內容

[1. 專案簡介 2](#_Toc210864138)

[2. 實驗方法 2](#_Toc210864139)

[2.1 數據集 2](#_Toc210864140)

[2.2 分類器 3](#_Toc210864141)

[2.3 實驗流程 3](#_Toc210864142)

[3. 實驗結果與分析 4](#_Toc210864143)

[3.1 Breast Cancer 數據集 (二元分類) 4](#_Toc210864144)

[3.2 Banknote Authentication 數據集 (二元分類) 5](#_Toc210864145)

[3.3 Digits數據集 (多類別分類) 7](#_Toc210864146)

[3.4 Dry Bean 數據集 (多類別分類) 8](#_Toc210864147)

[4. 綜合結論 9](#_Toc210864148)

[附錄 10](#_Toc210864149)

[程式碼架構 10](#_Toc210864150)

[環境建立與如何運行 11](#_Toc210864151)

[Main 11](#_Toc210864152)

[classifiers 17](#_Toc210864153)

[utils 19](#_Toc210864154)

[evaluation 22](#_Toc210864155)

[visualize 27](#_Toc210864156)

[GitHub 32](#_Toc210864157)

# 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

1. **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

1. **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

1. **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']

1. **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)**

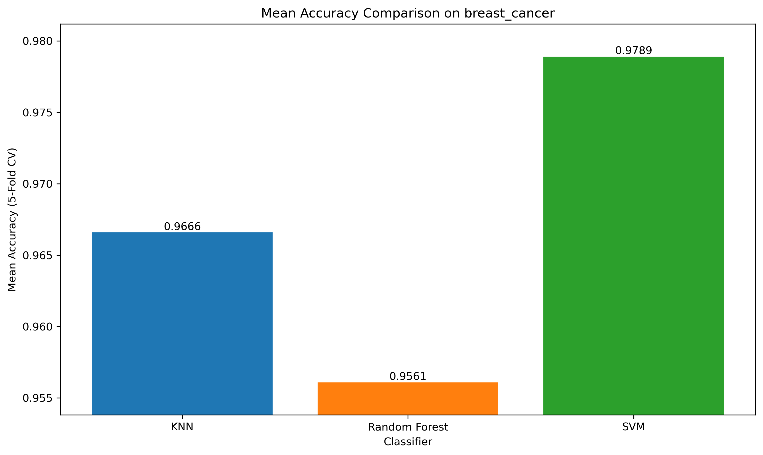
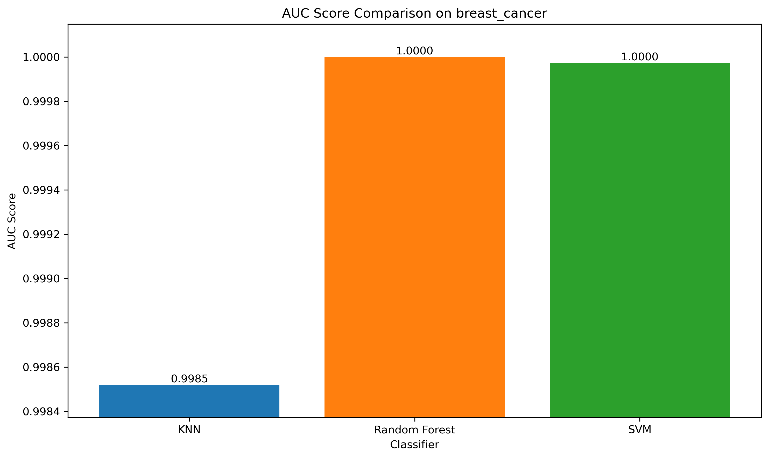
* + **輔助指標**

