

Blog Tuần 3 – Module 4

XGBoost, Tree Review, LightGBM, Docker và Sales Forecasting

Từ XGBoost cơ bản đến LightGBM nâng cao, containerization và dự án dự báo doanh số

Tác giả: Team GRID034

Tuần thứ ba của Module 4 tập trung vào các **thuật toán boosting tiên tiến, containerization và dự án dự báo doanh số**. Module này giúp người học nắm vững XGBoost, LightGBM, Docker và áp dụng vào dự án thực tế.

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

1. **XGBoost – Học qua ví dụ tinh tay** Hiểu XGBoost từ cơ bản thông qua các ví dụ tính toán chi tiết và minh họa trực quan.
2. **Tree Review – Tổng quan về Decision Trees** Các khái niệm cơ bản về decision trees, random forest và ensemble methods.
3. **LightGBM & SHAP – Gradient Boosting hiện đại** Thuật toán LightGBM, SHAP explainability và so sánh với các thuật toán boosting khác.
4. **Dockerize Applications – Containerization cho ML** Triển khai ứng dụng ML với Docker, containerization và deployment.
5. **Project Sales Forecasting – Dự báo doanh số** Dự án thực tế dự báo doanh số sử dụng time-series analysis và machine learning.

Giá trị nhận được sau khi đọc Blog

- Hiểu và triển khai được XGBoost từ cơ bản đến nâng cao.
- Nắm vững các khái niệm về decision trees và ensemble methods.
- Biết cách sử dụng LightGBM và SHAP cho model explainability.
- Triển khai ứng dụng ML với Docker và containerization.

- Áp dụng được time-series analysis cho dự báo doanh số.

Mục lục

XGBoost – Học qua ví dụ tính tay	7
• Ôn tập về Gradient Boosting	5
• Giới thiệu về XGBoost	7
• Cách xây dựng XGBoost	10
• Các ví dụ về XGBoost	13
• Kết luận	15
Tree Review – Tổng quan về Decision Trees	17
• Lời Dẫn Nhập	17
• Decision Tree - Nền Móng Của Trực Giác	19
• Random Forest - Sức Mạnh Của "Hội Đồng Cây"	20
• AdaBoost - Học Từ Sai Lầm	23
• Gradient Boosting - Tổng Quát Hóa Ý Tưởng Sửa Lỗi	24
• XGBoost - Đỉnh Cao Của Tối Ưu Hóa	25
• Kết Luận	27
LightGBM & SHAP – Gradient Boosting hiện đại	28
• Giới thiệu	28
• Ôn tập nhanh: Các mô hình trước LightGBM	29
• Cách LightGBM xây dựng cây	30
• Dữ liệu sử dụng	42
• Pipeline Phân loại: HEPMASS	43
• Pipeline Hồi quy: NYC Taxi	44
• Case Study: So sánh hiệu suất	45
• Giải thích mô hình bằng SHAP	47
• Kết luận	49
Dockerize Applications – Containerization cho ML	52
• Giới thiệu	52
• Đối tượng & mục tiêu	52
• Kiến trúc tổng quan	52
• Chuẩn bị theo hệ điều hành	52
• Tiêu chuẩn hoá bô cục dự án	53

• Ví dụ: FastAPI + PostgreSQL	53
• Ví dụ 2: Next.js + Redis	55
• CI/CD nâng cao	56
• Triển khai thử nghiệm	57
• Best practices nâng cao	57
• Inclusion	58
Project Sales Forecasting – Dự báo doanh số	59
• Giới thiệu	59
• Các công nghệ cốt lõi sử dụng	60
• Quy trình triển khai	61
• Phân tích Khả năng Giải thích (XAI)	63
• Kết quả và Thảo luận	64
• Hạn chế và Bài học Rút ra	65
• Các Hướng Phát triển Tương lai	65
• Kết luận	67

XGBoost: Từ lý thuyết Gradient Boosting đến một thuật toán thực chiến hàng đầu

Trịnh Nguyễn Huy Hoàng

Tóm tắt nội dung

Trong bối cảnh khoa học dữ liệu hiện đại, việc xây dựng các mô hình dự đoán hiệu quả là một nhiệm vụ cốt yếu. Một trong những phương pháp mạnh mẽ nhất để đạt được điều này là học tập tổ hợp (ensemble learning), một kỹ thuật kết hợp nhiều mô hình học máy cơ bản (còn gọi là các mô hình "yếu" - weak learners) để tạo ra một mô hình tổng hợp duy nhất với hiệu suất vượt trội. Trong các kỹ thuật học tập tổ hợp, phương pháp Boosting nổi lên như một chiến lược đặc biệt hiệu quả, trong đó các mô hình yếu được xây dựng một cách tuần tự, mỗi mô hình mới được huấn luyện để sửa chữa những lỗi mà mô hình trước đó đã mắc phải.

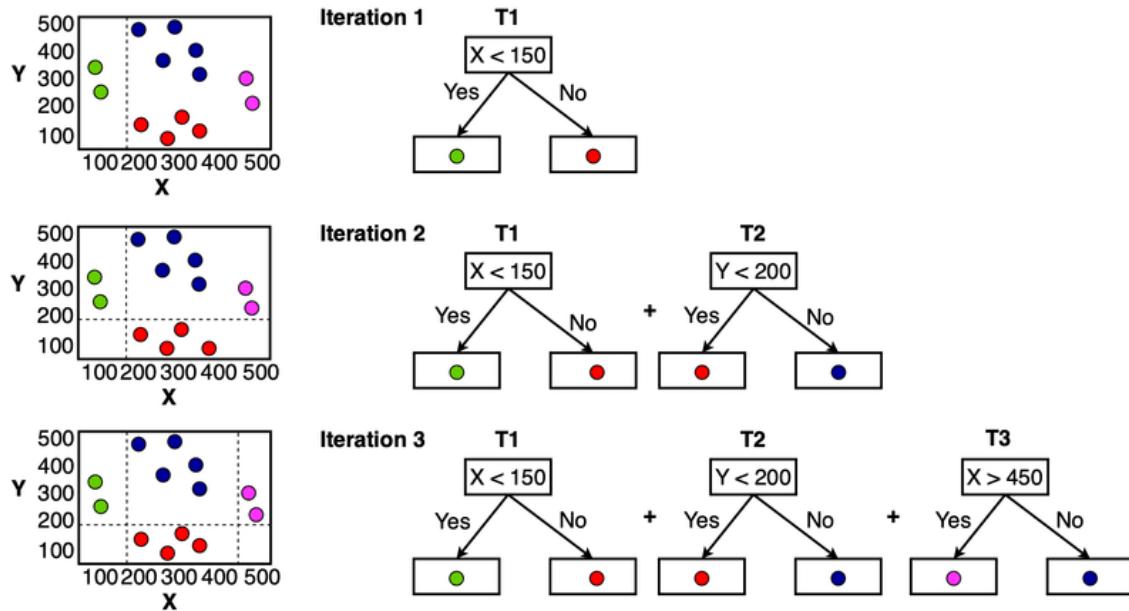
Điểm đỉnh cao của Boosting chính là sự ra đời của Gradient Boosting, một thuật toán đã cách mạng hoá lĩnh vực học máy. Tiếp nối sự thành công đó, XGBoost (eXtreme Gradient Boosting) đã được giới thiệu như một phiên bản tối ưu hoá, nhanh hơn và hiệu quả hơn đáng kể. Kể từ khi ra mắt, XGBoost đã nhanh chóng trở thành một công cụ không thể thiếu đối với các nhà khoa học dữ liệu và kỹ sư học máy, thống trị các cuộc thi Kaggle và được ứng dụng rộng rãi trong nhiều bài toán thực tế trên dữ liệu có cấu trúc.

Bài viết này đi sâu vào cơ chế hoạt động của XGBoost, từ những nguyên lý nền tảng của Gradient Boosting, các cải tiến cốt lõi, đến việc phân tích chi tiết các thành phần toán học và minh họa bằng các ví dụ tính toán thủ công. Mục tiêu là cung cấp một góc nhìn toàn diện và chuyên sâu về thuật toán mạnh mẽ này.

1. Ôn tập về Gradient Boosting

1.1 Nguyên lý hoạt động

Gradient Boosting (GB) là một phương pháp học máy thuộc nhóm ensemble (học mô hình tập hợp) sử dụng kỹ thuật boosting. Thay vì huấn luyện một mô hình phức tạp duy nhất, boosting xây dựng một tập hợp các mô hình đơn giản (yếu) theo tuần tự, mỗi mô hình mới được thêm vào nhằm sửa các lỗi của mô hình trước đó. Cụ thể, thuật toán khởi đầu bằng một mô hình dự đoán cơ sở, sau đó ở mỗi bước lặp sẽ huấn luyện một mô hình đơn giản (ví dụ: cây quyết định nông) để dự đoán phần sai số (residual) của mô hình hiện tại, rồi cộng bổ sung mô hình mới vào mô hình tập hợp. Quá trình này lặp lại nhiều lần giúp giảm dần lỗi dự đoán trên tập huấn luyện.



Hình 1: Minh họa khái niệm Gradient Boosting. Mỗi vòng lặp (Iteration 1, 2, 3) mô hình bổ sung một cây quyết định mới (T_1, T_2, T_3) nhằm dần dần sửa lỗi của mô hình trước. Cuối cùng, các cây yếu được cộng lại thành mô hình mạnh giúp dự đoán tốt hơn

1.2 Quy trình hoạt động

Quy trình hoạt động của thuật toán Gradient Boosting có thể được mô tả qua các bước lặp sau:

- Khởi tạo mô hình ban đầu:** Thuật toán bắt đầu bằng một mô hình dự đoán đơn giản, thường là một hằng số. Trong bài toán hồi quy, mô hình khởi tạo này là giá trị trung bình của biến mục tiêu.
- Tính toán phần dư (pseudo-residuals):** Đây là một bước quan trọng. Phần dư được tính bằng hiệu số giữa giá trị thực tế của biến mục tiêu và giá trị dự đoán của mô hình hiện tại. Các giá trị phần dư này đại diện cho "lỗi" mà mô hình đang mắc phải.
- Huấn luyện mô hình mới:** Một mô hình yếu mới, chẳng hạn như một cây quyết định đơn giản, được huấn luyện. Tuy nhiên, thay vì dự đoán biến mục tiêu ban đầu, cây này được huấn luyện để dự đoán các giá trị phần dư đã tính ở bước trước. Mục tiêu của nó là học các quy tắc để giải thích tại sao các dự đoán trước đó lại sai lệch.
- Cập nhật mô hình tổng hợp:** Dự đoán của cây mới được cộng vào mô hình tổng hợp hiện có, nhưng được điều chỉnh bởi một tham số gọi là tốc độ học (learning rate), thường có giá trị nhỏ (từ 0 đến 1). Tốc độ học giúp kiểm soát mức độ đóng góp của mỗi cây, làm cho quá trình học diễn ra từ từ và chính xác hơn, từ đó giảm nguy cơ quá khớp (overfitting).
- Lặp lại:** Các bước từ 2 đến 4 được lặp lại cho đến khi đạt được một tiêu chí dừng, chẳng hạn như số lượng cây tối đa đã được xây dựng hoặc sai số không còn giảm đáng kể.

1.3 Ví dụ minh họa

Giả sử ta có một bài toán hồi quy đơn giản dự đoán y từ x . Với 1 mô hình ban đầu dự đoán chưa tốt, ta thêm mô hình thứ 2 để hiệu chỉnh.

Chẳng hạn, với một điểm dữ liệu có giá trị thực tế $y = 1$, mô hình đầu tiên dự đoán $\hat{y}^{(1)} = 0.6$. Sai số ở điểm này có thể đo bằng hàm MSE: $L = (1 - 0.6)^2 = 0.16$. Gradient của loss so với dự đoán là $g = -2(y - \hat{y})$, và với MSE thì $g \propto (y - \hat{y})$. Ở ví dụ này $y - \hat{y} = 0.4$ chính là phần dư (residual) mà mô hình thứ hai cần học. Do đó, ta huấn luyện mô hình thứ hai để dự đoán giá trị 0.4 này từ đầu vào x .

Giả sử mô hình thứ hai cho ra $\hat{y}^{(2)} = 0.3$ đối với điểm trên. Lúc này, ta cộng dự đoán của hai mô hình để thu được dự đoán cuối $y_{\text{ensemble}} = 0.6 + 0.3 = 0.9$. Giá trị 0.9 đã gần với nhãn thực tế 1 hơn so với 0.6 ban đầu.

Bằng cách đặt mục tiêu cho mô hình mới là gradient của lỗi (với MSE thì gradient tỉ lệ với phần dư $y - \hat{y}$), Gradient Boosting đảm bảo mỗi mô hình bổ sung sẽ tập trung sửa những lỗi còn lại của mô hình trước đó. Quá trình có thể lặp đi lặp lại với nhiều mô hình con, giúp mô hình tổng thể dần trở nên mạnh mẽ và chính xác hơn.

2. Giới thiệu về XGBoost

2.1 XGBoost là gì?

XGBoost (viết tắt của “Extreme Gradient Boosting”) là một triển khai nâng cao của thuật toán gradient boosting do Tianqi Chen và Carlos Guestrin phát triển (công bố năm 2016). Đúng như tên gọi, XGBoost vẫn tuân theo ý tưởng boosting trên nền gradient: huấn luyện tuần tự nhiều cây quyết định để tối ưu một hàm mất mát bất kỳ. Tuy nhiên, XGBoost được gọi là “cực đại” bởi nó đã mở rộng và cải tiến gradient boosting truyền thống trên nhiều phương diện, từ hàm mục tiêu, tối ưu toán học cho đến hiệu năng tính toán.

2.2 Vai trò của khai triển Taylor cấp hai đến XGBoost

Ôn tập về Gradient Descent và khai triển Taylor

Gradient descent là một thuật toán tối ưu cơ bản dựa trên việc di chuyển theo hướng ngược độ dốc (negative gradient) để tìm giá trị nhỏ nhất của hàm mục tiêu. Trong gradient descent, update rule có dạng:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

trong đó η là tốc độ học (learning rate), và $\nabla L(\theta_t)$ là gradient của hàm mất mát tại θ_t . Khai triển Taylor là một công cụ toán học mạnh mẽ để xấp xỉ một hàm phức tạp bằng một đa thức. Khai triển Taylor của hàm $f(x)$ quanh điểm a có dạng:

$$f(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \dots$$

Trong học máy, người ta thường sử dụng xấp xỉ bậc hai:

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)(\Delta x)^2$$

Điểm cốt lõi của XGBoost nằm ở việc nó tối ưu hàm mục tiêu bằng cách sử dụng khai triển Taylor bậc hai của hàm mất mát (tích hợp cả đạo hàm bậc nhất và bậc hai). Cần nhớ rằng, ở gradient boosting truyền thống, mỗi mô hình mới được thêm vào dựa trên gradient (đạo hàm bậc nhất) của hàm mất mát nhằm giảm lỗi (cách tiếp cận tương tự gradient descent). XGBoost tiến xa hơn một bước: tại mỗi bước thêm cây, XGBoost thực hiện xấp xỉ hàm mất mát bằng một đa thức bậc hai (sử dụng cả gradient và Hessian – đạo hàm bậc hai của hàm mất mát) quanh giá trị dự đoán hiện tại. Điều này cho phép sử dụng phương pháp tối ưu gần với phương pháp Newton-Raphson thay vì chỉ descent đơn thuần, giúp tìm bước cập nhật tối ưu chính xác hơn.

Cụ thể, ký hiệu $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ là gradient và $h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$ là Hessian của hàm mất mát đối với dự đoán hiện tại của mẫu i . Khi thêm cây thứ t , XGBoost xấp xỉ sự giảm loss bằng công thức Taylor bậc hai như sau:

$$\text{obj}^{(t)} \approx \sum_{i=1}^n \left[g_i f_i(x_i) + \frac{1}{2} h_i [f_i(x_i)]^2 \right] + \Omega(f_i)$$

trong đó, $f_t(x)$ là cây quyết định mới được thêm ở bước t , và $\Omega(f_t)$ là hệ số phạt độ phức tạp của cây đó. Biểu thức trên cho thấy XGBoost chỉ phụ thuộc vào g_i , h_i (tính từ mô hình hiện tại) chứ không phụ thuộc trực tiếp vào y_i . Nhờ đó, ta có một công thức tổng quát để tối ưu bất kỳ hàm mất mát nào (miễn là hàm đó khả vi tới cấp hai) – chỉ cần cung cấp công thức tính gradient và Hessian, XGBoost đều có thể booster hoá được (từ hồi quy MSE đến phân loại logistic).

2.3 So sánh XGBoost và Gradient Boosting truyền thống

XGBoost	Gradient Boosting
Regularization: L1 & L2	Regularization: Limited
Parallelization Yes	Parallelization No
Handling Missing Data Yes	Handling Missing Data: Limited
Tree Pruning Depth-wise	Tree Pruning Level-wise
Training Speed Fast	Training Speed Slower

Hình 2: Sơ đồ so sánh XGBoost và Gradient Boosting truyền thống

XGBoost thừa hưởng ý tưởng cơ bản của gradient boosting, nhưng có nhiều cải tiến quan trọng khiến nó thường cho hiệu năng và tốc độ vượt trội so với các cách triển khai Gradient Boosting trước đây. Dưới đây là các điểm khác biệt chính:

- **Regularization (Chính quy hoá):** Thuật toán GBM truyền thống (như trong scikit-learn) thường chỉ tập trung tối thiểu hoá loss trên training, dễ dẫn đến overfitting nếu không kiểm soát độ phức tạp của cây. XGBoost tích hợp trực tiếp các thành phần chính quy L_1 và L_2 vào hàm mục tiêu để phạt các mô hình phức tạp. Cụ thể, XGBoost định nghĩa hàm phạt mô hình cây

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

trong đó T là số lá, w_j là giá trị tại lá j . Tham số λ kiểm soát độ lớn các giá trị lá (L_2 regularization), còn γ quy định mức giảm loss tối thiểu để phép chia được chấp nhận (giống một dạng regularization về cấu trúc cây). Ngoài ra, XGBoost cũng hỗ trợ phạt L_1 (tham số α) để khuyến khích nhiều giá trị lá bằng 0 (điều này tương tự việc tẩy bớt những lá không đóng góp nhiều). Nhờ regularization, mô hình XGBoost có xu hướng đơn giản hơn, tránh overfit tốt hơn GBM thường.

- **Parallelization (Song song hoá):** Việc huấn luyện mô hình boosting là tuần tự qua các vòng lặp, nhưng bên trong việc xây dựng mỗi cây quyết định, XGBoost đã tối ưu để có thể tận dụng tính song song. Thuật toán chia dữ liệu và tìm kiếm điểm chia tốt nhất trên các thuộc tính có thể được thực hiện đồng thời trên nhiều lõi CPU, rút ngắn đáng kể thời gian huấn luyện so với các triển khai GBM thông thường.Thêm vào đó, XGBoost giới thiệu cấu trúc dữ liệu DMatrix tối ưu cho việc duyệt cây, giúp giảm truy cập bộ nhớ và tăng tốc tính toán.
- **Handling Missing Values (Xử lý dữ liệu khuyết):** XGBoost có khả năng xử lý giá trị thiếu một cách tự động trong quá trình xây dựng cây. Thuật toán sử dụng phương pháp “sparsity-aware split finding” – khi gặp dữ liệu thưa (bao gồm giá trị thiếu, zero hoặc one-hot nhiều zero), XGBoost thử gán các mẫu khuyết vào nhánh trái hoặc phải, chọn cách gán nào tối ưu loss nhỏ nhất. Nhờ đó, ta không cần xử lý thiếu dữ liệu trước khi đưa vào XGBoost, trong khi GBM truyền thống yêu cầu tiền xử lý như điền giá trị trung bình hoặc tạo cờ đánh dấu.
- **Tree Pruning (Tỉa cây):** Cả GBM và XGBoost đều có chiến lược tránh overfitting bằng cách giới hạn độ sâu tối đa của cây. Tuy nhiên, khi quyết định ngưng phát triển một nhánh, GBM truyền thống thường sử dụng chiến lược “dừng sớm tại chỗ” – nếu một split không giảm đủ lỗi thì ngừng chia tiếp tại nút đó. Cách này mang tính cục bộ và có thể bỏ lỡ những phân chia hữu ích ở tầng dưới. Ngược lại, XGBoost chọn cách xây cây “đầy đủ” đến độ sâu cho trước, sau đó hậu xử lý cắt tỉa các nhánh không hiệu quả (prune) dựa trên giá trị gain thực tế. Cụ thể, XGBoost tính độ giảm loss (Gain) của mỗi split và sẽ cắt bỏ các nhánh mà Gain < γ (tham số phạt cấu trúc). Cách tiếp cận “lớn trước, cắt sau” này giúp xem xét bức tranh toàn cục, đảm bảo không bỏ sót các tương tác sâu có ý nghĩa.
- **In-built Cross-Validation:** Thêm một điểm tiện lợi là XGBoost hỗ trợ sẵn cơ chế cross-validation trong quá trình huấn luyện (through API cv()), giúp theo dõi trực tiếp độ lỗi trên tập kiểm tra sau mỗi vòng boosting để tìm điểm dừng tối ưu, thay vì phải tự tách tập và theo dõi ngoài như GBM thông thường.

