Cây quyết định (Decision Tree) thuộc nhóm học có giám sát (supervised learning), được sử dụng cho cả bài toán phân loại (classification) và hồi quy (regression).

Ý tưởng chính: Tìm cách phân chia dữ liệu theo từng đặc trưng sao cho kết quả trong mỗi nhánh là đồng nhất có thể.

Cây quyết định là mô hình dạng cây, trong đó:

- **Nút gốc (Root Node)**: điểm bắt đầu, chứa điều kiện đầu tiên để phân chia dữ liệu.
- **Nút quyết định (Decision Node)**: nơi kiểm tra điều kiện và chia dữ liệu.
- **Nhánh (Branch)**: kết quả của điều kiện kiểm tra.
- **Nút lá (Leaf Node)**: kết quả cuối cùng (nhãn hoặc giá trị dự đoán).

Nhãn (label) là giá trị đầu ra mà mô hình cần dự đoán.

- Trong **phân loại**: nhãn là lớp (ví dụ: "spam" hoặc "not spam").
- Trong **hồi quy**: nhãn là giá trị số (ví dụ: giá nhà, nhiệt độ).
- Mỗi mẫu dữ liệu trong tập huấn luyện đều có một nhãn tương ứng, và cây quyết định học cách phân chia dữ liệu để dự đoán nhãn đó.

Bước 1: Tính độ hỗn loạn của nút cha

Phân loại → dùng: **Gini** hoặc **Entropy**

Hồi quy → dùng: **Phương sai (Variance)** hoặc **MSE**

Bước 2: Thử các cách chia

Với mỗi thuộc tính (feature) và ngưỡng chia:
→ Chia dữ liệu thành 2 nhóm (nút con)
→ Tính độ hỗn loạn của mỗi nút con
→ Tính **trung bình có trọng số** theo số mẫu

❖

Bước 3: Tính độ cải thiện

$$\text{Gain} = \text{Độ hỗn loạn của cha} - \text{Trung bình trọng số của con}$$

→ Chọn cách chia có Gain lớn nhất

Bước 4: Lặp lại đệ quy cho các nút con

→ Dừng khi đạt điều kiện dừng (sâu quá, ít mẫu, lớp thuần...)

Điều kiện dừng:

- Tất cả mẫu trong một nút thuộc cùng một lớp.

- Không còn thuộc tính nào để phân chia.
- Đạt đến độ sâu tối đa của cây hoặc số lượng mẫu tối thiểu tại nút.

Dự đoán:

- Khi dự đoán cho một mẫu mới, mẫu sẽ đi qua cây từ nút gốc, qua các nút bên trong theo các điều kiện phân chia, cho đến khi đến nút lá để nhận kết quả.

tại mỗi bước, thuật toán chỉ chọn giải pháp tốt nhất cục bộ (tức là phân chia tốt nhất tại nút hiện tại) mà không xem xét tác động tổng thể hoặc tối ưu toàn cục của cây.

Các thuật toán xây dựng Cây Quyết định

Tổng quan về tiêu chí phân chia : Mỗi nút trong cây quyết định cần chọn "thuộc tính" (hoặc tập giá trị) sao cho dữ liệu ở mỗi nhánh sau khi chia là thuần nhất nhất. Chỉ số phân tách (impurity measure) giúp đánh giá độ "lộn xộn" của nhãn trong một nút, từ đó chọn điểm chia tốt nhất.

Cây phân loại Hai tiêu chí phổ biến nhất là :

Gini Index

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

Entropy

$$I_H = -\sum_{j=1}^{c} p_j log_2(p_j)$$

:

• c: số lớp. • p_j: xác suất thuộc lớp j. • Gini Index đo mức **tạp nhiễu** (impurity) trong một tập dữ liệu. • Gini càng **thấp** ⇒ dữ liệu càng **thuần nhất** (ít pha trộn các lớp). • Gini bằng **0** nếu tất cả phần tử thuộc **cùng một lớp**. • Gini đạt giá trị **tối đa** khi các lớp phân bố **đều nhau**. Ví dụ với 2 lớp, $p_1 = 0.7$, $p_2 = 0.3$: $\sum p_j^2 = 0.7^2 + 0.3^2 = 0.49 + 0.09 = 0.58$ $I_G = 1 - 0.58 = 0.42$	Entropy đo **mức độ không chắc chắn** hoặc hỗn loạn của dữ liệu • $log_2(p\_j)$: log cơ số 2, đo "lượng thông tin" của lớp j. • Giá trị: • *IH = 0* : tập hoàn toàn thuần nhất (chỉ có 1 lớp). • *IH* lớn nhất khi các lớp có tỷ lệ bằng nhau. • Ví dụ 1: Nếu $p_1 = 0.5, p_2 = 0.5$ $I_H = -[0.5 \log_2(0.5) + 0.5 \log_2(0.5)] = -[0.5 * (-1) + 0.5 * (-1)] = 1$ bit Ví dụ 2: Nếu $p_1 = 1, p_2 = 0 \Rightarrow I_H = 0$

