TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO
PRACTICE COMPETITION

Nhập môn học máy

Lớp : 22KHDL1

Giảng viên: Bùi Duy Đăng

Huỳnh Lâm Hải Đăng

Trần Trung Kiên

Sinh viên thực hiện: Nguyễn Đăng Nhân – 22127302

Chu Thúy Quỳnh – 22127359

Thành phố Hồ Chí Minh – 2025

MỤC LỤC

I. PHÂN CÔNG:

Công việc	Đăng Nhân	Thúy Quỳnh	Đánh giá
Data preprocessing	x	x	100%
Model design		x	100%
Model training	x	x	100%
Scikit	x	x	100%
Tensorflow		x	100%
LightGBM	x		100%
XGBoost		x	100%
Evaluation	x	x	100%
Report	x		100%

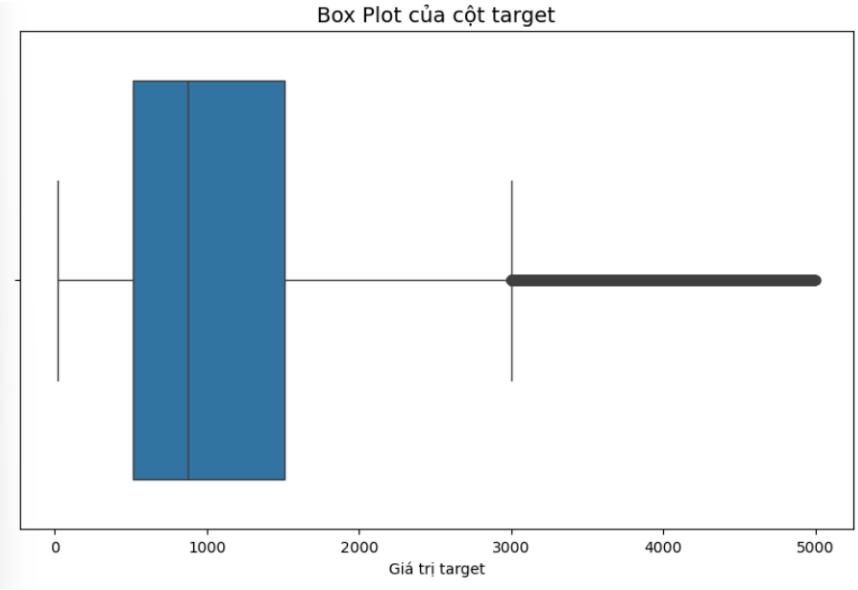
II. GIAI ĐOẠN 1: SKCIT-LEARN

1. Tiền xử lý dữ liệu (Data Preprocessing):

- Bộ dữ liệu 'train.csv' bao gồm 1.200.000 dòng và 21 cột với các đặc trưng có kiểu dữ liệu:

```
id              int64
Age           float64
Gender         object
feature_0     float64
feature_1      object
feature_2     float64
feature_3      object
feature_4      object
feature_5     float64
feature_6      object
feature_7      object
feature_8     float64
feature_9     float64
feature_10    float64
feature_11    float64
feature_12     object
feature_13     object
feature_14     object
feature_15     object
feature_16     object
target        float64
dtype: object
```

- Chúng ta cần phải loại bỏ các giá trị ngoại lệ (outliers) để mô hình không bị overfitting hoặc học sai mối quan hệ thực tế, giúp mô hình học tốt hơn, tránh ảnh hưởng đến việc train.
- Ta tiến hành kiểm tra outlier của cột target.



- Dựa vào hình trên, ta có thể thấy dữ liệu bị lệch phải (right-skewed), đuôi bên phải dài, có nhiều outlier vì vậy giá trị lớn bất thường, không phân phối chuẩn. Các điểm xa vạch phải, cho thấy có nhiều giá trị cực đại cần xử lý. Trung vị gần phía trái của hộp nên đây là phân bố không đối xứng.
- Vì vậy ta sẽ loại bỏ outlier trong cột 'target' trong bộ dữ liệu train dựa vào khoảng tứ phân vị.

```
Q1 = train['target'].quantile(0.25)
Q3 = train['target'].quantile(0.75)
IQR = Q3 - Q1
train = train[(train['target'] >= Q1 - 1.5 * IQR) & (train['target'] <= Q3 + 1.5 * IQR)]
```

- Đối với 'feature_12' là ngày, tháng, năm nên ta sẽ chuyển sang định dạng datetime, sau đó trích xuất year, month, day và xóa cột `feature_12` gốc.
- Cột id trong train không có ý nghĩa huấn luyện nên loại bỏ cột id trong train.

a. Xử lý dữ liệu bị thiếu

- Ta sẽ kiểm tra dữ liệu thiếu và giải quyết bằng cách điền trung vị:
  - Ta sẽ duyệt qua các cột số trong bộ dữ liệu train, nếu có giá trị thiếu thì sẽ được thay thế bằng giá trị trung vị của chính cột đó.

```
numeric_cols = train.select_dtypes(include=[np.number]).columns
for col in numeric_cols:
    train[col].fillna(train[col].median(), inplace=True)
```

