Machine Learning

Homework 2

系級：醫工三　　學號：B12508025　　姓名：盧雅筠

1. For each fold of leave-one-out cross-validation
    a. Determination of k , the number of Eigenvalue Selected
        i. Methodology: 為了在 Leave-One-Out Cross-Validation 的 103 Folds中客觀且自動地決定最佳主成分數量 k，採用 Maximum Distance Method 來尋找 Scree Graph 上的 Elbow Point。Scree Graph 繪製了特徵值 λ_i 與其對應的主成分順序 i。曲線的 Elbow Point 代表了「邊際效益遞減」的關鍵位置——在此點之前的主成分包含了數據的主要結構與變異，而此點之後的主成分則多為雜訊。定義 Elbow Point 為曲線上距離「連接首尾特徵值之直線」垂直距離最遠的點。
        ii. Mathematical Formulation: PCA 計算出的特徵值序列為 $\lambda_1, \lambda_2, \ldots, \lambda_d$，將 Scree Graph 視為 2D 平面上的一組座標點集 P：

$$P_i = (x_i, y_i) = (i, \lambda_i), \quad \text{for } i = 1, \ldots, d$$

定義一條連接第一個點 P_1 與最後一個點 P_d 的直線向量$\overrightarrow{L}$：

$$\overrightarrow{L} = P_d - P_1 = (d-1, \lambda_d - \lambda_1)$$

對於曲線上任意一點 P_i ，計算從起點指向該點的向量 $\overrightarrow{V_i}$：

$$\overrightarrow{V_i} = P_i - P_1 = (i-1, \lambda_i - \lambda_1)$$

為了找出轉折點，計算 P_i 到直線 $\overrightarrow{L}$ 的 Perpendicular Euclidean Distance $d(P_i, \overrightarrow{L})$。這可以通過向量的正射影與排斥Rejection原理求得：

$$d(P_i, \overrightarrow{L}) = \| (\overrightarrow{V_i} - \frac{\overrightarrow{V_i} \cdot \overrightarrow{L}}{\|\overrightarrow{L}\|^2} \overrightarrow{L}) \| |$$

最後，最佳的維度 k 被定義為具有最大垂直距離的索引值：

$$k = \arg\max_i (dist_i)$$

        iii. Code Explanation: 程式碼如 Figure 7，用 `find_elbow_point` 函式來執行上述運算，該函式被應用於 LOOCV 的每一次迭代中 。
        iv. Report of Selected k for each of the 103 folds: 結果如 Figure 1。

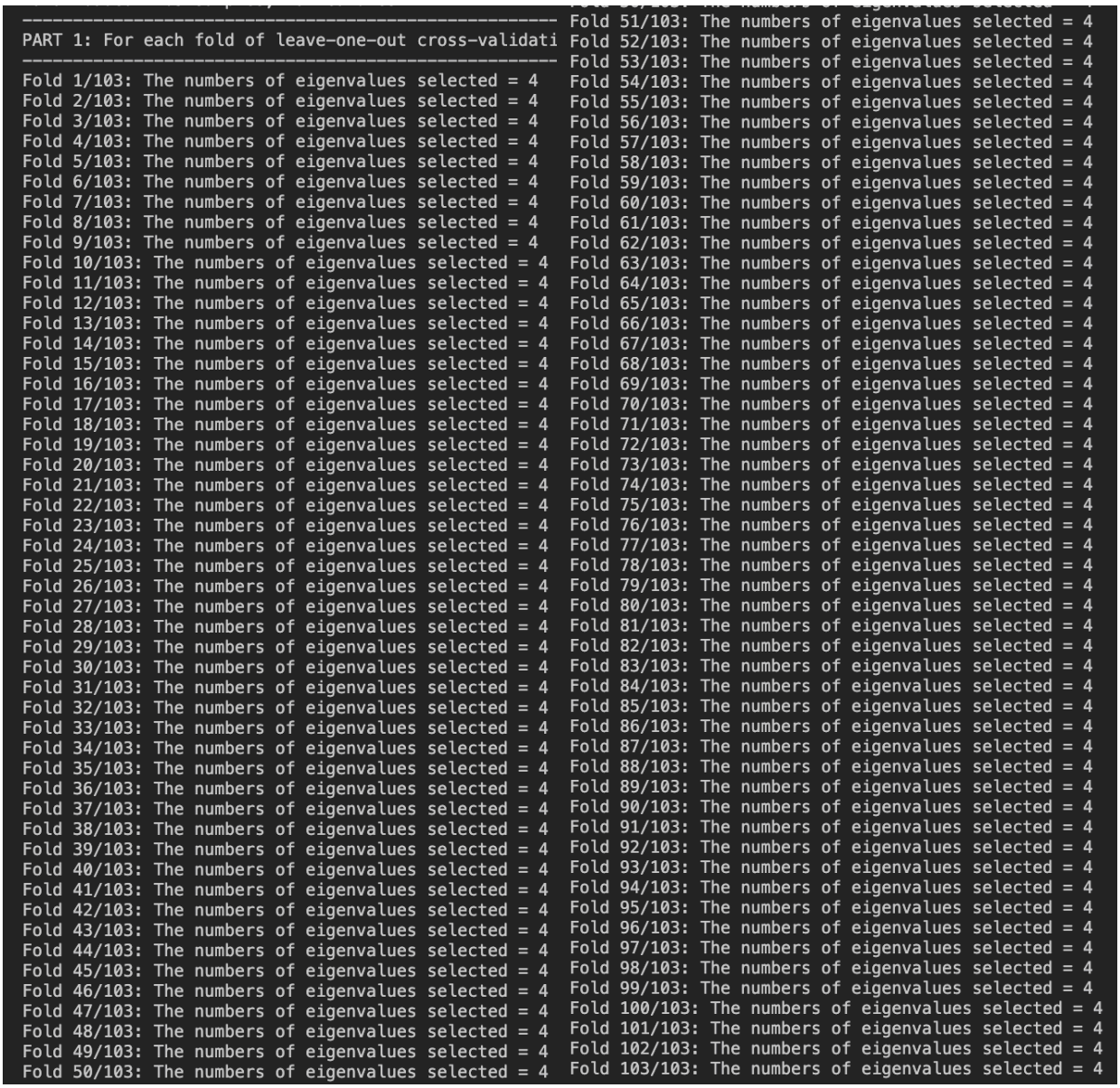

Figure 1

b. Use the 103 probability values to report the test performance: accuracy, sensitivity, specificity 如 Figure 2。ROC curve 與 AUC the area under curve of your ROC curve 如 Figure 3。

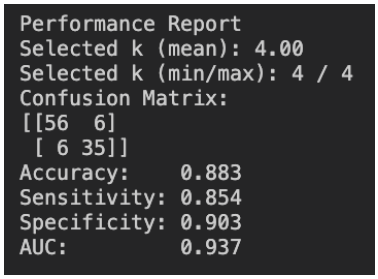

Figure 2

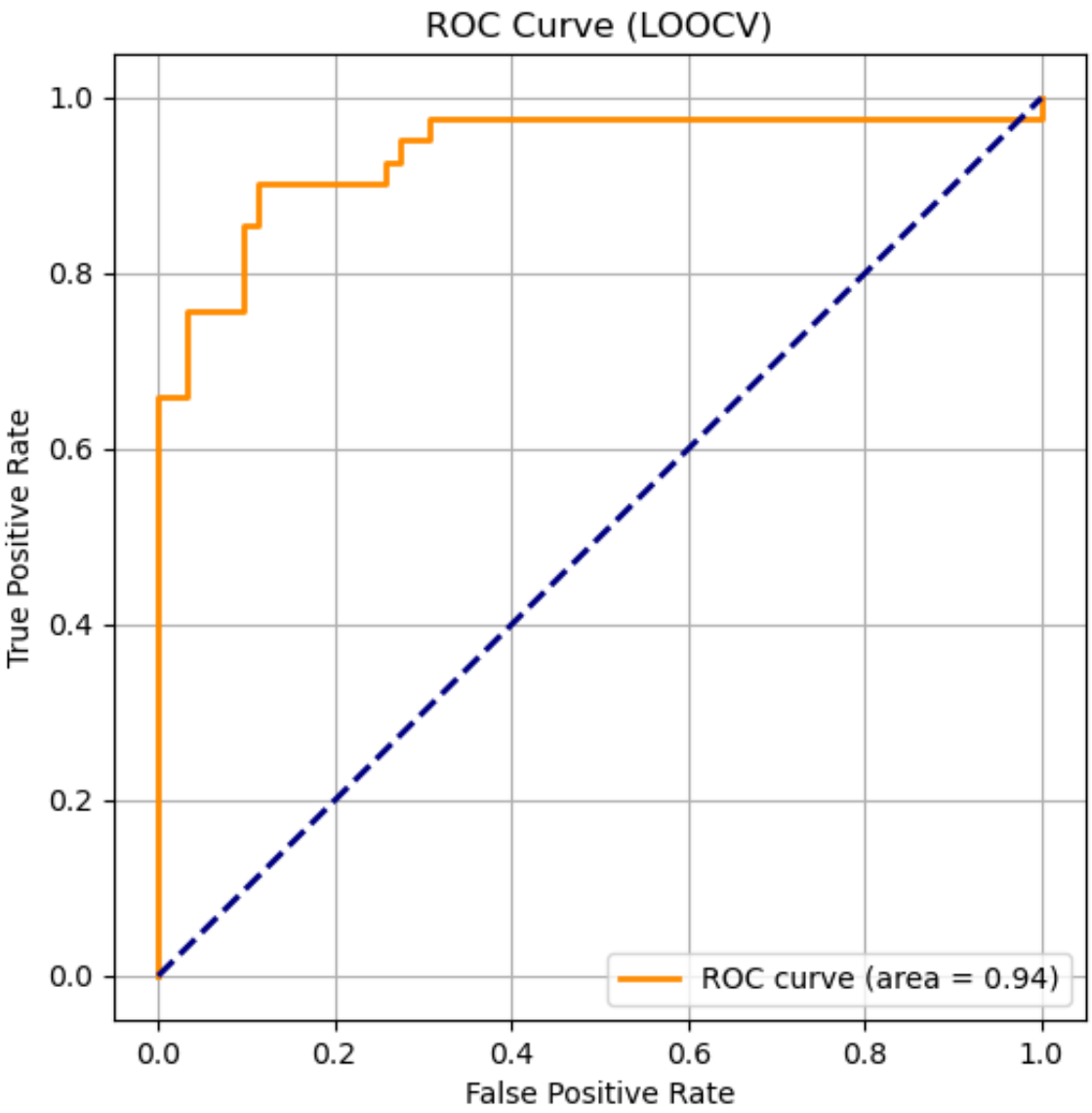
ROC Curve (LOOCV)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.94)

Figure 3

2. Use ALL 103 data

   a. Eigenvalue Analysis and Variance Explained: Scree Graph 如 Figure 4 展示了特徵值隨主成分順序的遞減趨勢。可以看到在前幾個主成分後，特徵值迅速下降，顯示數據的主要變異集中在少數幾個維度上。Proportion of Variance Explained 如 Figure 4 展示了前 m 個特徵向量所解釋的累積變異量比例。

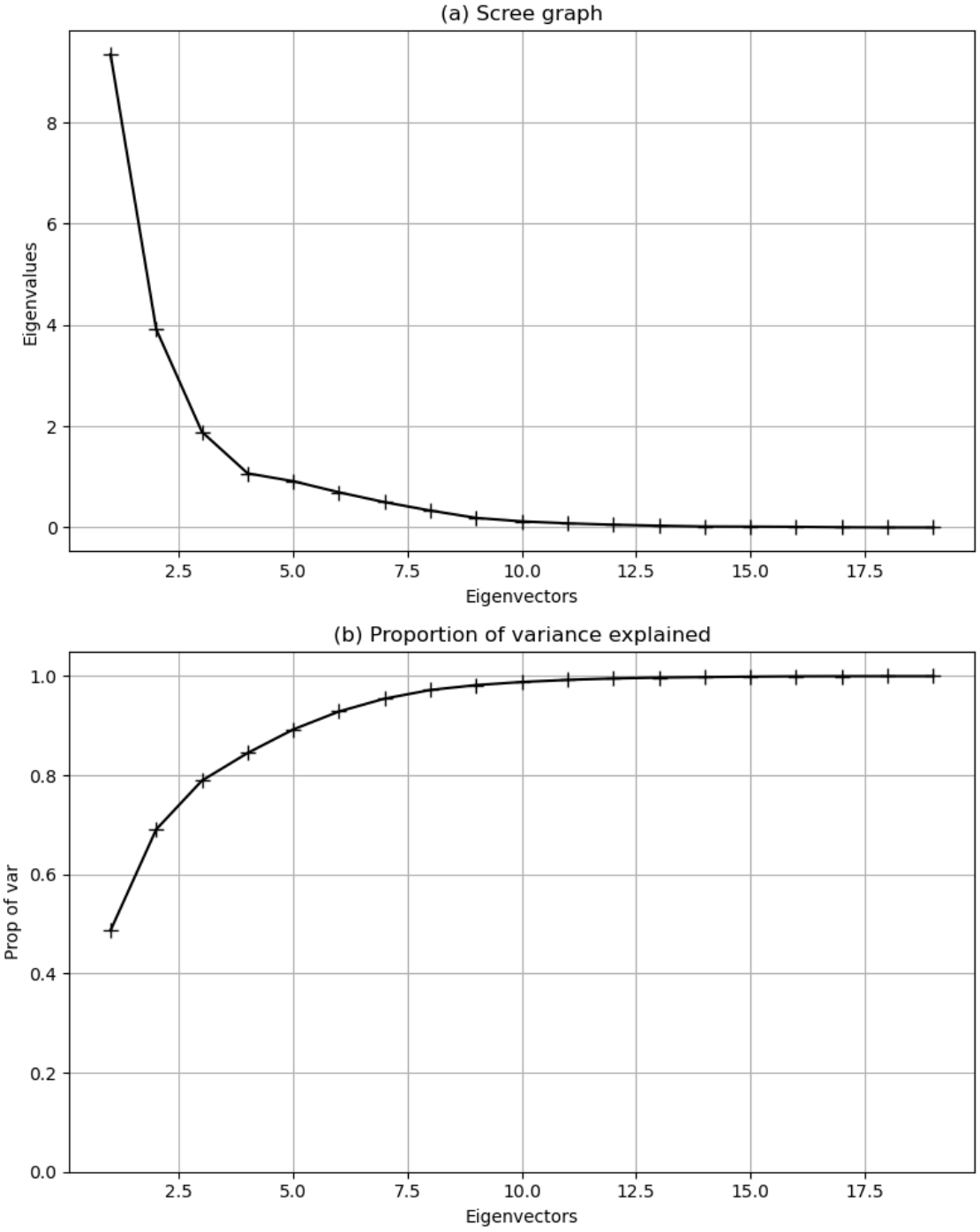

Figure 4

b. The largest five eigenvalues of PCA results: 首先對數據進行特徵分解 ，並將特徵值按降序排列。Top 5 Eigenvalues 根據 PCA 分析結果，最大的五個特徵值如 Figure 5 所示。

```
--------------------------
PART 2: Use ALL 103 data
--------------------------
Performance Report
Top 5 Eigenvalues:
  1: 9.3448
  2: 3.9126
  3: 1.8877
  4: 1.0692
  5: 0.9131
```

Figure 5

c. Bivariate Gaussian Bayesian Decision Model: 為了將原本的高維數據可視化並進行分類，將所有數據投影到由前兩個特徵向量 (對應最大兩個特徵值) 所張成的 2D 空間，分別標記為 PC1 與 PC2 。在此 2D 子空間中，構建了一個 Bivariate Gaussian Bayesian Decision Model。該模型的視覺化結果如 Figure 6 所示。橫軸與縱軸分別為 PC1 與 PC2 。Class 1 以紅色加號 (+) 表示；Class 0 以藍色圓圈 (o) 表示 。可以看到兩類數據在 PC1 與 PC2 空間中有一定的分離度。而 Contour Map 中，實線代表 Class 1 的分佈；虛線代表 Class 0 的分佈。最後，Decision Boundary 為黑色粗線，代表$\Delta = 0$的軌跡。

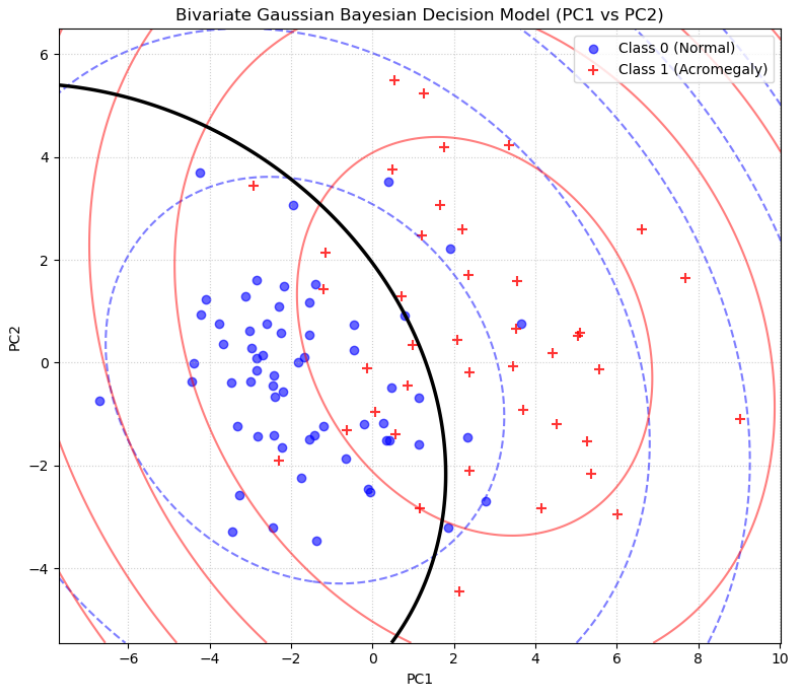

Figure 6

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

''' Multivariate Gaussian Bayesian classifier '''
def Multivariate_Gaussian_Distribution_log_likelihood(X, mu, cov):
    X = np.atleast_2d(X)
    d = X.shape[1]

    # Cholesky decomposition, check for positive definiteness
    try:
        L = np.linalg.cholesky(cov)
    except np.linalg.LinAlgError:
        # If not positive definite, add small ridge to diagonal
        cov = cov + np.eye(d) * 1e-4
        L = np.linalg.cholesky(cov)

    Z = np.linalg.solve(L, (X - mu).T)
    quad = np.sum(Z**2, axis=0)
    logdet = 2.0 * np.sum(np.log(np.diag(L)))
    log_likelihood = -0.5 * (d * np.log(2 * np.pi) + logdet + quad)
    return log_likelihood

def Bayesian_decision_classifier(X, mu0, mu1, cov0, cov1, p0, p1):
    ll0 = Multivariate_Gaussian_Distribution_log_likelihood(X, mu0, cov0) + np.log(p0)
    ll1 = Multivariate_Gaussian_Distribution_log_likelihood(X, mu1, cov1) + np.log(p1)
    delta = ll1 - ll0
    m = np.maximum(ll0, ll1)
    num1 = np.exp(ll1 - m)
    den  = np.exp(ll0 - m) + num1
    posterior_1 = num1 / den
    return delta, posterior_1

''' Screen Graph of Elbow Method to find optimal k '''
def find_elbow_point(eigenvalues):
    n_points = len(eigenvalues)
    # if less than 2 points, return 1 as k
    if n_points < 2:
        return 1

    all_coords = np.vstack((range(n_points), eigenvalues)).T # each row is [x, y] = [index, eigenvalue]
    first_point = all_coords[0]
    last_point = all_coords[-1]

    vec_line = last_point - first_point # vector from first to last point
    vec_line_norm = vec_line / np.sqrt(np.sum(vec_line**2))

    vec_from_first = all_coords - first_point # vectors from first point to all points
    scalar_product = np.sum(vec_from_first * vec_line_norm, axis=1) # projection length on line
    vec_from_first_parallel = np.outer(scalar_product, vec_line_norm) # projection vectors on line(parallel)
    vec_to_line = vec_from_first - vec_from_first_parallel # vectors from points to line(perpendicular)
    dist_to_line = np.sqrt(np.sum(vec_to_line ** 2, axis=1)) # distances to line

    # Index of the point with maximum distance to line
    best_k = np.argmax(dist_to_line) + 1

    return max(1, best_k)
```

```python
''' Custom PCA Implementation '''
class CustomPCA:
    def __init__(self, n_components=None):
        self.n_components = n_components
        self.components_ = None
        self.mean_ = None
        self.explained_variance_ = None
        self.explained_variance_ratio_ = None

    def fit(self, X):
        # 1. Centering
        self.mean_ = np.mean(X, axis=0)
        X_centered = X - self.mean_

        # 2. Covariance Matrix ~ X^T X
        cov_matrix = np.cov(X_centered, rowvar=False) # rowvar=False 表示每一行是一個特徵

        # 3. Eigendecomposition: Cv = λv
        eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

        # 4. Sort Eigenvalues and Eigenvectors, descending order
        idx = np.argsort(eigenvalues)[::-1]
        self.explained_variance_ = eigenvalues[idx]
        sorted_vectors = eigenvectors[:, idx]

        # 5. calculate explained variance ratio
        total_var = np.sum(self.explained_variance_)
        self.explained_variance_ratio_ = self.explained_variance_ / total_var

        # 6. Select top n_components
        if self.n_components is None:
            n_comp = X.shape[1] # all components
        else:
            n_comp = min(self.n_components, X.shape[1])
        self.components_ = sorted_vectors[:, :n_comp].T # W: (n_components, n_features)
        return self

    def transform(self, X):
        X_centered = X - self.mean_
        # Project data onto principal components: X_new = X_centered @ W.T
        return np.dot(X_centered, self.components_.T)

    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)


''' Load Data '''
data_path = 'AcromegalyFeatureSet.xlsx'
data = pd.read_excel(data_path)
data.rename(columns=lambda s: s.strip() if isinstance(s, str) else s, inplace=True)

X_raw = data.drop(columns=['SeqNum', 'Gender', 'GroundTruth']).values
y = data['GroundTruth'].values
n_samples, n_features = X_raw.shape

print(f"Data Loaded: {n_samples} samples, {n_features} features.")

''' Part1: Leave-One-Out Cross Validation with PCA and Bayesian Classifier '''
print("-" * 60)
print("PART 1: For each fold of leave-one-out cross-validation")
print("-" * 60)
posts1 = []
deltas = []
k_values = []
preds = []
```

```python
for i in range(n_samples):
    # 1. Split Data
    X_train_raw = np.delete(X_raw, i, axis=0)
    y_train = np.delete(y, i)
    X_test_raw = X_raw[i].reshape(1, -1) # (d,) -> (1, d)

    # 2. Standardization
    scaler = StandardScaler() # standard to zero mean and unit variance
    X_train_std = scaler.fit_transform(X_train_raw)
    X_test_std = scaler.transform(X_test_raw)

    # 3. PCA on Training Data
    pca_full = CustomPCA(n_components=None) # 計算所有成分
    pca_full.fit(X_train_std)

    # 4. Find k using Screen Graph of Elbow Method
    eigenvalues = pca_full.explained_variance_
    k = find_elbow_point(eigenvalues)
    k_values.append(k)
    print(f"Fold {i+1}/{n_samples}: The numbers of eigenvalues selected = {k}")

    # 5. Project Data
    pca_k = CustomPCA(n_components=k)
    X_train_pca = pca_k.fit_transform(X_train_std)
    X_test_pca = pca_k.transform(X_test_std)

    # 6. Train Bayesian Classifier
    X0 = X_train_pca[y_train == 0]
    X1 = X_train_pca[y_train == 1]

    mu0 = np.mean(X0, axis=0)
    mu1 = np.mean(X1, axis=0)

    cov0 = np.cov(X0, rowvar=False) + np.eye(k) * 1e-4
    cov1 = np.cov(X1, rowvar=False) + np.eye(k) * 1e-4

    p0 = len(X0) / len(X_train_pca)
    p1 = len(X1) / len(X_train_pca)

    # 7. Test
    delta, posterior1 = Bayesian_decision_classifier(X_test_pca, mu0, mu1, cov0, cov1, p0, p1)

    # --- FIX START: 使用 .item() 解決 DeprecationWarning ---
    posterior1_val = posterior1.item()
    delta_val = delta.item()

    posts1.append(posterior1_val)
    deltas.append(delta_val)
    preds.append(1 if delta_val >= 0 else 0)
    # --- FIX END ---

# Performance
posts1 = np.array(posts1)
deltas = np.array(deltas)
preds = np.array(preds)
k_values = np.array(k_values)

fpr, tpr, _ = roc_curve(y, posts1)
roc_auc = auc(fpr, tpr)
cm = confusion_matrix(y, preds)
TN, FP, FN, TP = cm.ravel()

accuracy = (TP + TN) / (TP + TN + FP + FN)
sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0
specificity = TN / (TN + FP) if (TN + FP) > 0 else 0
```

```python
print("-" * 30)
print("Performance Report")
print(f"Selected k (mean): {np.mean(k_values):.2f}")
print(f"Selected k (min/max): {np.min(k_values)} / {np.max(k_values)}")
print(f"Confusion Matrix:\n{cm}")
print(f"Accuracy:    {accuracy:.3f}")
print(f"Sensitivity: {sensitivity:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"AUC:         {roc_auc:.3f}")

# ROC curve plot
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (LOOCV)')
plt.legend(loc="lower right")
plt.grid(True)
plt.savefig('homework2_roc.png')

''' Part2: Full Data Analysis with PCA and Bayesian Classifier '''
print("\n" + "-" * 60)
print("PART 2: Use ALL 103 data")
print("-" * 60)

scaler_all = StandardScaler()
X_std_all = scaler_all.fit_transform(X_raw)

# 使用 CustomPCA 計算所有資料
pca_all = CustomPCA(n_components=None)
pca_all.fit(X_std_all)

eigenvalues_all = pca_all.explained_variance_
ratios = pca_all.explained_variance_ratio_
cumulative_ratio = np.cumsum(ratios)
print("Performance Report")
print("Top 5 Eigenvalues:")
for i, val in enumerate(eigenvalues_all[:5]):
    print(f"  {i+1}: {val:.4f}")

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 10))

ax1.plot(range(1, len(eigenvalues_all)+1), eigenvalues_all, 'k-+', markersize=8)
ax1.set_title('(a) Scree graph')
ax1.set_ylabel('Eigenvalues')
ax1.set_xlabel('Eigenvectors')
ax1.grid(True)

ax2.plot(range(1, len(cumulative_ratio)+1), cumulative_ratio, 'k-+', markersize=8)
ax2.set_title('(b) Proportion of variance explained')
ax2.set_ylabel('Prop of var')
ax2.set_xlabel('Eigenvectors')
ax2.set_ylim([0, 1.05])
ax2.grid(True)

plt.tight_layout()
plt.savefig('homework2_pca_scree.png')
# plt.show()
```

```python
# Plotting 2D Bivariate Gaussian Decision Boundary
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X_std_all)

X0_2d = X_pca_2d[y == 0]
X1_2d = X_pca_2d[y == 1]

mu0_2d = np.mean(X0_2d, axis=0)
mu1_2d = np.mean(X1_2d, axis=0)
cov0_2d = np.cov(X0_2d, rowvar=False) + np.eye(2) * 1e-4
cov1_2d = np.cov(X1_2d, rowvar=False) + np.eye(2) * 1e-4
p0_2d = len(X0_2d) / len(X_pca_2d)
p1_2d = len(X1_2d) / len(X_pca_2d)

x_min, x_max = X_pca_2d[:, 0].min() - 1, X_pca_2d[:, 0].max() + 1
y_min, y_max = X_pca_2d[:, 1].min() - 1, X_pca_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                     np.linspace(y_min, y_max, 200))
pts = np.c_[xx.ravel(), yy.ravel()]

LL0 = Multivariate_Gaussian_Distribution_log_likelihood(pts, mu0_2d, cov0_2d).reshape(xx.shape)
LL1 = Multivariate_Gaussian_Distribution_log_likelihood(pts, mu1_2d, cov1_2d).reshape(xx.shape)

boundary = (LL1 + np.log(p1_2d)) - (LL0 + np.log(p0_2d))

plt.figure(figsize=(8, 7))
plt.scatter(X0_2d[:, 0], X0_2d[:, 1], c='blue', marker='o', label='Class 0 (Normal)', alpha=0.6)
plt.scatter(X1_2d[:, 0], X1_2d[:, 1], c='red', marker='+', s=60, label='Class 1 (Acromegaly)', alpha=0.8)

plt.contour(xx, yy, LL0, levels=5, colors='blue', linestyles='dashed', alpha=0.5)
plt.contour(xx, yy, LL1, levels=5, colors='red', linestyles='solid', alpha=0.5)
plt.contour(xx, yy, boundary, levels=[0], colors='black', linewidths=2.5, linestyles='-')

plt.title('Bivariate Gaussian Bayesian Decision Model (PC1 vs PC2)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(loc='best')
plt.grid(True, linestyle=':', alpha=0.6)
plt.tight_layout()
plt.savefig('homework2_2d_boundary.png')
```

Figure 1