HW2 Report

0516034 楊翔鈞

這次作業主要分成三個部分，第一部分是用 FLD 與 LDA 做 dimension reduction；第二部分是利用 PCA 將原始資料降低維度，觀察降低維度前後對於不同 classifier 的表現差異；第三部分是用 eigenface 來做性別識別還有人臉識別。下面會分別說明這三個部分做了什麼實驗以及比較實驗結果。

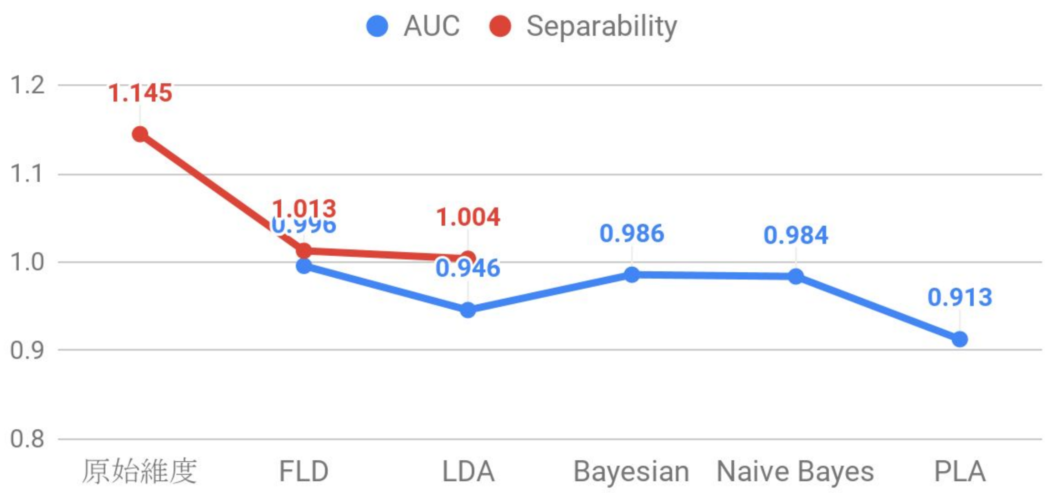
Task 1. FLD/LDA

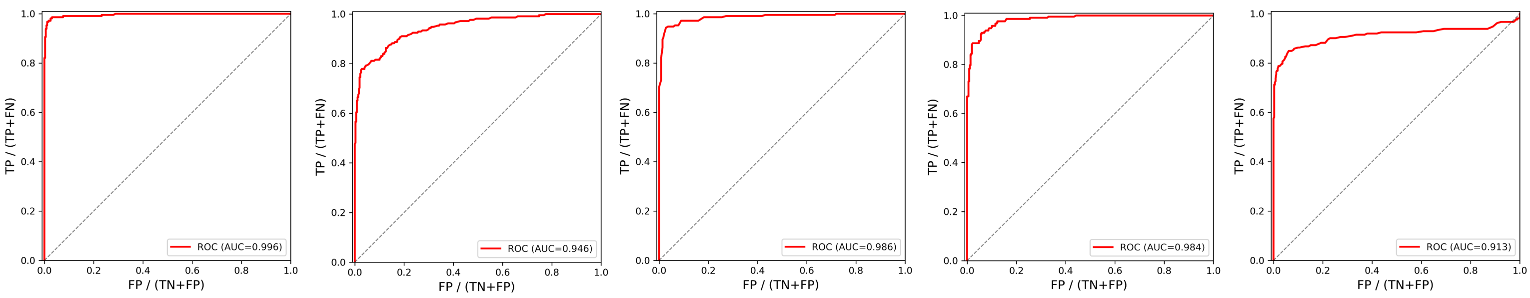
這部分使用的 dataset 是第一次作業的 breast cancer、ionosphere、iris、wine 跟 optical digits，利用 FLD 與 LDA 將資料降到一維後，計算降維前後的 separability，講義上有許多不同的 separability measures，不過大部分都要計算矩陣行列式，對於較高維度的矩陣，其行列式本來就會比較大，也就不容易做「不同維度間的 separability 比較」，所以我最後選擇用下面這個 separability measure：

$$J = \frac{tr\{S_m\}}{tr\{S_w\}}$$

Breast cancer

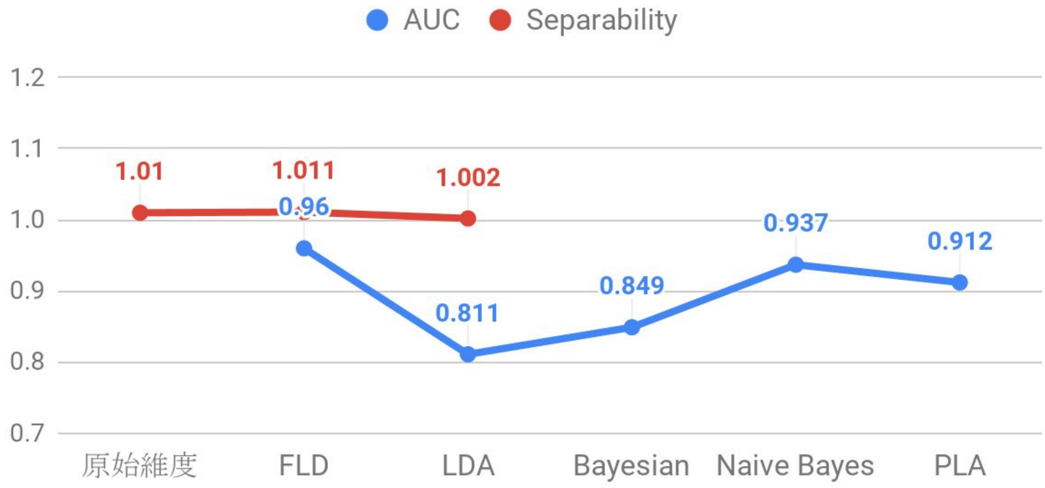
這組是 2-class 然後維度 30 的資料，下圖紅線的部分是做 dimension reduction 前後的 separability 比較，藍線是 FLD、LDA、Bayesian、Naive Bayes、Pocket Learning Algorithm（以下簡稱 PLA）的 AUC 比較，另外折線圖下方也附上 ROC 曲線，ROC 的順序由左而右分別對應折線圖的順序（不含原始維度）。

