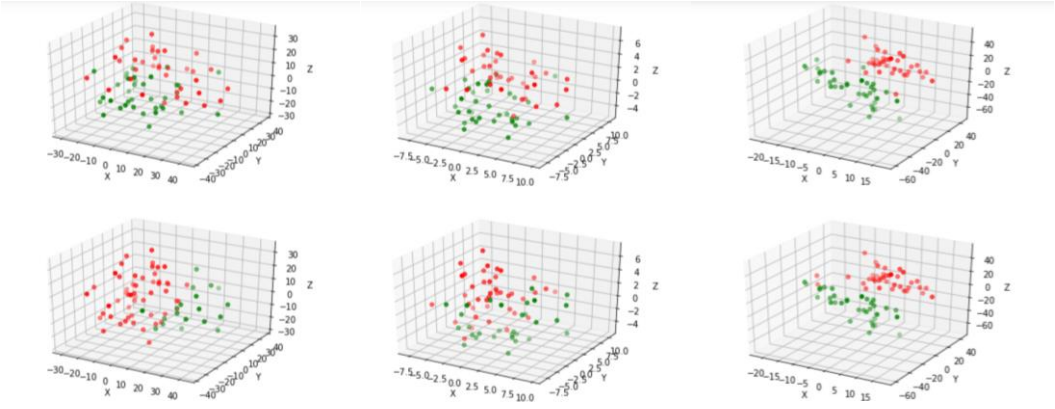
Lab1 PCA+K-MEANS

一、壓縮和讀檔

```python
#壓縮圖片
def compressImage(srcPath,dstPath):
    for filename in os.listdir(srcPath):
        if not os.path.exists(dstPath):
                os.makedirs(dstPath)

        #完整的檔案或資料夾路徑
        srcFile=os.path.join(srcPath,filename)
        dstFile=os.path.join(dstPath,filename)

        # 如果是檔案就處理
        if os.path.isfile(srcFile):

            if os.path.isfile(srcFile):
                try:
                    sImg=Image.open(srcFile)
                    dImg=sImg.resize((1,1),Image.ANTIALIAS) # 設定壓縮尺寸
                    dImg.save(dstFile)
                except Exception:
                    print(dstFile+"fail")

        # 如果是資料夾就遞迴
        if os.path.isdir(srcFile):
            compressImage(srcFile, dstFile)
```

一開始在做 PCA 時因為 memory 不夠，無法儲存這麼大的投影矩陣，所以決定在讀檔處理圖片時，就先壓縮圖片。



25*25　　5*5　　1*1

嘗試將不一樣大的圖片統一壓縮成不同的大小後，發現壓縮得越小，在分類的表現也會越好，同時注意到當我把圖片壓縮成 1*1 時，經過 PCA 降維處理後，在三維空間的分布跟前面相比呈現比較大的區別。

```python
compressImage("./sum/green/","./sum/green_compression/")     # 壓縮綠色芒果圖片
path='./sum/green_compression/'
files=os.listdir(path)
img_list = []

#讀檔
for file in files:

    p=path+file
    img = cv2.imread(p)
    img_2D = np.reshape(img,(-1,3))
    img_1D = np.reshape(img,(-1,1))                          # 將原先三維的數值展開成一維
    img_1D = img_1D.flatten()
    img_list.append(img_1D)
```

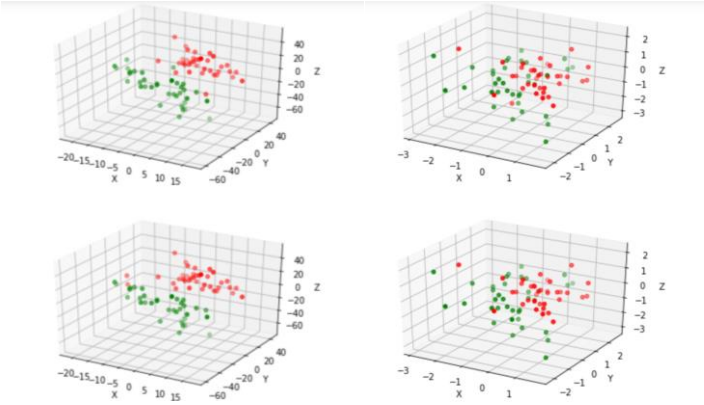
原本我是將每張圖片降維成二維 array，大小是一張圖片壓縮過後的 element 個數*每個樣本的屬性值個數(3)，然後對每張照片分別進行 PCA 處理，但結果顯示紅綠分布很明顯混雜在一起，所以後來改成先將每一張照片樣本的所有元素拉為一為陣列，再將代表每一張照片的 array 組成 matrix，最後一起丟入 PCA 進行處理。

二、PCA

```python
#標準化
def standardization(data):
    mu = np.mean(data, axis=0)
    sigma = np.std(data, axis=0)
    return (data - mu) / sigma
```

```python
#pca
def pca(XMat, k):
    m, n = np.shape(XMat)                       # 返回維度 m*n，m為照片總數
    average = np.mean(XMat, axis=0)             # 返回列的平均值。
    #data_adjust = XMat - average                # 對數據進行中心化處理
    data_adjust = standardization(XMat)
    #print(data_adjust.shape)

    covX = np.cov(data_adjust.T)                # 得到協方差矩陣

    eigenValue, eigenVec = np.linalg.eig(covX)  # 取特徵值和特徵向量

    if k > n:
        print('k must be lower than feature number')
        return
    else:
        selectVec = eigenVec[:, :k]             # 選取K個特徵向量，建立投影矩陣
        finalData = np.dot(data_adjust, selectVec) # 投影
        finalData = standardization(finalData)
        #reconData = np.dot(finalData, selectVec.T) + average
    return finalData, XMat
```

針對要準備進行 PCA 處理的數據，一開始我是採中心化處理，讓他們與平均相減，後來我又嘗試標準化的處理，分類效果也有提升。

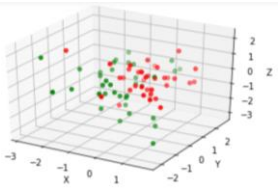
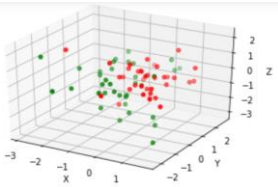
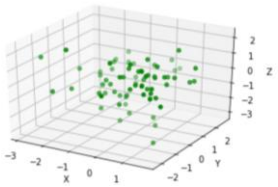
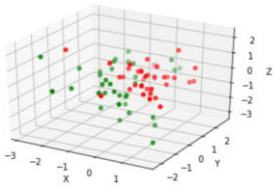


中心化　　標準化

三、K-MEANS

```python
def kmeans_xufive(ds, k):
    m, n = ds.shape                                 # m：樣本數量，n：每個樣本的屬性值個數
    result = np.empty(m, dtype=np.int)              # m個樣本的分類結果
    cores = np.empty((k, n))                        # k個質心
    cores = ds[np.random.choice(np.arange(m), k, replace=False)]
    #print(cores)
    #cores = [
    #    [np.random.randint(-3,1),np.random.randint(-2,2),np.random.randint(-3,2)]
    #    for i in range(2)
    #]                                               # 隨機生成質心

    while True: # 迭代計算
        d = np.square(np.repeat(ds, k, axis=0).reshape(m, k, n) - cores)
        distance = np.sqrt(np.sum(d, axis=2))       # 計算每個樣本距離k個質心的距離
        index_min = np.argmin(distance, axis=1)     # 標記每個樣本距離最近的質心

        if (index_min == result).all():             # 如果樣本分類沒有改變
            return result, cores                    # 則回傳分類結果和質心資料

        result[:] = index_min                       # 重新分類
        for i in range(k):
            items = ds[result==i]                   # 找出對應當前質心的分類樣本
            if len(items) != 0:
                cores[i] = np.mean(items, axis=0)   # 取分類樣本座標平均值作為新質心的位置
```

原先我是隨機生成兩個點做為分類的起始點，但後來發現這樣很容易讓分類不均，甚至遇到全部都分成同一類，所以後來我改成隨機取 72 個樣本數中的兩個點做為起始點，分類效果明顯好了很多。

隨機在空間中取點　　隨機從樣本點取點