大數據資料分析實作

分類模型演算法

博雅(科技)課程

介紹分類模型代表性演算法

演算法名稱	實作方式	特徵
邏輯斯迴歸	損失函數型	將sigmoid函數的輸出值視為機率。分界線為直線。
支援向量機 Kernel method		利用Kernel method找出非直線的分界。
神經網路		利用增加隱藏層找出非直線的分界。
決策樹	決策樹型	以特定欄位值為基準，進行多次分組。
隨機森林		利用訓練資料的子集合建立多棵決策樹，並取多數決的結果。
XGBoost		將分類效果不佳的資料建立分類模型，以提高正確率。

損失函數（Loss Function）

📌 **簡單定義：損失函數就是用來衡量「模型預測」與「實際答案」有多接近的一個數學公式。**

- 當模型預測得很好，損失值就很小。
- 當模型預測得很差，損失值就很大。

損失函數（Loss Function）

訓練模型的目標是：

👉 **讓損失函數的值「越小越好」，也就是讓預測越來越準確。**

我們可以把機器學習的訓練過程想像成「打靶」：

- 🎯 真正的目標：正確答案（例如類別 1）
- 🔮 模型預測：你射出的箭（預測為 0.8 的機率）
- 📏 損失函數：幫你量出這支箭離靶心有多遠

損失函數型演算法（Loss Function-based）

- 損失函數型（Lost function） - 利用誤差最小化來建立分類模型。
- 這類模型的核心目的是「最小化預測錯誤」，也就是透過損失函數（Loss Function）來衡量「預測值」與「真實值」的差距，並藉由訓練過程不斷調整模型參數，讓損失越來越小。

演算法名稱	實作方式	特徵
邏輯斯迴歸	損失函數型	將sigmoid函數的輸出值視為機率。分界線為直線。
支援向量機 Kernel method		利用Kernel method找出非直線的分界。
神經網路		利用增加隱藏層找出非直線的分界。

損失函數型演算法（Loss Function-based）

1 邏輯斯迴歸（Logistic Regression）

◆定義與原理：

- 一種**線性**模型，用來做**分類**（尤其是二元分類：是/否、真/假）。
- 使用**sigmoid** 函數將線性組合的結果轉為機率值（0～1），並以機率門檻進行分類。

◆應用場景：

- 信用卡詐騙偵測
- 電子郵件是否為垃圾信
- 客戶是否會流失（Churn analysis）

◆特色：

- 模型**簡單**，**速度快**，容易理解。
- 適合**特徵彼此獨立**的情況。
- 若特徵與結果間非線性關係，效果較差。

損失函數型演算法（Loss Function-based）

2 支援向量機（Support Vector Machine, SVM）

◆定義與原理：

- 尋找一條**最佳超平面**，將資料分成兩類，並使分類邊界距離最近的資料點最遠（最大化邊界）。
- 支援**非線性**分類，透過**Kernel 核心**技巧將資料映射到高維空間進行分隔。

◆應用場景：

- 圖像辨識
- 生物資訊（癌症細胞分類）
- 文本分類（情感分析）

◆特色：

- 對**高維特徵**有效。
- 支援線性與非線性分類。
- 訓練速度較**慢**，**不適合大規模資料**。

損失函數型演算法（Loss Function-based）

3 神經網路（Neural Network）

◆定義與原理：

- 模仿人腦**神經元**運作，透過「**輸入層 → 隱藏層 → 輸出層**」的方式進行資訊處理與分類。
- 每層神經元計算加權總和後通過非線性激活函數，如 **sigmoid** 或 **ReLU**。

◆應用場景：

- 語音辨識、影像識別、自然語言處理（NLP）
- 自動駕駛、金融風險評估

◆特色：

- 適合處理**非線性**、**複雜**問題。
- 可學習**高階特徵**，但需大量資料與運算資源。
- 較難解釋模型內部邏輯（黑盒）。

課堂實作範例操作流程

實作步驟：畫出 sigmoid 函數的圖形

◆機器學習中很常見的一個激活函數：sigmoid 函數

- 整個公式是：

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

📌 用途：sigmoid函數會把任何實數 x 映射到 0 到 1 之間，常用在：

- 分類問題中預測機率
- 神經網路的激活函數

實作步驟：畫出 sigmoid 函數的圖形

◆ sigmoid 函數的定義

```
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```

- def 是定義函數的關鍵字，這裡定義一個叫做 sigmoid 的函數，參數是 x。
- np.exp(-x) 是 指數函數 e 的 -x 次方。來自 NumPy 函式庫。
- return 是回傳計算結果。

實作步驟：畫出 sigmoid 函數的圖形

🔷 產生輸入資料 x

