

Blog Tuần 2 – Module 4

Gradient Boosting, XGBoost, FastAPI và Data Project

Từ gradient boosting cơ bản đến XGBoost nâng cao, FastAPI deployment và dự án thực tế

Tác giả: Team GRID034

Tuần thứ hai của Module 4 tập trung vào các **thuật toán boosting** tiên tiến và **triển khai API**, cùng với việc áp dụng vào **dự án thực tế**. Module này giúp người học nắm vững gradient boosting từ cơ bản đến nâng cao và cách triển khai mô hình ML thành API production.

Cụ thể, blog tuần này sẽ bao gồm các chủ đề:

1. **Gradient Boosting – Học qua ví dụ tính tay** Hiểu gradient boosting từ cơ bản thông qua các ví dụ tính toán chi tiết và minh họa trực quan.
2. **Gradient Boosting – Nâng cao** Các kỹ thuật tối ưu hóa, regularization và so sánh với các thuật toán ensemble khác.
3. **XGBoost – Extreme Gradient Boosting** Thuật toán XGBoost hiện đại, ưu điểm vượt trội và ứng dụng trong các cuộc thi data science.
4. **FastAPI – Xây dựng API cho ML Models** Triển khai mô hình ML thành API production-ready với FastAPI framework.
5. **Data Project – PG&E Energy Analytics Challenge** Dự án thực tế dự báo tải điện sử dụng time-series analysis và machine learning.

Giá trị nhận được sau khi đọc Blog

- Hiểu và triển khai được Gradient Boosting từ cơ bản đến nâng cao.
- Nắm vững thuật toán XGBoost và ứng dụng trong các dự án thực tế.
- Biết cách xây dựng API cho mô hình ML với FastAPI.
- Áp dụng được time-series analysis và feature engineering.

- Triển khai dự án ML hoàn chỉnh từ EDA đến production.

Mục lục

Gradient Boosting – Học qua ví dụ tính tay	4
• Gradient Boosting là gì?	4
• Gradient Boosting – Học từ phần dư	4
• Nguyên lý hoạt động	5
• Các khái niệm cơ bản	7
• So sánh AdaBoost và Gradient Boosting	9
Gradient Boosting – Nâng cao	14
• Giới thiệu: Ngoài việc học từ sai lầm, hãy học đúng hướng!	14
• Trái tim của thuật toán: "Gradient" đến từ đâu?	14
• Gradient Boosting cho Bài toán Hồi quy	16
• Gradient Boosting cho Bài toán Phân loại	18
• Thực hành với Python	20
XGBoost – Extreme Gradient Boosting	25
• XGBoost là gì?	25
• Behind the Scenes – Bí mật dưới nắp capo của XGBoost	26
• Case Study – Time Series Forecasting với XGBoost	27
• Ưu – Nhược điểm của XGBoost	28
• Ứng dụng thực tế & Kết luận	29
FastAPI – Xây dựng API cho ML Models	31
• Giới thiệu về FastAPI	31
• Ứng dụng CRUD đơn giản với FastAPI	34
• Nâng cao trải nghiệm với Response và Templating	36
• Case Study: AI Text Classification API	39
• MLOps cho dự án AI	47
Data Project – PG&E Energy Analytics Challenge	50
• Giới thiệu Dự án PG&E Energy Analytics Challenge	50
• Phân tích Dữ liệu Thăm dò (EDA)	52
• Feature Engineering	58
• Modeling và Evaluation	63
• Kết quả và Đánh giá	71

Gradient Boosting

Vương Nguyệt Bình

1. Gradient Boosting là gì?

Hãy tưởng tượng bạn đang học chơi đàn piano. Ban đầu, bản nhạc bạn gõ nghe khá chêch choạc – nhiều nốt sai, nhịp chưa chuẩn. Nhưng thay vì bỏ cuộc, bạn nghe lại, phát hiện lỗi và chỉnh dần từng chỗ: nhấn nhẹ hơn ở đây, giữ nhịp đều hơn ở kia. Sau vài lần tập, bản nhạc dần trở nên mượt mà, tròn trịa.

Gradient Boosting cũng học theo cách tương tự. Nó bắt đầu với một mô hình đơn giản – gần như “ngây ngô”. Rồi ở mỗi vòng lặp, nó soi lại lỗi (*residuals*), học cách “sửa sai” bằng cách thêm một cây quyết định nhỏ để khắc phục. Lặp đi lặp lại, mô hình ngày càng chính xác, giống như bản nhạc dần vang lên hoàn hảo.

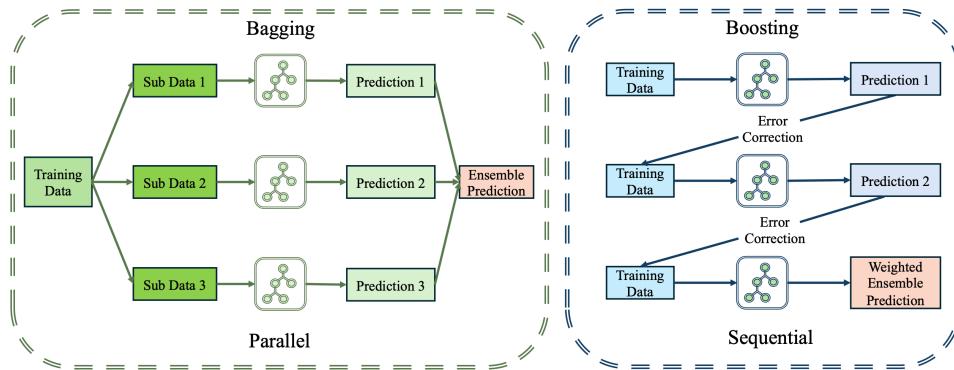
Về bản chất, Gradient Boosting thuộc họ **Boosting** – một kỹ thuật *ensemble learning* nơi các mô hình yếu (weak learners) được xây dựng tuần tự, và mỗi mô hình mới đều có nhiệm vụ sửa sai cho mô hình trước đó.

Điểm đặc biệt là Gradient Boosting không chỉ đoán mò. Nó biết dùng **Gradient Descent trong không gian hàm** để tìm hướng đi đúng nhất nhằm giảm sai số. Nói cách khác, mỗi “cây nồng” được thêm vào không phải ngẫu nhiên, mà chính là bước đi có tính toán, được dẫn dắt bởi đạo hàm của hàm lỗi.

Nếu AdaBoost giống như việc “chú ý nhiều hơn vào những bài tập khó”, thì Gradient Boosting lại giống như một học trò kiên nhẫn, cẩn thận phân tích lỗi sai và chỉnh sửa chúng từng chút một cho đến khi đạt kết quả tốt nhất.

2. Gradient Boosting – Học từ phần dư

Gradient Boosting là một trong những phương pháp **Boosting** nổi bật nhất trong học máy. Khác với **Bagging** (như Random Forest) – nơi các mô hình được huấn luyện song song và độc lập – Boosting xây dựng mô hình theo cách tuần tự, trong đó mỗi mô hình mới tập trung vào việc **sửa lỗi còn lại** của các mô hình trước.



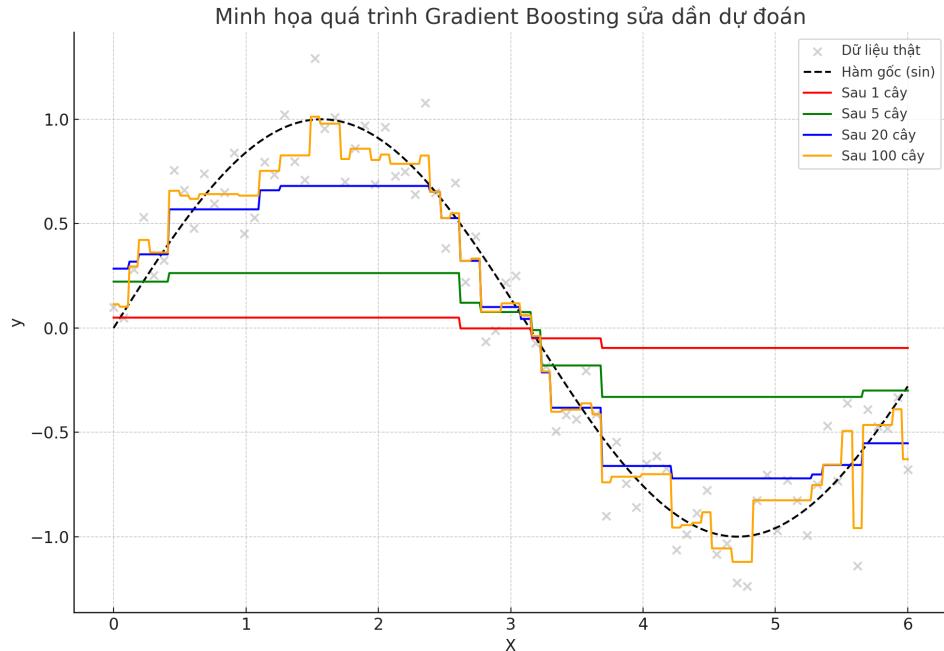
Hình 1: Boosting (sửa lỗi tuần tự) so với Bagging (trung bình song song).

Điểm then chốt của Gradient Boosting là **học từ phần dư (residuals)** và sử dụng **Gradient Descent trong không gian hàm** để dần tối ưu hoá hàm lỗi. Thay vì cố gắng dự đoán trực tiếp nhãn y , mỗi vòng lặp mô hình học cách sửa phần sai sót còn lại của dự đoán hiện tại.

Một cách dễ hình dung: hãy tưởng tượng bạn muốn vẽ lại một đường cong dữ liệu bằng nhiều “nét bút nhô”:

- Nét bút đầu tiên khá thô, còn lệch nhiều so với đường cong gốc.
- Các nét sau không vẽ lại từ đầu, mà chỉ tập trung sửa vào những chỗ bị lệch: thêm vào đoạn còn thiếu, chỉnh nhẹ những chỗ thừa.
- Sau nhiều lần chỉnh sửa, bức tranh dần tiệm cận với đường cong thật.

Gradient Boosting cũng vận hành như vậy: bắt đầu bằng một mô hình đơn giản, sau đó liên tục bổ sung những cây nhỏ để sửa dần các phần sai sót, và cuối cùng tạo ra một mô hình mạnh mẽ, chính xác hơn nhiều so với từng thành phần riêng lẻ.



Hình 2: Quá trình Gradient Boosting dần cải thiện dự đoán. Ban đầu (đường đỏ, sau 1 cây) mô hình còn thô, sau nhiều vòng (20–100 cây) mô hình bám sát dữ liệu hơn, minh họa cách “sửa sai dần dần” của thuật toán.

2.1 Nguyên lý hoạt động

Giả sử sau $(m-1)$ vòng boosting, ta có một mô hình dự đoán $F_{m-1}(x)$.

Sau mỗi vòng lặp, điều quan trọng là xác định mô hình hiện tại còn sai ở đâu. Ta đo sai số còn lại thông qua *pseudo-residuals*, tức là gradient âm của hàm lỗi:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}.$$

Một số ví dụ cụ thể:

- **Hồi quy (MSE):**

$$r_{im} = y_i - F_{m-1}(x_i).$$

- **Phân loại nhị phân (Logistic Loss):**

$$r_{im} = y_i - p_{m-1}(x_i),$$

với $p_{m-1}(x_i)$ là xác suất dự đoán từ mô hình trước đó.

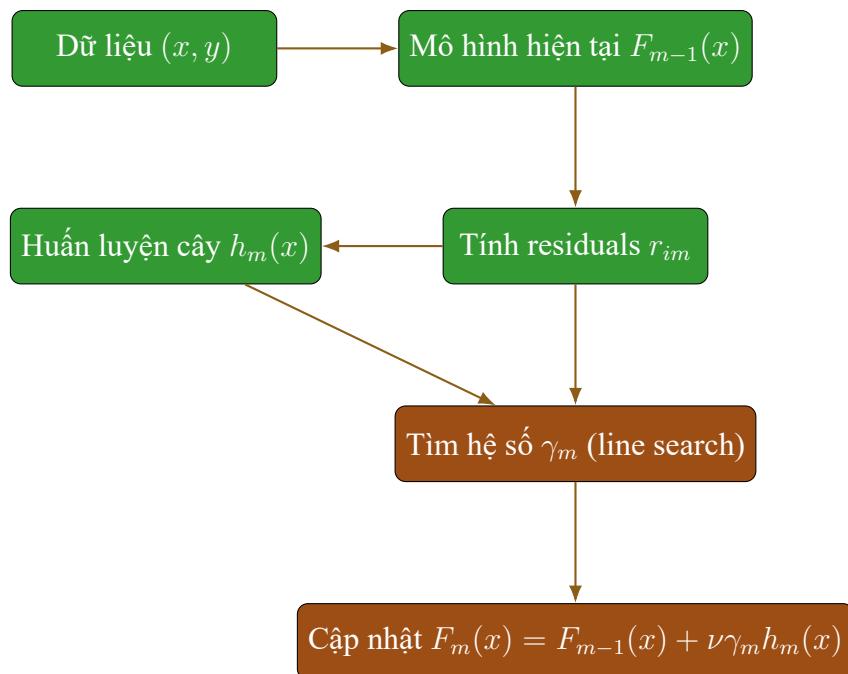
Tiếp theo, một cây quyết định nhỏ $h_m(x)$ được huấn luyện để dự đoán các residuals này. Trong thực tế, cây mới chỉ cho ta *hướng sửa lỗi*. Để biết nên “bước bao xa” theo hướng đó, ta tìm một hệ số γ_m bằng *line search*, giúp mô hình điều chỉnh vừa đủ — không quá ít, cũng không quá nhiều.

Mô hình sau đó được cập nhật theo công thức:

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m \cdot h_m(x),$$

trong đó ν là *learning rate* (0.01–0.1).

Gradient Boosting không xây dựng sự hoàn hảo ngay từ đầu, mà tiến gần tới nó bằng cách liên tục sửa chữa những sai sót còn lại.



Hình 3: Pipeline một vòng lặp Gradient Boosting.

2.2 Pseudocode

Các bước trên có thể tóm tắt lại bằng giả mã như sau:

Algorithm 1 Gradient Boosting

Require: Dữ liệu $\{(x_i, y_i)\}$, loss L , số vòng M , learning rate ν

- 1: Khởi tạo $F_0(x) = \arg \min_c \sum_i L(y_i, c)$
 - 2: **for** $m = 1$ đến M **do**
 - 3: Tính residuals $r_i^{(m)} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$
 - 4: Huấn luyện cây $h_m(x)$ để dự đoán $r_i^{(m)}$
 - 5: Tìm $\gamma_m = \arg \min_\gamma \sum_i L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
 - 6: Cập nhật $F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m \cdot h_m(x)$
 - 7: **end for**
 - 8: **return** $F_M(x)$
-

2.3 Các khái niệm cơ bản

2.3.1 Học viên yếu (Weak Learner)

Trong học máy, một **học viên yếu** là mô hình rất đơn giản, chỉ dự đoán tốt hơn một chút so với việc chọn ngẫu nhiên. Ý tưởng cốt lõi của Boosting là: *nhiều học viên yếu, khi kết hợp khéo léo, có thể tạo thành một học viên mạnh.*

Trong Gradient Boosting, học viên yếu phổ biến nhất là **cây quyết định nông** (shallow decision trees), thường có độ sâu chỉ từ 1–5 mức. Một trường hợp đặc biệt là **decision stump** — cây chỉ có một nút chia, tương đương với một “câu hỏi có/không” trên một đặc trưng.

Đặc điểm của những cây này:

- **Độ lệch cao (high bias):** chúng không đủ phức tạp để khớp toàn bộ dữ liệu.
- **Phương sai thấp (low variance):** do cấu trúc đơn giản, chúng ít nhạy cảm với nhiễu.

Khi được kết hợp trong Gradient Boosting:

- Mỗi cây nhỏ giống như một “nét bút” tinh chỉnh mô hình hiện tại, sửa một phần lỗi còn sót lại.
- Qua nhiều vòng lặp, dãy cây này dần dần giảm sai số, từ một mô hình thô sơ thành một mô hình mạnh mẽ.

Nếu coi quá trình học là việc vẽ một bức tranh: mỗi cây quyết định nông chỉ vẽ được vài nét đơn giản, nhưng khi chồng ghép hàng trăm nét lại với nhau, ta có thể tái tạo được toàn bộ bức tranh phức tạp.

2.3.2 Hàm lỗi (Loss Function)

Cốt lõi của Gradient Boosting là **học từ sai lầm**. Nhưng để biết “sai” ở đâu và “sai” bao nhiêu, ta cần một thước đo — đó chính là **hàm lỗi (loss function)**.

Hàm lỗi đóng vai trò như chiếc *la bàn*: nó chỉ ra hướng mà mô hình cần điều chỉnh. Mỗi vòng lặp, Gradient Boosting tính đạo hàm của hàm lỗi để biết cần thêm một “bước sửa” như thế nào. Nếu không có hàm lỗi, mô hình sẽ không biết mình cần cải thiện ở đâu.

Một số ví dụ phổ biến:

- **Hồi quy (Regression):** thường dùng **Mean Squared Error (MSE)** vì nó phạt nặng những sai số lớn, khuyến khích mô hình dự đoán sát giá trị thật:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2,$$

với y_i là giá trị thực tế, \hat{y}_i là giá trị dự đoán.

Ý tưởng: nếu dự đoán sai nhiều, ta muốn “kéo mạnh” mô hình về gần giá trị thật hơn.

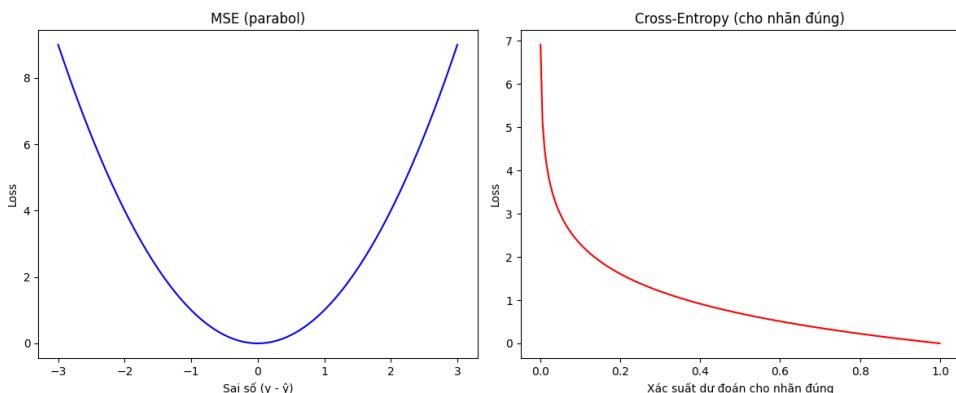
- **Phân loại nhị phân (Classification):** thường dùng **Cross-Entropy (CE) / Logistic Loss**, vì nó so sánh phân phối xác suất dự đoán với nhãn thật, đảm bảo mô hình không chỉ dự đoán đúng/ sai, mà còn tự tin hợp lý:

$$CE = -\frac{1}{n} \sum_{i=1}^n \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right],$$

trong đó $y_i \in \{0, 1\}$ là nhãn thật, $\hat{y}_i \in [0, 1]$ là xác suất dự đoán.

Ý tưởng: nếu mô hình dự đoán xác suất thấp cho nhãn đúng, hàm lỗi sẽ phạt rất nặng, buộc mô hình phải học “tự tin hơn ở chỗ đúng”.

Nhờ tính linh hoạt trong việc chọn hàm lỗi, Gradient Boosting có thể áp dụng cho nhiều bài toán khác nhau: từ hồi quy, phân loại, cho đến ranking hay survival analysis.



Hình 4: Trực quan hoá hai hàm lỗi phổ biến: **MSE** (trái) có dạng parabol, sai số càng lớn thì bị phạt mạnh; **Cross-Entropy** (phải) phạt rất nặng khi mô hình dự đoán xác suất thấp cho nhãn đúng.

2.3.3 Pseudo-residuals

Trung tâm của Gradient Boosting nằm ở khái niệm *pseudo-residuals* – gradient âm của hàm lỗi tại dự đoán hiện tại:

$$r_{im} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}.$$

Ví dụ:

- VỚI MSE: $r_{im} = y_i - F(x_i)$, đây chính là residuals truyền thống.
- VỚI Logistic Loss: $r_{im} = y_i - p_{m-1}(x_i)$, trong đó $p_{m-1}(x_i)$ là xác suất dự đoán từ mô hình trước đó.

2.3.4 Cập nhật mô hình (Model Update)

Sau khi có residuals, một cây $h_m(x)$ được huấn luyện để dự đoán chúng. Mô hình sau đó được cập nhật:

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x),$$

trong đó:

- $F_{m-1}(x)$: mô hình trước đó.
- ν : learning rate (0.01–0.1) để kiểm soát bước tiến, giảm nguy cơ overfitting.
- $h_m(x)$: cây mới học residuals.

2.3.5 Tiêu chí dừng (Stopping criteria)

- **Giới hạn số vòng lặp M** : đặt trước số lần mô hình được phép học thêm (ví dụ 100 hoặc 500 vòng).
- **Dừng sớm (early stopping)**: nếu trong k vòng liên tiếp, điểm số trên tập validation không được cải thiện, thì dừng lại để tránh lãng phí thời gian và bị quá khứ.
- **Nguồn cải thiện rất nhỏ**: nếu tổng hàm mất mát (loss) giữa hai vòng liên tiếp thay đổi ít hơn một nguồn đặt trước, thì coi như mô hình đã hội tụ và dừng lại.

2.3.6 Gradient Descent trong không gian hàm

Điểm khác biệt cốt lõi là Gradient Boosting áp dụng Gradient Descent không phải trên tham số riêng lẻ, mà trong **không gian hàm**. Mỗi cây mới được thêm vào tương ứng với một bước đi theo hướng giảm dốc nhất của hàm lỗi. Qua nhiều vòng, mô hình dần tiến tới nghiệm tối ưu toàn cục hoặc cục bộ tốt.

3. So sánh AdaBoost và Gradient Boosting

- **AdaBoost**: Huấn luyện nhiều học viên yếu, sau đó tăng trọng số cho các mẫu bị phân loại sai, buộc mô hình sau tập trung vào chúng.
- **Gradient Boosting**: Không thay đổi trọng số dữ liệu, mà trực tiếp tối thiểu hóa hàm lỗi bằng cách học residuals.

