

# Lecture outline

- Classification
- Decision-tree classification

# What is classification?

	<i>binary</i>	<i>categorical</i>	<i>continuous</i>	<i>class</i>
Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

# What is classification?

- **Classification** is the task of *learning a target function*  $f$  that maps attribute set  $x$  to one of the predefined class labels  $y$

	binary	categorical	continuous	class
Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

# What is classification?

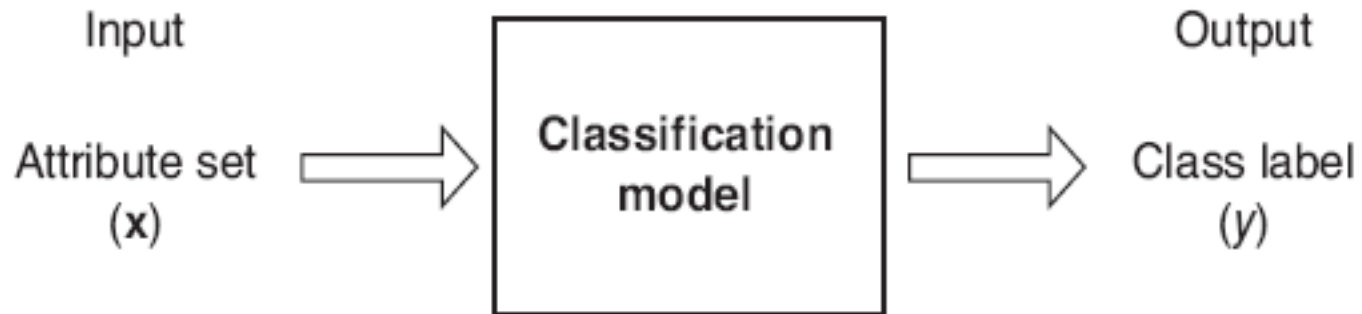


Figure 4.2. Classification as the task of mapping an input attribute set  $x$  into its class label  $y$ .

# Why classification?

- The target function **f** is known as a ***classification model***
- **Descriptive modeling:** ***Explanatory tool*** to distinguish between objects of different classes (e.g., description of who can pay back his loan)
- **Predictive modeling:** Predict a class of a previously ***unseen*** record

# Typical applications

- credit approval
- target marketing
- medical diagnosis
- treatment effectiveness analysis

# General approach to classification

- ***Training set*** consists of records with ***known class labels***
- Training set is used to ***build a classification model***
- The classification model is applied to the ***test set*** that consists of records with ***unknown labels***

# General approach to classification

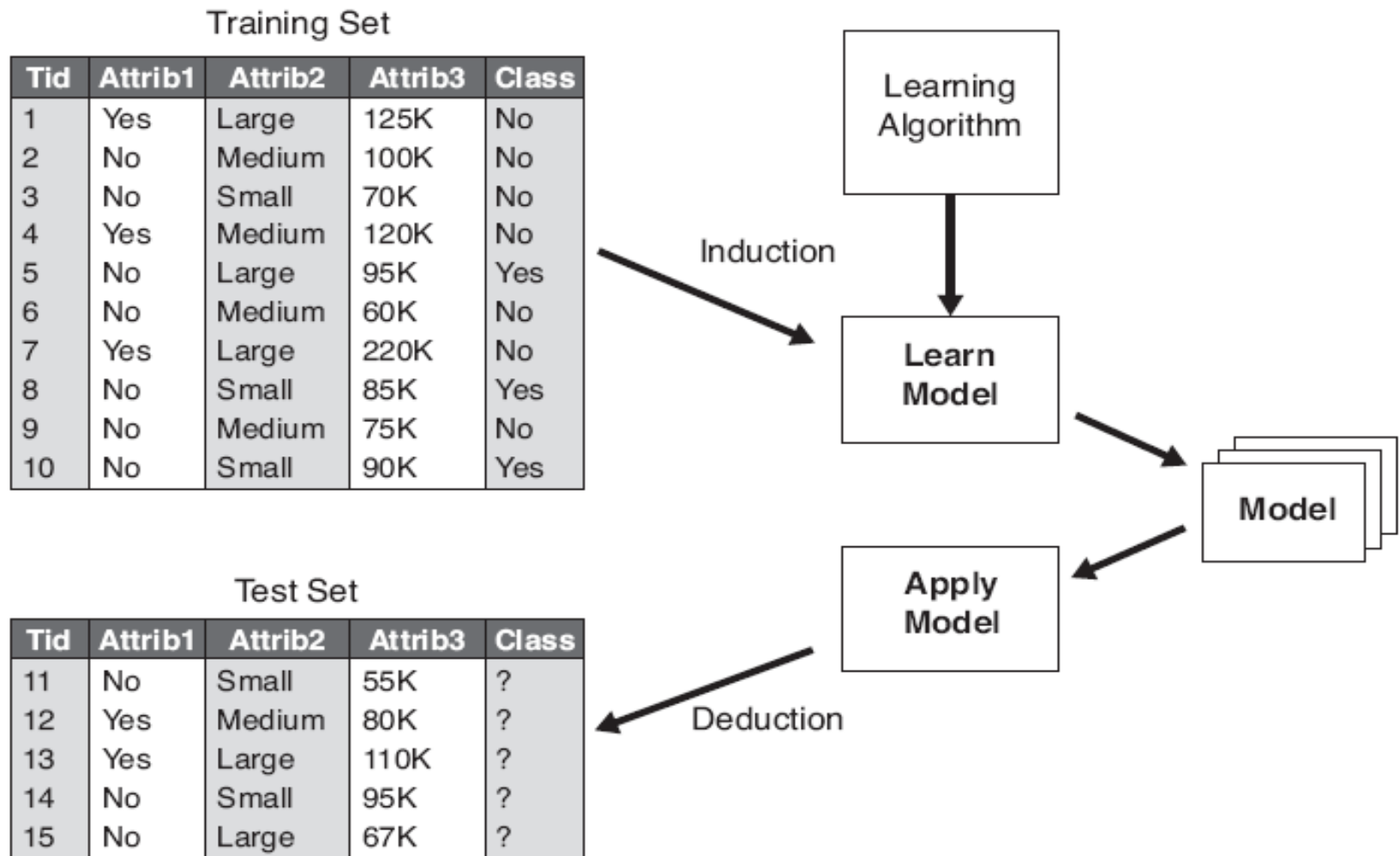


Figure 4.3. General approach for building a classification model.



# Evaluation of classification models

- Counts of test records that are correctly (or incorrectly) predicted by the classification model

- **Confusion matrix**

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	$f_{11}$	$f_{10}$
	Class = 0	$f_{01}$	$f_{00}$

$$\text{Accuracy} = \frac{\# \text{ correct prediction s}}{\text{total \# of prediction s}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$\text{Error rate} = \frac{\# \text{ wrong prediction s}}{\text{total \# of prediction s}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

# Supervised vs. Unsupervised Learning

- **Supervised learning (classification)**
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - New data is classified based on the training set
- **Unsupervised learning (clustering)**
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