```
x = np.linspace(-5, 5, 101)
```

- np.linspace 是 NumPy 裡常用來產生等間距數值的陣列。
- -5 到 5：代表 x 軸的範圍。
- 101：表示要產生 101 個點（剛好每 0.1 一個點）。

📌 用途：這樣可以讓圖形平滑，看起來漂亮。

實作步驟：畫出 sigmoid 函數的圖形

🔷 套用 sigmoid 函數產生對應 y 值

```
y = sigmoid(x)
```

- 把整個 x 陣列傳進 sigmoid 函數裡，每一個值都套用公式。
- 回傳一個一樣長度的陣列 y，是對應的 sigmoid 值。

📌 用途：有了 x 和 y，就可以畫出一條曲線。

實作步驟：畫出 sigmoid 函數的圖形

🔷 畫出 sigmoid 曲線

```
plt.plot(x, y, label='sigmoid函數', c='b', lw=2)
```

- plt.plot() 是畫線圖的函數。
- x, y 是 X 軸與 Y 軸的資料。
- label='sigmoid函數'：給這條線一個名稱，待會圖例要用。
- c='b'：線的顏色是藍色（blue）。
- lw=2：線的寬度是 2。

📌 用途：這就是 sigmoid 的 S 型曲線圖！

實作步驟：畫出 sigmoid 函數的圖形

🔷 顯示圖例

```
plt.legend()
```

- 顯示之前用 label 設定的名稱。
- 會自動幫你放在右上角（或自動選擇不擋到線的位置）。

📌 用途：圖中多條線時，用來分辨哪條是哪一個函數。

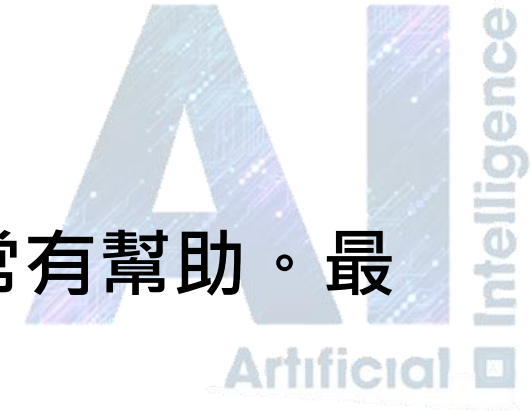
實作步驟：畫出 sigmoid 函數的圖形

🔷 加上格線並顯示整個圖形

```
plt.grid()
plt.show()
```

- 顯示背景的格線，幫助看數據的對應位置。
- 把圖形顯示出來，是畫圖的最後一步。如果沒 plt.show() 這行程式，圖可能不會顯示（尤其是在 Jupyter Notebook 之外執行時）。

📌 **用途：讓圖更清楚，尤其是對讀圖不熟的人來說非常有幫助。最後一行程式是 matplotlib 繪圖的標準結尾。**



實作步驟：畫出 sigmoid 函數的圖形

🔚 以上程式碼的流程是：

- 定義 sigmoid 函數。
- 建立從 -5 到 5 的等間距 x 資料。
- 計算對應的 y 值。
- 把 x 對 y 畫成一條線。
- 加上圖例、格線，美化圖形。
- 顯示圖表。

✅ 使用時機說明：

- 當在做**邏輯回歸**或**神經網路**時，想要了解 **sigmoid 函數**如何把「**輸入值**」轉換為「**機率值**」
- 為什麼它**適合分類**（例如 0 與 1）
- 透過**視覺化**了解sigmoid 函數

實作步驟：畫出 tanh 函數（雙曲正切函數）

◆機器學習中另一個激活函數：tanh 函數

- 整個公式是：

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- 輸入為任意實數，輸出介於 -1 和 1 之間。

📌 用途：在神經網路中作為激活函數，能強調輸入正負的差異，並保持平滑的非線性。

實作步驟：畫出 tanh 函數（雙曲正切函數）

◆ 定義 tanh 函數

