Application of ML for Networking Lab 2

Student ID: 310581027

Student Name： 宋煜祥

- Read Data
  - Data: raw_data.csv
    - Rows: 4317, Columns: 86
  - Cluster: cluster.csv

Read Data

```python
data = pd.read_csv("dataset/raw_data.csv")
cluster_y = pd.read_csv("dataset/cluster.csv")
cluster_y = cluster_y['Cluster'] # Ground truth label
```

	ID	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Timestamp	Flow.Duration	Total.Fwd.Packets	...	mi
0	1651	172.217.29.66-10.200.7.196-443-39485-6	10.200.7.196	39485	172.217.29.66	443	6	26/04/201711:11:25	2021337	9	...	
1	6460	179.1.4.244-10.200.7.196-443-43024-6	10.200.7.196	43024	179.1.4.244	443	6	26/04/201711:11:53	65552	14	...	
2	6578	179.1.4.244-10.200.7.196-443-43031-6	10.200.7.196	43031	179.1.4.244	443	6	26/04/201711:11:54	107032	14	...	
3	7219	179.1.4.244-10.200.7.196-443-43064-6	10.200.7.196	43064	179.1.4.244	443	6	26/04/201711:11:58	75351	14	...	
4	7683	179.1.4.244-10.200.7.196-443-43076-6	10.200.7.196	43076	179.1.4.244	443	6	26/04/201711:12:00	65862	15	...	
...	...	...	...	...	...	...	...	...	...	...	...	
4312	3572701	172.16.255.183-10.200.7.7-53-59979-17	10.200.7.7	59979	172.16.255.183	53	17	15/05/201705:43:49	119040676	2146	...	
4313	3572728	172.16.255.183-10.200.7.7-53-59979-17	10.200.7.7	59979	172.16.255.183	53	17	15/05/201705:45:49	31408313	647	...	
4314	3573244	172.16.255.200-10.200.7.9-53-48859-17	10.200.7.9	48859	172.16.255.200	53	17	15/05/201705:36:16	76350907	4	...	
4315	3573361	172.16.255.200-10.200.7.9-53-48859-17	10.200.7.9	48859	172.16.255.200	53	17	15/05/201705:40:33	13621158	4	...	
4316	3573425	172.16.255.200-10.200.7.9-53-48859-17	10.200.7.9	48859	172.16.255.200	53	17	15/05/201705:43:33	104320155	4	...	

4317 rows × 86 columns

- # Data-Processing

  - ## One-hot-encode

```
col = list(data.columns)
# data[col] = data[col].apply(pd.to_numeric, errors='coerce').fillna(0.0)
data = pd.get_dummies(data[col])
# data = data.replace([np.inf, -np.inf], np.nan).dropna(axis=1)
data = pd.DataFrame(data, dtype='float')
```

- # Drop 內容全為 0 的 column, col_nums 從 86 → 74

```
col = list(data.columns)
zero_list = []
for i in col:
    n = 0
    for j in data[i].values:
        if j == 0:
            n += 1
    if n == len(data.index):
        zero_list.append(i)
zero_list
```

```
['Bwd.PSH.Flags',
 'Fwd.URG.Flags',
 'Bwd.URG.Flags',
 'RST.Flag.Count',
 'CWE.Flag.Count',
 'ECE.Flag.Count',
 'Fwd.Avg.Bytes.Bulk',
 'Fwd.Avg.Packets.Bulk',
 'Fwd.Avg.Bulk.Rate',
 'Bwd.Avg.Bytes.Bulk',
 'Bwd.Avg.Packets.Bulk',
 'Bwd.Avg.Bulk.Rate']
```

```
data = data.drop(columns=zero_list)
data
```

	ID	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Timestamp	Flow.Duration	Total.Fwd.Packets	...	mi
0	1651	172.217.29.66-10.200.7.196-443-39485-6	10.200.7.196	39485	172.217.29.66	443	6	26/04/201711:11:25	2021337	9	...	
1	6460	179.1.4.244-10.200.7.196-443-43024-6	10.200.7.196	43024	179.1.4.244	443	6	26/04/201711:11:53	65552	14	...	
2	6578	179.1.4.244-10.200.7.196-443-43031-6	10.200.7.196	43031	179.1.4.244	443	6	26/04/201711:11:54	107032	14	...	
3	7219	179.1.4.244-10.200.7.196-443-43064-6	10.200.7.196	43064	179.1.4.244	443	6	26/04/201711:11:58	75351	14	...	
4	7683	179.1.4.244-10.200.7.196-443-43076-6	10.200.7.196	43076	179.1.4.244	443	6	26/04/201711:12:00	65862	15	...	
...	...	...	...	...	...	...	...	...	...	...	...	
4312	3572701	172.16.255.183-10.200.7.7-53-59979-17	10.200.7.7	59979	172.16.255.183	53	17	15/05/201705:43:49	119040676	2146	...	
4313	3572728	172.16.255.183-10.200.7.7-53-59979-17	10.200.7.7	59979	172.16.255.183	53	17	15/05/201705:45:49	31408313	647	...	
4314	3573244	172.16.255.200-10.200.7.9-53-48859-17	10.200.7.9	48859	172.16.255.200	53	17	15/05/201705:36:16	76350907	4	...	
4315	3573361	172.16.255.200-10.200.7.9-53-48859-17	10.200.7.9	48859	172.16.255.200	53	17	15/05/201705:40:33	13621158	4	...	
4316	3573425	172.16.255.200-10.200.7.9-53-48859-17	10.200.7.9	48859	172.16.255.200	53	17	15/05/201705:43:33	104320155	4	...	

4317 rows × 74 columns

- Transform
  - MinMaxScaler
    - 使用 MinMaxScaler Method

      使 data 縮放至 0 到 1 之間

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

min_max_scaler = MinMaxScaler() #  MinMaxScaler
min_max_scaler = min_max_scaler.fit(data)
data = min_max_scaler.transform(data)
data
```

```
array([[0.00000000e+00, 6.47741068e-01, 7.38591840e-03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.34638978e-03, 7.05797434e-01, 7.38591840e-03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.37942658e-03, 7.05912267e-01, 7.38591840e-03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [9.99949325e-01, 8.01519079e-01, 8.83642608e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [9.99982082e-01, 8.01519079e-01, 8.83642608e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 8.01519079e-01, 8.83642608e-04, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

  - PCA
    - 使用 PCA 將 feature 降至 3 維

      減少所需運算資源

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3) # PCA
pca.fit(data)
data = pca.transform(data)
data
```

```
array([[-0.35305975,  1.09697427,  0.25197482],
       [-0.34875805,  1.26272821,  0.35944478],
       [-0.34433078,  1.3196265 ,  0.41531745],
       ...,
       [ 0.46690672,  0.36754749, -0.42890163],
       [ 0.07044577,  0.45201016, -0.63901119],
       [ 0.64366885,  0.32989155, -0.33522627]])
```

Cluster algorithms 1: K-means

K-Means ¶

```
from sklearn import cluster, datasets, metrics
```

```
kmeans_fit = cluster.KMeans(n_clusters = 4, random_state=46).fit(data)
cluster_labels = kmeans_fit.labels_
predict = kmeans_fit.predict(data)
```

- **Measure performance (0.737)**
  - Adjusted mutual info score

Measure performance

```
from sklearn.metrics.cluster import adjusted_mutual_info_score
print("Adjusted Mutual Information: %0.3f" % adjusted_mutual_info_score(cluster_y,predict))
```

```
Adjusted Mutual Information: 0.737
```

- **Visualize data & Visualize clusters (2D)**

Visualize data & Visualize clusters (2D)

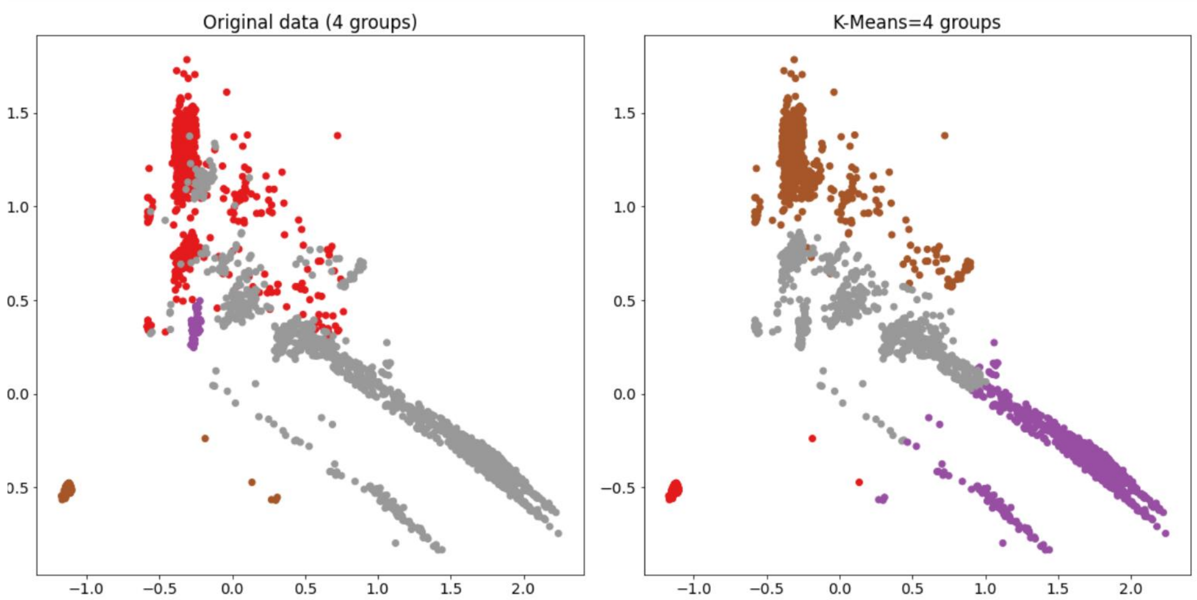
```
plt.rcParams['font.size'] = 14
plt.figure(figsize=(16, 8))
plt.subplot(121)
plt.title('Original data (4 groups)')
plt.scatter(data[:,0], data[:,1], c=cluster_y, cmap=plt.cm.Set1)

plt.subplot(122)
plt.title('K-Means=4 groups')
plt.scatter(data[:,0], data[:,1], c=predict, cmap=plt.cm.Set1)

plt.tight_layout()
plt.show()
```

- **Visualize data & Visualize clusters (3D)**

Visualize data & Visualize clusters (3D)

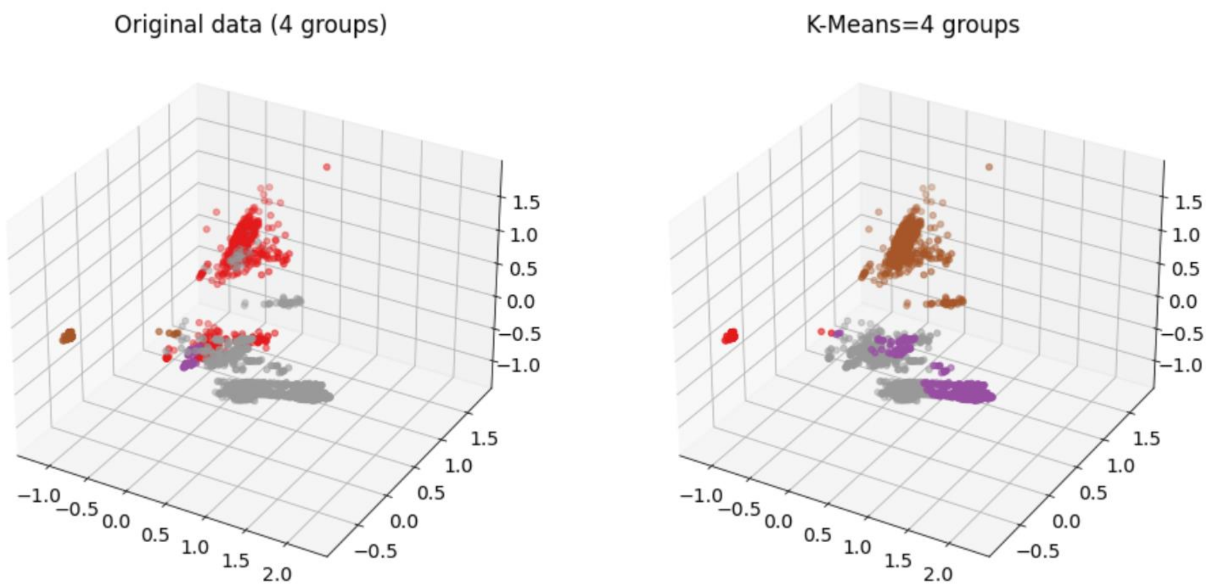
```
plt.rcParams['font.size'] = 14
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(121, projection='3d')
plt.title('Original data (4 groups)')
ax.scatter(data[:,0], data[:,1],data[:,2] ,c=cluster_y, cmap=plt.cm.Set1)
ax = fig.add_subplot(122, projection='3d')
plt.title('K-Means=4 groups')
ax.scatter(data[:,0], data[:,1],data[:,2] ,c=predict, cmap=plt.cm.Set1)
plt.show()
```

- 2-D



Original data (4 groups)
K-Means=4 groups

- 3-D



Original data (4 groups)
K-Means=4 groups

● Cluster algorithms 2: Birch

```
from sklearn.cluster import Birch
brc = Birch(n_clusters=4, threshold = 0.5, branching_factor = 20)
brc.fit(data)
predict = brc.predict(data)
```

```
predict
```

```
array([3, 3, 3, ..., 0, 0, 0], dtype=int64)
```

● Measure performance (0.740)

Measure performance

```
from sklearn.metrics.cluster import adjusted_mutual_info_score
print("Adjusted Mutual Information: %0.3f" % adjusted_mutual_info_score(cluster_y,predict))
```

```
Adjusted Mutual Information: 0.740
```

● Visualize data & Visualize clusters (2D)

Visualize data & Visualize clusters (2D)

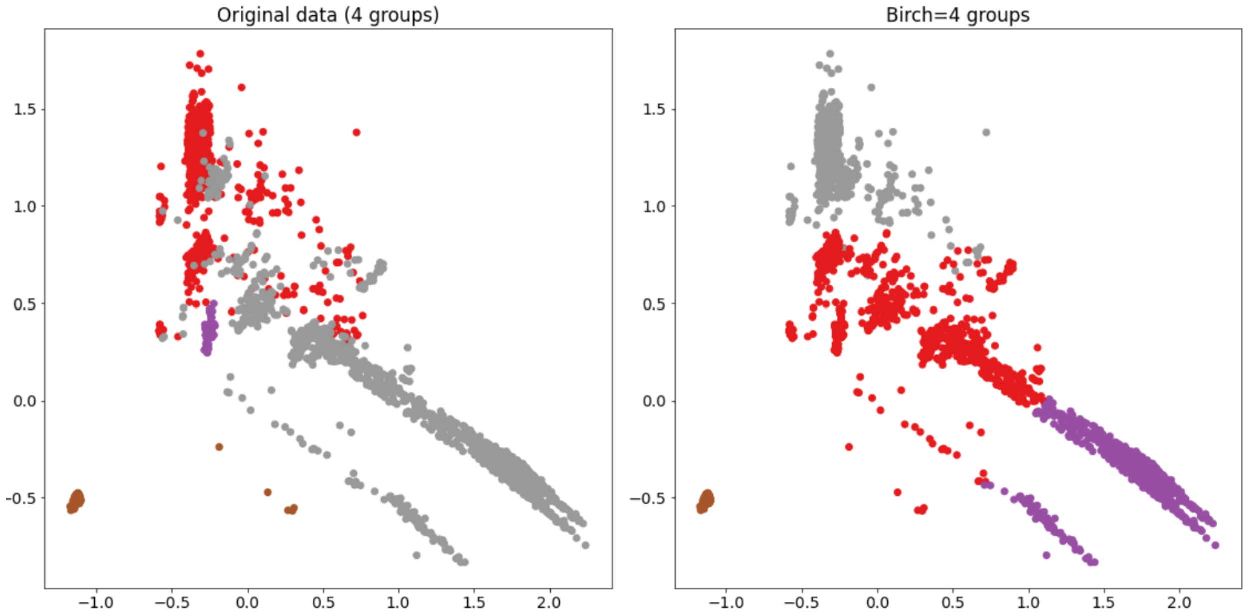
```
plt.rcParams['font.size'] = 14
plt.figure(figsize=(16, 8))
plt.subplot(121)
plt.title('Original data (4 groups)')
plt.scatter(data[:,0], data[:,1], c=cluster_y, cmap=plt.cm.Set1)

plt.subplot(122)
plt.title('Birch=4 groups')
plt.scatter(data[:,0], data[:,1], c=predict, cmap=plt.cm.Set1)

plt.tight_layout()
plt.show()
```

● Visualize data & Visualize clusters (3D)

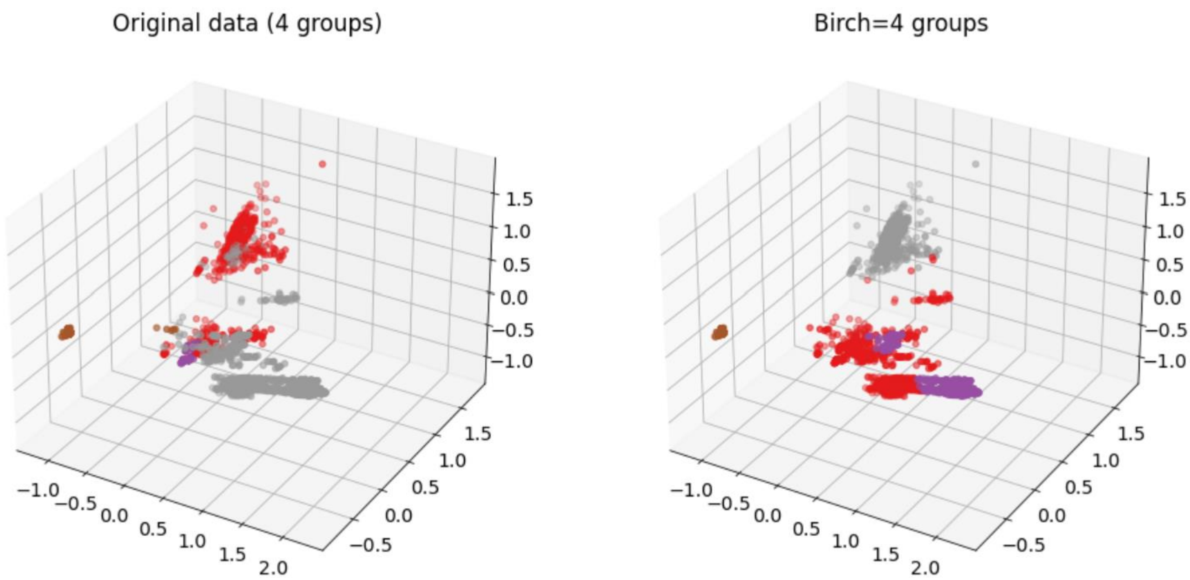
Visualize data & Visualize clusters (3D)

```
plt.rcParams['font.size'] = 14
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(121, projection='3d')
plt.title('Original data (4 groups)')
ax.scatter(data[:,0], data[:,1],data[:,2] ,c=cluster_y, cmap=plt.cm.Set1)
ax = fig.add_subplot(122, projection='3d')
plt.title('Birch=4 groups')
ax.scatter(data[:,0], data[:,1],data[:,2] ,c=predict, cmap=plt.cm.Set1)
plt.show()
```

● 2-D


Original data (4 groups)
Birch=4 groups
1.5
1.0
0.5
0.0
-0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

● 3-D


Original data (4 groups)
Birch=4 groups
1.5
1.0
0.5
0.0
−0.5
−1.0
2.0

● Cluster algorithms 1: AgglomerativeClustering

AgglomerativeClustering

```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward').fit(data)
predict = clustering.labels_
```

```
predict
```

```
array([3, 3, 3, ..., 0, 0, 0], dtype=int64)
```

● Measure performance (0.726)

Measure performance

```
from sklearn.metrics.cluster import adjusted_mutual_info_score
print("Adjusted Mutual Information: %0.3f" % adjusted_mutual_info_score(cluster_y,predict))
```

```
Adjusted Mutual Information: 0.726
```

● Visualize data & Visualize clusters (2D)

Visualize data & Visualize clusters (2D) ¶

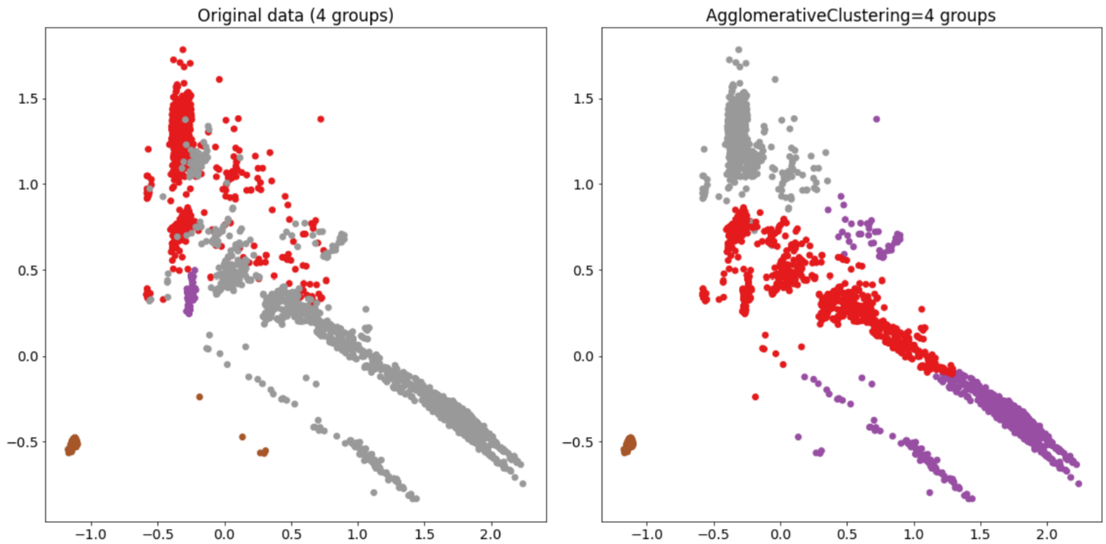
```
plt.rcParams['font.size'] = 14
plt.figure(figsize=(16, 8))
plt.subplot(121)
plt.title('Original data (4 groups)')
plt.scatter(data[:,0], data[:,1], c=cluster_y, cmap=plt.cm.Set1)

plt.subplot(122)
plt.title('AgglomerativeClustering=4 groups')
plt.scatter(data[:,0], data[:,1], c=predict, cmap=plt.cm.Set1)

plt.tight_layout()
plt.show()
```

● Visualize data & Visualize clusters (3D)

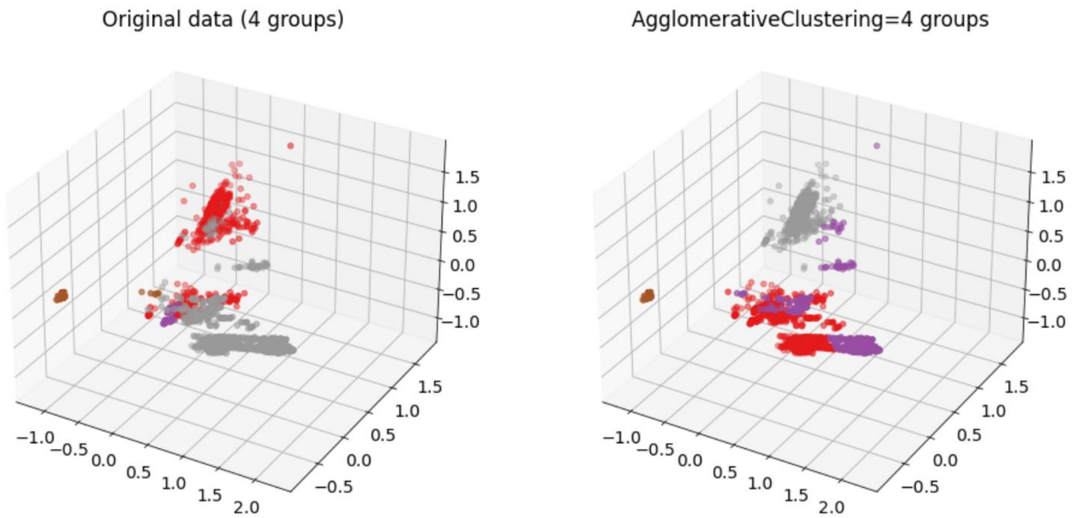
Visualize data & Visualize clusters (2D) ¶

```
plt.rcParams['font.size'] = 14
plt.figure(figsize=(16, 8))
plt.subplot(121)
plt.title('Original data (4 groups)')
plt.scatter(data[:,0], data[:,1], c=cluster_y, cmap=plt.cm.Set1)

plt.subplot(122)
plt.title('AgglomerativeClustering=4 groups')
plt.scatter(data[:,0], data[:,1], c=predict, cmap=plt.cm.Set1)

plt.tight_layout()
plt.show()
```

● 2-D


Original data (4 groups)
AgglomerativeClustering=4 groups

● 3-D


Original data (4 groups)
AgglomerativeClustering=4 groups

Discussions and Conclusion

在此 Case，我們採用了 3 種不同的演算法，分別是 K-means、Birch 以及 AgglomerativeClustering。

三者在 Measure performance 的部分為 0.72 至 0.74 左右， cluster 的成效約在 7 成左右。 在 K-means 中，我們採用 4 個 cluster 以及 random_state = 46，設定常數保證每次分群結果相同。在 BIRCH 中，我們藉由調整 threshold 以及 branching_factor 去調整 CT Tree 的規模。在 AgglomerativeClustering 中，我們給與 affinity 去調整距離的計算方式，以及下 linkage 調整群與群之間的距離。