RV College of Engineering

Go, change the world

UNIT -3

- **Supervised Learning**
- **Decision Tree Classifier**
- **Model Overfitting**
- **Model Selection**
- **Model Evaluation**

UNIT-3 Supervised Learning

Basic Concepts, General Framework for Classification

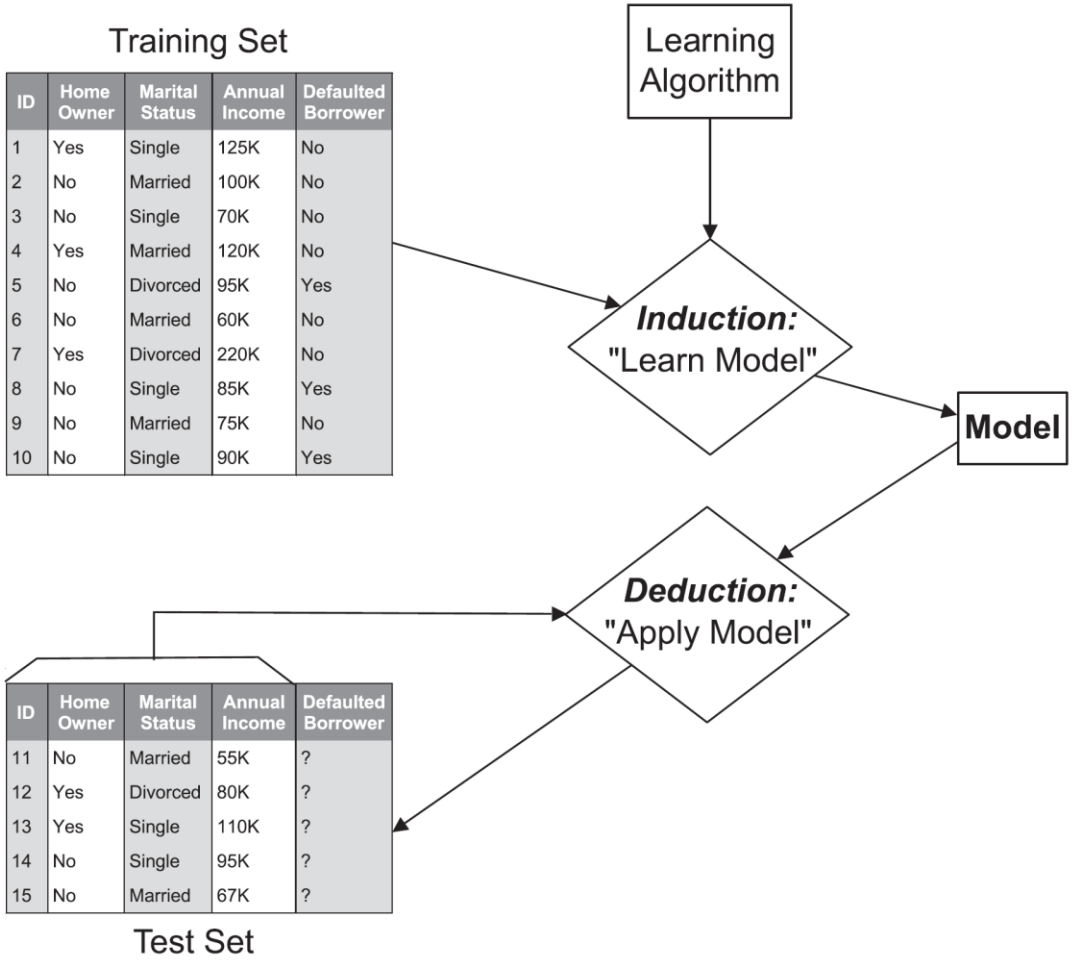
Classification: Definition

- Given a collection of records (training set)
  - Each record is by characterized by a tuple ($\boldsymbol{x}$,y), where $\boldsymbol{x}$ is the attribute set and y is the class label
    - $\boldsymbol{x}$: attribute, predictor, independent variable, input
    - y: class, response, dependent variable, output
- Task:
  - Learn a model that maps each attribute set $\boldsymbol{x}$ into one of the predefined class labels y

Examples of Classification Task

Task	Attribute set, $\boldsymbol{x}$	Class label, y
Categorizing email messages	Features extracted from email message header and content	spam or non-spam
Identifying tumor cells	Features extracted from x-rays or MRI scans	malignant or benign cells
Cataloging galaxies	Features extracted from telescope images	Elliptical, spiral, or irregular-shaped galaxies

General Approach for Building Classification Model



ID	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

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
11	No	Married	55K	?
12	Yes	Divorced	80K	?
13	Yes	Single	110K	?
14	No	Single	95K	?
15	No	Married	67K	?

Figure 3.3. General framework for building a classification model.

General Approach for Building Classification Model

Table 3.4. Confusion matrix for a binary classification problem.

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

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}.$$

$$\text{Accuracy} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

Classification Techniques

The Type of Data

- Data sets differ in a number of ways.
- For example, the attributes used to describe data objects can be of different types—***quantitative or qualitative***
- data sets often have ***special characteristics;***
  - e.g., some *data sets contain time series or objects with explicit relationships* to one another.
- the ***type of data determines*** which tools and techniques can be used to analyze the data.

Classification Techniques

The Types of Data

- A **data set** can often be viewed as a collection of **data objects**.
  - Other names for a data object are *record*, *point*, *vector*, *pattern*, *event*, *case*, *sample*, *instance*, *observation*, or *entity*.
- data objects are described by a number of **attributes** that capture the characteristics of an object, such as the mass of a physical object or the time at which an event occurred.
  - Other names for an attribute are *variable*, *characteristic*, *field*, *feature*, or *dimension*.

Classification Techniques

Attributes and Measurement

- An **attribute** is a property or characteristic of an object that can vary, either from one object to another or from one time to another.
  - For example, eye color varies from person to person, while the temperature of an object varies over time.
  - eye color is a symbolic attribute with a small number of possible values *{brown, black, blue, green, hazel, etc.}*,
  - temperature is a numerical attribute with a potentially unlimited number of values.
- At the most basic level, attributes are not about numbers or symbols.

Classification Techniques

The Different Types of Attributes

- A useful (and simple) way **to specify the type of an attribute** is to *identify the properties of numbers that correspond to underlying properties of the attribute.*
- For example, an attribute such as length has many of the properties of numbers. It makes sense to compare and order objects by length, as well as to talk about the differences and ratios of length.
- The following properties (operations) of numbers are typically used to describe attributes.

1. **Distinctness** $=$ and $\neq$
2. **Order** $<$, $\leq$, $>$, and $\geq$
3. **Addition** $+$ and $-$
4. **Multiplication** $\times$ and $/$

Classification Techniques

The Different Types of Attributes

- Given these properties, we can define four types of attributes: **nominal**, **ordinal**, **interval**, and **ratio**.
- Table 2.2 gives the definitions of these types, along with information about the statistical operations that are valid for each type.
- Each attribute type possesses all of the properties and operations of the attribute types above it.
- Consequently, any property or operation that is valid for nominal, ordinal, and interval attributes is also valid for ratio attributes.

Classification Techniques

The Different Types of Attributes

Table 2.2. Different attribute types.

Attribute Type		Description	Examples	Operations
Categorical (Qualitative)	Nominal	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another. $(=, \neq)$	zip codes, employee ID numbers, eye color, gender	mode, entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. $(<, >)$	hardness of minerals, $\{good, better, best\}$, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. $(+, -)$	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviation, Pearson's correlation, t and F tests
	Ratio	For ratio variables, both differences and ratios are meaningful. $(\times, /)$	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

Classification Techniques

- Nominal and ordinal attributes are collectively referred to as **categorical** or **qualitative** attributes.
  - qualitative attributes, such as employee ID, lack most of the properties of numbers.
  - Even if they are represented by numbers, i.e., integers, they should be treated more like symbols.
- The remaining two types of attributes, interval and ratio, are collectively referred to as **quantitative** or **numeric** attributes.
  - Quantitative attributes are represented by numbers and have most of the properties of numbers.
  - quantitative attributes can be integer-valued or continuous.

Classification Techniques

- Base Classifiers
  - Decision Tree based Methods
  - Rule-based Methods
  - Nearest-neighbor
  - Naïve Bayes and Bayesian Belief Networks
  - Support Vector Machines
  - Neural Networks, Deep Neural Nets

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

UNIT-3 : Decision Tree Classifier

A Basic Algorithm to Build a Decision Tree, Methods for Expressing Attribute Test Conditions, Measures for Selecting an Attribute Test Condition, Algorithm for Decision Tree Induction, Characteristics of Decision Tree Classifiers,

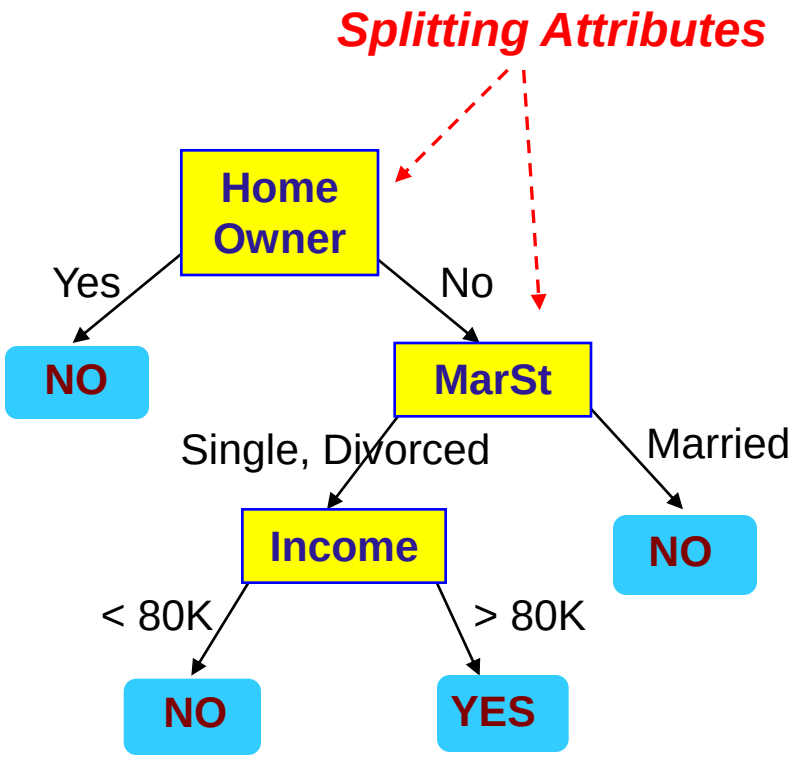
Decision Tree Classifier

- a simple classification technique
- To illustrate how a decision tree works, consider the classification problem (Examples)

Example of a Decision Tree

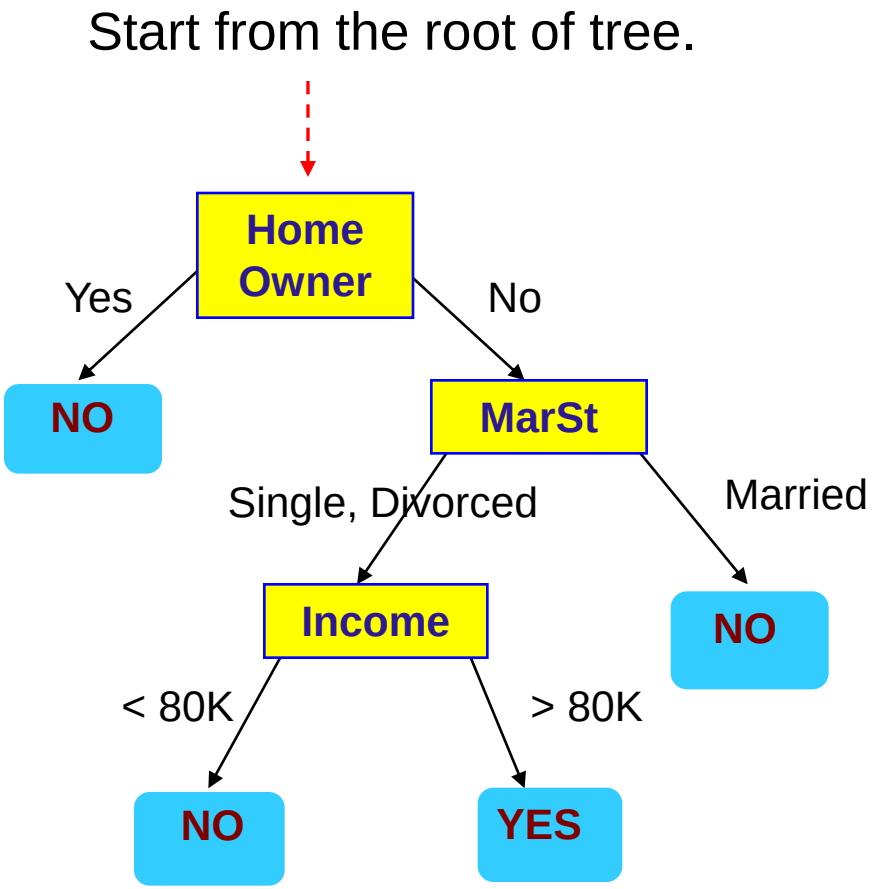
ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
		categorical	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

Apply Model to Test Data



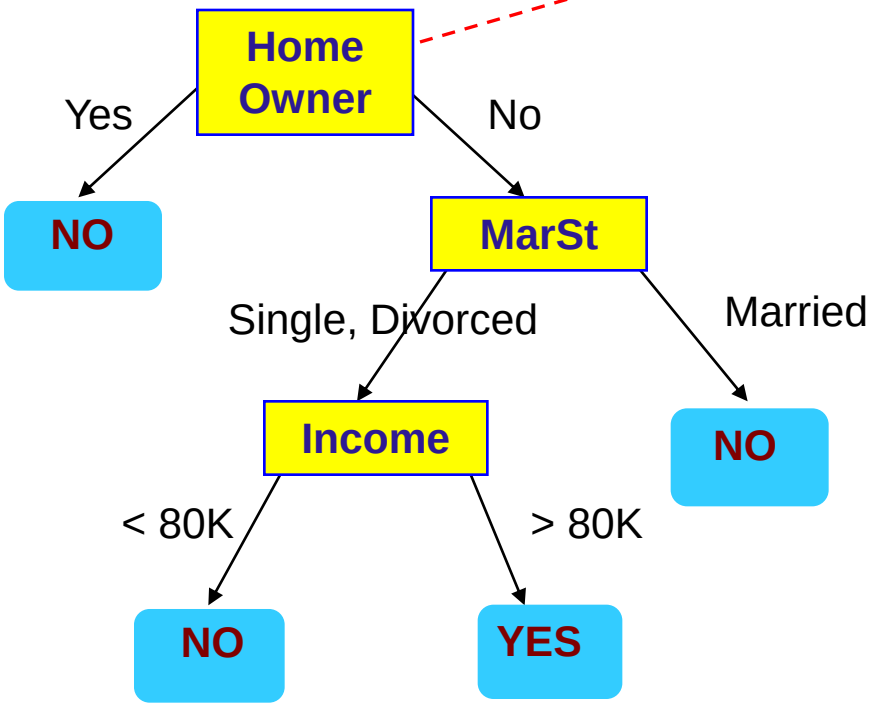
Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Apply Model to Test Data