Tóm lại, XGBoost vẫn dựa trên nền tảng gradient boosting nhưng “extreme” ở chỗ bổ sung đầy đủ các yếu tố regularization, tối ưu tính toán và tiện ích, giúp nó trở thành một trong những thuật toán mạnh mẽ và phổ biến nhất cho dữ liệu bảng (tabular data).

3. Cách xây dựng XGBoost

3.1 Bài toán

Tương tự GBM, mục tiêu của XGBoost là tìm mô hình tổng $F(x) = \sum_{t=0}^T f_t(x)$ (tổng của mô hình cơ sở F_0 và T cây quyết định f_t) sao cho hàm mục tiêu dưới đây được tối thiểu hóa:

$$\text{Obj} = \sum_{i=1}^n l(y_i, F(x_i)) + \sum_{i=1}^T \Omega(f_i)$$

trong đó $l(y, \hat{y})$ là hàm mất mát (ví dụ MSE, log-loss, v.v.) và $\Omega(f_t)$ là hàm phạt độ phức tạp của cây f_t như đã nêu (phụ thuộc vào số lá T và độ lớn các giá trị lá w_j). So với bài toán của GBM thuận túy, XGBoost thêm về thứ hai Ω để kiểm soát mô hình không quá phức tạp.

3.2 Cách xây dựng từng bước

Xây dựng mô hình XGBoost

1. **Khởi tạo** F_0 : $F_0(x) = \arg \min_{\theta} \sum_{i=1}^N l(y_i, \theta)$, chặng hạn:

- Với bài toán hồi quy với hàm mất mát MSE: $F_0 = \frac{1}{N} \sum_i y_i$.
- Với bài toán phân loại nhị phân với hàm mất mát logistics: $F_0 = \log \frac{\hat{y}}{1 - \hat{y}}$ với $\hat{y} = \sum_i y_i$.

2. **Lặp với** $t = 1, 2, \dots, M$:

(a) **Tính gradient và Hessian**:

$$g_i = \left. \frac{\partial l(y_i, \hat{y})}{\partial \hat{y}} \right|_{\hat{y}=F_{t-1}(x_i)}, \quad h_i = \left. \frac{\partial^2 l(y_i, \hat{y})}{\partial \hat{y}^2} \right|_{\hat{y}=F_{t-1}(x_i)}$$

Ví dụ:

- Với bài toán hồi quy với hàm mất mát MSE: $g_i = \hat{y}_i - y_i$, $h_i = 1$.
- Với bài toán phân loại nhị phân với hàm mất mát logistics: $g_i = p_i - y_i$, $h_i = p_i(1 - p_i)$ với $p_i = \sigma(\hat{y}_i)$.

(b) **Xây cây f_t** :

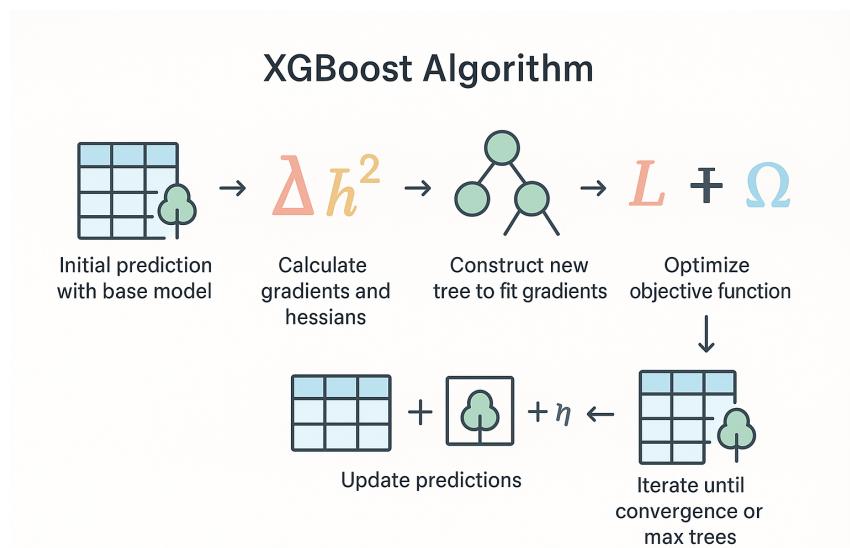
- Tại mỗi nút chứa tập mẫu I , đặt $G = \sum_{i \in I} g_i$, $H = \sum_{i \in I} h_i$.
- Với mỗi đặc trưng k , sắp xếp toàn bộ các mẫu trong node theo giá trị của đặc trưng k . Ban đầu đặt $G_L = 0$, $H_L = 0$, $G_R = G$, $H_R = H$. Sau đó, thử từng ngưỡng là trung bình các cặp giá trị từ trái sang phải, mỗi ngưỡng tính toán gradient và hessian của các mẫu bên trái và bên phải.
- Tính Gain: $\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} + \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$.
- Giữ lại ngưỡng có gain lớn nhất. Nếu $\text{Gain} \leq 0$ hoặc đạt ngưỡng dừng, biến nút thành nút lá. Ngược lại, tách nút và lặp lại.

(c) **Tính trọng số lá**: Với mỗi lá j có tập I_j :

$$G_j = \sum_{i \in I_j} g_i, \quad H_j = \sum_{i \in I_j} h_i, \quad w_j^* = -\frac{G_j}{H_j + \lambda}$$

(d) **Cập nhật mô hình**: $F_t(x) = F_{t-1}(x) + \eta f_t(x)$, trong đó $f_t(x) = w_{q_t(x)}$ là trọng số lá mà x rơi vào.

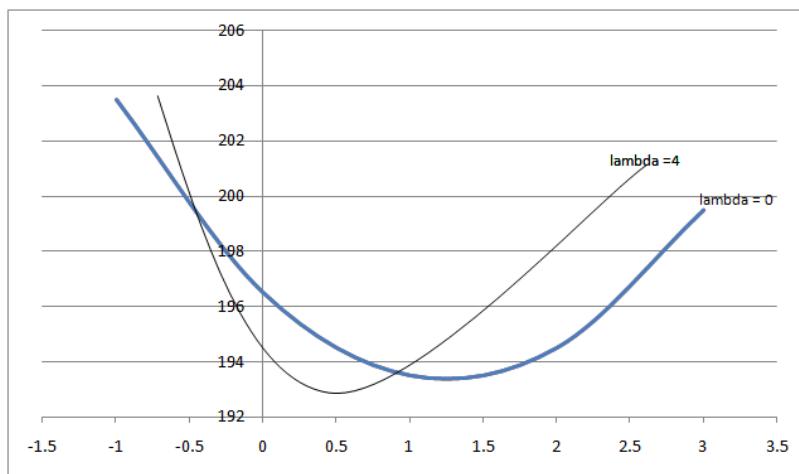
3. **Kết quả cuối cùng**: Sau M vòng, ta thu được mô hình $F_M(x)$.



Hình 3: Quy trình xây dựng XGBoost từng bước với tối ưu hàm mục tiêu

Quy trình trên cho thấy cách XGBoost xây dựng cây tương tự như GBM nhưng thêm các yếu tố hình phạt và tối ưu Newton. Nhờ đó, mỗi bước thêm cây của XGBoost thường đạt được giảm loss nhiều hơn so với GBM thông thường (với cùng cấu trúc cây), đồng thời kiểm soát độ phức tạp để tránh overfit.

Hiệu quả của regularization λ : Để hình dung tác động của tham số λ (L2) lên giá trị lá, ta xét đồ thị hàm loss theo giá trị lá (với cấu trúc cây cố định). Nếu không có regularization ($\lambda = 0$), giá trị tối ưu của lá chính là đỉnh đáy của parabol loss (thường có độ lớn lớn hơn). Khi tăng λ , đồ thị loss sẽ bị “phẳng” hơn (ít dốc hơn), làm điểm tối ưu dịch chuyển về gần 0 hơn. Nói cách khác, λ cao buộc các giá trị lá gần 0 hơn, tức mô hình điều chỉnh ít hơn và tránh overfit.



As the value of lambda increases, the lowest point of parabola shift towards zero, and this is what regularization does.

Hình 4: Ảnh hưởng của tham số regularization λ đến giá trị tối ưu của một lá (minimize loss). Đường màu xanh ứng với $\lambda = 0$ có điểm tối ưu (đáy parabol) lệch xa 0 hơn so với đường màu đen ứng với $\lambda = 4$ – điểm tối ưu dịch chuyển rõ rệt về 0. Regularization làm mô hình “thận trọng” hơn khi điều chỉnh dự đoán.

4. Các ví dụ về XGBoost

4.1 Bài toán hồi quy

Xét một bài toán hồi quy đơn giản: dự đoán giá trị y từ biến đầu vào x . Giả sử tập dữ liệu huấn luyện gồm 3 điểm: $(x=1, y=3)$, $(x=2, y=5)$, $(x=3, y=7)$. Ta sẽ bước qua 2 vòng lặp của XGBoost (với tham số $\lambda = 0$ để đơn giản việc tính toán, và tốc độ học $\eta = 1$).

1. Vòng lặp 1:

- (a) **Khởi tạo $F_0(x)$:** Ở bài toán hồi quy, ta khởi đầu bằng dự đoán hằng bằng trung bình của các y . Do các giá trị y là 3, 5, 7 nên trung bình $\bar{y} = 5$. Vậy mô hình ban đầu $F_0(x)$ dự đoán 5 cho tất cả các x . Lúc này vector dự đoán ban đầu: $\hat{y} = [5, 5, 5]$ cho các điểm 1,2,3.
- (b) **Tính residual 1:** Tính phần dư (residual) của từng điểm: $r_i = y_i - F_0(x_i)$. Vector residual đầu tiên là $[-2, 0, 2]$. Residual âm nghĩa là dự đoán đang cao hơn thực tế (cần điều chỉnh xuống), residual dương nghĩa là dự đoán đang thấp (cần điều chỉnh tăng).
- (c) **Xây dựng cây $f_1(x)$:** Dựa trên các residual này, ta huấn luyện một cây quyết định nhỏ (một gốc hai lá) để dự đoán r từ x . Trực quan, residual có vẻ âm cho x nhỏ và dương cho x lớn, nên một cây quyết định với một split sẽ phù hợp. Giả sử thuật toán tìm được điểm chia tối ưu là $x \leq 2$:
 - Các điểm thỏa mãn $x \leq 2$ gồm $x = 1, 2$ (residual lần lượt -2 và 0). Gọi tập này là I_L (nhánh trái).
 - Các điểm $x > 2$ gồm $x = 3$ (residual 2). Gọi tập này là I_R (nhánh phải).
 Giá trị lá tối ưu có thể tính nhanh bằng trung bình residual của mỗi nhánh (do đang minimize MSE và $\lambda = 0$).
 - Nhánh trái I_L : trung bình $\frac{-2 + 0}{2} = -1$.
 - Nhánh phải I_R : trung bình $\frac{2}{1} = 2$.

Ta cũng có thể kiểm chứng bằng công thức đã nêu: $w_L^* = -\frac{\sum_{i \in I_L} g_i}{\sum_{i \in I_L} h_i}$, với MSE thì $g_i = F_0(x_i) - y_i = -r_i, h_i = 1$.

Do đó $\sum_{I_L} g = (-(-2) + -(0)) = 2, \sum_{I_L} h = 2$, suy ra $w_L^* = -\frac{2}{2} = -1$. Tương tự $w_R^* = -\frac{\sum_{I_R} g}{\sum_{I_R} h} = -\frac{-2}{1} = 2$ – trùng khớp với kết quả trên. Vậy cây $f_1(x)$ sẽ dự đoán -1 với $x \leq 2$ và $+2$ với $x > 2$.

- (d) **Cập nhật mô hình:** Mô hình mới $F_1(x) = F_0(x) + f_1(x)$ (vì $\eta = 1$). Nghĩa là:
 - Với các điểm $x \leq 2$; $F_1(x) = 5 + (-1) = 4$.
 - Với $x > 2$; $F_1(x) = 5 + 2 = 7$.

Áp dụng vào dữ liệu:

- Điểm $x = 1$: $F_1(1) = 4$ (dự đoán giảm từ 5 xuống 4, gần với giá trị thực 3 hơn trước).
- Điểm $x = 2$: $F_1(2) = 4$ (dự đoán 4, trong khi thực tế 5, ta đã “tụt hơi quá đà” một chút).
- Điểm $x = 3$: $F_1(3) = 7$ (dự đoán tăng từ 5 lên 7, khớp đúng giá trị thực 7).

Sau vòng 1, tổng bình phương lỗi đã giảm đáng kể (trước đó điểm 1 lệch 2 đơn vị, điểm 3 lệch 2 đơn vị; nay điểm 3 đã khớp, điểm 1 lệch 1 đơn vị, điểm 2 lệch 1 đơn

vị).

2. Vòng lặp 2:

(a) **Tính residual 2:** Tính tiếp residual so với mô hình $F_1: r^{(2)} = y - F_1(x)$. Ta được:

- $x = 1: 3 - 4 = -1$,
- $x = 2: 5 - 4 = 1$,
- $x = 3: 7 - 7 = 0$.

Residual mới: $[-1, 1, 0]$. (Điểm 1 dự đoán hơi cao, điểm 2 dự đoán hơi thấp, điểm 3 đã dự đoán đúng nên residual = 0).

(b) **Xây dựng cây $f_2(x)$:** Lần này, ta huấn luyện cây để dự đoán các residual $[-1, 1, 0]$.

Một lần nữa, rõ ràng có sự phân tách: điểm $x = 1$ có residual âm -1 , còn $x = 2, 3$ có residual không âm (1 và 0). Cây mới có thể lại chọn ngưỡng $x \leq 1.5$ (tức tách điểm 1 riêng một nhánh). Nhánh trái I_L ($x=1$) có residual -1 , trung bình lá $w_L = -1$. Nhánh phải I_R ($x=2,3$) residual trung bình $(1 + 0)/2 = 0.5$, nên $w_R = 0.5$. Vậy $f_2(x)$ dự đoán -1 cho $x \leq 1.5$ và $+0.5$ cho $x > 1.5$.

(c) **Cập nhật mô hình:** $F_2(x) = F_1(x) + f_2(x)$:

- Điểm $x = 1: F_2(1) = 4 + (-1) = 3$ (giờ khớp chính xác $y = 3$).
- Điểm $x = 2: F_2(2) = 4 + 0.5 = 4.5$ (dự đoán 4.5 so với $y=5$, còn thiếu 0.5).
- Điểm $x = 3: F_2(3) = 7 + 0.5 = 7.5$ (dự đoán 7.5 so với $y=7$, hơi dư 0.5).

Sau 2 vòng, dự đoán đã rất gần giá trị thực (sai số nhỏ 0.5). Nếu tiếp tục vòng 3, mô hình sẽ điều chỉnh những sai số nhỏ còn lại (với $\eta = 1$ có thể điều chỉnh đúng luôn trong vòng 3). Trong thực tế, ta thường dùng $\eta < 1$ và nhiều vòng lặp để mô hình dần dần hội tụ ổn định.

4.2 Bài toán phân loại nhị phân

Xét bài toán phân loại nhị phân đơn giản: dự đoán liệu một người có mua sản phẩm không (y) (1 - mua, 0 - không mua) dựa trên hai đặc trưng. Ví dụ, giả sử ta có 4 điểm dữ liệu huấn luyện:

ID	Thu nhập (x_1)	Tuổi (x_2)	Mua sản phẩm (y)
1	15	25	1
2	30	35	0
3	25	30	1
4	18	40	1

Ở đây, ta sẽ minh họa vòng lặp đầu tiên của XGBoost sử dụng hàm mất mát log-loss, tốc độ học $\eta = 0.1$ và $\lambda = 1$.

1. **Khởi tạo dự đoán ban đầu:** Đối với phân loại nhị phân, dự đoán ban đầu thường là một giá trị log-odds. Để đơn giản, chúng ta có thể giả sử dự đoán ban đầu có xác suất là 0.5 ($p^{(0)} = 0.5$) cho tất cả các mẫu.
2. **Tính toán Gradient và Hessian:** Đạo hàm bậc nhất (g_i) và đạo hàm bậc hai (h_i) của hàm này theo xác suất p được tính như sau:

- $g_i = p^{(t-1)} - y_i$
- $h_i = p^{(t-1)}(1 - p^{(t-1)})$

Với vòng lặp đầu tiên:

ID	y	$g_i = 0.5 - y_i$	$h_i = 0.5(1 - 0.5)$
1	1	$0.5 - 1 = -0.5$	0.25
2	0	$0.5 - 0 = 0.5$	0.25
3	1	$0.5 - 1 = -0.5$	0.25
4	1	$0.5 - 1 = -0.5$	0.25

3. **Xây dựng cây đầu tiên:** Thuật toán tìm điểm chia tốt nhất bằng cách tối đa hóa Gain. Giả sử thuật toán tìm thấy điểm chia tốt nhất là $\text{Thu nhập} < 20$.

- **Lá trái:** Chứa các mẫu có Thu nhập < 20 (ID 1, 4).
- **Lá phải:** Chứa các mẫu có Thu nhập ≥ 20 (ID 2, 3).

4. **Tính toán giá trị lá:** XGBoost tính giá trị tối ưu của mỗi lá (w_j^*) bằng công thức:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

- **Lá trái:** $\sum g_i = -1.0$, $\sum h_i = 0.5$, suy ra $w_L^* = -\frac{-1.0}{0.5 + 1} = -\frac{-1}{1.5} \approx 0.667$.
- **Lá phải:** $\sum g_i = 0$, $\sum h_i = 0.5$, suy ra $w_R^* = 0$.

5. **Cập nhật dự đoán:** Dự đoán mới (trên thang log-odds) được cập nhật bằng cách cộng dự đoán cũ với giá trị của lá tương ứng, nhân với tốc độ học η :

$$\text{Pred}^{(1)} = \text{Pred}^{(0)} + \eta \cdot w_{\text{lá}}$$

Tuy nhiên, vì dự đoán ban đầu là xác suất (0.5), chúng ta cần chuyển đổi nó về log-odds để thực hiện phép toán:

$$\text{Pred}^{(0)} = \ln \frac{p}{1-p} = \ln \frac{0.5}{0.5} = 0$$

- **ID 1 & 4 (Lá trái):** $\text{Pred}_{\text{log-odds}}^{(1)} = 0 + 0.1 \cdot 0.667 = 0.0667$.
- **ID 2 & 3 (Lá phải):** $\text{Pred}_{\text{log-odds}}^{(1)} = 0 + 0.1 \cdot 0 = 0$.

Cuối cùng, chúng ta chuyển đổi lại về xác suất bằng hàm sigmoid: $p = \frac{1}{1 + e^{-\text{Pred}_{\text{log-odds}}}}$.

- **ID 1 & 4 (Lá trái):** $p^{(1)} = \frac{1}{1 + e^{-0.0667}} \approx 0.5167$.
- **ID 2 & 3 (Lá phải):** $p^{(1)} = \frac{1}{1 + e^{-0}} = 0.5$.

Quá trình lặp lại với các giá trị dự đoán mới này cho đến khi đạt tiêu chí dừng.

5. Kết luận

XGBoost là một minh chứng xuất sắc cho sự kết hợp giữa lý thuyết nền tảng vững chắc và các tối ưu hóa kỹ thuật thông minh. Bằng cách kế thừa nguyên lý của Gradient Boosting và cải tiến nó với khai triển Taylor bậc hai, các thành phần điều chỉnh tích hợp, và khả năng tính toán song song, XGBoost đã giải quyết hiệu quả nhiều hạn chế của các thuật toán tiền nhiệm.

Ngày nay, với hiệu suất vượt trội, tốc độ huấn luyện nhanh và khả năng xử lý dữ liệu lớn, XGBoost tiếp tục là một trong những thuật toán được lựa chọn hàng đầu cho các bài toán trên dữ liệu có cấu trúc. Mặc dù việc tinh chỉnh các siêu tham số có thể đòi hỏi kinh nghiệm và hiểu biết sâu sắc, việc nắm vững cơ chế hoạt động của thuật toán là chìa khóa để khai thác tối đa sức mạnh của nó, mang lại những kết quả ấn tượng trong cả nghiên cứu và ứng dụng thực tế.

