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。

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

◆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**。

💡【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。

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。

三者比較

項目	決策樹	隨機森林	XGBoost
結構	一棵樹	多棵隨機樹	一棵棵逐步修正的樹
是否易過擬合	✅（容易）	❌（不易）	❌（可透過參數控制）
解釋性	✅ 很高	中等	❌ 較低（需配SHAP）
訓練速度	快	中等	較慢（但有並行運算）
準確率	中等	高	很高
可處理資料量	小～中	中	大
特徵重要性	有	有	有（且配SHAP效果佳）

三者比較

模型名稱	說明	優點	缺點
決策樹 (Decision Tree)	以條件方式進行分類，適合直覺理解	解釋性強 訓練快	易過擬合 穩定性差
隨機森林 (Random Forest)	多棵隨機決策樹集成的模型	抗過擬合 效果穩定	訓練成本較高 難解釋
XGBoost	強化型決策樹集成法，適用於競賽與實務應用	效能強大 可調參數多	複雜度高 學習曲線陡峭

課堂實作範例操作流程

實作步驟：決策樹（Decision Tree）分類器

◆載入資料與建立模型

```
# 匯入額外的函式庫
import seaborn as sns
```

- seaborn 是一個資料視覺化函式庫，但這裡我們是使用它提供的範例資料集，例如 iris 花卉資料集。

```
# 載入樣本資料
df_iris = sns.load_dataset("iris")
```

- 透過 sns.load_dataset("iris") 載入內建的 Iris 花卉資料集，這是機器學習最常用的分類練習資料集之一，共有 150 筆資料、3 種花類別（Setosa、Versicolor、Virginica）。

實作步驟：決策樹（Decision Tree）分類器

◆載入資料與建立模型

```
# 縮小到兩種花卉
df2 = df_iris[50:150]
```

- **Iris 資料集中：**
  - ✓ **前 50 筆是 Setosa（容易分類）**
  - ✓ **50～150 是 Versicolor 與 Virginica（較難區分）**
- **我們這裡取第 51～150 筆，縮小成二元分類問題（只比較後兩種花）。**

實作步驟：決策樹（Decision Tree）分類器

◆資料預處理

```
# 資料分離
X = df2.drop('species', axis=1)
y = df2['species']
```

- X 是特徵（四個欄位：花萼長、花萼寬、花瓣長、花瓣寬）
- y 是目標標籤（species：花的種類）
- drop(..., axis=1) 表示「從欄位中」移除 species，只保留特徵。

實作步驟：決策樹（Decision Tree）分類器

◆建立並訓練決策樹模型

```
# 學習
from sklearn.tree import DecisionTreeClassifier
algorithm = DecisionTreeClassifier(random_state=random_seed)
algorithm.fit(X, y)
```

- **DecisionTreeClassifier() 是 sklearn 中的決策樹分類器**
- **fit(X, y) 表示用資料 X 和標籤 y 來訓練模型**
- **random_state 是為了結果可重現**

實作步驟：決策樹（Decision Tree）分類器

決策樹圖形輸出（產生一個決策樹的 .dot 檔案）

```
from sklearn import tree
with open('iris-dtree.dot', mode='w') as f:
    tree.export_graphviz( algorithm, out_file=f,
        feature_names=X.columns, filled=True, rounded=True,
        special_characters=True, impurity=False, proportion=False
    )
```

- **export_graphviz 將訓練好的模型輸出成 Graphviz 格式**
- **filled=True：節點會用顏色填滿，幫助視覺理解**
- **rounded=True：節點框線圓角**
- **impurity=False：不要顯示基尼不純度**
- **proportion=False：不顯示每個節點樣本比例**

實作步驟：決策樹（Decision Tree）分類器

🌳 決策樹圖形輸出（將 .dot 檔轉成圖片並顯示出來）

```
import pydotplus
from IPython.display import Image
graph = pydotplus.graphviz.graph_from_dot_file('iris-dtree.dot')
graph.write_png('iris-dtree.png')
Image(graph.create_png())
```

- 使用 **pydotplus** 將 **.dot** 格式轉為圖形（**png**）
- 使用 **IPython.display.Image()** 在 **Jupyter Notebook** 中顯示圖片

📌 這段非常實用：能直接在 **notebook** 中視覺化我們的決策邏輯！

實作步驟：決策樹（Decision Tree）分類器

📉 決策邊界的顯示：視覺化分類效果

```
from sklearn.tree import DecisionTreeClassifier
algorithm = DecisionTreeClassifier(random_state=random_seed)

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

- 使用之前定義的 **plot_boundaries()** 函數
  - ✓ 它會對三種人工資料集（線性、新月、圓形）進行訓練
  - ✓ 並畫出分類邊界與訓練/測試資料的分布
- 這裡先用預設的決策樹設定

實作步驟：決策樹（Decision Tree）分類器

🌿 限制樹深度後再顯示分類效果

```
algorithm = DecisionTreeClassifier(max_depth=3, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

