Machine Learning

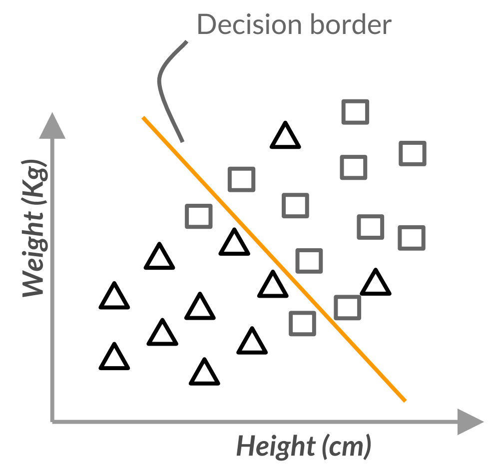
Session 6. Supervised Learning

- **Introduction**
- **Decision Trees**
- **Random Forests**
- **Ensemble methods: bagging, boosting and stacking**

Chap 14 of C. Bishop book

Introduction: Classification

- A classifier system pretends **to infer** from examples (training data) a function that relates a set of features/variables **to a label or class**.
- The goal is to predict the label or class for other observations of the features/variables.
- The class or label could be binary or a range of values. Examples:
  - Predict the character from an image
  - Is the person a woman or man?
  - Is this person likely to be left or right ideology?
  - Is this person likely to buy a product?
  - …

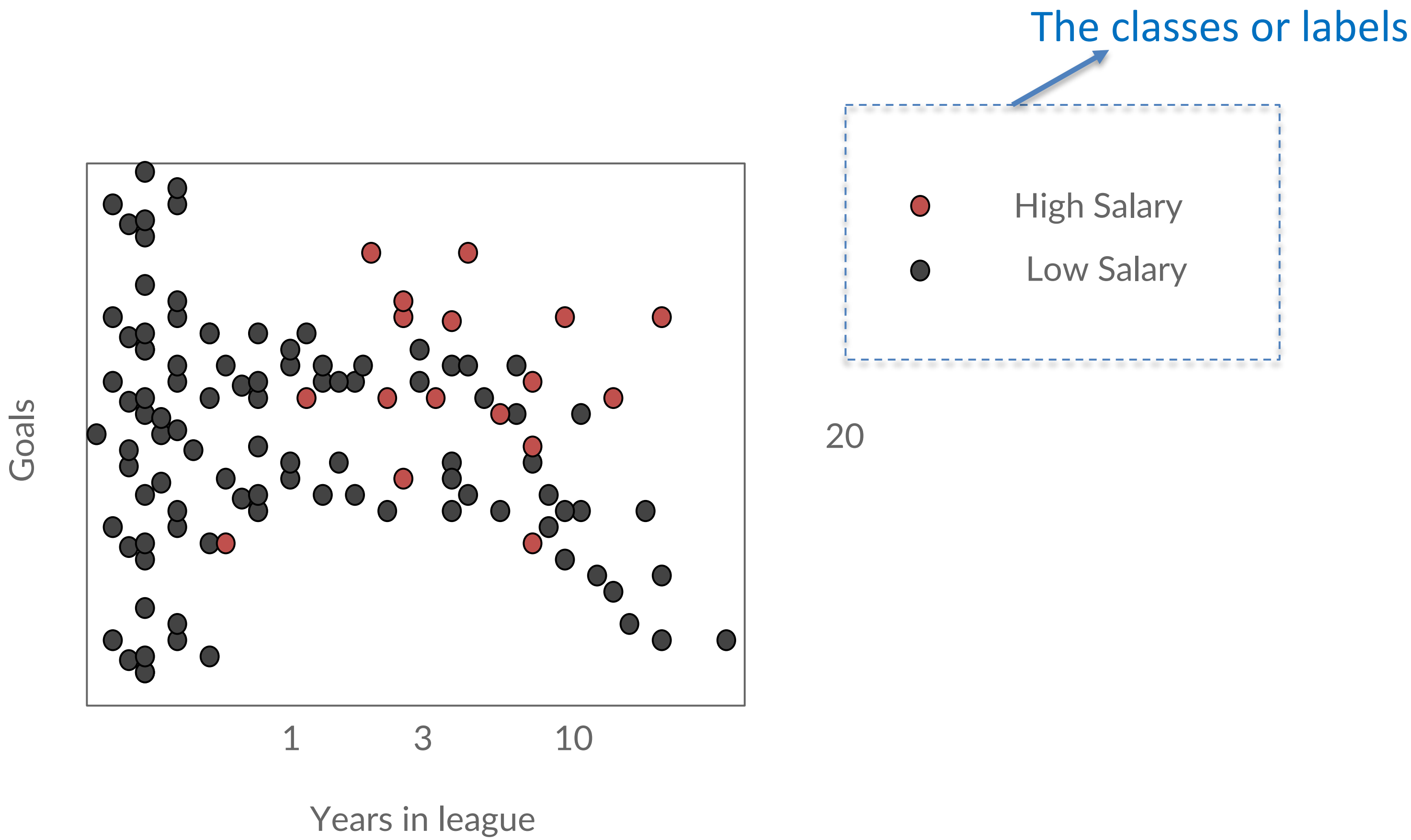


- Given a dataset composed of:
  - N observations $\{\mathbf{x}_n, t_n\}$, where $n = 1, \dots, N$
  - $\mathbf{x}_n$: input vector (in general D dimensions), t_n: target value
- Goal: predict the value of t for **a new** value of $\mathbf{x}$

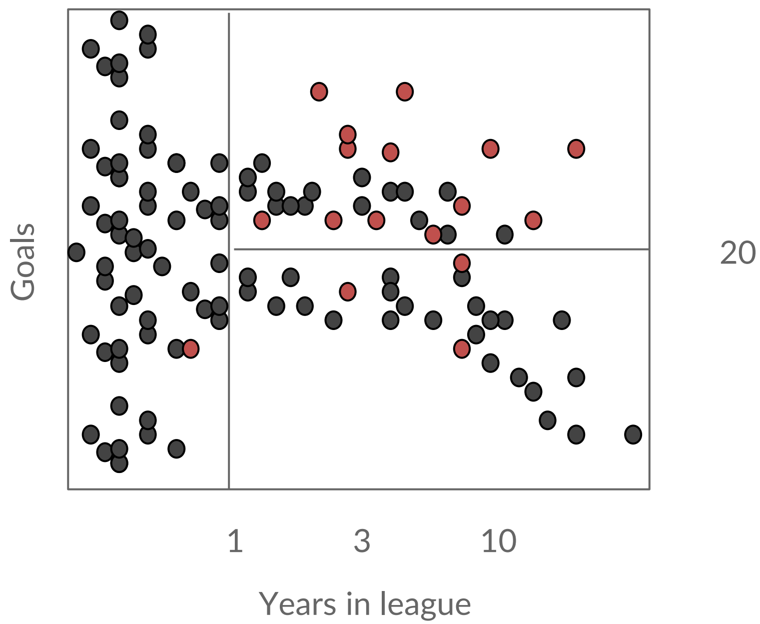
Introduction: Decision Trees

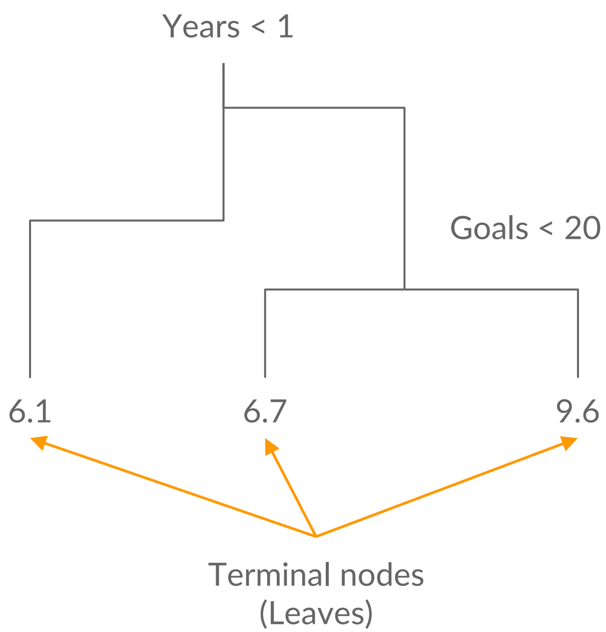
- Decision trees (DTs) are a powerful prediction method and very popular because:
  - Explainability/Interpretability: a prediction comes with a meaningful sequence of decisions
  - They have high accuracy and stability
- DTs also provide the foundation for more advanced ensemble methods such as:
    - **Bagging**
    - **Random forests**
    - **Boosting**
- DTs and variants can be applied for **regression** and **binary** and **multiclass** classification

Example: Divide the 2D space to classify football players in High and Low salary



Example: Divide the 2D space to classify football players in High and Low salary

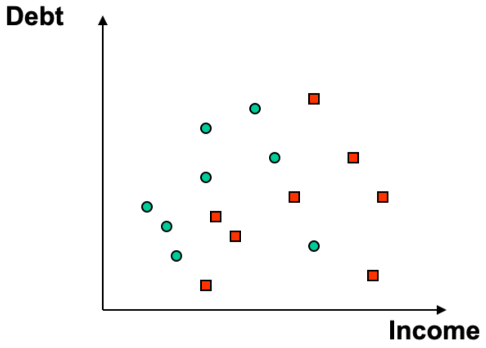




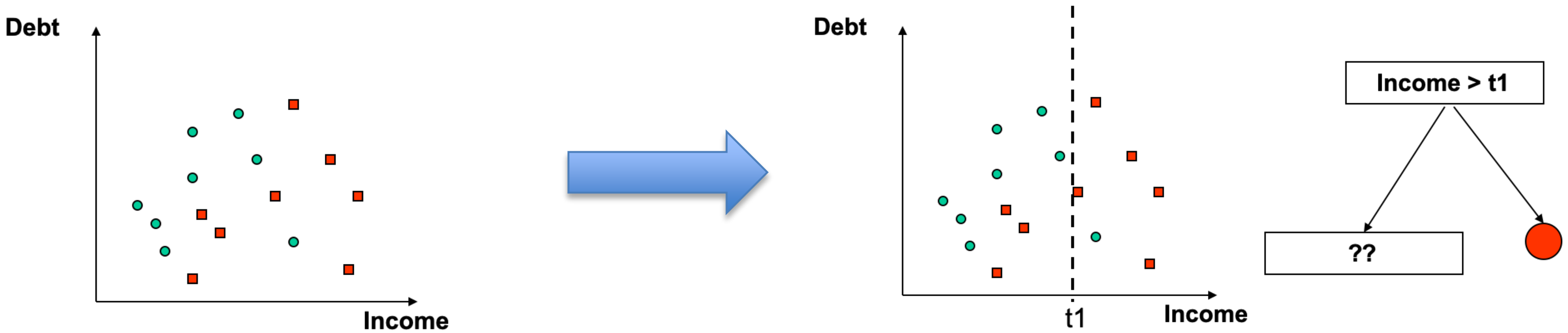
Classification and Regression Trees (CART)

- Unlike simpler (linear) models, they map **non-linear** relationships quite well
- It works for both **categorical** and **continuous** input and output variables
- DTs split the nodes in all variables or features of the dataset
- The split decision looks for sub-nodes more homogeneous
- There are several methodologies to decide the best split:
  - Gini
  - Information Gain
  - Chi-square

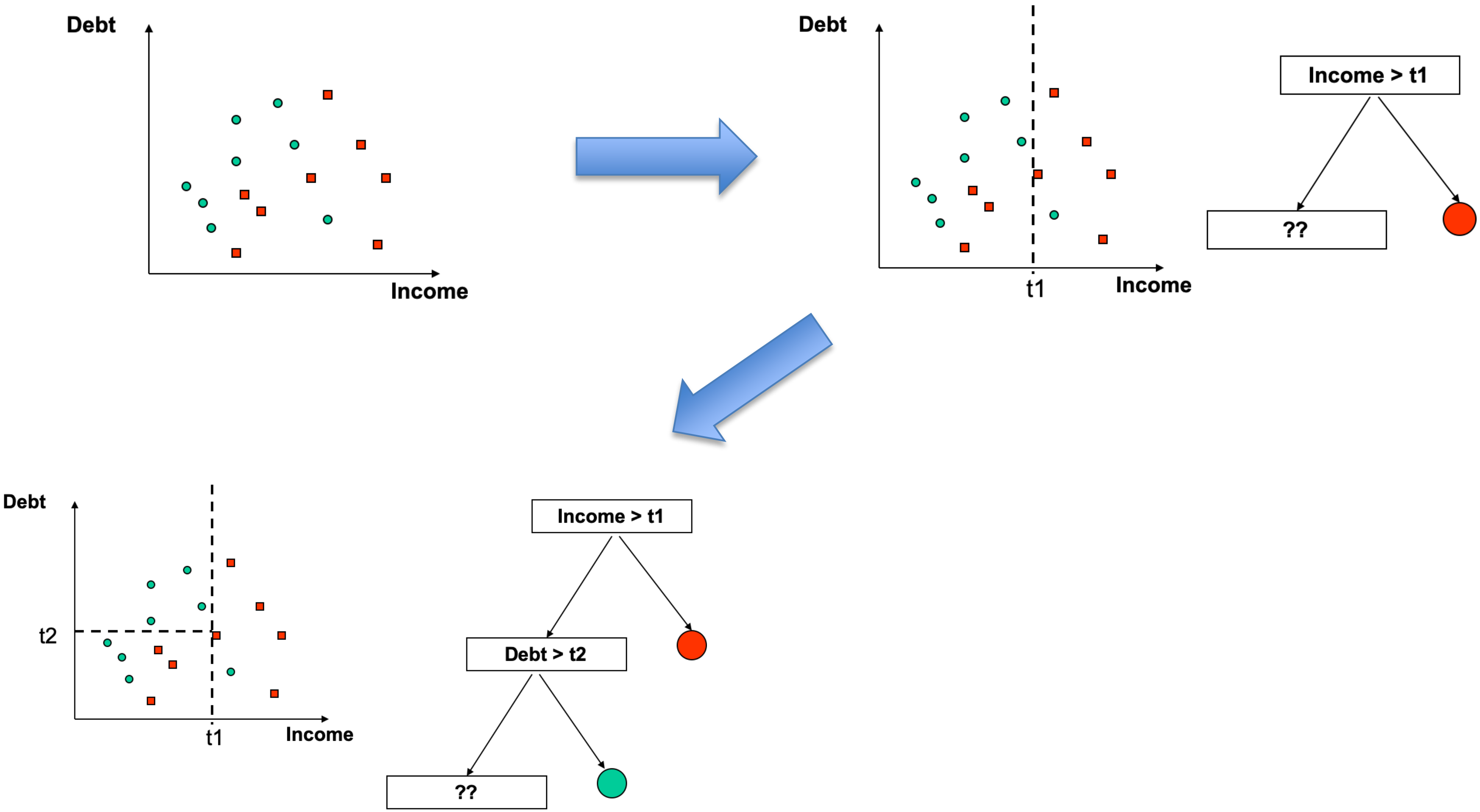
Classification and Regression Trees (CART)



