LẬP TRÌNH SONG SONG ỨNG DỤNG

BÁO CÁO TIẾN ĐỘ

PLANT LEAF DISEASE DETECTION USING XGBOOST

Nhóm Double Slash

20120165 - Hồng Nhất Phương

19120522 - Phạm Quốc Hưng

01

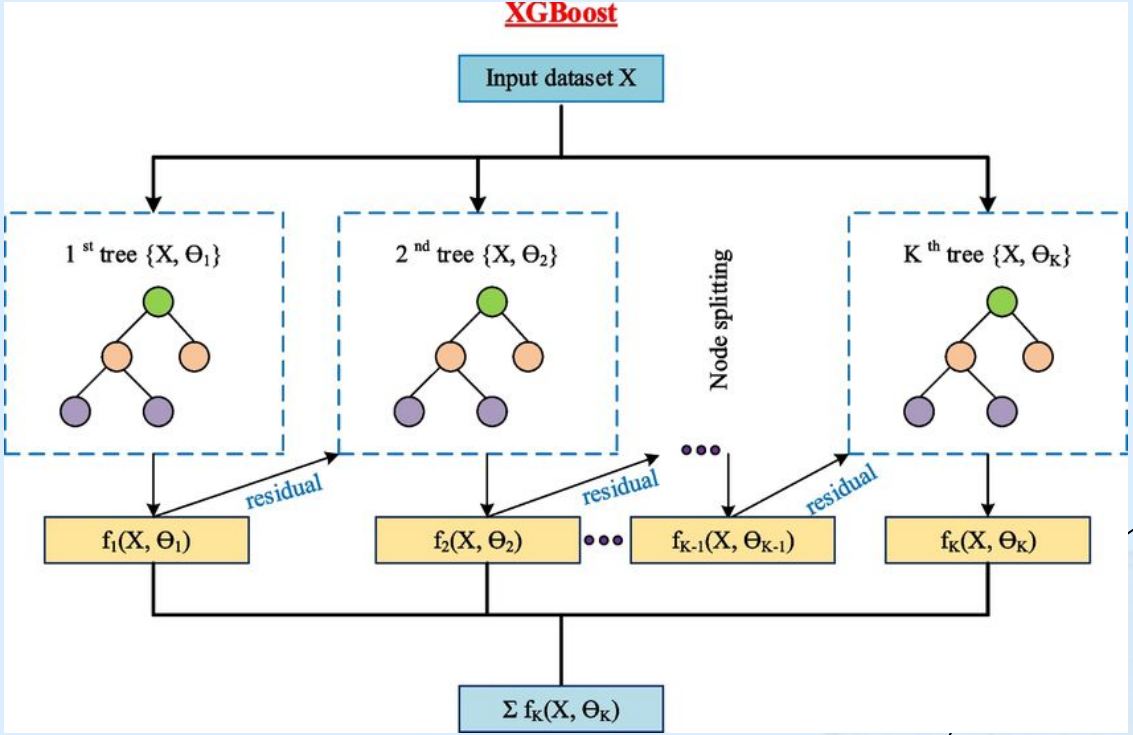
Sơ lược về XGBoost

Sơ lược về XGBoost

XGBoost là một mô hình học máy sử dụng kĩ thuật Gradient Boosting để giải quyết các bài toán phân lớp và hồi quy.

Ý tưởng của Gradient Boosting là kết hợp nhiều weak models, mỗi model sẽ cố gắng giảm thiểu độ lỗi trong kết quả dự đoán của các mô hình trước đó, để có được kết quả dự đoán cuối cùng tốt hơn.

