particular class)

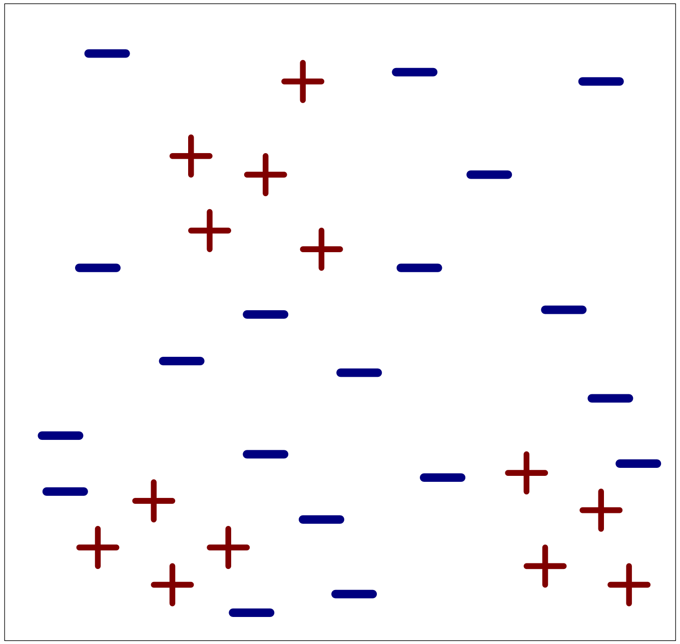
    cost of misclassifying records from a given class

1. Start from an empty rule
2. Grow a rule using the Learn-One-Rule function
3. Remove training records covered by the rule
4. Repeat Step (2) and (3) until stopping criterion is met

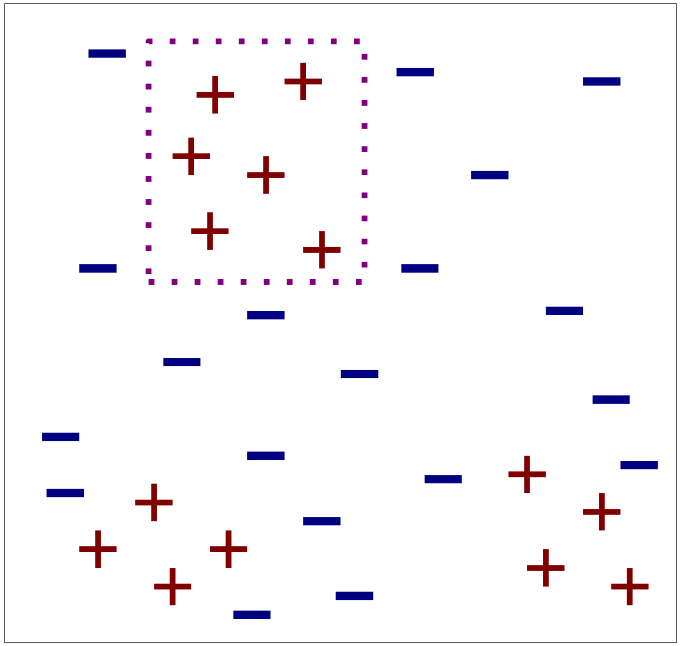
# Sequential Covering

- For any class *y*
  - All the training records belong to class *y* are positive examples
  - Other training records are the negative examples for class *y*
- Desirable rule for *Learn-One-Rule function*
  - Covers most of the *+ve examples and almost no –ve examples*
  - Once the rule is found, training records covering the rules are eliminated
  - Finding an optimal rule is computationally expensive