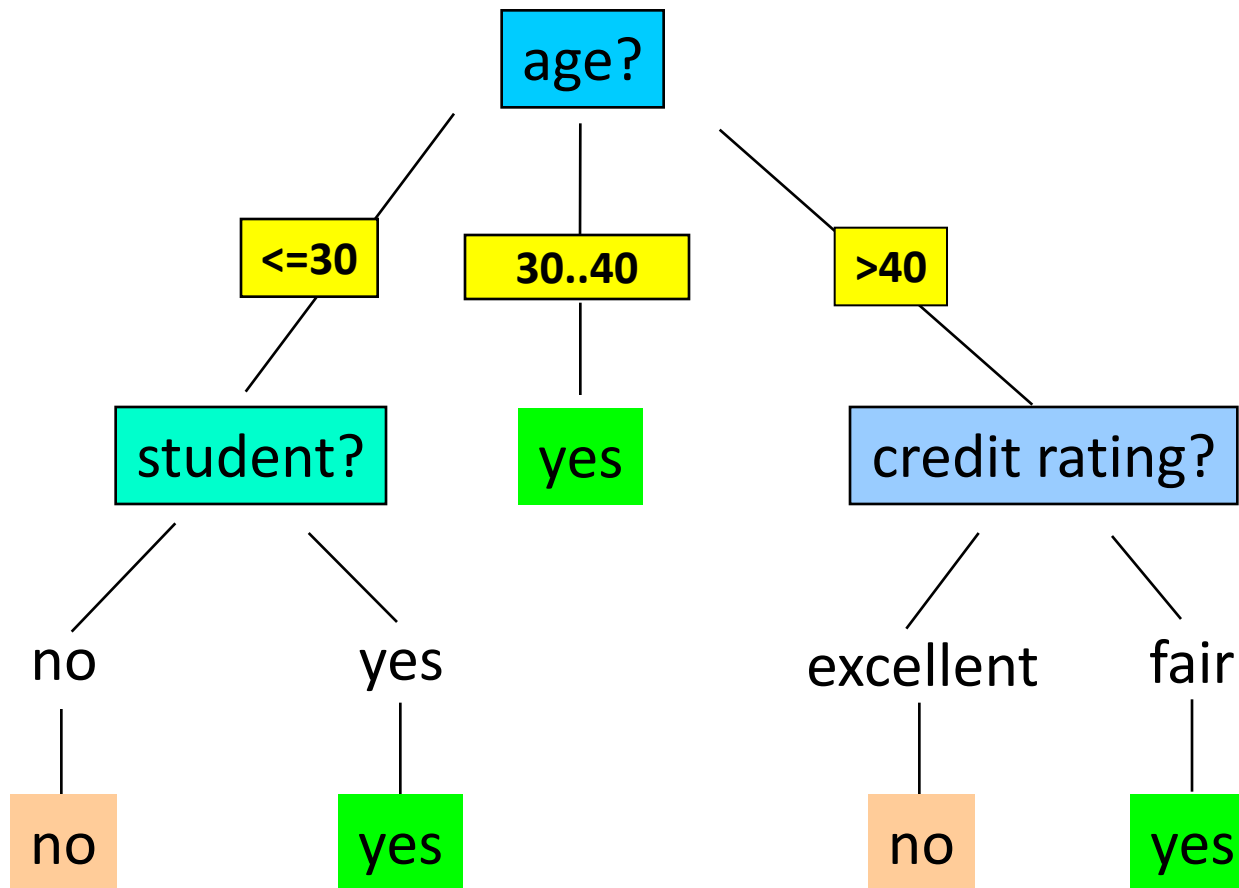
# Decision Trees

- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Decision tree generation consists of two phases
  - **Tree construction**
    - At start, all the training examples are at the root
    - Partition examples recursively based on selected attributes
  - **Tree pruning**
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree: Classifying an unknown sample
  - Test the attribute values of the sample against the decision tree

# Training Dataset

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

# Output: A Decision Tree for *“buys\_computer”*



# Constructing decision trees

- Exponentially many decision trees can be constructed from a given set of attributes
- Finding the most accurate tree is NP-hard
- **In practice: greedy algorithms**
  - Grow a decision tree by making a series of *locally optimum decisions on which attributes to use* for partitioning the data

# Constructing decision trees: the Hunt's algorithm

- $X_t$ : the set of training records for node  $t$
- $y = \{y_1, \dots, y_c\}$ : class labels
- **Step 1:** If all records in  $X_t$  belong to the same class  $y_t$ , then  $t$  is a leaf node labeled as  $y_t$
- **Step 2:** If  $X_t$  contains records that belong to more than one class,
  - select *attribute test condition* to partition the records into smaller subsets
  - Create a *child node* for each outcome of test condition
  - Apply algorithm *recursively* for each *child*

# Decision-tree construction (Example)

	binary	categorical	continuous	class
Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

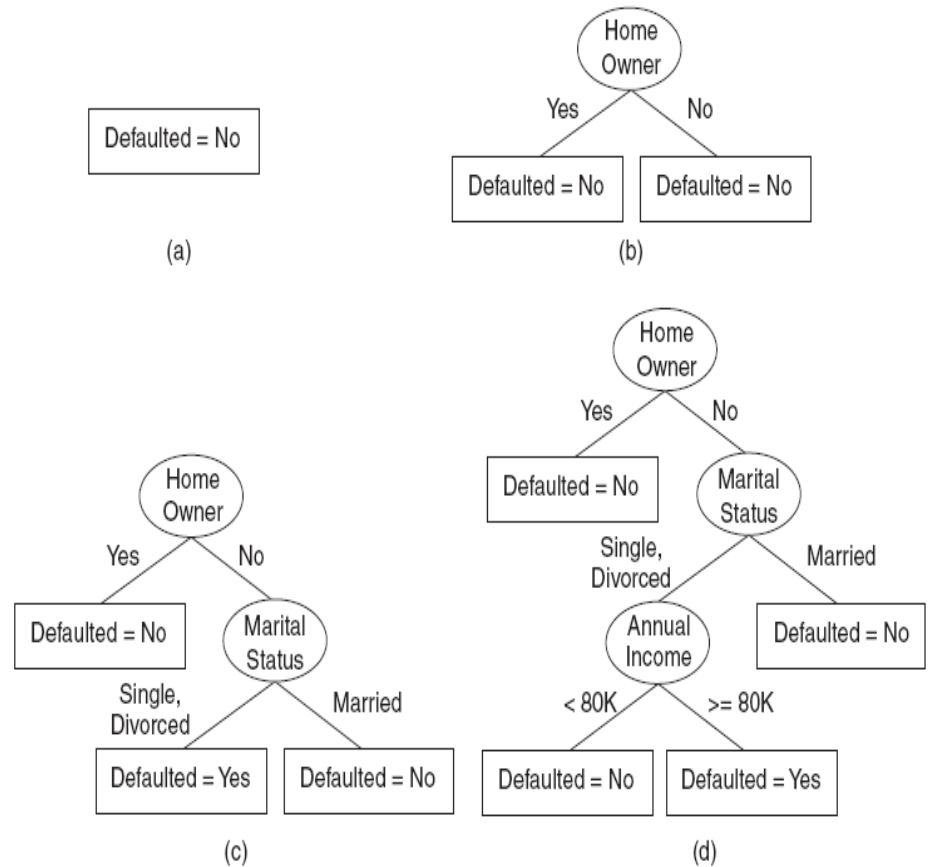


Figure 4.7. Hunt's algorithm for inducing decision trees.



# Design issues

- How should the training records be split?
- How should the splitting procedure stop?

# Splitting methods

- Binary attributes

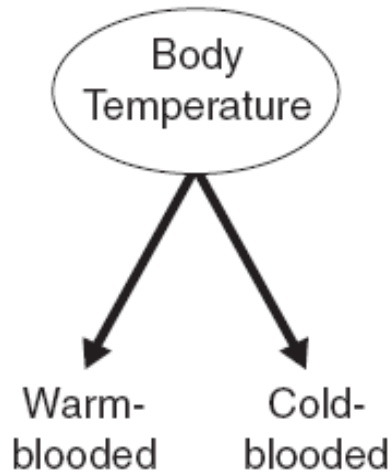


Figure 4.8. Test condition for binary attributes.

