1. SVM (Support Vector Machine) 的訓練過程

SVM 的核心目標是找到一個**最大邊界超平面**，將不同類別的數據點分開。這是一個**二次規劃 (Quadratic Programming)** 問題。

找出最佳的 W 和 b 的數學推導

A. 原始優化問題 (Primal Problem)

對於線性可分 (Hard Margin) 的數據，目標是最大化邊界 $\frac{2}{||w||}$，等價於最小化 $\frac{1}{2}||\mathrm{w}||^2$。

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

$$\text{s.t.} \quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \quad \text{for } i = 1, \ldots, m$$

其中 $y^{(i)} \in \{-1, 1\}$ 是類別標籤

B. 轉換為對偶問題 (Dual Problem)

為了解決帶有不等式約束的優化問題，我們使用**拉格朗日乘數法**引入乘子 $a_i \geq 0$。

拉格朗日函數：

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i \left[1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right]$$

求解 w 的條件：

對 w 求偏導並令其為零，得出 w 的表達式：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \implies \mathbf{w}^* = \sum_{i=1}^{m} \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

求解 b 的條件：

對 b 求偏導並令其為零，得出 α 的約束：

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \implies \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

對偶目標函數 (Dual Objective Function)：

將 w* 和 b 的條件代回 L，形成一個只關於 α 的優化問題：

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \quad \text{and} \quad \alpha_i \geq 0$$

C. 結論

1. 通過求解 α 的二次規劃問題（通常使用 **SMO 算法**），得到最佳乘子 α*。
2. 最佳權重 w*：由支持向量（即 αi*>0 的樣本）計算得到：

$$\mathbf{w}^* = \sum_{i=1}^{m} \alpha_i^* y^{(i)} \mathbf{x}^{(i)}$$

3. **最佳偏置 b***：由任一支持向量x_k滿足邊界條件 $y^k(w^{*T}x^k + \mathrm{b}) = 1$ 解出。

2. MLP (Multi-Layer Perceptron) 的訓練過程

MLP 的訓練是通過**反向傳播算法 (Backpropagation)** 結合**梯度下降法 (Gradient Descent)** 來迭代調整權重 $\mathbf{W}$ 和偏置 b。

找出最佳的 W 的數學推導

A. 最小化目標函數

MLP 訓練的目標是最小化**損失函數 (Loss Function)** J(W,b)。

損失函數範例 (均方誤差 MSE)：

$$J(\mathbf{W}, b) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$$

其中 y^ 是網絡的預測輸出。

目標： 找到 W* 和 b* 使得 J(W,b) 最小。

B. 梯度計算：反向傳播 (Backpropagation)

使用**鏈式法則**計算損失函數 J 對於每一層權重 w^i 的梯度 $\frac{\partial J}{\partial w^i}$。

1. 輸出層誤差 δ(L)：

   $$\boldsymbol{\delta}^{(L)} = \frac{\partial J}{\partial \mathbf{z}^{(L)}} = (\hat{\mathbf{y}} - \mathbf{y}) \odot g'(\mathbf{z}^{(L)})$$

   （以 MSE 和 Sigmoid 輸出為例， z(L) 是輸出層的加權和， g′ 是激活函數的導數）。

2. 隱藏層誤差 δ(l) 的遞歸：

   誤差從後一層向前傳播：

$$\boldsymbol{\delta}^{(l)} = \left((\mathbf{W}^{(l)})^T \boldsymbol{\delta}^{(l+1)} \right) \odot g'(\mathbf{z}^{(l)})$$

權重梯度 ∂W(l)∂J：

利用誤差項和前一層的激活值 a(l) 計算梯度：

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \frac{1}{m} \boldsymbol{\delta}^{(l+1)} (\mathbf{a}^{(l)})^T$$

C. 權重更新 (梯度下降)

計算出梯度後，利用梯度下降法迭代更新權重，以逼近最佳解 W*：

權重更新規則：

$$\mathbf{W}^* = \mathbf{W} - \eta \cdot \frac{\partial J}{\partial \mathbf{W}}$$

程式中如何實現: $W^* = \mathbf{W} + \Delta\mathbf{W}$

無論是 MLP 的梯度下降，還是 SVM 的 SGD 變體，程式中權重的迭代更新都遵循以下模式：

$$W^* = \mathbf{W} + \Delta\mathbf{W}$$

概念	**數學表達式**	**程式實現 (偽代碼)**
當前權重	W	w
更新量 ΔW	−η · ∇J(w)	delta_w = -learning_rate * gradient_J_w

概念	數學表達式	程式實現 (偽代碼)
學習率	η	learning_rate
梯度	∇J(w)	gradient_J_w
更新步驟	$w^* = w - \eta \frac{\partial J}{\partial W}$	w = w + delta_w

程式碼範例 (Python/NumPy 概念):

```python
# 1. 計算梯度 (通過反向傳播)
gradient_J_w = calculate_gradient(data, w)

# 2. 設定學習率
learning_rate = 0.01

# 3. 計算更新量 (Delta W)
delta_w = -learning_rate * gradient_J_w

# 4. 更新權重 (W* = W + Delta W)
w = w + delta_w
# 此時的 w 即為下一輪的 w*
```