

Learn to Build Loss Function for Linear Regression & Logistic Regression from the Ground Up

Nhóm AIO_TimeSeries

Ngày 18 tháng 11 năm 2025

Toàn bộ nội dung Logistic Regression được tổ chức thành 3 phần lớn, mỗi phần giải quyết một lớp vấn đề khác nhau - từ trực giác nền tảng, đến cách xây dựng mô hình dựa trên xác suất, và cuối cùng là phân tích toán học sâu để hiểu bản chất tại sao Logistic Regression lại hoạt động tốt như vậy.

Phần 1: Từ Linear Regression đến Logistic Regression

Giải quyết vấn đề cốt lõi: tại sao Linear Regression + MSE không phù hợp cho Classification. Trực giác về Sigmoid, Bernoulli Distribution, Likelihood và Log-Likelihood được xây dựng từ đầu. Từ đó hình thành Binary Cross-Entropy như Negative Log-Likelihood một cách tự nhiên.

Phần 2: Mở rộng - Logistic Regression khi đưa về dạng Vector-Matrix

Phát triển lại toàn bộ công thức dưới dạng vector hoá:

- chuyển dữ liệu sang ma trận X ,
- viết Loss, Gradient gọn bằng đại số tuyến tính,
- giải thích từng phép nhân ma trận từ nguyên lý cơ bản.

Phần này giúp mô hình mở rộng sang tập dữ liệu lớn, đồng thời tạo cầu nối sang phần đạo hàm bậc hai (Hessian).

Phần 3: Tại sao BCE là hàm Convex - Phân tích bằng Hessian Matrix

Dùng kiến thức Second Derivative mở rộng từ 1 biến sang nhiều biến. Giải thích toàn bộ quy trình:

- hiểu Hessian là gì từ ví dụ $f(x, y)$,
- dựng Hessian của BCE từ gradient,
- phân tích cấu trúc $H = X^T D X$,
- chứng minh Hessian luôn Positive Semi-Definite vì $D = \text{diag}(h(1 - h))$.

Từ đó, chứng minh BCE là một hàm convex trong không gian tham số - lý do căn bản khiến Logistic Regression học ổn định và không rơi vào local minima.

Phần 1: Từ Linear Regression đến Logistic Regression

Thay vì đi thẳng vào giải thích lý thuyết và công thức, **mục tiêu** của mình lần này sẽ là đi cùng bạn hiểu từng thành phần từ ground up bắt đầu từ trực giác và giả thuyết hình thành nên Linear Regression rồi phát triển lên Logistic Regression.

1.1 Vấn đề với Linear Regression

Trong bài toán dự đoán giá cổ phiếu với một biến đầu vào x và một biến đầu ra y , ý tưởng của Linear Regression là tìm một hàm dự đoán dạng đường thẳng $f(x)$ sao cho với mỗi giá trị x_i mới, ta có thể dự đoán được y_i . Đây là một giả thuyết rất đơn giản nhưng hoạt động hiệu quả nhờ vào khả năng nắm bắt xu hướng bậc nhất của dữ liệu.

Kể cả khi dữ liệu thực tế dao động mạnh, khó khớp, xu hướng chung (global trend) vẫn thường có thể được mô tả bằng một đường thẳng. Điều này phản ánh đúng bản chất toán học: đường thẳng chính là bậc đầu tiên trong khai triển Taylor, tức là mức đơn giản nhất để mô tả quan hệ giữa các biến.

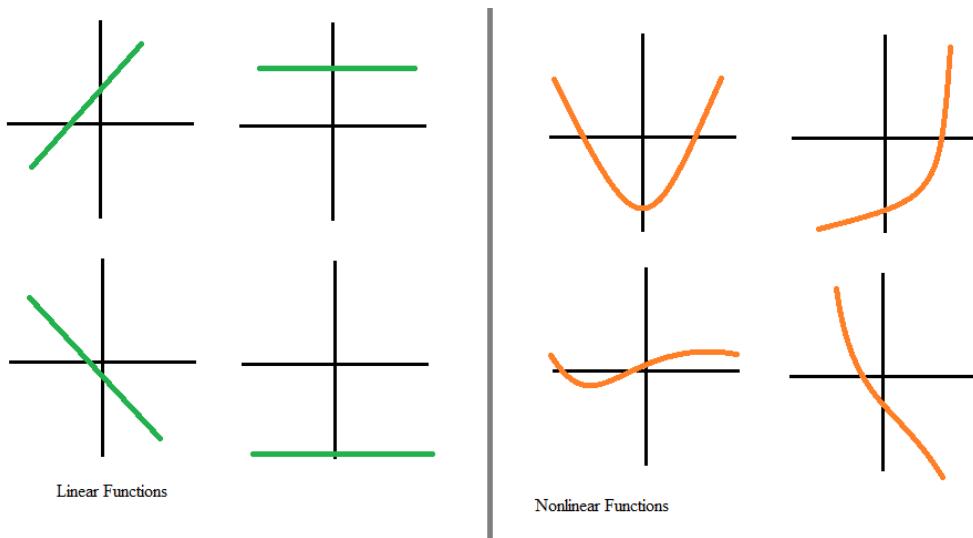


Figure 1: linear function vs nonlinear function

Linear Regression chỉ cần hai tham số để mô tả dữ liệu: intercept θ_0 điều chỉnh độ cao của đường thẳng và slope θ_1 mô tả độ dốc – tốc độ tăng giảm của dữ liệu. Sai số ϵ_i phản ánh mức lêch giữa dự đoán và giá trị thực tế; nếu toàn bộ $\epsilon_i = 0$, tất cả các điểm dữ liệu sẽ nằm trên đường thẳng. Chính vì đơn giản như vậy, Linear Regression có thể dễ dàng được tối ưu bằng đạo hàm thông qua các hàm số bậc hai như Square Loss.

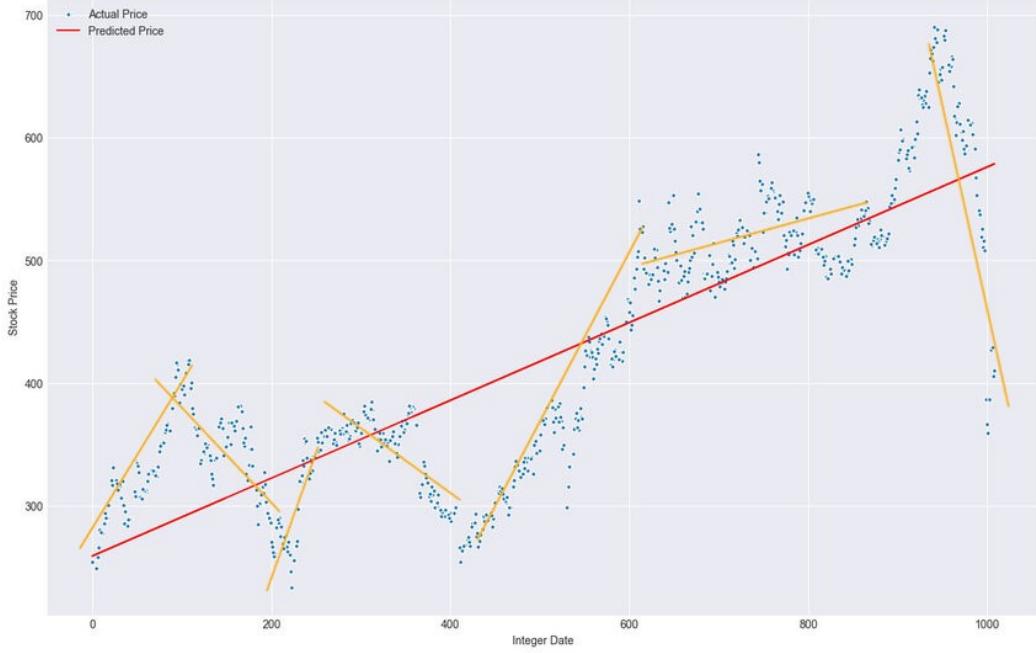


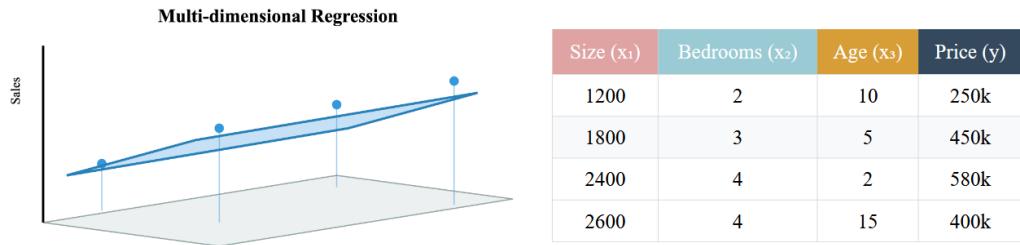
Figure 2: Linear Regression

Mặc dù mô hình tuyến tính có khả năng mô tả xu hướng mạnh nhất trong dữ liệu, bản chất của Linear Regression cũng mang theo một giả định quan trọng: đầu ra của mô hình là một giá trị thực trong khoảng $(-\infty, +\infty)$. Do đó, khi dữ liệu thực tế có bản chất không tuyến tính, có giới hạn tự nhiên (như xác suất chỉ nằm trong $[0, 1]$), hoặc khi bài toán là phân loại nhị phân 0/1, Linear Regression sẽ tạo ra các dự đoán sai lệch hoặc thậm chí vô nghĩa. Không chỉ vậy, mô hình còn ngầm giả định rằng sai số của dữ liệu tuân theo phân phối Gaussian – điều hoàn toàn không phù hợp với bài toán phân loại vốn có bản chất Bernoulli.

1.2 Cách Linear Regression chọn hàm Loss

Để hiểu rõ việc xây dựng hàm Loss cho Linear Regression, mình sử dụng ví dụ dự đoán giá nhà với ba đặc trưng x_1, x_2, x_3 và một nhãn y . Giống như cách Andrew Ng mô tả trong phần thiết kế thuật toán học, Supervised Learning luôn đi theo workflow: **Training Dataset → Learning Algorithm → Hypothesis Function**. Hypothesis $H(\theta)$ chính là hàm dự đoán mô hình học được, và nhiệm vụ của Learning Algorithm là tìm ra bộ tham số θ sao cho $H(\theta)$ dự đoán gần đúng nhất giá trị y thực tế của từng mẫu.

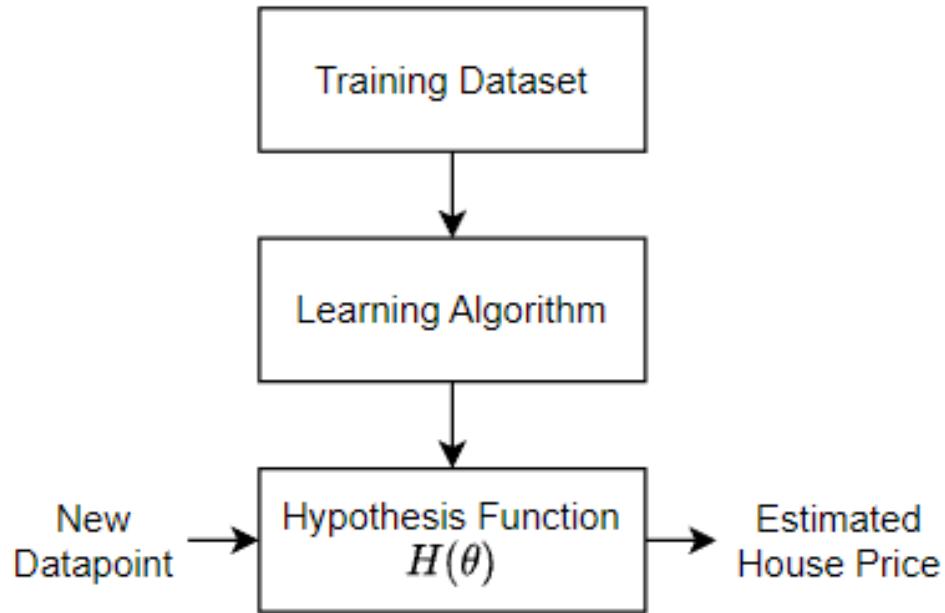
$$\text{Model: Price} = \theta_0 + \theta_1(\text{Size}) + \theta_2(\text{Bedrooms}) + \theta_3(\text{Age})$$



Để đơn giản hóa, ta gom các tham số thành vector $\theta = [\theta_0, \theta_1, \theta_2, \theta_3]^T$ và mỗi điểm dữ liệu thành

vector $X_i = [1, x_1, x_2, x_3]^T$. Khi đó hàm dự đoán của Linear Regression trở thành

$$\hat{y}_i = h_{\theta}(X_i) = \theta^T X_i.$$



Mục tiêu của học máy là điều chỉnh dần các giá trị trong θ sao cho dự đoán \hat{y} ngày càng gần giá trị thật y . Điều này dẫn chúng ta đến việc phải xây dựng một hàm đo sai số – hay hàm Loss. ách tự nhiên nhất để đo mức độ “tệ” của một dự đoán là xem mức chênh lệch $\hat{y} - y$ và bình phương nó để tránh âm dương triệt tiêu nhau. Gom toàn bộ dữ liệu lại, ta thu được hàm Loss quen thuộc:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (\theta^T X_i - y_i)^2 \text{ hay } \frac{1}{2m} \sum_{i=1}^m (\hat{y}_i - y_i)^2.$$

Hàm Loss này có dạng bậc hai theo θ , nghĩa là mặt cong của nó luôn có hình parabol và 1 điểm cực tiểu (global minimum), đây là hàm Loss Convex (Lồi). Điều này rất quan trọng vì nó đảm bảo rằng bất kỳ phương pháp tối ưu nào có tính “đi xuống dàn” cũng sẽ luôn hội tụ về đúng nghiệm tối ưu. Và để tối thiểu hóa độ tệ hay hàm Loss này, mình sẽ sử dụng Gradient Descent để tìm nghiệm tối ưu.

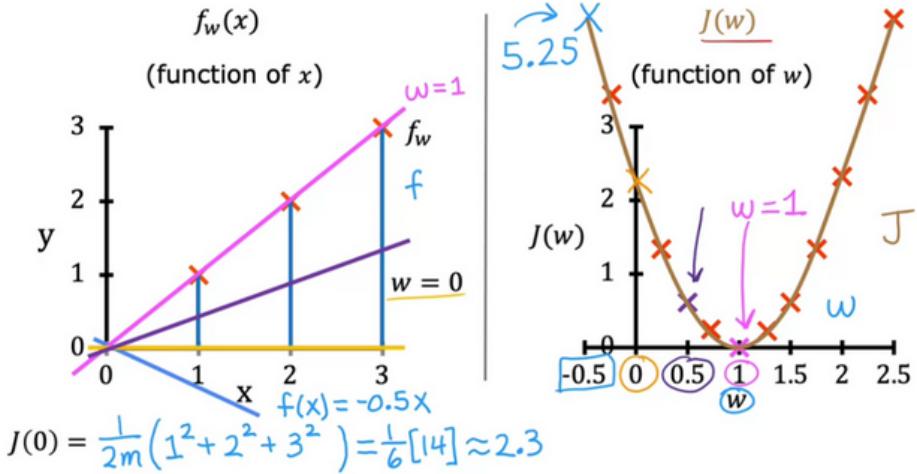


Figure 3: Minh họa hàm Loss của Linear Regression trong 2D ($w \approx \theta$): Với $f_{w,b}(x) = wx + b$ có $b = 0$, Khi mình thử mọi giá trị w từ -0.5 đến 2.5 cho $f(x)$, mình sẽ thấy hàm Loss của Linear Regression sẽ là một hình parabol.

