K-Means 分群演算法-機器學習

一、前言:

我選擇測試的是機器學習中的非監督式學習的 K-means 分群演算法，非監督式學習與監督式學習最大的不同之處就在於它的訓練資料是沒有標籤（答案）的。在這個測試裡面是使用鳶尾花的資料，對於花瓣（Petal）的長和寬跟花萼（Sepal）的長和寬來練習 K-Means 的分群演算法，並觀察其中運作的流程與狀況。

二、資料取得:

資料是使用 scikit-learn 資料集中鳶尾花的資料做為測試，在鳶尾花資料中，有三種品種，我們的目的是利用花瓣及花萼的長寬資料，使用 K-Means 分群法去讓電腦自動分類。資料格式如下圖:

Petal Length	Petal width	Calyx length	Calyx width	Target
5.1	3.5	1.4	0.2	0
5.8	2.7	3.9	1.2	1
6	2.7	5.1	1.6	1
6.9	3.1	5.4	2.1	2
6.7	3.1	5.6	2.4	2

target 0: setosa
target 1: versicolor
target 2: virginica

三、K-Means 分群法:

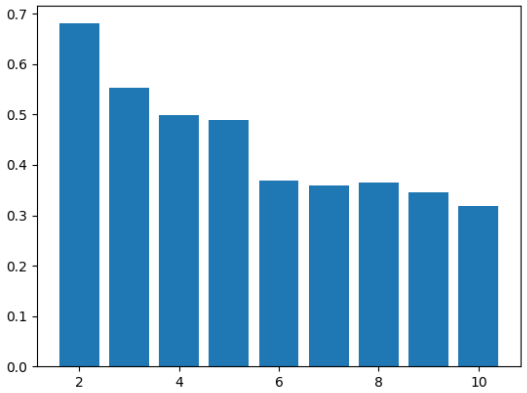
K-Means 演算法可以非常快速地完成分群任務，但是如果觀測值具有雜訊（Noise）或者極端值，其分群結果容易被這些雜訊與極端值影響，適合處理分布集中的大型樣本資料。

將 k 值設成 3 的分類結果如下圖，從分類結果可看出 k-means 分群法可以蠻準確地區分三個品種，但是在 setosa 和 versicolor 品種上分類似乎搞反了，詳細原因還需要再進一步研究。

```
分群結果：
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2
 2 0]
---
真實品種：
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

四、k 值的選擇:

隨著 k 值的增加，K-Means 演算法的績效一定會愈來愈好，當 k = 觀測值數目的時候，我們會得到一個**組間差異最大，組內差異最小**的結果，但這不是我們想要的，實務上我們讓程式幫忙選擇一個適合的 k。在這裡我測試 k 值從 2 到 10，如右圖，發現績效大概在 k=2 或 3 時，有最好表現。



五、參數變更

在 `sklearn.cluster.KMeans` 中有許多參數，如前面使用到的 `n_clusters`，在這裡我嘗試改變 `n_init` 及 `algorithm`，並探討其對績效與運算時間的改變。

(1)將 `n_init` 設定 10,30,50，`n_init` 為獲得初始聚類中心的更迭次數，因初始中心是隨機取得的，故此參數為修正初始中心造成的影響，故我預期其績效會會相同，但運算時間會隨更迭次數越多越久。

(2)將 `algorithm` 設定'auto', 'full', 'elkan'三種演算法，full 為 EM-style。

六、績效與運算時間評估

在此練習中績效使用的是 Silhouette 係數，其數值愈接近 1 表示績效愈好，反之愈接近 -1 表示績效愈差。

(1)變更 `n_init`

績效及運算時間的結果如下圖，由上而下，分別是 `n_init`=10,30,50，可以發現績效如預期的三者相同，而運算時間則依 `n_init` 的次數越多越久。而就分類的精準度而言，`n_init` 的次數越多似乎有越準的趨勢。

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
Time elapsed:    0.031243562698364258
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 0 2 0 2
 0 2]
Time elapsed:    0.07259392738342285
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
Time elapsed:    0.10373878479003906
[0.5525919445213676, 0.5525919445213676, 0.5525919445213676]
```

(2) 變更 algorithm

績效及運算時間的結果如下圖 a，由上而下，分別是 algorithm='auto', 'full', 'elkan'，可以發現在績效上三者是差不多的，而運算時間也差別不大，甚至再執行第二次時(如圖 b)，運算時間也是差不多的。而在分類的結果上，三種演算法都可蠻準的區分不同的品種，但是分類的正確性在執行兩次中，使用演算法 elkan 有不同的結果。以上是我初步的試驗結果，至於為何會有這樣的結果目前還不清楚，有待更深入的加強演算法的知識以及觀念的釐清。

```
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
  2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2
  2 0]
 Time elapsed:    0.03124213218688965
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
  0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
  0 2]
 Time elapsed:    0.02695631980895996
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  2 2 2 2 2 2 2 2 2 2 2 2 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0
  0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0
  0 1]
 Time elapsed:    0.026984691619873047
 [0.5525919445213676, 0.5525919445213676, 0.5525919445213676]
```

圖 a

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2
 2 0]
Time elapsed:    0.0312497615814209
auto
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
Time elapsed:    0.028842926025390625
full
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
Time elapsed:    0.032980918884277344
elkan
[0.5525919445213676, 0.5525919445213676, 0.5525919445213676]
```

圖 b