- 將決策樹的最大深度設為 3
  - ✓ **max_depth=3** 表示樹最多只能長到第 3 層（可用來避免過度擬合）
- 這樣做的好處：
  - ✓ 可視化模型學習「多複雜」
  - ✓ 控制模型複雜度，提高泛化能力

實作：隨機森林（Random Forest）分類器

🌲🌲🌲 隨機森林的概念

是一種 集成學習法（ensemble method），是由多棵「決策樹（Decision Trees）」所組成的模型，透過「投票」來進行分類。

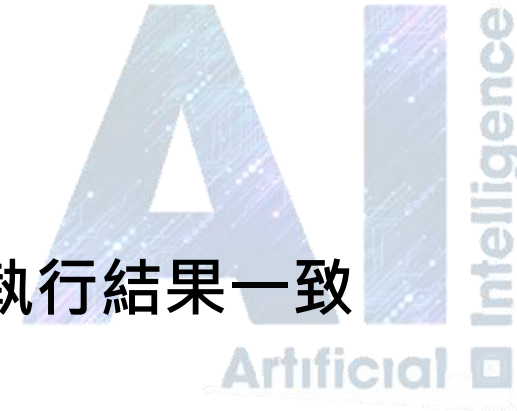
- 建立多棵隨機化的決策樹（每棵只用部分特徵與資料）
- 多數決（majority vote）來判斷分類結果
- 可以降低過擬合、提升穩定性和準確度

實作：隨機森林（Random Forest）分類器

🌲🌲🌲 建立分類模型

```
# 選擇演算法
from sklearn.ensemble import RandomForestClassifier
algorithm = RandomForestClassifier(random_state=random_seed)
```

- 從 **sklearn.ensemble** 模組中匯入 **RandomForestClassifier**
- **algorithm** 是定義的模型名稱（也可以改用 **clf, model** 等）
- **RandomForestClassifier(...)** 創建一個隨機森林分類器
- **random_state=random_seed** 設定隨機種子，確保每次執行結果一致（方便比對、除錯）



實作：隨機森林（Random Forest）分類器

🌲🌲🌲 建立分類模型

- 可以自行調整的參數還包括：

參數	功能說明
n_estimators	森林中樹的數量（預設 100）
max_depth	每棵樹的最大深度（防止過擬合）
max_features	建立每棵樹時使用的最大特徵數
min_samples_split	決定節點何時拆分

- 🛠️ 範例：

```
RandomForestClassifier(n_estimators=200, max_depth=4)
```

實作：隨機森林（Random Forest）分類器

🌲🌲🌲 建立分類模型

```
# 顯示演算法的參數
print(algorithm)
```

- 印出模型的詳細設定，這樣可以看到模型目前使用的是哪些預設參數
- 若自訂了 n_estimators=200 或其他參數，此行程式就會一併顯示出來

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

- 用來視覺化分類效果（重要！）
  - ✓ **plot_boundaries()** 是之前定義過的函式（用來繪製資料散佈圖與分類邊界）
  - ✓ **algorithm** 則是定義的隨機森林模型
  - ✓ **DataList** 是包含三組人工資料的清單（線性、新月、圓形）

🟦 視覺化內容：

- ✓ 訓練資料（用「X」顯示）
- ✓ 測試資料（用點點顯示）
- ✓ 顏色區域表示模型如何做分類（就是 **decision boundary**）

實作：隨機森林（Random Forest）分類器

✅ 總結

步驟	說明
1 匯入 RandomForestClassifier	使用集成式的強大分類器
2 建立模型物件	可以自訂參數：樹的數量、深度等
3 印出模型參數	確認目前設定
4 呼叫 plot_boundaries	對三種資料集畫出分類效果與邊界

實作：隨機森林（Random Forest）分類器

🧪 針對隨機森林（Random Forest）演算法，嘗試以下參數調整：

1. 調整 n_estimators 的值（比如改成 5、20、100、500 看分類效果變化）
2. 調整 max_depth 觀察模型是否過擬合
3. 結合 GridSearchCV 自動挑選最佳參數組合

實作：隨機森林（Random Forest）分類器

🧠 認識隨機森林（Random Forest）重要參數 — n_estimators

- n_estimators 是隨機森林中「樹的數量」
- 值愈大，模型愈穩定、準確，但訓練時間也愈長
- 常見的設置值：5（非常少）、20（適中）、100（常見）、500（高準確但成本高）

實作：隨機森林（Random Forest）分類器

🧪 實作：逐步觀察不同 n_estimators 的分類效果

```
# 1. 匯入必要套件
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

# 假設你已經定義好以下：
# random_seed：隨機種子（確保每次執行結果一致）
# DataList：三個資料集（線性、月牙、同心圓）
# plot_boundaries()：幫你畫分類圖的自訂函式
```

實作：隨機森林（Random Forest）分類器

✅ 步驟一：測試 n_estimators = 5

```
algorithm = RandomForestClassifier(n_estimators=5, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

🟦 說明：只用 **5 棵樹**，模型容易不穩定，分類邊界可能不夠準確。

🔎 初學者學習重點：觀察邊界是否有明顯雜訊或錯誤分類。

實作：隨機森林（Random Forest）分類器

✅ 步驟二：測試 n_estimators = 20

