MLFN Lab2 Report

資科工一 游庭瑋

1. Data processing:

   首先使用 pandas 讀取 csv 資料集，會發現有部分欄位為 ID 等不重複資料，因此必須先將其丟棄

```
data = raw_data.drop(["ID", "Flow.ID", "Timestamp"], axis=1)
data
```

✓ 0.5s

	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	Total.Length.of.Fwd.Packets	Total.Length.of.Bwd.Packets	...
0	10.200.7.196	39485	172.217.29.66	443	6	2021337	9	5	795	625	...
1	10.200.7.196	43024	179.1.4.244	443	6	65552	14	8	373	5252	...
2	10.200.7.196	43031	179.1.4.244	443	6	107032	14	12	373	10784	...
3	10.200.7.196	43064	179.1.4.244	443	6	75351	14	11	373	10784	...
4	10.200.7.196	43076	179.1.4.244	443	6	65862	15	13	373	11396	...
...	...	...	...	...	...	...	...	...	...	...	...
4312	10.200.7.7	59979	172.16.255.183	53	17	119040676	2146	2138	80152	234873	...
4313	10.200.7.7	59979	172.16.255.183	53	17	31408313	647	642	24421	65673	...
4314	10.200.7.9	48859	172.16.255.200	53	17	76350907	4	0	180	0	...
4315	10.200.7.9	48859	172.16.255.200	53	17	13621158	4	0	180	0	...
4316	10.200.7.9	48859	172.16.255.200	53	17	104320155	4	0	180	0	...

4317 rows × 83 columns

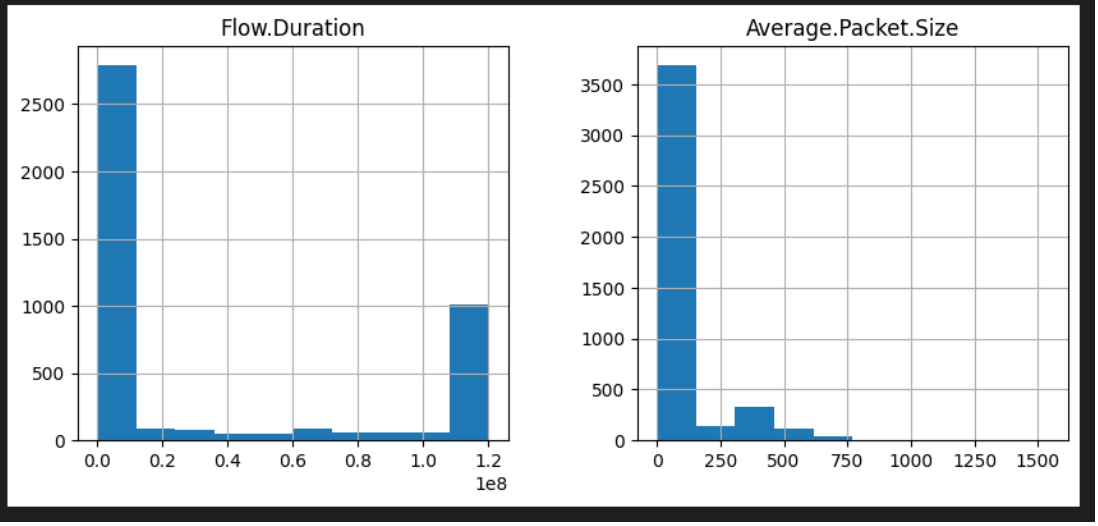
2. Data visualization:

   這裡選用一些比較顯著的特徵來視覺化

```
data[["Flow.Duration", "Average.Packet.Size"]].hist(figsize=(10, 4))
```

✓ 0.2s

```
array([[<AxesSubplot: title={'center': 'Flow.Duration'}>,
        <AxesSubplot: title={'center': 'Average.Packet.Size'}>]],
      dtype=object)
```

