Machine Learning

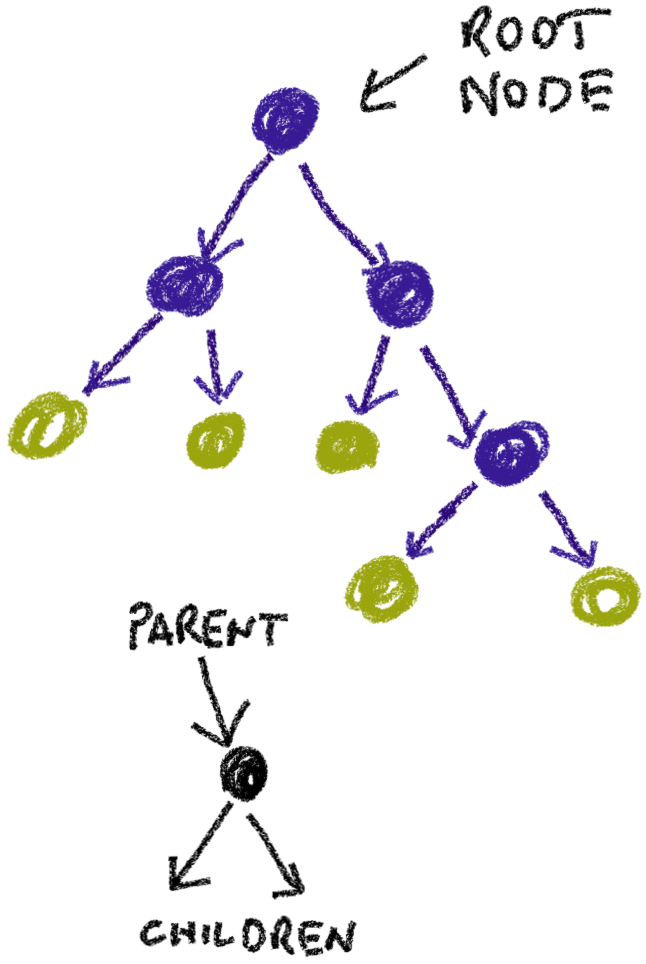
A) Decision trees,
B) Random forest and Ensemble learning

Cigdem Beyan

A. Y. 2024/2025

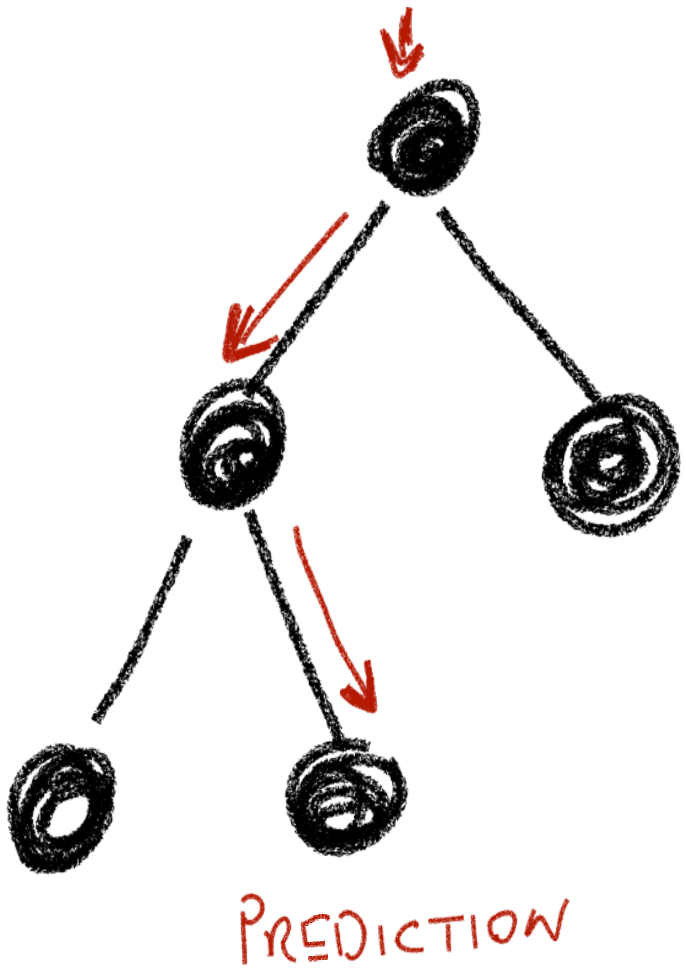
Decision trees

- A tree-structured, prediction model.
- It is composed of terminal (or leaf) nodes and non-terminal nodes.
- **Non-terminal** nodes have 2+ children and implement a routing function.
- **Leaf nodes** have no children (i.e. terminal) and implement a prediction function.
- There are no cycles, all nodes have at most 1 parent excepting one a.k.a. **root node**



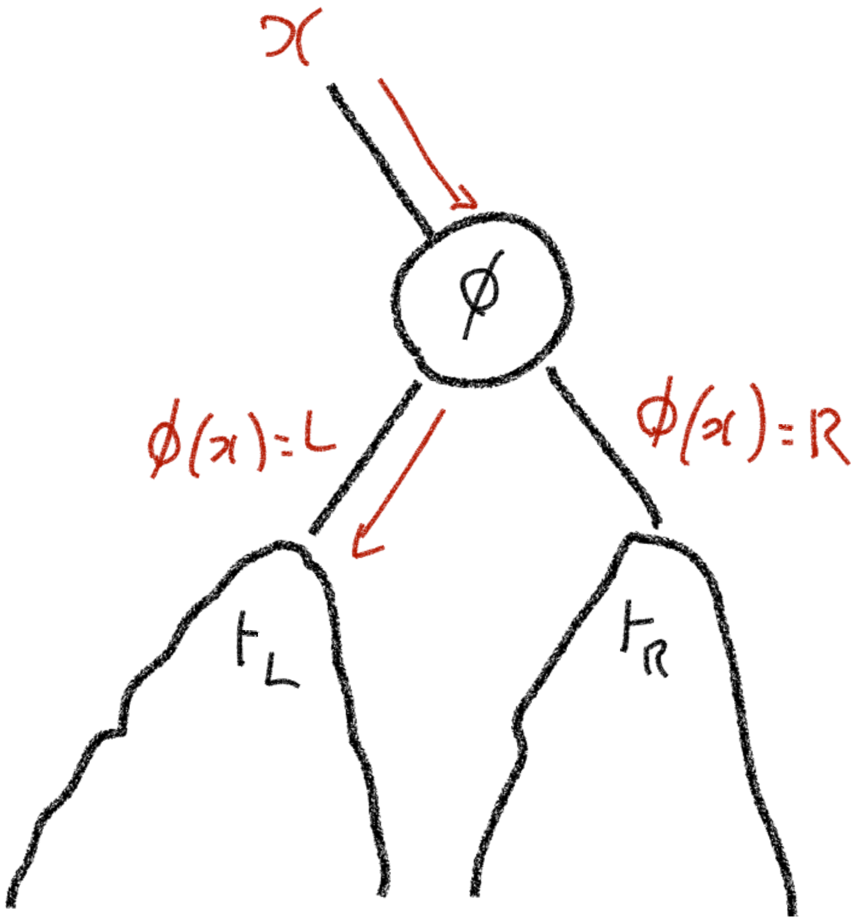
Decision trees

- A decision tree takes an input $x \in \mathcal{X}$ and routes it through its nodes until it reaches a leaf node.
- Each node represents a variable (feature).
- An edge to a child node represents the predicted value for the class based on the values.
- In the leaf a prediction takes place.