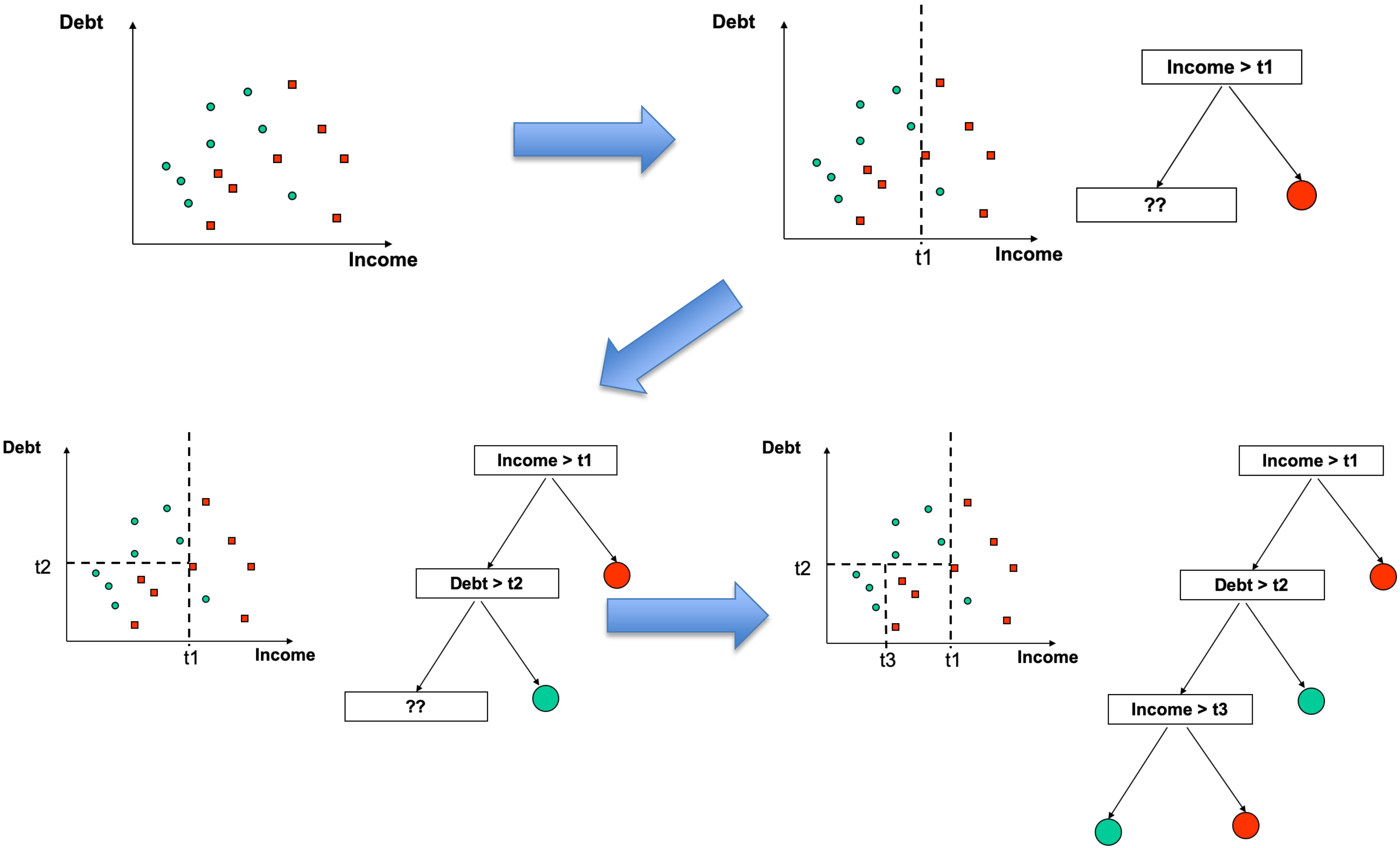
Classification and Regression Trees (CART)



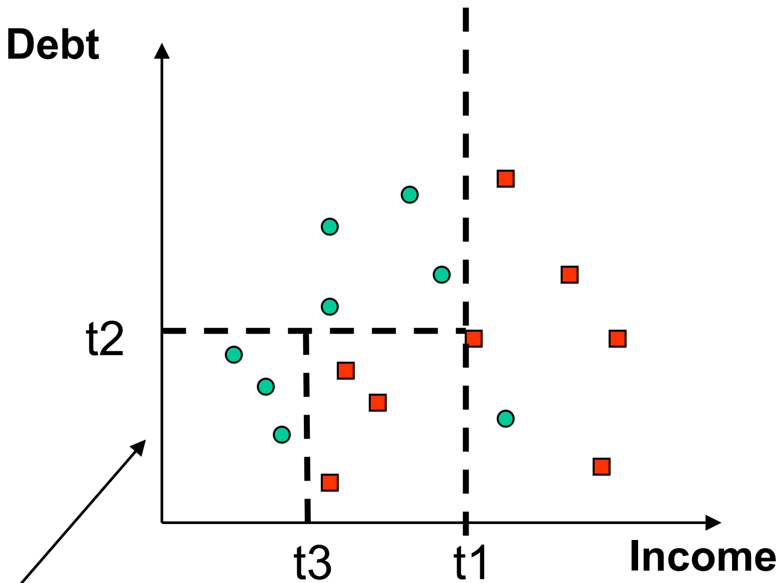
Classification and Regression Trees (CART)

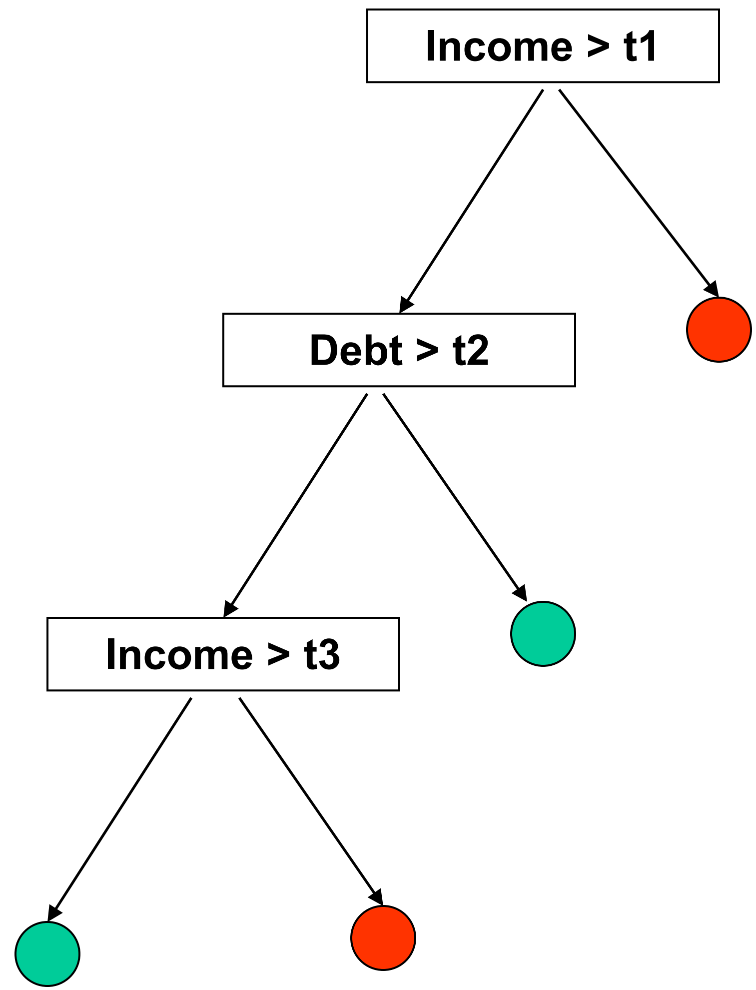


Classification and Regression Trees (CART)



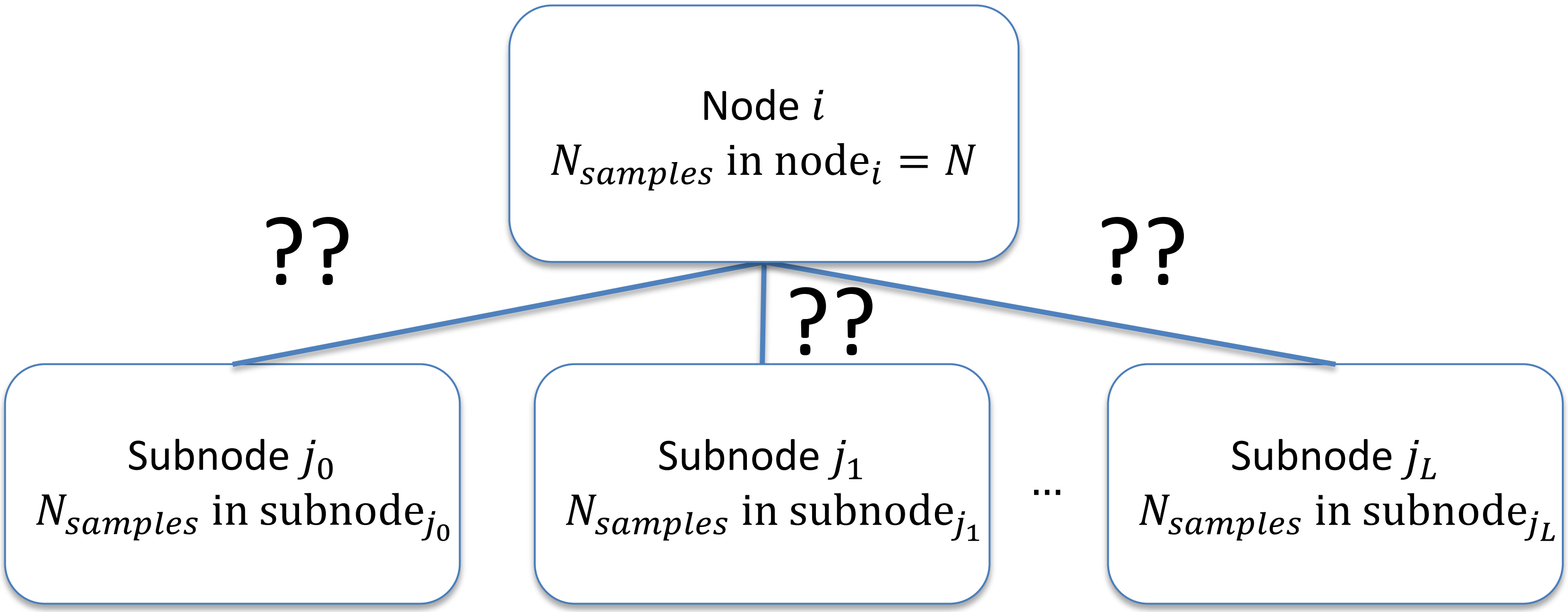
Classification and Regression Trees (CART)





Note: tree boundaries are linear and axis-parallel

How to select the best splitting feature?



How to select the best splitting feature?

- **Based on Gini Impurity**
  - **Step 1:** Calculate the **Gini** of each subnode **:**

$$\text{Gini Impurity}_{\text{subnode } j} = 1 - \text{Gini}_{\text{subnode } j}$$

$$\text{Gini}_{\text{subnode } j} = \sum_{l=1}^{K} p_{jl}^2$$

Where $p_{jl} = \frac{N_{samples} \text{ in subnode}_j \text{ of class } l}{N_{samples} \text{in subnode}_j}$

$K = \#\text{classes}$

How to select the best splitting feature?

- **Based on Gini Impurity**
  - **Step 1:** Calculate the **Gini** of each subnode :

$$\text{Gini Impurity}_{\text{subnode } j} = 1 - \text{Gini}_{\text{subnode } j}$$

$$\text{Gini}_{\text{subnode } j} = \sum_{l=1}^{K} p_{jl}^2$$

Where $p_{jl} = \frac{N_{samples} \text{ in subnode}_j \text{ of class } l}{N_{samples} \text{in subnode}_j}$

$$K = \#\text{classes}$$

  - **Step 2:** Calculate the **Weighted Gini impurity** of the split as:

$$\text{Weighted Gini Impurity Split} = \sum_{j=1}^{L} W_j \cdot \text{Gini}_{\text{subnode } j}$$

$$W_j = \frac{N_{samples} \text{in subnode}_j}{N_{samples} \text{in node}_i}$$

$$L = \text{Number of subnodes of split}$$

How to select the best splitting feature?

