

# Giải Quyết Các Vấn Đề Gặp Khi Sử Dụng Linear Regression

Nhóm 1  
Nhập môn trí tuệ nhân tạo  
Hanoi University of Science and Technology

**Abstract**—Linear Regression là một trong những phương pháp thống kê cơ bản và được sử dụng rộng rãi trong việc dự báo và phân tích dữ liệu. Bài báo cáo này chúng tôi sẽ trình bày về các vấn đề có trong bài toán hồi quy tuyến tính và nêu ra các biện pháp để giải quyết vấn đề đó.

## I. INTRODUCTION

Trong bài báo cáo này, chúng ta sẽ nói về các vấn đề chính gặp phải khi giải bài toán Linear Regression và cách giải quyết chúng.

Để giải bài toán Linear Regression, chúng ta sẽ cần phải giải quyết các vấn đề chính như sau:

- 1) Chọn mô hình chung cho bài toán Linear Regression
- 2) Tìm một hàm để định nghĩa về độ đo hiệu năng  $P$  (cost function)
- 3) Giải cost function để tìm nghiệm  $\theta$ 
  - a) Chúng ta có thể giải cost function theo hai cách:
    - i) Tìm nghiệm chính xác bằng cách sử dụng gradient
    - ii) Tìm nghiệm tương đối bằng cách sử dụng các phương pháp tối ưu hóa (optimizations)
  - b) Tìm nghiệm chính xác và tìm nghiệm bằng các phương pháp tối ưu hóa đều có ưu nhược điểm riêng. Tùy vào từng trường hợp, chúng ta sẽ chọn phương pháp phù hợp. Vấn đề này sẽ được nói chi tiết trong báo cáo.
- 4) Chúng ta cũng cần quan tâm đến vấn đề dung lượng của mô hình. Nếu dung lượng của mô hình quá thấp thì mô hình sẽ không khớp với tập dữ liệu (underfitting), còn nếu dung lượng của mô hình quá cao thì mô hình sẽ khớp quá mức với tập dữ liệu (overfitting), dẫn đến mô hình không khớp với quá trình sinh dữ liệu thực sự (do mô hình có phương sai lớn).
  - Vậy làm sao để chọn được một mô hình có dung lượng tối ưu?. => Chúng ta sẽ giải quyết underfitting và overfitting

## II. CHỌN MÔ HÌNH CHUNG CHO BÀI TOÁN LINEAR REGRESSION

Hồi quy tuyến tính được sử dụng để giải quyết bài toán hồi quy. Nói cách khác, mục đích của nó là xây dựng hệ thống có thể nhận một vector đầu vào  $x \in \mathbb{R}^n$  và dự đoán một giá trị vô hướng  $y \in \mathbb{R}$  ở đầu ra. Gọi  $\hat{y}$  là giá trị giá trị dự đoán mà mô hình  $y$  sẽ nhận. Ta định nghĩa đầu ra theo công thức

$$\hat{y} = \mathbf{w}^T \mathbf{x} + \mathbf{b}$$

Trong đó  $w \in \mathbb{R}^n$  là một vector trọng số,  $b$  là hệ số đánh chặn (intercept)

Trọng số quyết định mức độ ảnh hưởng của mỗi đặc trưng đối với kết quả dự đoán. Nếu trọng số  $w_i$  ứng với đặc trưng(feature) thứ  $i$  là số dương, thì việc tăng giá trị của đặc trưng đó sẽ làm tăng giá trị của kết quả dự đoán  $\hat{y}$ . Và ngược lại nếu một đặc trưng có trọng số âm, thì việc tăng giá trị của đặc trưng đó sẽ làm giảm giá trị của kết quả dự đoán. Trọng số của một đặc trưng có giá trị lớn có nghĩa là đặc trưng đó có mức độ ảnh hưởng lớn đến kết quả dự đoán, còn nếu trọng số của một đặc trưng bằng 0, thì tức là đặc trưng này không ảnh hưởng đến kết quả dự đoán

Tổng kết lại, ta có thể định nghĩa về tác vụ  $T$  của bài toán này như sau: Dự đoán giá trị của  $y$  từ  $x$  bằng công thức  $\hat{y} = \mathbf{w}^T \mathbf{x}$ . Tiếp theo, ta sẽ cần một định nghĩa về độ đo hiệu năng  $P$

## III. ĐỘ ĐO HIỆU NĂNG $P$ (COST FUNCTION)

Xét một tập gồm  $m$  mẫu:

$$X = \{x^{(1)}, \dots, x^{(m)}\} \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(x) \quad (1)$$

trong đó  $p_{\text{data}}(x)$  là một phân phối sinh dữ liệu chưa biết,  $p_{\text{model}}(x; \theta)$  là họ phân phối xác suất mô hình, ánh xạ một mẫu  $x$  bất kỳ đến một số thực là giá trị ước lượng của xác suất thực sự  $p_{\text{data}}(x)$ , và  $\hat{p}_{\text{data}}(x)$  là phân phối xác suất thực nghiệm.

Bây giờ chúng ta tìm  $\theta$  để độ phân kỳ KL giữa phân phối xác suất thực nghiệm  $\hat{p}_{\text{data}}(x)$  và phân phối xác suất mô hình  $p_{\text{model}}(x; \theta)$  là nhỏ nhất:

$$\theta^* = \arg \min_{\theta} D_{KL}(\hat{p}_{\text{data}} \| p_{\text{model}}) \quad (2)$$

$$= \arg \min_{\theta} \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(x) - \log p_{\text{model}}(x)] \quad (3)$$

Do  $\log \hat{p}_{\text{data}}(x)$  không liên quan đến  $\theta$  nên:

$$\Rightarrow \theta^* = \arg \min_{\theta} \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [-\log p_{\text{model}}(x)] \quad (4)$$

$$= \arg \min_{\theta} \sum_{i=1}^m \hat{p}_{\text{data}}(x^{(i)}) [-\log p_{\text{model}}(x^{(i)})] \quad (5)$$

$$= \arg \min_{\theta} \sum_{i=1}^m [-\log p_{\text{model}}(x^{(i)})] \quad (6)$$

Ta biến đổi  $p_{\text{model}}(x^{(i)})$  theo phân phối chuẩn Gauss:

$$p_{\text{model}}(x^{(i)}) = \mathcal{N}(y^{(i)}, \hat{y}^{(i)}, \sigma^2) \quad (7)$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}\right) \quad (8)$$

$$\Rightarrow \log p_{\text{model}}(x^{(i)}) = -\frac{1}{2} \log(2\pi\sigma^2) - \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2} \quad (9)$$

Thay vào phương trình (6), ta được:

$$\theta^* = \arg \min_{\theta} \sum_{i=1}^m \left[ \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2} + \frac{1}{2} \log(2\pi\sigma^2) \right] \quad (10)$$

$$= \arg \min_{\theta} \sum_{i=1}^m \left[ \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2} \right] \quad (11)$$

Để thuận tiện cho việc mở rộng mô hình, ta sẽ chia cho  $m$ , vậy ta sẽ có phương trình cuối cùng như sau:

$$\theta^* = \arg \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2 \quad (12)$$

$$= \arg \min_{\theta} \frac{1}{2m} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 \quad (13)$$

