0416025 呂翊愷

圖形識別 Assignment #2

Methods you have implemented

I. FLD

先將原始dataset的順序隨機排序，然後將前3/4筆data當作training data，而剩下1/4的data當作test data。接著將training data的scatter matrix Sw和Sb依照公式算出來，再利用Sw和training data的mean將w算出來。在test data的部分，一筆一筆資料處理，將每筆test data用w做投影，做法是算出wTx，得到投影後的向量。最後，再將投影後的向量用上次作業的Bayesian classifier, Naive-Bayes classifier, linear classifier來預測每筆test data的類別。此外，在投影前後都有依據公式算出separability measures，以便分析投影的效果。

$$S_w = \sum_{i=1}^{M} P_i \Sigma_i \quad \text{with} \quad \Sigma_i = \underset{x \in \omega_i}{E}\left[(\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^T\right], \quad \boldsymbol{\mu}_i = \underset{x \in \omega_i}{E}[\boldsymbol{x}]$$

$$S_b = \sum_{i=1}^{M} P_i (\boldsymbol{\mu}_i - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_i - \boldsymbol{\mu}_0)^T \quad \text{with} \quad \boldsymbol{\mu}_0 = \sum_{i-1}^{M} P_i \boldsymbol{\mu}_i = E[\boldsymbol{x}]$$

$$\boldsymbol{w} = S_w^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \qquad y = \boldsymbol{w}^T \boldsymbol{x} \qquad J = \operatorname{tr}\left\{S_w^{-1} S_b\right\}$$

II. PCA and classification

先將原始dataset的順序隨機排序，然後將前3/4筆data當作training data，而剩下1/4的data當作test data。接著分別將training data和test data做標準化，再用標準化後的training data算出covariance matrix，再求出covariance的eigenvalue和eigenvector，取出最大的L個eigenvalue的eigenvector當作投影矩陣A。此處的L值是降維後的維度，經過實測，隨著dataset的不同，每種dataset有各自適合的L值，而且L值也會受到分類器的影響。最後，利用A矩陣將training data和test data做投影，並將投影後的資料用上次作業的Bayesian classifier, Naive-Bayes classifier, linear classifier來預測每筆test data的類別。

$$\boldsymbol{y} = A^T \boldsymbol{x}$$

$$\Sigma_x \boldsymbol{e}_j = \lambda_j \boldsymbol{e}_j$$

III. Eigenface and classification

作業的eigenface實作部分，是使用性別辨識和人臉辨識兩種dataset。由於兩種dataset結構的不同，因此用了不同的資料取樣方法。

性別辨識的部分，讀取每張臉的影像中每個pixel的灰階值，再將每張臉的影像水平翻轉，得到翻轉後的臉的影像，如此一來得到原本影像中2倍數量的臉。之後，再將這些臉的dataset順序隨機排序，然後將前3/4筆data當作training data，而剩下1/4的data當作test data。

人臉辨識的部分，由於每種臉的原始dataset只有5個，為了避免隨機排序後取得的training data中，某些種類的臉取得樣本太少(例如1, 2個甚至沒有)，導致無法準確分類，因此我不採取隨機取樣。我將每種臉的前4張影像以及這4張臉水平翻轉後的影像分到training data，每種臉剩下的1張影像分到test data。

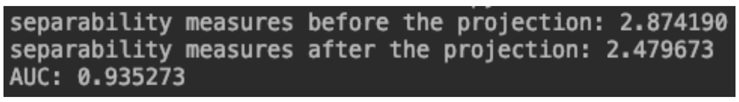
將dataset分成training data和test data後，先將training data標準化，把每筆training data當作向量放到X矩陣的每一行。接著算出XT * X，並得到XT * X的eigenvalue和eigenvector，再將X乘以eigenvector得到新的eigenvector並normalize。之後，取出最大的L個eigenvalue的新的eigenvector當作投影矩陣A。此處的L值是降維後的維度，經過實測，隨著dataset的不同，每種dataset有各自適合的L值，而且L值也會受到分類器的影響。最後，利用A矩陣將training data和test data做投影，並將投影後的資料用上次作業的Bayesian classifier, Naive-Bayes classifier, linear classifier來預測每筆test data的類別。

Experiments you have done, and the results

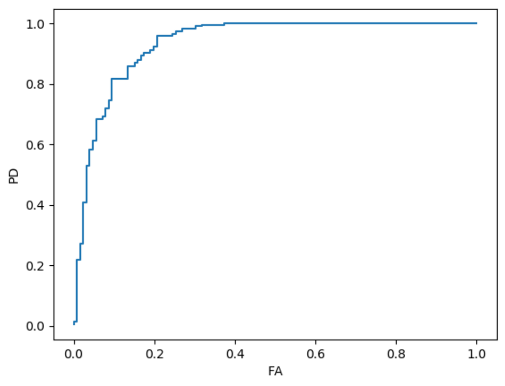
I. FLD

經過多次的測試，發現FLD在測試的每個dataset表現不錯，其ROC curve和auc值都比上次作業實作的linear classifier略高，而且透過計算出來的separability measures會發現，投影後資料確實有變得比較集中。然而，FLD在gender dataset，有時auc值會特別低，大約0.2，原因至今我仍想不透。

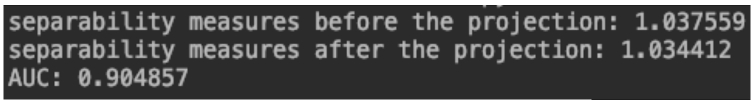
Ionosphere dataset



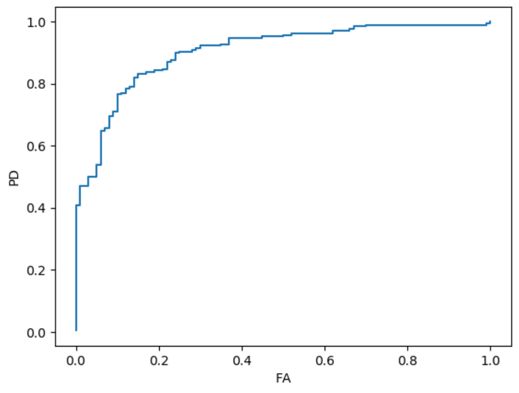
```
separability measures before the projection: 2.874190
separability measures after the projection: 2.479673
AUC: 0.935273
```



Vertebral dataset

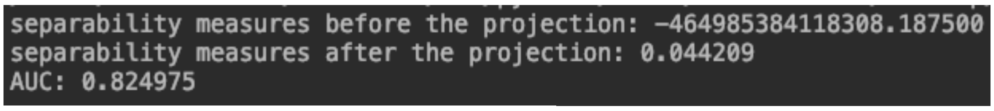


```
separability measures before the projection: 1.037559
separability measures after the projection: 1.034412
AUC: 0.904857
```

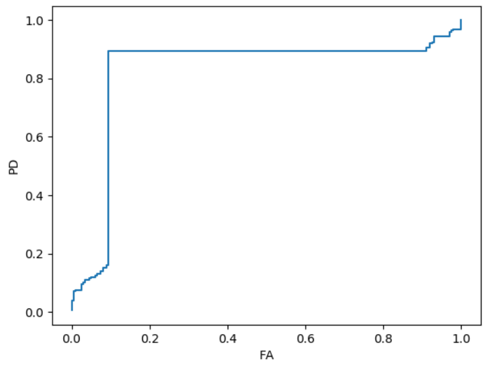


0416025 呂翊愷

Gender dataset



```
separability measures before the projection: -464985384118308.187500
separability measures after the projection: 0.044209
AUC: 0.824975
```

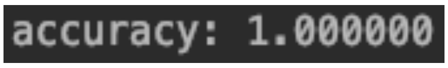


II. PCA and classification

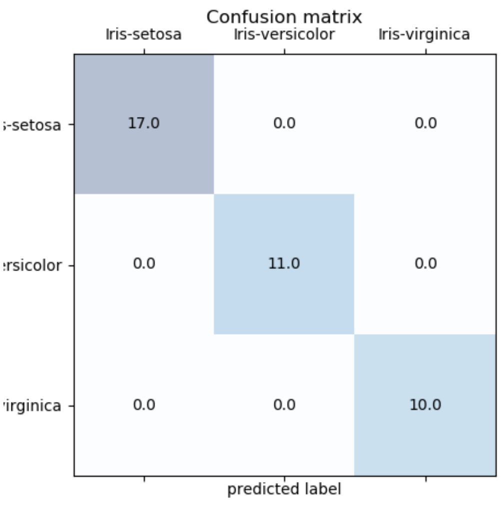
經過多次的測試，發現如果dataset在10維以下，降維到3維效果最好；如果dataset在10維以上、20維以下，降維到10維效果最好；如果dataset在20維以上，降維到20維效果最好。我覺得可能是因為如果維度太低，可能會因為資訊太少而不準確；如果維度太高，可能會因為overfitting而準確率降低。除此之外，和上次作業的實作結果相比，不論使用哪個dataset，用PCA降維後的分類表現有時候比較差，我覺得原因應該是因為降維後有些資訊損失，包含的種類資訊太少，導致誤判的可能性較高。

