

DATA MINING

LECTURE 10

Classification

Basic Concepts

Decision Trees

Evaluation

Nearest-Neighbor Classifier

Catching tax-evasion

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

Tax-return data for year 2011

A new tax return for 2012
Is this a cheating tax return?

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

An instance of the classification problem: learn a method for discriminating between records of different **classes** (**cheaters** vs **non-cheaters**)

What is classification?

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

categorical
categorical
continuous
class

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

One of the attributes is the **class attribute**
In this case: Cheat

Two **class labels** (or **classes**): **Yes (1)**, **No (0)**

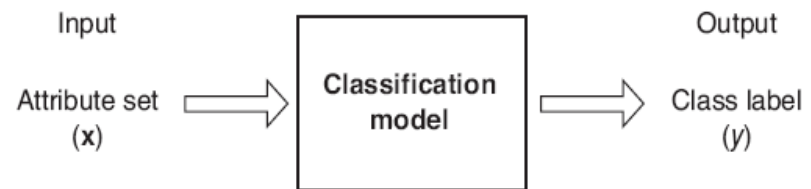


Figure 4.2. Classification as the task of mapping an input attribute set x into its class label y .

Why classification?

- The target function f is known as a **classification model**
- **Descriptive modeling:** **Explanatory tool** to distinguish between objects of different classes (e.g., understand why people cheat on their taxes)
- **Predictive modeling:** Predict a class of a **previously unseen** record

Examples of Classification Tasks

- Predicting tumor cells as benign or malignant
- Classifying credit card transactions as legitimate or fraudulent
- Categorizing news stories as finance, weather, entertainment, sports, etc
- Identifying spam email, spam web pages, adult content
- Understanding if a web query has commercial intent or not

General approach to classification

- **Training set** consists of records with **known class labels**
- Training set is used to **build** a classification model
- A **labeled test set** of **previously unseen** data records is used to **evaluate** the quality of the model.
- The classification model is **applied** to new records with **unknown class labels**

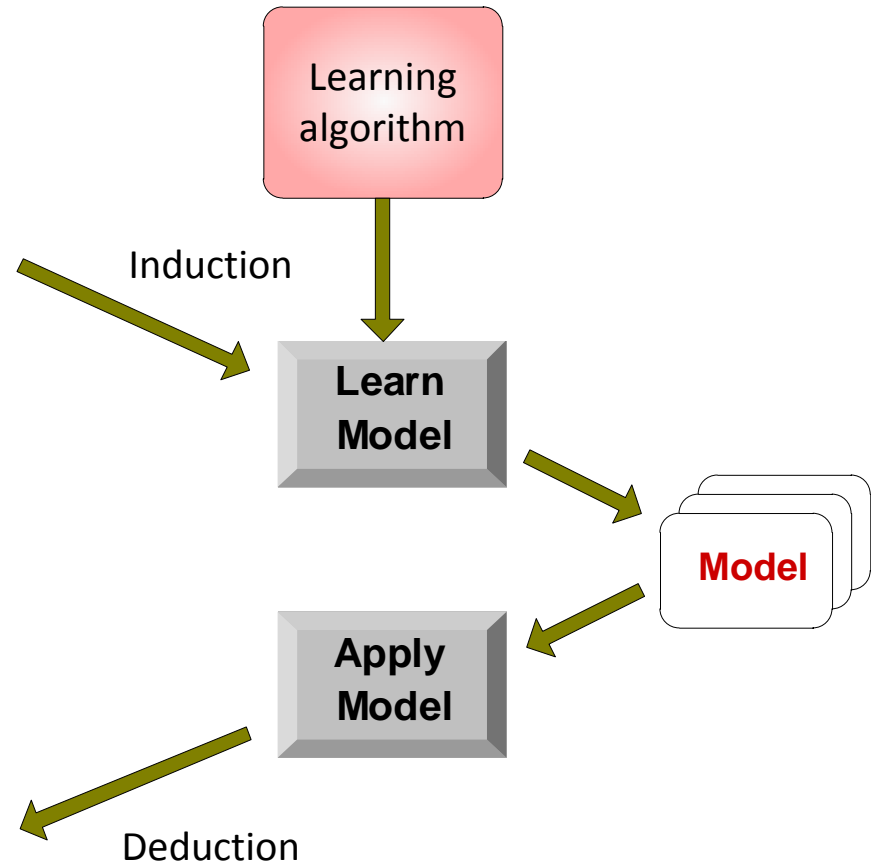
Illustrating Classification Task

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



Evaluation of classification models

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

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

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

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

Classification Techniques

- Decision Tree based Methods
- Rule-based Methods
- Memory based reasoning
- Neural Networks
- Naïve Bayes and Bayesian Belief Networks
- Support Vector Machines

Classification Techniques

- Decision Tree based Methods
- Rule-based Methods
- Memory based reasoning
- Neural Networks
- Naïve Bayes and Bayesian Belief Networks
- Support Vector Machines

Decision Trees

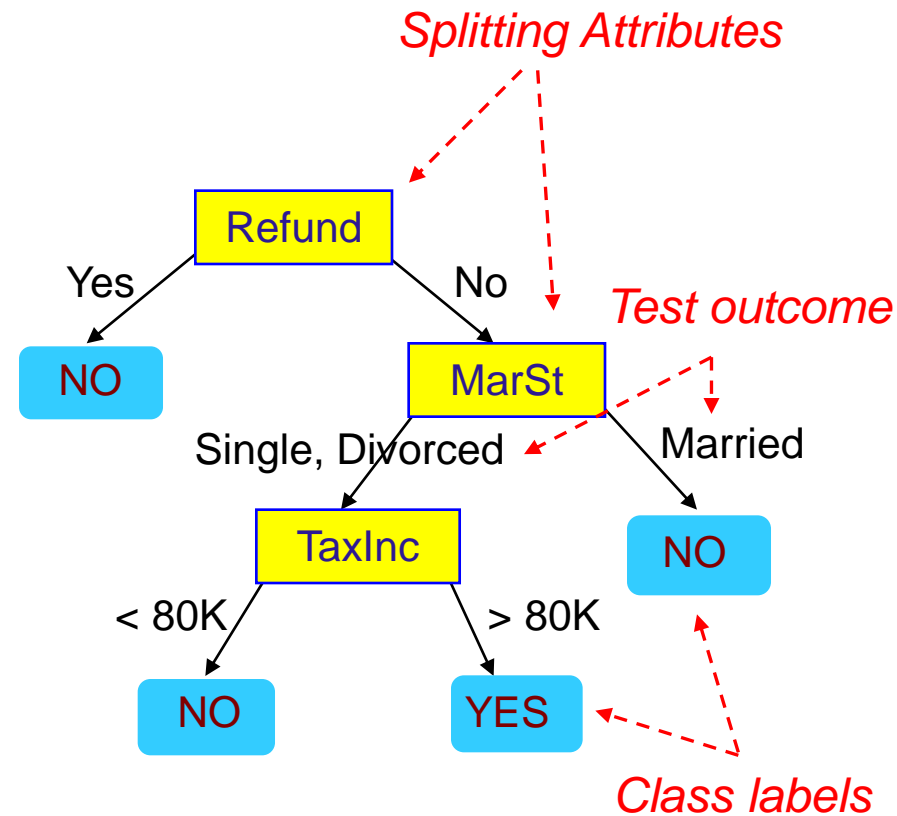
- Decision tree
 - A **flow-chart-like tree** structure
 - **Internal node** denotes a **test on an attribute**
 - **Branch** represents an **outcome of the test**
 - **Leaf nodes** represent **class labels** or class distribution

Example of a Decision Tree

categorical
categorical
continuous
class

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

Training Data

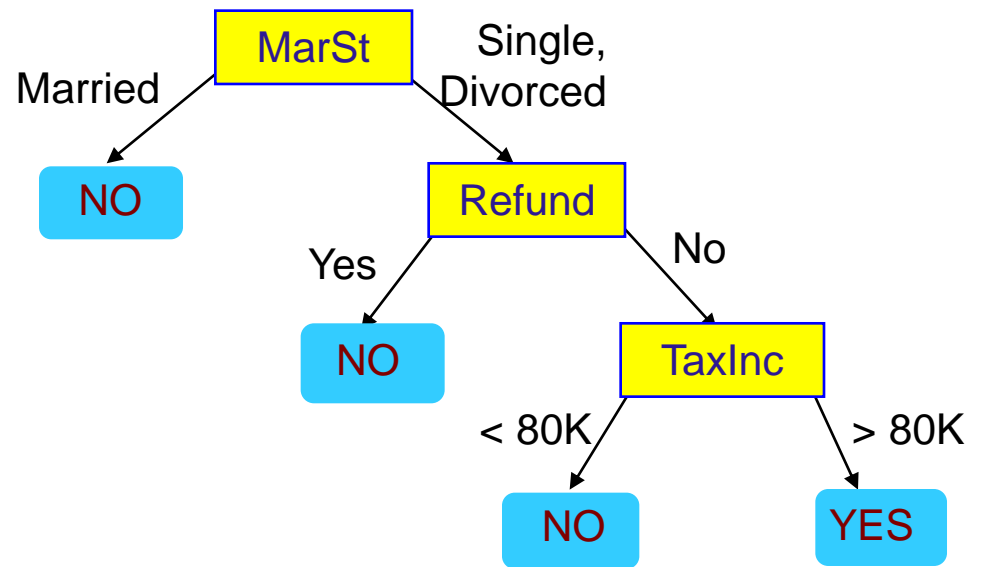


Model: Decision Tree

Another Example of Decision Tree

categorical
categorical
continuous
class

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



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

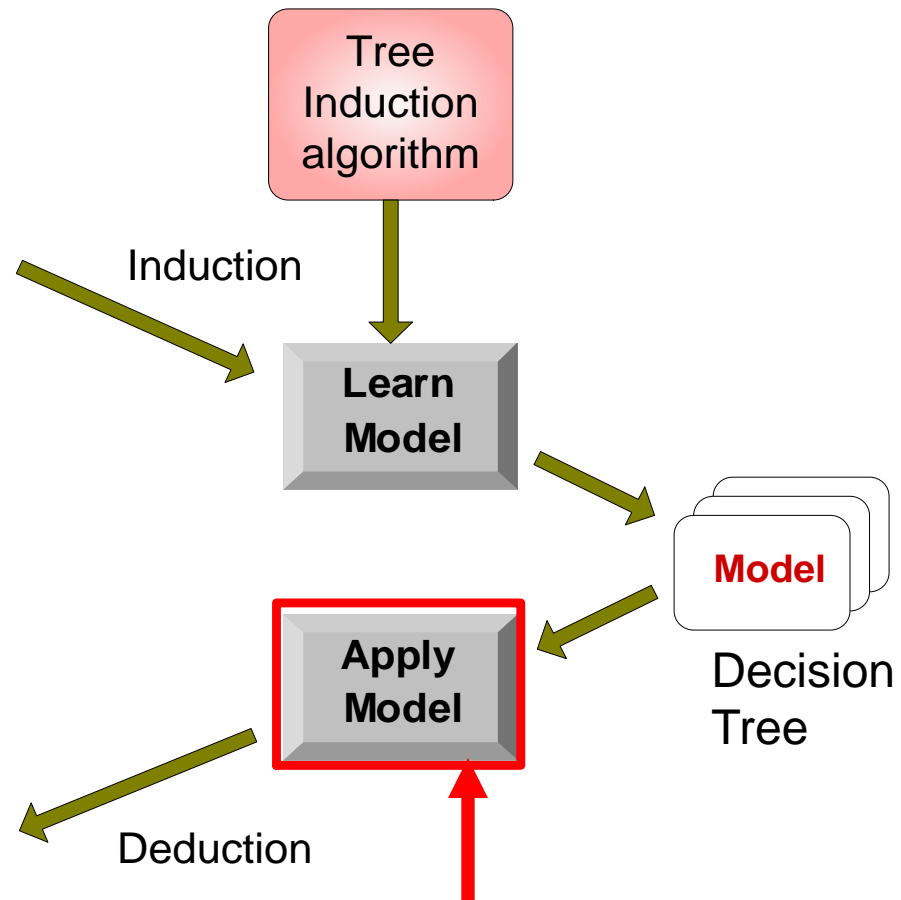
Decision Tree Classification Task

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

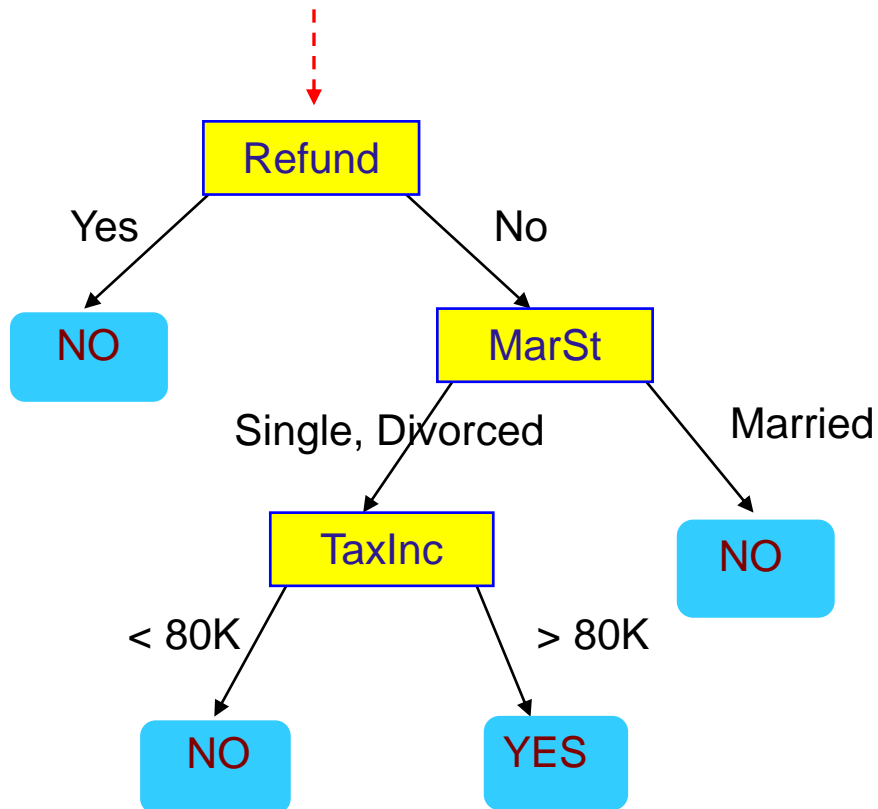
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Apply Model to Test Data

Start from the root of tree.



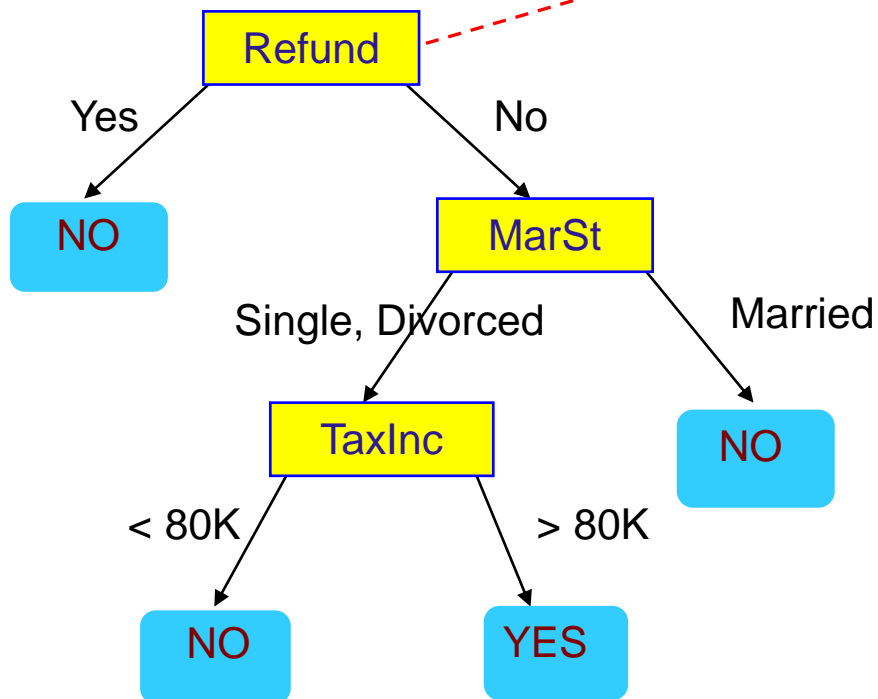
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