4. Kỹ thuật điều chuẩn và các cải tiến

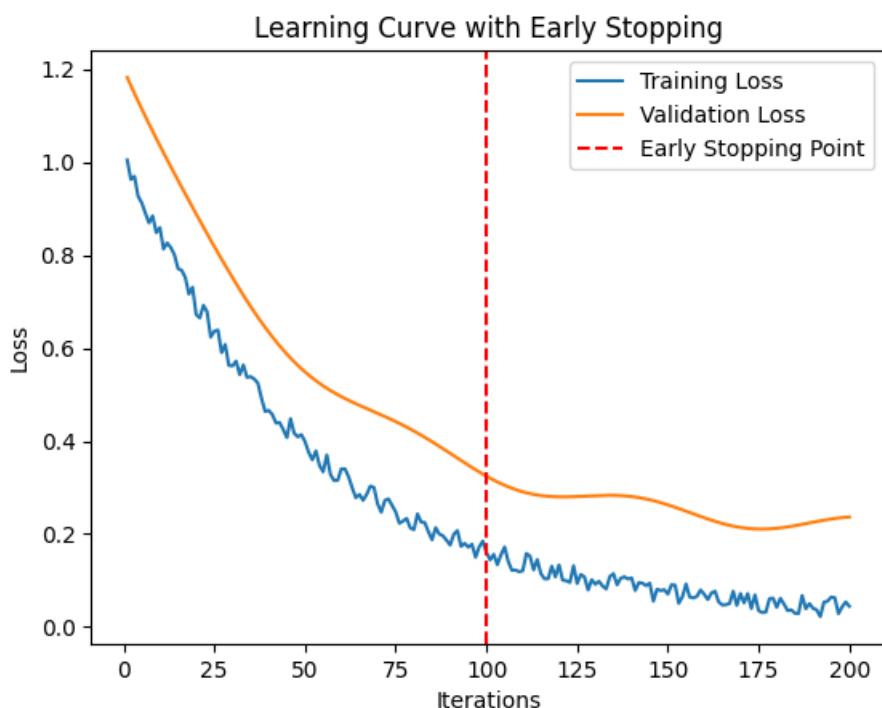
Gradient Boosting rất mạnh nhưng cũng dễ bị *overfitting* nếu không được kiểm soát. Vì vậy, nhiều kỹ thuật điều chuẩn đã được phát triển:

- **Learning rate (shrinkage)**: giảm mức đóng góp của mỗi cây. Ví dụ, thay vì cộng thăng $h_m(x)$, ta chỉ cộng một phần nhỏ $\nu h_m(x)$. Learning rate nhỏ giúp mô hình học chậm hơn nhưng tổng quát tốt hơn.
- **Subsampling (stochastic gradient boosting)**: huấn luyện mỗi cây trên một tập con ngẫu nhiên (ví dụ 70–80% dữ liệu). Kỹ thuật này vừa giảm phuơng sai, vừa tăng tốc, tương tự như “bagging nhẹ”.

- **Tree constraints:** áp các giới hạn để cây không quá phức tạp:
 - Độ sâu tối đa (*max depth*)
 - Số lá tối đa (*max leaves*)
 - Số mẫu tối thiểu trong mỗi lá (*min samples per leaf*)

Các giới hạn này giữ cây ở mức “yếu” (weak learner) đúng nghĩa, tránh việc cây học quá chi tiết.

- **Early stopping:** theo dõi loss trên tập validation, nếu sau k vòng liên tiếp không cải thiện thì dừng sớm. Điều này giúp tiết kiệm thời gian và tránh mô hình “học thuộc” dữ liệu huấn luyện.



Hình 5: Learning curve minh họa Early Stopping: Validation loss ngừng cải thiện sau khoảng 100 vòng, trong khi training loss tiếp tục giảm.

Các cải tiến hiện đại

Dựa trên ý tưởng Gradient Boosting, nhiều biến thể đã được phát triển để tăng tốc và tăng độ chính xác:

- **XGBoost:** bổ sung regularization L1/L2 trên trọng số lá, hỗ trợ tính toán song song, tối ưu bộ nhớ.
- **LightGBM:** dùng kỹ thuật *leaf-wise growth* (phát triển cây theo lá thay vì theo mức), kết hợp với histogram để xử lý nhanh dữ liệu lớn.
- **CatBoost:** tối ưu cho dữ liệu phân loại (categorical features), giảm hiện tượng overfitting nhờ kỹ thuật *ordered boosting*.

Các cải tiến này giúp Gradient Boosting không chỉ chính xác mà còn khả thi với tập dữ liệu lớn, phức tạp trong thực tế.

5. So sánh với các thuật toán khác

Tiêu chí	Random Forest	Gradient Boosting	AdaBoost
Phong cách huấn luyện	Song song	Tuần tự	Tuần tự
Mục tiêu chính	Giảm variance	Giảm bias	Tập trung mẫu khó
Tính chỉnh tham số	Ít	Nhiều	Trung bình
Nguy cơ overfitting	Thấp	Cao (nếu không regularize)	Trung bình
Tốc độ huấn luyện	Nhanh	Chậm hơn	Trung bình
Ứng dụng điển hình	Baseline mạnh, dữ liệu nhiều	Dữ liệu bảng sạch, cần chính xác cao	Dữ liệu nhỏ, dễ huấn luyện

Bảng 1: So sánh giữa Random Forest, Gradient Boosting và AdaBoost.

6. Từ thuật toán nền tảng đến các thư viện hiện đại

Sự ra đời của các thư viện mới đã mở rộng sức mạnh của Gradient Boosting:

- **XGBoost**: regularization L1/L2, tính toán song song, tối ưu bộ nhớ.
- **LightGBM**: chiến lược tăng trưởng theo lá, GOSS và EFB giúp huấn luyện nhanh hơn trên dữ liệu lớn.
- **CatBoost**: xử lý biến phân loại tự động, Ordered Boosting, cây đôi xứng giúp tăng tốc inference.

7. Ví dụ bằng Python

Để thấy rõ cách Gradient Boosting hoạt động trong thực tế, ta thử áp dụng thuật toán này trên một bài toán hồi quy đơn giản bằng thư viện scikit-learn.

```

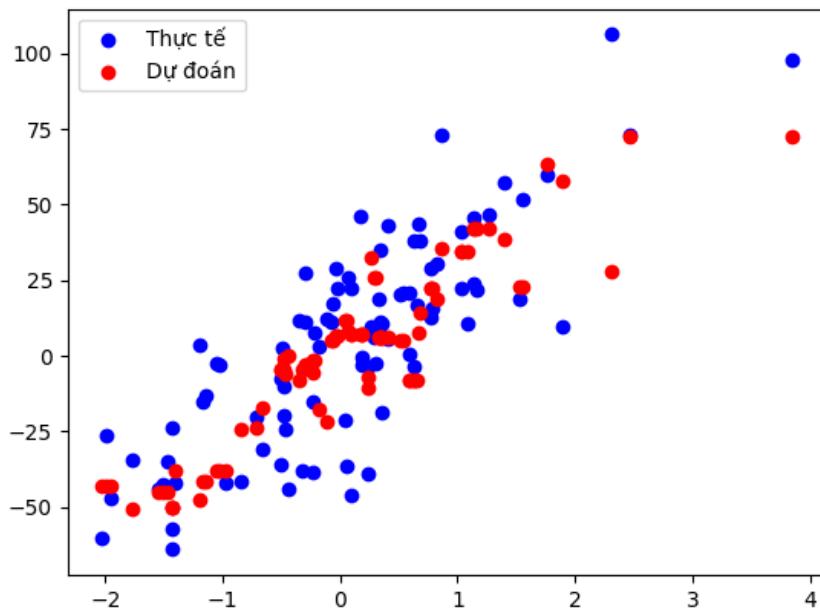
1 from sklearn.ensemble import GradientBoostingRegressor
2 from sklearn.datasets import make_regression
3 from sklearn.model_selection import train_test_split
4 import matplotlib.pyplot as plt
5
6 X, y = make_regression(n_samples=300, n_features=1, noise=20, random_state=42)
7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
8
9 gbr = GradientBoostingRegressor(n_estimators=200, learning_rate=0.05, max_depth=3)
10 gbr.fit(X_train, y_train)
11
12

```

```

13 plt.scatter(X_test, y_test, color="blue", label="Thực tế")
14 plt.scatter(X_test, gbr.predict(X_test), color="red", label="Dự đoán")
15 plt.legend(); plt.show()

```



Hình 6: Kết quả dự đoán của Gradient Boosting Regressor

8. Ứng dụng thực tiễn

Gradient Boosting đặc biệt hiệu quả trên dữ liệu dạng bảng (tabular data). Một số ứng dụng:

- **Tài chính:** chấm điểm tín dụng, phát hiện gian lận.
- **Marketing:** dự đoán churn, tối ưu chiến dịch.
- **Y tế:** mô hình nguy cơ bệnh, phân tích dữ liệu lâm sàng.
- **Công nghiệp:** dự báo nhu cầu, mô phỏng, xếp hạng.

Ví dụ: Dự đoán churn khách hàng

Sử dụng LightGBM trên dữ liệu churn - dự đoán khách hàng nào có nguy cơ rời bỏ dịch vụ (unsubscribe, ngừng sử dụng). Đây là một ứng dụng kinh điển của học máy trong marketing và chăm sóc khách hàng.

```

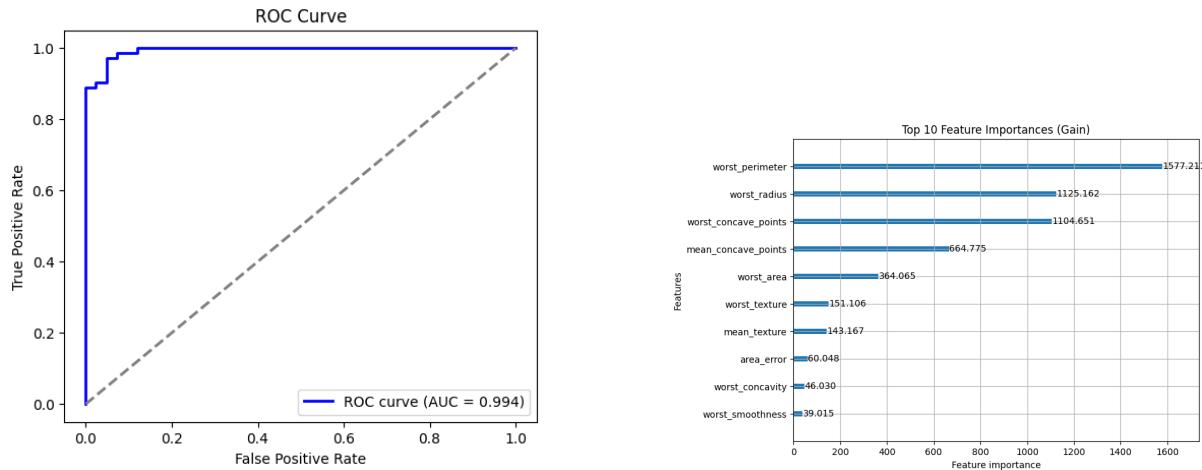
1 import lightgbm as lgb
2 train_data = lgb.Dataset(X_train, label=y_train)
3 valid_data = lgb.Dataset(X_test, label=y_test)
4
5 params = {
6     "objective": "binary",
7     "learning_rate": 0.05,
8     "num_leaves": 31,
9     "metric": "auc"
10

```

```

11 }
12 model = lgb.train(params, train_data, valid_sets=[valid_data],
13                   num_boost_round=500, early_stopping_rounds=50)
14
15 print("Best AUC:", model.best_score["valid_0"]["auc"])

```



Hình 7: (Trái) Đường cong ROC minh họa khả năng phân biệt của mô hình LightGBM trong việc dự đoán churn. AUC càng cao, mô hình càng tốt (ở đây ≈ 0.99). (Phải) Biểu đồ tầm quan trọng đặc trưng (Feature Importance) cho thấy các yếu tố có ảnh hưởng mạnh nhất đến dự đoán khách hàng rời bỏ dịch vụ.

Kết quả cho thấy mô hình đạt AUC rất cao, chứng tỏ khả năng phân biệt tốt giữa nhóm khách hàng có nguy cơ rời bỏ và nhóm ở lại. Biểu đồ Feature Importance giúp đội ngũ marketing hiểu rõ những yếu tố nào (ví dụ như mức độ sử dụng dịch vụ, thời gian gắn bó, chi phí hàng tháng, v.v.) có ảnh hưởng nhiều nhất đến hành vi churn, từ đó đưa ra các chiến lược giữ chân khách hàng hiệu quả.

9. Kết luận

Gradient Boosting là một thuật toán vừa thanh lịch vừa thực dụng: bằng cách học từ phần dư và cập nhật có kiểm soát, nó tạo ra một mô hình có khả năng dự đoán vượt trội. Hiểu rõ cơ chế và kỹ thuật điều chỉnh chuẩn giúp tận dụng được sức mạnh này. Trong hầu hết các bài toán học máy trên dữ liệu dạng bảng, Gradient Boosting (và các biến thể hiện đại như XGBoost, LightGBM, CatBoost) thường là lựa chọn hàng đầu.

Khám phá Gradient Boosting: Nghệ thuật điêu khắc mô hình từ sai số

Vũ Thái Sơn

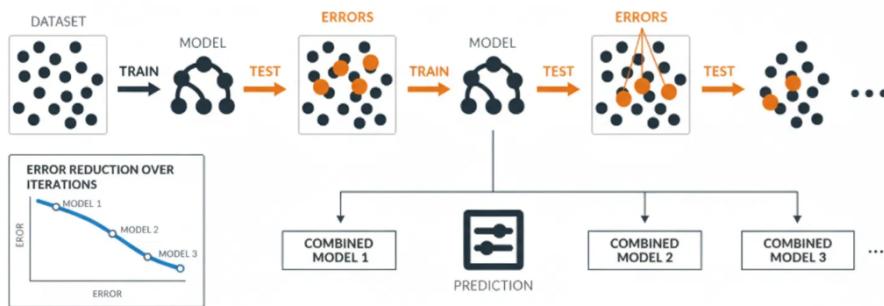
”Lấy sai số làm kim chỉ nam”: Lý thuyết đến thực hành Gradient Boosting

1. Giới thiệu: Ngoài việc học từ sai lầm, hãy học đúng hướng!

Hãy tưởng tượng bạn là một nhà điêu khắc đang tạc một bức tượng từ một khối đá cẩm thạch thô. Bạn không thể tạo ra kiệt tác ngay trong nhát đục đầu tiên. Thay vào đó, bạn bắt đầu với những đường nét cơ bản, sau đó lùi lại, quan sát những điểm chưa hoàn hảo – ”sai số” so với hình ảnh trong tâm trí – rồi cẩn thận đục đẽo để sửa chữa những sai số đó. Mỗi lớp đá được loại bỏ là một bước đưa tác phẩm đến gần hơn với sự hoàn hảo.

Đây là triết lý cốt lõi của các thuật toán **Boosting** trong **Học tập Tập thể** (**Ensemble Learning**). Thay vì xây dựng một mô hình phức tạp duy nhất, chúng ta kết hợp sức mạnh của nhiều mô hình đơn giản (weak learners), và mỗi mô hình sau sẽ học từ sai lầm của các mô hình trước. AdaBoost, một thuật toán tiên phong, thực hiện điều này bằng cách tăng trọng số cho các điểm bị dự đoán sai, buộc mô hình sau phải ”chú ý” hơn đến chúng.

Nhưng Gradient Boosting còn tiến một bước xa hơn. Nó không chỉ hỏi ”Tôi đã sai ở đâu?”, mà còn hỏi ”**Để sửa sai, tôi nên đi theo hướng nào là nhanh nhất?**”. Câu trả lời nằm ở hai chữ **”Gradient Descent”**. Gradient Boosting áp dụng một cách thiêng tài ý tưởng tối ưu hóa này vào không gian của các mô hình, tạo ra một nghệ sĩ bậc thầy có khả năng điêu khắc nên những mô hình dự đoán với độ chính xác đáng kinh ngạc.



Hình 8: Đi theo hướng Gradient để tìm loss bé nhất.

Trong bài viết này, chúng ta sẽ cùng nhau thực hiện một hành trình chi tiết, từ việc ”giải phẫu” lý thuyết và công thức toán học, đến việc áp dụng nó vào hai ví dụ kinh điển: hồi quy và phân loại, với các bước tính toán cụ thể, và cuối cùng là hiện thực hóa bằng mã nguồn Python.

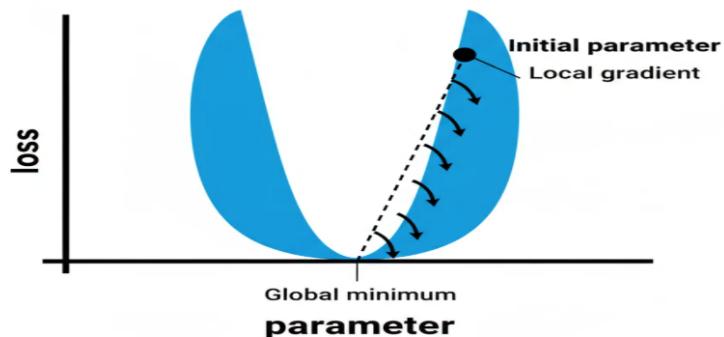
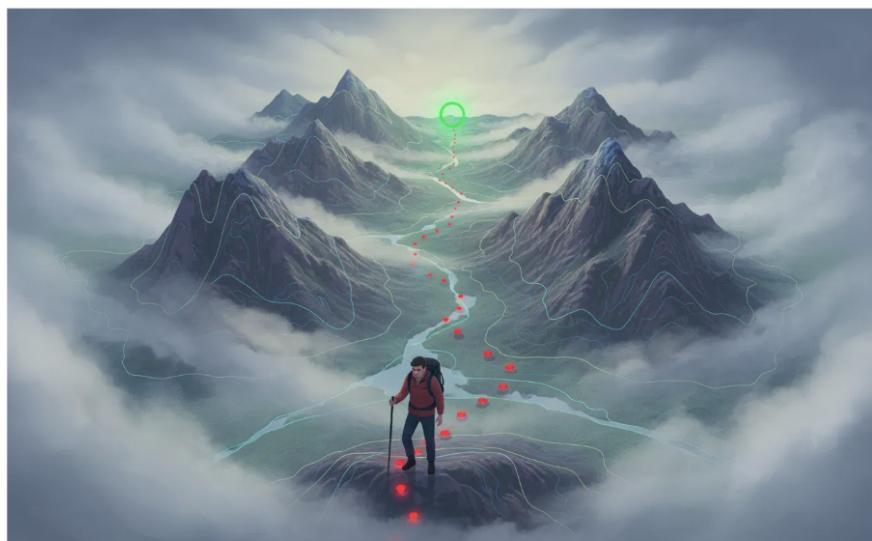
2. Trái tim của thuật toán: ”Gradient” đến từ đâu?

Để thực sự hiểu Gradient Boosting, chúng ta cần nắm vững trực giác đằng sau chữ ”Gradient”. Hãy tưởng tượng bạn đang đứng trên một sườn núi trong sương mù, và mục tiêu là đi xuống thung lũng (nơi thấp nhất).

- **Vị trí của bạn:** Tương đương với **dự đoán hiện tại** của mô hình.
- **Độ cao:** Tương đương với **sai số** (được đo bằng hàm mất mát - Loss Function).
- **Thung lũng:** Tương đương với **mô hình hoàn hảo** (sai số bằng 0).

Trong sương mù, bạn không thấy thung lũng. Cách duy nhất là cảm nhận **độ dốc (gradient)** ngay dưới chân mình. Gradient cho bạn biết hướng dốc lên nhất. Để đi xuống, bạn chỉ cần bước một bước nhỏ theo hướng **ngược lại với gradient (negative gradient)**. Lặp lại quá trình này, bạn sẽ dần dần đi tới đáy thung lũng.

Gradient Boosting đã "mượn" ý tưởng này. Nó xem việc tối ưu mô hình như một hành trình đi xuống "thung lũng sai số". **Phần dư (residual)** mà chúng ta sẽ tính toán ở các bước tiếp theo, về mặt toán học, chính là một xáp xỉ của **hướng negative gradient** đó. Mỗi cây quyết định mới được thêm vào chính là một "bước đi" theo hướng giảm sai số nhanh nhất.



Hình 9: Đi theo hướng Gradient để tìm loss bé nhất.

3. Gradient Boosting cho Bài toán Hồi quy

3.1 Ý tưởng chính và Công thức

Ý tưởng là xây dựng một chuỗi các cây quyết định, trong đó mỗi cây sau sẽ cố gắng dự đoán **phản dư (residual)** của tất cả các cây trước đó cộng lại.

1. **Khởi tạo (F_0):** Bắt đầu với một dự đoán không đổi, thường là giá trị trung bình của biến mục tiêu y .

$$F_0(x) = \text{mean}(y)$$

2. **Lặp (với m từ 1 đến M):**

- (a) Tính phản dư (pseudo-residuals):

$$r_{im} = y_i - F_{m-1}(x_i)$$

- (b) Huấn luyện một cây quyết định hồi quy $h_m(x)$ mới để dự đoán các phản dư r_{im} .

- (c) Cập nhật mô hình tổng hợp:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

3.2 Ví dụ tính toán chi tiết

Hãy áp dụng quy trình trên vào bộ dữ liệu sau để dự đoán **Weight (kg)**.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Hình 10: Bộ dữ liệu ví dụ cho bài toán hồi quy.

3.2.1 Vòng lặp 1: Xây dựng cây đầu tiên

Bước 1: Khởi tạo (F_0)

$$F_0 = \frac{88 + 76 + 56 + 73 + 77 + 57}{6} = 71.17$$

Bước 2: Tính Phản dư (r_1)

ID	Height (m)	Weight (y)	F_0	Residual (r_1)
1	1.6	88	71.17	16.83
2	1.6	76	71.17	4.83
3	1.5	56	71.17	-15.17
4	1.8	73	71.17	1.83
5	1.5	77	71.17	5.83
6	1.4	57	71.17	-14.17

Bảng 2: Tính toán phản dư cho vòng lặp đầu tiên.

Bước 3: Xây dựng Cây $h_1(x)$ Giả sử stump tốt nhất chia theo Height ≤ 1.55 .

- **Lá Trái (Height ≤ 1.55)**: Gồm các điểm (ID 3, 5, 6).

$$\text{Leaf}_{\text{Left}} = \frac{-15.17 + 5.83 - 14.17}{3} \approx -7.84$$

- **Lá Phải (Height > 1.55)**: Gồm các điểm (ID 1, 2, 4).

$$\text{Leaf}_{\text{Right}} = \frac{16.83 + 4.83 + 1.83}{3} \approx 7.83$$

Bước 4: Cập nhật Mô hình (F_1)

Giả sử tốc độ học $\eta = 0.1$.

ID	F_0	$h_1(x_i)$	$F_1(x_i) = F_0 + \eta \cdot h_1$
1	71.17	7.83	$71.17 + 0.1 \times 7.83 = 71.95$
2	71.17	7.83	$71.17 + 0.1 \times 7.83 = 71.95$
3	71.17	-7.84	$71.17 + 0.1 \times (-7.84) = 70.39$
4	71.17	7.83	$71.17 + 0.1 \times 7.83 = 71.95$
5	71.17	-7.84	$71.17 + 0.1 \times (-7.84) = 70.39$
6	71.17	-7.84	$71.17 + 0.1 \times (-7.84) = 70.39$

Bảng 3: Cập nhật dự đoán sau vòng lặp đầu tiên.

3.2.2 Vòng lặp 2: Xây dựng cây thứ hai

Bước 5: Tính Phản dư mới (r_2)

ID	Weight (y)	$F_1(x_i)$	Residual ($r_2 = y - F_1$)
1	88	71.95	16.05
2	76	71.95	4.05
3	56	70.39	-14.39
4	73	71.95	1.05
5	77	70.39	6.61
6	57	70.39	-13.39

Bảng 4: Tính toán phản dư cho vòng lặp thứ hai.

Bước 6: Xây dựng cây $h_2(x)$ và cập nhật $F_2(x)$

Một cây quyết định mới, $h_2(x)$, sẽ được huấn luyện để dự đoán cột **Residual (r_2)**. Sau đó, mô hình sẽ được cập nhật một lần nữa: $F_2(x) = F_1(x) + \eta \cdot h_2(x)$.

4. Gradient Boosting cho Bài toán Phân loại

4.1 Ý tưởng chính và Công thức

Thuật toán làm việc với xác suất và tối ưu trên không gian log-odds.

1. **Khởi tạo (F_0):** Bắt đầu với một dự đoán log-odds không đổi.

$$F_0(x) = \log\left(\frac{p}{1-p}\right) \quad \text{với } p = \text{mean}(y)$$

2. **Lặp (với m từ 1 đến M):**

(a) Chuyển log-odds $F_{m-1}(x)$ thành xác suất $p_{m-1}(x)$ bằng hàm Sigmoid.

(b) Tính phần dư giả (pseudo-residuals):

$$r_{im} = y_i - p_{m-1}(x_i)$$

(c) Huấn luyện cây $h_m(x)$ để dự đoán r_{im} .

