Module 4 - Tuần 2 - XGBoost: Một hệ thống tăng cường cây (tree boosting) có khả năng mở rộng

Time-Series Team

Ngày 21 tháng 9 năm 2025

Mục lục

1 Giới thiệu

Học máy và các phương pháp dựa trên dữ liệu ngày càng trở nên quan trọng trong nhiều lĩnh vực. Các bộ lọc thư rác thông minh bảo vệ email của chúng ta bằng cách học từ khối lượng lớn dữ liệu spam và phản hồi của người dùng; hệ thống quảng cáo học cách ghép nối quảng cáo phù hợp với ngữ cảnh; hệ thống phát hiện gian lận giúp ngân hàng chống lại kẻ tấn công; hệ thống phát hiện sự kiện bất thường hỗ trợ các nhà vật lý tìm ra hiện tượng dẫn đến khám phá khoa học mới.

Hai yếu tố quan trọng thúc đẩy thành công của các ứng dụng này là:

1. Sử dụng các mô hình thống kê hiệu quả để nắm bắt sự phụ thuộc phức tạp trong dữ liệu.
2. Các hệ thống học có khả năng mở rộng để huấn luyện mô hình từ tập dữ liệu khổng lồ.

Trong số các phương pháp học máy thực tiễn, **gradient tree boosting** là một kỹ thuật nổi bật trong nhiều ứng dụng. Tree boosting đã được chứng minh mang lại kết quả tiên tiến nhất trên nhiều bộ dữ liệu phân loại chuẩn. **LambdaMART**, một biến thể của tree boosting dùng cho bài toán xếp hạng, cũng đạt kết quả hàng đầu. Ngoài ra, tree boosting không chỉ được dùng độc lập mà còn tích hợp vào pipeline sản xuất trong các hệ thống dự đoán tỉ lệ nhấp quảng cáo. Đây cũng là phương pháp **ensemble** mặc định trong nhiều cuộc thi, tiêu biểu như **Netflix Prize**.

Trong bài báo này, chúng tôi giới thiệu **XGBoost**, một hệ thống học máy có khả năng mở rộng cho tree boosting, hiện có dưới dạng **mã nguồn mở**. Tác động của XGBoost được công nhận rộng rãi qua nhiều thách thức học máy và khai phá dữ liệu. Ví dụ, trong số 29 lời giải thắng cuộc trên Kaggle được công bố năm 2015, có tới **17 giải pháp sử dụng XGBoost**. Trong đó, 8 lời giải chỉ dùng duy nhất XGBoost, còn lại thường kết hợp với mạng nơ-ron trong các mô hình ensemble. Phương pháp phổ biến thứ hai là mạng nơ-ron sâu, chỉ xuất hiện trong 11 lời giải. Tại **KDD Cup 2015**, tất cả 10 đội đứng đầu đều sử dụng XGBoost, và họ báo cáo rằng mô hình ensemble chỉ nhỉnh hơn XGBoost một chút.

Những kết quả này cho thấy XGBoost mang lại hiệu suất tiên tiến trên nhiều loại bài toán khác nhau, bao gồm: dự đoán doanh số cửa hàng, phân loại sự kiện vật lý năng lượng cao, phân loại văn bản web, dự đoán hành vi khách hàng, phát hiện chuyển động, dự đoán tỉ lệ nhấp quảng cáo, phân loại mã độc, phân loại sản phẩm, dự đoán rủi ro thiên tai, dự đoán tỷ lệ bỏ học trong các khóa học trực tuyến quy mô lớn.

Yếu tố quan trọng nhất đứng sau thành công của XGBoost là **khả năng mở rộng vượt trội trong mọi tình huống**. Hệ thống chạy nhanh gấp hơn 10 lần so với các giải pháp phổ biến trên một máy đơn, và mở rộng đến hàng tỷ mẫu trong môi trường phân tán hoặc hạn chế bộ nhớ.

Những cải tiến bao gồm:

- Thuật toán học cây mới xử lý dữ liệu thưa.
- Kỹ thuật **weighted quantile sketch** để xử lý trọng số trong học cây xấp xỉ.
- Khả năng tính toán song song và phân tán, giúp tăng tốc huấn luyện.
- Khai thác **out-of-core computation**, cho phép xử lý hàng trăm triệu mẫu ngay cả trên máy tính cá nhân.

Quan trọng hơn, việc kết hợp các kỹ thuật này tạo nên một hệ thống đầu-cuối có thể mở rộng đến tập dữ liệu cực lớn với lượng tài nguyên cụm nhỏ nhất.

2 Tóm lược về Tree Boosting

Phần suy diễn xuất phát từ ý tưởng trong các công trình trước về gradient boosting, đặc biệt phương pháp bậc hai được đề xuất bởi Friedman et al.

2.1 Regularized Learning Objective

Với một tập dữ liệu gồm n mẫu và m đặc trưng:

$$D = \{(x_i, y_i)\}, \quad |D| = n,\ x_i \in \mathbb{R}^m,\ y_i \in \mathbb{R},$$

một mô hình **tập hợp cây (tree ensemble model)** (Hình **??**) sử dụng K hàm cộng dồn để dự đoán đầu ra:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F},$$

trong đó $\mathcal{F} = \{f(x) = w_{q(x)}\}$ là không gian của các **cây hồi quy** (CART). Ở đây q biểu diễn cấu trúc của cây ánh xạ một mẫu vào chỉ số lá tương ứng, và T là số lá trong cây. Mỗi f_k tương ứng với một cấu trúc cây độc lập q và trọng số lá w.

Khác với cây quyết định, mỗi cây hồi quy chứa một **giá trị liên tục** tại mỗi lá. Ký hiệu w_i là giá trị tại lá thứ i. Với một mẫu x_i, ta áp dụng các luật phân nhánh trong cây (cho bởi q) để tìm lá tương ứng, và dự đoán cuối cùng bằng cách cộng dồn giá trị ở các lá đó (cho bởi w).

Để học được tập hợp các hàm trong mô hình, ta cực tiểu hóa **hàm mục tiêu có điều chuẩn**:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k),$$

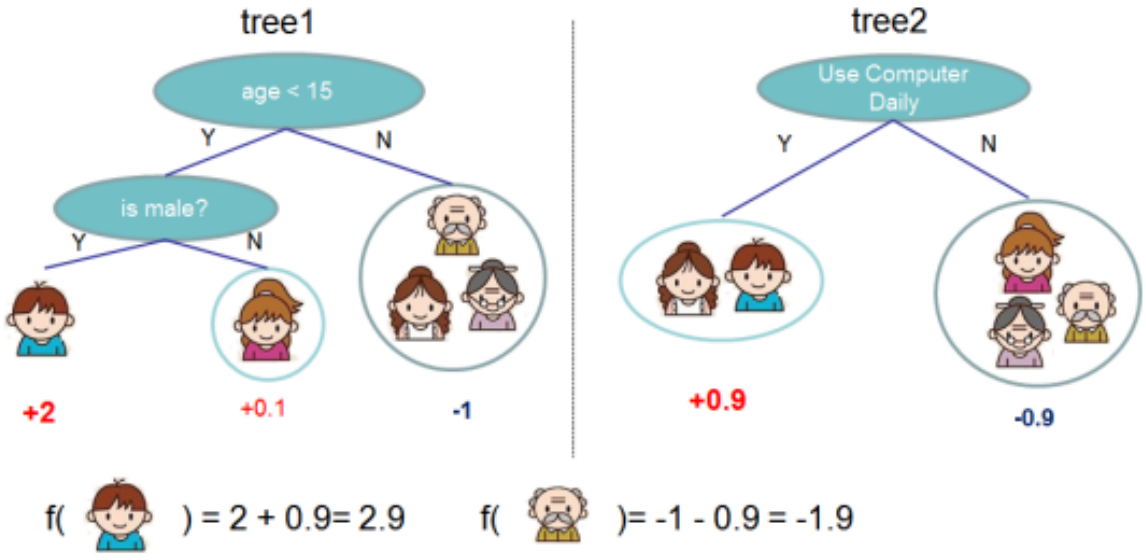
trong đó:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2.$$

Ở đây l là một hàm mất mát lồi khả vi, đo sự khác biệt giữa dự đoán $\hat{y}_i$ và nhãn thực y_i. Thành phần Ω dùng để phạt độ phức tạp của mô hình (tức là các cây hồi quy).

Thuật ngữ điều chuẩn bổ sung này giúp **làm trơn các trọng số học được**, nhằm tránh hiện tượng quá khớp (overfitting). Trực giác cho thấy, hàm mục tiêu điều chuẩn sẽ ưu tiên chọn mô hình đơn giản nhưng vẫn có sức mạnh dự đoán.

Kỹ thuật điều chuẩn tương tự cũng đã được sử dụng trong mô hình **Regularized Greedy Forest (RGF)** Tuy nhiên, hàm mục tiêu và thuật toán học của chúng tôi đơn giản hơn RGF và dễ song song hóa hơn. Khi tham số điều chuẩn bằng 0, mô hình trở về **gradient tree boosting truyền thống**.



Hình 1: Mô hình Tree Ensemble: Dự đoán cuối cùng cho một mẫu là tổng dự đoán từ các cây con.

Giải thích

Trong XGBoost, hàm mất mát được viết dưới dạng:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k),$$

trong đó:

$$\Omega(f) = \gamma T + \tfrac{1}{2}\lambda \|w\|^2.$$

- Thành phần $l(\hat{y}_i, y_i)$ đo lường sai số dự đoán so với giá trị thật.
- Thành phần $\sum_k \Omega(f_k)$ là **điều chuẩn**, giúp phạt mô hình quá phức tạp:
  - γT: phạt theo số lá T trong cây, hạn chế cây quá nhiều lá.
  - $\frac{1}{2}\lambda\|w\|^2$: phạt bình phương trọng số lá, ngăn giá trị dự đoán cực đoan.

Nhờ vậy, XGBoost cân bằng giữa **độ chính xác huấn luyện** và **khả năng tổng quát**, tránh overfitting.

Ví dụ minh họa

Bài toán: dự đoán điểm thi từ số giờ học. Dữ liệu: 8 điểm cơ bản + 60 điểm nhiễu để kiểm tra khả năng tổng quát hóa.

Code thực nghiệm

Bài toán dự đoán điểm số theo giờ học của các học sinh, dựa trên dữ liệu thực tế và sau đó tạo thêm 60 điểm dữ liệu gây nhiễu cho mô hình

Bảng 1: Dữ liệu điểm thi theo số giờ học

Số giờ học (x)	Điểm thi (y)
1	55
2	60
3	62
4	70
5	75
6	78
7	85
8	90

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Dataframe and data noise
rng = np.random.RandomState(42)
X_base = np.array([1,2,3,4,5,6,7,8]).reshape(-1,1)
y_base = np.array([55,60,62,70,75,78,85,90])
```

```python

X_extra = rng.uniform(0.5, 9.5, size=60).reshape(-1,1)
y_extra = 48 + 5.2*X_extra.squeeze() + \
          4*np.tanh((X_extra.squeeze()-6)/1.2) + \
          rng.normal(0, 3.5, size=X_extra.shape[0])
X = np.vstack([X_base, X_extra])
y = np.hstack([y_base, y_extra])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.28, random_state=0
)

# Model with regularization
model_reg = xgb.XGBRegressor(
    n_estimators=250, max_depth=3, learning_rate=0.08,
    subsample=0.85, colsample_bytree=0.9,
    reg_lambda=5.0, gamma=1.0, min_child_weight=5,
    random_state=0, tree_method="hist"
)
model_reg.fit(X_train, y_train)

# Model without regularization
model_no = xgb.XGBRegressor(
    n_estimators=500, max_depth=6, learning_rate=0.1,
    subsample=1.0, colsample_bytree=1.0,
    reg_lambda=0.0, gamma=0.0, min_child_weight=1,
    random_state=0, tree_method="hist"
)
model_no.fit(X_train, y_train)

# Evaluation
def eval_model(model, X_tr, y_tr, X_te, y_te):
    yhat_tr = model.predict(X_tr)
    yhat_te = model.predict(X_te)
    return {
        "RMSE_train": np.sqrt(mean_squared_error(y_tr, yhat_tr)),
        "RMSE_test": np.sqrt(mean_squared_error(y_te, yhat_te)),
        "R2_train": r2_score(y_tr, yhat_tr),
        "R2_test": r2_score(y_te, yhat_te),
    }

print(eval_model(model_reg, X_train, y_train, X_test, y_test))
print(eval_model(model_no, X_train, y_train, X_test, y_test))
```

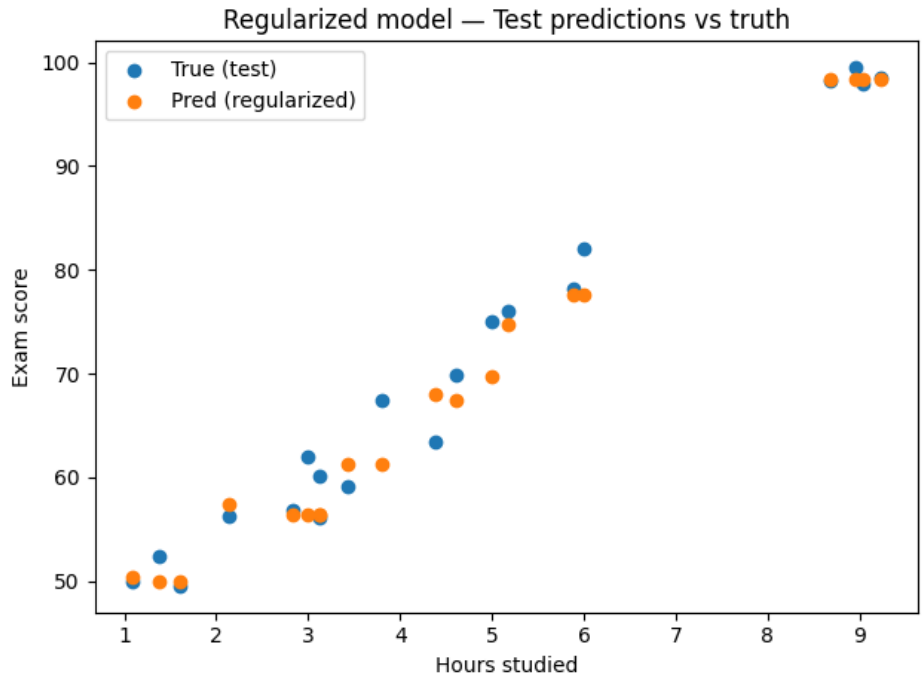
Kết quả trực quan

Hình 2, Hình 3, Hình 4 và Hình 5 so sánh dự đoán của hai mô hình:

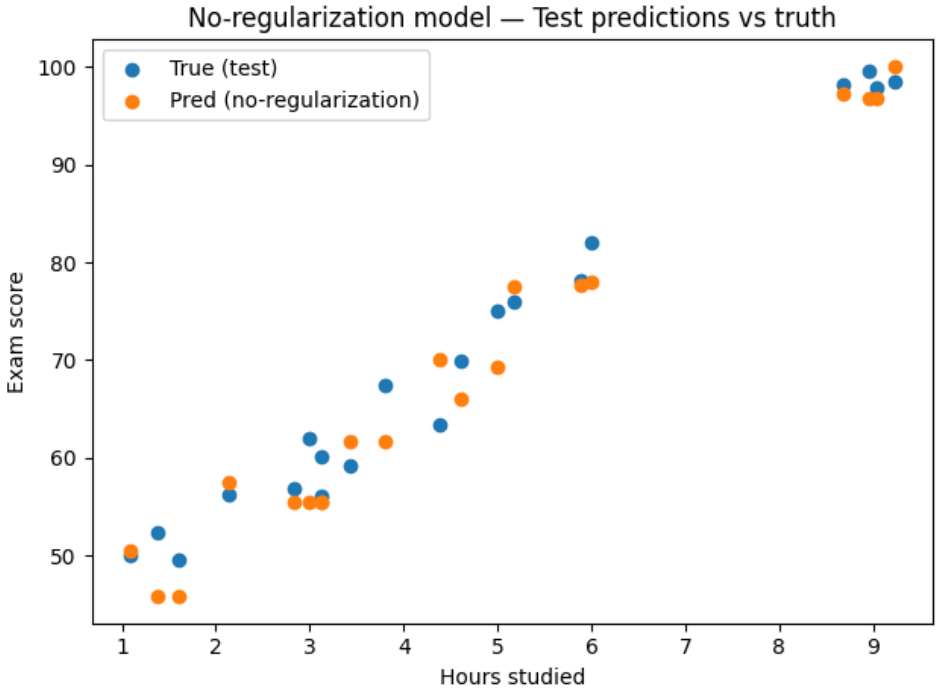
Model	RMSE_train	RMSE_test	MAE_train	MAE_test	R2_train	R2_test
Regularized	2.054491	2.956352	1.557721	2.169023	0.984709	0.968500
No Regularization	0.001254	3.802145	0.001029	3.115111	1.000000	0.947897

Avg leaves per tree — Regularized: 3.77 | No-Reg: 7.63

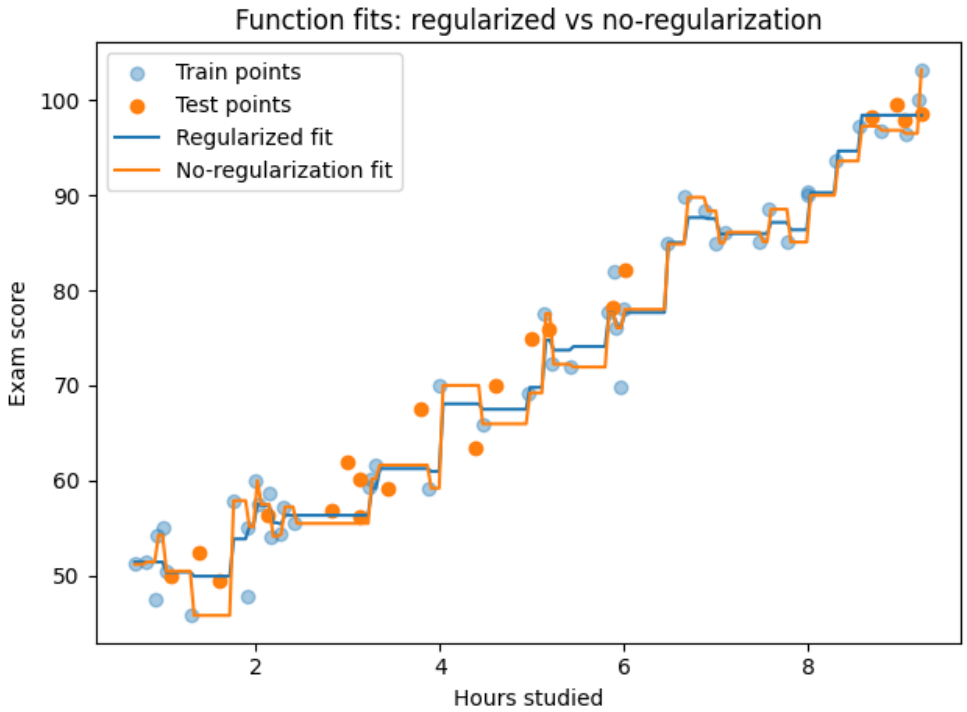
Hình 2: So sánh RSME, MAE giữa tập train và test, giữa hai mô hình có regualization và không có regulization



Hình 3: Mô hình có regularization — dự đoán test khớp mượt với dữ liệu thật.



Hình 4: Mô hình không regularization — có xu hướng overfit, đường dự đoán gấp khúc hơn.



Hình 5: So sánh trực tiếp đường fit: mô hình không regularization phức tạp hơn, trong khi regularization cho đường mượt hơn.

Kết luận

Phần $\Omega(f)$ trong hàm mất mát giúp XGBoost khống chế số lá và giá trị tại lá. Kết quả cho thấy:

- Mô hình không regulization: huấn luyện rất sát, nhưng test kém hơn, có xu hướng overfit.
- Mô hình có regulization: dự đoán mượt, sai số test nhỏ hơn, khả năng tổng quát tốt hơn.

2.2 Gradient Tree Boosting

Mô hình tập hợp cây trong Công thức (2) có các hàm như là tham số, nên không thể tối ưu bằng các phương pháp tối ưu truyền thống trong không gian Euclid. Thay vào đó, mô hình được huấn luyện theo cách **cộng dồn từng bước** (additive training).

Ký hiệu $\hat{y}_i^{(t)}$ là dự đoán của mẫu i tại vòng lặp t. Ta cần tìm một hàm f_t mới để cực tiểu hoá hàm mục tiêu:

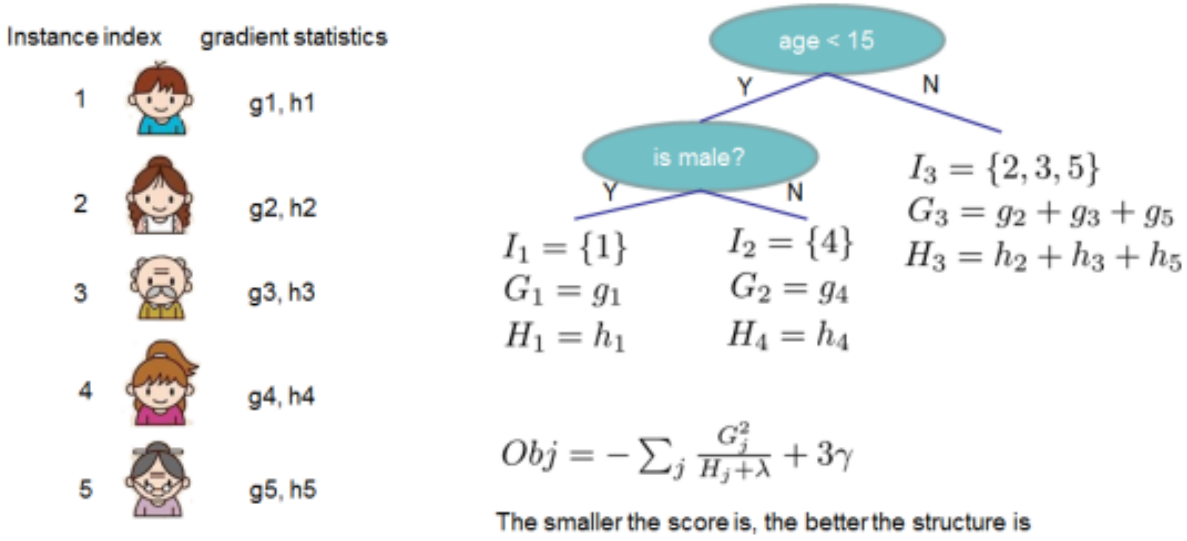
$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\big(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\big) + \Omega(f_t).$$

Điều này có nghĩa là ta tham lam chọn f_t sao cho cải thiện nhiều nhất mô hình theo Công thức (2). Để giải nhanh, ta sử dụng khai triển **xấp xỉ bậc hai** cho hàm mất mát [**?**]:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \Big[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i) \Big] + \Omega(f_t),$$

trong đó:

$$g_i = \frac{\partial}{\partial \hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \frac{\partial^2}{\partial (\hat{y}^{(t-1)})^2} l(y_i, \hat{y}^{(t-1)}).$$



Hình 6: Tính điểm cấu trúc - Cộng các giá trị gradient và thống kê đạo hàm bậc hai trên mỗi lá, sau đó áp dụng công thức tính điểm để thu được điểm chất lượng.

Bỏ đi các hằng số không ảnh hưởng, ta có hàm mục tiêu rút gọn:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t). \tag{3}$$

Tính toán theo lá. Ký hiệu $I_j = \{i \mid q(x_i) = j\}$ là tập các mẫu rơi vào lá j. Khi khai triển $\Omega(f)$, ta được:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \tfrac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T. \tag{4}$$

Với một cấu trúc cây $q(x)$ cố định, nghiệm tối ưu cho trọng số lá j là:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}. \tag{5}$$

Khi thay ngược lại, giá trị tối ưu của hàm mục tiêu là:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\tfrac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{6}$$

Công thức (6) được dùng như một **hàm chấm điểm (scoring function)** để đánh giá chất lượng cấu trúc cây q. Nó tương tự như thước đo độ thuần khiết (impurity) trong cây quyết định, nhưng tổng quát hơn cho nhiều hàm mất mát.

—

Tìm điểm tách (Split finding). Trong thực tế, không thể liệt kê tất cả các cấu trúc cây. Do đó, ta dùng thuật toán **tham lam**: bắt đầu từ một lá duy nhất và lần lượt thêm nhánh để tách lá.

Giả sử sau khi tách, ta có hai nút con với tập mẫu I_L và I_R, trong khi tập cha là $I = I_L \cup I_R$. Khi đó, độ giảm mất mát do tách được tính bằng:

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{7}$$

Công thức (7) thường được sử dụng trong thực tế để đánh giá và chọn điểm tách tốt nhất.

Giải thích các thành phần trong Gradient Tree Boosting

Một trong những cải tiến quan trọng của XGBoost so với boosting truyền thống là việc sử dụng **đạo hàm bậc hai** và **hàm điều chuẩn** $\Omega(f)$.

Vì sao dùng đạo hàm bậc hai? Trong boosting truyền thống, cập nhật thường chỉ dựa vào gradient bậc nhất g_i. XGBoost bổ sung thêm Hessian h_i (đạo hàm bậc hai), cho phép ước lượng *độ cong* của hàm mất mát. Điều này tương tự như phương pháp Newton-Raphson trong tối ưu: vừa xét hướng, vừa xét độ dốc, nên cập nhật nhanh hơn và chính xác hơn.

Vì sao có công thức trọng số lá w_j? Với một cấu trúc cây cố định, giá trị dự đoán tại lá j được tìm bằng cách cực tiểu hóa mục tiêu rút gọn:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T}\left[\Big(\sum_{i\in I_j} g_i\Big)w_j + \frac{1}{2}\Big(\sum_{i\in I_j} h_i + \lambda\Big)w_j^2\right] + \gamma T.$$

Đây là một hàm bậc hai theo w_j. Lấy đạo hàm và cho bằng 0, ta có nghiệm:

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

Như vậy, mỗi lá không chỉ đơn thuần lấy trung bình nhãn, mà giá trị dự đoán được tính toán tối ưu nhờ gradient và Hessian.

Vì sao $\Omega(f)$ chỉ còn γT? Hàm phạt ban đầu:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2.$$

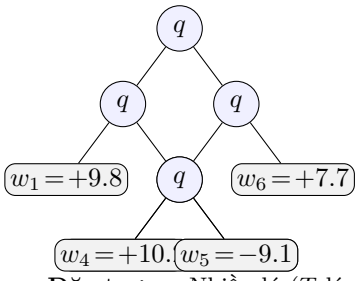
Khi thay w_j^* vào, phần $\frac{1}{2}\lambda\|w\|^2$ đã được gộp vào công thức (6) trong mẫu số $\sum h_i + \lambda$. Do đó, sau khi rút gọn, $\Omega(f)$ chỉ còn lại thành phần γT, đóng vai trò như một chi phí cố định cho mỗi lá.

Liên hệ với thực nghiệm. Trong ví dụ *dự đoán điểm thi từ số giờ học*, ta quan sát:

- Không regularization ($\gamma = 0, \lambda = 0$): cây tạo nhiều lá, trọng số cực trị, huấn luyện sát nhưng test kém $\Rightarrow$ overfitting.
- Có regularization ($\gamma = 1, \lambda = 5$): số lá ít hơn, trọng số nhỏ và mượt, sai số test giảm, tổng quát tốt hơn.

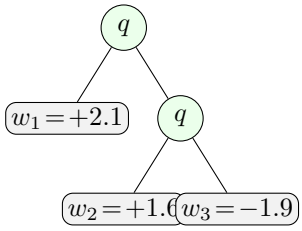
Như vậy, việc dùng đạo hàm bậc hai, công thức w_j^*, và $\Omega(f)$ chính là cơ chế giúp XGBoost mạnh mẽ hơn so với boosting truyền thống.

(A) Không regularization: $\gamma = 0,\ \lambda = 0$



Đặc trưng: Nhiều lá (T lớn), w_j lớn $\Rightarrow$ overfit.

(B) Có regularization: $\gamma > 0,\ \lambda > 0$



$w_j^* = -\frac{\sum g_i}{\sum h_i + \lambda} \Rightarrow$ ít lá, w mượt, tổng quát tốt.

Hình 7: Regularization kiểm soát số lá (γT) và biên độ trọng số ($\lambda \|w\|^2$).

2.3 Shrinkage và Column Subsampling trong XGBoost

Bên cạnh hàm mục tiêu điều chuẩn đã trình bày trong Mục 2.1, hai kỹ thuật bổ sung được sử dụng để ngăn ngừa overfitting.

2.3.1 Shrinkage (η) trong công thức

Kỹ thuật thứ nhất là shrinkage, được giới thiệu bởi Friedman. Shrinkage thực hiện việc nhân các trọng số mới thêm vào sau mỗi bước boosting của cây với một hệ số . Tương tự như learning rate trong tối ưu ngẫu nhiên, shrinkage giúp giảm ảnh hưởng của từng cây riêng lẻ và để dành "không gian" cho các cây tiếp theo cải thiện mô hình.

Shrinkage *không* xuất hiện trực tiếp trong hàm mục tiêu, mà nằm ở **bước cập nhật mô hình** sau khi đã huấn luyện xong cây thứ t:

$$\hat{y}^{(t)}(x) \;=\; \hat{y}^{(t-1)}(x) \;+\; \eta\, f_t(x).$$

Ở đây $f_t(x)$ là cây mới, được tối ưu theo mục tiêu xấp xỉ bậc hai:

$$\tilde{\mathcal{L}}^{(t)} \;=\; \sum_i \Big[g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i) \Big] \;+\; \Omega(f_t).$$

Sau khi tìm cấu trúc cây và trọng số lá tối ưu w_j^* (Công thức (5) trong bài gốc), ta **nhân thêm** η trước khi cộng vào dự đoán tích luỹ. Vai trò của η tương tự *learning rate*: mỗi cây chỉ đóng góp một bước nhỏ, chừa không gian cho các cây sau, nhờ đó giảm overfitting.

2.3.2 Column Subsampling trong công thức

Kỹ thuật thứ hai là lấy mẫu cột (feature subsampling). Kỹ thuật này vốn được sử dụng trong RandomForest, và cũng đã được triển khai trong phần mềm thương mại TreeNet cho gradient boosting, nhưng chưa có mặt trong các gói mã nguồn mở. Việc lấy mẫu theo cột còn giúp ngăn ngừa overfitting hiệu quả hơn cả việc lấy mẫu theo hàng (row subsampling, vốn cũng được hỗ trợ). Ngoài ra, việc lấy mẫu cột còn giúp tăng tốc độ tính toán cho thuật toán song song được mô tả ở các phần sau.

Column subsampling tác động vào *bộ đặc trưng được xét khi tìm split*. Trong giả mã tìm split (Algorithm 1/2), vòng

$$\text{for } k = 1 \text{ to } m$$

được thay bởi

$$\text{for } k \in \mathcal{S}_t \subset \{1, \dots, m\},$$

với $\mathcal{S}_t$ là **tập con đặc trưng được chọn ngẫu nhiên** cho mỗi cây (bytree) hoặc cho từng node (bynode). Các công thức chấm điểm split (ví dụ độ giảm loss, Eq. (7)) *không đổi*; chỉ là ta *chỉ xét trên các cột đã được lấy mẫu*. Hệ quả: cây ở mỗi vòng "nhìn" các chiều khác nhau, giảm tương quan giữa cây, đa dạng hoá ensemble, giảm overfitting và tăng tốc (ít cột cần quét).

Trình tự một vòng lặp boosting (tóm tắt)

1. Tính g_i, h_i từ loss tại $\hat{y}^{(t-1)}$.
2. (Tuỳ cấu hình) **Lấy mẫu cột** tạo $\mathcal{S}_t$ và/hoặc lấy mẫu hàng.
3. Tìm cây f_t bằng cách tối đa hoá giảm loss (Eq. (7)) **chỉ trên các cột trong $\mathcal{S}_t$**.
4. Với cấu trúc cây đã có, tính **trọng số lá tối ưu**

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

5. **Shrinkage**: cập nhật $\hat{y}^{(t)} = \hat{y}^{(t-1)} + \eta f_t(x)$.
6. Lặp lại cho đến khi đủ số vòng hoặc đạt tiêu chí dừng.

2.3.3 Code thực nghiệm Shrinkage và Column Sampling

Bối cảnh. Ta xét bài toán hồi quy *dự đoán điểm thi* từ các đặc trưng như: số giờ học, số lần làm bài tập, điểm kỳ trước, v.v. Mục tiêu là minh hoạ tác động của (i) **shrinkage** (hệ số học/learning rate η) và (ii) **column subsampling** (lấy mẫu ngẫu nhiên tập con đặc trưng) tới hiện tượng overfitting và khả năng tổng quát hoá.

Kỳ vọng.

- **Shrinkage:** η nhỏ hơn $\Rightarrow$ mỗi cây đóng góp nhỏ hơn, mô hình học dần dần, ít overfit hơn, thường `RMSE_test` giảm so với η lớn.
- **Column subsampling:** các tham số như `colsample_bytree`, `colsample_bynode`, `subsample` < 1.0 giúp tăng đa dạng cây, giảm tương quan giữa cây, từ đó giảm overfitting và tăng tốc tính toán.

Thiết lập thực nghiệm. Ta so sánh ba cấu hình:

1. **No_Reg (tham chiếu xấu):** $\eta = 0.3$, `colsample_bytree` $= 1.0$, `subsample` $= 1.0$, `max_depth` lớn hơn (dễ overfit).
2. **Shrinkage:** giảm η (0.08), giữ nguyên số vòng lặp để xem tác động học chậm hơn $\Rightarrow$ thường `RMSE_test` tốt hơn.
3. **Shrinkage + Column Subsampling:** như (2) nhưng thêm `colsample_bytree` < 1.0, `subsample` < 1.0.

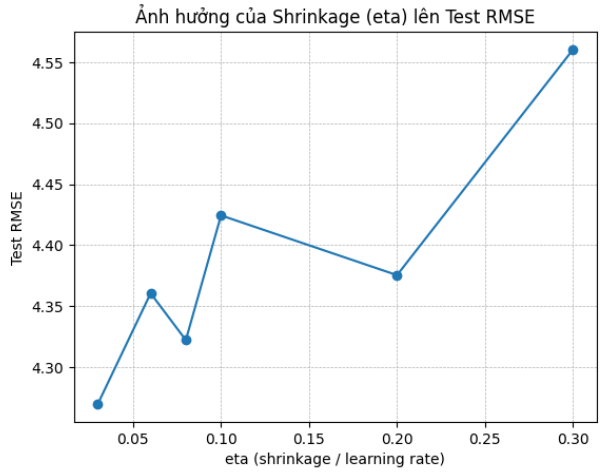
```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import xgboost as xgb
import matplotlib.pyplot as plt

# ====== 1) Generate data: "hours -> exam score" + noise and extra features ======
rng = np.random.RandomState(42)
n = 400

hours = rng.uniform(0.5, 9.5, size=n)     # strong feature
homework = rng.randint(0, 12, size=n)     # number of assignments
last_score = rng.normal(70, 8, size=n)    # previous exam score
sleep = rng.normal(7, 1.2, size=n)        # hours slept
social = rng.uniform(0, 3, size=n)        # social media hours

X = np.vstack([hours, homework, last_score, sleep, social]).T

# Label generation: mainly from hours + last_score + small effects + noise
y = (48 + 5.2*hours + 0.4*last_score
     + 0.8*np.tanh((hours-6)/1.2)
     - 0.6*social + 0.3*homework
     + rng.normal(0, 3.5, size=n))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.28, random_state=0
)

def evaluate(model, name):
    yhat_tr = model.predict(xgb.DMatrix(X_train))
    yhat_te = model.predict(xgb.DMatrix(X_test))
    metrics = {
        "model": name,
        "RMSE_train": float(np.sqrt(mean_squared_error(y_train, yhat_tr))),
        "RMSE_test":  float(np.sqrt(mean_squared_error(y_test,  yhat_te))),
        "MAE_train":  float(mean_absolute_error(y_train, yhat_tr)),
        "MAE_test":   float(mean_absolute_error(y_test,  yhat_te)),
        "R2_train":   float(r2_score(y_train, yhat_tr)),
        "R2_test":    float(r2_score(y_test,  yhat_te)),
    }
    return metrics

# ====== 2) Common params ======
params_base = dict(
    objective="reg:squarederror",
    tree_method="hist",
    random_state=0
)

# ====== 3) Three baseline configs ======
# 3a) No_Reg (likely to overfit): larger eta, deeper trees, no subsampling
params_no_reg = dict(params_base)
params_no_reg.update(
    dict(eta=0.30, max_depth=6, min_child_weight=1,
         colsample_bytree=1.0, colsample_bynode=1.0, subsample=1.0,
         reg_lambda=0.0, reg_alpha=0.0, gamma=0.0)
)
model_no_reg = xgb.train(
    params=params_no_reg,
    dtrain=xgb.DMatrix(X_train, label=y_train),
```

```
    num_boost_round=500
)

# 3b) Shrinkage: smaller eta and depth, mild regularization
params_shrink = dict(params_base)
params_shrink.update(
    dict(eta=0.08, max_depth=3, min_child_weight=5,
         colsample_bytree=1.0, colsample_bynode=1.0, subsample=1.0,
         reg_lambda=5.0, reg_alpha=0.0, gamma=1.0)
)
model_shrink = xgb.train(
    params=params_shrink,
    dtrain=xgb.DMatrix(X_train, label=y_train),
    num_boost_round=300
)

# 3c) Shrinkage + Column Subsampling:
params_shrink_col = dict(params_base)
params_shrink_col.update(
    dict(eta=0.08, max_depth=3, min_child_weight=5,
         colsample_bytree=0.75, colsample_bynode=0.8, subsample=0.85,
         reg_lambda=5.0, reg_alpha=0.0, gamma=1.0)
)
model_shrink_col = xgb.train(
    params=params_shrink_col,
    dtrain=xgb.DMatrix(X_train, label=y_train),
    num_boost_round=300
)

# ====== 4) Evaluate the 3 baseline configs ======
for name, model in [
    ("No_Reg", model_no_reg),
    ("Shrinkage", model_shrink),
    ("Shrinkage+ColSub", model_shrink_col)
]:
    print(evaluate(model, name))

# ====== 5) EXTRA: sweep shrinkage (eta) and colsample_bytree + plots ======

# 5a) Sweep eta (shrinkage)
etas = [0.03, 0.06, 0.08, 0.1, 0.2, 0.3]
rmse_eta = []

for eta in etas:
    params = dict(params_shrink)   # regularized base
    params["eta"] = eta
    # simple rule: smaller eta -> more boosting rounds
    num_boost_round = 450 if eta <= 0.06 else 320 if eta <= 0.1 else 220
    model = xgb.train(
        params=params,
        dtrain=xgb.DMatrix(X_train, label=y_train),
        num_boost_round=num_boost_round
    )
    yhat = model.predict(xgb.DMatrix(X_test))
    rmse_eta.append(float(np.sqrt(mean_squared_error(y_test, yhat))))

plt.figure()
plt.plot(etas, rmse_eta, marker="o")
plt.xlabel("eta (shrinkage / learning rate)")
plt.ylabel("Test RMSE")
```

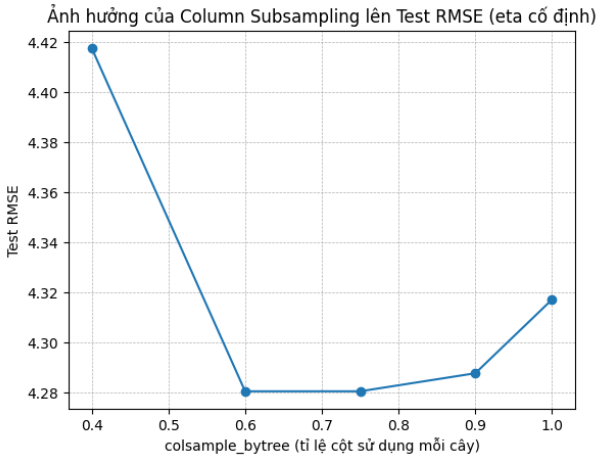
```
plt.title("Effect of Shrinkage (eta) on Test RMSE")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()

# 5b) Sweep colsample_bytree (fix eta)
col_fracs = [0.4, 0.6, 0.75, 0.9, 1.0]
rmse_col = []

for c in col_fracs:
    params = dict(params_shrink)     # same shrinkage config
    params["colsample_bytree"] = c
    params["subsample"] = 0.9        # mild row subsampling
    model = xgb.train(
        params=params,
        dtrain=xgb.DMatrix(X_train, label=y_train),
        num_boost_round=300
    )
    yhat = model.predict(xgb.DMatrix(X_test))
    rmse_col.append(float(np.sqrt(mean_squared_error(y_test, yhat))))

plt.figure()
plt.plot(col_fracs, rmse_col, marker="o")
plt.xlabel("colsample_bytree (feature fraction per tree)")
plt.ylabel("Test RMSE")
plt.title("Effect of Column Subsampling on Test RMSE (fixed eta)")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()

print("RMSE vs eta:", list(zip(etas, rmse_eta)))
print("RMSE vs colsample_bytree:", list(zip(col_fracs, rmse_col)))
```

Kết quả thực tế. Đọc kết quả thực nghiệm:

- **Shrinkage (eta):** đường cong không đơn điệu; η lớn (≈ 0.30) cho RMSE cao nhất $\Rightarrow$ dễ overfit/dao động. Vùng tốt nằm quanh $\eta \in [0.03, 0.10]$; trong lần chạy này giá trị tối ưu gần $\eta \simeq 0.03$ (RMSE ≈ 4.27) và $\eta \simeq 0.08$ cũng tốt (RMSE ≈ 4.32).
- **Column subsampling:** tốt nhất khi `colsample_bytree` $\in [0.6, 0.7]$ (RMSE ≈ 4.28). Chọn 1.0 (dùng tất cả cột) kém hơn chút (RMSE ≈ 4.32); quá thấp như 0.4 làm thiếu tín hiệu (RMSE ≈ 4.42).



(a) Ảnh hưởng của Shrinkage lên Test RMSE



(b) Ảnh hưởng của Column Subsampling lên Test RMSE

Hình 8: So sánh ảnh hưởng của Shrinkage và Column Subsampling lên Test RMSE.

2.4 Tìm Điểm Chia (Split) trong XGBoost: Từ Trực Giác Đời Thường đến Thuật Toán

Trong XGBoost, mỗi nút cây cần chọn một *vạch cắt* (split) sao cho chất lượng dự đoán tăng nhiều nhất. Hai tình huống thực tế khiến bài toán thú vị (và khó) hơn:

1. Dữ liệu cần được ưu tiên khác nhau theo *trọng số* (Hessian) ⇒ dùng **Weighted Quantile Sketch** để chọn *vài* vạch cắt thông minh.
2. Dữ liệu bị *thưa/thiếu giá trị* (missing, nhiều 0, one-hot) ⇒ dùng **Sparsity-aware Split** để vừa nhanh vừa xử lý thiếu hợp lý.

2.4.1 Phiên bản đời thường

Split là gì?

Hình dung bạn có cột dữ liệu *"giờ học"* và muốn dự đoán *"điểm thi"*. Cây quyết định sẽ đặt một **vạch cắt** (ví dụ: giờ học ≤ 5 hay > 5) để chia dữ liệu thành hai nhóm sao cho dự đoán tốt nhất. Bài toán cốt lõi là: **đặt vạch cắt ở đâu thì ngon nhất?**

Cách 1: Thử hết mọi chỗ cắt (*exact greedy*)

- Máy sẽ **thử tất cả vị trí có thể** trên **mọi cột**, chọn chỗ cho điểm tốt nhất.
- Để làm nhanh, cần **sắp xếp** các giá trị theo từng cột rồi rà theo thứ tự.
- Ưu: **chính xác**. Nhược: **tốn thời gian và RAM**. Hợp với dữ liệu **vừa và nhỏ**.

Cách 2: Thử theo "danh sách rút gọn" (*approximate*)

Khi dữ liệu quá lớn, thử hết sẽ rất nặng. Ta làm gọn:

1. **Chọn sẵn vài mốc** cho mỗi cột (theo *phân vị* như 25%, 50%, 75%).
2. **Gộp** giá trị vào các *xô* (bucket) giữa những mốc này, tính **điểm tổng hợp**.

3. **Chỉ** thử cắt tại các mốc đã chọn, lấy mốc tốt nhất.

Ưu: **nhanh hơn**, **ít tốn bộ nhớ**. Nếu chọn đủ mốc, **độ chính xác gần như thử hết**.

Global vs. Local

- **Global**: chọn danh sách mốc *một lần từ đầu*, dùng cho mọi tầng. Ít khâu chuẩn bị, nhưng thường *cần nhiều mốc* vì không tinh chỉnh theo từng lần cắt.
- **Local**: *cắt xong mới chọn lại mốc* cho phần dữ liệu còn lại. Tinh chỉnh dần, nên *cần ít mốc hơn*; hợp với **cây sâu**. Đổi lại, phải đề xuất mốc *nhiều lần*.

Ví dụ (trực giác theo khoảng giá trị).

Giả sử tại gốc, một đặc trưng liên tục x có dải giá trị $[0, 10]$. Ta chọn mức xấp xỉ dùng 3 phân vị (25%, 50%, 75%), tương ứng các *mốc*: $\{2.5,\ 5.0,\ 7.5\}$.

- **Bước 1 (split tại gốc):** thuật toán chọn cắt tại $x = 5.0$ (một trong các mốc), sinh ra hai nút con: Left $= [0, 5]$ và Right $= [5, 10]$.
- **Global (toàn cục):** ở các mức sâu hơn, bộ mốc vẫn là $\{2.5, 5.0, 7.5\}$. Tại nút $[0, 5]$ chỉ còn $\{2.5, 5.0\}$ hữu ích (7.5 nằm ngoài khoảng); nhưng $\{2.5, 5.0\}$ *tương đối thưa* so với dải $[0, 5]$.
- **Local (cục bộ):** *đề xuất lại* mốc theo phân vị *trên chính* dữ liệu của nút $[0, 5]$, ví dụ $\{1.25,\ 2.5,\ 3.75\}$. Các mốc *được tinh chỉnh* và *phủ đều* dải nhỏ, giúp tìm split tốt hơn ở mức sâu.

Ví dụ đời thường

- **Exact** = bạn cân nhắc *từng milimet* trên thước để cắt bánh chuẩn nhất.
- **Approx** = bạn *kẻ vài vạch* (1/4, 1/2, 3/4) rồi chỉ thử cắt ở các vạch đó. *Nhanh hơn* nhiều, và thường vẫn ngon.

Dùng lúc nào?

- **Dữ liệu nhỏ, cần chính xác tối đa**: dùng **exact**.
- **Dữ liệu lớn/không vừa RAM/phân tán/cần tốc độ**: dùng **approx** (hist/quantile).
- **Cây sâu**: ưu tiên **local**; **cây nông/nhanh**: **global** cũng ổn nếu chọn *đủ* mốc.

2.4.2 Thuật toán tham lam chính xác (Exact Greedy)

Bài toán then chốt là tìm *điểm chia tốt nhất* như Công thức (7). Thuật toán *exact greedy* sẽ **liệt kê tất cả các điểm chia có thể** trên **mọi đặc trưng**, tính điểm và chọn tốt nhất. Với đặc trưng liên tục, để hiệu quả:

- **Sắp xếp** dữ liệu theo giá trị từng đặc trưng;
- **Duyệt theo thứ tự** để **cộng dồn** các thống kê gradient cho công thức chấm điểm.

Độ giảm mất mát khi chia một nút (viết gọn $G = \sum g,\ H = \sum h$) là:

$$\mathcal{L}_{\text{split}} = \tfrac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}\right] - \gamma.$$

Thuật toán này được hỗ trợ trong các thư viện máy đơn như *scikit-learn*, *R gbm* và *XGBoost*.

2.4.3 Thuật toán xấp xỉ (Approximate)

Khi dữ liệu không vừa bộ nhớ hoặc trong môi trường phân tán, *exact greedy* không còn thực tế. Khung xấp xỉ thường gồm:

1. **Đề xuất** điểm chia ứng viên theo *phân vị* (percentile) của từng đặc trưng;
2. **Lượng hoá** đặc trưng liên tục thành các *bucket* bởi những mốc này, **gộp** thống kê (G, H);
3. **Chọn** điểm chia tốt nhất *trong tập ứng viên*.

Có hai biến thể:

- **Global**: đề xuất *một lần*, dùng lại cho mọi mức; cần *nhiều mốc* hơn.
- **Local**: *đề xuất lại sau mỗi split*; thường cần *ít mốc* hơn, phù hợp cây sâu.

Nhiều thuật toán phân tán cũng theo khung này; có thể dùng *histogram* xấp xỉ (G, H) hoặc chiến lược *binning* khác ngoài *quantile*. Thực nghiệm lớn (ví dụ Higgs 10M) cho thấy, nếu mức xấp xỉ đủ mịn (tham số ε nhỏ $\Rightarrow$ số xô $\approx 1/\varepsilon$ đủ nhiều), **độ chính xác xấp xỉ tiệm cận exact**.

Ví dụ tính tay: Split xấp xỉ bằng "quantile buckets" (1 vòng lặp, loss bình phương)

Thiết lập. Xét một đặc trưng liên tục x và nhãn y gồm 9 điểm (đã sắp theo x). Ở vòng lặp đầu tiên, giả sử dự đoán ban đầu $\hat{y}^{(0)} = 0$. Với hàm mất mát bình phương $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, ta có

$$g_i = \frac{\partial l}{\partial \hat{y}}\Big|_{\hat{y}=0} = -y_i, \qquad h_i = \frac{\partial^2 l}{\partial \hat{y}^2} = 1.$$

Bảng 2: Dữ liệu và đạo hàm tại $\hat{y}^{(0)} = 0$.

Chỉ số i	1	2	3	4	5	6	7	8	9
x_i	1	2	3	4	5	6	7	8	9
y_i	3	4	5	6	7	8	10	11	14
$g_i = -y_i$	-3	-4	-5	-6	-7	-8	-10	-11	-14
$h_i = 1$	1	1	1	1	1	1	1	1	1

Tổng toàn cục:

$$G = \sum_{i=1}^{9} g_i = -68, \qquad H = \sum_{i=1}^{9} h_i = 9.$$

Bước 1: Chia "xô" theo phân vị (quantile buckets). Chọn 3 xô theo phân vị 33% và 66%:

$$\text{xô 1: } \{x = 1, 2, 3\}, \qquad \text{xô 2: } \{x = 4, 5, 6\}, \qquad \text{xô 3: } \{x = 7, 8, 9\}.$$

Tổng theo xô (cộng gradient và đếm số điểm):

$$\begin{aligned}
&\text{Bucket 1: } (G_1, H_1) = (-12,\ 3) \quad (\text{từ } -3 - 4 - 5)\\
&\text{Bucket 2: } (G_2, H_2) = (-21,\ 3) \quad (\text{từ } -6 - 7 - 8)\\
&\text{Bucket 3: } (G_3, H_3) = (-35,\ 3) \quad (\text{từ } -10 - 11 - 14)
\end{aligned}$$

Bước 2: Chỉ thử cắt ở *ranh giới xô*. Với $\lambda = \gamma = 0$, độ lợi khi chia tại một vị trí là

$$\text{Gain} = \frac{1}{2}\left(\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right).$$

Điểm "trước khi cắt" (dùng cho so sánh):

$$\frac{G^2}{H} = \frac{(-68)^2}{9} = \frac{4624}{9} \approx 513.78.$$

(A) Cắt sau xô 1 (giữa $x = 3$ và $x = 4$): trái có 3 điểm, phải có 6 điểm.

$$G_L = -12,\ H_L = 3 \Rightarrow \frac{G_L^2}{H_L} = 48, \qquad G_R = G - G_L = -56,\ H_R = 6 \Rightarrow \frac{G_R^2}{H_R} = \frac{3136}{6} \approx 522.67.$$

$$\text{Gain}_A = \frac{1}{2}\Big(48 + 522.67 - 513.78\Big) \approx \boxed{28.44}.$$

(B) Cắt sau xô 2 (giữa $x = 6$ và $x = 7$): trái có 6 điểm, phải có 3 điểm.

$$G_L = -12+(-21) = -33,\ H_L = 6 \Rightarrow \frac{G_L^2}{H_L} = \frac{1089}{6} = 181.5, \qquad G_R = -35,\ H_R = 3 \Rightarrow \frac{G_R^2}{H_R} = \frac{1225}{3} \approx 408.33.$$

$$\text{Gain}_B = \frac{1}{2}\Big(181.5 + 408.33 - 513.78\Big) \approx \boxed{38.03}.$$

Kết luận (xấp xỉ): chọn **cắt giữa x = 6** và **7** vì $\text{Gain}_B > \text{Gain}_A$.

Bước 3 (tuỳ chọn): Trọng số hai lá sau khi cắt. Với loss bình phương và $\lambda = 0$, trọng số lá tối ưu $w^* = -G/H$. Do đó:

$$w_L^* = -\frac{G_L}{H_L} = -\frac{-33}{6} = 5.5, \qquad w_R^* = -\frac{G_R}{H_R} = -\frac{-35}{3} \approx 11.67.$$

Ở vòng đầu (nếu $\eta = 1$ để minh hoạ), cây mới dự đoán 5.5 cho nhóm $x \le 6$ và 11.67 cho nhóm $x \ge 7$. Nếu dùng shrinkage $\eta < 1$, giá trị cộng thêm sẽ được nhân với η.

Vì sao gọi là "xấp xỉ"?

- **Exact (chính xác):** thử *mọi* vị trí cắt (ở đây là 8 vị trí giữa các cặp liên tiếp).
- **Approx (xấp xỉ):** chỉ thử *vài ranh giới* là *ranh xô* (ở đây chỉ 2 vị trí).

Số xô càng nhiều ⇒ *độ mịn* càng cao ⇒ kết quả thường càng gần exact, nhưng chi phí tính toán cũng tăng.

2.4.4 Phác thảo phân vị có trọng số (Weighted Quantile Sketch)

Trực giác

Thay vì thử *mọi* ngưỡng cắt, ta chỉ thử ở *một số vạch* trên "thước" giá trị đặc trưng. Các vạch được đặt sao cho mỗi khoảng giữa hai vạch mang *tổng trọng số* (Hessian) ≈ nhau. Nhờ vậy:

- số ngưỡng thử ít hơn ⇒ nhanh, ít RAM;
- vẫn đại diện tốt các vùng dữ liệu quan trọng ⇒ gần như chính xác nếu số vạch đủ (độ mịn cao).

Ví dụ (chi tiết): Weighted Quantile Sketch và chọn split trên ranh giới "xô"

Bước 1: Xác định các vạch (quantile có trọng số). Giả sử một đặc trưng x (đã sắp tăng) và trọng số h (Hessian) như sau:

Giá trị x	2	3	8	9	12
Trọng số h	1	2	3	3	3

Tổng trọng số $H = \sum h = 12$. Ta muốn 3 vạch tại các *phân vị theo trọng số* 25%, 50%, 75% $\Rightarrow$ mốc cộng dồn $\{3, 6, 9\}$. Cộng dồn h theo thứ tự x:

Sau $x = 2:\ 1$, Sau $x = 3:\ 3\ (=25\%)$, Sau $x = 8:\ 6\ (=50\%)$, Sau $x = 9:\ 9\ (=75\%)$, Sau $x = 12:\ 12$.

Vậy **các vạch ứng viên** là {3, 8, 9}. Từ đó, ta lượng hoá x thành các *khoảng* (xô):

$$(-\infty, 3], \quad (3, 8], \quad (8, 9], \quad (9, +\infty).$$

Bước 2: Gộp thống kê theo "xô" để chấm điểm split. Để *minh hoạ*, giả sử tại một vòng boosting, ta có gradient g (tính từ loss) cho từng điểm

$$g(x{=}2) = -2,\ g(3) = -1,\ g(8) = +1,\ g(9) = +2,\ g(12) = +3.$$

Khi cần chấm điểm split ở *ranh giới giữa các xô*, ta chỉ cần $\sum g$ và $\sum h$ ở *về trái* (L) và *về phải* (R) của ranh giới đó. Đặt $\lambda = \gamma = 0$ cho gọn (có thể thêm lại sau):

$$\text{Gain} \ = \ \frac{1}{2}\left(\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right), \qquad G = \sum g,\ H = \sum h.$$

Ở đây $G = -2 - 1 + 1 + 2 + 3 = 3$, $H = 12 \ \Rightarrow \ \dfrac{G^2}{H} = \dfrac{9}{12} = 0.75$.

(A) Cắt sau xô 1: ranh $(-\infty, 3] \mid (3, 8] \cup (8, 9] \cup (9, +\infty)$.

$$G_L = (-2) + (-1) = -3, \quad H_L = 1 + 2 = 3; \qquad G_R = 3 - (-3) = 6, \quad H_R = 12 - 3 = 9.$$

$$\text{Gain}_A = \frac{1}{2}\left(\frac{(-3)^2}{3} + \frac{6^2}{9} - 0.75\right) = \frac{1}{2}\,(3 + 4 - 0.75) = \boxed{3.125}.$$

(B) Cắt sau xô 2: ranh $(-\infty, 8] \mid (8, 9] \cup (9, +\infty)$.

$$G_L = -2 + (-1) + (+1) = -2, \quad H_L = 1 + 2 + 3 = 6; \quad G_R = 3 - (-2) = 5, \quad H_R = 12 - 6 = 6.$$

$$\text{Gain}_B = \frac{1}{2}\left(\frac{(-2)^2}{6} + \frac{5^2}{6} - 0.75\right) = \frac{1}{2}\,(0.6667 + 4.1667 - 0.75) = \boxed{2.0417}.$$

(C) Cắt sau xô 3: ranh $(-\infty, 9] \mid (9, +\infty)$.

$$G_L = -2 + (-1) + (+1) + (+2) = 0, \quad H_L = 1 + 2 + 3 + 3 = 9; \quad G_R = 3 - 0 = 3, \quad H_R = 12 - 9 = 3.$$

$$\text{Gain}_C = \frac{1}{2}\left(\frac{0^2}{9} + \frac{3^2}{3} - 0.75\right) = \frac{1}{2}\,(0 + 3 - 0.75) = \boxed{1.125}.$$

Kết luận: $\text{Gain}_A > \text{Gain}_B > \text{Gain}_C$ nên chọn **cắt tại 3** (ranh giới sau xô 1). Điểm mấu chốt: ta *không* cần thử mọi ngưỡng, chỉ thử ở *ranh giới xô* do quantile có trọng số sinh ra; vẫn chấm điểm bằng (G, H) như thường.

Ghi chú thêm.

- Nếu dùng $\lambda > 0$ (L2) và $\gamma > 0$ (phạt tách), thay $H \to H + \lambda$ trong các phân số và trừ thêm γ vào Gain.
- Với *exact greedy*, bạn sẽ thử *mọi* ngưỡng giữa các giá trị x; với *approx*, bạn chỉ thử các ranh giới xô (ít hơn rất nhiều). Khi số xô đủ mịn, kết quả thường tiệm cận exact.

Ghi chú. Nếu tất cả $h_i = 1$ (không trọng số), phân vị sẽ theo số điểm; khi có trọng số, vị trí vạch *dịch chuyển* về nơi có tổng h lớn hơn (vùng "quan trọng").

2.4.5 Sparsity-aware Split: xử lý dữ liệu thưa/missing thông minh

Trực giác

- **Thiếu giá trị** ở một đặc trưng không bị bỏ qua; tại mỗi split, ta *học* **hướng mặc định** cho missing (rẽ trái hay rẽ phải) bằng cách chọn phương án cho *độ lợi* (gain) cao hơn.
- Khi tính toán, ta **chỉ duyệt** các bản ghi *không thiếu* ở cột đó $\Rightarrow$ thời gian tỉ lệ với *số phần tử không thiếu*. Rất nhanh khi dữ liệu thưa (nhiều 0, one-hot).

Ví dụ tính tay

Xét một cột x có 8 mẫu (đã sort theo x; "?" là missing). Dùng loss bình phương, đặt $h_i = 1$ cho dễ tính. Giả sử gradient g_i ở vòng hiện tại là:

Chỉ số i	1	2	3	4	5	6	7	8
x_i	?	2	3	?	6	8	?	10
g_i	-2	-2	-1	-2	$+1$	$+2$	-1	$+3$

Phần **không thiếu** trên cột x: $\{2, 3, 6, 8, 10\}$. Phần **thiếu** có 3 điểm: $g_{\text{miss}} = -2 + (-2) + (-1) = -5$. Tổng toàn cục: $G = \sum g_i = -2$, $H = 8$.

Giả sử thử **ngưỡng** giữa 3 và 6.

- **Chỉ trên dữ liệu không thiếu**:

$$\text{Trái } (\le 3): \ G_L = -2 + (-1) = -3, \ H_L = 2; \quad \text{Phải } (\ge 6): \ G_R = +1 + 2 + 3 = 6, \ H_R = 3.$$

- **Kịch bản A (missing→phải)**: thêm $(G, H) = (-5, 3)$ vào phải $\Rightarrow \ G_R = 1, \ H_R = 6$.
- **Kịch bản B (missing→trái)**: thêm $(G, H) = (-5, 3)$ vào trái $\Rightarrow \ G_L = -8, \ H_L = 5$.

Dùng thước đo split chuẩn (bỏ hằng số vì so sánh tương đối là đủ):

$$\text{Score} \propto \frac{G_L^2}{H_L} + \frac{G_R^2}{H_R}.$$

Tính nhanh:

$$\text{A: } \frac{(-3)^2}{2} + \frac{1^2}{6} = 4.5 + 0.1667 = 4.6667, \qquad \text{B: } \frac{(-8)^2}{5} + \frac{6^2}{3} = 12.8 + 12 = 24.8.$$

Kết luận: chọn **missing→trái** (kịch bản B) vì điểm cao hơn nhiều.

Điểm mấu chốt. Ta chỉ *sort & quét* 5 phần tử *không thiếu*; phần thiếu chỉ cần *gộp* vào trái hoặc phải (hai kịch bản) để so sánh. Nhờ vậy, độ phức tạp tính toán *tuyến tính* theo *số phần tử không thiếu*.

Công thức gain

Ở một split, độ lợi chuẩn (λ, γ là điều chuẩn L2 và phạt tách) là

$$\text{Gain} = \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right) - \gamma.$$

Trong các ví dụ tính tay trên, ta đặt $\lambda = \gamma = 0$ và tập trung so sánh tương đối.

2.4.6 Tóm tắt toàn bộ thuật toán tìm điểm chia trong XGBoost

1) Exact Greedy

Nguyên lý. Liệt kê *mọi* ngưỡng hợp lệ trên *mọi* đặc trưng tại mỗi nút, tính Gain

$$\text{Gain} = \tfrac{1}{2}\Big(\tfrac{G_L^2}{H_L+\lambda} + \tfrac{G_R^2}{H_R+\lambda} - \tfrac{G^2}{H+\lambda}\Big) - \gamma,$$

chọn ngưỡng cho Gain lớn nhất (tham lam "đầy đủ"). Cần sắp xếp cột và quét theo thứ tự.

Khi nào dùng. Dữ liệu **nhỏ–vừa**, **vừa RAM**, môi trường máy đơn, cần độ chính xác cao nhất cho từng split.

Trong XGBoost. Hỗ trợ ở chế độ máy đơn (lựa chọn lịch sử). Hiện thực tiệm cận mặc định là `tree_method=hist` (nhanh hơn & tiết kiệm bộ nhớ); `exact` có thể chậm/hao RAM với dữ liệu lớn.

2) Approximate Split Finding

Nguyên lý. Không thử mọi ngưỡng, mà: (i) *sinh ứng viên* (cut points) theo *quantile/histogram*, (ii) *lượng hoá* đặc trưng vào các *xô* (bins), (iii) *gộp* thống kê (G, H) theo xô, (iv) *chỉ* chấm điểm tại *ranh giới xô*. Có hai biến thể đề xuất ngưỡng: **global** (một lần từ gốc) và **local** (đề xuất lại sau mỗi split).

Khi nào dùng. Dữ liệu lớn, **out-of-core** hoặc **phân tán**, khi exact không khả thi. Độ mịn (số xô) đủ lớn $\Rightarrow$ chất lượng tiệm cận exact.

Trong XGBoost. `tree_method=hist` (CPU) hoặc `gpu_hist` (GPU). Tham số hữu ích: `max_bin` (số xô), `gamma`, `reg_lambda`, `subsample`, `colsample_bytree`.

3) Weighted Quantile Sketch (WQS)

Nguyên lý. Là *kỹ thuật sinh ngưỡng ứng viên* dựa trên *phân vị có trọng số* (trọng số thường là Hessian h_i): tìm các mốc $\{s_j\}$ sao cho chênh lệch *hạng có trọng số* $r_k(\cdot)$ giữa mốc liên tiếp $\le \varepsilon$. Hỗ trợ *merge/prune* với bảo đảm sai số — phù hợp phân tán.

Khi nào dùng. Khi bạn cần đặt "*vạch thông minh*" phản ánh *độ quan trọng* (qua h_i), nhất là trên dữ liệu lớn/phân tán.

Trong XGBoost. Được dùng *bên trong* Approximate/Histogram để dựng các *cut points*. Điều chỉnh gián tiếp qua `max_bin` (và một số cấu hình sketch nội bộ tuỳ build).

4) Sparsity-aware Split

Nguyên lý. Dữ liệu thưa/missing: chỉ *quét phần tử không thiếu* trên cột, và *học hướng mặc định* cho missing (thử *missing→trái/phải*, chọn hướng cho Gain cao hơn). Độ phức tạp tuyến tính theo #phần tử *không thiếu*.

Khi nào dùng. Dữ liệu **nhiều giá trị thiếu**, **one-hot**, **cao chiều** (CTR, text, tabular lớn).

Trong XGBoost. Truyền `missing` (mặc định `NaN`) vào `DMatrix`; XGBoost tự học hướng mặc định tại từng split. Không cần điền giá trị thiếu thủ công.

2.4.7 Ví dụ:

Bài toán và công thức chấm điểm split

Xét một nút với tập chỉ số I. Ký hiệu thống kê đạo hàm bậc nhất/bậc hai theo loss tại vòng hiện tại:

$$G = \sum_{i \in I} g_i, \qquad H = \sum_{i \in I} h_i.$$

Nếu tách thành nhánh trái/phải có tổng (G_L, H_L) và (G_R, H_R) (với $G = G_L + G_R,\ H = H_L + H_R$), độ lợi của split là

$$\text{Gain} = \frac{1}{2}\Big(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}\Big) - \gamma.$$

Trong các ví dụ tính tay bên dưới, trừ khi nói rõ, ta đặt $\lambda = \gamma = 0$ cho gọn.

Dữ liệu dùng chung (1 đặc trưng, vòng đầu, không thiếu)

Dùng loss bình phương, tại vòng đầu (giả sử $\hat{y}^{(0)} = 0$) có $g_i = -y_i,\ h_i = 1$.

i	1	2	3	4	5	6	7
x_i	1	2	3	6	8	9	12
y_i	2	2	1	-1	-2	-2	-3
$g_i = -y_i$	-2	-2	-1	$+1$	$+2$	$+2$	$+3$
h_i	1	1	1	1	1	1	1

Tại gốc: $G = \sum g_i = 3,\ H = \sum h_i = 7,\ \dfrac{G^2}{H} = \dfrac{9}{7} \approx 1.2857.$

(1) Exact Greedy: thử *mọi* ngưỡng

Các ngưỡng giữa các giá trị khác nhau: $t \in \{1.5,\ 2.5,\ 4.5,\ 7,\ 8.5,\ 10.5\}$. Tính G_L, H_L, G_R, H_R rồi thay vào công thức Gain (với $\lambda = \gamma = 0$), cho kết quả:

Ngưỡng t	(G_L, H_L)	(G_R, H_R)	Gain
1.5	$(-2, 1)$	$(5, 6)$	$\frac{1}{2}\big(\frac{4}{1} + \frac{25}{6} - \frac{9}{7}\big) = \mathbf{3.4405}$
2.5	$(-4, 2)$	$(7, 5)$	$\frac{1}{2}\big(\frac{16}{2} + \frac{49}{5} - \frac{9}{7}\big) = \mathbf{8.2571}$
4.5	$(-5, 3)$	$(8, 4)$	$\frac{1}{2}\big(\frac{25}{3} + \frac{64}{4} - \frac{9}{7}\big) = \mathbf{11.5238}$
7.0	$(-4, 4)$	$(7, 3)$	$\frac{1}{2}\big(\frac{16}{4} + \frac{49}{3} - \frac{9}{7}\big) = \mathbf{9.5238}$
8.5	$(-2, 5)$	$(5, 2)$	$\frac{1}{2}\big(\frac{4}{5} + \frac{25}{2} - \frac{9}{7}\big) = \mathbf{6.0071}$
10.5	$(0, 6)$	$(3, 1)$	$\frac{1}{2}\big(0 + \frac{9}{1} - \frac{9}{7}\big) = \mathbf{3.8571}$

Kết luận (Exact): chọn $t = \boxed{4.5}$ (giữa $x = 3$ và $x = 6$).

Ảnh hưởng của điều chuẩn λ, γ tại $t = 4.5$ Lấy $\lambda = 1,\ \gamma = 0.5$:

$$\text{Gain} = \frac{1}{2}\Big(\frac{25}{3+1} + \frac{64}{4+1} - \frac{9}{7+1}\Big) - 0.5 = \tfrac{1}{2}(6.25 + 12.8 - 1.125) - 0.5 = \mathbf{8.4625}.$$

Nhận xét: λ phạt biên độ (vào mẫu số), γ phạt hành vi tách (trừ trực tiếp). Nếu Gain≤ 0, nút không được tách.

(2) Sparsity-aware: có thiếu giá trị (học hướng mặc định cho missing)

Giả sử x_1 bị thiếu (NaN). Ở ngưỡng $t = 4.5$, ta chỉ quét phần *không thiếu*:

$$\text{Trái } (\le 3):\ (G_L, H_L) = (-3, 2), \quad \text{Phải } (> 3):\ (G_R, H_R) = (8, 4), \quad (G_{\text{miss}}, H_{\text{miss}}) = (-2, 1).$$

Thử hai kịch bản và chọn hướng cho Gain lớn hơn:

$$\begin{aligned} \text{missing} \to \text{trái}: &\quad (G'_L, H'_L) = (-5, 3),\ (G'_R, H'_R) = (8, 4) \\ &\Rightarrow \text{Gain} = \tfrac{1}{2}\Big(\tfrac{25}{3} + \tfrac{64}{4} - \tfrac{9}{7}\Big) = \mathbf{11.5238}. \\ \text{missing} \to \text{phải}: &\quad (G'_L, H'_L) = (-3, 2),\ (G'_R, H'_R) = (6, 5) \\ &\Rightarrow \text{Gain} = \tfrac{1}{2}\Big(\tfrac{9}{2} + \tfrac{36}{5} - \tfrac{9}{7}\Big) = \mathbf{5.2071}. \end{aligned}$$

Chọn: missing→**trái**. Trong cây, hướng mặc định này được lưu tại node ("missing = left").

(3) Approximate (hist/quantile): chỉ thử ở ranh giới "xô"

Giả sử lưới *thô* có ranh giới tại 3 và 9 ⇒ chỉ xét $t \in \{3, 9\}$ (xử lý missing như trên nếu có).

$$\begin{aligned} t = 3: &\quad (G_L, H_L) = (-4, 2),\ (G_R, H_R) = (7, 5) \\ &\Rightarrow \text{Gain} = \tfrac{1}{2}\Big(\tfrac{16}{2} + \tfrac{49}{5} - \tfrac{9}{7}\Big) = \mathbf{8.2571}. \\ t = 9: &\quad (G_L, H_L) = (0, 6),\ (G_R, H_R) = (3, 1) \\ &\Rightarrow \text{Gain} = \tfrac{1}{2}\Big(0 + \tfrac{9}{1} - \tfrac{9}{7}\Big) = \mathbf{3.8571}. \end{aligned}$$

Kết luận (Approx/lưới thô): chọn $t = \boxed{3}$ (kém hơn Exact do lưới không đủ mịn). Tăng số "xô" ⇒ ngưỡng tiệm cận 4.5.

(4) Weighted Quantile Sketch (WQS): vì sao "ưu tiên" vùng $x \ge 8$?

WQS đặt vạch theo *phân vị có trọng số* $\tilde{h}_i = w_i^{(\text{sample})} \cdot h_i$. Để minh hoạ rõ ràng, xét luôn tình huống có thiếu (x_1 missing) nên tập không-thiếu là $\{2, 3, 6, 8, 9, 12\}$. Đặt *trọng số mẫu*:

$$w_i = \begin{cases} 3, & x_i \ge 8, \\ 1, & \text{khác}. \end{cases} \quad \Rightarrow \quad \tilde{h} = \{1, 1, 1, 3, 3, 3\}, \quad \text{tổng } \sum \tilde{h} = 12.$$

Tích luỹ theo x cho ta các mốc $25\%, 50\%, 75\%$ lần lượt rơi tại $x = \{6, 8, 9\}$ (so với không trọng số là $\{3, 6, 9\}$), tức *các vạch bị kéo về phía đuôi phải* (vùng được ưu tiên).

Giờ chỉ thử tách ở các vạch WQS $\{6, 8, 9\}$. Với tổng có trọng số trên toàn nút:

$$G = \sum \tilde{h}_i g_i = 17, \quad H = \sum \tilde{h}_i = 13, \quad \frac{G^2}{H} = \frac{289}{13} \approx 22.2308.$$

Tại $t = 6$ (missing→trái là tốt hơn trong cấu hình này):

$$(G_L, H_L) = (-4, 4), \quad (G_R, H_R) = (21, 9) \quad \Rightarrow \quad \text{Gain} = \tfrac{1}{2}\Big(\tfrac{16}{4} + \tfrac{441}{9} - \tfrac{289}{13}\Big) \approx \boxed{15.385}.$$

Các ngưỡng 8 hoặc 9 cho Gain thấp hơn 6 trong cấu hình trọng số này.

Kết luận (WQS+Approx): do *vạch ứng viên* là $\{6, 8, 9\}$ (không có 3), thuật toán chọn $t = \boxed{6}$. So với Approx-uniform (3) hay Exact (4.5), *ngưỡng thay đổi* vì lưới ứng viên *bị chi phối bởi trọng số*.

3 Đằng sau hoạt động mô hình XGBoost

Xét mô hình là tổng K cây:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \qquad f_k \in \mathcal{F}.$$

Mỗi cây f là bản đồ lá: $f(x) = w_{q(x)}$ với $q : \mathbb{R}^m \to \{1, \dots, T\}$ và $w = (w_1, \dots, w_T)$. Hàm mục tiêu có điều chuẩn

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell(\hat{y}_i, y_i) \; + \; \sum_{k=1}^{K} \Omega(f_k), \qquad \Omega(f) = \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2, \tag{1}$$

với ℓ *lồi* theo $\hat{y}$ (vì thế $h_i \geq 0$ ở dưới).

Tại vòng t, ta thêm cây f_t vào dự đoán hiện có $\hat{y}_i^{(t-1)}$.

3.1 Xấp xỉ bậc hai và quy về bài toán bình phương có trọng số

Đặt

$$g_i := \partial_{\hat{y}} \ell(\hat{y}, y_i)\big|_{\hat{y}=\hat{y}_i^{(t-1)}}, \qquad h_i := \partial_{\hat{y}}^2 \ell(\hat{y}, y_i)\big|_{\hat{y}=\hat{y}_i^{(t-1)}} \; (\geq 0).$$

Khai triển Taylor bậc hai quanh $\hat{y}_i^{(t-1)}$ và bỏ hằng số (không phụ thuộc f_t):

$$\tilde{\mathcal{L}}^{(t)}(f_t) \; = \; \sum_{i=1}^{n} \Big[g_i \, f_t(x_i) + \tfrac{1}{2} \, h_i \, f_t(x_i)^2 \Big] \; + \; \Omega(f_t). \tag{2}$$

[Diễn giải WLS] Viết lại $\sum_i \frac{1}{2} h_i \big(f_t(x_i) - \frac{g_i}{h_i} \big)^2 + \Omega(f_t)$ (cộng/ trừ hằng số), suy ra *bài toán tại vòng* t chính là hồi quy bình phương có trọng số với "nhãn" g_i/h_i và trọng số h_i.

3.2 Phân rã theo lá và nghiệm tối ưu w_j

Ký hiệu $I_j = \{i : q(x_i) = j\}$, $G_j = \sum_{i \in I_j} g_i$, $H_j = \sum_{i \in I_j} h_i$. Vì $f_t(x) = w_{q(x)}$,

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T} \Big(G_j \, w_j + \tfrac{1}{2} H_j \, w_j^2 \Big) \; + \; \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2 \; + \; \gamma T \tag{3}$$

$$= \sum_{j=1}^{T} \Big(G_j \, w_j + \tfrac{1}{2} (H_j + \lambda) \, w_j^2 \Big) \; + \; \gamma T. \tag{4}$$

[Bài toán lồi tách theo lá] $\tilde{\mathcal{L}}^{(t)}$ là tổng của T hàm bậc hai lồi (vì $H_j + \lambda \geq 0$), do đó tối ưu từng w_j *độc lập*.

[Nghiệm tối ưu tại lá] Cho cấu trúc cây q cố định, nghiệm tối ưu là

$$\boxed{w_j \overset{=-\frac{G_j}{H_j+\lambda}}{}} \qquad \text{(duy nhất vì } H_j + \lambda > 0 \text{ khi } \lambda > 0 \text{ hoặc } H_j > 0\text{)}.$$

Chứng minh. Lấy đạo hàm theo w_j: $G_j + (H_j + \lambda) w_j = 0 \Rightarrow w_j \overset{=-G_j/(H_j+\lambda)}{}$. Đạo hàm bậc hai: $H_j + \lambda > 0$ $\Rightarrow$ nghiệm duy nhất tối tiểu toàn cục. □

[Điểm cấu trúc của cây] Thế w vào $\tilde{\mathcal{L}}^{(t)}$ được

$$\boxed{\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T}.$$

Chứng minh. Với từng lá: $\min_{w_j} G_j w_j + \frac{1}{2}(H_j+\lambda)w_j^2 = -\frac{1}{2}\frac{G_j^2}{H_j+\lambda}$. Cộng theo j và thêm γT. □

[Độ lợi (gain) của một split] Giả sử một nút có tổng (G, H) tách thành trái/ phải (G_L, H_L), (G_R, H_R). Độ giảm mục tiêu (improvement) khi tách là

$$\boxed{\text{Gain} = \frac{1}{2}\Big(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\Big) - \gamma}.$$

Chứng minh. Đóng góp của nút cha trước khi tách (theo Equation 3.2 với T=1): $-\frac{1}{2}\frac{G^2}{H+\lambda} + \gamma$. Sau tách thành hai lá: $-\frac{1}{2}\frac{G_L^2}{H_L+\lambda} - \frac{1}{2}\frac{G_R^2}{H_R+\lambda} + 2\gamma$. Độ giảm mục tiêu là (cha) trừ (con) $= \frac{1}{2}(\cdots) - \gamma$. □

4 Ví dụ tính tay 1: Regression (MSE)

Dữ liệu (1 chiều, vòng đầu)

i	1	2	3	4	5	6	7
x_i	1	2	3	6	8	9	12
y_i	2	2	1	-1	-2	-2	-3
$g_i = -y_i$	-2	-2	-1	$+1$	$+2$	$+2$	$+3$
h_i	1	1	1	1	1	1	1

Tại gốc: $G = \sum g_i = 3$, $H = \sum h_i = 7$.

Bước 1: duyệt *mọi* ngưỡng (Exact Greedy)

Các ngưỡng giữa các giá trị x: $t \in \{1.5, 2.5, 4.5, 7, 8.5, 10.5\}$. Tính (G_L, H_L), (G_R, H_R) và Gain $= \frac{1}{2}\left(\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right)$.

t	(G_L, H_L)	(G_R, H_R)	Gain
1.5	$(-2, 1)$	$(5, 6)$	$\frac{1}{2}\left(\frac{4}{1} + \frac{25}{6} - \frac{9}{7}\right) = \mathbf{3.4405}$
2.5	$(-4, 2)$	$(7, 5)$	$\frac{1}{2}\left(\frac{16}{2} + \frac{49}{5} - \frac{9}{7}\right) = \mathbf{8.2571}$
4.5	$(-5, 3)$	$(8, 4)$	$\frac{1}{2}\left(\frac{25}{3} + \frac{64}{4} - \frac{9}{7}\right) = \mathbf{11.5238}$
7.0	$(-4, 4)$	$(7, 3)$	$\frac{1}{2}\left(\frac{16}{4} + \frac{49}{3} - \frac{9}{7}\right) = \mathbf{9.5238}$
8.5	$(-2, 5)$	$(5, 2)$	$\frac{1}{2}\left(\frac{4}{5} + \frac{25}{2} - \frac{9}{7}\right) = \mathbf{6.0071}$
10.5	$(0, 6)$	$(3, 1)$	$\frac{1}{2}\left(0 + \frac{9}{1} - \frac{9}{7}\right) = \mathbf{3.8571}$

(5)

Chọn split gốc: $t = \boxed{4.5}$.

Bước 2: tính w_j^* cho hai lá (lấy $\lambda = 0$)

$$\text{Trái } (\leq 4.5): \quad (G_L, H_L) = (-5, 3) \Rightarrow w_L^* = -\frac{-5}{3} = \frac{5}{3} \tag{6}$$

$$\text{Phải } (> 4.5): \quad (G_R, H_R) = (8, 4) \Rightarrow w_R^* = -\frac{8}{4} = -2 \tag{7}$$

Cập nhật dự đoán: $\hat{y}^{(1)} = \hat{y}^{(0)} + \eta f_1(x)$. Nếu $\eta = 1$: điểm số lá là $\left\{\frac{5}{3}, -2\right\}$.

(Tuỳ chọn) Bước 3: split thêm mỗi nhánh (độ sâu 2)

Nhánh trái ($x \leq 4.5$): chỉ có $\{1, 2, 3\}$. Thử $t = 1.5, 2.5$, tính Gain so với tổng của nút trái $(G, H) = (-5, 3)$:

$$t = 1.5: \quad (G_L, H_L) = (-2, 1), \quad (G_R, H_R) = (-3, 2) \tag{8}$$

$$\Rightarrow \text{Gain} = \frac{1}{2}\left(\frac{4}{1} + \frac{9}{2} - \frac{25}{3}\right) = 0.0833 \tag{9}$$

$$t = 2.5: \quad (G_L, H_L) = (-4, 2), \quad (G_R, H_R) = (-1, 1) \tag{10}$$

$$\Rightarrow \text{Gain} = \frac{1}{2}\left(\frac{16}{2} + \frac{1}{1} - \frac{25}{3}\right) = \mathbf{0.3333} \tag{11}$$

Nhánh phải ($x > 4.5$): $\{4, 5, 6, 7\}$. Thử $t = 7, 8.5, 10.5$ và tính tương tự (ngưỡng tốt nhất: $t = 7$).

Ảnh hưởng điều chuẩn. Nếu $\lambda > 0$, thay $H_\bullet \to H_\bullet + \lambda \Rightarrow |w^*|$ nhỏ hơn; nếu $\gamma > 0$, Gain bị trừ γ — split yếu có thể bị loại.

5 Ví dụ tính tay 2: Classification (logistic)

Dữ liệu (1 chiều, vòng đầu)

i	1	2	3	4	5	6
x_i	1	2	3	6	8	10
y_i	0	0	1	1	1	0
$\hat{y}^{(0)}$	0	0	0	0	0	0
$p_i = \sigma(\hat{y}^{(0)})$	0.5	0.5	0.5	0.5	0.5	0.5
$g_i = p_i - y_i$	+0.5	+0.5	−0.5	−0.5	−0.5	+0.5
$h_i = p_i(1 - p_i)$	0.25	0.25	0.25	0.25	0.25	0.25

Tại gốc: $G = \sum g_i = 0$, $H = \sum h_i = 6 \times 0.25 = 1.5$.

Bước 1: duyệt *mọi* ngưỡng (Exact Greedy)

Ngưỡng $t \in \{1.5, 2.5, 4.5, 7, 9\}$. Gain $= \frac{1}{2}\left(\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right)$ (lưu ý $G = 0$).

t	(G_L, H_L)	(G_R, H_R)	Gain
1.5	$(+0.5, 0.25)$	$(-0.5, 1.25)$	$\frac{1}{2}\left(\frac{0.25}{0.25} + \frac{0.25}{1.25} - 0\right) = \mathbf{0.6}$
2.5	$(+1.0, 0.5)$	$(-1.0, 1.0)$	$\frac{1}{2}\left(\frac{1}{0.5} + \frac{1}{1} - 0\right) = \mathbf{1.5}$
4.5	$(+0.5, 0.75)$	$(-0.5, 0.75)$	$\frac{1}{2}\left(\frac{0.25}{0.75} + \frac{0.25}{0.75}\right) = \mathbf{0.3333}$
7.0	$(0.0, 1.0)$	$(0.0, 0.5)$	0
9.0	$(-0.5, 1.25)$	$(+0.5, 0.25)$	**0.6**

(12)

Chọn split gốc: $t = \boxed{2.5}$.

Bước 2: tính w_j^* (lấy $\lambda = 0$)

$$\text{Trái } (\leq 2.5): \quad (G_L, H_L) = (+1.0, 0.5) \Rightarrow w_L^* = -\frac{1.0}{0.5} = -2 \tag{13}$$

$$\text{Phải } (> 2.5): \quad (G_R, H_R) = (-1.0, 1.0) \Rightarrow w_R^* = -\frac{-1.0}{1.0} = +1 \tag{14}$$

Cập nhật logit: $\hat{y}^{(1)} = \hat{y}^{(0)} + \eta f_1(x)$. Nếu $\eta = 1$:

$$\hat{y}^{(1)}(x) = \begin{cases} -2, & x \leq 2.5, \\ +1, & x > 2.5, \end{cases} \tag{15}$$

$$p^{(1)}(x) = \sigma(\hat{y}^{(1)}(x)) = \begin{cases} \sigma(-2) \approx 0.119, & x \leq 2.5, \\ \sigma(1) \approx 0.731, & x > 2.5. \end{cases} \tag{16}$$

Các xác suất sau cập nhật phản ánh đúng phân bố nhãn: vùng nhỏ đa số $y = 0$, vùng lớn đa số $y = 1$.