Test Data

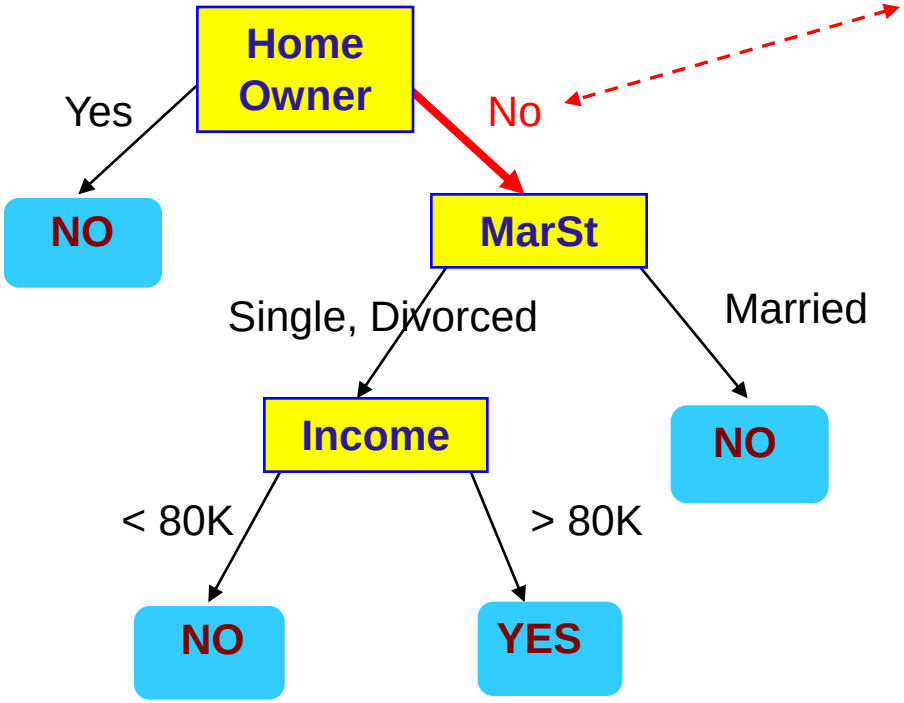
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

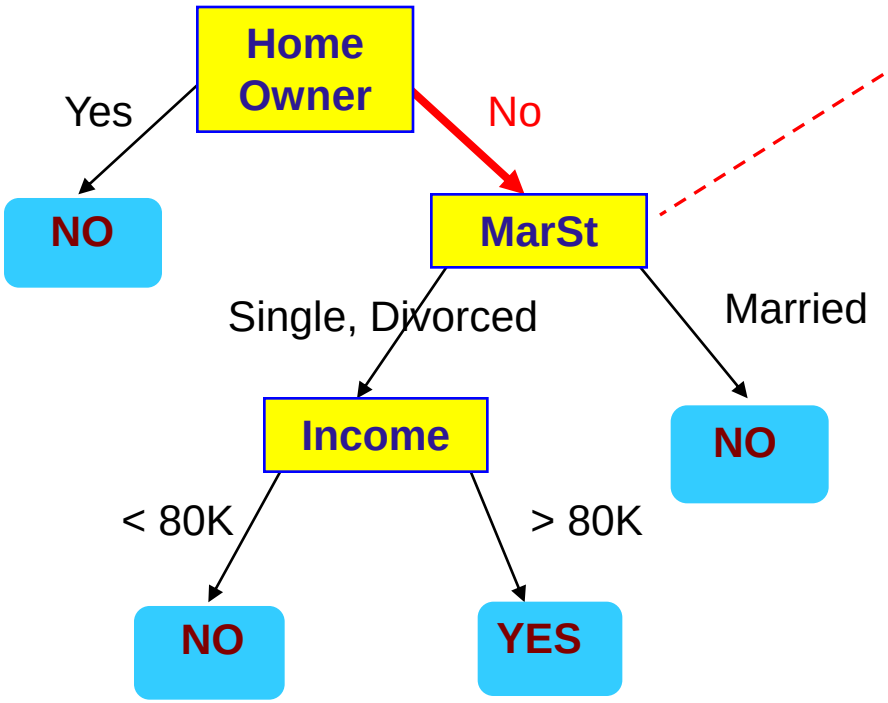
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

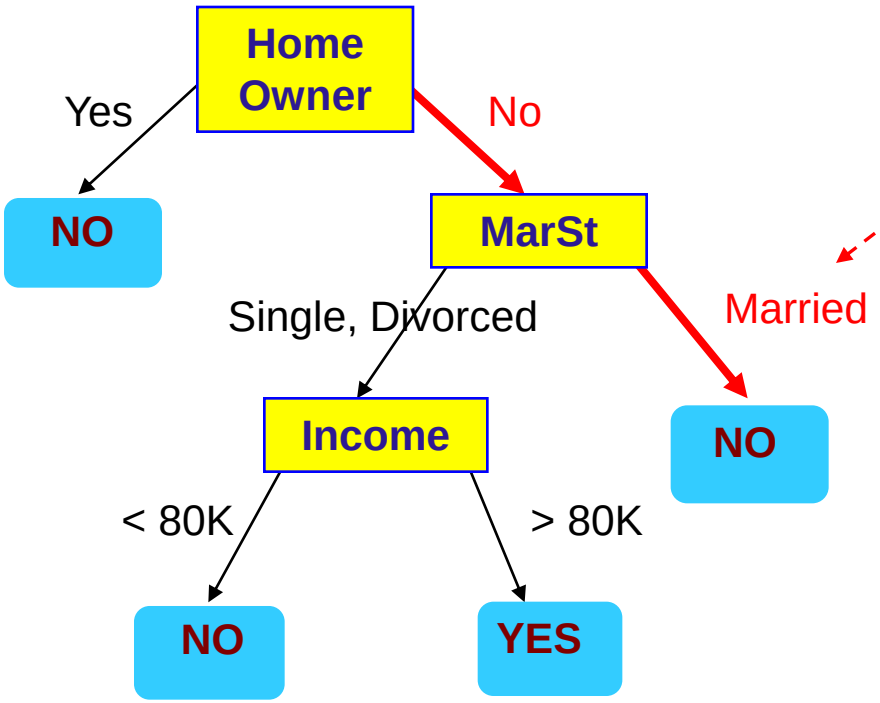
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

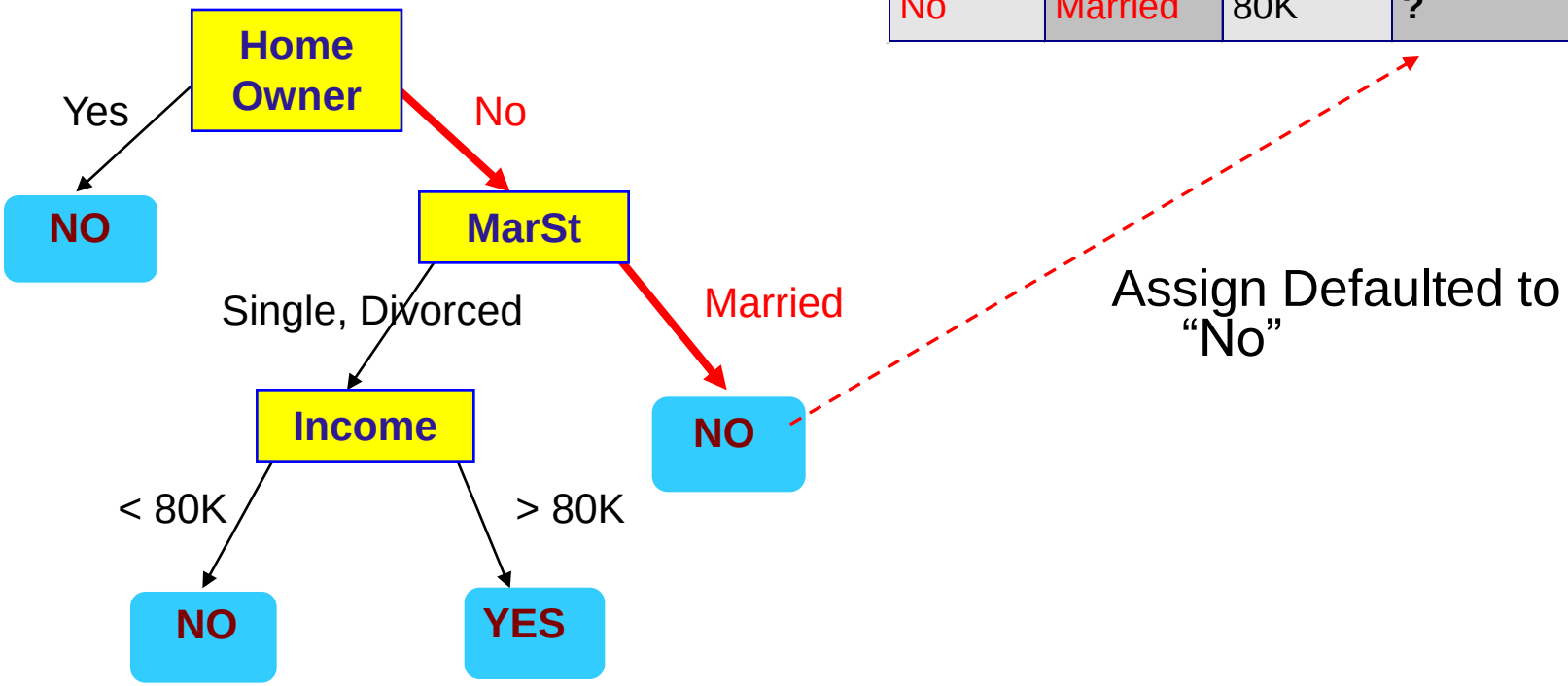
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

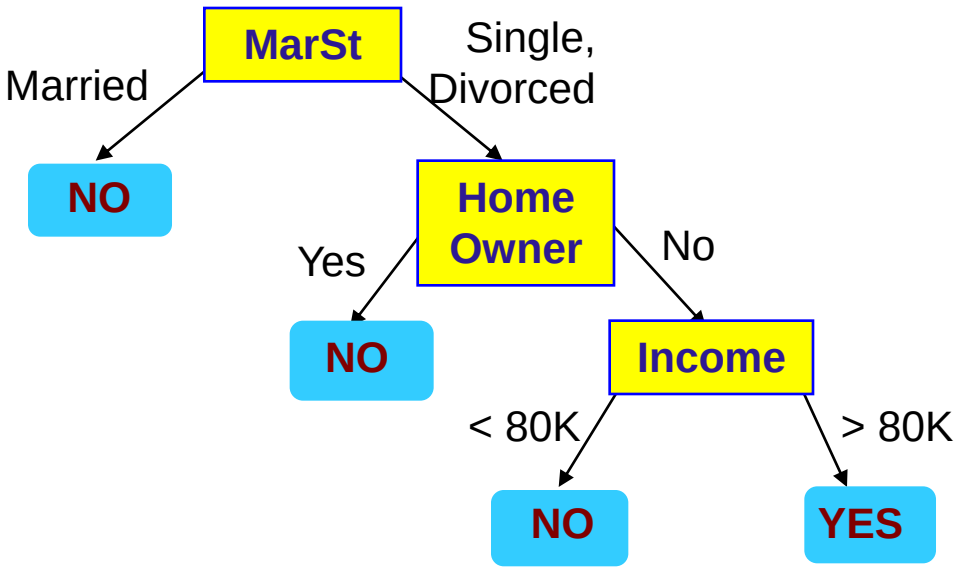
Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Another Example of Decision Tree

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
	categorical	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

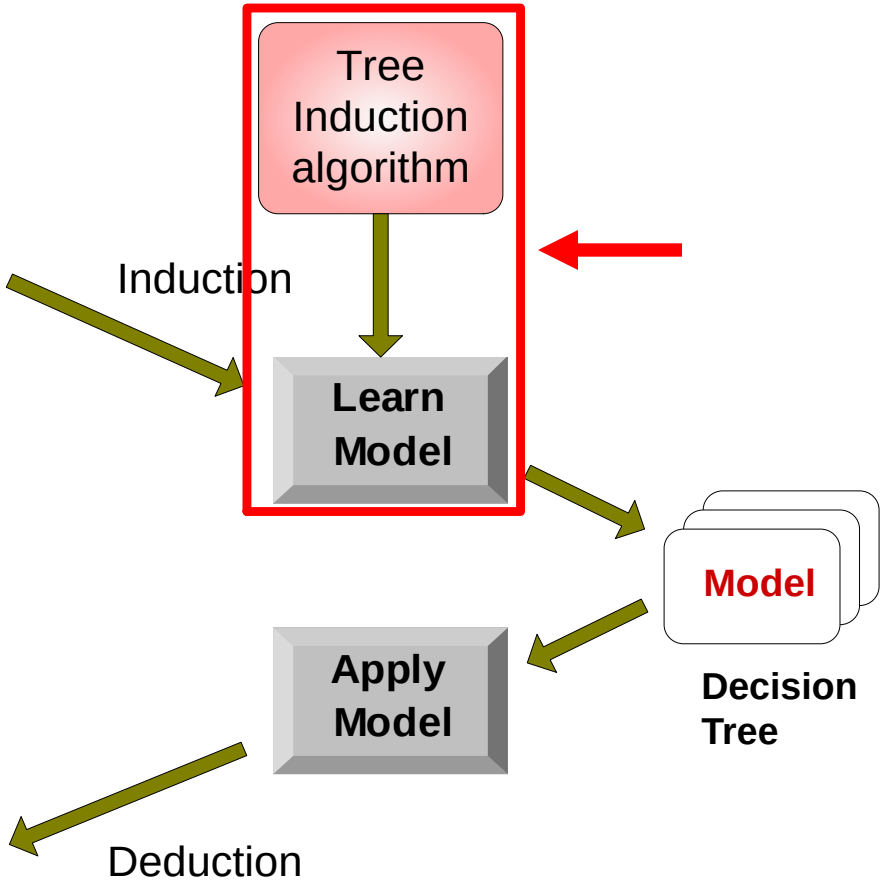


There could be more than one tree that fits the same data!

Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set



Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

Decision Tree Classification Task

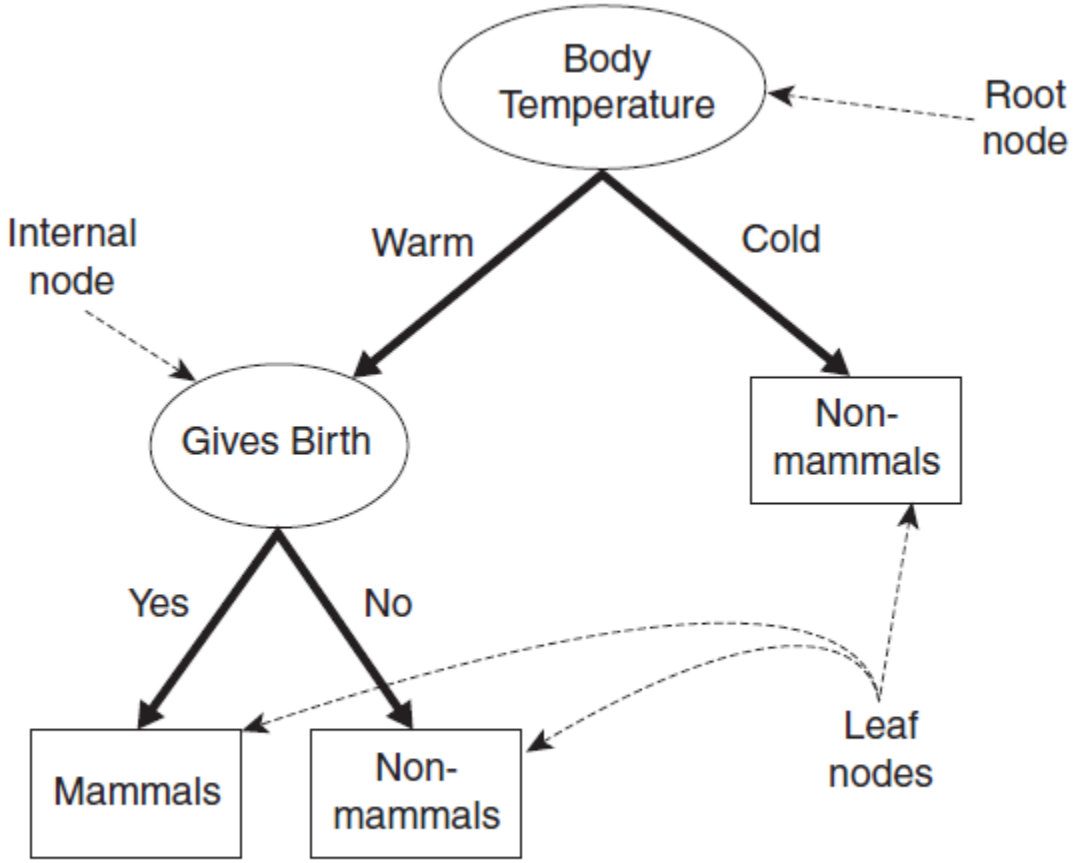


Figure 3.4. A decision tree for the mammal classification problem.

