

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.

1. Cơ Chế Phân Chia Nút

Bước 1: Tính "độ không đồng nhất" của nút cha

- ❖ **Phân loại:** Gini Impurity/Entropy.
- ❖ **Hồi quy:** Phương sai/MSE.

Bước 2: Thử các cách phân chia

- ❖ Với mỗi thuộc tính và ngưỡng (liên tục) hoặc giá trị (rời rạc):
 - Chia dữ liệu → tạo **các nút con**.
 - Tính "độ không đồng nhất" **cho từng nút con**.
 - Tính **trung bình có trọng số** (theo số lượng mẫu).

Bước 3: Chọn phân chia tối ưu

- ❖ **Công thức chung:**
- ❖ Độ cải thiện = Độ không đồng nhất(nút cha) - Trung bình trọng số(nút con)
- ❖ Chọn phân chia có **độ cải thiện lớn nhất**.
- ❖ **Bước 4: Lặp đệ quy** → Dừng khi gặp điều kiện dừng.

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 :

$$\begin{array}{cc} \text{Gini} & \text{Entropy} \\ \text{Index} & \\ I_G = 1 - \sum_{j=1}^c p_j^2 & I_H = - \sum_{j=1}^c p_j \log_2(p_j) \end{array}$$

Gini Index là một thước đo độ tinh khiết của tập dữ liệu, thường được sử dụng để đánh giá mức độ hỗn loạn giữa các lớp trong một nút của cây quyết định.

- c : số lớp.
- p_j : xác suất thuộc lớp j .

Độ lộn xộn Gini càng nhỏ khi một lớp chiếm ưu thế, lớn nhất khi các lớp đều bằng nhau.

Ví dụ với 2 lớp, $p_1 = 0.7$, $p_2 = 0.3$:

- $\sum p_j^2 = 0.7^2 + 0.3^2 = 0.49 + 0.09 = 0.58$
- $IG = 1 - 0.58 = 0.42$

Entropy là một thước đo sự không chắc chắn hoặc độ hỗn loạn trong tập dữ liệu. Nó dựa trên lý thuyết thông tin và được sử dụng để đánh giá mức độ lẫn lộn giữa các lớp.

- $\log_2(p_j)$: log cơ số 2, đo “lượng thông tin” của lớp j .
- Khoảng giá trị: từ 0 (hoàn toàn thuần nhất) đến $\log_2(2/c) \approx \log_2(c)$ (rất hỗn loạn).

Ví dụ với 2 lớp, $p_1 = 0.5$, $p_2 = 0.5$:

$$I_H = -[0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)] = 1 \text{ bit}$$

Nếu $p_1 = 1$, $p_2 = 0$ thì $I_H = 0$

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:

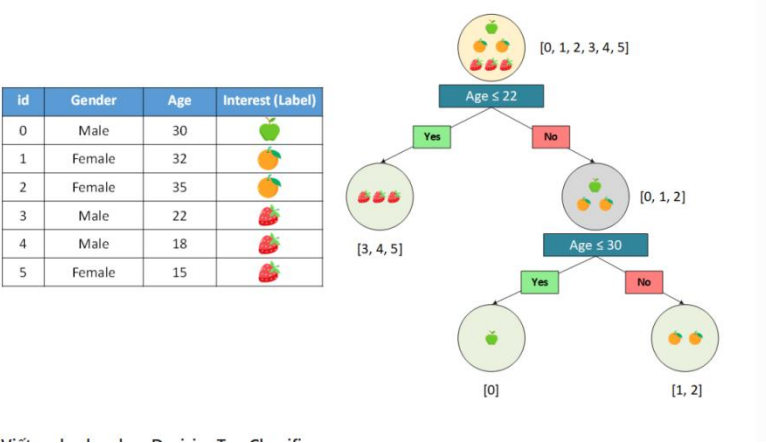
- Tập dữ liệu S gồm n mẫu: $S = \{(x_i, y_i)\}_{i=1}^n$
- Tập này được chia thành:
 - Nhánh trái S_L (có n_L mẫu)
 - Nhánh phải S_R (có n_R mẫu)

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

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 :



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 $Gini \approx 0.444$, nhóm *Female* (3 mẫu) có nhãn {Orange, Orange, Strawberry} với $Gini \approx 0.444$. Trọng số gộp $Gini_{Gender} = 63 \times 0.444 + 63 \times 0.444 = 0.444$.

Nếu phân chia theo **Age**, ta xem xét các ngưỡng tiềm năng (ví dụ giữa 22 và 30 là ngưỡng 26) rồi tính Gini. $Gini_{Age \leq 22} = 0.222$ 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.

<pre>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()</pre>	<ul style="list-style-type: none">• 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
<pre>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</pre> <ul style="list-style-type: 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.	<ul style="list-style-type: none">• 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.•

```
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`.

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".

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

Kết quả là một từ điển `{label: count}`

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.
- $p = \text{count} / m$
- Tính xác suất (tỉ lệ) xuất hiện của nhãn đó:

$\text{impurity} -= p**2$

- Trừ bình phương xác suất khỏi biến `impurity`.
- Vì $\text{Gini} = 1 - \text{tổng bình phương xác suất}$.
- Trả về độ không thuần Gini đã tính xong.

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

```
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
            # Tính độ không thuần Gini cho hai nhánh
            left_impurity = self._gini(left_y)
            right_impurity = self._gini(right_y)
            gain = current_impurity - (len(left_y)/len(X)*left_impurity + len(right_y)/len(X)*right_impurity)
            if gain > best_gain:
                best_gain = gain
                best_feat, best_thresh = feature, thresh
    return best_feat, best_thresh
```

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.

`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 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]`.

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]`

- $\text{thresh} = (1.0 + 1.4)/2 = 1.2$, rồi $(1.4 + 1.7)/2 = 1.55$

Duyệt từng dòng dữ liệu `x` trong `X`, nếu giá trị tại feature đó \leq 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.

- **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

ếu 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 dự đoán tại lá:

```
# 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

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

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**.

- **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)

<p>tương ứng.</p>	<ul style="list-style-type: none">Nếu điều kiện đúng, thêm vào nhánh trái:<ul style="list-style-type: none">xi → left_Xyi → left_y
<pre># 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}</pre> <p>left_X: dữ liệu ở nhánh trái</p> <p>left_y: nhãn tương ứng</p> <p>depth+1: tăng độ sâu thêm 1 vì đang xuống 1 cấp</p>	<ul style="list-style-type: none">Tạo một nút quyết định (decision node) trong cây với thông tin:type: 'node': Đây là nút nội bộ, không phải nút lá <p>'feature': chỉ số của đặc trưng dùng để chia</p> <ul style="list-style-type: none">'threshold': ngưỡng dùng để chia <p>'left': cây con bên trái (xây từ left_X, left_y)</p> <p>'right': cây con bên phải</p>
<pre>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]</pre> <ul style="list-style-type: none">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.	<ul style="list-style-type: none">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. <p>Nếu giá trị đặc trưng của xi tại feature đang xét ≤ threshold → đi sang nhánh trái.</p> <ul style="list-style-type: none">Nếu không thì đi sang nhánh phải.Gọi lại _predict_one một cách đệ quy, cho đến khi gặp lá. <p>Hàm predict(self, X) Dự đoán toàn bộ tập dữ liệu X (nhiều dòng).</p> <ul style="list-style-type: none">Duyệt từng dòng xi trong XGọi _predict_one để dự đoán nhãn cho dòng đóTrả về danh sách nhãn dự đoán
<pre># 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)</pre>	

