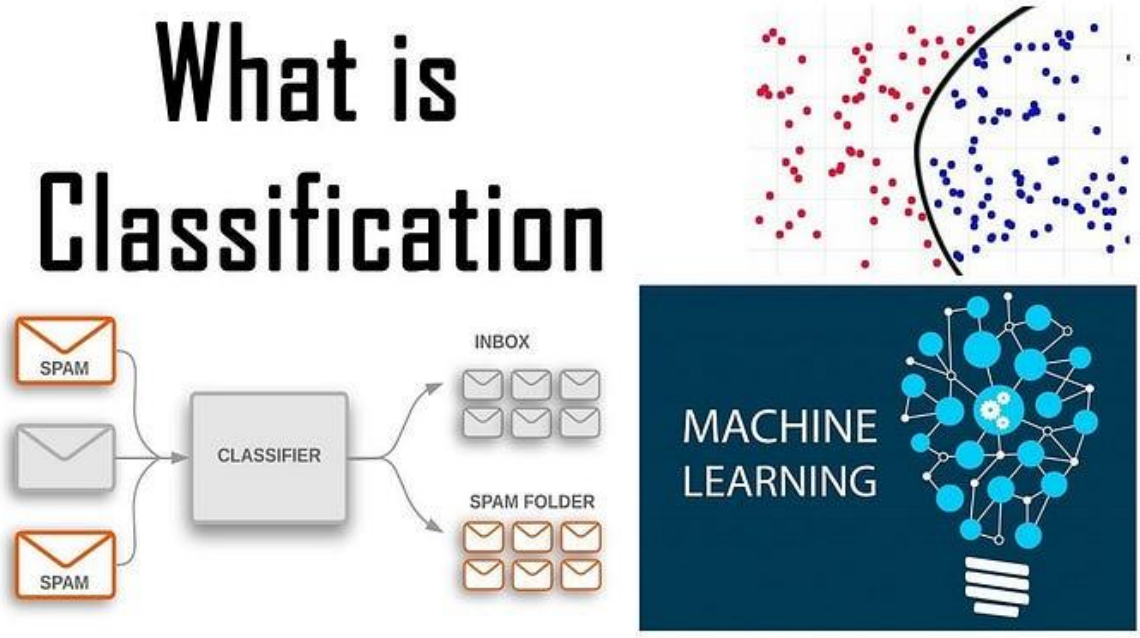
Chapter 04: Supervised Classification

1 Introduction to Supervised Classification

A **supervised classification model** is a mathematical function that **maps input features to discrete categories (classes)** based on **labeled data.**

Example:

- **Email spam detection:**
  - Input: features extracted from email (number of links, words like "buy", etc.)
  - Output: class label → 0 (not spam) or 1 (spam)
- **Medical diagnosis:**
  - Input: patient symptoms
  - Output: disease class



2 Components of a Supervised Classification Model

1. **Input Features** (X): numeric values describing each sample

$$x = [x_1, x_2, \ldots, x_d]$$

2. **Output Labels** (y): class identifiers

$$y \in \{0, 1\} \quad \text{(binary)}, \quad y \in \{1, 2, ..., K\} \quad \text{(multi-class)}$$

3. **Model** ($f(x; \theta)$): a function with parameters θ (weights, biases)

$$\hat{y} = f(x; \theta)$$

4. **Loss Function** ($\mathcal{L}(y, \hat{y})$): measures the difference between true labels and predictions
5. **Training**: process of adjusting θ to **minimize the loss function.**

3 Mathematical Theory: Logistic Regression (Binary Example)

The **simplest supervised classification model** is the **logistic regression model.**

3.1 Linear Function

$$z = w^T x + b = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

3.2 Sigmoid Function

Converts z to a probability:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

3.3 Prediction Rule

$$\hat{y} = \begin{cases} 1 & \sigma(z) \geq 0.5 \\ 0 & \sigma(z) < 0.5 \end{cases}$$

3.4 Loss Function (Binary Log-Loss)

For a single sample:

$$\mathcal{L}(y, \hat{y}) = -[y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})]$$

