Decision Tree

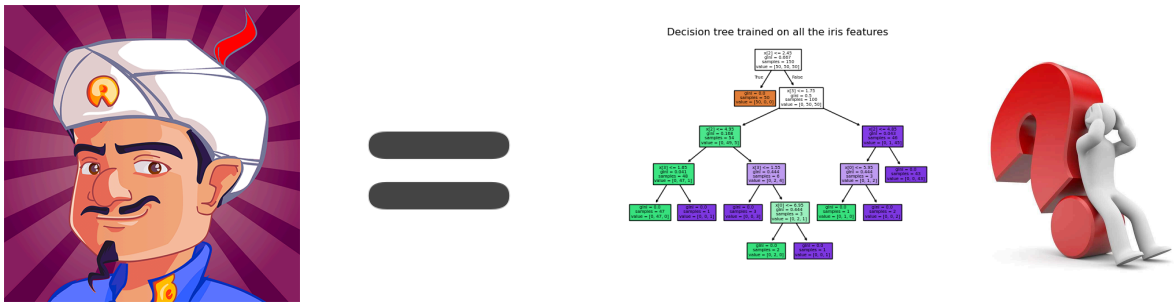
Hồ Nguyễn Phú

24 tháng 8, 2025

I. Giới thiệu:

Thuật toán Cây Quyết Định (Decision Tree) giúp chúng ta đưa ra quyết định thông qua việc phân tích các lựa chọn rẽ nhánh và kết quả của chúng (Müller & Guido, 2016; GeeksforGeeks, 2025). Thuật toán có thể dùng cho cả tác vụ hồi quy (regression) và phân loại (classification).

Decision Tree có cấu trúc dạng cây, với mỗi node là một lần rẽ nhánh thông qua các "câu hỏi" if–else (Müller & Guido, 2016). Một ví dụ **tượng trưng** là game Akinator, nơi thần đèn sẽ hỏi các câu hỏi liên quan đến một khái niệm, nhân vật, hoặc bất cứ thứ gì người chơi nghĩ ra trong đầu, để dần thu hẹp và đi đến một kết luận.



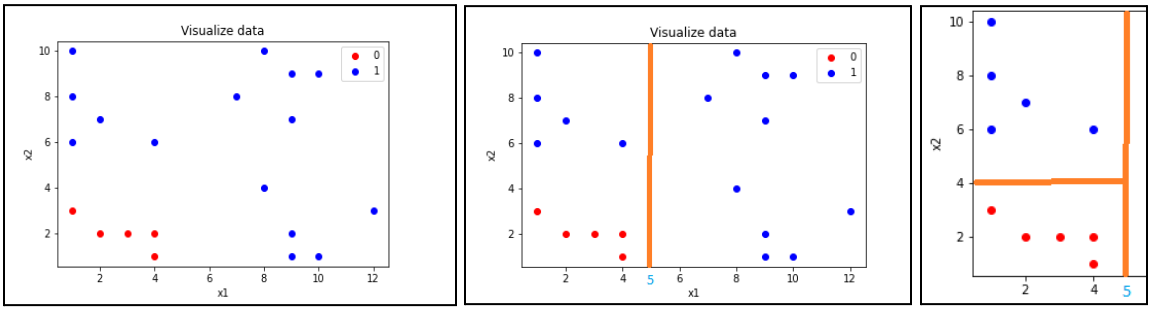
Hình 1. Game Akinator (chỉ minh họa – Akinator dùng thuật toán khác).