Phương trình  $\arg \min_{\theta} \frac{1}{2m} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$  được gọi là hàm chi phí (cost function) và hàm này còn có một cái tên khác là *trung bình bình phương sai số* (mean squared error (MSE)). Tiếp theo chúng ta sẽ đi giải cost function để tìm nghiệm  $\theta$

#### IV. GIẢI COST FUNCTION ĐỂ TÌM NGHIỆM $\theta$

##### A. Tìm phương trình nghiệm chuẩn (normal equation)

Để tìm nghiệm chuẩn  $\theta$  thì chúng ta sẽ có 2 cách:

- 1) Giải bằng cho gradient = 0
- 2) Giải bằng phương pháp hình chiếu

1) *Giải bằng cách cho gradient = 0*: Theo (13) ta có:

$$\theta^* = \arg \min_{\theta} \frac{1}{2m} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2, \quad (14)$$

$$\Rightarrow \nabla_{\theta} \frac{1}{2m} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = 0, \quad (15)$$

$$\Rightarrow \nabla_{\theta} \frac{1}{2m} \|\mathbf{y} - \mathbf{X}\theta^*\|_2^2 = 0, \quad (16)$$

$$\Rightarrow \nabla_{\theta} \frac{1}{2} (\mathbf{y} - \mathbf{X}\theta^*)^T (\mathbf{y} - \mathbf{X}\theta^*) = 0, \quad (17)$$

$$\Rightarrow \nabla_{\theta} (\theta^{*T} \mathbf{X}^T \mathbf{X} \theta^* - 2\theta^{*T} \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y}) = 0, \quad (18)$$

$$\Rightarrow 2\mathbf{X}^T \mathbf{X} \theta^* - 2\mathbf{X}^T \mathbf{y} = 0, \quad (19)$$

$$\Rightarrow \theta^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (20)$$

**Kết luận:**  $\theta^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$  cũng chính là nghiệm chuẩn mà chúng ta cần tìm

2) *Sử dụng phương pháp hình chiếu để tìm công thức nghiệm chuẩn*: Để chứng minh công thức chuẩn ta còn 1 phương pháp là sử dụng hình chiếu, trước hết ta sẽ bắt đầu với trường hợp đơn giản: có ít mẫu train và ít feature.

Giả sử có 3 mẫu dữ liệu dùng để train  $x^{(1)}, x^{(2)}, x^{(3)}$ , mỗi dữ liệu sẽ chỉ có 1 feature, và đều ra ứng với mỗi mẫu dữ liệu lần lượt là  $y^{(1)}, y^{(2)}, y^{(3)}$ . Khi đó ta có hệ phương trình như sau:

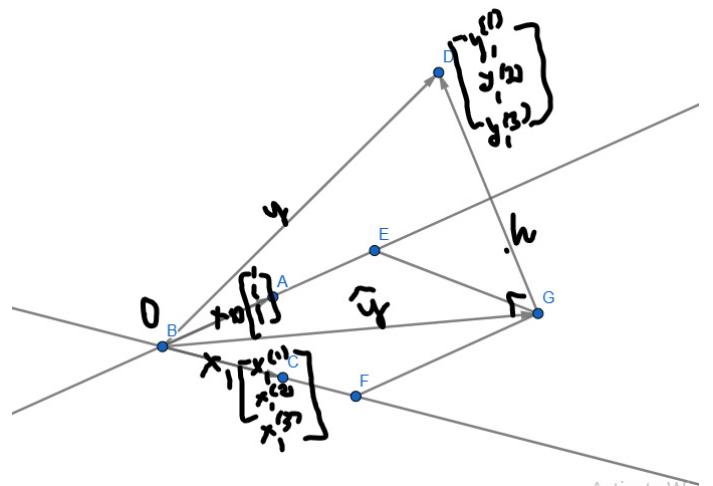
$$y^{(1)} = \theta_0 + x^{(1)}\theta_1 \quad (21)$$

$$y^{(2)} = \theta_0 + x^{(2)}\theta_1 \quad (22)$$

$$y^{(3)} = \theta_0 + x^{(3)}\theta_1 \quad (23)$$

$$\Rightarrow \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix} = \theta_0 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \theta_1 \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix}$$

$$\text{Ta đặt } X = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ 1 & x^{(3)} \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, x_1 = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



Hình 1: Hình ảnh ví dụ về phương pháp hình chiếu trong trường hợp dữ liệu có  $R^{3 \times 1}$  chiều. Hầu hết trong mọi trường hợp thì  $y$  sẽ không thể biểu diễn tuyến tính qua  $x_0$  và  $x_1$  do  $y \notin \text{span}(x_0, x_1)$ . Vì vậy, để khoảng cách giữa  $y$  và  $\hat{y}$  là nhỏ nhất thì  $\hat{y}$  sẽ là hình chiếu của  $y$  lên  $\text{span}(x_0, x_1)$ .

$\hat{y}$  sẽ là hình chiếu của  $y$  lên  $\text{span}(x_0, x_1)$  nên khi đó ta có:

$$h \perp \text{span}(x_0, x_1) \quad (24)$$

$$\Rightarrow h \perp x_0 \text{ and } h \perp x_1 \quad (25)$$

$$\Rightarrow h \perp X^T \quad (26)$$

$$\Rightarrow X^T h = 0 \quad (27)$$

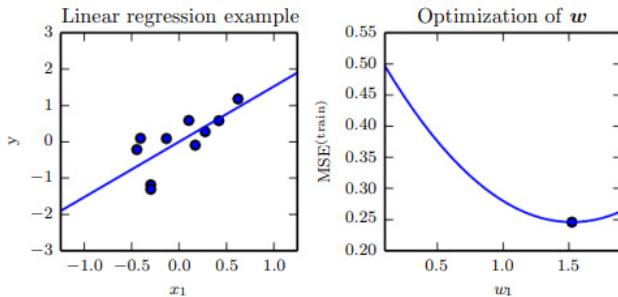
$$\Rightarrow X^T (\hat{y} - y) = 0 \quad (28)$$

$$\Rightarrow X^T (X\theta - y) = 0 \quad (29)$$

$$\Rightarrow X^T X\theta = X^T y \quad (30)$$

$$\Rightarrow \theta = (X^T X)^{-1} X^T y \quad (31)$$

Trong trường hợp ta có số chiều và số mẫu lớn hơn thì chúng ta cũng sẽ chứng minh tương tự như ở trên. Vậy là, chúng ta đã chứng minh công thức chuẩn bằng phương pháp hình chiếu, tiếp theo chúng ta sẽ nói về ưu và nhược điểm của công thức chuẩn.



Hình 2: Ví dụ về một bài toán hồi quy tuyến tính, với tập huấn luyện chứa 10 điểm dữ liệu, mỗi điểm có 1 đặc trưng. Bởi vì điểm dữ liệu chỉ có một đặc trưng, vector trọng số  $w$  chỉ chứa duy nhất một thông số cần học  $w_1$ . (Hình trái) Ta thấy, thuật toán hồi quy tuyến tính cố gắng học  $w_1$  sao cho đường thẳng  $y = w_1 x$  càng gần tất cả các điểm trong tập huấn luyện càng tốt. (Hình phải) Điểm in đậm màu xanh trên hình bên phải là giá trị  $w_1$  tìm được từ nghiệm của phương trình chuẩn, và ta cũng có thể thấy là tại đó, giá trị MSE trên tập huấn luyện cũng là nhỏ nhất.