Tuy nhiên, khi chuyển sang bài toán Phân loại (Classification) với Logistic Regression, mình không thể sử dụng hàm MSE này nữa. Lý do là khi kết hợp với hàm Sigmoid của Logistic Regression, hàm Loss MSE tổng thể sẽ có nhiều Local Minimum, đây là 1 hàm Loss Non-convex (Không Lồi).

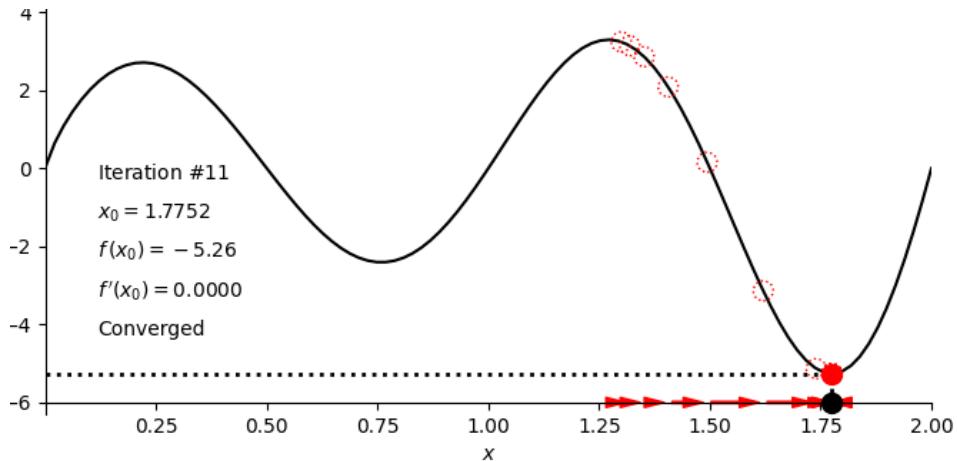


Figure 4: Minh họa hàm Loss của Linear Regression trong 2D

Từ đây, ta áp dụng Gradient Descent để tìm bộ tham số tối ưu. Nếu bạn chưa quen với tối ưu hóa hoặc quên toán, có thể hiểu đơn giản rằng Gradient Descent giống như đang đứng trên một quả đồi (hàm Loss) và luôn nhìn xem hướng đi xuống dốc nhất là hướng nào. Mỗi bước cập nhật θ chính là một bước chân nhỏ đi về phía thấp hơn, và vì hàm Loss là hình parabol, con đường đi xuống này sẽ luôn dẫn tới đáy duy nhất của nó – tức nghiệm tối ưu của mô hình.

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}, \quad \frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (\theta^T X_i - y_i) X_{ij}.$$

Viết lại toàn bộ dưới dạng vector hoá cho gọn hơn, ta có:

$$J(\theta) = \frac{1}{2m}(X\theta - Y)^T(X\theta - Y), \quad \nabla_{\theta} J = \frac{1}{m}X^T(X\theta - Y).$$

Nếu ta không muốn lặp Gradient Descent mà muốn nhảy thẳng đến nghiệm tối ưu thì Linear Regression còn có nghiệm đóng được suy ra từ **NormalEquation**:

$$X^T X \theta = X^T Y \implies \boxed{\theta = (X^T X)^{-1} X^T Y}.$$

1.3 Giới thiệu Logistic Regression

Trong phần trước, mình đã cùng xây dựng toàn bộ Linear Regression từ trực giác, đến thiết kế hàm Loss, cách tối ưu bằng Gradient Descent và nghiệm đóng dạng ma trận. Khi nhìn lại workflow đó, ta thấy Linear Regression vận hành rất tốt cho các bài toán dự đoán giá trị liên tục, nhưng lại không phù hợp khi bài toán yêu cầu dự đoán theo xác suất hoặc phân loại nhị phân (0 và 1). Đây chính là điểm bắt đầu để ta phát triển Logistic Regression.

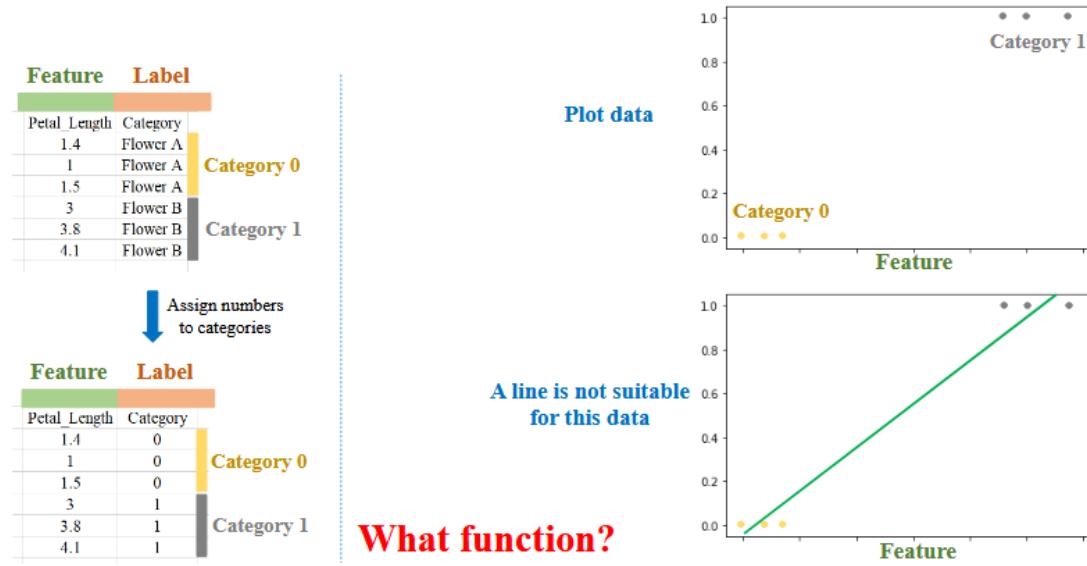


Figure 5: Nhược điểm của Linear Regression với giá trị không tuyến tính

Trước khi đi vào chi tiết, mình sẽ tham khảo cách Andrew Ng trình bày quá trình “tái thiết kế” một mô hình phân loại từ Linear Regression, trong đó ông giải thích rằng chìa khóa nằm ở việc chọn hàm ánh xạ đầu ra sao cho phù hợp bản chất xác suất ([linear regression and gradient descent](#)).

Với suy nghĩ này, ta vẫn giữ phần “lõi” của Linear Regression – tức là vẫn tính một đại lượng tuyến tính:

$$z = \theta^T x,$$

nhưng thay vì dùng trực tiếp z làm dự đoán như Linear Regression, Logistic Regression sẽ đưa z đi qua một hàm phi tuyến đặc biệt để ánh xạ giá trị thực $(-\infty, +\infty)$ về đoạn $[0, 1]$ — đó chính là hàm Sigmoid:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Điều này khiến mô hình trở thành một bộ dự đoán xác suất:

$$\hat{y} = P(y = 1 | x).$$

Trục giác của Sigmoid rất đơn giản: khi z càng lớn thì xác suất càng tiến gần 1, khi z càng âm thì xác suất tiến gần 0, còn vùng gần $z = 0$ là vùng “không chắc chắn”, nơi mô hình phản ứng nhạy nhất. Nhờ đặc tính cong hình chữ S này, Logistic Regression có thể mô hình hóa ranh giới phân loại tốt hơn Linear Regression.

Tóm lại, Logistic Regression chính là:

Linear Regression + hàm Sigmoid.

Phản tiếp theo sẽ tập trung trả lời câu hỏi: Nếu đầu ra giờ đã là xác suất, vậy ta cần xây dựng hàm Loss mới như thế nào để việc tối ưu hóa vẫn có ý nghĩa?

1.4 Xây dựng hàm Loss cho Logistic Regression từ Suy Luận

1.4.1 Hành trình tìm hàm Loss: thử nghiệm vì sao MSE không hợp cho Logistic Regressions

Trước khi lựa chọn một hàm Loss mới cho Logistic Regression, mình muốn cho bạn thấy rõ ràng hơn vì sao **Mean Squared Error (MSE)**—vốn hoạt động rất tốt trong Linear Regression—lại **không phù hợp** khi kết hợp với hàm Sigmoid. Thay vì chỉ nói rằng MSE làm Loss trở nên non-convex (nhiều điểm cực tiểu, khó hội tụ), chúng ta sẽ kiểm chứng điều này từ ba góc nhìn: hành vi của nó khi thực nghiệm, phản ứng của Loss theo từng điểm dữ liệu, và cuối cùng là đạo hàm.

(1) Quan sát từ thử nghiệm thực tế

Hãy xem thử điều gì xảy ra khi ta huấn luyện Logistic Regression nhưng vẫn dùng Loss là MSE. Ở hình dưới, mô hình đã cố gắng dự đoán phân loại 0 và 1, nhưng ta nhanh chóng thấy rằng hàm Loss gần như dừng lại rất sớm và không giảm được nữa. Accuracy đôi khi đạt mức “tạm được”, nhưng mô hình không học sâu hơn, và sai số vẫn cao dù chạy rất lâu.

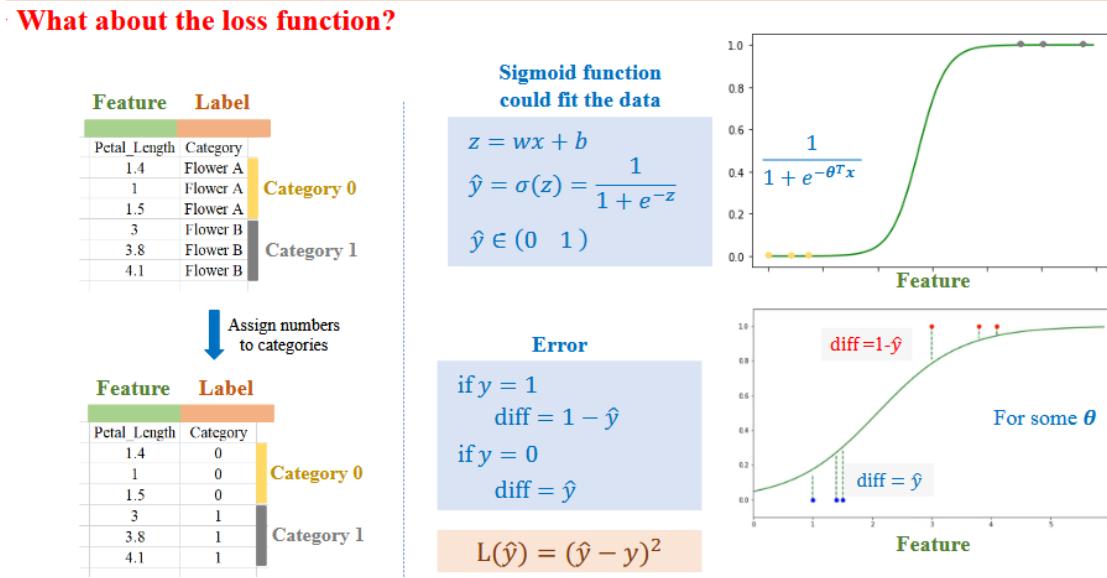


Figure 6: Huấn luyện Logistic Regression với MSE: Loss dừng quá sớm, mô hình không học tiếp

Điều rõ ràng ở đây là mô hình rơi vào vùng “mặt phẳng” của Loss—nơi gradient gần như bằng 0—nên không thể tiếp tục cập nhật tham số. Đây là hậu quả trực tiếp của việc Sigmoid bóp \hat{y} về gần

0 hoặc 1, làm cho $(\hat{y} - y)^2$ trở nên gần như phẳng khi đạo hàm nhỏ.

(2) Quan sát sai số theo từng điểm dữ liệu

Ở hình tiếp theo ta thấy cách MSE tương tác với từng điểm dự đoán. Với một θ nhất định, Sigmoid có vùng rất phẳng ở hai đầu (đoạn gần 0 và gần 1). Khi tính MSE, vùng phẳng này làm cho sai số thay đổi rất ít ngay cả khi mô hình dự đoán sai nghiêm trọng. Điều đó nghĩa là mô hình không được “phạt đủ mạnh” để học tiếp.

Logistic Regression-MSE

❖ Result

Done?

Model and Loss

$$z = \theta^T x = x^T \theta$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$L = (\hat{y} - y)^2$$

$$\frac{\partial L}{\partial \theta_i} = 2x_i(\hat{y} - y)\hat{y}(1 - \hat{y})$$

Feature	Label
Petal_Length	Category
1.4	0
1	0
1.5	0
3	1
3.8	1
4.1	1

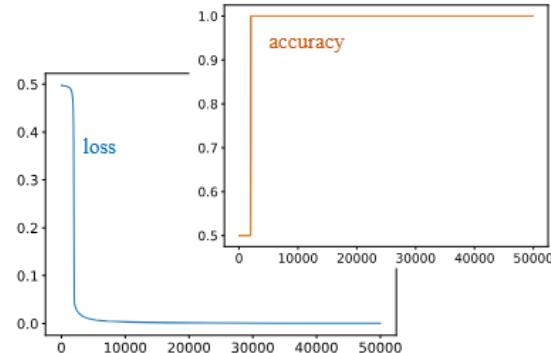


Figure 7: Sigmoid làm mất độ nhạy của MSE tại hai vùng 0 và 1, mô hình không được phạt đủ mạnh

Khi dữ liệu là phân loại, ta cần một hàm Loss phản ứng mạnh khi dự đoán sai, đặc biệt khi mô hình sai nhưng lại tự tin. Nhưng MSE hoàn toàn không làm điều này.

(3) Phân tích đạo hàm của MSE trong Logistic Regression

Đây là phần quan trọng nhất để thấy vấn đề nằm ở đâu. Với Logistic Regression, ta có:

$$z = \theta^T x, \quad \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad L = (\hat{y} - y)^2.$$

Lấy đạo hàm theo từng tham số θ_i :

$$\frac{\partial L}{\partial \theta_i} = 2(\hat{y} - y) \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta_i} = 2(\hat{y} - y) \hat{y}(1 - \hat{y}) x_i.$$

Điều quan trọng nằm ở nhân tử:

$$\hat{y}(1 - \hat{y})$$

Đây chính là đạo hàm của Sigmoid, và nó luôn rất nhỏ khi \hat{y} gần 0 hoặc gần 1. Khi nhân vào công thức đạo hàm tổng, gradient trở nên vô cùng nhỏ—gây ra hiện tượng:

$$\text{gradient} \approx 0 \Rightarrow \text{học rất chậm hoặc không học được.}$$

Hình dưới đây mô tả đầy đủ các bước đạo hàm và thành phần gradient gây ra vấn đề.

