

# MLCheatsheet

## 1. Bài toán Gốc: Toán học (Theory)

- Thiết lập:**  $x, w \in \mathbb{R}^{M-1}$ ; Siêu phẳng:  $w^T x + b = 0$ .
- Dẫn dắt:** Max Margin  $\Leftrightarrow \text{Min } \|w\| \Leftrightarrow \text{Min } \frac{1}{2} \|w\|^2$ .
- Công thức Primal:**

$$\begin{aligned} \min_{w,b} \frac{1}{2} \|w\|^2 \\ \text{s.t. } t_n(w^T x_n + b) \geq 1, \forall n \end{aligned}$$

## 2. Bài toán Gốc: Implementation (CVXOPT)

Solver:  $\min \frac{1}{2} u^T P u + q^T u$  s.t.  $Gu \leq h$ . Biến:  $u = [\mathbf{w}; \mathbf{b}]_{(M \times 1)}$ .

Mapping:

$$\begin{aligned} P &= \text{diag}(1, \dots, 1, 0)_{(M \times M)} \\ q &= \mathbf{0}_{(M \times 1)} \\ G &= -[\text{diag}(\mathbf{t}) \cdot \mathbf{X}, \mathbf{t}]_{(N \times M)} \\ h &= -\mathbf{1}_{(N \times 1)} \end{aligned}$$

## 3. Lagrangian & Điều kiện KKT

1. **Hàm Lagrangian:**

$$\begin{aligned} \mathcal{L} &= \frac{1}{2} \|w\|^2 + \sum_{n=1}^N \underbrace{\alpha_n}_{\substack{\text{dùng: } \leq 0 \\ \text{sai: } > 0}} \{1 - t_n(w^T x_n + b)\} \\ &= \frac{1}{2} \|w\|^2 + E_{\text{data}} \text{ (cost trên dữ liệu)} \end{aligned}$$

Tính chất: Với  $\alpha_n \geq 0$ ,  $(w^*, b^*)$  tối ưu:

$$\mathcal{L}(w^*, b^*, \alpha) \leq \frac{1}{2} \|w^*\|^2$$

2. Tính đạo hàm:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= w - \sum_{n=1}^N \alpha_n t_n x_n \\ \frac{\partial \mathcal{L}}{\partial b} &= -\sum_{n=1}^N \alpha_n t_n \end{aligned}$$

3. Điều kiện dừng (KKT-1):

- $\nabla_w \mathcal{L} = 0 \Rightarrow w = \sum_{n=1}^N \alpha_n t_n x_n$  (1)

- $\nabla_b \mathcal{L} = 0 \Rightarrow \sum_{n=1}^N \alpha_n t_n = 0$  (2)

4. Các điều kiện KKT khác:

- (KKT-2) Ràng buộc gốc:  $1 - t_n(w^T x_n + b) \leq 0$
- (KKT-3) Ràng buộc dual:  $\alpha_n \geq 0, \forall n$
- (KKT-4) Điều kiện bù:  $\alpha_n \{1 - t_n(w^T x_n + b)\} = 0$

Ý nghĩa: Thỏa KKT  $\Rightarrow (w, b, \alpha) = (w^*, b^*, \alpha^*)$

5. Tiêu chuẩn Slater (cho SVM):

Nếu phần trong của tập khả thi không rỗng thì:

$$\exists (w, b) \text{ s.t. } 1 - t_i(w^T x_i + b) < 0, \forall i$$

$\Rightarrow$  strong duality ( $p^* = d^*$ )  $\Leftrightarrow$  duality gap = 0

## 4. Bài toán Dual: Biến đổi (Math)

Thay (1), (2) vào  $\frac{1}{2} w^T w$ :

A. **Thay  $w$  vào  $\frac{1}{2} w^T w$ :**

$$\frac{1}{2} \left( \sum \alpha_i t_i x_i \right)^T \left( \sum \alpha_j t_j x_j \right) = \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j (x_i^T x_j) (*)$$