### 3) *Ưu và nhược điểm của công thức chuẩn:*

#### \* *Ưu điểm:*

- a) *Đơn giản và dễ hiểu:*: Phương pháp này khá trực quan và dễ hiểu, vì chỉ cần một phép tính ma trận nghịch đảo và một phép nhân ma trận.
- b) *Hiệu quả khi số lượng feature nhỏ:*: Với số lượng feature nhỏ, tính toán ma trận nghịch đảo có thể được thực hiện nhanh chóng.

#### \* *Nhược điểm:*

- c) *Độ phức tạp tính toán:*: Tính ma trận nghịch đảo ( $\mathbf{X}^T \mathbf{X}$ ) $^{-1}$  có độ phức tạp tính toán là  $O(n^3)$  với  $n$  là số lượng feature. Khi số lượng feature quá lớn (ví dụ > 10,000), tính toán này trở nên rất tốn kém về mặt thời gian và bộ nhớ.
- d) *Yêu cầu bộ nhớ cao:*: Lưu trữ và xử lý các ma trận lớn đòi hỏi nhiều bộ nhớ, đặc biệt khi số lượng mẫu và số lượng feature đều lớn.

## B. *Tìm nghiệm tương đối bằng cách sử dụng các phương pháp tối ưu hóa (optimizations)*

1) **Gradient Descent:** Giả sử ta có hàm số  $y = f(x)$  với  $x, y$  đều là số thực. Đạo hàm của hàm này được ký hiệu là  $f'(x)$  hay  $\frac{dy}{dx}$ . Đạo hàm  $f'(x)$  cho biết độ dốc của hàm  $f(x)$  tại điểm  $x$ . Nói cách khác, nó cho biết mức độ thay đổi của đầu ra tỉ lệ với một sự thay đổi nhỏ ở đầu vào như thế nào  $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$ .

Do đó, đạo hàm hữu ích trong việc cực tiểu hóa một hàm số vì nó cho ta biết nên thay đổi  $x$  như thế nào để cải thiện hàm mục tiêu một lượng nhỏ. Ví dụ, ta biết rằng với  $\epsilon$  đủ nhỏ thì  $f(x - \epsilon \text{sign}(f'(x)))$  sẽ nhỏ hơn  $f(x)$ . Từ quan sát này, ta có thể giảm giá trị hàm  $f(x)$  bằng cách tăng giảm  $x$  một lượng nhỏ về hướng ngược với hướng đạo hàm. Kĩ thuật này được gọi là *trượt gradient* (gradient descent).

Trượt gradient để xuất đi tới điểm mới:

$$x \leftarrow x - \epsilon f'(x)$$

trong đó  $\epsilon$  là **tốc độ học** (learning rate), một đại lượng vô hướng có giá trị dương xác định độ lớn của bước nhảy. Chúng ta có thể chọn  $\epsilon$  theo nhiều cách khác nhau. Một cách phổ biến là cho  $\epsilon$  nhận giá trị là một hằng số nhỏ. Đôi khi, ta có thể tính được bước nhảy tối ưu để khiến đạo hàm triệt tiêu. Một cách tiếp cận khác là tính  $f(x - \epsilon \nabla_x f(x))$  với một số giá trị của  $\epsilon$  và lựa chọn giá trị dẫn tới kết quả nhỏ nhất của hàm mục tiêu. Chiến lược này còn được gọi là *dò đường* (line search). Ngoài ra chúng ta cũng có thể tìm được  $\epsilon$  tối ưu tại mỗi điểm bằng cách dựa vào đạo hàm bậc 1 và ma trận Hesse.

Thuật toán trượt gradient hội tụ khi mọi phần tử của gradient bằng 0 (hoặc, trong thực tế là gần 0). Trong một số trường hợp, ta không cần cập nhật từng vòng lặp mà có thể nhảy trực tiếp đến điểm tối hạn bằng cách giải phương trình  $\nabla_x f(x) = 0$ .

#### **Mã giả cho thuật toán Gradient Descent:**

```
def gradient_descent(J, compute_gradient,
theta_initial, alpha, num_iterations):
    # Khởi tạo theta ban đầu
    theta = theta_initial

    # Lặp qua số lần lặp
    for i in range(num_iterations):
        # Tính gradient theo theta
        gradient = compute_gradient(J, theta)

        # Cập nhật các tham số theta
        theta = theta - alpha * gradient

        # Tính toán hàm chi phí
        cost = J(theta)

    # Trả về các tham số đã tối ưu hóa
    return theta
```

#### a) *Ưu và nhược điểm của gradient descent:*

#### \* *Ưu điểm:*

- . Hiệu quả với dữ liệu lớn.
- . Thích hợp cho việc triển khai song song và phân tán.

#### \* *Nhược điểm:*

- . Cần phải chọn siêu tham số learning rate và có thể mất nhiều thời gian để hội tụ.
- . Cần chuẩn hóa dữ liệu để đảm bảo tốc độ hội tụ.

**if**  $i = m$ : Chúng ta sẽ shuffle dữ liệu

$$\theta \leftarrow \theta - \epsilon(x^{(i)2}\theta - x^{(i)}y^{(i)})$$

Cập nhật:  $i \leftarrow i + 1$   
**end while**

---

### Mã Giả cho SGD

---

```
def sgrad(w, i, rd_id):
    true_i = rd_id[i]
    xi = Xbar[true_i, :]
    yi = y[true_i]
    a = np.dot(xi, w) - yi
    return (xi*a).reshape(2, 1)

def SGD(w_init, grad, eta):
    w = [w_init]
    w_last_check = w_init
    iter_check_w = 10
    N = X.shape[0]
    count = 0
    for it in range(10):
        # shuffle data
        rd_id = np.random.permutation(N)
        for i in range(N):
            count += 1
            g = sgrad(w[-1], i, rd_id)
            w_new = w[-1] - eta*g
            w.append(w_new)
        if count%iter_check_w == 0:
            w_this_check = w_new
            if np.linalg.norm(w_this_check - w_last_check)/len(w_init) < 1e-3:
                return w
    w_last_check = w_this_check
return w
```

---

Hình 3: Hoạt ảnh thể hiện quá trình cập nhật  $\theta$  bằng Gradient Descent

2) **Stochastic gradient descent:** Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu  $x_i$  rồi cập nhật  $\theta$  dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán rất đơn giản này trên thực tế lại làm việc rất hiệu quả.

Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật  $\theta$ , với SGD thì mỗi epoch ứng với  $N$  lần cập nhật  $\theta$  với  $N$  là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn (chủ yếu là Deep Learning mà chúng ta sẽ thấy trong phần sau của blog) và các bài toán yêu cầu mô hình thay đổi liên tục, tức online learning.

**Thứ tự lựa chọn điểm dữ liệu:** Một điểm cần lưu ý đó là: sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu năng của SGD.

