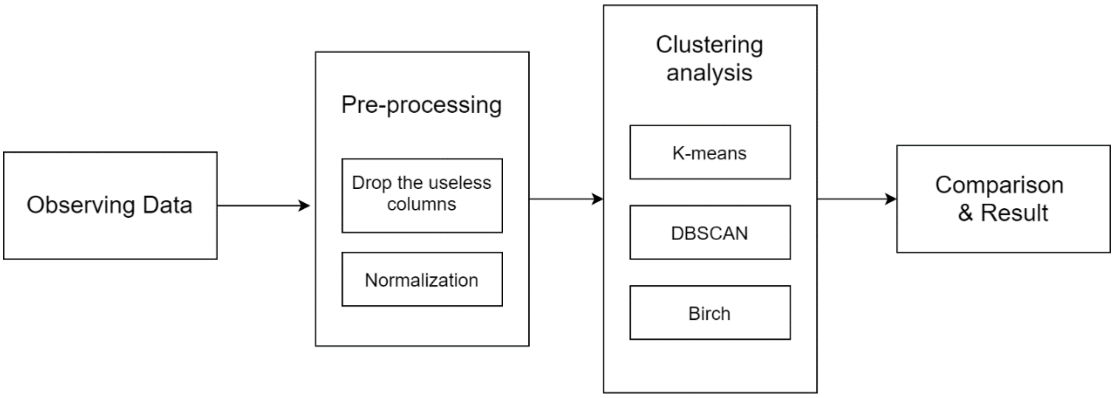
Data Science HW2

A10715006 張秋霞

Dataset：data.csv、test.csv
目標：透過 19 個 attributes 去分析各個 data 是否爲同一群。
程式碼檔案：hw2.py
執行方式：執行‘python hw2.py’即可。

一・程式架構



二・演算法流程&實作思路

觀察資料集

```
db = pd.read_csv(r'D:\2020\資料科學\hw2\data.csv',engine='python')
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2100 entries, 0 to 2099
Data columns (total 20 columns):
id          2100 non-null int64
feature1    2100 non-null int64
feature2    2100 non-null int64
feature3    2100 non-null int64
feature4    2100 non-null float64
feature5    2100 non-null float64
feature6    2100 non-null float64
feature7    2100 non-null float64
feature8    2100 non-null float64
feature9    2100 non-null float64
feature10   2100 non-null float64
feature11   2100 non-null float64
feature12   2100 non-null float64
feature13   2100 non-null float64
feature14   2100 non-null float64
feature15   2100 non-null float64
```

```
db.head()
```

	id	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	feature11	feature12	feature13	feature14	feature1
0	0	86	140	9	0.0	0.0	5.444444	4.768726	3.055556	2.678447	9.925926	6.000000	15.111111	8.666667	-11.777778	15.55555
1	1	34	93	9	0.0	0.0	0.555556	0.272165	0.388889	0.389682	14.814815	10.444445	23.444445	10.555555	-13.111111	25.88889
2	2	134	148	9	0.0	0.0	0.111111	0.172133	0.055556	0.136083	0.037037	0.000000	0.111111	0.000000	-0.111111	0.22222
3	3	199	144	9	0.0	0.0	0.333333	0.516398	0.333333	0.365148	0.444444	0.111111	1.111111	0.111111	-1.000000	2.00000
4	4	197	236	9	0.0	0.0	2.444444	6.829628	3.333333	7.599998	16.074074	13.111111	16.666668	18.444445	-8.888889	1.77777

首先通過 head()和 info()觀察資料集，可以看到資料集沒有缺失值等需要處理的值。

資料處理

```
db=db.drop(["feature1","feature2","feature3","feature4","feature5","feature6","feature7","feature8","feature9",
            "feature17","feature18","feature19","id"],axis=1)
# db=db.drop(["feature1","feature2","feature3","feature4","feature5","feature6","feature7","feature8","feature9","id"],axis=1)
# db=db.drop(["feature3","feature4","feature5","id"],axis=1)
#db=preprocessing.scale(db)
```

```
db.head()
```

	feature10	feature11	feature12	feature13	feature14	feature15	feature16
0	9.925926	6.000000	15.111111	8.666667	-11.777778	15.555555	-3.777778
1	14.814815	10.444445	23.444445	10.555555	-13.111111	25.888890	-12.777778
2	0.037037	0.000000	0.111111	0.000000	-0.111111	0.222222	-0.111111
3	0.444444	0.111111	1.111111	0.111111	-1.000000	2.000000	-1.000000
4	16.074074	13.111111	16.666668	18.444445	-8.888889	1.777778	7.111111

```
minmax = preprocessing.MinMaxScaler()
db = minmax.fit_transform(db)
```

刪除不重要的 feature，保留重要的 feature，並使用 MinMaxScaler()對其進行 normalization。

聚類分析

此次使用 sk-learn 套件，該套件提供了很多可以用於 Clustering 的函式，主要有：

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large n_samples, medium n_clusters with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium n_samples, small n_clusters	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large n_samples, medium n_clusters	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large n_clusters and n_samples	Large dataset, outlier removal, data reduction.	Euclidean distance between points

這次我選用 K-means、DBSCAN、Birch 這三個比較具有代表性的方法對資料集進行 Clustering analysis，並通過他們的 silhouette score 和 calinski harabasz score 進行對比並初步分析選用哪個方法。

其中，K-means 的使用如下圖：

```
#------------------------------------------------------K-means
for n_clusters in range(3,15,1):
    n_clusters=n_clusters
    clusterer = KMeans(n_clusters=n_clusters,random_state=10).fit(db)
    cluster_labels=clusterer.labels_
    silhouette_avg = silhouette_score(db,cluster_labels)
    cal=calinski_harabasz_score(db,cluster_labels)
    print(n_clusters,':',cal)
    print("n_cluster = ",n_clusters,'. The average silhouette_score is:', silhouette_avg)
```

```
3 : 5470.428841583426
n_cluster =  3 . The average silhouette_score is: 0.5502529946006443
4 : 5510.94422808757
n_cluster =  4 . The average silhouette_score is: 0.5288498409560366
5 : 6229.199985904982
n_cluster =  5 . The average silhouette_score is: 0.5442262370154631
6 : 6444.036250029173
n_cluster =  6 . The average silhouette_score is: 0.5204696355870047
7 : 6508.596968281795
n_cluster =  7 . The average silhouette_score is: 0.4890527388155857
8 : 6394.856137255166
n_cluster =  8 . The average silhouette_score is: 0.48797790939662017
9 : 6351.588483341102
n_cluster =  9 . The average silhouette_score is: 0.4913359419776868
10 : 6524.743201247439
n_cluster =  10 . The average silhouette_score is: 0.463064894326624
11 : 6550.824834982346
```

DBSCAN 的使用如下圖：

```
#------------------------------------------------------DBSCAN
for e in [0.019,0.021,0.022,0.023,0.024]:
    for m in [0.5,1,2,3]:
        clusterer = DBSCAN(eps =e,min_samples=m).fit(db)
        cluster_labels=clusterer.labels_
        silhouette_avg = silhouette_score(db,cluster_labels)
        cal=calinski_harabasz_score(db,cluster_labels)
        print(e,'_',m,':',cal)
        print("eps_min",e,'_',m,'. The average silhouette_score is:', silhouette_avg)
```

```
0.019 _ 0.5 : 372.5967185738437
eps_min 0.019 _ 0.5 . The average silhouette_score is: -0.006935653875546354
0.019 _ 1 : 372.5967185738437
eps_min 0.019 _ 1 . The average silhouette_score is: -0.006935653875546354
0.019 _ 2 : 19.40207158614366
eps_min 0.019 _ 2 . The average silhouette_score is: -0.14945616172141382
0.019 _ 3 : 26.28451609113237
eps_min 0.019 _ 3 . The average silhouette_score is: -0.14387538975162495
0.021 _ 0.5 : 383.638114525653
eps_min 0.021 _ 0.5 . The average silhouette_score is: -0.049518685369375724
0.021 _ 1 : 383.638114525653
eps_min 0.021 _ 1 . The average silhouette_score is: -0.049518685369375724
0.021 _ 2 : 29.641309454493808
eps_min 0.021 _ 2 . The average silhouette_score is: -0.09558720187221625
0.021 _ 3 : 38.431028128815626
```

