內容

1. 專案簡介

本研究旨在對三種基礎但具有代表性的機器學習分類器——K-近鄰演算法 (KNN)、隨機森林 (Random Forest) 和支持向量機 (SVM)——進行系統性的性能評估。為了全面地測試這些模型，選用了四個來自 UCI 機器學習庫和 Scikit-learn 的公開數據集，涵蓋了二元分類與多類別分類、不同樣本規模及特徵維度的場景

本報告將詳細闡述實驗的設計、流程、所採用的評估指標，並對實驗結果進行深入的分析與比較，以期得出各分類器在不同任務下的適用性與相對優劣

2. 實驗方法

2.1 數據集

選用了以下四個數據集進行實驗：

1. **Breast Cancer Wisconsin (乳癌數據集):**
    - **類型**：二元分類
    - **任務**：根據 30 個從乳房腫塊細針穿刺數位影像中計算出的特徵，判斷其為惡性或良性
    - **特性**：特徵維度較高，樣本數較少 (569 筆)
2. **Banknote Authentication (鈔票鑑定數據集):**
    - **類型**：二元分類
    - **任務**：根據從鈔票影像小波轉換中提取的 4 個特徵，判斷其為真鈔或偽鈔
    - **特性**：特徵維度低，分類邊界清晰
3. **Digits Dataset (手寫數字數據集):**
    - **類型**：多類別分類 (10 類)
    - **任務**：辨識 8x8 像素的手寫數字圖片 (0-9)
    - **特性**：經典的多類別分類問題，特徵為 64 個像素值
4. **Dry Bean Dataset (乾豆數據集):**
    - **類型**：多類別分類(7 類)
    - **任務**：根據 16 種外觀形態特徵，將乾豆分為 7 個不同的品種
    - **特性**：樣本數最多 (約 13,611 筆)，類別較多，是本次實驗中最具挑戰性的數據集

2.2 分類器

1. **K-Nearest Neighbors (KNN)：**

   一種基於實例的非參數演算法，一個樣本的類別由其最近的 K 個鄰居的類別投票決定

2. **Random Forest (RF)：**

   一種集成學習方法，構建多個決策樹並將它們的預測結果進行集成（投票或平均），以獲得更準確、更穩定的預測，通常具有很好的抗過擬合能力

3. **Support Vector Machine (SVM)：**

   一種強大的監督學習模型，其目標是找到一個能將不同類別的數據點以最大間隔 (margin) 分開的超平面透過 kernel trick，SVM 也能高效地處理非線性問題

2.3 實驗流程

1. **數據預處理**

   在訓練每個模型前，對數據進行了**標準化(Standardization)** 處理，將所有特徵縮放到均值為 0、標準差為 1

   此步驟被封裝在 Scikit-learn 的 Pipeline 中，以確保標準化的參數僅從訓練集學習，避免數據洩漏

2. **超參數優化**

   使用 GridSearchCV 搭配 **5-Fold Cross-Validation** 來為每個分類器在每個數據集上尋找最佳的超參數組合，搜索的參數網格如下：

   - **KNN**：n_neighbors：[3, 5, 7]
   - **Random Forest**：n_estimators：[50, 100, 200]
   - **SVM**：C：[0.1, 1, 10], kernel：['linear', 'rbf']

3. **模型評估**

   - **主要指標**

     從 5-Fold Cross-Validation 中獲取每個最佳模型的**平均準確率 (Accuracy)、平均精確率 (Precision-Macro)、平均召回率 (Recall-Macro) 和平均 F1-Score (Macro)**

   - **輔助指標**

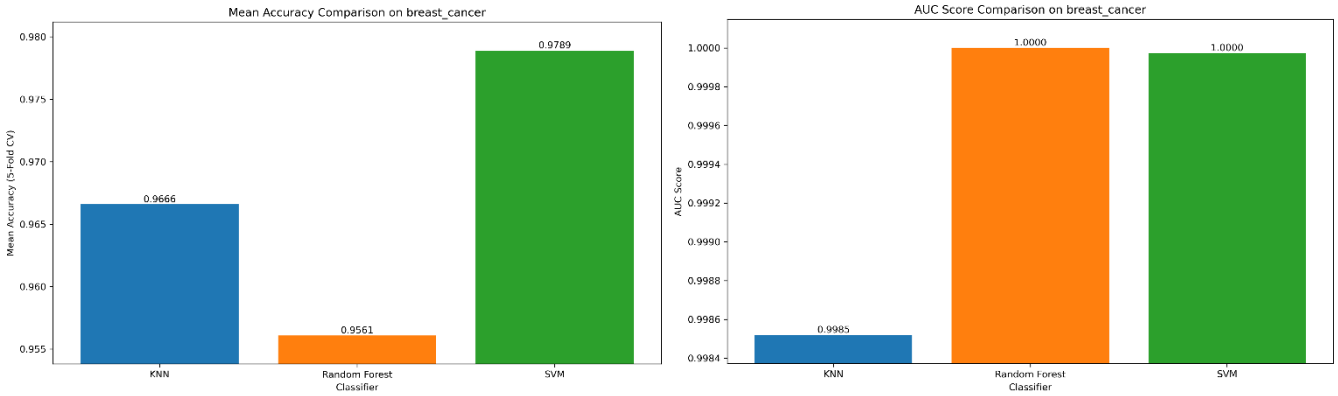
     在保留測試集 (一開始分割出去的那部分完全沒有參與訓練的 data) 上計算 **AUC (Area Under the ROC Curve)** 分數，並生成**混淆矩陣 (Confusion Matrix)** 以進行更深