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

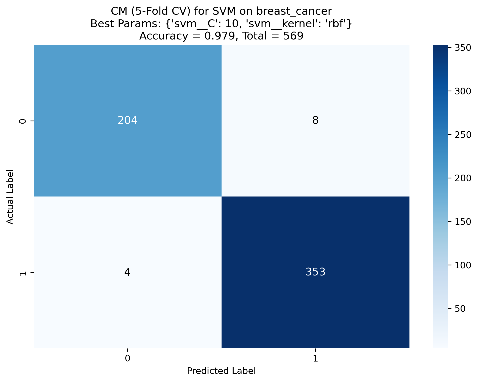
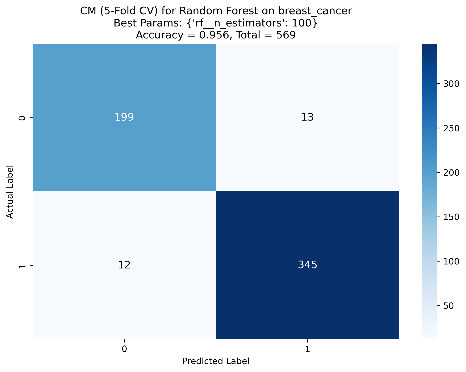
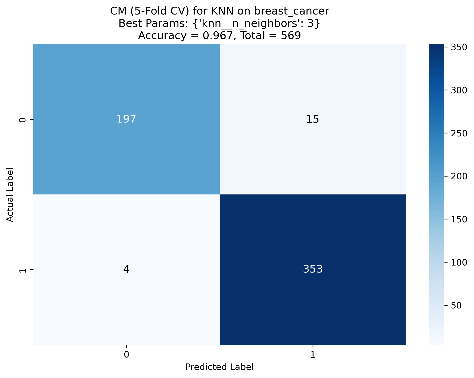
# 3. 實驗結果與分析

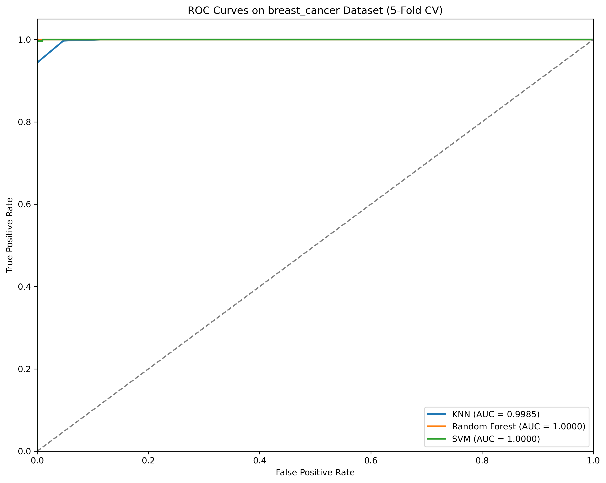
## 3.1 Breast Cancer 數據集 (二元分類)

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

[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/ACC_BAR_breast_cancer.png)[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/AUC_BAR_breast_cancer.png)

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：

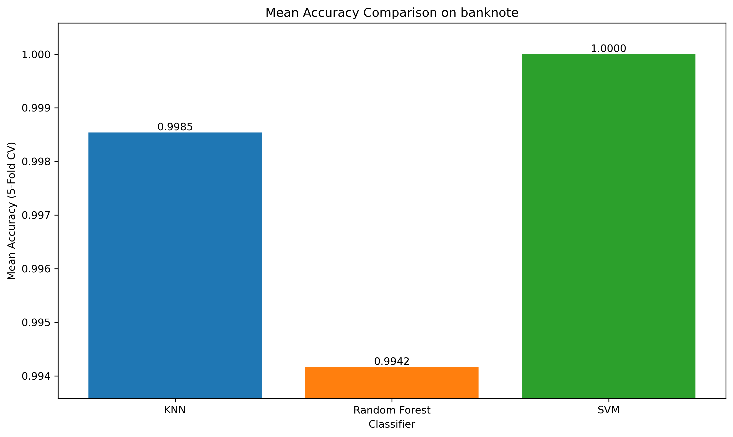
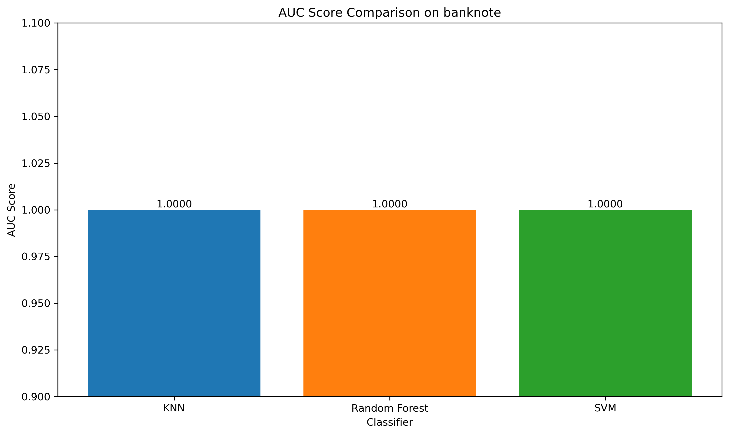