# Example of Sequential Covering

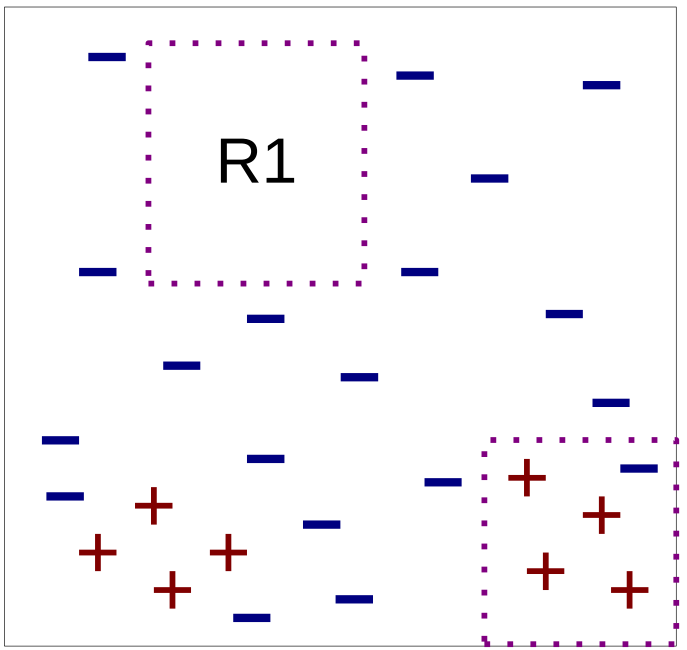


(i) Original Data

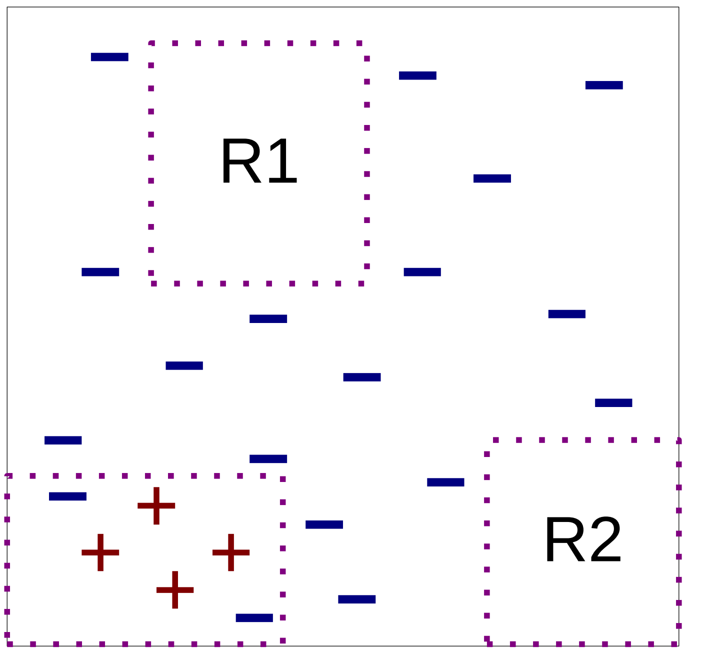


(ii) Step 1

# Example of Sequential Covering…




(iii) Step 2




(iv) Step 3

# Aspects of Sequential Covering

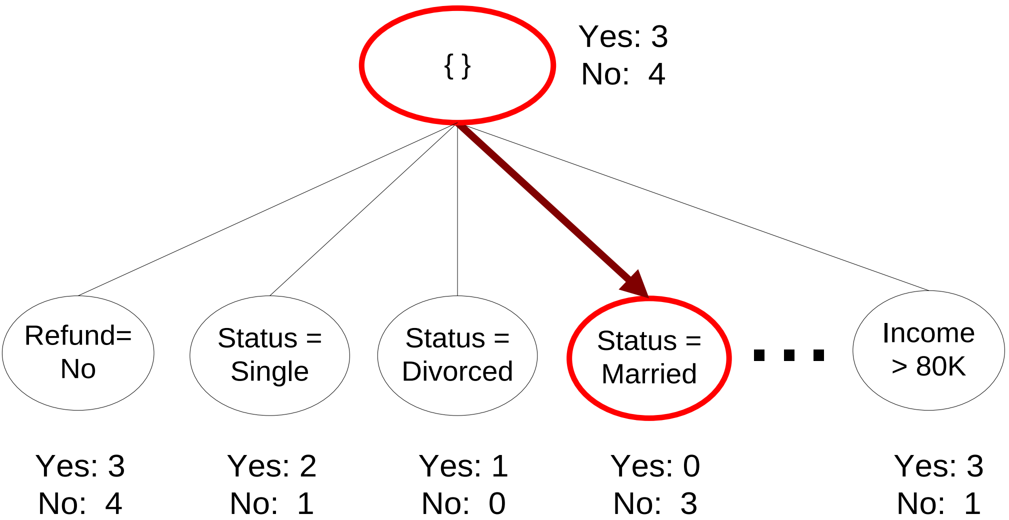
- Rule Growing
- Instance Elimination
- Rule Evaluation
- Stopping Criterion
- Rule Pruning

# Example

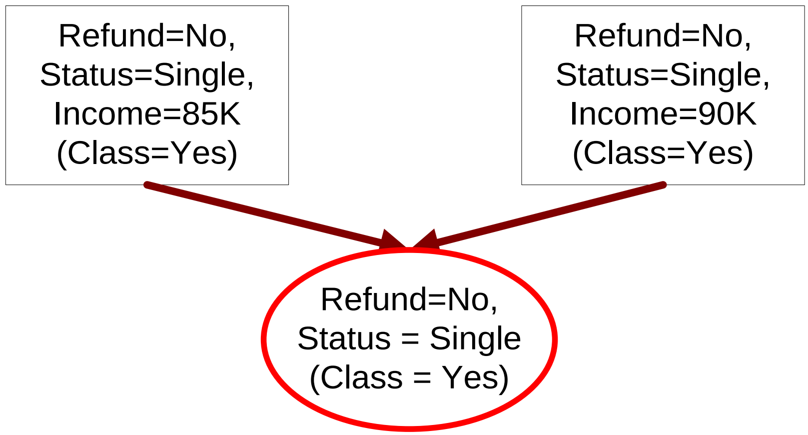
| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | **No** |
| 2 | No | **Married** | 100K | **No** |
| 3 | No | Single | 70K | **No** |
| 4 | Yes | **Married** | 120K | **No** |
| 5 | No | Divorced | 95K | **Yes** |
| 6 | No | **Married** |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