入的錯誤分析

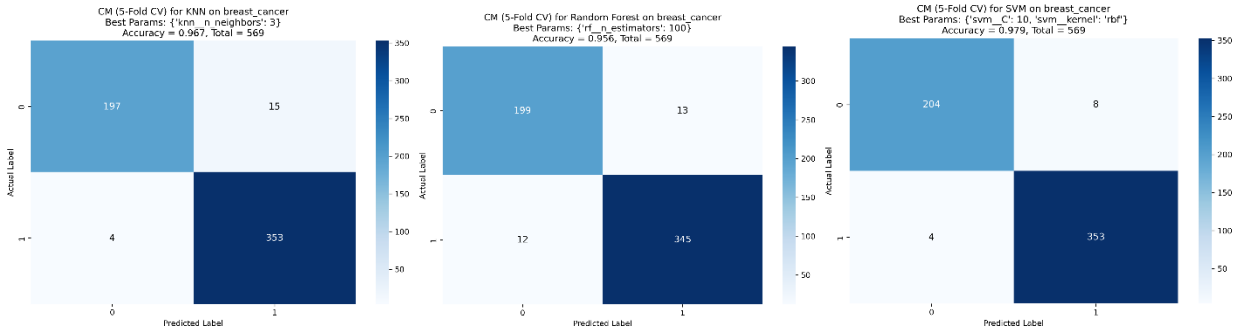
3. 實驗結果與分析

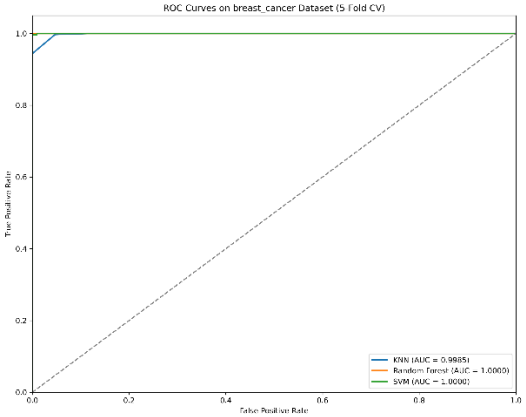
3.1 Breast Cancer 數據集 (二元分類)

在此數據集上，SVM 表現最為出色，特別是其線性核心 (kernel='linear') 版本取得了最高的平均準確率和 AUC 分數



以下為使用各個分類器，在各自的**最佳超參數**下得到的混淆矩陣：

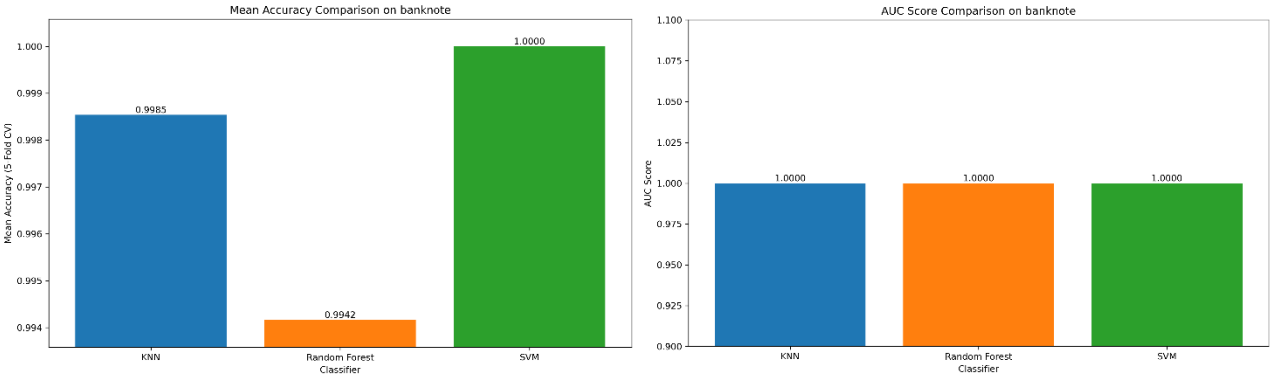




分析：線性 SVM 的勝出強烈暗示此數據集的特徵在經過標準化後，具有高度的線性可分性。KNN 和 Random Forest 也表現不俗，但 SVM 尋找最大間隔超平面的能力使其在此任務上略勝一籌

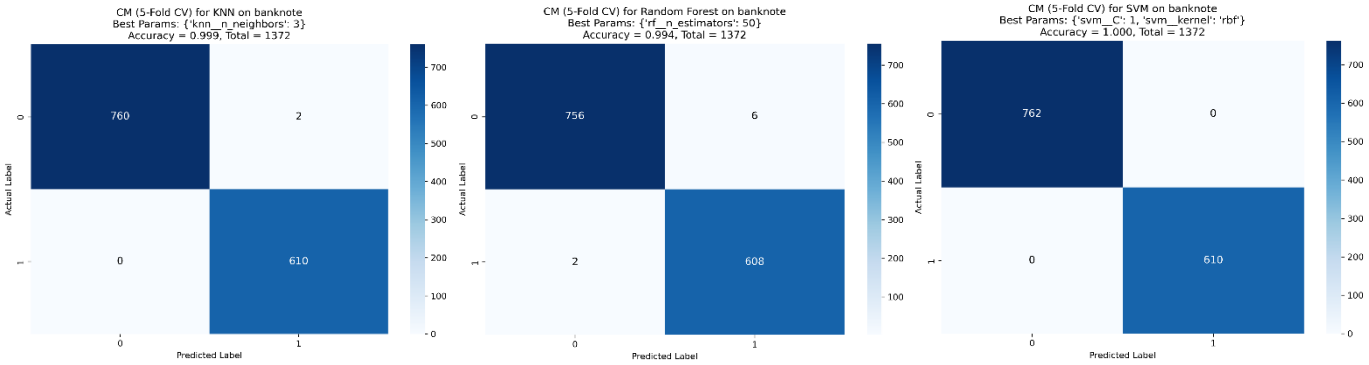
3.2 Banknote Authentication 數據集 (二元分類)

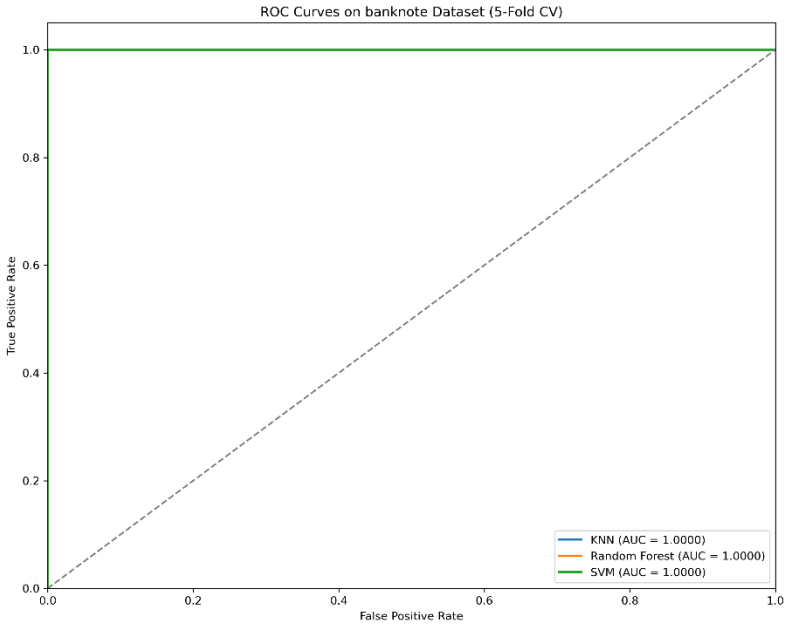
這是一個相對簡單的數據集，在 SVM 下曾在最佳超參數的配置下達到 100%的準確率