# Splitting methods

- Nominal attributes

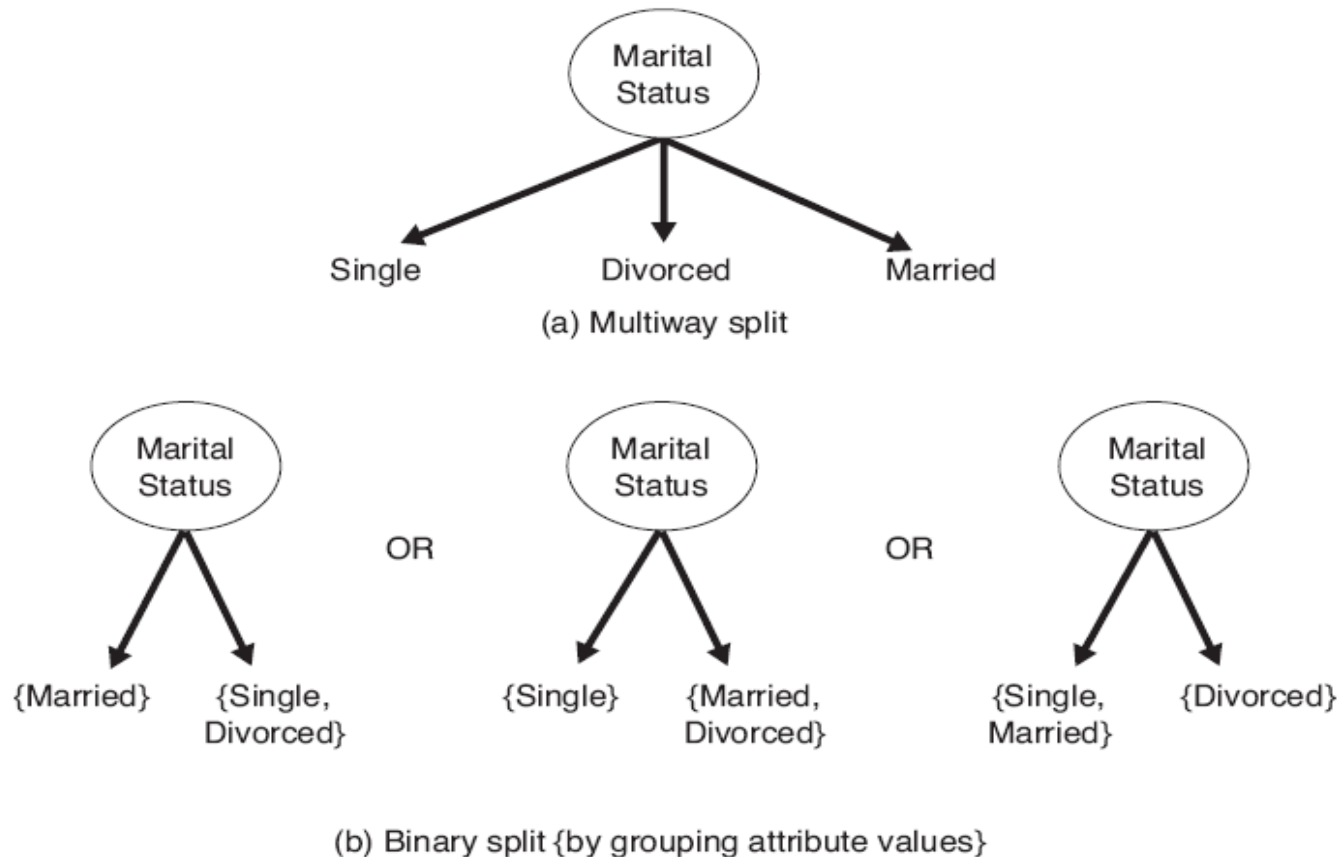


Figure 4.9. Test conditions for nominal attributes.

# Splitting methods

- Ordinal attributes

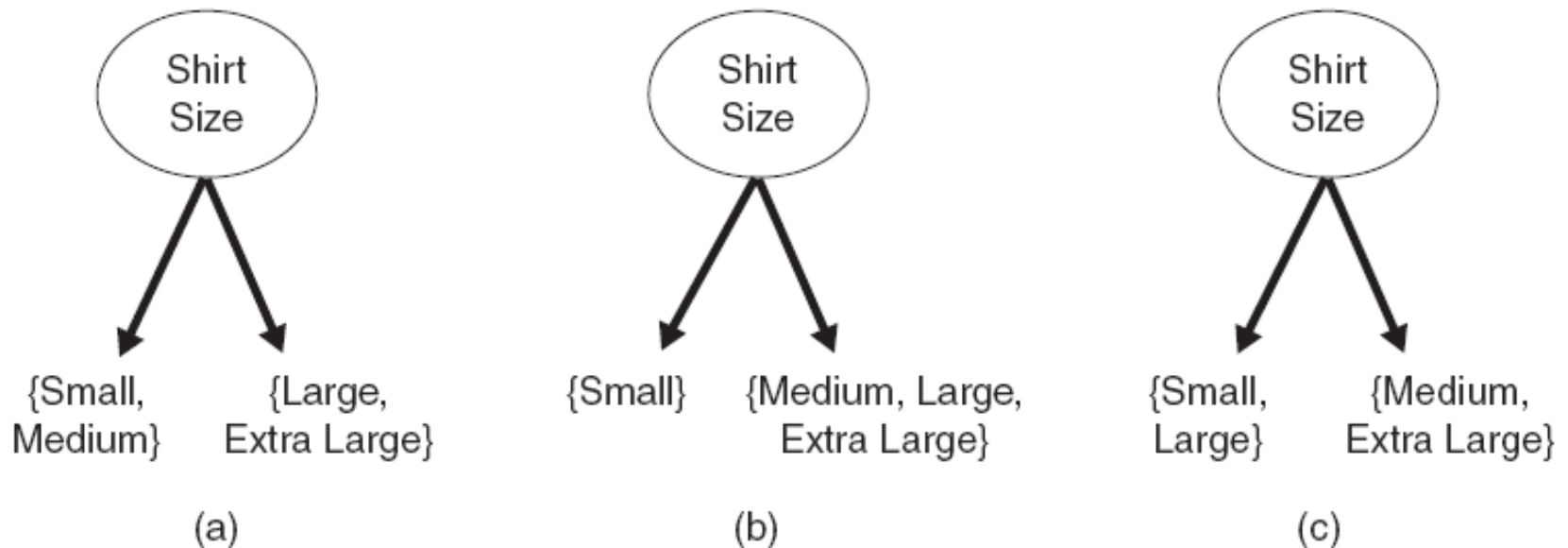


Figure 4.10. Different ways of grouping ordinal attribute values.