- **Based on Gini Impurity**
  - **Step 1:** Calculate the **Gini** of each subnode :

$$\text{Gini Impurity}_{\text{subnode } j} = 1 - \text{Gini}_{\text{subnode } j}$$

$$\text{Gini}_{\text{subnode } j} = \sum_{l=1}^{K} p_{jl}^2$$

Where $p_{jl} = \dfrac{N_{samples} \text{ in subnode}_j \text{ of class } l}{N_{samples} \text{in subnode}_j}$

$$K = \#\text{classes}$$

  - **Step 2:** Calculate the **Weighted Gini impurity** of the split as:

$$\text{Weighted Gini Impurity Split} = \sum_{j=1}^{L} W_j \cdot \text{Gini}_{\text{subnode } j}$$

$$W_j = \frac{N_{samples} \text{in subnode}_j}{N_{samples} \text{in node}_i}$$

$$L = \text{Number of subnodes of split}$$

  - **Step 3:** Select the Split with lower **Weighted Gini impurity**

- The lower the Gini impurity, the higher the homogeneity

How to select the best splitting feature?

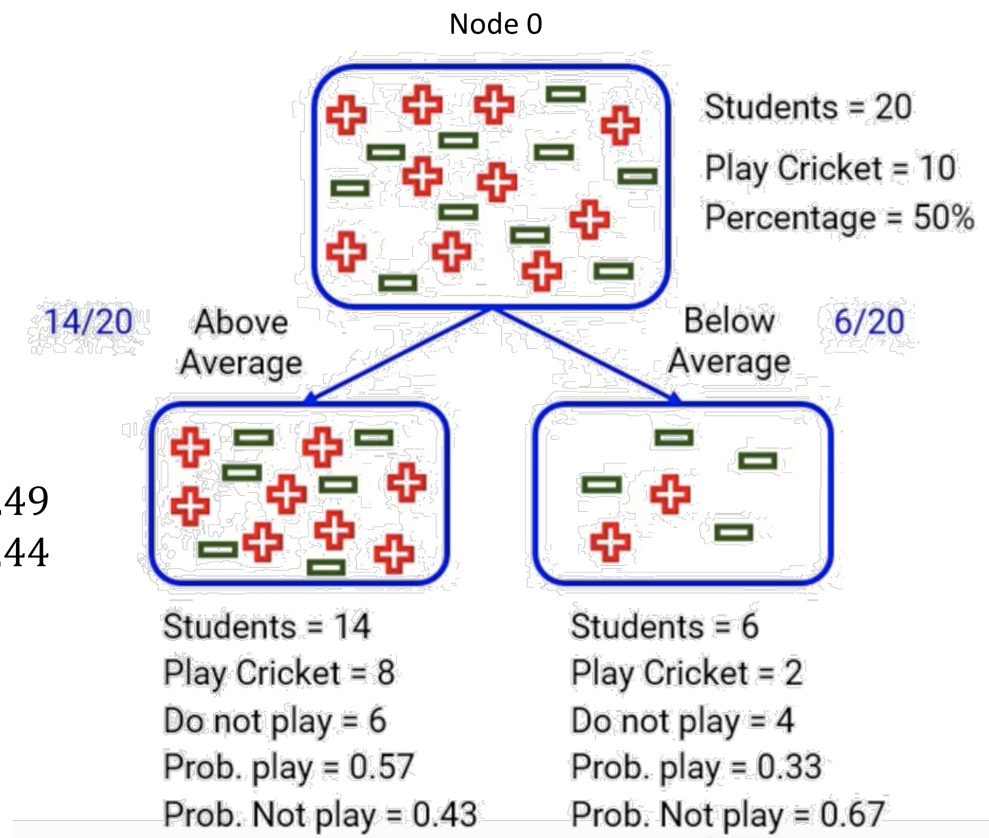
- **Based on Gini Impurity**
- Example

  Two classes:

  $t = \{\text{play cricket}, \text{not play cricket}\}$

Feature 1 : Academic Performance

$\text{Academic Performance} \in \{\text{above avg}, \text{below avg}\}$



$$\text{Gini Impurity}_{\text{subnode 1}} = 1 - (0.57^2 + 0.43^2) = 0.49$$
$$\text{Gini Impurity}_{\text{subnode 2}} = 1 - (0.33^2 + 0.67^2) = 0.44$$

$$\textbf{Weighted Gini Impurity}_{\textbf{split}} = \left(\frac{\mathbf{14}}{\mathbf{20}}\right) \cdot \mathbf{0.49} + \left(\frac{\mathbf{6}}{\mathbf{20}}\right) \cdot \mathbf{0.44} = \mathbf{0.475}$$

How to select the best splitting feature?

- **Based on Gini Impurity**
- Example

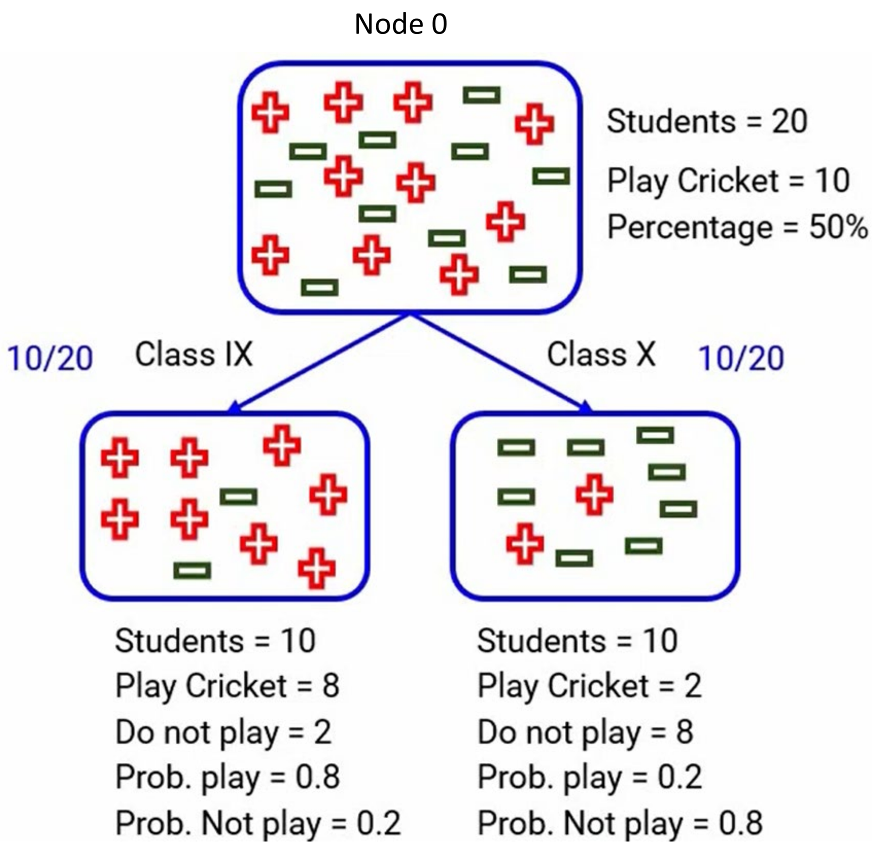
  Two classes:

  $t = \{\text{play cricket}, \text{not play cricket}\}$

Feature 2 : Class

Academic Performance $\in \{\text{class IX}, \text{class X}\}$

$$\text{Gini Impurity}_{\text{subnode 1}} = 1 - (0.8^2 + 0.2^2) = 0.32$$
$$\text{Gini Impurity}_{\text{subnode 2}} = 1 - (0.2^2 + 0.8^2) = 0.32$$



$$\textbf{Weighted Gini Impurity}_{\textbf{split}} = \left(\frac{\mathbf{10}}{\mathbf{20}}\right) \cdot \mathbf{0.32} + \left(\frac{\mathbf{10}}{\mathbf{20}}\right) \cdot \mathbf{0.32} = \mathbf{0.32}$$

How to select the best splitting feature?

- **Based on Gini Impurity**
- Example
  Two classes:

  $t = \{\text{play cricket}, \text{not play cricket}\}$

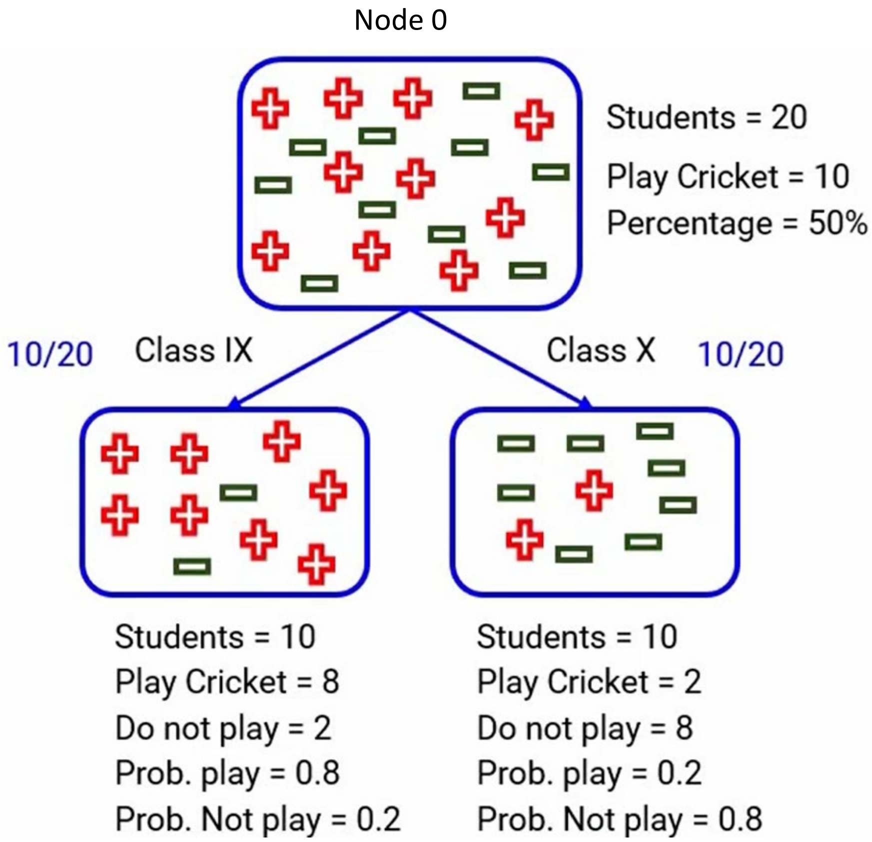
Feature 2 : Class

Academic Performance $\in \{\text{class IX}, \text{class X}\}$

$$\text{Gini Impurity}_{\text{subnode 1}} = 1 - (0.8^2 + 0.2^2) = 0.32$$
$$\text{Gini Impurity}_{\text{subnode 2}} = 1 - (0.2^2 + 0.8^2) = 0.32$$

$$\mathbf{0.32 < 0.475}$$

Feature 2 Class would be selected



$$\textbf{Weighted Gini Impurity}_{\textbf{split}} = \left(\frac{\mathbf{10}}{\mathbf{20}}\right) \cdot \mathbf{0.32} + \left(\frac{\mathbf{10}}{\mathbf{20}}\right) \cdot \mathbf{0.32} = \mathbf{0.32}$$

How to select the best splitting feature?

- **Based on Information Gain**
  - **Step 1: Calculate the Entropy and Information Gain of each node and subnode:**

$$\text{Entropy}_{subnode_j} = -\sum_{l=1}^{K} p_{jl} \cdot \log_2 p_{jl}$$

Where $p_{jl} = \frac{N_{samples} \text{ in subnode}_j \text{ of class } l}{N_{samples} \text{in subnode}_j}$

$$K = \#\text{classes}$$

$$\text{Information Gain}_{\text{subnode } j} = 1 - \text{Entropy}_{\text{subnode } j}$$

How to select the best splitting feature?

- **Based on Information Gain**
  - **Step 1: Calculate the Entropy and Information Gain of each node and subnode:**

$$\text{Entropy}_{subnode_j} = -\sum_{l=1}^{K} p_{jl} \cdot \log_2 p_{jl}$$

Where $p_{jl} = \frac{N_{samples} \text{ in subnode}_j \text{ of class } l}{N_{samples} \text{in subnode}_j}$

$$K = \#\text{classes}$$

$$\text{Information Gain}_{\text{subnode } j} = 1 - \text{Entropy}_{\text{subnode } j}$$

  - **Step 2: Calculate the Weighted Entropy of the split as:**

$$\text{Weighted Entropy} = \sum_{j=1}^{L} W_j \cdot \text{Entropy}_{\text{subnode}_\text{j}}$$

$$W_j = \frac{N_{samples} \text{in subnode}_j}{N_{samples} \text{in node}_i}$$

L= $Number\ of\ subnodes\ of\ split$

How to select the best splitting feature?

- **Based on Information Gain**
  - **Step 1: Calculate the Entropy and Information Gain of each node and subnode:**

$$\text{Entropy}_{subnode_j} = -\sum_{l=1}^{K} p_{jl} \cdot \log_2 p_{jl}$$

Where $p_{jl} = \frac{N_{samples} \text{ in subnode}_j \text{ of class } l}{N_{samples} \text{in subnode}_j}$

$$K = \#\text{classes}$$

$$\text{Information Gain}_{\text{subnode } j} = 1 - \text{Entropy}_{\text{subnode } j}$$

  - **Step 2: Calculate the Weighted Entropy of the split as:**

$$\text{Weighted Entropy} = \sum_{j=1}^{L} W_j \cdot \text{Entropy}_{\text{subnode}_j}$$

$$W_j = \frac{N_{samples} \text{in subnode}_j}{N_{samples} \text{in node}_i}$$

$$L = \text{Number of subnodes of split}$$

  - **Step 3: Select the Split with lower Weighted Entropy**
- The lower the entropy, the higher the homogeneity in the node

How to select the best splitting feature?

