

# 1 Linear regression

## Định nghĩa bài toán

### Đầu vào

Cho tập dữ liệu gồm  $N$  mẫu:

- Tập các vector đặc trưng:

$$\mathbf{x}_n = [x_{n,0}, x_{n,1}, \dots, x_{n,D-1}]^T, \quad n = 1, \dots, N \quad \mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \eta \nabla E_D(\mathbf{w}) \quad (8)$$

với  $x_{n,0} = 1$ .

- Vector nhãn:

$$\mathbf{t} = [t_1, t_2, \dots, t_N]^T$$

Dữ liệu được tổ chức dưới dạng:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,D-1} \\ 1 & x_{2,1} & \dots & x_{2,D-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \dots & x_{N,D-1} \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

### Giả thiết

Giả sử nhãn được sinh theo mô hình:

$$t = h(\mathbf{x}) + \varepsilon \quad (1)$$

Trong đó:

- $h(\mathbf{x})$ : hàm hồi quy tối ưu (chưa biết)
- $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ : nhiễu Gauss

Các mẫu dữ liệu và nhiễu được giả thiết là *i.i.d.*

### Đầu ra

Mô hình hồi quy tuyến tính:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} \quad (2)$$

với:

$$\mathbf{w} = [w_0, w_1, \dots, w_{D-1}]^T$$

### Giải bài toán

#### Nguyên tắc chung

Bài toán được giải bằng cách:

- Xây dựng hàm hợp lý
- Cực đại hóa hợp lý (Maximum Likelihood)

#### Hợp lý cực đại

Với giả thiết nhiễu Gauss:

$$p(t_n | \mathbf{x}_n, \mathbf{w}, \beta) = \mathcal{N}(t_n | \mathbf{w}^T \mathbf{x}_n, \beta^{-1}) \quad (3)$$

Hàm negative log-likelihood:

$$L(\mathbf{w}, \beta) = \beta E_D(\mathbf{w}) - \frac{N}{2} \log \beta + \frac{N}{2} \log(2\pi) \quad (4)$$

với:

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 \quad (5)$$

#### Nghiem giải tích

Cực tiểu  $E_D(\mathbf{w})$  cho nghiệm:

$$\mathbf{w}_{ML} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t} \quad (6)$$

Ước lượng phương sai nhiễu:

$$\beta_{ML}^{-1} = \frac{1}{N} \sum_{n=1}^N (t_n - \mathbf{w}_{ML}^T \mathbf{x}_n)^2 \quad (7)$$

### Giải thuật lặp (Gradient Descent)

Cập nhật tham số:

### Hồi quy cho quan hệ phi tuyến

Ánh xạ dữ liệu sang không gian đặc trưng mới:

$$\mathbf{x} \rightarrow \phi(\mathbf{x}) = [\phi_0(\mathbf{x}), \dots, \phi_{M-1}(\mathbf{x})]$$

Sau đó áp dụng hồi quy tuyến tính trên không gian mới.

### Đánh giá mô hình

#### Dự báo

$$\hat{\mathbf{y}} = \mathbf{X} \mathbf{w}_{ML} \quad (9)$$

### Các độ đo

Mean Squared Error:

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^N (t_n - \hat{y}_n)^2 \quad (10)$$

Root Mean Squared Error:

$$\text{RMSE} = \sqrt{\text{MSE}} \quad (11)$$

### Hạn chế quá khớp

#### Ridge Regression

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \quad (12)$$

Nghiem:

$$\mathbf{w}_{ridge} = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t} \quad (13)$$

### LASSO

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \lambda \sum_{m=1}^M |w_m| \quad (14)$$

### Dự báo cho nhiều biến

Với  $K$  biến đầu ra:

$$\mathbf{W} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{T} \quad (15)$$

Trong đó:

- $\mathbf{T} \in \mathbb{R}^{N \times K}$
- $\mathbf{W} \in \mathbb{R}^{M \times K}$

## 2 Logistic regression

### Giới thiệu bài toán

#### Đầu vào

#### Dữ liệu đầu vào

Ma trận dữ liệu:

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,(M-1)} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,(M-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \dots & x_{N,(M-1)} \end{bmatrix}$$

- Mỗi hàng tương ứng với một điểm dữ liệu

- $X$  có kích thước  $N \times M$

### Tập nhãn

- Bài toán có hai nhãn

- Mỗi nhãn được mã hóa bằng chỉ số  $\{0, 1\}$

### Nhãn dữ liệu

$$t = (t_1, t_2, \dots, t_N)^T$$

### Quy ước

- $X$ : ma trận dữ liệu, kích thước  $N \times M$

- $t$ : vector nhãn

- $y$ : biểu diễn dạng số của nhãn

- $N$ : số điểm dữ liệu

- $M$ : số đặc trưng

- $\hat{y}$ : giá trị dự báo từ mô hình

### Phương pháp xây dựng mô hình

#### Ý tưởng

- Hai lớp:  $C_0$  và  $C_1$

- Mô hình dự báo xác suất  $x$  thuộc lớp  $C_1$

- Xác suất thuộc lớp  $C_0$  là  $1 - \hat{y}$

Bên trong mô hình:

- Sử dụng mô hình tuyến tính:

$$z = \mathbf{w}^T \mathbf{x}$$

- Đưa  $z$  qua hàm sigmoid

#### Hàm sigmoid

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

#### Mô hình dự báo

$$\hat{y} = p(C_1 | x, w) = \sigma(\mathbf{w}^T \mathbf{x}) \quad (16)$$

Quy tắc phân lớp:

- Nếu  $\hat{y} \geq \lambda$  thì  $x \in C_1$

- Ngược lại,  $x \in C_0$

### Ước lượng tham số của mô hình

#### Xây dựng hàm mục tiêu

Với một điểm dữ liệu  $(x, y)$ :

$$p(y | x, w) = \hat{y}^y (1 - \hat{y})^{1-y}$$

Với  $N$  điểm dữ liệu:

$$p(t | X, w) = \prod_{n=1}^N \hat{y}_n^{y_n} (1 - \hat{y}_n)^{1-y_n}$$

Sử dụng **negative log-likelihood**: Hàm softmax được định nghĩa:

$$L(w) = - \sum_{n=1}^N [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]$$

$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{i=1}^K \exp(z_i)}$$

Hàm này còn được gọi là **cross-entropy**.  
**Tìm hệ số của mô hình**  
Gradient của hàm mất mát:

$$\nabla L(w) = \sum_{n=1}^N (\hat{y}_n - y_n) x_n = X^T (\hat{y} - y)$$

**Giải thuật lặp với đạo hàm bậc 2**  
Ma trận Hessian:

$$H = \nabla^2 L(w) = \sum_{n=1}^N \hat{y}_n (1 - \hat{y}_n) x_n x_n^T = X^T R X$$

Trong đó  $R$  là ma trận đường chéo:

$$R_{nn} = \hat{y}_n (1 - \hat{y}_n)$$

Phương pháp sử dụng:

- Gradient Descent
- Newton–Raphson
- Iterative Re-weighted Least Squares (IRLS)

### 3 Softmax regression

**Dữ liệu đầu vào**  
**Ma trận dữ liệu**

Giả sử tập dữ liệu đầu vào được biểu diễn bởi ma trận:

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,(M-1)} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,(M-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,(M-1)} \end{bmatrix}$$

trong đó  $X \in \mathbb{R}^{N \times M}$ .

**Nhân và biểu diễn one-hot**

Mỗi nhân được mã hóa dưới dạng vectơ one-hot kích thước  $K$ . Ví dụ với  $K = 4$ :

Nhân	Chỉ số	One-hot
Chó	0	[1, 0, 0, 0]
Mèo	1	[0, 1, 0, 0]
Chuột	2	[0, 0, 1, 0]
Thỏ	3	[0, 0, 0, 1]

**Mô hình tuyến tính với Softmax**

**Mô hình dự báo**

Ma trận tham số của mô hình:

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_K^T \end{bmatrix} \in \mathbb{R}^{K \times M}$$

Các bước tính toán:

$$Z = XW^T \quad (N \times K) \tag{20}$$

$$\hat{Y} = \text{softmax}(Z) \tag{21}$$

**Dự đoán**

Nhân dự đoán của mẫu dữ liệu được xác định bằng:

$$\text{prediction} = \arg \max_k \hat{y}_k$$

**Hàm mục tiêu và tối ưu**

**Hàm hợp lý**

Xác suất của tập nhân:

$$p(t|X, W) = \prod_{n=1}^N \prod_{k=1}^K \hat{y}_{n,k}^{y_{n,k}}$$

**Hàm mất mát Cross-Entropy**

Hàm mất mát được xây dựng bằng cách lấy log và đổi dấu:

$$L(W) = - \sum_{n=1}^N \sum_{k=1}^K y_{n,k} \log(\hat{y}_{n,k})$$

Mục tiêu là tìm  $W$  sao cho  $L(W)$  đạt giá trị nhỏ nhất.

**Ước lượng tham số**

**Gradient Descent**

Gradient của hàm mất mát đối với đầu vào softmax:

$$\frac{\partial L}{\partial z} = (\hat{y} - y)^T$$

Gradient theo tham số:

$$\Delta W = (\hat{y} - y)^T x^T$$

Cập nhật trọng số:

$$W \leftarrow W - \eta \Delta W$$

trong đó  $\eta$  là hệ số học.

**Mở rộng cho đường biên phi tuyến**

Để xử lý dữ liệu không phân tách tuyến tính, có thể:

- Biến đổi đặc trưng sang không gian mới
- Sử dụng hàm cơ sở đa thức
- Áp dụng mạng nơ-ron để học đặc trưng

### 4 MLP

**MLP: Computational Architecture View**

An MLP consists of:

- A **feature transformer**: stacked linear layers with nonlinear activations.
- An **output head**:
  - Linear head for regression
  - Logistic or softmax head for classification

**Core idea**: deep learning = nonlinear feature extraction + simple linear head.

**MLP: Mathematical Model**

**Forward Pass**

Let  $h^{(0)} = x$ . Each hidden layer computes:

$$h^{(l)} = \phi \left( W^{(l)} h^{(l-1)} + b^{(l)} \right), \quad l = 1, \dots, L \tag{22}$$

where  $\phi(\cdot)$  is a nonlinear activation function.

