

# HW1 Report

0516034 楊翔鈞

作業主要分成兩個部分，第一部分為實作 Bayesian classifier、Naive Bayes classifier 與任選另一個課堂上提過的 linear classifier，我選擇的是 Pocket Algorithm。第二部分為評估模型，用 ROC curve 與 AUC 來比較各模型間的表現。而有關訓練用的 dataset，我從 UCI 上面挑選了五組：

- 2-class Breast Cancer (<https://bit.ly/2wb8MFF>)
- 2-class Ionosphere (<https://bit.ly/2JAxFp>)
- 3-class Iris (<https://bit.ly/39J3ui5>)
- 10-class Optical Recognition of Handwritten Digits (<https://bit.ly/344q83w>)
- 3-class Wine (<https://bit.ly/2JCzcmQ>)

而為了方便程式讀取檔案，我有修改資料的 feature 的順序，讓每行的最後一個值是該筆資料的 label（修改過後的檔案放在 <https://bit.ly/2wPDqVu>）。

## #1 模型

這部分會先介紹上述的三個模型，還有實作過程中遇到的問題與我的解決方法。

### #1.1 Bayesian classifier

我們假設每個 class 本身都是一個 multivariate Gaussian distribution，則計算  $x$  屬於  $i$ -th class 的機率的公式為：

$$p(\omega_i|x) = p(x|\omega_i)p(\omega_i)$$

$p(\omega_i)$  是  $i$ -th class 出現的機率，又叫做 prior； $p(x|\omega_i)$  是  $x$  在  $i$ -th class 這個 Gaussian distribution 出現的機率，又叫做 likelihood，兩者相乘起來就是  $x$  屬於  $i$ -th class 的機率。另外，為了防止相乘後的機率太小，導致 Python 的浮點數無法處理使得機率變 0，所以我把整個機率計算公式取 log，由於 log 是嚴格遞增函數，取 log 後並不會影響數字之間的大小關係。而為了最大化 likelihood，multivariate Gaussian distribution 的 mean 跟 covariance matrix 分別可以透過以下算式求得：

$$\mu = \frac{1}{N} \sum_{k=1}^N x_k$$

$$\Sigma = \frac{1}{N} \sum_{k=1}^N (x - \mu)(x - \mu)^T$$

有了 mean 跟 covariance matrix 後， $x$  屬於  $i$ -th class 的機率（也就是 likelihood）就可以用以下公式算出來，公式中的  $d$  是 feature 的數量：

$$N(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)^T}$$

在實作模型的時候我也發現一些問題，如果 covariance matrix 是 singular 的話，會沒辦法計算  $\frac{1}{\sqrt{(2\pi)^d |\Sigma|}}$ ，而且  $\sqrt{(2\pi)^d |\Sigma|}$  也會因為行列式為 0 而出現問題，所我的解決方法是在 covariance matrix 的所有對角項加上 0.01，就能確保矩陣是 nonsingular。

## #1.2 Naive Bayes classifier

Naive Bayes 與 Bayesian 主要的差異在於 Naive Bayes 忽略變數之間彼此的關係，每個變數都是一個獨立的 Gaussian distribution，所以 covariance 都是 0，機率的計算公式就會變成這樣：

$$N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

而在實作的時候，同樣也會遇到 variance 為 0 的情況，跟 Bayesian 的處理方法類似，我會把所有 variance 都加上 0.01，同時，為了避免機率的乘積過小，這邊一樣把計算結果取 log。

## #1.3 Pocket Algorithm

Perceptron Learning Algorithm 的變形，假如數據不是 linear separable，那我們只能透過一定次數的疊代找到比較好的解。另  $w$  為一個能將兩個類別區盡可能分開的 hyperplane 的法向量，則  $w$  的求法如下：

