Università degli Studi di Padova

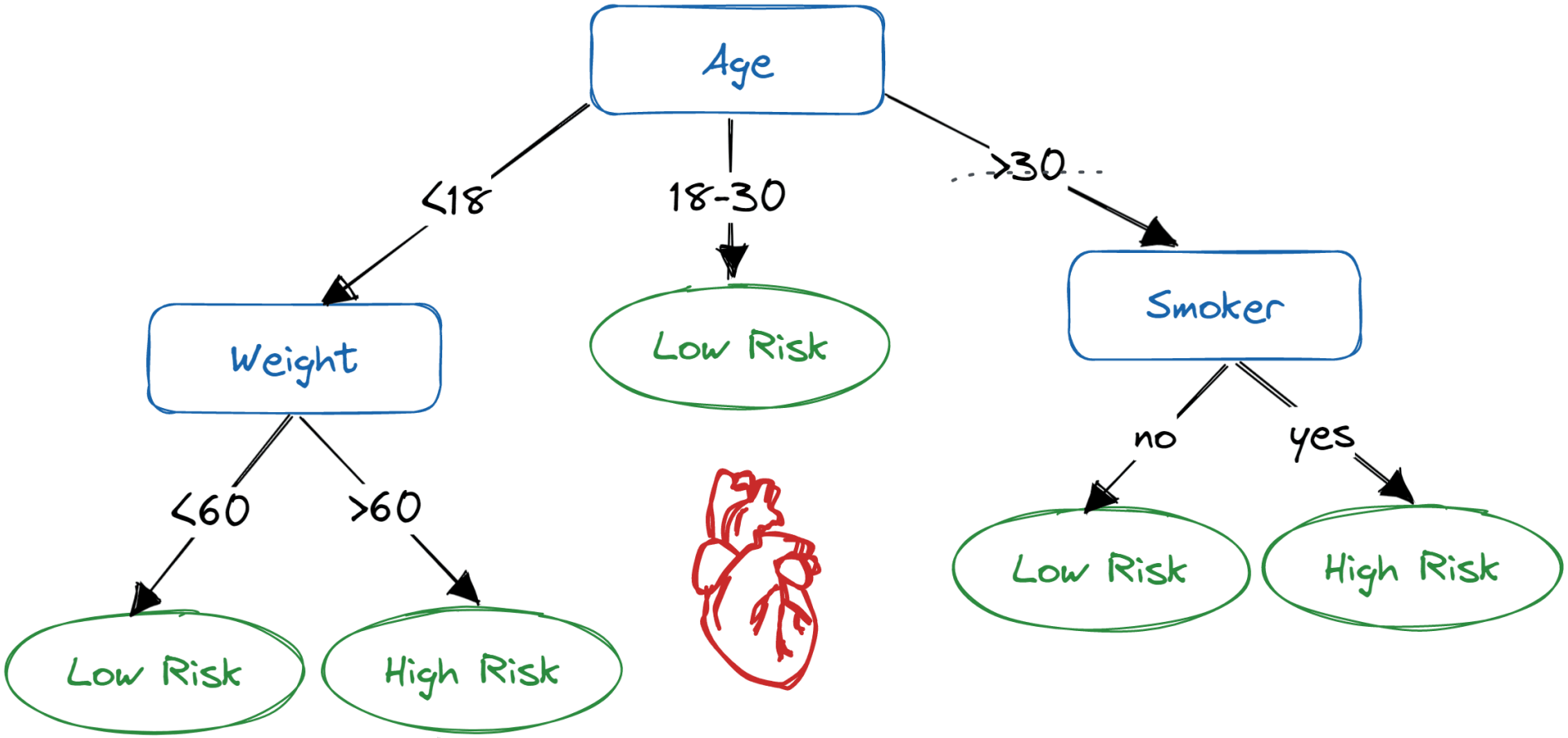
Machine Learning
2024/2025

AMCO
Artificial Intelligence, Machine Learning and Control Research Group

Lecture #16
Decision Trees & Random Forests

Gian Antonio Susto

Let's consider this 'decision-making' example



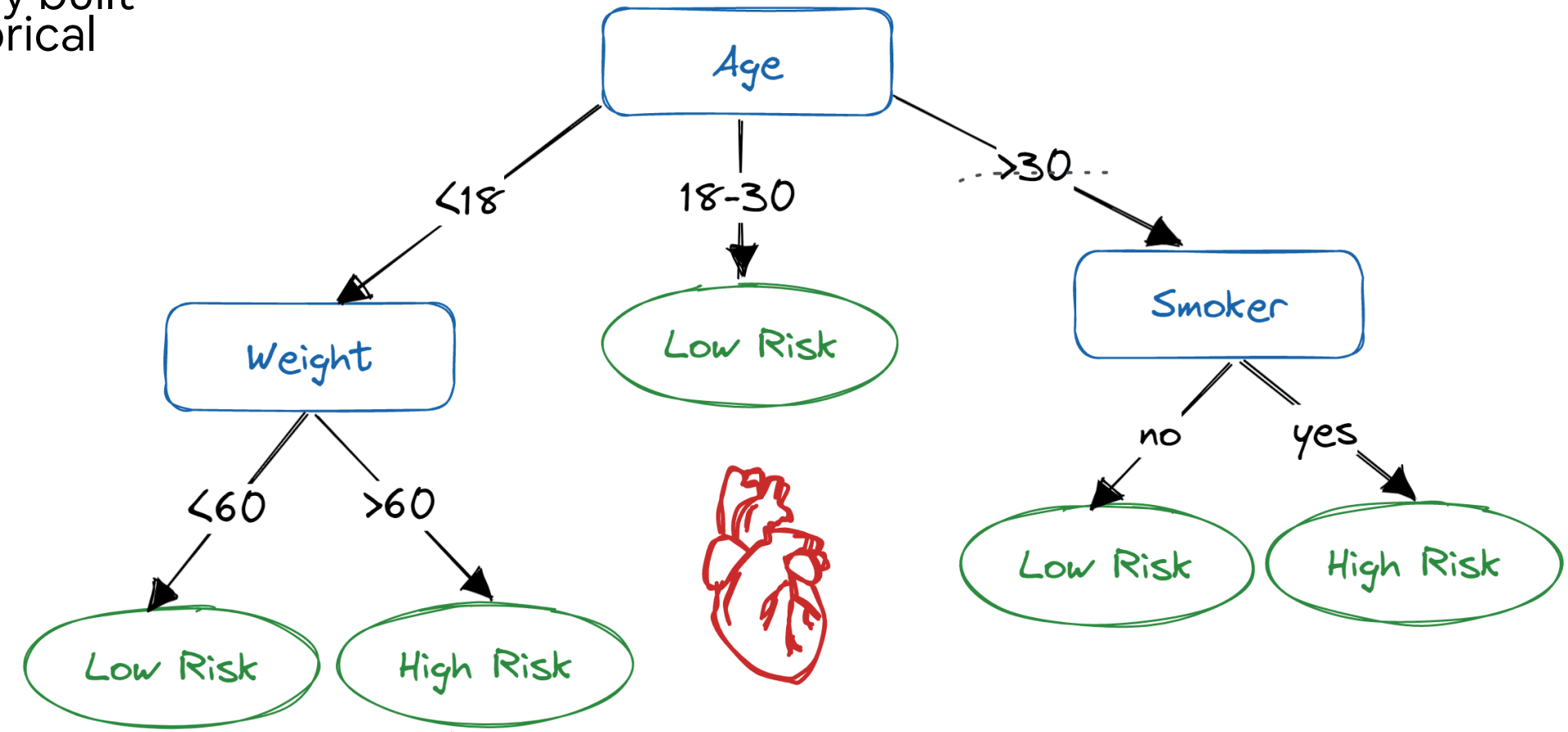
Understanding the risks to prevent a heart attack.

Let's consider this 'decision-making' example

This was probably built on historical data!

X = ?

Y = ?



Understanding the risks to prevent a heart attack.

Let's consider this 'decision-r

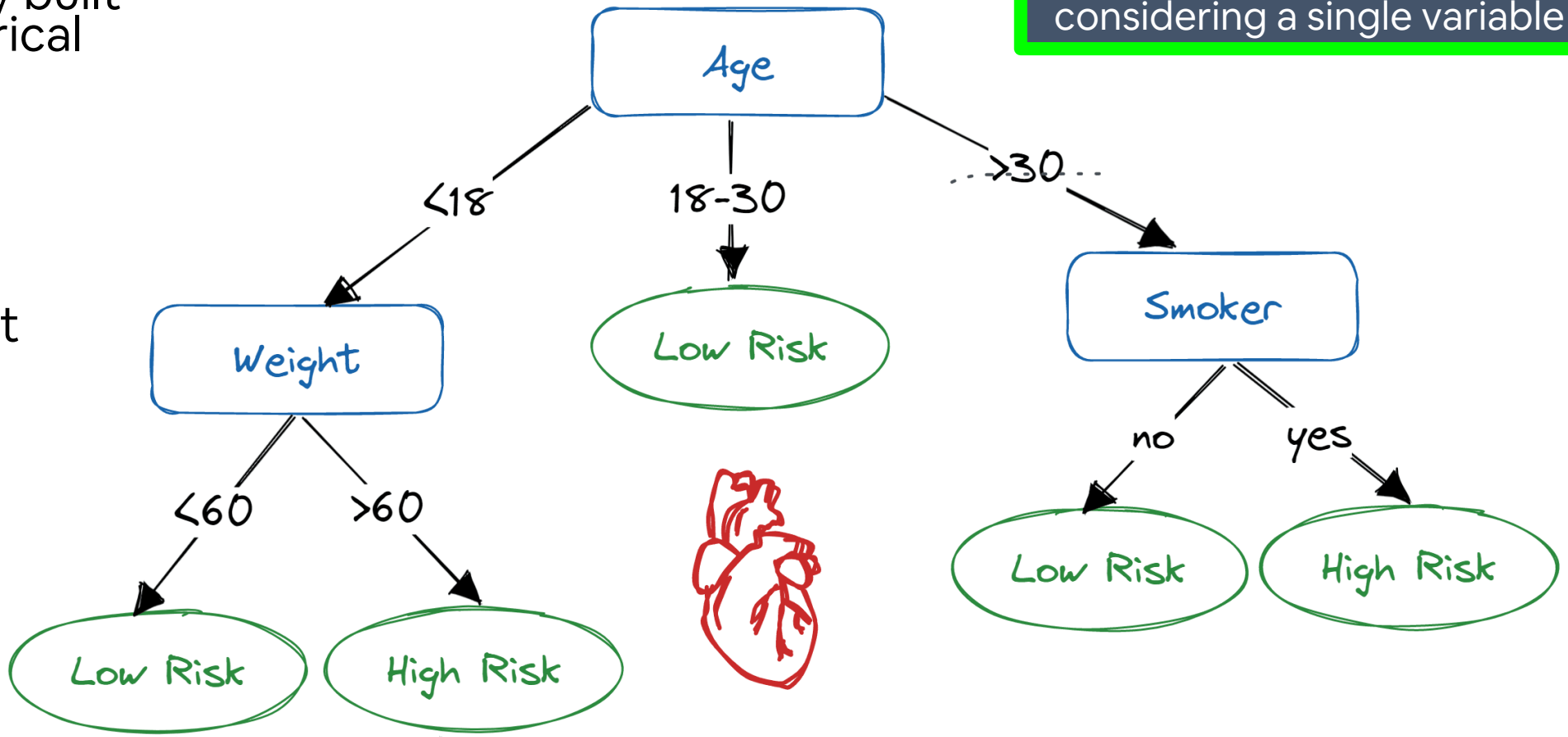
We are dealing with a multivariate supervised classification problem!

However, each 'decisions' are made by considering a single variable at each time!

This was probably built on historical data!

X = Age, Weight, Smoker

Y = Heart Attack



Understanding the risks to prevent a heart attack.

Let's consider this 'decision-r

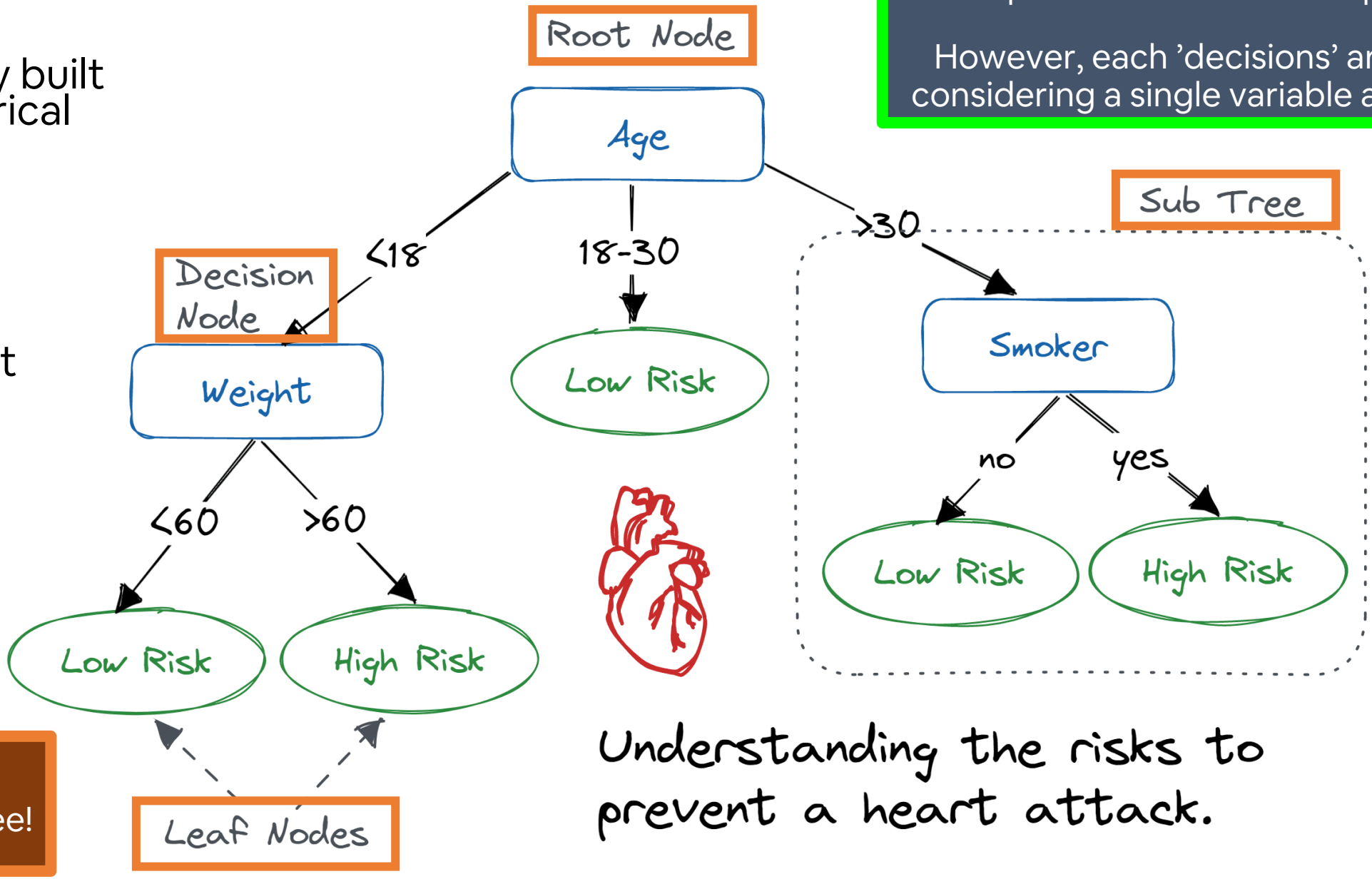
We are dealing with a multivariate supervised classification problem!

However, each 'decisions' are made by considering a single variable at each time!

This was probably built on historical data!

X = Age, Weight, Smoker

Y = Heart Attack



This is a Decision Tree!

Understanding the risks to prevent a heart attack.

🌳 Tree-based Approaches

Tree-based methods are among the most effective techniques for supervised learning, particularly when working with smaller datasets (with *n* fewer than 10,000 samples).

Interestingly, the core concepts behind them are quite straightforward...

Why do tree-based models still outperform deep learning on typical tabular data? Part of Advances in Neural Information Processing Systems 35 (NeurIPS 2022)

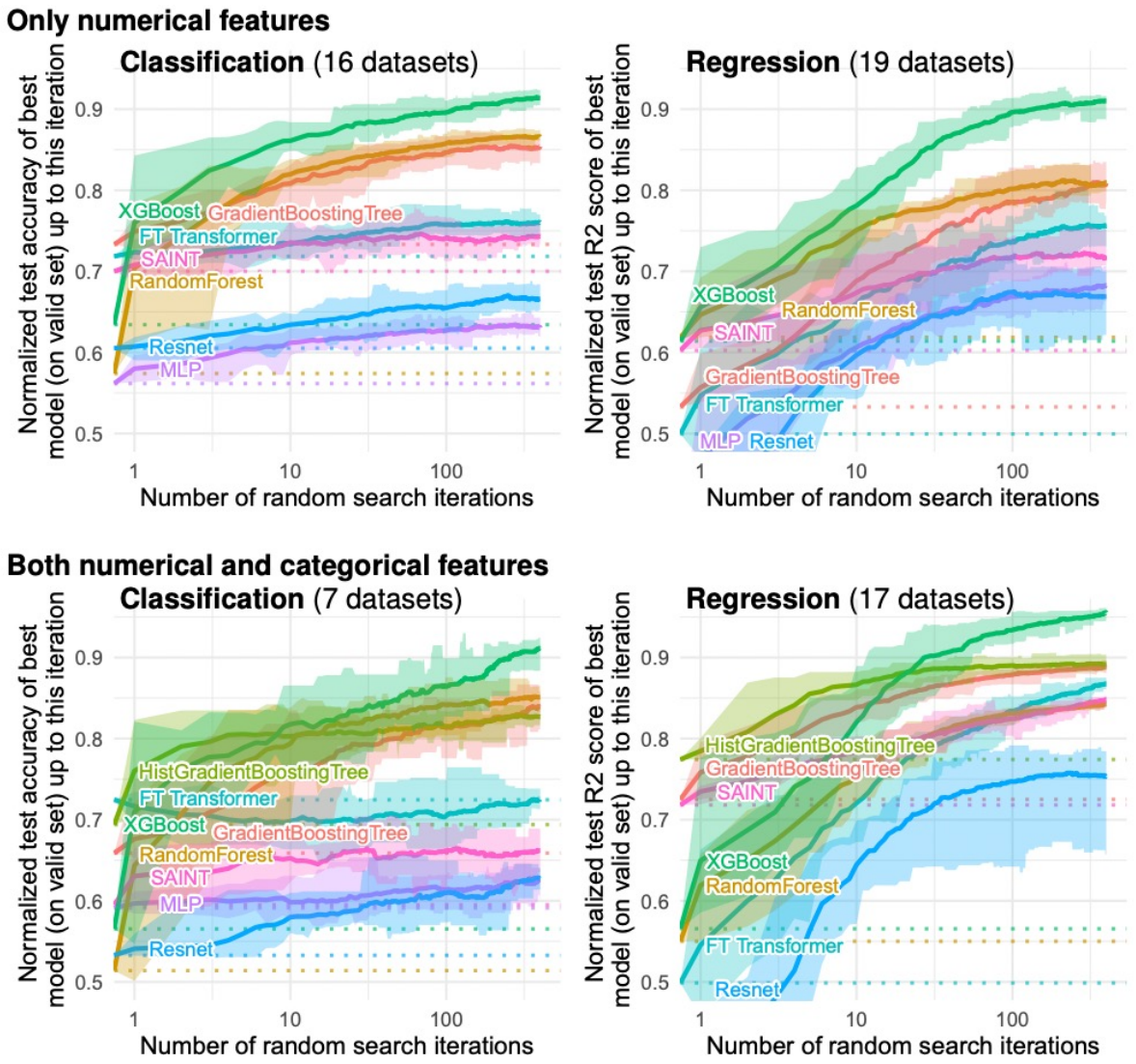
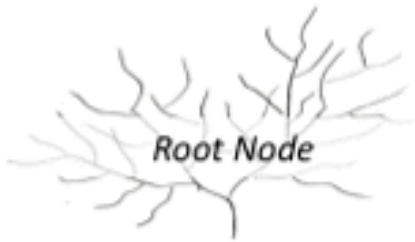
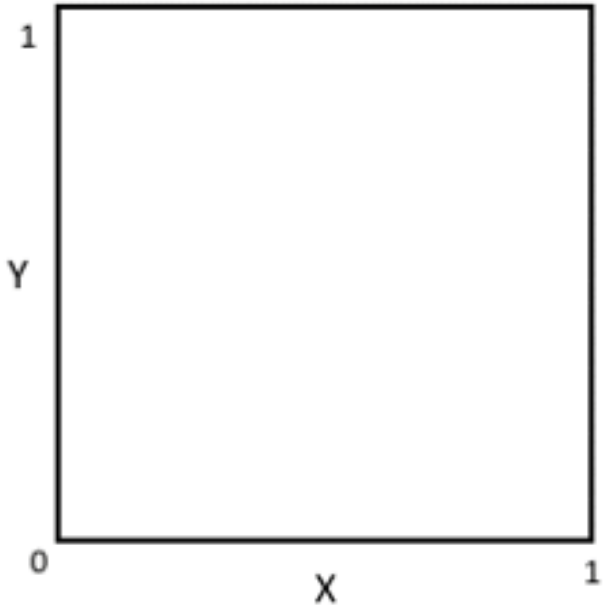


Figure 1: **Benchmark on medium-sized datasets**, top only numerical features; bottom: all features. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to minimum and maximum scores on these 15 shuffles.

