



# 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 )。





# 決策樹型演算法（ Tree-based ）

- ◆ 決策樹型（ Decision Tree ） - 利用條件判斷建立樹狀結構分類模型
- ◆ 這類模型以「若...則...」的邏輯為核心，逐步分裂資料，形成類似流程圖的結構（像一棵倒掛的樹），最終決定分類結果。

演算法名稱	實作方式	特徵
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。

# 決策樹型演算法 ( Tree-based )

## 4 決策樹 ( Decision Tree )

### ◆ 定義與原理：

- 根據資料特徵，進行**條件式分割**（如：身高 > 170？是 → 分左邊）。
- 每個節點代表一個條件，葉節點代表最終分類。

### ◆ 應用場景：

- 客戶分群
- 醫療診斷
- 法律判斷支援

### ◆ 特色：

- **視覺化容易**，直觀明瞭。
- 容易**過度擬合**，對雜訊敏感。
- **不須標準化**特徵。



# 決策樹型演算法 ( Tree-based )

## 5 隨機森林 ( Random Forest )

### ◆ 定義與原理：

- 建立**多顆決策樹**，並綜合它們的結果來進行分類（多數決）。
- 使用「**Bagging（自助抽樣）**」與**隨機特徵選取**來提升**泛化能力**。

### ◆ 應用場景：

- 顧客信用評分
- 診斷疾病風險
- 商品推薦

💡【補充說明】：所謂**泛化能力(generalization ability)**是指一個機器學習算法對於沒有見過的樣本資料有很好的識別能力。

### ◆ 特色：

- **穩定性高、準確度佳**。
- **抗過擬合**能力強。
- 訓練時間較長，**可平行處理**。



# 決策樹型演算法 ( Tree-based )

## 6 XGBoost ( Extreme Gradient Boosting )

### ◆定義與原理：

- 屬於「**梯度提升樹 ( Gradient Boosting Tree )**」的一種。
- 每棵樹根據前一棵樹的預測誤差進行修正，透過「**逐步學習**」來強化整體模型。
- 是一種**集成學習 ( Ensemble Learning )**。

### ◆應用場景：

- 金融風控、詐騙偵測
- 電商推薦系統

### ◆特色：

- 高效能、**準確率高**。
- 支援缺失值處理、自動特徵選擇。
- 參數多但具彈性，適合進階調整。



# 各種演算法比較表

演算法	支援非線性	可視化性	解釋性	精準度	訓練速度	資料需求
Logistic	✗	✓	✓	中	快	低
SVM	✓	中	中	高	慢	中
Neural Net	✓	✗	✗	高	慢	高
Decision Tree	✗	✓	✓	中	快	中
Random Forest	✓	中	中	高	中	中
XGBoost	✓	中	中	最高	中慢	中

# 課堂實作範例操作流程



# 實作步驟：安裝與設定

## ◆ 安裝與設定日文字體支援

```
!pip install japanize-matplotlib | tail -n 1
```

- `!pip install japanize-matplotlib`：這是使用 Jupyter Notebook 的一種方式，透過 **!** 來執行系統指令，這裡安裝一個可以讓 **matplotlib** 支援日文顯示的套件。
- `| tail -n 1`：只顯示安裝結果的最後一行，避免太多輸出資訊。

📌 用途：若要在圖表中顯示日文，這個套件可以讓字不會亂碼。




# 實作步驟：安裝與設定

## ◆ 調整模型列印設定

```
from sklearn import set_config  
set_config(print_changed_only=False)
```

- `from sklearn import set_config`：匯入 `scikit-learn` 的設定工具。
- `set_config(print_changed_only=False)`：設定模型顯示時，不只顯示使用者變更過的參數，也一併列出所有預設參數。


 **用途：**當在使用機器學習模型時，這樣可以完整看到模型的所有設定細節。

# 實作步驟：安裝與設定

## ◆ 隱藏不必要的警告

```
import warnings  
warnings.filterwarnings('ignore')
```

- `import warnings`：匯入 Python 的警告模組。
- `warnings.filterwarnings('ignore')`：設定忽略所有警告訊息。