**Output Layer**

**Regression:**

$$\hat{y} = W^{(L+1)} h^{(L)} + b^{(L+1)} \tag{23}$$

**Classification (Softmax):**

$$\hat{p}_k = \frac{\exp(w_k^T h^{(L)} + b_k)}{\sum_j \exp(w_j^T h^{(L)} + b_j)} \tag{24}$$

**Function Composition View**

An MLP is a composition of functions:

$$f(x; \theta) = f^{(L+1)} \circ \phi \circ f^{(L)} \circ \dots \circ \phi \circ f^{(1)}(x) \tag{25}$$

More layers imply higher representation power.

**MLP: Layers**

**Fully Connected (Linear) Layer**

For a single sample:

$$y = Wx + b \tag{26}$$

For a mini-batch  $X \in \mathbb{R}^{B \times N}$ :

$$Y = XW^T + \mathbf{1}b^T \tag{27}$$

**Activation Functions**

**Sigmoid**

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{28}$$

**Tanh**

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{29}$$

**ReLU**

$$\text{ReLU}(z) = \max(0, z) \tag{30}$$

**Leaky ReLU**

$$\text{LReLU}(z) = \begin{cases} z, & z \geq 0 \\ \alpha z, & z < 0 \end{cases} \tag{31}$$

**SiLU (Swish)**

$$\text{SiLU}(z) = z \sigma(z) \tag{32}$$

### 5 Training ANN

**Problem Setup** Given a dataset  $\{(x_i, y_i)\}_{i=1}^n$  and a model  $\hat{y}_i = f_\theta(x_i)$ , training aims to solve:

$$\min_{\theta} L(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, \hat{y}_i).$$

**Regression Losses**

Mean Squared Error (MSE)

L\_MSE = 1/n \sum\_{i=1}^n (y\_i - \hat{y}\_i)^2.

MSE penalizes large errors strongly but is sensitive to outliers.

Mean Absolute Error (MAE)

L\_MAE = 1/n \sum\_{i=1}^n |y\_i - \hat{y}\_i|.

MAE is more robust to outliers but is not differentiable at zero.

Huber Loss

L\_\delta(e\_i) = \begin{cases} \frac{1}{2}e\_i^2, & |e\_i| \leq \delta, \\ \delta(|e\_i| - \frac{1}{2}\delta), & |e\_i| > \delta, \end{cases}

where e\_i = y\_i - \hat{y}\_i.

Classification Losses

Binary Cross-Entropy (BCE) For y\_i \in \{0, 1\} and p\_i = \sigma(z\_i):

L\_{BCE} = -\frac{1}{n} \sum\_{i=1}^n [y\_i \log p\_i + (1 - y\_i) \log(1 - p\_i)].

Categorical Cross-Entropy For K classes with one-hot labels:

L\_{CE} = -\frac{1}{n} \sum\_{i=1}^n \sum\_{k=1}^K y\_{ik} \log p\_{ik}.

Training Process Training proceeds iteratively through three main steps:

- 1. **Forward pass:** compute predictions and loss.
- 2. **Backward pass:** compute gradients via backpropagation.
- 3. **Update step:** update parameters using an optimizer.

Stochastic Gradient Descent

\theta \leftarrow \theta - \eta \nabla\_\theta L.

Training Algorithm

Algorithm 1 SGD Training Procedure	
1:	Initialize parameters $\theta$
2:	<b>for</b> epoch = 1 to $E$ <b>do</b>
3:	Shuffle dataset and create mini-batches
4:	<b>for</b> each mini-batch $(X, y)$ <b>do</b>
5:	Forward pass
6:	Compute loss $L$
7:	Backward pass: compute $\nabla_\theta L$
8:	Update: $\theta \leftarrow \theta - \eta \nabla_\theta L$
9:	<b>end for</b>
10:	<b>end for</b>

Optimization Methods SGD with Momentum

v\_t = \mu v\_{t-1} + g\_t, \quad \theta \leftarrow \theta - \eta v\_t.

Adam

m\_t = \beta\_1 m\_{t-1} + (1 - \beta\_1) g\_t, \\ v\_t = \beta\_2 v\_{t-1} + (1 - \beta\_2) g\_t^2, \\ \theta \leftarrow \theta - \eta \frac{\hat{m}\_t}{\sqrt{\hat{v}\_t + \epsilon}}.

AdamW AdamW decouples weight decay from the gradient update, improving generalization, especially in large-scale models.

Training Techniques Learning Rate Scheduling Common strategies include step decay, cosine annealing, and warm restarts.

