HW7 REPORT

- PCA project

The PCA class

```
4  class PCA:
5      def __init__(self,dataArray,classArray=None):
6          #flattens array to 2-D
7          if dataArray.ndim>2:
8              dataArray=dataArray.reshape(dataArray.shape[0],-1)
9          self.dataArray=dataArray
10         self.classArray = classArray
11         self.dataSize, self.dataDimension = self.dataArray.shape
```

__init__:

將資料壓成 [資料比數 , feature 數]，正確的 class 可有可無，如果有給的話最後的圖會上色。定義一些基本的變數。

```
12     def project(self,projectDimension=2):
13         #project process here
14         self.projectDimension=projectDimension
15
16         print("Calculating covariance matrix...")
17         print(self.dataArray.shape)
18         covarArray=np.cov(self.dataArray,rowvar=False)
19         #covarArray=np.corrcoef(self.dataArray,rowvar=False)
20
21         print("Solving Eigen Values...")
22         self.eigenValue, self.eigenVector = lng.eig(covarArray)
23         #reverse argsort() indexes (was sorted in ascending)
24         sortIndices=self.eigenValue.argsort()[::-1][:]
25         self.eigenValue=self.eigenValue[sortIndices]
26         self.eigenVector = self.eigenVector[:, sortIndices]
27
28         print("Projecting data...")
29         #select eigenvectors with largest eigen values
30         self.projectEigenVector=self.eigenVector[:,:self.projectDimension]
31         self.projectData=self.dataArray@self.projectEigenVector
32         print("Finished projecting!")
```

project():

將資料投射到 projectDimension 的維度空間。covariance 因為自己用迴圈跑很慢因此借用 np.cov。算 eigenvalues 用的是 numpy 的 linalg。argsort()回傳的是 ascending 的 index order，我要取最大值所以將其反向。Project 就是矩陣乘法乘剛剛的 eigenvectors，這邊因為方向的關係可能跟常見公式不同但算法是一樣的。

```
34      def plot(self):
35          #plt figure
36          fig = plt.figure()
37          ax = fig.add_subplot(111)
38          #draw corresponding first two eigenvectors values..
39          dataX=self.projectData[:,0]
40          dataY=self.projectData[:,1]
41          scatter = ax.scatter(dataX, dataY, c=self.classArray, s=20)
42          ax.set_xlabel('x')
43          ax.set_ylabel('y')
44          #classArray can be empty (it's unsupervised after all)
45          if self.classArray is not None:
46              plt.colorbar(scatter)
47          plt.show()
```

plot():

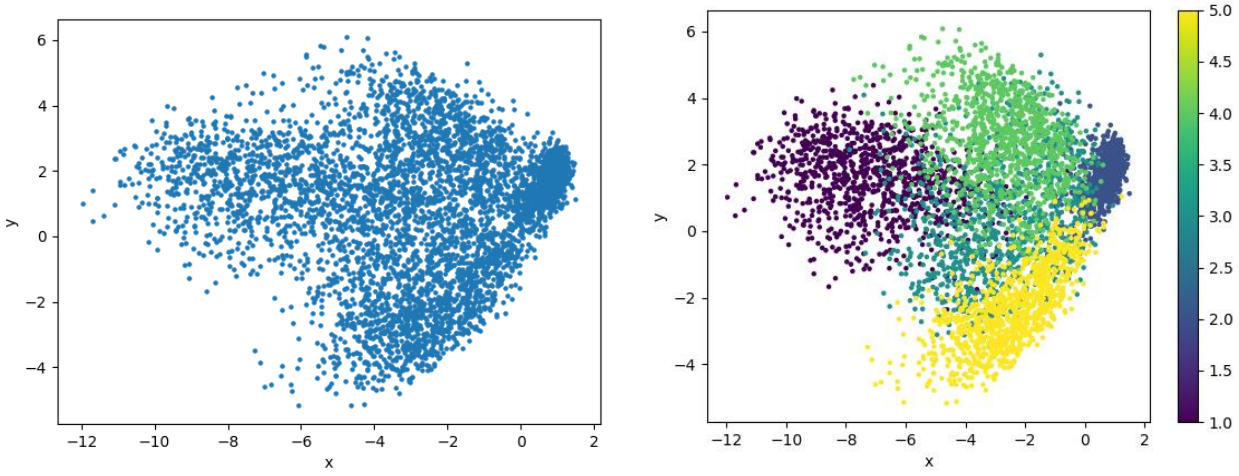
使用 matplotlib 做圖，只取前兩個值作圖，因此若 projectDimension>2 會沒辦法呈現後面的值。如果有給各個資料的 class(label)的話會上色。