Test Data

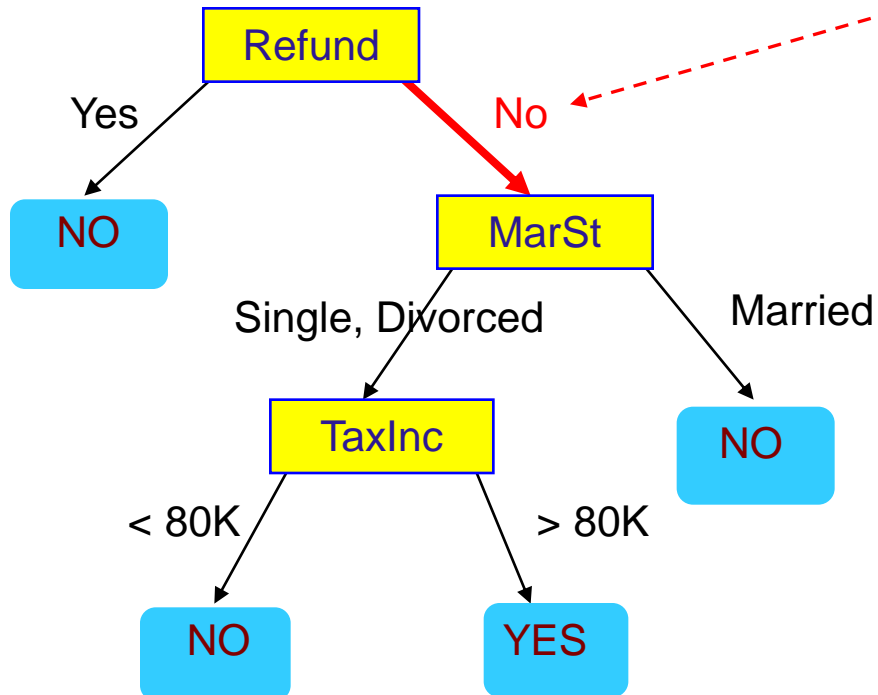
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

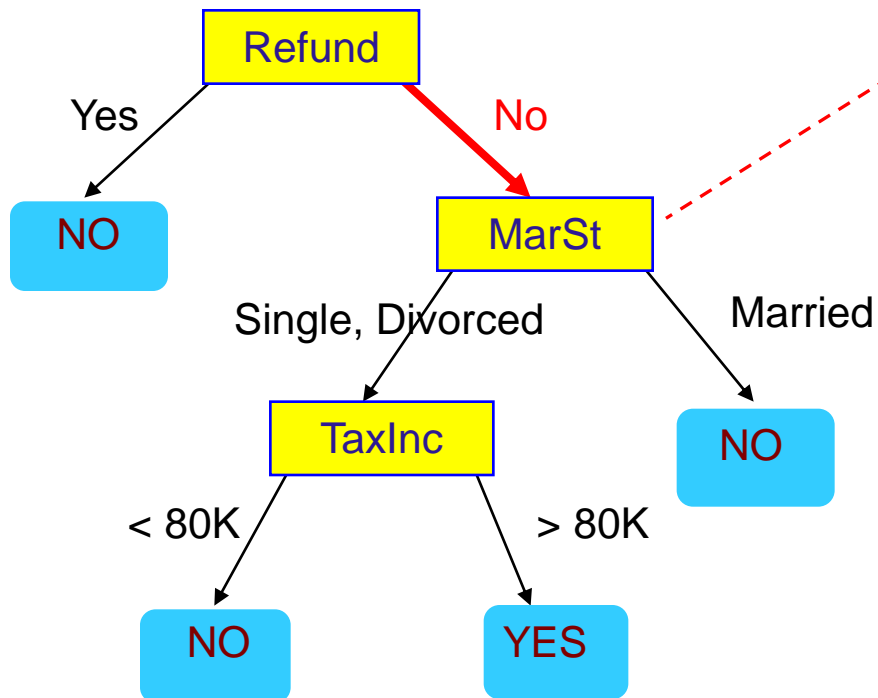
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

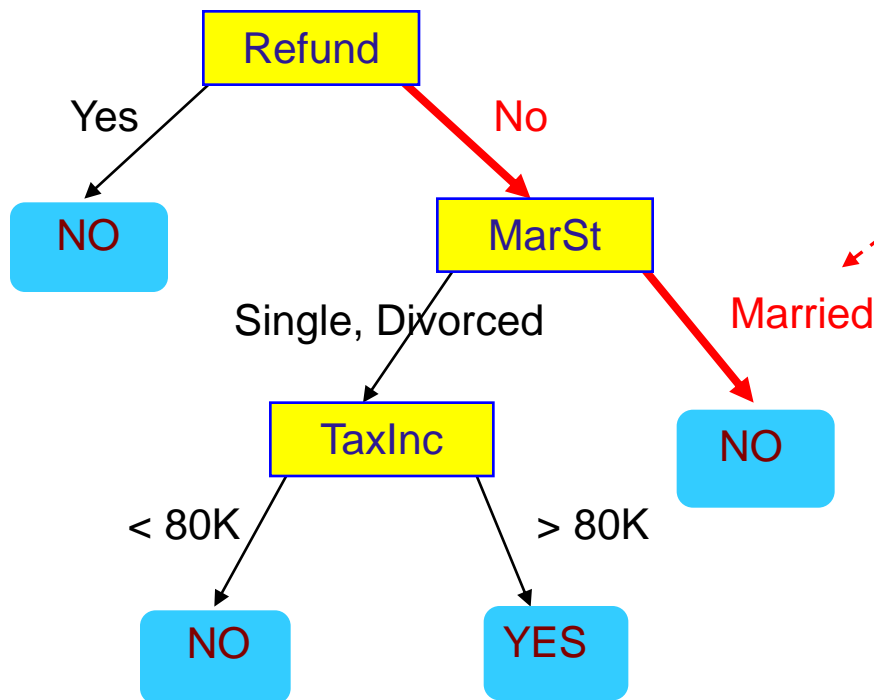
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

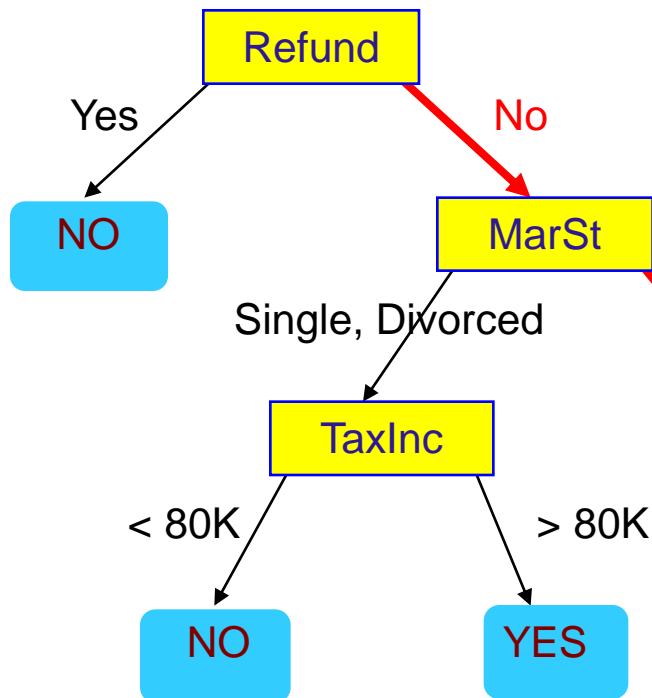
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Assign Cheat to "No"

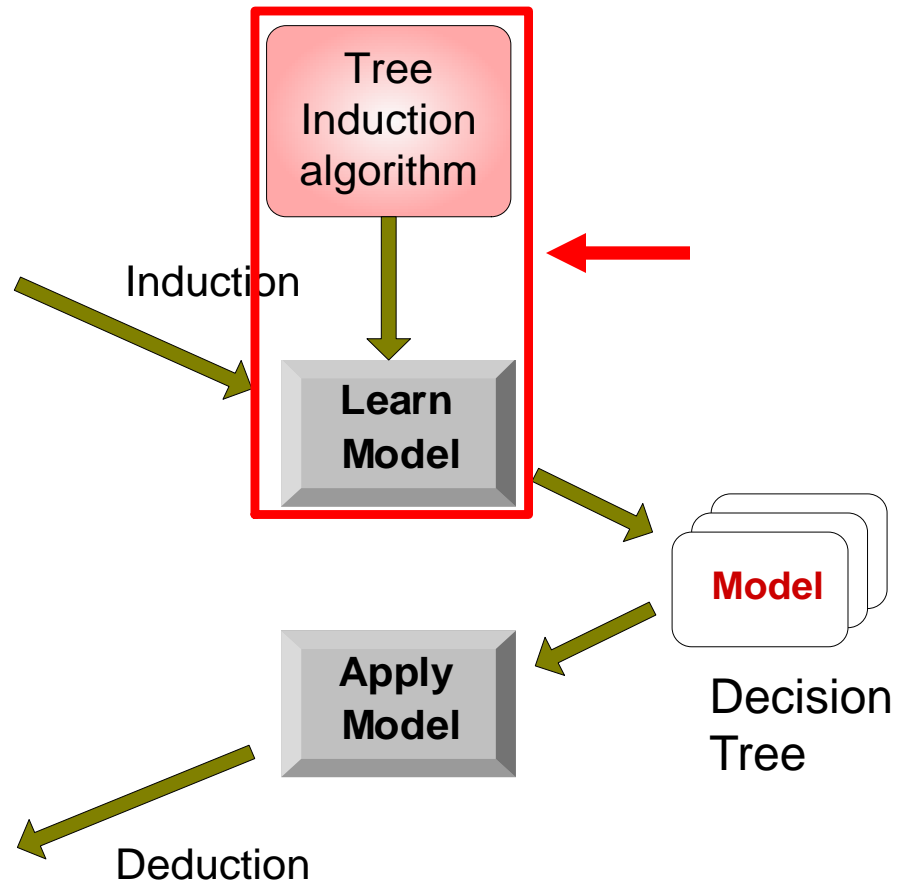
Decision Tree Classification Task

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



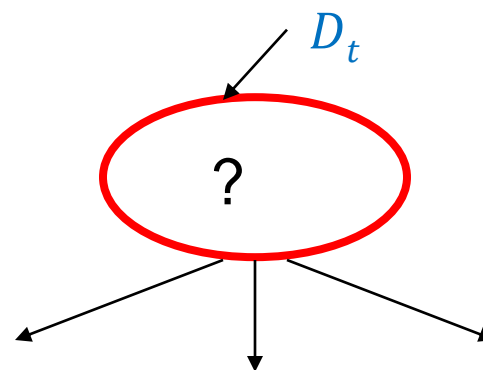
Tree Induction

- Finding the best decision tree is **NP-hard**
- **Greedy** strategy.
 - Split the records based on an attribute test that optimizes **certain criterion**.
- Many Algorithms:
 - Hunt's Algorithm (one of the earliest)
 - CART
 - ID3, C4.5
 - SLIQ, SPRINT

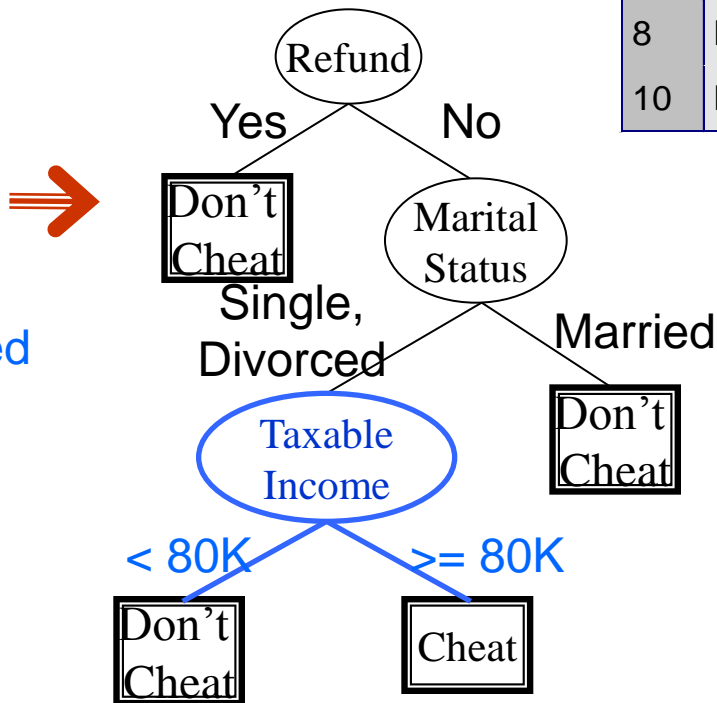
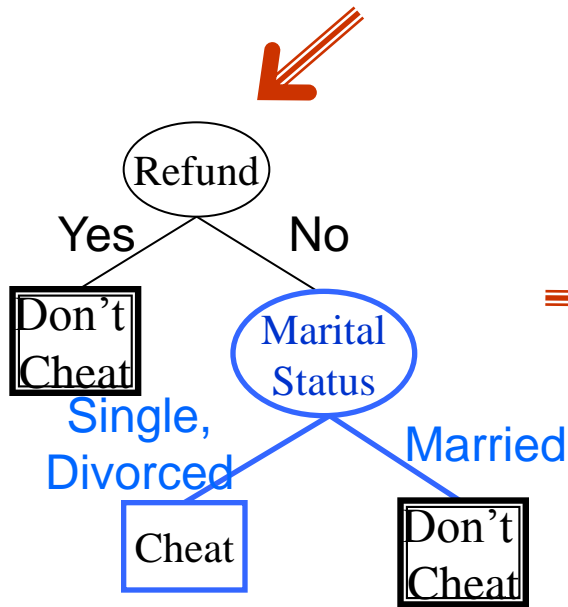
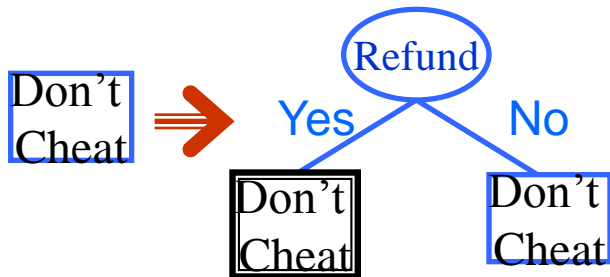
General Structure of Hunt's Algorithm

- 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 with the **same attribute values**, then t is a leaf node labeled with the **majority class** y_t
 - If D_t is an **empty set**, then t is a leaf node labeled by the **default class**, y_d
 - 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.

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



Hunt's Algorithm



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

Constructing decision-trees (pseudocode)

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

1. If **stopping_condition**(**S**,**F**) = true then
 - a. leaf = **createNode**()
 - b. leaf.label = **Classify**(**S**)
 - c. return leaf
2. root = **createNode**()
3. root.test_condition = **findBestSplit**(**S**,**F**)
4. **V** = {**v** | **v** a possible outcome of root.test_condition}
5. for each value **v** ∈ **V**:
 - a. **S_v** := {**s** | root.test_condition(**s**) = **v** and **s** ∈ **S**};
 - b. child = **GenDecTree**(**S_v**, **F**) ;
 - c. Add child as a descent of root and label the edge (root → child) as **v**
6. return root

Tree Induction

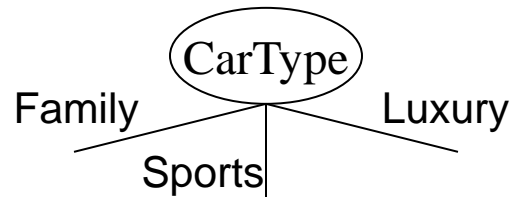
- Issues
 - How to **Classify** a leaf node
 - Assign the **majority class**
 - If leaf is empty, assign the **default class** – the class that has the highest popularity.
 - Determine how to split the records
 - **How to specify the attribute test condition?**
 - **How to determine the best split?**
 - Determine when to stop splitting

How to Specify Test Condition?

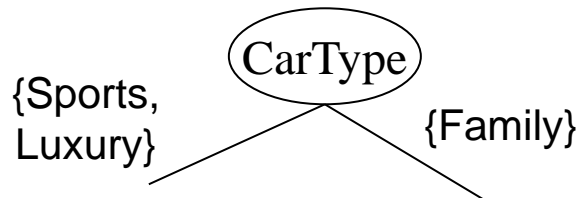
- Depends on attribute types
 - Nominal
 - Ordinal
 - Continuous
- Depends on number of ways to split
 - 2-way split
 - Multi-way split

Splitting Based on Nominal Attributes

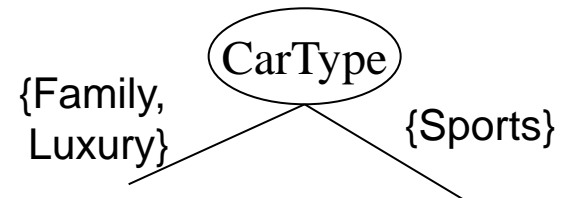
- **Multi-way split:** Use as many partitions as distinct values.



- **Binary split:** Divides values into two subsets. Need to find optimal partitioning.

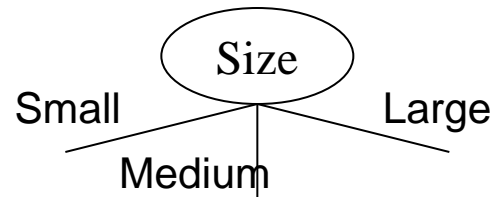


OR

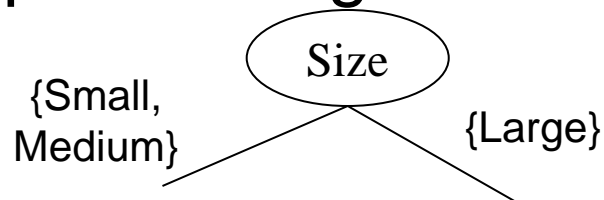


Splitting Based on Ordinal Attributes

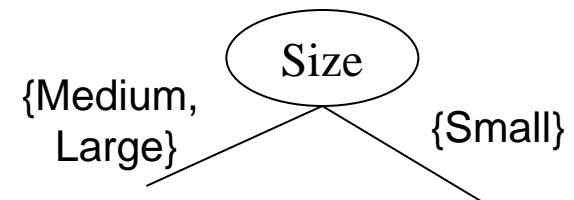
- **Multi-way split:** Use as many partitions as distinct values.