🌳 Tree-based Approaches

Our goal is to find ‘decisions’ — that is, splitting points defined by pairs of (variable, value) — which allow us to partition the data into distinct scenarios where one class becomes dominant or more easily predictable.





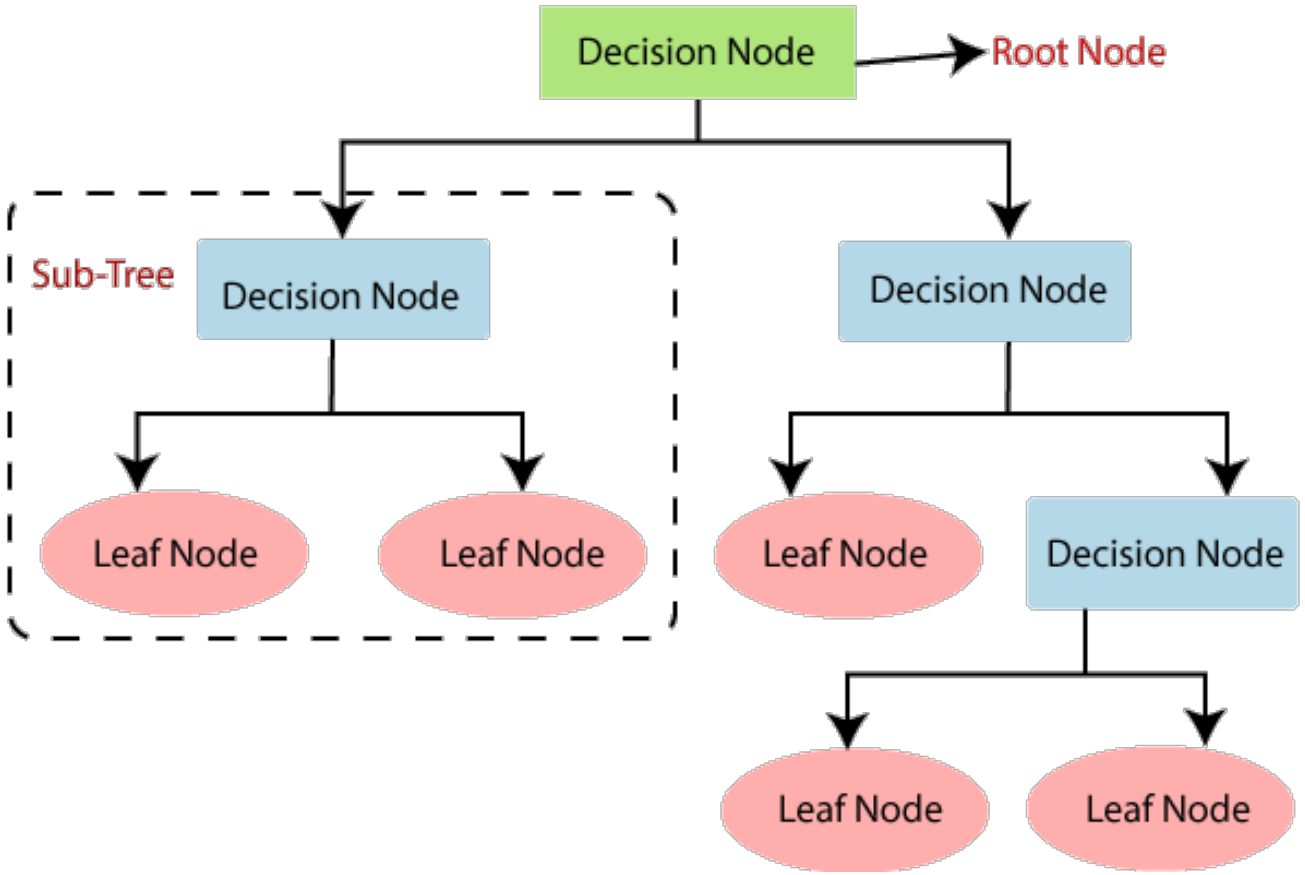
The result is a tree-like structure where:

- Internal nodes represent decision rules based on features.
- Leaf nodes represent predictions — typically the most frequent class

The Decision Tree

At the heart is the decision tree, a structure that mimics human decision-making by splitting data into branches based on feature values.

Each internal node of the tree represents a decision based on a feature, each branch represents the outcome of that decision, and each leaf node corresponds to a prediction or outcome.



Decision Tree (DT): how do we build one?

- We will investigate the following dataset: we aim from historical behaviour of our sparring tennis partner to predict if he will play tennis or not with us, based on the weather

n = 15

p = 4

Only categorical variable

Day (sample)	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
#1	Sunny	Hot	High	Weak	No
#2	Sunny	Hot	High	Strong	No
#3	Overcast	Hot	High	Weak	Yes
#4	Rain	Mild	High	Weak	Yes
#5	Rain	Cool	Normal	Weak	Yes
#6	Rain	Cool	Normal	Strong	No
#7	Overcast	Cool	Normal	Strong	Yes
#8	Sunny	Mild	High	Weak	No
#9	Sunny	Cool	Normal	Weak	Yes
#10	Rain	Mild	Normal	Weak	Yes
#11	Sunny	Mild	Normal	Strong	Yes
#12	Overcast	Mild	High	Strong	Yes
#13	Overcast	Hot	Normal	Weak	Yes
#14	Rainy	Mild	High	Strong	No

From: 'Machine Learning', Mitchell 1997

Decision Tree (DT): how do we build one?

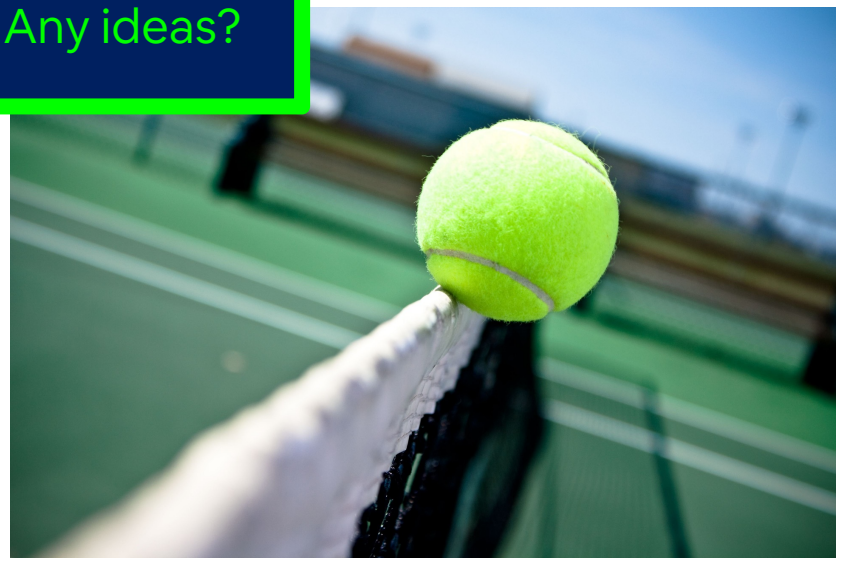
We will investigate the following dataset: we aim from historical behaviour of our sparring tennis partner to predict if he will play tennis or not with us, based on the weather

n = 15

p = 4

Only categorical variable

Any ideas?



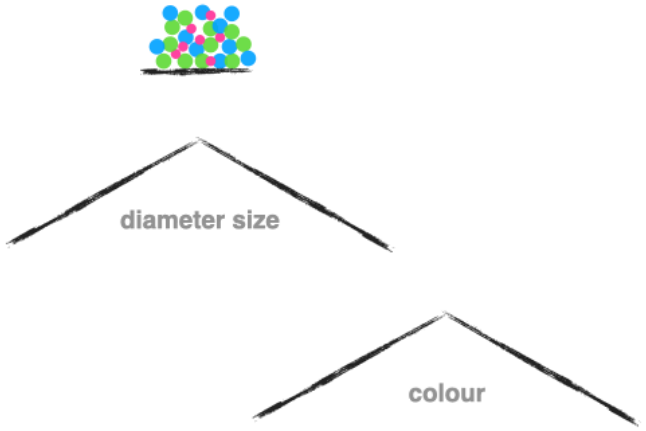
Day (sample)	Outlook (x_1)	Temperature (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
#1	Sunny	Hot	High	Weak	No
#2	Sunny	Hot	High	Strong	No
#3	Overcast	Hot	High	Weak	Yes
#4	Rain	Mild	High	Weak	Yes
#5	Rain	Cool	Normal	Weak	Yes
#6	Rain	Cool	Normal	Strong	No
#7	Overcast	Cool	Normal	Strong	Yes
#8	Sunny	Mild	High	Weak	No
#9	Sunny	Cool	Normal	Weak	Yes
#10	Rain	Mild	Normal	Weak	Yes
#11	Sunny	Mild	Normal	Strong	Yes
#12	Overcast	Mild	High	Strong	Yes
#13	Overcast	Hot	Normal	Weak	Yes
#14	Rainy	Mild	High	Strong	No

From: 'Machine Learning', Mitchell 1997

Decision Tree (DT): how do we build one?

We will use a ‘recursive’ procedure:

- We start building a tree from the root
- We choose the variable to be associated to the decision based on the one that better ‘simplifies’* the problem: a scenario where we have a dominant/only class
- We iterate this, until classes are separated or until we reach a given ‘depth’ of the tree

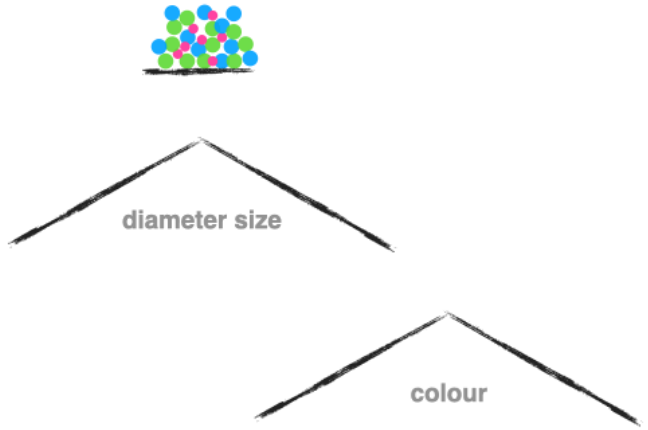


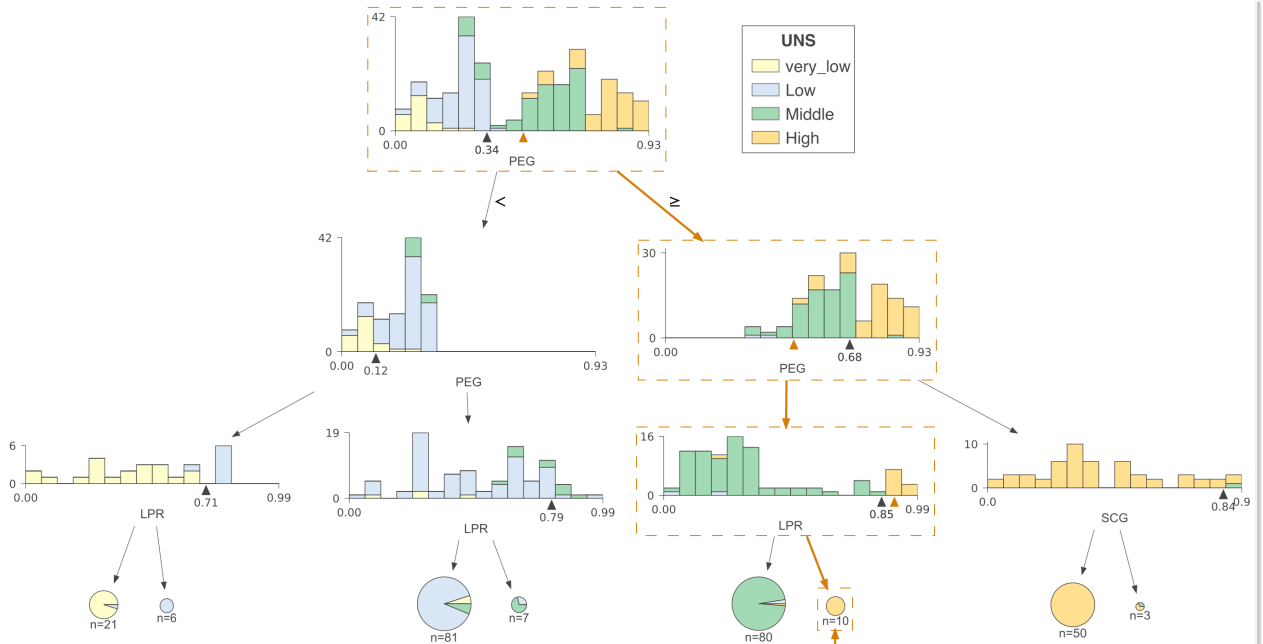
Decision Tree (DT): how do we build one?

We will use a ‘recursive’ procedure:

- We start building a tree from the root
- We choose the variable to be associated to the decision based on the one that better ‘simplifies’* the problem: a scenario where we have a dominant/only class
- We iterate this, until classes are separated or until we reach a given ‘depth’ of the tree

*We need a quantitative metric to define how ‘simple’ a decision is at a leaf level!





Entropy (Measure of Impurity / Uncertainty)

Information Gain measures the decrease in entropy after the dataset is split on an attribute. It tells us how much "information" a feature gives us about the target variable.

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

Where:

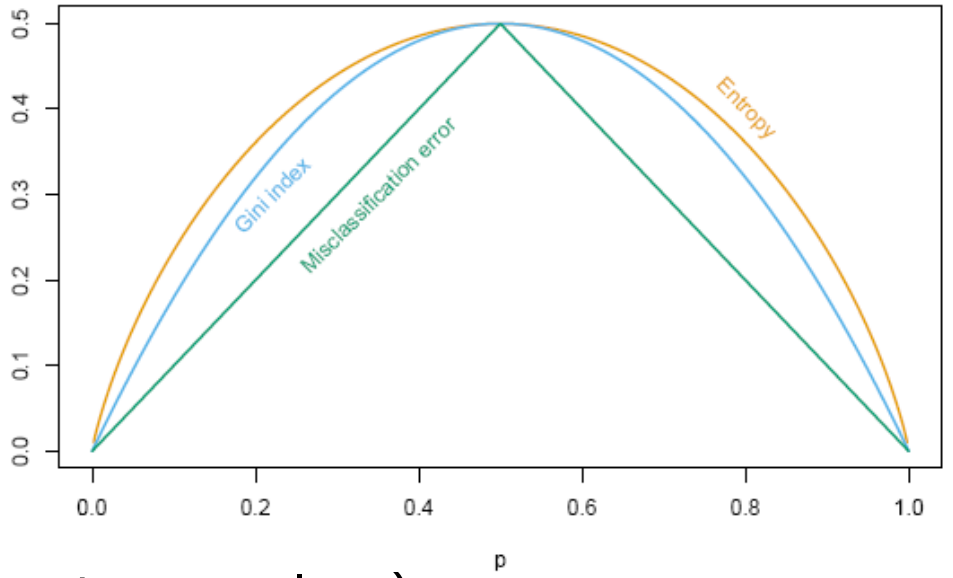
- S is the original dataset
- A is the attribute we split on
- v are the possible values of A
- S_v is the subset of S where $A = v$

Interpretation:

- High IG: splitting on this attribute significantly reduces uncertainty → it's a good choice!
- Low IG: the attribute doesn't help much in separating the classes.

Information Gain (Reduction in Entropy After a Split)

Entropy quantifies the uncertainty or impurity in a dataset. In the context of classification, it tells us how mixed the class labels are in a set.



For a dataset S with c classes

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Where p_i is the proportion of samples in class i.

- Entropy = 0: the dataset is pure (all examples belong to one class)
- Higher entropy: more class mixing, greater uncertainty.