```
algorithm = RandomForestClassifier(n_estimators=20, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

📘 說明：20 棵樹已較穩定，結果會明顯比 5 棵樹好。

🔎 學習重點：開始接近實用等級的模型穩定度。

實作：隨機森林（Random Forest）分類器

✅ 步驟三：測試 n_estimators = 100

```
algorithm = RandomForestClassifier(n_estimators=100, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

📘 說明：100 棵樹是實務中常用的設置，效果穩定，分類結果通常已很理想。

🔎 學習重點：觀察與 20 棵樹相比是否有明顯提升。

實作：隨機森林（Random Forest）分類器

✅ 步驟四：測試 n_estimators = 500

```
algorithm = RandomForestClassifier(n_estimators=500, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

📘 說明：更多的樹可以提升準確率，但訓練時間較長。

🔎 學習重點：是否值得用這麼多樹，視應用需求而定。

實作：隨機森林（Random Forest）分類器

📊 學習任務建議

n_estimators	準確率變化	訓練時間	適用情境
5	不穩定、容易錯誤	快	測試或教學
20	基本可用	快	小型專案
100	穩定、好結果	中	實務常用
500	非常穩定、高準確	慢	高準確需求

實作：隨機森林（Random Forest）分類器

🧠 認識隨機森林（Random Forest）另一個重要參數 — max_depth

- 每棵樹的最大深度
- 這個參數會直接影響模型的學習能力與是否過度擬合（Overfitting）

🎯 認識 max_depth

概念	說明
max_depth	限制每棵決策樹的最大深度
無限制（預設 None）	樹會長到不能再分為止（可能過擬合）
小的深度	模型學不到複雜結構（可能欠擬合）
適中深度	可以取得好的泛化能力

實作：隨機森林（Random Forest）分類器

🧪 練習步驟（以 RandomForestClassifier 為例）

🌱 事前準備

```
from sklearn.ensemble import RandomForestClassifier


# 假設你已經定義好這些變數：
# random_seed：隨機種子
# DataList：三組資料（線性、月牙、同心圓）
# plot_boundaries()：繪圖函式
```

實作：隨機森林（Random Forest）分類器

🌿 步驟一：max_depth = 2（限制太淺，容易欠擬合）

```
algorithm = RandomForestClassifier(max_depth=2, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

✅ 觀察點：分類邊界可能太簡單，無法正確分辨資料。

❗ 學習重點：欠擬合（underfitting）會導致訓練集和測試集都不準。

實作：隨機森林（Random Forest）分類器

🌲 步驟二：max_depth = 5（中等深度）

```
algorithm = RandomForestClassifier(max_depth=5, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

✅ 觀察點：開始能分出有意義的邊界。

❗ 學習重點：可能是「適中」的模型，效果穩定。

實作：隨機森林（Random Forest）分類器

🌳 步驟三：max_depth = 10（樹較深）

```
algorithm = RandomForestClassifier(max_depth=10, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

✅ 觀察點：邊界更貼合資料，但風險是開始「過擬合」。

❗ 學習重點：留意測試集準確率有沒有下降。

實作：隨機森林（Random Forest）分類器

🌴 步驟四：max_depth 無限制（預設 None）

```
algorithm = RandomForestClassifier(max_depth=None, random_state=random_seed)
print(algorithm)
plot_boundaries(algorithm, DataList)
```

✅ 觀察點：分類邊界非常複雜，貼合所有訓練資料。

❗ 學習重點：訓練集準確率幾乎100%，但測試集可能變差 → 過擬合（Overfitting）

實作：隨機森林（Random Forest）分類器

📊 整理對照表

max_depth	模型表現	過擬合風險	建議用途
2	欠擬合	❌低	快速測試
5	穩定，適中	⚠️中等	小型專案預設
10	表現不錯	⚠️偏高	精細調整
None	可能過擬合	✅ 高	當資料量夠多時考慮

實作：隨機森林（Random Forest）分類器

✅ 在實務上非常常見使用〝GridSearchCV〞 來自動找出最佳的隨機森林（Random Forest）參數組合。

✅ 使用〝GridSearchCV〞可以幫助：

- 找出不同參數對模型效果的影響
- 減少手動調參的工作量
- 自動找出訓練與驗證表現都好的「最佳參數組合」

實作：隨機森林（Random Forest）分類器

🧠 初學者要先懂的名詞

名詞	解釋
GridSearchCV	幫你嘗試每組參數，找出表現最好的組合
param_grid	你想要測試的參數與範圍（以字典方式表示）
cv	幾折交叉驗證（通常是 5 或 10）
best_params_	最佳參數組合
best_score_	交叉驗證時的平均準確率

📘 練習目標：嘗試用 GridSearchCV 來找出：

- n_estimators（樹的數量）
- max_depth（最大深度）
- 最適合分類的參數組合

實作：隨機森林（Random Forest）分類器

🧪 步驟教學：GridSearchCV + RandomForestClassifier

✅ 步驟 1：匯入必要的工具

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

✅ 步驟 2：定義參數網格