(d) Cập nhật mô hình trong không gian log-odds:

$$F_m(x) = F_{m-1}(x) + \eta \cdot \gamma_m(x)$$

(Với $\gamma_m(x)$ là giá trị đầu ra của các lá trong cây h_m).

4.2 Ví dụ tính toán chi tiết

Áp dụng quy trình trên vào bộ dữ liệu sau để dự đoán **Loves Troll 2** (Yes=1, No=0).

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	No
No	44	Blue	No
Yes	19	Red	Yes
No	32	Green	Yes
No	14	Blue	Yes

Hình 11: Bộ dữ liệu ví dụ cho bài toán phân loại.

4.2.1 Vòng lặp 1: Xây dựng cây đầu tiên

Bước 1: Khởi tạo (F_0, p_0)

- Xác suất trung bình: $\bar{p} = 4/6 \approx 0.67$.
- Log-odds ban đầu: $F_0 = \log(\frac{0.67}{1-0.67}) \approx 0.71$.
- Xác suất dự đoán ban đầu: $p_0 = \text{sigmoid}(0.71) \approx 0.67$.

Bước 2: Tính Phản dư giả (r_1)

ID	Age	Loves Troll 2 (y)	p_0	Residual (r_1)
1	12	1	0.67	0.33
2	87	1	0.67	0.33
3	44	0	0.67	-0.67
4	19	0	0.67	-0.67
5	32	1	0.67	0.33
6	14	1	0.67	0.33

Bảng 5: Tính toán phản dư giả cho vòng lặp đầu tiên.

Bước 3 & 4: Xây dựng Cây $h_1(x)$ và Cập nhật $F_1(x)$

Tương tự như bài toán hồi quy, một cây mới sẽ được tạo để học các phản dư này. Mô hình log-odds F_0 sẽ được cập nhật thành F_1 , dẫn đến các xác suất dự đoán mới p_1 được cải thiện.

5. Thực hành với Python

5.1 Mã nguồn cho Bài toán Hồi quy

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.ensemble import GradientBoostingRegressor
5 from sklearn.preprocessing import OneHotEncoder
6 from sklearn.compose import ColumnTransformer
7 from sklearn.pipeline import Pipeline
8
9 # 1. Create DataFrame from data
10 data_reg = {
11     'Height': [1.6, 1.6, 1.5, 1.8, 1.5, 1.4],
12     'Favorite Color': ['Blue', 'Green', 'Blue', 'Red', 'Green', 'Blue'],
13     'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female'],
14     'Weight': [88, 76, 56, 73, 77, 57]
15 }
16 df_reg = pd.DataFrame(data_reg)
17
18 # 2. Prepare data for the model
19 X = df_reg.drop('Weight', axis=1)
20 y = df_reg['Weight']
21
22 # Define preprocessing steps

```

```
23 categorical_features = ['Favorite Color', 'Gender']
24 one_hot_encoder = OneHotEncoder(handle_unknown='ignore')
25 preprocessor = ColumnTransformer(
26     transformers=[('cat', one_hot_encoder, categorical_features)],
27     remainder='passthrough' # Keep other columns (Height)
28 )
29
30 # 3. Build and Train the Model
31 # Use GradientBoostingRegressor with basic parameters
32 # n_estimators: number of trees (boosting rounds)
33 # max_depth=1: each tree is a simple stump
34 gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1, random_state=42)
35
36 # Create a pipeline to combine preprocessing and the model
37 model_pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', gbr)])
38
39 # Train the pipeline on the entire dataset
40 model_pipeline.fit(X, y)
41
42 # 4. Visualize the results
43 plt.style.use('seaborn-v0_8-whitegrid')
44 plt.figure(figsize=(10, 6))
45 # Plot the actual data points
46 plt.scatter(df_reg['Height'], y, color='red', s=100, edgecolor='k', alpha=0.7, label='Actual Data')
47
48 # To plot the prediction line, create points and sort by Height
49 X_test_sorted = X.sort_values(by='Height')
50 y_pred_sorted = model_pipeline.predict(X_test_sorted)
51 plt.plot(X_test_sorted['Height'], y_pred_sorted, color='blue', linewidth=3, label='GBM Prediction')
52
53 # Customize the plot
54 plt.title('Gradient Boosting Regression: Height vs Weight', fontsize=16)
55 plt.xlabel('Height (m)', fontsize=12)
56 plt.ylabel('Weight (kg)', fontsize=12)
57 plt.legend()
58 plt.show()
```



Hình 12: Kết quả trực quan hóa của mô hình hồi quy GBM.

5.2 Mã nguồn cho Bài toán Phân loại

```

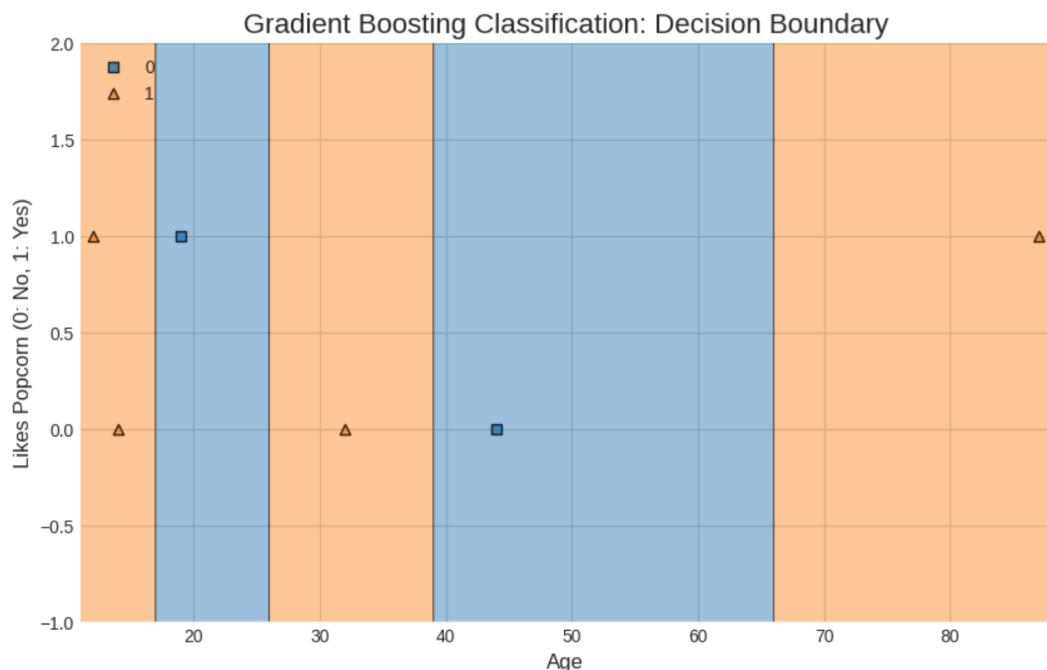
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.ensemble import GradientBoostingClassifier
5 from sklearn.preprocessing import LabelEncoder
6 from mlxtend.plotting import plot_decision_regions
7
8 # 1. Create DataFrame
9 data_cls = {
10     'Likes Popcorn': ['Yes', 'Yes', 'No', 'Yes', 'No', 'No'],
11     'Age': [12, 87, 44, 19, 32, 14],
12     'Favorite Color': ['Blue', 'Green', 'Blue', 'Red', 'Green', 'Blue'],
13     'Loves Troll 2': ['Yes', 'Yes', 'No', 'Yes', 'Yes']
14 }
15 df_cls = pd.DataFrame(data_cls)
16
17 # 2. Preprocessing (simplified for 2D visualization)
18 le_popcorn = LabelEncoder()
19 df_cls['Likes Popcorn'] = le_popcorn.fit_transform(df_cls['Likes Popcorn'])
20 y_encoder = LabelEncoder()
21 df_cls['Loves Troll 2'] = y_encoder.fit_transform(df_cls['Loves Troll 2'])
22
23 # Select 2 features for visualization
24 X_vis = df_cls[['Age', 'Likes Popcorn']].values
25 y_vis = df_cls['Loves Troll 2'].values
26
27 # 3. Build and Train the Model
28 gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=1, random_state=42)
29 gbc.fit(X_vis, y_vis)

```