 **用途**：清除程式執行時會出現的警告提示（但不建議在正式開發時使用，容易忽略問題）。

# 實作步驟：安裝與設定

## ◆ 匯入常用的資料處理與視覺化套件

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

- pandas (pd)：用於資料讀取與資料框處理。
- numpy (np)：用來處理數學、陣列等運算。
- matplotlib.pyplot (plt)：Python 的繪圖工具，可以用來畫折線圖、長條圖等。

 用途：資料分析與視覺化的標配三寶。

# 實作步驟：安裝與設定

## ◆ Jupyter 專用：漂亮地顯示資料框

```
from IPython.display import display
```

- display() 函數可在 Jupyter Notebook 中更漂亮地顯示資料框 ( DataFrame ) 。

📌 用途：比 print() 更適合用來看表格資料。



# 實作步驟：安裝與設定

## ◆ 設定資料顯示的樣式與精度

```
np.set_printoptions(suppress=True, precision=4)
```

- `suppress=True`：避免用科學記號顯示數字（如 `1.23e+05`）。
- `precision=4`：設定小數點後顯示 4 位數。

```
pd.options.display.float_format = '{:.4f}'.format
```

- 設定 pandas 資料框內的浮點數顯示格式為小數點後 4 位。



# 實作步驟：安裝與設定

## ◆ 設定資料顯示的樣式與精度

```
pd.set_option("display.max_columns",None)
```

- 顯示資料框時不省略欄位（即使欄位很多，也全部顯示出來）。

```
plt.rcParams["font.size"] = 14
```

- 設定圖表中文字的預設字型大小為 14。


 **用途：**這些設定可以更方便閱讀資料與圖形。

# 實作步驟：安裝與設定

## ◆ 設定隨機數種子

```
random_seed = 123
```

- random\_seed 是為了確保每次執行時，隨機結果一致（可重現）。

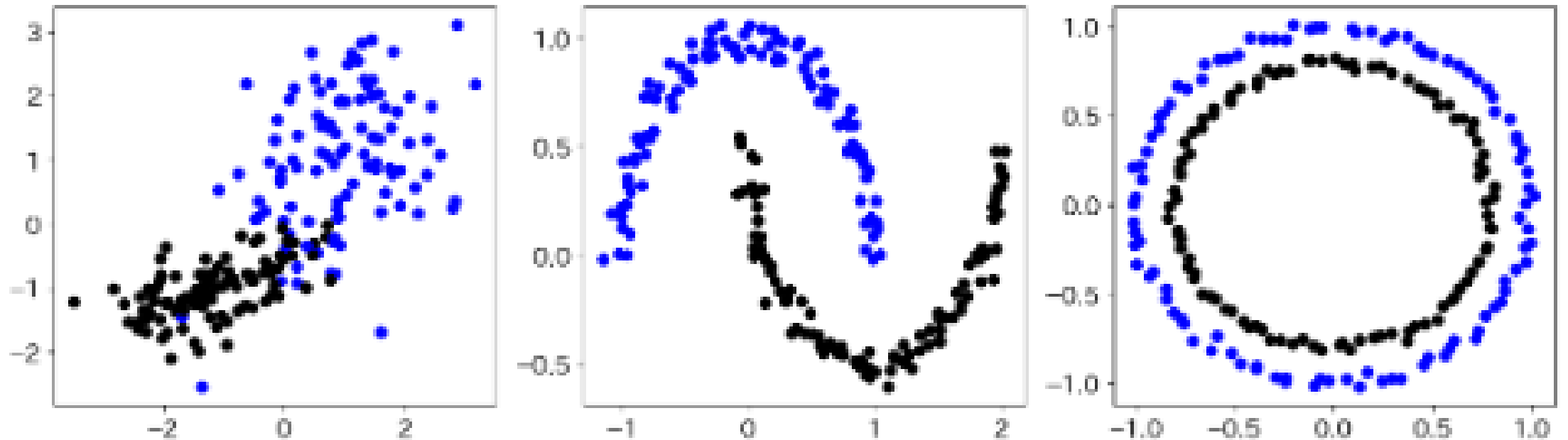
 **用途：**當你使用隨機取樣、機器學習等會有隨機性的操作時，這能確保你每次得到相同結果，有助於除錯與比較。