Decision Tree Classification Task

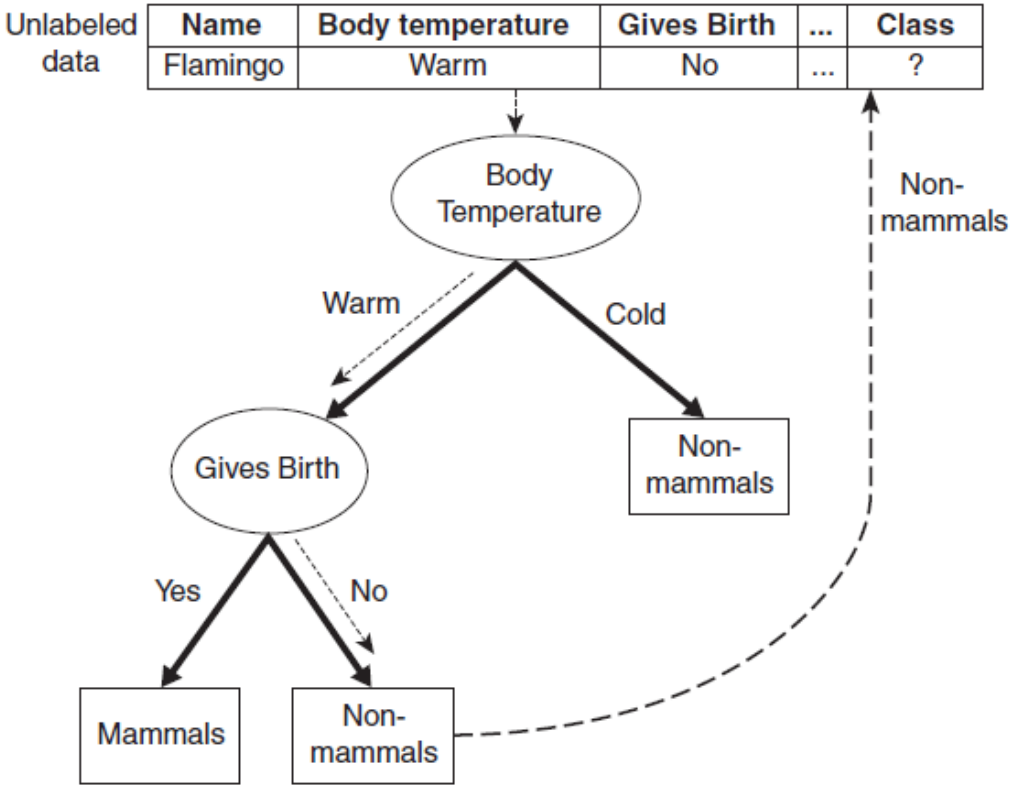


Figure 3.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the `Non-mammals` class.

Decision Tree Classifier

- The previous ***example illustrates how we can solve a classification problem***
  - by asking a series of carefully crafted questions about the attributes of the test instance.
  - Each time we receive an answer, we could ask a follow-up question until we can conclusively decide on its class label.
  - The series of questions and their possible answers can be organized into a hierarchical structure called a ***decision tree***.

The **tree has three types of nodes:**

- A **root node**, with no incoming links and zero or more outgoing links.
- **Internal nodes**, each of which has exactly one incoming link and two or more outgoing links.
- **Leaf** or **terminal** nodes, each of which has exactly one incoming link and no outgoing links.

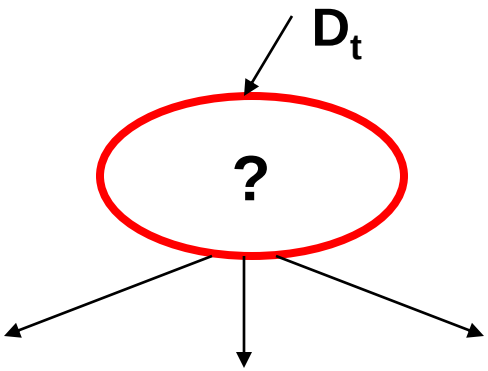
Decision Tree Induction

- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ,SPRINT

General Structure of Hunt's Algorithm

- Let D_t be the set of training records that reach a node t
- General Procedure:
  - If D_t contains records that belong the same class y_t, then t is a leaf node labeled as y_t
  - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

ID	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



Hunt's Algorithm

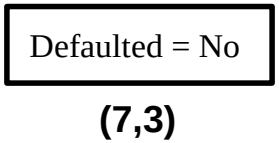
Defaulted = No

(7,3)

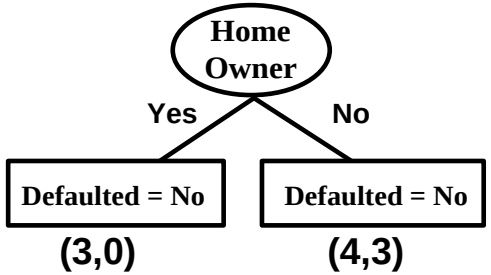
(a)

ID	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

Hunt's Algorithm



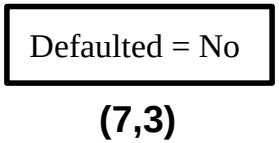
(a)



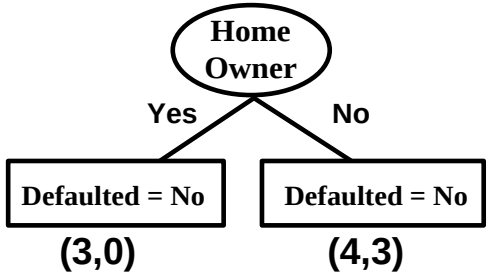
(b)

ID	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

Hunt's Algorithm

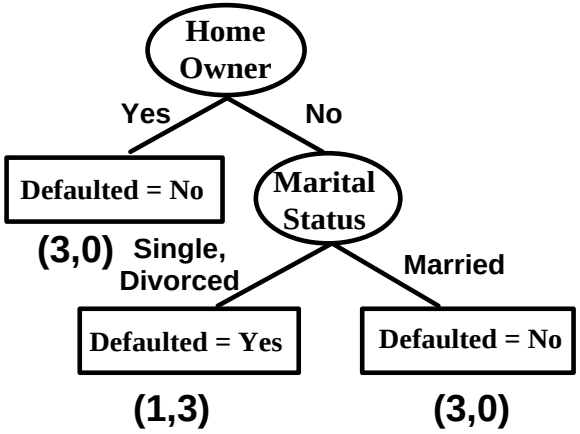


(a)



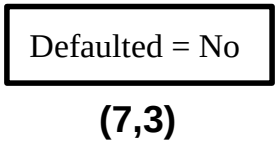
(b)

ID	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

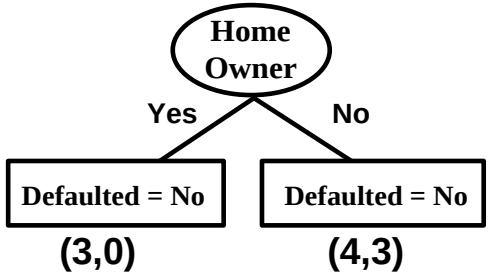


(c)

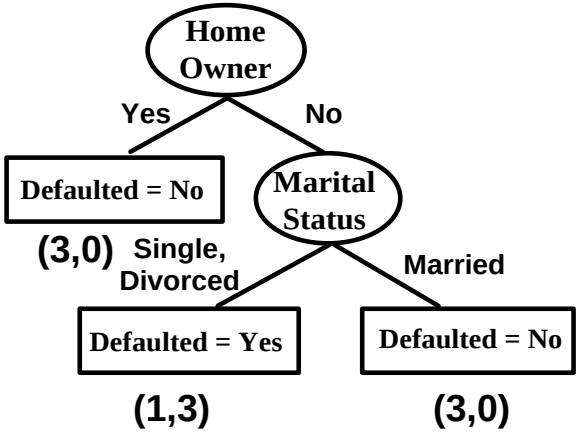
Hunt's Algorithm



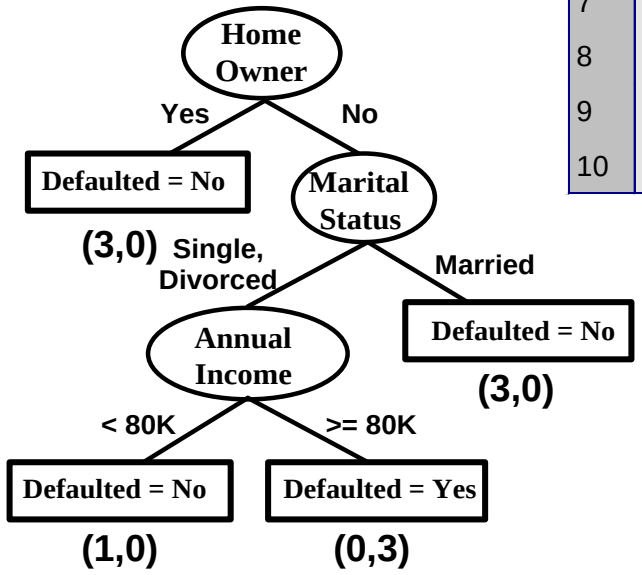
(a)



(b)



(c)



(d)

ID	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

Design Issues of Decision Tree Induction

- **Hunt's algorithm** is a generic procedure for growing decision trees in a greedy fashion. To implement the algorithm, ***there are two key design issues that must be addressed.***
  - *What is the splitting criterion?*
  - *What is the stopping criterion?*
- How should training records be split?
  - Method for expressing test condition
    - depending on attribute types
  - Measure for evaluating the goodness of a test condition
- How should the splitting procedure stop?
  - Stop splitting if all the records belong to the same class or have identical attribute values
  - Early termination

Methods for Expressing Test Conditions

- **Decision tree induction algorithms** must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types.
- Depends on attribute types
  - Binary
  - Nominal
  - Ordinal
  - Continuous

Test Condition for Binary Attributes

Binary Attributes

- The test condition for a binary attribute generates two potential outcomes

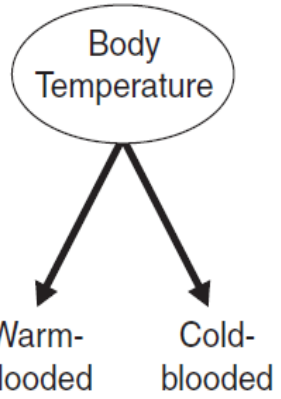
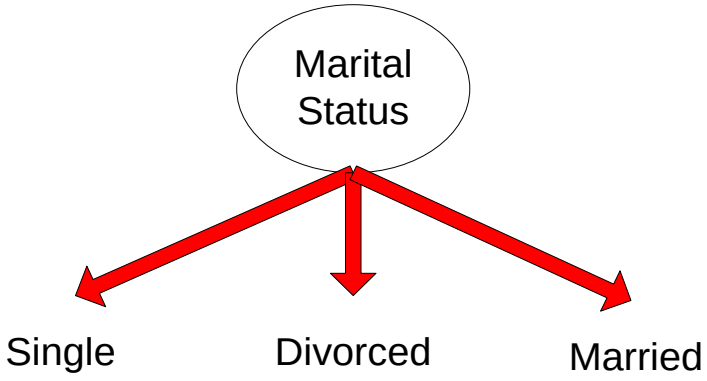


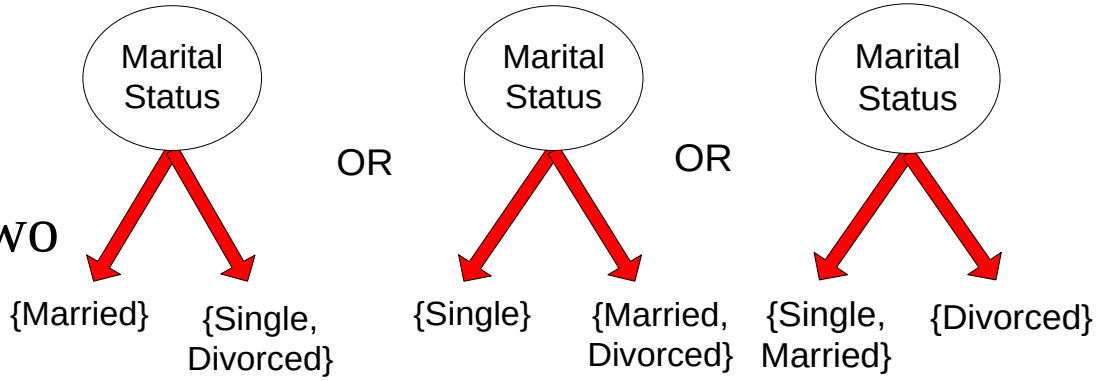
Figure 3.7. Attribute test condition for a binary attribute.

Test Condition for Nominal Attributes

- A **nominal attribute** can have many values, its attribute test condition can be expressed in two ways,
  - as a multiway split or
  - a binary split

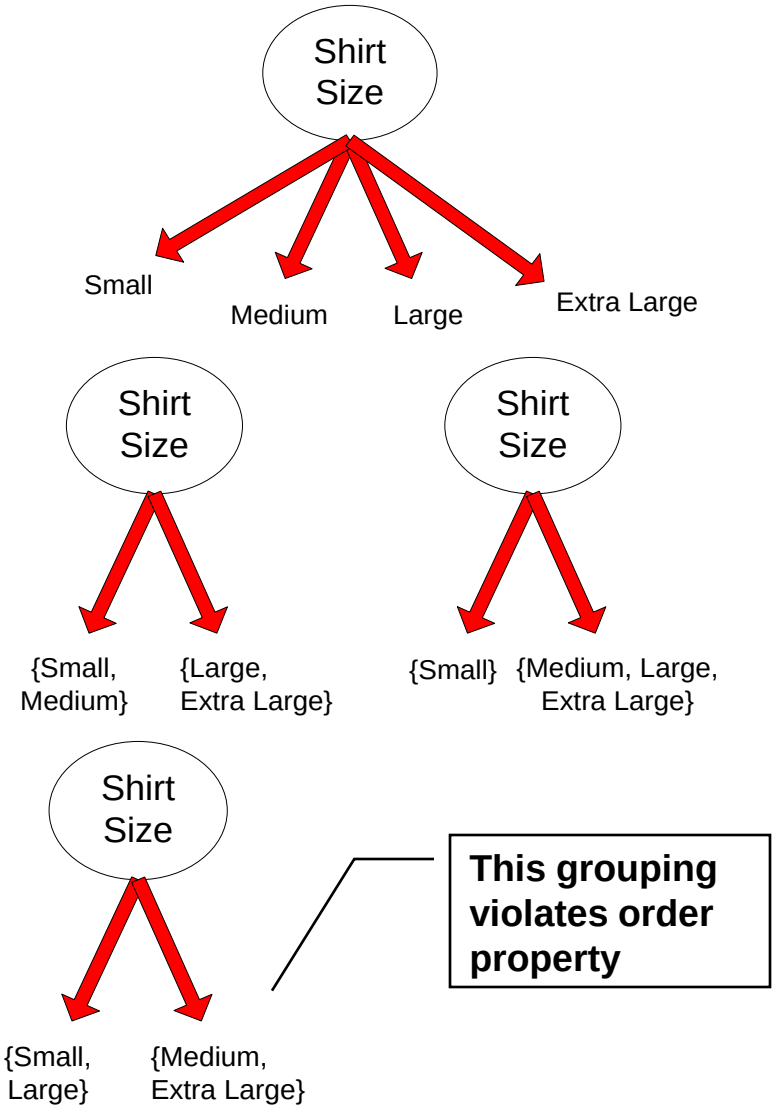


- Multi-way split:
  - Use as many partitions as distinct values.
- Binary split:
  - Divides values into two subsets

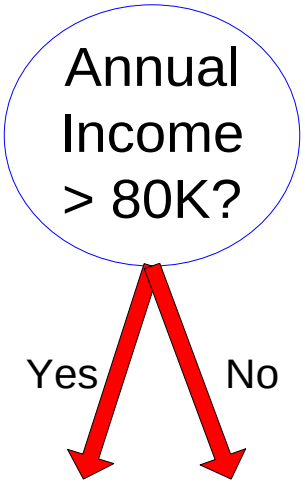


Test Condition for Ordinal Attributes

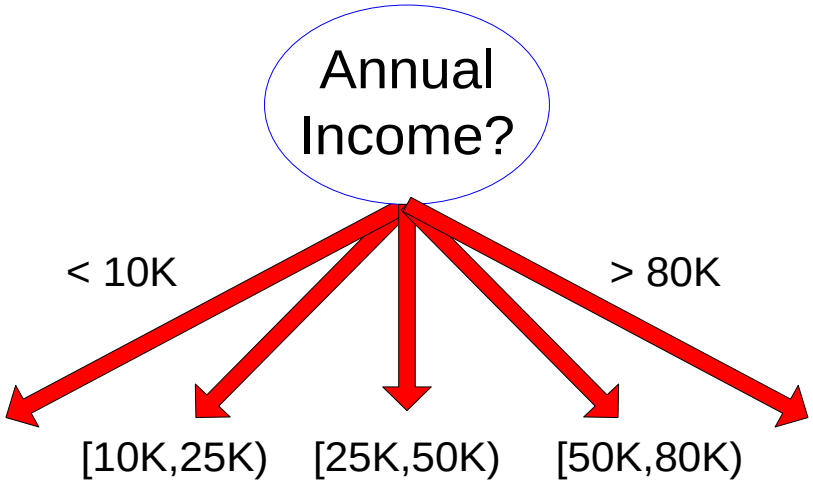
- Ordinal attributes can also produce ***binary or multiway splits***.
- Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values.
- Multi-way split:
  - Use as many partitions as distinct values
- Binary split:
  - Divides values into two subsets
  - Preserve order property among attribute values



Test Condition for Continuous Attributes



(i) Binary split



(ii) Multi-way split

Splitting Based on Continuous Attributes

- **For continuous attributes, the attribute test condition** can be expressed –
  - a comparison test (e.g., $A < v$) producing a ***binary split,***
  - or as a range query of the form $vi \leq A < vi+1$, for $i = 1, \ldots, k$, producing a ***multiway split***.
- The difference between these approaches is shown in Figure (previous slide)
- For the **binary split,** any possible value v between the minimum and maximum attribute values in the training data can be used for constructing the comparison test $A < v$. However, it is sufficient to only consider distinct attribute values in the training set as candidate split positions.
- For the **multiway split**, any possible collection of attribute value ranges can be used, as long as they are mutually exclusive and cover the entire range of attribute values between the minimum and maximum values observed in the training set.

Splitting Based on Continuous Attributes

Different ways of handling:

One approach for constructing multiway splits - apply the **discretization** strategies

- Discretization to form an ordinal categorical attribute

  Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.

  - Static – discretize once at the beginning
  - Dynamic – repeat at each node

- Binary Decision: $(A < v)$ or $(A \geq v)$

  - consider all possible splits and finds the best cut
  - can be more compute intensive

Measures for Selecting an Attribute Test Condition

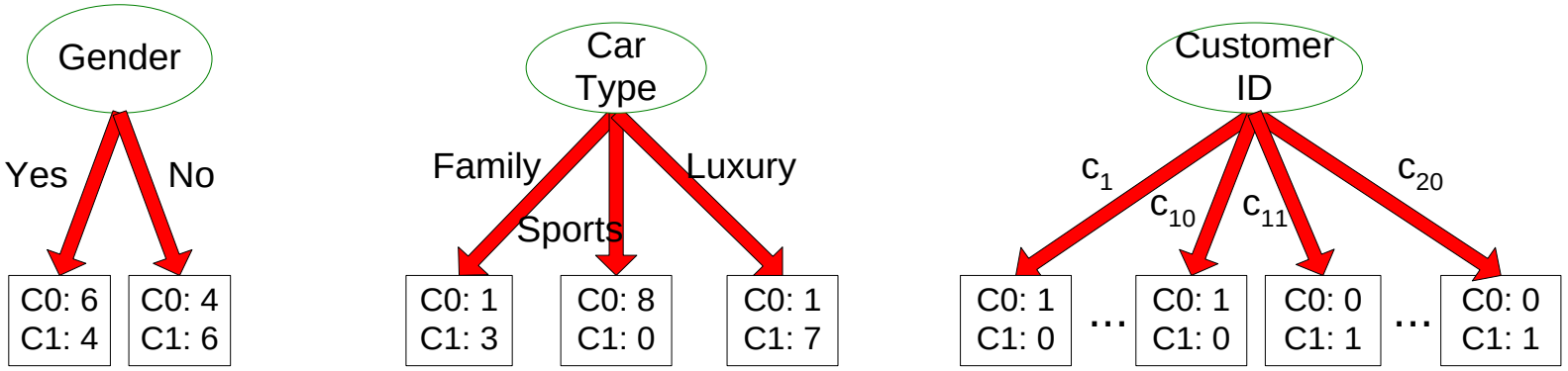
Measures for Selecting an Attribute Test Condition

- There are ***many measures*** that can be used to ***determine the goodness of an attribute test condition.***
  - These measures -- ***give preference to attribute test*** conditions that partition the training instances into ***purer* subsets** in the child nodes, which mostly have the same class labels.
  - ***purer nodes is useful*** --since a node that has all of its training instances from the same class *does not need to be expanded further*.
  - In contrast, an **impure node** containing training instances from multiple classes is likely to *require several levels of node expansions*, thereby increasing the depth of the tree considerably.
  - **Larger trees** are less desirable as they are ***more susceptible to model overfitting,*** a condition that may degrade the classification performance on unseen instances.
    - They are also ***difficult to interpret and incur more training and test time*** as compared to smaller trees.

How to determine the Best Split

Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Before Splitting: 10 records of class 0,
10 records of class 1



Which test condition is the best?

How to determine the Best Split

- Greedy approach:
  - Nodes with purer class distribution are preferred
- Need a measure of node impurity:

C0: 5 C1: 5	C0: 9 C1: 1
High degree of impurity	**Low degree of impurity**

Measures of Node Impurity

- Gini Index

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

Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node **t**, and $\boldsymbol{c}$ is the total number of classes

- Entropy

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

- Misclassification error

$$Classification\ error = 1 - \max[p_i(t)]$$

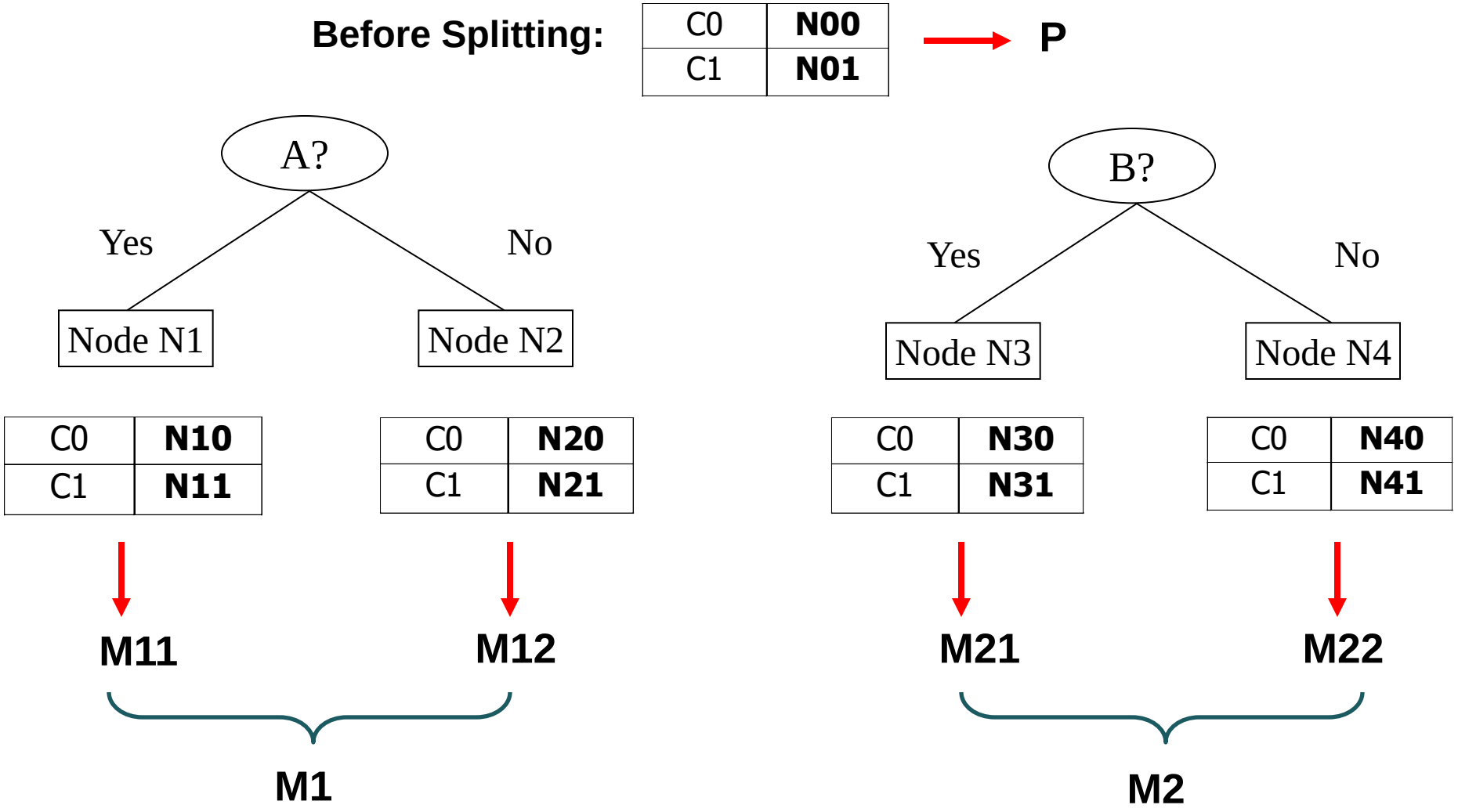
Finding the Best Split

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting
   - Compute impurity measure of each child node
   - M is the weighted impurity of child nodes
3. Choose the attribute test condition that produces the highest gain

Gain = P - M

or equivalently, lowest impurity measure after splitting (M)

Finding the Best Split



Measure of Impurity: GINI

- Gini Index for a given node $\boldsymbol{t}$

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

Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node $\boldsymbol{t}$, and $\boldsymbol{c}$ is the total number of classes

- Maximum of $1 - 1/c$ when records are equally distributed among all classes, implying the least beneficial situation for classification
- Minimum of 0 when all records belong to one class, implying the most beneficial situation for classification
- Gini index is used in decision tree algorithms such as CART, SLIQ, SPRINT

Measure of Impurity: GINI

- Gini Index for a given node t :

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

  - For 2-class problem (p, 1 – p):
    - GINI = $1 - p^2 - (1 - p)^2 = 2p\ (1\text{-}p)$

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Computing Gini Index of a Single Node

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

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – (1/6)² – (5/6)² = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – (2/6)² – (4/6)² = 0.444

Computing Gini Index for a Collection of Nodes

- When a node p is split into k partitions (children)

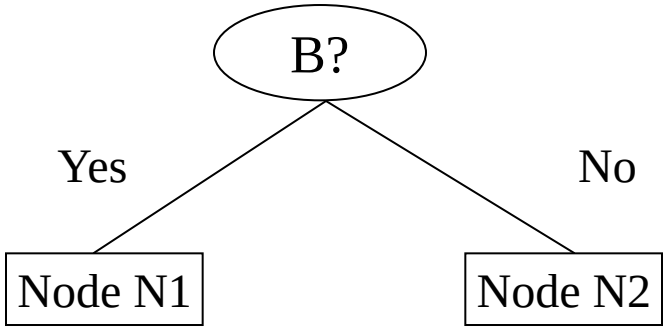
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at parent node p.

Binary Attributes: Computing GINI Index

- Splits into two partitions (child nodes)
- Effect of Weighing partitions:
  - Larger and purer partitions are sought



	Parent
C1	7
C2	5
Gini = 0.486	

Gini(N1)
= 1 – (5/6)² – (1/6)²
= 0.278

Gini(N2)
= 1 – (2/6)² – (4/6)²
= 0.444

	N1	N2
C1	5	2
C2	1	4
Gini=0.361		

Weighted Gini of N1 N2
= 6/12 * 0.278 +
6/12 * 0.444
= 0.361

Gain = 0.486 – 0.361 = 0.125

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

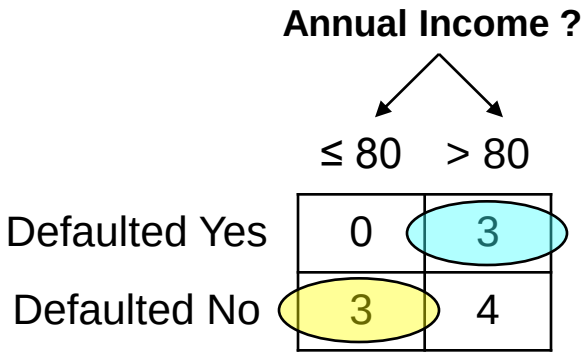
	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

Which of these is the best?

Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, A ≤ v and A > v
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work.