```
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [3, 5, 10, None]
}
```

🔎 這會測試 3（樹的數量）× 4（深度）= 12 種組合

實作：隨機森林（Random Forest）分類器

✅ 步驟 3：建立 GridSearchCV 物件並訓練

以下以其中一筆資料為例（例如 DataList[0]）

```
from sklearn.model_selection import train_test_split

# 使用第一組資料
X, y = DataList[0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=random_seed)

# 建立基礎分類器
rf = RandomForestClassifier(random_state=random_seed)

# 建立 GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5)

# 開始搜尋
grid_search.fit(X_train, y_train)
```

實作：隨機森林（Random Forest）分類器

✅ 步驟 4：觀察結果

```
print("最佳參數組合：", grid_search.best_params_)
print("最佳交叉驗證準確率：", grid_search.best_score_)
```

✅ 步驟 5：用最佳參數畫出分類邊界

```
best_model = grid_search.best_estimator_

# 使用之前寫好的圖形函式
plot_boundary(plt.gca(), X, y, best_model)
plt.show()
```

實作：隨機森林（Random Forest）分類器

🧪 進階延伸：三個資料集自動 GridSearch

可以這樣寫個迴圈，一次處理所有資料集：

```
for i, (X, y) in enumerate(DataList):
    print(f"\n資料集 {i+1} 的 GridSearchCV 結果：")
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=random_seed)

    rf = RandomForestClassifier(random_state=random_seed)
    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5)
    grid_search.fit(X_train, y_train)

    print("最佳參數：", grid_search.best_params_)
    print("交叉驗證準確率：", grid_search.best_score_)
    print("測試集準確率：", grid_search.score(X_test, y_test))

    # 顯示圖形
    plt.figure(figsize=(5, 4))
    plot_boundary(plt.gca(), X, y, grid_search.best_estimator_)
    plt.title(f"資料集 {i+1}")
    plt.show()
```

實作：隨機森林（Random Forest）分類器

🎯 學習重點回顧

重點	說明
GridSearchCV	幫你自動找出最佳參數組合
param_grid	指定你想測試的參數值
best_params_	查詢最好的參數
plot_boundary()	幫助你視覺化分類效果
cv=5	使用 5 折交叉驗證，避免過擬合

實作步驟：XGBoost 分類器

📘 XGBoost分類模型

- XGBoost 是「Extreme Gradient Boosting」的簡稱，是目前廣泛使用於競賽與實務的機器學習工具

✅ 若還沒安裝，可以執行以下指令安裝：

```
pip install xgboost
```

實作步驟：XGBoost 分類器

📘 建立 XGBoost 分類器（Classifier）物件

```
# 選擇演算法
import xgboost
algorithm = xgboost.XGBClassifier(random_state=random_seed)
```

- import xgboost → 匯入 xgboost 模組（這是一個功能強大、速度快、準確率高的梯度提升演算法套件）
- XGBClassifier 是一個針對分類任務（例如「貓或狗」）的模型
- 建立一個 XGBoost 分類器（Classifier）物件，並命名為 algorithm，之後用這個變數來訓練與預測
- random_state=random_seed：固定隨機種子，讓結果穩定、可重現

實作步驟：XGBoost 分類器

🛠️ XGBoost 常見可調參數

參數名稱	說明
n_estimators	弱分類器（樹）數量，預設為 100
learning_rate	每次學習的步長，預設為 0.3
max_depth	每棵樹的最大深度，避免過擬合
subsample	每棵樹訓練時隨機抽樣的資料比例
colsample_bytree	每棵樹訓練時使用特徵的比例

- 🔍 **範例：加強模型控制**

```
algorithm = xgboost.XGBClassifier(n_estimators=200, max_depth=4, learning_rate=0.1, random_state=random_seed)
```

實作步驟：XGBoost 分類器

🛠️ XGBoost 常見可調參數