# 實作步驟：建立不同資料分布樣本與繪圖

- 🎯 建立三種不同的資料分布樣本（**線性可分離**、**新月形**、**同心圓**），然後畫出這些資料的**散佈圖**與分類模型的**決策邊界**。




# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 產生樣本資料

```
from sklearn.datasets import make_moons
from sklearn.datasets import make_circles
from sklearn.datasets import make_classification
```

- random 這三個函數是用來生成「**模擬資料**」的。
  - make\_classification：產生**線性可分離**的分類資料。
  - make\_moons：產生**新月形狀**的資料，通常是**非線性**資料。
  - make\_circles：產生**兩個同心圓**的資料，也是**非線性**資料。

 **用途：**建立人工資料來測試分類模型。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 建立三種資料

# 線性分離

```
X1, y1 = make_classification(n_features=2, n_redundant=0,  
                             n_informative=2, random_state=random_seed,  
                             n_clusters_per_class=1, n_samples=200, n_classes=2)
```

- 建立200筆線性可分的資料。
- X1 是資料點的位置（x座標、y座標），y1 是每筆資料的分類標籤（0 或 1）。
- n\_features=2：每筆資料有兩個特徵（對應平面上的兩個座標）。
- random\_state=random\_seed：固定亂數種子，讓每次結果一樣。
- n\_samples=200：產生 200 筆資料。
- n\_classes=2：有兩種分類（0 與 1）。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 建立三種資料

```
# 新月形 (線性不可分割)
```

```
X2, y2 = make_moons(noise = 0.05, random_state=random_seed,  
                    n_samples=200)
```

- 產生 200 筆新月形資料，加上一些雜訊讓資料不那麼規則。

```
# 圓形 (無法進行線性分離)
```

```
X3, y3 = make_circles(noise = 0.02, random_state=random_seed,  
                    n_samples=200)
```

- 產生 200 筆同心圓資料，加上一點雜訊。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 儲存與初步設定

```
# 在 DataList 中儲存三種類型的資料  
DataList = [(X1, y1), (X2, y2), (X3, y3)]  
  
# N：資料類型的數量  
N = len(DataList)
```

- 把三組資料包成一個列表DataList，方便後續處理。
- N 是資料種類數（3 種）。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 畫出三種資料的散佈圖

```
# 散點圖顯示
```

```
plt.figure(figsize=(15,4))
```

- 建立一個圖形，寬 15 吋，高 4 吋。

```
from matplotlib.colors import ListedColormap
```

```
cmap = ListedColormap(['#0000FF', '#000000'])
```

- 定義點的顏色：一種是藍色、一種是黑色，分別對應兩個分類標籤（0、1）。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 畫出三種資料的散佈圖

```
for i, data in enumerate(DataList):  
    X, y = data  
    ax = plt.subplot(1, N, i+1)  
    ax.scatter(X[:,0], X[:,1], c=y, cmap=cmap)
```

- 使用迴圈一張張畫出三種資料，並使用 **subplot** 把圖畫在**同一排**。
- **scatter()** 畫散佈圖，而 $X[:,0]$  和  $X[:,1]$  分別是 X 軸和 Y 軸的資料。
- **c=y** 表示根據標籤來**配色**。

```
plt.show()
```

- 顯示圖形。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 準備訓練與視覺化模型 - 匯入訓練測試分割工具

```
from sklearn.model_selection import train_test_split
```

- 匯入切割資料用的工具，把資料分成「訓練集」與「測試集」。

## ◆ 定義分類邊界顯示函式

```
def plot_boundary(ax, x, y, algorithm):
```

- 定義一個函式，可以畫出分類器的「決策邊界（Decision Boundary）」。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 定義分類邊界顯示函式

```
x_train, x_test, y_train, y_test = train_test_split(x, y,  
                                                    test_size=0.5, random_state=random_seed)
```