由於 AUC 本身評比的是**預測分數排序是否完美**，因此此處其他兩個分類器即使沒達到過 Accuracy=1，仍有 AUC=1 的分數

以下為使用各個分類器，在各自的**最佳超參數**下得到的混淆矩陣：





分析：此數據集的清晰可分性使得所有模型都表現優異。KNN 在此類低維度、結構清晰的問題上非常高效。SVM 同樣找到了完美的分類邊界。

補充：不過考慮到出現 accuracy = 1 本身是一個不正常的跡象，因此此處我進行過一番檢查

1. test data 混入 train data

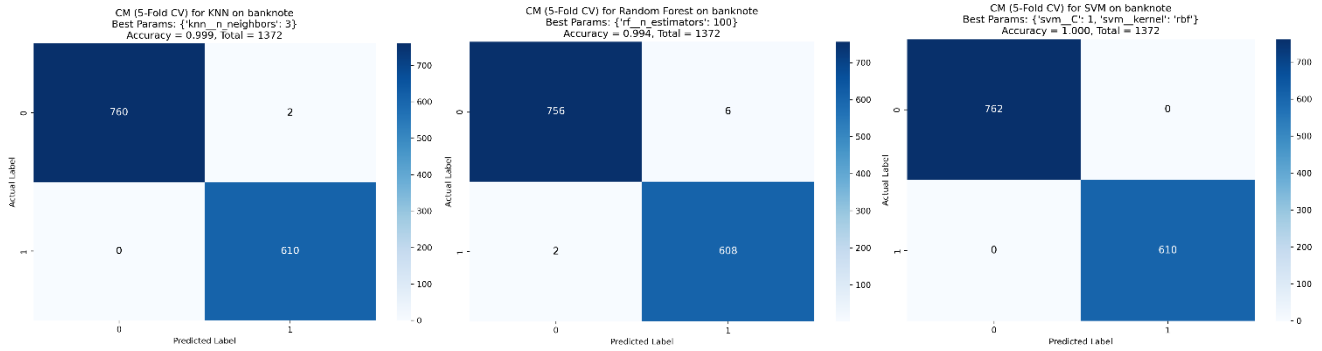
   這是我首先懷疑的，不過經過檢查 test data 並沒有混入 train data

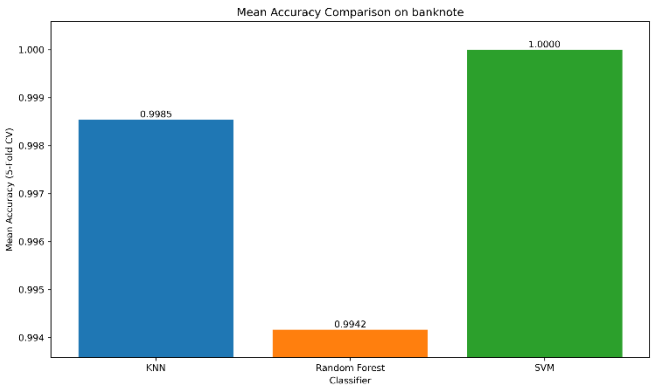
   在程式中我使用的 sklearn.model_selection 的 StratifiedKFold 進行資料分割，所有的 dataset 是用的都是同一套切割邏輯，但只有此數據集出現了 accuracy 為 1 的狀況

2. 更改初始化的 random seed

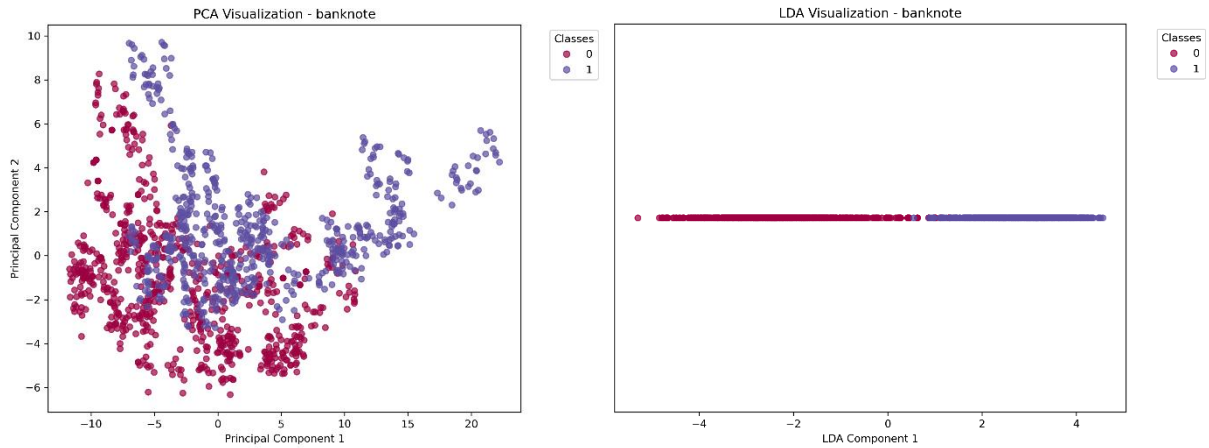
   另一個可能就是運氣很好真的撞到了（不過考慮到驗證的時候也是以 5-Fold Cross-Validation 進行，其實不太可能）

   因此我有嘗試更改種子為：random_state=133，結果仍舊完全一樣





3. 資料可視化



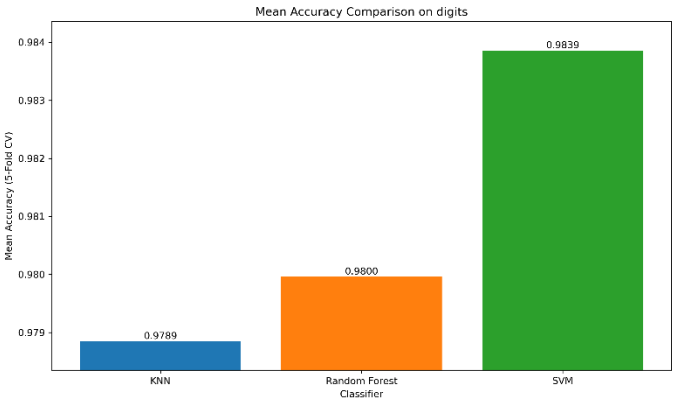
此處分別為資料集進行了 PCA 以及 LDA 的資料可視化。從 LDA 可以清晰地看出，紅色與藍色幾乎完全分開、中間重疊極少（幾乎沒有交錯區域）。也就是說，在這條線上模型可以找到一個分界點，使得兩類的機率分佈幾乎不重疊