B. **Thay  $w$  vào  $-\sum \alpha_n t_n w^T x_n$ :**

$$-\sum_n \alpha_n t_n \underbrace{\left( \sum_m \alpha_m t_m x_m \right)^T}_{w^T} x_n = -2 \times (*)$$

C. **Kết quả ( $A + B = -A$ ):**

$$\begin{aligned} \max g(\alpha) &= \sum_n \alpha_n - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j (x_i^T x_j) \\ \text{s.t. } \alpha_n &\geq 0; \quad \sum \alpha_n t_n = 0 (g(\alpha) \text{ luôn là hàm lồi}) \end{aligned}$$

## 5. Bài toán Dual: Implementation (CVXOPT)

Solver:  $\min \frac{1}{2} \alpha^T P \alpha + q^T \alpha$  s.t.  $G \alpha \leq h, A \alpha = b$ . Biến:  $\alpha \in \mathbb{R}^N$ .

Mapping:

$$\begin{aligned} P &= \mathbf{K}_{\text{Gram}}_{(N \times N)} \quad (K_{ij} = t_i t_j x_i^T x_j) \\ q &= -\mathbf{1}_{(N \times 1)} \quad (\text{Max } \Sigma \rightarrow \text{Min } -\Sigma) \\ G &= -I_{(N \times N)}; \quad h = \mathbf{0}_{(N \times 1)} \\ A &= t^T; \quad b = 0 \end{aligned}$$

## 6. Soft Margin & Kernel

1. **Soft Margin (ý tưởng):** Cho phép một số điểm vi phạm margin bằng biến slack  $\xi_n$ , đổi lại bị phạt trong hàm mục tiêu.

2. **Bài toán gốc (Primal – soft margin):**

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

Ý nghĩa: C: C lớn  $\Rightarrow$  phạt mạnh (ít sai, lề hẹp); C nhỏ  $\Rightarrow$  cho sai nhiều (lề rộng).

3. **Dạng đối ngẫu (Dual):**

$$\begin{aligned} \mathcal{L} &= \frac{1}{2} \|w\|^2 + C \sum_n \xi_n - \sum_n \alpha_n \{t_n(w^T x_n + b) - 1 + \xi_n\} - \sum_n \mu_n \xi_n \\ &\max_{\alpha} \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j x_i^T x_j \\ &\text{s.t. } \boxed{0 \leq \alpha_n \leq C}, \quad \sum_{n=1}^N \alpha_n t_n = 0 \end{aligned}$$

Phân loại theo KKT:

- $\alpha_n = 0$ : điểm ngoài margin (không support)
- $0 < \alpha_n < C$ : support vector trên margin
- $\alpha_n = C$ : support vector vi phạm / phân loại sai

4. **Kernel Trick:** Thay thế tích vô hướng trong không gian đặc trưng:

$$\begin{aligned} \langle \phi(x_i), \phi(x_j) \rangle &\longrightarrow k(x_i, x_j) \\ k(x, z) &= \phi(x)^T \phi(z) \end{aligned}$$

Hàm quyết định của SVM:

$$f(x) = \sum_{n \in SV} \alpha_n t_n k(x_n, x) + b, \quad y = \text{sign}(f(x))$$

## Deep Learning

Goal: Multiclass classification ( $y \in \{1, \dots, K\}$ ).

Model (Softmax):

$$p_{ik} = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_i + b_k)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_i + b_j)}$$

Loss (Categorical Cross-Entropy):

$$\mathcal{L}_{\text{CE}} = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{ik} \log p_{ik} = -\frac{1}{n} \sum_{i=1}^n \log p_{i,y_i}$$

Prediction:  $\hat{y}_i = \arg \max_k p_{ik}$

Notes:

- $y_{ik} \in \{0, 1\}$ : nhãn one-hot của mẫu  $i$  tại lớp  $k$
- $p_{i,y_i}$ : xác suất dự đoán của lớp đúng của mẫu  $i$

Vectorized form:

- Vector (single sample): for  $\mathbf{x} \in \mathbb{R}^N$ ,  $\mathbf{W} \in \mathbb{R}^{M \times N}$ ,  $\mathbf{b} \in \mathbb{R}^M$

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^M$$

- Matrix (mini-batch): for  $\mathbf{X} \in \mathbb{R}^{B \times N}$  (rows are samples)

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^T + \mathbf{b} \in \mathbb{R}^{B \times M}$$

(broadcast  $\mathbf{b}$  to all rows)

Activation functions:

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \tanh'(x) = 1 - \tanh^2(x)$$

ReLU

$$\text{ReLU}(x) = \max(0, x), \quad \text{ReLU}'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x \leq 0 \end{cases}$$

Leaky ReLU

$$\text{Leaky ReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x \leq 0 \end{cases}, \quad \text{Leaky ReLU}'(x) = \begin{cases} 1, & x \geq 0 \\ \alpha, & x \leq 0 \end{cases}$$

SiLU (Swish)

$$\text{SiLU}(x) = x \sigma(x), \quad \text{SiLU}'(x) = \sigma(x) + x \sigma(x)(1 - \sigma(x))$$

Linear Regression Solution:

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

Loss (MSE):

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

Closed-form Solution (Normal Equation):

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Hồi quy Ridge:

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \\ \mathbf{w}^* &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

Hồi quy LASSO:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_1$$

$\mathbf{w}^*$  (không có nghiệm dạng đóng, cần giải bằng phương pháp lặp)

Hồi quy Logistic:

Mô hình dự báo:

$$\hat{y}_n = p(y_n = 1 | \mathbf{x}_n, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_n) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}_n)}$$

Hàm mục tiêu (negative log-likelihood / cross-entropy):

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

Gradient của hàm mất mát:

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

$\mathbf{w}^*$  (không có nghiệm dạng đóng, giải bằng phương pháp lặp)

## 1. Convolution Cơ Bản

### 1. Định nghĩa Convolution:

$$Y(u, v) = X * W = \sum_{i=-r}^r \sum_{j=-r}^r X(u-i, v-j)W(i, j)$$

với  $r = \lfloor k/2 \rfloor$  (bán kính kernel)

#### Cross-correlation:

$$Y(u, v) = X * W = \sum_{i=-r}^r \sum_{j=-r}^r X(u+i, v+j)W(i, j)$$

=> Hầu hết thư viện Deep Learning (CNN) gọi là convolution nhưng thực chất đang dùng cross-correlation

#### Kích thước output:

$$o_1 \times o_2 = (i_1 - k_1 + 1) \times (i_2 - k_2 + 1)$$

#### Thuật toán:

1. Xoay kernel 180° → Rot180(W)
2. Flatten kernel thành vector
3. Padding input (nếu cần)
4. Cân chỉnh kernel với góc trên-trái
5. Lấy sub-image, flatten và tính dot-product
6. Trượt kernel theo stride, lặp lại

## 2. Padding & Stride

### 1. Half Padding: Padding size: $p = \lfloor k/2 \rfloor$

Output:  $i_1 \times i_2$  (giữ nguyên kích thước)

### 2. Full Padding: Padding size: $p = k - 1$

Output:  $(i_1 + k_1 - 1) \times (i_2 + k_2 - 1)$

### 3. Non-unit Stride: Với stride $s_1, s_2$ :

$$\text{Output: } \left\lfloor \frac{i_1 + 2p_1 - k_1}{s_1} \right\rfloor + 1 \times \left\lfloor \frac{i_2 + 2p_2 - k_2}{s_2} \right\rfloor + 1$$

