這次的作業我們要用機器學習將不同芒果的影像圖片根據其顏色分成紅、綠兩類，過程中使用 PCA 將圖片向量降維，接著用 Kmeans 分類做預測，以下將依依說明本次作業所使用的步驟和過程：

1. Image Pre-processing

因為 dataset 裏頭的芒果影像大小不依所以在進行 PCA 前必須先將其所小成一樣的大小，一開始有點貪心把長寬都縮成 1000*1000（😜），但如下圖中可以看到這樣將導致 Memory 的問題，所以最後是用 30*30 的大小去對每張圖片做 resize。

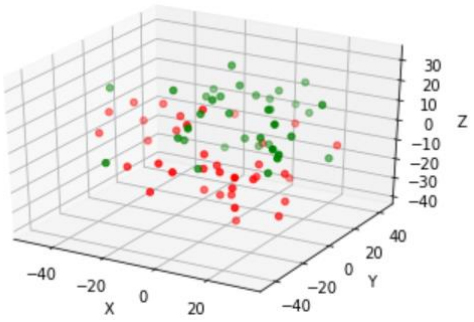
```
MemoryErrorTraceback (most recent call last)
<ipython-input-2-8a83bae8956f> in <module>()
     34 imgSet = np.asarray(imgSet)
     35
---> 36 transformed_data = PCA(imgSet)
     37 transformed_data = transformed_data.astype(float)
     38 print(len(transformed_data))

<ipython-input-1-5ee56598ebe2> in PCA(data, n_components)
     18     standardized_data = centered_data / std_vector
     19
---> 20     cov = np.cov(standardized_data.T)
     21     eig_values, eig_vectors = np.linalg.eig(cov)
     22

/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py in cov(m, y, rowvar, bias, ddof, fweights, aweights)
   3106     else:
   3107         X_T = (X*w).T
-> 3108     c = dot(X, X_T.conj())
   3109     c *= 1. / np.float64(fact)
   3110     return c.squeeze()

MemoryError:
```

[太大的圖形矩陣會導致在做 PCA 的時候 Memory 負荷不了]

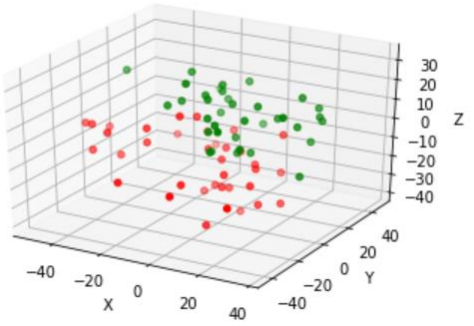
另外在做圖形前處理時，我又另外利用了 Gaussian Blur 的手法，可以減掉一些 noise，下列三張圖片分別是原圖、經過 resized、加上 Gaussian Blur 的圖片：

從上面的範例圖可以看到，經過 Gaussian Blur 處理的圖片芒果上的斑紋就沒那麼明顯，對整體芒果顏色的影響就沒那麼大，可能會較有利進行芒果的顏色判斷。

以下兩張 PCA 三維圖形是比較未經過 Gaussian Blur 和有經過 Gaussian Blur 出來的資料點分布，由這兩張圖判斷，經過 Gaussian Blur 前處理的資料似乎分布比較密集。



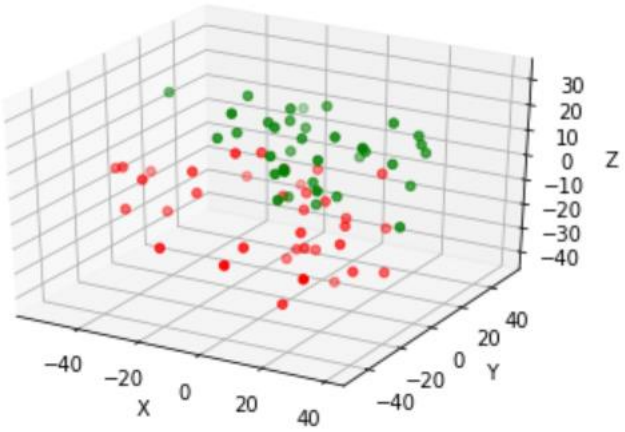
[Original Resized PCA distribution]



[Resized + Gaussian Blur PCA distribution]

2. Primary Component Analysis (PCA)

將檔案裡的圖片依依讀入做了簡單的前處理後，接下來就要將三維的圖片（這裡是：30*30*3 = 2700）轉成一維的向量，再將這一維的向量餵入 PCA 降維成 3 個 features，形成的 3D 視覺劃分佈如下：



```python
def PCA(data, n_components=3):
    mean_vector = np.mean(data, axis=0) # calculate the mean of each column
    std_vector = np.std(data, axis=0) # calculate standard deviation of each column


    # normalize here
    centered_data = data - mean_vector # center data by subtracting mean
    standardized_data = centered_data / std_vector



    cov = np.cov(standardized_data.T) # caluculate covariance matrix of centered matrix (
    eig_values, eig_vectors = np.linalg.eig(cov) # get eigenvalues and eigenvectors


    max_indexes = []
    for i in range(0, n_components):
        maxIndex = np.argmax(eig_values)
        max_indexes.append(maxIndex)
        eig_values[maxIndex] = 0

    # final_features = np.empty(0)
    final_features = eig_vectors[:, max_indexes]
```

這次的作業要求 PCA 要進行手刻，所以我就把 PCA 寫成一個 function，argument 吃輸入資料以及所要降成的維度，方便之後在優化 training 的時候可以用不同的維度去做。

3. KMeans Clustering

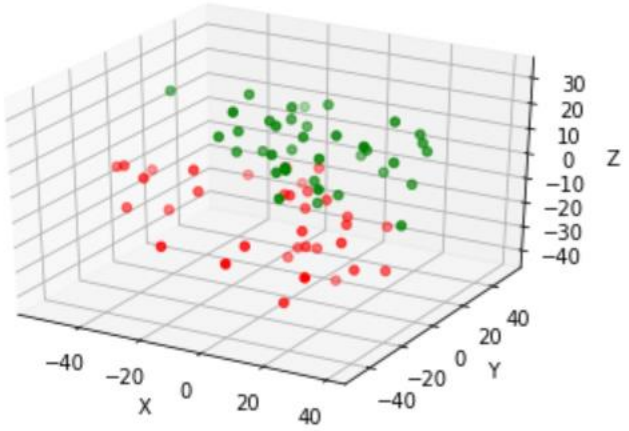
類似 PCA，這次的 KMeans 也要自己做，所以我就用 class 的方式去做，值得注意的是，雖然我在函示中有加上 max_iter 這個上限，但其實跑過 kmeans 後都發覺沒有到 300 個 iterations，資料還蠻快被分類的。如下可以看到一個 function 叫做 fit，就是要將 training data 進行分類

```python
class K_means:

    def __init__(self, n_clusters=2, tol=0.0001, max_iter=300):
        self.n_clusters = n_clusters
        self.tol = tol
        self.max_iter = max_iter

    def fit(self, data, initial_centroids):
        self.centroids = {}
        for i in range(self.n_clusters):
            self.centroids[i] = data[initial_centroids[i]]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.n_clusters):
                self.classifications[i] = []


            for datapoint in data:
                distances = [np.linalg.norm(datapoint - self.centroids[centroid]) for centroid in self.centroids
                classification = distances.index(min(distances))
```

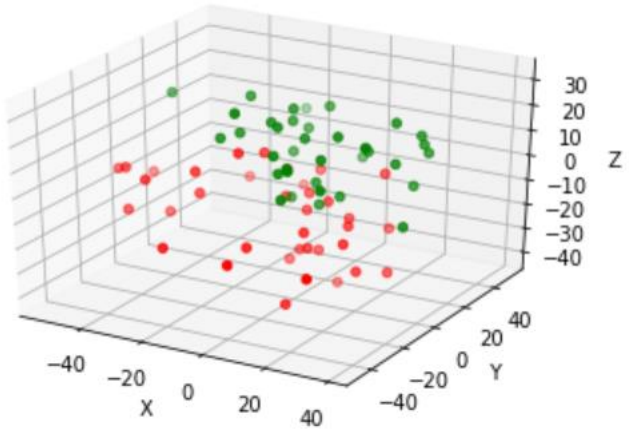
最後再用 predict 的方式將最後的分類中心點套用到 testing data 將其分類預測

```python
def predict(self, datapoint):
    distances = [np.linalg.norm(datapoint - self.centroids[centroid]) for centroid in self.centroids]
    classification = distances.index(min(distances))
    return classification
```

但因為這次作業是針對同一筆資料去做分類，所以基本上做的分類和 fit 的最後一個 iteration 是一樣的，在分類後會將圖片分成下圖的兩個 class



[classified dataset]

