Trường Đại học Khoa học Tự Nhiên - VNUHCM
Khoa Công nghệ Thông tin

---o0o---

Báo cáo bài tập cuối kì

Môn học : **Nhập môn học máy**

GVHD : **TS. Bùi Tiến Lên**

GV. Lê Nhật Nam

GV. Võ Nhật Tân

GV. Nguyễn Thanh Tình

Lớp : **22KHMT1**

Sinh viên : **Nguyễn Gia Huy – 22127154**

Tp. Hồ Chí Minh, Tháng 1 năm 2025

Mục lục

Mục lục **2**
1. Giới thiệu mô hình **3**
1.1. Mô tả bài toán ***3***
1.2. Lí do chọn mô hình ***5***
1.3. Cách thức hoạt động của mô hình ***7***
1.4. Ưu và nhược điểm của mô hình. ***9***
2. Báo cáo kết quả thực nghiệm **11**
2.1. Quá trình khám phá và tiền xử lý dữ liệu ***11***
2.2. Quá trình xây dựng mô hình ***14***
2.3. Quá trình huấn luyện ***17***
3. Giải thích cấu trúc thư mục code **21**
3.1. utils ***21***
3.1.1. exploring.py *21*
3.1.2. preprocessing.py *22*
3.2. models ***23***
3.2.1. base.py *23*
3.2.2. tree.py *23*
3.2.3. random_forest.py *23*
3.2.4. lightgbm.py *24*
3.2.5. tuning/random_forest_tuning.py *24*
3. 2.6. tuning/lightgbm_tuning.py *24*
3.3. data ***25***
3.4. output ***25***
3.5. logs và html ***25***

Ghi chú:

1. Google Drive: xem tại đây
2. GitHub: xem tại đây

1. Giới thiệu mô hình

1.1. Mô tả bài toán

Bài toán "Mobile Price Classification" là một bài toán phân loại nhiều lớp (multiclass classification), trong đó mục tiêu là dự đoán giá bán (price_range) của điện thoại dựa trên các đặc trưng kỹ thuật như **bộ nhớ RAM**, **dung lượng pin**, **số lượng lõi CPU**, và các tính năng khác như **hỗ trợ 4G/3G**, **Bluetooth**, và **màn hình cảm ứng**.

Tập dữ liệu:

- **Dữ liệu đầu vào (features):** Gồm 20 đặc trưng, bao gồm cả dữ liệu rời rạc (binary), liên tục (numerical), và đặc trưng kỹ thuật số.
- **Dữ liệu đầu ra (target):** Một biến phân loại với 4 nhãn:
    - 0: Low cost
    - 1: Medium cost
    - 2: High cost
    - 3: Very high cost

Đặc điểm bài toán:

- **Loại bài toán:** Phân loại.
- **Mục tiêu:** Tối ưu hóa mô hình để đạt được độ chính xác cao nhất khi dự đoán price_range.
- **Thách thức:**
    - Cần xử lý và chuẩn hóa dữ liệu đầu vào.
    - Đảm bảo mô hình không bị overfitting trên tập huấn luyện.
    - Đánh giá hiệu suất mô hình trên tập kiểm tra.

Các mô hình đề xuất để giải bài toán:

1. **Logistic Regression (Softmax):** Mô hình tuyến tính, phù hợp với dữ liệu được chuẩn hóa và phân phối đồng đều.
2. **Decision Tree:** Mô hình đơn giản, mạnh mẽ, dễ dàng giải thích kết quả.
3. **Random Forest:** Tăng cường Decision Tree bằng cách tổng hợp kết quả của nhiều cây, giảm nguy cơ overfitting.
4. **K-Nearest Neighbors (KNN):** Phù hợp với dữ liệu có khoảng cách rõ ràng giữa các lớp.
5. **Naive Bayes:** Hiệu quả với dữ liệu có mối quan hệ độc lập giữa các đặc trưng.

6. **Support Vector Machine:** Hiệu quả cao với dữ liệu nhỏ, hỗ trợ kernel trick để giải quyết dữ liệu phi tuyến tính.
7. **Gradient Boosting Models (e.g., LightGBM):** Hiệu suất cao, kháng nhiễu, tối ưu cho dữ liệu tabular.
8. **Neural Network:** Tốt với dữ liệu phi tuyến tính và phức tạp.

1.2. Lí do chọn mô hình

Dựa trên yêu cầu bài toán và tính chất dữ liệu, hai mô hình được chọn là **Random Forest** và **LightGBM**.

1.2.1. Bảng so sánh các mô hình:

Mô hình	Ưu điểm	Nhược điểm
Logistic Regression	Đơn giản, dễ hiểu, hiệu quả với dữ liệu tuyến tính.	Hiệu suất thấp khi dữ liệu phi tuyến tính hoặc phức tạp.
Decision Tree	Dễ triển khai, diễn giải rõ ràng, hiệu quả trên dữ liệu nhỏ.	Dễ overfitting, kém hiệu quả với dữ liệu nhiễu.
Random Forest	***Kháng nhiễu tốt, giảm overfitting, hiệu quả cao mà không cần nhiều tinh chỉnh.***	***Kích thước mô hình lớn, thời gian dự đoán chậm hơn Decision Tree.***
KNN	Đơn giản, không cần huấn luyện.	Chậm với dữ liệu lớn, nhạy cảm với dữ liệu nhiễu.
Naive Bayes	Nhanh, dễ triển khai.	Hiệu suất kém nếu các đặc trưng không độc lập.
Support Vector Machine	Tối ưu biên phân cách giữa các lớp. Hiệu quả khi số lượng mẫu ít.	Chậm khi xử lý dữ liệu lớn. Cần chuẩn hóa dữ liệu trước.
Gradient Boosting (XGBoost, CatBoost, LightGBM, ...)	***Hiệu suất cao trên dữ liệu phức tạp. Kháng nhiễu tốt, khả năng tổng quát cao. Điều chỉnh linh hoạt các siêu tham số (learning rate, max depth, n_estimators).***	***Chậm hơn so với Random Forest và Decision Tree. Cần nhiều công sức tinh chỉnh.***
*Neural Network	Mạnh mẽ, linh hoạt, học được mối quan hệ phi tuyến tính phức tạp.	Yêu cầu tài nguyên tính toán, dễ overfitting nếu không được regular hóa tốt.

