

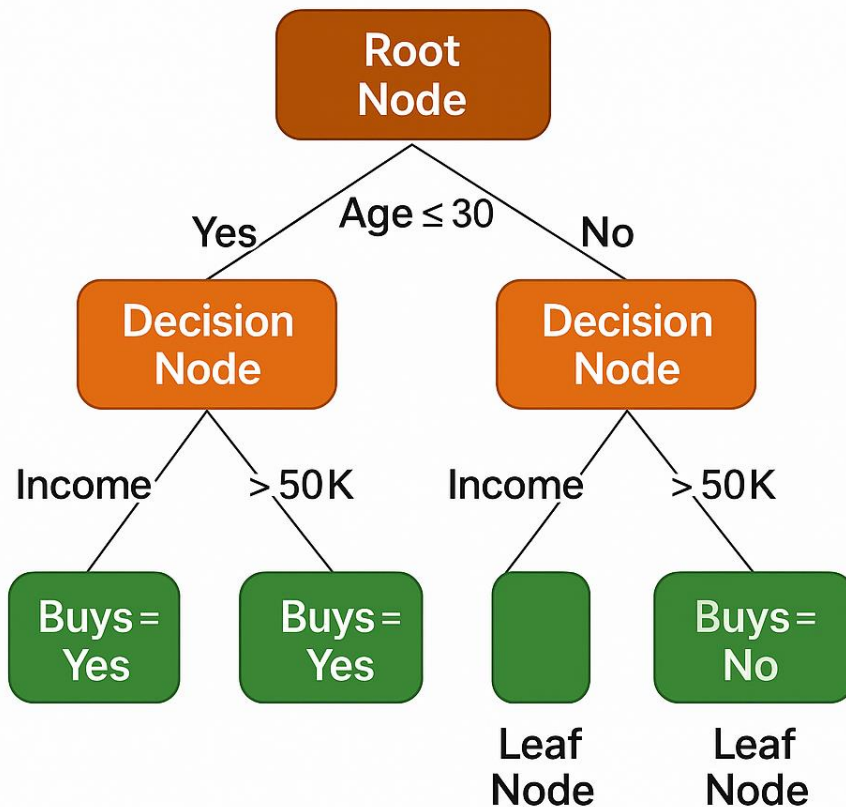
Decision Tree Notes with Diagrams

Understanding Purity and Impurity

In a Decision Tree, every node represents a group of data points. Each node can be:

- Pure → mostly or entirely one class
- Impure → mixed classes

Decision Tree



☐ Pure Node Example: Most data belongs to a single class (e.g., 83% non-default).

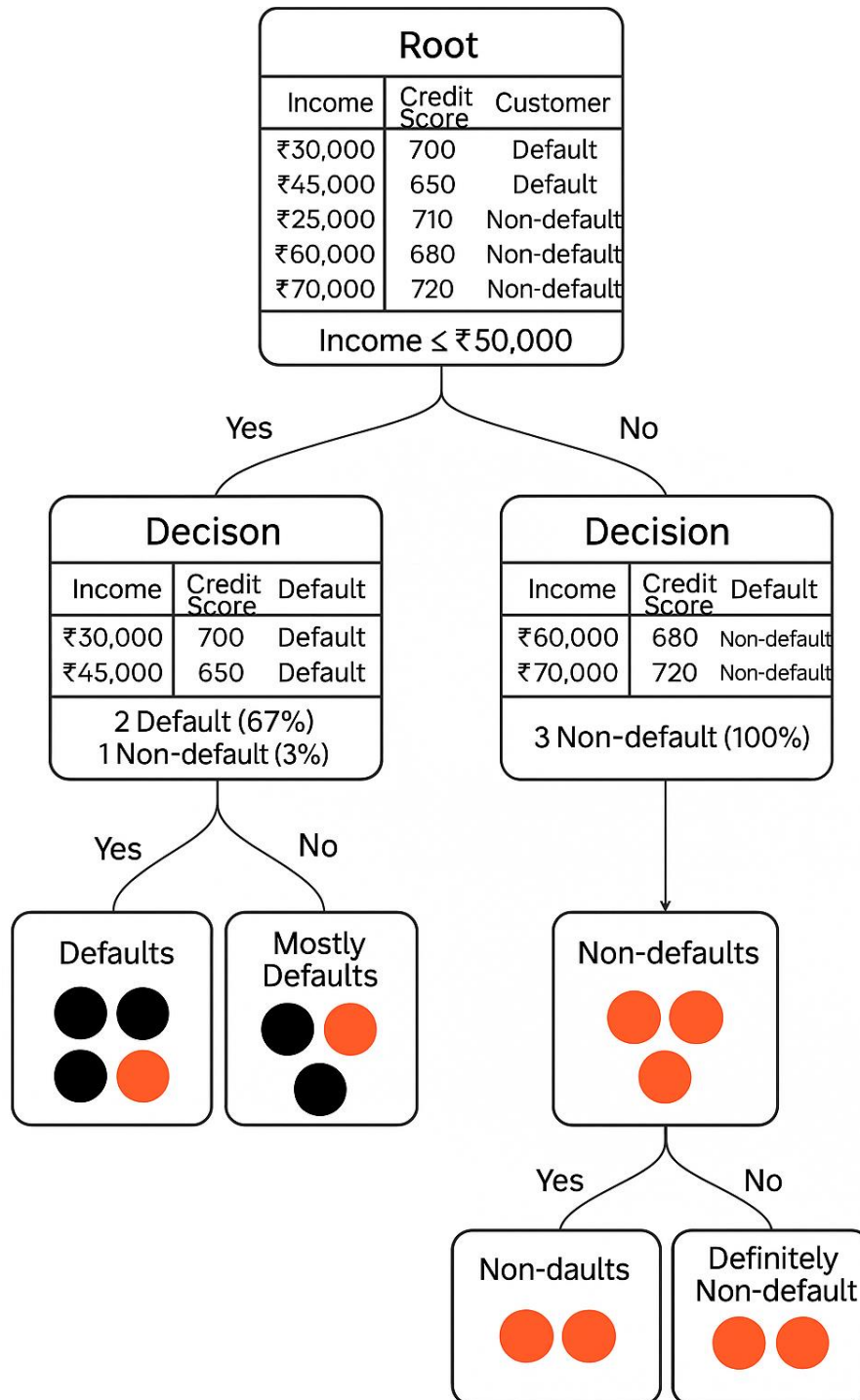
☐ Impure Node Example: Classes are evenly mixed (e.g., 50% default, 50% non-default).

