homework 8

(finally)

Question 1

apply k-means and hierarchical clustering to the ORLface dataset. set k=2 in k-means and select 2 clusters in hierarchical clustering. do the clustering results match the two gender?

First, we load the dataset.

```
import numpy as np
from PIL import Image

data = []
for i in range(1, 41):
    for j in range(1,11):
        image_dir = f"C:/Users/user/Desktop/ORL faces/{i}_{j}.png"
        img = Image.open(image_dir)
        img_array = np.asarray(img)
        data.append(img_array.flatten())
data = np.array(data)
print(data.shape)
```

```
(400, 2576)
```

then, we use k-means to proceed data. We assume that the bigger group is group 1(man), and the smaller group is group 0 (woman).

```
from sklearn.cluster import KMeans, AgglomerativeClustering
kmeans = KMeans(n_clusters=2, random_state=622)
kmeans.fit(data)
kmeans_labels = kmeans.labels_
count0 = list(kmeans_labels).count(0)
count1 = list(kmeans_labels).count(1)
if count0 > count1:
    num = 0
    for i in kmeans.labels:
        if i == 0:
            kmeans.labels[num] = 1
        else:
            kmeans.labels[num] = 0
        num += 1

count0 = list(kmeans_labels).count(0)
count1 = list(kmeans_labels).count(1)
print(count0)
print(count1)
```

```
182
218
```

We can see the labels in 0 and 1 are balance, which are not we expected, but OK. Then, we use hierarchical clustering and do the same thing.

```
hierarchical = AgglomerativeClustering(n_clusters=2)
hierarchical.fit(data)
hierarchical_labels = hierarchical.labels_

count0 = list(hierarchical_labels).count(0)
count1 = list(hierarchical_labels).count(1)
if count0 > count1:
    num = 0
    for i in hierarchical_labels:
        if i == 0:
            hierarchical_labels[num] = 1
        else:
            hierarchical_labels[num] = 0
        num += 1
count0 = list(hierarchical_labels).count(0)
count1 = list(hierarchical_labels).count(1)
print(count0)
print(count1)
```

```
195
205
```

We found that the label 0 and label 1 are balanced, too. Let apply the trye labels and see the result of some indexes.

```
gender = 
[0,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1
,1,1,1,1,1]
true_labels = []
for i in gender:
    for _ in range(10):
        true_labels.append(i)
true_labels = np.array(true_labels)

from sklearn.metrics import adjusted_rand_score, 
normalized_mutual_info_score
# Calculate the evaluation metrics for k-means clustering
kmeans_ari = adjusted_rand_score(true_labels, kmeans_labels)
kmeans_nmi = normalized_mutual_info_score(true_labels, kmeans_labels)

# Calculate the evaluation metrics for hierarchical clustering
hierarchical_ari = adjusted_rand_score(true_labels, 
hierarchical_labels)
hierarchical_nmi = normalized_mutual_info_score(true_labels, 
hierarchical_labels)
```

```python
print("Evaluation metrics for K-means Clustering:")
print("ARI: {:.4f}".format(kmeans_ari))
print("NMI: {:.4f}".format(kmeans_nmi))
print()

print("Evaluation metrics for Hierarchical Clustering:")
print("ARI: {:.4f}".format(hierarchical_ari))
print("NMI: {:.4f}".format(hierarchical_nmi))
```

```
Evaluation metrics for K-means Clustering:
ARI: -0.0021
NMI: 0.0378

Evaluation metrics for Hierarchical Clustering:
ARI: -0.0002
NMI: 0.0062
```

The Adjusted Rand Index (ARI) measures the similarity between two clusterings, taking into account all pairs of samples and their labels. It returns a value between -1 and 1, where a higher value indicates better agreement between the clustering and true labels. The Normalized Mutual Information (NMI) measures the mutual information between two clusterings, normalized by the entropy of the clusterings. It also returns a value between 0 and 1, with a higher value indicating better agreement.

By comparing the ARI and NMI values for both k-means clustering and hierarchical clustering, you can assess their performance and determine which method gives better results for your dataset.

Question 2

drop the origin variable from AUTOMPG and apply k-means. hierarchical clustering, and DBSCAN to the AUTOMPG dataset. Check if the clustering result match the origin.

```python
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.preprocessing import StandardScaler
data = pd.read_csv("C:/Users/user/Desktop/autompg.csv")
data.replace('?', np.nan, inplace=True)
data.dropna(inplace=True)

X = data.iloc[:, :-2] # -1 is car_name
y = data.iloc[:, -2]

print("no. of 1: ", list(y).count(1))
print("no. of 2: ", list(y).count(2))
print("no. of 3: ", list(y).count(3))
```

```
no. of 1:  245
no. of 2:  68
no. of 3:  79
```

We first apply StandardScaler, and fit three models.