Model and Loss	Derivative
$z = \boldsymbol{\theta}^T \mathbf{x} = \mathbf{x}^T \boldsymbol{\theta}$	$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta_i}$
$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$	$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$
$L = (\hat{y} - y)^2$	$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$$\frac{\partial L}{\partial \theta_i} = 2x_i(\hat{y} - y)\hat{y}(1 - \hat{y})$$

Figure 8: Đạo hàm của MSE trong Logistic Regression: gradient bị nhân với $\hat{y}(1 - \hat{y})$ khiến cập nhật gần bằng 0

Kết luận từ ba quan sát trên:

- Khi Sigmoid “bóp đầu ra” về 0 hoặc 1, đạo hàm Sigmoid $\hat{y}(1 - \hat{y})$ gần bằng 0.
- Khi nhân với $(\hat{y} - y)$ trong công thức của MSE, gradient càng bị làm nhỏ thêm.
- Kết quả: Loss MSE trở nên phẳng, khó tối ưu, và mô hình không thể hội tụ tốt.
- Hậu quả nghiêm trọng nhất: **toàn bộ hàm Loss trở thành non-convex có nhiều cực tiểu và khó hội tụ..**

Chính vì vậy, MSE không phải là hàm Loss phù hợp cho Logistic Regression. **Vậy một hàm Loss đúng cho phân loại cần có đặc điểm gì?**. Để trả lời, ta trở lại bản chất của Classification:

- Nếu mô hình dự đoán đúng và tự tin, Loss phải rất nhỏ.
- Nếu mô hình dự đoán sai nhưng lại tự tin, Loss phải tăng thật nhanh.
- Hàm phải có dạng trơn (ie. hàm có thể đạo hàm vô tận, 1 đường cong ko bị gãy), liên tục, dễ tối ưu bằng Gradient Descent.

Với các điều kiện trên, ta thử quan sát các dạng hàm quen thuộc và xem phản ứng của chúng khi đầu vào tiến gần 0 hoặc 1.

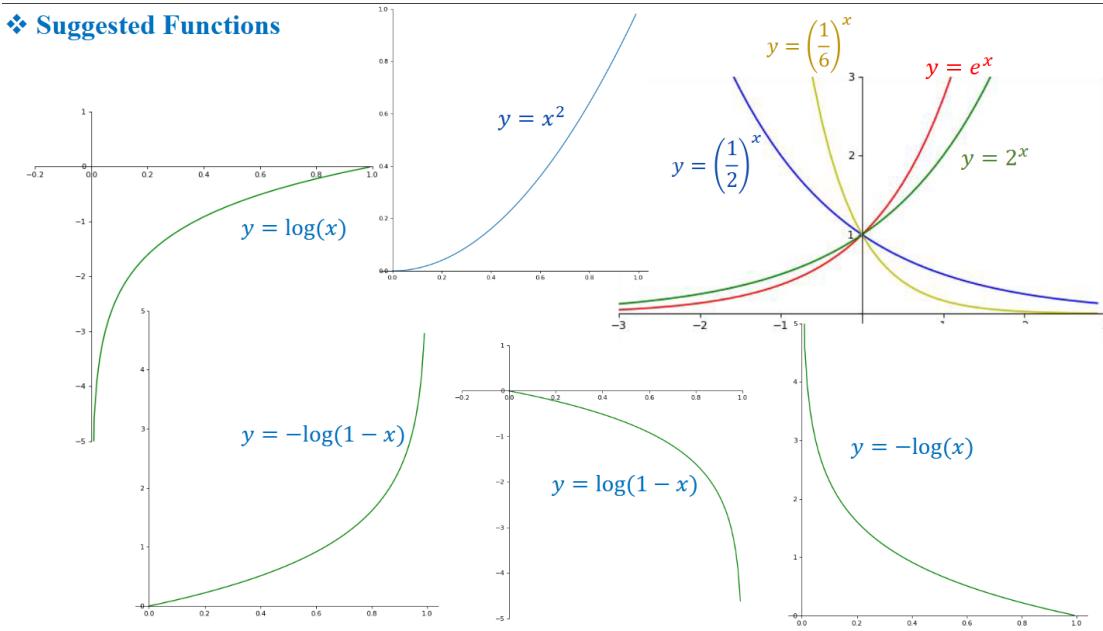


Figure 9: Quan sát các dạng hàm: chỉ các hàm logarithm tăng rất mạnh khi \hat{y} tiến về 0. Từ hình này, ta nhận thấy hai ứng viên tự nhiên cho bài toán phân loại là:

1. $-\log(\hat{y})$ khi $y = 1$
2. $-\log(1 - \hat{y})$ khi $y = 0$

Cả hai đều “bùng nổ” khi mô hình sai một cách tự tin—đúng thứ chúng ta cần, và trái ngược hoàn toàn với MSE vốn phạt rất nhẹ.

Thay vì viết theo kiểu `if/else`, ta chỉ cần kết hợp hai hàm bằng biểu thức gọn gàng:

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}).$$

Dây chính là **Binary Cross-Entropy (BCE)**—hàm Loss chuẩn cho Logistic Regression. Không chỉ phù hợp trực giác, BCE còn khớp hoàn hảo với bản chất xác suất khi mô hình hoá nhãn y như một biến Bernoulli, và công thức trên chính là negative log-likelihood.

Khi kết hợp với Logistic Regression:

$$z = \theta^T x, \quad \hat{y} = \sigma(z),$$

gradient trở nên cực kỳ gọn:

$$\frac{\partial L}{\partial \theta_i} = x_i(\hat{y} - y),$$

tương tự Linear Regression nhưng mang đầy đủ ý nghĩa xác suất.

Construct loss

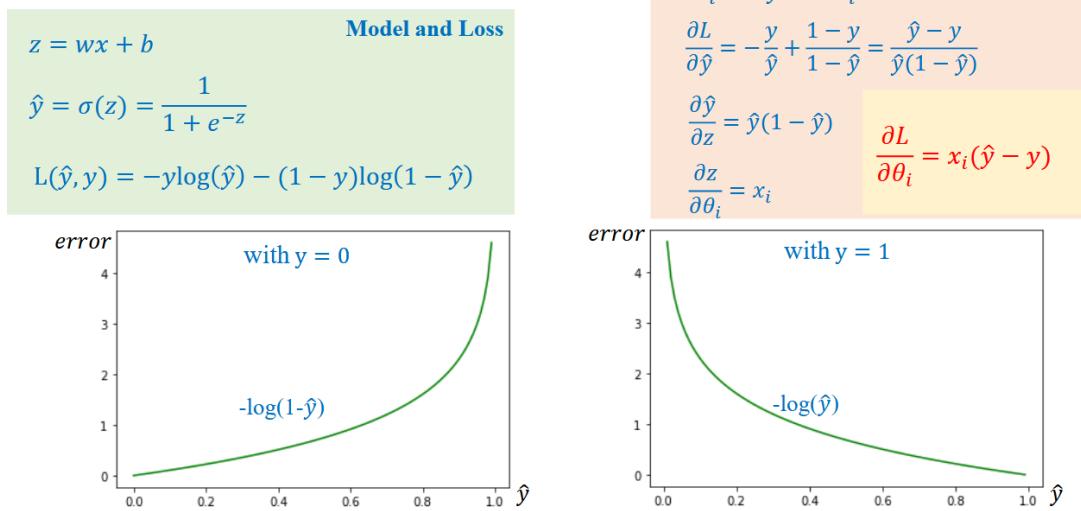


Figure 10: Pipeline Logistic Regression + BCE: đơn giản và chặt chẽ

Từ trực giác đến công thức, BCE đáp ứng tất cả nhu cầu của bài toán phân loại: phạt mạnh khi sai, ổn định khi đúng, đạo hàm đơn giản và học rất nhanh. Điều quan trọng nhất là khi quan sát mặt Loss của BCE, ta thấy nó có dạng “bát úp” ở không gian tham số—một dấu hiệu mạnh mẽ của **Convexity**.

Convexity chính là chìa khóa giúp Logistic Regression hội tụ ổn định và tránh bẫy local minima. Vì vậy, trước khi chứng minh BCE convex và so sánh BCE với MSE, ta cần xây dựng trực giác về “độ cong” của một hàm thông qua đạo hàm bậc hai — đây chính là chủ đề của phần tiếp theo.

1.4.2 Convexity là gì? Trực giác của đạo hàm bậc hai

Trước khi chứng minh hàm Loss của Logistic Regression là convex, mình muốn cùng bạn quay lại trực giác toán học nền tảng: **đạo hàm bậc hai cho ta biết Rate of change of "the rate of change"** – hay nói cách khác, nó cho ta biết độ cong (curvature) của một hàm. Cách hình tượng và dễ hiểu nhất để nắm được điều này là nhìn vào chuyển động của một chiếc xe.

Khi ta mô tả chuyển động của xe theo thời gian t , ta có ba đại lượng quen thuộc: quãng đường $x(t)$, vận tốc $v(t)$ và gia tốc $a(t)$. Đây chính là ví dụ trực tiếp mà bài giảng của DeepLearning.AI đã dùng để giải thích đạo hàm bậc một và bậc hai. Vận tốc là đạo hàm bậc một của quãng đường, $v = \frac{dx}{dt}$, còn gia tốc là đạo hàm bậc hai (ie. tốc độ thay đổi của vận tốc, nghe rất hợp lý đúng không :)), $a = \frac{dv}{dt} = \frac{d^2x}{dt^2}$. Nếu vận tốc đang tăng thì gia tốc dương, nếu vận tốc giảm thì gia tốc âm, còn nếu vận tốc không đổi thì gia tốc bằng 0. Điều này giúp ta có trực giác: đạo hàm bậc hai cho ta biết “hàm đang cong lên hay cong xuống”.

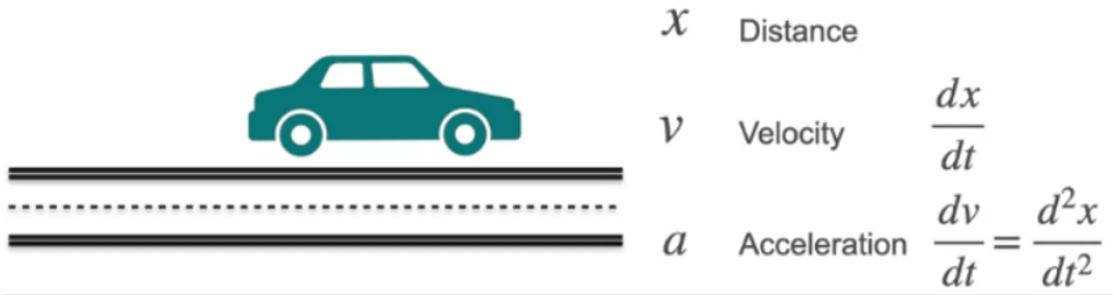


Figure 11: x , v , a : Khoảng cách – Vận tốc – Gia tốc

Hình dưới đây mô tả rõ hơn: đồ thị phía trên là quãng đường theo thời gian, mỗi điểm được mô tả bằng 1 đường thẳng, đồ thị dưới là vận tốc (đạo hàm bậc một). Ở giai đoạn đầu, vận tốc liên tục tăng nên gia tốc dương; ở đoạn giữa, xe chạy đều nên gia tốc bằng 0; ở đoạn sau, xe giảm tốc nên gia tốc âm.

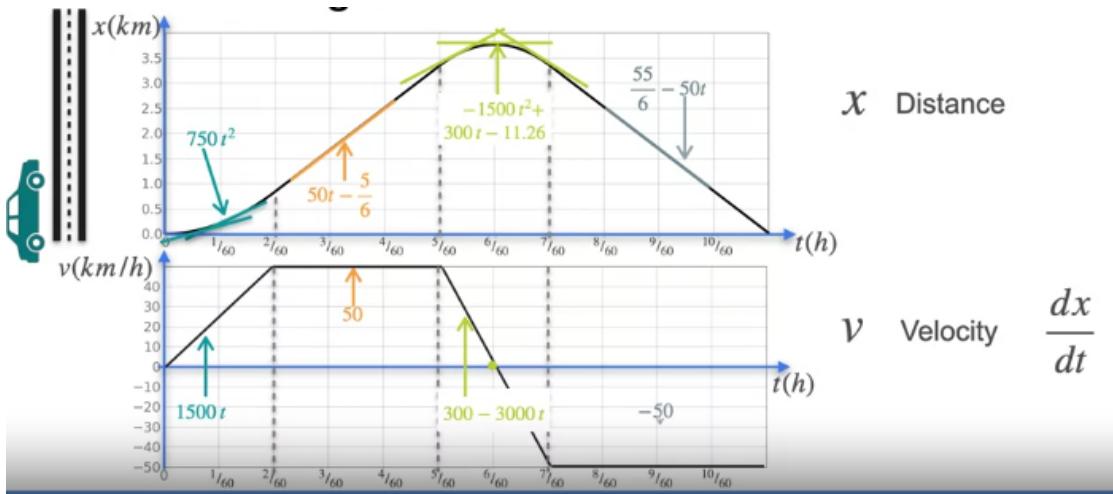


Figure 12: Đạo hàm bậc một: Vận tốc thay đổi theo thời gian

Understanding Second Derivative

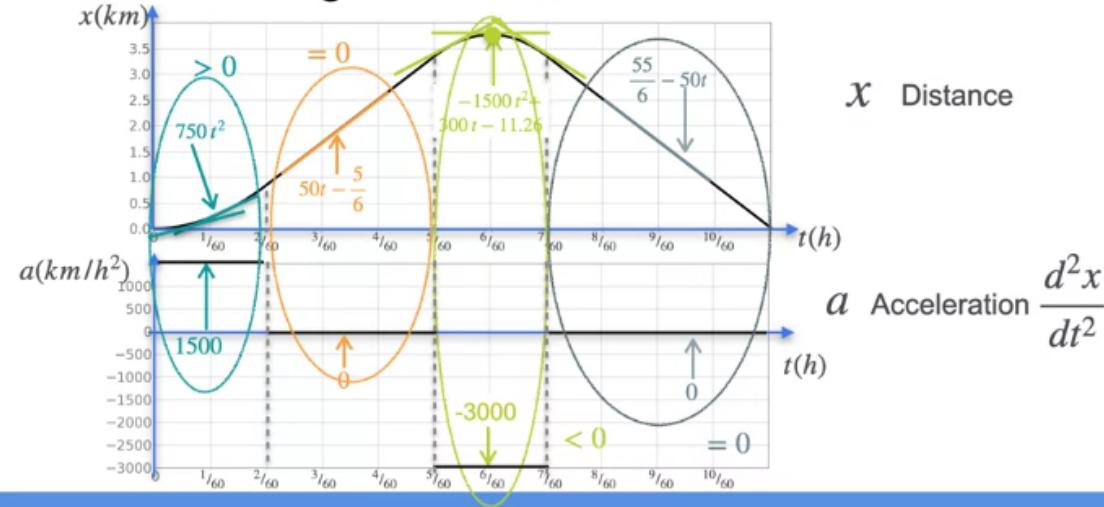


Figure 13: Hình dưới là Đạo hàm bậc hai: Gia tốc – tốc độ thay đổi của tốc độ thay đổi (tốc độ thay đổi của vận tốc)

Khi nhìn vào đồ thị quang đường, ta có thể phân tích độ cong dựa vào dấu của đạo hàm bậc hai: nếu $\frac{d^2x}{dt^2} > 0$, đồ thị cong lên (concave up / convex); nếu $\frac{d^2x}{dt^2} < 0$, đồ thị cong xuống (concave down); còn nếu bằng 0 thì ta cần thêm thông tin vì đồ thị có thể thẳng hoặc chỉ đang đổi chế độ cong.

