

Logistic regression và softmax

☑ Trắc nghiệm 39 ☑ Thực hành 2 ⬇ PDF bài

Một bộ lọc thư rác nhận email “Chúc mừng! Bạn đã trúng iPhone, bấm vào đây để nhận”. Nó không cần biết chắc đây là spam. Điều nó cần là một con số kiểu “xác suất spam là 0.97”, để người dùng tự quyết định chặn từ mức nào. Linear regression của bài 2.1 cho ra một số thực bất kỳ, không phải xác suất. Bài này sửa đúng chỗ đó: giữ phần tuyến tính, nhưng ép đầu ra về khoảng $(0, 1)$ và học bằng một hàm mất mát hợp với xác suất. Kết quả là logistic regression, mô hình phân loại được dùng nhiều nhất trong thực tế và cũng chính là một neuron của mạng neural ở chương sau.

MỤC TIÊU

- Biết vì sao linear regression không dùng cho phân loại.
- Viết logistic regression bằng NumPy và so với scikit-learn.
- Chọn ngưỡng quyết định khác 0.5 khi cần.
- Dùng softmax cho bài nhiều lớp.

Dữ liệu: lọc thư rác

Mỗi email được mô tả bằng hai con số: số đường link và số từ quảng cáo như “miễn phí”, “trúng thưởng”, “khuyến mãi”. Nhãn là 1 nếu là spam, 0 nếu là thư thường. Giống bài 2.1, ta tự sinh dữ liệu từ một quy luật đã biết để sau này đối chiếu xem mô hình học được gì. Quy luật dùng hàm sigmoid, sẽ được giải thích ngay ở mục sau; lúc này chỉ cần biết email càng nhiều link và từ quảng cáo thì càng dễ là spam.

Python

```
import numpy as np

rng = np.random.default_rng(20)
n = 400

links = rng.poisson(2.0, n)           # số đường link
kw = rng.poisson(1.5, n)             # số từ quảng cáo
```

```

z_true = 1.1 * links + 0.9 * kw - 4.5 # quy luật thật, mô hình không được
biết
y = (rng.random(n) < 1 / (1 + np.exp(-z_true))).astype(float) # 1 = spam

X = np.column_stack([links, kw]).astype(float) # (400, 2)
X_tr, X_te, y_tr, y_te = X[:300], X[300:], y[:300], y[300:]
print(X_tr.shape, X_te.shape, " tỉ lệ spam:", y.mean())
print(X[:4], y[:4])

```

KẾT QUẢ

```

(300, 2) (100, 2)  tỉ lệ spam: 0.3125
[[1. 1.]
 [0. 0.]
 [2. 0.]
 [0. 3.]] [0. 0. 0. 0.]

```

Khoảng một phần ba số email là spam. 300 email đầu dùng để học, 100 email sau để kiểm tra.

Vì sao không dùng linear regression?

Cách nghĩ đầu tiên: nhãn là số 0 và 1, vậy cứ chạy linear regression rồi coi đầu ra là xác suất.

Python

```

from sklearn.linear_model import LinearRegression

lin = LinearRegression().fit(X_tr, y_tr)
p_lin = lin.predict(X_te)
print("dự đoán nhỏ nhất:", p_lin.min().round(2), " lớn nhất:", p_lin.max().round(2))
print("số email có 'xác suất' ngoài [0, 1]:", np.sum((p_lin < 0) | (p_lin > 1)))

```

KẾT QUẢ

```

dự đoán nhỏ nhất: -0.2  lớn nhất: 1.34
số email có 'xác suất' ngoài [0, 1]: 54

```

Vấn đề thứ nhất hiện ngay trong output: một phần đáng kể dự đoán nằm ngoài [0, 1]. “Xác suất -0.2” không có nghĩa gì.

Vấn đề thứ hai nghiêm trọng hơn. Để dễ nhìn, tạm chỉ dùng số link. Linear regression cho một đường thẳng; email được coi là spam khi đường thẳng vượt 0.5, tức số link vượt một ngưỡng nào đó. Giờ thêm vào tập train 20 email “siêu spam” có từ 15 đến 24 link. Những email này rõ ràng là spam, phân loại chúng chẳng khó gì, nên lẽ ra chúng không được làm thay đổi ranh giới.

Python

```
def link_threshold(model):
    """Số link tại đó đường thẳng cắt mức 0.5."""
    return (0.5 - model.intercept_) / model.coef_[0]

def accuracy_at(t):
    """Accuracy trên tập test khi coi email có từ t link trở lên là spam."""
    return np.mean((X_te[:, 0] >= t) == y_te)

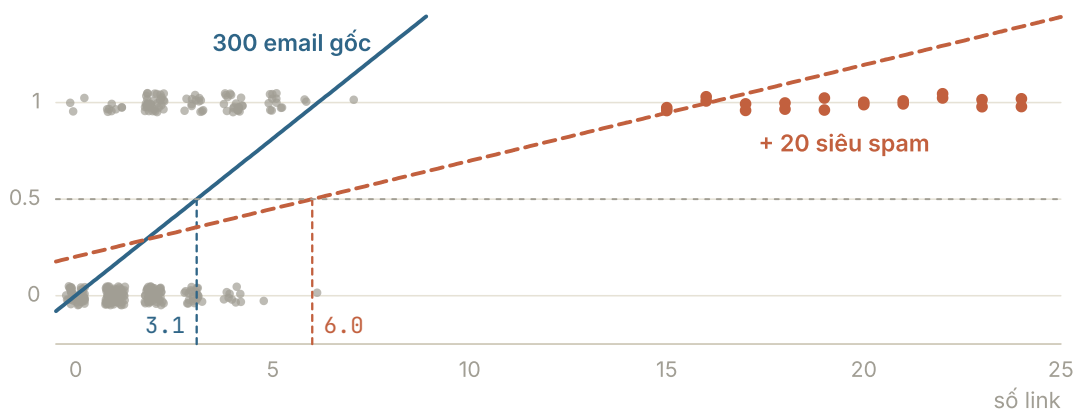
L_tr = X_tr[:, :1] # chỉ số link, shape (300, 1)
extra = np.repeat(np.arange(15, 25), 2)[: , None] # 20 email siêu spam
L_more, y_more = np.vstack([L_tr, extra]), np.append(y_tr, np.ones(20))

for name, L, yy in [("300 email gốc", L_tr, y_tr), ("thêm 20 siêu spam", L_more,
y_more)]:
    t = link_threshold(LinearRegression().fit(L, yy))
    print(f"{name}: ngưỡng = {t:.2f} link    accuracy test = {accuracy_at(t):.2f}")
```

KẾT QUẢ

```
300 email gốc      : ngưỡng = 3.07 link    accuracy test = 0.78
thêm 20 siêu spam: ngưỡng = 6.00 link    accuracy test = 0.68
```

Ngưỡng nhảy từ khoảng 3 link lên 6 link, và accuracy trên tập test tụt 10 điểm phần trăm. Lý do nằm ở hàm mất mát bình phương: email 24 link có nhãn 1 nhưng đường thẳng dự đoán cho nó một số lớn hơn 1 nhiều, nên bị tính là “sai nặng”. Để giảm cái sai vô lý đó, đường thẳng phải thoải xuống, kéo theo ngưỡng dịch sang phải. Mô hình bị phạt vì quá chắc chắn ở chỗ đúng.



Thêm 20 email siêu spam, ngưỡng của linear regression dịch từ 3.1 lên 6 link.

Hai vấn đề có chung một gốc: đầu ra không bị chặn trong $(0, 1)$, và hàm mất mát không hợp với xác suất. Hai mục tiếp theo sửa lần lượt từng cái.

HỎI NHANH

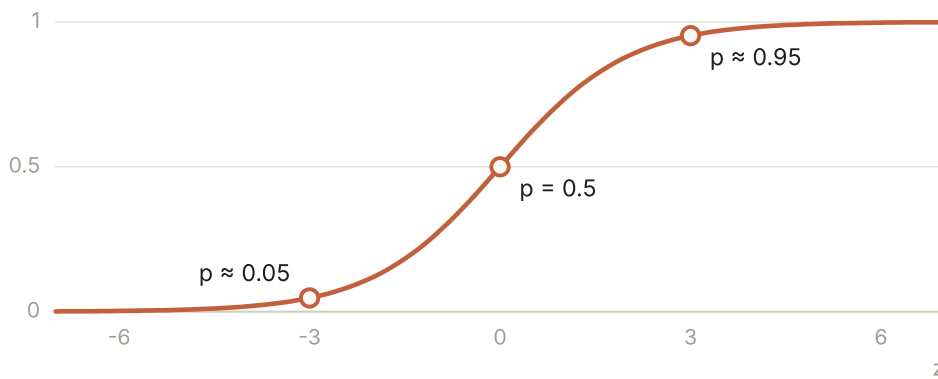
Thêm vào train 20 email có từ 15 đến 24 link, email nào cũng là spam thật. Vì sao ngưỡng của linear regression lại dịch từ khoảng 3 link lên 6 link?

- A Email dễ, đã đúng nhãn, vẫn bị phạt vì dự đoán vượt quá 1
- B Tỷ lệ spam trong train tăng nên ngưỡng phải tăng theo
- C 20 email ấy thật ra bị gán nhãn sai nên làm hỏng mô hình
- D Mô hình bị overfit vì có thêm mẫu ở vùng xa

Sigmoid và log-odds

Giữ phần tuyến tính $z = \mathbf{w}^\top \mathbf{x} + b$ như cũ, rồi ép nó về khoảng $(0, 1)$ bằng hàm sigmoid:

$$p = P(y = 1 \mid \mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}.$$



Sigmoid nhận mọi số thực z và trả về một xác suất trong $(0, 1)$.

Sigmoid nhận mọi số thực và trả về một số trong $(0, 1)$. $z = 0$ cho $p = 0.5$; $z = 3$ cho $p \approx 0.95$; $z = -3$ cho $p \approx 0.05$. Đi xa khỏi 0 thì đường cong phẳng dần: tăng z từ 10 lên 20 gần như không đổi p . Tên có chữ “hồi quy” vì mô hình hồi quy ra một xác suất, nhưng việc của nó là phân loại: cắt xác suất ở một ngưỡng là có nhãn.

Để hiểu z là gì, giải ngược phương trình trên theo z :

$$z = \log \frac{p}{1-p}.$$

Tỉ số $\frac{p}{1-p}$ gọi là odds, cách nói “mấy ăn một” quen thuộc trong cá cược: $p = 0.75$ thì odds là 3, nghĩa là khả năng spam gấp 3 khả năng không spam. z là log của odds, gọi là log-odds hay logit. Vậy logistic regression là một mô hình tuyến tính, nhưng tuyến tính cho log-odds chứ không cho xác suất. Đi ngược lại cũng dễ: $z = \ln 3$ cho odds 3, tức $p = 3/4$. Muốn lúc đầu mô hình đoán khoảng 0.01 cho bài chỉ có 1% mẫu dương thì đặt bias bằng $\ln(0.01/0.99) \approx -4.6$.

Hệ quả trực tiếp: tăng x_j thêm một đơn vị thì z tăng w_j , nên odds nhân thêm e^{w_j} . Bài báo y khoa hay viết “hút thuốc làm odds mắc bệnh tăng gấp 3” chính là đọc hệ số của logistic regression theo cách này.

HỎI NHANH

Mô hình cho một email $p = 0.8$. Odds spam của email ấy bằng bao nhiêu?

A 4

B 0.8

C 0.2

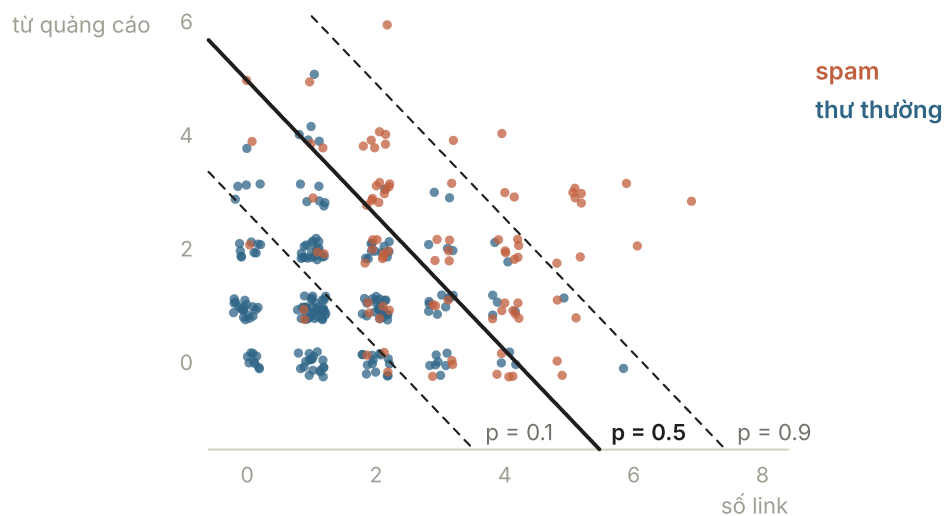
D 1.25

Ranh giới quyết định

Muốn ra nhãn thì cần xác suất ở một ngưỡng, mặc định là 0.5. Vì $\sigma(z) \geq 0.5$ khi và chỉ khi $z \geq 0$, ranh giới giữa hai lớp là tập điểm có

$$\mathbf{w}^\top \mathbf{x} + b = 0.$$

Với hai đặc trưng, đó là một đường thẳng; với d đặc trưng, một siêu phẳng. Sigmoid chỉ quyết định xác suất thay đổi nhanh hay chậm khi đi xuyên qua ranh giới, không làm ranh giới cong đi. Logistic regression vì vậy vẫn là một bộ phân loại tuyến tính. Muốn ranh giới cong thì làm như bài 2.1: thêm đặc trưng x_1^2, x_2^2, x_1x_2 , khi đó ranh giới thẳng trong không gian đặc trưng mới thành đường cong trong mặt phẳng cũ. Chẳng hạn thêm một cột $x_1^2 + x_2^2$ thì ranh giới là đường tròn, hoặc vẫn là đường thẳng nếu hệ số của cột đó bằng 0.



Các mức xác suất là những đường thẳng song song; $p = 0.5$ là ranh giới.

Với một đặc trưng, ranh giới là một điểm: $x = -b/w$. Nhân cả $\mathbf{w}$ và b với một số dương không đổi dấu của z , nên không đổi nhãn nào; chỉ xác suất dốc hơn hay thoải hơn quanh ranh giới. Khoảng cách từ điểm $\mathbf{x}$ tới ranh giới là $|\mathbf{w}^\top \mathbf{x} + b| / \|\mathbf{w}\|$. Lặp lại thí nghiệm 20 email siêu spam, lần này bằng logistic regression của scikit-learn. Tham số $C=np.inf$ tắt khoản phạt của bài 2.2 để so cho công bằng với linear regression không phạt; mục sau sẽ nói kỹ hơn.

Python

```
from sklearn.linear_model import LogisticRegression

for name, L, yy in [("300 email gốc", L_tr, y_tr), ("thêm 20 siêu spam", L_more,
y_more)]:
    m = LogisticRegression(C=np.inf).fit(L, yy)
    t = -m.intercept_[0] / m.coef_[0, 0]
    print(f"{name}: ngưỡng = {t:.2f} link    accuracy test = {accuracy_at(t):.2f}")
```

KẾT QUẢ

```
300 email gốc      : ngưỡng = 3.01 link    accuracy test = 0.78
thêm 20 siêu spam: ngưỡng = 3.01 link    accuracy test = 0.78
```

Ngưỡng không nhúc nhích, ít nhất đến hai chữ số thập phân. Với email 24 link, mô hình đã cho p rất gần 1; đúng nhãn và đúng tự tin thì gần như không bị phạt, nên những email này không kéo ranh giới đi đâu cả. Lý do nằm ở hàm mất mát của mục tiếp theo.

HỎI NHANH

Logistic regression một feature có $w = 2$, $b = -6$. Ranh giới quyết định nằm ở đâu?

A $x = -3$

B $x = 6$

C $x = 0.5$

D $x = 3$

Hàm mất mát: cross-entropy

Mô hình gán xác suất p_i cho nhãn 1 và $1 - p_i$ cho nhãn 0. Một cách tự nhiên để chấm điểm là hỏi: mô hình gán bao nhiêu xác suất cho đúng cái nhãn đã xảy ra? Với một mẫu, con số đó là p_i nếu $y_i = 1$ và $1 - p_i$ nếu $y_i = 0$, viết gọn là $p_i^{y_i} (1 - p_i)^{1 - y_i}$. Coi các email độc lập, xác suất cho cả tập dữ liệu là tích của chúng. Chọn tham số làm tích này lớn nhất gọi là maximum likelihood.

Tích của hàng trăm số nhỏ hơn 1 rất khó làm việc, nên ta lấy log để biến tích thành tổng, đổi dấu để thành bài toán cực tiểu, rồi chia cho n . Kết quả là binary cross-entropy, còn gọi là log loss:

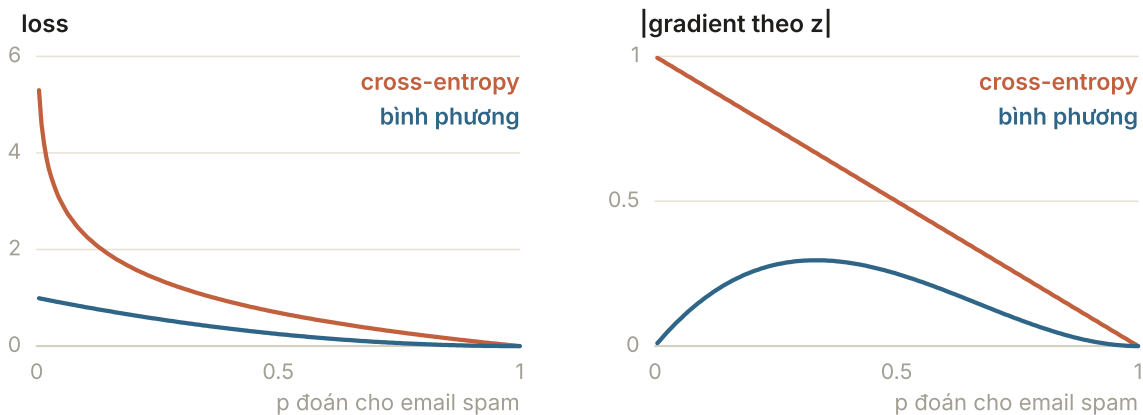
$$J(\mathbf{w}, b) = -\frac{1}{n} \sum_{i=1}^n \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right].$$

Trong mỗi số hạng chỉ một trong hai phần còn lại: với email spam là $-\log p_i$, với thư thường là $-\log(1 - p_i)$. Nhìn vào một email spam thật: đoán $p = 0.99$ mất 0.01; đoán $p = 0.5$ mất 0.69; đoán $p = 0.01$ mất 4.6. Khi $p \rightarrow 0$, loss tăng vô hạn. Cross-entropy phạt rất nặng sự tự tin sai, và gần như không phạt sự tự tin đúng. Đây chính là lý do 20 email siêu spam không kéo được ranh giới. Với thư thường mà mô hình đoán $p = 0.9$, loss là $-\log 0.1 \approx 2.30$. Nhưng mọi email vẫn góp một khoản dương, dù nhỏ, nên ranh giới phụ thuộc vào tất cả các điểm chứ không chỉ những điểm gần nó.

Tại sao không giữ hàm bình phương $(p - y)^2$ cho quen? Hãy nhìn gradient theo z . Với một email spam mà mô hình đang đoán $p = 0.01$, tức sai rất nặng:

- Bình phương: $\frac{\partial}{\partial z}(p - 1)^2 = 2(p - 1)p(1 - p) \approx -0.0196$.
- Cross-entropy: $\frac{\partial}{\partial z}(-\log p) = p - 1 = -0.99$.

Gradient của hàm bình phương nhỏ hơn 50 lần, đúng ở chỗ mô hình cần sửa nhiều nhất. Thừa số $p(1 - p)$ đến từ đạo hàm của sigmoid và gần bằng 0 ở hai đầu, nơi đường cong phẳng. Cross-entropy có log triệt tiêu đúng thừa số đó. Thêm nữa, hàm bình phương ghép với sigmoid không còn lỗi, còn cross-entropy thì lỗi: không có cực tiểu địa phương nào để mắc kẹt.



Spam bị đoán p nhỏ: cross-entropy cho gradient lớn, bình phương gần 0.

HỎI NHANH

Với $p = \sigma(z)$, nếu dùng bình phương sai số thay cho cross-entropy, thì khi mô hình rất tự tin mà sai, chuyện gì xảy ra?

- A Gradient theo z vẫn luôn bằng $p - y$
- B Bình phương sai số không tính được với nhãn 0 và 1
- C Gradient theo z có thể rất nhỏ, do thừa số $p(1 - p)$
- D Không có gradient, vì sigmoid không có đạo hàm

Gradient và tự cài bằng NumPy

Đạo hàm của sigmoid có dạng đẹp: $\sigma'(z) = \sigma(z)(1 - \sigma(z)) = p(1 - p)$. Với một mẫu, loss là $\ell = -[y \log p + (1 - y) \log(1 - p)]$. Theo quy tắc chuỗi:

$$\frac{\partial \ell}{\partial z} = \underbrace{\left(-\frac{y}{p} + \frac{1-y}{1-p}\right)}_{\partial \ell / \partial p} \cdot \underbrace{p(1-p)}_{\partial p / \partial z} = -y(1-p) + (1-y)p = p - y.$$

Mọi phân số triệt tiêu, chỉ còn lại “xác suất đoán trừ nhãn thật”. Vì $z = \mathbf{w}^\top \mathbf{x} + b$, nhân thêm $\mathbf{x}$ để có đạo hàm theo $\mathbf{w}$, rồi lấy trung bình trên cả tập:

$$\nabla_{\mathbf{w}} J = \frac{1}{n} X^\top (\mathbf{p} - \mathbf{y}), \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_i (p_i - y_i).$$

So với linear regression ở bài 2.1, công thức giống hệt, chỉ thay $\hat{\mathbf{y}}$ bằng $\mathbf{p}$ và không có hệ số 2. Sự trùng hợp này không phải tình cờ: cả hai thuộc cùng một họ mô hình (generalized linear model), và ở đó gradient luôn có dạng “dự đoán trừ thực tế, nhân với đặc trưng”.

Thử với một mẫu $x = 2$, nhãn $y = 1$, mô hình $w = 1$, $b = -1$. Khi đó $z = 1 \cdot 2 - 1 = 1$, $p = \sigma(1) = 0.731$, $\text{loss} = -\log 0.731 = 0.313$. Gradient theo w là $(p - y)x = (0.731 - 1) \cdot 2 = -0.538$. Gradient âm nghĩa là tăng w sẽ giảm loss: một bước với learning rate 0.5 cho $w = 1 + 0.5 \cdot 0.538 = 1.269$, và mô hình tự tin hơn vào nhãn đúng.

Bảng shape cho 300 email train với 2 đặc trưng:

Đại lượng	Biểu thức	Shape
Đặc trưng	$\mathbf{X}$	(300, 2)
Tham số	w, b	(2,), số
Điểm tuyến tính	$z = \mathbf{X} @ \mathbf{w} + b$	(300,)
Xác suất	$p = \text{sigmoid}(z)$	(300,)
Sai lệch	$p - y$	(300,)

Đại lượng	Biểu thức	Shape
Gradient theo w	$X.T @ (p - y) / n$	(2,)

Cài đặt theo đúng cách của bài 2.1, gộp (w, b) vào một vector theta:

Python

```
def sigmoid(z):
    # sigmoid(z) = exp(-log(1 + exp(-z))), tính ổn định
    return np.exp(-np.logaddexp(0., -z))

def loss(theta, X, y):
    logits = X @ theta[:-1] + theta[-1]
    # cross-entropy tính thẳng từ logits: không tràn số, không cần kẹp
    return np.mean(np.logaddexp(0., logits) - y * logits)

def grad(theta, X, y):
    r = sigmoid(X @ theta[:-1] + theta[-1]) - y      # p - y, shape (n,)
    return np.append(X.T @ r / len(y), r.mean())
```

Loss được tính thẳng từ logits bằng logaddexp, nên không tràn số và không cần kẹp xác suất. Nếu kẹp xác suất ngay trong loss dùng để học, gradient $p - y$ sẽ không còn khớp ở vùng bị kẹp. Trước khi huấn luyện, kiểm gradient bằng sai phân trung tâm như bài 2.1, bước rẻ nhất để bắt lỗi dấu hoặc lỗi chia n :

Python

```
def numerical_grad(f, theta, h=1e-5):
    g = np.zeros_like(theta)
    for j in range(len(theta)):
        e = np.zeros_like(theta)
        e[j] = h
        g[j] = (f(theta + e) - f(theta - e)) / (2 * h)
    return g

def rel_error(a, b):
    return np.linalg.norm(a - b) / (np.linalg.norm(a) + np.linalg.norm(b))

theta0 = np.random.default_rng(1).normal(size=3)
```

```
num = numerical_grad(lambda t: loss(t, X_tr, y_tr), theta0)
print(f"sai số tương đối: {rel_error(grad(theta0, X_tr, y_tr), num):.1e}")
```

KẾT QUẢ

sai số tương đối: 5.8e-12

Sai số cỡ 10^{-12} : gradient đúng. Giờ chạy gradient descent. Hai đặc trưng đều là số đếm nhỏ, cùng thang đo, nên ở đây không cần chuẩn hóa.

Python

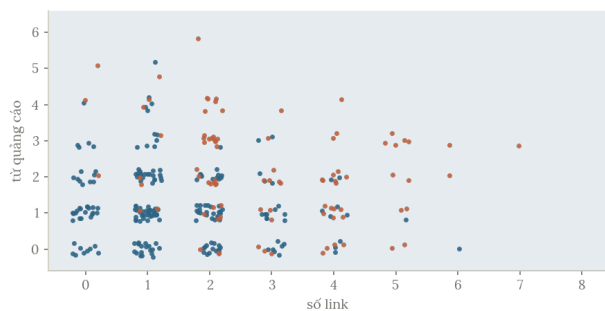
```
theta = np.zeros(3)
history = []
for epoch in range(5000):
    history.append(loss(theta, X_tr, y_tr))
    theta -= 0.5 * grad(theta, X_tr, y_tr)

print("w =", theta[:2].round(3), " b =", theta[2].round(3))
print("loss ban đầu:", round(history[0], 4), " sau 5000 vòng:", round(history[-1], 4))
```

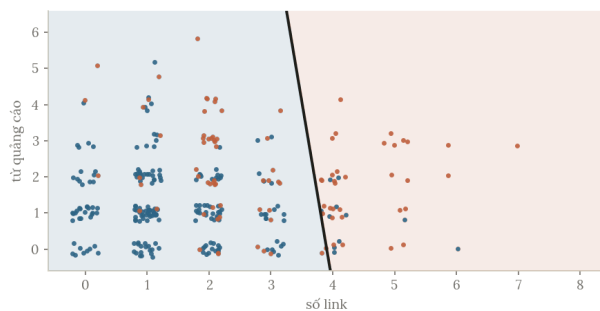
KẾT QUẢ

w = [1.124 0.947] b = -4.73
loss ban đầu: 0.6931 sau 5000 vòng: 0.3995

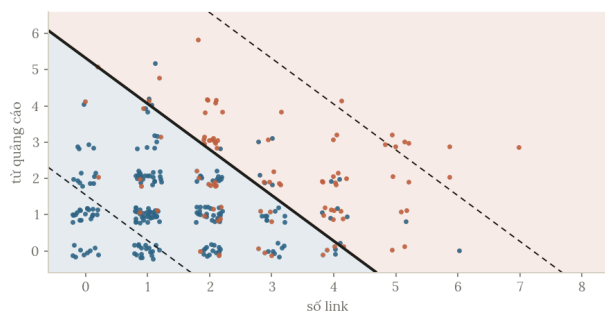
Loss ban đầu là $0.6931 = \log 2$, không phải ngẫu nhiên: với θ bằng 0, mọi email có $p = 0.5$ và mỗi email mất $-\log 0.5$. Đây là một mốc nên nhớ để kiểm code: bài hai lớp khởi tạo bằng 0 thì loss ban đầu phải là $\log 2$ (dùng class weight thì mốc này đổi).



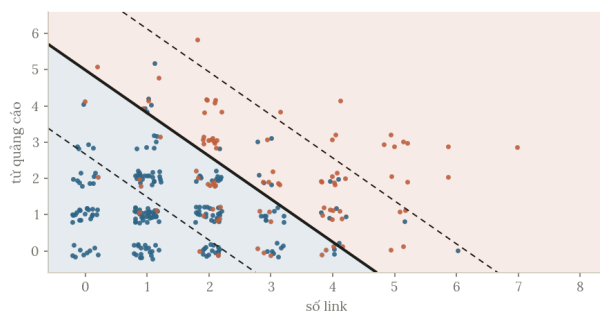
vòng 1: loss 0.6751; $w = (-0.02, -0.04)$, $b = -0.10$



vòng 10: loss 0.5903; $w = (0.17, 0.02)$, $b = -0.68$



vòng 100: loss 0.4203; $w = (0.73, 0.58)$, $b = -3.10$



vòng 5000: loss 0.3995; $w = (1.12, 0.95)$, $b = -4.73$

Gradient descent của bài trên 300 email: ranh giới $p = 0.5$ (nét liền) và hai mức 0.1, 0.9 (nét đứt) dịch dần về chỗ tách spam.

Tham số học được nằm gần quy luật thật (1.1, 0.9, -4.5); phần lệch là do chỉ có 300 email. Đối chiếu với scikit-learn:

Python

```
model = LogisticRegression(C=np.inf).fit(X_tr, y_tr)
print("scikit-learn: w =", model.coef_[0].round(3), " b =", model.intercept_.round(3))
print("trùng gradient descent:", np.allclose(theta, np.append(model.coef_[0],
model.intercept_), atol=1e-3))
```

KẾT QUẢ

```
scikit-learn: w = [1.124 0.947] b = [-4.73]
trùng gradient descent: True
```

Hai cách cho cùng một nghiệm. scikit-learn không dùng gradient descent mà dùng L-BFGS, một phương pháp dùng thêm thông tin độ cong nên hội tụ trong vài chục bước thay vì vài nghìn. Khác với linear regression, ở đây không có nghiệm đóng: đặt gradient bằng 0 cho một hệ phương trình chứa sigmoid, không giải ra được bằng biến đổi đại số.

VỀ THAM SỐ C

LogisticRegression mặc định có phạt L2 với $C=1.0$, trong đó C tỉ lệ nghịch với λ của bài 2.2. Ở trên ta dùng $C=np.inf$ để tắt phạt và so được với code tự viết. Trong thực tế nên giữ phạt: khi dữ liệu tách được hoàn toàn bằng một đường thẳng, logistic regression không phạt sẽ đẩy trọng số lên vô hạn để mọi p tiến về đúng 0 hoặc 1, và thuật toán không bao giờ dừng ở một nghiệm hữu hạn.

HỎI NHANH

Khởi tạo mọi tham số bằng 0 cho một bài phân loại hai lớp. Cross-entropy lúc đầu bằng bao nhiêu?

A 0

B 0.5

C $\log 2 \approx 0.693$

D Không xác định được

Đọc hệ số qua odds

Hệ số của logistic regression không đọc được như linear regression, kiểu “thêm một link thì xác suất tăng bao nhiêu”. Cách đọc đúng là qua odds, như mục trước đã nói.

Python

```
w = model.coef_[0]
print("thêm 1 link: odds nhân", np.exp(w[0]).round(2), " thêm 1 từ quảng cáo: odds nhân",
      np.exp(w[1]).round(2))

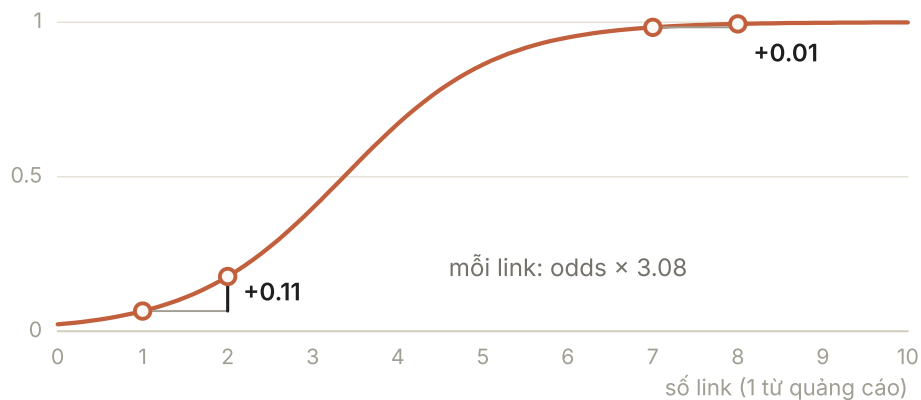
emails = np.array([[1, 1], [2, 1], [7, 1], [8, 1]]) # (số link, số từ quảng cáo)
print("P(spam):", model.predict_proba(emails)[: , 1].round(3))
```

KẾT QUẢ

thêm 1 link: odds nhân 3.08 thêm 1 từ quảng cáo: odds nhân 2.58

$P(\text{spam})$: [0.065 0.177 0.983 0.995]

Mỗi link thêm vào nhân odds spam lên khoảng 3 lần, bất kể email đang ở đâu. Nhưng xác suất thì không tăng đều. Đi từ 1 lên 2 link, xác suất tăng khoảng 0.11; đi từ 7 lên 8 link, khi email đã gần như chắc là spam, xác suất chỉ nhích thêm khoảng 0.01 vì không thể vượt quá 1. Odds nhân đều, xác suất bão hòa. Muốn so độ quan trọng giữa các đặc trưng có đơn vị khác nhau thì chuẩn hóa trước, giống bài 2.2.



Mỗi link nhân odds lên 3 lần, nhưng xác suất tăng chậm dần khi gần 1.

HỎI NHANH

Một email đang có $p = 0.5$. Hệ số của “số link” là $w = \ln 3$. Thêm một link thì p thành bao nhiêu?

A 1.5

B 0.83

C 0.6

D 0.75

Ngưỡng quyết định

CHỌN NGƯỠNG TRÊN VALIDATION

Tập gọi là test trong ví dụ nhỏ dưới đây được dùng để minh họa nhiều ngưỡng. Nếu dùng bảng đó để chọn ngưỡng, nó đã đóng vai trò validation. Trong một bài thật, chọn ngưỡng theo chi phí hoặc metric trên validation và chỉ chấm test để riêng sau khi chốt lựa chọn.

Ngưỡng 0.5 hợp lý khi hai loại sai có giá như nhau. Với bộ lọc thư rác thì không: để lọt một email spam chỉ hơi phiền, còn chặn nhầm một email thật (thư mời nhập học, kết quả thi) có thể rất đắt. Xem ngưỡng thay đổi điều gì trên 100 email test:

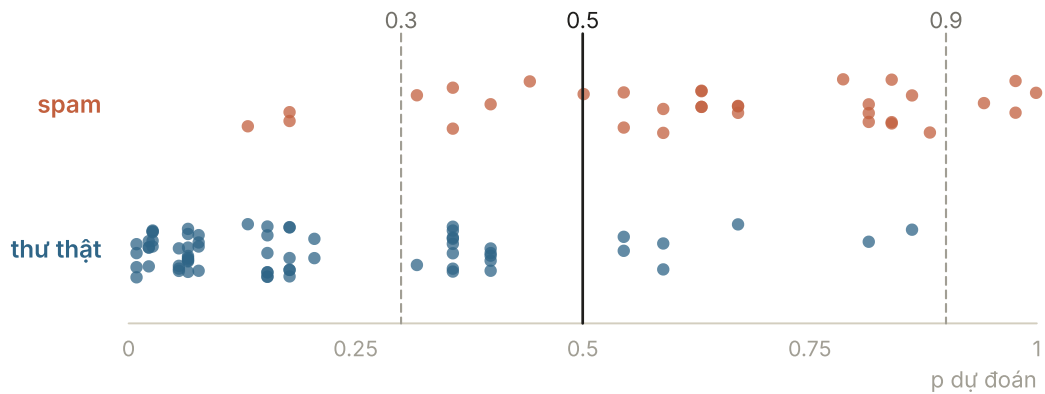
Python

```
p_te = model.predict_proba(X_te)[: , 1]
print("predict trùng p > 0.5:", np.array_equal(model.predict(X_te), (p_te >
0.5).astype(float)))
for t in [0.3, 0.5, 0.9]:
    block = p_te >= t
    print(f"ngưỡng {t}: chặn đúng {np.sum(block & (y_te == 1))>2} spam, "
          f"chặn nhầm {np.sum(block & (y_te == 0))>2} thư thật, lọt {np.sum(~block & (y_te
== 1))>2} spam")
print("accuracy:", model.score(X_te, y_te), "  luôn đoán 'không spam':", np.mean(y_te == 0))
```

KẾT QUẢ

```
predict trùng p > 0.5: True
ngưỡng 0.3: chặn đúng 30 spam, chặn nhầm 21 thư thật, lọt 3 spam
ngưỡng 0.5: chặn đúng 25 spam, chặn nhầm 7 thư thật, lọt 8 spam
ngưỡng 0.9: chặn đúng 4 spam, chặn nhầm 0 thư thật, lọt 29 spam
accuracy: 0.85  luôn đoán 'không spam': 0.67
```

Dòng đầu xác nhận predict chính là cắt $p > 0.5$ (đúng bằng 0.5 thì scikit-learn chọn lớp 0). Hạ ngưỡng thì chặn được nhiều spam hơn nhưng chặn nhầm nhiều thư thật hơn; nâng ngưỡng thì ngược lại. Không có ngưỡng nào tốt nhất cho mọi bài: ngưỡng là quyết định nghiệp vụ, chọn theo cái giá của từng loại sai. Mô hình chỉ cung cấp xác suất. Bài 1.5 có các thước đo cho đúng sự đánh đổi này (precision, recall, đường ROC).



Dời ngưỡng là đổi số thư thật bị chặn nhầm lấy số spam lọt qua.

Dòng cuối là một lời nhắc khác. Một “mô hình” luôn đoán “không spam” đã được accuracy khoảng 0.67, vì hai phần ba số email là thư thường. Accuracy của mô hình thật phải được so với mốc đó, không phải với 0. Dữ liệu càng lệch thì mốc này càng cao: nếu chỉ 1% email là spam, đoán bừa “không spam” đã có accuracy 0.99.

Khi đề bài chấm bằng log loss, cái cần nộp là xác suất. Mốc tương ứng là đoán cho mọi email cùng một xác suất bằng tỉ lệ spam của tập train:

Python

```
from sklearn.metrics import log_loss

base = np.full(len(y_te), y_tr.mean())
print("log loss mốc:", round(log_loss(y_te, base), 3), " mô hình:", round(log_loss(y_te,
p_te), 3))
```

KẾT QUẢ

log loss mốc: 0.635 mô hình: 0.371

HỎI NHANH

Với bộ lọc thư rác, chặn nhầm một thư thật tốn kém hơn nhiều so với để lọt một thư spam. Ta nên chỉnh ngưỡng thế nào?

- A Hạ ngưỡng xuống 0.3 để chặn được nhiều spam hơn
- B Nâng ngưỡng lên, và chọn mức cụ thể trên validation
- C Giữ 0.5, vì đó là ngưỡng chuẩn của mô hình
- D Huấn luyện thêm nhiều vòng cho mô hình chắc hơn

Nhiều lớp: softmax regression

Nhận dạng chữ số viết tay có 10 lớp, từ 0 đến 9. Mở rộng tự nhiên: mỗi lớp k có vector trọng số $\mathbf{w}_k$ và bias b_k riêng, cho một điểm $z_k = \mathbf{w}_k^\top \mathbf{x} + b_k$ (gọi chung là logits). Hàm softmax biến K điểm thành một phân phối xác suất:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}.$$

Mọi p_k dương và cộng lại bằng 1. Lớp có điểm cao nhất có xác suất cao nhất, nhưng các lớp khác không bị đẩy về đúng 0, nên “max” mềm hơn, đúng như tên gọi. Với $K = 2$, $p_1 = \frac{1}{1+e^{-(z_1-z_2)}} = \sigma(z_1 - z_2)$: softmax hai lớp chính là logistic regression trên hiệu hai điểm. Softmax luôn chia hết xác suất cho các lớp đã biết: đưa ảnh một chiếc ô tô vào mô hình chó, mèo, chim thì nó vẫn chọn một trong ba, có khi rất chắc.

Hàm mất mát vẫn là cross-entropy: $\ell = -\log p_y$, trừ log xác suất của lớp đúng. Gradient theo logits cũng giữ dạng quen thuộc:

$$\frac{\partial \ell}{\partial \mathbf{z}} = \mathbf{p} - \mathbf{e}_y,$$

trong đó $\mathbf{e}_y$ là vector one-hot của nhãn thật. Mô hình đẩy xác suất của lớp đúng lên và của các lớp khác xuống, đúng bằng lượng đang chênh. Mỗi thành phần của $\mathbf{p} - \mathbf{e}_y$ nằm trong $[-1, 1]$: âm ở lớp đúng, dương ở các lớp khác.

Đại lượng	Biểu thức	Shape (1300 ảnh, 64 điểm ảnh, 10 lớp)
Ảnh	X	(1300, 64)
Trọng số	W, b	(64, 10), (10,)
Logits	$Z = X @ W + b$	(1300, 10)
Xác suất	$P = \text{softmax}(Z)$	(1300, 10), mỗi hàng cộng bằng 1
Nhãn one-hot	Y	(1300, 10)
Gradient theo W	$X.T @ (P - Y) / n$	(64, 10)

Cài softmax có một cái bẫy. e^{1000} vượt quá số lớn nhất mà kiểu float64 biểu diễn được, nên thành inf, và inf / inf là nan. Cách chữa dựa trên một nhận xét: cộng cùng một hằng số c vào mọi z_k không đổi softmax, vì e^c xuất hiện ở cả tử lẫn mẫu và triệt tiêu. Chọn $c = -\max_k z_k$ thì số mũ lớn nhất là 0, không bao giờ tràn. Cùng lý do, nhân mọi logit với một số dương giữ nguyên thứ tự nên giữ nguyên lớp dự đoán, chỉ làm phân phối nhọn hơn hay bẹt hơn.

Python

```
def softmax_naive(z):
    e = np.exp(z)
    return e / e.sum()

def softmax(Z):
    Z = Z - Z.max(axis=-1, keepdims=True) # trừ max từng hàng, kết quả không đổi
    E = np.exp(Z)
    return E / E.sum(axis=-1, keepdims=True)

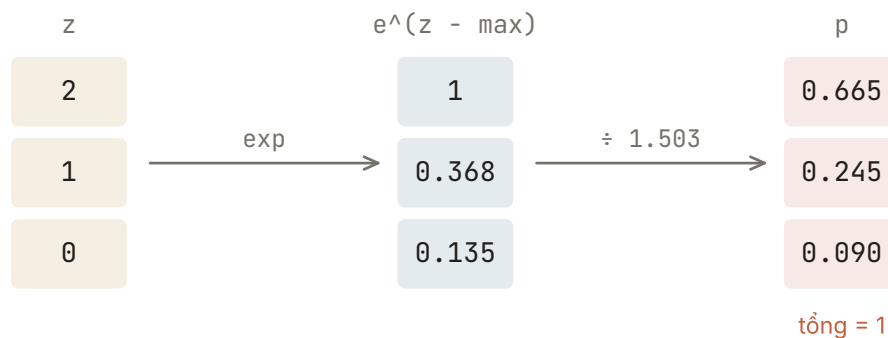
z = np.array([2.0, 1.0, 0.0])
print(softmax(z).round(3))
print(softmax_naive(z + 1000), softmax(z + 1000).round(3))
```

KẾT QUẢ

[0.665 0.245 0.09]

[nan nan nan] [0.665 0.245 0.09]

Dòng đầu tính tay được: trừ max được $(0, -1, -2)$, lấy mũ được $(1, 0.368, 0.135)$, tổng là 1.503, chia ra được $(0.665, 0.245, 0.090)$. Lớp đầu hơn lớp thứ hai 1 đơn vị logit, và xác suất gấp $e \approx 2.7$ lần. Nếu nhãn đúng là lớp thứ hai thì loss là $-\log 0.245 = 1.41$, và gradient theo logits là $\mathbf{p} - \mathbf{e}_2 = (0.665, -0.755, 0.090)$: kéo lớp đầu xuống, đẩy lớp thứ hai lên.



Softmax của $(2, 1, 0)$: trừ max, lấy mũ, chia cho tổng.

Khi cần $\log p_k$ (để tính loss) thì đừng tính softmax rồi mới lấy log, vì p_k rất nhỏ có thể làm tròn thành 0. Dùng thẳng log-sum-exp với cùng mệo trừ max:

$$\log p_k = z_k - m - \log \sum_j e^{z_j - m}, \quad m = \max_j z_j.$$

Thử trên bộ chữ số viết tay của scikit-learn: 1797 ảnh 8×8 , mỗi điểm ảnh có giá trị từ 0 đến 16.

Python

```
from sklearn.datasets import load_digits

D, t = load_digits(return_X_y=True)
D = D / 16 # điểm ảnh về [0, 1]
idx = np.random.default_rng(0).permutation(len(t))
D, t = D[idx], t[idx]
```

```
D_tr, D_te, t_tr, t_te = D[:1300], D[1300:], t[:1300], t[1300:]
print(D_tr.shape, D_te.shape)
```

KẾT QUẢ

(1300, 64) (497, 64)

Huấn luyện bằng gradient descent. Để so được với `LogisticRegression()` mặc định ($C=1$), ta thêm đúng khoản phạt L2 mà nó dùng: hàm mất mát của scikit-learn là $C \sum_i l_i + \frac{1}{2} \|W\|^2$, chia cho Cn thì thành loss trung bình cộng $\frac{1}{2Cn} \|W\|^2$.

Python

```
n, K = len(t_tr), 10
Y = np.eye(K)[t_tr] # one-hot, (1300, 10)
lam = 1 / (1.0 * n) # khớp C = 1
W, b = np.zeros((D_tr.shape[1], K)), np.zeros(K)
for epoch in range(5000):
    G = (softmax(D_tr @ W + b) - Y) / n # (1300, 10)
    W -= 2.0 * (D_tr.T @ G + lam * W)
    b -= 2.0 * G.sum(axis=0)

pred = np.argmax(D_te @ W + b, axis=1)
print("tự cài: accuracy test =", round(np.mean(pred == t_te), 3))

sk = LogisticRegression(C=1.0, max_iter=1000).fit(D_tr, t_tr)
print("scikit-learn: accuracy test =", round(sk.score(D_te, t_te), 3))
print("nhãn trùng nhau:", np.mean(sk.predict(D_te) == pred),
      " xác suất lệch lớn nhất:", np.abs(sk.predict_proba(D_te) - softmax(D_te @ W +
b)).max().round(3))
```

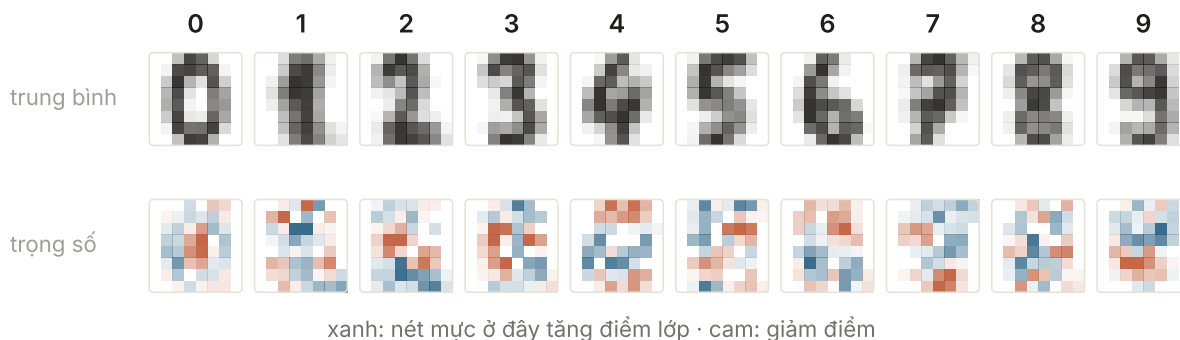
KẾT QUẢ

```
tự cài: accuracy test = 0.978
scikit-learn: accuracy test = 0.978
nhãn trùng nhau: 1.0 xác suất lệch lớn nhất: 0.032
```

Một mô hình tuyến tính đơn giản đã nhận đúng khoảng 98% chữ số. Hai cài đặt cho cùng nhãn trên mọi ảnh test; xác suất còn lệch nhau một chút vì gradient descent sau 5000 vòng vẫn chưa hội tụ hẳn, trong khi L-BFGS của scikit-learn đã tới nơi.

Mỗi cột của W có 64 số, xếp lại thành ảnh 8×8 thì đọc được mô hình đang “nhìn” gì. Điểm ảnh có trọng số dương là nơi nét mực làm tăng điểm của lớp đó, trọng số âm là nơi

nét mực làm giảm điểm. Hình dưới không giống những chữ số mẫu: mô hình không học “chữ 0 trông thế nào” mà học “chỗ nào có mực thì phân biệt được 0 với chín lớp còn lại”.



Trên: ảnh trung bình mỗi chữ số. Dưới: 64 trọng số của lớp đó.

HỎI NHANH

Cộng thêm 100 vào mọi logit $z_1, \dots, z_K$. Xác suất softmax thay đổi thế nào?

- A Không thay đổi
- B Lớp có logit lớn nhất tăng lên gần 1
- C Mọi xác suất đều tăng lên
- D Thành nan

Khi đi thi

- Để chấm log loss thì nộp `predict_proba`, kiểm đúng thứ tự cột lớp.
- Để chấm F1 hay có lớp hiếm thì chọn ngưỡng trên validation, đừng để mặc định 0.5.
- Tự cài loss thì tính từ logits bằng `logaddexp` hoặc mẹo trừ max, không lấy log của softmax.

Tóm tắt

- Linear regression trên nhãn 0/1 cho “xác suất” ngoài $[0, 1]$ và bị những điểm dễ phân loại kéo lệch ranh giới.
- Logistic regression là mô hình tuyến tính cho log-odds: $p = \sigma(\mathbf{w}^\top \mathbf{x} + b)$. Mỗi đơn vị của x_j nhân odds lên e^{w_j} . Ranh giới quyết định vẫn là đường thẳng.
- Học bằng cực tiểu cross-entropy, sinh ra từ maximum likelihood. Hàm lỗi, gradient $\frac{1}{n} \mathbf{X}^\top (\mathbf{p} - \mathbf{y})$ cùng dạng với linear regression; không có nghiệm đóng.
- Ngưỡng 0.5 chỉ là mặc định; chọn ngưỡng theo cái giá của từng loại sai. So accuracy với mốc đoán lớp đồng nhất.
- Softmax mở rộng ra K lớp với gradient $\mathbf{p} - \mathbf{e}_y$; khi cài phải trừ max và kẹp xác suất.

Kiểm tra cuối bài

Mười câu dưới đây bao quát cả bài. Làm hết rồi xem điểm; câu nào sai thì đọc lại mục tương ứng.

CÂU 1

Bài phân loại hai lớp có ba feature: tuổi từ 18 đến 80, thu nhập từ 20 đến 500 triệu, và điểm tin dụng từ 300 đến 850. Logistic regression không chuẩn hóa đạt accuracy 72%, thêm StandardScaler thì đạt 84%. Vì sao?

- A Chuẩn hóa loại bỏ các giá trị thu nhập bất thường, chống overfit nhóm giàu
- B Chuẩn hóa đưa dữ liệu về quanh 0, vào vùng gần tuyến tính của sigmoid
- C Cùng thang đo thì tối ưu hội tụ tốt, cột thu nhập không còn lấn át gradient
- D Chuẩn hóa tăng learning rate cho feature ít biến thiên nhất

ĐỀ THI NOAI 2026 Singapore · Vòng sơ loại · Câu 305

CÂU 2

Hệ số của một feature là $w_j = 0.7$. Tăng feature ấy thêm 1 đơn vị, giữ nguyên các feature khác. Điều gì thay đổi?

- A Xác suất tăng thêm đúng 0.7
- B Odds nhân thêm $e^{0.7} \approx 2$
- C Xác suất tăng gấp khoảng 2 lần
- D Odds cộng thêm đúng 0.7

CÂU 3

Một email spam thật, $y = 1$, được mô hình đoán $p = 0.1$. Cross-entropy của mẫu này bằng bao nhiêu?

- A 0.1
- B 0.105
- C 0.9
- D 2.30

CÂU 4

Mô hình lọc spam cho ra logit z , qua sigmoid thành p . Ta nên huấn luyện bằng loss nào, và vì sao?

- A Bình phương $(y - p)^2$, vì dễ hiểu, coi phân loại như hồi quy
- B Hinge $\max(0, 1 - yz)$, vì nó tối đa lề và chống được spam cố ý lách
- C Trị tuyệt đối $|y - p|$, vì ít nhạy với nhãn bị gán nhầm
- D Cross-entropy, vì gradient theo z là $p - y$, mạnh khi sai

ĐỀ THI NOAI 2026 Singapore · Vòng sơ loại · Câu 273

CÂU 5

Logistic regression hai lớp dùng trực tiếp hai feature x_1, x_2 , vector trọng số khác 0, ngưỡng 0.5. Trong mặt phẳng feature, ranh giới quyết định có dạng gì?

- A Một đường cong tùy ý, vì xác suất đi qua hàm phi tuyến
- B Đường thẳng, nhưng chỉ khi $b = 0$
- C Đường thẳng $w_1x_1 + w_2x_2 + b = 0$
- D Đường sigmoid hình chữ S trong mặt phẳng x_1, x_2

CÂU 6

Softmax hai lớp với logit (3, 1). Xác suất của lớp thứ nhất bằng bao nhiêu?

- A 0.75
- B $\sigma(2) \approx 0.88$
- C $\sigma(3) \approx 0.95$
- D 0.5

CÂU 7

`np.exp(np.array([1000., 1001.]))` cho ra `[inf inf]`, khiến softmax thành nan. Sửa thế nào cho đúng?

- A Trừ đi logit lớn nhất trước khi lấy mũ
- B Chia mọi logit cho 1000 trước khi lấy mũ
- C Dùng `float32` thay cho `float64`
- D Kẹp mọi logit vào đoạn `[-10, 10]`

CÂU 8

Validation có 1000 email, chỉ 50 email là spam. Mô hình A đoán tất cả là không spam. Mô hình B có TP = 30, FP = 40, FN = 20, TN = 910. Nếu chọn theo F1 của lớp spam, mô hình nào thắng?

- A Hòa, vì cả hai đều bỏ sót spam
- B B, với F1 bằng 0.50, dù accuracy của A cao hơn
- C A, vì accuracy là 0.95 nên F1 cũng là 0.95
- D A, vì A không có FP nên F1 cao hơn

PHÒNG THEO [USA-NA-AIO 2026 · Vòng 1 · Part 1.4](#)

CÂU 9

Bài hai lớp yêu cầu nộp xác suất của lớp spam để chấm log loss. `model.classes_` là `["spam", "valid"]`. Trong `predict_proba(X_test)`, ta lấy cột nào?

- A Cột 0
- B Cột 1
- C Lấy nhãn từ `predict`, rồi đổi spam thành 1
- D Lấy cột có xác suất trung bình cao hơn

CÂU 10

Feature nhị phân x_1 rất hiếm: trong train, email nào có $x_1 = 1$ cũng là spam. Logistic regression không có regularization. Trọng số tối ưu $\hat{w}_1$ sẽ ra sao?

- A Hữu hạn, bằng log-odds spam khi $x_1 = 1$
- B Bằng 0, vì feature quá hiếm nên không mang thông tin
- C Tăng mãi, vì gradient theo w_1 không bao giờ về 0
- D Âm, để bù cho việc feature hiếm khi bật lên

PHÒNG THEO CMU 10-601 · Exam 2 Practice · Q2.3(c)

Đọc thêm

- James, Witten, Hastie, Tibshirani — [An Introduction to Statistical Learning](#), chương 4. Logistic regression, cách đọc odds và so sánh với các bộ phân loại khác, có lab Python.
- Jurafsky, Martin — [Speech and Language Processing](#), chương Logistic Regression. Trình bày từ bài toán phân loại văn bản, suy gradient từng bước, rất hợp để đọc sau bài này.
- Ng, Ma — [CS229 Lecture Notes](#), phần Classification và Generalized Linear Models. Giải thích vì sao gradient của linear và logistic regression có cùng dạng.
- Zhang và cộng sự — [Dive into Deep Learning](#), mục Softmax Regression. Softmax nhìn từ phía mạng neural, có phần log-sum-exp; cầu nối sang chương 3.
- Murphy — [Probabilistic Machine Learning: An Introduction](#), chương 10. Logistic regression đầy đủ: maximum likelihood, phạt L2, các thuật toán tối ưu bậc hai.
- Tài liệu chính thức: [scikit-learn — Logistic regression](#), các tham số `C`, `class_weight` và lựa chọn solver.