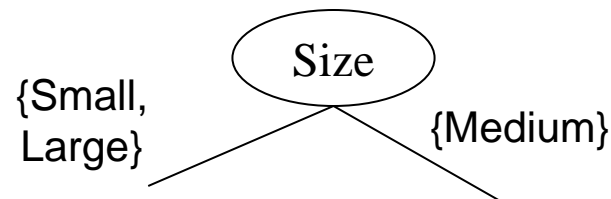
- **Binary split:** Divides values into two subsets – respects the order. Need to find optimal partitioning.



OR



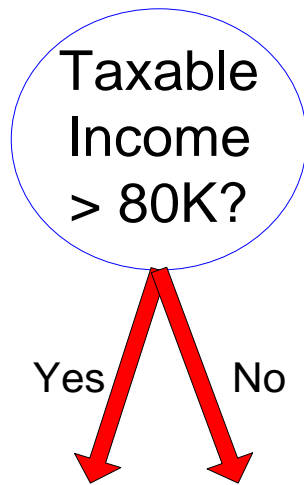
- What about this split?



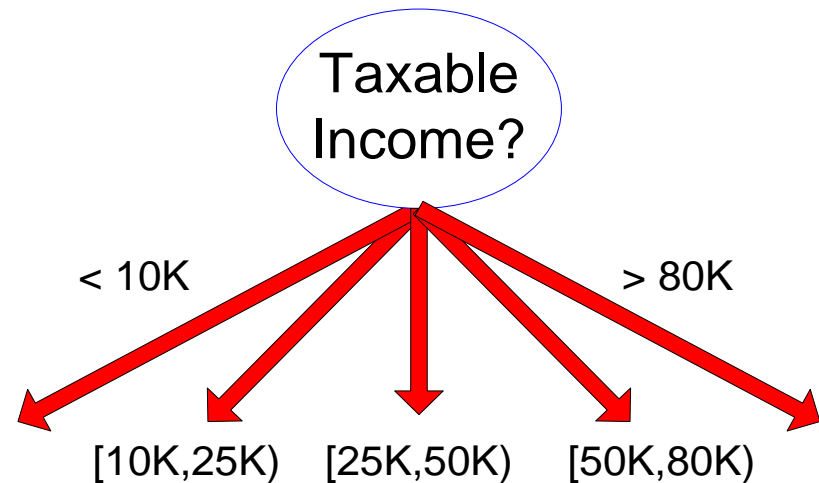
Splitting Based on Continuous Attributes

- Different ways of handling
 - **Discretization** to form an **ordinal** categorical attribute
 - **Static** – discretize once at the beginning
 - **Dynamic** – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
 - **Binary Decision**: $(A < v)$ or $(A \geq v)$
 - consider all possible splits and finds the best cut
 - can be more compute intensive

Splitting Based on Continuous Attributes



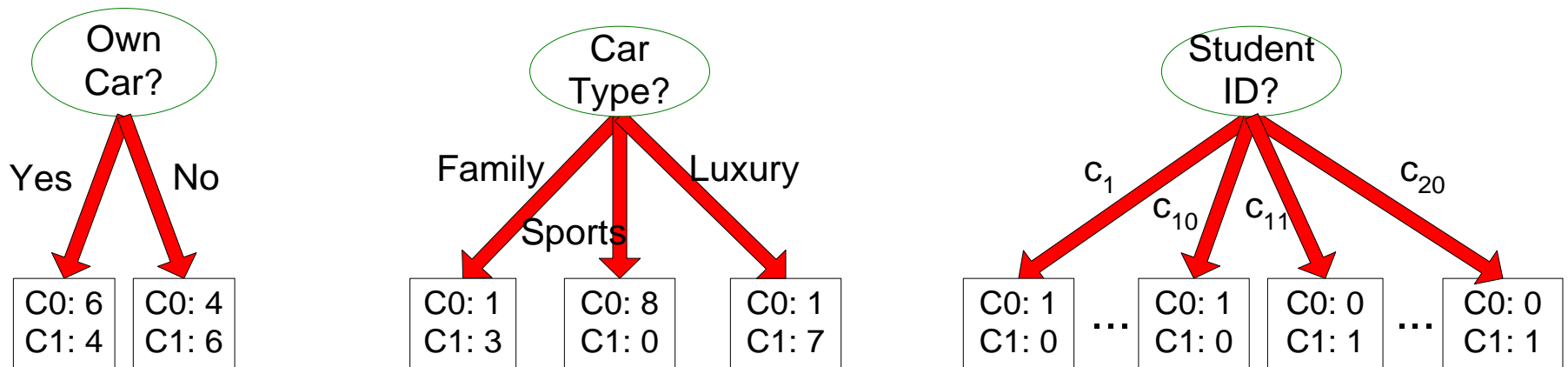
(i) Binary split



(ii) Multi-way split

How to determine the Best Split

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:
 - Creation of nodes with **homogeneous** class distribution is preferred
- Need a measure of node **impurity**:

C0: 5
C1: 5

Non-homogeneous,
High degree of impurity

C0: 9
C1: 1

Homogeneous,
Low degree of impurity

- Ideas?

Measuring Node Impurity

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

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

- Used in ID3 and C4.5

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

- Used in CART, SLIQ, SPRINT.

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

Gain

- **Gain of an attribute split:** compare the impurity of the parent node with the average impurity of the child nodes

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

- **Maximizing** the **gain** \Leftrightarrow **Minimizing** the weighted average **impurity** measure of children nodes
- If **I() = Entropy()**, then Δ_{info} is called **information gain**

Example

C1	0
C2	6

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$

$$\text{Gini} = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$$

$$\text{Entropy} = -0 \log 0 - 1 \log 1 = -0 - 0 = 0$$

$$\text{Error} = 1 - \max(0, 1) = 1 - 1 = 0$$

C1	1
C2	5

$$P(C1) = 1/6 \quad P(C2) = 5/6$$

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

$$\text{Entropy} = - (1/6) \log_2 (1/6) - (5/6) \log_2 (5/6) = 0.65$$

$$\text{Error} = 1 - \max(1/6, 5/6) = 1 - 5/6 = 1/6$$

C1	2
C2	4

$$P(C1) = 2/6 \quad P(C2) = 4/6$$

$$\text{Gini} = 1 - (2/6)^2 - (4/6)^2 = 0.444$$

$$\text{Entropy} = - (2/6) \log_2 (2/6) - (4/6) \log_2 (4/6) = 0.92$$

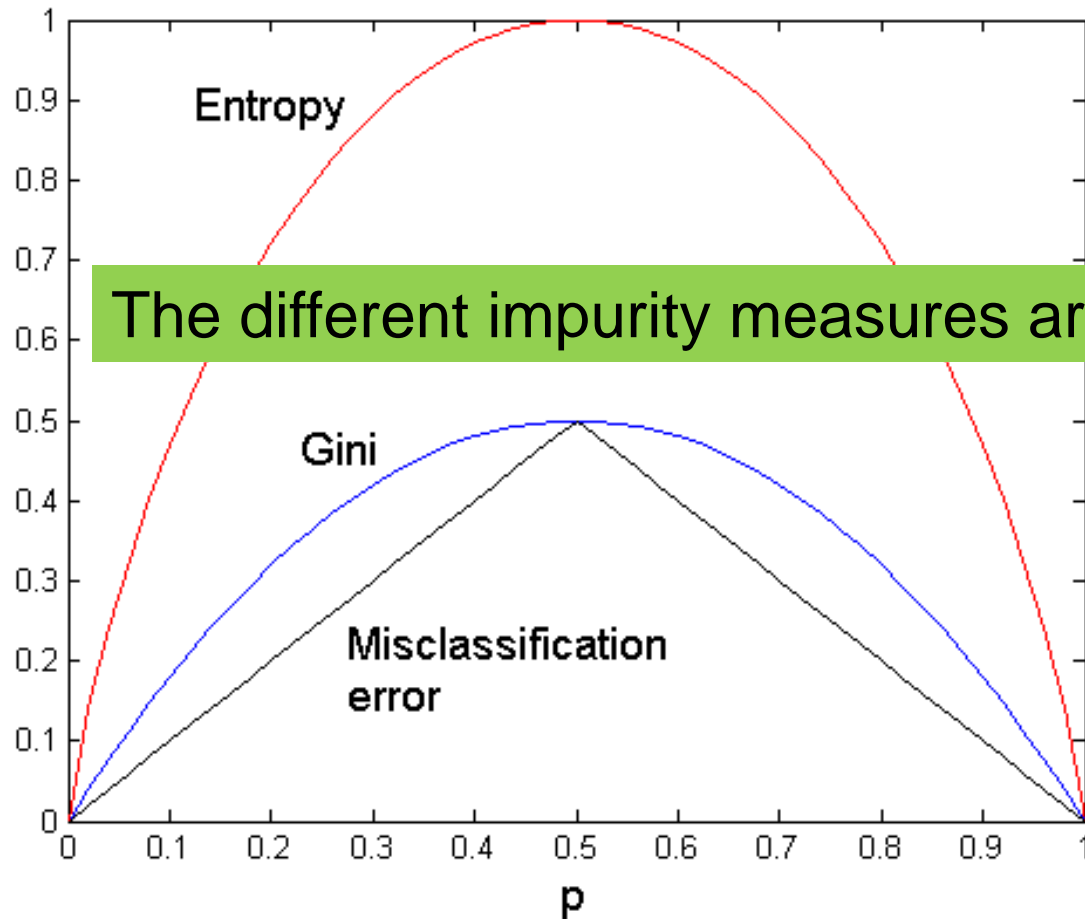
$$\text{Error} = 1 - \max(2/6, 4/6) = 1 - 4/6 = 1/3$$

Impurity measures

- All of the impurity measures take value zero (**minimum**) for the case of a pure node where a single value has probability 1
- All of the impurity measures take **maximum** value when the class distribution in a node is **uniform**.

Comparison among Splitting Criteria

For a 2-class problem:



The different impurity measures are **consistent**

Categorical Attributes

- For **binary** values split in two
- For **multivalued** attributes, 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	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

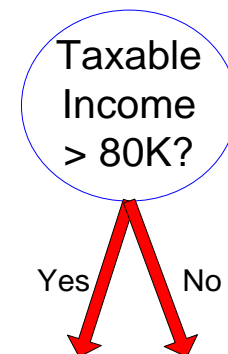
	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Continuous Attributes

- Use Binary Decisions based on one value
- 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 $A \geq v$
- **Exhaustive** method to choose best v
 - For each v , scan the database to gather count matrix and compute the impurity index
 - Computationally Inefficient! Repetition of work.

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



Continuous Attributes

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

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No		
Taxable 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	0	3	0	
No	0	7	1	6	2	5	3	4	3	4	3	4
Gini	0.420	0.400	0.375	0.343	0.417	0.400	<u>0.300</u>	0.343	0.375	0.400	0.420	

Splitting based on impurity

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

Splitting based on INFO

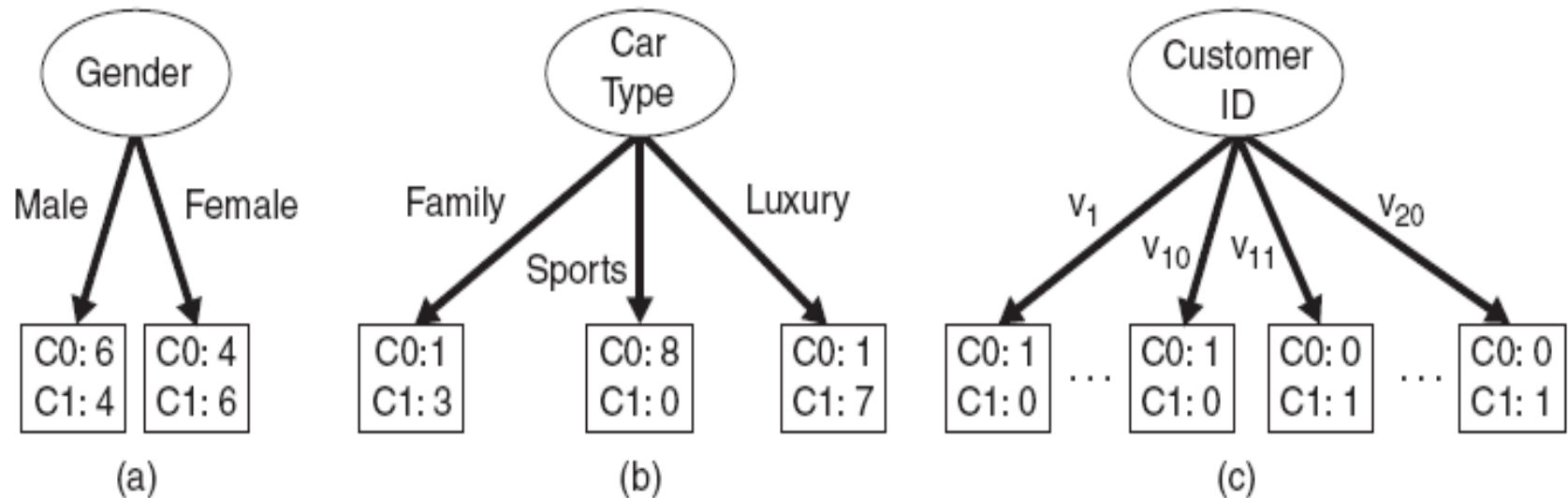


Figure 4.12. Multiway versus binary splits.

Gain Ratio

- Splitting using information gain

$$\textit{GainRATIO}_{split} = \frac{\textit{GAIN}_{Split}}{\textit{SplitINFO}} \quad \textit{SplitINFO} = -\sum_{i=1}^k \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions

n_i is the number of records in partition i

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

Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class
- Stop expanding a node when all the records have similar attribute values
- **Early termination** (to be discussed later)

Decision Tree Based Classification

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

Example: C4.5

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

Other Issues

- Data Fragmentation
- Expressiveness

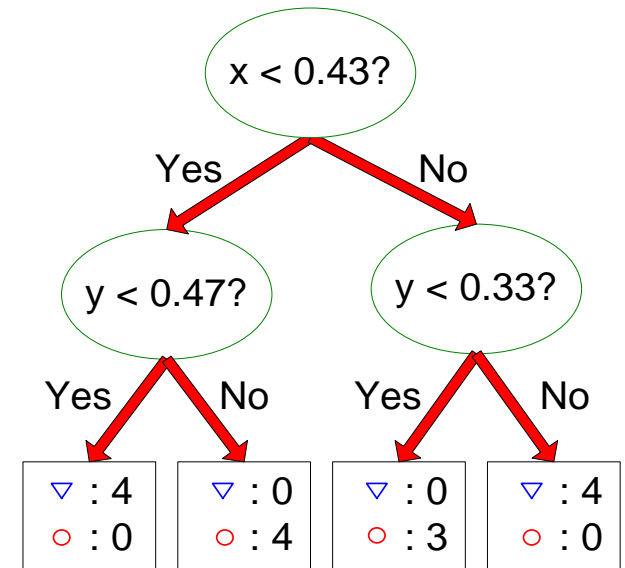
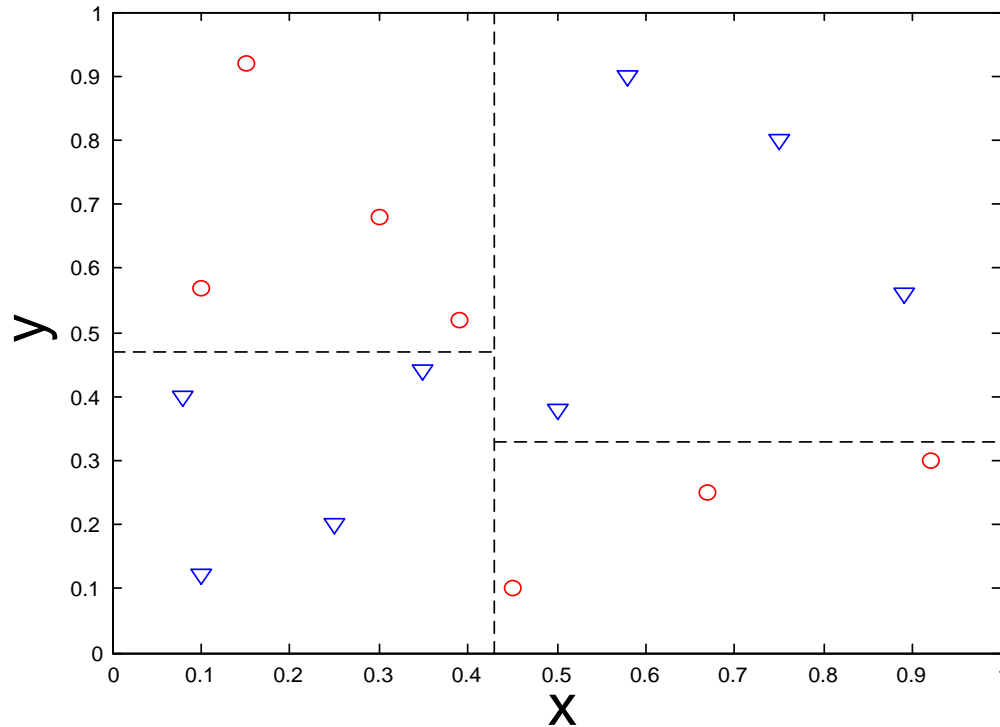
Data Fragmentation

- Number of instances gets smaller as you traverse down the tree
- Number of instances at the leaf nodes could be too small to make any statistically significant decision
- You can introduce a lower bound on the number of items per leaf node in the stopping criterion.

