Logistic Regression

Lê Hồng Phương
Data Science Laboratory
Vietnam National University, Hanoi
<*phuonglh@hus.edu.vn*>

October 25, 2021

Content

Logistic Regression

- Xét bài toán phân loại nhị phân, mỗi đối tượng $\mathbf{x}$ cần được phân vào một trong hai lớp $y \in \{0, 1\}$.
- Ta chọn hàm dự báo $h_\theta(\mathbf{x})$ như sau:

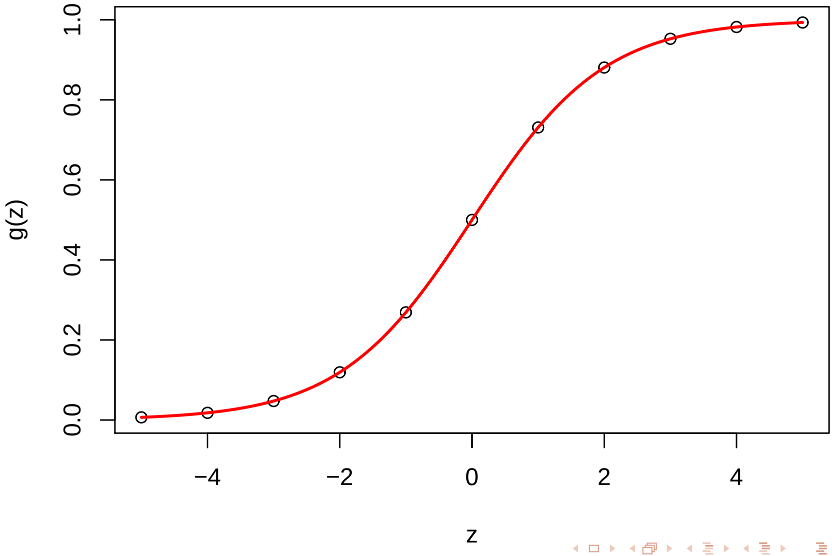
$$h_\theta(\mathbf{x}) = g(\theta^\top \mathbf{x}) = \frac{1}{1 + \exp(-\theta^\top \mathbf{x})},$$

trong đó

$$g(z) = \frac{1}{1 + \exp(-z)}$$

được gọi là *hàm logistic* hoặc *hàm sigmoid*.

Sigmoid Function: $g(z) = \frac{1}{1+\exp(-z)}$



Sigmoid Function: $g(z) = \frac{1}{1+\exp(-z)}$

Nhận xét:

- $g(z) \to 1$ khi $z \to \infty$
- $g(z) \to 0$ khi $z \to -\infty$.
- $g(z)$ và $h_\theta(\mathbf{x})$ luôn nằm trong đoạn $[0, 1]$.

Đạo hàm của hàm logistic:

$$g'(z) = g(z)(1 - g(z)).$$

Mô hình hồi quy logistic:

$$\begin{aligned} P(y = 1 | \mathbf{x}; \theta) &= h_\theta(\mathbf{x}) \\ P(y = 0 | \mathbf{x}; \theta) &= 1 - h_\theta(\mathbf{x}) \end{aligned}$$

trong đó $\theta \in \mathbb{R}^{D+1}$ là véc-tơ tham số của mô hình.

Logistic Regression

Giả sử đã biết véc-tơ tham số θ, ta sử dụng mô hình để phân loại như sau:

- Xếp đối tượng **x** vào lớp 1 nếu

$$P(y = 1|\,\mathbf{x}; \hat{\theta}) > P(y = 0|\,\mathbf{x}; \hat{\theta}) \Leftrightarrow h_{\hat{\theta}}(\mathbf{x}) > 1/2 \Leftrightarrow \boxed{\hat{\theta}^\top \mathbf{x} > 0}.$$

- Ngược lại thì **x** được xếp vào lớp 0.

Quy tắc phân loại dựa vào một tổ hợp tuyến tính của x_j và θ_j nên mô hình hồi quy logistic thuộc dạng mô hình phân loại tuyến tính.