[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/ROC_breast_cancer.png)

**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

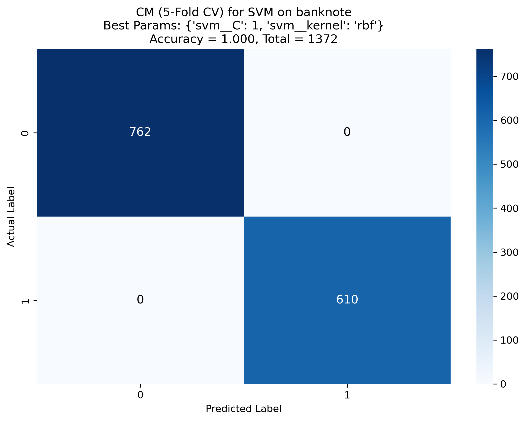
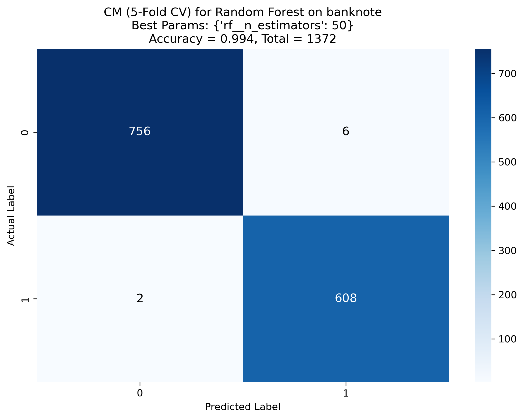
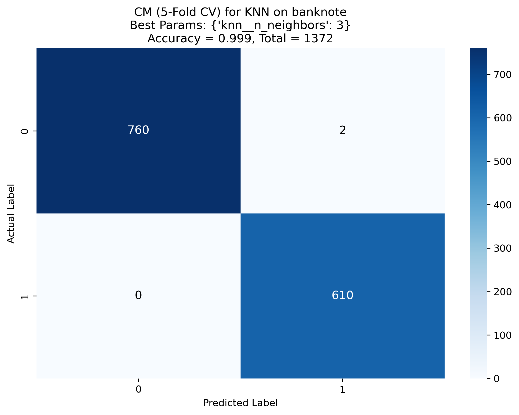
## 3.2 Banknote Authentication 數據集 (二元分類)