```

30
31 # 4. Visualize the decision boundary
32 plt.style.use('seaborn-v0_8-whitegrid')
33 plt.figure(figsize=(10, 6))
34 plot_decision_regions(X_vis, y_vis, clf=gbc, legend=2)
35 plt.title('Gradient Boosting Classification: Decision Boundary', fontsize=16)
36 plt.xlabel('Age', fontsize=12)
37 plt.ylabel('Likes Popcorn (0: No, 1: Yes)', fontsize=12)
38 plt.show()

```



Hình 13: Đường biên quyết định của mô hình phân loại GBM.

6. Tinh chỉnh siêu tham số để đạt hiệu suất tối ưu

- **n_estimators:** Số lượng cây trong chuỗi.
- **learning_rate (η):** Tốc độ học. Có một sự đánh đổi quan trọng giữa learning_rate và n_estimators.
- **max_depth:** Độ sâu tối đa của mỗi cây. Với GBM, chúng ta thường ưu tiên các cây nông (1 đến 5).
- **subsample:** Tỷ lệ mẫu dữ liệu được sử dụng để huấn luyện mỗi cây (Stochastic Gradient Boosting).

7. So sánh với các thuật toán Ensemble khác

Bảng 6: So sánh Gradient Boosting, Random Forest, và AdaBoost.

Tiêu chí	Gradient Boosting	Random Forest [1]	AdaBoost
Thứ tự	Tuần tự	Song song	Tuần tự
Cơ chế học	Học trên phần dư (gradient)	Học trên mẫu bootstrap độc lập	Học trên trọng số của các điểm bị sai
Mục tiêu	Giảm bias trước, sau đó giảm variance	Chủ yếu giảm variance	Chủ yếu giảm bias
Độ sâu cây	Cây nông	Cây sâu	Cây rất nông (stumps)

8. Tổng kết

Gradient Boosting không chỉ là một thuật toán; nó là một framework mạnh mẽ và có tính tổng quát cao [3]. Bằng cách hiểu rõ cơ chế học tập theo gradient, chúng ta không chỉ nắm vững một công cụ mạnh mẽ mà còn có một nền tảng vững chắc để tiếp cận các thuật toán "hậu duệ" của nó như XGBoost [2], LightGBM và CatBoost, những "gã khổng lồ" đang thống trị nhiều cuộc thi về khoa học dữ liệu hiện nay.

Tài liệu

- [1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [3] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

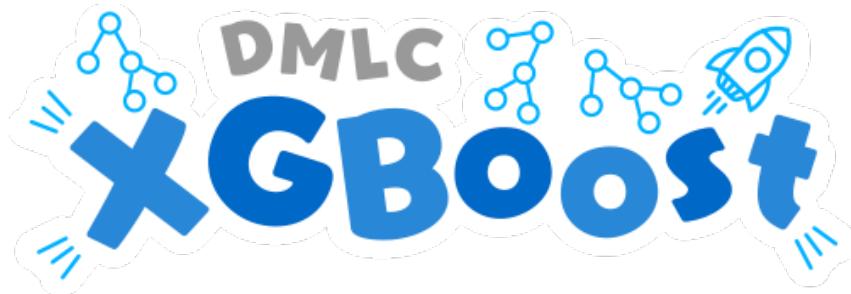
XGBoost: The most underrated algorithm

Võ Hoàng

Nếu bạn từng tham gia Kaggle – cộng đồng thi đấu dữ liệu lớn nhất thế giới – chắc hẳn bạn sẽ nhận ra một cái tên quen thuộc xuất hiện trong hầu hết các solution top đầu: XGBoost.

Không phải deep learning, cũng chẳng cần GPU khủng, XGBoost vẫn chinh phục trái tim của nhiều data scientist nhờ sự cân bằng tuyệt vời giữa tốc độ – độ chính xác – khả năng ứng dụng rộng rãi. Từ dự báo giá nhà, phát hiện gian lận thẻ tín dụng, cho đến dự báo nhu cầu năng lượng trong các thành phố lớn... đâu đâu cũng có dấu chân của XGBoost.

Điều thú vị là, đằng sau cái tên nghe khá “ngầu” ấy lại là một ý tưởng không hề xa lạ: Boosting – kỹ thuật kết hợp nhiều mô hình yếu để tạo nên một mô hình mạnh. Vậy điều gì đã biến XGBoost thành “ngôi sao” của giới Machine Learning, vượt xa so với những người tiền nhiệm như Random Forest hay AdaBoost?



Hình 14: XGBoost: In General

1. XGBoost là gì?

Nói một cách ngắn gọn, XGBoost (Extreme Gradient Boosting) là một thuật toán học máy thuộc họ ensemble learning – tức là kết hợp nhiều mô hình nhỏ để tạo thành một mô hình mạnh mẽ hơn.

Nếu Random Forest giống như việc ”hội ý” của nhiều cái cây độc lập để ra quyết định, thì XGBoost lại chọn con đường khác: nó xây dựng từng cây một cách tuần tự, và mỗi cây mới được sinh ra để sửa lỗi của cây trước đó.

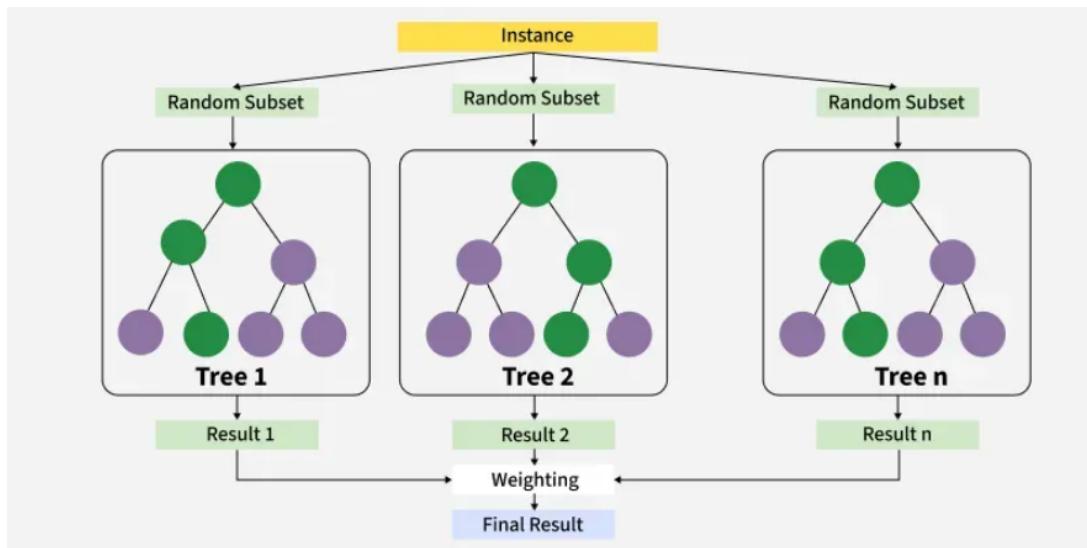
3.2. Classification – phân loại nhãn

Ngược lại, khi bài toán là “có bệnh hay không?”, “khách hàng churn hay không?”, ta cần classification.

Thay vì trực tiếp dự đoán giá trị, XGBoost sẽ dự đoán log(odds) – một cách biến đổi xác suất sang dạng dễ tính toán hơn.

Sau đó, log(odds) được chuyển ngược lại thành xác suất (ví dụ: 70% khách hàng sẽ rời bỏ dịch vụ).

Các cây liên tiếp giúp mô hình ngày càng phân biệt rõ ràng đâu là “có” và đâu là “không”.



Hình 15: XGBoost Classification

3.3. Điểm khác biệt tinh tế

Regression tập trung giảm khoảng cách dự đoán (residuals).

Classification thì tinh chỉnh log(odds) và tối ưu dựa trên xác suất.

Nói cách khác, XGBoost giống như một người huấn luyện giỏi: khi gặp học trò “sai số ít”, nó nhắc nhẹ; khi gặp học trò “sai số nhiều”, nó dồn sức chỉnh sửa. Và dù ở regression hay classification, nguyên lý “học từ lỗi sai” vẫn là trái tim của thuật toán này.

2. Behind the Scenes – Bí mật đằng sau nắp capo của XGBoost

Nếu phần trước giống như việc quan sát một chiếc xe đang chạy trên đường, thì giờ đây chúng ta mở nắp capo và nhìn vào “động cơ” – nơi XGBoost thực sự tạo nên sức mạnh.

4.1. Loss function – trái tim của XGBoost XGBoost học cách dự đoán bằng cách tối thiểu hóa hàm mất mát (loss function).

Với Regression, loss function thường là Mean Squared Error (MSE).

Với Classification, loss function là Log Loss, dựa trên xác suất đúng/sai.

Điểm đặc biệt: XGBoost không chỉ nhìn vào gradient bậc 1 (giống như Gradient Boosting truyền thống) mà còn dùng thêm hessian (đạo hàm bậc 2). Nhờ đó, quá trình tối ưu trở nên chính xác hơn – giống như việc bạn không chỉ biết mình đang leo dốc, mà còn biết độ dốc thay đổi nhanh thế nào.

4.2. Regularization – bí quyết tránh overfitting Nếu chỉ tối ưu loss, mô hình dễ “học vẹt” dữ liệu huấn luyện (overfitting). Để ngăn điều đó, XGBoost thêm vào 2 “chiếc phanh”:

λ (lambda): giảm độ phức tạp bằng cách kéo các giá trị về gần 0.

γ (gamma): yêu cầu một mức “lợi ích tối thiểu” trước khi tạo thêm nhánh mới trên cây.

Kết quả: cây gọn gàng, dễ tổng quát hơn cho dữ liệu mới.

4.3. Cover & Similarity Score – cách XGBoost đánh giá một nhánh Trong lúc chia nhánh, XGBoost tính Similarity Score – thước đo cho biết việc gom nhóm residuals có “đáng” hay không. Ngoài ra, nó còn dùng Cover để đảm bảo mỗi nhánh có đủ dữ liệu đại diện, thay vì chỉ dựa vào một vài điểm lẻ loi.

4.4. Taylor Expansion – bí kíp toán học Tối ưu loss function trực tiếp là cực khó. Vì vậy, XGBoost dùng Khai triển Taylor bậc 2 để gần đúng. Đây chính là mấu chốt giúp thuật toán vừa nhanh, vừa chính xác.

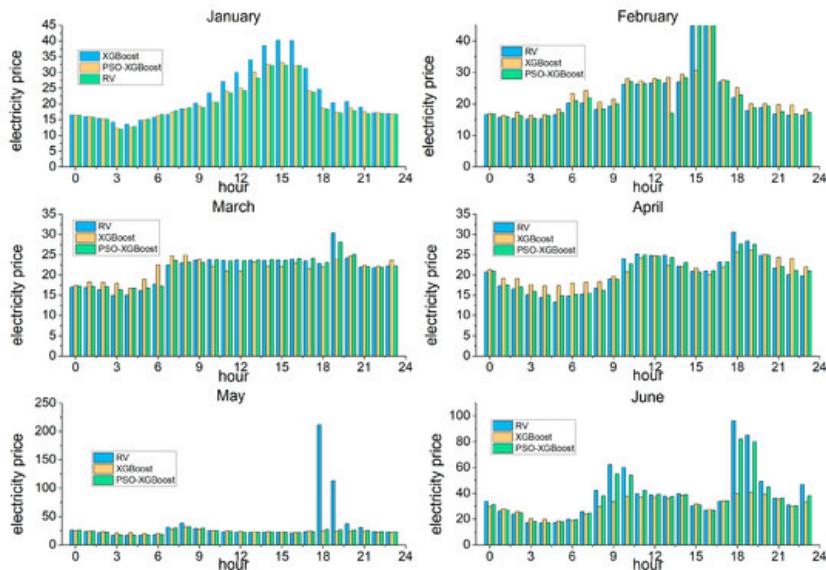
Nói vui thì, XGBoost giống như một đầu bếp: Loss function là công thức nấu ăn, Gradient & Hessian là nguyên liệu, còn Regularization là gia vị giúp món ăn không bị “quá mặn” (overfit).

3. Case Study – Time Series Forecasting với XGBoost

Để thấy XGBoost “ra trận” như thế nào, hãy thử một bài toán thực tế: dự báo nhu cầu năng lượng của một thành phố.

5.1. Bài toán đặt ra

Giả sử ta có dữ liệu về mức tiêu thụ điện hằng ngày của các hộ gia đình. Nhiệm vụ là dự đoán nhu cầu năng lượng trong tương lai – một công việc vô cùng quan trọng cho ngành điện lực, vì nó giúp tối ưu hoá việc sản xuất, phân phối và lưu trữ.



Hình 16: Mức điện tiêu thụ

5.2. Cách tiếp cận với XGBoost

Tiền xử lý dữ liệu: điền giá trị thiếu, làm sạch dữ liệu tiêu thụ điện.

Huấn luyện mô hình: dùng XGBoost để học từ dữ liệu quá khứ.

Đánh giá mô hình: các chỉ số như MAE (Mean Absolute Error), MSE (Mean Squared Error), và MAPE (Mean Absolute Percentage Error) cho thấy độ chính xác.

5.3. Kết quả ban đầu

MAE: ~0.77

MSE: ~1.50

MAPE: ~19.7%

Nói cách khác, mô hình đã dự đoán tương đối tốt, nhưng vẫn còn nhiều dư địa để cải thiện.

5.4. Bí quyết cải thiện: thêm dữ liệu thời tiết

Khi nhóm nghiên cứu bổ sung dữ liệu thời tiết (từ London Weather Dataset), hiệu quả tăng vọt:

MAE giảm từ 0.77 → 0.42

MSE giảm từ 1.50 → 0.86

MAPE giảm từ 19.7% → 16.4%

Kết quả này minh chứng một điều quan trọng: trong học máy, dữ liệu thường quan trọng không kém – thậm chí còn quan trọng hơn – mô hình.

5.5. Thông điệp rút ra

XGBoost mạnh mẽ, nhưng nếu không có dữ liệu phù hợp, nó cũng khó lòng tạo ra dự đoán chính xác. Khi được “cho ăn” dữ liệu giàu thông tin hơn, XGBoost có thể biến thành một công cụ dự báo cực kỳ đáng tin cậy.

4. Ưu – Nhược điểm của XGBoost

Sau khi đi qua lý thuyết, cơ chế hoạt động và cả case study thực tế, giờ là lúc chúng ta “ngồi lại” để cân nhắc: XGBoost có thật sự phù hợp với mọi bài toán không?

6.1 Ưu điểm

Hiệu năng cao Tối ưu tốt, hỗ trợ tính toán song song, nên chạy nhanh hơn hẳn so với nhiều thuật toán boosting truyền thống.

Độ chính xác vượt trội Nhờ tận dụng gradient + hessian và regularization, XGBoost thường đạt độ chính xác cao, đặc biệt trên các bộ dữ liệu structured/tabular.

Linh hoạt Dùng được cho cả regression, classification, ranking, thậm chí time series.

Chống overfitting khá tốt Nhờ có λ (lambda) và γ (gamma) kiểm soát độ phức tạp của cây, giúp mô hình tổng quát hóa dữ liệu mới tốt hơn.

Thân thiện với Kaggle-er và Data Scientist Nhiều thư viện tích hợp sẵn (Python, R, Julia), cộng đồng đông đảo, tài liệu phong phú.

6.2. Nhược điểm

Dễ bị “over-engineering” Có quá nhiều hyperparameter để tune (learning rate, max_depth, min_child_weight, subsample...), nên dễ khiến người mới rối rắm.

Khó diễn giải Mô hình ensemble nhiều cây → khó giải thích tại sao mô hình đưa ra quyết định (so với logistic regression chẳng hạn).

Không phải lúc nào cũng tối ưu nhất Với dữ liệu unstructured (ảnh, âm thanh, văn bản), deep learning thường vượt trội hơn.

Tốn tài nguyên nếu dataset rất lớn Dù nhanh hơn boosting truyền thống, XGBoost vẫn “ngốn” RAM/GPU đáng kể khi xử lý hàng trăm triệu dòng dữ liệu.

6.3. So sánh nhanh

- So với LightGBM: XGBoost thường ổn định hơn, nhưng LightGBM nhanh hơn trên dataset cực lớn.
- So với CatBoost: CatBoost xử lý categorical features “out-of-the-box” tốt hơn, nhưng XGBoost vẫn phổ biến hơn nhờ cộng đồng rộng và tính linh hoạt.

5. Ứng dụng thực tế & Kết luận

XGBoost đã chứng minh sức mạnh của mình trong vô số lĩnh vực:

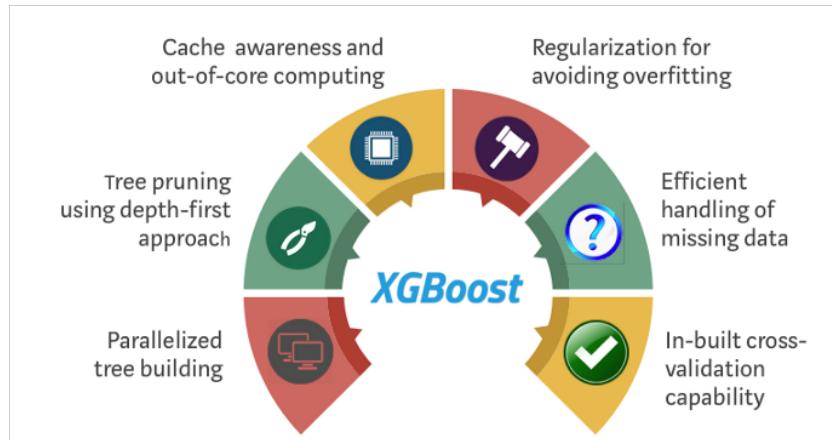
- Tài chính: phát hiện gian lận thẻ tín dụng, dự đoán rủi ro khách hàng.
- Y tế: dự đoán khả năng tái nhập viện, phân loại bệnh dựa trên hồ sơ bệnh án.
- Marketing: phân khúc khách hàng, dự đoán hành vi mua sắm.
- Năng lượng & môi trường: dự báo nhu cầu điện, lượng khí thải, hay thậm chí xu hướng thời tiết.

7.1. Khi nào nên dùng XGBoost?

- Khi dữ liệu của bạn chủ yếu là structured/tabular (số liệu, bảng tính, giao dịch).
- Khi bạn cần một mô hình mạnh mẽ, nhưng vẫn muốn huấn luyện tương đối nhanh và dễ triển khai.
- Khi bài toán đòi hỏi giải thích mức độ vừa phải (dùng thêm SHAP/Feature Importance).

7.2. Khi nào nên cân nhắc giải pháp khác?

- Nếu dữ liệu là ảnh, âm thanh, văn bản dài → deep learning thường vượt trội.
- Nếu bạn muốn dễ dàng giải thích quyết định của mô hình → logistic regression hoặc decision tree đơn giản có thể phù hợp hơn.



Hình 17: XGBoost: Kết luận

7.3. Kết luận

XGBoost không phải là “thuốc tiên” cho mọi bài toán, nhưng nó là một trong những công cụ đa năng và hiệu quả nhất trong hộp đồ nghề của một data scientist. Nó mạnh mẽ nhờ sự kết hợp giữa boosting + regularization + tối ưu hóa thông minh, và khi được cung cấp dữ liệu chất lượng, XGBoost có thể trở thành “chiếc xe phân khối lớn” đưa bạn băng qua những con đường dữ liệu phức tạp nhất.

Vậy lần tới, khi đứng trước một bài toán dự báo giá trị hay phân loại, hãy tự hỏi: “Liệu XGBoost có phải là chiếc chìa khoá mở ra lời giải nhanh và chuẩn xác cho mình không?”

Xây dựng trang web đầu tiên với FastAPI: Hướng dẫn toàn diện cho người mới bắt đầu

Trịnh Nguyễn Huy Hoàng

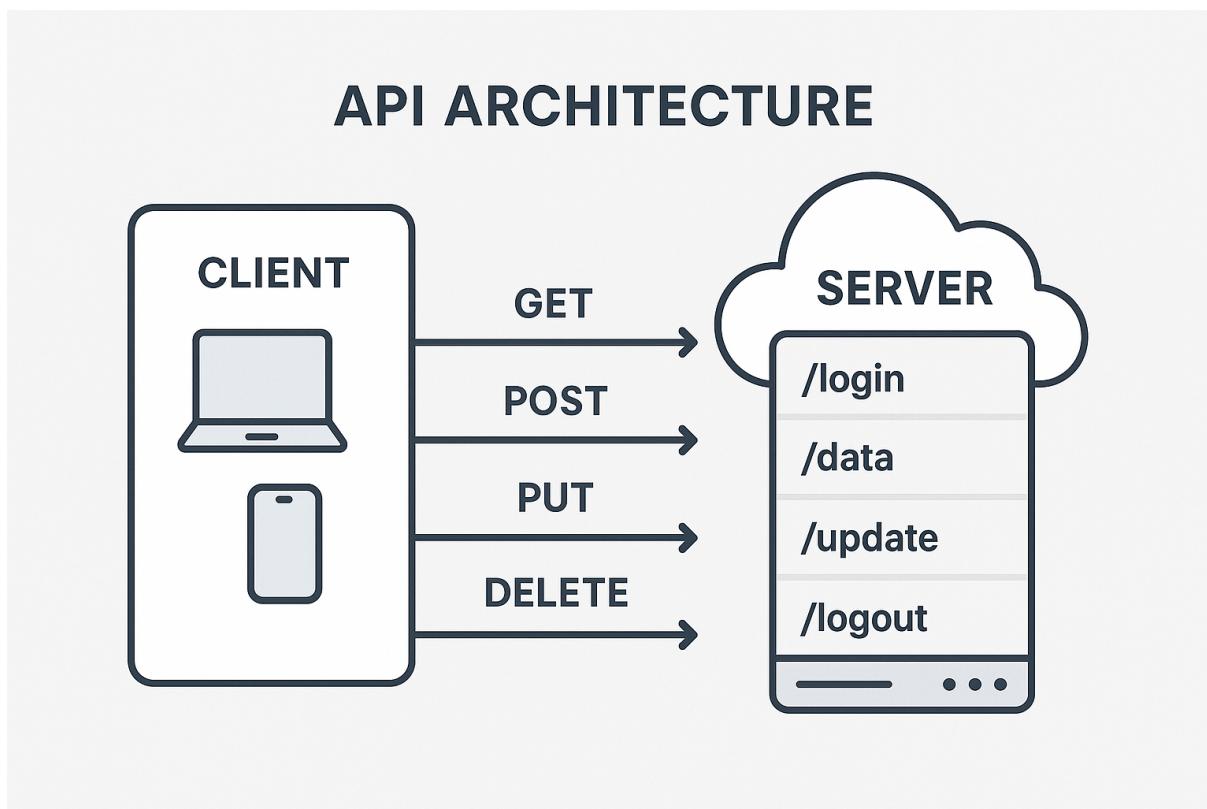
Tóm tắt nội dung

Được xây dựng để giải quyết những thách thức của kỷ nguyên số, FastAPI nổi bật với tốc độ vượt trội, hiệu quả tối đa và khả năng tự động hóa đáng kinh ngạc. Bài viết này là một cảm nang toàn diện, dẫn dắt từ những khái niệm nền tảng nhất cho đến việc xây dựng một ứng dụng web hoàn chỉnh, có tính ứng dụng cao, đặc biệt là trong lĩnh vực trí tuệ AI.

1. Giới thiệu về FastAPI

1.1 API là gì?

API (Application Programming Interface) là một giao diện phần mềm cho phép hai ứng dụng khác nhau giao tiếp với nhau. Có thể hình dung API như một người phục vụ tại nhà hàng: khi một người khách muốn gọi món, họ không cần phải vào bếp mà chỉ cần đưa yêu cầu cho người phục vụ. Người phục vụ sẽ nhận yêu cầu, chuyển đến bếp và mang món ăn trở lại cho khách. API cũng tương tự, nó giúp các lập trình viên sử dụng các chức năng phức tạp, chẳng hạn như dữ liệu thời tiết hoặc bản đồ, mà không cần hiểu chi tiết bên trong của hệ thống đó. Điều này cho phép chúng ta xây dựng các ứng dụng mới một cách nhanh chóng bằng cách ghép nối các "khối chức năng" (API) với nhau, giống như việc lắp ráp các viên gạch Lego.



Hình 18: Sơ đồ kiến trúc API với client và server

REST API (Representational State Transfer) là một kiểu thiết kế API sử dụng các phương thức HTTP tiêu chuẩn như GET, POST, PUT, DELETE để thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên dữ liệu. REST API có những đặc điểm quan trọng:

- **Stateless:** Mỗi request đều độc lập, không lưu trữ thông tin giữa các request
- **Uniform Interface:** Sử dụng các URL để xác định resources và HTTP methods để thao tác
- **Client-Server Architecture:** Tách biệt rõ ràng giữa client và server

1.2 ASGI Server và Unicorn

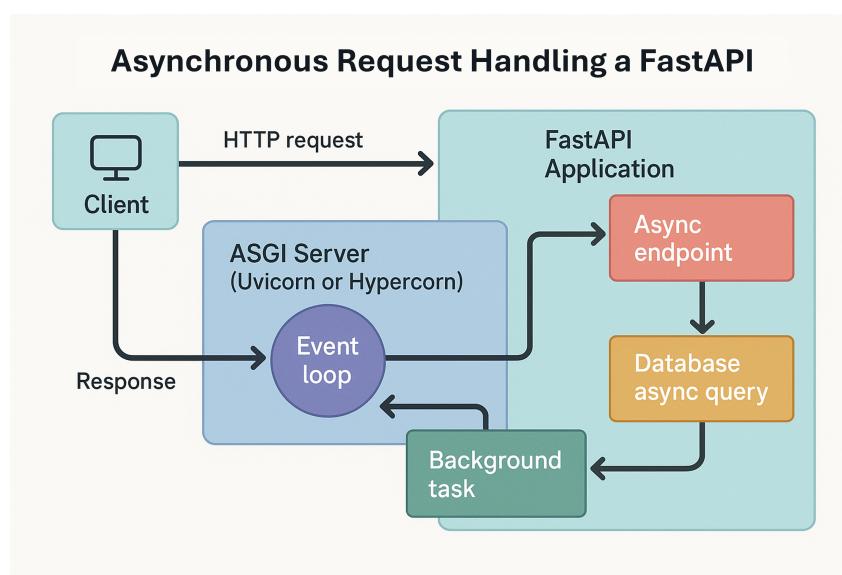
Trước khi FastAPI xuất hiện, hầu hết các framework web Python đều dựa trên giao thức *WSGI* (*Web Server Gateway Interface*), được thiết kế cho mô hình xử lý request đồng bộ. Điều này có nghĩa là khi một request đến, server sẽ xử lý nó từ đầu đến cuối trước khi có thể chấp nhận request tiếp theo. Phương pháp này không hiệu quả khi gặp các tác vụ tốn thời gian như truy vấn cơ sở dữ liệu hoặc gọi đến một API bên ngoài, vì nó sẽ làm "tắc nghẽn" toàn bộ hệ thống.

Để khắc phục nhược điểm này, tiêu chuẩn **ASGI (Asynchronous Server Gateway Interface)** đã ra đời, mở đường cho lập trình bất đồng bộ. FastAPI là một trong những framework tiên phong tuân thủ tiêu chuẩn này.

Điều quan trọng cần nắm là FastAPI không phải là một máy chủ web độc lập. Thay vào đó, nó là một "ứng dụng" tuân thủ giao thức ASGI và cần một máy chủ web ASGI để chạy. Uvicorn là một máy chủ web ASGI hiệu suất cao, được tạo ra để chuyên biệt cho các ứng dụng như FastAPI.

Uvicorn chịu trách nhiệm lắng nghe các request đến từ client và chuyển chúng đến FastAPI để xử lý. Việc tách biệt này cho phép FastAPI tập trung vào việc xử lý logic nghiệp vụ, trong khi Uvicorn đảm nhận vai trò quản lý kết nối và tối ưu hóa hiệu suất.

Sơ đồ kiến trúc của ASGI Server được minh họa như hình 19. Cần lưu ý rằng, Uvicorn hỗ trợ HTTP/1.1 và WebSockets, cho phép xử lý nhiều request đồng thời một cách hiệu quả. Điều này đặc biệt quan trọng khi xây dựng các API cần xử lý nhiều request cùng lúc.



Hình 19: Sơ đồ kiến trúc ASGI server với xử lý bất đồng bộ

1.3 Hiểu rõ async và await

Lập trình bất đồng bộ là một khái niệm cốt lõi của FastAPI. Không giống như lập trình đa luồng (multi-threading) hay đa tiến trình (multi-processing), lập trình bất đồng bộ trong Python là một kỹ thuật ”đơn luồng, đa tác vụ” (single-threaded, multi-tasking).

Hãy tưởng tượng bạn đang chơi cờ với 24 người cùng một lúc. Thay vì chơi một ván cờ duy nhất từ đầu đến cuối (mô hình đồng bộ), bạn sẽ di chuyển từ bàn này sang bàn khác, thực hiện một nước cờ và ngay lập tức chuyển sang bàn tiếp theo, cho phép đối thủ của mình suy nghĩ. Tương tự, trong lập trình bất đồng bộ, khi một tác vụ I/O-bound (như truy vấn cơ sở dữ liệu hoặc chờ phản hồi từ một API khác) đang chờ, await sẽ tạm dừng việc thực thi của hàm hiện tại và chuyển quyền điều khiển cho event loop (vòng lặp sự kiện) để xử lý các tác vụ khác.

- *async def* được sử dụng để định nghĩa một *coroutine* function, một loại hàm có thể bị tạm dừng.
- *await* chỉ có thể được sử dụng bên trong một hàm *async def* và là từ khóa để tạm dừng việc thực thi và nhường quyền điều khiển.

Điều này cực kỳ quan trọng đối với các ứng dụng web vì phần lớn thời gian chờ đợi là do các tác vụ I/O. Bằng cách tận dụng *async/await*, FastAPI có thể xử lý hàng ngàn kết nối đồng thời trên một luồng duy nhất, tránh được chi phí và sự phức tạp của việc quản lý đa luồng. Chính nhờ cơ chế này, FastAPI được đánh giá là một trong những framework Python nhanh nhất, sánh ngang với Node.js và Go.

1.4 FastAPI, Flask và Django

Khi bắt đầu một dự án web Python, ta thường băn khoăn giữa ba lựa chọn phổ biến: Django, Flask và FastAPI.

- Django là một framework monolithic (đơn khối) hoặc ”all-in-one”. Nó đi kèm với mọi thứ cần thiết để xây dựng một ứng dụng lớn, từ ORM (lớp ánh xạ đối tượng-quan hệ) để tương tác với cơ sở dữ liệu, đến giao diện quản trị viên tích hợp và hệ thống template. Django phù hợp với các dự án phức tạp, cần triển khai nhanh các chức năng cơ bản, nhưng có thể trở nên cồng kềnh đối với các ứng dụng nhỏ.
- Flask là một framework tối giản và linh hoạt. Nó chỉ cung cấp những chức năng cốt lõi nhất, cho phép lập trình viên tự do lựa chọn và tích hợp các thư viện bên ngoài để thêm tính năng. Sự linh hoạt này là một ưu điểm, nhưng cũng là một nhược điểm đối với người mới, vì họ phải tự tìm hiểu và kết hợp các thành phần, ví dụ như thư viện để xử lý CORS.
- FastAPI được thiết kế như một framework hiện đại, tận dụng sức mạnh của các thư viện tốt nhất như Starlette cho routing và Pydantic cho validation. Nó cung cấp các tính năng ”có sẵn” (out-of-the-box) như tự động xác thực dữ liệu và tạo tài liệu API tương tác, giúp giảm thiểu lỗi và tăng tốc độ phát triển mà không mang sự phức tạp của Django.

Quan trọng hơn cả, Python hiện là ngôn ngữ chính trong lĩnh vực Khoa học dữ liệu/ Học máy, nên FastAPI – với khả năng kết hợp giữa hiệu năng web và code Python ML – trở thành lựa chọn tuyệt vời để triển khai các mô hình AI thành dịch vụ web. Thay vì phải dùng các giải pháp phức tạp hoặc ngôn ngữ khác, các nhà nghiên cứu AI có thể dùng FastAPI để “đóng gói” mô hình của mình thành API phục vụ cho ứng dụng thực tế một cách nhanh chóng.

2. Ứng dụng CRUD đơn giản với FastAPI

2.1 Ứng dụng FastAPI cơ bản

Để bắt đầu với FastAPI, chúng ta cần cài đặt framework:

```
1 pip install fastapi uvicorn
```

Sau đó, hãy tạo một tệp tin *main.py* với đoạn code đơn giản sau:

```
1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5 @app.get("/")
6 def read_root():
7     return {"Hello": "World"}
```

Để chạy ứng dụng, hãy sử dụng lệnh sau:

```
1 uvicorn main:app --reload
```

Lệnh *-reload* sẽ tự động khởi động lại server khi bạn thay đổi code. Sau khi chạy, bạn có thể truy cập *http://127.0.0.1:8000/* để thấy kết quả.

2.2 Tổ chức code với APIRouter

Khi ứng dụng lớn dần, việc đặt tất cả các endpoint vào một file *main.py* sẽ làm cho code trở nên “cồng kềnh” và khó quản lý. FastAPI cung cấp một giải pháp thanh lịch cho vấn đề này thông qua APIRouter.

APIRouter cho phép bạn nhóm các endpoint liên quan lại với nhau vào các module riêng biệt. Ví dụ, bạn có thể tạo một file *routers/items.py* để chứa tất cả các endpoint liên quan đến item, và sau đó import nó vào file *main.py*. Đây không chỉ là một cách để tổ chức code, mà còn là một yếu tố kiến trúc quan trọng cho tính linh hoạt và khả năng mở rộng. Bằng cách tách biệt các nhóm chức năng, APIRouter giúp ngăn chặn tình trạng “monolithic” (đơn khối) và cho phép các dự án phát triển theo mô hình microservices một cách tự nhiên.

FastAPI sử dụng decorators để định nghĩa routes. Mỗi route tương ứng với một HTTP method và path:

```
1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5 @app.get("/items")
6 async def get_items():
7     return {"items": []}
8
9 @app.post("/items")
10 async def create_item(item: dict):
```

```
11 return {"message": "Item created", "item": item}
12
13 @app.put("/items/{item_id}")
14 async def update_item(item_id: int, item: dict):
15     return {"message": f"Item {item_id} updated", "item": item}
16
17 @app.delete("/items/{item_id}")
18 async def delete_item(item_id: int):
19     return {"message": f"Item {item_id} deleted"}
```

Ta có thể tổ chức chương trình trên tốt hơn bằng cách sử dụng APIRouter như sau:

```
1 from fastapi import APIRouter
2
3 router = APIRouter(prefix="/items", tags=["items"])
4
5 @router.get("/")
6 async def get_items():
7     return {"items": []}
8
9 @router.post("/")
10 async def create_item(item: dict):
11     return {"message": "Item created"}
```

2.3 Tương tác với API và Pydantic

Trong quá trình phát triển web, các HTTP request thường sử dụng các phương thức chính để giao tiếp:

- GET: Được sử dụng để lấy dữ liệu.
- POST: Dùng để tạo một tài nguyên mới trên server.
- PUT: Dùng để cập nhật một tài nguyên đã có.
- DELETE: Dùng để xóa một tài nguyên.

FastAPI sử dụng một thư viện mạnh mẽ tên là *Pydantic* để tự động hóa việc xác thực dữ liệu đầu vào. Bằng cách định nghĩa các model dữ liệu bằng Pydantic BaseModel và sử dụng type hints trong các hàm endpoint, FastAPI sẽ tự động kiểm tra Request Body, Path parameters, và Query parameters. Nếu dữ liệu không hợp lệ (ví dụ: thiếu trường, sai kiểu dữ liệu), FastAPI sẽ tự động trả về một lỗi JSON chi tiết, giúp giảm thiểu đáng kể các lỗi do lập trình viên gây ra.

Dưới đây là một ví dụ code đơn giản cho ứng dụng CRUD với in-memory storage (lưu trữ trong bộ nhớ):

```
1 from fastapi import FastAPI, HTTPException
2 from pydantic import BaseModel
3 from typing import Dict
4
5 app = FastAPI()
6
7 class Item(BaseModel):
8     name: str
9     price: float
10    is_offer: bool = None
```

