# Introduction to Data Mining

## Chapter 3 Classification – Basic Concepts

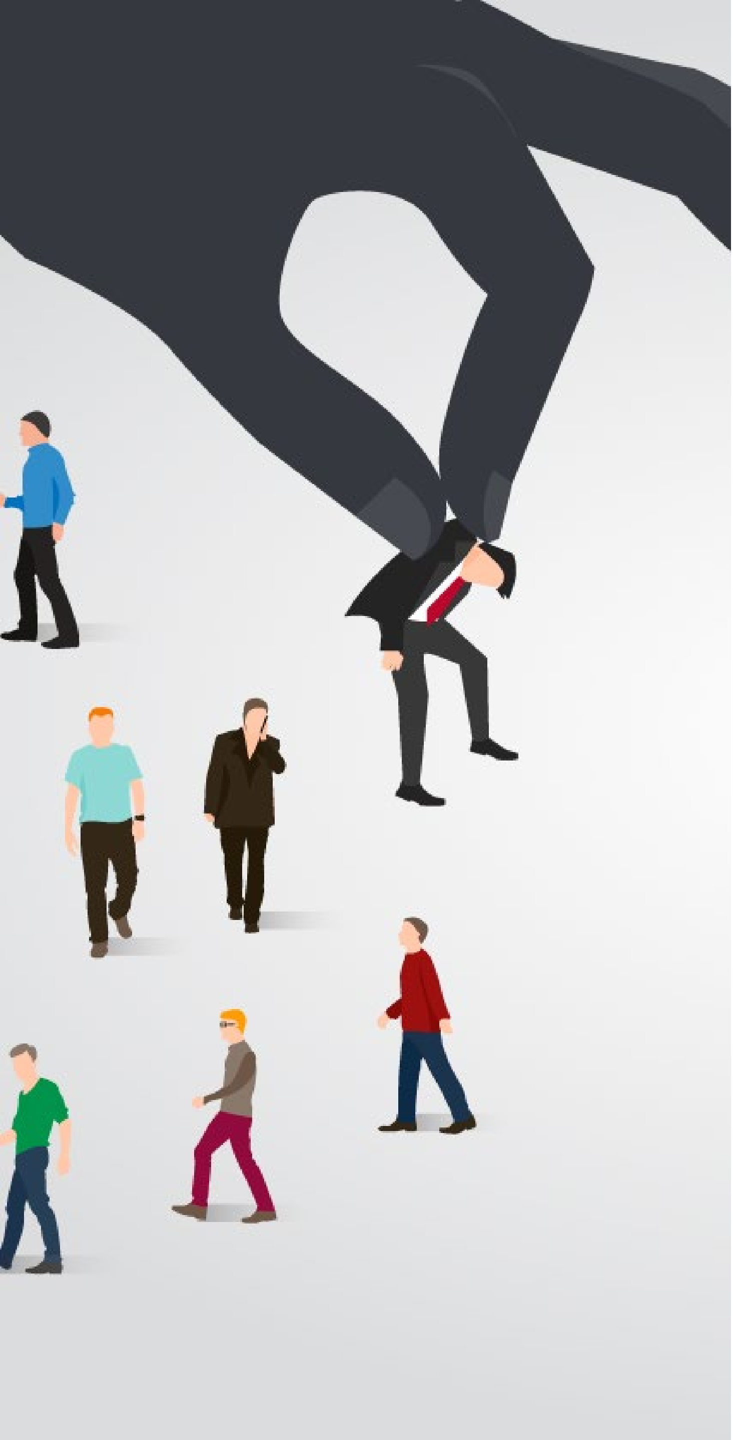
by Michael Hahsler

Based in Slides by Tan, Steinbach, Karpatne, Kumar

# R Code Examples

- Available R Code examples are indicated on slides by the R logo

- The Examples are available at https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/



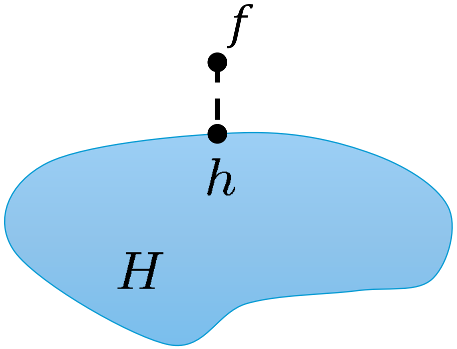
# Topics

- **Introduction**
- Decision Trees
  - —Overview
  - —Tree Induction
- Overfitting and other Practical Issues
- Model Selection and Evaluation
  - —Metrics for Performance Evaluation
  - —Methods to Obtain Reliable Estimates
  - —Model Comparison (Relative Performance)
- Feature Selection

# Supervised Learning – Learning from Examples

- Examples
  - Input-output pairs: $\mathrm{E} = (x_1, y_1), \dots, (x_i, y_i), \dots, (x_N, y_N)$.
  - We assume that the examples are produced iid (with noise and errors) from a target function $y = f(x)$.

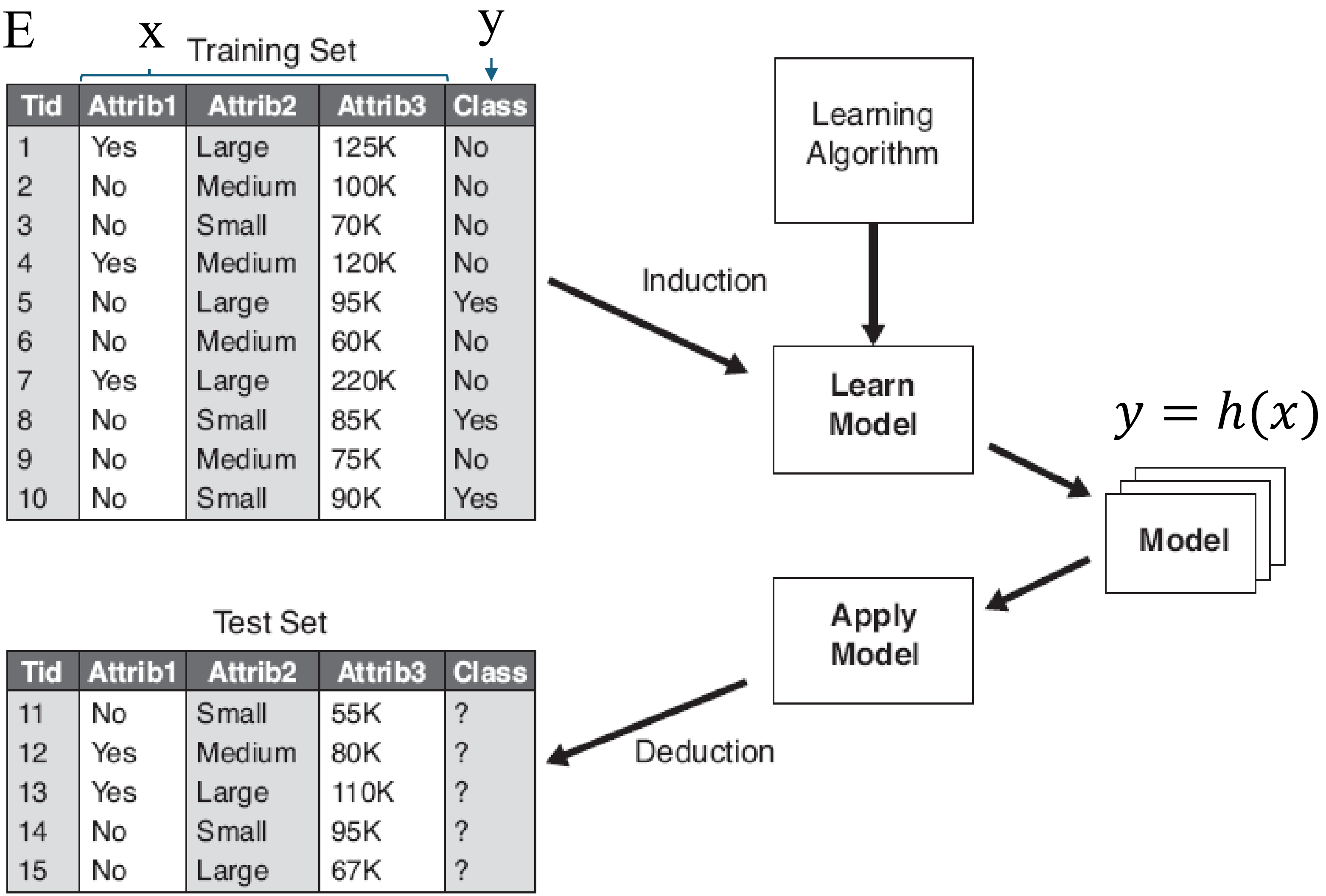
- Learning problem
  - Given a hypothesis space H
  - Find a hypothesis $h \in H$ such that $\hat{y}_i = h(x_i) \approx y_i$
  - That is, we want to approximate $f$ by $h$ using $\mathrm{E}$.



- Includes
  - **Regression** (outputs = real numbers). Goal: Predict the number accurately. E.g., x is a house and $f(x)$ is its selling price.
  - **Classification** (outputs = class labels). Goal: Assign new records to a class. E.g., $x$ is an email and $f(x)$ is spam / ham

You already know linear regression. We focus on Classification.

# Illustrating Classification Task



Training Set

| 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 |

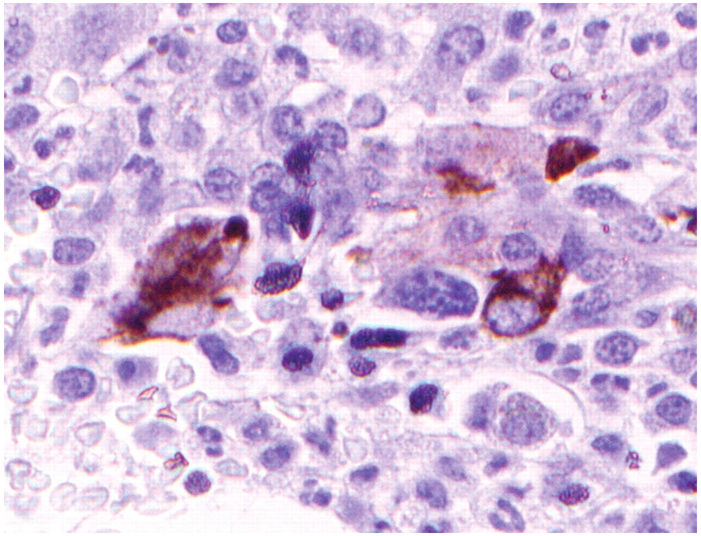
Test Set

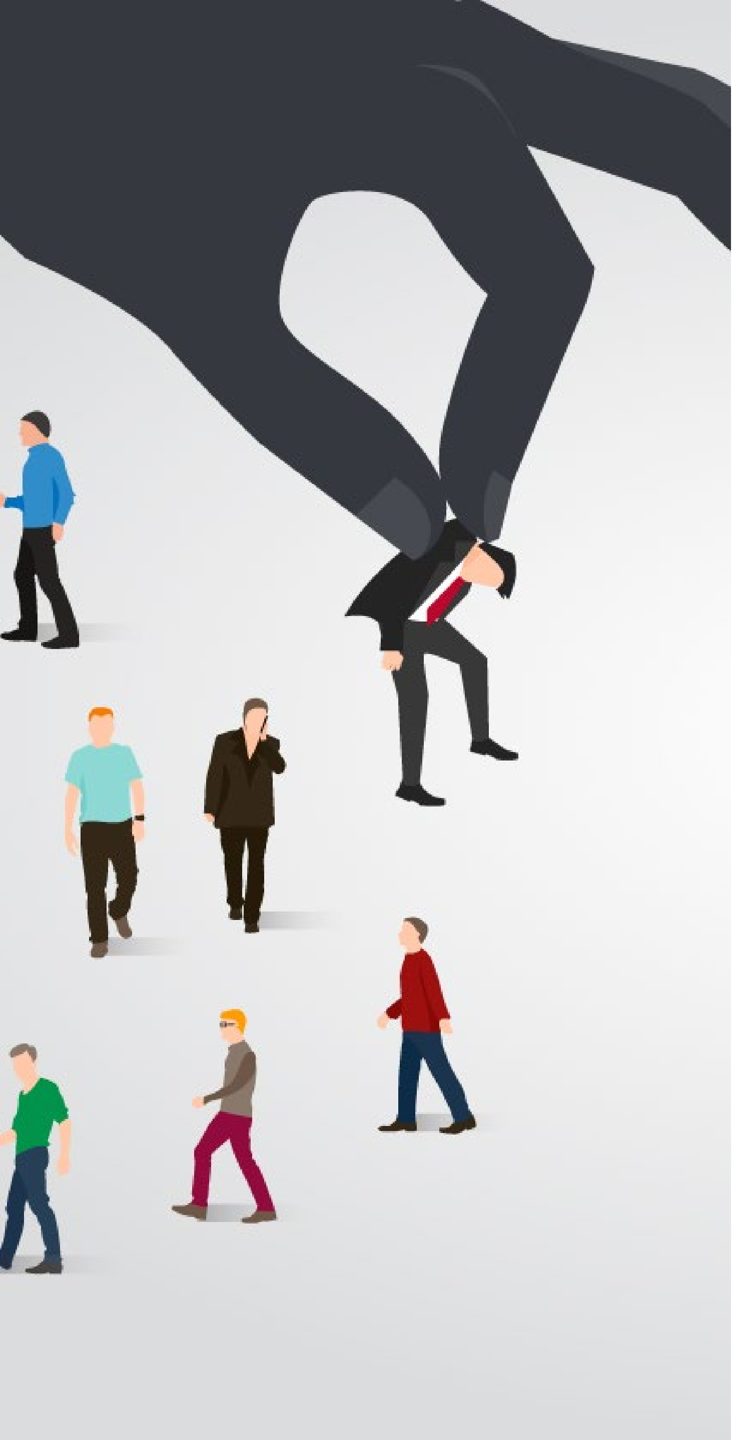
| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|---|---|---|---|---|
| 11 | No | Small | 55K | ? |
| 12 | Yes | Medium | 80K | ? |
| 13 | Yes | Large | 110K | ? |
| 14 | No | Small | 95K | ? |
| 15 | No | Large | 67K | ? |

**Figure 4.3.** General approach for building a classification model.

# Examples of Classification Task

- Predicting tumor cells as benign or malignant.
- Classifying credit card transactions as legitimate or fraudulent.
- Categorizing news stories as finance, weather, entertainment, sports, etc.





# Topics

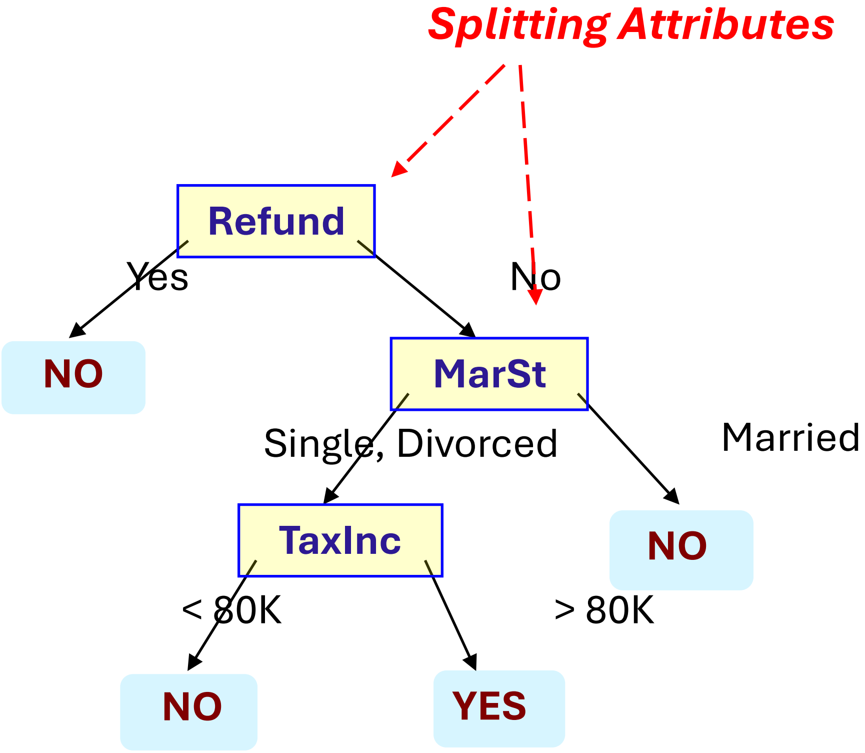
- Introduction
- **Decision Trees**
  - —**Overview**
  - —Tree Induction
- Overfitting and other Practical Issues
- Model Selection and Evaluation
  - —Metrics for Performance Evaluation
  - —Methods to Obtain Reliable Estimates
  - —Model Comparison (Relative Performance)
- Feature Selection