Regularization

- L2 weight decay
- Dropout
- Early stopping

Normalization Batch Normalization and Layer Normalization stabilize training and allow larger learning rates.

Practical Considerations Initialization Xavier initialization is suitable for sigmoid/tanh, while He initialization is designed for ReLU activations.

Gradient Issues Vanishing and exploding gradients can be mitigated using ReLU activations, residual connections, and gradient clipping.

6 SVM primal problem

Giới thiệu bài toán Đầu vào

Dữ liệu huấn luyện gồm:

- Ma trận dữ liệu:

X = \begin{bmatrix} x\_{1,1} & x\_{1,2} & \cdots & x\_{1,(M-1)} \\ x\_{2,1} & x\_{2,2} & \cdots & x\_{2,(M-1)} \\ \vdots & \vdots & \ddots & \vdots \\ x\_{N,1} & x\_{N,2} & \cdots & x\_{N,(M-1)} \end{bmatrix}

với X \in \mathbb{R}^{N \times (M-1)}.

- Véc-tơ nhãn:

\mathbf{t} = \begin{bmatrix} t\_1 \\ t\_2 \\ \vdots \\ t\_N \end{bmatrix}, \quad t\_n \in \{-1, +1\}

Giả thiết Dữ liệu thuộc hai lớp có thể phân tách tuyến tính, tức tồn tại một siêu phẳng sao cho các điểm mang nhãn +1 và -1 nằm ở hai phía khác nhau. Mục tiêu

Xác định đường biên quyết định:

\mathbf{w}^T \mathbf{x} + b = 0

sao cho lề (margin) giữa hai lớp là lớn nhất. SVM với scikit-learn Ví dụ sử dụng SVM trong scikit-learn:

```
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
clf.predict([[2., 2.]])
```

Phương pháp xây dựng bộ phân lớp Nguyên tắc Quy trình gồm ba bước chính:

1. Chuyển về bài toán tối ưu có ràng buộc với mục tiêu cực đại lề (bài toán gốc).
2. Chuyển sang bài toán đối ngẫu (Dual problem).
3. Giải bài toán tối ưu bằng các thư viện tối ưu lồi như CVXOPT.

Bài toán gốc (Primal Problem) Khoảng cách từ điểm đến đường thẳng Siêu phẳng trong không gian đặc trưng (M - 1) chiều:

\mathbf{w}^T \mathbf{x} + b = 0

Khoảng cách có dấu từ điểm \mathbf{x} đến siêu phẳng:

d(\mathbf{x}) = \frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|}

Khoảng cách hình học:

|d(\mathbf{x})| = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|}

Hàm quyết định Hàm quyết định:

y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b

Quy tắc phân lớp:

\text{class}(\mathbf{x}) = \text{sign}(y(\mathbf{x}))

Lề (Margin) Lề trên tập huấn luyện:

m\_{\mathbf{w}} = \min\_n \frac{t\_n(\mathbf{w}^T \mathbf{x}\_n + b)}{\|\mathbf{w}\|}

Cực đại lề Có thể chuẩn hóa sao cho:

t\_n(\mathbf{w}^T \mathbf{x}\_n + b) \geq 1, \quad \forall n

Hàm mục tiêu Cực đại lề tương đương với bài toán:

\min\_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2

với ràng buộc:

$$t_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1, \quad n = 1, \dots, N$$

## Bài toán gốc

$$\mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{s.t. } t_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1, \quad \forall n \text{ trong đó}$$

## Giải bằng thư viện CVXOPT

Dạng chuẩn:

$$\min_{\mathbf{x}} \frac{1}{2} \mathbf{x}^T K \mathbf{x} + \mathbf{p}^T \mathbf{x} \quad \text{s.t. } G \mathbf{x} \leq \mathbf{h}$$

Trong đó:

$$\mathbf{x} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

## 7 SVM dual problem

### Bài toán đối ngẫu

#### Hàm Lagrangian

Hàm Lagrangian của bài toán (??) là:

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^N \alpha_n [t_n(w^T x_n + b) - 1], \quad (33)$$

với

$$\alpha_n \geq 0, \quad n = 1, \dots, N.$$

#### Điều kiện KKT

Bài toán thỏa hệ điều kiện KKT:

- **(KKT-1)** Điều kiện dừng:

$$\nabla_{w,b} L(w, b, \alpha) = 0$$

- **(KKT-2)** Ràng buộc gốc

- **(KKT-3)** Ràng buộc đối ngẫu:  $\alpha_n \geq 0$

- **(KKT-4)** Điều kiện bù:

$$\alpha_n [1 - t_n(w^T x_n + b)] = 0$$

#### Xây dựng hàm đối ngẫu

Lấy đạo hàm theo  $w$  và  $b$ :

$$\frac{\partial L}{\partial w} = w - \sum_{n=1}^N \alpha_n t_n x_n = 0 \quad (34)$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^N \alpha_n t_n = 0 \quad (35)$$