Sơ lược về XGBoost

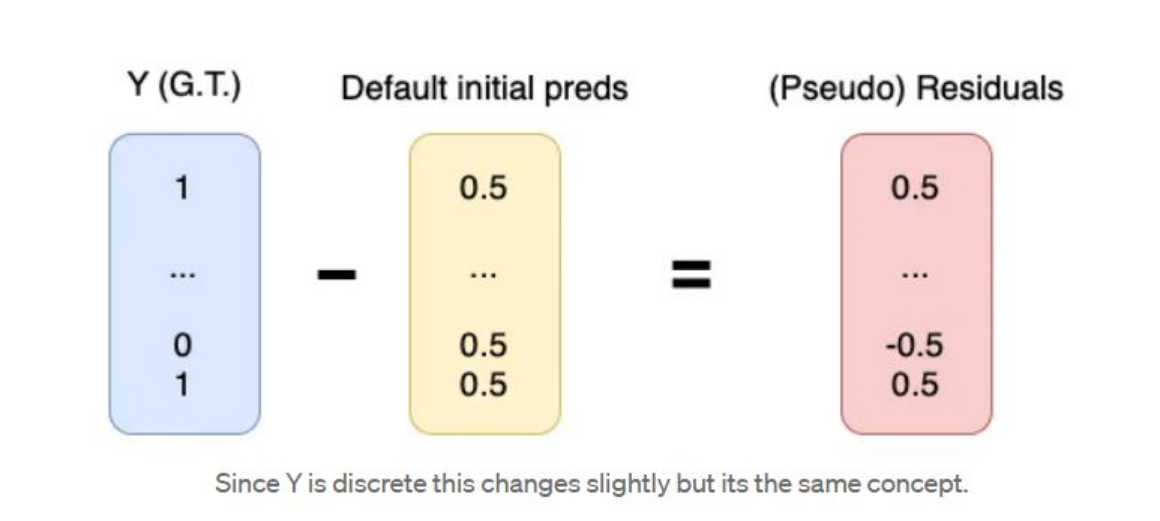


02

Cách hoạt động

Residual

Residual = y - p



Similarity Score

XGBoost sử dụng Similarity Score để đánh giá và chọn ra split point trong quá trình xây dựng cây quyết định

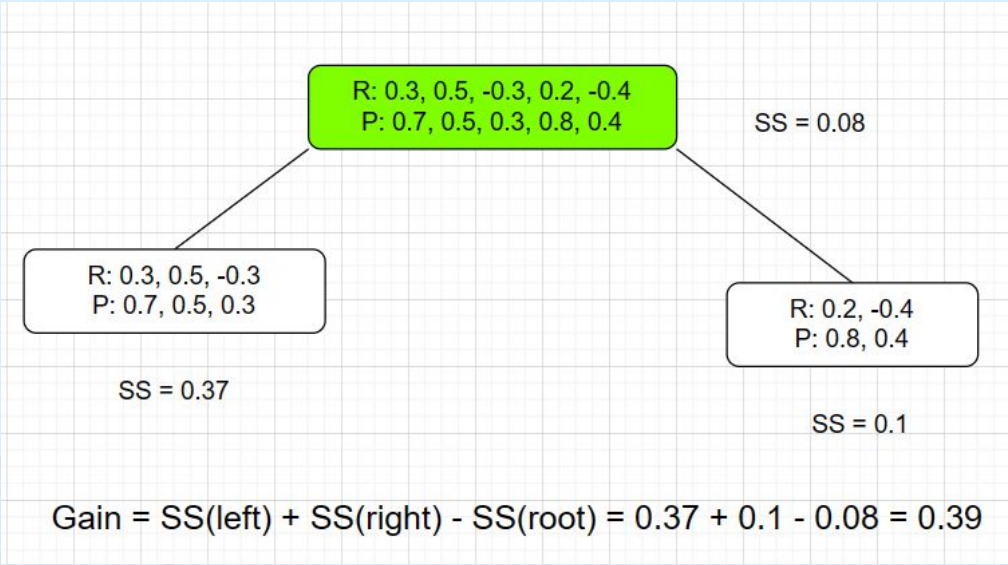
$$\text{Similarity Score} = \frac{\sum(residuals)^2}{\sum P*(1-P)+\lambda}$$

R: 0.3, 0.5, -0.3, 0.2, -0.4
P: 0.7, 0.5, 0.3, 0.8, 0.4

$$\frac{(0.3+0.5-0.3+0.2-0.4)^2}{(0.7*0.3+0.5*0.5+0.3*0.7+0.8*0.2+0.4*0.6)} = \frac{9}{107} \approx 0.08$$

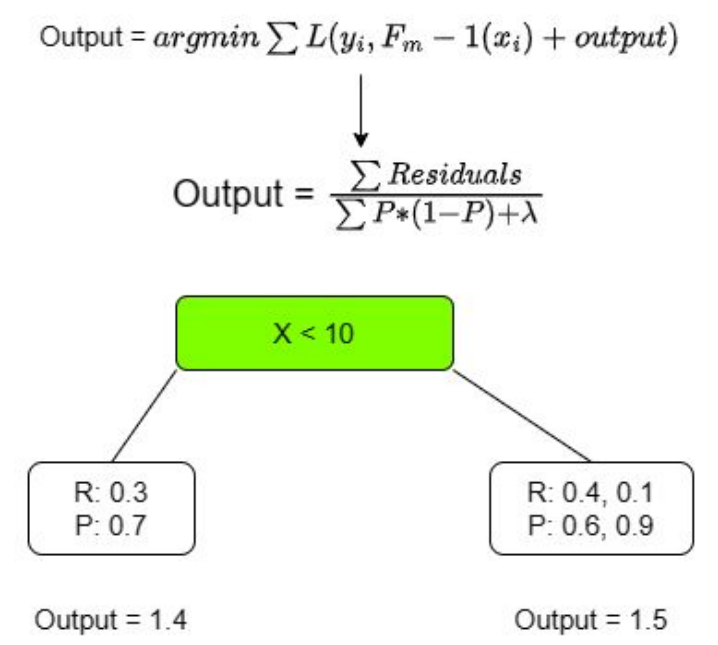
Chọn ra split tốt nhất

Lần lượt duyệt qua từng feature, feature và value có chỉ số Gain cao nhất sẽ được chọn làm split point



Output value

Tính toán output value ở node lá. Output value ở đây là một giá trị tối ưu sao cho loss function L(yi, Fm-1(xi) + output) là nhỏ nhất