Neural Network là một mô hình tốt, tuy nhiên sau khi thử nghiệm thì em cảm thấy mô hình không phù hợp với bài toán này. Lí do thầy có thể xem kết quả của quá trình huấn luyện tại đây.

1.2.2. Lý do chọn Random Forest:

- **Kháng nhiễu tốt:** Random Forest kết hợp nhiều Decision Tree độc lập để tăng độ ổn định và giảm overfitting.
- **Hiệu quả mà không cần tinh chỉnh nhiều:** Random Forest hoạt động tốt trên dữ liệu tabular mà không yêu cầu nhiều tối ưu hóa.
- **Dễ diễn giải:** Các cây thành phần giúp phân tích tầm quan trọng của từng đặc trưng.

1.2.3. Lý do chọn LightGBM (thuộc họ Gradient Boosting)

- **Tốc độ nhanh:** LightGBM sử dụng thuật toán "leaf-wise" để tăng tốc độ huấn luyện mà không làm giảm hiệu suất.
- **Khả năng mở rộng:** Hiệu quả với dữ liệu lớn và nhiều đặc trưng.
- **Hiệu suất cao:** LightGBM thường đạt độ chính xác vượt trội so với các mô hình khác trên dữ liệu tabular.

1.2.4. So sánh giữa các mô hình thuộc Gradient Boosting:

Độ phức tạp:

- **XGBoost:** Phức tạp nhất vì cần chuẩn bị dữ liệu và tinh chỉnh siêu tham số kỹ lưỡng.
- **LightGBM:** Dễ triển khai nhất do tốc độ nhanh và ít phụ thuộc vào cấu hình.
- **CatBoost:** Đơn giản khi dữ liệu có categorical features, không cần mã hóa, nhưng chậm hơn nếu không tối ưu hóa.

Khả năng diễn giải:

- **XGBoost và LightGBM:** Khả năng diễn giải tương đương, hỗ trợ tính feature importance và các công cụ như SHAP, LIME.
- **CatBoost:** Có lợi thế hơn khi xử lý categorical features và tích hợp tốt hơn với SHAP để diễn giải.

Tóm tắt:

- **XGBoost:** Phù hợp khi cần hiệu suất ổn định, dữ liệu đã được xử lý kỹ, hoặc cần tùy chỉnh sâu.
- **LightGBM:** Tốt nhất khi làm việc với dữ liệu lớn, tốc độ quan trọng, hoặc có nhiều đặc trưng.
- **CatBoost:** Lựa chọn lý tưởng khi có nhiều categorical features hoặc không muốn mất công mã hóa thủ công.

1.3. Cách thức hoạt động của mô hình

1.3.1. Random Forest

Random Forest là một mô hình học máy thuộc nhóm thuật toán Ensemble Learning, kết hợp nhiều Decision Tree để cải thiện hiệu suất tổng thể. Cách thức hoạt động chính:

1. **Bootstrap Sampling:**
    - Tạo nhiều tập dữ liệu huấn luyện từ tập ban đầu bằng cách lấy mẫu ngẫu nhiên có hoàn lại (Bagging).
    - Mỗi Decision Tree được xây dựng trên một tập dữ liệu khác nhau.
2. **Xây dựng Decision Tree:**
    - Tại mỗi node, Random Forest chỉ xem xét một tập con ngẫu nhiên của các đặc trưng để tìm điểm chia tốt nhất, giảm sự tương quan giữa các cây.
    - Tiêu chí chia: Thường sử dụng Gini Impurity hoặc Entropy.
3. **Tích hợp dự đoán:**
    - Đối với bài toán phân loại, Random Forest sử dụng cơ chế bỏ phiếu đa số (majority voting) trên các cây thành phần để đưa ra kết quả cuối cùng.

Các siêu tham số tối ưu:

- **n_estimators** (số lượng cây): Tăng số lượng cây sẽ cải thiện độ chính xác nhưng tăng thời gian tính toán.
- **max_depth** (độ sâu tối đa): Hạn chế độ sâu để tránh overfitting.
- **max_features** (số lượng đặc trưng được xem xét tại mỗi node): Giảm sự phụ thuộc giữa các cây.
- **min_samples_split** (số lượng mẫu tối thiểu để chia): Điều chỉnh để kiểm soát overfitting.

1.3.2. LightGBM (Light Gradient Boosting Machine)

LightGBM là thuật toán Gradient Boosting hiệu quả, sử dụng cây quyết định làm mô hình cơ sở và tối ưu hóa quá trình học bằng kỹ thuật Gradient Descent. Cách thức hoạt động chính:

1. **Tree Growth (Leaf-wise Growth):**
    - Tạo cây quyết định dựa trên kỹ thuật chia theo chiều sâu (leaf-wise) thay vì cấp độ (level-wise), giúp giảm lỗi nhanh hơn.
2. **Gradient Boosting:**
    - Mỗi cây mới được xây dựng để sửa lỗi của cây trước đó bằng cách giảm gradient của hàm mất mát.

