Data Mining
Classification: Basic Concepts and Techniques

Lecture Notes for Chapter 3

Introduction to Data Mining, 2nd Edition

by

Tan, Steinbach, Karpatne, Kumar

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

Classification: Definition

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

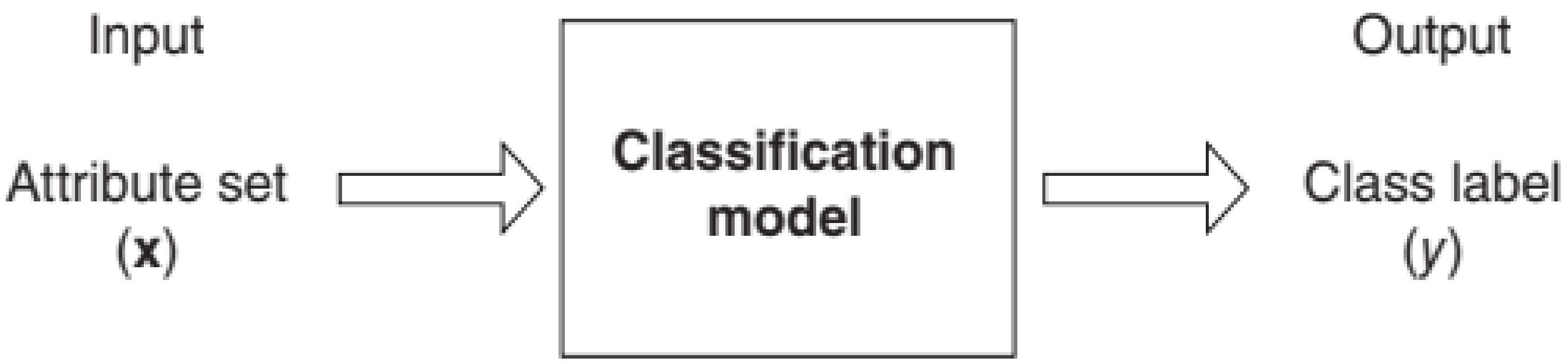
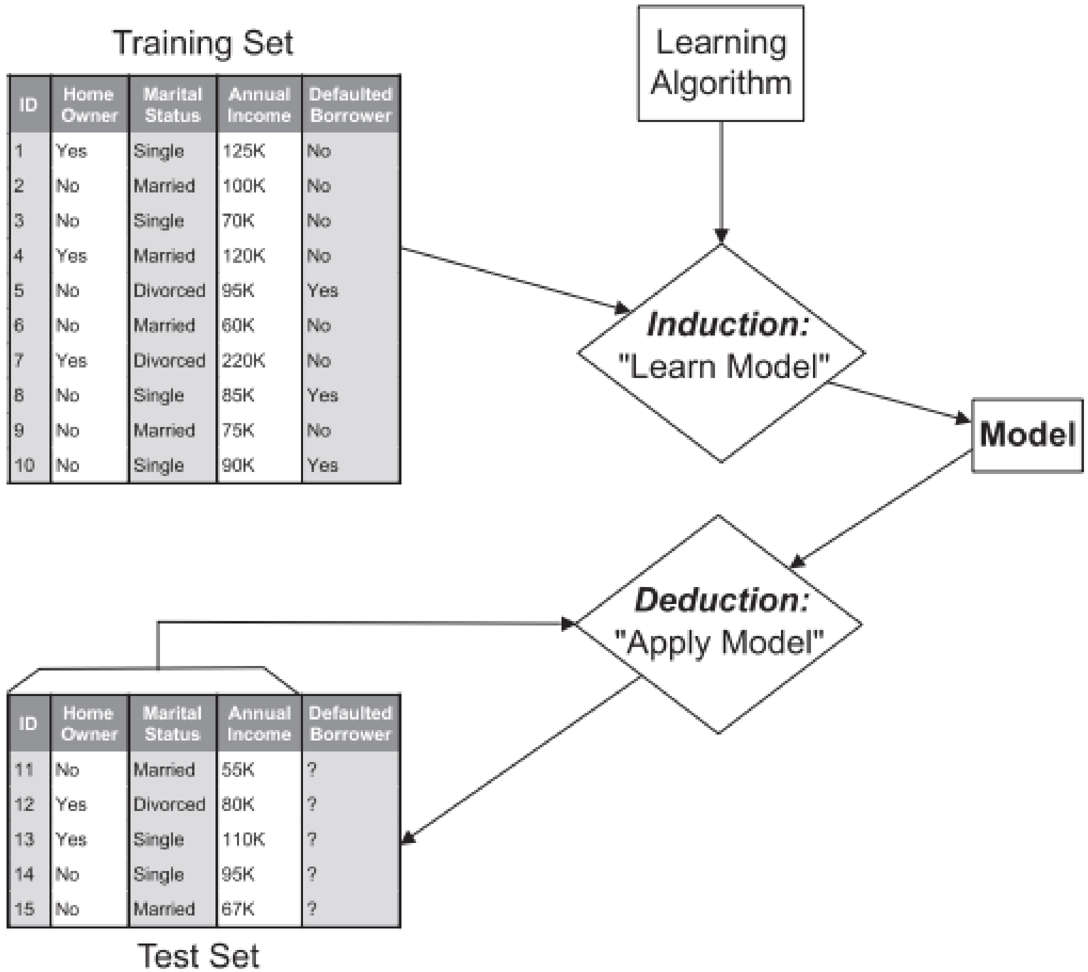


Figure 3.2. A schematic illustration of a classification task.

General Approach for Building Classification Model



Training Set

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

Test Set

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.

Classification: Definition

- A classification model serves two important roles in data mining
  - It is used as a **predictive model** to classify previously unlabeled instances.
    - A good classification model must provide accurate predictions with a fast response time.
  - It serves as a **descriptive model** to identify the characteristics that distinguish instances from different classes.
    - This is particularly useful for critical applications, such as medical diagnosis, where it is insufficient to have a model that makes a prediction without justifying how it reaches such a decision.

Classification Techniques

- Base Classifiers
  - Decision Tree based Methods
  - Rule-based Methods
  - Nearest-neighbor
  - Naïve Bayes
  - Support Vector Machines
  - etc.

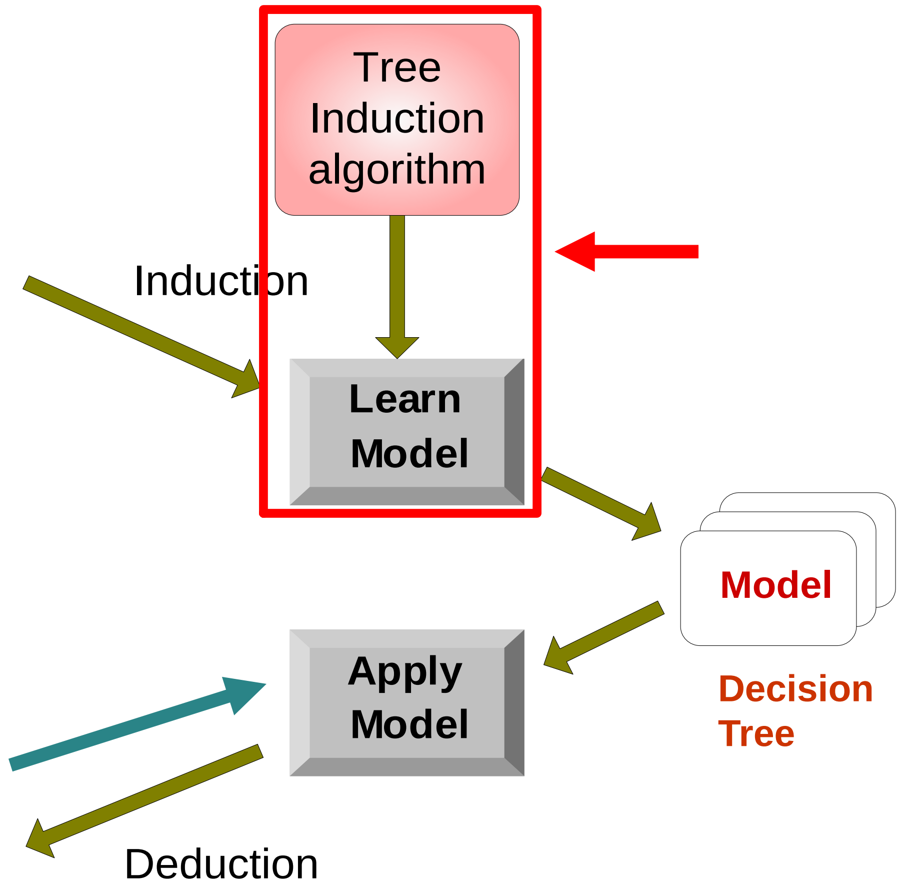
Decision Tree Classification Task

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

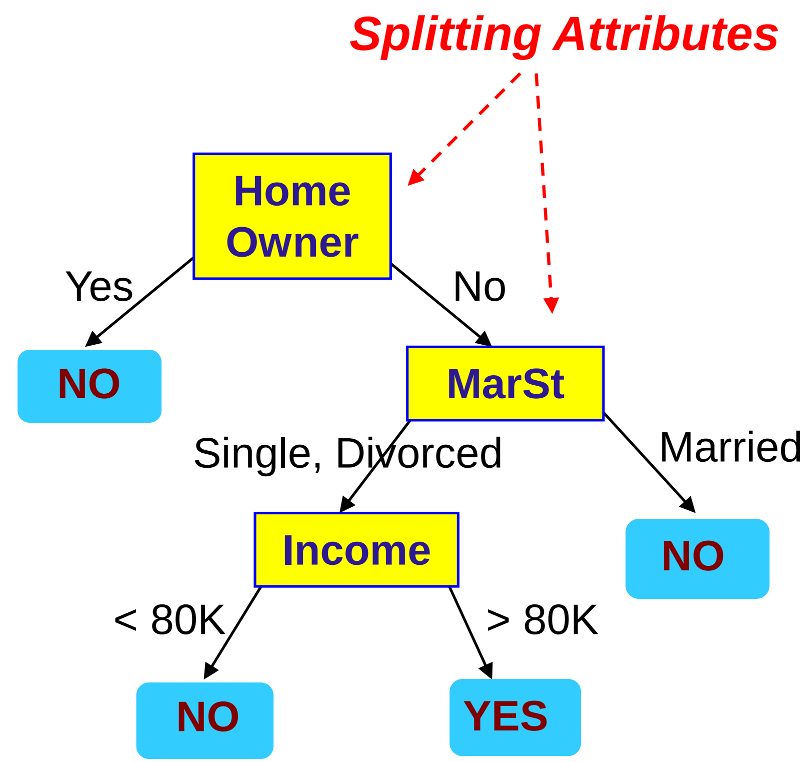


Example of a Decision Tree

categorical categorical continuous class

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

Training Data

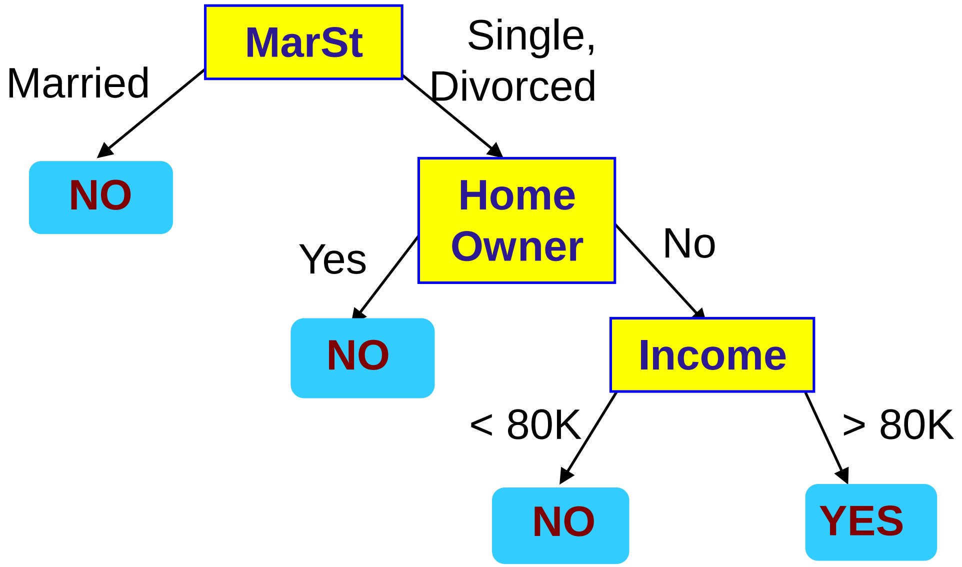


Model: Decision Tree

Another Example of Decision Tree

categorical categorical continuous class

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

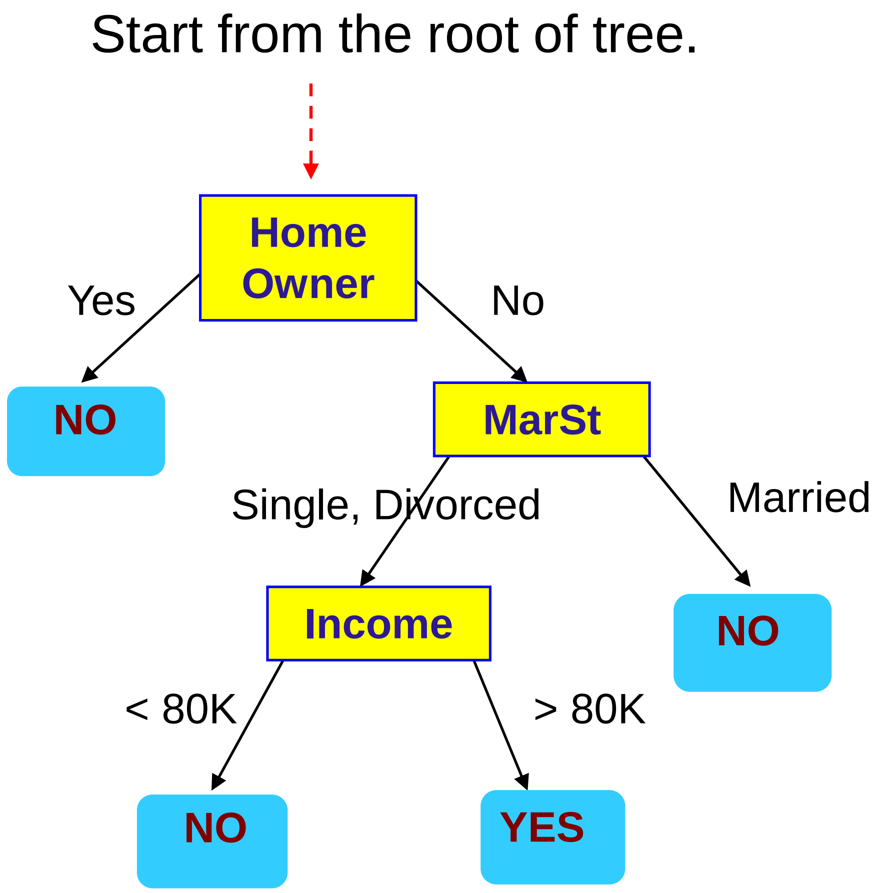


There could be more than one tree that fits the same data!

Important

- Many possible decision trees can be constructed from a particular data set
- While some trees are better than others, finding an optimal one is computationally expensive due to the exponential size of the search space
- Efficient algorithms have been developed to induce a reasonably accurate, albeit suboptimal, decision tree in a reasonable amount of time
- These algorithms usually employ a greedy strategy to grow the decision tree in a top-down fashion by making a series of locally optimal decisions about which attribute to use when partitioning the training data