Training

Ta có thể viết gọn xác suất của lớp y dưới dạng

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

Giả sử rằng tập dữ liệu huấn luyện được sinh độc lập nhau, khi đó hợp lí của dữ liệu với tham số θ là

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{N} P(y_i|\,\mathbf{x}_i;\theta) \\ &= \prod_{i=1}^{N} (h_\theta(\mathbf{x}_i))^{y_i} (1 - h_\theta(\mathbf{x}_i))^{1-y_i}. \end{aligned}$$

Training

- Log của hợp lí

$$\begin{aligned}\ell(\theta) &= \log L(\theta)\\ &= \sum_{i=1}^{N}[y_i \log h_\theta(\mathbf{x}_i) + (1 - y_i)\log(1 - h_\theta(\mathbf{x}_i))].\end{aligned}$$

- Sử dụng phương pháp hợp lí cực đại để ước lượng θ, ta cần giải bài toán tối ưu:

$$\hat{\theta} = \arg\min_\theta[-\ell(\theta) + \lambda R(\theta)],$$

  trong đó $R(\theta)$ là hàm hiệu chỉnh.
- Tham số $\lambda \geq 0$ dùng để điều khiển tính cân bằng của mô hình trong việc phù hợp với dữ liệu quan sát và việc hiệu chỉnh tham số.

Regularization Methods

Có ba dạng hiệu chỉnh thường gặp:

- Nếu $R(\theta) = \|\theta\|_1 = \sum_{j=1}^{D} |\theta_j|$ thì ta có mô hình hồi quy logistic hiệu chỉnh dạng L_1.
- Nếu $R(\theta) = \|\theta\|_2 = \sum_{j=1}^{D} \theta_j^2$ thì ta có mô hình hồi quy logistic hiệu chỉnh dạng L_2.
- Nếu $R(\theta) = \sum_{j=1}^{D} \log\left(\frac{e^{\theta_j} + e^{-\theta_j}}{2}\right)$ thì ta có mô hình hồi quy logistic hiệu chỉnh dạng hyperbolic-L_1.

Iterative Methods for Optimization

Ta cần chọn θ làm cực tiểu hoá hàm mục tiêu

$$J(\theta) = -\ell(\theta) + \lambda R(\theta).$$

Hai thuật toán lặp để tìm θ:

- Thuật toán giảm gradient (ngẫu nhiên)
- Thuật toán Newton-Raphson

Gradient Descent

Ta xuất phát từ một giá trị khởi đầu nào đó của θ và lặp để cập nhật θ theo công thức

$$\theta := \theta - \alpha \nabla J(\theta),$$

trong đó $\nabla J(\theta)$ là gradient của $J(\theta)$:

$$\nabla J(\theta) = \left(\frac{\partial J(\theta)}{\partial \theta_0}, \frac{\partial J(\theta)}{\partial \theta_1}, \dots, \frac{\partial J(\theta)}{\partial \theta_D} \right).$$

Mỗi tham số θ_j được cập nhật bởi quy tắc:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}, \quad \forall j = 0, 1, \dots, D.$$

Gradient Descent

Giả sử $N = 1$, tức tập huấn luyện chỉ có một mẫu $(\mathbf{x}, y)$ và không sử dụng hiệu chỉnh. Ta có

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \left(y \frac{1}{h_\theta(\mathbf{x})} - (1 - y) \frac{1}{1 - h_\theta(\mathbf{x})} \right) \frac{\partial h_\theta(\mathbf{x})}{\partial \theta_j}.$$

Vì

$$\begin{aligned}
\frac{\partial h_\theta(\mathbf{x})}{\partial \theta_j} &= \frac{\partial g(\theta^\top \mathbf{x})}{\partial \theta_j} \\
&= g(\theta^\top \mathbf{x})(1 - g(\theta^\top \mathbf{x})) \frac{\partial (\theta^\top \mathbf{x})}{\partial \theta_j} \\
&= g(\theta^\top \mathbf{x})(1 - g(\theta^\top \mathbf{x})) x_j,
\end{aligned}$$

Gradient Descent

nên

$$\begin{aligned}\frac{\partial \ell(\theta)}{\partial \theta_j} &= [y(1 - g(\theta^\top \mathbf{x})) - (1 - y)g(\theta^\top \mathbf{x})]x_j \\ &= [y - g(\theta^\top \mathbf{x})]x_j \\ &= [y - h_\theta(\mathbf{x})]x_j.\end{aligned}$$

Từ đó

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_j} &= -\frac{\partial \ell(\theta)}{\partial \theta_j} \\ &= [h_\theta(\mathbf{x}) - y]x_j.\end{aligned}$$

Gradient Descent

Do đó, nếu chỉ có một mẫu huấn luyện $(\mathbf{x}_i, y_i)$ thì ta có quy tắc giảm gradient sau:

$$\theta_j := \theta_j - \alpha[h_\theta(\mathbf{x}_i) - y_i]x_{ij}, \quad \forall j = 0, 1, \ldots, D.$$

Về mặt trực quan, ta thấy θ_j được cập nhật tỉ lệ với độ lớn của sai số $(h_\theta(\mathbf{x}_i) - y_i)$

- Nếu sai số dự báo càng lớn thì trọng số tương ứng càng cần thay đổi nhiều
- Nếu không có sai số thì không cần cập nhật trọng số.