Cây hồi quy không dùng **Gini hay Entropy** như cây phân loại, mà dùng **phương sai**. Tại mỗi node, ta chia dữ liệu thành 2 phần (trái và phải) sao cho **giảm được nhiều phương sai nhất** trong dữ liệu đầu ra (y).

Giả sử tại một node ta có:

- Tập dữ liệu S gồm n mẫu: $S = \{(x_i, y_i)\}_{i=1}^{n}$
- Tập này được chia thành:
  - Nhánh trái S_L (có n_L mẫu)
  - Nhánh phải S_R (có n_R mẫu)

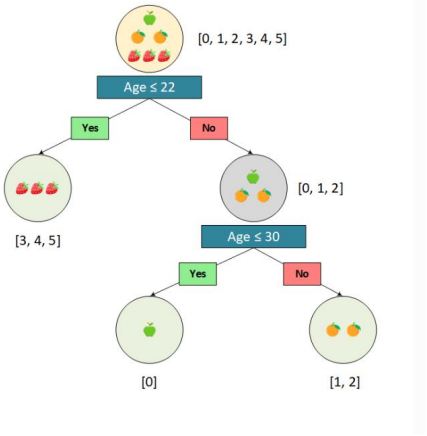
với $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

Lượng giảm phương sai (Gain): Cách chia nào có **Gain lớn nhất → chọn chia đó.**

$$\text{Gain} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}_{\text{Trước chia}} - \left[\underbrace{\frac{n_L}{n} \cdot \frac{1}{n_L}\sum_{i \in S_L}(y_i - \bar{y}_L)^2}_{\text{Nhánh trái}} + \underbrace{\frac{n_R}{n} \cdot \frac{1}{n_R}\sum_{i \in S_R}(y_i - \bar{y}_R)^2}_{\text{Nhánh phải}}\right]$$

ví dụ :

id	Gender	Age	Interest (Label)
0	Male	30	🍏
1	Female	32	🍊
2	Female	35	🍊
3	Male	22	🍓
4	Male	18	🍓
5	Female	15	🍓



Ví dụ dữ liệu đầu vào gồm ba cột **Gender** (giới tính), **Age** (tuổi) và **Interest** (sở thích, tức nhãn cần phân loại). Cây quyết định được xây dựng từ dữ liệu này như trong hình trên, với chỉ số Gini để đánh giá chất lượng phân chia.

Tại nút gốc (root), thuật toán sẽ thử chia dữ liệu theo các thuộc tính khác nhau (giới tính hoặc tuổi) và lựa chọn điều kiện phân chia làm sao để *chỉ số Gini được giảm nhiều nhất.*

Tại sao chọn Age thay vì Gender? Ta tính Gini của phép chia theo từng thuộc tính. Phân chia theo **Gender** tạo hai nhóm: nhóm *Male* (3 mẫu) có nhãn {Apple, Strawberry, Strawberry} với Gini ≈ 0.444, nhóm *Female* (3 mẫu) có nhãn {Orange, Orange, Strawberry} với Gini ≈ 0.444. Trọng số gộp GiniGender =63 ×0.444+63 ×0.444=0.444.

Nếu phân chia theo **Age**, ta xem xét các ngưỡng tiềm năng (ví dụ giữa 22 và 30 là ngưỡng 26) rồi tính Gini. GiniAge≤22 = 0.222 Đây là giá trị Gini tổng thấp hơn Do vậy, thuật toán chọn phân chia theo **Age ≤ 22** tại nút gốc, vì **độ tinh khiết Gini được giảm mạnh nhất**

Chọn ngưỡng 22 và 30 vì : Thuật toán thử các ngưỡng giữa 15-18 (≈16.5), 18-22 (20), 22-30 (26), 30-32 (31), 32-35 (33.5) và tính Gini tổng. Kết quả cho thấy ngưỡng giữa 22 và 30 (26) cho Gini tổng ≈0.222, tốt nhất. Ta có thể chọn ngưỡng là 22 (vì sau 22 là 30) để được cùng kết quả phân chia: nhóm ≤22 có ba mẫu (Strawberry) và nhóm >22 có ba mẫu còn lại.