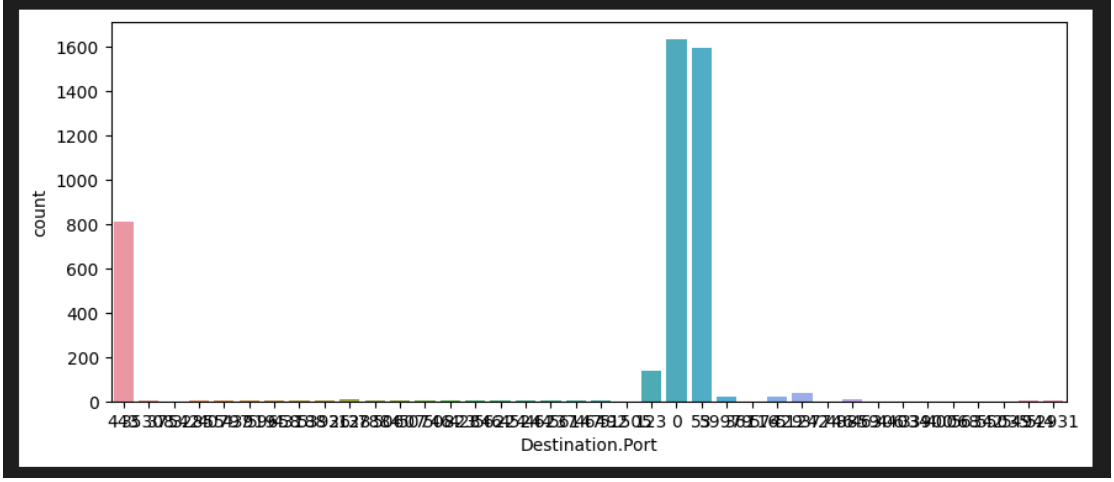


   從 Flow Duration 可以看出，絕大部分的 Flow 時間佔用都是比較少的，而從 average packet size 也可以看到，絕大多數的封包都小於 250bytes

```
import matplotlib.pyplot as plt
import seaborn as sns

_, axes = plt.subplots(figsize=(10, 4))

sns.countplot(x="Destination.Port", data=data, ax=axes)
```

✓ 0.2s

```
<AxesSubplot: xlabel='Destination.Port', ylabel='count'>
```



從 Destination Port 這邊則可以看出，大部分的 Flow 也都集中在特定的幾個 Port 上。並從 value_counts 函數可以看出，這些 Flow 主要的 port 如下:

```
display(data["Destination.Port"].value_counts())
```

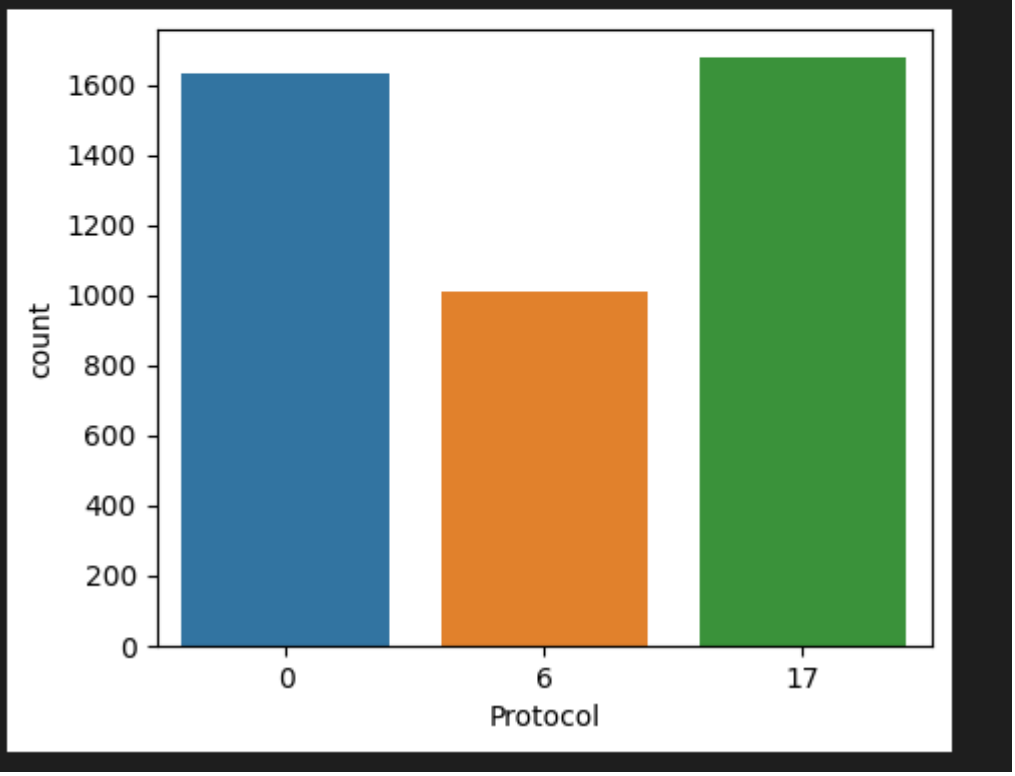
✓ 0.2s

```
Output exceeds the size limit. Open the full output data in a text editor
0        1631
53       1593
443       812
123       135
51242      40
57429      20
59979      18
48859       8
3128        7
54944       4
52931       4
46237       3
```