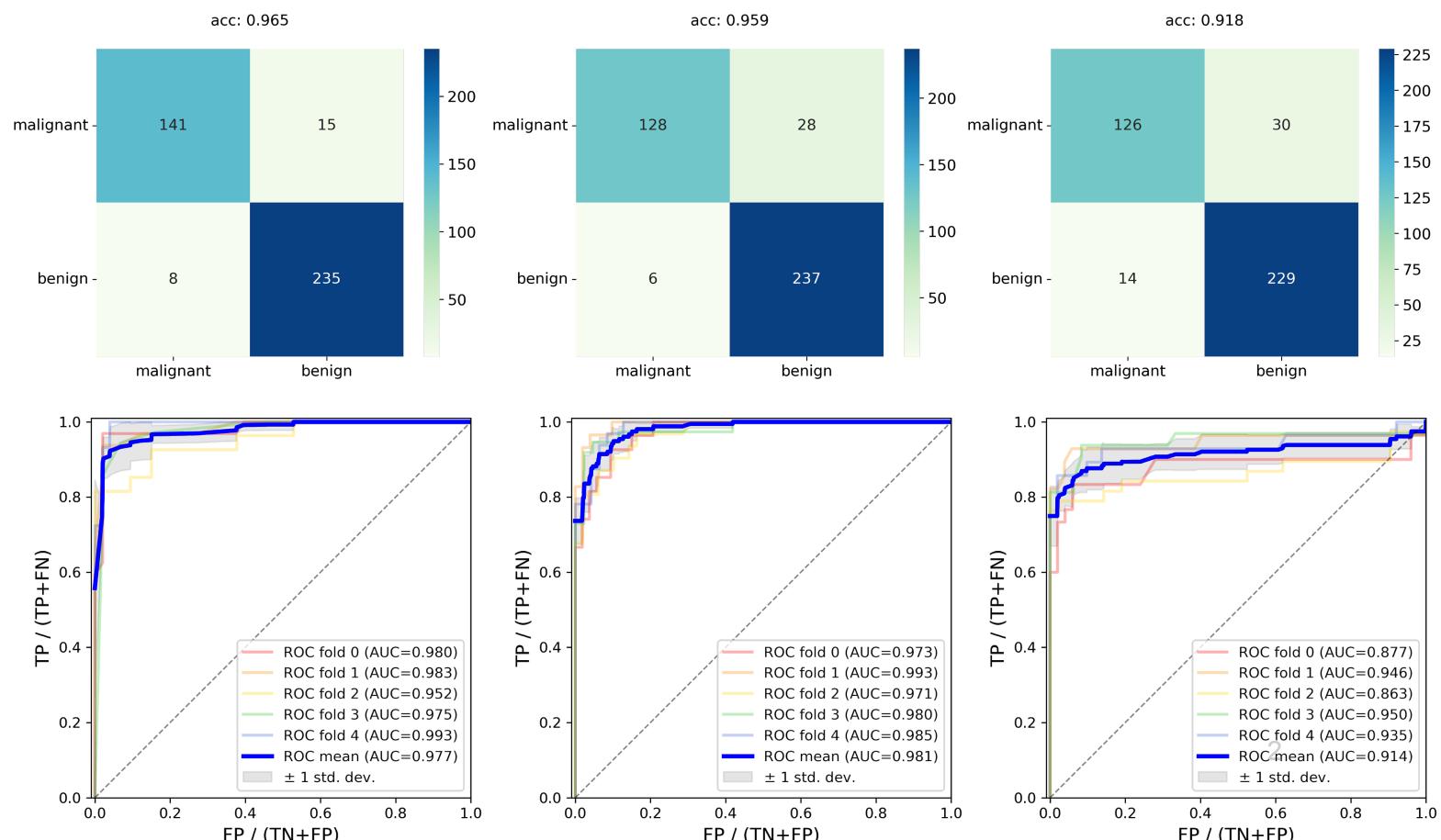
$$w(t+1) = \begin{cases} w(t) + \gamma_t x(t), & \text{for } w(t)^T x(t) \leq 0 \text{ and } x(t) \in \omega_1 \\ w(t) - \gamma_t x(t), & \text{for } w(t)^T x(t) \geq 0 \text{ and } x(t) \in \omega_2 \\ w(t), & \text{otherwise} \end{cases}$$

## #2 模型間的比較

以下的實驗都是先把資料以 7 比 3 的比例分成兩堆，7 等分做訓練、3 等分做測試，並用 5-fold cross validation 來評估模型（confusion matrix）。另外，因為 Breast Cancer 跟 Ionosphere 是二元分類的資料，所以除了 confusion matrix 外還會附上 ROC 曲線，圖片下方會用表格標明各個模型的準確度以及 AUC。

## #2.1 Breast Cancer

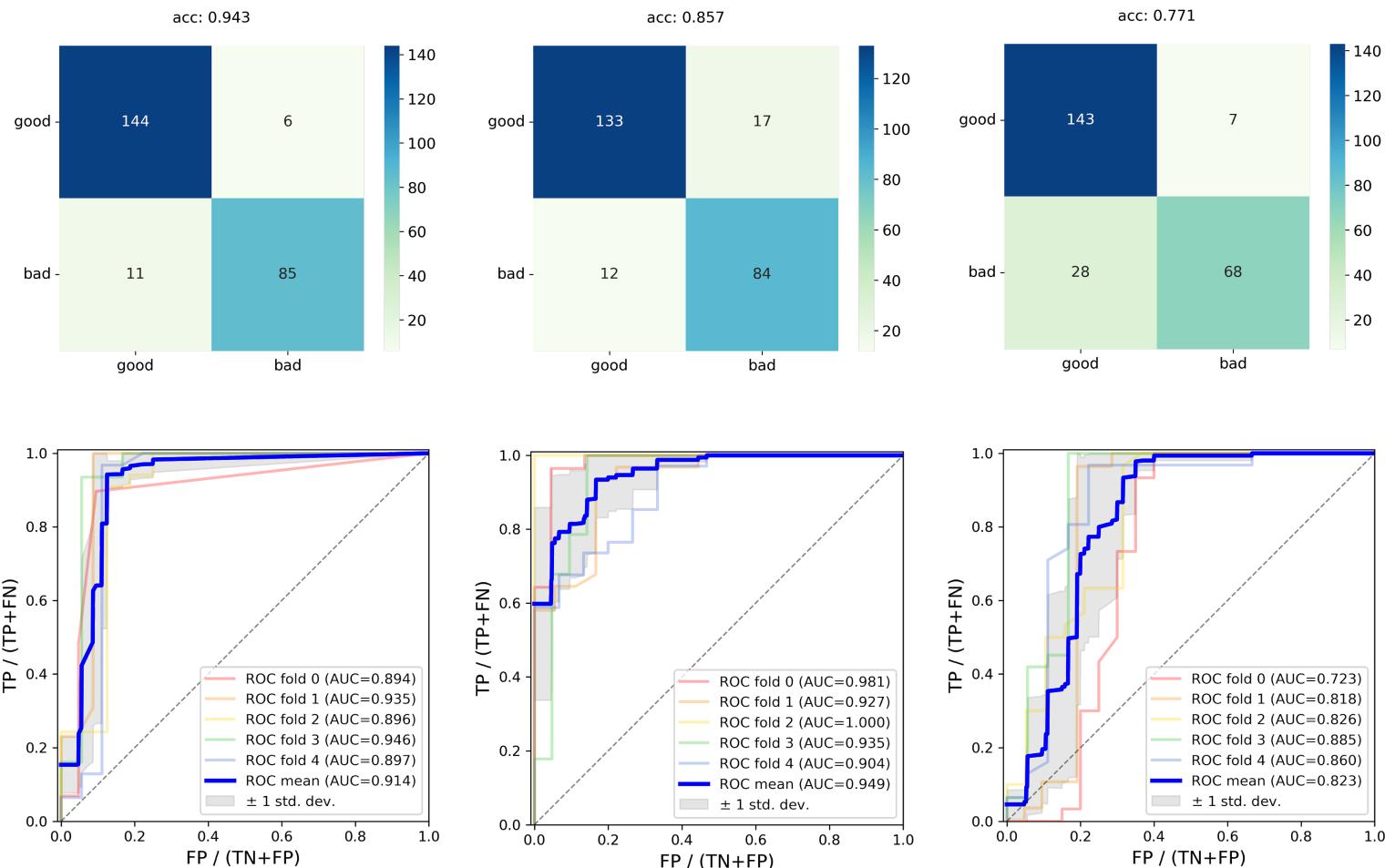
資料有兩個類別，malignant 與 benign，分別代表惡性與良性的乳腺腫瘤。下圖為三個模型的 confusion matrix 以及 ROC 曲線，由左而右為 Bayesian、Naive Bayes、Pocket Algorithm。