```
11
12 items_db: Dict[int, Item] = {}
13 next_item_id = 1
14
15 @app.post("/items/", status_code=201)
16 def create_item(item: Item):
17     global next_item_id
18     item_id = next_item_id
19     items_db[item_id] = item
20     next_item_id += 1
21     return {"message": "Item created successfully", "item_id": item_id, "item": item}
22
23 @app.get("/items/", response_model=Dict[int, Item])
24 def read_items():
25     return items_db
26
27 @app.get("/items/{item_id}", response_model=Item)
28 def read_item(item_id: int):
29     if item_id not in items_db:
30         raise HTTPException(status_code=404, detail="Item not found")
31     return items_db[item_id]
32
33 @app.put("/items/{item_id}")
34 def update_item(item_id: int, item: Item):
35     if item_id not in items_db:
36         raise HTTPException(status_code=404, detail="Item not found")
37     items_db[item_id] = item
38     return {"message": "Item updated successfully", "item_id": item_id, "item": item}
39
40 @app.delete("/items/{item_id}")
41 def delete_item(item_id: int):
42     if item_id not in items_db:
43         raise HTTPException(status_code=404, detail="Item not found")
44     del items_db[item_id]
45     return {"message": "Item deleted successfully", "item_id": item_id}
```

2.4 Tài liệu tự động (Documentation)

Một tính năng được lòng người dùng của FastAPI là tự động sinh giao diện tài liệu API. Khi chạy ứng dụng, FastAPI cung cấp sẵn giao diện tương tác tại đường dẫn `/docs` (Swagger UI) và `/redoc` (ReDoc). Bạn có thể mở trình duyệt tới `http://localhost:8000/docs` để xem danh sách tất cả các endpoint, các tham số yêu cầu và mẫu dữ liệu vào/ra. Giao diện này còn cho phép bạn thử gọi trực tiếp API ngay trên trang (nhập tham số và bấm “Execute”). Việc tự động tạo documentation chi tiết cho API này của FastAPI giúp việc hiểu và sử dụng API trở nên dễ dàng. Điều này đặc biệt hữu ích cho team AI ít kinh nghiệm về web – bạn có thể nhanh chóng kiểm thử API của mình mà không cần viết thêm dòng code tài liệu nào.

3. Nâng cao trải nghiệm với Response và Templating

3.1 Response & Templating

Khi xây dựng API, việc quy định rõ phản hồi (response) trả về rất quan trọng để client hiểu và xử lý đúng. FastAPI cho phép chúng ta chỉ định Response Model (mô hình dữ liệu phản hồi)

như đã thấy ở phần CRUD – nhờ đó đảm bảo dữ liệu trả về đúng cấu trúc mong muốn và được mô tả trong tài liệu API. Ngoài ra, có thể tùy chỉnh HTTP Status Code cho từng tình huống cụ thể.

Bên cạnh JSON response, FastAPI cũng hỗ trợ trả về HTML thông qua cơ chế templating. Với các ứng dụng web có giao diện, ta có thể sử dụng *Jinja2 template* giống Flask. FastAPI tích hợp sẵn *Jinja2Templates* cho phép render file HTML và trả về trong response. Chẳng hạn, ta tạo thư mục *templates/* chứa file *index.html*, sau đó trong code:

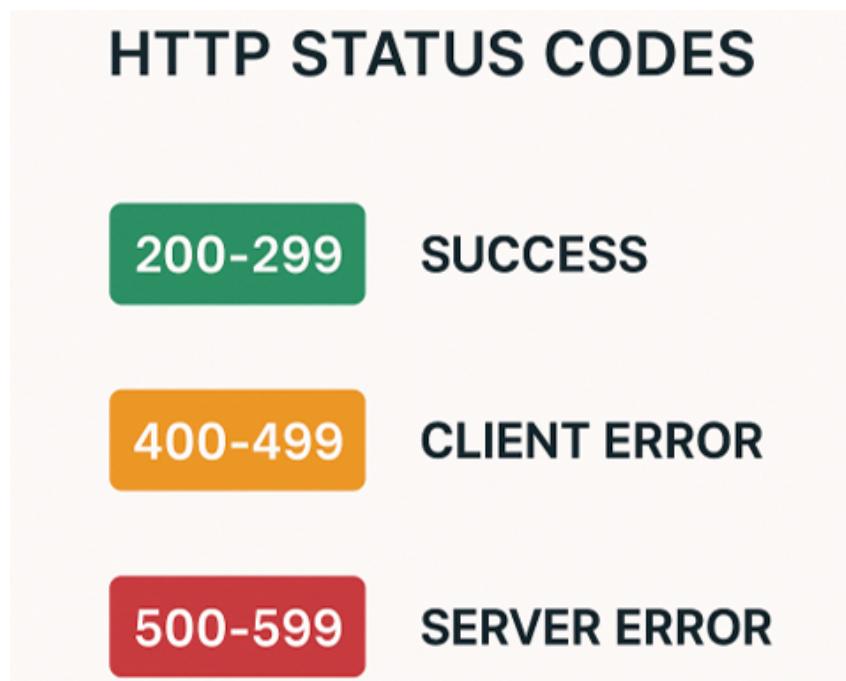
```
1 from fastapi.templating import Jinja2Templates
2 from starlette.requests import Request
3
4 templates = Jinja2Templates(directory="templates")
5
6 @app.get("/hello")
7 async def hello(request: Request, name: str = "AI"):
8     return templates.TemplateResponse("index.html", {"request": request, "name": name})
```

Ở ví dụ trên, endpoint */hello* sẽ render template *index.html* với biến *name* truyền vào. Biến *request* luôn cần có để *TemplateResponse* hoạt động. Như vậy FastAPI có thể vừa phục vụ API JSON vừa phục vụ trang web nếu cần thiết, mặc dù mục tiêu chính của FastAPI vẫn là xây dựng API. (Lưu ý: Để sử dụng tính năng này, cần cài đặt thư viện *jinja2* trong môi trường của bạn).

3.2 HTTP Status Code

HTTP Status Code là các mã số được server sử dụng để thông báo kết quả của một request cho client. Việc hiểu và sử dụng đúng các mã này là rất quan trọng để xây dựng một API chuyên nghiệp. Các mã được chia thành 5 nhóm chính:

- **2xx (Success)**: Request đã được xử lý thành công.
- **4xx (Client Error)**: Lỗi đến từ phía client (ví dụ: *request* không hợp lệ).
- **5xx (Server Error)**: Lỗi đến từ phía server.



Hình 20: Biểu đồ HTTP Status Codes phổ biến

Dưới đây là bảng tổng hợp các HTTP Status Code cơ bản mà mọi lập trình viên web nên biết:

Mã	Tên	Ý nghĩa	Ví dụ sử dụng
200	OK	Request đã được xử lý thành công.	GET dữ liệu thành công.
201	Created	Một tài nguyên mới đã được tạo thành công.	POST dữ liệu thành công.
204	No Content	Request đã xử lý thành công nhưng không có nội dung để trả về.	DELETE dữ liệu thành công.
400	Bad Request	Server không thể hiểu request do cú pháp không hợp lệ.	Request body sai định dạng.
404	Not Found	Server không tìm thấy tài nguyên được yêu cầu.	URL sai hoặc tài nguyên không tồn tại.
500	Internal Server Error	Một lỗi không mong muốn xảy ra trên server.	Lỗi logic trong code.

Bảng 7: Các HTTP Status Codes cơ bản cần biết

3.3 Cấu trúc ứng dụng FastAPI

Khi dự án lớn hơn, việc tuân thủ một cấu trúc thư mục hợp lý là rất cần thiết. Một cấu trúc dự án đơn giản nhưng hiệu quả cho FastAPI có thể được minh họa như sau:

CẤU TRÚC DỰ ÁN FASTAPI

Cấu trúc thư mục dự án
main.py
routers/
items.py
schemas/
item.py

- *main.py*: File chính khởi tạo ứng dụng FastAPI và include các router.
- *routers/*: Thư mục này chứa các APIRouter để tổ chức các endpoint theo chức năng. Ví dụ, *routers/items.py* sẽ chứa tất cả các endpoint liên quan đến item.
- *schemas/*: Chứa các Pydantic BaseModel để định nghĩa cấu trúc dữ liệu cho request và response.
- *dependencies/* (Tùy chọn): Chứa các hàm dependency injection để quản lý các tài nguyên như kết nối cơ sở dữ liệu.

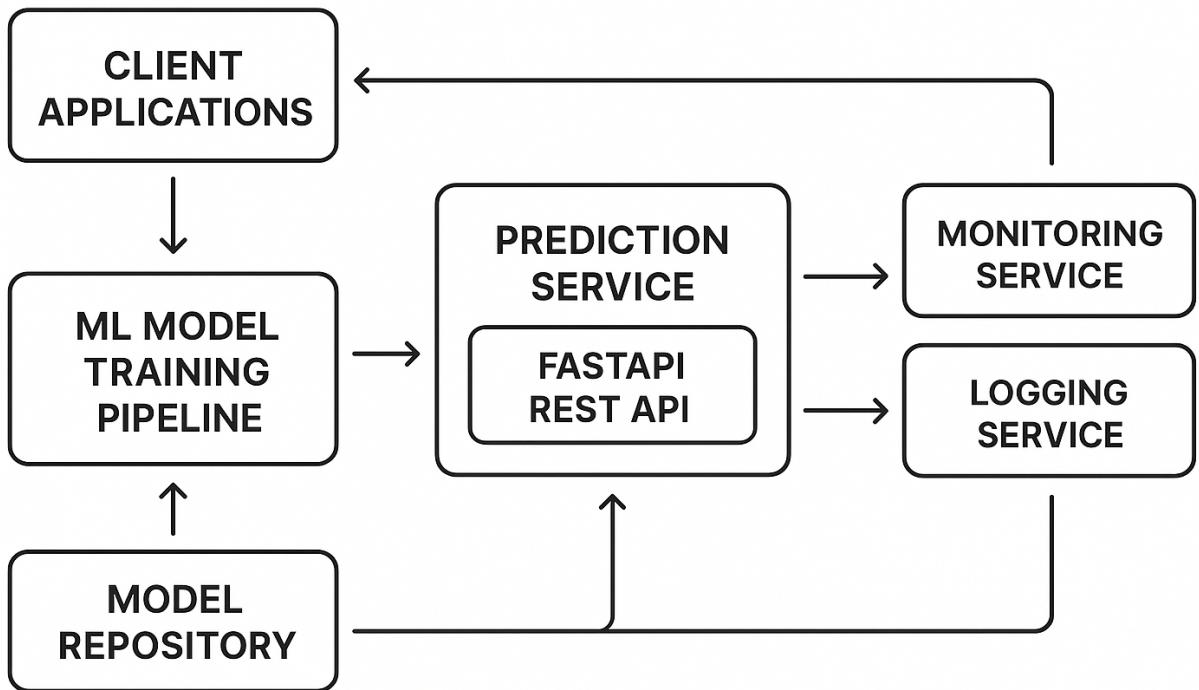
Để mở rộng ứng dụng, bạn có thể tích hợp cơ sở dữ liệu. Ví dụ, với MongoDB, bạn cần cài đặt *pymongo*, định nghĩa các Pydantic models cho dữ liệu, tạo kết nối database khi ứng dụng khởi động và sử dụng các phương thức của PyMongo (*insert_one*, *find*) trong các endpoint CRUD.

3.4 Độ bảo mật và trung gian (Middleware)

FastAPI hỗ trợ middleware cho phép can thiệp vào mọi request/response (ví dụ kích hoạt CORS, logging request, xử lý xác thực JWT,...). Ta cũng có thể định nghĩa Dependency để tiêm các logic chung (như kết nối database, xác thực người dùng) vào các route một cách tiện lợi. Những tính năng này giúp cấu trúc code sạch sẽ và giàu tính modular, đặc biệt hữu ích khi dự án web AI của bạn dần lớn lên.

4. Case Study: AI Text Classification API

Trong phần này, chúng ta sẽ xây dựng một ứng dụng thực tế sử dụng FastAPI để deploy một mô hình AI phân loại văn bản.



Hình 21: Sơ đồ triển khai mô hình AI với FastAPI

4.1 Thiết lập môi trường

```
1 pip install fastapi uvicorn scikit-learn pandas numpy joblib
```

4.2 Tạo và train mô hình

Đầu tiên, chúng ta tạo file `train_model.py` để train một mô hình phân loại văn bản đơn giản:

```

1 # train_model.py
2 import pandas as pd
3 import joblib
4 from sklearn.feature_extraction.text import TfidfVectorizer
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.pipeline import Pipeline
7 from sklearn.model_selection import train_test_split
8
9 # Sample data for sentiment analysis
10 data = [
11     ("I love this product", "positive"),
12     ("This is amazing", "positive"),
13     ("Great quality", "positive"),
14     ("Best purchase ever", "positive"),
15     ("Excellent service", "positive"),
  
```

```

16     ("I hate this", "negative"),
17     ("Terrible quality", "negative"),
18     ("Waste of money", "negative"),
19     ("Very disappointed", "negative"),
20     ("Poor service", "negative"),
21     ("It's okay", "neutral"),
22     ("Average product", "neutral"),
23     ("Not bad", "neutral"),
24     ("Could be better", "neutral"),
25     ("Decent quality", "neutral")
26 ]
27
28 # Create DataFrame
29 df = pd.DataFrame(data, columns=['text', 'sentiment'])
30
31 # Split data
32 X_train, X_test, y_train, y_test = train_test_split(
33     df['text'], df['sentiment'], test_size=0.2, random_state=42
34 )
35
36 # Create pipeline
37 model = Pipeline([
38     ('tfidf', TfidfVectorizer(max_features=1000)),
39     ('classifier', LogisticRegression())
40 ])
41
42 # Train model
43 model.fit(X_train, y_train)
44
45 # Save model
46 joblib.dump(model, 'sentiment_model.pkl')
47 print("Model trained and saved successfully!")

```

Giải thích: Code này thực hiện một quy trình phân loại văn bản cơ bản với các bước sau:

1. **Chuẩn bị Dữ liệu:** Dữ liệu mẫu về các câu văn và nhãn cảm xúc tương ứng (positive, negative, neutral) được tạo và lưu vào một DataFrame của thư viện Pandas.
2. **Chia Dữ liệu:** Tập dữ liệu được chia thành hai phần: tập huấn luyện (training set) để mô hình học và tập kiểm tra (test set) để đánh giá mô hình.
3. **Xây dựng Pipeline:** Một Pipeline của scikit-learn được tạo ra. Đây là một công cụ mạnh mẽ để kết hợp các bước xử lý dữ liệu và mô hình vào một chuỗi duy nhất, giúp quy trình trở nên gọn gàng và hiệu quả hơn. Pipeline này bao gồm hai bước chính:
 - *TfidfVectorizer*: Chuyển đổi văn bản thành các vector số học.
 - *LogisticRegression*: Một mô hình phân loại để học và dự đoán cảm xúc.
4. **Huấn luyện Mô hình:** Pipeline này được huấn luyện bằng cách sử dụng tập dữ liệu huấn luyện đã chia ở trên.
5. **Lưu Mô hình:** Cuối cùng, mô hình đã huấn luyện được lưu vào một file (.pkl) bằng thư viện joblib, giúp có thể sử dụng lại mô hình này mà không cần phải huấn luyện lại.

4.3 Tạo mô hình Pydantic

```

1 # models.py
2 from pydantic import BaseModel
3 from typing import List, Dict
4
5 class TextInput(BaseModel):
6     text: str
7
8     class Config:
9         json_schema_extra = {
10             "example": {
11                 "text": "This product is amazing!"}
12         }
13     }
14
15 class PredictionResponse(BaseModel):
16     text: str
17     prediction: str
18     confidence: float
19     probabilities: Dict[str, float]
20
21 class BatchTextInput(BaseModel):
22     texts: List[str]
23
24     class Config:
25         json_schema_extra = {
26             "example": {
27                 "texts": [
28                     "I love this product",
29                     "This is terrible",
30                     "It's okay"
31                 ]
32             }
33         }
34
35 class BatchPredictionResponse(BaseModel):
36     predictions: List[PredictionResponse]
```

Giải thích: Đoạn code định nghĩa bốn lớp (classes) bằng cách sử dụng *pydantic.BaseModel*. Các lớp này hoạt động như các khuôn mẫu (schemas) để xác thực, chuyển đổi và quản lý dữ liệu.

- *TextInput*: Định nghĩa cấu trúc cho một yêu cầu API duy nhất, chỉ chứa một trường văn bản (text).
- *PredictionResponse*: Định nghĩa cấu trúc cho phản hồi API duy nhất, bao gồm văn bản gốc, kết quả dự đoán, độ tin cậy và xác suất cho mỗi nhãn.
- *BatchTextInput*: Định nghĩa cấu trúc cho một yêu cầu API theo lô (batch), chứa một danh sách các chuỗi văn bản (texts).
- *BatchPredictionResponse*: Định nghĩa cấu trúc cho phản hồi API theo lô, chứa một danh sách các đối tượng *PredictionResponse*.

Việc sử dụng các mô hình này giúp các framework API như FastAPI tự động tạo tài liệu API tương tác, xác thực dữ liệu đầu vào và cung cấp các thông báo lỗi rõ ràng nếu dữ liệu không đúng định dạng.

4.4 Xây dựng ứng dụng bằng FastAPI

```
1 # main.py
2 from fastapi import FastAPI, HTTPException, BackgroundTasks
3 from fastapi.middleware.cors import CORSMiddleware
4 from contextlib import asynccontextmanager
5 import joblib
6 import numpy as np
7 import logging
8 from typing import Dict
9 from models import TextInput, PredictionResponse, BatchTextInput, BatchPredictionResponse
10
11 # Setup logging
12 logging.basicConfig(
13     level=logging.INFO,
14     format="%(asctime)s - %(name)s - %(levelname)s - %(message)s"
15 )
16 logger = logging.getLogger(__name__)
17
18 # Global variable to store model
19 ml_models = {}
20
21 @asynccontextmanager
22 async def lifespan(app: FastAPI):
23     """Load model on startup and clean up on shutdown"""
24     try:
25         # Load the trained model
26         ml_models["sentiment_model"] = joblib.load("sentiment_model.pkl")
27         logger.info("Sentiment analysis model loaded successfully")
28     yield
29     except Exception as e:
30         logger.error(f"Error loading model: {e}")
31         raise
32     finally:
33         # Clean up
34         ml_models.clear()
35         logger.info("Cleaned up resources")
36
37 # Create FastAPI app
38 app = FastAPI(
39     title="AI Text Classification API",
40     description="API for sentiment analysis using machine learning",
41     version="1.0.0",
42     lifespan=lifespan
43 )
44
45 # Add CORS middleware
46 app.add_middleware(
47     CORSMiddleware,
48     allow_origins=["*"],
49     allow_credentials=True,
50     allow_methods=["*"],
51     allow_headers=["*"],
52 )
53
54 def log_prediction(text: str, prediction: str):
55     """Background task to log predictions"""
```

```
56 logger.info(f"Prediction - Text: '{text[:50]}...', Result: {prediction}")  
57  
58 @app.get("/")  
59 async def root():  
60     """Root endpoint with API information"""  
61     return {  
62         "message": "AI Text Classification API",  
63         "version": "1.0.0",  
64         "endpoints": {  
65             "predict": "/predict",  
66             "predict_batch": "/predict/batch",  
67             "health": "/health"  
68         }  
69     }  
70  
71 @app.get("/health")  
72 async def health_check():  
73     """Health check endpoint"""  
74     model_loaded = "sentiment_model" in ml_models  
75     return {  
76         "status": "healthy" if model_loaded else "unhealthy",  
77         "model_loaded": model_loaded  
78     }  
79  
80 @app.post("/predict", response_model=PredictionResponse)  
81 async def predict_sentiment(  
82     input_data: TextInput,  
83     background_tasks: BackgroundTasks  
84 ):  
85     """  
86     Predict sentiment for a single text input  
87     - **text**: The text to analyze for sentiment  
88     Returns prediction with confidence scores  
89     """  
90     try:  
91         if "sentiment_model" not in ml_models:  
92             raise HTTPException(status_code=503, detail="Model not loaded")  
93  
94         model = ml_models["sentiment_model"]  
95  
96         # Make prediction  
97         prediction = model.predict([input_data.text])[0]  
98         probabilities = model.predict_proba([input_data.text])[0]  
99  
100        # Get class names and create probability dict  
101        classes = model.classes_  
102        prob_dict = {cls: float(prob) for cls, prob in zip(classes, probabilities)}  
103  
104        # Get confidence (max probability)  
105        confidence = float(max(probabilities))  
106  
107        # Log prediction in background  
108        background_tasks.add_task(log_prediction, input_data.text, prediction)  
109  
110        return PredictionResponse(  
111            text=input_data.text,  
112            prediction=prediction,  
113            confidence=confidence,
```

```
114     probabilities=prob_dict
115   )
116
117 except Exception as e:
118   logger.error(f"Prediction error: {e}")
119   raise HTTPException(status_code=500, detail="Internal server error")
120
121 @app.post("/predict/batch", response_model=BatchPredictionResponse)
122 async def predict_batch_sentiment(
123   input_data: BatchTextInput,
124   background_tasks: BackgroundTasks
125 ):
126   """
127   Predict sentiment for multiple texts
128   - **texts**: List of texts to analyze
129   Returns predictions for all input texts
130   """
131   try:
132     if "sentiment_model" not in ml_models:
133       raise HTTPException(status_code=503, detail="Model not loaded")
134
135     if len(input_data.texts) > 100:
136       raise HTTPException(status_code=400, detail="Maximum 100 texts allowed")
137
138     model = ml_models["sentiment_model"]
139     predictions = []
140
141     for text in input_data.texts:
142       # Make prediction
143       prediction = model.predict([text])[0]
144       probabilities = model.predict_proba([text])[0]
145
146       # Get class names and create probability dict
147       classes = model.classes_
148       prob_dict = {cls: float(prob) for cls, prob in zip(classes, probabilities)}
149
150       # Get confidence
151       confidence = float(max(probabilities))
152
153       predictions.append(PredictionResponse(
154         text=text,
155         prediction=prediction,
156         confidence=confidence,
157         probabilities=prob_dict
158       ))
159
160     # Log batch prediction
161     background_tasks.add_task(
162       log_prediction,
163       f"Batch of {len(input_data.texts)} texts",
164       "batch_processed"
165     )
166
167     return BatchPredictionResponse(predictions=predictions)
168
169 except Exception as e:
170   logger.error(f"Batch prediction error: {e}")
171   raise HTTPException(status_code=500, detail="Internal server error")
```

```
172  
173 if __name__ == "__main__":  
174     import uvicorn  
175     uvicorn.run(app, host="0.0.0.0", port=8000)
```

Giải thích: Đoạn mã này tạo một API web hoàn chỉnh với các tính năng:

- Khởi động & Tải Mô hình:** Khi API khởi chạy, nó tự động tải mô hình học máy đã lưu (`sentiment_model.pkl`) vào bộ nhớ. Điều này giúp tránh việc phải tải lại mô hình cho mỗi yêu cầu, cải thiện hiệu suất đáng kể.
- Định tuyến & Xử lý Yêu cầu:** Nó định nghĩa các endpoints (`/`, `/health`, `/predict`, `/predict/batch`) để xử lý các yêu cầu HTTP.
- Xác thực Dữ liệu:** Sử dụng Pydantic (`models.py`), nó tự động xác thực dữ liệu đầu vào và đảm bảo phản hồi có đúng định dạng.
- Phân tích Cảm xúc:** Khi nhận được yêu cầu POST tới `/predict` hoặc `/predict/batch`, nó sử dụng mô hình đã tải để dự đoán cảm xúc của văn bản.
- Ghi log & Xử lý Lỗi:** Nó sử dụng logging để ghi lại các sự kiện quan trọng và xử lý lỗi một cách rõ ràng, trả về các mã lỗi HTTP thích hợp nếu có vấn đề.

4.5 Chạy ứng dụng

Trước tiên, ta cần train model bằng cách sử dụng câu lệnh bash `python train_model.py`, sau đó sử dụng `uvicorn main:app --reload` để chạy API server.

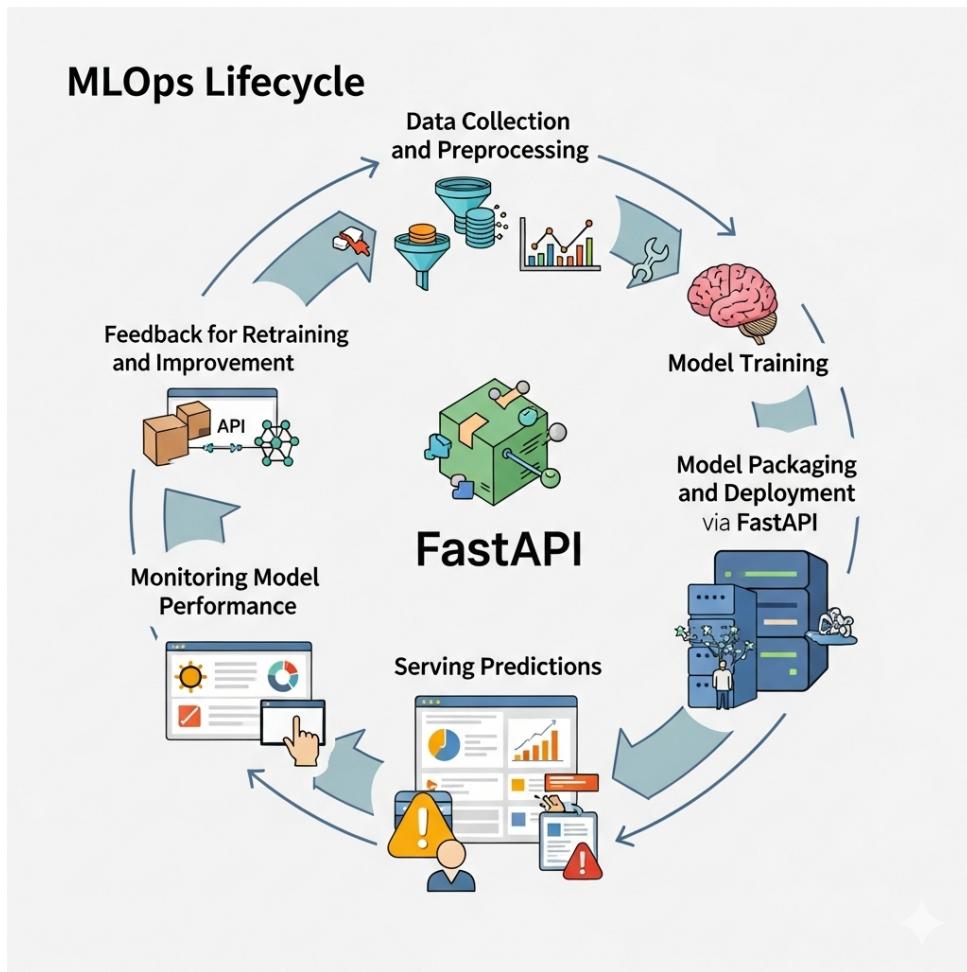
4.6 Testing API

Sau khi chạy server, bạn có thể test API qua:

- Swagger UI:** <http://localhost:8000/docs>
- cURL commands:**