Về thuật ngữ, tham khảo Machine Learning cơ bản (2018) và Nguyễn (2021) thì Decision Tree bao gồm các khái niệm: *root node, branches, internal nodes* và *leaf nodes* (*node lá*, còn gọi là *terminal node*), *non–leaf node* (node không phải *leaf node*) và *child node* (node con). Trong đó, *root node* là điểm bắt đầu cho toàn bộ dataset; tại đây, câu hỏi *yes / no* đầu tiên được đặt ra (với binary decision tree). Từ đây, dataset được chia thành các subset nhỏ hơn dựa trên câu trả lời tại các node – tạo thành các branches. Mỗi branch là một nhánh rẽ của *non–leaf node* bên trên, và dẫn đến một *child node*. Cây tiếp tục rẽ nhánh ở mỗi *child node* cho đến khi không còn câu hỏi hữu ích nào, hoặc đạt tới độ sâu tối đa (*max–depth*) thì ngừng lại; lúc này, ta có *leaf node* (*terminal node*), tức nơi đưa ra dự đoán hoặc quyết định cuối cùng.

Để trực quan hóa về mặt Toán học, ta hình dung dataset ban đầu như tập hợp các điểm dữ liệu (hình 2a). Decision Tree sẽ lần lượt đặt ra các điều kiện và chia dataset ra dựa trên điều kiện đó. Đầu tiên, "nếu $x_1 > 5$ đúng thì tất cả các dữ liệu thuộc lớp 1," nên điều kiện đặt ra là $x_1 > 5$ (hình 2b). Tại đây ta chia dataset ban đầu thành hai subset, với subset $x_1 \leq 5$ vẫn chứa 2 class là đỏ và xanh. Do đó, ta tiếp tục đặt điều kiện $x_2 > 4$ để tách hoàn toàn class đỏ và xanh (hình 2c).

Vào khoảng những năm 1960s, Decision Tree bị các nhà thống kê coi là một "ngõ cụt" (dead–end), vì nó thiếu đi cơ sở lý thuyết hoàn chỉnh (Steinberg, 2009). Tuy nhiên, Richard Olshen và Jerome Friedman đã dựa vào sự tương đồng giữa Nearest Neighbor classifier và terminal node của Decision Tree để nhận định rằng nó cũng có cơ sở lý thuyết vững chắc (Steinberg, 2009).

Điểm đặc biệt của Decision Tree là nó hoạt động tốt với các feature (hay attribute) dạng categorical và liên tục. Thông thường, data dạng categorical thường yêu cầu đưa về one–hot encoding còn data liên tục cần chuẩn hóa. Nhưng do đặc điểm của mình, Decision Tree không cần thực hiện các bước đó "vì các câu hỏi đều có thể được đưa về dạng đúng sai" (Machine learning cơ bản, 2018). Một số biến thể của Decision Tree cũng hoạt động tốt với dữ liệu thiếu (missing data).



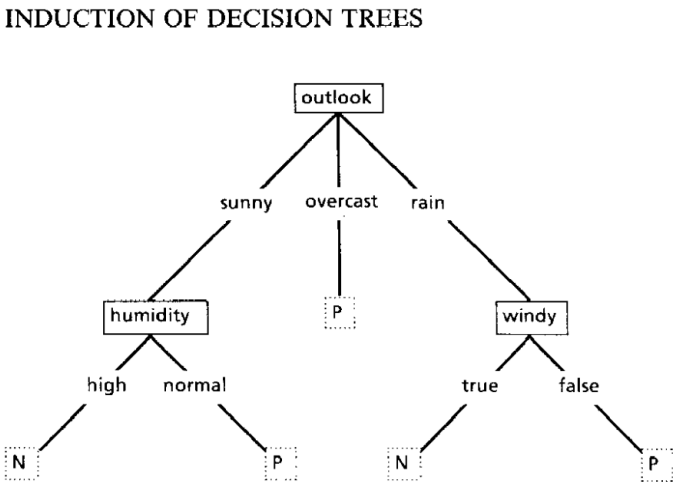
(a) Các điểm dữ liệu *(b) Phân nhánh lần 1* *(c) Kết quả cuối cùng*

Hình 2. Minh họa Decision Tree (Nguyễn, 2021).

Nội dung ta cần thảo luận qua trong phần lý thuyết của Decision Tree bao gồm: các thuật toán (như ***ID3***, ***C4.5***, ***CART***, ***CHAID***, ***MARS***) cùng với Toán học tương ứng, và các phương pháp pruning (cắt tỉa).

