0416025 呂翊愷

圖形識別 Assignment #3

Methods you have implemented

I. HCM

① 先將原始dataset的順序隨機排序，接著再隨機取c個sample座標當作群的中心點，也就是初始的prototype
② 將每筆sample和c個群中心點的歐氏距離算出來，把該筆sample分配到距離最近的群
③ 接著更新prototype，把新的群裡的所有點座標取平均，當作新的群中心點座標
④ 重複以上②③步驟，直到prototype不再改變

II. FCM

① 先將原始dataset的順序隨機排序，接著再隨機取c個sample座標當作群的中心點，也就是初始的prototype
② 將每筆sample和c個群中心點的歐氏距離算出來，再依據公式算出每個sample和每個群的membership

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{ik}} \right)^{\frac{2}{q-1}}}$$

③ 依據公式算出prototype

$$\boldsymbol{v}_j = \frac{\sum_{i=1}^{N} (u_{ij})^q \boldsymbol{x}_i}{\sum_{i=1}^{N} (u_{ij})^q}$$

④ 重複以上②③步驟，直到membership改變的最大值小於設定的threshold

III. PCM

① 先將原始dataset的順序隨機排序，接著再隨機取c個sample座標當作群的中心點，也就是初始的prototype
② 將每筆sample和c個群中心點的歐氏距離算出來，再依據公式算出每個sample和每個群的membership

$$u_{ij} = \frac{1}{1 + \left(\frac{d_{ij}^2}{\eta_j} \right)^{\frac{1}{q-1}}}$$

③ 依據公式算出prototype

$$v_j = \frac{\sum_{i=1}^{N} (u_{ij})^q x_i}{\sum_{i=1}^{N} (u_{ij})^q}$$

④ 重複以上②③步驟，直到membership改變的最大值小於設定的threshold

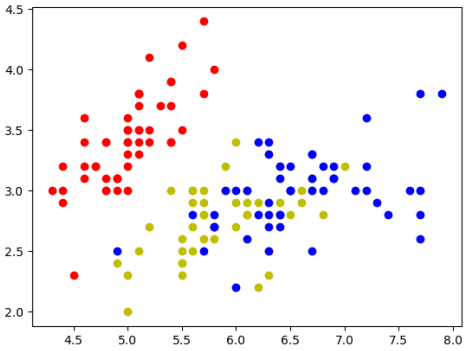
Experiments you have done, and the results

為了方便觀察分群的效果，我將迭代的次數還有ARI印出來觀察。另外，我將sample的其中兩維座標拿出來畫成一張二維的圖，並將每個點分群的結果以顏色表示，相同顏色代表分到相同的群。為了對照sample正確的分類，我也將sample實際的類別畫成二維的圖，將類別以顏色表示。此外，FCM和PCM經過測試後，threshold = 0.1, 0.01, 0.001的結果都差不多，因此後面的測試皆將threshold設為0.001。

I. FCM和PCM在不同fuzzy factor下的結果

Iris dataset

真實類別

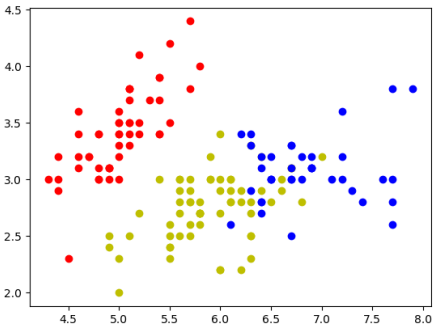


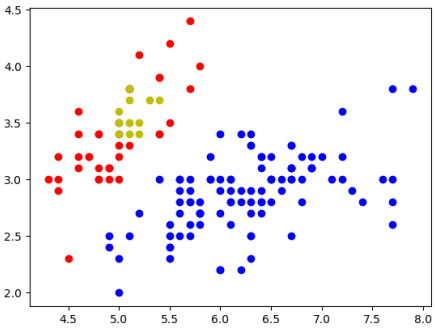
fuzzy factor = 1.5

FCM

PCM

```
FCM
iteration: 17
ARI: 0.421641
PCM
iteration: 4
ARI: 0.47519
```



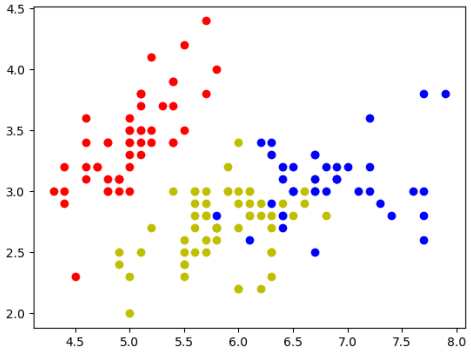


0416025 呂翊愷

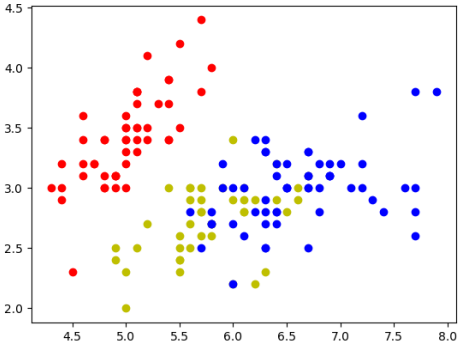
fuzzy factor = 2

```
FCM
iteration: 17
ARI: 0.72942
PCM
iteration: 24
ARI: 0.818358
```