### 4. Multi-channel Input: Input $D$ channels → mỗi filter có $D$ channels

$$y_{u,v} = \sum_{d=1}^D \sum_{i,j} W_{i,j}^{(d)} X_{u+i, v+j}^{(d)}$$

### 5. Multi-filters Layer: $K$ filters → output có $K$ channels

## VD: Rotate 180° Kernel

#### Ma trận gốc W:

$$W = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

1. Flip horizontal:  $\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

2. Flip vertical:  $\text{Rot180}(W) = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 2 & 0 & 1 \end{bmatrix}$

## 3. Backprop qua Conv2D

**Forward:**  $Y = X * W$

**Backward - Tính  $\Delta W$ :**

$$\Delta W = \text{Rot180}(\Delta Y) * X$$

- Rot180( $\Delta Y$ ) làm input, no padding
- X làm kernel

**Chi tiết:**

$$\delta w_{ij} = \sum_{u,v} \delta y_{u,v} \cdot x_{u+i, v+j}$$

**Backward - Tính  $\Delta X$ :**

$$\Delta X = \text{Rot180}(W) * \Delta Y$$

- Rot180(W) làm input, full padding
- $\Delta Y$  làm kernel

**Chi tiết:**

$$\delta x_{uv} = \sum_{i,j} \delta y_{ij} \cdot w_{u-i, v-j}$$

## 4. Matrix Form Convolution

#### Ký hiệu:

- $\mathbb{X}_{conv}$ : matrix từ im2col (X thành columns)
- $\mathbb{W}_{flat}$ : flatten kernel thành row
- $\mathbb{W}_{conv}$ : kernel2row matrix

**Forward:**

$$\begin{aligned} X * W &= \text{Reshape}_{o_1 \times o_2}(\mathbb{W}_{flat} \times \mathbb{X}_{conv}) \\ &= \text{Reshape}_{o_1 \times o_2}(\mathbb{W}_{conv} \times \mathbb{X}_{flat}) \end{aligned}$$

**Backward -  $\Delta W$ :**

$$\Delta W = \text{Reshape}_{k_1 \times k_2}(\Delta Y_{flat} \times \mathbb{X}_{conv}^T)$$

**Backward -  $\Delta X$ :**

$$\Delta X = \text{Reshape}_{i_1 \times i_2}(\mathbb{W}_{conv}^T \times \Delta Y_{flat})$$

**Ưu điểm:** Tận dụng phép nhân ma trận tối ưu

## 5. Fully-Connected Layer

**Forward:**

$$y = Wx + b$$

với  $W \in \mathbb{R}^{N \times M}$ ,  $x \in \mathbb{R}^M$ ,  $b \in \mathbb{R}^N$

**Số tham số:**  $M \times N + N$

**Backward -  $\Delta W$ :**

$$\Delta W = \Delta y \cdot x^T$$

$$\delta w_{n,m} = \delta y_n \cdot x_m$$

**Backward -  $\Delta b$ :**

$$\Delta b = \Delta y$$

**Backward -  $\Delta X$ :**

$$\Delta X = W^T \cdot \Delta y$$

$$\delta x_m = \sum_{n=1}^N \delta y_n \cdot w_{n,m}$$

**Update trọng số:**

$$W \leftarrow W - \alpha \Delta W, \quad b \leftarrow b - \alpha \Delta b$$

## 6. Conv vs FC

#### Fully-Connected:

- Dense connectivity: mỗi output nối với mọi input
- Mỗi output có bộ weights riêng
- Số tham số:  $(i_1 \times i_2) \times (o_1 \times o_2) + (o_1 \times o_2)$
- VD: input  $3 \times 3$ , output  $2 \times 2 \rightarrow 9 \times 4 + 4 = 40$  params

#### Convolution:

- Sparse connectivity: output chỉ nói vùng local
- Parameter sharing: tất cả output dùng chung weights
- Số tham số:  $k_1 \times k_2 \times D + 1$  (với D channels)
- VD: kernel  $2 \times 2 \rightarrow 4 + 1 = 5$  params