II. Iterative Dichotomiser 3 (ID3):

ID3 được ra đời từ rất sớm và phổ biến (Machine learning cơ bản, 2018). Trong ID3, các thuộc tính đều ở dạng categorical. Tại mỗi bước, ta chọn một thuộc tính tốt nhất để phân chia dữ liệu vào các nhóm khác nhau (như outlook thành nhóm sunny, overcast, rain trong hình 6), tạo thành các *child node*. Tiếp tục thực hiện bước trên với *child node* cho đến khi đạt đến *leaf node*.



Hình 6. Một cây quyết định đơn giản – Quinlan (1986)

Chọn ra một thuộc tính **tốt nhất tại mỗi bước** như thế (Kempe và Rosenberg, 2019) được gọi là cách chọn *greedy* (*tham lam*) – có thể không tối ưu, nhưng gần với tối ưu và đơn giản hóa bài toán (Machine learning cơ bản, 2018).

Cách phân chia tốt là gì? Đó là để cho dữ liệu trong mỗi *child node* hoàn toàn thuộc vào một class – tức child node trở thành *leaf node* và không cần phân nhánh nữa (Machine learning cơ bản, 2018). Nếu vẫn có sự lẫn lộn lớn giữa các class trong một *child node* thì sự phân chia chưa thực sự tốt. Và để đánh giá điều này, ta dùng "hàm số đo *độ tinh khiết* (*purity*), hoặc độ vẩn đục (*impurity*) của một phép phân chia." Hàm đạt cực tiểu khi các điểm dữ liệu trong mỗi *child node* có cùng một class và đạt cực đại khi mỗi child node chứa dữ liệu hỗn tạp, thuộc nhiều class khác nhau (Machine learning cơ bản, 2018; Morgan, 2014). Một số hàm được sử dụng là Entropy (ID3) và Gini Impurity (CART).

1. **Hàm entropy:** (xem thêm về cross–entropy trong mục 2a của **bài viết này**)

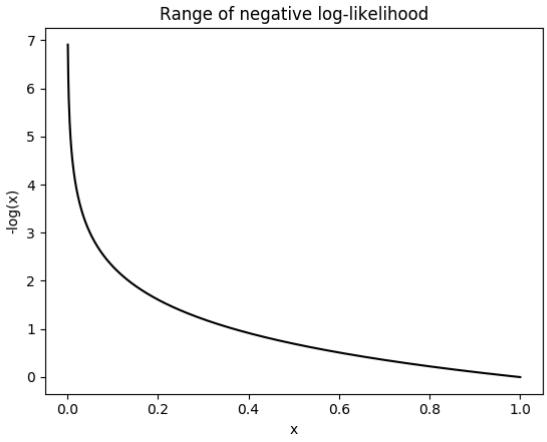
Entropy là đại lượng đo độ không chắc chắn và hỗn loạn (Wikipedia, 2025). Nói một cách trừu tượng thì, entropy là lượng "thông tin" cần để mô tả một tín hiệu, xét đến phân phối xác suất của các biến cố trong không gian mẫu. Hàm entropy có dạng như sau:

$$H = -\sum_{i=1}^{N} p_i \log(p_i)$$

Trong đó, *–log(p_i)* là *self–information* của xác suất p_i , và hàm entropy *H(p)* là giá trị kỳ vọng của *self–information* đó.

a. **Self–information:**

Bước đầu tiên ta tính *–log(p)* để có một vector self–information tương ứng với phân phối xác suất *p*. Do đặc điểm của hàm log, khi p nằm trong khoảng $(0,\ 1]$ thì *–log(p)* tiến từ +∞ đến *0*. Điều này có nghĩa: xác suất p của biến cố A càng nhỏ (gần *0*), thì khi A xảy ra, lượng thông tin biến cố A chứa sẽ càng lớn (tiến tới +∞).



