STAT 557 Data Mining Project 3

**Music Genre Clustering**

*Report written by Yuxuan Liu and Peicong He*

*December 4, 2021*

**Table of Content**

1. Introduction
2. EDA and Preprocess
   1. Feature Selection
   2. Sample Selection
   3. Other Preprocess
3. K-Means
   1. K-Means with fixed K
   2. K-Means with different K
4. Gaussian Mixture Model
5. Two-dimensional Visualization
6. Conclusion
7. **Introduction**

The dataset is about the music genre. The goal is to use the information about the song to predict which genre it belongs to. In this particular project, the goal is to use these features to perform unsupervised classification and use the target to determine the effectiveness of classification. We used Python to do this project. Peicong He is responsible for the EDA, preprocess, and K-means. Yuxuan Liu is responsible for the preprocess, Gaussion Mixture Model and 2-dimensional Visualization.

1. **EDA and Preprocessing**

The original music genre dataset contains about 50,000 songs, each described with 18 features. The 5 categorical features are ‘artist\_name’, ‘track\_name’, ‘key’, ‘mode’, ‘music\_genre’. Among them, ‘music\_genre’ is the target, which includes 10 genres like ‘Classical’, ‘Jazz’, ‘Hip-Hop’, ‘Country’, etc. Each genre has 5000 songs, which means it is balanced. The 13 continuous features are ‘popularity’, ‘danceability’, ‘tempo’, ‘instrumentalness’, etc.

**2.1 Feature Selection**

In the EDA, we found that the features ‘instance\_id’ and ‘obtained\_date’ are totally irrelevant information that is not helpful in classifying music genres. Also, ‘artist\_name’ and ‘track\_name’ should not be used to determine music genres. So, we drop them from the dataset.

* 1. **Sample Selection**

In the EDA, we found that there are no duplicated samples and the data is relatively clean. There are some invalid samples. For example, ‘duration\_ms’ = -1. In supervised learning, we should try to process it like replace with mean or use KNN because we want as many training samples as possible. However, in unsupervised learning, we do not need as many samples as possible. Instead, we want to keep significant and representative samples. Therefore, we can drop the invalid samples without concerns.

Another thing to notice is that in the original dataset, there are 10 genres. To simplify the problem and make the visualization more obvious, we selected 4 genres that are most different from each other. The 4 genres are ‘Anime’, ‘Classical’, ‘Hip-Hop’, and ‘Jazz’.

We also paid attention to outliers since outliers usually have a significant impact on unsupervised learning. Luckily, there are no obvious outliers.

* 1. **Other Preprocess**

We only have one categorical left after the feature selection. We simply used 0 and 1 to encode since it has only two values. We also standardized all features because standardized data are desired for unsupervised learning. After all preprocessing, the data has 16,241 rows and 12 features.

1. **K-Means**

**3.1 K-Means with fixed K**

First, we applied K-Means with fixed k=4 and 10 different random initializations. The parameter choice is as below:

|  |  |  |  |
| --- | --- | --- | --- |
| parameter | value | meaning | why |
| n\_components | 4 | Number of clusters | We know that there are 4 categories |
| init | ‘random’ | Initialization method | We want random initialization |
| max\_iter | 300 | Max number of iterations | Default. No other evidence shows that we need a larger value. |
| tol | 1e-4 | Tolerance to converge | Default |
| algorihm | ‘auto’ | K-means algorithm | Default |

Then calculate the adjusted rand indexes based on our classification result and true result. Below is the plot of the result:

图表, 折线图

描述已自动生成

We can see that in the 10 times, the adjusted rand indexes are slightly different. Overall, it has a mean of about 0.4.

* 1. **K-Means with different K**

Next, we applied K-Means with different K. We run 10 times for each K and calculate the average adjusted rand index for each K. Below is the plot of the result:

图表, 折线图

描述已自动生成

We can see that when K=4, the adjusted rand index is the highest, which means most consistent with the true classification result.

We also had the plot of mean square error at different K:

图表, 折线图

描述已自动生成

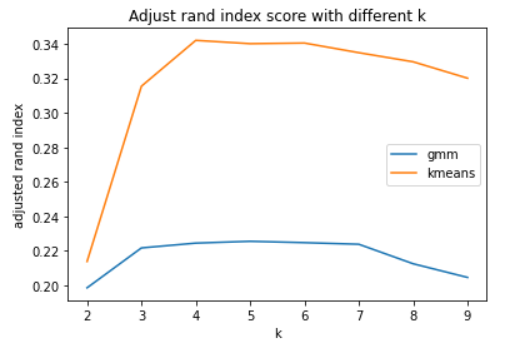
When we have no idea how many true categories exist, it is the plot to look at to determine how many clusters we want. Usually, we choose K at the turning point. However, there is no obvious turning point. Maybe we would choose K=3 based on the plot. No obvious turning point also means the data points are not obviously separated.

1. **Gaussian Mixture Model**

GMM is a probabilistic algorithm that uses soft clustering approach for distributing the points in different clusters. The advantages of GMM are that 1) it can analyze more complex and mixed data. 2) It can handle outliers better.

|  |  |  |  |
| --- | --- | --- | --- |
| parameter | value | meaning | why |
| n\_components | k | Number of clusters |  |
| covariance\_type | ‘full’ | Each component has its own general covariance matrix | default |
| tol | 1e-4 | Tolerance to converge | Use a smaller tolerance to converge better |
| max\_iter | 300 | Max number of iterations | Use a larger number to make sure convergence. |
| Init\_params | ‘kmeans’ | Initialize the weights, the means and the precisions using kmeans | Default. Kmeans initialization often yields good result. |

Similar to K-Means, we run 10 times for each K and calculated the average adjusted rand index. Below is the plot of the result:



From the plot, we can see that when K=5, the adjusted rand index is the highest. Also, notice that the highest rand index 0.235 is less than the highest rand index in K-means, which is 0.34. This means GMM performs not as well as K-means.

1. **Two-dimensional Visualization**

We first apply PCA with n\_components=2 to transform the data into two dimensions. By the PCA result, the first two principal components explain 47.42% of the total variance, which is very good. Then, we apply K-Means and GMM with k from 2 to 6. The cluster visualization is shown as below:

When k = 2:

图表, 散点图

描述已自动生成

From the scatter plot of actual clustering, we can see that although the data points are not completely separated, the clusters are relatively obvious. For example, the cluster of ‘Classical’ is far from the cluster of ‘Hip-Hop’, which makes sense in terms of music genres. There is no obvious difference between K-means and GMM when k = 2. The only difference is that the decision boundary of K-means is a straight line and the boundary of GMM is a curve. This is because GMM tends to group data into a bivariate normal distribution, which usually has an elliptical shape.

图表, 散点图

描述已自动生成

When k =3, K-means separates ‘Jazz’ into three parts and assigns to the remaining three clusters. While GMM combines ‘Anime’ and ‘Jazz’ into one cluster.

When k = 4:

图表, 散点图

描述已自动生成

These are the plots we should pay most attention to since we know that there are 4 clusters actually. When k = 4, the two algorithms yield similar clusters, both of which are close to the real situation. Therefore, we think both algorithms did good job.

图表, 散点图

描述已自动生成

When k = 5, both algorithms try to separate ‘Classical’ further. Again, one decision boundary of GMM is curved.

图表, 散点图

描述已自动生成

When k = 6, in both algorithms, ‘Hip-Hop’ is further split into two clusters. The difference is that K-means divides ‘Classical’ up and down, while GMM divides ‘Classical’ left and right.

Below is a comparison plot of adjusted rand index on two algorithms and different K.

图表, 折线图

描述已自动生成

We can see that both algorithms have similar adjusted rand indexes, which means there is no preference of which algorithm does better on this dataset. Moreover, the adjusted rand index is the highest when k = 4, which is ideal.

1. **Conclusion**

The dataset is very clean and easy to use. So, we did not spend too much time on preprocessing. For parameters, since it is unsupervised learning, we cannot do cross validation to determine whether the parameters are good. Therefore, we chose reasonable parameters based on our judgment. For the dataset before PCA, K-means has a better performance than GMM. The adjusted rand indexes are high when k = 4, 5, and 6. After PCA, both algorithms have similar adjusted rand index and relatively high accuracy. In this dataset, there are no obvious differences in the cluster results between K-means and GMM. We learned that GMM tends to cluster data points into elliptical shapes. It is actually quite interesting to see the scatter plot of how music genres differ from each other.

Python Code:

# Import packages

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

from sklearn.metrics import adjusted\_rand\_score

from pandas\_profiling import ProfileReport

from sklearn.preprocessing import StandardScaler, LabelEncoder

import imageio

from PIL import Image

import warnings

warnings.filterwarnings("ignore")

pd.set\_option('display.max\_columns', 50)

pd.set\_option('display.max\_rows', 100)

# EDA and preprocess

# drop some cols when read in

music\_df = pd.read\_csv("music\_genre.csv").drop(columns=['instance\_id', 'obtained\_date', 'artist\_name', 'track\_name', 'key'])

# select four genres to simplify

music\_df = music\_df[music\_df['music\_genre'].isin(['Anime', 'Classical', 'Hip-Hop', 'Jazz'])]

# generate report

report = ProfileReport(music\_df)

report

# drop invalid samples

drop\_index = music\_df[(music\_df['duration\_ms']==-1) | (music\_df['tempo']=='?')].index

music\_df.drop(index = drop\_index, inplace=True)

# Label encoding for music\_genre

le = LabelEncoder()

music\_df['music\_genre'] = le.fit\_transform(music\_df['music\_genre'])

music\_df['mode'] = le.fit\_transform(music\_df['mode'])

music\_df['tempo'] = np.round(music\_df['tempo'].astype(float), 3)

X = music\_df.drop(columns=['music\_genre'])

# Standardize all columns except for target

std = StandardScaler()

std\_X = pd.DataFrame(std.fit\_transform(X),

index = X.index,

columns = X.columns)

# K Means

# Question 1

# Kmeans with different initializations

score\_ls = []

for i in range(10):

kmeans = KMeans(n\_clusters = 4, init='random', random\_state=i, n\_init=1)

music\_df['pred'] = kmeans.fit\_predict(std\_X)

print("Number of iterations: ", kmeans.n\_iter\_)

score\_ls.append(round(adjusted\_rand\_score(music\_df['pred'], music\_df['music\_genre']), 4))

music\_df['pred'].value\_counts()

# Plot adjuested rand index score at different run

plt.plot(pd.Series(score\_ls))

plt.title("Adjust rand index score with different random initilizations in Kmeans")

plt.xlabel('run time')

plt.ylabel('adjusted rand index')

# Question 2

# In K-means, try k from 2 to 10, each with 10 random initializations

k\_scores = []

scores = []

mse = []

k\_mse = []

for k in range(2, 10):

for i in range(10):

kmeans = KMeans(n\_clusters = k, init='random', random\_state=i)

music\_df['pred'] = kmeans.fit\_predict(std\_X)

scores.append(round(adjusted\_rand\_score(music\_df['music\_genre'], music\_df['pred']), 4))

mse.append(round(kmeans.inertia\_))

k\_scores.append(np.mean(scores))

k\_mse.append(np.mean(mse))

# Plot adjuested rand index score

score\_df = pd.DataFrame({'k\_score':k\_scores, 'k':range(2,10)})

plt.plot(score\_df['k'], score\_df['k\_score'])

plt.title("Adjust rand index score with different k in Kmeans")

plt.xlabel('k')

plt.ylabel('adjusted rand index')

# Plot mean square error

mse\_df = pd.DataFrame({'mse':k\_mse, 'k':range(2,10)})

plt.plot(mse\_df['k'], mse\_df['mse'])

plt.title("Mean Square Error with different k in Kmeans")

plt.xlabel('k')

plt.ylabel('mse')

# Question 3

# Gaussian mixture model with k from 2 to 10

gaussian\_scores = []

scores = []

for k in range(2, 10):

for i in range(10):

gm = GaussianMixture(n\_components=k, init\_params='kmeans', tol=1e-4, max\_iter=300, random\_state=0)

music\_df['pred'] = gm.fit\_predict(std\_X)

scores.append(adjusted\_rand\_score(music\_df['music\_genre'], music\_df['pred']))

gaussian\_scores.append(np.mean(scores))

# Plot adjuested rand index score

score\_df['gmm\_score'] = gaussian\_scores

plt.plot(score\_df['k'], score\_df['gmm\_score'], label='gmm')

plt.plot(score\_df['k'], score\_df['k\_score'], label='kmeans')

plt.title("Adjust rand index score with different k")

plt.xlabel('k')

plt.ylabel('adjusted rand index')

plt.legend()