```
# 顯示演算法的參數
print(algorithm)
```

- 印出目前模型的所有參數設定，有助於了解：
  - ✓ 模型是否使用了預設值
  - ✓ 是否已套用了自訂的參數

實作步驟：XGBoost 分類器

◆ 畫出模型在三組資料集上的分類效果（含邊界）

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

- 之前已定義好 **plot_boundaries()** 函式：
  1. 會將每筆資料分成訓練 / 測試集
  2. 用 **algorithm** 對訓練資料進行訓練
  3. 預測測試資料，計算準確率
  4. 將分類區域（**decision boundary**）畫出來
  5. 顯示測試與訓練資料的分佈

- 對每種資料集（線性、新月、圓形）都會畫出以下效果圖：
  - ✓ 背景顏色：表示模型判斷的分類區域
  - ✓ 🔵🔴 點點：實際的測試資料與類別
  - ✓ 訓練資料（x 標記）
  - ✓ 圖上會顯示模型準確率

實作步驟：XGBoost 分類器

✅ 總結

步驟	內容
1 匯入 xgboost 模組	使用前需先安裝
2 建立 XGBClassifier 模型物件	可設定參數如 max_depth, n_estimators
3 印出模型參數	確認設定
4 使用 plot_boundaries	可視化模型效果（分類區域、準確率）

實作步驟：XGBoost 分類器

🔍 延伸建議：可以進一步嘗試

✅ 調整 n_estimators 與 max_depth，看是否會過擬合或欠擬合

✅ 使用 GridSearchCV 尋找最佳參數

實作步驟：XGBoost 分類器

✅ 當使用 XGBoost 進行分類時，調整 n_estimators（決策樹的數量）和 max_depth（單棵樹的最大深度）可以影響模型的表現，進而影響是否會發生過擬合或欠擬合。

實作步驟：XGBoost 分類器

✅ 步驟一：匯入必要的庫

首先，需要導入相關的庫。**XGBoost 需要額外安裝，但可以使用 pip install xgboost 來安裝。**

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import xgboost as xgb
```

實作步驟：XGBoost 分類器

✅ 步驟二：生成分類資料集

將使用 **make_classification** 函數生成一個簡單的分類資料集，用來進行訓練與測試。

```
# 生成資料集
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
                           n_classes=2, random_state=42)

# 分割資料集為訓練集和測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

實作步驟：XGBoost 分類器

✅ 步驟三：訓練一個基本的 XGBoost 模型（無剪枝）

將從一個基本的 XGBoost 模型開始，這個模型的參數不會做太多調整，主要觀察其在訓練集和測試集上的表現。

```
# 創建並訓練基本的 XGBoost 模型
model_basic = xgb.XGBClassifier(n_estimators=100, max_depth=3, random_state=42)
model_basic.fit(X_train, y_train)

# 預測並計算準確度
y_pred_basic = model_basic.predict(X_test)
accuracy_basic = accuracy_score(y_test, y_pred_basic)
print(f"基本 XGBoost 模型的準確度: {accuracy_basic:.4f}")
```

- 這個模型使用了 n_estimators=100（決策樹的數量）和 max_depth=3（單棵樹的最大深度）。接下來，將觀察如何通過調整這兩個參數來控制模型的過擬合與欠擬合。

實作步驟：XGBoost 分類器

✅ 步驟四：調整 n_estimators 參數

將試著調整 n_estimators 參數，並觀察不同決策樹數量對模型表現的影響。

1. 測試 n_estimators=5：

```
# 創建並訓練 XGBoost 模型（n_estimators=5）
model_5_estimators = xgb.XGBClassifier(n_estimators=5, max_depth=3, random_state=42)
model_5_estimators.fit(X_train, y_train)

# 預測並計算準確度
y_pred_5_estimators = model_5_estimators.predict(X_test)
accuracy_5_estimators = accuracy_score(y_test, y_pred_5_estimators)
print(f"決策樹數量為 5 的 XGBoost 模型準確度: {accuracy_5_estimators:.4f}")
```

實作步驟：XGBoost 分類器

2. 測試 n_estimators=50：

```
# 創建並訓練 XGBoost 模型（n_estimators=50）
model_50_estimators = xgb.XGBClassifier(n_estimators=50, max_depth=3, random_state=42)
model_50_estimators.fit(X_train, y_train)

# 預測並計算準確度
y_pred_50_estimators = model_50_estimators.predict(X_test)
accuracy_50_estimators = accuracy_score(y_test, y_pred_50_estimators)
print(f"決策樹數量為 50 的 XGBoost 模型準確度: {accuracy_50_estimators:.4f}")
```

實作步驟：XGBoost 分類器

3. 測試 n_estimators=200：

```
# 創建並訓練 XGBoost 模型（n_estimators=200）
model_200_estimators = xgb.XGBClassifier(n_estimators=200, max_depth=3, random_state=42)
model_200_estimators.fit(X_train, y_train)

# 預測並計算準確度
y_pred_200_estimators = model_200_estimators.predict(X_test)
accuracy_200_estimators = accuracy_score(y_test, y_pred_200_estimators)
print(f"決策樹數量為 200 的 XGBoost 模型準確度: {accuracy_200_estimators:.4f}")
```

- 隨著 n_estimators 的增大，模型可能會變得更加強大，並且對訓練資料的擬合更好，但也有可能發生過擬合（過多的樹可能導致對噪聲的過度學習）。

實作步驟：XGBoost 分類器

✅ 步驟五：調整 max_depth 參數

將調整 max_depth 參數，來觀察樹的深度如何影響模型的表現。

1. 測試 max_depth=2：

```
# 創建並訓練 XGBoost 模型（max_depth=2）
model_max_depth_2 = xgb.XGBClassifier(n_estimators=100, max_depth=2, random_state=42)
model_max_depth_2.fit(X_train, y_train)

