Feature Engineering

1. Feature Scaling

Khái niệm

Khi một đặc trưng có phạm vi giá trị lớn hơn đặc trưng khác, đường cong chi phí (cost function) có hình **ellipse dẹt** – khiến Gradient Descent:

- Cập nhật chậm
- Mất thời gian "lắc lư" để hội tụ đến điểm tối ưu toàn cục (global minimum)

Giải pháp: Feature Scaling (Chuẩn hóa đặc trưng):

- Biến đổi dữ liệu đầu vào để các đặc trưng có **phạm vi giá trị tương đương** (ví dụ, cùng nằm trong khoảng từ 0 đến 1).
- Khi đó, đường contour của hàm mất mát trở nên **gần như tròn** → giúp gradient descent **hội tụ nhanh hơn** và **ổn định hơn**.

Lợi ích chính:

- Giảm số bước cần thiết để gradient descent hội tụ.
- Tránh tình trạng cập nhật quá lớn hoặc quá nhỏ cho các trọng số

Một số phương pháp scaling

Min-Max Scaling

Cách làm: Chia mỗi giá trị cho **giá trị lớn nhất** của đặc trưng đó.

Công thức:

$$x_{\text{scaled}} = \frac{x}{\max(x)}$$

Ưu điểm: Đơn giản, nhanh.

Nhược điểm: Nhạy cảm với giá trị cực trị (outliers).

Mean Normalization

Cách làm: Trừ đi trung bình, chia cho độ dài đoạn giá trị (max - min).

Công thức:

$$x_{\text{normalized}} = \frac{x - \mu}{\max - \min}$$

Ưu điểm: Đưa dữ liệu về trung tâm (mean = 0).

Thường dùng cho: Dữ liệu không quá lệch (not skewed).

Z-score Normalization (Standardization)

Cách làm: Trừ trung bình và chia cho độ lệch chuẩn.

Công thức:

$$x_{\text{zscore}} = \frac{x - \mu}{\sigma}$$

Ưu điểm:

- Dữ liệu có mean = 0, std = 1.
- Phù hợp khi các đặc trưng có phân phối gần chuẩn (Gaussian).

Khi nào nên dùng Feature Scaling?

- Khi các đặc trưng có **đơn vị đo lường khác nhau** hoặc chênh lệch phạm vi lớn.
- Khi dùng các thuật toán nhạy với độ lớn đặc trưng như:
  - Gradient Descent (trong Linear/Logistic Regression, Neural Networks)
  - K-Means, SVM, PCA...

2. Convergence

Đầu tiên Vẽ biểu đồ học (learning curve) của mô hình, ta có:

- Giá trị **cost J giảm dần qua mỗi lần lặp**.
- Đường cong dốc lúc đầu → sau đó **phẳng dần (leveling off)**.
- Khi đường cong gần như không thay đổi → có thể xem như đã **hội tụ**.
- Nếu J tăng lên sau một số vòng lặp → thường do learning rate (α) quá lớn hoặc lỗi trong code.

Kiểm tra hội tụ tự động:

- Dùng một ngưỡng nhỏ gọi là ε (ví dụ 0.001 hoặc 10^{-3}).
- Nếu J giảm **ít hơn ε** trong 1 vòng lặp → coi nh hội tụ.
- Tuy nhiên, **việc chọn ε tốt khá khó**, nên thường **quan sát biểu đồ học** vẫn tốt hơn.

Chiến lược chọn learning rate α tốt:

- Thử **nhiều giá trị α**, ví dụ:
  - Bắt đầu từ: `0.001`
  - Tăng gấp ~3 lần: `0.003`, `0.01`, `0.03`, `0.1`, `0.3`, ...
- **Vẽ biểu đồ học (learning curve)** tương ứng với từng α.
- Chọn α mà:
  - **J giảm đều, nhanh** qua các vòng lặp.
  - Không làm J tăng hay dao động quá mạnh.

3. Feature engineering

- Việc chọn đúng **đặc trưng (feature)** có **ảnh hưởng rất lớn** đến hiệu suất của thuật toán học.
- Trong nhiều ứng dụng thực tế, **thiết kế hoặc chọn đặc trưng tốt** là bước quan trọng nhất để mô hình hoạt động hiệu quả.

Feature Engineering là gì?

- Là quá trình **biến đổi hoặc kết hợp các đặc trưng gốc** để tạo ra đặc trưng mới có ích hơn.
- Dựa vào **kiến thức chuyên môn hoặc trực giác về bài toán**.