3. **Weighted Histogram:**
    - Sử dụng kỹ thuật lượng tử hóa histogram để giảm kích thước dữ liệu, tăng tốc độ xử lý.

Các siêu tham số tối ưu:

- **learning_rate**: Tốc độ học; giảm giá trị này giúp cải thiện độ chính xác nhưng yêu cầu nhiều cây hơn.
- **num_leaves**: Số lượng lá tối đa trên mỗi cây; tăng giá trị này có thể gây overfitting.
- **max_depth**: Hạn chế độ sâu cây để tránh overfitting.
- **n_estimators**: Số lượng cây trong mô hình.
- **min_data_in_leaf**: Số lượng mẫu tối thiểu trong một lá; kiểm soát overfitting.

1.4. Ưu và nhược điểm của mô hình.

Tiêu chí	Random Forest	LightGBM
Ưu điểm		
Kháng nhiễu tốt	Giảm overfitting nhờ cơ chế Bagging và chọn tập con đặc trưng ngẫu nhiên.	Hoạt động tốt ngay cả trên dữ liệu không đồng nhất hoặc nhiều nhiễu.
Hiệu quả trên dữ liệu nhỏ	Không cần chuẩn hóa hoặc mã hóa dữ liệu đặc trưng.	Xử lý nhanh và hiệu quả trên dữ liệu lớn và nhiều đặc trưng.
Diễn giải rõ ràng	Có thể phân tích tầm quan trọng của đặc trưng bằng các cây quyết định thành phần.	Cung cấp tầm quan trọng đặc trưng, nhưng khó diễn giải hơn Random Forest.
Nhược điểm		
Hiệu suất kém với dữ liệu lớn	Cần nhiều bộ nhớ và thời gian để huấn luyện trên dữ liệu lớn.	Nhạy cảm với dữ liệu nhiễu hoặc các giá trị ngoại lai.
Không linh hoạt với bài toán	Không hiệu quả khi cần tích hợp xử lý dữ liệu categorical phức tạp.	Yêu cầu chuẩn hóa dữ liệu kỹ lưỡng và tinh chỉnh nhiều siêu tham số.
Kích thước mô hình lớn	Sử dụng nhiều cây dẫn đến bộ nhớ lớn.	Đôi khi quá khớp khi số lượng lá (num_leaves) hoặc chiều sâu cây quá cao.

Giải pháp cho Random Forest:

1. **Giảm kích thước mô hình:** Giới hạn số lượng cây (n_estimators) và độ sâu cây (max_depth) để giảm bộ nhớ sử dụng.
2. **Khắc phục hiệu suất kém trên dữ liệu lớn:** Sử dụng kỹ thuật song song hóa (parallelization) để tăng tốc huấn luyện.
3. **Tăng tính linh hoạt:** Kết hợp với các kỹ thuật xử lý trước dữ liệu categorical (one-hot encoding hoặc label encoding).

Giải pháp cho LightGBM:

1. **Kiểm soát overfitting:** Giảm số lượng lá (num_leaves) và sử dụng regularization

(lambda_l1, lambda_l2).

2. **Xử lý dữ liệu nhiễu:** Chuẩn hóa dữ liệu đầu vào và loại bỏ giá trị ngoại lai trước khi huấn luyện.
3. **Tối ưu tốc độ:** Điều chỉnh max_bin và sử dụng phiên bản GPU của LightGBM để tăng tốc độ tính toán.

2. Báo cáo kết quả thực nghiệm

2.1. Quá trình khám phá và tiền xử lý dữ liệu

Mục tiêu: Tiến hành khám phá dữ liệu và xử lý các vấn đề liên quan đến dữ liệu để chuẩn bị cho quá trình xây dựng và huấn luyện mô hình.

2.1.1. Đọc và mô tả dữ liệu

Dữ liệu được tải lên từ hai file CSV: **train.csv** và **test.csv**. Sau đó, các thao tác sau được thực hiện:

- **Loại bỏ cột id**: Đây là cột không liên quan đến bài toán phân loại.
- **Tóm tắt dữ liệu**:
  - Dùng **summarize_data()** để cung cấp thông tin cơ bản (số dòng, số cột, kiểu dữ liệu).
  - Kiểm tra các giá trị bị thiếu và thống kê số liệu của từng đặc trưng.

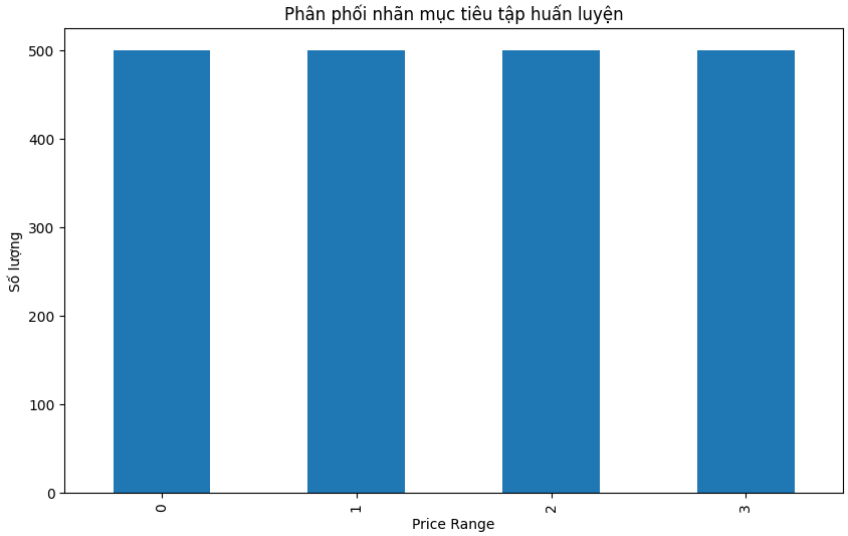
Kết quả:

- Tập huấn luyện có số dòng, số cột phù hợp.
- Không có giá trị bị thiếu ở cả hai tập dữ liệu.