# Example of a Decision Tree

| | 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** |

**Training Data**

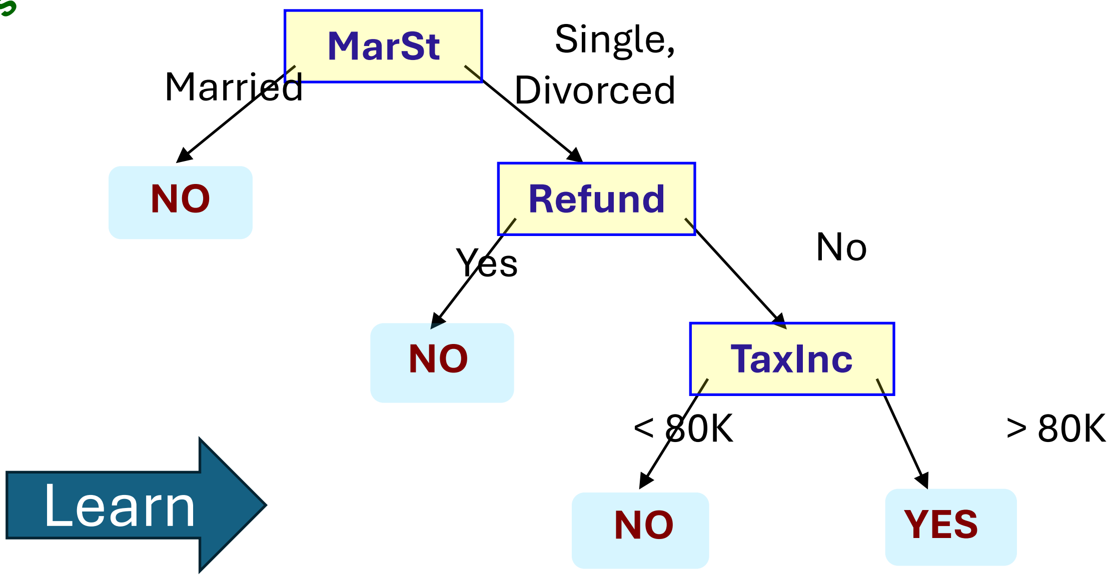
Learn



**Model: Decision Tree**

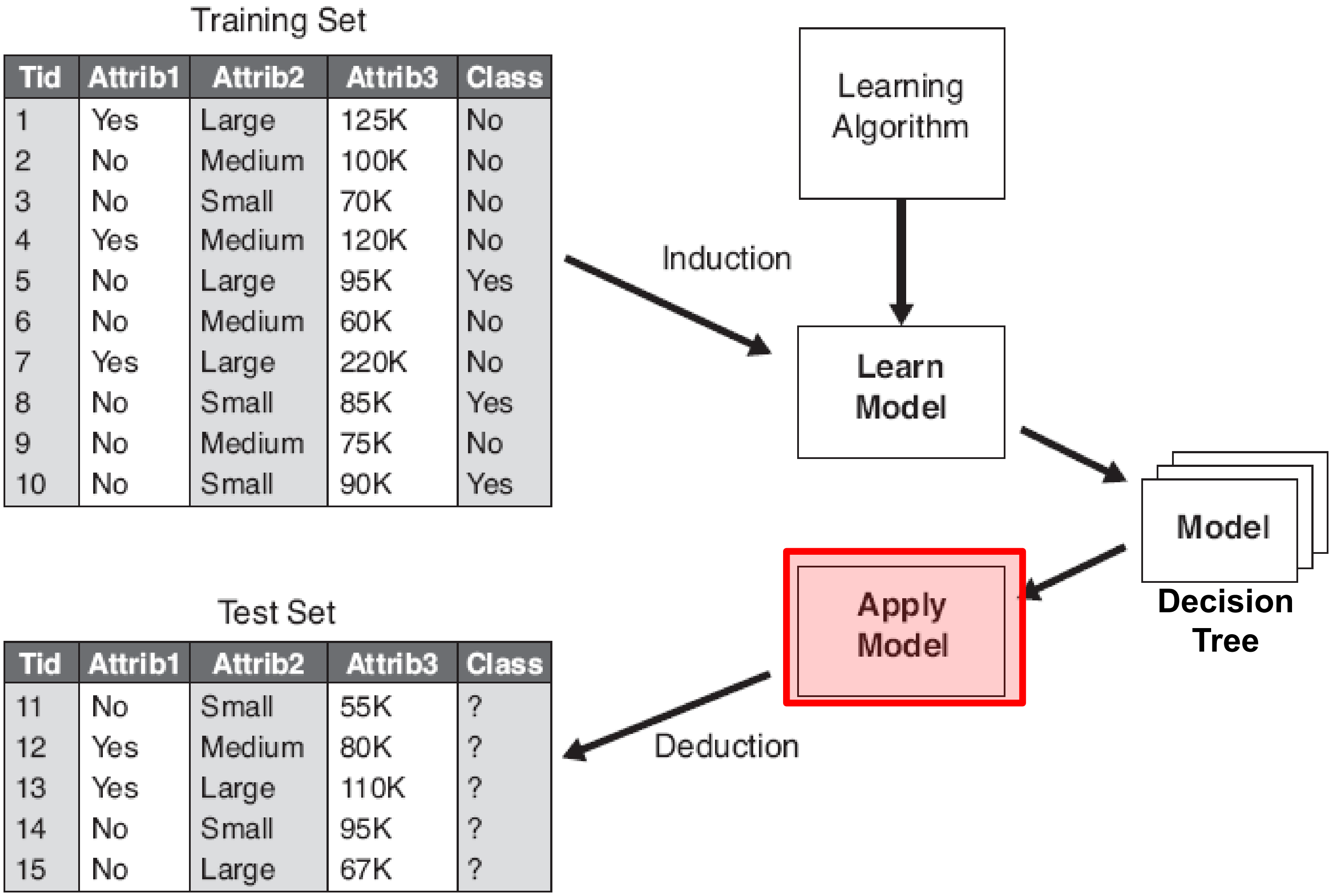
# Another Example of Decision Tree

| | 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** |



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

# Decision Tree: Deduction



Training Set

| 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 |

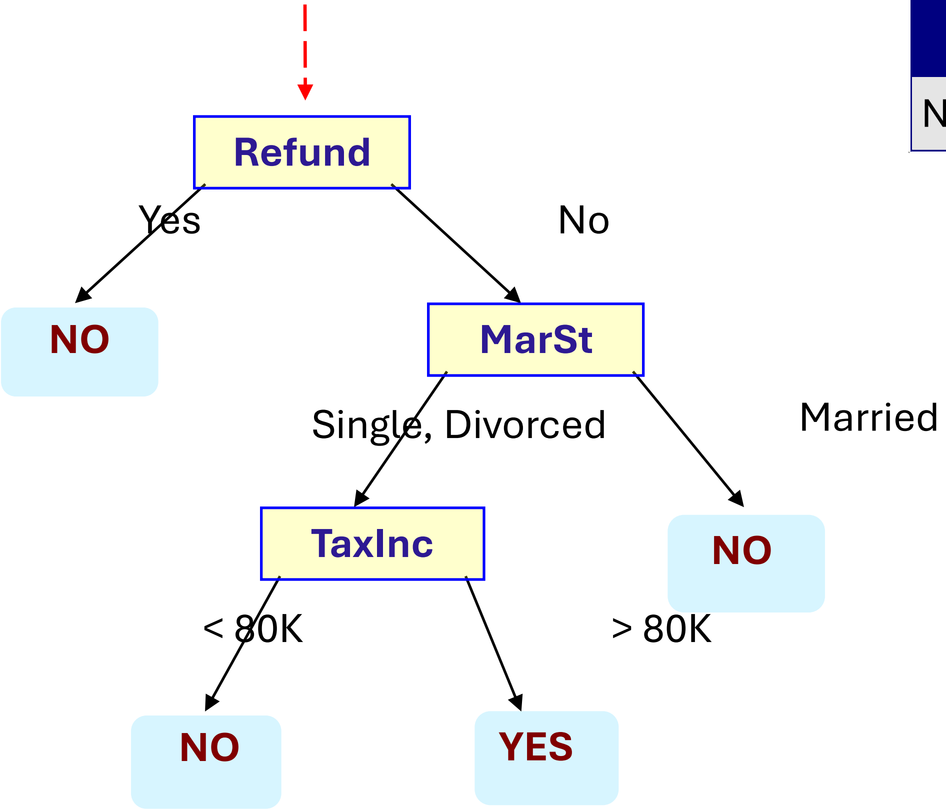
Test Set

| 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 | ? |

**Figure 4.3.** General approach for building a classification model.

# Apply Model to Test Data

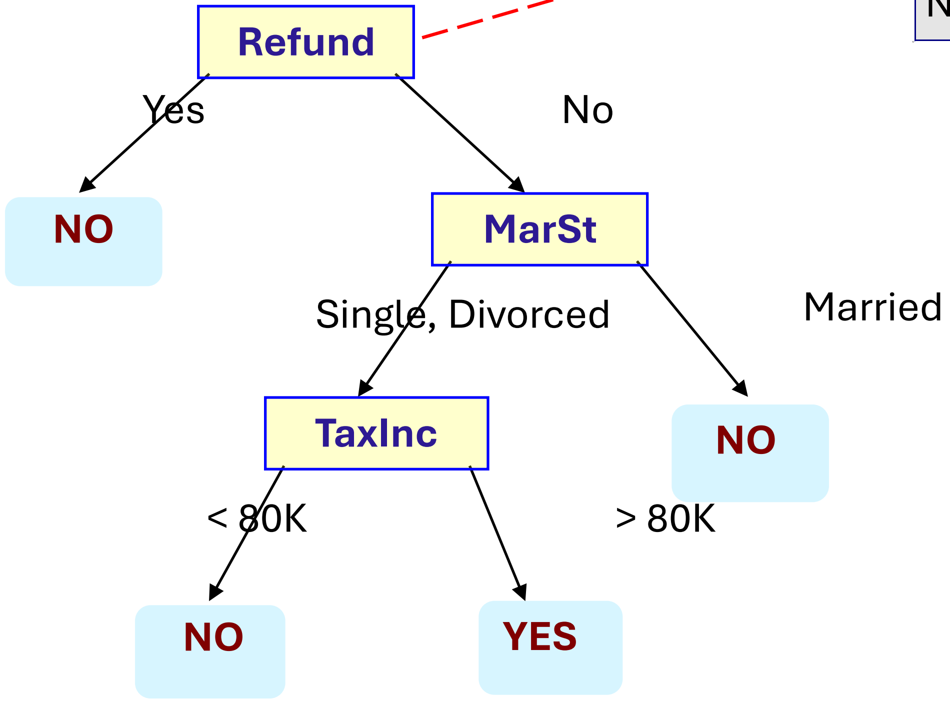
Start from the root of tree.



**Test Data**

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

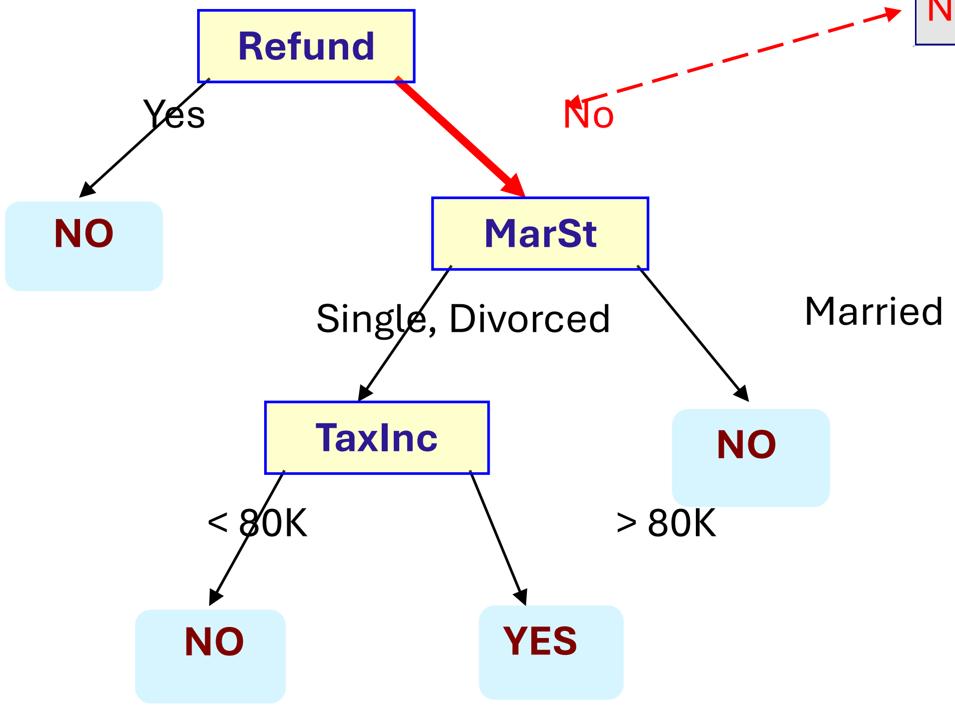
# Apply Model to Test Data



**Test Data**

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

# Apply Model to Test Data



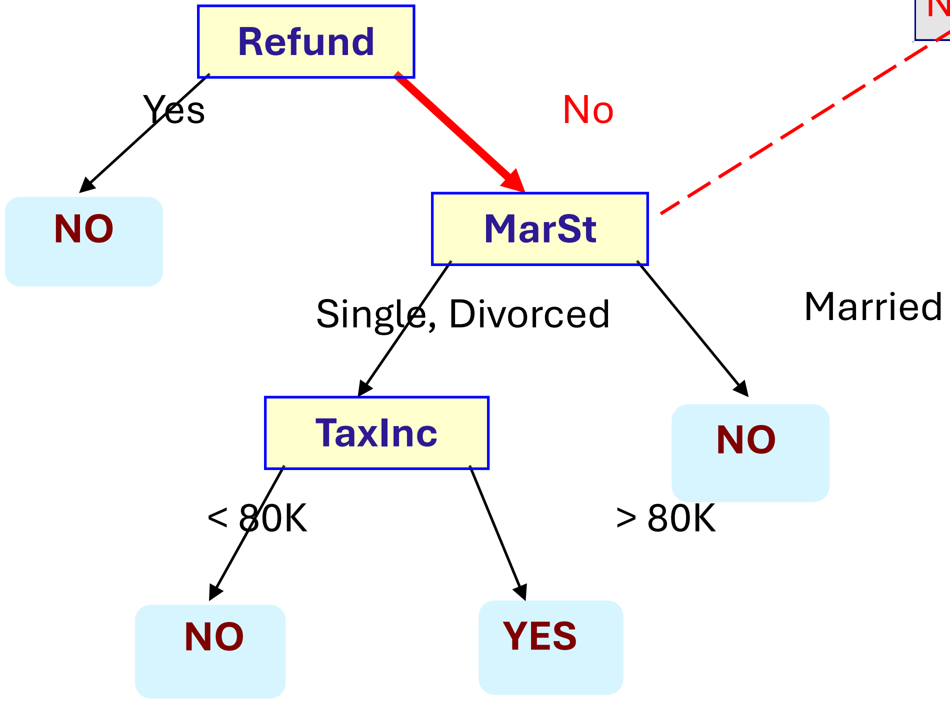
**Test Data**

| 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 | ? |



Refund

Yes

No

NO

MarSt

Single, Divorced

Married

TaxInc

NO

< 80K

> 80K

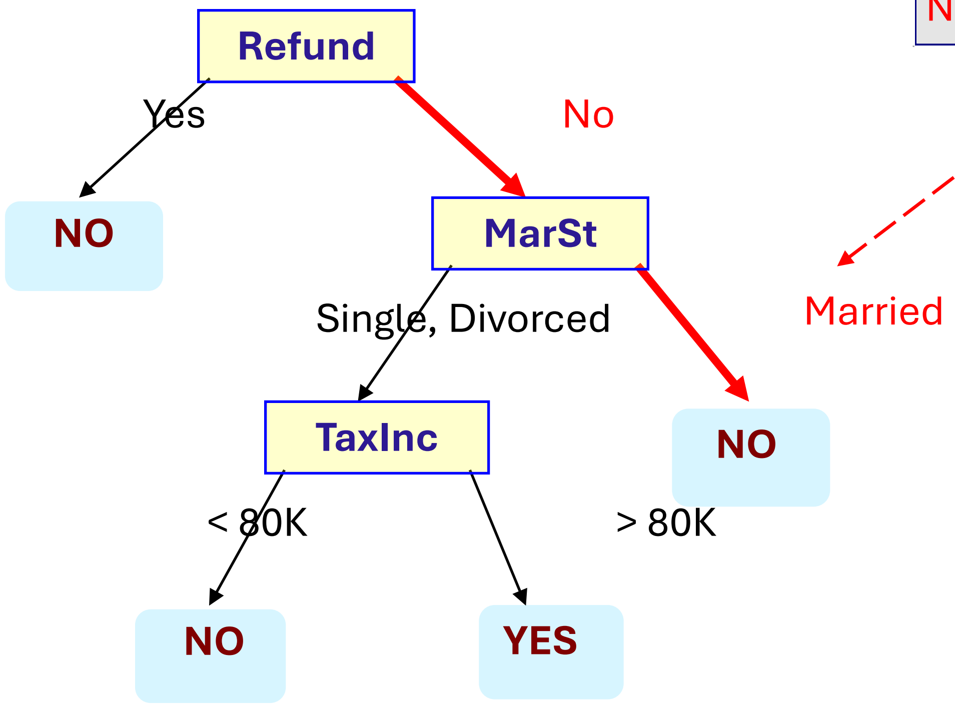
NO

YES

# Apply Model to Test Data

**Test Data**

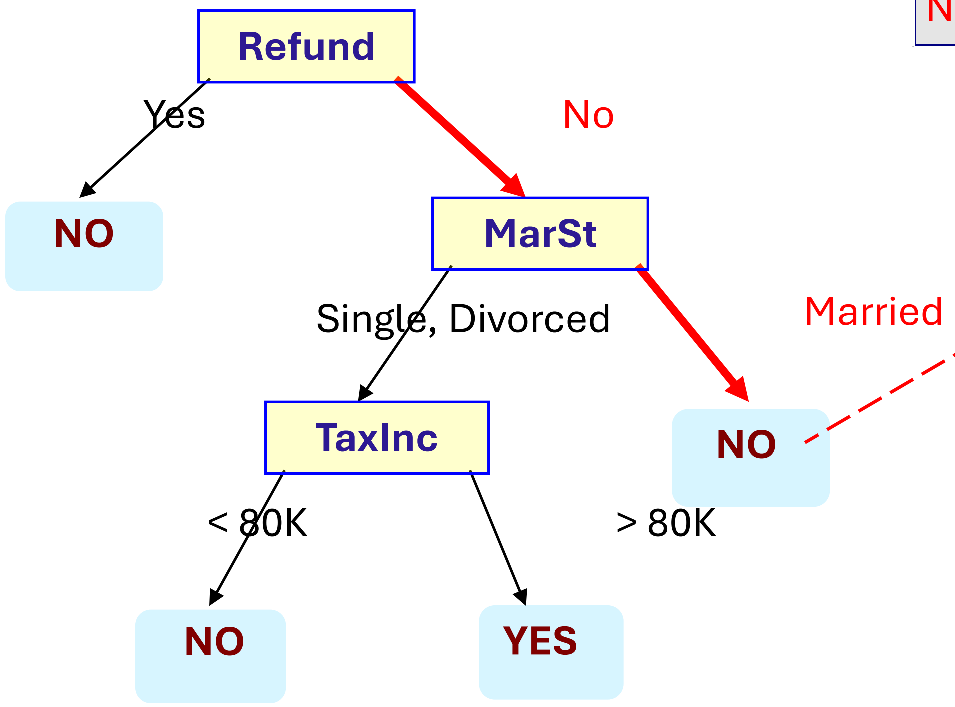
| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |



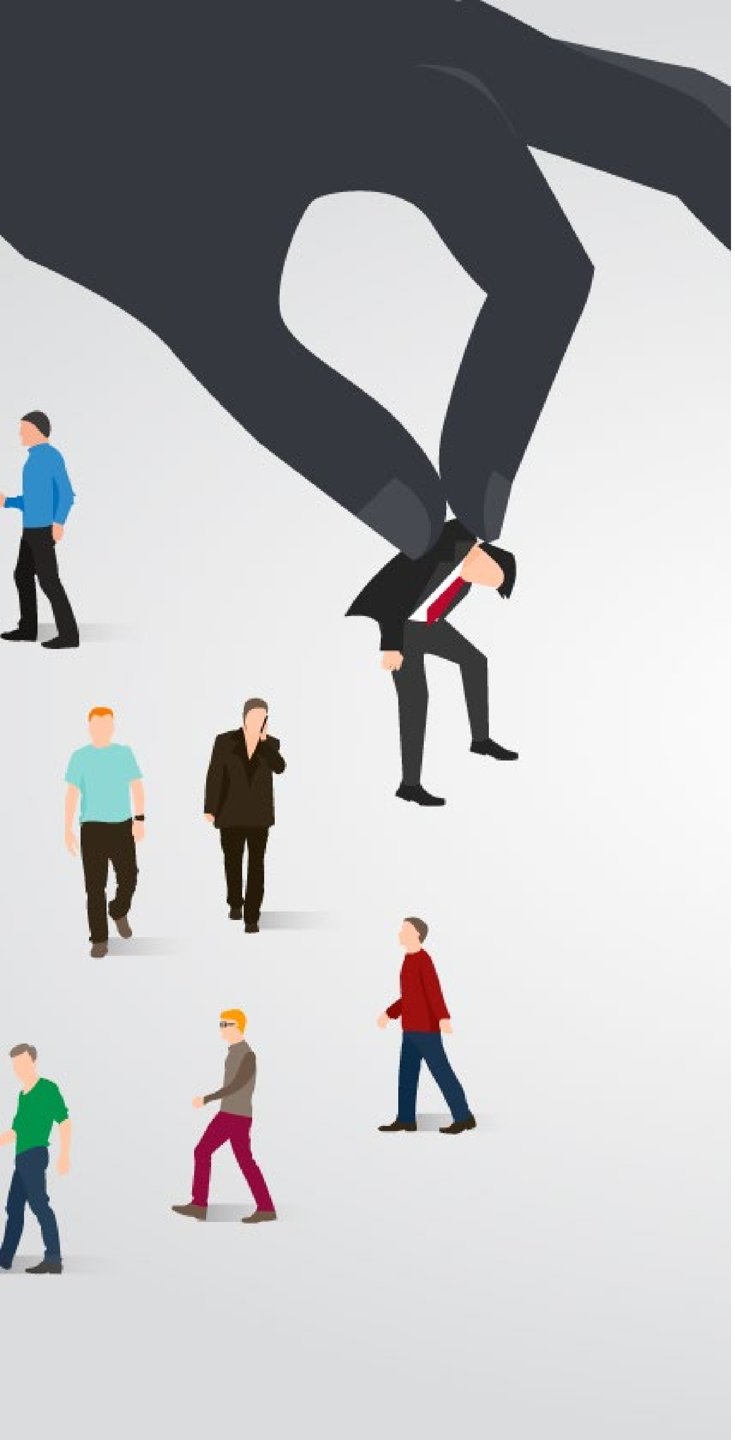
# Apply Model to Test Data

**Test Data**

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |



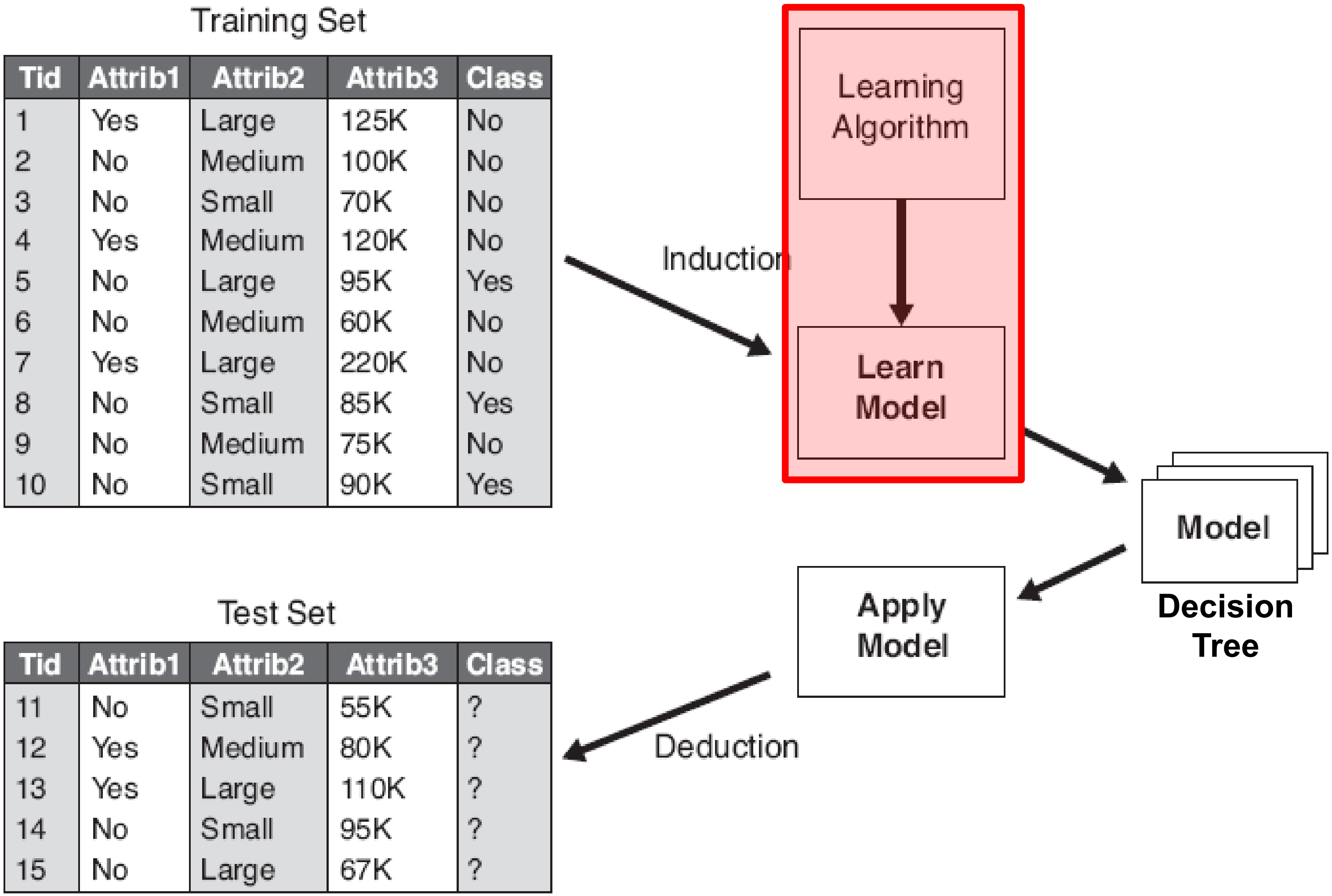
Assign Cheat to “No”



# Topics

- Introduction
- Decision Trees
  - —Overview
  - —**Tree Induction**
- Overfitting and other Practical Issues
- Model Selection and Evaluation
  - —Metrics for Performance Evaluation
  - —Methods to Obtain Reliable Estimates
  - —Model Comparison (Relative Performance)
- Feature Selection

# Decision Tree: Induction



Training Set

| 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 |

Test 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 | ? |

**Figure 4.3.** General approach for building a classification model.

# Decision Tree Induction

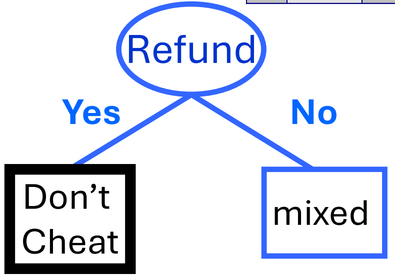
Many Algorithms:

- Hunt's Algorithm (one of the earliest)
- CART (Classification And Regression Tree)
- ID3, C4.5, C5.0 (by Ross Quinlan, introduced information gain)
- CHAID (CHi-squared Automatic Interaction Detection)
- MARS (Improvement for numerical features)
- SLIQ, SPRINT
- Conditional Inference Trees (recursive partitioning using statistical tests)

All algorithms use a simple, greedy top-down splitting strategy!

# The Effect of a Split

| *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 |



| *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~~ |

No:
3
Yes:
0

No:
4
Yes:
3

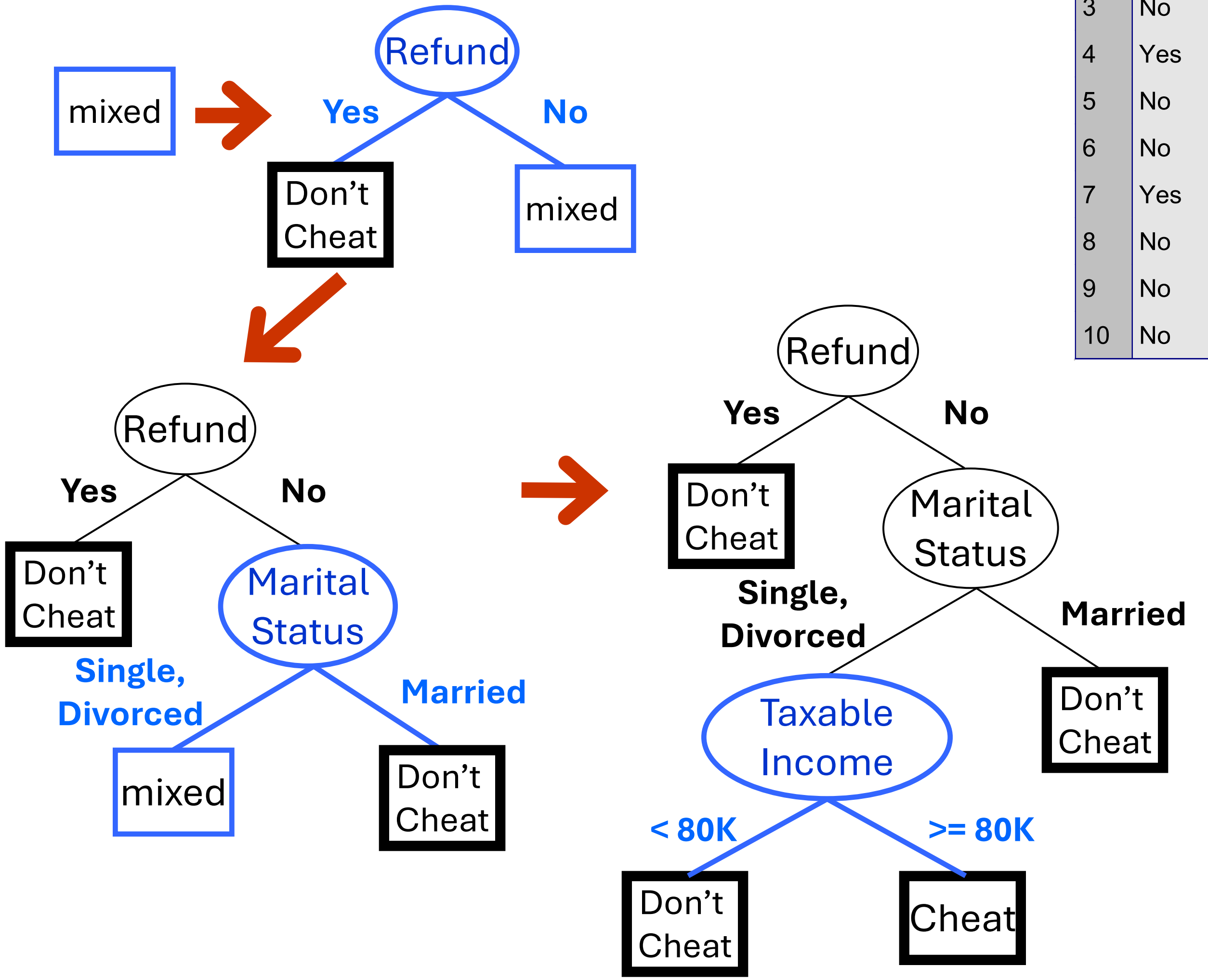
| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| ~~1~~ | ~~Yes~~ | ~~Single~~ | ~~125K~~ | ~~No~~ |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| ~~4~~ | ~~Yes~~ | ~~Married~~ | ~~120K~~ | ~~No~~ |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married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

Every split partitions the data set into two subsets.

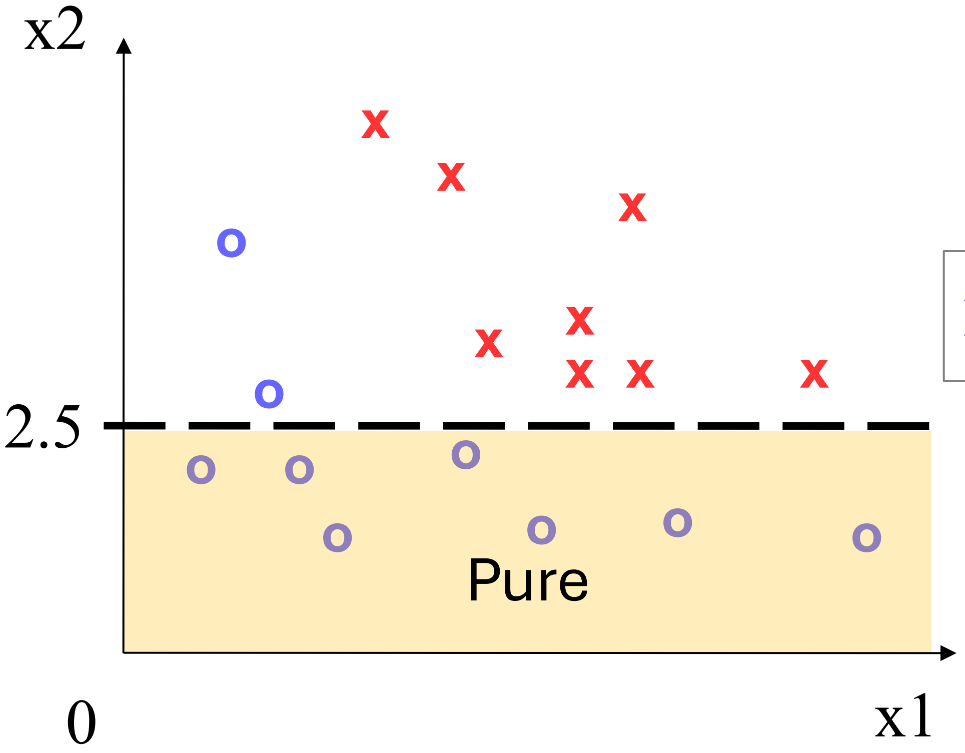
# Hunt’s Algorithm

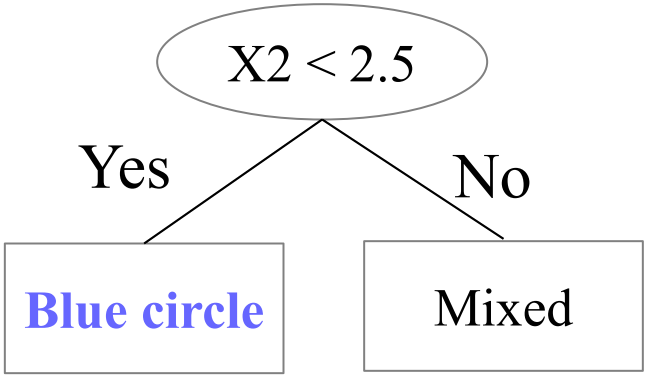
"Use attributes to split the data recursively, till each split contains only a single class."

| *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 |



# Example: Creating a Decision Tree





X2 < 2.5

Yes

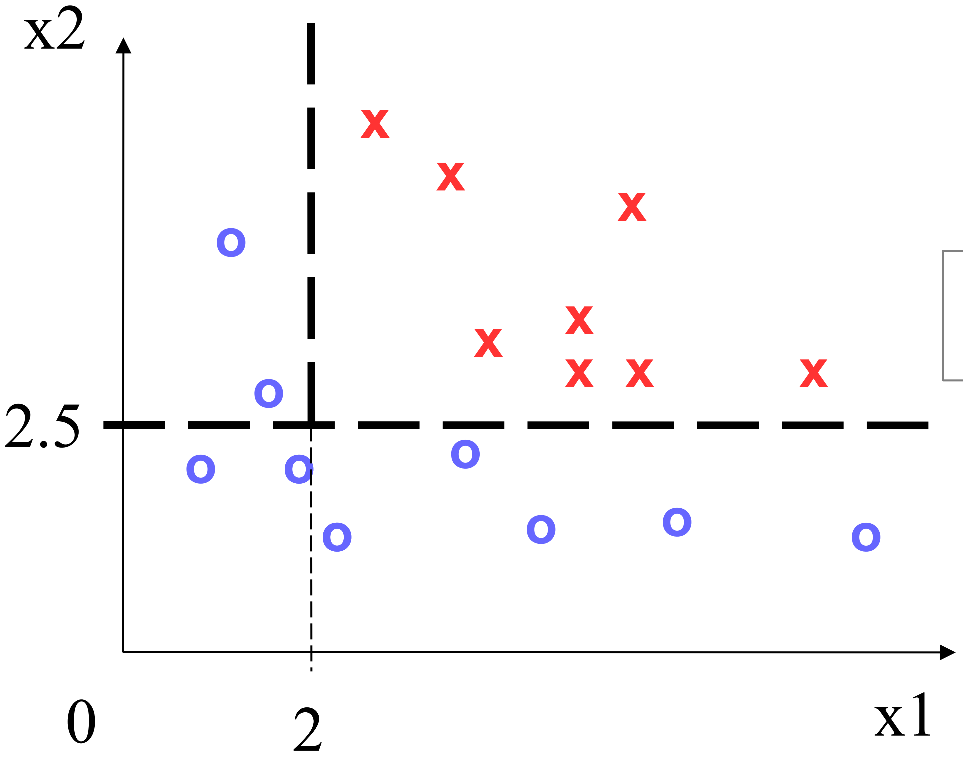
No

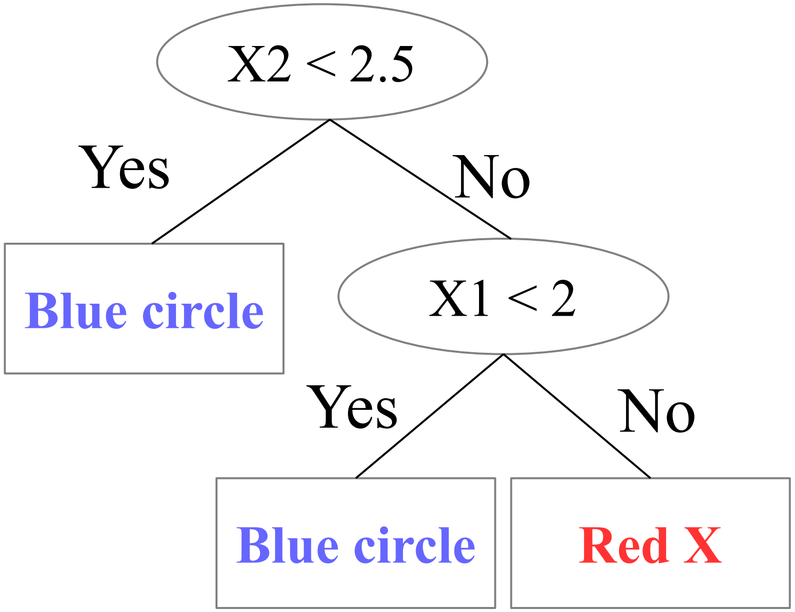
**Blue circle**

Mixed

Decision trees can only cut parallel to an axis!

# Example: Creating a Decision Tree





X2 < 2.5

Yes

No

Blue circle

X1 < 2

Yes

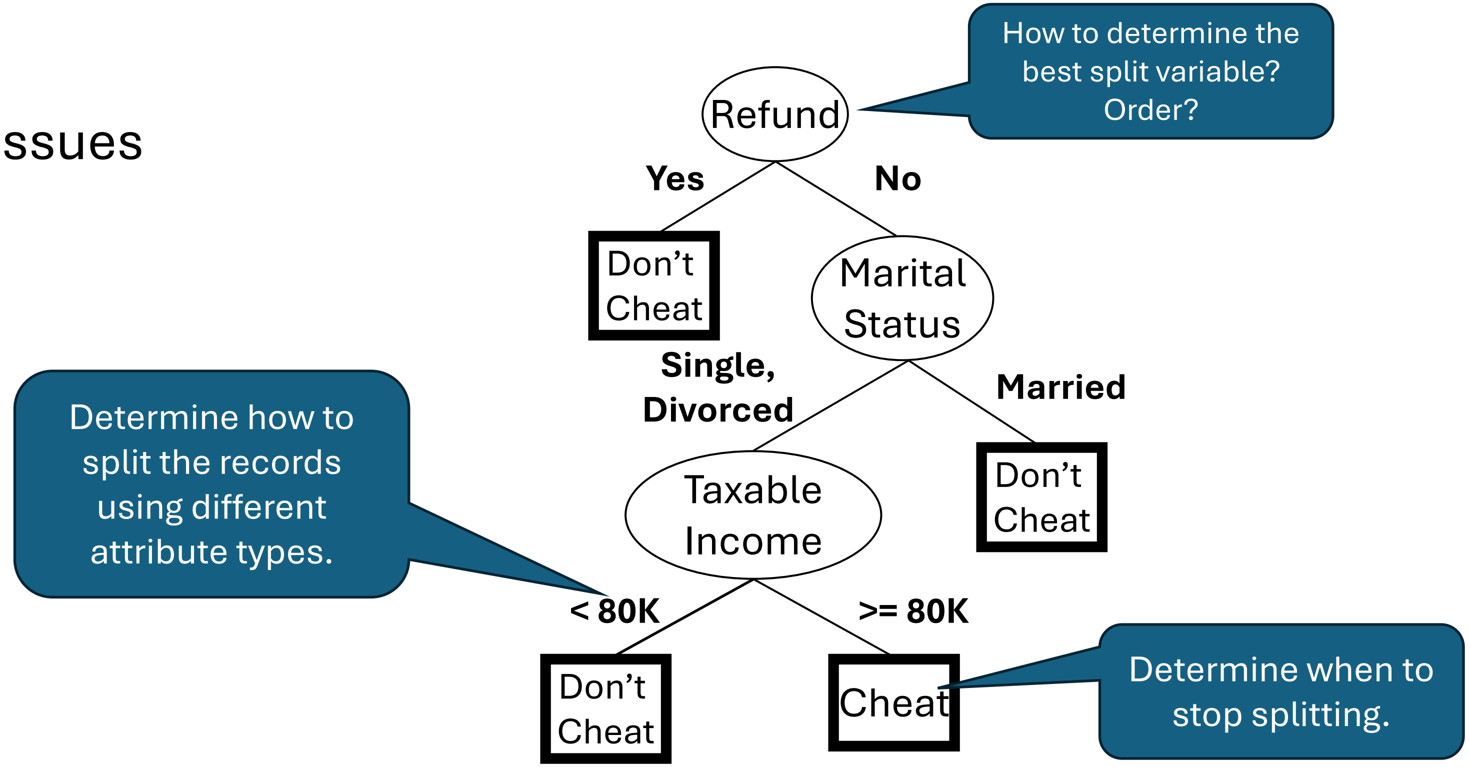
No

Blue circle

Red X

# Tree Induction

- Greedy strategy
  - Split the records based on an attribute test that optimizes a certain criterion.
- Issues



How to determine the best split variable? Order?

Determine how to split the records using different attribute types.

Determine when to stop splitting.

# Tree Induction

- Greedy strategy
  - Split the records based on an attribute test that optimizes a certain criterion.

