



# CLASSIFICATION

Business Intelligence  
Dr Sara Migliorini

# CLASSIFICATION

- Classification is the task that predicts the class label associated to a given object (data instance).
- Each data instance of the classification task is characterized by the tuple  $(\mathbf{x}, y)$  where:
  - $\mathbf{x}$  is the set of values that describe the instance
  - $y$  is the class label of the instance (categorical value)

# CLASSIFICATION MODEL

- A classification model is an abstract representation of the relationship between the attribute set and the class label.

$$f(x) = \hat{y}$$

- The classification is correct if

$$\hat{y} = y$$

# TYPES OF CLASSIFICATION

- A classification is said to be binary, if each data instance can be categorized into one of two classes.
- A classification is said to be multiclass, if the number of classes is greater than 2.
- Each class label must be of nominal type.
  - This distinguishes classification from other predictive models, such as regression, where the predicted value is often quantitative.

# PURPOSES OF THE CLASSIFICATION

- A classification model serves two important roles:
  - *Predictive model*: if it is used to classify previously unlabeled instances.
  - *Descriptive model*: if it is used to identify the characteristics that distinguish instances from different classes.
    - E.g.: medical diagnosis → how the model reaches such a decision?
- Not all attributes may be relevant for the classification task and some other might not be able to distinguish the classes by themselves (they must be used in concert with other attributes).

# GENERAL FRAMEWORK

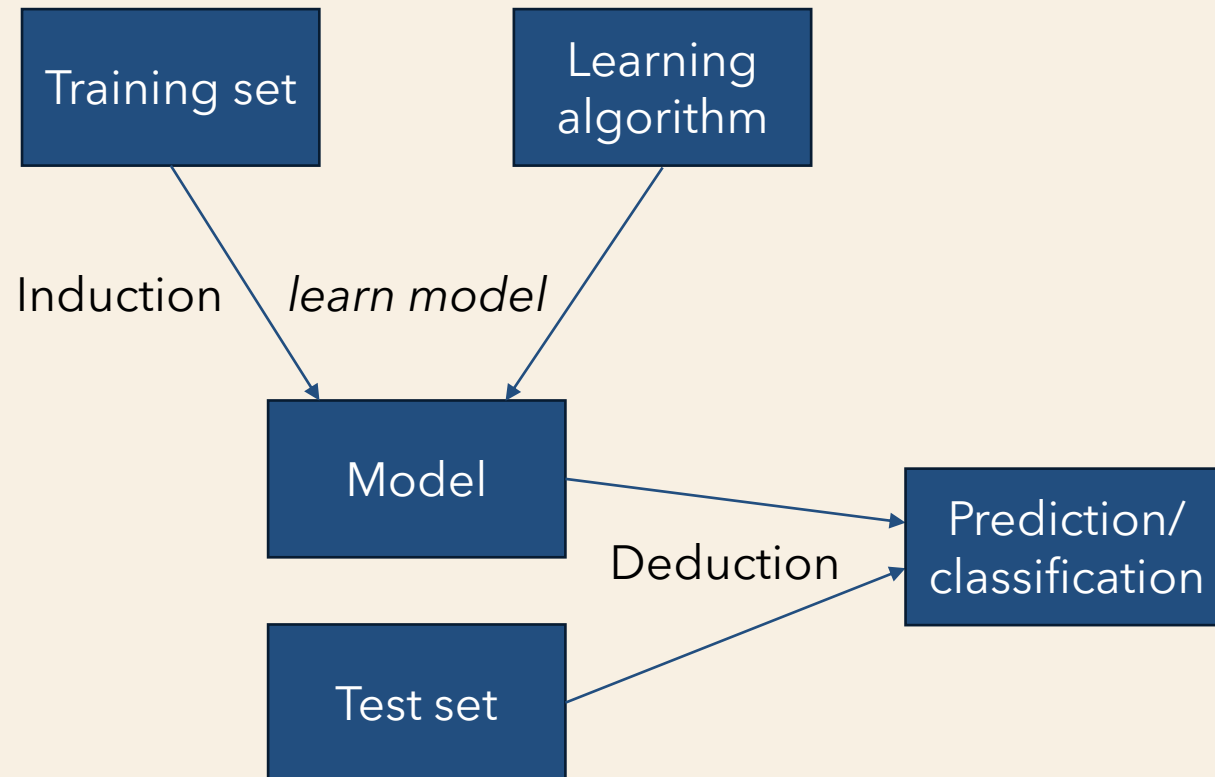
- A classifier is a tool used to perform a classification task (e.g., assigning labels to unlabeled data instances)
  - It is typically described in terms of a model.
- The classification task involves two phases:
  - Induction: construction of a classification model through the application of a learning algorithm on a training set.
    - The training set contains attribute values and class labels for each instance.
  - Deduction: application of a classification model on unseen test instances to predict their class labels.

# GENERAL FRAMEWORK

- Classification technique: a general approach to classification. It consists of a family of related models and a number of algorithms for learning these models.
  - Decision tree
  - Classification rules
  - Association rules
  - Bayesian networks
  - Etc.

Every model defines a classifier, but not every classifier is defined by a single model.

# GENERAL FRAMEWORK



- Generalization performance: capability to provide good accuracy during the deduction step.

# CLASSIFICATION EVALUATION

- Confusion matrix

		Predicted Class	
		Class = P	Class = N
		PP = True positive	NP = False negative
Actual class	Class = P	$f_{PP}$	$f_{PN}$
	Class = N	$f_{NP}$	$f_{NN}$
		NP = False positive	NN = True negative

# CLASSIFICATION EVALUATION

- Accuracy

acc = # of correct predictions / total # of predictions

$$\text{acc} = (f_{PP} + f_{NN}) / (f_{PP} + f_{PN} + f_{NP} + f_{NN})$$

# CLASSIFICATION EVALUATION

- Error rate

err = # of wrong predictions / total # of predictions

$$\text{err} = (f_{\text{PN}} + f_{\text{NP}}) / (f_{\text{PP}} + f_{\text{PN}} + f_{\text{NP}} + f_{\text{NN}})$$

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is solid magenta. The bottom-left square is solid light blue with a white circle. The bottom-right square is dark purple with white concentric circles. The entire graphic is set against a solid blue background.

# TYPES OF CLASSIFIER

# TYPES OF CLASSIFIERS

- Binary vs Multiclass:
  - Binary classifiers assign each data instance to one of two possible labels.
  - If there are more than two possible labels available, the technique is known as multiclass classifier.
- Deterministic vs Probabilistic:
  - A deterministic classifier produces a discrete-valued label to each data instance it classifies.
  - A probabilistic classifier assigns a continuous score between 0 to 1 to indicate how likely it is that an instance belongs to a particular class.
    - Additional information about the confidence in assigning a label to a class.

# TYPES OF CLASSIFIERS

- Linear vs nonlinear:
  - A linear classifier uses a linear separating hyperplane.
  - A nonlinear classifier enables the construction of more complex, nonlinear decision surfaces.
- Global vs local:
  - A global classifier fits a single model to the entire data set.
  - A local classifier partitions the input space into smaller regions and fits a distinct model to training instances in each region.

# TYPES OF CLASSIFIERS

- Generative vs Discriminative:
  - Generative classifiers learn a generative model of every class in the process of predicting class labels.
    - They describe the underlying mechanism that generates the instances belonging to every class label.
  - Discriminative classifiers directly predict the class labels without explicitly describing the distribution of each of them.

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark blue with a white circle. The bottom-right square is dark purple with white concentric circles. The entire graphic is set against a dark blue background.

# DECISION TREE CLASSIFIER

# DECISION TREE

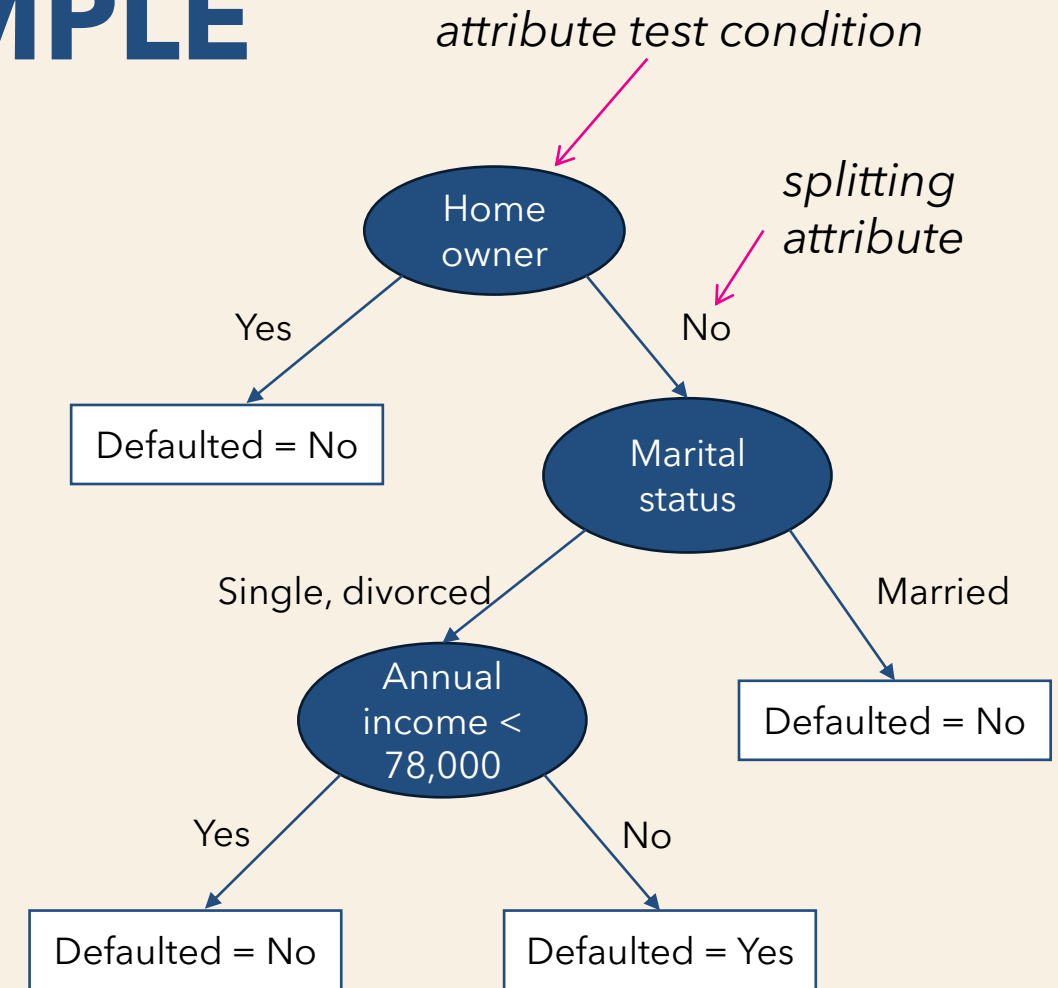
- A decision tree classifier solves the classification problem by asking a series of carefully crafted questions about the attributes of the test instances.
- The series of questions and their possible answers can be organized into a hierarchical structure called a decision tree.
  - Root node
  - Internal nodes: contain attribute test conditions
  - Leaf or terminal nodes: each one is associated with a class label

# DECISION TREE

- Given a decision tree, classifying a test instance consists in traversing the tree starting from the root node, applying the test conditions and following the appropriate branch based on the outcome of the test.
- Once a leaf is reached, we assign the class label associated with that node to the test instance.

# DECISION TREE: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



# DECISION TREE: CONSTRUCTION

- Many possible decision tree can be constructed from a particular data set.
- Some trees are better than others → finding the optimal tree is a computationally expensive task (exponential size of the search space)
- Efficient algorithms have been developed to obtain a reasonable accuracy with suboptimal trees in a reasonable amount of time
  - Greedy algorithms to build the tree in a top-down fashion
  - Local optimal choices in deciding which attribute to use.

# DECISION TREE: ALGORITHMS

- Hunt's Algorithm
- CART
- ID3, C4.5, C5.0
- SLIQ, SPRING

# HUNT'S ALGORITHM

- The Hunt's algorithm builds the decision tree in a *recursive* way.
- The tree initially contains a **single node** that is associated with all the training instances.
- If the node is associated with instances from more than one class, it is expanded by using an attribute test condition that is determined by a **splitting criterion**.

# HUNT'S ALGORITHM

- [Expansion step] A child leaf node is created for each possible outcome of the attribute test condition and the instances associated with the parent node are distributed to the children based on the test outcomes.
- The node expansion step is applied recursively to each child node, as long as it has labels from more than one classes.
- Once all instances associated with the same leaf node have identical labels, the algorithm terminate.

# HUNT'S ALGORITHM: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes

Step 1

Defaulted = No

Since, the majority of the instances is not defaulted.

# HUNT'S ALGORITHM: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes

Step 1

Defaulted = No

Since, the majority of the instances is not defaulted.

Error rate = 30%

# HUNT'S ALGORITHM: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes

Step 1

Defaulted = No

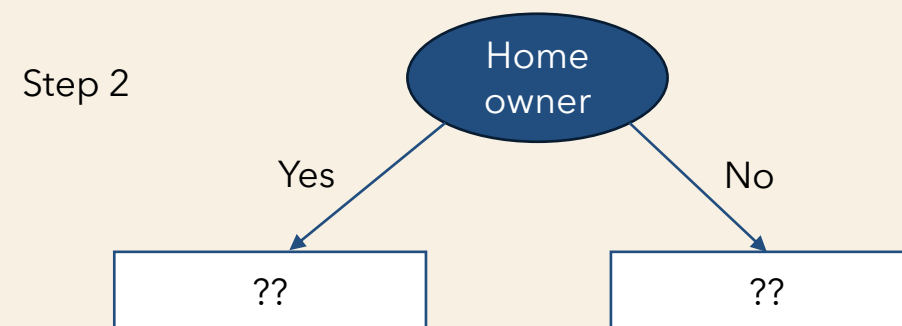
Since, the majority of the instances is not defaulted.

Error rate = 30%

The leaf node can be further expanded, since it contains instances with different labels

# HUNT'S ALGORITHM: EXAMPLE

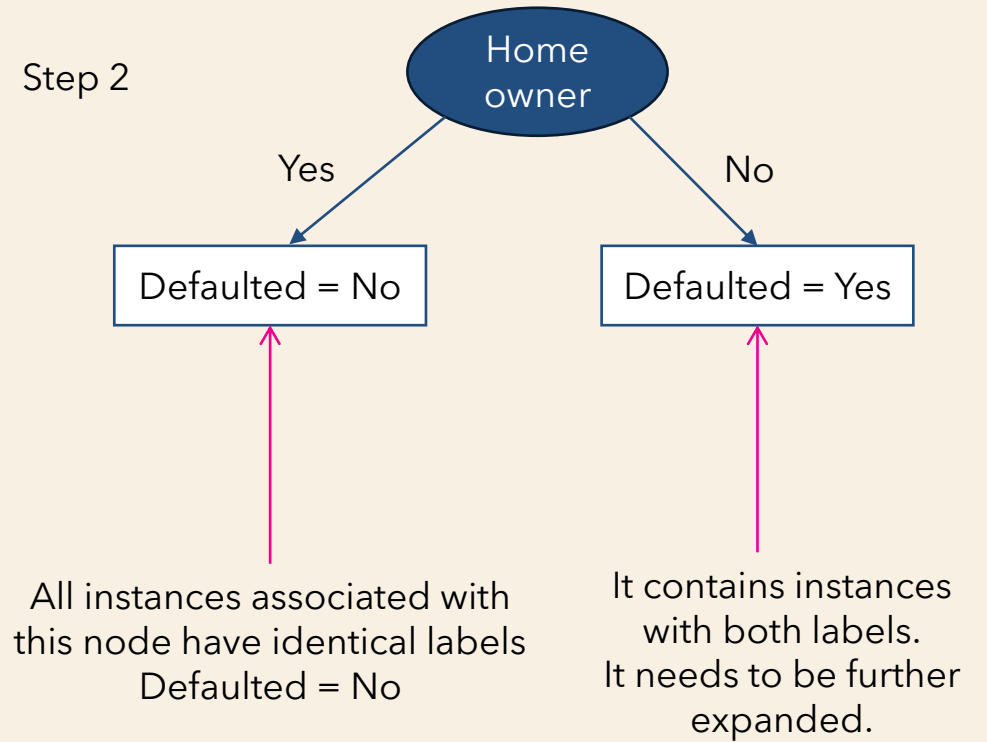
ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



We choose the "home owner" as the attribute for the splitting

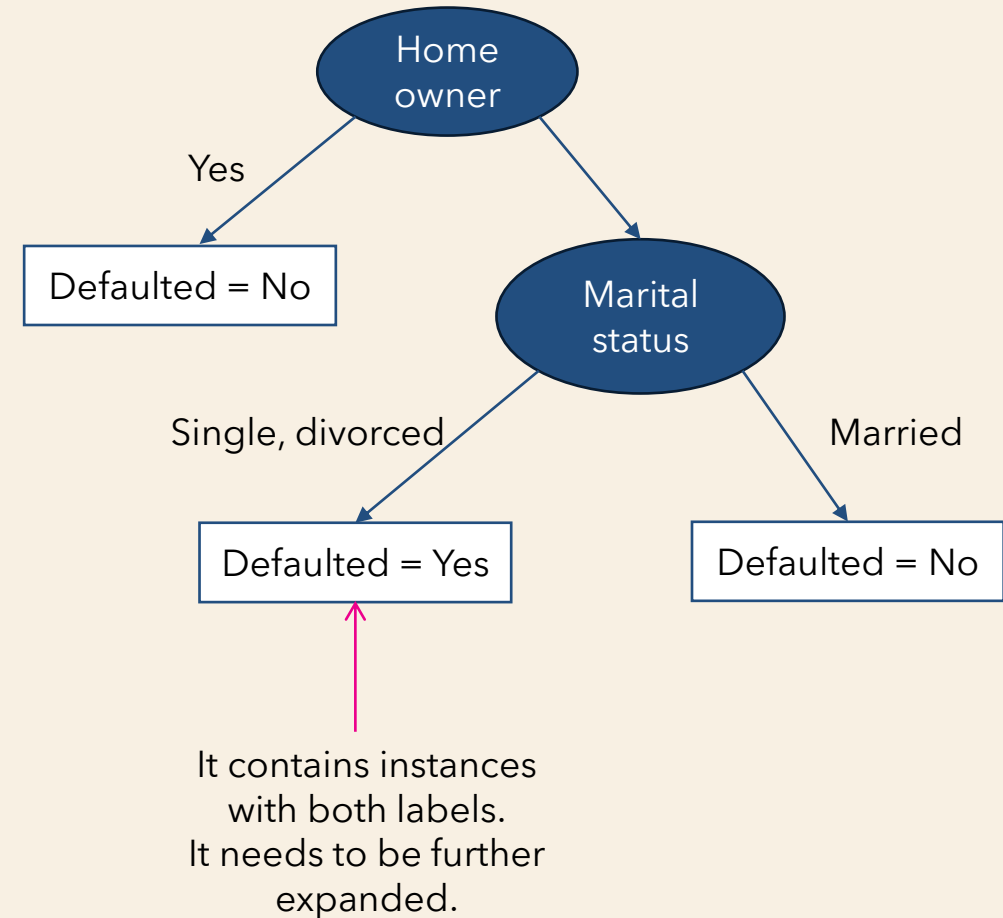
# HUNT'S ALGORITHM: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