```
def tanh(x):
    return np.tanh(x)
```

- def：定義一個函數叫 tanh，輸入為 x。
- np.tanh(x)：NumPy 提供的雙曲正切函數。來自 NumPy 函式庫。
- return 是回傳計算結果。

實作步驟：畫出 tanh 函數（雙曲正切函數）

🔷 建立 x 軸資料

```
x = np.linspace(-5, 5, 101)
```

- np.linspace(start, end, num)：產生 num 個從 start 到 end 的等距數字。
- 這裡從 -5 到 5 共 101 點，所以每個間距大約是 0.1。

📌 用途：這樣可以讓圖形平滑，看起來漂亮。

實作步驟：畫出 tanh 函數（雙曲正切函數）

🔷 計算 y 軸資料

```
y = tanh(x)
```

- 將 x 陣列代入剛剛定義的 tanh 函數中。
- 得到每個對應的 y 值，存成另一個陣列。

📌 用途：這是畫圖的 Y 軸資料來源。

實作步驟：畫出 tanh 函數（雙曲正切函數）

◆ 畫出 tanh 圖形

```
plt.plot(x, y, label='tanh 函數', c='green', lw=2)
```

- plt.plot(x, y)：用 x 和 y 資料畫線圖。
- label='tanh 函數'：這條線的名字，會出現在圖例中。
- c='green'：線的顏色是綠色。
- lw=2：線的粗細為 2（數字越大越粗）。

實作步驟：畫出 tanh 函數（雙曲正切函數）

◆ 顯示圖例、加上格線，並顯示整個圖形

```
plt.legend()
plt.grid()
plt.show()
```

- 會顯示前面用 label 設定的線條名稱。
- 增加格線，有助於對齊與觀察圖形細節。
- 最後將整張圖顯示出來，也是畫圖的必要步驟。

實作步驟：sigmoid 和 tanh 比較

📈 tanh vs sigmoid：小比較

函數	範圍	中心點	對稱性
sigmoid	(0, 1)	0.5	非對稱
tanh	(-1, 1)	0	原點對稱

📌 **tanh** 通常在訓練**神經網路**中比 **sigmoid** **表現更好**一些，因為它的輸出平均值是 **0**，更容易**幫助梯度下降收斂**。

實作步驟：畫出 ReLU 函數

◆ **ReLU（Rectified Linear Unit）：神經網路中非常重要的激活函數，也是目前在深度學習中最常用的激活函數之一。**

- 整個公式是：

$$\mathrm{ReLU}(x) = \max(0, x)$$

📌 用途：讓神經網路中只保留正值訊號，負值直接「砍掉」。這樣可以加速學習並解決梯度消失問題。

實作步驟：畫出 ReLU 函數

◆ 定義 ReLU 函數

```
def relu(x):
    return np.maximum(0, x)
```

- np.maximum(0, x)：這是 NumPy 的一個函數，會**比較 0 和 x 兩個值**，選出「**較大的那個**」。
- 如果 x 是負的，回傳 0；如果 x 是正的，回傳 x 自己。
- return 是回傳計算結果。

實作步驟：畫出 ReLU 函數

◆ 建立 x 軸資料

```
x = np.linspace(-5, 5, 101)
```

- np.linspace(start, end, num)：產生 num 個從 start 到 end 的等距數字。
- 從 -5 到 5 產生 101 個等間距的點。
- 為的是讓圖形細膩平滑。

實作步驟：畫出 ReLU 函數

◆ 計算 y 軸資料

```
y = relu(x)
```

- 把整個 x 陣列送入 ReLU 函數，得到對應的輸出 y。
- 得到每個對應的 y 值，存成另一個陣列。y 陣列如下所示。