Batch Gradient Descent

Algorithm 1: Batch Gradient Descent for Logistic Regression

Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

Result: θ

$\theta \leftarrow \vec{0}$;

repeat

$\quad \theta \leftarrow \theta - \alpha \sum_{i=1}^{N} [h_\theta(\mathbf{x}_i) - y_i] x_i$;

until *converged*;

Stochastic Gradient Descent

Algorithm 2: Stochastic Gradient Descent for Logistic Regression

Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$

Result: θ

$\theta \leftarrow \vec{0}$;

repeat

  Shuffle the training set randomly;

  for $i = 1$ *to* N **do**

    $\theta \leftarrow \theta - \alpha[h_\theta(\mathbf{x}_i) - y_i]x_i$;

until *converged*;

L_2 Regularization

- Nếu ta dùng dạng hiệu chỉnh L_2:

$$J(\theta) = -\ell(\theta) + \frac{1}{2}\lambda \sum_{j=1}^{D} \theta_j^2.$$

- Giảm gradient theo loạt:

$$\theta_0 := \theta_0 - \alpha \sum_{i=1}^{N} [h_\theta(\mathbf{x}_i) - y_i] x_{i0}$$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{N} [h_\theta(\mathbf{x}_i) - y_i] x_{ij} - \lambda\theta_j, \quad \forall j = 1, \dots, D.$$

- Giảm gradient ngẫu nhiên:

$$\theta_0 := \theta_0 - \alpha [h_\theta(\mathbf{x}_i) - y_i] x_{i0}$$

$$\theta_j := \theta_j - \alpha [h_\theta(\mathbf{x}_i) - y_i] x_{ij} - \lambda\theta_j, \quad \forall j = 1, \dots, D.$$

Newton-Raphson Method

- Một phương pháp khác hay dùng khác để cực tiểu hoá $J(\theta)$ là sử dụng thuật toán Newton.
- Các phương pháp Newton và giả-Newton là các phương pháp thường tối ưu dùng trong giải tích số.
- Cơ sở của phương pháp này như sau:
  - Giả sử ta có hàm thực khả vi $f : \mathbb{R} \to \mathbb{R}$ và ta cần tìm $\theta \in \mathbb{R}$ sao cho $f(\theta) = 0$.
  - Phương pháp Newton cập nhật dần θ theo công thức:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}.$$

Newton-Raphson Method

Về mặt trực quan:

- Ta xấp xỉ hàm f bởi một hàm tuyến tính là tiếp tuyến của f tại giá trị θ hiện tại.
- Cập nhật giá trị mới của θ là hoành độ giao điểm giữa tiếp tuyến này với trục hoành.
- Lặp lại quá trình này cho tới khi hội tụ (sai số cập nhật đủ bé).

Chú ý rằng

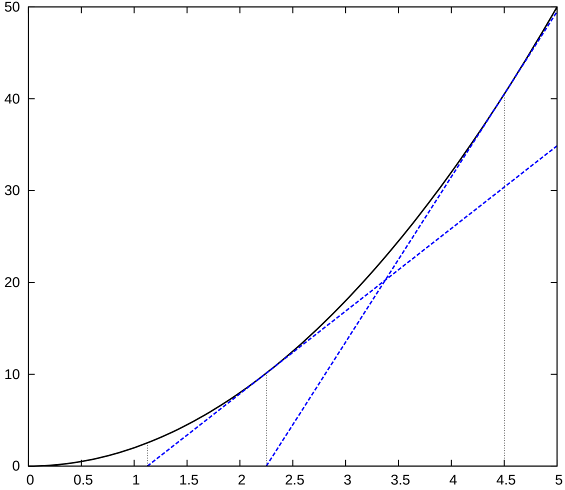
$$f'(\theta^{(n)}) = \frac{\triangle f}{\triangle \theta} = \frac{f(\theta^{(n)}) - 0}{\theta^{(n)} - \theta^{(n+1)}}.$$

Từ đó ta có

$$\theta^{(n+1)} = \theta^{(n)} - \frac{f(\theta^{(n)})}{f'(\theta^{(n)})}.$$

Newton-Raphson Method

Phương pháp Newton tìm nghiệm của hàm $f(\theta) = 0$.