# Splitting methods

- Continuous attributes

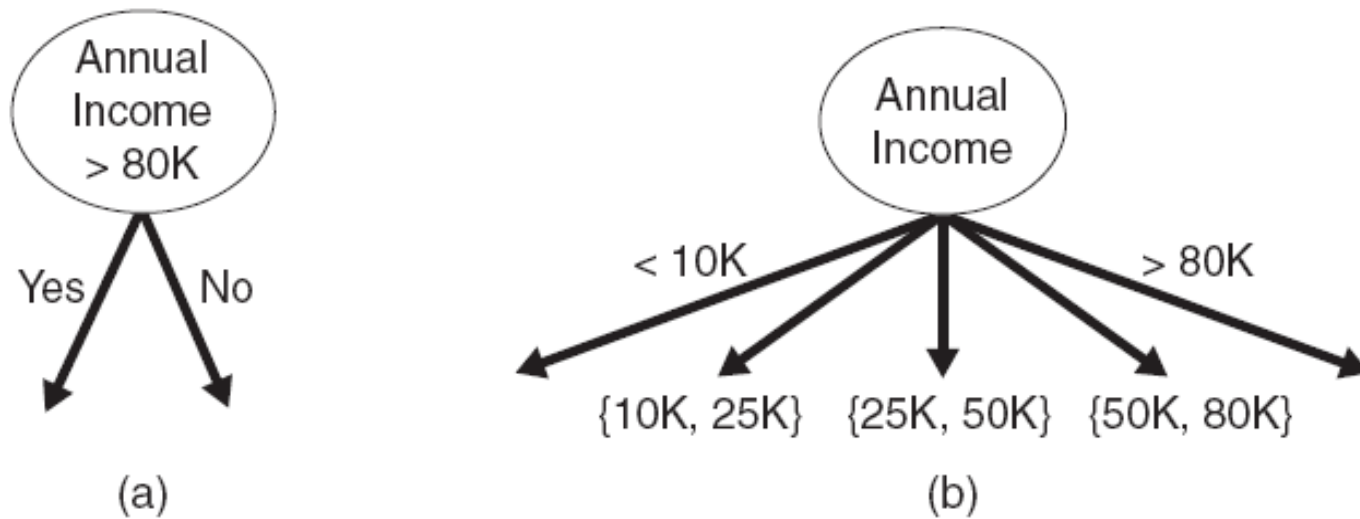
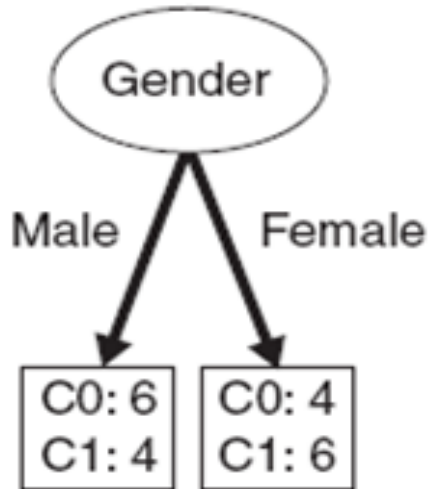


Figure 4.11. Test condition for continuous attributes.

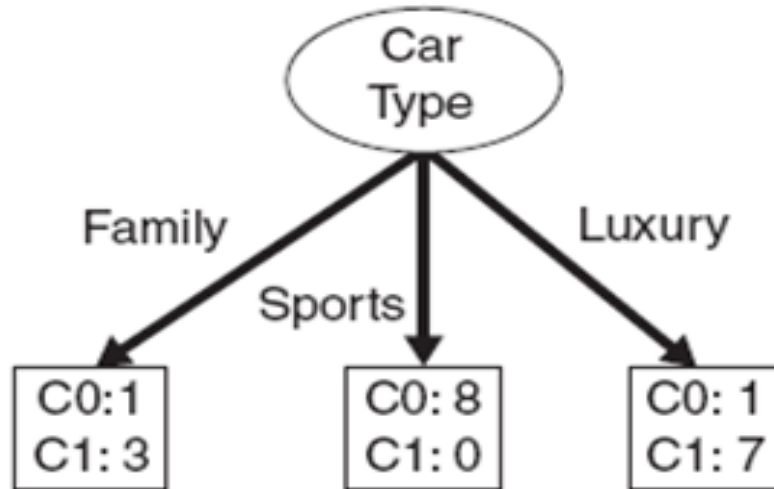
# Selecting the best split

- $p(i|t)$ : fraction of records belonging to class  $i$
- **Best split** is selected based on the degree of **impurity** of the child nodes
  - Class distribution **(0,1)** has **high purity**
  - Class distribution **(0.5,0.5)** has the **smallest purity (highest impurity)**
- **Intuition:** high purity  $\rightarrow$  small value of impurity measures  $\rightarrow$  better split

# Selecting the best split



(a)



(b)

# Selecting the best split: Impurity measures

- $p(i|t)$ : fraction of records associated with node  $t$  belonging to class  $i$

$$\text{Entropy}(t) = - \sum_{i=1}^c p(i|t) \log p(i|t)$$

$$\text{Gini}(t) = 1 - \sum_{i=1}^c [p(i|t)]^2$$

$$\text{Classification error}(t) = 1 - \max_i [p(i|t)]$$



# Range of impurity measures

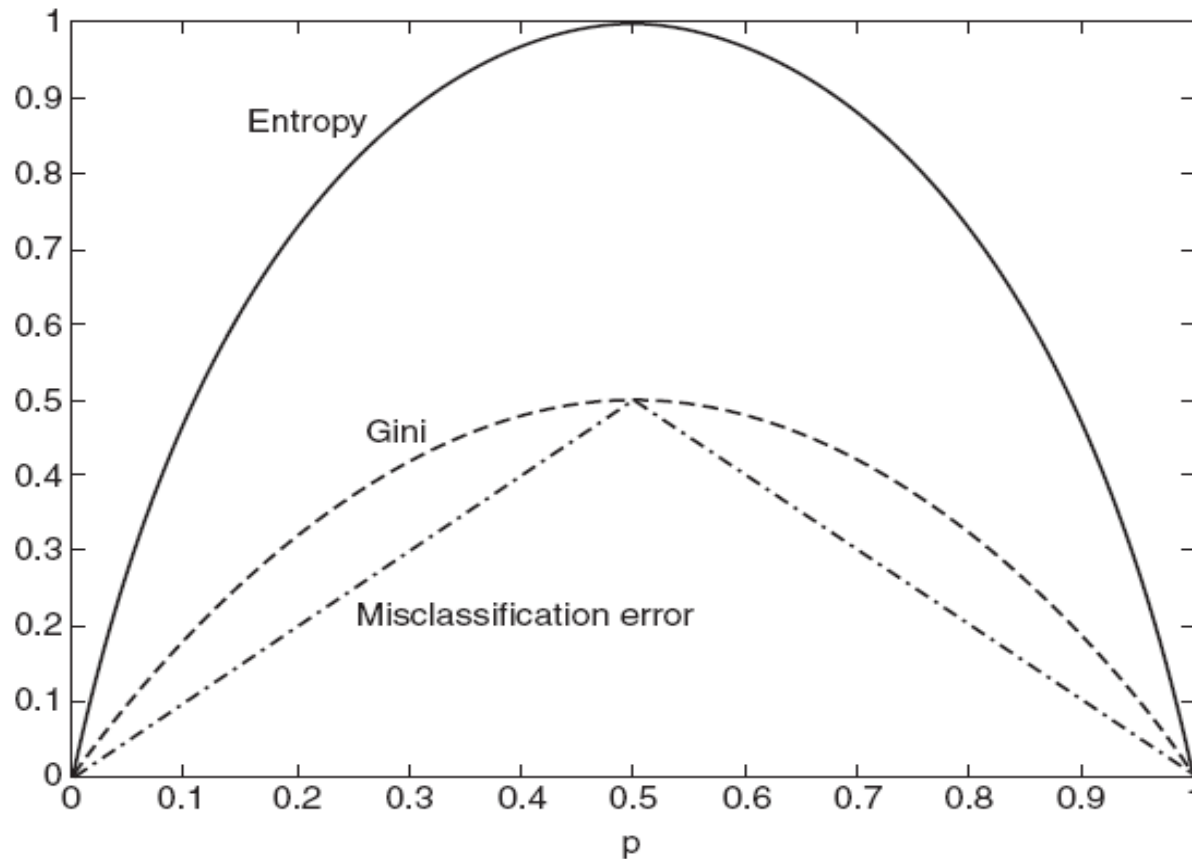


Figure 4.13. Comparison among the impurity measures for binary classification problems.

# Impurity measures

- In general the different impurity measures are ***consistent***
- ***Gain of a test condition:*** compare the impurity of the parent node with the impurity of the child nodes

$$\Delta = I(\textit{parent}) - \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j)$$

- Maximizing the gain == minimizing the weighted average impurity measure of children nodes
- If  $I() = \text{Entropy}()$ , then  $\Delta_{\text{info}}$  is called **information gain**

# Computing gain: example

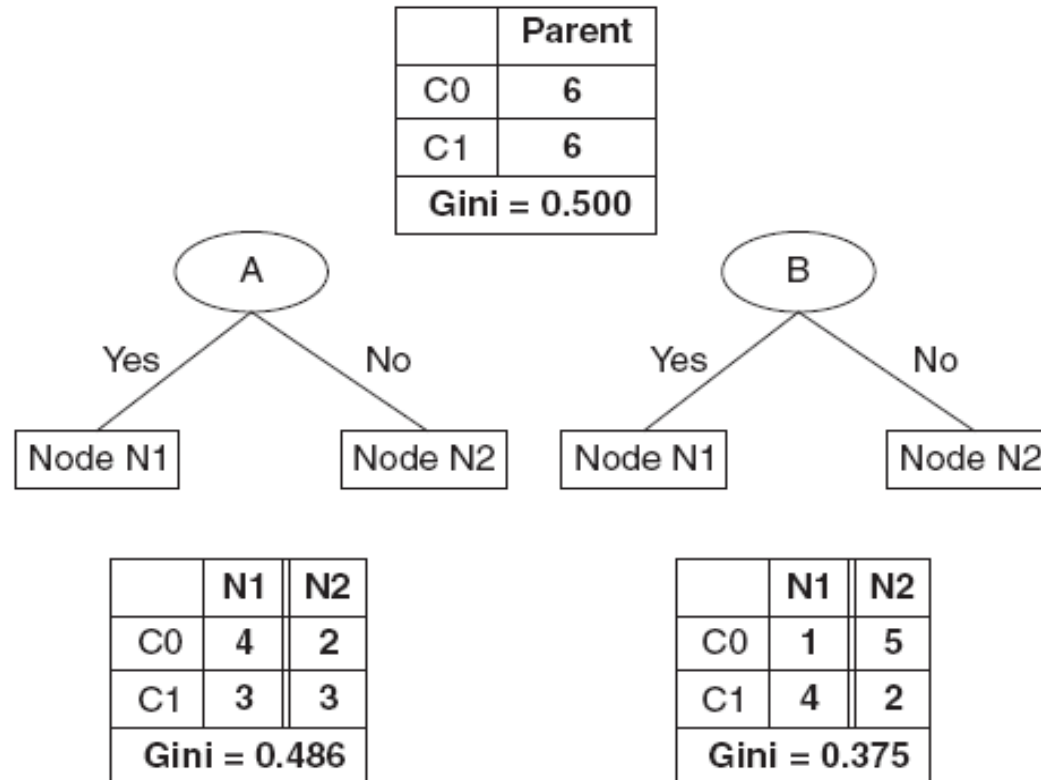


Figure 4.14. Splitting binary attributes.

# Is minimizing impurity/ maximizing $\Delta$ enough?

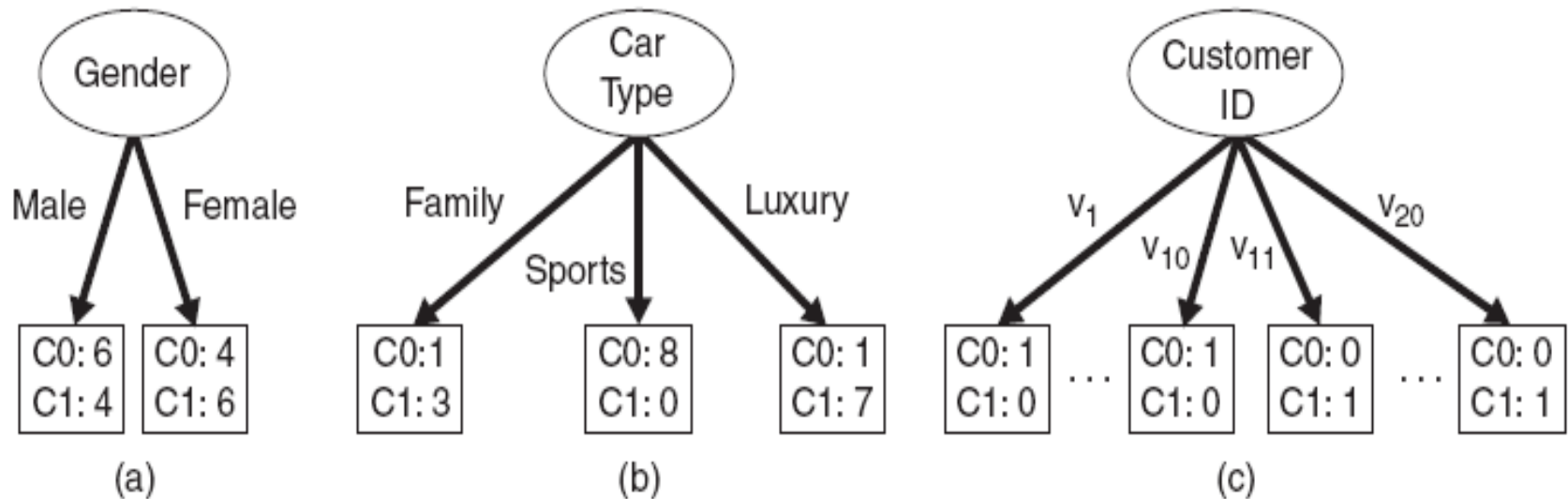


Figure 4.12. Multiway versus binary splits.

# Is minimizing impurity/ maximizing $\Delta$ enough?

- Impurity measures favor attributes with large number of values
- A test condition with large number of outcomes may not be desirable
  - # of records in each partition is too small to make predictions

# Gain ratio

- **Gain ratio** =  $\Delta_{\text{info}} / \text{SplitInfo}$
- **SplitInfo** =  $-\sum_{i=1\dots k} p(v_i) \log(p(v_i))$
- **k**: total number of splits
- If each attribute has the same number of records, **SplitInfo** = **log k**
- Large number of splits  $\rightarrow$  large **SplitInfo**  $\rightarrow$  small gain ratio

# Constructing decision-trees (pseudocode)

**GenDecTree**(Sample **S**, Features **F**)

1. If **stopping\_condition**(**S**,**F**) = true then
  - a. leaf = **createNode**()
  - b. leaf.label = **Classify**(**S**)
  - c. return leaf
2. root = **createNode**()
3. root.test\_condition = **findBestSplit**(**S**,**F**)
4. **V** = {**v** | **v** a possible outcome of root.test\_condition}
5. for *each* value **v** ∈ **V**:
  - a. **S<sub>v</sub>** := {**s** | root.test\_condition(**s**) = **v** and **s** ∈ **S**};
  - b. child = **TreeGrowth**(**S<sub>v</sub>**,**F**);
  - c. Add **child** as a descent of **root** and label the edge (**root** → **child**) as **v**
6. return root

# Stopping criteria for tree induction

- Stop expanding a node when all the records belong to the same class
- Stop expanding a node when all the records have similar attribute values
- Early termination



# Advantages of decision trees

- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees
- Accuracy is comparable to other classification techniques for many simple data sets

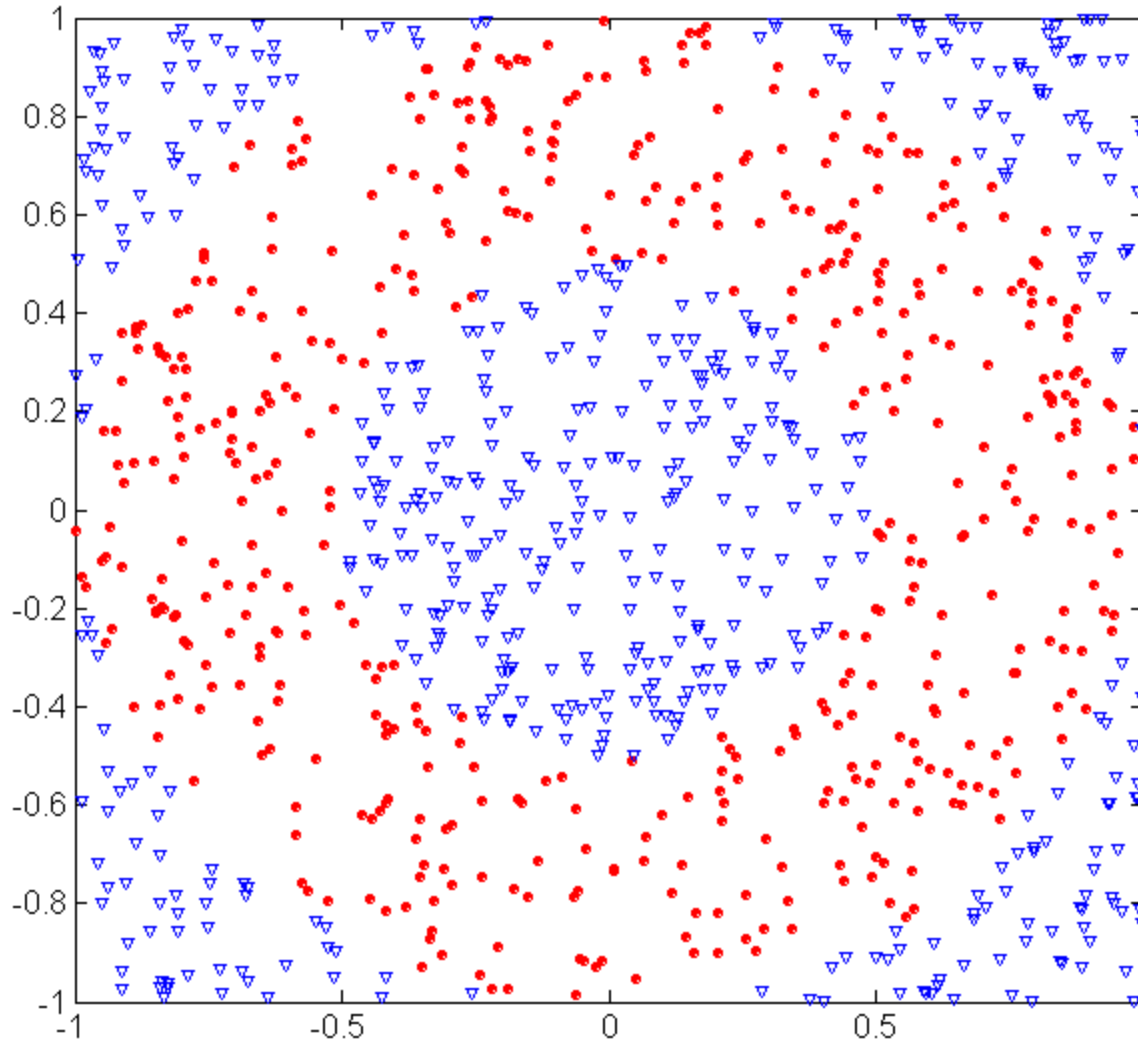
# Example: C4.5 algorithm

- Simple depth-first construction.
- Uses Information Gain
- Sorts Continuous Attributes at each node.
- Needs entire data to fit in memory.
- Unsuitable for Large Datasets.
  
- You can download the software from:  
<http://www.cse.unsw.edu.au/~quinlan/c4.5r8.tar.gz>

# Practical problems with classification

- Unerfitting and overfitting
- Missing values
- Cost of classification

# Underfitting and overfitting



triangular data points.

**Circular points:**

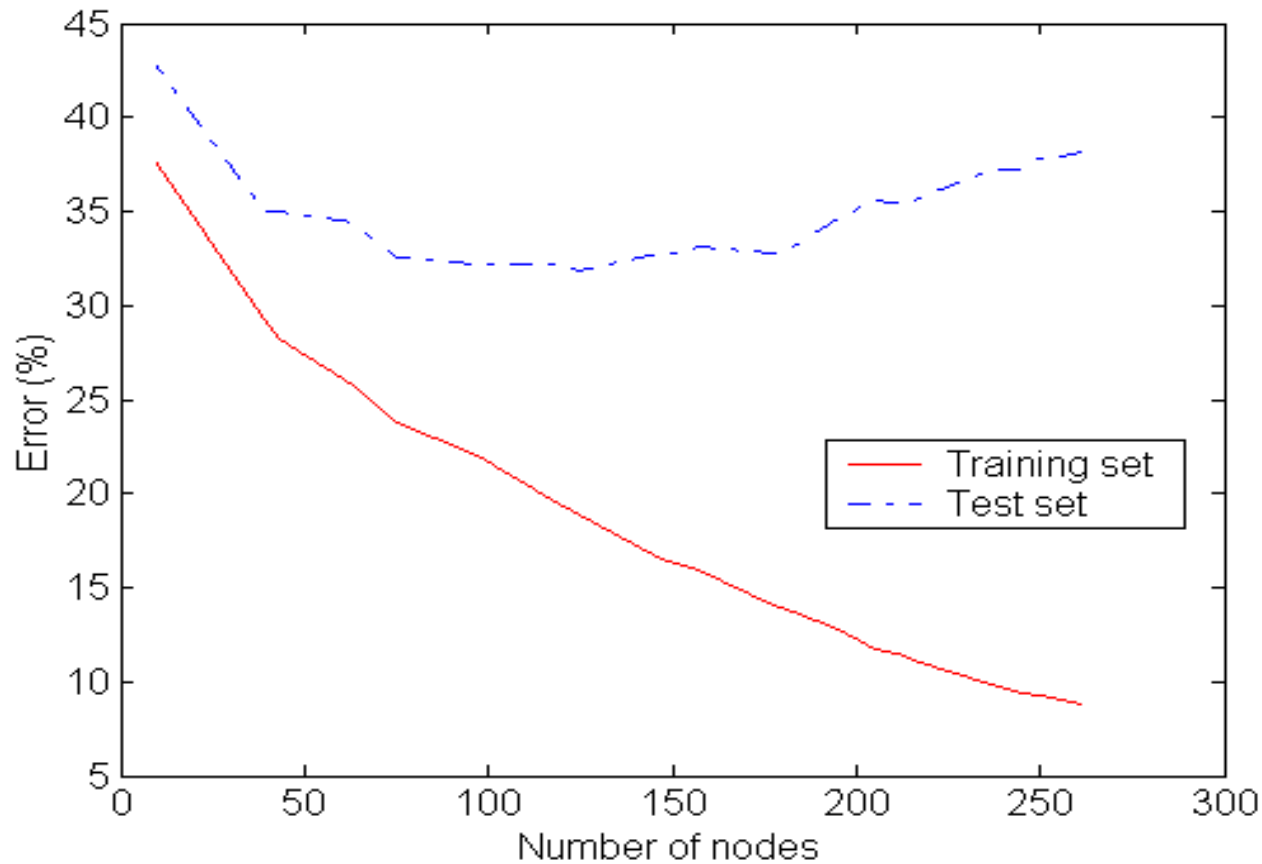
$$0.5 \leq \text{sqrt}(x_1^2 + x_2^2) \leq 1$$

**Triangular points:**

$$\text{sqrt}(x_1^2 + x_2^2) > 1 \text{ or}$$

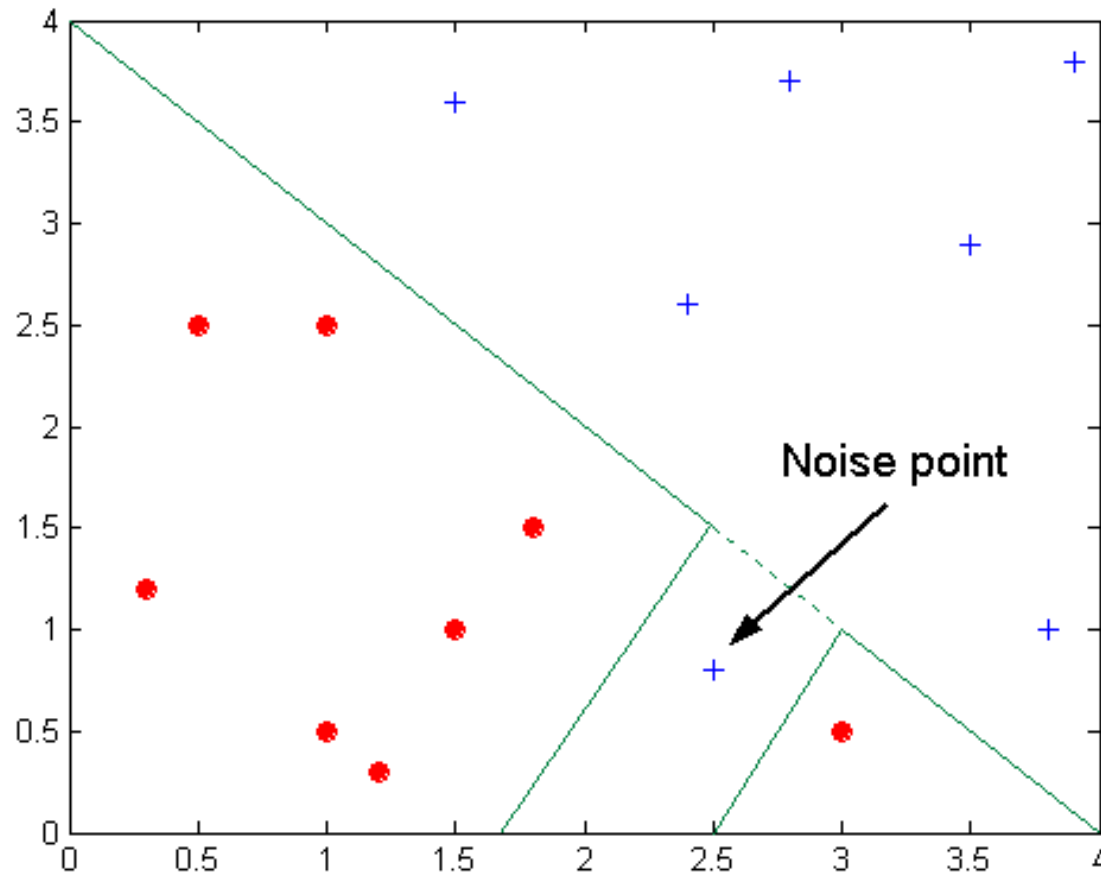
$$\text{sqrt}(x_1^2 + x_2^2) < 0.5$$

# Overfitting and underfitting



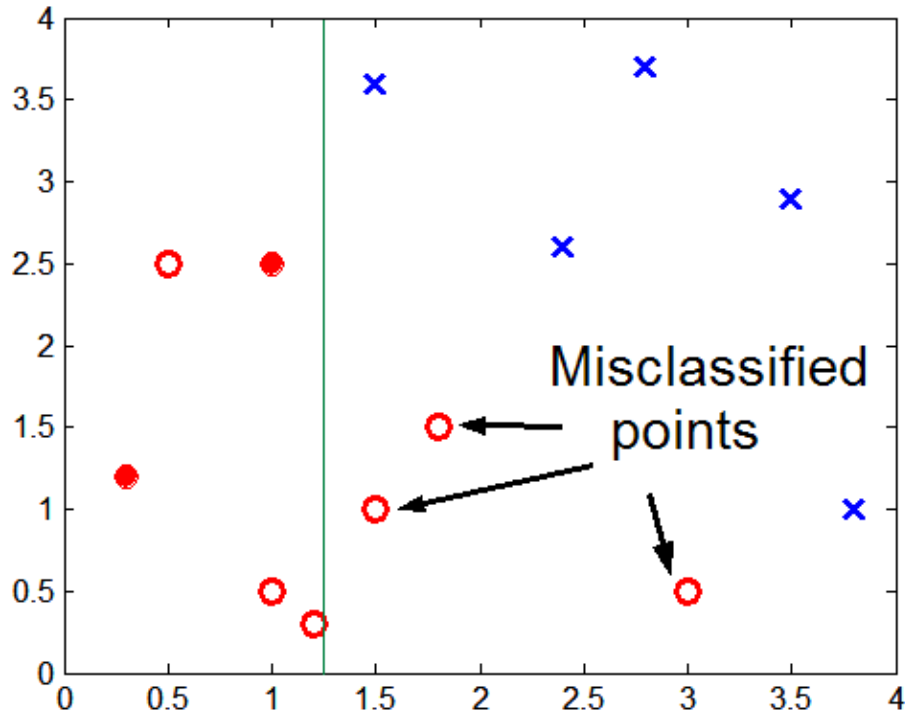
**Underfitting:** when model is too simple, both training and test errors are large

# Overfitting due to noise



Decision boundary is distorted by noise point

# Overfitting due to insufficient samples



Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

# Overfitting: course of action

- Overfitting results in decision trees that are more complex than necessary
- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records
- Need new ways for estimating errors



# Methods for estimating the error

- **Re-substitution errors:** error on training ( $\sum e(t)$  )
- **Generalization errors:** error on testing ( $\sum e'(t)$ )
- Methods for estimating generalization errors:
  - **Optimistic approach:**  $e'(t) = e(t)$
  - **Pessimistic approach:**
    - For each leaf node:  $e'(t) = (e(t)+0.5)$
    - Total errors:  $e'(T) = e(T) + N \times 0.5$  (N: number of leaf nodes)
    - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
      - Training error =  $10/1000 = 1\%$
      - Generalization error =  $(10 + 30 \times 0.5)/1000 = 2.5\%$
  - **Reduced error pruning (REP):**
    - uses **validation data set** to estimate generalization error

# Addressing overfitting: Occam's razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
- For complex models, there is a greater chance that it was fitted accidentally by errors in data
- Therefore, one should include model complexity when evaluating a model

