**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM**

**KHOA KHOA HỌC MÁY TÍNH**



**BÁO CÁO ĐỒ ÁN MÔN HỌC**

DECISION TREE

**Môn học:** IT003 – Cấu trúc dữ liệu và giải thuật

**Thực hiện bởi sinh viên Phan Nhật Tân – 23521405.**

**Thời gian thực hiện:** 16/4/2024 - <Thời gian kết thúc>

Contents

[Chương I. DECISION TREE LÀ GÌ? 3](#_Toc165710150)

[Chương II. THUẬT TOÁN XÂY DỰNG DECISION TREE. 5](#_Toc165710151)

[Chương III. XỬ LÝ DỮ LIỆU CỤ THỂ VỚI DECISION TREE. 12](#_Toc165710152)

[Chương IV. CẢI TIẾN. 14](#_Toc165710153)

[CODE VÀ DATASET 17](#_Toc165710154)

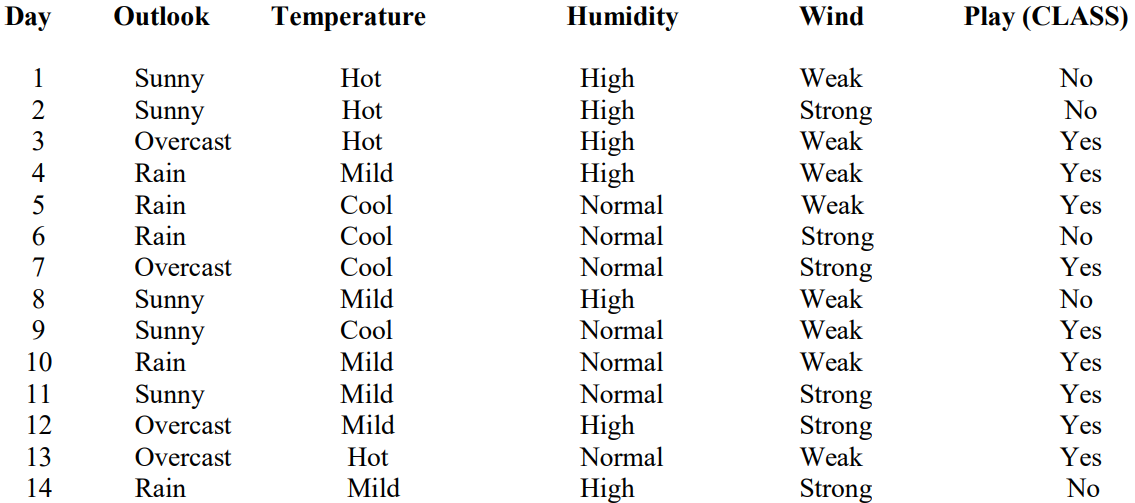
[NGUỒN THAM KHẢO 18](#_Toc165710155)