# Question 4

# Use PCA to reduce dimension to 2

pca = PCA(n\_components=2)

pca\_X = pd.DataFrame(data=pca.fit\_transform(std\_X), columns=['pc1', 'pc2'])

print('Total variance explained: ', round(pca.explained\_variance\_ratio\_.sum(), 4))

pca\_X['true'] = music\_df['music\_genre'].values

# Scatter plots of the 2-domensional data

fig, axs = plt.subplots(5, 3, figsize=(15,25))

fig.suptitle("Scatter plots with different k")

pca\_X['true'] = music\_df['music\_genre'].values

scores\_df = pd.DataFrame(columns=['k', 'kmeans', 'gm'])

for k in range(2, 7):

kmeans = KMeans(n\_clusters = k, init='random')

pca\_X['kmeans\_pred'] = kmeans.fit\_predict(pca\_X[['pc1', 'pc2']])

gm = GaussianMixture(n\_components = k)

pca\_X['gaussian\_pred'] = gm.fit\_predict(pca\_X[['pc1', 'pc2']])

kmeans\_score = adjusted\_rand\_score(pca\_X['kmeans\_pred'], pca\_X['true'])

gm\_score = adjusted\_rand\_score(pca\_X['gaussian\_pred'], pca\_X['true'])

scores\_df.loc[k-2] = [k, kmeans\_score, gm\_score]

kmeans\_centroids = kmeans.cluster\_centers\_

gmm\_centroids = np.array([[pca\_X.loc[pca\_X['gaussian\_pred']==z, 'pc1'].mean(),

pca\_X.loc[pca\_X['gaussian\_pred']==z, 'pc2'].mean()] for z in range(k)])

true\_centroids = np.array([[pca\_X.loc[pca\_X['true']==z, 'pc1'].mean(),

pca\_X.loc[pca\_X['true']==z, 'pc2'].mean()] for z in range(4)])

music\_genre = ['Anime', 'Classical', 'Hip-Hop', 'Jazz']

for i in range(k):

axs[k-2, 0].scatter(pca\_X.loc[pca\_X['kmeans\_pred']==i, 'pc1'], pca\_X.loc[pca\_X['kmeans\_pred']==i, 'pc2'], s = 0.2)

axs[k-2, 0].scatter(kmeans\_centroids[:, 0], kmeans\_centroids[:, 1], marker="^", color='black')

axs[k-2, 0].set\_title("K-means")

axs[k-2, 1].scatter(pca\_X.loc[pca\_X['gaussian\_pred']==i, 'pc1'], pca\_X.loc[pca\_X['gaussian\_pred']==i, 'pc2'], s = 0.2)

axs[k-2, 1].scatter(gmm\_centroids[:, 0], gmm\_centroids[:, 1], marker="^", color='black')

axs[k-2, 1].set\_title("Gaussian Mixture Model")

for j in range(4):

axs[k-2, 2].scatter(pca\_X.loc[pca\_X['true']==j, 'pc1'], pca\_X.loc[pca\_X['true']==j, 'pc2'], s = 0.2, label = music\_genre[j])

axs[k-2, 2].scatter(true\_centroids[:, 0], true\_centroids[:, 1], marker="^", color='black')

axs[k-2, 2].set\_title("Actual")

axs[k-2, 2].legend(markerscale=10, loc='upper right')

plt.show()

# Plot ARI score and compare K-means and GMM

plt.figure(figsize=(8,5))

plt.plot(scores\_df['k'], scores\_df['kmeans'], label='kmeans')

plt.plot(scores\_df['k'], scores\_df['gm'], label='gmm')

plt.title("Adjust rand index score with different k on two algorithms")

plt.xlabel('k')

plt.ylabel('adjusted rand index')

plt.legend()

# Plot the iterations

max\_iter\_ls = [i+1 for i in range(10)]

for max\_iter in max\_iter\_ls:

kmeans = KMeans(n\_clusters = 4, max\_iter=max\_iter, random\_state=0,

init = np.array([[-3, 0], [-1, 0], [1, 0], [3, 0]]))

pca\_X['kmeans\_pred'] = kmeans.fit\_predict(pca\_X[['pc1', 'pc2']])

kmeans\_centroids = kmeans.cluster\_centers\_

color = ['g', 'r', 'royalblue', 'darkorange']

for i in range(4):

plt.scatter(pca\_X.loc[pca\_X['kmeans\_pred']==i, 'pc1'], pca\_X.loc[pca\_X['kmeans\_pred']==i, 'pc2'], s = 0.2, color = color[i])

plt.scatter(kmeans\_centroids[:, 0], kmeans\_centroids[:, 1], marker="^", color='black')

plt.title(f"iteration\_{max\_iter}")

plt.savefig(f'./iter{max\_iter}.jpg')

plt.show()

# Generate a gif

image\_path = [f'iter{i+1}.jpg' for i in range(10)]

gif\_images = []

for path in image\_path:

gif\_images.append(imageio.imread(path))

imageio.mimsave("kmeans.gif", gif\_images,fps=2)

im = Image.open('kmeans.gif')

im