- **Based on Information Gain**:
- Example:

  Two classes:

  $t = \{\text{play cricket, not play cricket}\}$

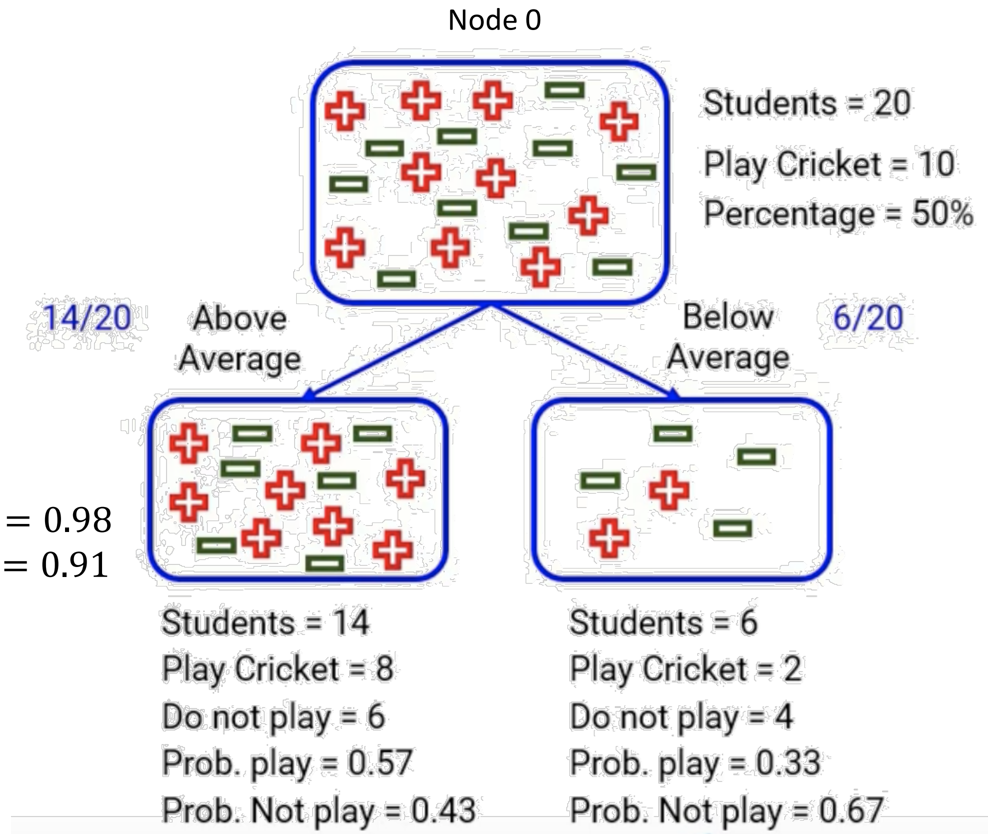
Feature 1 : Academic Performance

Academic Performance $\in \{\text{above avg, below avg}\}$

Entropy_$node_0 = -0.5 * \log_2(0.5) - 0.5 * \log_2(0.5) = 1$

Entropy_sub$node_1 = -0.57 * \log_2 0.57 - 0.43 * \log_2 0.43 = 0.98$

Entropy_sub$node_2 = -0.33 * \log_2 0.33 - 0.67 * \log_2 0.67 = 0.91$



$$\textbf{Weighted Entropy}_{\textbf{split}} = \left(\frac{\mathbf{14}}{\mathbf{20}}\right) * \mathbf{0.98} + \left(\frac{\mathbf{6}}{\mathbf{20}}\right) * \mathbf{0.91} = \mathbf{0.959}$$

$$\textbf{Information Gain}_{split} = \mathbf{1} - \textbf{Weighted Entropy}_{split} = \mathbf{1} - \mathbf{0.959} = \mathbf{0.041}$$

Main parameters of CART algorithms

- **Minimum samples for a node Split**
  - Defines the mínimum number of simples (observations) required for a node to be considered for splitting
  - Controls over-fitting. Higher values prevent a model from learning too specific relations to the particular sample
  - Too high values can lead to under−fitting hence, it should be tuned using Cross−Validation

Main parameters of CART algorithms

- **Minimum samples for a node Split**
  - Defines the mínimum number of simples (observations) required for a node to be considered for splitting
  - Controls over-fitting. Higher values prevent a model from learning too specific relations to the particular sample
  - Too high values can lead to under−fitting hence, it should be tuned using Cross−Validation

- **Minimum samples for a terminal node (leaf)**
  - Defines the minimum samples (or observations) required in a terminal node or leaf
  - Used to control over−fitting similar to min_samples_split
  - Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small

Main parameters of CART algorithms

- **Maximum depth of tree (vertical depth)**
  - The maximum depth of a tree.
  - Used to control over−fitting as higher depth will allow model to learn relations very specific to a particular sample.
  - Should be tuned using CV.

Main parameters of CART algorithms

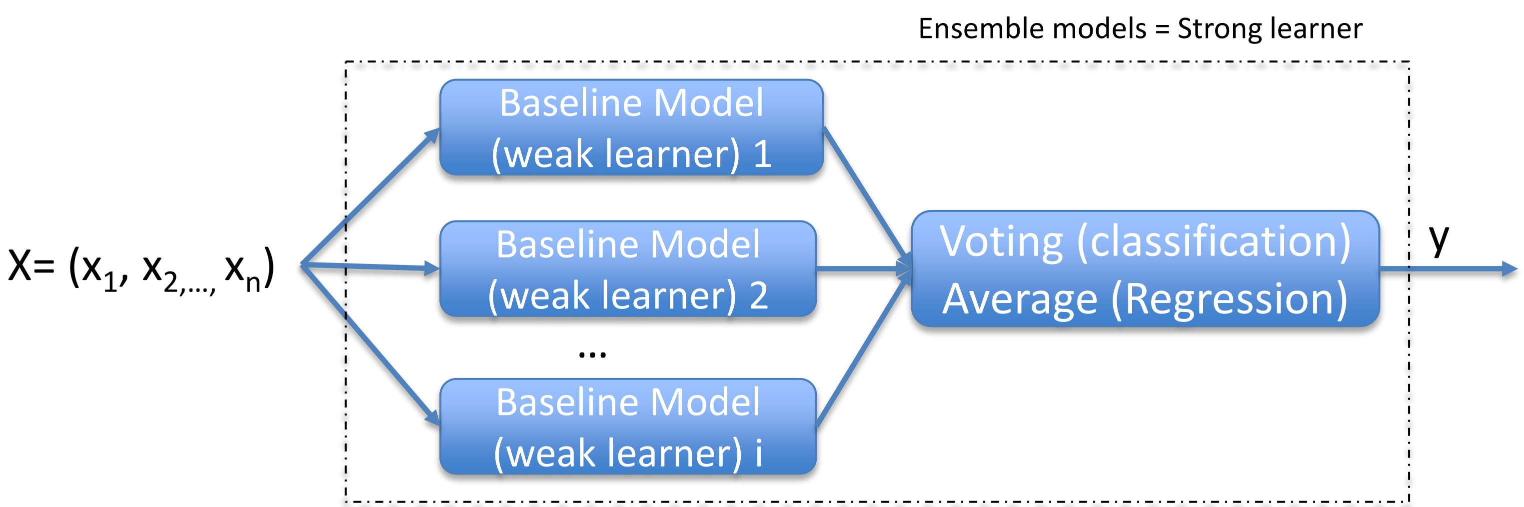
- **Maximum depth of tree (vertical depth)**
  - The maximum depth of a tree.
  - Used to control over−fitting as higher depth will allow model to learn relations very specific to a particular sample.
  - Should be tuned using CV.
- **Maximum number of terminal nodes**
  - The maximum number of terminal nodes or leaves in a tree.
  - Can be defined in place of max_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves.

Main parameters of CART algorithms

- **Maximum depth of tree (vertical depth)**
  - The maximum depth of a tree.
  - Used to control over–fitting as higher depth will allow model to learn relations very specific to a particular sample.
  - Should be tuned using CV.
- **Maximum number of terminal nodes**
  - The maximum number of terminal nodes or leaves in a tree.
  - Can be defined in place of max_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of leaves.
- **Maximum features to consider for split**
  - The number of features to consider while searching for a best split. These will be randomly selected.
  - As a thumb–rule, square root of the total number of features works great but we should check upto 30–40% of the total number of features.
  - Higher values can lead to over–fitting but depends on case to case.

Ensemble Methods

- Ensemble learning is a paradigm where multiple models (often called “weak learners” or baseline models) are trained to **solve the same problem** and **combined to get better results**.
- The main hypothesis is that when **weak learners** are correctly combined we can obtain more accurate and/or robust models.



Combining weak models

- There are 3 methods to combine weak learners
  - **Bagging:** considers homogeneous weak learners, learns them independently from each other, in parallel, and combines them following some kind of deterministic averaging process

Combining weak models

- There are 3 methods to combine weak learners

  - **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 an adaptive way (a base model depends on the previous ones) and combines them following a deterministic strategy

Combining weak models

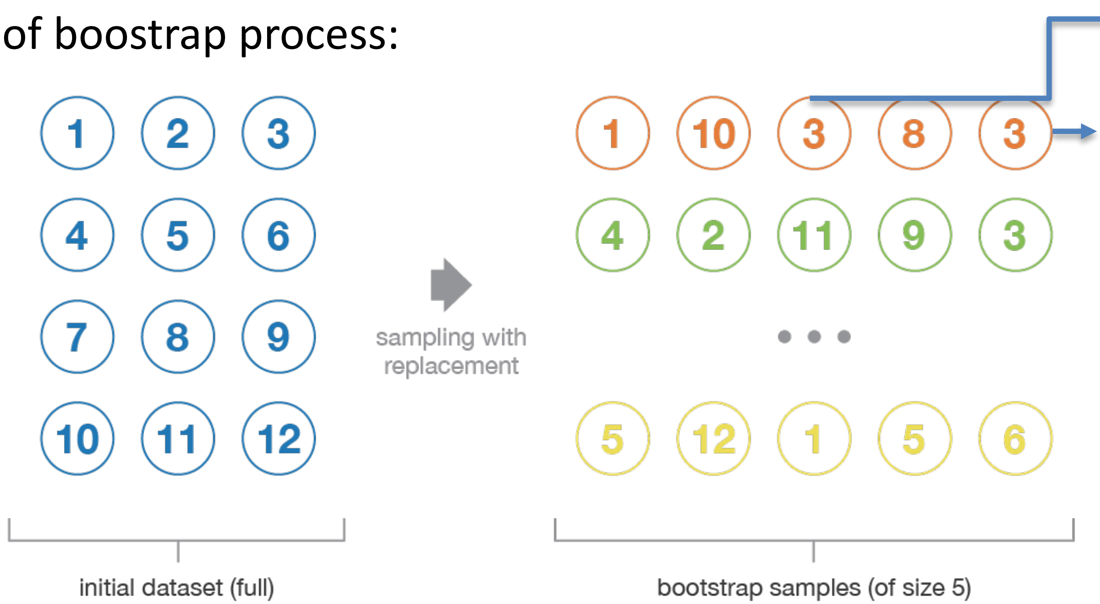
- There are 3 methods to combine weak learners
  - **Bagging:** considers homogeneous weak learners, learns them independently from each other, in parallel, and combines them following some kind of deterministic averaging process
  - **Boosting:** considers homogeneous weak learners, learns them sequentially in an 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

- ***Bagg**ing* = ***B**ootstrap **Agg**regat**ing***

What is a bootstrap sample ?
• Consider a data set $\mathcal{D}$ with n data points.
• A bootstrap sample $\mathcal{D}_i$ chooses n' data points from $\mathcal{D}$ randomly ***with replacement.***
• For a large value of n, on average 63% of the points in $\mathcal{D}$ will be selected (we will not prove this)

Example of boostrap process:



Sampling with replacement allows repetitions, e.g., number three, in the same $\mathcal{D}_i$

Bagging

- Bagging methodology:
  - **1st step:** Create B bootstrap samples
  - **2nd step:** Fit a weak learner for each of the B samples
    - ***B* Classifiers** $\in \{-1, 1\}$: $c^1, c^2, c^3, \ldots, c^B$
    - ***B* Estimated probabilities** $\in [0, 1]$: $p^1, p^2, p^3, \ldots, p^B$
  - **3rd step:** Aggregate them such that we kind of "average" their outputs