ID	Home Owner	Marital Status	Annual Income	Defaulted
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



Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Annual Income										
Sorted Values →	60	70	75	85	90	95	100	120	125	220

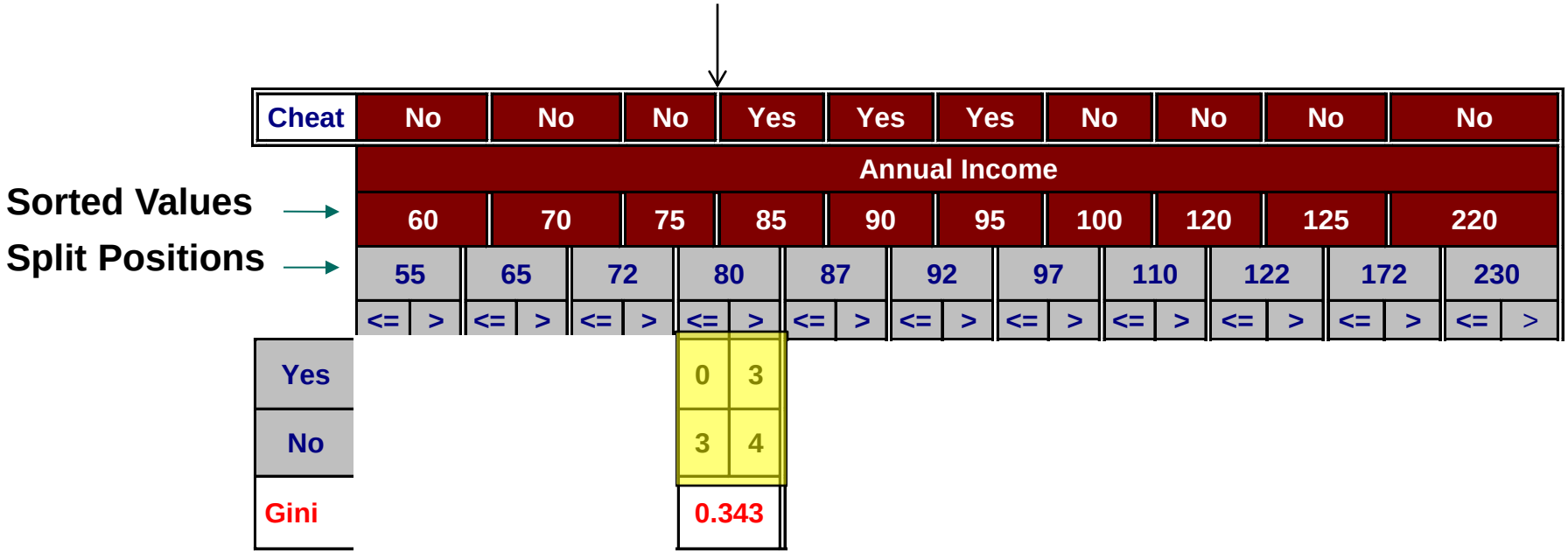
Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Annual Income										
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >

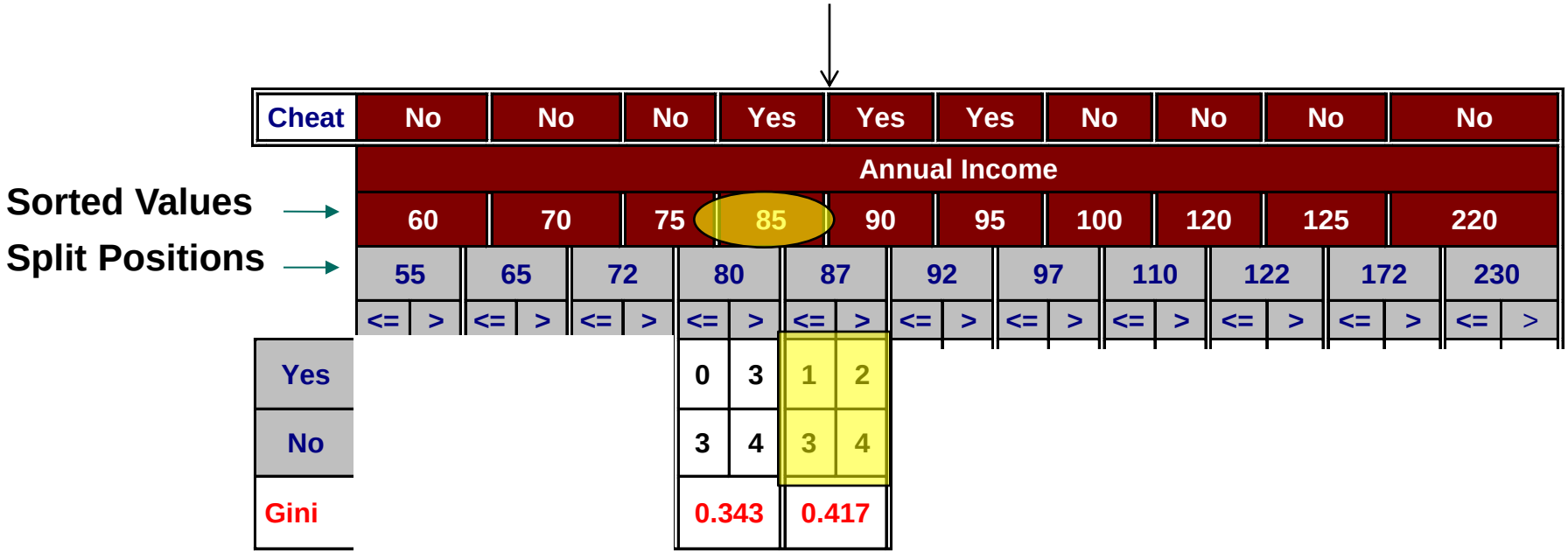
Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index



Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index



Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Annual Income										
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	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

Measure of Impurity: Entropy

- Entropy at a given node $\boldsymbol{t}$

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node $\boldsymbol{t}$, and $\boldsymbol{c}$ is the total number of classes

  - Maximum of $\log_2 c$ when records are equally distributed among all classes, implying the least beneficial situation for classification
  - Minimum of 0 when all records belong to one class, implying most beneficial situation for classification

- Entropy based computations are quite similar to the GINI index computations

Computing Entropy of a Single Node

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Computing Information Gain After Splitting

- Information Gain:

$$Gain_{split} = Entropy(p) - \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)$$

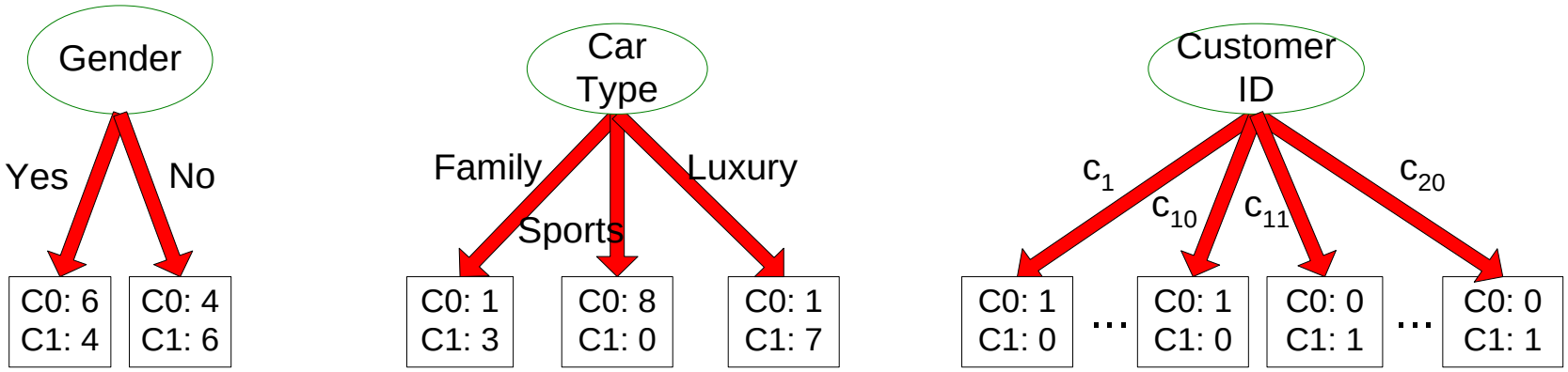
Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

- Choose the split that achieves most reduction (maximizes GAIN)
- Used in the ID3 and C4.5 decision tree algorithms
- Information gain is the mutual information between the class variable and the splitting variable

Problem with large number of partitions

- Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure



  - Customer ID has highest information gain because entropy for all the children is zero

Gain Ratio

- Gain Ratio:

$$Gain\ Ratio = \frac{Gain_{split}}{Split\ Info} \qquad Split\ Info = -\sum_{i=1}^{k} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

- Adjusts Information Gain by the entropy of the partitioning ($Split\ Info$).
  - Higher entropy partitioning (large number of small partitions) is penalized!
- Used in C4.5 algorithm
- Designed to overcome the disadvantage of Information Gain

Gain Ratio

- Gain Ratio:

$$Gain\ Ratio = \frac{Gain_{split}}{Split\ Info} \qquad Split\ Info = \sum_{i=1}^{k} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

SplitINFO = 1.52

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

SplitINFO = 0.72

	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

SplitINFO = 0.97

Measure of Impurity: Classification Error

- Classification error at a node t

$$Error(t) = 1 - \max_i [p_i(t)]$$

  - Maximum of $1 - 1/c$ when records are equally distributed among all classes, implying the least interesting situation
  - Minimum of 0 when all records belong to one class, implying the most interesting situation

Computing Error of a Single Node

$$Error(t) = 1 - \max_i [p_i(t)]$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

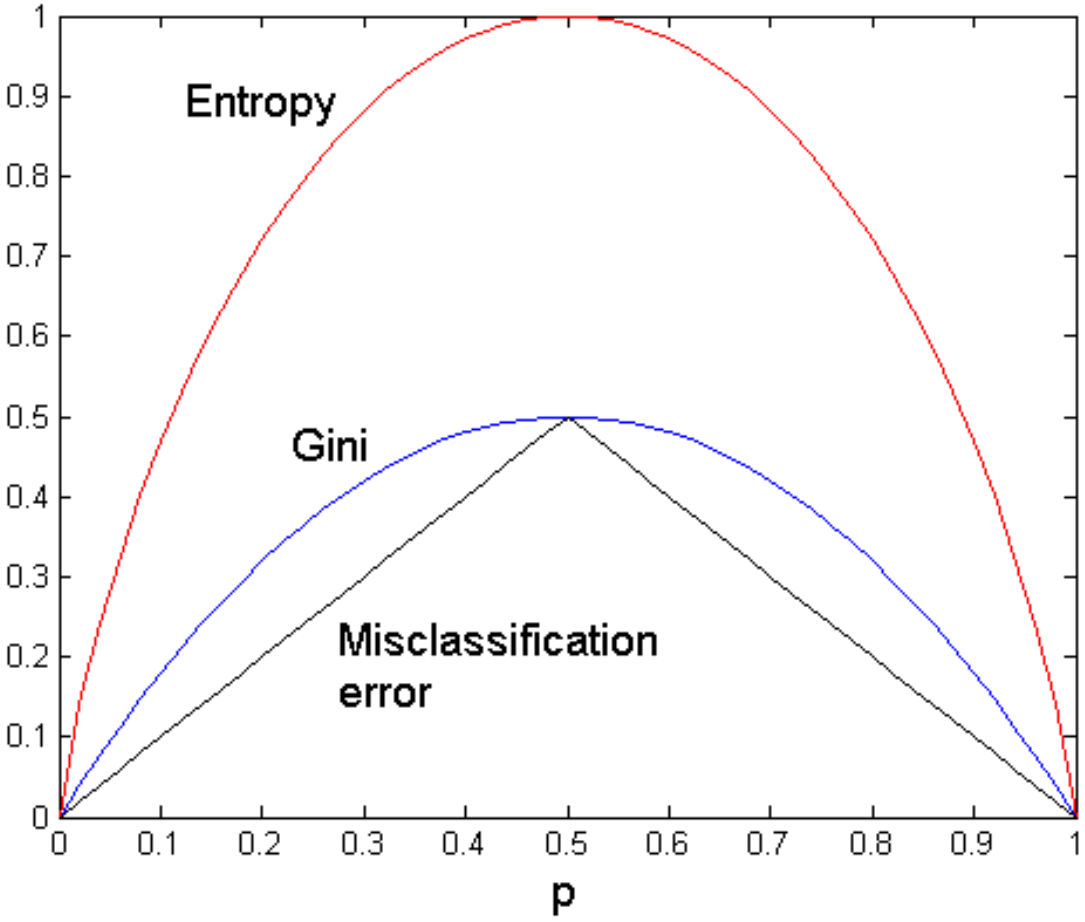
C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

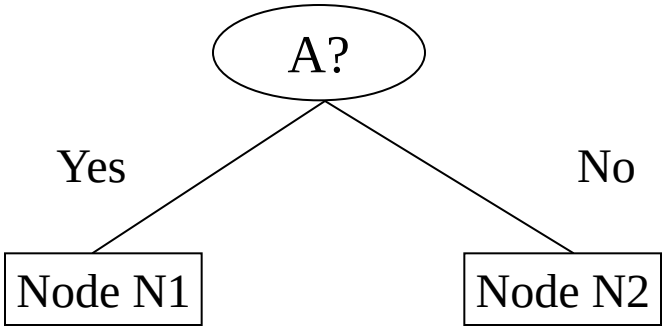
Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison among Impurity Measures

For a 2-class problem:



Misclassification Error vs Gini Index



	Parent
C1	7
C2	3
Gini = 0.42	

Gini(N1)
= 1 – (3/3)² – (0/3)²
= 0

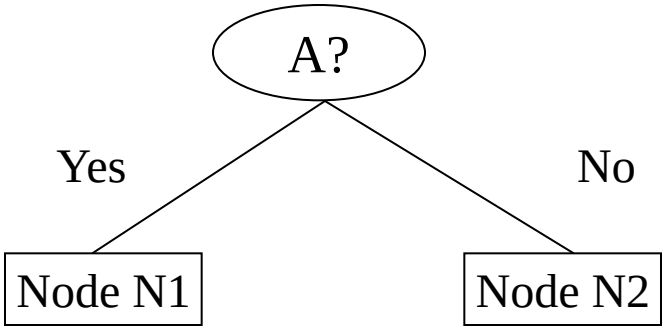
Gini(N2)
= 1 – (4/7)² – (3/7)²
= 0.489

	N1	N2
C1	3	4
C2	0	3
Gini=0.342		

Gini(Children)
= 3/10 * 0
+ 7/10 * 0.489
= 0.342

Gini improves but error remains the same!!

Misclassification Error vs Gini Index



	Parent
C1	7
C2	3
Gini = 0.42	

	N1	N2
C1	3	4
C2	0	3
Gini=0.342		

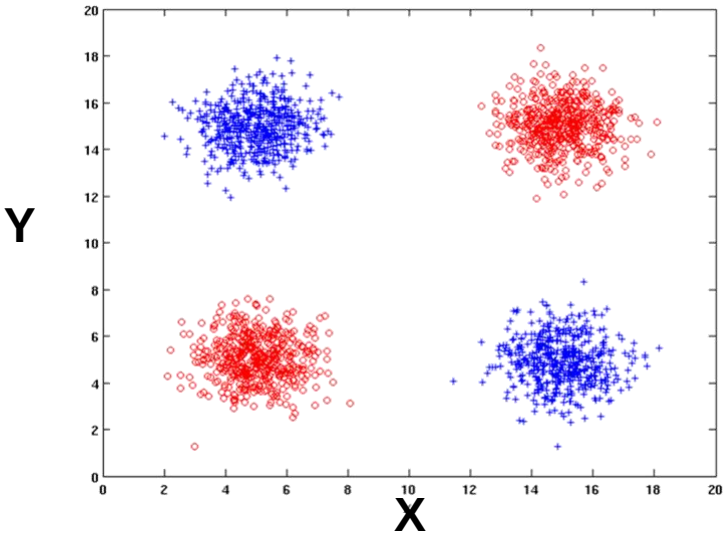
	N1	N2
C1	3	4
C2	1	2
Gini=0.416		

Misclassification error for all three cases = 0.3 !