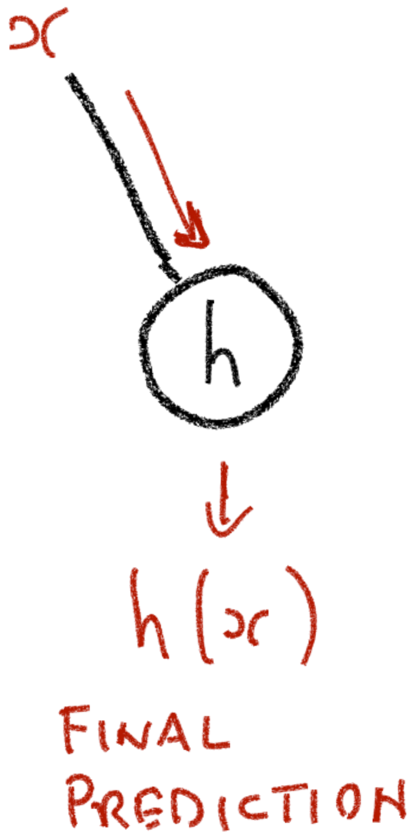
Decision trees - inference

- Each non-terminal node $\texttt{Node}(\phi, t_L, t_R)$ holds a **routing function** $\phi \in \{L,R\}^{\mathcal{X}}$ such that there exist a left child t_L and right child t_R
- When x reaches the node it will go to the left child t_L or the right child t_R depending on the value of $\phi(x) \in \{L,R\}$
- Here we assume a binary tree, thus there exist only 2 childs: left and right.



Decision trees - inference

- Each leaf node Leaf(h) holds a **prediction** function $h \in \mathcal{F}_{\text{task}}$ (typically a constant)
- Depending on the task we want to solve it can be $h \in \mathcal{Y}^{\mathcal{X}}$ e.g., **classification** or **regression**.
- Once x reaches the leaf the final prediction is given by $h(x)$.



UNIVERSITÀ di VERONA

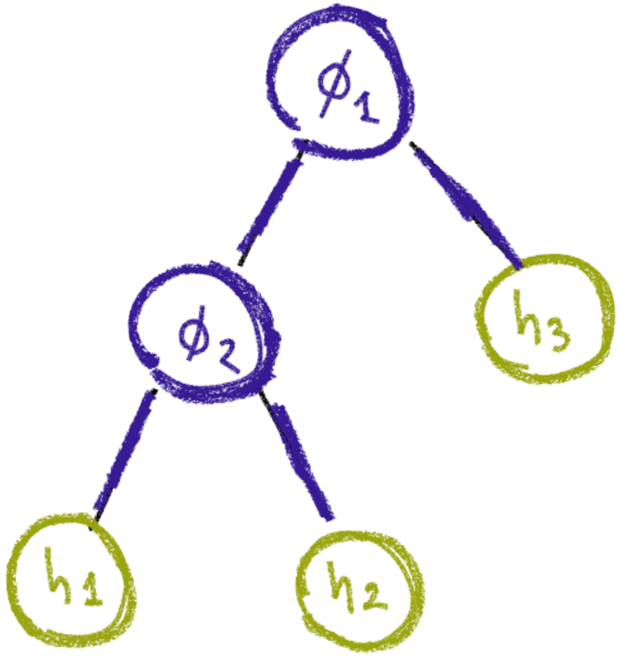
Decision trees - inference

- The decision tree function:

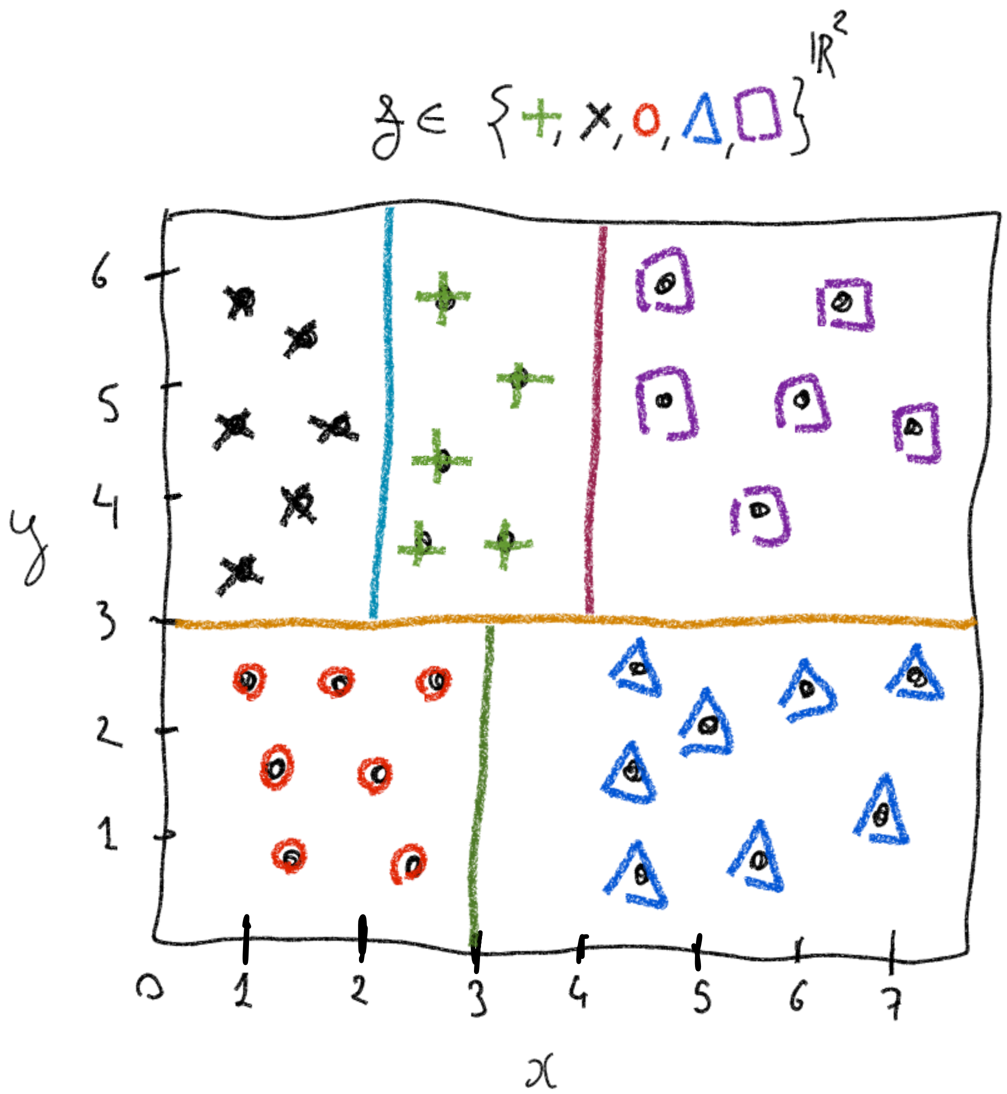
$$f_t(x) = \begin{cases} h(x) & \textbf{if } t = \texttt{Leaf}(h) \\ f_{t_{\phi(x)}}(x) & \textbf{if } t = \texttt{Node}(\phi, t_L, t_R) \end{cases}$$