☐ How Decision Trees Handle Purity:

1. Root Node – Starts impure (mixed data).
2. Decision Nodes – Each split reduces impurity.

3. Leaf Nodes – Ideally pure (each contains one class).

Example: Predicting Loan Default



Root node uses Income \leq ₹50,000 to split data. Left branch is still mixed, so a second split occurs on Credit Score \leq 680.

Decision Tree Parameters

🔗 Step-by-step Explanation

1🔗 Model Initialization

```
rf = RandomForestClassifier(random_state=42)
```

🔗 Creates a Random Forest model.

`random_state=42` → keeps results the same every time (for reproducibility).

2🔗 Parameter Grid (Things we're testing)

```
rf_param_grid = {  
    'n_estimators': [100, 200, 300],  
    'criterion': ['gini', 'entropy'],  
    'max_depth': [5, 10, 15, 20, None],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 4],  
    'max_features': ['sqrt', 'log2', None]  
}
```

🔗 Random Forest Hyperparameters — Detailed and Simple Explanation

🔗 1🔗 `n_estimators`

- 🔗 It means the **number of trees** in your Random Forest.

- A Random Forest is literally a **collection of many Decision Trees**.
- The more trees you have:
 - ✓ The more stable and accurate the predictions (because many trees vote together).
 - ✗ But it takes longer to train and predict.

Example:

If `n_estimators=100`, the model builds 100 trees and takes a majority vote of all their predictions.

If 60 trees say “diabetic” and 40 say “not diabetic”, the model will predict “diabetic”.

🔍 **Tip:** More trees → less variance → better generalization (up to a point). Usually, 100–300 is good.

🔍 2🔍 criterion

- This defines **how the tree decides where to split** the data at each step (or question).
- It measures **purity** of a node (how mixed the classes are).

There are two main options:

Criterion	Meaning
gini	Measures how often a randomly chosen element would be incorrectly classified. Lower = purer.
entropy	Based on information gain (like in information theory). Measures the randomness (disorder).

Example:

If a node has 8 diabetics and 2 non-diabetics, that node is quite pure (mostly one class).

The algorithm will pick a split (question) that makes both resulting child nodes even more pure.

🔗 **Tip:**

Both *gini* and *entropy* give very similar results.

Gini is slightly faster; entropy is a bit more mathematical.

🔗 3 🔗 **max_depth**

- This controls **how deep each decision tree can go** (how many “questions” it can ask).
- Think of it like: each tree starts from the root and keeps splitting (“Is glucose > 120?”, “Is BMI > 25?” etc.).
- Each level = one question or decision.
- The deeper it goes, the more specific and fine-grained the rules become.

Example:

- `max_depth=3` → each tree can ask up to 3 questions before giving a prediction.
- `max_depth=None` → the tree can grow until all leaves are pure (every sample is perfectly classified).