# Rule Growing

- Two common strategies




(a) General-to-specific




(b) Specific-to-general

- **Antecedent is empty and consequent is the class**
- **Conjuncts are added to improve the rule's quality**
- **Stop when adding does not improve the quality of rule**

- **Choose one +ve example randomly**
- **Rule is generalized by removing one of its conjuncts so that it covers more +ve examples**
- **Stop when rule starts covering -ve examples**

# Rule Growing (Examples)

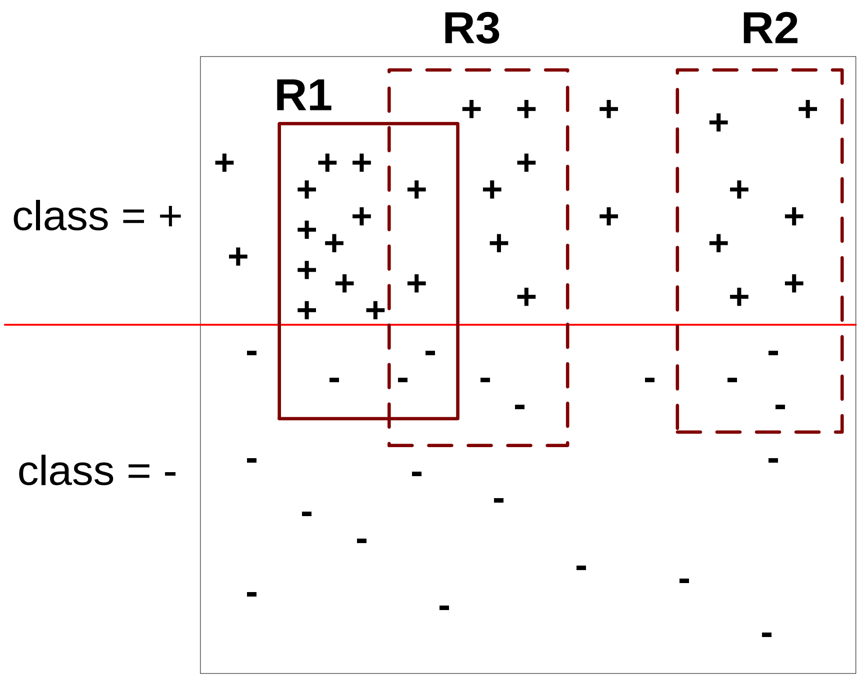
- **CN2 Algorithm**:
  - Start from an empty conjunct: {}
  - Add conjuncts that minimizes the entropy measure: {A}, {A,B}, …
  - Determine the rule consequent by taking majority class of instances covered by the rule
- **RIPPER Algorithm**:
  - Start from an empty rule: {} => class
  - Add conjuncts that maximizes FOIL's [first-order inductive learner (**FOIL**)] information gain measure:
    - R0: {} => class (initial rule)
    - R1: {A} => class (rule after adding conjunct)
    - Gain(R0, R1) =p1[ log (p1/(p1+n1)) – log (p0/(p0 + n0)) ]
    - 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

# Instance Elimination

**Accuracies:**

**R1: 12/15 (80%)**
**R2: 7/10 (70%)**
**R3: 8/12 (66.7%)**

- Why do we need to eliminate instances?
  - Otherwise, the next rule is identical to previous rule
- Why do we remove positive instances?
  - Ensure that the next rule is different
  - Avoid overestimation (if +ve examples of R1 are not removed then R3 is overestimated)
- Why do we remove negative instances?
  - Prevent underestimating accuracy of rule
  - Compare rules R2 and R3 in the diagram with respect to R1
  - End up in preferring R2 over R3
  - Half of the false positive errors are already accounted for by R1




*Check R1+R3 vs R1+R2!*

# Rule Evaluation

- Metrics:
  - Accuracy $= \dfrac{n_c}{n}$
  - Laplace $= \dfrac{n_c + 1}{n + k}$
  - M-estimate $= \dfrac{n_c + kp}{n + k}$

*n :* Number of instances

$n_c$ *:* Number of instances covered by rule

*k* : Number of classes

*p* : Prior probability of classes

# Stopping Criterion and Rule Pruning

- Stopping criterion
  - Compute the gain
  - If gain is not significant, discard the new rule

- Rule Pruning
  - Similar to post-pruning of decision trees
  - Reduced Error Pruning:
    - Remove one of the conjuncts in the rule
    - Compare error rate on validation set before and after pruning
    - If error improves, prune the conjunct

# Summary of Direct Method

- Grow a single rule
- Remove Instances from rule
- Prune the rule (if necessary)
- Add rule to Current Rule Set
- Repeat

# Direct Method: RIPPER

- For 2-class problem, choose one of the classes as positive class, and the other as negative class
  - Learn rules for the minority class
  - Majority class is the default one
- 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