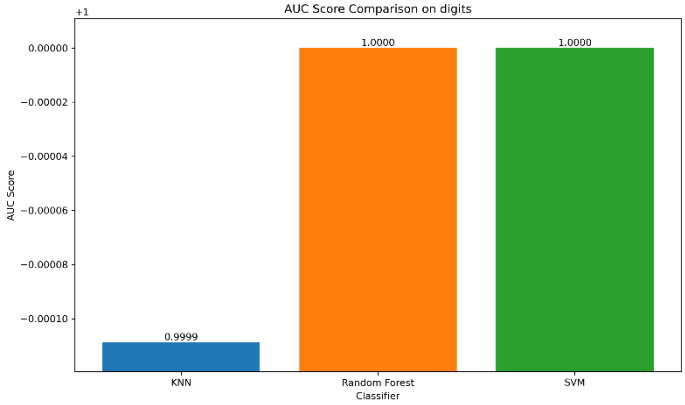
這證明了 Banknote 是一個幾乎**完美線性可分**的資料集，對於 **SVM 這種**學習一個穩定的「超平面」或「平滑邊界」的分類器而言，很容易就能找到能幾乎將資料集完美分割的邊界

個人認為，這是 SVM 在此資料集上能多次達到 accuracy=1 的原因

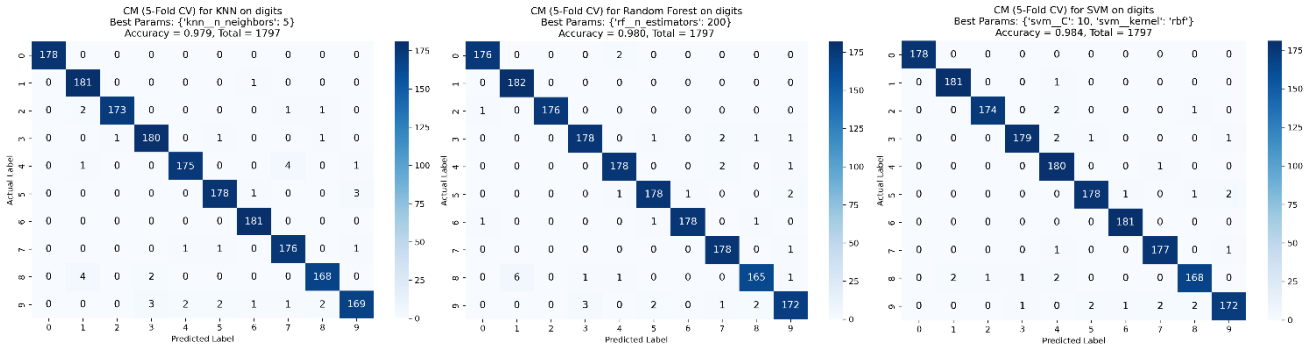
3.3 Digits 數據集 (多類別分類)

在手寫數字辨識這個多類別任務中，SVM 再次取得了最高的平均準確率和 AUC 分數





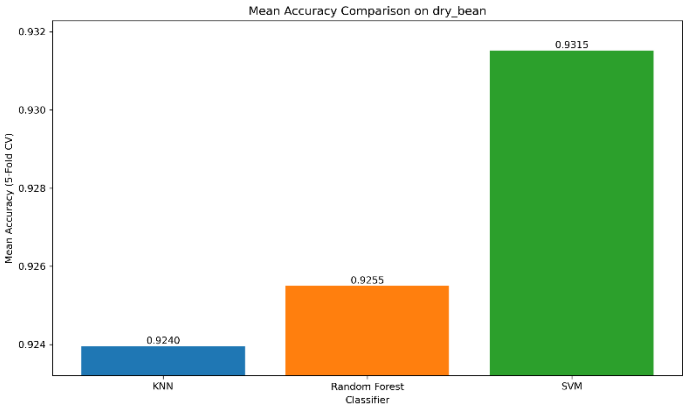
以下為使用各個分類器，在各自的**最佳超參數**下得到的混淆矩陣：

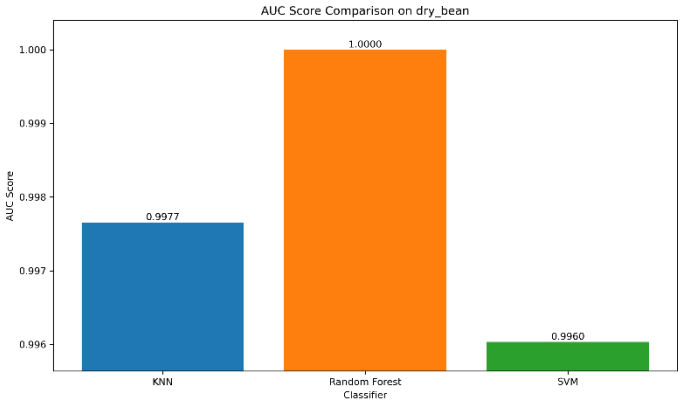


分析：所有三個分類器都表現出了強大的多類別分類能力，準確率均超過 97%。SVM 透過其核函數技巧，在處理 64 維像素特徵時展現了其優越性。值得注意的是，200 棵樹的 Random Forest 表現也極具競爭力，顯示了集成方法的效率

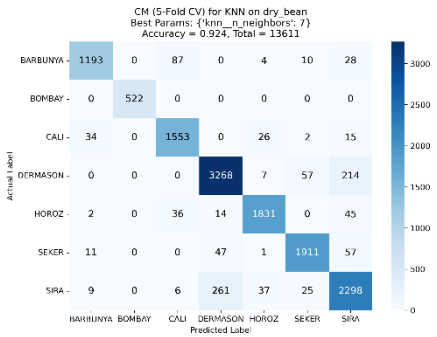
3.4 Dry Bean 數據集 (多類別分類)