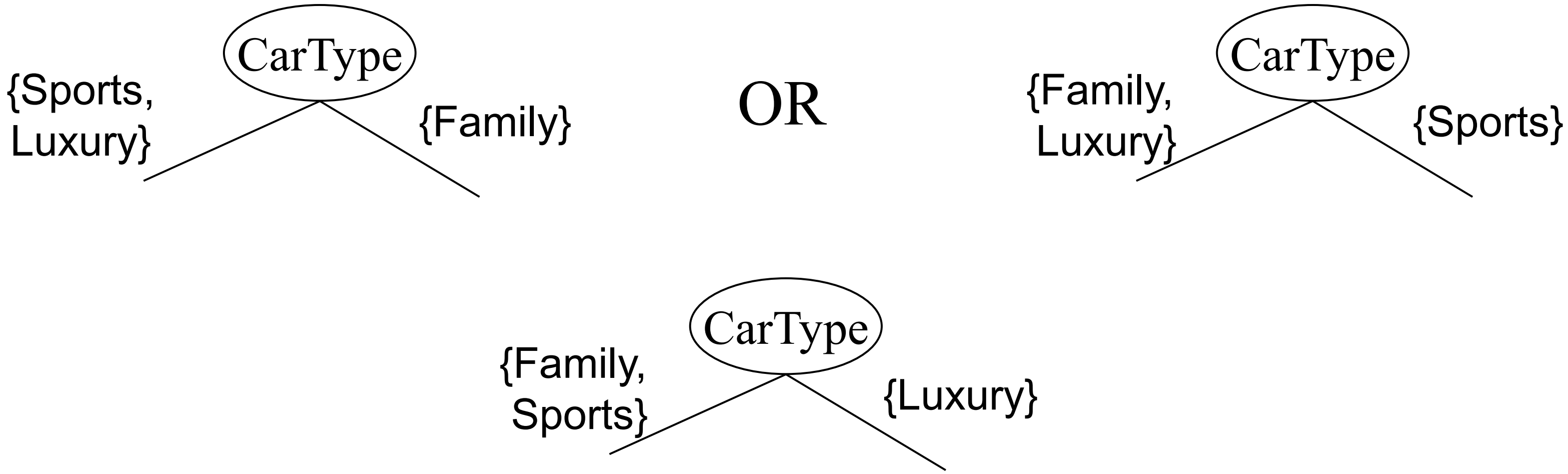
- Issues
  - **Determine how to split the records using different attribute types.**
  - How to determine the best split variable?
  - Determine when to stop splitting.

# How to Specify Test Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous (interval/ratio)

# Splitting Based on Nominal Attributes

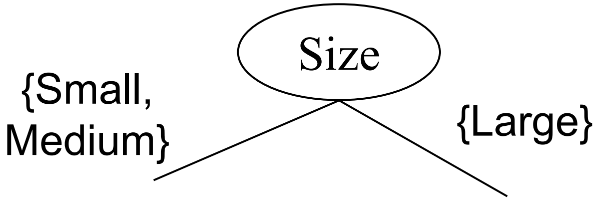
- Divide the unordered values into two subsets.
- We need to find optimal partitioning.



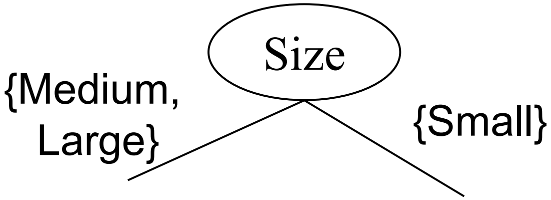
Best decision depends on what we want to predict!

# Splitting Based on Ordinal Attributes

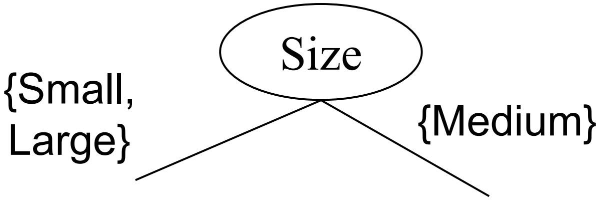
- Divide the ordered values into two subsets.



OR

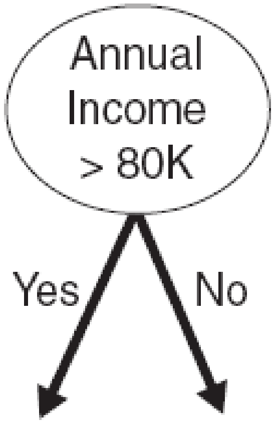


- What about this split?

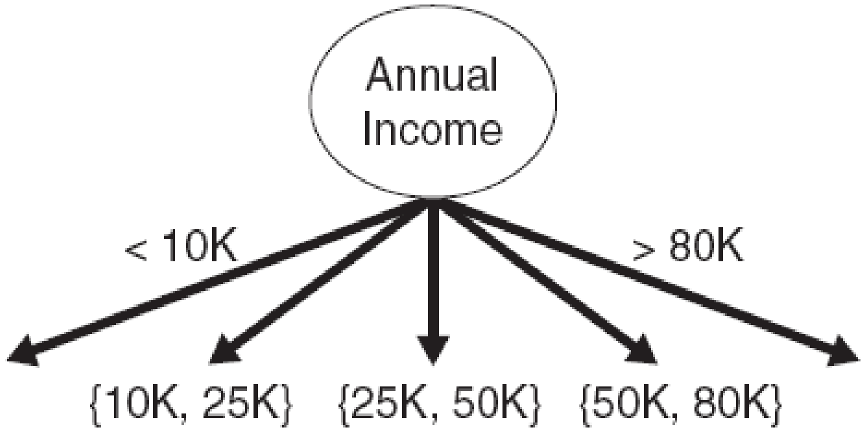


# Splitting Based on Continuous Attributes

Binary split



Multi-way split



Discretization to form an ordinal categorical attribute:

- **Static** – discretize the data set once at the beginning (equal interval, equal frequency, etc.).
- **Dynamic** – discretize during the tree construction.
  - Example: For a binary decision ($A < v$) or ($A \geq v$) consider all possible splits and finds the best cut. This can be done efficiently.

# Tree Induction

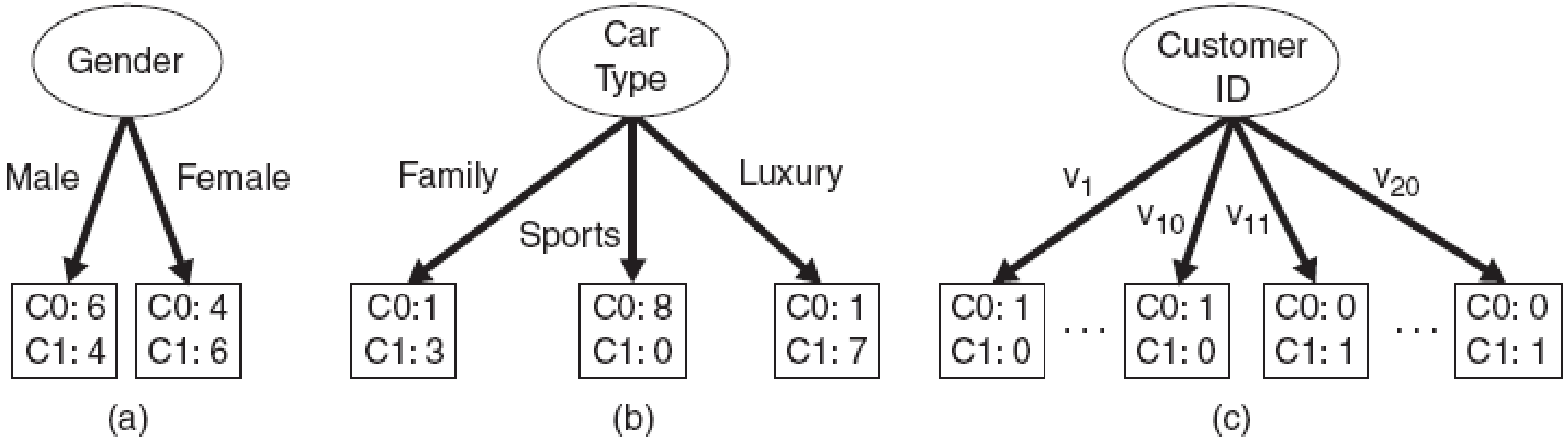
- Greedy strategy
  - Split the records based on an attribute test that optimizes a certain criterion.

- Issues
  - Determine how to split the records using different attribute types.
  - **How to determine the best split variable?**
  - Determine when to stop splitting

# How to determine the Best Split

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

C0: 10
C1: 10



**Which splitting variable is the best?**

# Determine the Quality of a Node: Node Impurity

- Nodes represent a subset of data that satisfy the splitting condition.
- We want to create nodes with homogeneous class distributions.
- Need a measure of node impurity:

| | |
|---|---|
| C0: | **5** |
| C1: | **5** |

**Non-homogeneous,**
**High degree of impurity**

This is preferred

| | |
|---|---|
| C0: | **9** |
| C1: | **1** |

**Homogeneous,**
**Low degree of impurity**

- General rule for measures of impurity:
  - —Smaller is better.
  - —0 represents the complete purity.

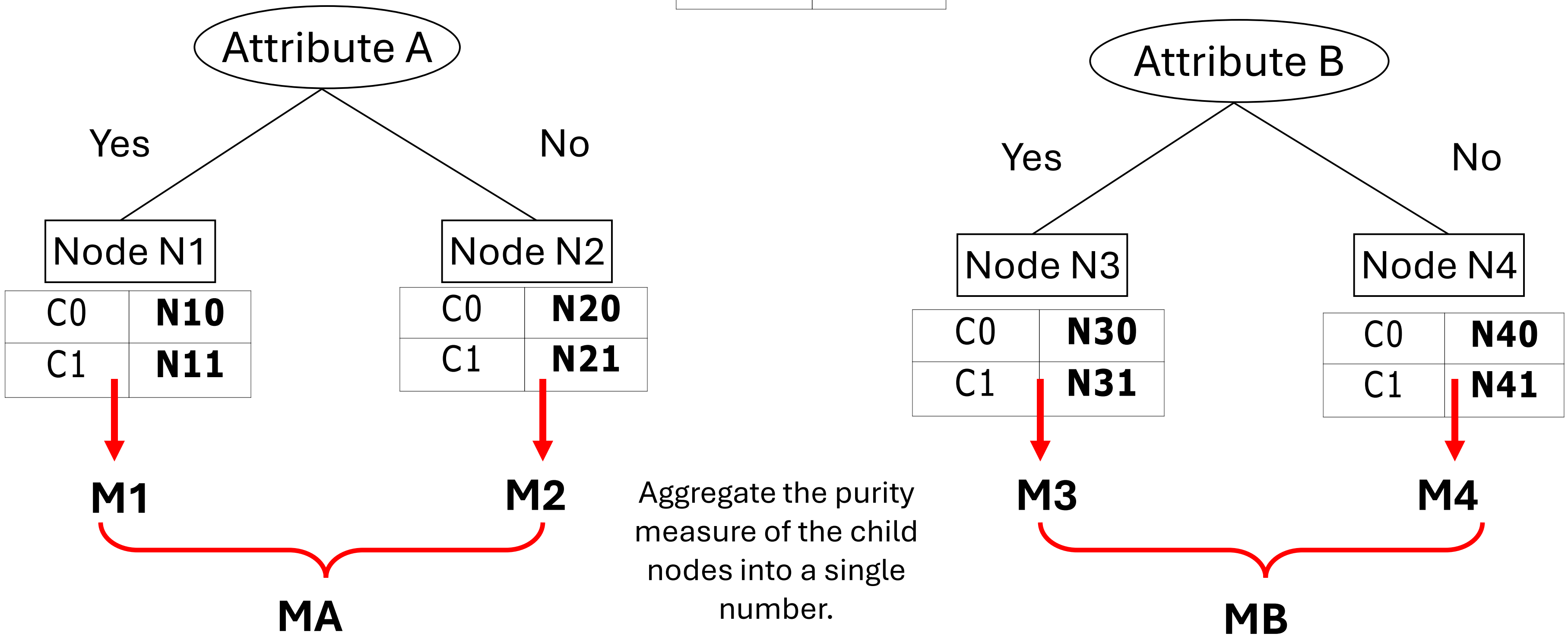
# Find the Best Split: General Framework

Assume we have a measure **M** that tells us how "pure" a node is.

**Before Splitting:**

| | |
|---|---|
| C0 | **N00** |
| C1 | **N01** |

→ **M0**



Aggregate the purity measure of the child nodes into a single number.

We look at the improvement called the gain:

**Gain = M0 – MA vs. M0 – MB** → Choose best split

# Measures of Node Impurity

**Gini Index**

Entropy

Classification error

# Measure of Impurity: Gini Index of a Node

- Gini Index for a given node t :

$$GINI(t) = \sum_{j} p(j\,|t)(1 - p(j\,|t)) = 1 - \sum_{j} p(j\,|t\,)^2$$

$p(\,j\,|\,t)$ is estimated as the relative frequency of class j at node t

- Origin: The Gini index is a measure of statistical dispersion intended to represent the income inequality within nations. Here it is used as a statistical measure that quantifies how mixed or impure the class distribution in a node is.

- Maximum Impurity: $1 - 1/n_c$ (number of classes) when records are equally distributed among all classes. For a binary decision it is 0.5.
- Minimum Impurity: 0 when all records belong to one class.

- Examples:

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

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

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

| | |
|---|---|
| C1 | **3** |
| C2 | **3** |
| **Gini=0.500** | |

# Examples: Gini Index of a Node

$$GINI(t) = 1 - \sum_j p(j \mid t)^2$$

| | |
|---|---|
| C1 | **0** |
| C2 | **6** |

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

Gini = 1 – $P(C1)^2$ – $P(C2)^2$ = 1 – 0 – 1 = **0**

| | |
|---|---|
| C1 | **1** |
| C2 | **5** |

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

Gini = 1 – $(1/6)^2$ – $(5/6)^2$ = **0.278**

| | |
|---|---|
| C1 | **2** |
| C2 | **4** |

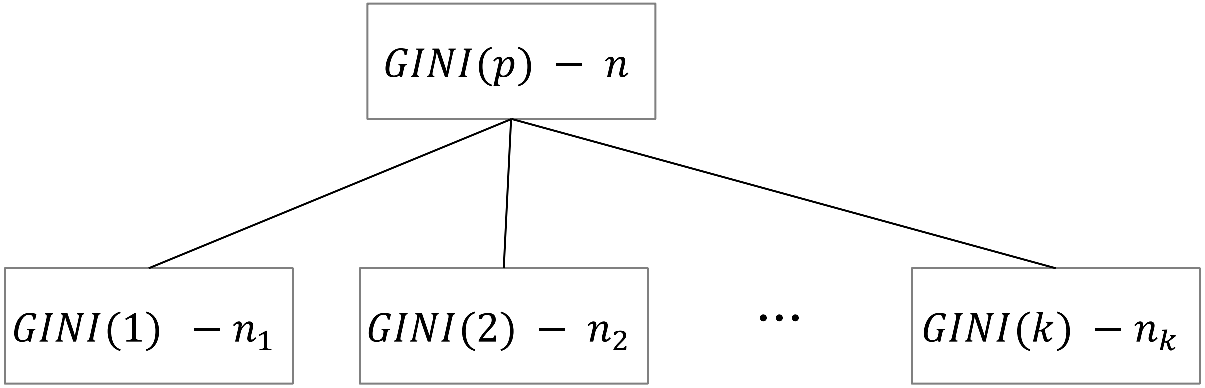
P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – $(2/6)^2$ – $(4/6)^2$ = **0.444**

Maximal impurity here is ½ = .5

# Splitting Based on the Gini Index

When a node p is split into k partitions (children), the quality of the split is computed as a weighted sum:



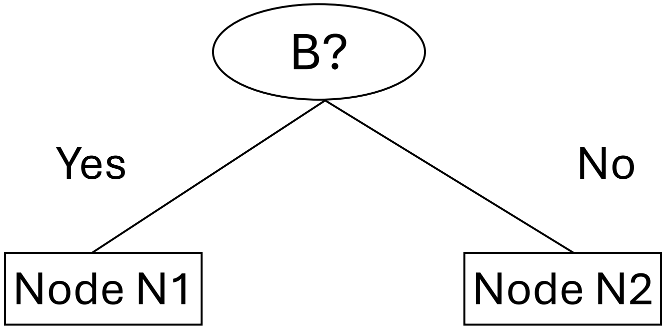
$$\boxed{GINI_{split} = \sum_{i}^{k} \frac{n_i}{n} GINI(i)}$$

where $n_i$ is the number of records at child $i$,
and $n$ is the number of records at node $p$.

Used in the algorithms CART, SLIQ, SPRINT.

# Example: Splitting based on the Gini Index

- Effect of weighing partitions: Larger **and** purer partitions are preferred.



|  | **Parent** |
|---|---|
| C1 | **6** |
| C2 | **6** |
| **Gini = 0.500** |  |

Gini(N1)
$= 1 - (5/8)^2 - (3/8)^2$
$= 0.469$

Gini(N2)
$= 1 - (1/4)^2 - (3/4)^2$
$= 0.375$

|  | **N1** | **N2** |
|---|---|---|
| C1 | **5** | **1** |
| C2 | **3** | **3** |
| **Gini=0.438** |  |  |

Gini of the split
$= 8/12 * 0.469 +$
$4/12 * 0.375$
$= 0.438$

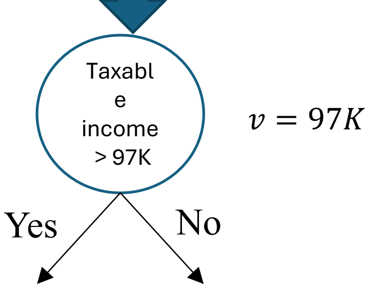
Gain = 0.5 – 0.438
= 0.062

**GINI improves!**

# Continuous Attributes: Computing Gini Index

- How does the algorithm choose the splitting value $v$? (= dynamic discretization)
  - —Number of possible splitting values = Number of distinct values
- Efficient Method: 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 smallest Gini index

| *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 | 90 | Yes |



$v = 97K$

| 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 | 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 | |

# Measures of Node Impurity

Gini Index

**Entropy**

Classification error

# Measure of Impurity: Entropy

- Entropy at a given node t:

$$\boxed{\text{Entropy(t)} = -\sum_{j} p(j \mid t) \log(p(j \mid t))}$$

$p(\, j \mid t)$ is the relative frequency of class j at node t;
$0 \log(0) \overset{\text{def}}{=} 0$ is used!

- Origin: In information theory, entropy quantifies the amount of uncertainty involved in the value of a random. Here the random variable is the class label of a randomly chosen observation in a node.

- Maximum Impurity: $\log(n_c)$ when records are equally distributed among all classes.
- Minimum Impurity: 0 when all records belong to one class. We can perfectly predict the class label of each observation in the node.

# Examples: Entropy

$$\text{Entropy(t)} = -\sum_{j} p(j \mid t) \log(p(j \mid t))$$

| | |
|---|---|
| C1 | **0** |
| C2 | **6** |

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

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

| | |
|---|---|
| C1 | **1** |
| C2 | **5** |

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

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

| | |
|---|---|
| C1 | **3** |
| C2 | **3** |

P(C1) = 3/6 P(C2) = 3/6

Entropy = – (3/6) $\log_2$ (3/6) – (3/6) $\log_2$ (3/6) = 1

# Splitting based on Information Gain

$$\boxed{GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)\right)}$$

Parent Node, p is split into k partitions;

$n_i$ is number of records in partition i

- Measures reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
- Used in ID3, C4.5 and C5.0
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

# Splitting based on the Gain Ratio

$$\boxed{GainRato_{split} = \frac{GAIN_{split}}{SplitInfo}}$$

$$SplitInfo = -\sum_{i=1}^{k} \frac{n_i}{n} log\left(\frac{n_i}{n}\right)$$

Parent Node, p is split into k partitions;
$n_i$ is 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 Information Gain.

# Measures of Node Impurity

Gini Index

Entropy

**Classification error**

# Splitting Criteria based on Classification Error

- Classification error at a node t :

$$\boxed{Error(t) = 1 - \max_{i} p(i\,|t)}$$

$p(\,j \mid t)$ is the relative frequency of class $j$ at node $t$

- Measures the classification error made in a node by a simple classifier that always predict the majority class (given by the $\max$ in the equation).