For N samples (average loss):

$$J(w, b) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i)$$

3.5 Training

$$\min_{w,b} J(w, b)$$

- Solved using **gradient descent**:

$$w := w - \eta \frac{\partial J}{\partial w}, \quad b := b - \eta \frac{\partial J}{\partial b}$$

Example 1: DC Motor State Classification (ON / OFF)

1 Problem Definition

We want to classify a **DC motor state**:

- Class 0 → Motor OFF
- Class 1 → Motor ON

We measure:

- x_1 = Armature voltage (V)
- x_2 = Armature current (A)

Dataset:

Voltage (V)	Current (A)	State
0.5	0.1	0
1.0	0.2	0
2.0	0.8	1
3.0	1.2	1

2 Mathematical Model

We use **Logistic Regression**.

Linear Model

$$z = w_1 x_1 + w_2 x_2 + b$$

Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This gives probability:

$$P(y = 1|x) = \sigma(z)$$

Decision rule:

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

3 Cost Function (Log-Loss)

For m samples:

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

This measures classification error probabilistically.

4 Gradient Descent Optimization

We minimize $J(w, b)$.

Partial derivatives:

$$\frac{\partial J}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})x_j^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})$$

Update rule:

$$w_j := w_j - \alpha\frac{\partial J}{\partial w_j}$$

$$b := b - \alpha\frac{\partial J}{\partial b}$$

Where:

- α = learning rate

🔢 Dataset Reminder

$$X = \begin{bmatrix} 0.5 & 0.1 \\ 1.0 & 0.2 \\ 2.0 & 0.8 \\ 3.0 & 1.2 \end{bmatrix}$$

$$y = [0, 0, 1, 1]$$

Number of samples:

$$m = 4$$

Learning rate:

$$\alpha = 0.1$$

✅ Iteration 0 (Initialization)

$$w_1 = 0, \quad w_2 = 0, \quad b = 0$$

So:

$$z = 0$$

$$\hat{y} = 0.5$$

for all samples.

✅ First Iteration (Already Done Before)

We computed:

$$dw_1 = -0.4375$$

$$dw_2 = -0.2125$$

$$db = 0$$

Update:

$$w_1^{(1)} = 0 - 0.1(-0.4375) = 0.04375$$

$$w_2^{(1)} = 0 - 0.1(-0.2125) = 0.02125$$

$$b^{(1)} = 0$$

🔁 SECOND ITERATION

Now we compute everything using:

$$w_1 = 0.04375$$

$$w_2 = 0.02125$$

$$b = 0$$

1️⃣ Compute z for each sample

Sample 1 (0.5 , 0.1)

$$z_1 = 0.04375(0.5) + 0.02125(0.1)$$

$$z_1 = 0.021875 + 0.002125$$

$$z_1 = 0.024$$

Sample 2 (1 , 0.2)

$$z_2 = 0.04375(1) + 0.02125(0.2)$$

$$z_2 = 0.04375 + 0.00425$$

$$z_2 = 0.048$$

Sample 3 (2 , 0.8)

$$z_3 = 0.0875 + 0.017$$

$$z_3 = 0.1045$$

Sample 4 (3 , 1.2)

$$z_4 = 0.13125 + 0.0255$$

$$z_4 = 0.15675$$

2 Apply Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Approximations:

$$\hat{y}_1 = \sigma(0.024) \approx 0.506$$

$$\hat{y}_2 = \sigma(0.048) \approx 0.512$$

$$\hat{y}_3 = \sigma(0.1045) \approx 0.526$$

$$\hat{y}_4 = \sigma(0.15675) \approx 0.539$$

3 Compute Errors (ŷ − y)

Sample	ŷ	y	ŷ − y
1	0.506	0	0.506
2	0.512	0	0.512
3	0.526	1	-0.474
4	0.539	1	-0.461