Update prediction

Cập nhật kết quả dự đoán mới dựa vào giá trị output vừa tính được

$$\text{pred (log odd)} = log(\frac{p_{i-1}}{1-p_{i-1}}) + lr * output_i$$

$$\downarrow$$

$$p = \frac{e^{logodd}}{1+e^{logodd}}$$

03

Cài đặt tuần tự

Cài đặt tuần tự

Tính toán Similarity Score và giá trị output ở leaf node

```python
#Compute the Similarity Score
def similarity(self, residual, probs):
    nu = np.sum(residual) ** 2
    de = np.sum(probs * (1 - probs)) + self.lambda_
    return nu / de

#Compute the output value in leaf node
def compute_output(self, residual, probs):
    nu = np.sum(residual)
    de = np.sum(probs * (1 - probs)) + self.lambda_
    return nu / de
```

Cài đặt tuần tự

Tìm ra feature tốt nhất để làm split point dựa vào Gain Score

```
gain = self.similarity(r_left, p_left) + self.similarity(r_right, p_right) - self.similarity(residual, probs)

if gain > best_gain:
    best_gain = gain
    best_split_feature_idx = feature_idx
    best_split_value = value
```

Cài đặt tuần tự

Tính toán phần dư và hàm chuyển đổi giữa giá trị log odd và xác suất

```python
#Turn probability into log odds value
def compute_logodds(self, p):
    return np.log(p / (1 - p))

#Caclculate pseudo residuals
def residual(self, y_true, y_pred):
    return (y_true - y_pred)

#Change log odds value back to probability
def compute_prob(self, logodds_p):
    return np.exp(logodds_p) / (1 + np.exp(logodds_p))
```

Cài đặt tuần tự

Cập nhật kết quả dự đoán theo xác suất

```
logodds_p = log_odds + self.lr * model.predict(X)
p = self.compute_prob(logodds_p)
```

04

Huấn luyện và đánh giá mô hình

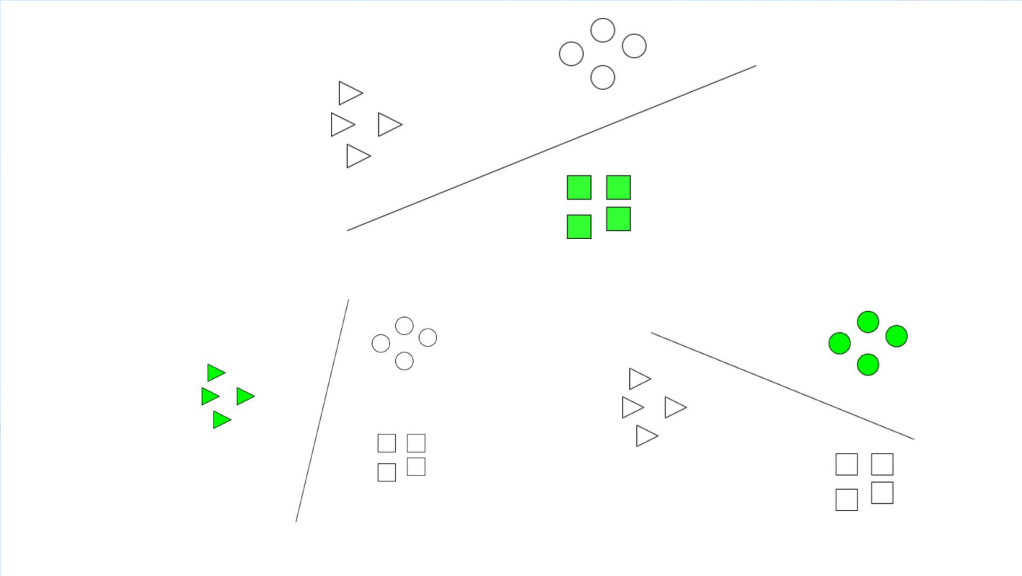
Multiclass problem

XGBoost phân loại label dựa trên xác suất một observation có khả năng thuộc về label đó (tương tự như logistic regression) nên chỉ giới hạn cho các bài toán phân loại hai lớp.

Giải pháp:
Sử dụng phương pháp One vs Rest (OvR) để giải quyết bài toán phân loại đa lớp

OvR strategy

Chiến thuật của **OvR** là phân tách bài toán phân loại đa lớp ban đầu thành **n** bài toán phân loại hai lớp tương ứng với **n** lớp phải phân loại.



