

1 LINEAR REGRESSION

1.1 Định nghĩa bài toán

1.2 Đầu vào

Cho tập dữ liệu gồm N mẫu:

- Tập các vector đặc trưng:

$$\mathbf{x}_n = [x_{n,0}, x_{n,1}, \dots, x_{n,D-1}]^T, \quad n = 1, \dots, N$$

với $x_{n,0} = 1$.

- Vector nhãn:

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

Dữ liệu được tổ chức dưới dạng:

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

1.3 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.*

1.4 Đầu ra

Mô hình hồi quy tuyến tính:

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

với:

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

1.5 Giải bài toán

1.6 Nguyên tắc chung

Bài toán được giải bằng cách:

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

1.7 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:

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

1.8 Nghiệm 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:

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

1.9 Giải thuật lặp (Gradient Descent)

Cập nhật tham số:

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \eta \nabla E_D(\mathbf{w}) \quad (8)$$

2 LOGISTIC REGRESSION

2.1 Giới thiệu bài toán

2.2 Đầu vào

2.2.1 Dữ liệu đầu vào

Ma trận dữ liệu:

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

- Mỗi hàng tương ứng với một điểm dữ liệu

- \mathbf{X} có kích thước $N \times M$

2.2.2 Tập nhãn

- Bài toán có hai nhãn

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

2.2.3 Nhãn dữ liệu

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

2.2.4 Quy ước

- \mathbf{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

2.3 Phương pháp xây dựng mô hình

2.4 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:

$$z = w^T x$$

- Đưa z qua hàm sigmoid

2.5 Hàm sigmoid

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

1.17 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:

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

2.6 Mô hình dự báo

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

Quy tắc phân lớp:

- Nếu $\hat{y} \geq \lambda$ thì $x \in C_1$
- Ngược lại, $x \in C_0$

2.7 Ước lượng tham số của mô hình

2.8 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**:

$$L(w) = - \sum_{n=1}^N [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)] \quad (17)$$

Hàm này còn được gọi là **cross-entropy**.

2.9 Tìm hệ số của mô hình

Gradient của hàm mất mát:

$$\nabla L(w) = \sum_{n=1}^N (\hat{y}_n - y_n) x_n = X^T (\hat{y} - y) \quad (18)$$

2.10 Giải thuật lặp với đạo hàm bậc 2

Ma trận Hessian:

$$H = \nabla^2 L(w) = \sum_{n=1}^N \hat{y}_n (1 - \hat{y}_n) x_n x_n^T = X^T R X \quad (19)$$

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

3.1 Dữ liệu đầu vào

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

3.3 Nhãn và biểu diễn one-hot

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

Nhãn	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]

3.4 Mô hình tuyến tính với Softmax

3.5 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:

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

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

Hàm softmax được định nghĩa:

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

3.6 Dự đoán

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

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

3.7 Hàm mục tiêu và tối ưu

3.8 Hàm hợp lý

Xác suất của tập nhãn:

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

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.

3.10 Ước lượng tham số

3.11 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 η là hệ số học.

3.12 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

4.1 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.

4.2 MLP: Mathematical Model

4.3 Forward Pass

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

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

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

4.4 Output Layer

Regression:

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

Classification (Softmax):

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

4.5 Function Composition View

An MLP is a composition of functions:

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

More layers imply higher representation power.

4.6 MLP: Layers

4.7 Fully Connected (Linear) Layer

For a single sample:

$$y = Wx + b \quad (26)$$

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

$$Y = XW^\top + \mathbf{1}b^\top \quad (27)$$

4.8 Activation Functions

Sigmoid

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

Tanh

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

ReLU

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

Leaky ReLU

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

SiLU (Swish)

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

5 Training ANN

5.1 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:

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

5.1.1 Regression Losses

Mean Squared Error (MSE)

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

MSE penalizes large errors strongly but is sensitive to outliers.

Mean Absolute Error (MAE)

$$L_{\text{MAE}} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|.$$

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 $e_i = y_i - \hat{y}_i$.

5.1.2 Classification Losses

Binary Cross-Entropy (BCE) For $y_i \in \{0, 1\}$ and $p_i = \sigma(z_i)$:

$$L_{\text{BCE}} = -\frac{1}{n} \sum_{i=1}^n [y_i \log p_i + (1 - y_i) \log(1 - p_i)].$$

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

$$L_{\text{CE}} = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{ik} \log p_{ik}.$$

5.2 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.