# 預測並計算準確度
y_pred_max_depth_2 = model_max_depth_2.predict(X_test)
accuracy_max_depth_2 = accuracy_score(y_test, y_pred_max_depth_2)
print(f"最大深度為 2 的 XGBoost 模型準確度: {accuracy_max_depth_2:.4f}")
```

實作步驟：XGBoost 分類器

2. 測試 max_depth=5：

```
# 創建並訓練 XGBoost 模型（max_depth=5）
model_max_depth_5 = xgb.XGBClassifier(n_estimators=100, max_depth=5, random_state=42)
model_max_depth_5.fit(X_train, y_train)

# 預測並計算準確度
y_pred_max_depth_5 = model_max_depth_5.predict(X_test)
accuracy_max_depth_5 = accuracy_score(y_test, y_pred_max_depth_5)
print(f"最大深度為 5 的 XGBoost 模型準確度: {accuracy_max_depth_5:.4f}")
```

實作步驟：XGBoost 分類器

3. 測試 max_depth=10：

```
# 創建並訓練 XGBoost 模型（max_depth=10）
model_max_depth_10 = xgb.XGBClassifier(n_estimators=100, max_depth=10, random_state=42)
model_max_depth_10.fit(X_train, y_train)

# 預測並計算準確度
y_pred_max_depth_10 = model_max_depth_10.predict(X_test)
accuracy_max_depth_10 = accuracy_score(y_test, y_pred_max_depth_10)
print(f"最大深度為 10 的 XGBoost 模型準確度: {accuracy_max_depth_10:.4f}")
```

實作步驟：XGBoost 分類器

✅ 步驟六：分析過擬合與欠擬合

- 欠擬合：當 n_estimators 或 max_depth 設置過小時，模型可能無法捕捉到數據中的規律，因此在訓練集和測試集上都會表現較差。這種情況下模型未能充分學習。
- 過擬合：當 n_estimators 或 max_depth 設置過大時，模型可能會過度擬合訓練數據，即模型學習了訓練數據中的噪聲，這樣在測試集上的表現會顯著下降。

實作步驟：XGBoost 分類器

✅ 小結

- **n_estimators**：決策樹的數量，對模型的性能有顯著影響。數量太少可能導致欠擬合，數量太多則可能導致過擬合。
- **max_depth**：單棵決策樹的最大深度，深度過深會使模型變得過於複雜並可能過擬合，過淺則可能欠擬合。
- 在實際應用中，可以通過交叉驗證和網格搜尋（GridSearchCV）來選擇最佳的參數組合。

實作步驟：XGBoost 分類器

進一步加入 交叉驗證 和 網格搜尋（GridSearchCV） 來選擇最佳的參數組合。這樣可以自動化選擇 n_estimators 和 max_depth 等參數，找到最佳的模型配置。

步驟 1：匯入必要的庫

在使用 GridSearchCV 之前，我們需要導入 GridSearchCV 和交叉驗證所需的庫。

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
import xgboost as xgb
```

實作步驟：XGBoost 分類器

步驟 2：設置模型和超參數搜尋範圍

接下來，會設定模型並設置 n_estimators 和 max_depth 的搜尋範圍。我們可以根據實際情況調整參數的範圍，這裡使用一個較為簡單的範圍。

```
# 設置模型
model = xgb.XGBClassifier(random_state=42)

# 設置需要搜尋的超參數範圍
param_grid = {
    'n_estimators': [50, 100, 200],    # 決策樹的數量
    'max_depth': [3, 5, 7]             # 單棵樹的最大深度
}
```

實作步驟：XGBoost 分類器

步驟 3：使用 GridSearchCV 進行超參數調整

使用 GridSearchCV 來搜尋最佳的參數組合。這裡的 cv=5 代表進行 5 折交叉驗證，scoring='accuracy' 代表是根據準確度來選擇最好的模型。

```
# 使用 GridSearchCV 進行網格搜尋
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           cv=5, scoring='accuracy', verbose=1, n_jobs=-1)

# 執行搜尋
grid_search.fit(X_train, y_train)

# 顯示最佳參數
print(f"最佳參數: {grid_search.best_params_}")
```

實作步驟：XGBoost 分類器

步驟 4：使用最佳參數訓練模型

獲得最佳參數之後，將使用這些參數來重新訓練模型，並在測試集上進行預測，最後計算準確度。

```
# 使用最佳參數訓練模型
best_model = grid_search.best_estimator_

# 預測並計算準確度
y_pred_best = best_model.predict(X_test)
accuracy_best = accuracy_score(y_test, y_pred_best)

# 顯示最佳模型的準確度
print(f"最佳模型的準確度: {accuracy_best:.4f}")
```

實作步驟：XGBoost 分類器

步驟 5：查看交叉驗證結果

GridSearchCV 也會記錄每次交叉驗證的結果，這有助於理解模型的表現如何隨著不同參數組合的變化而變化。

```
# 顯示每個參數組合的結果
print("所有參數組合的結果:")
print(grid_search.cv_results_)
```