Expressiveness

- A classifier defines a **function** that discriminates between two (or more) classes.
- The **expressiveness** of a classifier is the **class of functions** that it can model, and the kind of data that it can **separate**
 - When we have **discrete** (or binary) values, we are interested in the class of **boolean functions** that can be modeled
 - If the data-points are real vectors we talk about the **decision boundary** that the classifier can model

Decision Boundary

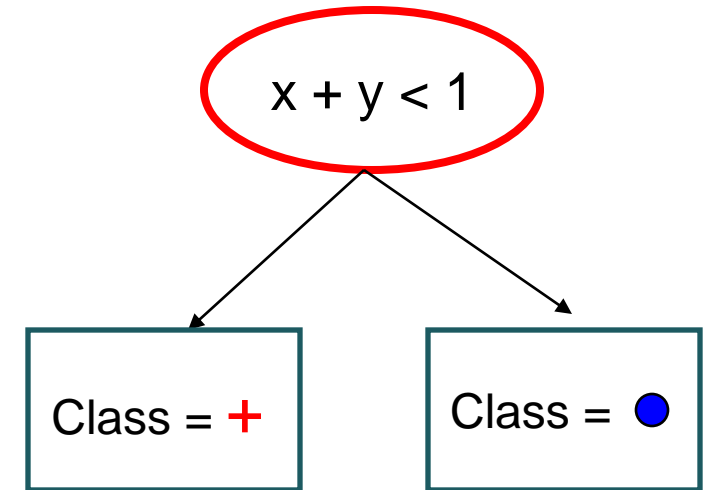
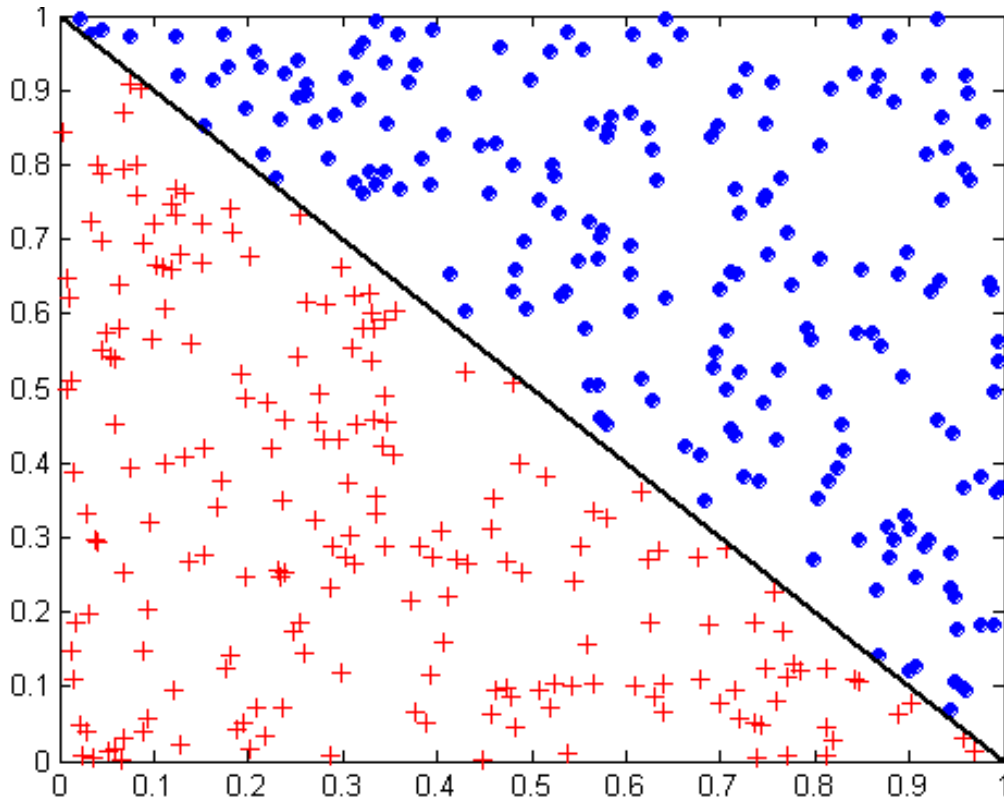


- Border line between two neighboring regions of different classes is known as **decision boundary**
- Decision boundary is **parallel to axes** because test condition involves a single attribute at-a-time

Expressiveness

- Decision tree provides **expressive** representation for learning discrete-valued function
 - But they do not generalize well to certain types of Boolean functions
 - Example: **parity function**:
 - Class = 1 if there is an **even** number of Boolean attributes with truth value = True
 - Class = 0 if there is an **odd** number of Boolean attributes with truth value = True
 - For accurate modeling, must have a complete tree
- Less expressive for modeling continuous variables
 - Particularly when test condition involves only a single attribute at-a-time

Oblique Decision Trees

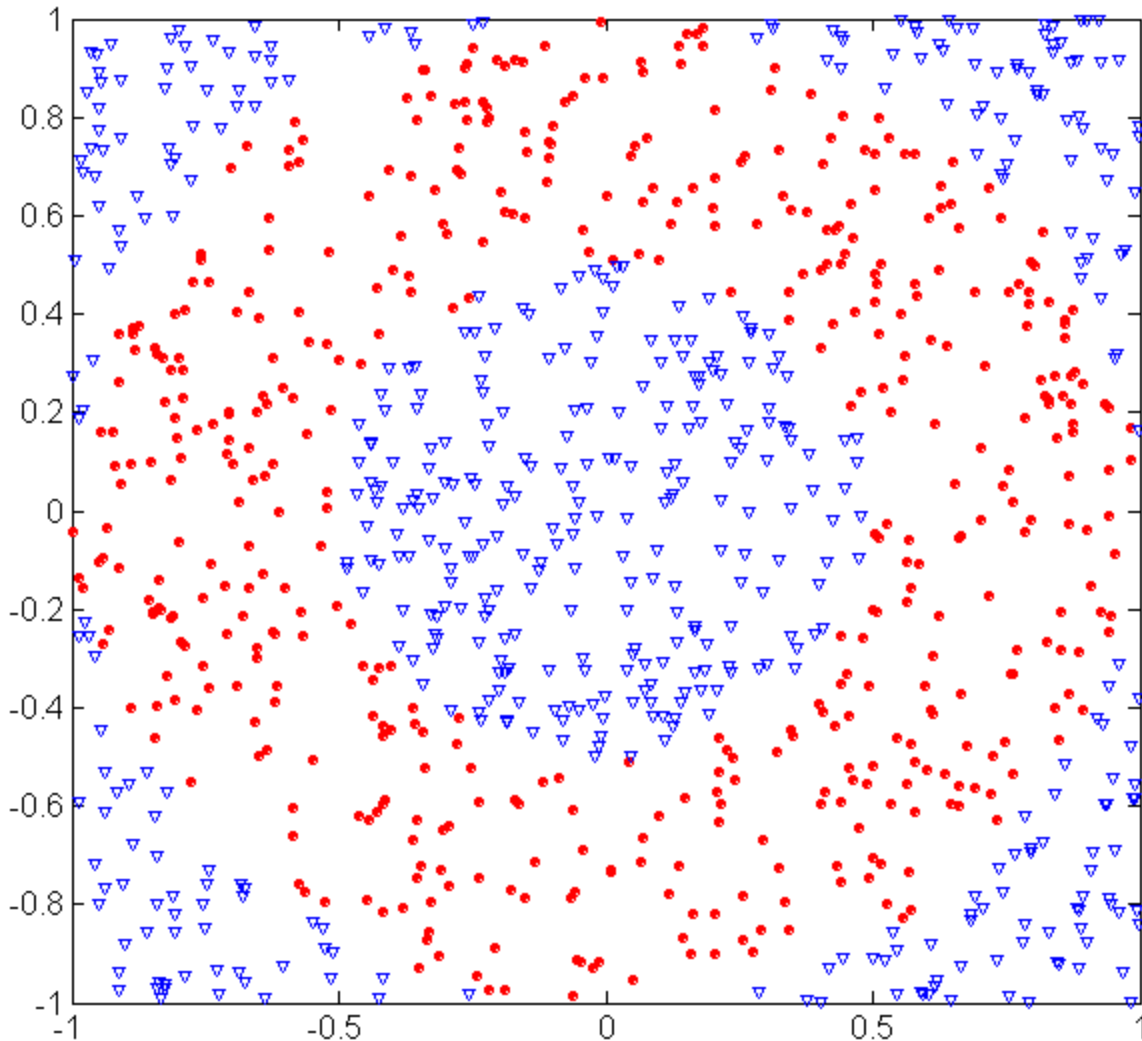


- Test condition may involve multiple attributes
- More expressive representation
- Finding optimal test condition is computationally expensive

Practical Issues of Classification

- Underfitting and Overfitting
- Evaluation

Underfitting and Overfitting (Example)



500 circular and 500
triangular data points.

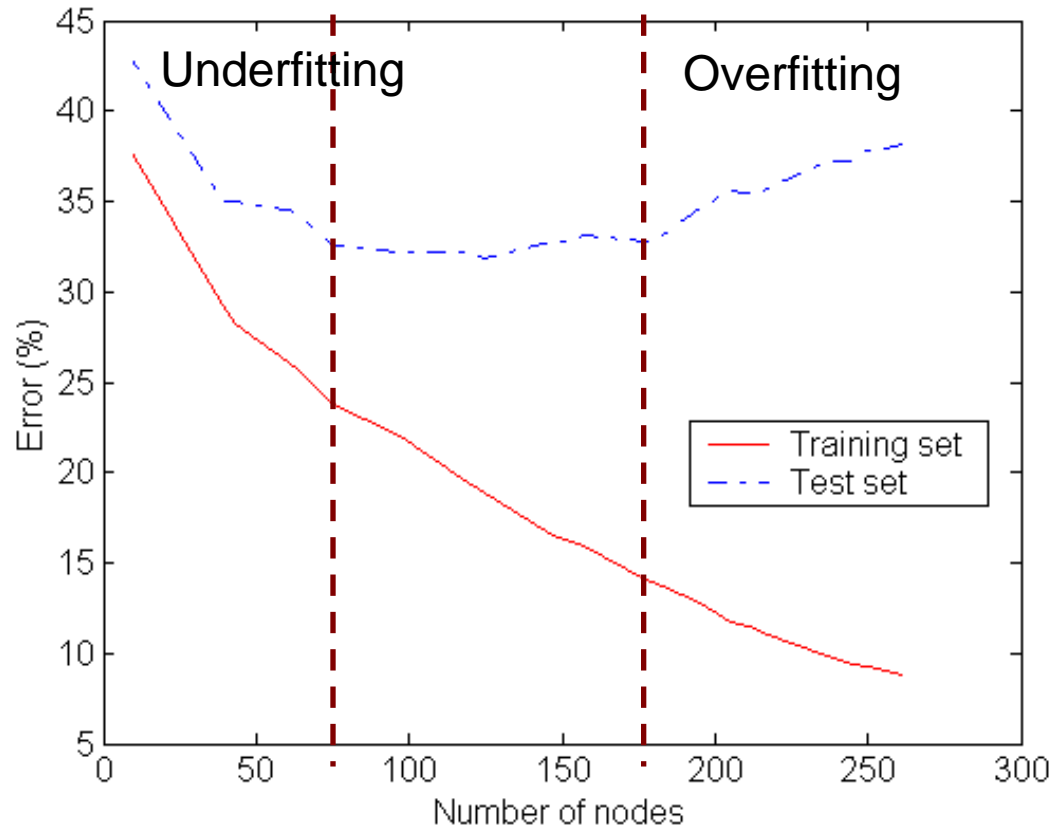
Circular points:

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

Triangular points:

$$\text{sqrt}(x_1^2 + x_2^2) > 0.5 \text{ or} \\ \text{sqrt}(x_1^2 + x_2^2) < 1$$

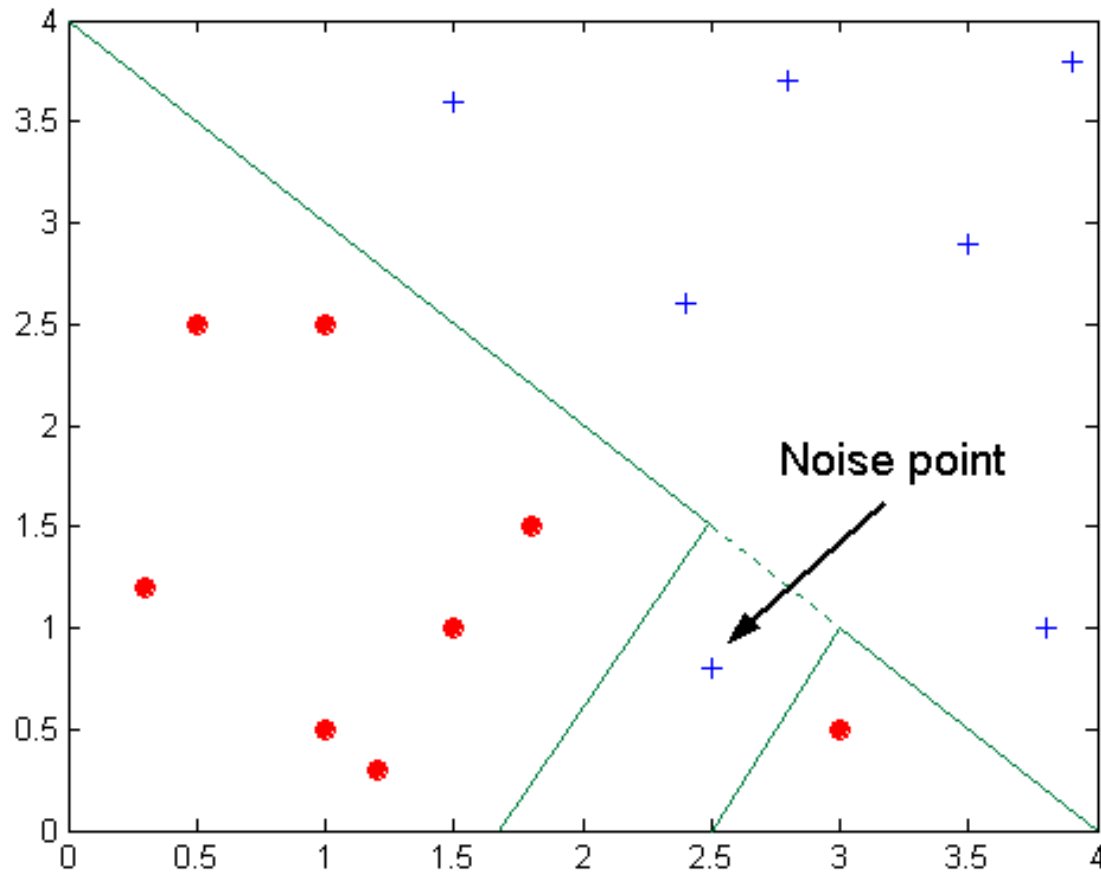
Underfitting and Overfitting



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

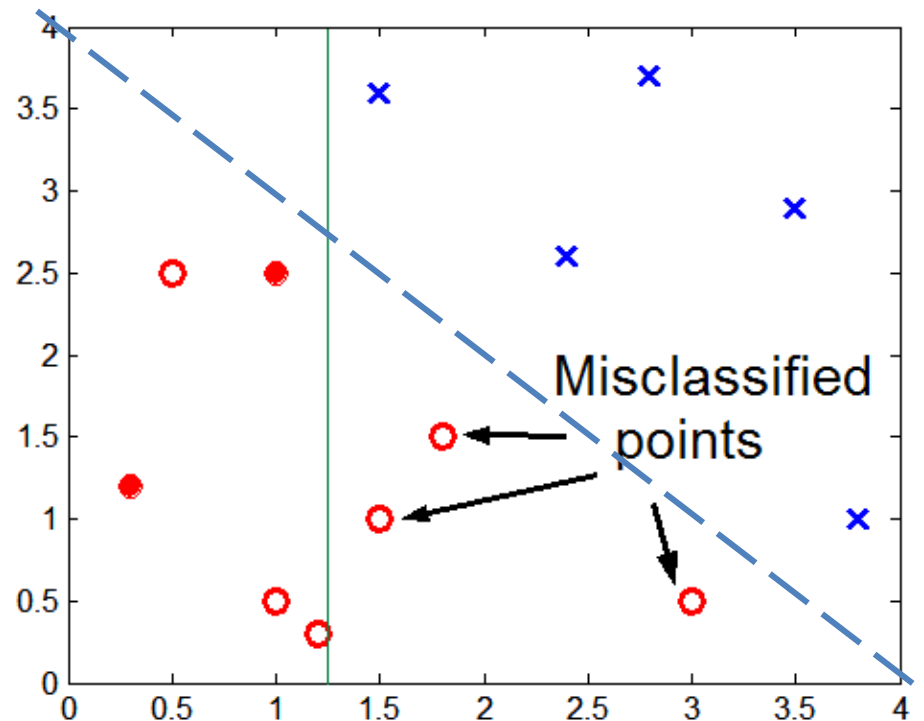
Overfitting: when model is **too complex** it models the details of the training set and fails on the test set

Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples



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

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

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records
 - The model does not **generalize** well
- Need new ways for estimating errors

Estimating Generalization Errors

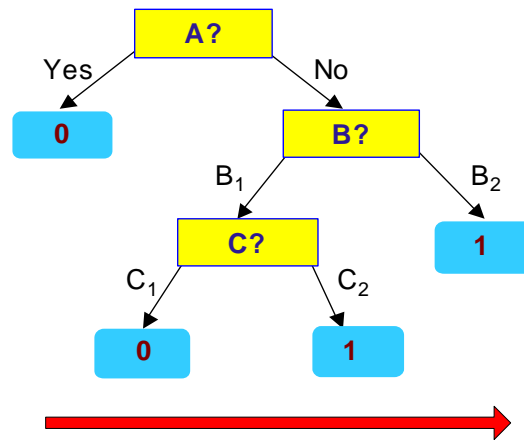
- **Re-substitution errors:** error on **training** ($\sum e(t)$)
- **Generalization errors:** error on **testing** ($\sum e(t)$)
- Methods for estimating generalization errors:
 - **Optimistic approach:** $e'(t) = e(t)$
 - **Pessimistic approach:**
 - For each leaf node: $e'(t) = (e(t) + 0.5)$
 - Total errors: $e'(T) = e(T) + N \times 0.5$ (N: number of leaf nodes)
 - Penalize large trees
 - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances)
 - Training error = $10/1000 = 1\%$
 - Generalization error = $(10 + 30 \times 0.5)/1000 = 2.5\%$
 - **Using validation set:**
 - Split data into **training**, **validation**, **test**
 - Use **validation dataset** to estimate generalization error
 - Drawback: less data for training.