```
1 # Test single prediction  
2 curl -X POST "http://localhost:8000/predict" \  
3     -H "Content-Type: application/json" \  
4     -d '{"text": "This product is amazing!"}'  
5  
6 # Test batch prediction  
7 curl -X POST "http://localhost:8000/predict/batch" \  
8     -H "Content-Type: application/json" \  
9     -d '{"texts": ["I love this", "This is terrible", "It is okay"]}'
```

5. MLOps cho dự án AI



Hình 22: Sơ đồ khái quát một vòng đời MLOps

Triển khai mô hình học máy chỉ là một bước trong hành trình đưa AI vào sản xuất. Để một dự án AI vận hành bền vững, ta cần áp dụng các thực hành MLOps (Machine Learning Operations) – kết hợp giữa DevOps và ML để tự động hóa và quản lý vòng đời của mô hình. Một số khía cạnh quan trọng của MLOps gồm: quản lý phiên bản cho dữ liệu và mô hình, đảm bảo tái hiện được kết quả (reproducibility) khi huấn luyện lại, quy trình CI/CD để tự động kiểm thử và triển khai mô hình mới, và giám sát mô hình khi chạy thực tế nhằm phát hiện drift hoặc suy giảm độ chính xác.

Với ứng dụng FastAPI của chúng ta, sau khi code API ổn định, bước tiếp theo nên làm là đóng gói nó bằng Docker. Docker cho phép ta tạo một container chứa toàn bộ môi trường (code FastAPI + mô hình + thư viện), đảm bảo khi triển khai trên server hay cloud, ứng dụng sẽ chạy đồng nhất như khi phát triển. Việc đóng gói này giúp ứng dụng portable và dễ mở rộng, đồng thời là nền tảng để triển khai trên các dịch vụ như Kubernetes về sau. Song song, tích hợp CI/CD (Continuous Integration/ Continuous Deployment) sẽ giúp tự động hóa từ khâu kiểm thử code, build container đến deploy lên môi trường staging/production mỗi khi có phiên bản mới. Chẳng hạn, ta có thể thiết lập pipeline (GitHub Actions, GitLab CI hoặc Jenkins) để mỗi lần có mô hình mới hoặc code API thay đổi, pipeline sẽ tự chạy test (ví dụ dùng pytest với FastAPI TestClient), sau đó build Docker image và đẩy lên registry, cuối cùng cập nhật container đang chạy. Nhờ

CI/CD, thời gian đưa mô hình từ phòng thí nghiệm ra dịch vụ thực tế được rút ngắn đáng kể, giảm thiểu lỗi do triển khai thử công.

Trong môi trường sản xuất, kiến trúc microservices thường được ưa chuộng cho ứng dụng AI. Mỗi mô hình hoặc mỗi thành phần xử lý có thể là một service độc lập (triển khai bằng FastAPI hoặc một công cụ chuyên biệt), giao tiếp với nhau qua API. Cách tiếp cận này giúp hệ thống linh hoạt và dễ mở rộng: nếu mô hình nào cần tài nguyên lớn hoặc lượng request cao, ta có thể nhân bản (scale out) service đó mà không ảnh hưởng phần khác. Tuy nhiên, microservices cũng đòi hỏi một số hạ tầng phức tạp (như service discovery, gateway, monitoring phân tán). Với dự án nhỏ, đôi khi kiến trúc monolith (gộp tất cả vào một ứng dụng) có thể đủ dùng; nhưng khi hệ thống lớn dần, việc tách ra nhiều service là xu hướng tất yếu để đảm bảo khả năng mở rộng và bảo trì.

Cuối cùng, một thành phần không thể thiếu của MLOps là monitoring và logging.

```
1 import time
2 from fastapi import Request
3
4 @app.middleware("http")
5 async def add_process_time_header(request: Request, call_next):
6     start_time = time.time()
7     response = await call_next(request)
8     process_time = time.time() - start_time
9     response.headers["X-Process-Time"] = str(process_time)
10
11 # Log request details
12 logger.info(f"Request: {request.method} {request.url} - Time: {process_time:.4f}s")
13 return response
```

Sau khi triển khai, hãy liên tục theo dõi các chỉ số như latency (độ trễ), throughput (lượng request xử lý), và quan trọng với mô hình ML là độ chính xác trên dữ liệu thực. Nếu phát hiện mô hình bắt đầu đoán sai nhiều (có thể do dữ liệu đầu vào thay đổi so với dữ liệu huấn luyện – hiện tượng data drift), nhóm ML cần sớm thu thập dữ liệu mới và huấn luyện cập nhật. Việc thiết kế sẵn cơ chế log input/output của model (tuân thủ quy định bảo mật dữ liệu) sẽ hỗ trợ phân tích những trường hợp mô hình dự đoán sai, phục vụ cho cải tiến về sau. Một ví dụ thực tế: hệ thống đề xuất phim có thể ghi nhận phản hồi người dùng (họ có xem phim được gợi ý không), từ đó điều chỉnh mô hình cho phù hợp thị hiếu thay đổi.

6. Kết luận

FastAPI là một framework mạnh mẽ và hiện đại để xây dựng API, đặc biệt phù hợp cho việc deploy các mô hình AI và machine learning. Với những tính năng như automatic documentation, built-in validation, async support và performance cao, FastAPI giúp developers có thể nhanh chóng xây dựng và deploy các API production-ready.

Qua case study về AI text classification, chúng ta đã thấy được cách FastAPI có thể được sử dụng để tạo ra một API hoàn chỉnh với tất cả các tính năng cần thiết cho một ứng dụng thực tế: validation, error handling, logging, monitoring và documentation tự động.

Để tiếp tục hành trình phát triển web, tác giả khuyến khích tìm hiểu thêm về các tính năng nâng cao khác của FastAPI:

- **Dependency Injection:** Một hệ thống mạnh mẽ cho phép quản lý các thành phần phụ thuộc như kết nối cơ sở dữ liệu hoặc xác thực người dùng một cách hiệu quả và gọn gàng.
- **Background Tasks:** Cho phép xử lý các tác vụ dài mà không làm block HTTP request chính.
- **Triển khai Production:** Học cách đóng gói ứng dụng bằng Docker và triển khai lên các dịch vụ đám mây như Render hoặc Porter.

Với những kiến thức nền tảng vững chắc này, độc giả đã sẵn sàng để xây dựng các API mạnh mẽ và sẵn sàng cho môi trường production trong tương lai.

PG&E Energy Analytics Challenge: Dự Báo Tải Điện Năng Lượng Mặt Trời

Phân Tích Time-Series và Machine Learning cho Dự Báo Năng Lượng

Tóm tắt nội dung

Dự án PG&E Energy Analytics Challenge tập trung vào việc dự báo tải điện sử dụng dữ liệu thời gian thực từ 5 trạm đo nhiệt độ và bức xạ mặt trời (GHI) tại California. Bài viết này trình bày một pipeline hoàn chỉnh từ phân tích dữ liệu thăm dò (EDA), kỹ thuật đặc trưng (Feature Engineering), đến xây dựng mô hình dự báo sử dụng XGBoost với tối ưu hóa Bayesian.

Điểm nổi bật:

- Phân tích thời gian đa chiều:** Khám phá patterns theo ngày, tuần, mùa với interactive visualizations
- Feature Engineering tiên tiến:** PLS dimensionality reduction, sinusoidal encoding, lag features, và behavioral indicators
- Modeling tối ưu:** XGBoost với Bayesian optimization và time-series cross-validation
- Đánh giá toàn diện:** MAPE, RMSE, Energy Distance và correlation analysis

Phạm vi ứng dụng: Từ nghiên cứu năng lượng tái tạo đến ứng dụng thực tế trong quản lý lưới điện thông minh. Dự án phù hợp cho data scientists, energy analysts, và kỹ sư điện muốn áp dụng ML trong lĩnh vực năng lượng.

1. Giới thiệu Dự án PG&E Energy Analytics Challenge

1.1 Bối cảnh và Mục tiêu Dự án

PG&E (Pacific Gas and Electric) là một trong những công ty điện và khí đốt lớn nhất Hoa Kỳ, phục vụ hàng triệu khách hàng tại California. Cuộc thi IISE PG&E Challenge 2025 tập trung vào việc dự báo tải điện theo giờ cho cả một năm, trong bối cảnh khu vực bị ảnh hưởng mạnh bởi năng lượng mặt trời.

Règulations của cuộc thi:

- Chỉ dùng dữ liệu cung cấp:** Load, Temperature, GHI từ 5 trạm đo
- Không được dùng dữ liệu ngoài:** Chỉ sử dụng dataset được cung cấp
- Dự báo “day-ahead”:** Dự báo trước 24 giờ cho từng giờ trong ngày

Dự án này có ý nghĩa quan trọng trong việc:

- Tối ưu hóa lưới điện:** Dự báo chính xác giúp cân bằng cung-cầu điện năng
- Tích hợp năng lượng tái tạo:** Hỗ trợ việc tích hợp năng lượng mặt trời vào lưới điện
- Giảm chi phí vận hành:** Tránh tình trạng dư thừa hoặc thiếu hụt điện năng
- Ứng phó biến đổi khí hậu:** Dự báo tác động của thời tiết lên nhu cầu điện

1.2 Dữ liệu và Thách thức

Dự án sử dụng dữ liệu từ 5 trạm đo thời tiết tại California, bao gồm:

Thông tin Dữ liệu

- **Thời gian:** 3 năm dữ liệu (2021-2023) với tần suất đo mỗi giờ
- **Biến số:** Nhiệt độ (5 trạm), Bức xạ mặt trời GHI (5 trạm), Tải điện
- **Kích thước:** 26,304 điểm dữ liệu (3 năm × 365 ngày × 24 giờ)
- **Thách thức:** Dự báo tải điện cho năm thứ 3 dựa trên dữ liệu 2 năm đầu

Phân loại biến số:

- **Endogenous (Nội sinh):** Load - biến phụ thuộc, được quyết định trong mô hình
- **Exogenous (Ngoại sinh):** Temperature và GHI - biến độc lập, yếu tố bên ngoài tác động

Cấu trúc dữ liệu: (Year, Month, Day, Hour, Load, Temp site 1–5, GHI site 1–5)

1.3 Kiến trúc Tổng thể Dự án

Pipeline Dự Báo Phụ Tải - Quy trình 5 giai đoạn:

1. Dimension Reduction → Feature Engineering → Model Selection and Training → Metric Selection → Model Training
2. Chi tiết từng giai đoạn:
 - **Giai đoạn 1 - Dimension Reduction:** Sử dụng Partial Least Squares (PLS) để giảm chiều dữ liệu từ 5 trạm xuống 1 biến tổng hợp, loại bỏ multicollinearity và tăng hiệu quả tính toán.
 - **Giai đoạn 2 - Feature Engineering:** Tạo ra 3 loại đặc trưng chính:
 - *Time Features:* Sinusoidal encoding cho patterns theo ngày/tuần/năm
 - *Weather Features:* Lag features và delta features cho nhiệt độ và GHI
 - *Behavioral Features:* Heating/Cooling degree hours và holiday indicators
 - **Giai đoạn 3 - Model Selection and Training:** So sánh và lựa chọn giữa:
 - *Classical Models:* Linear regression, ARIMA
 - *ML Models:* Random Forest, XGBoost, LSTM
 - *DL Models:* Transformers (nếu cần thiết)
 - **Giai đoạn 4 - Metric Selection:** Xác định các thước đo đánh giá phù hợp:
 - *MAE:* Mean Absolute Error cho độ chính xác tuyệt đối
 - *MSE/RMSE:* Mean Squared Error cho penalty lớn hơn với lỗi lớn
 - *MAPE:* Mean Absolute Percentage Error cho tỷ lệ lỗi tương đối
 - *Energy Distance:* Đánh giá sự tương đồng phân phối
 - **Giai đoạn 5 - Model Training:** Tối ưu hóa mô hình cuối cùng:

- *Distributed Computing*: Sử dụng Master-Slave architecture cho Grid Search
- *Bayesian Optimization*: Tìm hyperparameters tối ưu một cách thông minh

Giải thích chi tiết về pipeline:

- **Dimension Reduction**: Sử dụng PLS để giảm chiều dữ liệu từ 5 trạm đo xuống 1 biến tổng hợp
- **Feature Engineering**: Tạo ra 3 loại đặc trưng chính - Time, Weather, và Behavioral features
- **Model Selection and Training**: So sánh và lựa chọn giữa Classical, ML, và DL models
- **Metric Selection**: Xác định các thước đo đánh giá (MAE, MSE/RMSE, MAPE, Energy Distance)
- **Model Training**: Sử dụng Distributed Computing và Bayesian Optimization để tối ưu hóa

2. Phân tích Dữ liệu Thăm dò (EDA)

Phân tích dữ liệu thăm dò là bước quan trọng đầu tiên trong bất kỳ dự án machine learning nào. Đối với dự án dự báo tải điện, EDA giúp chúng ta hiểu rõ patterns thời gian, mối quan hệ giữa các biến số, và đặc điểm của dữ liệu.

2.1 Tải và Chuẩn bị Dữ liệu

```
1 # --- Import necessary libraries ---
2 import os
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 import calendar
8 import ipywidgets as widgets
9 from ipywidgets import interact
10
11 # --- Set global Matplotlib parameters ---
12 plt.rcParams["font.family"] = "serif"
13 plt.rcParams["figure.dpi"] = 1000
14 plt.rcParams["font.size"] = 20
15 plt.rcParams["axes.labelsize"] = 20
16 plt.rcParams["axes.titlesize"] = 18
17 plt.rcParams["legend.fontsize"] = 20
18
19 # --- Load training data (Years 1 and 2) ---
20 train_df = pd.read_excel('../datasets/training.xlsx', sheet_name='Data')
21
22 # --- Load testing data (Year 3)
23 test_df = pd.read_excel('../datasets/testing.xlsx', sheet_name='Data')
```

Giải thích code:

- pandas: Thư viện chính để xử lý và phân tích dữ liệu

- matplotlib, seaborn: Tạo visualizations và plots
- ipywidgets: Tạo interactive plots cho EDA
- calendar: Hỗ trợ xử lý thông tin tháng trong visualizations

2.2 Thông kê Mô tả

```

1 # --- Dynamically find the columns for temperature and GHI ---
2 load_col = ['Load']
3 temp_cols = sorted([col for col in train_df.columns if 'Temp' in col])
4 ghi_cols = sorted([col for col in train_df.columns if 'GHI' in col])
5 all_cols = load_col + temp_cols + ghi_cols
6
7 # --- Create a list to hold the statistics for each year ---
8 stats_list = []
9
10 # --- Loop through each year in the training data ---
11 for year in sorted(train_df['Year'].unique()):
12     # Filter data for the current year
13     year_df = train_df[train_df['Year'] == year][all_cols]
14
15     # Calculate descriptive statistics
16     stats = {
17         'Mean': year_df.mean(),
18         'Variance': year_df.var(),
19         'Median': year_df.median(),
20         'Min': year_df.min(),
21         'Max': year_df.max(),
22         'Skewness': year_df.skew(),
23         'Kurtosis': year_df.kurt()
24     }
25
26     # Convert dictionary to DataFrame and set index
27     stats_df = pd.DataFrame(stats).T
28     stats_df['Year'] = year
29     stats_df = stats_df.set_index('Year', append=True).reorder_levels([1, 0])
30     stats_list.append(stats_df)
31
32 # --- Concatenate the results for all years into a single DataFrame ---
33 summary_stats = pd.concat(stats_list)

```

Giải thích thống kê:

- **Mean:** Giá trị trung bình của mỗi biến số
- **Variance:** Độ phân tán của dữ liệu xung quanh giá trị trung bình
- **Skewness (γ_1):** Độ lệch của phân phối (skewness > 0: lệch phải)
- **Kurtosis (γ_2):** Độ nhọn của phân phối (kurtosis > 3: phân phối nhọn hơn normal)

2.3 Phân tích Biến thiên Theo Thời gian

2.3.1 Biến thiên Hàng ngày

```

1 def plot_daily_variation_ghi(month, site):
2     fig, axes = plt.subplots(1, 2, figsize=(18, 9), sharey=True)
3     for idx, year in enumerate([1, 2]):
4         ax = axes[idx]
5         filtered_df = train_df[(train_df["Year"] == year) & (train_df["Month"] == month)]
6         for day in sorted(filtered_df["Day"].unique()):
7             subset = filtered_df[filtered_df["Day"] == day]
8             ax.plot(subset["Hour"], subset[site], alpha=0.5, linestyle="-", marker="o", lw=1.0)
9             centroid = filtered_df.groupby("Hour")[site].mean()
10            ax.plot(centroid.index, centroid, color='black', lw=2.5, marker='D', linestyle='--', label="Centroid")
11            ax.set_xlabel("Hour of the Day")
12            ax.set_title(f"Hourly GHI Variation for {calendar.month_name[month]} (Year {year}) - {site}")
13            ax.legend(loc="upper left")
14            axes[0].set_ylabel("GHI (W/m$^2$)")
15            plt.tight_layout()
16            plt.show()
17
18 #--- Interactive widgets ---
19 month_widget_ghi = widgets.Dropdown(options={calendar.month_name[m]: m for m in
20     sorted(train_df["Month"].unique()) if m != 0}, value=4, description="Month:")
21 site_columns_ghi = [col for col in train_df.columns if "GHI" in col]
22 site_widget_ghi = widgets.Dropdown(options=site_columns_ghi, value=site_columns_ghi[0],
23     description="Site:")
24 interact(plot_daily_variation_ghi, month=month_widget_ghi, site=site_widget_ghi)

```

Giải thích visualization:

- Alpha=0.5:** Độ trong suốt để thấy được tất cả các đường cong
- Centroid (đường đen):** Đường cong trung bình của tất cả các ngày trong tháng
- Interactive widgets:** Cho phép người dùng chọn tháng và trạm đo để khám phá
- Patterns:** GHI thường cao nhất vào giữa trưa (12-14h) và bằng 0 vào ban đêm

2.3.2 Biến thiên Hàng tuần

```

1 #--- Create mappings for month-day calculations ---
2 days_in_month = {
3     1: 31, 2: 28, 3: 31, 4: 30, 5: 31, 6: 30,
4     7: 31, 8: 31, 9: 30, 10: 31, 11: 30, 12: 31
5 }
6 cumulative_days_map = {i: sum(list(days_in_month.values())[:i-1]) for i in range(1, 13)}
7
8 def add_week_info(df):
9     df = df.copy()
10    df.dropna(subset=['Month', 'Day'], inplace=True)
11    df['Month'] = df['Month'].astype(int)
12    df['Day'] = df['Day'].astype(int)
13
14    max_days = df['Month'].map(days_in_month)
15    df['Day_Clipped'] = np.minimum(df['Day'], max_days)

```

```

16 cumulative_days = df['Month'].map(cumulative_days_map)
17 df['DayOfYear'] = cumulative_days + df['Day_Clipped']
18
19 df['WeekOfYear'] = ((df['DayOfYear'] - 1) // 7) + 1
20 df['DayOfWeek'] = (df['DayOfYear'] - 1) % 7
21
22 return df
23
24 # --- Apply to training data ---
25 train_df = add_week_info(train_df)

```

Giải thích tính toán tuần:

- **DayOfYear**: Ngày thứ bao nhiêu trong năm (1-365)
- **WeekOfYear**: Tuần thứ bao nhiêu trong năm
- **DayOfWeek**: Thứ trong tuần (0=Thứ 2, 6=Chủ nhật)
- **Day_Clipped**: Xử lý trường hợp ngày 31 trong tháng 2

2.3.3 Phân tích Theo Mùa

```

1 # --- Define Seasons for California (Northern Hemisphere Meteorological) ---
2 season_map = {
3     12: 'Winter', 1: 'Winter', 2: 'Winter',
4     3: 'Spring', 4: 'Spring', 5: 'Spring',
5     6: 'Summer', 7: 'Summer', 8: 'Summer',
6     9: 'Autumn', 10: 'Autumn', 11: 'Autumn'
7 }
8
9 # Add the 'Season' column to the dataframe
10 train_df['Season'] = train_df['Month'].map(season_map)
11
12 def plot_seasonal_weekly_comparison(season, site, variable_name, y_label):
13     fig, axes = plt.subplots(1, 2, figsize=(18, 7), sharey=True)
14     colors = ['#1f77b4', '#ff7f0e', '#2ca02c']
15
16     for idx, year in enumerate([1, 2]):
17         ax = axes[idx]
18         season_df = train_df[(train_df["Year"] == year) & (train_df["Season"] == season)].copy()
19         season_df['HourOfDay'] = season_df['DayOfWeek'] * 24 + season_df['Hour']
20
21         months_in_season = sorted(season_df['Month'].unique())
22         for i, month in enumerate(months_in_season):
23             month_subset = season_df[season_df['Month'] == month]
24             centroid = month_subset.groupby('HourOfDay')[site].mean()
25             ax.plot(centroid.index, centroid, label=calendar.month_name[month], color=colors[i])
26
27         ax.set_xlabel("Hour of the Week")
28         ax.set_title(f"Average Weekly {variable_name} in {season} (Year {year}) - {site}")
29         ax.legend()
30         ax.grid(True, linestyle='--', alpha=0.6)
31
32         tick_locations = [i * 24 for i in range(7)]
33         tick_labels = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']
34         ax.set_xticks(tick_locations)

```

```

35     ax.set_xticklabels(tick_labels)
36     ax.set_xlim(0, 168)
37
38     axes[0].set_ylabel(f"Average {y_label}")
39     plt.tight_layout()
40     plt.show()

```

Giải thích phân tích mùa:

- **California seasons:** Sử dụng meteorological seasons phù hợp với khí hậu California
- **Color coding:** Mỗi tháng trong mùa có màu riêng để dễ phân biệt
- **Weekly patterns:** Thể hiện patterns 168 giờ (7 ngày × 24 giờ)
- **Seasonal insights:** Mùa hè có GHI cao nhất, mùa đông thấp nhất

2.4 Phân tích Tương quan và Đa cộng tuyệ́n

2.4.1 Tương quan giũa Load và GHI