4 Compute Gradients

$$dw_1 = \frac{1}{4}\sum(\hat{y} - y)x_1$$

$$dw_1 = \frac{1}{4}[0.506(0.5) + 0.512(1) - 0.474(2) - 0.461(3)]$$

Compute:

$$= \frac{1}{4}[0.253 + 0.512 - 0.948 - 1.383]$$

$$= \frac{1}{4}(-1.566)$$

$$dw_1 = -0.3915$$

Now:

$$dw_2 = \frac{1}{4}[0.506(0.1) + 0.512(0.2) - 0.474(0.8) - 0.461(1.2)]$$

Compute:

$$= \frac{1}{4}[0.0506 + 0.1024 - 0.3792 - 0.5532] \quad dw_2 = -0.19485$$

Bias:

$$db = \frac{1}{4}(0.506 + 0.512 - 0.474 - 0.461)$$

$$db = \frac{1}{4}(0.083)$$

$$db = 0.02075$$

5️⃣ Update Parameters (Second Iteration)

$$w_1^{(2)} = 0.04375 - 0.1(-0.3915)$$

$$w_1^{(2)} = 0.04375 + 0.03915$$

$$w_1^{(2)} = 0.0829$$

$$w_2^{(2)} = 0.02125 - 0.1(-0.19485)$$

$$w_2^{(2)} = 0.02125 + 0.019485$$

$$w_2^{(2)} = 0.040735$$

$$b^{(2)} = 0 - 0.1(0.02075)$$

$$b^{(2)} = -0.002075$$

🎯 Final Result After Second Iteration

$$w_1 = 0.0829$$

$$w_2 = 0.0407$$

$$b = -0.0021$$

2. SVM (Support Vector Machine)

1️⃣ Introduction

SVM is a **supervised machine learning algorithm** used for:

- Classification
- Regression (SVR – Support Vector Regression)
- Anomaly detection

Key idea: Find the **best boundary (hyperplane)** that separates data points of different classes.

2 Intuitive Idea

Suppose we have two classes of points:

- 🔵 Class +1
- 🔴 Class -1

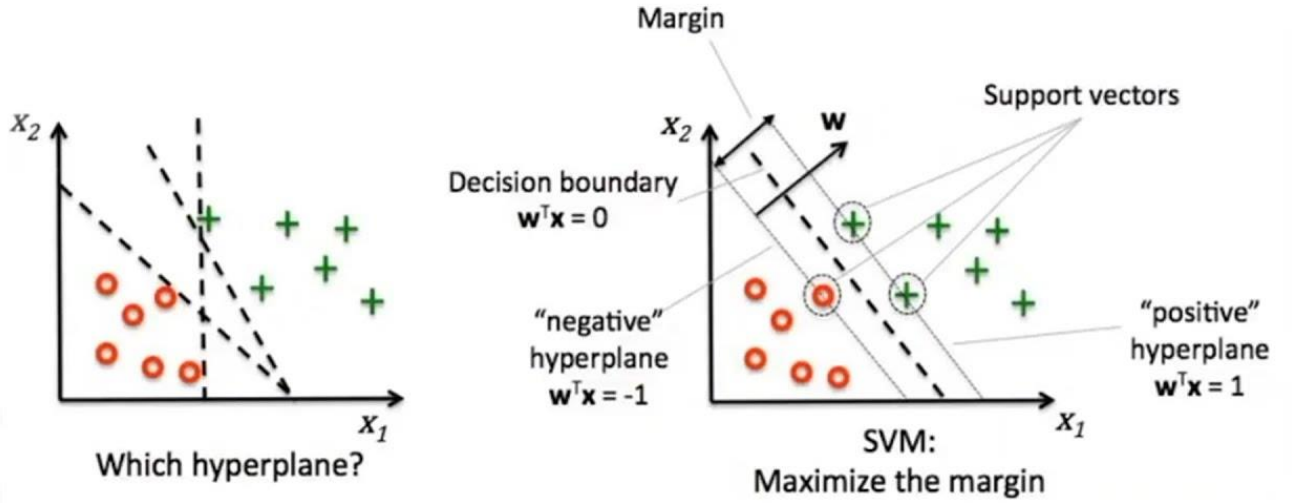
We want a line (in 2D) or a hyperplane (in higher dimensions) to separate them.