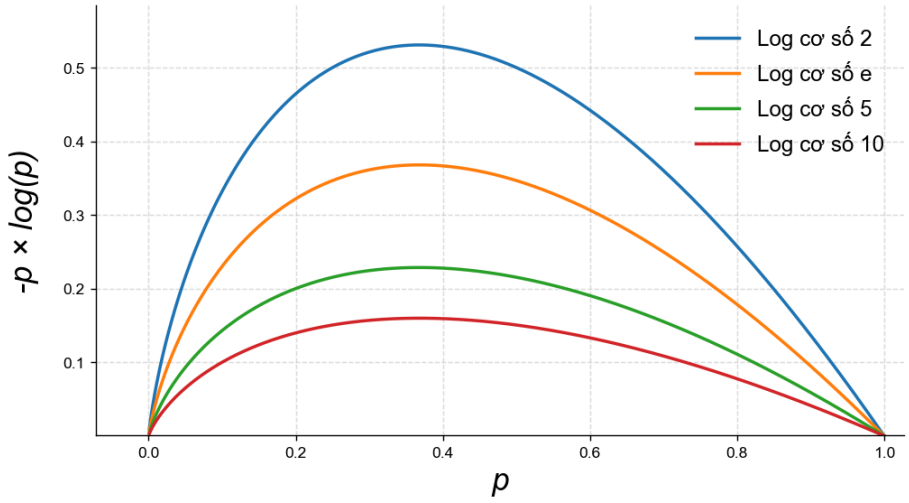
Hình 3. Đồ thị negative log–likelihood

b. **Entropy:**

Sau khi có được self–information của từng biến cố p_i trong không gian mẫu *p,* ta sẽ tính giá trị kỳ vọng của chúng thông qua tổng có trọng số (*weighted sum*) như trên công thức. Giá trị nhận được sẽ là entropy của cả phân phối *p*.

Để phân tích thêm, một điểm khá đặc biệt của tổng có trọng số trên là đồ thị các số hạng thành phần của nó (tạm gọi là *w*). Đặt *w(p)* = $-\ p\ \times\ log(p)$.

Với *self–information*, p càng gần 1 thì *self–information* càng nhỏ, đạt cực tiểu bằng 0 tại $p\ =\ 1$. Nhưng khi nhân trọng số *self–information* với chính *p*, ta có đồ thị hàm số *w(p)*, chuyển từ hình dạng như hình 3 sang hình 4.



Hình 4. *Self–information có trọng số* với từng hệ cơ số log.

Như vậy, dù $\lim_{p\to 0}\ (-\ log(p))\ =\ \infty$, nhưng $\lim_{p\to 0}\ (-\ p\ \times\ log(p))$ lại bằng 0 (chứng minh như hình 5 hoặc theo L' Hospital).

Chứng minh:

Đặt: $-\ t\ =\ log(p) \Leftrightarrow p\ =\ e^{-t}$

Thay $p\ =\ e^{-t}$ vào $-\ p\ \times\ log(p)$, ta được:

$$-\ e^{-t}\ \times\ log(e^{-t})\ =\ -\ e^{-t}\ \times\ (-\ t)$$
$$=\ e^{-t}\ \times\ t$$

Theo đó, hàm $-\ p\ \times\ log(p)$ sẽ trả về giá trị càng nhỏ khi p càng gần hai đầu mút của khoảng $[0;\ 1]$, và đạt giá trị càng lớn khi p nằm ở giữa khoảng (đồ thị hơi lệch về bên trái). Điều này có nghĩa, ta có hai trường hợp khi xét cả không gian mẫu: ***(a)*** nếu phân phối có một biến cố với xác suất xảy ra rất lớn (tiệm cận 1), và các biến cố còn lại rất nhỏ (gần bằng 0); và ***(b)*** phân phối có xác suất phân bố khá đều. Với ***(a)*** thì entropy sẽ thấp, do ta gần như xác định dễ dàng "điều gì sẽ xảy ra"; nhưng với ***(b)***, do có quá nhiều biến cố xác suất tương đương ("hỗn loạn"), nên entropy sẽ cao (bàn thêm trong mục Gini Index của CART bên dưới).