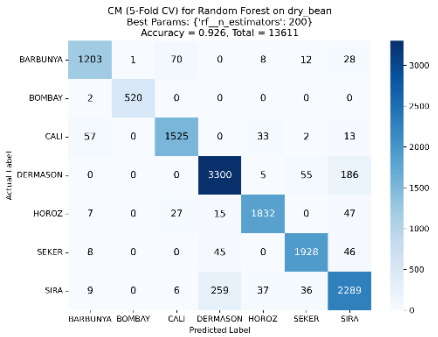
所有模型的性能都非常接近，SVM 最終以微弱的優勢在 ACC 指標上勝出

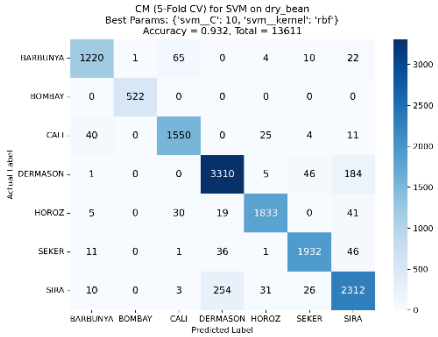




以下為使用各個分類器，在各自的**最佳超參數**下得到的混淆矩陣：







分析：

在這個樣本量大、類別多的複雜問題上，模型之間的差距被縮小。SVM (C=10, kernel='rbf') 表現最好，說明一個經過良好調整的非線性 SVM 在處理複雜、高維且有大量數據的問題時是強大的工具。所有模型的 Accuracy 分數均在 0.92-0.94 之間，表明它們在所有 7 個類別上都有相當均衡的表現

另外，值得關注的是，這三種分類器的原理並不相同，但它們在這份資料集上的**混淆矩陣分佈卻非常相似**

在混淆矩陣裡，可以看到幾個固定的混淆現象：

- **DERMASON ↔ SIRA**

  這兩種豆的物理形狀特徵最接近，模型常混淆，不管是距離（KNN）、樹分裂（RF）或超平面（SVM），都難以區分

- **BARBUNYA ↔ CALI**：

也是形狀或顏色類似的類別，屬於次要混淆對

- **BOMBAY**：
  幾乎完美分類，表示這個類別的特徵分佈非常獨立、清晰

再考慮到此資料集的特徵多為形狀、大小、顏色、紋理等連續值特徵，加上部分類別之間（例如 DERMASON vs SIRA）本身在物理外觀上就有部分重疊，可以推論不論使用哪種模型，只要它能捕捉到主要特徵結構，分類邊界可能就會很接近

因此模型雖然不同，但它們學到的決策邊界其實都在相同的資料分佈結構上，錯誤樣本也會重疊

4. 綜合結論

經過搭建並執行了完整的分類器比較流程。所有實驗的詳細數值結果總結如下：

dataset	classifier	best_params	mean_accuracy	mean_precision	mean_recall	mean_f1_score	auc
breast_cancer	KNN	{'knn__n_neighbors': 3}	0.9665890389691041	0.9703585040690303	0.9590428228284436	0.9636588598824034	0.998520162782094
breast_cancer	Random Forest	{'rf__n_estimators': 100}	0.9560937742586555	0.9557435041160833	0.9526707402034947	0.9528924697936147	1.0
breast_cancer	SVM	{'svm__C': 10, 'svm__kernel': 'rbf'}	0.9789163173420278	0.9799307432106413	0.9754488819220232	0.9772669033859678	0.9999735743353946
banknote	KNN	{'knn__n_neighbors': 3}	0.9985401459854014	0.9983739837398374	0.9986842105263157	0.998523607462787	1.0
banknote	Random Forest	{'rf__n_estimators': 50}	0.9941632382216323	0.993840347916362	0.9944175872822525	0.9940963445153678	1.0
banknote	SVM	{'svm__C': 1, 'svm__kernel': 'rbf'}	1.0	1.0	1.0	1.0	1.0
digits	KNN	{'knn__n_neighbors': 5}	0.9788502011761064	0.9794170504000224	0.9788219648219648	0.9787964332704355	0.9998912440201753
digits	Random Forest	{'rf__n_estimators': 200}	0.979962859795729	0.9806820738182968	0.9798818734701088	0.9799519376359956	1.0
digits	SVM	{'svm__C': 10, 'svm__kernel': 'rbf'}	0.9838594862271742	0.9842239109963569	0.983882882882883	0.983836601552334	1.0
dry_bean	KNN	{'knn__n_neighbors': 7}	0.923959154917036	0.9382790896970314	0.9344295812705029	0.9361396611031605	0.9976572093025303
dry_bean	Random Forest	{'rf__n_estimators': 200}	0.9255020570679516	0.9379431626428717	0.9345859964132149	0.9361462875861128	1.0
dry_bean	SVM	{'svm__C': 10, 'svm__kernel': 'rbf'}	0.9315265530006316	0.9439156881279256	0.9411645795147809	0.942449483893051	0.9960355447330907

總體觀察:

- **SVM 是其中表現最好的分類器：**在所有四個任務中，經過超參數優化的 SVM 均取得了最佳或並列最佳的性能

  這證明了它作為一個強大且靈活的 baseline model 的價值

- **沒有萬能模型：**

  雖然 SVM 表現最好，但其他模型在特定場景下也極具競爭力
  例如，KNN 在簡單問題上高效且準確；Random Forest 則提供了無需過多調參就能獲得的穩定、良好性能

- **超參數優化的重要性：**

  這裡可能看不太出來，不過再我使用不同的種子碼（42、133）時，會出現能夠達到最佳表現的的超參數組合出現變化的情況。這凸顯了超參數搜索對於發揮模型全部潛力的關鍵作用

附錄

程式碼架構

- main.py: 主執行腳本，負責協調整個實驗流程，包括數據加載、模型訓練、超參數搜索和評估。
- utils.py: 工具模組，提供加載所有數據集的統一接口。
- visualize.py： 負責資料集的資料分佈視覺化（PCA、LDA）。
- classifiers.py: 分類器模組，封裝了所有分類器，並內建了數據標準化 Pipeline。
- evaluation.py: 評估模組，提供繪製混淆矩陣、ROC 曲線和性能比較長條圖的功能。
- requirements.txt: 專案的 Python 依賴包列表。
- data/: 存放本地數據集的資料夾。
- plots/: 存放所有生成圖表的資料夾。
- Readme.md: 本說明檔案，記錄專案細節與成果。