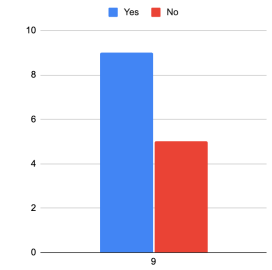
Example: our tennis dataset 🎾 has 9 ‘yes’ and 5 ‘no’

$$Entropy = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.94$$

Example

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

$$Entropy = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.94$$

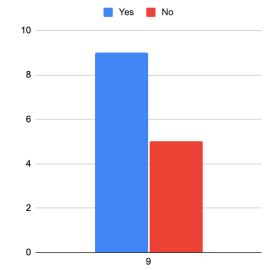


Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Example

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

$$Entropy = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.94$$

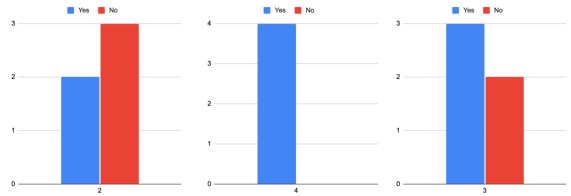


Outlook

Sunny (5) → 2 Yes, 3 No → Entropy = 0.971

Overcast (4) → 4 Yes → Entropy = 0

Rain (5) → 3 Yes, 2 No → Entropy = 0.971



$$E(Outlook) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.694$$

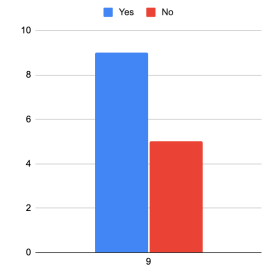
$$IG(Outlook) = 0.94 - 0.694 = 0.246$$

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Example

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

$$Entropy = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.94$$

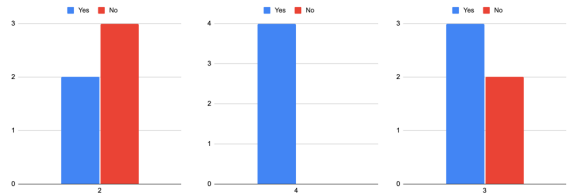


Outlook

Sunny (5) → 2 Yes, 3 No → Entropy = 0.971

Overcast (4) → 4 Yes → Entropy = 0

Rain (5) → 3 Yes, 2 No → Entropy = 0.971



$$E(Outlook) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.694$$

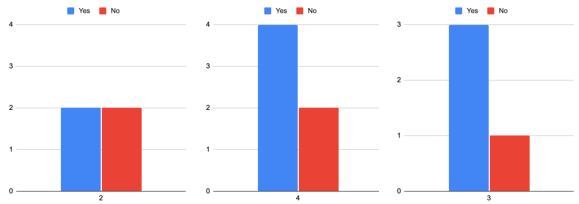
$$IG(Outlook) = 0.94 - 0.694 = 0.246$$

Temperature

Hot (4) → 2 Yes, 2 No → Entropy = 1.0

Mild (6) → 4 Yes, 2 No → Entropy = 0.918

Cool (4) → 3 Yes, 1 No → Entropy = 0.811



$$E(Temp) = \frac{4}{14} \cdot 1.0 + \frac{6}{14} \cdot 0.918 + \frac{4}{14} \cdot 0.811 \approx 0.911$$

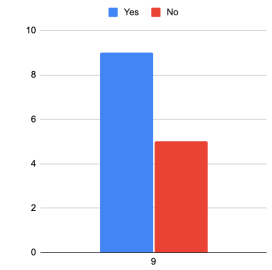
$$IG(Temp) = 0.94 - 0.911 = 0.029$$

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Example

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

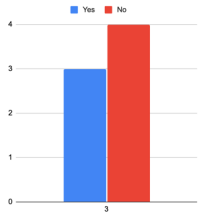
$$Entropy = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.94$$

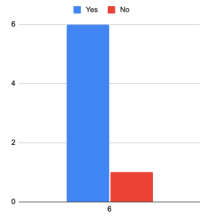


Humidity

High (7) → 3 Yes, 4 No → Entropy = 0.985

Normal (7) → 6 Yes, 1 No → Entropy = 0.592





$$E(Humidity) = \frac{7}{14} \cdot 0.985 + \frac{7}{14} \cdot 0.592 = 0.789$$

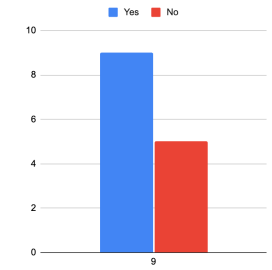
$$IG(Humidity) = 0.94 - 0.789 = 0.151$$

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Example

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

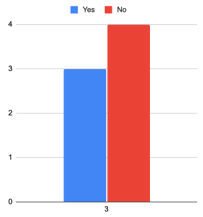
$$Entropy = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.94$$

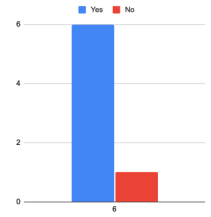


Humidity

High (7) → 3 Yes, 4 No → Entropy = 0.985

Normal (7) → 6 Yes, 1 No → Entropy = 0.592





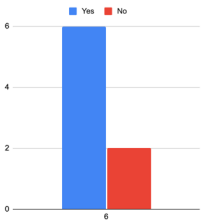
$$E(Humidity) = \frac{7}{14} \cdot 0.985 + \frac{7}{14} \cdot 0.592 = 0.789$$

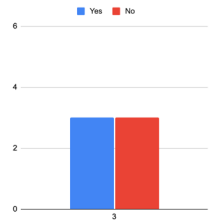
$$IG(Humidity) = 0.94 - 0.789 = 0.151$$

Wind

Weak (8) → 6 Yes, 2 No → Entropy = 0.811

Strong (6) → 3 Yes, 3 No → Entropy = 1.0



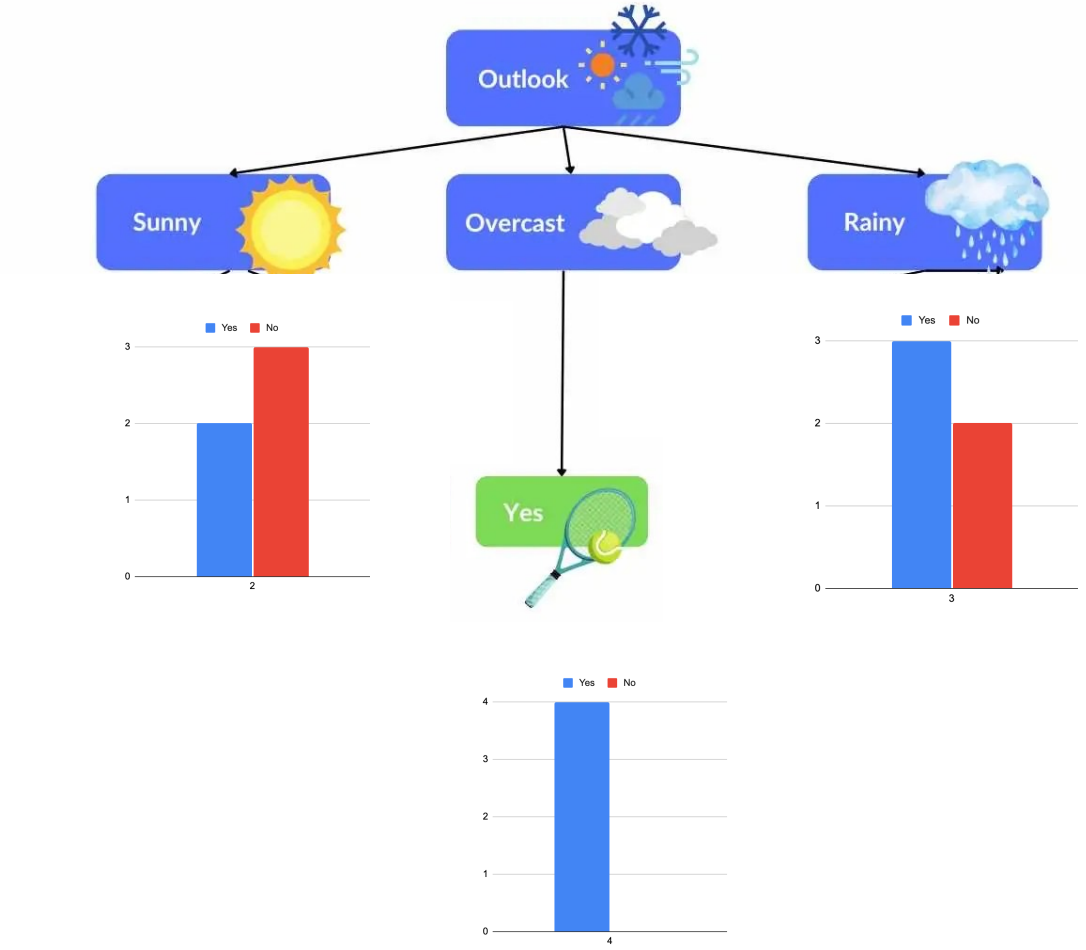


$$E(Wind) = \frac{8}{14} \cdot 0.811 + \frac{6}{14} \cdot 1.0 = 0.892$$

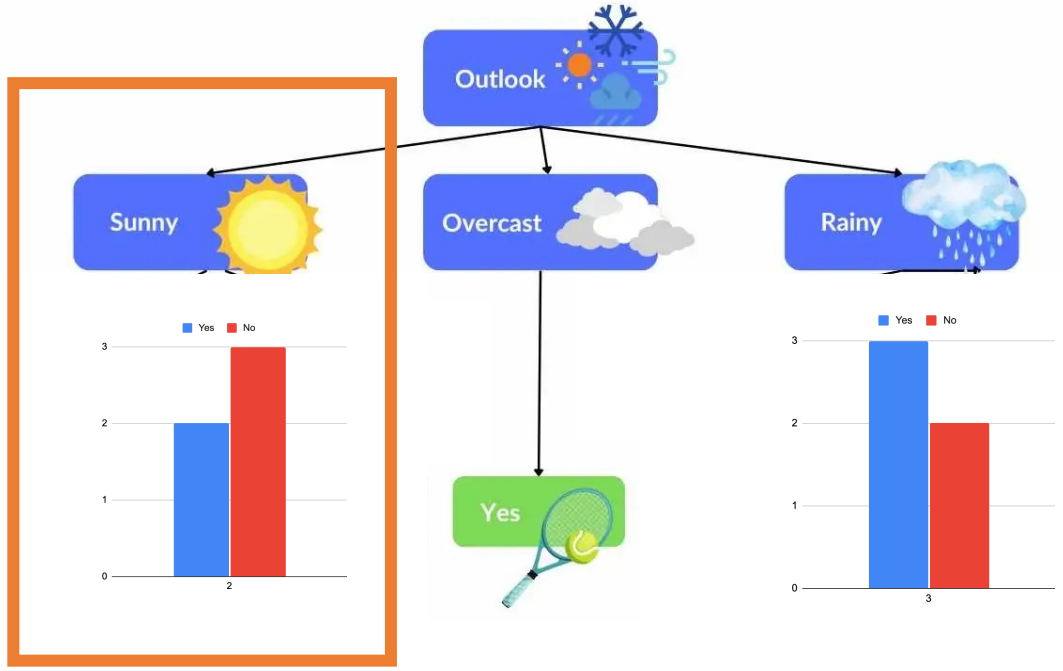
$$IG(Wind) = 0.94 - 0.892 = 0.048$$

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Example


Outlook
Sunny
Overcast
Rainy
Yes
Yes
No
3
2
1
0
2
Yes
No
3
2
1
0
3
Yes
Yes
No
4
3
2
1
0
4

🎾 Example



✅ 2 Yes, ❌ 3 No

Entropy ≈ 0.971 (as computed earlier)

<u>Humidity</u>

High (3) → 0 Yes, 3 No → Entropy = 0

Normal (2) → 2 Yes, 0 No → Entropy = 0

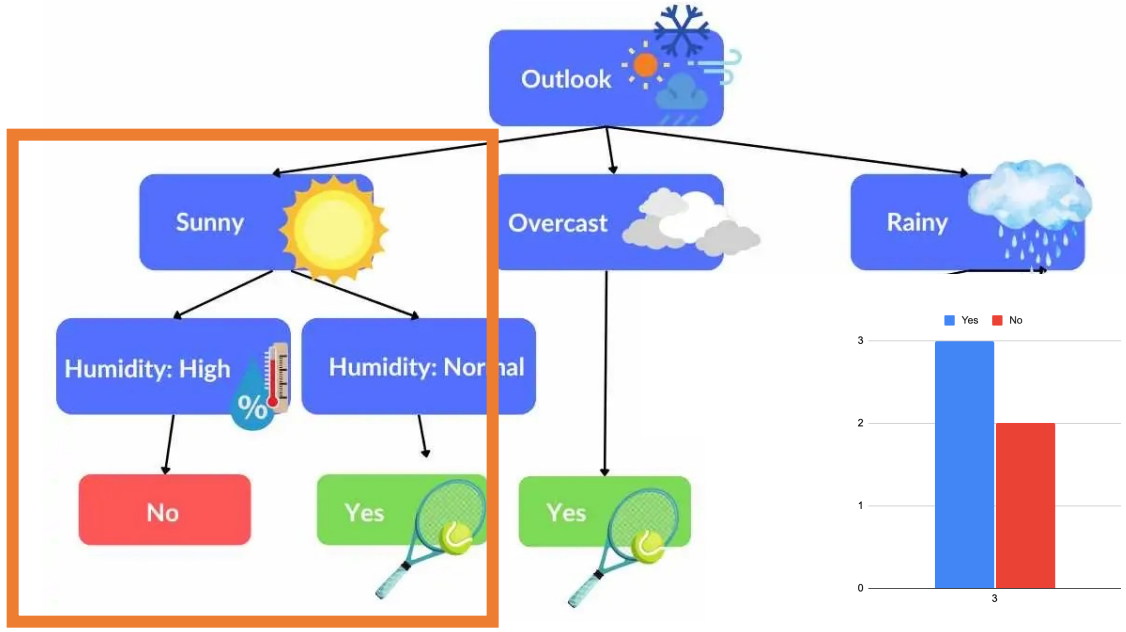
$$E(Humidity) = \frac{3}{5} \cdot 0 + \frac{2}{5} \cdot 0 = 0$$

$$IG(Humidity|Sunny) = 0.971 - 0 = 0.971$$

✅ Highest possible gain! We split on Humidity.

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

Example



✅ 2 Yes, ❌ 3 No

Entropy ≈ 0.971 (as computed earlier)

Humidity

High (3) → 0 Yes, 3 No → Entropy = 0

Normal (2) → 2 Yes, 0 No → Entropy = 0

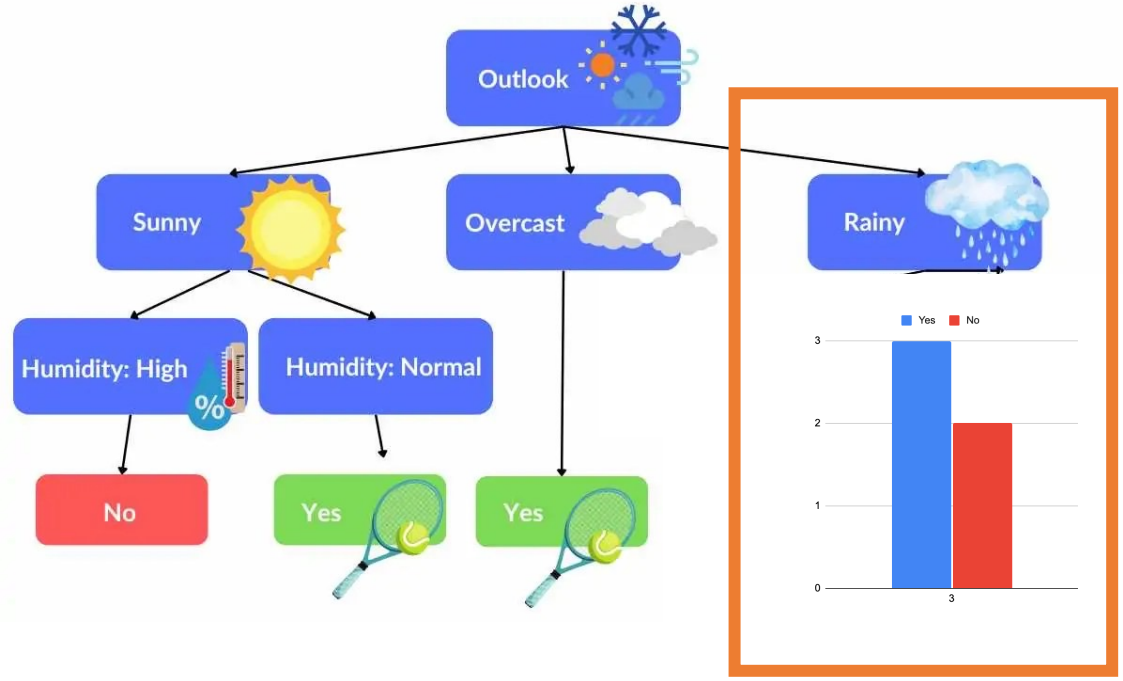
$$E(Humidity) = \frac{3}{5} \cdot 0 + \frac{2}{5} \cdot 0 = 0$$

$$IG(Humidity|Sunny) = 0.971 - 0 = 0.971$$

✅ Highest possible gain! We split on Humidity.

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

🎾 Example



✅ 3 Yes, ❌ 2 No

Entropy ≈ 0.971 (as computed earlier)

Wind

Weak (3) → 3 Yes, 0 No → Entropy = 0

Strong (2) → 0 Yes, 2 No → Entropy = 0

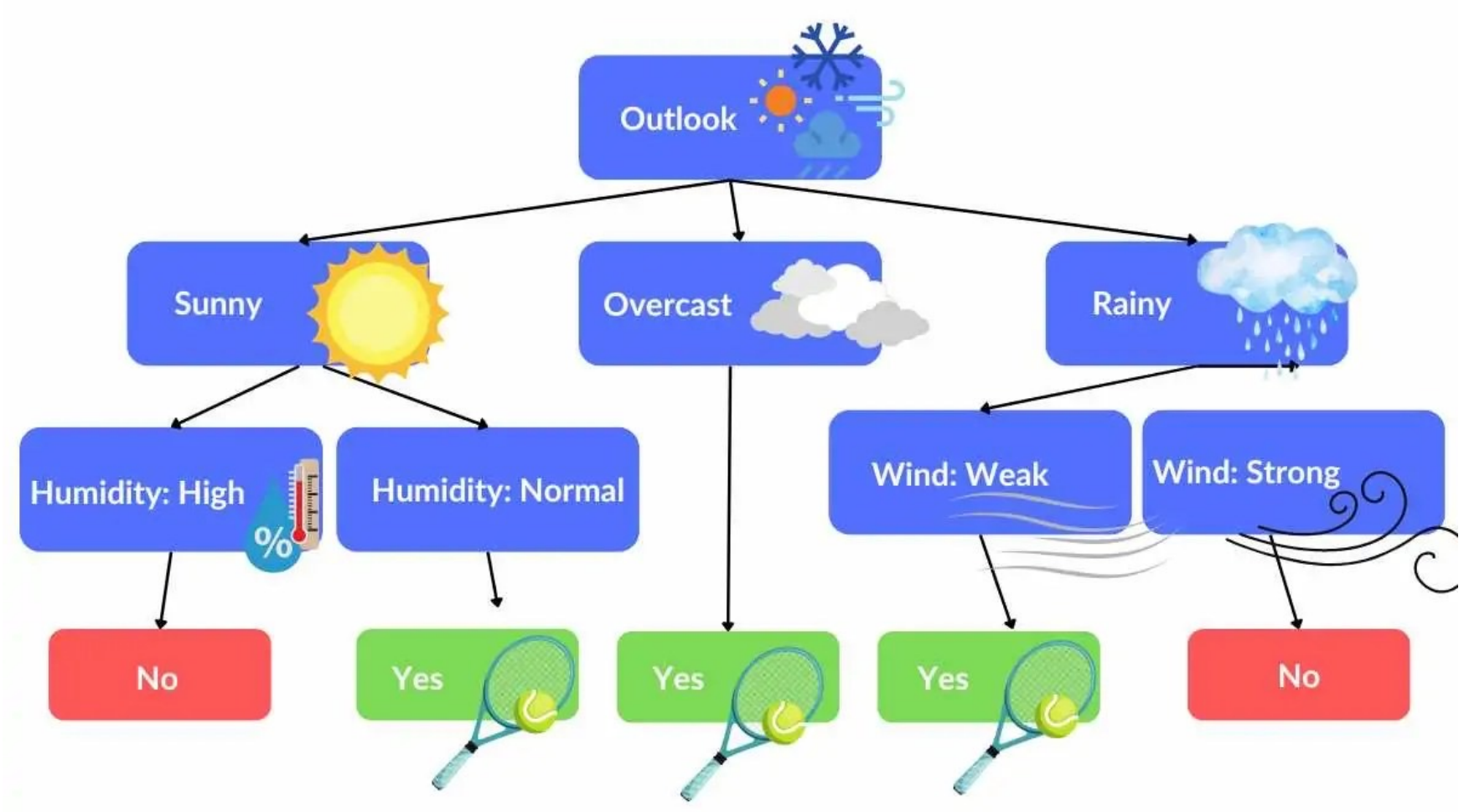
$$E(Wind) = \frac{3}{5} \cdot 0 + \frac{2}{5} \cdot 0 = 0$$

$$IG(Wind|Rain) = 0.971 - 0 = 0.971$$

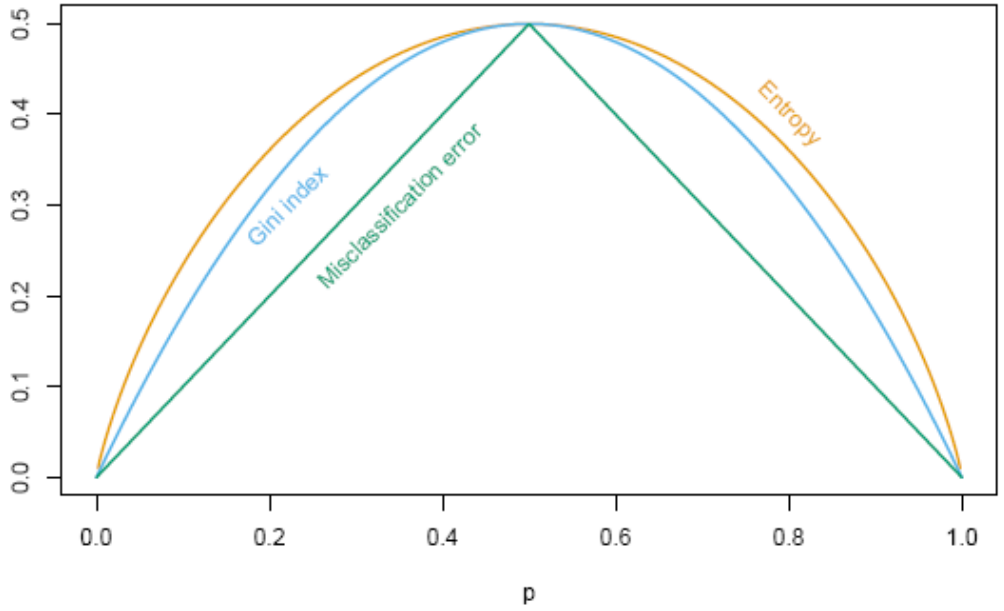
✅ Highest possible gain! We split on Wind.

Outlook (x_1)	Temp. (x_2)	Humidity (x_3)	Wind (x_4)	Play Tennis (y)
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Example


Outlook
Sunny
Overcast
Rainy
Humidity: High
Humidity: Normal
Wind: Weak
Wind: Strong
No
Yes
Yes
Yes
No

Another metric: Gini Index



The Gini Index (or Gini Impurity) is a measure of how impure or mixed a dataset is.

For a dataset S with c classes:

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

Where p_i is the proportion of examples in class i.

- Gini = 0 → Perfect purity (all examples belong to one class)
- Higher Gini → More mixed classes (more impurity)

Gini for a Split:

$$Gini_{split} = \frac{n_{left}}{n} \cdot Gini(left) + \frac{n_{right}}{n} \cdot Gini(right)$$

Decision Tree Algorithm (CART)

Function CART(data):

If all examples have the same class -> Return a leaf with that class

best_split ← find the attribute and value that gives lowest Gini

If no good split is found (no improvement in Gini) -> Return a leaf with the majority class

left_data, right_data ← split data using best_split

left_tree ← CART(left_data)
right_tree ← CART(right_data)

Return a decision node with:

- the split condition
- left_tree and right_tree as children

A recursive algorithm: the function is internally recalled

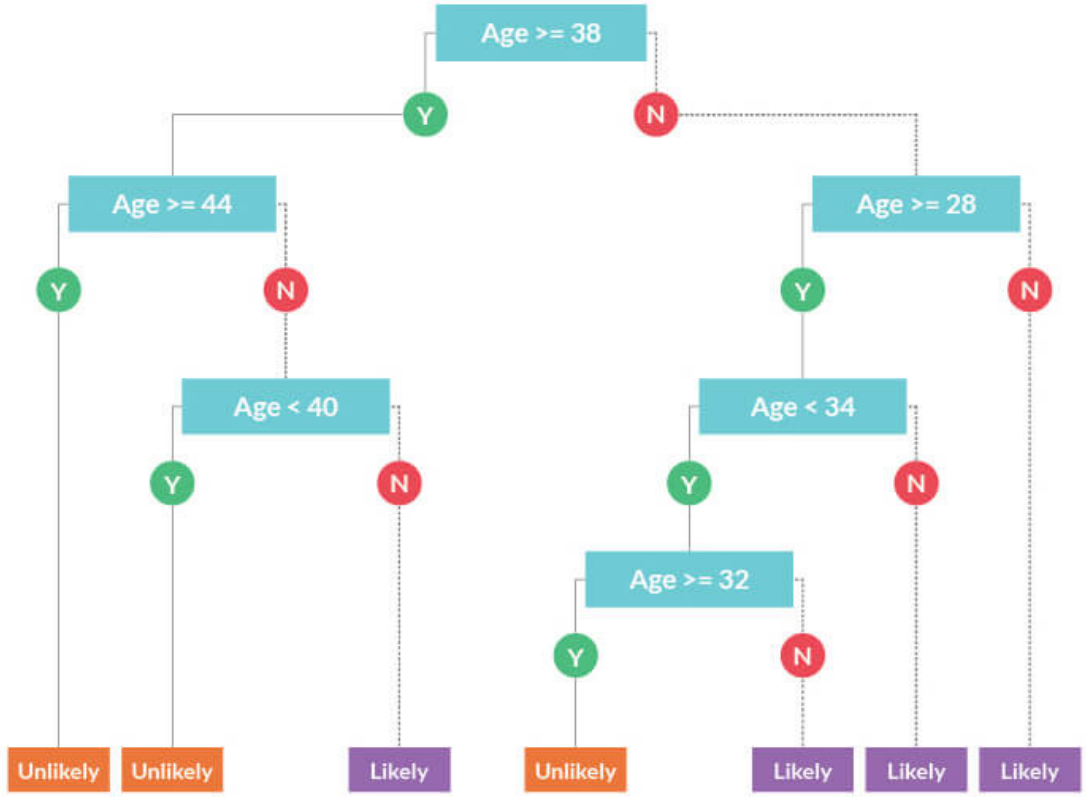
Numerical Variables

For a given numerical feature (e.g. sepal length), CART considers binary splits of the form:

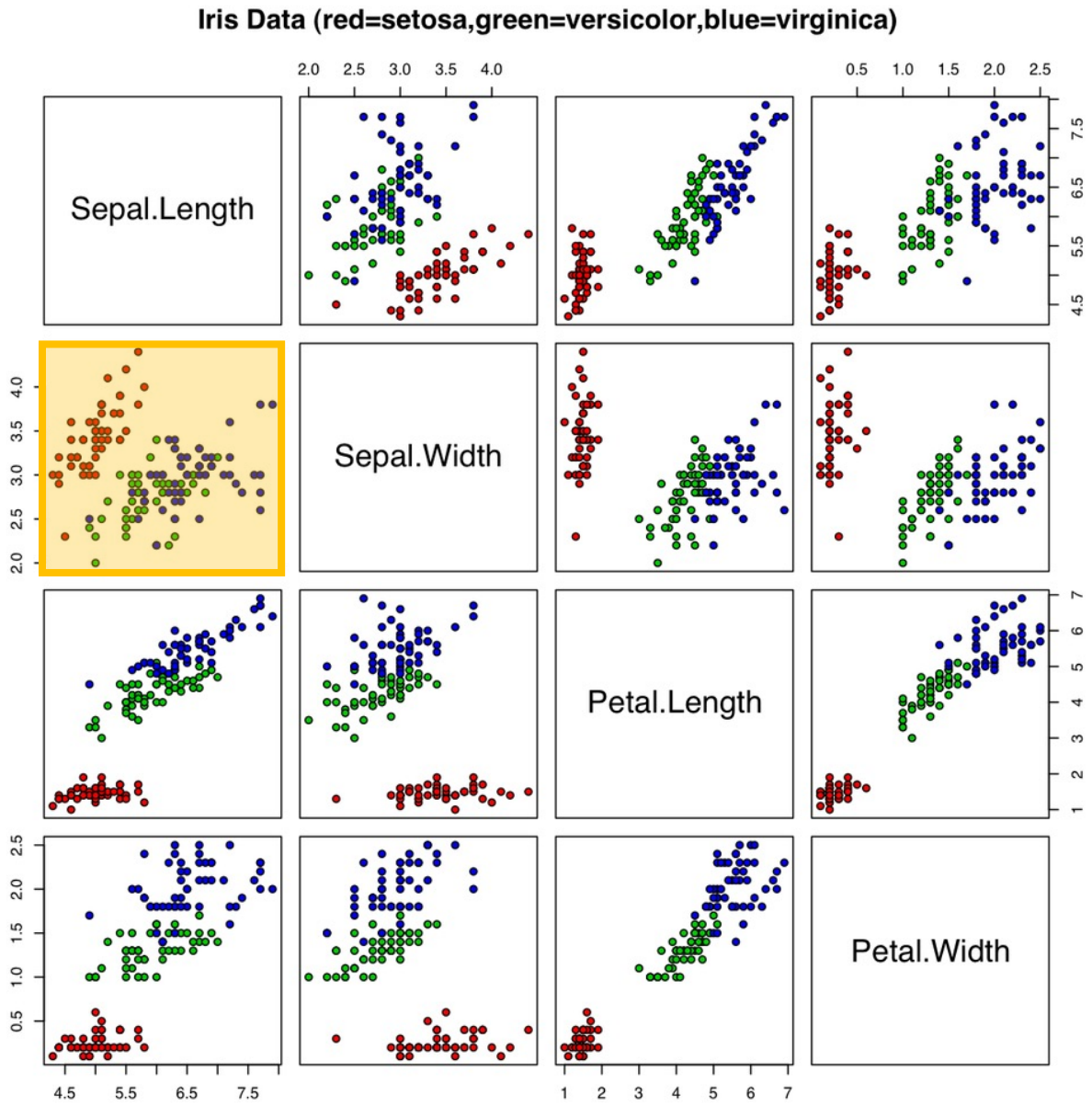
Feature $\leq t$

where t is a threshold value.

The algorithm tries many possible values of t and chooses the one that minimizes the impurity.

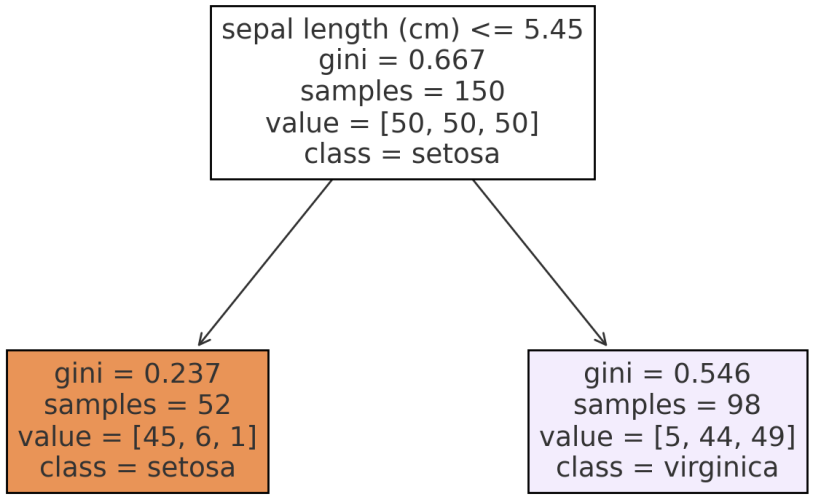


CART in action: Iris Dataset

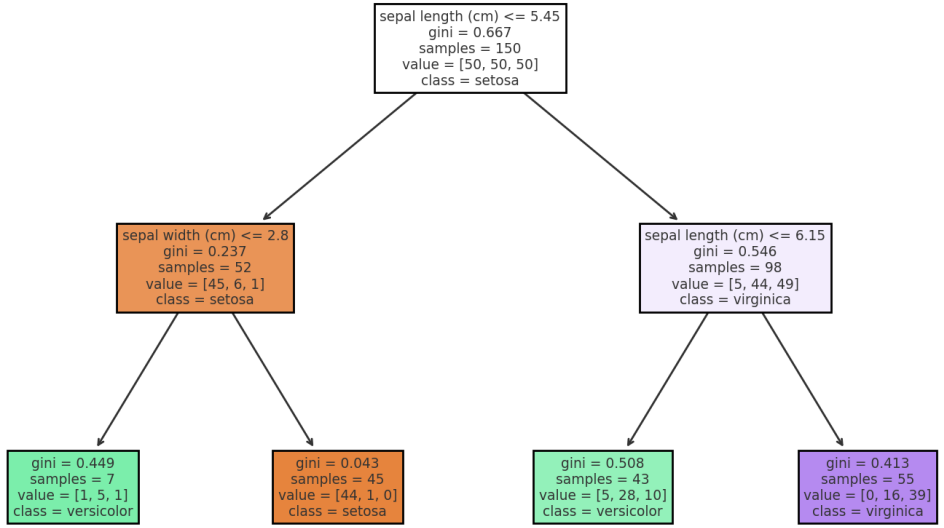


Wikipedia

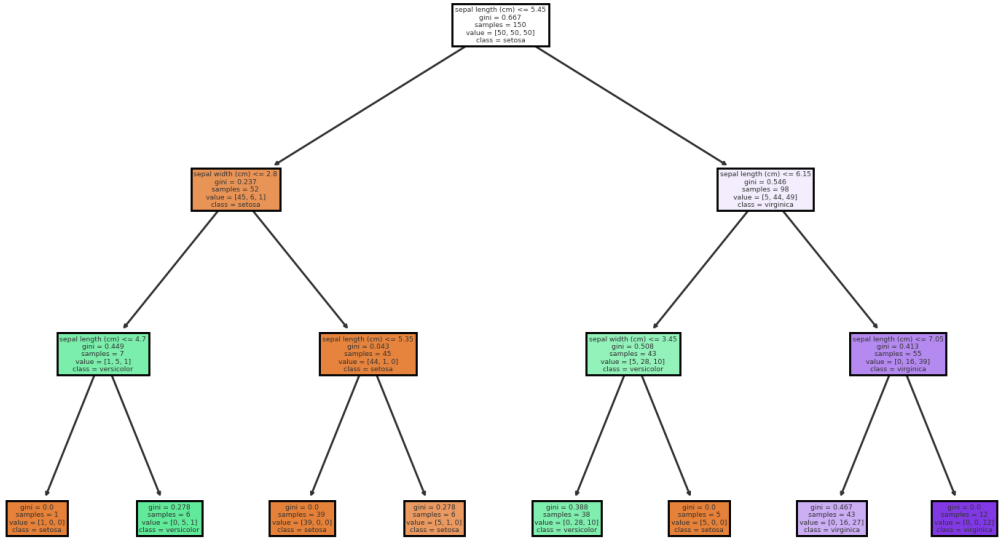
max_depth = 1



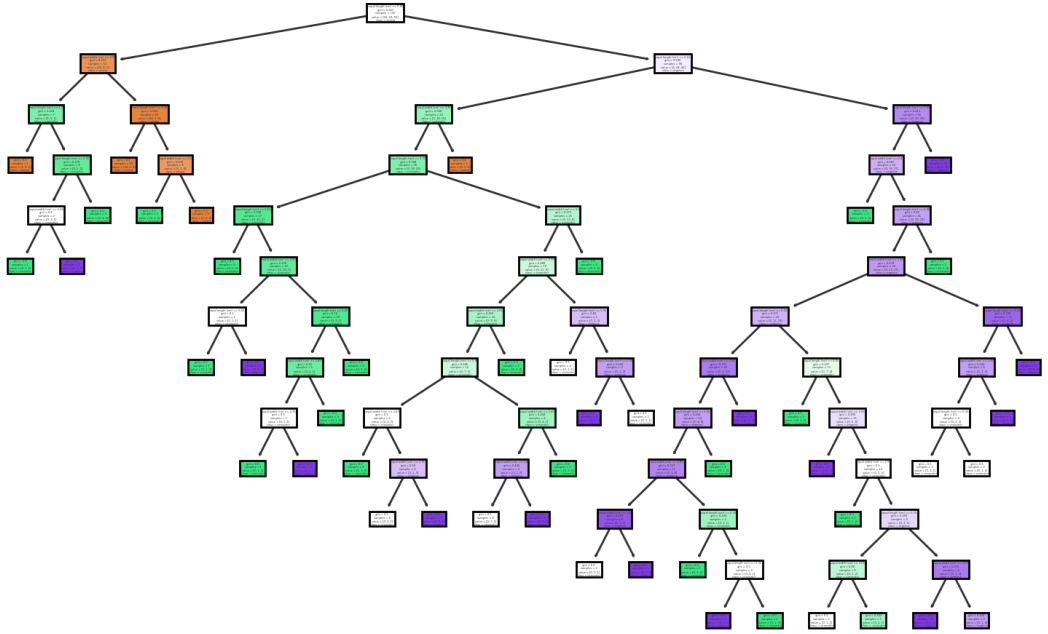
max_depth = 2

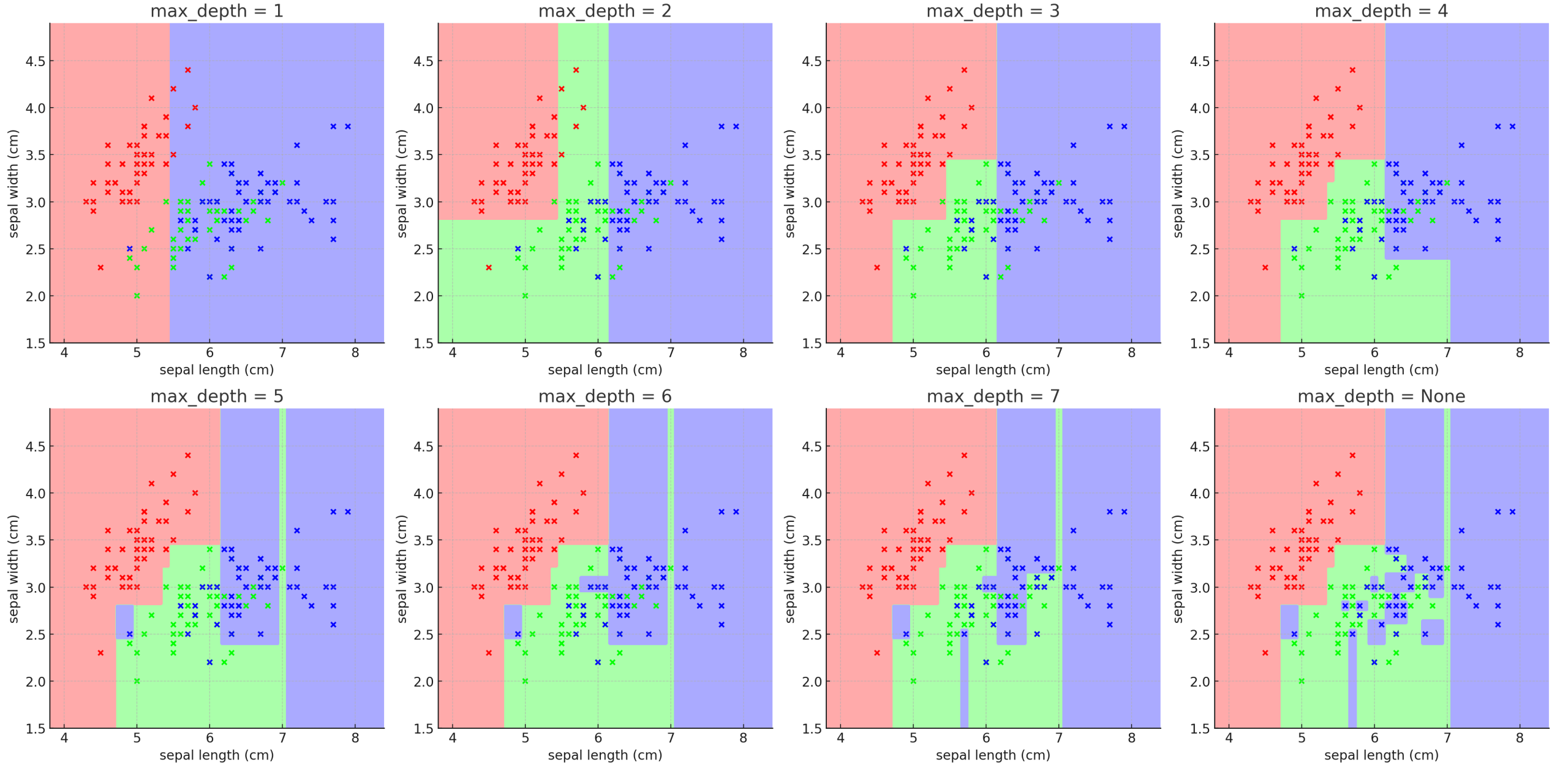


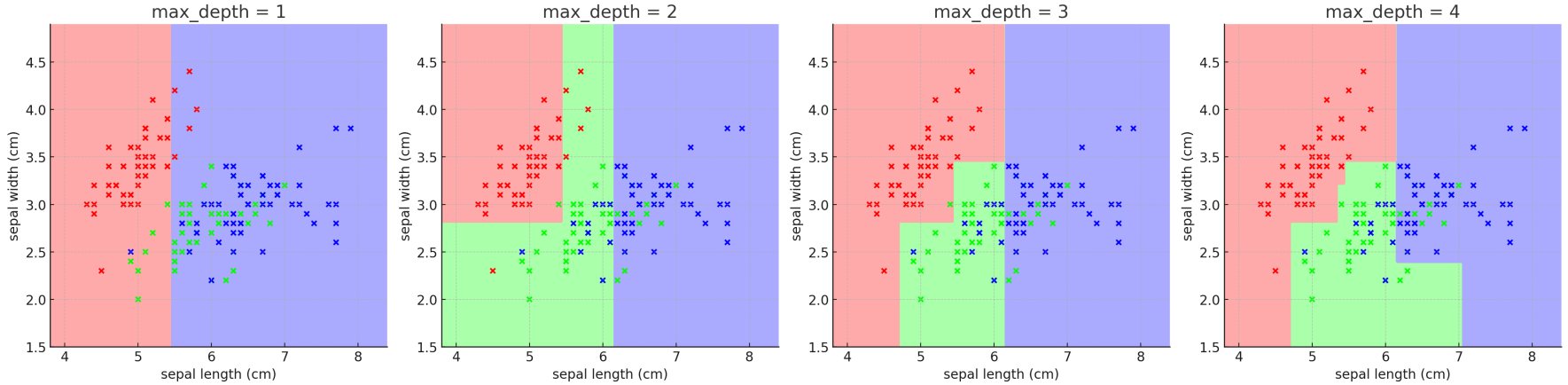
max_depth = 3

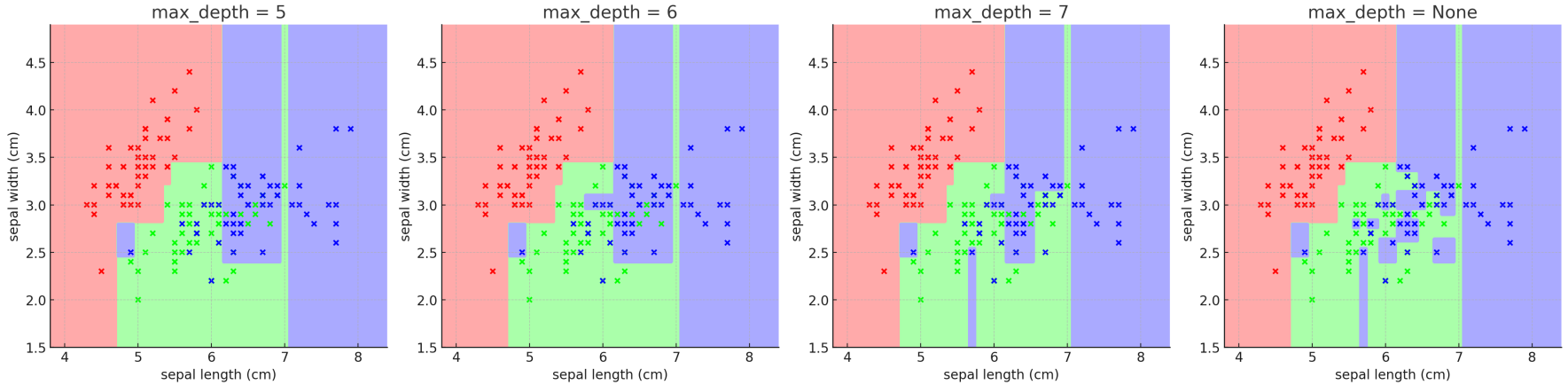


max_depth = None (full tree)

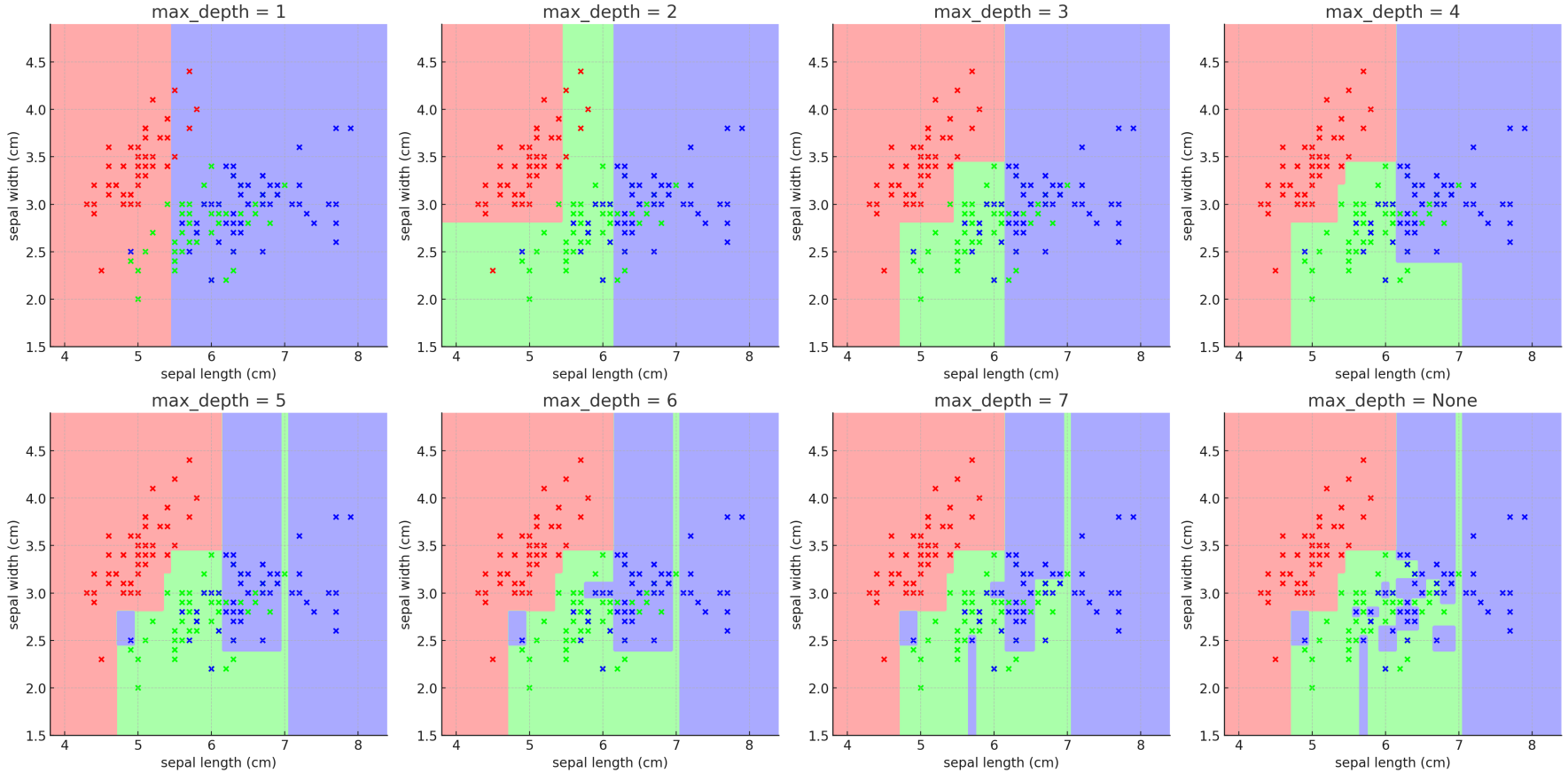



max_depth = 1
max_depth = 2
max_depth = 3
max_depth = 4
max_depth = 5
max_depth = 6
max_depth = 7
max_depth = None
sepal width (cm)
sepal length (cm)
1.5
2.0
2.5
3.0
3.5
4.0
4.5
4
5
6
7
8

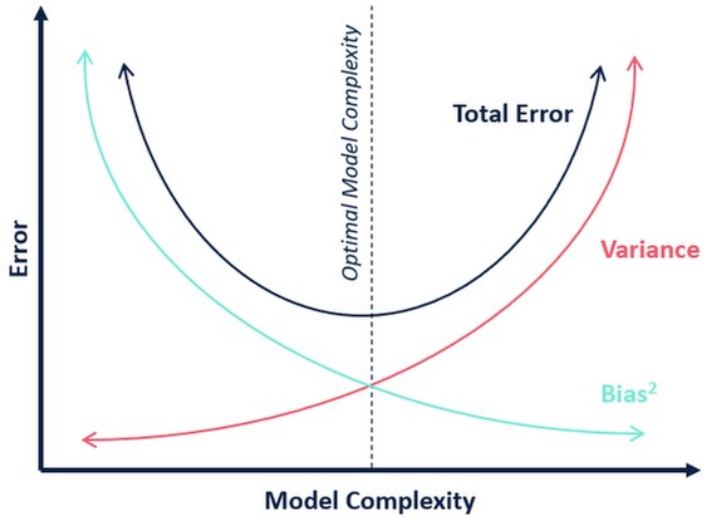


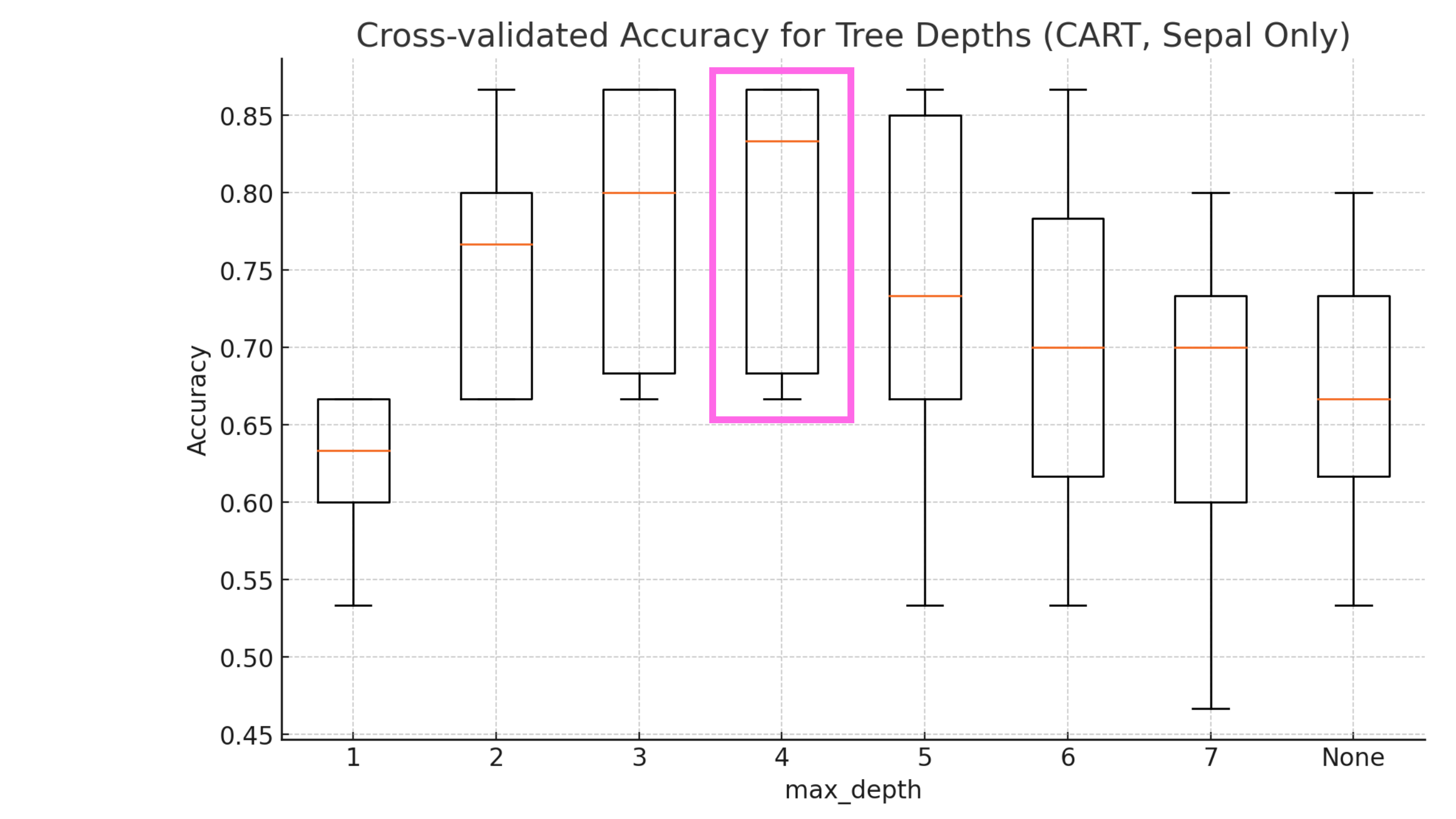


Don’t we see a similar behaviour that we had with other algorithms?



Don’t we see a similar behaviour that we had with other algorithms?




Cross-validated Accuracy for Tree Depths (CART, Sepal Only)
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
1
2
3
4
5
6
7
None
max_depth

✂️ Another way to deal with underfitting vs overfitting: Pruning

Pruning is the process of removing parts of a fully-grown tree to avoid overfitting and improve generalization.

CART uses Cost-Complexity Pruning, also called Minimal Cost-Complexity Pruning. It finds subtrees that balance the trade-off between tree accuracy & tree simplicity.

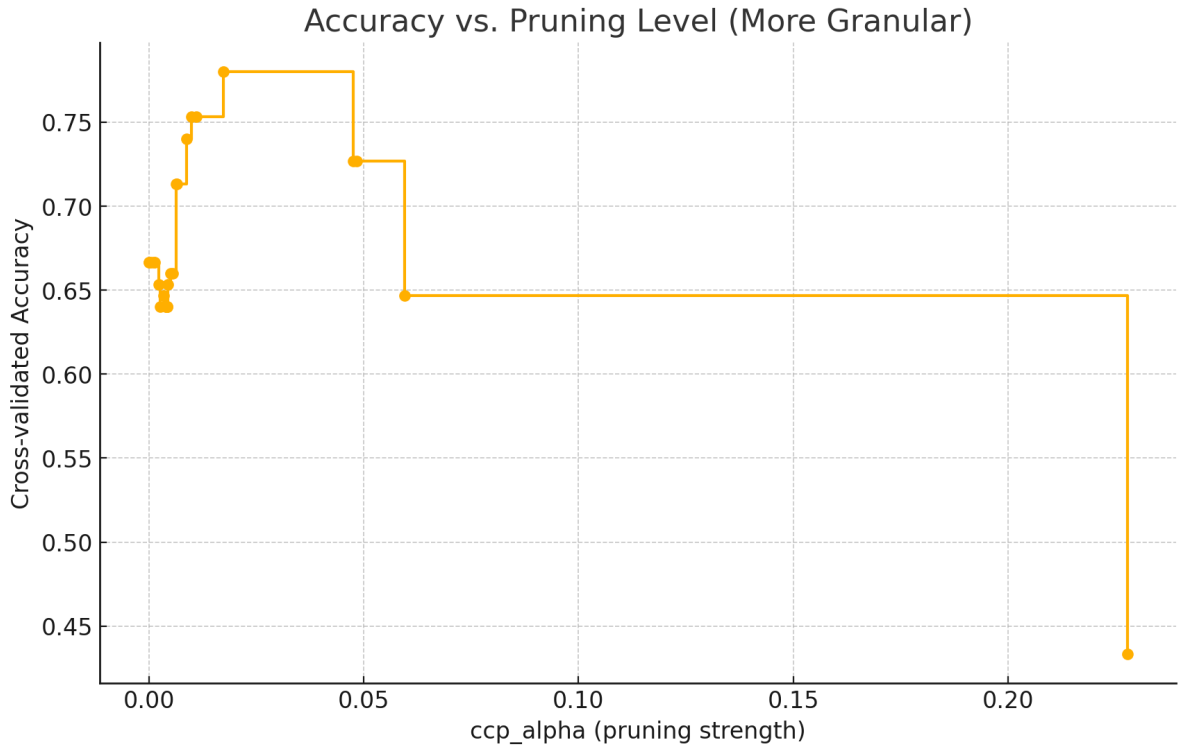
$$R_\alpha(T) = R(T) + \alpha \cdot |T|$$

where:

- $R(T)$: error (e.g., Gini impurity or misclassification rate)

- $|T|$: number of terminal (leaf) nodes

- α: pruning parameter (higher = more pruning)



🌲 Max Depth vs ✂️ Pruning

This is a pre-pruning technique (also called early stopping): it limits the depth of the tree before it's fully grown.

The tree building process stops once the specified depth is reached, even if further splits would improve the fit.

✅ Pros

- Fast and easy to control
- Prevents overfitting in small datasets

❌ Cons

- May miss useful splits just below the cutoff
- Not based on how meaningful those splits are

This is a post-hoc operation: you let the tree grow fully, then remove branches that don't improve generalization (based on validation error, cost-complexity, etc.)

✅ Pros

- More flexible: it allows complex splits only if they help
- Often results in better generalization than hard depth limits

❌ Cons

- Can be slower (requires evaluating subtrees)
- Needs careful tuning

Alternatives to CART

1.ID3, C4.5, and C5.0

1. Earlier and extended versions of CART.
2. C4.5 (and its commercial version C5.0) uses **information gain ratio** instead of Gini/entropy.
3. C5.0 is faster and more memory-efficient than CART.

2.CHAID (Chi-squared Automatic Interaction Detector)

1. Based on chi-square tests for splitting.
2. Handles categorical data well and can create multiway splits (not just binary).
3. Popular in social sciences and marketing.

3.QUEST (Quick, Unbiased, Efficient Statistical Tree)

1. Designed to reduce bias in variable selection.
2. Handles both categorical and continuous variables well.

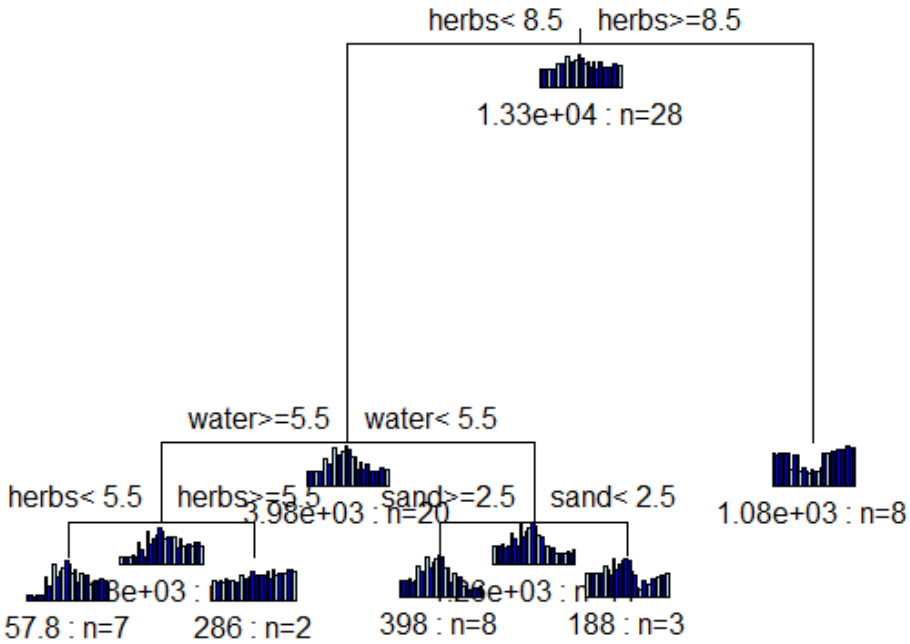
CART: Classification & Regression Tree

We can use regression trees! Splitting criteria changes:

1) MSE - At each split, the algorithm calculates the MSE for the left and right child nodes, and tries to minimize the weighted average MSE:

$$\text{MSE}_{\text{split}} = \frac{n_L}{n} \cdot \text{MSE}_L + \frac{n_R}{n} \cdot \text{MSE}_R$$

Where: n_L, n_R are number of samples in left/right child.
Lower MSE = better split!



CART: Classification & Regression Tree

We can use regression trees! Splitting criteria changes:

1) MSE - At each split, the algorithm calculates the MSE for the left and right child nodes, and tries to minimize the weighted average MSE:

$$\text{MSE}_{\text{split}} = \frac{n_L}{n} \cdot \text{MSE}_L + \frac{n_R}{n} \cdot \text{MSE}_R$$

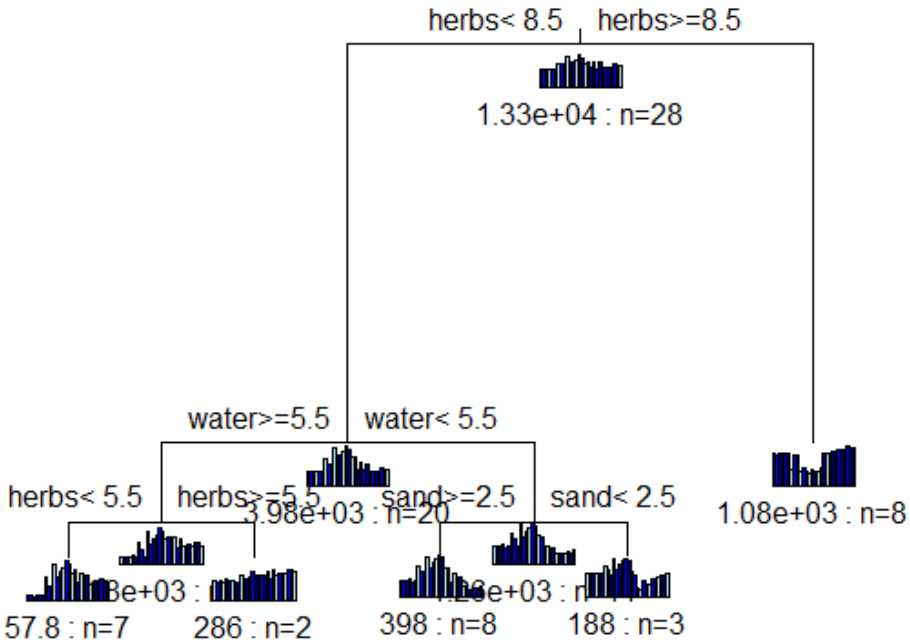
Where: n_L, n_R are number of samples in left/right child. Lower MSE = better split!

2) Variance reduction:

$$\text{Var}(y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$\text{Reduction} = \text{Var}(\text{parent}) - \left(\frac{n_L}{n} \cdot \text{Var}_L + \frac{n_R}{n} \cdot \text{Var}_R\right)$$

So — maximizing variance reduction = minimizing MSE.



CART: Classification & Regression Tree

We can use regression trees! Splitting criteria changes:

1) MSE - At each split, the algorithm calculates the MSE for the left and right child nodes, and tries to minimize the weighted average MSE:

$$\text{MSE}_{\text{split}} = \frac{n_L}{n} \cdot \text{MSE}_L + \frac{n_R}{n} \cdot \text{MSE}_R$$

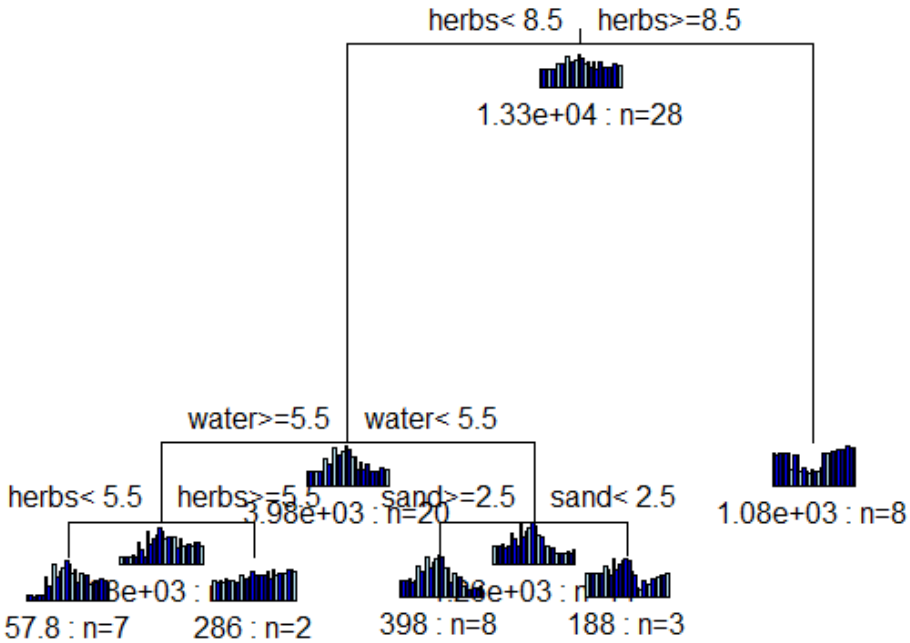
Where: n_L, n_R are number of samples in left/right child. Lower MSE = better split!

2) Variance reduction:

$$\text{Var}(y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$\text{Reduction} = \text{Var}(\text{parent}) - \left(\frac{n_L}{n} \cdot \text{Var}_L + \frac{n_R}{n} \cdot \text{Var}_R\right)$$

So — maximizing variance reduction = minimizing MSE.



Leaf output changes: leaves output the mean of the target variable in the node!

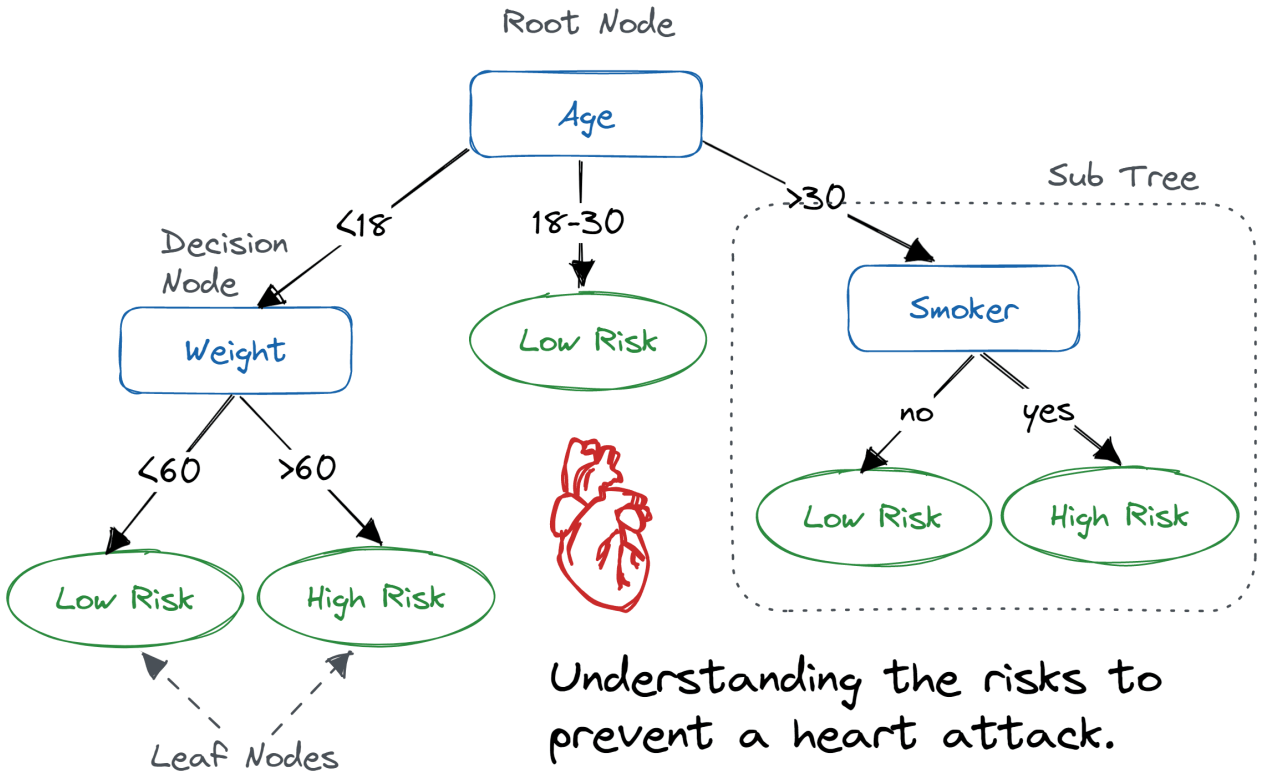
Decision Tree

Advantages

- Easily interpretable
- They require no data normalization
- The classification is almost immediate
- The computational expensive part is done off-line (once)

Drawbacks

- Really high variance classifiers→ Prone to overfitting! Typically, poor generalization performances!



Decision Tree -> Random Forest

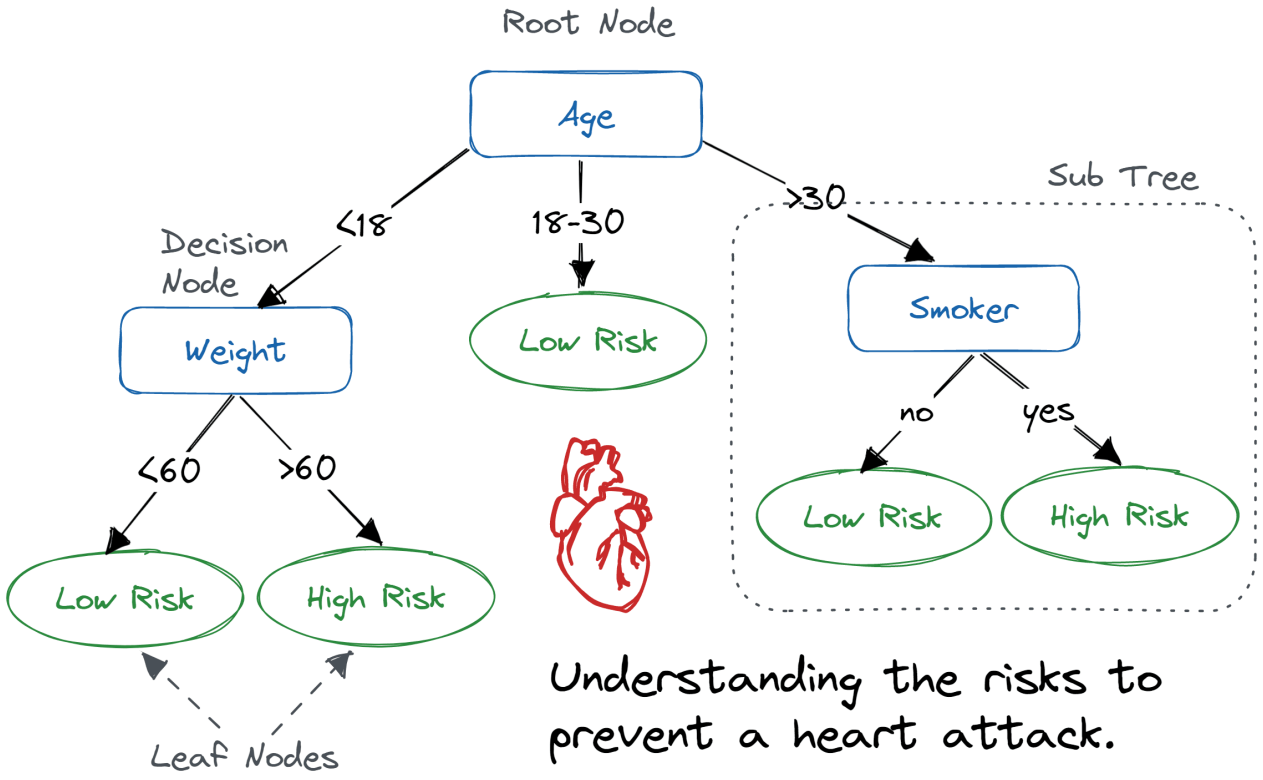
Advantages

- Easily interpretable
- They require no data normalization
- The classification is almost immediate
- The computational expensive part is done off-line (once)

Drawbacks

- Really high variance classifiers→ Prone to overfitting! Typically, poor generalization performances!

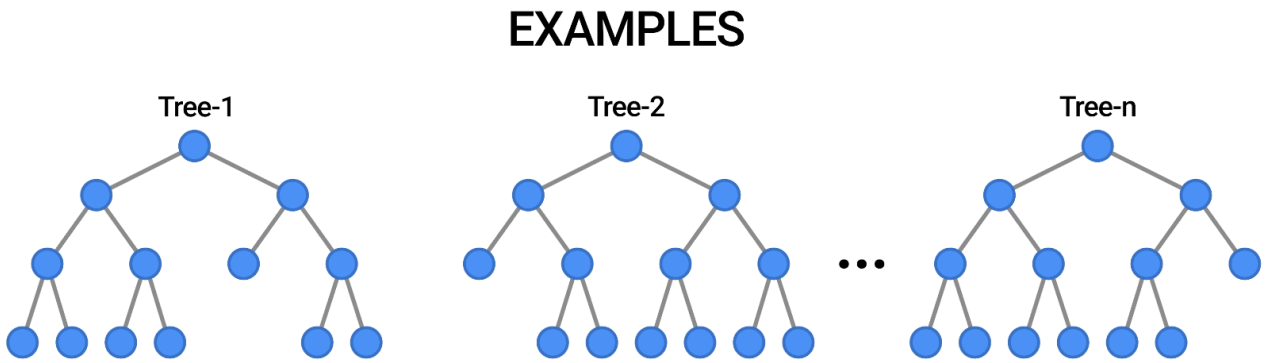
We can solve this by considering many trees: a forest!!!!

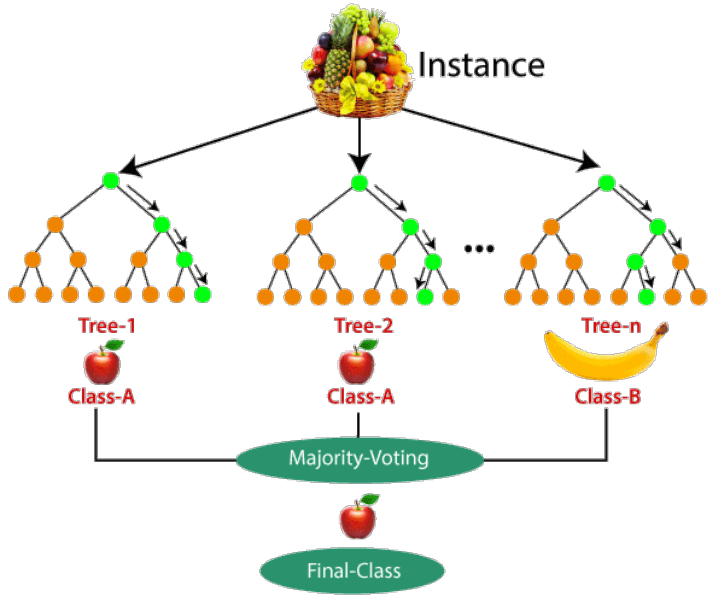


Random Forest (RF)

A RF is composed by many ‘weak’ learners (decision trees): we cleverly combine DTs reducing overfitting!

We construct slightly different DTs (more on this later) and, in classification, we decide by a majority-voting (we choose following the mode) the final class. In regression, the final decision is the average.



