Decision tree

🌳 What is a Decision Tree?

A **Decision Tree** is a **supervised learning algorithm** used for both **classification** and **regression**.

It works like a **flowchart**:

- Each **internal node** represents a *test* on a feature (e.g., "Age > 30?")
- Each **branch** represents the *outcome* of the test (Yes/No)
- Each **leaf node** represents a *final prediction* (e.g., “Will buy = Yes”)

🧠 Intuitive Example

Imagine you’re deciding whether to play tennis:

Weather	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Overcast	Mild	Normal	Strong	Yes
Rain	Cool	Normal	Weak	Yes

A Decision Tree might look like:

```
        [Outlook?]
       /   |    \
    Sunny  Overcast  Rain
     /       |      \
 [Humidity?]   Yes   [Wind?]
  /  \              /   \
High  Normal       Weak  Strong
 No    Yes          Yes   No
```

So, to decide if you’ll play tennis:

- Check the **Outlook** first
- Then possibly **Humidity** or **Wind**
- Follow the branches until you reach a **decision (leaf)**

⚙️ How Decision Trees Work

At each step, the algorithm asks:

> "Which feature best splits the data into pure groups?"

1️⃣ Start at the root

All training data is in one node.

2️⃣ Choose the best feature to split

We evaluate all possible features and pick the one that best separates the classes.

This is done using **splitting criteria** like:

- **Gini Impurity**
- **Entropy / Information Gain**
- **Variance Reduction** (for regression trees)

3️⃣ Split the dataset

Divide the data into branches according to feature values.

4️⃣ Repeat recursively

For each branch, keep splitting until:

- All samples in a node belong to the same class (pure), or
- A stopping criterion is reached (like max depth).

5️⃣ Assign labels

Each leaf node takes the **majority class** (classification) or **average value** (regression).

📈 Splitting Metrics

Metric	Used In	Meaning
Entropy	ID3 Algorithm	Measures impurity (disorder) in a dataset. Lower = better.
Information Gain	ID3, C4.5	How much entropy decreases after a split.
Gini Impurity	CART Algorithm	Measures how often a random sample would be misclassified. Lower = better.

Example:

$Gini = 1 - \sum_i p_i^2$

where p_i is the fraction of samples of class i in a node.

🧩 Classification vs Regression Trees

Type	Target	Example	Output
Classification Tree	Categorical	“Yes/No”, “Spam/Not Spam”	Class label
Regression Tree	Continuous	“House Price”, “Temperature”	Numeric value (average in leaf)

💡 Advantages

Advantage	Description
Easy to interpret	Works like a set of human decision rules
No scaling required	Works with unnormalized data
Handles both categorical and numerical features	Very flexible
Nonlinear boundaries	Can learn complex decision regions
Feature importance	You can see which features matter most

⚠️ Disadvantages

Disadvantage	Description
Overfitting	Deep trees can perfectly fit training data but perform poorly on new data
Unstable	Small data changes can cause a very different tree
Greedy	Chooses best split locally, not globally optimal
Bias toward features with more levels	Features with many possible values may dominate

✅ Solution: Use **Ensemble Methods**:

- **Random Forests** → average multiple trees
- **XGBoost / Gradient Boosting** → build trees sequentially to reduce errors

🔍 Example Use Cases

Area	Use
🧾 Finance	Credit risk analysis, loan approval
🧬 Healthcare	Disease diagnosis based on symptoms
🏠 Real Estate	Predicting house prices (Regression Tree)
🛒 Retail	Customer segmentation, product recommendation
📧 Email	Spam detection

🧠 Summary

Concept	Decision Tree Summary
Type	Supervised learning (classification & regression)
Structure	Tree-like model of decisions
Key Idea	Split data recursively based on features to create pure nodes
Splitting Criteria	Gini, Entropy, Variance

Concept	Decision Tree Summary
Output	Majority class or average value
Pros	Interpretable, flexible, no scaling needed
Cons	Overfitting, unstable, greedy

🌳 Decision Tree — Loss Function & Parameter Selection

A **Decision Tree (DT)** is a *supervised learning* algorithm that splits data based on questions about the features to minimize some **loss (or impurity)** — making the groups (nodes) as "pure" as possible.

🧮 1. Loss Function in Decision Trees

The **loss function** in Decision Trees depends on the **type of problem**:

A. For Classification

We measure how *impure* (or mixed) a node is using an **impurity metric**.

The **goal**: at every split, choose the feature and threshold that **reduce impurity the most**.

Common impurity measures:

1. **Gini Impurity** (default in sklearn)

   $Gini = 1 - \sum_{k=1}^{K} p_k^2$

   where (p_k) = proportion of class (k) in the node.