Iris dataset(4維→3維)

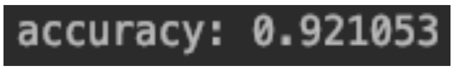
A. Bayes classifier



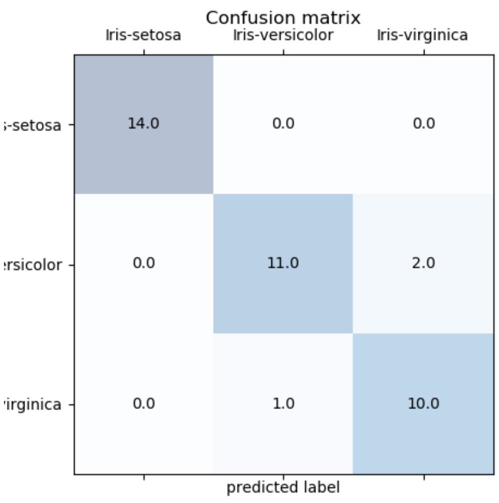
```
accuracy: 1.000000
```



B. Naïve-Bayes classifier



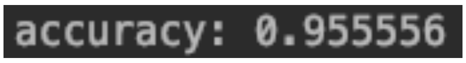
```
accuracy: 0.921053
```



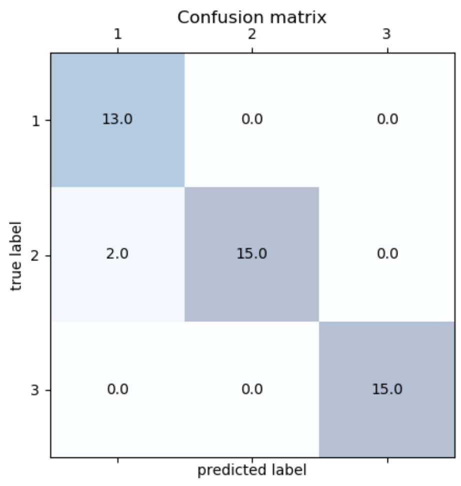
0416025 呂翊愷

Wine dataset(13維→10維)

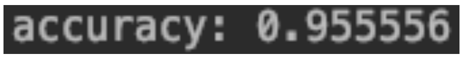
A. Bayes classifier



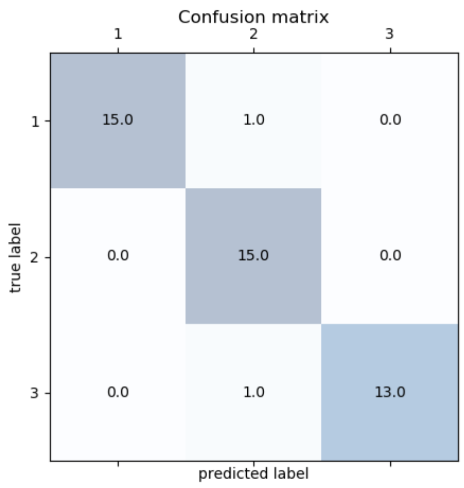
```
accuracy: 0.955556
```



B. Naïve-Bayes classifier

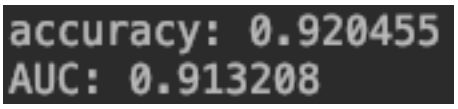


```
accuracy: 0.955556
```

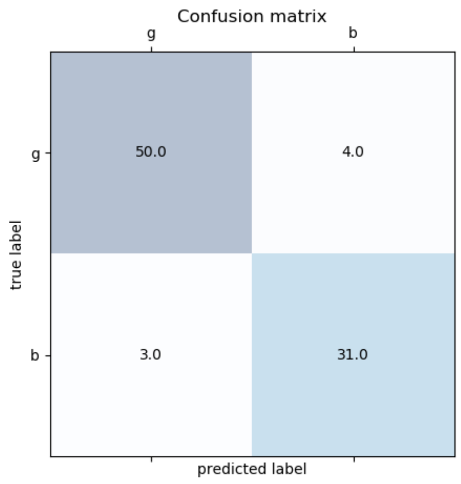


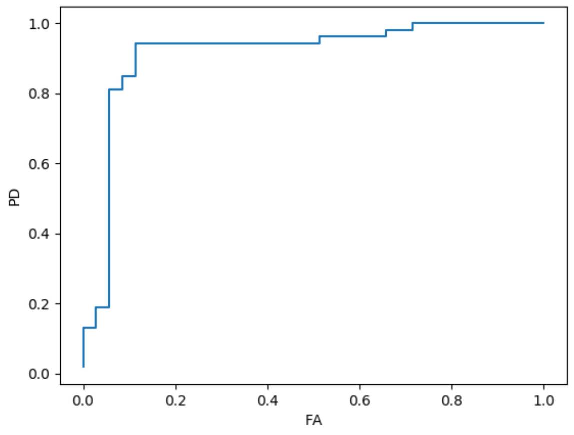
Ionosphere dataset(32維→20維)

A. Bayes classifier

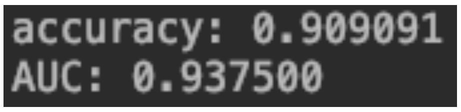


```
accuracy: 0.920455
AUC: 0.913208
```

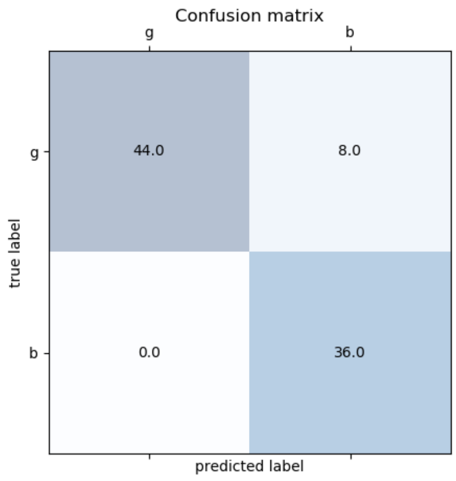


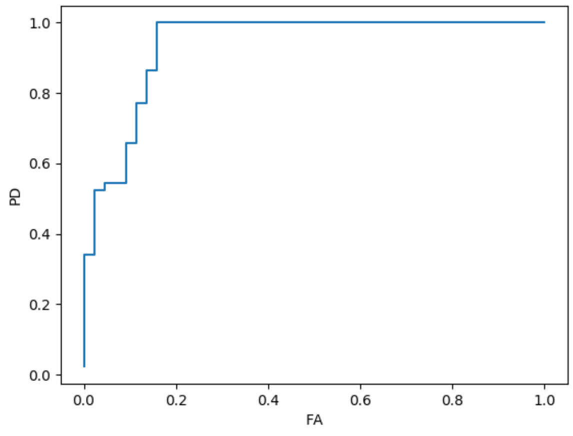


B. Naïve-Bayes classifier



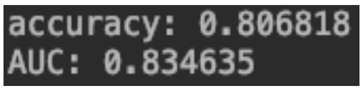
```
accuracy: 0.909091
AUC: 0.937500
```



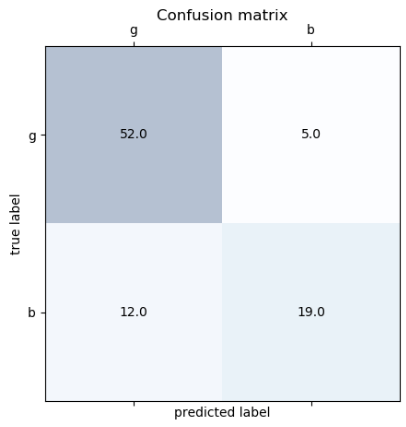


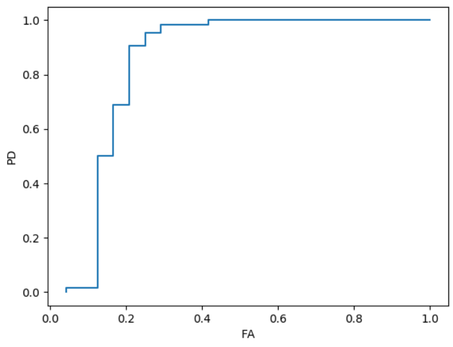
0416025 呂翊愷

C. Linear classifier



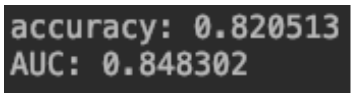
```
accuracy: 0.806818
AUC: 0.834635
```



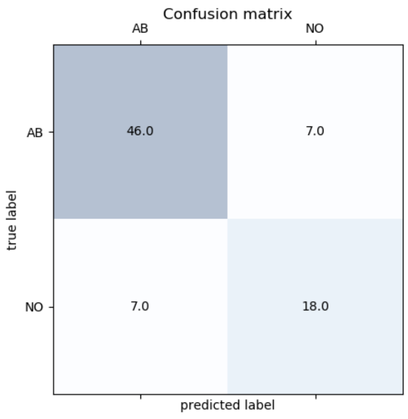


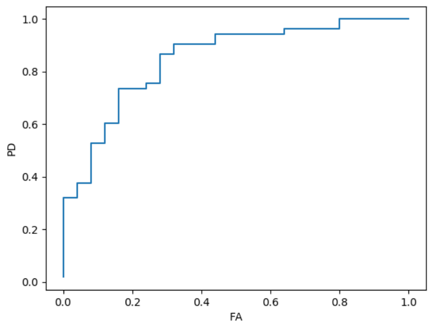
Vertebral dataset(6維→3維)

A. Bayes classifier

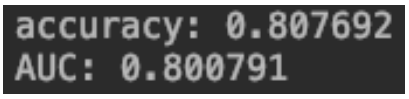


```
accuracy: 0.820513
AUC: 0.848302
```

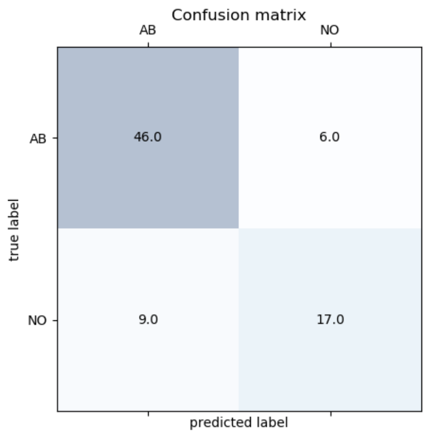


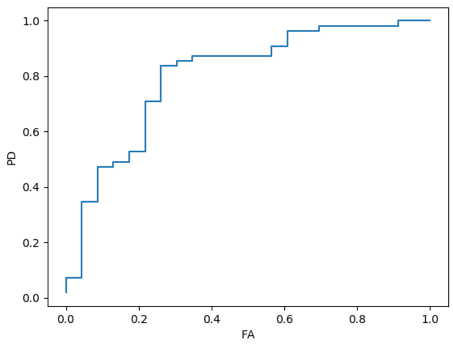


B. Naïve-Bayes classifier

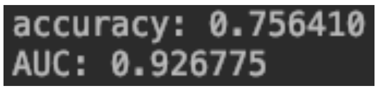


```
accuracy: 0.807692
AUC: 0.800791
```

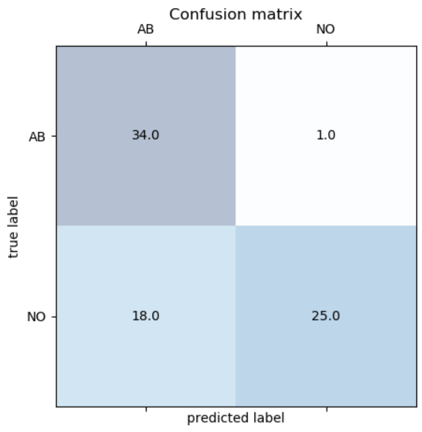


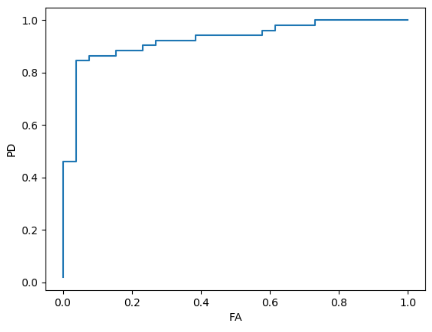


C. Linear classifier



```
accuracy: 0.756410
AUC: 0.926775
```





III. Eigenface and classification

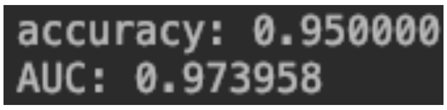
性別辨識的部分，我採用了Bayes classifier, Naïve-Bayes classifier, Linear classifer做最後的分類。人臉辨識的部分，經過多次測試，由於Bayes classifier不知道為什麼，在算probability density function的時候會一直算出singular matrix導致程式無法繼續，所以最後我只用Naïve-Bayes classifier分類。

為了瞭解降維對於資料分類表現的影響，在性別辨識的時候，我將資料維度從2到200的準確率和auc值做成折線圖，結果發現準確率和auc值在維度2~30的時候逐漸攀升，但到了30以後卻逐漸下降。而在人臉辨識的時候，到了維度100以後準確率和auc值也逐漸下降。我覺得可能是維度太高的話會overfitting，導致分類表現反而會變得更差。

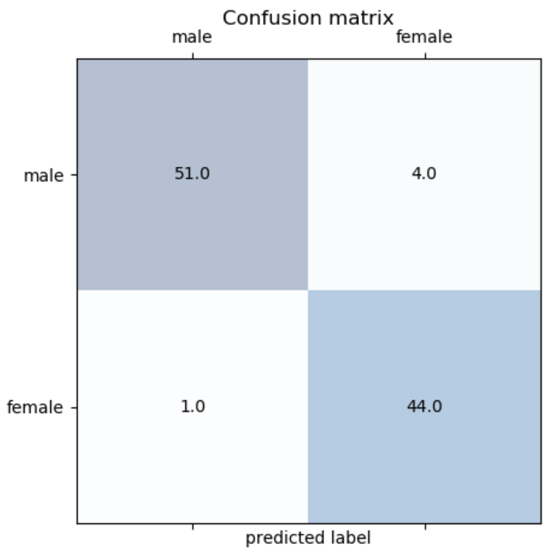
Gender classification

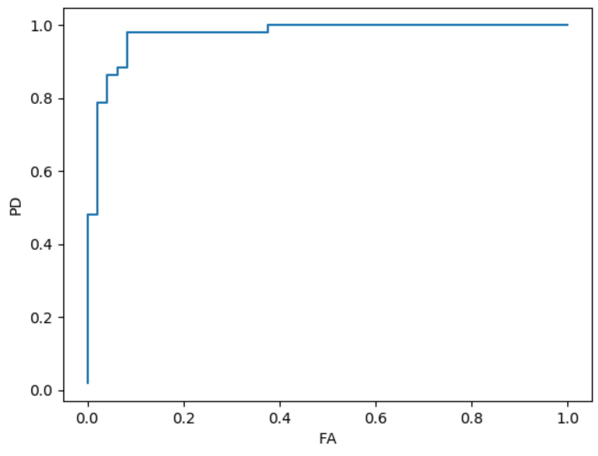
A. Bayes classifier

最佳分類表現為1600維→50維：

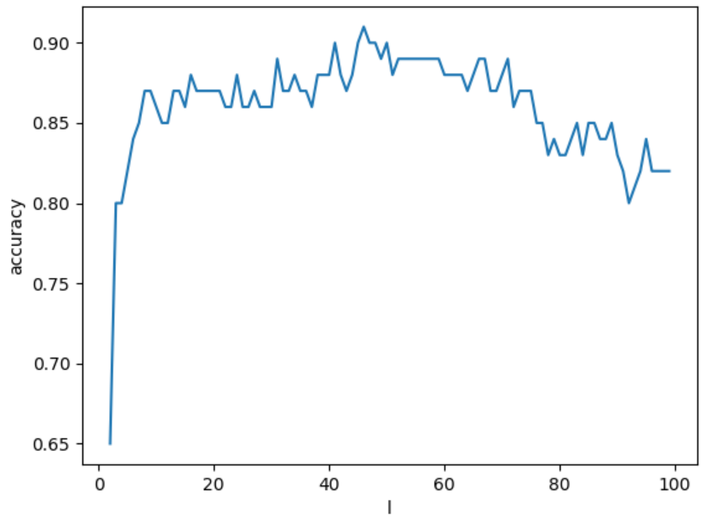


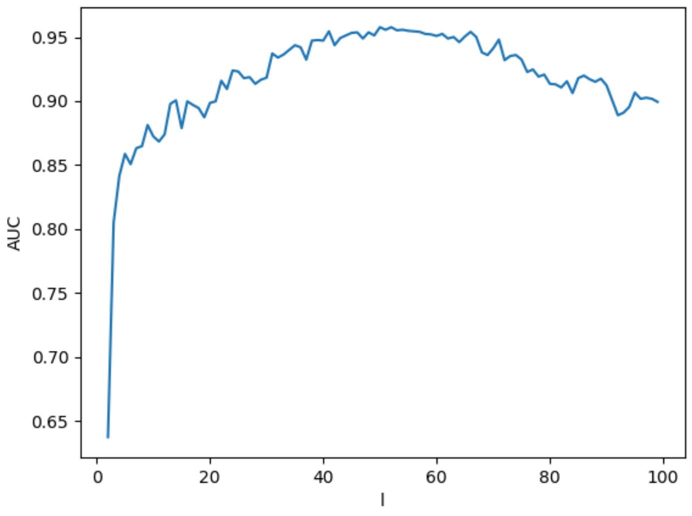
```
accuracy: 0.950000
AUC: 0.973958
```





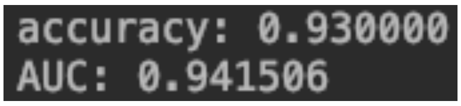
從0到100維Bayes的分類表現：



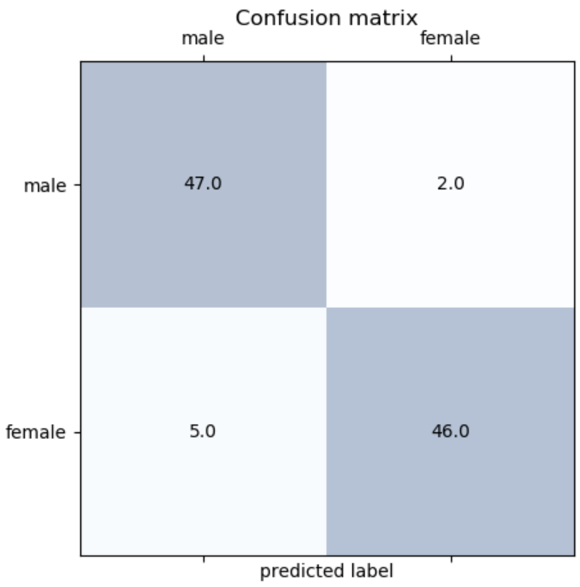


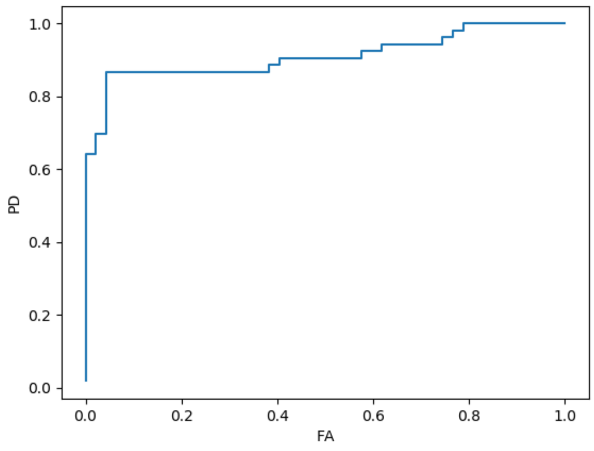
B. Naïve-Bayes classifier

最佳分類表現為1600維→30維：

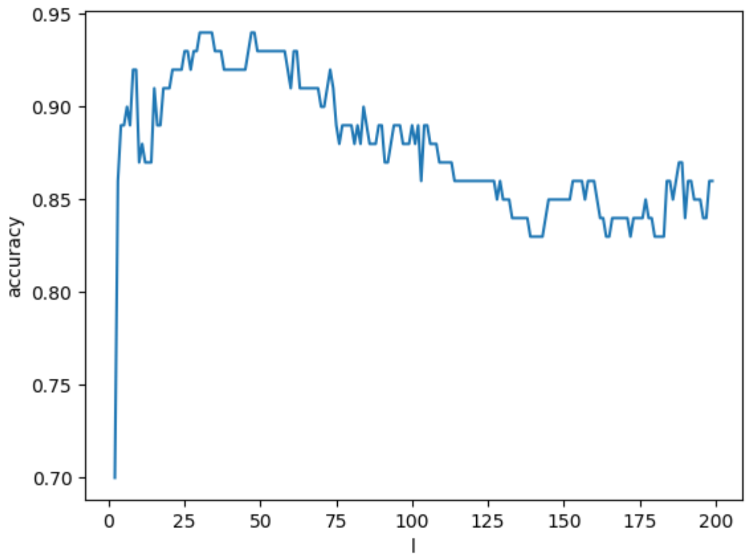


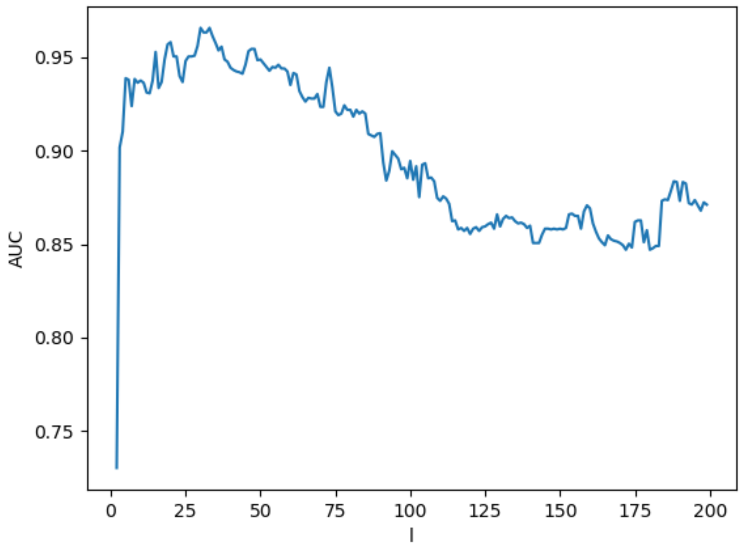
```
accuracy: 0.930000
AUC: 0.941506
```





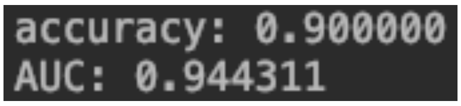
從0到200維Naïve-Bayes的分類表現：



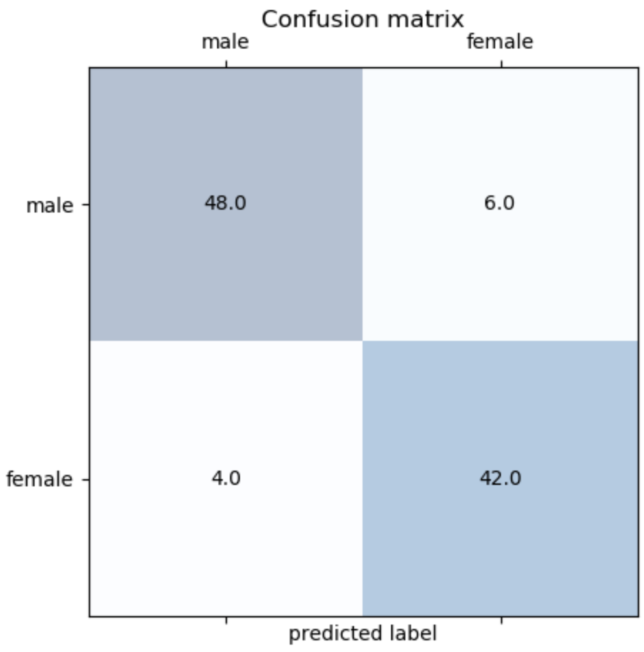


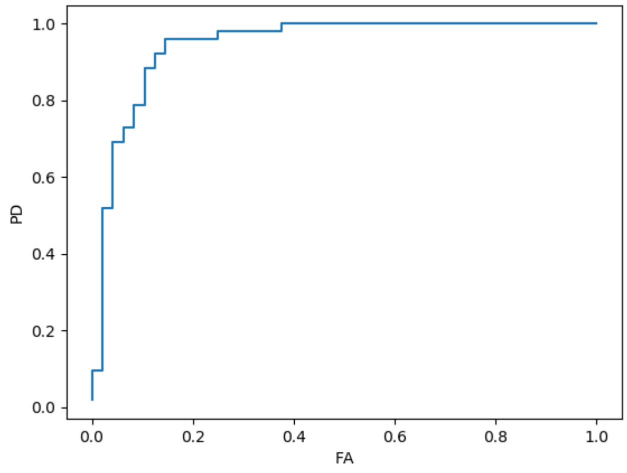
C. Linear classifier

最佳分類表現為1600維→30維：

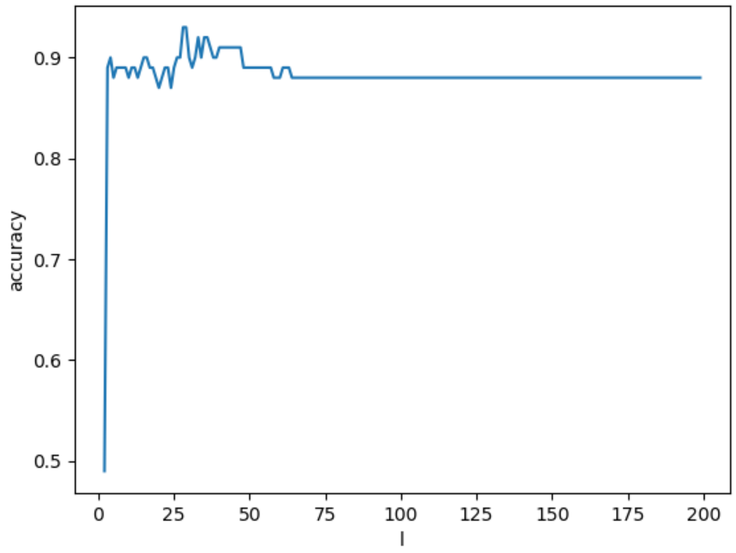


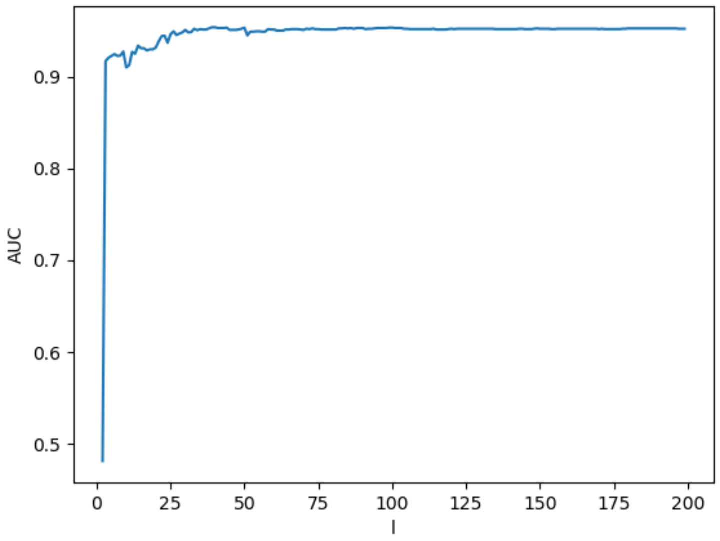
```
accuracy: 0.900000
AUC: 0.944311
```





從0到200維Linear classifier的分類表現：

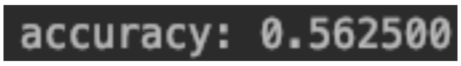




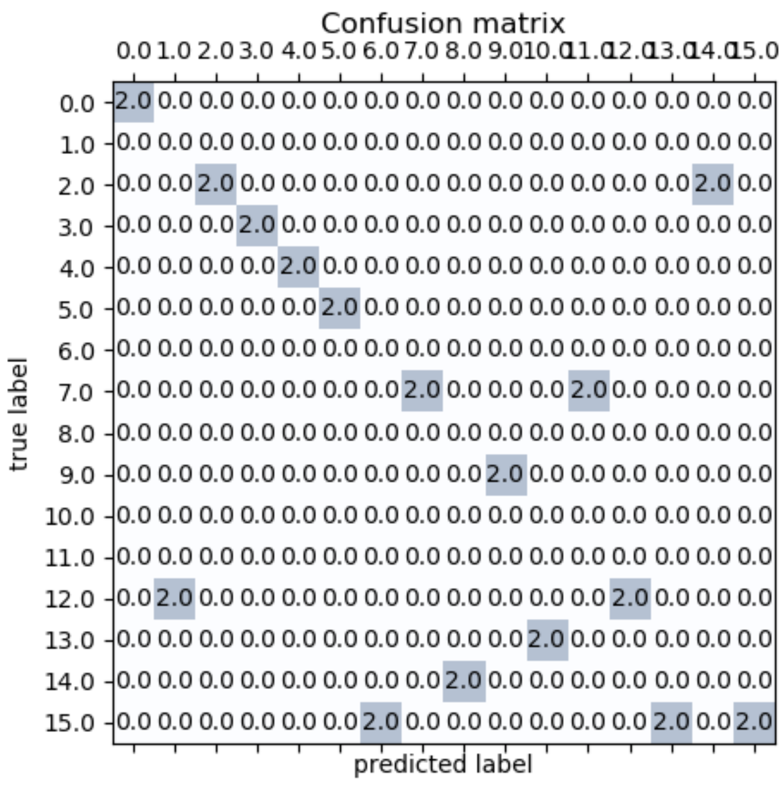
0416025 呂翊愷

Face classification by Naïve-Bayes classifier

最佳分類表現為1600維→30維：

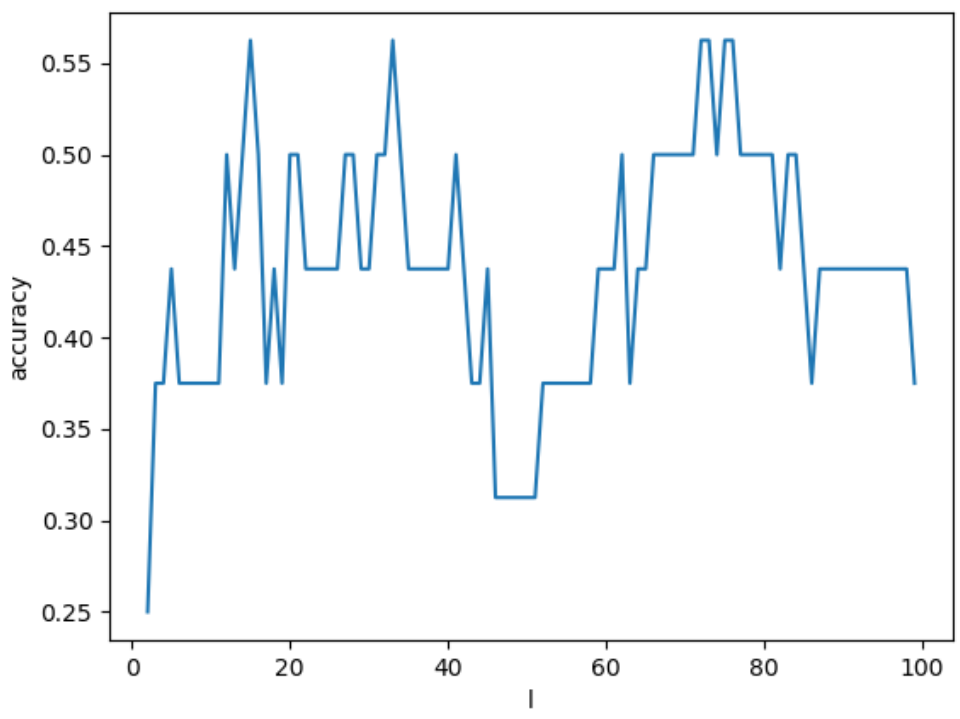


```
accuracy: 0.562500
```


Confusion matrix
true label
predicted label

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
3.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0
8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0
10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
13.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0
14.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	2.0

從0到100維Naïve-Bayes的分類表現：



0416025 呂翊愷

Analysis - Are the results what you expect? Why?

比較這三種方法，我發現PCA表現得最好，但PCA和eigenface都有個很大的缺點，就是受維度影響太大。如果沒有取適當數量的維度，分類表現就會很差，因此選擇適當的維度很重要。FDA的缺點就是只能分類兩種class的dataset，但FDA整體表現不差。

和上次的作業相比，在用相同分類器的情況下，降維後的表現明顯差一些，甚至表現很不穩定，準確率有時候很高，有時候很低，看來可能是因為降維後training data的資訊變少，這時training data取樣的好壞就會影響很大。整體而言，我認為這次作業的結果比我想像的還差，雖然原本就有預料到降維後可能表現較差，但沒想到會差這麼多，尤其是人臉辨識的表現並不是很好。

另外，我發現在上次作業中，不同分類器下不同dataset的準確率高低排序相當一致，依序為Wine, Iris, Ionosphere, Vertebral，而這次作業在降維後，不同dataset的準確率高低排序仍和上次作業的一致，看來降維前後不會影響dataset本身好壞的排序。

Code

I. Header.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import random
import cv2

# load iris dataset from csv
def loadIris():
    names = ['sepal.length', 'sepal.width', 'petal.length', 
'petal.width', 'variety']
    Dataset = pd.read_csv('./iris.csv', names=['sepal.length', 
'sepal.width', 'petal.length', 'petal.width', 'variety'], skiprows=0)
    DataVariety = Dataset['variety'].unique()
    return Dataset, DataVariety, names

# load wine dataset from csv
def loadWine():
    names=['variety', 'alco', 'malic', 'ash', 'alcal', 'mag', 'total', 
'flav', 'nonflav', 'proan', 'color', 'hue', 'OD', 'proline']
    Dataset = pd.read_csv('./wine.csv', names=['variety', 'alco', 
'malic', 'ash', 'alcal', 'mag', 'total',
                                                  'flav', 'nonflav', 
'proan', 'color', 'hue', 'OD', 'proline'], skiprows=0)
    DataVariety = Dataset['variety'].unique()
    return Dataset, DataVariety, names


# load ionos dataset from csv
def loadIonos():
    names = ['c', 'd', 'e', 'f', 'g',
             'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 
't',
             'u', 'v', 'w', 'x', 'y', 'z', 'ab', 'bc', 'cd', 'de', 'ef',
             'fg', 'gh', 'hi', 'variety']
    Dataset = pd.read_csv('./ionosphere.csv', names=['c', 'd', 'e', 'f', 
'g',
                                                    'h', 'i', 'j', 'k', 
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
                                                    'u', 'v', 'w', 'x', 
'y', 'z', 'ab', 'bc', 'cd', 'de', 'ef',
                                                    'fg', 'gh', 'hi', 
'variety'], skiprows=0)
    DataVariety = Dataset['variety'].unique()
    return Dataset, DataVariety, names


# load vertebral dataset from csv
def loadVertebral():
    names = ['a', 'b', 'c', 'd', 'e', 'f', 'variety']
    Dataset = pd.read_csv('./vertebral.csv', names=['a', 'b', 'c', 'd', 
'e', 'f', 'variety'], skiprows=0)
    DataVariety = Dataset['variety'].unique()
    return Dataset, DataVariety, names

# load face dataset
def loadFaceImg():
    img1 = cv2.imread('mP1.bmp', cv2.IMREAD_GRAYSCALE)
    img2 = cv2.imread('fP1.bmp', cv2.IMREAD_GRAYSCALE)

    img1Data = np.zeros([100, 1600])
    img2Data = np.zeros([100, 1600])
    originX = 0
    originY = 0
    img1Idx = 0
    img2Idx = 0
    idx = 0
    for p in range(10):
        for q in range(10):
            for i in range(40):
                for j in range(40):
                    img1Data[img1Idx][idx] = img1[originX + i][originY + 
j]
                    img2Data[img1Idx][idx] = img2[originX + i][originY + 
j]
                    idx += 1
            img1Idx += 1
            img2Idx += 1
            idx = 0
            originY += 40
        originY = 0
        originX += 40

    img1Flip = cv2.flip(img1Data, 1)
    img2Flip = cv2.flip(img2Data, 1)

    img1DF = pd.DataFrame(img1Data)
    img1DF['variety'] = 'male'
    img2DF = pd.DataFrame(img2Data)
    img2DF['variety'] = 'female'

    img1FDF = pd.DataFrame(img1Flip)
    img1FDF['variety'] = 'male'
    img2FDF = pd.DataFrame(img2Flip)
    img2FDF['variety'] = 'female'

    frames = [img1DF, img2DF, img1FDF, img2FDF]
    Dataset = pd.concat(frames, ignore_index=True)
    DataVariety = Dataset['variety'].unique()

    # randomize data
    randomData = Dataset.sample(frac=1)
    datasize = randomData.shape[0]

    # split the train data and test data
    trainData = randomData[:int(datasize * 0.75)]
    testData = randomData[int(datasize * 0.75):]

    return trainData, testData, DataVariety

def loadFaceImg2():
    img = cv2.imread('facesP1.bmp', cv2.IMREAD_GRAYSCALE)

    imgTrain = np.zeros([64, 1600])
    imgTest = np.zeros([16, 1600])

    originX = 0
    originY = 0
    imgTrainIdx = 0
    imgTestIdx = 0
    idx = 0
    for p in range(5):
        for q in range(16):
            for i in range(40):
                for j in range(40):
                    if p != 4:
                        imgTrain[imgTrainIdx][idx] = img[originX + i]
[originY + j]
                    else:
                        imgTest[imgTestIdx][idx] = img[originX + i]
[originY + j]
                    idx += 1
            if p != 4:
                imgTrainIdx += 1
            else:
                imgTestIdx += 1
            idx = 0
            originY += 40
        originY = 0
        originX += 40

    imgTrainFlip = cv2.flip(imgTrain, 1)
    imgTrainDF = pd.DataFrame(imgTrain)
    imgTrainFlipDF = pd.DataFrame(imgTrainFlip)
    trainFrame = pd.concat([imgTrainDF, imgTrainFlipDF], axis=0,
ignore_index=True)

    imgTestFlip = cv2.flip(imgTest, 1)
    imgTestDF = pd.DataFrame(imgTest)
    imgTestFlipDF = pd.DataFrame(imgTestFlip)
    testFrame = pd.concat([imgTestDF, imgTestFlipDF], axis=0,
ignore_index=True)

    trainVarArr = np.zeros([128, 1])
    var = 0
    for j in range(128):
        trainVarArr[j] = var
        var += 1
        if var % 16 == 0:
            var = 0

    testVarArr = np.zeros([32, 1])
    var = 0
    for j in range(32):
        testVarArr[j] = var
        var += 1
        if var % 16 == 0:
            var = 0

    trainVarDF = pd.DataFrame(trainVarArr, columns=['variety'])
    trainDataset = pd.concat([trainFrame, trainVarDF], axis=1)

    testVarDF = pd.DataFrame(testVarArr, columns=['variety'])
    testDataset = pd.concat([testFrame, testVarDF], axis=1)

    DataVariety = trainDataset['variety'].unique()

    return trainDataset, testDataset, DataVariety


# calculate accuracy
def calAccuracy(Success, Dataset):
    accuracy = Success / Dataset.shape[0]
    print("accuracy: %f" %(accuracy))

# draw confusion matrix
def drawConfusionMat(Confusion_mat, DataVariety):
    fig, axis = plt.subplots(figsize=(5, 5))
    axis.matshow(Confusion_mat, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(Confusion_mat.shape[0]):
        for j in range(Confusion_mat.shape[1]):
            axis.text(x=j, y=i, s=Confusion_mat[i,j], va='center',
ha='center')

    tick_marks = np.arange(len(DataVariety))
    plt.xticks(tick_marks, DataVariety, rotation=0)
    plt.yticks(tick_marks, DataVariety)

    plt.xlabel('predicted label')
    plt.ylabel('true label')
    plt.title('Confusion matrix')
    plt.show()

# draw ROC curve & calculate AUC
def ROC_AUC(Dataset, DataVariety, G, TestIdx):
    if (len(DataVariety) == 2):
        # draw ROC curve
        g_sort = sorted(G, reverse=True)
        TPR = []
        FPR = []
        labels = np.zeros([Dataset.shape[0]])
        predPosProb = np.zeros([Dataset.shape[0]])

        for i in range(len(g_sort)):
            TP = 0
            FP = 0
            FN = 0
            TN = 0
            threshold = g_sort[i]
            for j in range(len(G)):
                if G[j] >= g_sort[i] and Dataset.ix[TestIdx[j],
"variety"] == DataVariety[0]:
                    TP += 1
                    predPosProb[TestIdx[j]] += 1
                    labels[TestIdx[j]] = 1
                elif G[j] >= g_sort[i] and Dataset.ix[TestIdx[j],
"variety"] == DataVariety[1]:
                    FP += 1
                    predPosProb[TestIdx[j]] += 1
                    labels[TestIdx[j]] = 0
                elif G[j] < g_sort[i] and Dataset.ix[TestIdx[j],
"variety"] == DataVariety[0]:
                    FN += 1
                    labels[TestIdx[j]] = 1
                else:
                    TN += 1
                    labels[TestIdx[j]] = 0
            TPR.append(TP / (TP + FN))
            FPR.append(FP / (FP + TN))

        plt.xlabel('FA')
        plt.ylabel('PD')
        plt.plot(FPR, TPR)
        plt.show()

        # calculate AUC
        auc = 0
        for i in range(len(TPR) - 1):
            auc += (TPR[i] + TPR[i + 1]) * (FPR[i + 1] - FPR[i]) / 2
        print("AUC: %f" % (auc))
```

II. Likelihood.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import statistics as stat
import random
from scipy.stats import norm
from scipy.stats import multivariate_normal
import math

def maxLikelihood(data):
    data = pd.DataFrame(data)
    newData = data.drop("variety", axis=1)

    mu = newData.sum()
    mu = mu.values.reshape([newData.shape[1], 1])
    mu = mu / newData.shape[0]

    cov = np.zeros([newData.shape[1], newData.shape[1]])
    for index, row in newData.iterrows():
        xk = row.values.reshape((newData.shape[1], 1))
        vector = xk - mu

        for i in range(newData.shape[1]):
            for j in range(newData.shape[1]):
                cov[i][j] += vector[i] * vector[j]

    cov = cov / newData.shape[0]

    return mu, cov

def gi(data, mu, cov, P, dataVarLen):
    data = pd.DataFrame(data)
    newData = data.drop("variety", axis=0)

    x = np.zeros([newData.shape[0], 1])
    for i in range(len(x)):
        x[i] = newData.ix[i]


    detCov = np.linalg.det(cov)

    expPow = -1 / 2 * np.dot(np.transpose(x - mu), 
np.dot(np.linalg.inv(cov), x - mu))
    div = math.pow(2 * math.pi, dataVarLen) * abs(detCov)
    pdfVal = 1 / math.sqrt(div) * math.exp(expPow)

    gi = pdfVal * P
    # gi = math.log(pdfVal * P)

    return gi


def likelihood(trainData, testData, dataVariety):
    success = 0
    confusion_mat = np.zeros([len(dataVariety), len(dataVariety)])
    g_unsort = []
    testIdX = []

    mu = np.zeros((len(dataVariety), trainData.shape[1] - 1, 1))
    cov = np.zeros((len(dataVariety), trainData.shape[1] - 1, 
trainData.shape[1] - 1))

    prob = []

    # train data
    for i in range(len(dataVariety)):
        classData = trainData[trainData['variety'] == dataVariety[i]]
        mu[i], cov[i] = maxLikelihood(classData)
        prob.append(classData.shape[0] / trainData.shape[0])

    # test data
    for index, row in testData.iterrows():
        g = np.zeros([len(dataVariety)])
        for i in range(len(dataVariety)):
            g[i] = gi(row, mu[i], cov[i], prob[i], len(dataVariety))
        g_unsort.append(g[0] - g[1])
        testIdX.append(index)
        predict = np.argmax(g)


        if (dataVariety[predict] == row["variety"]):
            success += 1

        real = -1
        for i in range(len(dataVariety)):
            if dataVariety[i] == row["variety"]:
                real = i
        confusion_mat[predict][real] += 1
    return g_unsort, testIdX, success, confusion_mat
```

III. Bayes.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import statistics as stat
import random
from scipy.stats import norm
import math

def pdf(x, mean, var):
    pdfVal = 1 / math.sqrt(2 * math.pi * var) * math.exp(-pow((x - mean),
2) / (2 * var))
    return pdfVal


def bayes(trainData, testData, dataVariety, names):
    success = 0
    confusion_mat = np.zeros([len(dataVariety), len(dataVariety)])
    g = []
    testIdX = []

    # train data
    mean = np.zeros((len(dataVariety), trainData.shape[1] - 1))
    var = np.zeros((len(dataVariety), trainData.shape[1] - 1))

    prob = []
    for i in range(len(dataVariety)):
        classData = trainData[trainData['variety'] == dataVariety[i]]
        classData = classData.drop("variety", axis=1)
        prob.append(classData.shape[0] / trainData.shape[0])
        for j in range(classData.shape[1]):
            classDataAttr = classData.iloc[:, j]
            mean[i][j] = stat.mean(classDataAttr)
            var[i][j] = stat.variance(classDataAttr)
    # test data
    for index, row in testData.iterrows():
        pdfVal = np.zeros((len(dataVariety), testData.shape[1] - 1))
        post = np.zeros([len(dataVariety)])
        for i in range(len(dataVariety)):
            multiPdf = 1
            newj = 0
            for j in range(testData.shape[1]):
                if names[j] == "variety":
                    continue
                pdfVal[i][newj] = pdf(row.ix[j], mean[i][newj], var[i]
[newj])
                multiPdf = multiPdf * pdfVal[i][newj]
                newj = newj + 1
            post[i] = prob[i] * multiPdf
        g.append(post[0] - post[1])
        testIdX.append(index)
        predict = np.argmax(post)

        if (dataVariety[predict] == row["variety"]):
            success += 1

        real = -1
        for i in range(len(dataVariety)):
            if dataVariety[i] == row["variety"]:
                real = i
        confusion_mat[predict][real] += 1

    return g, testIdX, success, confusion_mat
```

IV. LinearClassifier.py

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


# perception algorithm
def sign(z):
    if z > 0:
        return 1
    else:
        return -1

# inner product
def dot(w, yy, x):
    for i in range(len(w)):
        w[i] = w[i] + yy[0] * x[i]
    return w

def linearClassifier(trainData, testData, dataVariety):
    success = 0
    confusion_mat = np.zeros([len(dataVariety), len(dataVariety)])
    g = []
    testIdx = []

    w = np.zeros((trainData.shape[1]))

    error = 1
    iterator = 0

    while error != 0 and iterator < 10:
        error = 0
        for index, row in trainData.iterrows():
            arr = np.array(row[:trainData.shape[1] - 1])
            x = np.concatenate((np.array([1.]), arr))
            if row['variety'] == dataVariety[0]:
                y = np.array([1.])
            elif row['variety'] == dataVariety[1]:
                y = np.array([-1.])
            if sign(np.dot(w, x)) != y[0]:
                w = dot(w, y, x)
                error += 1
                iterator += 1

    for index, row in testData.iterrows():
        arr = np.array(row[:trainData.shape[1] - 1])
        x = np.concatenate((np.array([1.]), arr))
        if row['variety'] == dataVariety[0]:
            real = 0
            y = np.array([1.])
        elif row['variety'] == dataVariety[1]:
            real = 1
            y = np.array([-1.])

        g.append(np.dot(w, x))
        testIdx.append(index)

        predict = sign(np.dot(w, x))
        if predict == y[0]:
            success += 1

        if predict == 1:
            confusion_mat[0][real] += 1
        elif predict == -1:
            confusion_mat[1][real] += 1

    return g, testIdx, success, confusion_mat
```

V. FLD.py

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

```
def SwSb(dataset, dataVariety):
    mean = np.zeros((len(dataVariety), dataset.shape[1] - 1, 1))
    cov = np.zeros((len(dataVariety), dataset.shape[1] - 1,
dataset.shape[1] - 1))

    prob = []
    Sw = np.zeros((dataset.shape[1] - 1, dataset.shape[1] - 1))
    Sb = np.zeros((dataset.shape[1] - 1, dataset.shape[1] - 1))

    # calculate Sw
    for i in range(len(dataVariety)):
        classData = dataset[dataset['variety'] == dataVariety[i]]
        newData = classData.drop("variety", axis=1)

        mu = newData.sum()
        mu = mu.values.reshape([newData.shape[1], 1])
        mu = mu / newData.shape[0]
        mean[i] = mu

        for index, row in newData.iterrows():
            covar = np.zeros([newData.shape[1], newData.shape[1]])
            xk = row.values.reshape((newData.shape[1], 1))
            vector = xk - mu

            np.matmul(vector, np.transpose(vector), covar)
            cov[i] += covar

        cov[i] = cov[i] / newData.shape[0]

        prob.append(classData.shape[0] / dataset.shape[0])

    for i in range(len(dataVariety)):
        Sw += prob[i] * cov[i]

    Sw /= len(dataVariety)

    # calculate Sb
    mean0 = np.zeros((dataset.shape[1] - 1, 1))
    for i in range(len(dataVariety)):
        mean0 += prob[i] * mean[i]

    mat = np.zeros((dataset.shape[1] - 1, dataset.shape[1] - 1))
    for i in range(len(dataVariety)):
        vector = mean[i] - mean0
        np.matmul(vector, np.transpose(vector), mat)
        Sb += prob[i] * mat
    return mean, Sw, Sb

def FLD(dataset, dataVariety):
    # randomize data
    randomData = dataset.sample(frac=1)
    datasize = randomData.shape[0]

    # split the train data and test data
    trainData = randomData[:int(datasize * 0.75)]
    testData = randomData[int(datasize * 0.75):]

    mean, Sw, Sb = SwSb(trainData, dataVariety)

    # calculate w
    w = np.zeros((trainData.shape[1] - 1, 1))
    vector = mean[0] - mean[1]
    np.matmul(np.linalg.inv(Sw), vector, w)

    # calculate J before projection
    mean1, Sw1, Sb1 = SwSb(dataset, dataVariety)
    matTemp1 = np.zeros((dataset.shape[1] - 1, dataset.shape[1] - 1))
    np.matmul(np.linalg.inv(Sw1), Sb1, matTemp1)
    J1 = np.trace(matTemp1)
    print("separability measures before the projection: %f" %J1)

    # projection of x
    G = []
    testIdX = []
    dataVar = []
    for index, row in dataset.iterrows():
        arr = np.array(row[:dataset.shape[1] - 1])
        x = arr.reshape((dataset.shape[1] - 1, 1))
        g = np.dot(np.transpose(w), x)
        G.append(g[0][0])
        testIdX.append(index)
        dataVar.append(row[dataset.shape[1] - 1])

    Gframe = pd.DataFrame(G, columns=['gValue'])
    varFrame = pd.DataFrame(dataVar, columns=['variety'])
    newFrame = pd.concat([Gframe, varFrame], axis=1,
join_axes=[Gframe.index])

    # calculate J after projection
    mean2, Sw2, Sb2 = SwSb(newFrame, dataVariety)
    matTemp2 = np.zeros((newFrame.shape[1] - 1, newFrame.shape[1] - 1))
    np.matmul(np.linalg.inv(Sw2), Sb2, matTemp2)
    J2 = np.trace(matTemp2)
    print("separability measures after the projection: %f" % J2)

    return G, testIdX
```

VI. PCA.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import random
import statistics as stat
import heapq

import Likelihood
import Bayes
import Header
```

0416025 呂翊愷

```
import LinearClassifier

def normalize(data, mu):
    dataVar = np.array(data['variety'])
    dataVar = dataVar.reshape([data.shape[0], 1])
    dropData = data.drop("variety", axis=1)

    var = []
    for i in range(dropData.shape[1]):
        list = []
        list = dropData.ix[:, i]
        var.append(stat.variance(list))

    arrData = np.array(dropData)
    for i in range(arrData.shape[0]):
        for j in range(arrData.shape[1]):
            arrData[i][j] -= mu[j]
            arrData[i][j] /= var[j]
    DataF = pd.DataFrame(arrData)

    dataVarFrame = pd.DataFrame(dataVar, columns=['variety'])
    normFrame = pd.concat([DataF, dataVarFrame], axis=1)

    return normFrame

def calCov(data):
    data = pd.DataFrame(data)

    mu = data.sum()
    mu = mu.values.reshape([data.shape[1], 1])
    mu = mu / data.shape[0]

    cov = np.zeros([data.shape[1], data.shape[1]])

    for index, row in data.iterrows():
        xk = row.values.reshape((data.shape[1], 1))
        vector = xk - mu

        for i in range(data.shape[1]):
            for j in range(data.shape[1]):
                cov[i][j] += vector[i] * vector[j]

    cov = cov / data.shape[0]

    return cov

def project(A, data):
    # projection of data
    dataVar = np.array(data['variety'])
    dataVar = dataVar.reshape([data.shape[0], 1])

    newData1 = data.drop("variety", axis=1)
    newData1 = np.array(newData1)
    newData1 = np.transpose(newData1)
    projData = np.zeros([A.shape[1], newData1.shape[1]])
    np.matmul(np.transpose(A), newData1, projData)
    projData = np.transpose(projData)
    projDF = pd.DataFrame(projData)

    dataVarFrame = pd.DataFrame(dataVar, columns=['variety'])
    newFrame = pd.concat([projDF, dataVarFrame], axis=1)

    return newFrame

def PCA(dataset, dataVariety, names):
    # randomize data
    randomData = dataset.sample(frac=1)
    datasize = randomData.shape[0]

    # split the train data and test data
    trainData = randomData[:int(datasize * 0.75)]
    testData = randomData[int(datasize * 0.75):]

    # standardize train data
    dropTrainData = trainData.drop("variety", axis=1)
    trainMean = dropTrainData.sum()
    trainMean = trainMean.values.reshape([dropTrainData.shape[1], 1])
    trainMean = trainMean / dropTrainData.shape[0]
    newtrainData = normalize(trainData, trainMean)

    # standardize test data
    newtestData = normalize(testData, trainMean)

    # calculate covX
    dropTrainData2 = newtrainData.drop("variety", axis=1)
    covX = calCov(dropTrainData2)

    # calculate A
    eigValX, eigVecX = np.linalg.eig(covX)
    L = 3
    maxEigIdx = np.argsort(-eigValX)
    A = []
    for i in range(L):
        A.append(eigVecX[:, maxEigIdx[i]])
    A = np.array(A)
    A = np.transpose(A)

    # projection of train data
    projTrainFrame = project(A, newtrainData)

    # projection of test data
    projTestFrame = project(A, newtestData)


    # # classify test data by likelihood
    # g1, testIdx1, success1, confusion_mat1 =
Likelihood.likelihood(projTrainFrame, projTestFrame, dataVariety)
    # Header.calAccuracy(success1, projTestFrame)
    # Header.ROC_AUC(projTestFrame, dataVariety, g1, testIdx1)
    # Header.drawConfusionMat(confusion_mat1, dataVariety)
    # # classify test data by bayes
    # names = []
    # for i in range(projTestFrame.shape[1] - 1):
    #     names.append('0')
    # names.append('variety')
    # g2, testIdx2, success2, confusion_mat2 =
Bayes.bayes(projTrainFrame, projTestFrame, dataVariety, names)
    # Header.calAccuracy(success2, projTestFrame)
    # Header.ROC_AUC(projTestFrame, dataVariety, g2, testIdx2)
    # Header.drawConfusionMat(confusion_mat2, dataVariety)

    # classify test data by linear classifier
    g3, testIdx3, success3, confusion_mat3 =
LinearClassifier.linearClassifier(projTrainFrame, projTestFrame,
dataVariety)
    Header.calAccuracy(success3, projTestFrame)
    Header.ROC_AUC(projTestFrame, dataVariety, g3, testIdx3)
    Header.drawConfusionMat(confusion_mat3, dataVariety)
```

VII. Eigenface.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
import random
import statistics as stat
import heapq

import Likelihood
import Bayes
import Header
import LinearClassifier
import PCA


def eigenface(trainData, testData, dataVariety):
    # standardize train data
    dropTrainData = trainData.drop("variety", axis=1)
    trainMean = dropTrainData.sum()
    trainMean = trainMean.values.reshape([dropTrainData.shape[1], 1])
    trainMean = trainMean / dropTrainData.shape[0]
    newtrainData = PCA.normalize(trainData, trainMean)

    # calculate xT * x and its eigenvector
    normTrainData = newtrainData.drop("variety", axis=1)
    normTrainData = np.array(normTrainData)
    X = np.transpose(normTrainData)

    tempMat = np.zeros([X.shape[1], X.shape[1]])
    np.matmul(np.transpose(X), X, tempMat)
    eigValX, eigVecX = np.linalg.eig(tempMat)
    # calculate X * eigenvector
    newEigVecX = np.zeros([X.shape[0], eigVecX.shape[1]])
    newEigVecX = np.matmul(X, eigVecX)

    # normalize eigenvector
    newEigVecX = np.transpose(newEigVecX)
    length = np.linalg.norm(newEigVecX, axis=1)
    for i in range(newEigVecX.shape[0]):
        newEigVecX[i] /= length[i]
    normEigVec = np.transpose(newEigVecX)


    # calculate A
    L = 20
    maxEigIdx = np.argsort(-eigValX)
    A = []
    for i in range(L):
        A.append(normEigVec[:, maxEigIdx[i]])
    A = np.array(A)
    A = np.transpose(A)

    newtestData = PCA.normalize(testData, trainMean)

    # projection of train data
    projTrainFrame = PCA.project(A, newtrainData)

    # projection of test data
    projTestFrame = PCA.project(A, newtestData)

    # # classify test data by likelihood
    # g1, testIdx1, success1, confusion_mat1 = 
Likelihood.likelihood(projTrainFrame, projTestFrame, dataVariety)
    # Header.calAccuracy(success1, projTestFrame)
    # Header.ROC_AUC(projTestFrame, dataVariety, g1, testIdx1)
    # Header.drawConfusionMat(confusion_mat1, dataVariety)

    # classify test data by bayes
    names = []
    for i in range(projTestFrame.shape[1] - 1):
        names.append('0')
    names.append('variety')
    g2, testIdx2, success2, confusion_mat2 = Bayes.bayes(projTrainFrame, 
projTestFrame, dataVariety, names)
    Header.calAccuracy(success2, projTestFrame)
    Header.drawConfusionMat(confusion_mat2, dataVariety)

    # classify test data by linear classifier
    # g3, testIdx3, success3, confusion_mat3 = 
LinearClassifier.linearClassifier(projTrainFrame, projTestFrame, 
dataVariety)
    # Header.calAccuracy(success3, projTestFrame)
    # Header.ROC_AUC(projTestFrame, dataVariety, g3, testIdx3)
    # Header.drawConfusionMat(confusion_mat3, dataVariety)
```

VIII. main.py

```
import pandas as pd

import Header
import Likelihood
import Bayes
import LinearClassifier
import FLD
import PCA
import Eigenface

# dataset, dataVariety, names = Header.loadIris()
# dataset, dataVariety, names = Header.loadWine()
# dataset, dataVariety, names = Header.loadIonos()
# dataset, dataVariety, names = Header.loadVertebral()
# trainData, testData, dataVariety = Header.loadFaceImg()
trainData, testData, dataVariety = Header.loadFaceImg2()

# g, testIdx, success, confusion_mat = Likelihood.likelihood(dataset,
dataVariety)
# g, testIdx, success, confusion_mat = Bayes.bayes(dataset, dataVariety,
names)
# g, testIdx, success, confusion_mat =
LinearClassifier.linearClassifier(dataset, dataVariety)

# frames = [trainData, testData]
# dataset = pd.concat(frames, ignore_index=True)

# g, testIdx = FLD.FLD(dataset, dataVariety)
# Header.ROC_AUC(dataset, dataVariety, g, testIdx)

# PCA.PCA(dataset, dataVariety, names)
Eigenface.eigenface(trainData, testData, dataVariety)

# Header.calAccuracy(success, dataset)
# Header.drawConfusionMat(confusion_mat, dataVariety)
```