FCM



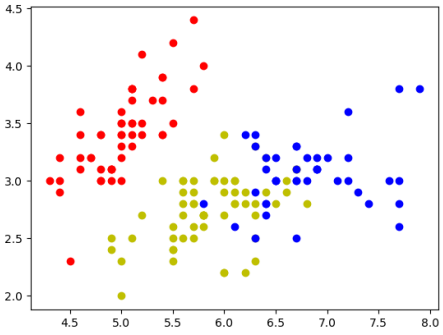
PCM



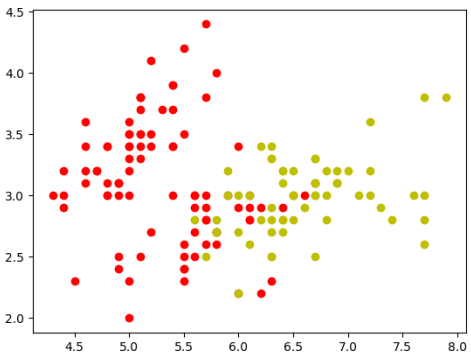
fuzzy factor = 3

```
FCM
iteration: 19
ARI: 0.742975
PCM
iteration: 54
ARI: 0.453893
```

FCM

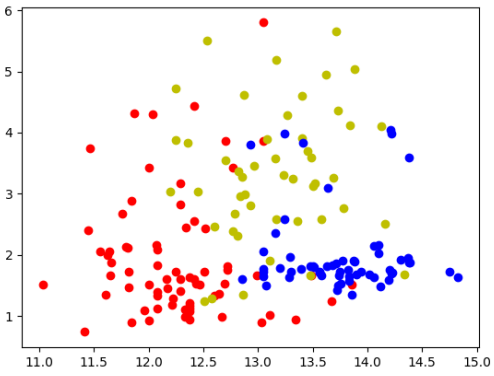


PCM



Wine dataset

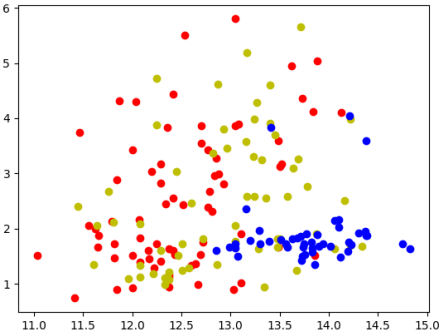
真實類別



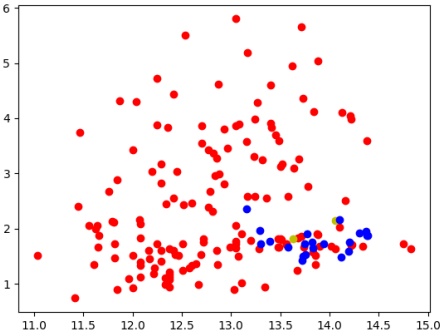
fuzzy factor = 1.5

```
FCM
iteration: 31
ARI: 0.353902
PCM
iteration: 2
ARI: 0.0753876
```

FCM



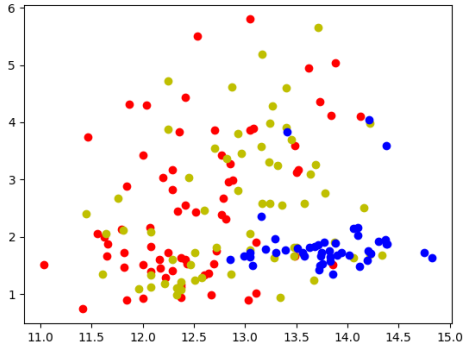
PCM



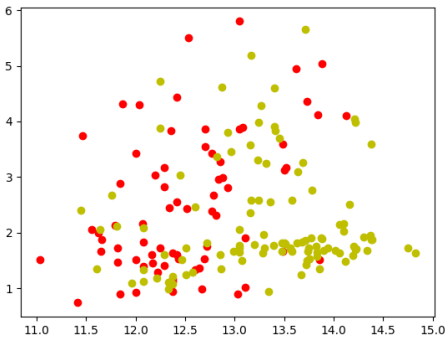
fuzzy factor = 2

```
FCM
iteration: 27
ARI: 0.360233
PCM
iteration: 78
ARI: 0.238617
```

FCM



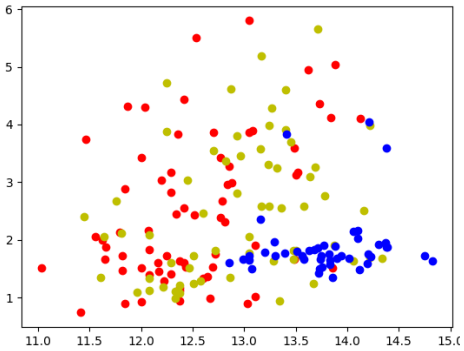
PCM



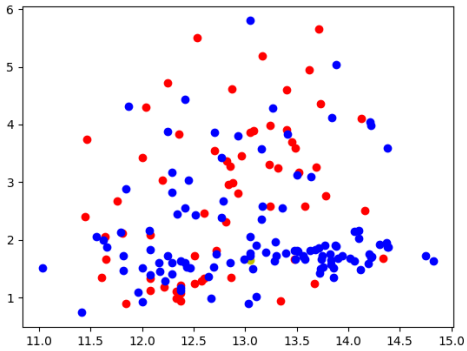
fuzzy factor = 3

```
FCM
iteration: 10
ARI: 0.360233
PCM
iteration: 2
ARI: 0.182766
```

FCM



PCM



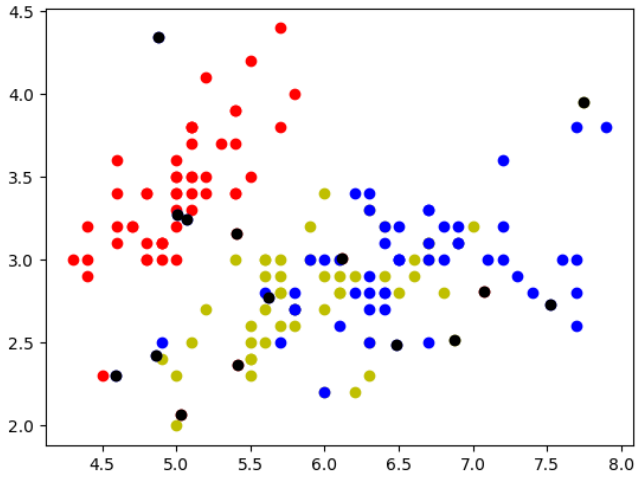
以fuzzy factor = 1.5, 2, 3分別做測試後發現，FCM和PCM在fuzzy factor = 2的時候表現都不錯，在兩個dataset中，ARI都是最高的，分類出的結果與sample真實類別最接近，因此之後的測試都以fuzzy factor = 2做其他項目的測試。

II. 加入noise的結果

將原有的dataset再加入一些雜訊，雜訊的取法是在每維座標的最大值和最小值之間隨機取一個數當作新的座標點，而雜訊的真實類別將隨機分配。為了觀察時得以區別雜訊，雜訊在真實類別的圖中以黑色表示，但在分群時雜訊將與原本的sample點一起被分群。

Iris dataset

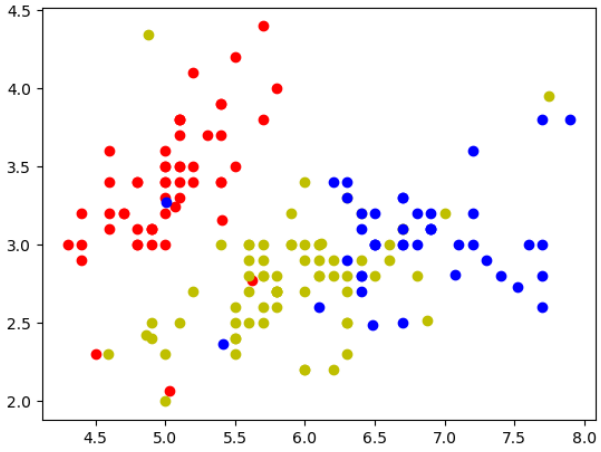
真實類別



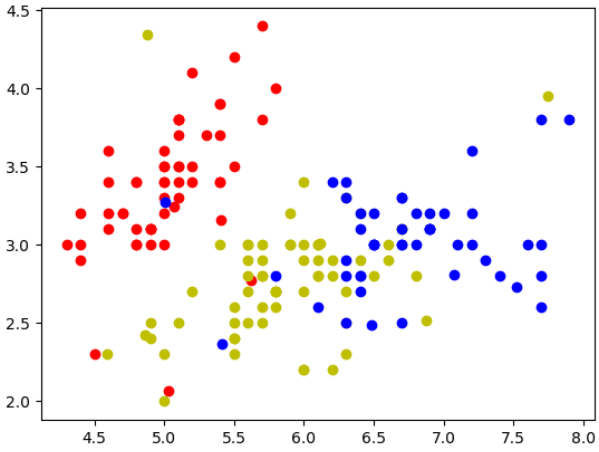
```
HCM
iteration: 9
ARI: 0.580044
FCM
iteration: 12
ARI: 0.654571
PCM
iteration: 24
ARI: 0.672447
```

0416025 呂翊愷

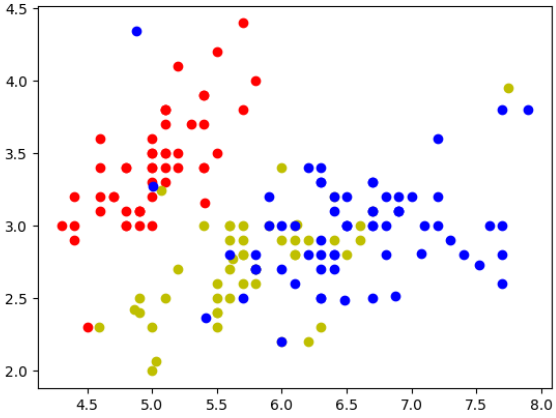
HCM



FCM

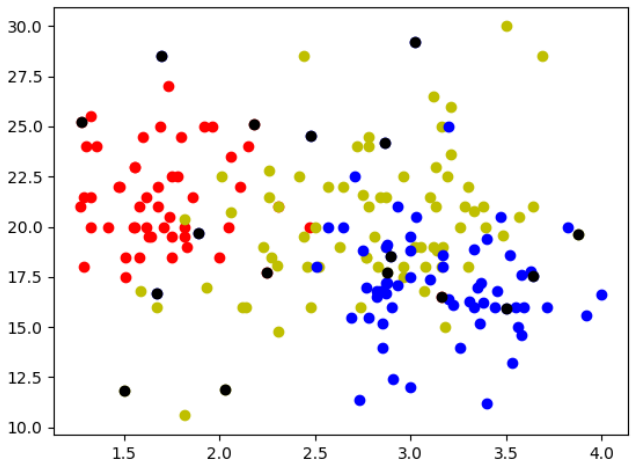


PCM



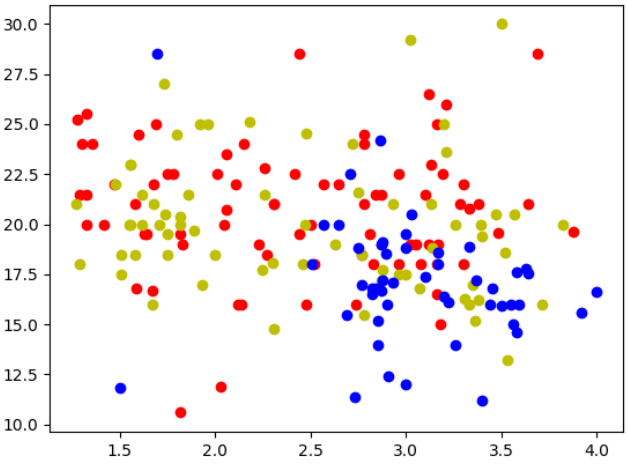
Wine dataset

真實類別

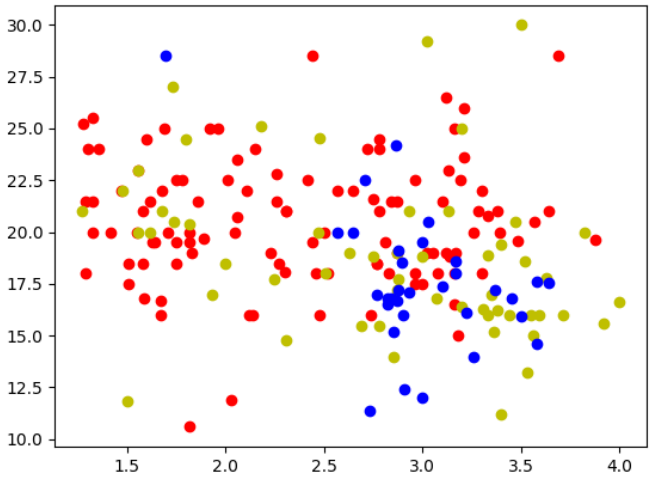



```
HCM
iteration: 4
ARI: 0.29231
FCM
iteration: 36
ARI: 0.31981
PCM
iteration: 2
ARI: 0.243584
```

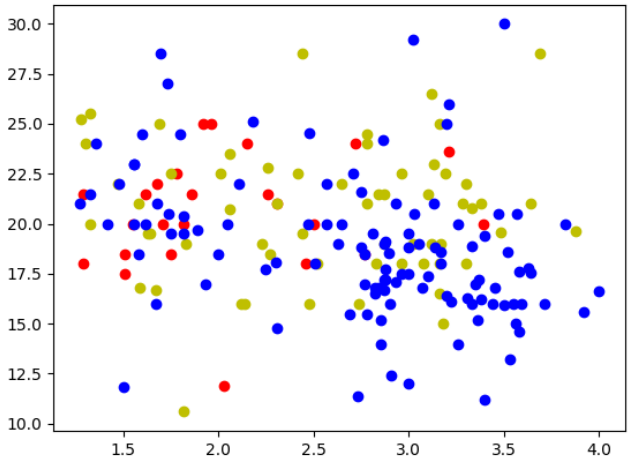
HCM



FCM



PCM



從結果可以發現，不管是HCM, FCM, PCM，加入雜訊後ARI都降低了，表示分群的表現都變差了，因此可以推論加入雜訊確實會影響分群表現。然而，雖然經過多次的測試，我無法證實PCM受到雜訊的影響最小，因為每次雜訊的影響大小都不一，難以做出整體的判斷。不過，我倒是有發現PCM的分類結果常常變化很大，有時ARI很高，有時ARI很低，也許是因為PCM對於初始取的群中心好壞很敏感，才會有這樣子的表現。

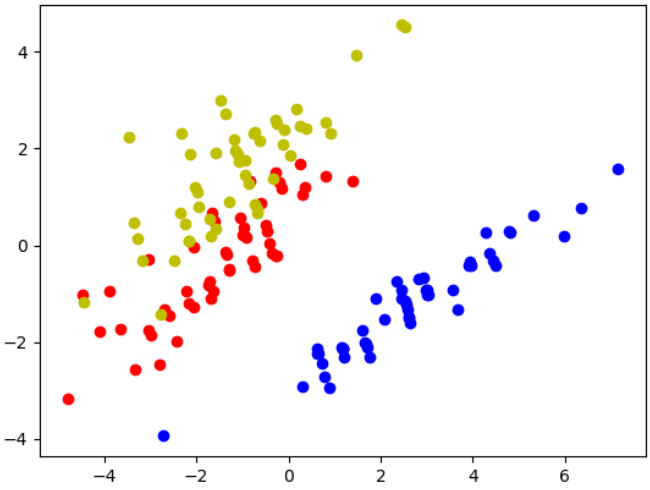
III. 先用PCA降維再分群

為了讓圖能夠真實呈現sample所在的位置，而不是只用其中兩維的座標來畫圖，我先用PCA將dataset降成2維再去分類，也順便觀察降維後的分類表現。

0416025 呂翊愷

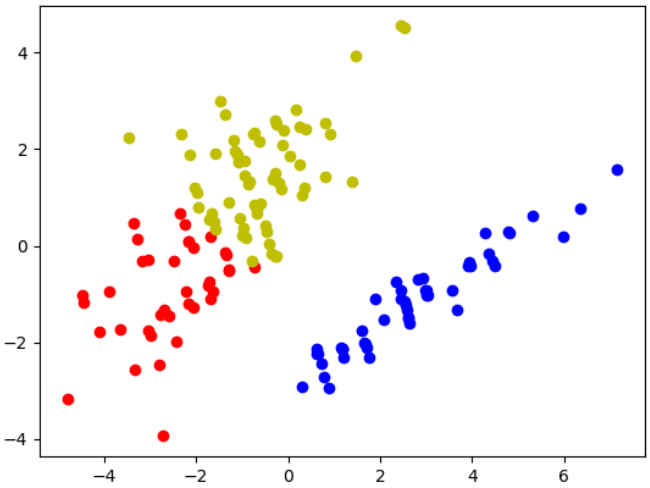
Iris dataset

真實類別

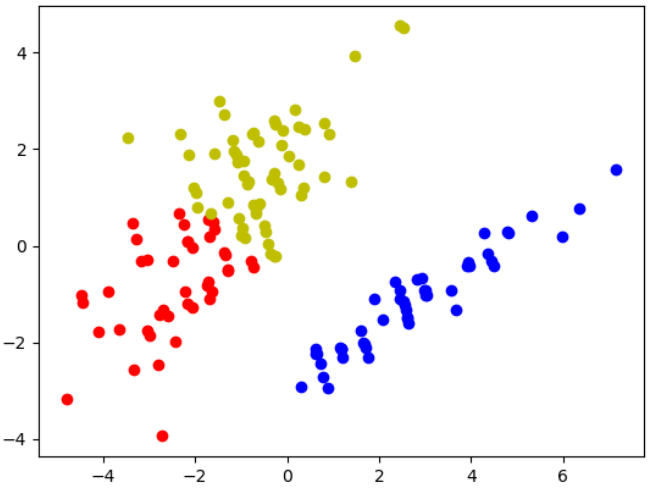


```
HCM
iteration: 10
ARI: 0.526533
FCM
iteration: 13
ARI: 0.522981
PCM
iteration: 27
ARI: 0.165391
```

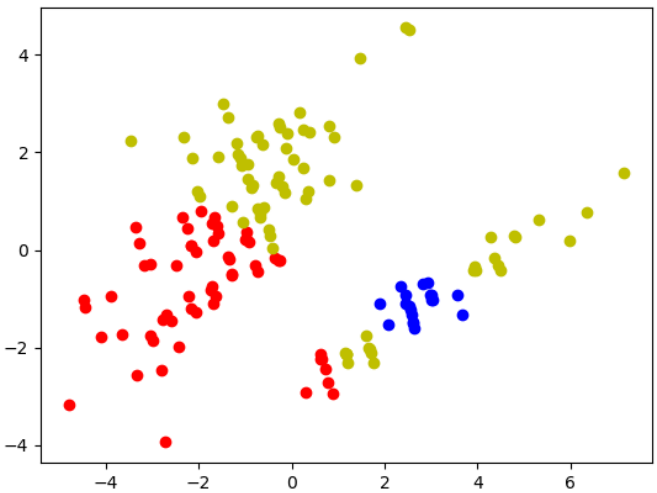
HCM



FCM



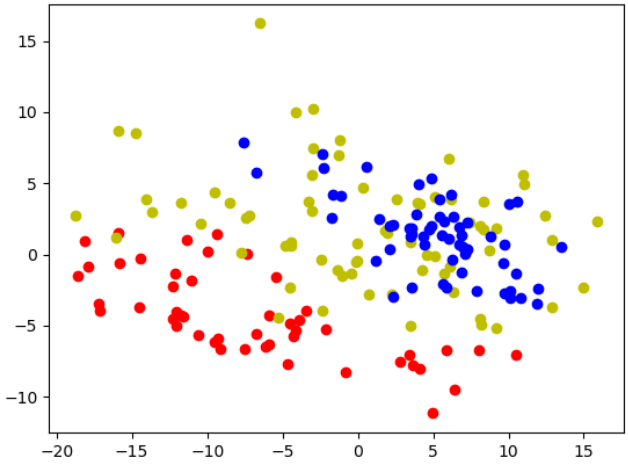
PCM



0416025 呂翊愷

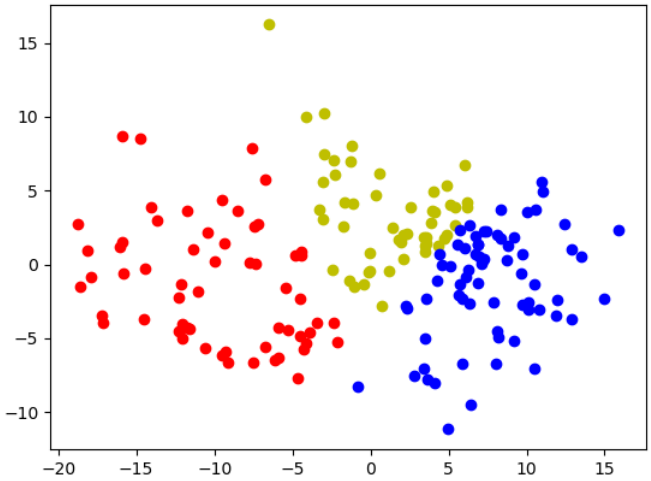
Wine dataset

真實類別

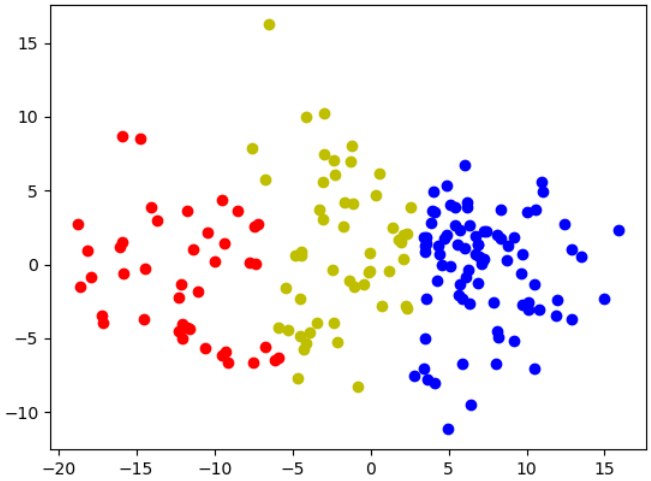


```
HCM
iteration: 6
ARI: 0.169545
FCM
iteration: 31
ARI: 0.151265
PCM
iteration: 42
ARI: 0.0476978
```