Nhánh trái (Age ≤ 22) thu về ba mẫu đều thuộc cùng lớp (cả ba đều "Strawberry"), nên Gini của nhánh trái này bằng 0; **nhánh phải** còn lại gồm ba mẫu với phân bố hỗn hợp (1 apple, 2 orange) và tiếp tục được chia .

Tiếp tục với nhánh phải ban đầu (các mẫu có *Age > 22* tức {30, 32, 35} tương ứng các lá 0,1,2), thuật toán lại xét các ngưỡng giữa 30-32 (31) và 32-35 (33.5). Thử ngưỡng *Age ≤ 30* (tức nhóm trái chỉ chứa mẫu 30; nhóm phải chứa

32,35) cho kết quả **nhánh trái một lớp (Apple) và nhánh phải một lớp (Orange)**, nên cả hai nhánh con đều Gini = 0, Gini tổng = 0. Đây là phân chia tốt nhất. (Nếu thử 31 hoặc 33.5, ta vẫn được Gini tổng = 0.25, kém hơn). Vì vậy **Age ≤ 30** được chọn tại nút này. Nhánh trái (Age ≤ 30) chứa mẫu Apple, nhánh phải chứa hai mẫu Orange, cả hai đều đã thuần nhất về nhãn. Mô hình cây kết thúc ở đây với các nút lá tương ứng label Strawberry, Apple, Orange.

Phương pháp ngăn chặn overfitting

- **Overfitting**: Hiện tượng cây quyết định quá phức tạp, học cả nhiễu trong dữ liệu huấn luyện, dẫn đến hiệu suất kém trên dữ liệu mới.
- **Các phương pháp ngăn chặn**:
  - **Pruning (Cắt tỉa)**: Loại bỏ các nhánh không cải thiện hiệu suất hoặc làm tăng lỗi trên tập kiểm tra.
  - **Giới hạn độ sâu tối đa**: Không cho cây phát triển quá sâu.
  - **Số lượng mẫu tối thiểu tại nút lá**: Đảm bảo mỗi nút lá có đủ số lượng mẫu để tránh phân chia quá chi tiết.

5. Ưu điểm và nhược điểm

Ưu điểm:

- Dễ hiểu và diễn giải, phù hợp để giải thích các quyết định.
- Không cần chuẩn hóa dữ liệu (như chuẩn hóa khoảng giá trị).
- Xử lý được cả dữ liệu số và dữ liệu danh mục.
- Hỗ trợ khám phá dữ liệu, giúp xác định các thuộc tính quan trọng.

Nhược điểm: *Việc liên tục phân chia để đạt được độ thuần khiết cao nhất trên dữ liệu huấn luyện có thể khiến cây trở nên quá cụ thể, học cả nhiễu và các mẫu không đại diện cho dữ liệu tổng thể. Đây là một yếu tố trực tiếp gây ra vấn đề quá khớp (overfitting)*

- Dễ bị overfitting nếu không được kiểm soát.
- Không ổn định: Thay đổi nhỏ trong dữ liệu có thể tạo ra cây hoàn toàn khác.
- Hiệu suất kém với các bài toán có mối quan hệ phức tạp giữa các thuộc tính.

```python
import pandas as pd
from collections import Counter

# Đọc dữ liệu Iris
iris = pd.read_csv('iris.csv')
X = iris.drop(columns=['species']).values.tolist()
y = iris['species'].tolist()
```

- **pandas**: Đọc và xử lý dữ liệu từ file CSV
- **Counter**: Đếm tần suất xuất hiện của các nhãn (dùng cho tính toán Gini)
- X: Danh sách các mẫu dữ liệu (features)
- y: Danh sách nhãn tương ứng

```python
class DecisionTreeClassifierFromScratch:
    def __init__(self, max_depth=None, min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.tree = None
```