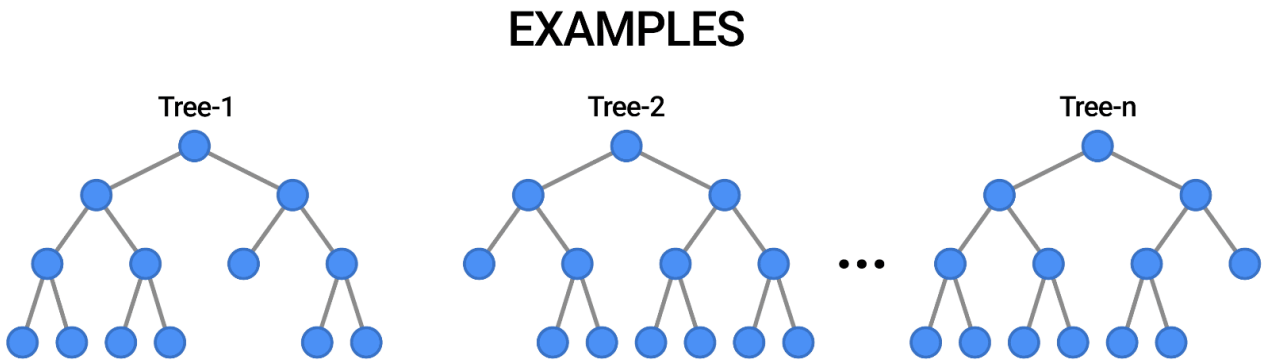


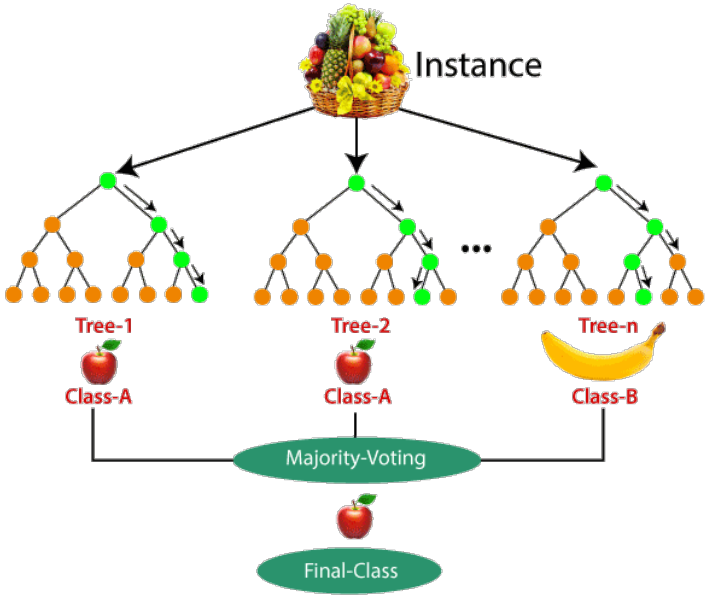
Random Forest (RF)

A RF is composed by many ‘weak’ learners (decision trees): we cleverly combine DTs reducing overfitting!

We construct slightly different DTs (more on this later) and, in classification, we decide by a majority-voting (we choose following the mode) the final class. In regression, the final decision is the average.

This in an ‘ensemble’ approach: we combine multiple models (often called base learners or weak learners) to produce a stronger model.





Random Forest 

Journal of Machine Learning Research 15 (2014) 3133-3181 Submitted 11/13; Revised 4/14; Published 10/14

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Abstract

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

4. Conclusion

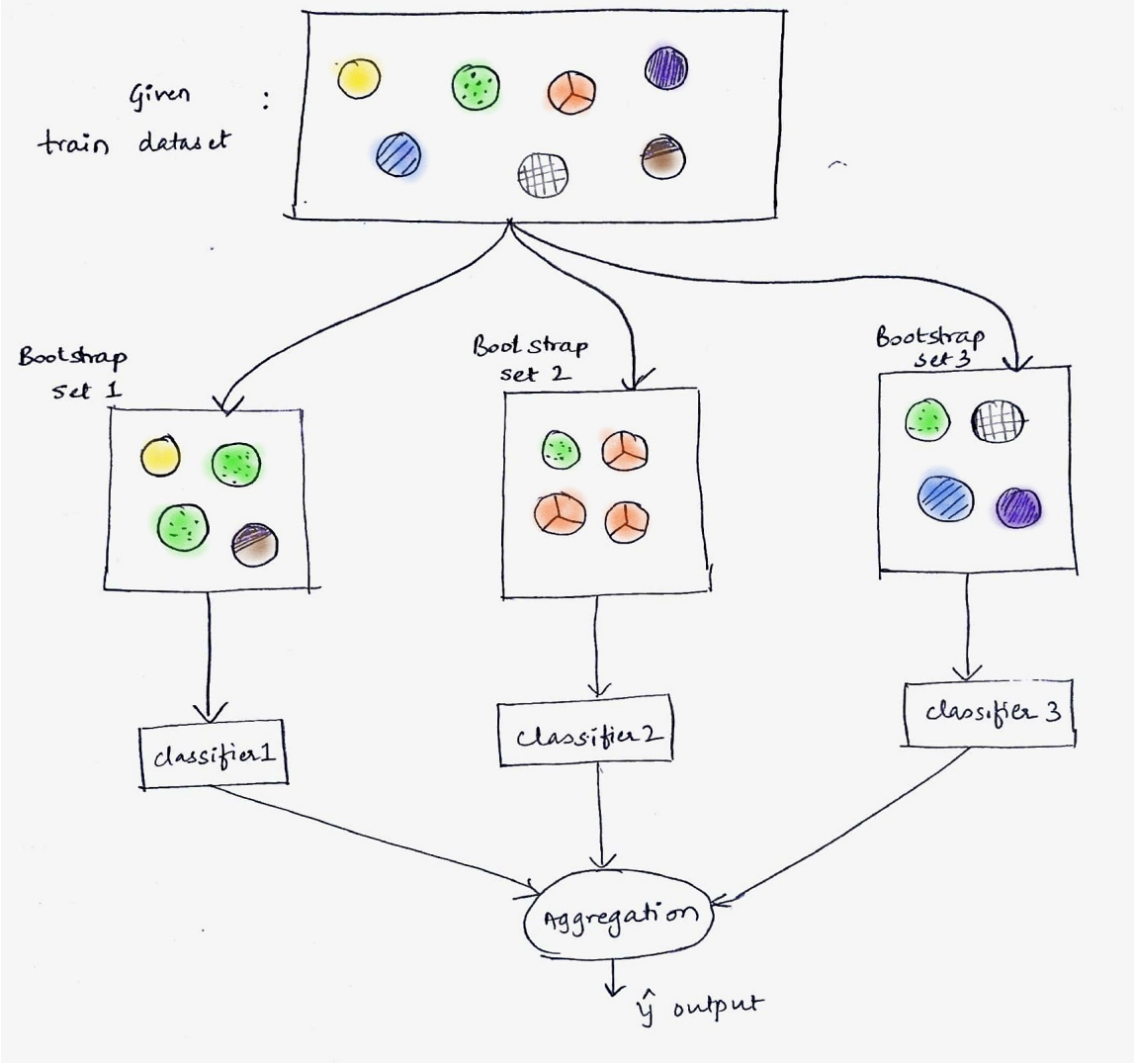
This paper presents an exhaustive evaluation of 179 classifiers belonging to a wide collection of 17 families over the whole UCI machine learning classification database, discarding the large-scale data sets due to technical reasons, plus 4 own real sets, summing up to 121 data sets from 10 to 130,064 patterns, from 3 to 262 inputs and from 2 to 100 classes. **The best results are achieved by the parallel random forest** (parRF_t), implemented in R with caret, tuning the parameter mtry. The parRF_t achieves in average 94.1% of the maximum accuracy over all the data sets (Table 5, lower part), and overcomes the 90% of

🔧 How to Build a Random Forest (Conceptually)

Let's assume you want to build a forest with T trees.

For each tree:

- Sample the dataset with replacement (bootstrap sample). This procedure is called Bagging (bootstrap aggregating).

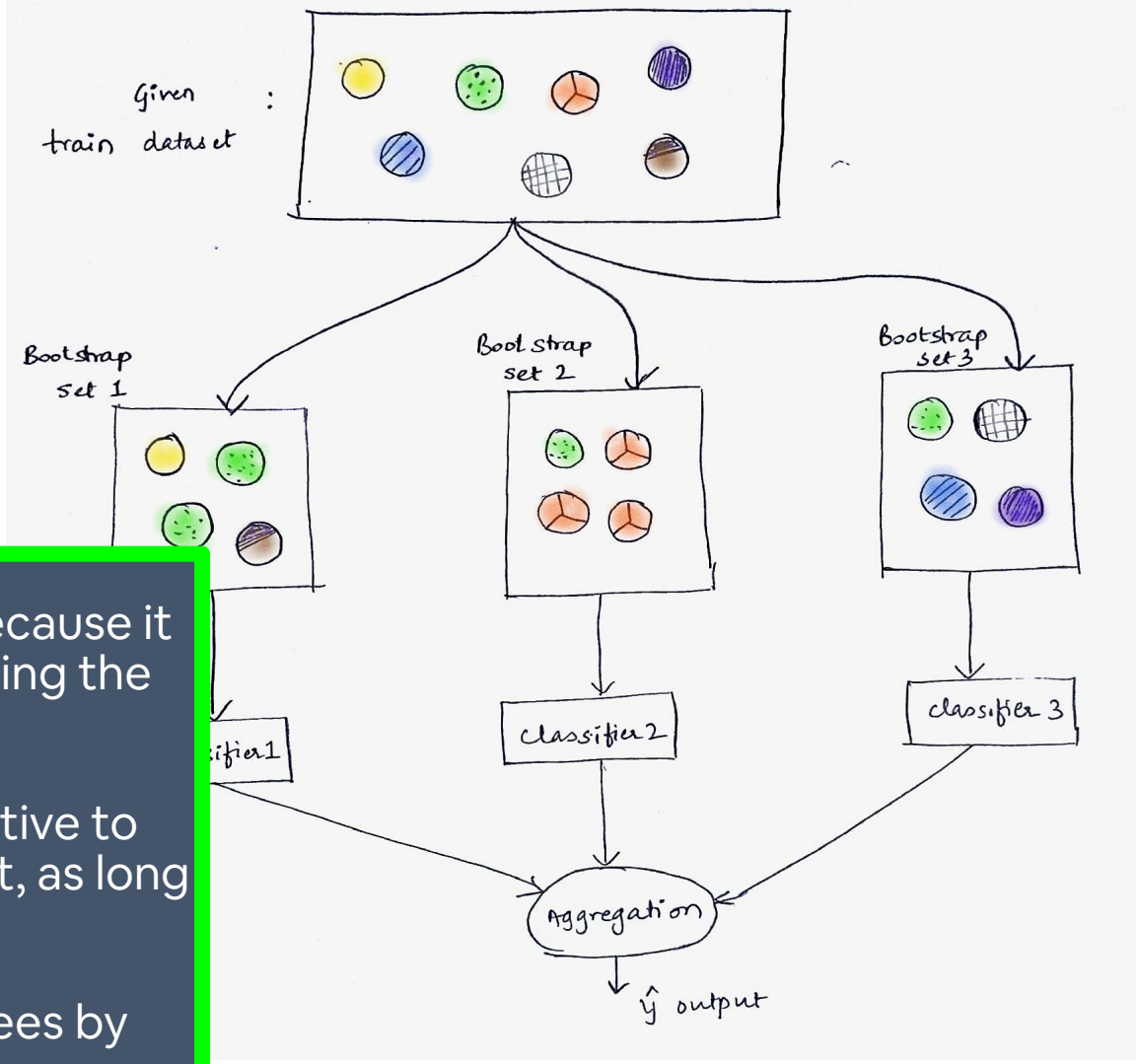


🔧 How to Build a Random Forest (Conceptually)

Let's assume you want to build a forest with T trees.

For each tree:

- Sample the dataset with replacement (bootstrap sample). This procedure is called Bagging (bootstrap aggregating).



This procedure leads to better model performance because it decreases the variance of the model, without increasing the bias.

While the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated.

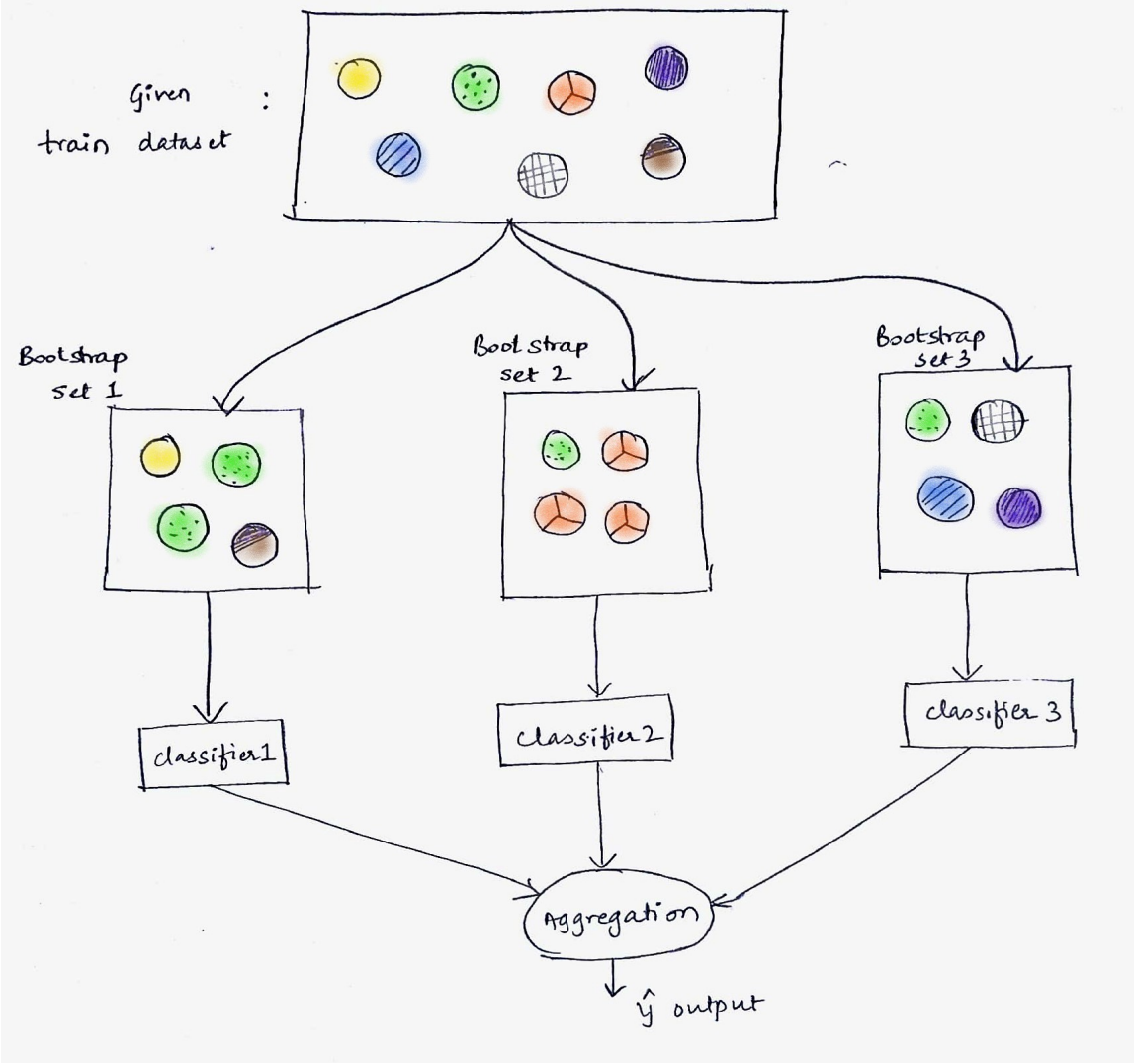
Bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

🔧 How to Build a Random Forest (Conceptually)

Let's assume you want to build a forest with T trees.

For each tree:

- Sample the dataset with replacement (bootstrap sample). This procedure is called Bagging (bootstrap aggregating).
- Build a decision tree: but at each split, instead of evaluating all features, pick a random subset (e.g., $\sqrt{p}$). This procedure is called Feature Bagging.

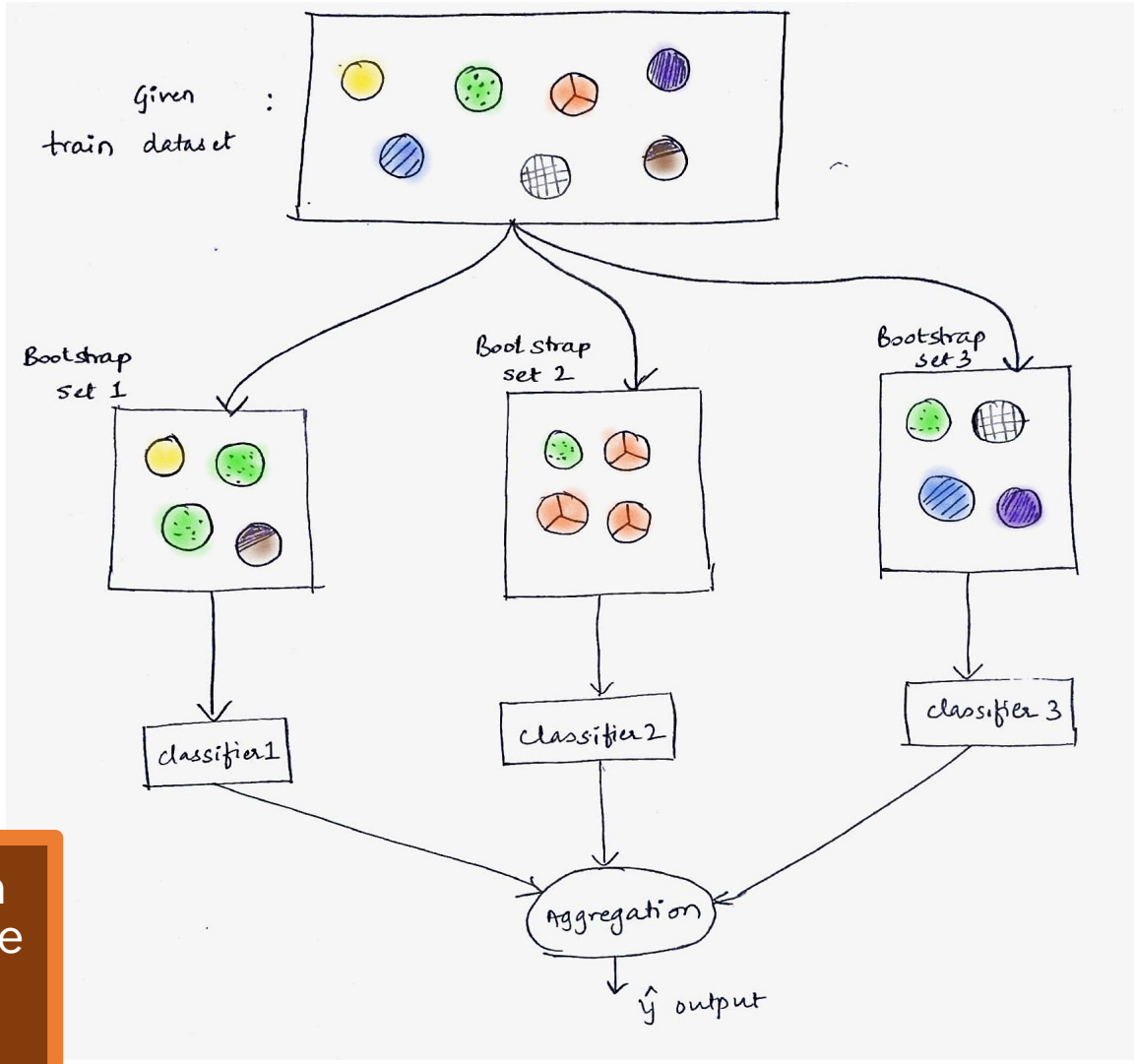


🔧 How to Build a Random Forest (Conceptually)

Let's assume you want to build a forest with T trees.

For each tree:

- Sample the dataset with replacement (bootstrap sample). This procedure is called Bagging (bootstrap aggregating).
- Build a decision tree: but at each split, instead of evaluating all features, pick a random subset (e.g., $\sqrt{p}$). This procedure is called Feature Bagging.



The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the T trees, causing them to become correlated.

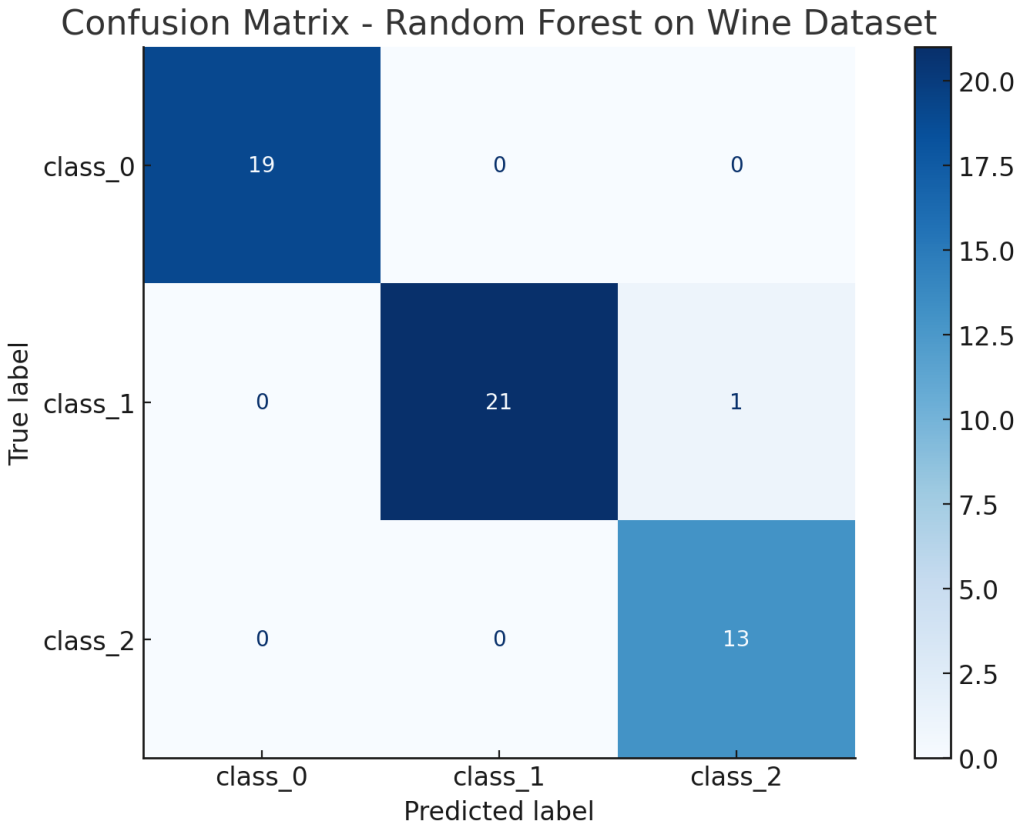
Example of RF in action: 🍇 Wine Dataset

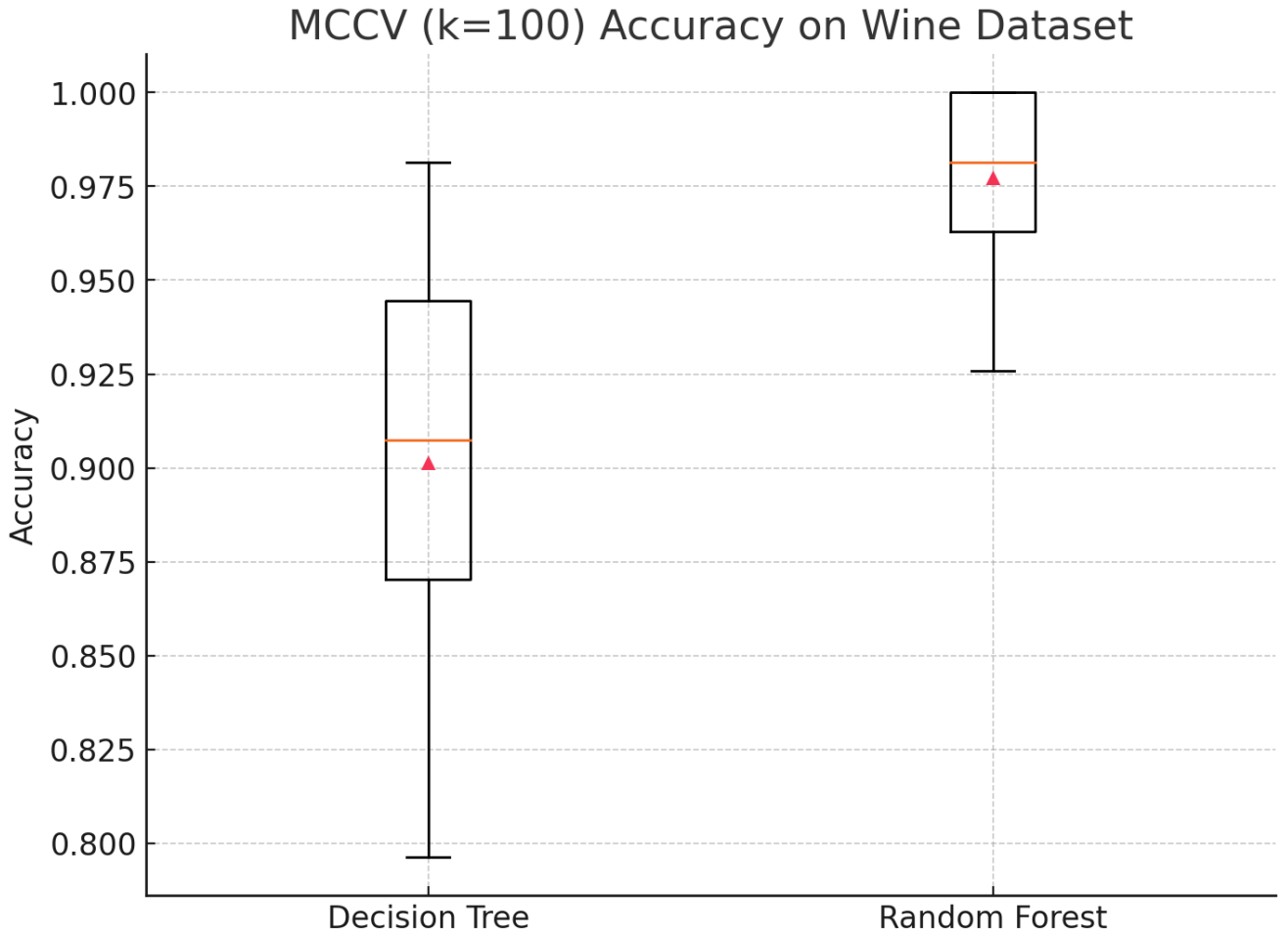
📦 Description

- Objective: Classify wines into 3 classes (based on grape varieties)
- Features (p): 13 chemical measurements per wine
- Samples (n): 178
- Classes: Class_0, Class_1, Class_2

📊 Features include Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Color intensity, Hue, OD280/OD315 of diluted wines, Proline (and more...)

Example of RF in action: 🍇 Wine Dataset



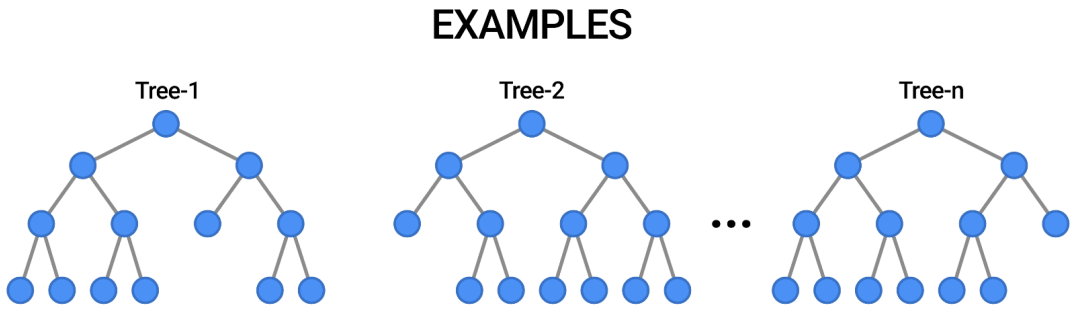


Random Forests RF in a Nutshell

-Hyperparameters:
- number of trees (we just put ‘enough trees’)

-Pro:
- Multi-class
- No data normalization required
- The computational expensive part is done off-line (once)
- The classification is almost immediate
- High (highest!) classification accuracy

-Drawbacks:
-Time consuming computation (but it can be easily parallelized)

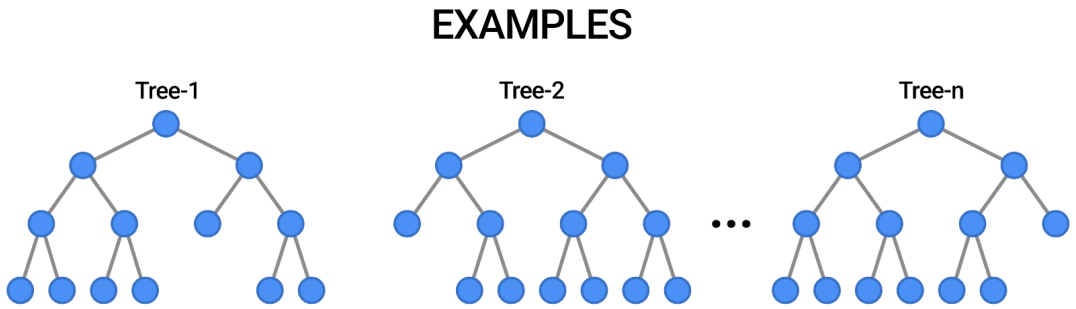


Random Forests RF in a Nutshell

-Hyperparameters:
- number of trees (we just put 'enough trees')

-Pro:
- Multi-class
- No data normalization required
- The computational expensive part is done off-line (once)
- The classification is almost immediate
- High (highest!) classification accuracy

-Drawbacks:
-Time consuming computation (but it can be easily parallelized)
-Lack of interpretability... is it so?



RF: feature importance

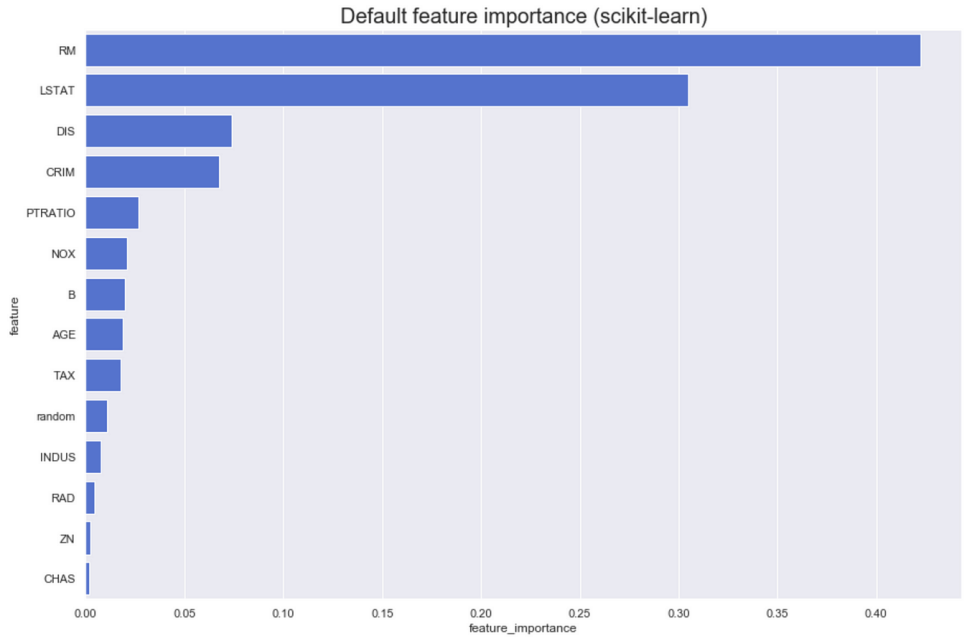
Feature importance reflects how useful or valuable each feature is for making predictions in a model. For decision trees (and ensembles like Random Forests), it's typically based on:

🔍 How much each feature decreases impurity (e.g., Gini index or entropy) when it’s used to split the data

📊 Intuition

- If a feature is consistently chosen for important splits (i.e., it helps reduce impurity a lot), it gets high importance.

- Features that are rarely used or don’t reduce impurity much get low or zero importance.



RF: feature importance

Feature importance reflects how useful or valuable each feature is for making predictions in a model. For decision trees (and ensembles like Random Forests), it's typically based on:

🔍 How much each feature decreases impurity (e.g., Gini index or entropy) when it’s used to split the data

📊 Intuition

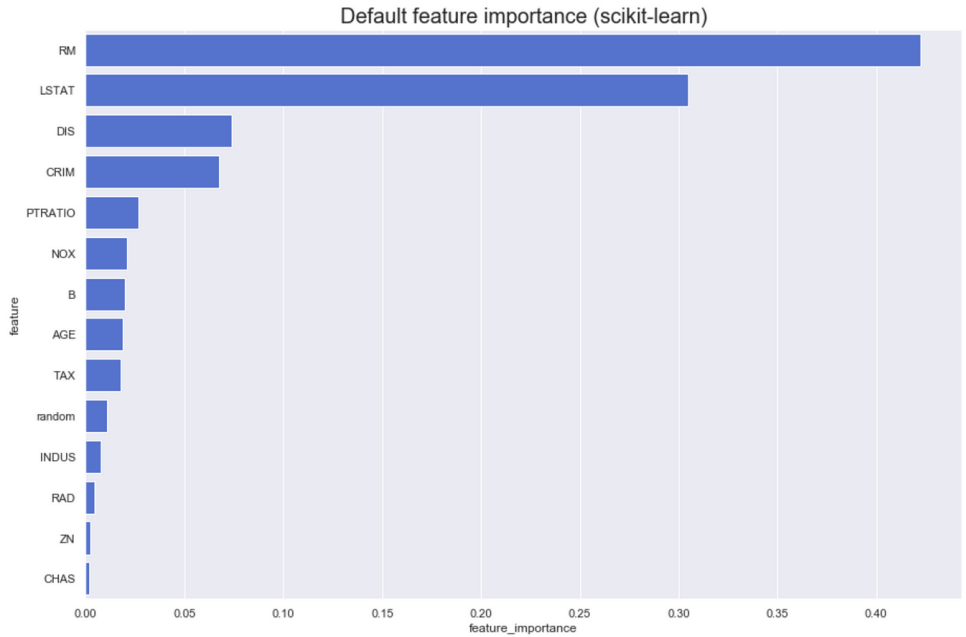
- If a feature is consistently chosen for important splits (i.e., it helps reduce impurity a lot), it gets high importance.

- Features that are rarely used or don’t reduce impurity much get low or zero importance.

This is a ‘eXplainable Artificial Intelligence (XAI)’ approach.

It is a ’global’ approach: provide us with info on the whole model structure

Any idea how can this information be exploited?



RF: feature importance - Derivation

Let's consider the Gini impurity, and we have a decision tree:

1. At every split, the algorithm calculates how much that split reduces impurity:

$$\Delta Gini = Gini(\text{parent}) - \left(\frac{n_{left}}{n} \cdot Gini(\text{left}) + \frac{n_{right}}{n} \cdot Gini(\text{right})\right)$$

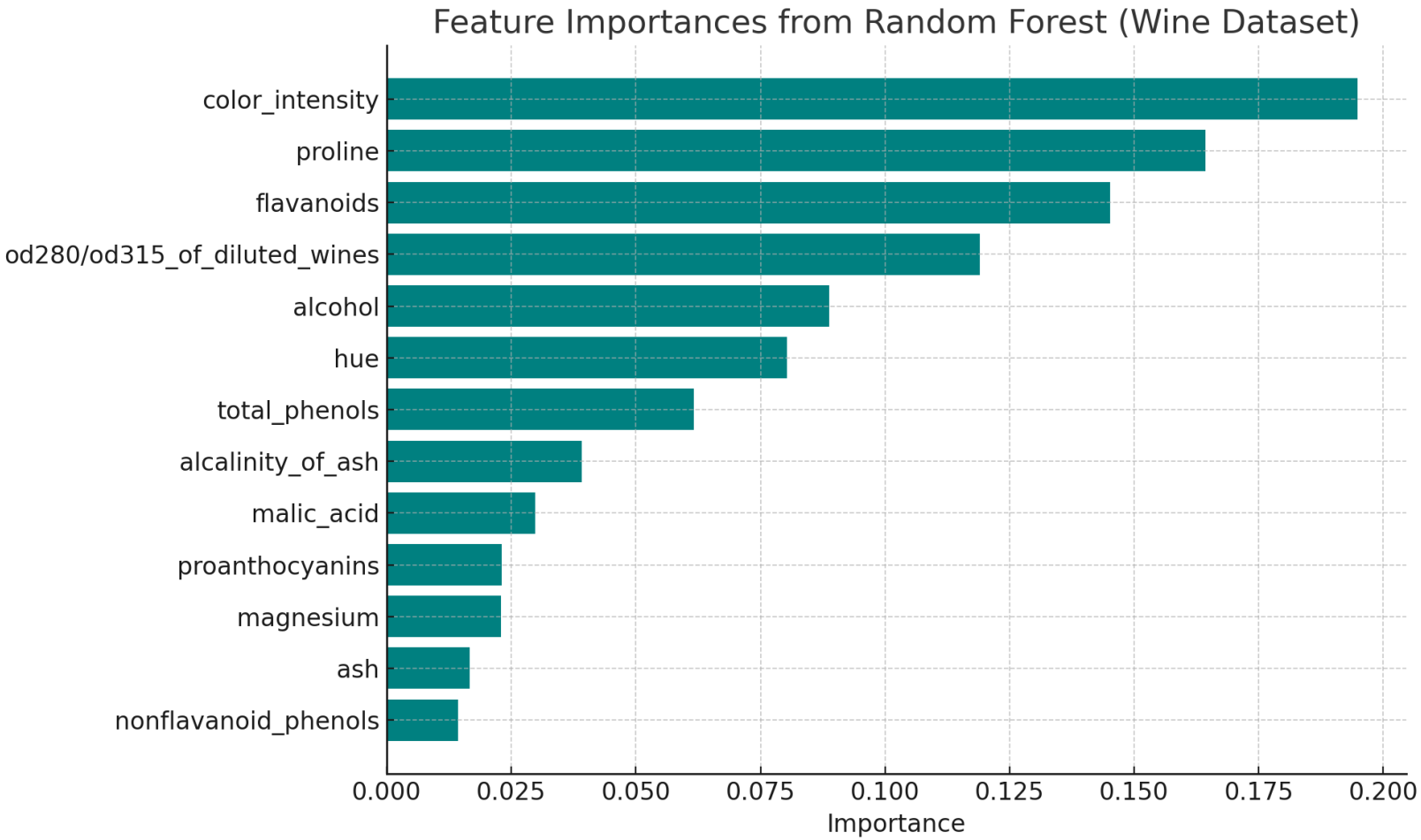
2. The contribution of a feature is the sum of all impurity decreases where that feature was used to split:

$$Importance(feature) = \sum_{\text{nodes using feature}} \Delta Gini$$

3. In a Random Forest, we average this importance over all the trees in the forest.
4. (Optional) the importances are normalized so they sum to 1:

$$\text{Normalized Importance} = \frac{\text{Raw Importance}}{\sum \text{Raw Importances}}$$

On the wine dataset



Thank you!

Gian Antonio Susto