Kết quả

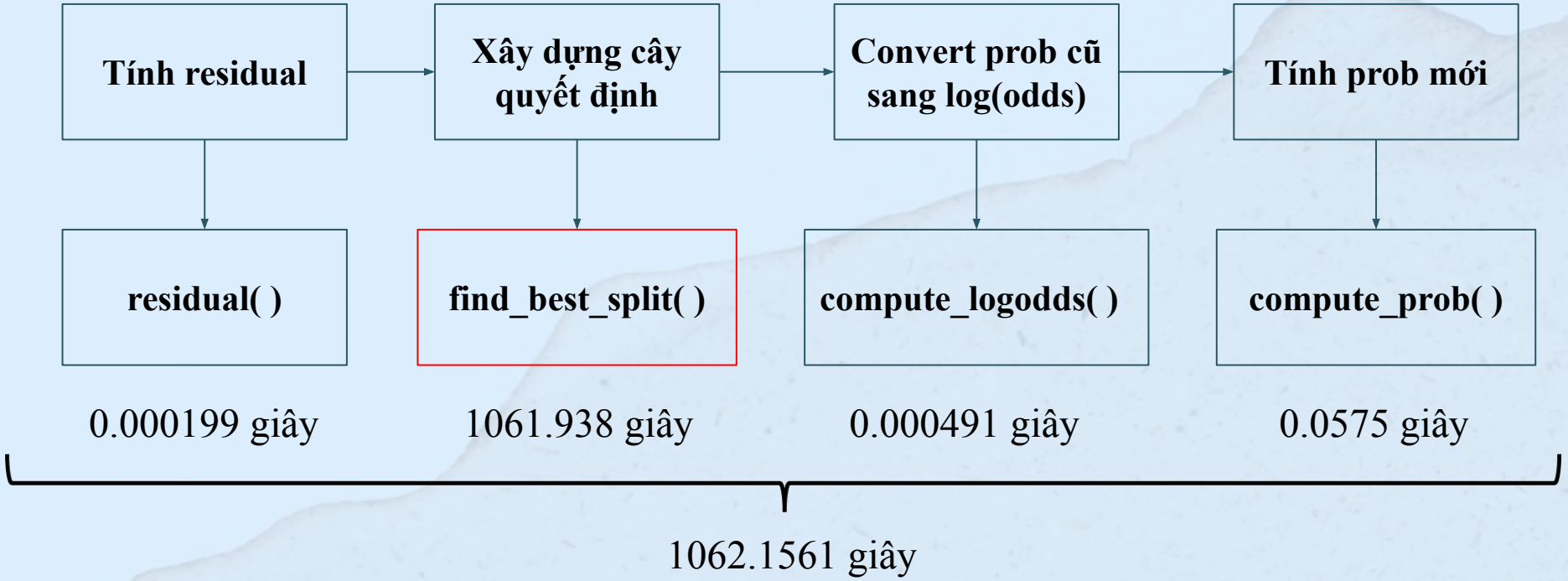
Số classes	Kích thước dữ liệu	Accuracy	Thời gian chạy (seconds)	Thời gian chạy (minutes)
2	3000	~0.95	350-370s	~6m
3	3000	~0.95	1060-1150s	17-19m

Thời gian huấn luyện của mô hình trên tập dữ liệu hai lớp và ba lớp có sự chênh lệch lớn.

05

Cài đặt song song

Khảo sát thời gian thực thi của các giai đoạn chính trong XGBoost

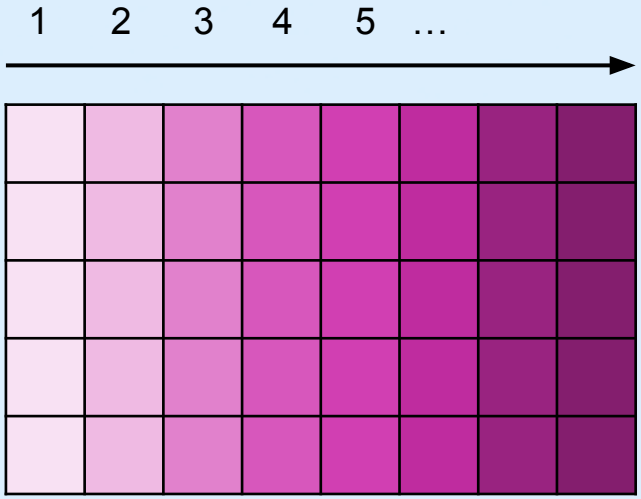


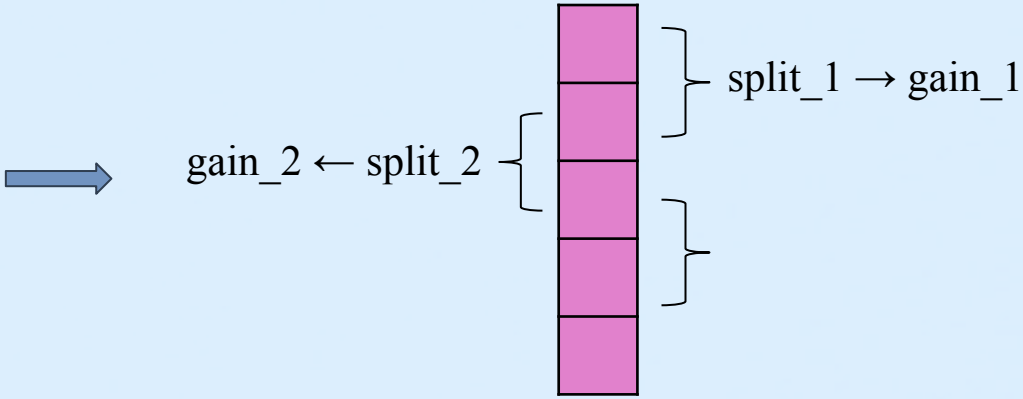
Song song hóa hàm find_best_split()

Mô tả hàm: find_best_split() là hàm tìm giá trị split sao cho gain của chúng khi rẽ nhánh đạt giá trị lớn nhất => đây là giá trị split tốt nhất dùng để rẽ nhánh.

Ý tưởng tuần tự: Vét cạn tất cả trường hợp: Xét feature thứ i, tìm từng giá trị split theo feature i, tính gain và so sánh với best_gain, lặp lại đến hết và sau cùng lưu vị trí feature cùng với giá trị split và best_gain.

Ý tưởng tuần tự





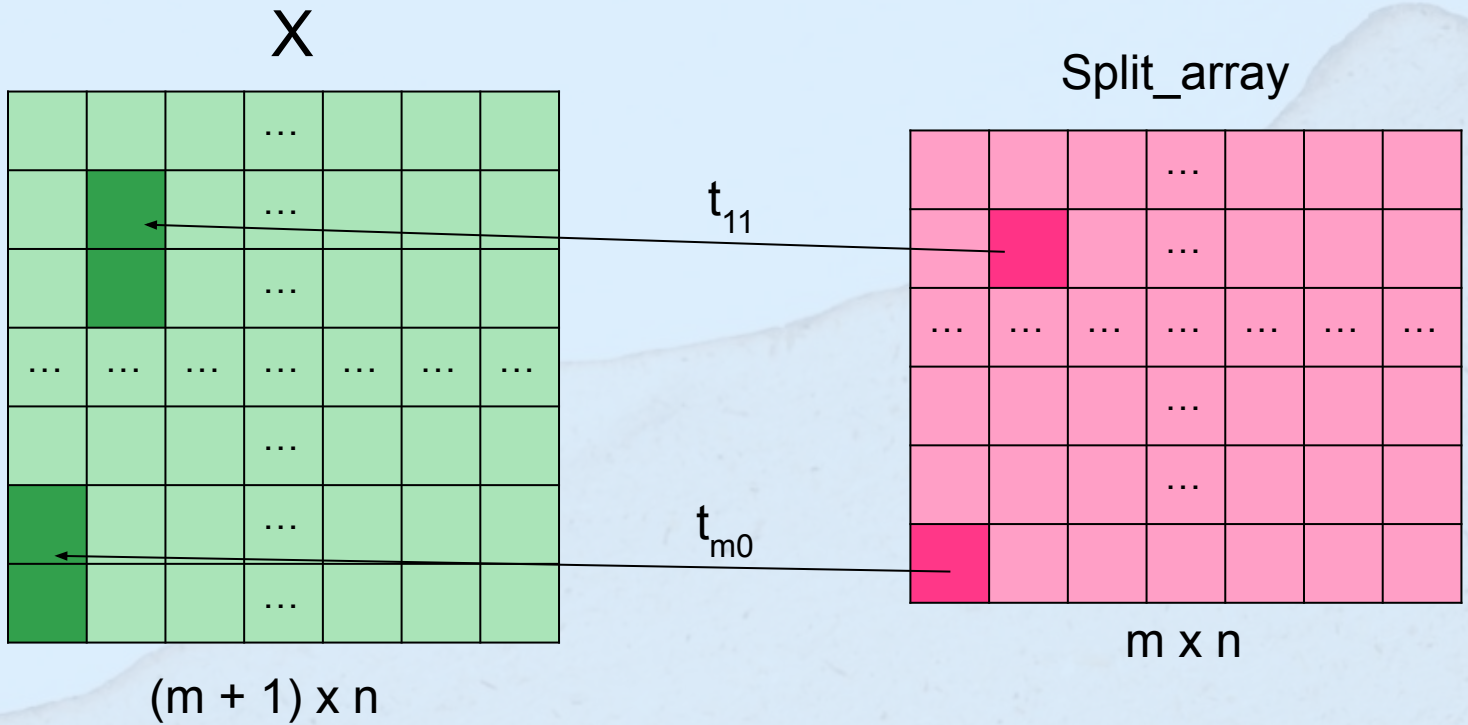
- Max gain = …
- Best split value = …
- Best split feature = …

Song song hóa hàm find_best_split()

Ý tưởng song song: Chia làm 2 hàm con:

- Tính tất cả giá trị split value, mỗi thread sẽ đảm nhiệm tính một giá trị split
- Sau khi xác định được mảng giá trị split, với mỗi giá trị split, ta tiến hành rẽ nhánh và tính gain.

Bước 1: Tính split_array



X

2	3.5	5	9
2	1	10	9
5	7	6	11

sorted_X

2	1	5	4
2	3.5	6	9
5	7	10	9

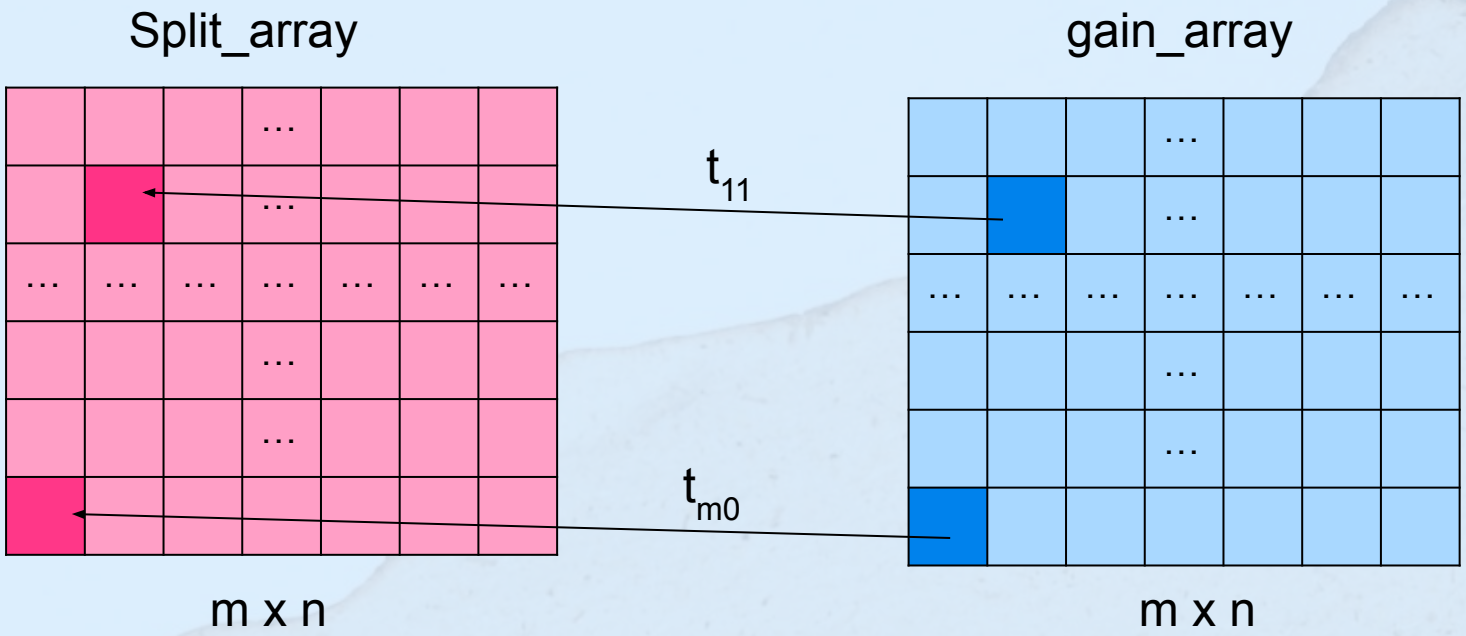
sort từng cột

Split_array

NaN	2.25	5.5	6.5
3.5	5.25	8	NaN

$$thread(i,j) \rightarrow \begin{cases} \dfrac{X[i,j]+X[i+1,j]}{2} \; if \; X[i,j] \neq X[i+1,j] \\ NaN \; if X[i,j] \neq X[i+1,j] \end{cases}$$