**Ưu điểm Conv:** Giảm tham số, học local patterns, translation invariant

## 7. Pooling Layer

#### Mục đích:

- Downsampling feature maps
- Loại bỏ redundant features
- Giảm overfitting

#### Max Pooling:

$$y_{u,v} = \max_{(i,j) \in \text{window}} x_{i,j}$$

#### Average Pooling:

$$y_{u,v} = \frac{1}{k_1 \times k_2} \sum_{(i,j) \in \text{window}} x_{i,j}$$

#### Output size (stride = 1):

$$(i_1 - k_1 + 1) \times (i_2 - k_2 + 1)$$

#### Output size (stride = s):

$$\left\lfloor \frac{i_1 - k_1}{s_1} \right\rfloor + 1 \times \left\lfloor \frac{i_2 - k_2}{s_2} \right\rfloor + 1$$

**Thường dùng:** window  $2 \times 2$  hoặc  $3 \times 3$ , stride = 2

## 8. Gradient Descent

**Forward Pass:** Tính output từ input qua các layers

**Backward Pass (Backpropagation):** Tính gradients theo chain rule:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial w}$$

#### Batch Gradient Descent:

$$\Delta w = \frac{1}{m} \sum_{i=1}^m \Delta w^{(i)}$$

$$w \leftarrow w - \alpha \Delta w$$

với  $\alpha$  là learning rate,  $m$  là batch size

#### Quy trình:

1. Forward qua tất cả samples trong batch
2. Backward tính gradients cho từng sample
3. Trung bình gradients
4. Update weights

## 1. Giới thiệu & Mục tiêu PCA

### 1. Đầu vào:

- Ma trận dữ liệu  $X$  có kích thước  $N \times D$  ( $N$  mẫu,  $D$  chiều). Thường  $D$  rất lớn.
- Dộ phức tạp tính toán tăng lên: ma trận Kgram, kernel, hiệp phương sai, khoảng cách giữa các điểm dữ liệu, số nơron lớp ẩn tăng. Và cần mô hình phức tạp hơn, điểm dữ liệu lớn hơn dễ gây overfit.

### 2. Mục tiêu:

- Giảm số chiều  $D \rightarrow M$  sao cho  $M \ll D$ .
- Dặc trưng mới không tương quan (uncorrelated).
- Giữ lại thông tin quan trọng nhất (Variance lớn nhất).

## 2. Kiến thức Toán nền tảng

### 1. Hình chiếu: Chiếu vector $x$ lên vector $u$ : $l = \frac{u^T x}{\|u\|}$ .

### 2. Dạng bậc hai & Đạo hàm:

- $u = [u_1, u_2, \dots, u_N]^T$ ,  $v = [v_1, v_2, \dots, v_M]^T$ .
- $A = [a_{ij}]_{N \times M}$ .
- $u^T A v = \sum_{i=1}^N \sum_{j=1}^M a_{ij} u_i^T v_j$ . ( $a_{ij}$  và  $u_i^T v_j$  là vô hướng).
- $\frac{\partial u^T A u}{\partial u} = 2Au$  (với  $A$  đối xứng).

### 3. Eigenvectors & Eigenvalues:

$$Au = \lambda u \Leftrightarrow (A - \lambda I)u = 0 \quad (3.7)$$

$$\Rightarrow \det(A - \lambda I) = 0 \quad (3.8)$$

Giải 3.8 để tìm  $\lambda_i$  (có tối đa  $N$  eigenvalues), thay vào 3.7 để tìm  $u_i$ .

### 4. Tính chất eigenvectors và eigenvalues $S$ :

- Ma trận  $A$  là đối xứng chứa các số thực.
- Có  $N$  eigenvector trực giao và  $N$  eigenvalue thực không âm.
- Hang của  $A$  bằng số eigenvalue lớn hơn 0.
- Định thức của  $A$  là tích các eigenvalue. Trace của  $A$  là tổng các eigenvalue.
- Khi hạng của  $A$  là  $N$ , ma trận  $A^{-1}$  có  $N$  eigenvalue  $= 1/\lambda_i$ . ( $i = 1, \dots, N$ )

