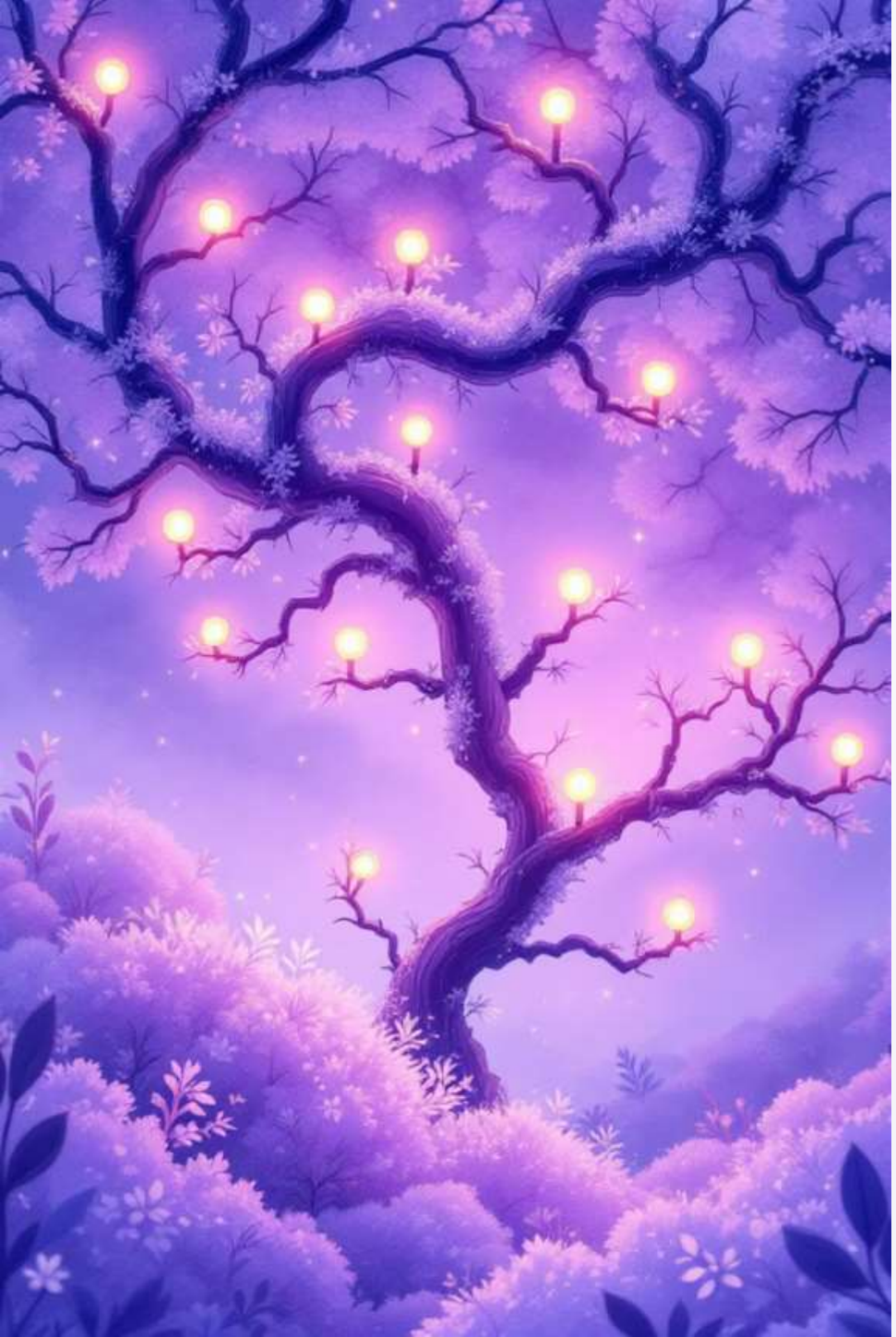


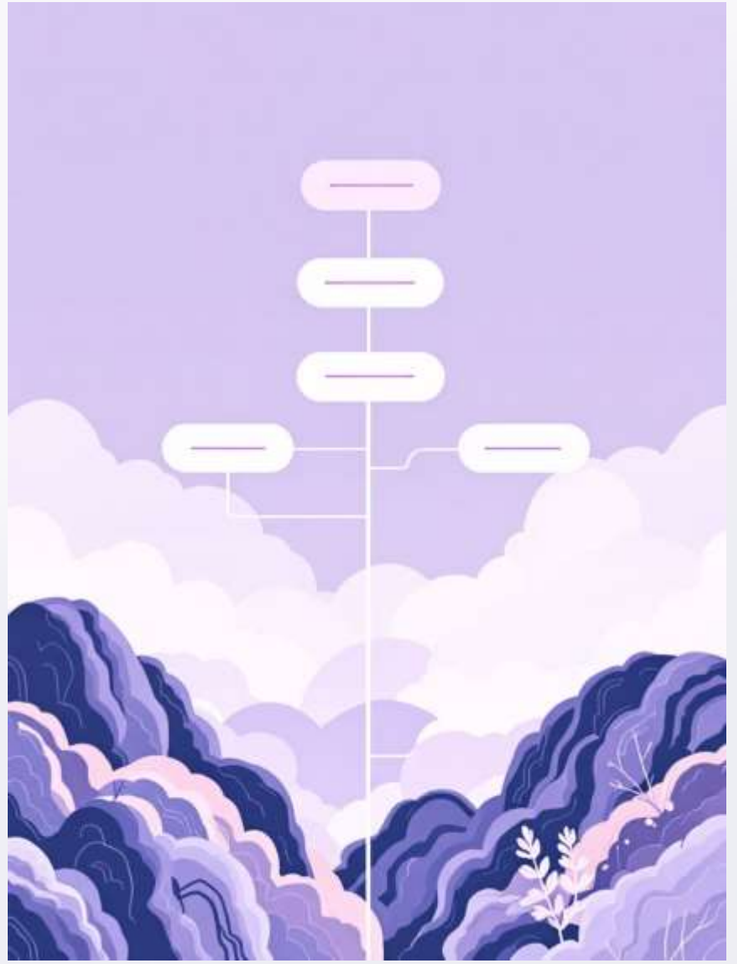
Decision Tree Algorithm

A Complete Conceptual & Practical Guide

What is a Decision Tree?

A decision tree is a **supervised machine learning algorithm** that can be used for both *classification* and *regression* tasks. It uses a tree-like structure to represent decisions and their outcomes.

Imagine playing a game of 20 questions—the tree asks a series of yes/no questions about features in your data, progressively narrowing down to a prediction.



Each internal node represents a **test on an attribute**, each branch represents the outcome of the test, and each leaf node represents a class label (classification) or value (regression).

Why Use Decision Trees?

Interpretability

Easy to understand and visualize—no complex mathematics required for interpretation

No Scaling Needed

Unlike neural networks, decision trees don't require feature standardization or normalization

Versatile Data Types

Handles both numerical and categorical data seamlessly

Feature Importance

Naturally identifies which features are most important for prediction

Businesses prefer decision trees because they can explain predictions to stakeholders in plain language—a critical advantage in regulated industries.

How Decision Trees Work

Decision trees use a **greedy top-down approach** with recursive partitioning. The algorithm starts at the root and splits the data into subsets that are increasingly "pure."

Root Selection

Begin at the top with all training instances

Feature Selection

Choose the best feature to split using criteria like Gini or Information Gain

Data Partitioning

Split dataset into subsets based on feature values

Recursive Process

Repeat splitting for each subset

Termination

Stop when criteria are met—pure node, maximum depth, or minimum samples

Mathematical Foundation

Entropy & Information Gain

Entropy measures uncertainty in a dataset. The formula is:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where *p_i* is the probability of class *i*.