   Intuition:

   Gini is low when most samples in a node belong to a single class.

   → A pure node (only one class) has (Gini = 0).

1. **Entropy (Information Gain)**

   $Entropy = -\sum_{k=1}^{K} p_k \log_2(p_k)$

Information Gain (IG) = reduction in entropy after a split:

$$IG = Entropy(parent) - \sum_i \frac{n_i}{n_{total}} Entropy(child_i)$$

Intuition:

Measures the "uncertainty" in a node. Splitting should maximize IG → more certain/pure nodes.

✅ **In short:**

> The Decision Tree loss function = weighted impurity of all leaves
>
> → choose splits that **minimize this impurity**.

B. For Regression

We measure how *different* the target values are within a node.

Common measures:

1. **Mean Squared Error (MSE):**

   $$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y})^2$$

2. **Mean Absolute Error (MAE):**

   $$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}|$$

Goal: choose splits that reduce the variance (spread) of target values.

⚙️ 2. Choosing Parameters for Best Results

Decision Trees are **powerful but prone to overfitting**.

So, tuning **hyperparameters** is key to control complexity.

Parameter	Meaning	Effect
`max_depth`	Maximum depth of tree	Prevents overfitting (limit complexity)
`min_samples_split`	Minimum samples required to split a node	Larger → simpler tree

Parameter	Meaning	Effect
`min_samples_leaf`	Minimum samples per leaf	Prevents small noisy leaves
`max_features`	Number of features to consider at each split	Adds randomness (used in ensembles)
`criterion`	Loss function: `'gini'`, `'entropy'`, `'mse'`	Choose impurity type
`max_leaf_nodes`	Limit number of leaves	Balances bias vs variance

✅ Best Practices for Choosing Parameters

1. **Start simple:**
   - Set `max_depth` small (e.g. 3–5) to avoid overfitting early.
2. **Use Cross-Validation:**
   - Tune hyperparameters using GridSearchCV or RandomizedSearchCV.
3. **Look at validation accuracy:**
   - If training accuracy ≫ validation accuracy → overfitting.
   - Increase regularization (reduce depth, increase min_samples_leaf).
4. **Visualize the tree:**
   - Helps understand splits & features contributing most.

🎯 Example (Classification)

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

params = {
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}

dt = DecisionTreeClassifier(random_state=42)
```

```
grid = GridSearchCV(dt, params, cv=5)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
```

🧠 Intuition Recap:

- **Loss function:** measures impurity (how mixed the node is).
- **Goal:** choose splits that reduce impurity (or variance).
- **Hyperparameters:** control model complexity → balance bias & variance.

How Decision Trees are **built from scratch** 🌳

🧩 Step 0: The Big Picture

A **Decision Tree** is built **top-down**, starting from the **root node**, by **recursively splitting** the dataset into smaller and purer groups.

Each **split** is chosen to *maximize information gain* (or *reduce impurity/variance*).

We stop splitting when:

- The node is *pure enough* (all samples belong to one class), or
- We reach a *stopping criterion* (e.g., max depth, min samples, etc.)

🌱 Step 1: Start With the Whole Dataset

Let's say you have a dataset:

Age	Income	Buys Product (Y)
25	Low	No
35	High	Yes
40	Medium	Yes

Age	Income	Buys Product (Y)
22	Low	No
30	High	Yes

This is your **root node** — it contains all training examples.

🔍 Step 2: For Each Feature, Try All Possible Splits

For each feature (e.g. *Age*, *Income*), the algorithm tries:

- Every possible **threshold** or **category** to split on.
- Computes **impurity** of resulting child nodes.

Example:

If you split on `Age < 30`, you'll have two groups:

- Left node: Age < 30 → [25, 22] → both "No"
- Right node: Age ≥ 30 → [35, 40, 30] → all "Yes"

This split perfectly separates the classes → **pure nodes** ✅

⚖️ Step 3: Measure How Good Each Split Is

We compute impurity before and after the split.

For classification:

- Use **Gini Impurity** or **Entropy**.

For example,

If parent node has impurity (I_p),

and children have (I_L) and (I_R),

then the **Information Gain (IG)** is:

$$IG = I_p - \left(\frac{N_L}{N} I_L + \frac{N_R}{N} I_R\right)$$

Pick the feature and threshold that **maximize IG** (i.e., most purity gained).

🌳 Step 4: Split the Dataset

Perform the best split → this becomes your **decision rule**:

> "If Age < 30 → predict No, else → predict Yes."

Now you have two branches — repeat the same process **recursively** for each.

🧭 Step 5: Stopping Conditions

Stop when any of these are true:

- All samples in a node belong to one class.
- No feature provides improvement.
- You reached a parameter limit (`max_depth`, `min_samples_split`, etc.)

Each final node (leaf) gets a **prediction**:

- For classification → majority class.
- For regression → mean of target values.

🔢 Step 6: Output the Tree

You now have a full **tree structure** of decision rules.

Example:

```
If Age < 30:
    → Predict: No
Else:
    → Predict: Yes
```

🧠 Intuition Recap

Step	What Happens	Why
Start	Begin with all data	Root node
Split	Try all features & thresholds	Find best question

Step	What Happens	Why
Evaluate	Compute impurity/information gain	Choose best split
Recurse	Repeat for sub-nodes	Build deeper structure
Stop	When pure or small nodes	Prevent overfitting
Predict	Leaf gives class or value	Decision made

⚙️ In sklearn (simple code)

```
from sklearn.tree import DecisionTreeClassifier

# Create and train model
dt = DecisionTreeClassifier(
    criterion='gini',
    max_depth=3,
    random_state=42
)
dt.fit(X_train, y_train)

# Predict
y_pred = dt.predict(X_test)
```

🌱 The Essence of a Decision Tree

A **Decision Tree** is a *rule-based*, *nonlinear* model that tries to partition the feature space into smaller regions where the output is as pure as possible.

Example:

> "If income > 80k and age < 35, predict YES — otherwise NO."

It literally "asks questions" until it reaches a decision.

This makes it **intuitive** and **interpretable** — but also prone to overfitting.

⚙️ Why We Might Prefer a Decision Tree (Pros)

✅ Pros of Decision Trees

1. **Easy to interpret and visualize**
   - You can literally read the logic as if-else rules.
   - Non-math stakeholders (like business or domain experts) can understand model decisions.
2. **No need to scale or normalize features**
   - Trees split on thresholds (e.g., x > 10), so feature scaling doesn't affect performance.
3. **Handles both numerical and categorical data**
   - Works directly with categorical variables (after encoding if needed).
4. **Captures nonlinear relationships**
   - Unlike Logistic Regression, Decision Trees can model curved or complex decision boundaries.
5. **Automatic feature selection**
   - The splitting process inherently picks the most informative features first.
6. **Robust to irrelevant features**
   - Unimportant features are often ignored since they don't reduce impurity.
7. **Can handle missing data (to an extent)**
   - Some tree algorithms (like CART) can handle missing values via surrogate splits.
8. **Fast to train and interpret**
   - Usually computationally efficient for small to medium datasets.

⚠️ When to Be Careful (Cons of Decision Trees)

🚫 Cons of Decision Trees

1. **High risk of overfitting**
    - Trees can keep splitting until they perfectly fit training data (zero training error).
    - This leads to poor generalization on unseen data.
2. **Unstable**
    - Small changes in data can result in a completely different tree.
3. **Biased toward features with many levels**
    - Features with many unique values can dominate splits (especially in categorical features).
4. **Less powerful alone compared to ensemble methods**
    - A single tree is rarely the best model — Random Forests or Gradient Boosted Trees (like XGBoost) are much stronger.
5. **Non-smooth decision boundaries**
    - Decision surfaces look like blocky steps — not ideal for smooth, continuous relationships.

🤝 Decision Trees vs Logistic Regression

Aspect	**Decision Tree**	**Logistic Regression**
Type	Nonlinear, non-parametric	Linear, parametric
Interpretability	Visual, intuitive	Mathematical, less visual
Handles nonlinearity	✅ Yes	❌ Only if features transformed
Feature scaling	❌ Not needed	✅ Needed (for convergence)
Outliers	Robust	Sensitive
Overfitting	High risk	Low risk (with regularization)
Data requirements	Works well with mixed types	Works best with continuous variables
Speed	Fast	Fast

Aspect	**Decision Tree**	**Logistic Regression**
Generalization	Poor alone	Good baseline
Best use case	Complex, nonlinear patterns	Simple, linear boundaries

🧠 When to Use Each

Situation	Use
You want a simple, interpretable rule-based model	**Decision Tree**
You have nonlinear relationships between features and labels	**Decision Tree**
You have mostly linear relationships and want probabilistic outputs	**Logistic Regression**
You want a strong model with good generalization	**Ensembles of trees (Random Forest, XGBoost)**
You need probabilities calibrated well	**Logistic Regression** (better-calibrated probabilities)

🌳 TL;DR

Model	Strength	Weakness
Decision Tree	Easy to interpret, handles complex patterns	Overfits, unstable
Logistic Regression	Stable, interpretable, mathematically elegant	Only models linear relationships

✅ In short:

> Use Decision Trees when interpretability + nonlinear patterns matter.
>
> Use Logistic Regression when the data is roughly linear and you want stable, probabilistic predictions.