Occam's Razor

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

Minimum Description Length (MDL)

X	y
X ₁	1
X ₂	0
X ₃	0
X ₄	1
...	...
X _n	1



X	y
X ₁	?
X ₂	?
X ₃	?
X ₄	?
...	...
X _n	?

- $\text{Cost}(\text{Model}, \text{Data}) = \text{Cost}(\text{Data} | \text{Model}) + \text{Cost}(\text{Model})$
 - Search for the least costly model.
- $\text{Cost}(\text{Data} | \text{Model})$ encodes the **misclassification errors**.
- $\text{Cost}(\text{Model})$ encodes the **decision tree**
 - node encoding (number of children) plus splitting condition encoding.

How to Address Overfitting

- **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).

How to Address Overfitting...

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

Example of Post-Pruning

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

Training Error (Before splitting) = 10/30

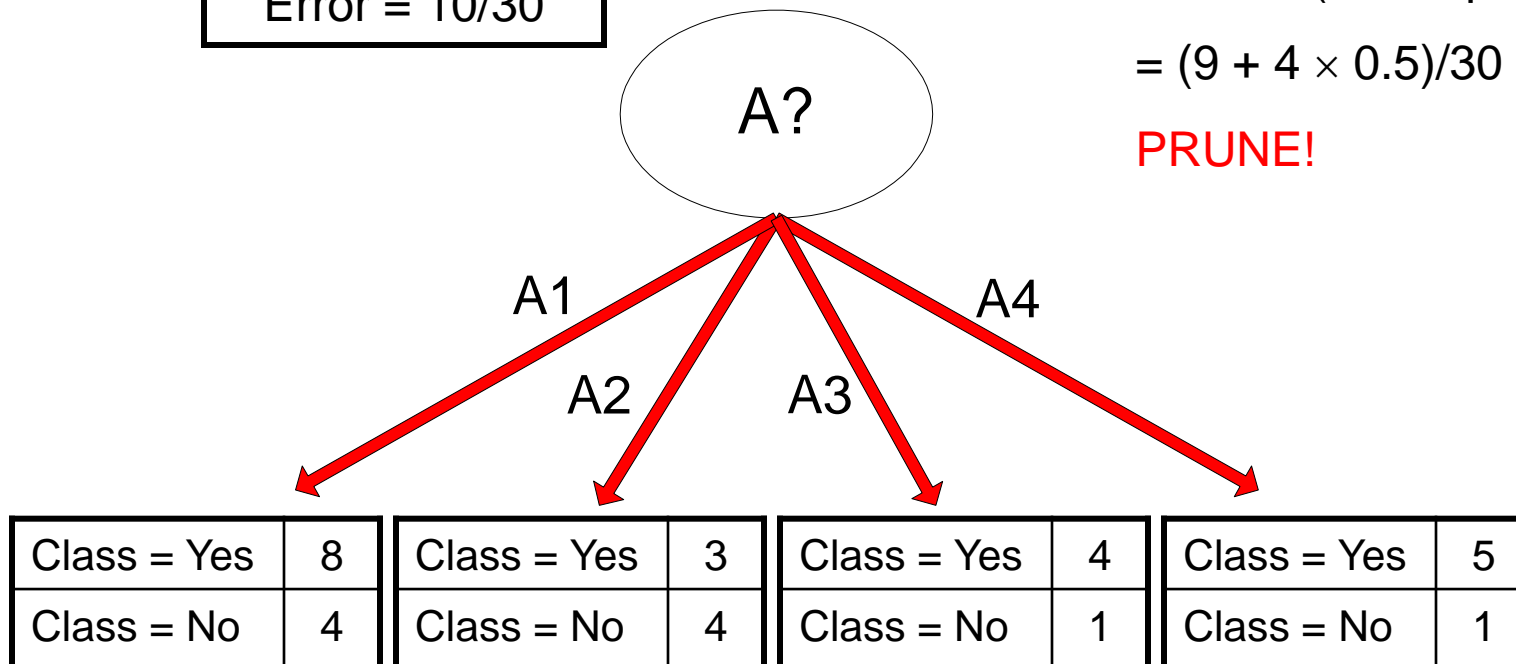
Pessimistic error = $(10 + 0.5)/30 = 10.5/30$

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= $(9 + 4 \times 0.5)/30 = 11/30$

PRUNE!



Model Evaluation

- Metrics for Performance Evaluation
 - How to evaluate the performance of a model?
- Methods for Performance Evaluation
 - How to obtain reliable estimates?
- Methods for Model Comparison
 - How to compare the relative performance among competing models?

Model Evaluation

- **Metrics for Performance Evaluation**
 - How to evaluate the performance of a model?
- **Methods for Performance Evaluation**
 - How to obtain reliable estimates?
- **Methods for Model Comparison**
 - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the **predictive capability** of a model
 - Rather than how fast it takes to classify or build models, scalability, etc.
- **Confusion Matrix:**

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

- a: TP (true positive)
- b: FN (false negative)
- c: FP (false positive)
- d: TN (true negative)

Metrics for Performance Evaluation...

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

- Most widely-used metric:

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

Limitation of Accuracy

- Consider a 2-class problem
 - Number of Class 0 examples = 9990
 - Number of Class 1 examples = 10
- If model predicts everything to be class 0, accuracy is $9990/10000 = 99.9\%$
 - Accuracy is misleading because model does not detect any class 1 example

Cost Matrix

	PREDICTED CLASS		
	$C(i j)$	Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	$C(\text{Yes} \text{Yes})$	$C(\text{No} \text{Yes})$
	Class=No	$C(\text{Yes} \text{No})$	$C(\text{No} \text{No})$

$C(i|j)$: Cost of classifying class j example as class i

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
	C(i j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 90%

Cost = 4255

Cost vs Accuracy

Count	PREDICTED CLASS		
	Class=Yes	Class=No	
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

Accuracy is proportional to cost if

1. $C(\text{Yes}|\text{No})=C(\text{No}|\text{Yes}) = q$
2. $C(\text{Yes}|\text{Yes})=C(\text{No}|\text{No}) = p$

$$N = a + b + c + d$$

$$\text{Accuracy} = (a + d)/N$$

Cost	PREDICTED CLASS		
	Class=Yes	Class=No	
ACTUAL CLASS	Class=Yes	p	q
	Class=No	q	p

$$\begin{aligned} \text{Cost} &= p(a + d) + q(b + c) \\ &= p(a + d) + q(N - a - d) \\ &= qN - (q - p)(a + d) \\ &= N[q - (q - p) \times \text{Accuracy}] \end{aligned}$$

Precision-Recall

$$\text{Precision (p)} = \frac{a}{a+c} = \frac{TP}{TP+FP}$$

$$\text{Recall (r)} = \frac{a}{a+b} = \frac{TP}{TP+FN}$$

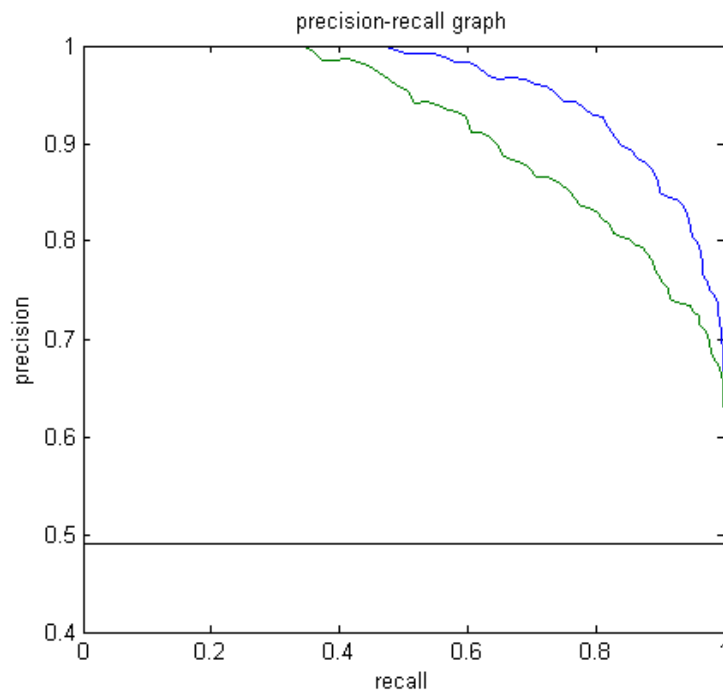
$$\text{F-measure (F)} = \frac{1}{\left(\frac{1/r+1/p}{2}\right)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c} = \frac{2TP}{2TP+FP+FN}$$

Count	PREDICTED CLASS		
	Class=Yes	Class=No	
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

- Precision is biased towards **C(Yes|Yes)** & **C(Yes|No)**
- Recall is biased towards **C(Yes|Yes)** & **C(No|Yes)**
- F-measure is biased towards all **except C(No|No)**

Precision-Recall plot

- Usually for parameterized models, it controls the precision/recall tradeoff



Model Evaluation

- Metrics for Performance Evaluation
 - How to evaluate the performance of a model?
- **Methods for Performance Evaluation**
 - How to obtain reliable estimates?
- Methods for Model Comparison
 - How to compare the relative performance among competing models?

Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
 - Class distribution
 - Cost of misclassification
 - Size of training and test sets

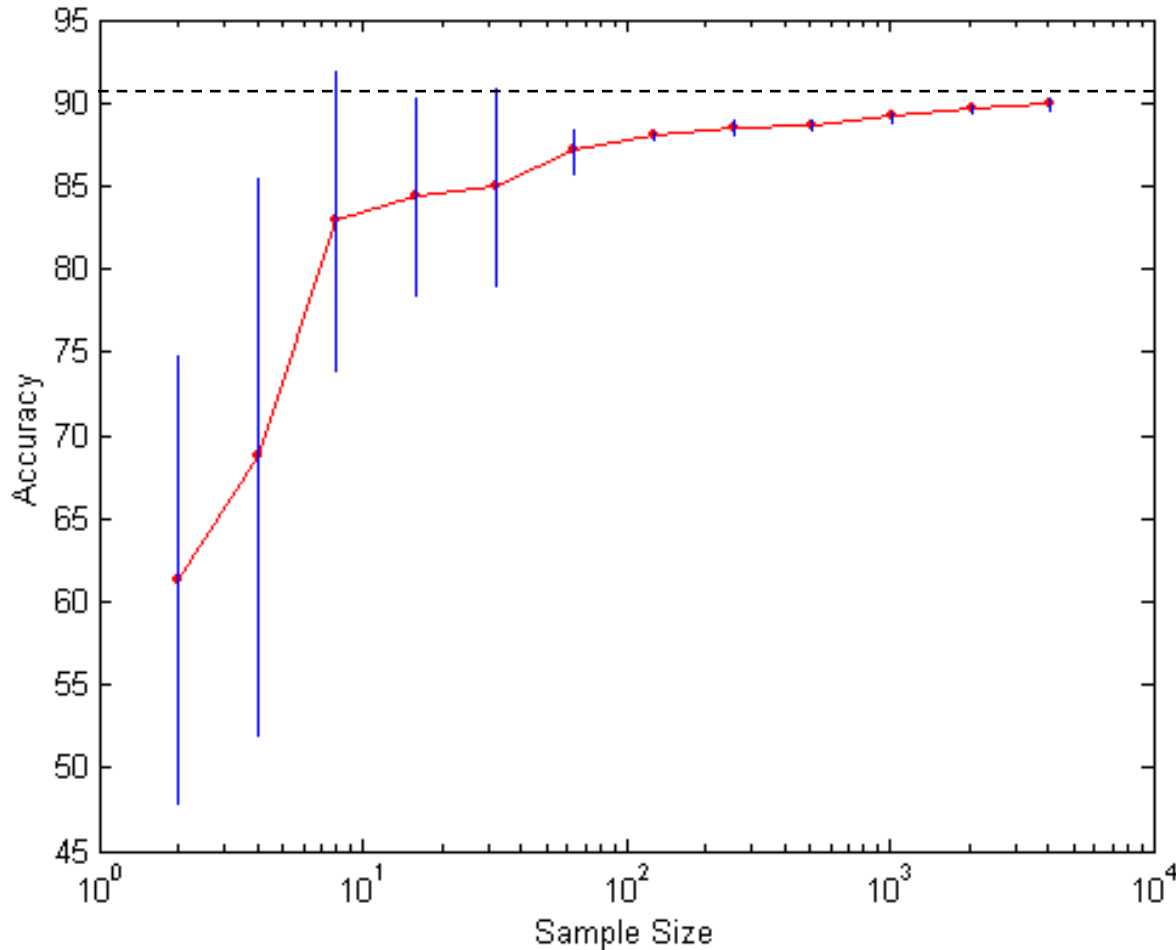
Methods of Estimation

- **Holdout**
 - Reserve **2/3** for training and **1/3** for testing
- **Random subsampling**
 - One sample may be biased -- Repeated holdout
- **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**
 - Guarantees that each record is used the same number of times for training and testing
- **Bootstrap**
 - Sampling with replacement
 - ~63% of records used for training, ~27% for testing

Dealing with class Imbalance

- If the class we are interested in is very rare, then the classifier will ignore it.
 - The class imbalance problem
- Solution
 - We can modify the optimization criterion by using a cost sensitive metric
 - We can **balance** the class distribution
 - Sample from the larger class so that the size of the two classes is the same
 - Replicate the data of the class of interest so that the classes are balanced
 - Over-fitting issues

Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Requires a sampling schedule for creating learning curve

Effect of small sample size:

- Bias in the estimate
- Variance of estimate

Model Evaluation

- Metrics for Performance Evaluation
 - How to evaluate the performance of a model?
- Methods for Performance Evaluation
 - How to obtain reliable estimates?
- **Methods for Model Comparison**
 - How to compare the relative performance among competing models?

ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
 - Characterize the trade-off between positive hits and false alarms
- **ROC** curve plots **TPR** (on the **y**-axis) against **FPR** (on the **x**-axis)

$$TPR = \frac{TP}{TP + FN}$$

Fraction of **positive instances** predicted **correctly**

$$FPR = \frac{FP}{FP + TN}$$

Fraction of **negative instances** predicted **incorrectly**

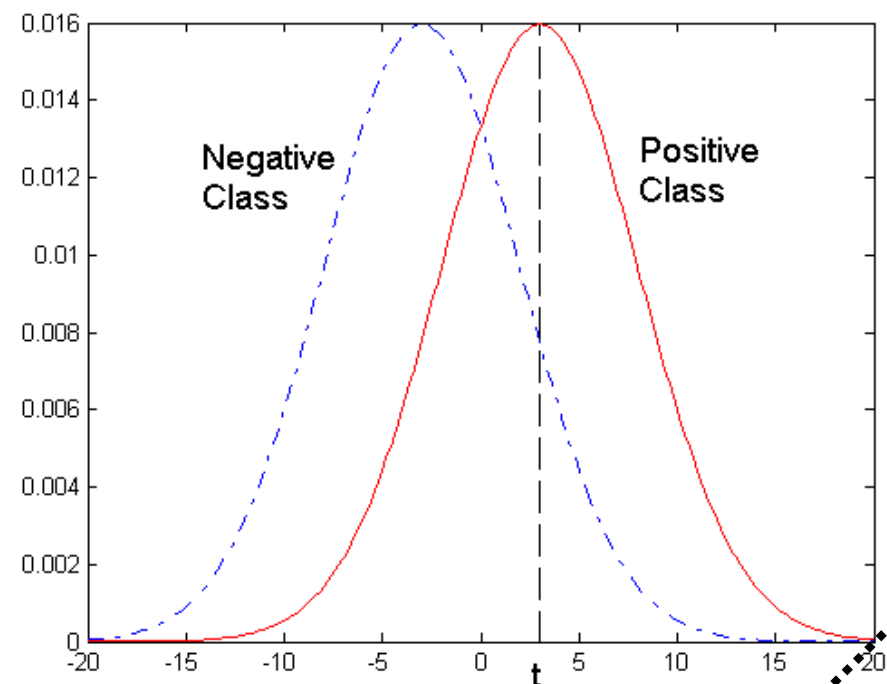
		PREDICTED CLASS	
		Yes	No
Actual	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

ROC (Receiver Operating Characteristic)

- Performance of a classifier represented as a **point** on the **ROC** curve
- Changing some **parameter** of the algorithm, **sample** distribution or **cost matrix** changes the location of the point

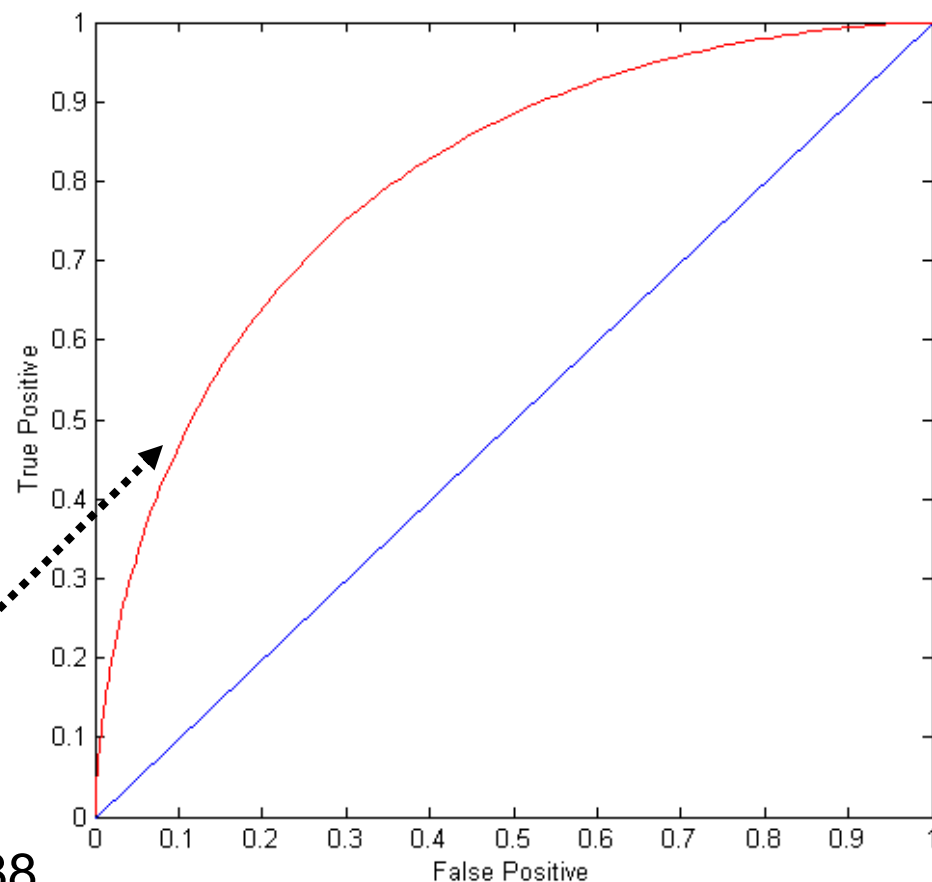
ROC Curve

- **1**-dimensional data set containing **2** classes (*positive* and *negative*)
- any points located at $x > t$ is classified as *positive*



At threshold t :

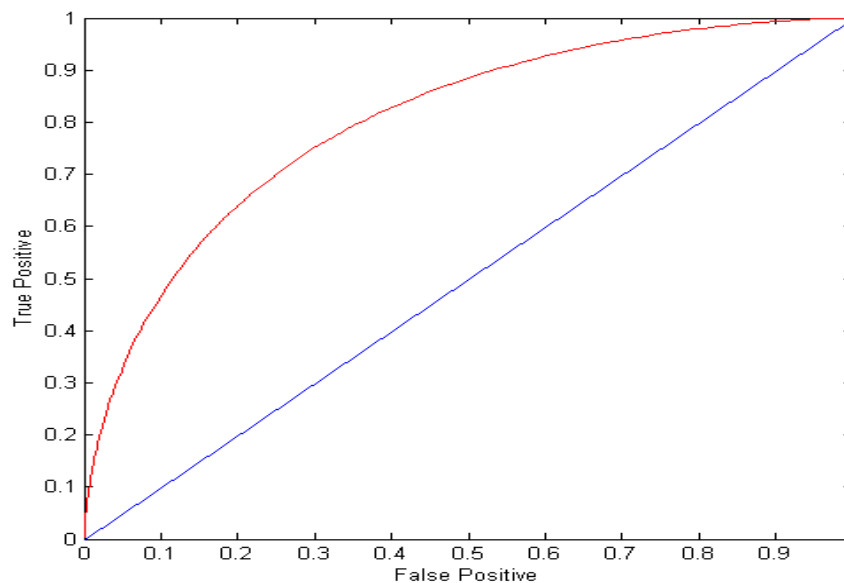
TP=0.5, FN=0.5, FP=0.12, FN=0.88



ROC Curve

(TP,FP):

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

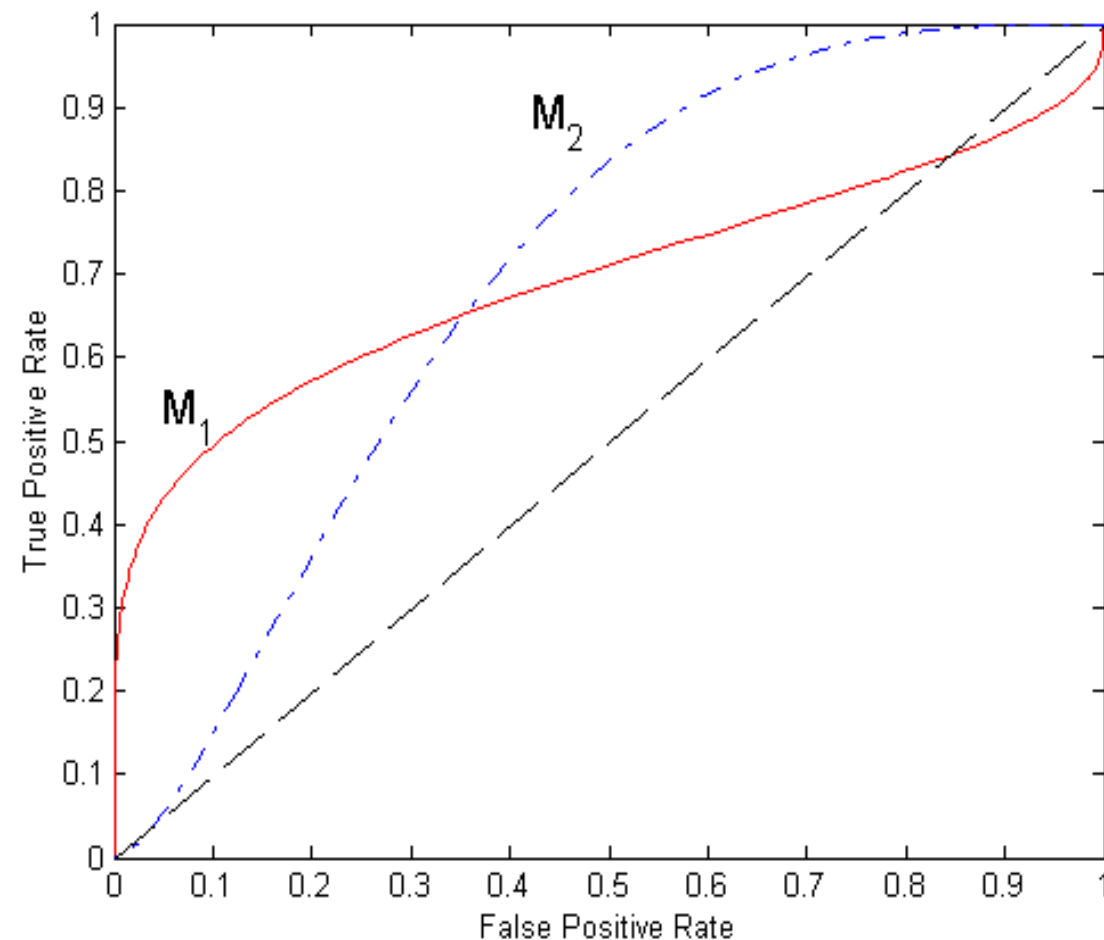


• Diagonal line:

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

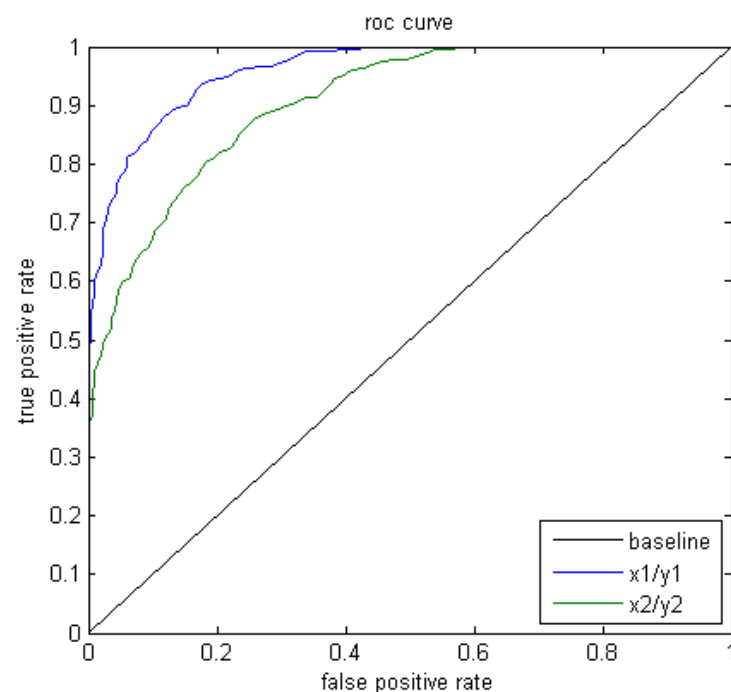
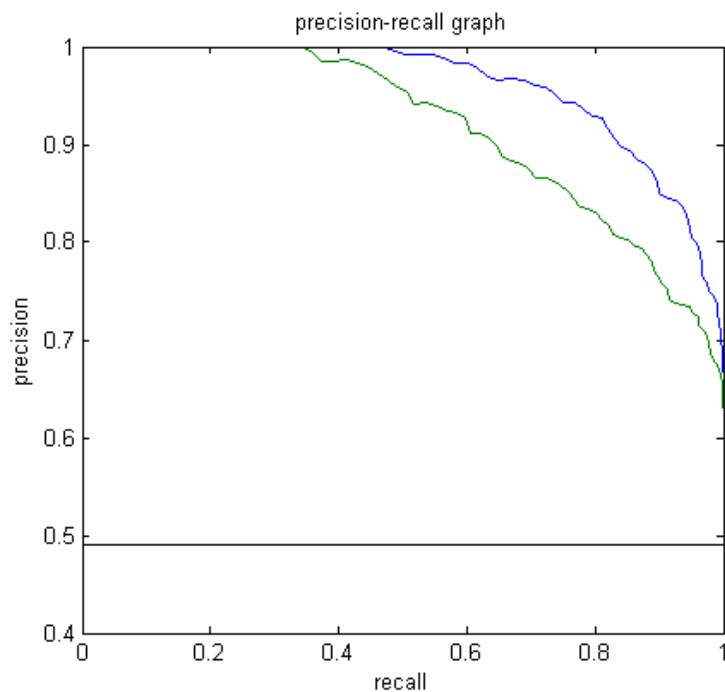
		PREDICTED CLASS	
		Yes	No
Actual	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

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 (**AUC**)
 - Ideal: Area = 1
 - Random guess:
 - Area = 0.5