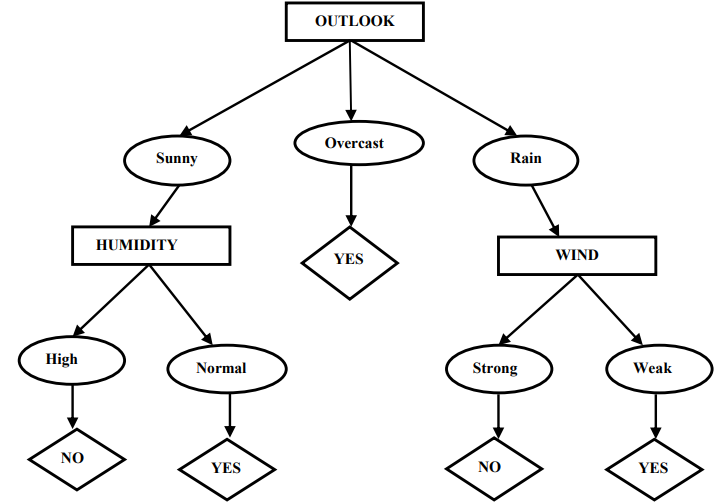
# 

# Chương I. DECISION TREE LÀ GÌ?

* Thuật toán Decision Tree CART (Classification And Regression Tree) có sơ đồ giống như cấu trúc cây để hiển thị các dự đoán là kết quả của một loạt các phân tích dự đoán trên đặc tính.Nó bắt đầu với một nút gốc và kết thúc bằng một quyết định của các nút lá.
* Chúng ta có thể hiểu cây quyết định một cách đơn giản là một đồ thị có hình dạng giống như cây và nó có hướng được sử dụng để cho việc ra quyết định. Ví dụ bạn cần đưa ra quyết định của mình có nên đi chơi hay không khi phụ thuộc vào các yếu tố như thời tiết, nhiệt độ, đổ ẩm, gió…

**VÍ DỤ:**

****



* Decision Tree thường có 3 thành phần:
* Nút gốc: Nút cao nhất thể hiện mục tiêu cuối cùng hoặc quyết định của mình.
* Nhánh: Các nhánh xuất phát từ gốc, sẽ cho các lựa chọn khác nhau hoặc các hướng hành động.
* Nút lá: Được gắn ở cuối các nhánh, thể hiện các kết quả sau mỗi hành động.

# Chương II. THUẬT TOÁN XÂY DỰNG DECISION TREE.

1. **Các chỉ số và tiêu chí:**
2. **Entropy:**

* **Entropy** là một thước đo **độ hỗn loạn (độ tạp nhiễm)** của một nút trong cây quyết định. Nói cách khác, nó cho ta biết mức **độ lộn xộn** của thông tin tại nút đó.
* Cho một phân phối xác suất của một biến rời rạc x có thể nhận n giá trị khác nhau x1 , x2 ,..., xn .  Giả sử rằng xác suất để x nhận các giá trị này là pi=p(x=xi) .Entropy của phân phối này được định nghĩa là:
* Giá trị Entropy **cao nhất** là 1 khi p1 = p2 = 0.5, nghĩa là thông tin tại nút đó **hoàn toàn ngẫu nhiên, không có xu hướng rõ ràng** về lớp nào.
* Giá trị Entropy **thấp nhất** là 0 khi p1 = 1 hoặc p2 = 1, nghĩa là tất cả các mẫu tại nút đó đều thuộc **cùng một lớp.**
* **Lưu ý:** Entropy chỉ được sử dụng cho các bài toán phân loại nhị phân (hai lớp).

1. **Gini Index:**

* Là một cách đo khác để tính **độ tạp nhiễm.**
* Với mỗi node t chứa Nt điểm dữ liệu, tính Gini bằng công thức:

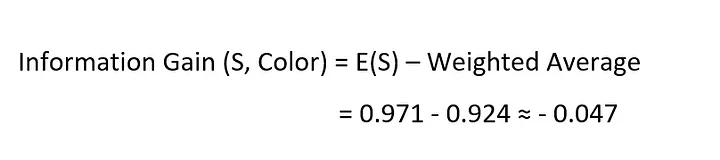
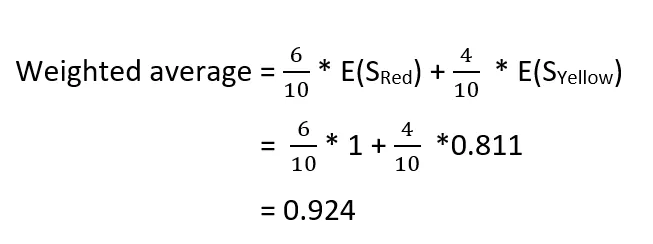
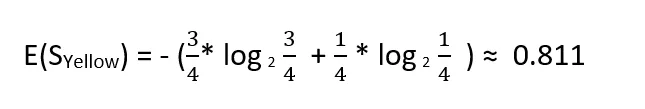
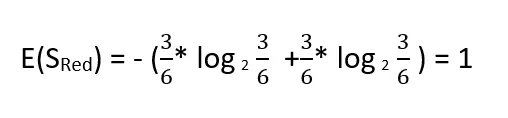
với pi là xác suất của điểm dữ liệu trong node t thuộc lớp i, và c là số lớp.