2.1.2. Trực quan hóa phân phối nhãn

Sử dụng biểu đồ histogram để quan sát phân phối của biến mục tiêu (**price_range**) trên cả tập huấn luyện và kiểm tra.

Kết quả: Phân phối nhãn tương đối đồng đều trên tất cả các lớp (0-3), giúp đảm bảo bài toán phân loại không bị mất cân bằng dữ liệu nghiêm trọng.



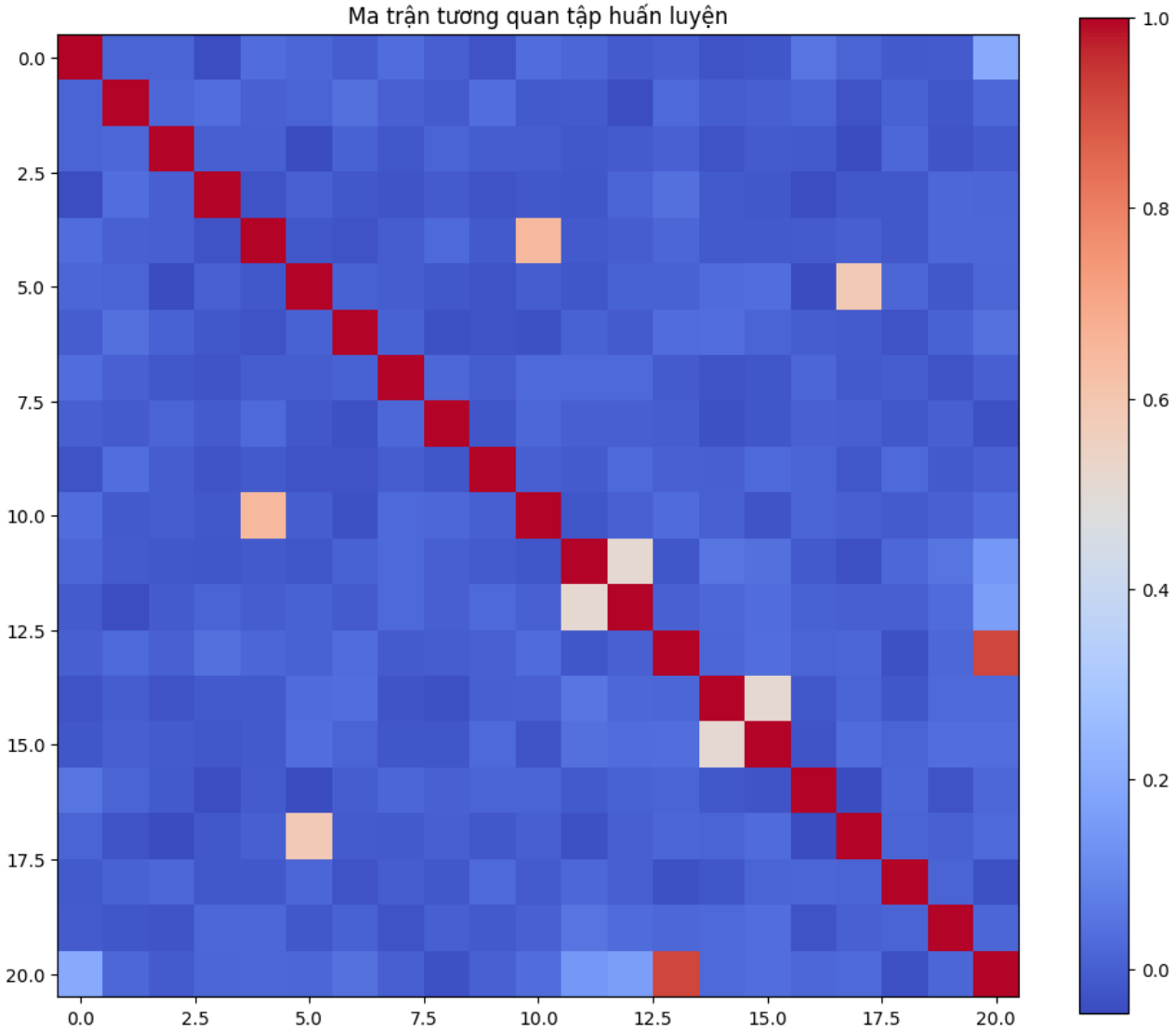
2.1.3. Trực quan hóa ma trận tương quan

Ma trận tương quan được vẽ để phân tích mối liên hệ giữa các đặc trưng:

- Các đặc trưng có hệ số tương quan cao (gần hoặc trên 0.9) có thể gây dư thừa.

Kết quả:

- Một số cặp đặc trưng có tương quan cao được xác định và xử lý trong bước tiếp theo.



2.1.4. Xử lý loại bỏ các đặc trưng có tương quan cao

Dùng hàm **identify_high_correlation()** để tự động xác định và loại bỏ các đặc trưng có tương quan cao với ngưỡng 0.9.

Kết quả: Không có đặc trưng nào có tương quan cao cần loại bỏ.

2.1.5. Xử lý giá trị thiếu

Mặc dù không có giá trị bị thiếu, hàm handle_missing_values() vẫn được thực hiện để đảm bảo tính toàn vẹn dữ liệu.

Kết quả: Không có giá trị bị thiếu nào cần xử lý.

