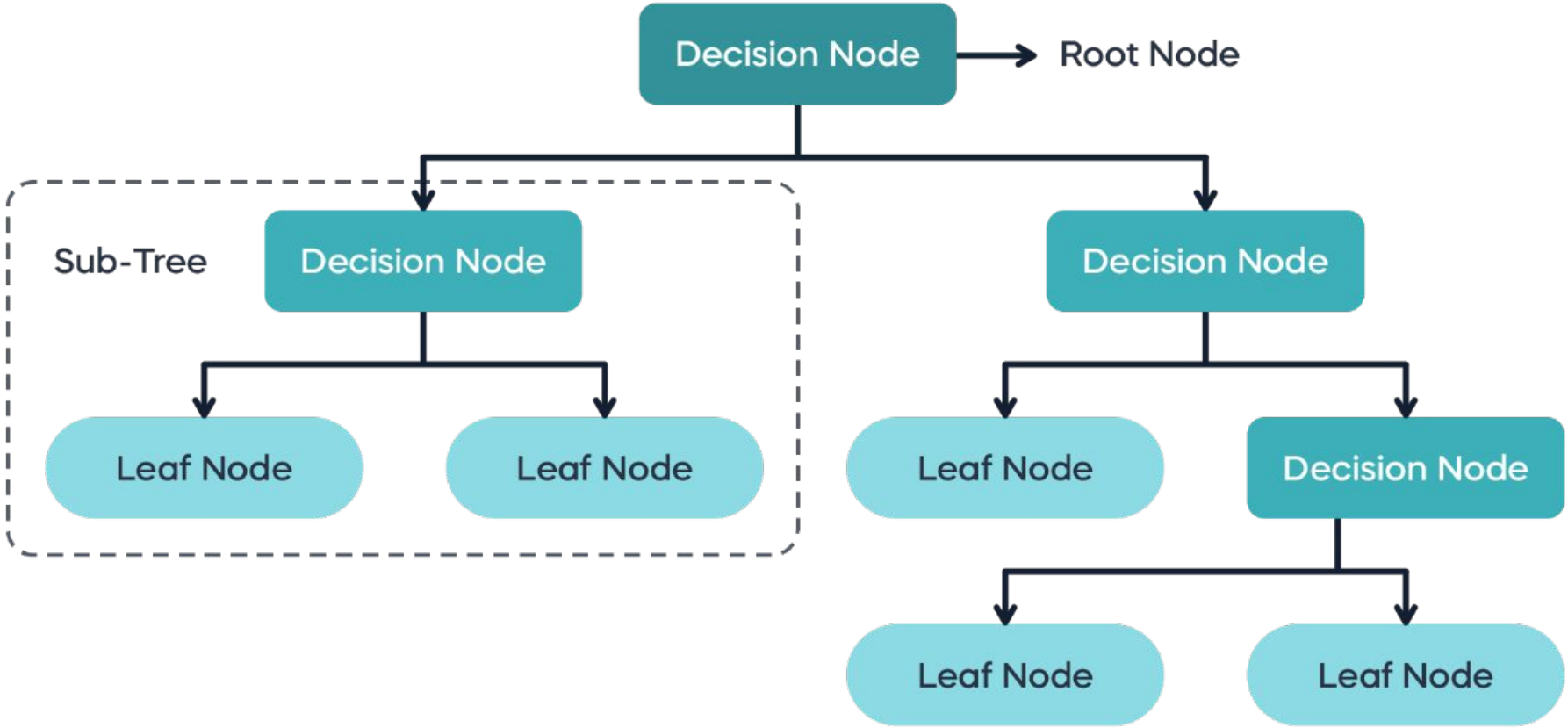
Decision Trees


Decision Node
Root Node
Sub-Tree
Decision Node
Decision Node
Leaf Node
Leaf Node
Leaf Node
Decision Node
Leaf Node
Leaf Node

CART Algorithm

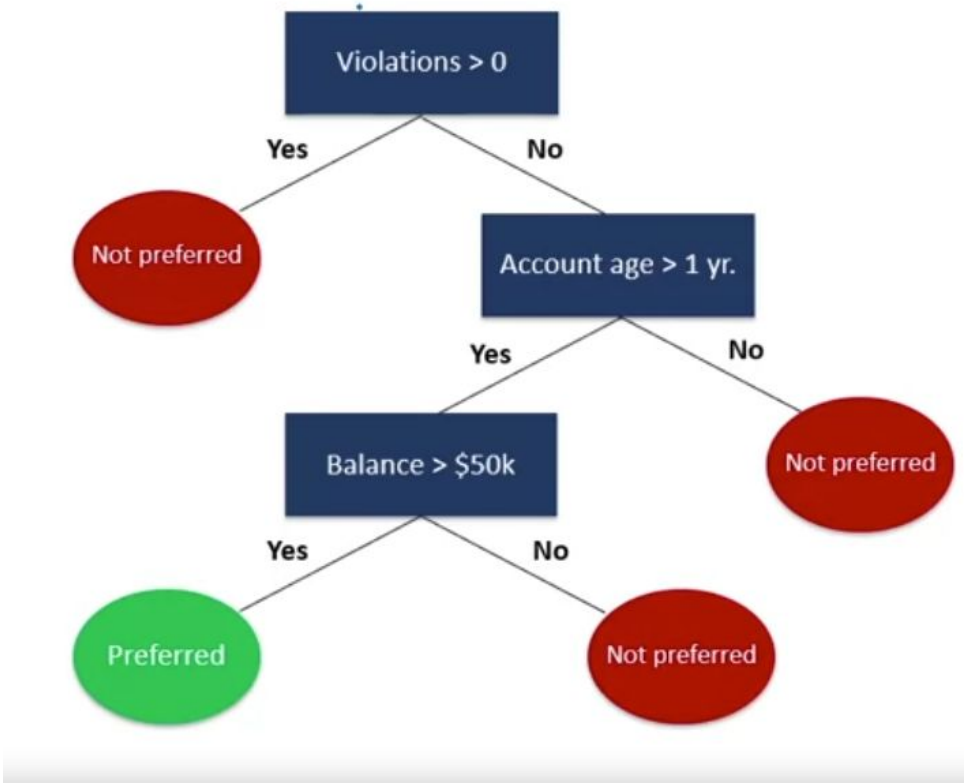
CART Algorithm:

Stands for Classification and Regression Tree.

- **Purpose:** Used in decision trees for both classification and regression tasks.
- **Binary Splits:** At each decision node, the dataset is split into two parts based on a selected feature.

Process:

- Select a feature from the training dataset.
- Split the dataset at each decision node based on the feature.
- Continue splitting recursively using other features.



Binary Splits in CART

Binary Splitting:

- Each node splits into exactly two child nodes.
- Simplifies the tree structure.

Feature Selection:

- Uses the Gini Index to determine the best feature to split on.
- The feature with the lowest Gini Index is selected for splitting.

Gini Index

Purpose: Measures the impurity or purity of a node.

$$G = 1 - \sum_{i=1}^{c} (p_i)^2$$

- G: Gini Index.
- P_i: Probability of an item being classified into a particular class.
- c: Total number of classes.

Notes:

- **Gini Index = 0**: Node is pure; all samples belong to one class.
- **Higher Gini Index**: Indicates more impurity.

Steps in Calculating the Gini Index

1. **Compute for Each Feature**:
   - Calculate the Gini Index for each feature relative to the label.

$$G = 1 - \sum_{i=1}^{c} (p_i)^2$$

2. **Weighted Sum**:
   - Combine the Gini Indices of each split using a weighted sum based on the number of samples.

$$\text{Weighted Gini Index} = \sum_{i=1}^{n} \left(\frac{N_i}{N_{\text{total}}} \times Gini_i \right)$$

   - Using the Weighted Gini Index helps in selecting the feature that best partitions the dataset into homogeneous subsets

3. **Select Feature with Lowest Gini Index**
   - This feature becomes the root node or the next decision node.

: Example Scenario: Determine if a bank customer deserves "Preferred" status.

Account age > 1	Violations > 0	Balance > $50,000	Preferred
Yes	No	Yes	No
Yes	Yes	Yes	No
No	No	Yes	No
Yes	No	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
No	No	Yes	Yes
No	Yes	Yes	No

Account age > 1	Violations > 0	Balance > $50,000	Preferred
Yes	No	Yes	No
Yes	Yes	Yes	No
No	No	Yes	No
Yes	No	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
No	No	Yes	Yes
No	Yes	Yes	No

- **Data for 'Yes' in Account Age:**
  - Total samples: 5.
  - Preferred 'Yes': 2.
  - Preferred 'No': 3.
- **Gini Index Calculation:**

$$G_{\text{Yes}} = 1 - \left(\left(\frac{2}{5} \right)^2 + \left(\frac{3}{5} \right)^2 \right) = 0.48$$

- **Data for 'No' in Account Age:**
  - Total samples: 3.
  - Preferred 'Yes': 1.
  - Preferred 'No': 2.
- **Gini Index Calculation:**

$$G_{\text{No}} = 1 - \left(\left(\frac{1}{3} \right)^2 + \left(\frac{2}{3} \right)^2 \right) = 0.44$$

Calculating Gini Index for 'Account Age'

Account age > 1	Violations > 0	Balance > $50,000	Preferred
Yes	No	Yes	No
Yes	Yes	Yes	No
No	No	Yes	No
Yes	No	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
No	No	Yes	Yes
No	Yes	Yes	No

- **Data for 'Yes' in Account Age:**
  - Total samples: 5.
  - Preferred 'Yes': 2.
  - Preferred 'No': 3.
- **Gini Index Calculation:**

$$G_{\text{Yes}} = 1 - \left(\left(\frac{2}{5}\right)^2 + \left(\frac{3}{5}\right)^2\right) = 0.48$$

- **Data for 'No' in Account Age:**
  - Total samples: 3.
  - Preferred 'Yes': 1.
  - Preferred 'No': 2.
- **Gini Index Calculation:**

$$G_{\text{No}} = 1 - \left(\left(\frac{1}{3}\right)^2 + \left(\frac{2}{3}\right)^2\right) = 0.44$$

- **Weighted Gini Index:**

$$G_{\text{Account Age}} = \left(\frac{5}{8} \times 0.48\right) + \left(\frac{3}{8} \times 0.44\right) = 0.46$$

- **Interpretation:**
  - This value represents the impurity of the dataset when split by 'Account Age'.