Model	Accuracy	AUC
Bayesian	96.5%	0.977
Naive Bayes	95.9%	0.981
Pocket Algorithm	91.8%	0.914

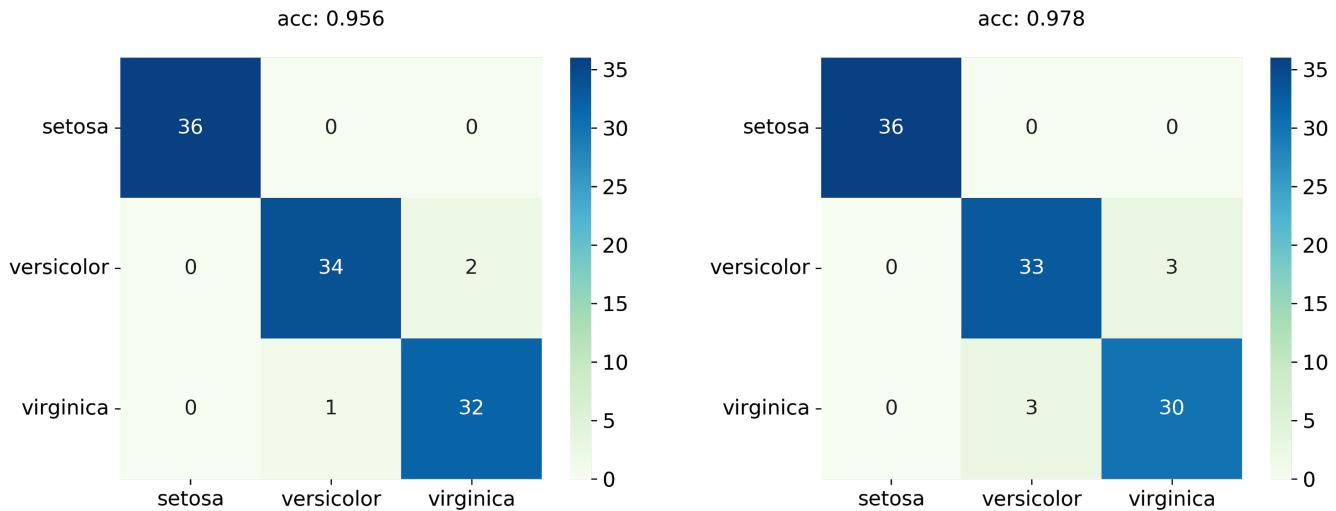
## #2.2 Ionosphere

資料有兩個類別，good 與 bad。下圖為三個模型的 confusion matrix 以及 ROC 曲線，由左而右為 Bayesian、Naive Bayes、Pocket Algorithm。



## #2.3 Iris

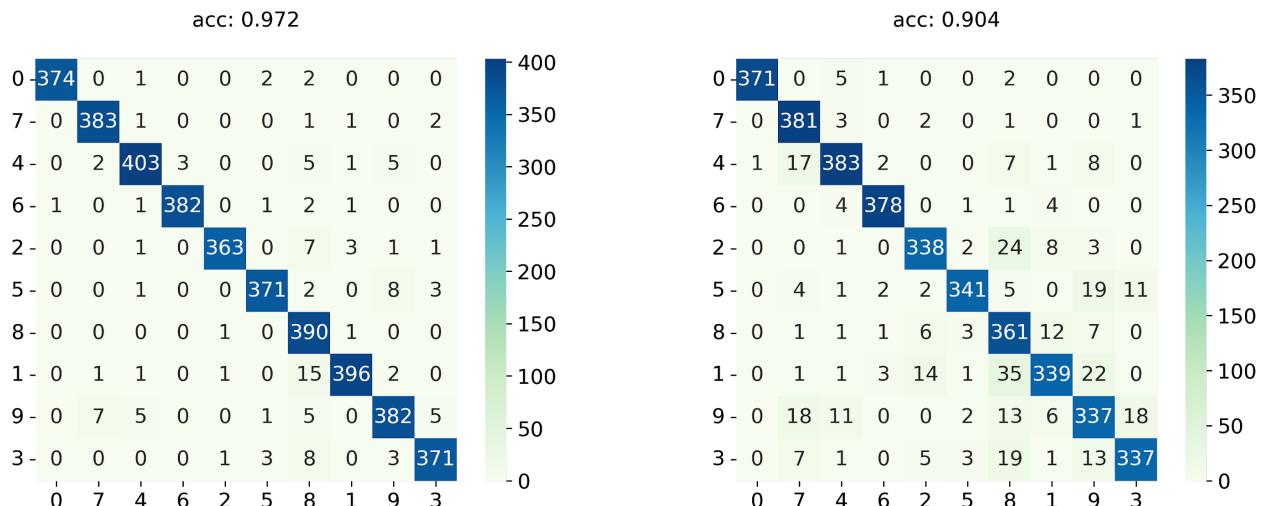
資料有三個類別，Setosa、Versicolor 與 Virginica，分別代表三種不同的鳶尾花。下圖為兩個模型的 confusion matrix，由左而右為 Bayesian、Naive Bayes。