Newton-Raphson Method

- Để cực tiểu hoá $J(\theta)$, ta cần tìm θ sao cho $J'(\theta) = 0$.
- Như vậy, nếu J là hàm khả vi cấp hai thì ta có thể sử dụng phương pháp Newton để tìm θ theo công thức sau

$$\theta := \theta - \frac{J'(\theta)}{J''(\theta)}.$$

Newton-Raphson Method

Trong không gian nhiều chiều, θ là một véc-tơ, phương pháp Newton có công thức tổng quát như sau

$$\theta := \theta - H^{-1}(\nabla J(\theta)),$$

trong đó H là Hessian của J được xác định bởi

$$H_{ij} = \frac{\partial^2 J(\theta)}{\partial\theta_i \partial\theta_j}, \forall i, j = 0, 1, \dots, D.$$

Newton-Raphson Method

Cụ thể là:

- Véc-tơ gradient:

$$\nabla J(\theta) = \begin{pmatrix} \frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)x_{i0} \\ \frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)x_{i1} \\ \vdots \\ \frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)x_{iD} \end{pmatrix}$$

- Ma trận Hessian với kích thước $(D+1) \times (D+1)$:

$$H = \frac{1}{N}\sum_{i=1}^{N} h(\mathbf{x}_i)(1 - h(\mathbf{x}_i))\,\mathbf{x}_i\,\mathbf{x}_i^\top .$$

Newton-Raphson Method

Nếu ta dùng dạng hiệu chỉnh L_2:

- Véc-tơ gradient:

$$\nabla J(\theta) = \begin{pmatrix} \frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)x_{i0} & \\ \frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)x_{i1} & -\lambda\theta_1 \\ \vdots & \\ \frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)x_{iD} & -\lambda\theta_D \end{pmatrix}$$

- Ma trận Hessian với kích thước $(D+1) \times (D+1)$:

$$H = \frac{1}{N}\sum_{i=1}^{N} h(\mathbf{x}_i)(1 - h(\mathbf{x}_i))\,\mathbf{x}_i\,\mathbf{x}_i^\top + \lambda \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \vdots & 0 & 1 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \cdots & 1 \end{pmatrix}.$$

Gradient Descent versus Newton-Raphson

- Trong phương pháp giảm gradient, ta cần chọn tốc độ học α, còn trong phương pháp Newton, ta không cần chọn tham số này.
- Phương pháp Newton thường hội tụ nhanh hơn phương pháp giảm gradient, chỉ cần một số bước lặp ít hơn để đạt được cực trị.
  - Thông thường, nếu số đặc trưng là nhỏ hơn 100 thì phương pháp Newton hội tụ sau khoảng 15 bước lặp.
- Tuy nhiên, mỗi bước lặp của phương pháp Newton lại cần tính toán nhiều hơn nếu số chiều D của ma trận Hessian là lớn.
  - Mỗi bước lặp của phương pháp giảm gradient có độ phức tạp $O(D)$, trong khi mỗi bước lặp của phương pháp Newton có độ phức tạp $O(D^3)$.
  - Khi D lớn (ví dụ, lớn hơn 50,000) thì ta nên dùng phương pháp giảm gradient, còn khi D nhỏ (ví dụ, nhỏ hơn 1,000) thì ta có thể tính được nghịch đảo của ma trận Hessian, do đó nên dùng phương pháp Newton.

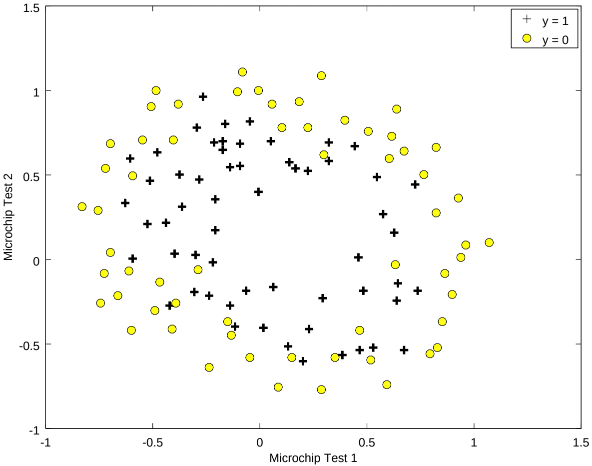
Breast Cancer

- Nếu dùng 10 đặc trưng thì độ chính xác của mô hình trên tập dữ liệu huấn luyện là 94.72%.
- Nếu dùng 30 đặc trưng thì độ chính xác là 100%, không phụ thuộc vào sử dụng hiệu chỉnh hay không.
- Nếu sử dụng hiệu chỉnh L_2 với các tham số hiệu chỉnh $\lambda \in \{10^{-6}, 10^{-3}, 10^{-1}\}$ thì mô hình cũng cho độ chính xác tương tự.

Microchip Quality

- Predict whether microchips from a fabrication plant passes quality assurance (QA).
- During QA, each microchip goes through various tests to ensure it is functioning correctly.
- We have test results for some microchips on two different tests. From these two tests, we would like to determine whether the microchip should be accepted or rejected.

Microchip Quality – Scatter Plot



Microchip Quality – Data

- It is obvious that our dataset cannot be separated into positive and negative examples by a straight-line through the plot.
- Since logistic regression is only able to find *a linear decision boundary*, a straightforward application of logistic regrssion will not perfom well on this dataset.
- Solution? Use feature mapping technique.

Feature Mapping

- One way to fit the data better is to create more features from each data point.
- For each data point $\mathbf{x} = (x_1, x_2)$, we map the features into all polynomial terms of x_1 and x_2 up to the sixth power:

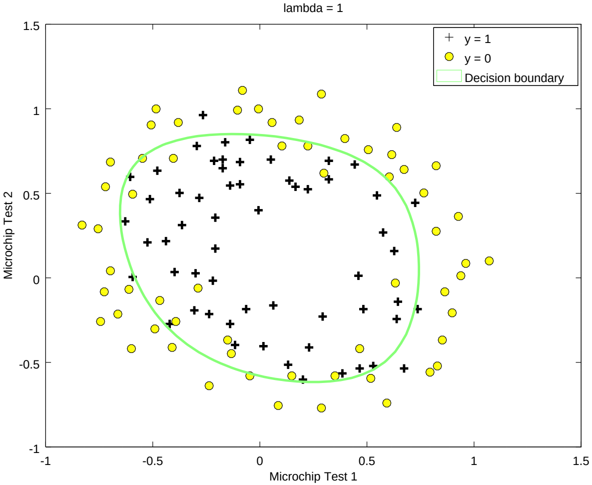
$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \dots \\ x_1 x_2^5 \\ x_2^6 \end{pmatrix}$$

Feature Mapping

- A vector of two features has been transformed to a 28-dimensional vector.
- A logistic regression classifier trained on this higher-dimension feature vector will have a more complex decision boundary and will appear nonlinear when drawn in our 2-dimensional plot.
- Note that while the feature mapping allows us to build a more powerful classifier, it is also more susceptible to overfitting.
- Regularization technique helps us to prevent overfitting problem.

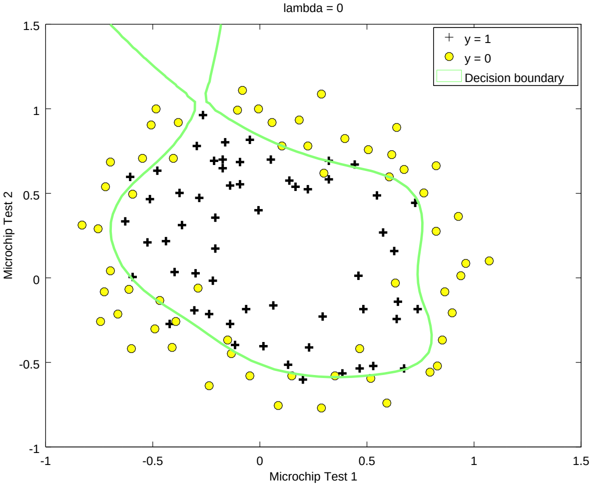
Microchip Quality – Decision Boundary

Training data with decision boundary ($\lambda = 1$)



Microchip Quality – Decision Boundary

No regularization ($\lambda = 0$) – overfitting



Microchip Quality – Decision Boundary

Too much regularization ($\lambda = 100$)