



SINGAPORE UNIVERSITY OF  
TECHNOLOGY AND DESIGN

Established in collaboration with MIT

# Classification

---

*PROF. D. HERREMANS*

50.038 Computational data science

# Data science tasks

---

| Task                   | Supervised Methods | Unsupervised Methods |
|------------------------|--------------------|----------------------|
| *Classification        | ✓                  |                      |
| *Regression            | ✓                  |                      |
| Causal Modeling        | ✓                  |                      |
| Similarity Matching    | ✓                  | ✓                    |
| Link Prediction        | ✓                  | ✓                    |
| Data Reduction         | ✓                  | ✓                    |
| *Clustering            |                    | ✓                    |
| Co-occurrence Grouping |                    | ✓                    |
| Profiling              |                    | ✓                    |



# What is classification?

---

***“The act or process of dividing things into groups according to their type”***

Cambridge Dictionary

# Examples of Classification

- Email, spam or scam?

Wed 4/7/2018 4:40 PM

SUTD IT Service Desk <ITServiceDesk@sutd.ed-u.sg>  
IT Service Desk - Account Verification

To: Clement Low

**SUTD**  
SINGAPORE UNIVERSITY OF  
TECHNOLOGY AND DESIGN

Dear Staff,

It was brought to our attention that there was unauthorized access to your account recently.

**Sign-in details**  
Country/region: Netherlands  
IP address: 188.166.210.8  
Date: 7/4/2018 11:07 AM (SGT)  
Platform: Windows 10  
Browser: Chrome

You are required to reset your password now to prevent further compromise.

Please visit the <http://www.sutd.ed-u.sg?nid=ruo84rv> to reset your account.  
Click or tap to follow link.

[Reset Password](#)

Best regards,  
SUTD IT Service Desk

The email domain may look similar, but it is not from SUTD.

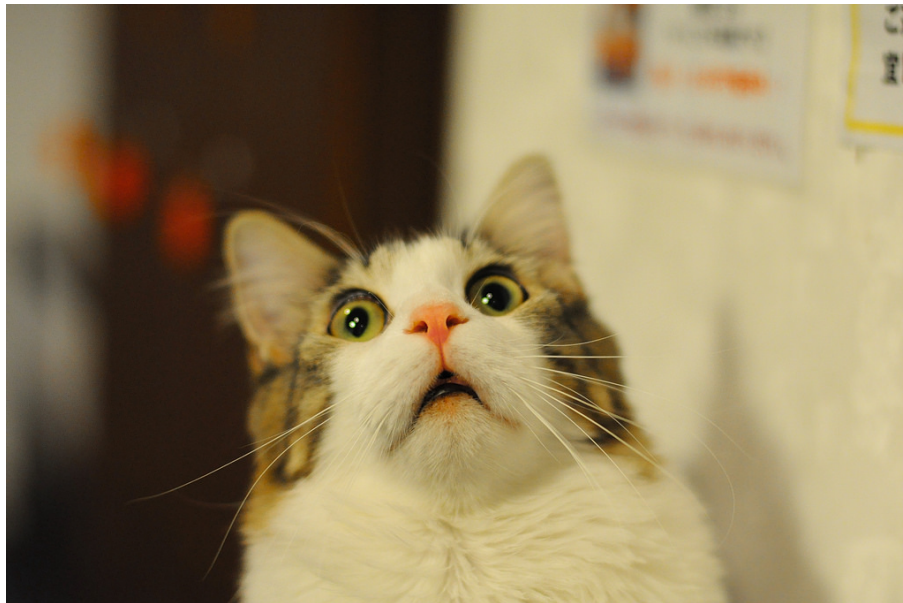
Non-personalised email may indicate they do not know you and is part of a mass emailing.

Be cautious of suspicious links. Hover over embedded links to view where the link will take you.  
Reset password by visiting SUTD website and not click on the link.

# Examples of Classification

---

- Cat or dog?



<https://www.flickr.com/photos/dat-pics/4553277701>



<https://www.flickr.com/photos/mydoglikes/27249865650>

# Examples of Classification

- Traffic jam or not?



# Examples of Classification

---

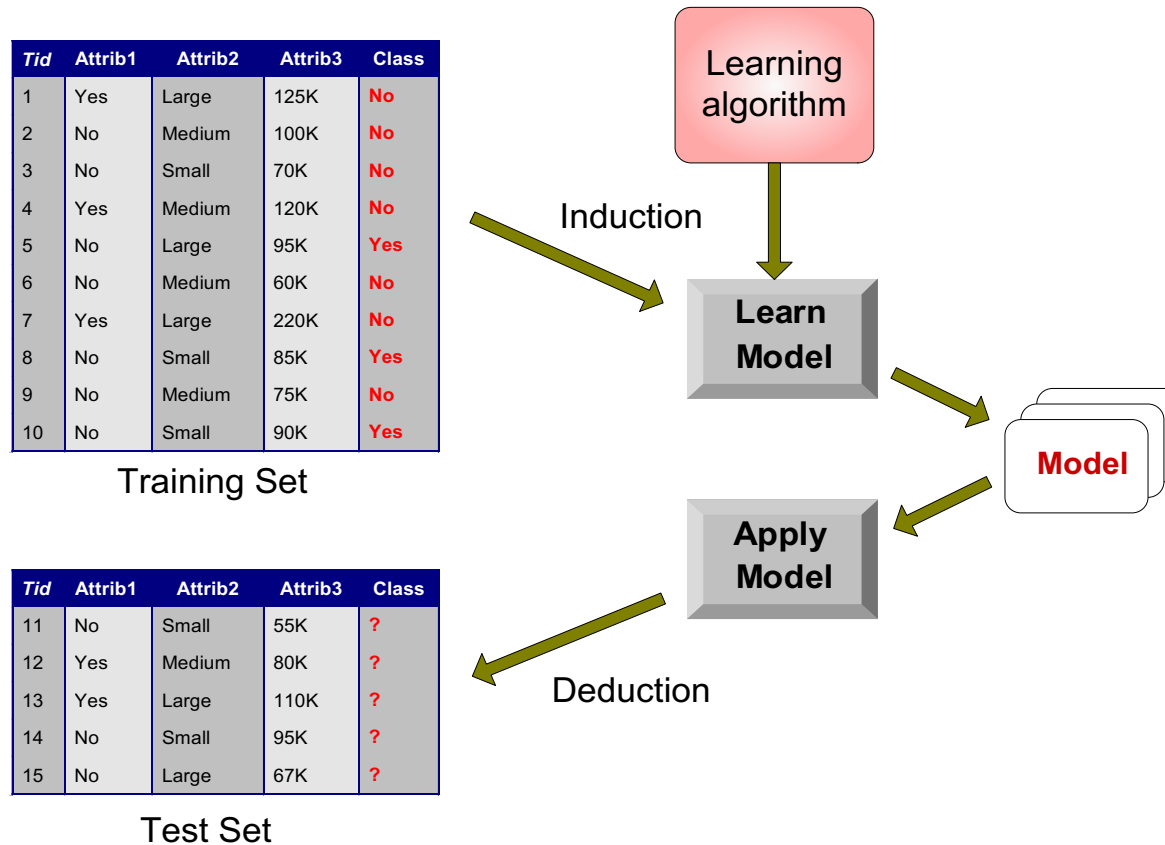
- Is a movie review positive or negative?
- What species of an animal, plant, etc?
- Where was a message/tweet/post sent from?
- What genre a song belongs to?
- And many more...

# Types of dataset / sources

---

- Text
- Audio
- Images
- Videos
- ...

# General Approach to Classification Tasks



Source: Introduction to Data Mining, 2<sup>nd</sup> Edition



# Classification: Definition

---

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



# Classification Techniques

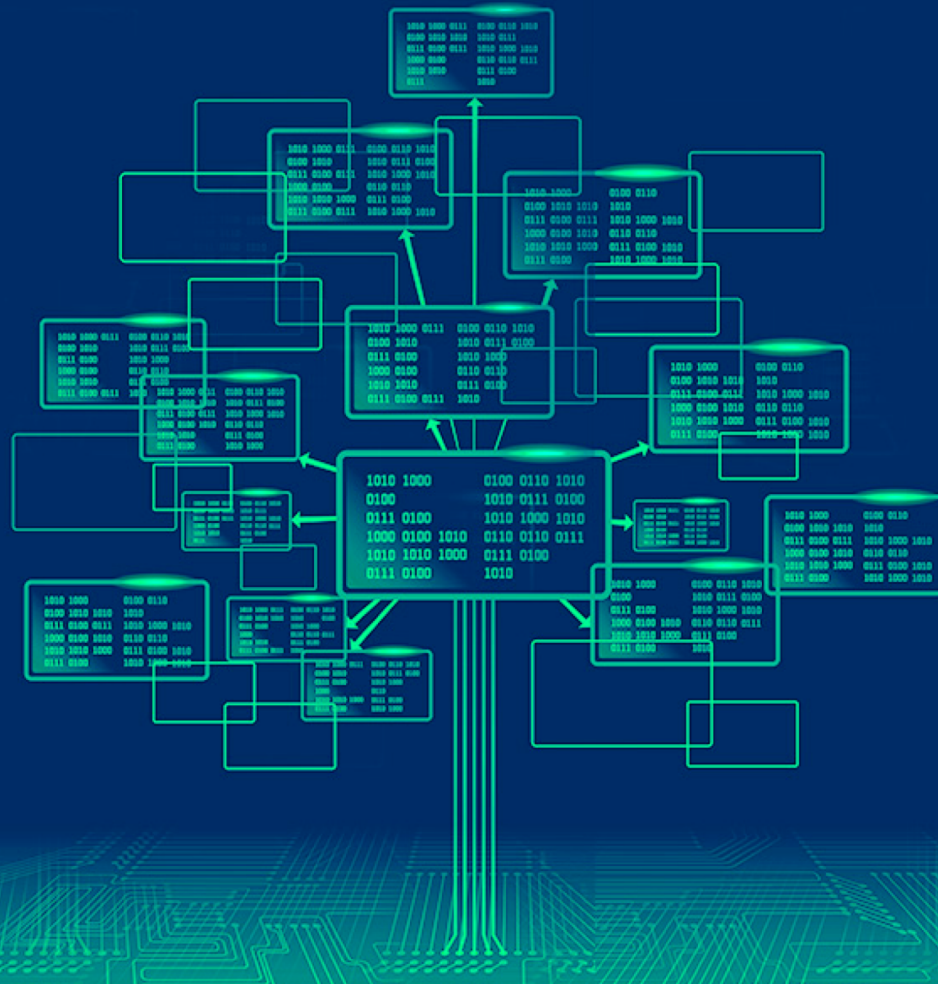
---

- Base Classifiers
  - Decision Tree based Methods
  - Rule-based Methods
  - Nearest-neighbor
  - Logistic regression
  - Naïve Bayes and Bayesian Belief Networks
  - Support Vector Machines
  - Neural Networks
  - Deep Learning
- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

# Which classification method should we use?

---

- The one that gives the **best predictions...**
  - On the training data
  - On the (unseen) test data
  - On the (held-out) validation data
- The one that is **computationally** efficient...
  - during training
  - during classification
- The most **interpretable** one...
  - In terms of its parameters
  - as a model
- The one that is **easiest to implement...**
  - For learning
  - For classification



# Decision trees

# Decision Tree

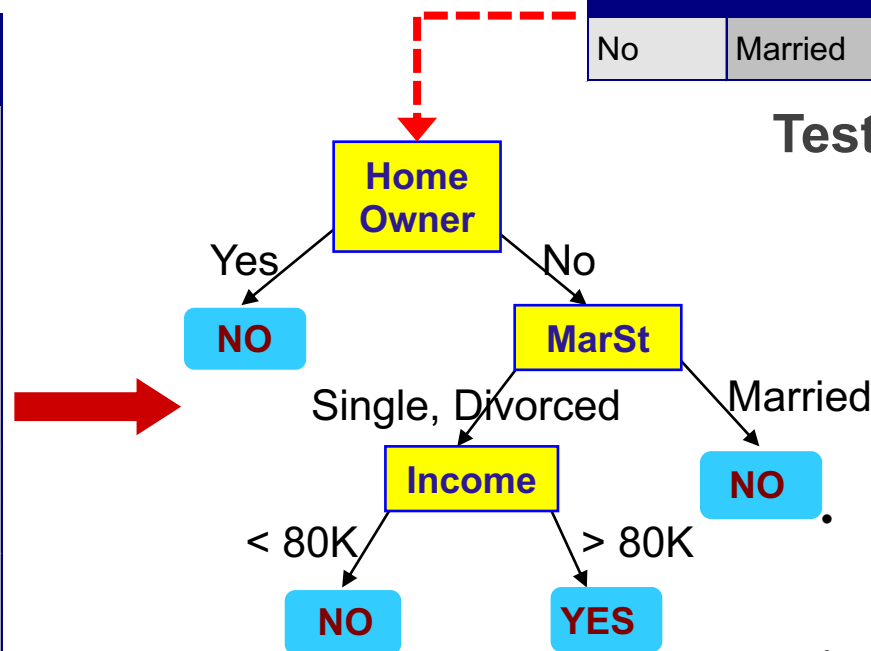
- Decision point at each feature, until reaching the leaf node (category)

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

Training Data

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

Test Data



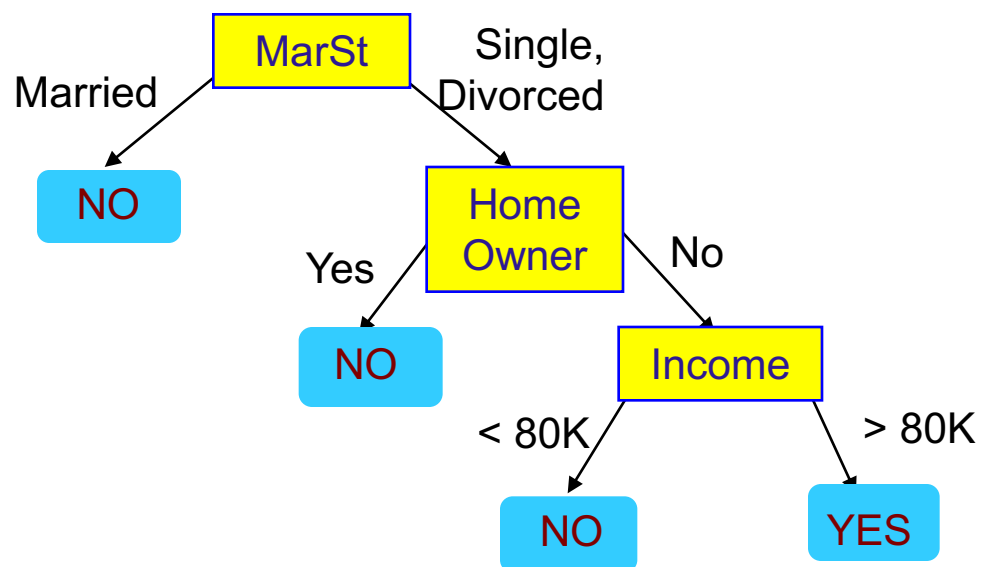
Model: Decision Tree

- Internal node: test of the value of a given attribute
- Branch: value of an attribute
- Leaf: predicted class

# Another Example of Decision Tree

categorical categorical continuous class

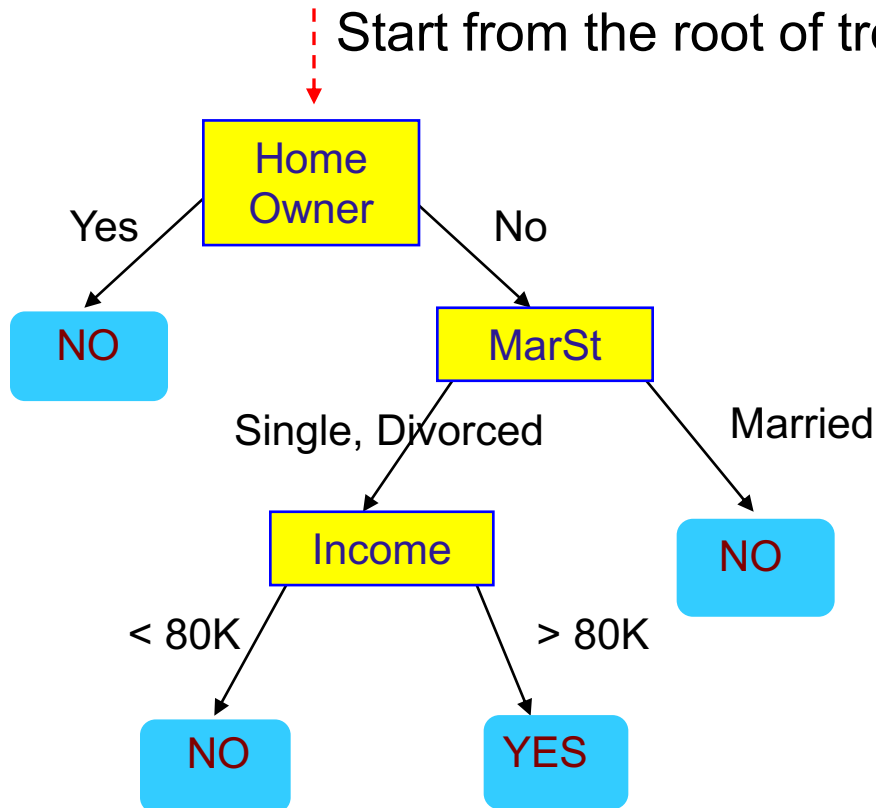
| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No                 |
| 2  | No         | Married        | 100K          | No                 |
| 3 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



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

# Apply Model to Test Data

Start from the root of tree.



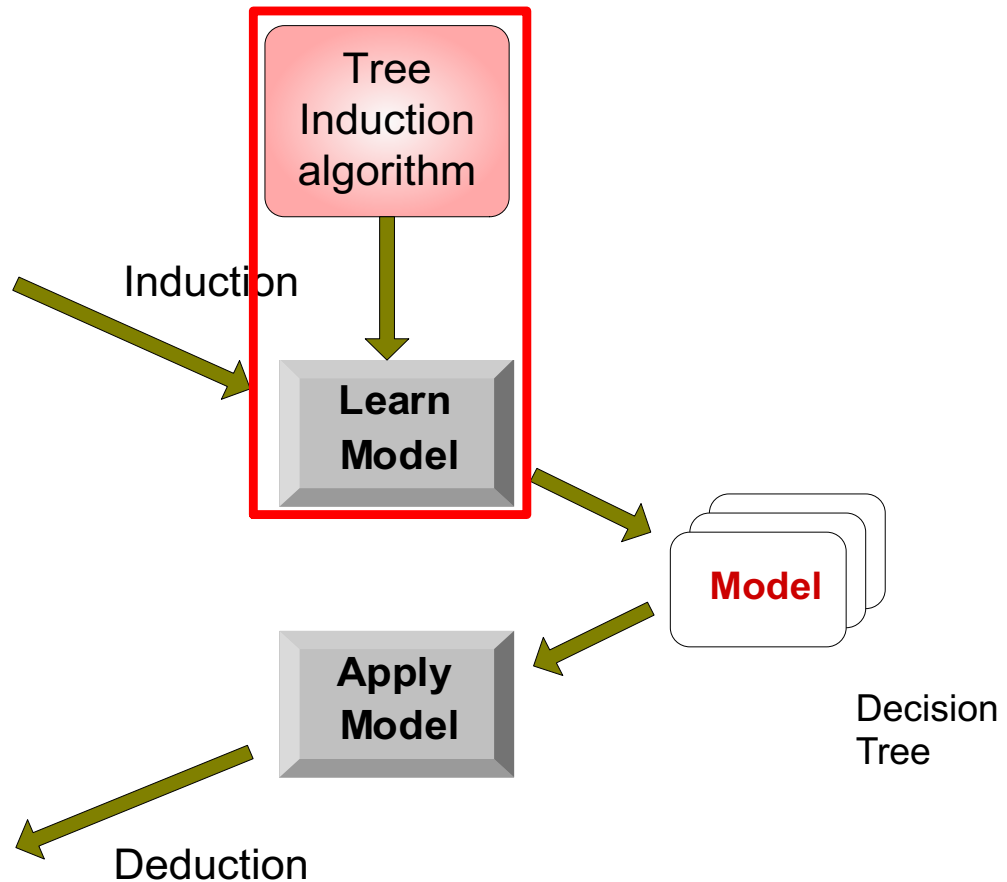
| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |

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

Training Set

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

Test Set



# Decision Tree Induction

---

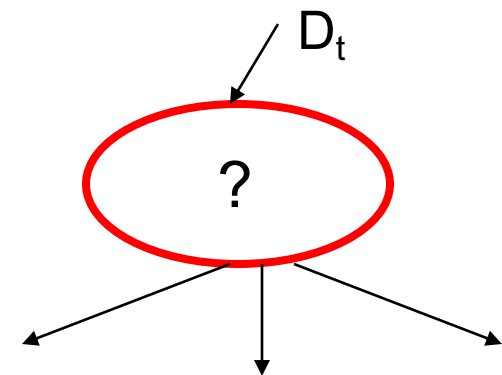
- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART (Classification And Regression Trees)
  - ID3 (Iterative Dichotomiser 3)
  - C4.5 (successor of ID3)
  - SLIQ (Supervised Learning In Quest,)
  - SPRINT (Scalable PaRallelizable INduction of decision Trees)



# 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 only belong the class  $y_t$ , then  $t$  is a **leaf** node labeled as  $y_t$
  - If  $D_t$  contains records that belong to more than one class, use an **attribute** test to split the data into smaller subsets.
  - Recursively apply the procedure to each subset.

