kMeans

2018 年 7 月 5 日

1 kMeans(分群)

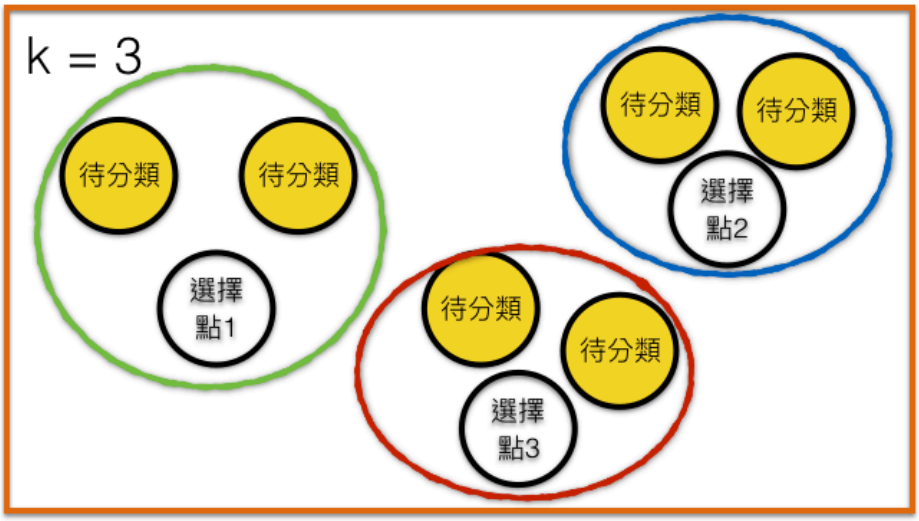
1.1 介紹

kMeans 跟之前介紹的決策樹和 kNN 有一個決定性的不同，就是他是一個非監督式演算法

有時候我們的數據太大量了，又沒人可以幫忙填寫正確的標籤，我們就希望電腦可以自動的幫我把標籤算出來。

你也可以想像成：非監督式學習就是讓電腦去判斷標籤的過程

注意: 雖然他跟 kNN 都有個 k，但實際的內容卻是完全不一樣的



1.2 理論基礎

1.2.1 KMeans

k 意味著在所有的資料裡選出 k 個質量中心，也就是以這 k 個質量中心分成 k 類，詳細步驟如下

1. 隨機選擇 k 個點當中心
2. 對於剩餘的點歸類到 k 類
3. 對於分類好的資料再次選擇一次 k 個質量中心 (讓質量中心更接近理想)
4. 重複步驟 2 和 3 直到穩定

1.2.2 KMeans++

你也發現了，我們的第一步驟，有可能選到非常爛的點當初始中心，那你就會花很多的步驟達到最後的穩定。

所以後來的人做了個改進，就是在選初始 k 點的時候盡量選遠一點的！這就是 kMeans++

1.3 k 值的選擇

k 值的選擇總共有兩種方法，我們先介紹第一種方法，後續再用第二種方法比較

第一種方法非常簡單，你想像成你已經知道數據有 n 類了 (ex. 鳶尾花有三類)，只是沒有人幫你標注這些類別

那毫無疑問的就是直接將 k 設定成你知道的 n!

1.4 開始撰寫程式

1.4.1 Step 0. 讀入我們的鳶尾花數據集作為練習

這裡我們用鳶尾話數據集來做實驗，但在訓練模型的時候我當作完全沒有 target 這件事 target 只用來在最後我的分群完成以後偷偷來看一下分的好不好

```
In [1]: from sklearn.datasets import load_iris
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        # 為了顯示的漂亮，我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
        # 大家練習的時候可以去掉下面兩行
        pd.set_option('display.max_rows', 15)
        pd.set_option('display.max_columns', 10)

        # 使用 scikit-learn 提供的鳶尾花資料庫
        iris = load_iris()
        df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
        df["target"] = iris["target"]
        df
```

Out[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	
5	5.4	3.9	1.7	0.4	
6	4.6	3.4	1.4	0.3	
..	...	...	...	...	
143	6.8	3.2	5.9	2.3	
144	6.7	3.3	5.7	2.5	
145	6.7	3.0	5.2	2.3	
146	6.3	2.5	5.0	1.9	
147	6.5	3.0	5.2	2.0	
148	6.2	3.4	5.4	2.3	

```
149                5.9               3.0                5.1                1.8


     target
0         0
1         0
2         0
3         0
4         0
5         0
6         0
..      ...
143       2
144       2
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

1.4.2 Step 1. (略過) 畫圖

因為我們已經畫過很多次了，這次就先把 heatmap 略過，讀者可以自行練習一下!

```
In [2]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練
        from sklearn.model_selection import train_test_split
        # 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
        data_train, data_test, target_train, target_test = train_test_split(iris['data'],
                                                                            iris['target'],
                                                                            test_size=0.1)
```

1.4.3 Step 2. 訓練模型

我們使用 kMeans 來訓練

1. 創好一個 Cluster
2. 使用 fit 將你要訓練的數據餵進來

```
In [3]: from sklearn.cluster import KMeans
        # 我事先已經有三類了，只是別人沒有幫我標註
        # 所以這裡要注意!! 我完全沒有帶入 target 喔
        clu = KMeans(n_clusters = 3)
        clu.fit(data_train)
```

```
Out[3]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
            n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
            random_state=None, tol=0.0001, verbose=0)
```

```
In [4]: # 我們大概可以看到資料已經被分成三類了
        clu.labels_
```

```
Out[4]: array([2, 1, 2, 1, 1, 0, 1, 1, 2, 1, 0, 0, 1, 0, 0, 1, 0, 2, 2, 1, 0, 2, 1,
               2, 1, 1, 0, 1, 2, 2, 1, 0, 1, 2, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0,
               1, 0, 1, 1, 0, 1, 1, 2, 0, 2, 0, 0, 0, 1, 2, 1, 1, 0, 2, 0, 0, 1, 1,
               2, 1, 2, 1, 1, 1, 0, 2, 0, 1, 1, 1, 1, 2, 0, 1, 1, 1, 2, 1, 1, 1, 0,
               0, 0, 1, 0, 1, 0, 1, 2, 0, 1, 1, 0, 0, 2, 0, 2, 2, 1, 0, 0, 2, 0, 0,
               1, 1, 0, 0, 0, 0, 1, 1, 0, 2, 2, 2, 1, 0, 2, 1, 0, 1, 0, 0], dtype=int32)
```

```
In [5]: from sklearn.metrics import accuracy_score

        predict = clu.predict(data_test)
        print("預測標籤:", predict)
        print("正確標籤:", target_test)
```

```
預測標籤: [0 1 0 2 2 1 1 2 0 1 0 2 1 2 0]
正確標籤: [0 1 0 2 2 1 1 2 0 1 0 2 1 2 0]
```

你可以看到我們已經正確的預測了，不過這裡有時候要小心，因為我們沒有事先給標籤
所以預測的 1 並不一定是正確標籤的 1。你要稍微做個轉換再來對照

1.5 不知道 k 的時候

當我們連 k 都不知道的時候 (ex. 分類人的性格，你不知道要分成幾種分類)
我們只能一個一個開始試，不過我們有一個很好的方法可以幫我們測試選的 k 究竟好不好

1.5.1 Silhouette 方法

Silhouette 是檢查一個點是不是分在最佳群的方法

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Which can be also written as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

From the above definition it is clear that

$$-1 \leq s(i) \leq 1$$

***a* 是這個點離他所在群內的其他點的平均距離，*b* 是這個點離他最鄰近的群的點的平均距離**

這個值會在-1~1 之間

算出來的值越大，代表這個 k 的選擇越好，我們只看上面的 1 - a/b，這東西要是 1 的話，a 必須為 0，也就是這個群根本就完美的聚集在一個點上

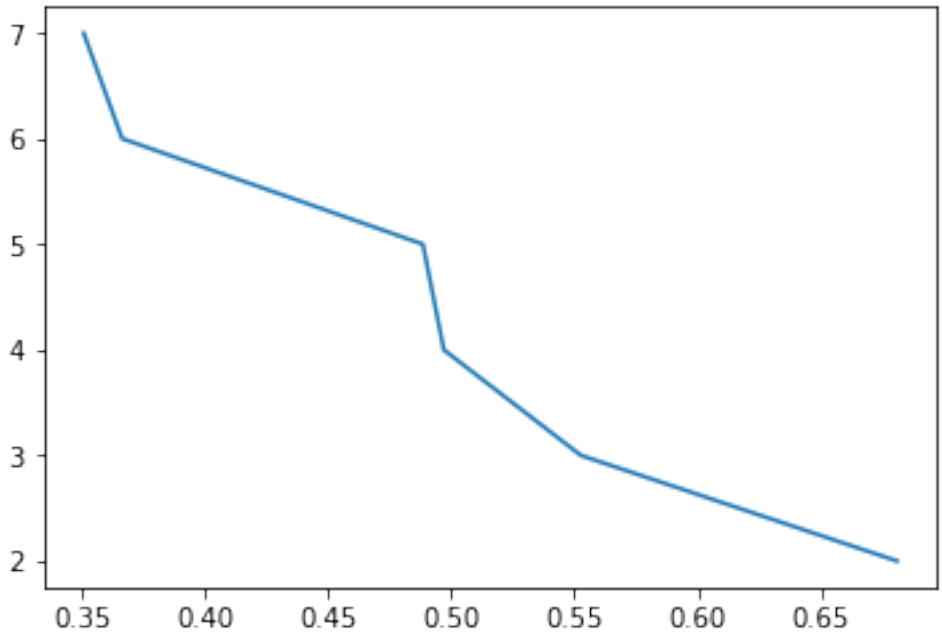
所以簡單來說，我們希望每一個點離他所在的群越近，離另外的群越遠，就是最棒的分類

```
In [9]: from sklearn.metrics import silhouette_score
        import matplotlib.pyplot as plt
        %matplotlib inline

        scores = []
        ks = []

        for i in range(2, 8):
            clu = KMeans(n_clusters = i)
            clu.fit(iris['data'])
            clu_score = silhouette_score(iris['data'], clu.labels_)
            scores.append(clu_score)
            ks.append(i)

        print("分數:", scores)
        print("K 值:", ks)
        plt.plot(scores, ks)
```

```
分數: [0.68081362027135084, 0.55259194452136762, 0.49699284994925963, 0.48851755085386322, 0.36
```

```
K 值: [2, 3, 4, 5, 6, 7]
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x10bd148d0>]
```



你可以看到，大概只有 k = 2 和 k = 3 的時候是一個合理的選擇，符合我們所知道的！

總共有三類的鳶尾花，那 2 為什麼也有很高的 score 呢？我們可以合理的推測其實這三類有兩類是很像的！