- Maximum Impurity: $1 - \frac{1}{n_c}$ when records are equally distributed among all classes (maximal error).
- Minimum Impurity: 0 when all records belong to one class = maximal purity (no error)

# Examples: Classification Error

$$\boxed{Error(t) = 1 - \max_{i} p(i\,|t)}$$

| | |
|---|---|
| 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 | **3** |
| C2 | **3** |

P(C1) = 3/6     P(C2) = 3/6

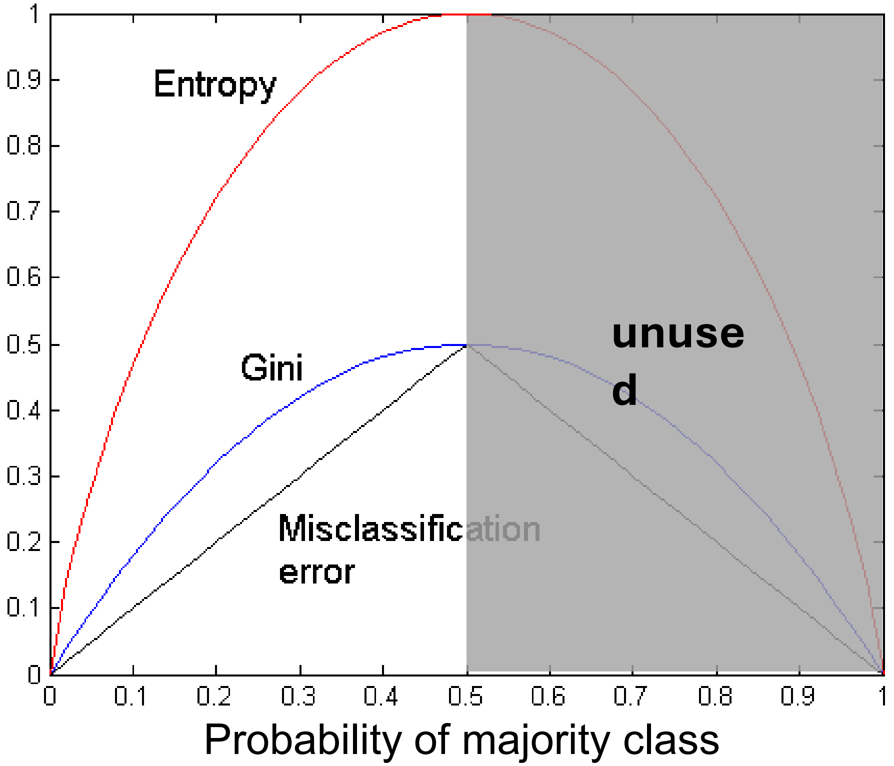
Error = 1 – max (3/6, 3/6) = 1 – 3/6 = .5

# Splitting based on the Classification Error

- Use weighted averages or gain as for the other indices to make the splitting decision.

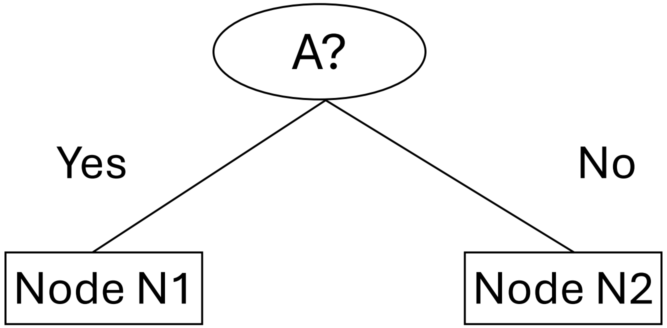
# Comparison among Splitting Criteria

For a 2-class problem: Probability of the majority class p is always > .5



**Note:** The order is the same no matter what splitting criterion is used, however, the gain (differences) are not since they depend on the slope.

# Classification Error vs Gini Index



|  | **Parent** |
|---|---|
| C1 | **7** |
| C2 | **3** |
| **Gini = 0.42**<br>**Error = 0.30** |  |

$\text{Gini}(N1) = 1 - (3/3)^2 - (0/3)^2 = 0$

$\text{Gini}(N2) = 1 - (4/7)^2 - (3/7)^2 = 0.489$

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

Error(N1) = 1-3/3=0

Error(N2)=1-4/7=3/7

Error(Split)= 3/10*0 + 7/10*3/7 = 0.3

|  | **N1** | **N2** |
|---|---|---|
| C1 | **3** | **4** |
| C2 | **0** | **3** |
| **Gini=0.342**<br>**Error = 0.30** |  |  |

**Gini improves!**

**Error does not!!!**

# Tree Induction

- Greedy strategy
  - Split the records based on an attribute test that optimizes a certain criterion.

- Issues
  - Determine how to split the record using different attribute types.
  - How to determine the best split?
  - **Determine when to stop splitting**

# Stopping Criteria for Tree Induction

- Stop expanding a node when **all the records belong to the same class** (used Hunt's algorithm).

- Stop expanding a node when all the records in the node have the **same attribute values**. Splitting becomes impossible.

- **Early termination criterion.** Stop when more splits will lead to overfitting the training data. We will discuss this later with tree pruning.

**Standar d method**

# Advantages of Decision Trees

INEXPENSIVE TO CONSTRUCT

EXTREMELY FAST AT CLASSIFYING UNKNOWN RECORDS

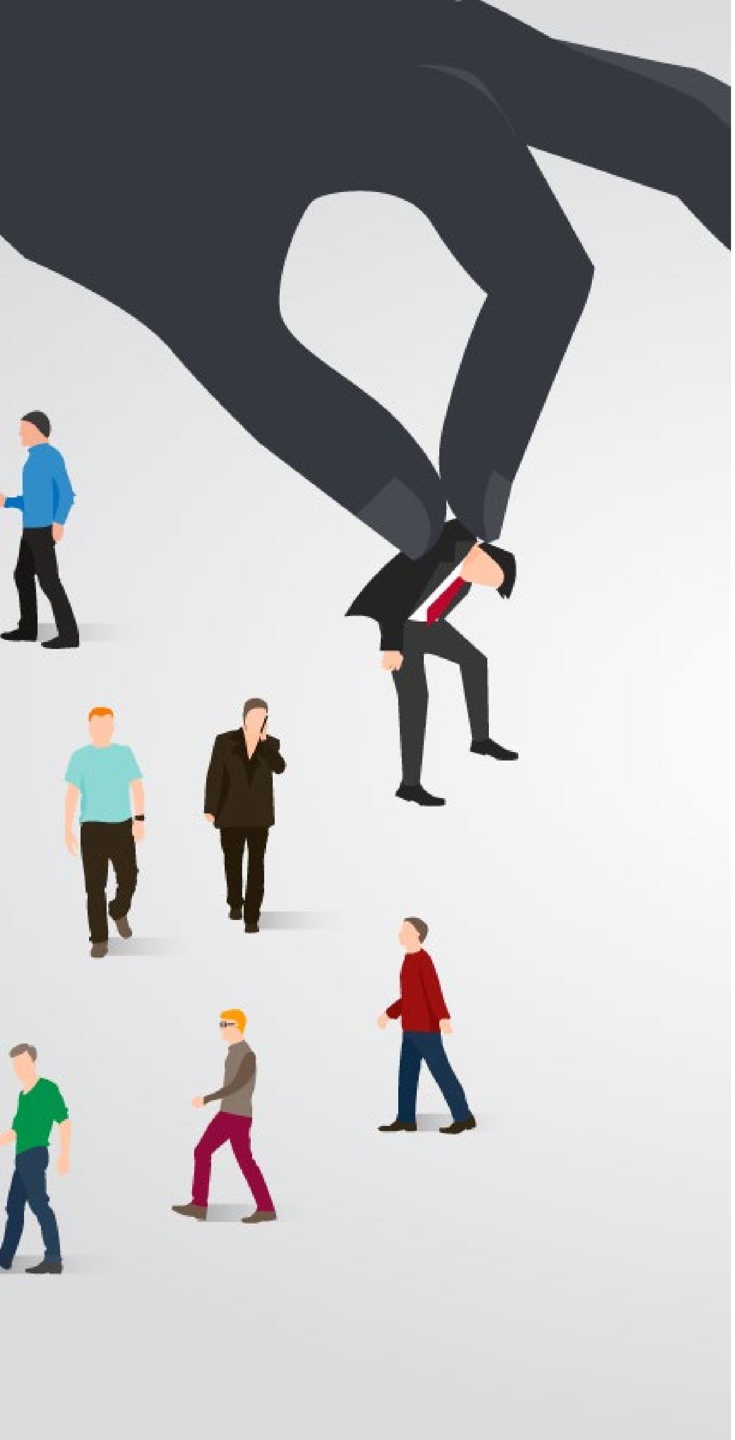
EASY TO INTERPRET FOR SMALL-SIZED TREES

ACCURACY IS COMPARABLE TO OTHER CLASSIFICATION TECHNIQUES FOR MANY SIMPLE DATA SETS

# Example: C4.5

- Simple depth-first construction.
- Uses Information Gain (improvement of the entropy measure).
- Handling both continuous and discrete attributes (continuous attributes are split at threshold).
- Needs entire data to fit in memory (unsuitable for large datasets).
- Final trees are pruned to remove branches that hurt performance.

- Code available at
  - http://www.cse.unsw.edu.au/~quinlan/c4.5r8.tar.gz
  - Open-Source implementation as J48 in Weka/rWeka



# Topics

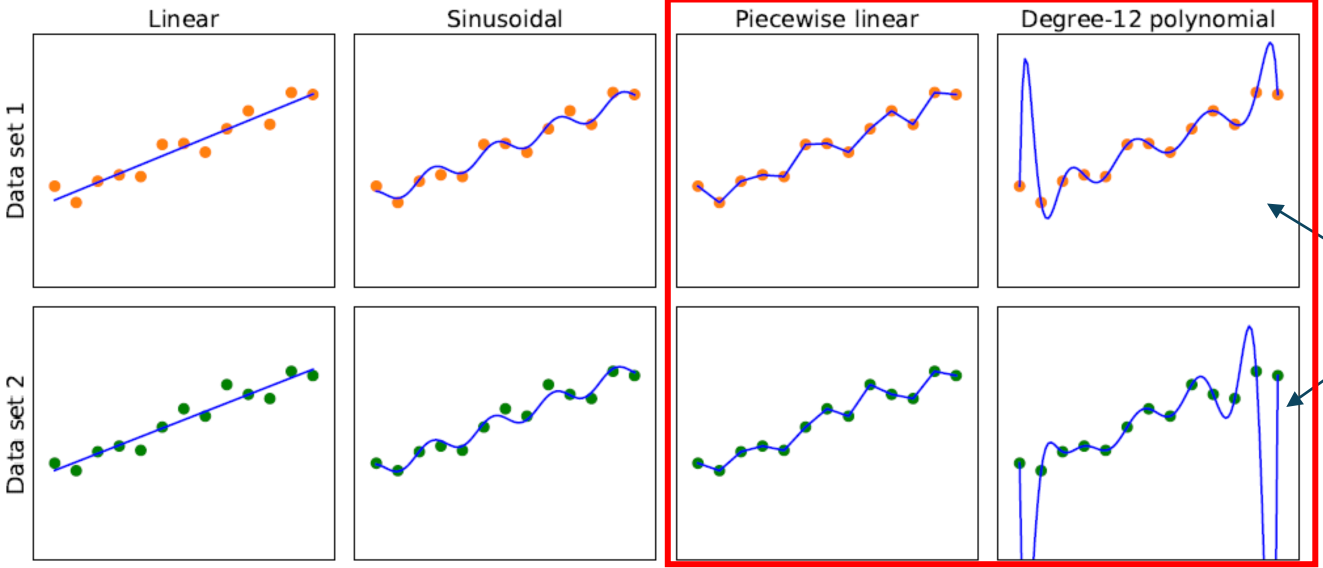
- Introduction
- Decision Trees
  - —Overview
  - —Tree Induction
- **Overfitting and other Practical Issues**
- Model Selection and Evaluation
  - —Metrics for Performance Evaluation
  - —Methods to Obtain Reliable Estimates
  - —Model Comparison (Relative Performance)
- Feature Selection

# Model Selection: Bias vs. Variance

Simplicity (fewer parameters)

Consistency (fit)

Overfitting to training data



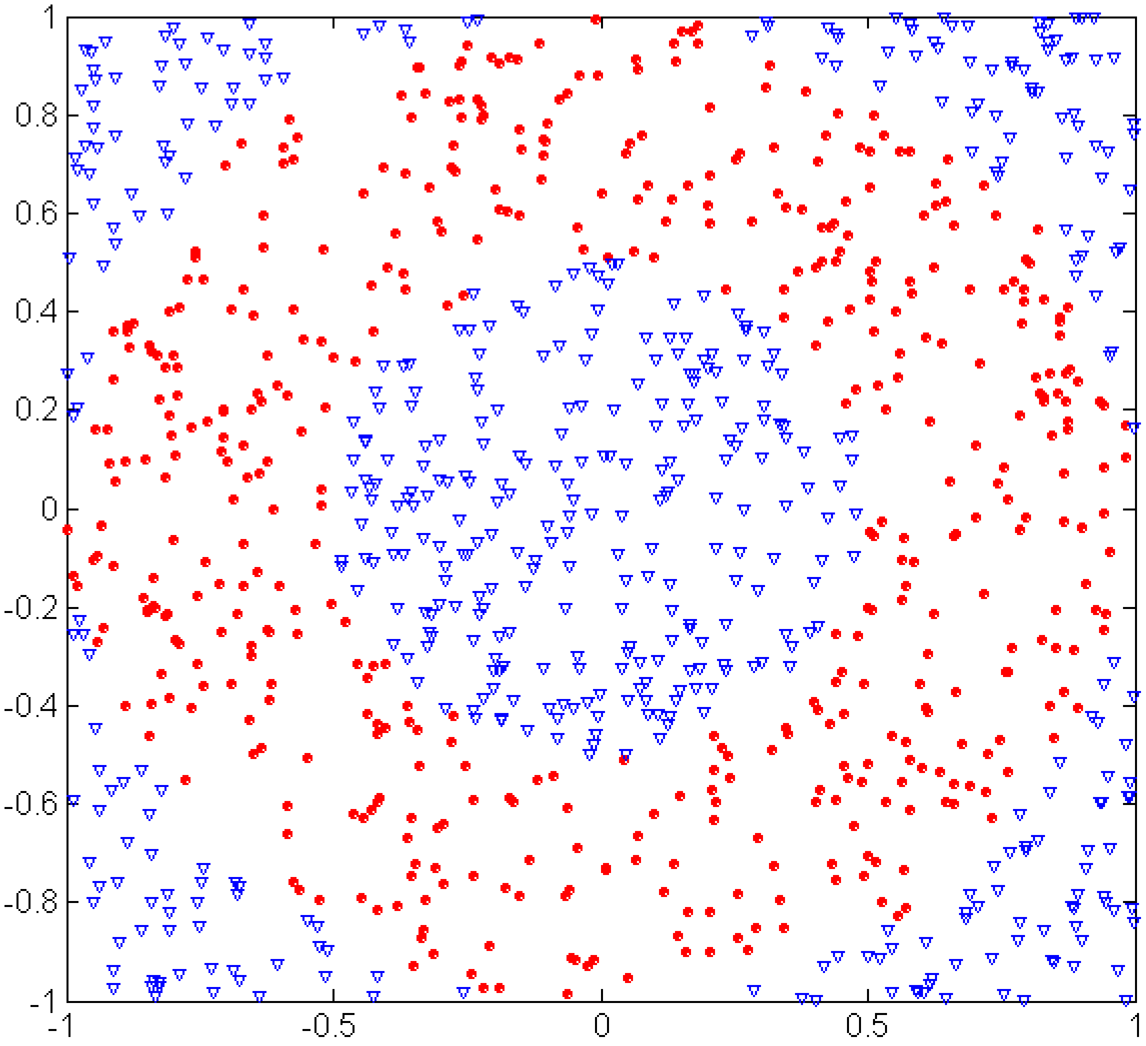
Two samples from the same function $f$ (points) with the learned function $h$ (lines).

High **Bias**: restrictions by the model class Low

Low **Variance:** difference in the model due to slightly different data. high

Note: This trade-off applies to any model.

# Example: Underfitting and Overfitting



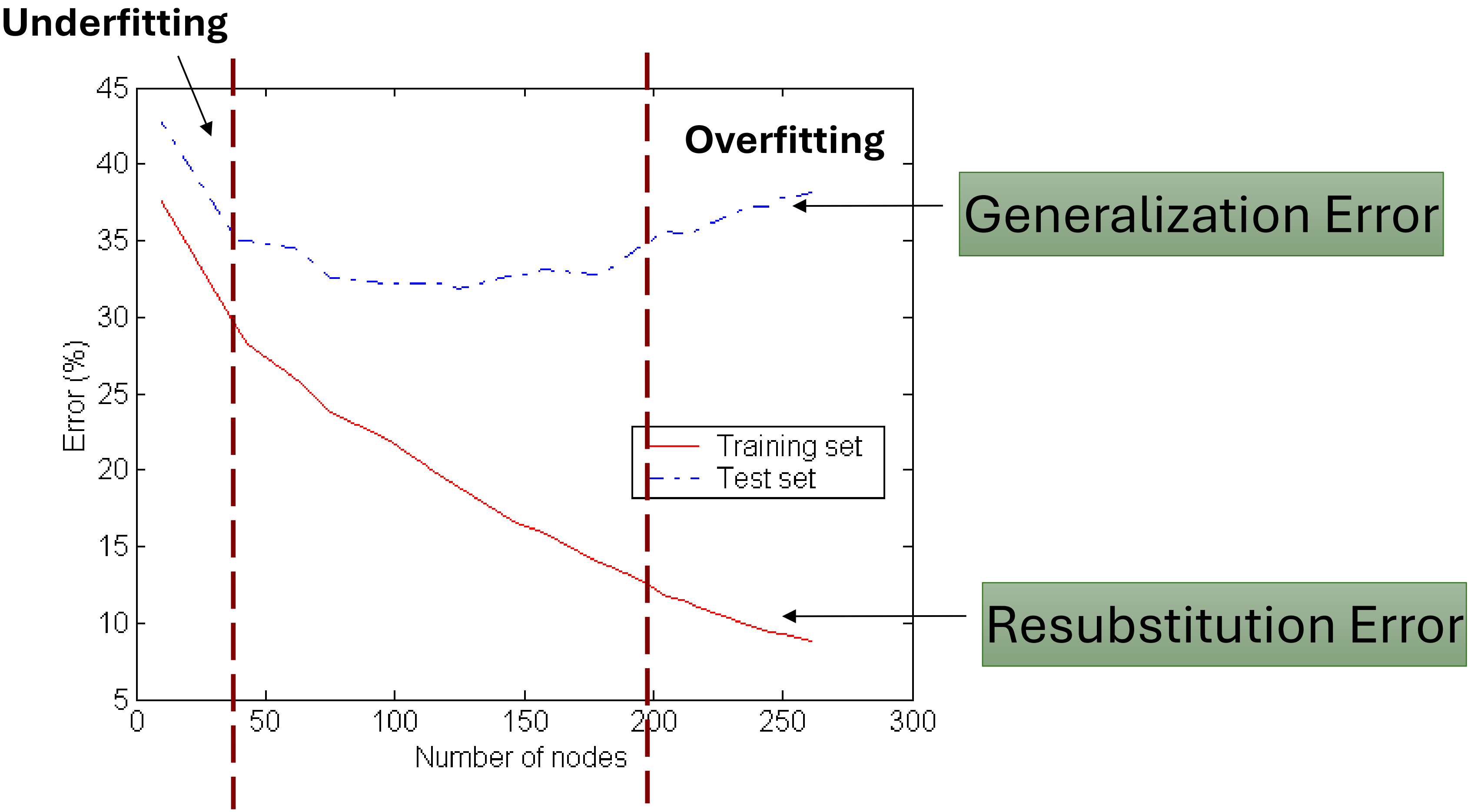
500 circular and 500 triangular data points.