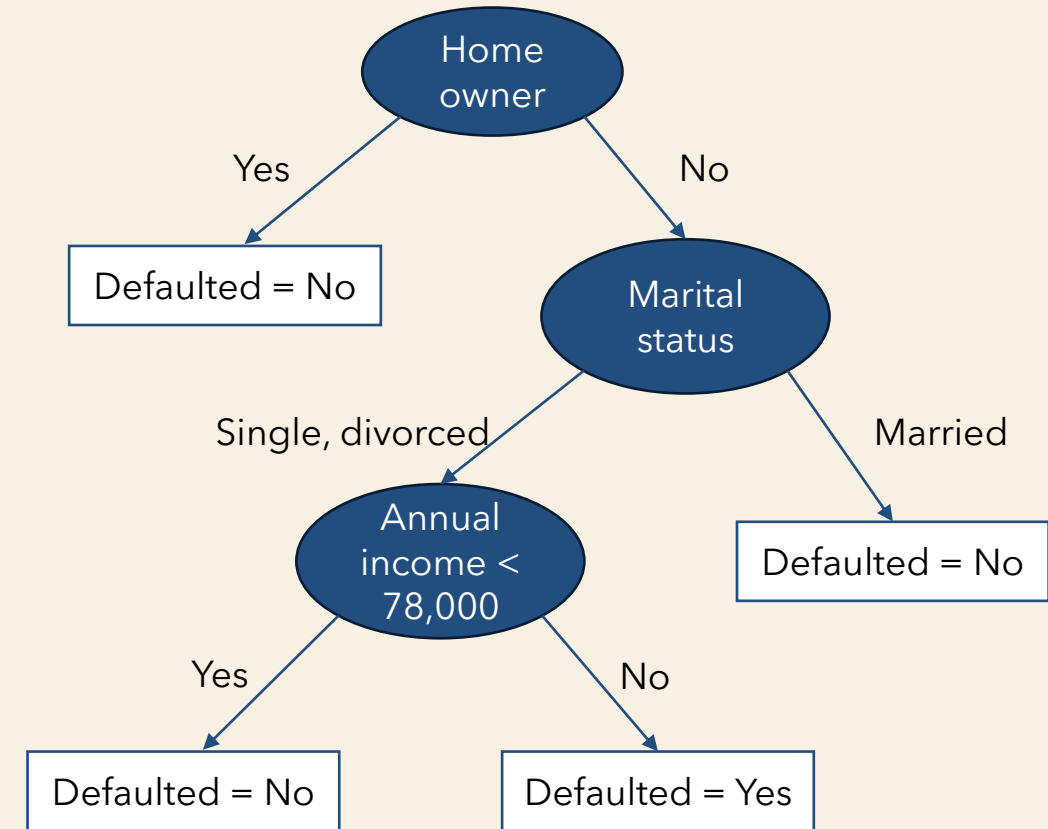
# HUNT'S ALGORITHM: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



# HUNT'S ALGORITHM: EXAMPLE

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



# HUNT'S ALGORITHM: LIMITATIONS

- Some of the child nodes can be empty if none of the training instances have a particular attribute value
  - Declare each of them as a leaf node and assign it the class label that occurs most frequent in the parent node.
- If all training instances associated with a node have identical attribute values but different class labels, it is not possible to expand this node any further.
  - Declare it as a leaf node and assign it the class label that occurs most frequently in the training instances.

# HUNT'S ALGORITHM: LIMITATIONS

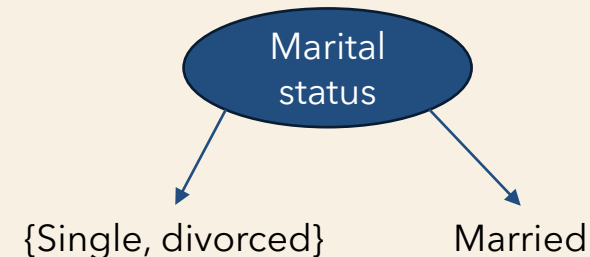
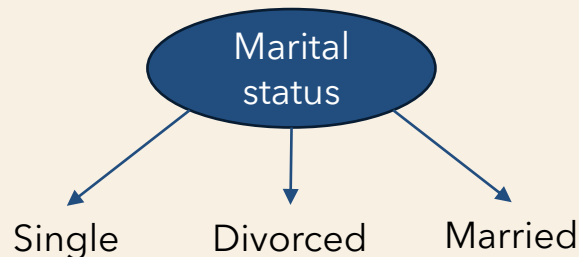
- Splitting criterion: the decision tree is built with a greedy strategy
  - The “best” attribute for the split is selected locally and this can lead to a stuck in a local optimum (not a global optimum)!
- Stopping criterion: the algorithm stops to expand a node only when all the training instances associated with it belongs to the same class → not always the best solution → early termination

# SPLITTING CRITERION: EXPRESSING ATTRIBUTE TEST CONDITIONS

- Binary attributes: only two possible outcomes (simplest case)

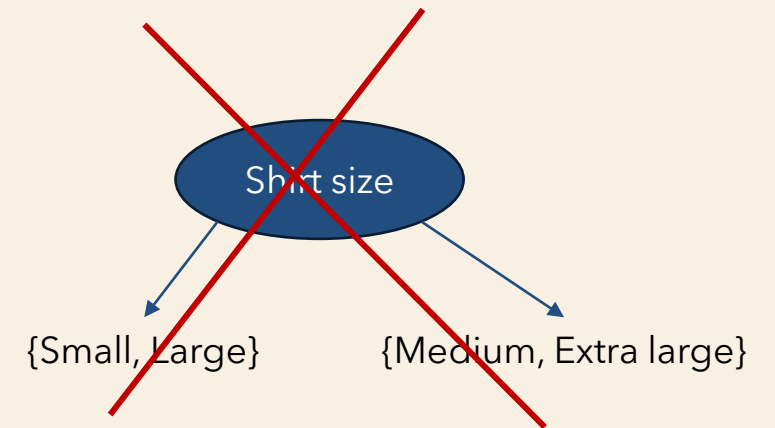
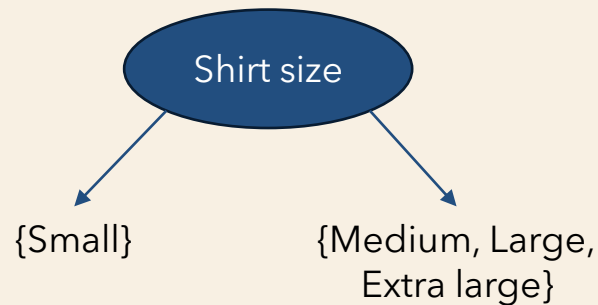
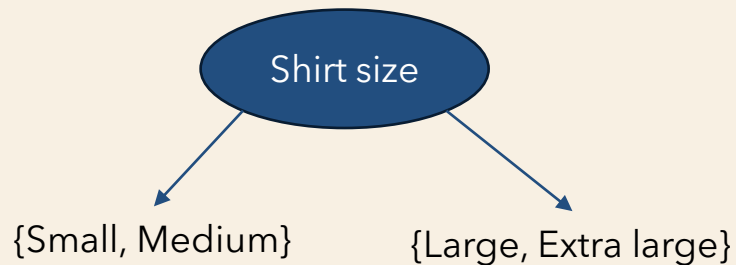


- Nominal attributes: it can have multiple values
  - a) Multi-way split
  - b) Binary split: aggregate the possible values into two groups:  $2^k - 1$  options



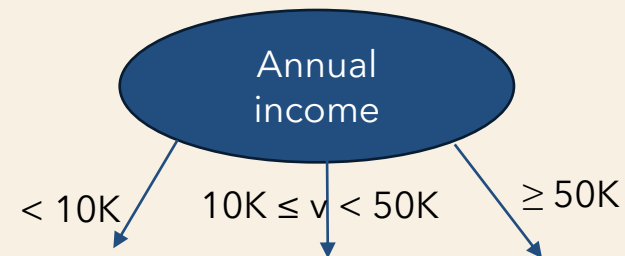
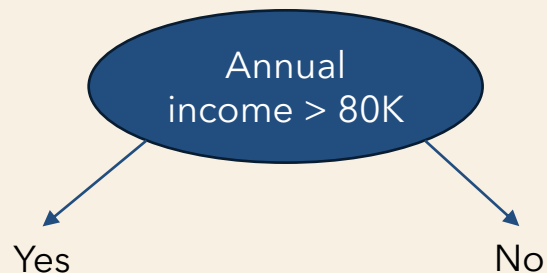
# SPLITTING CRITERION: EXPRESSING ATTRIBUTE TEST CONDITIONS

- Ordinal attributes: like the nominal attributes they can produce binary or multi-way splits
  - The possible aggregations are only those that do not violate the order property of the attribute values.



# SPLITTING CRITERION: EXPRESSING ATTRIBUTE TEST CONDITIONS

- Continuous attributes: the attribute test condition can be expressed as a comparison test producing a binary split (e.g.  $A < v$ ) or a multi-way split (range query) → discretization strategy
  - Multi-way split: value ranges should be mutually exclusive and cover the entire range of the attribute values.



# SELECT THE BEST ATTRIBUTE CONDITION

- Many measures to determine the goodness of an attribute test condition.
- IDEA: give preference to attribute test conditions that partition the training instances into purer subsets in the child nodes.
  - Pure = same class label
  - Pure nodes do not need to be expanded any further.
- Larger trees
  - They are more susceptible to model overfitting
  - They are more difficult to interpret
  - They require more training and test time

# IMPURITY MEASURES

- The impurity of a node measures how dissimilar the class labels are for the data instances belonging to a common node.
- Impurity of a node  $t$  (less is better):
  - Entropy =  $-\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$
  - Gini index =  $1 - \sum_{i=0}^{c-1} p_i(t)^2$
  - Classification error =  $1 - \max_i(p_i(t))$

All measures gives 0 impurity if the node contains only instances from a single class and maximum impurity when the node has equal proportion of instances from multiple classes

$p_i(t)$  is the relative frequency of training instances that belong to class  $i$  at node  $t$

$c$  is the total number of classes

$$0 \log_2 0 = 0$$

# IMPURITY MEASURES

Node N1	Count
Class = 0	0
Class = 1	6

$$\text{Entropy} = - [(0/6)\log_2(0/6) + (6/6)\log_2(6/6)] = 0$$

$$\text{Gini index} = 1 - [(0/6)^2 + (6/6)^2] = 0$$

$$\text{Error} = 1 - \max[0/6, 6/6] = 0$$

Node N2	Count
Class = 0	1
Class = 1	5

$$\text{Entropy} = - [(1/6)\log_2(1/6) + (5/6)\log_2(5/6)] = 0.650$$

$$\text{Gini index} = 1 - [(1/6)^2 + (5/6)^2] = 0.278$$

$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

Node N3	Count
Class = 0	3
Class = 1	3

$$\text{Entropy} = - [(3/6)\log_2(3/6) + (3/6)\log_2(3/6)] = 1$$

$$\text{Gini index} = 1 - [(3/6)^2 + (3/6)^2] = 0.5$$

$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

# IMPURITY MEASURES

Node N1	Count
Class = 0	0
Class = 1	6

$$\text{Entropy} = - [(0/6)\log_2(0/6) + (6/6)\log_2(6/6)] = 0$$

$$\text{Gini index} = 1 - [(0/6)^2 + (6/6)^2] = 0$$

$$\text{Error} = 1 - \max[0/6, 6/6] = 0$$

Node N2	Count
Class = 0	1
Class = 1	5

$$\text{Entropy} = - [(1/6)\log_2(1/6) + (5/6)\log_2(5/6)] = 0.650$$

$$\text{Gini index} = 1 - [(1/6)^2 + (5/6)^2] = 0.278$$

$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

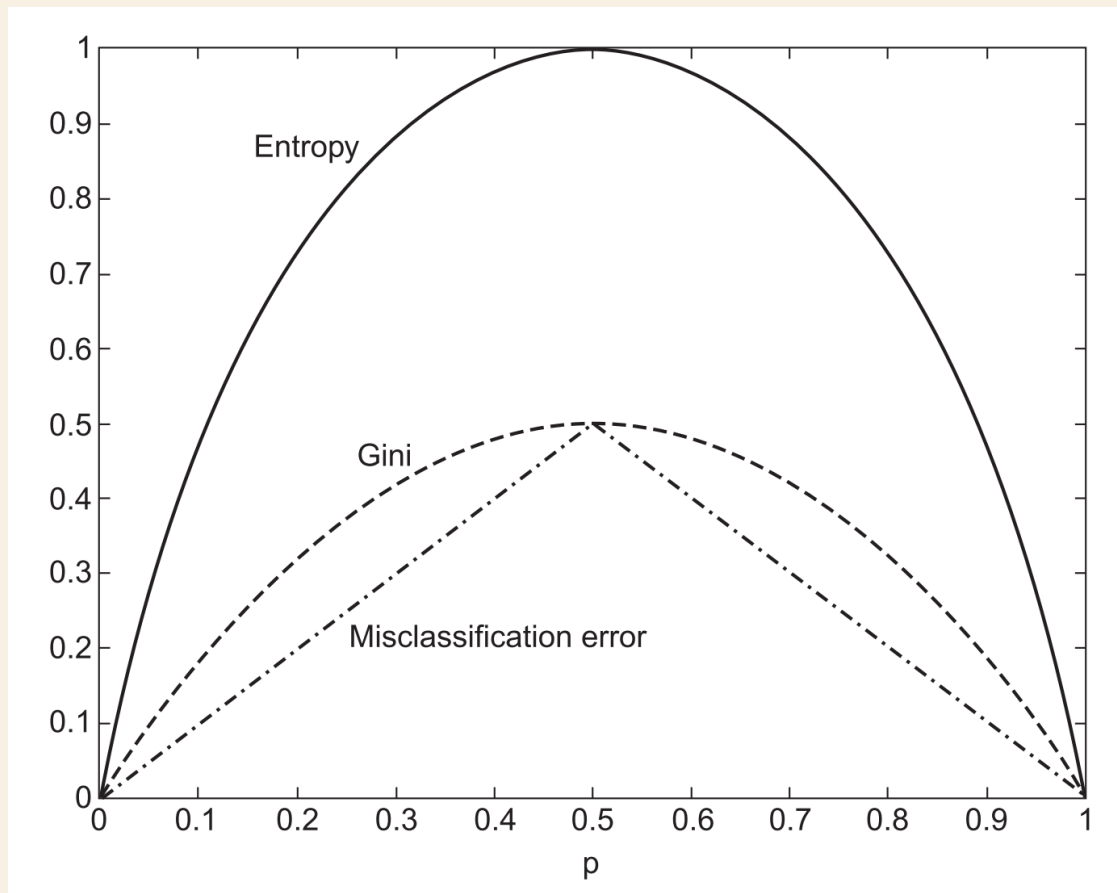
Node N3	Count
Class = 0	3
Class = 1	3

$$\text{Entropy} = - [(3/6)\log_2(3/6) + (3/6)\log_2(3/6)] = 1$$

$$\text{Gini index} = 1 - [(3/6)^2 + (3/6)^2] = 0.5$$

$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

# IMPURITY MEASURES: COMPARISON



Different values but  
consistency among the  
impurity measures!

