AI VIET NAM
@aivietnam.edu.vn

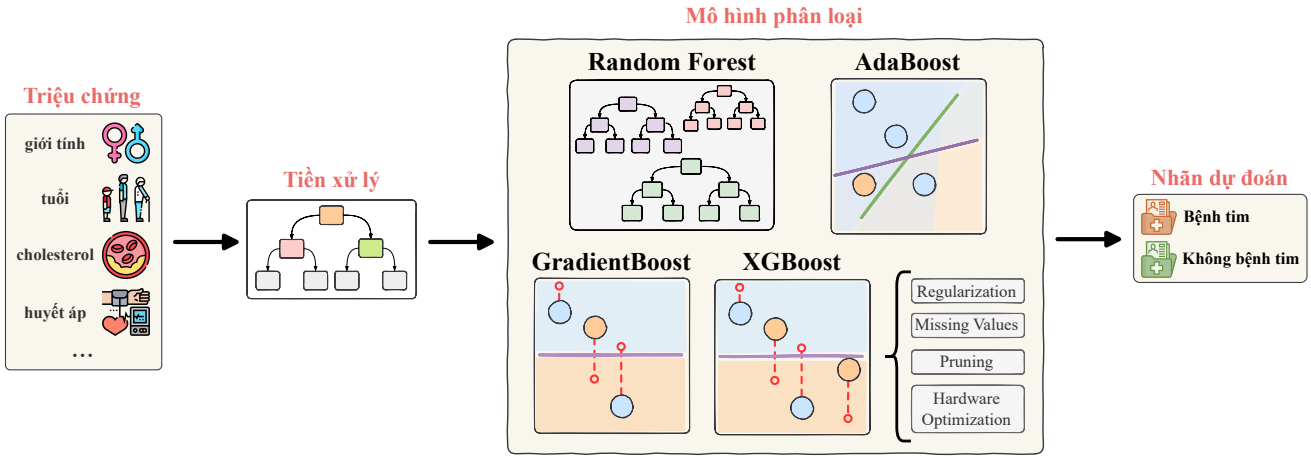
AI VIET NAM – AI COURSE 2025

Project: Heart Disease Diagnosis Part 2

Dương Đình Thắng Vũ Yến Linh Nguyễn Anh Khôi
Dương Trường Bình Đinh Quang Vinh

I. Giới thiệu

Ở phần trước, chúng ta đã đặt những "viên gạch nền" cho bài toán dự đoán bệnh tim: hiểu cấu trúc dữ liệu Cleveland, chuẩn hóa quy trình tiền xử lý, áp dụng kỹ thuật tạo đặc trưng và xây dựng các mô hình Machine Learning (ML) cơ bản như Naive Bayes, K-Nearest Neighbors, Decision Tree, thậm chí thử nghiệm một biến thể ensemble cơ bản. Kết quả cho thấy các mô hình truyền thống, khi được huấn luyện bằng dữ liệu đã xử lý cẩn thận, có thể đạt hiệu năng đáng khích lệ và giúp ta đọc vị bài toán khá hiệu quả.



Hình 1: Pipeline Project Cleveland Heart Disease Diagnosis.

Bước sang phần 2 này, chúng ta tiếp tục nâng cấp cho hệ thống bằng những kỹ thuật ensemble nâng cao với những phương pháp vốn rất mạnh trên dữ liệu bảng:

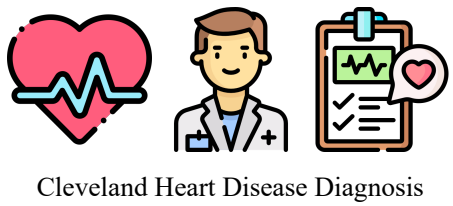
- Random Forest đại diện cho họ Bagging, kết hợp nhiều cây độc lập để giảm phương sai và tăng ổn định.
- AdaBoost và Gradient Boosting đại diện cho họ Boosting, nơi các mô hình liên tiếp "sửa sai" lẫn nhau.
- XGBoost hiện thực Boosting một cách tối ưu hơn với nhiều cải tiến về regularization và hiệu năng tính toán.

Mục lục

Trong toàn bộ dự án này, mọi điều kiện thực nghiệm được giữ nguyên, nhằm tái sử dụng các tập dữ liệu train/val/test và hai cấu hình dữ liệu gồm bản gốc và bản đã xử lý đặc trưng (feature engineering). Chúng ta cũng sử dụng bộ chỉ số quen thuộc như Accuracy, Precision, Recall, F1. Sự thay đổi chính sẽ tập trung vào việc xây dựng một bộ dữ liệu mới bằng mô hình Decision Tree; và huấn luyện các bộ dữ liệu này trên các mô hình Ensemble nâng cao. Chương tiếp theo sẽ trình bày cách triển khai thực nghiệm trong dự án này.

II. Cài đặt chương trình

II.1. Bộ dữ liệu



Hình 2: Bộ dữ liệu dự đoán bệnh tim Cleveland Heart Disease Diagnosis.

Để đảm bảo tính nhất quán và so sánh hiệu quả giữa các mô hình, dự án này tiếp tục sử dụng bộ dữ liệu **Cleveland Heart Disease** [1] đã được giới thiệu chi tiết ở dự án trước. Bộ dữ liệu bao gồm thông tin y tế của **303 bệnh nhân**, với 13 đặc trưng đầu vào (như tuổi, giới tính, cholesterol, v.v.) và một biến mục tiêu để chẩn đoán bệnh nhân có mắc bệnh tim hay không. Bảng dưới đây tóm tắt lại ý nghĩa của từng đặc trưng.

Bảng 1: Một số mẫu dữ liệu từ bộ Cleveland Heart Disease.

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	1
67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

Bảng mô tả đặc trưng bộ dữ liệu Cleveland Heart Disease

Đặc trưng	Mô tả và giá trị mã hóa
`age`	**Tuổi của bệnh nhân** (năm).
`sex`	**Giới tính** (1 = nam, 0 = nữ).
`cp`	**Loại đau ngực**. (1 = đau thắt ngực điển hình, 2 = không điển hình, 3 = đau không do tim, 4 = không triệu chứng)
`trestbps`	**Huyết áp tâm thu lúc nghỉ** (mmHg).
`chol`	**Nồng độ cholesterol huyết thanh** (mg/dL).
`fbs`	**Đường huyết lúc đói > 120 mg/dL**. (1 = đúng, 0 = sai)
`restecg`	**Điện tâm đồ lúc nghỉ**. (0 = bình thường, 1 = bất thường ST-T, 2 = phì đại thất trái)
`thalach`	**Nhịp tim tối đa đạt được** (lần/phút).
`exang`	**Đau ngực khi gắng sức**. (1 = có, 0 = không)
`oldpeak`	**Mức độ trầm ST do gắng sức so với nghỉ**.
`slope`	**Độ dốc đoạn ST**. (1 = dốc lên, 2 = bằng phẳng, 3 = dốc xuống)
`ca`	**Số mạch máu chính được nhuộm màu** (0–3).
`thal`	**Thalassemia**. (3 = bình thường, 6 = tổn thương cố định, 7 = tổn thương có thể đảo ngược)
`num`	**Nhãn mục tiêu**. (0 = không bệnh, 1–4 = có bệnh)

Các giá trị rời rạc được mã hóa trong ngoặc để phục vụ huấn luyện mô hình.

II.2. Cải thiện đặc trưng

Trong phần này, chúng ta thực hiện kỹ thuật xử lý đặc trưng (Feature Engineering - FE) tương tự như dự án trước để nhận được 2 bộ dữ liệu gồm: bộ gốc và bộ FE (cả 2 tập đều có 13 cột/đặc trưng). Sau đó, sử dụng mô hình Decision Tree (DT) để tìm các đặc trưng quan trọng và tạo bộ dữ liệu mới với $K = 10$ đặc trưng quan trọng nhất. Cuối cùng, ta sẽ có tổng cộng 4 bộ dữ liệu: bộ gốc, bộ FE, bộ gốc áp dụng DT, bộ FE áp dụng DT.

Công việc trước tiên là tải các thư viện và thiết lập giá trị ngẫu nhiên, tạo nền tảng vững

chắc cho quá trình xử lý.

Tải các thư viện cần thiết

```
import os
import json
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from pathlib import Path
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_selection import mutual_info_classif
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler

os.environ['PYTHONHASHSEED'] = '42'
np.random.seed(42)
random.seed(42)
```

Sau đó, tải tập dữ liệu Cleveland được lưu trữ trên Google Drive.

Tải bộ dữ liệu gốc (chưa được xử lý)

```
# https://drive.google.com/file/d/16HPyuXWXPptt5g3xvS_kR_wXAfjpR1Ju/view?usp=sharing
!gdown 16HPyuXWXPptt5g3xvS_kR_wXAfjpR1Ju
```

Tiếp đến, tập dữ liệu được đọc và cấu trúc hóa với các cột được đặt tên, dữ liệu số được chuyển đổi, các giá trị thiếu được xác định và cột mục tiêu dự đoán được mã hóa nhị phân để sẵn sàng cho phân tích.

Đọc và cấu trúc hóa tập dữ liệu

```
DATA_PATH = 'cleveland.csv'
COLUMNS = ['age','sex','cp','trestbps','chol','fbs','restecg',
           'thalach','exang','oldpeak','slope','ca','thal','target']

numeric_cols = ['age','trestbps','chol','thalach','oldpeak']
categorical_cols = ['sex','cp','fbs','restecg','exang','slope','ca','thal']

K_features = 10
raw = pd.read_csv(DATA_PATH, header=None)
raw.columns = COLUMNS

for c in ['age','trestbps','chol','thalach','oldpeak','ca','thal']:
```

```
    raw[c] = pd.to_numeric(raw[c], errors='coerce')

raw['target'] = (raw['target'] > 0).astype(int)
print('Shape:', raw.shape)
display(raw.head())
display(raw.isna().sum())
```

Tiến hành chia dữ liệu thành các tập train, validation (val) và test.

Chia tập dữ liệu

```
TARGET = 'target'
raw_feature_cols = [c for c in raw.columns if c != TARGET]
X_all = raw[raw_feature_cols]
y_all = raw[TARGET]

X_train, X_temp, y_train, y_temp = train_test_split(
    X_all, y_all, test_size=0.2, stratify=y_all, random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42
)
```

Một quy trình pipeline được xây dựng để tự động hóa việc tiền xử lý: xử lý các giá trị thiếu và chuẩn hóa dữ liệu, để đảm bảo tính nhất quán giữa các tập. Các tập dữ liệu đã được xử lý này sau đó được lưu thành các tệp CSV riêng biệt, tương ứng với train/val/test.

Xây dựng bộ dữ liệu gốc (đã được xử lý)

```
cat_proc = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('scaler', MinMaxScaler())
])
num_proc = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

preprocess = ColumnTransformer([
    ('num', num_proc, numeric_cols),
    ('cat', cat_proc, categorical_cols),
])
raw_pipeline = Pipeline([
    ('preprocess', preprocess),
])

X_raw_train = raw_pipeline.fit_transform(X_train, y_train)
X_raw_val = raw_pipeline.transform(X_val)
X_raw_test = raw_pipeline.transform(X_test)

```

```
preprocessed_feature_names = []
for name, transformer, columns in preprocess.transformers_:
    if hasattr(transformer, 'get_feature_names_out'):
        preprocessed_feature_names.extend(transformer.get_feature_names_out(columns))
    else:
        preprocessed_feature_names.extend(columns)

X_raw_train_df = pd.DataFrame(
    X_raw_train, columns=preprocessed_feature_names, index=X_train.index)
X_raw_val_df = pd.DataFrame(
    X_raw_val, columns=preprocessed_feature_names, index=X_val.index)
X_raw_test_df = pd.DataFrame(
    X_raw_test, columns=preprocessed_feature_names, index=X_test.index)

out_dir = Path('splits'); out_dir.mkdir(parents=True, exist_ok=True)
pd.concat([X_raw_train_df, y_train.rename(TARGET)],
          axis=1).to_csv(out_dir / 'raw_train.csv', index=False)
pd.concat([X_raw_val_df, y_val.rename(TARGET)],
          axis=1).to_csv(out_dir / 'raw_val.csv', index=False)
pd.concat([X_raw_test_df, y_test.rename(TARGET)],
          axis=1).to_csv(out_dir / 'raw_test.csv', index=False)

display(X_raw_train_df)
```

Sử dụng mô hình Decision Tree để chọn lọc các đặc trưng quan trọng nhất. Độ quan trọng của mỗi đặc trưng được tính toán và sắp xếp, từ đó chọn ra các đặc trưng hàng đầu để tạo ra một bộ dữ liệu mới, tối ưu hóa cho mô hình.

Tạo bộ dữ liệu mới với Decision Tree

```
dt_feature_selection_pipeline = Pipeline([
    ('preprocess', preprocess),
    ('decision_tree', DecisionTreeClassifier(random_state=42))
])

dt_feature_selection_pipeline.fit(X_train, y_train)
feature_importance_series = pd.Series(
    dt_feature_selection_pipeline.named_steps['decision_tree'].feature_importances_,
    index=preprocessed_feature_names
)
sorted_feature_importances = feature_importance_series.sort_values(ascending=False)
display(sorted_feature_importances)

selected_features = sorted_feature_importances.head(K_features).index.tolist()
print(f'Top {K_features} selected features: {selected_features}')

X_dt_train = X_raw_train_df[selected_features]
X_dt_val = X_raw_val_df[selected_features]
X_dt_test = X_raw_test_df[selected_features]
display(X_dt_train.head())

pd.concat([X_dt_train, y_train.rename(TARGET)],
```

```
          axis=1).to_csv(out_dir / 'dt_train.csv', index=False)
pd.concat([X_dt_val, y_val.rename(TARGET)],
          axis=1).to_csv(out_dir / 'dt_val.csv', index=False)
pd.concat([X_dt_test, y_test.rename(TARGET)],
          axis=1).to_csv(out_dir / 'dt_test.csv', index=False)
```

Hình 3: Mức độ ảnh hưởng của các đặc trưng dựa vào Decision Tree trên bộ `dt_train`.

Tiếp tục quá trình bằng cách tạo các đặc trưng mới (Feature Engineering), như tỷ lệ cholesterol theo tuổi, để làm giàu dữ liệu. Các đặc trưng này được tích hợp vào một pipeline riêng, để áp dụng lên bộ dữ liệu gốc (chưa được xử lý).

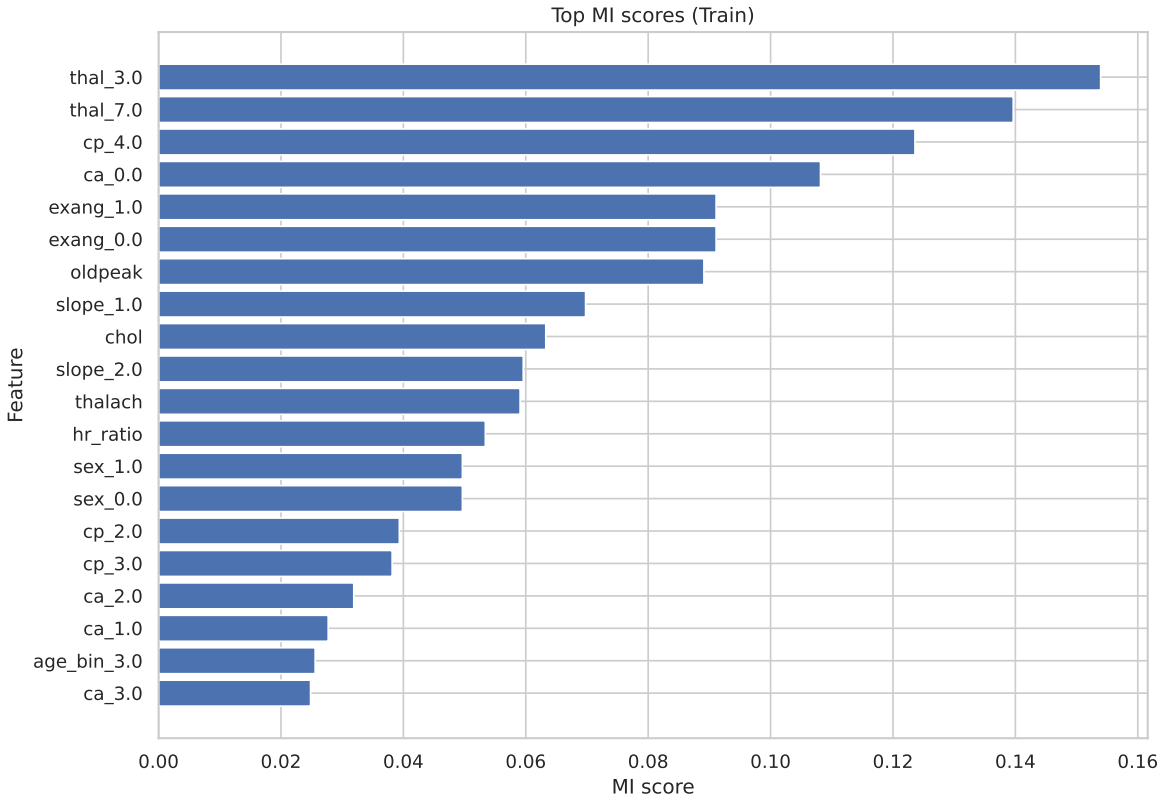
```
def add_new_features_func(df):
    df = df.copy()
    if {'chol','age'} <= set(df.columns):
        df['chol_per_age'] = df['chol']/df['age']
    if {'trestbps','age'} <= set(df.columns):
        df['bps_per_age'] = df['trestbps']/df['age']
    if {'thalach','age'} <= set(df.columns):
        df['hr_ratio'] = df['thalach']/df['age']
    if 'age' in df.columns:
        df['age_bin'] = pd.cut(
            df['age'], bins=5, labels=False
        ).astype('category')
    return df

```

```python
class AddNewFeaturesTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        self.columns_ = X.columns
        self.new_features_ = []
        if {'chol','age'} <= set(X.columns):
            self.new_features_.append('chol_per_age')
        if {'trestbps','age'} <= set(X.columns):
            self.new_features_.append('bps_per_age')
        if {'thalach','age'} <= set(X.columns):
            self.new_features_.append('hr_ratio')
        if 'age' in X.columns:
            self.new_features_.append('age_bin')
        return self

    def transform(self, X):
        return add_new_features_func(X)

    def get_feature_names_out(self, input_features=None):
        return list(self.columns_) + self.new_features_


gen_num = ['chol_per_age','bps_per_age','hr_ratio']
gen_cat = ['age_bin']
all_nums = [c for c in numeric_cols] + gen_num
all_cats = [c for c in categorical_cols] + gen_cat

num_proc = Pipeline([('imp', SimpleImputer(strategy='median')),
                     ('sc',  StandardScaler())])
cat_proc = Pipeline([('imp', SimpleImputer(strategy='most_frequent')),
                     ('ohe', OneHotEncoder(handle_unknown='ignore', sparse_output=False
                                           ))])
pre = ColumnTransformer([
    ('num', num_proc, all_nums),
    ('cat', cat_proc, all_cats),
], verbose_feature_names_out=False).set_output(transform='pandas')
fe_pre = Pipeline([
    ('add', AddNewFeaturesTransformer()),
    ('pre', pre),
]).set_output(transform='pandas')

Xt_tr = fe_pre.fit_transform(X_train, y_train)
Xt_va = fe_pre.transform(X_val)
Xt_te = fe_pre.transform(X_test)

nz_cols = Xt_tr.columns[Xt_tr.nunique(dropna=False) > 1]
Xt_tr = Xt_tr[nz_cols]
Xt_va = Xt_va[nz_cols]
Xt_te = Xt_te[nz_cols]
```

Áp dụng phương pháp Mutual Information (như dự án trước) để đánh giá và chọn lọc các đặc

trưng có mối liên hệ cao nhất với biến mục tiêu. Sau đó, lưu lại bộ dữ liệu FE mới được tạo này.

```
ohe = fe_pre.named_steps['pre'].named_transformers_['cat'].named_steps['ohe']
cat_names = list(ohe.get_feature_names_out(all_cats))
is_discrete = np.array(
    [c in cat_names for c in Xt_tr.columns],
    dtype=bool
)
mi = mutual_info_classif(Xt_tr.values, y_train.values,
                         discrete_features=is_discrete,
                         random_state=42)
mi_series = pd.Series(
    mi, index=Xt_tr.columns).sort_values(ascending=False)

N = min(20, len(mi_series))
topN = mi_series.head(N).iloc[::-1]
plt.figure(figsize=(10, max(6, 0.35*N)))
plt.barh(topN.index, topN.values)
plt.title('Top MI scores (Train)')
plt.xlabel('MI score')
plt.ylabel('Feature')
plt.tight_layout()
plt.savefig('top_mi_scores.pdf', bbox_inches='tight')
plt.show()

K = raw.columns.drop('target').shape[0]
topk_cols = list(mi_series.head(K).index)
fe_tr = Xt_tr[topk_cols].assign(target=y_train.values)
fe_va = Xt_va[topk_cols].assign(target=y_val.values)
fe_te = Xt_te[topk_cols].assign(target=y_test.values)

out = Path('splits'); out.mkdir(parents=True, exist_ok=True)
fe_tr.to_csv(out/'fe_train.csv', index=False)
fe_va.to_csv(out/'fe_val.csv', index=False)
fe_te.to_csv(out/'fe_test.csv', index=False)

display(pd.Series(
    topk_cols, name='fe_topk_features'
).reset_index(drop=True))
```



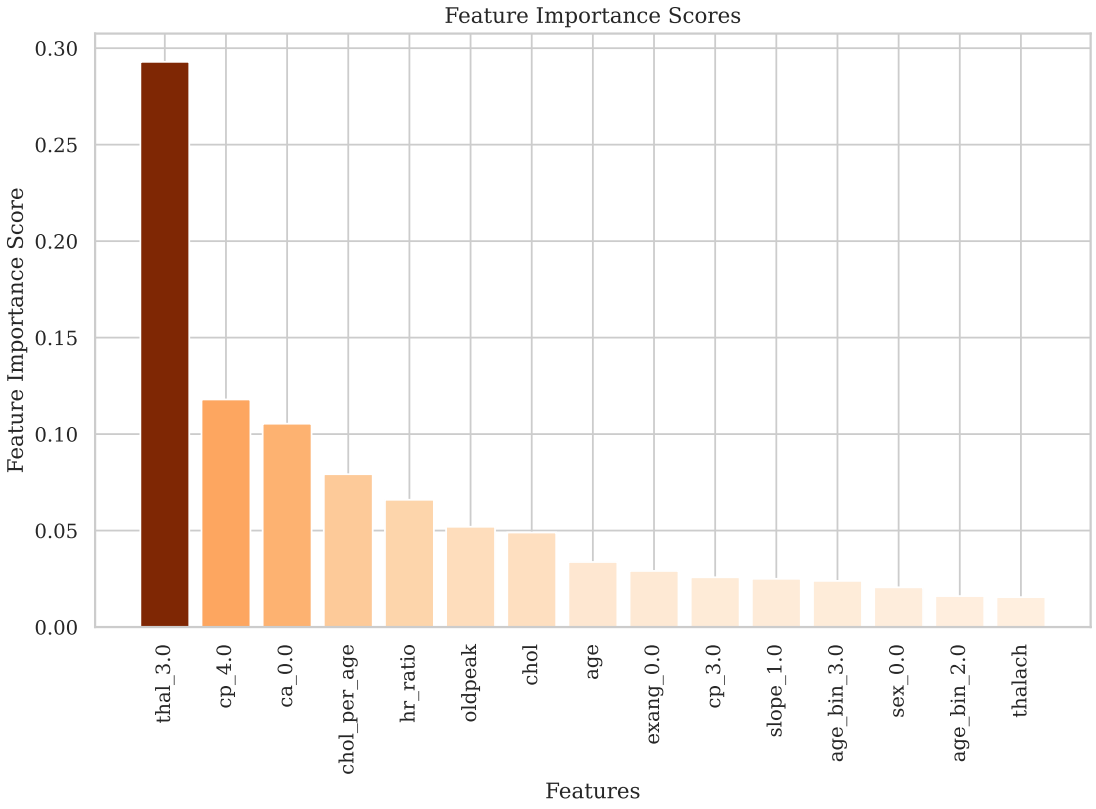
Hình 4: Mức độ ảnh hưởng của các đặc trưng theo phương pháp Mutual Information trên bộ `train` của tập gốc (đã được xử lý).

Cuối cùng, một lần nữa sử dụng mô hình Decision Tree để chọn lọc đặc trưng từ bộ dữ liệu đã qua kỹ thuật tạo đặc trưng (FE), xác định các đặc trưng có tác động lớn nhất. Từ đó, tạo ra các tập dữ liệu cuối cùng với các đặc trưng này.

Tạo tập dữ liệu mới trên tập FE với Decision Tree

```
dt_fe_feature_selection_pipeline = Pipeline([
    ('preprocess', fe_pre),
    ('decision_tree', DecisionTreeClassifier(random_state=42))
])

dt_fe_feature_selection_pipeline.fit(X_train, y_train)
pipeline_feature_names = dt_fe_feature_selection_pipeline.named_steps['preprocess'].
                                                    get_feature_names_out()
feature_importance_series = pd.Series(
    dt_fe_feature_selection_pipeline.named_steps['decision_tree'].feature_importances_,
    index=pipeline_feature_names
)
sorted_feature_importances = feature_importance_series.sort_values(ascending=False)

selected_features = sorted_feature_importances.head(K_features).index.tolist()
print(f'Top {K_features} selected features: {selected_features}')

X_fe_dt_train = Xt_tr[selected_features]
```

```
X_fe_dt_val = Xt_va[selected_features]
X_fe_dt_test = Xt_te[selected_features]
display(X_fe_dt_train.head())

pd.concat([X_fe_dt_train, y_train.rename(TARGET)],
          axis=1).to_csv(out_dir / 'fe_dt_train.csv', index=False)
pd.concat([X_fe_dt_val, y_val.rename(TARGET)],
          axis=1).to_csv(out_dir / 'fe_dt_val.csv', index=False)
pd.concat([X_fe_dt_test, y_test.rename(TARGET)],
          axis=1).to_csv(out_dir / 'fe_dt_test.csv', index=False)
```



Hình 5: Mức độ ảnh hưởng của các đặc trưng dựa vào Decision Tree trên bộ `train` của tập FE.

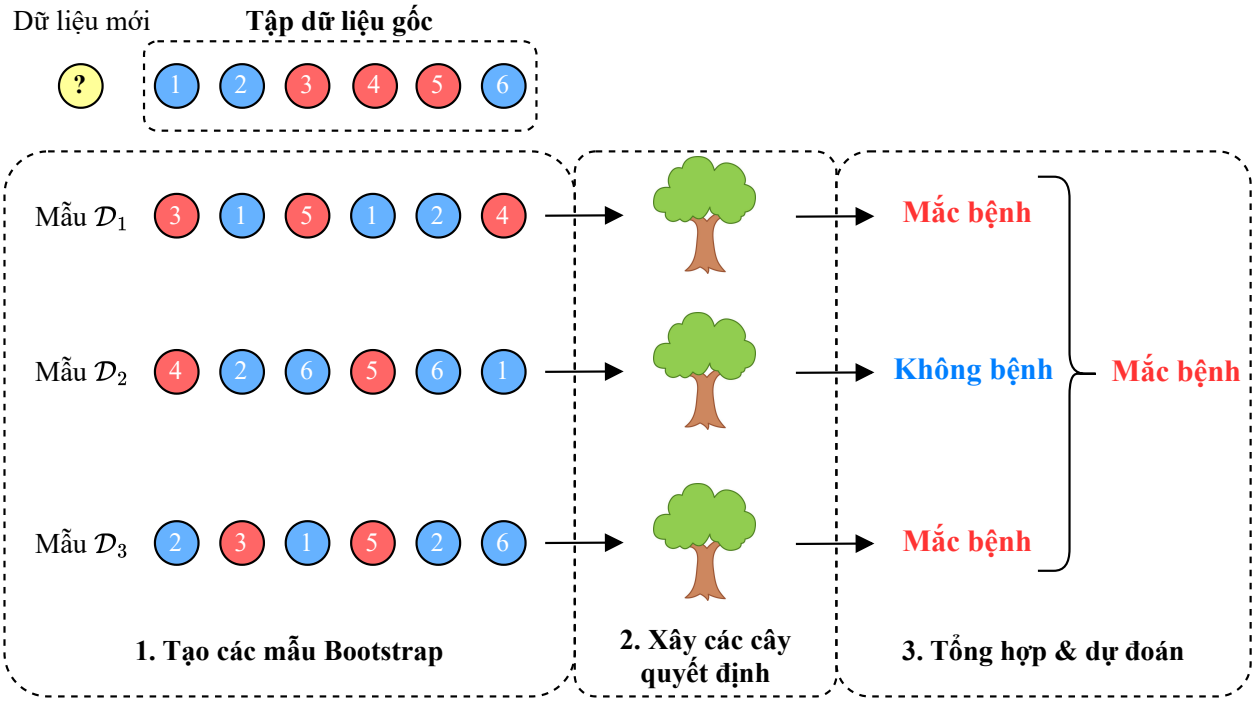
Sử dụng mã bên dưới để nén các tập dữ liệu thành `data.zip` để tải xuống từ Google Colab.

Nén các tập dữ liệu để tải xuống

```
!zip -r dataset.zip splits
```

II.3. Random Forest

II.3.1. Giới thiệu



Hình 6: Minh họa cách hoạt động của Random Forest.

Random Forest là một thuật toán học máy có giám sát thuộc nhóm ensemble, hoạt động bằng cách xây dựng một "rừng" gồm nhiều Decision Tree trong quá trình huấn luyện. Mục tiêu chính của thuật toán là cải thiện độ chính xác và khắc phục nhược điểm lớn của một cây quyết định đơn lẻ: xu hướng bị overfitting (quá khớp) và có phương sai cao.

Để đảm bảo các cây trong rừng là khác biệt và không bị tương quan cao với nhau, Random Forest áp dụng hai kỹ thuật ngẫu nhiên hóa cốt lõi:

(i) **Bagging (Bootstrap Aggregating):** Mỗi cây được huấn luyện trên một tập dữ liệu con khác nhau, được lấy ngẫu nhiên có hoàn lại từ tập dữ liệu huấn luyện ban đầu.

(ii) **Feature Randomness:** Tại mỗi bước phân nhánh của một cây, thay vì xem xét toàn bộ các đặc trưng, thuật toán chỉ chọn một tập con ngẫu nhiên của các đặc trưng để tìm ra điểm chia tối ưu.

Khi dự đoán trên dữ liệu mới, kết quả cuối cùng được tổng hợp từ tất cả các cây. Cụ thể, mô hình sẽ lấy lớp có nhiều phiếu bầu nhất cho bài toán **phân loại** (cơ chế *majority voting*), hoặc tính giá trị trung bình của các dự đoán cho bài toán **hồi quy**. Bằng cách tổng hợp từ nhiều cây đa dạng, Random Forest tạo ra một mô hình tổng thể vừa mạnh mẽ, chính xác, vừa có khả năng khái quát hóa tốt trên dữ liệu chưa từng thấy.

II.3.2. Triển khai

Lưu ý: Các khối code được đánh dấu " * " nghĩa là được sử dụng nhiều lần mà không cần sửa đổi giữa các mô hình trong dự án này.
Bắt đầu bằng cách nhập các thư viện cần thiết, thiết lập một số ngẫu nhiên chung để đảm bảo tính tái lập của kết quả.

***Tải các thư viện cần thiết**

```
import os
import random
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import StratifiedKFold, cross_val_score

warnings.filterwarnings('ignore')

SEED = 42
os.environ['PYTHONHASHSEED'] = str(SEED)
np.random.seed(SEED)
random.seed(SEED)
print(f'Seed: {SEED}')
```

Sau đó, tải xuống bộ dữ liệu được lưu trữ trên Google Drive.

***Tải các bộ dữ liệu**

```
# https://drive.google.com/drive/folders/1cMoqIDEgGYDVzv8B7cKp3csxujQ4OFp7?usp=
                                                drive_link
!gdown --folder 1cMoqIDEgGYDVzv8B7cKp3csxujQ4OFp7
```

Tiếp theo, một hàm được tạo để đọc các tệp `.csv`, hiển thị năm dòng đầu tiên, đếm số lượng các nhãn mục tiêu và in kích thước của dữ liệu.

***Hàm đọc dữ liệu trong file .csv**

```
def read_csv(file_path):
    df = pd.read_csv(file_path)
    display(df.head())

```

```
    X = df.drop('target', axis=1)
    y = df['target']
    display(y.value_counts())

    print('Shape df: ', df.shape)
    print('Shape X: ', X.shape)
    print('Shape y: ', y.shape)

    return X, y
```

Sử dụng hàm vừa định nghĩa, tiến hành đọc và tải các tập dữ liệu huấn luyện, bao gồm các phiên bản dữ liệu gốc, dữ liệu đã qua xử lý kỹ thuật xử lý đặc trưng (FE) và các tập dữ liệu được tạo bằng cách chọn lọc đặc trưng bằng Decision Tree (DT) trên 2 bộ trên.

***Đọc các tập dữ liệu**

```
# Original Dataset
X_train, y_train = read_csv('dataset_v3/raw_train.csv')
X_val, y_val = read_csv('dataset_v3/raw_val.csv')
X_test, y_test = read_csv('dataset_v3/raw_test.csv')

# FE Dataset
X_fe_train, y_fe_train = read_csv('dataset_v3/fe_train.csv')
X_fe_val, y_fe_val = read_csv('dataset_v3/fe_val.csv')
X_fe_test, y_fe_test = read_csv('dataset_v3/fe_test.csv')

# Original + DT Dataset
X_dt_train, y_dt_train = read_csv('dataset_v3/dt_train.csv')
X_dt_val, y_dt_val = read_csv('dataset_v3/dt_val.csv')
X_dt_test, y_dt_test = read_csv('dataset_v3/dt_test.csv')

# FE + DT Dataset
X_fe_dt_train, y_fe_dt_train = read_csv('dataset_v3/fe_dt_train.csv')
X_fe_dt_val, y_fe_dt_val = read_csv('dataset_v3/fe_dt_val.csv')
X_fe_dt_test, y_fe_dt_test = read_csv('dataset_v3/fe_dt_test.csv')
```

Sau đó, định nghĩa một hàm để tìm số lượng cây tối ưu cho mô hình Random Forest bằng phương pháp Stratified K-Fold Cross-Validation. Quá trình này giúp đánh giá hiệu suất mô hình trên các giá trị `n_estimators` (số cây con) khác nhau để chọn ra giá trị tốt nhất.

Hàm tìm số lượng cây con tối ưu

```
def find_optimal_rf(
    X_train,y_train,n_estimators_range=range(50, 501, 50),cv_splits=3,
    max_depth=5,min_samples_split=2,min_samples_leaf=1,
    max_features='sqrt',bootstrap=True, class_weight=None
):
    cv = StratifiedKFold(n_splits=cv_splits, shuffle=True, random_state=SEED)
    scores = []
    for n in n_estimators_range:
```

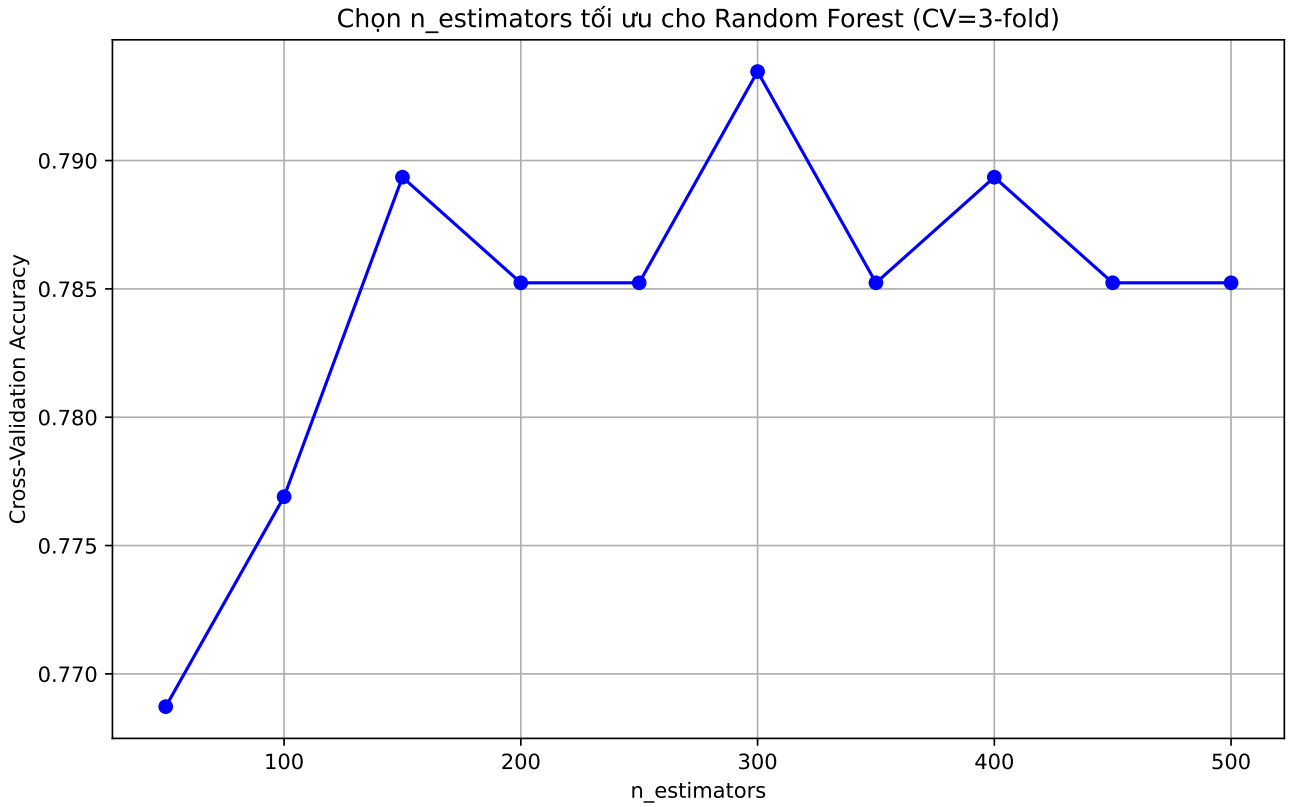
```
        rf = RandomForestClassifier(
            n_estimators=n,max_depth=max_depth,min_samples_split=min_samples_split,
            min_samples_leaf=min_samples_leaf,max_features=max_features,
            bootstrap=bootstrap,class_weight=class_weight,n_jobs=-1,random_state=SEED
        )
        cv_score = cross_val_score(rf,X_train,y_train,
        cv=cv,scoring='accuracy',n_jobs=-1)
        scores.append(cv_score.mean())
    plt.figure(figsize=(10, 6))
    plt.plot(list(n_estimators_range), scores, 'bo-')
    plt.title(f'Chọn n_estimators tối ưu cho Random Forest (CV={cv_splits}-fold)')
    plt.xlabel('n_estimators')
    plt.ylabel('Cross-Validation Accuracy')
    plt.grid(True)
    plt.show()

    best_n = list(n_estimators_range)[int(np.argmax(scores))]
    print(f'n_estimators tối ưu (CV): {best_n}')

    best_model = RandomForestClassifier(
        n_estimators=best_n,max_depth=max_depth,min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,max_features=max_features,
        bootstrap=bootstrap,class_weight=class_weight,n_jobs=-1,random_state=SEED
    )
    best_model.fit(X_train, y_train)
    return best_model, best_n, max(scores)
```

Tiếp đến, một hàm khác được tạo để huấn luyện và đánh giá mô hình trên tập dữ liệu huấn luyện và thẩm định, sau đó dự đoán và đánh giá hiệu suất trên tập kiểm thử.

Hàm train/val và hàm test

```
def evaluate_val_rf(X_train, y_train, X_val, y_val,
                    n_estimators_range=range(50, 501, 50),cv_splits=5,max_depth=5,
                    min_samples_split=2,min_samples_leaf=1,max_features='sqrt',
                    bootstrap=True,class_weight=None):
    print('Tìm n_estimators tối ưu cho Random Forest...')
    rf_model, best_n, cv_acc = find_optimal_rf(
        X_train,y_train,n_estimators_range=n_estimators_range,cv_splits=cv_splits,
        max_depth=max_depth,min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,max_features=max_features,
        bootstrap=bootstrap,class_weight=class_weight
    )

    val_pred = rf_model.predict(X_val)
    val_acc = accuracy_score(y_val, val_pred)
    print(f'\nĐộ chính xác Random Forest trên tập validation: {val_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_val, val_pred))
    return rf_model, val_acc, {'n_estimators': best_n}

def evaluate_test_rf(rf_model, X_test, y_test):
```

```
    test_pred = rf_model.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    print(f'\nĐộ chính xác Random Forest trên tập test: {test_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_test, test_pred))
    return test_acc
```



Hình 7: Biểu đồ độ chính xác khi áp dụng Stratified K-Fold Cross-Validation cho mô hình Random Forest tại các giá trị `n_estimators` khác nhau.

Sử dụng các hàm trên, tiến hành huấn luyện và đánh giá mô hình Random Forest trên bốn bộ dữ liệu khác nhau: dữ liệu gốc, dữ liệu FE, dữ liệu gốc kết hợp với DT, và dữ liệu FE kết hợp với DT.

Huấn luyện và đánh giá trên các tập dữ liệu

```
# RF on Original Dataset
rf_model, val_acc, best_params = evaluate_val_rf(
    X_train, y_train, X_val, y_val
)
test_acc = evaluate_test_rf(rf_model, X_test, y_test)

# RF on Feature Engineering Dataset
```

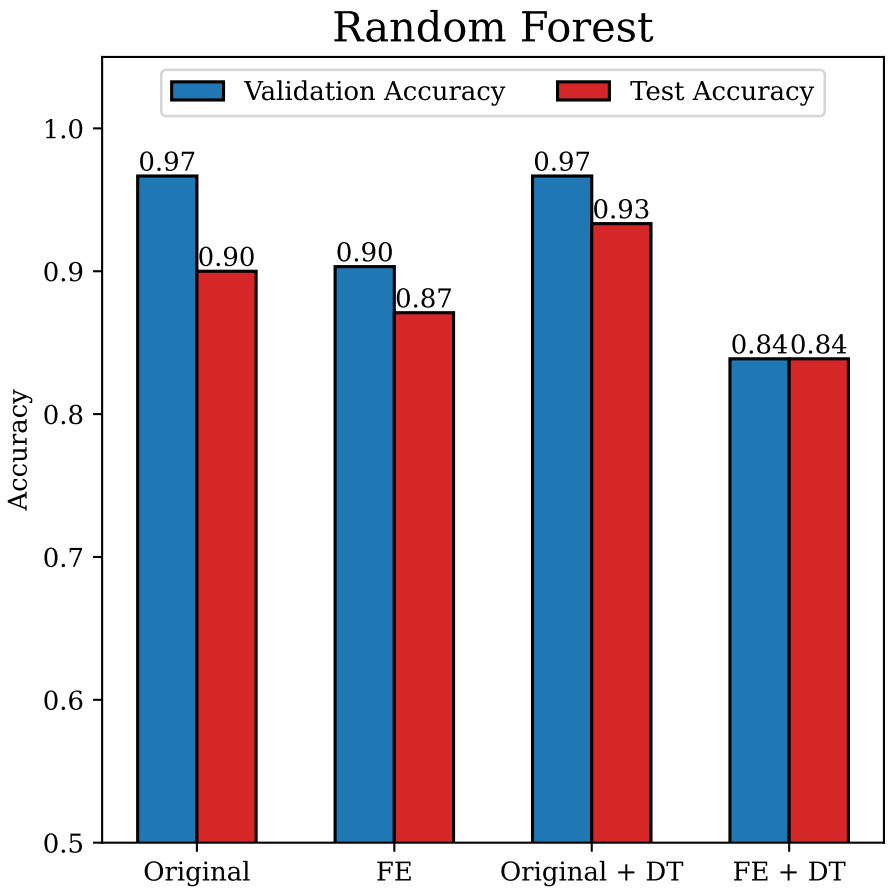
```
rf_model, val_fe_acc, best_params = evaluate_val_rf(
    X_fe_train, y_fe_train, X_fe_val, y_fe_val
)

# RF on Original DT Dataset
rf_model, val_dt_acc, best_params = evaluate_val_rf(
    X_dt_train, y_dt_train, X_dt_val, y_dt_val
)
test_dt_acc = evaluate_test_rf(rf_model, X_dt_test, y_dt_test)

# RF on Feature Engineering DT Dataset
rf_model, val_fe_dt_acc, best_params = evaluate_val_rf(
    X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
)
test_fe_dt_acc = evaluate_test_rf(rf_model, X_fe_dt_test, y_fe_dt_test)
```

Cuối cùng, một biểu đồ cột được vẽ để so sánh trực quan hiệu suất (độ chính xác) của mô hình trên các tập dữ liệu thẩm định và kiểm thử, giúp dễ dàng nhận biết bộ dữ liệu nào mang lại kết quả tốt nhất.

***Vẽ biểu đồ so sánh hiệu suất mô hình trên các tập dữ liệu**

```
plt.rcParams['font.family'] = 'Serif'

labels = ['Original', 'FE', 'Original + DT', 'FE + DT']
val_accs = [val_acc, test_acc, val_dt_acc, test_dt_acc]
test_accs = [val_fe_acc, test_fe_acc, val_fe_dt_acc, test_fe_dt_acc]

x = np.arange(len(labels))
width = 0.3

fig, ax = plt.subplots(figsize=(5, 5))

rects1 = ax.bar(x - width/2, val_accs,  width,
                label='Validation Accuracy',
                color='tab:blue', edgecolor='black', linewidth=1.2)
rects2 = ax.bar(x + width/2, test_accs, width,
                label='Test Accuracy',
                color='tab:red', edgecolor='black', linewidth=1.2)

ax.set_ylim(0.5, 1.05)
ax.set_ylabel('Accuracy')
ax.set_title('Random Forest', fontsize=16)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(ncol=2, loc='upper center')

def autolabel(rects):
    for rect in rects:
        h = rect.get_height()
        ax.annotate(f'{h:.2f}', xy=(rect.get_x()+rect.get_width()/2, h),
                    ha='center', va='bottom')
```

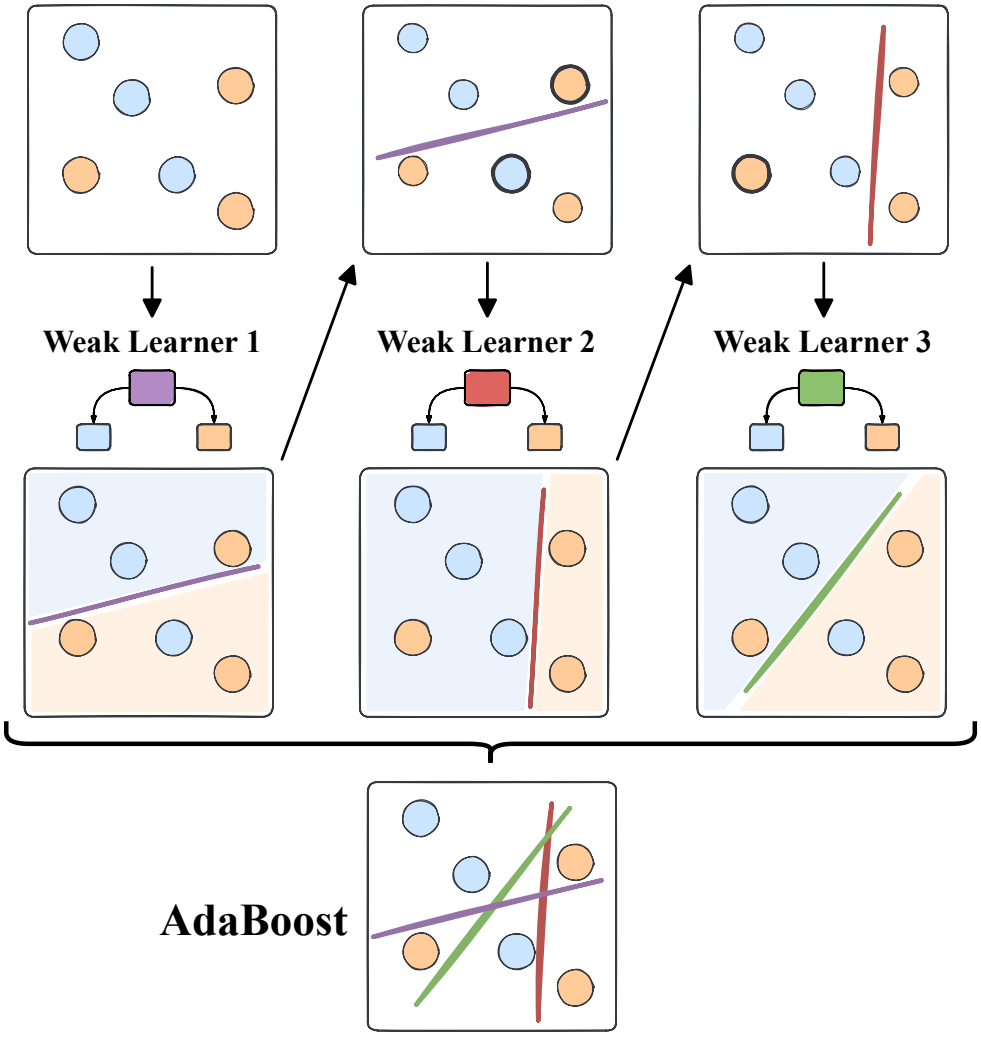
```
31
32 autolabel(rects1)
33 autolabel(rects2)
34
35 fig.tight_layout()
36 plt.show()
```



Hình 8: Hiệu suất của mô hình Random Forest trên 4 bộ dữ liệu.

II.4. AdaBoost

II.4.1. Giới thiệu



Hình 9: Hình minh họa thuật toán AdaBoost cho bài toán phân loại.

AdaBoost [2] hoạt động theo một quy trình lặp lại, chú trọng vào các điểm dữ liệu mà các mô hình trước đó đã phân loại sai. Cụ thể như sau:

- Ban đầu, mỗi điểm dữ liệu trong tập huấn luyện đều có một trọng số như nhau.
- Một mô hình yếu (Weak Learner 1), chẳng hạn như một cây quyết định đơn giản (stump), được huấn luyện trên dữ liệu.
- Sau khi huấn luyện mô hình yếu đầu tiên, trọng số của các điểm dữ liệu sẽ được điều chỉnh. Các điểm dữ liệu mà mô hình phân loại sai sẽ được tăng trọng số, trong khi các điểm phân loại đúng sẽ giảm trọng số. Điều này buộc mô hình yếu tiếp theo phải tập trung hơn vào các điểm dữ liệu khó.

- Quá trình này lặp lại nhiều lần với các mô hình yếu mới (Weak Learner 2, Weak Learner 3). Mỗi mô hình mới sẽ được huấn luyện trên tập dữ liệu đã được điều chỉnh trọng số, do đó tập trung vào những lỗi của mô hình trước đó.
- Cuối cùng, AdaBoost sẽ kết hợp tất cả các mô hình yếu lại với nhau thành một mô hình tổng thể. Mỗi mô hình yếu sẽ được gán một trọng số dựa trên hiệu suất của nó - những mô hình có độ chính xác cao sẽ có trọng số lớn hơn trong quyết định cuối cùng.

Kết quả là một mô hình dự đoán mạnh mẽ và có độ chính xác cao, được xây dựng từ sự đóng góp của nhiều mô hình yếu, mỗi mô hình đều tập trung vào việc khắc phục những điểm yếu của mô hình trước đó.

II.4.2. Triển khai

Xây dựng một hàm để tìm số lượng cây con tối ưu cho mô hình AdaBoost bằng cách sử dụng Stratified K-Fold Cross-Validation để đánh giá hiệu suất qua các giá trị `n_estimators` khác nhau.

Hàm tìm số lượng cây con tối ưu

```python
def find_optimal_ada(
    X_train, y_train,
    n_estimators_range=range(50, 501, 50),
    cv_splits=3,
    learning_rate=0.1,
    base_max_depth=1,
    algorithm='SAMME'
):
    cv = StratifiedKFold(n_splits=cv_splits, shuffle=True, random_state=SEED)
    scores = []

    for n in n_estimators_range:
        ada = AdaBoostClassifier(
            estimator=DecisionTreeClassifier(max_depth=base_max_depth, random_state=
                                             SEED),
            n_estimators=n, learning_rate=learning_rate,
            algorithm=algorithm, random_state=SEED
        )
        cv_score = cross_val_score(
            ada, X_train, y_train, cv=cv, scoring='accuracy', n_jobs=-1
        )
        scores.append(cv_score.mean())

    plt.figure(figsize=(10, 6))
    plt.plot(list(n_estimators_range), scores, 'bo-')
    plt.title(f'Chọn n_estimators tối ưu cho AdaBoost (CV={cv_splits}-fold)')
    plt.xlabel('n_estimators')
    plt.ylabel('Cross-Validation Accuracy')
    plt.grid(True)
    plt.show()

    best_n = list(n_estimators_range)[int(np.argmax(scores))]
    print(f'n_estimators tối ưu (CV): {best_n}')
```

```
    best_model = AdaBoostClassifier(
        estimator=DecisionTreeClassifier(max_depth=base_max_depth, random_state=SEED),
        n_estimators=best_n, learning_rate=learning_rate,
        algorithm=algorithm, random_state=SEED
    )
    best_model.fit(X_train, y_train)
    return best_model, best_n, max(scores)
```

Tiếp theo, tạo các hàm để huấn luyện và đánh giá mô hình AdaBoost. Một hàm sẽ huấn luyện mô hình trên tập train và đánh giá trên tập val, sau đó hàm còn lại sẽ đánh giá hiệu suất trên tập test.

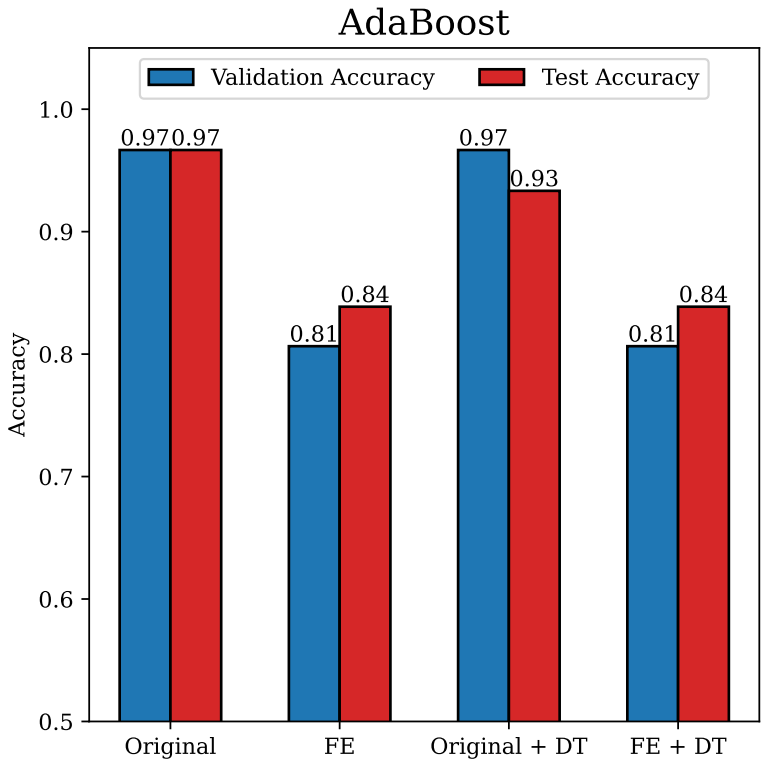
Hàm train/val và hàm test

```
def evaluate_val_ada(X_train, y_train, X_val, y_val,
                     n_estimators_range=range(50, 501, 50),
                     cv_splits=3,
                     learning_rate=0.1,
                     base_max_depth=1,
                     algorithm='SAMME'):
    print('Tìm n_estimators tối ưu cho AdaBoost...')
    ada_model, best_n, cv_acc = find_optimal_ada(
        X_train, y_train,
        n_estimators_range=n_estimators_range,
        cv_splits=cv_splits,
        learning_rate=learning_rate,
        base_max_depth=base_max_depth,
        algorithm=algorithm
    )

    val_pred = ada_model.predict(X_val)
    val_acc = accuracy_score(y_val, val_pred)
    print(f'\nĐộ chính xác AdaBoost trên tập validation: {val_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_val, val_pred))
    return ada_model, val_acc, {'n_estimators': best_n}

def evaluate_test_ada(ada_model, X_test, y_test):
    test_pred = ada_model.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    print(f'\nĐộ chính xác AdaBoost trên tập test: {test_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_test, test_pred))
    return test_acc
```

Cuối cùng, sử dụng các hàm đã định nghĩa để huấn luyện và đánh giá mô hình trên bốn bộ dữ liệu khác nhau, bao gồm dữ liệu gốc, dữ liệu đã qua kỹ thuật đặc trưng (FE), dữ liệu kết hợp với Decision Tree (DT), và dữ liệu FE kết hợp với DT.

Huấn luyện và đánh giá trên các tập dữ liệu

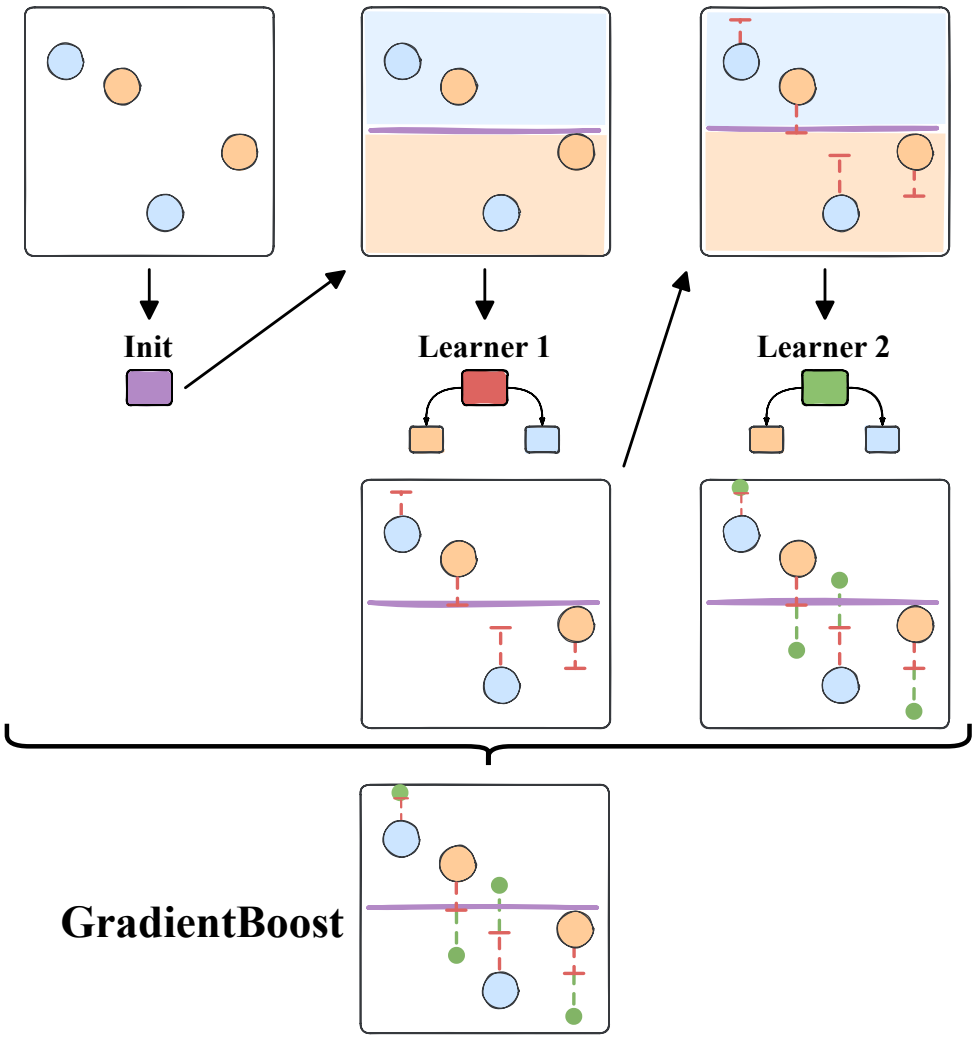
```python
# AB on Original Dataset
ada_model, val_acc, best_params = evaluate_val_ada(
    X_train, y_train, X_val, y_val
)
test_acc = evaluate_test_ada(ada_model, X_test, y_test)

# AB on FE Dataset
ada_model, val_fe_acc, best_params = evaluate_val_ada(
    X_fe_train, y_fe_train, X_fe_val, y_fe_val
)
test_fe_acc = evaluate_test_ada(ada_model, X_fe_test, y_fe_test)

# AB on Original DT Dataset
ada_model, val_dt_acc, best_params = evaluate_val_ada(
    X_dt_train, y_dt_train, X_dt_val, y_dt_val
)
test_dt_acc = evaluate_test_ada(ada_model, X_dt_test, y_dt_test)

# AB on FE + DT Dataset
ada_model, val_fe_dt_acc, best_params = evaluate_val_ada(
    X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
)
test_fe_dt_acc = evaluate_test_ada(ada_model, X_fe_dt_test, y_fe_dt_test)
```



Hình 10: Hiệu suất của mô hình AdaBoost trên 4 bộ dữ liệu.

II.5. Gradient Boosting

II.5.1. Giới thiệu



Hình 11: Hình minh họa thuật toán Gradient Boosting cho bài toán phân loại.

Gradient Boosting bắt đầu với một mô hình ban đầu rất đơn giản (Init), thường là một giá trị trung bình hoặc một hằng số. Sau đó lặp lại quá trình sau:

- Tại mỗi bước, thuật toán sẽ tính toán phần dư, tức là sự khác biệt giữa giá trị thực tế và giá trị dự đoán của mô hình hiện tại. Phần dư này chính là "lỗi"mà mô hình cần phải học cách sửa.
- Một mô hình mới, thường là một cây quyết định (Learner 1, Learner 2, ...), được huấn luyện để dự đoán chính xác những phần dư đó. Thay vì dự đoán kết quả cuối cùng, mô hình này tập trung vào việc sửa lỗi của mô hình trước.
- Mô hình mới này sẽ được thêm vào mô hình tổng thể, cùng với một hệ số học (learning rate) nhỏ để kiểm soát tốc độ học, giúp tránh việc mô hình trở nên quá phức tạp (overfitting) một cách nhanh chóng.

Quá trình này lặp lại cho đến khi mô hình tổng thể đạt được độ chính xác mong muốn hoặc đã xây dựng đủ số lượng cây. Kết quả cuối cùng là một mô hình tổng hợp, cực kỳ hiệu quả, được tạo thành từ tổng hợp của tất cả các mô hình nhỏ đã được huấn luyện tuần tự.

II.5.2. Triển khai

Tương tự thuật toán trước, tạo một hàm để tìm số lượng cây con tối ưu cho mô hình Gradient Boosting bằng cách sử dụng Stratified K-Fold Cross-Validation nhằm xác định hiệu suất qua các giá trị `n_estimators`.

Hàm tìm số lượng cây con tối ưu

```python
def find_optimal_gb(
    X_train, y_train,
    n_estimators_range=range(50, 501, 50),
    cv_splits=3
):
    cv = StratifiedKFold(n_splits=cv_splits, shuffle=True, random_state=SEED)
    scores = []

    for n in n_estimators_range:
        gb = GradientBoostingClassifier(
            n_estimators=n, learning_rate=0.1,
            max_depth=5, subsample=1.0, random_state=SEED
        )
        cv_score = cross_val_score(
            gb, X_train, y_train,
            cv=cv, scoring='accuracy', n_jobs=-1
        )
        scores.append(cv_score.mean())

    plt.figure(figsize=(10, 6))
    plt.plot(list(n_estimators_range), scores, 'bo-')
    plt.title(f'Chọn n_estimators tối ưu cho Gradient Boosting (CV={cv_splits}-fold)')
    plt.xlabel('n_estimators')
    plt.ylabel('Cross-Validation Accuracy')
    plt.grid(True)
    plt.show()

    best_n = list(n_estimators_range)[int(np.argmax(scores))]
    print(f'n_estimators tối ưu (CV): {best_n}')

    best_model = GradientBoostingClassifier(
        n_estimators=best_n,
        learning_rate=0.1,
        max_depth=5,
        subsample=1.0,
        random_state=SEED
    )
    best_model.fit(X_train, y_train)
    return best_model, best_n, max(scores)
```

Tiếp theo, xây dựng các hàm để huấn luyện trên tập train và đánh giá mô hình Gradient Boosting trên tập val và tập test.

Hàm train/val và hàm test

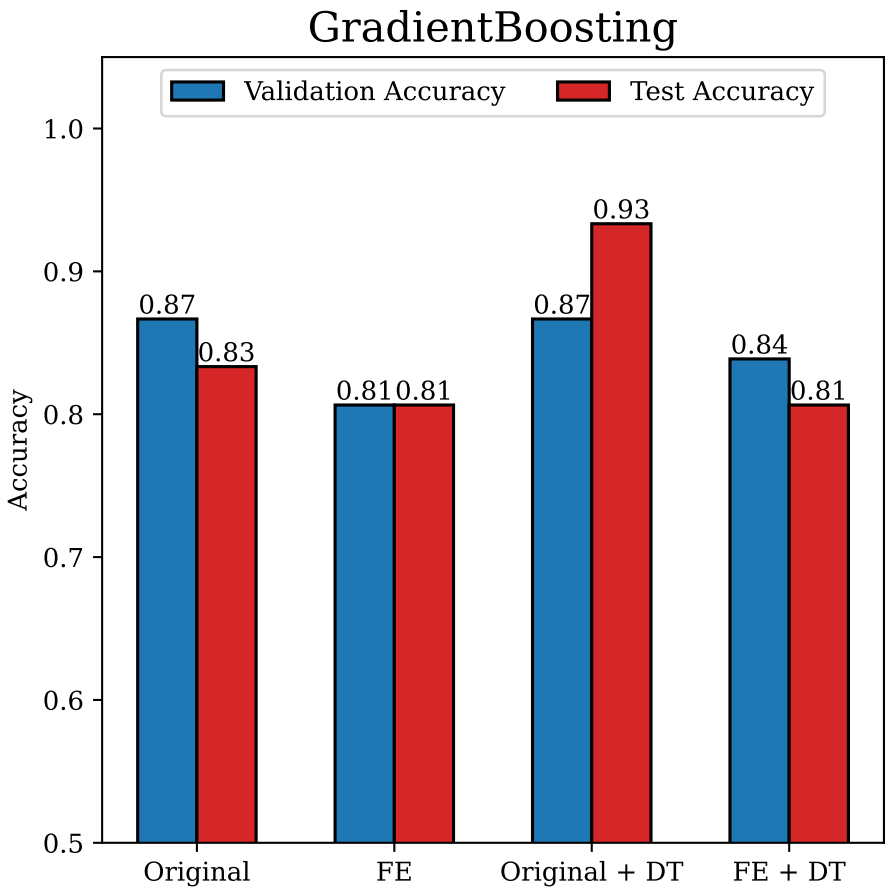
```
def evaluate_val_gb(X_train, y_train, X_val, y_val,
                    n_estimators_range=range(50, 501, 50),
                    cv_splits=3):
    print('Tìm n_estimators tối ưu cho Gradient Boosting...')
    gb_model, best_n, cv_acc = find_optimal_gb(
        X_train, y_train,
        n_estimators_range=n_estimators_range,
        cv_splits=cv_splits
    )

    val_pred = gb_model.predict(X_val)
    val_acc = accuracy_score(y_val, val_pred)
    print(f'\nĐộ chính xác GB trên tập validation: {val_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_val, val_pred))
    return gb_model, val_acc, {'n_estimators': best_n}

def evaluate_test_gb(gb_model, X_test, y_test):
    test_pred = gb_model.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    print(f'\nĐộ chính xác GB trên tập test: {test_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_test, test_pred))
    return test_acc
```

Sau đó, tiến hành huấn luyện và đánh giá mô hình trên bốn bộ dữ liệu khác nhau.

Huấn luyện và đánh giá trên các tập dữ liệu

```
# GB on Original Dataset
gb_model, val_acc, best_params = evaluate_val_gb(
    X_train, y_train, X_val, y_val
)
test_acc = evaluate_test_gb(gb_model, X_test, y_test)

# GB on Feature Engineering Dataset
gb_model, val_fe_acc, best_params = evaluate_val_gb(
    X_fe_train, y_fe_train, X_fe_val, y_fe_val
)
test_fe_acc = evaluate_test_gb(gb_model, X_fe_test, y_fe_test)

# GB on Original DT Dataset
gb_model, val_dt_acc, best_params = evaluate_val_gb(
    X_dt_train, y_dt_train, X_dt_val, y_dt_val
)
test_dt_acc = evaluate_test_gb(gb_model, X_dt_test, y_dt_test)

```

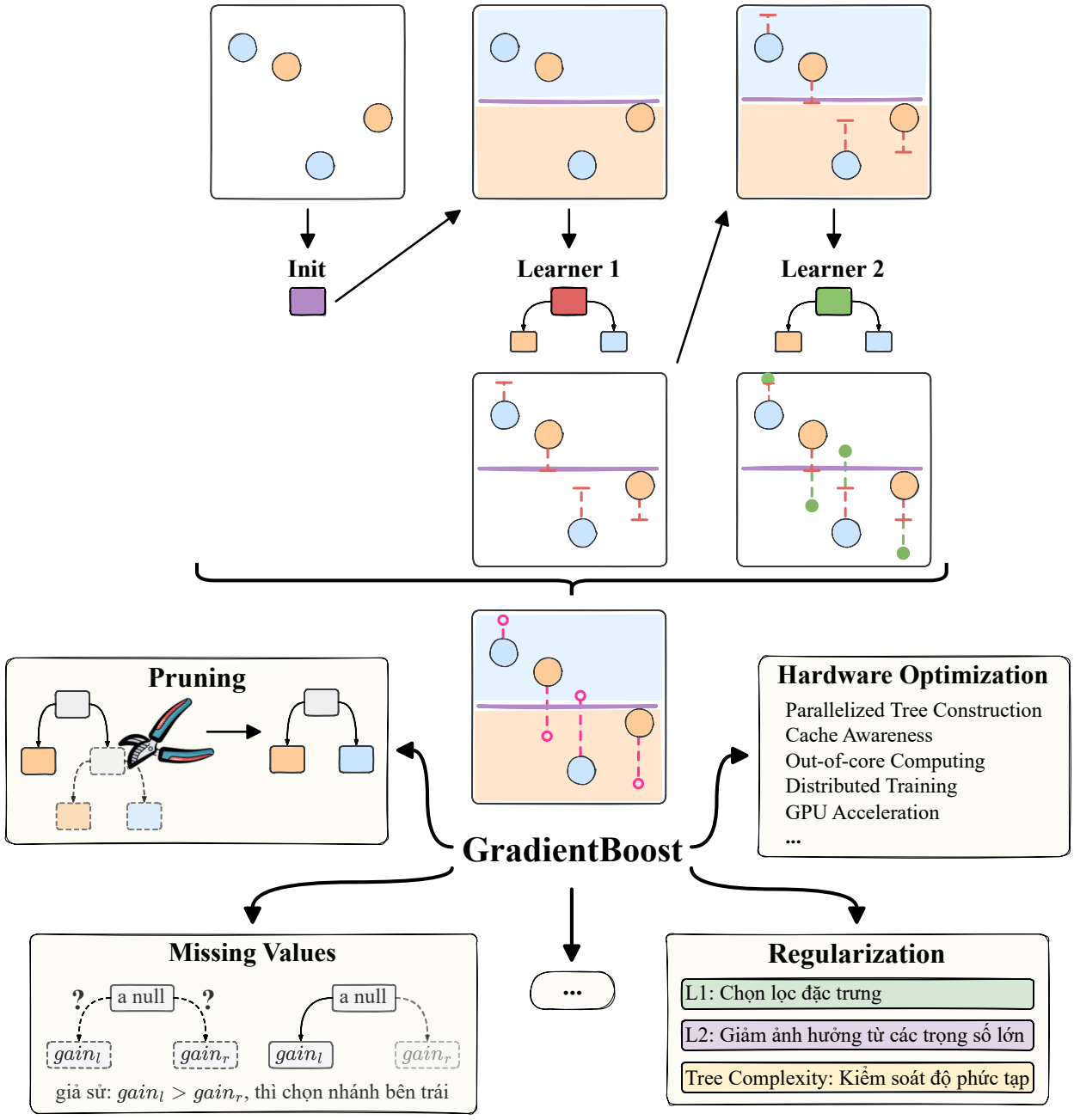
```python
# GB on Feature Engineering DT Dataset
gb_model, val_fe_dt_acc, best_params = evaluate_val_gb(
    X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
)
test_fe_dt_acc = evaluate_test_gb(gb_model, X_fe_dt_test, y_fe_dt_test)
```



Hình 12: Hiệu suất của mô hình Gradient Boosting trên 4 bộ dữ liệu.

II.6. XGBoost

II.6.1. Giới thiệu



Hình 13: Hình minh họa thuật toán XGBoost cho bài toán phân loại.

XGBoost [3] cải tiến từ Gradient Boosting bằng cách kết hợp nhiều kỹ thuật để tối ưu hóa hiệu suất và giảm thiểu các vấn đề như quá khớp (overfitting). Cụ thể, các đặc điểm nổi bật của nó bao gồm:

- Sử dụng kỹ thuật như Parallelized Tree Construction (xây dựng cây song song) và GPU

Acceleration (tăng tốc bằng GPU) để tăng tốc độ tính toán, đặc biệt với các bộ dữ liệu lớn.

- Tự động xử lý các Missing Values (các giá trị bị thiếu) bằng cách học cách phân nhánh tốt nhất ngay cả khi dữ liệu bị thiếu.
- Áp dụng các kỹ thuật Regularization (điều chuẩn) như L1 và L2 để kiểm soát độ phức tạp của mô hình và tránh việc mô hình học quá sát với dữ liệu huấn luyện. Pruning (cắt tỉa cây) cũng được sử dụng để loại bỏ các nhánh không cần thiết.

Về cơ bản, XGBoost hoạt động bằng cách xây dựng một loạt các cây quyết định một cách tuần tự (như trong sơ đồ là Learner 1 và Learner 2). Mỗi cây mới được thêm vào sẽ cố gắng sửa lỗi của các cây trước đó, dần dần cải thiện độ chính xác tổng thể của mô hình.

II.6.2. Triển khai

Bắt đầu tương tự như các thuật toán trên với một hàm được tìm số lượng cây con tối ưu cho mô hình XGBoost bằng cách sử dụng Stratified K-Fold Cross-Validation với các giá trị `n_estimators` khác nhau.

Hàm tìm số lượng cây con tối ưu

```
def find_optimal_xgb(
    X_train, y_train,
    n_estimators_range=range(50, 501, 50),
    cv_splits=3,
    learning_rate=0.1,
    max_depth=5,
    subsample=1.0,
    use_gpu=False
):
    cv = StratifiedKFold(n_splits=cv_splits, shuffle=True, random_state=SEED)
    scores = []

    n_classes = len(np.unique(y_train))
    objective = 'binary:logistic' if n_classes == 2 else 'multi:softprob'
    eval_metric = 'logloss' if n_classes == 2 else 'mlogloss'

    for n in n_estimators_range:
        xgb = XGBClassifier(
            n_estimators=n, learning_rate=learning_rate, max_depth=max_depth, subsample
                                                =subsample,
            objective=objective, eval_metric=eval_metric, random_state=SEED, n_jobs=-1,
            tree_method='gpu_hist' if use_gpu else 'hist', verbosity=0
        )
        cv_score = cross_val_score(
            xgb, X_train, y_train,
            cv=cv, scoring='accuracy', n_jobs=-1
        )
        scores.append(cv_score.mean())

    plt.figure(figsize=(10, 6))
    plt.plot(list(n_estimators_range), scores, 'bo-')
```

```
    plt.title(f'Chọn n_estimators tối ưu cho XGBoost (CV={cv_splits}-fold)')
    plt.xlabel('n_estimators')
    plt.ylabel('Cross-Validation Accuracy')
    plt.grid(True)
    plt.show()

    best_n = list(n_estimators_range)[int(np.argmax(scores))]
    print(f'n_estimators tối ưu (CV): {best_n}')

    best_model = XGBClassifier(
        n_estimators=best_n, learning_rate=learning_rate, max_depth=max_depth,
                                            subsample=subsample,
        objective=objective, eval_metric=eval_metric, random_state=SEED, n_jobs=-1,
        tree_method='gpu_hist' if use_gpu else 'hist', verbosity=0
    )
    best_model.fit(X_train, y_train)
    return best_model, best_n, max(scores)
```

Dựa trên hàm tìm kiếm tham số tối ưu, các hàm tiếp theo được tạo để huấn luyện và đánh giá hiệu suất của mô hình.

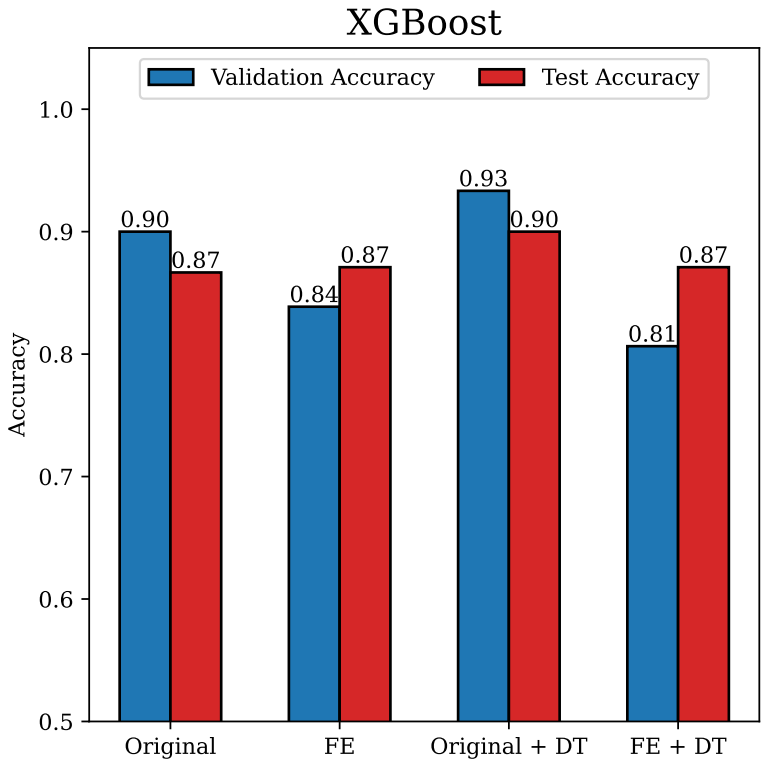
Hàm train/val và hàm test

```
def evaluate_val_xgb(X_train, y_train, X_val, y_val,
                     n_estimators_range=range(50, 501, 50), cv_splits=3,
                     learning_rate=0.1, max_depth=5, subsample=1.0,
                     colsample_bytree=1.0, use_gpu=False):
    print('Tìm n_estimators tối ưu cho XGBoost...')
    xgb_model, best_n, cv_acc = find_optimal_xgb(
        X_train, y_train, n_estimators_range=n_estimators_range,
        cv_splits=cv_splits, learning_rate=learning_rate,
        max_depth=max_depth, subsample=subsample, use_gpu=use_gpu
    )

    val_pred = xgb_model.predict(X_val)
    val_acc = accuracy_score(y_val, val_pred)
    print(f'\nĐộ chính xác XGBoost trên tập validation: {val_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_val, val_pred))
    return xgb_model, val_acc, {'n_estimators': best_n}

def evaluate_test_xgb(xgb_model, X_test, y_test):
    test_pred = xgb_model.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    print(f'\nĐộ chính xác XGBoost trên tập test: {test_acc:.4f}')
    print('Classification Report:')
    print(classification_report(y_test, test_pred))
    return test_acc
```

Sau khi đã có các hàm cần thiết, tiến hành huấn luyện và đánh giá mô hình trên bốn bộ dữ liệu khác nhau.

Huấn luyện và đánh giá trên các tập dữ liệu

```python
# XGB on Original Dataset
xgb_model, val_acc, best_params = evaluate_val_xgb(
    X_train, y_train, X_val, y_val
)
test_acc = evaluate_test_xgb(xgb_model, X_test, y_test)

# XGB on Feature Engineering Dataset
xgb_model, val_fe_acc, best_params = evaluate_val_xgb(
    X_fe_train, y_fe_train, X_fe_val, y_fe_val
)
test_fe_acc = evaluate_test_xgb(xgb_model, X_fe_test, y_fe_test)

# XGB on Original DT Dataset
xgb_model, val_dt_acc, best_params = evaluate_val_xgb(
    X_dt_train, y_dt_train, X_dt_val, y_dt_val
)
test_dt_acc = evaluate_test_xgb(xgb_model, X_dt_test, y_dt_test)

# XGB on Feature Engineering DT Dataset
xgb_model, val_fe_dt_acc, best_params = evaluate_val_xgb(
    X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
)
test_fe_dt_acc = evaluate_test_xgb(xgb_model, X_fe_dt_test, y_fe_dt_test)
```



Hình 14: Hiệu suất của mô hình XGBoost trên 4 bộ dữ liệu.

II.7. Tổng hợp kết quả

Trong phần này, ta lập một bảng ghi nhận độ chính xác của các phương pháp trên tập val và test của từng bộ dữ liệu. Đây là một cách làm thường thấy trong các bài báo khoa học, cung cấp cho người đọc một góc nhìn toàn diện về khả năng của tất cả các phương pháp học máy được xem xét, trong trường hợp này là thuộc xuyên suốt 2 phần của dự án chẩn đoán bệnh tim của chúng ta. Theo đó, các phương pháp được đặt lần lượt theo thứ tự trình bày trong project. Có 4 tập dữ liệu bao gồm bộ gốc (Origin) và các bộ với các kiểu phối hợp tiền xử lý khác nhau như Feature Engineering (FE) và chọn lọc đặc trưng với Decision Tree (DT). Hiệu suất của các mô hình được đo bằng Accuracy (độ chính xác).

Bảng 2: Bảng tổng hợp các kết quả thực nghiệm trên cả 2 dự án (Part 1 và 2). Trong đó, FE là viết tắt của "Feature Engineering"; DT là viết tắt của "Decision Tree".

Model	**Val**				**Test**			
	Origin	FE	Origin+DT	FE+DT	Origin	FE	Origin+DT	FE+DT
Naive Bayes	.90	.90	.93	.93	.84	.84	.84	.84
KNN	.90	.90	**.97**	.87	.84	.84	.87	.84
KMeans	.70	.80	.83	.63	**.87**	**.87**	.84	.77
Decision Tree	.93	**.93**	.93	**.93**	.81	.81	.81	.81
Ensemble (KNN,DT,NB)	.87	**.93**	.90	.87	.84	.90	.84	.84
Random Forest	**.97**	.90	**.97**	.84	.90	.87	**.93**	.84
AdaBoost	**.97**	.81	**.97**	.81	**.97**	.84	**.93**	.84
Gradient Boosting	.87	.81	.87	.84	.83	.81	**.93**	.81
XGBoost	.90	.84	.93	.81	.87	.87	.90	**.87**

Dựa vào Bảng 2, ta nhận thấy độ chính xác cao nhất có thể đạt được trong bài là 97%. Các phương pháp thuộc nhóm Ensemble Learning, cụ thể với Random Forest và AdaBoost, cho kết quả khả quan nhất trên cả 4 kiểu dataset. Bên cạnh đó, việc áp dụng tiền xử lý dữ liệu nhìn chung có thể giúp các phương pháp học máy cải thiện độ chính xác một cách tương đối. Điều này phần nào phản ánh cho chúng ta tầm quan trọng của việc áp dụng tiền xử lý dữ liệu cũng như độ hiệu quả của các phương pháp có ứng dụng Ensemble Learning.

II.8. Cài đặt UI và Deploy

Để nâng cao tính hoàn thiện và tính trực quan của dự án, việc xây dựng một bản demo với giao diện đơn giản là cần thiết. Bản demo đóng vai trò minh hoạ trực tiếp cho hiệu quả của mô hình, đồng thời hỗ trợ quá trình kiểm thử. Qua đó, kết quả nghiên cứu không chỉ dừng lại ở mức lý thuyết, mà còn được cụ thể hoá thành một sản phẩm có thể sử dụng được.

Google Colab không được thiết kế để lưu trữ ứng dụng web, nên để chạy và truy cập giao diện Streamlit từ Colab ta cần tạo một đường hầm (tunnel) an toàn. Quy trình gồm: (1) viết mã ứng dụng vào app.py, (2) cài các phụ thuộc và khởi chạy Streamlit trên Colab, (3) dùng Cloudflared để mở tunnel từ máy chủ Colab ra ngoài, qua đó nhận một URL công khai để truy cập giao diện từ trình duyệt.

Bước 1: Tạo file ứng dụng app.py

Đầu tiên, chúng ta cần viết toàn bộ code của ứng dụng Streamlit vào một file Python. Trong môi trường Colab, ta sử dụng "magic command" `%%writefile` để thực hiện việc này. Cell đầu tiên này sẽ tạo ra file `app.py` và chứa toàn bộ logic của ứng dụng, từ tải dữ liệu, huấn luyện mô hình cho đến hiển thị giao diện.

Khai báo thư viện và Cấu hình

app.py - Phần 1

```
%%writefile app.py
import streamlit as st
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

st.set_page_config(page_title='Heart Disease Prediction', layout='wide')
URL = 'https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/
                                        processed.cleveland.data'
COLUMNS = ['age','sex','cp','trestbps','chol','fbs','restecg',
           'thalach','exang','oldpeak','slope','ca','thal','target']
NUMERIC = ['age','trestbps','chol','thalach','oldpeak']
CATEGORICAL = ['sex','cp','fbs','restecg','exang','slope','ca','thal']
```

- **Giải thích:** Import các thư viện cần thiết, bao gồm `GridSearchCV` để tự động tối ưu tham số. Chúng ta cũng cấu hình tiêu đề, icon cho trang web và định nghĩa các hằng số chứa URL dữ liệu và danh sách các cột.

Tải dữ liệu, Tối ưu hóa và Huấn luyện Mô hình

app.py - Phần 2

```
@st.cache_resource
def find_best_model():
    df = pd.read_csv(URL, header=None, names=COLUMNS, na_values='?')
    df = df.dropna()
    df['target'] = (df['target'] > 0).astype(int)

    X = df[NUMERIC + CATEGORICAL]
    y = df['target']

    num_pipe = Pipeline([('imputer', SimpleImputer(strategy='median'))])
    cat_pipe = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
```

```
        ('ohe', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
    ])

    preprocessor = ColumnTransformer([
        ('num', num_pipe, NUMERIC),
        ('cat', cat_pipe, CATEGORICAL)
    ])

    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', DecisionTreeClassifier(random_state=42))
    ])

    param_grid = {'classifier__max_depth': range(3, 11)}

    grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-
                                                1)
    grid_search.fit(X, y)

    return grid_search

model = find_best_model()
```

- **Giải thích:** Đây là phần logic cốt lõi. Hàm `find_best_model` thực hiện toàn bộ quy trình:
  - Tải dữ liệu từ URL và làm sạch cơ bản.
  - Xây dựng một `Pipeline` hoàn chỉnh để xử lý dữ liệu số, dữ liệu phân loại và đưa vào mô hình Cây quyết định.
  - Định nghĩa một không gian tham số (`param_grid`) để tìm kiếm độ sâu (`max_depth`) tốt nhất cho cây.
  - Sử dụng `GridSearchCV` để thực hiện kiểm định chéo (cross-validation) 5-fold, tự động tìm ra tham số tối ưu và huấn luyện lại mô hình tốt nhất trên toàn bộ dữ liệu.
  - Decorator `@st.cache_resource` đảm bảo toàn bộ quá trình tốn nhiều thời gian này chỉ chạy một lần duy nhất khi ứng dụng khởi động. Kết quả (mô hình đã được tối ưu) sẽ được lưu vào cache và tái sử dụng, giúp ứng dụng phản hồi ngay lập tức.

Xây dựng Giao diện Người dùng (UI)

app.py - Phần 3

```
st.title('Heart Disease Prediction App')
st.write('Enter patient data in the sidebar. The model will predict the likelihood of
                                            heart disease.')
st.sidebar.info(f'Optimal model depth found: {model.best_params_['classifier__max_depth
                                            ']}')

with st.sidebar.form('input_form'):
    st.header('Patient Parameters')
    age = st.slider('Age', 20, 80, 50)
```

```
    sex = st.selectbox('Gender', [('Male', 1), ('Female', 0)], format_func=lambda x: x[0])[1]
    cp = st.selectbox('Chest Pain Type', [('Typical Angina', 1), ('Atypical Angina', 2), ('Non-anginal Pain', 3), ('Asymptomatic', 4)], format_func=lambda x: x[0])[1]
    trestbps = st.slider('Resting Blood Pressure (mmHg)', 90, 200, 120)
    chol = st.slider('Serum Cholesterol (mg/dl)', 120, 570, 240)
    fbs = st.selectbox('Fasting Blood Sugar > 120 mg/dl', [('False', 0), ('True', 1)], format_func=lambda x: x[0])[1]
    restecg = st.selectbox('Resting ECG', [('Normal', 0), ('ST-T Abnormality', 1), ('LV Hypertrophy', 2)], format_func=lambda x: x[0])[1]
    thalach = st.slider('Max Heart Rate Achieved', 70, 210, 150)
    exang = st.selectbox('Exercise-induced Angina', [('No', 0), ('Yes', 1)], format_func=lambda x: x[0])[1]
    oldpeak = st.slider('ST Depression', 0.0, 6.2, 1.0, step=0.1)
    slope = st.selectbox('Slope of Peak ST', [('Upsloping', 1), ('Flat', 2), ('Downsloping', 3)], format_func=lambda x: x[0])[1]
    ca = st.slider('Major Vessels colored by Fluoroscopy', 0, 3, 0)
    thal = st.selectbox('Thalassemia', [('Normal', 3), ('Fixed Defect', 6), ('Reversible Defect', 7)], format_func=lambda x: x[0])[1]
    submitted = st.form_submit_button('Predict')
```

- **Giải thích:** Đoạn code này tạo tiêu đề và các thành phần tương tác. `st.sidebar.form` tạo một form ở thanh bên, gom tất cả các widget nhập liệu lại. Một điểm mới là `st.sidebar.info`, nó sẽ hiển thị độ sâu tối ưu mà `GridSearchCV` đã tìm được, giúp người dùng biết mô hình đang chạy với cấu hình nào.

Xử lý, Dự đoán và Trực quan hóa

app.py - Phần 4

```
if submitted:
    input_data = pd.DataFrame([{'age': age, 'sex': sex, 'cp': cp, 'trestbps': trestbps, 'chol': chol, 'fbs': fbs, 'restecg': restecg, 'thalach': thalach, 'exang': exang, 'oldpeak': oldpeak, 'slope': slope, 'ca': ca, 'thal': thal}])
    proba = model.predict_proba(input_data)[0]
    prob_disease = proba[1]

    st.subheader('Prediction Result')
    if prob_disease > 0.5:
        st.error(f'**Disease Likely** - Probability: {prob_disease:.2\%}')
    else:
        st.success(f'**Disease Unlikely** - Probability of No Disease: {(1-prob_disease):.2\%}')

st.subheader('Optimal Model Decision Tree Visualization')
```

```
st.write('This chart shows the decision rules of the best model found via Cross-Validation.')

best_pipeline = model.best_estimator_
preprocessor = best_pipeline.named_steps['preprocessor']
classifier = best_pipeline.named_steps['classifier']

try:
    feature_names = preprocessor.get_feature_names_out()
    fig, ax = plt.subplots(figsize=(25, 12))
    plot_tree(classifier, feature_names=feature_names, class_names=['No Disease', 'Disease'], filled=True, rounded=True, ax=ax, fontsize=10)
    st.pyplot(fig)
except Exception as e:
    st.warning(f'Could not display the tree visualization. Error: {e}')
```

- **Giải thích:** Khi người dùng nhấn "Predict", code sẽ thu thập dữ liệu, tạo DataFrame và đưa vào mô hình để dự đoán xác suất. Kết quả được hiển thị trực quan. Phần cuối cùng trực quan hóa cây quyết định. Lưu ý rằng `model` lúc này là một đối tượng `GridSearchCV`, vì vậy chúng ta cần truy cập `model.best_estimator_` để lấy ra pipeline tốt nhất. Từ đó, ta trích xuất bộ phân loại (classifier) và vẽ cây quyết định tương ứng với mô hình đã được tối ưu.

Bước 2: Cài đặt môi trường

Chúng ta cần cài đặt Streamlit và công cụ **Cloudflared** để tạo đường hầm.

Cài đặt Streamlit

```
!pip -q install streamlit
```

- **Giải thích:** Lệnh này sử dụng `pip` để cài đặt thư viện `streamlit`.

Tải và cấp quyền cho Cloudflared

```
!wget -q https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-linux-amd64 -O cloudflared
!chmod +x cloudflared
```

- **Giải thích:** Tải về file thực thi của Cloudflared và cấp quyền để có thể chạy nó.

Bước 3: Khởi chạy ứng dụng

Đây là bước cuối cùng để khởi động server và tạo link truy cập công khai.

Chạy Streamlit Server dưới nền

```
!streamlit run app.py \
  --server.port 8501 \
  --server.address 0.0.0.0 \
  --server.headless true \
  --server.enableCORS false \
  --server.enableXsrfProtection false &>/content/logs.txt &
```

- **Giải thích:** Lệnh này khởi chạy ứng dụng `app.py` trên cổng 8501 và chạy ngầm dưới nền, cho phép chúng ta thực hiện lệnh tiếp theo.

Tạo đường hầm với Cloudflared

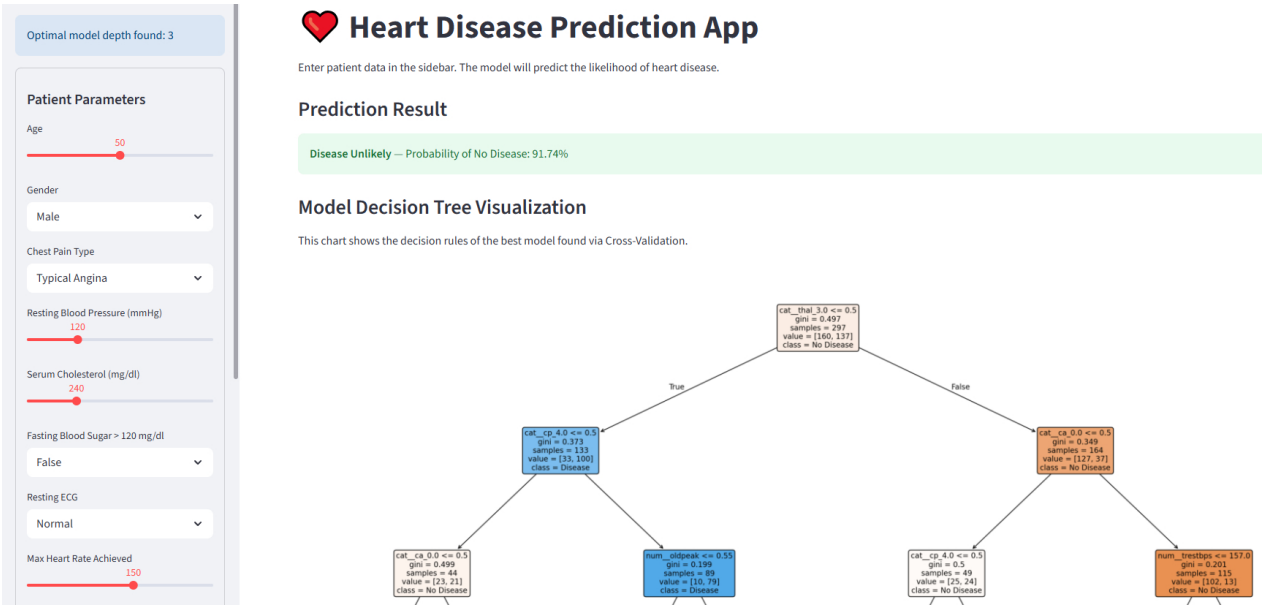
```
!./cloudflared tunnel --url http://localhost:8501 --no-autoupdate
```

- **Giải thích:** Lệnh này tạo một đường hầm công khai đến ứng dụng đang chạy cục bộ trên cổng 8501 của server Colab.

Kết quả và Cách truy cập

Sau khi chạy cell cuối cùng, bạn sẽ thấy một loạt các dòng output. Hãy tìm một đường link có dạng **`https://....trycloudflare.com`**. Đây chính là địa chỉ công khai của ứng dụng Streamlit của bạn.

Hãy nhấp vào đường link để mở ứng dụng trong một tab mới!



Hình 15: Giao diện demo.

III. Câu hỏi trắc nghiệm

1. Xét đoạn code chia dữ liệu sau:

```
X_train, X_temp, y_train, y_temp = train_test_split(
    X_all, y_all, test_size=0.2, stratify=y_all, random_state=42)
```

Tham số `stratify=y_all` có vai trò quan trọng nhất là gì trong bài toán này?

(a) Đảm bảo các đặc trưng trong tập huấn luyện và kiểm thử có cùng phân phối.

(b) Giúp quá trình chia dữ liệu nhanh hơn bằng cách lấy mẫu có thứ tự.

(c) Đảm bảo tỷ lệ người có bệnh và không có bệnh được giữ nguyên trong các tập dữ liệu sau khi chia, tránh thiên vị do mất cân bằng nhãn.

(d) Xáo trộn dữ liệu ngẫu nhiên để loại bỏ sự phụ thuộc về thứ tự.

2. Cho các pipeline xử lý dữ liệu sau:

```
cat_proc = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('scaler', MinMaxScaler())
])
num_proc = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

```

Sau khi áp dụng `fit_transform`, một cột phân loại (ví dụ: `sex`) và một cột số (ví dụ: `age`) sẽ có đặc điểm gì?

(a) Cột `sex` có giá trị trong khoảng [0, 1] và cột `age` có trung bình 0, độ lệch chuẩn 1.

(b) Cả hai cột đều có trung bình 0 và độ lệch chuẩn 1.

(c) Cột `sex` được điền bằng giá trị mode, cột `age` được điền bằng giá trị median, không có scaling.

(d) Cả hai cột đều có giá trị trong khoảng [0, 1].

3. So sánh Hình 4 (Mutual Information) và Hình 5 (Decision Tree), sự khác biệt chính trong cách hai phương pháp đánh giá đặc trưng là gì?

(a) Cả hai đều xếp hạng `thal`, `cp`, `ca` cao nhất, cho thấy chúng hoạt động theo cùng một nguyên lý.

(b) Mutual Information đánh giá đặc trưng đã mã hóa one-hot (ví dụ: `thal_3.0`) dựa trên sự phụ thuộc thống kê với nhãn, trong khi Decision Tree đánh giá đặc trưng gốc (ví dụ: `thal`) dựa trên khả năng giảm impurity tại các điểm chia.

(c) Decision Tree ưu tiên các đặc trưng số như `oldpeak` và `chol` hơn Mutual Information.

(d) Mutual Information chỉ áp dụng được trên dữ liệu FE, còn Decision Tree dùng được cho cả dữ liệu gốc.

4. Hàm `find_optimal_rf` cố định `max_depth=5`. Việc giới hạn độ sâu cây trong khi tìm kiếm `n_estimators` có thể gây ra hệ quả tiềm tàng nào?

```
def find_optimal_rf(
    # ...
    max_depth=5, min_samples_split=2, min_samples_leaf=1,
    max_features='sqrt', bootstrap=True, class_weight=None
): # ...

```

(a) Mô hình chắc chắn sẽ bị quá khớp (overfitting).

(b) Giúp mô hình hội tụ nhanh và luôn tìm được cấu hình tốt nhất toàn cục.

(c) Có thể khiến mô hình bị chưa khớp (underfitting) vì mỗi cây không đủ phức tạp để nắm bắt các mối quan hệ phức tạp, ngay cả khi có nhiều cây.

(d) Tham số này không ảnh hưởng đến hiệu suất, chỉ có `n_estimators` là quan trọng.

5. Trong đoạn code XGBoost sau, tham số `tree_method` đang áp dụng kỹ thuật tối ưu hóa hiệu năng nào?

```
xgb = XGBClassifier(
    # ...
    tree_method='gpu_hist' if use_gpu else 'hist', verbosity=0
)

```

(a) Kỹ thuật điều chuẩn L2 để kiểm soát trọng số của mô hình.

(b) Thuật toán xây dựng cây dựa trên histogram giúp tăng tốc độ tìm điểm chia, và có tùy chọn tăng tốc bằng GPU.

(c) Cơ chế tự động xử lý giá trị thiếu mà không cần imputation.

(d) Kỹ thuật cắt tỉa cây (pruning) sau khi xây dựng hoàn chỉnh.

6. Trong code Streamlit, biến `model` được dùng theo hai cách. Tại sao cần truy cập `model.best_estimator_` để vẽ cây, trong khi có thể dùng `model` trực tiếp để dự đoán?

```
# Cách 1: Dự đoán
proba = model.predict_proba(input_data)[0]

# Cách 2: Lấy thông tin để vẽ cây
best_pipeline = model.best_estimator_
```

```
classifier = best_pipeline.named_steps['classifier']
plot_tree(classifier, ...)

```

(a) Đây là quy ước đặt tên, `model` và `model.best_estimator_` thực chất là một.

(b) Lệnh `plot_tree` yêu cầu một pipeline, còn `predict_proba` chỉ cần một mô hình phân loại.

(c) Biến `model` là đối tượng `GridSearchCV`. Các hàm như `predict_proba` được ủy quyền cho mô hình tốt nhất, nhưng để truy cập các thành phần nội tại của pipeline (như classifier), ta phải tường minh lấy ra pipeline tốt nhất qua thuộc tính `.best_estimator_`.

(d) Chỉ cần truy cập `.best_estimator_` trong lần chạy đầu tiên.

7. Trong Gradient Boosting, cơ chế "sửa sai" hoạt động bằng cách nào, và "sai số" được mô hình kế tiếp học dựa trên đại lượng gì?

(a) Mô hình kế tiếp học trên các mẫu bị phân loại sai, với trọng số của các mẫu này được tăng lên.

(b) "Sai số" là phần chênh lệch giữa nhãn thực tế và dự đoán hiện tại, mô hình kế tiếp được huấn luyện để dự đoán chính các phần dư này.

(c) "Sai số" được tính bằng gradient bậc hai của hàm mất mát và dùng để cập nhật các nút lá.

(d) Mỗi mô hình mới được huấn luyện trên một mẫu bootstrap, và "sai số" được tổng hợp bằng cách lấy trung bình.

8. Đâu là sự khác biệt lý thuyết cốt lõi trong cơ chế "sửa sai" tuần tự giữa AdaBoost và Gradient Boosting được mô tả trong tài liệu?

(a) Cả hai thuật toán đều tính toán phần dư (residual), nhưng AdaBoost chỉ áp dụng cho bài toán phân loại nhị phân.

(b) AdaBoost điều chỉnh trọng số của các mẫu huấn luyện, buộc mô hình sau tập trung vào các mẫu bị phân loại sai; trong khi Gradient Boosting huấn luyện mô hình sau để trực tiếp dự đoán phần dư của mô hình tổng hợp trước đó.

(c) AdaBoost luôn sử dụng các cây quyết định rất nông (stumps) làm mô hình yếu, trong khi Gradient Boosting phải sử dụng cây sâu hơn để tính toán phần dư.

(d) AdaBoost sử dụng learning rate để kết hợp các mô hình yếu, trong khi Gradient Boosting thì không.

9. Việc dùng `fit_transform` cho tập train nhưng chỉ `transform` cho tập val/test trong đoạn code sau nhằm tránh vấn đề cốt lõi nào?

```
X_raw_train = raw_pipeline.fit_transform(X_train, y_train)
X_raw_val = raw_pipeline.transform(X_val)
```

```
X_raw_test = raw_pipeline.transform(X_test)

```

(a) Lỗi tương thích phiên bản của Scikit-learn.

(b) Giảm thời gian tính toán vì `transform` nhanh hơn.

(c) Rò rỉ dữ liệu, bằng cách đảm bảo các tham số (ví dụ: trung vị, tham số scaling) chỉ được học từ tập huấn luyện.

(d) Ngăn ngừa overfitting bằng cách giảm số đặc trưng của tập val/test.

10. Lý do chính đằng sau việc tạo ra đặc trưng tương tác như `chol/age` là gì?

```
if {'chol','age'} <= set(df.columns):
    df['chol_per_age'] = df['chol']/df['age']

```

(a) Để chuẩn hóa giá trị cholesterol về cùng thang đo với tuổi.

(b) Giả định rằng tác động của cholesterol lên nguy cơ bệnh tim có thể thay đổi theo độ tuổi (ví dụ, cholesterol cao ở người trẻ sẽ nguy hiểm hơn).

(c) Đây là kỹ thuật bắt buộc để xử lý giá trị ngoại lai trong cột `chol` và `age`.

(d) Để giảm số chiều dữ liệu bằng cách kết hợp hai đặc trưng.

IV. Tài liệu tham khảo

[1] A. Janosi et al., *Heart disease*, UCI Machine Learning Repository, 1989. DOI: 10.24432/C52P4X. [Online]. Available: https://doi.org/10.24432/C52P4X.

[2] R. E. Schapire, "A brief introduction to boosting", in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden: Morgan Kaufmann, 1999, pp. 1401–1406.

[3] T. Chen et al., "XGBoost: A scalable tree boosting system", in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA: ACM, Aug. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. [Online]. Available: https://doi.org/10.1145/2939672.2939785.

Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại đây.
2. **Hint:** Các file code gợi ý có thể được tải tại đây.
3. **Demo:** Chương trình demo cho nội dung trong bài có thể được truy cập tại đây.
4. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây.
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	- Hiểu bối cảnh bệnh tim mạch và tầm quan trọng của chẩn đoán sớm. - Nắm được thông tin về bộ dữ liệu Cleveland Heart Disease (303 mẫu, 13 đặc trưng). - Biết các đặc trưng y tế chính (age, sex, cp, trestbps, chol, thalach, oldpeak, slope, ca, thal, ...).	- Trình bày được lý do chọn bộ dữ liệu Cleveland và cách nhị phân hoá nhãn mục tiêu. - Nhận biết và giải thích ý nghĩa các đặc trưng trong dữ liệu. - Hiểu mối liên hệ giữa dữ liệu y tế và mô hình học máy; nêu được cảnh báo phi y khoa.
II.	- Kiến thức về chọn đặc trưng bằng Decision Tree importance và Mutual Information. - Hiểu cấu trúc 4 biến thể dữ liệu: Original, FE, Original+DT(Top-K), FE+DT(Top-K).	- Giải thích vì sao chọn K đặc trưng và tác động đến bias/variance. - Phân biệt cách xử lý biến phân loại (giữ nguyên, OHE, scaling). - Trình bày đúng pipeline fit/transform cho train/val/test, tránh data leakage.
III.	- Kiến thức về các mô hình ensemble nâng cao: Random Forest, AdaBoost, Gradient Boosting, XGBoost. - Kiến thức về baseline Deep Learning (MLP) trên dữ liệu bảng nhỏ. - Hiểu sự khác biệt Bagging vs Boosting và cơ chế học sửa sai.	- Giải thích cơ chế của RF (giảm phương sai) và Boosting (giảm bias). - Nêu nguyên lý trọng số lỗi trong AdaBoost, residual trong Gradient Boosting. - Trình bày ưu/nhược điểm của XGBoost; khi nào nên dùng GPU/hist. - Nhận diện khi nào MLP hữu ích hoặc dễ overfit trên bộ nhỏ.
IV.	- Kiến thức triển khai bằng scikit-learn/xgboost: pipeline, GridSearchCV, cross-validation. - Biết sử dụng StratifiedKFold và chia train/val/test cố định để so sánh công bằng. - Biết đánh giá bằng Accuracy, Precision, Recall, F1 (ROC-AUC nếu có), và trực quan hoá.	- Viết và hiểu các hàm `evaluate_val_*`, `evaluate_test_*`. - Giải thích mục đích của StratifiedKFold trong tìm tham số tối ưu. - Đọc và phân tích được chênh lệch val/test (overfit/underfit). - So sánh hiệu quả giữa các biến thể dữ liệu và mô hình khác nhau.

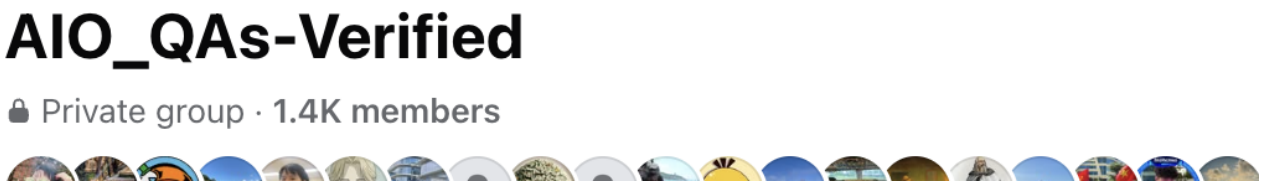
V.	- Kiến thức về tái lập thí nghiệm: cố định seed, quản lý dữ liệu nhất quán. - Nhận diện và tránh data leakage. - Biết tổ chức code: module hóa, ghi log, tách rõ phần build dữ liệu và train mô hình.	- Cho thấy chạy lại với cùng seed cho kết quả tương đương. - Chỉ ra các điểm dễ gây rò rỉ dữ liệu và cách khắc phục. - Code rõ ràng, có chú thích; quản lý đường dẫn/thư mục đúng chuẩn.
VI.	- Kiến thức về xây dựng giao diện với Streamlit. - Biết cách deploy trên Colab thông qua cloudflared. - Biết đóng gói pipeline (preprocessing + model) để dự đoán trực tiếp từ input.	- Ứng dụng chạy được: nhập tham số, trả dự đoán và xác suất. - Vẽ cây quyết định từ best estimator; hiển thị kết quả dễ hiểu. - Xử lý đúng vấn đề version (sklearn, OHE sparse_output).

6. **Đề xuất phương án cải tiến project:** Trong quá trình triển khai, project đã xây dựng được pipeline hoàn chỉnh từ xử lý dữ liệu, huấn luyện mô hình cho đến ứng dụng demo. Tuy nhiên, vẫn có nhiều hướng mở rộng và tinh chỉnh để nâng cao chất lượng kết quả cũng như khả năng ứng dụng. Các gợi ý dưới đây tập trung vào bốn khía cạnh chính: dữ liệu, mô hình, quản lý thí nghiệm và sản phẩm demo.

   - **Cải thiện dữ liệu:**
     - Thử nghiệm nhiều phương pháp chuẩn hoá khác nhau như: StandardScaler, MinMaxScaler, RobustScaler..., và so sánh ảnh hưởng đến hiệu suất.
     - Sử dụng `VarianceThreshold` để loại bỏ đặc trưng ít biến thiên; áp dụng ma trận tương quan để loại bỏ đặc trưng trùng lặp.
     - Tạo thêm đặc trưng mới từ dữ liệu sẵn có (feature engineering) tương tự như một cách làm ví dụ được trình bày ở nội dung bài viết phần 1.
     - Ứng dụng `SelectKBest` hoặc PolynomialFeatures để khai thác đặc trưng bậc cao.
   - **Đa dạng hóa mô hình:**
     - Bổ sung Logistic Regression để so sánh với các mô hình đã triển khai.
     - Thử nhiều giá trị k trong KNN hoặc sử dụng weighted KNN để đánh giá độ ổn định.
     - Áp dụng Support Vector Machine (SVM) với kernel tuyến tính hoặc RBF để kiểm chứng khả năng phân tách dữ liệu.
   - **Quản lý thí nghiệm:**
     - Đặt seed cố định để đảm bảo tính tái lập kết quả.
     - Ghi chú và lưu lại cấu hình, tham số, kết quả nhằm thuận tiện cho việc so sánh.
     - Sử dụng Optuna hoặc GridSearchCV mở rộng để tự động hóa quá trình tối ưu tham số.
   - **Cải thiện sản phẩm demo:**
     - Thiết kế giao diện nhập liệu trực quan hơn: dùng `st.slider` cho biến số (tuổi, cholesterol) và `st.selectbox` cho biến phân loại (giới tính, loại đau ngực).

- Hiển thị xác suất dự đoán bên cạnh nhãn phân loại để làm rõ mức độ rủi ro.
- Bổ sung phần giải thích đặc trưng quan trọng (feature importance) từ mô hình.
- Lưu lại lịch sử dự đoán trong ứng dụng hoặc triển khai bằng Docker/Streamlit Cloud để tăng tính khả chuyển.

7. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại đây. Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 tiếng.



Hình 16: Hình ảnh group facebook AIO Q&A.