```
[0, 0, 0, ..., 0, 0.1, 0.2, ..., 5.0]
```

實作步驟：畫出 ReLU 函數

◆ 畫出 ReLU 圖形

```
plt.plot(x, y, label='ReLU 函數', c='orange', lw=2)
```

- plt.plot(x, y)：用 x 和 y 資料畫線圖。
- label='ReLU 函數'：這條線的名字，會出現在圖例中。
- c='orange'：線的顏色是綠色。
- lw=2：線的粗細為 2。

✅ **說明：用橘色畫出一條代表 ReLU 函數的線。**

實作步驟：畫出 ReLU 函數

◆ 顯示圖例、加上格線，並顯示圖形

```
plt.legend()
plt.grid()
plt.show()
```

- 顯示 label 設定的名稱（ReLU 函數）。
- 增加格線，有助於判讀圖形位置與數值。
- 最後把圖表實際畫出來，顯示在畫面下方。

實作步驟：激活函數差異比較

📊 三種激活函數圖形比較（sigmoid、tanh、ReLU）

函數	輸出範圍	對稱性	是否非線性	缺點
Sigmoid	**0 ~ 1**	非對稱	✅	可能造成**梯度消失**（飽和區）
Tanh	**-1 ~ 1**	原點對稱	✅	飽和區仍可能導致**梯度消失**
ReLU	**0 ~ ∞**	非對稱	✅	對負數輸入為 0（可能導致「**死神經元**」）

實作步驟：支援向量機（SVM）模型

◆ SVM定義與原理：

- 尋找一條**最佳超平面**，將資料分成兩類，並使**分類邊界距離最近的資料點最遠**（**最大化邊界**）。
- 支援**非線性分類**，透過**Kernel 核心**技巧將資料映射到**高維空間**進行分隔。

📘 說明：SVC 是支援向量機（SVM）中最常見的分類模型，用來找出可以區分資料的「**最佳邊界**」。

📌 用途：使用 SVC（支援向量分類器）來分類資料，並用前面寫好的 plot_boundaries() 函式把分類結果畫出來。

實作步驟：支援向量機（SVM）模型

📌 1. 引入支援向量機演算法類別

```
from sklearn.svm import SVC
```

- 從 sklearn.svm 模組中，匯入 SVC 類別，它是「Support Vector Classifier」的縮寫。

實作步驟：支援向量機（SVM）模型

📌 2. 建立分類器，指定使用的「核心函數」

```
algorithm = SVC(kernel='rbf', random_state=random_seed)
```

- 建立一個 SVC 物件，並設定以下參數：

參數	說明
kernel='rbf'	使用 **RBF（高斯徑向基底）**核心函數，是最常用的非線性分類方式
random_state=random_seed	設定隨機種子，確保實驗可重現（前面有設定 random_seed = 123）

📘 **說明：SVC 支援多種 kernel，有 linear（線性）、poly（多項式）、rbf（高斯）、sigmoid，選擇不同 kernel 可以對應不同資料型態。**

實作步驟：支援向量機（SVM）模型

📌 3. 列印出演算法的參數設定

```
print(algorithm)
```

- 印出 algorithm（也就是剛建立的 SVC 模型）的完整參數設定。
- 幫助確認目前模型的設定狀況，是否使用了正確的 kernel、C 值、gamma 等
- 若顯示完整參數，輸出範例如下：

```
SVC(C=1.0, kernel='rbf', gamma='scale', random_state=123)
```

實作步驟：支援向量機（SVM）模型

📌 4. 呼叫畫圖函數，顯示分類結果