* Giá trị Chỉ số Gini **cao nhất** là 1 khi tất cả các lớp có tỷ lệ **bằng nhau (hoàn toàn hỗn loạn)**.
* Giá trị Chỉ số Gini **thấp nhất** là 0 khi tất cả các mẫu tại nút đó đều thuộc **cùng một lớp (hoàn toàn tinh khiết).**

1. **Information Gain:**

* **Information Gain** là một thước đo được sử dụng trong cây quyết định để lựa chọn thuộc tính phân chia (splitting attribute) hiệu quả nhất. Nói cách khác, nó giúp xác định thuộc tính nào mang lại nhiều thông tin nhất để **giảm thiểu** độ **tạp nhiễm** của các nút trong cây.
* **Information Gain** sử dụng khái niệm **Entropy** để tính toán mức độ giảm thiểu độ tạp nhiễm khi phân chia một nút theo một thuộc tính cụ thể.
* **Ví dụ:** Tập dữ liệu sau có 10 quan sát thuộc hai lớp YES và NO. Trong đó có 6 quan sát thuộc lớp YES và quan sát thuộc lớp NO.

Màu đỏ có 3 kết quả là YES và 3 kết quả là NO trong khi màu vàng có 3 kết quả là YES và 1 kết quả là NO.



* Ý nghĩa của Information Gain:
* Information Gain cao: Thuộc tính phân chia mang lại nhiều thông tin để giảm thiểu độ tạp nhiễm, do đó là lựa chọn tốt.
* Information Gain thấp: Thuộc tính phân chia không mang lại nhiều thông tin để cải thiện độ tinh khiết, nên cân nhắc lựa chọn khác.

1. **Gain Ratio (Tỷ lệ thông tin):**

* **Gain Ratio** là một thước đo được sử dụng trong cây quyết định để khắc phục hạn chế của **Information Gain.**
* Vấn đề của **Information Gain**: Information Gain có xu hướng ưu tiên các thuộc tính có nhiều giá trị riêng biệt (distinct values). Điều này có thể dẫn đến việc chọn các thuộc tính không thực sự hữu ích cho việc phân loại, chỉ đơn giản là chúng có nhiều nhánh hơn. Do đó giải pháp là dùng **Gain Ratio**.
* **Gain Ratio** "chuẩn hóa" **Information Gain** bằng cách tính toán thêm một thành phần gọi là **Split Info.**
* Ý nghĩa của **Gain Ratio**:
* Gain Ratio cao: Thuộc tính vừa mang lại nhiều thông tin để giảm thiểu độ tạp nhiễm (giống Information Gain), vừa có Split Info thấp (ít nhánh thừa).
* Gain Ratio thấp: Thuộc tính không hiệu quả do Information Gain thấp hoặc Split Info quá cao (nhiều nhánh thừa).

1. **Twoing Criteria:**

* **Twoing Criteria** là một thước đo được sử dụng trong cây quyết định để khắc phục hạn chế của Gini Index trong một số trường hợp.
* **Twoing Criteria** là một **tiêu chuẩn nhị phân** (binary criterion) được sử dụng thay thế cho Gini Index trong trường hợp thuộc tính đích có nhiều lớp.
* Ý tưởng chính của Twoing Criteria:

1. **Chia tất cả các lớp thành hai nhóm:** Nhóm thứ nhất và nhóm thứ hai.
2. **Tính toán Gini Index** cho sự phân chia nhị phân này.
3. **Lặp lại bước 1 và 2** cho tất cả các cách chia lớp có thể thành hai nhóm.
4. **Chọn cách chia nhóm** dẫn đến **Gini Index lớn nhất** - Tức là sự phân chia nhị phân mang lại nhiều thông tin nhất để giảm thiểu độ tạp nhiễm.

1. **Tiêu chí dừng:**

* Tất cả các phiên bản trong tập huấn luyện đều thuộc về một giá trị của y.
* Đã đạt đến độ sâu tối đa của cây.
* Số lượng trường hợp ở nút đầu cuối ít hơn số lượng trường hợp tối thiểu cho các nút cha.
* Nếu nút được tách ra, số trường hợp trong một hoặc nhiều nút con sẽ nhỏ hơn mức tối thiểu số trường hợp cho các nút con.
* Tiêu chí phân chia tốt nhất không được lớn hơn ngưỡng nhất định.

**B. Các thuật toán Decision Tree:**

1. **ID3:**

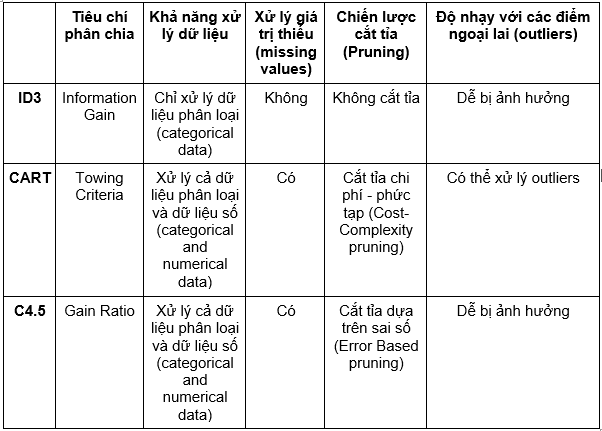
* **Thuật toán ID3 (Iterative Dichotomiser 3)** là một thuật toán xây dựng cây quyết định cơ bản và phổ biến, được phát triển bởi Ross Quinlan vào năm 1983.
* Trong ID3, tổng có trọng số của entropy tại các leaf-node sau khi xây dựng decision tree được coi là hàm mất mát của decision tree đó. Công việc của ID3 là tìm các cách phân chia hợp lý (thứ tự chọn thuộc tính hợp lý) sao cho hàm mất mát cuối cùng đạt giá trị càng nhỏ càng tốt. Việc này đạt được bằng cách chọn ra thuộc tính sao cho nếu dùng thuộc tính đó để phân chia, entropy tại mỗi bước giảm đi một lượng lớn nhất.
* Ưu điểm:
* Quy tắc dự đoán dễ hiểu
* Xây dựng cây nhanh nhất
* Cây quyết định nhỏ gọn
* Chỉ cần kiểm tra đủ số thuộc tính để phân loại toàn bộ dữ liệu
* Bằng cách xác định các nút lá, ID3 cho phép cắt tỉa dữ liệu thử nghiệm, giảm số lần kiểm tra
* Tận dụng toàn bộ tập dữ liệu để xây dựng cây quyết định, đảm bảo tận dụng tối đa thông tin có sẵn
* Nhược điểm:
* Có thể bị overfitting nếu bộ dữ liệu huấn luyện nhỏ
* ID3 chỉ kiểm tra từng thuộc tính một lần để đưa ra quyết định phân loại. Điều này có thể bỏ qua các mối quan hệ phức tạp tiềm ẩn giữa các thuộc tính
* Không xử lý dữ liệu số và thiếu giá trị

1. **CART:**

* **CART (Classification and Regression Trees)** là một thuật toán xây dựng cây quyết định được phát triển bởi Leo Breiman và cộng sự vào năm 1984.
* Đặc điểm của CART:
* **Xây dựng cây nhị phân:** Mỗi nút bên trong của cây CART chỉ có đúng hai nhánh con (binary tree).
* **Tiêu chuẩn Twoing:** Thuật toán sử dụng tiêu chuẩn Twoing để chọn thuộc tính để phân chia dữ liệu tại mỗi nút.
* **Cắt tỉa cây:** CART sử dụng phương pháp cắt tỉa chi phí-phức tạp (cost-complexity pruning) để loại bỏ các nhánh thừa, tránh overfitting.
* **Xử lý chi phí phân loại:** CART có thể tính đến chi phí phân loại khác nhau cho các lớp (class) khi xây dựng cây.
* **Phân bố xác suất trước:** Người dùng có thể cung cấp phân bố xác suất trước cho các lớp.
* **Cây hồi quy:** CART không chỉ xây dựng cây phân loại (classification tree) mà còn có thể xây dựng cây hồi quy (regression tree) để dự đoán các giá trị liên tục.
* Ưu điểm:
* Xử lý cả dữ liệu số và categorical
* **Xác định biến quan trọng:** CART tự động xác định các biến quan trọng nhất để phân loại hoặc dự đoán và loại bỏ các biến không quan trọng.
* Xử lý các điểm ngoại lai (outlier) tương đối tốt.
* Nhược điểm:
* Cây quyết định do CART tạo ra có thể không ổn định. Một thay đổi nhỏ trong dữ liệu huấn luyện (chẳng hạn như loại bỏ một vài mẫu) có thể dẫn đến thay đổi đáng kể trong cấu trúc cây (số lượng nhánh, các biến phân chia,...).
* Chỉ phân chia theo một biến.