Suy ra:

$$w = \sum_{n=1}^N \alpha_n t_n x_n. \quad (36)$$

Thay vào Lagrangian, ta thu được hàm đối ngẫu:

$$g(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{r=1}^N \sum_{c=1}^N \alpha_r \alpha_c t_r t_c x_r^T x_c. \quad (37)$$

#### Bài toán đối ngẫu

Bài toán đối ngẫu tương đương:

$$\alpha^* = \arg \min_{\alpha} \frac{1}{2} \alpha^T K \alpha - \mathbf{1}^T \alpha$$

$$\text{s.t. } \alpha_n \geq 0, \quad n = 1, \dots, N, \quad (38)$$

$$\sum_{n=1}^N \alpha_n t_n = 0,$$

$$K_{rc} = t_r t_c x_r^T x_c.$$

#### Tiêu chuẩn Slater

Vì tồn tại  $(w, b)$  sao cho

$$t_n(w^T x_n + b) > 1, \quad \forall n,$$

nên bài toán thỏa tiêu chuẩn Slater. Do đó:

$$\min_{w,b} \max_{\alpha} L(w, b, \alpha) = \max_{\alpha} \min_{w,b} L(w, b, \alpha).$$

Suy ra **duality gap bằng 0**.

#### Công thức dự báo

Với tập véc-tơ hỗ trợ  $S = \{n : \alpha_n > 0\}$ ,

$$y(x) = \sum_{n \in S} \alpha_n t_n x_n^T x + b. \quad (39)$$

Nhãn dự báo:

$$\hat{y} = \text{sign}(y(x)).$$

## 8 SVM soft margin

### Giới thiệu bài toán không khả tách tuyến tính

Trong thực tế, dữ liệu thường **không khả tách tuyến tính**. Khi đó:

- Tập nghiệm khả thi của hard-margin là rỗng.
- Không tồn tại  $w, b$  thỏa tất cả các ràng buộc.

Do đó, cần mở rộng mô hình bằng cách cho phép một số điểm vi phạm ràng buộc.

#### Nguyên tắc Soft Margin

Ý tưởng chính:

1. Nới lỏng ràng buộc bằng biến phạt  $\xi_n$ .
2. Phạt các điểm nằm sai vị trí thông qua hàm mục tiêu.

Mô hình này được gọi là **Soft Margin SVM**.

#### Bài toán gốc với lề mềm

##### Ràng buộc mới

$$t_n(w^T x_n + b) \geq 1 - \xi_n, \quad n = 1, \dots, N, \quad (40)$$

$$\xi_n \geq 0. \quad (41)$$

#### Hàm mục tiêu mới

$$f_0(w, b, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{n=1}^N \xi_n, \quad (42)$$

trong đó  $C > 0$  là siêu tham số điều chỉnh mức phạt.

#### Bài toán tối ưu

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} \|w\|^2 + C \sum_{n=1}^N \xi_n \\ \text{s.t.} \quad & t_n(w^T x_n + b) \geq 1 - \xi_n, \\ & \xi_n \geq 0, \quad n = 1, \dots, N. \end{aligned} \quad (43)$$

#### Bài toán đối ngẫu

##### Hàm Lagrangian

$$\begin{aligned} L(w, b, \xi, \alpha, \mu) = & \frac{1}{2} \|w\|^2 + C \sum_{n=1}^N \xi_n \\ & - \sum_{n=1}^N \alpha_n [t_n(w^T x_n + b) - 1 + \xi_n] \\ & - \sum_{n=1}^N \mu_n \xi_n \end{aligned} \quad (44)$$

#### Điều kiện KKT

$$w = \sum_{n=1}^N \alpha_n t_n x_n, \quad (45)$$

$$\sum_{n=1}^N \alpha_n t_n = 0, \quad (46)$$

$$0 \leq \alpha_n \leq C. \quad (47)$$

#### Bài toán đối ngẫu

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{r=1}^N \sum_{c=1}^N \alpha_r \alpha_c t_r t_c x_r^T x_c - \sum_{n=1}^N \alpha_n \\ \text{s.t.} \quad & 0 \leq \alpha_n \leq C, \\ & \sum_{n=1}^N \alpha_n t_n = 0. \end{aligned} \quad (48)$$

#### Công thức dự báo

Sau khi tìm được  $\alpha$  và  $b$ , hàm quyết định là:

$$y(x) = \sum_{n \in S} \alpha_n t_n x_n^T x + b, \quad (49)$$

trong đó  $S$  là tập các véc-tơ hỗ trợ.

Nhãn dự báo:

$$\text{label} = \text{sign}(y(x)).$$

#### Cài đặt với CVXOPT

Bài toán đối ngẫu có dạng chuẩn:

$$\min_{\alpha} \quad \frac{1}{2} \alpha^T K \alpha + p^T \alpha \quad \text{s.t.} \quad G \alpha \leq h, \quad A \alpha = b. \quad (50)$$

Các ràng buộc hộp  $0 \leq \alpha_n \leq C$  được mã hóa trong ma trận  $G$  và  $h$ .