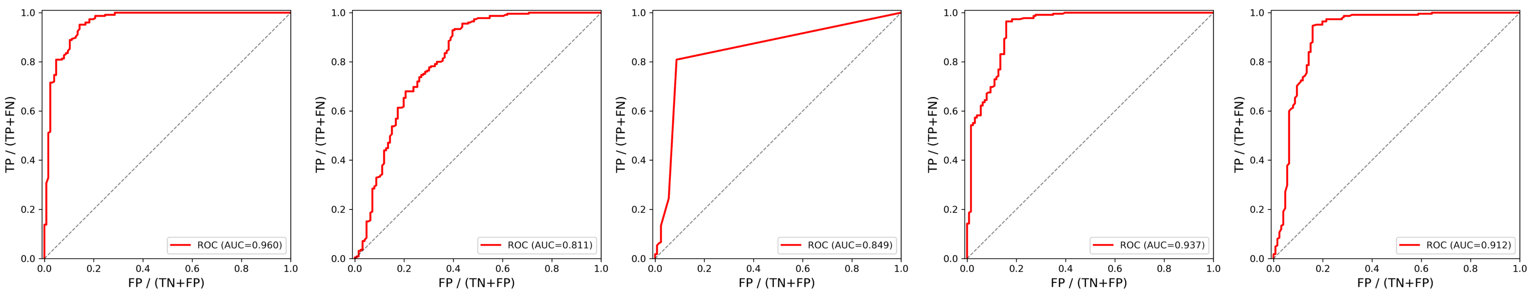




Ionosphere

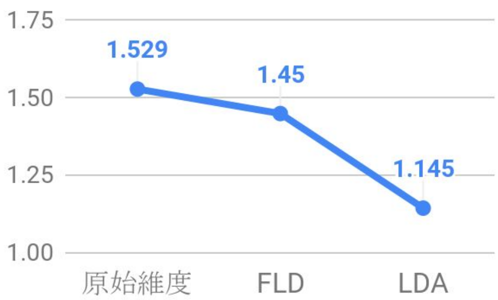
這組是 2-class 然後維度 34 的資料，跟上面折線圖的格式一樣，紅色是 separability，藍色是 AUC，ROC 的順序由左而右分別對應折線圖的順序（不含原始維度）。

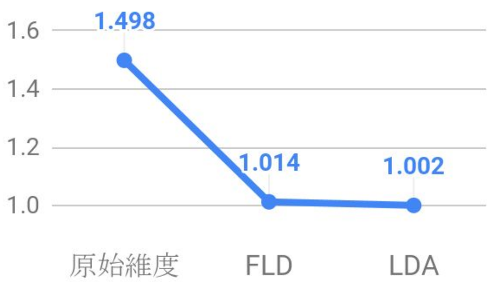


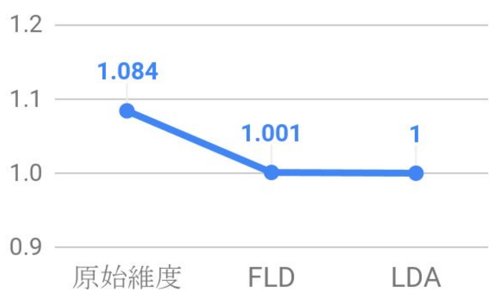


Iris、Wine、Optical digits

再來是三組 multi-class 的資料，由左而右為 iris、wine、optical digits，維度分別是 4、13、64，因為是 multi-class 所以只有計算 separability，沒有 ROC 跟 AUC。







觀察

- Separability：在這五組資料中，FLD 的 separability 都比 LDA 的高，不過我原本以為 FLD 跟 LDA 的 separability 會比原始維度的還低，因為降低維度的同時，資料含有的訊息也跟著減少了，但是唯獨 ionosphere 這組資料跑出來的結果不是這樣，我覺得原因可能是資料本身，或許 ionosphere 在原始維度很凌亂，透過 FLD 降維之後把一些雜亂的 attribute 過濾掉，使資料比較好被區分。
- AUC：FLD 與 LDA 的 AUC 大小關係跟兩者的 separability 一樣，FLD 比 LDA 高，代表在同樣投影到一維的情況下，FLD 比起 LDA 更容易將兩個 class 區分開來。另外也可以發現 FLD 的 AUC 比 Bayesian 和 Naive Bayes 的都還要高，如果要針對這兩組 2-class 的資料設計 classifier 的話，FLD 或許是個比較好的選擇，相較於 Bayesian 跟 Naive Bayes 需要計算各種機率分布，FLD 只要計算投影向量然後把資料投影上去，並找出合適的閾值就好，而且做出來的 AUC 還比較高。

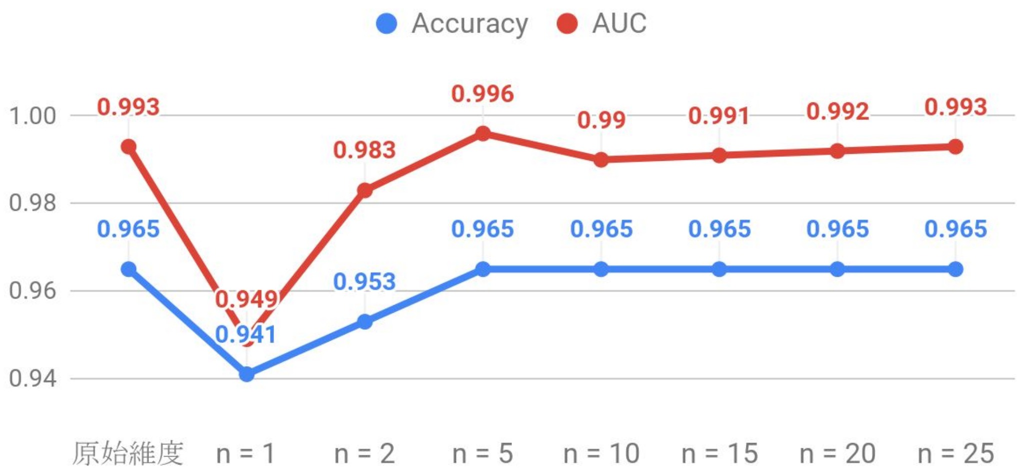
Task 2. PCA and classification

這部分要比較不同 principle component 數量的 PCA 對於訓練 classifier 會有什麼影響，然後我選擇的是 breast cancer 與 ionosphere 這兩組資料；classifier 則是 Bayesian、Naive Bayes 與 PLA，另外資料會被切成七比三，七等分用於訓練模型。