Why it matters:

- Small depth → underfitting (too simple, misses patterns).
- Large depth → overfitting (memorizes the training data).

🔗 **Tip:** Try depths between **5-15** usually.

🔗 4 🔗 **min_samples_split**

- Minimum number of samples required **to split an internal node** (make a new branch).
- Prevents the tree from making splits based on very few samples (which can cause overfitting).

Example:

- If `min_samples_split=2`, even 2 samples can trigger a new split (very sensitive).
- If `min_samples_split=10`, at least 10 samples must be in a node before it can split further — this keeps trees more general and avoids noise-based splits.

🔍 **Tip:** Higher values = smoother, less complex model.

🔍 5 🔍 **min_samples_leaf**

- Minimum number of samples required **in each leaf node** (final output node).
- A leaf is the final answer — e.g. “Diabetic” or “Non-Diabetic”.

Example:

- If `min_samples_leaf=1`, leaves can have even a single sample (can overfit).
- If `min_samples_leaf=5`, every leaf must have at least 5 samples — it makes leaves broader and more stable.

🔍 **Tip:**

For classification problems, values between **1–4** usually work best.

Increasing this helps avoid extremely specific rules that fit just one data point.

🔍 6 🔍 **max_features**

- Controls **how many features (columns)** are considered when finding the best split in each tree.
- Random Forests randomly select a subset of features to make trees **diverse** and **less correlated**.

Value	Meaning
'sqrt'	Uses $\sqrt{(\text{total features})}$. Default for classification.
'log2'	Uses $\log_2(\text{total features})$.
None	Uses all features.

Example:

If you have 9 features:

- 'sqrt' → 3 features will be randomly chosen at each split.
- 'log2' → about 3 features.
- None → all 9 features will be used at every split.

📌 **Tip:** 'sqrt' is the most common — it keeps trees diverse but still accurate.


✓ In One Line Summary

Parameter	Meaning
n_estimators	How many trees are in the forest.
criterion	How each split is chosen (Gini or Entropy).

max_depth	How deep the tree can go (how many questions it can ask).
min_samples_split	Minimum samples needed to split a node.
min_samples_leaf	Minimum samples at a leaf (final decision node).
max_features	Number of features to consider for each split.

3 Cross-Validation Setup

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

 Splits data into 5 parts (folds) and ensures both classes (0 & 1) are balanced in each fold. Helps test model performance more fairly.

4 GridSearchCV (The Auto-Tuner)

```
rf_grid = GridSearchCV(
    estimator=rf,                # model to tune
    param_grid=rf_param_grid,    # parameters to test
    scoring='f1',                # performance metric (F1 balances
precision & recall)
    cv=cv,                      # 5-fold cross validation
    n_jobs=-1,                  # use all CPU cores for faster
training
    verbose=1                   # show progress while running
)
```

 **GridSearchCV** tests **all parameter combinations** and finds the one with the **best F1-score**.

5. Fit and Print Best Parameters

```
rf_grid.fit(x_train_bal, y_train_bal)

print('RF best params', rf_grid.best_params_)

best_rf = rf_grid.best_estimator_
```

✓ This trains all combinations, selects the best, and saves it as `best_rf`.

6. In One Line

GridSearchCV automatically tests many Random Forest settings (tree size, split rules, etc.) using cross-validation and picks the one that gives the **best F1-score** for balanced performance.

🔍 Real-Life Use Case: Loan Approval Prediction (Banking Sector)

🔍 Objective:

A bank wants to predict whether a **loan applicant should be approved or not** based on past customer data.

🔍 Dataset Features (Independent Variables):

- **Income** – Applicant's monthly income
- **Credit Score** – Creditworthiness of the customer
- **Employment Type** – Salaried / Self-employed
- **Loan Amount** – Requested loan size
- **Previous Defaults** – Number of times customer has defaulted before

Target (Dependent Variable):

- **Loan Status:** Approved ✓ or Rejected ✗
-

🔍 How the Decision Tree Works:

1. Root Node:

- The algorithm checks all features and chooses the one that gives the best split (highest information gain).
- Example:
 - 🔍 *"Credit Score > 700?"*

2. First Split:

- If **Yes** → High chance of approval.
- If **No** → Check next feature (like income).

3. Second Split:

- *"Income > ₹40,000?"*
 - Yes → Approve
 - No → Reject
-

🔍 Outcome:

- The tree helps the bank **automatically decide** loan approvals based on past patterns.
 - Every new application passes through this logical flow, ensuring consistent, data-driven decisions.
-

The algorithm chooses the **split that reduces impurity the most**, meaning it makes the child nodes as **pure** as possible.

One Line Summary:

Both Gini and Entropy tell the tree how "impure" a node is — lower value means purer, a better split.