c. Code hàm entropy (Numpy):

```
from typing import Union
import numpy as np
from numpy import array

def entropy(probs: array, b: Union[int, float] = 2) -> float:
    """
    Calculate the entropy of a probability distribution.

    Parameters
    ----------
    probs : numpy.array
        An array of probabilities (must sum to 1).
    b : int or str, optional
        The logarithm base. Common choices:
        - 2   : entropy in bits (default).
        - np.e : entropy in nats (natural log).
        - 10  : entropy in bans (log base 10).

    Returns
    -------
    float
        The entropy of the distribution.
    """
    # Self-information
    probs = probs[probs > 0]
    info = -(np.log(probs) / np.log(b)) # -log_b(probs)

    # Expected self-information (entropy)
    weighted_info = probs*info
    return np.sum(weighted_info)
```

Hình 4. Code hàm tính entropy của một array phân phối xác suất

Trong code, ta cần lọc ra các giá trị 0 với mã "***probs = probs[probs > 0]***" để đảm bảo ổn định số học. Tính toán $log(p)$ với $p\ =\ 0$ sẽ trả về giá trị dương vô cùng, và đại diện cho giá

trị đó trong Python là ***float("inf")***. Trên lý thuyết, $\lim_{p \to 0} (-\ p \times log(p)) = 0$ (chứng minh bên trên); nhưng trong lập trình, "inf" nhân với *0* sẽ trả về giá trị ***NaN***. Để tránh điều này, ta loại bỏ các giá trị *0* đi, vì self–information của biến cố có xác suất bằng *0* thì bằng *0*.

Bước tính thứ hai "***info = –(np.log(probs) / np.log(b))***" là phép chuyển đổi cơ số cho log. Sau đó ta thực hiện tính giá trị kỳ vọng của các giá trị self–information và trả về kết quả là một số thực trong khoảng $[0; \infty)$.

Bên dưới là kết quả của một vài phân phối xác suất đầu vào cho hàm entropy:

```
P_1 = array([1., 0, 0, 0, 0])
P_2 = array([0.5, 0.2, 0.1, 0.1, 0.1])
P_3 = array([0.2, 0.2, 0.2, 0.2, 0.2])
P_4 = array([0.1, 0.1, 0.1, 0.1, 0.1,
             0.1, 0.1, 0.1, 0.1, 0.1])

# Print out entropy values
print(entropy(P_1)) # 0.0
print(entropy(P_2)) # 1.9609640474436814
print(entropy(P_3)) # 2.321928094887362
print(entropy(P_4)) # 3.321928094887362
```

Hình 5. Kết quả hàm tính entropy

2. Information gain:

Information gain (IG) của thuật toán ID3 là thông tin nhận được sau phép phân nhánh. Nó được tính theo công thức:

$$IG(P,\ A) = H(P) - \sum_{v \in values(A)} \frac{|P_v|}{|P|} \times H(P_v)$$

Trực giác đằng sau nó khá đơn giản. IG là thông tin chênh lệch giữa ***entropy phân phối dữ liệu ban đầu*** – H(P), và ***kỳ vọng của entropy các phân phối dữ liệu sau khi tách*** – vế thứ hai của phép tính. Ta tính kỳ vọng vì các nhóm dữ liệu có thể không có cùng số điểm dữ liệu.

3. **Code ID3 và implementation:** Link Github.

III. Classification and Regression Tree (CART):

CART là một binary decision tree (mỗi node phân làm 2 nhánh). Một số điểm đặc biệt của CART được đề cập bởi Steinberg (2009) và Kempe và Rosenberg (2019) bao gồm: **(1)** cây không cần đặt độ sâu (depth) tối đa khi phân chia (split), **(2)** sau đó được cắt tỉa (prune) ngược lại root thông qua cost–complexity pruning.