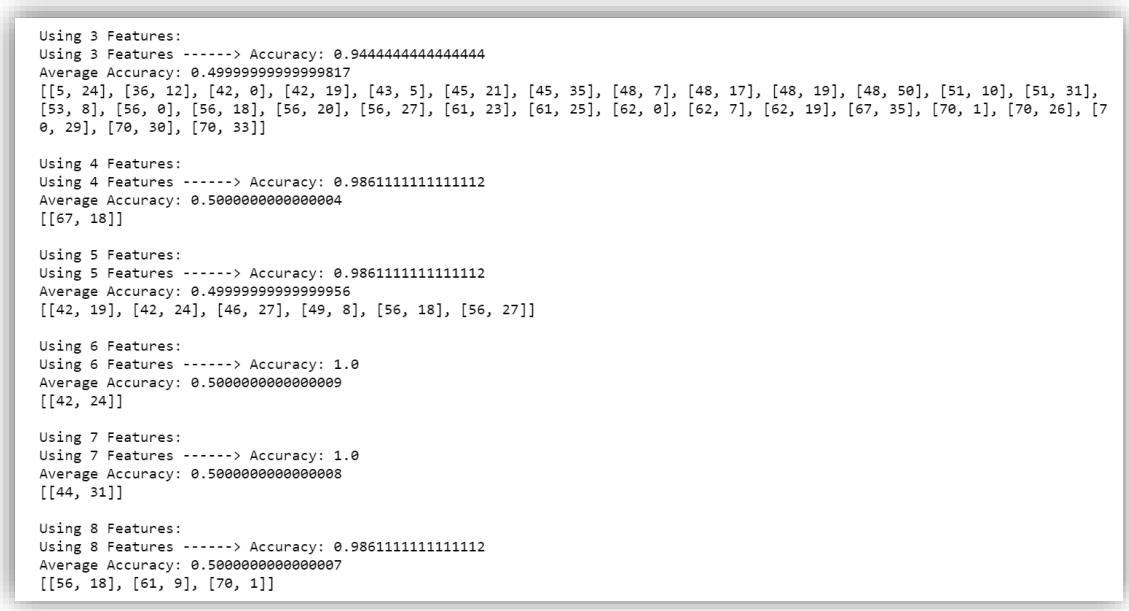


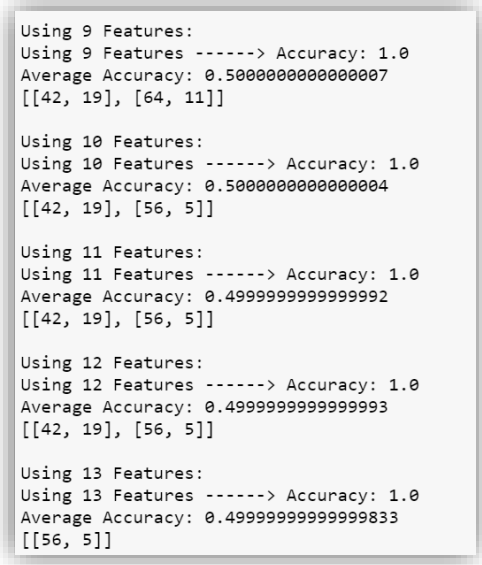
[original dataset]

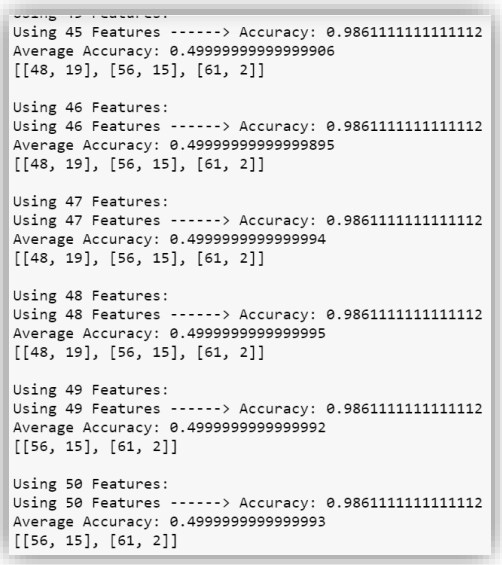
在和原始圖片顏色分類比較，整體正確率還蠻可觀的，唯獨中間綠紅色相間平面有幾點判斷錯誤外，Kmeans 出來的結果是和圖片真正顏色分類蠻相似的，後面會繼續探討如何將預測優化提高判斷準確率。

4. Classification Optimization

繼先前做的 3 feature classification，在所有起始點組合下，最高的 accuracy 也只能做到 0.944，所以我就針對不同的 feature 數去做分類看可不可以提高 classification 的 accuracy







上圖擷取了部分用不同維度去做分類的結果，分別列了所使用維度、最高準確率、最高準確率起始中心點、平均準確率，針對這幾點做了幾項觀測討論：

1) 平均準確率

不管是用多少的維度去做平均準確率大致上都是 0.5，就跟一般瞎猜均勻分配的二分類資料集命中機率一樣，蠻符合直覺的

2) 中心點圖片特質

這次最終結果我是選用 9 維度，因為可以看到，維度再增加其實還是會有一筆的錯誤，針對 9 維的中心分類圖片如下：


可以看到兩者都是蠻明顯的紅色和綠色芒果，顏色分佈也蠻均勻的，背景中也沒有另一類顏色的雜訊干擾判別。

總結這次的優化經驗，在進行優化的過程，我發現影響 accuracy 最大的兩因素就是：

1) feature 數量
2) Kmeans 起始點

可以從上圖的實驗得知，這應該也是此份作業的用意之一，讓我體悟到其實參數的設定能對自己的 ml task 有舉足輕重的影響。