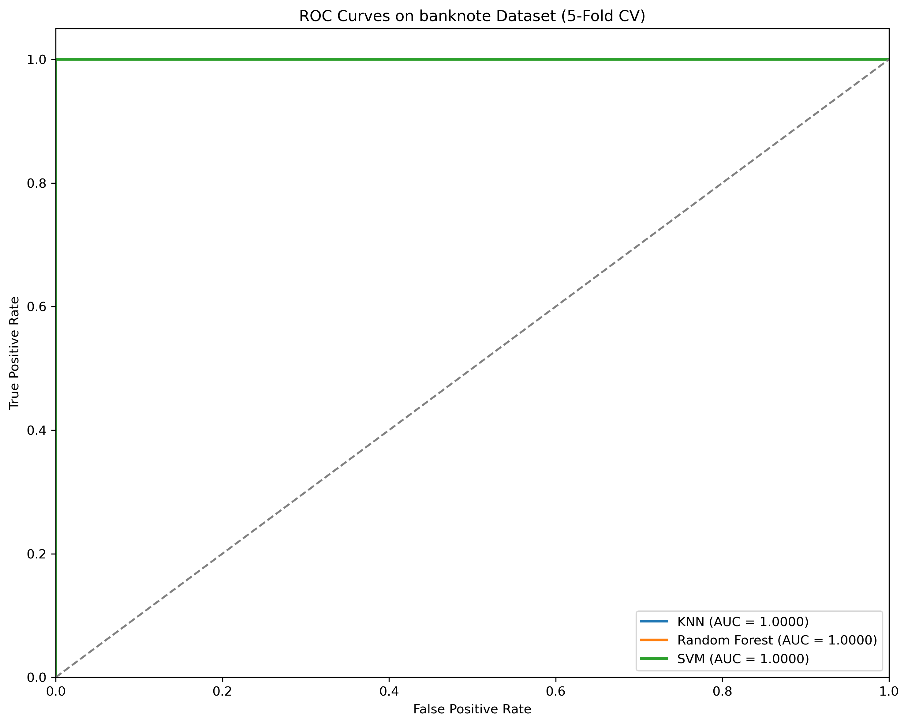
這是一個相對簡單的數據集，在SVM下曾在最佳超參數的配置下達到100%的準確率

[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/ACC_BAR_banknote.png)[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/AUC_BAR_banknote.png)

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：



[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/ROC_banknote.png)

**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

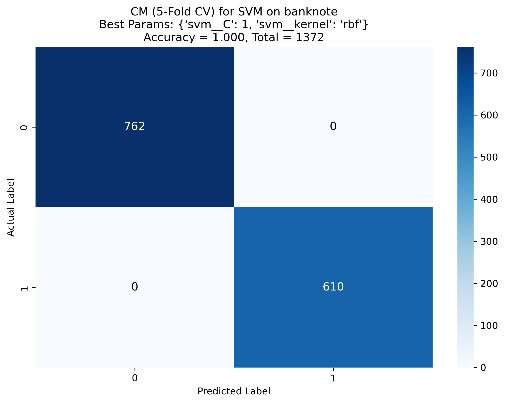
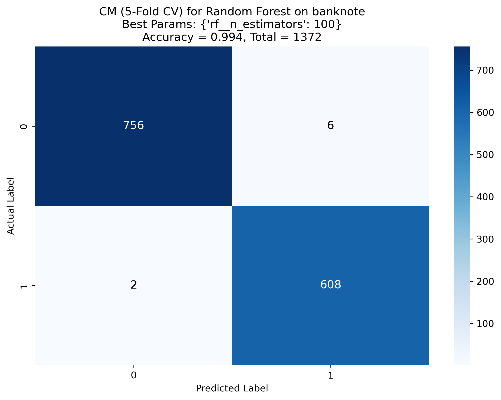
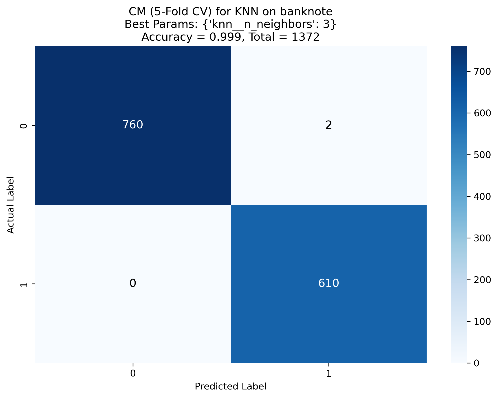
這是我首先懷疑的，不過經過檢查test data並沒有混入train data