```
plot_boundaries(algorithm, DataList)
```

- 呼叫我們之前寫好的 plot_boundaries() 函數，將 algorithm（SVC 模型）套用到 DataList 中的每個資料集上。

📘 回顧一下 plot_boundaries() 功能：

- 把資料分成訓練集與測試集。
- 使用 SVM 模型來學習分類邊界。
- 畫出分類區域、訓練點與測試點。
- 顯示準確率（訓練 / 測試）。

實作步驟：支援向量機（SVM）模型

📊 結果畫面說明：

執行後會看到三張圖，分別對應：

- 線性可分的資料
- 新月型資料
- 同心圓資料

📊 結果畫面說明：

每張圖會顯示：

- 藍 / 黑 點：資料點，依類別分類
- 顏色背景：模型預測的分類區域
- 訓練點與測試點用不同標記區分（如 x）
- 左下角顯示準確率（例：驗證: 0.93 訓練: 0.95）

實作步驟：支援向量機（SVM）模型

📘 延伸學習建議：

想更進一步了解 SVM，可以試試調整以下參數看看結果差異：

- C=0.1 / C=10：控制**對誤差的容忍度**
- gamma='auto' / gamma=0.1：影響**決策邊界的彎曲程度**
- kernel='linear' / 'poly'：**切換核心函數**，對應不同資料型態

實作步驟：支援向量機（SVM）模型

✅ **嘗試將SVM 支援的四種常見核心（Kernel）在三種不同資料集（線性、月亮型、圓形）上的分類效果繪製呈現出差異比較**

📌比較四種 kernel：

- linear（線性）
- poly（多項式）
- rbf（高斯徑向基底）
- sigmoid（類神經型）

🎯 最終效果圖說明

你會看到 4 列 × 3 欄 的圖，共 12 張子圖：

- 每一列代表一種 kernel
- 每一欄代表一種資料型態
- 每張圖都會標出訓練準確率與測試準確率

實作步驟：支援向量機（SVM）模型

📘 觀察這些結果

Kernel	線性資料集效果	月亮型效果	圓形效果
linear	很好 👍	不理想 👎	不理想 👎
poly	可能還行 👌	視階數而定	可用 👌
rbf	非常靈活 👍	很好 👍	很好 👍
sigmoid	效果不穩定 🤔	有時可用	常常不佳 👎

實作步驟：支援向量機（SVM）模型

🔍 什麼是 C？

參數	說明
C	是 SVM 的**懲罰參數**，用來平衡「**分類邊界的寬度**」和「**錯誤分類的容忍度**」

- C **小** → 容忍錯誤較**多**，邊界較**平滑**（容易**underfit**）
- C **大** → 容忍錯誤較**少**，模型更**嚴格**（容易 **overfit**）

實作步驟：支援向量機（SVM）模型

✅ 比較不同 C 值的效果

- 使用 **rbf 核心**（最靈活），試試**不同 C 值**在三個資料集上的**分類表現**。

➕ 調整的 C 值範例：

```
C_list = [0.01, 0.1, 1, 10, 100]
```

實作步驟：支援向量機（SVM）模型

📌 程式碼：不同 C 值下的分類效果圖

```
from sklearn.svm import SVC

# 設定要比較的 C 值
C_list = [0.01, 0.1, 1, 10, 100]

# 設定畫布大小（5 種 C）
plt.figure(figsize=(20, 12))
```

- 一開始先設定 SVC 的 C 值，以及最後繪製分類圖的呈現大小。

實作步驟：支援向量機（SVM）模型

📌 程式碼：不同 C 值下的分類效果圖

```
# 每個 C 值建模並畫圖
for i, C_val in enumerate(C_list):
    model = SVC(kernel='rbf', C=C_val, random_state=random_seed)

    for j, (X, y) in enumerate(DataList):
        ax = plt.subplot(len(C_list), len(DataList), i * len(DataList) + j + 1)
        plot_boundary(ax, X, y, model)

        # 加上標籤說明
        if j == 0:
            ax.set_ylabel(f'C = {C_val}', fontsize=14)
        if i == 0:
            titles = ['線性分隔資料', '新月形資料', '圓形資料']
            ax.set_title(titles[j], fontsize=14)