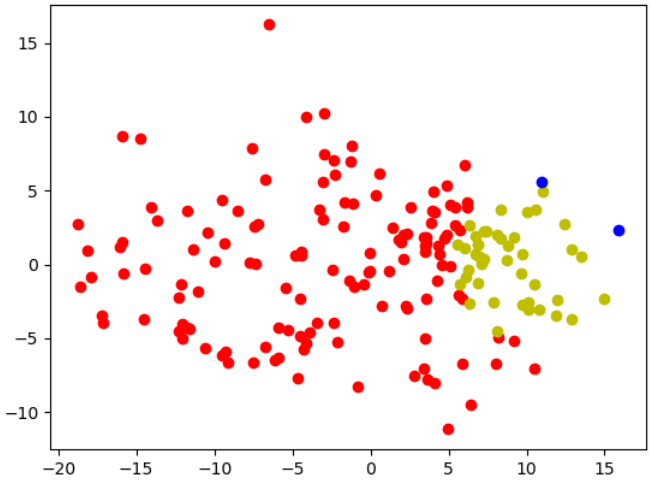
HCM



FCM



PCM



透過真實類別的圖可以發現，用PCA降維後，許多不同類別的sample會重疊在一起，導致分群上的困難，容易分錯，因此降維後的ARI也比較低。

Analysis - Are the results what you expect? Why?

經過三種測試後，測試的結果與我預期的其實差不多。上網查相關資料，許多網路上的高手都說fuzzy factor取2最適合，結果測試後發現真的是如此，我推測原因可能是fuzzy factor如果太低會分得太細，如果太高會分得越模糊。加入noise後，分群會受到noise的影響，容易將其他sample分錯，原因我想也是十分直觀，因為noise會干擾分群的過程。而降維後分群表現比較不好，仔細想想還蠻合理，降維本來就會失去一些資訊，導致分群上的困難。因此，這次作業的結果大致上與我想的相差不遠。

Code

I. Header.py

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

def addNoise(dataset):
    dropData = dataset.drop("variety", axis=1)
    dataArr = np.array(dropData)

    temp = []
    noise = []
    for i in range(dropData.shape[1]):
        for j in range(dropData.shape[0]):
            temp.append(dataArr[j][i])
        noise.append(np.random.uniform(min(temp), max(temp),
int(dataset.shape[0] * 0.1)))
        temp = []
    noisePoint = np.array(noise)
    noisePoint = np.transpose(noisePoint)

    noisePoint = pd.DataFrame(noisePoint, columns=dropData.columns)
    dataVar = []
    for i in range(noisePoint.shape[0]):
        dataVar.append('noise')
    dataVarFrame = pd.DataFrame(dataVar, columns=['variety'])

    noiseFrame = pd.concat([noisePoint, dataVarFrame], axis=1)
    newFrame = pd.concat([dataset, noiseFrame], axis=0,
ignore_index=True)

    return newFrame

# load iris dataset from csv
def loadIris(noise):
    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)

    if noise == 1:
        dataset = addNoise(dataset)

    randomData = dataset.sample(frac=1)
    dataVariety = randomData['variety'].unique()
    datasize = randomData.shape[0]

    return randomData, dataVariety, datasize

# load wine dataset from csv
def loadWine(noise):
```

```
    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)

    if noise == 1:
        dataset = addNoise(dataset)

    randomData = dataset.sample(frac=1)
    dataVariety = dataset['variety'].unique()
    datasize = randomData.shape[0]

    return randomData, dataVariety, datasize

# load ionos dataset from csv
def loadIonos(noise):
    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)

    if noise == 1:
        dataset = addNoise(dataset)

    randomData = dataset.sample(frac=1)
    dataVariety = dataset['variety'].unique()
    datasize = randomData.shape[0]

    return randomData, dataVariety, datasize

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

    if noise == 1:
        dataset = addNoise(dataset)

    randomData = dataset.sample(frac=1)
    dataVariety = dataset['variety'].unique()
    datasize = randomData.shape[0]

    return randomData, dataVariety, datasize
```

```
def calEucDist(vecX, vecY):
    return np.sqrt(np.sum(np.power(vecX - vecY, 2)))

def clustSorting(dataArr, dataLabel):
    dataVariety = np.unique(dataLabel)

    clust = []
    for i in range(len(dataVariety)):
        index = np.argwhere(dataLabel == dataVariety[i])
        clusterData = np.zeros([index.shape[0], dataArr.shape[1]])
        for j in range(index.shape[0]):
            clusterData[j] = dataArr[index[j]]
        clust.append(clusterData)

    x_mean = np.zeros([len(dataVariety)])
    for i in range(len(dataVariety)):
        x = clust[i][:, 0]
        x_mean[i] = np.mean(x)

    sortClust = []
    sortX_mean = np.argsort(x_mean)
    sortLabel = np.zeros([dataLabel.shape[0]])
    for i in range(dataLabel.shape[0]):
        temp = np.argwhere(dataVariety == dataLabel[i])
        sortLabel[i] = np.argwhere(sortX_mean == temp[0])

    for i in range(len(dataVariety)):
        sortClust.append(clust[sortX_mean[i]])
    return sortClust, sortLabel

def plotFig(clust, filename):
    color = ['ro', 'yo', 'bo']
    for i in range(len(clust)):
        # iris: 0, 1
        # wine: 6, 7, 8, 9, 10, 12 (6, 12)
        x = clust[i][:, 0]
        x = x.tolist()
        y = clust[i][:, 1]
        y = y.tolist()
        plt.plot(x, y, color[i])
    plt.draw()
    plt.savefig(filename)
    plt.show()

def plotNoiseFig(clust, filename, noiseArr):
    color = ['ro', 'yo', 'bo']

    for i in range(len(clust)):
        # iris: 0, 1
        # wine: 6, 7, 8, 9, 10, 12 (6, 12)
        x = clust[i][:, 0]
        x = x.tolist()
        y = clust[i][:, 1]
        y = y.tolist()
        plt.plot(x, y, color[i])
```

```
    noiseX = noiseArr[:, 0]
    noiseX = noiseX.tolist()
    noiseY = noiseArr[:, 1]
    noiseY = noiseY.tolist()
    plt.plot(noiseX, noiseY, 'ko')

    plt.draw()
    plt.savefig(filename)
    plt.show()

def realDataSorting(dataArr, dataLabel, dataVariety, noise):
    if noise == 1:
        # find noise data
        noiseIdx = np.argwhere(dataLabel == 'noise')
        noiseId = []
        for i in range(noiseIdx.shape[0]):
            noiseId.append(noiseIdx[i][0])
        noiseList = []
        for i in noiseId:
            noiseList.append(dataArr[i])
        noiseArr = np.array(noiseList)

    # cluster the normal data
    if noise == 1:
        idx = np.argwhere(dataVariety == 'noise')
        dataVariety = np.delete(dataVariety, idx)
        for i in noiseId:
            varIdx = np.random.randint(0, 3, dtype='int')
            dataLabel[i] = dataVariety[varIdx]

    clust = []
    for i in range(len(dataVariety)):
        index = np.argwhere(dataLabel == dataVariety[i])
        clusterData = np.zeros([index.shape[0], dataArr.shape[1]])
        for j in range(index.shape[0]):
            clusterData[j] = dataArr[index[j][0]]
        clust.append(clusterData)

    x_mean = np.zeros([len(dataVariety)])
    for i in range(len(dataVariety)):
        x = clust[i][:, 0]
        x_mean[i] = np.mean(x)

    sortClust = []
    sortX_mean = np.argsort(x_mean)
    sortLabel = np.zeros([dataLabel.shape[0]])
    for i in range(dataLabel.shape[0]):
        temp = np.argwhere(dataVariety == dataLabel[i])
        sortLabel[i] = np.argwhere(sortX_mean == temp[0])

    for i in range(len(dataVariety)):
        sortClust.append(clust[sortX_mean[i]])

    if noise == 1:
        return sortClust, sortLabel, noiseArr
    else:
```

```
        return sortClust, sortLabel
```

II. PCA.py

```
import numpy as np
import pandas as pd
import statistics as stat

import Header

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):
    # standardize data
    dropData = dataset.drop("variety", axis=1)
    dataMean = dropData.sum()
    dataMean = dataMean.values.reshape([dropData.shape[1], 1])
    dataMean = dataMean / dropData.shape[0]
    newData = normalize(dataset, dataMean)

    # calculate covX
    dropData2 = newData.drop("variety", axis=1)
    covX = calCov(dropData2)

    # calculate A
    eigValX, eigVecX = np.linalg.eig(covX)
    L = 2
    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 data
    projFrame = project(A, newData)

    return projFrame
```

III. HCM.py

```
import numpy as np
import random
from sklearn import metrics

import Header
```

```
def HCM(randomData, dataVariety, datasize, c, noise):
    # dataframe to numpy array
    dataLabel = np.array(randomData['variety'])
    dataLabel = dataLabel.reshape([datasize])

    dropData = randomData.drop("variety", axis=1)
    dataArr = np.array(dropData)
    dataList = dataArr.tolist()

    # randomly choose means
    means = random.sample(dataList, c)

    lastCluster = np.zeros([datasize])
    iteration = 0
    while 1:
        iteration += 1

        cluster = np.zeros([datasize])
        for i in range(datasize):
            dataDist = np.zeros([c])
            for j in range(c):
                dataDist[j] = Header.calEucDist(dataArr[i], means[j])
            cluster[i] = np.argmin(dataDist)
        if (lastCluster == cluster).all():
            break

        # recalculate means
        for i in range(c):
            index = np.argwhere(cluster == i)
            clusterData = np.zeros([index.shape[0], dataArr.shape[1]])
            for j in range(index.shape[0]):
                clusterData[j] = dataArr[index[j]]
            means[i] = np.mean(clusterData, axis=0)
        lastCluster = cluster
    print("HCM")
    print("iteration: %d" % iteration)

    predData, predLabel = Header.clustSorting(dataArr, lastCluster)
    if noise == 0:
        trueData, trueLabel = Header.clustSorting(dataArr, dataLabel)
    else:
        trueData, trueLabel, noiseArr = Header.realDataSorting(dataArr,
dataLabel, dataVariety, noise)

    Header.plotFig(predData, "HCM_figure1")
    if noise == 0:
        Header.plotFig(trueData, "HCM_figure2")
    else:
        Header.plotNoiseFig(trueData, "HCM_figure2", noiseArr)

    ARI = metrics.adjusted_rand_score(predLabel, trueLabel)
    print("ARI: %g" % ARI)
```

IV. FCM.py

```
import numpy as np
import random
from sklearn import metrics

import Header

def FCM(randomData, dataVariety, datasize, c, fuzzyFac, threshold,
noise):
    # dataframe to numpy array
    dataLabel = np.array(randomData['variety'])
    dataLabel = dataLabel.reshape([datasize])

    dropData = randomData.drop("variety", axis=1)
    dataArr = np.array(dropData)
    dataList = dataArr.tolist()

    # randomly choose means
    means = random.sample(dataList, c)

    iteration = 0
    d = np.zeros([datasize, c])
    lastU = np.zeros([datasize, c])
    cluster = np.zeros([datasize])
    while 1:
        iteration += 1

        # recalculate uij
        u = np.zeros([datasize, c])
        for i in range(datasize):
            for j in range(c):
                d[i][j] = Header.calEucDist(dataArr[i], means[j])

        for i in range(datasize):
            for j in range(c):
                temp = 0
                if d[i][j] == 0:
                    u[i][j] = 1
                else:
                    for k in range(c):
                        flag = 0
                        if d[i][k] == 0:
                            u[i][j] = 0
                        else:
                            flag = 1
                            temp += np.power((d[i][j] / d[i][k]), (2 /
(fuzzyFac - 1)))
                    if flag == 1:
                        u[i][j] = 1 / temp
            cluster[i] = np.argmax(u[i])

        # recalculate means
        for j in range(c):
            temp1 = np.zeros([1, dataArr.shape[1]])
            temp2 = 0
```

```
            for i in range(datasize):
                temp1 += np.power(u[i][j], fuzzyFac) * dataArr[i]
                temp2 += np.power(u[i][j], fuzzyFac)
            means[j] = temp1 / temp2

        # stopping criteria
        dif = u - lastU
        if abs(np.max(dif)) < threshold:
            break
        lastU = u
    print("FCM")
    print("iteration: %d" % iteration)

    predData, predLabel = Header.clustSorting(dataArr, cluster)
    if noise == 0:
        trueData, trueLabel = Header.clustSorting(dataArr, dataLabel)
    else:
        trueData, trueLabel, noiseArr = Header.realDataSorting(dataArr,
dataLabel, dataVariety, noise)

    Header.plotFig(predData, "FCM_figure1")
    if noise == 0:
        Header.plotFig(trueData, "FCM_figure2")
    else:
        Header.plotNoiseFig(trueData, "FCM_figure2", noiseArr)

    ARI = metrics.adjusted_rand_score(predLabel, trueLabel)
    print("ARI: %g" % ARI)
```

V. PCM.py

```
import numpy as np
import random
from sklearn import metrics

import Header

def PCM(randomData, dataVariety, datasize, c, fuzzyFac, threshold,
noise):
    # dataframe to numpy array
    dataLabel = np.array(randomData['variety'])
    dataLabel = dataLabel.reshape([datasize])

    dropData = randomData.drop("variety", axis=1)
    dataArr = np.array(dropData)
    dataList = dataArr.tolist()

    # randomly choose means
    means = random.sample(dataList, c)

    iteration = 0
    d = np.zeros([datasize, c])
    lastU = np.zeros([datasize, c])
    n = np.ones([c])
```

```
    cluster = np.zeros([datasize])
    while 1:
        iteration += 1

        # recalculate uij
        u = np.zeros([datasize, c])
        for i in range(datasize):
            for j in range(c):
                d[i][j] = Header.calEucDist(dataArr[i], means[j])
        for i in range(datasize):
            for j in range(c):
                if d[i][j] == 0:
                    u[i][j] = 1
                else:
                    temp = np.power(d[i][j], 2) / n[j]
                    u[i][j] = 1 / (1 + np.power(temp, 1 / (fuzzyFac -
1)))
            cluster[i] = np.argmax(u[i])

        # calculate n
        for j in range(c):
            temp1 = 0
            temp2 = 0
            for i in range(datasize):
                temp1 += np.power(u[i][j], fuzzyFac) * np.power(d[i][j],
2)
                temp2 += np.power(u[i][j], fuzzyFac)
            n[j] = temp1 / temp2

        # recalculate means
        for j in range(c):
            temp1 = np.zeros([1, dataArr.shape[1]])
            temp2 = 0
            for i in range(datasize):
                temp1 += np.power(u[i][j], fuzzyFac) * dataArr[i]
                temp2 += np.power(u[i][j], fuzzyFac)
            means[j] = temp1 / temp2

        # stopping criteria
        dif = u - lastU
        if abs(np.max(dif)) < threshold:
            break
        lastU = u
    print("PCM")
    print("iteration: %d" % iteration)

    predData, predLabel = Header.clustSorting(dataArr, cluster)
    if noise == 0:
        trueData, trueLabel = Header.clustSorting(dataArr, dataLabel)
    else:
        trueData, trueLabel, noiseArr = Header.realDataSorting(dataArr,
dataLabel, dataVariety, noise)

    Header.plotFig(predData, "PCM_figure1")
    if noise == 0:
        Header.plotFig(trueData, "PCM_figure2")
```

```
    else:
        Header.plotNoiseFig(trueData, "PCM_figure2", noiseArr)

    ARI = metrics.adjusted_rand_score(predLabel, trueLabel)
    print("ARI: %g" % ARI)
```

VI. main.py

```
import Header
import PCA
import HCM
import FCM
import PCM

noise = 0

randomData, dataVariety, datasize = Header.loadIris(noise)
# randomData, dataVariety, datasize = Header.loadWine(noise)
# randomData, dataVariety, datasize = Header.loadIonos(noise)
# randomData, dataVariety, datasize = Header.loadVertebral(noise)

# randomData = PCA.PCA(randomData)

HCM.HCM(randomData, dataVariety, datasize, 2, noise)
FCM.FCM(randomData, dataVariety, datasize, 2, 2, 0.001, noise)
PCM.PCM(randomData, dataVariety, datasize, 2, 2, 0.001, noise)
```