Regression

$$c_{bag} = \frac{1}{B} \sum_{b=1}^{B} c^b(\mathbf{x})$$

Classification

Hard-voting

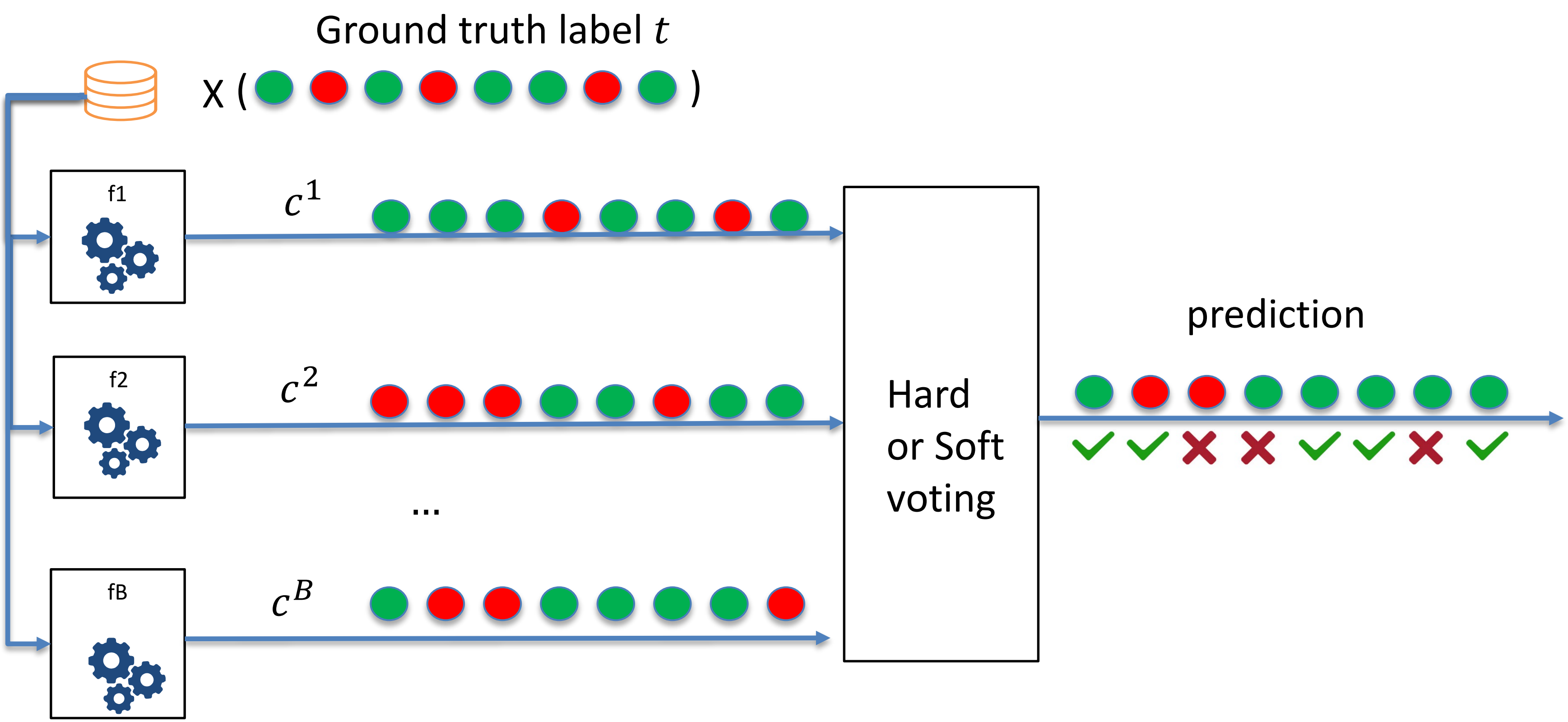
$$c_{bag}(\mathbf{x}) = \operatorname{sign}\left(\frac{1}{B} \sum_{b=1}^{B} c^b(\mathbf{x}) \right)$$

Soft-voting

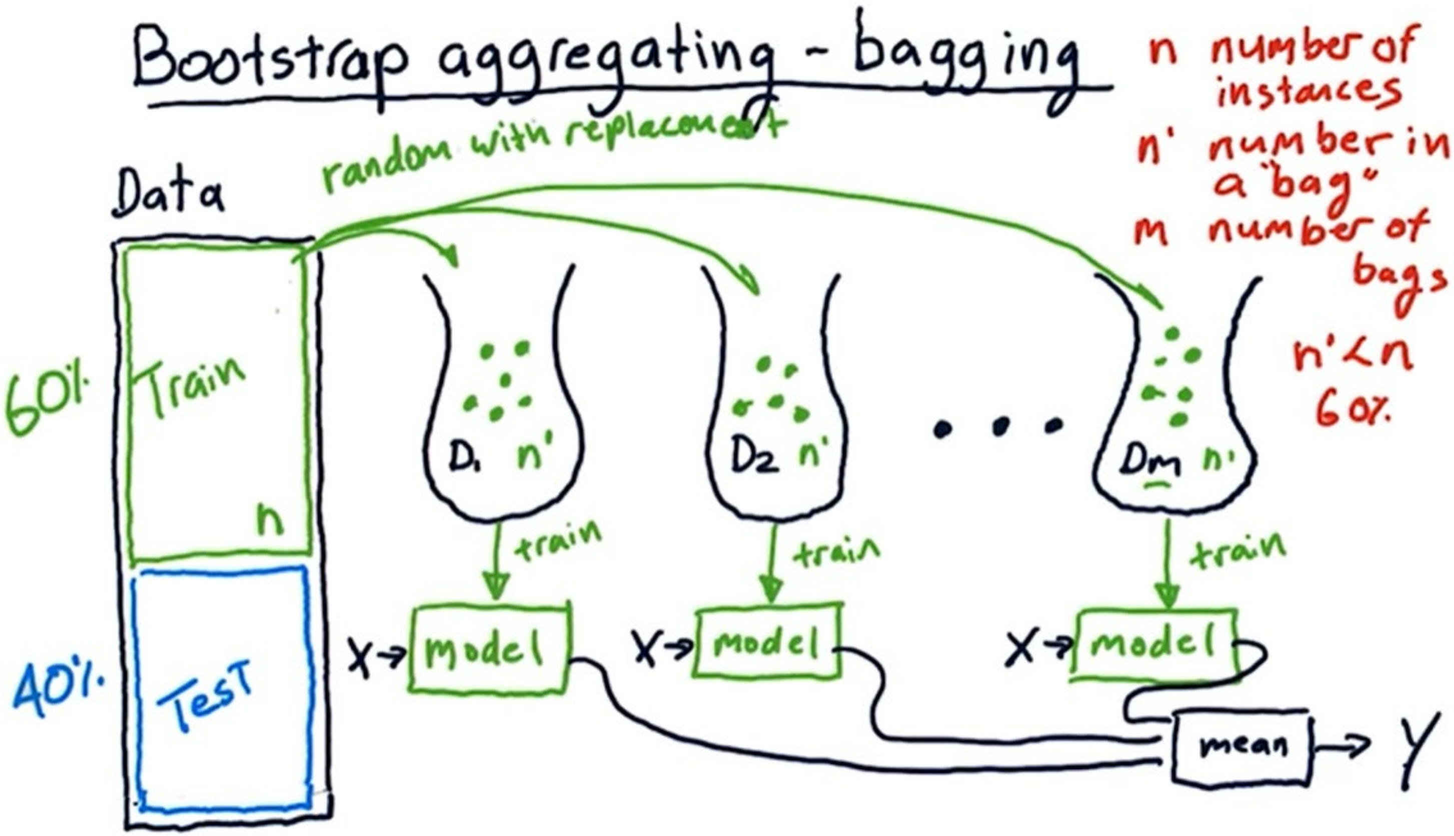
$$p_{bag}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} p^b(\mathbf{x})$$

Bagging

- Example:



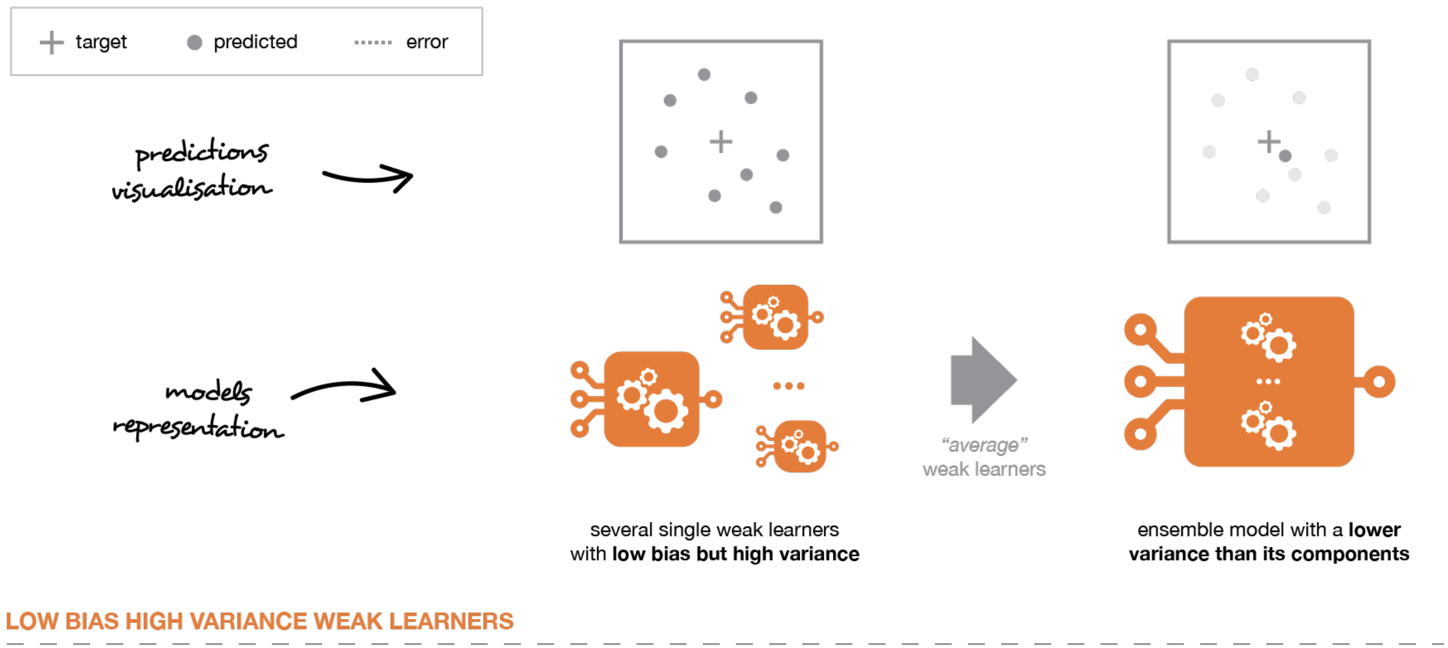
Bagging



Extracted from: https://www.youtube.com/watch?time_continue=171&v=2Mg8QD0F1dQ

Bagging

- With bagging we obtain an ensemble model with ***less variance*** that its components
  - "Averaging" weak learners outputs do not change the expected answer but reduce its variance (just like averaging i.i.d. random variables preserve expected value but reduce variance)

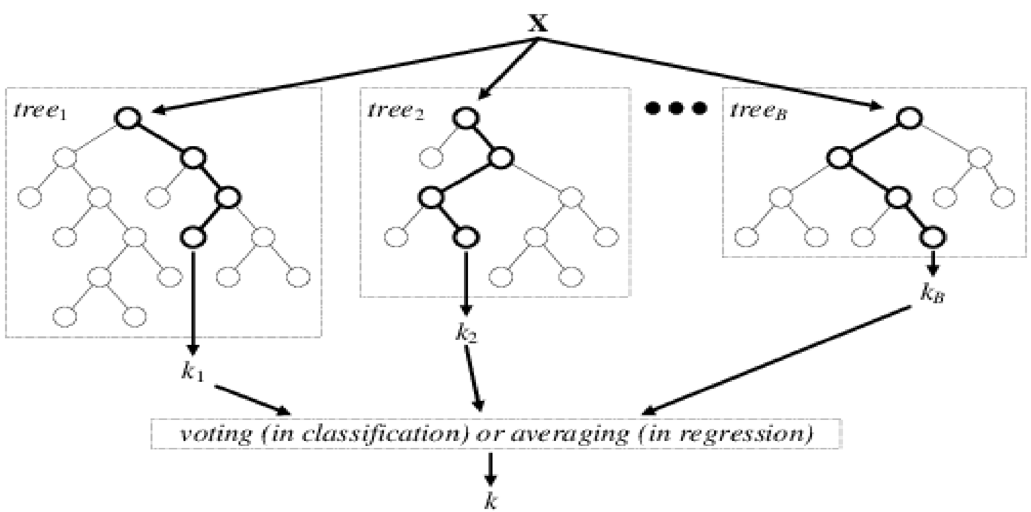


Bagging-Random Forest

- The ***random forest*** approach is a bagging method where ***deep trees***, fitted on bootstrap samples, are combined to produce an output ***with lower variance***.
- Besides, to make fitted trees less correlated with each other, Random Forest:

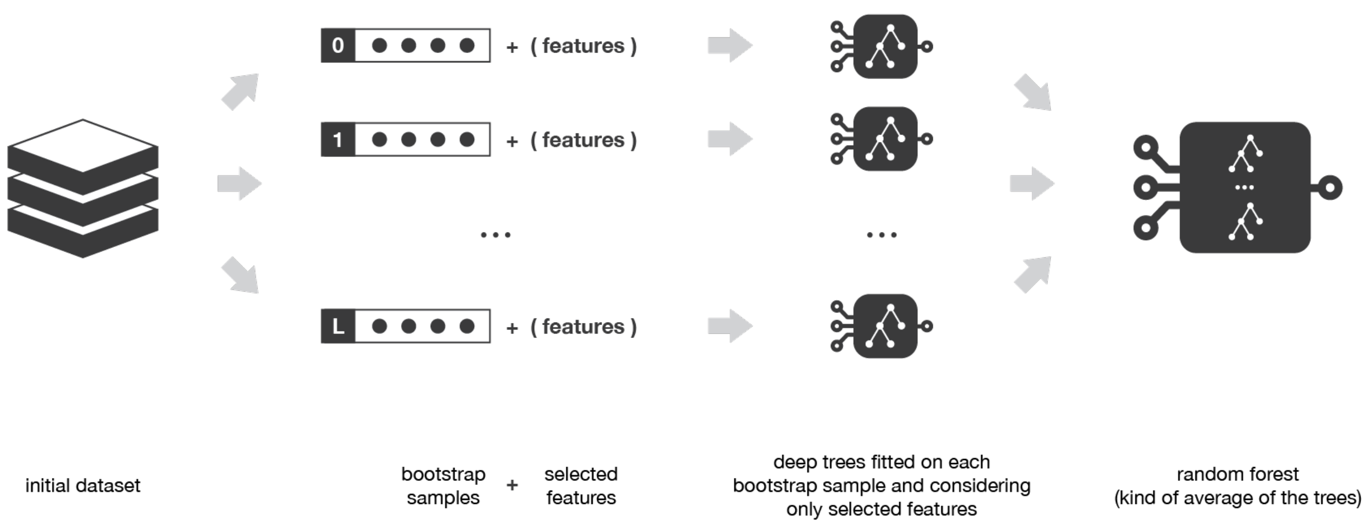
  1) **samples observations** in the dataset to generate a bootstrap sample