Model	Accuracy
Bayesian	95.6%
Naive Bayes	97.8%

## #2.4 Optical Recognition of Handwritten Digits

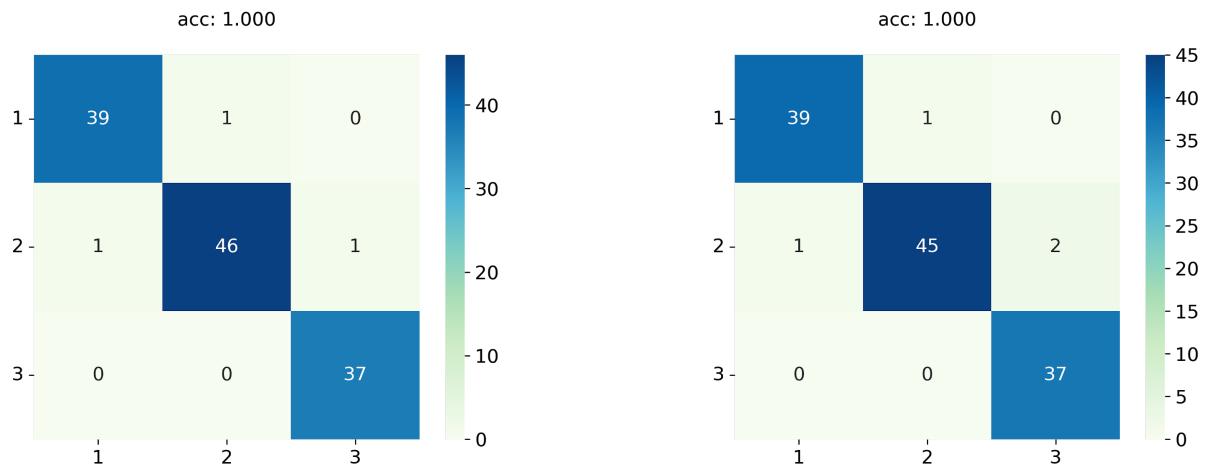
資料有十個類別，分別是 0 到 9 這十個數字。下圖為兩個模型的 confusion matrix，由左而右為 Bayesian、Naive Bayes。



Model	Accuracy
Bayesian	97.2%
Naive Bayes	90.4%

## #2.5 Wine

資料有三個類別，分別對應三種不同的葡萄酒。下圖為兩個模型的 confusion matrix，由左而右為 Bayesian、Naive Bayes。



Model	Accuracy
Bayesian	100%
Naive Bayes	100%

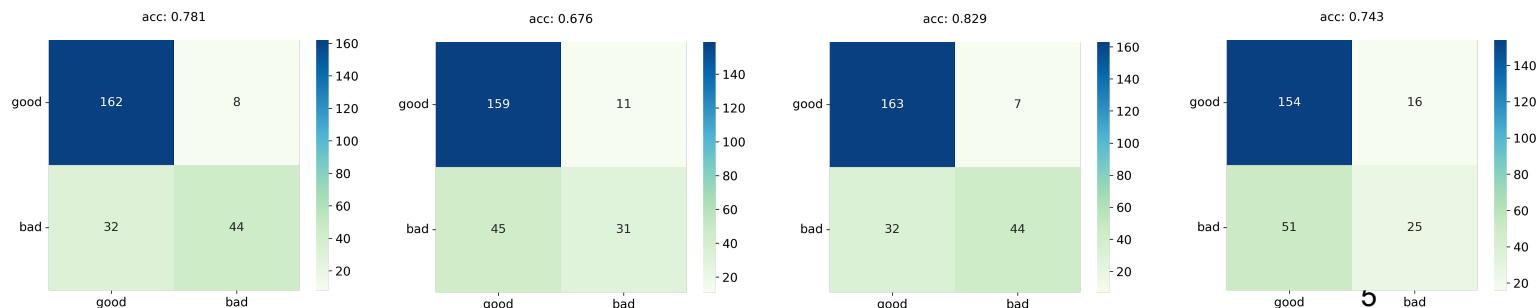
## #2.6 觀察

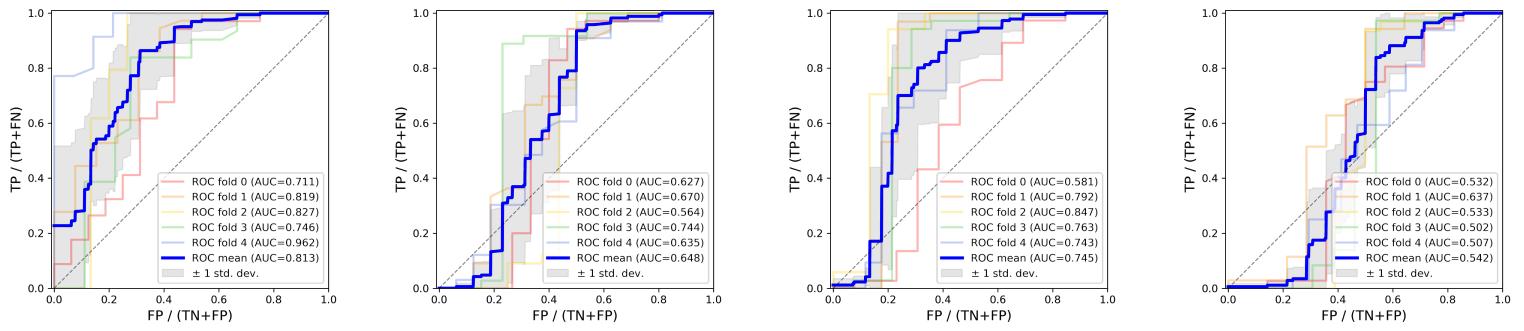
- Bayesian 的準確度比 Naive Bayes 高，因為 Naive Bayes 假設變數之間都是獨立的，並不考慮 covariance，因此在計算上會快一點，但同時可能會忽略變數間的關係；而 Bayesian 不但考慮變數自身的機率分布，也考慮到變數間的關聯性，所以 Bayesian 的準確度會比 Naive Bayes 高一些，尤其當 feature 數量多的時候更明顯。
- Naive Bayes 的 ROC 曲線比 Bayesian 圓滑許多，但 Bayesian 的曲線相較於 Naive Bayes 的更靠近左上角，不過如果只從 AUC 的角度來看的話兩者並沒有明顯的差異。
- Pocket Algorithm 的準確度最低，因為資料通常不是 linear separable，所以只能透過不斷疊代找到一組比較好的解，但因為分類的方法只是單純找出一個 hyperplane，盡可能去把兩個類別區分開，能將訓練資料分開的 hyperplane 不一定能把測試資料分得很準確，所以 Pocket Algorithm 的準確度就比較低。補充一下：我對於疊代的終止條件是訓練資料的準確度大於 0.9 或是疊代次數達到 5000 次。

## #3 改變 feature 數量 (2-class)

之前例子都是將 dataset 的所有 feature 拿去訓練得到的結果，這邊我將嘗試減少 feature 的數量，看看不同數量的 feature 對於模型會有什麼影響，至於要保留哪些 feature 與捨棄哪些，都是由程式隨機選擇的。以下是三個模型對於不同數量的 feature 的訓練結果（以 Ionosphere 為例），一樣是 5-fold cross validation、訓練比測試 7 比 3。

### #3.1 Pocket Algorithm

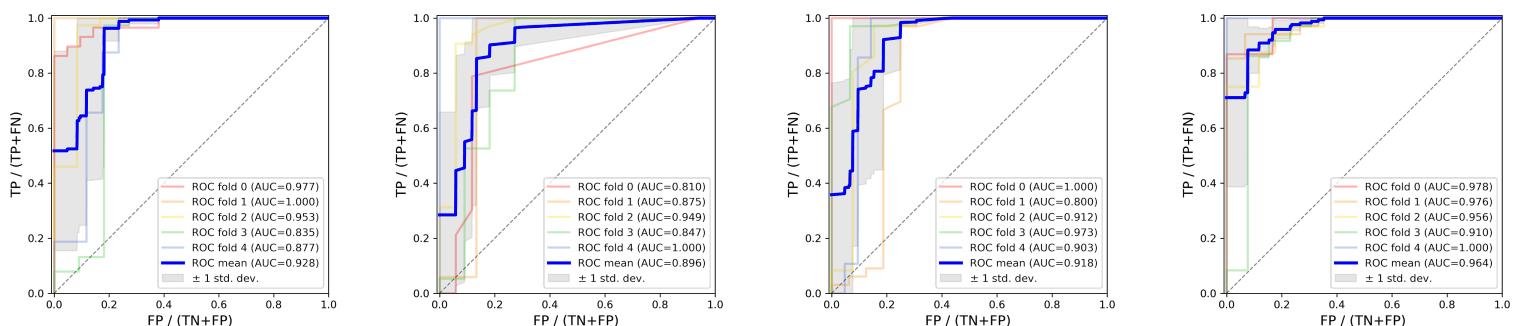
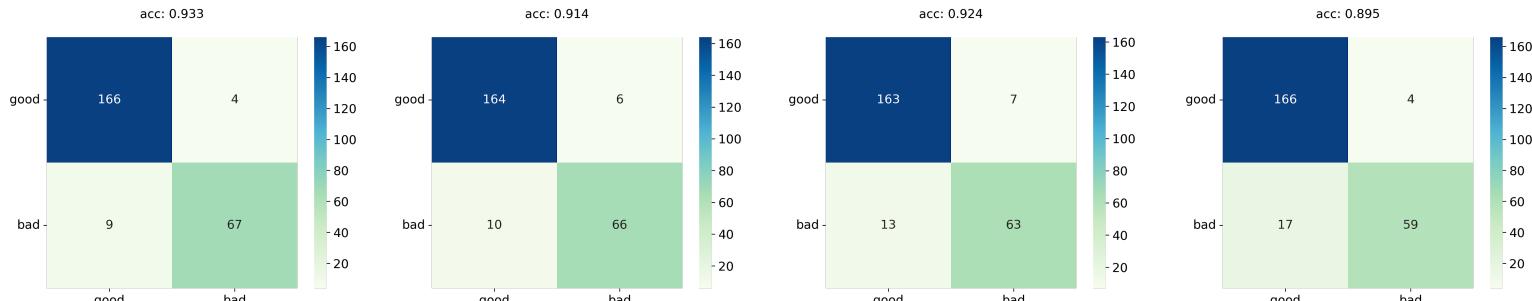




由左而右分別為取 4/4、3/4、2/4、1/4 的 feature 來訓練模型，準確度以及 AUC 如下表：

Proportion of features	Accuracy	AUC
4/4	78.1%	0.813
3/4	67.6%	0.648
2/4	82.9%	0.745
1/4	74.3%	0.542