ROC curve vs Precision-Recall curve



Area Under the Curve (AUC) as a single number for evaluation

NEAREST NEIGHBOR CLASSIFICATION

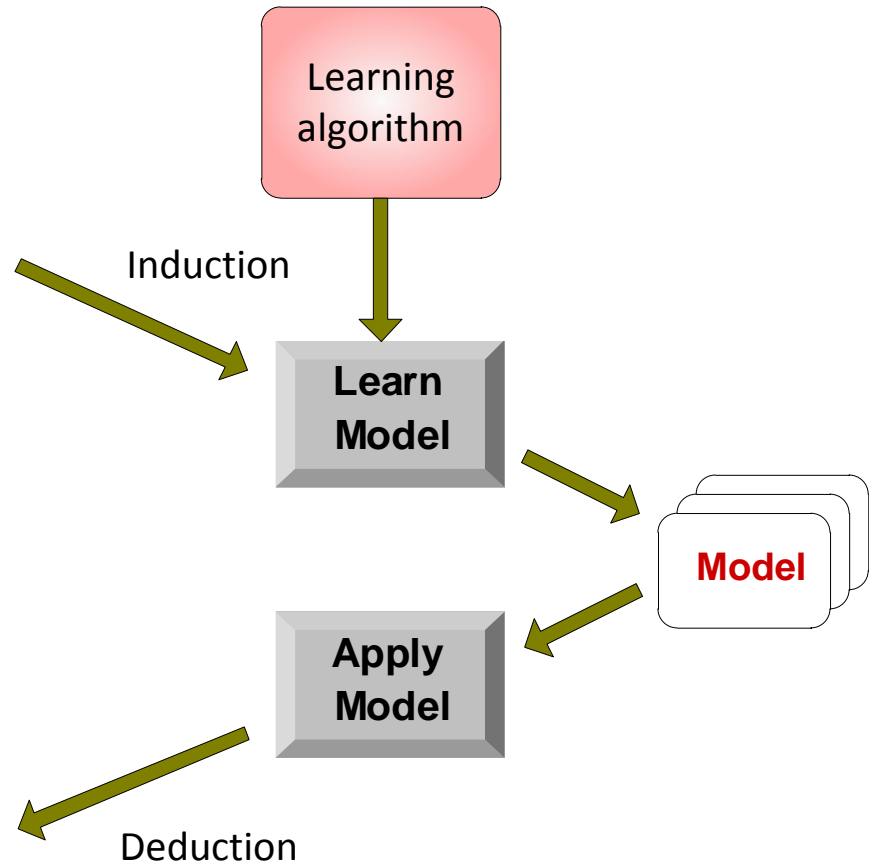
Illustrating Classification Task

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



Instance-Based Classifiers

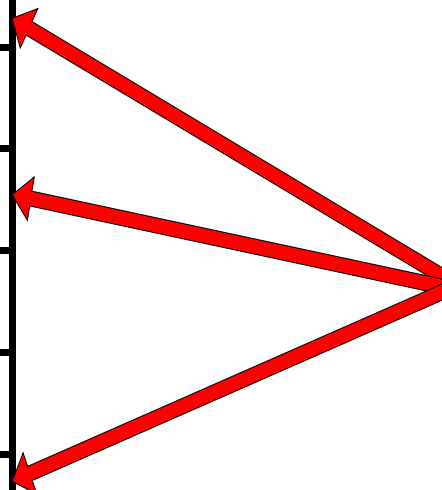
Set of Stored Cases

Atr1	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

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

Unseen Case

Atr1	AtrN

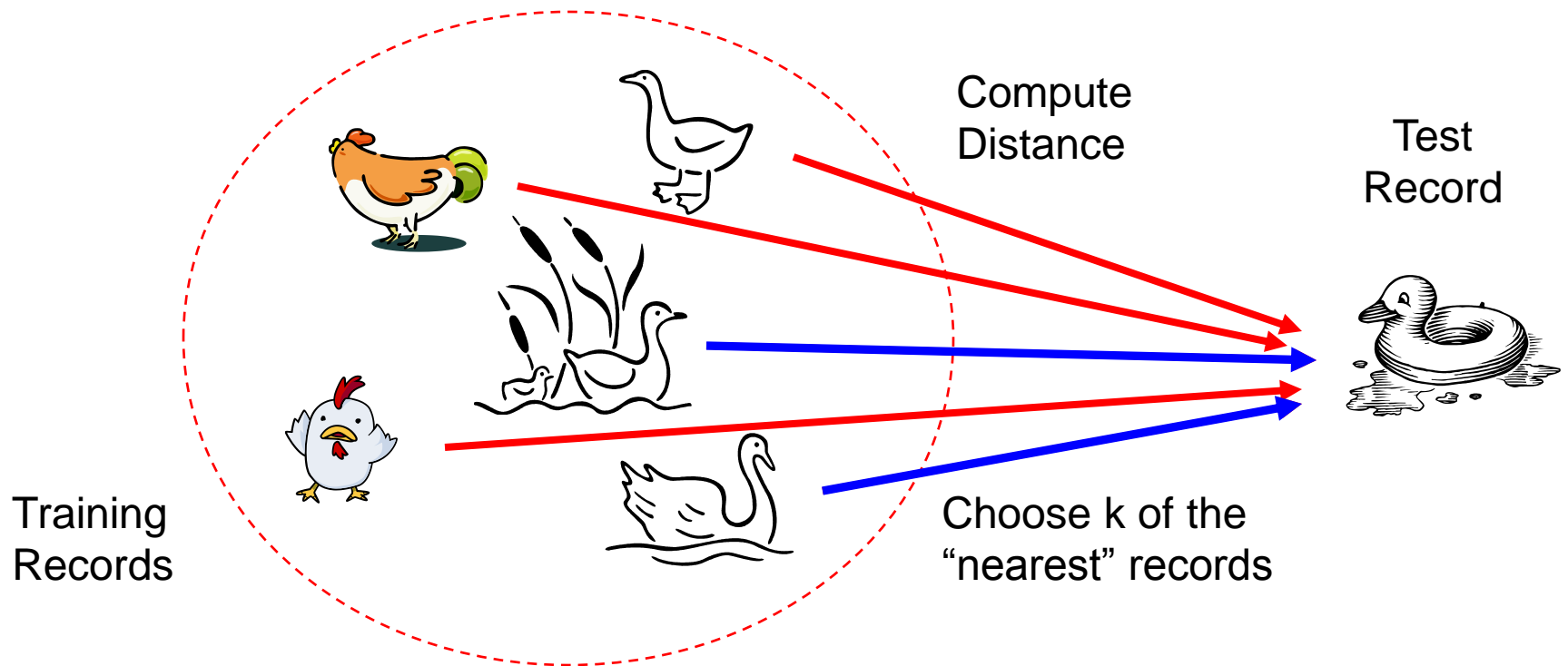


Instance Based Classifiers

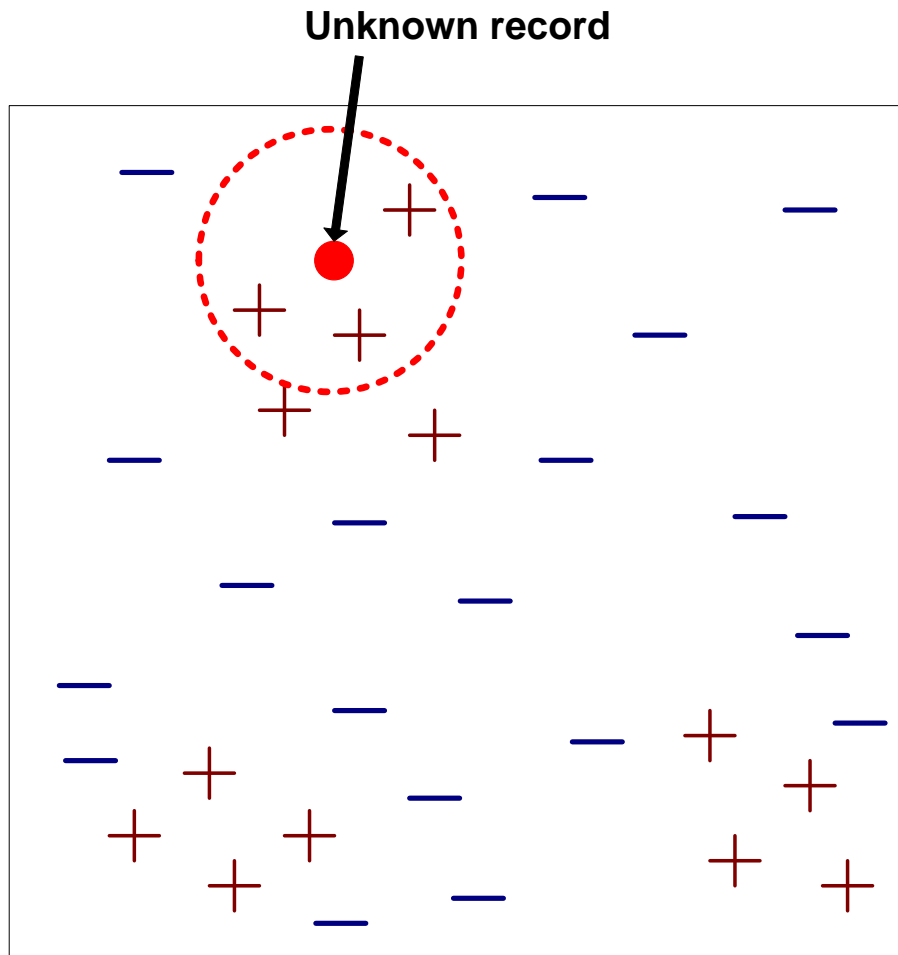
- Examples:
 - **Rote-learner**
 - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
 - **Nearest neighbor classifier**
 - Uses k “closest” points (nearest neighbors) for performing classification

Nearest Neighbor Classifiers

- Basic idea:
 - *“If it walks like a duck, quacks like a duck, then it’s probably a duck”*

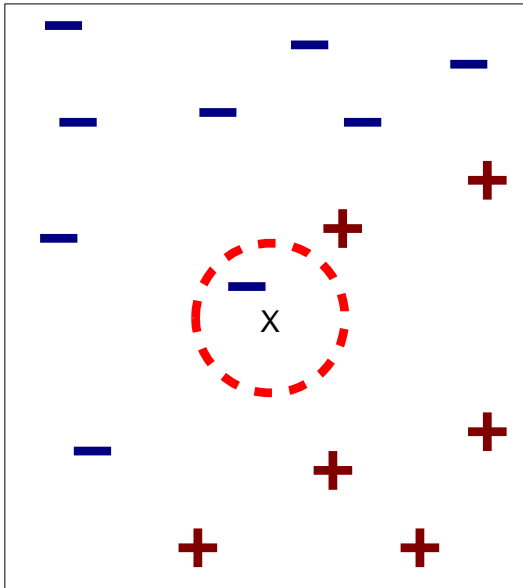


Nearest-Neighbor Classifiers

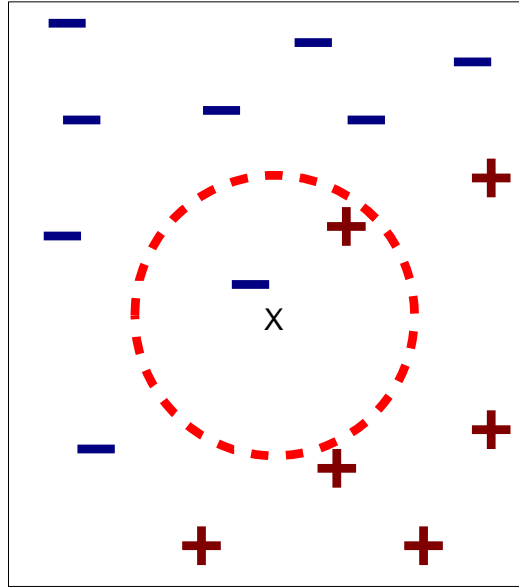


- Requires three things
 - The set of **stored records**
 - **Distance Metric** to compute distance between records
 - The value of **k** , the **number of nearest neighbors** to retrieve
- To classify an unknown record:
 1. **Compute distance** to other training records
 2. Identify **k nearest neighbors**
 3. Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking **majority vote**)

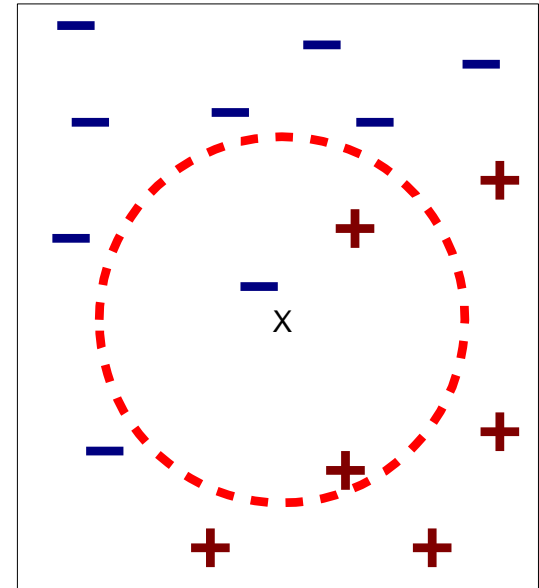
Definition of Nearest Neighbor



(a) 1-nearest neighbor



(b) 2-nearest neighbor

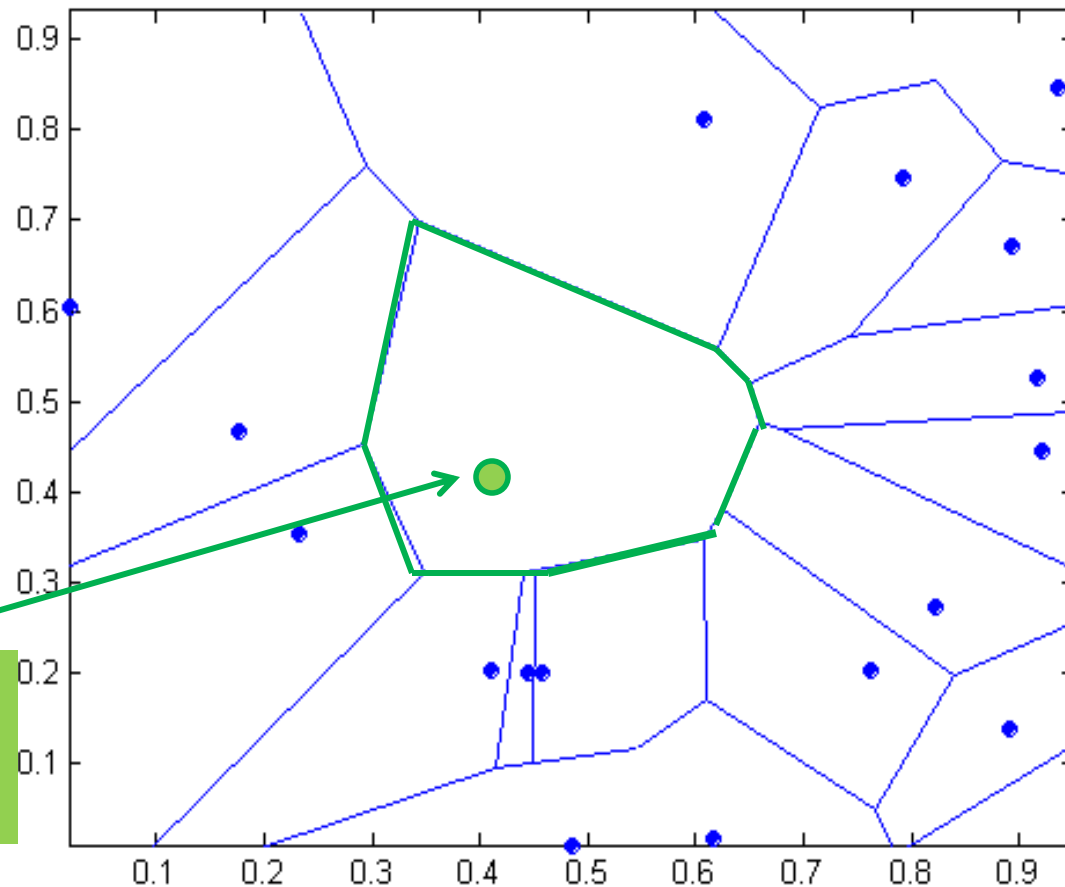


(c) 3-nearest neighbor

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

1 nearest-neighbor

Voronoi Diagram defines the classification boundary



The area takes the class of the green point

Nearest Neighbor Classification

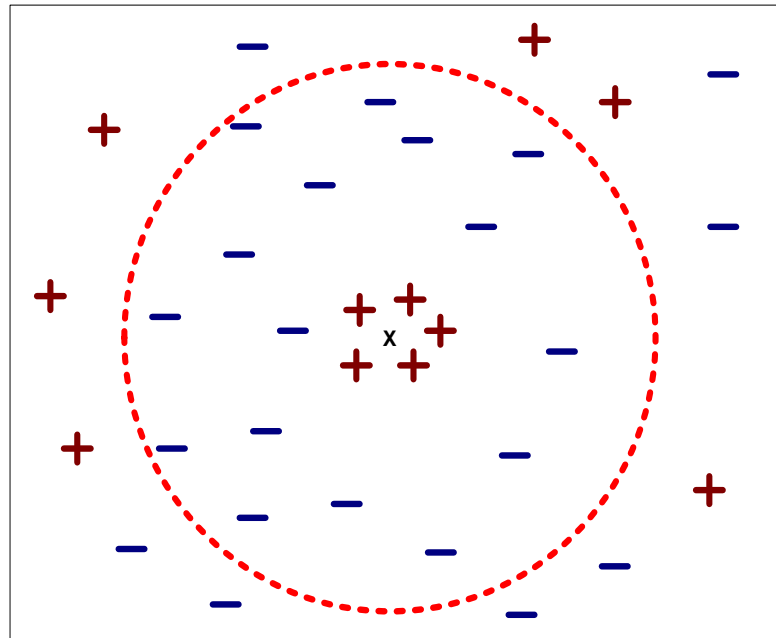
- Compute distance between two points:
 - Euclidean distance

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

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

Nearest Neighbor Classification...

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



Nearest Neighbor Classification...

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

Nearest Neighbor Classification...

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

1 1 1 1 1 1 1 1 1 1 1 0

vs

1 0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1 1 1 1 1

0 0 0 0 0 0 0 0 0 0 0 1

$d = 1.4142$

$d = 1.4142$

- ◆ Solution: Normalize the vectors to unit length

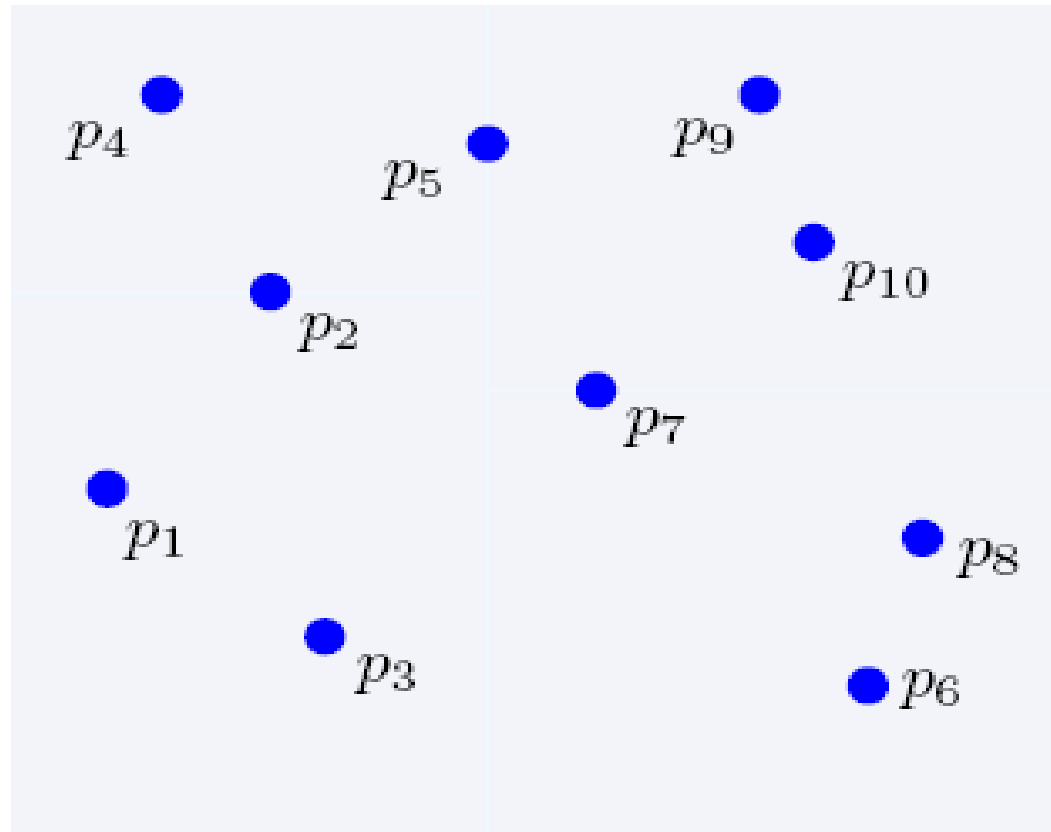
Nearest neighbor Classification...

- k-NN classifiers are **lazy learners**
 - It does not build models explicitly
 - Unlike **eager learners** such as decision trees
- Classifying unknown records are relatively expensive
 - Naïve algorithm: $O(n)$
 - Need for structures to retrieve nearest neighbors fast.
 - The **Nearest Neighbor Search** problem.