5.2.1 Stochastic Gradient Descent

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L.$$

5.2.2 Training Algorithm

Algorithm 1 SGD Training Procedure

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

5.3 Optimization Methods

5.3.1 SGD with Momentum

$$v_t = \mu v_{t-1} + g_t, \quad \theta \leftarrow \theta - \eta v_t.$$

5.3.2 Adam

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1)g_t, \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2)g_t^2, \\ \theta &\leftarrow \theta - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \end{aligned}$$

5.3.3 AdamW

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

5.4 Training Techniques

5.4.1 Learning Rate Scheduling

Common strategies include step decay, cosine annealing, and warm restarts.

5.4.2 Regularization

- L2 weight decay
- Dropout
- Early stopping

5.4.3 Normalization

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

5.5 Practical Considerations

5.5.1 Initialization

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

5.5.2 Gradient Issues

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

6 SVM PRIMAL PROBLEM

6.1 Giới thiệu bài toán

6.2 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:

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}, \quad t_n \in \{-1, +1\}$$

6.3 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.

6.4 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.

6.5 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.]])
```

6.6 Phương pháp xây dựng bộ phân lớp

6.7 Nguyên tắc

Quy trình gồm ba bước chính:

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

6.8 Bài toán gốc (Primal Problem)

6.9 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:

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

6.10 Hàm quyết định

Hàm quyết định:

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

Quy tắc phân lớp:

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

6.11 Lề (Margin)

Lề trên tập huấn luyện:

$$m_{\mathbf{w}} = \min_n \frac{t_n(\mathbf{w}^T \mathbf{x}_n + b)}{\|\mathbf{w}\|}$$

6.12 Cực đại lề

Có thể chuẩn hóa sao cho:

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

6.13 Hàm mục tiêu

Cực đại lề tương đương với bài toán:

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

với ràng buộc:

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

6.14 Bài toán gốc

$$\begin{aligned} \mathbf{w}^*, b^* &= \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t. } &t_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1, \quad \forall n \end{aligned}$$

6.15 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

7.1 Bài toán đối ngẫu

7.2 Hàm Lagrangian

Hàm Lagrangian của bài toán (??) là:

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^N \alpha_n [t_n(w^T x_n + b) - 1] \quad \begin{aligned} &\text{với } \alpha_n \geq 0, \quad n = 1, \dots, N. \\ &\text{đó:} \end{aligned}$$

7.3 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:

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

7.4 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:

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

7.5 Bài toán đối ngẫu

Bài toán đối ngẫu tương đương:

$$\begin{aligned} \alpha^* &= \arg \min_{\alpha} \frac{1}{2} \alpha^T K \alpha - \mathbf{1}^T \alpha \\ \text{s.t. } &\alpha_n \geq 0, \quad n = 1, \dots, N, \\ &\sum_{n=1}^N \alpha_n t_n = 0, \end{aligned} \quad (38)$$

trong đó

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

7.6 Tiêu chuẩn Slater

Vì tồn tại (w, b) sao cho

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

hàm Lagrangian của 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.

7.7 Công thức dự báo

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

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

Nhận dự báo:

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

8 SVM SOFT MARGIN

8.1 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.

8.2 Nguyên tắc Soft Margin

Ý tưởng chính:

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

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

8.3 Bài toán gốc với lè mềm

8.4 Ràng buộc mới

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

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

8.5 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.

8.6 Bài toán tối ưu

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

8.7 Bài toán đối ngẫu

8.8 Hàm Lagrangian

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

8.9 Điều kiện KKT

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

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

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

8.10 Bài toán đối ngẫu

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

$$\sum_{n=1}^N \alpha_n t_n = 0. \quad (48)$$

8.11 Công thức dự báo

Sau khi tìm được α 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:

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

8.12 Cài đặt với CVXOPT

Bài toán đối ngẫu có dạng chuẩn:

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

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

9 SVM KERNEL

9.1 SVM hai lớp với lè mềm

Xét tập huấn luyện $\{(x_i, t_i)\}_{i=1}^N$ với $t_i \in \{-1, +1\}$. Bài toán đối ngẫu của SVM lè mềm được viết dưới dạng:

$$\alpha^* = \arg \min_{\alpha} \frac{1}{2} \alpha^T K \alpha + p^T \alpha \quad (51)$$

với các ràng buộc:

$$G \alpha \leq h, \quad A \alpha = b \quad (52)$$

Trong đó, ma trận kernel K được xác định bởi tích vô hướng giữa các điểm dữ liệu.

9.2 Dự báo

Giá trị bias b được ước lượng bởi:

$$b = \frac{1}{N_M} (t_M - K_{MS} [\alpha_S \odot t_S])^T \mathbf{1} \quad (53)$$

Hàm dự báo:

$$y = K_{BS} [\alpha_S \odot t_S] + b \quad (54)$$

$$\text{label} = \text{sign}(y) \quad (55)$$

9.3 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:

$$\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$$

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.

9.4 Phương pháp Kernel

Phương pháp kernel cho phép tính:

$$\langle \Phi(x_i), \Phi(x_j) \rangle$$

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

$$k(x_i, x_j)$$

9.5 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:

$$\sum_{i=1}^N \sum_{j=1}^N c_i c_j k(x_i, x_j) \geq 0$$

9.6 Huấn luyện và dự báo với Kernel

Ma trận Gram kernel:

$$K_{Gram} = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_N) \\ \vdots & \ddots & \vdots \\ k(x_N, x_1) & \cdots & k(x_N, x_N) \end{bmatrix} \quad (56)$$

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

9.7 Các kernel thông dụng

Một số kernel phổ biến trong thực tế:

- Linear:

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

- Polynomial:

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

- RBF (Gaussian):

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

- Sigmoid:

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

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

9.8 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

9.9 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

10.1 Mô tả bài toán

Giả sử tập dữ liệu đầu vào:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{bmatrix}$$

trong đó:

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

Mục tiêu:

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

10.2 Kiến thức toán học liên quan

10.3 Phương sai và hiệp phương sai

Trung bình của dữ liệu:

$$\mu = \frac{1}{N} \sum_{n=1}^N x_n$$

Dữ liệu được chuẩn hóa:

$$z_n = x_n - \mu$$

Ma trận hiệp phương sai:

$$S = \frac{1}{N} \sum_{n=1}^N (x_n - \mu)(x_n - \mu)^T$$

10.4 Eigenvalue và Eigenvector

Với ma trận vuông A , bài toán eigen:

$$Au = \lambda u$$

trong đó u là eigenvector và λ là eigenvalue.

10.5 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.

10.6 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:

$$\max_u \quad u^T S u \quad \text{s.t.} \quad u^T u = 1$$

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.

10.7 Thu giảm số chiều

Chọn M eigenvector tương ứng với M eigenvalue lớn nhất, tạo thành ma trận:

$$\hat{U} = [u_1, u_2, \dots, u_M]$$

Chiều dữ liệu:

$$X_{\text{PCA}} = (X - \mu^T) \hat{U}$$

Phục hồi xấp xỉ:

$$\hat{X} = \mu^T + X_{\text{PCA}} \hat{U}^T$$

10.8 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_)
```