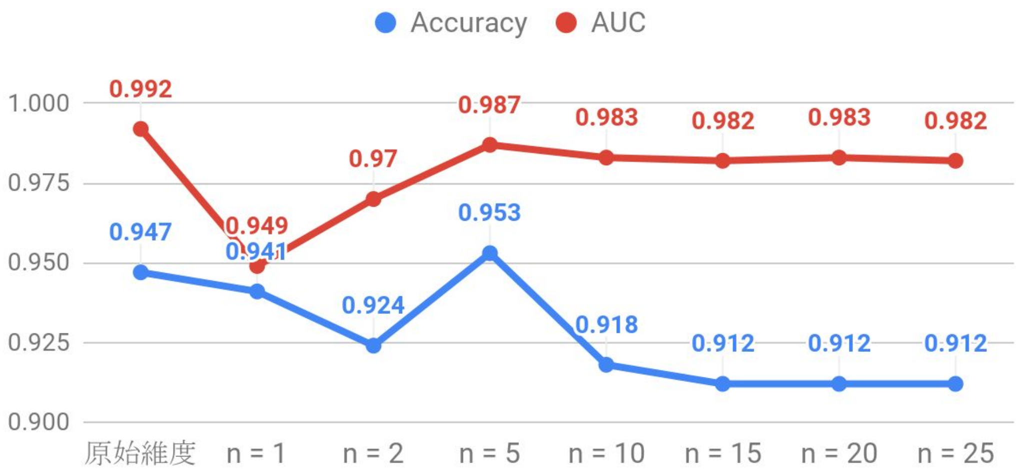
Breast cancer

這組是 2-class 然後維度為 30 的資料，利用 PCA 將原始資料投影到 1、2、5、10、15、20、25 維後，觀察各種 classifier 的訓練結果。

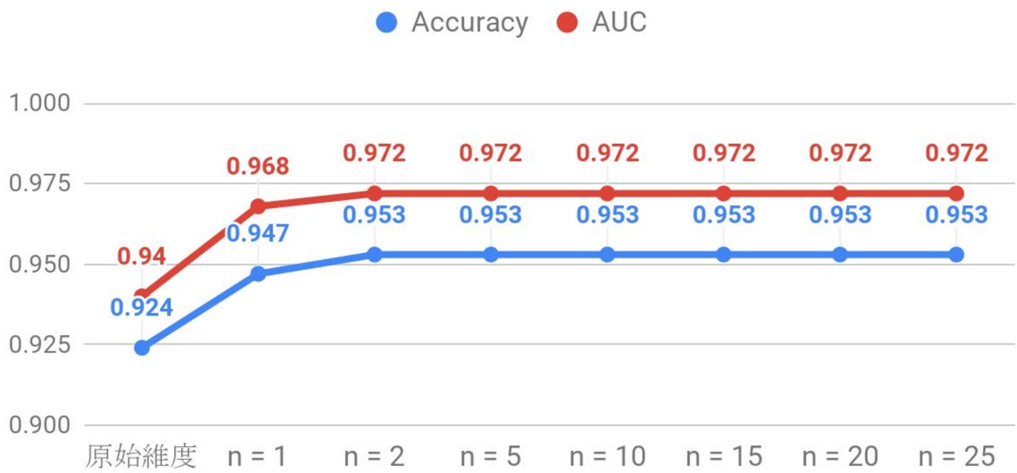
首先是 Bayesian 的結果，藍色為準確度；紅色為 AUC，折線圖中橫軸的 n 代表 principle component 的數量。