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



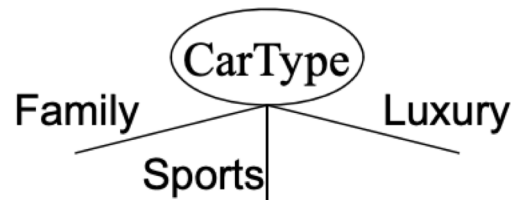
# Design Issues of Decision Tree Induction

---

- **How** should training records be split?
  - *Which attribute* to use in a split node split?
    - How to determine the best split
  - How to specify the attribute *test condition*?
    - E.g.  $x < 1$ ? Or  $x+y < 1$  ?
  - Shall we use 2-way or *multi-way split*?
- How should the splitting procedure **stop**?
  - Stop splitting if all the records belong to the same class or have identical attribute values
  - Early termination

# Splitting of nominal attributes

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



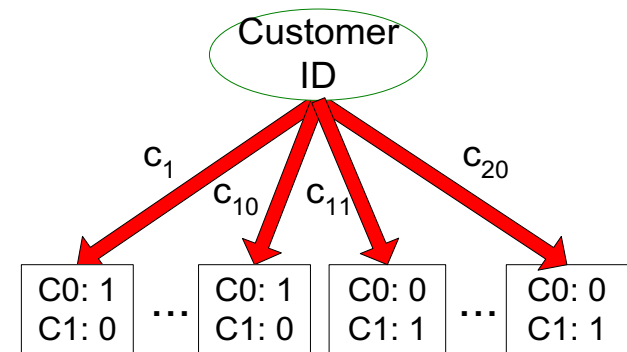
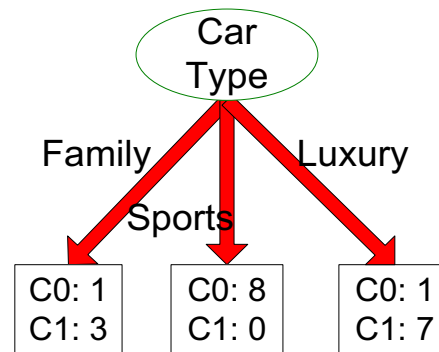
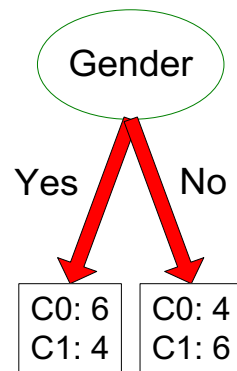
- Binary split: divide values into two subsets. Need to find optimal partitioning



# How to determine the Best Split

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

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



Which test condition is the best?

# How to determine the Best Split

---

- Greedy approach:
  - Nodes with **purier, homogeneous** class distribution are preferred
- Need a measure M of node impurity:

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

Non-homogeneous,  
High degree of impurity

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

Homogeneous,  
Low degree of impurity

# Measures M of Node Impurity

---

- Entropy
- Gini Index
- Misclassification error

# Finding the Best Split

---

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

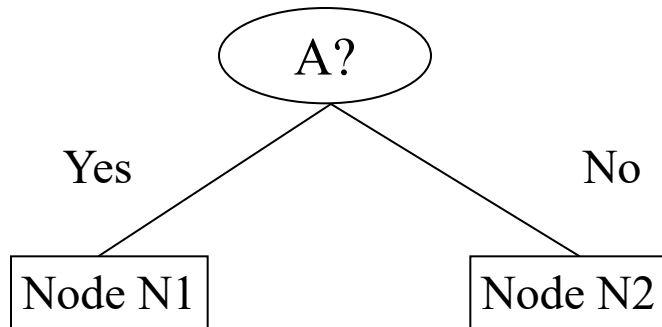
$$\text{Gain} = P - M$$

# Finding the Best Split

Before Splitting:

|    |            |
|----|------------|
| C0 | <b>N00</b> |
| C1 | <b>N01</b> |

→ P



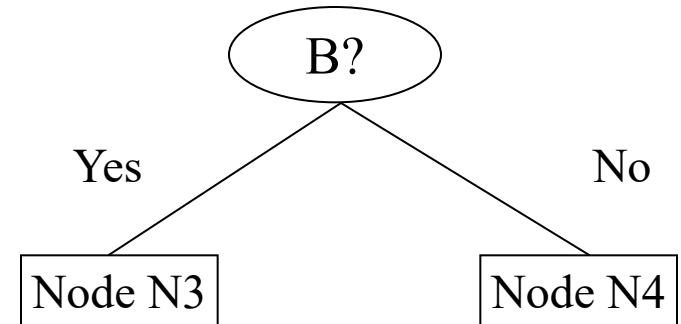
|    |            |
|----|------------|
| C0 | <b>N10</b> |
| C1 | <b>N11</b> |

|    |            |
|----|------------|
| C0 | <b>N20</b> |
| C1 | <b>N21</b> |

↓  
M11

↓  
M12

└───  
M1



|    |            |
|----|------------|
| C0 | <b>N30</b> |
| C1 | <b>N31</b> |

|    |            |
|----|------------|
| C0 | <b>N40</b> |
| C1 | <b>N41</b> |

↓  
M21

↓  
M22

└───  
M2

Gain = P – M1    vs    P – M2

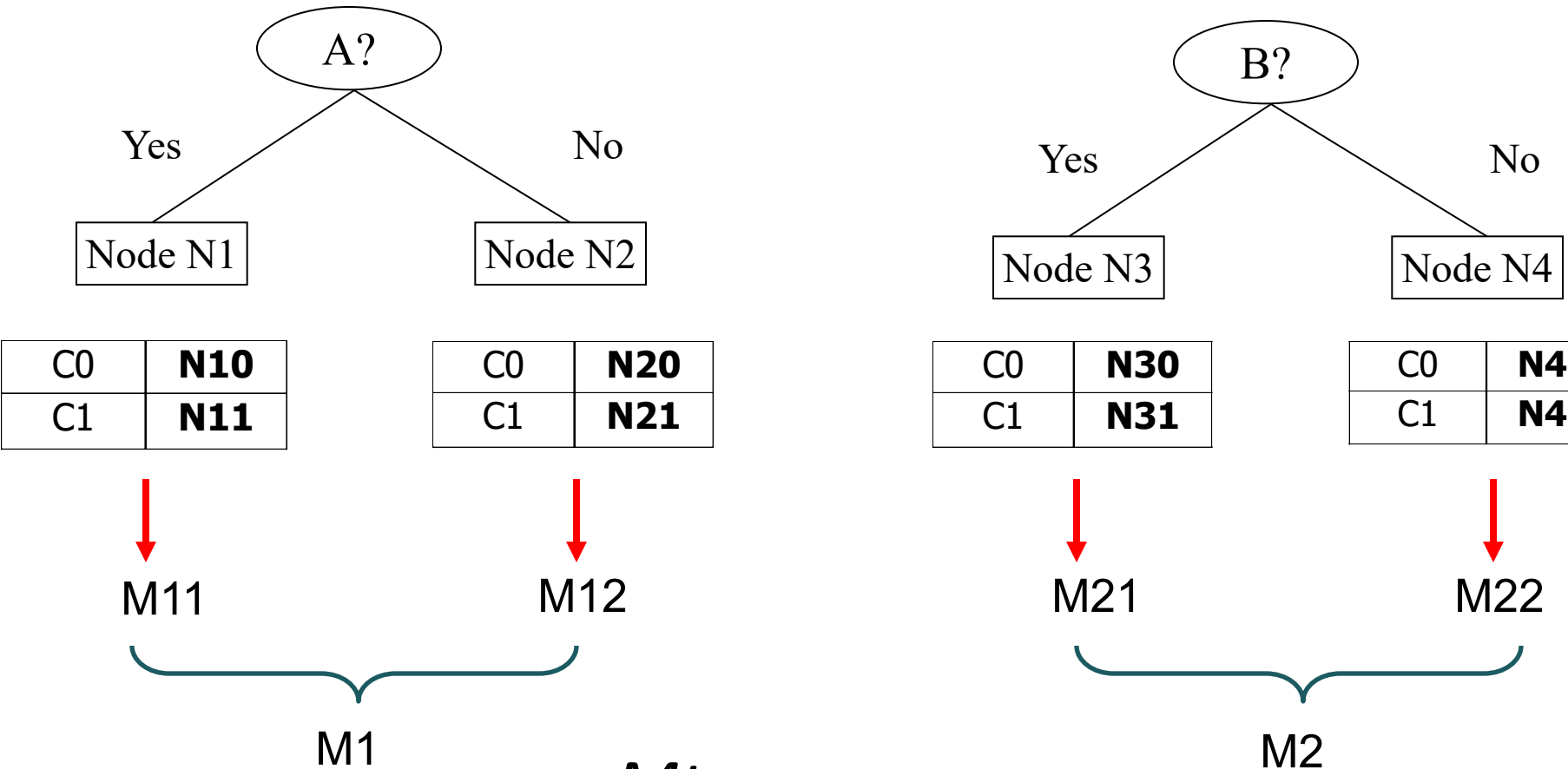


# Finding the Best Split

Before Splitting:

|    |            |
|----|------------|
| C0 | <b>N00</b> |
| C1 | <b>N01</b> |

→ P



$Mt =$

$$Entropy(t) = -\sum_j p(j | t) \log p(j | t)$$

## Measure of Impurity: Entropy

---

- Entropy at a given node  $t$ :

$$Entropy(t) = -\sum_j p(j | t) \log p(j | t)$$

(NOTE:  $p(j | t)$  is the relative frequency of class  $j$  at node  $t$ ).

- Maximum ( $\log n_c$ ) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information
- Entropy based computations are quite similar to the GINI index computations

# Computing Entropy of a Single Node

$$Entropy(t) = -\sum_j p(j | t) \log_2 p(j | t)$$

|    |          |
|----|----------|
| C1 | <b>0</b> |
| C2 | <b>6</b> |

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

$$Entropy = -0 \log 0 - 1 \log 1 = -0 - 0 = 0$$

|    |          |
|----|----------|
| C1 | <b>1</b> |
| C2 | <b>5</b> |

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

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

|    |          |
|----|----------|
| C1 | <b>2</b> |
| C2 | <b>4</b> |

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

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

# Splitting based on Information Gain

---

Information Gain:

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

- Parent Node,  $p$  is split into  $k$  partitions;
- $n_i$  is number of records in partition  $i$
- $GAIN_{split}$  measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
  - Used in ID3 and C4.5
- Disadvantage: bias toward attributes with large number of values
  - Large trees with many branches are preferred
  - What happens if there is an ID attribute?

# Splitting based on Gain Ratio

---

- Gain Ratio:

$$GainRatio_{split} = \frac{GAIN_{Split}}{SplitINFO} \quad SplitINFO = -\sum_{i=1}^k \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

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 algorithm
- Designed to overcome the disadvantage of Information Gain

# Split information

---

$$SplitINFO = -\sum_{i=1}^k \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

| <b>A = 1</b> | <b>A = 2</b> | <b>A = 3</b> | <b>A = 4</b> | <b><i>SplitINFO</i></b> |
|--------------|--------------|--------------|--------------|-------------------------|
| 32           | 0            | 0            | 0            | 0                       |
| 16           | 16           | 0            | 0            | 1                       |
| 16           | 8            | 8            | 0            | 1.5                     |
| 16           | 8            | 4            | 4            | 1.75                    |
| 8            | 8            | 8            | 8            | 2                       |

# Other measures of impurity

---

- Gini Index for a given node  $t$  :

$$GINI(t) = 1 - \sum_j [p(j | t)]^2$$

(NOTE:  $p(j | t)$  is the relative frequency of class  $j$  at node  $t$ )

- Classification error at a node  $t$  (*with classes  $i$* ):

$$Error(t) = 1 - \max_i P(i | t)$$

# Computing Gini Index of a Single Node

$$GINI(t) = 1 - \sum_j [p(j|t)]^2$$

|    |          |
|----|----------|
| C1 | <b>0</b> |
| C2 | <b>6</b> |

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

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

|    |          |
|----|----------|
| C1 | <b>1</b> |
| C2 | <b>5</b> |

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

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

|    |          |
|----|----------|
| C1 | <b>2</b> |
| C2 | <b>4</b> |

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

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



# Computing Error of a Single Node

$$Error(t) = 1 - \max_i P(i | t)$$

|    |          |
|----|----------|
| C1 | <b>0</b> |
| C2 | <b>6</b> |

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

$$Error = 1 - \max(0, 1) = 1 - 1 = 0$$

|    |          |
|----|----------|
| C1 | <b>1</b> |
| C2 | <b>5</b> |

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

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

|    |          |
|----|----------|
| C1 | <b>2</b> |
| C2 | <b>4</b> |

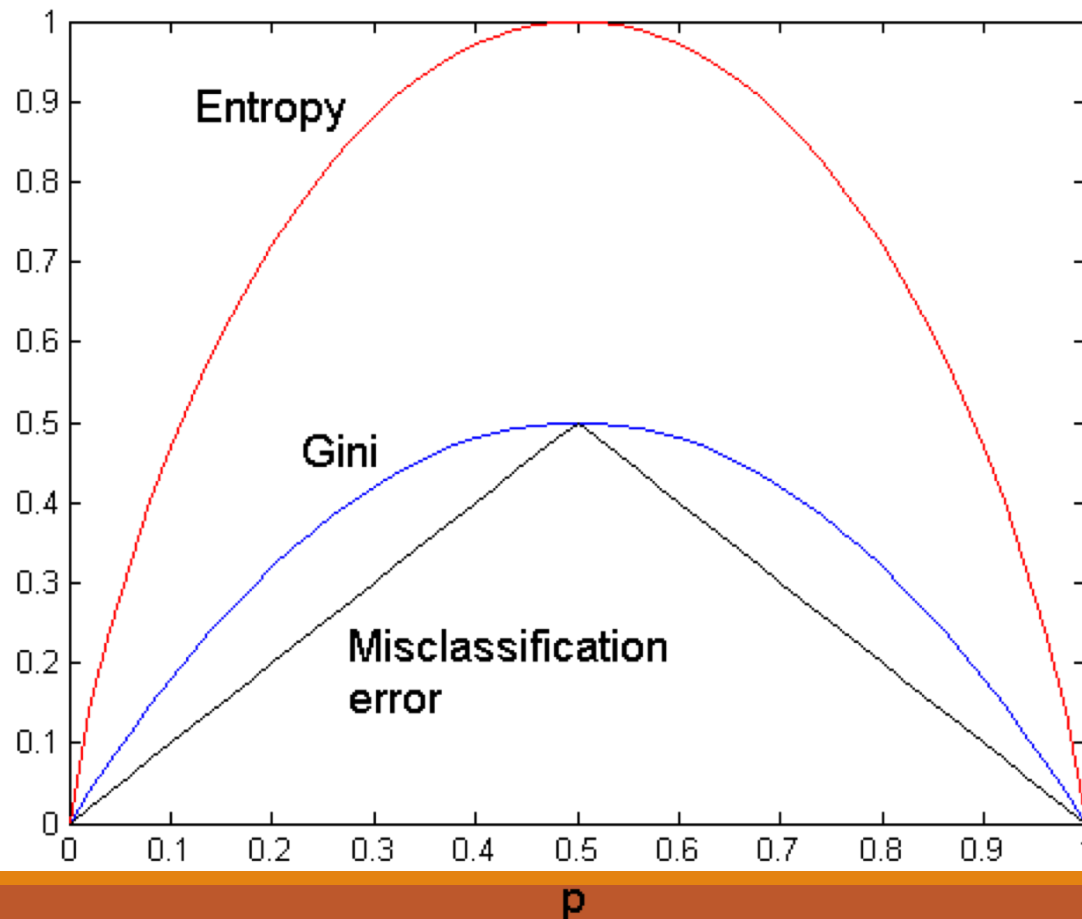
$$P(C1) = 2/6 \quad P(C2) = 4/6$$

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

If there are children nodes, they will be weighted depending on how many instances they contain

# Comparison among Impurity Measures

For a 2-class problem:



# Practical issues

---

- Conclusion: decision trees are built by greedy search algorithms, guided by some heuristic that measures “impurity”
- In real-world applications we need also to consider
  - Continuous attributes
  - Missing values – handled during preprocessing
  - Improving computational efficiency
  - Overfitted trees

# Splitting Based on Continuous Attributes

---

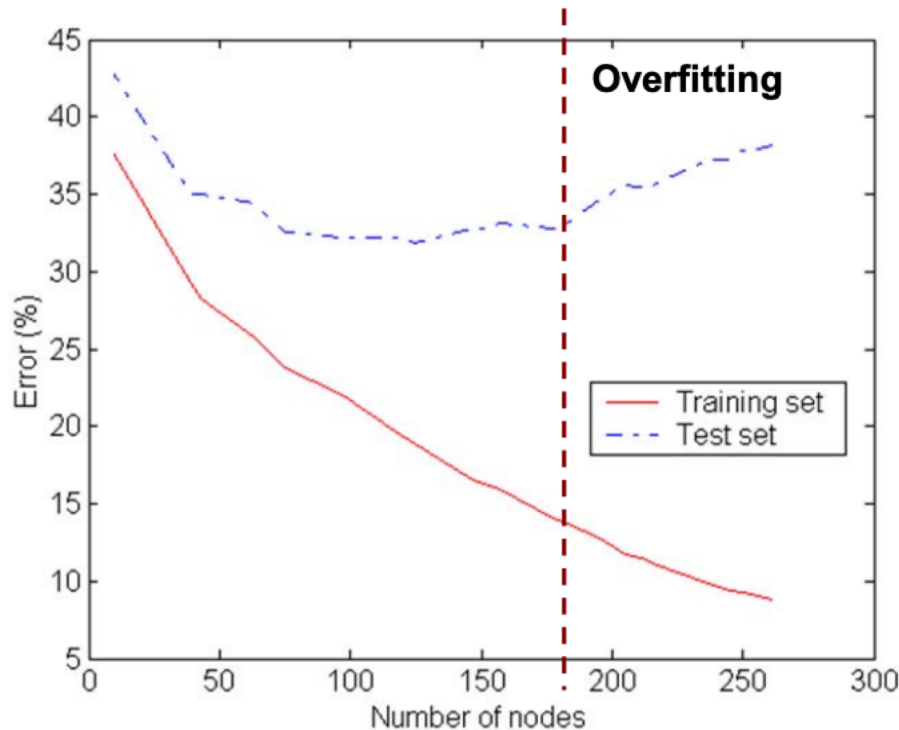
- Different ways of handling

- **Discretization** to form an ordinal categorical attribute

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

- Static – discretize once at the beginning
    - Dynamic – repeat at each node [computationally intensive!]
  - **Binary Decision:**  $(A < v)$  or  $(A \geq v)$ 
    - consider all possible splits and finds the best cut
    - can be more computationally intensive

# Overfitting and underfitting



When the tree becomes too large, its test error rate begins increasing while its training error rate continues to decrease.