Với **(1)**, CART có thể xử lý được cả dữ liệu categorical và dữ liệu liên tục, và có cơ chế riêng để xử lý classification và regression (Steinberg, 2009). Ngoài ra, nó cũng có thể xử lý dữ liệu thiếu. Khác với ID3 có thể có max depth, CART sẽ split đến khi (a) đạt số mẫu tối thiểu một leaf node hoặc (b) "pure" với tất cả sample cùng một class (Kempe & Rosenberg, 2019).

Sâu hơn với **(2)**, mỗi lần prune ta sẽ loại bỏ một split (node?) đóng góp ít nhất vào performance của model trong training data (có thể nhiều node bị prune cùng lúc). Cơ chế CART này hướng đến việc tạo ra không chỉ một, mà là một sequence những nested pruned trees, tạo nên những ứng viên (candidate) cho cây tối ưu nhất (optimal tree); cây thỏa mãn "right sized" hoặc "honest" là cây **dự đoán tốt trên tập test độc lập** (!) (Steinberg, 2009, p. 181).

CART được dùng trong thuật toán Random Forest (Breiman, 2001) – đây là một trong những thuật toán nổi bật của Machine Learning, với số lượt trích dẫn rất cao (khoảng 160 ngàn lượt – gần bằng paper "Attention is All You Need" với 190 ngàn lượt).

1. Gini index:

Gini index tương tự IG, dùng để đánh giá sự phân chia ở node điều kiện (Machine learning cơ bản, 2018; Morgan, 2011, p. 5). Có thể xem thêm phần giải thích của Machine learning cơ bản (2018):

> Để tính Gini index, trước hết mình sẽ tính chỉ số Gini, chỉ số Gini tính ở từng node.
>
> $$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Trong đó C là số lớp cần phân loại, $p_i = \frac{n_i}{N}, n_i$ là số lượng phần tử ở lớp thứ i. Còn N là tổng số lượng phần tử ở node đó, $N = \sum_{i=1}^{N} n_i \Rightarrow \sum_{i=1}^{N} p_i = 1$.

Do $0 <= p_i <= 1 \forall i$ và $\sum_{i=1}^{N} p_i = 1$ nên:

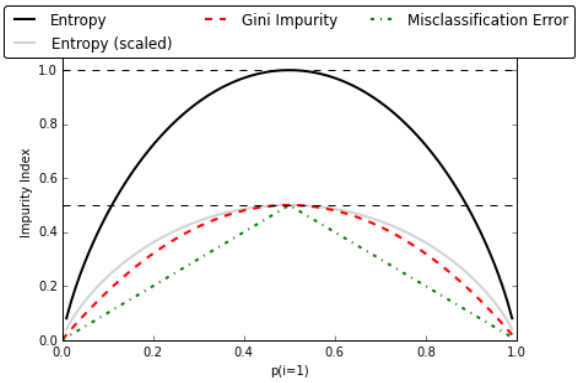
$\sum_{i=1}^{C} (p_i)^2 <= (\sum_{i=1}^{C} p_i)^2 = 1 \Rightarrow Gini \geq 0$, dấu bằng xảy ra khi $\exists j : p_j = 1$ và $p_k = 0 \, \forall k \neq j$

$\sum_{i=1}^{C} (p_i)^2 >= \frac{(\sum_{i=1}^{C} p_i)^2}{C} = \frac{1}{C} \Rightarrow Gini \leq \frac{C-1}{C}$, dấu bằng xảy ra khi $p_j = \frac{1}{C} \, \forall j$

Hình 6. Phân tích về Gini Index (Gini Impurity) của Machine learning cơ bản (2018).