再來是 Naive Bayes 的結果。



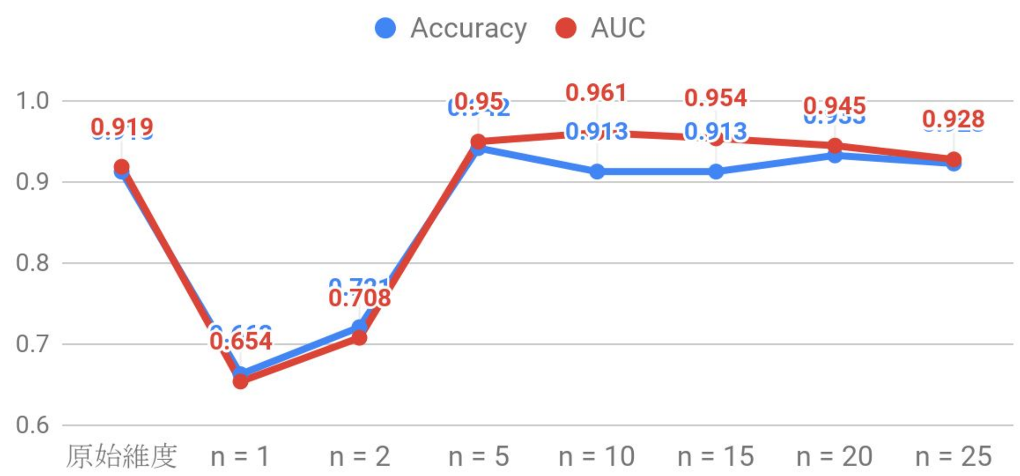
最後是 PLA 的結果。



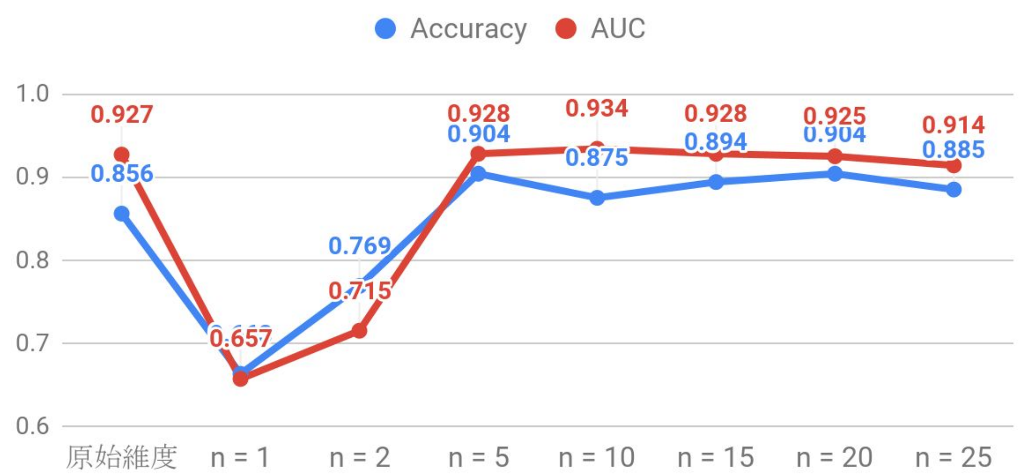
Ionosphere

這組是 2-class 然後維度為 34 的資料，利用 PCA 將原始資料投影到 1、2、5、10、15、20、25 維後，觀察各種 classifier 的訓練結果。

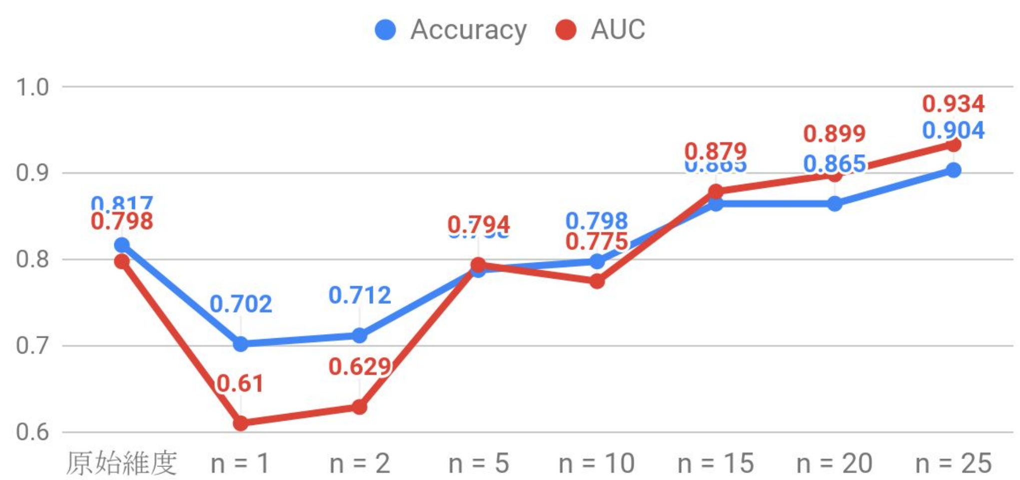
首先是 Bayesian 的結果。



再來是 Naive Bayes 的結果。



最後是 PLA 的結果。



觀察

因為準確度只是選定某一個閾值去做分類計算出來的結果，就不特別比較，主要還是以 AUC 為主，我自己對 AUC 的理解是：AUC 越大表示資料越容易被區分開，也就是模型辨別不同 class 的能力越好。而我從上面六張圖表觀察到兩件事情：

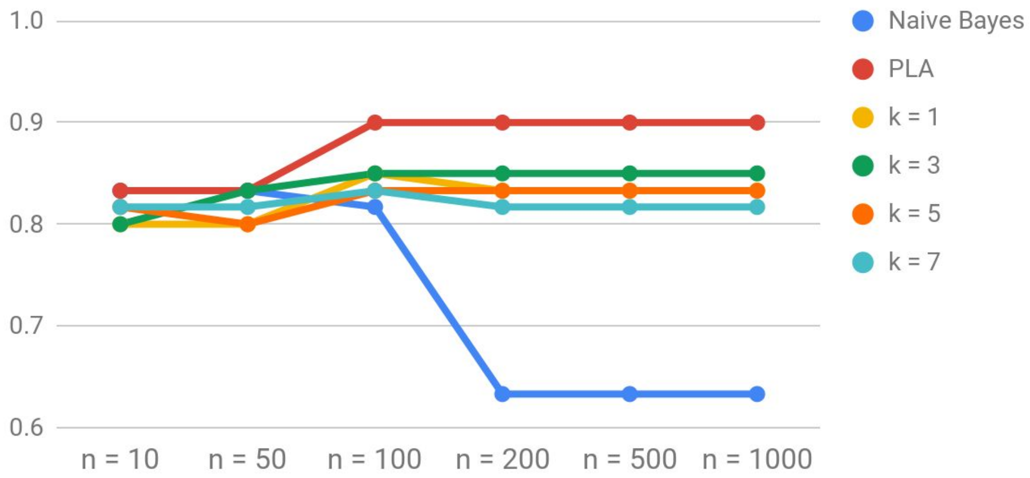
- PLA 的 AUC 在投影維度最大的時候有最大值，我覺得這跟 PLA 的運作原理有關，PLA 分類的方法是尋找一個能將資料分開的 hyperplane，如果現在是投影到一維空間，那 hyperplane 就只能選擇一個一維的點去把投影後的資料分開；投影到二維空間就只能選一條線把資料分開，投影到三維則是選一個三維空間中的面，依此類推，所以資料的維度越高，hyperplane 的選擇就越多元，AUC 也就隨著 n 變大而逐漸增加。
- Bayesian 與 Naive Bayes 的 AUC 則是在 n = 5 到 10 這個區間內產生最大值，這部分我不知道為什麼，我原先以為是 principle compenent 數量在這個區間的時候會有最大的 separability，但是後來把資料在不同 principle component 下的 separability 算出來後，發現 separability 跟 principle component 的數量成正比，並沒有在 5 到 10 的區間出現較高的值，所以從實驗中只知道較多的 principle component 不一定會讓模型的辨識效果變好，但還不清楚原因。

Task 3. eigenface and classification

這部分要用 eigenface 來做性別辨識以及人臉辨識，每個人臉皆為 40 x 40 的圖片，總共 1600 個像素，利用 PCA 選出 10、50、100、200、500、1000 個 principle component 作為 eigenface，並用 Naive Bayes、PLA、KNN 三種模型來做辨識，PLA 只會用在性別辨識的地方，這邊不用 Bayesian 是因為資料維度太高，算矩陣行列式跟反矩陣的時候 Python 的小數會溢位，另外，每張訓練用的圖片都會被左右翻轉後加入訓練集當中，用以增加訓練資料的數量。

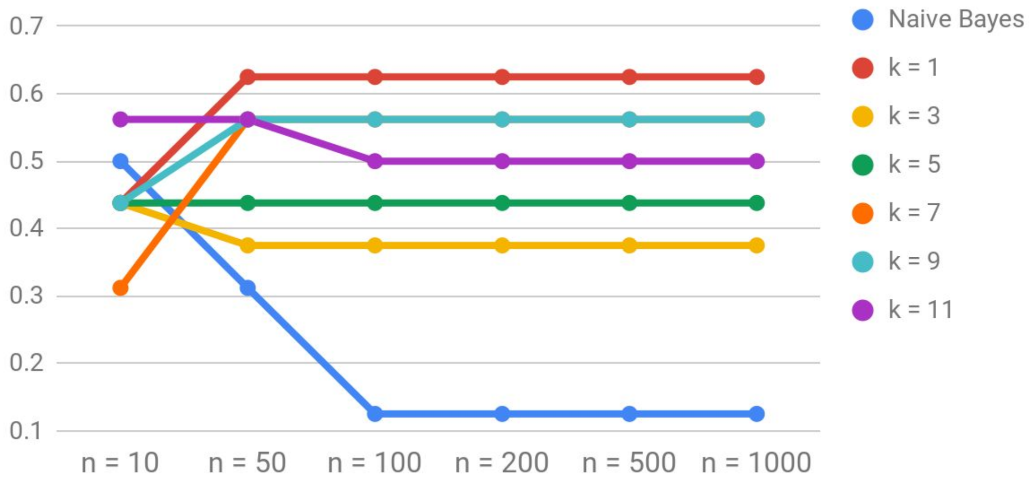
Gender classification

男生、女生的臉部照片各 100 張，將這些照片分成七比三後，七等分用於訓練。模型的準確度如下，橫軸的 n 代表 principle component 的數量，圖例裡面的 k 是 KNN 選用的 neighbor 數。



Face classification

16 個不同人的臉部照片，每人各五張，總共 80 張照片，每個人都選一張作為測試，其他四張拿去訓練，所以總共有 64 張照片會被用於訓練模型。模型的準確度如下，橫軸的 n 代表 principle component 的數量，圖例裡面的 k 是 KNN 選用的 neighbor 數。



觀察

- 在性別辨識中，以 PLA 的表現最好，就像在 Task 2 的觀察中所提到的，資料維度越高對 PLA 越有利，KNN 則是在 k 等於 3 的時候最好，不過不管 k 是多少，表現其實都不會差太多。
- 在臉部辨識中，最好的是 KNN 的 k 等於 1 的時候，儘管如此，他的準確度也只有 65% 左右，整體來看不論是哪個 classifier 的表現都不太好，原因可能是訓練資料不夠，雖然照片會被左右翻轉以增加訓練資料，但是加入翻轉過的圖後，每個人也只有八張相片會被用於訓練，樣本還是數太少了。
- 在兩種辨識當中，投影維度 n 很大的時候，Naive Bayes 的表現都很差，我覺得是因為 principle component 取太多了，principle component 是依照資料的變異數去選擇，變異數越大不代表 class 間的差異就會越大，只能說該 attribute 在每筆資料間的變動幅度很大，而 Naive Bayes 是透過計算每個 attribute 的機率分布去辨識資料，過多的 principle component 對於 Naive Bayes 或許也是一種干擾。

```
import cv2
import random
import numpy as np
import matplotlib.pyplot as plt
from abc import ABC, abstractmethod
from numpy.linalg import det, eigh, inv, norm, pinv


class Classifier(ABC):
    @abstractmethod
    def __init__(self, clf=None):
        pass

    @abstractmethod
    def train(self, X, y):
        pass

    @abstractmethod
    def test(self, X):
        pass


class BayesianClassifier(Classifier):
    def __init__(self):
        self.class_num = 0
        self.x_dim = 0
        self.prior = None
        self.mu = None
        self.cov = None
        self.cov_inv = None
        self.cov_det = None

    def train(self, X, y):
        _, cnt = np.unique(y, return_counts=True)
        self.class_num = cnt.shape[0]
        self.x_dim = X.shape[1]
        self.prior = np.zeros(self.class_num, dtype=np.float64)
        self.mu = np.zeros((self.class_num, self.x_dim),
dtype=np.float64)
        self.cov = np.zeros(
            (self.class_num, self.x_dim, self.x_dim),
            dtype=np.float64)
        self.cov_inv = np.zeros(
            (self.class_num, self.x_dim, self.x_dim),
            dtype=np.float64)
        self.cov_det = np.zeros(self.class_num, dtype=np.float64)

        for i, c in enumerate(cnt):
            x = X[y==i]
            self.prior[i] = c / X.shape[0]
```

```python
            self.mu[i] = np.mean(x, axis=0)
            self.cov[i] = np.cov(x.T) + np.eye(self.x_dim) * 1e-2
            self.cov_inv[i] = inv(self.cov[i])
            self.cov_det[i] = det(self.cov[i])

    def test(self, X):
        g = np.zeros((X.shape[0], self.class_num), dtype=np.float64)
        pred = np.zeros(X.shape[0], dtype=np.int32)

        for i in range(self.class_num):
            likelihood = -0.5 * self.x_dim * np.log(2*np.pi)
            likelihood += -0.5 * np.log(self.cov_det[i])
            delta = X - self.mu[i]
            t = delta @ self.cov_inv[i] @ delta.T
            likelihood += -0.5 * np.diagonal(t)
            g[:, i] = likelihood + np.log(self.prior[i])

        g = (g.T / np.abs(np.sum(g, axis=1))).T
        pred = np.argmax(g, axis=1)

        return g, pred


class NaiveBayesClasssifier(Classifier):
    def __init__(self):
        self.class_num = 0
        self.x_dim = 0
        self.prior = None
        self.mu = None
        self.var = None

    def train(self, X, y):
        _, cnt = np.unique(y, return_counts=True)

        self.class_num = cnt.shape[0]
        self.x_dim = X.shape[1]
        self.prior = np.zeros(self.class_num, dtype=np.float64)
        self.mu = np.zeros((self.class_num, self.x_dim),
dtype=np.float64)
        self.var = np.zeros((self.class_num, self.x_dim),
dtype=np.float64)

        for i, c in enumerate(cnt):
            x = X[y==i]
            self.prior[i] = c / X.shape[0]
            self.mu[i] = np.mean(x, axis=0)
            self.var[i] = np.var(x, axis=0) + 1e-2

    def test(self, X):
        g = np.zeros((X.shape[0], self.class_num), dtype=np.float64)
```

```
        pred = np.zeros(X.shape[0], dtype=np.int32)

        for i in range(self.class_num):
            t = -0.5 * np.log(2*np.pi*self.var[i])
            likelihood = np.sum(t)
            t = -0.5 * (X-self.mu[i])**2 / self.var[i]
            likelihood += np.sum(t, axis=1)
            g[:, i] = likelihood + np.log(self.prior[i])

        g = (g.T / np.abs(np.sum(g, axis=1))).T
        pred = np.argmax(g, axis=1)

        return g, pred


class PLA(Classifier):
    def __init__(self, clf=None):
        if clf is None:
            self.class_num = 2
            self.x_dim = 0
            self.w = None
        else:
            self.class_num = clf.class_num
            self.x_dim = clf.x_dim
            self.w = np.copy(clf.w)

    def sign(self, num):
        return 1 if num >= 0 else -1

    def error_rate(self, w, X, y):
        error = 0
        for i in range(X.shape[0]):
            if self.sign(X[i].dot(w)) != y[i]:
                error += 1
        return error / X.shape[0]

    def train(self, X, y):
        _y = np.array(y)
        np.place(_y, _y==0, -1)
        self.x_dim = X.shape[1]
        pocket = np.zeros(self.x_dim, dtype=np.float64)
        w = np.zeros(self.x_dim, dtype=np.float64)
        idx, iteration = 0, 0
        while 1:
            if self.sign(X[idx].dot(w)) != _y[idx]:
                yx = _y[idx] * X[idx]
                w += 0.2 * yx
                if self.error_rate(w, X, _y) < self.error_rate(pocket,
X, _y):
                    pocket = np.copy(w)
```

```
            idx = (idx+1) % X.shape[0]
            iteration += 1
            if iteration>=5000 or self.error_rate(pocket, X, _y)<0.1:
                break
        self.w = np.copy(pocket)

    def test(self, X):
        g = np.zeros((X.shape[0], 1), dtype=np.float64)
        pred = np.zeros(X.shape[0], dtype=np.int32)

        for i, x in enumerate(X):
            g[i, 0] = x.dot(self.w)
            pred[i] = 0 if self.sign(g[i]) < 0 else 1

        return g, pred


def read_bmp(filename):
    image = cv2.imread(filename)
    return image[:, :, 0]


def parse_image_gender(image):
    faces = []
    for i in range(0, image.shape[0], 40):
        for j in range(0, image.shape[1], 40):
            faces.append(image[i:i+40, j:j+40])
    return faces


def parse_image_face(image):
    X_train = []
    X_test = []
    y_train = []
    y_test = []
    for i in range(0, image.shape[1], 40):
        faces = []
        for j in range(0, image.shape[0], 40):
            faces.append(image[j:j+40, i:i+40].ravel())
        test_idx = random.choice(range(5))
        train_idx = list(range(5))
        train_idx.remove(test_idx)
        X_test.append(faces[test_idx])
        y_test.append(i//40)
        for idx in train_idx:
            X_train.append(faces[idx])
            y_train.append(i//40)
    X_train = np.asarray(X_train)
    X_test = np.asarray(X_test)
    y_train = np.asarray(y_train)
```

```
    y_test = np.asarray(y_test)
    return X_train, X_test, y_train, y_test


def load_data(path):
    label = {}
    encode_y = 0
    X, y = [], []
    with open(path, 'r') as file:
        for line in file:
            if not line.strip():
                break
            t = line.strip().split(',')
            if label.get(t[-1]) is None:
                label[t[-1]] = encode_y
                encode_y += 1
            X.append(np.asarray(t[:-1]).astype(np.float64))
            y.append(label[t[-1]])
    X = np.asarray(X)
    y = np.asarray(y)
    return X, y, label


def fld(X, y):
    (n, d) = X.shape
    c = np.unique(y)
    X_i = X[np.where(y==c[0])[0], :]
    mu_i = np.mean(X_i, axis=0)
    S_w = X_i.shape[0] / X.shape[0] * (X_i-mu_i).T @ (X_i-mu_i)
    delta_mu = mu_i
    for i in c[1:]:
        X_i = X[np.where(y==i)[0], :]
        mu_i = np.mean(X_i, axis=0)
        S_w += X_i.shape[0] / X.shape[0] * (X_i-mu_i).T @ (X_i-mu_i)
        delta_mu -= mu_i
    w = pinv(S_w) @ delta_mu
    return w.reshape(-1, 1)


def lda(X, y, dims):
    (n, d) = X.shape
    c = np.unique(y)
    mu = np.mean(X, axis=0)
    S_w = np.zeros((d, d), dtype=np.float64)
    S_b = np.zeros((d, d), dtype=np.float64)
    for i in c:
        X_i = X[np.where(y==i)[0], :]
        mu_i = np.mean(X_i, axis=0)
        S_w += X_i.shape[0] / X.shape[0] * (X_i-mu_i).T @ (X_i-mu_i)
        S_b += X_i.shape[0] / X.shape[0] * (mu_i-mu).T @ (mu_i-mu)
```

```
    eigen_val, eigen_vec = eigh(pinv(S_w)@S_b)
    for i in range(eigen_vec.shape[1]):
        eigen_vec[:, i] = eigen_vec[:, i] / norm(eigen_vec[:, i])
    idx = np.argsort(eigen_val)[::-1]
    w = eigen_vec[:, idx][:, :dims].real
    return w


def pca(X, dims):
    mu = np.mean(X, axis=0)
    cov = np.cov(X.T)
    eigen_val, eigen_vec = eigh(cov)
    for i in range(eigen_vec.shape[1]):
        eigen_vec[:, i] = eigen_vec[:, i] / norm(eigen_vec[:, i])
    idx = np.argsort(eigen_val)[::-1]
    w = eigen_vec[:, idx][:, :dims].real
    return w, mu


def separability_measures(X, y):
    (n, d) = X.shape
    c = np.unique(y)
    mu = np.mean(X, axis=0)
    S_w = np.zeros((d, d), dtype=np.float64)
    S_b = np.zeros((d, d), dtype=np.float64)
    for i in c:
        X_i = X[np.where(y==i)[0], :]
        mu_i = np.mean(X_i, axis=0)
        S_w += X_i.shape[0] / X.shape[0] * (X_i-mu_i).T @ (X_i-mu_i)
        S_b += X_i.shape[0] / X.shape[0] * (mu_i-mu).T @ (mu_i-mu)
    S_m = S_w + S_b
    return np.trace(S_m)/np.trace(S_w)


def get_roc_cm(y, g, thres, opposite):
    cm = np.zeros((2, 2), dtype=np.int32)
    truth = y==0
    for t, score in zip(truth, g):
        # Confusion Matrix: cm
        # ---------------------------------------
        # |   cm[0, 0]: TP    |   cm[0, 1]: FN    |
        # |                   |                   |
        # |   predict true,   |   predict false,  |
        # |   actually true   |   actually true   |
        # ---------------------------------------
        # |   cm[1, 0]: FP    |   cm[1, 1]: TN    |
        # |                   |                   |
        # |   predict true,   |   predict false,  |
        # |   actually false  |   actually false  |
        # ---------------------------------------
```

```
        if opposite:
            cm[int(~t)][int(~(score<thres))] += 1
        else:
            cm[int(~t)][int(~(score>thres))] += 1
    cm_dict = {
        'tp': cm[0, 0], 'fp': cm[1, 0],
        'tn': cm[1, 1], 'fn': cm[0, 1]
    }
    return cm_dict


def get_auc(fpr, tpr):
    x = np.asarray(fpr+[1])
    y = np.asarray(tpr+[0])
    return 0.5 * np.abs(np.dot(x, np.roll(y, -1))-np.dot(x, np.roll(y,
1)))


def get_roc(y, g, opposite=False):
    def get_fpr(cm):
        return cm['fp'] / (cm['fp']+cm['tn']) if (cm['fp']+cm['tn'])
else 0

    def get_tpr(cm):
        return cm['tp'] / (cm['tp']+cm['fn']) if (cm['tp']+cm['fn'])
else 0

    low, high = np.min(g), np.max(g)
    step = (high-low) / 1000
    thres = np.arange(low-2*step, high+2*step, step)
    cms = []
    for t in thres:
        cms.append(get_roc_cm(y, g, t, opposite))
    fpr = list(map(get_fpr, cms))
    tpr = list(map(get_tpr, cms))
    return fpr, tpr, thres


def roc_curve(y, g, filename):
    fpr, tpr, thres = get_roc(y, g[:, 0])
    auc = get_auc(fpr, tpr)
    if auc < 0.5:
        fpr, tpr, thres = get_roc(y, g[:, 0], True)
        auc = get_auc(fpr, tpr)
    plt.plot(fpr, tpr, c='r', lw=2, label=f'ROC (AUC={auc:>.3f})')
    plt.plot([0, 1], [0, 1], 'k--', lw=1, alpha=0.5)
    plt.xlim([-0.01, 1])
    plt.ylim([0, 1.01])
    plt.xlabel('FP / (TN+FP)', fontsize=13)
    plt.ylabel('TP / (TP+FN)', fontsize=13)
```

```python
    plt.legend(loc='lower right')
    plt.gca().set_aspect('equal')
    print(f'AUC: {auc:>.3f}')
    plt.savefig(filename, dpi=300, transparent=True)
    # plt.show()
    plt.clf()


def recognition(X_train, y_train, X_test, y_test, n_component,
two_class):
    print(f'n component: {n_component}')
    w, mu = pca(X_train, n_component)
    X_train_proj = (X_train-mu) @ w
    X_test_proj = (X_test-mu) @ w

    clf = NaiveBayesClasssifier()
    clf.train(X_train_proj, y_train)
    res = clf.test(X_test_proj)
    correct = np.sum((res[1]==y_test).astype(np.int32))
    acc = correct / y_test.shape[0]
    print(f'Naive Bayes accuracy: {acc:.3f}
({correct}/{y_test.shape[0]})')

    if two_class:
        clf = PLA()
        clf.train(X_train_proj, y_train)
        res = clf.test(X_test_proj)
        correct = np.sum((res[1]==y_test).astype(np.int32))
        acc = correct / y_test.shape[0]
        print(f'PLA accuracy: {acc:.3f} ({correct}/{y_test.shape[0]})')

    train_num = X_train_proj.shape[0]
    test_num = X_test_proj.shape[0]
    dist_mat = []
    for i in range(test_num):
        dist = []
        for j in range(train_num):
            d = np.sqrt(np.sum((X_test_proj[i]-X_train_proj[j])**2))
            dist.append((d, y_train[j]))
        dist_mat.append(sorted(dist, key=lambda t: t[0]))

    for k in [1, 3, 5, 7, 9, 11, 13, 15]:
        correct = 0
        for i in range(test_num):
            dist = dist_mat[i]
            neighbor = np.asarray([x[1] for x in dist[:k]])
            neighbor, count = np.unique(neighbor, return_counts=True)
            predict = neighbor[np.argmax(count)]
            if predict == y_test[i]:
                correct += 1
```

```
        print(f'KNN, K={k:>2}, accuracy: {correct/test_num:.3f}
({correct}/{test_num})')


def check_performance(clf, X_train, y_train, X_test, y_test, name,
n_component):
    clf.train(X_train, y_train)
    g, pred = clf.test(X_test)
    correct = np.sum((pred==y_test).astype(np.int32))
    acc = correct / y_test.shape[0]
    filename = f'{name}_roc'
    if n_component:
        filename += f'_PCA{n_component}'
        print(f'{name} with PCA, accuracy: {acc:.3f}
({correct}/{y_test.shape[0]})')
    else:
        print(f'{name}, accuracy: {acc:.3f}
({correct}/{y_test.shape[0]})')
    roc_curve(y_test, g, f'{filename}.png')


def train_test_split(X, y, train_ratio):
    def split_one_class(X, y, label):
        X_class = X[y==label]
        y_class = y[y==label]
        train_size = int(np.ceil(X_class.shape[0] * train_ratio))
        idx = np.arange(X_class.shape[0])
        train_idx_class = np.random.choice(idx, train_size,
replace=False)
        train_idx_class = np.sort(train_idx_class)
        mask = np.ma.array(idx, mask=False)
        mask.mask[train_idx_class] = True
        test_idx_class = mask.compressed()
        X_train = X_class[train_idx_class]
        X_test = X_class[test_idx_class]
        y_train = y_class[train_idx_class]
        y_test = y_class[test_idx_class]
        return X_train, X_test, y_train, y_test

    X_class0_train, X_class0_test, y_class0_train, y_class0_test =
split_one_class(X, y, 0)
    X_class1_train, X_class1_test, y_class1_train, y_class1_test =
split_one_class(X, y, 1)

    X_train = np.vstack((X_class0_train, X_class1_train))
    X_test = np.vstack((X_class0_test, X_class1_test))
    y_train = np.append(y_class0_train, y_class1_train)
    y_test = np.append(y_class0_test, y_class1_test)

    return X_train, X_test, y_train, y_test
```

```
if __name__ == '__main__':
    dataset = './ionosphere/data'
    X, y, label = load_data(dataset)

    # Task 1
    print('Task 1. separability measure')
    res = separability_measures(X, y)
    print(f'origin: {res}')

    w = fld(X, y)
    res = separability_measures(X@w, y)
    print(f'FLD ({X.shape[1]}-dim to 1-dim): {res}')
    g = X @ w
    roc_curve(y, g, 'FLD_roc.png')

    w = lda(X, y, 1)
    res = separability_measures(X@w, y)
    print(f'LDA ({X.shape[1]}-dim to 1-dim): {res}')
    g = X @ w
    roc_curve(y, g, 'LDA_roc.png')

    clf = BayesianClassifier()
    check_performance(clf, X, y, X, y, 'Bayesian all', False)

    clf = NaiveBayesClasssifier()
    check_performance(clf, X, y, X, y, 'Naive Bayes all', False)

    clf = PLA()
    check_performance(clf, X, y, X, y, 'PLA all', False)

    # Task 2
    print('Task 2. PCA and classification')
    X_train, X_test, y_train, y_test = train_test_split(X, y, 0.7)
    clfs = [
        (NaiveBayesClasssifier(), 'Naive Bayes'),
        (BayesianClassifier(), 'Bayesian'),
        (PLA(), 'PLA'),
    ]
    for (clf, name) in clfs:
        check_performance(clf, X_train, y_train, X_test, y_test, name,
None)
        for n_component in [1, 2, 5, 10, 15, 20, 25]:
            print('---------------------')
            print(f'n component: {n_component}')
            w, mu = pca(X_train, n_component)
            X_train_proj = (X_train-mu) @ w
            X_test_proj = (X_test-mu) @ w
            check_performance(clf, X_train_proj, y_train, X_test_proj,
```

```
y_test, name, n_component)
        print('====================')

    # Task 3
    print('Task 3-1. gender classification')
    image_male = read_bmp('mP1.bmp')
    image_female = read_bmp('fP1.bmp')
    faces_male = parse_image_gender(image_male)
    faces_female = parse_image_gender(image_female)
    y = np.zeros(len(faces_male))
    y = np.concatenate((y, np.ones(len(faces_female))))
    y = y.astype(np.int32)
    X = []
    for f in faces_male:
        X.append(f.ravel())
    for f in faces_female:
        X.append(f.ravel())
    X = np.asarray(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, 0.7)
    X_train_flip = np.array(X_train)
    for i in range(X_train_flip.shape[0]):
        X_train_flip[i] = cv2.flip(X_train_flip[0].reshape(40, 40),
1).ravel()
    X_train = np.vstack((X_train, X_train_flip))
    y_train = np.concatenate((y_train, y_train))
    for n_component in [10, 50, 100, 200, 500, 1000]:
        recognition(X_train, y_train, X_test, y_test, n_component, True)
    print()

    print('Task 3-2. face classification')
    image = read_bmp('facesP1.bmp')
    X_train, X_test, y_train, y_test = parse_image_face(image)
    X_train_flip = np.array(X_train)
    for i in range(X_train_flip.shape[0]):
        X_train_flip[i] = cv2.flip(X_train_flip[0].reshape(40, 40),
1).ravel()
    X_train = np.vstack((X_train, X_train_flip))
    y_train = np.concatenate((y_train, y_train))
    for n_component in [10, 50, 100, 200, 500, 1000]:
        recognition(X_train, y_train, X_test, y_test, n_component,
False)
```