Một cách toán học, quy tắc cập nhật của SGD là:

$$\theta \leftarrow \theta - \epsilon \nabla_{\theta} J(\theta; x_i; y_i)$$

trong đó  $J(\theta; x_i; y_i)$  là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là  $(x_i, y_i)$ .

**Chú ý:** chúng ta hoàn toàn có thể áp dụng các thuật toán tăng tốc GD như Momentum, AdaGrad,... vào SGD.

### Áp dụng vào linear regression

---

**Giải thuật SGD:** Thuật toán cực tiểu hoá hàm chi phí bằng phương pháp trượt gradient ngẫu nhiên, bắt đầu từ một giá trị  $\theta$  tùy ý.

**Require:** Tham số ban đầu  $\theta$

**Require:** Vùng hội tụ  $\delta$

**Require:** Số mẫu train  $m$

$i \leftarrow 1$

**while**  $\|x^{(i)2}\theta - x^{(i)}y^{(i)}\| > \delta$  **do**

Hình 4: Hoạt ảnh thể hiện quá trình cập nhật  $\theta$  bằng SGD

Hình bên trái mô tả đường đi của nghiệm. Chúng ta thấy rằng đường đi khá là zigzag chứ không mượt như khi sử dụng GD. Điều này là dễ hiểu vì một điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu được. Tuy nhiên, chúng ta cũng thấy rằng thuật toán hội tụ khá nhanh đến vùng lân cận của nghiệm. Với 1000 điểm dữ liệu, SGD chỉ cần gần 3 epoches (3161 tương ứng với 3161 lần cập nhật, mỗi lần lấy 1 điểm). Nếu so với con số 49 vòng lặp (epoches) như kết quả tốt nhất có được bằng GD, thì kết quả này lợi hơn rất nhiều.

**3) Minibatch gradient descent:** Khác với SGD, mini-batch sử dụng một số lượng  $n$  lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu  $N$  rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các mini-batch, mỗi mini-batch có  $n$  điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu  $N$  không chia hết cho  $n$ ). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; \mathbf{x}_{i:i+n}, \mathbf{y}_{i:i+n})$$

Với  $\mathbf{x}_{i:i+n}$  được hiểu là dữ liệu từ thứ  $i$  tới thứ  $i+n-1$  (theo ký hiệu của Python). Dữ liệu này sau mỗi epoch là khác nhau vì chúng cần được xáo trộn. Một lần nữa, các thuật toán khác cho GD như Momentum, Adagrad, Adadelta,... cũng có thể được áp dụng vào đây.

Mini-batch GD được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong Deep Learning. Giá trị  $n$  thường được chọn là khoảng từ 50 đến 100.

Dưới đây là ví dụ về giá trị của hàm mất mát mỗi khi cập nhật tham số  $\theta$  của một bài toán khác phức tạp hơn.

**a) Ưu và nhược điểm của Minibatch :**

\* Ưu điểm:

- . Giảm thiểu dao động so với SGD và nhanh hơn so với Gradient Descent.
- . Có thể tận dụng tối đa khả năng tính toán song song của phần cứng hiện đại.

\* Nhược điểm:

- . Cần chọn kích thước mini-batch và learning rate phù hợp.
- . Cần tài nguyên bộ nhớ đủ lớn để xử lý từng mini-batch.

**b) Ưu và nhược điểm của SGD:**

\* Ưu điểm:

- . Cập nhật trọng số sau mỗi mẫu, do đó có thể xử lý dữ liệu trực tuyến (online learning).
- . Tiết kiệm bộ nhớ hơn so với gradient descent thông thường.

\* Nhược điểm:

- . Hội tụ có thể không ổn định và có thể dao động xung quanh điểm tối ưu.
- . Cần điều chỉnh learning rate thích hợp.

**4) Newton's Method:** Các phương pháp như GD, SGD, minibatch còn được gọi là first-order method tức là cập nhật dựa trên đạo hàm bậc 1, ngoài ra còn 1 phương pháp là Second-Order Method, cập nhật dựa trên đạo hàm bậc 2, Tiêu biểu chính là phương pháp newton's method.

Phương pháp Newton được phát triển dựa trên việc khai thác thông tin từ ma trận Hesse. Phương pháp Newton dựa trên khai triển chuỗi Taylor bậc hai để xấp xỉ  $f(x)$  gần một điểm  $x^{(0)}$ :

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^T \nabla_x f(x^{(0)}) + \frac{1}{2}(x - x^{(0)})^T H(f)(x^{(0)})(x - x^{(0)}) \quad (32)$$

$$\Rightarrow \nabla_x f(x) = \nabla_x f(x^{(0)}) + H(f)(x^{(0)})(x - x^{(0)}) \quad (33)$$

$$\nabla_x f(x) = 0 \Leftrightarrow x = x^{(0)} - H(f)(x^{(0)})^{-1} \nabla_x f(x^{(0)}) \quad (34)$$

Nếu  $f$  là hàm bậc hai xác định dương, thì ta sử dụng phương pháp Newton bằng cách áp dụng phương trình (33) một lần và có thể nhảy đến ngay điểm cực tiểu của hàm số. Nếu hàm  $f$  không phải là bậc hai nhưng có thể xấp xỉ cục bộ bởi một hàm bậc hai xác định dương, thì ta thực hiện phương pháp Newton bằng cách áp dụng phương trình (33) nhiều lần. Lặp lại việc cập nhật hàm xấp xỉ và nhảy đến cực tiểu của hàm xấp xỉ có thể đưa ta đến điểm tối hạn nhanh hơn khá nhiều so với thuật toán trượt gradient. Đây là một tính chất hữu ích khi ta đang ở gần một điểm cực tiểu cục bộ, nhưng cũng có thể là một tính chất có hại nếu ta ở gần điểm yên ngựa.(Bởi xấp xỉ bậc 2 những điểm yên ngựa sẽ cho ra 1 đường thẳng có giá trị tại tất cả mọi điểm bằng nhau nên ta sẽ bị dừng tại điểm yên ngựa chứ không thể tiếp tục tiến về điểm cực tiểu)

Hình 5: Hoạt ảnh thể hiện quá trình cập nhật  $\theta$  bằng Newton Method

Khi áp dụng phương pháp Newton vào bài toán Hồi quy tuyến tính, ta sẽ hội tụ sau một lần lặp duy nhất. Lý do là vì phương pháp Newton dựa trên khai triển Taylor bậc hai của hàm mất mát, và hàm mất mát MSE (Mean Squared Error) trong Hồi quy tuyến tính là một hàm bậc hai. Do đó, việc xấp xỉ bậc hai chính là hàm mất mát thực sự.

Khi áp dụng phương pháp Newton để tìm nghiệm, thực chất chúng ta đang tìm nghiệm dựa trên việc gradient của hàm mất mát bằng 0. Tuy nhiên, phương pháp Newton sử dụng gradient và ma trận Hessian (ma trận của các đạo hàm bậc hai) của hàm mất mát MSE, trong khi công thức chuẩn sử dụng gradient của MSE.

Vì thế, phương pháp Newton sẽ hội tụ ngay lập tức chỉ sau một lần lặp.