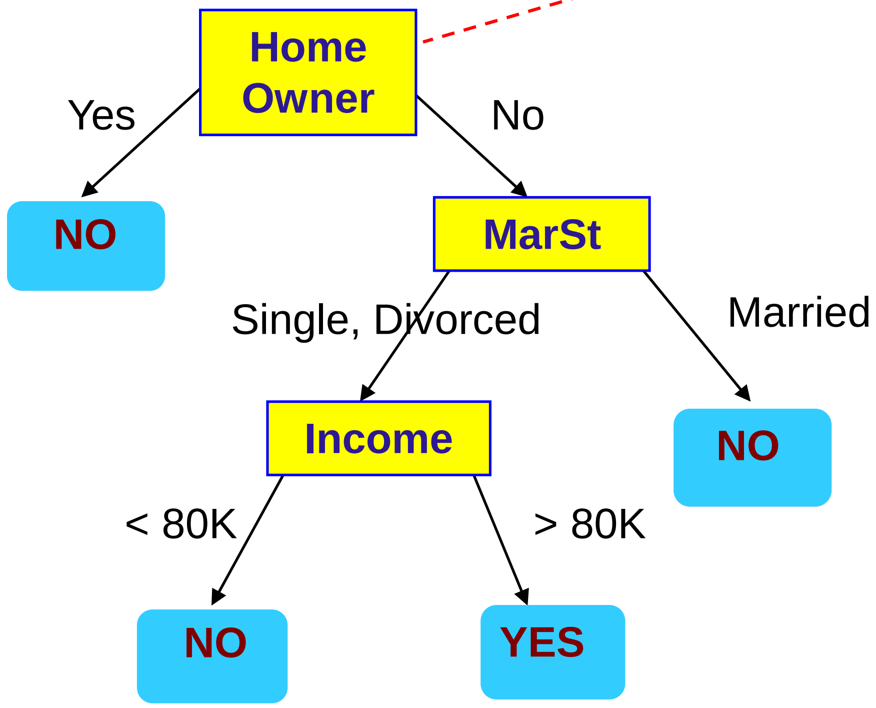
Apply Model to Test Data



Test Data

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

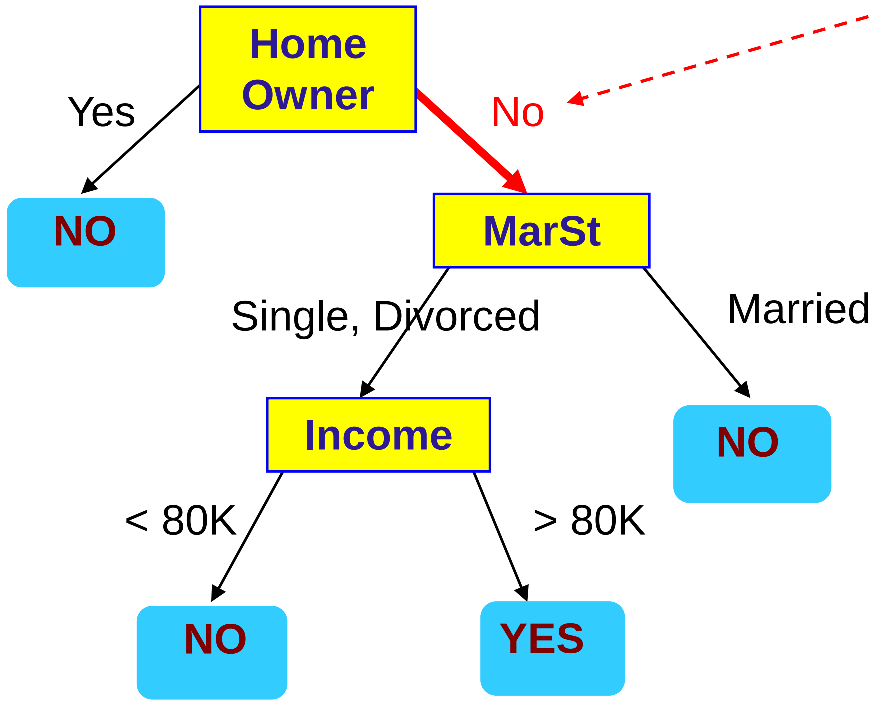
Apply Model to Test Data



Test Data

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

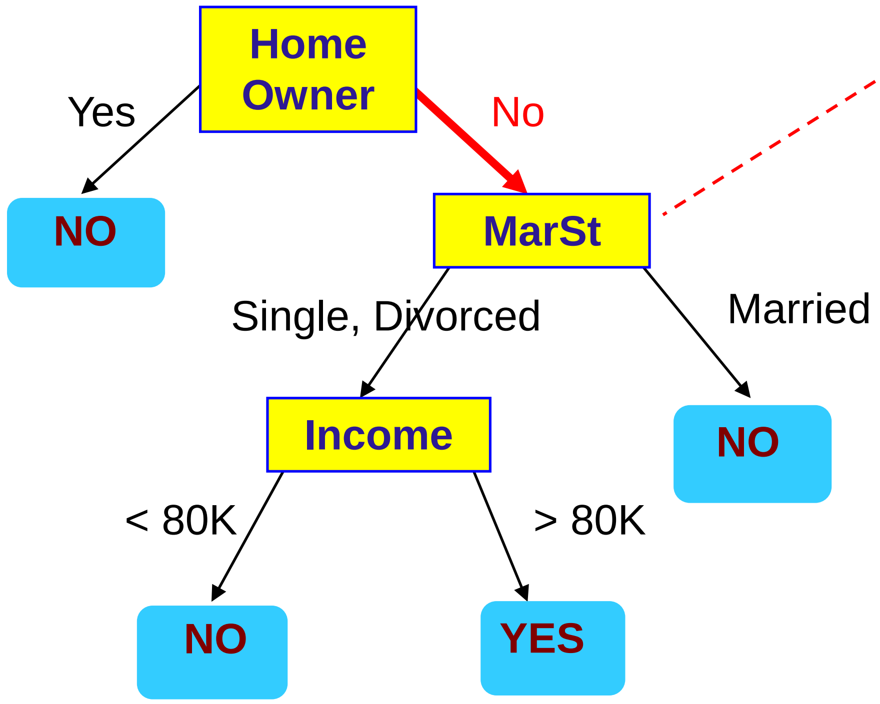
Apply Model to Test Data



Test Data

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

Apply Model to Test Data



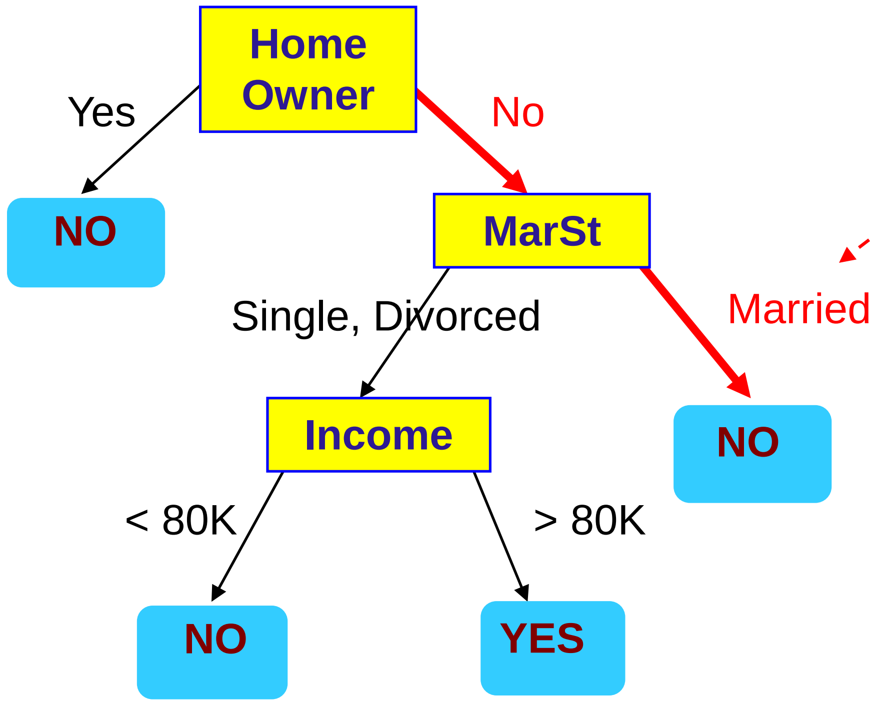
Test Data

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

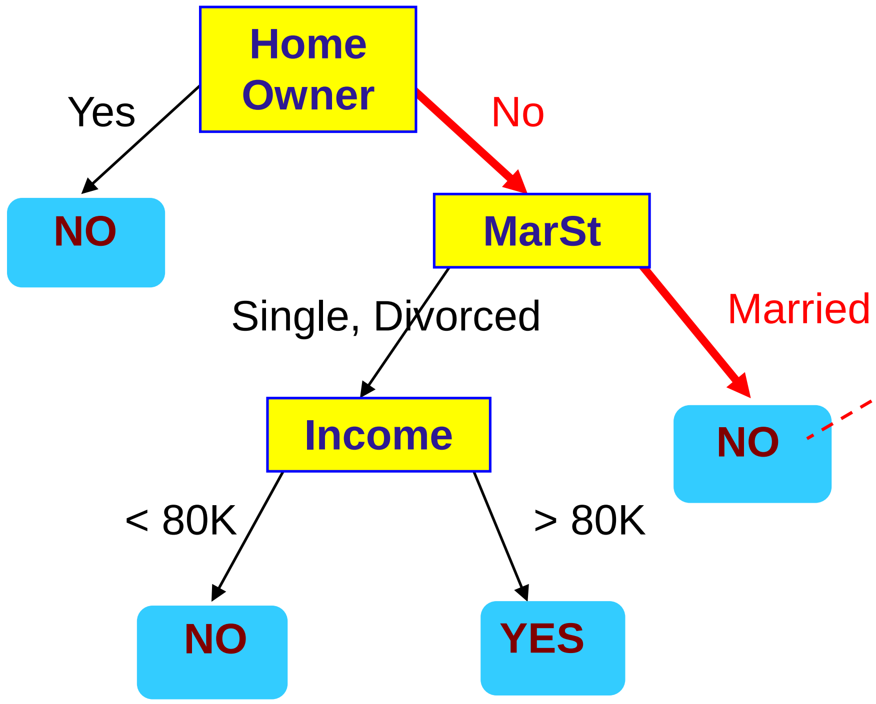
Apply Model to Test Data

Test Data

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



Apply Model to Test Data



Test Data

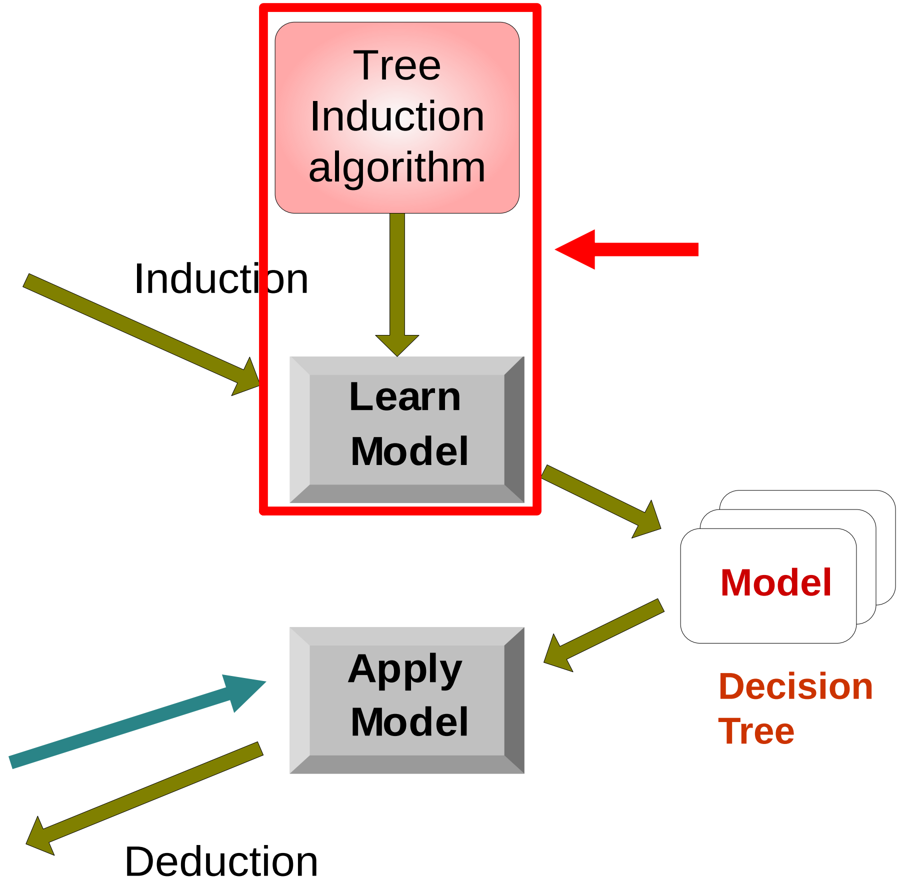
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Assign Defaulted to "No"

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 Induction

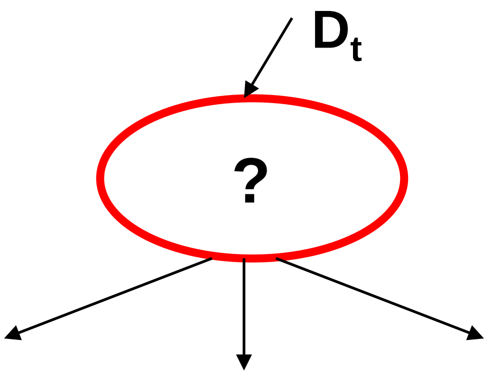
- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
    - The basis for many current implementations of decision tree classifiers
      - It is possible to see some of the design issues that must be considered when building a decision tree
      - It is a generic procedure for growing decision trees in a greedy fashion
  - CART
  - ID3, C4.5

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

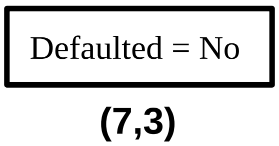
Defaulted = No

(7,3)

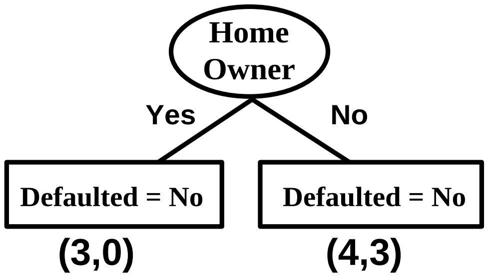
(a)

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



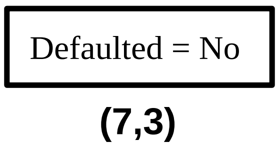
(a)



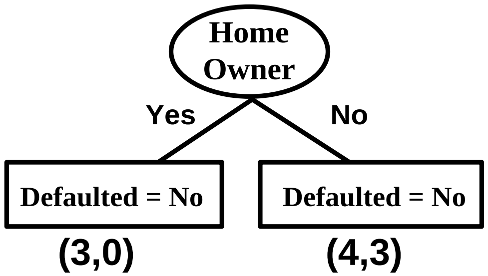
(b)

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

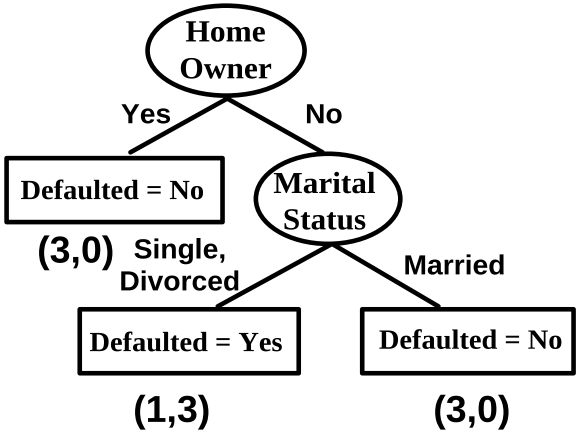
Hunt's Algorithm



(a)



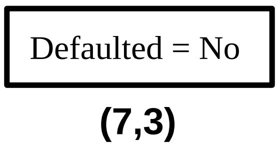
(b)



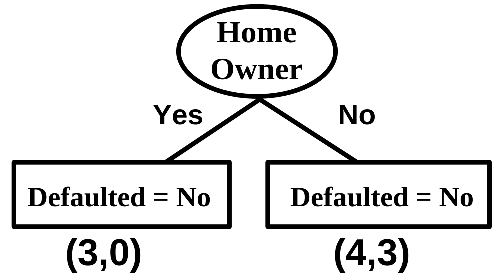
(c)

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

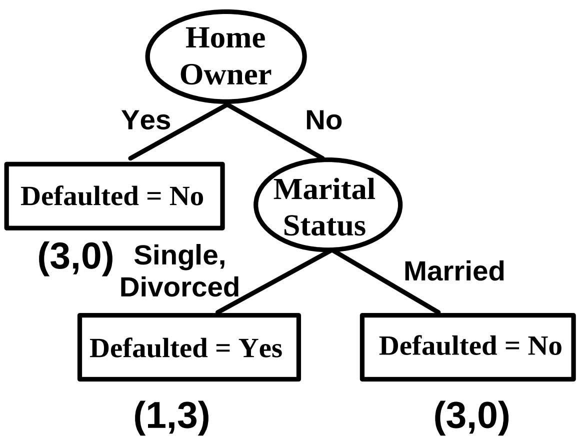


(a)

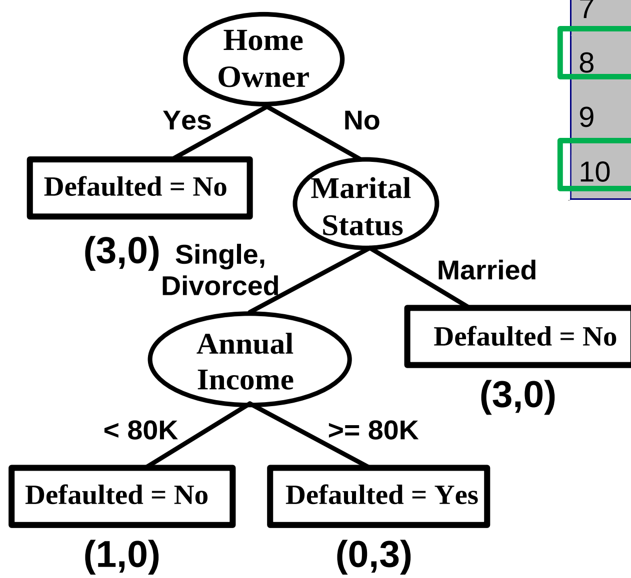


(b)

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)



(d)

Design Issues of Decision Tree Induction

- How should training records be split?
  - Method for specifying 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

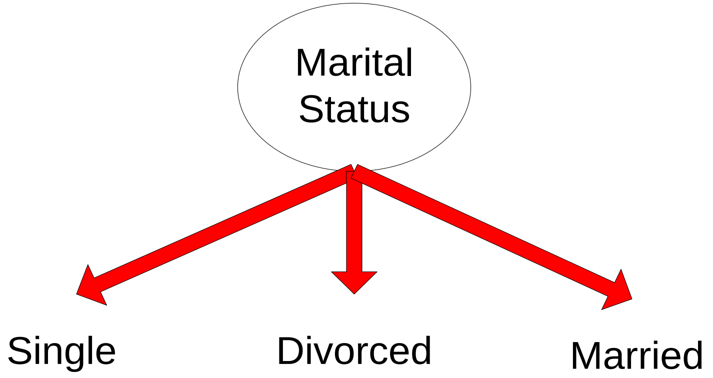
Methods for Expressing Test Conditions

- Depends on attribute types
  - Binary
  - Nominal
  - Ordinal
  - Continuous

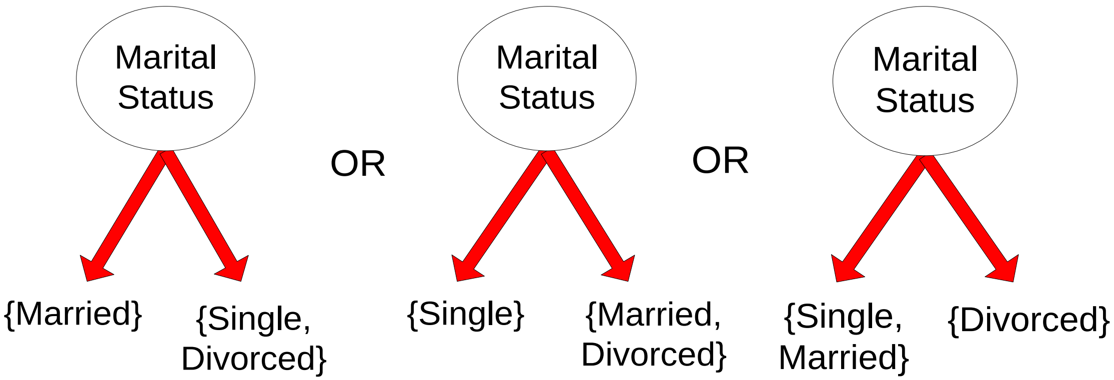
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Test Condition for Nominal Attributes

- Multi-way split:
  - Use as many partitions as distinct values

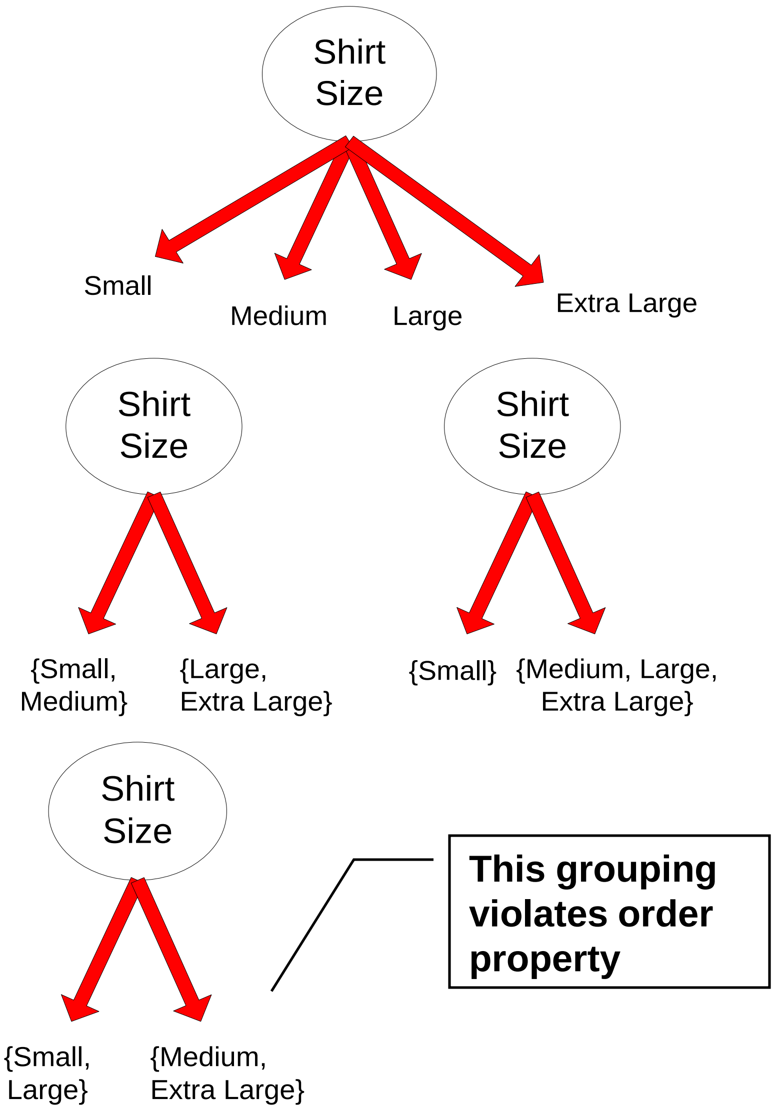


- Binary split:
  - Divides values into two subsets

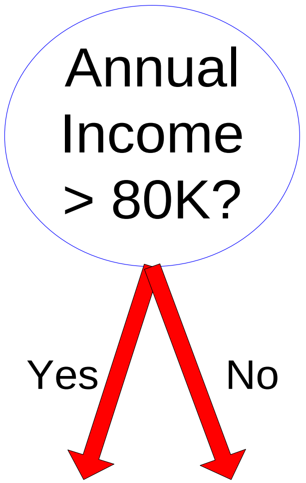


Test Condition for Ordinal Attributes

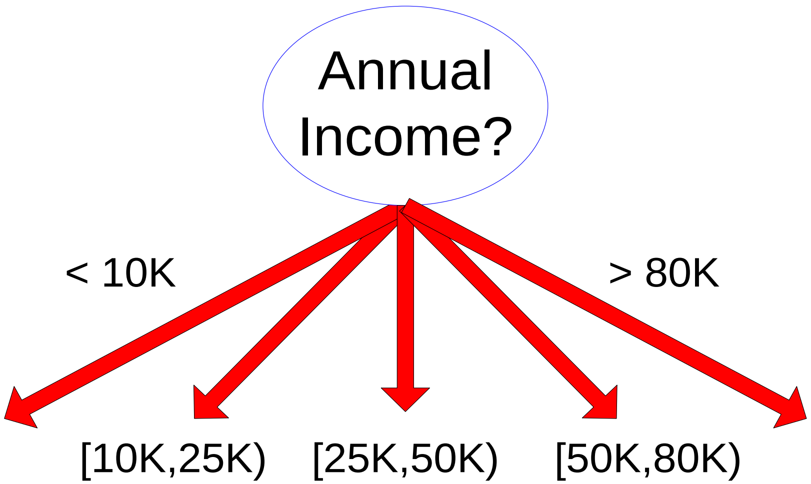
- Multi-way split:
  - Use as many partitions as distinct values
- Binary split:
  - Divides values into two subsets
  - Preserve order property among attribute values