10.9 Singular Value Decomposition (SVD)

Phân rã SVD:

$$X = USV^T$$

- 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.

10.10 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

11.1 Giới thiệu về LDA

11.2 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 .

11.3 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.

11.4 LDA qua API

Ví dụ sử dụng Scikit-learn:

```
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as Lda
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

11.5 Bài toán tối ưu

11.6 Tâm của mỗi lớp

Với lớp k :

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

11.7 Between-class scatter matrix

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

11.8 Within-class scatter matrix

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

11.9 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:

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

11.10 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:

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

11.11 Trường hợp có C lớp

11.12 Hàm mục tiêu tổng quát

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

11.13 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 $C - 1$.

11.14 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

12.1 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

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

12.2 Bagging (Bootstrap Aggregating)

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

12.3 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:

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

12.4 Properties

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

12.5 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.

12.6 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_{sub} .

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

12.7 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.

12.8 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 α_t is determined by the weighted classification error of h_t .

12.9 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.

12.10 Voting and Stacking

12.11 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.

12.12 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.

12.13 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 ALGORITHMS

13.1 Key Components of Genetic Algorithms

13.2 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)

13.3 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.

13.4 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.

13.5 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)

13.6 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

13.7 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.

13.8 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 $x = 15$.

13.9 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.

13.10 Advantages and Disadvantages

13.11 Advantages

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

13.12 Disadvantages

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

13.13 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

13.14 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.