**What causes overfitting?**

- Noise

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

# How to address underfitting

---

## Pre-Pruning (Early Stopping Rule)

- Stop the algorithm before it becomes a fully-grown tree
  - Stop if **all instances belong to the same class**
  - Stop if **all the attribute values are the same**
- Early stopping conditions:
  - Stop if number of instances is less than some **user-specified threshold**
    - e.g. 5 to 10 records per node
  - Stop if class distribution of instances are independent of the available attributes (e.g., using Chi Sq test)
  - Stop if splitting the current node improves the **impurity measure** (e.g. Gini or information gain) below a given **threshold**

# How to address overfitting

---

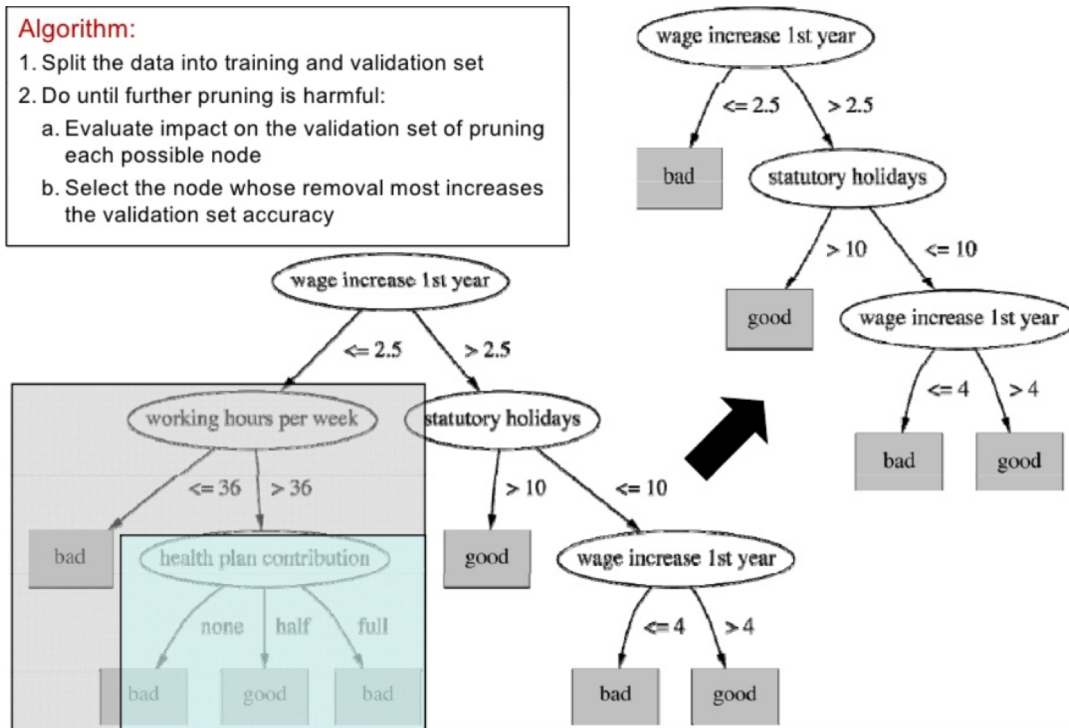
## Post-pruning

- Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
  - **Reduced-error pruning** *[remove subtrees if error does not increase]*
- Use a dataset not used in the training pruning set
  - Three data sets are needed: training set, test set, **pruning set**
- If test error in the pruning set improves after trimming then prune the tree
  - Post-pruning can be achieved in two ways:
    - **Sub-tree replacement:** selects a subtree and replaces it with a single leaf.
    - **Sub-tree raising:** selects a subtree and replaces it with the child one (ie, a "sub-subtree" is deleted)

# Subtree replacement

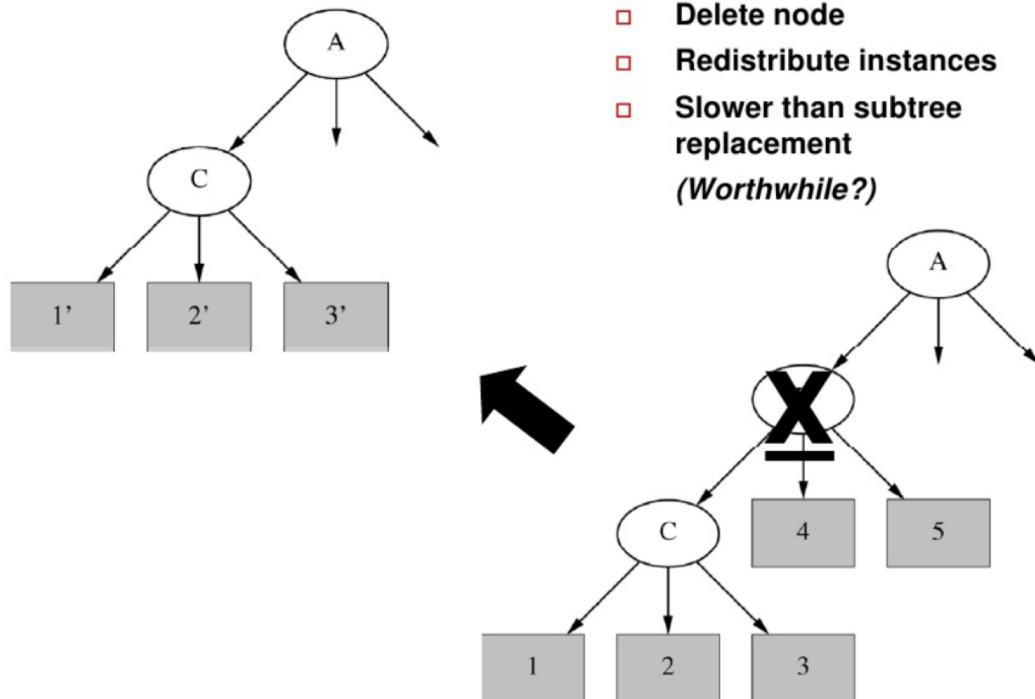
**Algorithm:**

1. Split the data into training and validation set
2. Do until further pruning is harmful:
  - a. Evaluate impact on the validation set of pruning each possible node
  - b. Select the node whose removal most increases the validation set accuracy





# Subtree raising



# ID3, C4.5, C5.0, CART

---

- Ross Quinlan
  - ID3 uses the Hunt's algorithm with information gain criterion and gain ratio
  - C4.5 (1994) improves ID3
    - Needs entire data to fit in memory
    - Handles missing attributes and continuous attributes
    - Performs tree post-pruning
  - C5.0 is the current commercial successor of C4.5
- Breiman et al.
  - CART builds multivariate decision (binary) trees, supports regression.

# Decision Tree Based Classification

---

- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Robust to noise (especially when methods to avoid overfitting are employed)
  - Can easily handle redundant or irrelevant attributes (unless the attributes are interacting)
- Disadvantages:
  - Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the best tree.
  - Does not take into account interactions between attributes
  - Each decision boundary involves only a single attribute

**RULES**

Rule  
based  
methods

---

# Example of a rule set

| Name          | Blood Type | Give Birth | Can Fly | Live in Water | Class      |
|---------------|------------|------------|---------|---------------|------------|
| human         | warm       | yes        | no      | no            | mammals    |
| python        | cold       | no         | no      | no            | reptiles   |
| salmon        | cold       | no         | no      | yes           | fishes     |
| whale         | warm       | yes        | no      | yes           | mammals    |
| frog          | cold       | no         | no      | sometimes     | amphibians |
| komodo        | cold       | no         | no      | no            | reptiles   |
| bat           | warm       | yes        | yes     | no            | mammals    |
| pigeon        | warm       | no         | yes     | no            | birds      |
| cat           | warm       | yes        | no      | no            | mammals    |
| leopard shark | cold       | yes        | no      | yes           | fishes     |
| turtle        | cold       | no         | no      | sometimes     | reptiles   |
| penguin       | warm       | no         | no      | sometimes     | birds      |
| porcupine     | warm       | yes        | no      | no            | mammals    |
| eel           | cold       | no         | no      | yes           | fishes     |
| salamander    | cold       | no         | no      | sometimes     | amphibians |
| gila monster  | cold       | no         | no      | no            | reptiles   |
| platypus      | warm       | no         | no      | no            | mammals    |
| owl           | warm       | no         | yes     | no            | birds      |
| dolphin       | warm       | yes        | no      | yes           | mammals    |
| eagle         | warm       | no         | yes     | no            | birds      |

**R1:** (Give Birth = no)  $\wedge$  (Can Fly = yes)  $\rightarrow$  Birds

**R2:** (Give Birth = no)  $\wedge$  (Live in Water = yes)  $\rightarrow$  Fishes

**R3:** (Give Birth = yes)  $\wedge$  (Blood Type = warm)  $\rightarrow$  Mammals

**R4:** (Give Birth = no)  $\wedge$  (Can Fly = no)  $\rightarrow$  Reptiles

**R5:** (Live in Water = sometimes)  $\rightarrow$  Amphibians

# Application of rules - covering

A rule **r** **covers** an instance **x** if the attributes of the instance satisfy the condition (LHS) of the rule

- $R1: (Give\ Birth = no) \wedge (Can\ Fly = yes) \rightarrow Birds$
- $R2: (Give\ Birth = no) \wedge (Live\ in\ Water = yes) \rightarrow Fishes$
- $R3: (Give\ Birth = yes) \wedge (Blood\ Type = warm) \rightarrow Mammals$
- $R4: (Give\ Birth = no) \wedge (Can\ Fly = no) \rightarrow Reptiles$
- $R5: (Live\ in\ Water = sometimes) \rightarrow Amphibians$

The rule **R1** **covers** a hawk  $\Rightarrow$  Class = Bird

| Name         | Blood Type | Give Birth | Can Fly | Live in Water | Class |
|--------------|------------|------------|---------|---------------|-------|
| hawk         | warm       | no         | yes     | no            | ?     |
| grizzly bear | warm       | yes        | no      | no            | ?     |

# From rules to decision tree

---

Consider the rule set

- Attributes A, B, C, and D can have values 1, 2, and 3

**A = 1  $\wedge$  B = 1  $\rightarrow$  Class = Y**

**C = 1  $\wedge$  D = 1  $\rightarrow$  Class = Y**

**Otherwise, Class = N**

How to represent it as a decision tree?

- The rules need a common attribute

**A = 1  $\wedge$  B = 1  $\rightarrow$  Class = Y**

**A = 1  $\wedge$  B = 2  $\wedge$  C = 1  $\wedge$  D = 1  $\rightarrow$  Class = Y**

**A = 1  $\wedge$  B = 3  $\wedge$  C = 1  $\wedge$  D = 1  $\rightarrow$  Class = Y**

**A = 2  $\wedge$  C = 1  $\wedge$  D = 1  $\rightarrow$  Class = Y**

**A = 3  $\wedge$  C = 1  $\wedge$  D = 1  $\rightarrow$  Class = Y**

**Otherwise, Class = N**

More verbose!

# Building classification rules

---

- Direct Method

- Extract rules directly from data
- e.g.: RIPPER, Holte's 1R (OneR)

- Indirect Method

- Extract rules from other classification models (e.g. decision trees, etc).
- e.g: C4.5rules



# A direct method: sequential covering

---

- Let  $E$  be the training set
  - Extract rules one class at a time

For each class  $C$

1. Initialize set  $S$  with  $E$
2. While  $S$  contains instances in class  $C$ 
  3. Learn one rule  $R$  for class  $C$
  4. Remove training records covered by the rule  $R$

Goal: to create rules that cover many examples of a class  $C$  and none (or very few) of other classes

# Direct method: RIPPER

---

- For 2-class problem, choose one of the classes as positive class, and the other as negative class
  - Learn rules for positive class
  - **Negative class will be default class**
- For multi-class problem
  - **Order** the classes according to **increasing class prevalence** (fraction of instances that belong to a particular class)
  - Learn the rule set for smallest class first, treat the rest as negative class
  - Repeat with next smallest class as positive class

# Direct method: RIPPER

---

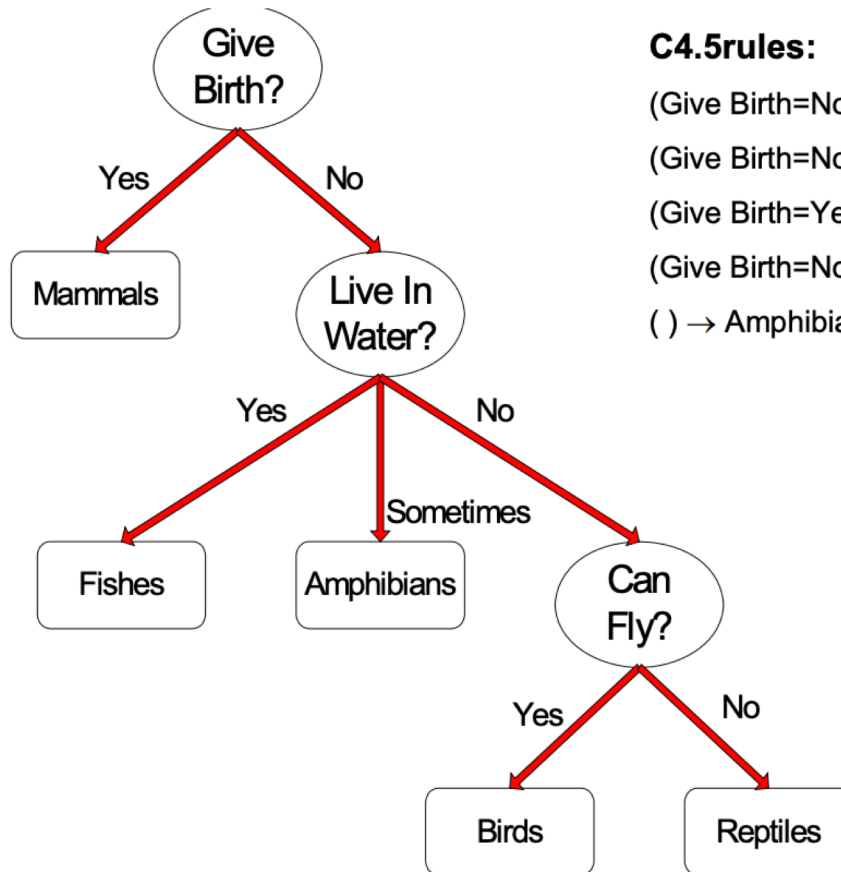
- Start from empty rule
- Add conjuncts as long as they improve information gain (FOIL)
  - Stop when rule no longer covers negative examples
- Prune the rule immediately using **reduced error pruning**
  - Measure for pruning:  $W(R) = (p-n)/(p+n)$ 
    - $p$ : number of positive examples covered by the rule in the validation set
    - $n$ : number of negative examples covered by the rule in the validation set
- Pruning starts from the last test added to the rule
  - May create rules that cover some negative examples (accuracy < 1)
  - A global optimization (pruning) strategy is also applied

# Direct method: RIPPER

---

- Use sequential covering algorithm
  - Find the best rule that covers the current set of positive examples
  - Eliminate both positive and negative samples covered by the rule from the dataset
- Each time a rule is added to the rule set, compute the new description length
  - Stop adding new rules when the new description length is  $d$  bits longer than the smallest description length obtained so far.

# C4.5 rules versus RIPPER



## C4.5rules:

(Give Birth=No, Can Fly=Yes) → Birds

(Give Birth=No, Live in Water=Yes) → Fishes

(Give Birth=Yes) → Mammals

(Give Birth=No, Can Fly=No, Live in Water=No) → Reptiles

( ) → Amphibians

## RIPPER:

(Live in Water=Yes) → Fishes

(Have Legs=No) → Reptiles

(Give Birth=No, Can Fly=No, Live In Water=No)  
→ Reptiles

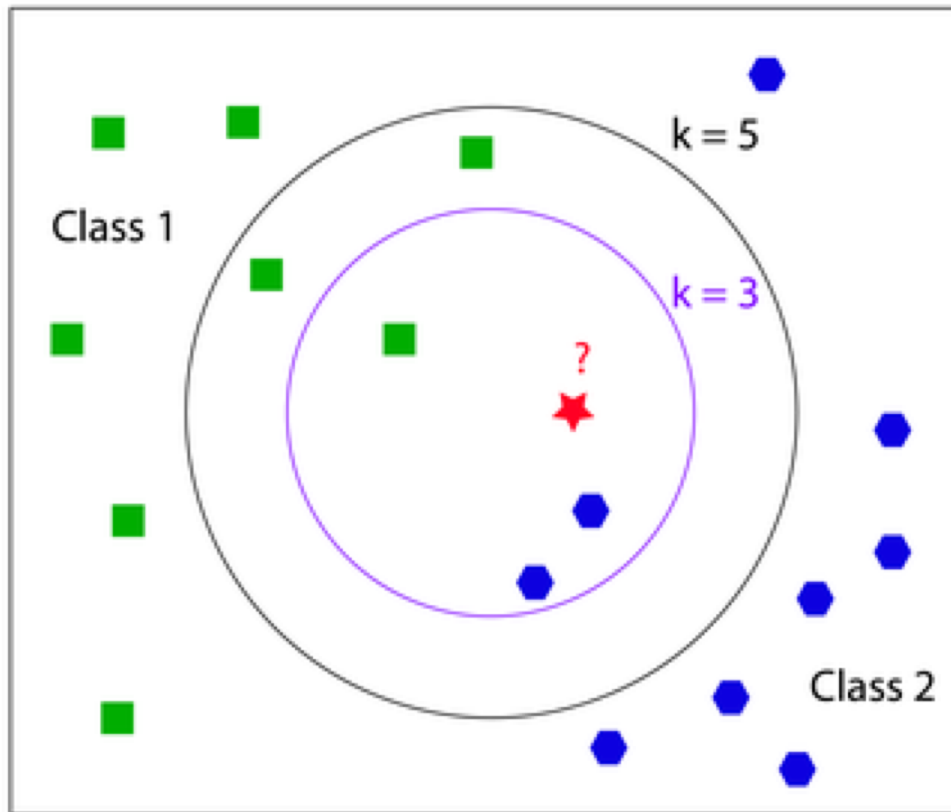
(Can Fly=Yes, Give Birth=No) → Birds

( ) → Mammals

# Advantages

---

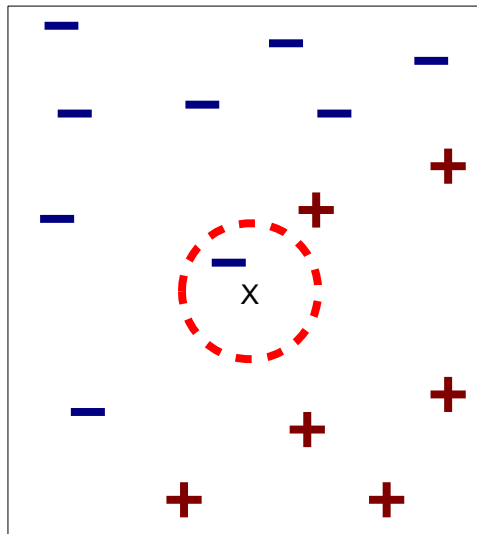
- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees
- Can easily handle missing values and numeric attributes



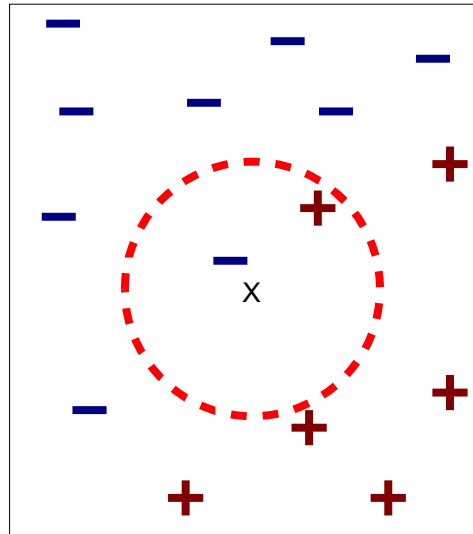
K-nearest  
neighbour

# k-Nearest Neighbour

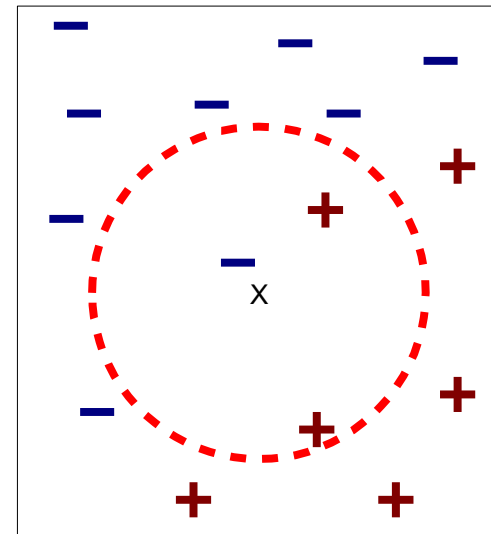
- Assign observation to nearest neighbour, or k-nearest neighbour (with majority voting)
- Euclidean distance (takes a while to compute for all pairs)



(a) 1-nearest neighbor



(b) 2-nearest neighbor



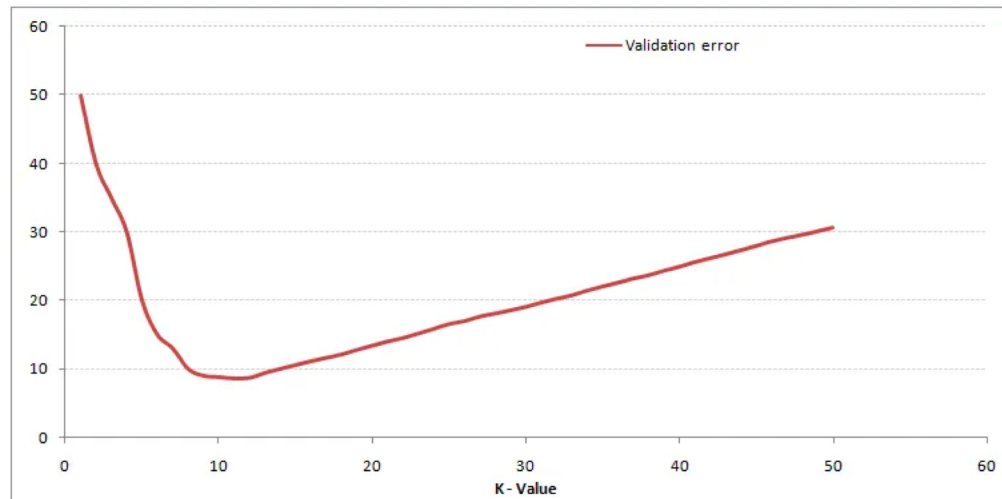
(c) 3-nearest neighbor



# How to choose k

---

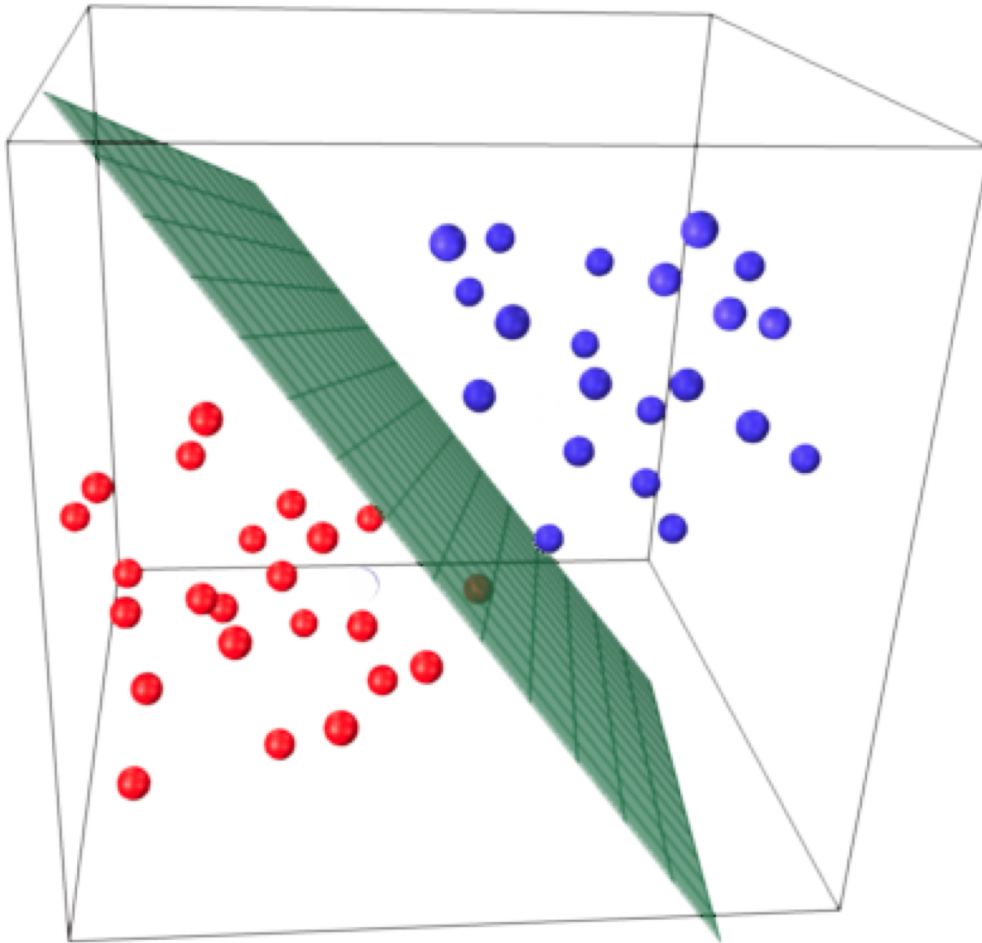
- Make this plot for your data:



# Characteristics of k-NN

---

- Straightforward algorithm
  - Simply assign to nearest neighbour or majority voting
- Sensitive to value of  $k$ 
  - Too small, overly sensitive to noise/outliers
  - Too large, neighbours include points from other classes
- Computationally expensive
  - Need to compute distances to all points (but can speed up)
- Space requirement
  - Need to store all data points



# Logistic regression

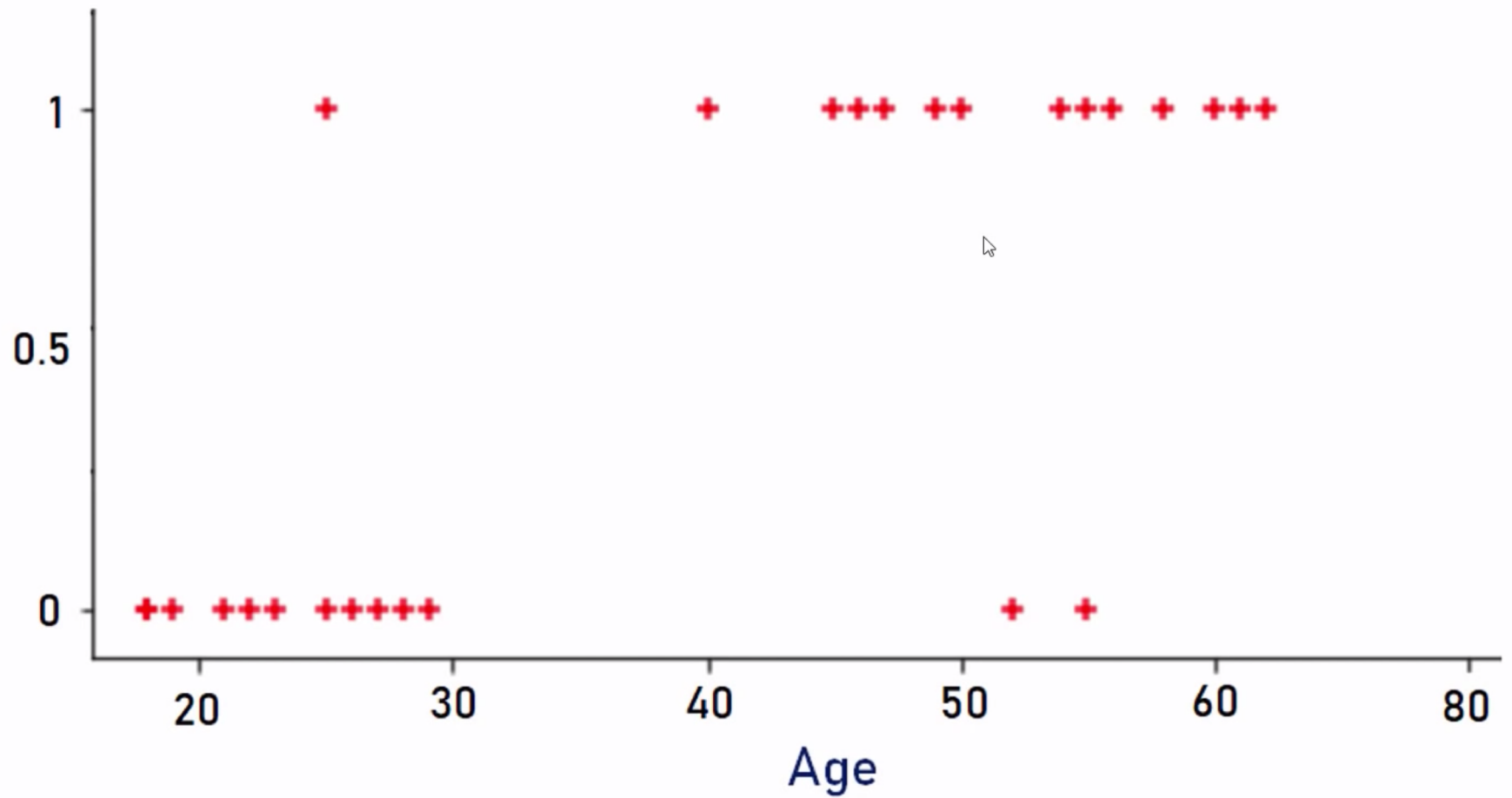
# Binary classification

---

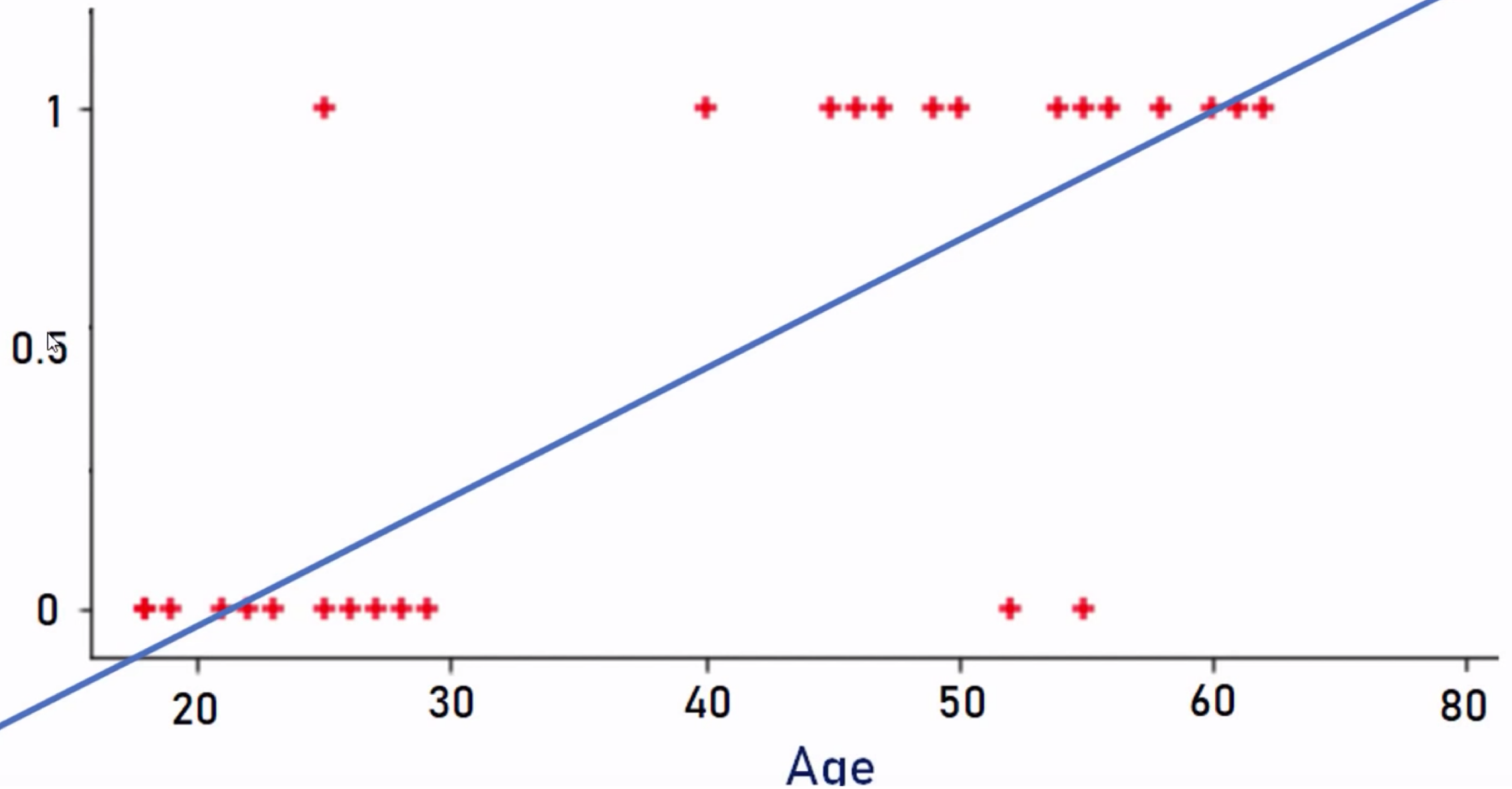
| age | have_insurance |
|-----|----------------|
| 22  | 0              |
| 25  | 0              |
| 47  | 1              |
| 52  | 0              |
| 46  | 1              |
| 56  | 1              |
| 55  | 0              |
| 60  | 1              |
| 62  | 1              |
| 61  | 1              |
| 18  | 0              |
| 28  | 0              |
| 27  | 0              |
| 29  | 0              |
| 49  | 1              |

<https://www.youtube.com/watch?v=zM4VZR0px8E>

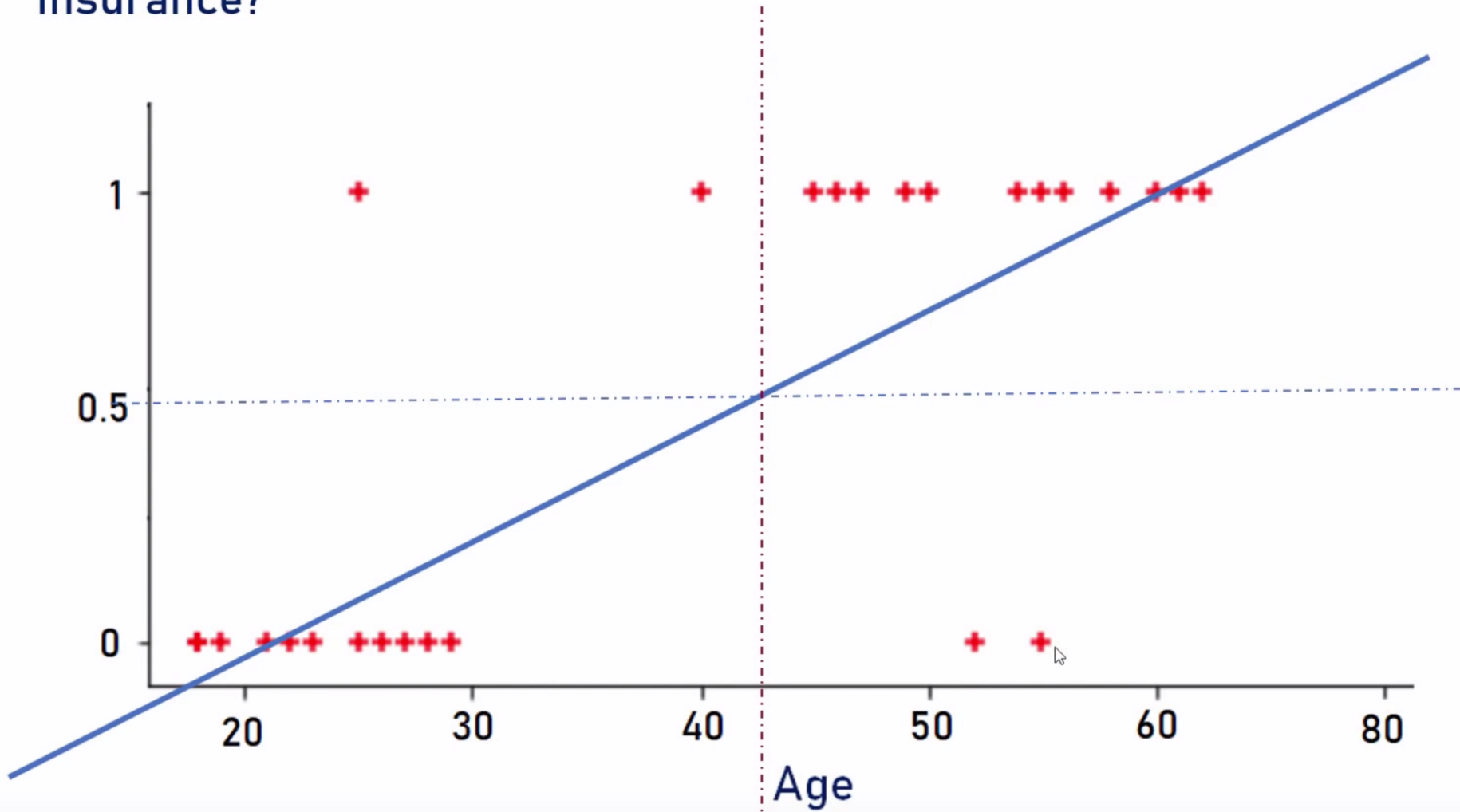
Have  
Insurance?



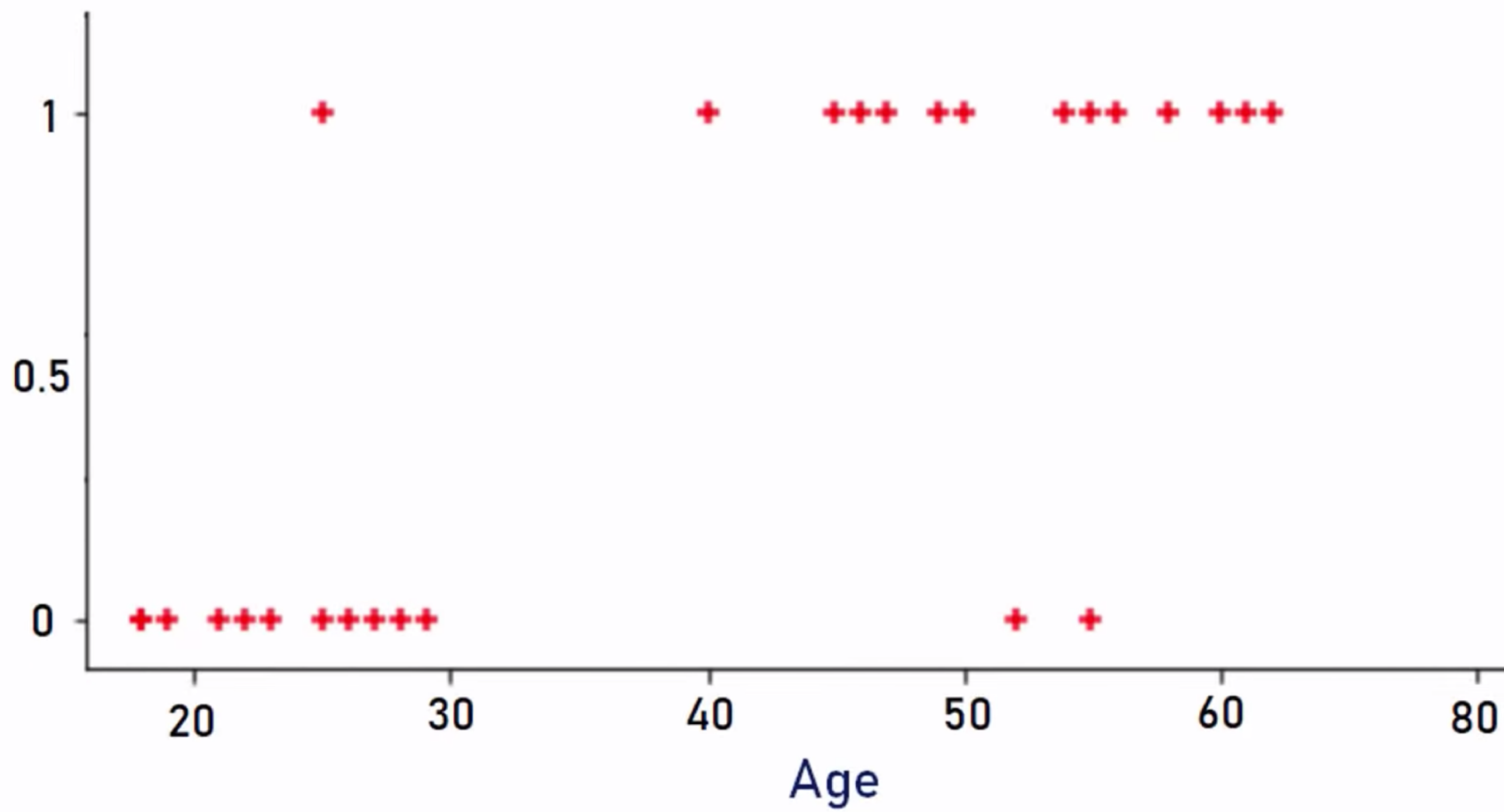
Have  
Insurance?



Have  
Insurance?

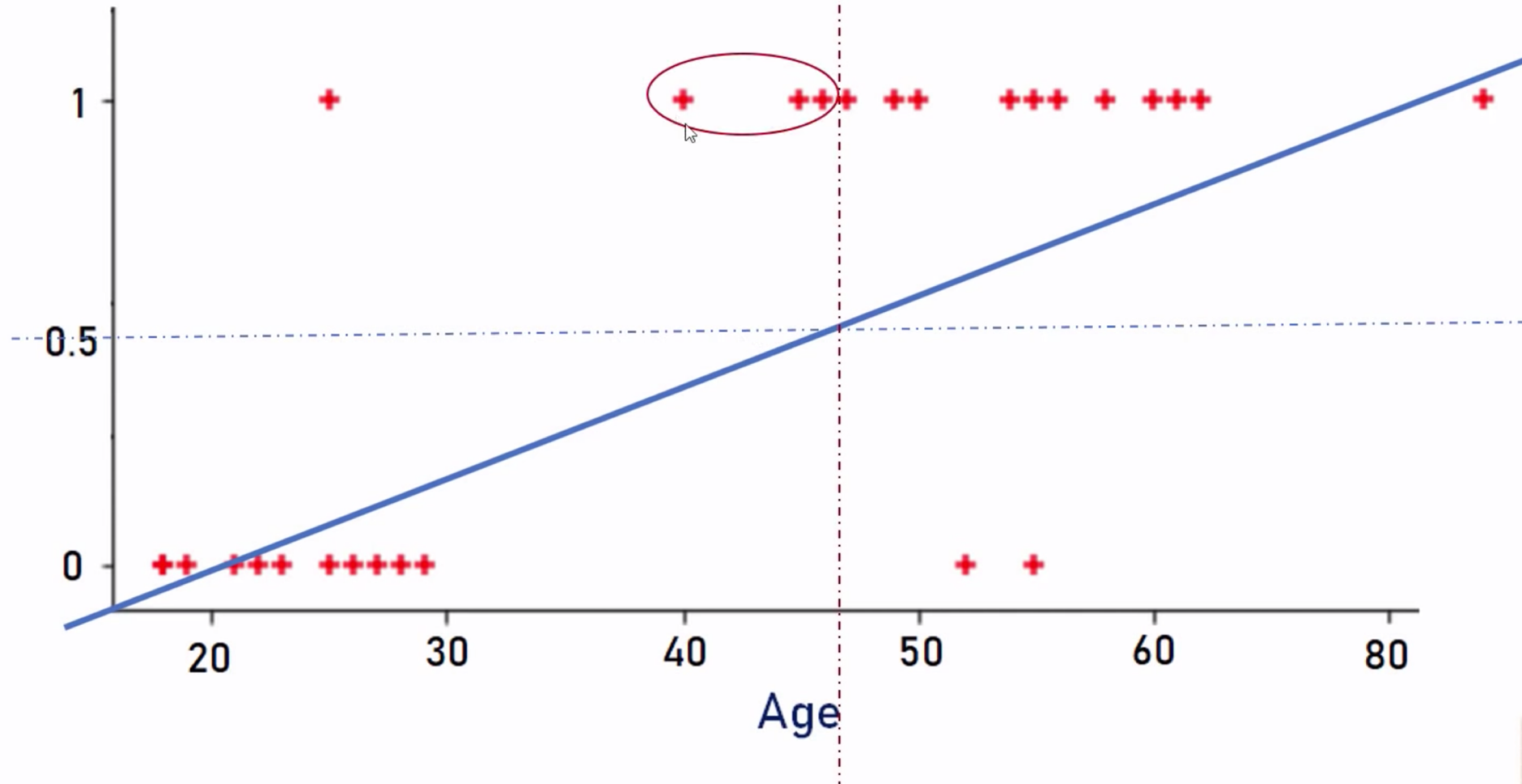


Have  
Insurance?

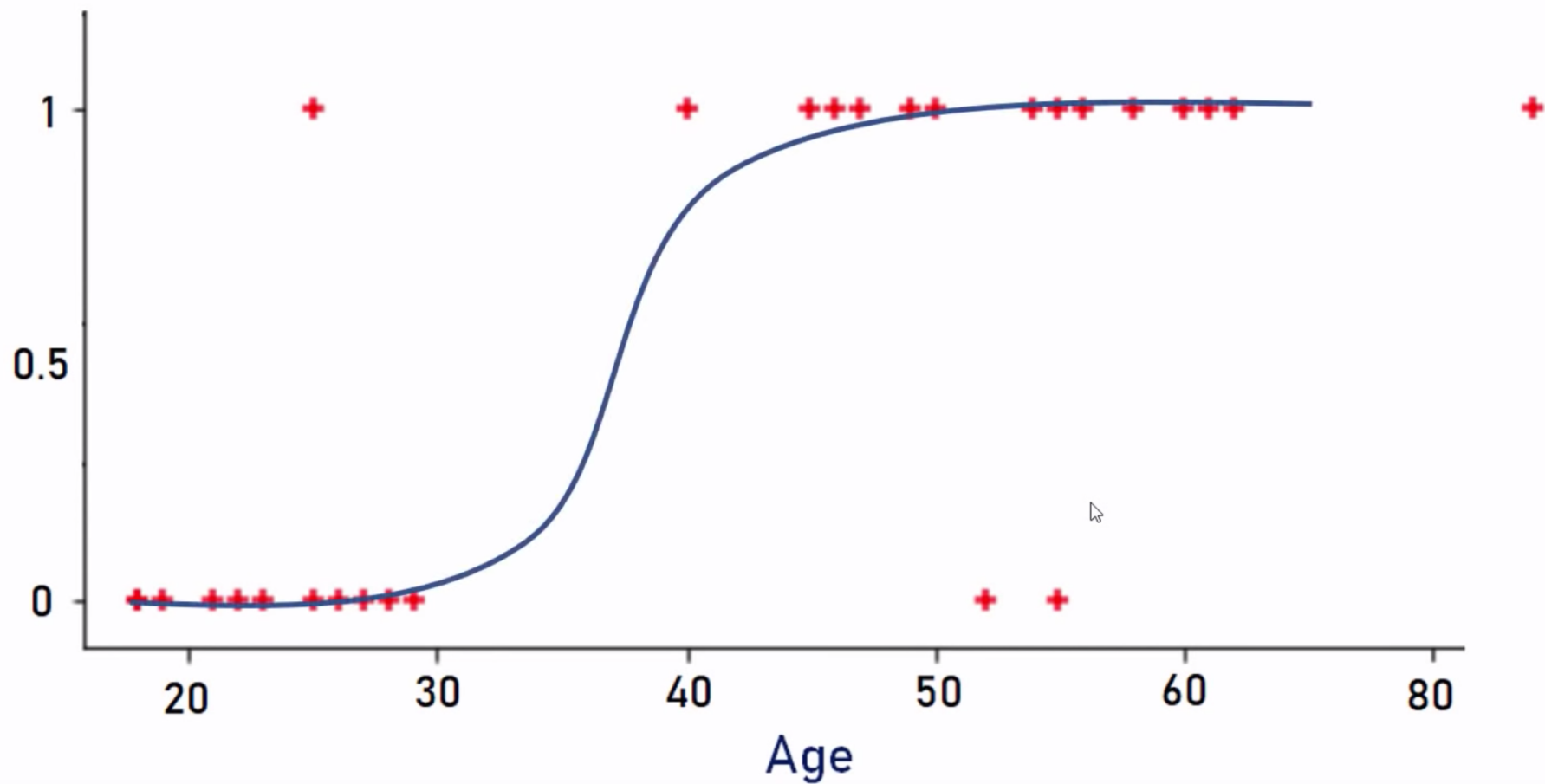




Have  
Insurance?

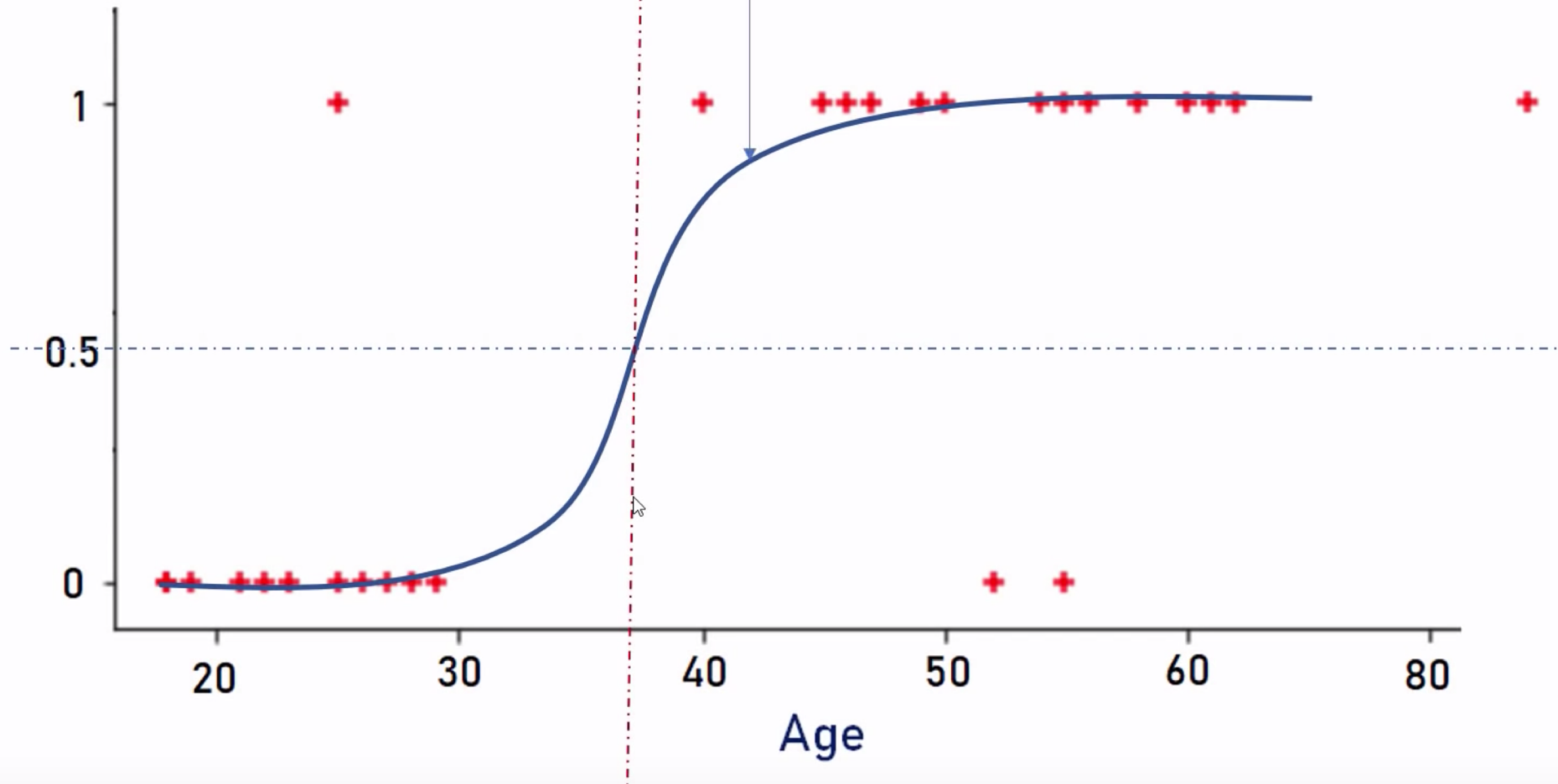


Have  
Insurance?

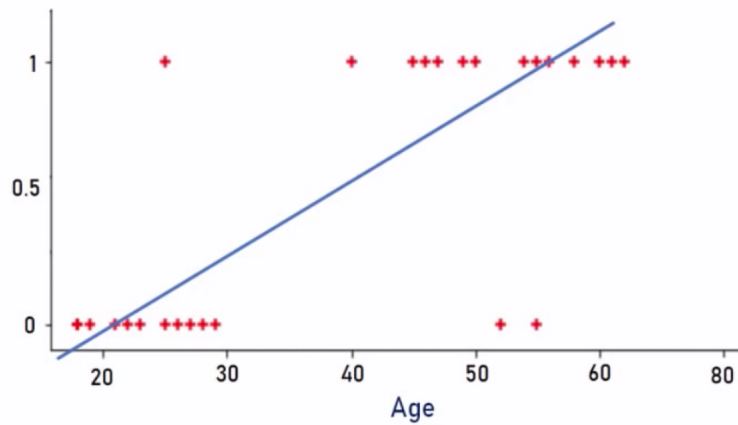


## Sigmoid or Logit Function

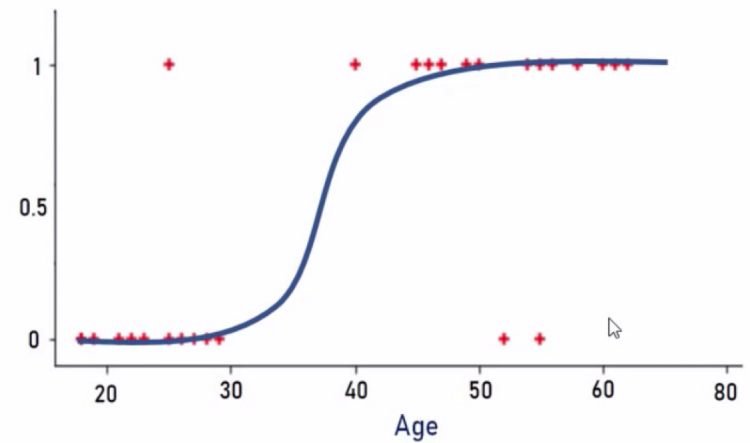
Have  
Insurance?



$$y = m * x + b$$



$$y = \frac{1}{1 + e^{-(m*x+b)}}$$



# Threshold value

---

- Always 0.5?
- No. Choose a good value based on confusion matrix

|           |          | Actual                |                       |
|-----------|----------|-----------------------|-----------------------|
|           |          | Positive              | Negative              |
| Predicted | Positive | <b>True Positive</b>  | <b>False Positive</b> |
|           | Negative | <b>False Negative</b> | <b>True Negative</b>  |

# Logit

---

Logistic regression (multiple variables):

- $P(\text{class is 1}) = \frac{1}{1+e^{-(m_1 * x_1 + m_2 * x_2 + \dots + b)}}$
- $P(\text{class is 0}) = 1 - \frac{1}{1+e^{-(m_1 * x_1 + m_2 * x_2 + \dots + b)}}$

Logit:

Class is 1 when:

$$\text{Odds} = \frac{P(\text{class is 1})}{P(\text{class is 0})} = e^{(m_1 * x_1 + m_2 * x_2 + \dots + b)} > 1$$

Logit(P):  $\log(\text{Odds}) = m_1 * x_1 + m_2 * x_2 + \dots + b \sim \text{linear regression}$

# Deriving logit

$$\begin{aligned}\frac{P(\text{class 1})}{P(\text{class 0})} &= \frac{\frac{1}{1+e^{-z}}}{1 - \frac{1}{1+e^{-z}}} = \frac{\frac{1}{1+e^{-z}}}{\frac{(1+e^{-z})-1}{1+e^{-z}}} \\ &= \frac{\frac{1}{1+e^{-z}}}{\frac{e^{-z}}{1+e^{-z}}} = \frac{\frac{1}{\cancel{1+e^{-z}}}}{\frac{e^{-z}}{\cancel{1+e^{-z}}}} \cdot \cancel{(1+e^{-z})} = \frac{1}{e^{-z}} = \underline{\underline{e^z}}\end{aligned}$$

with  $z = m_1 x_1 + m_2 x_2 + \dots + b$

# Multiclass classification

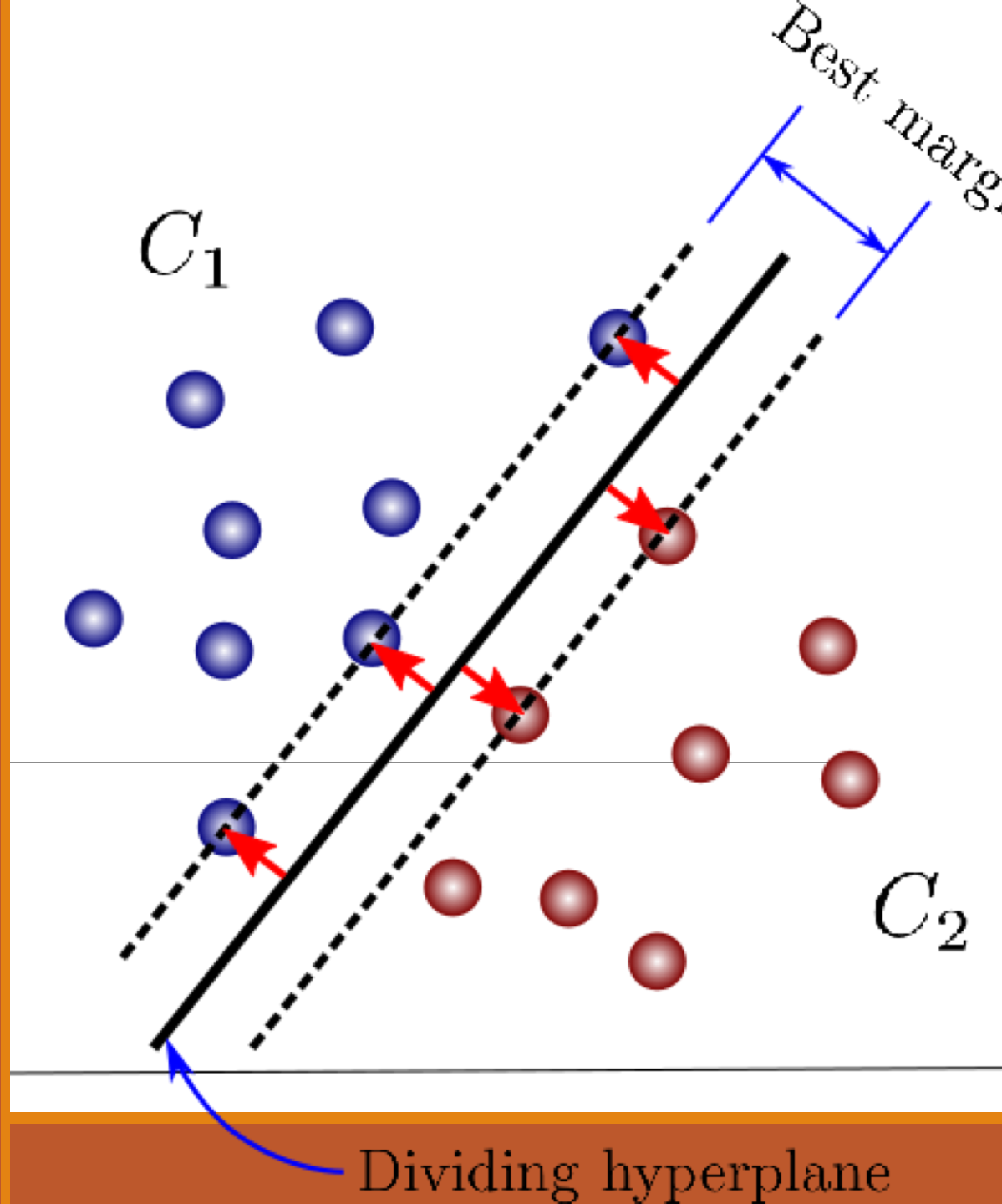
---

- In this case, a regression model is learned for each class, highest prediction = class.
- Disadvantage: independent predictions, models are built independent of each other.
- Example: cheap, expensive, car, tree  
*If cars are expensive, and a car is detected, the model for expensive will not know there is a car detected.*



# Support vector machines

SVM



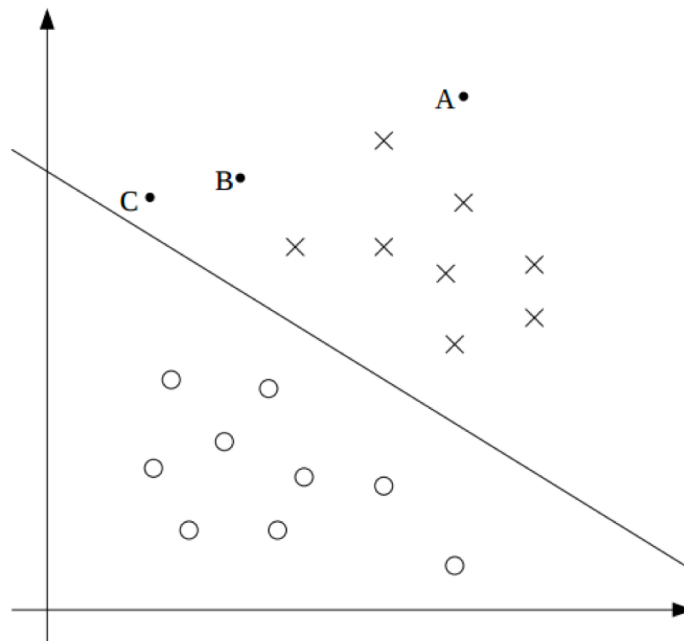
# SVM

---

- A classifier derived from statistical learning theory by Vapnik, et al. in 1992
- Became famous as it performed as well as neural networks on the handwriting recognition task
- Widely used in:
  - Object detection & recognition
  - Content-based image retrieval
  - Text recognition
  - Biometrics
  - Speech recognition
  - ...