Hành Trình Của Các Thuật Toán Cây: Từ Nhánh Cây Đơn Độc Đến Khu Rừng Bất Bại

Vũ Thái Sơn

”Từ một cây quyết định đơn lẻ đến sức mạnh của cả khu rừng”

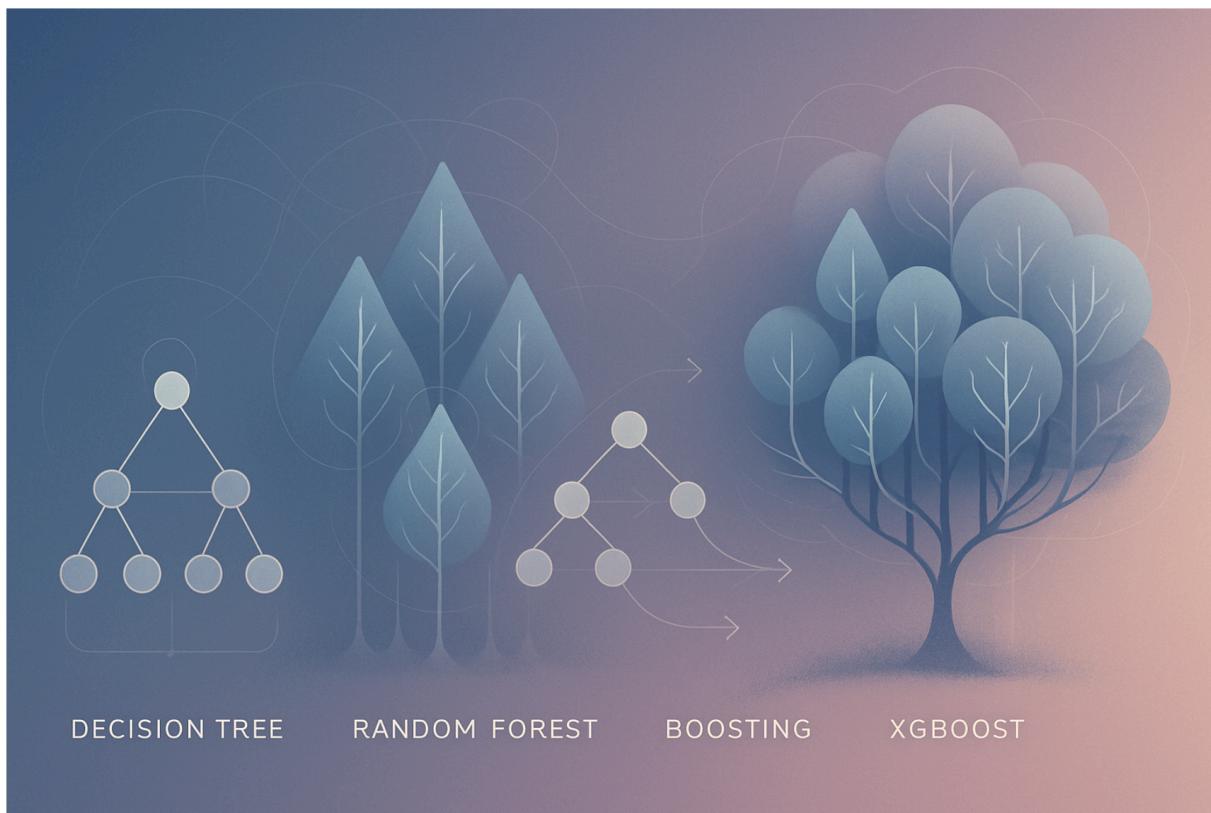
1. Lời Dẫn Nhập

Trong thế giới Trí tuệ Nhân tạo, khi nhắc đến dữ liệu phi cấu trúc như hình ảnh hay ngôn ngữ, Deep Learning dường như là một vị vua không ngai. Tuy nhiên, khi bước vào địa hạt của dữ liệu dạng bảng (tabular data) – loại dữ liệu phổ biến nhất trong các bài toán kinh doanh, từ dự báo tài chính, phân tích rủi ro đến hành vi khách hàng – một họ thuật toán khác lại vươn lên chiếm lĩnh vị thế thống trị: các mô hình cây (Tree-based Models).

Một nghiên cứu quy mô lớn năm 2022 của Grinsztajn và các cộng sự đã chỉ ra rằng, trên hàng chục bộ dữ liệu dạng bảng, các mô hình cây tăng cường như XGBoost thường xuyên cho hiệu suất vượt trội so với cả những kiến trúc Deep Learning phức tạp nhất [1]. Tại sao lại có sự ưu ái này? Điều gì đã tạo nên sức mạnh đáng kinh ngạc của chúng?

Bài viết này sẽ đưa bạn vào một cuộc hành trình khám phá sự tiến hóa của các thuật toán cây. Chúng ta sẽ bắt đầu từ ”tổ tiên” sơ khai nhất là **Cây Quyết Định (Decision Tree)**, chứng kiến cách nó khắc phục điểm yếu thông qua ”trí tuệ đám đông” của **Random Forest**, học cách ”sửa sai” một cách thông minh qua các thế hệ **Boosting**, và cuối cùng đạt đến đỉnh cao với **XGBoost** – nhà vô địch của các cuộc thi khoa học dữ liệu.

Xuyên suốt hành trình, chúng ta sẽ sử dụng một ví dụ duy nhất – một tập dữ liệu con về **Dự đoán Bệnh tim** – để mô tả cách từng thuật toán suy nghĩ và đưa ra quyết định.



Hình 5: Sự tiến hóa của các thuật toán cây từ Decision Tree đến XGBoost.

2. Ví Dụ Xuyên Suốt: Tập Dữ Liệu Bệnh Tim

Để các khái niệm toán học không trở nên khô khan, chúng ta sẽ áp dụng chúng vào một bài toán thực tế. Giả sử chúng ta có một tập dữ liệu nhỏ gồm 8 bệnh nhân, với mục tiêu dự đoán liệu họ có mắc bệnh tim hay không (HeartDisease = 1) dựa trên ba đặc trưng:

Bảng 1: Tập dữ liệu ví dụ về bệnh tim.

Bệnh nhân	Age	MaxHeartRate	ChestPain	HeartDisease
1	29	202	0	0
2	48	150	1	1
3	52	168	1	1
4	55	145	0	0
5	58	120	1	1
6	61	162	0	0
7	65	148	1	1
8	68	130	0	1

Lưu ý: *ChestPain = 1 có nghĩa là loại đau ngực có nguy cơ cao, 0 là loại không nguy cơ.*

3. Chương 1: Decision Tree - Nền Móng Của Trực Giác

3.1 Ý Nghĩa và Phương Pháp

Decision Tree là thuật toán mô phỏng lại quá trình ra quyết định của con người một cách tự nhiên nhất. Nó là một cấu trúc dạng cây, trong đó mỗi nút trong đại diện cho một "câu hỏi" về một đặc trưng, mỗi nhánh đại diện cho một "câu trả lời", và mỗi nút lá đại diện cho một quyết định cuối cùng.

Để xây dựng cây, thuật toán phải trả lời câu hỏi cốt lõi: *Tại mỗi bước, đâu là câu hỏi tốt nhất để chia dữ liệu?* "Tốt nhất" ở đây có nghĩa là phép chia đó tạo ra các nhóm con "thuần khiết" nhất về mặt nhãn. Với bài toán phân loại, độ thuần khiết thường được đo bằng:

- **Gini Impurity:** Đo lường xác suất một mẫu được chọn ngẫu nhiên từ một nhóm sẽ bị phân loại sai nếu chúng ta gán nhãn cho nó theo phân phối của nhóm đó. Gini càng gần 0, nhóm càng thuần khiết. Công thức là: $Gini = 1 - \sum_j p_j^2$.
- **Information Gain (Dựa trên Entropy):** Đo lường mức độ giảm "hỗn loạn" (Entropy) sau khi thực hiện phép chia. Thuật toán sẽ ưu tiên phép chia nào mang lại Information Gain cao nhất. $H(X) = -\sum_{i=1}^n p_i \log_2 p_i$.

3.2 Áp dụng vào ví dụ

Tại nút gốc, chúng ta có 8 bệnh nhân, trong đó 4 người bị bệnh (1) và 4 người không (0). Độ "ô uế" Gini ban đầu của nút gốc là:

$$Gini_{root} = 1 - ((4/8)^2 + (4/8)^2) = 1 - (0.25 + 0.25) = 0.5$$

Bây giờ, thuật toán thử chia bằng câu hỏi **MaxHeartRate ≤ 156** :

- **Nhánh True (≤ 156):** Gồm bệnh nhân {2, 4, 5, 7, 8}. Nhóm này có 4 người bệnh (1) và 1 người không (0).

$$Gini_{true} = 1 - ((4/5)^2 + (1/5)^2) = 1 - (0.64 + 0.04) = 0.32$$

- **Nhánh False (> 156):** Gồm bệnh nhân {1, 3, 6}. Nhóm này có 1 người bệnh (1) và 2 người không (0).

$$Gini_{false} \approx 1 - ((1/3)^2 + (2/3)^2) \approx 1 - (0.11 + 0.44) = 0.45$$

Gini có trọng số sau khi chia:

$$Gini_{split} = (5/8) \times Gini_{true} + (3/8) \times Gini_{false} = 0.625 \times 0.32 + 0.375 \times 0.45 \approx 0.369$$

Lợi ích (Information Gain) của phép chia này là: $Gini_{root} - Gini_{split} = 0.5 - 0.369 = 0.131$. Thuật toán sẽ tính toán tương tự cho tất cả các phép chia khả dĩ khác (ví dụ: Age ≤ 56 ? hay ChestPain = 1?) và chọn phép chia có lợi ích lớn nhất.

3.3 Ưu và Nhược Điểm

- **Ưu điểm:**

- **Dễ diễn giải:** Đây là mô hình "hộp trắng" (white-box), cho phép chúng ta hiểu rõ logic ra quyết định.
- **Không yêu cầu chuẩn hóa dữ liệu:** Vì các quyết định dựa trên ngưỡng, việc co giãn thang đo của đặc trưng không ảnh hưởng đến kết quả.

- **Nhược điểm:**

- **Phương sai cao (High Variance):** Cây rất nhạy cảm với những thay đổi nhỏ trong dữ liệu huấn luyện. Một vài điểm dữ liệu khác biệt có thể tạo ra một cấu trúc cây hoàn toàn khác.
- **Dễ bị quá khớp (Overfitting):** Cây có xu hướng phát triển rất phức tạp để khớp hoàn hảo với dữ liệu huấn luyện, dẫn đến việc hoạt động kém trên dữ liệu mới.

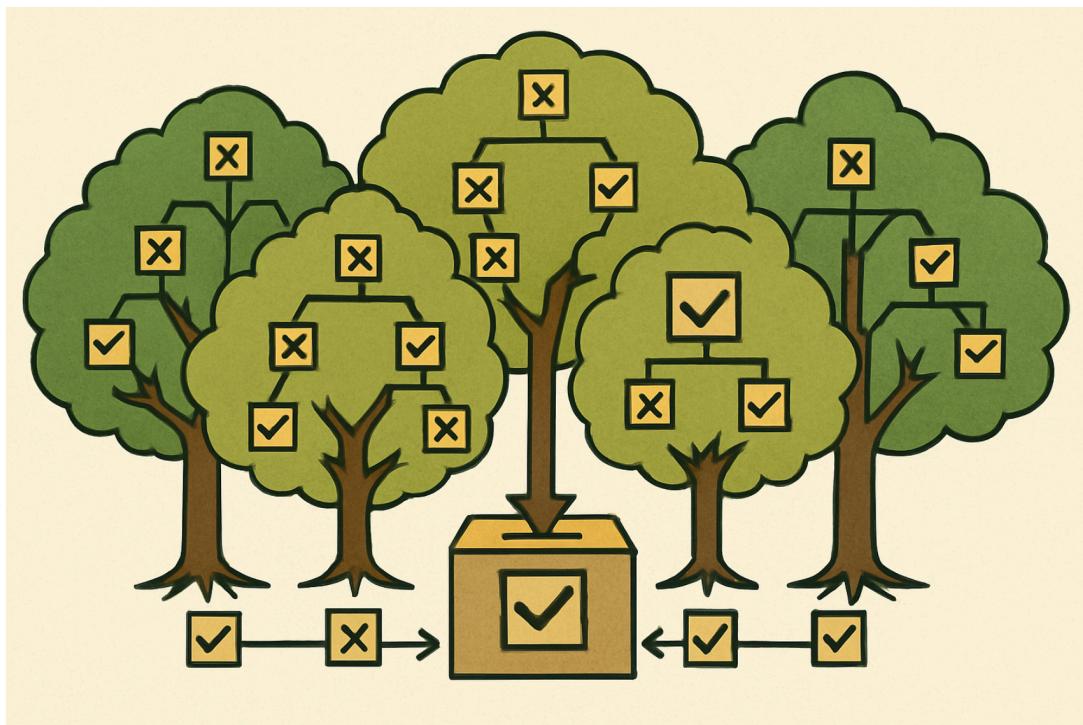
3.4 Phân Tích Chuyên Sâu

Nhược điểm chí mạng về overfitting là động lực chính cho sự ra đời của các thuật toán tiếp theo. Một cây quyết định đơn lẻ giống như một chuyên gia chỉ giỏi về một lĩnh vực rất hẹp và có nhiều định kiến; nó có thể ghi nhớ tất cả các chi tiết của dữ liệu đã thấy nhưng lại bối rối trước các tình huống mới. Để khắc phục điều này và có một quyết định khách quan hơn, chúng ta cần một "hội đồng chuyên gia". Đây chính là ý tưởng đằng sau Random Forest.

4. Chương 2: Random Forest - Sức Mạnh Của "Hội Đồng Cây"

4.1 Ý Tưởng Cải Tiết

Random Forest giải quyết bài toán overfitting của Decision Tree bằng cách xây dựng một "khu rừng" gồm hàng trăm, hàng nghìn cây quyết định khác nhau và lấy kết quả dự đoán bằng cách bỏ phiếu theo số đông. Chìa khóa ở đây là làm cho các cây trong rừng trở nên **đa dạng** và **ít tương quan** với nhau. Sự đa dạng này được tạo ra bởi hai kỹ thuật cốt lõi.

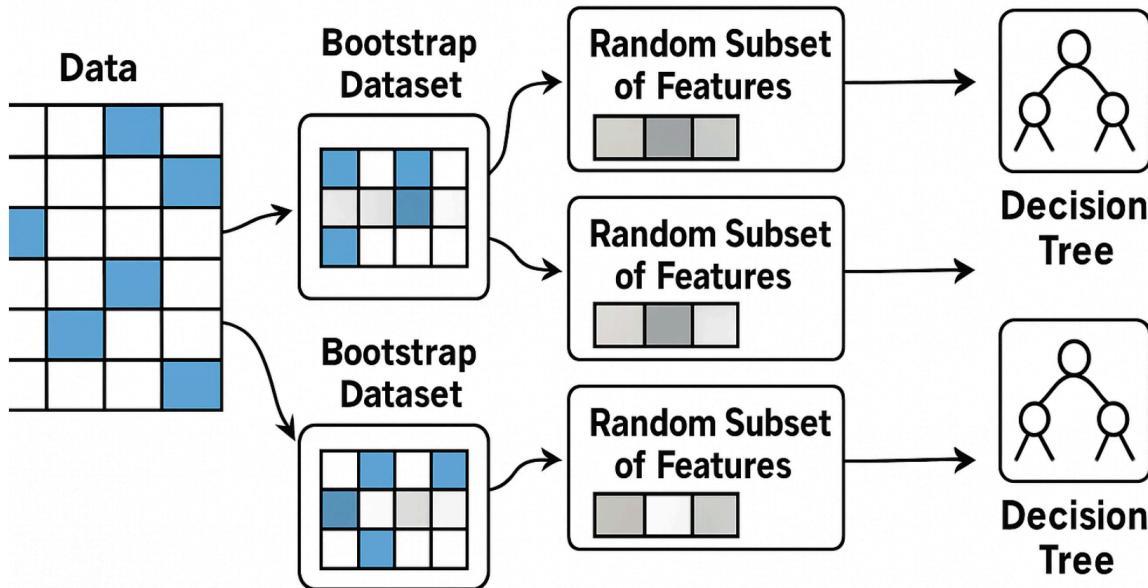


Hình 6: Random Forest kết hợp dự đoán từ nhiều cây.

4.2 Phương Pháp

- **Bagging (Bootstrap Aggregating):** Mỗi cây trong rừng không được huấn luyện trên toàn bộ dữ liệu gốc. Thay vào đó, nó được huấn luyện trên một mẫu con được lấy ngẫu nhiên có lặp lại từ dữ liệu gốc (bootstrap sample). Điều này đảm bảo mỗi cây sẽ "nhìn" vào một phiên bản hơi khác của dữ liệu.
- **Feature Randomness (Ngẫu nhiên hóa đặc trưng):** Tại mỗi nút của mỗi cây, khi tìm kiếm điểm chia tốt nhất, thuật toán không xem xét tất cả các đặc trưng. Thay vào đó, nó chỉ chọn một *tập con ngẫu nhiên* các đặc trưng. Ví dụ, thay vì xét cả Age, MaxHeartRate, ChestPain, nó có thể chỉ chọn ngẫu nhiên Age và ChestPain để tìm điểm chia tốt nhất.

Bagging and Feature Randomness in Random Forest



Hình 7: Kỹ thuật Bagging và Feature Randomness trong Random Forest.

Sự ngẫu nhiên kép này (cả trên mẫu và trên đặc trưng) tạo ra một tập hợp các cây rất đa dạng và **ít tương quan (de-correlated)** với nhau. Mỗi cây sẽ có những điểm mạnh và điểm yếu riêng, nhưng khi kết hợp lại, các sai sót của chúng sẽ tự triệt tiêu lẫn nhau.

4.3 Ưu và Nhược Điểm

- **Điểm cải tiến:**
 - **Giảm phương sai mạnh mẽ:** Bằng cách lấy trung bình dự đoán từ nhiều cây khác nhau, các lỗi riêng lẻ của từng cây sẽ tự triệt tiêu, giúp mô hình trở nên ổn định và chống overfitting cực kỳ hiệu quả.
- **Ưu điểm:**
 - Độ chính xác thường cao hơn nhiều so với một cây đơn.
 - Ít cần tinh chỉnh siêu tham số và hoạt động tốt ngay từ đầu.
- **Nhược điểm:**
 - **Mất tính diễn giải:** Mô hình trở thành ”hộp đen” (black-box). Khó để hiểu tại sao hàng trăm cây lại đưa ra một quyết định cụ thể.
 - Tốn nhiều tài nguyên tính toán hơn để huấn luyện và dự đoán.

4.4 Phân Tích Chuyên Sâu: Bài Toán Cân Bằng Bias-Variance

Trong học máy, lỗi của một mô hình có thể được phân rã thành **Bias (độ chêch)** và **Variance (phương sai)**.

- **Bias** là sai số do các giả định đơn giản hóa của mô hình. Mô hình có bias cao không thể nắm bắt được quy luật phức tạp của dữ liệu.
- **Variance** là sai số do độ nhạy của mô hình với những thay đổi nhỏ trong dữ liệu huấn luyện. Mô hình có phương sai cao bị overfitting.

Một cây quyết định đơn lẻ (để sâu) thường có **bias thấp** (vì nó có thể khớp với mọi chi tiết) nhưng **phương sai rất cao** (rất không ổn định). Random Forest là một kỹ thuật **giảm phương sai (variance reduction)** điển hình. Nó chấp nhận một mô hình cơ sở có phương sai cao và dùng phương pháp Bagging để kiểm soát phương sai đó. Tuy nhiên, nó không làm gì để giảm độ chêch (bias) của mô hình. Nếu mô hình cơ sở vốn đã chêch, thì việc lấy trung bình nhiều mô hình chêch cũng sẽ chỉ cho ra một kết quả chêch.

Điều này đặt ra một câu hỏi mới: Liệu có cách nào xây dựng một mô hình mà có thể chủ động *giảm cả bias* không? Câu trả lời nằm ở họ thuật toán Boosting.

5. Chương 3: AdaBoost - Học Từ Sai Lầm

5.1 Ý Tưởng Cải Tiến

Thay vì xây dựng các cây một cách độc lập như Random Forest, họ thuật toán Boosting tiếp cận theo hướng khác: xây dựng cây một cách **tuần tự**, trong đó *mô hình sau sẽ học cách khắc phục sai lầm của toàn bộ các mô hình trước đó*. AdaBoost (Adaptive Boosting) là một trong những thuật toán tiên phong cho ý tưởng này.

5.2 Phương Pháp

AdaBoost tập trung vào những mẫu dữ liệu bị phân loại sai.

1. **Khởi tạo:** Ban đầu, tất cả các mẫu dữ liệu có **trọng số (sample weight)** bằng nhau (ví dụ: $1/N$).
2. **Lặp:**

- (a) Huấn luyện một mô hình yếu (thường là một cây quyết định rất nông, gọi là **stump**) trên dữ liệu có trọng số.
- (b) Đánh giá mô hình. Tính toán tổng lỗi có trọng số (ϵ) và xác định "tiếng nói" hay tầm quan trọng (α) của mô hình này. Mô hình nào ít lỗi hơn sẽ có tiếng nói lớn hơn.

$$\alpha = \frac{1}{2} \ln \left(\frac{1 - \epsilon}{\epsilon} \right)$$

- (c) **Cập nhật trọng số:** Tăng trọng số của các mẫu bị phân loại sai và giảm trọng số của các mẫu đúng. Điều này buộc mô hình tiếp theo phải tập trung hơn vào các mẫu "khó".

3. **Dự đoán cuối cùng:** Là một cuộc bỏ phiếu có trọng số của tất cả các mô hình yếu, với trọng số là "tầm quan trọng" (α) của chúng.