# Direct Method: RIPPER

- Growing a rule:
  - Start from empty rule
  - Add conjuncts as long as they improve FOIL's information gain
    - r: A→ + (covers p0 +ve and n0 –ve examples); r1→ A ^ B → + (covers p1 + ve and n1 –ve examples)

$$Gain = p1(\log\frac{p1}{p1+n1} - \log\frac{p0}{p0+no})$$

  - 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

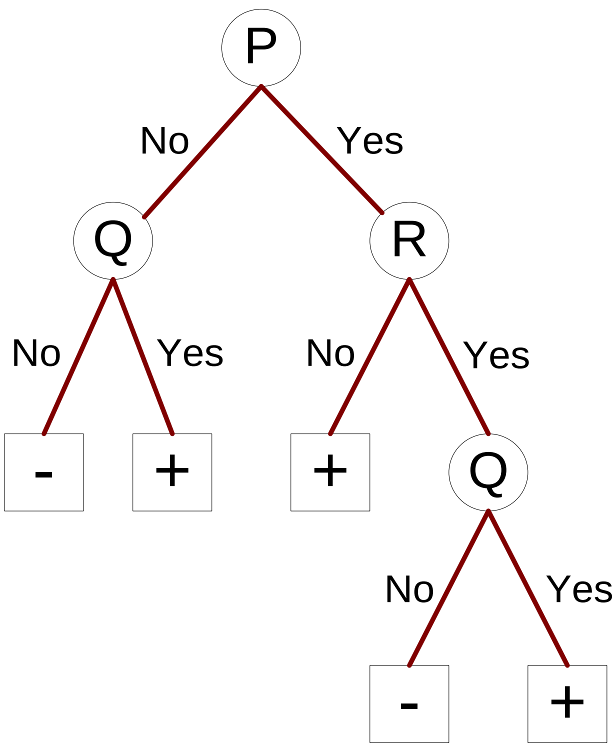
# Direct Method: RIPPER

- 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 new description length
    - Stop adding new rules when the new description length is *d* bits longer than the smallest description length obtained so far

# 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

# Indirect Methods




**<u>Rule Set</u>**

r1: (P=No,Q=No) ==> -

r2: (P=No,Q=Yes) ==> +

r3: (P=Yes,R=No) ==> +

r4: (P=Yes,R=Yes,Q=No) ==> -

r5: (P=Yes,R=Yes,Q=Yes) ==> +

# Indirect Method: 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 of r against all r's
  - Prune if one of the r's has lower pessimistic error rate
  - Repeat until we can no longer improve generalization error