2.1.6. Feature Engineering

Tạo hai đặc trưng mới:

- aspect_ratio = px_height / px_width
- power_to_weight = battery_power / mobile_wt

Kết quả: Đặc trưng mới cung cấp thêm thông tin về hình dạng màn hình và hiệu năng năng lượng.

2.1.7. Chuẩn hóa dữ liệu liên tục

Áp dụng Min-Max Scaling để chuẩn hóa các đặc trưng liên tục, đưa chúng về khoảng [0, 1].

Kết quả: Tất cả các đặc trưng liên tục được chuẩn hóa thành công.

2.1.8. Mã hóa các đặc trưng phân loại

Dùng kỹ thuật One-Hot Encoding cho các đặc trưng phân loại (**n_scores**, ...), giúp biến đổi chúng thành dạng số.

Kết quả: Tăng cường khả năng sử dụng của dữ liệu phân loại trong mô hình.

2.1.9. Chia dữ liệu

Dữ liệu được chia thành:

- **Tập huấn luyện** (X_train, y_train).
- **Tập kiểm tra** (X_test, y_test).
- **Tập validation** (X_val, y_val) để tối ưu hóa siêu tham số.

Kết quả:

- Tập huấn luyện và validation sẵn sàng để huấn luyện.
- Dữ liệu được lưu lại vào các file **train_processed.csv** và **test_processed.csv**.

Tổng kết

Quá trình tiền xử lý đã đảm bảo tính sẵn sàng và chất lượng của dữ liệu. Dữ liệu đã được chuẩn hóa, giảm đa cộng tuyến và bổ sung các đặc trưng mới hữu ích.

2.2. Quá trình xây dựng mô hình

Trong phần này, hai mô hình học máy đã được xây dựng từ đầu: **Random Forest** và **LightGBM**. Cả hai đều dựa trên cấu trúc **DecisionTree**, được tối ưu hóa để đạt hiệu suất tốt nhất trên bài toán phân loại "Mobile Price Classification".

2.2.1. Kiến trúc lớp ModelTrainer

- **Mục đích:** Cung cấp một giao diện cơ sở cho các mô hình, bao gồm các phương thức train, predict, evaluate, và khả năng lưu/tải trọng số mô hình.
- **Chức năng chính:**
  - **Huấn luyện (train)**: Các lớp con triển khai chi tiết logic huấn luyện.
  - **Dự đoán (predict)**: Trả về dự đoán của mô hình.
  - **Đánh giá (evaluate)**: Tính toán độ chính xác trên tập kiểm tra dựa trên dự đoánbase.

2.2.2. Mô hình Random Forest

- **Cấu trúc:** Sử dụng **RandomForestModel**, mỗi cây thành phần trong Random Forest được xây dựng bằng **DecisionTreeClassifier**.
- **Quy trình xây dựng:**
  1. **Khởi tạo mô hình:**
     - Các siêu tham số chính: **n_estimators, max_depth, min_samples_split, min_samples_leaf,** và **criterion**.
  2. **Huấn luyện:**
     - Sử dụng phương pháp **Bagging**, mỗi cây được huấn luyện trên một tập con ngẫu nhiên (**Bootstrap sampling**).
     - Cây được xây dựng với tiêu chí chia dựa trên **Gini Impurity** hoặc **Entropy**.
  3. **Dự đoán:**
     - Dự đoán của từng cây được tổ hợp bằng phương pháp **Majority Voting** để đưa ra kết quả cuối cùng.
- **Ưu điểm của cấu trúc:**
  - Khả năng giảm overfitting nhờ tổ hợp nhiều cây.
  - Kháng nhiễu và hiệu quả trên dữ liệu tabularrandom_forest.

2.2.3. Mô hình LightGBM

- **Cấu trúc:** Sử dụng **LightGBMModel**, mỗi cây được xây dựng dựa trên thuật toán **Gradient Boosting** với **DecisionTreeRegressor**.
- **Quy trình xây dựng:**
  1. **Khởi tạo mô hình:** Các siêu tham số chính: n**_estimators, learning_rate, max_depth, lambda_l1,** và **lambda_l2**.
  2. **Huấn luyện:**
     - Tính **residuals** (phần dư) từ nhãn thực tế và dự đoán của mô hình.
     - Huấn luyện cây mới để dự đoán residuals.
     - Cập nhật trọng số cây bằng cách sử dụng regularization (**lambda_l1, lambda_l2**) và tỷ lệ học (**learning_rate**).
  3. **Dự đoán:** Kết hợp dự đoán của tất cả các cây với trọng số tương ứng để đưa ra kết quả.
- **Ưu điểm của cấu trúc:**
  - Tối ưu hóa hiệu suất qua Gradient Boosting.
  - Xử lý tốt dữ liệu nhiều chiều và phức tạp.

2.2.4. Triển khai các lớp cây quyết định

- **DecisionTreeClassifier:** Được sử dụng trong Random Forest.
  - Chia nhánh dựa trên **Gini Impurity** hoặc **Entropy**.
  - Tạo cây bằng phương pháp đệ quy đến khi đạt ngưỡng dừng (độ sâu tối đa hoặc số mẫu tối thiểu).
- **DecisionTreeRegressor:** Được sử dụng trong LightGBM.
  - Sử dụng **Mean Squared Error (MSE)** để đánh giá chất lượng chia nhánh.
  - Tối ưu hóa residuals để cải thiện dự đoán.

---

5. Quy trình tối ưu hóa

- **Random Forest:** Thử nghiệm với các giá trị khác nhau cho n_estimators, max_depth, và các siêu tham số khác để tìm cấu hình tối ưu.
- **LightGBM:** Điều chỉnh learning_rate và số lượng cây (n_estimators) để đạt cân bằng giữa hiệu suất và tốc độ.