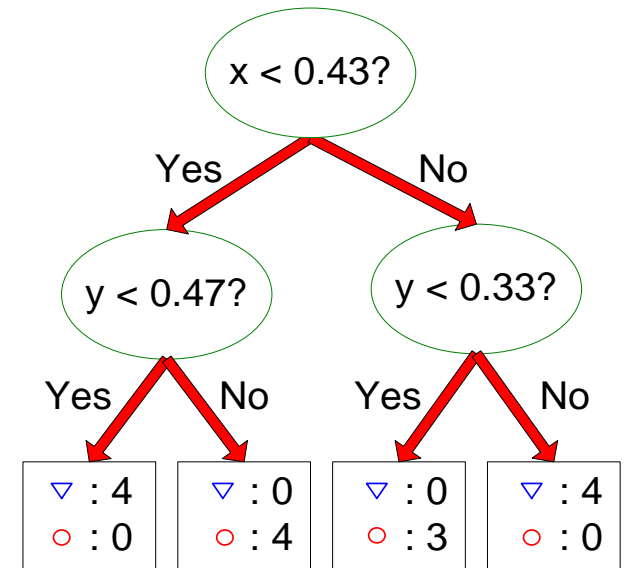
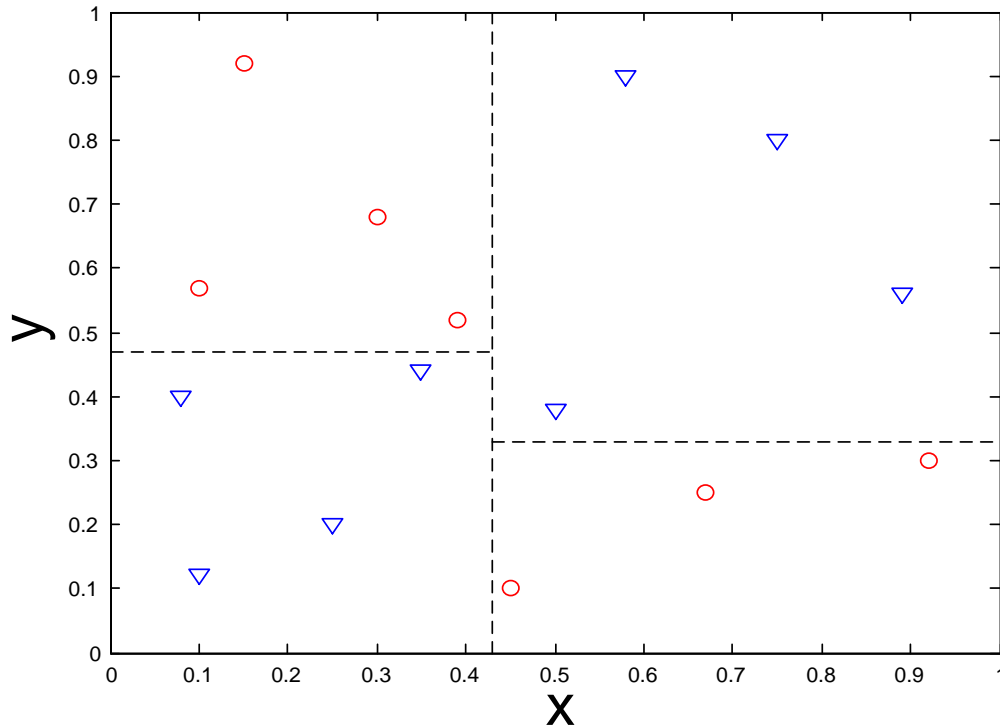
# Addressing overfitting: postpruning

- Grow decision tree to its entirety
- Trim the nodes of the decision tree in a bottom-up fashion
- If generalization error improves after trimming, replace sub-tree by a leaf node.
- Class label of leaf node is determined from majority class of instances in the sub-tree
- Can use MDL for post-pruning

# Addressing overfitting: preprunning

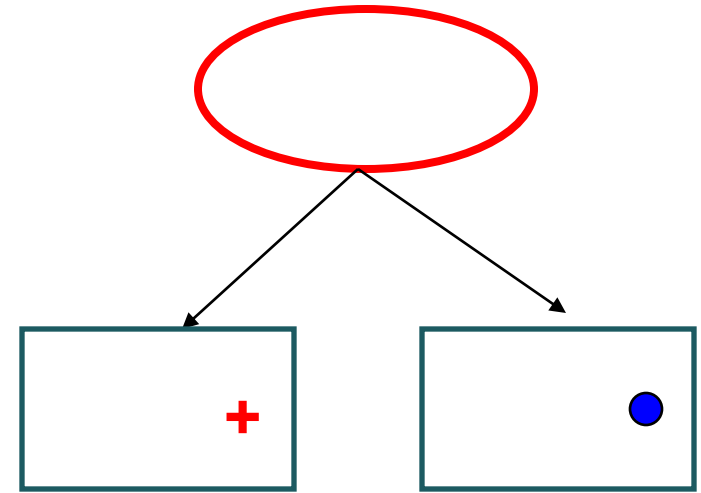
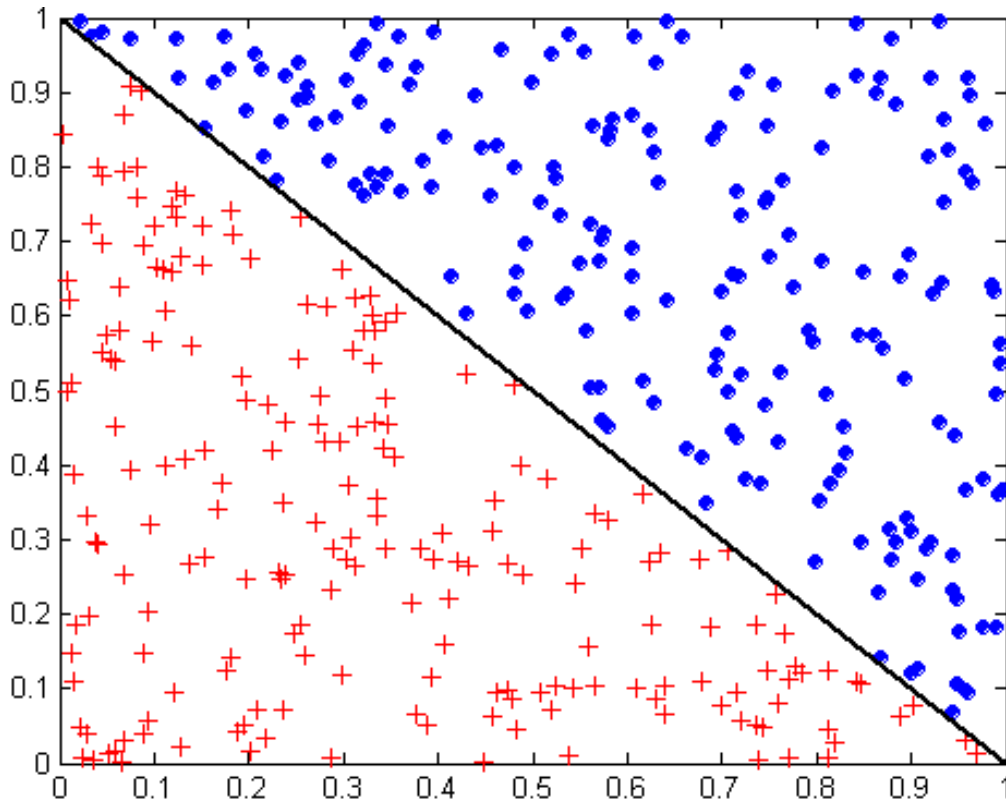
- Stop the algorithm before it becomes a fully-grown tree
- Typical stopping conditions for a node:
  - Stop if all instances belong to the same class
  - Stop if all the attribute values are the same
- More restrictive conditions:
  - Stop if number of instances is less than some user-specified threshold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

# Decision boundary for decision trees



- Border line between two neighboring regions of different classes is known as **decision boundary**
- **Decision boundary in decision trees** is parallel to axes because test condition involves a single attribute at-a-time

# Oblique Decision Trees



**Not all datasets can be partitioned optimally using test conditions involving single attributes!**