Test Condition for Continuous Attributes



(i) Binary split



(ii) Multi-way split

Splitting Based on Continuous Attributes

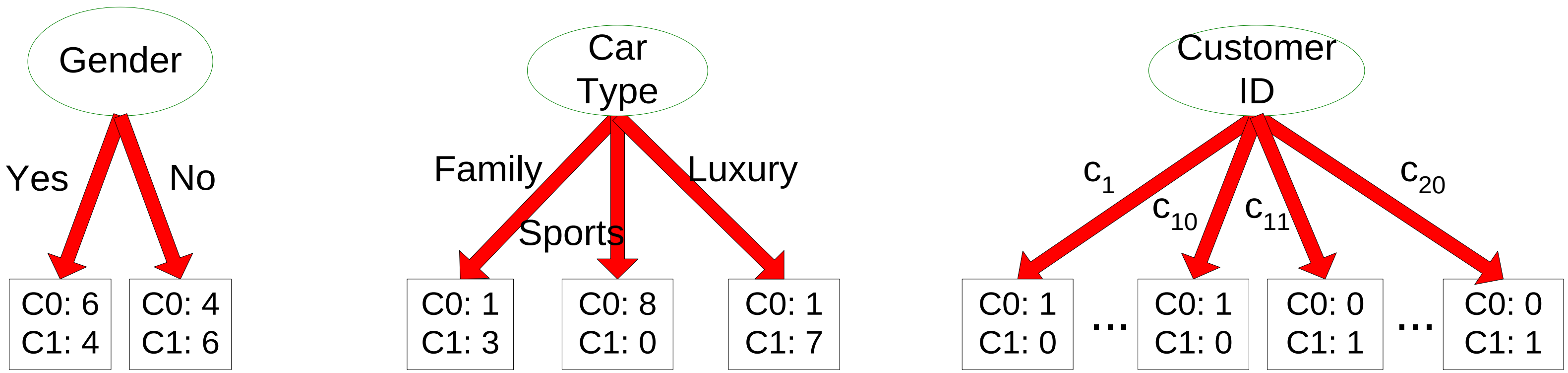
- Different ways of handling
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive
  - Discretization to form an ordinal categorical attribute

    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

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	C0: 9
C1: 5	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 $\mathbf{t}$, 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

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

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

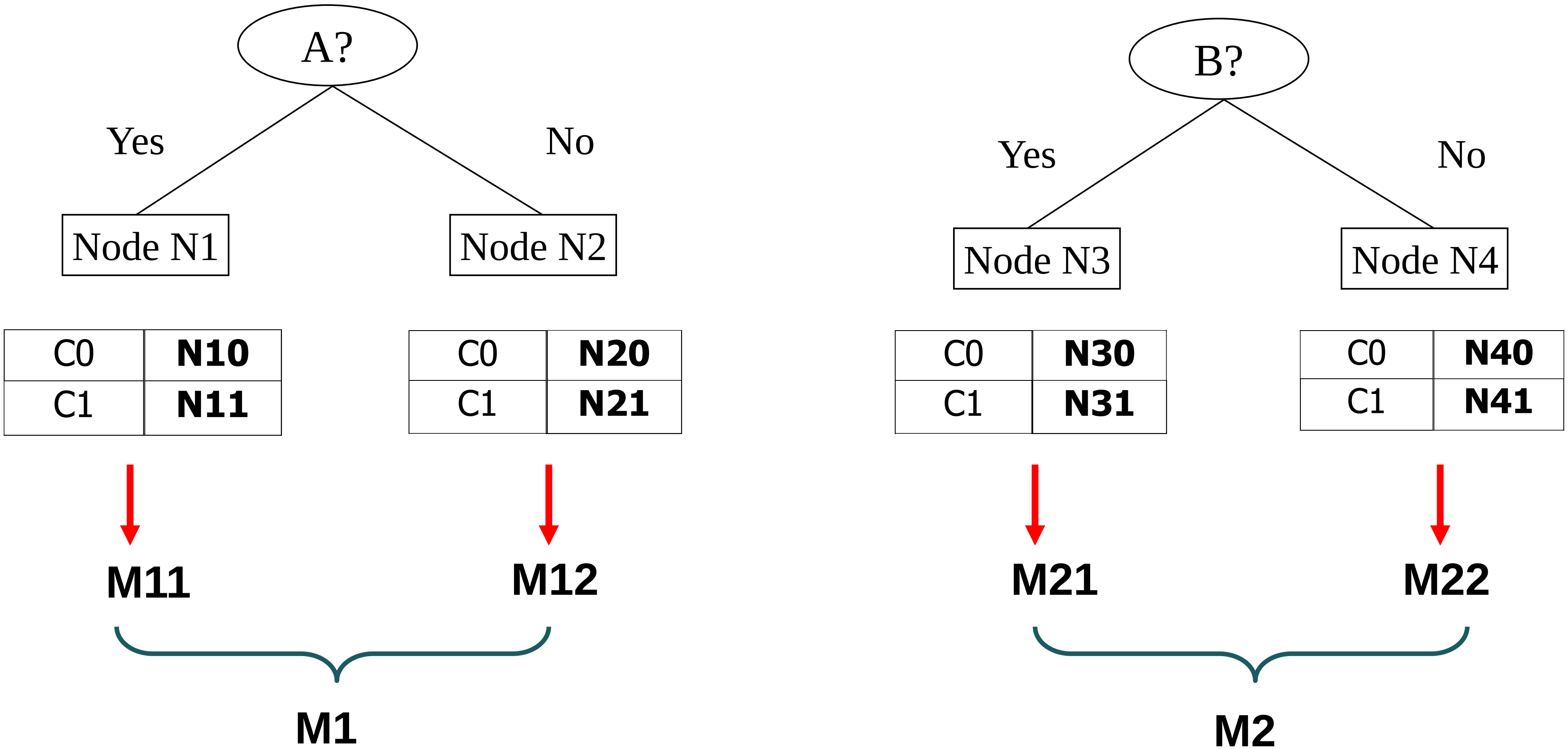
or equivalently, lowest impurity measure after splitting (M)

Finding the Best Split

Before Splitting:

C0	**N00**
C1	**N01**

→ **P**



Gain = P – M1 vs P – M2

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

Computing Gini Index of a Single Node

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

C1	**3**
C2	**3**
Gini=0.50	

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)

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

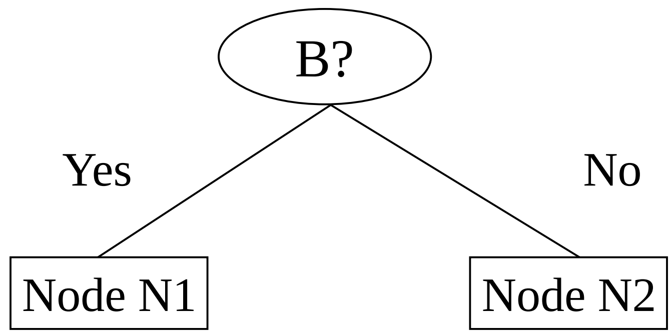
where, n_i = number of records at child i,
n = number of records at parent node p.

- Choose the attribute that minimizes weighted average Gini index of the children

- Gini index is used in decision tree algorithms such as CART, SLIQ, SPRINT

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

Annual Income ?

	≤ 80	> 80
Defaulted Yes	0	3
Defaulted No	3	4

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

Class		No		No		No		Yes		Yes		Yes		No		No		No		No		
		Annual Income (in '000s)																				
Sorted Values →		60		70		75		85		90		95		100		120		125		220		
Split Positions →	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	

Figure 3.14. Splitting continuous attributes.

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	**3**
C2	**3**
Entropy=1.00	

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:

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

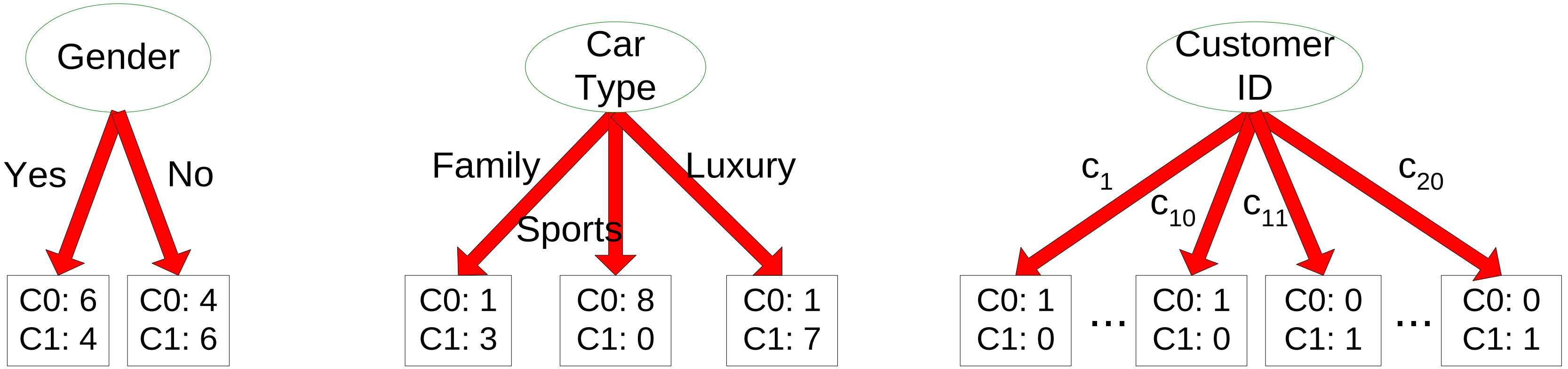
Parent Node p is split into k partitions (children)

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

Problem with large number of partitions

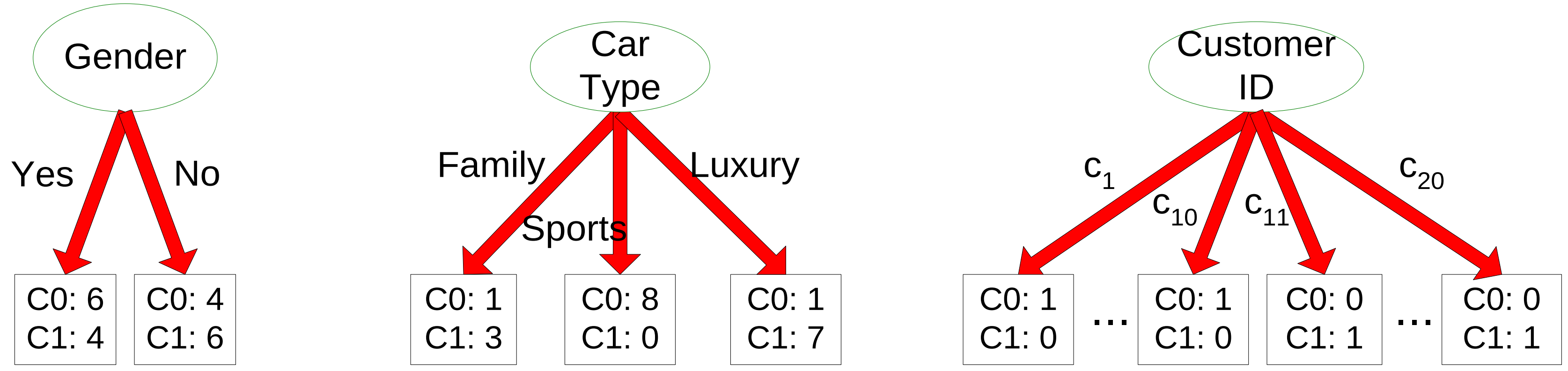
- Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure



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

Problem with large number of partitions

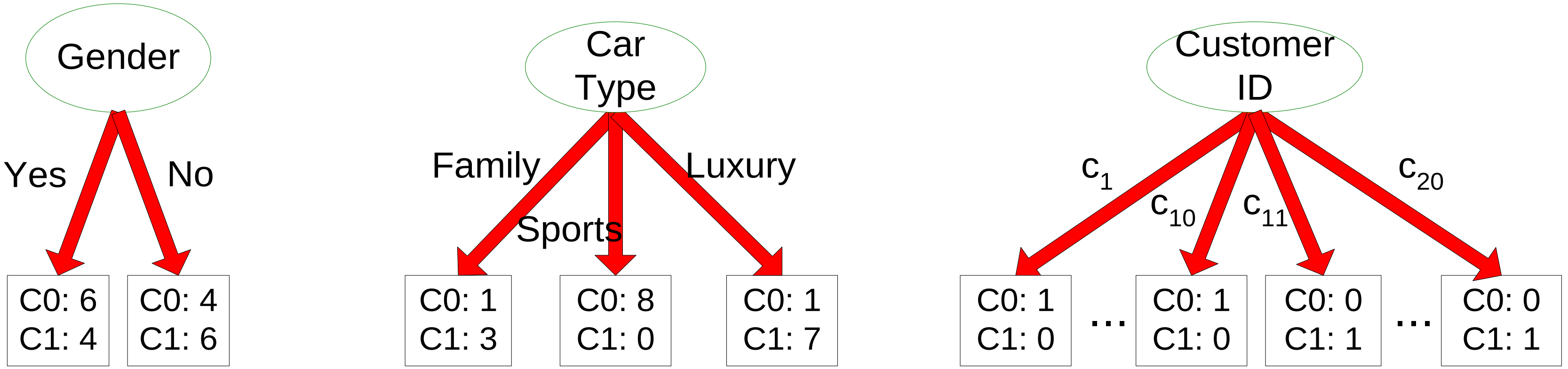
- Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure



  – Having more number of child nodes can make a decision tree more complex and consequently more susceptible to overfitting!!!

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



  – Hence, the number of children produced by the splitting attribute should also be taken into consideration

Gain Ratio

- Solutions:
  - Generate only binary decision trees, thus avoiding the difficulty of handling attributes with varying number of partitions (CART)
  - Modify the splitting criterion to take into account the number of partitions produced by the attribute (C4.5 (next slide))

Gain Ratio

- Gain Ratio:

Evaluates if the split results in a larger number of equally-sized child nodes or not

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

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

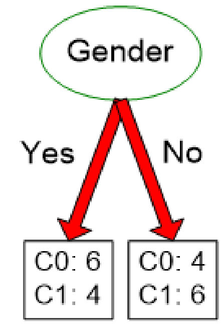
Gain Ratio

$$\text{Entropy(parent)} = -\frac{10}{20}\log_2\frac{10}{20} - \frac{10}{20}\log_2\frac{10}{20} = 1.$$

If **Gender** is used as attribute test condition:

$$\text{Entropy(children)} = \frac{10}{20}\left[-\frac{6}{10}\log_2\frac{6}{10} - \frac{4}{10}\log_2\frac{4}{10}\right] \times 2 = 0.971$$

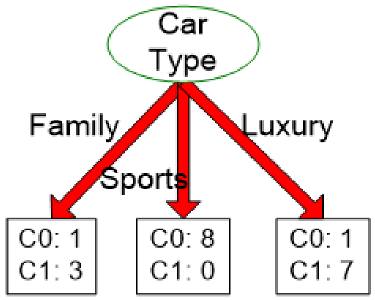
$$\text{Gain Ratio} = \frac{1 - 0.971}{-\frac{10}{20}\log_2\frac{10}{20} - \frac{10}{20}\log_2\frac{10}{20}} = \frac{0.029}{1} = 0.029$$



If **Car Type** is used as attribute test condition:

$$\text{Entropy(children)} = \frac{4}{20}\left[-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}\right] + \frac{8}{20} \times 0 + \frac{8}{20}\left[-\frac{1}{8}\log_2\frac{1}{8} - \frac{7}{8}\log_2\frac{7}{8}\right] = 0.380$$

$$\text{Gain Ratio} = \frac{1 - 0.380}{-\frac{4}{20}\log_2\frac{4}{20} - \frac{8}{20}\log_2\frac{8}{20} - \frac{8}{20}\log_2\frac{8}{20}} = \frac{0.620}{1.52} = 0.41$$



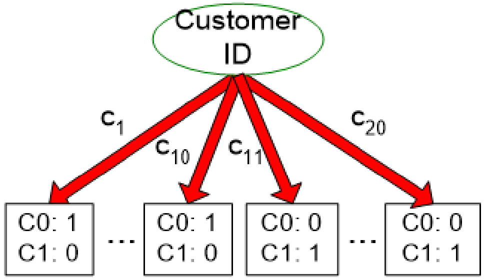
Gain Ratio

$$\text{Entropy(parent)} = -\frac{10}{20}\log_2\frac{10}{20} - \frac{10}{20}\log_2\frac{10}{20} = 1.$$

Finally, if `Customer ID` is used as attribute test condition:

$$\text{Entropy(children)} = \frac{1}{20}\left[-\frac{1}{1}\log_2\frac{1}{1} - \frac{0}{1}\log_2\frac{0}{1}\right] \times 20 = 0$$

$$\text{Gain Ratio} = \frac{1-0}{-\frac{1}{20}\log_2\frac{1}{20} \times 20} = \frac{1}{4.32} = 0.23$$



Gender = 0.029; 0.029
CarType = 0.620; 0.41
Customer ID = 1; 0.23

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

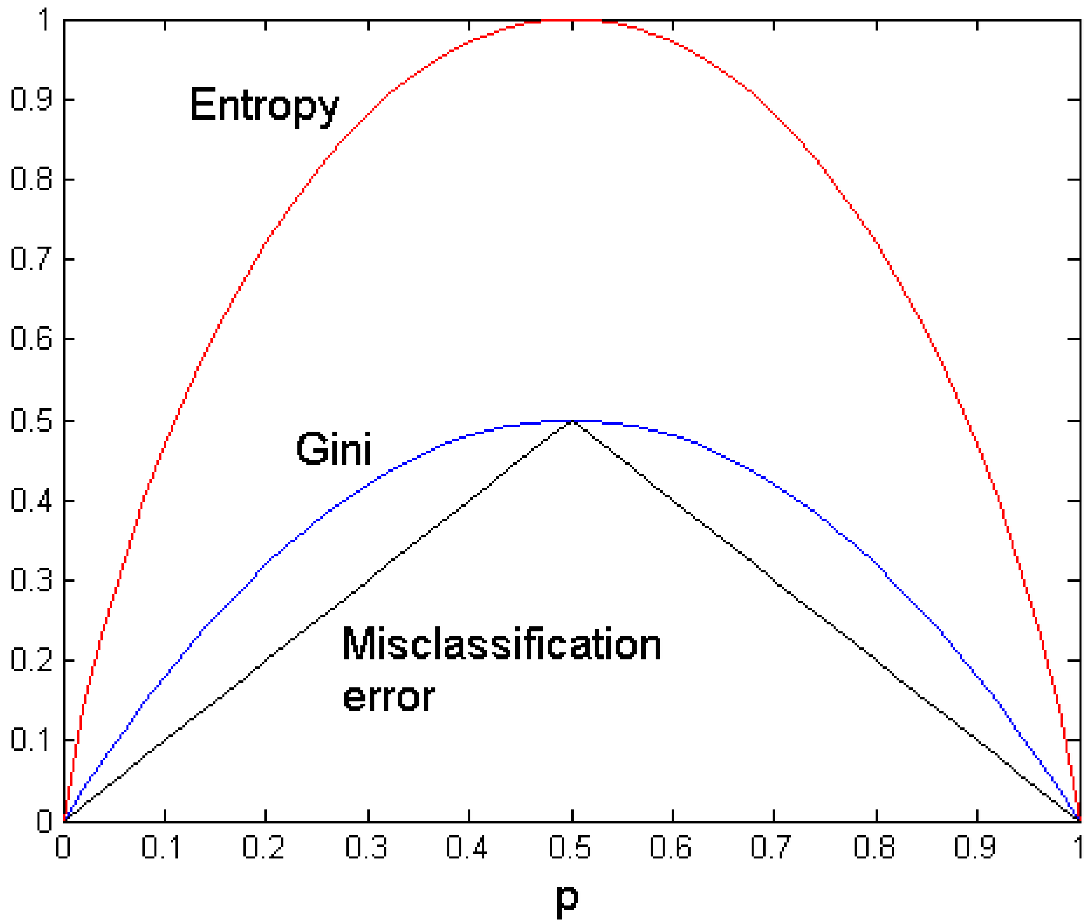
C1	**2**
C2	**4**

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:



Algorithm for Decision Tree Induction

Algorithm 3.1 A skeleton decision tree induction algorithm.