6. Tổng kết

Cả hai mô hình đều được triển khai từ đầu với cấu trúc linh hoạt, đảm bảo khả năng tùy chỉnh và hiệu suất cao. Việc sử dụng tổ hợp cây trong Random Forest và Gradient Boosting trong LightGBM giúp tối ưu hóa kết quả trên bài toán phân loại.

2.3. Quá trình huấn luyện

2.3.1. Tổng quan

Quá trình huấn luyện được thực hiện trên hai mô hình:

- **Random Forest**: Một mô hình tổ hợp cây quyết định sử dụng kỹ thuật Bagging.
- **LightGBM**: Một thuật toán Gradient Boosting tối ưu hóa quá trình học thông qua residuals.

2.3.2. Mô hình Random Forest

1. **Tối ưu hóa siêu tham số:**
    - Sử dụng lưới tham số (**param_grid**) bao gồm:
        - **n_estimators**: [10, 20, 30, 40, 50, ...]
        - **max_depth**: [3, 5, 7, ..., None]
        - **min_samples_split**: [2, 5, 10, ...]
        - **min_samples_leaf**: [1, 2, 4, ...]
    - Tổng cộng có **180** tổ hợp siêu tham số được thử nghiệm. Tổng thời gian huấn luyện nếu không can thiệp vào CPU, tuần tự là hơn **6 giờ** trong điều kiện lí tưởng (hệ điều hành tự động cấp **60-70%** CPU chạy tuần tự). Thực tế trong điều kiện không tốt thì thời gian có thể gấp **5-6** lần (hệ điều hành chỉ cấp **10%** CPU mặc dù không có tác vụ nào đang chạy).
2. **Quy trình:**
    - Sử dụng **ProcessPoolExecutor** để thực hiện tối ưu hóa trên 18 luồng, tận dụng toàn bộ tài nguyên CPU.
    - Kết quả của mỗi tổ hợp được ghi vào file CSV để lưu trữ và phân tích.
    - Độ chính xác tốt nhất trên tập validation đạt **90%**. (ứng với **train_acc = 97%**)
3. **Kết quả huấn luyện:** (toàn bộ kết quả huấn luyện xem tại đây)

n_estimators	max_depth	min_split	min _leaf	train_acc	val_acc	training_time (s)
20	***7***	***2***	***2***	***0.97***	***0.9***	***84.77***
20	***7***	***5***	***2***	***0.97***	***0.9***	***83.38***
40	7	2	2	0.97	0.89	178.74
40	7	5	2	0.97	0.89	163.19

40	7	10	2	0.97	0.89	170.38
40	None	2	2	0.99	0.89	190.44
40	None	5	2	0.99	0.89	192.78
40	None	10	2	0.98	0.89	278.18
20	7	2	4	0.96	0.89	81.21
20	7	5	4	0.96	0.89	81.2

- 2 bộ siêu tham số đầu tiên là bộ siêu tham số tối ưu của mô hình với tập dữ liệu
- Độ chính xác trên tập kiểm tra: **23.7%** (trung bình khoảng **25%**)
- Mô hình Random Forest hoạt động hiệu quả với cấu hình tối ưu, cân bằng tốt giữa hiệu suất và khả năng tổng quát hóa. Tuy nhiên kết quả trên tập kiểm tra chưa được tốt, bị overfitting, em đã cố gắng nhưng vẫn không cải thiện được.

2.3.3. Mô hình LightGBM

1. **Tối ưu hóa siêu tham số:**
   - Sử dụng lưới tham số (**param_grid**) bao gồm:
     - **n_estimators**: [10, 20, 30, 40, 50, ...]
     - **learning_rate**: [0.01, 0.05, 0.1]
     - **max_depth**: [3, 5, 7, 9, ..., None]
     - **lambda_l1**: [0.0, 0.001, 0.01, 0.1, 1.0]
     - **lambda_l2**: [0.0, 0.001, 0.01, 0.1, 1.0]
   - Tổng cộng có **216** tổ hợp siêu tham số được thử nghiệm. Tổng thời gian huấn luyện nếu không can thiệp vào CPU, tuần tự là hơn **7.5 giờ** trong điều kiện lí tưởng (hệ điều hành tự động cấp **60-70%** CPU chạy tuần tự). Thực tế trong điều kiện không tốt thì thời gian có thể gấp **5-6** lần (hệ điều hành chỉ cấp **10%** CPU mặc dù không có tác vụ nào đang chạy).
2. **Quy trình:**
   - Sử dụng phương pháp tương tự **Random Forest** với **ProcessPoolExecutor** và ghi kết quả vào file CSV
   - Độ chính xác tốt nhất trên tập validation đạt **91%**. (ứng với **train_acc = 97%**)

3. **Kết quả huấn luyện:** (toàn bộ kết quả huấn luyện xem tại đây)

n_estimators	learning rate	max depth	l1 lambda	l2 lambda	train accuracy	val accuracy	training time
50	0.1	3	0	0.01	0.97	0.91	138.01
50	0.1	3	0	0	0.97	0.91	138.41
50	0.1	3	0	0	0.97	0.91	139.01
50	0.1	3	0	0	0.97	0.91	138.56
50	0.1	3	0.01	0	0.97	0.91	137.81
50	0.1	3	0	0.01	0.96	0.91	138.51
50	0.1	3	0.01	0	0.97	0.91	138.15
50	0.1	3	0	0	0.96	0.91	138.82
50	0.1	3	0.01	0.01	0.97	0.9	138.45
50	0.1	5	0	0	1.00	0.9	147.03

- 8 bộ siêu tham số đầu tiên đều là bộ siêu tham số tối ưu của mô hình với tập dữ liệu cho ra kết quả độ chính xác trên tập validation cao nhất **91%** với thời gian chạy gần như tương đương nhau.
- Độ chính xác trên tập kiểm tra: **23.7%**
- Mô hình LightGBM đạt hiệu suất vượt trội nhờ tối ưu hóa quá trình học residuals và giảm thiểu overfitting.