- max_depth: Giới hạn chiều sâu tối đa của cây. Nếu None, cây sẽ phát triển cho đến khi tất cả các lá đều thuần nhất.
- min_samples_split: Số lượng mẫu tối thiểu cần có tại một nút để có thể chia tiếp. Nếu ít hơn, nút sẽ trở thành lá.
- tree: Biến lưu trữ cấu trúc cây sau khi huấn luyện.

```python
def _gini(self, labels):
    # Tính chỉ số Gini cho tập nhãn
    m = len(labels)
    if m == 0: return 0
    counts = Counter(labels)
    impurity = 1.0
    for count in counts.values():
        p = count / m
        impurity -= p**2
    return impurity
```

- Định nghĩa hàm nội bộ (`_gini`) thuộc class `DecisionTreeClassifierFromScratch` .
- Nhận vào `labels` : danh sách các nhãn của tập dữ liệu hiện tại (ví dụ: `['setosa', 'setosa', 'versicolor', 'versicolor', 'virginica']`).

Tính số lượng phần tử trong danh sách labels, gán vào biến m.m chính là **tổng số mẫu** trong tập hiện tại.

Ví dụ: nếu labels = ['A', 'A', 'B', 'B', 'B'] → m = 5

Nếu không có nhãn nào (tức tập rỗng), thì trả về độ không thuần = 0.

Vì tập rỗng thì không có gì để phân biệt → không có "không thuần khiết". Gini của tập rỗng = 0 (theo quy ước).

Dùng Counter để đếm số lần xuất hiện của mỗi nhãn trong labels.

Kết quả là một từ điển {label: count}

ví dụ: {'A': 2, 'B': 3}

-

Khởi tạo độ không thuần là 1.0 – đây là giá trị tối đa (có thể bị trừ dần đi sau).

- **for count in counts.values():**
- Duyệt qua số lượng mẫu của từng nhãn.

- ***p = count / m***

- Tính xác suất (tỉ lệ) xuất hiện của nhãn đó:

Tính xác suất (tỉ lệ) xuất hiện của nhãn đó:

$$p_i = \frac{\text{số mẫu lớp i}}{\text{tổng số mẫu}}$$

📌 Ví dụ: nếu có 3 mẫu "B" trong 5 → p = 3/5 = 0.6

impurity -= p**2

- Trừ bình phương xác suất khỏi biến impurity.
- Vì Gini = 1 - tổng bình phương xác suất.
- Trả về độ không thuần Gini đã tính xong.

Gini càng thấp Càng “thuần”, tốt để dừng phân chia

```python
def _best_split(self, X, y):
    best_gain = 0
    best_feat, best_thresh = None, None
    current_impurity = self._gini(y)
    n_features = len(X[0])
    # Duyệt từng thuộc tính để tìm ngưỡng phân chia tốt nhất
    for feature in range(n_features):
        values = sorted(set(x[feature] for x in X))
        for i in range(1, len(values)):
            thresh = (values[i-1] + values[i]) / 2
            left_y = [y[j] for j,x in enumerate(X) if x[feature] <= thresh]
            right_y = [y[j] for j,x in enumerate(X) if x[feature] >  thresh]
            if not left_y or not right_y:
                continue
```

Tìm thuộc tính (feature) nào và ngưỡng chia (threshold) nào giúp **giảm Gini nhiều nhất**, tức là tách tập thành hai phần "thuần" nhất có thể.

- X: ma trận đặc trưng (list of lists)
- y: danh sách nhãn tương ứng

best_gain: theo dõi độ giảm impurity tốt nhất đã tìm được.

best_feat, best_thresh: theo dõi vị trí và ngưỡng tạo ra best_gain đó.

-Duyệt từng cột (feature) trong X, để kiểm tra chia tại từng feature đó.
-Tập các giá trị duy nhất trong cột feature đang xét.

- set loại bỏ trùng lặp.
- sorted để duyệt theo thứ tự tăng dần → chuẩn bị lấy các ngưỡng ở giữa.

Ví dụ: với feature = chiều dài cánh hoa, nếu có các giá trị [1.0, 1.4, 1.4, 1.7] thì values = [1.0, 1.4, 1.7].

Duyệt các ngưỡng chia khả thi giữa các giá trị

Lấy trung điểm giữa 2 giá trị liên tiếp → đó là một ngưỡng chia tiềm năng.

Ví dụ:

values = [1.0, 1.4, 1.7]

- thresh = (1.0 + 1.4)/2 = 1.2, rồi (1.4 + 1.7)/2 = 1.55

current_impurity: độ không thuần Gini hiện tại (chưa chia).

n_features = len(X[0])

Số lượng đặc trưng (feature), tức là số cột của mỗi dòng trong X.Ví dụ: nếu X[0] = [5.1, 3.5, 1.4, 0.2] thì n_features = 4

Duyệt từng dòng dữ liệu x trong X, nếu giá trị tại feature đó ≤ ngưỡng thì cho vào nhánh trái.

- Ngược lại, cho vào nhánh phải.
- left_y, right_y: chứa nhãn tương ứng của các mẫu ở mỗi nhánh.

Nếu 1 trong 2 nhánh rỗng (ngưỡng quá lệch), thì bỏ qua → không chia được.

```
        # Tính Gini con trái và phải
        gini_left = self._gini(left_y)
        gini_right = self._gini(right_y)
        p_left = len(left_y) / len(y)
        gain = current_impurity - (p_left * gini_left + (1 - p_left) * gini_right)
        if gain > best_gain:
            best_gain, best_feat, best_thresh = gain, feature, thresh
    return best_gain, best_feat, best_thresh
```

Tính Gini của 2 nhánh trái – phải.

- **p_left = len(left_y) / len(y)**
- Tính **tỉ lệ phần trăm số mẫu rơi vào nhánh trái**.
- Dùng để tính trung bình trọng số Gini sau chia.

Nếu chia này giúp tăng IG, thì lưu lại chia này là tốt nhất tính đến hiện tại.

Trả về bộ 3:

- gain: độ cải thiện Gini
- feature: chỉ số cột nào chia
- thresh: ngưỡng chia bao nhiêu

- Tính độ giảm Gini (Information Gain)

$$IG = Gini_{hiệntại} - (p_{trái} \cdot Gini_{trái} + p_{phải} \cdot Gini_{phải})$$

- IG (Information Gain) = độ cải thiện sau khi chia.
- Mục tiêu là **chọn chia để IG cao nhất**.

```
def _build_tree(self, X, y, depth=0):
    # Nếu tất cả nhãn giống nhau hoặc không thể chia thêm, tạo lá
    if len(set(y)) == 1 or depth == self.max_depth or len(y) < self.min_samples_split:
        return {'type': 'leaf', 'class': Counter(y).most_common(1)[0][0]}
    gain, feature, thresh = self._best_split(X, y)
    # Nếu không còn cải thiện, tạo lá
    if gain == 0 or feature is None:
        return {'type': 'leaf', 'class': Counter(y).most_common(1)[0][0]}
```

Hàm _build_tree nhận vào dữ liệu X, nhãn y, và độ sâu hiện tại depth, để **xây toàn bộ cây phân loại**.

- nhận X, y là tập dữ liệu hiện tại.
- depth: độ sâu hiện tại của nút trong cây (gốc có depth = 0).
- Hàm sẽ gọi lại chính nó (đệ quy) cho nhánh trái và phải sau mỗi lần chia.

Dòng 2–3: Điều kiện dừng – tạo lá

nếu xảy ra **một trong ba điều kiện dừng**, thì cây **không chia nữa**, trả về lá:

Tất cả nhãn giống nhau: len(set(y)) == 1

Đã đạt tới độ sâu tối đa

Quá ít mẫu để chia tiếp:

Khi dừng, ta chọn **nhãn phổ biến nhất** làm lớp

Dòng 6–7: Không cải thiện → tạo lá
Nếu không tìm thấy chia nào làm giảm Gini → dừng chia → tạo lá.

Counter(y) nó đếm số lần xuất hiện của từng phần tử trong y.

.most_common(1):

- Trả về **1 phần tử phổ biến nhất** cùng với số lần xuất hiện.
- Kết quả sẽ là một **danh sách chứa tuple**: [(label, count)]

[0][0]:

- most_common(1) trả về [(label, count)] → một danh sách chứa tuple.
- [0] → lấy phần tử đầu tiên của danh sách.
- [0][0] → lấy phần tử đầu tiên của tuple, tức là **label phổ biến nhất**.

dự đoán tại lá:

Counter(y).most_common(1)[0][0]
Ví dụ: nếu y = ['A', 'A', 'B'] → nhãn phổ biến nhất là 'A'.
Dòng 4: Gọi tìm ngưỡng chia tốt nhất

```
# Nếu không còn cải thiện, tạo lá
if gain == 0 or feature is None:
    return {'type': 'leaf', 'class': Counter(y).most_common(1)[0][0]}
# Chia dữ liệu theo ngưỡng tìm được
left_X, left_y, right_X, right_y = [], [], [], []
for xi, yi in zip(X, y):
    if xi[feature] <= thresh:
        left_X.append(xi); left_y.append(yi)
    else:
        right_X.append(xi); right_y.append(yi)
```

Nếu không tìm thấy chia nào làm giảm Gini → dừng chia → tạo lá.
thực hiện **tách dữ liệu** tại một điểm chia (ngưỡng thresh) trên **một đặc trưng (feature)**.

- Trái nếu xi[feature] ≤ thresh
- Phải nếu xi[feature] > thresh
- left_X chứa đặc trưng, left_y chứa nhãn tương ứng.

- **for xi, yi in zip(X, y):**
- Duyệt qua từng cặp (xi, yi) trong tập dữ liệu:
  - xi: một mẫu dữ liệu (vector các đặc trưng, ví dụ [5.1, 3.5, 1.4, 0.2])
  - yi: nhãn tương ứng (ví dụ 'setosa')
- **if xi[feature] <= thresh:**
- Kiểm tra giá trị tại đặc trưng số feature:
  - Ví dụ xi[feature] = 2.5
  - thresh = 3.0
    ⇒ **2.5 <= 3.0**, thì điểm đó thuộc **nhánh trái**

left_X.append(xi);left_y.append(yi)

- Nếu điều kiện đúng, thêm vào nhánh trái:
  - xi → left_X
  - yi → left_y

```
# Đệ quy xây dựng cây con trái và phải
left_tree  = self._build_tree(left_X, left_y, depth+1)
right_tree = self._build_tree(right_X, right_y, depth+1)
return {'type': 'node', 'feature': feature, 'threshold': thresh,
        'left': left_tree, 'right': right_tree}
```

left_tree = self._build_tree(left_X, left_y, depth+1) Gọi đệ quy để **xây dựng cây con trái**

left_X: dữ liệu ở nhánh trái

left_y: nhãn tương ứng

depth+1: tăng độ sâu thêm 1 vì đang xuống 1 cấp

- Tạo một **nút quyết định (decision node)** trong cây với thông tin:
- type': 'node': Đây là nút nội bộ, không phải nút lá

'feature': chỉ số của đặc trưng dùng để chia

- 'threshold': ngưỡng dùng để chia

'left': cây con bên trái (xây từ left_X, left_y)

'right': cây con bên phải

luồng hoạt độngMỗi lần gọi _build_tree(...):

1. Kiểm tra điều kiện dừng → tạo **nút lá**
2. Nếu chưa dừng:
   a. Tìm **feature và threshold tối ưu**
   b. Chia X, y thành 2 phần trái và phải
   c. Gọi **đệ quy** để xây tiếp trái và phải
   d. Tạo một **nút quyết định** và trả về

```
def fit(self, X, y):
    self.tree = self._build_tree(X, y)
```

```
def _predict_one(self, xi, node):
    if node['type'] == 'leaf':
        return node['class']
    if xi[node['feature']] <= node['threshold']:
        return self._predict_one(xi, node['left'])
    else:
        return self._predict_one(xi, node['right'])

def predict(self, X):
    return [self._predict_one(xi, self.tree) for xi in X]
```

- Hàm fit(self, X, y)Hàm này là để **huấn luyện mô hình**.
- Nó gọi đến _build_tree(X, y) để xây dựng toàn bộ **cây quyết định** từ dữ liệu X và nhãn y.
- Sau đó lưu vào self.tree để dùng sau này khi dự đoán.

Dự đoán **1 mẫu duy nhất** xi (ví dụ như 1 dòng trong X) bằng cách đi theo cây từ gốc đến lá.

- Nếu node hiện tại là **nút lá** → trả về lớp đã lưu ở đó (node['class']).Đây là điểm dừng của đệ quy.

- Dự đoán **1 mẫu duy nhất** xi (ví dụ như 1 dòng trong X) bằng cách đi theo cây từ gốc đến lá.
- Nếu node hiện tại là **nút lá** → trả về lớp đã lưu ở đó (node['class']).
- Đây là điểm dừng của đệ quy.

Nếu **giá trị đặc trưng** của xi tại feature đang xét ≤ threshold → đi sang **nhánh trái**.

- Nếu không thì đi sang **nhánh phải**.
- Nếu **giá trị đặc trưng** của xi tại feature đang xét ≤ threshold → đi sang **nhánh trái**.Nếu không thì đi sang **nhánh phải**.Gọi lại _predict_one một cách **đệ quy**, cho đến khi gặp lá.
-

Hàm predict(self, X) Dự đoán toàn bộ tập dữ liệu X (nhiều dòng).

- Duyệt từng dòng xi trong X
- Gọi _predict_one để dự đoán nhãn cho dòng đó
- Trả về danh sách nhãn dự đoán


```
def fit(self, X, y):
    self.tree = self._build_tree(X, y)

def _predict_one(self, xi, node):
    if node['type'] == 'leaf':
        return node['class']
    if xi[node['feature']] <= node['threshold']:
        return self._predict_one(xi, node['left'])
    else:
        return self._predict_one(xi, node['right'])

def predict(self, X):
    return [self._predict_one(xi, self.tree) for xi in X]
```


_predict_one(self, xi, node) Dự đoán **1 mẫu dữ liệu duy nhất** xi bằng cách **duyệt cây** từ gốc đến lá.

- Nếu node hiện tại là **lá** ('leaf'), thì trả về **lớp nhãn** lưu ở 'class'.
- Nếu không phải lá, thì:
- Kiểm tra thuộc tính xi[node['feature']]: nếu nhỏ hơn hoặc bằng threshold, đi sang **nhánh trái** ('left').
- Ngược lại, đi sang **nhánh phải** ('right').
- Dùng **đệ quy** để tiếp tục kiểm tra cho đến khi gặp lá.

predict(self, X) Dự đoán **toàn bộ tập dữ liệu X** gồm nhiều mẫu.Duyệt từng mẫu xi trong X.

- Gọi _predict_one(xi, self.tree) để dự đoán nhãn cho mẫu đó.
- Kết quả trả về là **một danh sách các nhãn dự đoán**.

```
# Tạo và đánh giá mô hình cây phân loại
clf = DecisionTreeClassifierFromScratch(max_depth=3)
clf.fit(X, y)
predictions = clf.predict(X)
accuracy = sum(p == t for p, t in zip(predictions, y)) / len(y)
print("Độ chính xác (scratch):", accuracy)
```

Khởi tạo một đối tượng mô hình cây quyết định có chiều sâu tối đa là 3.

- fit(X, y): gọi phương thức huấn luyện (fit) để **dạy mô hình học từ dữ liệu đầu vào X (các đặc trưng)** và **nhãn y (loài hoa)**.
- Dòng này sẽ chạy qua toàn bộ cây, áp dụng thuật toán chia nhánh (dựa trên Gini) để xây dựng cây. **Mục tiêu :** Mô hình học từ dữ liệu huấn luyện.

predict(X): gọi phương thức dự đoán đã tự cài, áp dụng mô hình vừa học được lên tập X.Trả về danh sách predictions chứa loài hoa dự đoán cho từng điểm dữ liệu.

(accuracy) Tính độ chính xác

- zip(predictions, y): gom cặp (dự đoán, thật) lại với nhau.
- p == t: kiểm tra dự đoán có đúng không.
- sum(...): đếm số lượng dự đoán đúng.
- len(y): tổng số mẫu.
- Cuối cùng chia ra để tính **tỉ lệ dự đoán đúng (accuracy)**.

CÂY HỒI QUY :

```
def _variance(self, y):
    if len(y) == 0:
        return 0
    mean = sum(y) / len(y)
    return sum((val - mean) ** 2 for val in y) / len(y)
```

Nhận đầu vào là y – danh sách các giá trị mục tiêu (giá trị liên tục)

Dòng 2: if len(y) == 0:

- Kiểm tra nếu danh sách y rỗng (không có phần tử nào).
- Trường hợp này xảy ra khi chia nhánh mà không có mẫu nào rơi vào một nhánh cụ thể.
- Nếu không có dữ liệu, phương sai = 0.
- Điều này giúp tránh lỗi chia cho 0 và coi như không có độ biến thiên.

Dòng 4: mean = sum(y) / len(y)

- Tính **giá trị trung bình (mean)** của danh sách y.

Tính **phương sai** theo công thức: