

Random Forest

Hồ Nguyễn Phú (tham khảo AI VN và một số nguồn khác)

03 tháng 09, 2025

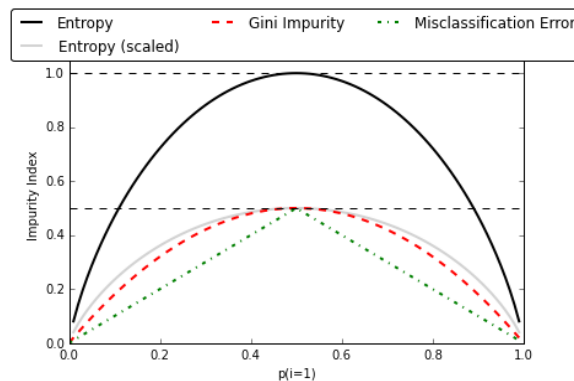
I. Giới thiệu:

1. Ôn lại về Decision Tree:

Ôn lại Decision Tree: Slide 1–7 của buổi thứ Ba, tuần 1, module 4 (AI VN) và tài liệu Decision Tree ([Hồ Nguyễn Phú](#)). Tóm tắt ý chính:

a. Gini & entropy:

Entropy cùng Information Gain (ID3) và Gini (CART) và là các hàm số dùng để phân tích độ tinh khiết (purity) của một nhánh cây trong giải thuật Decision Tree.



Hình 1. So sánh các impurity criteria ([Github](#)).

Ý tưởng chính của các chỉ số này là:

“**Với Entropy**, 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 [và Gini] 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).”

[Với Gini]: **(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.

b. Thuật toán:

```
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 2. Thuật toán CART (đơn giản hóa) được minh họa bởi Steinberg (2001).

Theo đó,

- + Bước 1: Chọn ra thuộc tính khi split đem lại cải thiện nhất về chỉ số purity.
- + Bước 2: Split dựa trên thuộc tính đó, tạo thành các nhánh cây con.
- + Bước 3: Kiểm tra điều kiện “số lượng mẫu trong node” và “độ sâu” để quyết định split tiếp hay không. Nếu không nhảy đến bước 5.
- + Bước 4: Quay lại bước 1 (đệ quy tại mỗi nhánh con).
- + Bước 5: Đặt node hiện tại làm leaf node – có thể đưa ra dự đoán.

c. Pruning:

Decision Tree dễ bị overfit vào data, do đó để xử lý ta thường dùng phương pháp tỉa nhánh để cắt tỉa mô hình. Do vậy, ta có học qua khái niệm Tree Complexity Penalty – tính bởi công thức hình 3. Bất tiện là, chỉ số alpha α trong công thức trên cần thực nghiệm và tuning khá nhiều (admin).

Gọi:

- T_0 : cây ban đầu ("big tree")
- $\hat{R}(T)$: "empirical risk" của cây T (ví dụ như Squared Error, entropy)
- Do quá trình phân nhánh, mọi cây con T có $\hat{R}(T)$ 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 :

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

Hình 3. Công thức tính Tree Complexity Penalty.

d. Hạn chế của Decision Tree: Dễ overfit vào data huấn luyện (các node lá có rất ít mẫu, độ sâu rất lớn → trên lý thuyết có thể đạt đến mỗi node lá một mẫu nếu không đặt số mẫu ít nhất hoặc độ sâu tối đa).

2. Ensemble Learning:

Các phương pháp ensemble được sinh ra để làm việc với bias-variance trade-off. Trong đó, ta có thuật ngữ mô hình mạnh (strong learner – dự đoán với độ chính xác rất cao, đến mức khó giải thích) và mô hình yếu (weak learner – dự đoán chỉ tốt hơn đoán đại, "random guessing," một chút). Mô hình mạnh thường được tạo ra thông qua sự kết hợp của các mô hình yếu lại với nhau – ensemble learning.

Các phương pháp ensemble learning: Bagging, Boosting, Stacking. Bagging có thể giảm Variance, còn Boosting và Stacking giảm Bias (và đôi khi cả Variance).

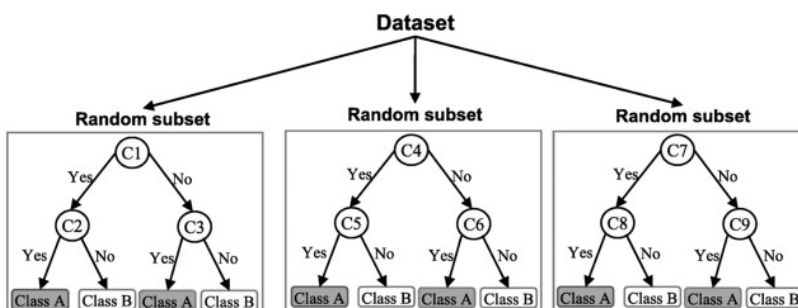
3. Về Random Forest:

Random Forest được phát triển bởi Breiman, L. (2001). Trong đó "Random forest là một classifier bao gồm tập hợp k classifier có cấu trúc cây, với mỗi cây bầu chọn [vote]" để đưa ra kết quả cuối cùng (phân loại hoặc hồi quy).

Theo [AI Candy](#), "random" là ngẫu nhiên trong **chọn mẫu** và **chọn feature**.

Random Forest nôm na có thể coi như là một tập hợp các Decision Tree. Trong nhóm các thuật toán Ensemble Learning, Random Forest thuộc về thuật toán Bagging.

II. Vắn tắt về thuật toán:



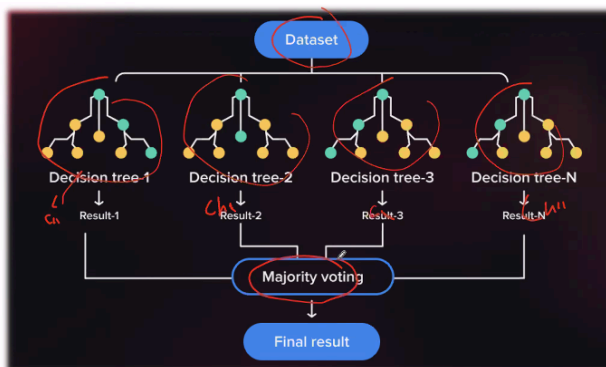
Hình 4. Ví dụ về các cây con trong Random Forest.

1. Ý tưởng và động lực:

- Decision Tree là một thuật toán mạnh (built-in feature selection, có cơ chế xử lý dữ liệu thiếu, vv.) nhưng **dễ bị overfit**.
- Các thuật toán ensemble learning thời điểm đó, đặc biệt là với các tree classifier.

2. Cơ chế:

- Mô hình chọn ngẫu nhiên một số mẫu từ dataset đầy đủ, sau đó tạo thành các bootstrap dataset (xem thêm phần III).
- Các cây Decision Tree con (CART) sẽ được huấn luyện trên từng bootstrap dataset.
- Đưa ra dự đoán dựa trên bầu chọn của các cây con (lựa chọn nhiều nhất, hay argmax, với phân loại; giá trị trung bình với hồi quy).

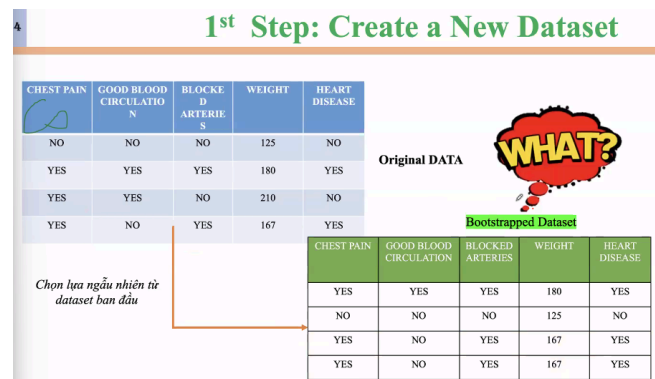


Hình 5. Minh họa về majority voting của Random Forest (AI VN, trang 29).

III. Bootstrap dataset:

1. Phương pháp train mỗi cây trên một tập Bootstrap dataset:

Từ dữ liệu ban đầu, ta chọn ngẫu nhiên một tập data. Ví dụ với n mẫu, ta random trong khoảng từ **1 đến n** để lấy các giá trị để có được tập data mới. Vấn đề: Overfit do duplicate (họ sẽ giải quyết trong bước kế tiếp)? Đây được gọi là Bootstrapped Dataset (random sampling with replacement – khi các dữ liệu được lấy ra để tạo dataset mới, nhưng trong tập dữ liệu gốc nó vẫn còn đó, tức được replace).



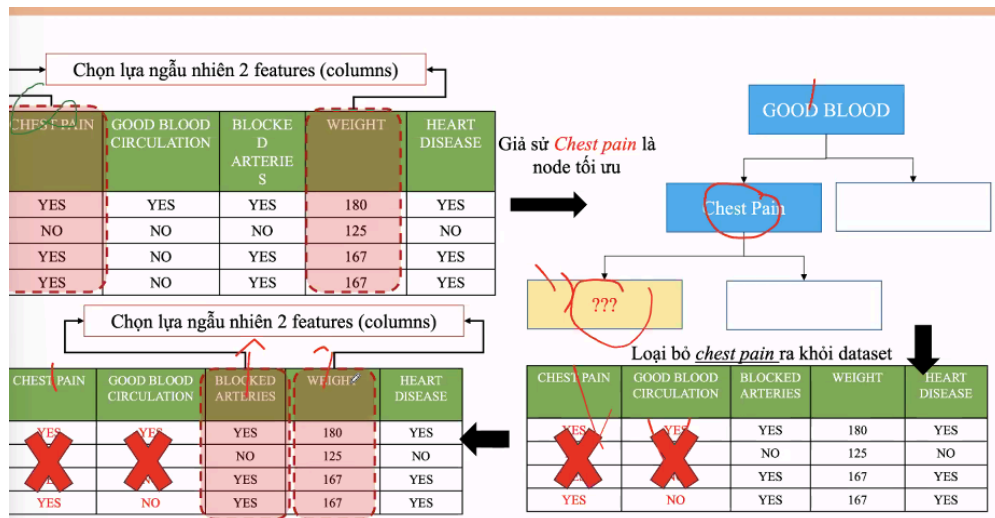
Hình 6. Bước 1: Tạo ra một Bootstrapped Dataset (có size).

Họ sẽ không chọn tất cả các giá trị (attribute) để tính Gini, mà chỉ tính Gini trên *điều kiện định nghĩa trước* (ví dụ, chỉ có 2 thuộc tính Good Blood Circulation và Blocked Arteries). Để giải thích, **nếu ta chọn tất cả thuộc tính thay vì chỉ 2** (một siêu tham số thực nghiệm), **ta không khác gì đang tạo một decision tree với toàn bộ dữ liệu**. Trong bài báo gốc: “The simplest random forest [...] is formed by selecting at random [...] input variables to split on.”



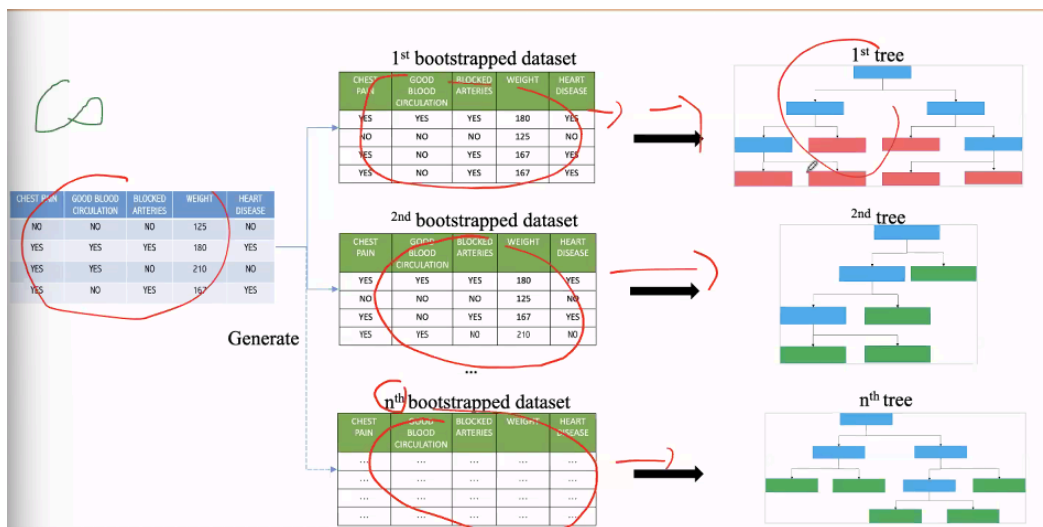
Hình 7. Bước 2: Chọn ra 2 feature ngẫu nhiên (Good Blood Circulation và Blocked Arteries) và lấy feature có Gini thấp hơn (Good Blood Circulation).

Bước tiếp theo, ta chọn thuộc tính tốt nhất làm node tiếp theo (ví dụ: Chest Pain) và loại bỏ nó đi trong quá trình tìm kiếm ngẫu nhiên 2 thuộc tính lần tiếp theo.



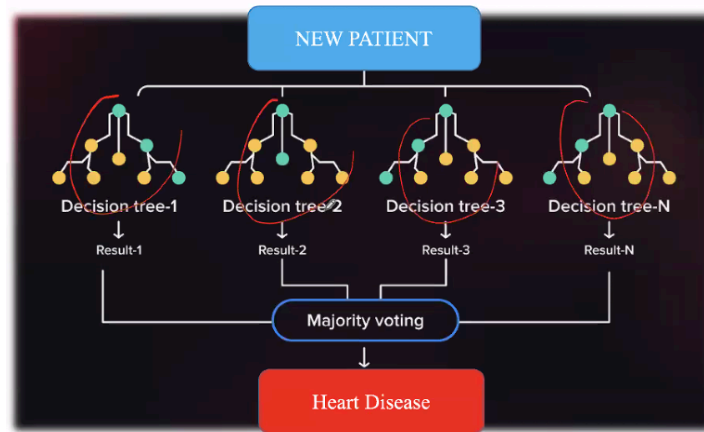
Hình 6. Bước 3: loại bỏ tiêu chí đã chọn trước đó và tiếp tục random.

Sau đó, ta sẽ huấn luyện N cây trên N bootstrapped dataset. Điều này sẽ giúp cho mỗi cây nhìn được một phần ngẫu nhiên của dữ liệu, tránh overfitting. **Khác với Decision Tree (CART) cần thực hiện pruning**, Random Forest không prune và giữ nguyên cây; điều này do “quy luật Số lớn cho thấy chúng sẽ luôn hội tụ” và overfitting không phải vấn đề.



Hình 7. Bước 4: Tạo ra N cây từ N bootstrapped dataset (mỗi cây sẽ giải quyết một phần dữ liệu khác nhau).

N cây này sẽ bầu chọn ra kết quả đầu ra: phân loại hoặc hồi quy.



Hình 8. Bước 5: Vote kết quả với N cây.

→ **RANDOM FOREST**

2. Tại sao ta lại sử dụng bootstrap dataset:

Lí do sử dụng Bootstrap dataset và Out-of-bag được tác giả nêu: giúp tăng độ chính xác trong bối cảnh feature selection ngẫu nhiên; và tăng tính tổng quát hóa của mô hình.

IV. Dữ liệu bị trùng lặp – và Out-of-Bag Error:



Hình 9. Dữ liệu bị lặp khi tạo data test (!)

Tại sao họ lại dùng dữ liệu bị lặp như vậy? Vì nhận định rằng có một phần dữ liệu không được gặp đến trong bootstrapped dataset. Ta có thể dùng chúng làm dữ liệu test (**out-of-bag error** – dữ liệu nằm ngoài bootstrapped dataset, dùng để đo lường độ chính xác của giải thuật Random Forest).

V. Missing Data:

Do sự cố kỹ thuật, một số vị trí trong dữ liệu có thể bị missing. Với dữ liệu bị missing, một giải pháp là thực hiện impute giá trị (như trung bình các dữ liệu – không hiệu quả với các giá trị categorical).

Với Random Forest ta sẽ gán giá trị xuất hiện là Training, còn giá trị thiếu là Predict. **Các giá trị thiếu sẽ được dự đoán (Guessing the Data) thông qua giá trị trung bình:**

- + Giá trị number (continuous): tính trung bình các giá trị khác trong cột.
- + Giá trị text (categorical): giá trị xác suất cao nhất.

Sau đó giá trị dự đoán sẽ được tinh chỉnh:

- + Thuật ngữ: Proximity Matrix – với M mẫu trong bootstrapped data tạo thành ma trận $M \times M$.
- + Ban đầu xây dựng các cây thông qua các giá trị không bị thiếu (ngẫu nhiên các giá trị, nhưng không bao gồm cột có giá trị thiếu).
- + Dùng Proximity Matrix để nội suy dữ liệu của data bị thiếu – ta xét đến sự phụ thuộc / sự tương tự của data bị thiếu với các mẫu data khác (ở đây tức là số lần các cây hiện có sẽ vote cho 2 mẫu có cùng class – tức sự tương đồng). **Về số cây cần có trước khi thực hiện impute, ta đặt hyperparameter (Appendix, câu hỏi 1).**

	1	2	3	4
1		2	1	1
2	2		1	1
3	1	1		8
4	1	1	8	

Hình 10. Ma trận tương quan (Proximity Matrix) của từng mẫu theo cặp (pairwise).

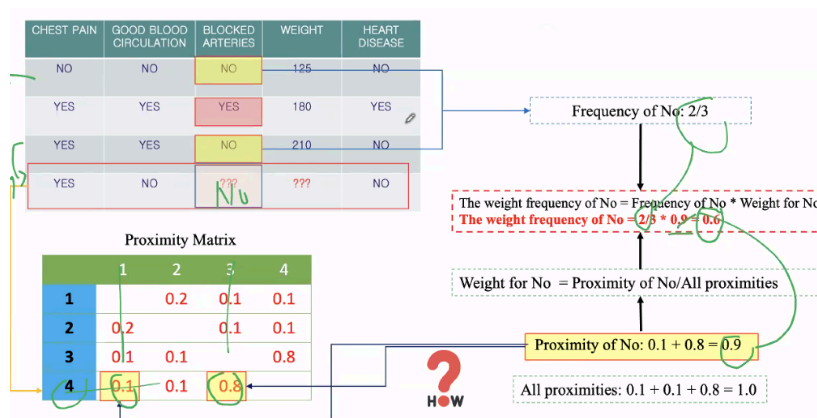
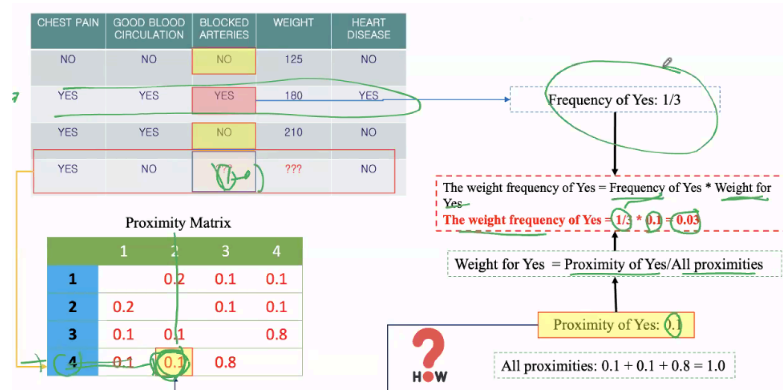
- + Khi nội suy data bị thiếu, ta sẽ mong muốn đặt trọng số cho các sample “liên quan” hơn đến data bị thiếu (thông qua nhãn dự đoán bởi các cây sẵn có) sao cho chúng cao hơn so với các sample còn lại (hình 11).

CHEST PAIN	GOOD BLOOD CIRCULATION	BLOCKED ARTERIES	WEIGHT	HEART DISEASE
NO	NO	NO	125	NO
YES	YES	YES	180	YES
YES	YES	NO	210	NO
YES	NO	NO	167.5	NO

Mỗi dòng thể h

Hình 11. Thực hiện dự đoán dữ liệu liên tục bị thiếu (chỗ này em tính ra 172).

Ví dụ, ở trường hợp bên dưới, ta xử lý trọng số của các sample 1, 2, 3 đối với sample 4:



Hình 12 và 13. Thực hiện dự đoán dữ liệu categorical bị thiếu.

VI. Code:

Ta dùng API của thư viện sklearn là chủ yếu. Sau đó, có thể thực hiện lưu mô hình qua thư viện joblib.

1. Về class RandomForestClassifier:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *,
criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None,
random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0,
max_samples=None, monotonic_cst=None)
```

[\[source\]](#)

Hình 14. Class RandomForestClassifier của sklearn.

a. Tham số:

Qua đây, ta có thể thấy một số tham số đáng chú ý:

- **n_estimators [int]:** số lượng các cây con
- **criterion [str]:** chỉ số purity – bao gồm {"gini", "entropy", "log_loss"}
- **max_depth [int]:** độ sâu tối đa của cây
- **min_samples_split [int]:** số lượng mẫu tối thiểu có thể split
- **min_samples_leaf [int]:** số lượng mẫu tối thiểu mỗi leaf node
- **bootstrap [bool]:** liệu có thực hiện bootstrap không
- **random_state [int]:** đảm bảo reproducibility

b. Inference:

Các bước bao gồm:

- (1) tạo một instance
- (2) fit instance đó trên dữ liệu
- (3) inference thông qua phương thức predict()

2. **Init và train mô hình:** Như đoạn dưới (ví dụ hình 15).

3. **Lưu mô hình và inference:** Thư viện joblib (ví dụ hình 15).

4. Ví dụ:

```
import joblib

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt


# Read data

df = pd.read_csv("StressLevelDataset.csv")

df.head()


# Lấy cột stress_level làm output và Train/Test split

X = df.drop(columns=["stress_level"])

y = df["stress_level"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Tạo một instance và fit instance đó trên data

random_forest_model = RandomForestClassifier(n_estimators=250)

random_forest_model.fit(X_train, y_train)


# Inference trên tập test

y_pred = random_forest_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))

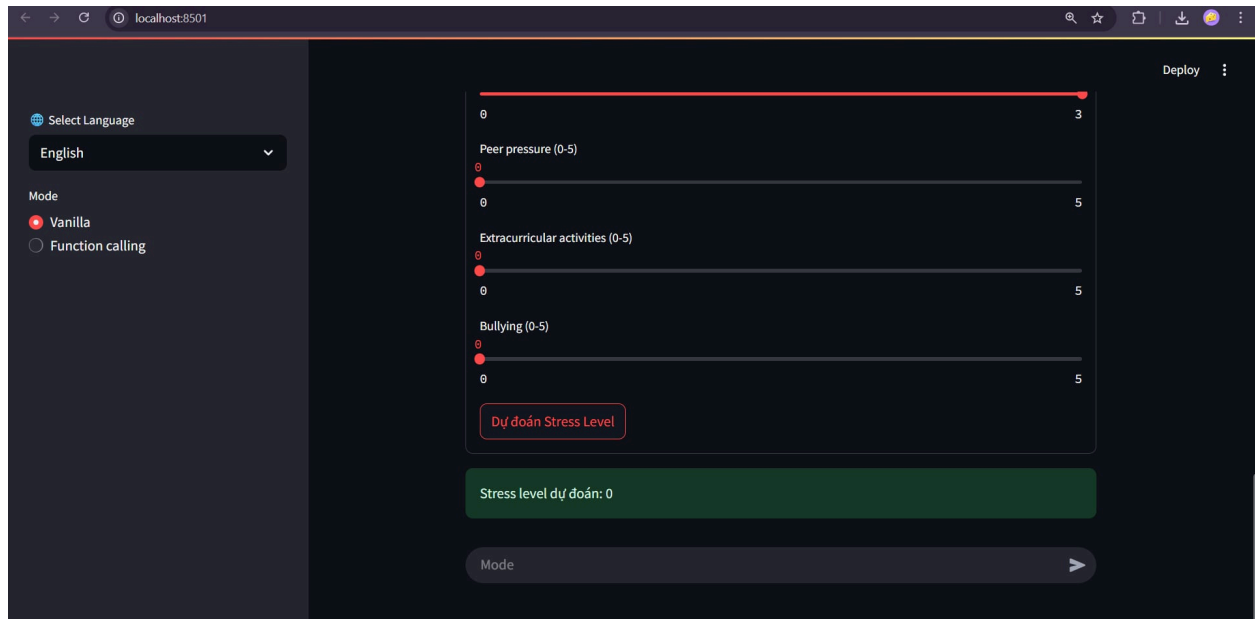
print("\nClassification Report:\n", classification_report(y_test, y_pred))


# Lưu mô hình

joblib.dump(random_forest_model, "random_forest_model.pkl")
```

Hình 16. Code mẫu huấn luyện mô hình.

5. Project idea:



The screenshot shows a web application running on localhost:8501. On the left sidebar, there is a 'Select Language' dropdown set to 'English' and a 'Mode' section with two radio buttons: 'Vanilla' (selected) and 'Function calling'. The main area contains four horizontal sliders for input factors: 'Peer pressure (0-5)' with a value of 3, 'Extracurricular activities (0-5)' with a value of 5, 'Bullying (0-5)' with a value of 5, and an unlabeled slider at the bottom with a value of 0. Below these sliders is a red button labeled 'Dự đoán Stress Level'. At the bottom of the main area, a green bar displays the output: 'Stress level dự đoán: 0'. A 'Deploy' button is visible in the top right corner.

Hình 15. Implement và kết hợp form khảo sát với Chatbot counselling (*tương lai*).

Ý tưởng (chưa implement sâu – demo) ở hình trên:

- Streamlit: back-end và front-end.
- MongoDB free database? SQLite?
- Mô hình Random Forest của Scikit Learn, inference thông qua file “.pkl” như class ban đầu – dùng hàm `joblib.load()`.
- LLM: Gemma 3 12B (test thử, khá mạnh về mảng tâm lý) kết hợp function calling của thư viện `llama_cpp_agent` và RAG của `llama_index`.

Code huấn luyện mô hình gốc được đưa ra bên dưới.