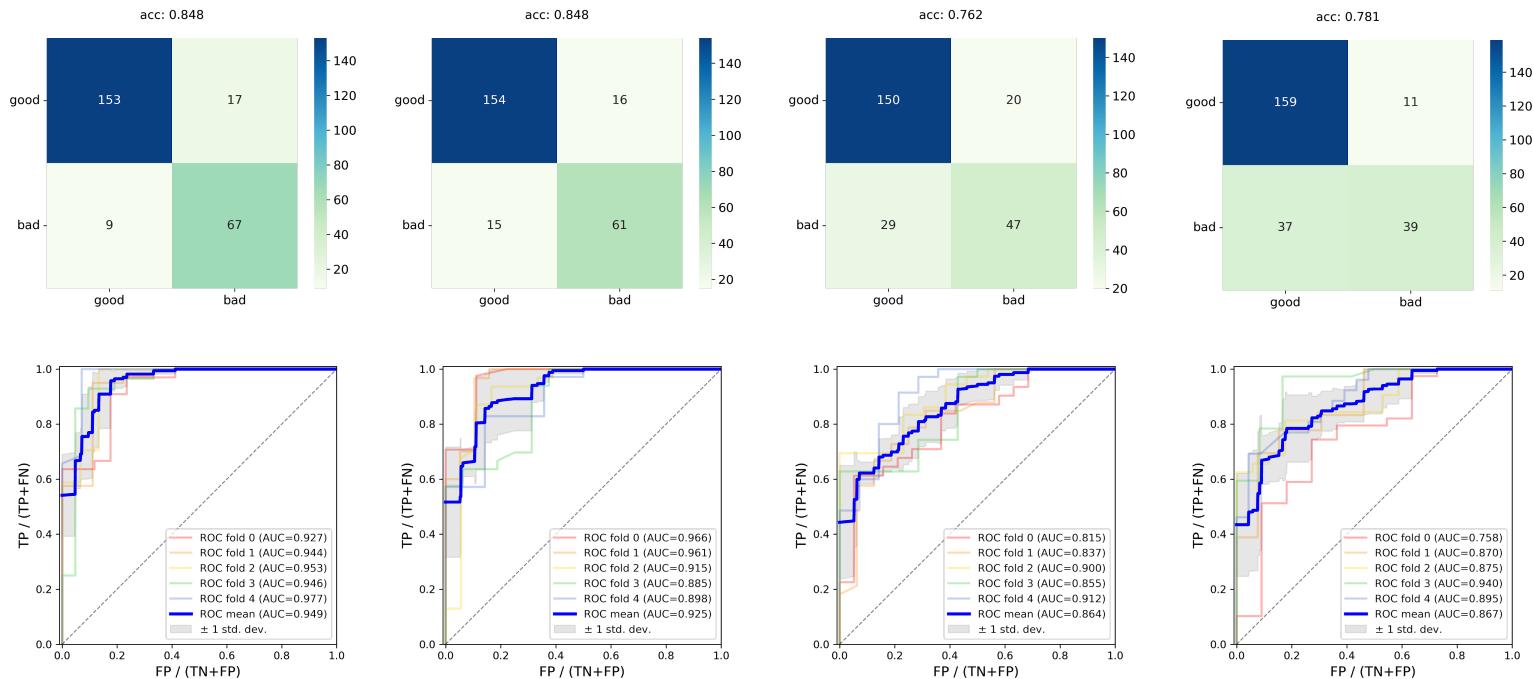
### #3.2 Bayesian



由左而右分別為取 4/4、3/4、2/4、1/4 的 feature 來訓練模型，準確度以及 AUC 如下表：

Proportion of features	Accuracy	AUC
4/4	93.3%	0.928
3/4	91.4%	0.896
2/4	92.4%	0.918
1/4	89.5%	0.964

### #3.3 Naive Bayse



由左而右分別為取 4/4、3/4、2/4、1/4 的 feature 來訓練模型，準確度以及 AUC 如下表：

Proportion of features	Accuracy	AUC
4/4	84.8%	0.949
3/4	84.8%	0.925
2/4	76.2%	0.864
1/4	78.1%	0.867

### #3.4 觀察

- 三種模型以 Pocket Algorithm 的變動幅度最大，隨著 feature 的數量下降，準確度與 AUC 也有下降的趨勢。先前提到 Pocket Algorithm 是在尋找一個盡可能分開兩種類別的 hyperplane，feature 的數量下降就代表資料的維度下降，也就更難找到這樣的 hyperplane（高維度的 hyperplane 比低維度的更靈活，少了一個維度就等於少了一個自由度）。
- Bayesian 的變動幅度比 Naive Bayes 小，我認為這跟 covariance 有關係，因為 Bayesian 在計算機率的時候會考慮變數間的關聯，但 Naive Bayes 只會參考變數自身的 variance，如下圖所示：



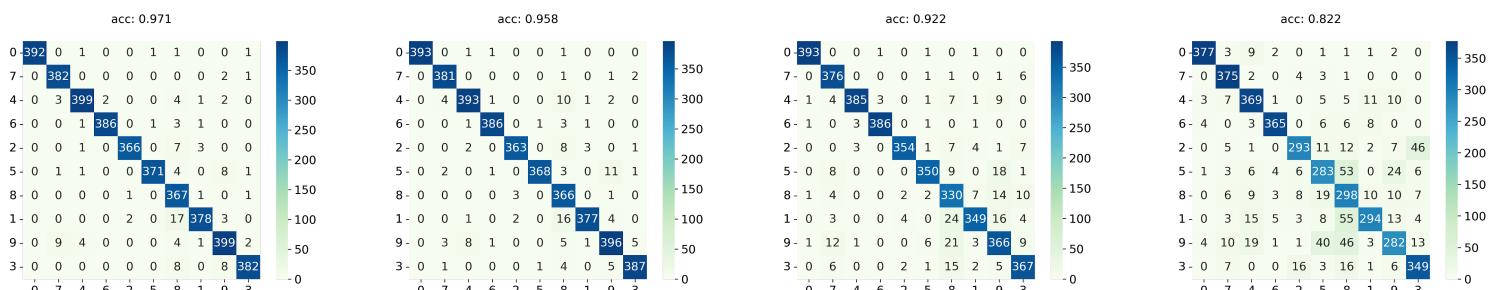
Bayesian 是左邊藍色那組，Naive Bayes 是右邊那組，紅點為 variance，藍點為 covariance，灰點為 Naive Bayes 不採用的數據。可以看到當 feature 從 7 個減少為 5 個時，Bayesian 能用的數據還是很多，Naive Bayes 却從 7 個變 5 個，所以減少 feature 對 Naive Bayes 的影響也就比較大。

- 還有一點我目前比較不清楚的，就是隨著 feature 數量減少，Bayesian 的 AUC 不減反增，我一開始以為只是剛好跑出這樣的結果，但後來重複了幾次實驗，發現只用 1/4 的 feature 跑模型的 AUC 會比其他情況略高一點。

## #4 改變 feature 數量 (multi-class)

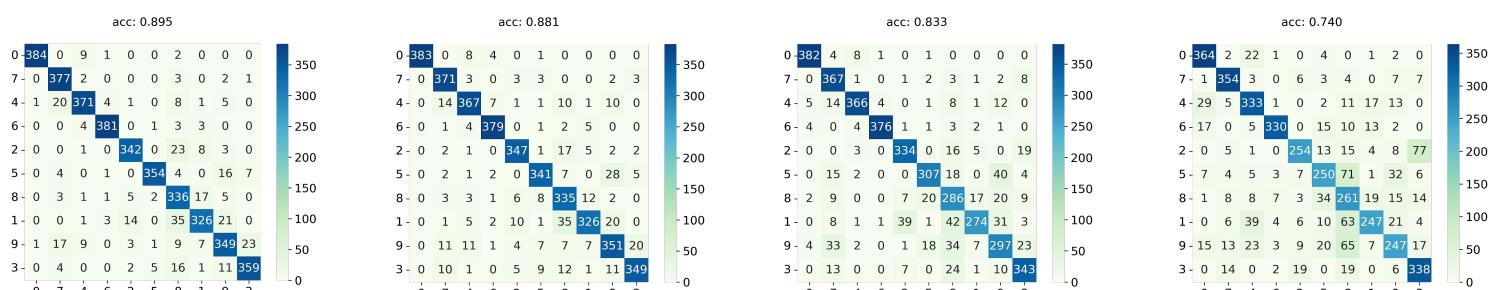
上面舉了 2-class 的例子，現在換成 multi-class 的 dataset，比較 Bayesian 跟 Naive Bayes 之間的差異（這邊以 Optical Recognition of Handwritten Digits 這個 dataset 為例），cross validation 一樣維持 5-fold，訓練比測試一樣是 5 比 3。

### #4.1 Bayesian



Proportion of features	Accuracy
4/4	97.1%
3/4	95.8%
2/4	92.2%
1/4	82.2%

### #4.2 Naive Bayes



Proportion of features	Accuracy
4/4	89.5%
3/4	88.1%
2/4	83.3%
1/4	74%

### #4.3 觀察

- 如同上一個環節提到的，因為 Naive Bayes 不考慮 covariance，當 feature 變少時用於訓練的數據就更少了，準確度直接掉到 74%，Bayesian 則還維持在 80% 以上。

### #5 改變訓練資料的大小

我後來也嘗試用 10%、20%、30% 一直到 90% 的資料去做訓練，跑出來的結果如下表（這邊是拿 Optical Recognition of Handwritten Digits 做試驗）：

	Accuracy of Bayesian	Accuracy of Naive Bayes
90%	97.5%	90%
80%	96.4%	90.6%
70%	96.8%	91.2%
60%	97%	90.8%
50%	96.4%	90%
40%	96.2%	89.9%
30%	95.3%	90.2%
20%	94.7%	89.8%
10%	72.6%	88%

本以為 10% 的那組表現會很差，但得到的結果卻還算能接受，而且 Bayesian 在 20% 以上都表現得不錯，我覺得一部份原因可能是每次取的訓練資料中，各類別的比例都很相近，所以模型可以有足夠的樣本去計算差異並區分不同類別，所以準確度才沒有太大的差異。

```

import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from numpy.linalg import inv, det
from abc import ABC, abstractmethod

title_size = 13
text_size = 13
annot_size = 13

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, clf=None):
        if clf is None:
            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
        else:
            self.class_num = clf.class_num
            self.x_dim = clf.x_dim
            self.prior = np.copy(clf.prior)
            self.mu = np.copy(clf.mu)
            self.cov = np.copy(clf.cov)
            self.cov_inv = np.copy(clf.cov_inv)
            self.cov_det = np.copy(clf.cov_det)

    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]
        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 NaiveBayesClassifier(Classifier):
    def __init__(self, clf=None):
        if clf is None:
            self.class_num = 0
            self.x_dim = 0
            self.prior = None
            self.mu = None
            self.var = None
        else:
            self.class_num = clf.class_num

```

```

        self.x_dim = clf.x_dim
        self.prior = np.copy(clf.prior)
        self.mu = np.copy(clf.mu)
        self.var = np.copy(clf.var)

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

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

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

```

```

        return error / X.shape[0]

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 train(self, X, y):
        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 sign(X[idx].dot(w)) != y[idx]:
                yx = y[idx] * X[idx]
                w += 0.2 * yx
                if error_rate(w, X, y) < error_rate(pocket, X, y):
                    pocket = np.copy(w)
            idx = (idx+1) % X.shape[0]
            iteration += 1
            if iteration>=5000 or 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] = sign(g[i])

        return g, pred

def load_data(path, is_PLA):
    start_idx = {True: -1, False: 0}
    step = {True: 2, False: 1}

    label = {}
    encode_y = start_idx[is_PLA]

```

```

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 += step[is_PLA]

        X.append(np.asarray(t[:-1]).astype(np.float64))
        y.append(label[t[-1]])

X = np.asarray(X)
if is_PLA:
    X = np.hstack((np.ones((X.shape[0], 1)), X))
y = np.asarray(y)

return X, y, label

def train_test_split(X, y, train_ratio):
    train_size = int(np.ceil(X.shape[0] * train_ratio))
    idx = np.arange(X.shape[0])
    train_idx = np.random.choice(idx, train_size, replace=False)
    train_idx = np.sort(train_idx)

    mask = np.ma.array(idx, mask=False)
    mask.mask[train_idx] = True
    test_idx = mask.compressed()

    return X[train_idx], X[test_idx], y[train_idx], y[test_idx]

def get_cm(y, pred, class_num):
    cm = np.zeros((class_num, class_num), dtype=np.int32)
    for i, j in zip(y, pred):
        cm[i, j] += 1
    return cm

def get_roc_cm(y, g, thres, is_PLA):
    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 is_PLA:
    cm[int(~t)][int(~(scoreelse:
    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_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

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, is_PLA):
    low, high = np.min(g), np.max(g)
    step = (abs(low) + abs(high)) / 1000
    thres = np.arange(low-2*step, high+2*step, step)
    if is_PLA:
        thres = thres[::-1]

    cms = []
    for t in thres:
        cms.append(get_roc_cm(y, g, t, is_PLA))

    fpr = list(map(get_fpr, cms))

```

```

tpr = list(map(get_tpr, cms))

return fpr, tpr, thres

def evaluation(y, pred, g):
    new_y = np.copy(y)
    new_pred = np.copy(pred)
    if g.shape[1] == 1:
        np.place(new_y, new_y===-1, 0)
        np.place(new_pred, new_pred===-1, 0)
    class_num = max(g.shape[1], 2)
    cm = get_cm(new_y, new_pred, class_num)
    if g.shape[1] <= 2:
        roc = get_roc(new_y, g[:, 0], g.shape[1]==1)
    else:
        roc = None

    return cm, roc

def k_fold(clf, X_train, y_train, X_test, y_test, label, k):
    print(label)

    data_idx = np.arange(X_train.shape[0])
    np.random.shuffle(data_idx)
    fold = np.asarray(np.array_split(data_idx, k))
    k_idx = np.arange(k)

    class_num = len(label.keys())
    cms = np.zeros((class_num, class_num), dtype=np.int32)
    rocs = []
    color = ['r', 'darkorange', 'gold', 'limegreen', 'royalblue',
             'blueviolet']

    for i, f in enumerate(fold):
        train_idx = np.concatenate(fold[k_idx[k_idx!=i]])
        test_idx = np.concatenate(fold[k_idx[k_idx==i]])
        X_train_sub, y_train_sub = X_train[train_idx],
        y_train[train_idx]
        X_test_sub, y_test_sub = X_train[test_idx], y_train[test_idx]
        clf.train(X_train_sub, y_train_sub)
        g, pred = clf.test(X_test_sub)

        acc = (y_test_sub==pred).astype(np.int32)
        acc = np.sum(acc) / y_test_sub.shape[0]
        print(f'accuracy fold {i}: {acc:>.3f}')

```

```

cm, roc = evaluation(y_test_sub, pred, g)
cms += cm
rocs.append(roc)

clf.train(X_train, y_train)
g, pred = clf.test(X_test)

test_acc = (y_test==pred).astype(np.int32)
test_acc = np.sum(test_acc) / y_test.shape[0]
print(f'accuracy test: {test_acc:.3f}')

ax = sn.heatmap(
    cms, annot=True, annot_kws={'size': annot_size}, cmap='GnBu',
    square=True, fmt='g', cbar_kws={'format': '%d'})

cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=annot_size)
plt.gca().set_xticklabels(label.keys(), fontsize=text_size)
plt.gca().set_yticklabels(
    label.keys(), va='center', rotation=0, fontsize=text_size)

plt.title(f'acc: {test_acc:.3f}\n', fontsize=title_size)
plt.savefig('cm.png', dpi=300, transparent=True)
plt.clf()

if class_num == 2:
    mean_fpr = np.linspace(0.0, 1.0, 1000)
    tprs = []
    avg_auc = 0
    for i, roc in enumerate(rocs):
        fpr, tpr, thres = roc
        tprs.append(np.interp(mean_fpr, fpr[::-1], tpr[::-1]))
        auc = get_auc(fpr, tpr)
        avg_auc += auc
        plt.plot(
            fpr, tpr, c=color[i%len(color)], lw=2, alpha=0.3,
            label=f'ROC fold {i} (AUC={auc:.3f})')

    mean_tpr = np.mean(tprs, axis=0)
    plt.plot(mean_fpr, mean_tpr, c='b', lw=3, alpha=1.0, label=f'ROC
mean (AUC={avg_auc/k:.3f})')

    std_tpr = np.std(tprs, axis=0)
    tprs_upper = np.minimum(mean_tpr+std_tpr, 1)
    tprs_lower = np.maximum(mean_tpr-std_tpr, 0)
    plt.fill_between(
        mean_fpr, tprs_lower, tprs_upper, color='grey',
        alpha=0.2, label=r'$\pm$ 1 std. dev.')

```

```

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=text_size)
plt.ylabel('TP / (TP+FN)', fontsize=text_size)
plt.legend(loc='lower right')
plt.gca().set_aspect('equal')
plt.savefig('roc.png', dpi=300, transparent=True)
plt.clf()

if __name__ == '__main__':
    dataset = 'ionosphere'

X, y, label = load_data(f'./{dataset}/data', False)
X_train, X_test, y_train, y_test = train_test_split(X, y, 0.7)

clf = BayesianClassifier()
k_fold(clf, X_train, y_train, X_test, y_test, label, 5)

```