## 3. Bài toán PCA

### 1. Hai quan điểm về PCA:

- PCA chuyển dữ liệu đầu vào là  $N$  điểm thuộc  $D$  chiều sang không gian con có kích thước  $M$  chiều ( $M < D$ ) sao cho:
- Khi chiếu dữ liệu lên không gian con này, phương sai của dữ liệu là lớn nhất:  $\text{argmax}_u \frac{1}{N} \sum_{k=1}^N x_{u,k}^2$
- Hoặc khi chiếu dữ liệu lên không gian con này, khoảng cách từ điểm dữ liệu đến không gian con là nhỏ nhất:  $\text{argmin}_u \sum_{k=1}^N d_{u,k}^2$

### 2. Cực đại hóa phương sai:

- Tìm vector  $u$  đầu tiên cho ra phương sai lớn nhất. (Tối ưu có ràng buộc và dùng Lagrangian để giải)
- Giả sử đã tìm ra  $M$  vector đầu tiên: Các vector đều có chiều dài 1 và trực giao nhau đôi một.
- Tìm vector thứ  $M+1$  sao cho trực giao với  $M$  vector trước và có phương sai lớn nhất.

## Công thức tính định thức ma trận

### Ma trận $2 \times 2$ :

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

### Ma trận $3 \times 3$ (quy tắc Sarrus):

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

### a. Khi $k = 1$ :

- Mục tiêu: Tìm  $u$  để phương sai của dữ liệu chiếu lên  $u$  là lớn nhất.
- Điểm trung tâm của dữ liệu:  $\mu = \frac{1}{N} \sum_{n=1}^N x_n$ .  $z_n = x_n - \mu$  là điểm tương ứng của  $x_n$  sau khi dịch về  $\mu$ .
- Variance của dữ liệu chiếu lên  $u$ :  $\sigma^2 = \frac{1}{N} \sum_{n=1}^N (u^T z_n)^2 = \frac{1}{N} \sum_{n=1}^N (u^T z_n)(z_n^T u) = \frac{1}{N} \sum_{n=1}^N u^T [(x_n - \mu)(x_n - \mu)^T] u = u^T S u$ .
- Covariance matrix (ma trận hiệp phương sai):  $S = \frac{1}{N} \sum_{n=1}^N (x_n - \mu)(x_n - \mu)^T = Z^T Z = (X - \mu)^T (X - \mu)$ .  $S$  là ma trận vuông, đối xứng  $D \times D$ , bán định dương.
- Bài toán tối ưu:  $\text{argmax}_u u^T S u$  với ràng buộc  $u^T u = 1$ . ( $S$  bán định dương  $\Rightarrow$  hàm mục tiêu lồi). Giải bằng pp nhân tử Lagrange.
- Lagrangian:  $\mathcal{L}(u, \lambda) = u^T S u - \lambda(u^T u - 1)$ . Đạo hàm theo  $u$  và đặt bằng 0:  $\frac{\partial \mathcal{L}}{\partial u} = 0 \Rightarrow S u - \lambda u = 0 \Rightarrow S u = \lambda u$ .
- Bài toán trở thành bài toán tìm eigenvector và eigenvalue của ma trận  $S$ :  $S u_1 = \lambda_1 u_1$ . Nghiệm  $u_1$  là eigenvector cho variance lớn nhất  $\lambda_1$ .
- Tính chất của eigenvalues và eigenvectors của  $S$ :  $\lambda_1 = u_1^T S u_1$  (phương sai lớn nhất).

### b. Khi $k = M$ :

- Bài toán tối ưu:  $u_M = \text{argmax}_u u^T S u$  với ràng buộc  $u^T u = 1$  và  $u^T u_k = 0$  với  $k = 1, 2, \dots, (M-1)$ .
- Mục tiêu: Tìm vector  $u_M$  sao cho trực giao với tất cả các vector trước và khi chiếu dữ liệu lên hướng này có phương sai lớn nhất (Vẫn  $\leq \lambda_1, \lambda_2, \dots, \lambda_{M-1}$ ). Độ lớn của  $u_M$  là 1.
- Giả thiết: Đã tìm được cặp  $u_k, \lambda_k$  với  $k = 2, \dots, M$  từ bài toán tối ưu ở trên.