Calculating Gini Index for Violations and Balance

Account age > 1	Violations > 0	Balance > $50,000	Preferred
Yes	No	Yes	No
Yes	Yes	Yes	No
No	No	Yes	No
Yes	No	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
No	No	Yes	Yes
No	Yes	Yes	No

Gini Indices:

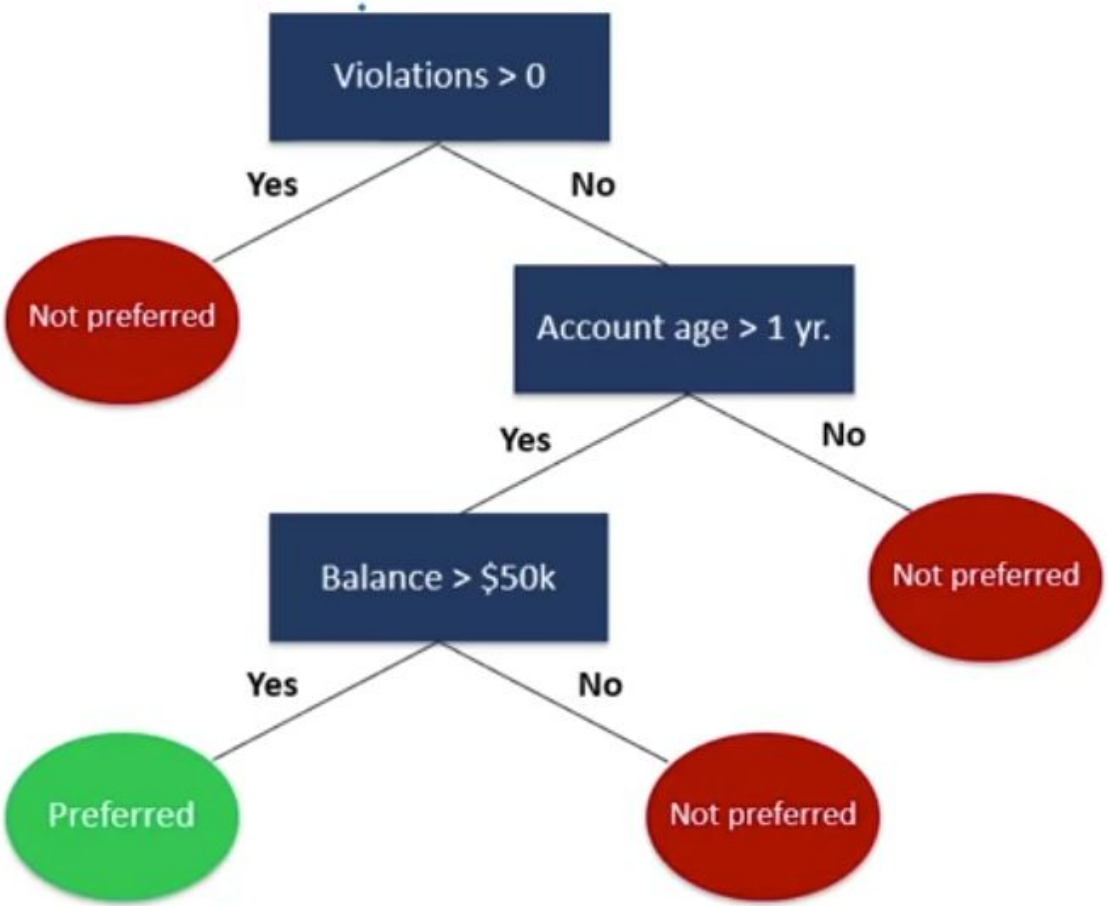
- **Violations**: 0.38
- **Balance**: 0.43
- **Account Age**: 0.46

Comparing Gini Indices of All Features to get decision:

- Since **Violations** has the lowest Gini Index, it **becomes** the **root decision node.**

Building the Decision Tree

- **Root Node**: 'Violations'
- **Split Criteria:**
  - Yes: Customer has violations.
  - No: Customer has no violations.
- **Next Steps**:
  - For each subset resulting from the split on 'Violations', calculate the Gini Index for remaining features.
  - Select the feature with the lowest Gini Index as the next decision node.
- **Process Continues**:
  - Repeat until all nodes are pure (Gini Index = 0) or stopping criteria are met.



Determine the next best feature to split on.

Remaining Features: 'Account Age', 'Balance'

Recalculate Gini Indices for Remaining Features

- Compare Gini Indices:
  - Assume:
    - $G_{\text{Account Age}} = 0.4$
    - $G_{\text{Balance}} = 0.43$
- Select '**Account Age**' as it has a lower Gini Index.