環境建立與如何運行

1. **安裝依賴:** 本專案的所有 Python 依賴都記錄在 requirements.txt 中。請運行以下指令進行安裝

   ```
   pip install -r requirements.txt
   ```

2. **準備數據**
   - Breast Cancer Wisconsin 數據集會自動從網路下載
   - Banknote Authentication 需手動下載（放在 data 資料夾）：UCI 連結
   - Digits Dataset 數據集會自動從網路下載
   - Dry Bean Dataset 需手動下載（放在 data 資料夾）：UCI 連結
3. **執行實驗:** 所有實驗流程都已整合到 main.py 中。直接運行此腳本即可：

   ```
   python main.py
   ```

腳本會自動執行所有數據集的超參數搜索、模型評估、生成所有圖表至 plots/ 資料夾、匯出 results_summary.csv。

Main

```
import numpy as np

import pandas as pd

import warnings

import os

from utils import load_dataset

from classifiers import BaseClassifierWrapper, KNeighborsClassifierWrapper, RandomForestClassifierWrapper, SupportVectorMachineWrapper

from evaluation import plot_confusion_matrix, plot_roc_curves, plot_metric_comparison

from sklearn.model_selection import GridSearchCV, StratifiedKFold, cross_val_predict

from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix

from visualize import visualize_pca, visualize_lda


warnings.filterwarnings("ignore", category=UserWarning)


def main():
    """主函式：使用 5-Fold Cross Validation 執行所有數據集的模型搜尋與評估"""

    os.makedirs('plots', exist_ok=True)

    dataset_names = ['breast_cancer', 'banknote', 'digits', 'dry_bean']

    final_results = []

    for dataset_name in dataset_names:
        print(f"\n{'='*50}")
        print(f"處理數據集: {dataset_name}")
```

```
print(f"{'='*50}")


# === 載入資料 ===
X, y = load_dataset(dataset_name)
visualize_pca(X, y, dataset_name)
visualize_lda(X, y, dataset_name)
if X is None:
    continue


is_binary = len(np.unique(y)) == 2


# === 參數設定 ===
param_grids = {
    "KNN": {'knn__n_neighbors': [3, 5, 7]},
    "Random Forest": {'rf__n_estimators': [50, 100, 200]},
    "SVM": {'svm__C': [0.1, 1, 10], 'svm__kernel': ['linear', 'rbf']}
}


classifiers = {
    "KNN": KNeighborsClassifierWrapper(),
    "Random Forest": RandomForestClassifierWrapper(random_state=42),
    "SVM": SupportVectorMachineWrapper(random_state=42)
}


roc_results_for_dataset = []
metrics_for_dataset = {'Accuracy': {}, 'AUC': {}}
```

```
# === Cross-validation 分割設定 ===
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, clf_wrapper in classifiers.items():
    print(f"\n--- 為 {name} 執行 5-Fold Cross Validation on {dataset_name} ---")

    scoring = {
        'accuracy': 'accuracy',
        'precision': 'precision_macro',
        'recall': 'recall_macro',
        'f1_score': 'f1_macro'
    }

    # GridSearchCV for parameter tuning
    grid_search = GridSearchCV(
        clf_wrapper.model,
        param_grids[name],
        cv=cv,
        scoring=scoring,
        refit='accuracy',
        n_jobs=-1
    )
    grid_search.fit(X, y)
    best_model = grid_search.best_estimator_
    print(f"找到的最佳參數: {grid_search.best_params_}")

    # Cross-validation 預測（用於 Confusion Matrix & ROC）
```

```
            y_pred = cross_val_predict(best_model, X, y, cv=cv)
            y_proba, _ = BaseClassifierWrapper._get_scores(best_model, X)

            # === 平均 CV 結果 ===
            mean_accuracy = grid_search.cv_results_['mean_test_accuracy'][grid_search.best_index_]
            mean_precision = grid_search.cv_results_['mean_test_precision'][grid_search.best_index_]
            mean_recall = grid_search.cv_results_['mean_test_recall'][grid_search.best_index_]
            mean_f1 = grid_search.cv_results_['mean_test_f1_score'][grid_search.best_index_]

            auc_score = None
            if y_proba is not None:
                if is_binary:
                    auc_score = roc_auc_score(y, y_proba[:, 1])
                    roc_results_for_dataset.append((y, y_proba[:, 1], name))
                else:
                    auc_score = roc_auc_score(y, y_proba, multi_class='ovr', average='macro')

            print(f"平均 CV 準確率: {mean_accuracy:.4f}，精確率: {mean_precision:.4f}，召回率: {mean_recall:.4f}, F1: {mean_f1:.4f}")
            if auc_score is not None:
                print(f"整體資料 AUC: {auc_score:.4f}")

            final_results.append({
                'dataset': dataset_name, 'classifier': name,
                'best_params': str(grid_search.best_params_),
                'mean_accuracy': mean_accuracy,
                'mean_precision': mean_precision,
```

```
                'mean_recall': mean_recall,
                'mean_f1_score': mean_f1,
                'auc': auc_score
            })
            metrics_for_dataset['Accuracy'][name] = mean_accuracy
            if auc_score is not None:
                metrics_for_dataset['AUC'][name] = auc_score

            # === 繪製混淆矩陣 ===
            cm = confusion_matrix(y, y_pred)
            cm_title = f'CM (5-Fold CV) for {name} on {dataset_name}\nBest Params: {grid_search.best_params_}'
            cm_save_path = f'plots/CM_{name}_{dataset_name}.png'
            plot_confusion_matrix(y, y_pred, title=cm_title, save_path=cm_save_path)

        # === 繪製 ROC 與比較圖 ===
        if is_binary:
            plot_roc_curves(roc_results_for_dataset, title=f'ROC Curves on {dataset_name} Dataset (5-Fold CV)',
                            save_path=f'plots/ROC_{dataset_name}.png')

        plot_metric_comparison(metrics_for_dataset['Accuracy'], metric_name='Mean Accuracy (5-Fold CV)',
                               title=f'Mean Accuracy Comparison on {dataset_name}', save_path=f'plots/ACC_BAR_{dataset_name}.png')

        if metrics_for_dataset['AUC']:
            plot_metric_comparison(metrics_for_dataset['AUC'], metric_name='AUC Score',
```

```
                                                    title=f'AUC Score Comparison on {dataset_name}',
save_path=f'plots/AUC_BAR_{dataset_name}.png')


    # ===  匯出結果  ===
    print(f"\n\n{'='*50}")
    print("所有實驗已完成 - 匯出結果摘要")
    print(f"{'='*50}")

    results_df = pd.DataFrame(final_results)
    results_df.to_csv('results_summary.csv', index=False)
    print("結果摘要已儲存至  results_summary.csv")

if __name__ == "__main__":
    main()
```