plt.tight_layout()
plt.show()
```

實作步驟：支援向量機（SVM）模型

📊 結果觀察：不同 C 值下的分類效果

- C **很小** (**0.01**)：分類邊界很平滑，但**容易分類錯誤**。
- C **適中** (**1**)：平衡表現，通常**效果佳**。
- C **很大** (**100**)：分類邊界變得非常彎曲，幾乎貼近所有訓練資料，**容易 overfit**。

🧠 延伸學習：請固定 C，改調整 gamma 看看邊界變化

實作步驟：支援向量機（SVM）模型

🔍 什麼是 gamma？

參數	說明
gamma	控制一個訓練樣本影響力的範圍

- 小 gamma → 影響範圍大 → 模型平滑、**泛化能力強**
- 大 gamma → 影響範圍小 → 模型更專注於個別點，**容易 overfit**

實作步驟：支援向量機（SVM）模型

🔍 什麼是 gamma？

參數	說明
gamma	控制一個訓練樣本影響力的範圍

- 小 gamma → 影響範圍大 → 模型平滑、**泛化能力強**
- 大 gamma → 影響範圍小 → 模型更專注於個別點，**容易 overfit**

實作步驟：支援向量機（SVM）模型

✅ 比較不同 gamma 值的分類效果

- 使用 **rbf kernel**，**固定 C=1**，變化**不同的 gamma**，觀察在三個資料集上的分類表現。

➕ 調整的 gamma 值範例：

```
gamma_list = [0.01, 0.1, 1, 10, 100]
```

實作步驟：支援向量機（SVM）模型

📌 **程式碼：不同 gamma 值下的分類效果圖**

```
from sklearn.svm import SVC


# 要測試的 gamma 值
gamma_list = [0.01, 0.1, 1, 10, 100]


# 畫布大小（5 個 gamma 值 × 3 資料集）
plt.figure(figsize=(20, 12))
```

- 一開始先設定 SVC 不同的 gamma 值，以及最後繪製分類圖的呈現大小。

實作步驟：支援向量機（SVM）模型

📌 程式碼：不同 gamma 值下的分類效果圖

```
# 每個 gamma 值建立模型並畫圖
for i, gamma_val in enumerate(gamma_list):
    model = SVC(kernel='rbf', C=1, gamma=gamma_val, random_state=random_seed)

    for j, (X, y) in enumerate(DataList):
        ax = plt.subplot(len(gamma_list), len(DataList), i * len(DataList) + j + 1)
        plot_boundary(ax, X, y, model)

        # 加上標籤與標題
        if j == 0:
            ax.set_ylabel(f'γ = {gamma_val}', fontsize=14)
        if i == 0:
            titles = ['線性分隔資料', '新月形資料', '圓形資料']
            ax.set_title(titles[j], fontsize=14)

plt.tight_layout()
plt.show()
```

實作步驟：支援向量機（SVM）模型

📊 結果觀察：不同 gamma 值下的分類效果

gamma 值	預期效果
0.01	分類邊界很平緩，有點 underfit
0.1	還不錯，適合簡單資料集
1	通常**平衡效果好**
10	分類邊界變彎曲，可能開始 overfit
100	過度貼合訓練資料，邊界複雜，容易 overfit

🧠 延伸學習：要「找出最好的 C 與 gamma」的組合，可使用 GridSearchCV 來做交叉驗證找最佳參數組合！

實作步驟：支援向量機（SVM）模型

📊 結果觀察：不同 gamma 值下的分類效果

gamma 值	預期效果
0.01	分類邊界很平緩，有點 underfit
0.1	還不錯，適合簡單資料集
1	通常**平衡效果好**
10	分類邊界變彎曲，可能開始 overfit
100	過度貼合訓練資料，邊界複雜，容易 overfit

🧠 延伸學習：要「找出最好的 C 與 gamma」的組合，可使用 GridSearchCV 來做交叉驗證找最佳參數組合！

實作步驟：支援向量機（SVM）模型

什麼是 GridSearchCV？

- 全名是「**Grid Search with Cross Validation**」
- 就是將指定的所有參數組合都試一遍，透過**交叉驗證（cross-validation）**評估每種組合的效果，挑出效果最好的模型參數組合！
- GridSearchCV 的強大之處 — **自動挑出最佳參數組合**（C 和 gamma）

實作步驟：支援向量機（SVM）模型

✅ 程式碼範例（針對「新月形資料」）

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

- 使用 SVC（rbf kernel）

```
# 取新月資料集（非線性）
X, y = DataList[1]

# 定義 SVM 模型（rbf）
svc = SVC(kernel='rbf', random_state=random_seed)
```

實作步驟：支援向量機（SVM）模型

✅ 程式碼範例（針對「新月形資料」）

- 設定一個參數網格：試不同的 C 和 gamma