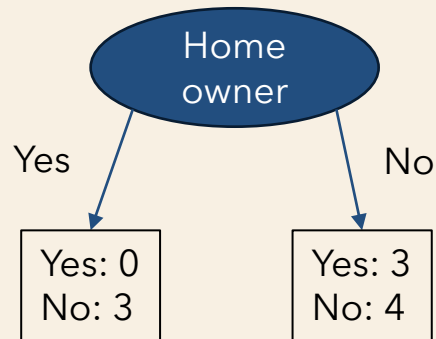
# COLLECTIVE IMPURITY

- Let us consider an attribute test condition that splits a node containing  $N$  training instances into  $k$  children  $\{v_1, v_2, \dots, v_n\}$ .
- Let  $N(v_j)$  the number of training instances associated with child node  $v_j$  whose impurity is  $I(v_j)$

- $$I(\text{children}) = \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j)$$

Collective impurity = average of the node impurity weighted by the number of instances associated to each node

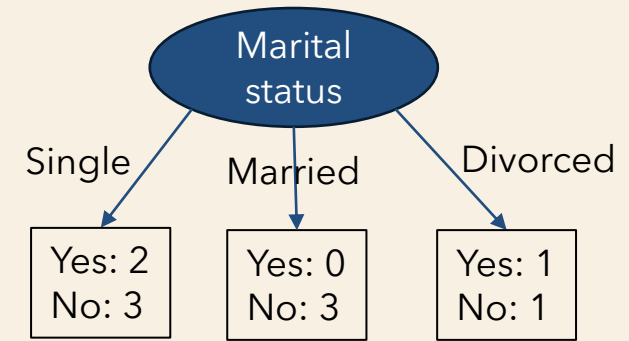
# COLLECTIVITY IMPURITY: WEIGHTED ENTROPY



$$I(\text{Home owner} = \text{Yes}) = - [(0/3)\log_2(0/3) + (3/3)\log_2(3/3)] = 0$$

$$I(\text{Home owner} = \text{No}) = - [(3/7)\log_2(3/7) + (4/7)\log_2(4/7)] = 0.985$$

$$I(\text{Home owner}) = 3/10 * 0 + 7/10 * 0.985 = 0.690$$



$$I(\text{MS} = \text{Single}) = - [(2/5)\log_2(2/5) + (3/5)\log_2(3/5)] = 0.971$$

$$I(\text{MS} = \text{Married}) = - [(0/3)\log_2(0/3) + (3/3)\log_2(3/3)] = 0$$

$$I(\text{MS} = \text{Divorced}) = - [(1/2)\log_2(1/2) + (1/2)\log_2(1/2)] = 1$$

$$I(M) = 5/10 * 0.971 + 3/10 * 0 + 2/10 * 1 = 0.686$$

Collective impurity

# IDENTIFYING THE BEST ATTRIBUTE TEST CONDITION

- To determine the goodness of an attribute test condition we need to compare the degree of impurity of the parent node (before splitting) with the weighted degree of impurity of the child nodes (after splitting).
- The larger the difference, the better the test condition.
- $\text{Gain} = I(\text{parent}) - I(\text{children})$

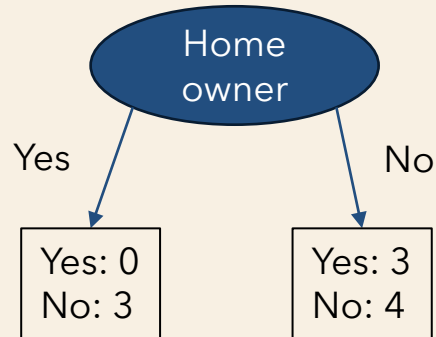
# IDENTIFYING THE BEST ATTRIBUTE TEST CONDITION

- Gain is always positive, since  $I(\text{parent}) \geq I(\text{children})$
- Splitting criterion: select the attribute which maximizes the gain
- If the impurity measure is the entropy, the gain is also called information gain  $\Delta_{\text{gain}}$

# IDENTIFYING THE BEST ATTRIBUTE TEST CONDITION: EXAMPLE

$$I(\text{parent}) = - [(3/10)\log_2(3/10) + (7/10)\log_2(7/10)] = 0.881$$

Before splitting 7 instances = No, 3 instances = Yes

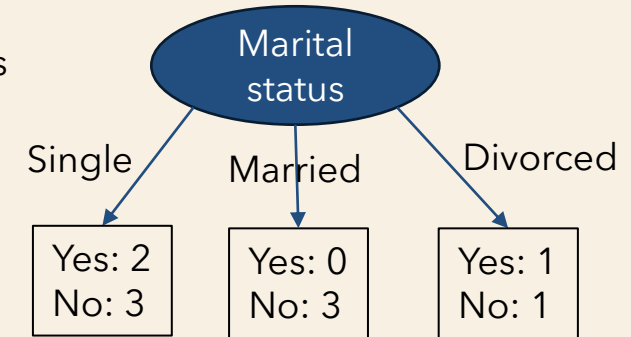


$$I(\text{Home owner} = \text{Yes}) = - [(0/3)\log_2(0/3) + (3/3)\log_2(3/3)] = 0$$

$$I(\text{Home owner} = \text{No}) = - [(3/7)\log_2(3/7) + (4/7)\log_2(4/7)] = 0.985$$

$$I(\text{Home owner}) = 3/10 * 0 + 7/10 * 0.985 = 0.690$$

$$\Delta_{\text{gain}} = I(\text{parent}) - I(\text{Home owner}) = 0.881 - 0.690 = 0.191$$



$$I(\text{MS} = \text{Single}) = - [(2/5)\log_2(2/5) + (3/5)\log_2(3/5)] = 0.971$$

$$I(\text{MS} = \text{Married}) = - [(0/3)\log_2(0/3) + (3/3)\log_2(3/3)] = 0$$

$$I(\text{MS} = \text{Divorced}) = - [(1/2)\log_2(1/2) + (1/2)\log_2(1/2)] = 1$$

$$I(\text{M}) = 5/10 * 0.971 + 3/10 * 0 + 2/10 * 1 = 0.686$$

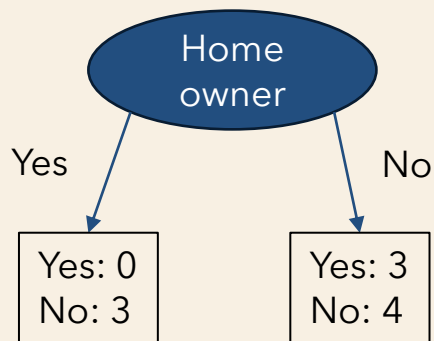
$$\Delta_{\text{gain}} = I(\text{parent}) - I(\text{MS}) = 0.881 - 0.686 = 0.195$$

# SPLITTING BASED ON GINI: CATEGORICAL ATTRIBUTES

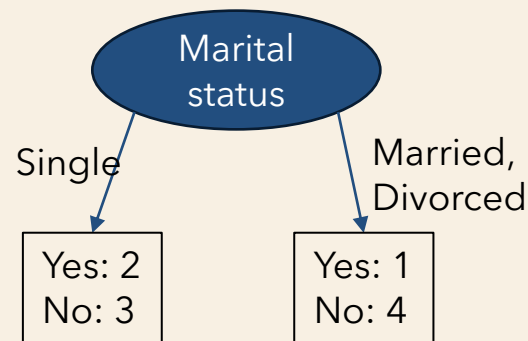
$$\text{Gini index} = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

Parent	
No	7
Yes	3
Gini = 0.420	

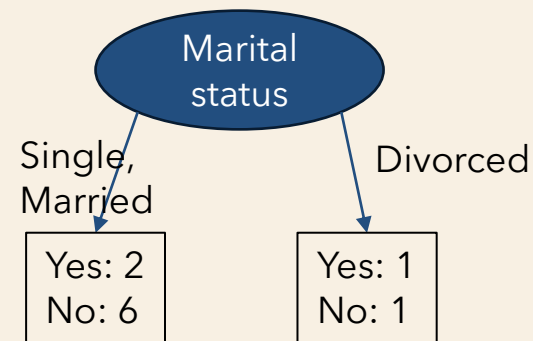
$$\text{Gini} = 1 - [(7/10)^2 + (3/10)^2] = 0.420$$



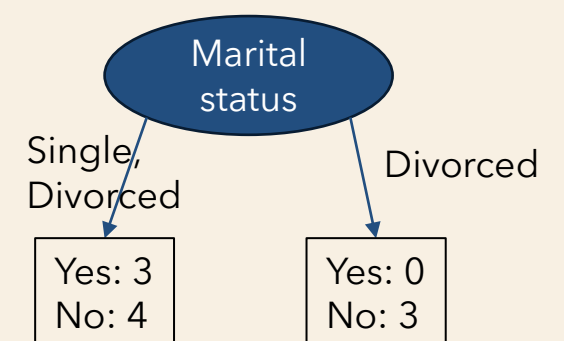
Gini = 0.343



Gini = 0.400



Gini = 0.400



Gini = 0.343

# SPLITTING BASED ON GINI: QUANTITATIVE ATTRIBUTES

- Annual income  $\leq \tau$
- $\tau$  can be chosen equal to any value between the minimum and the maximum
- It is sufficient to only consider the values observed in the training set as candidate splitting positions
- We can compute the Gini index at each candidate position and chose the  $\tau$  that produces the lowest value.

# SPLITTING BASED ON GINI: QUANTITATIVE ATTRIBUTES

- $O(N)$  operations for computing the Gini index for each possible position
- $O(N^2)$  operations for computing the Gini index and perform the comparisons → brute force approach
  - For each possible value ( $N$ ), scan each instance ( $N$ ) for counting the number instances  $<$  and  $\geq$
- $O(N \log N)$  if we previously sort the training instances based on the value in the splitting attribute → the candidate split positions is given by the **midpoints** between every two adjacent sorted values
  - Sort the values in  $O(N \log N)$  + scan each value for updating a count matrix and computing the Gini index  $O(N)$

# SPLITTING BASED ON GINI: QUANTITATIVE ATTRIBUTES

Class		No		No		No		Yes		Yes		Yes		No		No		No		No			
Sorted Values Split Positions	→	Annual Income (in '000s)																					
		60		70		75		85		90		95		100		120		125		220			
	→	55		65		72.5		80		87.5		92.5		97.5		110		122.5		172.5		230	
		<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes		0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No		0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini		0.420		0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400		0.420	

- The best split position is the one that produces the lowest Gini index.

# SPLITTING BASED ON GINI: QUANTITATIVE ATTRIBUTES

Sorted Values Split Positions	Class	No		No		No		Yes		Yes		Yes		No		No		No		No				
	Annual Income (in '000s)																							
	60		70		75		85		90		95		100		120		125		220					
	55		65		72.5		80		87.5		92.5		97.5		110		122.5		172.5		230			
	<=		>		<=		>		<=		>		<=		>		<=		>		<=		>	
	Yes		0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No		0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0	
Gini		0.420		0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400		0.420		

- Further reduction if we consider as candidate splits only positions where the instances change their label.

# INFORMATION GAIN

- One potential limitation of the entropy and Gini index is that they tend to favor qualitative attributes with larger number of distinct values.
- Having a low impurity value alone is insufficient to find a good attribute test condition for a node.
- Having a greater number of children can make the decision tree more complex and consequently more susceptible to overfitting
  - The number of children produced by the splitting attribute should also be taken into consideration while deciding the best attribute test condition.

# INFORMATION GAIN

- Solutions:

- Generate only binary decision trees (CART).

- Gain Ratio (C4.5)

- $$\text{Gain Ratio} = \frac{\Delta_{info}}{\text{split info}} = \frac{\text{Entropy}(\text{parent}) - \text{Entropy}(\text{children})}{-\sum_{i=1}^k \frac{N(v_i)}{N} \log_2 \frac{N(v_i)}{N}}$$
- The split information measures the entropy of splitting a node into its child nodes and evaluates if the split results in a larger number of equally-sized child nodes or not.
- If each partition has the same number of instances  $N(v_i)/N = 1/k \rightarrow \text{split information} = \log(k)$
- If the number of splits (k) is also large  $\rightarrow$  the gain ratio is reduced

# ALGORITHM FOR DECISION TREE INDUCTION (CONSTRUCTION)

**Algorithm 3.1** A skeleton decision tree induction algorithm.

```
TreeGrowth ( $E, F$ )  $\leftarrow$ 
1: if stopping_cond( $E, F$ ) = true then
2:   leaf = createNode().
3:   leaf.label = Classify( $E$ ).
4:   return leaf.
5: else
6:   root = createNode().
7:   root.test_cond = find_best_split( $E, F$ ).  $\leftarrow$ 
8:   let  $V = \{v \mid v \text{ is a possible outcome of } \textit{root.test\_cond} \}$ .
9:   for each  $v \in V$  do
10:     $E_v = \{e \mid \textit{root.test\_cond}(e) = v \text{ and } e \in E\}$ .
11:    child = TreeGrowth( $E_v, F$ ).  $\leftarrow$ 
12:    add child as descendent of root and label the edge ( $\textit{root} \rightarrow \textit{child}$ ) as  $v$ .
13:  end for
14: end if
15: return root.
```

$E$  = set of training instances

$F$  = attribute set

Recursively select the best attribute to split the data ...

... and expand the nodes of the tree

# ALGORITHM FOR DECISION TREE INDUCTION (CONSTRUCTION)

**Algorithm 3.1** A skeleton decision tree induction algorithm.

TreeGrowth ( $E, F$ )

```
1: if stopping_cond( $E, F$ ) = true then
2:   leaf = createNode().
3:   leaf.label = Classify( $E$ ).
4:   return leaf.
5: else
6:   root = createNode().
7:   root.test_cond = find_best_split( $E, F$ ).
8:   let  $V = \{v \mid v \text{ is a possible outcome of } root.test\_cond \}$ .
9:   for each  $v \in V$  do
10:     $E_v = \{e \mid root.test\_cond(e) = v \text{ and } e \in E\}$ .
11:    child = TreeGrowth( $E_v, F$ ).
12:    add child as descendent of root and label the edge ( $root \rightarrow child$ ) as  $v$ .
13:   end for
14: end if
15: return root.
```

Extend the decision tree by creating a new node: each node has a test condition (node.test\_cond) or a label (node.label)

# ALGORITHM FOR DECISION TREE INDUCTION (CONSTRUCTION)

**Algorithm 3.1** A skeleton decision tree induction algorithm.

TreeGrowth ( $E, F$ )

```
1: if stopping_cond( $E, F$ ) = true then
2:   leaf = createNode().
3:   leaf.label = Classify( $E$ ).
4:   return leaf.
5: else
6:   root = createNode().
7:   root.test_cond = find_best_split( $E, F$ ). ←
8:   let  $V = \{v \mid v \text{ is a possible outcome of } root.test\_cond \}$ .
9:   for each  $v \in V$  do
10:     $E_v = \{e \mid root.test\_cond(e) = v \text{ and } e \in E\}$ .
11:    child = TreeGrowth( $E_v, F$ ).
12:    add child as descendent of root and label the edge ( $root \rightarrow child$ ) as  $v$ .
13:  end for
14: end if
15: return root.
```

It determines the best attribute test condition for partitioning the training instances associated with a node by using an impurity measure (e.g., entropy, Gini index)

# ALGORITHM FOR DECISION TREE INDUCTION (CONSTRUCTION)

**Algorithm 3.1** A skeleton decision tree induction algorithm.

TreeGrowth ( $E, F$ )

```
1: if stopping_cond( $E, F$ ) = true then
2:   leaf = createNode().
3:   leaf.label = Classify( $E$ ). ←
4:   return leaf.
5: else
6:   root = createNode().
7:   root.test_cond = find_best_split( $E, F$ ).
8:   let  $V = \{v \mid v \text{ is a possible outcome of } root.test\_cond \}$ .
9:   for each  $v \in V$  do
10:     $E_v = \{e \mid root.test\_cond(e) = v \text{ and } e \in E\}$ .
11:    child = TreeGrowth( $E_v, F$ ).
12:    add child as descendent of root and label the edge ( $root \rightarrow child$ ) as  $v$ .
13:  end for
14: end if
15: return root.
```

It determines the class label to be assigned to a leaf node.

$$leaf.label = \underset{i}{\operatorname{argmax}} p(i|t)$$

$p(i|t)$  = fraction of training instances from class  $i$  assigned to node  $t$

label = the one that occurs most frequently in the training instances

# ALGORITHM FOR DECISION TREE INDUCTION (CONSTRUCTION)

**Algorithm 3.1** A skeleton decision tree induction algorithm.

TreeGrowth ( $E, F$ )

```
1: if stopping_cond( $E, F$ ) = true then ←
2:   leaf = createNode().
3:   leaf.label = Classify( $E$ ).
4:   return leaf.
5: else
6:   root = createNode().
7:   root.test_cond = find_best_split( $E, F$ ).
8:   let  $V = \{v \mid v \text{ is a possible outcome of } root.test\_cond \}$ .
9:   for each  $v \in V$  do
10:     $E_v = \{e \mid root.test\_cond(e) = v \text{ and } e \in E\}$ .
11:    child = TreeGrowth( $E_v, F$ ).
12:    add child as descendent of root and label the edge ( $root \rightarrow child$ ) as  $v$ .
13:   end for
14: end if
15: return root.
```

stopping\_cond() checks if all instances have identical class label or attribute values, or a threshold could be used to prevent the data fragmentation problem (= too few training instances on a leaf node to make statistically significant decisions)

# CHARACTERISTICS OF DECISION TREES

## Applicability

- It is a nonparametric approach which does not require any prior assumption about the probability distribution governing the classes and attributes of the data.
  - It can be applied to a great vary of datasets.
- It is applicable to both categorical and continuous values.
- It can deal with both binary and multiclass problems.
- The built models are relatively easy to interpret.

# CHARACTERISTICS OF DECISION TREES

## Expressiveness

- It can encode any function of discrete-valued attributes.
- Every leaf node represent a combination of attributes.

## Computational efficiency

- Decision tree algorithms apply a heuristic to build the trees.
- Once a decision tree has been built, classifying a record is extremely fast.

# CHARACTERISTICS OF DECISION TREES

## Handling missing values

- Probabilistic split (C4.5)
  - Distribute the data instances based on the probability that the missing attribute attributes have a particular value
- Surrogate split method (CART)
  - Instances with a missing value are assigned to one of the child nodes based on the value of another non-missing surrogate attribute.
- Separate class method (CHAID)
  - Missing values are considered as a separate categorical value.
- Other strategies
  - E.g., they are discarded during pre-processing.

# CHARACTERISTICS OF DECISION TREES

## Handling interactions among attributes

- Attributes are considered interacting if they are able to distinguish between classes when used together, but individually they provide little or no information
- Trees can perform poorly when there are interacting attributes

## Handling irrelevant attributes

- An attribute is irrelevant if it is not useful for the classification task.
- The presence of a small number of irrelevant attributes will not impact the decision tree construction process.

# CHARACTERISTICS OF DECISION TREES

## Handling redundant attributes

- An attribute is redundant if it is strongly correlated with another attribute.
- Redundant attributes show similar gains in purity → only one of them will be selected.