Circular points:
$0.5 \geq \ sqrt(x_1{}^2 + x_2{}^2) \leq 1$

Triangular points:

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

# Example: Underfitting and Overfitting



**Underfitting**

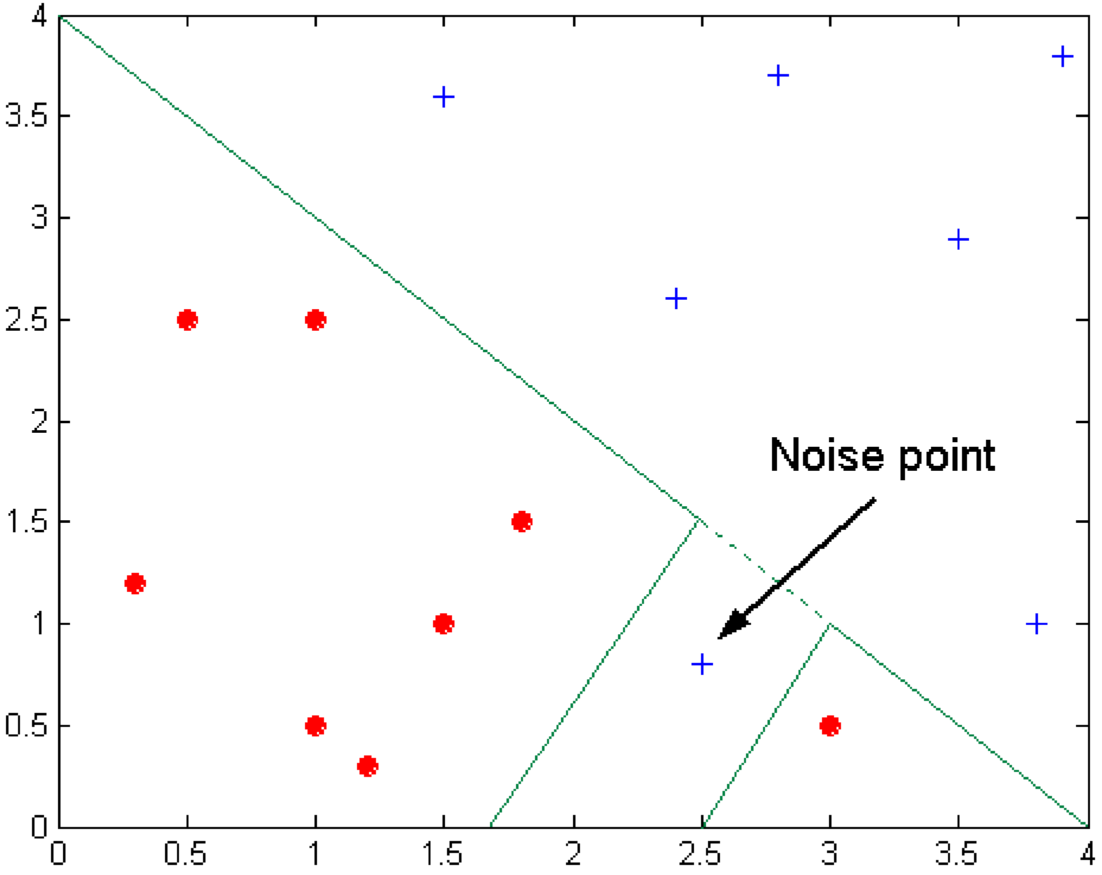
**Overfitting**

Generalization Error

Resubstitution Error

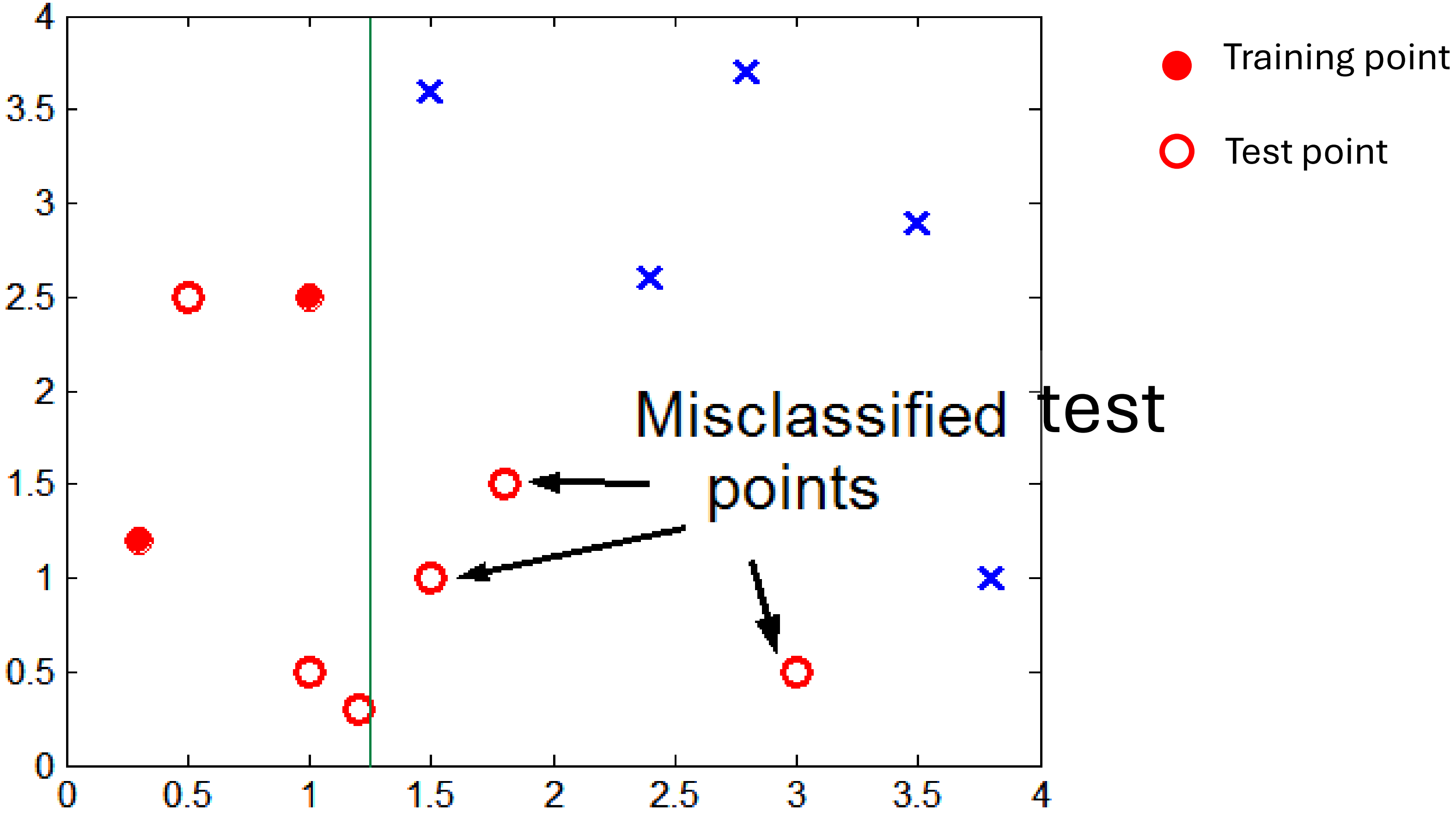
**Underfitting**: when model is too simple, both training and test errors are large.
**Overfitting**: when model is too complicated and starts memorizing the training data.
Generalization error goes up again.

# Overfitting due to Noise



**Decision boundary is distorted to accommodate a noise point**

# Overfitting due to Insufficient Examples



Misclassified test points

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

# Training Error vs. Generalization Error

- Overfitting results in decision trees that **are more complex than necessary.**

- Training error does not provide a good estimate of how well the tree will perform on previously unseen records (e.g., test data).

- We need to estimate the **Generalization Error.**

# Estimating the Generalization Error

- **Resubstitution error $e$**: error on training set
- **Generalization error $e'$:** error on testing set

Methods for estimating generalization errors:

1. **Optimistic approach**: assume $e' = e$
2. **Pessimistic approach:**
   - Estimate as $e' = e + N \times 0.5$ ($N$: number of leaf nodes)
   - For a tree with 30 leaf nodes and 10 errors on training out of 1000 training instances:

     Training error $e = 10/1000 = 1\%$

     Estimated generalization error $e' = (10 + 30\ x\ 0.5)/1000 = 2.5\%$

Penalty for model complexity! 0.5 per leave node is often used for binary splits.

3. **Validation approach:**
   - uses a validation (test) data set (or cross-validation) to estimate the generalization error.

# Occam’s Razor - The Principle of Parsimony

## "Simpler is better"

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model.
- Reason: Complex models have a greater chance of overfitting. I.e., it fitted accidentally errors in the training data.

**Therefore, one should include model complexity when evaluating a model.**

# How to Address Overfitting in Decision Trees

**Pre-Pruning** (Early Stopping Rule): Stop the algorithm before the tree is fully-grown.

- Full tree
  - Stop if all instances belong to the **same class.**
  - Stop if all the **attribute values are the same.**

- Early stopping
  - Stop if **number of instances** is less than some user-specified threshold (estimates become bad for small sets of instances).
  - Stop if class distribution of instances are **independent** of the available features (e.g., using a $\chi^2$ test).
  - Stop if expanding the current node **does not improve impurity** measures more than a user-specified threshold (e.g., Gini or information gain).

# How to Address Overfitting in Decision Trees

**Post-pruning**

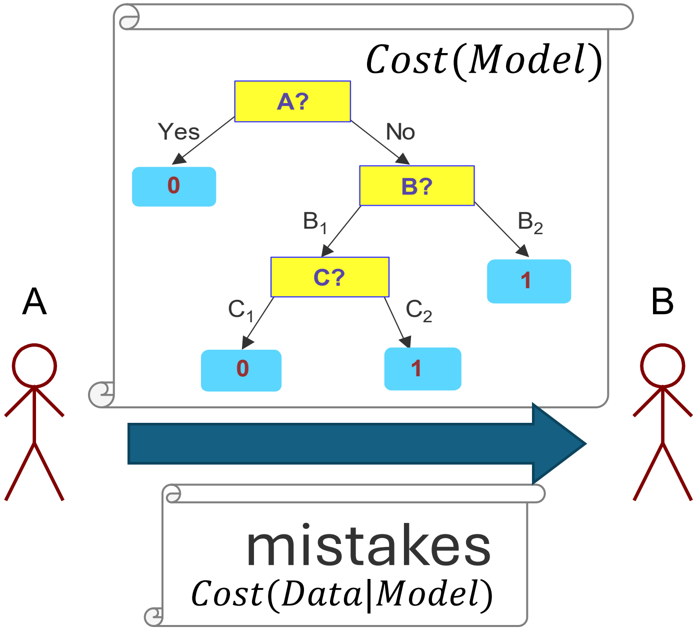
1. Grow complete decision tree.
2. Try to prune sub-trees of the decision tree in a bottom-up fashion.

- If generalization error improves after pruning a sub-tree, replace the sub-tree by a leaf node with the majority class of the training instances as the predicted label.

- You can use MDL instead of error for post-pruning.

# Refresher: Minimum Description Length (MDL)

| X | y |
|---|---|
| $X_1$ | 1 |
| $X_2$ | 0 |
| $X_3$ | 0 |
| $X_4$ | 1 |
| … | … |
| $X_n$ | 1 |



| X | y |
|---|---|
| $X_1$ | ? |
| $X_2$ | ? |
| $X_3$ | ? |
| $X_4$ | ? |
| … | … |
| $X_n$ | ? |

Penalty for model complexity!

- $Cost(Model, Data) = \ Cost(Data|Model) + \ Cost(Model) \rightarrow min$
  - —Cost is the number of bits needed for encoding.
- $Cost(Model)$ encodes each node (splitting condition and children).
- $Cost(Data|Model)$ encodes information to correct misclassification errors.

This is equivalent to the pessimistic generalization error

# Example: Post-Pruning

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

**Before split:**

Training Error = 10/30

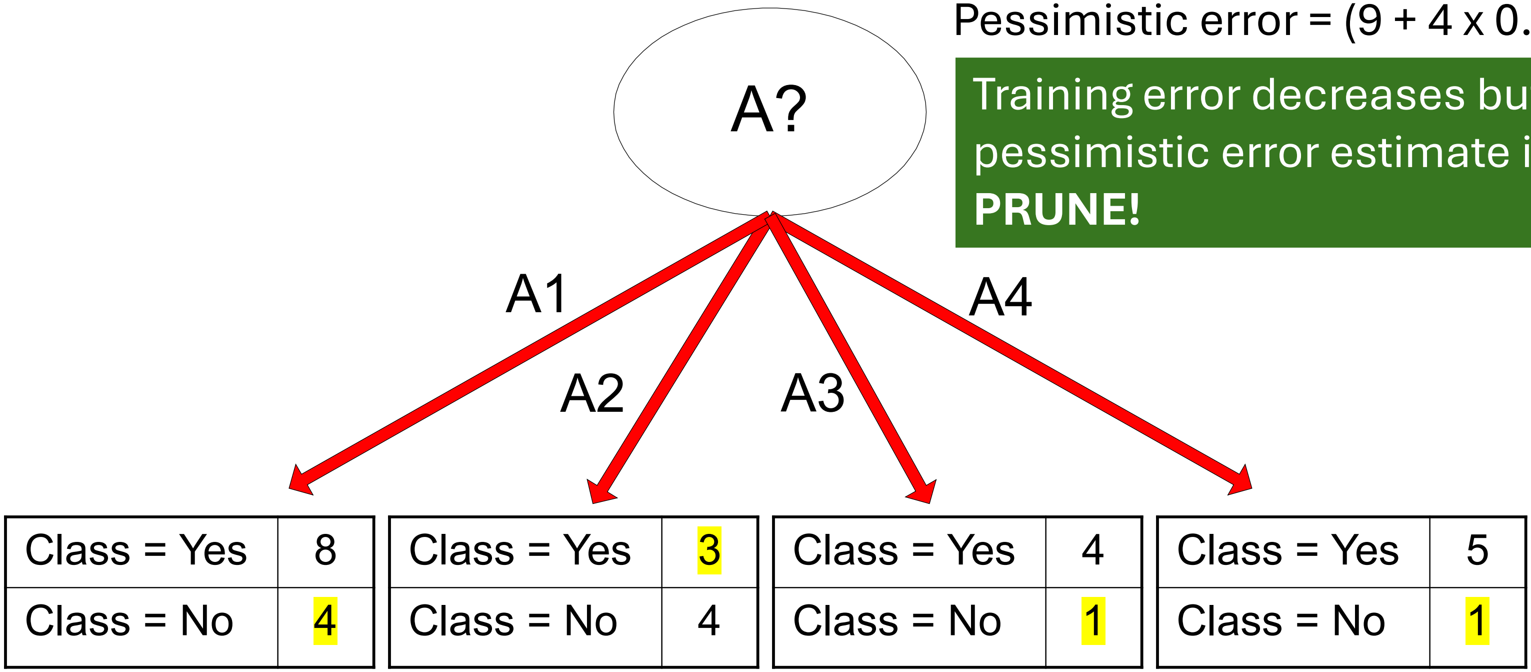
Pessimistic error = (10 + 1 x 0.5)/30 = 10.5/30

**After split:**

Training Error = 9/30

Pessimistic error = (9 + 4 x 0.5)/30 = 11/30

Training error decreases but pessimistic error estimate increases! **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 |

Error = 9

# Other issues:
# Data Fragmentation and Search Strategy

## Data Fragmentation

- Number of instances gets smaller as you traverse down the tree and can become too small to make a statistically significant decision (splitting or determining the class in a leaf node)

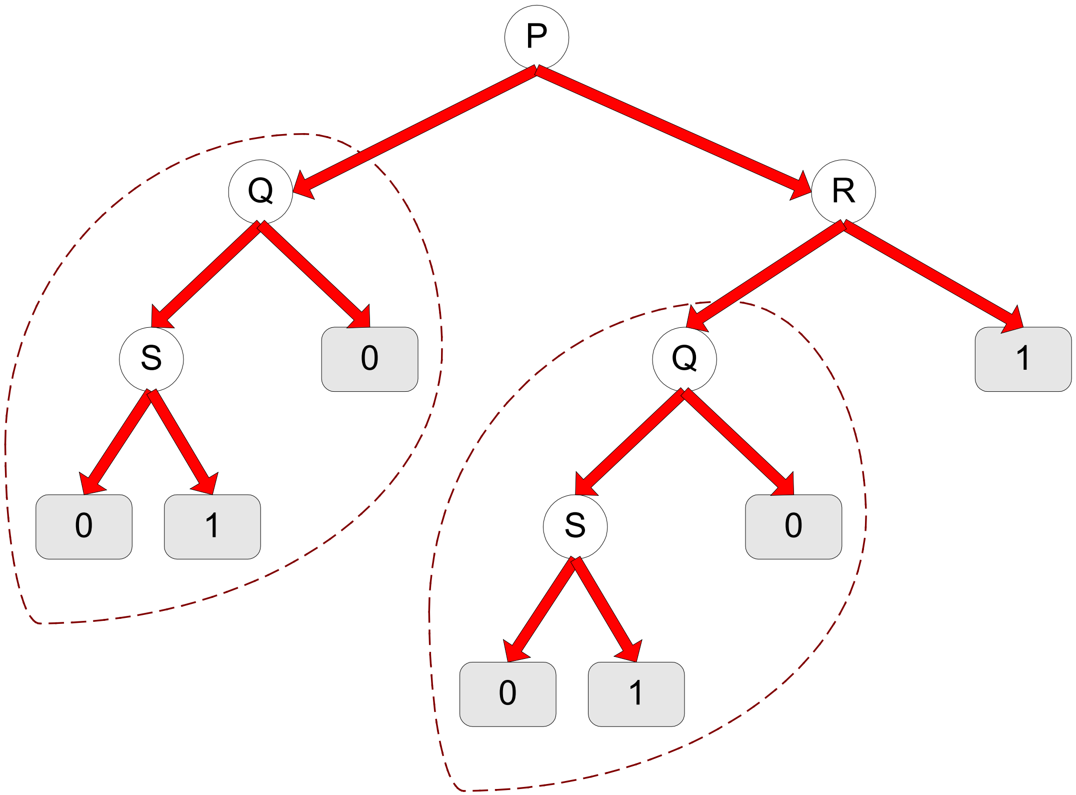
→ Many algorithms **stop when a node has not enough instances**.

## Search Strategy

- Finding an optimal decision tree is NP-hard

→ Most algorithm use a **greedy, top-down, recursive partitioning strategy** to induce a reasonable solution.

# Other issues: Tree Replication



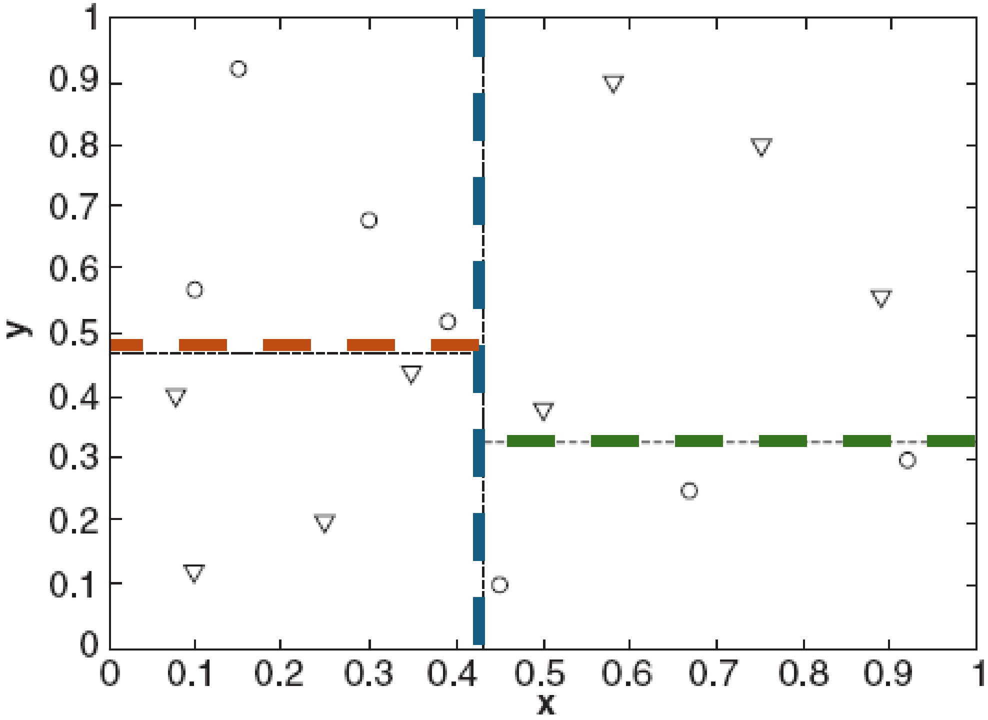
- Same subtree appears in multiple branches
- Makes the model more complicated and harder to interpret

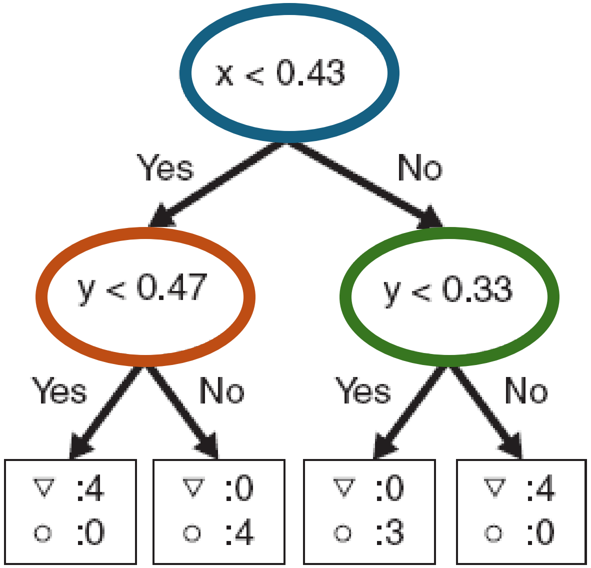
# Expressiveness of Decision Trees

- Decision tree can learn discrete-valued functions to separate classes.
- This function represents the **decision boundary.**

- Issues
  - Not expressive enough for modeling continuous variables directly. Discretization is performed for the splits.
  - Do not generalize well to certain types of Boolean functions like the parity function (Class = 1 if there is an even number of Boolean attributes with truth value = True and 0 otherwise). These functions lead to excessive tree replication.