Understanding Second Derivative

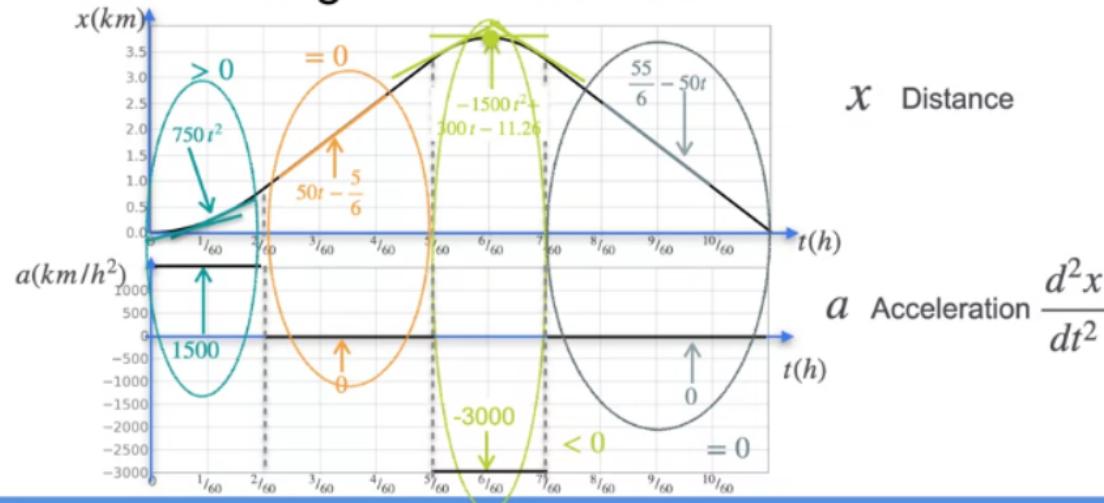


Figure 14: Ý nghĩa dấu của đạo hàm bậc hai và độ cong của hàm

Điều này liên kết trực tiếp với tối ưu hóa: tại các điểm mà đạo hàm bậc một bằng 0, **đạo hàm bậc hai cho ta biết điểm đó là cực tiểu hay cực đại**. Nếu đạo hàm bậc hai dương, đó là cực tiểu (đồ thị cong lên). Nếu âm, đó là cực đại (đồ thị cong xuống). Nếu bằng 0, ta không kết luận được. Đây chính là lý do đạo hàm bậc hai quan trọng trong phân tích convexity.

Từ góc nhìn trực giác: hàm convex là hàm luôn “cong lên”, nghĩa là đạo hàm bậc hai của nó luôn không âm. Bạn có thể hình dung rằng nếu đồ thị luôn cong lên giống hình một cái bát úp ngược, thì

dù bạn đứng ở bất kỳ điểm nào, hướng đi “xuống dốc nhất” cũng luôn dẫn bạn về cùng một đáy duy nhất. Đây là nền tảng của tối ưu hóa: hàm convex đảm bảo không có nhiều cực trị cục bộ; Gradient Descent sẽ không bao giờ bị mắc kẹt.

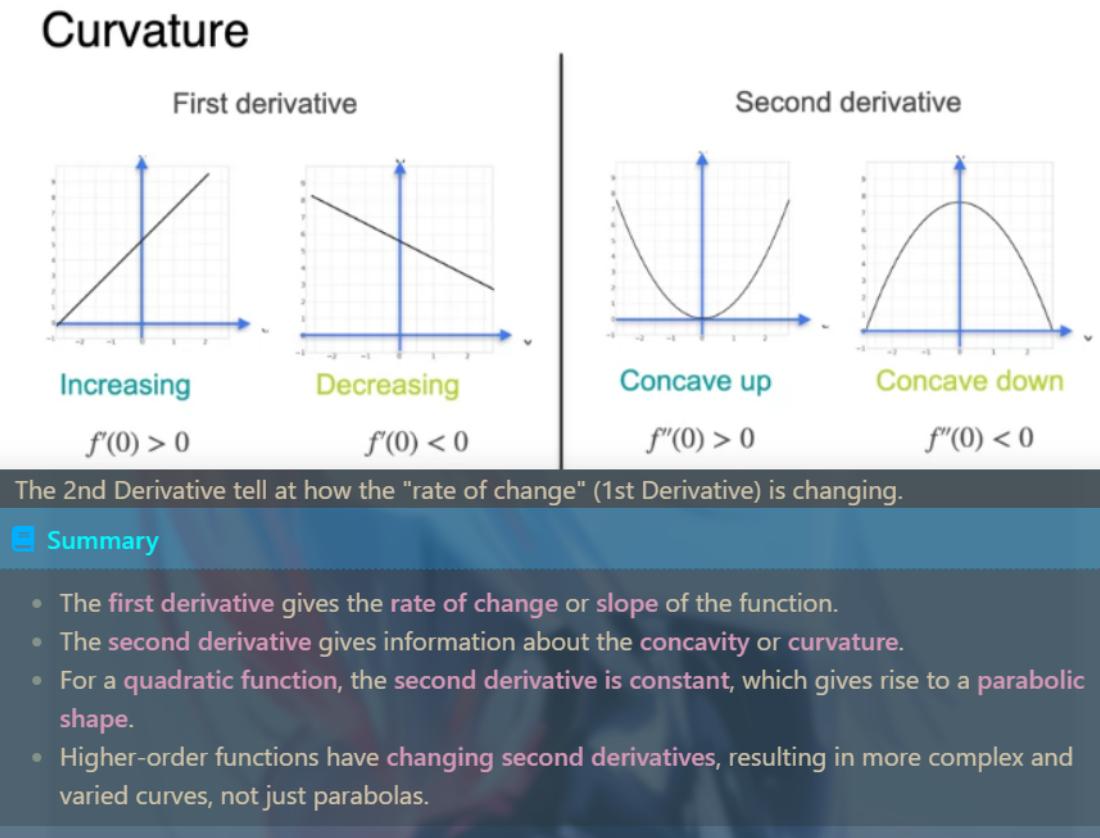


Figure 15: Tóm tắt trực giác đạo hàm bậc một và bậc hai

Khi chuyển từ ví dụ chiếc xe sang bài toán học máy, ta có thể nhìn convexity như một sự mở rộng của những gì mình vừa thấy: nếu đạo hàm bậc hai của một hàm số luôn không âm trên toàn miền, thì hàm đó là convex. Đối với hàm nhiều biến, khái niệm này được tổng quát bằng **ma trận Hessian**. Một hàm nhiều biến được gọi là convex nếu Hessian luôn là ma trận positive semi-definite. Điều này đồng nghĩa mọi eigenvalue của Hessian đều không âm, chính xác như logic của đạo hàm bậc hai trong trường hợp một biến.

Vì vậy, khi ta nói rằng một hàm Loss là convex, điều đó có nghĩa là gradient luôn dẫn mô hình về nghiệm tối ưu duy nhất. Đây sẽ là chìa khóa giải thích vì sao Binary Cross-Entropy của Logistic Regression là hàm Loss học ổn định và không gặp vấn đề local minima. Phần tiếp theo sẽ chứng minh trực tiếp điều này bằng cách phân tích đạo hàm và Hessian của BCE.

1.4.3 Chứng Minh Convex (độ lồi) của MSE và BCE

Ở phần trước, ta thấy rằng Logistic Regression dùng MSE không những học chậm mà còn dễ rơi vào các vùng gradient bằng 0. Trực giác ban đầu chỉ cho thấy “MSE không phù hợp”, nhưng để kết luận một cách chắc chắn, ta cần nhìn vào hình dạng của mặt Loss. Đây là lúc khái niệm **Convexity** trở nên quan trọng: nếu Loss là convex (độ lồi), Gradient Descent sẽ luôn hội tụ về nghiệm tối ưu duy nhất; nhưng nếu Loss non-convex, mô hình có thể mắc kẹt ở các local minima.

NON-CONVEXITY ISSUES

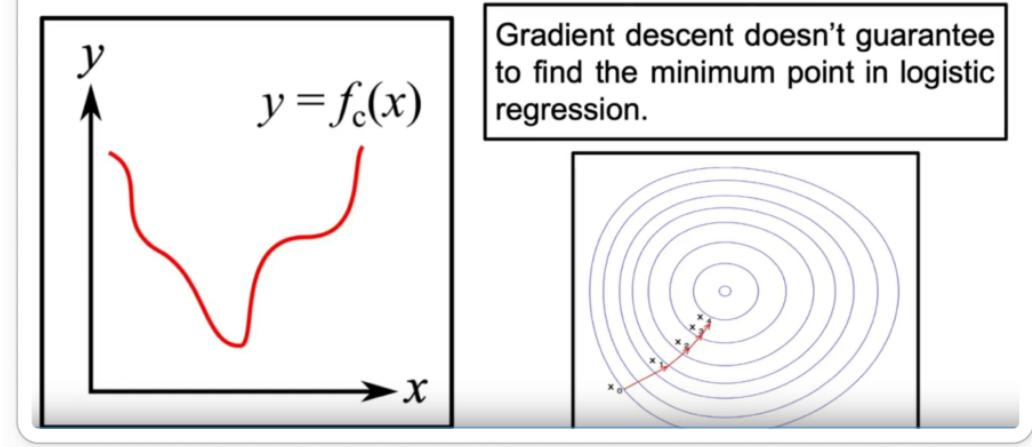


Figure 16: Đạo hàm bậc hai của MSE đổi dấu → Non-Convex

Phần này tập trung vào ba điểm chính:

- (1) **MSE khi kết hợp với Sigmoid trở thành non-convex**

Hàm Loss bị biến dạng thành dạng “gợn sóng”, xuất hiện nhiều vùng phẳng và đáy cục bộ, khiến Gradient Descent dễ mắc kẹt.

- (2) **BCE tạo ra một hàm Loss convex**

Độ cong của BCE khi kết hợp với Sigmoid luôn không âm, bề mặt Loss cong mượt và có duy nhất một cực tiểu toàn cục.

- (3) **Do đó BCE phù hợp hơn MSE cho Logistic Regression**

BCE tạo không gian tối ưu hóa ổn định, có gradient rõ ràng, không bị bão hòa như MSE.

Chứng Minh MSE là Non-Convex trong Logistic Regression

Với Logistic Regression ta có:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad L = (\hat{y} - y)^2.$$

Lấy đạo hàm bậc hai của Loss theo từng tham số θ_i :

$$\frac{\partial^2 L}{\partial \theta_i^2} = 2x_i^2 \hat{y}(1 - \hat{y}) [-3\hat{y}^2 + 2\hat{y} - y + 2y\hat{y}] .$$

Điểm quan trọng nằm ở biểu thức ngoặc vuông. Khi \hat{y} chạy trong khoảng $(0, 1)$, giá trị trong ngoặc có thể dương hoặc âm tùy vùng. Do đó đạo hàm bậc hai có thể âm → **Loss có vùng cong lên và cong xuống xen kẽ → non-convex**

Mean Squared Error

Model and Loss $z = \theta^T x = x^T \theta$ $\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$ $L = (\hat{y} - y)^2$	Derivative $\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta_i} \quad \frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$ $\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y) \quad \frac{\partial z}{\partial \theta_i} = x_i$ $\frac{\partial L}{\partial \theta_i} = 2x_i(\hat{y} - y)\hat{y}(1 - \hat{y})$
---	--

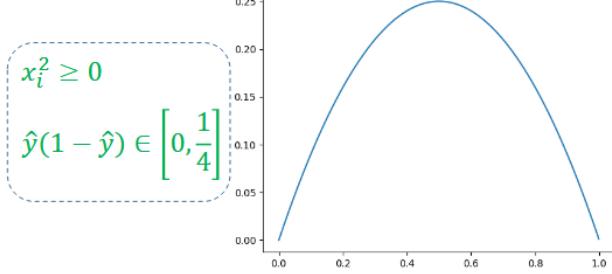
$$\frac{\partial L}{\partial \theta_i} = 2x_i(\hat{y} - y)\hat{y}(1 - \hat{y}) = 2x_i(-\hat{y}^3 + \hat{y}^2 - y\hat{y} + y\hat{y}^2)$$

$$\begin{aligned} \frac{\partial^2 L}{\partial \theta_i^2} &= \frac{\partial}{\partial \theta_i} [2x_i(-\hat{y}^3 + \hat{y}^2 - y\hat{y} + y\hat{y}^2)] \\ &= 2x_i[-3\hat{y}^2x_i\hat{y}(1 - \hat{y}) + 2x_i\hat{y}\hat{y}(1 - \hat{y}) - yx_i\hat{y}(1 - \hat{y}) + 2x_iy\hat{y}\hat{y}(1 - \hat{y})] \\ &= 2x_i^2\hat{y}(1 - \hat{y})[-3\hat{y}^2 + 2\hat{y} - y + 2y\hat{y}] \end{aligned}$$

Figure 17: Đạo hàm bậc hai của MSE đổi dấu \rightarrow Non-Convex.

Mean Squared Error

$$\frac{\partial^2 L}{\partial \theta_i^2} = 2x_i^2\hat{y}(1 - \hat{y})[-3\hat{y}^2 + 2\hat{y} - y + 2y\hat{y}]$$



$$y = 0 \quad f(\hat{y}) = -3\hat{y}^2 + 2\hat{y}$$

$$y = 1 \quad f(\hat{y}) = -3\hat{y}^2 + 4\hat{y} - 1$$

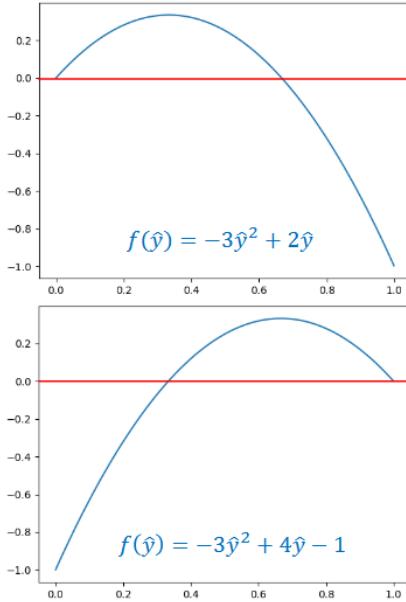


Figure 18: Một số vùng $\frac{\partial^2 L}{\partial \theta_i^2} < 0$ chứng minh Loss không lồi. Vậy Logistic Regression + MSE tạo ra mặt Loss Non-Convex

Đây chính là lý do thực nghiệm MSE học rất chậm: khi Loss có nhiều vùng cong xuống, Gradient Descent thường bị kẹt hoặc cập nhật rất nhỏ.

Ngược lại, với **Binary Cross-Entropy**:

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}), \quad \hat{y} = \sigma(z).$$

Đạo hàm bậc nhất rất đơn giản:

$$\frac{\partial L}{\partial \theta_i} = x_i(\hat{y} - y).$$

Lấy đạo hàm bậc hai:

$$\frac{\partial^2 L}{\partial \theta_i^2} = x_i^2 \hat{y}(1 - \hat{y}).$$

Vì:

$$x_i^2 \geq 0, \quad \hat{y}(1 - \hat{y}) \in [0, \frac{1}{4}],$$

nên đạo hàm bậc hai luôn không âm → **hàm Loss BCE luôn convex**

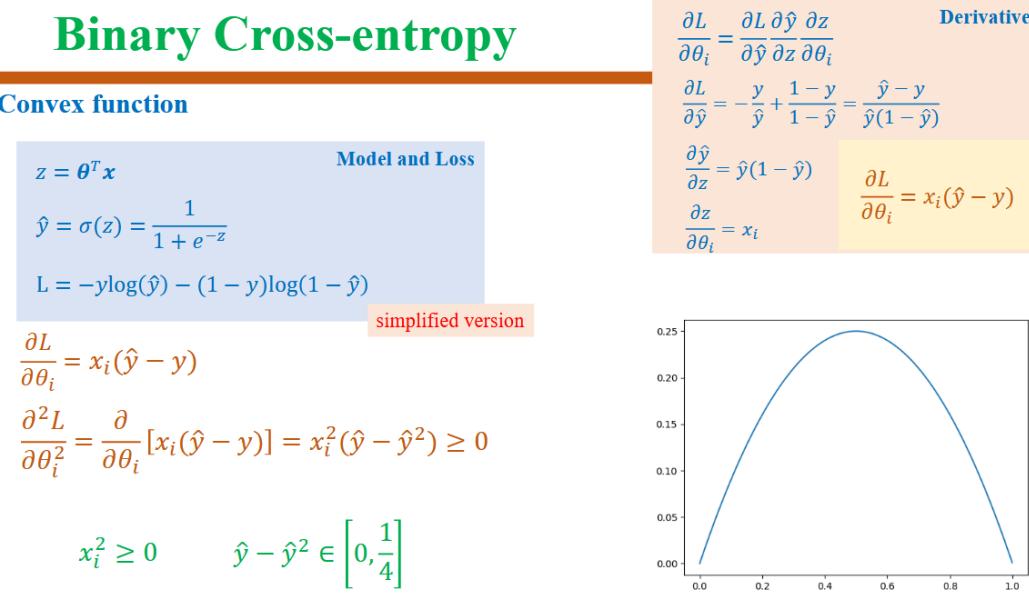


Figure 19: Đạo hàm bậc hai BCE luôn dương → mặt Loss convex

Điều này khẳng định BCE luôn có **một nghiệm tối ưu duy nhất** và Gradient Descent sẽ hội tụ ổn định.

1.4.4 So sánh MSE và BCE

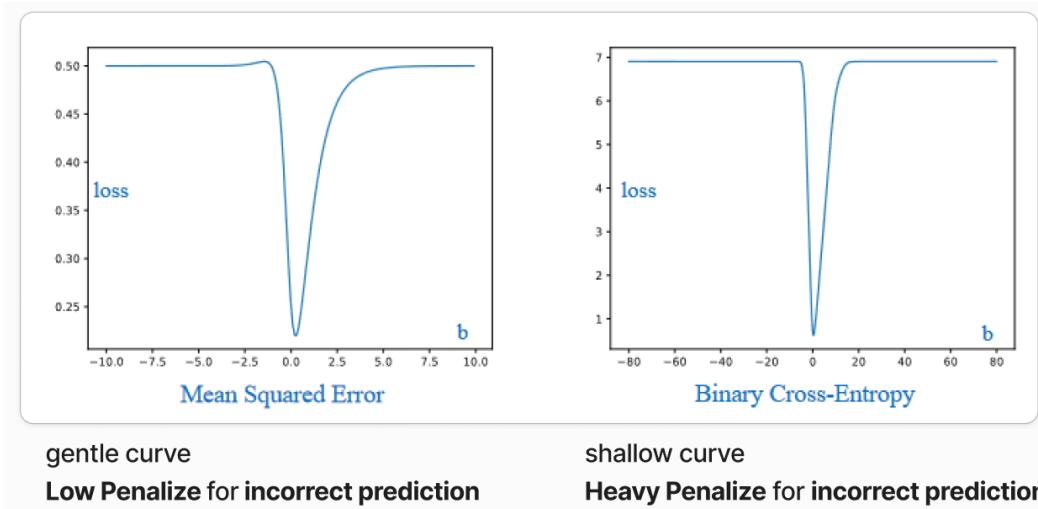


Figure 20: So sánh hàm Loss MSE và BCE

MSE: đường cong nhẹ, dải giá trị bị chặn

Loss chỉ dao động trong khoảng hẹp ($\approx 0.25 \rightarrow 0.5$), độ cong thấp và nhiều vùng bằng phẳng khi \hat{y} bao hòa. Điều này khiến gradient nhỏ, mô hình dễ đứng yên và khó điều chỉnh tham số.

BCE: đường cong sâu, dải giá trị mở rộng

Loss tăng rất mạnh khi dự đoán sai tự tin (ví dụ $\hat{y} \approx 0$ nhưng $y = 1$), tạo thành “đáy rộng” và độ cong lớn. Nhờ đó gradient rõ ràng, mô hình học ổn định và hội tụ đúng.

Xây dựng Hàm Loss cho Logistic Regression từ Xác Suất Thống Kê

2.1 Bắt đầu từ vấn đề: Vì sao Logistic Regression không thể dựa vào MSE?

Trong bài toán phân loại nhị phân, đầu ra chỉ có hai giá trị:

$$y \in \{0, 1\}.$$

Nếu dùng MSE như Linear Regression, mô hình có thể dự đoán giá trị lớn hơn 1 hoặc nhỏ hơn 0, điều này không phù hợp với ý nghĩa “xác suất”. Ngoài ra, MSE kết hợp với Sigmoid tạo ra bề mặt Loss non-convex, khiến việc tối ưu hóa bằng Gradient Descent trở nên khó khăn.

Vì thế, ta cần một cách tiếp cận đúng bản chất hơn: *xây dựng hàm Loss dựa trên xác suất thống kê*. Để làm được điều đó, ta sẽ bắt đầu từ những khái niệm nền tảng nhất.

2.2 Ôn lại nền tảng: Logarithm và Natural Logarithm

Trước khi đi vào xác suất, ta cần hiểu lý do vì sao Logarithm (log) và Natural Logarithm (ln) đóng vai trò quan trọng trong Logistic Regression.

(a) Logarithm là gì?

Logarithm trả lời câu hỏi:

$$\log_b(a) = c \quad \text{khi} \quad b^c = a.$$

Nói cách khác, log tìm ra **phần số mũ** cần có để tạo ra một số. Đây là lý do log đặc biệt hữu ích khi ta làm việc với dữ liệu có dạng tăng giảm theo bội số, hoặc khi các xác suất nhỏ được nhân liên tiếp với nhau.

(b) Ứng dụng: Log chuẩn hóa tỉ lệ để đưa về cùng một thang đo

Xét ví dụ trực quan trong hình 21. Hai giá trị:

$$8 = 2^3, \quad \frac{1}{8} = 2^{-3}$$

cách nhau 8 lần so với 1, nhưng trên trục số thông thường, khoảng cách $\frac{1}{8} \leftrightarrow 1$ và $1 \leftrightarrow 8$ không đối xứng.

Tuy nhiên, nếu dùng log cơ số 2:

$$\log_2(8) = 3, \quad \log_2(1/8) = -3,$$

thì hai giá trị này trở nên đối xứng. Điều này cho thấy log biến tỉ lệ (ratio) thành khoảng cách tuyến tính. Việc chuẩn hóa này cực kỳ quan trọng khi các trị số quá nhỏ hoặc quá lớn khiến việc tối ưu với gradient khó khăn.

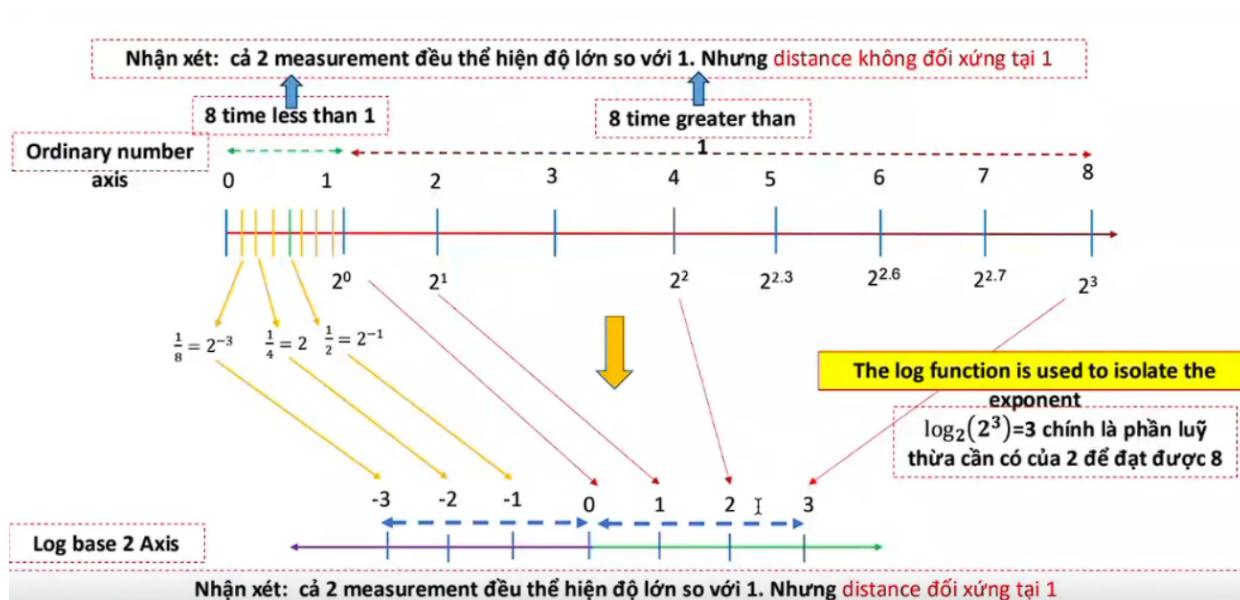


Figure 21: Log biến tỉ lệ lũy thừa thành khoảng cách tuyến tính đối xứng: một công cụ cực kỳ quan trọng khi xử lý xác suất nhỏ và dữ liệu trải dài nhiều bậc độ lớn.

(c) Vì sao dùng $\ln(x)$ thay vì $\log_{10}(x)$?

Trong học máy, ta dùng $\ln(x)$ vì:

- \ln liên quan trực tiếp đến hàm mũ e^x , vốn xuất hiện liên tục trong xác suất và tối ưu hóa.
- Đạo hàm của \ln rất đơn giản:

$$\frac{d}{dx} \ln(x) = \frac{1}{x}.$$

- Khi tối ưu hóa Likelihood, toàn bộ biểu thức sẽ trở nên gọn, tránh phải mang thêm hệ số $\ln(10)$.

Tóm lại: ln giúp công thức đẹp, dễ đạo hàm, và phù hợp với mọi phân phối dùng trong thống kê hiện đại.

2.3 Probability và Likelihood (giải thích liên mạch bằng ví dụ đồng xu)

Để hiểu Logistic Regression dựa vào xác suất như thế nào, điều quan trọng nhất là phân biệt **Probability** và **Likelihood**.

Probability là xác suất ta *giả định trước khi quan sát dữ liệu*. Hãy tưởng tượng bạn có một đồng xu công bằng. Trước khi tung, bạn cho rằng xác suất ra mặt ngửa là $P(\text{Head}) = 0.5$ và xác suất ra mặt sấp cũng là 0.5. Khi tính Probability, ta đang hỏi: “Nếu mô hình đúng như ta tin (đồng xu công bằng), thì khả năng xảy ra một sự kiện là bao nhiêu?” Đây chính là *probability without testing* — hoàn toàn dựa trên giả định mô hình, không dựa vào dữ liệu thực tế.

Trong khi đó, **Likelihood** xuất hiện ở tình huống ngược lại: **khi đã quan sát dữ liệu**, và muốn biết tham số mô hình có hợp lý không. Ví dụ: bạn tung đồng xu 50 lần, nhưng chỉ thấy 14 lần ra mặt ngửa. Điều này khiến bạn nghi ngờ rằng giả định $P(\text{Head}) = 0.5$ có thể không đúng. Lúc này, ta hỏi: “Với dữ liệu này, giá trị nào của p là hợp lý nhất?” Đây là *probability after testing*. Dữ liệu đã cố định, còn tham số p là thứ thay đổi. Likelihood đo độ hợp lý của từng giá trị p trong việc sinh ra dữ liệu quan sát.

Tóm lại:

Probability: mô hình cố định → dữ liệu thay đổi, Likelihood: dữ liệu cố định → mô hình thay đổi.

Và Logistic Regression dùng Likelihood để tìm ra tham số tốt nhất.

2.4 Bernoulli Distribution: Mô hình chính xác cho phân loại nhị phân

Trong phân loại nhị phân, y chỉ có hai giá trị 0 hoặc 1. Đây chính xác là định nghĩa của biến Bernoulli. Phân phối Bernoulli được viết:

$$P(Y = 1) = p, \quad P(Y = 0) = 1 - p.$$

Để gọn hơn, ta viết dưới dạng hợp nhất:

$$P(Y = y) = p^y(1 - p)^{1-y}.$$

Điều này phản ánh chính xác ý nghĩa của y :

- nếu $y = 1$, biểu thức còn lại là p ,
- nếu $y = 0$, biểu thức còn lại là $1 - p$.

2.5 Mô hình Logistic Regression: Mô hình hóa xác suất $P(y = 1 | x)$

Ý tưởng của Logistic Regression đơn giản:

$$p = P(y = 1 | x)$$

Nhưng vì $\theta^T x$ có thể âm hoặc dương, ta cần một hàm ép giá trị về đoạn $[0, 1]$. Sigmoid làm điều này rất tự nhiên:

$$h_\theta(x) = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = \theta^T x.$$

2.6 Xây dựng Likelihood cho toàn bộ tập dữ liệu

Trong bối cảnh mô hình hóa bằng Bernoulli, ta giả định rằng mỗi điểm dữ liệu $(x^{(i)}, y^{(i)})$ được sinh ra bởi một biến ngẫu nhiên có xác suất:

$$P(y^{(i)} = 1 | x^{(i)}, \theta) = h_\theta(x^{(i)}), \quad P(y^{(i)} = 0 | x^{(i)}, \theta) = 1 - h_\theta(x^{(i)}).$$

Dùng dạng hợp nhất của phân phối Bernoulli, ta có thể viết xác suất cho từng mẫu:

$$P(y^{(i)} | x^{(i)}, \theta) = h_\theta(x^{(i)})^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}.$$

Trực giác của biểu thức này:

- Nếu mẫu có nhãn $y^{(i)} = 1$, biểu thức trở thành $h_\theta(x^{(i)})$. Mô hình được “thưởng” nếu $h_\theta(x^{(i)})$ lớn (xác suất cao cho lớp đúng).
- Nếu $y^{(i)} = 0$, biểu thức trở thành $1 - h_\theta(x^{(i)})$. Mô hình được “thưởng” nếu nó giảm xác suất dự đoán dương.
- Số mũ $y^{(i)}$ và $1 - y^{(i)}$ đóng vai trò như “công tắc” (on/off). Chúng bật đúng biểu thức cần dùng cho từng nhãn. Cách viết này cực kỳ hiệu quả vì cùng một công thức xử lý được cả hai trường hợp.

2.7 Vậy Likelihood cho toàn bộ tập dữ liệu là gì?

Vì các điểm dữ liệu được giả định là **độc lập** (IID — independent and identically distributed), xác suất quan sát cả tập dữ liệu chính là:

$$\mathcal{L}(\theta) = \prod_{i=1}^m P(y^{(i)} | x^{(i)}, \theta) = \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}.$$

Tại sao dùng phép nhân (Product) thay vì phép cộng ? Còn trong xác suất, đây là điểm rất nhiều người hay nhầm khi mới học xác suất.

- Với các sự kiện độc lập, xác suất xảy ra đồng thời bằng **tích** các xác suất thành phần.
- Mỗi điểm dữ liệu đều là 1 tham số θ độc lập.
- Vì thế, để xem toàn bộ tập dữ liệu “ủng hộ” tham số θ bao nhiêu, ta phải nhân toàn bộ các xác suất lại.

Đây chính là nơi xuất hiện khái niệm “**likelihood của mô hình**” — nó là mức độ mô hình “**hợp lý**” khi nhìn vào toàn bộ dữ liệu.

Ôn tập lại kiến thức toán cơ bản

Biểu thức Likelihood sử dụng lại hai quy tắc quen thuộc:

- **Product Rule** Nếu các sự kiện độc lập,

$$P(A \cap B) = P(A)P(B).$$

- **Power Rule** Khi viết lại p hoặc $(1 - p)$ tuỳ theo nhãn y , ta dùng quy tắc:

$$a^1 = a, \quad a^0 = 1.$$

Đây là lý do công thức Bernoulli gộp được 2 trường hợp vào một.

Nhìn vào một ví dụ thực tế để hiểu rõ hơn

Giả sử ta cần dự đoán khả năng mắc ung thư dựa trên chỉ số PSA (theo ví dụ trong tài liệu bạn cung cấp). Mỗi bệnh nhân có:

- giá trị PSA (đầu vào $x^{(i)}$), - nhãn: Cancer (1) hoặc Healthy (0), - mô hình dự đoán xác suất $p^{(i)} = h_\theta(x^{(i)})$.

Ví dụ một đoạn dữ liệu (rút gọn):

PSA	y	$h_\theta(x)$
3.8	1	0.99
3.4	1	0.97
2.9	1	0.92
2.1	0	0.50
1.2	0	0.13

Likelihood của mô hình là:

$$\mathcal{L}(\theta) = 0.99^1 \cdot 0.97^1 \cdot 0.92^1 \cdot (1 - 0.50)^1 \cdot (1 - 0.13)^1.$$

Nếu nhãn đầy đủ:

$$0.99 \cdot 0.97 \cdot 0.92 \cdot 0.50 \cdot 0.87 \approx 0.386.$$

Trực giác:

- Mô hình càng chính xác \rightarrow các xác suất đúng càng cao \rightarrow tích càng lớn.
- Mô hình sai tự tin \rightarrow một số hạng rất nhỏ \rightarrow kéo toàn bộ tích xuống gần 0. Đây chính là logic cốt lõi của Likelihood.

Tới đây, ta đã xây dựng xong Likelihood. Ở phần tiếp theo, ta sẽ đổi mắt với vấn đề lớn nhất của Likelihood: *tích nhiều xác suất nhỏ khiến giá trị nhanh chóng tiến về 0*. Và đó là lúc **Log-Likelihood** xuất hiện .

2.8 Log-Likelihood: Khi phép nhân không còn chịu nổi dữ liệu thực tế

Một trong những vấn đề lớn nhất của Likelihood là giá trị của nó **giảm rất nhanh** khi nhân nhiều xác suất nhỏ lại với nhau. Trong thực tế, các mô hình phân loại thường dự đoán xác suất như 0.93, 0.71, 0.85, ... hoặc có những điểm “khó đoán” với xác suất 0.02 hoặc 0.15.

Khi nhân 100–10,000 giá trị như vậy, ta sẽ thu được những con số nhỏ tới mức:

$$10^{-50}, 10^{-200}, 10^{-500} \text{ hoặc thậm chí nhỏ hơn.}$$

Kết quả là máy tính sẽ tràn số (underflow), biến toàn bộ kết quả về 0. Hoặc ngược lại overflow khi số quá lớn. Điều này khiến việc tối ưu trở nên bất khả thi.

Vậy làm thế nào để biến một tích “siêu nhỏ” thành thứ có thể tính toán, đạo hàm và tối ưu được? Câu trả lời chính là: **dùng Logarithm**.

Tại sao log giúp giải quyết vấn đề ? Hiểu bằng trực giác trước

Ta có quy tắc cực kì quan trọng của log

$$\ln(a \cdot b \cdot c) = \ln(a) + \ln(b) + \ln(c).$$

Một phép nhân hàng trăm số nhỏ

$$0.99 \cdot 0.92 \cdot 0.85 \cdots$$

trở thành một phép cộng hoàn toàn vô hại:

$$\ln(0.99) + \ln(0.92) + \ln(0.85) + \cdots$$

Phép cộng luôn ổn định và dễ đạo hàm hơn phép nhân.

Không chỉ vậy, log còn “kéo” các xác suất về một dải giá trị dễ quản lý:

- $\ln(0.99) \approx -0.01$
- $\ln(0.92) \approx -0.083$
- $\ln(0.85) \approx -0.16$

Các con số nhỏ gọn, không bị “lọt” vào vùng quá nhỏ như 10^{-50} .

Dùng log để viết lại Likelihood

Ta đã có Likelihood gốc:

$$\mathcal{L}(\theta) = \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}.$$

Lấy log hai vế (dùng Product Rule và Power Rule):

$$\ln(a \cdot b) = \ln(a) + \ln(b), \quad \ln(a^k) = k \ln(a).$$

Ta được

$$\ell(\theta) = \ln \mathcal{L}(\theta) = \sum_{i=1}^m \left[y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_\theta(x^{(i)})) \right].$$

Hàm được gọi là “Log-Likelihood”? vì

- $\mathcal{L}(\theta)$ là Likelihood (một phép nhân).
- $\ell(\theta) = \ln(\mathcal{L}(\theta))$ là Log-Likelihood (một phép cộng). Điều quan trọng nhất là Hàm log là hàm đơn điệu, tức là việc tối đa hóa Likelihood và tối đa hóa Log-Likelihood là như nhau:

$$\theta^* = \arg \max_{\theta} \ell(\theta) \iff \theta^* = \arg \max_{\theta} \mathcal{L}(\theta).$$

Tóm lại mình mất công đổi qua log là vì: Log-Likelihood dễ tính loss hơn, dễ đạo hàm hơn, dễ phân tích hình dạng hơn, mượt hơn cho việc tối ưu.

2.9 Một ví dụ rõ ràng: Log-Likelihood trong bài toán dự đoán ung thư

Giả sử bạn có 5 bệnh nhân, nhãn như sau (giống ví dụ ở phần Likelihood):

PSA	y	$h_\theta(x)$
3.8	1	0.99
3.4	1	0.97
2.9	1	0.92
2.1	0	0.50
1.2	0	0.13

Ta đã tính được Likelihood trước đó:

$$\mathcal{L}(\theta) \approx 0.386.$$

Giờ tính Log-Likelihood:

$$\ell(\theta) = \ln(0.99) + \ln(0.97) + \ln(0.92) + \ln(0.50) + \ln(0.87).$$

Tính từng giá trị:

$$\begin{aligned}\ln(0.99) &\approx -0.01005, \\ \ln(0.97) &\approx -0.03046, \\ \ln(0.92) &\approx -0.08338, \\ \ln(0.50) &\approx -0.69315, \\ \ln(0.87) &\approx -0.13926.\end{aligned}$$

Cộng lại:

$$\ell(\theta) \approx -0.956.$$

So sánh trực giác:

- Likelihood: $0.386 \rightarrow$ rất nhỏ nhưng khó so sánh giữa mô hình.

- Log-Likelihood: $-0.956 \rightarrow$ con số “tự nhiên”, dễ đọc, dễ so sánh.

Nếu một mô hình khác có LL = -10 \rightarrow chắc chắn tệ hơn.

Nếu LL = -0.3 \rightarrow tốt hơn.

Ý nghĩa sâu hơn: Log-Likelihood thường/phạt mô hình thế nào?

Quan sát cấu trúc:

$$y \ln(h) \quad \text{và} \quad (1 - y) \ln(1 - h).$$

Nếu $y = 1$:

$$\ell_i = \ln(h_\theta(x^{(i)})).$$

Nếu mô hình dự đoán đúng và tự tin (h gần 1), log gần 0 \rightarrow **điểm thưởng lớn**. Nếu dự đoán sai tự tin (h gần 0), log tiến về $-\infty \rightarrow$ **phạt rất mạnh**.

Nếu $y = 0$:

$$\ell_i = \ln(1 - h_\theta(x^{(i)})).$$

- Nếu mô hình gần 0 \rightarrow tốt

- Nếu mô hình gần 1 \rightarrow log rất âm \rightarrow phạt mạnh. Đây chính là lý do Log-Likelihood phản ánh tốt chất lượng mô hình phân loại: mô hình sai tự tin sẽ bị trừ phạt rất nặng.

2.10 Binary Cross-Entropy: Hàm Loss xuất phát tự nhiên từ Log-Likelihood

Sau khi đã xây dựng Log-Likelihood, ta có một biểu thức mô tả mức độ “hợp lý” của tham số θ khi nhìn vào toàn bộ dữ liệu. Tuy nhiên, trong học máy, ta thường quen với *tối thiểu hóa* thay vì *tối đa hóa*. Vì vậy, theo lẽ rất tự nhiên là tối thiểu hàm Loss:

$$J(\theta) = -\ell(\theta).$$

Biểu thức này chính là **Binary Cross-Entropy (BCE)**. Không phải được định nghĩa tùy ý, mà là kết quả tất yếu khi ta lấy dấu trừ của log-likelihood.

$$J(\theta) = - \sum_{i=1}^m \left[y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln(1 - h_\theta(x^{(i)})) \right].$$

Nếu mẫu thuộc lớp 1 ($y = 1$), ta nhìn vào $-\ln(h_\theta(x))$.

- Nếu mô hình “tự tin đúng” ($h \approx 1$), $-\ln(h)$ gần 0 → **ít phạt**.
- Nếu mô hình “tự tin sai” ($h \approx 0$), $-\ln(h)$ tiến tới $+\infty$ → **phạt cực mạnh**.

Nếu mẫu thuộc lớp 0 ($y = 0$), ta nhìn vào $-\ln(1 - h_\theta(x))$.

- Nếu mô hình “tự tin đúng” ($h \approx 0$), $-\ln(1 - h)$ gần 0.
- Nếu mô hình “tự tin sai” ($h \approx 1$), $-\ln(1 - h)$ tiến tới $+\infty$.

2.11 Tổng kết bước của Logistic Regression

Để hiểu sâu Binary Cross-Entropy, ta cần nhìn lại toàn bộ mô hình Logistic Regression ở dạng cơ bản nhất. Mục tiêu của Logistic Regression là mô hình hóa xác suất một điểm dữ liệu thuộc lớp 1 dưới dạng:

$$h_\theta(x) = P(y = 1 | x; \theta).$$

Hãy phân tích từng thành phần một cách liền mạch và trực quan.

(1) Mô hình tuyến tính: $z = \theta^T x$

$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n.$$

Trong đó, x là vector đặc trưng đầu vào, θ là vector trọng số mô hình cần học và $x_0 = 1$ để biểu diễn hệ số chẵn (bias). Mô hình lấy tổng có trọng số của các đặc trưng để tạo thành một dạng “điểm số” có thể âm, dương hoặc rất lớn, nhưng chưa thể diễn giải như xác suất; vì thế ta cần đến hàm Sigmoid.

(2) Biến đổi z thành xác suất bằng hàm Sigmoid

$$h_\theta(x) = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Sigmoid ép đầu ra về đoạn $[0, 1]$, cho phép diễn giải như xác suất: khi z lớn thì $h \rightarrow 1$, khi z nhỏ thì $h \rightarrow 0$, còn khi $z = 0$ thì $h = 0.5$ thể hiện trạng thái “không biết / trung lập”. Vì vậy, Sigmoid đóng vai trò như bộ chuyển đổi giữa không gian tuyến tính và không gian xác suất.

(3) Xác suất dự đoán cho từng nhãn y

Theo phân phối Bernoulli:

$$P(y | x, \theta) = h_\theta(x)^y (1 - h_\theta(x))^{1-y}.$$

Công thức này hoạt động nhờ số mũ y và $1 - y$ đóng vai trò như “công tắc”: khi $y = 1$ nó giữ lại h , còn khi $y = 0$ nó giữ lại $(1 - h)$. Như vậy, một công thức duy nhất có thể biểu diễn cả hai trường hợp.

(4) Likelihood cho cả tập dữ liệu

Vì các mẫu độc lập theo giả định IID, ta nhân tất cả xác suất lại để đo độ “hợp lý tổng thể” của mô hình:

$$\mathcal{L}(\theta) = \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}.$$

Phép nhân xuất hiện vì mỗi điểm dữ liệu là một bằng chứng độc lập ủng hộ hoặc bác bỏ θ , và tích của chúng thể hiện mức độ mô hình phù hợp với toàn bộ dữ liệu.

(5) Log-Likelihood

Tích trên rất nhỏ khi nhân nhiều xác suất \rightarrow dẫn đến underflow, nên ta lấy log để chuyển tích thành tổng dễ tính và ổn định:

$$\ell(\theta) = \sum_{i=1}^m \left[y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_\theta(x^{(i)})) \right].$$

Việc lấy log giúp số nhỏ trở nên lớn hơn, loại bỏ vấn đề tràn số, đồng thời giữ nguyên điểm cực đại vì hàm log là hàm đơn điệu.

(6) Binary Cross-Entropy: NegLogLikelihood

$$J(\theta) = -\ell(\theta).$$

$$J(\theta) = - \sum_{i=1}^m \left[y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln (1 - h_\theta(x^{(i)})) \right].$$

Việc thêm dấu trừ biến bài toán tối đa hóa thành tối thiểu hóa, giúp BCE trở thành hàm Loss chuẩn. BCE thường mô hình đoán đúng và phạt rất mạnh mô hình đoán sai tự tin, đồng thời phản ánh đúng bản chất thống kê của phân phối Bernoulli.

Giải thích pipeline của Logistic Regression theo trực giác liền mạch

- $\theta^T x$ tạo ra một điểm số tuyến tính dựa trên đặc trưng đầu vào.
- $\sigma(\theta^T x)$ chuyển điểm số đó thành xác suất hợp lệ nằm trong khoảng $[0, 1]$.
- $h_\theta(x)^y (1 - h_\theta(x))^{1-y}$ mô tả xác suất mô hình “cho điểm” đúng sai cho từng mẫu dữ liệu.
- $\prod_{i=1}^m$ kết hợp tất cả bằng chứng độc lập thành một mức độ “hợp lý tổng thể” của mô hình.
- log biến tích thành tổng để tránh underflow và làm cho việc đạo hàm dễ dàng hơn.

- log biến bài toán thành tối thiểu hóa, phù hợp với các thuật toán tối ưu chuẩn như Gradient Descent.
- Tổng cuối cùng trở thành một giá trị Loss: mô hình càng chính xác thì Loss càng nhỏ.

Kết luận cõi động

$$\text{Logistic Regression} = \text{Bernoulli Model} + \text{Log-Likelihood} \Rightarrow \text{Binary Cross-Entropy.}$$

Mọi biểu thức đều xuất phát từ nguyên lý xác suất.

Phần 3 Mở rộng: Nhân ma trận với Logistic Regression

Ở phần trước, ta đã xây dựng được hàm Loss của Logistic Regression ở dạng từng mẫu (không dùng ma trận). Tuy nhiên, **khi số lượng đặc trưng và số lượng mẫu trở nên lớn, cách viết thường minh theo từng điểm dữ liệu trở nên cồng kềnh và khó tối ưu hóa**. Để xử lý mô hình ở quy mô lớn, ta cần chuyển toàn bộ bài toán về dạng **vector–matrix**. Ngoài ra, nó cũng giống việc viết hàm tính toán trong code gọn và dễ dàng hơn.

3.1 Khởi động bằng một bài toán cụ thể

Giả sử ta có ba bệnh nhân với ba giá trị PSA và ta muốn mô hình Logistic Regression dự đoán xác suất mắc bệnh. Bảng dữ liệu:

Bệnh nhân	$x^{(i)}$	$y^{(i)}$
1	1.0	1
2	2.0	1
3	3.0	0

Hệ số mô hình:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \end{bmatrix}.$$

Ta muốn tính:

$$h_\theta(x^{(i)}) = \sigma(\theta_0 + \theta_1 x^{(i)}).$$

Nếu làm thủ công từng mẫu:

$$\begin{aligned} z^{(1)} &= -3 + 2(1) = -1, & h^{(1)} &= \sigma(-1), \\ z^{(2)} &= -3 + 2(2) = 1, & h^{(2)} &= \sigma(1), \\ z^{(3)} &= -3 + 2(3) = 3, & h^{(3)} &= \sigma(3). \end{aligned}$$

Cách này hoàn toàn đúng nhưng không thể mở rộng cho 1000 mẫu, 20 đặc trưng.

3.2 Ôn lại nhân ma trận từ góc nhìn “tổng trọng số”

Nhân ma trận chỉ là:

$$(\hàng\text{ của ma trận}) \cdot (\text{vector}) = \text{tổng có trọng số}.$$

Ví dụ:

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = 1 \cdot 3 + 2 \cdot 4 = 11.$$

Trực giác:

- ma trận là tập hợp nhiều hàng,
- mỗi hàng là một mẫu dữ liệu,
- nhân ma trận với vector θ đơn giản là tính điểm số tuyến tính z cho tất cả mẫu cùng lúc.

3.3 Chuyển dữ liệu sang dạng ma trận

Ta gom tất cả đầu vào vào một ma trận thiết kế:

$$X = \begin{bmatrix} 1 & 1.0 \\ 1 & 2.0 \\ 1 & 3.0 \end{bmatrix}.$$

- Cột đầu tiên là 1 (cho bias term θ_0).
- Mỗi hàng là một bệnh nhân.

Khi đó:

$$z = X\theta = \begin{bmatrix} 1 & 1.0 \\ 1 & 2.0 \\ 1 & 3.0 \end{bmatrix} \begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 3 \end{bmatrix}.$$

3.4 Áp sigmoid lên toàn bộ vector cùng lúc

$$h = \sigma(z) = \begin{bmatrix} \sigma(-1) \\ \sigma(1) \\ \sigma(3) \end{bmatrix}.$$

Bây giờ ta đã tính được xác suất cho toàn bộ dataset chỉ bằng hai bước:

- nhân ma trận: $z = X\theta$,
- áp sigmoid: $h = \sigma(X\theta)$.

3.5 Loss của Logistic Regression ở dạng vector

Theo định nghĩa ở phần trước, hàm Loss cho Logistic Regression ở dạng từng mẫu là:

$$J(\theta) = - \sum_{i=1}^m \left[y^{(i)} \ln h^{(i)} + (1 - y^{(i)}) \ln(1 - h^{(i)}) \right].$$

Để xử lý toàn bộ dữ liệu đồng thời, ta chuyển sang dạng vector hoá. Khi đó:

$$J(\theta) = - (y^T \ln(h) + (1 - y)^T \ln(1 - h)),$$

trong đó:

- $y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]^T$ là vector nhãn.
- $h = [h^{(1)}, h^{(2)}, \dots, h^{(m)}]^T$ là vector xác suất mô hình dự đoán.

- $\ln(h)$ là log từng phần tử:

$$\ln(h) = [\ln h^{(1)}, \ln h^{(2)}, \dots, \ln h^{(m)}]^T.$$

Công thức vector hoá này giúp Loss được viết gọn gàng hơn và cho phép tính gradient và hessian dễ dàng.

Ví dụ đầy đủ: Tính toàn bộ Loss bằng vector cho bài toán PSA

Ta sử dụng lại dataset nhỏ gồm 3 bệnh nhân:

$$X = \begin{bmatrix} 1 & 1.0 \\ 1 & 2.0 \\ 1 & 3.0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \theta = \begin{bmatrix} -3 \\ 2 \end{bmatrix}.$$

Bước 1: Tính $z = X\theta$

$$z = \begin{bmatrix} 1 & 1.0 \\ 1 & 2.0 \\ 1 & 3.0 \end{bmatrix} \begin{bmatrix} -3 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 3 \end{bmatrix}.$$

Bước 2: Tính $h = \sigma(z)$

$$h = \begin{bmatrix} \sigma(-1) \\ \sigma(1) \\ \sigma(3) \end{bmatrix} = \begin{bmatrix} 0.2689 \\ 0.7311 \\ 0.9526 \end{bmatrix}.$$

Bước 3: Tính log từng phần tử

$$\ln(h) = \begin{bmatrix} \ln 0.2689 \\ \ln 0.7311 \\ \ln 0.9526 \end{bmatrix} = \begin{bmatrix} -1.313 \\ -0.313 \\ -0.048 \end{bmatrix},$$

$$\ln(1 - h) = \begin{bmatrix} \ln(1 - 0.2689) \\ \ln(1 - 0.7311) \\ \ln(1 - 0.9526) \end{bmatrix} = \begin{bmatrix} -0.314 \\ -1.313 \\ -3.046 \end{bmatrix}.$$

Bước 4: Tính từng phần của Loss

$$y^T \ln(h) = [1 \ 1 \ 0] \begin{bmatrix} -1.313 \\ -0.313 \\ -0.048 \end{bmatrix} = (-1.313) + (-0.313) + (0) = -1.626.$$

$$(1 - y)^T \ln(1 - h) = [0 \ 0 \ 1] \begin{bmatrix} -0.314 \\ -1.313 \\ -3.046 \end{bmatrix} = -3.046.$$

Bước 5: Loss cuối cùng

$$J(\theta) = -(-1.626 + (-3.046)) = 4.672.$$

Ý nghĩa ví dụ

- Hai mẫu đầu có nhãn 1 nhưng mô hình dự đoán không quá cao (0.26 và 0.73), dẫn đến giá trị $\ln(h)$ âm đáng kể.
- Mẫu thứ ba có nhãn 0 nhưng mô hình lại dự đoán gần 1 (0.95), dẫn đến $\ln(1 - h)$ rất âm và tạo Loss lớn.
- Loss tổng cộng 4.672 là khá cao cho dataset nhỏ như thế này, phản ánh mô hình chưa tốt.

Nhờ ví dụ này, ta thấy rõ:

- dạng vector hoá của Logistic Regression hoàn toàn tương đương dạng từng mẫu,
- nhưng cô đọng và phù hợp cho việc triển khai bằng phần mềm và cho phân tích đạo hàm vector ở phần sau.

Kết nối sang phần Đạo hàm và Hessian

Ngay sau đây, khi tính đạo hàm của Loss, ta sẽ thấy dạng vector đem lại lợi thế vượt trội.

Công thức đẽp:

$$\nabla_{\theta} J(\theta) = X^T(h - y)$$

3.6 Convexity of BCE for Multiple-Variable using Hessian Matrix

Ở phần trước, ta đã thiết lập Logistic Regression dưới dạng vector và tính được Gradient của hàm Loss. Phần này sẽ mở rộng lên đạo hàm bậc hai (Second Derivative) để xây dựng **Hessian Matrix** cho ta tinh chỉnh nhiều θ weight cùng 1 lúc, giúp tìm loss hiệu quả hơn, từ đó chứng minh rằng Binary Cross-Entropy (BCE) là một hàm convex.

1. Ôn lại Second Derivative: từ 1 biến đến nhiều biến

Trong hàm một biến $f(x)$:

$$f'(x) = \text{tốc độ thay đổi của } f(x), \quad f''(x) = \text{tốc độ thay đổi của tốc độ thay đổi.}$$

Second Derivative

	1 variable	2 variables
Function	$f(x)$	$f(x, y)$
First derivative	$f'(x)$ Rate of change of $f(x)$	$f_x(x, y)$ Rate of change w.r.t x $f_y(x, y)$ Rate of change w.r.t y $\nabla f = \begin{bmatrix} f_x(x, y) \\ f_y(x, y) \end{bmatrix}$
Second derivative	$f''(x)$ Rate of change of the rate of change of $f(x)$???

Figure 22: So sánh đạo hàm bậc nhất và bậc hai giữa một biến và hai biến. (w.r.t x - with respect to / đạo hàm 1 phần theo x)

Với hàm hai biến $f(x, y)$, đạo hàm bậc nhất trở thành vector gradient:

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}.$$

Vậy đạo hàm bậc hai của hàm hai biến trông thế nào?

Ta sẽ xem xét một ví dụ cụ thể để trực giác rõ hơn.

2. Ví dụ thực tế: Tính đạo hàm bậc hai cho $f(x, y)$

Xét hàm:

$$f(x, y) = 2x^2 + 3y^2 - xy.$$

Second Derivative

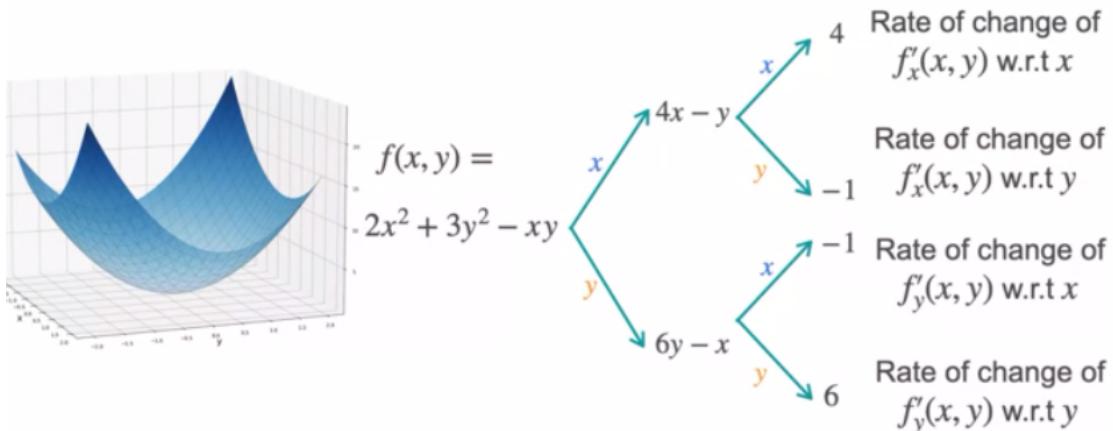


Figure 23: Cây đạo hàm bậc nhất và bậc hai cho hàm $f(x, y) = 2x^2 + 3y^2 - xy$.

Đạo hàm bậc nhất:

$$f_x = 4x - y, \quad f_y = 6y - x.$$

Đạo hàm bậc hai:

$$f_{xx} = 4, \quad f_{yy} = 6, \quad f_{xy} = f_{yx} = -1.$$

What Do These Mean?

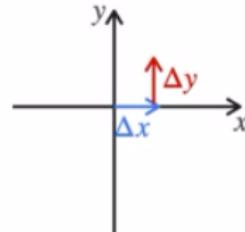
$f_x(x, y)$ w.r.t x $f_y(x, y)$ w.r.t y	Change in the change in the function w.r.t tiny changes in x and y	Same idea as with one variable!
$f_x(x, y)$ w.r.t y $f_y(x, y)$ w.r.t x	1. Change in the slope along one coordinate axis w.r.t tiny changes along an orthogonal coordinate axis 2. They are the same! (In most cases)	

Figure 24: Ý nghĩa của các đạo hàm bậc hai và đạo hàm chéo.

Các đạo hàm bậc hai mô tả “change in the change”: thay đổi của đạo hàm theo một hướng khi ta biến đổi theo một hướng khác (có thể trực giao - orthogonal - 2 vector vuông góc).

3. Ký hiệu đạo hàm bậc hai: Leibniz và Lagrange

Hình 25 minh họa hai kiểu ký hiệu:

Leibniz's notation	Lagrange's notation
$\frac{\partial^2 f}{\partial x^2}$	$f_{xx}(x, y)$
$\frac{\partial^2 f}{\partial y^2}$	$f_{yy}(x, y)$
$\frac{\partial^2 f}{\partial x \partial y}$	$f_{xy}(x, y)$
$\frac{\partial^2 f}{\partial y \partial x}$	$f_{yx}(x, y)$

Figure 25: Hai hệ ký hiệu cho đạo hàm bậc hai: Leibniz và Lagrange.

- Leibniz: $\frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial x \partial y}$.
- Lagrange: f_{xx}, f_{xy} .

4. Gom toàn bộ đạo hàm bậc hai vào một cấu trúc: Hessian Matrix

Như minh họa ở Hình 26:

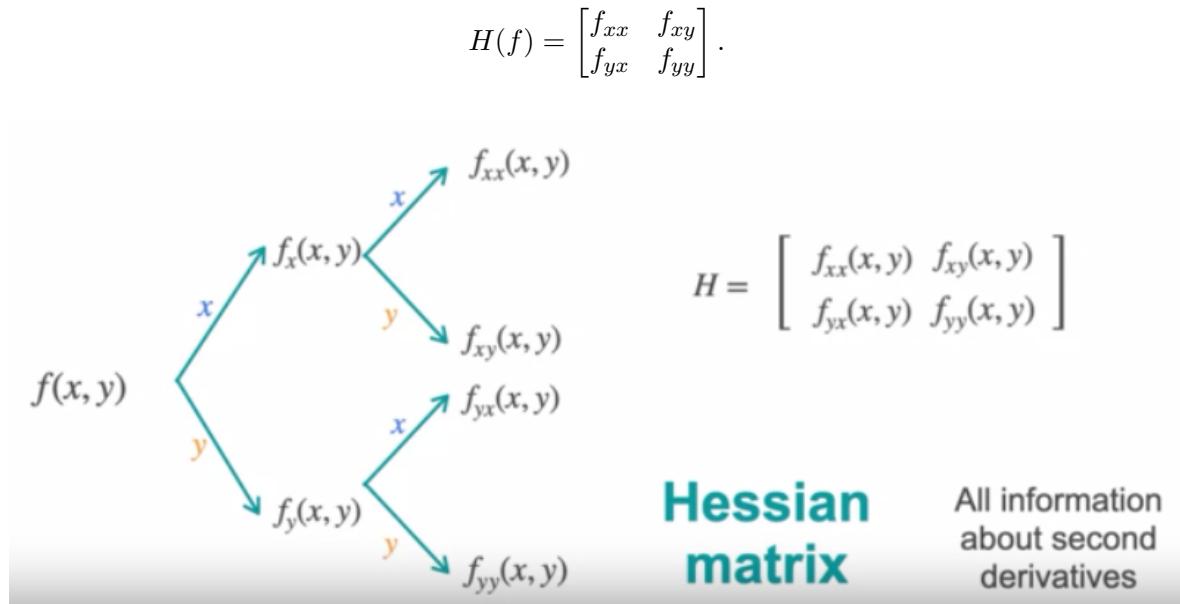


Figure 26: Cách gom các đạo hàm bậc hai vào ma trận Hessian.

Hessian chứa toàn bộ thông tin về độ cong (curvature) của hàm $f(x, y)$. Tương tự như lúc ta cùng giải thích đạo hàm bậc 2 ở phần trước.

5. Dùng Hessian để phân loại điểm tối ưu

Bằng cách phân tích ma trận Hessian, ta có thể xác định 1 điểm là cực tiểu, cực đại hay là điểm yên ngựa (khi cực tiểu của hàm bé hơn 0, nhưng đạo hàm ra 0) dựa vào Eigen value của nó ((giá trị riêng của ma trận).

! Bonus Fact: By analyzing the Hessian matrix, we can determine whether a point is a minimum, maximum, or a saddle point

Second Derivative

	1 variable		2 variables	
Function	$f(x)$		$f(x, y)$	
First derivative	$f'(x)$	Rate of change of $f(x)$	$f_x(x, y)$	Rate of change w.r.t x
			$f_y(x, y)$	Rate of change w.r.t y
			$\nabla f = \begin{bmatrix} f_x(x, y) \\ f_y(x, y) \end{bmatrix}$	
Second derivative	$f''(x)$	Rate of change of the rate of change of $f(x)$	$H(x, y) = \begin{bmatrix} f_{xx}(x, y) & f_{xy}(x, y) \\ f_{yx}(x, y) & f_{yy}(x, y) \end{bmatrix}$	

Figure 27: So sánh bậc hai trong 1 biến và nhiều biến.

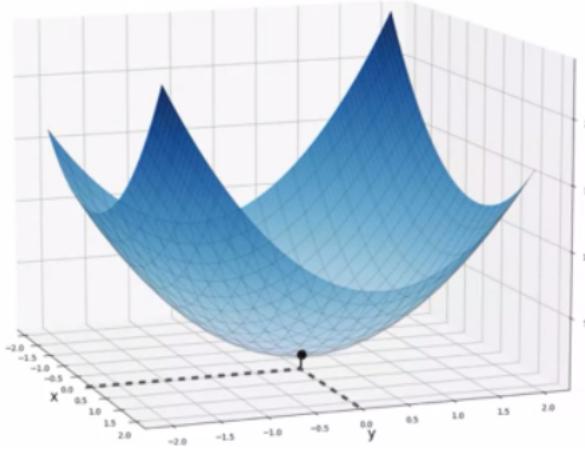
Quy tắc phân loại:

- nếu mọi eigenvalue của Hessian $> 0 \rightarrow$ hàm cong lên (local minimum),
- nếu mọi eigenvalue $< 0 \rightarrow$ cong xuống (local maximum),
- nếu có eigenvalue > 0 và $< 0 \rightarrow$ saddle point,
- nếu có eigenvalue $= 0 \rightarrow$ không đủ thông tin.

Ví dụ minh họa qua cách xác định cực tiểu dựa trên giá trị của ma trận Hessian:

Concave Up (cực tiểu)

Concave Up



$$f(x, y) = 2x^2 + 3y^2 - xy$$

$$H(0,0) = \begin{bmatrix} 4 & -1 \\ -1 & 6 \end{bmatrix}$$

$$\begin{aligned} \det(H(0,0) - \lambda I) &= \det \left(\begin{bmatrix} 4-\lambda & -1 \\ -1 & 6-\lambda \end{bmatrix} \right) \\ &= (4-\lambda)(6-\lambda) - (-1)(-1) \\ &= \lambda^2 - 10\lambda + 23 \end{aligned}$$

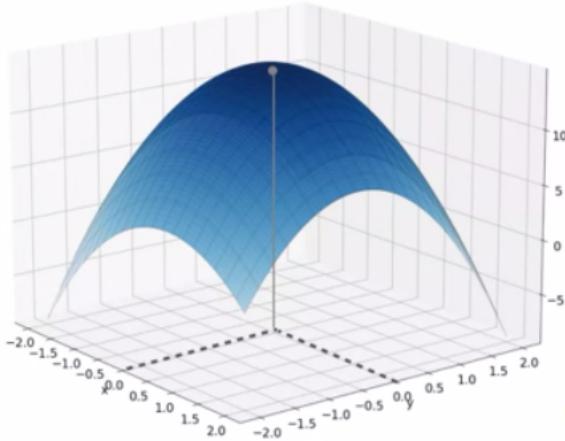
$$\lambda_1 = 6.41$$

$$\lambda_2 = 3.59$$

Figure 28: Hessian có eigenvalue dương \rightarrow concave up \rightarrow điểm $(0,0)$ là cực tiểu.

Concave Down (cực đại)

Concave Down



$$f(x, y) = -2x^2 - 3y^2 - xy + 15$$

$$\nabla f(x, y) = \begin{bmatrix} -4x - y \\ -x - 6y \end{bmatrix}$$

$$H(0,0) = \begin{bmatrix} -4 & -1 \\ -1 & -6 \end{bmatrix}$$

$$\begin{aligned} \det(H(0,0) - \lambda I) &= \\ &(-4-\lambda)(-6-\lambda) - (-1)(-1) \\ &= \lambda^2 + 10\lambda + 23 \end{aligned}$$

$$\lambda_1 = -3.59$$

$$\lambda_2 = -6.41$$

(0,0) is a maximum!

< 0

Figure 29: Hessian có eigenvalue âm \rightarrow concave down \rightarrow cực đại.

Saddle Point (Điểm Yên Ngựa) Khi đạo hàm bằng 0 thì ngoài các trường hợp rơi vào các điểm cực trị thì còn một điểm nữa chính là điểm yên ngựa. Một điểm yên ngựa là điểm mà tại đó đạo hàm

bằng 0 nhưng không phải là một điểm cực tiểu. Xét hàm số $f(x) = 5^3$ hàm số này có đạo hàm bằng 0 tại $x=0$ nhưng $x=0$ lại không phải giá trị cực tiểu của hàm số này vì nó có thể âm.

Saddle Point

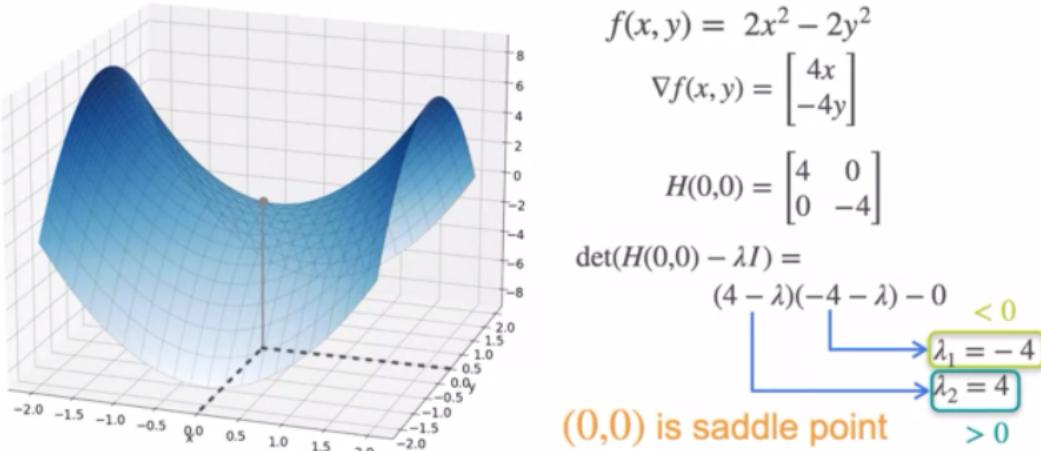


Figure 30: Eigenvalue trái dấu → saddle point.

6. Ứng dụng trực tiếp vào Logistic Regression

Sau khi đã hiểu Hessian của hàm hai biến ở các phần trước, ta quay trở lại Logistic Regression để xây dựng Hessian của Binary Cross-Entropy (BCE). Toàn bộ quá trình dưới đây được viết cực kỳ chi tiết nhằm giúp người đọc “ôn lại toán từ gốc”.

6.1 Nhắc lại dạng Loss và Gradient

Hàm Loss dạng vector của Logistic Regression:

$$J(\theta) = -[y^T \ln(h) + (1-y)^T \ln(1-h)], \quad h = \sigma(X\theta).$$

Gradient (đạo hàm bậc nhất theo vector θ):

$$\nabla_{\theta} J(\theta) = X^T(h - y).$$

BCE là convex nếu Hessian (đạo hàm bậc hai) của $J(\theta)$ là Positive Semi-Definite (i.e. chỉ cần các trị riêng của nó là không âm là positive semi-definite / bán dương).

6.2 Bắt đầu từ Gradient để tìm Hessian

Ta cần đạo hàm của:

$$X^T(h - y)$$

theo θ .

6.3 Đạo hàm của $h = \sigma(X\theta)$

Đầu tiên ta xét:

$$z = X\theta.$$

Giải thích: Đây là phép nhân ma trận cơ bản: mỗi hàng của X nhân với vector θ tạo nên một giá trị $z^{(i)}$. Nhớ rằng với ma trận $(m \times n)$ nhân vector $(n \times 1)$:

$$(X\theta)_i = \sum_{j=1}^n X_{ij}\theta_j.$$

(a) Đạo hàm của Sigmoid

Sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Đạo hàm:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)).$$

Khi áp dụng cho vector:

$$h = \begin{bmatrix} h^{(1)} \\ h^{(2)} \\ \vdots \\ h^{(m)} \end{bmatrix}, \quad h^{(i)} = \sigma(z^{(i)}).$$

Do đó:

$$\frac{\partial h^{(i)}}{\partial z^{(i)}} = h^{(i)}(1 - h^{(i)}).$$

(b) Đạo hàm của $z = X\theta$

Ví:

$$z^{(i)} = X_{i,:}\theta$$

trong đó i là hàng, : nghĩa là mọi cột nên:

$$\frac{\partial z^{(i)}}{\partial \theta} = X_{i,:}$$

Đây là quy tắc tuyến tính quen thuộc: đạo hàm của một tổng tuyến tính theo vector chính là vector hệ số.

(c) Áp dụng Chain Rule để lấy $\frac{\partial h}{\partial \theta}$

Theo chain rule đa biến:

$$\frac{\partial h^{(i)}}{\partial \theta} = \frac{\partial h^{(i)}}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial \theta}.$$

Thay vào:

$$\frac{\partial h^{(i)}}{\partial \theta} = h^{(i)}(1 - h^{(i)}) \cdot X_{i,:}$$

Gộp tất cả mẫu thành một ma trận:

$$\frac{\partial h}{\partial \theta} = DX,$$

với:

$$D = \text{diag} \left(h^{(i)}(1 - h^{(i)}) \right).$$

trong đó $\text{diag}()$ là đường chéo của ma trận, đại diện cho các giá trị Eigenvalue (vector riêng) của mỗi phép tính trong ma trận.

6.4 Bây giờ đạo hàm Gradient để tìm Hessian

Gradient (đạo hàm bậc 1):

$$\nabla_{\theta} J(\theta) = X^T(h - y).$$

Đạo hàm bậc 2 theo θ :

$$H = \nabla_{\theta}^2 J(\theta) = X^T \frac{\partial h}{\partial \theta}.$$

Thay $\frac{\partial h}{\partial \theta} = DX$:

$$H = X^T D X.$$

Đây chính là Hessian của BCE.

6.5 Giải thích hình thức của $H = X^T D X$

Để hiểu sâu hơn, ta phân rã theo từng lớp:

- X : mô tả đặc trưng của dữ liệu.
- X^T : tổng hợp nghịch đảo từng tham số θ_j .
- D : mô tả độ cong của Sigmoid tại từng mẫu, vì $h(1 - h)$ chính là độ nhạy (sensitivity) của Sigmoid.

Do đó $X^T D X$ là cách mô hình “kết hợp” độ cong từ Sigmoid với cấu trúc của dữ liệu.

Hình 26 đã minh họa việc gom các đạo hàm bậc hai vào ma trận Hessian; ta đang làm điều tương tự nhưng mở rộng lên nhiều biến và nhiều mẫu.

6.6 Vì sao D luôn không âm (PSD component)?

Ta nhắc lại:

$$D_{ii} = h^{(i)}(1 - h^{(i)}).$$

Ta biết rằng Sigmoid trả về giá trị từ 0 đến 1:

$$0 < h^{(i)} < 1.$$

Do đó:

$$h^{(i)}(1 - h^{(i)}) > 0 \quad \text{và} \quad h^{(i)}(1 - h^{(i)}) \leq \frac{1}{4}.$$

Nghĩa là mỗi phần tử đường chéo của D luôn không âm.

Vì vậy, D là một ma trận đường chéo **Positive Semi-Definite**.

6.7 Chứng minh Hessian PSD: phân rã từng bước

Ta cần chứng minh:

$$v^T H v \geq 0 \quad \forall v.$$

Thay $H = X^T D X$:

$$v^T X^T D X v.$$

Nhóm lại:

$$(Xv)^T D (Xv).$$

Đặt:

$$z = Xv.$$

Khi đó:

$$z^T D z = \sum_{i=1}^m D_{ii} z_i^2.$$

Vì: $D_{ii} \geq 0$ và $z_i^2 \geq 0$

nên mỗi số hạng đều không âm \rightarrow tổng cũng không âm:

$$\Rightarrow z^T D z \geq 0.$$

Suy ra:

H là Positive Semi-Definite.

Và do đó:

$J(\theta)$ là một hàm convex.

6.8 Trực giác hình học cuối cùng

- Sigmoid luôn có độ cong dương: $h(1 - h)$.
- Dữ liệu chỉ định hướng của độ cong thông qua X .
- $H = X^T D X$ thu gom toàn bộ độ cong của Sigmoid theo mọi hướng trong không gian tham số.
- Không tồn tại hướng nào mà Loss cong xuống \rightarrow không có local minima.

Convexity của BCE đến từ độ cong nội tại của Sigmoid + cấu trúc tuyến tính của mô hình.