- 將資料隨機分成一半訓練、一半測試。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 定義分類邊界顯示函式

```
from matplotlib.colors import ListedColormap  
cmap1 = plt.cm.bwr # 背景顏色 (紅藍)  
cmap2 = ListedColormap(['#0000FF', '#000000']) # 點顏色
```

- 設定兩組顏色對應圖：背景（邊界）用紅藍，點用藍黑。

```
h = 0.005
```

- 設定網格解析度，數值越小，邊界圖越細緻。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 定義分類邊界顯示函式

```
algorithm.fit(x_train, y_train)
```

- 訓練模型：使用分類器學習資料（訓練）。

```
score_test = algorithm.score(x_test, y_test)  
score_train = algorithm.score(x_train, y_train)
```

- 評估模型的準確率，分別針對測試資料和訓練資料。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 定義分類邊界顯示函式

```
f1_min = x[:, 0].min() - 0.5  
f1_max = x[:, 0].max() + 0.5  
f2_min = x[:, 1].min() - 0.5  
f2_max = x[:, 1].max() + 0.5
```

- 決策邊界繪圖範圍：找出座標範圍，並加一點空間。

```
f1, f2 = np.meshgrid(np.arange(f1_min, f1_max, h),  
                     np.arange(f2_min, f2_max, h))
```

- 建立一張網格，即 X-Y 座標的格點，用來準備畫圖顯示決策邊界。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 定義分類邊界顯示函式

```
if hasattr(algorithm, "decision_function"):  
    Z = algorithm.decision_function(np.c_[f1.ravel(), f2.ravel()])  
    Z = Z.reshape(f1.shape)  
    ax.contour(f1, f2, Z, levels=[0], linewidth=2)
```

- 若分類器有 `decision_function`，就算出每個網格點的預測值，用來畫邊界線。

```
else:  
    Z = algorithm.predict_proba(np.c_[f1.ravel(), f2.ravel()])[:, 1]  
    Z = Z.reshape(f1.shape)
```

- 否則用 `predict_proba`（預測每類的機率），也就是用預測機率來畫出邊界。
- `Z = Z.reshape(f1.shape)`：把結果變成和網格一樣形狀。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 定義分類邊界顯示函式

```
ax.contourf(f1, f2, Z, cmap=cmap1, alpha=0.3)
```

- 填色顯示分區（背景決策區域），即畫出顏色區域，也就是模型決策的邊界。

```
ax.scatter(x_test[:,0], x_test[:,1], c=y_test, cmap=cmap2)
```

```
ax.scatter(x_train[:,0], x_train[:,1], c=y_train, cmap=cmap2, marker='x')
```

- 畫出訓練資料和測試資料（測試資料是圓點，訓練資料是叉叉）。

```
text = f'測試:{score_test:.2f} 訓練:{score_train:.2f}'
```

```
ax.text(f1.max() - 0.3, f2.min() + 0.3, text, horizontalalignment='right', fontsize=18)
```

- 顯示訓練與測試的準確率。



# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 整合畫出多圖的函式

```
def plot_boundaries(algorithm, DataList):  
    plt.figure(figsize=(15,4))  
    for i, data in enumerate(DataList):  
        X, y = data  
        ax = plt.subplot(1, N, i+1)  
        plot_boundary(ax, X, y, algorithm)  
    plt.show()
```

- 定義函式，讓某個模型把三種資料的決策邊界都畫出結果。即傳入一個模型（像是 SVC() 或 LogisticRegression()），就會對三組資料各自訓練、畫圖。
- for迴圈是針對每種資料畫出一張圖，並呼叫上面寫的 plot\_boundary 函式。

# 實作步驟：建立不同資料分布樣本與繪圖

## ◆ 選用分類演算法

```
from sklearn.linear_model import LogisticRegression  
model = LogisticRegression()  
plot_boundaries(model, DataList)
```

- 使用一行指令，就可以看見測試模型在三種資料上的分類效果！



# 「大數據資料分析實作」課程