Dựa trên phân tích đó ta cùng mở rộng về: ***(a)*** cực đại và cực tiểu của chỉ số Gini; ***(b)*** so sánh Gini Impurity và Entropy. Với ***(a)***, Gini Impurity đạt cực tiểu khi $\exists p_j = 1$, tức một biến cố chắc chắn xảy ra (lượng thông tin / entropy thấp); và ngược lại, khi xác suất phân phối đều cho các biến cố, với trường hợp cực đoan nhất là tất cả có xác suất bằng $\frac{1}{C}$, thì Gini Impurity đạt cực đại (tương tự với Entropy, như trường hợp của Binary Entropy Function). Với ***(b)***, trang Github đề xuất bởi Machine learning cơ bản (2018) cho rằng cả hai chỉ số Gini Impurity và Entropy thường cho kết quả tương tự nhau, và ta nên thử nghiệm với cắt tỉa cây thay vào đó ("different pruning cut–offs"); còn theo Phạm Đình Khánh (2021) thì Gini Impurity được dùng cho thuật toán CART của Scikit Learn.

Ngoài Entropy và Gini Impurity ta còn có Classification Error (hình 7). Nhưng chỉ số trên thường không nhạy cảm với thay đổi của xác suất các lớp (Kempe & Rosenberg, 2019, p. 31; **Github**).



Hình 7. So sánh các impurity criteria (Github).

2. Giải thích thuật toán:

a) Cơ chế phân chia (splitting):

Pseudo–code cho CART được đề cập bên hình 8. Nhìn chung thuật toán khá tương tự như ID3, khác ở điểm CART cho phép split với data dạng liên tục – thông qua "allowable splitting rule" (Steinberg, 2001).

```
BEGIN:  Assign all training data to the root node
        Define the root node as a terminal node

SPLIT:
New_splits=0
FOR every terminal node in the tree:
   If the terminal node sample size is too small or all instances in the
   node belong to the same target class goto GETNEXT
   Find the attribute that best separates the node into two child nodes
   using an allowable splitting rule
   New_splits+1
GETNEXT:
NEXT
```

Hình 8. Thuật toán CART (đơn giản hóa) được minh họa bởi Steinberg (2001).

Việc tìm ra điểm phân chia cho giá trị liên tục (split point) không quá khó khăn. Với một feature x bao gồm n giá trị được sắp xếp $x_1, x_2, \ldots, x_n$, ta chỉ cần $n - 1$ điểm tách ở giữa mỗi 2 điểm. Ví dụ, điểm tách giữa x_1 và x_2 sẽ là $s_1 = \frac{x_1 + x_2}{2}$ (Kempe & Rosenberg, 2019).

Cơ chế cho phân chia giá trị categorical trên lý thuyết sẽ rất phù hợp cho decision tree, nhưng các implementation thực tế lại dùng cách tách numerical thay vào đó (câu trả lời trên StackExchange). Theo Kempe và Rosenberg (2019), CART sẽ coi các class như những giá trị numerical và thực hiện tách. Tuy nhiên, độ phức tạp thuật toán sẽ rất lớn: với q giá trị không sắp xếp, ta có đến $2^{q-1} - 1$ cách phân chia (xem thêm về độ phức tạp để phân chia từng kiểu dữ liệu trong Kempe & Rosenberg, 2019). Vấn đề trên có thể giải quyết khi làm việc với binary classification (Kempe & Rosenberg, 2019): tính toán tỷ lệ class 0 cho từng giá trị của feature, sau đó sắp xếp feature theo thứ tự tăng dần / nhỏ dần. Với cách này ta có thể dùng luôn cách split point cho dữ liệu liên tục bên trên: với {A, B, C, D} ta sẽ tách thành {A}, {B, C, D}; {A, B}, {C, D}; {A, B, C}, {D} với độ phức tạp tuyến tính $O(q - 1)$ – về mặt lý thuyết, do đặc điểm hàm entropy và gini (hình 7), nên cách phân chia này có ý nghĩa và bao quát hết các trường hợp.