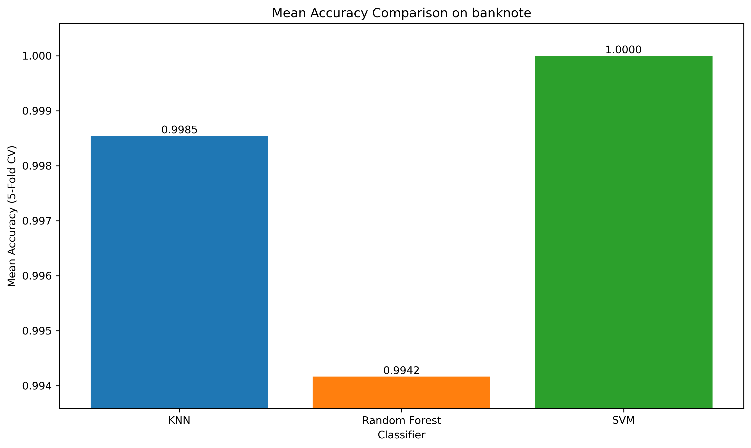
在程式中我使用的sklearn.model\_selection的StratifiedKFold進行資料分割，所有的dataset是用的都是同一套切割邏輯，但只有此數據集出現了accuracy為1的狀況

1. 更改初始化的random seed

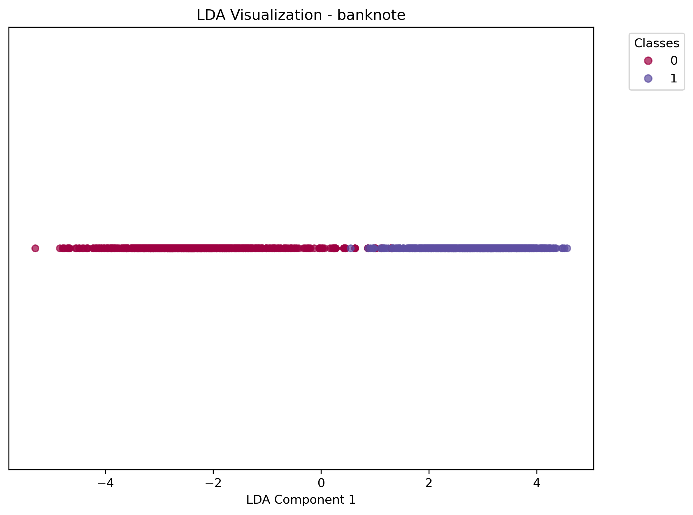
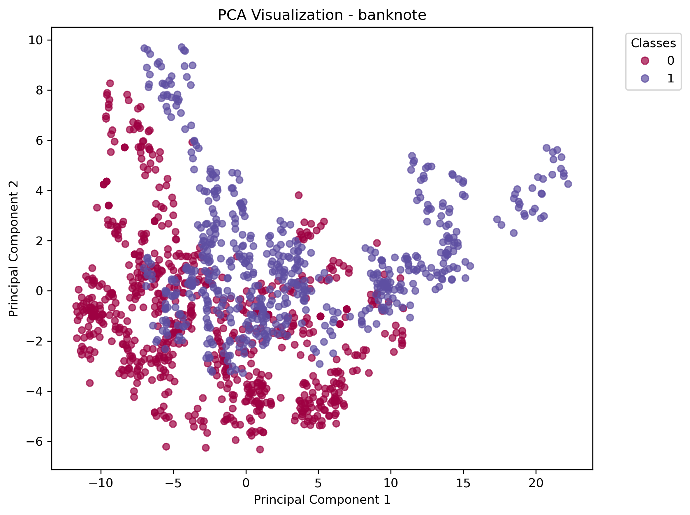
另一個可能就是運氣很好真的撞到了（不過考慮到驗證的時候也是以5-Fold Cross-Validation進行，其實不太可能）