9 SVM kernel

**SVM hai lớp với lề mềm** Xét tập huấn luyện  $\{(x_i, t_i)\}_{i=1}^N$  với  $t_i \in \{-1, +1\}$ . Bài toán tối ưu của SVM lề mềm được viết dưới dạng:

alpha^\* = arg min\_alpha 1/2 alpha^T K alpha + p^T alpha (51)

với các ràng buộc:

G alpha <= h, A alpha = b (52)

Trong đó, ma trận kernel K được xác định bởi tích vô hướng giữa các điểm dữ liệu. **Dự báo** Giá trị bias b được ước lượng bởi:

b = 1/N\_M (t\_M - K\_MS[alpha\_S o t\_S])^T 1 (53)

Hàm dự báo:

y = K\_BS[alpha\_S o t\_S] + b (54)

Nhân dự đoán:

label = sign(y) (55)

**Khi đường biên giới phi tuyến** Trong nhiều bài toán thực tế, dữ liệu không thể phân tách tuyến tính. Giải pháp là ánh xạ dữ liệu thông qua hàm trích đặc trưng:

Phi : R^d -> H

Tuy nhiên, việc tính trực tiếp Phi(x) có thể tốn kém hoặc không khả thi khi không gian đặc trưng có số chiều rất lớn hoặc vô hạn. **Phương pháp Kernel** Phương pháp kernel cho phép tính:

<Phi(x\_i), Phi(x\_j)>

mà không cần biết tường minh Phi(x), thông qua một hàm kernel:

k(x\_i, x\_j)

**Điều kiện Mercer** Một hàm k(x\_i, x\_j) là kernel hợp lệ nếu:

- Đối xứng: k(x\_i, x\_j) = k(x\_j, x\_i)
- Bán định dương:

sum\_{i=1}^N sum\_{j=1}^N c\_i c\_j k(x\_i, x\_j) >= 0

**Huấn luyện và dự báo với Kernel** Ma trận Gram kernel:

K\_Gram = [ [k(x\_1, x\_1) ... k(x\_1, x\_N)] [k(x\_N, x\_1) ... k(x\_N, x\_N)] ] (56)

Việc sử dụng kernel đảm bảo bài toán tối ưu là lồi và có nghiệm toàn cục.

**Các kernel thông dụng** Một số kernel phổ biến trong thực tế:

- **Linear:**

k(x, x') = x^T x'

- **Polynomial:**

k(x, x') = (gamma^T x' + r)^d

- **RBF (Gaussian):**

k(x, x') = exp(-gamma ||x - x'||^2)

- **Sigmoid:**

k(x, x') = tanh(gamma^T x' + r)

Kernel RBF tương ứng với không gian đặc trưng vô hạn chiều.

**Thiết kế Kernel** Việc thiết kế kernel phụ thuộc mạnh vào kiến thức miền (domain knowledge), với mục tiêu:

- k(x\_i, x\_j) lớn nếu x\_i, x\_j cùng lớp
- k(x\_i, x\_j) nhỏ nếu khác lớp

**Minh họa** Các thí nghiệm với kernel đa thức và kernel RBF cho thấy khả năng phi tuyến hóa đường biên phân lớp một cách hiệu quả.

10 PCA

**Mô tả bài toán** Giả sử tập dữ liệu đầu vào:

X = [ [x\_1,1 x\_1,2 ... x\_1,D] [x\_2,1 x\_2,2 ... x\_2,D] [x\_N,1 x\_N,2 ... x\_N,D] ]

trong đó:

- X in R^{N x D};
- D rất lớn và các chiều có tương quan với nhau.

**Mục tiêu:**

1. Giảm số chiều từ D xuống M với M << D;
2. Các đặc trưng mới không còn tương quan tuyến tính.

**Kiến thức toán học liên quan Phương sai và hiệp phương sai** Trung bình của dữ liệu:

mu = 1/N sum\_{n=1}^N x\_n

Dữ liệu được chuẩn hóa:

z\_n = x\_n - mu

Ma trận hiệp phương sai:

S = 1/N sum\_{n=1}^N (x\_n - mu)(x\_n - mu)^T

**Eigenvalue và Eigenvector** Với ma trận vuông A, bài toán eigen:

Au = lambda u

trong đó u là eigenvector và lambda là eigenvalue.

**Cơ sở lý luận của PCA** PCA có thể được nhìn theo hai cách:

- Cực đại hóa phương sai của dữ liệu sau khi chiếu;
- Cực tiểu hóa sai số phục hồi dữ liệu.

Hai cách tiếp cận này là tương đương về mặt toán học.

**Cực đại hóa phương sai** Với một vectơ đơn vị u, phương sai của dữ liệu khi chiếu lên u là:

sigma^2 = u^T S u

Bài toán tối ưu:

max\_u u^T S u s.t. u^T u = 1

Sử dụng nhân tử Lagrange dẫn đến:

Su = lambda u

Do đó:

- Các trục chính của PCA là các eigenvector của S;
- Phương sai tương ứng là các eigenvalue.