Decision Tree Based Classification

- Advantages:
  - Relatively inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Robust to noise (especially when methods to avoid overfitting are employed)
  - Can easily handle redundant attributes
  - Can easily handle irrelevant attributes (unless the attributes are interacting)
- Disadvantages: .
  - Due to the greedy nature of splitting criterion, interacting attributes (that can distinguish between classes together but not individually) may be passed over in favor of other attributed that are less discriminating.
  - Each decision boundary involves only a single attribute

Handling interactions



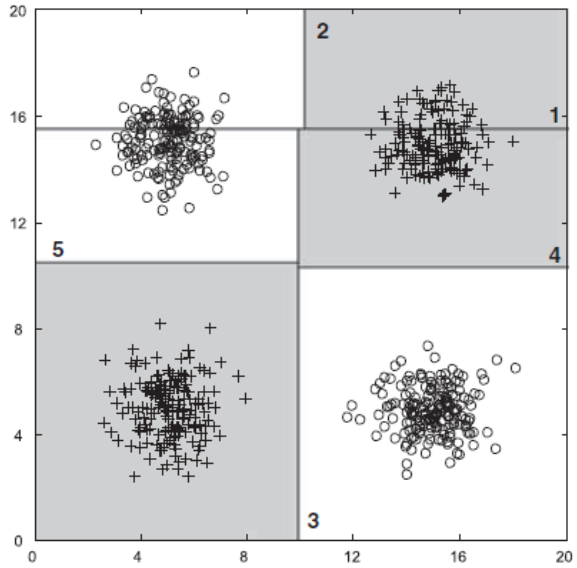
+ : 1000 instances

o : 1000 instances

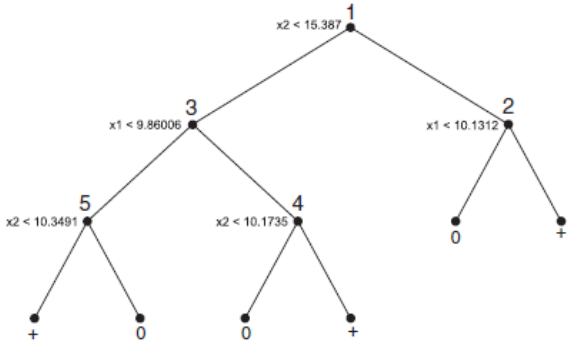
Entropy (X) : 0.99

Entropy (Y) : 0.99

Handling interactions



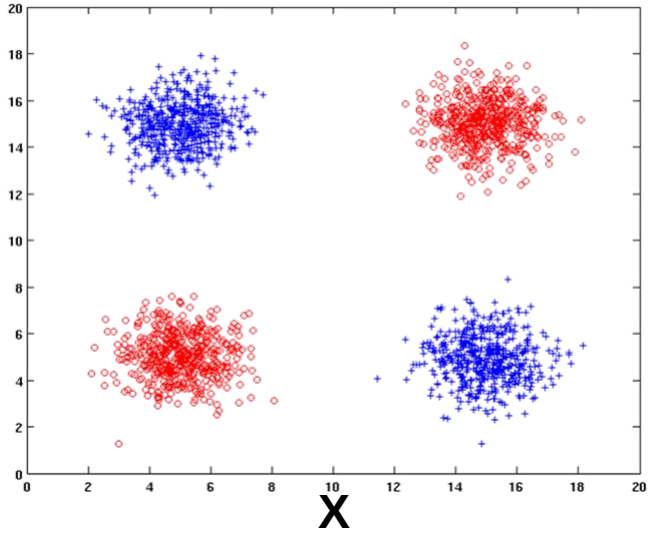
(a) Decision boundary for tree with 6 leaf nodes.



(b) Decision tree with 6 leaf nodes.

Figure 3.28. Decision tree with 6 leaf nodes using X and Y as attributes. Splits have been numbered from 1 to 5 in order of other occurrence in the tree.

Handling interactions given irrelevant attributes



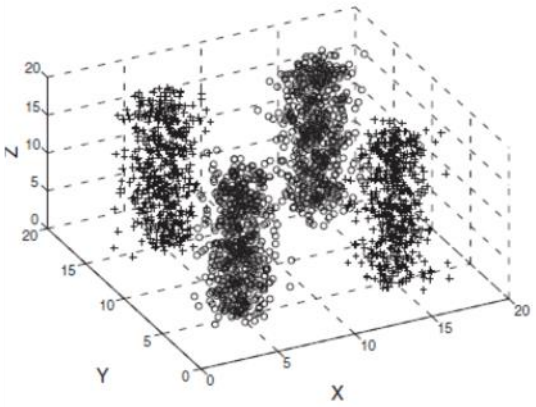
+ : 1000 instances

o : 1000 instances

Adding Z as a noisy attribute generated from a uniform distribution

Entropy (X) : 0.99
Entropy (Y) : 0.99
Entropy (Z) : 0.98

Attribute Z will be chosen for splitting!



(a) Three-dimensional data with attributes X, Y, and Z.