classifiers

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

class BaseClassifierWrapper:
    """提供一個通用的方法來獲取模型的預測分數"""
    @staticmethod
```

```
    def _get_scores(model, X):
        """優先嘗試 decision_function，其次是 predict_proba（回傳每個類別的機率）。"""
        # 先嘗試取得原生的分數，若不行再嘗試獲取每個類別的機率
        decision_values = None
        proba_values = None
        if hasattr(model, 'decision_function'):
            try:
                decision_values = model.decision_function(X)
            except Exception:
                pass # 忽略錯誤
        if hasattr(model, 'predict_proba'):
            try:
                proba_values = model.predict_proba(X)
            except Exception:
                pass # 忽略錯誤
        return proba_values, decision_values


class KNeighborsClassifierWrapper(BaseClassifierWrapper):
    """KNN 分類器的封裝。"""
    def __init__(self, n_neighbors=5):
        self.model = Pipeline([
            ('scaler', StandardScaler()),
            ('knn', KNeighborsClassifier(n_neighbors=n_neighbors))
        ])

    def train(self, X_train, y_train):
        self.model.fit(X_train, y_train)
```

```
    def test(self, X_test):
        y_pred = self.model.predict(X_test)
        proba, dec = self._get_scores(self.model, X_test)
        return y_pred, proba, dec

class RandomForestClassifierWrapper(BaseClassifierWrapper):
    """Random Forest 分類器的封裝。"""
    def __init__(self, n_estimators=100, random_state=42):
        self.model = Pipeline([
            ('scaler', StandardScaler()),
            ('rf', RandomForestClassifier(n_estimators=n_estimators, random_state=random_state))
        ])
    def train(self, X_train, y_train):
        self.model.fit(X_train, y_train)

    def test(self, X_test):
        y_pred = self.model.predict(X_test)
        proba, dec = self._get_scores(self.model, X_test)
        return y_pred, proba, dec

class SupportVectorMachineWrapper(BaseClassifierWrapper):
    """SVM 分類器的封裝。"""
    def __init__(self, C=1.0, kernel='rbf', probability=True, random_state=42):
        # 註：probability=True 讓 SVC 能夠使用 predict_proba，但會增加訓練時間。
        self.model = Pipeline([
            ('scaler', StandardScaler()),
```

```
            ('svm', SVC(C=C, kernel=kernel, probability=probability, random_state=random_state))
        ])
    def train(self, X_train, y_train):
        self.model.fit(X_train, y_train)

    def test(self, X_test):
        y_pred = self.model.predict(X_test)
        proba, dec = self._get_scores(self.model, X_test)
        return y_pred, proba, dec
```

utils

```
import os
import pandas as pd
from sklearn.datasets import load_breast_cancer, load_digits

def load_dataset(name: str):
    """
    根據名稱載入指定的數據集，採用本地優先策略。

    Args:
        name (str):  數據集名稱。可選值:
                        'breast_cancer', 'digits', 'banknote', 'dry_bean'

    Returns:
        tuple: (X, y) or (None, None) if loading fails.
                X:  特徵數據  (np.ndarray)
```

```
            y: 標籤 (np.ndarray)
    """
    if name == 'breast_cancer':
        data = load_breast_cancer()
        return data.data, data.target

    elif name == 'digits':
        data = load_digits()
        return data.data, data.target

    elif name == 'banknote':
        local_path = 'data/data_banknote_authentication.txt'
        if os.path.exists(local_path):
            print(f"從本地路徑加載 '{name}' 數據集: {local_path}")
            df = pd.read_csv(local_path, header=None)
        else:
            url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/00267/data_banknote_authentication.txt'
            print(f"本地檔案未找到，嘗試從網路後備連結加載: {url}")
            try:
                df = pd.read_csv(url, header=None)
            except Exception as e:
                print(f"無法從網路讀取 banknote 數據，請手動下載至 '{local_path}'。錯誤: {e}")
                return None, None
        X = df.iloc[:, :-1].values
        y = df.iloc[:, -1].values
        return X, y
```

```
    elif name == 'dry_bean':
        # 根據建議，我們預期用戶已將解壓縮後的檔案放在 data/DryBeanDataset/ 目錄下
        local_path = 'data/DryBeanDataset/Dry_Bean_Dataset.xlsx'
        if not os.path.exists(local_path):
            print(f"錯誤: 找不到 '{local_path}'。")
            print("請確認您已手動從 Kaggle 下載數據集，並將其解壓縮後的 .xlsx 檔案放置在正確的路徑中。")
            return None, None

        print(f"從本地路徑加載 '{name}' 數據集: {local_path}")
        df = pd.read_excel(local_path)
        X = df.iloc[:, :-1].values
        y = df.iloc[:, -1].values
        return X, y