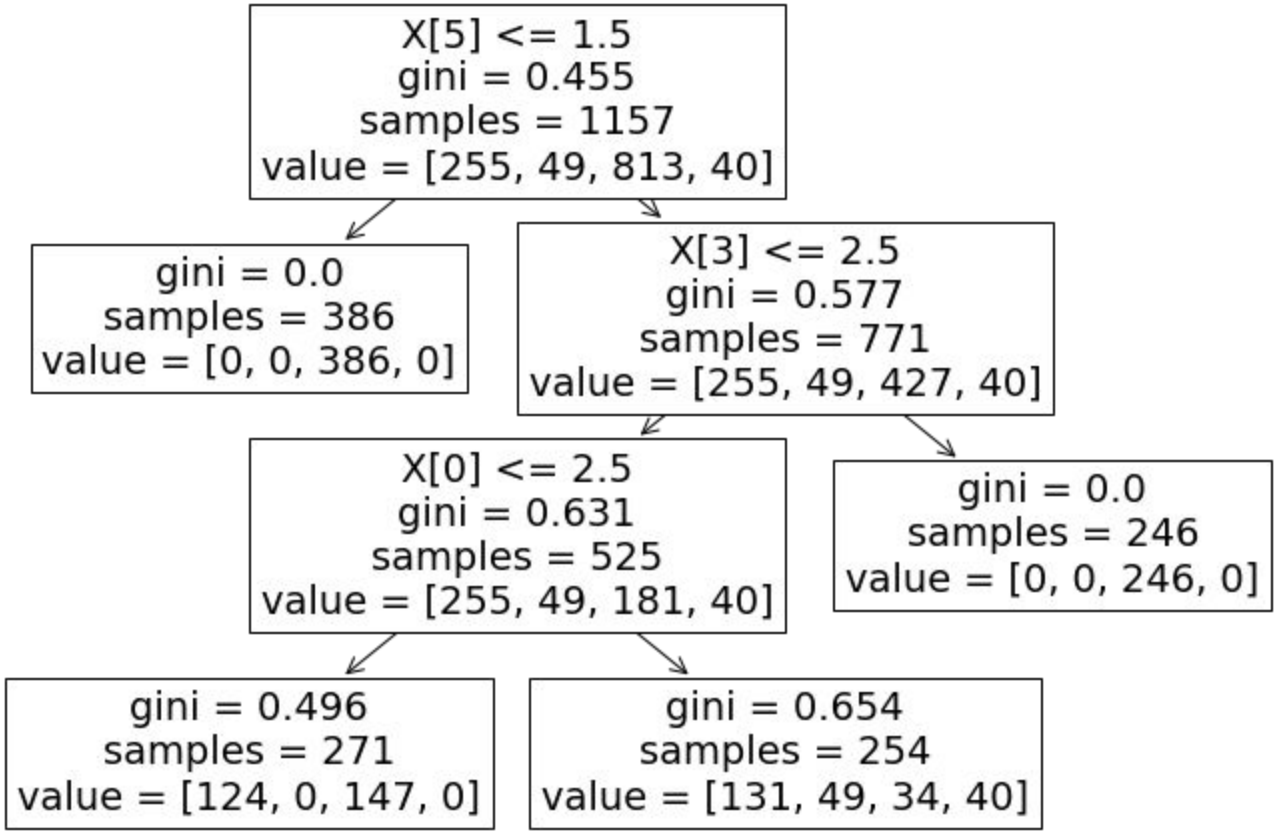

X[5] <= 1.5
gini = 0.455
samples = 1157
value = [255, 49, 813, 40]
gini = 0.0
samples = 386
value = [0, 0, 386, 0]
X[3] <= 2.5
gini = 0.577
samples = 771
value = [255, 49, 427, 40]
X[0] <= 2.5
gini = 0.631
samples = 525
value = [255, 49, 181, 40]
gini = 0.0
samples = 246
value = [0, 0, 246, 0]
gini = 0.496
samples = 271
value = [124, 0, 147, 0]
gini = 0.654
samples = 254
value = [131, 49, 34, 40]

Table 1 Comparisons between different Decision Tree Algorithms

Algorithms	ID3	C4.5	C5.0	CART
Type of data	Categorical	Continuous and Categorical	Continuous and Categorical, dates, times, timestamps	continuous and nominal attributes data
Speed	Low	Faster than ID3	Highest	Average
Pruning	No	Pre-pruning	Pre-pruning	Post pruning
Boosting	Not supported	Not supported	Supported	Supported
Missing Values	Can't deal with	Can't deal with	Can deal with	Can deal with
Formula	Use information entropy and information Gain	Use split info and gain ratio	Same as C4.5	Use Gini diversity index

Hyperparameter Tuning in CART Algorithm

- **What are Hyperparameters?**

  They are parameters that we can modify to influence how the machine learning model is built.

- **Why do we tune Hyperparameters in the CART Algorithm?**

  To control how the nodes are split and how large the tree grows.

  Because CART is prone to overfitting, tuning helps us avoid this issue.

Importance of Hyperparameter Tuning

- **Model Generalization**

  Tuning hyperparameters helps ensure the model can make good predictions on new data.

- **Overfitting**
  - Without tuning, the tree can learn too many details from the training data and may not generalize well.

Key Hyperparameters in Scikit-learn:

- `max_depth`
- `min_samples_split`
- `min_samples_leaf`
- `splitter`

Hyperparameter (`max_depth`)

- **What it does:**
  Controls the maximum depth of the tree.
- **Default behavior:**
  If `max_depth` is not set, the tree will keep splitting until the leaves are pure (Gini Index = 0).
- **Why tune it?**
  To control how large the tree grows and how many splits are allowed.
  Helps prevent the tree from **becoming too complex** (overfitting).
- **How to find the right value?**
  Evaluate model performance with different `max_depth` values and choose the best one.

Hyperparameters 2 & 3 - `min_samples_split` & `min_samples_leaf`

- **`min_samples_split`:**
  The minimum number of samples required to split a node.
  Default value: 2.
- **`min_samples_leaf`:**
  The minimum number of samples that a leaf node must have.
  Default value: 1.
- **Why tune them?**
  Increasing these values can reduce the likelihood of overfitting.
  However, be careful to avoid underfitting (the model becomes too simple).
- **Tips:**
  Try different values and observe their effect on model performance.
  Check if the model starts to lose predictive power.

Hyperparameter 4 - `splitter`

What it does:

Determines the strategy used to split nodes.