2.3.4. So sánh hiệu suất

Mô hình	Validation Accuracy	Test Accuracy
Random Forest	**90%**	**23.7%**
LightGBM	**91%**	**23.7%**

2.3.5. Tổng kết

- **Random Forest**: Phù hợp với dữ liệu tabular có cấu trúc đơn giản, dễ triển khai và diễn giải.
- **LightGBM**: Đạt hiệu suất cao hơn một ít, đặc biệt trên tập validation tổng thể, nhờ thuật toán Gradient Boosting và khả năng tối ưu hóa residuals.

Cả hai mô hình đều được tối ưu hóa và đạt hiệu suất cao trên bài toán phân loại "Mobile Price Classification".

3. Giải thích cấu trúc thư mục code

3.1. utils

Thư mục **utils** chứa các module hỗ trợ quá trình khám phá, tiền xử lý dữ liệu, và chuẩn bị dữ liệu cho các mô hình học máy. Hai file chính trong thư mục này là **exploring.py** và **preprocessing.py**, cung cấp các chức năng cụ thể như sau:

3.1.1. exploring.py

File này hỗ trợ khám phá dữ liệu, trực quan hóa và tạo cái nhìn tổng quan về dữ liệu ban đầu.

1. **load_data(train_path, test_path)**:
    - Chức năng: Đọc dữ liệu từ các file CSV và loại bỏ cột không cần thiết (id).
    - Ý nghĩa: Chuẩn bị dữ liệu thô để sẵn sàng cho các bước tiền xử lý.
2. **summarize_data(df, name="Dataset")**:
    - Chức năng: Tóm tắt thông tin cơ bản (kiểu dữ liệu, số lượng giá trị bị thiếu) và các thống kê mô tả của dữ liệu.
    - Ý nghĩa: Giúp kiểm tra chất lượng dữ liệu ban đầu và phát hiện vấn đề tiềm năng.
3. **visualize_distribution(df, column, title, xlabel, ylabel)**:
    - Chức năng: Vẽ biểu đồ phân phối của một cột dữ liệu cụ thể.
    - Ý nghĩa: Hiển thị trực quan phân phối của nhãn mục tiêu hoặc các đặc trưng quan trọng.
4. **plot_correlation_matrix(df, title)**:
    - Chức năng: Vẽ ma trận tương quan giữa các đặc trưng.
    - Ý nghĩa: Giúp phát hiện các mối quan hệ mạnh hoặc đặc trưng tương quan cao cần loại bỏ.
5. **visualize_continuous_features(df, columns, title_prefix)**:
    - Chức năng: Vẽ biểu đồ histogram cho các đặc trưng liên tục trong danh sách columns.
    - Ý nghĩa: Hiển thị sự phân phối của dữ liệu liên tục, giúp đánh giá tính đồng nhất.

3.1.2. preprocessing.py

File này cung cấp các hàm xử lý dữ liệu để đảm bảo chất lượng dữ liệu trước khi xây dựng mô hình.

1. **train_test_split(data, test_size=0.2, random_state=None)**:
   - Chức năng: Chia tập dữ liệu thành hai phần: huấn luyện và kiểm tra.
   - Ý nghĩa: Cần thiết để đánh giá mô hình trên dữ liệu chưa từng thấy.
2. **identify_high_correlation(df, threshold=0.9)**:
   - Chức năng: Xác định các cặp đặc trưng có hệ số tương quan cao hơn threshold.
   - Ý nghĩa: Giúp loại bỏ các đặc trưng dư thừa, giảm đa cộng tuyến.
3. **handle_missing_values(df)**:
   - Chức năng: Xử lý các giá trị bị thiếu bằng cách thay thế bằng trung vị (median).
   - Ý nghĩa: Đảm bảo dữ liệu không bị gián đoạn do giá trị trống.
4. **fit_transform(data, continuous_features) và transform(data, continuous_features, scaling_params)**:
   - Chức năng: Chuẩn hóa các đặc trưng liên tục bằng Min-Max Scaling.
   - Ý nghĩa: Đưa dữ liệu về cùng thang đo, cải thiện hiệu suất mô hình.
5. **feature_engineering(train_data, test_data)**:
   - Chức năng: Tạo thêm các đặc trưng mới như:
     - aspect_ratio: Tỷ lệ chiều cao/chiều rộng màn hình.
     - power_to_weight: Tỷ lệ giữa công suất pin và trọng lượng thiết bị.
   - Ý nghĩa: Tăng khả năng dự đoán của mô hình bằng cách thêm thông tin quan trọng.
6. **categorical_encoding(train_data, test_data, categorical_features)**:
   - Chức năng: Mã hóa các đặc trưng phân loại bằng One-Hot Encoding.
   - Ý nghĩa: Biến đổi dữ liệu phân loại thành dạng số, phù hợp với các thuật toán học máy.

3.2. models

Thư mục **models** chứa các module liên quan đến việc xây dựng, huấn luyện và dự đoán bằng các mô hình học máy. Mỗi file đại diện cho một phần cụ thể trong cấu trúc của các mô hình hoặc thuật toán. Dưới đây là mô tả chi tiết:

3.2.1. base.py

Ý nghĩa: Lớp cơ sở (ModelTrainer) cung cấp giao diện chung cho tất cả các mô hình.