1. **C4.5:**

* C4.5 là thuật toán xây dựng cây quyết định được phát triển bởi Ross Quinlan, là một cải tiến quan trọng so với thuật toán ID3.
* Đặc điểm của C4.5:
* Cây quyết định được xây dựng theo chiến lược đi sâu trước (từ gốc đến ngọn).
* C4.5 sử dụng Gain Ratio để chọn thuộc tính phân chia dữ liệu tại mỗi nút. Gain Ratio khắc phục nhược điểm của Information Gain trong ID3, tránh ưu tiên các thuộc tính có nhiều giá trị riêng biệt.
* **Xử lý dữ liệu liên tục:** C4.5 có thể xử lý các thuộc tính liên tục (numerical) bằng cách tạo ngưỡng phân chia dữ liệu thành hai nhánh con.
* **Xử lý dữ liệu thiếu:** C4.5 có thể xử lý các giá trị thuộc tính bị thiếu (missing values) bằng cách bỏ qua các giá trị này khi tính toán Gain Ratio.
* C4.5 cũng sử dụng phương pháp cắt tỉa cây để loại bỏ các nhánh thừa, tránh overfitting.
* Ưu điểm:
* Xử lý cả dữ liệu số và categorical.
* Xử lý các giá trị thuộc tính bị thiếu trong dữ liệu.
* Xử lý dữ liệu nhiễu tốt.
* Nhược điểm:
* Quá trình xây dựng cây đôi khi có thể tạo ra các nhánh trống, làm tăng kích thước và độ phức tạp của cây.
* Dễ bị overfitting.
* So với ID3, C4.5 có thể tốn nhiều tính toán hơn do cách xử lý dữ liệu liên tục và thiếu.



# Chương III. XỬ LÝ DỮ LIỆU CỤ THỂ VỚI DECISION TREE.

**A. Dataset:**

* Bộ dữ liệu được lấy từ link URL dẫn đến tập dữ liệu từ trang UCI mà không cần tải xuống.

|  |
| --- |
| Title                            : Balance Scale Weight & Distance Database  Number of Instances  : 625 (49 balanced, 288 left, 288 right)  Number of Attributes  : 4 (numeric) + class name = 5  **Attribute Information:**  **1. Class Name (Target variable):** 3         L [balance scale tip to the left]         B [balance scale be balanced]         R [balance scale tip to the right]  2. Left-Weight: 5 (1, 2, 3, 4, 5)  3. Left-Distance: 5 (1, 2, 3, 4, 5)  4. Right-Weight: 5 (1, 2, 3, 4, 5)  5. Right-Distance: 5 (1, 2, 3, 4, 5)  **Missing Attribute Values:** None  **Class Distribution:**        1. 46.08 percent are L        2. 07.84 percent are B        3. 46.08 percent are R |