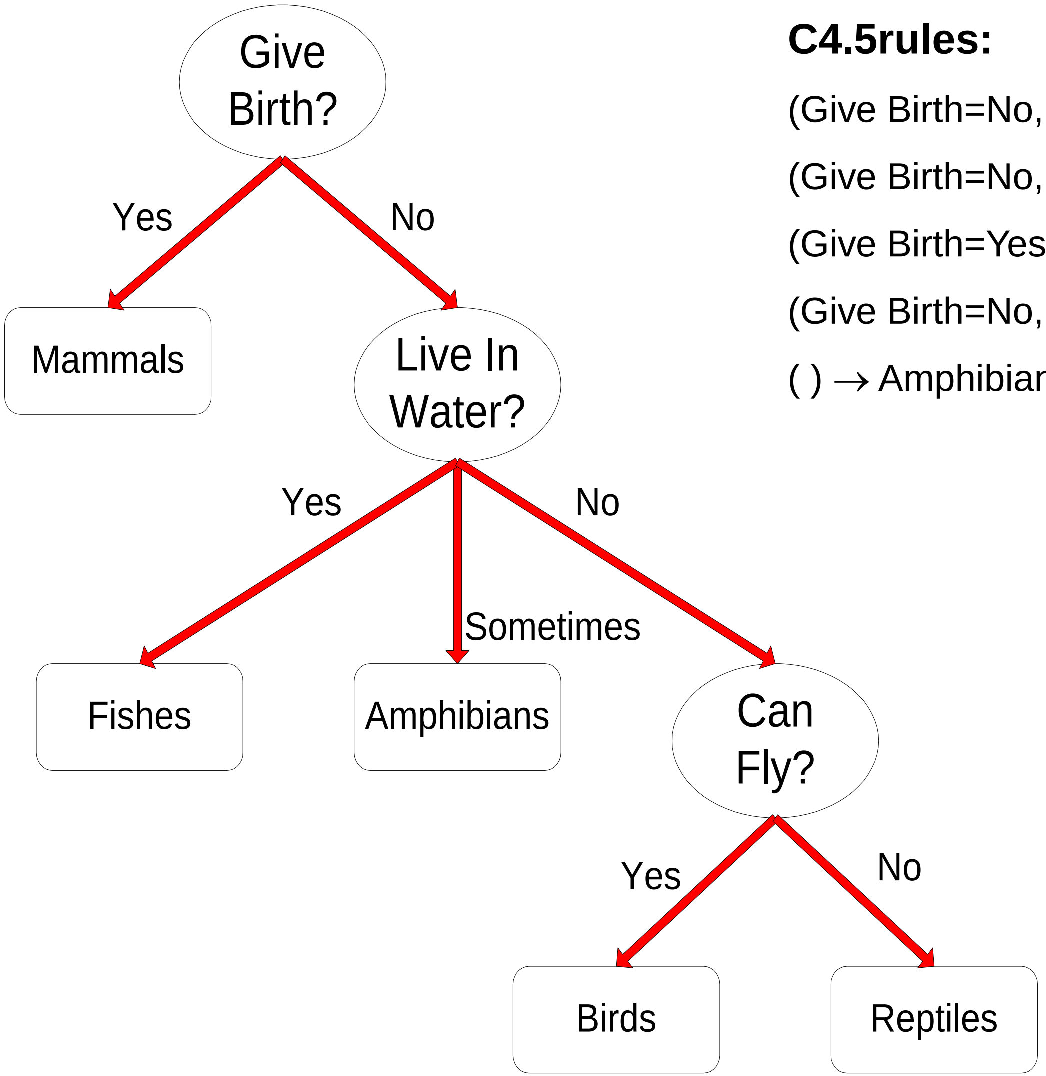
# Indirect Method: C4.5 rules

- Instead of ordering the rules, order subsets of rules (class ordering)
  - Each subset is a collection of rules with the same rule consequent (class)
  - Compute description length of each subset
    - Description length = L(error) + g L(model)
    - g is a tuning parameter that takes into account the presence of redundant attributes in a rule set (default value = 0.5)
    - L (error)-# bits needed to encode the misclassified examples
    - L (model)= # bits needed to encode the model

# 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 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

# C4.5 versus C4.5rules versus RIPPER

**C4.5 rules:**

| | | | 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:**

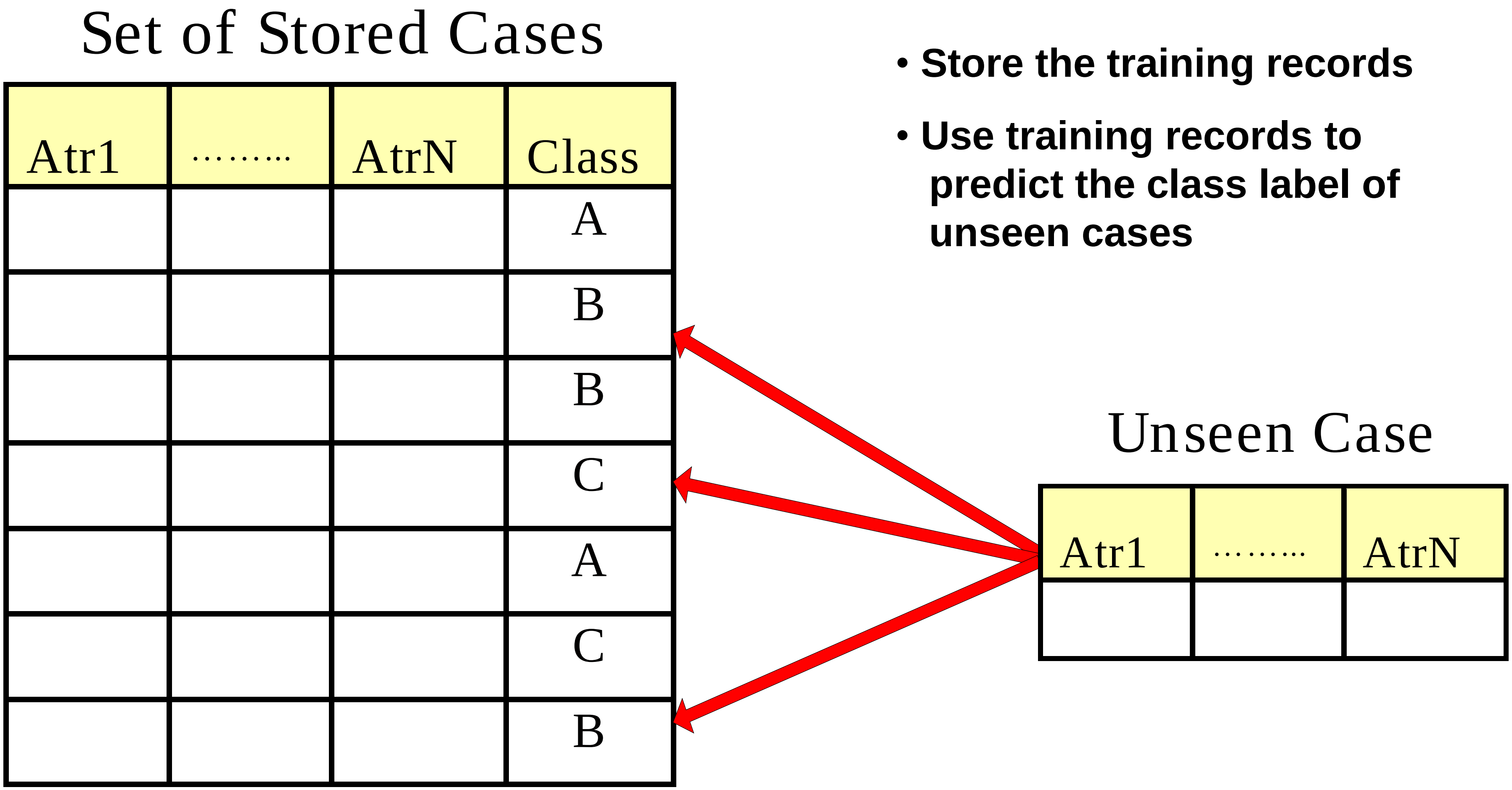
| | | | PREDICTED CLASS | | | |
|---|---|---|---|---|---|---|
| | | Amphibians | Fishes | Reptiles | Birds | Mammals |
| ACTUAL | Amphibians | 0 | 0 | 0 | 0 | 2 |
| CLASS | Fishes | 0 | 3 | 0 | 0 | 0 |
| | Reptiles | 0 | 0 | 3 | 0 | 1 |
| | Birds | 0 | 0 | 1 | 2 | 1 |
| | Mammals | 0 | 2 | 1 | 0 | 4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