```
# 定義要搜尋的參數組合
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1, 10, 100]
}
```

實作步驟：支援向量機（SVM）模型

✅ 程式碼範例（針對「新月形資料」）

- 用 **5-fold 交叉驗證** 來挑最好的組合

```
# 建立 GridSearchCV 物件
grid = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)

# 執行搜尋（會花幾秒）
grid.fit(X, y)

# 印出最佳參數
print("最佳參數組合：", grid.best_params_)
print("最佳交叉驗證分數：", grid.best_score_)
```

實作步驟：支援向量機（SVM）模型

📊 使用最佳模型來畫圖

```
# 取出最佳模型
best_model = grid.best_estimator_


# 畫出分類邊界
plt.figure(figsize=(6, 4))
ax = plt.subplot(1, 1, 1)
plot_boundary(ax, X, y, best_model)
plt.title("最佳參數下的分類效果")
plt.show()
```

實作步驟：支援向量機（SVM）模型

💡 補充說明與提醒

- 若把參數範圍調整得更細，例如 np.logspace (-2, 2, 10) ，可來試更連續的 C 或 gamma
- 如果資料量很大，n_jobs = -1 可加速運算（使用所有 CPU）
- verbose = 1 可以顯示搜尋進度

實作步驟：神經網路（MLPClassifier）模型

1234 第一次神經網路分類（使用預設設定）

```
# 選擇演算法
from sklearn.neural_network import MLPClassifier
```

- 從 scikit-learn 套件中匯入 MLPClassifier，這是用來建立**多層感知器（Multi-layer Perceptron，簡稱 MLP）**神經網路分類器的工具。

```
algorithm = MLPClassifier(random_state=random_seed)
```

- 建立一個 MLPClassifier，並設定 random_state 為 random_seed（這是為了讓每次執行時隨機結果相同）。

實作步驟：神經網路（MLPClassifier）模型

1234 第一次神經網路分類（使用預設設定）

- 如果沒設定其他參數，它會使用預設值：
  - ✓ 隱藏層大小為 (100,)（表示只有一層有 100 個神經元）
  - ✓ 激活函數為 'relu'
  - ✓ 學習演算法為 'adam'

```
# 顯示演算法的參數
print(algorithm)
```

- 列印出剛才建立的 MLP 模型的所有參數設定，例如：隱藏層結構、學習率、最大訓練次數等資訊，方便確認模型設定。

實作步驟：神經網路（MLPClassifier）模型

1234 第一次神經網路分類（使用預設設定）

```
# 呼叫顯示功能
plot_boundaries(algorithm, DataList)
```

- 這是之前寫好的自訂函數，會對三個資料集：線性可分、新月形（非線性）、圓形（非線性），使用神經網路進行訓練，並畫出分類邊界與預測結果的視覺化圖表。

實作步驟：神經網路（MLPClassifier）模型

第二次神經網路分類（自訂隱藏層結構）

```
# 隱藏層節點數 = (100,100)
from sklearn.neural_network import MLPClassifier
algorithm = MLPClassifier(hidden_layer_sizes=(100,100), random_state=random_seed)
```

- 這次改用 hidden_layer_sizes=(100,100) 表示模型會有兩層隱藏層，每層各 100 個神經元。
- 改變隱藏層的結構，可以讓模型有更多的能力去**擬合複雜資料**（例如：非線性分布）

實作步驟：神經網路（MLPClassifier）模型

第二次神經網路分類（自訂隱藏層結構）

```
# 顯示演算法的參數
print(algorithm)
```

- 這會顯示模型所有設定參數，以便檢查這次有兩層隱藏層。

```
# 呼叫顯示功能
plot_boundaries(algorithm, DataList)
```

- 再一次訓練模型並顯示圖形結果，可以和前面的一層隱藏層模型進行視覺比較，看哪個分類效果更好。

實作步驟：神經網路（MLPClassifier）模型

🧠 補充說明：為何要用 MLPClassifier？

- **MLP** 是一種「**前饋神經網路**」，適合：
  - ✓ 處理**非線性分類**問題（像新月、同心圓）
  - ✓ **可以調整參數與層數**，以提升表現
  - ✓ 是深度學習的基本入門方法

「大數據資料分析實作」課程