Algorithm for Decision Tree Induction

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|v is a possible outcome of root.test_cond }.
 9:    for each v ∈ V do
10:       E_v = {e | root.test_cond(e) = v and e ∈ E}.
11:       child = TreeGrowth(E_v, F).
12:       add child as descendent of root and label the edge (root → child) as v.
13:    end for
14: end if
15: return root.
```

Algorithm for Decision Tree Induction

i) The **createNode()** function extends the decision tree by creating a new node. A node in the decision tree either has a test condition, denoted as node.test_cond, or a class label, denoted as node.label.

(ii) The **find_best_split()** function determines the attribute test condition for partitioning the training instances associated with a node. The splitting attribute chosen depends on the impurity measure used.

(iii) The **Classify() function** determines the class label to be assigned to a leaf node. For each leaf node t, let p(i|t) denote the fraction of training instances from class i associated with the node t. The label assigned to the leaf node is typically the one that occurs most frequently in the training instances that are associated with this node.

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

where the argmax operator returns the class i that maximizes p(i|t).

(iv) The stopping **cond() function** is used to terminate the tree-growing process by checking whether all the instances have identical class label or attribute values

Characteristics of Decision Tree Classifiers

Important characteristics of decision tree induction algorithms.

1. **Applicability**
2. **Expressiveness**
3. **Computational Efficiency**
4. **Handling Missing Values**
5. **Handling Interactions among Attributes**
6. **Handling Irrelevant Attributes**
7. **Handling Redundant Attributes**
8. **Using Rectilinear Splits**
9. **Choice of Impurity Measure**

UNIT-3: Model Overfitting

Reasons for Model Overfitting

Classification Errors

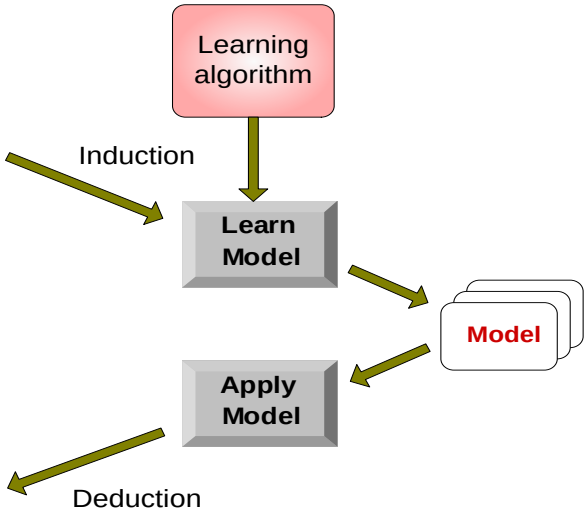
- **Training errors**: Errors committed on the training set
- **Test errors**: Errors committed on the test set
- **Generalization errors**: Expected error of a model over random selection of records from same distribution

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

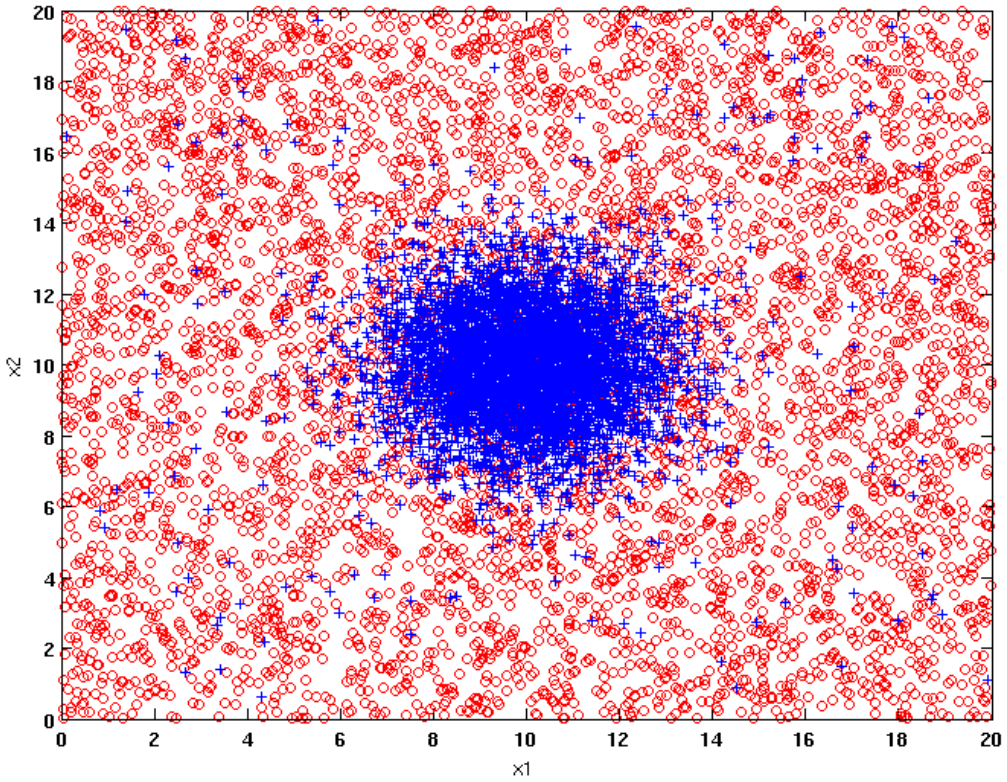
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Example Data Set

Examples of training and test sets of a two-dimensional classification problem.



Two class problem:

o : 5400 instances

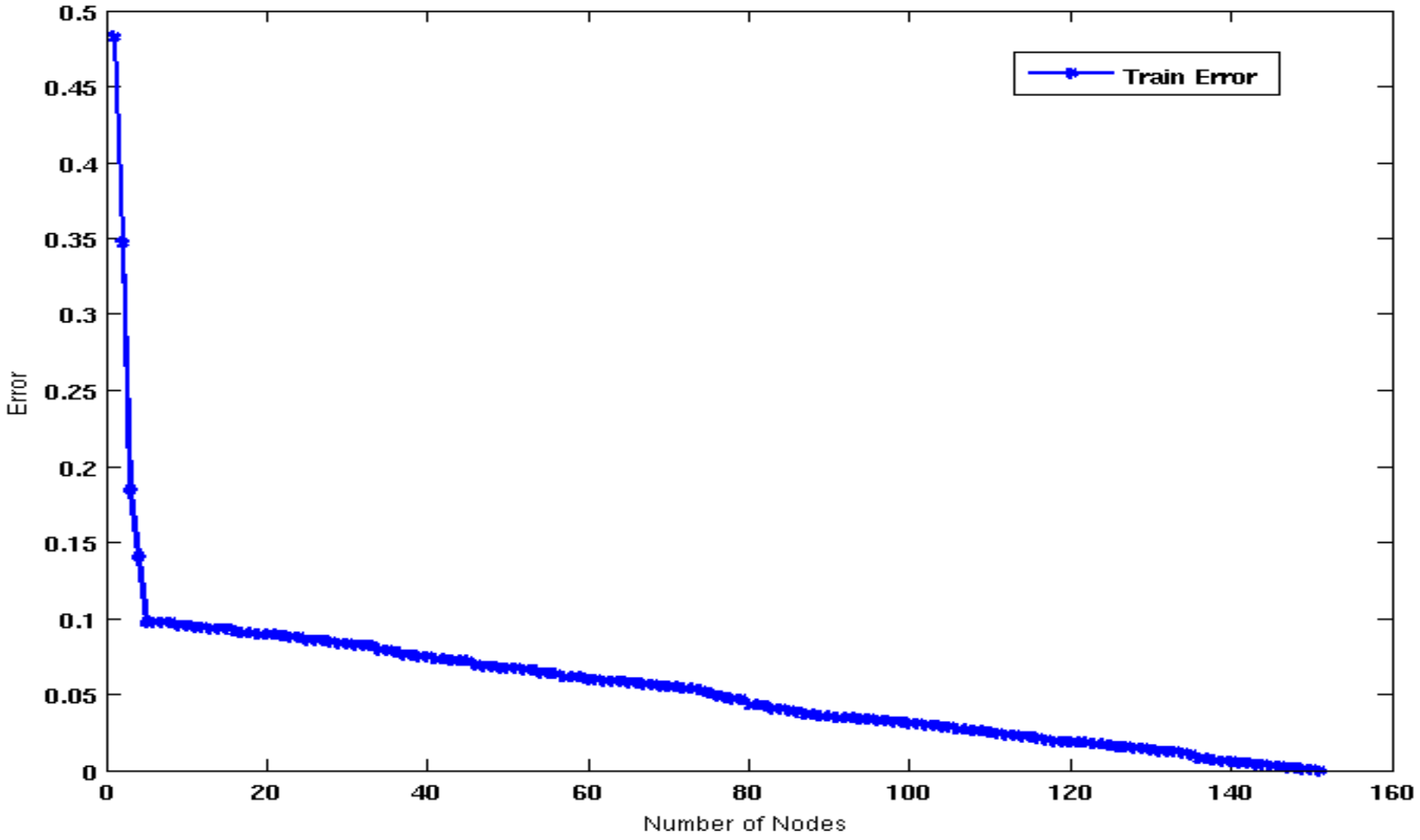
- **Generated from a uniform distribution**

+ : 5400 instances

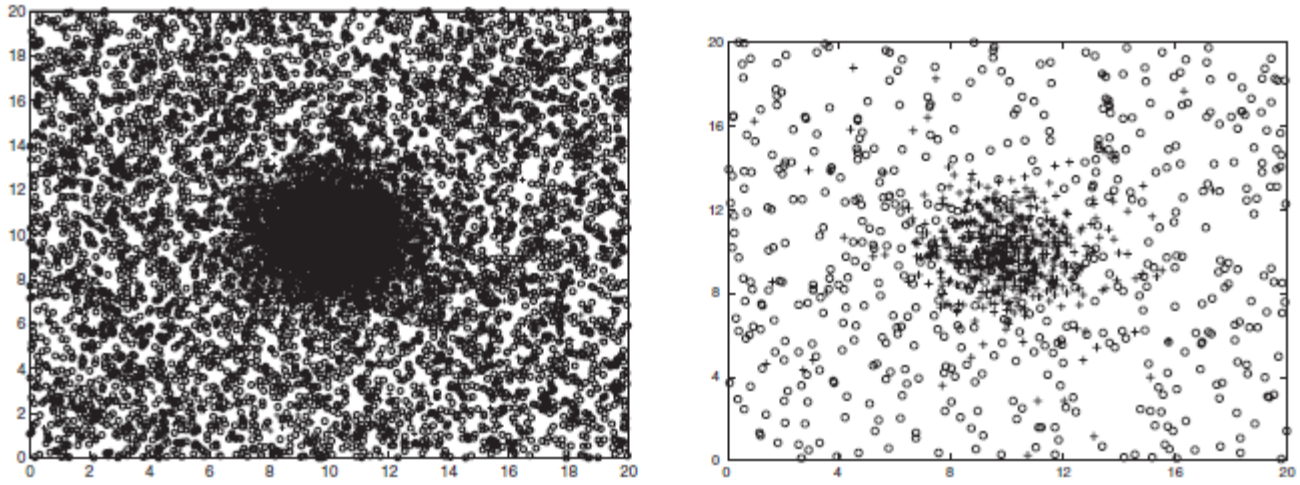
- **5000 instances generated from a Gaussian distribution centered at (10,10)**
- **400 noisy instances sampled from Uniform distribution**

10 % of the data used for training and 90% of the data used for testing

Increasing number of nodes in Decision Trees



- Even if a model fits well over the training data, it can still show poor generalization performance, a phenomenon known as **model overfitting.**

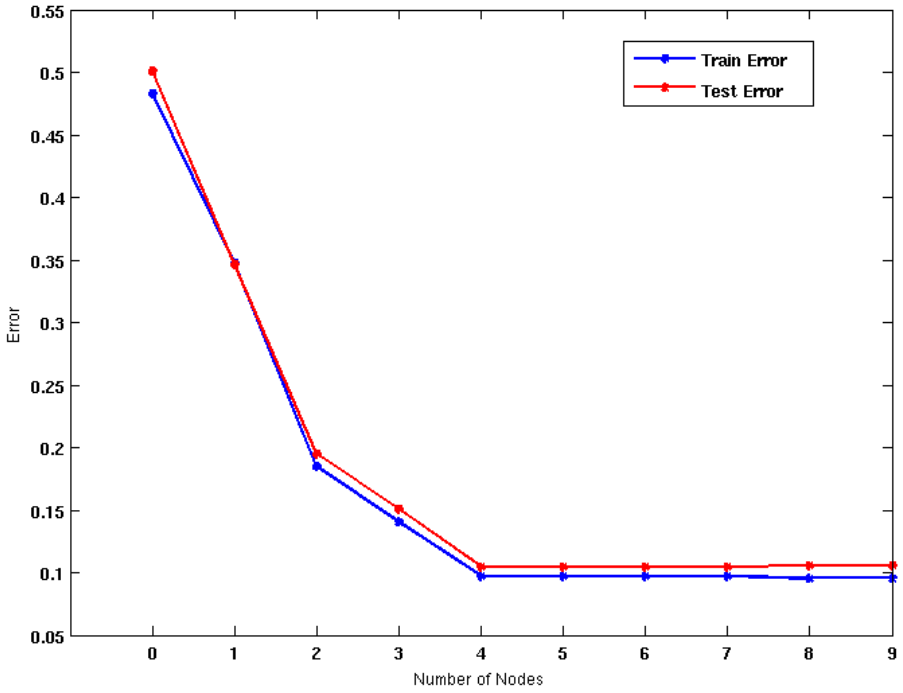


(a) Example of a 2-D data.

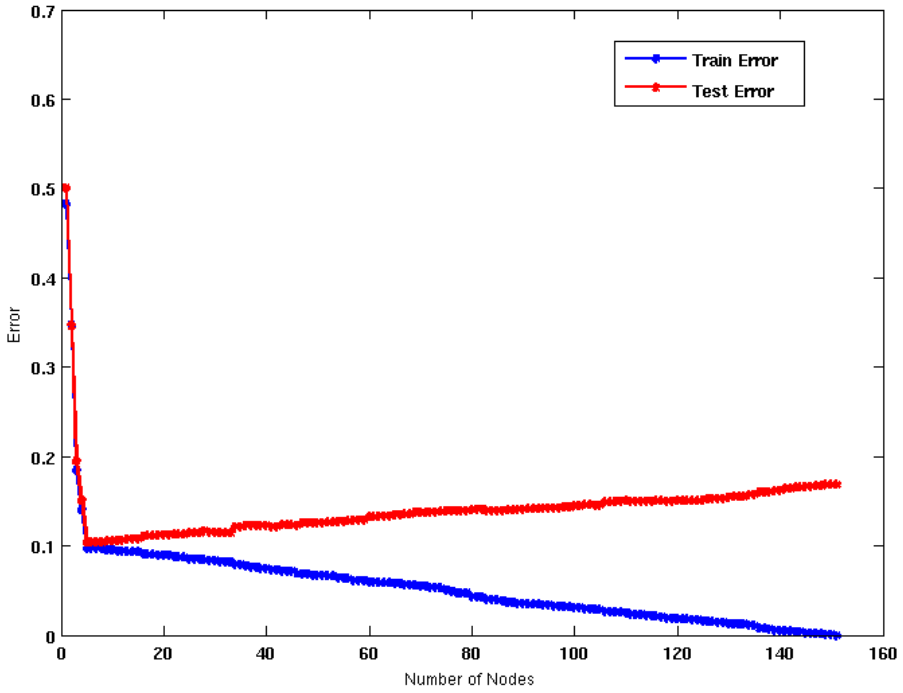
(b) Training set using 10% data.

Figure 3.22. Examples of training and test sets of a two-dimensional classification problem.

Model Underfitting and Overfitting



(a) Varying tree size from 1 to 8.



(b) Varying tree size from 1 to 150.

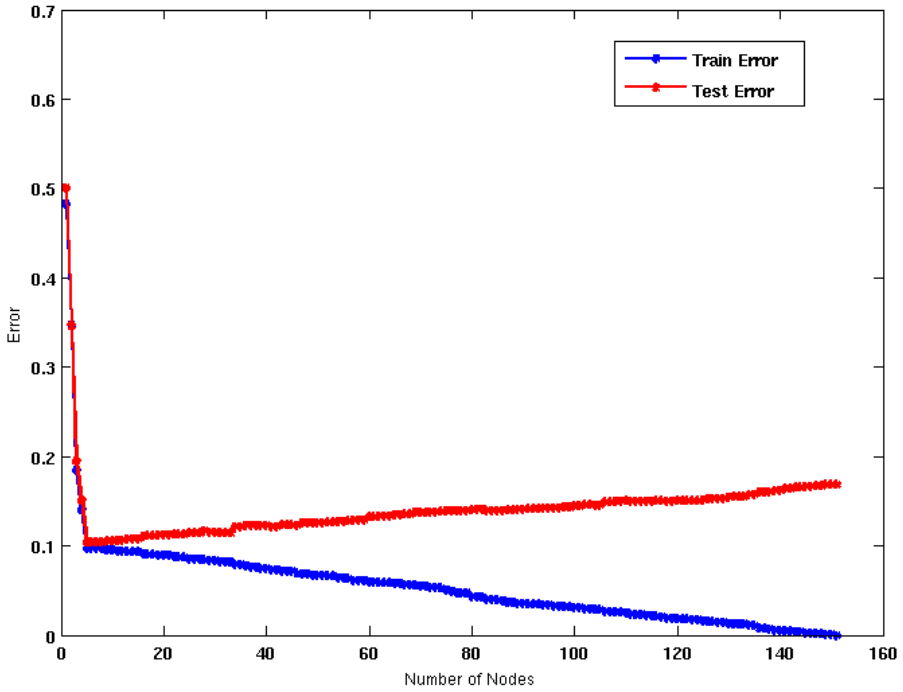
Effect of varying tree size (number of leaf nodes) on training and test errors.

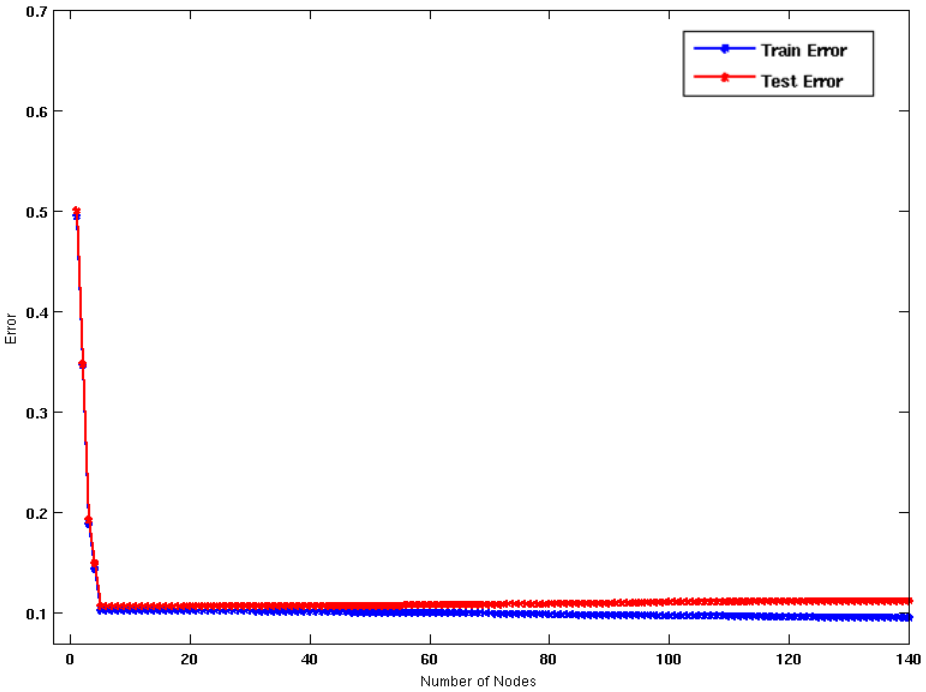
- As the model becomes more and more complex, test errors can start increasing even though training error may be decreasing

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

Overfitting: when model is too complex, training error is small but test error is large

Model Overfitting – Impact of Training Data Size





Using twice the number of data instances

- Increasing the size of training data reduces the difference between training and testing errors at a given size of model

Overfitting & Underfitting

- Figure 3.23(a) shows changes in the training and test error rates as the size of the tree varies from 1 to 8.
- Both error rates are initially large when the tree has only one or two leaf nodes. This situation is known as **model underfitting**.
- **Underfitting** occurs when the learned decision tree is too simplistic, and thus, incapable of fully representing the true relationship between the attributes and the class labels.
- As we **increase the tree size from 1 to 8, we can observe two effects**:
  - First, both the error rates decrease since larger trees are able to represent more complex decision boundaries.
  - Second, the training and test error rates are quite close to each other, which indicates that the performance on the training set is fairly representative of the generalization performance.