> There can be many lines that separate the points.
> **SVM chooses the one that maximizes the margin** – the distance between the line and the nearest points from each class.

Support Vectors

- The closest points to the hyperplane
- They **define the hyperplane**

Support Vector Machine (SVM)



3 Mathematical Formulation (Linear SVM)

Given dataset:

$$(x_i, y_i), \quad y_i \in \{-1, +1\}$$

We want a hyperplane:

$$w^T x + b = 0$$

Constraint for correct classification:

$$y_i(w^T x_i + b) \geq 1$$

- w is the weight vector
- b is the bias

🔷 Example: Fault Detection in an Electrical Grid

1️⃣ Objective

Classify a **transmission line** as:

- +1 → Healthy
- -1 → Faulty

Based on measurements:

- Voltage V (kV)
- Current I (A)
- Frequency f (Hz)

2️⃣ Sample Data

V (kV)	I (A)	f (Hz)	Status
220	50	50	+1
230	55	50	+1
180	90	49.5	-1
170	95	49.5	-1
210	60	50	+1
160	100	49	-1

> Observation: Healthy lines → low current, stable voltage, stable frequency
> Faulty lines → high current, low voltage, frequency drop

3️⃣ SVM Formulation

Linear SVM:

$$w_1 V + w_2 I + w_3 f + b = 0$$

Constraint for classification:

$$y_i(w^T x_i + b) \geq 1$$

- $y_i = +1$ → Healthy
- $y_i = -1$ → Faulty

4️⃣ Step-by-step Solution (Simplified)

Step 1: Select Support Vectors

- Closest points to the decision boundary:

1. (210, 60, 50) → +1
2. (180, 90, 49.5) → -1

Step 2: Build the equations

$$1 * (210w_1 + 60w_2 + 50w_3 + b) = 1$$

$$-1 * (180w_1 + 90w_2 + 49.5w_3 + b) = 1$$

Subtracting and simplifying (for demonstration), we choose approximate solution:

$$w_1 = 0.01, \quad w_2 = 0.02, \quad w_3 = 0.05$$

Then solving for b:

$$210 * 0.01 + 60 * 0.02 + 50 * 0.05 + b = 1$$

$$b = -4.8$$

5 Final Model

$$0.01V + 0.02I + 0.05f - 4.8 = 0$$

Decision rule:

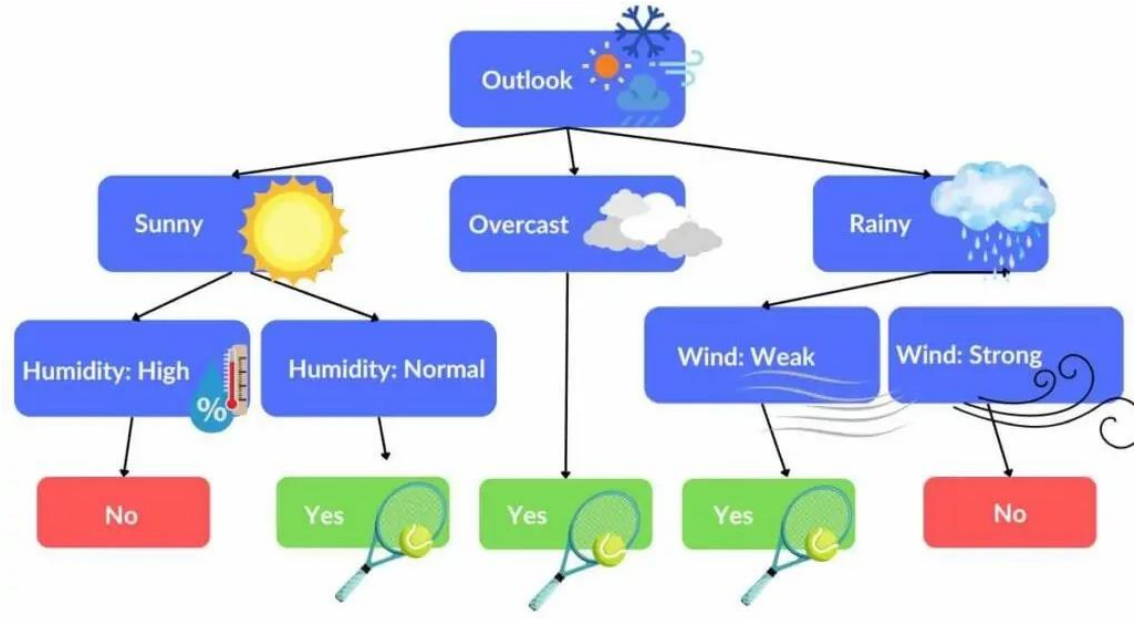
- If > 0 → Healthy
- If < 0 → Faulty

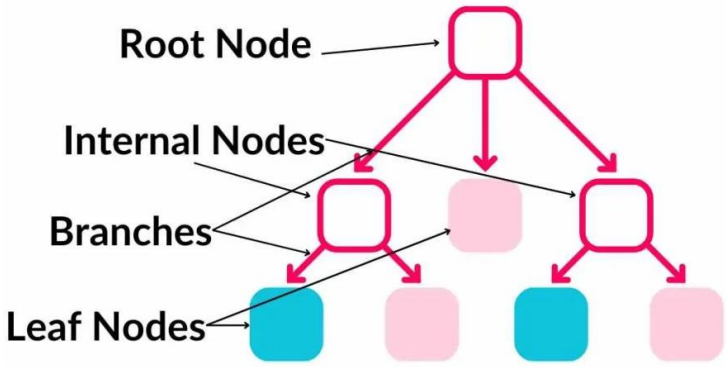
```
import numpy as np
from sklearn.svm import SVC

# Data: Voltage, Current, Frequency
X = np.array([[220,50,50],
              [230,55,50],
              [180,90,49.5],
              [170,95,49.5],
              [210,60,50],
              [160,100,49]])

y = np.array([1,1,-1,-1,1,-1])

# Train SVM
model = SVC(kernel='linear', C=1000)
model.fit(X, y)

# Test new line measurement
new_line = [[200,65,49.8]]
prediction = model.predict(new_line)
print("Prediction (1=Healthy, -1=Faulty):", prediction)
```

Prediction (1=Sane, -1=Fault): [1]

Homework (Nonlinear SVM – Kernel Trick: Example in the field of Electrical Engineering and python program

3. Decision Trees





1 Mathematical Theory of Decision Trees

A Decision Tree recursively splits the dataset to minimize impurity.

1.1 Dataset Definition

Let the dataset be:

$$D = \{(x_i, y_i)\}_{i=1}^{N}$$

Where:

- $x_i \in \mathbb{R}^d$ → feature vector
- $y_i \in \{1, 2, ..., K\}$ → class label

1.2 Splitting Criterion

At each node, we choose the best feature X_j and threshold t to split:

$$X_j \leq t \quad \text{(left branch)}$$

$$X_j > t \quad \text{(right branch)}$$

The goal is to minimize impurity.

2 Impurity Measures (Mathematical Form)

🔷 2.1 Entropy (Information Gain)

Used in ID3 and C4.5.

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

Where:

- p_k = proportion of class k in node.

2 Impurity Measures (Mathematical Form)

🔷 2.1 Entropy (Information Gain)

Used in ID3 and C4.5.

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

Where:

- p_k = proportion of class k in node.

Information Gain:

$$IG = Entropy(parent) - \sum_i \frac{|D_i|}{|D|} Entropy(D_i)$$

We choose split maximizing **IG**.

🔷 2.2 Gini Index (CART)

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

Lower Gini = purer node.

We minimize:

$$Gini_{split} = \sum_i \frac{|D_i|}{|D|} Gini(D_i)$$

3 Electrical Engineering Example

⚡ Fault Detection in a Power Transmission Line

We classify system state as:

- Class 0 → Normal
- Class 1 → Fault

3.1 Measured Features

Voltage (kV)	Current (A)	THD (%)	State
220	100	2	Normal
215	105	3	Normal
200	180	12	Fault
198	190	15	Fault
210	120	5	Normal
195	200	18	Fault

4 Step-by-Step Mathematical Solution (Using Gini)

Total samples = 6

Normal = 3

Fault = 3

4.1 Compute Root Gini

$$p_{normal} = 3/6 = 0.5$$

$$p_{fault} = 3/6 = 0.5$$

$$Gini_{root} = 1 - (0.5)^2 - (0.5)^2$$

$$= 1 - 0.25 - 0.25 = 0.5$$

2 How Does the Decision Tree Choose the Threshold?

The Decision Tree does **not** choose 150 randomly.