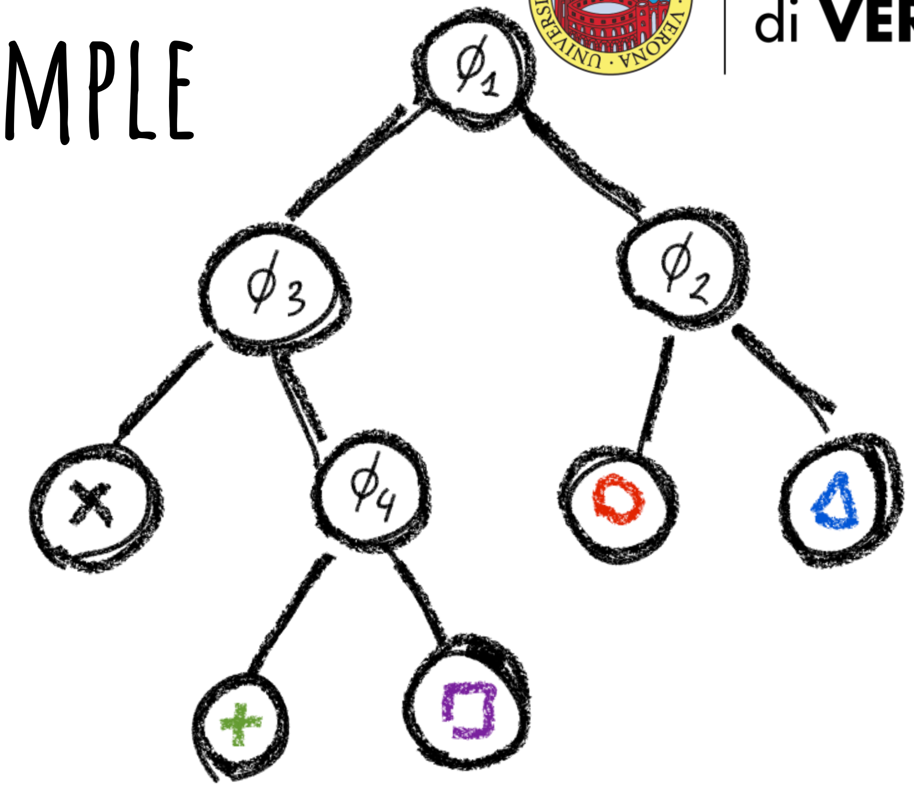
Where to go

Making prediction



Decision trees- inference example





$$\phi_1(x, y) = \begin{cases} L & \textbf{if } y \geq 3 \\ R & else \end{cases}$$

$$\phi_2(x, y) = \begin{cases} L & \textbf{if } x \leq 3 \\ R & else \end{cases}$$

$$\phi_3(x, y) = \begin{cases} L & \textbf{if } x \leq 2 \\ R & else \end{cases}$$

$$\phi_4(x, y) = \begin{cases} L & \textbf{if } x \leq 4 \\ R & else \end{cases}$$

Decision trees- learning algorithm

- Given a training set: $\mathscr{D}_n = \{z_1, \ldots, z_n\}$, we want to find a tree $t^\star$

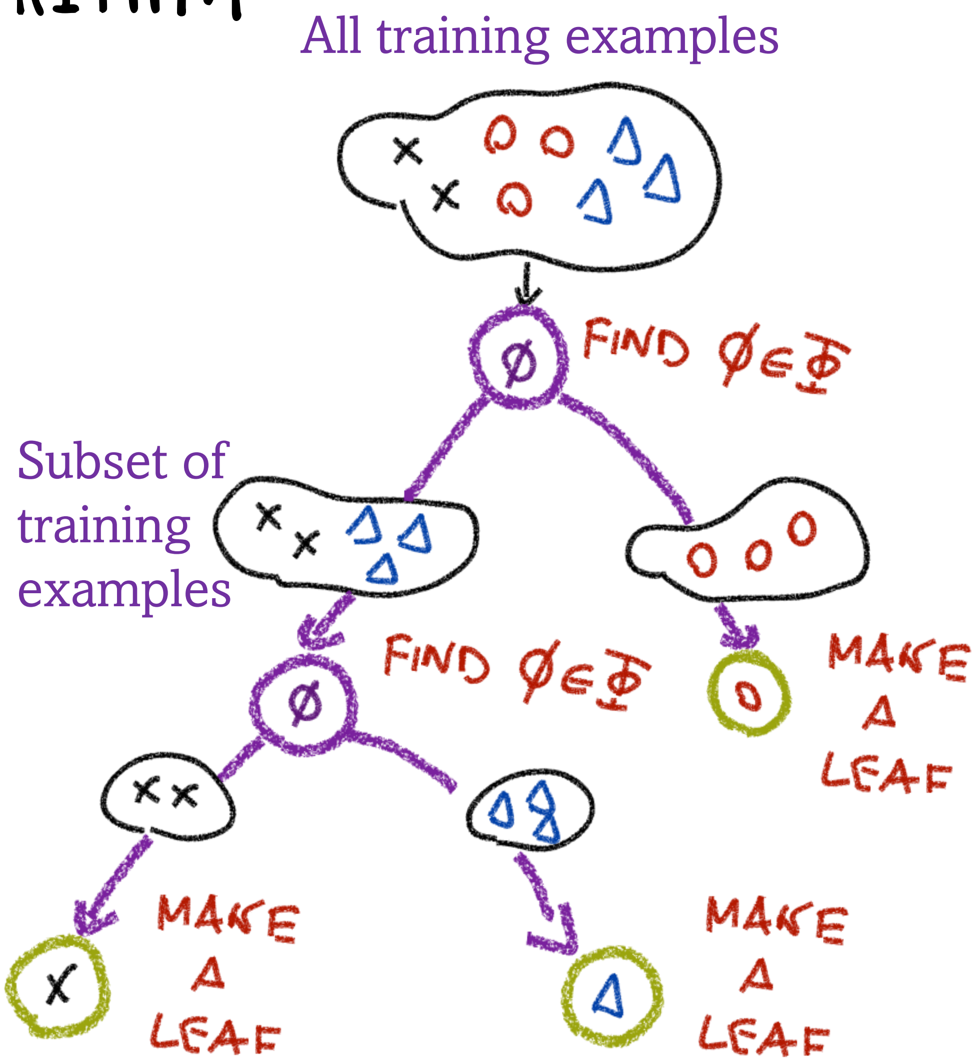
$$\boxed{t^\star \in \underbrace{\arg\min}_{t \in \mathscr{T}} E(f_t ; \mathscr{D}_n)}$$

set of decision trees

- This optimization problem has many solutions
  - Thus, we need to impose constraints, e.g., most compact tree, otherwise it could be NP-hard

Decision trees- learning algorithm

- We need to fix a set of leaf predictions $\mathscr{H}_{\text{leaf}} \subset \mathscr{F}_{\text{task}}$ (e.g., constant functions)
- Fix a set of possible splitting functions $\Phi \subset \{L, R\}^{\mathscr{X}}$
- Tree-growing strategy recursively partitions the training set and decides whether to grow the leaves or non-terminal nodes.
  - ID3 algorithm by Ross Quinlan
  - CaRT by Breiman et al.



ID3/CART ALGORITHM

UNIVERSITÀ di VERONA

Input

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

[See: Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997]

- 14 training samples
- 4 attributes /features
- 9 of them class: ”Yes”
- 5 of them class: “NO”

Id3/CART algorithm
How do you pick the best attribute (feature)

- Which split do you find better and why?

- Outlook, because it has a pure subset

ID3/CART ALGORITHM
HOW DO YOU PICK THE BEST ATTRIBUTE (FEATURE)

- If you look at “wind”, which subset of the wind is better than other? Why?