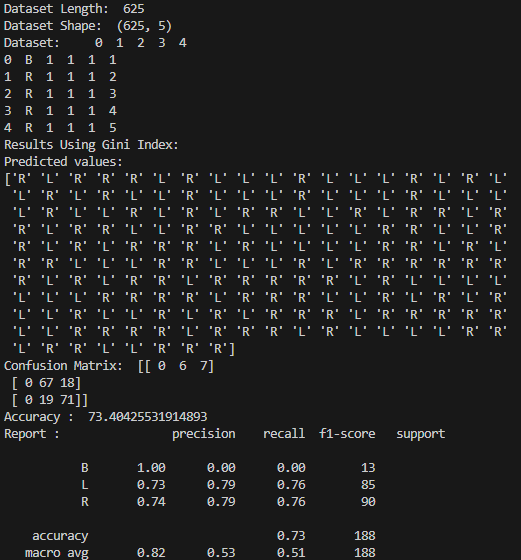
1. **Các thư viện sử dụng:**

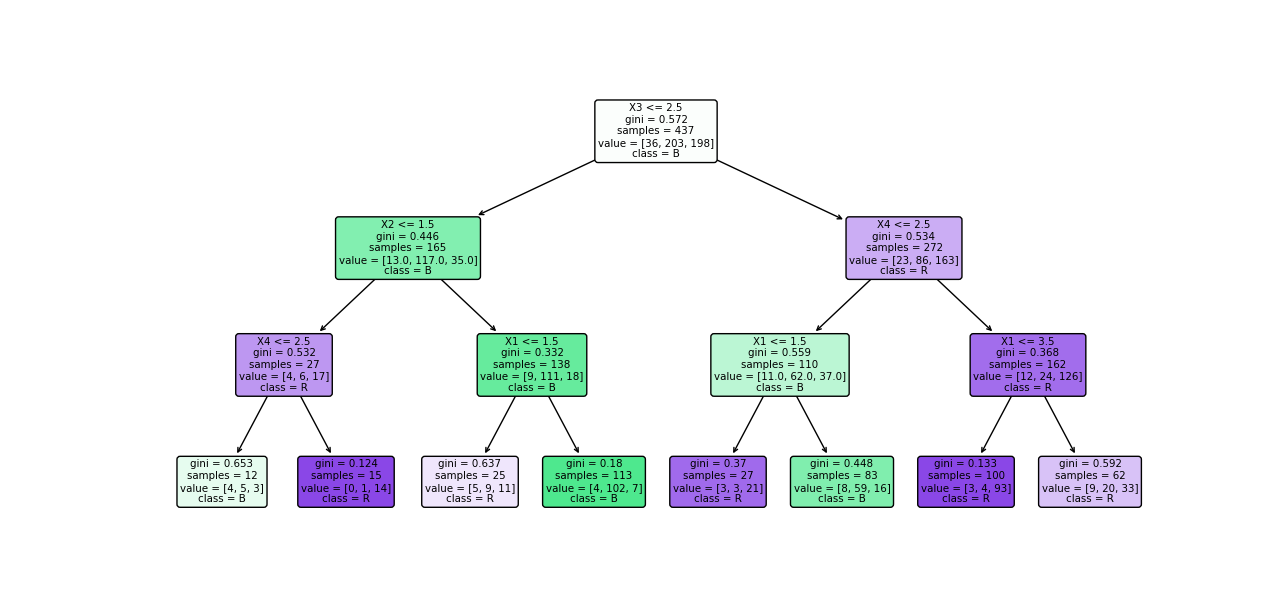
* sklearn: Thư viện sklearn trong Python có nhiều thuật toán học máy
* NumPy: Cung cấp các hàm toán học để tính toán nhanh. Ngoài ra còn giúp đọc dữ liệu trong mảng.
* Pandas: Đọc ghi file. Thao tác với dữ liệu dễ dàng.

1. **Cài đặt Decision Tree với Python:**

[**github.com/nhattan2005/Decision-Tree/blob/main/Code/DecisionTre.py**](http://github.com/nhattan2005/Decision-Tree/blob/main/Code/DecisionTree.py)

Output:

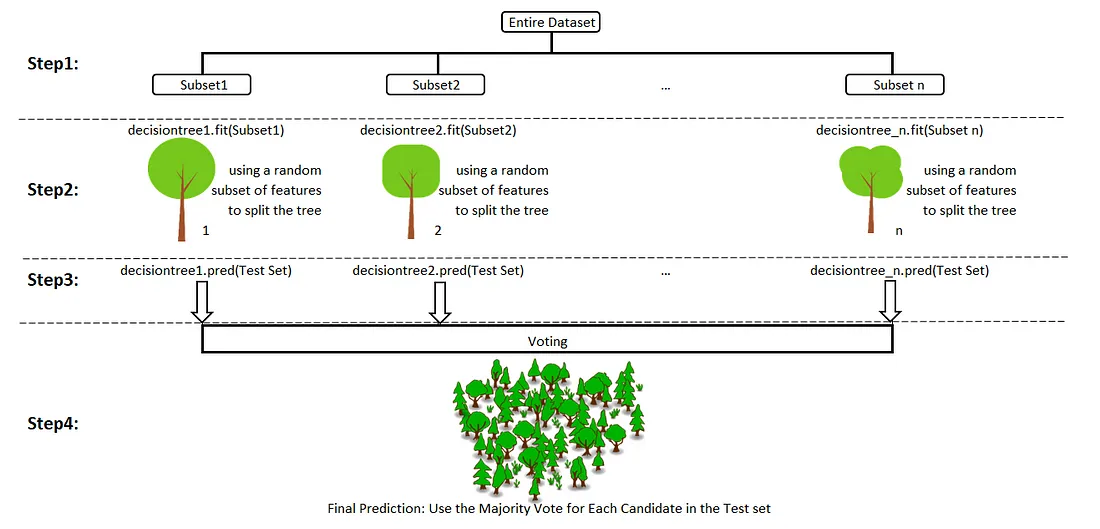




# Chương IV. CẢI TIẾN.

1. **Random Forest:**

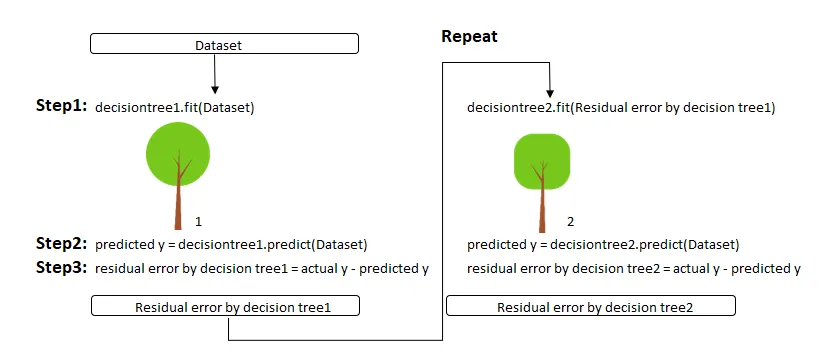
* Random Forest là một **mô hình tổng hợp (ensemble model)** sử dụng phương pháp **gộp túi (bagging)** để kết hợp nhiều Decision Tree yếu thành một mô hình mạnh hơn. Hãy tưởng tượng như một khu rừng với nhiều cây riêng lẻ, mỗi Decision Tree đưa ra một dự đoán, và Random Forest sẽ đưa ra dự đoán cuối cùng dựa trên "biểu quyết đa số" của các cây đó.