#### Lợi ích:

- Hội tụ rất nhanh (theo cấp số nhân) khi gần điểm tối ưu.
- Thích hợp cho bài toán với số lượng đặc trưng nhỏ.

#### Hạn chế:

- Tính toán ma trận Hessian và nghịch đảo của nó rất tốn kém (độ phức tạp  $O(n^3)$ ).
- Không phù hợp với dữ liệu lớn hoặc khi số lượng đặc trưng rất lớn.

### C. Khi nào nên chọn thuật toán nào để có thể tìm nghiệm θ 1 cách tối ưu nhất ?

#### 1) Công thức chuẩn (Normal Equation):

##### 2) Khi nào nên chọn::

- Khi dữ liệu có kích thước nhỏ hoặc trung bình (số lượng mẫu  $m$  và số lượng đặc trưng  $n$  không quá lớn).
- Khi số lượng đặc trưng  $n < 10,000$  và số lượng mẫu  $m$  không quá lớn.
- Khi bạn cần một phương pháp đơn giản và nhanh chóng để giải bài toán hồi quy tuyến tính.
- Khi bạn có đủ bộ nhớ để lưu trữ ma trận trọng số và thực hiện các phép toán ma trận.

#### 3) Gradient Descent:

##### 4) Khi nào nên chọn::

- Khi dữ liệu có kích thước lớn và không thể tính toán được ma trận nghịch đảo (sử dụng công thức chuẩn).
- Khi số lượng đặc trưng  $n \geq 10,000$  hoặc số lượng mẫu  $m \geq 100,000$ .
- Khi bạn có thể chấp nhận việc tối ưu hóa qua nhiều lần lặp lại (iterations).

#### 5) Stochastic Gradient Descent (SGD): Khi nào nên chọn:

- Khi bạn có dữ liệu rất lớn và không thể xử lý toàn bộ dữ liệu trong một lần (batch).
- Khi số lượng mẫu  $m$  rất lớn, có thể lên đến hàng triệu hoặc nhiều hơn.
- Khi bạn muốn tối ưu hóa nhanh chóng với việc cập nhật trọng số thường xuyên hơn.

#### 6) Mini-batch Gradient Descent: Khi nào nên chọn:

- Khi bạn muốn có sự cân bằng giữa Gradient Descent và SGD.

- Khi bạn có dữ liệu lớn nhưng vẫn muốn cập nhật trọng số thường xuyên hơn so với Gradient Descent.
- Khi số lượng mẫu  $m$  lớn, từ vài chục nghìn đến hàng triệu.

#### 7) Newton's Method: Khi nào nên chọn:

- Khi bạn cần tốc độ hội tụ nhanh hơn so với Gradient Descent.
- Khi số lượng đặc trưng  $n < 1,000$  và số lượng mẫu  $m$  không quá lớn.
- Khi bạn có dữ liệu không quá lớn và có thể tính toán được ma trận Hessian (ma trận đạo hàm bậc hai của hàm mục tiêu).

#### 8) Tóm tắt:

- Công thức chuẩn:** Khi số lượng đặc trưng  $n < 10,000$  và số lượng mẫu  $m$  không quá lớn.
- Gradient Descent:** Khi số lượng đặc trưng  $n \geq 10,000$  hoặc số lượng mẫu  $m \geq 100,000$ .
- SGD:** Khi số lượng mẫu  $m$  rất lớn, có thể lên đến hàng triệu hoặc nhiều hơn.
- Mini-batch Gradient Descent:** Khi số lượng mẫu  $m$  lớn, từ vài chục nghìn đến hàng triệu.
- Newton's Method:** Khi số lượng đặc trưng  $n < 1,000$  và số lượng mẫu  $m$  không quá lớn.

Chọn phương pháp tối ưu hóa phù hợp sẽ giúp bạn giải quyết bài toán hồi quy tuyến tính một cách hiệu quả và chính xác.

## V. CHỌN MÔ HÌNH CÓ DUNG LƯỢNG TỐI UU

### A. Underfitting, Overfitting, Capacity

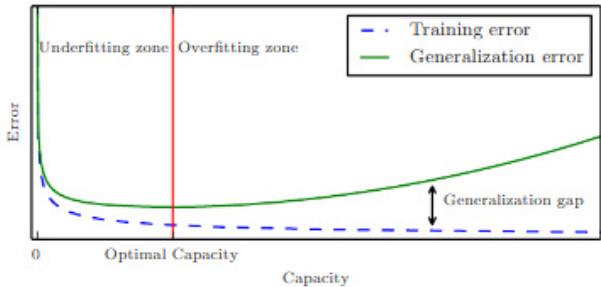
Các yếu tố để đánh giá một thuật toán học máy có tốt hay không là khả năng của nó khi cố gắng làm cho:

1) Sai số huấn luyện nhỏ đi

2) Khoảng cách giữa sai số huấn luyện và sai số kiểm thử nhỏ đi

Hai yếu tố trên tương ứng với hai thách thức chính trong học máy: vị khớp (underfitting) và quá khớp (overfitting). Vị khớp xảy ra khi mô hình không thể đạt được một sai số đủ nhỏ trên tập huấn luyện. Quá khớp xảy ra khi khoảng cách giữa sai số huấn luyện và sai số kiểm thử là quá lớn.

Chúng ta có thể kiểm soát việc một mô hình sẽ có nhiều khả năng quá khớp hay vị khớp hơn bằng cách thay đổi dung lượng (capacity) của nó. Nói nôm na, dung lượng của mô hình là khả năng khớp của mô hình đó với nhiều loại hàm số khác nhau. Các mô hình có dung lượng thấp có thể gặp khó khăn để khớp với tập huấn luyện. Ngược lại, mô hình với dung lượng cao có thể gặp tình trạng quá khớp khi ghi nhớ một cách máy móc thuộc tính của tập huấn luyện dẫn đến mô hình không có khả năng tổng quát trên tập kiểm thử.



Hình 6: Mối quan hệ điển hình giữa dung lượng mô hình và sai số. Sai số huấn luyện và sai số kiểm thử có xu hướng khác nhau. Ở bên trái của đồ thị, sai số huấn luyện và sai số khái quát hoá đều cao. Nơi này là vùng vị khớp. Khi tăng dung lượng của một mô hình lên, sai số huấn luyện giảm dần, nhưng khoảng cách giữa sai số huấn luyện và sai số khái quát hoá lại tăng lên. Khi khoảng cách tăng nhanh hơn tốc độ giảm của sai số huấn luyện, chúng ta sẽ rơi vào vùng quá khớp - nơi mà dung lượng mô hình quá lớn, vượt quá dung lượng tối ưu.

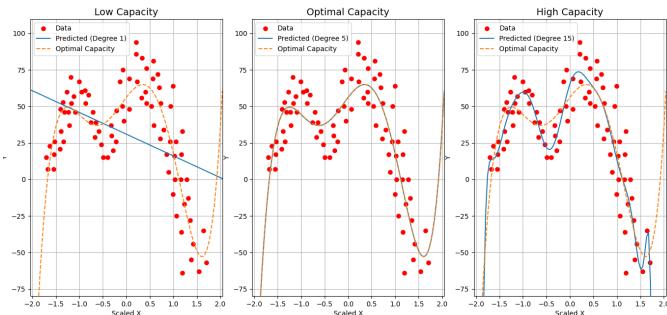
$$\hat{y} = b + \sum_{i=1}^n w_i x^i$$

Trong đó:

- $\hat{y}$  là giá trị dự đoán của mô hình.
- $b$  là hệ số tự do (intercept).
- $w_i$  là trọng số tương ứng với feature  $x^i$ .
- $x^i$  là feature là lũy thừa bậc  $i$  của  $x$ .