- “Weak” is better, even though it is not pure.

ID3/CART ALGORITHM
HOW DO YOU PICK THE BEST ATTRIBUTE (FEATURE)

UNIVERSITÀ di VERONA

- We want to measure "purity" of the split.
  - We want to have more certain about Yes/No after the split.
  - Pure set (4 yes / 0 no) ➔ completely certain 100%
  - Impure set (3 yes / 3 no) ➔ completely uncertain 50%
- This measure should be symmetric!
  - 4 yes / 0 no is as pure as 0 yes / 4 no
- A lot of different ways to measure the purity (e.g., entropy)

Entropy

- Assume that we have binary classes (positives and negatives)
- p(+) is the proportion of positives in a subset
- p(-) is the proportion of negatives in a subset
- Entropy of a subset ➔ $H(S) = - p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)}$
- Impure (3 yes / 3 no): $H(S) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1$
- Pure (4 yes / 0 no): $H(S) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0$

Higher numbers to the subset that are less pure!!!

Information gain

- We want to have many items in the pure sets.
- We calculate the expected drop in entropy after each split:

$$Gain(S,A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V)$$

V: possible values of A
S: set of examples {X}
Sv: subset where X_A= V

$$-\tfrac{9}{14}\log_2\tfrac{9}{14} - \tfrac{5}{14}\log_2\tfrac{5}{14}$$

H(S) = 0.94

$$-\tfrac{6}{8}\log_2\tfrac{6}{8} - \tfrac{2}{8}\log_2\tfrac{2}{8}$$

$H(S_{weak}) = 0.81$

$$-\tfrac{3}{6}\log_2\tfrac{3}{6} - \tfrac{3}{6}\log_2\tfrac{3}{6}$$

$H(S_{strong}) = 1.0$

Mutual information between A and class labels of S:

Gain (S, Wind)

$$= H(S) - {}^{8}/_{14}\, H(S_{weak}) - {}^{6}/_{14}\, H(S_{weak})$$
$$= 0.94 - {}^{8}/_{14} * 0.81 - {}^{6}/_{14} * 1.0$$
$$= 0.049$$

Information gain

$-\frac{9}{14}\log_2\frac{9}{14}-\frac{5}{14}\log_2\frac{5}{14}$

H(S) = 0.94

$-\frac{6}{8}\log_2\frac{6}{8}-\frac{2}{8}\log_2\frac{2}{8}$

$H(S_{weak}) = 0.81$

$-\frac{3}{6}\log_2\frac{3}{6}-\frac{3}{6}\log_2\frac{3}{6}$

$H(S_{strong}) = 1.0$

Mutual information between A and class labels of S:

Gain (S, Wind)

$= H(S) - {}^{8}/_{14}\, H(S_{weak}) - {}^{6}/_{14}\, H(S_{weak})$

$= 0.94 - {}^{8}/_{14} * 0.81 - {}^{6}/_{14} * 1.0$

$= 0.049$

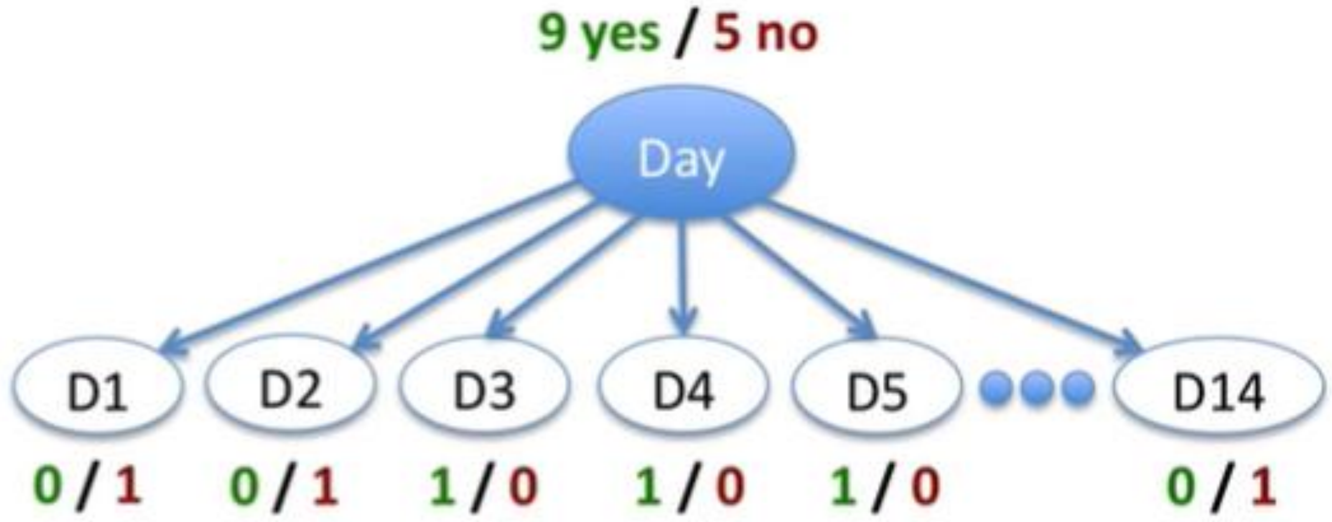
We use information gain to decide which attribute to pick. We want to maximize the "information gain".

Information gain

- We use information gain to decide which attribute to pick.
  - Take every attribute that you have in your data
  - Compute gain for that attribute
  - Select the attribute that has the highest information gain.
  - Highest? That’s the attribute which reduce the uncertainty the most, aka lead the purest possible split out of all attributes.
- We consider one level split at a time, but remember the procedure is recursive.

Information Gain-- problems

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No



All subsets perfectly pure ➔optimal split

According to definition of *Information Gain*, "Day" is a perfect attribute. But….

Problem!!!!!

Won't work for a new data!

"D15 Rain High Weak"

Generalize poor in the testing data

Information Gain– Cons.

UNIVERSITÀ di VERONA

According to definition of *Information Gain*, "Day" is a perfect attribute. But….

Generalize poor in the testing data

How to handle this? One possibility if using **GainRatio**

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitEntropy(S,A)}$$

$$SplitEntropy(S,A) = -\sum_{V \in Values(A)} \frac{|S_V|}{|S|} \log \frac{|S_V|}{|S|}$$

A: candidate attribute
V: possible values of A
S: set of examples {X}
Sv: subset where $X_A = V$

Penalize attributes with many values

Information Gain– pros.

- Decision trees are interpretable.

- It is possible to read the rules of the tree. There is concise description of what makes an item positive/negative.

- No “black box”
  - Important for users!