### c. Khi $k = M+1$ :

- Bài toán tối ưu và mục tiêu: Tương tự như trên nhưng thay  $M$  bằng  $M+1$  và  $M-1$  bằng  $M$ . Cần suy ra bài toán tối ưu ở đây có nghiệm.
- Lagrangian:  $\mathcal{L}(u, \lambda, \mu) = u^T S u - \lambda(u^T u - 1) - \sum_{k=1}^M \mu_k (u^T u_k)$ .
- Đạo hàm theo  $u$ :  $\frac{\partial \mathcal{L}}{\partial u} = 2Su - 2\lambda u - \sum_{k=1}^M \mu_k u_k$ .
- Điểm làm lagrangian cực tiểu là nghiệm khi đạo hàm bằng 0:  $2Su - 2\lambda u - \sum_{k=1}^M \mu_k u_k = 0$ .
- Ở phương trình này,  $\mu_k = 0$  vì  $u$  trực giao với tất cả  $u_k$ . Vậy ta nhân hai vế với  $u_k$  và có lại phương trình  $Su = \lambda u$ .

### 2. Giải thuật thu giảm số chiều:

- Tính ma trận hiệp phương sai và tìm các cặp  $u_k, \lambda_k$  (numpy.linalg.eig, numpy.linalg.eigh).
- Chọn  $M$  eigenvector  $e_1, e_2, \dots, e_M$ . Chọn  $M$  bằng kiểm thử chéo.
- Chiếu dữ liệu gốc lên  $M$  vector để thu dữ liệu mới.

## 3. Singular Value Decomposition (SVD):

- SVD là kĩ thuật tổng quát của PCA.
- PCA: Tính ma trận hiệp phương sai  $S$  rồi phân rã eigen trên  $S$ .
- SVD: Phân rã SVD trên chính ma trận gốc ( $X$ ), quá trình phân rã ổn định hơn.
- Ma trận  $X$  được SVD phân rã thành  $X = USV^T$ . Với  $U(N \times N)$ : có cột là eigenvectors của  $XX^T$ .  $V(D \times D)$ : có cột là eigenvectors của  $X^T X$ .  $S(N \times D)$ : ma trận có đường chéo chính là singular values xếp từ lớn đến nhỏ.

### Giải thuật

- Tính  $Z = X - m^T$  với  $m$  là total mean =  $\frac{1}{N} \sum_{n=1}^N x_n$ .
- Dùng SVD phân rã  $Z = USV^T$  (numpy.linalg.svd).
- Chọn  $M$  vectors đầu tiên của  $V$  tương ứng  $M$  singular values lớn nhất tạo  $\hat{V}$ . (Singular Value là căn bậc 2 của eigenvalue)
- Chiếu dữ liệu  $Z$  lên  $M$  eigenvectors:  $X_{pca} = Z\hat{V}$ .

## 4. Tổng kết

### PCA:

- $A = UDU^T = \sum_{i=1}^r \lambda_i u_i u_i^T$ .
- $A$ : ma trận hiệp phương sai.
- $r$ : hạng của  $A$ .
- Thu giảm chiều: Chỉ giữ lại  $M$  eigenvector và chiếu  $X$  lên các vector này.

### SVD:

- $A = USV^T = \sum_{i=1}^r s_i u_i v_i^T$ .
- $A$ : ma trận bất kỳ.
- $r$ : hạng của  $A$ .
- Thu giảm chiều: Chỉ giữ lại  $M$  eigenvector và xấp xỉ  $X$  lên các vector này.

## 5. PCA qua API

- import numpy as np
- from sklearn.decomposition import PCA
- $X = np.array([-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2])$
- pca = PCA(n\_components=2)
- pca.fit(X)
- print(pca.explained\_variance\_ratio\_)
- print(pca.singular\_values\_)

## 1. Giới thiệu về LDA

### 1. Đầu vào:

- $X$  có kích thước  $N \times D$ ,  $D$  rất lớn.
- $t_k$  là nhãn lớp, có  $C$  lớp khác nhau.