Bằng cách thêm vào các feature  $x^2, x^3, \dots, x^n$ , chúng ta mở rộng không gian giả thuyết của mô hình, cho phép mô hình nắm bắt được các quan hệ phi tuyến tính giữa  $x$  và  $y$ , từ đó giảm thiểu hiện tượng underfitting.

Trước tiên, chúng ta sẽ quan sát: Khi tăng dung lượng của mô hình lên thì mô hình sẽ thay đổi như thế nào



Hình 7: Hình bên trái thể hiện khi dung lượng của mô hình quá nhỏ ( $= 1$ ) so với dung lượng tối ưu ( $5$ ), khi đó mô hình sẽ có sai số huấn luyện và sai số kiểm thử đều lớn  $\Rightarrow$  underfitting. Hình bên phải thể hiện khi dung lượng của mô hình quá lớn ( $= 20$ ) so với dung lượng tối ưu thì khi đó mô hình sẽ có sai số huấn luyện nhỏ nhưng sai số kiểm thử lại lớn  $\Rightarrow$  overfitting. Hình ở giữa là hình có dung lượng tối ưu, ta có thể thấy khi đó mô hình sẽ khớp với cả tập huấn luyện và tập kiểm thử  $\Rightarrow$  sai số huấn luyện và sai số kiểm thử đều đạt mức OK.

### B. Giải quyết vấn đề underfitting

Để giải quyết vấn đề underfitting, chúng ta cần tăng dung lượng của mô hình bằng cách thêm vào nhiều feature hơn hoặc tăng độ phức tạp của mô hình. Phương pháp để kiểm soát dung lượng của một thuật toán học tập là lựa chọn không gian giả thuyết (hypothesis space) của nó, tức là tập hợp các hàm số mà thuật toán học tập được phép chọn làm giải pháp.

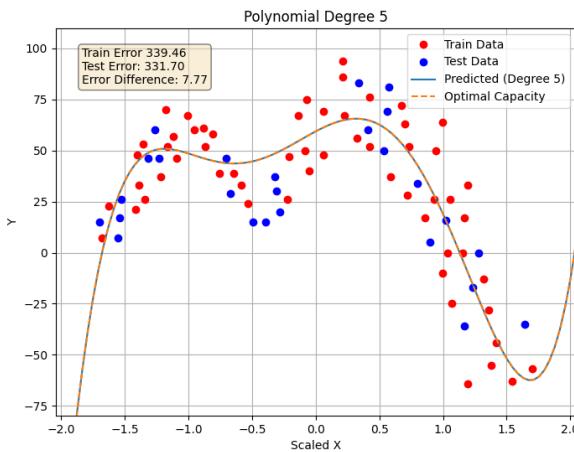
Trong hồi quy tuyến tính, chúng ta có thể kiểm soát dung lượng của một mô hình bằng cách thêm các feature là lũy thừa bậc cao của  $x$ . Công thức tổng quát của một mô hình hồi quy đa thức có thể được viết như sau:

Hình 8: Hình ảnh thể hiện sự thay đổi của mô hình khi dung lượng tăng dần đến 10

Từ hoạt ảnh trên chúng ta có thể thấy khi dung lượng của mô hình = 5, thì khoảng cách giữa sai số huấn luyện và sai số kiểm thử là nhỏ nhất ( $= 7.77$ ), và mô hình này cũng có sai số huấn luyện khá nhỏ ( $= 339.46$ ). Từ đó chúng ta có thể lựa chọn được mô hình là:

$$\hat{y} = b + \sum_{i=1}^5 w_i x^i$$

tổng quát hoá lên, và tránh được overfitting



Hình 9: Khi dung lượng mô hình = 5 = dung lượng tối ưu

### C. Giải quyết vấn đề overfitting

Để giải quyết được vấn đề overfitting, chúng ta sẽ cần phải suy giảm dung lượng của mô hình về gần dung lượng tối ưu nhất.

Vậy làm sao để suy giảm dung lượng?

⇒ Câu trả lời chính là chúng ta suy giảm hoặc cắt bỏ feature.

Chúng ta có rất nhiều feature, vậy chúng ta nên suy giảm hoặc cắt bỏ feature nào?

⇒ Câu trả lời chính là chúng ta sẽ chọn những feature có sức ảnh hưởng đối với đầu ra nhỏ, sau đó chúng ta sẽ suy giảm sức ảnh hưởng (trọng số) của feature đó.

Vậy làm sao để chúng ta chọn được đâu là feature có sức ảnh hưởng nhỏ?

⇒ Dựa trên ý tưởng suy giảm những feature có sức ảnh hưởng nhỏ để tránh overfitting, các nhà khoa học đã đưa ra một phương pháp tương ứng chính là L2 Parameter Regularization. Còn dựa trên ý tưởng cắt giảm feature thì chúng ta sẽ có L1 Regularization, điều này cũng tạo ra tính riêng biệt của L1 so với L2

Ngoài trừ cắt bỏ/ suy giảm feature có sức ảnh hưởng yếu thì chúng ta có còn phương pháp nào để tránh overfitting hay không?

⇒ Câu trả lời chính là có, Có 1 phương pháp đó chính là trong quá trình cập nhật  $\theta$ , thì thay vì chúng ta sẽ chọn 1 bộ parameter làm cho sai số huấn luyện nhỏ nhất thì chúng ta sẽ chọn 1 bộ parameter làm cho khoảng cách giữa sai số huấn luyện và sai số kiểm thử là nhỏ nhất. Phương pháp này còn được gọi là **kết thúc sớm**. Ngoài ra chúng ta cũng còn 1 phương pháp khác đó chính là bagging, thay vì chúng ta dùng 1 mô hình để đánh giá kết quả thì chúng ta sẽ dùng nhiều mô hình để đánh giá kết quả, điều này sẽ làm tăng tính

Tiếp theo chúng ta sẽ đi nói chi tiết hơn và  $L1$  và  $L2$

#### 1) Cơ chế phạt chuẩn $L^2$ ( $L^2$ Parameter Regularization):

Phạt chuẩn  $L^2$ , hay *suy giảm trọng số* (weight decay), là một chiến lược giúp giảm tác động của các feature có đóng góp nhỏ bằng cách điều chỉnh trọng số của chúng tiến gần về 0. Ý tưởng chính là các feature có hiệp phương sai thấp với đầu ra thường liên quan đến các hướng mà hàm mục tiêu ít ưu tiên. Về mặt toán học, những feature này tương ứng với các hướng mà ma trận Hessian của hàm mục tiêu có giá trị nhỏ. Để kiểm soát điều này, hàm mục tiêu được bổ sung thêm hạng tử  $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$ , giúp điều chỉnh các trọng số  $w$  về gần gốc tọa độ, làm tăng bias và giảm variance, từ đó cải thiện tính tổng quát hóa của mô hình. Phạt chuẩn  $L^2$  còn được gọi là *hồi quy ngọn sóng* (ridge regression) hay *kiểm soát Tikhonov*.

Trong hồi quy tuyến tính, mục tiêu là tìm các trọng số  $w$  tối ưu bằng cách giảm thiểu hàm mất mát, bao gồm cả hạng tử phạt chuẩn  $L^2$ :

$$J(w) = \frac{1}{2m} \sum_{i=1}^m (y_i - \hat{y}_i)^2 + \frac{\lambda}{2} \|w\|_2^2$$

Trong đó:

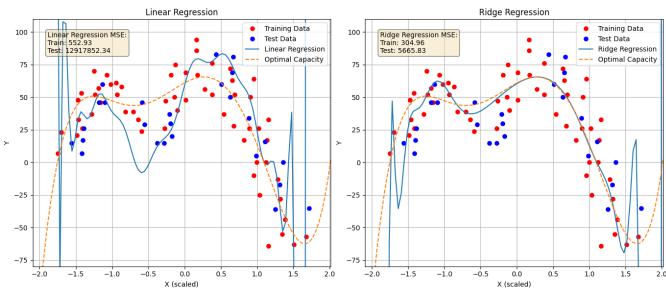
- $m$  là số lượng mẫu huấn luyện,
- $y_i$  là giá trị thực tế,
- $\hat{y}_i$  là giá trị dự đoán,
- $\lambda$  là hệ số điều chỉnh cho hạng tử phạt chuẩn  $L^2$ .

Cập nhật trọng số được thực hiện thông qua gradient descent:

$$w := w - \alpha \left( \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)x_i + \lambda w \right)$$

Trong đó: -  $\alpha$  là tốc độ học (learning rate), -  $x_i$  là vector feature của mẫu thứ  $i$ .

Với cách tiếp cận này, các trọng số của feature có đóng góp nhỏ sẽ được điều chỉnh về gần 0, giảm thiểu overfitting và tăng khả năng tổng quát hóa của mô hình.



Hình 10: Hình bên trái cho thấy khi chưa sử dụng  $L_2$  thì mô hình sẽ có sai số huấn luyện nhỏ, nhưng sai số kiểm thử lại vô cùng lớn,  $\Rightarrow$  overfitting. Còn hình bên phải cho chúng ta thấy sau khi sử dụng  $L_2$  thì sai số huấn luyện có tăng lên 1 ít nhưng sai số kiểm thử lại giảm rất nhiều so với khi chưa dùng  $L_2$ , điều này đã khiến cho mô hình tránh được overfitting, và làm tăng tính tổng quát hóa của mô hình

2) **Cơ chế kiểm soát  $L^1$  ( $L^1$  Regularization):** Ngoài  $L^2$  là dạng suy giảm trọng số phổ biến nhất, còn có nhiều giải pháp khác để áp đặt mức phát độ lớn của tham số mô hình. Một trong số đó là cơ chế kiểm soát  $L^1$ . Cơ chế kiểm soát  $L^1$  có ý tưởng gần như tương tự  $L^2$  là giảm trọng số của các feature có sự đóng góp không đáng kể để giảm dung lượng mô hình. Tuy nhiên  $L^1$  khác  $L^2$  ở chỗ  $L^1$  sẽ trực tiếp cắt bỏ những feature có sự đóng góp nhỏ thay vì suy giảm sức ảnh hưởng của những feature đó gần về 0 như  $L^2$ , điều này cũng khiến cho dữ liệu đầu vào được biểu diễn thừa (Sparse Representations). Chính việc khiến dữ liệu được biểu diễn thừa của  $L^1$  khiến cho mô hình được tổng quát hoá hơn.

Công thức toán học cho LASSO được định nghĩa như sau:

$$\min_{\theta} \left\{ \frac{1}{2n} \sum_{i=1}^n (y_i - \theta_0 - \sum_{j=1}^p \theta_j x_{ij})^2 + \lambda \sum_{j=1}^p |\theta_j| \right\}$$

Trong đó:

- $y_i$  là giá trị thực tế của điểm dữ liệu thứ  $i$ ,
- $x_{ij}$  là giá trị của feature thứ  $j$  tại điểm dữ liệu thứ  $i$ ,
- $\theta_0$  là hằng số (intercept),
- $\theta_j$  là hệ số của feature thứ  $j$ ,
- $n$  là số lượng điểm dữ liệu,
- $p$  là số lượng features,
- $\lambda$  là tham số điều chỉnh mức độ suy giảm (regularization parameter).

Tham số  $\lambda$  đóng vai trò rất quan trọng trong việc điều chỉnh mức độ phạt mà ta muốn áp dụng lên các trọng số  $\theta_j$ . Khi  $\lambda$  tăng, số lượng features có trọng số bằng 0 cũng sẽ tăng, dẫn đến mô hình đơn giản hơn nhưng có thể mất đi một số thông tin quan trọng. Ngược lại, khi  $\lambda$  giảm, mô hình sẽ trở nên phức tạp hơn nhưng có khả năng khớp tốt hơn với dữ liệu huấn luyện.

Thuật toán Gradient Descent để tìm nghiệm cho bài toán LASSO có thể được thực hiện như sau:

#### Gradient Descent cho LASSO:

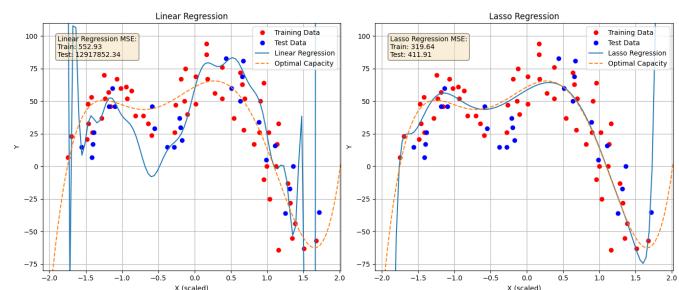
- Khởi tạo các trọng số  $\theta_j$  với giá trị ban đầu (thường là 0).

2) Cập nhật các trọng số  $\theta_j$  dựa trên gradient của hàm lỗi kết hợp với ràng buộc  $L^1$ :

$$\theta_j \leftarrow \theta_j - \eta \left( \frac{\partial}{\partial \theta_j} \left\{ \frac{1}{2n} \sum_{i=1}^n (y_i - \theta_0 - \sum_{j=1}^p \theta_j x_{ij})^2 \right\} + \lambda \cdot \text{sign}(\theta_j) \right)$$

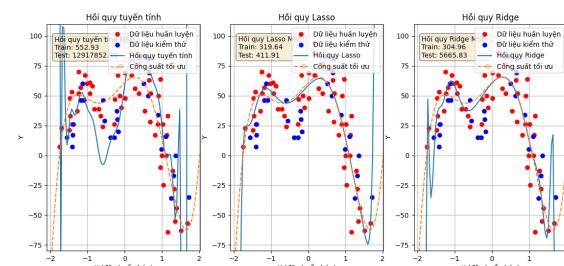
Trong đó  $\eta$  là tốc độ học (learning rate).

Với thuật toán này, ta có thể tìm ra được các trọng số tối ưu cho mô hình hồi quy tuyến tính có kiểm soát  $L^1$ , giúp cải thiện tính tổng quát của mô hình và giảm bớt số lượng features không cần thiết.

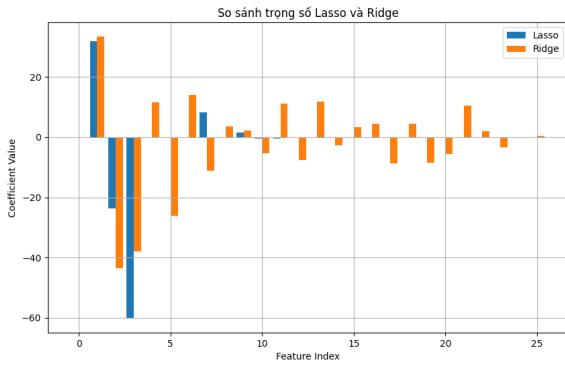


Hình 11: Minh họa cơ chế kiểm soát  $L^1$

3)  $L_1$  khác  $L_2$  thế nào, và khi nào nên dùng  $L_1$ , khi nào nên dùng  $L_2$ : Trước tiên chúng ta sẽ quan sát 2 biểu đồ, so sánh giữa  $L_1$  và  $L_2$



Hình 12: trái: Linear Regression, giữa: Linear Regression với kiểm soát  $L_1$ , phải: Linear Regression với kiểm soát  $L_2$



Hình 13: Hình ảnh thể trọng số của các feature sau khi sử dụng kiểm soát  $L_1$  và  $L_2$ , trọng số ( $L_1$ ): đường màu cam, trọng số ( $L_2$ ): đường màu xanh

Từ hình 12 ta có thể thấy được sau khi sử dụng  $L_1$  thì sẽ có vài feature có trọng số = 0, còn sử dụng  $L_2$  thì sẽ chỉ suy giảm trọng số của những feature đó về gần 0 chứ không cắt bỏ hoàn toàn, điều này chính là tính thua của  $L_1$

#### Khi nào nên chọn $L_1$ , khi nào nên chọn $L_2$ ?

Trong học máy, việc chọn  $L_1$  (Lasso) hay  $L_2$  (Ridge) phụ thuộc vào đặc điểm của dữ liệu và mục tiêu của mô hình. Dưới đây là một số gợi ý để giúp bạn quyết định khi nào nên chọn  $L_1$  và khi nào nên chọn  $L_2$ :

Tính huống	Nên chọn
Cần loại bỏ bớt các đặc trưng không quan trọng	$L_1$
Các đặc trưng có tương quan cao	$L_1$
Cần mô hình có khả năng giải thích cao	$L_1$
Cần ổn định hơn với ít sự thay đổi nhỏ trong dữ liệu	$L_2$
Các đặc trưng có độ quan trọng gần giống nhau	$L_2$
Số lượng đặc trưng lớn, cần giảm bớt số lượng đặc trưng	$L_1$
Mục tiêu là giảm thiểu lỗi dự đoán (predictive accuracy)	$L_2$

Bảng I: Khi nào nên chọn  $L_1$  và khi nào nên chọn  $L_2$

#### L1 (Lasso) Regularization:

$$\lambda \sum_{i=1}^n |w_i|$$

- Ưu điểm: Có khả năng đưa một số trọng số  $w_i$  về 0, do đó giúp chọn lọc đặc trưng (feature selection).
- Nhược điểm: Đôi khi không ổn định khi các đặc trưng có tương quan cao.

#### L2 (Ridge) Regularization:

$$\lambda \sum_{i=1}^n w_i^2$$

- Ưu điểm: Không đưa trọng số về 0, do đó tất cả các đặc trưng đều được giữ lại. Thường làm việc tốt hơn khi các đặc trưng có tương quan cao.
- Nhược điểm: Không thực hiện việc chọn lọc đặc trưng.

Tóm lại, nếu bạn muốn giảm số lượng đặc trưng và loại bỏ những đặc trưng không quan trọng, hãy chọn  $L_1$ . Nếu bạn cần một mô hình ổn định và không quá lo lắng về việc có quá nhiều đặc trưng, hãy chọn  $L_2$ .

## VI. KẾT LUẬN

Sau khi đọc xong bài cáo này, thì chúng ta cũng đã trả lời được các câu hỏi trong mục introduction.

- Chọn mô hình chung cho bài toán Linear Regression  $\hat{y} = \mathbf{w}^T \mathbf{x} + \mathbf{b}$
- Tìm một hàm để định nghĩa về độ đo hiệu năng  $P$  (cost function)  $\Rightarrow \theta^* = \arg \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$
- Khi nào nên chọn thuật nào để giải cost function ?
  - Công thức chuẩn:** Khi số lượng đặc trưng  $n < 10,000$  và số lượng mẫu  $m$  không quá lớn.
  - Gradient Descent:** Khi số lượng đặc trưng  $n \geq 10,000$  hoặc số lượng mẫu  $m \geq 100,000$ .
  - SGD:** Khi số lượng mẫu  $m$  rất lớn, có thể lên đến hàng triệu hoặc nhiều hơn.
  - Mini-batch Gradient Descent:** Khi số lượng mẫu  $m$  lớn, từ vài chục nghìn đến hàng triệu.
  - Newton's Method:** Khi số lượng đặc trưng  $n < 1,000$  và số lượng mẫu  $m$  không quá lớn.
- Làm sao để chọn lựa mô hình tốt nhất? Để chọn lựa mô hình tốt nhất thì chúng ta phải giải quyết được overfitting và underfitting
  - Làm sao để tránh underfitting?  $\Rightarrow$  sử dụng phương pháp lựa chọn không gian giả thuyết để tìm dung lượng làm cho sai số huấn luyện nhỏ, và khoảng cách giữa sai số huấn luyện và sai số mô hình nhỏ nhất
  - Làm sao để tránh overfitting?  $\Rightarrow$  sử dụng các bộ kiểm soát  $L_1, L_2$
- Khi nào chọn  $L_1$ , khi nào chọn  $L_2$**

Tính huống	Nên chọn
Cần loại bỏ bớt các đặc trưng không quan trọng	$L_1$
Các đặc trưng có tương quan cao	$L_1$
Cần mô hình có khả năng giải thích cao	$L_1$
Cần ổn định hơn với ít sự thay đổi nhỏ trong dữ liệu	$L_2$
Các đặc trưng có độ quan trọng gần giống nhau	$L_2$
Số lượng đặc trưng lớn, cần giảm bớt số lượng đặc trưng	$L_1$
Mục tiêu là giảm thiểu lỗi dự đoán (predictive accuracy)	$L_2$

Bảng II: Khi nào nên chọn  $L_1$  và khi nào nên chọn  $L_2$

## VII. PHÂN CHIA CÔNG VIỆC TRONG NHÓM

### TÀI LIỆU THAM KHẢO

#### TÀI LIỆU

- [1] H. Kopka and P. W. Daly, *A Guide to L<sup>A</sup>T<sub>E</sub>X*, 3rd ed. Harlow, England: Addison-Wesley, 1999.