`TreeGrowth` (E, F)
if `stopping_cond`$(E,F) = true$ **then**
    $leaf =$ `createNode`$()$.
    $leaf.label =$ `Classify`(E).
    return $leaf$.
else
    $root =$ `createNode`$()$.
    $root.test\_cond =$ `find_best_split`(E, F).
    let $V = \{v | v$ is a possible outcome of $root.test\_cond$ $\}$.
    for each $v \in V$ **do**
        $E_v = \{e \mid root.test\_cond(e) = v$ and $e \in E\}$.
        $child =$ `TreeGrowth`(E_v, F).
        add $child$ as descendent of $root$ and label the edge $(root \rightarrow child)$ as v.
    end for
end if
return $root$.

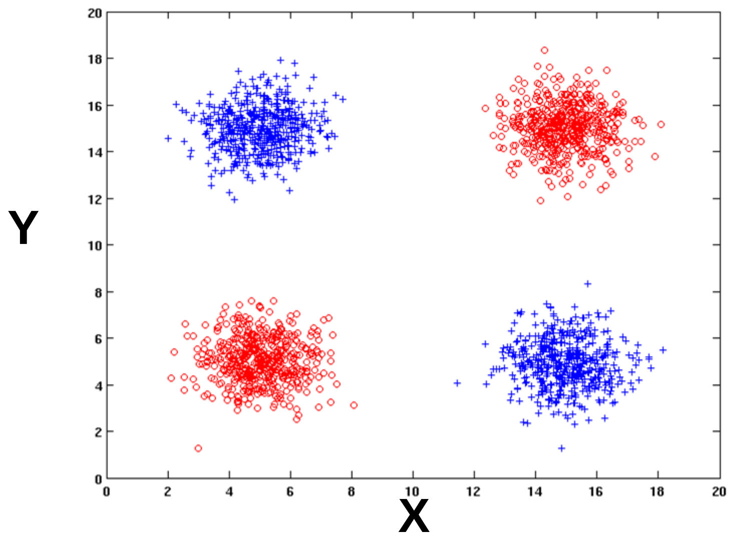
Decision Tree Based Classification

- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Can easily handle redundant or irrelevant attributes (unless the attributes are interacting)
    - Since redundant attributes show similar gains in purity if they are selected for splitting, only one of them will be selected
    - An attribute is irrelevant if it is not useful for the classification task. Since they are poorly associated with the target class labels, they will provide little or no gain in purity and thus will be passed over by other more relevant features

Decision Tree Based Classification

- Disadvantages:
  - Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the best tree.
  - Does not take into account interactions between attributes
    - Attributes are considered interacting if they are able to distinguish between classes when used together, but individually they provide little or no information
    - Such attributes could be passed over in favor of other attributes that are not as useful, resulting in more complex decision trees than necessary
  - Each decision boundary involves only a single attribute

Handling interactions



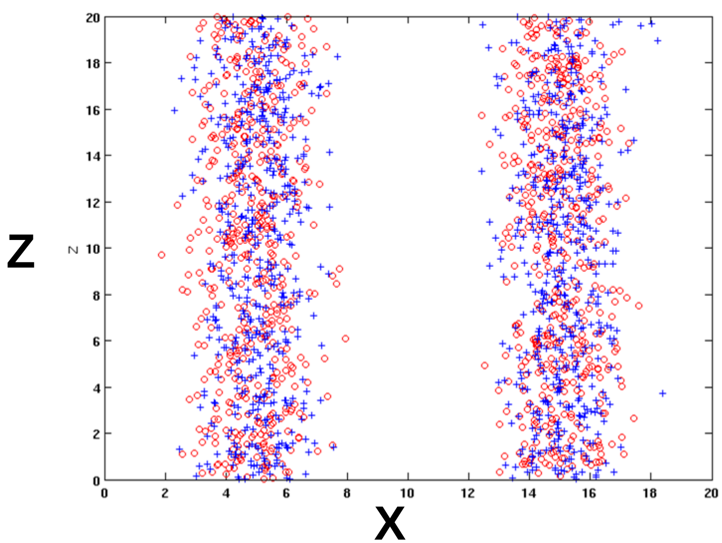
+ : 1000 instances

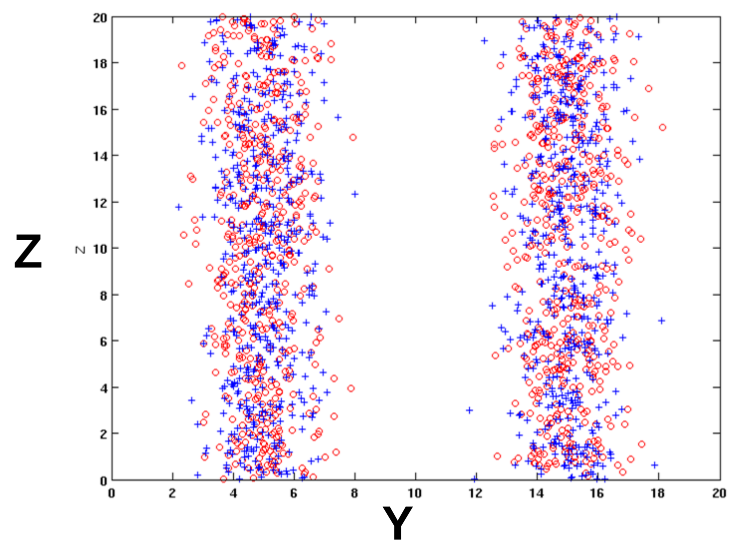
o : 1000 instances

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!





Limitations of single attribute-based decision boundaries

Usa Divisões Retilíneas

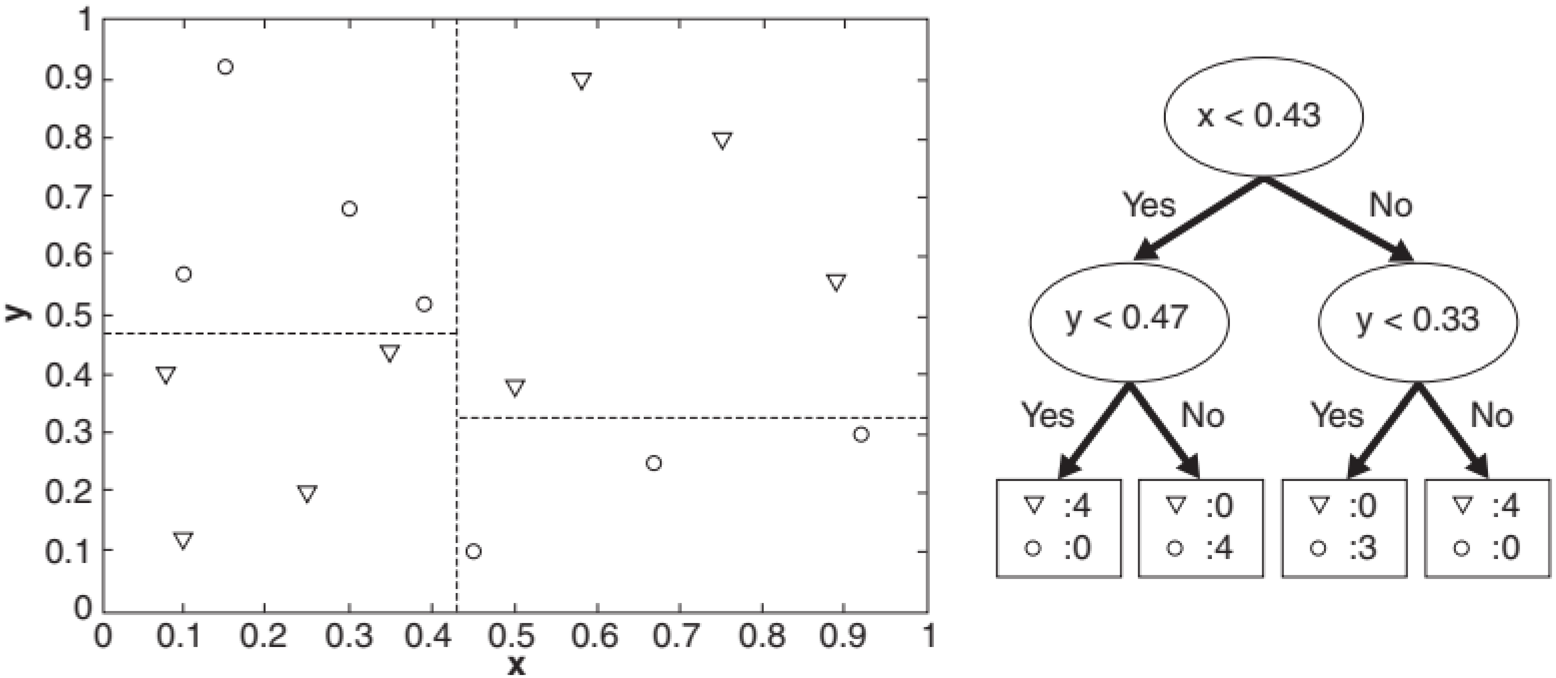


Figure 3.20. Example of a decision tree and its decision boundaries for a two-dimensional data set.

Limitations of single attribute-based decision boundaries

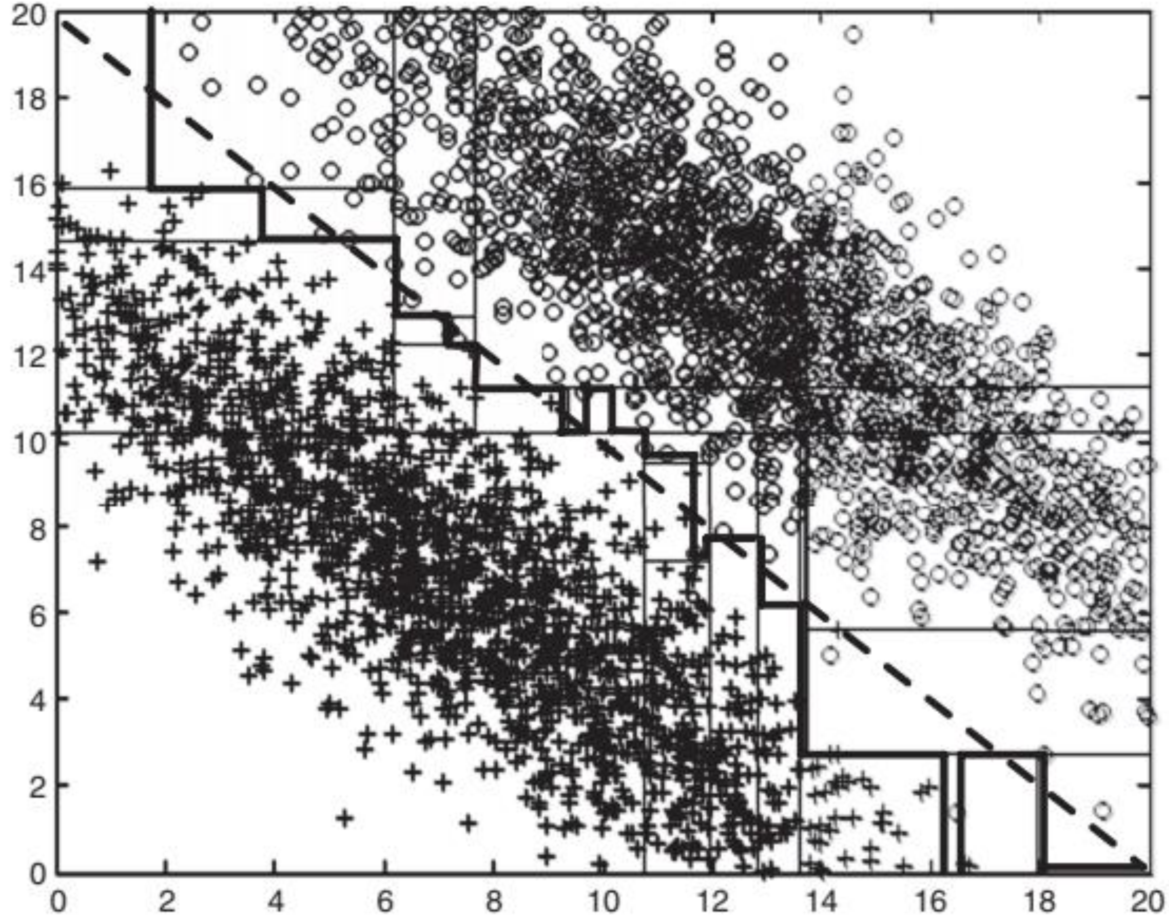


Figure 3.21. Example of data set that cannot be partitioned optimally using a decision tree with single attribute test conditions. The true decision boundary is shown by the dashed line.

Data Mining

Model Overfitting and Evaluation

Introduction to Data Mining, 2nd Edition

by

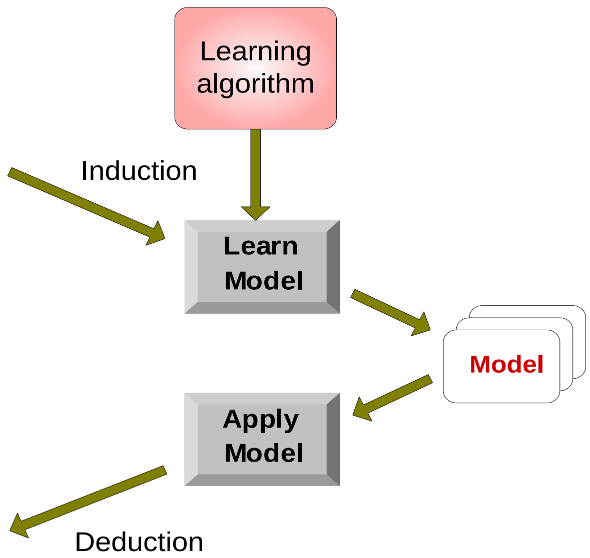
Tan, Steinbach, Karpatne, Kumar

Classification Errors

- **Training errors (apparent 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

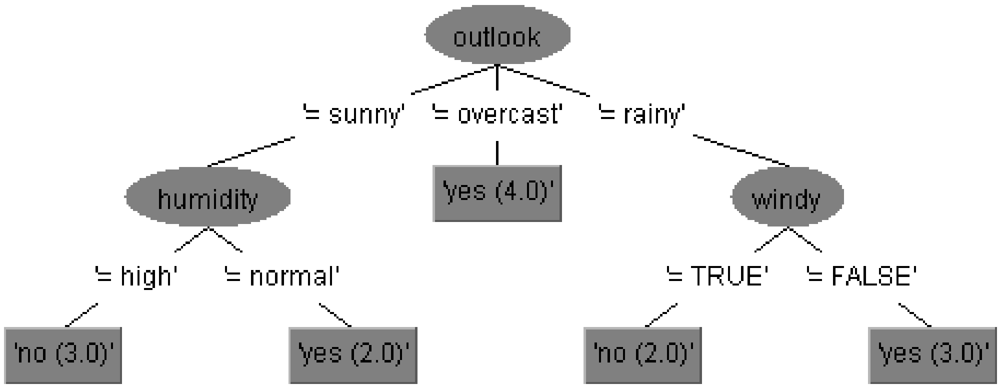
Training Set



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

Classification Errors



		PREDICTED CLASS	
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	9	0
	Class=No	0	5

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	**play** Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

Classification Errors

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

Ref. [?]