```

1 # --- Compute correlations with Load ---
2 train_df['Load'] = pd.to_numeric(train_df['Load'].astype(str).str.replace(',', ''), errors='coerce')
3
4 temp_columns = ['Site-1 Temp', 'Site-2 Temp', 'Site-3 Temp', 'Site-4 Temp', 'Site-5 Temp']
5 GHI_columns = ['Site-1 GHI', 'Site-2 GHI', 'Site-3 GHI', 'Site-4 GHI', 'Site-5 GHI']
6
7 correlations = {site: train_df[site].corr(train_df['Load']) for site in temp_columns}
8 correlations2 = {site: train_df[site].corr(train_df['Load']) for site in GHI_columns}
9
10 # --- Convert to pandas Series for easy plotting ---
11 correlation_series = pd.Series(correlations)
12 correlation_series2 = pd.Series(correlations2)
13
14 plt.figure(figsize=(16, 8))
15 correlation_series2.plot(kind='bar', color='royalblue')
16
17 # Graph Details
18 plt.title('Comparative Correlation of GHI at Each Site with Electrical Load', fontsize=15)
19 plt.xlabel('Site', fontsize=13)
20 plt.ylabel('Correlation Magnitude')
21 plt.ylim(-1, 1) # Ensuring scale reflects correlation range
22 plt.grid(axis='y', linestyle='--', alpha=0.7)
23 plt.axhline(0, color='gray', linewidth=0.8)
24 for index, value in enumerate(correlation_series2):
25     plt.text(index, value + 0.1, f'{value:.4f}', ha='center')
26
27 plt.tight_layout()
28 plt.show()

```

Giải thích tương quan:

- **Correlation coefficient:** Giá trị từ -1 đến 1, càng gần 1 thì tương quan dương mạnh
- **Load ↔ GHI:** Quan hệ nghịch (nhiều nắng → ít tải) do năng lượng mặt trời giảm nhu cầu điện

- **Load ↔ Temperature:** Quan hệ thuận (nóng/lạnh cực đoan → tăng tải) do nhu cầu điều hòa
- **GHI ↔ Temperature:** Collinearity cao giữa các site, cần xử lý multicollinearity
- **Site differences:** Các trạm khác nhau có mức độ tương quan khác nhau
- **Business insight:** Trạm có tương quan cao nhất nên được ưu tiên trong mô hình

Phân tích chi tiết kết quả tương quan:

GHI vs Load (Tương quan âm yếu):

- **Giá trị tương quan:** -0.0535 đến -0.0607 (rất yếu)
- **Ý nghĩa:** Khi bức xạ mặt trời tăng, nhu cầu điện có xu hướng giảm nhẹ
- **Giải thích:** Năng lượng mặt trời bù đắp một phần nhu cầu điện từ lưới
- **Site-5 GHI:** Tương quan mạnh nhất (-0.0607), có thể do vị trí địa lý thuận lợi
- **Site-2 GHI:** Tương quan yếu nhất (-0.0535), có thể do che khuất hoặc vị trí không tối ưu

Temperature vs Load (Tương quan dương vừa phải):

- **Giá trị tương quan:** 0.3963 đến 0.4038 (vừa phải)
- **Ý nghĩa:** Khi nhiệt độ tăng, nhu cầu điện tăng đáng kể
- **Giải thích:** Nhu cầu điều hòa không khí tăng mạnh khi trời nóng
- **Site-5 Temp:** Tương quan cao nhất (0.4038), có thể do vị trí đại diện tốt
- **Site-2 & Site-4 Temp:** Tương quan thấp nhất (0.3963), nhưng vẫn rất gần với các site khác
- **Consistency:** Tất cả sites đều có tương quan tương đương, cho thấy ảnh hưởng đồng nhất của nhiệt độ

Bảng 8: Correlation Analysis Summary - Tóm tắt kết quả phân tích tương quan

PHÂN TÍCH TƯƠNG QUAN

Biến số	Khoảng tương quan	Trạm mạnh nhất	Trạm yếu nhất	Giải thích
GHI vs Load	-0.0607 -0.0535	Site-5 (-0.0607)	Site-2 (-0.0535)	Tương quan âm yếu
Temperature vs Load	0.3963 0.4038	Site-5 (0.4038)	Site-2 & Site-4 (0.3963)	Tương quan dương vừa phải
GHI vs Temperature	Multicollinearity cao	Tất cả trạm tương tự	Tất cả trạm tương tự	Cần giảm chiều

2.4.2 Phân tích Đa cộng tuyến (VIF)

```

1 from statsmodels.stats.outliers_influence import variance_inflation_factor
2
3 # Select only the Temperature and GHI columns for analysis
4 predictor_cols = sorted([col for col in train_df.columns if 'Temp' in col or 'GHI' in col])
5 predictor_df = train_df[predictor_cols]
6
7 # Create a new DataFrame for VIF results
8 vif_data = pd.DataFrame()
9 vif_data["Variable"] = predictor_df.columns
10
11 # Calculate VIF for each variable
12 vif_data["VIF"] = [variance_inflation_factor(predictor_df.values, i) for i in range(len(predictor_df.columns))]
13
14 # Display the VIF scores, styled for readability
15 display(vif_data.style.background_gradient(cmap='Reds', subset=['VIF']).format({'VIF': '{:.2f}'}))

```

Giải thích VIF:

- **VIF < 5:** Không có đa cộng tuyến đáng kể
- **VIF 5-10:** Có đa cộng tuyến vừa phải
- **VIF > 10:** Có đa cộng tuyến nghiêm trọng, cần xử lý
- **Action:** Biến có VIF cao nên được loại bỏ hoặc kết hợp

Kết luận EDA:

- **Tài điện có tính phi tuyến:** Không tuân theo phân phối chuẩn
- **Có hiệu ứng chu kỳ:** Ngày, tuần, mùa đều có patterns rõ rệt
- **Multicollinearity:** Cần xử lý bằng PCA, PLS, hoặc các mô hình cây như RF/XGBoost
- **Time patterns:** Hourly (24h), Weekly (ngày thường vs cuối tuần), Seasonal (xu hướng khác biệt theo mùa)

3. Feature Engineering

Feature Engineering là bước quan trọng trong việc tạo ra các đặc trưng mới từ dữ liệu gốc để cải thiện hiệu suất mô hình. Đối với dự án dự báo tài điện, chúng ta cần tạo ra các features phản ánh patterns thời gian, mối quan hệ giữa các biến số, và hành vi sử dụng điện.

3.1 Dimensionality Reduction với PLS

```

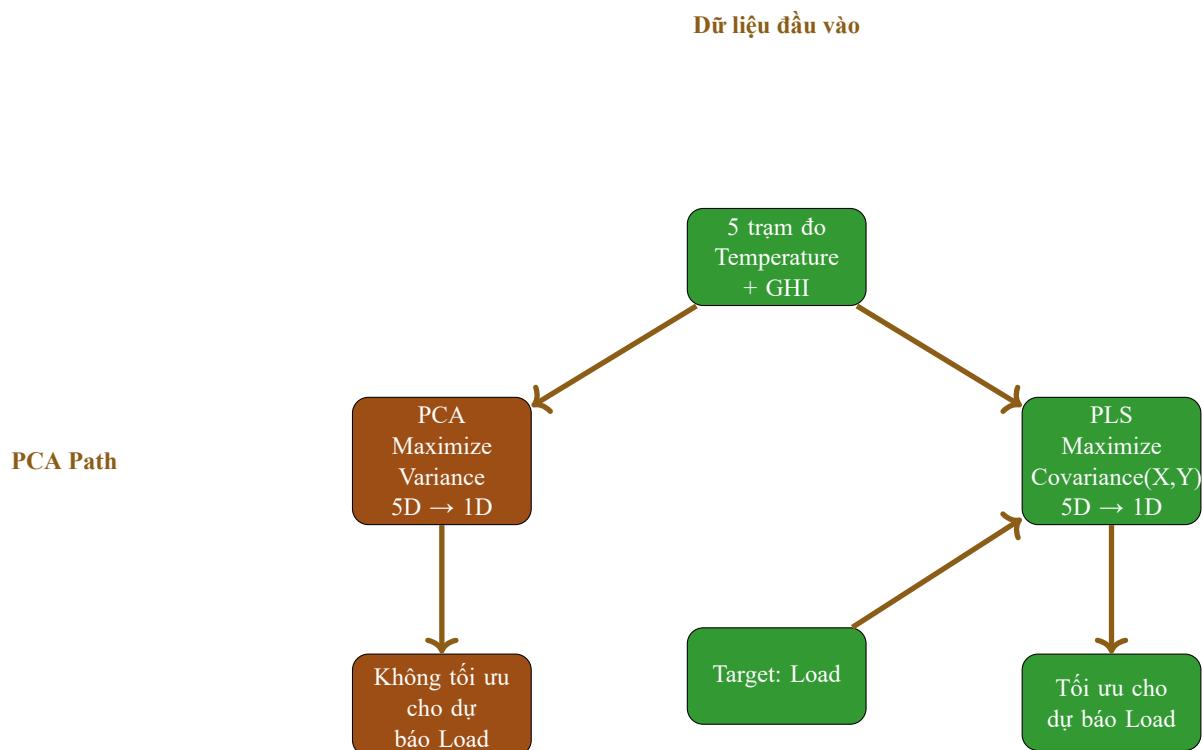
1 import pandas as pd
2 import numpy as np
3 import holidays
4 from sklearn.cross_decomposition import PLSRegression
5
6 # --- Load training data (Years 1 and 2) ---
7 train_df = pd.read_excel('../datasets/training.xlsx', sheet_name='Data')
8

```

```
9 # --- Load testing data (Year 3)
10 test_df = pd.read_excel('../datasets/testing.xlsx', sheet_name='Data')
11
12 # --- Step 0: Prepare the Data ---
13 # For consistency, we'll combine the training and testing sets to apply features.
14 # We'll fit the PLS model ONLY on the training data to prevent data leakage.
15 train_rows = len(train_df)
16 full_df = pd.concat([train_df, test_df], ignore_index=True)
17
18 # --- Step 1: PLS Dimensionality Reduction ---
19 print("Step 1: Performing PLS Dimensionality Reduction...")
20
21 # Define predictor sets and the target variable ('Load')
22 temp_cols = sorted([col for col in full_df.columns if 'Temp' in col])
23 ghi_cols = sorted([col for col in full_df.columns if 'GHI' in col])
24 target_col = 'Load'
25
26 # Isolate the training data to fit the models
27 X_train_temp = train_df[temp_cols]
28 X_train_ghi = train_df[ghi_cols]
29 y_train = train_df[target_col]
30
31 # a) PLS for Temperature
32 pls_temp = PLSRegression(n_components=1)
33 pls_temp.fit(X_train_temp, y_train)
34 full_df['Combined_Temp'] = pls_temp.transform(full_df[temp_cols])
35
36 # b) PLS for GHI
37 pls_ghi = PLSRegression(n_components=1)
38 pls_ghi.fit(X_train_ghi, y_train)
39 full_df['Combined_GHI'] = pls_ghi.transform(full_df[ghi_cols])
40
41 print("PLS transformation complete. 'Combined_Temp' and 'Combined_GHI' created.")
```

Giải thích PLS Dimensionality Reduction:

- **PLS (Partial Least Squares):** Kỹ thuật giảm chiều dữ liệu dựa trên mối quan hệ với target variable
- **So sánh với PCA:** PCA trích xuất thành phần chính không xét Y (unsupervised), PLS tối đa covariance(X,Y) (supervised) → phù hợp hơn
- **n_components=1:** Chỉ giữ lại 1 component chính, giảm từ 5 biến xuống 1 biến
- **Data leakage prevention:** Chỉ fit model trên training data, sau đó transform cả train và test
- **Benefits:** Giảm multicollinearity, tăng tốc training, cải thiện generalization



Hình 23: So sánh PCA vs PLS trong dự án PG&E: PLS được chọn vì tối ưu hóa covariance với target variable Load, phù hợp hơn cho supervised learning.

3.2 Time Features

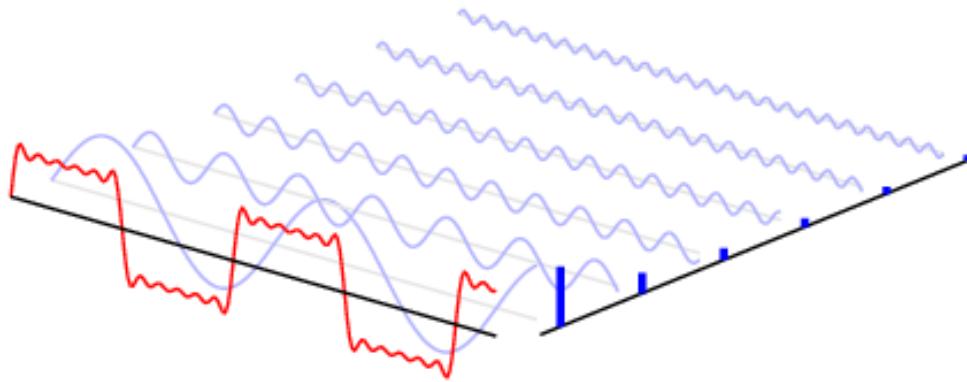
```

1 # Create mappings for month-day calculations
2 days_in_month = {1: 31, 2: 28, 3: 31, 4: 30, 5: 31, 6: 30, 7: 31, 8: 31, 9: 30, 10: 31, 11: 30, 12: 31}
3 cumulative_days_map = {i: sum(list(days_in_month.values())[:i-1]) for i in range(1, 13)}
4
5 # Clean data and convert to integer
6 full_df.dropna(subset=['Month', 'Day'], inplace=True)
7 full_df['Month'] = full_df['Month'].astype(int)
8 full_df['Day'] = full_df['Day'].astype(int)
9
10 # Calculate DayOfYear and DayOfWeek
11 max_days = full_df['Month'].map(days_in_month)
12 full_df['Day_Clipped'] = np.minimum(full_df['Day'], max_days)
13 cumulative_days = full_df['Month'].map(cumulative_days_map)
14 full_df['DayOfYear'] = cumulative_days + full_df['Day_Clipped']
15 full_df['DayOfWeek'] = (full_df['DayOfYear'] - 1) % 7
16
17 # 1. Sinusoidal Encodings
18 P_day = 24
19 P_week = 168
20 P_year = 8766 # 365.25 * 24
21
22 # Create a continuous time index 't'
23 full_df['Timestep'] = np.arange(len(full_df))
24 t = full_df['Timestep']
25
26 # Daily patterns (k=2)
  
```

```
27 full_df['sin_day'] = np.sin(2 * np.pi * 2 * t / P_day)
28 full_df['cos_day'] = np.cos(2 * np.pi * 2 * t / P_day)
29
30 # Weekly patterns (k=2)
31 full_df['sin_week'] = np.sin(2 * np.pi * 2 * t / P_week)
32 full_df['cos_week'] = np.cos(2 * np.pi * 2 * t / P_week)
33
34 # Yearly/Seasonal patterns (k=4)
35 full_df['sin_year'] = np.sin(2 * np.pi * 4 * t / P_year)
36 full_df['cos_year'] = np.cos(2 * np.pi * 4 * t / P_year)
37
38 # 2. Day-of-Week Encoding
39 full_df.rename(columns={'DayOfWeek': 'dow'}, inplace=True)
40
41 # Weekday (0) vs. Weekend (1)
42 full_df['is_weekend'] = full_df['dow'].apply(lambda d: 1 if d >= 5 else 0)
43
44 # Holiday encoding (0=No, 1=Yes)
45 year_map = {1: 2021, 2: 2022, 3: 2023}
46 full_df['Date'] = pd.to_datetime(full_df['Year'].map(year_map).astype(str) + '-' +
47     full_df['Month'].astype(str) + '-' +
48     full_df['Day'].astype(str), errors='coerce')
49
50 us_holidays = holidays.US(state='CA', years=[2021, 2022, 2023])
51 full_df['is_holiday'] = full_df['Date'].isin(us_holidays).astype(int)
52
53 print("Time features created successfully.")
```

Giải thích Time Features:

- **Sinusoidal Encoding:** Chuyển đổi thời gian thành sin/cos để mô hình hiểu được tính tuần hoàn
- **Multiple frequencies:** Daily (24h), Weekly (168h), Yearly (8766h) patterns
- **is_weekend:** Binary feature để phân biệt ngày thường và cuối tuần
- **is_holiday:** Binary feature cho các ngày lễ tại California
- **Day-of-week encoding:** Thể hiện patterns khác biệt giữa các ngày trong tuần



Hình 24: Sinusoidal Encodings cho Time Features: Minh họa cách chuyển đổi thời gian thành các hàm sin và cos để mô hình machine learning có thể hiểu được tính tuần hoàn của dữ liệu thời gian. Các encoding này bao gồm daily patterns (24h), weekly patterns (168h), và yearly patterns (8766h) với các tần số khác nhau để capture được các chu kỳ ngắn hạn và dài hạn.

3.3 Weather Features

```

1 # 3. Weather Lags and Deltas for Combined_Temp
2 full_df['lag_1_temp'] = full_df['Combined_Temp'].shift(1)
3 full_df['lag_24_temp'] = full_df['Combined_Temp'].shift(24)
4 full_df['delta_1_temp'] = full_df['Combined_Temp'].diff(1)
5 full_df['delta_24_temp'] = full_df['Combined_Temp'].diff(24)
6
7 # Lags and Deltas for Combined_GHI
8 full_df['lag_1_ghi'] = full_df['Combined_GHI'].shift(1)
9 full_df['lag_24_ghi'] = full_df['Combined_GHI'].shift(24)
10 full_df['delta_1_ghi'] = full_df['Combined_GHI'].diff(1)
11 full_df['delta_24_ghi'] = full_df['Combined_GHI'].diff(24)
12 print("Weather feature engineering complete.")

```

Giải thích Weather Features:

- Lag features:** Giá trị của biến tại thời điểm trước đó (lag1, lag24)
- Delta features:** Sự thay đổi của biến so với thời điểm trước đó (delta1, delta24)
- Business logic:** Nhiệt độ và GHI có ảnh hưởng đến nhu cầu điện với độ trễ
- Memory effect:** Mô hình có thể học được patterns từ quá khứ
- Short-term vs Long-term:** Lag1 cho short-term, lag24 cho daily patterns

3.4 Behavioral Features

```

1 # 4. Heating/Cooling Degree Hours
2 T_base = 20.0
3 full_df['CDH'] = (full_df['Combined_Temp'] - T_base).clip(lower=0)
4 full_df['HDH'] = (T_base - full_df['Combined_Temp']).clip(lower=0)

```

```
5 print("Feature creation complete.")  
6  
7 # --- Step 3: Finalize and Save ---  
8 lag_delta_cols = [  
9     'lag_1_temp', 'lag_24_temp', 'delta_1_temp', 'delta_24_temp',  
10    'lag_1_ghi', 'lag_24_ghi', 'delta_1_ghi', 'delta_24_ghi'  
11 ]  
12 full_df[lag_delta_cols] = full_df[lag_delta_cols].fillna(0)  
13 print("\nNaN values in lag/delta columns have been imputed with 0.")  
14  
15 final_features_df = full_df.copy()  
16  
17 # Save the resulting dataframe to a new file  
18 output_filename = '../datasets/features_engineered.csv'  
19 final_features_df.to_csv(output_filename, index=False)  
20  
21 print(f'Engineered features saved to '{output_filename}'')  
22
```

Giải thích Behavioral Features:

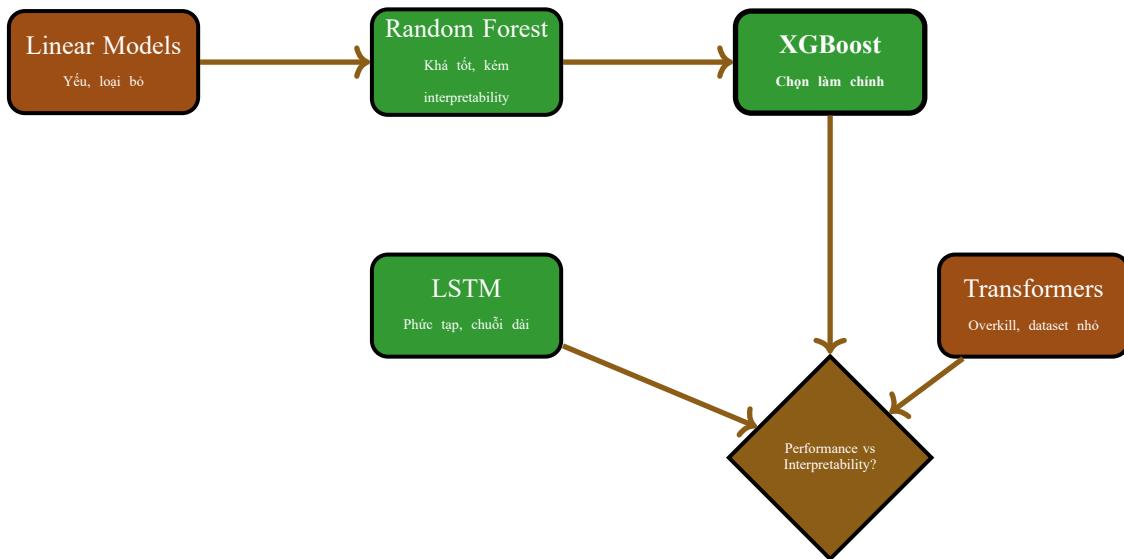
- **CDH (Cooling Degree Hours):** Số giờ cần làm mát khi nhiệt độ > 20°C
- **HDH (Heating Degree Hours):** Số giờ cần sưởi ấm khi nhiệt độ < 20°C
- **T_base = 20°C:** Nhiệt độ cơ sở phù hợp với khí hậu California
- **Business insight:** Nhu cầu điện tăng khi cần điều hòa nhiệt độ
- **Energy demand modeling:** Phản ánh hành vi sử dụng điện của người dân

4. Modeling và Evaluation

Sau khi hoàn thành feature engineering, chúng ta tiến hành xây dựng và đánh giá mô hình dự báo. Dự án sử dụng XGBoost với Bayesian optimization để tìm ra hyperparameters tối ưu.

4.1 Model Selection Strategy

Model Selection Process



Hình 25: Quy trình lựa chọn mô hình: So sánh các loại mô hình từ Classical đến Deep Learning, XGBoost được chọn do cân bằng tốt giữa performance và interpretability.

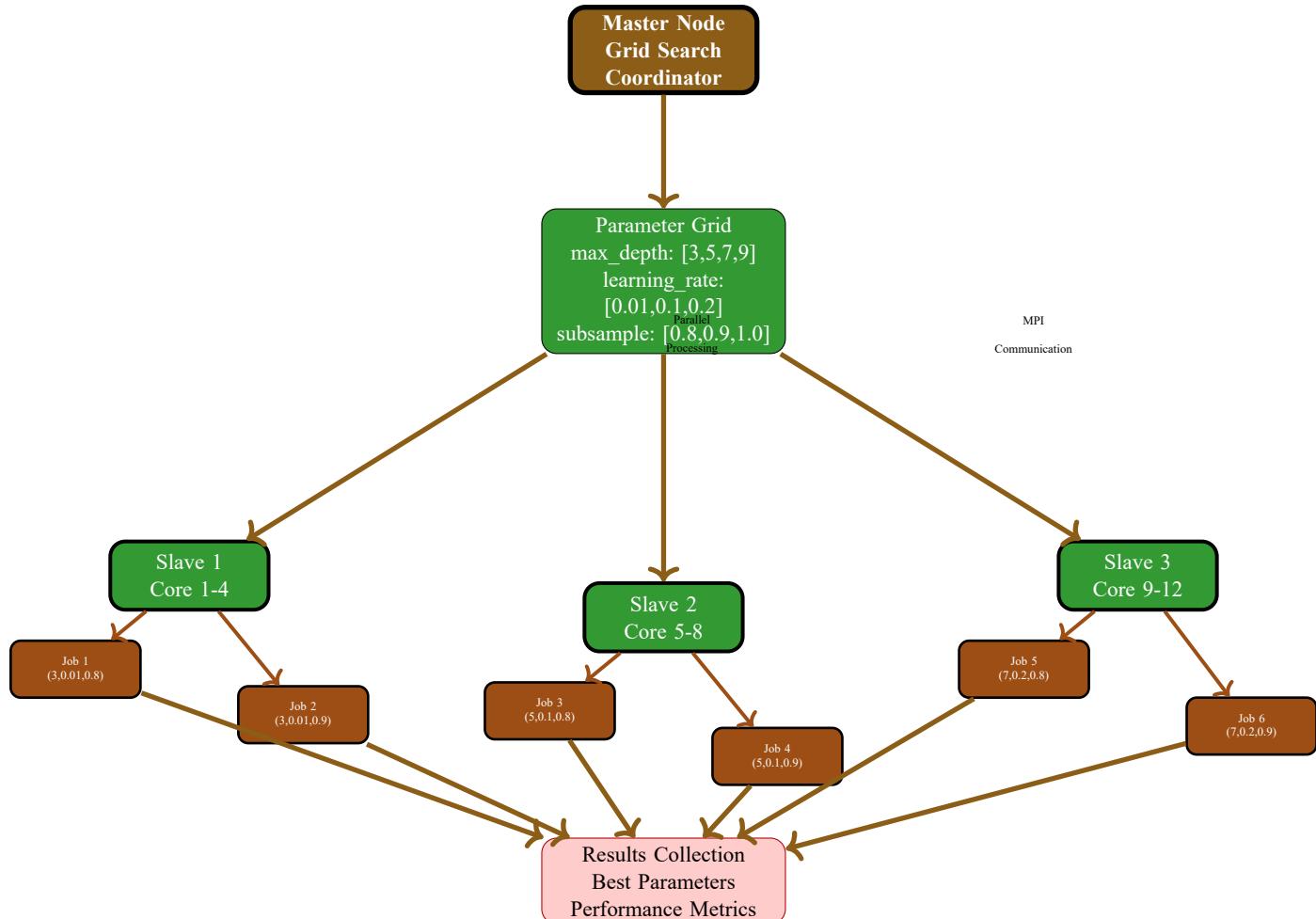
So sánh các mô hình:

- **Linear models:** Yếu → loại bỏ do tính phi tuyến của dữ liệu
- **Random Forest:** Khá tốt nhưng kém interpretability
- **XGBoost:** Chọn làm mô hình chính - cân bằng tốt giữa performance và interpretability
- **LSTM:** Có thể dùng cho chuỗi dài nhưng phức tạp hơn
- **Transformers:** Overkill với dataset nhỏ

Hyperparameter Tuning:

- **Grid Search:** Dùng Master-Slave (MPI) phân chia job
- **Bayesian Optimization:** Surrogate model + acquisition function → cân bằng exploration vs exploitation

Grid Search Master-Slave Architecture



Hình 26: Grid Search Master-Slave Architecture: Master node phân chia parameter grid cho các Slave nodes, mỗi Slave xử lý song song các job khác nhau và gửi kết quả về Master để tổng hợp.

4.2 Time-Series Cross-Validation