Rule: (Outlook = Overcast) $\vee$
(Outlook = Rain $\wedge$ Wind = Weak) $\vee$
(Outlook = Sunny $\wedge$ Humidity = Normal)

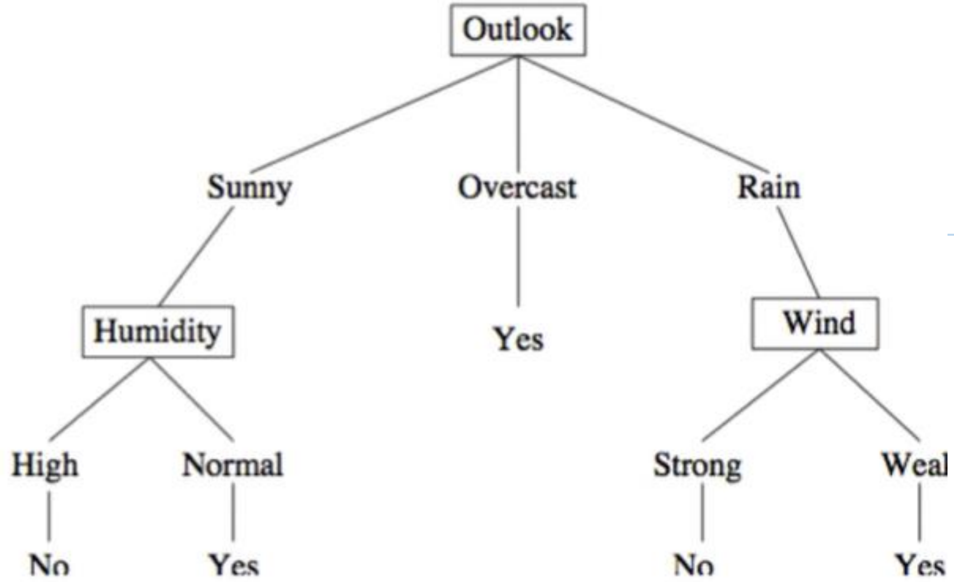


Figure credit: Tom Mitchell, 1997

Other measures?

Maximize the Gain

Entropy	Classification	$\sum_{i=1}^{C} -f_i log(f_i)$	f_i is the frequency of label i at a node and C is the number of unique labels.
Gini impurity	Classification	$\sum_{i=1}^{C} f_i(1-f_i)$	f_i is the frequency of label i at a node and C is the number of unique labels.
Variance	Regression	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2$	y_i is label for an instance, N is the number of instances and μ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$.

Minimize the variance

Other measures: Gini Index

- The original CART algorithm uses the GINI Impurity, whereas ID3 uses Entropy or Information Gain.

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

- p_i is the proportion of samples belonging to class i.

- While building the decision tree, we would prefer to choose the features with the **least Gini index** as the root node.

Decision trees and overfitting

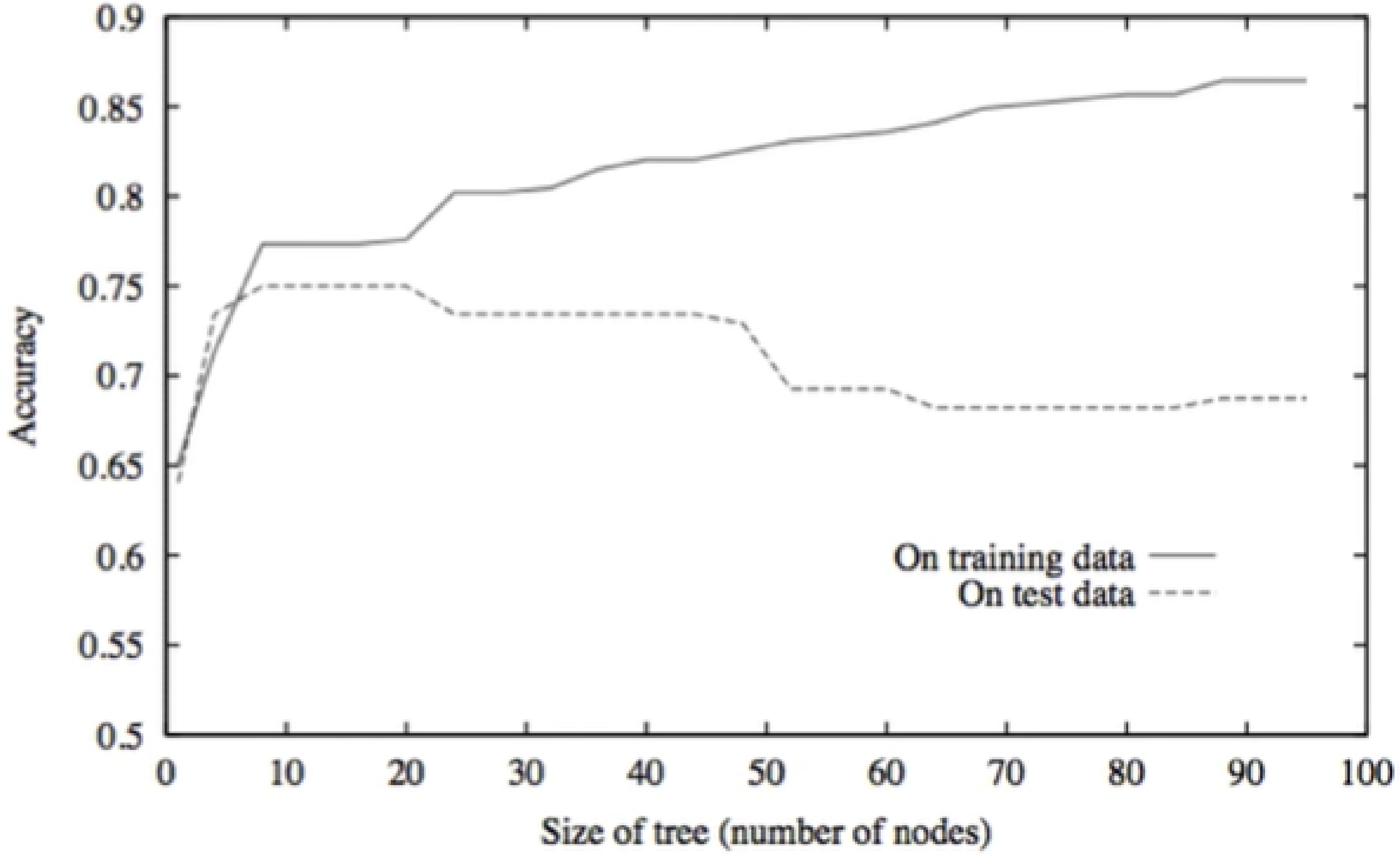
UNIVERSITÀ di VERONA

- Decision trees are **non-parametric models** with a structure that is determined by the data.
- As a result, they are flexible and can easily fit the training set, with a high risk of **overfitting.**

- Standard techniques to improve generalization apply also to decision trees (early stopping, regularization, data augmentation, complexity reduction, <u>ensembling</u>).

- A technique to reduce complexity a posteriori is called **pruning** (an operation that is applied after tree building).

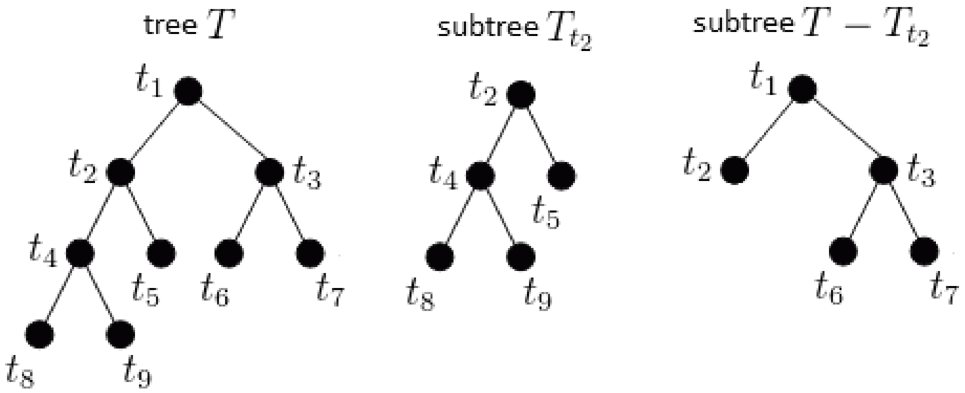
Decision trees and overfitting



- Early stopping!
- Pruning!