  2) **samples features** and keeps only a random subset of them to build the tree



Bagging-Random Forest

- Sampling over features creates more robust models because:
  - all trees are not trained with the same information and **there is less correlation between the different trees**
  - **the decision making process is more robust to missing data:** observations with missing data can be classified taking into account only features where their data are not missing



Boosting

- Freund and Schapire (1997), Breiman (1998)
- In adaptative boosting ("**Adaboost**") we try to define our ensemble model as a weighted sum of *B* weak learners

$$\boxed{c_{\text{boost}}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{B} \boldsymbol{\alpha}_i \cdot h_i(\mathbf{x})\right)}$$

where $h(\mathbf{x})$ is a weak learner

- Adaboost fits models **iteratively** such that the training of model at a given step depends on the models fitted at the previous steps
- **Therefore Adaboost has the ability to learn from past errors, since, at each iteration, the next classifier is built based on the misclassification error of the past one**

Boosting

- Methodology: weak learners are added one by one, looking at each iteration for the best possible pair (coefficient, weak learner) to add to the current ensemble model
- AdaBoost (a particular boosting technique)

Given a dataset $\mathcal{D} = \{\mathbf{x}, t\}_{n=1}^{N}$ (asume $t(n) \in \{-1, +1\}$)

1. Initialize observation weigths $\omega_1(n) = \frac{1}{N} \ \ \forall n$

Boosting

- Methodology: weak learners are added one by one, looking at each iteration for the best possible pair (coefficient, weak learner) to add to the current ensemble model
- AdaBoost (a particular boosting technique)

Given a dataset $\mathcal{D} = \{\mathbf{x}, t\}_{n=1}^{N}$ (asume $t(n) \in \{-1, +1\}$)

1. Initialize observation weigths $\omega_1(n) = \frac{1}{N} \;\; \forall n$
2. For $i = 1, \ldots, B$
   a) Fit a classifier h_i that minimizes the error for sample wieghts $\omega_i(n)$
   b) Compute error

$$\text{error}_i = \sum_{n=1}^{N} \omega_i(n)[t(n) \neq h_i(\mathbf{x}_n)]$$

Boosting

- Methodology: weak learners are added one by one, looking at each iteration for the best possible pair (coefficient, weak learner) to add to the current ensemble model
- AdaBoost (a particular boosting technique)

Given a dataset $\mathcal{D} = \{\mathbf{x}, t\}_{n=1}^{N}$ (asume $t(n) \in \{-1, +1\}$)

1. Initialize observation weigths $\omega_1(n) = \frac{1}{N}\ \forall n$
2. For $i = 1, \dots, B$
   a) Fit a classifier h_i that minimizes the error for sample wieghts $\omega_i(n)$
   b) Compute error

$$\text{error}_i = \sum_{n=1}^{N} \omega_i(n)[t(n) \neq h_i(\mathbf{x}_n)]$$

   c) Compute $\boldsymbol{\alpha}_i$

$$\boldsymbol{\alpha}_i = \frac{1}{2}\ln\left(\frac{1 - \text{error}_i}{\text{error}_i}\right)$$

$\text{error}_i < .5 \rightarrow \boldsymbol{\alpha}_i > 0$

$\text{error}_i > .5 \rightarrow \boldsymbol{\alpha}_i < 0$

Low error rate implies positive $\boldsymbol{\alpha}_i$

Boosting

- Methodology: weak learners are added one by one, looking at each iteration for the best possible pair (coefficient, weak learner) to add to the current ensemble model
- AdaBoost (a particular boosting technique)

Given a dataset $\mathcal{D} = \{\mathbf{x}, t\}_{n=1}^{N}$ (asume $t(n) \in \{-1, +1\}$)

1. Initialize observation weigths $\omega_1(n) = \frac{1}{N} \quad \forall n$
2. For $i = 1, \dots, B$
   a) Fit a classifier h_i that minimizes the error for sample wieghts $\omega_i(n)$
   b) Compute error

   $$\text{error}_i = \sum_{n=1}^{N} \omega_i(n)[t(n) \neq h_i(\mathbf{x}_n)]$$

   c) Compute $\boldsymbol{\alpha}_i$

   $$\boldsymbol{\alpha}_i = \frac{1}{2} \ln\left(\frac{1 - \text{error}_i}{\text{error}_i}\right)$$

   $\text{error}_i < .5 \rightarrow \boldsymbol{\alpha}_i > 0$

   $\text{error}_i > .5 \rightarrow \boldsymbol{\alpha}_i < 0$

   Low error rate implies positive $\boldsymbol{\alpha}_i$

   d) Update weights:

   $$\omega_{i+1}(n) = \left(\frac{\omega_i(n) \cdot e^{-\boldsymbol{\alpha}_i t_n h_i(\mathbf{x}_n)}}{Z_t}\right)$$

   where Z_t is a normalization factor chosen $\sum_{n=1}^{N} \omega_{i+1}(n) = 1$

Boosting

- Methodology: weak learners are added one by one, looking at each iteration for the best possible pair (coefficient, weak learner) to add to the current ensemble model
- AdaBoost (a particular boosting technique)

Given a dataset $\mathcal{D} = \{\mathbf{x}, t\}_{n=1}^{N}$ (asume $t(n) \in \{-1, +1\}$)

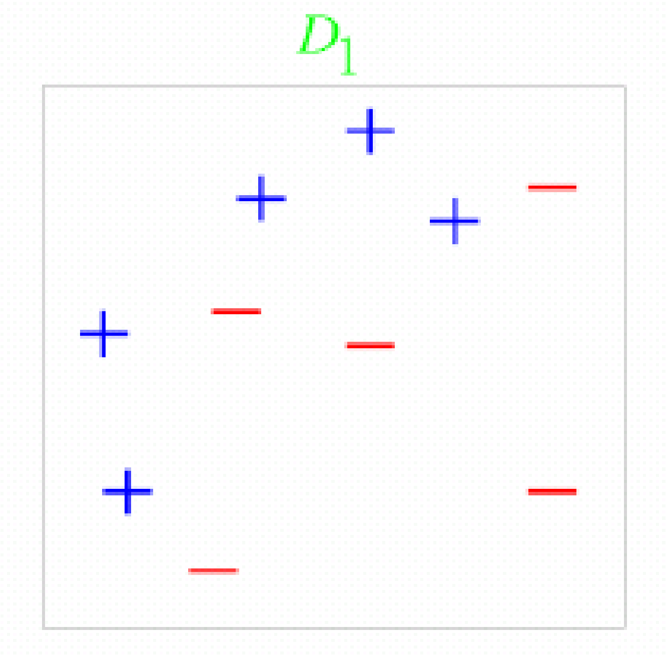
1. Initialize observation weigths $\omega_1(n) = \frac{1}{N} \quad \forall n$
2. For $i = 1, \dots, B$
   a) Fit a classifier h_i that minimizes the error for sample wieghts $\omega_i(n)$
   b) Compute error
   $$\text{error}_i = \sum_{n=1}^{N} \omega_i(n)[t(n) \neq h_i(\mathbf{x}_n)]$$
   c) Compute $\boldsymbol{\alpha}_i$
   $$\boldsymbol{\alpha}_i = \frac{1}{2}\ln\left(\frac{1 - \text{error}_i}{\text{error}_i}\right)$$
   $\text{error}_i < .5 \rightarrow \boldsymbol{\alpha}_i > 0$
   $\text{error}_i > .5 \rightarrow \boldsymbol{\alpha}_i < 0$
   Low error rate implies positive $\boldsymbol{\alpha}_i$
   d) Update weights:
   $$\omega_{i+1}(n) = \left(\frac{\omega_i(n) \cdot e^{-\boldsymbol{\alpha}_i t_n h_i(\mathbf{x}_n)}}{Z_t}\right)$$
   where Z_t is a normalization factor chosen $\sum_{n=1}^{N} \omega_{i+1}(n) = 1$

Final prediction:

$$c_{\text{boost}} = \text{sign}\left(\sum_{i=1}^{B} \boldsymbol{\alpha}_i \cdot h_i(\mathbf{x})\right)$$

Boosting

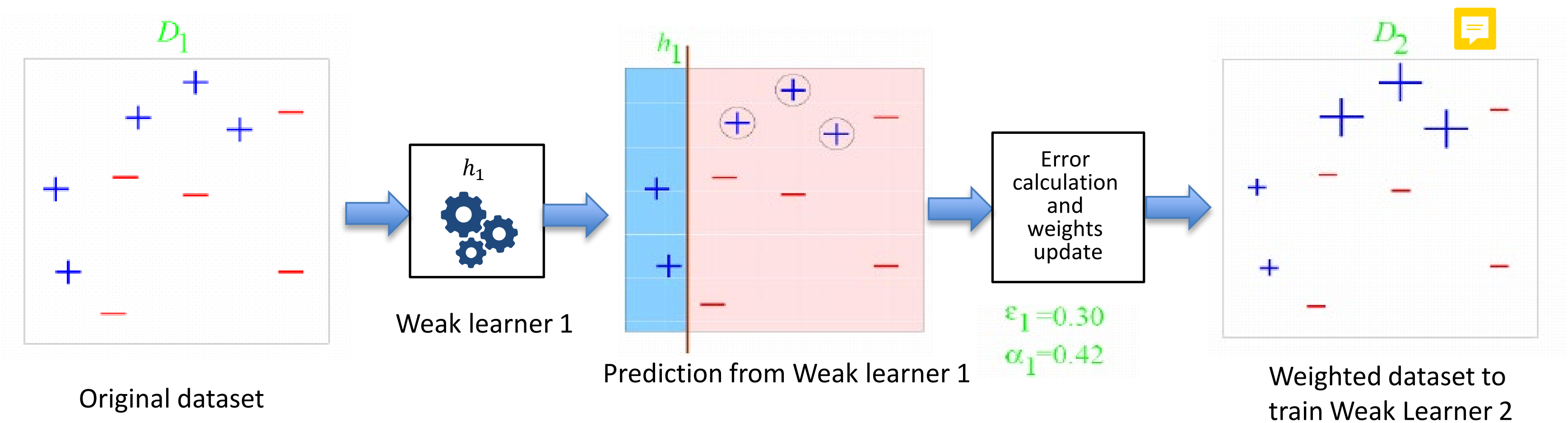
- Example: Training set: 10 points (represented by + and -)



Iteration 1:
Equal weights for all training samples

Boosting

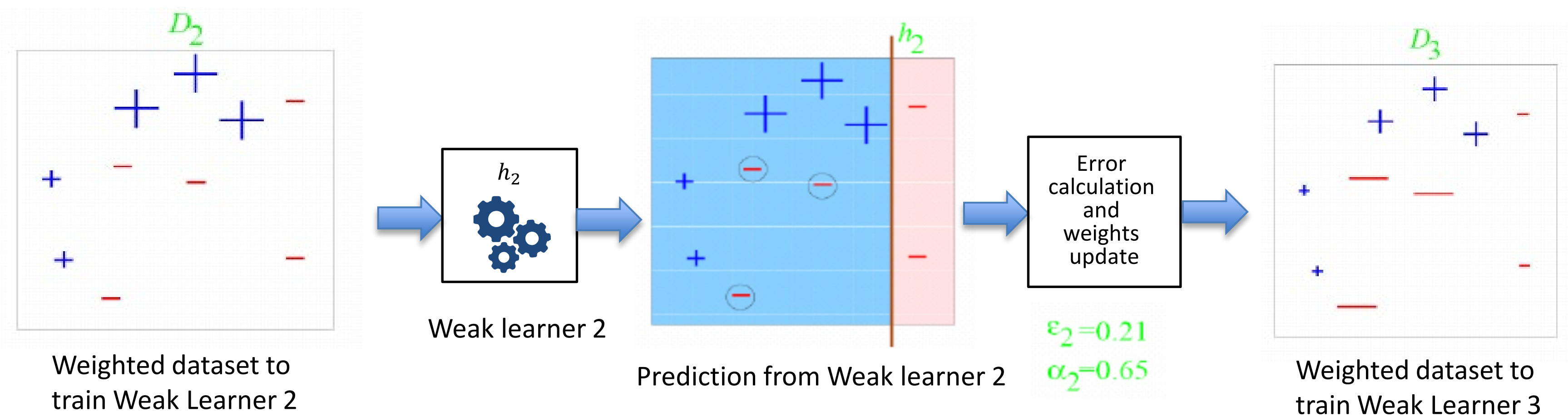
- Example: Training set: 10 points (represented by + and -)



Iteration 1:
Three + points are not correctly classified. They are given higher weights for the next weak learner

Boosting

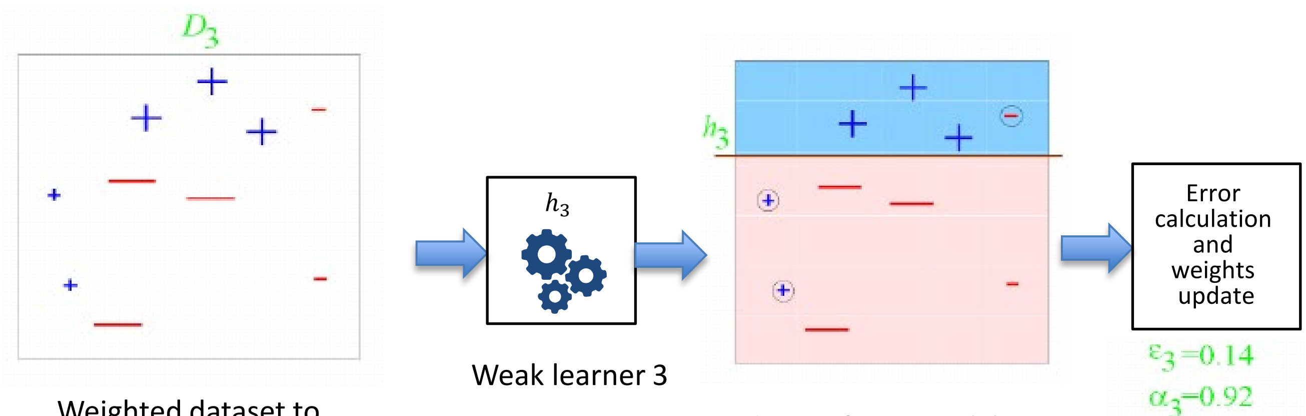
- Example: Training set: 10 points (represented by + and -)



Iteration 2:
Three - points are not correctly classified. They are given higher weights for the next weak learner

Boosting

- Example: Training set: 10 points (represented by + and -)



Weighted dataset to train Weak Learner 3

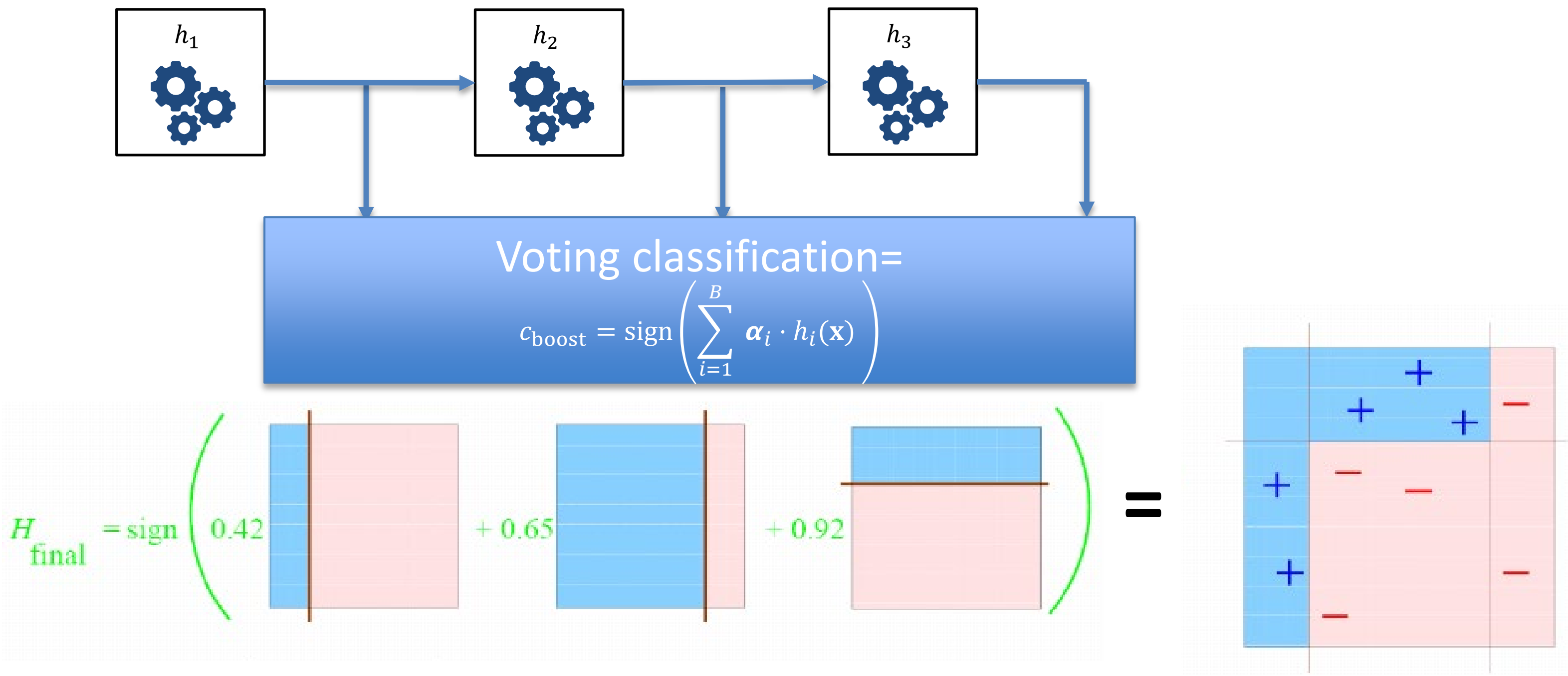
Weak learner 3

Prediction from Weak learner 3

Iteration 3:
One - and two + points are not correctly classified. They are given higher weights for the next weak learner

Boosting

- Example: Training set: 10 points (represented by + and -)

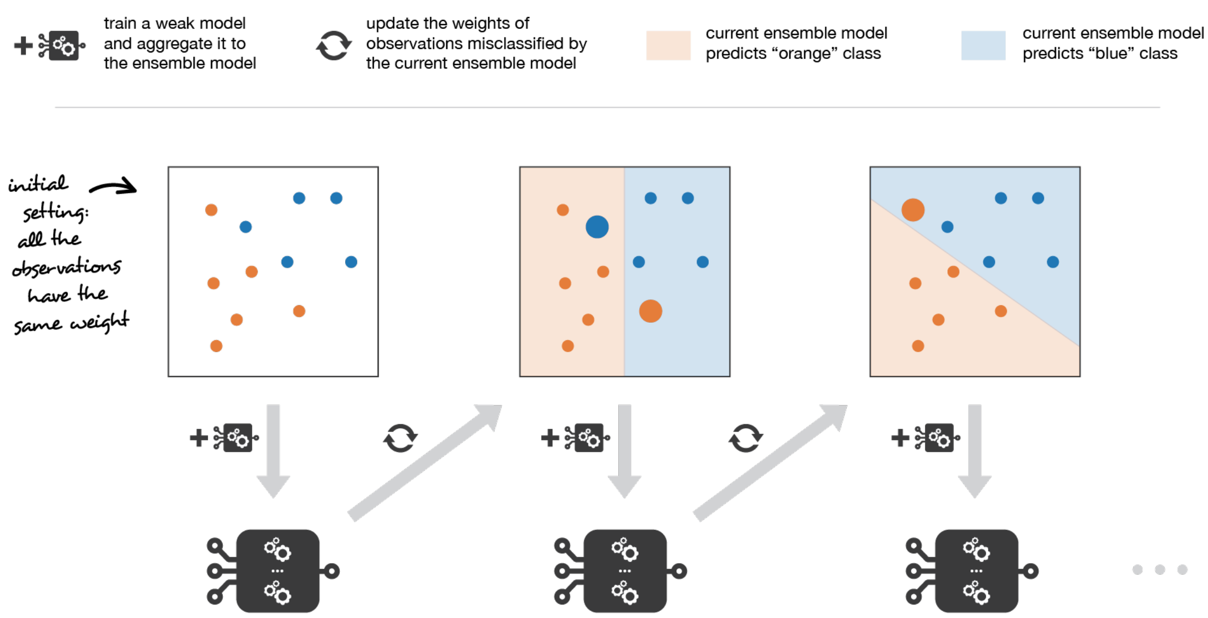


Final classifier:

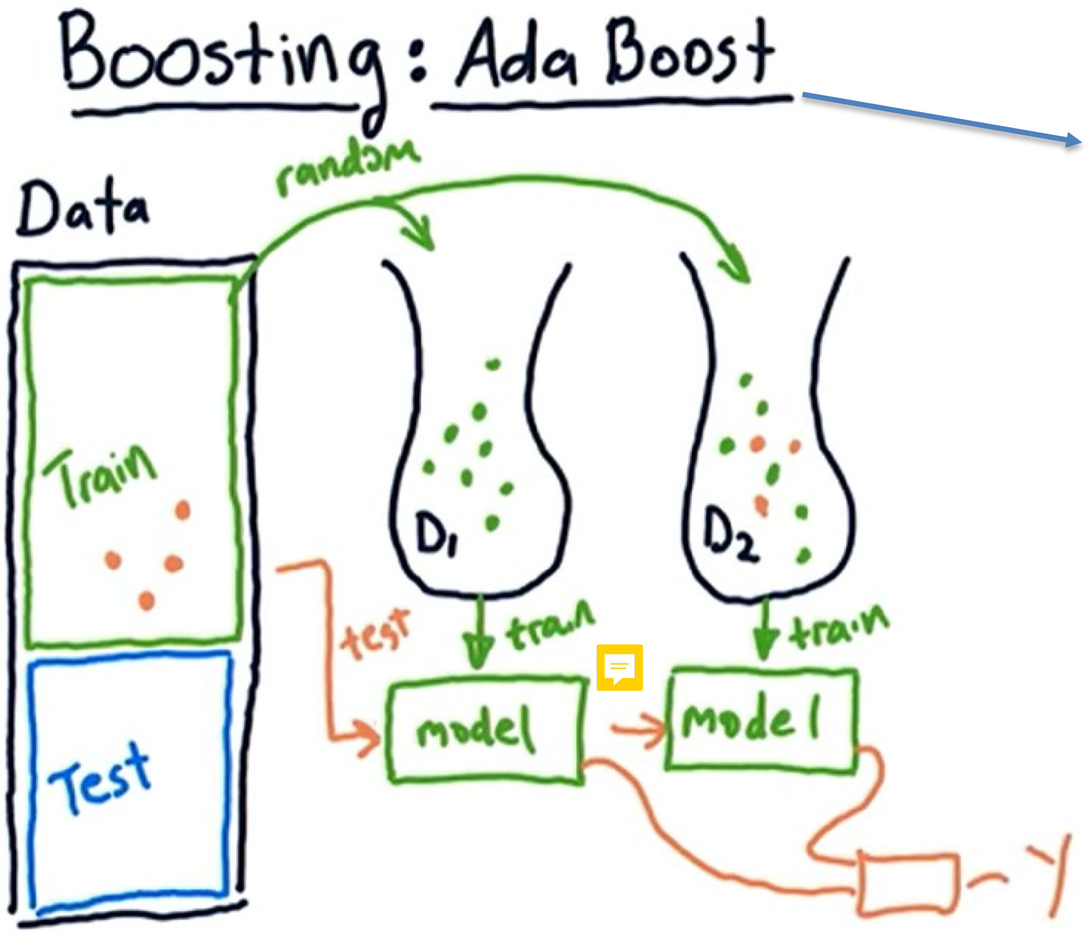
Integrate the three "weak" classifiers and obtain a final strong classifier

Boosting

- In Adaboost, the predictions will be trees with a heavier vote than others in opposition with bagging method.
- Those trees that performed the best during all the iterations (so, they showed very few misclassifications) will have “more importance” in the voting process



Boosting

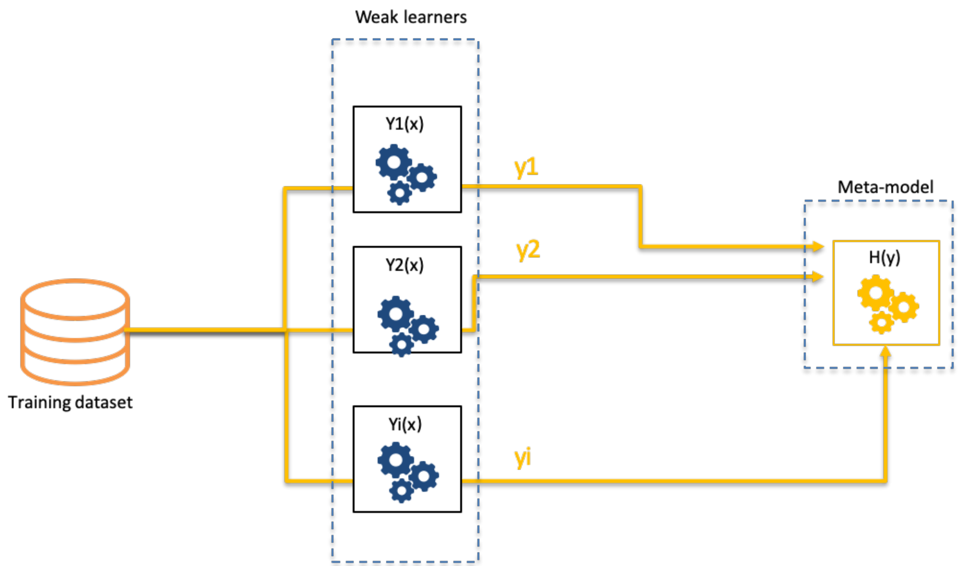


Adaptive
Boosting

Extracted from: https://www.youtube.com/watch?time_continue=143&v=GM3CDQfQ4sw

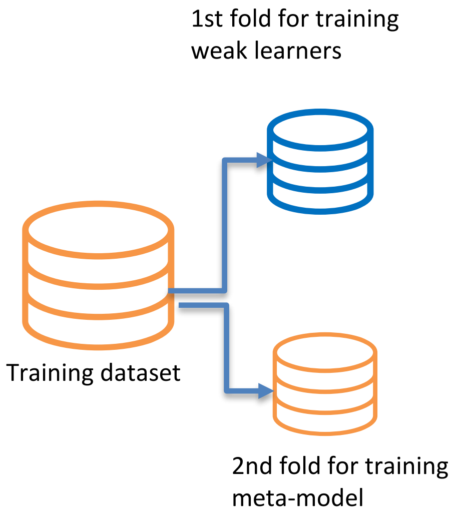
Stacking

- The idea of stacking is to 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, for a classification problem, 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