**Thu giảm số chiều** Chọn M eigenvector tương ứng với M eigenvalue lớn nhất, tạo thành ma trận:

U = [u\_1, u\_2, ..., u\_M]

Chiều dữ liệu:

X\_PCA = (X - mu^T) U

Phục hồi xấp xỉ:

X\_hat = mu^T + X\_PCA U^T

**PCA qua API** Ví dụ PCA trong scikit-learn:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
print(pca.explained_variance_ratio_)
```

**Singular Value Decomposition (SVD)** Phân rã SVD:

X = U S V^T

- PCA thực hiện eigen-decomposition trên ma trận hiệp phương sai;
- SVD phân rã trực tiếp trên ma trận dữ liệu và có độ ổn định số cao hơn.

**Ứng dụng của PCA**

- Nén dữ liệu;
- Trực quan hóa dữ liệu nhiều chiều;
- Tiền xử lý cho học máy;
- Nhận dạng mẫu và xử lý ảnh.

# 11 LDA

## Giới thiệu về LDA

### Đầu vào

Cho tập dữ liệu:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{bmatrix}, \quad t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

Trong đó:

- $X \in \mathbb{R}^{N \times D}$ , với  $D$  thường rất lớn.
- $t_k \in \{1, 2, \dots, C\}$  là nhãn lớp của điểm dữ liệu thứ  $k$ .

### Mục tiêu

Mục tiêu của LDA là:

1. Giảm số chiều từ  $D$  xuống  $M$ , với  $M \leq C - 1$ .
2. Dữ liệu sau khi chiếu có độ phân tách giữa các lớp là lớn nhất.

### LDA qua API

Ví dụ sử dụng Scikit-learn:

```
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
lda = LinearDiscriminantAnalysis(n_components=2)
X_r = lda.fit(X, y).transform(X)
```

### Bài toán tối ưu

#### Tâm của mỗi lớp

Với lớp  $k$ :

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{x}_n$$

#### Between-class scatter matrix

$$S_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$

#### Within-class scatter matrix

$$S_W = \sum_{k=1}^C \sum_{n \in C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T$$

#### Hàm mục tiêu của Fisher

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$$

Mục tiêu:

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} J(\mathbf{w})$$

### Tìm nghiệm

Giải bài toán tối ưu dẫn đến phương trình trị riêng:

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

Hướng chiếu tối ưu là:

$$\mathbf{w} \propto S_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

### Trường hợp có $C$ lớp

#### Hàm mục tiêu tổng quát

$$J(W) = \frac{\text{trace}(W^T S_B W)}{\text{trace}(W^T S_W W)}$$

#### Số chiều tối đa

Số chiều tối đa có thể chọn là:

$$M \leq C - 1$$

Do ma trận  $S_B$  có hạng tối đa là  $C - 1$ .

#### Giải thuật LDA

1. Tính  $S_B$  và  $S_W$ .
2. Tính  $A = S_W^{-1} S_B$ .
3. Thực hiện SVD hoặc eigen-decomposition.
4. Chọn  $M$  eigenvector tương ứng với eigenvalue lớn nhất.
5. Chiếu dữ liệu:  $\hat{X} = (X - m^T)W$ .

## 12. Ensemble

### Bias-Variance Perspective

Prediction error can be decomposed into bias and variance. Bias results from overly simplistic assumptions, while variance reflects sensitivity to training data fluctuations. Ensemble learning aims to reduce variance by averaging predictions of multiple weakly correlated models. For regression, the variance of an ensemble predictor can be approximated as

$$\text{Var} \left( \frac{1}{M} \sum_{m=1}^M \hat{y}^{(m)} \right) \approx \frac{1}{M^2} \sum_{m=1}^M \text{Var}(\hat{y}^{(m)}).$$

### Bagging (Bootstrap Aggregating)

Bagging is designed to reduce variance, particularly for unstable learners such as decision trees.

#### Method Description

Given a training dataset

$$D = \{(x_i, y_i)\}_{i=1}^n,$$

Bagging constructs an ensemble of  $M$  models as follows:

1. Draw  $M$  bootstrap datasets  $D_1, D_2, \dots, D_M$  by sampling  $n$  points with replacement from  $D$ .
2. Train a base learner on each bootstrap dataset to obtain models  $h_1, h_2, \dots, h_M$ .
3. Combine predictions of all models:

$$\hat{y}(x) = \begin{cases} \frac{1}{M} \sum_{m=1}^M h_m(x), & \text{regression,} \\ \text{majority vote,} & \text{classification.} \end{cases}$$

### Properties

Bagging is simple, highly parallelizable, and effective at variance reduction. However, it requires storing many models and results in slower inference.

### Random Forest

Random Forest extends Bagging by introducing randomness in feature selection. Each tree is trained on a bootstrap dataset, and at each split only a random subset of features is considered.

### Training Procedure

For each tree:

1. Sample a bootstrap dataset from the training set.
2. Grow a decision tree by recursively splitting nodes.
3. At each split, randomly select a subset of features  $F_{\text{sub}} \subset \{1, \dots, d\}$ .
4. Choose the best split using only features in  $F_{\text{sub}}$ .

Typical choices are  $|F_{\text{sub}}| = \sqrt{d}$  for classification and  $|F_{\text{sub}}| = d/3$  for regression.

### Boosting

Boosting methods train models sequentially, where each model focuses on samples misclassified by previous ones. Unlike Bagging, Boosting can reduce both bias and variance.

### AdaBoost

For binary classification with  $y_i \in \{-1, +1\}$ , AdaBoost maintains a weight distribution over training samples. At iteration  $t$ , a weak learner  $h_t$  is trained using weighted data. The final classifier is

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right),$$

where  $\alpha_t$  is determined by the weighted classification error of  $h_t$ .

### Gradient Boosting

Gradient Boosting views ensemble construction as gradient descent in function space. At each iteration, a new weak learner is fitted to the negative gradient of the loss function with respect to current predictions.

### Voting and Stacking

#### Voting

Voting combines predictions of multiple models using a fixed rule such as majority voting or probability averaging. It is simple and often serves as a strong baseline.

#### Stacking

Stacking trains a meta-learner on predictions of base models. To avoid overfitting, cross-validation is used to generate out-of-fold predictions, which are then used as inputs for the meta-model.

### Comparison and Practical Considerations

Bagging is most effective for high-variance models, Boosting can significantly improve accuracy but is sensitive to noise, and Stacking offers the greatest flexibility at the cost of increased complexity. In practice, Random Forests and Gradient Boosting are strong default choices for tabular data.

## 13 Genetic algorithm

### Key Components of Genetic Algorithms

#### Representation (Encoding)

A solution is encoded as a chromosome. Common encoding methods include:

- **Binary encoding:** chromosomes consist of bits (0 or 1)
- **Real-valued encoding:** genes are real numbers
- **Permutation encoding:** chromosomes represent ordered sequences
- **Tree encoding:** solutions are tree structures (used in genetic programming)

#### Fitness Function

The fitness function  $f(x)$  evaluates the quality of a solution  $x$ . For minimization problems, a transformation is typically applied, such as:

$$f_{\max}(x) = \frac{1}{1 + f_{\min}(x)}$$

The fitness function should be computationally efficient, as it is evaluated many times during the evolutionary process.

#### Selection

Selection chooses parent solutions based on fitness. Popular methods include:

- Roulette wheel selection
- Rank selection
- Tournament selection
- Elitism

Elitism ensures that the best individuals are preserved across generations.

#### Crossover

Crossover combines genetic material from two parents to produce offspring. Common techniques include:

- Single-point crossover

- Two-point crossover
- Uniform crossover
- Arithmetic crossover (for real-valued encoding)

#### Mutation

Mutation introduces random changes to maintain population diversity. Typical mutation strategies include:

- Bit flipping for binary encoding
- Gaussian or uniform noise for real-valued encoding
- Swap or inversion for permutation encoding

#### Genetic Algorithm Framework

A standard genetic algorithm follows these steps:

1. Initialize a population of  $N$  individuals
2. Evaluate fitness of each individual
3. Repeat until a termination condition is met:
  - (a) Select parents based on fitness
  - (b) Apply crossover to generate offspring
  - (c) Apply mutation to offspring
  - (d) Evaluate fitness of new individuals
  - (e) Form the next generation (with optional elitism)
4. Return the best solution found

Termination conditions may include a maximum number of generations, fitness convergence, stagnation, or time limits.

#### Simple Example

Consider maximizing the function:

$$f(x) = x^2, \quad x \in \{0, 1, \dots, 15\}$$

Binary encoding with 4 bits is used. An example initial population is shown in Table ??.

After selection, crossover, and mutation, the population gradually converges toward the optimal solution  $x = 15$ .

#### Parameters and Tuning

Key parameters include:

- Population size ( $N$ )
- Crossover probability ( $p_c$ )

- Mutation probability ( $p_m$ )

Typical values are:

- $N = 20\text{--}200$
- $p_c = 0.6\text{--}0.9$
- $p_m = 0.001\text{--}0.1$

Parameter selection is problem-dependent and often requires empirical tuning.

#### Advantages and Disadvantages

- Global search capability
- No requirement for gradient information
- Parallelizable structure
- Flexible solution representation

#### Disadvantages

- No guarantee of global optimality
- Computationally expensive
- Sensitive to parameter settings
- Risk of premature convergence

#### Applications

Genetic Algorithms have been successfully applied in:

- Combinatorial optimization (TSP, scheduling)
- Machine learning (feature selection, hyperparameter tuning)
- Engineering design (antennas, circuits)
- Bioinformatics (protein structure prediction)
- Game AI and procedural content generation

#### Variants and Extensions

Popular GA variants include:

- Real-Coded Genetic Algorithms (RCGA)
- Differential Evolution (DE)
- Genetic Programming (GP)
- Multi-objective Genetic Algorithms (e.g., NSGA-II)

These variants extend GA to continuous, programmatic, and multi-objective optimization problems.