Classification Errors

	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

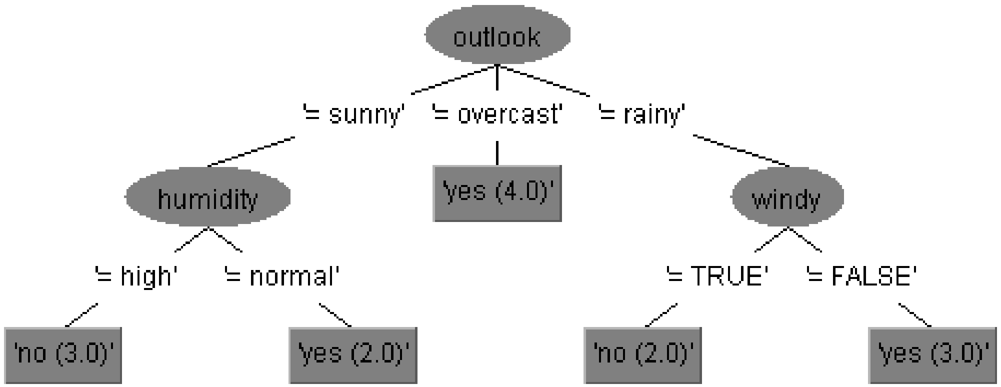
Classification Errors

	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Complement:

$$\text{Error} = \frac{b+c}{a+b+c+d} = \frac{FN+FP}{TP+TN+FP+FN}$$

Classification Errors



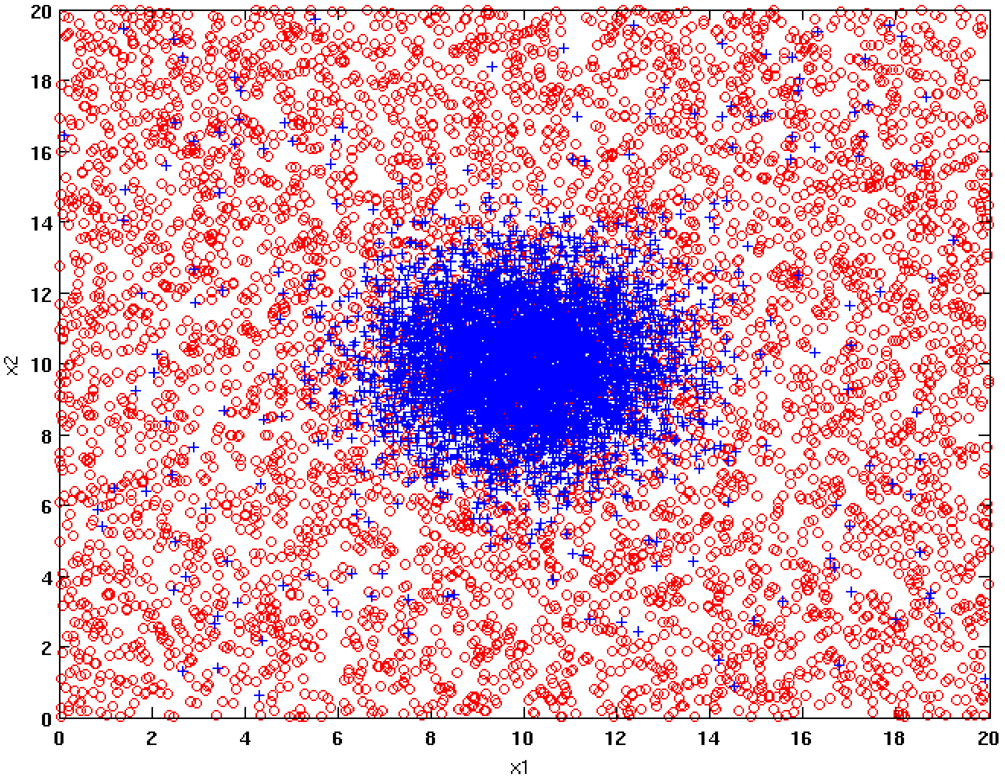
	PREDICTED CLASS	
ACTUAL CLASS	Class=Yes	Class=No
Class=Yes	9	0
Class=No	0	5

	PREDICTED CLASS	
ACTUAL CLASS	Class=Yes	Class=No
Class=Yes	5	4
Class=No	3	2

Error (Train) = 0/14 = 0

Error (Test) = (4+3)/14 = 0.5

Example Data Set



Two class problem:

+ : 5400 instances

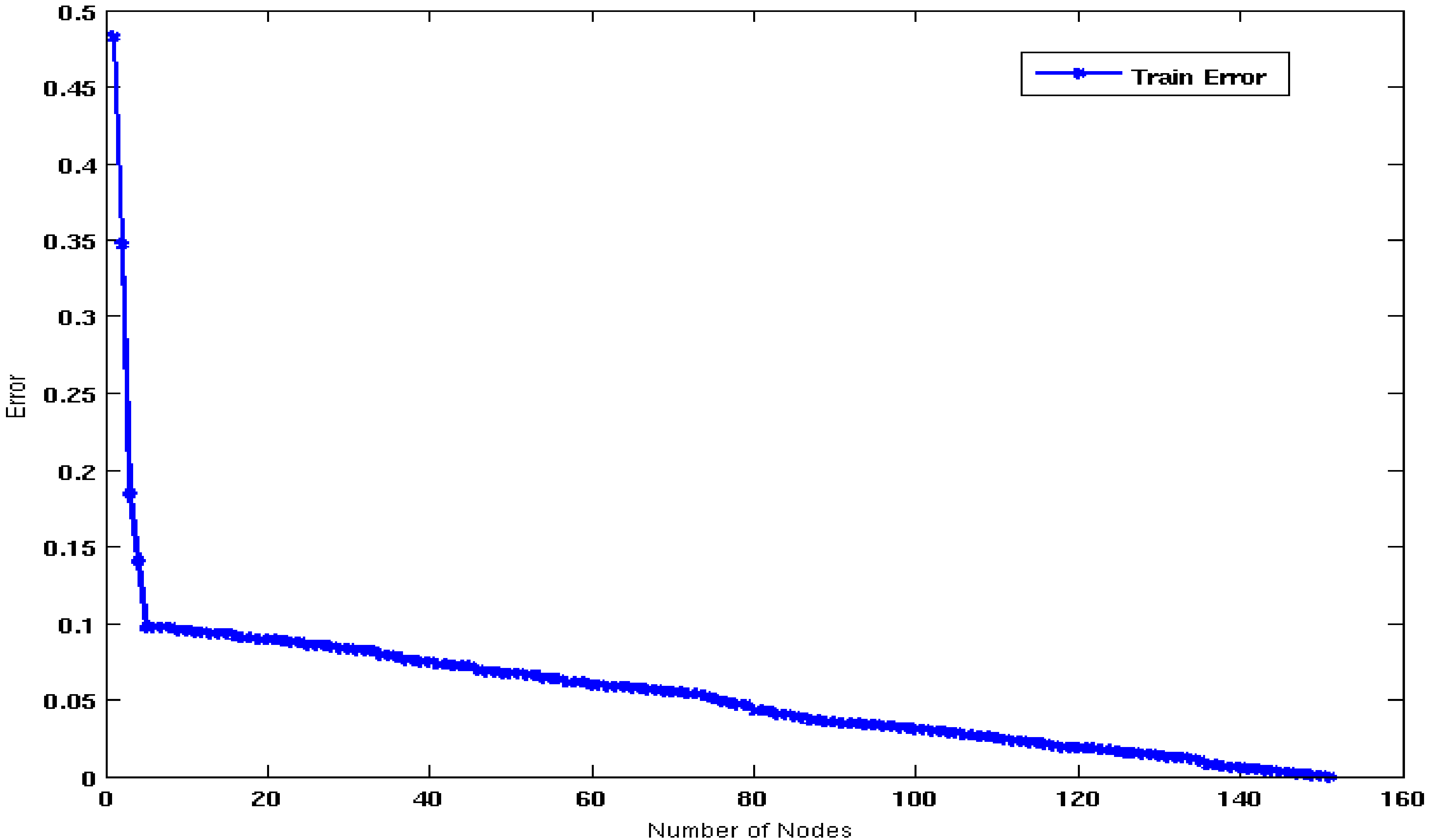
- **5000 instances generated from a Gaussian centered at (10,10)**
- **400 noisy instances added**

o : 5400 instances

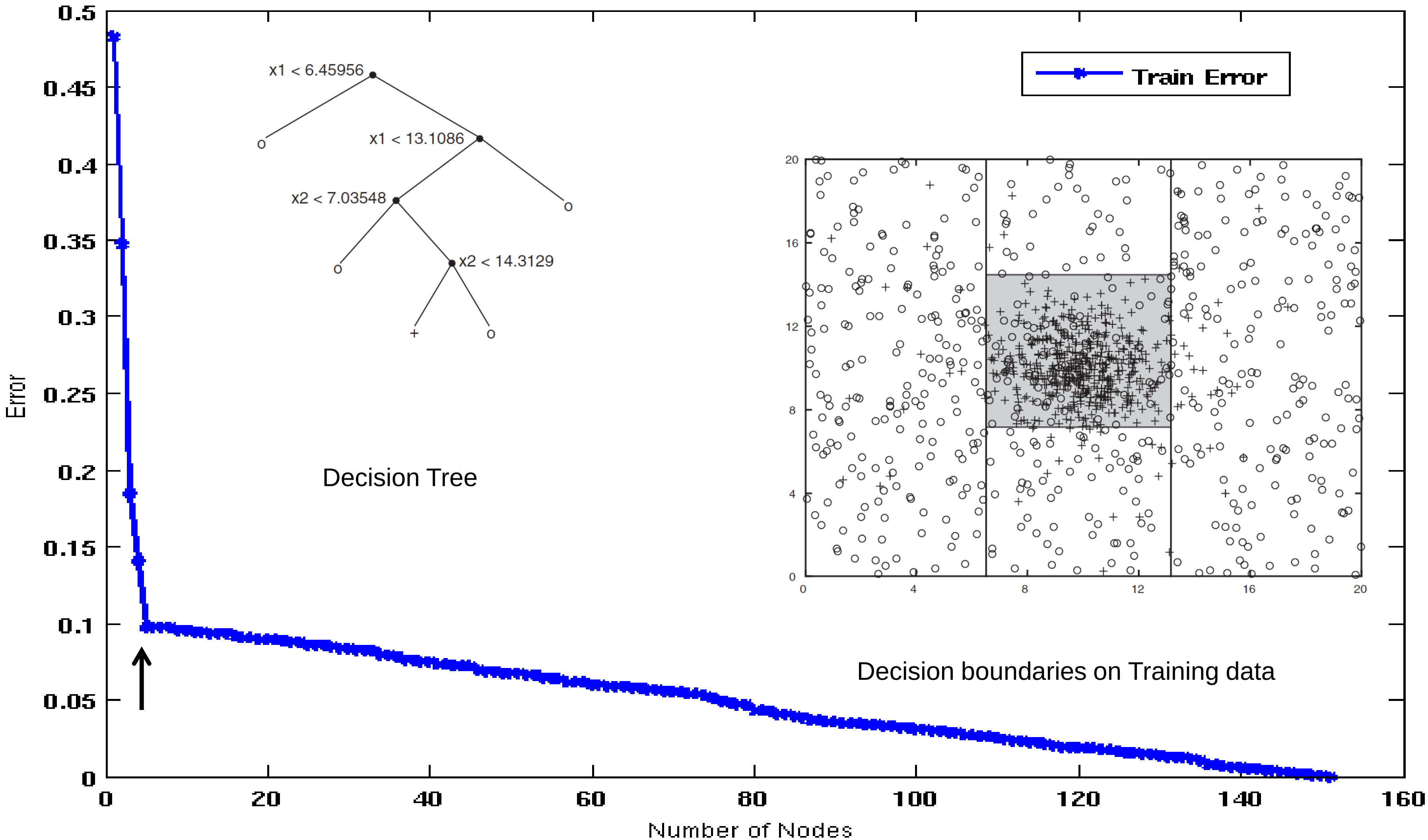
- **Generated from a uniform distribution**

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

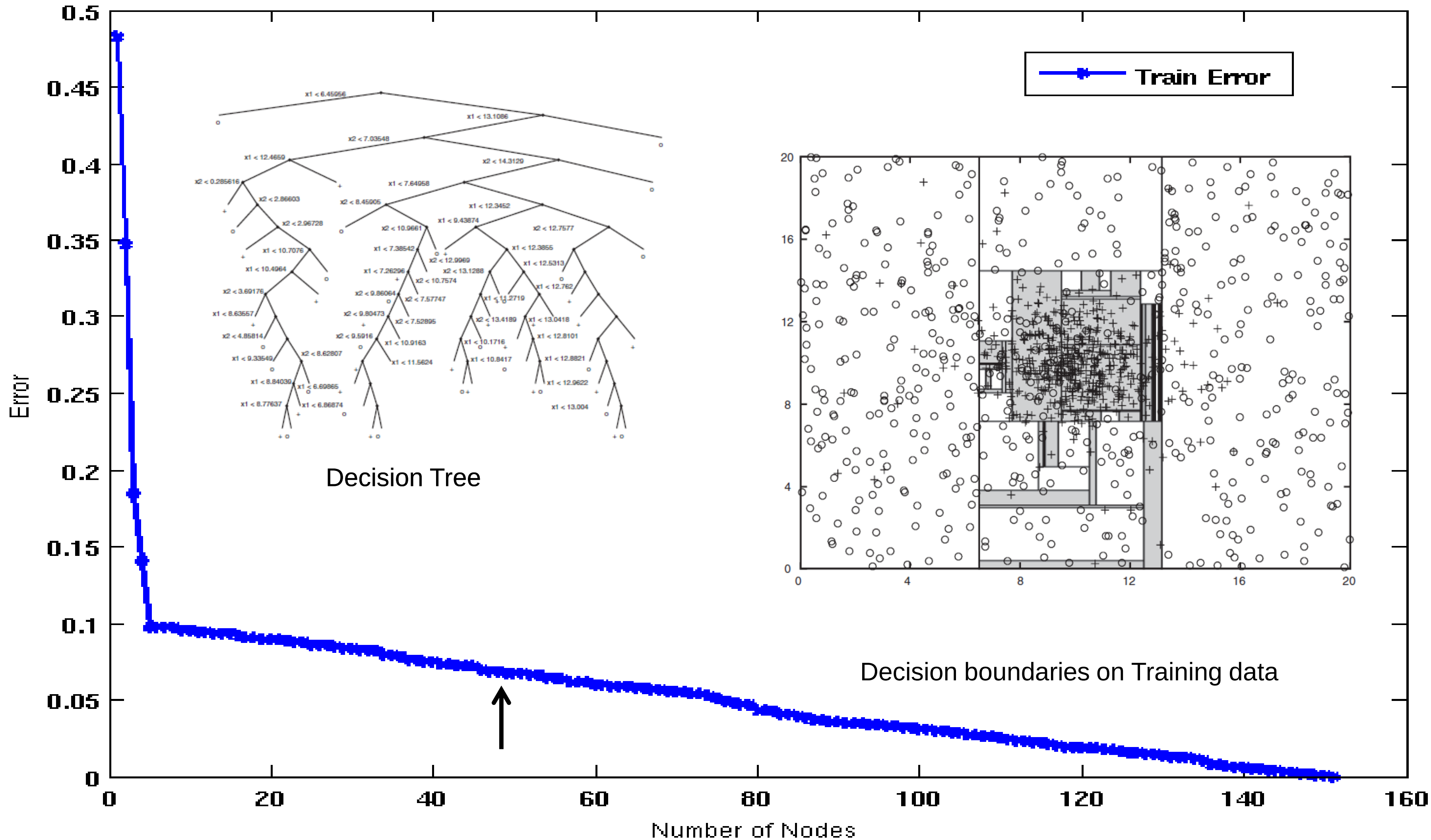
Increasing number of nodes in Decision Trees



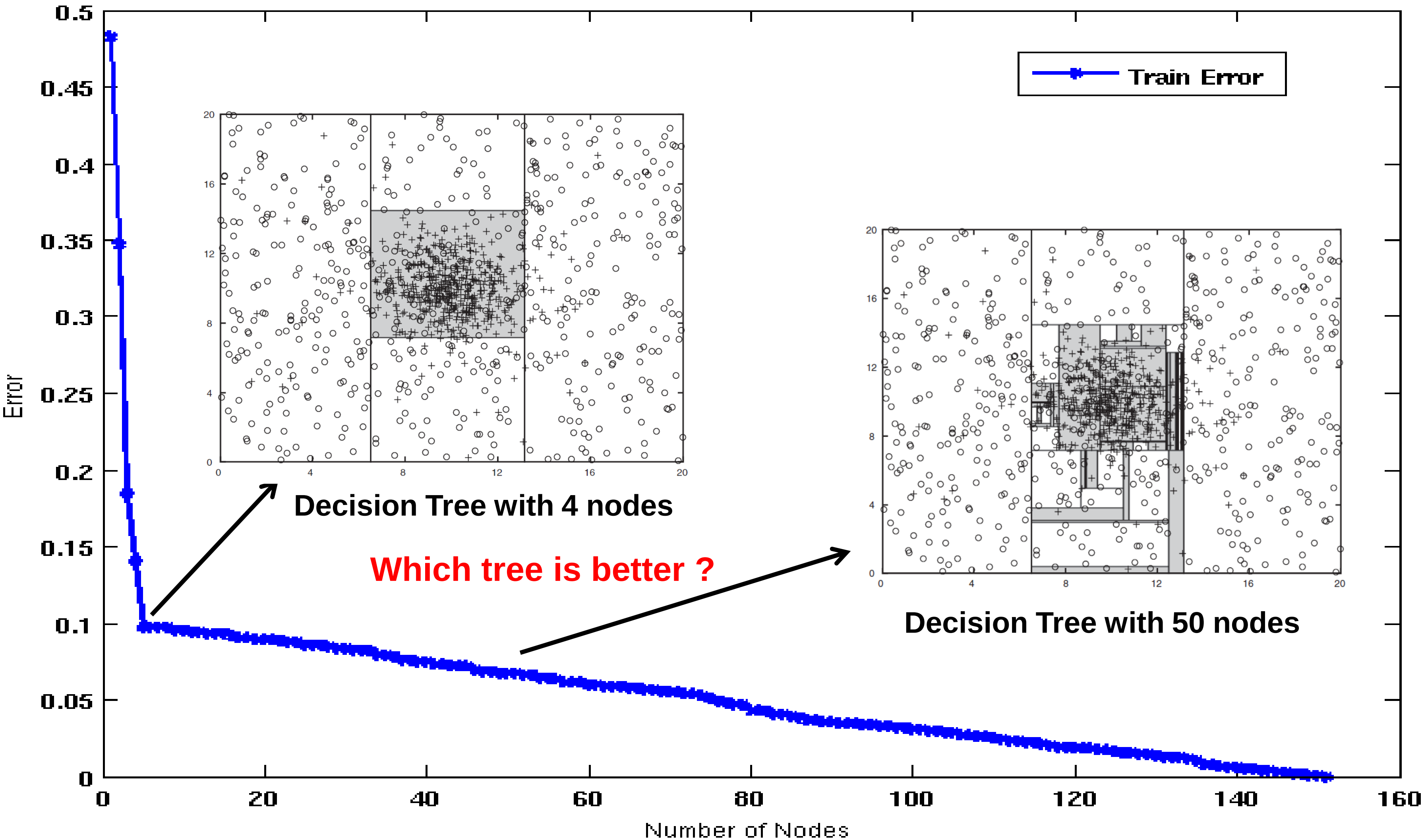
Decision Tree with 4 nodes



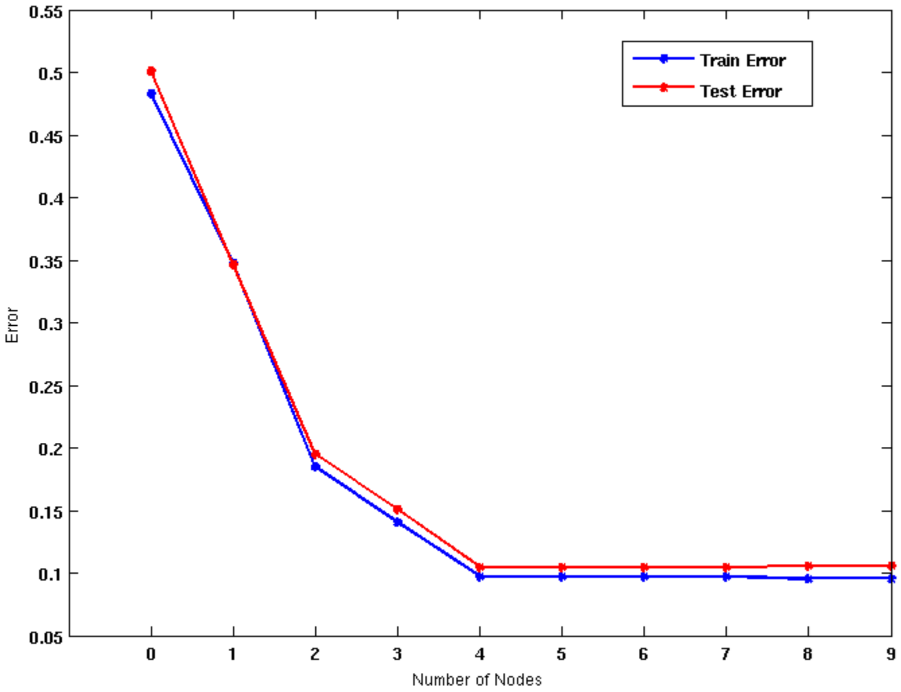
Decision Tree with 50 nodes

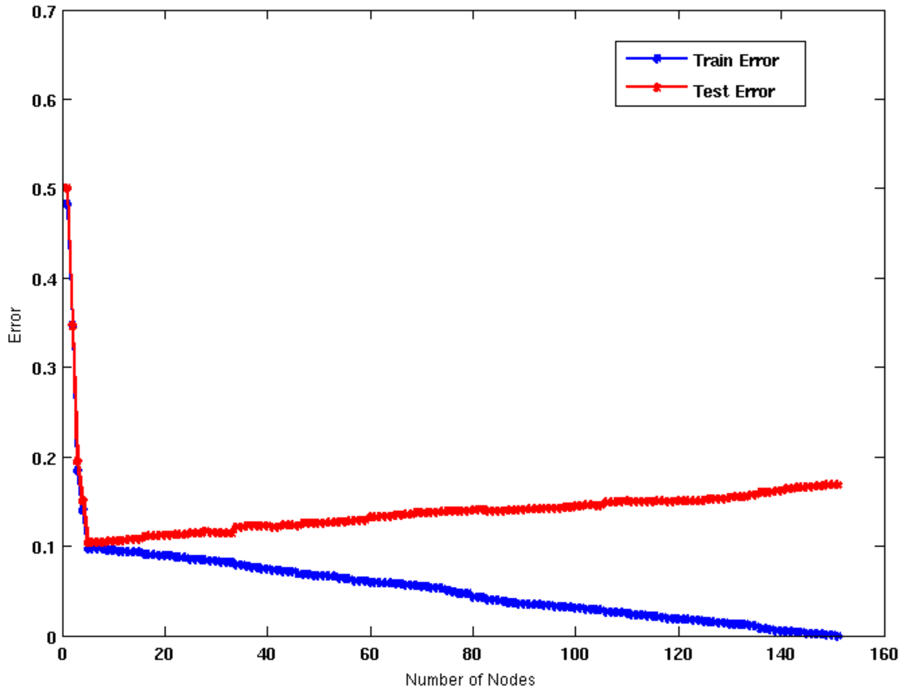


Which tree is better?