Figure credit: Tom Mitchell, 1997

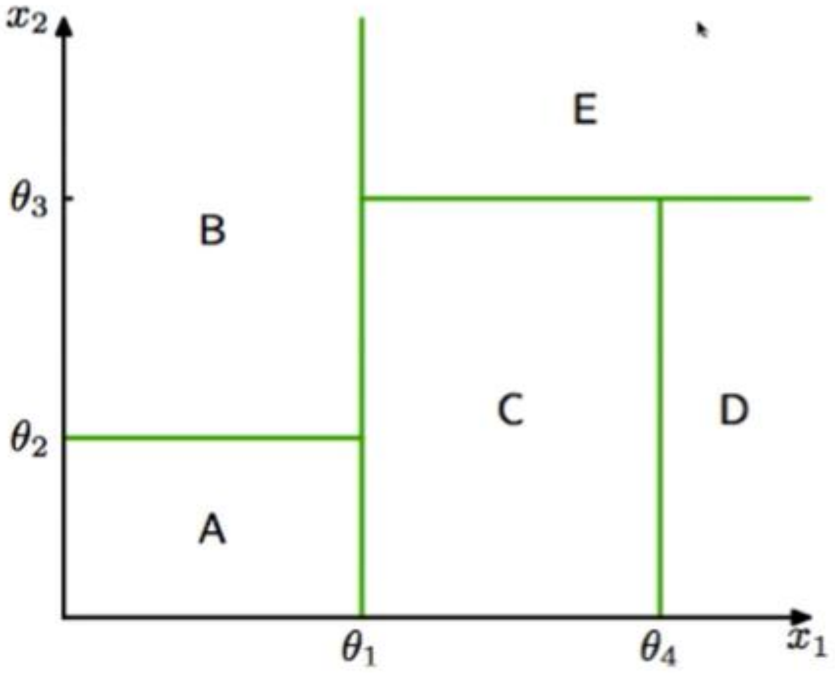
Pruning (extra)



- First, we build a decision tree until each leaf node is pure or until a predefined stopping criterion (e.g., maximum depth) is met.
- Then, starting from the bottom of the tree, we evaluate each node to see if pruning it would improve the tree's performance on a validation set.
- Define a cost-complexity measure that considers both the accuracy of the node and the size of the subtree rooted at that node.
- Identify the node with the smallest cost-complexity measure. Prune the subtree rooted at that node. This means turning that node into a leaf labelled with the majority class of the samples in that node.
- Repeat the process of evaluating and pruning nodes until a suitable tree is achieved or until further pruning does not significantly improve performance.

Decision trees- continuous attributes

- So far, we have investigated *categorical values*, but you can also use decision trees for *continuous attributes.*
- Pick a threshold to create a split! (e.g., temperature > 77.8)= true



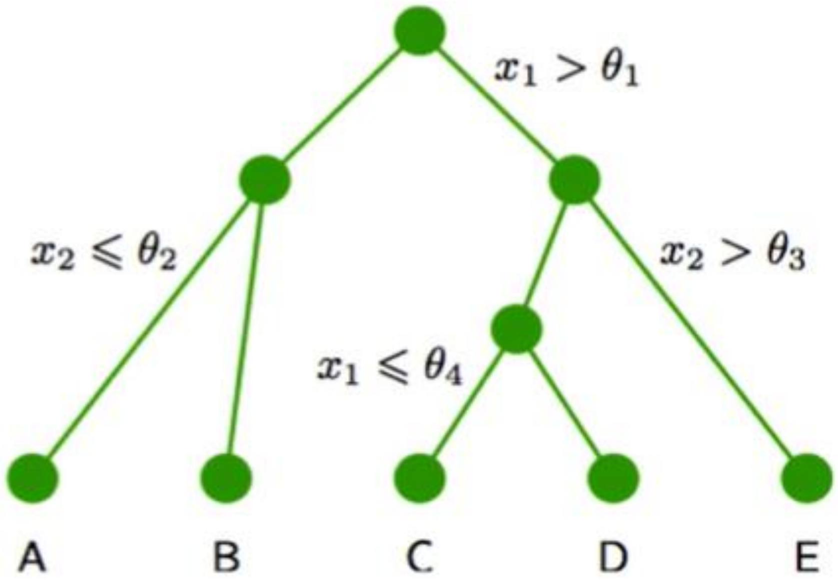


Figure credit: Chris Bishop, PRML

Decision trees- Multi-class classification

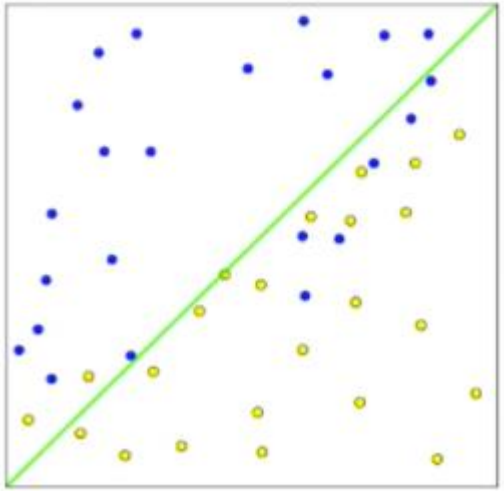
- Up until this point, we observed binary trees.
- Multi-class classification (i.e., *k* different classes)
  - Predict most frequent class in the subset
  - Entropy: $H(S) = -\Sigma_c \, p_{(c)} \log_2 p_{(c)}$
  - $p_{(c)}$= the proportion of the examples of class *c* in subset *S*

Decision trees- Pros & Cons

- Pros:
  - Interpretable: humans can understand the reason of decision
  - Easily handles irrelevant data (Gain=0)
  - Can handle missing data
  - Very compact: *#nodes* << *#training data after pruning*
  - Very fast at testing time: O(depth), *depth* << *#training data*

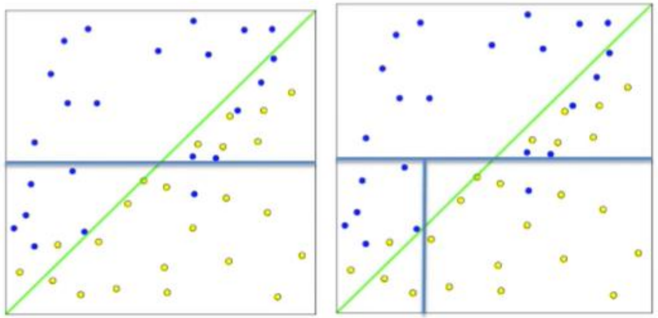
Decision trees- Pros & Cons

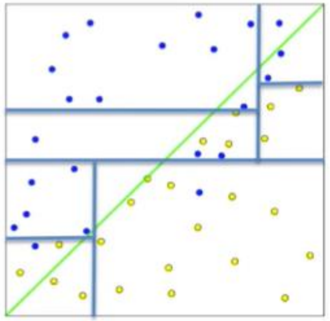
- Cons.
  - Greedy (may not find best tree)– not globally optimal!
    - Exponentially many possible trees– NP hard!
  - Only axis-aligned splits of data (especially in continuous data)



Decision boundary of a linear classifier

A decision tree can never find such a decision boundary



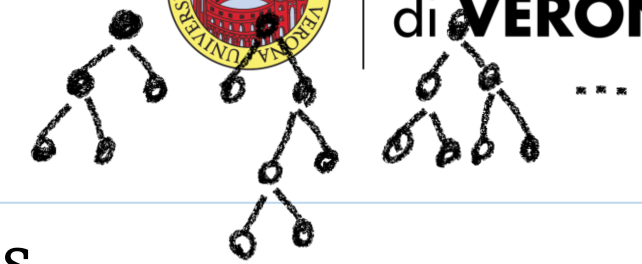


In decision trees, there is no diagonal cut!!!