Mặt khác, implementation Decision Tree / Random Forest của Scikit Learn vào 2016 vẫn chưa hỗ trợ input dữ liệu categorical vào như là một string – ta phải chuyển nó thành one–hot encoding để xử lý theo cách numerical, thông qua OrdinalEncoder của Scikit Learn hoặc get_dummies() của Pandas (cuộc thảo luận trên StackExchange).

b) Cơ chế xử lý dữ liệu thiếu:

Với dữ liệu thiếu, ta có thể loại bỏ, interpolate với giá trị trung bình, hoặc để missing là một category; nhưng với decision tree, ta có thể thực hiện "*surrogate splits*" (Kempe & Rosenberg, 2019). Xem thêm Steinberg (2001, p. 189).

c) Cơ chế cắt tỉa (pruning):

Cắt tỉa được Kempe và Rosenberg (2019) miêu tả là quá trình chọn một internal node, sau đó loại bỏ tất cả child node của nó. Ta được một leaf node mới từ internal node cũ.

Gọi:

- T_0 : cây ban đầu ("big tree")
- $\widehat{R}(T)$: "empirical risk" của cây *T* (ví dụ như Squared Error, entropy)
- Do quá trình phân nhánh, mọi cây con *T* có $\widehat{R}(T)$ nhỏ hơn của T_0 .
- $|T|$ là số leaf node của cây *T* và là tiêu chí đánh giá độ phức tạp.

Với cost–complexity pruning, ta sẽ có hàm cost–complexity criterion với tham số α:

$$C_\alpha(T) = \widehat{R}(T) + \alpha|T|$$

Hàm trên cho phép ta điều chỉnh và đánh đổi giữa "*empirical risk*" (*rủi ro thực nghiệm?*) và độ phức tạp của cây. Với mỗi α , ta tìm một cây con *T* để giảm $C_\alpha(T)$ trên tập huấn luyện và dùng cross–validation để tìm ra α phù hợp (Kempe & Rosenberg, 2019). Theo cùng tác giả, do hàm cost–complexity không khả vi, nên ta có vô số subtree T của T_0. Tuy nhiên, ta có thể giảm độ phức tạp thuật toán xuống còn *O(N)* với *N* là số internal node của T_0 thông qua thuật toán Cost–Complexity Greedy Pruning (xem thêm Kempe & Rosenberg, 2019, p. 49).

3. **Code Scikit Learn: Link (trang 14).**

IV. Tìm hiểu sâu hơn: C4.5 → C5; MARS; Sâu hơn về kỹ thuật pruning; Implementation Scikit Learn; Đánh giá và so sánh các biến thể Decision Tree.

Tài liệu tham khảo:

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

GeeksforGeeks. (2025, June 30). Decision Tree. Sanchhaya Education Private Limited. Retrieved August 16, 2025, from https://www.geeksforgeeks.org/machine-learning/decision-tree/

Kempe, J., & Rosenberg, D. S. (2019, April 9). *Classification and regression trees* [Lecture slides]. DS-GA 1003 / CSCI-GA 2567 Machine Learning, New York University. Retrieved from https://davidrosenberg.github.io/mlcourse/Archive/2019/Lectures/10a.trees.pdf

Machine Learning cơ bản. (2018, January 14). Decision Trees (1): Iterative Dichotomiser 3. Machine Learning cơ bản. Retrieved August 16, 2025, from https://machinelearningcoban.com/2018/01/14/id3/

Morgan, J. (2014). Classification and regression tree analysis (Technical Report No. 1). Health Policy and Management, Boston University School of Public Health. Retrieved from https://www.bu.edu/sph/files/2014/05/MorganCART.pdf

Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: A guide for data scientists. O'Reilly Media.

Nguyễn, T. (2021). Decision Tree algorithm [Webpage]. Machine Learning cho dữ liệu dạng bảng. Retrieved August 16, 2025, from https://machinelearningcoban.com/tabml_book/ch_model/decision_tree.html

Pham Dinh Khanh (2021). Mô hình cây quyết định (decision tree). Deep AI KhanhBlog. Retrieved August 17, 2025, from https://phamdinhkhanh.github.io/deepai-book/ch_ml/DecisionTree.html

Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1), 81–106.

Steinberg, D. (2009). *CART: Classification and regression trees* (Chapter 10). In *The Top Ten Algorithms in Data Mining* (pp. 180–189). Chapman & Hall/CRC.

Wikipedia (2025). Retrieved August 16, 2025, from https://en.wikipedia.org