PCA PROJECT

```
48  if __name__=="__main__":
49      from dataprocess import readData
50      trainData=readData("X_train.csv")
51      dataLabel=readData("T_train.csv")
52      pcaTest=PCA(trainData)
53      pcaTest.project()
54      pcaTest.plot()
```

```
5   def readData(name,delimiter=','):
6       #load csv file to numpy array
7       print("Reading data {}".format(name))
8       data=np.loadtxt(name,delimiter=delimiter)
9       if data.ndim==2:
10          return data[:,:]
11      elif data.ndim==1:
12          return data[:]
```

讀檔後丟進去。



右邊的上色是根據正確的 class 去做的，可以觀察到雖然有大面積的重疊，但是可以大致上看出各個數字有各自的區塊，將 784 壓至 2 有這樣的結果已經算是不錯

了。如果用一點想像去看，可以觀察到 235 這幾個比較像的數字的交疊在一起，

而 4 這個與其他的完全不像的數字則有很小的 variance。

- LDA Project

The LDA class

```
class LDA:
    def __init__(self,dataArray,classArray):
        self.dataArray=dataArray
        self.classArray = classArray
        self.realClasses,self.classDataCount=np.unique(self.classArray, return_counts=True)
        self.totalClassCount=self.realClasses.size
        self.classDict=dict(zip(self.realClasses,range(self.totalClassCount)))
        self.dataSize, self.dataDimension = self.dataArray.shape
```

__init__():

LDA 是 supervised learning，所以 data 跟 class 都是必要的。np.unique 是為了找出

class 的種類，classDict 則給在 array 中他對照的 index。

```
    def project(self,projectDimension=2):
        self.projectDimension=projectDimension
        self.inClassScatter=np.zeros((self.dataDimension,self.dataDimension))
        self.betweenClassScatter=np.zeros((self.dataDimension, self.dataDimension))
        #calculate class means
        print("Calculating in class scatter...")
        inClassMean=np.zeros((self.dataDimension,self.totalClassCount))
        for dataIndex in range(self.dataSize):
            inClassMean[:,self.classDict[self.classArray[dataIndex]]]+=self.dataArray[dataIndex,:]
        for classIndex in range(self.totalClassCount):
            inClassMean[:,classIndex]/=self.classDataCount[classIndex]
        #calculate inclass scatter (sum over in class)
        for dataIndex in range(self.dataSize):
            diffToClassMean=(self.dataArray[dataIndex, :] -
                             inClassMean[:, self.classDict[self.classArray[dataIndex]]])[:,np.newaxis]
            self.inClassScatter+=(diffToClassMean@np.transpose(diffToClassMean))

        #calculate between class scatter
        print("Calculate between class scatter..")
        allMean=np.average(inClassMean,axis=1,weights=self.classDataCount)[:,np.newaxis]
        for classIndex in range(self.totalClassCount):
            diffToTotalMean=inClassMean[:,classIndex]-allMean
            self.betweenClassScatter=self.classDataCount[classIndex]*(diffToTotalMean@np.transpose(diffToTotalMean))

        #calculate eigen vectors
        print("Calculate eigens...")
        inverseSwSb=lng.pinv(self.inClassScatter)@self.betweenClassScatter
        self.eigenValue, self.eigenVector = lng.eig(inverseSwSb)
        sortIndices = self.eigenValue.argsort()[::-1][:self.projectDimension]
        self.eigenValue = self.eigenValue[sortIndices]
        self.eigenVector = self.eigenVector[:, sortIndices]
        print(self.eigenVector)
        self.projectData = self.dataArray @ self.eigenVector
```

project():