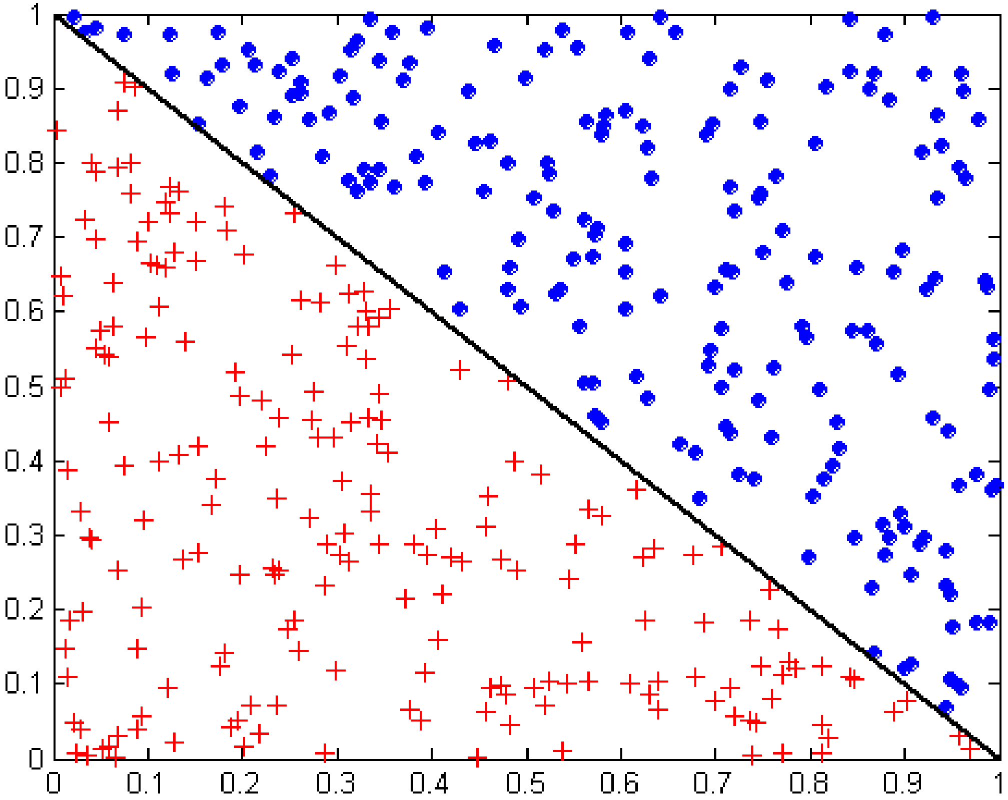
# Decision Boundary

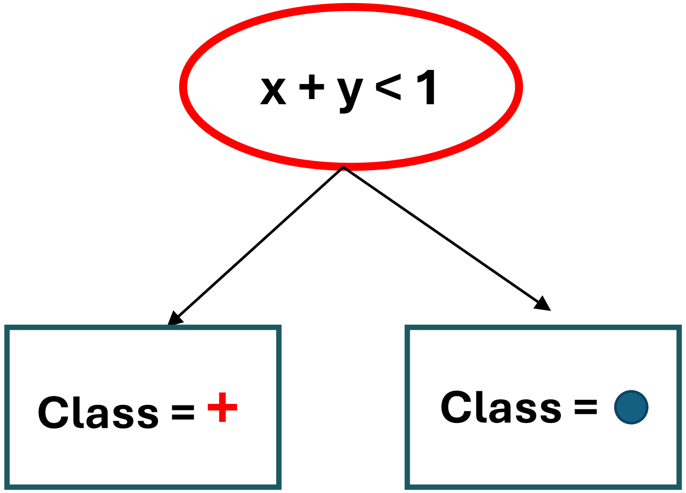




- The border line between two neighboring regions of different classes is known as the decision boundary.
- The decision boundary is parallel to the axes because each test condition represents a threshold on a single attribute.

# Oblique Decision Trees





$x + y < 1$

Class = +

Class = ●

- The test condition may involve multiple attributes.
- More expressive representation.
- Finding the optimal test condition is computationally expensive! -> Not used in practice.

# Topics

- Introduction
- Decision Trees
  - —Overview
  - —Tree Induction
- Overfitting and other Practical Issues
- **Model Selection and Evaluation**
  - **—Metrics for Performance Evaluation**
  - —Methods to Obtain Reliable Estimates
  - —Model Comparison (Relative Performance)
- Feature Selection

# Metrics for Performance Evaluation: Confusion Matrix

- Focuses on the predictive capability of a model (not speed, scalability, etc.)
- For simplicity, we will present a binary classification problem here, but most measures generalize to multi-class problems.

## Confusion Matrix

| | | PREDICTED CLASS | |
|---|---|---|---|
| ACTUAL CLASS | | Class=Yes | Class=No |
| | Class=Yes | **a**<br>**(TP)** | *b*<br>*(FN)* |
| | Class=No | *c*<br>*(FP)* | **d**<br>**(TN)** |

**a: TP (true positive)**

**b: FN (false negative)**

**c: FP (false positive)**

**d: TN (true negative)**

# Metrics for Performance Evaluation: Statistical Test

From Statistics: Null Hypotheses $H0$ is that the actual class is Yes.

| | | PREDICTED CLASS | |
|---|---|---|---|
| ACTUAL CLASS | | Class=Yes | Class=No |
| | Class=Yes | | **Type I error** (FN) |
| | Class=No | **Type II error** (FP) | |

$\longleftarrow H0$

Type I error: $P(\text{NO} \mid H0 \text{ is true})$ → Significance level $\alpha$

Type II error: $P(\text{Yes} \mid H0 \text{ is false})$ → Power $1 - \beta$

# Metrics for Performance Evaluation: Accuracy

Most widely-used metric:
How many do we predict correct (in percent)?

| | PREDICTED CLASS | | |
|---|---|---|---|
| ACTUAL CLASS | | Class=Yes | Class=No |
| | Class=Yes | a<br>(TP) | b<br>(FN) |
| | Class=No | c<br>(FP) | d<br>(TN) |

$$Accuracy = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{N}$$

# Limitation of Accuracy

Consider a 2-class problem with a total population of

- Number of Class 0 examples = 9990
- Number of Class 1 examples = 10

A model that predicts everything to be class 0, has an accuracy of

$$9990/10000 = 99.9\,\%$$

- Accuracy is misleading because the model does not detect any class 1 example!

→ This is called the **class imbalance problem**

# Cost Matrix

Different types of error can have different cost!

| | | PREDICTED CLASS | |
|---|---|---|---|
| | C(i\|j) | **Class=Yes** | **Class=No** |
| ACTUAL CLASS | **Class=Yes** | C(Yes\|Yes) | C(No\|Yes) |
| | **Class=No** | C(Yes\|No) | C(No\|No) |

$C(i \mid j)$: Cost of misclassifying class $j$ example as class $i$

# Computing Cost of Classification

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

Missing a '+' case is really expensive!

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

Accuracy = 80%

Cost = -1*150+100*40+
1*60+0*250 = 3910

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

Accuracy = 90%

Cost = 4255

# Cost vs Accuracy

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

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

Accuracy is only proportional to cost if

1. C(Yes|No)=C(No|Yes) = q
2. C(Yes|Yes)=C(No|No) = p

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

Accuracy $= (a + d)/N$

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

# Cost-Biased Measures

| | PREDICTED CLASS | | |
|---|---|---|---|
| ACTUAL CLASS | | Class Yes | Class No |
| | Class Yes | a<br>(TP) | b<br>(FN) |
| | Class No | c<br>(FP) | d<br>(TN) |

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

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

$$F - measure\ (F) = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

- 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)

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

# Kappa Statistic

**Idea**: Compare the accuracy of the classifier with a random classifier. The classifier should be better than random!

|  |  | PREDICTED CLASS |  |
|---|---|---|---|
| ACTUAL CLASS |  | Class Yes | Class No |
|  | Class Yes | a<br>(TP) | b<br>(FN) |
|  | Class No | c<br>(FP) | d<br>(TN) |

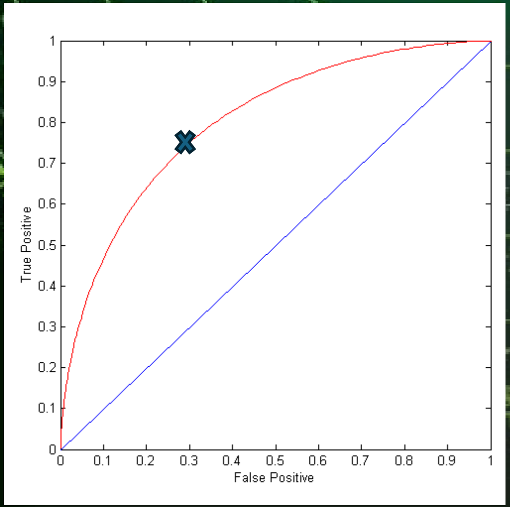
$$\kappa = \frac{\text{total accuracy } - \text{random accuracy}}{1 \ - \text{random accuracy}}$$

$$\text{total accuracy} = \frac{TP + TN}{N}$$

$$\text{random accuracy} = \frac{TP + FP \times TN + FN + FN + TN \times FP + TP}{N^2}$$

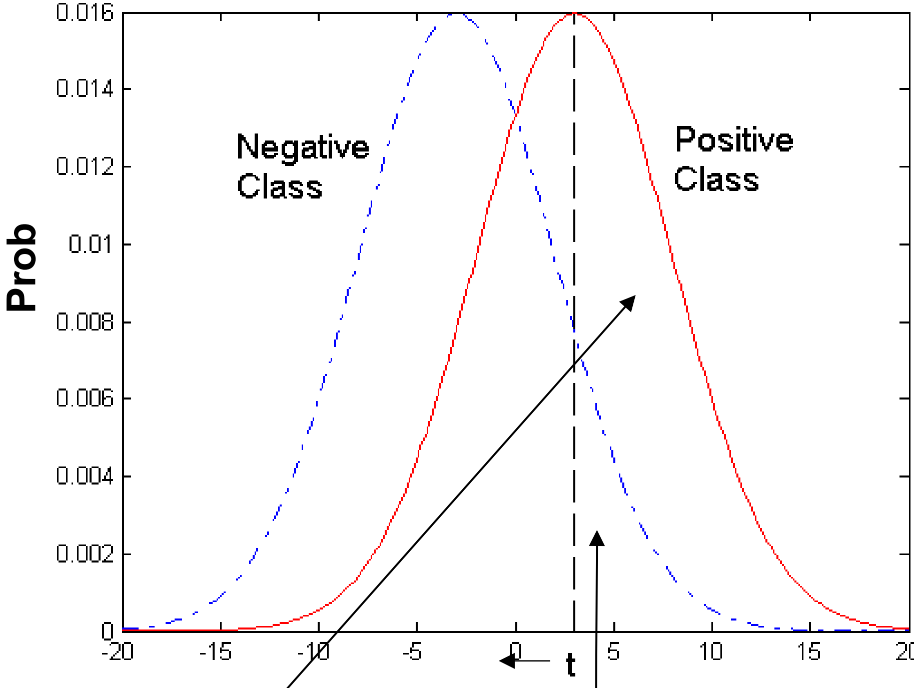
# Receiver Operating Characteristic (ROC)

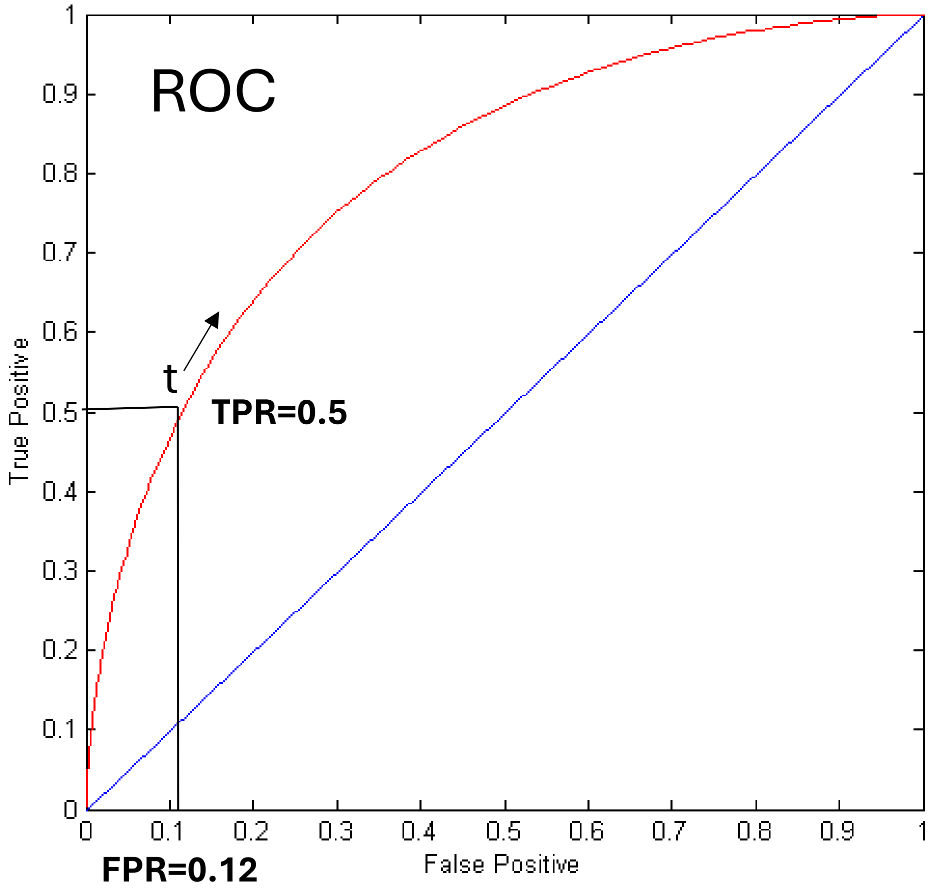
- Developed in 1950s for signal detection theory to analyze noisy signals to characterize the trade-off between positive hits and false alarms.
- Works only for binary classification (two-class problems).
- ROC curve plots TPR (true positive rate) on the y-axis against FPR (false positive rate) on the x-axis.
- Performance of each classifier represented as a point. Changing the threshold of the algorithm, sample distribution or cost matrix changes the location of the point and forms a curve.



# ROC Curve

- Example with 1-dimensional data set containing 2 classes (positive and negative)
- Any points located at x > t is classified as positive





**At threshold t:**

**TPR=0.5,** FNR=0.5, **FPR=0.12,** FNR=0.88

- Move t to get the other points on the ROC curve.

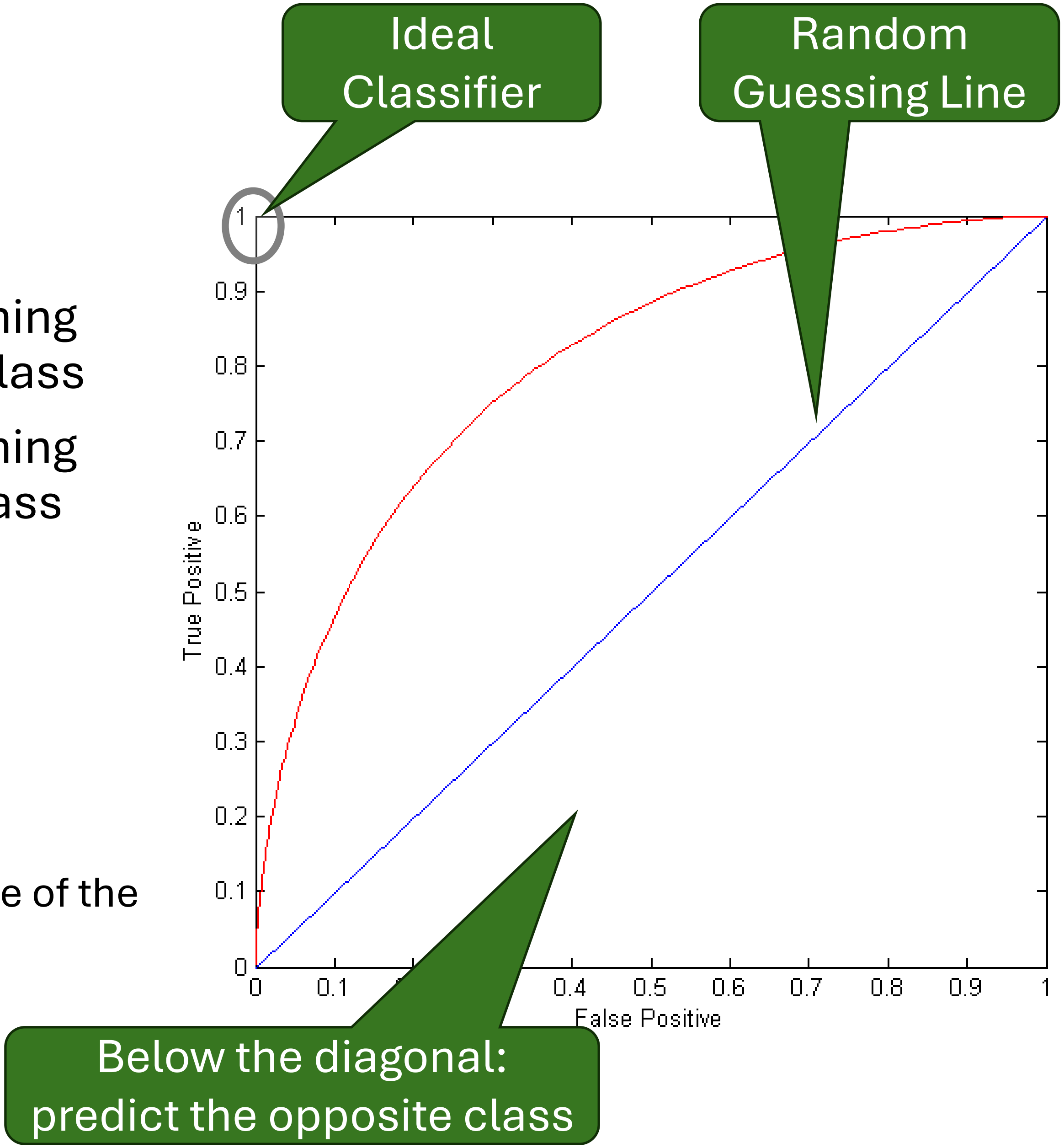
# ROC Curve

(TPR,FPR):

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

Diagonal line:

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

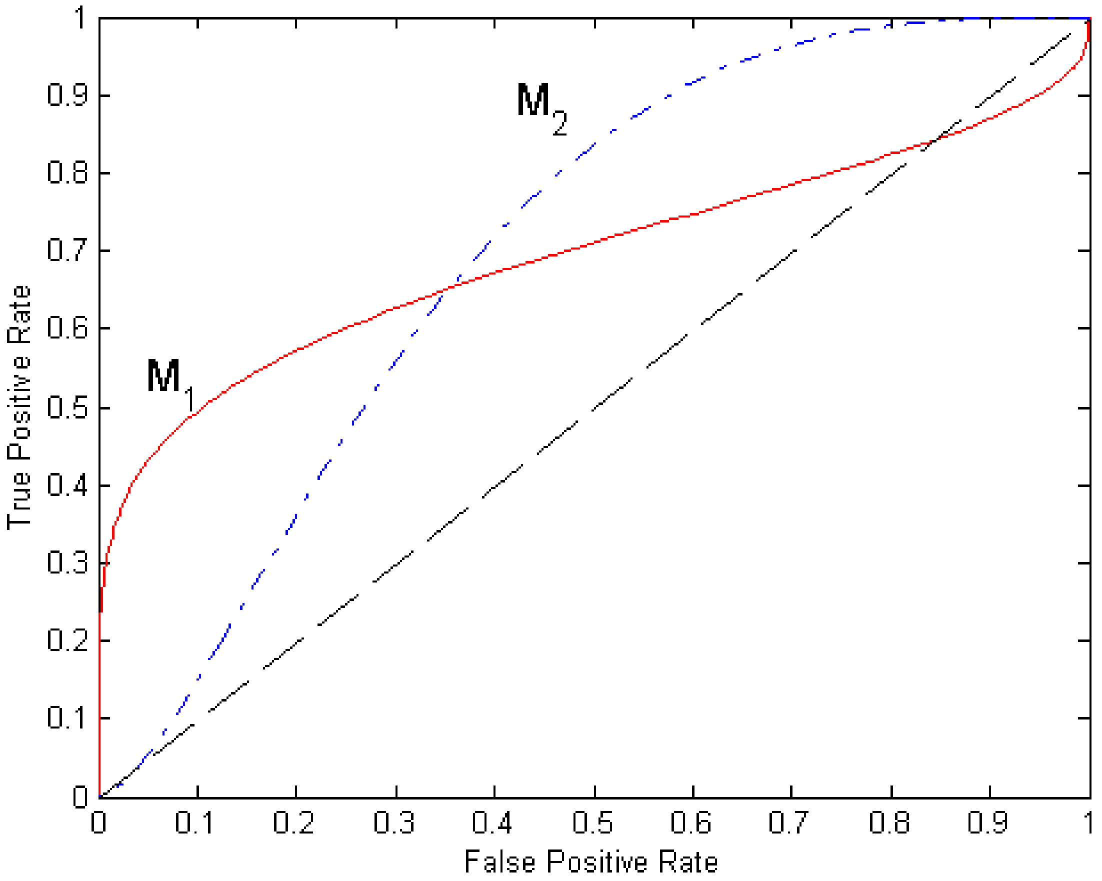


Ideal Classifier

Random Guessing Line

Below the diagonal: predict the opposite class

# Using ROC for Model Comparison



No model consistently outperform the other

- M1 is better for small FPR
- M2 is better for large FPR

**Area Under the ROC curve (AUC)**

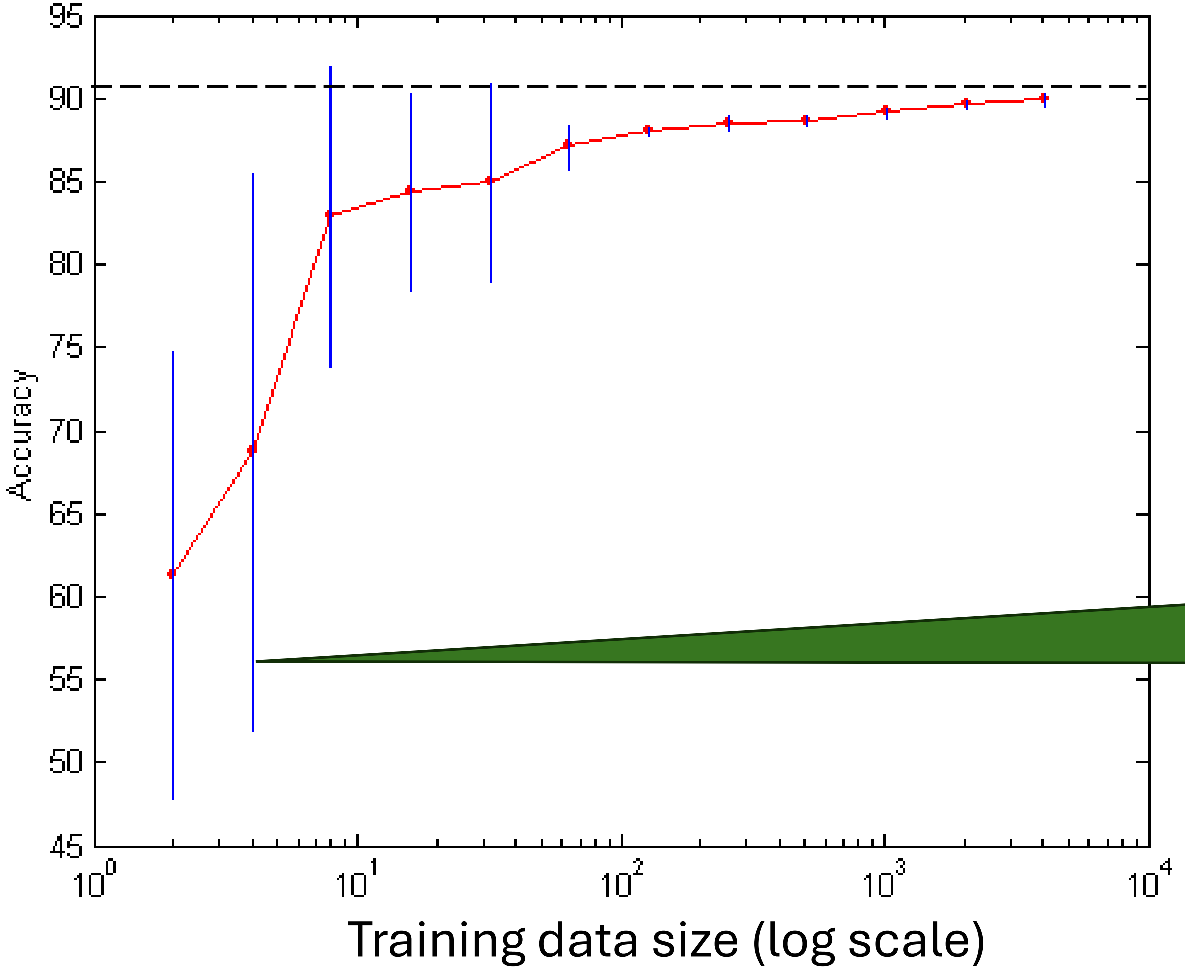
- Ideal:
  - AUC = 1
- Random guess:
  - AUC = 0.5

# Topics

- Introduction
- Decision Trees
  - —Overview
  - —Tree Induction
- Overfitting and other Practical Issues
- Model Selection and Evaluation
  - —Metrics for Performance Evaluation
  - —**Methods to Obtain Reliable Estimates**
  - —Model Comparison (Relative Performance)
- Feature Selection

# Learning Curve

Accuracy and variance between runs depend on the size of the training data.



Learning curve shows how accuracy on unseen examples changes with varying training sample size

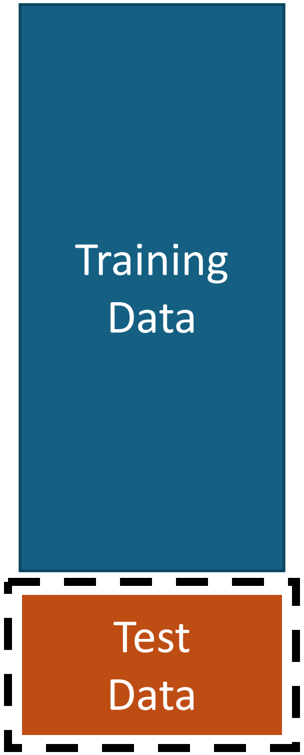
Variation for different runs

# Estimating the Generalization Error Using Test Data

- To estimate generalization error we need to separate the data into a set to train and a set to test.

- **Holdout testing/Random splits**: Split the data randomly into, e.g., 80% training and 20% testing.

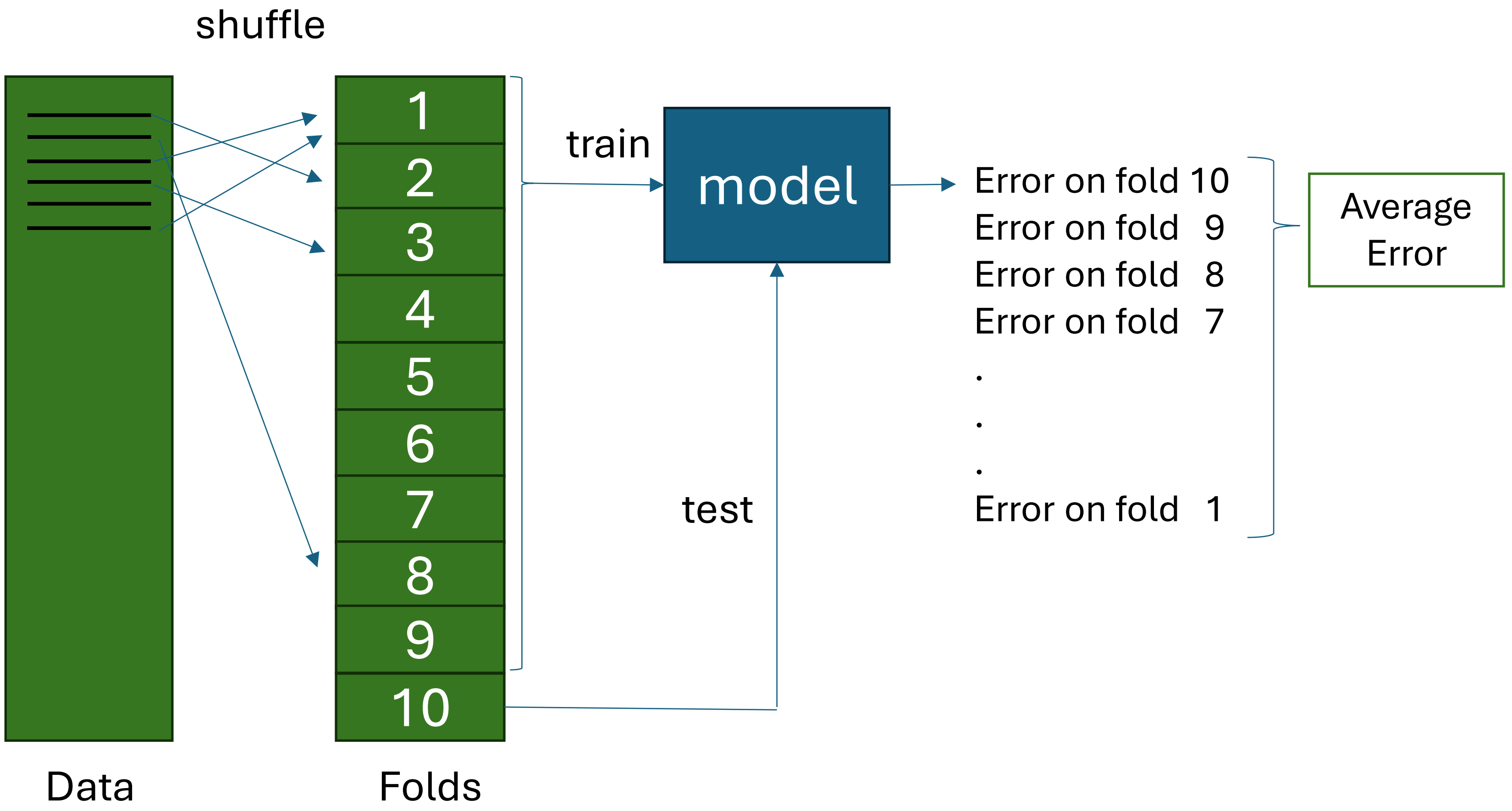
**Very important:** the algorithm can never look at the test set during learning!



# $k$-fold Cross Validation

**k-fold cross validation**: Use data better to estimate the generalization error:

- Split the data randomly into k folds.
- For $k$ rounds hold 1 fold back for testing and use the remaining $k-1$ folds for training.
- Use the average of the error/accuracy as a better estimate.
- Some algorithms/tools do that internally.



# Training and Testing with Hyperparameters

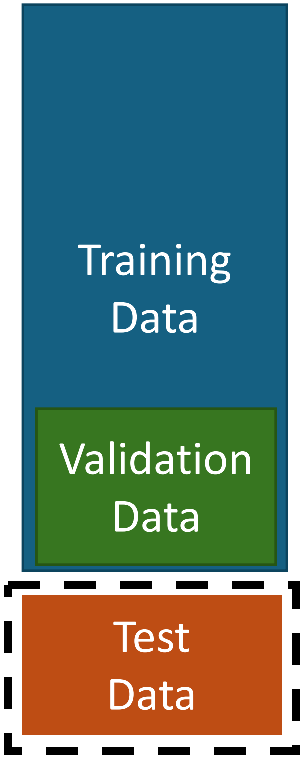
**Hyperparameters**: Many algorithms allow choices for learning. E.g.,

- maximal decision tree depth
- selected features

We do not want to overfit the hyperparameters!!!

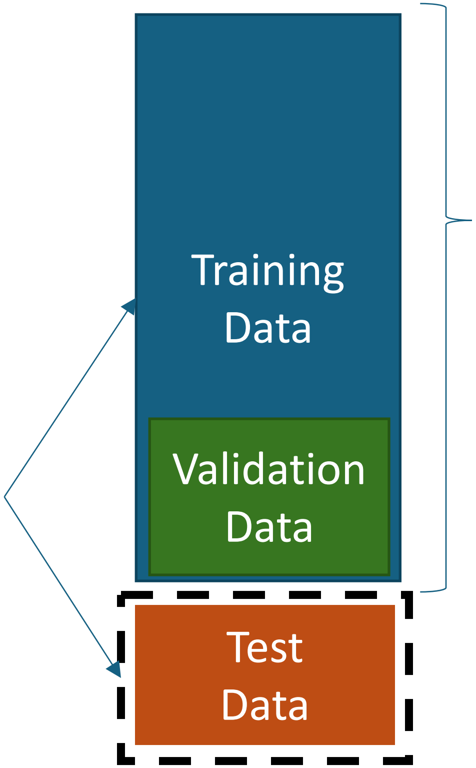
Use a generalization error estimate twice:

1. **Train:** Learn models on the **training data** (without the validation data) using different hyperparameters.
   - A grid of possible hyperparameter combinations
   - greedy search
2. **Model Selection:** Evaluate the models using the **validation data** and choose the hyperparameters with the best accuracy. Rebuild the model using all the training data.
3. **Test** the final model using the **test data**.



# Typical Data Use with Model Selection

**Test data**: Split the data randomly into 20% testing and 80% training + validation.



**Model Selection**: Use training & validation data  with 10-fold cross validation for choosing between models and hyper parameter tuning.

# Confidence Interval for Accuracy

- Each prediction can be regarded as a **Bernoulli trial**: A Bernoulli trial (a biased coin toss) has 2 possible outcomes:
  heads (correct) or tails (wrong)

  We use $p$ for the true chance that prediction is correct (= true accuracy).

- Predictions for a test set of size $N$ are a collection of $N$ Bernoulli trials. The number of correct predictions $x$ has a **Binomial distribution**:
  $$X \sim Binomial(N, p)$$

- Example: Toss a fair coin 50 times, how many heads would turn up? Expected number of heads $E[X] = Np = 50 \times 0.5 = 25$

- **Application for Accuracy**: If we observe $x$ correct predictions then the observed accuracy is
  $$\hat{p} = x/N$$

  Can we give bounds for the true accuracy of model $p$?

# Confidence Interval for Accuracy

For large test sets ($N > 30$) we can approximate the Binomial distribution

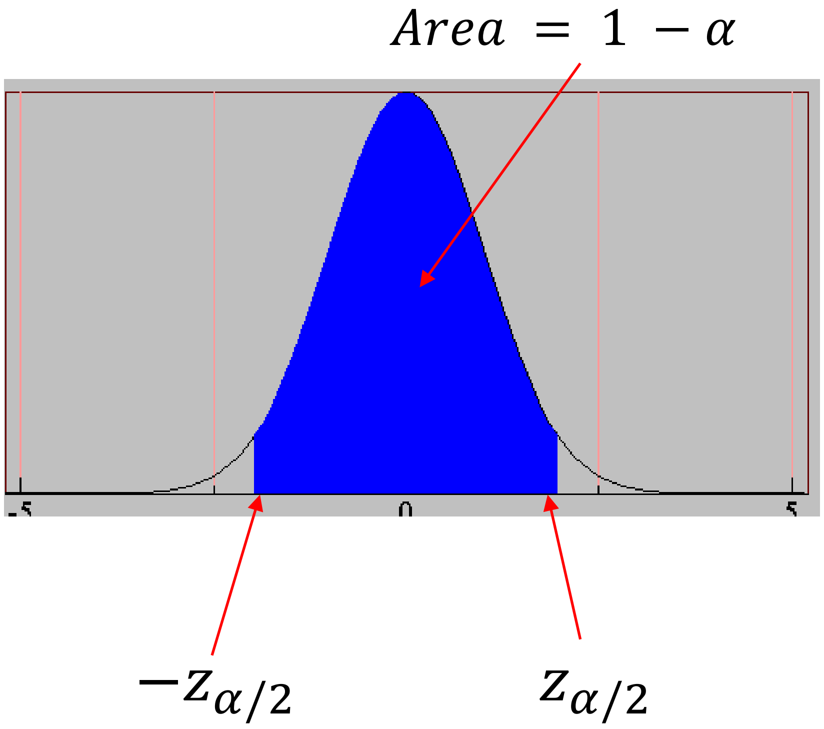
$$X \sim Binomial(N, p)$$

by a Normal distribution:

$$X \sim Normal(Np, Np(1-p))$$

Confidence Interval for $p = X/N$ (Wald Method):

$$\hat{p} \pm z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{N}}$$



# Confidence Interval for Accuracy

Consider a model that produces an accuracy of 80% when evaluated on 100 test instances:

- $N = 100,\ acc = 0.8$
- Let $1 - \alpha = 0.95$ (95% confidence)
- From probability table, $z_{\alpha/2} = 1.96$

| $1 - \alpha/2$ | $z_{\alpha/2}$ |
|---|---|
| 0.99 | 2.58 |
| 0.98 | 2.33 |
| 0.95 | 1.96 |
| 0.90 | 1.65 |

$$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{N}}$$

| N | 50 | 100 | 500 | 1000 | 5000 |
|---|---|---|---|---|---|
| p(lower) | 0.689 | 0.722 | 0.765 | 0.775 | 0.789 |
| p(upper) | 0.911 | 0.878 | 0.835 | 0.825 | 0.811 |

Table or
R $qnorm(1 - \alpha/2)$

Data mining tools typically calculate this for us.

# Topics

- Introduction
- Decision Trees
  - —Overview
  - —Tree Induction
- Overfitting and other Practical Issues
- Model Selection and Evaluation
  - —Metrics for Performance Evaluation
  - —Methods to Obtain Reliable Estimates
  - —**Model Comparison (Relative Performance)**
- Feature Selection

# Comparing Performance between 2 Models

Given two models, say $M_1$ and $M2$, which is better?

For large test sets ($N > 30$) we have approximately:
$acc_1 \sim Normal(p_1, Np_1(1-p_1))$
$acc_2 \sim Normal(p_2, Np_2(1-p_2))$

Perform a paired t-test with:

H0: There is no difference in accuracy between the models.

H1: There is a difference.

Comparing more than two models: You need to **correct for multiple comparisons!** For example, using Bonferroni correction or False Discovery Rate (FDR).

# Topics

- Introduction
- Decision Trees
  - Overview
  - Tree Induction
- Overfitting and other Practical Issues
- Model Selection and Evaluation
  - Metrics for Performance Evaluation
  - Methods to Obtain Reliable Estimates
  - Model Comparison (Relative Performance)
- **Feature Selection**

# Feature Selection

What features should be used in the model?

### Univariate feature importance score

- Measures how related each feature is to the class variable.
- E.g., chi-squared statistic, information gain.

### Feature subset selection

- Tries to find the best set of features.
- Often uses a black box approach where different subsets are evaluated using a greedy search strategy.
- E.g.: Stepwise backward selection tries to remove one feature at a time.

# Conclusion

- Classification is **supervised learning** with the goal to find a model that generalizes well.
- **Generalization error** can be estimated using test sets/cross-validation and should be used for model selection.
- Model evaluation and comparison needs to take **model complexity** into account.