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





1. 資料可視化



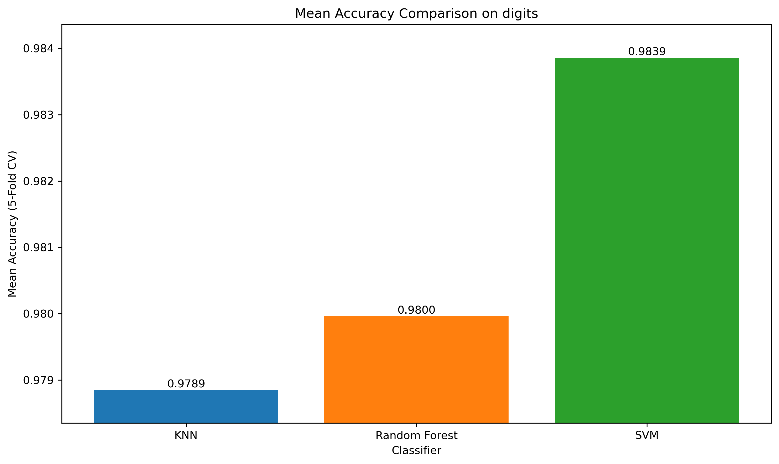
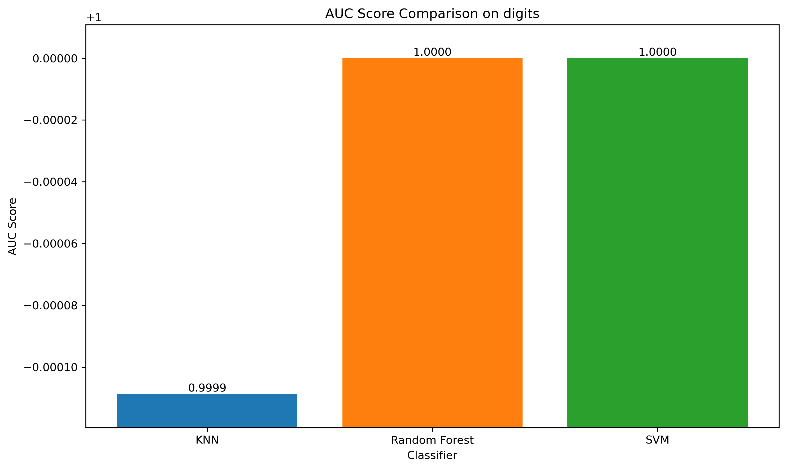
此處分別為資料集進行了PCA以及LDA的資料可視化。從LDA可以清晰地看出，紅色與藍色幾乎完全分開、中間重疊極少（幾乎沒有交錯區域）。也就是說，在這條線上模型可以找到一個分界點，使得兩類的機率分佈幾乎不重疊

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

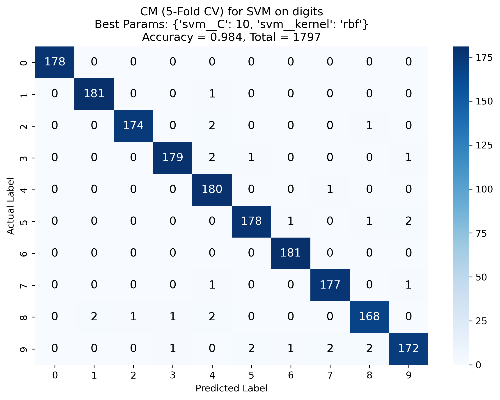
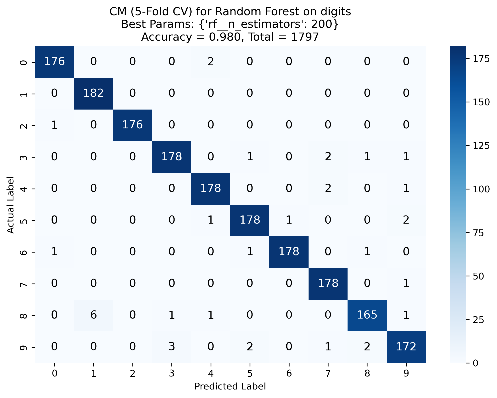
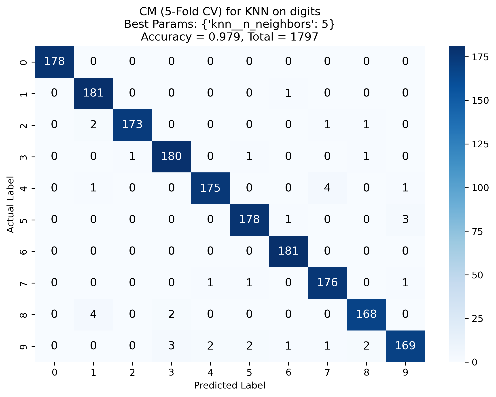
個人認為，這是SVM在此資料集上能多次達到accuracy=1的原因