# 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

# Instance-Based Classifiers

- Store the training records
- Use training records to predict the class label of unseen cases

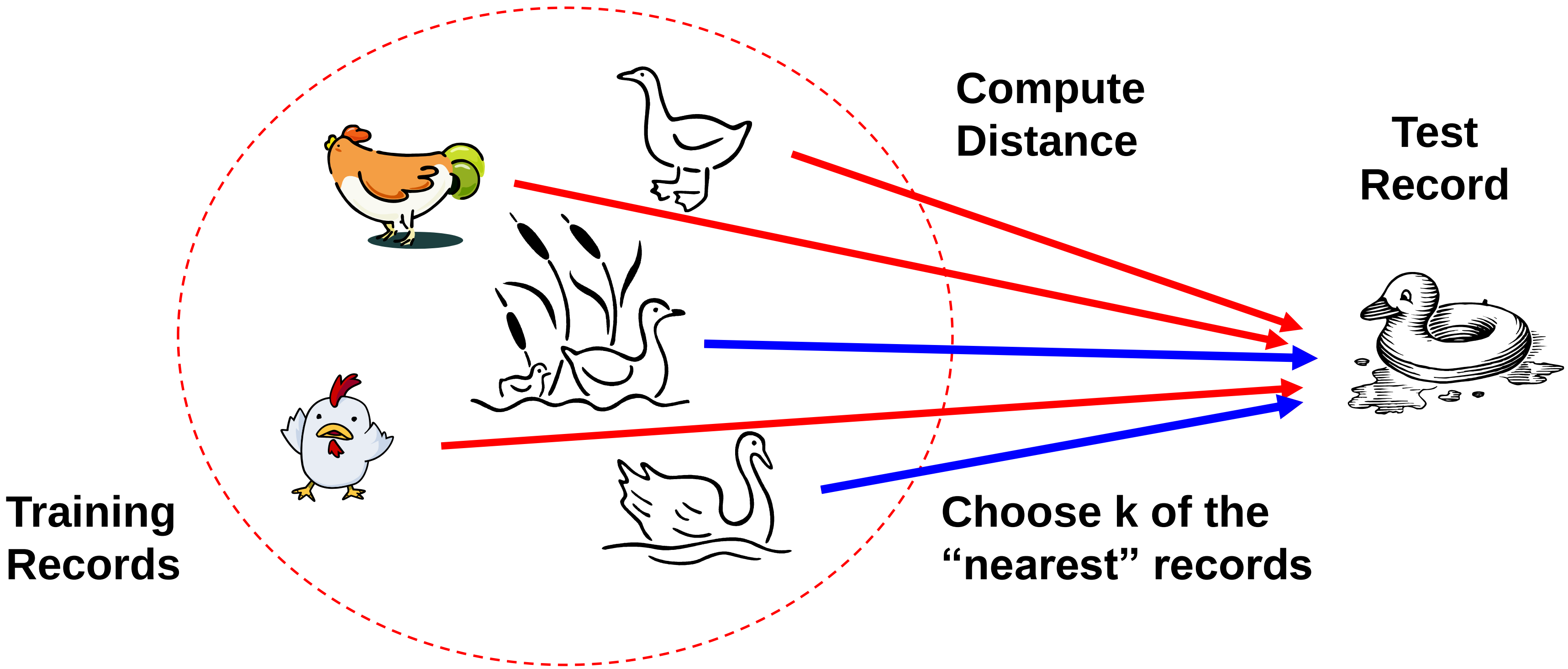


# Instance Based Classifiers

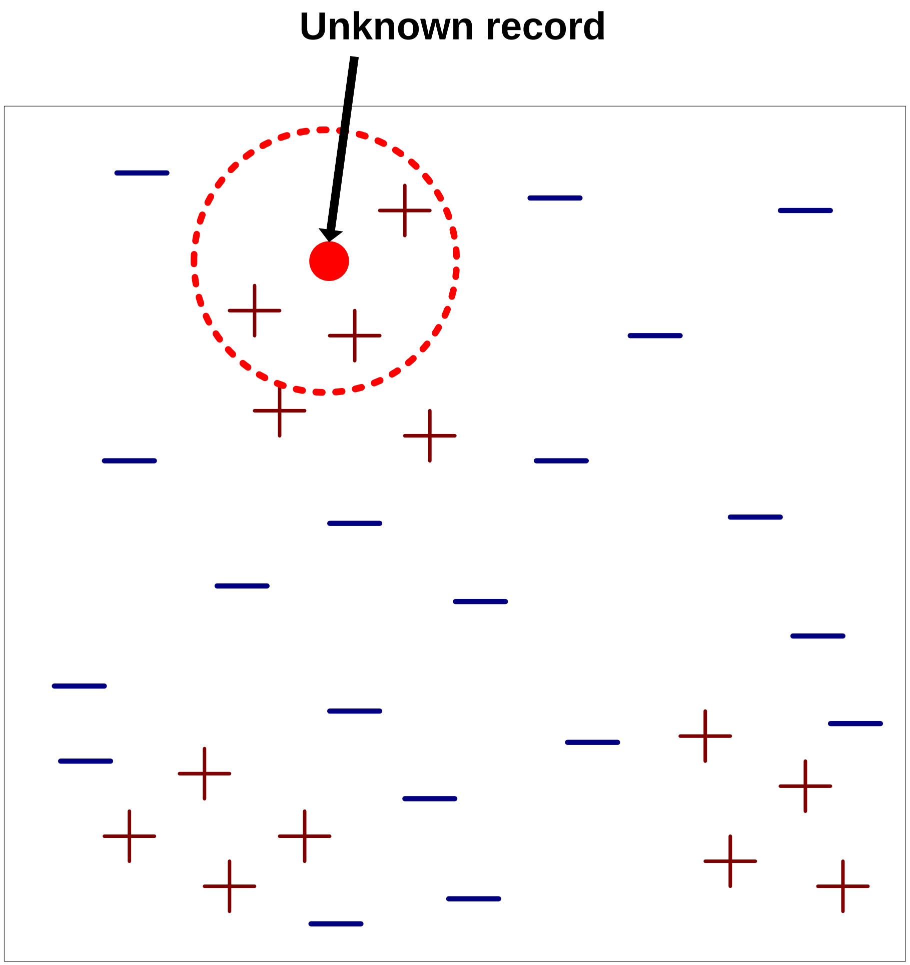
- Examples:
  - Rote-learner
    - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
  - Nearest neighbor
    - Uses k “closest” points (nearest neighbors) for performing classification

# Nearest Neighbor Classifiers

- Basic idea:
  – If it walks like a duck, quacks like a duck, then it's probably a duck



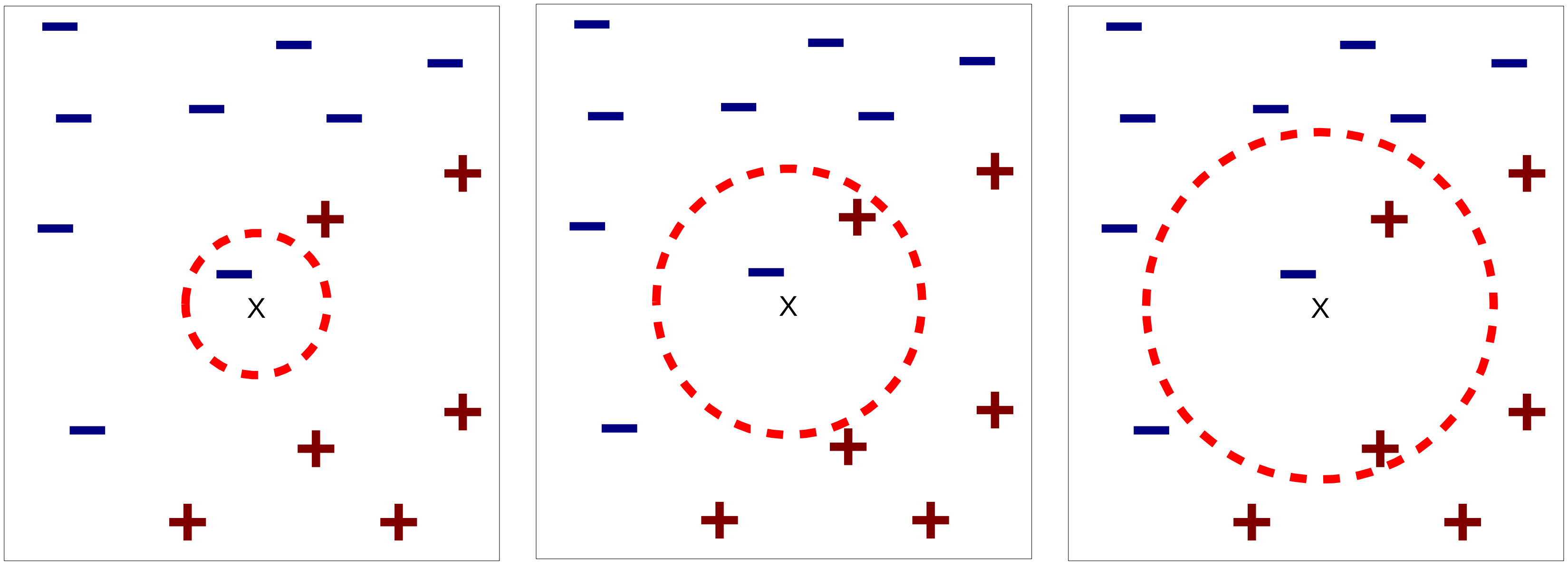
# Nearest-Neighbor Classifiers




- Requires three things
  - Set of stored records
  - Distance metric to compute distance between records
  - Value of *k*, the number of nearest neighbors to retrieve