## Choice of the impurity measure

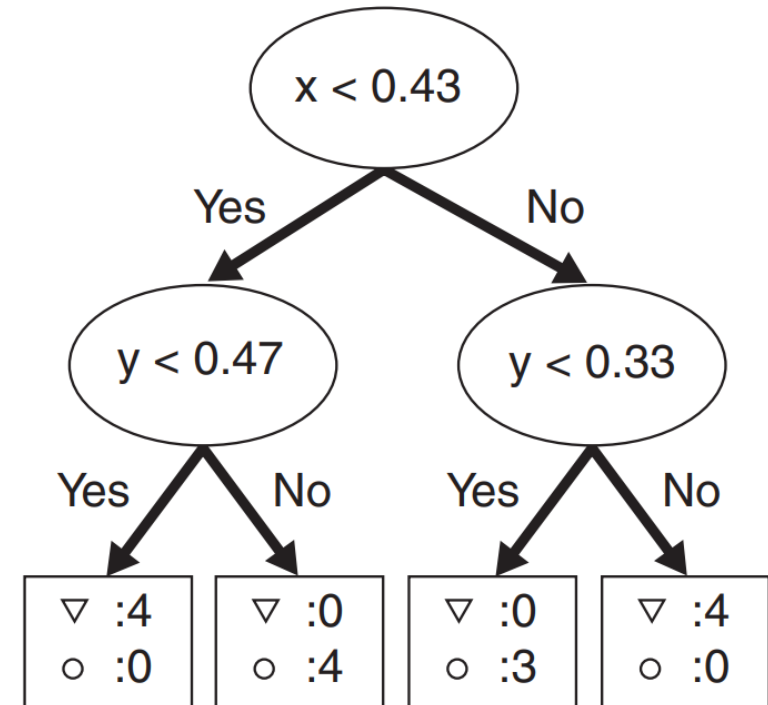
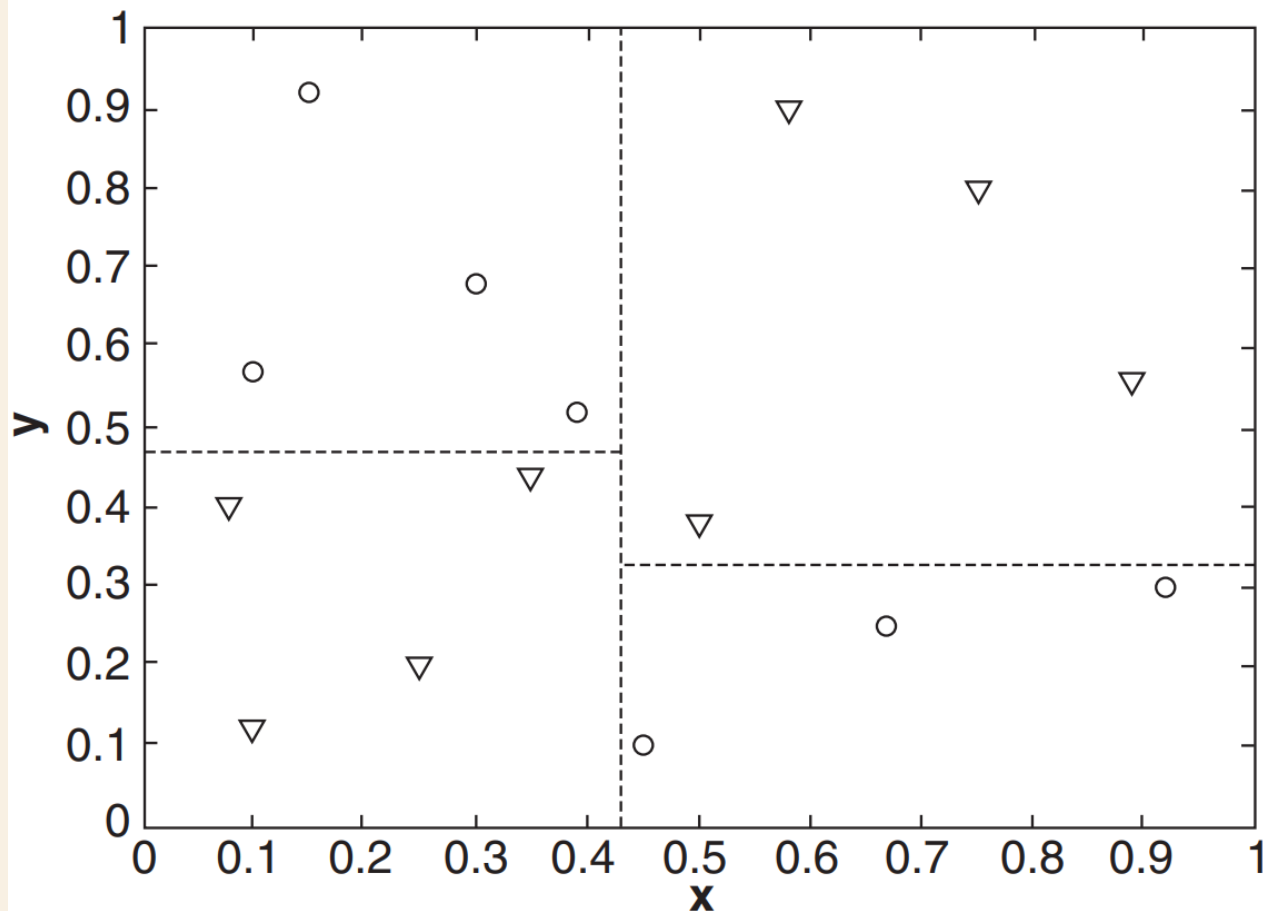
- It has little effect on the performance of the decision tree, since they are all consistent with each other.

# CHARACTERISTICS OF DECISION TREES

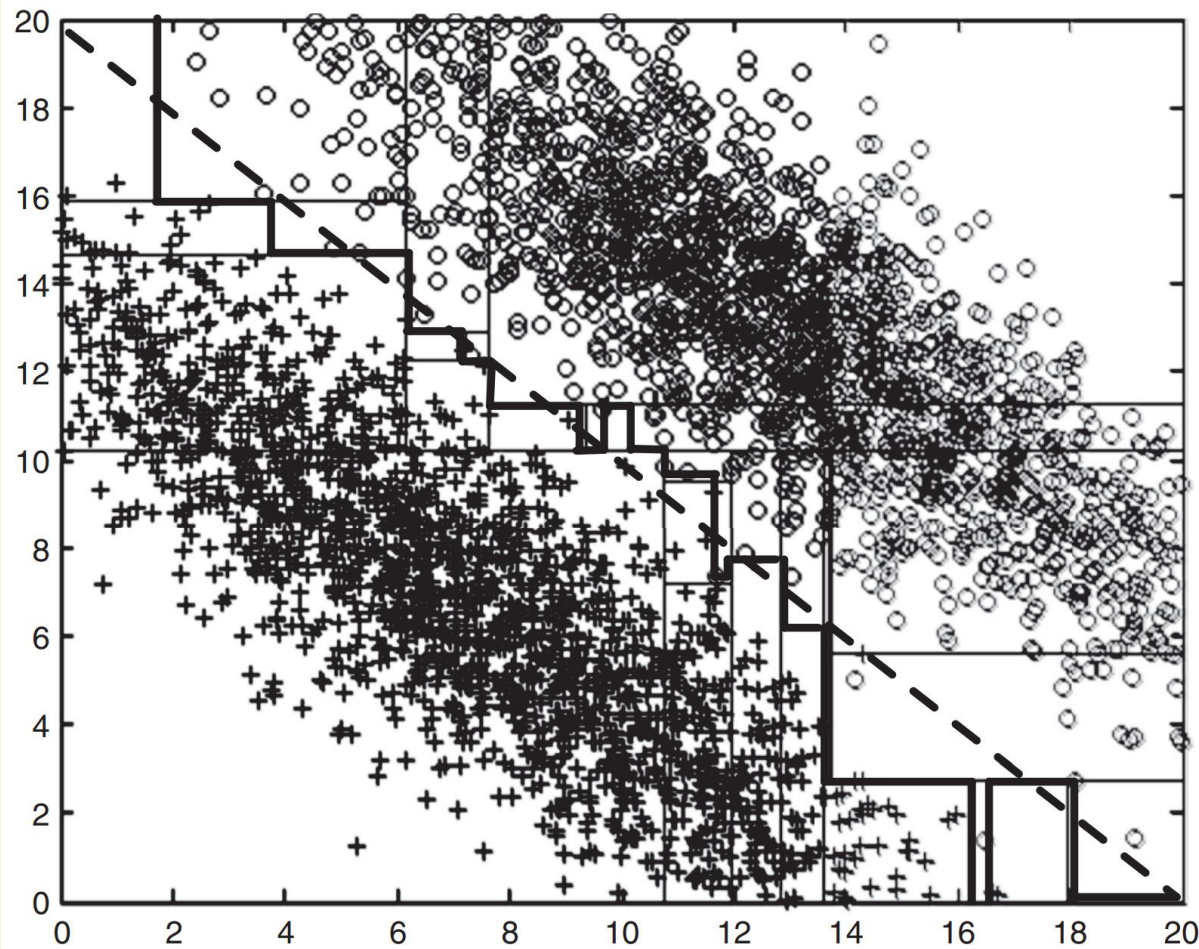
## Rectilinear splits

- The test conditions involves a single attribute at a time
- The tree-growing procedure can be viewed as the process of partitioning the attribute space into disjoint regions until each of them contains only records of the same class.
- The border of each region is called decision boundary
- The decision boundaries are rectilinear, i.e., parallel to the coordinate axes.

# CHARACTERISTICS OF DECISION TREES



# CHARACTERISTICS OF DECISION TREES



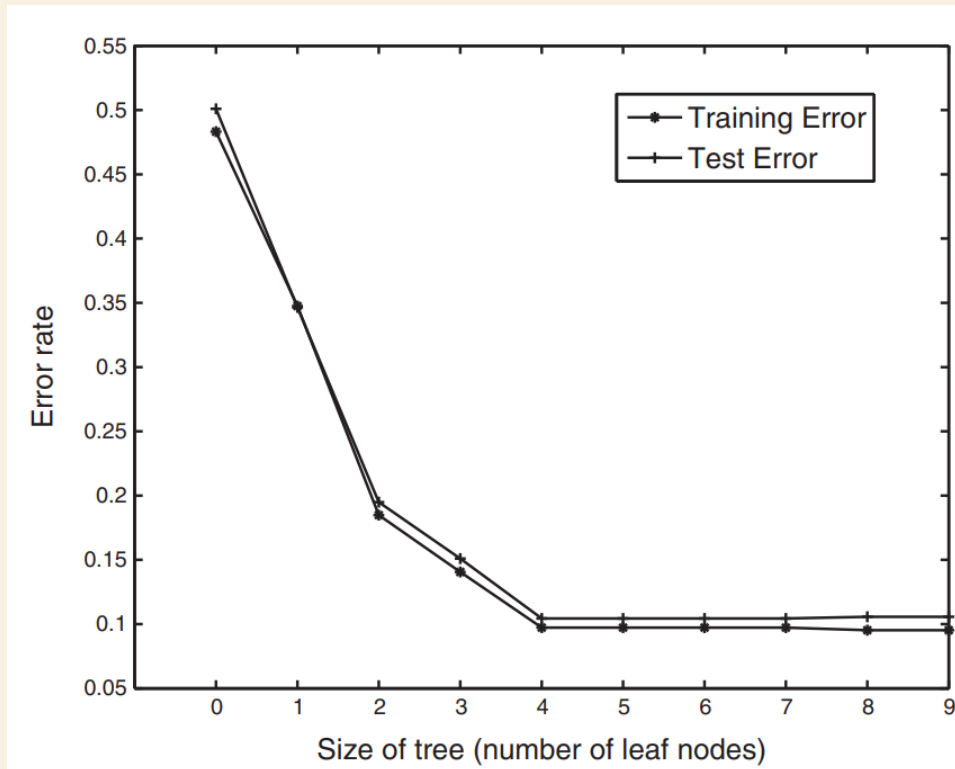
## Problem

- The true decision boundary is represented by the diagonal line.
- An oblique decision tree could be more expressive and can produce a more compact tree, but it is more computational expensive to build!

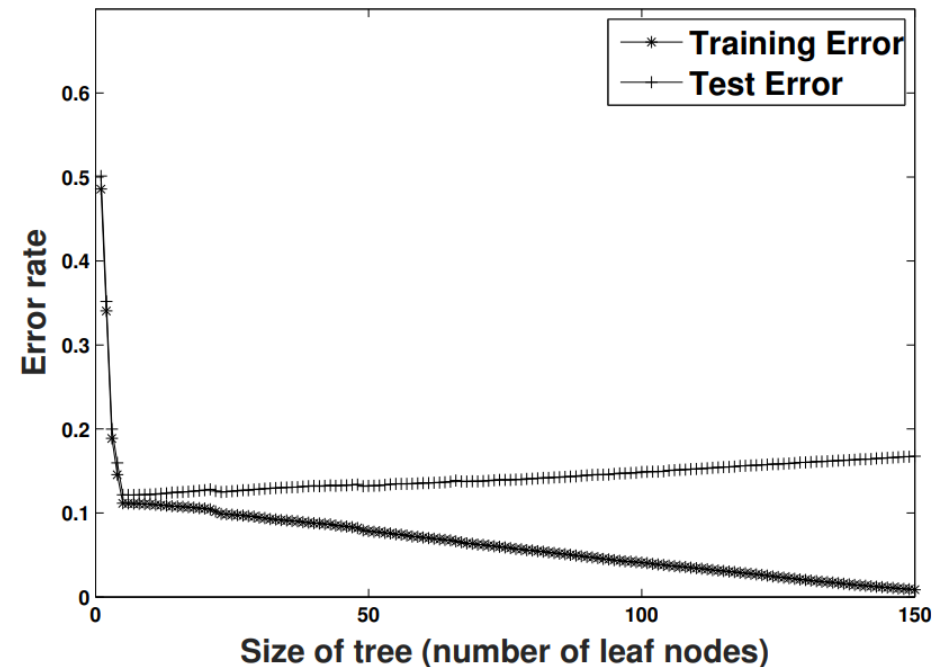
# MODEL OVERFITTING/UNDERFITTING

- Model overfitting happens when the model fits well over the training data, but it show poor generalization performances (low accuracy on the test data).
- Model underfitting happens when the model is too simplistic and thus incapable of fully representing the true relationship between the attributes and the class labels.

# MODEL OVERFITTING



Ok: the training and the test error rates are quite close to each other → the performances of the training set are quite representative.



Overfitting: the training error decreases while the test error increases!

# MODEL OVERFITTING: CAUSES

- Model overfitting happens when an overly complex model is selected that captures specific patterns in the training data but fails to learn the true nature of relationships between attributes and class labels in the overall data.

## Limited training size

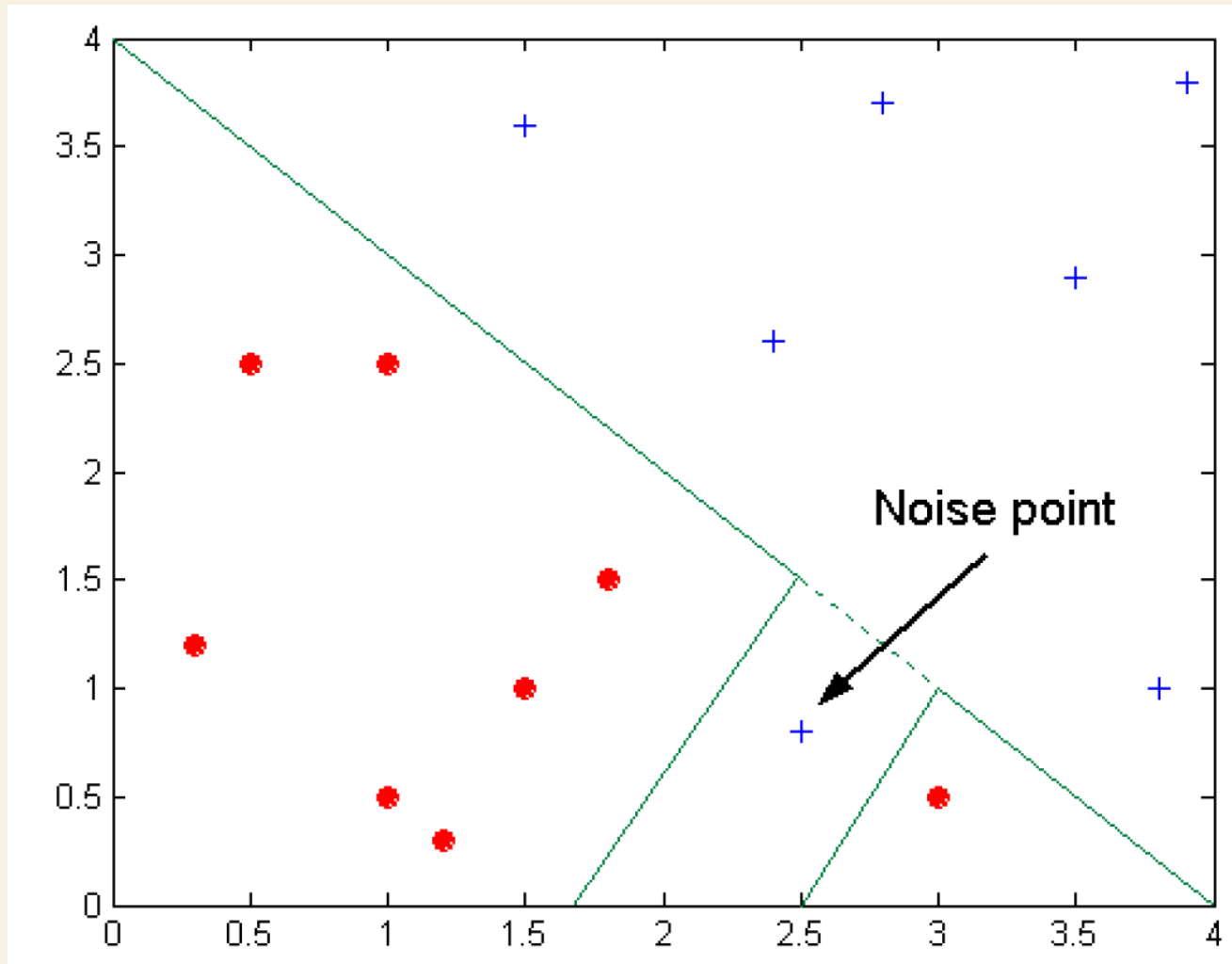
- The dataset can only provide a limited representation of the overall data → the learned pattern does not fully represent the true pattern in the data
- Remedy: increase the training size

# MODEL OVERFITTING: CAUSES

## High model complexity

- An overly complex model has a tendency to learn specific patterns in the training set that do not generalize well over unseen instances
- If the number of parameter combinations is large but the number of training instances is small, it is highly likely to obtain overfitting

# MODEL OVERFITTING: CAUSES



Decision boundary is distorted by a noise point

# MODEL SELECTION

- We want to select the model that shows lowest generalization error rate.
- Training error cannot be reliable used as the sole criterion for model selection.
- Generic approach:
  - use a validation set
  - Incorporating the model complexity in the error computation

# VALIDATION SET

- Evaluate the model on a separate validation set that has not been used for training the model.
- Given a training set  $D_{\text{train}}$ , partition it into two smaller parts:
  - $(2/3) D_{\text{tr}}$  : used for the training
  - $(1/3) D_{\text{val}}$  : used for computing the validation error rate
- The model that shows the lowest value of  $\text{error}_{\text{val}}(m)$  can be selected as the preferred model.

# MODEL COMPLEXITY

- The chance for model overfitting increases as a model becomes more complex.
- Given two models with the same errors, the simpler model is preferred over the more complex one (principle of parsimony).

$$\text{gen.err}(m) = \text{train.error}(m, D.\text{train}) + \alpha \times \text{complexity}(m)$$

# MODEL COMPLEXITY

- The complexity of a decision tree can be estimated as the ratio of the number of leaf nodes to the number of training instances.
- $k / N_{\text{train}}$ 
  - $k$  is the number of leaf nodes
  - $N_{\text{train}}$  is the number of training instances
- For a larger training size, we can learn a decision tree with larger number of leaf nodes without it becoming overly complex

# MODEL SELECTION: PRE-PRUNING (EARLY STOPPING RULE)

- Stop the algorithm before generating a fully-grown tree
- Typical stopping conditions for a node
  - Stop if all instances belong to the same class
  - Stop if all the attribute values are the same
- More restrictive conditions
  - Stop if number of instances is less than some user-specified threshold
  - Stop if class distribution of instances are independent of the available features
  - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain)

# MODEL SELECTION: PRE-PRUNING (EARLY STOPPING RULE)

- Advantage:
  - It avoids the computation associated with generating overly complex subtrees that overfit the training data.
- Disadvantage:
  - Optima subtrees could not be reached due to the greedy nature of the decision tree induction.

# MODEL SELECTION: POST-PRUNING

- Grow decision tree to its entirety
- Trim the nodes of the decision tree in a bottom-up fashion
- Subtree replacement: if generalization error improves after trimming, replace sub-tree by a leaf node.
- Subtree raising: the class label of leaf node is determined from majority class of instances in the sub-tree
- Advantage:
  - Better results because the pruning is based on a fully grown tree.
- Disadvantage:
  - Additional computations for growing the full tree.

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark purple with white concentric circles. The bottom-right square is bright pink with white concentric circles. A large, light gray circle is partially visible behind the grid.

# **RULE-BASED CLASSIFIER**

# RULE-BASED CLASSIFIER

- It uses a collection of "if ... then ..." rules (rule set) to classify data instances.
- Each rule  $r_i$  can be expressed as:  $r_i : (\text{condition}_i) \rightarrow y_i$ 
  - $\text{condition}_i$  is the antecedent or precondition
    - Conjunction of attribute test conditions:  $(A_1 \text{ op } v_1) \wedge (A_2 \text{ op } v_2) \wedge \dots \wedge (A_n \text{ op } v_n)$
    - $(A_i, v_i)$  are attribute-value pairs and op is a comparison operator  $\{=, \neq, >, \geq, <, \leq\}$
  - $y_i$  is the rule consequent (= the predicted class)

# RULE-BASED CLASSIFIER

- A rule  $r$  covers a data instance  $x$  if the precondition of  $r$  matches the attributes of  $x$ . ( $r$  is triggered or fired)
- The quality of a classification rule can be evaluated in terms of:
  - Coverage: given a dataset  $D$  and a classification rule  $r: A \rightarrow y$ , the coverage of the rule is the fraction of instances in  $D$  that trigger the rule  $r$ .

$$Coverage(r) = \frac{|A|}{|D|}$$

- Accuracy: given a dataset  $D$  and a classification rule  $r: A \rightarrow y$ , the accuracy or confidence is the fraction of instances triggered by  $r$  whose class labels are equal to  $y$ .

$$Accuracy(r) = \frac{|A \cap y|}{|A|}$$

# RULE-BASED CLASSIFIER

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

# RULE-BASED CLASSIFIER

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

Coverage(R1) =  $3/20 = 15\%$

Accuracy(R1) =  $3/3 = 100\%$

# RULE-BASED CLASSIFIER

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

Coverage(R2) = 5/20 = 25%

Accuracy(R1) = 3/5 = 60%

# RULE-BASED CLASSIFIER

- A rule-based classifier classifies a test instance based on the rule triggered by the instance.

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 → mammal

A turtle triggers both R4 and R5 → conflict classes → ??

A dogfish shark triggers none of the rules → ??

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

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

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

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

R5: (Live in Water = sometimes) → Amphibians

# RULE-BASED CLASSIFIER: CHARACTERISTICS OF THE RULES

- Mutually exclusive rule set
  - The rules in a rule set  $R$  are mutually exclusive if no two rules in  $R$  are triggered by the same instance.
    - Every instance is covered by at most one rule in  $R$ .
- Exhaustive rule set
  - A rule set  $R$  has exhaustive coverage if there is a rule for each combination of the attribute values.
    - Every instance is covered by at least one rule in  $R$ .

# RULE-BASED CLASSIFIER: CHARACTERISTICS OF THE RULES

- Example of mutually exclusive and exhaustive rule set.

R1: (Blood type = cold)  $\rightarrow$  Non-mammals

R2: (Blood type = warm)  $\wedge$  (Give birth = yes)  $\rightarrow$  Mammals

R2: (Blood type = warm)  $\wedge$  (Give birth = no)  $\rightarrow$  Non-mammals

- Together these two properties ensures that every instance is covered by exactly one rule.

# RULE-BASED CLASSIFIER: CHARACTERISTICS OF THE RULES

- If the rule set is not exhaustive, then a default rule  $r_d: () \rightarrow y_d$  must be added to cover the remaining cases.
  - Empty antecedent: it is triggered when all the other rules have failed
  - $y_d$  is the default class = the majority class not covered by the existing rules
- If the rule set is not exclusive, we can define an order inside the rule set, thus rules in  $R$  are ranked in decreasing order of their priority (decision list).
  - An instance is classified by using the highest-ranked rule that covers the instance.

# RULE-BASED CLASSIFIER: DIRECT METHODS FOR RULE EXTRACTION

- Direct methods extract rules directly from the data.
  - Examples: RIPPER, CN2, Holte's 1R
- RIPPER:
  - Scales almost linearly with the number of training instances and is suited for building models in case of imbalanced class distributions
  - It uses a sequential covering algorithm to extract rules directly from data.
    - Rules are grown in a greedy fashion one class at time.

# RULE-BASED CLASSIFIER: RIPPER

- Binary problem:
  - The majority class is the default one and the algorithm learns rules to detect instances from the minority class.
- Multiclass problem:
  - The classes are ordered in according to their prevalence in the training set.
    - $(y_1, y_2, \dots, y_c)$ :  $y_1$  is the least prevalent,  $y_c$  is the most prevalent class
    - All training instances belonging to  $y_1$  are labelled as positive instances, while all the others as negative instances. The sequential covering algorithm learns a set of rules to discriminate the positive from the negative instances.
    - Then all instances from  $y_2$  are labelled as positive instances, while all the others as negative, and the previous mechanism is repeated. The process finishes when only one class  $y_c$  remains (default class).

# RULE-BASED CLASSIFIER: RIPPER

## SEQUENTIAL COVERING ALGORITHM

---

**Algorithm 5.1** 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  $\{y_1, y_2, \dots, y_k\}$ .  $\longrightarrow$  Most prevalent
  - 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:      $r \leftarrow \text{Learn-One-Rule}(E, A, y)$ .  $\longrightarrow$  Grow a rule using the Learn-One-Rule function
  - 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  $R$ .
-

# RULE-BASED CLASSIFIER: RIPPER

## LEARN-ONE-RULE FUNCTION

- Finding an optimal rule is computationally expensive.
- The learn-one-rule function solves the problem by growing the rules in a greedy fashion.
  1.  $r: \{\} \rightarrow +$
  2. Refine the rule until a certain stopping criterion is met
- FOIL's information gain measure to choose the best conjunction to be added to the rule antecedent
  - The measure takes into consideration both the gain in accuracy and support of a candidate rule.

# RULE-BASED CLASSIFIER: RIPPER

## INFORMATION GAIN

- FOIL's information gain
  - $r: A \rightarrow +$  initially covers  $p_0$  positive examples and  $n_0$  negative examples
  - $r': A \wedge B \rightarrow +$  covers  $p_1$  positive examples and  $n_1$  negative examples
- FOIL's information gain =  $p_1 \times \left( \log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0} \right)$

# RULE-BASED CLASSIFIER: RIPPER

## INFORMATION GAIN

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	Mammals
python	cold-blooded	scales	no	no	no	no	yes	Reptiles
salmon	cold-blooded	scales	no	yes	no	no	no	Fishes
whale	warm-blooded	hair	yes	yes	no	no	no	Mammals
frog	cold-blooded	none	no	semi	no	yes	yes	Amphibians
komodo dragon	cold-blooded	scales	no	no	no	yes	no	Reptiles
bat	warm-blooded	hair	yes	no	yes	yes	yes	Mammals
pigeon	warm-blooded	feathers	no	no	yes	yes	no	Birds
cat	warm-blooded	fur	yes	no	no	yes	no	Mammals
guppy	cold-blooded	scales	yes	yes	no	no	no	Fishes
alligator	cold-blooded	scales	no	semi	no	yes	no	Reptiles
penguin	warm-blooded	feathers	no	semi	no	yes	no	Birds
porcupine	warm-blooded	quills	yes	no	no	yes	yes	Mammals
eel	cold-blooded	scales	no	yes	no	no	no	Fishes
salamander	cold-blooded	none	no	semi	no	yes	yes	Amphibians

- target class = mammals
- $r: \{\} \rightarrow \text{mammals}$ 
  - $p_0 = 5$
  - $n_0 = 10$
  - $\text{Accuracy}(r) = 5/15 = 0.33$



Candidate rule	p1	n1	Accuracy	Info Gain
{Skin cover = hair} $\rightarrow$ mammals	3	0	3/3=1.000	4.755
{Body temperature = warm-blooded} $\rightarrow$ mammals	5	2	5/7=0.714	5.498
{Has legs = no} $\rightarrow$ mammals	1	4	1/5=0.200	-0.737

# RULE-BASED CLASSIFIER: RIPPER

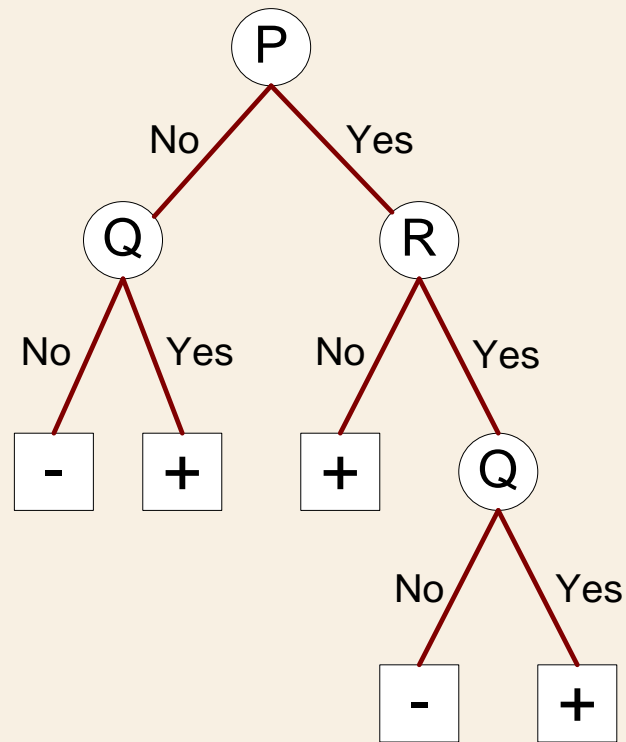
## RULE PRUNING

- The rules generated by the Learn-One-Rule function can be pruned to improve their generalization errors.
- The validation set is used to prune the rules:
  - $p$  ( $n$ ) is the number of positive (negative) examples in the validation set covered by the rule
  - $(p-n)/(p+n)$ : if the metric improves after the pruning, the conjunction is removed
- The pruning is performed starting from the last conjunction added to the rule:  $ABCD \rightarrow y$ , first check if  $D$  can be removed, then  $CD$  and finally  $BCD$

# RULE-BASED CLASSIFIER: INDIRECT METHODS FOR RULE EXTRACTION

- Indirect methods extract classification rules from a more complex classification method: e.g., from a decision tree.
- Every path from the root to a leaf node in a decision tree can be expressed as a classification rule.
- The test conditions encountered along the path form the conjunction of the rule antecedent, while the class label at the leaf node is assigned to the rule consequent.

# RULE-BASED CLASSIFIER: INDIRECT METHODS FOR RULE EXTRACTION



## Rule Set

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) ==> +

# RULE-BASED CLASSIFIERS: CHARACTERISTICS

## Expressiveness

- Very similar to a decision tree, since a decision tree can be represented by a set of mutually exclusive and exhaustive rules (rectilinear partitions of the attributes).
- Multiple rules can be triggered for a given instance, thus more complex models can be learnt.

# RULE-BASED CLASSIFIERS: CHARACTERISTICS

## Redundant attributes

- Their presence is handled because once an attribute has been used, the remaining redundant attributes will introduce a little or no information gain and would be ignored.

## Interacting attribute

- Poor performances like decision tree.

## Missing values in the test set

- They are not treated well, because if a rule involves an attribute that is missing in the test instance, it is difficult to ignore the rule and proceed with the subsequent on in the set.

# RULE-BASED CLASSIFIERS: CHARACTERISTICS

## Imbalanced class distributions

- They can be handled through the rule ordering.

## Efficiency

- Fast model building
- Very fast classification

## Robustness

- Robust to outliers



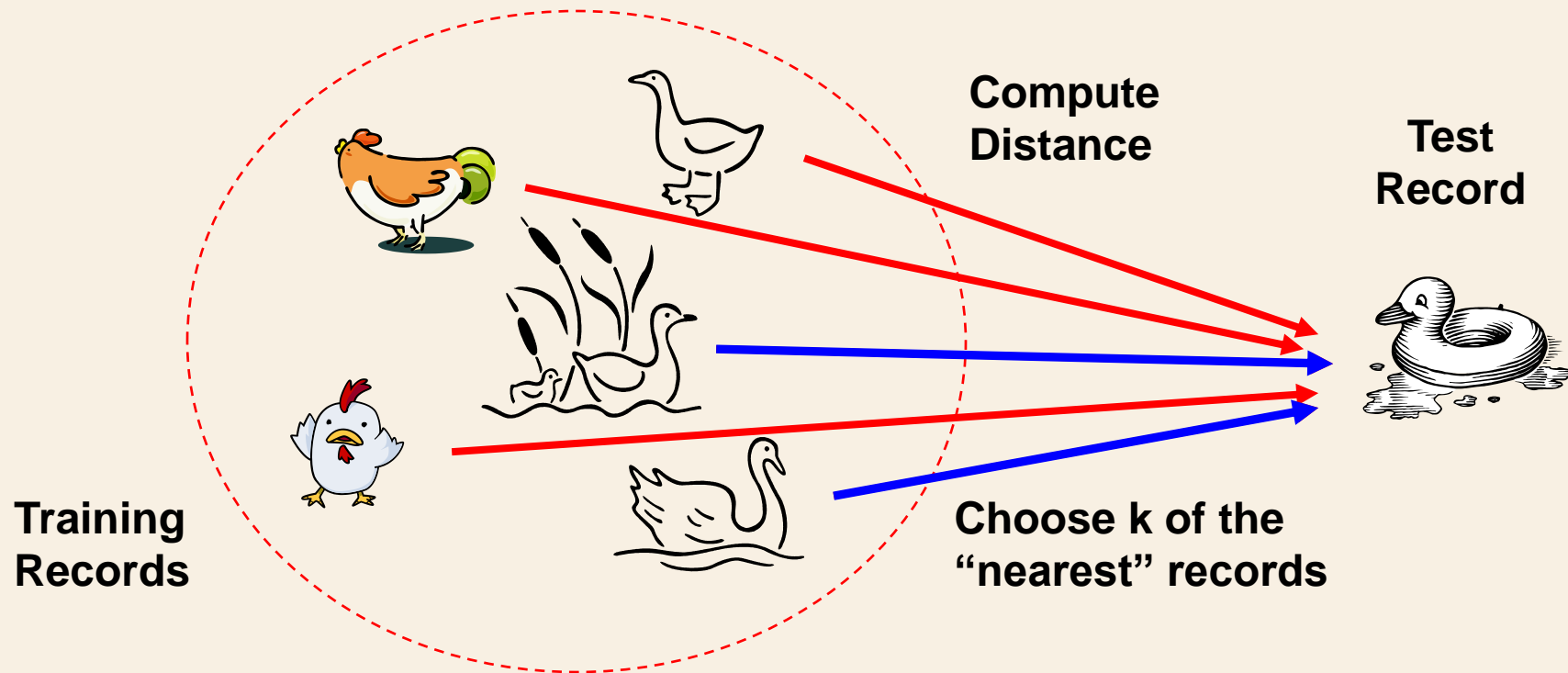
# NEAREST NEIGHBOR CLASSIFIER

# K-NEAREST NEIGHBOR

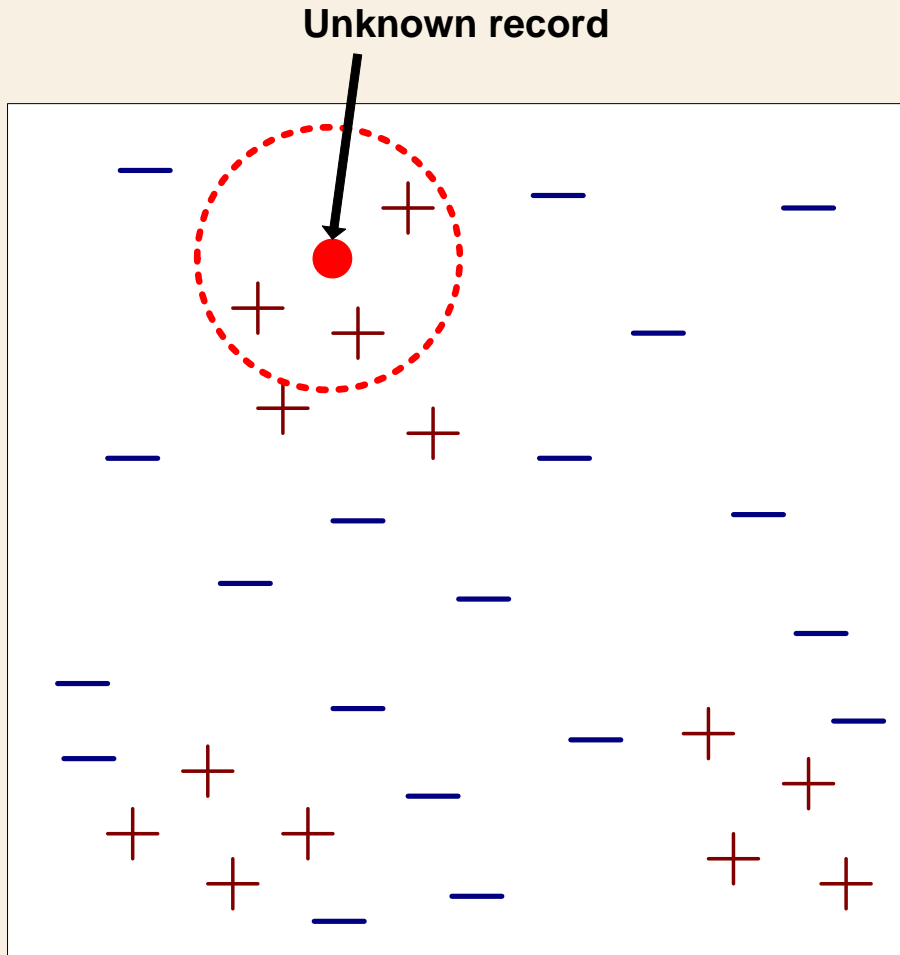
- A Nearest Neighbor classifier finds all the training examples that are relatively similar to the attributes of the test instances.
- Each example is represented as a data point in a  $d$ -dimensional space, where  $d$  is the number of attributes.
- Given a test instance, its proximity to the training instances is computed according to one of the available proximity measures.
- The  $k$ -nearest neighbors of a given instance  $z$  refer to the  $k$  training examples that are closer to  $z$ .

# NEAREST NEIGHBOR CLASSIFIERS

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



# NEAREST-NEIGHBOR CLASSIFIERS



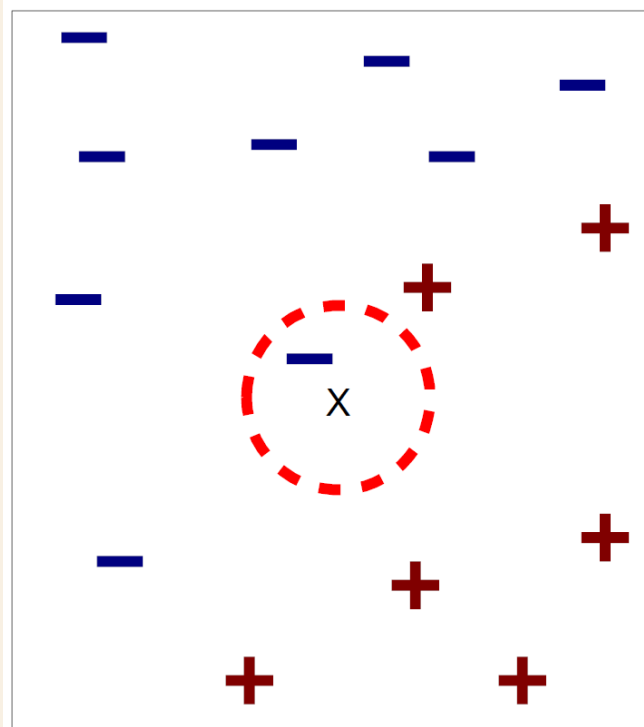
Requires the following:

- A set of labeled records
- Proximity metric to compute distance/similarity between a pair of records
  - e.g., Euclidean distance
- The value of  $k$ , the number of nearest neighbors to retrieve
- A method for using class labels of  $K$  nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

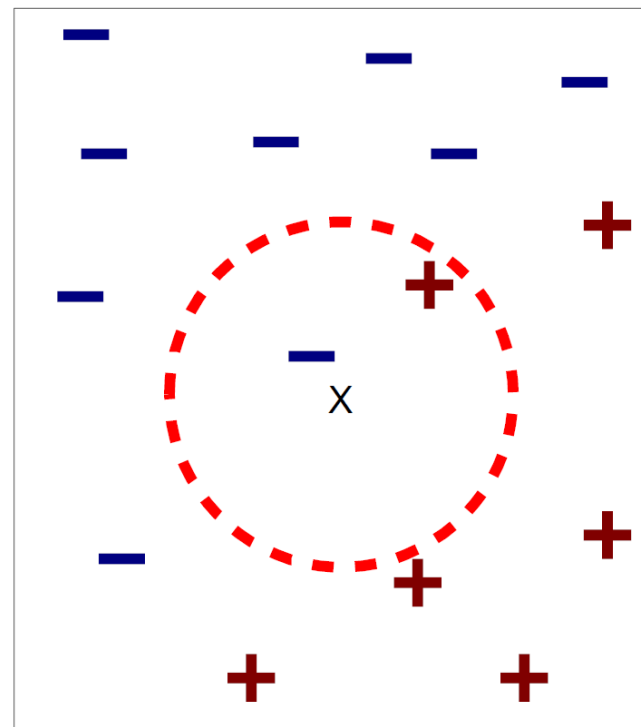
# K-NEAREST NEIGHBOR

- The instances is classified based on the class labels of its neighbors.
- In case where the neighbors have more than one label, the test instance is assigned to the majority class of its nearest neighbors.
- The decision of the value of  $k$  is crucial:
  - If  $k$  is too small, then the nearest neighbor classifier is subject to overfitting.
  - If  $k$  is too large, the nearest neighbor classifier can misclassify the test instance because the nearest neighbors includes training examples that are far away.

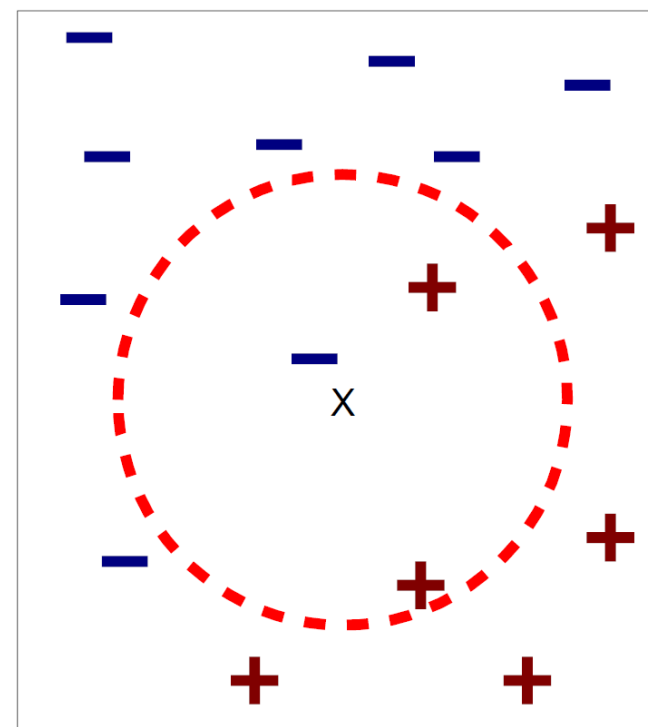
# K-NEAREST NEIGHBOR



(a) 1-nearest neighbor



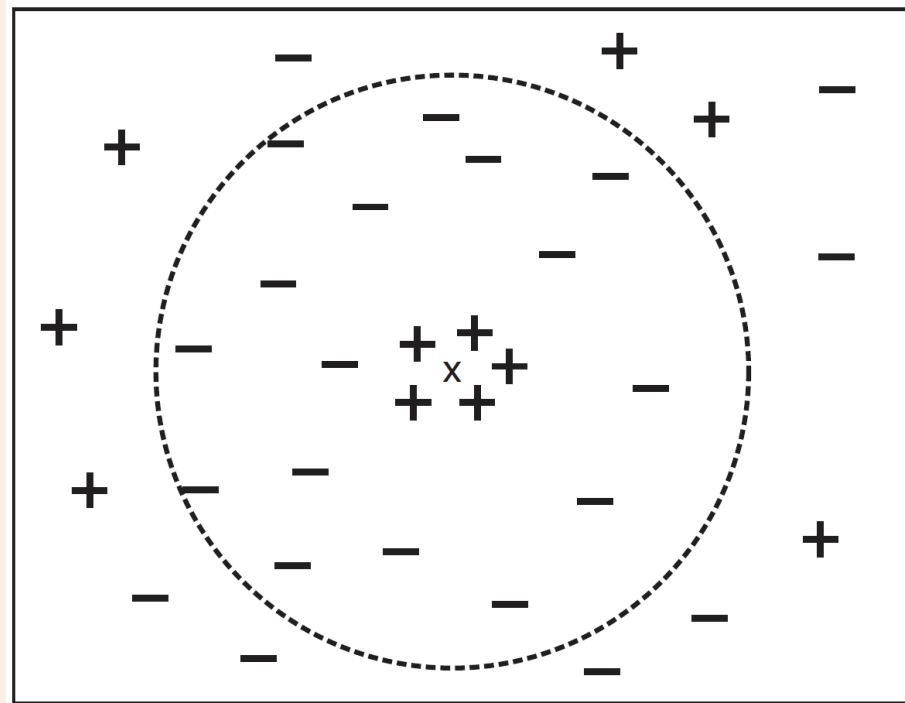
(b) 2-nearest neighbor



(c) 3-nearest neighbor

# K-NEAREST NEIGHBOR

- Too large k value



# K-NEAREST NEIGHBOR

---

**Algorithm 6.2** The  $k$ -nearest neighbor classifier.

---

- 1: Let  $k$  be the number of nearest neighbors and  $D$  be the set of training examples.
  - 2: **for** each test instance  $z = (\mathbf{x}', y')$  **do**
  - 3:   Compute  $d(\mathbf{x}', \mathbf{x})$ , the distance between  $z$  and every example,  $(\mathbf{x}, y) \in D$ .
  - 4:   Select  $D_z \subseteq D$ , the set of  $k$  closest training examples to  $z$ .
  - 5:    $y' = \operatorname{argmax}_v \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
  - 6: **end for**
- 

- The test instance is classified based on the majority class of its nearest neighbor  
→ every neighbor has the same impact on the classification.
- Distance-weighted Voting: use a weight based on the distance  $w_i = \frac{1}{d(x', x_i)^2}$

# K-NEAREST NEIGHBOR

**Algorithm 6.2** The  $k$ -nearest neighbor classifier.

- 1: Let  $k$  be the number of nearest neighbors and  $D$  be the set of training examples.
- 2: **for** each test instance  $z = (\mathbf{x}', y')$  **do**
- 3:   Compute  $d(\mathbf{x}', \mathbf{x})$ , the distance between  $z$  and every example,  $(\mathbf{x}, y) \in D$ .
- 4:   Select  $D_z \subseteq D$ , the set of  $k$  closest training examples to  $z$ .
- 5:    $y' = \operatorname{argmax}_v \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
- 6: **end for**

- The test instance is classified based on the majority class of its nearest neighbor  
→ every neighbor has the same impact on the classification.
- Distance-weighted Voting: use a weight based on the distance  $w_i = \frac{1}{d(x', x_i)^2}$

# NEAREST NEIGHBOR: CHARACTERISTICS

- Instance-based learning: it does not build a global model, but it uses the training examples to make predictions for a test instance.
  - Similarity based on the definition of a proximity measure.

## Expressiveness:

- It can build decision boundaries of arbitrary shape.

## Efficiency:

- It does not build a model, but it requires to compute every time the proximity value between the test and the training examples (slow classification)

# NEAREST NEIGHBOR: CHARACTERISTICS

## Interacting attribute

- It can handle interacting attributes without any problem.

## Irrelevant attributes

- They can distort commonly used proximity measures.

## Missing values in the test set

- Proximity computation normally require the presence of all attributes.

## Robustness

- With smaller values of  $k$ , it is susceptible to noise.
- It can produce wrong predictions unless the appropriate proximity measure and data pre-processing steps are taken.

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark purple with white concentric circles. The bottom-right square is bright pink with white concentric circles. A large, light gray circle is partially visible behind the grid.

# BAYESIAN CLASSIFIER

# NAÏVE BAYES CLASSIFIER

- Many classification problems involve uncertainty.
  - Attributes and class labels may be unreliable.
  - The set of attributes chosen for the classification may not be fully representative of the target class.
- There is the need to not only make a prediction of class labels but also provide a measure of confidence associated with every prediction.
- Probabilistic classification model.

# PROBABILITY

- The **probability** of an event  $e$ , e.g.  $P(X=x_i)$  measures how likely it is for the event  $e$  to occur (relative frequency).
- Variables that have probabilities associated with each possible outcome (values) are known as **random variables**.
- **Joint Probability**: two random variables  $X$  and  $Y$  which can take  $k$  distinct values. Let  $n_{ij}$  the number of times we observe  $X=x_i$  and  $Y=y_j$  out of a total number of  $N$  occurrences:

$$P(X=x_i, Y=y_j) = n_{ij}/N$$

# MARGINAL PROBABILITY

- **Marginal Probability:** By summing out the joint probabilities with respect to a random variable  $Y$ , we obtain the probability of observing the other variable  $X$  independently from  $Y$  (marginalization)

$$\sum_{j=1}^k P(X = x_i, Y = y_j) = \frac{\sum_{j=1}^k n_{ij}}{N} = \frac{n_i}{N} = P(X = x_i)$$

# CONDITIONAL PROBABILITY

- Let  $P(Y|X)$  denote the conditional probability of observing the random variable  $Y$  whenever the random variable  $X$  takes a particular value (= probability of observing  $Y$  conditioned on the outcome of  $X$ ).

$$P(Y|X) = \frac{P(X, Y)}{P(X)} \quad \leftarrow \text{Joint probability}$$


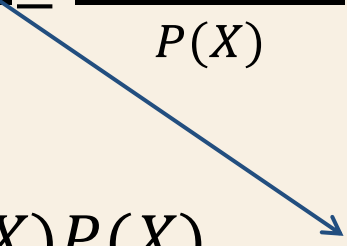
$$\begin{aligned} P(X, Y) &= P(Y|X) \times P(X) \\ &= P(X|Y) \times P(Y) \end{aligned} \quad \leftarrow \text{Joint probability is symmetric}$$

$P(X)$  = denote the probability of any generic outcome of  $X$   
 $P(x_i)$  = denote the probability of a specific outcome  $x_i$

# BAYES THEOREM

- $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$   Provides a relationship between  $P(Y|X)$  and  $P(X|Y)$

Dimostrazione

- $P(Y|X) = \frac{P(X,Y)}{P(X)} = \frac{P(X|Y)P(Y)}{P(X)}$   
  
•  $P(X,Y) = P(Y|X)P(X)$   •  $P(X,Y) = P(X|Y)P(Y)$

# BAYES THEOREM: EXAMPLE

- You know that Alex attends 40% of the parties he is invited to  $P(A=1) = 0.40$ . If Alex goes to a party, there is an 80% chance that Martha coming along  $P(M=1|A=1) = 0.80$ . Conversely, if Alex is not going to the party, the chance of Martha coming to the party is reduced to 30%:  $P(M=1|A=0) = 0.30$ .
- If Martha has responded she will come to the party, what is the probability that Alex will go also to the party?  $P(A=1|M=1)$ ?
- $P(A=1|M=1) = P(M=1|A=1) \times P(A=1) / P(M=1)$ 
  - $P(M=1) = P(M=1|A=0)P(A=0) + P(M=1|A=1)P(A=1)$
- $P(A=1|M=1) = 0.80 \times 0.40 / (0.30 \times 0.60 + 0.80 \times 0.40) = 0.64$

# BAYES THEOREM FOR CLASSIFICATION

- We are interested in computing the probability of observing a class label  $y$  for a data instance given its set of attribute values  $\mathbf{x}$ :  $P(y|\mathbf{x})$  *posterior probability of the target class*
- $P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}$
- $P(\mathbf{x}|y)$  is the *class-conditional probability*, it measures the likelihood of observing  $\mathbf{x}$  from the distribution instances belonging to  $y$ .
  - Fraction of the training instances of a given class for every possible combination of attribute values
- $P(y)$  is the *prior probability*, it measures the distribution of the class labels, independently from the observed attribute values.
  - Fraction of the training instances that belong to the class  $y$ .
- $P(\mathbf{x})$  is the probability of evidence  $\rightarrow P(\mathbf{x}) = \sum_i P(\mathbf{x}|y_i)P(y_i)$

# NAÏVE BAYES ASSUMPTION

- A large number of attribute value combinations can also result in poor estimates of the class-conditional probabilities.
- The naïve bayes assumption helps in obtaining reliable estimates of class-conditional probabilities even in presence of a large number of attributes.

$$P(\mathbf{x}|y) = \prod_{i=1}^d P(x_i|y)$$

- $X = \{x_1, \dots, x_d\}$  and  $x_i$  are independent of each other
- Compute each  $P(x_i|y)$  independently  $\rightarrow$  the number of parameters need to learn class-conditional probabilities is reduced from  $d^k$  to  $dk$ .

# NAÏVE BAYES CLASSIFIER: EXAMPLE

Outlook	Temperature	Humidity	Windy	Class
Sunny	Hot	High	False	N
Sunny	Hot	High	True	N
Overcast	Hot	High	False	P
Rain	Mild	High	False	P
Rain	Cool	Normal	False	P
Rain	Cool	Normal	True	N
Overcast	Cool	Normal	True	P
Sunny	Mild	High	False	N
Sunny	Cool	Normal	False	P
Rain	Mild	Normal	False	P
Sunny	Mild	Normal	True	P
Overcast	Mild	High	True	P
Overcast	Hot	Normal	False	P
Rain	Mild	High	True	N

$$P(y=P) = 9/14$$

$$P(y=N) = 5/14$$

$x_i = \text{Outlook}$

$$P(\text{sunny} | P) = 2/9$$

$$P(\text{sunny} | N) = 3/5$$

$$P(\text{overcast} | P) = 4/9$$

$$P(\text{overcast} | N) = 0/5$$

$$P(\text{rain} | P) = 3/9$$

$$P(\text{rain} | N) = 2/5$$

---

$x_i = \text{temperature}$

$$P(\text{hot} | P) = 2/9$$

$$P(\text{hot} | N) = 2/5$$

$$P(\text{mild} | P) = 4/9$$

$$P(\text{mild} | N) = 2/5$$

$$P(\text{cool} | P) = 3/9$$

$$P(\text{cool} | N) = 1/5$$

---

$x_i = \text{humidity}$

$$P(\text{high} | P) = 3/9$$

$$P(\text{high} | N) = 4/5$$

$$P(\text{normal} | P) = 6/9$$

$$P(\text{normal} | N) = 2/5$$

---

$x_i = \text{Windy}$

$$P(\text{true} | P) = 3/9$$

$$P(\text{true} | N) = 3/5$$

$$P(\text{false} | P) = 6/9$$

$$P(\text{false} | N) = 2/5$$

# NAÏVE BAYES CLASSIFIER: EXAMPLE

- Data to be labelled:  $X = \langle \text{rain, hot, high, false} \rangle$

- For class  $y=P$

- $P(X|y=P) \times P(y=P) =$   
 $P(\text{rain}|P) \times P(\text{hot}|P) \times P(\text{high}|P) \times P(\text{false}|P) \times P(P) =$   
 $3/9 \times 2/9 \times 3/9 \times 6/9 \times 9/14 = 0.010582$

- For class  $y=N$

- $P(X|y=N) \times P(y=N) =$   
 $P(\text{rain}|N) \times P(\text{hot}|N) \times P(\text{high}|N) \times P(\text{false}|N) \times P(N) =$   
 $2/5 \times 2/5 \times 4/5 \times 2/5 \times 5/14 = 0.018286$

# NAÏVE BAYES CLASSIFIER: CHARACTERISTICS

- It can easily work with incomplete information about data instances (missing values).
- It requires that attributes are conditionally independent of each other given the class labels.
  - Correlated attributes can degrade the performances
- It is robust to isolated noise points (they not significantly impact the conditional probability estimates).
- It is robust to irrelevant attributes.
- Fast model building and fast classification.

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark purple with white concentric circles. The bottom-right square is bright pink with white concentric circles. A large, light gray circle is partially visible behind the grid.

# OTHER CLASSIFIERS

# BAYESIAN NETWORKS

- Approach that relax the naïve Bayesian assumption.
- It allows to model probabilistic relationships between attributes and class labels.
- Bayesian networks are probabilistic graphical models where nodes represents random variables and edges between the nodes express the probabilistic relationships.

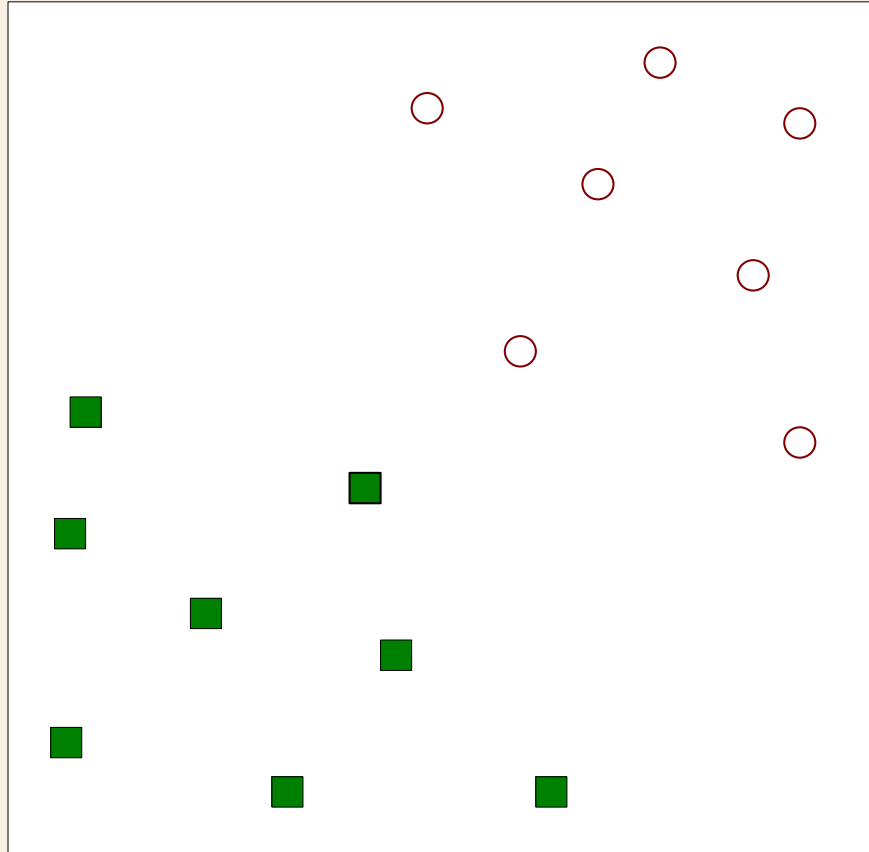
# LOGISTIC REGRESSION

- Naïve bayes classifiers and Bayesian networks are probabilistic generative models, because they describe the behavior of instances in the attribute space that are generated by the class  $y$ .
- Classification models that directly assign class labels without computing class-conditional probabilities are called discriminative models.
- Logistic regression is similar to regression models, but the computed values are the probabilities used to determine the class label of a data instance.

# SUPPORT VECTOR MACHINE (SVM)

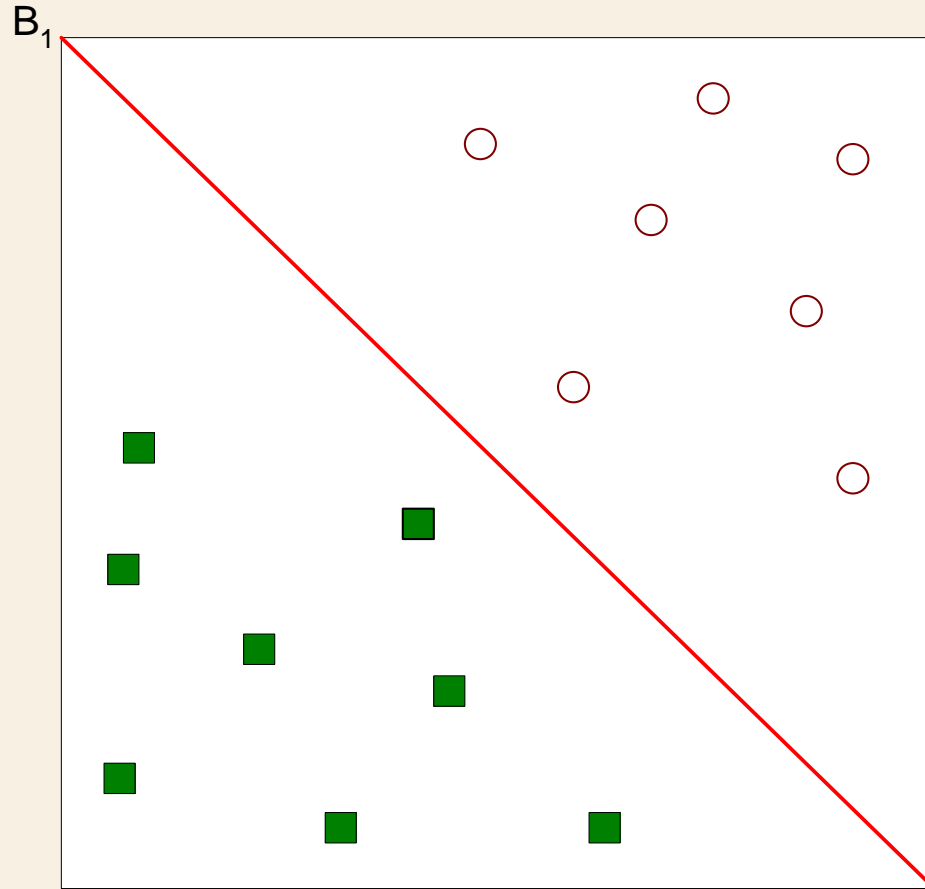
- SVM is a discriminative classification model that learns linear or non-linear decision boundaries in the attribute space to separate the classes.
- Separating hyperplane:  $\mathbf{w}^T \mathbf{x} + b = 0$ 
  - $\mathbf{x}$  represents the attributes
  - $(\mathbf{w}, b)$  represent the parameters of the hyperplane
  - A data instance  $x_i$  can belong to either side of the hyperplane depending on the sign of  $(\mathbf{w}^T \mathbf{x} + b)$

# SUPPORT VECTOR MACHINE (SVM)



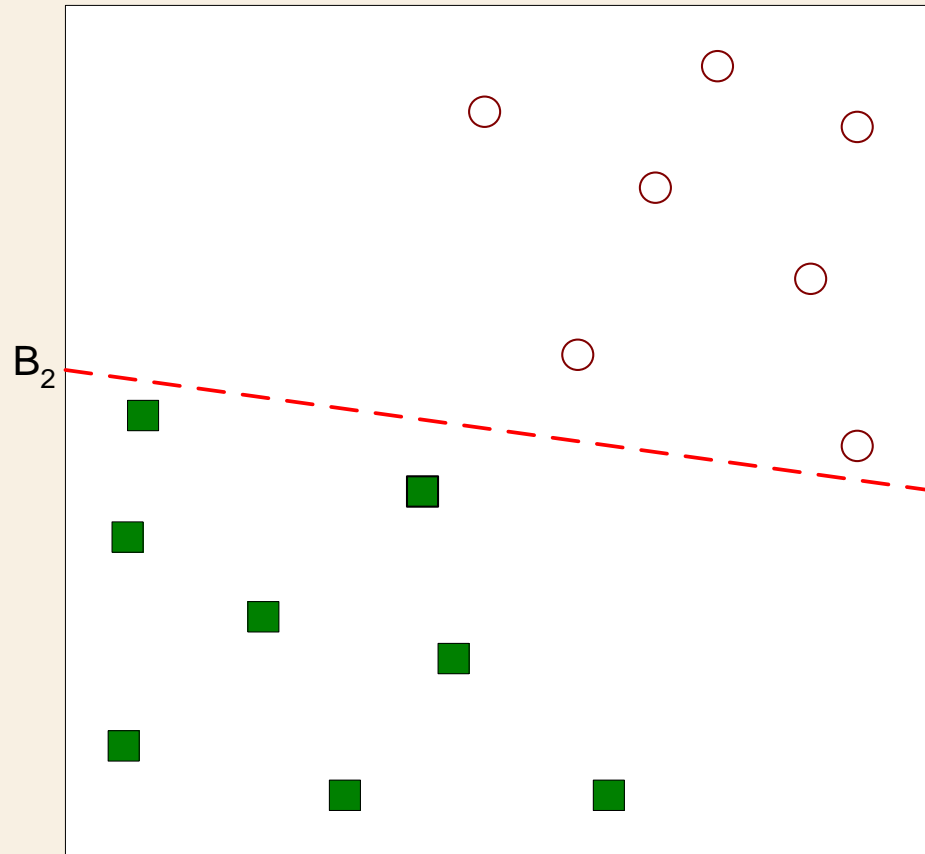
- Find a linear hyperplane (decision boundary) that will separate the data

# SUPPORT VECTOR MACHINE (SVM)



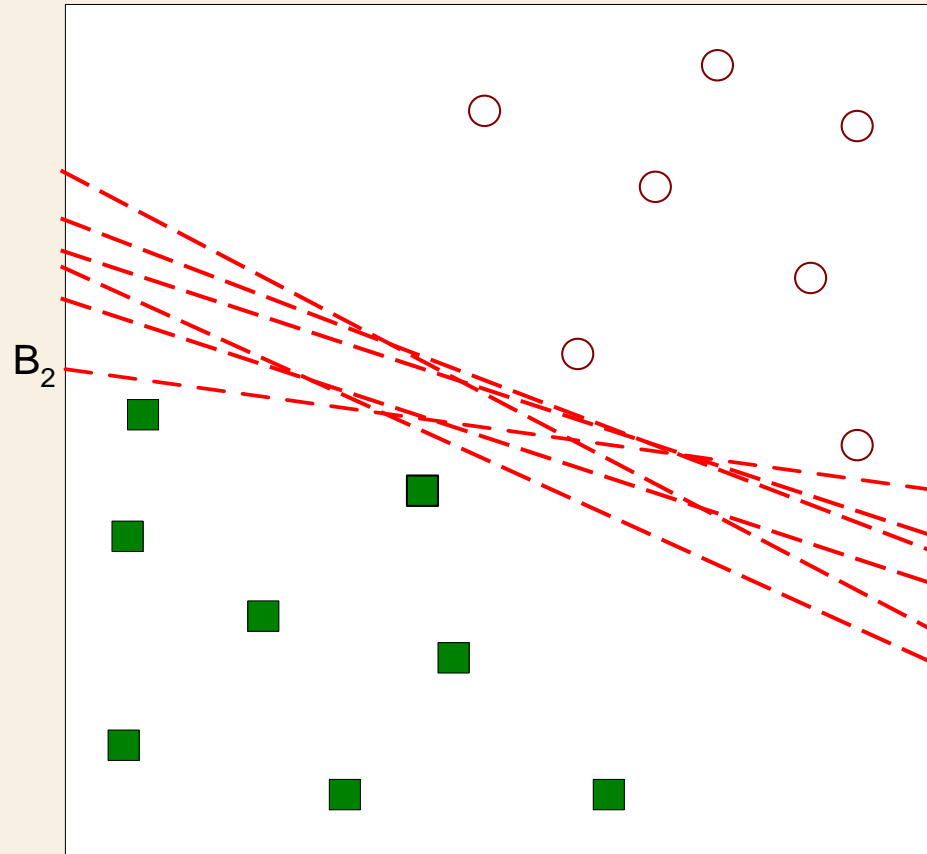
- One Possible Solution

# SUPPORT VECTOR MACHINE (SVM)



- Another possible solution

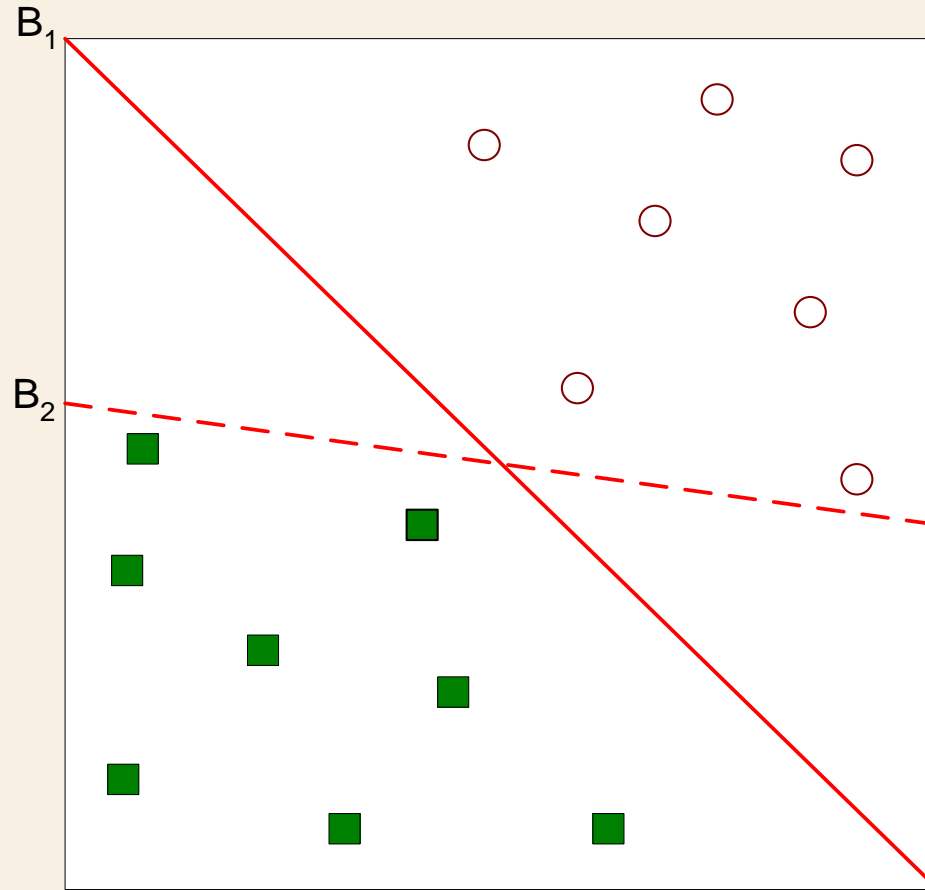
# SUPPORT VECTOR MACHINE (SVM)



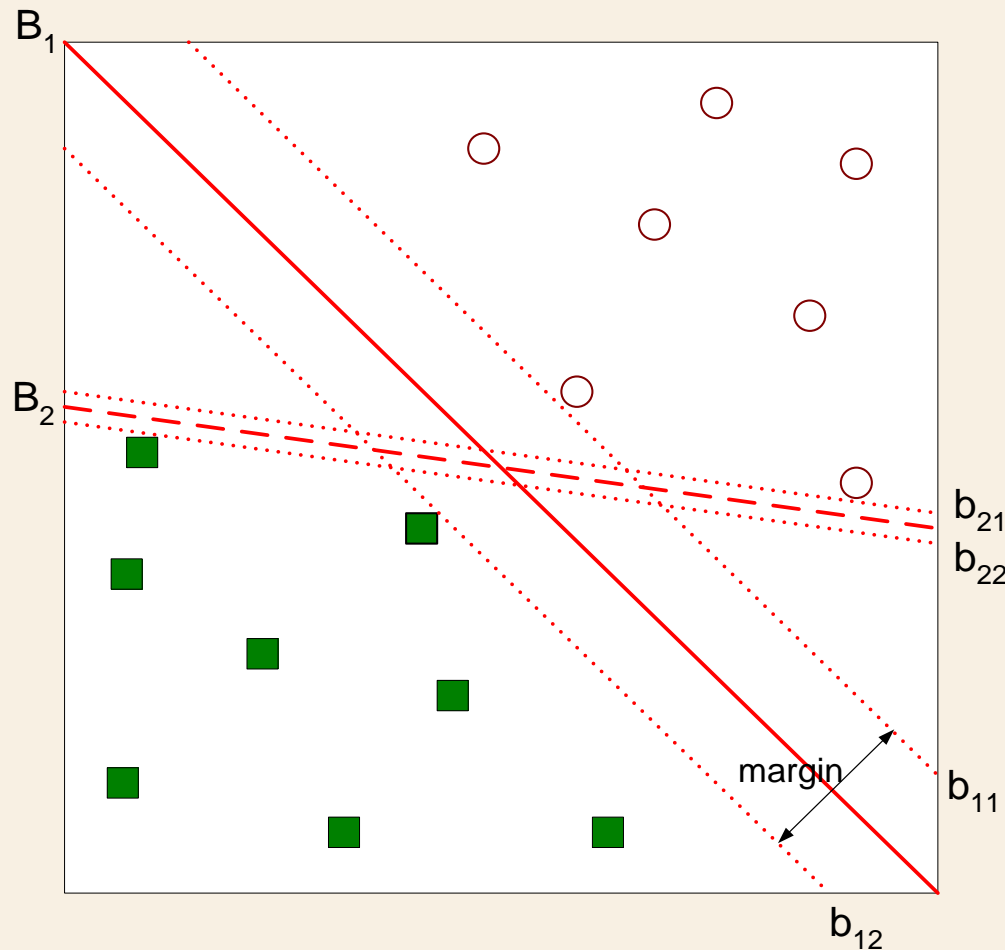
- Other possible solutions

# SUPPORT VECTOR MACHINE (SVM)

- Which one is better? B1 or B2?
- How do you define better?



# SUPPORT VECTOR MACHINE (SVM)



- Find hyperplane **maximizes** the margin  
→ B1 is better than B2

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark purple with white concentric circles. The bottom-right square is bright pink with white concentric circles. A large, light gray circle is partially visible on the left side, overlapping the top and bottom squares.

# ARTIFICIAL NEURAL NETWORKS

# ARTIFICIAL NEURAL NETWORKS

- Different tasks, different architectures:
- Image understanding → CNN
- Time series analysis → RNN
- Denoising → Auto-encoders
- Text and speech recognition → LLM
- etc... not part of the course (there are some available AI courses)

A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark purple with white concentric circles. The bottom-right square is bright pink with white concentric circles. A large, light gray circle is partially visible behind the grid.

# ENSEMBLE METHODS

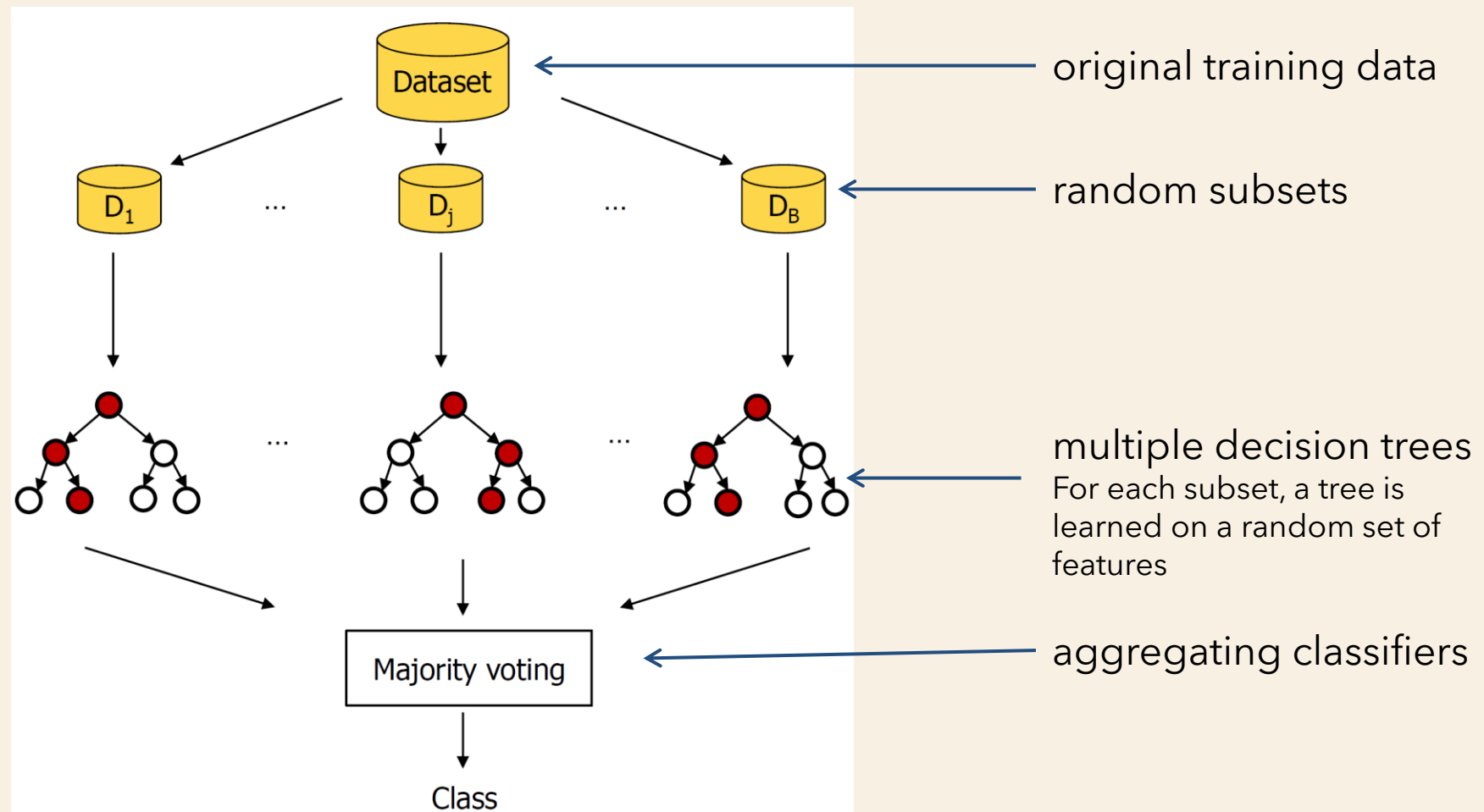
# ENSEMBLE METHODS

- Sometimes a single classification model is not enough.
- Classification accuracy can be improved by aggregating the predictions of multiple classifiers.
- The idea is to construct a set of base classifiers from the same dataset and then perform classification by taking a vote on the predictions made by each base classifier.

# ENSEMBLE METHODS: RANDOM FOREST

- A random forest is an ensemble learning technique represented by a set of decision trees.
- A number of decision trees are built at training time and the class is assigned by majority voting.

# ENSEMBLE METHODS: RANDOM FOREST





# CLASS IMBALANCE PROBLEM

# CLASS IMBALANCE PROBLEM

- In many data sets there are a disproportionate number of instances that belong to a certain class with respect to the other (skew or class imbalance).
- It can be difficult to find sufficiently many labeled samples of a rare class.
- A classifier trained over an imbalanced data set shows a bias toward improving its performance over the majority class.
- Accuracy is not well suited for evaluating models in the presence of class imbalance in the test data.

$$\text{acc} = (f_{PP} + f_{NN}) / (f_{PP} + f_{PN} + f_{NP} + f_{NN})$$

# DEALING WITH IMBALANCED DATA: TRAINING SET TRANSFORMATION

- The first step in learning with imbalanced data is transform the training set into a balanced training set.
- Undersampling: the frequency of the majority class is reduced to match the frequency of the minority class.
  - Some useful examples may not be chosen for training, obtaining an inferior classification model.
- Oversampling: artificial examples of the minority class are created to make them equal in proportion to the number of the majority class.
  - Artificially duplicate the instances of the minority class (doubling its weight).
  - Assigning higher weights to instances of the minority class.
  - Generate synthetic instances of the minority class in the neighborhood of the existing instances.

} Poor generalization capabilities

# DEALING WITH IMBALANCED DATA: EVALUATING PERFORMANCES

- Accuracy and Confusion matrix are not well-suited metrics in presence of imbalanced classes.

- Accuracy

$$\text{acc} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

# DEALING WITH IMBALANCED DATA: EVALUATING PERFORMANCES

- Consider a binary problem:
  - Cardinality of class 0 = 9900
  - Cardinality of class 1 = 100
- Model
  - $() \rightarrow$  class 0
  - Model predicts everything to be class 0
- Accuracy =  $9900/10000 = 99\%$
- Accuracy is misleading because the model does not detect any class 1 object.

# DEALING WITH IMBALANCED DATA: EVALUATING PERFORMANCES

## Recall

- $\text{recall (r)} = \text{TP} / \text{TP} + \text{FN}$
- Number of object correctly assigned to the positive class, divided by the number of objects really belonging to that class

## Precision

- $\text{precision (p)} = \text{TP} / \text{TP} + \text{FP}$
- Fraction of correct predictions of the positive class over the total number of positive predictions.

# DEALING WITH IMBALANCED DATA: EVALUATING PERFORMANCES

- F1
- $F_1 = \frac{2rp}{r+p}$
- It represents an harmonic mean between recall and precision.

# DEALING WITH IMBALANCED DATA: ROC CURVE

- ROC = Receiver Operating Characteristic
- A graphical approach for displaying trade-off between detection rate and false alarm rate
- ROC curve plots
  - y-axis = True Positive Rate  $TPR = TP / (TP + FN)$
  - x-axis = False Positive Rate  $FPR = FP / (FP + TN)$

# DEALING WITH IMBALANCED DATA: ROC CURVE

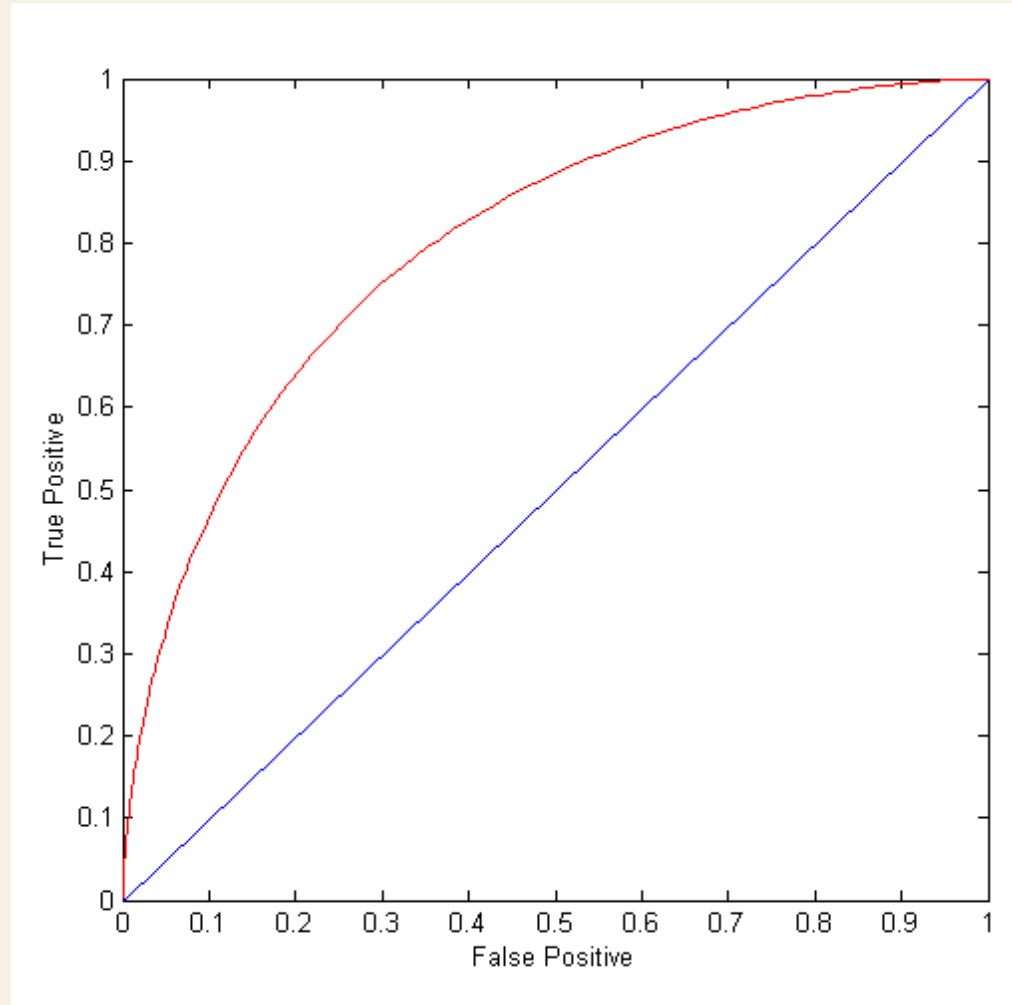
(TPR,FPR):

- (0,0): declare everything to be negative class
- (1,1): declare everything to be positive class
- (1,0): ideal

Diagonal line:

- It represents a random guessing
- Below diagonal line: prediction is opposite of the true class

A good classification model should be located as close as possible to the upper left corner in the diagram (above the diagonal!)



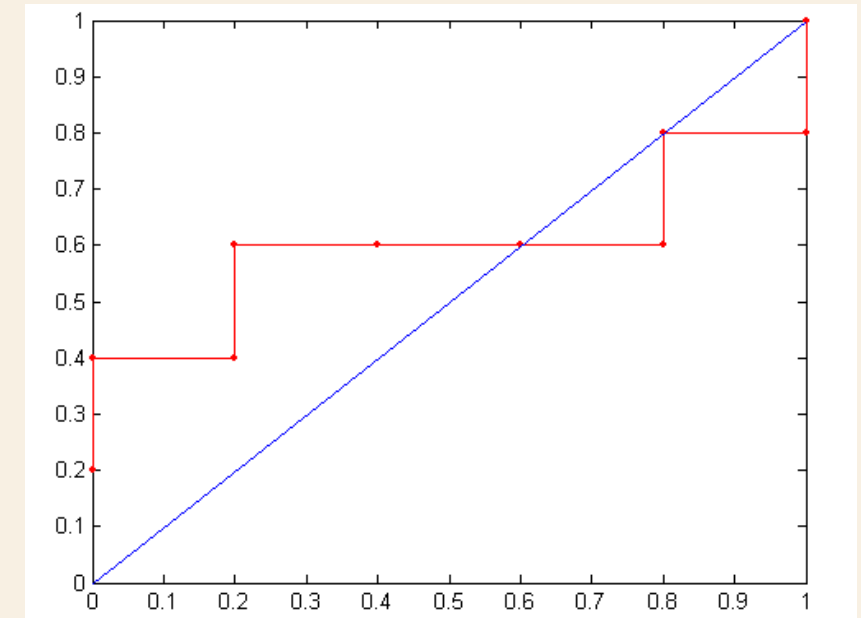
# HOW TO CONSTRUCT AN ROC CURVE

Instance	Score	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

- Use a classifier that produces a continuous-valued score for each instance
  - The more likely it is for the instance to be in the + class, the higher the score
- Sort the instances in decreasing order according to the score
- Select the lowest ranked test instance. Assign the selected instance and those above to the positive class, while those ranked below it as negative.
- Count the number of TP, FP, TN, FN at each threshold
  - $TPR = TP / (TP + FN)$
  - $FPR = FP / (FP + TN)$

# HOW TO CONSTRUCT AN ROC CURVE

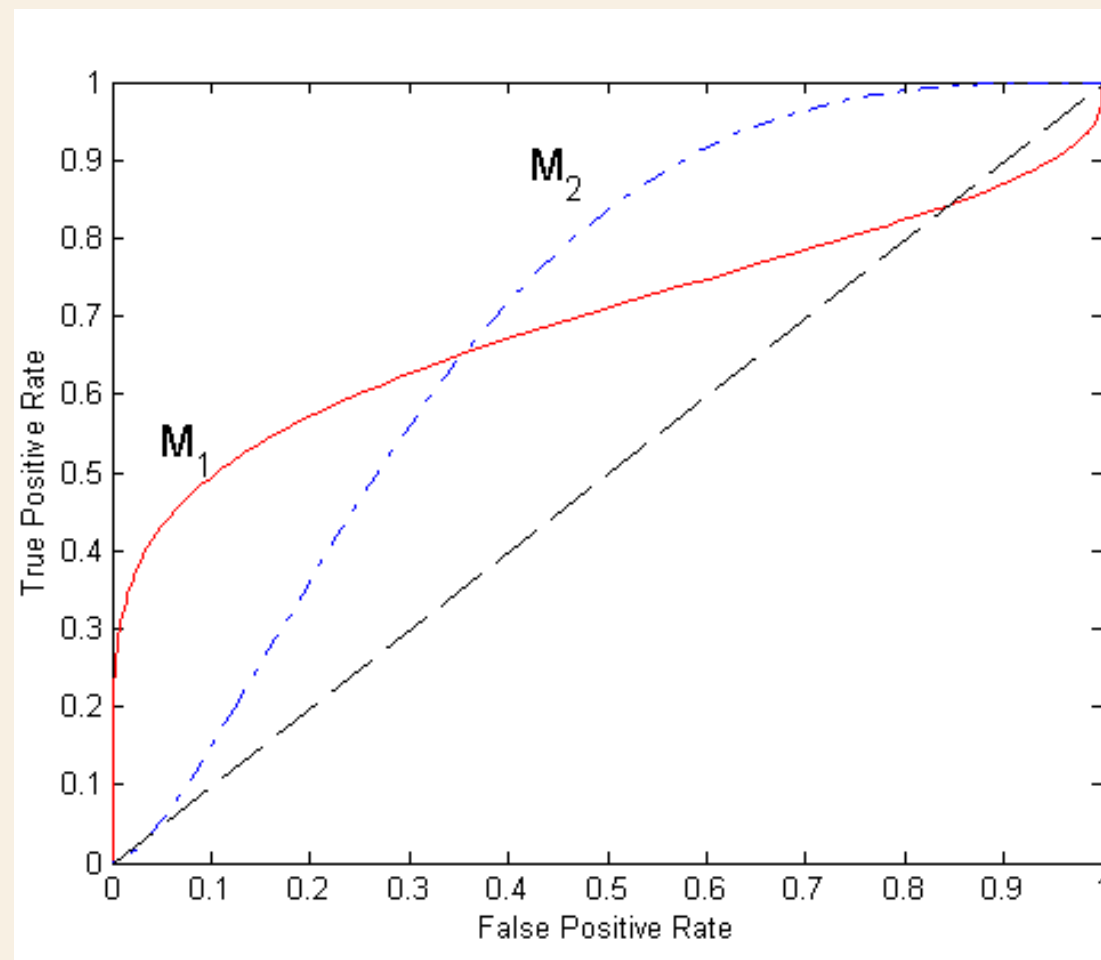
Class	+	-	+	-	-	-	+	-	+	+	
Threshold >=	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0



ROC Curve

# USING ROC FOR MODEL COMPARISON

- No model consistently outperforms the other
  - M1 is better for small FPR
  - M2 is better for large FPR
- Area Under the ROC curve (AUC)
  - Ideal: Area = 1
  - Random guess: Area = 0.5



A decorative graphic on the left side of the slide, consisting of a 2x2 grid of squares. The top-left square is dark purple with white concentric circles. The top-right square is bright pink. The bottom-left square is dark purple with white concentric circles. The bottom-right square is bright pink with white concentric circles. A large, light gray circle is partially visible behind the grid.

# MULTICLASS PROBLEM

# MULTICLASS PROBLEM

- (1-r approach) Decompose the multiclass problem into  $K$  binary problems
  - For each class  $y_i \in Y$ , a binary problem is created where all instances that belong to  $y_i$  are considered positive examples, while the remaining instances are considered negative examples.
- (1-1 approach) Constructs  $K(K-1)/2$  binary classifiers.
  - Each classifier is used to distinguish between a pair of classes  $(y_i, y_j)$ . Instances that do not belong to  $y_i$  or  $y_j$  are ignored.
- The final prediction is made by combining the predictions made by the binary classifiers.