這整塊簡單說就是 算 inclass scatter -> between class scatter -> eigenvectors->project

inclass mean 是指投影片中的 mj，接著將各個點和對應的 class mean 相減後加總

就是 in class scatter。Self.classDict[]就是為了對應實際 class 跟存在 array 中的位置

而已。 有些行數有加上[:,np.newaxis]，這是幫 1D 加上一個軸，這樣才能做矩陣乘法@。

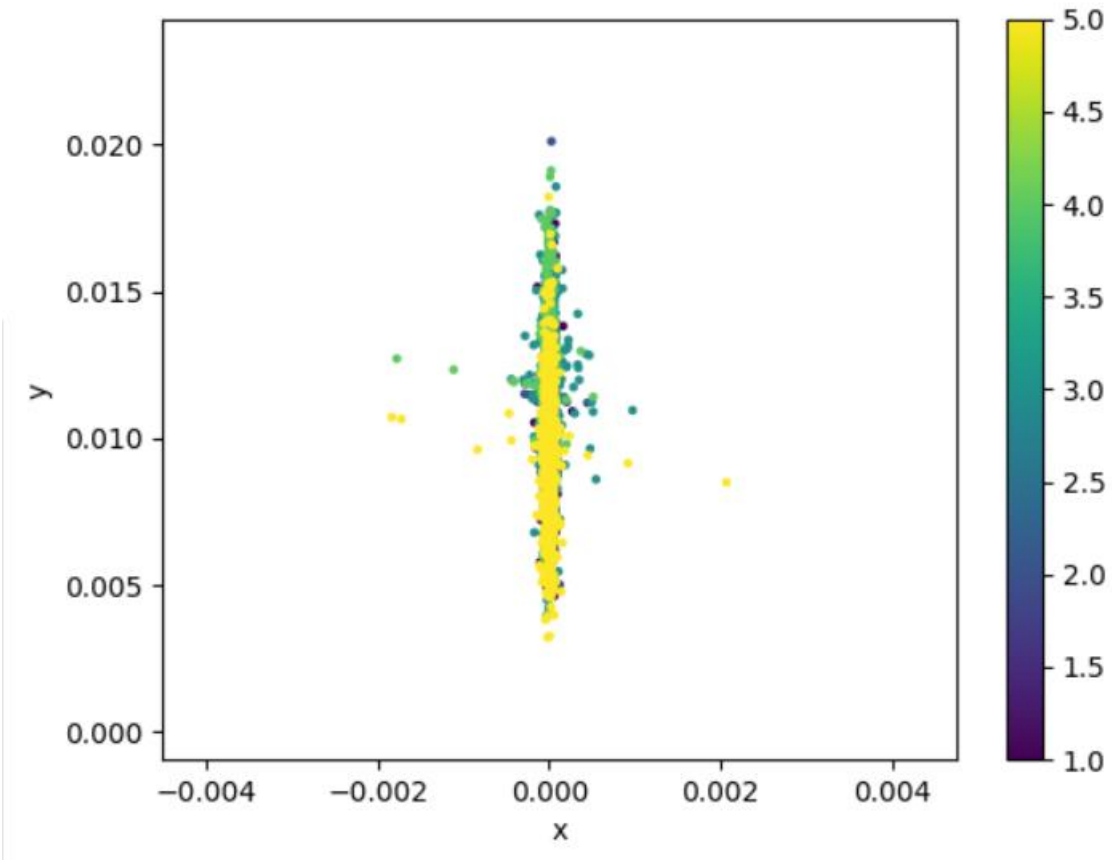
再來就是算出投射用的 eigenvector，是指(inclass scatter)^-1*(between class scatter)，在此就不推導。因為 inclass scatter 有可能沒有 inverse，因此使用 pinv(pseudo inverse)。最後取 eigenvalue 最大的 eigenvector 做 project。

```
46      def plot(self):
47          fig = plt.figure()
48          ax = fig.add_subplot(111)
49          dataX=self.projectData[:,0]
50          dataY=self.projectData[:,1]
51          scatter = ax.scatter(dataX, dataY, c=self.classArray, s=5)
52          ax.set_xlabel('x')
53          ax.set_ylabel('y')
54          plt.colorbar(scatter)
55          plt.show()
```

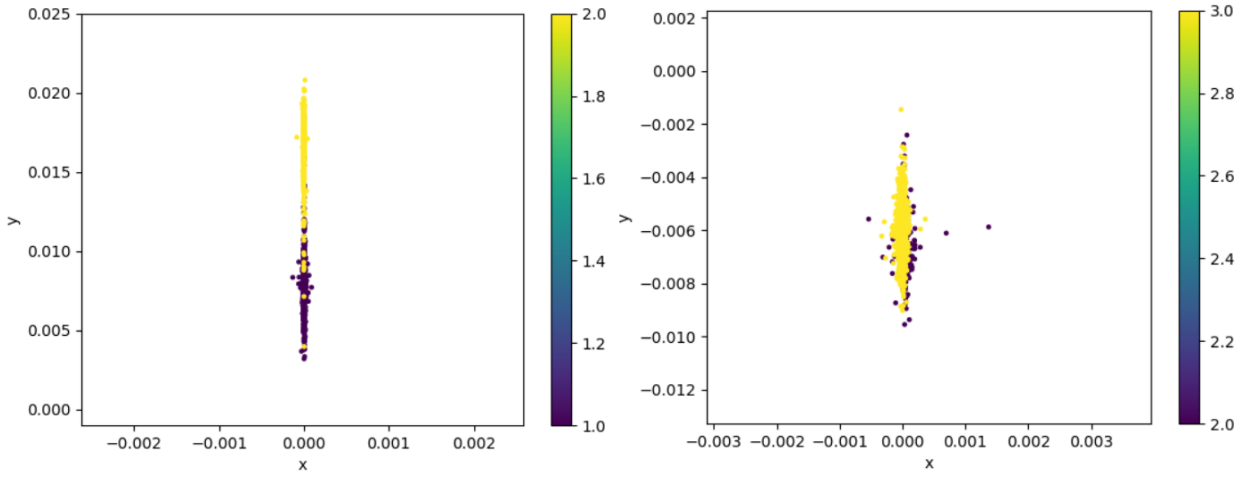
和 PCA 一樣的 plot。

LDA PROJECT

```
57  if __name__=="__main__":
58      from dataprocess import readData
59      trainData=readData("X_train.csv")
60      dataLabel=readData("T_train.csv")
61      ldaTest=LDA(trainData,dataLabel)
62      ldaTest.project()
63      ldaTest.plot()
```



這邊的 LDA 分布會都重疊，乍看之下很失敗，但若一次只用兩個 class 可以發現事實上是有的。



左圖是只用 1,2 class，可以觀察到有明顯的分開，而右圖 2,3 class 在 PCA 中可以看到本來就相似，因此幾乎無法分開，這算是預期的結果。

- Eigen face

```python
def plotSampleFace(imageArray,imageShape=None,sampleShape=(5,5),randomSelect=True,imageIndex=None,scaling=False):
    sampleHeight,sampleWidth=sampleShape
    if imageArray.ndim==2:
        imageHeight,imageWidth=imageShape
        imageCount=imageArray[0]
    elif imageArray.ndim==3:
        imageCount,imageHeight,imageWidth=imageArray.shape
    else:
        print("Illegal image array")
        return
    mergedFaces=np.zeros((sampleHeight*imageHeight,sampleWidth*imageWidth))
    if randomSelect:
        selectedImageIndex=np.random.randint(imageCount,size=sampleShape)
    else:
        if imageIndex is None:
            selectedImageIndex=np.arange(sampleHeight*sampleWidth).reshape(sampleShape)
        else:
            selectedImageIndex=imageIndex

    if scaling is True:
        for imageIndex in selectedImageIndex:
            if imageArray.ndim==2:
                imageArray[imageIndex, :] *= 255.0/imageArray[imageIndex, :].max()
            elif imageArray.ndim==3:
                imageArray[imageIndex, :,:] *= 255.0 / imageArray[imageIndex, :,:].max()
    for sampleRow in range(sampleHeight):
        if imageArray.ndim==2:
            mergedImageRow = mergeImageToRow(imageArray[selectedImageIndex[sampleRow, :],:], (imageHeight, imageWidth))
        elif imageArray.ndim==3:
            mergedImageRow=mergeImageToRow(imageArray[selectedImageIndex[sampleRow,:],:,:],(imageHeight,imageWidth))
        mergedFaces[sampleRow*imageHeight:(sampleRow+1)*imageHeight,:]=mergedImageRow

    plt.imshow(mergedFaces)
    plt.colorbar()
```

plotSampleFace():

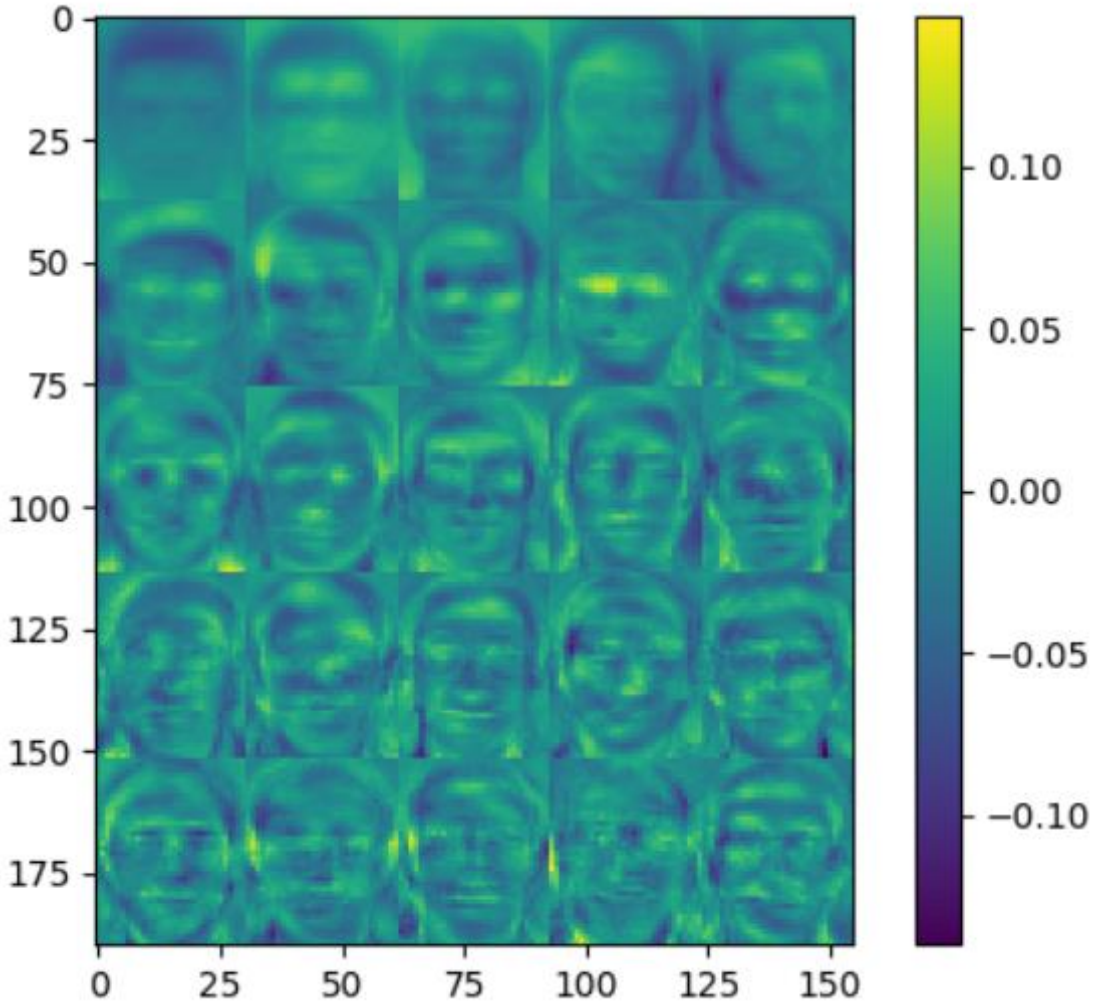
randomSelect 決定 index 是給定，按順序，或是隨機。Scaling 是將值移到 0~255 間，此處沒有用到。

這個函式最主要是要將影像的 row 接起來，其中 mergeImageToRow()就是負責將一行影像接起來的函式，我再將好幾行疊起來。

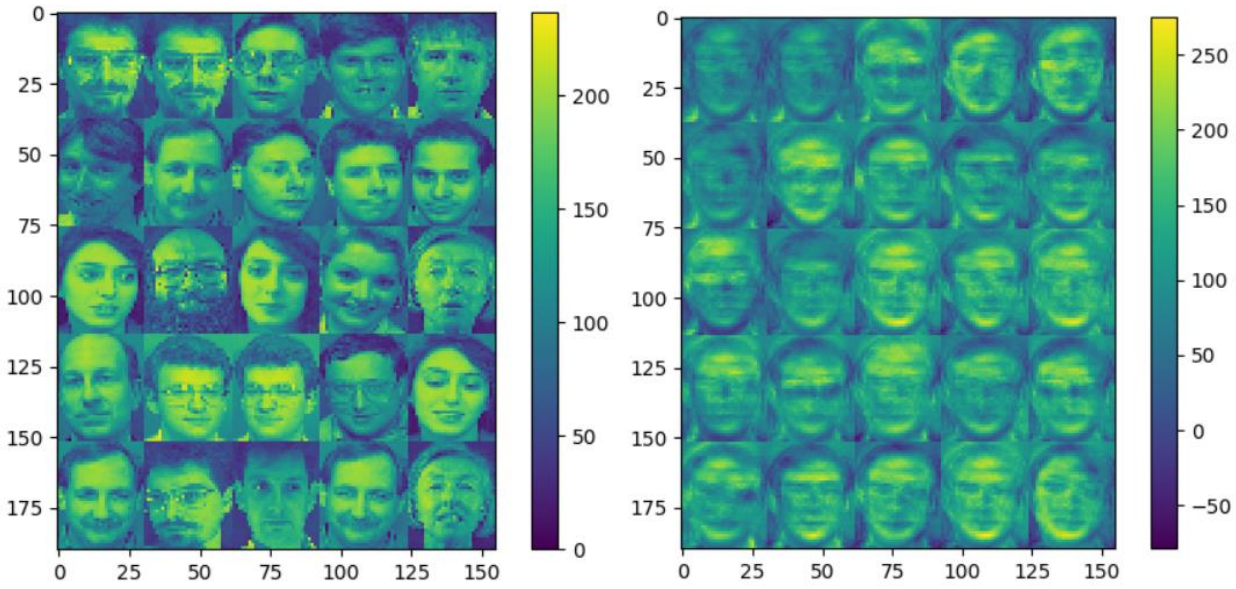
```python
def mergeImageToRow(imageArray,imageShape=None):
    #print(imageShape)
    #plt.imshow(imageArray[0,:,:])
    #plt.show()
    if imageArray.ndim==2:
        imageHeight,imageWidth=imageShape
        imageCount=imageArray.shape[0]
    elif imageArray.ndim==3:
        imageCount,imageHeight,imageWidth=imageArray.shape
    else:
        print("Illegal image array")
        return
    mergedImage=np.zeros((imageHeight,imageWidth*imageCount))
    for imageRow in range(imageHeight):
        for imageIndex in range(imageCount):
            if imageArray.ndim==2:
                mergedImage[imageRow, imageIndex*imageWidth:(imageIndex + 1) * imageWidth] = imageArray[imageIndex,i
            elif imageArray.ndim==3:
                mergedImage[imageRow, imageIndex*imageWidth:(imageIndex + 1) * imageWidth] = imageArray[imageIndex,
    return mergedImage
```

mergeImageToRow():

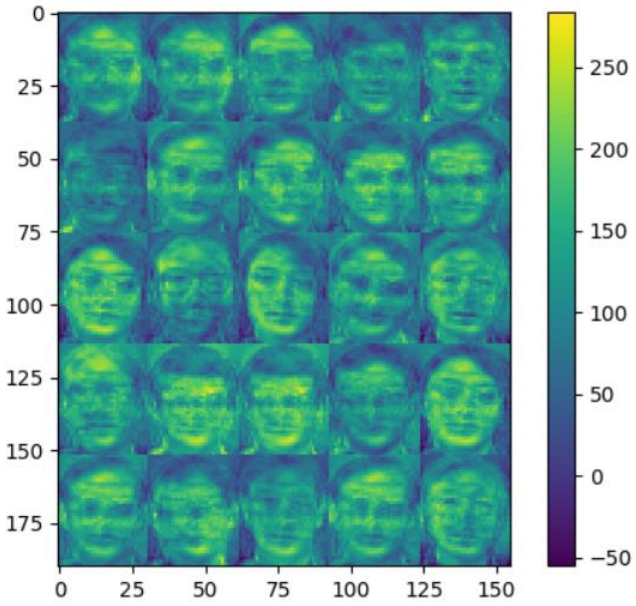
將一行影像接起來，以這次實驗為例就是輸出 1x5 的影像數。



這是根據 400 個臉做出的 eigenface，從左上到右下是 eigenface 的順序。觀察得到左上比較模糊，這是比重比較高比較通用的 eigenface，細節也會比較少。越是到右下角的細節就越來越多，是對於對應特定圖片細節的 face。