- Classifying an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

# Definition of Nearest Neighbor




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

K-nearest neighbors of a record x are data points that have the k smallest distance to x

# Nearest Neighbor Classification

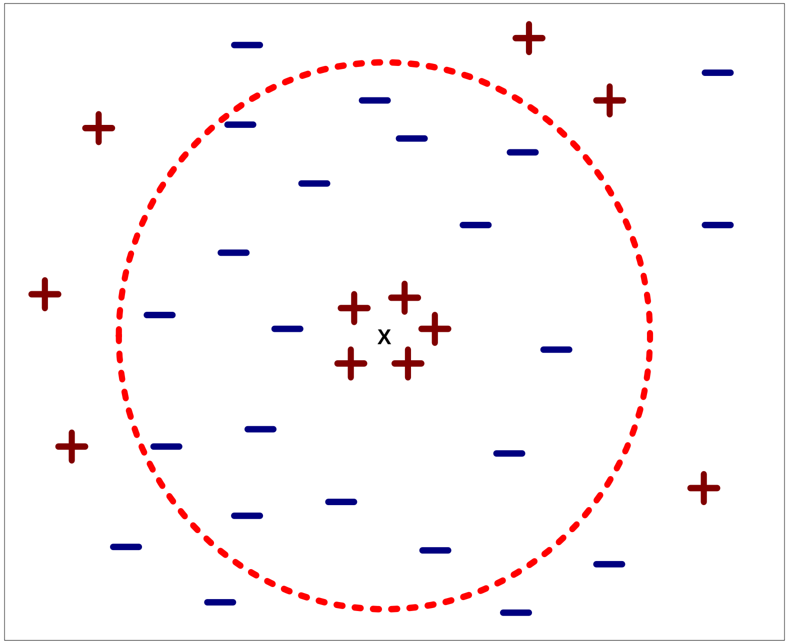
- Compute distance between two points:
  - Euclidean distance

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

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - weigh the vote according to distance
    - Weight factor, $w = 1/d^2$

# Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



# Nearest Neighbor Classification…

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

# Nearest Neighbor Classification…

- Problem with Euclidean measure:
  - High dimensional data
    - curse of dimensionality
  - Can produce counter-intuitive results

| | | | |
|---|---|---|---|
| **1 1 1 1 1 1 1 1 1 1 1 0** | vs | **1 0 0 0 0 0 0 0 0 0 0 0** |
| **0 1 1 1 1 1 1 1 1 1 1 1** | | **0 0 0 0 0 0 0 0 0 0 0 1** |
| **d = 1.4142** | | **d = 1.4142** |

***What is the solution??***

# Nearest neighbor Classification…

- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction and rule-based systems
  - Classifying unknown records are relatively expensive

# Example: PEBLS

- PEBLS: Parallel Examplar-Based Learning System (Cost & Salzberg)
  - Works with both continuous and nominal features
    - For nominal features, distance between two nominal values is computed using modified value difference metric (MVDM)
  - Each record is assigned a weight factor
  - Number of nearest neighbor, k = 1

# Example: PEBLS

| *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** |

Distance between nominal attribute values:

d(Single,Married)

= | 2/4 – 0/4 | + | 2/4 – 4/4 | = 1

d(Single,Divorced)

= | 2/4 – 1/2 | + | 2/4 – 1/2 | = 0

d(Married,Divorced)

= | 0/4 – 1/2 | + | 4/4 – 1/2 | = 1

d(Refund=Yes,Refund=No)

= | 0/3 – 3/3 | + | 3/3 – 4/7 | = 6/7

| Class | Marital Status | | |
|---|---|---|---|
| | Single | Married | Divorced |
| Yes | 2 | 0 | 1 |
| No | 2 | 4 | 1 |

| Class | Refund | |
|---|---|---|
| | Yes | No |
| Yes | 0 | 3 |
| No | 3 | 4 |

$$d(V_1, V_2) = \sum_i \left| \frac{n_{1i}}{n_1} - \frac{n_{2i}}{n_2} \right|$$

# Example: PEBLS

| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| X | Yes | Single | 125K | **No** |
| Y | No | Married | 100K | **No** |

Distance between record X and record Y:

$$\Delta(X,Y) = w_X w_Y \sum_{i=1}^{d} d(X_i, Y_i)^2$$

where:

$$w_X = \frac{\text{Number of times X is used for prediction}}{\text{Number of times X predicts correctly}}$$

$w_X \cong 1$ if X makes accurate prediction most of the time

$w_X > 1$ if X is not reliable for making predictions

**Acknowledgement:** Vipin Kumar et al., Introduction to Data Mining