Birch 的使用如下圖：

```
#------------------------------------------------------Birch
for n_clusters in [3,5,10,15]:
    for threshold in [0.005,0.1,0.5]:
        for branching_factor in [10,20,30]:
            clusterer = Birch(n_clusters = n_clusters, threshold = threshold, branching_factor = branching_factor).fit(db)
            cluster_labels=clusterer.labels_
            silhouette_avg = silhouette_score(db,cluster_labels)
            cal=calinski_harabasz_score(db,cluster_labels)
            print(n_clusters,'_',threshold,'_',branching_factor,':',cal)
            print(n_clusters,'_',threshold,'_',branching_factor,'. The average silhouette_score is:', silhouette_avg)
```

```
3 _ 0.005 _ 10 : 5026.2035582748085
3 _ 0.005 _ 10 . The average silhouette_score is: 0.5418437835549059
3 _ 0.005 _ 20 : 4993.863176834588
3 _ 0.005 _ 20 . The average silhouette_score is: 0.54043350261055
3 _ 0.005 _ 30 : 5038.177390364159
3 _ 0.005 _ 30 . The average silhouette_score is: 0.5424441322094951
3 _ 0.1 _ 10 : 4392.997458973866
3 _ 0.1 _ 10 . The average silhouette_score is: 0.5066442568570498
3 _ 0.1 _ 20 : 4373.591259667469
3 _ 0.1 _ 20 . The average silhouette_score is: 0.5034431990249539
3 _ 0.1 _ 30 : 4373.591259667469
```

對比及結果

整體來看，選用 K-means 或 Birch 的較好，於是分別選用它們對 data.csv 進行聚類分析，並判斷 test.csv 中的結果是否爲同一群，將結果上傳至 kaggle，最終發現使用 Birch 所得的分數更高，最終採用 Birch，程式碼如下：

```python
import warnings
warnings.filterwarnings("ignore")
test = pd.read_csv(r'D:\2020\資料科學\hw2\test.csv',engine='python')
test=test.drop(['index'],axis=1)
# for e in np.arange([0.019,0.021,0.022,0.023,0.024]):
#for n_clusters in np.arange(20,15,1):
for n_clusters in [10,11,12,13,14,15,16,17,18,19]:
    for threshold in [0.005]:
        for branching_factor in [10]:
            cluster = Birch(n_clusters = n_clusters, threshold = threshold, branching_factor = branching_factor).fit(db)
            out=cluster.fit_predict(db)
            a=list(test.iloc[:, 0])
            b=list(test.iloc[:, 1])
            y_test=[]
            for j in range(400):
                if(out[a[j]]==out[b[j]]):
                    ans=1
                else:
                    ans=0
                y_test.append(ans)
            dataframe = pd.DataFrame({'ans':y_test})
            n_clusters_s=str(n_clusters)
            threshold_s=str(threshold)
            branching_factor_s=str(branching_factor)
            path=r'D:\2020\資料科學\hw2\Birch'
            path += '\\'+n_clusters_s+'_'+threshold_s+'_'+branching_factor_s+'.csv'
            cal=calinski_harabasz_score(db,out)
            print(n_clusters,'_',threshold,'_',branching_factor,':',cal)
            dataframe.to_csv(path,index=True,index_label='index',sep=',')
```

判斷是否同一個群

保存結果