Model Overfitting





•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

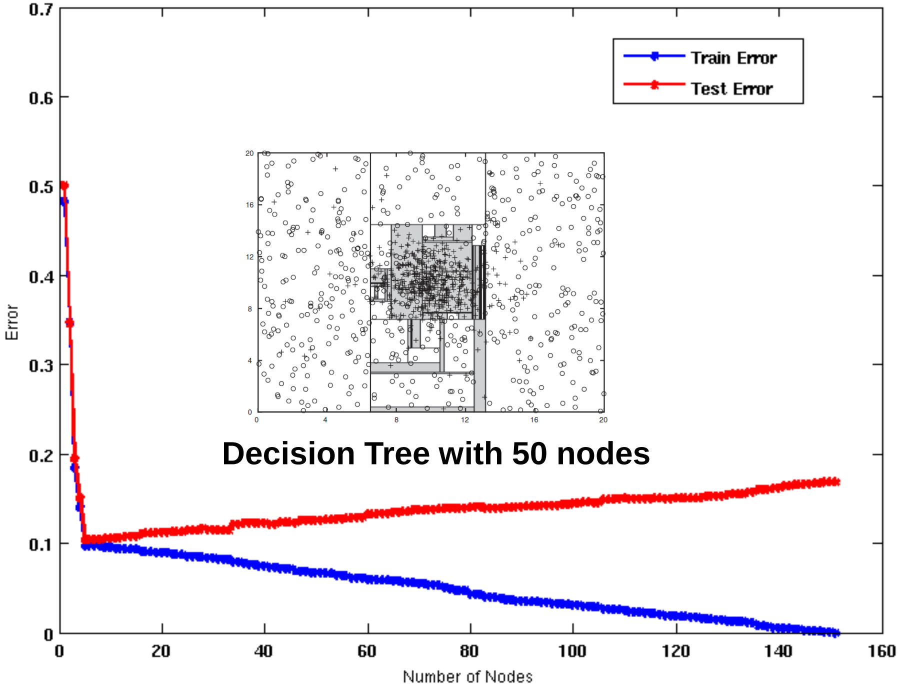
Overfitting: when model is too complex, training error is small but test error is large

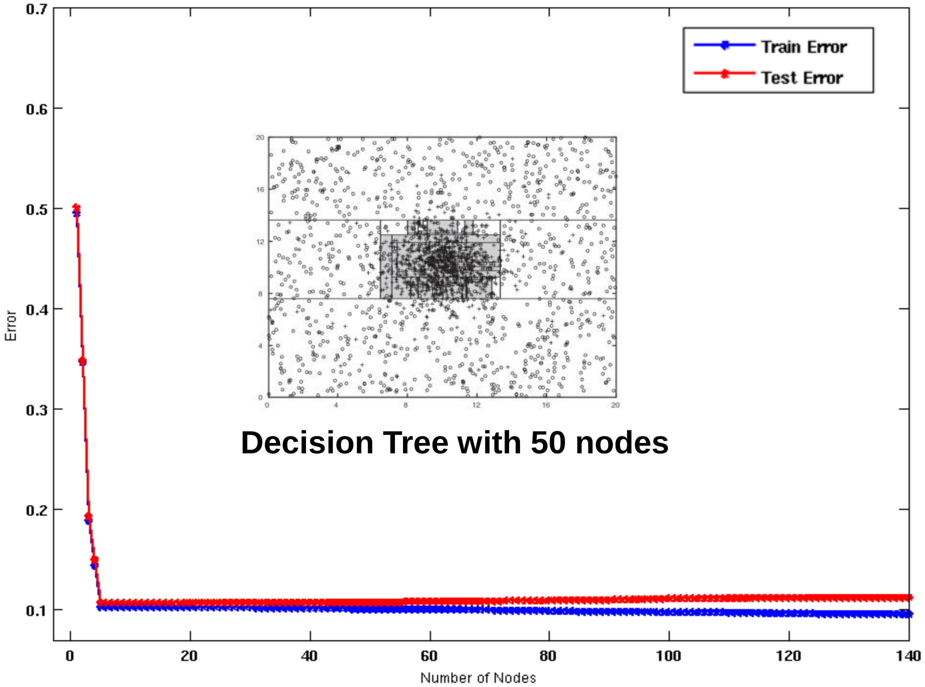
Reasons for Model Overfitting

- Limited Training Size

- High Model Complexity

Model Overfitting





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

Notes on Overfitting

- Overfitting results in decision trees that are <u>more complex</u> 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

Model Selection

- There are many possible classification models with varying levels of model complexity that can be used to capture patterns in the training data
- Among these possibilities, we want to select the model that shows lowest generalization error rate
- The process of selecting a model with the right level of complexity, which is expected to generalize well over unseen test instances, is known as **model selection**

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity
  - Estimating Statistical Bounds

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - **Using Validation Set**
  - **Incorporating Model Complexity**
  - Estimating Statistical Bounds

Model Selection:

Using Validation Set

- Divide training data into two parts:
  - Training set:
    - use for model building
  - Validation set:
    - use for estimating generalization error
    - Note: validation set is not the same as test set
- The use of validation set provides a generic approach for model selection
- Drawback:
  - Less data available for training

Model Selection:

Using Validation Set

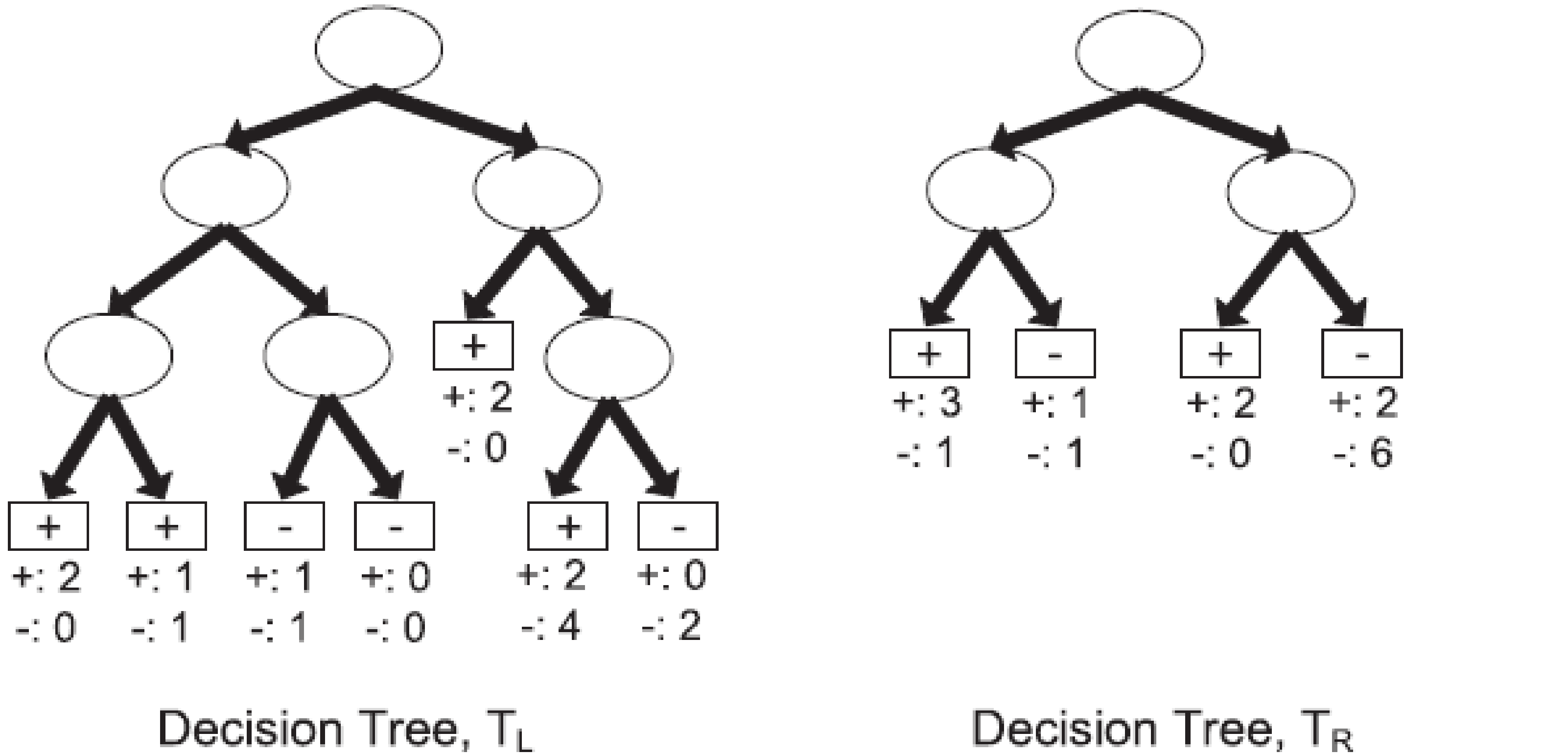


Figure 3.29. Class distribution of validation data for the two decision trees shown in Figure 3.30.

$err_{val}(T_L) = 6/16 = 0.375$

$err_{val}(T_R) = 4/16 = 0.25$

Model Selection:

Incorporating Model Complexity

- Rationale: Occam's Razor (Princípio)
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - A complex model has a greater chance of being fitted accidentally
  - Therefore, one should include model complexity when evaluating a model

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

α is a hyper-parameter that strikes a balance between minimizing training error and reducing model complexity**.** A higher value of *α* gives more emphasis to the model complexity in the estimation of generalization performance

Estimating the Complexity of Decision Trees

- **Pessimistic Error Estimate** of decision tree T with k leaf nodes:

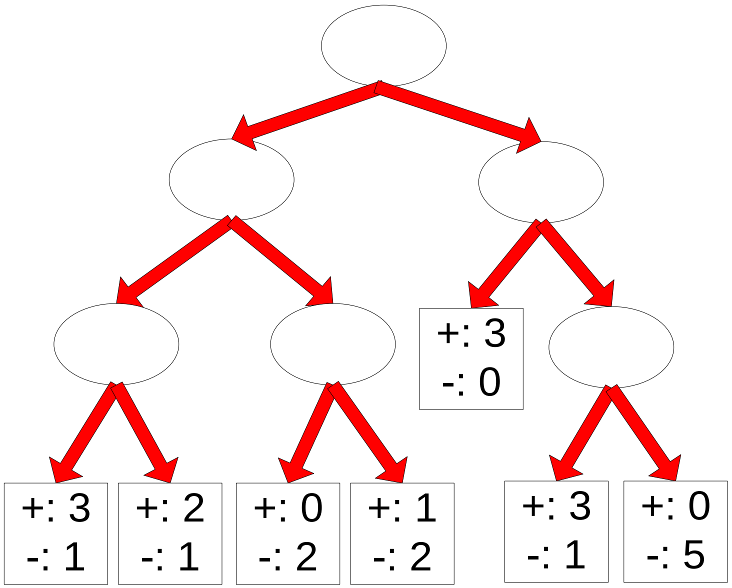
It is called pessimistic as it assumes the generalization error rate to be worse than the training error rate (**optimistic error estimate**), by adding a penalty term for model complexity

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}$$

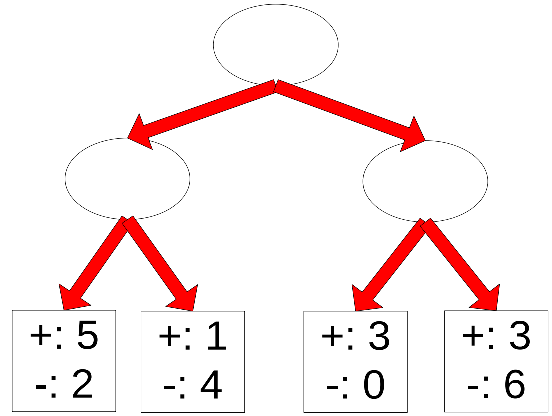
  - err(T): error rate on all training records
  - Ω: trade-off hyper-parameter (similar to α)
    - Relative cost of adding a leaf node
  - k: number of leaf nodes
  - N_{train}: total number of training records

Estimating the Complexity of Decision Trees: Example

Example of two decision trees generated from the same training data



Decision Tree, T_L



Decision Tree, T_R

$e(T_L) = 4/24 = 0{,}17$

$e(T_R) = 6/24 = 0{,}25$

$\Omega = 1$

$e_{gen}(T_L) = 4/24 + 1*7/24 = 11/24 = 0.458$

$e_{gen}(T_R) = 6/24 + 1*4/24 = 10/24 = 0.417$

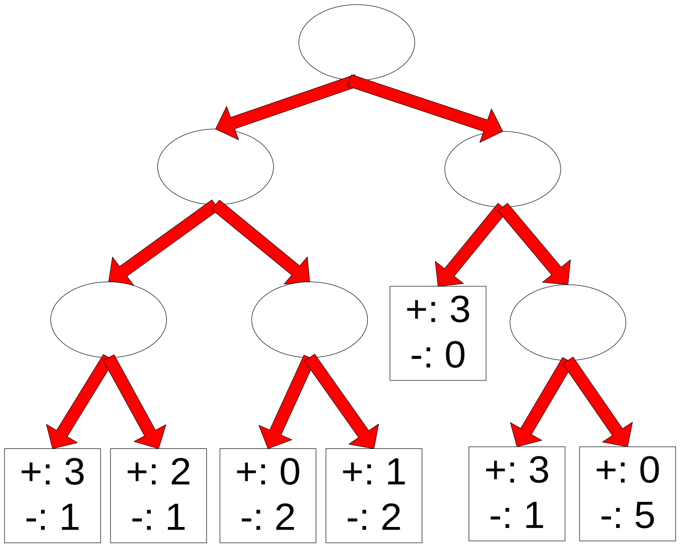
$\Omega = 0.5$

$e_{gen}(T_L) = 4/24 + 0.5*7/24 = 7.5/24 = 0.313$

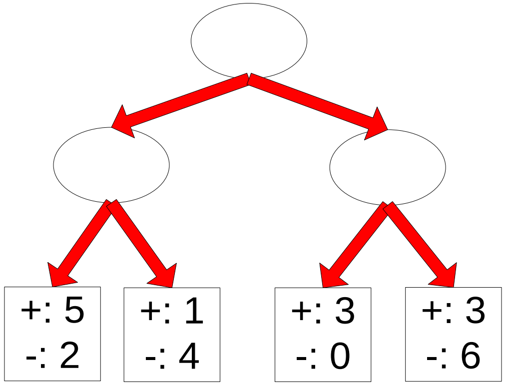
$e_{gen}(T_R) = 6/24 + 0.5*4/24 = 8/24 = 0.333$

Estimating the Complexity of Decision Trees

- Resubstitution Estimate:
  - Using training error as an optimistic estimate of generalization error
  - Referred to as optimistic error estimate



Decision Tree, T_L



Decision Tree, T_R

$e(T_L) = 4/24 = 0{,}17$

$e(T_R) = 6/24 = 0{,}25$

Model Selection for Decision Trees

- Building on the generic approaches presented above, we present two commonly used model selection strategies for decision tree induction
  - Pre-Pruning (Early Stopping Rule)
    - The tree-growing algorithm is halted before generating a fully grown tree that perfectly fits the entire training data
      - Drawback: subsequent splittings may result in better subtrees
  - Post-pruning
    - Tends to give better results because it makes pruning decisions based on a fully grown tree (additional computations needed)

Model Selection for Decision Trees

- Pre-Pruning (Early Stopping Rule)
  - 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 class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or Information Gain).
    - Stop if estimated generalization error falls below certain threshold (computed as previous)

Model Selection for Decision Trees

- Post-pruning
  - Grow decision tree to its entirety
  - Subtree replacement
    - Trim the nodes of the decision tree in a bottom-up fashion
    - If generalization error (computed as previous) 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
  - Subtree raising
    - Replace subtree with most frequently used branch

Example of Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

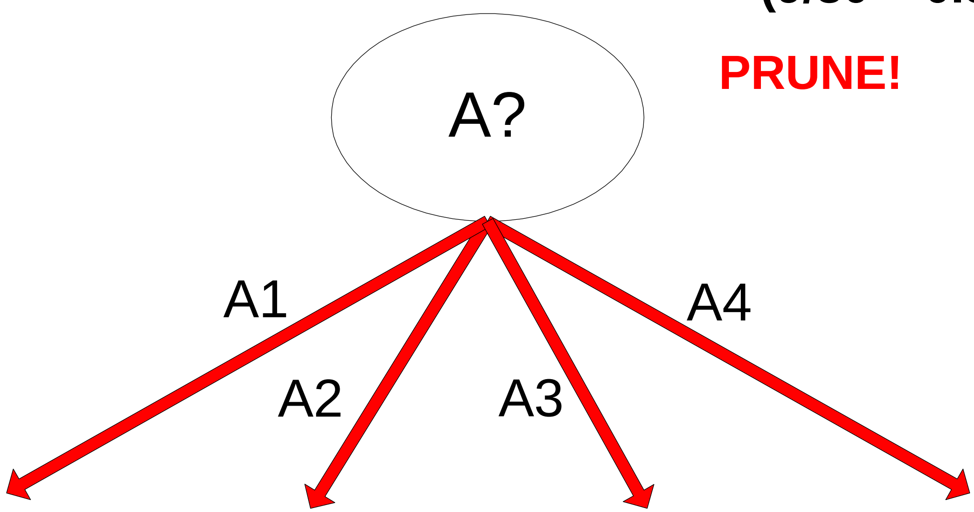
Pessimistic error = (10/30 + 0.5 * 1/30) = 0.35

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9/30 + 0.5 * 4/30) = 0.37

PRUNE!



