→ Decision Trees

- What is a decision tree
A decision tree is a supervised machine learning algorithm that models decisions using a tree-like structure, where:
- Internal nodes represent feature-based conditions
- Branches represent decision rules
- Leaf node represent final predictions.
It can be used for classification and regression.

- Purpose (When)
To make predictions by recursively splitting data into homogeneous subsets, mimicking human decision-making.
- When relationships are non-linear
- When interpretability is important
- When mixed data types exist (numerical + categorical)

- Why it is used
- Easy to understand and visualize
- Handles non-linear patterns naturally.
- No need for feature scaling
- Works with missing values

- How it works
- Start with entire dataset at the root node
- Select the best feature and split point.
- Split data into child node
- ~~Stop when stopping~~ Repeat recursively for each node
- Stop when stopping criteria are met.
- Assign prediction at leaf nodes.

- Splitting Criteria
  - For Classification:
    - Gini Impurity : $Gini = 1 - \sum p_i^2$
    - Entropy: $Entropy = -\sum p_i \log_2(p_i)$
    - Information Gain
      $IG = Entropy(parent) - \sum \frac{n_k}{n} Entropy(child_k)$
  - For Regressions:
    - MSE
    - MAE
    - Variance Reduction

- Types
  - Based on Task: Classification Tree
    Regression Tree
  - Based on Algorithm: ID3 (Entropy)
    C4.5 (Entropy + Pruning)
    CART (Gini / MSE)

- Formula
  - Prediction: Classification → majority class in leaf
    Regression → mean / median value in leaf
  - Variance Reduction:
    $VR = Var(parent) - \sum \frac{n_k}{n} Var(child_k)$

- Technical Details
  - Assumptions: No strict statistical assumptions
    Works without linearity or normality
  - Stopping Criteria: Max depth reached
    Minimum samples per node
    No further impurity reduction
  - Tree Growth: Top-down, greedy algorithm
    Locally optimal splits (not global)

- Parameters:
  - Model Parameters (Learned): Tree structure
    Split thresholds
    Leaf values
  - Hyperparameters (Tuning): max_depth
    min_sample_split
    min_sample_leaf
    max_features
    criterion (gini, entropy, mse)

- Pros:
  - Highly interpretable
  - Handles non-linear data
  - Works with mixed feature types
  - No feature scaling required
  - Fast inference

- Cons:
  - Prone to overfitting
  - Unstable to small data changes
  - Greedy (locally optimal splits)
  - Poor generalization alone
  - Biased toward dominant features

- Real-World Applications (where):
  - Business: Customer segmentation, churn prediction
  - Finance: Credit risk analysis, Loan approval systems
  - Healthcare: disease diagnosis, Treatment decision support
  - Engineering: Fault detection systems
  - ML Systems: Base learner in Random Forest, XGBoost, LightGBM.

→ Random Forest

- **What is Random Forest**

Random forest is a supervised ensemble machine learning algorithm that builds multiple decision trees and combines their outputs to produce a more accurate and robust prediction.
It works for both classification and regression.

- **Purpose (When)**

To improve prediction accuracy and generalization by reducing the overfitting and instability of individual tree.

- **Why it is used**

- Reduces overfitting of decision trees
- Handles non-linear relationships
- Works well with high-dimensional data
- Robust to noise and outliers
- Provides feature importance

- **How it works**

- Core ideas: Bagging (Bootstrap Aggregation)
  Random Feature selection
- Algorithm:
  - Draw multiple bootstrap samples from dataset
  - Train a decision tree on each sample
  - At every split, select a random subset of features
  - Grow trees independently and in parallel
  - Aggregate predictions:
    Classification → majority vote
    Regression → mean of predictions

- Formula

Classification Prediction : $\hat{y} = \text{model}\{T_1(x), T_2(x), \ldots, T_n(x)\}$

Regression Prediction : $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} T_i(x)$

Where T_i is the prediction of the i-th tree

- Technical Details

- Ensemble Type : Bagging based ensemble
- Tree properties : Trees are deep and unpruned
  - High variance, low bias individually
  - Ensemble reduces variance
- Randomness Source : Data sampling (bootstrap)
  - Feature sampling at splits

- Parameters

- Model Parameters (Learned) : Individual tree structures
  - Split thresholds
  - Leaf predictions
- Hyperparameters (Most important) :
  - n_estimators : no. of trees
  - max_depth : max depth of each tree
  - min_samples_split : min samples to split
  - min_samples_leaf : min samples at leaf
  - max_features : features considered per split
  - bootstrap : Whether sampling is with replacement
  - criterion : Gini, Entropy, MSE

- Feature Importance

Random Forest estimates feature importance using:

- Mean Decrease in Impurity (MDI)
- Permutation Importance

- Assumptions
  - No strict statistical assumptions
  - Assumes trees are weakly correlated

- Pros
  - High accuracy
  - Handles non-linear data well
  - Resistant to overfitting
  - Works with missing values
  - Scales well with large datasets

- Cons
  - Less interpretable than single trees
  - Computationally expensive
  - Slower inference
  - Large memory usage
  - Feature importance can be biased.

- Real World Application (Where)
  - Finance: Credit scoring
    Fraud detection
  - Healthcare: Disease prediction
    Risk stratification
  - Marketing: Customer churn prediction
    Recommendation systems
  - Engineering: Fault detection
    Predictive maintenance
  - ML Pipelines: Baseline model for tabular data
    Feature selection