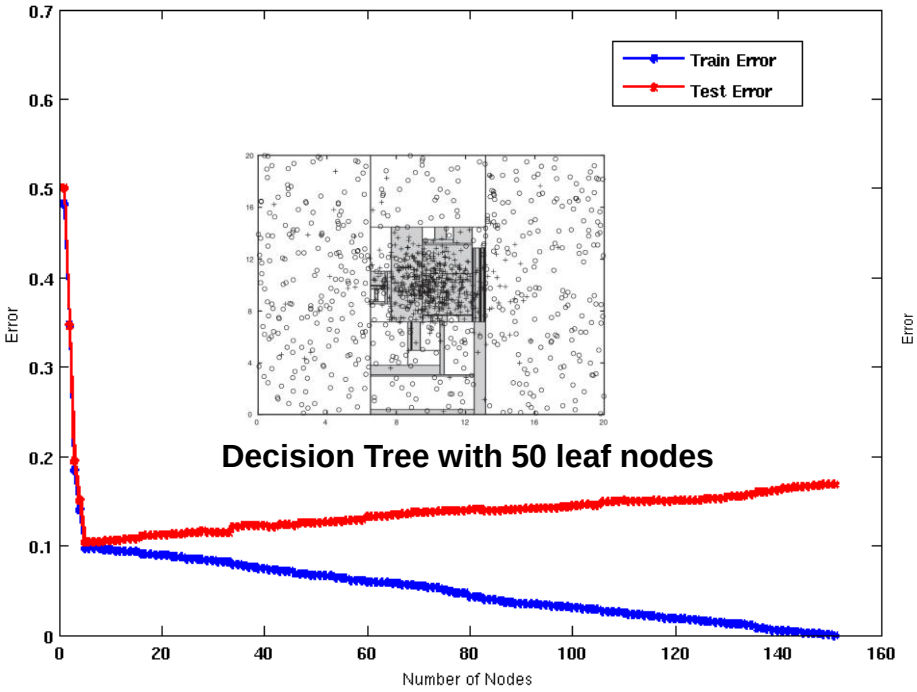
Overfitting & Underfitting

- As we further increase the size of the tree from 8 to 150, the training error continues to steadily decrease till it eventually reaches zero, as shown in Figure 3.23(b).
- However, in a striking contrast, the test error rate ceases to decrease any further beyond a certain tree size, and then it begins to increase.
- The training error rate thus grossly under-estimates the test error rate once the tree becomes too large.
- Further, the gap between the training and test error rates keeps on widening as we increase the tree size. This behavior, which may seem counter-intuitive at first, can be attributed to the phenomena of **model overfitting**.

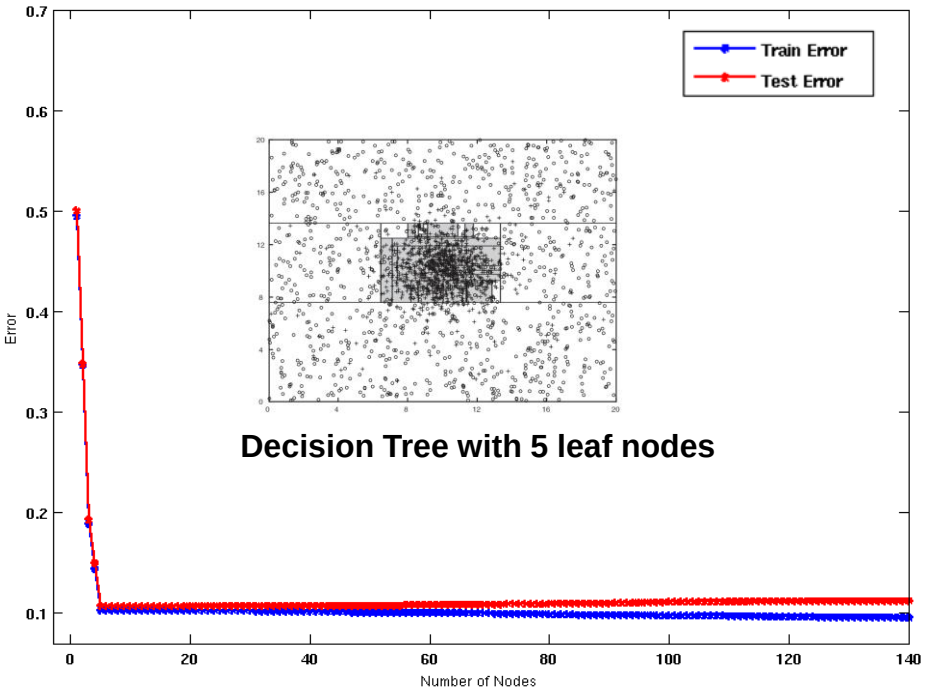
Reasons for Model Overfitting

- **Model overfitting** is the phenomena where, in the pursuit of minimizing the training error rate, 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.
- To illustrate this, Figure 3.24 shows decision trees and their corresponding decision boundaries (shaded rectangles represent regions assigned to the + class) for two trees of sizes 5 and 50.

Model Overfitting – Impact of Training Data Size



Decision Tree with 50 leaf nodes

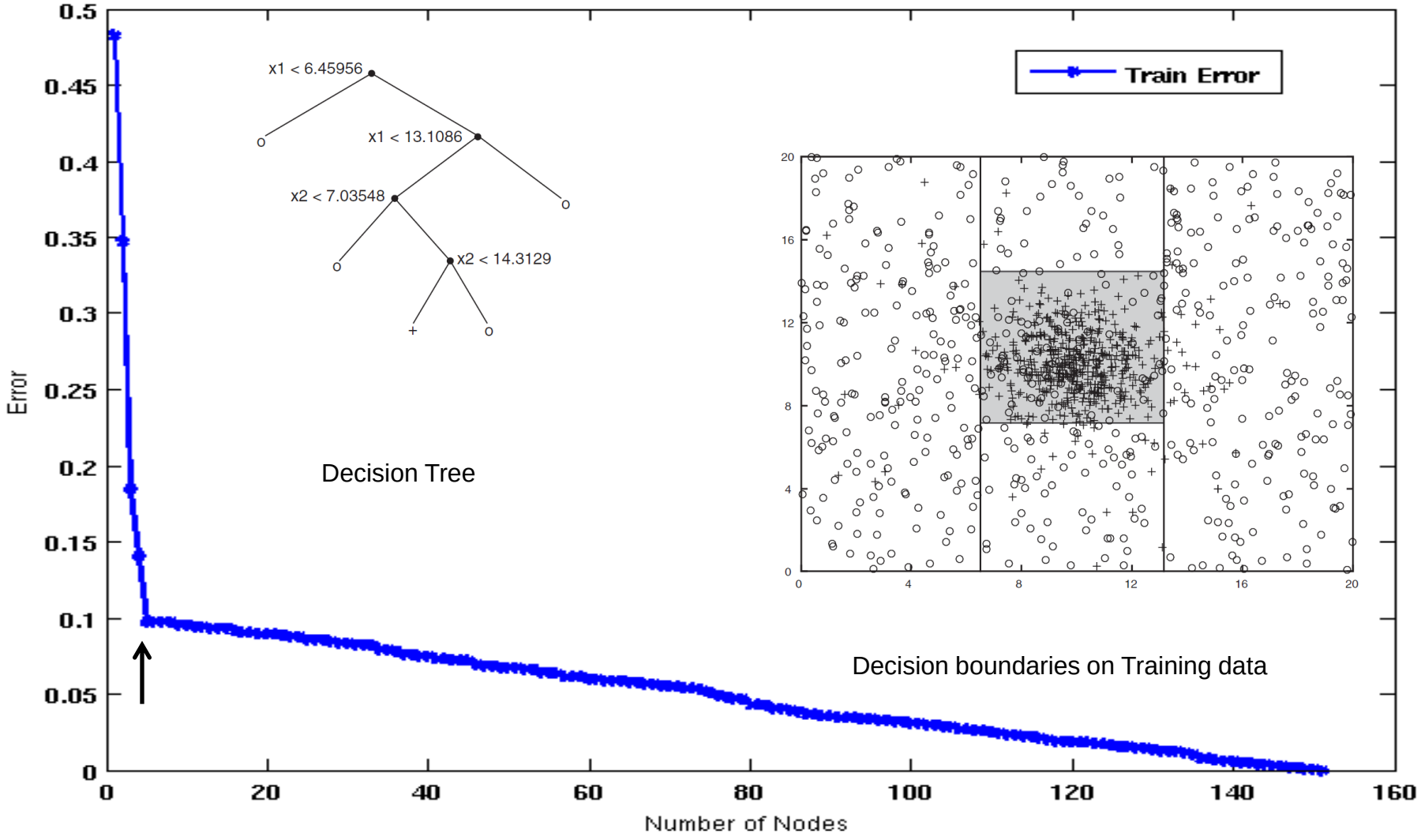


Decision Tree with 5 leaf nodes

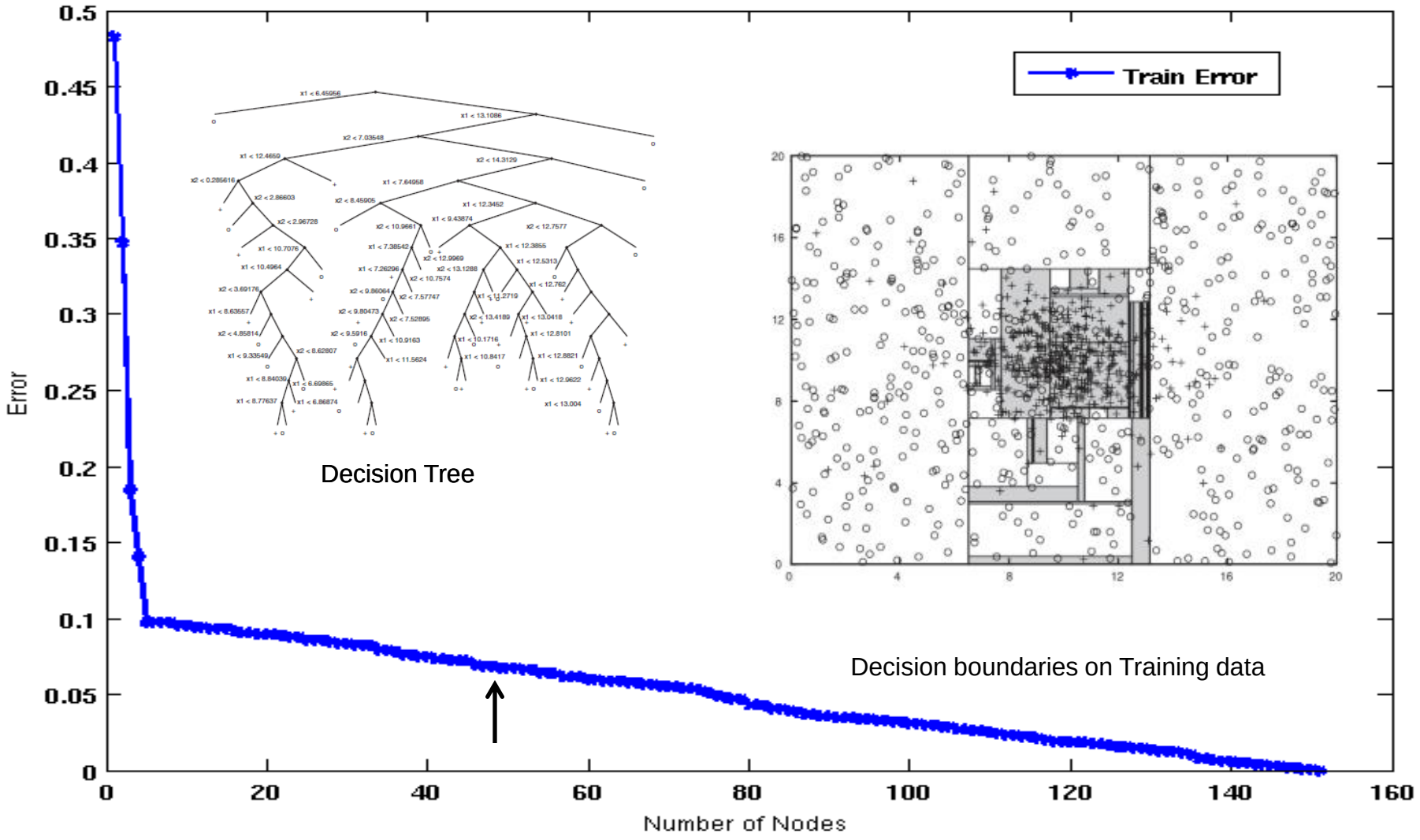
Using twice the number of data instances

- Increasing the size of training data reduces the difference between training and testing errors at a given size of model

Decision Tree with 4 nodes



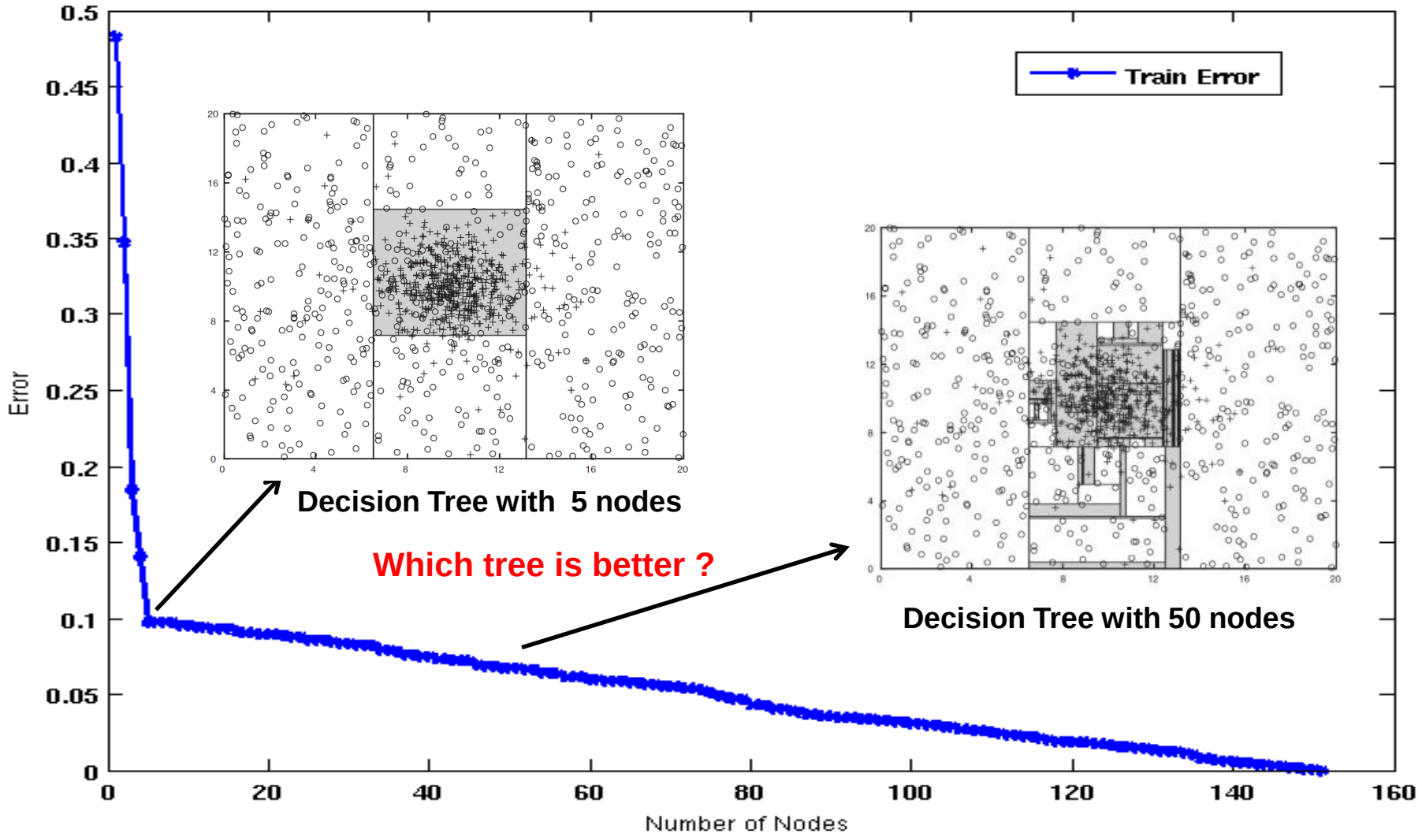
Decision Tree with 50 nodes



Decision Tree

Decision boundaries on Training data

Which tree is better?



Reasons for Model Overfitting

- Limited Training Size:
  - Not enough training data
  - the effect of overfitting can be reduced by increasing the training size
- High model complexity
  - Multiple Comparison Procedure

Reasons for Model Overfitting

High model complexity

- A more complex model has a better ability to represent complex patterns in the data. For example, decision trees with larger number of leaf nodes can represent more complex decision boundaries than decision trees with fewer leaf nodes.
- However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalize well over unseen instances. Models with high complexity should thus be ***judiciously used to avoid overfitting.***
- ***One measure of model complexity is the number of "parameters***" that need to be inferred from the training set.
- For example, in the case of decision tree induction, the attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set. A decision tree with larger number of attribute test conditions (and consequently more leaf nodes) thus involves more "parameters" and hence is more complex.

Effect of Multiple Comparison Procedure

- Consider the task of predicting whether stock market will rise/fall in the next 10 trading days

- Random guessing:

$$P(correct) = 0.5$$

- Make 10 random guesses in a row:

$$P(\#correct \geq 8) = \frac{\binom{10}{8} + \binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0547$$

Day 1	Up
Day 2	Down
Day 3	Down
Day 4	Up
Day 5	Down
Day 6	Down
Day 7	Up
Day 8	Up
Day 9	Up
Day 10	Down

Effect of Multiple Comparison Procedure

- Approach:
  - Get 50 analysts
  - Each analyst makes 10 random guesses
  - Choose the analyst that makes the most number of correct predictions

- Probability that at least one analyst makes at least 8 correct predictions

$$P(\#\,correct \geq 8) = 1 - (1 - 0.0547)^{50} = 0.9399$$

Effect of Multiple Comparison Procedure

- Many algorithms employ the following greedy strategy:
  - Initial model: M
  - Alternative model: M' = M $\cup$ γ, where γ is a component to be added to the model (e.g., a test condition of a decision tree)
  - Keep M' if improvement, Δ(M,M') > α
- Often times, γ is chosen from a set of alternative components, $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$
- If many alternatives are available, one may inadvertently add irrelevant components to the model, resulting in model overfitting

Effect of Multiple Comparison - Example

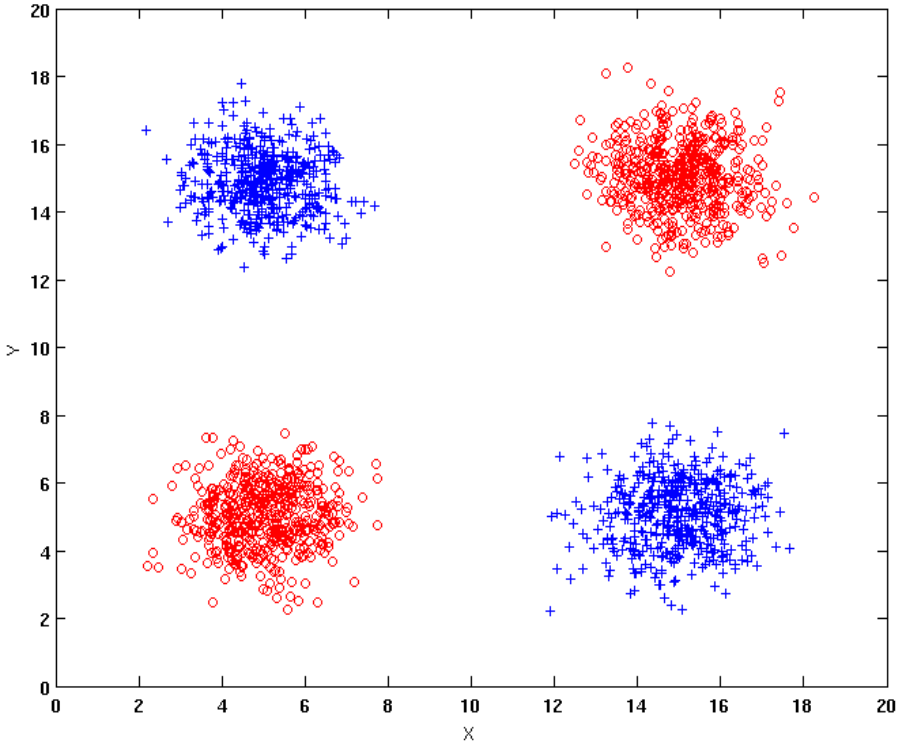
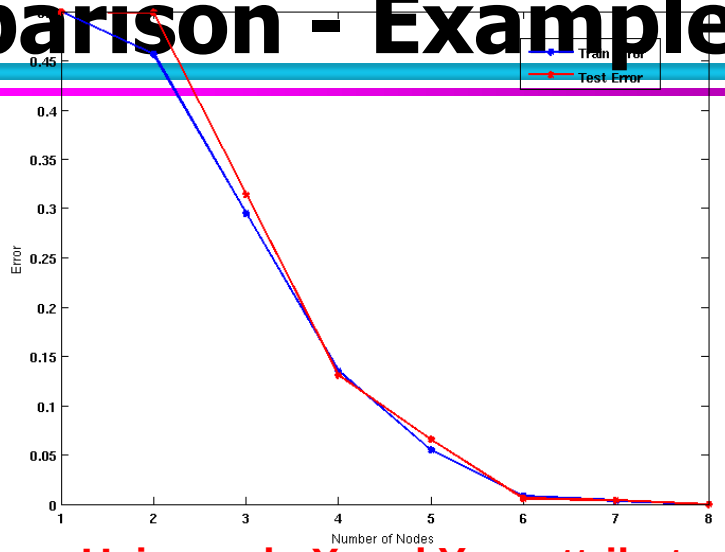


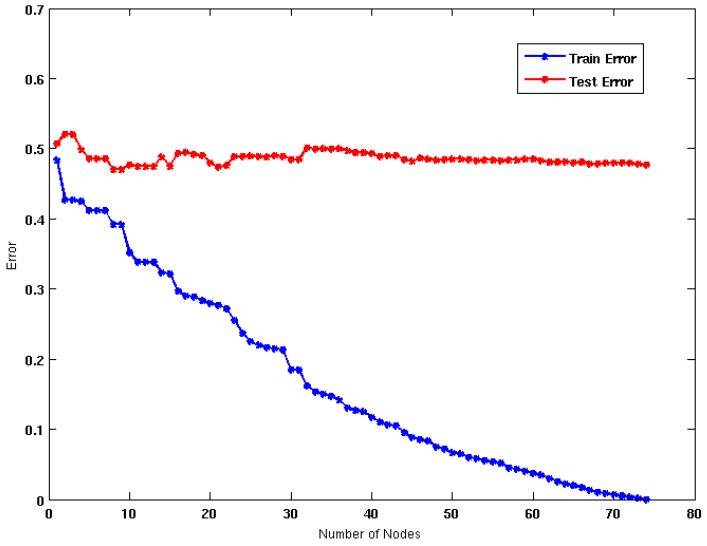
Figure 3.26. Example of a two-dimensional (X-Y) data set.

Use additional 100 noisy variables generated from a uniform distribution along with X and Y as attributes.

Use 30% of the data for training and 70% of the data for testing



Using only X and Y as attributes



b) After adding 100 irrelevant attributes.

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error does not provide a good estimate of how well the tree will perform on previously unseen records
- Need ways for estimating generalization errors