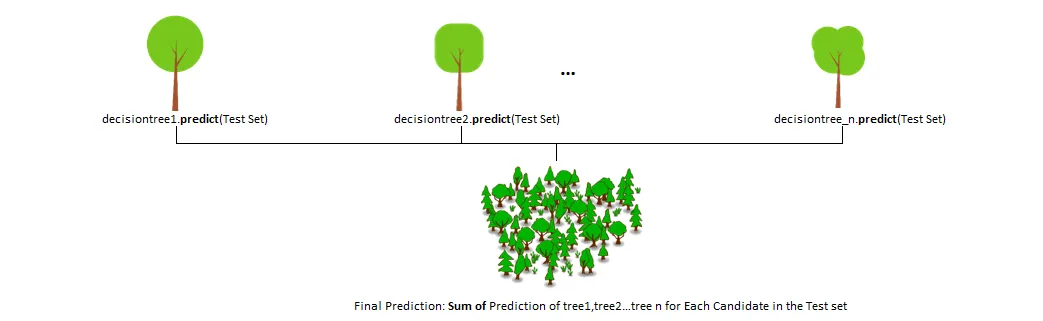
* **Các bước thực hiện:**
* Chọn n tập con (subset) lấy ngẫu nhiên từ tập huấn luyện.
* **Huấn luyện n Decision Tree:**
  + Mỗi subset ngẫu nhiên được sử dụng để huấn luyện 1 Decision Tree.
  + Để tăng thêm tính đa dạng, khi xây dựng mỗi Decision Tree, Random Forest chỉ sử dụng một tập hợp con ngẫu nhiên các thuộc tính (features) của dữ liệu. Ví dụ, nếu bạn có tổng cộng 10 thuộc tính, Random Forest có thể ngẫu nhiên chọn 5 trong số đó để phân chia tại mỗi nút của Decision Tree. Điều này giúp ngăn các Decision Tree quá giống nhau và phụ thuộc vào cùng một số thuộc tính.
* Mỗi cây riêng lẻ dự đoán kết quả trong test set một cách độc lập.
* **Đưa ra kết quả cuối cùng:** Đối với mỗi ứng cử viên trong tập kiểm tra, Random Forest sử dụng lớp (ví dụ: mèo hoặc chó) có đa số phiếu bầu làm dự đoán cuối cùng của ứng cử viên này.

1. **Gradient Boosting:**

* Giống như Random Forest, Gradient Boosting cũng là một mô hình tổng hợp (ensemble model) được sử dụng để cải thiện độ chính xác của mô hình học máy. Nhưng điểm khác biệt chính giữa Gradient Boosting và Random Forest là cách thức học hỏi.
* Random Forest tập trung vào tính đa dạng bằng cách xây dựng các Decision Tree độc lập. Ngược lại, Gradient Boosting tập trung học hỏi từ sai sót của các mô hình trước đó.



* Các bước thực hiện:
* Xây dựng Decision Tree đầu tiên.
* Dự đoán trên cây vừa mới huấn luyện.
* Tính toán sai sót (residual).
* Cây tiếp theo học hỏi từ sai sót (cho đến khi đạt đủ số cây chúng ta đặt ra để đào tạo): Gradient Boosting sẽ xây dựng một Decision Tree thứ hai chỉ tập trung học hỏi từ các residual error này. Nói cách khác, Cây Quyết Định thứ hai cố gắng "sửa chữa" những sai lầm của Decision Tree thứ nhất.
* Cuối cùng, các dự đoán của tất cả các Decision Tree được cộng dồn để tạo thành dự đoán cuối cùng.



1. **Code:**

* Phần code cài đặt ở link: <https://github.com/nhattan2005/Decision-Tree/blob/main/Code/RandomForest_GradientBoosting.py>

1. **Độ chính xác:**

* Độ chính xác của Decision Tree thông thường: 0.7340425531
* Độ chính xác của Random Forest: 0.851063829787234
* Độ chính xác của Gradient Boosting: 0.8617021276595744
* Độ chính xác của Random Forest cao hơn Decision Tree 15.70%
* Độ chính xác của Gradient Boosting cao hơn Decision Tree 17.76%
* Ngoài ra còn có các tiêu chí đánh giá khác bằng các metrics như Confusion Matrix, Precision, Recall, F1 score được show trong output của code.

# CODE VÀ DATASET

1. Code và bộ Dataset được lưu trữ trong github:

<https://github.com/nhattan2005/Decision-Tree/tree/main>

1. Link Dataset:

[archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data](https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data)

# NGUỒN THAM KHẢO

1. [Python | Decision tree implementation - GeeksforGeeks](https://www.geeksforgeeks.org/decision-tree-implementation-python/).
2. [Basic Ensemble Learning (Random Forest, AdaBoost, Gradient Boosting)- Step by Step Explained | by Lujing Chen | Towards Data Science](https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725).
3. Singh, S., & Giri, M. (2014). Comparative Study Id3, Cart And C4.5 Decision Tree Algorithm: A Survey. International Journal of Advanced Information Science and Technology (IJAIST), 3(7), 47-52.