Possible values:

- `"best"`: Uses the Gini Index to find the best split.
- `"random"`: Chooses the feature randomly for splitting.

Pros & Cons:

- `"best"`:
  - Provides the best split in terms of purity.
  - Can be slower because it calculates the Gini Index for all options.
- `"random"`:
  - Faster because it doesn’t calculate for all options.
  - Helps avoid overfitting but may not give the best possible split.

```python
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    criterion='gini'
)
```

- Don’t go deeper than **5 levels** (max depth).
- Only **split** if you have at least **10 samples**.
- Each **leaf node** must have at least **5 samples**.
- Try √ **(square root) of the number of features** at each split.
- **Evaluate the splits** using the **Gini Index**.

Avoiding Overfitting with Hyperparameters

- **How does overfitting occur?**
  - When the model learns too many details and noise from the training data.
- **Role of Hyperparameters:**
  - Control the complexity of the tree and the number of splits.
  - Balance between model complexity and its ability to generalize.

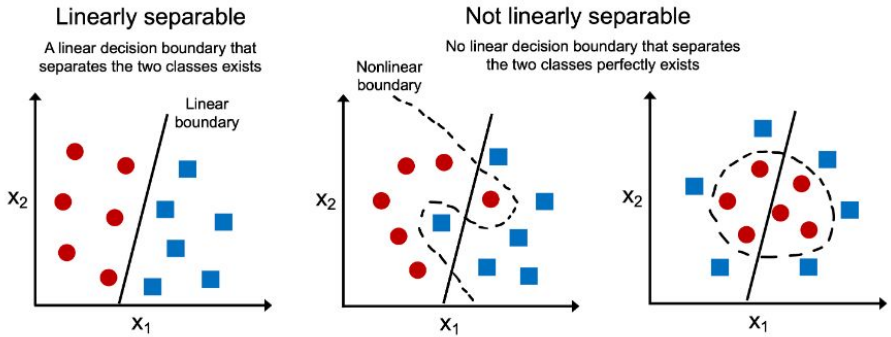
Strategies:

- Set `max_depth` to limit tree depth.
- Increase `min_samples_split` and `min_samples_leaf` carefully.
- Experiment with the `"random" splitter` in certain cases.

Assumptions For Logistic Regression (LR) and Decision Tree (DT)

- **Logistic Regression (LR)**
  - Assumes a **linear relationship** between features and the log-odds of the outcome.
  - Assumes independence between features (no multicollinearity).
  - Works best if data is relatively clean and well-distributed.
  - Outliers sensitivity
- **Decision Tree (DT)**
  - **Non-parametric** → makes **no assumption** about linearity or distribution.
  - Splits data into rules based on thresholds.
  - Very prone to **overfitting** if the tree grows too deep.
  - Outliers sensitivity

Logistic Regression = constrained model (needs assumptions).
Decision Tree = flexible model (fewer assumptions).



Decision Tree – Problems & Solutions

Main Problems	Practical Solutions
Overfitting – tree memorizes training data instead of generalizing	Use **Pruning** (reduce unnecessary branches)
Instability / High Variance – small data changes lead to big model changes	Use **Ensembles** (Random Forest, Gradient Boosting)
High Complexity – deep trees are hard to interpret	Set limits (max depth, min samples per leaf)

Pruning

Pruning

- Pruning is a technique that reduces the size of a decision tree by removing branches of the tree which provide little predictive power
- It is a **regularization** method that reduces the complexity of the final model, thus reducing overfitting
  - Decision trees are prone to overfitting!

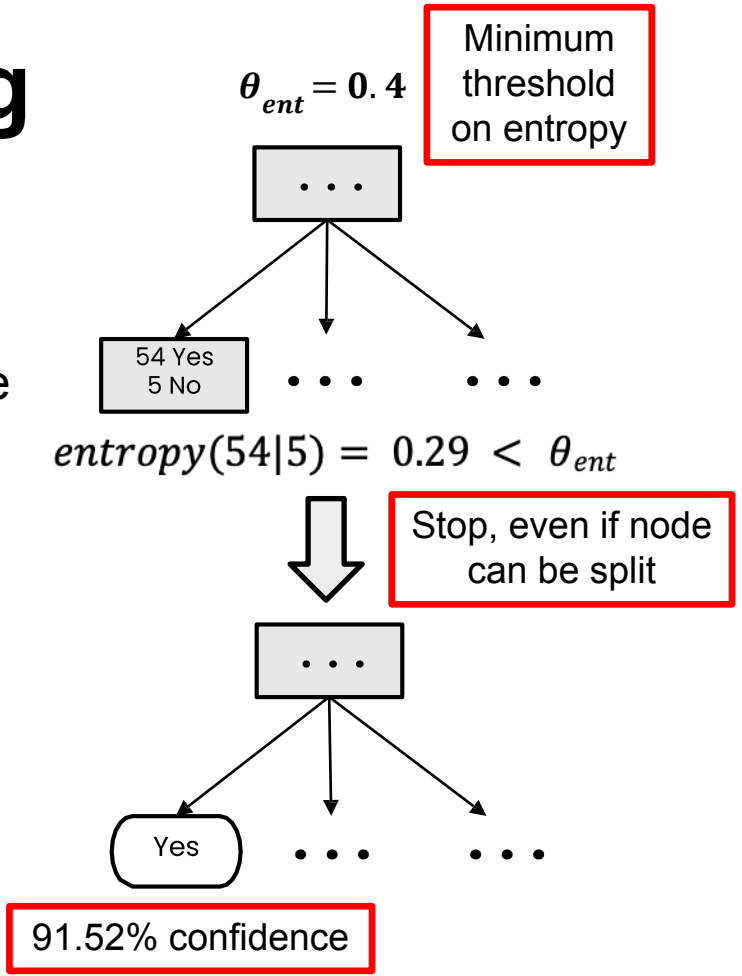
- Pruning methods:
  - Pre-pruning: Stop the tree building algorithm before it fully classifies the data
  - Post-pruning: Build the complete tree, then replace some non-leaf nodes with leaf nodes if this improves validation error