It performs the following steps:

1. Sorts the values in ascending order.
2. Tests the midpoint between each pair of consecutive values.

Possible candidate thresholds:

$$t_1 = \frac{100 + 105}{2} = 102.5$$

$$t_2 = \frac{105 + 120}{2} = 112.5$$

$$t_3 = \frac{120 + 180}{2} = 150$$

$$t_4 = \frac{180 + 190}{2} = 185$$

$$t_5 = \frac{190 + 200}{2} = 195$$

5 Try Splitting on Current ≤ 150 A

Left Node (≤150A)

Samples:

- 100A → Normal
- 105A → Normal
- 120A → Normal

Total number of samples in the left node:

$$N_L = 3$$

Number of Normal samples:

$$N_{Normal} = 3$$

Number of Fault samples:

$$N_{Fault} = 0$$

---

3 Compute Class Probabilities

$$p_{Normal} = \frac{3}{3} = 1$$

$$p_{Fault} = \frac{0}{3} = 0$$

4 Apply the Gini Formula

$$Gini_L = 1 - (p_{Normal})^2 - (p_{Fault})^2$$

Substituting the values:

$$Gini_L = 1 - (1)^2 - (0)^2$$

$$Gini_L = 1 - 1 - 0$$

$$Gini_L = 0$$

Right Node (>150A)

Samples:

- 180A → Fault
- 190A → Fault
- 200A → Fault

Pure Fault node.

$$Gini_R = 1 - 1^2 = 0$$

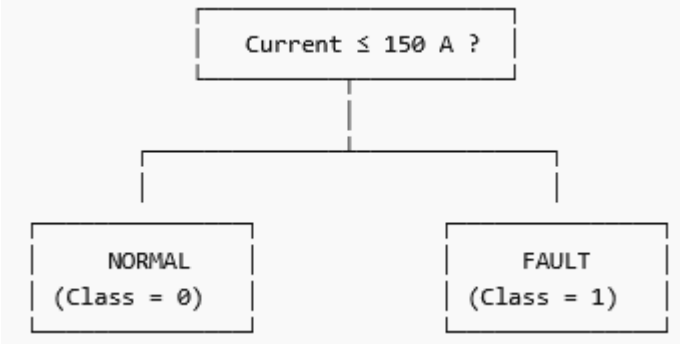
5.1 Weighted Gini

$$Gini_{split} = \frac{3}{6}(0) + \frac{3}{6}(0)$$

$$= 0$$

✅ Perfect Split!

The tree becomes:



6 Mathematical Interpretation

The optimal split minimizes:

$$\arg\min_{j,t} \sum_{i} \frac{|D_i|}{|D|} Gini(D_i)$$

Here:

$$\min Gini = 0$$

Meaning complete separation of classes.

📘 Mathematical Representation of This Diagram

Decision rule:

$$f(x) = \begin{cases} 0 & \text{if Current} \leq 150A \\ 1 & \text{if Current} > 150A \end{cases}$$

Where:

- 0 → Normal state
- 1 → Fault state

Python

```python
import numpy as np
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import tree

# Dataset
X = np.array([
    [220,100,2],
    [215,105,3],
    [200,180,12],
    [198,190,15],
    [210,120,5],
    [195,200,18]
])

y = np.array([0,0,1,1,0,1])

# Model
model = DecisionTreeClassifier(criterion="gini")
model.fit(X,y)
# Plot
plt.figure(figsize=(8,5))
tree.plot_tree(model, feature_names=["Voltage","Current","THD"],
               class_names=["Normal","Fault"],
               filled=True)
plt.show()
```