### 2. Mục tiêu:

- Giảm số chiều từ  $D$  xuống  $M$  và  $M \leq C - 1$ .
- Dữ liệu sau khi giảm có tính phân tách cao nhất.

## 2. LDA qua API (Scikit-learn)

```
1. from sklearn import datasets  
2. from sklearn.discriminant_analysis import LinearDiscriminantAnalysis  
3. iris = datasets.load_iris()  
4. X = iris.data  
5. y = iris.target  
6. target_names = iris.target_names  
7. lda = LinearDiscriminantAnalysis(n_components=2)  
8. X_r2 = lda.fit(X, y).transform(X)
```

## 3. Bài toán tối ưu

### 1. Các thành phần:

- Tâm của dữ liệu thuộc mỗi lớp:  $m_1 = \frac{1}{N_1} \sum_{n \in C_1} x_n$ ,  $m_2 = \frac{1}{N_2} \sum_{n \in C_2} x_n$
- Khoảng cách giữa 2 tâm khi chiếu lên vector  $w$ :  $d_{12} = w^T S_B w$
- Between-class covariance matrix:  $S_B = (m_2 - m_1)(m_1 - m_2)^T$
- Variance của lớp 1 sau khi chiếu lên  $w$ :  $\sigma_1^2 = w^T S_{W1} w$
- Within-class covariance matrix:  $S_{W1} = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T$
- Within-class covariance matrix của cả 2 lớp:  $S_W = S_{W1} + S_{W2}$
- Within-class variance:  $\sigma^2 = w^T S_W w$

### 2. Hàm mục tiêu (Quan điểm của Fisher):

Tìm hướng  $w^* = \arg \max_w J(w)$  để cực đại hóa tỷ số:

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

### 3. Tìm nghiệm:

- Đạo hàm  $\nabla_w J = 0 \Leftrightarrow S_B w = \lambda S_W w$ ,  $\lambda = \frac{w^T S_B w}{w^T S_W w}$
- $\Rightarrow$  Phương trình eigenvector:  $S_W^{-1} S_B w = \lambda w$
- Chúng ta chỉ cần hướng của  $w$ , nên dùng  $S_B w = (m_2 - m_1)(m_2 - m_1)^T w$
- Hướng của vector  $w$  là  $S_W^{-1}(m_2 - m_1)$ ; Cần chuẩn hóa  $w$  để có độ dài đơn vị.

## 4. Trường hợp có C lớp

===== Phần này chưa làm xong =====

### 1. Thiết lập tham số:

- Số lớp:  $C (C > 2)$ .
- Vector trung bình lớp  $k$ :  $m_k$ .
- Vector trung bình toàn cục:  $m = \frac{1}{N} \sum_{k=1}^C N_k m_k$ .

### 2. Xây dựng các Ma trận Scatter:

- Within-class ( $S_W$ ):** Tổng độ phân tán trong từng lớp.

$$S_W = \sum_{k=1}^C S_k$$

(với  $S_k$  là ma trận scatter của lớp  $k$ ).

- Between-class ( $S_B$ ):** Độ phân tán của các tâm lớp so với tâm chung.

$$S_B = \sum_{k=1}^C N_k (m_k - m)(m_k - m)^T$$

### 3. Hàm mục tiêu (Objective Function):

Tìm ma trận chiếu  $W$  để tối đa hóa tỷ số định thức:

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

### 4. Giải pháp và Kết quả:

- Bài toán trị riêng:  $S_B w_i = \lambda_i S_W w_i$  (hoặc  $S_W^{-1} S_B w_i = \lambda_i w_i$ ).
- $W$  được tạo thành từ các **eigenvectors** ứng với các **eigenvalues** lớn nhất.
- Số chiều tối đa:**  $K \leq C - 1$  (vì hạng của  $S_B$  tối đa là  $C - 1$ ).