Pre-pruning

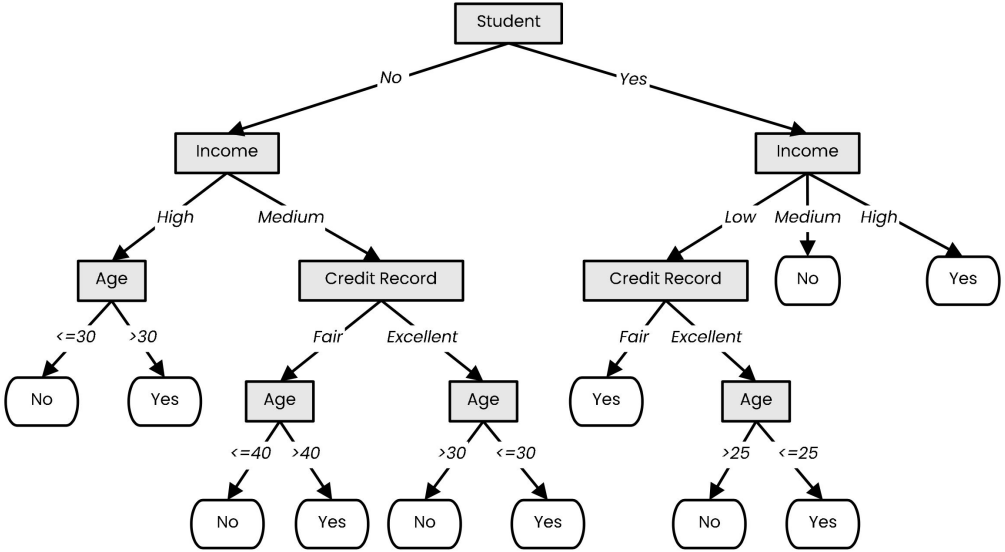
- Pre-pruning implies early stopping:
  - If some condition is met, the current node will not be split, even if it is not 100% pure
  - It will become a leaf node with the label of the majority class in the current set

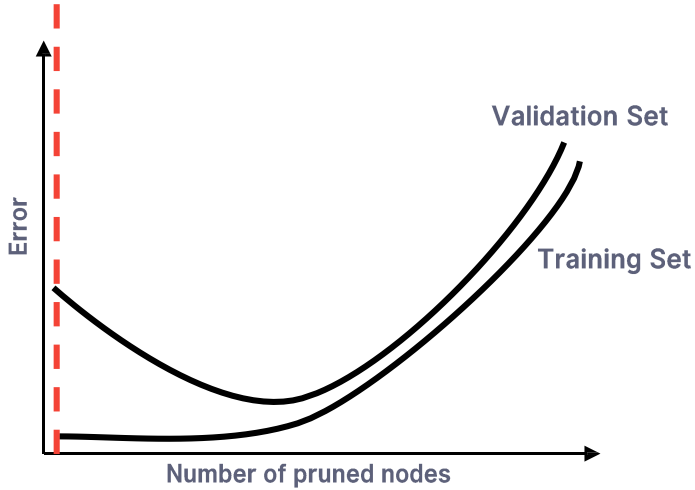
(the class distribution could be used as prediction confidence)

- Common stopping criteria include setting a threshold on:
  - Entropy (or Gini Impurity) of the current set
  - Number of samples in the current set
  - Gain of the best-splitting attribute
  - Depth of the tree