- Đối với các cột có giá trị không phải là số thì khi có giá trị thiếu thì sẽ được thay thế bằng giá trị xuất hiện nhiều nhất (mode).

```
non_numeric_cols = train.select_dtypes(exclude=['number']).columns
for col in non_numeric_cols:
    train[col].fillna(train[col].mode()[0], inplace=True)
```

b. Xử lý dữ liệu bị trùng lặp

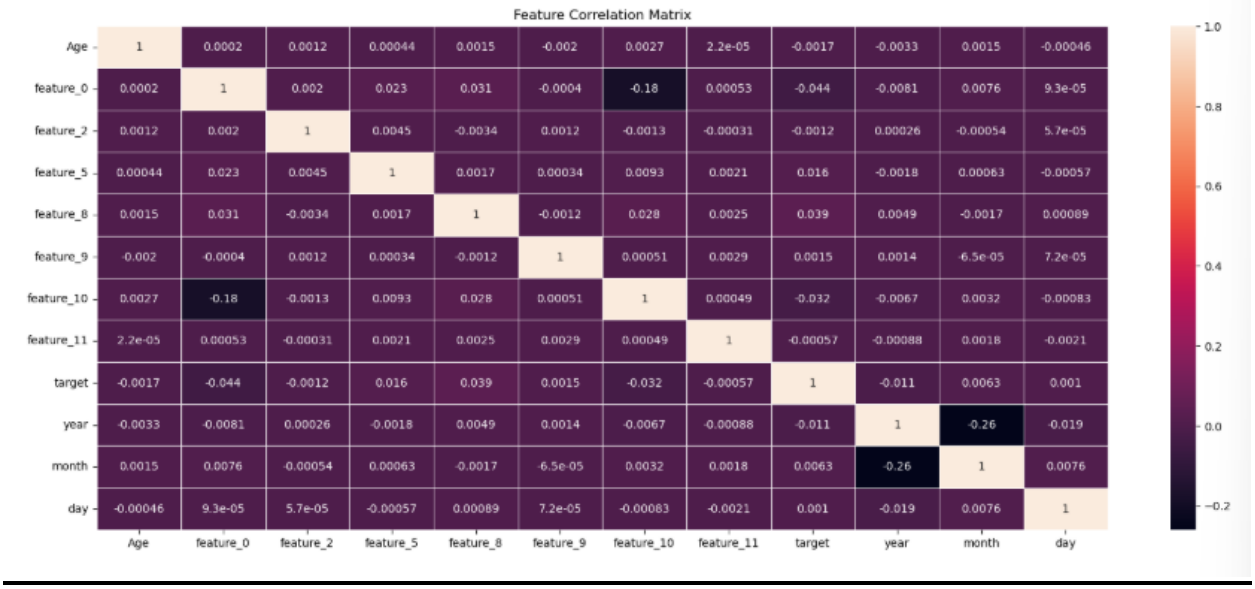
```
train.duplicated().sum()
✓ 1.3s

0
```

- Ta có thể thấy rằng trong bộ dữ liệu không có dữ liệu bị trùng lặp.

c. Kiểm tra đa cộng tuyến:

- Dựa trên heat map và bảng kết quả VIF thì ta có thể thấy rằng:



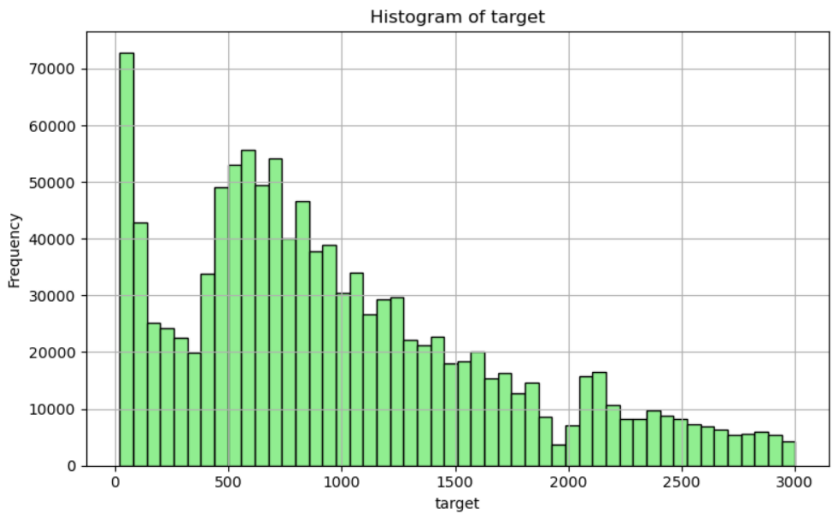
- Không có cặp biến nào có tương quan tuyệt đối cao và tất cả hệ số tương quan đều rất nhỏ, hầu hết gần bằng 0, có một số giá trị lớn hơn về mặt tương đối nhưng không đáng kể để lo ngại về đa cộng tuyến.

	Feature	VIF
0	const	2.019613e+06
1	Age	1.000027e+00
2	feature_0	1.037600e+00
3	feature_2	1.000040e+00
4	feature_5	1.000760e+00
5	feature_8	1.002131e+00
6	feature_9	1.000018e+00
7	feature_10	1.036905e+00
8	feature_11	1.000028e+00
9	year	1.073004e+00
10	month	1.072589e+00
11	day	1.000368e+00

- Tất cả các biến đều có VIF xấp xỉ 1, ngoại trừ const (cao vì là hằng số). Điều này cho thấy không có đa cộng tuyến giữa các biến độc lập.

d. Chuẩn hóa dữ liệu

- Ta sẽ chuyển các giá trị không phải số trong bộ dữ liệu train, chuyển thành các cột nhị phân (0 hoặc 1), sử dụng 'drop_first = True' để tránh hiện tượng đa cộng tuyến khi dùng trong các mô hình hồi quy.
- Sử dụng StandardScaler() từ thư viện sklearn.preprocessing để chuẩn hóa dữ liệu sao cho mỗi đặc trưng trong bộ dữ liệu có trung bình = 0 và độ lệch chuẩn = 1.
- Phân bố dữ liệu của cột target:



- Vì phân phối của 'target' bị lệch phải nên ta chuẩn hóa cột 'target' của bộ dữ liệu train bằng cách biến đổi logarit giúp giảm độ lệch

e. Feature engineering

- Ta cũng sẽ sử dụng LassoCV [1] với cross-validation 5 lần để chọn các đặc trưng quan trọng cho mô hình hồi quy để dự đoán giá trị mục tiêu bằng cách giảm các hệ số của những đặc trung không quan trọng về 0, giúp mô hình tránh bị overfitting và chọn ra tham số điều chỉnh tốt.

f. Chuẩn bị dữ liệu để huấn luyện

- Biến X sẽ chứa tất cả các cột trong bộ dữ liệu train và cột 'target' được loại bỏ để làm dữ liệu đầu vào cho mô hình.
- Biến y sẽ chứa cột 'target' từ bộ dữ liệu train và đây sẽ là giá trị mục tiêu mà mô hình sẽ dự đoán.
- Ta sẽ tiến hành chia bộ dữ liệu thành 3 phần: 70% train - 20% validation - 10% test

2. Xây dựng model

- **Lựa chọn mô hình: MLPRegressor**
    - MLPRegressor có khả năng học các mối quan hệ phi tuyến phức tạp, điều này giúp mô hình linh hoạt hơn.
    - Có khả năng xử lý và học từ dữ liệu phức tạp, giúp cải thiện hiệu suất mô hình khi làm việc với tập dữ liệu có mối quan hệ phức tạp.
    - Đối với bộ dữ liệu chúng ta đang xử lý, ta có thể thấy bộ dữ liệu có nhiều loại đặc trưng, có số lượng mẫu lớn và MLPRegressor có thể giúp mô hình hiệu quả hơn, tập dữ liệu lớn như thế này là điều kiện lý tưởng để tránh overfitting và tận dụng khả năgn học sâu của MLP. Bên cạnh đó, với kiến trúc nhiều lớp và node, MLP có thể tự học các đặc trưng trừu tượng, điều mà các mô hình đơn giản không làm được.
- **Thiết kế kiến trúc**
    - Là một mạng nơ-ron với các lớp ẩn, và cấu hình này được xác định qua tham số hidden_layer_sizes. Sau khi dùng Grid search thử với nhiều mạng nơ-ron khác nhau, kết quả tốt nhất là neural network có 3 lớp ẩn, với lớp ẩn thứ nhất 64 nơ-ron, lớp ẩn thứ hai: 32 nơ-ron. Lớp ẩn thứ ba16 nơ-ron.
    - Hàm kích hoạt ReLU (Rectified Linear Unit) được sử dụng trong các lớp ẩn vì đây là một hàm kích hoạt rất phổ biến và hiệu quả trong việc giải quyết vấn đề vanishing gradient, giúp mạng học nhanh và tốt hơn.
    - Adam là một trong những thuật toán tối ưu phổ biến và hiệu quả nhất, kết hợp giữa hai thuật toán tối ưu khác là Momentum và Adagrad, giúp tăng tốc quá trình huấn luyện và tự động điều chỉnh tốc độ học cho từng tham số.

- Tỷ lệ học ban đầu được chọn là 0.001, đây là một giá trị phổ biến cho Adam. Tuy nhiên, tỷ lệ học sẽ được điều chỉnh sau mỗi chu kỳ để giảm tốc độ học, giúp mô hình hội tụ tốt hơn trong suốt quá trình huấn luyện.
- Early stopping giúp ngừng huấn luyện khi mô hình không cải thiện trên tập xác nhận, giúp tránh overfitting và tiết kiệm tài nguyên tính toán.
- batch_size = 64: ối ưu tốc độ tính toán trên GPU.

- **Huấn luyện:** [2]
  - Trong quá trình huấn luyện, khi learning rate sẽ được giảm dần theo chu kỳ qua các chu kỳ huấn luyện. Mỗi chu kỳ sẽ giảm bằng exponential decay [3]. Vì trong quá trình thử nghiệm để tìm ra các tham số tốt nhất cho model, em nhận thấy rằng có vẻ khi learning rate giảm dần thì model sẽ tốt hơn vì thế em cho learning rate giảm theo từng chu kì.
  - Sử dụng exponential decay cũng giúp mô hình dần chậm lại và tập trung vào việc tinh chỉnh các tham số sau khi đã có sự học cơ bản trong các chu kỳ đầu.
  - Ngoài ra model tốt nhất cũng được lưu lại qua từng chu kì để tránh trường hợp learning rate giảm nhưng hiệu suất mô hình không tăng.

3. Đánh giá:

Chỉ số	Giá trị	Nhận xét
Mean Squared Error (MSE)	575942.33	MSE rất cao cho thấy mô hình có lỗi dự đoán lớn ở một số điểm. Điều này do phân phối target bị lệch. Khi MSE cao, mô hình có thể đã không học tốt đối với các giá trị cực lớn hoặc cực nhỏ.
Root Mean Squared Log Error (RMSLE)	1.0228	Giá trị thấp cho thấy rằng mô hình không bị phạt nặng do dự đoán thấp hơn thực tế (tốt với dữ liệu lệch).
Mean Absolute Percentage Error (MAPE)	177.83%	MAPE rất cao, đồng nghĩa với việc mô hình dự đoán kém với các giá trị nhỏ của target. Do MAPE chia lỗi tuyệt đối cho giá trị thực tế, các giá trị nhỏ sẽ làm MAPE phóng đại nếu dự đoán sai dù chỉ một chút. Điều này phản ánh rằng mô hình không cân bằng tốt giữa các giá trị lớn và nhỏ.
Tổng thời gian huấn luyện	2233.03 giây (~38 phút)	Thời gian huấn luyện khá nhanh với quá trình huấn luyện nhiều chu kỳ.

Bộ nhớ RAM peak	316.74 MB	Mức sử dụng bộ nhớ ở mức thấp và hợp lý, chứng tỏ mô hình có kích thước vừa phải và quá trình huấn luyện không gây áp lực lớn lên hệ thống.

Phân tích lỗi:

- Lỗi dự đoán lớn nhất xảy ra tập trung ở:
  - Các mẫu có giá trị mục tiêu (target) rất lớn → khiến sai số bình phương trung bình (MSE) tăng mạnh.
  - Các mẫu có target rất nhỏ → khiến sai số phần trăm tuyệt đối trung bình (MAPE) tăng cao.
- Nguyên nhân:
  - Việc dùng log rồi trả lại bằng expm1 có thể làm sai lệch các giá trị rất lớn hoặc rất nhỏ khi dự đoán.
  - Phân phối của target bị lệch (không đều) khiến mô hình khó học tốt cho cả hai đầu (giá trị rất lớn và rất nhỏ) cùng lúc.

III. GIAI ĐOẠN 2: TENSORFLOW

1. Tiền xử lý dữ liệu

- Tương tự phase 1.

2. Xây dựng model

- **Lựa chọn mô hình: MLPRegressor (Enhanced Regression Model)**
  - Trong giai đoạn này, chúng em lựa chọn xây dựng một mô hình mạng nơ-ron hồi quy nhiều lớp bằng thư viện TensorFlow và Keras. Mục tiêu của mô hình là dự đoán giá trị đầu ra liên tục. Việc sử dụng mạng nơ-ron sâu (deep learning) sẽ giúp mô hình có khả năng học được các mối quan hệ phức tạp giữa đầu vào và đầu ra, đặc biệt khi dữ liệu không tuyến tính. So với mô hình MLPRegressor của sklearn, mô hình sử dụng TensorFlow cho phép:
    - Tùy chỉnh sâu hơn về kiến trúc mạng.
    - Điều chỉnh chi tiết quá trình huấn luyện như learning rate, batch size, callbacks.
- **Thiết kế kiến trúc**
  - Mô hình được thiết kế theo kiểu Sequential, tức là các lớp được xếp chồng tuần tự từ đầu vào đến đầu ra:
    - Lớp đầu vào nhận vector có kích thước bằng số đặc trưng đầu vào (input_dim).

- Ba lớp ẩn liên tiếp với số neuron lần lượt là 64, 32, và 16 như mô hình ở giai đoạn 1, đều sử dụng hàm kích hoạt ReLU giúp cải thiện tốc độ huấn luyện và giảm hiện tượng vanishing gradient, khiến mạng có thể học các mối quan hệ phi tuyến tốt hơn. [4]
- Sau mỗi lớp ẩn, có thêm Batch Normalization giúp hội tụ nhanh hơn và tốt hơn. [5]
- Lớp đầu ra gồm một neuron duy nhất, không dùng hàm kích hoạt vì đây là bài toán hồi quy.
- Batch_size = 64 và learning rate 0.001 như giai đoạn 1.
- Tối ưu hóa Adam**:** là một bộ tối ưu mạnh mẽ thường được sử dụng trong các mô hình học sâu, tiến nhanh tới mức tối thiểu hơn các phương pháp khác. [6]

- **Huấn luyện:** [2]
  - Tiếp tục dùng ý tưởng giảm learning rate để tìm ra model tốt nhất. Sử dụng ReduceLROnPlateau giảm learning rate khi không thấy cải thiện trong val_loss sau một số epoch nhất định. [7]
  - EarlyStopping giúp dừng sớm khi không còn cải thiện trên tập validation, tránh overfitting.
  - ModelCheckpoint để lưu lại mô hình tốt nhất;

3. Đánh giá:

Chỉ số	Giá trị	Nhận xét
Mean Squared Error (MSE)	585137.49	Giá trị cao, cho thấy mô hình mắc lỗi lớn với một số mẫu. Nguyên nhân chính thường đến từ các target rất lớn, do MSE bình phương sai số nên cực kỳ nhạy với các giá trị dự đoán sai nghiêm trọng.
Root Mean Squared Log Error (RMSLE)	1.0249	Mức lỗi hợp lý và ổn định đối với các mô hình hồi quy sử dụng target log.
Mean Absolute Percentage Error (MAPE)	175.14%	Giá trị rất cao, cho thấy mô hình hoạt động kém với các mẫu có target nhỏ. Vì MAPE chia sai số tuyệt đối cho giá trị thật, nên khi target nhỏ, sai số nhỏ cũng làm tỉ lệ lỗi bị khuếch đại mạnh.
Tổng thời gian huấn luyện	5907.69giây (~98 phút)	Thời gian huấn luyện khá dài, do cấu trúc mạng nhiều tầng và batch normalization.

Bộ nhớ RAM peak	252.08 MB	Mức sử dụng bộ nhớ ở mức thấp và hợp lý, chứng tỏ mô hình có kích thước vừa phải và quá trình huấn luyện không gây áp lực lớn lên hệ thống.

Phân tích lỗi:

- Lỗi dự đoán lớn nhất xảy ra tập trung ở:
  - Các mẫu có giá trị mục tiêu (target) rất lớn → khiến sai số bình phương trung bình (MSE) tăng mạnh.
  - Các mẫu có target rất nhỏ → khiến sai số phần trăm tuyệt đối trung bình (MAPE) tăng cao.
- Nguyên nhân:
  - Việc dùng log rồi trả lại bằng expm1 có thể làm sai lệch các giá trị rất lớn hoặc rất nhỏ khi dự đoán.
  - Phân phối của target bị lệch (không đều) khiến mô hình khó học tốt cho cả hai đầu (giá trị rất lớn và rất nhỏ) cùng lúc.

IV. GIAI ĐOẠN 3: LIGHTGBM VÀ XGBOOST

1. Tiền xử lý dữ liệu

- Tương tự phase 1.

2. Xây dựng model

- **Lựa chọn mô hình: XGBoost**
  - Mô hình XGBoost là một thuật toán tăng cường dần (boosting) nổi tiếng, hiệu quả cao và được sử dụng phổ biến trong các bài toán hồi quy và phân loại.
  - Đây là mô hình học có giám sát, có khả năng xây dựng mô hình dựa trên tổ hợp nhiều cây quyết định (decision tree) để cải thiện độ chính xác và khả năng tổng quát.
  - XGBoost có thể xử lý tốt cả dữ liệu tuyến tính và phi tuyến, rất phù hợp cho các bài toán có nhiều đặc trưng tương tác phức tạp. Mô hình này cũng hỗ trợ nhiều kỹ thuật regularization (L1, L2), giúp hạn chế overfitting, và có khả năng mở rộng tốt cho dữ liệu lớn nhờ tree_method='hist'.
- **Thiết kế kiến trúc:**
  - Sau khi dùng Grid search thử với nhiều siêu tham số khác nhau, kết quả tốt nhất là:

        - objective: 'reg:squarederror': Mục tiêu tối ưu là hàm lỗi bình phương (Mean Squared Error - MSE), dùng cho bài toán hồi quy.
        - learning_rate: 0.001: Tốc độ học (learning rate) rất thấp, giúp mô hình học từ từ, tránh quá mức.
        - max_depth: 10: Độ sâu tối đa của mỗi cây quyết định, giúp kiểm soát độ phức tạp của mô hình.
        - min_child_weight: 5: Số lượng mẫu tối thiểu trong mỗi lá cây quyết định, tránh cây quá khớp (overfitting).
        - subsample: 0.9: Tỷ lệ mẫu được chọn ngẫu nhiên để huấn luyện cây, giúp giảm overfitting.
        - colsample_bytree: 0.85: Tỷ lệ số cột (đặc trưng) được chọn ngẫu nhiên cho mỗi cây, giúp làm giảm overfitting.
        - gamma: 0.2: Giá trị tối thiểu của độ giảm lỗi phải đạt được để phân tách nút cây, giúp điều chỉnh độ phức tạp của mô hình.
        - reg_alpha: 0.1: Lệch L1 (Lasso), giúp điều chỉnh mô hình để giảm overfitting.
        - reg_lambda: 1.0: Lệch L2 (Ridge), giúp điều chỉnh độ phức tạp mô hình và giảm overfitting.
        - max_bin: 512: Số lượng phân binned tối đa cho mỗi cột trong cây, giúp tăng tốc độ huấn luyện**.**
- **Huấn luyện:** [2]
    - Mô hình được huấn luyện qua 5000 vòng lặp, mỗi vòng sẽ điều chỉnh dự đoán để cải thiện kết quả.
    - Hiệu suất được theo dõi liên tục trên cả tập huấn luyện và tập validation.
    - Quá trình huấn luyện sẽ tự động dừng nếu không có sự cải thiện trong 50 vòng liên tiếp.
- **Lựa chọn mô hình: LightGBM**
    - LightGBM sử dụng cây quyết định để mô hình hóa mối quan hệ giữa đặc trưng và mục tiêu, có khả năng khái quát tốt, chống overfitting thông qua nhiều kỹ thuật regularization.
    - LightGBM được tối ưu hóa rất tốt về mặt tốc độ và bộ nhớ, cho phép huấn luyện nhanh hơn nhiều so với các mô hình boosting truyền thống.
    - LightGBM có thể huấn luyện hiệu quả trên tập dữ liệu có số lượng dòng và đặc trưng lớn nhờ max_bin và thuật toán histogram-based.
- **Thiết kế kiến trúc:**
    - Sau khi dùng Grid search thử với nhiều siêu tham số khác nhau, kết quả tốt nhất là:

- boosting_type='gbdt': sử dụng thuật toán gradient boosting truyền thống.
- objective='regression': bài toán hồi quy.
- metric=['l1', 'l2']: sử dụng cả MAE và MSE để đánh giá mô hình.
- learning_rate=0.001: tốc độ học nhỏ giúp quá trình cập nhật mô hình ổn định hơn.
- max_depth=10: kiểm soát độ sâu tối đa của cây, giúp hạn chế học quá mức.
- num_leaves=96: số lượng lá tối đa trong mỗi cây, cân bằng giữa độ chính xác và độ tổng quát.
- feature_fraction=0.9: tỷ lệ đặc trưng được lấy mẫu cho mỗi cây, giúp tránh overfitting.
- bagging_fraction=0.85 và bagging_freq=10: lấy mẫu dữ liệu huấn luyện giúp tăng khả năng tổng quát.
- max_bin=512: tăng độ chi tiết của histogram để huấn luyện chính xác hơn.

- **Huấn luyện:**
  - Khởi tạo huấn luyện với tham số hyper_params đã được thiết lập phù hợp cho bài toán.
  - Theo dõi hiệu suất trên cả hai tập train và val để đánh giá quá trình học.
  - Huấn luyện tối đa 5000 vòng nhưng sẽ dừng sớm nếu không có cải thiện trên tập validation sau 50 vòng
- **So sánh và đánh giá 2 mô hình:**

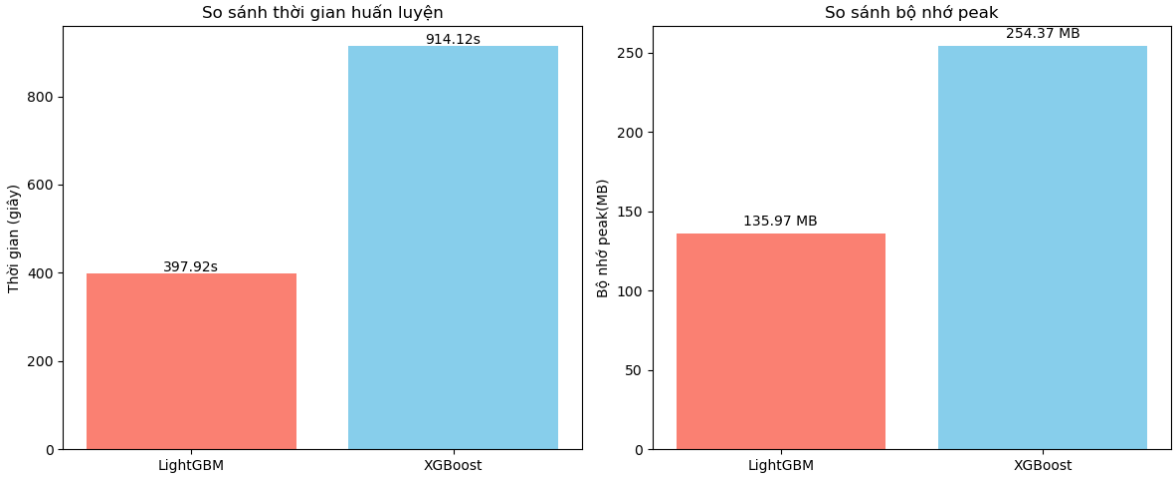
Chỉ số	XGBoots	LightGBM	Nhận xét
Mean Squared Error (MSE)	576535.66	576423.07	Cả hai mô hình có sai số bình phương trung bình rất gần nhau. LightGBM tốt hơn một chút ⇒ dự đoán ít sai số lớn hơn.
Root Mean Squared Log Error (RMSLE)	1.0175	1.0174	RMSLE đo lường độ sai lệch theo tỉ lệ (log-scale). LightGBM tốt hơn rất nhẹ, tức là mô hình học được mối quan hệ tỷ lệ giữa các đặc trưng và target hiệu quả hơn.
Mean Absolute Percentage	174.77%	174.89%	XGBoost nhỉnh hơn chút với MAPE thấp hơn, cho thấy tốt hơn khi dự đoán những giá trị nhỏ, nơi sai số tương đối rất nhạy. Tuy

Error (MAPE)			nhiên, sự khác biệt là không đáng kể.

Phân tích lỗi:

- Lỗi dự đoán lớn nhất xảy ra tập trung ở:
  - Các mẫu có giá trị mục tiêu (target) rất lớn → khiến sai số bình phương trung bình (MSE) tăng mạnh.
  - Các mẫu có target rất nhỏ → khiến sai số phần trăm tuyệt đối trung bình (MAPE) tăng cao.
- Nguyên nhân:
  - Việc dùng log rồi trả lại bằng expm1 có thể làm sai lệch các giá trị rất lớn hoặc rất nhỏ khi dự đoán.
  - Phân phối của target bị lệch (không đều) khiến mô hình khó học tốt cho cả hai đầu (giá trị rất lớn và rất nhỏ) cùng lúc.

So sánh thời gian huấn liệu và bộ nhớ được sử dụng nhiều nhất khi huấn luyện model:



- **Thời gian huấn luyện:**
  - LightGBM chỉ mất khoảng 397.92 giây để hoàn tất huấn luyện.
  - XGBoost tốn tới 914.12 giây, hơn gấp đôi thời gian của LightGBM.
  - LightGBM vượt trội rõ rệt về tốc độ, phù hợp khi cần huấn luyện nhiều lần.
  - Nguyên nhân là do LightGBM sử dụng Leaf-wise tree growth thay vì Level-wise như XGBoost.
- **Bộ nhớ sử dụng:**
  - LightGBM: 135.97 MB

- XGBoost: 254.37 MB
- LightGBM tiết kiệm RAM hơn đáng kể, chỉ sử dụng khoảng 50% tài nguyên bộ nhớ so với XGBoost.
- Điều này quan trọng khi làm việc trên máy cấu hình yếu, hoặc khi dữ liệu có kích thước lớn.

- **Kết luận:**
  - Với hiệu năng tương đương nhưng LightGBM vượt trội về tốc độ và tài nguyên, đây là lựa chọn phù hợp hơn để triển khai thực tế hoặc áp dụng trên pipeline lặp lại nhiều lần.

V. SO SÁNH CÁC MÔ HÌNH:

1. Tổng quan mô hình:

Đặc điểm	MLPRegressor (sklearn)	MLPRegressor (TensorFlow)	XGBoost	LightGBM
Thư viện	sklearn	TensorFlow + Keras	xgboost	lightgbm
Loại mô hình	Neural Network	Neural Network	Gradient Boosting Trees	Gradient Boosting Trees
Ứng dụng	Hồi quy	Hồi quy	Hồi quy	Hồi quy

2. Kết quả đánh giá mô hình:

Chỉ số	MLPRegressor (sklearn)	MLPRegressor (TensorFlow)	XGBoost	LightGBM
MSE	575,942.33	585,137.50	576,535.66	576,423.07
RMSLE	1.0228	1.0249	1.0175	1.0174
MAPE (%)	177.83%	175.14%	174.77%	174.89%

- **Nhận xét:**
  - LightGBM là mô hình nổi bật nhất, khi đồng thời đạt:
    - MSE thấp nhất (tức là ít sai số lớn nghiêm trọng nhất),
    - RMSLE thấp nhất (bắt tốt mối quan hệ tỷ lệ giữa đầu vào và đầu ra),
    - Và MAPE gần như thấp nhất (chỉ kém XGBoost một chút, không đáng kể).
  - XGBoost, mặc dù thua sát nút về MSE và RMSLE, nhưng lại tốt hơn về MAPE, cho thấy nó có khả năng dự đoán chính xác hơn với các giá trị

nhỏ. Tuy nhiên, mức độ chênh lệch giữa hai mô hình là rất nhỏ, và không đủ để đánh đổi nếu xét tới phần chi phí tài nguyên.

- MLPRegressor của sklearn và TensorFlow cho thấy hiệu năng của hai mạng nơ-ron này thua rõ rệt về MAPE (cao hơn tới gần 3% so với XGBoost và LightGBM), cho thấy mạng nơ-ron học không tốt ở đầu nhỏ hoặc lớn của phân phối target. Kết luận: đối với dữ liệu có target lệch, outlier nhiều, cây quyết định mạnh hơn mạng nơ-ron.

3. Tài nguyên và thời gian huấn luyện:

Thông số	MLPRegressor (sklearn)	MLPRegressor (TensorFlow)	XGBoost	LightGBM
Thời gian (giây)	2,233 (~38p)	5,709 (~95p)	914.12(~15p)	397.92(~7p)
RAM peak (MB)	316.74 MB	1,819.51 MB	254.37 MB	135.97 MB

- Bảng trên cho thấy sự vượt trội tuyệt đối của LightGBM về mặt tối ưu tài nguyên. Cùng một mức hiệu năng, LightGBM:
  - Huấn luyện nhanh nhất (chỉ 397.92giây),
  - Tiết kiệm RAM nhất (chỉ 135 MB), nhẹ hơn một nửa XGBoost, nhẹ hơn 13 lần TensorFlow.
  - Trong khi cho hiệu năng tương đương hoặc tốt hơn XGBoost, và vượt xa MLP.
- TensorFlow thì hoàn toàn không đáng xét nếu môi trường không có GPU hoặc giới hạn RAM, vì tốn gần 2GB RAM và hơn 90 phút để huấn luyện, mà hiệu năng chỉ ngang ngửa sklearn, không hề vượt trội.

- XGBoost, dù hiệu năng tốt, nhưng lại tốn thời gian gấp đôi LightGBM, và dùng nhiều RAM hơn – điều này khiến nó không thực dụng bằng LightGBM trong các bài toán cần chạy nhiều lần, hoặc đưa vào pipeline tự động.

4. Kết luận:

- Trong cả hai khía cạnh: hiệu năng dự đoán và tối ưu tài nguyên , LightGBM là mô hình phù hợp nhất để triển khai thực tế, vừa nhanh, vừa chính xác, vừa nhẹ tài nguyên. Những mô hình khác có thể mang tính tham khảo, học thuật hoặc dùng trong trường hợp cụ thể, nhưng với dữ liệu này và bài toán này, lựa chọn tốt nhất là LightGBM.

TÀI LIỆU THAM KHẢO

[1] S. Khan, “Lasso Regression: A Game-Changer for Feature Selection,” 7 Tháng 10 2024. [Trực tuyến]. Available: https://www.linkedin.com/pulse/lasso-regression-game-changer-feature-selection-shakil-khan-9li0c. [Đã truy cập 1 Tháng 4 2025].

[2] “Chat GPT”.(giúp thực hiện triển khai các ý tưởng)

[3] T. Đức, “Learning rate - Những điều có thể bạn đã bỏ qua,” 7 Tháng 2 2022. [Trực tuyến]. Available: https://viblo.asia/p/learning-rate-nhung-dieu-co-the-ban-da-bo-qua-gGJ59BV9KX2. [Đã truy cập 3 Tháng 4 2025].

[4] “ReLU Activation Function in Deep Learning,” 29 Tháng 2 2025. [Trực tuyến]. Available: https://www.geeksforgeeks.org/relu-activation-function-in-deep-learning/. [Đã truy cập 10 Tháng 4 2025].

[5] L. Nguyen, “3 Cấp độ hiểu về Batch Normalization,” 26 Tháng 1 2023. [Trực tuyến]. Available: https://viblo.asia/p/3-cap-do-hieu-ve-batch-normalization-bai-dich-johann-huber-Yym40mRmJ91#_224-tong-quan-ket-qua-7. [Đã truy cập 10 Tháng 4 2025].

[6] T. T. Trực, “Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,..),” 17 Tháng 10 2020. [Trực tuyến]. Available: https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8. [Đã truy cập 11 Tháng 4 2025].

[7] O. DANIEL, “How To Change the Learning Rate of TensorFlow,” 4 Tháng 7 2023. [Trực tuyến]. Available: https://medium.com/@danielonugha0/how-to-change-the-learning-rate-of-tensorflow-b5d854819050. [Đã truy cập 10 Tháng 4 2025].