Information Gain measures how much uncertainty is reduced by splitting on a feature:

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

Higher information gain = better split

Gini Index

Gini measures impurity:

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

Lower Gini = purer subset

Types of Decision Tree Algorithms

Algorithm	Key Features	Use Case
ID3	Uses Information Gain, only categorical data, no pruning	Simple datasets with discrete attributes
C4.5	Extension of ID3 using Gain Ratio, handles continuous attributes	More complex datasets with mixed data types
CART	Uses Gini Index, handles regression and classification	Most widely used (Scikit-learn implementation)
CHAID	Uses Chi-square tests, best for categorical targets	Market research and survey analysis

CART is the most commonly used algorithm, especially in Scikit-learn

Hyperparameters (Part 1)

criterion

What: Splitting metric ('gini', 'entropy', 'mse')

Effect: Different criteria may produce different trees. Gini is faster; Entropy may produce more balanced trees.

max_depth

What: Maximum depth of tree

Effect: Critical for preventing overfitting. Shallow trees underfit; deep trees overfit.

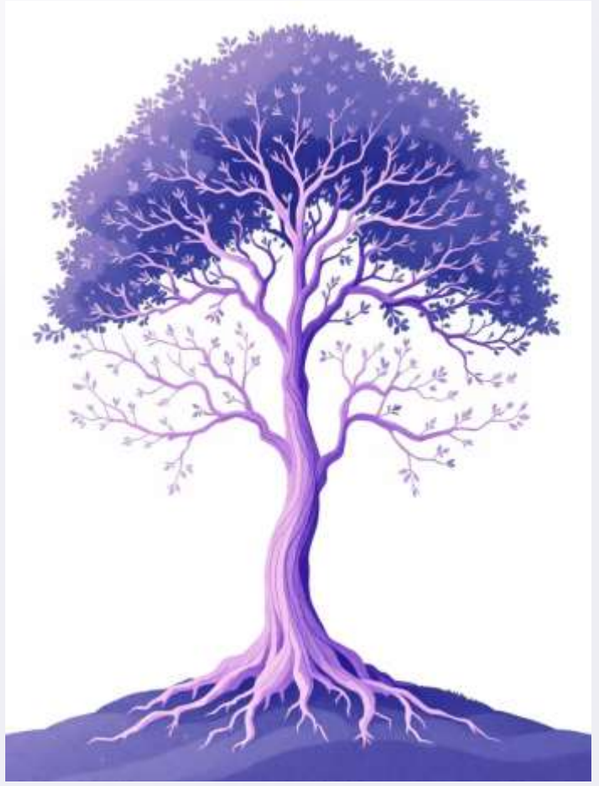
min_samples_split

What: Minimum samples needed to split

Effect: Higher values prevent splits on small subsets, reducing overfitting.

Visual: Shallow Tree

Visual: Deep Tree



Hyperparameters (Part 2)

1

min_samples_leaf

Minimum samples required in leaf nodes

2

max_features

Number of features considered for split

3

max_leaf_nodes

Maximum number of leaf nodes

4

min_impurity_decrease

Minimum impurity reduction for split

5

ccp_alpha

Cost complexity pruning parameter

6

class_weight

Weights for imbalanced classes

Key insight: Increasing minimum sample requirements and limiting tree depth prevents overfitting by reducing model complexity.

Python Implementation

Classification with Scikit-learn

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

# Load data
data = load_iris()
X, y = data.data, data.target

# Create and tune model
dt = DecisionTreeClassifier()
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10]
}
grid = GridSearchCV(dt, params, cv=5)
grid.fit(X, y)
```

Visualize & Extract Insights

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plot_tree(grid.best_estimator_,
 feature_names=data.feature_names,
 class_names=data.target_names,
 filled=True)
plt.show()

# Feature importance
print("Feature importance:",
 grid.best_estimator_.feature_importances_)
```

Practical Considerations

When to Use

- Interpretability is critical
- Small to medium datasets
- Extracting business rules
- Non-linear relationships
- Quick prototyping needed

Watch For

- **Overfitting:** Always use cross-validation and pruning
- **Variance:** Consider Random Forest for stability
- **Imbalanced data:** Use class_weight parameter
- **Instability:** Small data changes create different trees

Key takeaway: Decision trees are excellent for explainable AI and business applications, but ensemble methods like Random Forest often perform better in practice.