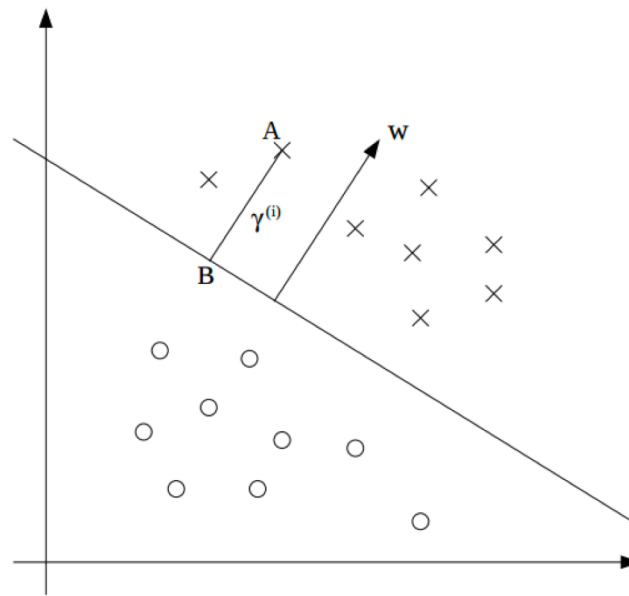
# Margins & confidence

---



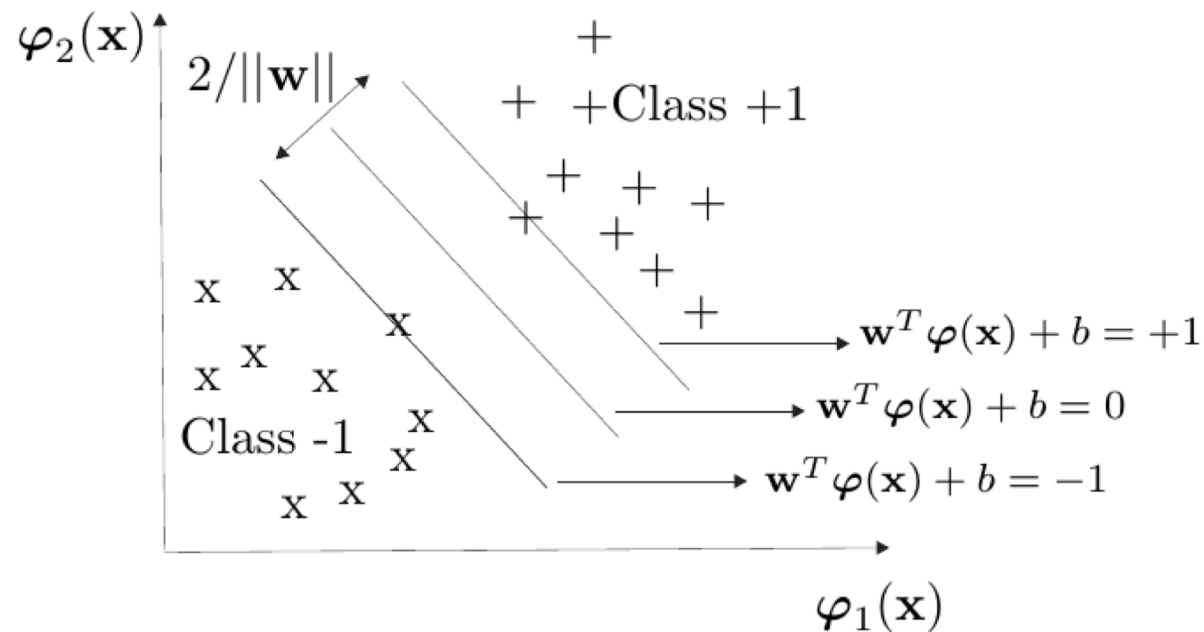
# Geometric margins

---



(Ng, 2014)

# Optimal margins



# Optimal margins

---

Given a training set of  $N$  data points  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$  with input data  $\mathbf{x}_i \in \mathbb{R}^n$  and corresponding binary class labels  $y_i \in \{-1, +1\}$ , the SVM classifier should fulfil following conditions. (vapnik, 1995):

$$\begin{cases} \mathbf{w}^T \varphi(\mathbf{x}_i) + b \geq +1, & \text{if } y_i = +1 \\ \mathbf{w}^T \varphi(\mathbf{x}_i) + b \leq -1, & \text{if } y_i = -1 \end{cases} \quad (5)$$

which is equivalent to

$$y_i[\mathbf{w}^T \varphi(\mathbf{x}_i) + b] \geq 1, \quad i = 1, \dots, N.$$

# SVM

---

- What's the:  $\varphi(\cdot)$ ?
- What if data is non-linearly separable?

SVM with a polynomial  
Kernel visualization

Created by:  
Udi Aharoni

# SVM

---

- ▶  $\varphi(\cdot)$  is a non-linear function that maps the input space to a high (possibly infinite) dimensional feature space
- ▶ Equation 5 constructs a hyperplane  $\mathbf{w}^T \varphi(\mathbf{x}) + b = 0$  between the two classes
- ▶ Goal: maximize margin between both classes  
→ minimizing  $\mathbf{w}^T \mathbf{w}$



# SVM (optional)

---

This convex optimization problem is defined as:

$$\min_{\mathbf{w}, b, \xi} \mathcal{J}(\mathbf{w}, b, \xi) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \xi_i \quad (7)$$

subject to

$$\begin{cases} y_i [\mathbf{w}^T \varphi(\mathbf{x}_i) + b] \geq 1 - \xi_i, & i = 1, \dots, N \\ \xi_i \geq 0, & i = 1, \dots, N. \end{cases} \quad (8)$$

- ▶  $\xi_i$  are slack variables which are needed to allow misclassifications in the set of inequalities
  - ▶  $C$  is the regularisation coefficient  $\rightarrow$  tuned
-

# SVM (optional)

---

- The objective function (Eq. 7):
  - First part:
    - max. margin between both classes in the feature space
    - regularisation mechanism that penalizes large weights
  - Second part:
    - min. the misclassification error

# SVM (optional)

---

This leads to the following classifier (Cristianini & Shawe-Taylor, 2000):

$$y(\mathbf{x}) = \text{sign}[\sum_{i=1}^N \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b], \quad (9)$$

- ▶ The Lagrange multipliers  $\alpha_i$  are determined by optimizing the dual problem
- ▶ Low-noise problems, many of the  $\alpha_i$  will typically be equal to zero (sparseness property)
- ▶ Support vectors: the training observations corresponding to non-zero  $\alpha_i$

# SVM

---

$K(\mathbf{x}_i, \mathbf{x}) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x})$  is taken with a positive definite kernel satisfying the Mercer theorem.

The following kernel functions  $K(\cdot, \cdot)$  were used:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}_i) &= (1 + \mathbf{x}_i^T \mathbf{x} / c)^d, & (\text{polynomial kernel}) \\ K(\mathbf{x}, \mathbf{x}_i) &= \exp\{-\|\mathbf{x} - \mathbf{x}_i\|_2^2 / \sigma^2\}, & (\text{RBF kernel}) \end{aligned}$$

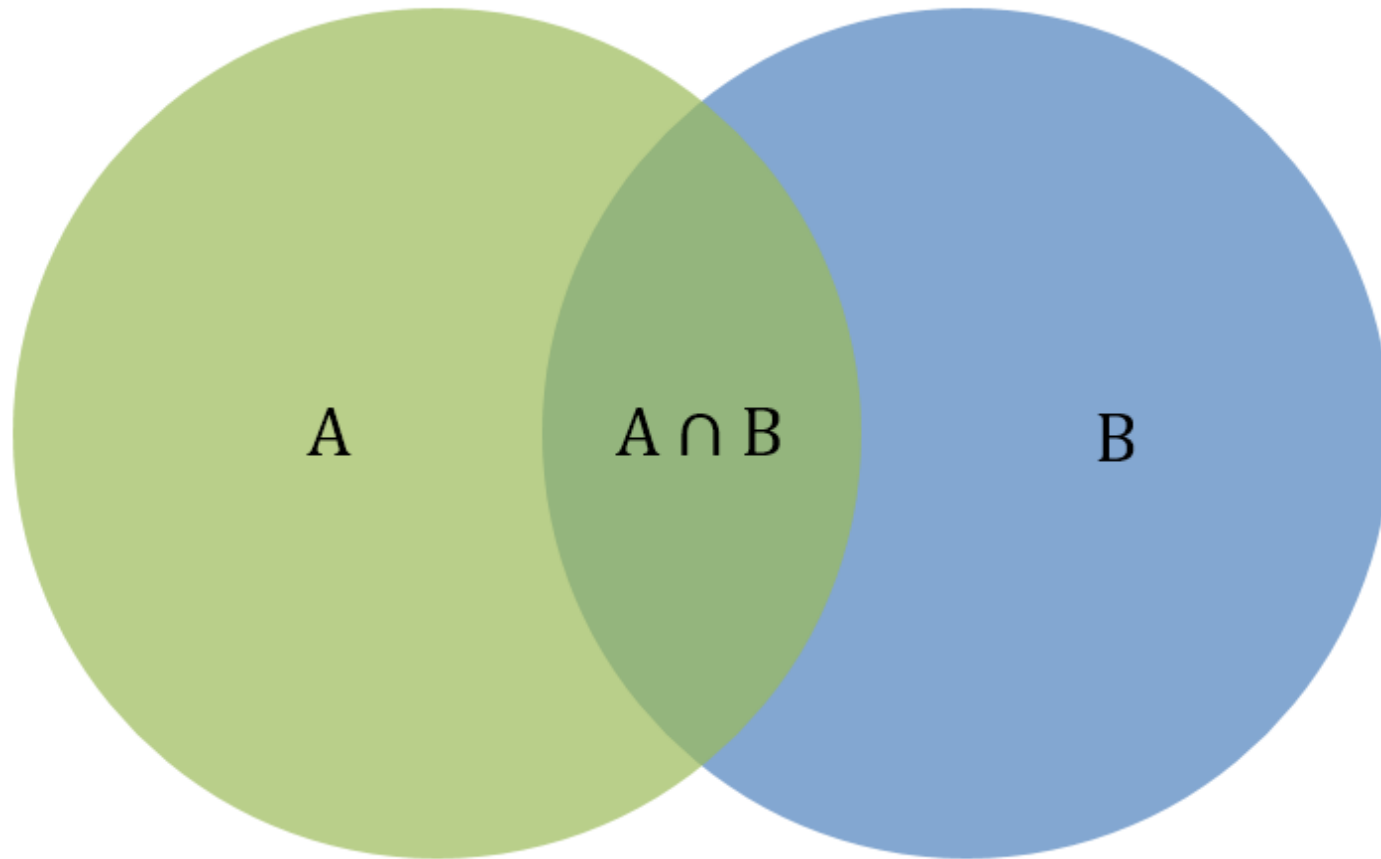
where  $d$ ,  $c$  and  $\sigma$  are constants.

→ black-box model

# Tuning hyperparameters

---

- GridSearch
  - Grid is determined by the two input parameters
  - 2-fold cross validation on the initial grid
  - 10-fold cross validation on the best point and its adjacent point (based on the weighted AUC by class size)
  - If a better pair is found: repeat procedure on its neighbours
  - Stop: no better pair found or border reached



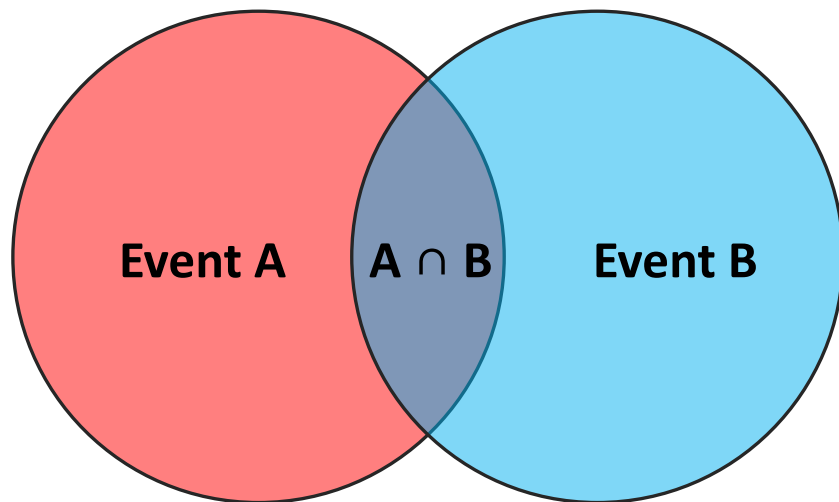
# Naïve Bayes

# Recap on Bayes Theorem

- Conditional Probability:

$$P(A | B) = \frac{P(A, B)}{P(B)}$$

$$P(B | A) = \frac{P(A, B)}{P(A)}$$



|         | Female | Male | Total |
|---------|--------|------|-------|
| Teacher | 8      | 12   | 20    |
| Student | 32     | 48   | 80    |
| Total   | 40     | 60   | 100   |

$$P(\text{Teacher} | \text{Male}) = \frac{P(\text{Teacher} \cap \text{Male})}{P(\text{Male})} = 12/60 = 0.2$$

# Recap on Bayes Theorem

---

- Conditional Probability:

$$P(A | B) = \frac{P(A, B)}{P(B)}$$

$$P(B | A) = \frac{P(A, B)}{P(A)}$$

$$P(A, B) = P(B | A) P(A)$$

- Bayes Theorem:

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$



# Naïve Bayes

---

- In real world problem: multiple X-variables
- **Assumes the features are independent:**

$$P(A = k | X_1 \dots X_n) = \frac{P(X_1 | A = k) * P(X_2 | A = k) * \dots * P(X_n | A = k) * P(A = k)}{P(X_1) * P(X_2) * \dots * P(X_n)}$$

$$\text{Probability of Outcome} \mid \textit{Evidence} = \frac{\textit{Probability of Likelihood of evidence} * \textit{Prior}}{\textit{Probability of Evidence}}$$

(Posterior Probability)

# Naïve Bayes example

---

- Dataset with probabilities:

| Fruit  | Long [x1] | Sweet [x2] | Yellow [x3] |
|--------|-----------|------------|-------------|
| Orange | 0         | 1          | 0           |
| Banana | 1         | 0          | 1           |
| Banana | 1         | 1          | 1           |
| Other  | 1         | 1          | 0           |
| ..     | ..        | ..         | ..          |

- Aggregated to a counts table:

| Type   | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|--------|------|----------|-------|-----------|--------|------------|-------|
| Banana | 400  | 100      | 350   | 150       | 450    | 50         | 500   |
| Orange | 0    | 300      | 150   | 150       | 300    | 0          | 300   |
| Other  | 100  | 100      | 150   | 50        | 50     | 150        | 200   |
| Total  | 500  | 500      | 650   | 350       | 800    | 200        | 1000  |

<https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/>

# Naïve Bayes example

---

- Objective: predict if a given fruit is Banana, Orange, or Other when only three features are known (long, sweet and yellow)
- Step 1: compute Prior probabilities for each class of fruits:
  - $P(Y=\text{Banana}) = 500 / 1000 = 0.50$
  - $P(Y=\text{Orange}) = 300 / 1000 = 0.30$
  - $P(Y=\text{Other}) = 200 / 1000 = 0.20$

| Type   | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|--------|------|----------|-------|-----------|--------|------------|-------|
| Banana | 400  | 100      | 350   | 150       | 450    | 50         | 500   |
| Orange | 0    | 300      | 150   | 150       | 300    | 0          | 300   |
| Other  | 100  | 100      | 150   | 50        | 50     | 150        | 200   |
| Total  | 500  | 500      | 650   | 350       | 800    | 200        | 1000  |

# Naïve Bayes example

---

- Step 2: Compute the probability of evidence that goes in the denominator.
  - $P(x_1=\text{Long}) = 500 / 1000 = 0.50$
  - $P(x_2=\text{Sweet}) = 650 / 1000 = 0.65$
  - $P(x_3=\text{Yellow}) = 800 / 1000 = 0.80$

| Type   | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|--------|------|----------|-------|-----------|--------|------------|-------|
| Banana | 400  | 100      | 350   | 150       | 450    | 50         | 500   |
| Orange | 0    | 300      | 150   | 150       | 300    | 0          | 300   |
| Other  | 100  | 100      | 150   | 50        | 50     | 150        | 200   |
| Total  | 500  | 500      | 650   | 350       | 800    | 200        | 1000  |

# Naïve Bayes example

---

- Step 3: Compute the probability of likelihood of evidences that goes in the numerator.
  - $P(x_1=\text{Long} \mid Y=\text{Banana}) = 400 / 500 = 0.80$
  - $P(x_2=\text{Sweet} \mid Y=\text{Banana}) = 350 / 500 = 0.70$
  - $P(x_3=\text{Yellow} \mid Y=\text{Banana}) = 450 / 500 = 0.90$

The overall probability of Likelihood of evidence for Banana =  $0.8 * 0.7 * 0.9 = 0.504$

| Type   | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|--------|------|----------|-------|-----------|--------|------------|-------|
| Banana | 400  | 100      | 350   | 150       | 450    | 50         | 500   |
| Orange | 0    | 300      | 150   | 150       | 300    | 0          | 300   |
| Other  | 100  | 100      | 150   | 50        | 50     | 150        | 200   |
| Total  | 500  | 500      | 650   | 350       | 800    | 200        | 1000  |

# Naïve Bayes example

---

- Step 4: Substitute all the 3 equations into the Naive Bayes formula, to get the probability that it is a banana.
  - $P(\text{Banana} \mid \text{Long, Sweet and Yellow}) = 0.252 \mid P(\text{Evidence})$ .  $\leftarrow 0.504 * 0.5$
  - $P(\text{Orange} \mid \text{Long, Sweet and Yellow}) = 0 \mid P(\text{Evidence})$
  - $P(\text{Other} \mid \text{Long, Sweet and Yellow}) = 0.01875 \mid P(\text{Evidence})$

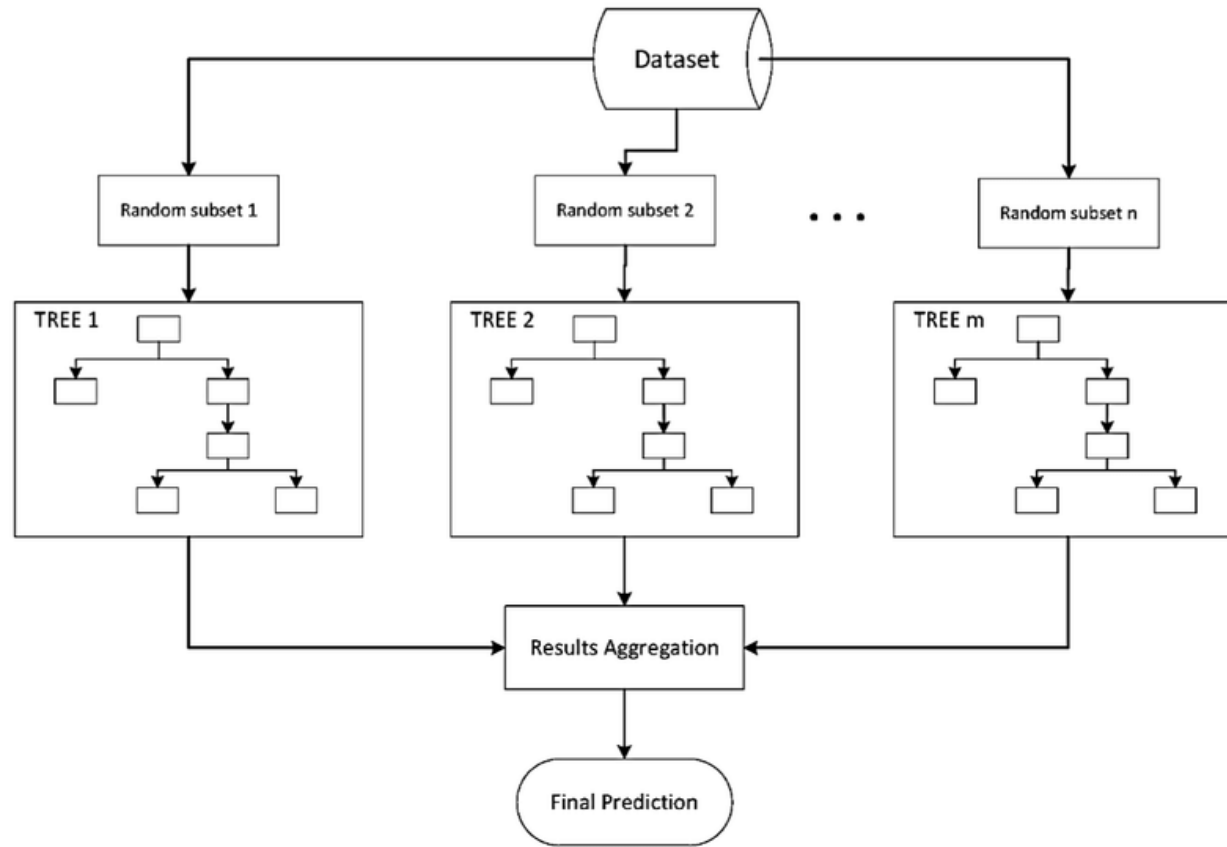
$P(\text{evidence})$  is the same for all, so optional to compute

| Type   | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|--------|------|----------|-------|-----------|--------|------------|-------|
| Banana | 400  | 100      | 350   | 150       | 450    | 50         | 500   |
| Orange | 0    | 300      | 150   | 150       | 300    | 0          | 300   |
| Other  | 100  | 100      | 150   | 50        | 50     | 150        | 200   |
| Total  | 500  | 500      | 650   | 350       | 800    | 200        | 1000  |

# Laplace Correction

---

- The value of  $P(\text{Orange} \mid \text{Long, Sweet and Yellow})$  was zero in the above example, because,  $P(\text{Long} \mid \text{Orange})$  was zero.
- When you have a model with many features, the entire probability will become zero because one of the feature's value was zero.
- To avoid this, we increase the count of the variable with zero to a small value (usually 1) in the numerator, so that the overall probability doesn't become zero.



# Ensemble methods



# Ensemble Methods

---

- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers

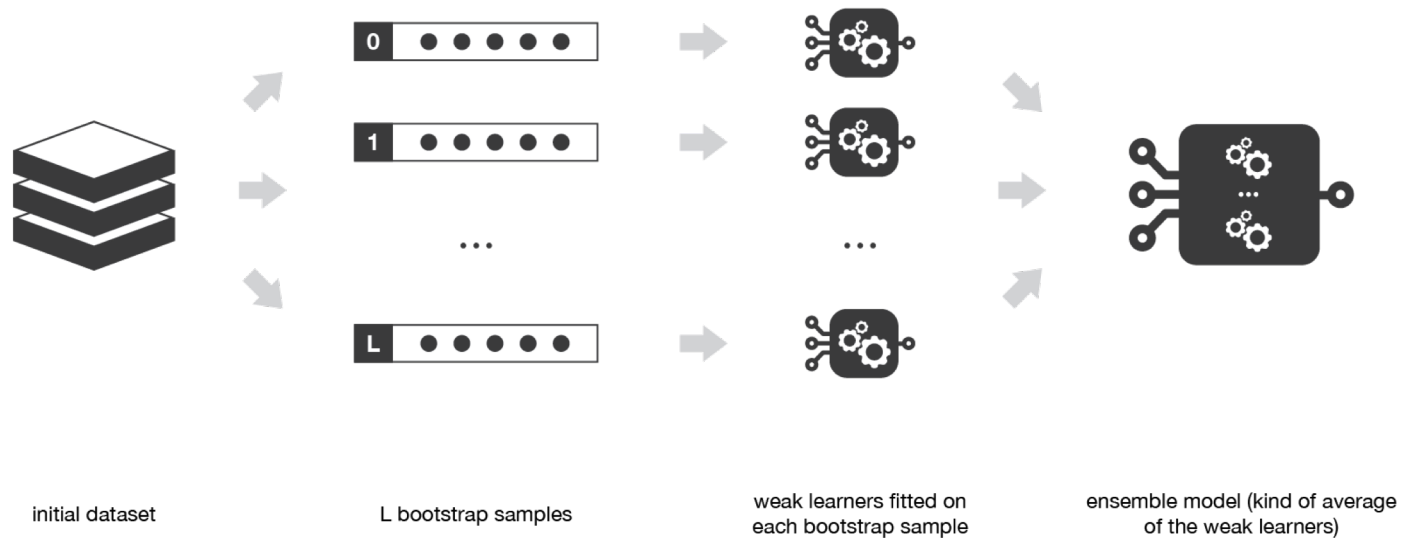
# Types of ensemble methods

---

- **Bagging:** considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process
- **Boosting:** considers homogeneous weak learners, learns them *sequentially* in a very adaptive way (a base model depends on the previous ones) and combines them following a deterministic strategy
- **Stacking:** considers *heterogeneous* weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions

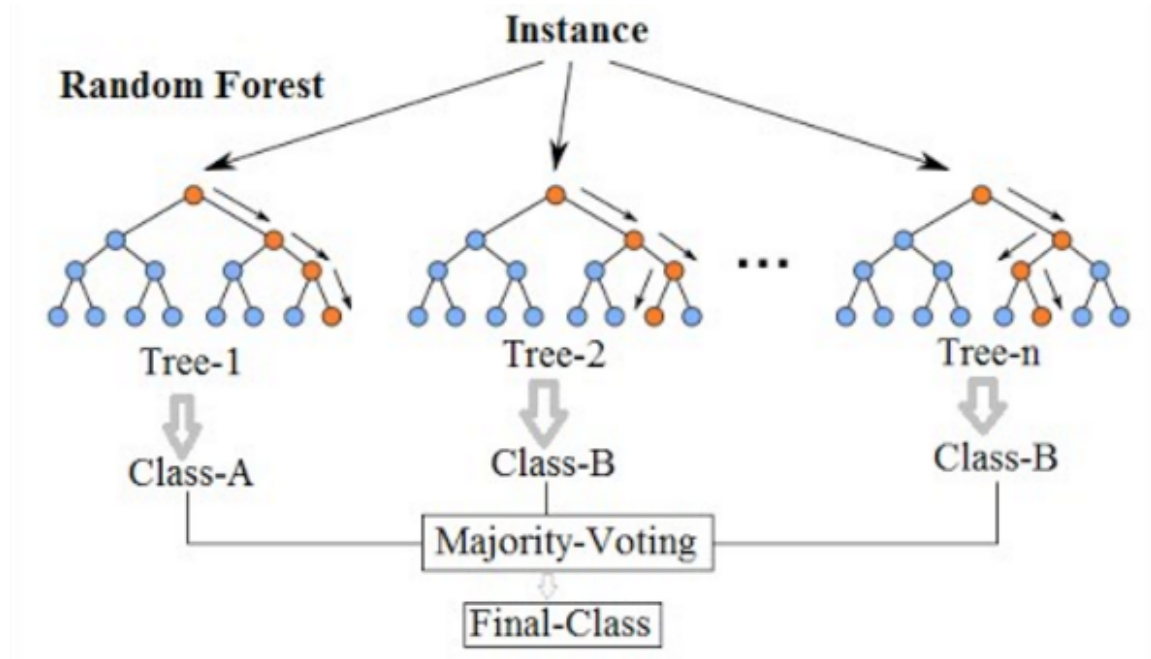
# Bagging

---



# Random Forest

- Special *bagging* case
- An ensemble approach that trains and combines the results of multiple decision trees
- Sample over features



[https://commons.wikimedia.org/wiki/File:Random\\_forest\\_diagram\\_complete.png](https://commons.wikimedia.org/wiki/File:Random_forest_diagram_complete.png)

# Characteristics of DT and RF

---

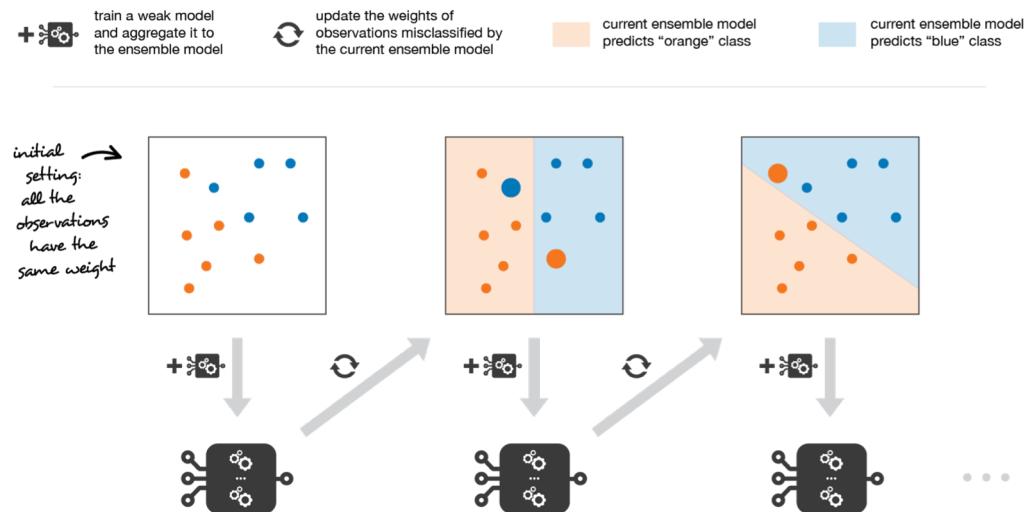
- Decision Trees
  - Inexpensive to construct tree
  - Results are easy to explain/interpret (for small trees!)
  - Prone to overfitting (if no pruning, early stopping)
- Random Forests
  - Multiple decision trees
    - Using different data sample (bagging) and random training features
  - Avoid overfitting
  - Improve accuracy

# Boosting

- Fit models iteratively such that the training of model at a given step depends on the models fitted at the previous steps
  - **Records that are wrongly classified will have their weights increased**
  - Records that are classified correctly will have their weights decreased

- Two popular techniques:

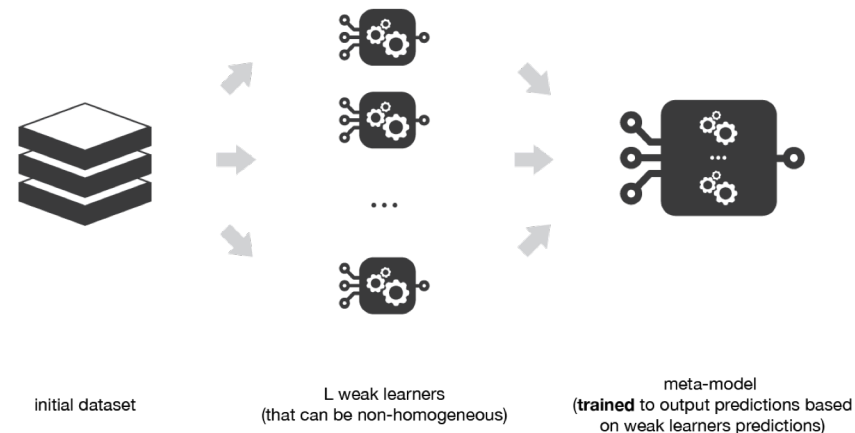
- Gradient boosting
- Adaptive boosting (AdaBoost)  
-> Fig. on right



# Stacking

- learn several different weak learners and combine them by training a meta-model to output predictions based on the multiple predictions returned by these weak models.
- For example, we can choose as weak learners a KNN classifier, a logistic regression and a SVM, and decide to learn a neural network as meta-model.

Then, the neural network will take as inputs the outputs of our three weak learners and will learn to return final predictions based on it.





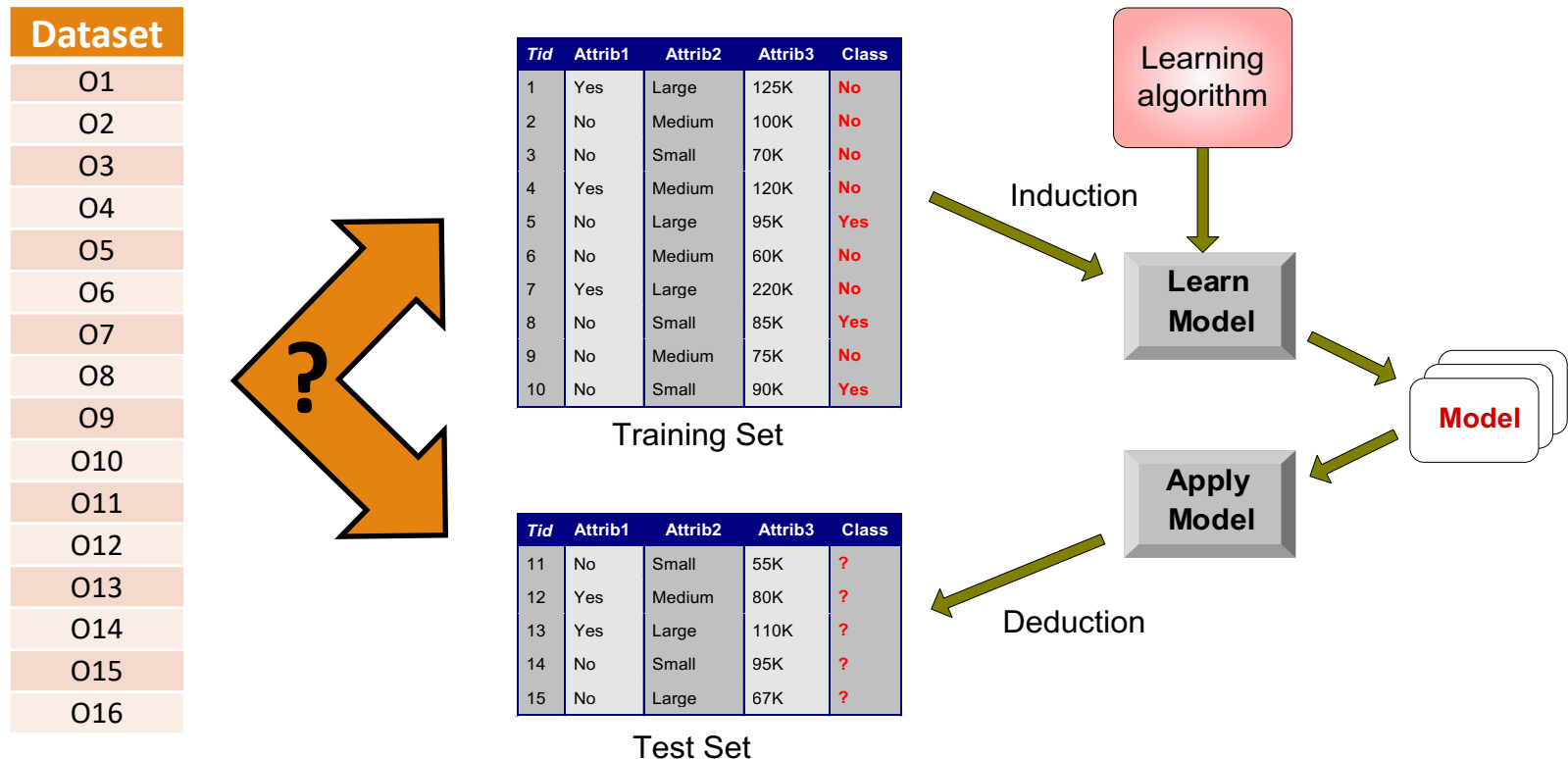
# Evaluation

---



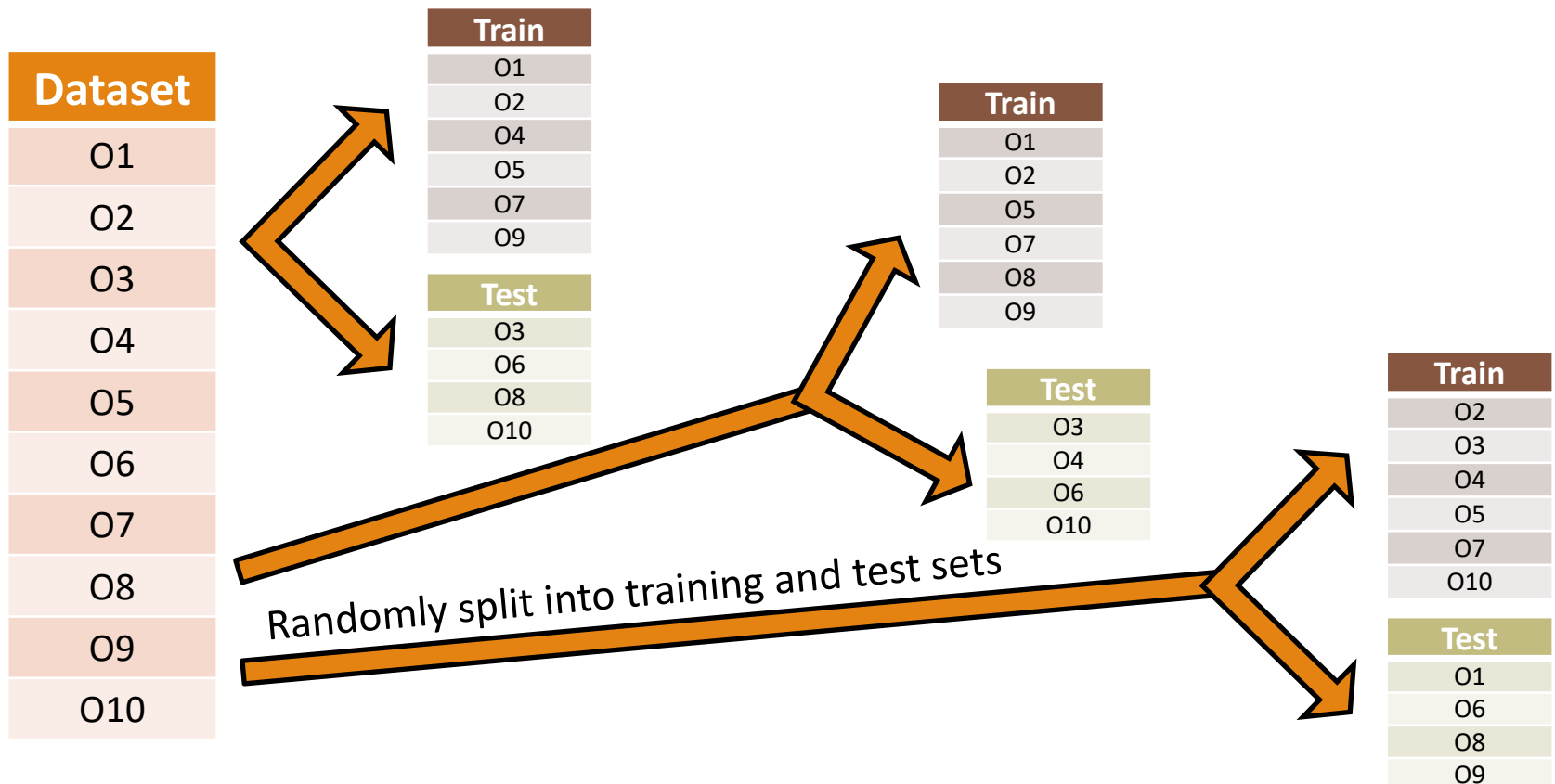
# Training and Test Splits

- How do we determine which observation goes into the training and test sets?



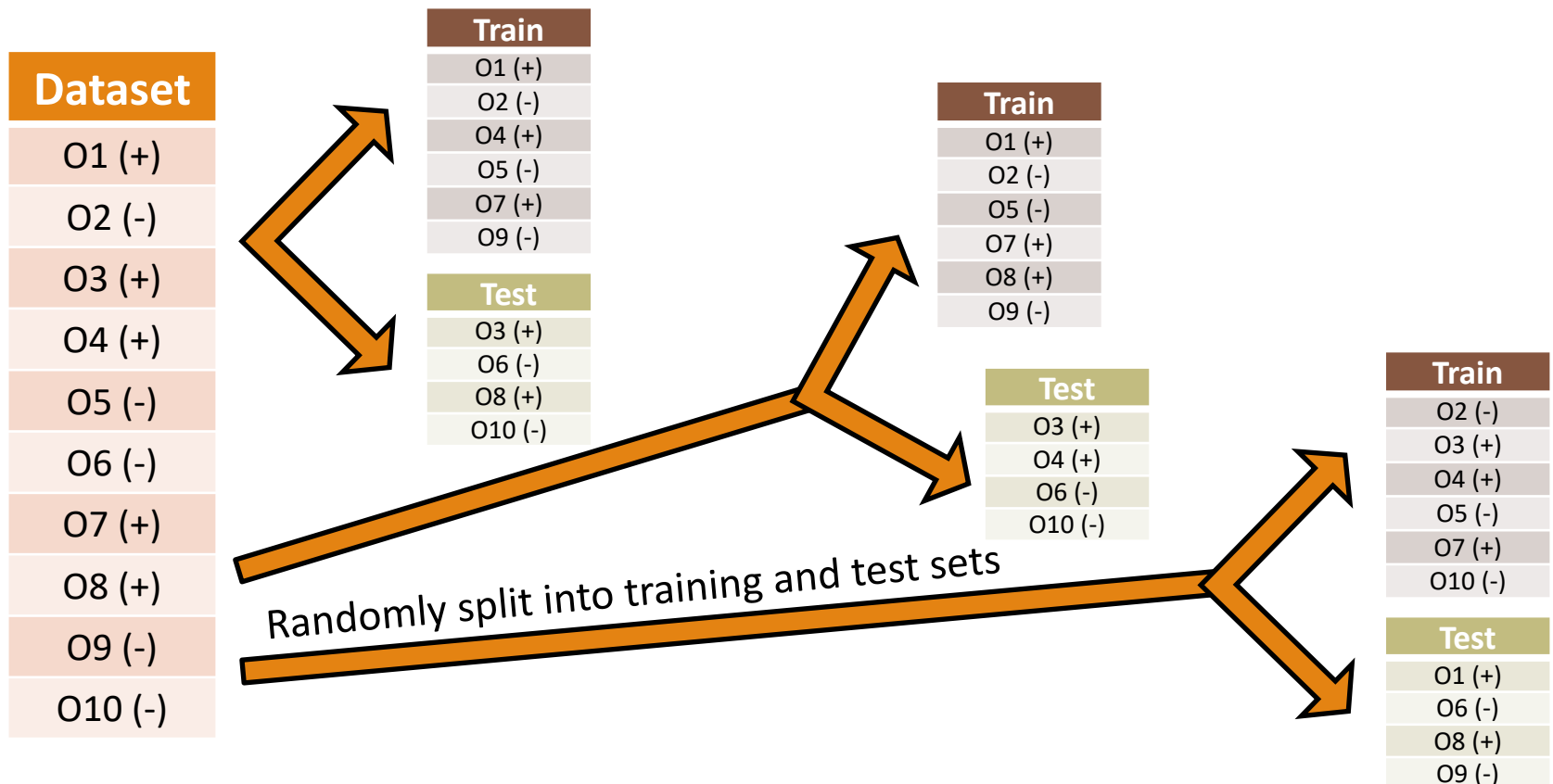
# Random Resampling

- Generate training and test sets by randomly sampling from the dataset



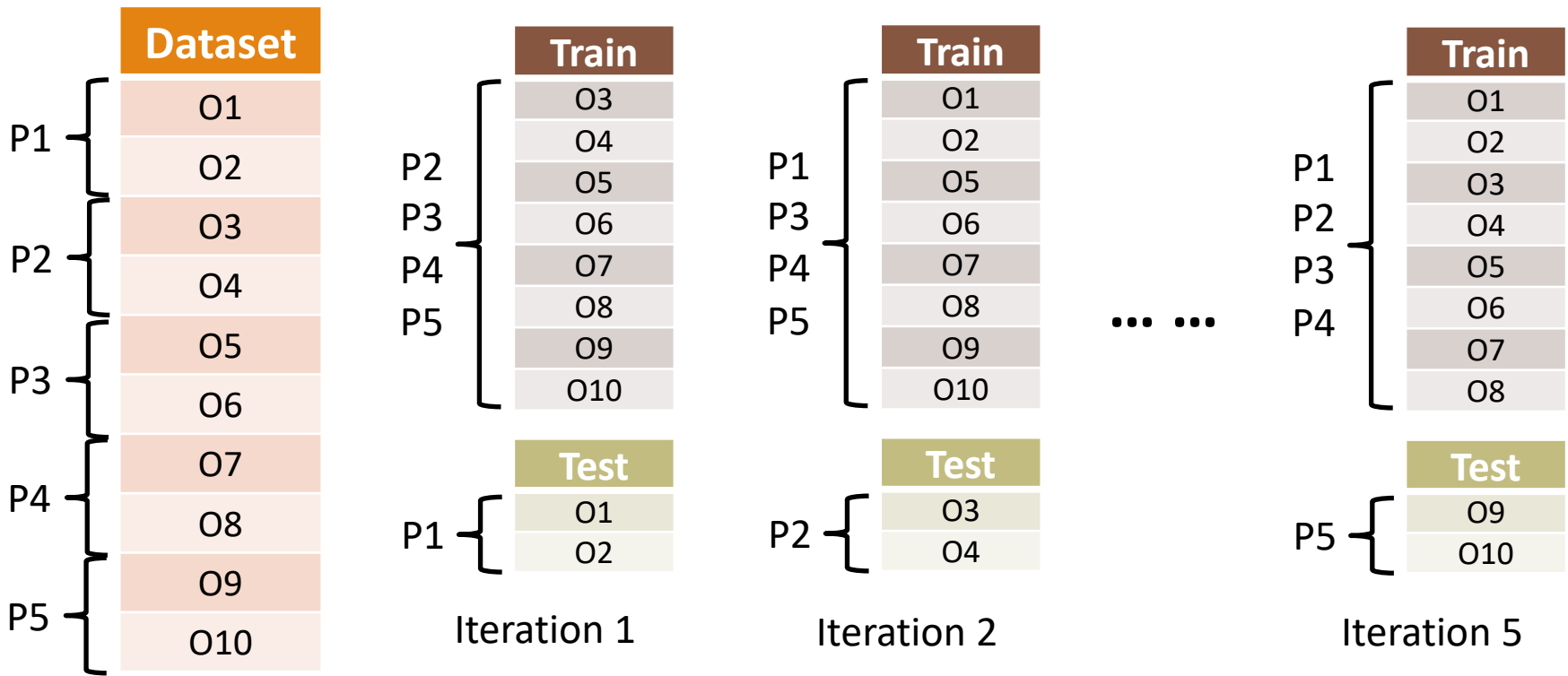
# Stratified Sampling

- Similar to random sampling, but maintain proportion of class labels



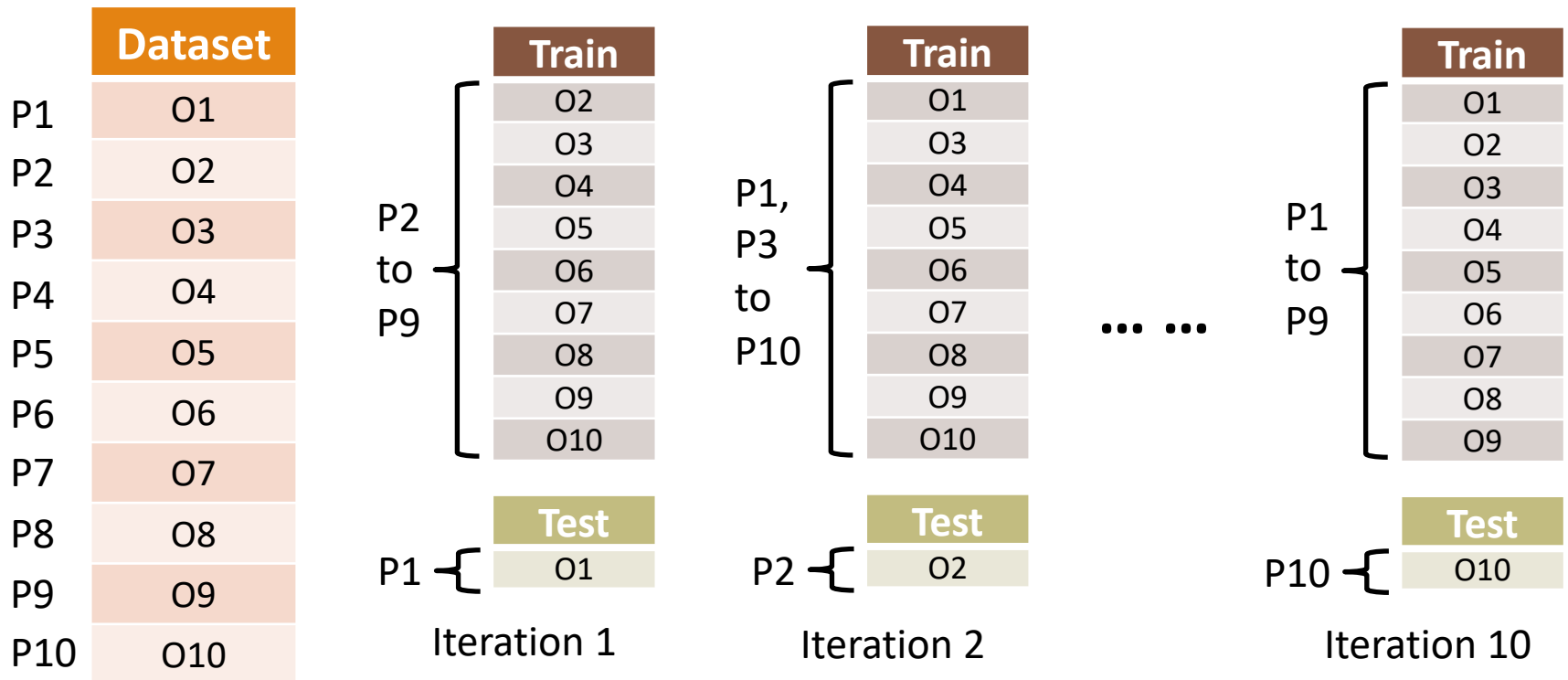
# K-Fold Cross Validation

- Split dataset into  $k$  partitions, iteratively use one partition as test set



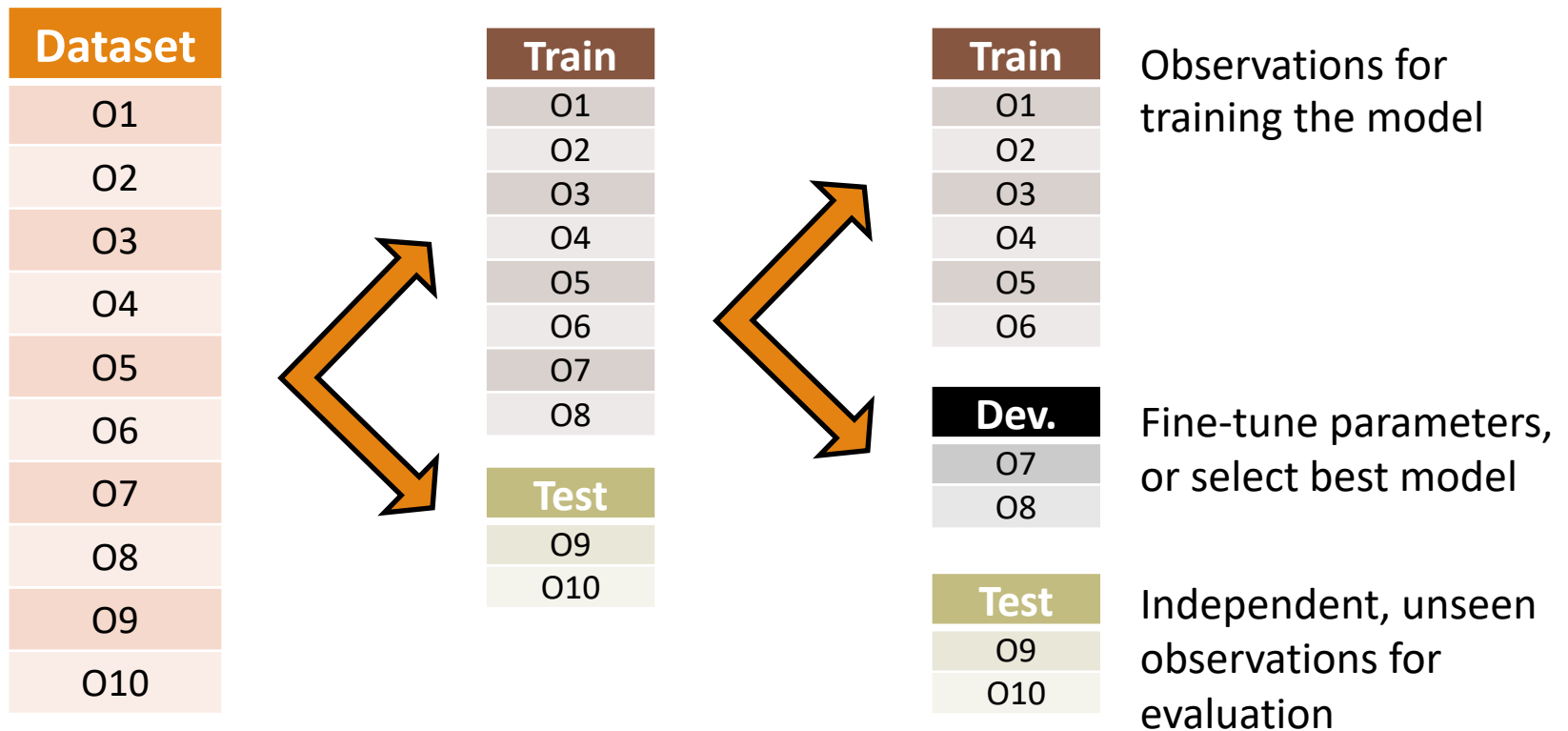
# Leave-one-out Validation

- Special case of  $k$ -fold cross validation, where  $k = \text{no. of observations}$



# Development Set

- Used for tuning hyper-parameters or select a model among many



# Confusion Matrix

---

- A confusion matrix allows us to understand the prediction results
  - E.g., predicting if a person has a cold (Yes) or not (No)
  - E.g., determining if a retrieved result is relevant (Yes) or not (No)

|                            |     | Ground Truth (Relevant) |    |
|----------------------------|-----|-------------------------|----|
|                            |     | Yes                     | No |
| Classification (Retrieved) | Yes | TP                      | FP |
|                            | No  | FN                      | TN |

# Accuracy

---

- Accuracy = % of correct classification, out of all observations  
$$= (TP + TN) / (TP + FP + TN + FN)$$
- Problem: unbalanced classes

|                            |     | Ground Truth (Relevant) |    |
|----------------------------|-----|-------------------------|----|
|                            |     | Yes                     | No |
| Classification (Retrieved) | Yes | TP                      | FP |
|                            | No  | FN                      | TN |

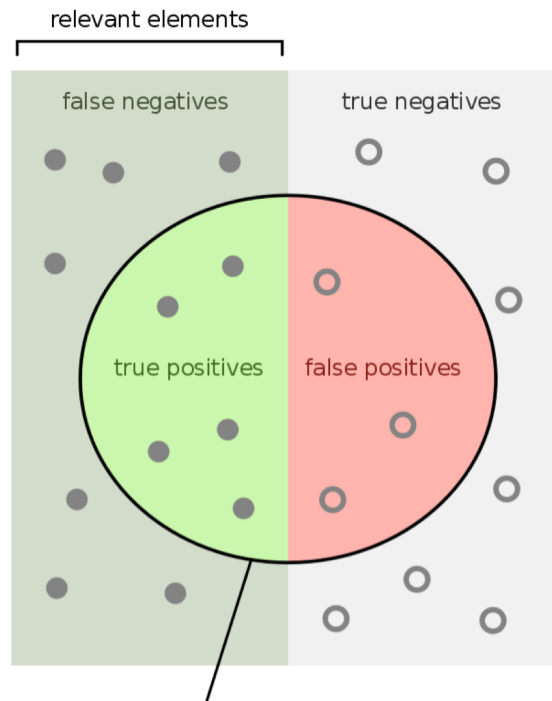


# Precision and Recall

---

- Precision = % of classified (retrieved) items that are correct (relevant)  
$$= TP / (TP + FP)$$
- Recall = % of correct (relevant) items that are classified (retrieved) (aka specificity)  
$$= TP / (TP + FN)$$

|                            |     | Ground Truth (Relevant) |    |
|----------------------------|-----|-------------------------|----|
|                            |     | Yes                     | No |
| Classification (Retrieved) | Yes | TP                      | FP |
|                            | No  | FN                      | TN |



selected elements

How many relevant items are selected?  
e.g. How many sick people are correctly identified as having the condition.

Sensitivity =



How many negative selected elements are truly negative?  
e.g. How many healthy people are identified as not having the condition.

Specificity =



# F1-score

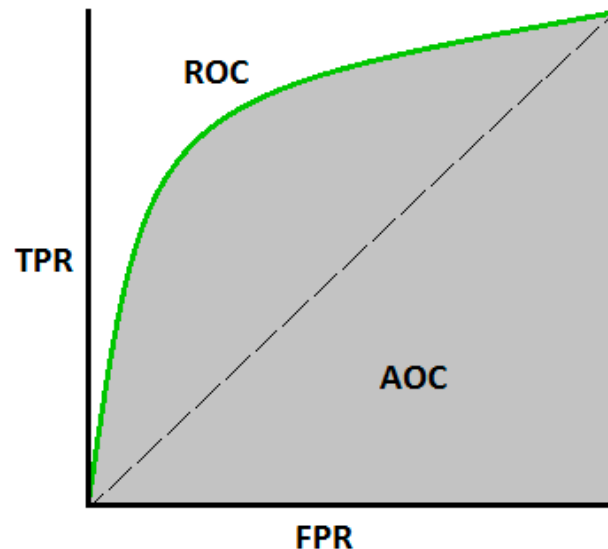
---

- Problems with Precision and Recall
  - Trade-off between precision and recall
  - To get 100% recall, just retrieve everything
- F1 score =  $\frac{2 \times \textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}}$ 
  - i.e., Taking the harmonic mean of Precision and Recall

# Receiver Operating Curve - ROC

---

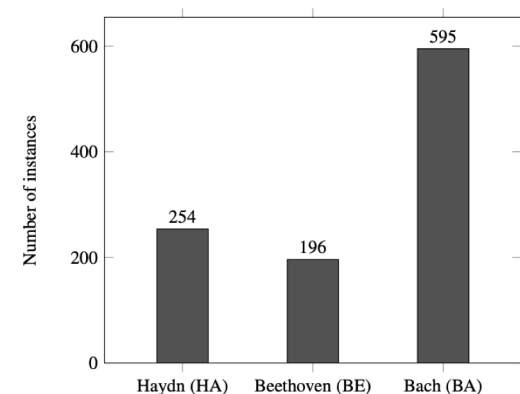
- Random classification = diagonal line
- AOC or AUC: area under the curve: 0.5 for random, higher is better.



# Classifying MIDI files per composer

**Table 2** Performance of the models with 10-fold cross-validation

| Method                                                      | Accuracy   | AUC        |
|-------------------------------------------------------------|------------|------------|
| C4.5 Decision tree                                          | 80%        | 79%        |
| RIPPER ruleset                                              | 81%        | 85%        |
| Logistic regression                                         | 83%        | 92%        |
| Naive Bayes                                                 | 80%        | 90%        |
| Support vector machines                                     | <b>86%</b> | <b>93%</b> |
| $p < 0.01$ : italic, $p > 0.05$ : bold, best: <u>bold</u> . |            |            |



Class distribution

**Table 9** Confusion matrix for support vector machines

| a          | b          | c          | classified as |
|------------|------------|------------|---------------|
| <b>204</b> | 26         | 24         | a = HA        |
| 49         | <b>127</b> | 20         | b = BE        |
| 22         | 10         | <b>563</b> | c = BA        |

[Book chapter: Herremans D., Martens D, Sørensen K., Meredith D.. 2015. Composer Classification Models for Music-Theory Building. Computational Music Analysis. Springer](#)