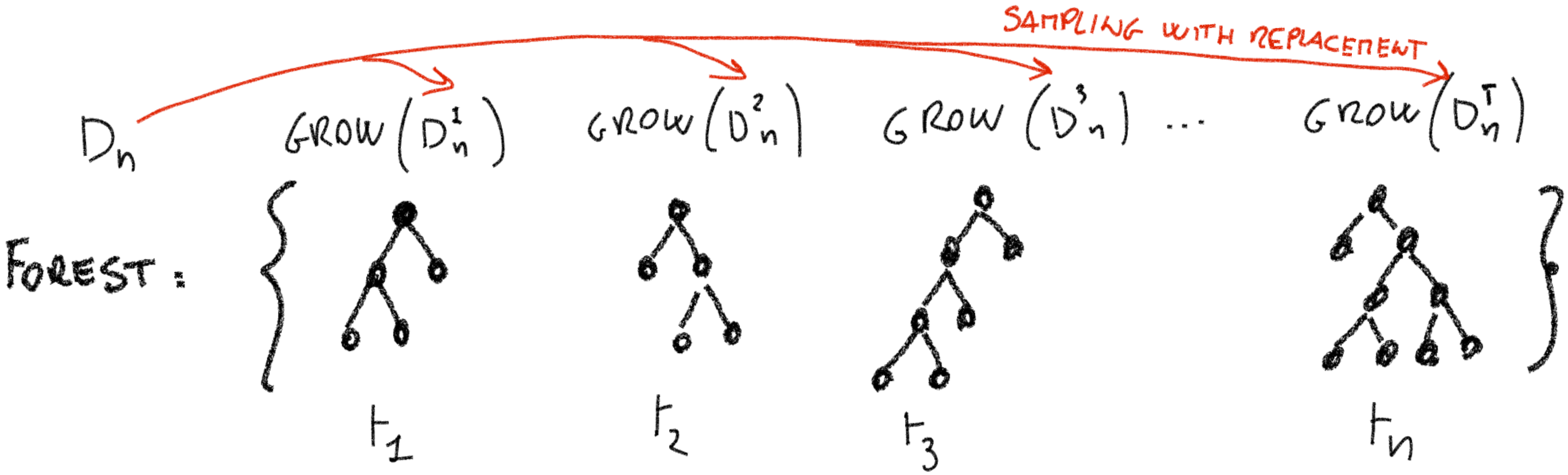
Decision trees-- summary

- ID3: grows decision tree from the root to down
  - Greedily selects next best attribute using INF. GAIN
  - Entropy: How uncertain we are in terms of Yes/No in a set
  - Inf. Gain: reduction in uncertainty following a split
- Searches a complete hypothesis space
- Prefers smaller trees, high gain at the root
- Overfitting addressed by post-pruning
  - Prune nodes, while accuracy increases on validation set
- Fast, compact, interpretable

RANDOM FOREST



- Random forests are ensembles of decision trees.
- Each tree is typically trained on a bootstrapped version of the training set (sampled with replacement).



Random Forest

- Split functions are optimized on randomly sampled features or are sampled completely at random (extremely randomized trees).
  - This helps obtaining decorrelated decision trees
- The final prediction of the forest is obtained by averaging the prediction of each tree in the ensemble $\mathcal{Q} = \{t_1, \ldots, t_T\}$

$$f_{\mathcal{Q}}(x) = \frac{1}{T} \sum_{j=1}^{T} f_t(x)$$

Average of T number of decision trees

Ensemble Methods

- The concept of combining different models to obtain more accurate predictions falls under the umbrella of 'ensemble methods' or ensemble learning.
- There are different types, including
  - Bagging,
  - Boosting, and
  - Stacking.

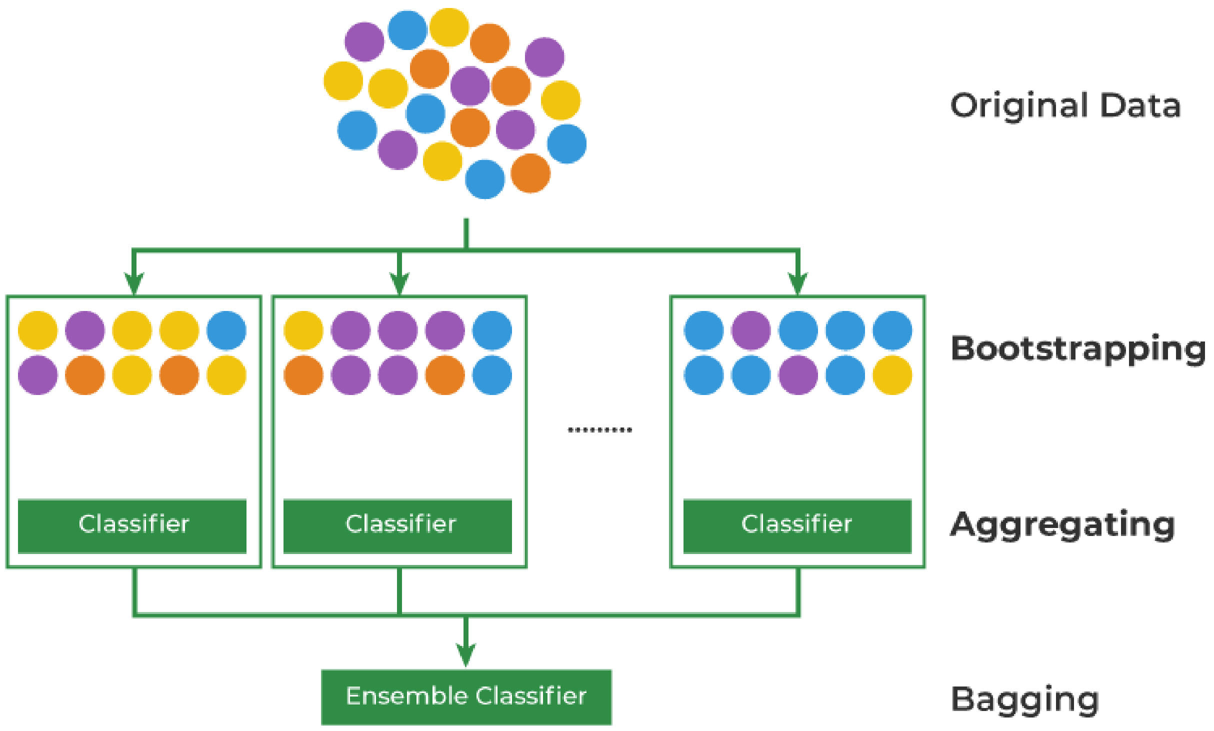
Ensemble Methods

UNIVERSITÀ di VERONA

- Many times ensembles are made with many 'weak learners', rather than having a few complex models.
  - **Weak Learner:** is a classification model that performs slightly better than random guessing.
    - For instance, for a binary classification task with balanced classes, this can be a classifier performing slightly better than 50% accuracy.
    - E.g., k-Nearest Neighbors, with k=1 operating on one or a subset of input variables.
    - Or a decision tree with a single node operating on one feature, the output of which makes a prediction directly.