左邊是原圖，右邊是 25 eigenvector 的重構。



這是 50 eigenvector 的重構。可以觀察到 25 eigenvector 的重構各圖片比較相似，

50 eigenvector 可以明顯地看到原圖的樣子。

- RATIO and NORMALIZED CUT

```
#Help with scipy to speed up the calculation
if kernel=='linear':
    self.graphMatrix=linear_kernel(self.dataArray)
elif kernel=='RBF':
    self.graphMatrix = rbf_kernel(self.dataArray, gamma=gamma)
    #self.graphMatrix=np.maximum(self.graphMatrix-(1e-4),0)
elif kernel=='linearRBF':
    self.graphMatrix=linear_kernel(self.dataArray)+rbf_kernel(self.dataArray, gamma=gamma)
```

用了 scipy 以加速 similarity matrix 的運算。

Ratio 需要 generalized eigenvector

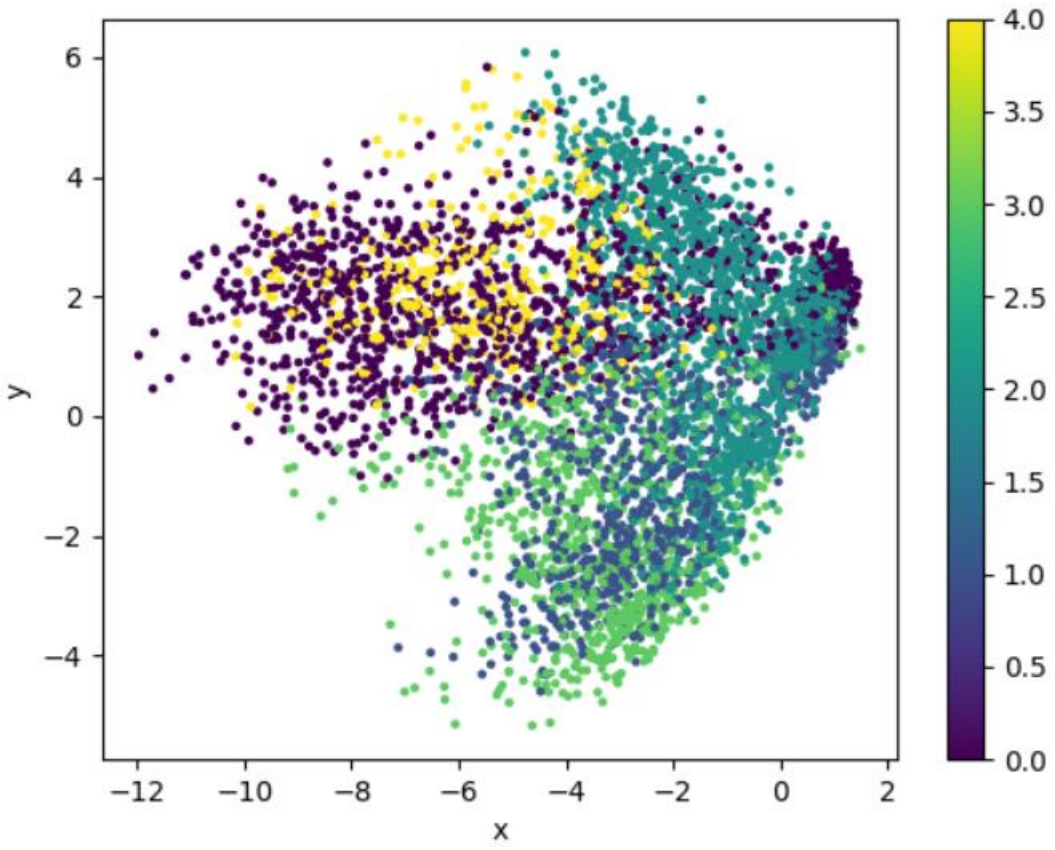
```
if mode=='ratio':
    self.eigenValue, self.eigenVector = lng.eig(self.graphLaplacian,self.degreeMatrix)
else:
    self.eigenValue,self.eigenVector=lng.eig(self.graphLaplacian)
```

Normalized cut 需要 normalized laplacian 以及對 row 做 normalize

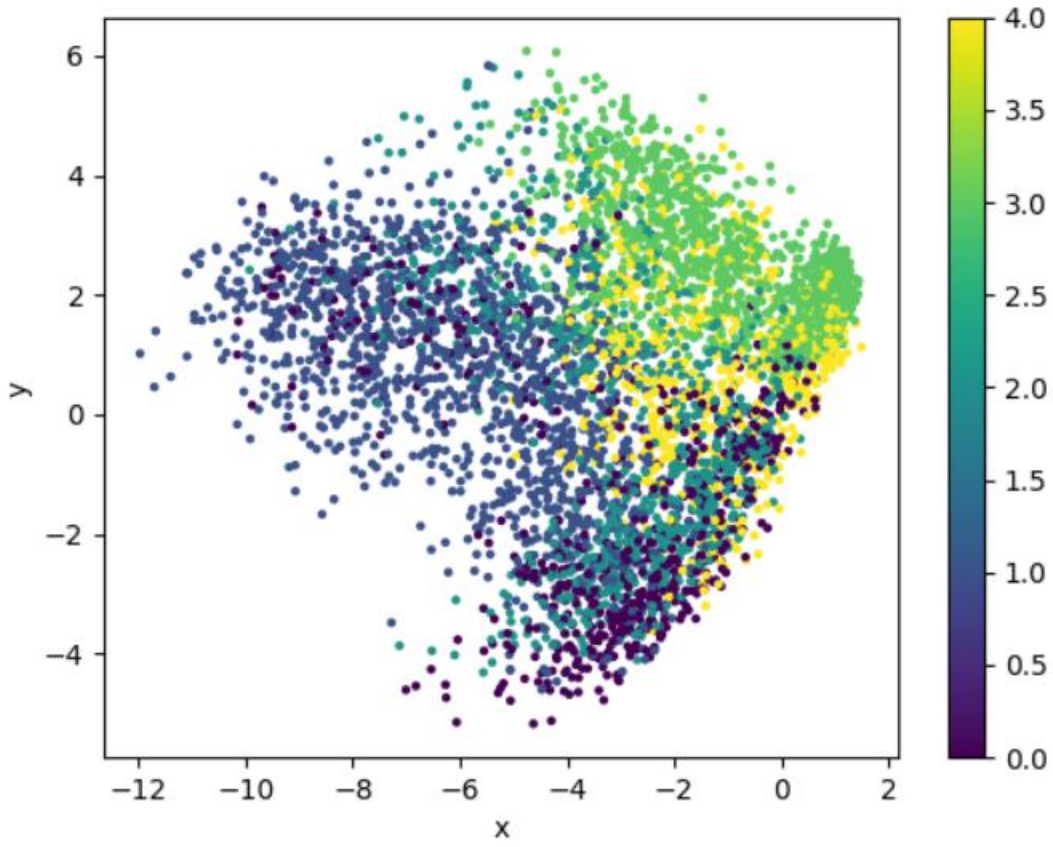
```
55          if mode=='ratio':
56              self.graphLaplacian=self.degreeMatrix-self.graphMatrix
57          elif mode=='normalized':
58              self.graphLaplacian = self.degreeMatrix - self.graphMatrix
59              degreeMatrixInvSqrt=sqrtm(lng.inv(self.degreeMatrix))
60              self.graphLaplacian=degreeMatrixInvSqrt@self.graphLaplacian@degreeMatrixInvSqrt
61          else:
62              print("unknown mode")
63              return
```

```
74          if mode=='normalized':
75              for eigenRow in range(self.eigenVector.shape[0]):
76                  normTerm=(np.square(self.eigenVector[eigenRow,:]))**0.5
77                  self.eigenVector[eigenRow,:]=self.eigenVector[eigenRow,:]/normTerm
```

- Normalized_Linear



- Normalized_Linear+RBF


6
4
2
0
−2
−4
y
−12
−10
−8
−6
−4
−2
0
2
x
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0