步驟 6：分析與解釋結果

在執行上述程式碼後，GridSearchCV 會自動調整參數並找出最佳組合。通過查看交叉驗證的結果，可以了解哪些參數配置最適合該資料集。

實作步驟：XGBoost 分類器

✅ 小結

- GridSearchCV 是一種非常有用的工具，通過設置參數範圍並使用交叉驗證來選擇最佳的模型超參數。
- n_estimators 和 max_depth 是兩個常見的超參數，可以使用 GridSearchCV 來找到最佳的組合，從而提高模型的表現。
- 使用交叉驗證可以更有效地避免過擬合和欠擬合，並提供更可靠的評估結果。

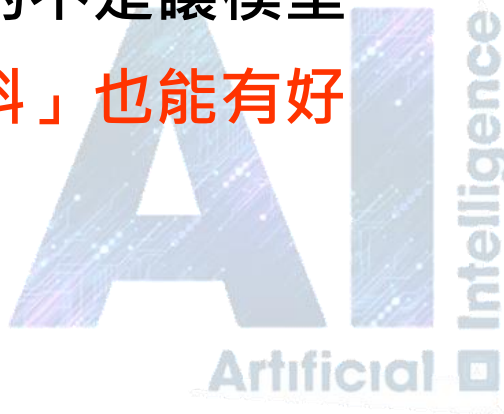
這樣的調整不僅能幫助我們挑選合適的超參數，也能讓我們的模型表現更加穩定和準確。

【補充說明】交叉驗證（Cross Validation）

✅ 交叉驗證（Cross Validation）是機器學習中非常關鍵的觀念，對於訓練出「真正有泛化能力」的模型至關重要

✅ 什麼是交叉驗證（Cross Validation）？

交叉驗證是一種評估機器學習模型泛化能力的方法，它的目的不是讓模型在「訓練資料」上表現好，而是確保模型對「未見過的新資料」也能有好表現。



【補充說明】交叉驗證（Cross Validation）

🎯 為什麼需要交叉驗證？

因為如果你只是把資料切成「訓練集」與「測試集」，那結果就可能會因為剛好切得好或不好而不穩定。

✅ 交叉驗證能：

- 減少過擬合風險
- 讓模型評估更穩定可信
- 幫助自動找出最佳參數組合（像 GridSearchCV 就會用交叉驗證）

【補充說明】交叉驗證（Cross Validation）

交叉驗證的運作方式（以 K-fold 為例）

最常見的交叉驗證是 K-fold Cross Validation，意思是：

把資料平均分成 K 等份（fold），每次拿其中 1 份做驗證，其餘 K-1 份做訓練。

然後重複 K 次，讓每一份資料都當過一次驗證集。

【補充說明】交叉驗證（Cross Validation）

📊 圖解範例（以 **K=5** 為例）：

次數	訓練集	驗證集
第1次	資料 2~5	資料 1
第2次	資料 1, 3~5	資料 2
第3次	資料 1, 2, 4, 5	資料 3
第4次	資料 1~3, 5	資料 4
第5次	資料 1~4	資料 5

然後將這5次的驗證準確率平均，就能得到更穩定的模型表現評

【補充說明】交叉驗證（Cross Validation）

幾折交叉驗證（K-Fold）怎麼選？

K 值	特點與使用時機
5-fold	**常用於一般模型訓練**，速度與穩定性平衡
10-fold	計算稍慢，但更穩定、更準
Leave-One-Out（LOOCV）	每次只留1筆做驗證，模型訓練非常多次但評估最精細（資料量很小時才用）

【補充說明】交叉驗證（Cross Validation）

📌 **GridSearchCV 的 cv=5 代表使用 5 折交叉驗證。**

💬 舉個生活化例子：

假設你要請 5 位老師來評你寫的作文，如果只讓一位老師評，可能會偏頗。但如果每位老師都評過一次，最後取平均分數，那麼這個分數就比較公平、可信了。

這就像是交叉驗證：不同分組去評你的模型準確率，最後平均分數才是你模型真正的「實力」！

【補充說明】交叉驗證（Cross Validation）

🧪 Python 實作範例（5-fold）

```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
scores = cross_val_score(model, X, y, cv=5)

print("5折交叉驗證準確率：", scores)
print("平均準確率：", scores.mean())
```

【補充說明】交叉驗證（Cross Validation）

✅ 總結：交叉驗證是什麼？

項目	說明
定義	將資料分成多份，**輪流訓練與驗證**，**平均得分**
用途	檢驗模型**是否過擬合**、**參數調整**
好處	穩定評估、避免運氣好壞影響結果
幾折好？	通常選用 5 或 10
哪些方法用到	**GridSearchCV**，**cross_val_score** 等