Class = Yes	8
Class = No	**4**

Class = Yes	**3**
Class = No	4

Class = Yes	4
Class = No	**1**

Class = Yes	5
Class = No	**1**

Examples of Post-pruning

Decision Tree:

```
depth = 1 :
|   breadth > 7 : class 1
|   breadth <= 7 :
|   |   breadth <= 3 :
|   |   |   ImagePages > 0.375 : class 0
|   |   |   ImagePages <= 0.375 :
|   |   |   |   totalPages <= 6 : class 1
|   |   |   |   totalPages > 6 :
|   |   |   |   |   breadth <= 1 : class 1
|   |   |   |   |   breadth > 1 : class 0
|   |   width > 3 :
|   |   |   MultiIP = 0:
|   |   |   |   ImagePages <= 0.1333 : class 1
|   |   |   |   ImagePages > 0.1333 :
|   |   |   |   |   breadth <= 6 : class 0
|   |   |   |   |   breadth > 6 : class 1
|   |   |   MultiIP = 1:
|   |   |   |   TotalTime <= 361 : class 0
|   |   |   |   TotalTime > 361 : class 1
depth > 1 :
|   MultiAgent = 0:
|   |   depth > 2 : class 0
|   |   depth <= 2 :
|   |   |   MultiIP = 1: class 0
|   |   |   MultiIP = 0:
|   |   |   |   breadth <= 6 : class 0
|   |   |   |   breadth > 6 :
|   |   |   |   |   RepeatedAccess <= 0.0322 : class 0
|   |   |   |   |   RepeatedAccess > 0.0322 : class 1
|   MultiAgent = 1:
|   |   totalPages <= 81 : class 0
|   |   totalPages > 81 : class 1
```

Subtree Raising

Subtree Replacement

Simplified Decision Tree:

```
depth = 1 :
|   ImagePages <= 0.1333 : class 1
|   ImagePages > 0.1333 :
|   |   breadth <= 6 : class 0
|   |   breadth > 6 : class 1
depth > 1 :
|   MultiAgent = 0: class 0
|   MultiAgent = 1:
|   |   totalPages <= 81 : class 0
|   |   totalPages > 81 : class 1
```

Model Evaluation

- Purpose:
  - To estimate performance of classifier on previously unseen data (test set)

- **Holdout**
  - Reserve k% for training and (100-k)% for testing
  - **Random subsampling**: repeated holdout
- **Bootstrap**
- **Cross validation**
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - **Leave-one-out**: k=n (custoso e, portanto, aplicado geralmente em amostras de dados pequenas)

Model Evaluation

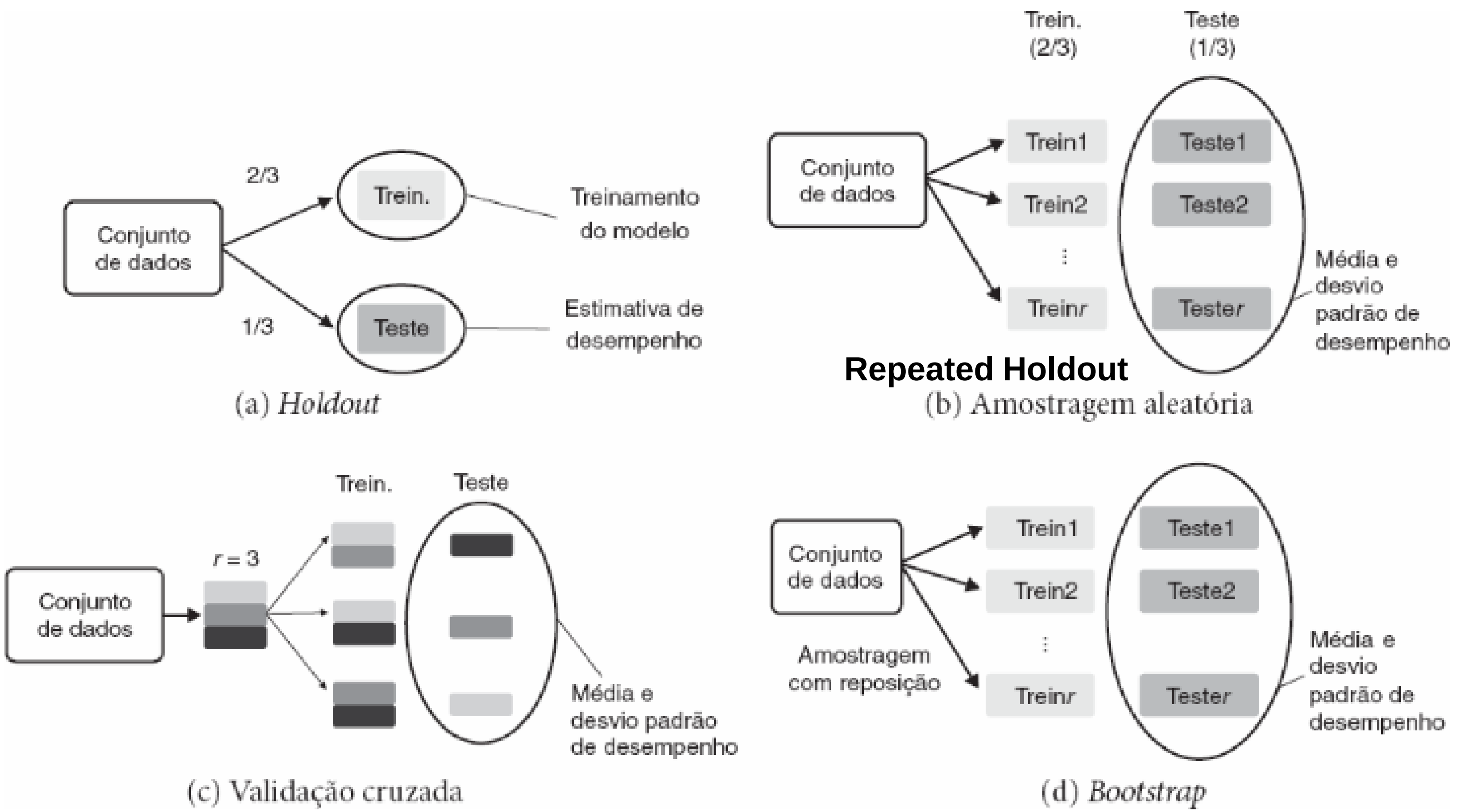


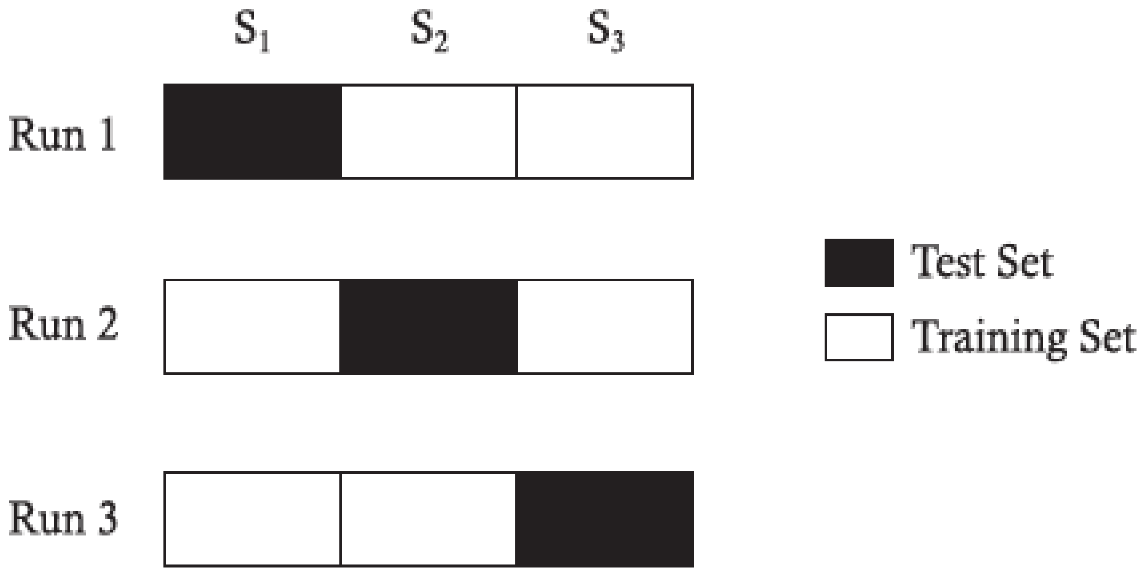
FIGURA 10.2 Métodos de amostragem.

Bootstrap

- r subconjuntos de treinamento são gerados a partir do conjunto de exemplos original por meio de amostragem aleatória com reposição
  - Um exemplo pode estar presente em determinado subconjunto de treinamento mais de uma vez
- Os exemplos não selecionados compõem os subconjuntos de teste ($\cong$ 36,8% para n grande)
- O resultado final é dado pela média do desempenho observado em cada subconjunto de teste
- Em geral, r$\geq$100, sendo, portanto, um procedimento custoso e aplicado geralmente em amostras de dados pequenas (desempenho estatisticamente equivalente ao do leave-one-out, com menor variância)

Cross-validation Example

- 3-fold cross-validation



Variations on Cross-validation

- Repeated cross-validation
  - Perform cross-validation a number of times
  - Gives an estimate of the variance of the generalization error
- Stratified cross-validation
  - Guarantee the same percentage of class labels in training and test
  - Important when classes are imbalanced and the sample is small
- Use nested cross-validation (or double cross-validation) approach for model selection and evaluation

Presence of Hyper-parameters

- Hyper-parameters are parameters of learning algorithms that need to be determined before learning the classification model

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

- The values of hyper-parameters need to be determined during model selection – a process known as **hyper-parameter selection** – and must be taken into account during model evaluation

Presence of Hyper-parameters

- **Using a validation set**
  - Let p be the hyper-parameter that needs to be selected from a finite range of values, $P = \{p_1, p_2, \ldots p_n\}$
  - Partition *D.train* into *D.tr* and *D.val*
  - For every choice of hyper-parameter value p_i, we can learn a model m_i on *D.tr*, and apply this model on *D.val* to obtain the validation error rate $err_{val}(p_i)$
  - Let p^* be the hyper-parameter value that provides the lowest validation error rate
  - Use the model m^* corresponding to p^* as the final choice of classification model

Presence of Hyper-parameters

- **Using a validation set**
  - The above approach, although useful, uses only a subset of the data, *D.train*, for training and a subset, *D.val*, for validation

Presence of Hyper-parameters

- **Using cross-validation**
  – At every run, one of the folds is used as *D.val* for validation, and the remaining two folds are used as *D.tr* for learning a model, for every choice of hyper-parameter value p_i
  – The overall validation error rate corresponding to each p_i is computed by summing the errors across all the three folds
  – Select the hyperparameter value p^* that provides the lowest validation error rate, and use it to learn a model m^* on the entire training set *D.train*

Presence of Hyper-parameters

- **Using cross-validation**

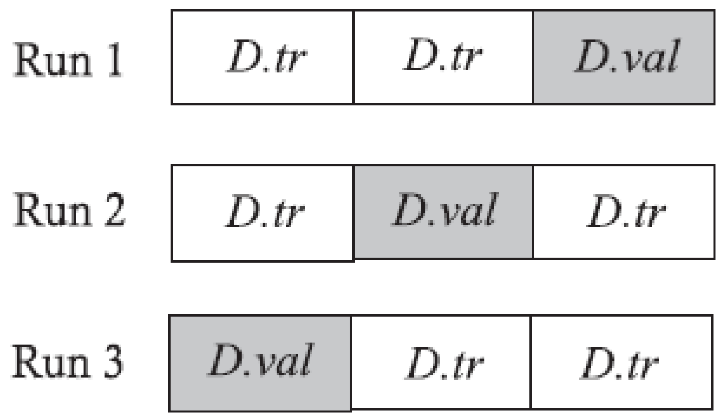
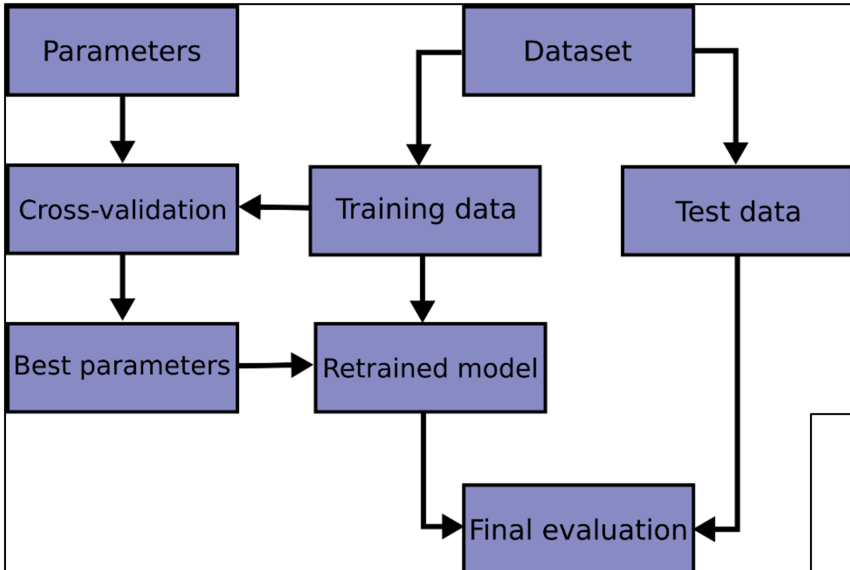
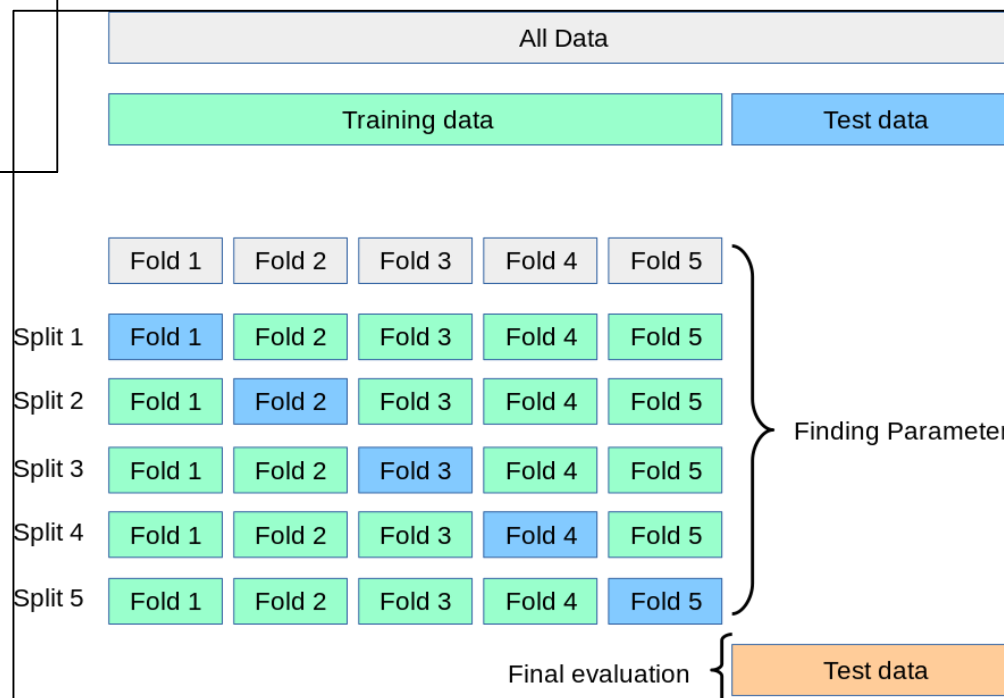


Figure 3.34. Example demonstrating the 3-fold cross-validation framework for hyper-parameter selection using $D.train$.

Model Selection and Evaluation



https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation



Model Selection and Evaluation

- Nested Cross-Validation

https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

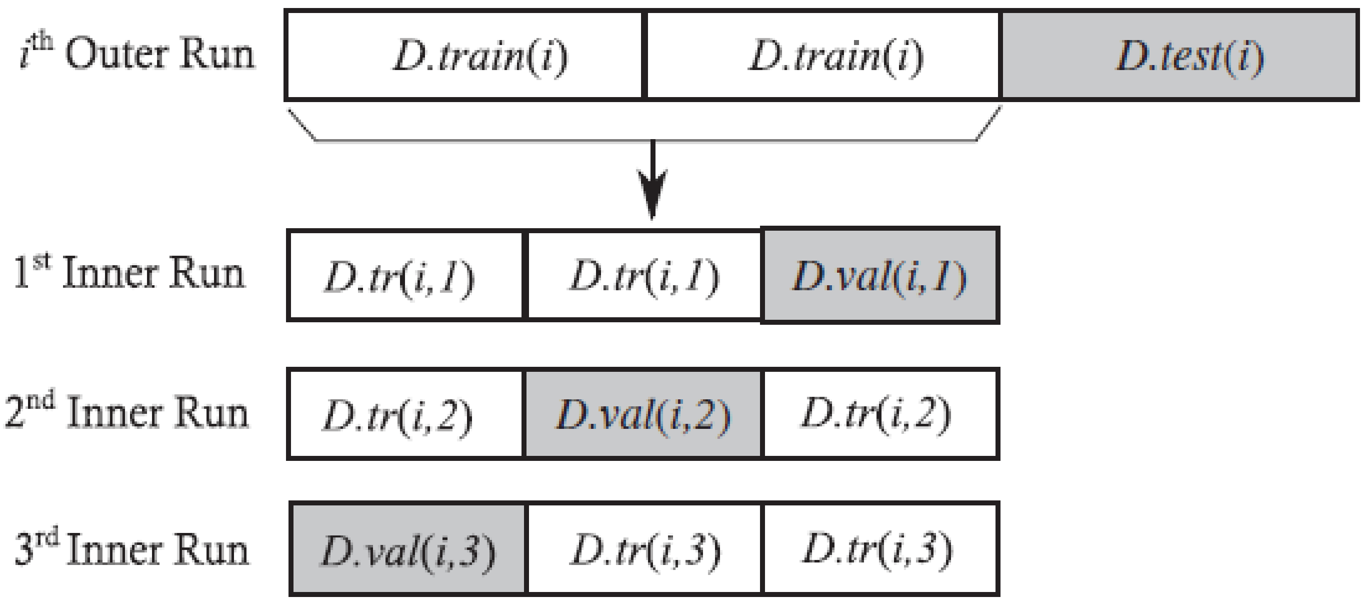


Figure 3.35. Example demonstrating 3-fold nested cross-validation for computing err_{test}.