Stacking

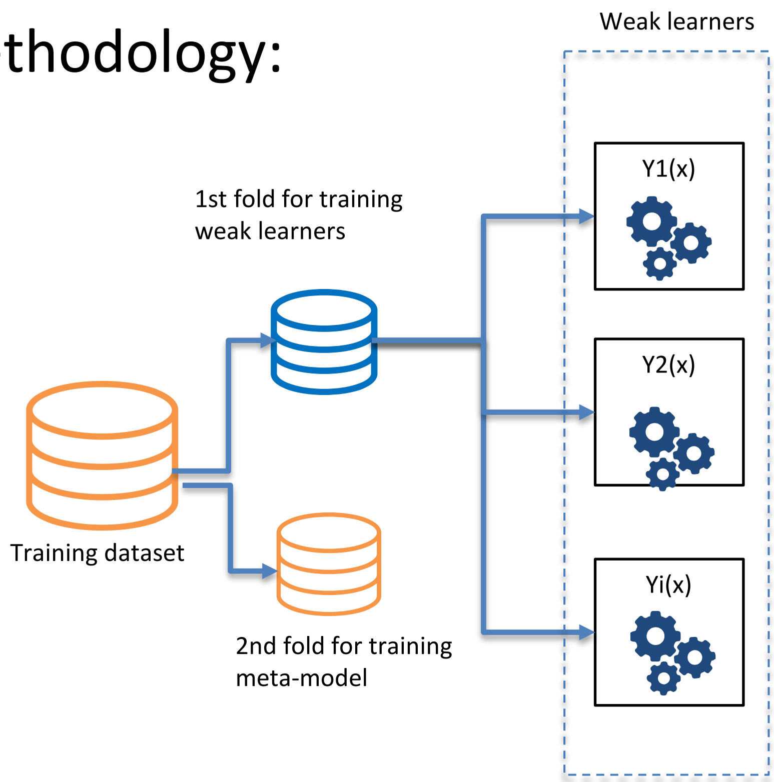
- Methodology:



Step 1:
Split the training dataset into 2 folds: 1 for training the weak learners and 1 for training the meta-model

Stacking

- Methodology:

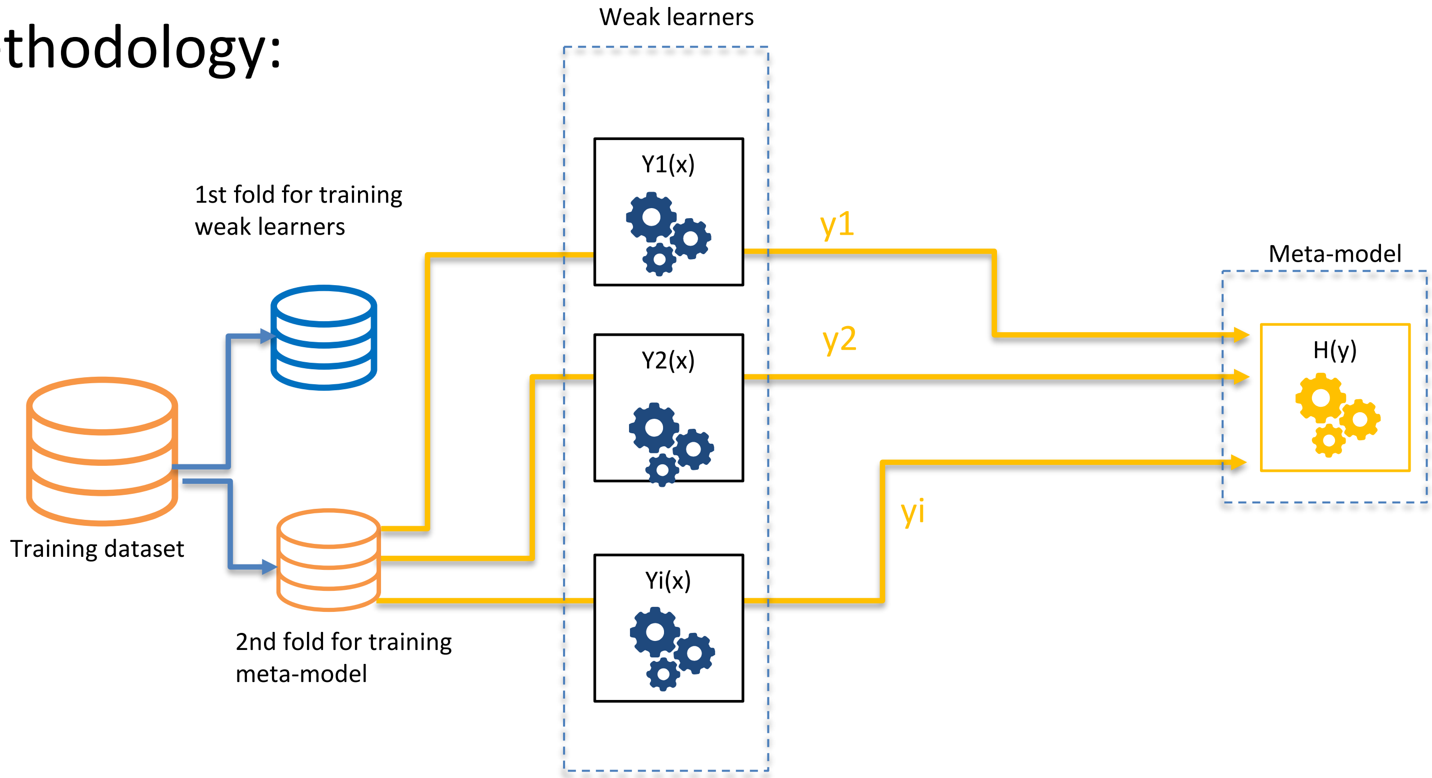


Step 2:

Train the weak learners with 1st fold

Stacking

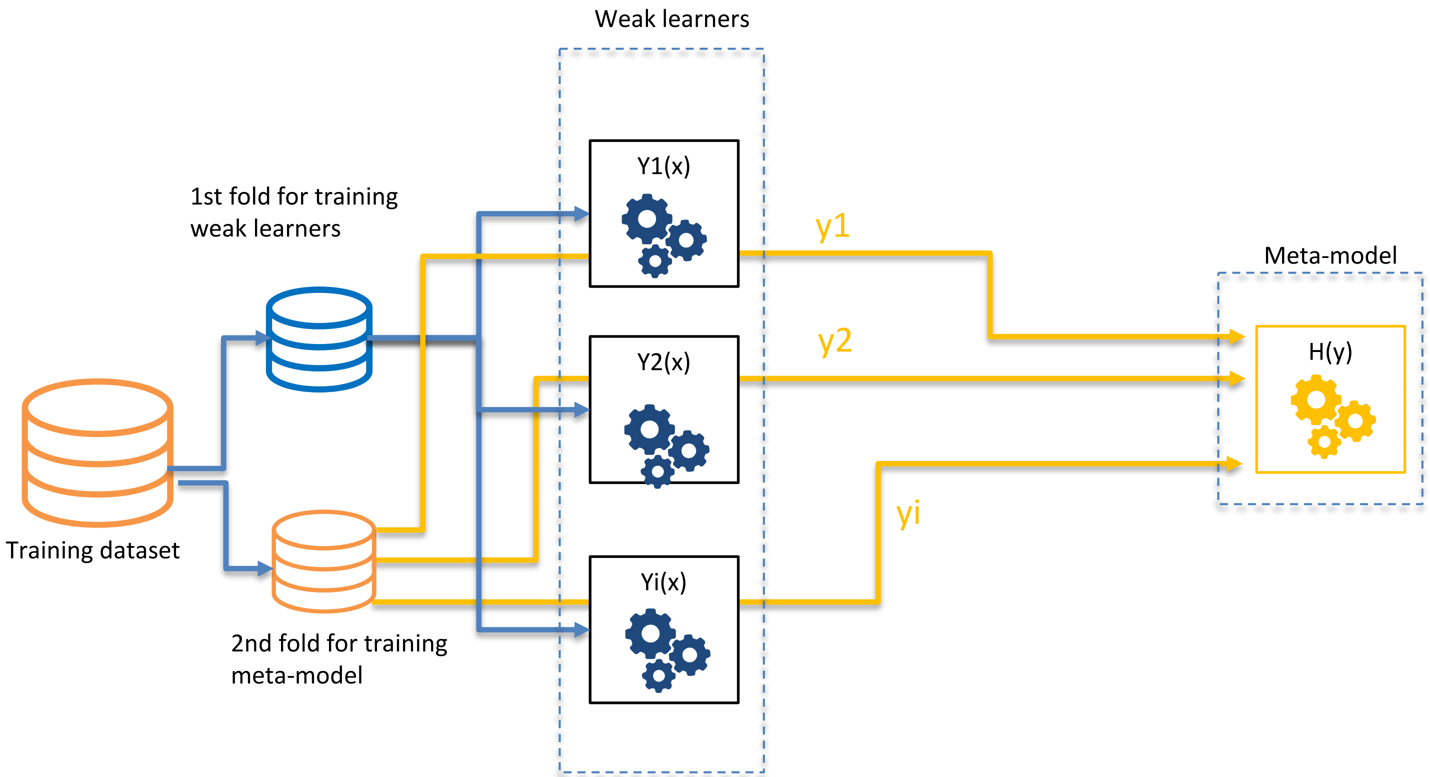
- Methodology:



Step 3:
Train the meta-model taking as an input the prediction of weak learners for 2^{nd} fold training dataset

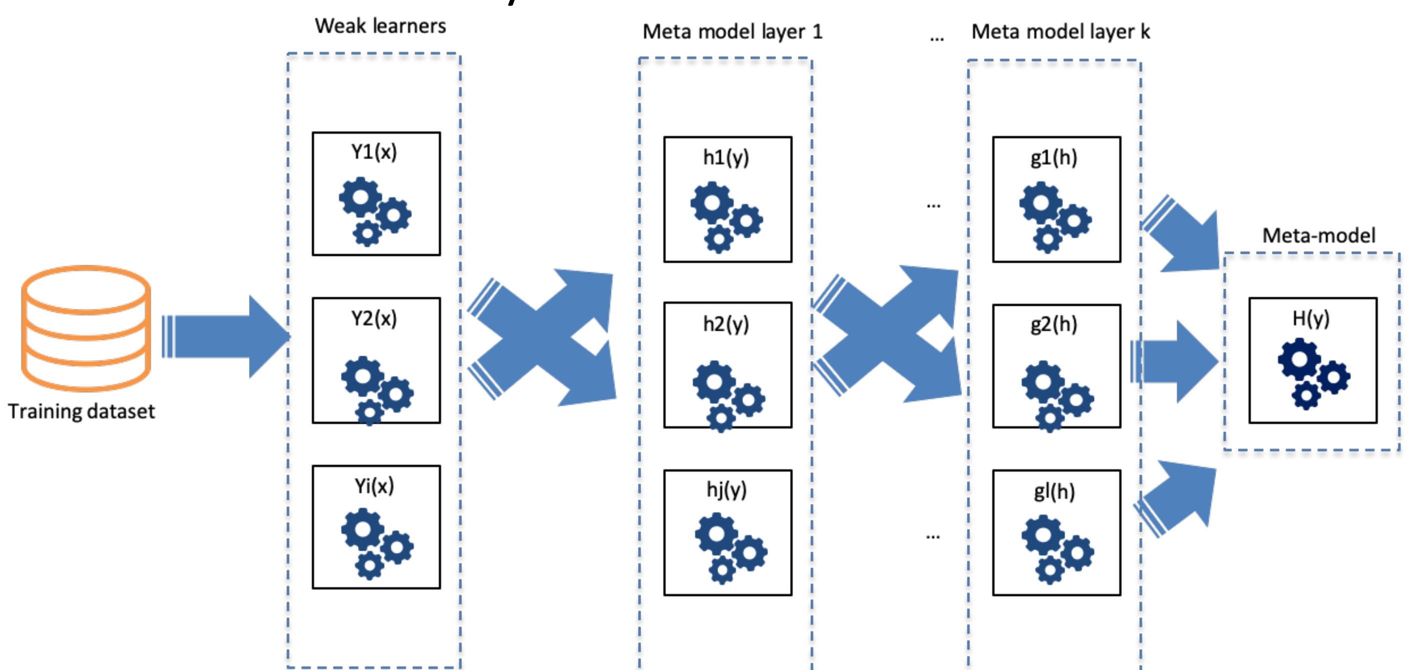
Stacking

- Methodology:



Stacking

- Multi-levels stacking consists in doing stacking with multiple layers of meta-models
  - First level: is formed by i weak learners
  - K-levels: are formed by several meta-models that are trained from the output of the previous k-1 layer
  - Last level: is formed by an only meta-model that takes as input of the previous meta-model layer



Summary

- Decision Trees are non-parametric classifiers that are very useful because:
  - Easy to explain results and how they work
  - High accuracy and stability
  - Solve regression and multi-class problems
- The process to calculate the Split of each node can be based on Gini or Information Gain/Entropy
- Ensemblers are the combination of weakers classifiers to improve the results:
  - Bagging
  - Boosting
  - Stacking
- Random Forest, XGBoost and AdaBoosting are ML techniques very useful today with similar performance that Deep Learning