Chức năng:

- **train và predict**: Được định nghĩa là các phương thức trừu tượng để các lớp con thực thi.
- **evaluate**: Tính độ chính xác dựa trên dự đoán và nhãn thực tế.
- **load_model_weights và save_model_weights**: Hỗ trợ lưu và tải trọng số của mô hình từ file JSON

3.2.2. tree.py

Ý nghĩa: Cung cấp các lớp DecisionTreeClassifier và DecisionTreeRegressor để triển khai thuật toán cây quyết định.

Chức năng:

- **DecisionTreeClassifier**:
  - Dùng tiêu chí **Gini Impurity** hoặc **Entropy** để đánh giá chất lượng phân chia.
  - Xây dựng cây bằng phương pháp đệ quy dựa trên dữ liệu và điều kiện dừng.
  - Hỗ trợ dự đoán bằng cách duyệt qua cây đã xây dựng.
- **DecisionTreeRegressor**:
  - Sử dụng Mean Squared Error (MSE) để đánh giá chất lượng phân chia.
  - Phù hợp cho các bài toán hồi quy với việc tối ưu hóa residuals

---

3.2.3. random_forest.py

Ý nghĩa: Triển khai mô hình **Random Forest** dựa trên DecisionTreeClassifier.

Chức năng:

- **train**:
  - Huấn luyện nhiều cây quyết định trên các tập con dữ liệu được tạo bằng phương pháp **Bootstrap Sampling**.

  - Các cây có thể được cấu hình với các siêu tham số như max_depth, min_samples_split, và criterion.

- **predict**:
  - Kết hợp dự đoán của các cây bằng phương pháp **Majority Voting**, trả về nhãn được dự đoán nhiều nhất.

Ưu điểm: Random Forest giúp giảm overfitting và cải thiện độ chính xác

---

3.2.4. lightgbm.py

Ý nghĩa: Triển khai thuật toán **Light Gradient Boosting Machine (LightGBM)** dựa trên DecisionTreeRegressor.

Chức năng:

- **train**:
  - Tối ưu hóa residuals qua nhiều vòng lặp bằng cách huấn luyện cây quyết định.
  - Sử dụng các siêu tham số như learning_rate, lambda_l1, và lambda_l2 để điều chỉnh regularization.
- **predict**:
  - Tổng hợp dự đoán của tất cả các cây, mỗi cây đóng góp một phần dự đoán dựa trên trọng số (learning_rate).

Ưu điểm: LightGBM giúp giảm thiểu overfitting và hoạt động hiệu quả trên dữ liệu lớn

---

3.2.5. tuning/random_forest_tuning.py

Ý nghĩa: Tối ưu hóa siêu tham số cho mô hình Random Forest.

Chức năng:

- **evaluate_model**: Đánh giá hiệu suất của mô hình với một tổ hợp siêu tham số cụ thể.
- **random_forest_hyperparameter_tuning**:
  - Tìm kiếm siêu tham số tốt nhất bằng cách thử nghiệm trên lưới tham số (grid search).
  - Sử dụng ProcessPoolExecutor để tăng tốc độ thực thi song song.

Kết quả: Ghi nhận hiệu suất và siêu tham số tốt nhất trong file CSV

3. 2.6. tuning/lightgbm_tuning.py

Ý nghĩa: Tối ưu hóa siêu tham số cho mô hình LightGBM.

Chức năng:

- **evaluate_model**: Đánh giá hiệu suất dựa trên các tổ hợp siêu tham số.
- **lightgbm_hyperparameter_tuning**:
    - Thực hiện tìm kiếm siêu tham số tương tự như Random Forest nhưng áp dụng cho LightGBM.
    - Hỗ trợ ghi nhận kết quả vào file để phân tích

3.3. data

- **train.csv**: tập dữ liệu huấn luyện gốc.
- **test.csv**: tập dữ liệu kiểm tra gốc.
- **train_processed.csv**: tập huấn luyện sau khi tiền xử lí (dùng cho sau này nếu dùng nhiều mô hình, không cần tiền xử lí lại mỗi lần chạy mô hình).
- **test_processed.csv**: tập kiểm tra sau khi tiền xử lí.

3.4. output

Thư mục này chứa các kết quả khi huấn luyện mô hình với các siêu tham số khác nhau. Định dạng ban đầu là **.json** nhưng sau đó em thấy làm vậy không hiệu quả vì lưu ở dạng bảng **.csv** sẽ chiếm ít dung lượng và dễ thao tác hơn (**.csv** không cần lưu **best_parameter** vì ta dễ dàng tìm được khi sắp xếp **val_acc** theo chiều giảm dần và chọn ra bộ siêu tham số tối ưu nhất).

Cấu trúc của tên file: {**datetime**}_{**model**}_{**n-hyperparameters**}_{**n-features**}.**csv/json**

Ý nghĩa của từng thành phần trong tên file:

- **datetime**: thời gian bắt đầu lần chạy các siêu tham số huấn luyện mô hình (YYYYmmDD_HHmmSS)
- **model**: tên của mô hình chạy huấn luyện (random_forest, lightgbm)
- **n-hyperparameters**: số lượng bộ tổ hợp siêu tham số trong lần huấn luyện đó.
- **n-features**: số lượng đặc trưng của dữ liệu trong lần huấn luyện (dùng cho mục đích thử với các cách tiền xử lí dữ liệu khác nhau)
- **entropy/gini** (nếu có): dành riêng cho mô hình Random Forest, thử huấn luyện với độ đo gini hoặc entropy

3.5. logs và html

Thư mục **logs** và **html** dùng để lưu trữ các file **.log** và **.html,** dùng cho mục đích làm minh chứng và trực quan hóa kết quả chạy được trong các lần huấn luyện mô hình (vì mỗi lần huấn luyện cần rất nhiều tài nguyên).