## 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

[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/ACC_BAR_digits.png) [](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/AUC_BAR_digits.png)

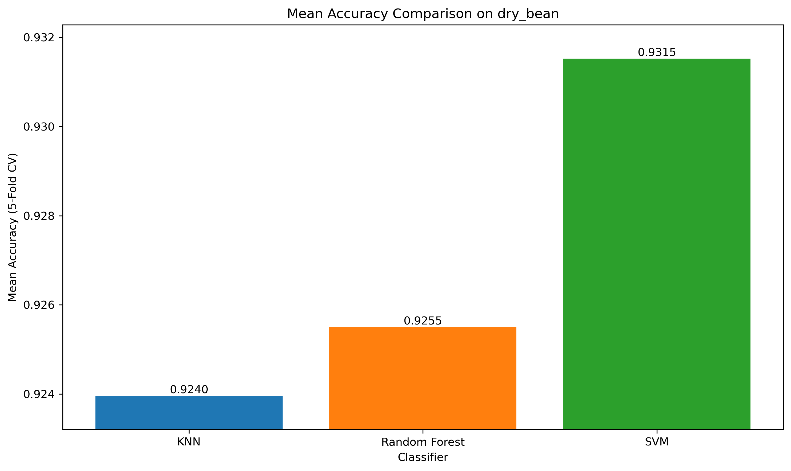
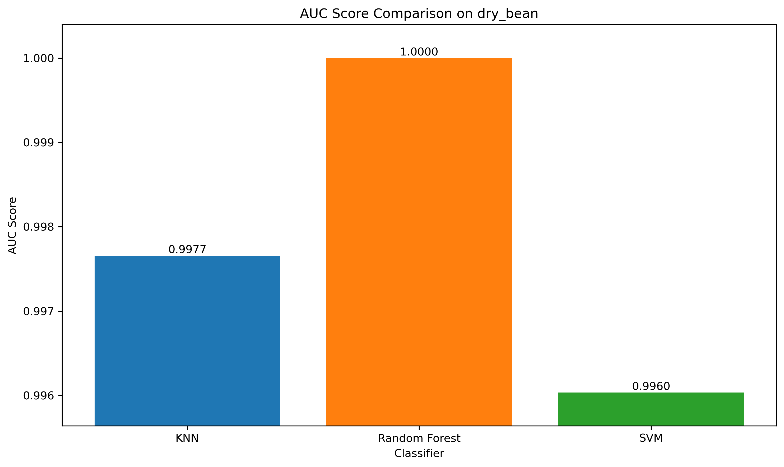
以下為使用各個分類器，在各自的**最佳超參數**下得到的混淆矩陣：