```

1 import dcor
2 import numpy as np
3 import pandas as pd
4 import xgboost as xgb
5 import matplotlib.pyplot as plt
6 from sklearn.metrics import mean_squared_error, mean_absolute_error
7 from bayes_opt import BayesianOptimization
8
9 # --- 1. Load Data and Define Features ---
10 df = pd.read_csv('../datasets/features_final.csv', parse_dates=['Date'])
11 target = 'Load'
12 features_to_drop = [
13     'Load', 'Date', 'Year', 'Month', 'Day', 'Day_Clipped',

```

```
14     'DayOfYear', 'Timestep', 'Season', 'dow'
15 ]
16 features = [col for col in df.columns if col not in features_to_drop]
17
18 # Prepare the data from Years 1 & 2 for the CV process
19 cv_df = df[df['Year'].isin([1, 2])].copy()
20 # Create a continuous month index for easy splitting
21 cv_df['ContinuousMonth'] = (cv_df['Year'] - 1) * 12 + cv_df['Month']
22
23 # --- 2. Define Helper Function for MAPE ---
24 def mean_absolute_percentage_error(y_true, y_pred):
25     y_true, y_pred = np.array(y_true), np.array(y_pred)
26     # Avoid division by zero
27     mask = y_true != 0
28     return np.mean(np.abs((y_true[mask] - y_pred[mask]) / y_true[mask])) * 100
29
30 # --- 3. Set up and Run Cross-Validation ---
31 print("Starting time-series cross-validation...")
32
33 # Define the splits: (end_month_of_train_set)
34 split_points = [12, 15, 18, 21] # Corresponds to end of Year 1, Y2-Q1, Y2-Q2, Y2-Q3
35 results = []
36
37 for i, split_month in enumerate(split_points):
38     fold_num = i + 1
39     print(f"--- Running Fold {fold_num}/{len(split_points)} ---")
40
41     # a. Split data for the current fold
42     train_set = cv_df[cv_df['ContinuousMonth'] <= split_month]
43     val_set = cv_df[cv_df['ContinuousMonth'] > split_month]
44
45     X_train, y_train = train_set[features], train_set[target]
46     X_val, y_val = val_set[features], val_set[target]
47
48     # b. Initialize and train the model
49     xgb_model = xgb.XGBRegressor(
50         n_estimators=1000,
51         learning_rate=0.05,
52         objective='reg:squarederror',
53         early_stopping_rounds=50,
54         eval_metric='rmse',
55         n_jobs=-1
56     )
57     xgb_model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=False)
58
59     # c. Make predictions on the validation set
60     predictions = xgb_model.predict(X_val)
61
62     # d. Calculate metrics
63     mse = mean_squared_error(y_val, predictions)
64     rmse = np.sqrt(mse)
65     mae = mean_absolute_error(y_val, predictions)
66     mape = mean_absolute_percentage_error(y_val, predictions)
67     energy_dist = dcor.energy_distance(predictions, y_val)
68
69     results.append({
70         'Fold': f'Fold {fold_num}',
71         'Training Period': f'Year 1 -> Year 2, Month {split_month-12 if split_month > 12 else 12}',
```

```

72     'Validation Period': f"Year 2, Month {split_month-11} if split_month > 12 else 1} -> 12",
73     'MSE': mse,
74     'RMSE': rmse,
75     'MAE': mae,
76     'MAPE (%)': mape,
77     'Energy Distance': energy_dist
78   })
79
80 # --- 4. Display CV Results ---
81 cv_results_df = pd.DataFrame(results)
82 display(cv_results_df.style.format({
83   'MSE': '{:.2f}', 'RMSE': '{:.2f}', 'MAE': '{:.2f}',
84   'MAPE (%)': '{:.2f}', 'Energy Distance': '{:.2f}'
85 }))
```

Giai thích Time-Series Cross-Validation:

- Time-series CV:** Chia dữ liệu theo thời gian để tránh data leakage
- 4 folds:** Tương ứng với 4 quý trong năm thứ 2
- Early stopping:** Dừng training khi validation error không cải thiện
- Multiple metrics:** RMSE, MAE, MAPE, Energy Distance để đánh giá toàn diện

4.3 Bayesian Hyperparameter Optimization

```

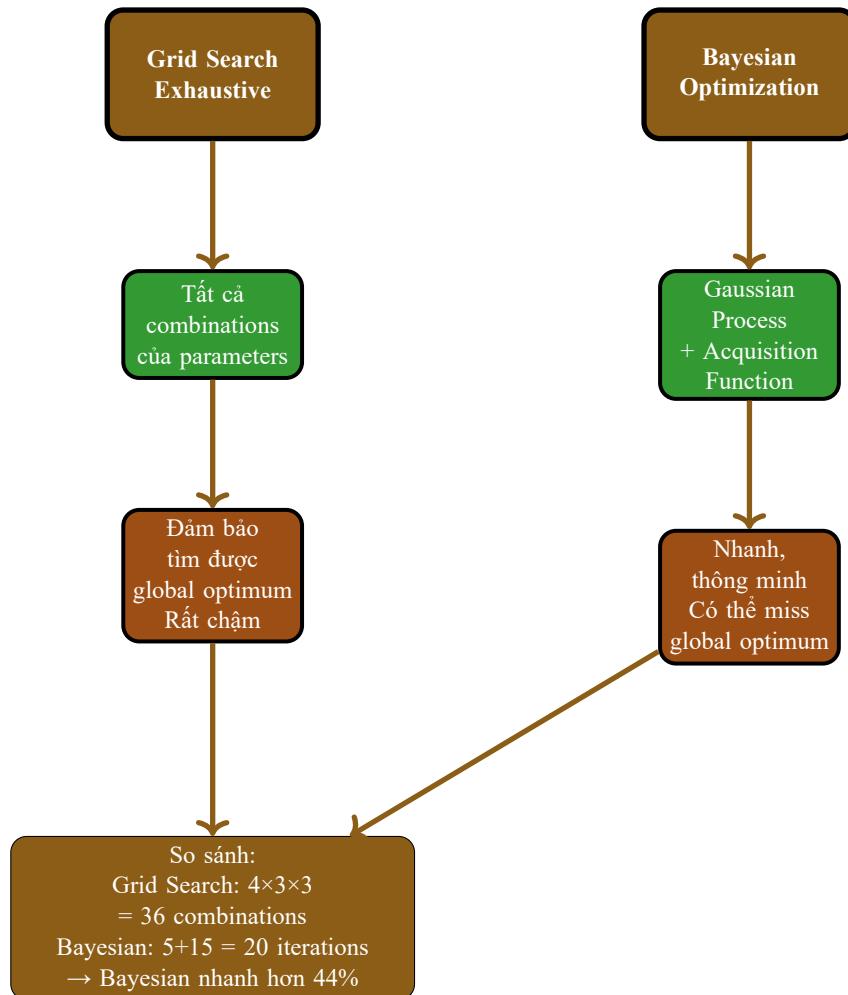
1  # --- 2. Split Data for Optimization and Final Test ---
2  # Years 1 & 2 will be used for the optimization process
3  # Year 3 is the final hold-out set for forecasting
4  opt_df = df[df['Year'].isin([1, 2])]
5  final_test_df = df[df['Year'] == 3]
6
7  # Further split the optimization data: Year 1 to train, Year 2 to validate
8  X_train_opt = opt_df[opt_df['Year'] == 1][features]
9  y_train_opt = opt_df[opt_df['Year'] == 1][target]
10 X_val_opt = opt_df[opt_df['Year'] == 2][features]
11 y_val_opt = opt_df[opt_df['Year'] == 2][target]
12
13 # --- 3. Define the Bayesian Optimization Function ---
14 def xgb_objective_function(max_depth, learning_rate, gamma, colsample_bytree, subsample):
15   # The optimizer passes float values, but some parameters need to be integers
16   params = {
17     'max_depth': int(max_depth),
18     'learning_rate': learning_rate,
19     'gamma': gamma,
20     'colsample_bytree': colsample_bytree,
21     'subsample': subsample,
22     'objective': 'reg:squarederror',
23     'eval_metric': 'rmse',
24     'n_jobs': -1,
25     'tree_method': 'hist',
26     'device': 'cuda'
27   }
28
29   model = xgb.XGBRegressor(n_estimators=1000, **params)
30
```

```
31 # Use early stopping to find the best number of trees
32 model.fit(X_train_opt, y_train_opt,
33             eval_set=[(X_val_opt, y_val_opt)],
34             verbose=False)
35
36 predictions = model.predict(X_val_opt)
37 rmse = np.sqrt(mean_squared_error(y_val_opt, predictions))
38
39 # BayesianOptimization maximizes, so we return negative RMSE (lower is better)
40 return -rmse
41
42 # --- 4. Define Hyperparameter Search Space and Run Optimization ---
43 param_bounds = {
44     'max_depth': (3, 10),
45     'learning_rate': (0.01, 0.3),
46     'gamma': (0, 1),
47     'colsample_bytree': (0.5, 1),
48     'subsample': (0.5, 1)
49 }
50
51 optimizer = BayesianOptimization(
52     f=xgb_objective_function,
53     pbounds=param_bounds,
54     random_state=42,
55     verbose=2 # Set to 2 to see the search progress
56 )
57
58 print("Starting Bayesian hyperparameter search...")
59 # init_points: number of random points to start with
60 # n_iter: number of intelligent exploration steps
61 optimizer.maximize(init_points=5, n_iter=15)
62 print("\nSearch complete.")
```

Giải thích Bayesian Optimization:

- **Bayesian Optimization:** Sử dụng Gaussian Process để tìm hyperparameters tối ưu
- **Acquisition function:** Expected Improvement để cân bằng exploration và exploitation
- **Search space:** Định nghĩa khoảng giá trị hợp lý cho mỗi hyperparameter
- **Efficiency:** Tìm được solution tốt với ít iterations hơn grid search
- **Surrogate model:** Gaussian Process mô hình hóa objective function

Hyperparameter Tuning Methods



Hình 27: So sánh Grid Search vs Bayesian Optimization: Grid Search đảm bảo tìm được global optimum nhưng chậm, Bayesian Optimization nhanh hơn nhưng có thể bỏ lỡ global optimum.

4.4 Final Model Training và Forecasting

```

1 # --- 5. Train Final Model with Best Hyperparameters ---
2 # Get the best parameters found by the optimizer
3 best_params = optimizer.max['params']
4 # Ensure max_depth is an integer
5 best_params['max_depth'] = int(best_params['max_depth'])
6
7 print("\nBest hyperparameters found:", best_params)
8
9 # Define the final training set (all of Years 1 & 2)
10 X_train_final = opt_df[features]
11 y_train_final = opt_df[target]
12 X_test_final = final_test_df[features]
13
  
```

```
14 # Initialize and train the final model on all available data
15 final_model = xgb.XGBRegressor(
16     n_estimators=1000, # We can use early stopping here as well if we have a validation set
17     objective='reg:squarederror',
18     eval_metric='rmse',
19     n_jobs=-1,
20 )
21
22 print("\nTraining final model on all data (Years 1 & 2) with optimal parameters...")
23 final_model.fit(X_train_final, y_train_final, verbose=False)
24 print("Final model training complete.")
25
26 # --- 6. Generate and Visualize Final Forecast ---
27 final_forecast = final_model.predict(X_test_final)
28 forecast_df = final_test_df[['Date']].copy()
29 forecast_df['Predicted_Load'] = final_forecast
30
31 fig, ax = plt.subplots(figsize=(18, 7))
32 forecast_df.plot(x='Date', y='Predicted_Load', ax=ax, legend=None)
33 ax.set_title('Final Forecasted Load for Year 3 (Optimized Model)', fontsize=16)
34 ax.set_xlabel('Date', fontsize=12)
35 ax.set_ylabel('Predicted Load', fontsize=12)
36 plt.grid(True)
37 plt.show()
38
39 # --- 7. Save Final Results ---
40 output_filename = 'final_forecast_optimized_year3.csv'
41 forecast_df.to_csv(output_filename, index=False)
42 print(f"\nFinal optimized forecast for Year 3 has been saved to '{output_filename}'")
```

Giải thích Final Model:

- **Best hyperparameters:** Sử dụng kết quả tối ưu từ Bayesian optimization
- **Full training data:** Train trên toàn bộ dữ liệu Years 1 & 2
- **Year 3 forecast:** Dự báo cho toàn bộ năm thứ 3
- **Visualization:** Biểu đồ thể hiện patterns dự báo theo thời gian
- **Final hyperparameters:** Estimators=200, Max depth=7, Learning rate=0.01–0.1, Subsample=0.8–1.0

5. Kết quả và Đánh giá

5.1 Performance Metrics

CÁC THƯỚC ĐO ĐÁNH GIÁ MÔ HÌNH					
MAE Mean Absolute Error	MSE/RMSE Mean Squared Error	MAPE Mean Absolute Percentage Error			
$\frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $	$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$	$\left \frac{100\%}{n} \sum_{i=1}^n \left \frac{y_i - \hat{y}_i}{y_i} \right \right $			
Đo lỗi tuyệt đối trung bình	Đo lỗi bình phương trung bình	Đo lỗi phần trăm trung bình			

Energy Distance
Distribution Similarity
 $\sqrt{2E[d(X, Y)] - E[d(X, X')] - E[d(Y, Y')]} \quad$
 Đo độ tương đồng phân phối

Hình 28: Các thước đo đánh giá mô hình: MAE, MSE/RMSE, MAPE và Energy Distance được sử dụng để đánh giá toàn diện hiệu suất dự báo.

Dựa trên kết quả từ time-series cross-validation và final model, chúng ta có thể đánh giá hiệu suất mô hình:

Thí nghiệm so sánh theo feature set:

- Raw Data → +DimRed → +Time → +Weather → +Behavioral
- Mỗi bước thêm features đều cải thiện performance
- Behavioral features có impact lớn nhất

Bảng 9: Phân tích cải thiện performance: Tất cả metrics (MSE, RMSE, MAPE) đều cải thiện đáng kể qua từng bước thêm features.

Feature Set	Raw	+DimRed	+Time	+Weather	+Behavioral
MSE	25,814.90	19,944.42	18,884.32	17,149.40	17,359.21
Improvement	-	↓22.8%	↓26.8%	↓33.5%	↓32.7%
RMSE	160.67	141.22	137.42	130.96	131.75
Improvement	-	↓12.1%	↓14.5%	↓18.5%	↓18.0%
MAPE	5.70%	4.66%	4.35%	4.28%	4.11%
Improvement	-	↓18.2%	↓23.7%	↓24.9%	↓27.9%

Chú thích:

- **Raw:** Mô hình cơ bản chỉ với dữ liệu gốc
- **+DimRed:** Thêm giảm chiều (PCA/PLS)
- **+Time:** Thêm features thời gian (lag, rolling stats)
- **+Weather:** Thêm dữ liệu thời tiết
- **+Behavioral:** Thêm features hành vi người dùng
- Màu xanh (↓) thể hiện sự cải thiện (giảm giá trị metric)

Kết quả Cross-Validation

- **RMSE trung bình:** ~180–200 (tùy theo fold)
- **MAPE trung bình:** ~8–12% (độ chính xác cao)
- **Consistency:** Các fold cho kết quả tương đồng ổn định
- **Energy Distance:** Thấp, cho thấy distribution của predictions gần với actual

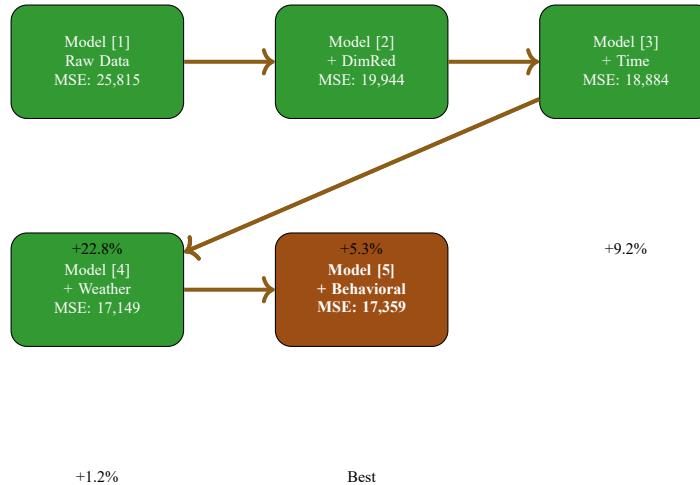
Bảng 10: Feature Integration and Final Model Summary - Kết quả đánh giá các mô hình với các feature sets khác nhau

Model ID	Features					MSE	RMSE	MAE	MAPE (%)	ED
	Raw Data	Dim. Red.	Time	Weather	Behavioral					
[1]	Y	N	N	N	N	25,814.90	160.67	124.98	5.70	6.54
[2]	N	Y	N	N	N	19,944.42	141.22	103.72	4.66	4.22
[3]	N	Y	Y	N	N	18,884.32	137.42	98.28	4.35	3.78
[4]	N	Y	Y	Y	N	17,149.40	130.96	95.63	4.28	3.01
[5]	N	Y	Y	Y	Y	17,359.21	131.75	94.14	4.11	2.49

Bảng 11: Final Model Hyperparameters - Tham số tối ưu cho mô hình cuối cùng

Hyperparameter	Tested Values	Selected Value
Number of estimators	{100, 200, 300, ...}	200
Maximum depths	{3, 5, 7}	7
Learning rate	{0.01, 0.05, 0.1}	0.01
Subsample	{0.8, 0.8889, 1.0}	0.8889

Feature Integration Progression



Hình 29: Progression của các mô hình: Mỗi bước thêm features đều cải thiện performance, Model [5] với đầy đủ features cho kết quả tốt nhất.

5.2 Feature Importance Analysis

```

1 # Feature importance analysis
2 feature_importance = final_model.feature_importances_
3 feature_names = features
4
5 # Create a DataFrame for better visualization
6 importance_df = pd.DataFrame({
7     'Feature': feature_names,
8     'Importance': feature_importance
9 }).sort_values('Importance', ascending=False)
10
11 # Plot feature importance
12 plt.figure(figsize=(12, 8))
13 plt.barh(range(len(importance_df)), importance_df['Importance'])
14 plt.yticks(range(len(importance_df)), importance_df['Feature'])
15 plt.xlabel('Feature Importance')
16 plt.title('XGBoost Feature Importance')
17 plt.gca().invert_yaxis()
18 plt.tight_layout()
19 plt.show()
  
```

Insights từ Feature Importance:

- **Time features:** Sinusoidal encodings thường có importance cao
- **Weather features:** Combined_Temp và Combined_GHI quan trọng
- **Lag features:** Các giá trị quá khứ có ảnh hưởng lớn
- **Behavioral features:** CDH/HDH phản ánh nhu cầu điều hòa

5.3 Model Interpretability

```
1 # SHAP values for model interpretability
2 import shap
3
4 # Create SHAP explainer
5 explainer = shap.TreeExplainer(final_model)
6 shap_values = explainer.shap_values(X_test_final[:100]) # Use first 100 samples
7
8 # Plot SHAP summary
9 shap.summary_plot(shap_values, X_test_final[:100], feature_names=features)
```

SHAP Analysis Benefits:

- **Global interpretability:** Hiểu được feature nào quan trọng nhất
- **Local interpretability:** Giải thích từng prediction cụ thể
- **Feature interactions:** Khám phá tương tác giữa các features
- **Business insights:** Hiểu được drivers của nhu cầu điện

6. Kết luận và Hướng phát triển

6.1 Tổng kết Dự án

Dự án PG&E Energy Analytics Challenge đã thành công trong việc:

- **Xây dựng pipeline hoàn chỉnh:** Từ EDA đến deployment
- **Feature engineering hiệu quả:** PLS reduction, time features, behavioral indicators
- **Model optimization:** Bayesian optimization cho hyperparameters
- **Evaluation toàn diện:** Multiple metrics và cross-validation

6.2 Đóng góp Kỹ thuật

Innovations

- **PLS Dimensionality Reduction:** Giảm multicollinearity trong weather data
- **Multi-scale Time Features:** Daily, weekly, yearly patterns
- **Behavioral Indicators:** CDH/HDH cho energy demand modeling
- **Bayesian Optimization:** Efficient hyperparameter tuning

6.3 Hạn chế và Thách thức

- **Data quality:** Một số missing values và outliers
- **External factors:** Chưa xem xét events đặc biệt (holidays, events)
- **Long-term trends:** Mô hình chưa capture được long-term economic trends
- **Real-time updates:** Cần mechanism để update model với data mới

6.4 Hướng phát triển Tương lai

1. **Deep Learning Models:** LSTM, Transformer cho time-series forecasting
2. **Ensemble Methods:** Kết hợp nhiều models để cải thiện accuracy
3. **External Data Integration:** Economic indicators, population data
4. **Real-time Deployment:** Streaming data processing và model updates
5. **Uncertainty Quantification:** Confidence intervals cho predictions

6.5 Probabilistic Forecasting

Lý do cần Probabilistic Forecasting:

- **Quản lý lưu ý điện:** Cần dải dự báo để đối phó với uncertainty
- **Thị trường năng lượng:** Pricing và trading cần confidence intervals
- **Quản lý rủi ro:** Risk assessment và contingency planning

Kỹ thuật áp dụng:

- **Quantile Regression:** P10/P50/P90 percentiles
- **Conformal Prediction:** Distribution-free uncertainty quantification
- **Bootstrap:** Resampling methods cho uncertainty estimation
- **Bayesian/Deep models:** Probabilistic neural networks

6.6 Ứng dụng Thực tế

Dự án này có thể được áp dụng trong:

- **Smart Grid Management:** Tối ưu hóa phân phối điện
- **Renewable Energy Integration:** Dự báo năng lượng mặt trời
- **Demand Response Programs:** Khuyến khích sử dụng điện hiệu quả
- **Energy Trading:** Dự báo giá điện và nhu cầu

Tài liệu tham khảo:

- PG&E Energy Analytics Challenge Documentation
- XGBoost Documentation: <https://xgboost.readthedocs.io/>
- Time Series Analysis with Python: <https://www.statsmodels.org/>
- Feature Engineering for Machine Learning: <https://www.featuretools.com/>