Bước 2: Tính gain_array



x_1	2	3.5	5	9
x_2	2	1	10	9
x_3	5	7	6	11

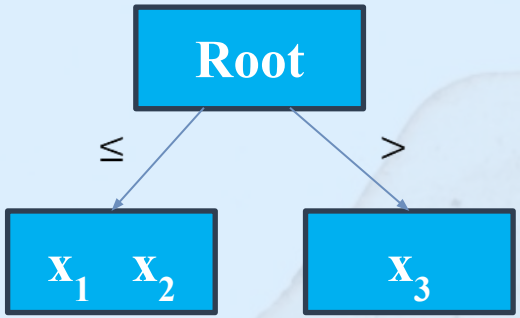
X

NaN	2.25	5.5	6.5
3.5	5.25	8	NaN

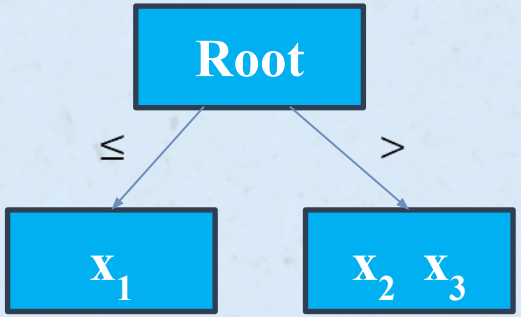
Split_array

g_{00}	g_{01}	g_{02}	g_{03}
g_{10}	g_{11}	g_{12}	g_{13}

gain_array



Split_value = 3.5



Split_value = 5.5

x_1	2	3.5	5	9
x_2	2	1	10	9
x_3	5	7	6	11

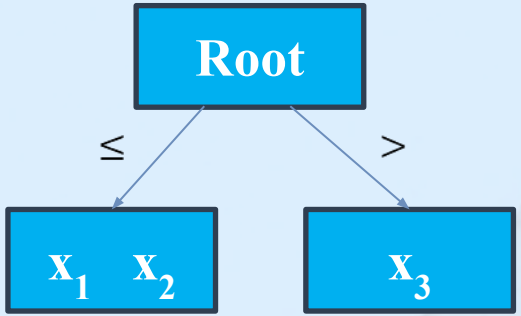
X

NaN	2.25	5.5	6.5
3.5	5.25	8	NaN

Split_array

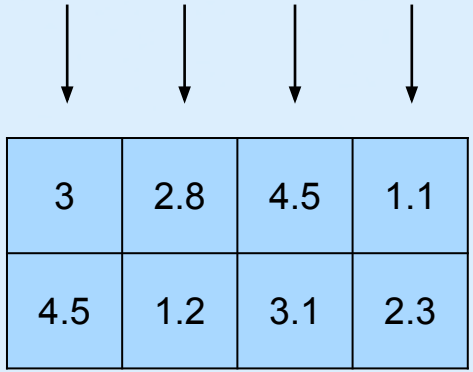
g_{00}	g_{01}	g_{02}	g_{03}
g_{10}	g_{11}	g_{12}	g_{13}

gain_array



Split_value = 3.5

$$gain_{3.5} = Similarity_{left} + Similarity_{right} - Similarity_{root}$$



3	2.8	4.5	1.1
4.5	1.2	3.1	2.3

gain_array

Phiên bản tuần tự xét theo từng feature (cột), do vậy:
max_gain = 4.5 tại vị trí **[1, 0]**

3	2.8	4.5	1.1
4.5	1.2	3.1	2.3

gain_array

Phiên bản song song dùng **np.argmax()**
Do đó để đảm bảo kết quả 2 phiên bản như nhau, cần **transpose** gain_array

3	4.5
2.8	1.2
4.5	3.1
1.1	2.3

argmax() → [0, 1] → [1, 0]

Kết quả của transpose

Song song hóa hàm residual()

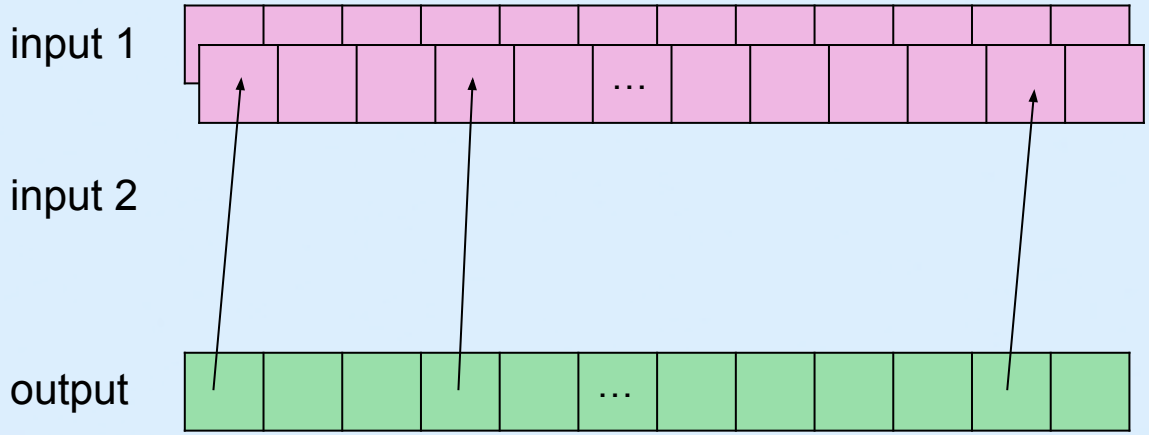
Mô tả hàm: residual() là hàm tính toán sự chênh lệch giữa y thực tế và kết quả dự đoán

Ý tưởng tuần tự: Sử dụng thư viện numpy để tính giá trị residual

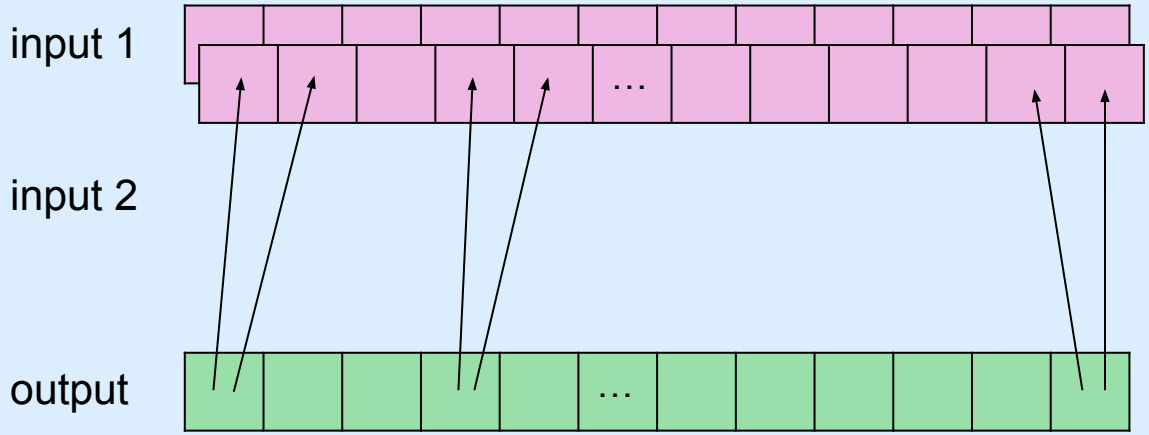
Ý tưởng song song: Mỗi một thread sẽ được dùng để tính giá trị residual của một mẫu (sample)