5.3 Áp dụng vào ví dụ

- **Vòng 1:** Tất cả 8 bệnh nhân có trọng số 1/8. Giả sử stump đầu tiên chia theo MaxHeartRate ≤ 156 và phân loại sai bệnh nhân số 8 (dự đoán là 0 trong khi thực tế là 1).
- **Vòng 2:** Trọng số của bệnh nhân số 8 sẽ được tăng lên đáng kể, trong khi trọng số của 7 người kia giảm xuống. Stump tiếp theo sẽ phải nỗ lực nhiều hơn để phân loại đúng cho bệnh nhân số 8, có thể bằng cách tìm ra một quy luật khác, ví dụ như Age > 67 ?

5.4 Ưu và Nhược Điểm

- **Điểm cải tiến:** Đây là một cách tiếp cận hoàn toàn khác, tập trung vào việc **giảm độ chêch (bias)** bằng cách tuần tự sửa lỗi. Nó biến một tập hợp các mô hình yếu thành một mô hình mạnh duy nhất.
- **Ưu điểm:**
 - Thường cho độ chính xác cao và dễ triển khai.
 - Linh hoạt với nhiều loại mô hình yếu khác nhau.
- **Nhược điểm:**
 - **Nhạy cảm với nhiễu và outliers:** Nếu có một mẫu bị gán nhãn sai, AdaBoost sẽ cố gắng "vết cạn" để học cho đúng mẫu đó, có thể làm hỏng mô hình tổng thể.
 - Huấn luyện tuần tự nên không thể song song hóa, làm chậm quá trình huấn luyện.

6. Chương 4: Gradient Boosting - Tổng Quát Hóa Ý Tưởng Sửa Lỗi

6.1 Ý Tưởng Cải Tiết

AdaBoost sửa lỗi bằng cách thay đổi trọng số dữ liệu. Gradient Boosting (GB) đưa ý tưởng này lên một tầm cao mới bằng cách tiếp cận nó như một bài toán tối ưu hóa: thay vì tập trung vào các mẫu bị phân loại sai, GB tập trung trực tiếp vào **sai số (error)** của mô hình.

6.2 Phương Pháp

Ý tưởng cốt lõi là mỗi cây mới sẽ được huấn luyện để dự đoán **phần dư (residual)** – tức là sự khác biệt giữa giá trị thực và giá trị dự đoán của toàn bộ các cây trước đó. Sai số này được gọi là **phần dư giả (pseudo-residual)**.

1. **Khởi tạo:** Bắt đầu với một dự đoán ban đầu đơn giản (F_0), ví dụ: giá trị trung bình của nhãn trong bài toán hồi quy, hoặc log-odds trong bài toán phân loại.
2. **Lặp (từ m = 1 đến M):**
 - (a) Tính toán phần dư giả cho mỗi mẫu (r_{im}): Phần dư này chính là **gradient âm của hàm mất mát** đối với dự đoán của vòng trước.

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$$

- (b) Xây dựng một cây quyết định mới (f_m) để học cách dự đoán các phần dư này, thay vì dự đoán nhãn gốc.
- (c) Cập nhật dự đoán tổng thể: Cộng dự đoán của cây mới này vào dự đoán tổng thể của vòng trước, nhưng có điều chỉnh bởi một **hệ số học (learning rate, η)**.

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

Learning rate (η) đóng vai trò điều chỉnh, buộc mô hình phải học một cách từ từ và thận trọng, giúp chống overfitting hiệu quả. Quá trình này lặp lại, với mỗi cây mới sẽ cố gắng sửa những sai sót còn lại của toàn bộ chuỗi cây trước đó.

6.3 Ưu và Nhược Điểm

- **Điểm cải tiến:**

- **Tổng quát hóa:** Bằng cách tối ưu hóa trực tiếp một hàm mất mát bất kỳ, GB có thể áp dụng cho vô số bài toán (hồi quy, phân loại, xếp hạng) với các thước đo lỗi khác nhau.

- **Ưu điểm:**

- Thường cho độ chính xác rất cao, là một trong những thuật toán "sẵn dùng" tốt nhất.

- **Nhược điểm:**

- Tinh chỉnh siêu tham số (số cây, độ sâu, learning rate) có thể phức tạp và tốn thời gian.
 - Tốc độ huấn luyện có thể chậm do tính tuần tự.

7. Chương 5: XGBoost - Đỉnh Cao Của Tối Ưu Hóa

7.1 Ý Tưởng Cải Tiến

XGBoost (eXtreme Gradient Boosting) không phải là một thuật toán hoàn toàn mới, mà là một phiên bản **tối ưu hóa đến cực đoan** của Gradient Boosting, cải tiến cả về công thức toán học và hiệu năng hệ thống.

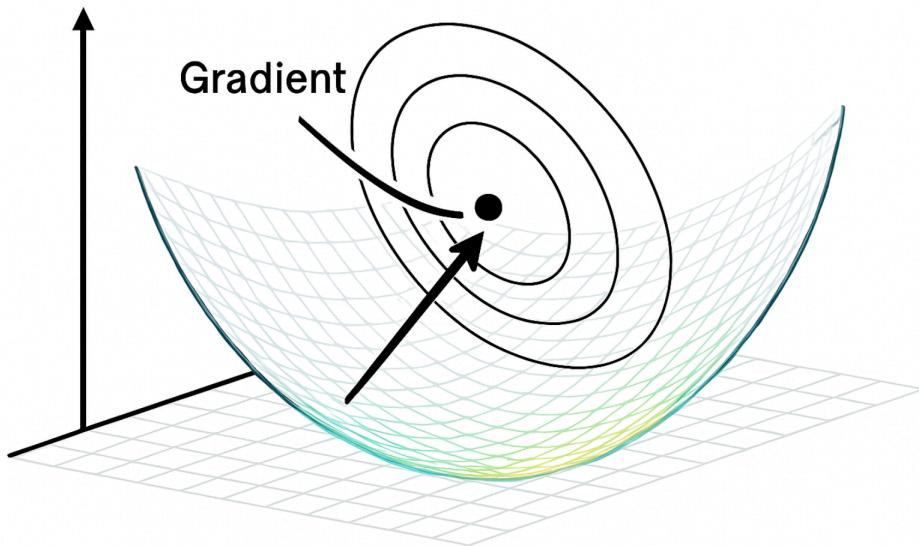
7.2 Phương Pháp và Phân Tích Chuyên Sâu

XGBoost kế thừa ý tưởng học trên phần dư của GB nhưng tinh vi hơn rất nhiều.

- **Xấp xỉ Taylor Bậc Hai:** Thay vì chỉ dùng Gradient (đạo hàm bậc nhất) như GB truyền thống, XGBoost sử dụng cả **Gradient (g_i)** và **Hessian (h_i , đạo hàm bậc hai)** trong hàm mục tiêu.

- *Trực quan:* Gradient cho biết *hướng dốc* của hàm lỗi, còn Hessian cho biết *độ cong*. Việc biết cả độ cong giúp XGBoost tìm đến điểm lỗi tối thiểu một cách nhanh và chính xác hơn, giống như một hệ thống định vị thông minh thay vì chỉ đi theo la bàn.

Gradient and Hessian in XGBoost



Hình 8: Sử dụng cả Gradient và Hessian trong XGBoost.

- **Hàm Mục Tiêu Có Điều Chuẩn (Regularized Objective Function):** Đây là cải tiến quan trọng nhất, giúp XGBoost chống overfitting một cách tự nhiên. Mục tiêu của XGBoost không chỉ là giảm thiểu hàm mất mát, mà là giảm thiểu:

$$Obj^{(t)} = \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Trong đó, $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2$ là thành phần "phạt" cho độ phức tạp.

- γ (gamma): Phạt cho việc thêm một lá mới, hoạt động như một ngưỡng cắt tỉa. Một nhánh chỉ được tách nếu lợi ích giảm lỗi lớn hơn chi phí γ .
- λ (lambda): Phạt điều chỉnh L2 lên trọng số của các nút lá (w_j). Nó làm các giá trị dự đoán của mỗi cây trở nên "dè dặt" hơn, tránh việc quá phụ thuộc vào một cây nào.
- **Xử lý Dữ liệu Thiếu (Sparsity-aware):** XGBoost không yêu cầu tiền xử lý giá trị thiếu. Thay vào đó, nó tự học hướng đi mặc định (trái hay phải) cho các giá trị thiếu tại mỗi nút để tối đa hóa lợi ích.
- **Tối ưu Hệ thống:** Tận dụng triệt để phần cứng thông qua các kỹ thuật như song song hóa quá trình tìm điểm chia, sắp xếp dữ liệu theo khối để tối ưu cache, và tính toán ngoài bộ nhớ (out-of-core) cho các tập dữ liệu khổng lồ.

7.3 Ưu và Nhược Điểm

- **Điểm cải tiến:** Là một gói hoàn chỉnh kết hợp độ chính xác thuật toán (Taylor bậc hai, điều chuẩn) và tốc độ hệ thống (song song hóa, tối ưu cache).
- **Ưu điểm:**

- **Hiệu suất đỉnh cao:** Thường xuyên là thuật toán chiến thắng trong các cuộc thi khoa học dữ liệu trên Kaggle.
- **Tốc độ và khả năng mở rộng:** Nhanh hơn đáng kể so với GB truyền thống.
- **Tính năng tích hợp:** Tự xử lý giá trị thiếu, điều chỉnh tích hợp sẵn.
- **Nhược điểm:**
 - Số lượng siêu tham số cần tinh chỉnh khá lớn, đòi hỏi người dùng có kiến thức sâu để khai thác tối đa hiệu năng.

8. Chương 6: Vượt Ra Ngoài XGBoost - Xu Hướng Hiện Tại

Sự tiến hóa không dừng lại ở XGBoost. Dựa trên những nguyên tắc của nó, các thư viện mới đã ra đời với những cải tiến riêng:

- **LightGBM (Light Gradient Boosting Machine):** Phát triển bởi Microsoft, thường nhanh hơn XGBoost. Thay vì phát triển cây theo từng tầng (level-wise), LightGBM phát triển theo từng lá (leaf-wise), giúp hội tụ nhanh hơn. Nó cũng sử dụng các kỹ thuật lấy mẫu thông minh (Gradient-based One-Side Sampling - GOSS) để tập trung vào các mẫu có gradient lớn, giúp tăng tốc độ mà không ảnh hưởng nhiều đến độ chính xác.
- **CatBoost (Categorical Boosting):** Phát triển bởi Yandex, cực kỳ mạnh mẽ trong việc xử lý **đặc trưng dạng chuỗi (categorical features)** một cách tự động và hiệu quả. Nó sử dụng một phương pháp hoán vị thông minh (ordered boosting) để mã hóa các đặc trưng này mà không gây ra hiện tượng rò rỉ mục tiêu (target leakage), một vấn đề phổ biến mà các thuật toán khác thường gặp phải.

9. Kết Luận

Hành trình từ một Cây Quyết Định đơn giản đến các hệ thống Boosting phức tạp như XGBoost là một minh chứng tuyệt vời cho sự phát triển của ngành học máy. Chúng ta đã đi từ một mô hình dễ diễn giải nhưng yếu và không ổn định, đến một "khu rừng" vững chắc có khả năng giảm phương sai (Random Forest), rồi đến một chuỗi các mô hình liên tục sửa sai để giảm độ chênh (Boosting). Cuối cùng, XGBoost và các hậu duệ của nó đã kết hợp những gì tốt nhất của các ý tưởng này và tối ưu chúng đến cực hạn, tạo ra những công cụ mạnh mẽ, linh hoạt và hiệu quả bậc nhất cho các bài toán trên dữ liệu dạng bảng ngày nay.

Tài liệu

- [1] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*, 2022.

LightGBM: Từ Cơ Bản đến Nâng Cao với SHAP

Đàm Nguyên Khánh

Tóm tắt nội dung

LightGBM (Light Gradient Boosting Machine) đại diện cho một bước tiến đột phá trong lĩnh vực Gradient Boosting, được phát triển bởi Microsoft để giải quyết những thách thức về tốc độ và hiệu quả bộ nhớ của các thuật toán truyền thống. Bài viết này trình bày một cách toàn diện về LightGBM, từ các khái niệm cơ bản đến các kỹ thuật tối ưu hóa tiên tiến, kết hợp với SHAP (SHapley Additive exPlanations) để giải thích mô hình.

Các kỹ thuật cốt lõi được phân tích chi tiết bao gồm: (1) **Leaf-wise Growth** - phát triển cây theo hướng lá thay vì level-wise, giúp giảm overfitting và tăng tốc huấn luyện; (2) **Histogram-based Split** - sử dụng histogram để tối ưu hóa việc tìm điểm chia, giảm đáng kể số phép tính từ hàng triệu xuống còn hàng trăm; (3) **Exclusive Feature Bundling (EFB)** - gom các đặc trưng categorical hiếm khi xuất hiện cùng nhau, tiết kiệm 60-80% bộ nhớ; (4) **Gradient-based One-Side Sampling (GOSS)** - lấy mẫu thông minh dựa trên gradient, giảm 80% thời gian huấn luyện mà vẫn giữ được thông tin quan trọng.

Kết quả thực nghiệm trên hai dataset lớn cho thấy hiệu quả vượt trội của LightGBM: (1) **NYC Taxi Trip Duration** (1.4M mẫu) - LightGBM đạt RMSE 0.3473 trong 3.67s, nhanh hơn GradientBoosting 200x; (2) **HEPMASS Classification** (3.5M mẫu) - đạt Accuracy 0.9179 và AUC-ROC 0.9711 trong 21.55s, cạnh tranh trực tiếp với XGBoost. **SHAP analysis** tiết lộ tác động phi tuyến của các đặc trưng, đặc biệt là tuổi trung niên (30-60) có ảnh hưởng mạnh nhất đến dự đoán mô hình.

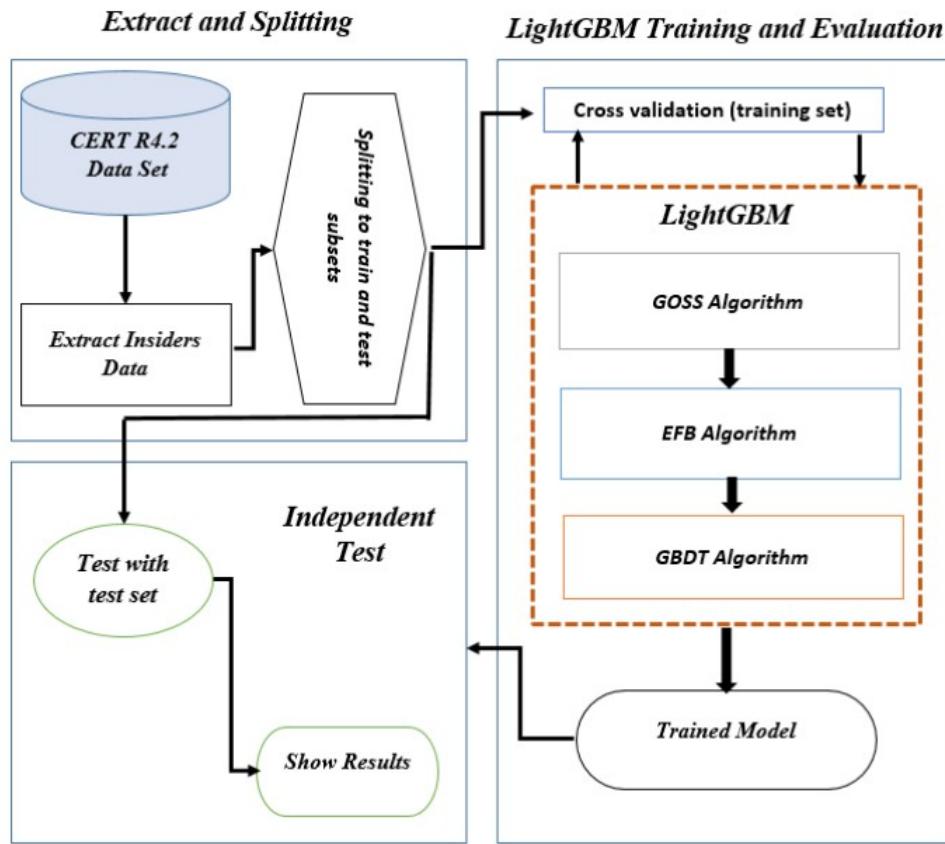
Bài viết cung cấp hướng dẫn thực hành chi tiết với code Python, so sánh hiệu suất với XGBoost và GradientBoosting, và minh họa cách sử dụng SHAP để giải thích các quyết định của mô hình. Kết quả cho thấy LightGBM là lựa chọn tối ưu cho các bài toán tabular data lớn, đặc biệt khi có nhiều đặc trưng categorical và yêu cầu tốc độ xử lý cao.

1. Giới thiệu

LightGBM (Light Gradient Boosting Machine) là một thư viện mạnh mẽ được phát triển bởi Microsoft, giúp huấn luyện mô hình Gradient Boosting nhanh hơn và tiết kiệm bộ nhớ hơn so với các phương pháp truyền thống.

Trong blog này, chúng ta sẽ:

- **Ôn lại các mô hình nền tảng:** Decision Tree, Random Forest, AdaBoost, Gradient Boosting, XGBoost
- **Hiểu rõ cải tiến cốt lõi của LightGBM:** Leaf-wise Growth, Histogram-based Split, GOSS, EFB
- **Trình bày qua các ví dụ chi tiết:** HEPMASS dataset (phân loại) và NYC Taxi Trip dataset (hồi quy)
- **Minh họa kết quả:** Sử dụng TikZ và gọi ý thêm hình ảnh thực tế
- **Giải thích mô hình:** Sử dụng SHAP để hiểu rõ đóng góp của từng đặc trưng



Hình 9: Quy trình tổng thể huấn luyện và đánh giá mô hình LightGBM: Bên trái (Extract and Splitting) - trích xuất dữ liệu từ CERT R4.2 Data Set và chia thành tập huấn luyện/kiểm thử. Bên phải (LightGBM Training and Evaluation) - huấn luyện với Cross Validation sử dụng 3 thuật toán chính: GOSS (chọn mẫu theo gradient), EFB (gom đặc trưng), GBDT (cây tăng cường), sau đó đánh giá trên tập test độc lập.

Chuẩn bị môi trường

Các thư viện cần thiết: pandas, numpy, scikit-learn, xgboost, lightgbm, shap.

Cài đặt nhanh:

```
pip install pandas numpy scikit-learn xgboost lightgbm shap
```

2. Ôn tập nhanh: Các mô hình trước LightGBM

Trước khi đi sâu vào LightGBM, hãy ôn lại các mô hình nền tảng trong họ Boosting:

Mô hình	Hàm loss Regression	Hàm loss Classification	Cách phát triển cây	Điểm mới chính
Decision Tree	MSE	Gini/Entropy	Level-wise (top-down)	Chia theo split tốt nhất
Random Forest	MSE	Gini/Entropy	Level-wise + Bagging	Giảm variance qua sampling
AdaBoost	MSE	Exponential Loss	Sequential boosting	Tăng trọng số điểm sai
Gradient Boosting	Any diff. loss	Log-loss	Sequential boosting	Boosting theo gradient
XGBoost	Any diff. loss + Reg	Log-loss Reg	+ Level-wise Reg.	Taylor bậc 2, regularization
LightGBM	Any diff. loss + Reg	Log-loss Reg	+ Leaf-wise Histogram	Tăng tốc, giảm bộ nhớ, GOSS, EFB

Bảng 2: So sánh các mô hình trong họ Boosting. LightGBM giữ nền tảng boosting nhưng tối ưu tốc độ và hiệu quả bộ nhớ.

Ý tưởng cốt lõi: LightGBM giữ nền tảng boosting nhưng tối ưu tốc độ và hiệu quả bộ nhớ thông qua các kỹ thuật đột phá.

3. Cách LightGBM xây dựng cây

LightGBM sử dụng một cách tiếp cận hoàn toàn khác biệt để xây dựng cây quyết định so với các thuật toán truyền thống. Thay vì mở rộng cây theo từng tầng (level-wise), LightGBM chọn chiến lược **leaf-wise growth** - một phương pháp thông minh hơn, hiệu quả hơn.

3.1 Level-wise vs. Leaf-wise: Sự khác biệt cốt lõi

3.1.1 Cách tiếp cận truyền thống (Level-wise)

Hầu hết các thuật toán cây quyết định (bao gồm XGBoost, Gradient Boosting) sử dụng chiến lược **level-wise**:

- Mở rộng **tất cả các nút** ở cùng một mức độ sâu
- Tạo ra cây **cân bằng** (balanced tree)
- Đảm bảo mọi nhánh đều có cùng độ sâu

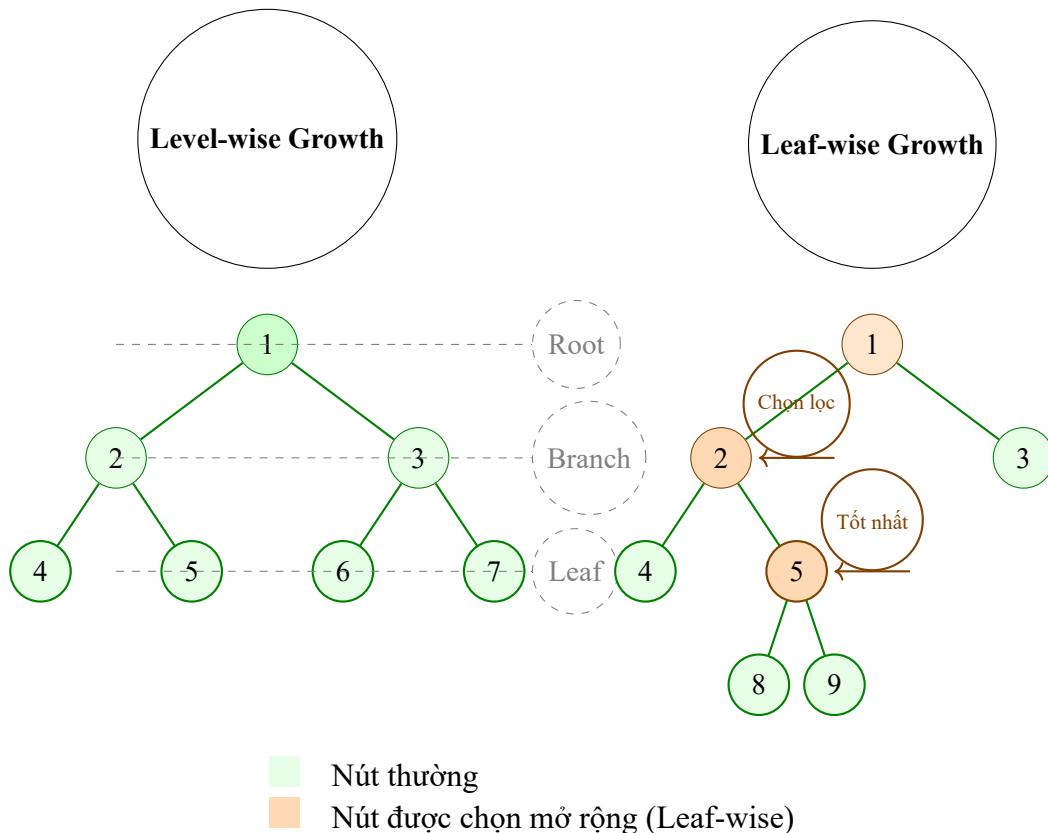
Ví dụ minh họa: Giả sử ta có một cây với độ sâu 3:

1. **Tầng 1:** Tạo 2 nút con từ nút gốc
2. **Tầng 2:** Tạo 4 nút con từ 2 nút tầng 1
3. **Tầng 3:** Tạo 8 nút con từ 4 nút tầng 2

3.1.2 Cách tiếp cận LightGBM (Leaf-wise)

LightGBM sử dụng chiến lược **leaf-wise**:

- Chỉ mở rộng **một lá** có tiềm năng giảm loss lớn nhất
- Tạo ra cây **không cân bằng** (unbalanced tree)
- Tập trung vào những nhánh quan trọng nhất



Hình 10: Level-wise mở rộng tất cả nút lá cùng level, Leaf-wise chỉ mở rộng nút lá có tiềm năng nhất (đỏ).

3.1.3 Tại sao Leaf-wise hiệu quả hơn?

1. Tập trung vào những gì quan trọng:

- Không phải mọi nhánh đều có tiềm năng cải thiện mô hình như nhau
- Một số nhánh có thể giảm loss đáng kể, số khác thì không
- Leaf-wise chỉ đầu tư vào những nhánh "có triển vọng"

2. Giảm overfitting:

- Cây không cân bằng tự nhiên có xu hướng ít phức tạp hơn
- Tránh việc tạo ra quá nhiều nút không cần thiết
- Mô hình tổng quát hóa tốt hơn trên dữ liệu mới

3. Tăng tốc huấn luyện:

- Ít phép tính hơn vì không cần mở rộng tất cả nhánh
- Tập trung tài nguyên vào những nút quan trọng
- Đạt được kết quả tốt với ít cây hơn

3.1.4 Thuật toán Leaf-wise Growth

Quá trình xây dựng cây trong LightGBM diễn ra như sau:

Algorithm 1 Leaf-wise Tree Building

Require: Dữ liệu huấn luyện D , số lá tối đa K

```

1: Khởi tạo cây với một nút gốc chứa toàn bộ dữ liệu
2:  $leaves = \{root\}$                                 ▷ Tập các lá hiện tại
3: for  $i = 1$  đến  $K - 1$  do
4:    $best\_leaf = \arg \max_{l \in leaves} Gain(l)$ 
5:    $best\_split = \text{FindBestSplit}(best\_leaf)$ 
6:   if  $Gain(best\_split) > threshold$  then
7:     Tạo 2 nút con từ  $best\_leaf$ 
8:     Cập nhật  $leaves$ : loại bỏ  $best\_leaf$ , thêm 2 nút con
9:   else
10:    break                                     ▷ Dừng nếu không có split tốt
11:   end if
12: end for

```

Giải thích từng bước:

1. **Bước 1:** Bắt đầu với một nút gốc chứa toàn bộ dữ liệu
2. **Bước 2:** Tính toán *gain* (lợi ích) cho mỗi lá hiện tại
3. **Bước 3:** Chọn lá có gain lớn nhất để chia
4. **Bước 4:** Tìm cách chia tốt nhất cho lá đó
5. **Bước 5:** Nếu gain đủ lớn, thực hiện chia và cập nhật danh sách lá
6. **Bước 6:** Lặp lại cho đến khi đạt số lá tối đa hoặc không còn split tốt

3.2 Histogram-based Split: Tối ưu hóa việc tìm điểm chia

Một trong những thách thức lớn nhất khi xây dựng cây quyết định là tìm **điểm chia tốt nhất** (best split point) cho mỗi đặc trưng. Cách tiếp cận truyền thống phải kiểm tra tất cả các giá trị có thể, điều này rất tốn kém về mặt tính toán.

3.2.1 Vấn đề với cách tiếp cận truyền thống

Thuật toán thông thường:

1. Sắp xếp tất cả giá trị của đặc trưng
2. Kiểm tra từng điểm chia có thể

3. Tính toán gain cho mỗi điểm chia

4. Chọn điểm chia có gain lớn nhất

Ví dụ: Với đặc trưng Petal_Width có 6 giá trị: {0.2, 0.5, 0.6, 0.7, 0.9, 1.3}

- Số điểm chia cần kiểm tra: 5 (giữa các giá trị liên tiếp)
- Với 1000 đặc trưng, mỗi đặc trưng có 100 giá trị: $1000 \times 99 = 99,000$ phép tính
- Với 1 triệu mẫu: $1,000,000 \times 99,000 = 99$ tỷ phép tính!

3.2.2 Giải pháp Histogram-based của LightGBM

LightGBM giải quyết vấn đề này bằng cách **gom nhóm** (binning) các giá trị liên tục thành các **bin** rời rạc:

Bin	Khoảng giá trị	Các giá trị thực tế
1	[0.2–0.567]	{0.2, 0.5}
2	(0.567–0.933]	{0.6, 0.7, 0.9}
3	(0.933–1.3]	{1.3}

Bảng 3: Ví dụ Histogram-based Split: Thay vì kiểm tra 5 điểm chia, chỉ cần kiểm tra 2 điểm chia giữa các bin.

3.2.3 Thuật toán tạo Histogram

Algorithm 2 Histogram Construction

Require: Đặc trưng X với n giá trị, số bin B

- 1: Sắp xếp các giá trị: $x_1 \leq x_2 \leq \dots \leq x_n$
 - 2: Tính khoảng cách bin: $\Delta = \frac{x_{max} - x_{min}}{B}$
 - 3: **for** $i = 1$ đến n **do**
 - 4: $bin_id = \lfloor \frac{x_i - x_{min}}{\Delta} \rfloor$
 - 5: $histogram[bin_id] += 1$
 - 6: **end for**
 - 7: **return** $histogram$
-

3.2.4 Tìm điểm chia tốt nhất trên Histogram

Thay vì kiểm tra tất cả điểm chia, LightGBM chỉ kiểm tra các điểm chia **giữa các bin**:

1. **Bước 1:** Tạo histogram cho đặc trưng
2. **Bước 2:** Chỉ xem xét các điểm chia giữa bin i và bin $i + 1$
3. **Bước 3:** Tính gain cho mỗi điểm chia này
4. **Bước 4:** Chọn điểm chia có gain lớn nhất

Ví dụ cụ thể: Với 3 bin, chỉ cần kiểm tra 2 điểm chia:

- Điểm chia 1: Giữa bin 1 và bin 2

- Điểm chia 2: Giữa bin 2 và bin 3

3.2.5 Lợi ích của Histogram-based Split

1. Giảm đáng kể số phép tính:

- Thay vì kiểm tra $n - 1$ điểm chia, chỉ kiểm tra $B - 1$ điểm chia
- Với $B = 255$ (mặc định), giảm từ hàng triệu xuống còn 254 phép tính
- Tốc độ tăng lên hàng nghìn lần

2. Tiết kiệm bộ nhớ:

- Chỉ cần lưu trữ histogram (mảng 255 phần tử)
- Không cần lưu trữ toàn bộ dữ liệu gốc
- Có thể xử lý dataset rất lớn

3. Tăng tốc song song:

- Mỗi đặc trưng có thể xử lý độc lập
- Dễ dàng song song hóa trên nhiều CPU
- Tận dụng tối đa tài nguyên máy tính

3.2.6 Công thức toán học

Gain của một điểm chia được tính như sau:

$$\text{Gain} = \frac{1}{2} \left[\frac{(\sum_{i \in L} g_i)^2}{\sum_{i \in L} h_i + \lambda} + \frac{(\sum_{i \in R} g_i)^2}{\sum_{i \in R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right]$$

Trong đó:

- g_i : gradient của mẫu thứ i
- h_i : hessian của mẫu thứ i
- L, R : tập mẫu bên trái và bên phải điểm chia
- I : tập mẫu gốc
- λ : tham số regularization

Với histogram, ta có thể tính nhanh:

$$\begin{aligned} \sum_{i \in L} g_i &= \sum_{b=1}^{\text{split_bin}} \text{histogram}[b].\text{sum_gradients} \\ \sum_{i \in L} h_i &= \sum_{b=1}^{\text{split_bin}} \text{histogram}[b].\text{sum_hessians} \end{aligned}$$

3.3 Exclusive Feature Bundling (EFB): Tối ưu hóa đặc trưng Categorical

Một thách thức lớn trong học máy là xử lý **đặc trưng categorical** (phân loại). Khi có nhiều đặc trưng categorical, việc one-hot encoding có thể tạo ra hàng nghìn cột, gây ra vấn đề về bộ nhớ và tốc độ tính toán.

3.3.1 Vấn đề với One-Hot Encoding truyền thống

Ví dụ thực tế: Dataset về khách hàng có các đặc trưng:

- Country: 200 quốc gia → 200 cột one-hot
- City: 1000 thành phố → 1000 cột one-hot
- Category: 50 danh mục → 50 cột one-hot
- Color: 10 màu sắc → 10 cột one-hot

Tổng cộng: 1260 cột chỉ từ 4 đặc trưng gốc!

Vấn đề phát sinh:

- **Bộ nhớ:** Cần lưu trữ ma trận $1M \times 1260 = 1.26$ tỷ số
- **Tốc độ:** Mỗi lần tính toán phải xử lý 1260 cột
- **Sparse data:** Hầu hết các cột có giá trị 0

3.3.2 Ý tưởng cốt lõi của EFB

LightGBM nhận ra rằng **không phải tất cả đặc trưng categorical đều cần thiết cùng lúc**. Một số đặc trưng có tính chất **exclusive** (loại trừ lẫn nhau):

- **Country** và **City**: Một khách hàng chỉ ở một quốc gia và một thành phố
- **Color** và **Size**: Một sản phẩm chỉ có một màu và một kích thước
- **Day_of_week** và **Month**: Một ngày chỉ thuộc một thứ và một tháng

3.3.3 Thuật toán EFB

Algorithm 3 Exclusive Feature Bundling

Require: Tập đặc trưng categorical $F = \{f_1, f_2, \dots, f_n\}$

- 1: Tạo đồ thị G với các nút là đặc trưng
 - 2: **for** mỗi cặp đặc trưng (f_i, f_j) **do**
 - 3: **if** f_i và f_j không bao giờ cùng $\neq 0$ **then**
 - 4: Thêm cạnh (f_i, f_j) vào G
 - 5: **end if**
 - 6: **end for**
 - 7: Tìm các thành phần liên thông của G
 - 8: **for** mỗi thành phần liên thông C **do**
 - 9: Tạo bundle B chứa tất cả đặc trưng trong C
 - 10: Gán mã số duy nhất cho mỗi tổ hợp giá trị trong B
 - 11: **end for**
-

3.3.4 Ví dụ minh họa chi tiết

Xét dataset về sản phẩm với 3 đặc trưng màu sắc:

Product ID	Red	Blue	Green	→	Color_bundle
P1	1	0	0	→	0
P2	0	1	0	→	1
P3	0	0	1	→	2
P4	1	0	0	→	0
P5	0	1	0	→	1

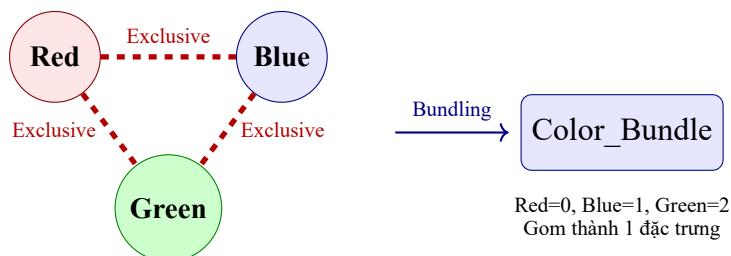
Bảng 4: EFB: Gom 3 cột one-hot thành 1 cột bundle, tiết kiệm 67% bộ nhớ.

3.3.5 Quá trình tạo Bundle

Bước 1: Phân tích tính exclusive

- Red = 1 → Blue = 0, Green = 0
- Blue = 1 → Red = 0, Green = 0
- Green = 1 → Red = 0, Blue = 0

Bước 2: Tạo đồ thị



Các đặc trưng loại trừ lẫn nhau

Hình 11: Quy trình Exclusive Feature Bundling (EFB): Các đặc trưng màu sắc (Red, Blue, Green) loại trừ lẫn nhau được gom thành một bundle duy nhất với mã số tương ứng (0, 1, 2), giảm từ 3 cột xuống 1 cột.

Bước 3: Tạo bundle

- Tất cả 3 đặc trưng thuộc cùng một thành phần liên thông
- Gom thành 1 bundle: Color_bundle
- Gán mã: Red=0, Blue=1, Green=2

3.3.6 Lợi ích của EFB

1. Tiết kiệm bộ nhớ đáng kể:

- Từ 3 cột → 1 cột (giảm 67%)
- Với 1000 đặc trưng categorical: có thể giảm từ 1000 → 200-300 cột
- Tiết kiệm hàng GB bộ nhớ với dataset lớn

2. Tăng tốc tính toán:

- Ít cột hơn → ít phép tính hơn
- Tận dụng cache hiệu quả hơn
- Giảm thời gian truy cập bộ nhớ

3. Giữ nguyên thông tin:

- Không mất thông tin gì so với one-hot encoding
- Có thể khôi phục lại đặc trưng gốc
- Kết quả mô hình không thay đổi

3.3.7 Ứng dụng thực tế

Ví dụ 1: E-commerce

- Product_category (1000 loại) + Product_brand (500 thương hiệu)
- Có thể bundle vì mỗi sản phẩm chỉ thuộc 1 loại và 1 thương hiệu
- Giảm từ 1500 cột xuống còn 1 cột

Ví dụ 2: Time series

- Hour (24 giá trị) + Day_of_week (7 giá trị) + Month (12 giá trị)
- Có thể bundle vì mỗi thời điểm chỉ có 1 giờ, 1 thứ, 1 tháng
- Giảm từ 43 cột xuống còn 1 cột

3.3.8 Công thức tính toán Bundle

Với bundle chứa k đặc trưng, mỗi đặc trưng có n_i giá trị:

$$\text{Bundle_value} = \sum_{i=1}^k f_i \times \prod_{j=1}^{i-1} n_j$$

Ví dụ với Red(2), Blue(2), Green(2):

- Red=1, Blue=0, Green=0 → Bundle = $1 \times 1 + 0 \times 2 + 0 \times 4 = 1$
- Red=0, Blue=1, Green=0 → Bundle = $0 \times 1 + 1 \times 2 + 0 \times 4 = 2$
- Red=0, Blue=0, Green=1 → Bundle = $0 \times 1 + 0 \times 2 + 1 \times 4 = 4$

3.4 Gradient-based One-Side Sampling (GOSS): Lấy mẫu thông minh

Khi làm việc với dataset rất lớn (hàng triệu mẫu), việc sử dụng toàn bộ dữ liệu cho mỗi cây có thể rất tốn kém. Tuy nhiên, không phải tất cả mẫu đều quan trọng như nhau. LightGBM sử dụng kỹ thuật **GOSS** để lấy mẫu một cách thông minh.

3.4.1 Ý tưởng cốt lõi

Trong Gradient Boosting, mỗi cây mới được huấn luyện để sửa lỗi của các cây trước đó. Những mẫu có **gradient lớn** (lỗi lớn) là những mẫu "khó học" và quan trọng hơn. Ngược lại, những mẫu có **gradient nhỏ** (lỗi nhỏ) đã được học tốt và ít quan trọng hơn.

Quan sát quan trọng:

- Mẫu có gradient lớn → Cần được học nhiều hơn
- Mẫu có gradient nhỏ → Đã học tốt, có thể bỏ qua một phần

3.4.2 Thuật toán GOSS

Algorithm 4 Gradient-based One-Side Sampling

Require: Dataset D với n mẫu, tỉ lệ lấy mẫu a, b

- Tính gradient g_i cho mỗi mẫu i
 - Sắp xếp mẫu theo $|g_i|$ giảm dần
 - Chọn top $a \times n$ mẫu có gradient lớn nhất
 - Lấy mẫu ngẫu nhiên $b \times n$ mẫu từ phần còn lại
 - Tạo dataset mới D' từ 2 nhóm trên
 - Huấn luyện cây trên D' với trọng số điều chỉnh
-

3.4.3 Ví dụ minh họa chi tiết

Giả sử có dataset với 1000 mẫu, gradient của một số mẫu:

Mẫu ID	Gradient	Loại	Quyết định
1	0.95	Lớn	<input type="checkbox"/> Giữ (top 10%)
2	0.87	Lớn	<input type="checkbox"/> Giữ (top 10%)
3	0.82	Lớn	<input type="checkbox"/> Giữ (top 10%)
...
100	0.12	Nhỏ	? Lấy mẫu ngẫu nhiên
101	0.08	Nhỏ	? Lấy mẫu ngẫu nhiên
...
1000	0.01	Nhỏ	? Lấy mẫu ngẫu nhiên

Bảng 5: Ví dụ GOSS: Giữ tất cả mẫu có gradient lớn, lấy mẫu ngẫu nhiên từ mẫu có gradient nhỏ.

3.4.4 Quá trình lấy mẫu

Bước 1: Tính gradient Với mỗi mẫu (x_i, y_i) , tính gradient:

$$g_i = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

Bước 2: Sắp xếp theo độ lớn gradient

$$|g_1| \geq |g_2| \geq \dots \geq |g_n|$$

Bước 3: Chia thành 2 nhóm

- **Nhóm A** ($a \times n$): Mẫu có gradient lớn nhất
- **Nhóm B** (còn lại): Mẫu có gradient nhỏ hơn

Bước 4: Lấy mẫu từ nhóm B

- Lấy ngẫu nhiên $b \times n$ mẫu từ nhóm B
- Tỉ lệ lấy mẫu: $b = 0.1$ (10%)

Bước 5: Tạo dataset mới

$$D' = A \cup \text{random_sample}(B, b \times n)$$

3.4.5 Điều chỉnh trọng số

Để đảm bảo tính không thiên lệch (unbiased), GOSS điều chỉnh trọng số:

$$w_i = \begin{cases} 1 & \text{nếu } i \in A \\ \frac{1-a}{b} & \text{nếu } i \in \text{random_sample}(B) \end{cases}$$

Giải thích:

- Mẫu trong nhóm A: trọng số = 1 (giữ nguyên)
- Mẫu được lấy mẫu từ nhóm B: trọng số = $\frac{1-a}{b}$ (tăng lên)

3.4.6 Ví dụ tính toán trọng số

Với $a = 0.1$, $b = 0.1$:

- Nhóm A (100 mẫu): $w = 1$
- Nhóm B được lấy mẫu (100 mẫu): $w = \frac{1-0.1}{0.1} = 9$

Tổng trọng số hiệu dụng:

$$\text{Effective weight} = 100 \times 1 + 100 \times 9 = 1000$$

3.4.7 Lợi ích của GOSS

1. Giảm đáng kể kích thước dataset:

- Từ 1M mẫu → 200K mẫu (với $a = 0.1, b = 0.1$)
- Giảm 80% thời gian huấn luyện
- Tiết kiệm bộ nhớ đáng kể

2. Giữ được thông tin quan trọng:

- Tất cả mẫu "khó học" được giữ lại
- Mẫu "dễ học" vẫn được đại diện qua lấy mẫu
- Chất lượng mô hình không giảm đáng kể

3. Tăng tốc song song:

- Mỗi cây có thể huấn luyện trên dataset nhỏ hơn
- Dễ dàng phân tán trên nhiều máy
- Tận dụng tài nguyên hiệu quả hơn

3.4.8 So sánh với các phương pháp lấy mẫu khác

Phương pháp	Cách lấy mẫu	Ưu điểm	Nhược điểm
Random Sampling	Ngẫu nhiên đều	Đơn giản	Có thể bỏ sót mẫu quan trọng
Stratified Sampling	Theo nhóm	Cân bằng nhóm	Phức tạp, không tối ưu
GOSS	Theo gradient	Thông minh, hiệu quả	Phức tạp hơn

Bảng 6: So sánh các phương pháp lấy mẫu trong Gradient Boosting.

3.4.9 Tham số GOSS trong LightGBM

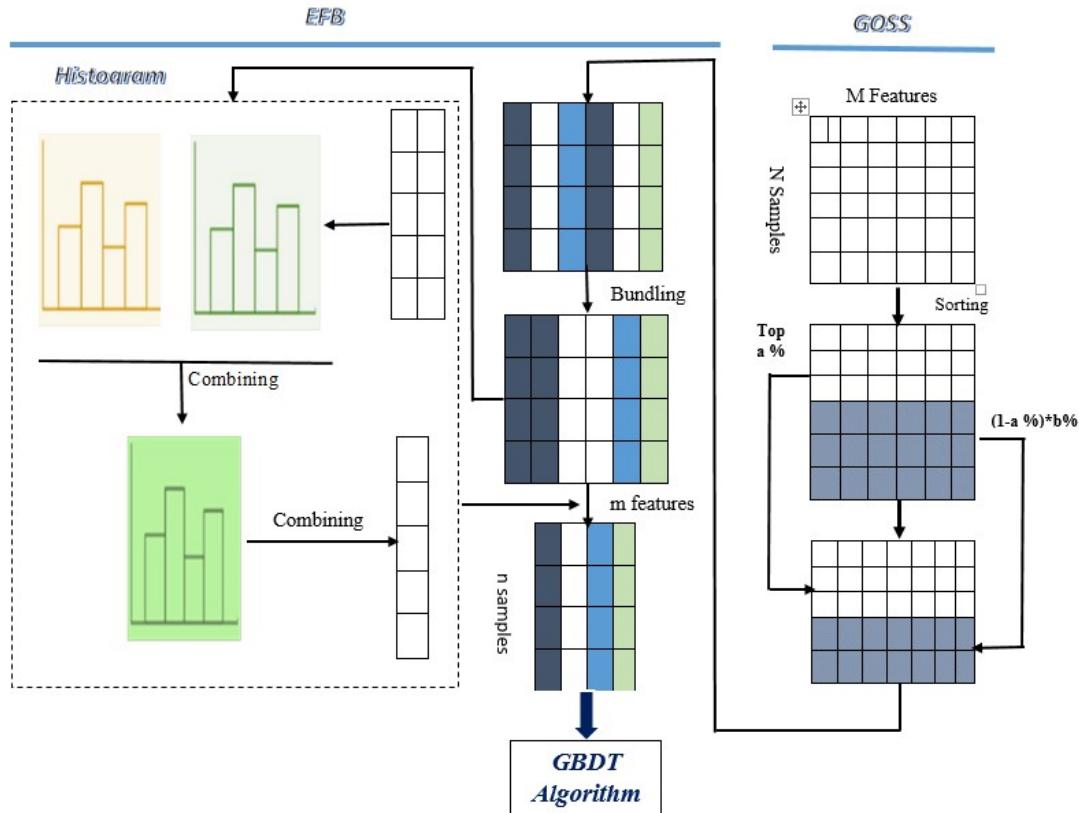
```

1 # Cấu hình GOSS trong LightGBM
2 params = {
3     'boosting_type': 'goss', # Sử dụng GOSS
4     'top_rate': 0.1,        # Tỉ lệ mẫu gradient lớn (a)
5     'other_rate': 0.1,      # Tỉ lệ lấy mẫu từ mẫu còn lại (b)
6     'verbose': -1
7 }
```

Khuyến nghị tham số:

- top_rate: 0.1-0.2 (10-20% mẫu quan trọng)
- other_rate: 0.1-0.2 (10-20% mẫu ngẫu nhiên)

- Tổng cộng: 20-40% dataset gốc



Hình 12: Các thuật toán tối ưu trong LightGBM (EFB và GOSS): EFB sử dụng histogram để mã hóa đặc trưng, gom các đặc trưng hiêm khi xuất hiện cùng nhau để giảm chiều từ M features → m features. GOSS thực hiện sorting theo gradient, lấy Top $a\%$ mẫu có gradient lớn nhất và ngẫu nhiên chọn thêm $(1-a\%)*b\%$ từ phần còn lại, giúp tăng tốc huấn luyện mà vẫn giữ thông tin quan trọng.

3.5 Tổng kết: Bốn cải tiến cốt lõi của LightGBM

LightGBM đạt được hiệu suất vượt trội nhờ sự kết hợp của bốn kỹ thuật đột phá:

Kỹ thuật	Vấn đề giải quyết	Cách hoạt động	Lợi ích
Leaf-wise Growth	Cây không cân bằng, overfitting	Mở rộng lá có gain lớn nhất	Giảm overfitting, tăng tốc
Histogram-based Split	Tìm điểm chia tôn kém	Gom giá trị thành bins	Tăng tốc hàng nghìn lần
Exclusive Feature Bundling	One-hot encoding tạo nhiều cột	Gom đặc trưng exclusive	Tiết kiệm bộ nhớ 60-80%
Gradient-based Sampling	Dataset quá lớn	Lấy mẫu theo gradient	Giảm 80% thời gian huấn luyện

Bảng 7: Tổng kết bốn cải tiến cốt lõi của LightGBM.

3.5.1 Sự kết hợp hoàn hảo

Bốn kỹ thuật này không hoạt động độc lập mà **bổ sung cho nhau**:

1. **Histogram** giúp tìm điểm chia nhanh chóng
2. **Leaf-wise** quyết định nên chia nút nào
3. **EFB** giảm số đặc trưng cần xử lý
4. **GOSS** giảm số mẫu cần xử lý

Kết quả: LightGBM có thể xử lý dataset lớn gấp 10-100 lần so với các thuật toán truyền thống mà vẫn đạt độ chính xác cao.

3.5.2 Khi nào sử dụng LightGBM

LightGBM phù hợp nhất với dataset lớn ($>100K$ mẫu), nhiều đặc trưng categorical, cần tốc độ huấn luyện nhanh và bộ nhớ hạn chế. Ngược lại, với dataset rất nhỏ ($<10K$ mẫu) hoặc chỉ có đặc trưng numerical, các thuật toán khác có thể phù hợp hơn.

Kết luận về LightGBM

LightGBM đại diện cho một bước tiến lớn trong lĩnh vực Gradient Boosting. Bằng cách kết hợp bốn kỹ thuật đột phá - Leaf-wise Growth, Histogram-based Split, Exclusive Feature Bundling, và Gradient-based One-Side Sampling - LightGBM đã giải quyết được những thách thức lớn nhất của các thuật toán truyền thống: tốc độ chậm, tiêu tốn bộ nhớ, và khó xử lý dữ liệu categorical. Đây là lý do tại sao LightGBM trở thành lựa chọn hàng đầu cho các bài toán tabular data trong thực tế.

4. Dữ liệu sử dụng

Phân loại (HEPMASS)

- **Mục tiêu:** phân biệt tín hiệu (signal) và nền (background) trong thí nghiệm vật lý năng lượng cao.
- **Đặc trưng:** 27 biến đã chuẩn hoá, không thiếu dữ liệu.
- **Định dạng:** CSV, cột đầu tiên là nhãn (0/1), sau là 27 cột đặc trưng.
- **Kích thước:** 3.5 triệu mẫu (50% signal, 50% background)

Hồi quy (NYC Taxi Trip Duration)

- **Mục tiêu:** dự đoán thời lượng chuyến đi taxi từ đặc trưng thời gian và toạ độ.¹
- **Định dạng:** CSV gồm pickup_datetime, passenger_count, toạ độ đón/trả, trip_duration (mục tiêu), v.v.
- **Kích thước:** 1.4 triệu chuyến taxi

¹Mô tả bài toán: [NYC Taxi Trip Duration \(Kaggle\)](#).

Đặt dữ liệu ở đâu?

Đặt các file CSV vào thư mục projects/LightGBM_SHAP/content/ và cập nhật đường dẫn trong mã nếu khác. Với dữ liệu lớn, nên chạy cục bộ thay vì môi trường hạn chế tài nguyên.

5. Pipeline Phân loại: HEPMASS

Notebook thực hiện: tách nhãn/đặc trưng, chia train/test theo tỉ lệ 80/20 có *stratify*, sau đó huấn luyện ba mô hình và so sánh **Accuracy** và **AUC-ROC**. Mã rút gọn minh họa:

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import accuracy_score, roc_auc_score
4 from sklearn.ensemble import GradientBoostingClassifier
5 import xgboost as xgb
6 import lightgbm as lgb
7
8 df = pd.read_csv('projects/LightGBM_SHAP/content/particles.csv')
9 y = df.iloc[:, 0].values
10 X = df.iloc[:, 1:].values
11
12 X_train, X_test, y_train, y_test = train_test_split(
13     X, y, test_size=0.2, random_state=42, stratify=y
14 )
15
16 # 1) GradientBoosting (sklearn)
17 gb = GradientBoostingClassifier(n_estimators=100, max_depth=6, learning_rate=0.1,
18                                 subsample=0.8, random_state=42)
19 gb.fit(X_train, y_train)
20
21 # 2) XGBoost (DMatrix API)
22 dtrain, dtest = xgb.DMatrix(X_train, label=y_train), xgb.DMatrix(X_test, label=y_test)
23 xgb_params = {'objective': 'binary:logistic', 'eval_metric': 'auc', 'max_depth': 6,
24                'learning_rate': 0.1, 'subsample': 0.8, 'colsample_bytree': 0.8,
25                'random_state': 42}
26 xgb_model = xgb.train(xgb_params, dtrain, num_boost_round=100)
27
28 # 3) LightGBM
29 lgb_train = lgb.Dataset(X_train, label=y_train)
30 lgb_model = lgb.train({'objective': 'binary', 'metric': 'auc', 'learning_rate': 0.1,
31                       'num_leaves': 63, 'max_depth': 6, 'feature_fraction': 0.8,
32                       'bagging_fraction': 0.8, 'bagging_freq': 5, 'random_state': 42},
33                       lgb_train, num_boost_round=100)
34
35 # Đánh giá
36 from sklearn.metrics import roc_auc_score
37 gb_auc = roc_auc_score(y_test, gb.predict_proba(X_test)[:, 1])
38 xgb_auc = roc_auc_score(y_test, xgb_model.predict(dtest))
39 lgb_auc = roc_auc_score(y_test, lgb_model.predict(X_test))
40 print({'gb_auc': gb_auc, 'xgb_auc': xgb_auc, 'lgb_auc': lgb_auc})

```

Minh họa: Level-wise vs Leaf-wise

LightGBM phát triển cây theo lá (leaf-wise), ưu tiên *giảm loss nhanh nhất* thay vì tăng đều theo mức (level-wise). Xem minh họa chi tiết ở [Hình 10](#).

6. Pipeline Hồi quy: NYC Taxi

Notebook thực hiện các bước làm sạch và *feature engineering* sau, sau đó so sánh RMSE:

- Loại bỏ cột không hữu ích (dropoff_datetime, store_and_fwd_flag).
- Lọc toạ độ trong phạm vi thành phố (bounding box NYC) và reset index.
- Trích chọn đặc trưng thời gian: hour, day_of_week, month, day_of_month.
- Tính *Haversine distance*, *Manhattan distance*; gom cụm toạ độ bằng MiniBatchKMeans.
- Biến đổi mục tiêu log_trip_duration để ổn định phương sai.

Mã rút gọn minh họa:

```

1 import pandas as pd, numpy as np
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import mean_squared_error
4 from sklearn.cluster import MiniBatchKMeans
5 import xgboost as xgb, lightgbm as lgb
6 from sklearn.ensemble import GradientBoostingRegressor
7
8 df = pd.read_csv('projects/LightGBM_SHAP/content/taxi_trip.csv')
9 df = df.drop(columns=[c for c in ['dropoff_datetime', 'store_and_fwd_flag'] if c in df])
10
11 # Giới hạn toạ độ NYC
12 lon_min, lon_max = -74.03, -73.75
13 lat_min, lat_max = 40.63, 40.85
14 df = df[(df['pickup_longitude'].between(lon_min, lon_max)) &
15         (df['dropoff_longitude'].between(lon_min, lon_max)) &
16         (df['pickup_latitude'].between(lat_min, lat_max)) &
17         (df['dropoff_latitude'].between(lat_min, lat_max))].reset_index(drop=True)
18
19 # Time features
20 df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
21 df['hour'] = df['pickup_datetime'].dt.hour
22 df['day_of_week'] = df['pickup_datetime'].dt.dayofweek
23 df['month'] = df['pickup_datetime'].dt.month
24 df['day_of_month'] = df['pickup_datetime'].dt.day
25
26 # Khoảng cách
27 def haversine(lat1, lon1, lat2, lon2):
28     R=6371
29     lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
30     dlat, dlon = lat2-lat1, lon2-lon1
31     a = np.sin(dlat/2)**2 + np.cos(lat1)*np.cos(lat2)*np.sin(dlon/2)**2
32     return 2*R*np.arcsin(np.sqrt(a))
33
34 df['distance_haversine'] = haversine(df['pickup_latitude'], df['pickup_longitude'],
35                                       df['dropoff_latitude'], df['dropoff_longitude'])
36 df['distance_manhattan'] =

```

```
37     haversine(df['pickup_latitude'], df['pickup_longitude'], df['pickup_latitude'], df['dropoff_longitude']) +  
38     haversine(df['pickup_latitude'], df['pickup_longitude'], df['dropoff_latitude'], df['pickup_longitude']))  
39 )  
40  
41 # Gom cụm vị trí  
42 kmeans = MiniBatchKMeans(n_clusters=50, random_state=42, batch_size=5000)  
43 kmeans.fit(np.vstack([  
44     df[['pickup_latitude','pickup_longitude']].values,  
45     df[['dropoff_latitude','dropoff_longitude']].values  
46 ]))  
47 df['pickup_cluster'] = kmeans.predict(df[['pickup_latitude','pickup_longitude']])  
48 df['dropoff_cluster'] = kmeans.predict(df[['dropoff_latitude','dropoff_longitude']])  
49  
50 df['log_trip_duration'] = np.log(df['trip_duration'] + 1)  
51  
52 features = ['vendor_id','passenger_count','pickup_longitude','pickup_latitude',  
53             'dropoff_longitude','dropoff_latitude','hour','day_of_week','month',  
54             'day_of_month','distance_haversine','distance_manhattan',  
55             'pickup_cluster','dropoff_cluster']  
56 X, y = df[features].copy(), df['log_trip_duration'].copy()  
57  
58 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
59  
60 # Huấn luyện nhanh 3 mô hình  
61 gb = GradientBoostingRegressor(n_estimators=100, max_depth=6, learning_rate=0.1,  
62                                 subsample=0.8, random_state=42).fit(X_train, y_train)  
63 xgb_model = xgb.train({'objective':'reg:squarederror','eval_metric':'rmse',  
64                         'max_depth':6,'learning_rate':0.1,'subsample':0.8,  
65                         'colsample_bytree':0.8,'random_state':42},  
66                         xgb.DMatrix(X_train,label=y_train), num_boost_round=100)  
67 lgb_model = lgb.train({'objective':'regression','metric':'rmse','learning_rate':0.1,  
68                         'num_leaves':63,'max_depth':6,'feature_fraction':0.8,  
69                         'bagging_fraction':0.8,'bagging_freq':5,'random_state':42},  
70                         lgb.Dataset(X_train,label=y_train), num_boost_round=100)  
71  
72 import numpy as np  
73 from sklearn.metrics import mean_squared_error  
74 rmse_gb = np.sqrt(mean_squared_error(y_test, gb.predict(X_test)))  
75 rmse_xgb = np.sqrt(mean_squared_error(y_test, xgb_model.predict(xgb.DMatrix(X_test))))  
76 rmse_lgb = np.sqrt(mean_squared_error(y_test, lgb_model.predict(X_test)))  
77 print({'rmse_gb': rmse_gb, 'rmse_xgb': rmse_xgb, 'rmse_lgb': rmse_lgb})
```

7. Case Study: So sánh hiệu suất

7.1 Taxi Trip Duration (NYC)

Dataset: 1.4 triệu chuyến taxi từ New York City.

Nhiệm vụ: Dự đoán thời gian chuyến đi dựa trên vị trí đón/trả, thời gian, số hành khách, v.v.

Model	RMSE ↓	Training time ↓
GradientBoosting	0.3457	504.61s
XGBoost	0.3483	2.98s
LightGBM	0.3473	3.67s

Bảng 8: Kết quả NYC Taxi Trip Duration. LightGBM đạt tốc độ nhanh nhất, RMSE gần tốt nhất.

Phân tích: LightGBM đạt tốc độ nhanh nhất (3.67s), RMSE gần tốt nhất (0.3473). XGBoost nhanh nhất nhưng RMSE cao hơn một chút.

7.2 HEPMASS (High Energy Physics)

Dataset: 3.5 triệu sự kiện va chạm hạt trong thí nghiệm vật lý năng lượng cao.

Nhiệm vụ: Phân loại tín hiệu (signal) vs. nhiễu (background).

Model	Accuracy ↑	AUC-ROC ↑	Training time ↓
GradientBoosting	0.9181	0.9711	4031.84s
XGBoost	0.9178	0.9712	19.77s
LightGBM	0.9179	0.9711	21.55s

Bảng 9: Kết quả HEPMASS Classification. Với dữ liệu số liệu dày đặc, XGBoost nhỉnh hơn một chút.

Phân tích: Với dữ liệu số liệu dày đặc (dense numeric), XGBoost nhỉnh hơn một chút về AUC-ROC (0.9712 vs 0.9711). Tuy nhiên, LightGBM vẫn rất cạnh tranh và nhanh hơn đáng kể so với GradientBoosting.

7.3 Tổng kết so sánh

Ưu điểm của LightGBM:

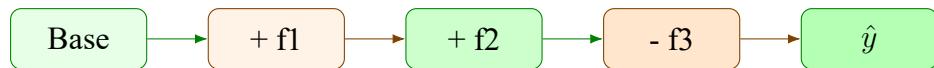
- **Tốc độ:** Nhanh hơn GradientBoosting 200x, cạnh tranh với XGBoost
- **Bộ nhớ:** Hiệu quả hơn nhờ Histogram và EFB
- **Độ chính xác:** Tương đương với các mô hình khác
- **Categorical features:** Xử lý tốt nhờ EFB

Khi nào nên dùng LightGBM:

- Dataset lớn ($>100K$ mẫu)
- Nhiều đặc trưng categorical
- Cần tốc độ huấn luyện nhanh
- Bộ nhớ hạn chế

8. Giải thích mô hình bằng SHAP

SHAP biểu diễn dự đoán \hat{y} như tổng của **base value** và **đóng góp** từ từng đặc trưng. Minh họa công thức:

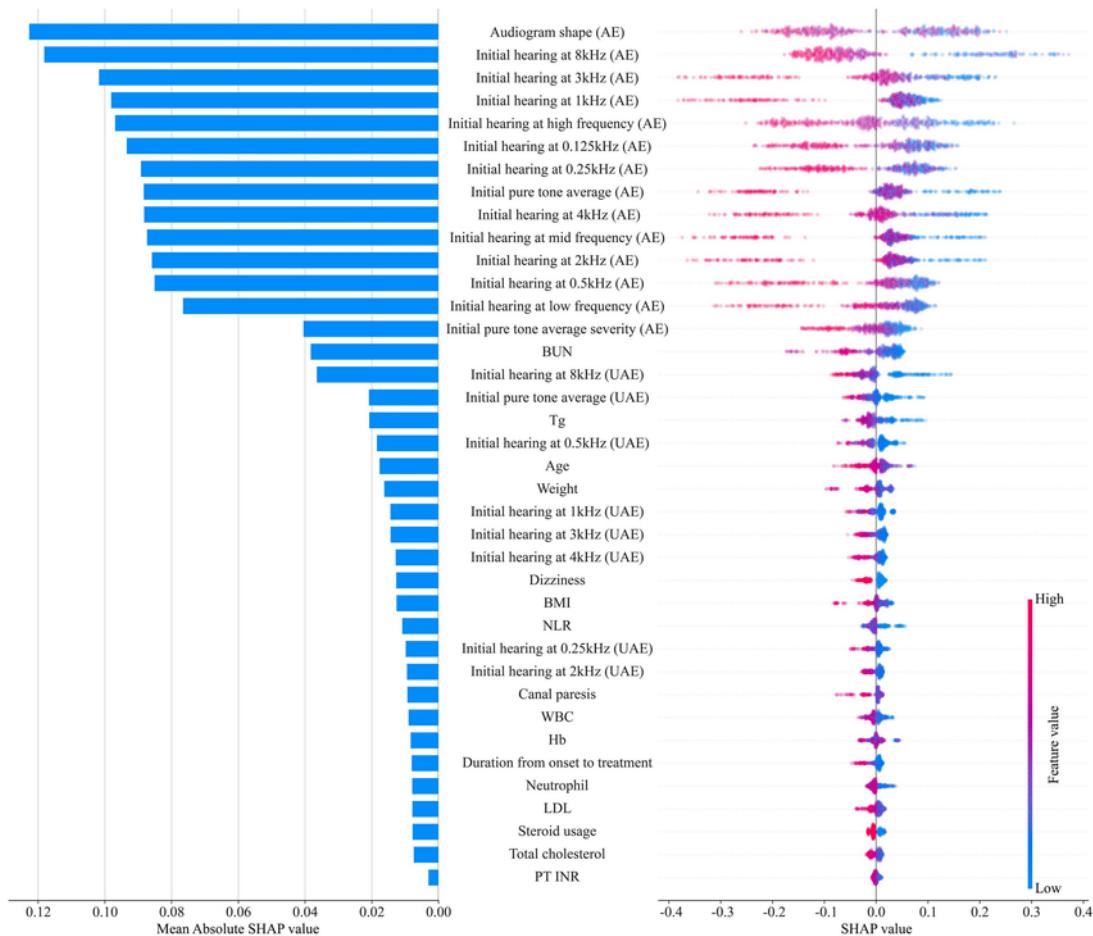


Hình 13: Tư duy cộng dồn của SHAP: $\hat{y} = \text{base} + \sum_j \phi_j$.

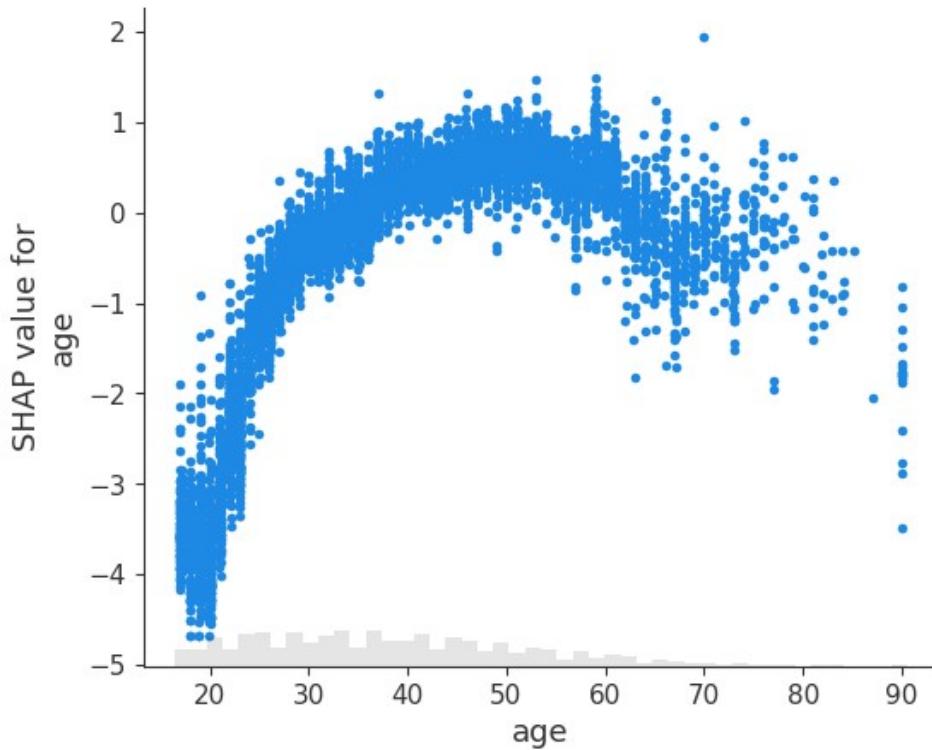
Mã minh họa tính SHAP cho XGBoost/LightGBM (nên lấy subset để hiển thị nhanh):

```

1 import shap, numpy as np
2 import xgboost as xgb, lightgbm as lgb
3
4 # Với XGBoost (binary/regression):
5 expl_xgb = shap.Explainer(xgb_model)
6 X_sample = xgb.DMatrix(X_test[:200]) # lấy 200 mẫu để vẽ
7 sv_xgb = expl_xgb(X_sample)
8 shap.plots.waterfall(sv_xgb[0])
9
10 # Với LightGBM:
11 expl_lgb = shap.Explainer(lgb_model)
12 sv_lgb = expl_lgb(X_test[:200])
13 shap.plots.beeswarm(sv_lgb)
  
```



Hình 14: Biểu đồ tầm quan trọng đặc trưng (Feature Importance với SHAP): Trái - Mean Absolute SHAP value cho các đặc trưng quan trọng nhất (thính lực: Audiogram shape, Initial hearing; sức khỏe: BUN, Tg, Age, Weight, BMI, NLR, Hb). Phải - SHAP summary plot với mỗi chấm là một mẫu, màu đỏ (High) và xanh (Low) biểu thị giá trị đặc trưng, vị trí trên trục X thể hiện mức độ tác động. Mô hình phụ thuộc nhiều vào đặc trưng thính lực và yếu tố sinh lý.



Hình 15: Biểu đồ SHAP giải thích mô hình theo độ tuổi: Trục X biểu diễn giá trị SHAP cho đặc trưng age (SHAP > 0: tuổi góp phần tăng xác suất dự đoán, SHAP < 0: giảm xác suất dự đoán). Trục Y là tuổi. Quan sát: tuổi < 25 có SHAP âm mạnh, tuổi 30-60 có SHAP tăng dần đạt đỉnh (ảnh hưởng tích cực mạnh), trên 70 tuổi SHAP giảm và biến động lớn. Tuổi trung niên có tác động lớn nhất đến kết quả dự đoán.

Lưu ý hiệu năng

Tính SHAP trên tập lớn có thể tốn thời gian/bộ nhớ. Lấy mẫu một phần (ví dụ 100–1,000 hàng) để minh họa đồ thị waterfall, beeswarm.

9. Mẹo tham số LightGBM (tham khảo tài liệu)

Một số gợi ý thường dùng (xem thêm [LightGBM Parameters](#)):

- **num_leaves:** kiểm soát độ phức tạp cây; quy tắc ngón tay: $\text{num_leaves} \approx 2^{\max_depth}$.
- **feature_fraction, bagging_fraction, bagging_freq:** giảm phương sai, tăng tốc; giá trị 0.7–0.9 thường ổn định.
- **min_child_samples:** tăng để tránh overfitting trên lá quá nhỏ.
- **early_stopping_rounds:** dùng tập validation để dừng sớm (ví dụ 50–100 vòng).

10. Kết luận

Qua việc so sánh ba mô hình Boosting trên hai dataset khác nhau, chúng ta có thể rút ra những kết luận quan trọng:

10.1 Ưu điểm của LightGBM

LightGBM vượt trội trong các trường hợp:

- **Dataset lớn:** Với >100K mẫu, LightGBM nhanh hơn GradientBoosting 200x
- **Nhiều đặc trưng categorical:** EFB giúp xử lý hiệu quả
- **Bộ nhớ hạn chế:** Histogram và EFB tiết kiệm bộ nhớ
- **Cần tốc độ huấn luyện nhanh:** Leaf-wise growth tối ưu

10.2 So sánh với XGBoost

Khi nào chọn LightGBM:

- Dataset có nhiều đặc trưng categorical
- Cần tiết kiệm bộ nhớ
- Dataset rất lớn (>1M mẫu)

Khi nào chọn XGBoost:

- Dataset nhỏ đến trung bình
- Chỉ có đặc trưng numerical
- Cần regularization mạnh

10.3 Tầm quan trọng của SHAP

Với nhu cầu giải thích mô hình, **SHAP kết hợp cùng LightGBM** là lựa chọn mạnh mẽ:

- Giải thích dự đoán cho từng mẫu cụ thể
- So sánh feature importance giữa các mô hình
- Kiểm chứng tính nhất quán của kết quả
- Truyền thông kết quả đến stakeholders

10.4 Triển vọng tương lai

LightGBM tiếp tục phát triển với:

- Hỗ trợ GPU training
- Cải tiến thuật toán GOSS và EFB
- Tích hợp sâu hơn với các framework ML
- Tối ưu cho edge computing

Khuyến nghị cuối cùng

LightGBM là lựa chọn hàng đầu cho các bài toán tabular data lớn, đặc biệt khi có nhiều đặc trưng categorical. Kết hợp với SHAP, bạn có được một công cụ mạnh mẽ vừa chính xác vừa có thể giải thích được.

Dựng môi trường phát triển bằng Docker trên máy cá nhân

Võ Hoàng

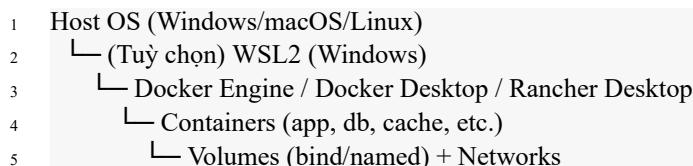
Giới thiệu

Bài viết này là một tutorial thực chiến giúp bạn thiết kế **môi trường phát triển (dev env)** dùng Docker ngay trên máy cá nhân. Mục tiêu: **nhanh, tái lập, ít “cồng kềnh”, dễ debug**, và tương thích nhóm.

1. Đối tượng & mục tiêu

- **Đối tượng:** Dev backend/frontend, data/ML, sinh viên, team nhỏ cần môi trường đồng nhất.
- **Mục tiêu:**
 - Chạy dự án local bằng docker compose up với hot-reload.
 - Tách biệt **dev** và **prod** bằng profiles.
 - Chuẩn hoá secrets, volumes, network.
 - Chạy tốt trên Windows (WSL2), macOS, Linux.

2. Kiến trúc tổng quan (host ↔ Docker)



Gợi ý:

- Windows: WSL2 + Docker Desktop (dễ nhất). Nếu tối giản: cài Docker Engine rootless trong WSL2.
- macOS: Docker Desktop hoặc Colima (nhẹ, nhanh, M-series friendly).
- Linux: Cài Docker Engine chính chủ.

3. Chuẩn bị theo hệ điều hành

3.1 Windows 10/11 (WSL2 + Docker Desktop)

```
1 wsl --install -d Ubuntu
2 wsl --set-default-version 2
```

Sau đó cài Docker Desktop, bật WSL 2 based engine, để project code trong WSL FS để tránh I/O chậm. Di chuyển dữ liệu Docker sang ổ D bằng thiết lập trong Docker Desktop hoặc wsl --export/--import.

3.2 macOS (Intel/M-series)

```
1 brew install colima docker  
2 colima start --cpu 4 --memory 8 --disk 50
```

3.3 Linux (Ubuntu/Debian)

```
1 sudo apt-get update  
2 sudo apt-get install -y ca-certificates curl gnupg lsb-release  
3 # Add Docker's official GPG key & repo  
4 sudo apt-get install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin  
5 sudo usermod -aG docker $USER  
6 newgrp docker
```

4. Tiêu chuẩn hóa bộ cục dự án

```
1 myapp/  
2   backend/  
3     app/  
4       main.py  
5     pyproject.toml  
6   Dockerfile  
7   Dockerfile.dev  
8   db/  
9     init.sql  
10  .env  
11  docker-compose.yml  
12  Makefile  
13  .dockerignore  
14  .editorconfig  
15  .gitattributes  
16  .devcontainer/  
17    devcontainer.json
```

Quy ước: Bind mount cho mã nguồn (dev), named volume cho dữ liệu (db, cache), secrets qua .env, profiles để tách dev/prod.

5. Ví dụ : FastAPI + PostgreSQL (hot-reload)

5.1 main.py

```
1 from fastapi import FastAPI  
2 from os import getenv  
3 import psycopg
```

```
4 app = FastAPI()
5
6 @app.get("/")
7 def read_root():
8     return {"msg": "Hello Docker Dev!", "env": getenv("APP_ENV", "dev")}
```

5.2 pyproject.toml

```
1 [project]
2 name = "myapp-backend"
3 version = "0.1.0"
4 dependencies = [
5     "fastapi",
6     "uvicorn[standard]",
7     "psycopg[binary]",
8 ]
```

5.3 Dockerfile.dev

```
1 FROM python:3.12-slim
2 WORKDIR /app
3 COPY pyproject.toml /app/
4 RUN pip install --no-cache-dir uv pip-tools && \
5     pip install --no-cache-dir fastapi uvicorn[standard] psycopg[binary]
6 CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```

5.4 docker-compose.yml

```
1 version: "3.9"
2 name: myapp
3
4 services:
5     api:
6         build:
7             context: ./backend
8             dockerfile: Dockerfile.dev
9         ports:
10            - "8000:8000"
11         env_file: .env
12         volumes:
13            - ./backend:/app:rw
14         depends_on:
15            - db
16         profiles: ["dev"]
17
18     db:
19         image: postgres:16
20         environment:
21             POSTGRES_USER: ${POSTGRES_USER}
22             POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
```

```
23 POSTGRES_DB: ${POSTGRES_DB}  
24 volumes:  
25   - pgdata:/var/lib/postgresql/data  
26   - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql:ro  
27 ports:  
28   - "5432:5432"  
29  
30 volumes:  
31 pgdata:
```

Chạy dev:

docker compose --profile dev up --build

Mở: <http://localhost:8000>.

6. Ví dụ 2: Next.js + Redis (hot-reload)

6.1 frontend/src/index.js

```
1 export default function Home() {  
2   return <h1>Hello Docker Dev with Next.js!</h1>  
3 }
```

6.2 frontend/package.json

```
1 {  
2   "name": "myapp-frontend",  
3   "version": "1.0.0",  
4   "scripts": {  
5     "dev": "next dev",  
6     "build": "next build",  
7     "start": "next start"  
8   },  
9   "dependencies": {  
10    "next": "14.0.0",  
11    "react": "18.2.0",  
12    "react-dom": "18.2.0"  
13  }  
14}
```

6.3 frontend/Dockerfile.dev

```
1 FROM node:20-slim  
2 WORKDIR /app  
3 COPY package.json package-lock.json* ./  
4 RUN npm install  
5 CMD ["npm", "run", "dev"]
```

6.4 docker-compose.override.yml (mở rộng)

```
1 services:  
2   frontend:  
3     build:  
4       context: ./frontend  
5       dockerfile: Dockerfile.dev  
6     ports:  
7       - "3000:3000"  
8     volumes:  
9       - ./frontend:/app  
10    depends_on:  
11      - redis  
12  profiles: ["dev"]  
13  
14  redis:  
15    image: redis:7  
16    ports:  
17      - "6379:6379"  
18    volumes:  
19      - redisdata:/data  
20  
21  volumes:  
22    redisdata:
```

Mở: <http://localhost:3000>. Hot reload hoạt động tự động.

7. CI/CD nâng cao (GitHub Actions + Docker Hub)

7.1 .github/workflows/deploy.yml

```
1 name: deploy  
2  
3 on:  
4   push:  
5     branches: [main]  
6  
7 jobs:  
8   build-and-push:  
9     runs-on: ubuntu-latest  
10    steps:  
11      - uses: actions/checkout@v4  
12      - name: Login Docker Hub  
13        uses: docker/login-action@v2  
14        with:  
15          username: ${{ secrets.DOCKERHUB_USERNAME }}  
16          password: ${{ secrets.DOCKERHUB_TOKEN }}  
17      - name: Build & Push  
18        run: |  
19          docker build -t myorg/myapp-backend:latest ./backend  
20          docker build -t myorg/myapp-frontend:latest ./frontend  
21          docker push myorg/myapp-backend:latest  
22          docker push myorg/myapp-frontend:latest
```

8. Triển khai thử nghiệm (Prod mini)

Chạy trên một server (VD: VPS):

```
git pull origin main  
docker compose --profile prod up -d --build
```

Muốn expose với domain + HTTPS:

- Dùng Traefik hoặc NGINX reverse proxy.
- Dùng Caddy (tự động cấp TLS).

8.1 Compose với Caddy

```
1 services:  
2   caddy:  
3     image: caddy:2  
4     ports:  
5       - "80:80"  
6       - "443:443"  
7     volumes:  
8       - ./Caddyfile:/etc/caddy/Caddyfile:ro  
9       - caddy_data:/data  
10      - caddy_config:/config  
11  
12 volumes:  
13   caddy_data:  
14   caddy_config:
```

8.2 Caddyfile mẫu

```
1 myapp.localhost {  
2   reverse_proxy frontend:3000  
3 }
```

9. So sánh Docker Desktop vs thay thế

Tính năng	Docker Desktop	Colima (macOS)	Rancher Desktop (Win/mac)
GUI quản lý container	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Dễ cài trên Win/mac	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Hỗ trợ Kubernetes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Dùng ít RAM hơn	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

10. Best practices nâng cao

- Luôn viết multi-stage build để image nhỏ gọn.
- Tách dev/prod config.

- Dùng linter + CI để check Dockerfile.
- Dùng healthcheck để giám sát service.
- Cache dependencies hợp lý (COPY requirements.txt trước).

11. Inclusion

Qua toàn bộ hành trình trên, bạn đã thấy rằng Docker không chỉ là công cụ chạy container đơn lẻ mà còn là nền tảng để chuẩn hóa môi trường phát triển. Khi biết cách tổ chức dự án, tách cấu hình dev/prod, quản lý volumes, secrets, và sử dụng thêm các công cụ như Makefile, Dev Containers, CI/CD pipeline, bạn sẽ có:

- Một môi trường dev nhất quán, không còn cảnh “works on my machine”.
- Quy trình on-boarding nhanh chóng: chỉ cần git clone và docker compose up.
- Khả năng mở rộng sang full-stack (backend + frontend + database + cache) chỉ với vài service trong Compose.
- Dễ dàng nâng cấp: từ local → staging/prod → Kubernetes mà không phải viết lại quá nhiều.
- Tích hợp liền mạch với các công cụ hiện đại: GitHub Actions, secret managers, GPU runtime cho AI/ML.

Điều quan trọng hơn cả: Docker giúp bạn tự duy theo hướng *infrastructure as code* – coi môi trường dev cũng là một phần của dự án, được quản lý version control, test, và tối ưu như chính mã nguồn. Đây chính là nền tảng để tiến tới CI/CD chuyên nghiệp và triển khai microservices ở quy mô lớn.

Hãy thử áp dụng các mẫu cấu trúc trong bài viết cho dự án của bạn, tuỳ biến cho phù hợp. Khi đã quen, bạn sẽ thấy việc tạo một môi trường dev hoàn chỉnh chỉ mất vài phút – và lợi ích mang lại thì vô cùng lớn.

Dự báo Doanh số với Explainable AI (XAI): Một Case Study kết hợp LightGBM và SHAP

Vương Nguyệt Bình

Tóm tắt nội dung

Dự báo doanh số bán hàng là một thách thức lớn đối với doanh nghiệp, khi mà hành vi tiêu dùng chịu ảnh hưởng đồng thời từ nhiều yếu tố như mùa vụ, ngày lễ, thời tiết, và xu hướng thị trường.

Trong nghiên cứu này, chúng tôi xây dựng hệ thống dự báo dựa trên thuật toán **LightGBM**, kết hợp với **SHAP** để giải thích kết quả dự báo, sử dụng **Optuna** cho tối ưu tham số và triển khai qua ứng dụng web **Streamlit**.

Điểm nổi bật là hệ thống vừa đạt độ chính xác cao, vừa minh bạch, tạo điều kiện cho nhà quản lý kinh doanh ra quyết định nhanh chóng và đáng tin cậy.

1. Giới thiệu

1.1 Bối cảnh kinh doanh

Trong ngành bán lẻ, việc dự báo doanh số đóng vai trò trung tâm. Một dự báo chính xác giúp doanh nghiệp:

- Lập kế hoạch tồn kho, tránh thiếu hoặc thừa hàng.
- Bố trí nhân sự hợp lý trong các ngày cao điểm.
- Lên chiến lược khuyến mãi và marketing phù hợp.

Thách thức chính: Doanh số bị ảnh hưởng bởi nhiều yếu tố *không tuyến tính* và *khó dự đoán*, ví dụ: thời tiết xấu làm khách hàng ít đi mua sắm, ngày lễ kéo doanh số tăng vọt, hoặc xu hướng đột ngột từ mạng xã hội.

1.2 Vấn đề của mô hình “hộp đen”

Nhiều thuật toán AI hiện đại cho kết quả chính xác nhưng không thể giải thích được. Nhà quản lý kinh doanh cần biết không chỉ “dự báo là bao nhiêu”, mà còn “tại sao lại như vậy”.

Ví dụ: Nếu hệ thống báo rằng tuần tới doanh số sẽ giảm 15%, một nhà quản lý sẽ đặt câu hỏi: “Giảm là do thời tiết, do đối thủ cạnh tranh, hay do mùa vụ?”

1.3 Mục tiêu dự án

Dự án này được thiết kế với hai nhóm mục tiêu chính: **mục tiêu kinh doanh** và **mục tiêu kỹ thuật**. Điều này đảm bảo rằng nỗ lực kỹ thuật không tách rời giá trị mà doanh nghiệp nhận được.

Mục tiêu Kinh doanh:

- **Độ chính xác:** đạt được mức dự báo cao, đủ để cải thiện kế hoạch và giảm lãng phí.
- **Khả năng giải thích:** cung cấp lý do đằng sau mỗi con số, giúp nhà quản lý tự tin hơn khi ra quyết định.

- **Khả năng mở rộng:** dễ dàng áp dụng cho nhiều chi nhánh hoặc cửa hàng khác nhau.
- **Tính khả dụng:** giao diện thân thiện, để cả nhân viên không chuyên về kỹ thuật cũng sử dụng được.

Mục tiêu Kỹ thuật:

- **Hiệu suất mô hình:** giảm sai số RMSE, MAE, MAPE, vượt qua baseline Prophet.
- **Kỹ thuật đặc trưng:** tạo ra bộ đặc trưng phong phú (thời gian, lag, rolling statistics...).
- **Khả năng giải thích:** cung cấp cả phân tích SHAP toàn cục (global) và cục bộ (local).
- **Sẵn sàng sản xuất:** mã nguồn module hóa, dễ bảo trì, có tài liệu đầy đủ.

Điểm thú vị là các mục tiêu này không tách biệt, mà bổ sung cho nhau: một mô hình chính xác nhưng khó hiểu thì không có giá trị, một mô hình dễ hiểu nhưng kém chính xác cũng vô dụng. Dự án này chọn con đường dung hòa cả hai.

Gợi ý hình minh họa: Bảng tóm tắt hai nhóm mục tiêu: cột trái là kinh doanh (accuracy, interpretability, scalability, usability), cột phải là kỹ thuật (RMSE, feature engineering, SHAP, production readiness).

2. Các công nghệ cốt lõi sử dụng

Không phải ngẫu nhiên mà dự án chọn LightGBM, SHAP, Optuna và Streamlit. Mỗi công nghệ đều được lựa chọn để giải quyết một “nút thắt” cụ thể. Cùng điểm qua:

LightGBM – Trái tim của mô hình dự báo LightGBM là một framework *gradient boosting* nổi tiếng, được tối ưu cho tốc độ và hiệu suất, đặc biệt phù hợp với dữ liệu dạng bảng (tabular data). Với hơn 50 đặc trưng được tạo ra, LightGBM xử lý nhanh chóng, khai thác được cả mối quan hệ phi tuyến tính trong dữ liệu. So với Prophet, LightGBM cho khả năng học sâu hơn từ các đặc trưng phức tạp, tạo ra dự báo chính xác hơn.

SHAP – Khi AI biết giải thích SHAP (*SHapley Additive exPlanations*) là công cụ biến mô hình “hộp đen” thành một mô hình **có thể giải thích được**. Nó dựa trên lý thuyết trò chơi để tính giá trị Shapley cho từng đặc trưng, nói cách khác: “mỗi yếu tố đóng góp bao nhiêu phần vào kết quả cuối cùng?”. Nhờ SHAP, các nhà quản lý không chỉ thấy con số dự báo, mà còn hiểu vì sao mô hình đưa ra kết quả đó.

Optuna – Tối ưu hoá thông minh Huấn luyện LightGBM không chỉ là cho dữ liệu vào rồi bấm nút. Hiệu suất của nó phụ thuộc mạnh mẽ vào các siêu tham số (hyperparameters) như num_leaves, learning_rate, reg_alpha, reg_lambda... Thay vì thử tay thủ công, nhóm dùng **Optuna** – một framework tối ưu hoá tự động. Nó giúp thử nghiệm hàng trăm cấu hình khác nhau và chọn ra bộ tham số tốt nhất, tiết kiệm rất nhiều thời gian và công sức.

Streamlit – Cầu nối đến người dùng cuối Một mô hình AI mạnh đến đâu mà không ai dùng thì cũng vô nghĩa. Streamlit giúp biến mô hình thành một **ứng dụng web tương tác**, nơi người dùng (kể cả không rành kỹ thuật) có thể:

- Xem dữ liệu lịch sử bán hàng.

- Nhận dự báo doanh số cho tương lai gần.
- Xem giải thích trực quan (biểu đồ SHAP) cho từng dự đoán.

Điều này biến AI từ một “món đồ trong phòng lab” thành một **công cụ kinh doanh thực thụ**.

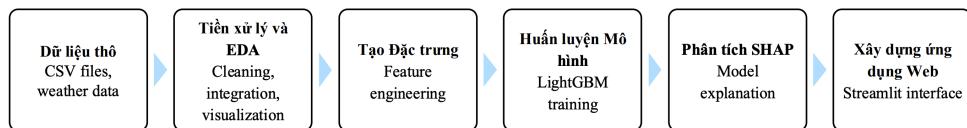
Các thư viện hỗ trợ khác Đằng sau các công nghệ chính, còn có cả một hệ sinh thái thư viện hỗ trợ:

- **Pandas, NumPy**: xử lý dữ liệu nhanh và hiệu quả.
- **Scikit-learn**: cung cấp công cụ chuẩn hóa và hỗ trợ huấn luyện.
- **Matplotlib, Plotly**: trực quan hóa dữ liệu và kết quả.

Sự kết hợp này thể hiện một tư duy trưởng thành trong việc xây dựng sản phẩm AI: không chỉ tập trung vào mô hình, mà nhìn vào **toàn bộ vòng đời giải pháp**, từ dữ liệu đến người dùng cuối.

3. Quy trình triển khai

3.1 Tổng quan pipeline



Hình 16: Các bước triển khai dự án

Nếu coi AI như một “đầu bếp dữ liệu”, thì quy trình triển khai mô hình dự báo chính là công thức nấu ăn. Một món ăn ngon không chỉ phụ thuộc vào nguyên liệu, mà còn ở cách chuẩn bị, cách nấu, và cách trình bày. Dự án này cũng vậy: từ dữ liệu thô đến mô hình hoạt động, mọi bước đều được thiết kế cẩn thận.

3.2 Bước 1: Tiền xử lý Dữ liệu

Mọi mô hình tốt đều bắt đầu từ dữ liệu tốt. Nhóm tích hợp hai nguồn dữ liệu chính:

- **Dữ liệu bán hàng** (2016–2017): bao gồm doanh số và số lượng giao dịch.
- **Dữ liệu thời tiết**: các yếu tố như nhiệt độ, độ ẩm, lượng mưa.

Sau khi gom dữ liệu, công việc quan trọng là **làm sạch**:

- *Giá trị thiếu*: điền bằng trung bình, trung vị hoặc nội suy.
- *Ngoại lệ*: phát hiện bằng IQR (interquartile range) hoặc Z-score và xử lý.

Mục tiêu của bước này: biến dữ liệu thô thành một bộ dữ liệu **sạch, nhất quán, đáng tin cậy**.

3.3 Bước 2: Phân tích Khám phá Dữ liệu (EDA)

EDA giống như giai đoạn “làm quen với nguyên liệu”. Nhóm phân tích dữ liệu để tìm ra:

- **Tính mùa vụ:** ví dụ, doanh số cao vào một số tháng nhất định.
- **Xu hướng dài hạn:** liệu doanh số có tăng đều qua các năm?
- **Tương quan:** giữa thời tiết (nhiệt độ, mưa) và doanh số bán hàng.

Kết quả của EDA giúp xác định các tín hiệu ẩn trong dữ liệu, đồng thời cung cấp trực giác quan trọng để thiết kế đặc trưng và chọn mô hình.

3.4 Bước 3: Kỹ thuật Tạo Đặc trưng (Feature Engineering)

Nếu dữ liệu là nguyên liệu, thì đặc trưng chính là “gia vị” quyết định độ ngon. Dự án tạo ra hơn **50 đặc trưng**, bao gồm:

- **Đặc trưng thời gian:** ngày trong tuần, tháng, quý, ngày lễ, cuối tuần.
- **Lag features (độ trễ):** doanh số của 1, 7, 14, 28 ngày trước.
- **Thống kê cuộn (rolling statistics):** trung bình động 7 ngày, độ lệch chuẩn 14 ngày.

Nhờ đó, mô hình không chỉ nhìn thấy “ngày hôm nay”, mà còn hiểu được ngữ cảnh lịch sử và quy luật ẩn sau dữ liệu.

3.5 Bước 4: Phát triển Mô hình

Nhóm áp dụng chiến lược hai tầng:

- **Prophet:** dùng làm baseline. Đây là mô hình phổ biến cho chuỗi thời gian, xử lý tốt mùa vụ, dễ triển khai.
- **LightGBM:** mô hình nâng cao, tận dụng đặc trưng phức tạp và phi tuyến tính để vượt qua Prophet.

Để đảm bảo kết quả khách quan, dự án dùng **time series splitting**, tránh việc rò rỉ dữ liệu từ tương lai về quá khứ (data leakage).

Kết quả: LightGBM vượt trội hơn Prophet, chứng minh giá trị của việc đầu tư vào đặc trưng và mô hình phi tuyến.

3.6 Bước 5: Tối ưu hoá Siêu tham số (Hyperparameter Tuning)

Mô hình mạnh chưa đủ, cần được “điều chỉnh” tinh vi. Optuna được sử dụng để tự động hóa quá trình này:

- Thử hơn 100 cấu hình tham số khác nhau.
- Tối ưu các tham số quan trọng như num_leaves, learning_rate, reg_alpha, reg_lambda.
- Áp dụng cross-validation cho chuỗi thời gian để tránh overfitting.

Kết quả là một mô hình **nghẹt thở, nhanh hơn, chính xác hơn**, sẵn sàng để bước sang giai đoạn triển khai thực tế.

4. Phân tích Khả năng Giải thích (XAI)

Một mô hình AI giỏi có thể nói cho bạn biết “*ngày mai sẽ bán được 1.200 sản phẩm*”. Nhưng một mô hình AI giỏi **và minh bạch** sẽ còn nói thêm: “*1.200 sản phẩm vì thời tiết đẹp, tuần trước bán chạy, và hôm nay rơi vào dịp lễ*”.

Đây chính là giá trị cốt lõi của **XAI – Explainable AI**, giúp biến dự đoán thành một **câu chuyện dễ hiểu và hành động được**.

Trong dự án này, nhóm đã dùng **SHAP** để phân tích mô hình LightGBM, với hai góc nhìn: toàn cục và cục bộ.

4.1 Giải thích Toàn cục (Global Explanations)

SHAP toàn cục cho một bức tranh tổng quan: những yếu tố nào ảnh hưởng mạnh nhất đến doanh số?

Ví dụ:

- Ngày lễ và cuối tuần thường đầy doanh số tăng cao.
- Thời tiết xấu (mưa nhiều) làm doanh số giảm đáng kể.
- Doanh số tuần trước là tín hiệu mạnh để dự báo tuần tới.

Với thông tin này, nhà quản lý có thể xây dựng chiến lược dài hạn: nếu dữ liệu cho thấy thời tiết có ảnh hưởng mạnh, thì các cửa hàng nên tăng cường khuyến mãi online vào mùa mưa, thay vì cố gắng kéo khách đến cửa hàng vật lý.

4.2 Giải thích Cục bộ (Local Explanations)

Nếu toàn cục là cái nhìn từ trên cao, thì cục bộ chính là soi kính lúp vào từng dự đoán riêng lẻ.

Ví dụ: mô hình dự báo rằng “*Ngày mai, cửa hàng A sẽ bán được ít hơn bình thường*”. Nhưng tại sao? Nhờ SHAP, nhà quản lý thấy rõ:

- “Doanh số tuần trước giảm” đóng góp -150 sản phẩm.
- “Thời tiết xấu” đóng góp -80 sản phẩm.
- “Hôm nay là thứ 7” đóng góp +30 sản phẩm.

Tổng hợp lại, kết quả dự báo thấp hơn bình thường là hợp lý. Và quan trọng hơn: người quản lý có thể **hành động**, ví dụ tung thêm khuyến mãi cuối tuần để bù lại tác động tiêu cực từ thời tiết.

4.3 Từ con số sang hành động

Khả năng giải thích biến mô hình dự báo từ một “máy tính toán” thành một **công cụ ra quyết định**. Người dùng không chỉ biết “*chuyện gì sẽ xảy ra*”, mà còn hiểu “*tại sao*” và “*cần làm gì*”.

Chẳng hạn:

- Nếu doanh số dự báo giảm do **thời tiết**, giải pháp có thể là tăng quảng bá online.

- Nếu do **thiếu ngày lễ**, có thể bù bằng các chương trình khuyến mãi nhân tạo “mini-event”.
- Nếu do **xu hướng giảm dài hạn**, cần xem xét lại chiến lược sản phẩm.

Nói cách khác, XAI không chỉ làm cho AI đáng tin cậy hơn, mà còn khiến nó trở thành một **người đồng hành chiến lược** trong doanh nghiệp.

5. Kết quả và Thảo luận

5.1 Hiệu suất Mô hình

Khi so sánh với Prophet – mô hình baseline phổ biến cho chuỗi thời gian – LightGBM cho thấy sự vượt trội rõ rệt. Trên mọi chỉ số đánh giá (RMSE, MAE, MAPE), LightGBM đều cho sai số thấp hơn.

Điều này chứng minh rằng việc đầu tư vào kỹ thuật tạo đặc trưng, cùng với khả năng xử lý phi tuyến của LightGBM, mang lại giá trị thiết thực vượt xa các mô hình truyền thống.

Gợi ý hình minh họa: Biểu đồ cột so sánh RMSE, MAE, MAPE giữa Prophet và LightGBM.

5.2 Phân tích SHAP

Bên cạnh độ chính xác, SHAP cho ta cái nhìn sâu sắc hơn về cách mô hình ra quyết định:

- **Mùa vụ** chiếm hơn 40% ảnh hưởng đến dự báo – minh chứng rằng các dịp lễ, cuối tuần đóng vai trò quyết định.
- **Lag features** (dữ liệu quá khứ) giúp mô hình nắm bắt được chu kỳ tiêu dùng, đặc biệt là xu hướng “tuần trước ảnh hưởng tuần sau”.
- **Thời tiết** không phải yếu tố chính, nhưng vẫn góp phần đáng kể trong nhiều trường hợp (ví dụ, mưa làm giảm doanh số trực tiếp).

Những phát hiện này vừa xác nhận trực giác kinh doanh (ngày lễ quan trọng), vừa hé lộ thêm chi tiết ẩn (chu kỳ tiêu dùng từ lag features).

Hình 17: Chú thích ảnh

5.3 Ứng dụng Thực tế

Từ góc nhìn kinh doanh, mô hình không chỉ đưa ra dự báo, mà còn biến thành công cụ hỗ trợ ra quyết định.

Người quản lý có thể:

- **Xem dự báo kèm giải thích minh bạch**: không còn là “con số từ hộp đen”.
- **Lập kế hoạch tồn kho chính xác hơn**: giảm lãng phí và thiếu hụt.
- **Ra quyết định dựa trên dữ liệu thay vì cảm tính**: tăng độ tin cậy và tính bền vững trong quản lý.

Nói cách khác, dự án đã vượt qua ranh giới của một mô hình dự báo thông thường, trở thành một công cụ **ra quyết định chiến lược**.

6. Hạn chế và Bài học Rút ra

Dù dự án đạt được nhiều thành tựu, vẫn còn tồn tại những hạn chế và bài học quý giá. Nhìn thẳng vào những điểm này giúp doanh nghiệp có cái nhìn thực tế, và là cơ sở để cải tiến trong tương lai.

6.1 Hạn chế

- **Nguồn dữ liệu còn hạn chế:** mới chỉ sử dụng dữ liệu bán hàng và thời tiết. Các yếu tố khác như dữ liệu đối thủ, chỉ số kinh tế vĩ mô, hay dữ liệu từ mạng xã hội chưa được tích hợp.
- **Mô hình dừng ở LightGBM:** mặc dù mạnh mẽ, nhưng dự án chưa thử nghiệm các kiến trúc deep learning (LSTM, Transformer) vốn có lợi thế với chuỗi thời gian dài và phức tạp.

Gợi ý hình minh họa: Infographic “Hiện tại vs Tiềm năng”: dữ liệu hiện dùng (bán hàng, thời tiết) và dữ liệu tiềm năng (kinh tế, đối thủ, social media).

6.2 Bài học Rút ra

- **Chất lượng dữ liệu là nền tảng:** không có dữ liệu sạch và giàu đặc trưng, mô hình sẽ không phát huy được sức mạnh.
- **Khả năng giải thích quan trọng không kém độ chính xác:** chính XAI giúp nhà quản lý tin tưởng và sẵn sàng hành động dựa trên AI.
- **Ứng dụng web thân thiện quyết định mức độ chấp nhận:** một giao diện dễ dùng như Streamlit đã biến mô hình từ “công cụ phòng lab” thành “người bạn đồng hành” của đội kinh doanh.

7. Các Hướng Phát triển Tương lai

Một PoC thành công chưa đủ để trở thành một sản phẩm doanh nghiệp thực thụ. Điều quan trọng là phải có một **lộ trình phát triển rõ ràng**, để dự án không chỉ dừng lại ở mức thử nghiệm mà còn tạo ra tác động lâu dài.

Dưới đây là bốn hướng phát triển được nhóm đề xuất:

7.1 Cải thiện Mô hình

Mô hình LightGBM đã chứng minh hiệu quả, nhưng vẫn còn nhiều tiềm năng để khai thác thêm:

- **Ensemble methods:** kết hợp nhiều mô hình (stacking, blending) để tăng độ chính xác và ổn định.

- **Deep learning:** tích hợp các kiến trúc như LSTM hoặc Transformer – vốn rất mạnh với chuỗi thời gian dài và phức tạp.
- **Online learning:** để mô hình có thể cập nhật liên tục từ dữ liệu mới, duy trì độ chính xác theo thời gian.

Gợi ý hình minh họa: Biểu đồ so sánh “single model” vs “ensemble” vs “deep learning”.

7.2 Mở rộng Kỹ thuật Tạo Đặc trưng

Đặc trưng chính là “ngôn ngữ” mà mô hình sử dụng để hiểu dữ liệu. Do đó, việc mở rộng kỹ thuật tạo đặc trưng có thể tăng đáng kể sức mạnh dự báo:

- **Tích hợp dữ liệu ngoài:** chỉ số kinh tế vĩ mô, dữ liệu mạng xã hội (social media sentiment), sự kiện đặc biệt (chiến dịch marketing lớn).
- **Đặc trưng nâng cao:** áp dụng các kỹ thuật như Wavelets, Fourier Transform để phân tích chu kỳ ẩn trong dữ liệu.
- **Đặc trưng liên cửa hàng (cross-store):** mô hình hóa mối quan hệ giữa các cửa hàng để học tác động chéo.

Gợi ý hình minh họa: Infographic các loại dữ liệu mở rộng: kinh tế, social media, sự kiện đặc biệt.

7.3 Nâng cao Khả năng Giải thích

SHAP là một bước tiến lớn, nhưng vẫn có thể đi xa hơn:

- **Counterfactual explanations:** trả lời câu hỏi “Điều gì sẽ xảy ra nếu...?”, ví dụ “Nếu tăng ngân sách marketing 10%, doanh số có thể thay đổi thế nào?”.
- **Causal inference:** phân tích quan hệ nhân quả, không chỉ dừng ở mức tương quan, để xác định yếu tố nào thực sự gây ra thay đổi doanh số.

Điều này biến mô hình dự báo thành một công cụ không chỉ giải thích quá khứ, mà còn gợi ý hướng đi cho tương lai.

7.4 Mở rộng Kinh doanh

Để thực sự đi vào hoạt động, hệ thống cần tích hợp sâu hơn với quy trình doanh nghiệp:

- **API development:** xây dựng kiến trúc hướng dịch vụ (SOA), để các hệ thống khác dễ dàng gọi dự báo.
- **Real-time predictions:** tích hợp dữ liệu streaming để dự báo tức thời, phục vụ quyết định nhanh.
- **Advanced analytics:** từ dự báo chuyên thành gợi ý hành động, ví dụ để xuất chiến lược tồn kho tối ưu.

Gợi ý hình minh họa: Roadmap phát triển 4 hướng: Model – Feature – XAI – Business.

7.5 Tầm nhìn dài hạn

Nếu các bước trên được thực hiện, dự án sẽ chuyển đổi từ một **PoC kĩ thuật** thành một **nền tảng chiến lược** cho doanh nghiệp. Nó không chỉ trả lời câu hỏi “*Doanh số ngày mai sẽ ra sao?*”, mà còn giúp doanh nghiệp hoạch định chiến lược dài hạn, ứng phó kịp thời với biến động thị trường, và tận dụng tối đa sức mạnh của dữ liệu.

8. Kết luận

Dự án chứng minh rằng AI có thể mang lại dự báo doanh số chính xác và minh bạch. Explainable AI (XAI) giúp xây dựng niềm tin, biến kết quả AI thành công cụ hỗ trợ ra quyết định thực sự trong kinh doanh.

Mã nguồn dự án: https://github.com/nguyenhads/sales_forecasting_xai