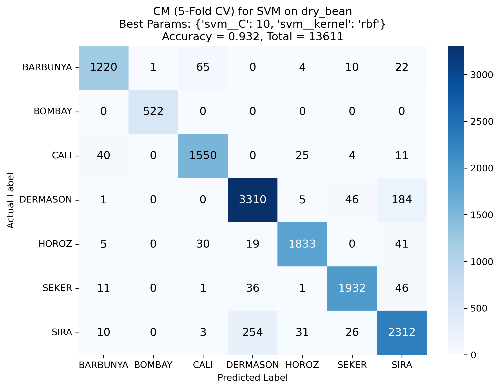
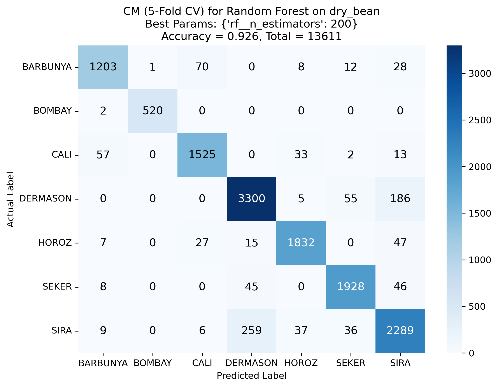
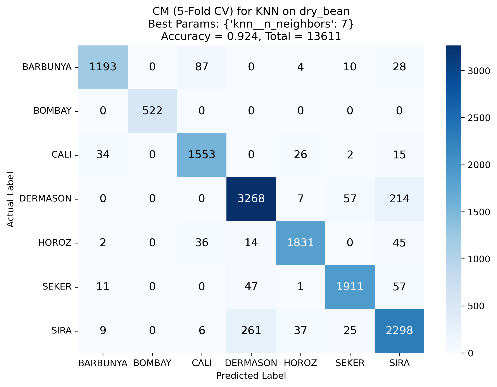
**分析：**所有三個分類器都表現出了強大的多類別分類能力，準確率均超過 97%。SVM透過其核函數技巧，在處理64維像素特徵時展現了其優越性。值得注意的是，200棵樹的Random Forest表現也極具競爭力，顯示了集成方法的效率

## 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

[](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/ACC_BAR_dry_bean.png) [](https://github.com/megrez33281/classifiers-experiment/blob/main/plots/AUC_BAR_dry_bean.png)

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：

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

AI 產生的內容可能不正確。

**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

* + - 部分ACU達到1的分數

在實驗結果中可以觀察到，部分分類器（例如 KNN 與 Random Forest）在個別資料集上曾於某些訓練/驗證分割中達到100%的分類準確率，但其平均準確率（mean accuracy）仍略低於1

這種現象屬於正常且可解釋的統計波動，並非程式錯誤或資料分割異常所造成

首先，本實驗採用Stratified 5-Fold Cross Validation，確保每個fold的訓練與測試資料互斥且類別比例一致，並利用Scikit-learn的cross\_val\_predict在嚴格的資料隔離下生成預測結果，因此不存在資料洩漏的情況

導致部分模型於單次 fold 達到完美分類的主要原因有三點：

1. **資料特性**：如Banknote Authentication資料集具有高度線性可分性，在部分隨機切割下，訓練與測試樣本之間的邊界極為明確，導致分類器能完全正確預測
2. **模型隨機性**：像Random Forest含隨機抽樣與樹分裂過程，不同隨機狀態下可能出現局部完美分類。KNN亦可能在某些資料分佈下，所有測試樣本剛好鄰近同類樣本
3. **AUC指標特性**：AUC反映的是模型對正負樣本排序的正確性，而非實際分類結果。即使模型在某些閾值下產生少量錯誤，只要分數排序完全正確，也可能得到AUC=1.0

# 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

1. **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](https://archive.ics.uci.edu/dataset/267/banknote+authentication)
   * 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](https://archive.ics.uci.edu/dataset/602/dry+bean+dataset)
2. **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}")

## Requirements

contourpy==1.3.3

cycler==0.12.1

et\_xmlfile==2.0.0

fonttools==4.60.1

joblib==1.5.2

kiwisolver==1.4.9

matplotlib==3.10.6

numpy==2.3.3

openpyxl==3.1.5

packaging==25.0

pandas==2.3.3

pillow==11.3.0

pyparsing==3.2.5

python-dateutil==2.9.0.post0

pytz==2025.2

scikit-learn==1.7.2

scipy==1.16.2

seaborn==0.13.2

six==1.17.0

tabulate==0.9.0

threadpoolctl==3.6.0

tzdata==2025.2

## GitHub

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