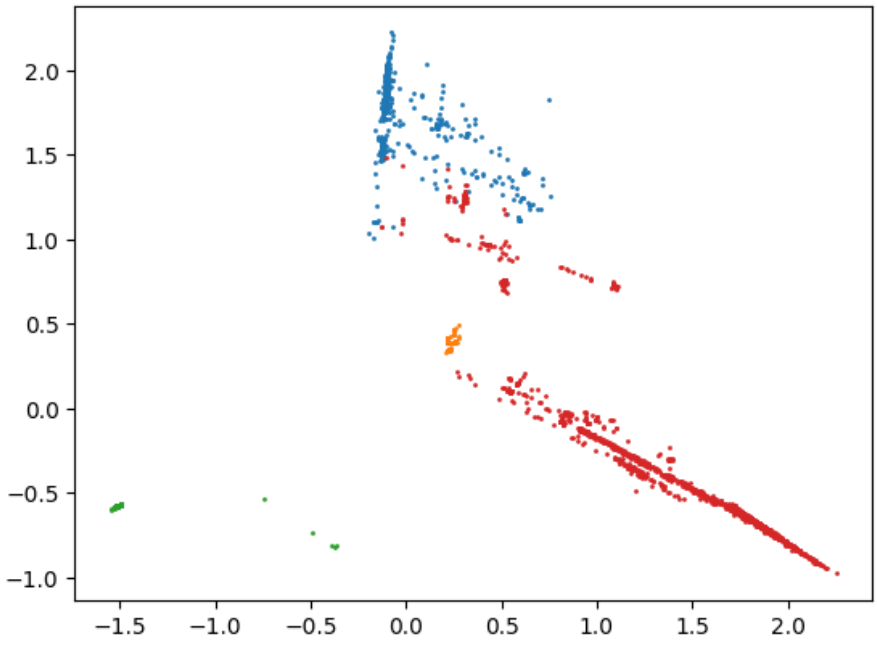
另外，就 Protocol 來統計則有以下結果

```
import matplotlib.pyplot as plt
import seaborn as sns

_, axes = plt.subplots(figsize=(5, 4))

sns.countplot(x="Protocol", data=data, ax=axes)
```

```
<AxesSubplot: xlabel='Protocol', ylabel='count'>
```



最後將所有 numerical attribute 正規化並將 nominal attribute 做 one-hot-encoding 後，使用 PCA 分析，將維度降至二維後比較 cluster.csv 中的正確答案可以得到下圖:



因此可以看出，此問題是可以被妥善分群的。

3. Feature engineering

在前面讀取資料時有發現到，有部分的 attribute 對於整個資料集的數據都相同，因此將有符合此特性的欄位皆刪除。留下共 70 個欄位。

```
for header in data.columns:
    if len(data[header].value_counts()) == 1:
        data = data.drop(header, axis=1)
data
```

✓ 0.5s

	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	Total.Length.of.Fwd.Packets	Total.Length.of.Bwd.Packets	...
0	10.200.7.196	39485	172.217.29.66	443	6	2021337	9	5	795	625	...
1	10.200.7.196	43024	179.1.4.244	443	6	65552	14	8	373	5252	...
2	10.200.7.196	43031	179.1.4.244	443	6	107032	14	12	373	10784	...
3	10.200.7.196	43064	179.1.4.244	443	6	75351	14	11	373	10784	...
4	10.200.7.196	43076	179.1.4.244	443	6	65862	15	13	373	11396	...
...	...	...	...	...	...	...	...	...	...	...	...
4312	10.200.7.7	59979	172.16.255.183	53	17	119040676	2146	2138	80152	234873	...
4313	10.200.7.7	59979	172.16.255.183	53	17	31408313	647	642	24421	65673	...
4314	10.200.7.9	48859	172.16.255.200	53	17	76350907	4	0	180	0	...
4315	10.200.7.9	48859	172.16.255.200	53	17	13621158	4	0	180	0	...
4316	10.200.7.9	48859	172.16.255.200	53	17	104320155	4	0	180	0	...

4317 rows × 70 columns

移除 IP、Label 等跟分群無關的資料後，對所有非數值資料做 one-hot-encding，目前共有 1177 個欄位。

```
nominal_col = ['Source.IP', 'Source.Port', 'Destination.IP', 'Destination.Port', 'Protocol', 'Label']

for col in nominal_col:
    try:
        data[col] = data[col].astype(str)
    except Exception as e:
        print(e)
pd.get_dummies(data)
```

[100] ✓ 0.5s

... 'Label'

Max	...	Destination.Port_55035	Destination.Port_56147	Destination.Port_56224	Destination.Port_56342	Destination.Port_57429	Destination.Port_59658	Destina
482	...	0	0	0	0	0	0	
436	...	0	0	0	0	0	0	
436	...	0	0	0	0	0	0	
660	...	0	0	0	0	0	0	
436	...	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
342	...	0	0	0	0	0	0	
331	...	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	

接著對丟棄所有非數值資料做 Min Max Scaling，方便後面分群的距離計算

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
tf_data = scaler.fit_transform(numeric_data)
tf_data = pd.DataFrame(tf_data, columns=numeric_data.columns)
tf_data
```

✓ 0.5s

	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	Total.Length.of.Fwd.Packets	Total.Length.of.Bwd.Packets	Fwd.Packet.Length.Max	Fwd.Pac
0	0.016844	0.001668	0.001715	0.002376	0.000131	0.035581	
1	0.000546	0.002711	0.002744	0.001115	0.001100	0.016293	
2	0.000892	0.002711	0.004117	0.001115	0.002258	0.016293	
3	0.000628	0.002711	0.003774	0.001115	0.002258	0.016293	
4	0.000549	0.002920	0.004460	0.001115	0.002386	0.016293	
...	...	...	...	...	...	...	
4312	0.992006	0.447341	0.733448	0.239550	0.049175	0.004822	
4313	0.261736	0.134724	0.220240	0.072987	0.013750	0.005041	
4314	0.636258	0.000626	0.000000	0.000538	0.000000	0.003288	
4315	0.113510	0.000626	0.000000	0.000538	0.000000	0.003288	
4316	0.869335	0.000626	0.000000	0.000538	0.000000	0.003288	

4317 rows × 65 columns

將以上正規化後的數值欄位及 one hot encoding 完的非數值欄位合併，即得到能夠直接分群用的資料集

```
for col in nominal_col:
    if data.get(col) is None: continue
    tf_data[col] = data[col].astype(str)
tf_data = pd.get_dummies(tf_data)
tf_raw_data = tf_data.copy()
tf_data
```

✓ 0.5s

	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packets	Total.Length.of.Fwd.Packets	Total.Length.of.Bwd.Pa
0	0.016844	0.001668	0.001715	0.002376	0.0
1	0.000546	0.002711	0.002744	0.001115	0.0
2	0.000892	0.002711	0.004117	0.001115	0.0
3	0.000628	0.002711	0.003774	0.001115	0.0
4	0.000549	0.002920	0.004460	0.001115	0.0
...	...	...	...	...	
4312	0.992006	0.447341	0.733448	0.239550	0.0
4313	0.261736	0.134724	0.220240	0.072987	0.0
4314	0.636258	0.000626	0.000000	0.000538	0.0
4315	0.113510	0.000626	0.000000	0.000538	0.0
4316	0.869335	0.000626	0.000000	0.000538	0.0

4317 rows × 1177 columns

同時匯入 cluster.csv 作為參考

```
ans = pd.read_csv("dataset/cluster.csv")
ans
```
✓ 0.2s

	ID	Cluster
0	1651	0
1	6460	0
2	6578	0
3	7219	0
4	7683	0
...	...	...
4312	3572701	3
4313	3572728	3
4314	3573244	3
4315	3573361	3
4316	3573425	3

4317 rows × 2 columns

```
ans["Cluster"].value_counts()
```
✓ 0.4s

```
3    1695
2    1631
0     856
1     135
Name: Cluster, dtype: int64
```

這裡使用 Iterative feature selection，每次加入一個特徵，並重新測量分群的準確度。最後得出使用 14 個欄位即可達到 91%的準確度。

```
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_mutual_info_score
from random import sample
accu = 0
best_list = []
best_score = []
for i in range(5):
    print(f"round {i}")
    selected = []
    columns = list(tf_data.columns.copy())
    columns = sample(columns, k=len(columns))
    for c in columns:
        selected.append(c)
        y_pred = KMeans(n_clusters=4).fit_predict(tf_data[selected])
        score = adjusted_mutual_info_score(ans["Cluster"], y_pred)
        if score > accu:
            accu = score
            print(accu)
        else:
            selected.remove(c)
    best_list.append(selected)
    best_score.append(score)

display(accu)
```
✓ 1m 28.6s

```
0.9182928870828644
```

```
display(best_list)
```
✓ 0.2s

```
[['Source.Port_59979',
  'Source.Port_50508',
  'Source.Port_39088',
  'Source.Port_35316',
  'Source.Port_39285',
  'Bwd.IAT.Max',
  'Source.Port_53534',
  'min_seg_size_forward',
  'Source.Port_38934',
  'Bwd.Packet.Length.Min',
  'Source.Port_50670',
  'Destination.Port_443',
  'Source.Port_0',
  'Destination.Port_53'],
 [],
 [],
 [],
 []]
```

4. Clustering:

先定義好用來繪製分群結果的函數

```
def draw_cluster(df: pd.DataFrame):
    lookup = {}
    for i in range(df.shape[0]):
        if lookup.get(df["cluster"][i]) is None:
            lookup[df["cluster"][i]] = [(df["x"][i], df["y"][i])]
        else:
            lookup[df["cluster"][i]].append((df["x"][i], df["y"][i]))
    for c, data in lookup.items():
        tmp = pd.DataFrame(data, columns=["x", "y"])
        plt.scatter(tmp["x"], tmp["y"], s=1)
```
✓ 0.2s

A. KMeans

```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4)
kmeans.fit(tf_data[selected])

cluster_labels = pd.DataFrame(kmeans.labels_, columns=["cluster"])
cluster_labels.value_counts()
```

✓ 0.6s

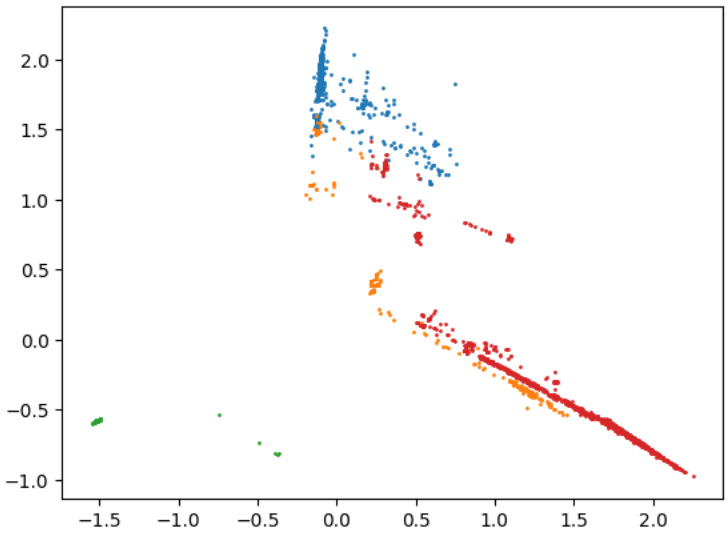
```
cluster
0          1631
1          1593
2           812
3           281
dtype: int64
```

```python
from sklearn.metrics import adjusted_mutual_info_score

adjusted_mutual_info_score(ans["Cluster"], cluster_labels['cluster'])
```

✓ 0.3s

```
0.9182928870828644
```

分群結果及視覺化



B. Gaussian Mixture

```python
from sklearn.mixture import GaussianMixture

gm = GaussianMixture(n_components=4)
gm.fit(tf_data[selected])
# display(db.predict(tf_data).shape)

_labels = pd.DataFrame(gm.predict(tf_data[selected]), columns=["cluster"])
_labels.value_counts()
```

✓ 0.6s

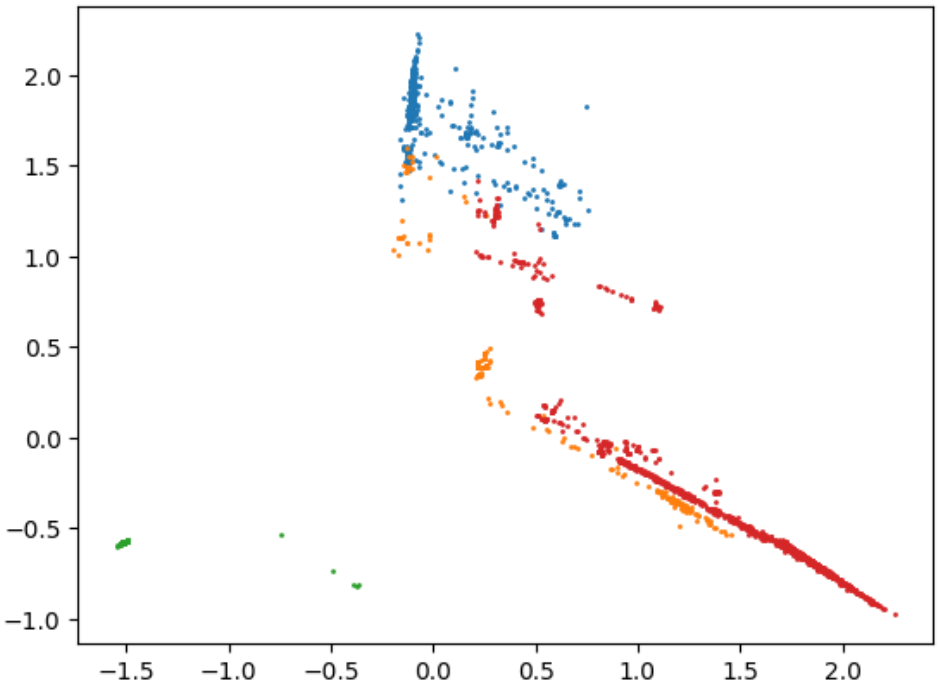
```
cluster
1          1631
2          1593
0           812
3           281
dtype: int64
```

```python
adjusted_mutual_info_score(ans["Cluster"], _labels["cluster"])
```

✓ 0.3s

```
0.9182928870828644
```

分析結果與視覺化



C. DBSCAN

DBSCAN

```python
from sklearn.cluster import DBSCAN
dbscan = DBSCAN()
y_pred = dbscan.fit_predict(tf_data[selected])
print(adjusted_mutual_info_score(ans["Cluster"], y_pred))
```

✓ 0.2s

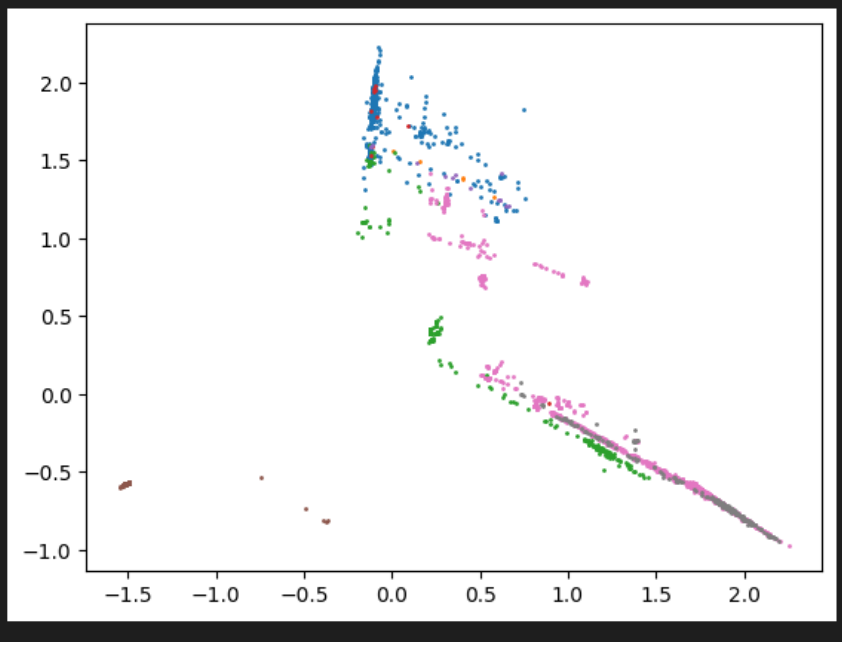
```
0.8491001310671713
```

分析結果與視覺化

```
    labels_ = pd.DataFrame(y_pred, columns=["cluster"])
    display(labels_.value_counts())
    _clustering = labels_.join(pd.DataFrame(reduced_data, columns=["x", "y"]))
    draw_cluster(_clustering)
```

✓ 0.2s

```
cluster
 4          1631
 5          1355
 0           784
 2           280
 6           237
 3            14
-1            11
 1             5
dtype: int64
```



可以看到，DBSCAN 由於不須指定 n_cluster，會有多分群的狀況，造成結果不盡理想，在稍後會繼續做參數調整。

5. Parameter adjustment

這裡直接調整 DBSCAN 的參數，定義 eps 介於[0.1, 2.0]，min_samples [2, 10]

grid search

```
    import numpy as np
    # (eps, min_sample)
    grid = [(a, b) for b in range(2, 11, 1) for a in np.arange(0.1, 2.1, 0.1)]
    best = {"score": 0, "param": tuple()}
    for p in grid:
        dbscan = DBSCAN(eps=p[0], min_samples=p[1])
        y_pred = dbscan.fit_predict(tf_data[selected])
        score = adjusted_mutual_info_score(ans["Cluster"], y_pred)
        if score > best["score"]:
            best["score"] = score
            best["param"] = p

    print(best["score"])
```

✓ 43.5s

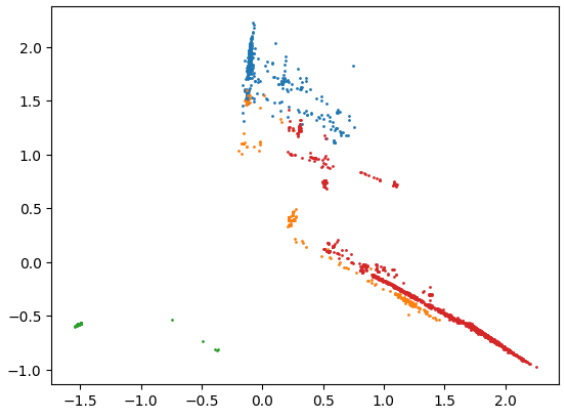
```
0.8518808915524809
```

搜尋最佳結果為 0.85，參數為 eps=0.7, min_samples=4

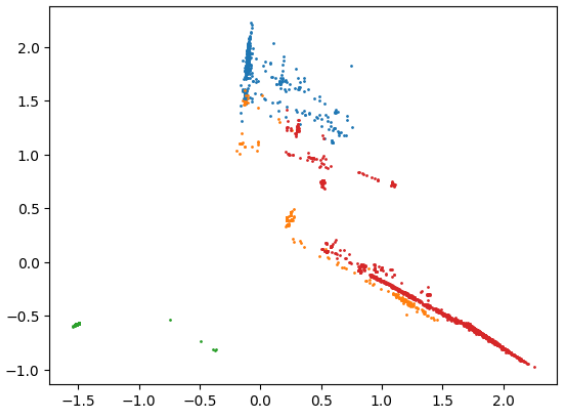
DBSCAN 的 eps 參數為資料點間的距離 Threshold，因此若 eps 太小時，可能會造成分出的群數太多，出現分數較低的狀況。

6. 分群視覺化

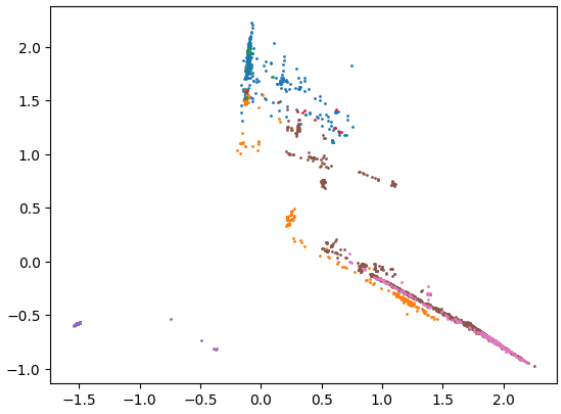
這裡直接比較三種分群方法及原本的資料



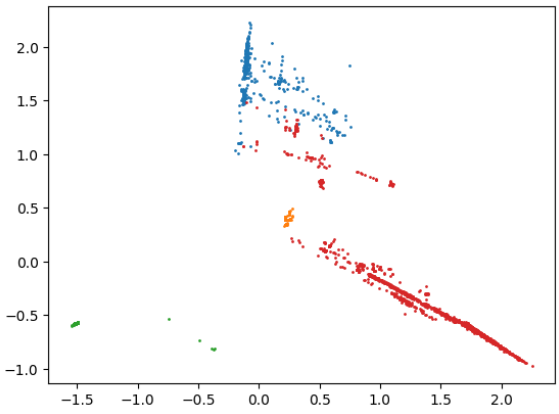
KMeans



GaussianMixture



DBSCAN



Raw data

7. By domain knowledge

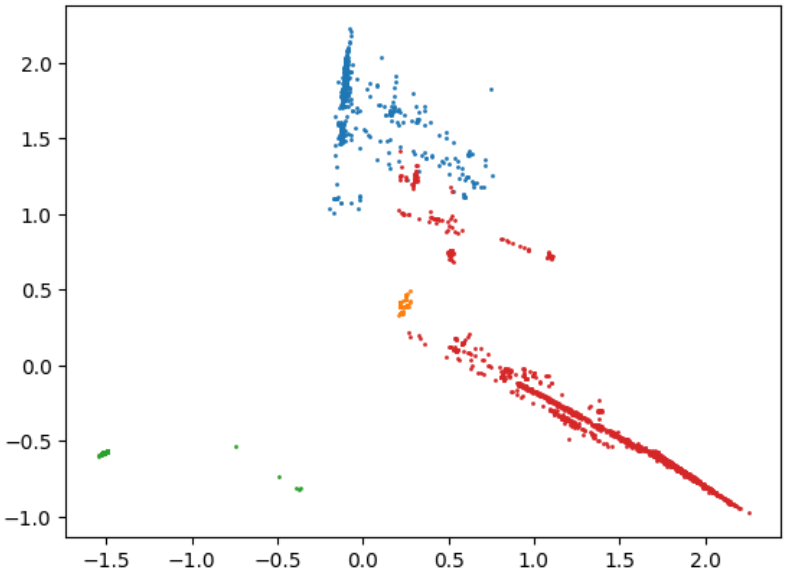
以 Flow 的分群來說，一般我們會最注重協定本身，這裡的協定泛指傳輸層協定及應用層協定。因為此資料集沒有提供應用層協定的資訊，所以這裡直接使用 Source.Port 及 Destination.Port 來判斷。

因此共有 3 種 attribute 可以使用，做完 one-hot-encoding 共 923 個欄位，並取得分群結果如下:

```
selected_feature = []
for a in tf_data_.columns:
    if a.find("Protocol")!=-1 or a.find("Destination.Port")!=-1 or a.find("Source.Port")!=-1:
        selected_feature.append(a)
y_pred = KMeans(n_clusters=4).fit_predict(tf_data_[selected_feature])
display(adjusted_mutual_info_score(ans["Cluster"], y_pred))
```

✓ 0.2s

```
0.9904195002627223
```



可以看到，直接以這三個 attribute 做分析即可得到 0.99 的分數，視覺化的結果也與原本的分群幾乎無異。

8. 結論與討論:

在這次的實驗中，我們發現:與分類一樣，分群依賴著大量的資料前處理過程，甚至因為沒有 Label 的關係，造成 Feature selection 的階段更難取得我們想要的特徵，這時如果就直接將所有資料丟到分群演算法中進行分群，就會造成不重要的特徵去影響重要的特徵，進而造成分群結果不理想。

由前面的 Feature selection 階段可以發現，在這次的分群的資料集中，只需要極少的特徵就可以達成較高的分群準度，並在後面的 Domain knowledge 階段發現，直接使用 Protocol, Source Port, Destination Port 就可以達到 99%的準度，說明了如果有 Domain knowledge 的幫助，可以更加方便的直接以人工提取 feature，並達到較高的準確度。