    else:
        raise ValueError(f"未知的數據集名稱: {name}。請從 'breast_cancer', 'digits', 'banknote', 'dry_bean' 中選擇。")
```

evaluation

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
from sklearn.metrics import confusion_matrix, roc_curve, auc

# ==========================================================
```

```
#   混淆矩陣繪圖
# ================================================================
def plot_confusion_matrix(y_true, y_pred, title: str, save_path: str):
    """
    計算並繪製混淆矩陣，並將其儲存為圖片檔案。
    支援任意類別數量與格式（含字串標籤）。
    """
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    labels = np.unique(np.concatenate((y_true, y_pred)))
    cm = confusion_matrix(y_true, y_pred, labels=labels)

    fig, ax = plt.subplots(figsize=(8, 6))
    sns.heatmap(cm, cmap="Blues", cbar=True, ax=ax,
                    xticklabels=labels, yticklabels=labels)

    # ===  手動標註每格  ===
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            value = cm[i, j]
            color = "white" if value > cm.max() / 2 else "black"
            ax.text(j + 0.5, i + 0.5, f"{value:d}",
                        ha="center", va="center", color=color, fontsize=12)

    # ===  額外資訊  ===
    acc = np.trace(cm) / np.sum(cm)
    total = np.sum(cm)
```

```
    ax.set_title(f"{title}\nAccuracy = {acc:.3f}, Total = {total}")
    ax.set_ylabel("Actual Label")
    ax.set_xlabel("Predicted Label")

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    plt.close()
    print(f"混淆矩陣已儲存至: {save_path}")

# ============================================================
#   ROC 曲線繪圖
# ============================================================
def plot_roc_curves(results, title: str, save_path: str):
    """
    在同一張圖上繪製多個分類器的 ROC 曲線，並儲存為檔案。
    results 格式為 [(y_true, y_score, name), ...]
    """
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    plt.figure(figsize=(10, 8))

    for y_true, y_scores, name in results:
        fpr, tpr, _ = roc_curve(y_true, y_scores)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.4f})')

    plt.plot([0, 1], [0, 1], color='gray', lw=1.5, linestyle='--')
    plt.xlim([0.0, 1.0])
```

```
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right", fontsize=10)

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    plt.close()
    print(f" ROC  曲線圖已儲存至: {save_path}")

# ============================================================
#   單一指標比較長條圖
# ============================================================
def plot_metric_comparison(results, metric_name: str, title: str, save_path: str):
    """
    繪製不同分類器在同一指標下的比較圖。
    results = {'SVM': 0.98, 'KNN': 0.95, ...}
    """
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    names = list(results.keys())
    scores = list(results.values())

    plt.figure(figsize=(10, 6))
    bars = plt.bar(names, scores, color=['#1f77b4', '#ff7f0e', '#2ca02c'])

    plt.xlabel('Classifier')
```

```
    plt.ylabel(metric_name)
    plt.title(title)

    # 動態調整 Y 軸範圍
    min_score, max_score = min(scores), max(scores)
    if min_score == max_score:
        plt.ylim([min_score * 0.95 - 0.05, max_score * 1.05 + 0.05])
    else:
        plt.ylim([min_score - (max_score - min_score) * 0.1,
                  max_score + (max_score - min_score) * 0.1])

    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2.0, yval, f'{yval:.4f}',
                 ha='center', va='bottom', fontsize=10)

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches="tight")
    plt.close()
    print(f" {metric_name} 比較圖已儲存至: {save_path}")

# ============================================================
#  Cross-Validation 穩定性圖（新功能）
# ============================================================
def plot_cv_stability_bar(results_dict, metric_name: str, title: str, save_path: str):
    """
    繪製不同分類器的 Cross-Validation 平均分數 ± 標準差。
```

```
results_dict = {
    'SVM': (mean, std),
    'KNN': (mean, std),
    ...
}
"""
os.makedirs(os.path.dirname(save_path), exist_ok=True)

names = list(results_dict.keys())
means = [v[0] for v in results_dict.values()]
stds = [v[1] for v in results_dict.values()]

plt.figure(figsize=(10, 6))
bars = plt.bar(names, means, yerr=stds, capsize=6,
                color=['#1f77b4', '#ff7f0e', '#2ca02c'], alpha=0.85)
plt.xlabel("Classifier")
plt.ylabel(metric_name)
plt.title(title)

for i, bar in enumerate(bars):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + stds[i],
             f"{means[i]:.4f}±{stds[i]:.4f}", ha="center", va="bottom", fontsize=9)

plt.tight_layout()
plt.savefig(save_path, dpi=300, bbox_inches="tight")
plt.close()
print(f" {metric_name} 穩定性比較圖已儲存至: {save_path}")
```

visualize

```
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import os
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
PLOTS_DIR = os.path.join(BASE_DIR, 'plots')
os.makedirs(PLOTS_DIR, exist_ok=True)

def visualize_pca(X, y, dataset_name):
    """將資料降維至 2D 並以 PCA 可視化，圖片儲存在 plots 資料夾中"""
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)

    # ===  新增：將字串標籤轉成整數  ===
    if y.dtype == object or isinstance(y[0], str):
        le = LabelEncoder()
        y_encoded = le.fit_transform(y)
        class_names = le.classes_
    else:
        y_encoded = y
        class_names = np.unique(y)
```

```
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(
        X_pca[:, 0], X_pca[:, 1],
        c=y_encoded, cmap='Spectral', alpha=0.7, s=30
    )
    plt.title(f"PCA Visualization - {dataset_name}")
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")

    # 新增：顯示圖例（每個顏色對應的類別）
    handles, _ = scatter.legend_elements()
    plt.legend(handles, class_names, title="Classes", bbox_to_anchor=(1.05, 1), loc='upper left')

    # 儲存圖
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    save_dir = os.path.join(BASE_DIR, 'plots')
    os.makedirs(save_dir, exist_ok=True)
    save_path = os.path.join(save_dir, f"PCA_{dataset_name}.png")
    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()
    print(f"PCA 圖已儲存至: {save_path}")

def visualize_lda(X, y, dataset_name: str):
    """
    使用 LDA（Linear Discriminant Analysis）進行有監督降維，
```

並將結果以散點圖可視化，輸出至 plots 資料夾。

Parameters

X : np.ndarray

    特徵矩陣

y : np.ndarray

    標籤向量（可為數字或字串）

dataset_name : str

    資料集名稱（將用於檔名與標題）

"""

=== 確保標籤是數字 ===

if y.dtype == object or isinstance(y[0], str):

    le = LabelEncoder()

    y_encoded = le.fit_transform(y)

    class_names = le.classes_

else:

    y_encoded = y

    class_names = np.unique(y)

n_classes = len(np.unique(y_encoded))

=== LDA 降維 ===

n_components = 2 if n_classes > 2 else 1

lda = LDA(n_components=n_components)

X_lda = lda.fit_transform(X, y_encoded)

```
# ===  畫圖  ===
plt.figure(figsize=(8, 6))
if n_components == 2:
    scatter = plt.scatter(
        X_lda[:, 0], X_lda[:, 1],
        c=y_encoded, cmap="Spectral", alpha=0.7, s=30
    )
    plt.xlabel("LDA Component 1")
    plt.ylabel("LDA Component 2")
else:
    scatter = plt.scatter(
        X_lda[:, 0], np.zeros_like(X_lda),
        c=y_encoded, cmap="Spectral", alpha=0.7, s=30
    )
    plt.xlabel("LDA Component 1")
    plt.yticks([])

plt.title(f"LDA Visualization - {dataset_name}")

#  圖例
handles, _ = scatter.legend_elements()
plt.legend(handles, class_names, title="Classes",
           bbox_to_anchor=(1.05, 1), loc='upper left')

# ===  儲存圖像  ===
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
save_dir = os.path.join(BASE_DIR, 'plots')
```

```
    os.makedirs(save_dir, exist_ok=True)
    save_path = os.path.join(save_dir, f"LDA_{dataset_name}.png")

    plt.tight_layout()
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()

    print(f"LDA 圖已儲存至: {save_path}")
```

GitHub

GitHub：https://github.com/megrez33281/classifiers-experiment