Nested cross-validation when selecting classifiers is overzealous for most practical applications (2021) – https://www.sciencedirect.com/science/article/abs/pii/S0957417421006540

Measures of Classification Performance

https://en.wikipedia.org/wiki/Confusion_matrix

	PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	TP	FN
	No	FP	TN

α **is the probability that we reject the null hypothesis when it is true. This is a Type I error or a false positive (FP)**

β **is the probability that we accept the null hypothesis when it is false. This is a Type II error or a false negative (FN)**

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$ErrorRate = 1 - accuracy$$

$$Precision = Positive\ Predictive\ Value = \frac{TP}{TP + FP}$$

$$Recall = Sensitivity = TP\ Rate = \frac{TP}{TP + FN}$$

$$Specificity = TN\ Rate = \frac{TN}{TN + FP}$$

$$FP\ Rate = \alpha = \frac{FP}{TN + FP} = 1 - specificity$$

$$FN\ Rate = \beta = \frac{FN}{FN + TP} = 1 - sensitivity$$

$$Power = sensitivity = 1 - \beta$$

Measures of Classification Performance

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.556     0.6        0.625     0.556     0.588      0.633    yes
                 0.4       0.444      0.333     0.4       0.364      0.633    no
Weighted Avg.    0.5       0.544      0.521     0.5       0.508      0.633

=== Confusion Matrix ===

 a b   <-- classified as
 5 4 | a = yes
 3 2 | b = no
```

Measures of Classification Performance

$$Precision = Positive\ Predictive\ Value = \frac{TP}{TP + FP}$$

$$Recall = Sensitivity = TP\ Rate = \frac{TP}{TP + FN}$$

$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	TP	FN
	No	FP	TN

Recall

Precision

Precision evaluates the fraction of correct classified instances among the ones classified as positive

[A precision score of 1.0 for a class C means that every instance labelled as belonging to class C does indeed belong to class C (but says nothing about the number of instances from class C that were not labelled correctly]

Recall is the fraction of total positive instances correctly classified as positive

[A recall of 1.0 means that every instance from class C was labeled as belonging to class C (but says nothing about how many instances from other classes were incorrectly also labeled as belonging to class C]

F-measure analyzes the trade-offs between <u>correctness</u> and <u>coverage</u> in classifying positive instances

Quantos elementos selecionados são relevantes? *vs*

Quantos elementos relevantes foram selecionados?

Measures of Classification Performance

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.556     0.6        0.625     0.556     0.588      0.633    yes
                 0.4       0.444      0.333     0.4       0.364      0.633    no
Weighted Avg.    0.5       0.544      0.521     0.5       0.508      0.633

=== Confusion Matrix ===

 a b   <-- classified as
 5 4 | a = yes
 3 2 | b = no
```

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	TP	FN
	No	FP	TN

Precision or **Positive Predictive Value** (PPV) = TP/(TP + FP) = 5/(5 + 3) = 0.625 (yes)

Precision or **Positive Predictive Value** (PPV) = TP/(TP + FP) = 2/(2 + 4) = 0.333 (no)

9 (yes), 5 (no) = ((0.625*9) + (0.333*5))/14 = 0.521

Measures of Classification Performance

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall   F-Measure   ROC Area   Class
                 1         0.053     0.833      1         0.909      0.947      soft
                 0.75      0.1       0.6        0.75      0.667      0.813      hard
                 0.8       0.111     0.923      0.8       0.857      0.811      none
Weighted Avg.    0.833     0.097     0.851      0.833     0.836      0.84

=== Confusion Matrix ===

  a  b  c   <-- classified as
  5  0  0 |  a = soft
  0  3  1 |  b = hard
  1  2 12 |  c = none
```

		Previsto			FN
		soft	hard	none	
Classe Real	soft	5	0	0	0
	hard	0	3	1	1
	none	1	2	12	3
FP		1	2	1	4

Measures of Classification Performance

		Previsto			FN
		soft	hard	none	
Classe Real	soft	5	0	0	0
	hard	0	3	1	1
	none	1	2	12	3
FP		1	2	1	4

soft	
5	0
1	15

hard	
3	1
2	17

none	
12	3
1	8

Prec-global = 20/(20+4) = 20/24 = 0.833 (**Micro**)

$$Precision = Positive\ Predictive\ Value = \frac{TP}{TP + FP}$$

Prec-soft: 5/(5+1) = 0.833
Prec-hard: 3/(3+2) = 0.6
Prec-none: 12/(12+1) = 0.923

((0.833 * 5) + (0.6 * 4) + (0.923 * 15))/24 = 0.850

Macro (0.785) OR
Macro Weighted Average (0.850)

ROC (Receiver Operating Characteristic)

- Let’s consider a classifier that gives a numeric score for an instance to be classified in the positive class
  - Instances with a higher score are more likely to have to be classified as positive
- Almost all classifiers generate positive or negative predictions by applying a threshold to a score

ROC (Receiver Operating Characteristic)

- The choice of this threshold will have an impact in the trade-offs of positive and negative errors
  - A higher threshold will reduce the FPR, as less instances will be classified as positive. However, the FNR would increase, as we are very restrictive in classifying an instance as positive
  - A lower threshold will reduce the FNR, as more instances are classified as positive. However, a larger FPR is expected, as we are more lenient in classifying instances as positive

ROC (Receiver Operating Characteristic)

- For evaluating a scoring classifier, we may choose an arbitrary threshold and use the metrics discussed before
  - However, we throw away the granularity given by the scores, as we cannot differentiate between more or less likely instances within each class, and we are committed to this arbitrary threshold
- To avoid these drawbacks, it would be interesting to evaluate the scoring classifications without having to select a specific threshold

ROC (Receiver Operating Characteristic)

- ROC curve is a graphical approach for displaying the tradeoff between true positive rate and false positive rate of a classifier
- ROC curve plots TPR (*y*-axis) against FPR (*x*-axis)
  - Performance of a model represented as a point in an ROC curve
  - Changing the threshold parameter of classifier changes the location of the point

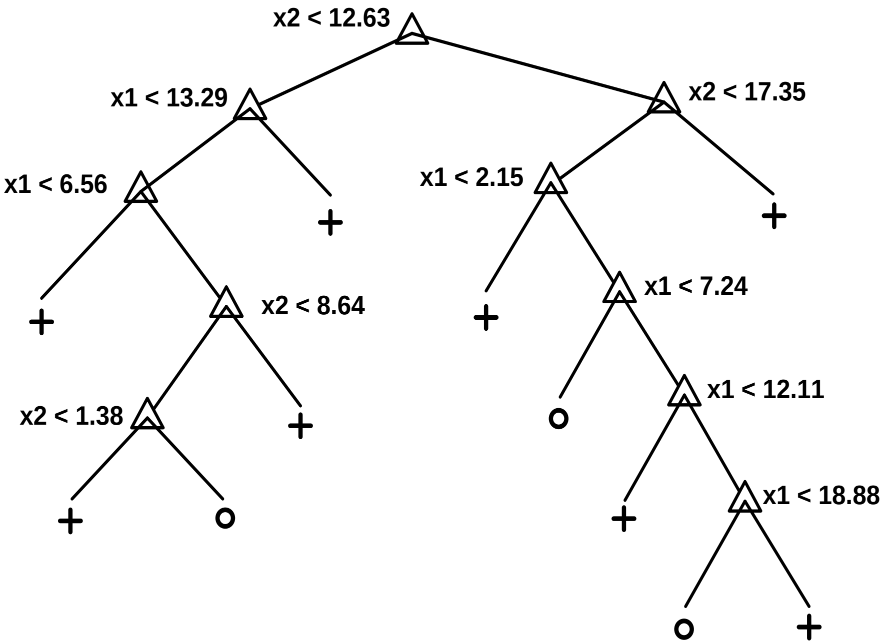
ROC (Receiver Operating Characteristic)

- To draw ROC curve, classifier must produce continuous-valued output
  - Outputs are used to rank test records, from the most likely positive class record to the least likely positive class record

- Many classifiers produce only discrete outputs (i.e., predicted class)
  - How to get continuous-valued outputs?
    - Decision trees, rule-based classifiers, k-nearest neighbors, …

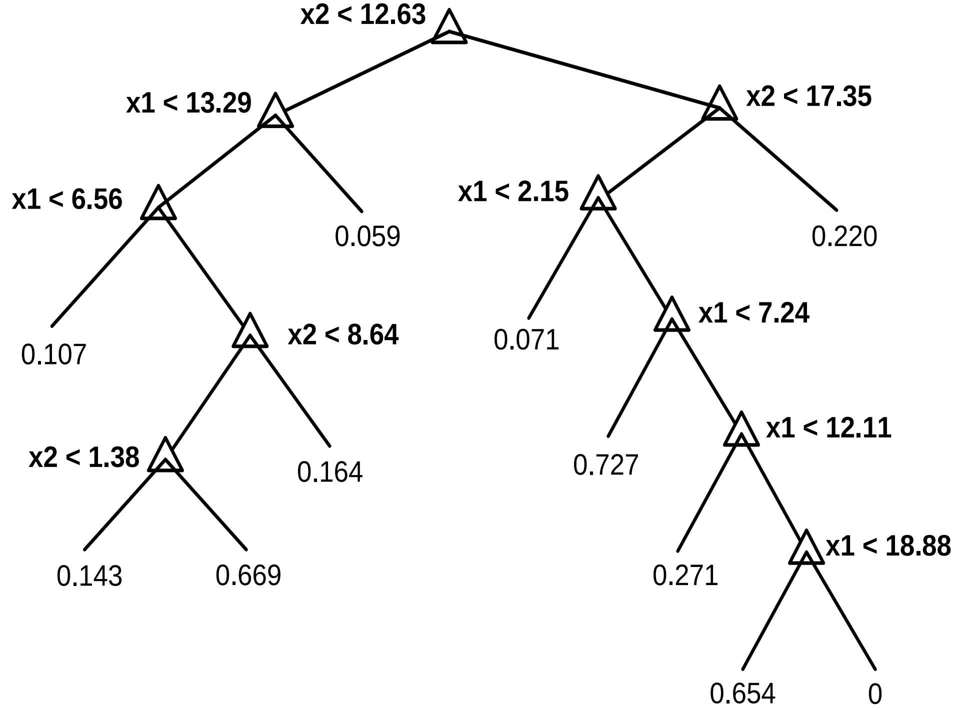
Example: Decision Trees

Decision Tree



Suppose that you have two classes T and F, and in your leaf node you have 10 instances with T and 5 instances with F, you can return a vector of scores: (scoreT, scoreF)=(10/15, 5/15)=(0.66, 0.33)

Continuous-valued outputs



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
- Apply a threshold at each unique value of the score
- Count the number of TP, FP, TN, FN at each threshold
  - TPR = TP/(TP+FN) = TP/Pos
  - FPR = FP/(FP + TN) = FP/Neg

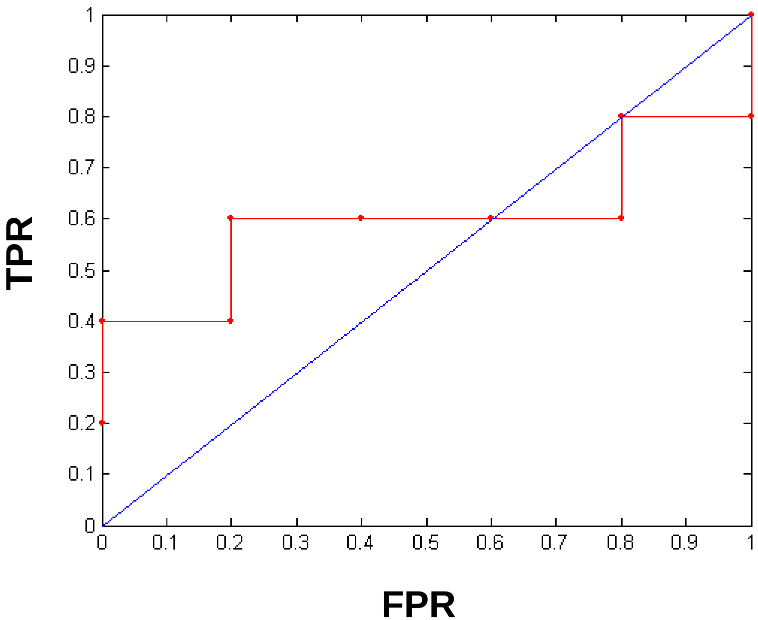
TPR = Recall = Sensibilidade = porcentagem de amostras corretamente classificadas como positivas dentre todas as positivas reais

FPR = 1 – Especificidade (TNR) = porcentagem de amostras erroneamente classificadas como positivas dentre todas as negativas reais

How to construct an ROC curve

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

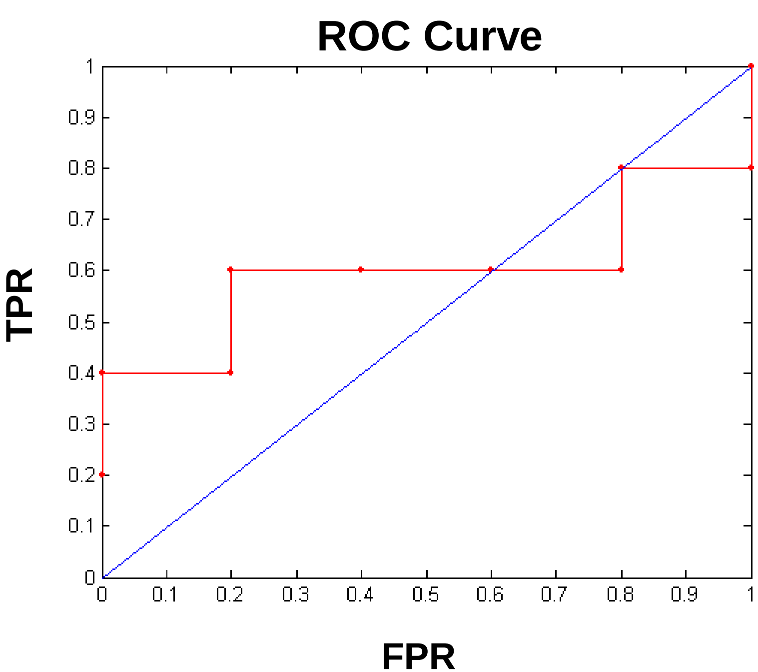
ROC Curve



	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	+

How to construct an ROC curve

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



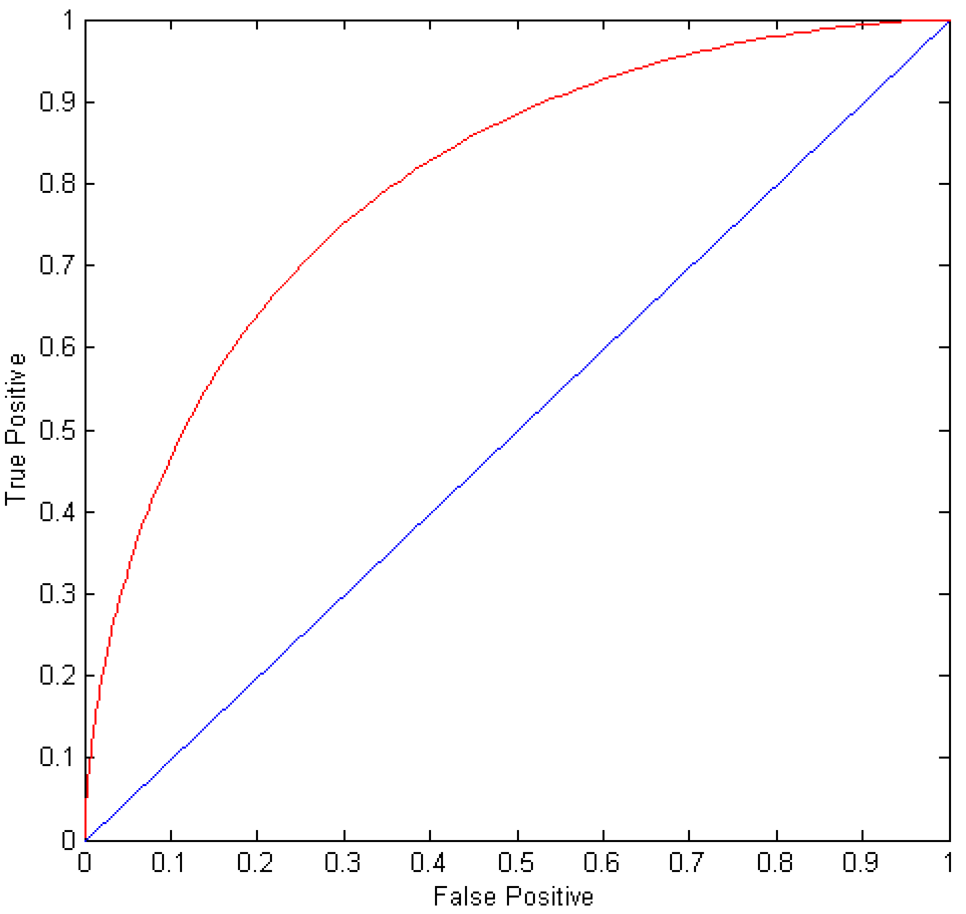
	Instance	Score	True Class
+	1	0.95	+
+	2	0.93	+
-	3	0.87	-
-	4	0.85	-
-	5	0.85	-
-	6	0.85	+
-	7	0.76	-
-	8	0.53	+
-	9	0.43	-
-	10	0.25	+

ROC Curve

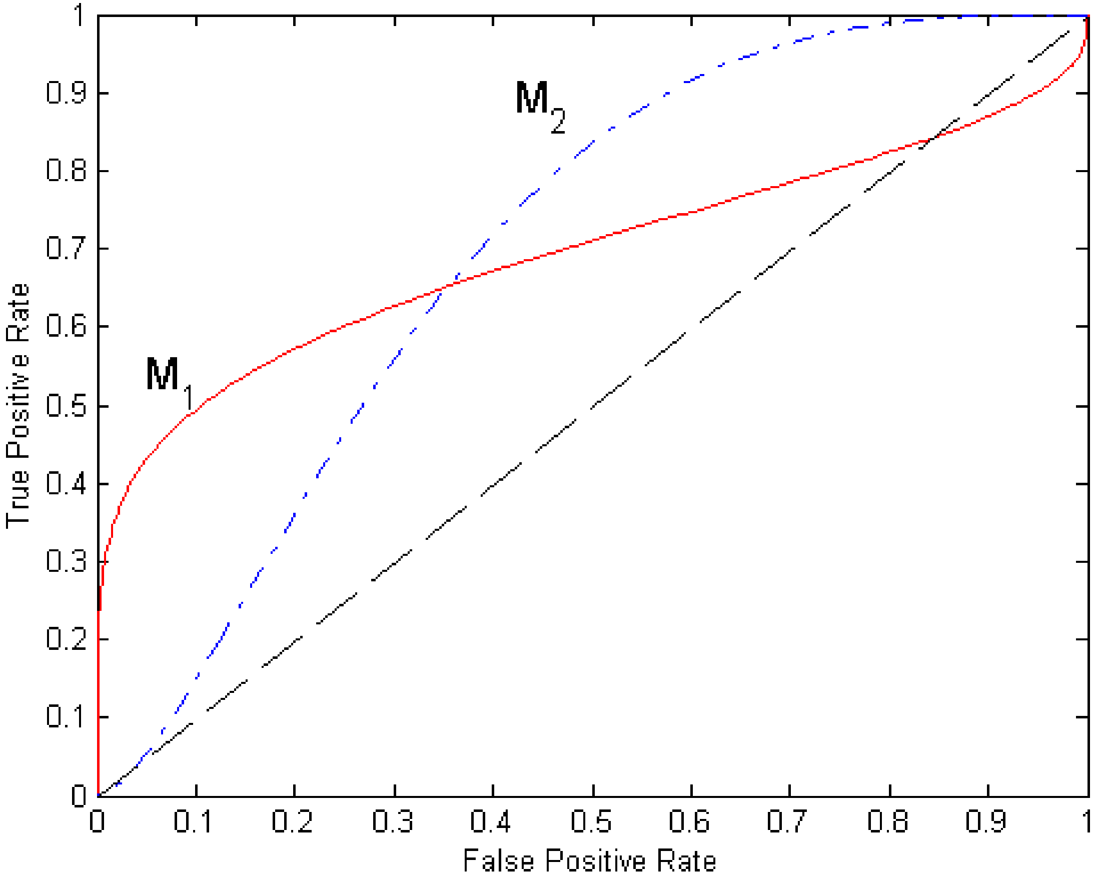
(TPR,FPR):

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

- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class



Using ROC for Model Comparison



- No model consistently outperform the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess:
    - Area = 0.5