Data Science HW2

B10915030 陳奕軒

1. 引入會用到的 library

```
# Import essential library
import pandas as pd
import numpy as np

#EDA
import seaborn as sns
from matplotlib import pyplot as plt

#Pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

#Preprocess
from sklearn.preprocessing import OneHotEncoder, StandardScaler

#PCA
from sklearn.decomposition import PCA

#Model
from sklearn.cluster import KMeans
from sklearn import mixture
```

2. 資料前處理

   i. 資料載入

```
train = pd.read_csv('../input/2022-ntust-data-science-hw2/train.csv')
test = pd.read_csv('../input/2022-ntust-data-science-hw2/test_3000.csv')
```

   ii. 特徵選擇

```
train = train.drop(['song_id'], axis=1)
```

   X 為除了 song id 以外的所有 Feature

   iii. 特殊處理

```
train['Feature 11'] = train['Feature 11'].astype(str)
```

   Feature 11 為大調/小調，比起 numerical feature，更像是

   categorical feature（Encoder 後的結果）。

iv. 分類

```
# category column
categorical_cols = [cname for cname in train.columns
                    if train[cname].dtype == "object"]


# numerical columns
numerical_cols = [cname for cname in train.columns
                  if train[cname].dtype in ['int64', 'float64']]

print("Categorical columns:")
print(categorical_cols)

print("Numerical columns:")
print(numerical_cols)
```

```
Categorical columns:
['Feature 11', 'Feature 13']
Numerical columns:
['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5', 'Feature 6', 'Feature 7', 'Feature 8', 'Feature 9', 'Feature 10', 'Feature 12']
```

3. 定義前處理器

```
# 數值前處理
# Use scaler to norm
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# 分類前處理
# Using OH encoder
categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# 前處理綁在一起
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

+ Code + Markdown

```
train = preprocessor.fit_transform(train)
```

為了簡化流程和程式碼，所以採用 pipeline 做為前處理器。

- Numerical feature：使用 StandardScaler 將 data 轉成平均值和標準差。
- Categorical feature：使用 OneHotEncoder 將 categorical feature 轉換成多個 column，並忽略沒有看過的 value。

4. PCA

PCA 是一種將資料的維度減少的方法，它透過數學公式計算出資料的主成分，我們稱這些主成分具有解釋力。只要選出前幾名的主成分，它們可以代表和解釋整份資料，進而達到資料降維的效果。

i. 試算 PCA

```
pca = PCA()
pca.fit(train)
```

```
PCA()
```

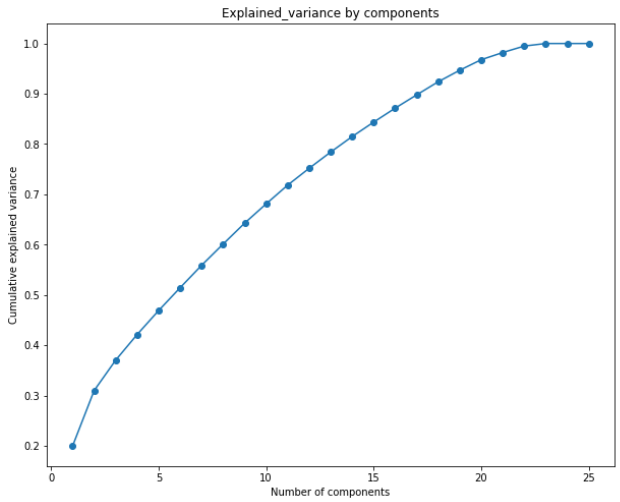
```
pca.explained_variance_ratio_
```

```
array([1.99803572e-01, 1.10039082e-01, 6.03934750e-02, 5.10628591e-02,
       4.76884446e-02, 4.54426806e-02, 4.45869009e-02, 4.22574669e-02,
       4.15748380e-02, 3.82926765e-02, 3.72729812e-02, 3.34742890e-02,
       3.22674266e-02, 3.08402221e-02, 2.87176995e-02, 2.78097909e-02,
       2.64100194e-02, 2.60075487e-02, 2.28992981e-02, 2.09832228e-02,
       1.42272870e-02, 1.28807985e-02, 5.06742003e-03, 5.26614280e-33,
       1.19335728e-33])
```

ii. 視覺化

```
plt.figure(figsize=(10, 8))
plt.plot(range(1, 26), pca.explained_variance_ratio_.cumsum(), marker='o')
plt.title('Explained_variance by components')
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

```
Text(0, 0.5, 'Cumulative explained variance')
```



iii. 決定 PCA 的 component

從圖中，我們可以觀察到 n_components=8 時，資料的 variance 已經來到 80%，已經足夠說明整份資料了

```
pca = PCA(n_components=8)
pca.fit(train)
train_pca = pca.transform(train)
```

5. 分群

- K-means

K-means 非常適合用在非監督學習(Unsupervised learning)的情境下，它的特色如他的名字，K 就是分成 K 群，而 Means 則代表群的均值，也可以叫做群心。有了群心的概念後，我們可以透過計算 data 和群心的距離（一般使用歐式距離），並將距離較近的放在一起。透過不斷的計算，群心也會一直變動，直到找到較為合理的分群結果（群心變動較小）為止。

```
kmeans= KMeans(n_clusters=3,  init='k-means++', random_state=1)
labels = kmeans.fit_predict(train_pca)
print(labels)
```

這裡還有用到 k-means++，比起 K-means 隨機選取 k 個點作為群心，K-means++會選擇距離已知群心較遠的點作為新的群心，例如：n+1 個群心會離 1~n 群心較遠。

- Gaussian mixture

Gaussian mixture 可以說是 K-means 的延伸，他將每個群都視

為不同的高斯分佈，並以機率來看說這個點應該屬於哪個群。他和 K-means 一樣需要先找到起始點（群心），找到之後進行一系列的運算，試圖找出最合適的高斯分佈。

```
model2 = mixture.GaussianMixture(n_components=3)
labels2 = model2.fit_predict(train_pca)
print(labels2)
```

6. 結果

i. 計算結果

```
songList1, songList2 = test['col_1'].tolist(), test['col_2'].tolist()

model_labels = [labels1, labels2]

result = []
for song1, song2 in zip(songList1, songList2):
    same = 0
    diff = 0
    for labels in model_labels:
        if labels[song1] == labels[song2] :
            same += 1
        else:
            diff += 1

    if diff > same:
        result.append(0)
    else:
        result.append(1)
```

最後，我們把兩個 model 的結果透過表決的方式組合起來，如果兩個 model 都輸出不同，則代表這兩首歌屬於不同類型，其餘只要有一個 model 輸出 true 才是相同類型。因為 model 分群涵蓋的邊界可能不太一樣，所以把兩個 model 取聯集，減少誤判。另外一個版本則是只有使用 k-means 作為輸出。

ii. 匯出

```
result_df = pd.DataFrame({'id': [id for id in range(len(result))],
                          'ans' : result})

result_df.to_csv('result.csv', index=False)
```