Nearest Neighbor Search

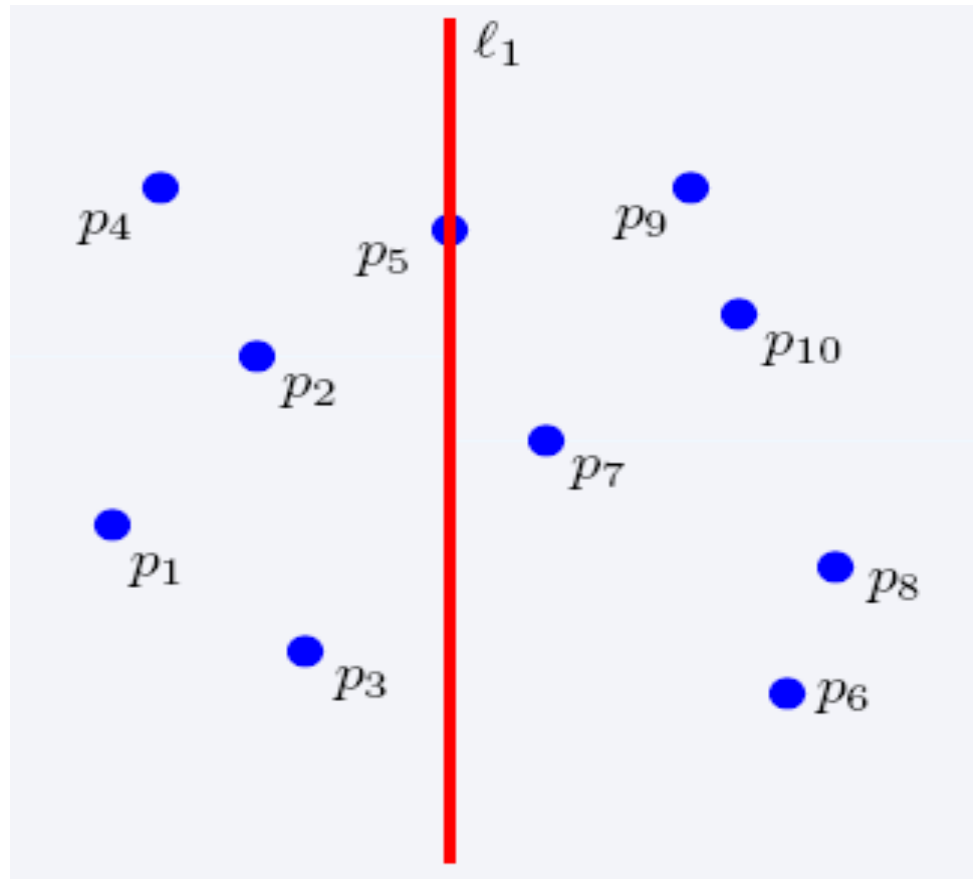
- Two-dimensional **kd-trees**
 - A data structure for answering nearest neighbor queries in \mathbb{R}^2
- kd-tree construction algorithm
 - Select the **x** or **y** dimension (alternating between the two)
 - Partition the space into two with a line passing from the median point
 - Repeat recursively in the two partitions as long as there are enough points

Nearest Neighbor Search



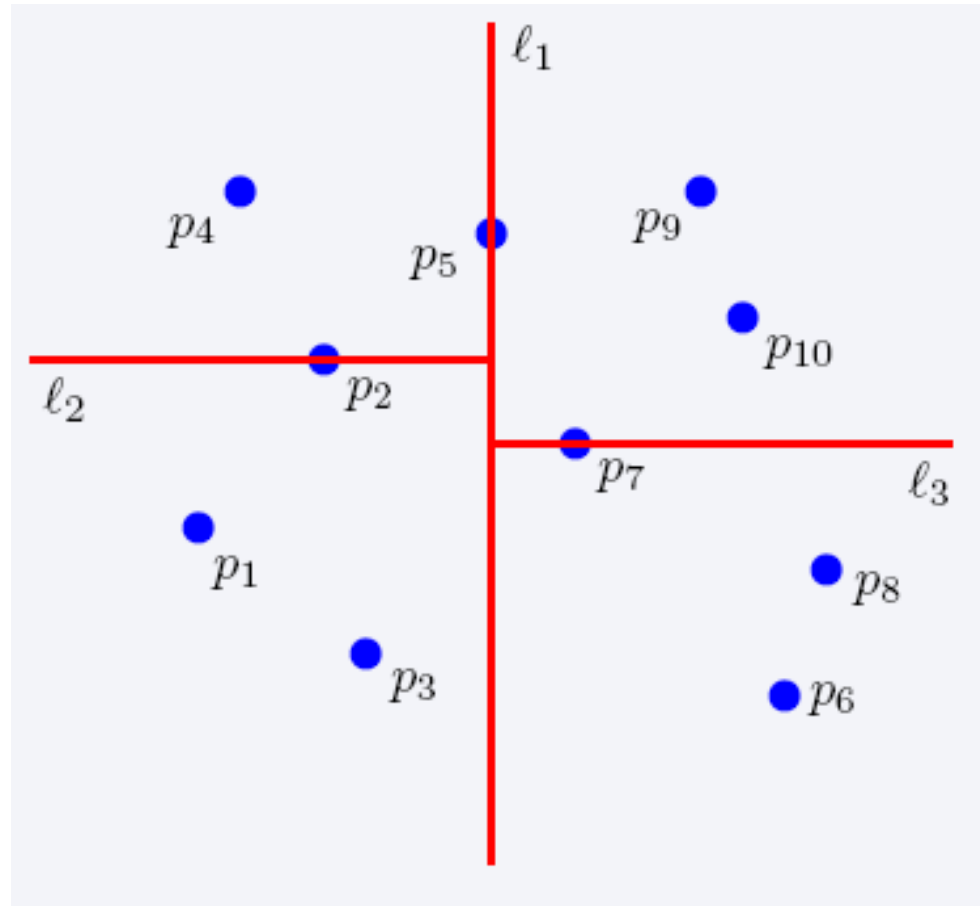
2-dimensional kd-trees

Nearest Neighbor Search



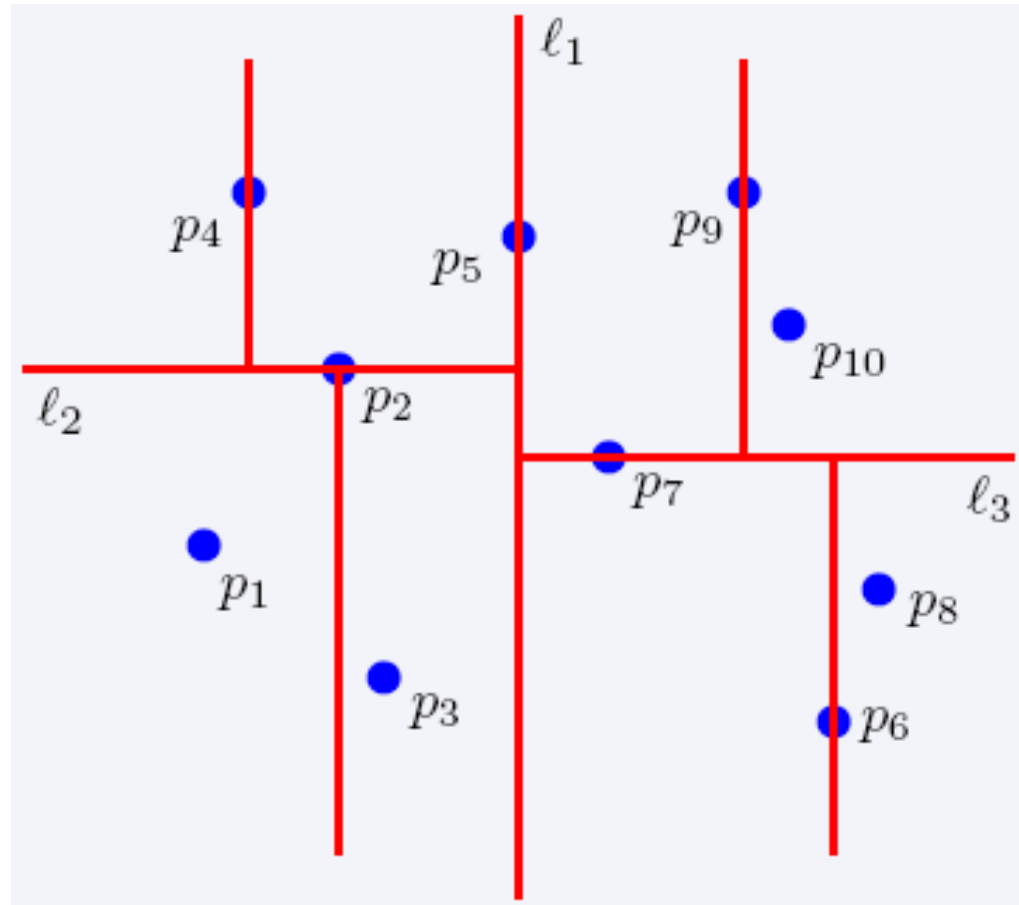
2-dimensional kd-trees

Nearest Neighbor Search



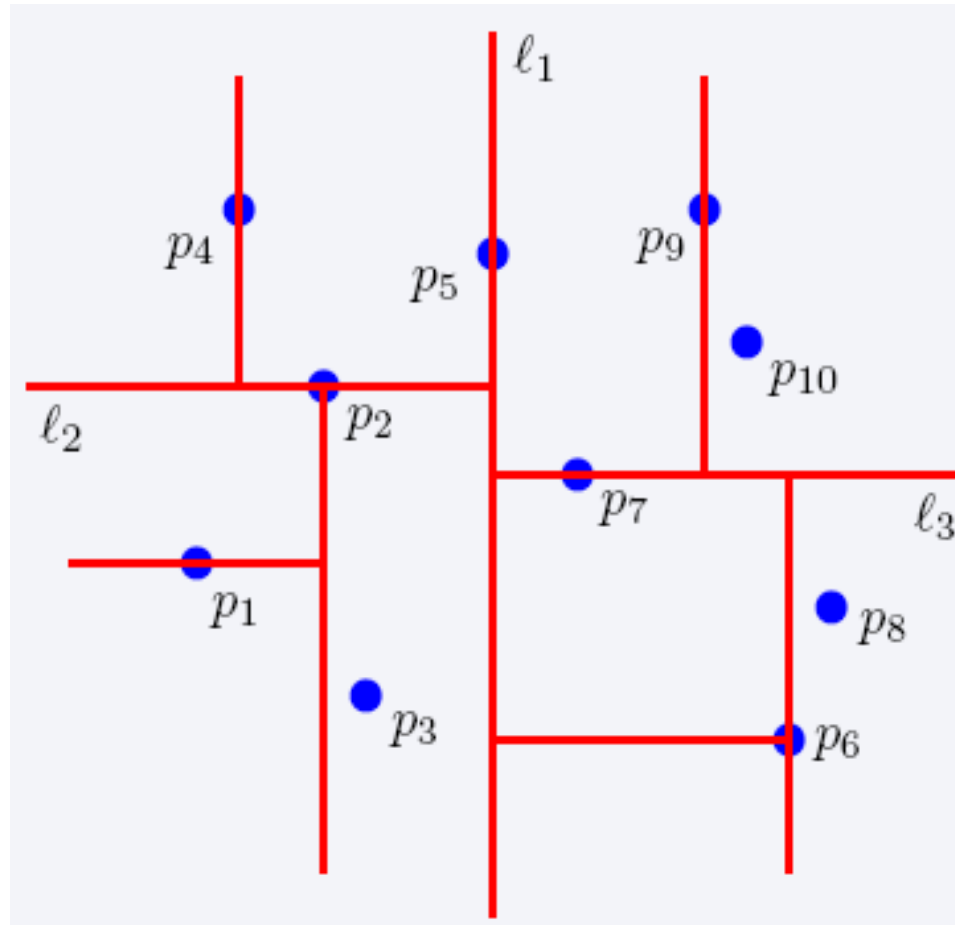
2-dimensional kd-trees

Nearest Neighbor Search



2-dimensional kd-trees

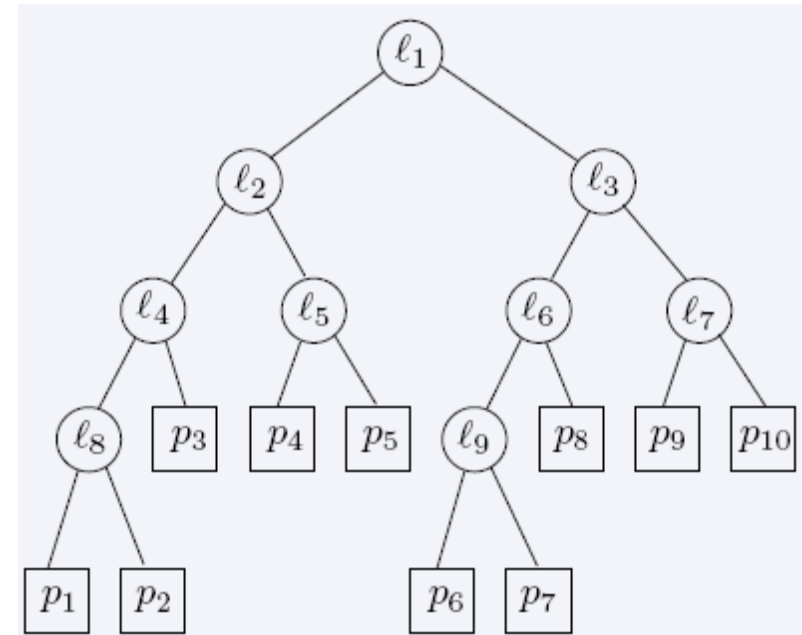
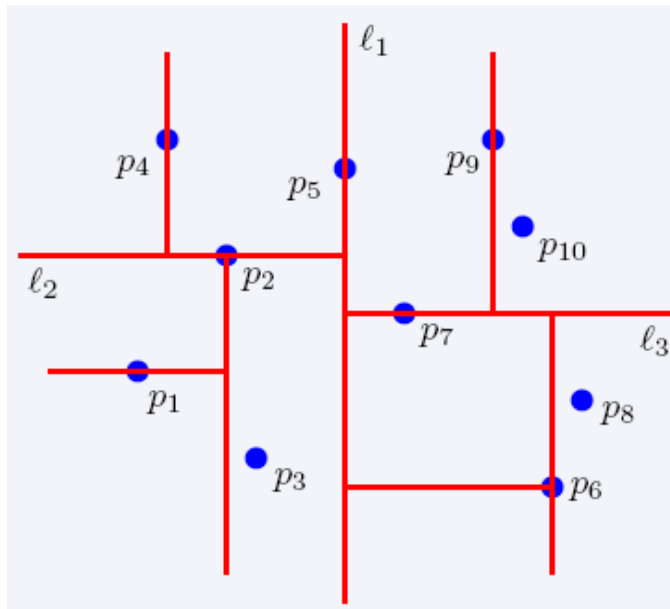
Nearest Neighbor Search



2-dimensional kd-trees

Nearest Neighbor Search

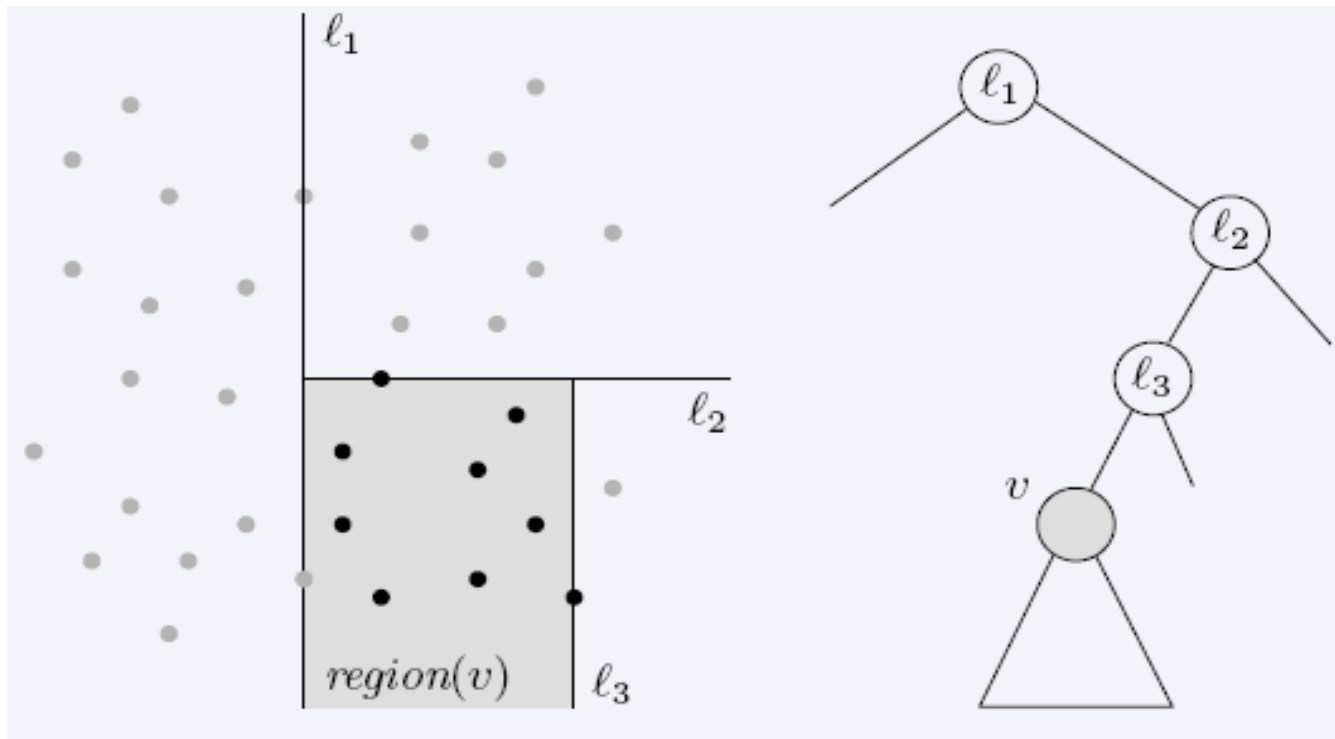
2-dimensional kd-trees



Nearest Neighbor Search

2-dimensional kd-trees

$\text{region}(u)$ – all the black points in the subtree of u



Nearest Neighbor Search

2-dimensional kd-trees

- A binary tree:
 - Size $O(n)$
 - Depth $O(\log n)$
 - Construction time $O(n \log n)$
 - Query time: worst case $O(n)$, but for many cases $O(\log n)$

Generalizes to d dimensions

- Example of Binary Space Partitioning