Ensemble Methods

- **Bagging**, or **bootstrapping**, is a technique in which we train **N models** on **N different samplings** of our dataset.
- This way, all models will have different predictions; we will average them in the end to obtain a single value.



Boosting

- Trained models are <u>weighted</u> based on their performance.

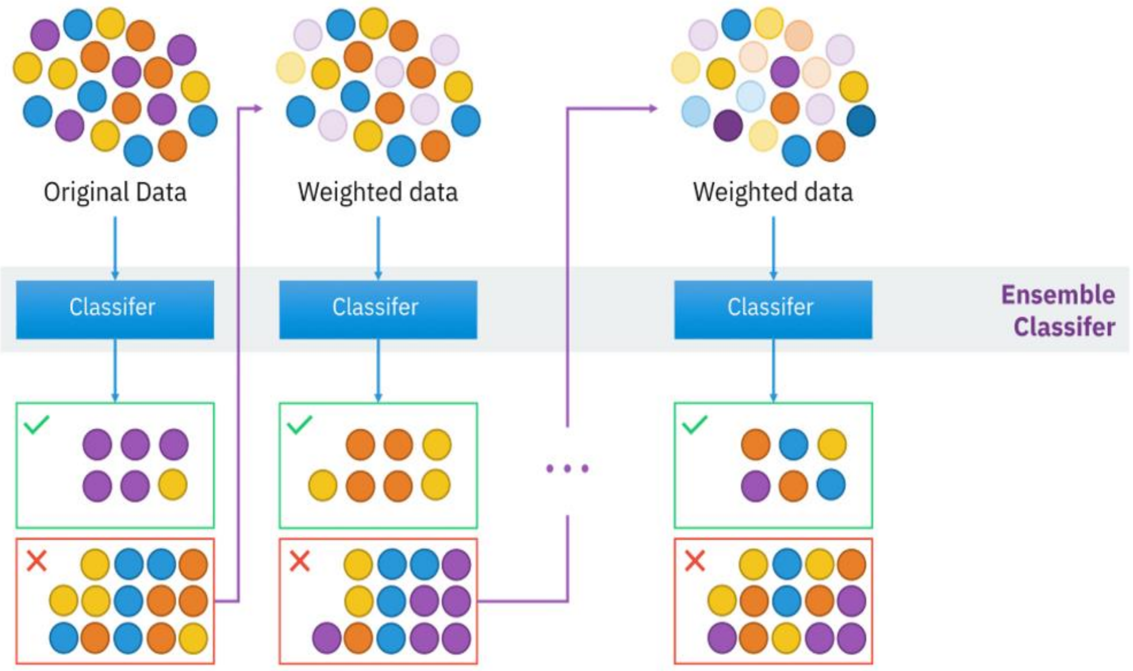
- The weak classifiers are learned sequentially.

- The more the previous classifier makes mistakes on a dataset, the more they are weighted for the next one.

- One example method is called **AdaBoost** (Adaptive Boosting).

- **AdaBoost** works by weighting the instances in the training dataset based on the accuracy of previous classifications.

AdaBoost



Step 1: Use all training data and assign equal weights to each training sample to build a weak classifier. Check which training samples are correctly classified and which are not correctly classified.
Step 2: Assign more weights to the incorrectly classified samples such that classifying them correctly becomes more important.
Step 3: Re-iterate Step 2 until all the data points have been correctly classified or a maximum iteration number is reached.

Stacking

- Combines the predictions of multiple ML models (*called base models*) to improve the overall performance.

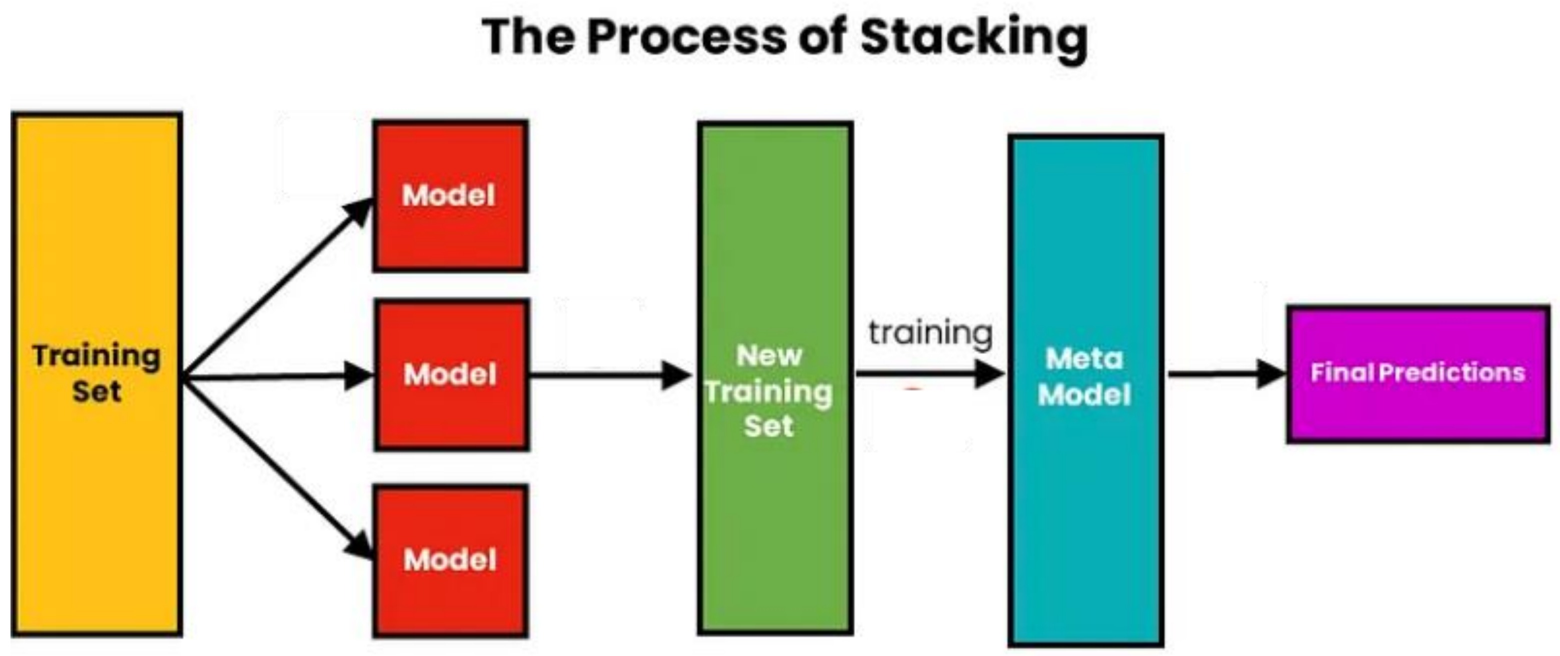
- Different from boosting and bagging, the used base models are <u>heterogenous</u>.

- Can be used for both **regression** and **classification** problems.

- A powerful ensemble learning algorithm that can often improve the performance of individual models.

Stacking

- The basic idea of stacking is to first train a set of base models on the same dataset. The predictions of these base models are then used to train a **meta-model**, which is a higher-level model that learns how to combine the predictions of the base models to make a better prediction.



Cigdem Beyan
cigdem.beyan@univr.it
https://cbeyan.github.io/