【補充說明】泛化能力

📌 機器學習中最核心的觀念之一：「泛化能力」

✅ 什麼是「泛化能力」？

泛化能力（Generalization Ability）是指：

👉 一個模型在新資料（不是訓練過的資料）上仍能做出準確預測的能力。

簡單來說：

✔ 訓練得好只是基本，

🎯 能預測沒看過的資料才是真功夫。

【補充說明】泛化能力

🧠 為什麼泛化能力很重要？

訓練模型的目的，不是讓它記住舊資料，而是：

🧪 未來面對新問題時，能正確做判斷！

🎯 泛化能力高 → 表示模型真的「學懂了」資料背後的邏輯規律。

【補充說明】泛化能力

🚫 過擬合 vs 欠擬合：與泛化能力的關係

狀況	描述	泛化能力
過擬合 **Overfitting**	模型「**學太細、記太多**」，甚至把訓練資料的雜訊也背下來了	❌ 泛化能力差 （在新資料表現爛）
欠擬合 **Underfitting**	模型**太簡單、學不夠**，連訓練資料都學不好	❌ 泛化能力差 （在舊資料、新資料都表現爛）
✅ **一般化** **Generalization**	模型**學到合理規則**、沒有過度記憶細節	✅ **泛化能力好** （在**新資料表現佳**）

【補充說明】泛化能力

🎨 圖像化理解（舉例可以想像一般學習狀況）：

- 欠擬合：像是一個不懂題目的人亂猜。
- 過擬合：像是死背解答的人，只會考古題，一換題型就不會。
- 一般化：像是理解題型的人，即使換題也能靈活解。

【補充說明】泛化能力

📊 評估泛化能力的方法

我們用交叉驗證（Cross Validation）或留出測試集來觀察：

若 “訓練準確率高，但測試準確率低” → 過擬合

若 “訓練準確率與測試準確率都低” → 欠擬合

若 “訓練與測試準確率差不多且都高” → 泛化能力佳

【補充說明】泛化能力

🧪 Python 範例：檢查泛化能力

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

model = RandomForestClassifier()
model.fit(X_train, y_train)

print("訓練集準確率：", model.score(X_train, y_train))
print("測試集準確率：", model.score(X_test, y_test))
```

【補充說明】泛化能力

🎯 小結

名詞	解釋
泛化能力	模型在新資料上的預測能力
過擬合	學太多，記太死，雜訊也記下來
欠擬合	學太少，無法掌握資料特徵
目標	訓練出泛化能力強的模型
解法	用交叉驗證、正則化、剪枝等技術避免過擬合

【補充說明】正則化（Regularization）

- ✅ 在訓練機器學習模型時，很容易遇到一個常見的問題—過擬合（Overfitting）。

- ✅ 過擬合（Overfitting），就是「模型在訓練資料上表現得很好，但在新資料（測試資料）上的表現卻很差」。

- ✅ 正則化（Regularization），就是一種用來避免過擬合的方法。

【補充說明】正則化（Regularization）

✅ 正則化（Regularization）的定義

正則化是在模型的損失函數（Loss Function）中加入一項「懲罰項（Penalty Term）」，來限制模型的複雜度，避免模型學得太精太過頭。

簡單來說，正則化就像是在「訓練模型時加上一點約束」，不要讓模型記太多沒必要的細節，以避免過擬合。

【補充說明】正則化（Regularization）

✅ 正則化的主要類型（以線性模型為例）

1. **L1 正則化（Lasso Regularization）**

   在損失函數中加入“所有參數絕對值之和（$\sum|w|$）”

   好處：可以讓部分權重變成 0 → 模型變得簡單、有特徵選擇的效果

2. **L2 正則化（Ridge Regularization）**

   在損失函數中加入“所有參數平方之和（$\sum w^2$）”

   好處：讓權重不會太大，讓模型穩定不誤判

【補充說明】正則化（Regularization）

✨ **L1 vs L2 差異簡單比較：**

項目	L1 正則化	L2 正則化
懲罰項	∑	w
權重效果	某些變數變成 0	所有變數較小但非 0
應用場景	特徵選擇	避免高變異、穩定性

【補充說明】正則化（Regularization）

✅ 為什麼正則化可以避免過擬合？

- 限制模型複雜度→ 不讓模型太自由地記住訓練資料所有細節
- 降低對訓練資料雜訊的敏感度→ 不會被少量異常值影響整體學習方向
- 提升泛化能力→ 模型能在未見過的新資料上也表現良好

【補充說明】正則化（Regularization）

✅ 實作方式（以邏輯斯迴歸為例）

在 sklearn 的 LogisticRegression 中，使用 C 參數來控制正則化強度

```
from sklearn.linear_model import LogisticRegression

# C 越小 → 正則化越強（限制越大）
model = LogisticRegression(C=0.01, penalty='l2')  # 使用 L2 正則化
model.fit(X_train, y_train)
```

penalty='l1' 使用 L1 正則化（需指定 solver）

C 是 正則化強度的倒數，數值越小，限制越強

【補充說明】正則化（Regularization）

✅ 如何避免過擬合？（除了正則化）

方法	說明
正則化（Regularization）	如上所述
資料擴充（Data Augmentation）	增加樣本數量、不同條件的資料
簡化模型	降低參數數量或模型深度
交叉驗證	評估模型是否泛化
提前停止（Early Stopping）	避免過多訓練輪次
丟棄法（Dropout）	對神經網路來說是一種正則化方式

「大數據資料分析實作」課程