$$residual = y_{true} - y_{pred}$$

Song song hóa hàm residual()



Ý tưởng cải tiến hàm residual()



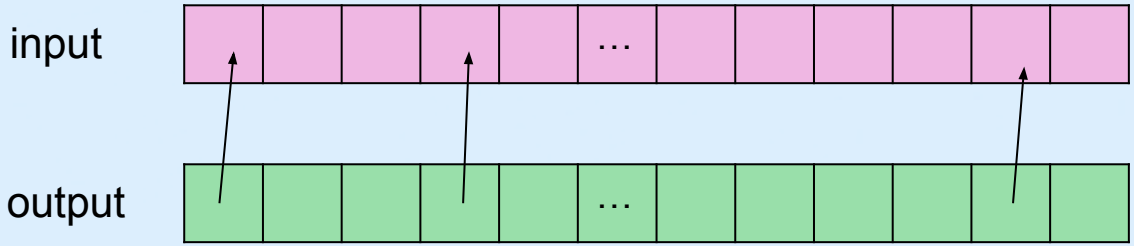
Song song hóa hàm compute_logodds() và compute_prob()

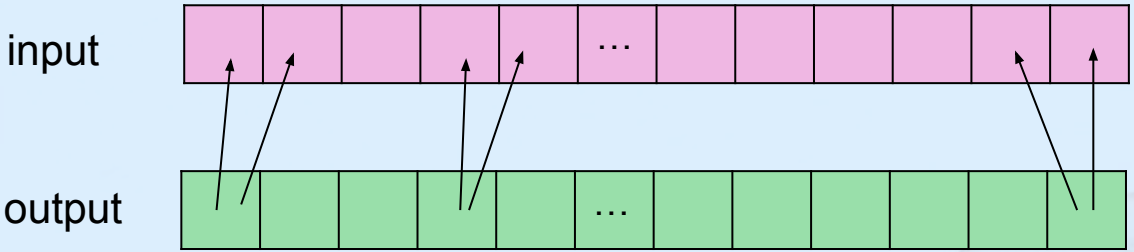
Mô tả hàm: compute_prob() là hàm chuyển giá trị log(odds) thành giá trị xác suất p
compute_logodds() là hàm chuyển giá trị xác suất p thành giá trị log(odds)
Ý tưởng tuần tự: Sử dụng thư viện numpy để tính giá trị xác suất p hoặc log(odds)
Ý tưởng song song: Mỗi một thread sẽ được dùng để tính giá trị p hoặc log(odds) của một mẫu (sample)

$$\log(odds) = ln\left(\frac{p}{1-p}\right)$$

$$p = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

Song song hóa hàm compute_logodds() và compute_prob()





06

Huấn luyện & đánh giá kết quả

Bộ dữ liệu huấn luyện giống với phiên bản tuần tự
Chiến lược: OvR giống với phiên bản tuần tự

Số classes	Kích thước dữ liệu	Accuracy tuần tự	Accuracy song song	Thời gian chạy tuần tự	Thời gian chạy song song
2	3000	0.95	0.95	350 – 370s	4 – 6s
3	3000	0.946	0.946	1060 – 1150s	9 – 11s

Bộ dữ liệu huấn luyện **10 lớp đầy đủ**
Chiến lược: OvR giống với phiên bản tuần tự

Số classes	Kích thước dữ liệu	Accuracy tuần tự	Thời gian chạy song song
2	3000	0.95	4 – 6s
3	3000	0.946	9 – 11s
10	10.000	0.891	2600 – 2800s

Bộ dữ liệu huấn luyện **10 lớp đầy đủ**
Chiến lược: **OvR** và **under sampling** (40 : 60)

Số classes	Kích thước dữ liệu	Accuracy tuần tự	Thời gian chạy song song	Note
2	3000	0.95	4 – 6s	
3	3000	0.946	9 – 11s	
10	10.000	0.891	2600 – 2800s	
10	10.000	0.903	600 – 700s	Dùng under sampling

07

Kế hoạch báo cáo tiếp theo

STT	Tên công việc	
1	Thu thập dữ liệu	✓
2	Trích xuất đặc trưng	✓
3	Cài đặt XGBoost tuần tự	✓
4	Cài đặt XGBoost song song	✓
5	Áp dụng kĩ thuật cải tiến	**Báo cáo vào lần tiếp theo**

THE END