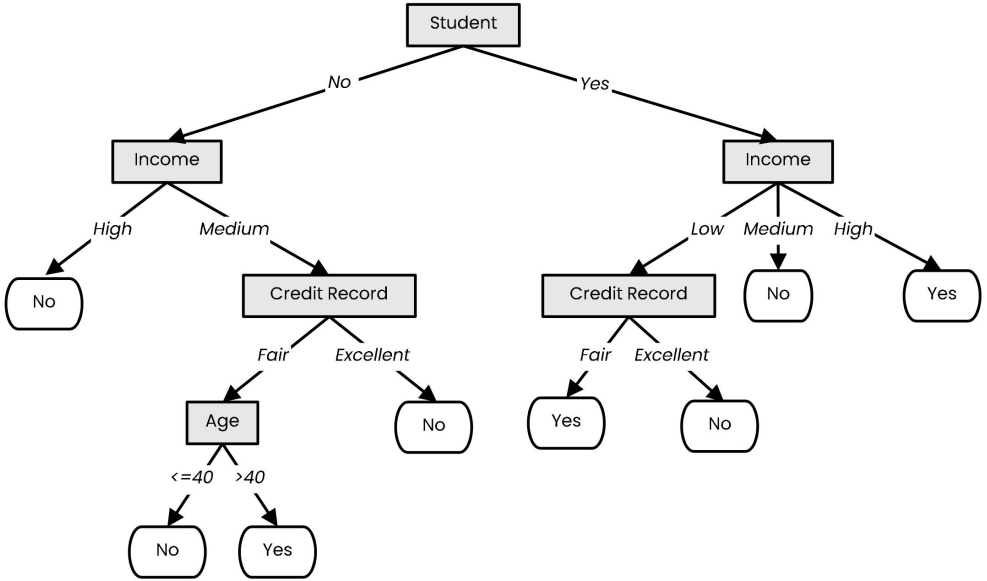
Post-pruning

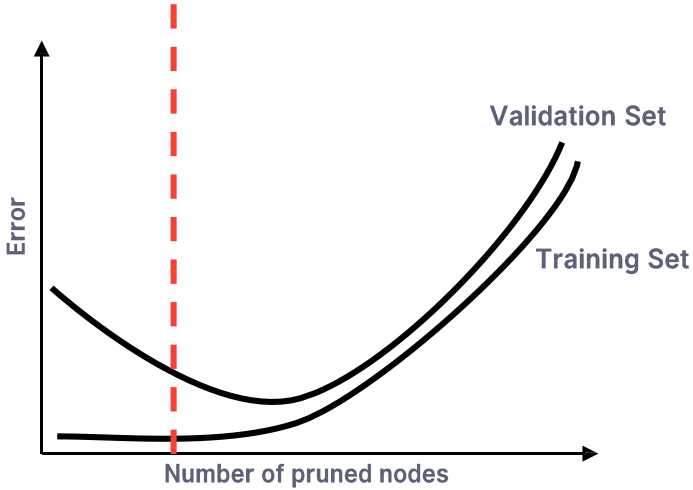




- Prune nodes in a bottom-up manner, if it decreases validation error

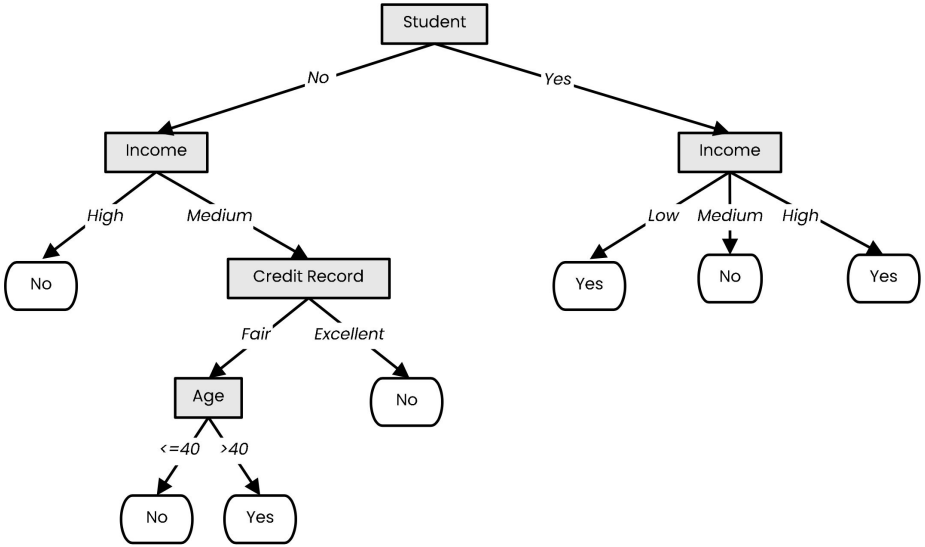
Post-pruning



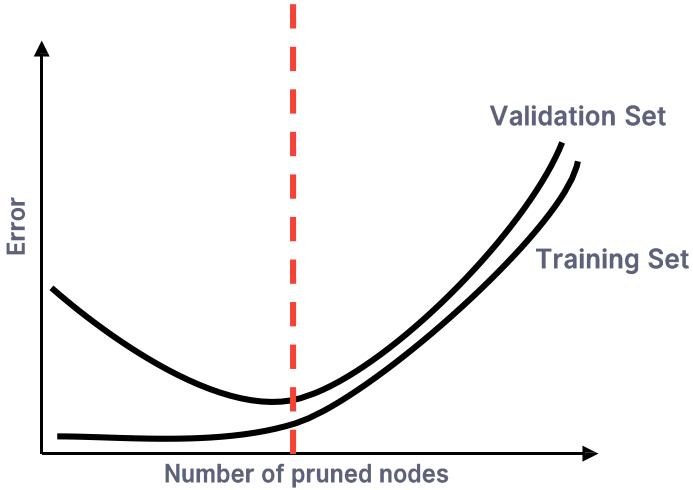


- Prune nodes in a bottom-up manner, if it decreases validation error

Post-pruning



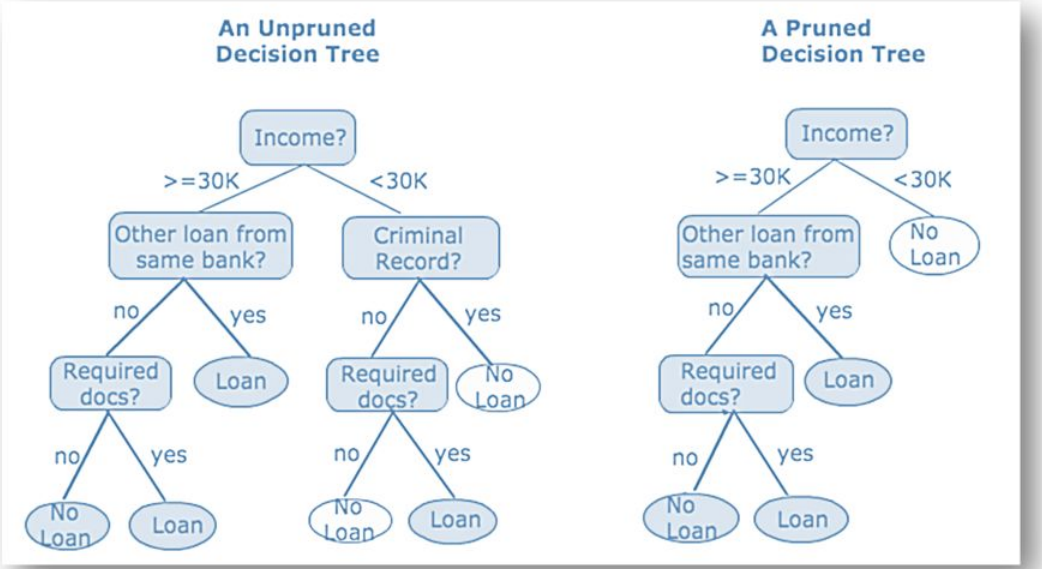
- Prune nodes in a bottom-up manner, if it decreases validation error



Self study

Pruning decision trees

1. **Reduced Error Pruning**
2. **Cost Complexity Pruning**
3. **Minimum Error Pruning**



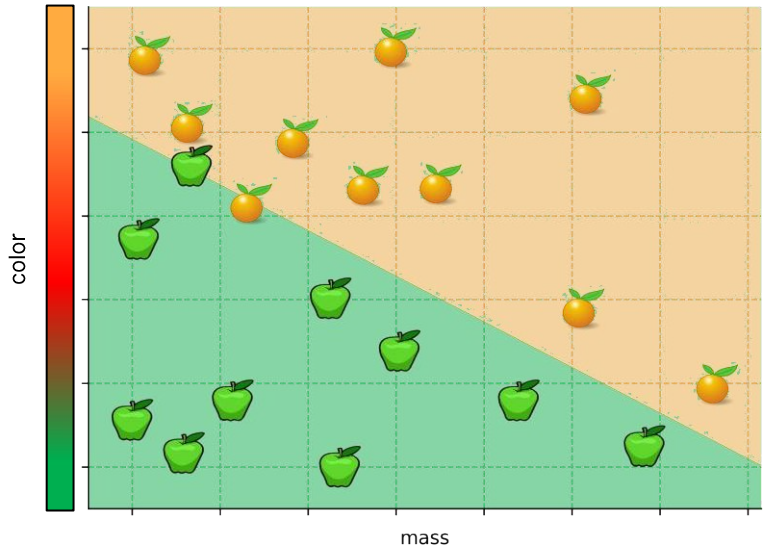
The process of IG-based pruning requires us to identify "twigs", nodes whose **children** are all **leaves**.

"Pruning" a twig removes all of the leaves which are the children of the twig, and makes the **twig** a **leaf**.

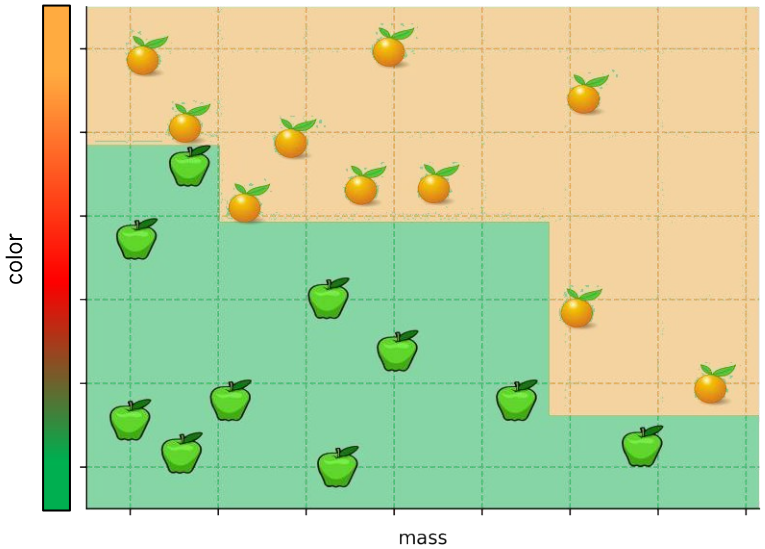
Decision Boundaries

- Decision trees produce non-linear decision boundaries

Logistec Regression



Decision Tree



Logistic Regression vs Decision Tree

Aspect	Logistic Regression (LR)	Decision Tree (DT)
Relationship with Data	Assumes a linear relationship between features and outcome	Handles both linear and non-linear relationships
Categorical Data Handling	Requires Encoding (e.g., Yes/No → 0/1)	Natively handles categorical variables without encoding, making preprocessing easier
Interpretability	Clear mathematically (coefficients = feature weights)	Clear visually (decision paths in a tree)
Stability	More stable with small changes in data	Can change significantly with small data variations
Performance on Large & Complex Data	Efficient and fast, but **struggles with complex/non-linear** patterns unless features engineered	Excels in capturing **complex, high-dimensional, non-linear patterns**, automatically handling feature interactions and variable selection

https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial