

Bài tập lý thuyết

☑ Trắc nghiệm 40 ⬇ PDF bài

Một số đề thi AI có phần lý thuyết: không cần code, chỉ cần lời giải viết ra kiểm được từng bước. Ví dụ: một bộ phát hiện bài nộp có lỗi “phát hiện được 90% bài lỗi”; vậy một bài bị cảnh báo có xác suất lỗi 90% không? Không nhất thiết, và phải dùng công thức Bayes để giải thích.

Bài này luyện các dạng bài lý thuyết về xác suất, đại số tuyến tính, tối ưu, kiến trúc mạng và đánh giá model; mỗi mục có ví dụ giải tay và một đoạn code ngắn để đối chiếu.

MỤC TIÊU

- Trình bày lời giải lý thuyết rõ ràng, từng bước.
- Giải bài về Bayes, kỳ vọng, log-loss.
- Kiểm shape phép nhân ma trận; tính gradient bằng quy tắc chuỗi.
- Tính số tham số của convolution và attention.

Một lời giải cần cho người đọc kiểm được

Trong CP, chứng minh thuật toán thường có invariant và độ phức tạp. Với AI, lời giải cũng cần một đường kiểm tra: đề cho gì, mô hình giả định gì, dùng định nghĩa nào và kết quả có ý nghĩa gì. Chẳng hạn “ $p = 0.9$ ” chưa đủ; phải nói đây là $P(\text{cảnh báo} \mid \text{có lỗi})$ hay $P(\text{có lỗi} \mid \text{cảnh báo})$.

Ta dùng quy trình bốn bước:

1. Xác định đại lượng cần trả lời, không tính ngay số để có nhất.
2. Viết ký hiệu cùng miền giá trị, shape hoặc đơn vị.
3. Biến đổi công thức trước khi thế số, để người đọc nhận ra phép tính dùng định nghĩa nào.
4. Kiểm bằng một trường hợp nhỏ hoặc code độc lập, rồi diễn giải kết quả.

Code là công cụ đối chiếu; dòng True không thay thế lời giải thích vì sao.

Hai chiều của việc kiểm không cân nhau. Một khẳng định “với mọi $\mathbf{w}$ ” không được chứng minh bằng hữu hạn lần chạy, dù nhiều đến đâu. Ngược lại, một phản ví dụ nhỏ, chạy đúng, đủ để bác bỏ nó.

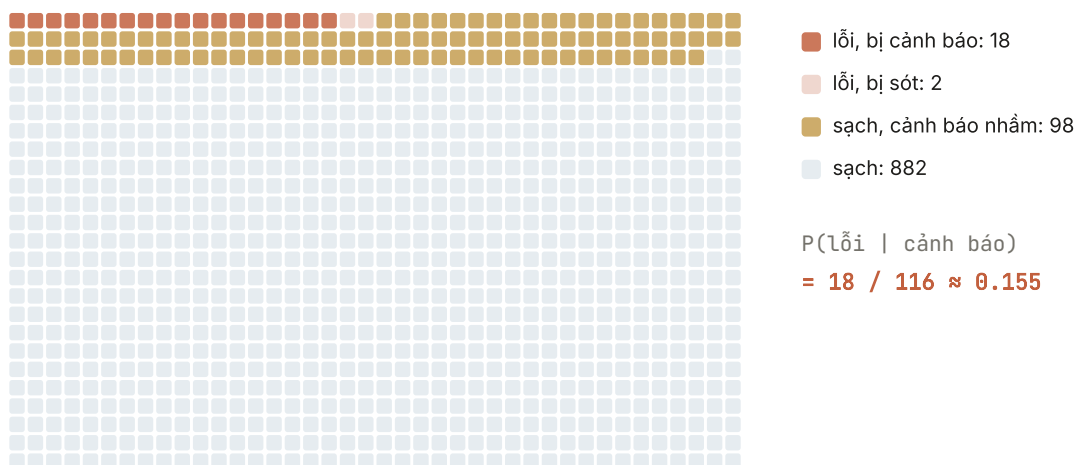
HỎI NHANH

Đề yêu cầu chứng minh $\text{MSE } \frac{1}{n} \sum_i (\hat{y}_i - y_i)^2$ không âm với mọi $\mathbf{w}$. Một bạn chạy 1000 bộ $\mathbf{w}$ ngẫu nhiên và thấy loss luôn từ 0 trở lên. Phần này có giá trị gì?

- A Đủ là một chứng minh, vì 1000 bộ đã là rất nhiều
- B Chỉ là kiểm tra, vẫn cần lập luận rằng bình phương không âm
- C Vô ích, vì bài viết không chấm phần chạy code
- D Đủ, nếu các bộ $\mathbf{w}$ phủ cả số âm lẫn số dương

Xác suất: điều kiện nằm ở phía nào?

Giả sử một tập kiểm có đúng 1000 bài nộp: 20 bài có lỗi và 980 bài không lỗi. Detector cảnh báo 18 trong 20 bài lỗi, đồng thời cảnh báo nhầm 98 trong 980 bài sạch.



Trong 116 cảnh báo chỉ có 18 bài thật sự lỗi.

Chọn ngẫu nhiên đều một bài trong tập này. Gọi B là biến cố “bài có lỗi”, F là “bị cảnh báo”. Ta có $P(B) = 20/1000 = 0.02$, $P(F \mid B) = 18/20 = 0.9$ và $P(F \mid \neg B) = 98/980 = 0.1$. Bayes đảo chiều điều kiện:

$$P(B \mid F) = \frac{P(F \mid B)P(B)}{P(F)}.$$

Mẫu số dùng total probability:

$$P(F) = P(F \mid B)P(B) + P(F \mid \neg B)P(\neg B) = 0.018 + 0.098 = 0.116.$$

Vậy $P(B \mid F) = 0.018/0.116 = 18/116 \approx 0.1552$, chỉ khoảng 15.5%. Khi lỗi hiếm, một tỉ lệ báo nhầm nhỏ trên nhóm sạch lớn vẫn sinh nhiều cảnh báo hơn nhóm lỗi. Không có mâu thuẫn với recall 90%; hai câu hỏi dùng hai tập làm mẫu số khác nhau. Recall và tỉ lệ báo nhầm là tính chất của detector; $P(B \mid F)$ còn phụ thuộc tỉ lệ lỗi $P(B)$ của nơi đem detector ra dùng.

Bộ phân loại Bayes chọn lớp có xác suất hậu nghiệm lớn nhất; không có đặc trưng nào thì chỉ còn tiền nghiệm, nên với hai lớp chiếm 2/3 và 1/3 ta đoán lớp đông.

Python

```
import numpy as np

tp, fn, fp, tn = 18, 2, 98, 882
```

```

total = tp + fn + fp + tn
prior = (tp + fn) / total
sensitivity = tp / (tp + fn)
false_alarm_rate = fp / (fp + tn)
alarm_probability = sensitivity * prior + false_alarm_rate * (1 - prior)
posterior = sensitivity * prior / alarm_probability
print(f"P(lỗi)={prior:.4f}, P(cảnh báo)={alarm_probability:.4f}")
print(f"P(lỗi | cảnh báo)={posterior:.4f}")
assert np.isclose(posterior, tp / (tp + fp))

```

KẾT QUẢ

$P(\text{lỗi})=0.0200$, $P(\text{cảnh báo})=0.1160$
 $P(\text{lỗi} \mid \text{cảnh báo})=0.1552$

Muốn nhân hai xác suất cho hai thuộc tính, phải kiểm điều kiện độc lập tương ứng. Công thức luôn đúng là $P(A \cap B) = P(A \mid B)P(B)$; chỉ khi độc lập mới đổi thành $P(A)P(B)$. Một feature xuất hiện nhiều trong bài lỗi không tự động độc lập với feature khác, cũng không chứng minh quan hệ nhân quả.

Vài phép đếm cơ bản: xúc xắc cân đối ra mặt từ 5 trở lên có xác suất $2/6$; xác suất có điều kiện chỉ đếm trong nhóm điều kiện, nên lớp có 12 bạn nữ, 3 bạn nữ đeo kính cho $P(\text{kính} \mid \text{nữ}) = 3/12$.

Naive Bayes (bài 6.1) dùng đúng giả định độc lập khi đã biết lớp, nên chỉ ước lượng từng $P(x_j \mid y)$: d đặc trưng nhị phân, hai lớp cần $2d + 1$ tham số. Nếu không có giả định này, ta phải đếm mọi tổ hợp, mà tổ hợp đủ n từ khóa của một thư mới thường không có thư nào trong train. Cái giá: nhân đôi một cột thì bằng chứng ấy bị đếm hai lần và xác suất thành quá tự tin.

HỎI NHANH

Gọi B là bài có lỗi, F là bài bị cảnh báo. Trong 20 bài đã biết là có lỗi, có 18 bài được cảnh báo. Tỷ lệ $18/20$ ước lượng xác suất nào?

A $P(B \cap F)$

B $P(B \mid F)$

C $P(F \mid B)$

D $P(F)$ trên mọi bài

HỎI NHANH

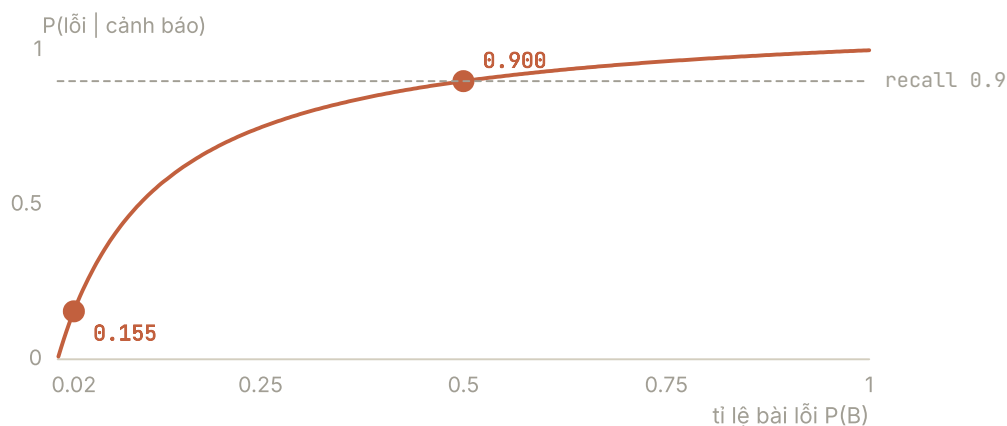
Giữ nguyên detector, bắt được 90% bài lỗi và báo nhầm 10% bài sạch. Đem dùng ở một lớp mà một nửa số bài có lỗi. Ở lớp này, một bài bị cảnh báo có lỗi với xác suất bao nhiêu?

A 0.9

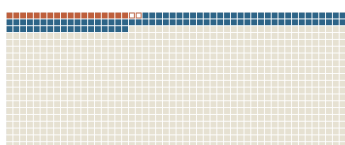
B 0.5

C 0.45

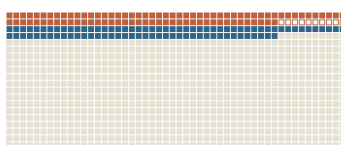
D 0.1552



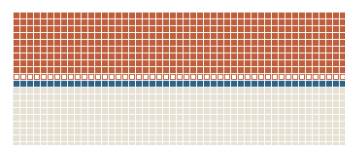
Tỉ lệ lỗi càng thấp, phần cảnh báo đúng càng nhỏ.



20 bài lỗi trên 1000: 116 cảnh báo, 18 đúng (cam đặc), 98 nhầm (xanh); $P(\text{lỗi} \mid \text{cảnh báo}) = 0.155$



100 bài lỗi trên 1000: 180 cảnh báo, 90 đúng (cam đặc), 90 nhầm (xanh); $P(\text{lỗi} \mid \text{cảnh báo}) = 0.500$



500 bài lỗi trên 1000: 500 cảnh báo, 450 đúng (cam đặc), 50 nhầm (xanh); $P(\text{lỗi} \mid \text{cảnh báo}) = 0.900$

Tỉ lệ lỗi càng thấp, phần cảnh báo đúng trong mọi cảnh báo càng nhỏ.

Kỳ vọng và log-loss: trước khi biết đáp án

Một biến Bernoulli $Y \in \{0, 1\}$ có $P(Y = 1) = q$ thì $\mathbb{E}[Y] = q$, vì $0(1 - q) + 1q = q$. Do $Y^2 = Y$, phương sai là $\text{Var}(Y) = q - q^2 = q(1 - q)$. Với $q = 0.02$, hai số lần lượt là 0.02 và 0.0196. Kỳ vọng mô tả trung bình qua nhiều lần rút; một nhóm 50 bài cụ thể vẫn có thể có 0, 1 hay 3 bài lỗi.

Nếu model xuất xác suất p của lớp 1 và nhãn thật là y , binary cross-entropy là

$$L(p, y) = -y \log p - (1 - y) \log(1 - p).$$

Ta dùng logarithm tự nhiên nên loss có đơn vị nats. Khi $y = 1, p = 0.8$ cho loss $-\log 0.8 \approx 0.2231$; dự đoán tự tin sai $p = 0.2$ cho khoảng 1.6094. Accuracy không thấy mức tự tin này nếu hai xác suất vẫn ở cùng một phía threshold.

BCE chính là $-\log$ của xác suất mà model gán cho nhãn thật, tức negative log-likelihood của một mẫu Bernoulli. Vì log tăng ngặt, tham số làm likelihood của cả tập lớn nhất cũng làm tổng log-loss nhỏ nhất: maximum likelihood và tối thiểu log-loss là cùng một bài toán.

Trước khi biết Y , kỳ vọng loss của dự đoán cố định p là $-q \log p - (1 - q) \log(1 - p)$. Giá trị nhỏ nhất đạt tại $p = q$ khi $0 < q < 1$; thay vào cho entropy $H(q)$. Nói cách khác, biết tỉ lệ lớp thật thì xác suất phản ánh tỉ lệ đó là dự đoán tối ưu cho log-loss, chưa chắc là quyết định tối ưu cho mọi chi phí hành động.

Python

```
def expected_log_loss(q, p):
    return -q * np.log(p) - (1 - q) * np.log(1 - p)

q = 0.02
print(f"E[Y]={q:.4f}, Var(Y)={q * (1 - q):.4f}")
print(f"y=1: loss tại p=0.8 là {-np.log(0.8):.4f}; tại p=0.2 là {-np.log(0.2):.4f}")
print(f"E[loss] tại p=0.02: {expected_log_loss(q, 0.02):.4f}")
print(f"E[loss] tại p=0.50: {expected_log_loss(q, 0.50):.4f}")
assert expected_log_loss(q, q) < expected_log_loss(q, 0.5)
```

KẾT QUẢ

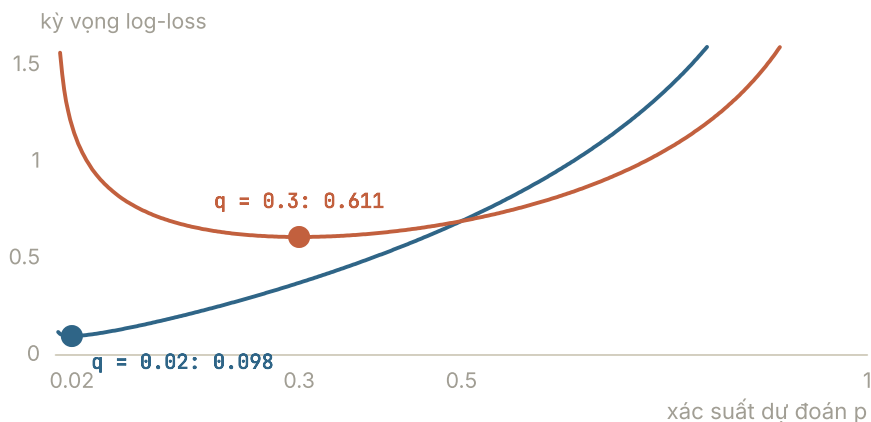
```
E[Y]=0.0200, Var(Y)=0.0196
y=1: loss tại p=0.8 là 0.2231; tại p=0.2 là 1.6094
E[loss] tại p=0.02: 0.0980
E[loss] tại p=0.50: 0.6931
```

Không lấy $\log(0)$ trong code tính tay; nó cho vô cùng. Với logits, thư viện có loss gộp sigmoid hoặc softmax để tính ổn định hơn. Một vector xác suất phải không âm và tổng bằng 1; một vector logits không có hai điều kiện này. Khi trình bày, cần nói model đang xuất đại lượng nào.

HỎI NHANH

Trong tập train, 30% nhãn là 1. Model chỉ được xuất **một** xác suất p cho mọi mẫu. Giá trị p nào cho log-loss trung bình nhỏ nhất?

- A $p = 0$, vì đa số nhãn là 0
- B $p = 0.3$, đúng bằng tỉ lệ nhãn 1
- C $p = 0.5$, vì chưa biết gì về mẫu
- D $p = 0.7$, đúng bằng tỉ lệ nhãn 0



Log loss kỳ vọng nhỏ nhất khi dự đoán đúng bằng tỉ lệ thật.

Đại số tuyến tính: phép nhân mô tả việc gì?

Ta có hai mẫu, mỗi mẫu hai feature, xếp thành hàng của X . Linear model với $\mathbf{w} = (2, -1)$ và $b = 1$ cho

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \hat{\mathbf{y}} = X\mathbf{w} + b = \begin{bmatrix} 1 \\ 3 \end{bmatrix}.$$

Hàng đầu có $1 \cdot 2 + 2 \cdot (-1) + 1 = 1$; hàng sau có $3 \cdot 2 + 4 \cdot (-1) + 1 = 3$. Bias cộng vào mỗi mẫu. Shape của X là $(2, 2)$, của $\mathbf{w}$ là $(2,)$, của kết quả là $(2,)$. Vector một chiều trong NumPy không tự thành cột hai chiều như cách vẽ trên giấy.

Python

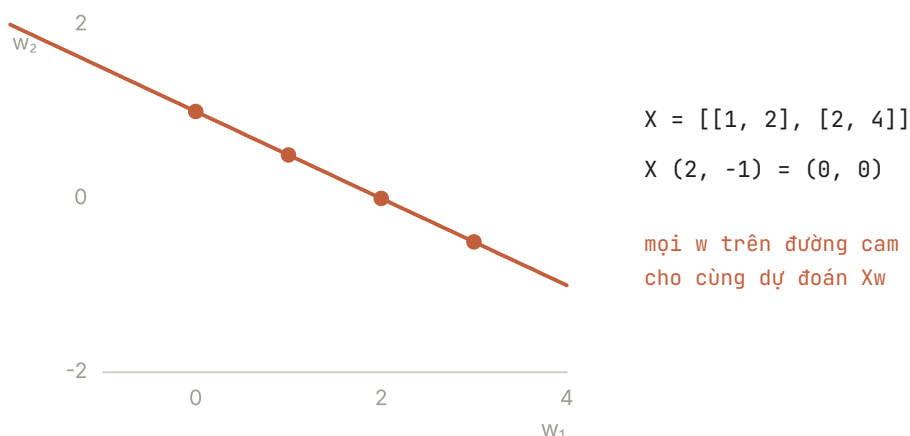
```
X = np.array([[1.0, 2.0], [3.0, 4.0]])
w_linear = np.array([2.0, -1.0])
b_linear = 1.0
prediction = X @ w_linear + b_linear
print("Shape:", X.shape, w_linear.shape, prediction.shape)
print("Dự đoán:", prediction)
assert np.allclose(prediction, [1.0, 3.0])

dependent = np.array([[1.0, 2.0], [2.0, 4.0]])
null_direction = np.array([2.0, -1.0])
print("Hướng không thay đổi dự đoán:", dependent @ null_direction)
print("Hạng ma trận:", np.linalg.matrix_rank(dependent))
```

KẾT QUẢ

```
Shape: (2, 2) (2,) (2,)
Dự đoán: [1. 3.]
Hướng không thay đổi dự đoán: [0. 0.]
Hạng ma trận: 1
```

Ma trận `dependent` có cột thứ hai gấp đôi cột thứ nhất, nên hạng bằng 1. Thêm $t(2, -1)$ vào trọng số không đổi dự đoán, vì tích với hướng ấy bằng 0. Bởi vậy một nghiệm hệ số chưa chắc duy nhất; muốn dùng nghịch đảo hay nói “nghiệm duy nhất”, phải kiểm điều kiện hạng. Đây là lý do ở bài 2.1 ta cần least squares và regularization khi feature phụ thuộc.



Cột thứ hai gấp đôi cột thứ nhất nên hệ số không duy nhất.

Tích vô hướng $\mathbf{u}^T \mathbf{v}$ vừa dùng để dự đoán, vừa đo quan hệ hình học. Với hai vector khác 0, cosine similarity chia tích vô hướng cho tích độ dài. Nó không đổi khi nhân một vector

với số dương; khoảng cách Euclidean thường đổi.

Vì vậy similarity và khoảng cách cần được chọn theo ý nghĩa của embedding, không chỉ theo công thức thuận tay. Vector 0 không có hướng nên cosine của nó không được định nghĩa bằng phép chia thông thường.

HỎI NHANH

x có shape $(5, 3)$, w có shape $(3,)$, còn b là một số. $x @ w + b$ có shape gì?

A $(3,)$

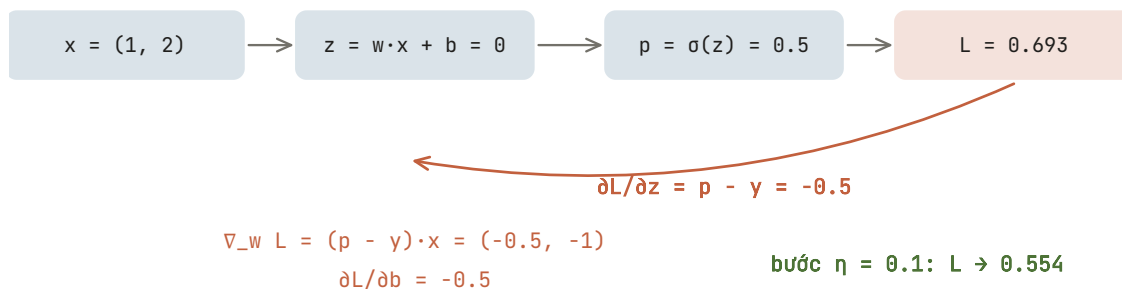
B $(5, 3)$

C $(5, 1)$

D $(5,)$

Gradient: chia sai số về đúng tham số

Giờ dùng một mẫu phân loại: $\mathbf{x} = (1, 2)$, nhãn $y = 1$, trọng số $\mathbf{w} = (0.2, -0.1)$, bias $b = 0$. Logit $z = \mathbf{w}^\top \mathbf{x} + b = 0$, xác suất $p = \sigma(z) = 0.5$. Loss là $\log 2 \approx 0.6931$.



Lượt xuôi cho loss 0.693; một bước gradient hạ còn 0.554.

Chain rule nói đạo hàm qua nhiều phép tính là tích các đạo hàm cục bộ. Với binary cross-entropy và sigmoid ghép đúng, các số hạng rút gọn thành

$$\frac{\partial L}{\partial z} = p - y, \quad \nabla_{\mathbf{w}} L = (p - y)\mathbf{x}, \quad \frac{\partial L}{\partial b} = p - y.$$

Vậy gradient của $\mathbf{w}$ là $(-0.5, -1)$, của b là -0.5 . Gradient descent với $\eta = 0.1$ cho $\mathbf{w}' = (0.25, 0)$, $b' = 0.05$. Logit mới bằng 0.3 , xác suất tăng lên khoảng 0.5744 , loss giảm còn khoảng 0.5544 . Nhân là 1 nên dấu thay đổi có nghĩa: model cần tăng xác suất lớp 1.

Ta gộp $\theta = (w_1, w_2, b)$ để kiểm cả ba thành phần bằng sai phân trung tâm. `logaddexp(0, z)` tính $\log(1 + e^z)$ ổn định; loss từ logit là biểu thức đó trừ yz .

Python

```
x = np.array([1.0, 2.0])
y = 1.0
theta = np.array([0.2, -0.1, 0.0])

def logistic_loss(parameters):
    z = x @ parameters[:2] + parameters[2]
    return float(np.logaddexp(0.0, z) - y * z)

def logistic_gradient(parameters):
    z = x @ parameters[:2] + parameters[2]
    p = 1.0 / (1.0 + np.exp(-z))
    return np.append((p - y) * x, p - y)

gradient = logistic_gradient(theta)
h = 1e-5
numerical = np.zeros_like(theta)
for j in range(len(theta)):
    shift = np.zeros_like(theta)
    shift[j] = h
    numerical[j] = (logistic_loss(theta + shift) - logistic_loss(theta - shift)) / (2 * h)
new_theta = theta - 0.1 * gradient
print("Gradient:", gradient)
print("Tham số mới:", new_theta)
print(f"Loss: {logistic_loss(theta):.4f} -> {logistic_loss(new_theta):.4f}")
print("Sai phân khớp:", np.allclose(gradient, numerical, atol=1e-8))
assert np.allclose(gradient, [-0.5, -1.0, -0.5])
assert logistic_loss(new_theta) < logistic_loss(theta)
```

KẾT QUẢ

Gradient: [-0.5 -1. -0.5]

Tham số mới: [0.25 0. 0.05]

Loss: 0.6931 -> 0.5544

Sai phân khớp: True

Gradient cho xấp xỉ bậc nhất $L(\boldsymbol{\theta} + \boldsymbol{\Delta}) \approx L(\boldsymbol{\theta}) + \nabla L^\top \boldsymbol{\Delta}$ khi $\boldsymbol{\Delta}$ nhỏ. Đạo hàm âm theo một tham số nghĩa là tăng tham số ấy một chút thì loss giảm, nên bước descent $\boldsymbol{\Delta} = -\eta \nabla L$ làm nó tăng.

Tình huống loss tăng ở đầu bài thường có một trong hai nguyên nhân: cộng gradient thay vì trừ, hoặc bước quá lớn, vượt khỏi vùng mà xấp xỉ tuyến tính còn dùng được.

HỎI NHANH

Tại $w_1 = 3$, đạo hàm $\partial L / \partial w_1 = -2$. Tăng w_1 lên 3.001, giữ nguyên các tham số khác. Loss thay đổi xấp xỉ bao nhiêu?

A Giảm khoảng 0.002

B Tăng khoảng 0.002

C Giảm khoảng 2

D Không đổi, vì bước quá nhỏ

Gradient bằng 0 là điểm dừng, không tự chứng minh đó là cực tiểu toàn cục của một mạng phi tuyến. Với loss lồi, như MSE của linear regression hay log-loss của logistic regression, mọi điểm dừng đều là cực tiểu toàn cục; cần nói rõ lớp hàm đang xét.

Nếu thêm penalty $\lambda \|\mathbf{w}\|^2$ thì gradient theo $\mathbf{w}$ có thêm $2\lambda \mathbf{w}$. Nếu đề dùng $\lambda \|\mathbf{w}\|^2 / 2$, số 2 biến mất. Trên một batch, loss lấy trung bình cũng làm gradient có hệ số $1/n$. Viết rõ loss trước khi đạo hàm là cách tránh chép nhầm quy ước.

HỎI NHANH

Gradient theo bias là -0.6 , learning rate là 0.2 , và bias ban đầu là 1 . Sau một bước gradient descent, bias mới bằng bao nhiêu?

A -0.12

B 0.88

C 1.12

D 1.6

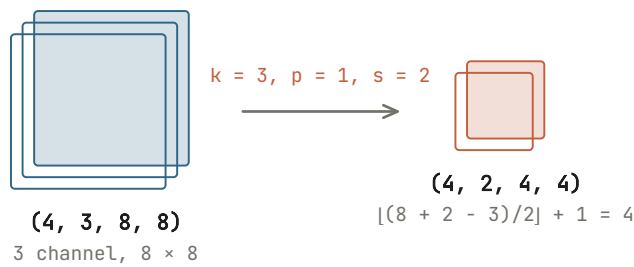
Kiến trúc mạng: đếm shape trước khi đếm tham số

Một câu lý thuyết về CNN thường chỉ cần phân biệt số mẫu, channel và không gian. Với convolution không dilation, input cao H , kernel k , padding p , stride s , chiều cao đầu ra là

$$H_{\text{out}} = \left\lfloor \frac{H + 2p - k}{s} \right\rfloor + 1.$$

Input $(N, C, H, W) = (4, 3, 8, 8)$, hai output channel, $k = 3, p = 1, s = 2$ cho $H_{\text{out}} = W_{\text{out}} = 4$. Shape output là $(4, 2, 4, 4)$. Mỗi output channel dùng $3 \cdot 3 \cdot 3 = 27$ weight và một bias, tổng $2(27 + 1) = 56$ tham số.

Tổng quát, Conv2d có $C_{\text{out}}(C_{\text{in}}k^2 + 1)$ tham số, bỏ số 1 khi `bias=False`; cách đặt này thường gặp khi BatchNorm đứng ngay sau, vì BatchNorm đã có tham số dịch riêng. Batch size và số vị trí ảnh không nhân vào số tham số: các weight được dùng lại ở nhiều vị trí.



tham số

$$2 \times (3 \cdot 3 \cdot 3 + 1) = 56$$

batch 4 không nhân vào

Conv2d: input (4, 3, 8, 8) ra (4, 2, 4, 4), 56 tham số.

Python

```
import torch

torch.manual_seed(0)
conv = torch.nn.Conv2d(3, 2, kernel_size=3, stride=2, padding=1)
images = torch.zeros(4, 3, 8, 8)
with torch.no_grad():
    conv_output = conv(images)
parameter_count = sum(parameter.numel() for parameter in conv.parameters())
print("Output shape:", tuple(conv_output.shape))
print("Số tham số:", parameter_count)
assert tuple(conv_output.shape) == (4, 2, 4, 4)
assert parameter_count == 56
```

KẾT QUẢ

Output shape: (4, 2, 4, 4)
Số tham số: 56

Một forward trên tensor toàn 0 đủ để kiểm shape và số tham số; không cần train. Nếu đề có dilation d , thay k bằng effective size $d(k - 1) + 1$. Kernel 1×1 vẫn có $C_{in}C_{out}$ weight: nó không nhìn lân cận nhưng trộn các channel tại từng vị trí. Pooling thường không có weight học được; residual addition yêu cầu shape hai nhánh khớp hoặc có projection để đưa về cùng shape.

HỎI NHANH

Input (8, 16, 32, 32) đi qua Conv2d(16, 32, kernel_size=3, padding=1), stride 1, có bias. Output có shape gì, và lớp có bao nhiêu tham số?

A (8, 32, 30, 30), 4640 tham số

B (8, 16, 32, 32), 4640 tham số

C (8, 32, 32, 32), 4608 tham số

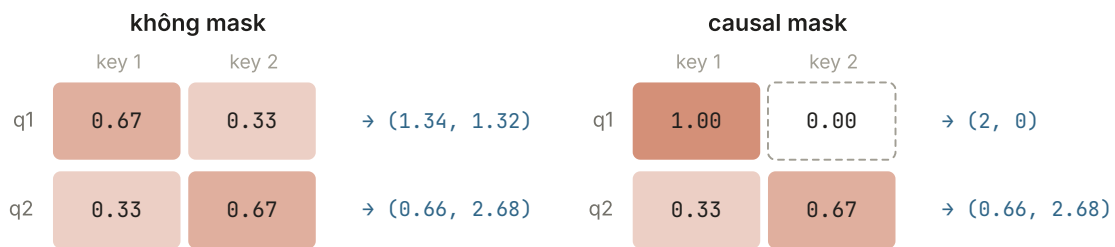
D (8, 32, 32, 32), 4640 tham số

Attention: một hàng là một cách lấy trung bình

Với một head, Q có shape (T_q, d_k) , K có shape (T_k, d_k) , V có shape (T_k, d_v) . Ma trận score $QK^\top / \sqrt{d_k}$ có shape (T_q, T_k) . Softmax theo trục key tạo một phân phối cho **mỗi query**, rồi nhân V để lấy tổng có trọng số:

$$O = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V.$$

Ta chọn $Q = K = I_2$, tức ma trận đơn vị 2 chiều, và hai value là $(2, 0)$, $(0, 4)$. Score của query đầu là $(1/\sqrt{2}, 0)$. Softmax cho trọng số khoảng $(0.6698, 0.3302)$, nên output đầu là $(1.3395, 1.3210)$. Đây là vector giá trị pha trộn, không phải index của một token được chọn.



mỗi hàng là một phân phối trên key, nhân với value (2, 0) và (0, 4)

Mỗi hàng trọng số cộng thành 1; causal mask chặn key tương lai.

Python

```
Q = np.eye(2)
K = np.eye(2)
values = np.array([[2.0, 0.0], [0.0, 4.0]])

def row_softmax(scores):
    shifted = scores - scores.max(axis=1, keepdims=True)
    exponentials = np.exp(shifted)
    return exponentials / exponentials.sum(axis=1, keepdims=True)

scores = Q @ K.T / np.sqrt(Q.shape[1])
weights = row_softmax(scores)
attention_output = weights @ values
print("Trọng số:\n", weights.round(4))
print("Output:\n", attention_output.round(4))
assert np.allclose(weights.sum(axis=1), 1.0)

masked_scores = scores.copy()
masked_scores[0, 1] = -np.inf # query đầu không nhìn token tương lai
causal_weights = row_softmax(masked_scores)
print("Output đầu với causal mask:", (causal_weights @ values)[0])
assert np.allclose(causal_weights[0], [1.0, 0.0])
```

KẾT QUẢ

Trọng số:

```
[[0.6698 0.3302]
 [0.3302 0.6698]]
```

Output:

```
[[1.3395 1.321 ]
 [0.6605 2.679 ]]
```

Output đầu với causal mask: [2. 0.]

Đặt mask vào score **trước** softmax làm xác suất bị cấm bằng 0 rồi chuẩn hóa lại phần được phép. Đặt một weight về 0 sau softmax mà không chuẩn hóa lại làm tổng hàng nhỏ hơn 1. Nếu một query bị mask toàn bộ key, softmax không có phân phối hợp lệ; cần xử lý theo thiết kế của model, không mong phép tính tự hiểu padding.

Trong một layer thật, $Q = XW_Q$, $K = XW_K$, $V = XW_V$ với X shape (T, d_{model}) , rồi các head được nối lại và đi qua một phép chiếu ra. Tham số nằm ở các ma trận chiếu, nên số tham số không phụ thuộc độ dài chuỗi T ; ngược lại, ma trận score (T, T) tăng theo T^2 . Đếm tham số và đếm bộ nhớ là hai câu hỏi khác nhau.

HỎI NHANH

Q có shape $(3, 2)$, K có shape $(5, 2)$, V có shape $(5, 4)$. Output của một head có shape gì?

A $(5, 4)$

B $(3, 5)$

C $(3, 4)$

D $(2, 4)$

HỎI NHANH

Query thứ hai chỉ được nhìn key 1 và key 2. Score cho ba key là $(0, \log 3, \log 2)$, với log tự nhiên. Đặt score của key 3 thành $-\infty$ trước khi softmax. Bộ trọng số thu được là gì?

A $(1/6, 3/6, 2/6)$

B $(1/4, 3/4, 0)$

C $(1/6, 3/6, 0)$

D $(0, 1, 0)$

Đánh giá: trở lại detector đầu bài

Bảng 1000 bài có $TP = 18$, $FN = 2$, $FP = 98$, $TN = 882$. Recall bằng $TP / (TP + FN) = 0.9$, trả lời “bắt được bao nhiêu bài lỗi?”. Precision bằng $TP / (TP + FP) = 18 / 116 \approx 0.1552$, trả lời “cảnh báo đáng tin tới đâu?”. Hai số đúng là hai xác suất có điều kiện đã gặp ở đầu bài.

Accuracy bằng $(TP + TN) / 1000 = 0.9$. Nhưng baseline luôn đoán sạch có accuracy 0.98 và recall 0. Không thể từ một số 0.98 kết luận baseline giải quyết mục tiêu tìm lỗi tốt hơn; cũng không thể bỏ qua 98 cảnh báo nhầm vì recall cao. Metric cần phản ánh yêu cầu và chi phí kiểm thủ công của bài.

F1 dùng trung bình điều hòa của precision và recall, tương đương

$$F1 = \frac{2TP}{2TP + FP + FN} = \frac{36}{136} \approx 0.2647.$$

Để đối chiếu với scikit-learn, ta dựng lại vector nhãn và vector dự đoán đúng theo bốn ô của bảng.

Python

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

labels = np.array([1] * (tp + fn) + [0] * (fp + tn))
predicted = np.array([1] * tp + [0] * fn + [1] * fp + [0] * tn)
accuracy = (tp + tn) / total
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1 = 2 * tp / (2 * tp + fp + fn)
library = np.array([
    accuracy_score(labels, predicted), precision_score(labels, predicted),
    recall_score(labels, predicted), f1_score(labels, predicted),
])
assert np.allclose(library, [accuracy, precision, recall, f1])
print(f"Accuracy={accuracy:.4f}, precision={precision:.4f}, recall={recall:.4f}, F1={f1:.4f}")
print(f"Baseline sạch: accuracy={(fp + tn) / total:.4f}, recall=0.0000")
print("Tính tay khớp thư viện:", True)
```

KẾT QUẢ

Accuracy=0.9000, precision=0.1552, recall=0.9000, F1=0.2647

Baseline sạch: accuracy=0.9800, recall=0.0000

Tính tay khớp thư viện: True

	cảnh báo	không
có lỗi	18	2
sạch	98	882

recall $18/20 = 0.900$

precision $18/116 = 0.155$

accuracy $900/1000 = 0.900$

F1 $36/136 = 0.265$

Luôn đoán sạch: acc 0.98, recall 0

Recall 0.9 nhưng precision chỉ 0.155; luôn đoán sạch vẫn có accuracy 0.98.

Khi chấm model thật, chỉ dùng train để fit tham số; chọn threshold, kiến trúc và hyperparameter bằng validation phù hợp, rồi giữ test để đánh giá cuối.

Với nhiều bài của cùng một người hoặc cùng một nguồn, random split có thể lọt kiểu viết giữa train và validation; lúc ấy cần group split. Nếu mục tiêu dự đoán ở tương lai, cách chia theo thời gian có thể phù hợp hơn. Stratification giữ tỉ lệ lớp nhưng không tự chữa leakage theo nhóm.

HỎI NHANH

Mỗi học sinh nộp nhiều bài, và bài của cùng một người có mặt ở cả train lẫn validation. Mục tiêu là dự đoán cho học sinh mới. Stratification theo nhãn có xử lý được rủi ro này không?

- A Không, cần chia theo người, tức group split
- B Có, vì tỉ lệ nhãn ở hai phần đã bằng nhau
- C Có, nếu model đủ nhỏ để không nhớ được từng người
- D Không, cần chia theo thời gian nộp bài

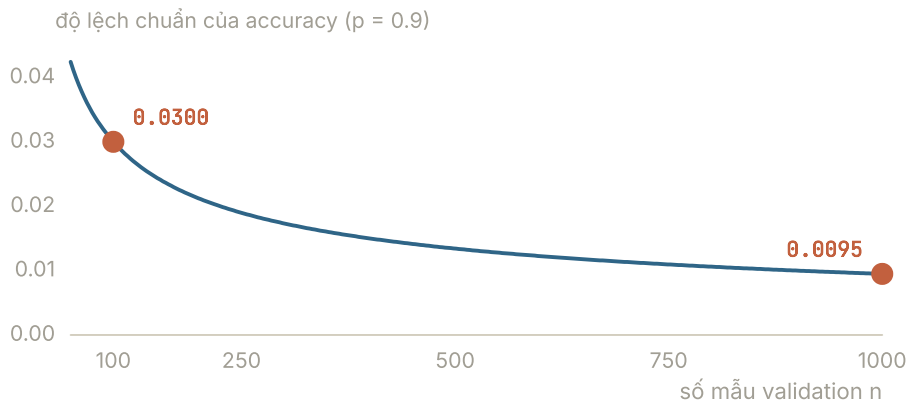
Một điểm trên tập nhỏ có sai số lấy mẫu. Nếu 1000 kết quả đúng hay sai thật sự độc lập và cùng xác suất đúng 0.9, độ lệch chuẩn của accuracy mẫu là $\sqrt{0.9 \cdot 0.1/1000} \approx 0.0095$. Giả thiết độc lập là phần quan trọng: nếu nhiều mẫu đến từ cùng một người, chúng tương quan và độ bất định thật lớn hơn con số này.

Hai model nên được so trên cùng split và đọc loại lỗi của chúng, không chỉ xếp hạng một điểm duy nhất.

HỎI NHANH

Với 1000 mẫu độc lập và accuracy thật là 0.9, độ lệch chuẩn của accuracy đo được khoảng 0.0095. Nếu validation chỉ có 100 mẫu thì khoảng bao nhiêu?

- A 0.00095
- B 0.0095
- C 0.03
- D 0.095



Giảm số mẫu 10 lần làm độ lệch chuẩn của accuracy tăng $\sqrt{10}$ lần.

Khi đi thi

- Viết đại lượng cần tìm trước khi thế số, nhất là điều kiện nằm ở phía nào: $P(B | F)$ hay $P(F | B)$.
- Đạo hàm từ loss viết đầy đủ, ghi rõ hệ số $1/n$; rồi kiểm dấu trên một mẫu.
- Bài kiến trúc thì tính shape từng lớp trước, rồi mới đếm tham số.

Tóm tắt

- Điều kiện xác suất quyết định mẫu số; Bayes, confusion matrix và precision có thể mô tả cùng một câu hỏi.
- Log-loss là negative log-likelihood; xác suất khớp tỉ lệ thật cho log-loss kỳ vọng nhỏ nhất.
- Đại số tuyến tính mô tả phép gộp feature, shape và tính duy nhất của nghiệm. Không nghịch đảo khi chưa kiểm điều kiện.
- Gradient đi qua từng phép tính; loss, dấu và phép lấy trung bình phải nhất quán với code.
- Convolution dùng lại tham số; attention chuẩn hóa theo key cho mỗi query. Shape giúp kiểm cả hai.

- Metric trả lời một mục tiêu cụ thể. Một điểm validation chỉ có ý nghĩa cùng cách chia, baseline và phạm vi dữ liệu.

Kiểm tra cuối bài

CÂU 1

Trong 1000 bài nộp, có 200 bài dùng đệ quy, gọi là biến cố A , 100 bài bị quá thời gian, gọi là biến cố T , và 40 bài vừa dùng đệ quy vừa quá thời gian. A và T có độc lập không?

- A Có, vì 40 bài chỉ là một phần nhỏ của 1000 bài
- B Có, vì 40 nhỏ hơn cả 100 lẫn 200
- C Chưa kết luận được khi chưa biết quan hệ nhân quả
- D Không, vì $P(T | A) = 0.2$ khác với $P(T) = 0.1$

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 62

CÂU 2

Hàm $f(\theta)$ có argmax duy nhất là θ^* , và có thể nhận giá trị âm, như một log-likelihood. Hàm nào **không** chắc có cùng argmax θ^* ?

A $f(\theta) + 2$

B $3f(\theta)$

C $e^{f(\theta)}$

D $f(\theta)^2$

PHÒNG THEO CMU 10-601 · Exam 1 Practice · Q5.1

CÂU 3

Hai model có cùng accuracy 95% trên một tập nhị phân. Model A luôn báo xác suất 0.999 cho lớp nó chọn, còn model B luôn báo 0.95. Log-loss trung bình, với log tự nhiên, của hai model là bao nhiêu?

A A khoảng 0.001, B khoảng 0.05

B A khoảng 0.35, B khoảng 0.20

C A khoảng 0.20, B khoảng 0.35

D Bằng nhau, vì cùng accuracy

CÂU 4

Linear regression với MSE, và ma trận dữ liệu có cột thứ hai gấp đôi cột thứ nhất. Kết luận nào đúng?

- A Không có cực tiểu toàn cục, vì $X^T X$ không khả nghịch
- B Có đúng một cực tiểu toàn cục, vì MSE là hàm lồi
- C Có vô số cực tiểu toàn cục, cùng cho một dự đoán trên train
- D GD có thể kẹt ở một cực tiểu địa phương kém hơn

PHÒNG THEO CMU 10-601 · Exam 2 Practice · Q1.2

CÂU 5

Logistic regression trên ba điểm: $(1, 1)$ nhãn 0, $(1, 0)$ nhãn 1, $(0, 0)$ nhãn 0. Model 1 là $\sigma(w_1 x_1 + w_2 x_2)$, còn model 2 có thêm bias w_0 . Đổi nhãn của $(0, 0)$ thành 1, rồi chạy GD từ cùng khởi tạo, với cùng learning rate. Chuyện gì xảy ra?

- A Model 1 đi đúng quỹ đạo cũ, còn model 2 đi khác
- B Model 1 đi khác, còn model 2 đi đúng quỹ đạo cũ
- C Cả hai model đều đi đúng quỹ đạo cũ
- D Cả hai model đều đi khác, vì dữ liệu đã đổi

PHÒNG THEO CMU 10-601 · Exam 2 Practice · Q2.2

CÂU 6

Logistic regression viết mục tiêu dạng tổng như scikit-learn: $J(\mathbf{w}) = \sum_{i=1}^m \ell_i + \frac{\lambda}{2} \|\mathbf{w}_1\|^2$, trong đó bias $w[0]$ không bị phạt. Code tối ưu J/m , và đã có $g = \mathbf{x} \cdot \mathbf{w}$ @ $(p - y) / m$. Dòng nào cộng phần phạt vào cho đúng?

A `g += lam * w`

B `g[1:] += lam * w[1:]`

C `g[1:] += (lam / m) * w[1:]`

D `g += (lam / m) * w`

PHÒNG THEO NOAI 2026 Singapore · Chung kết · Câu 5

CÂU 7

Đội dự đoán số khán giả $g = \boldsymbol{\theta}^\top \mathbf{x}$, và chịu chi phí $L = \alpha g + \beta \max(0, y - g)$: thuê g ghế với giá α mỗi ghế, và mất β cho mỗi người phải từ chối. Với $\alpha = 1$, $\beta = 4$, $\mathbf{x} = (1, 3)$, $y = 20$, và hiện tại $g = 12$, gradient $\nabla_{\boldsymbol{\theta}} L$ là gì?

A $(1, 3)$

B $(-3, -9)$

C $(-4, -12)$

D $(5, 15)$

PHÒNG THEO MIT 6.036 · Final, Spring 2022 · Q5(d)

CÂU 8

Một bottleneck block của ResNet-50, trong layer3, có ba convolution không bias, mỗi cái theo sau bởi BatchNorm: 1×1 từ 1024 xuống 256 channel, 3×3 từ 256 sang 256, và 1×1 từ 256 lên 1024. Ba convolution này có tổng cộng bao nhiêu tham số?

A 1 114 112

B 1 115 648

C 589 824

D 5 308 416

PHÒNG THEO USA-NA-AIO 2026 · Vòng 1 · Part 8.3

CÂU 9

Một layer multi-head attention có $d_{\text{model}} = 64$, $H = 4$ head, mỗi head có $d = 16$. Ba phép chiếu Q, K, V đi từ d_{model} lên Hd , phép chiếu ra đi từ Hd về d_{model} , tất cả đều không bias. Thêm relative positional encoding: mỗi head học một vector $\mathbf{r}_{j-i} \in \mathbb{R}^d$ cho mỗi độ lệch $j - i$, với chuỗi dài $T = 10$. Tổng số tham số là bao nhiêu?

A 13 504

B 16 688

C 17 024

D 17 600

PHÒNG THEO USA-NA-AIO 2026 · Vòng 2 · Ngày 1 · Part 1.3.1

CÂU 10

Detector A có precision 0.5 và recall 0.5. Detector B có precision 0.9 và recall 0.1. Trung bình cộng hai chỉ số của cả hai đều bằng 0.5. F1 của mỗi detector bằng bao nhiêu?

A A được 0.50, B được 0.50

B A được 0.50, B được 0.18

C A được 0.25, B được 0.09

D A được 0.50, B được 0.90

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 68

Đọc thêm

- Deisenroth, Faisal, Ong — [Mathematics for Machine Learning](#), chương 2, 5, 6 và 7. Đọc để nối đại số tuyến tính, đạo hàm nhiều biến, xác suất và tối ưu trong một hệ ký hiệu.
- Boyd, Vandenberghe — [Introduction to Applied Linear Algebra](#). Dùng các ví dụ vector, hạng và least squares để luyện giải thích một phép nhân ma trận bằng lời.
- James và cộng sự — [An Introduction to Statistical Learning with Applications in Python](#), chương 2, 4 và 5. Đọc lỗi tổng quát hóa, xác suất phân loại và validation để đặt phép tính metric vào đúng quy trình.
- Ma, Ng — [CS229 Lecture Notes](#). Đọc các phần logistic regression và neural networks để đối chiếu chain rule với loss từ logits.
- Vaswani và cộng sự — [Attention Is All You Need](#), mục 3.2. Đọc công thức scaled dot-product attention và lý do mask trước softmax.
- [PyTorch — Conv2d](#) và [scikit-learn — Model evaluation](#). Dùng để kiểm quy ước shape, bias, positive class và averaging của thư viện.