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3)
kmeans_labels = kmeans.fit_predict(scaled_data)

hierarchical = AgglomerativeClustering(n_clusters=3)
hierarchical_labels = hierarchical.fit_predict(scaled_data)

dbscan = DBSCAN(eps=0.8, min_samples=5)
dbscan_labels = dbscan.fit_predict(scaled_data)
```

I want to decide which labels the models gave are american cars, japanese cars, and so on. Therefore, I test six combinations and see which are the most possible combinations. We use accuracy scores to decide how to sequence the datas.

```
from sklearn.metrics import accuracy_score

# 利用 accuracy  scores 來決定 123 怎麼排
k_means_best_accuracy = 0
k_means_best_mapping = None

for mapping in [{0: 1, 1: 2, 2: 3}, {0: 1, 1: 3, 2: 2}, {0: 2, 1: 1, 
2: 3},{0: 2, 1: 3, 2: 1},{0: 3, 1: 1, 2: 2},{0: 3, 1: 2, 2: 1}]:
    kmeans_predicted_labels = np.array([mapping[label] for label in 
kmeans_labels])
    kmeans_accuracy = accuracy_score(y, kmeans_predicted_labels)
    if kmeans_accuracy > k_means_best_accuracy:
        k_means_best_accuracy = kmeans_accuracy
        k_means_best_mapping = mapping

kmeans_predicted_labels = np.array([k_means_best_mapping[label] for 
label in kmeans_labels])
```

We use the same notion to map the hierarchical, too.

```
hierarchical_best_accuracy = 0
hierarchical_best_mapping = None

for mapping in [{0: 1, 1: 2, 2: 3}, {0: 1, 1: 3, 2: 2}, {0: 2, 1: 1, 
2: 3},{0: 2, 1: 3, 2: 1},{0: 3, 1: 1, 2: 2},{0: 3, 1: 2, 2: 1}]:
    hierarchical_predicted_labels = np.array([mapping[label] for label 
in hierarchical_labels])
    hierarchical_accuracy = accuracy_score(y,
```

```
hierarchical_predicted_labels)
    if hierarchical_accuracy > hierarchical_best_accuracy:
        hierarchical_best_accuracy = hierarchical_accuracy
        hierarchical_best_mapping = mapping

hierarchical_predicted_labels =
np.array([hierarchical_best_mapping[label] for label in
hierarchical_labels])
# print(hierarchical_predicted_labels)
```

We use the same notion to map the DBSCAN, too. IF the dbscan model give use -1 or 3, we alter them too 1, which are the biggest part of the origin of AUTOmpg dataset.

```
dbscan_best_accuracy = 0
dbscan_best_mapping = None

for mapping in [{-1: 1,3: 1, 0: 1, 1: 2, 2: 3}, {-1: 1,3: 1, 0: 1, 1:
3, 2: 2}, {-1: 1,3: 1, 0: 2, 1: 1, 2: 3},{-1: 1,3: 1, 0: 2, 1: 3, 2:
1},{-1: 1,3: 1, 0: 3, 1: 1, 2: 2},{-1: 1,3: 1, 0: 3, 1: 2, 2: 1}]:
    dbscan_predicted_labels = np.array([mapping[label] for label in
dbscan_labels])
    dbscan_accuracy = accuracy_score(y, dbscan_predicted_labels)
    if dbscan_accuracy > dbscan_best_accuracy:
        dbscan_best_accuracy = dbscan_accuracy
        dbscan_best_mapping = mapping

dbscan_predicted_labels = np.array([dbscan_best_mapping[label] for
label in dbscan_labels])
# print(dbscan_predicted_labels)
```

We now see the accuracy score, and we can see all of them perform very bad.

```
print("k_means_best_accuracy", k_means_best_accuracy)
print("hierarchical_best_accuracy", hierarchical_best_accuracy)
print("dbscan_best_accuracy", dbscan_best_accuracy)
```

```
k_means_best_accuracy 0.45153061224489793
hierarchical_best_accuracy 0.45408163265306123
dbscan_best_accuracy 0.4770408163265306
```

I make a dataframe to see how the labels are distributed.

```
dic = {"true":y, "kmeans":kmeans_predicted_labels,
"hierarchical":hierarchical_predicted_labels,
"dbscan":dbscan_predicted_labels}
comparison = pd.DataFrame(dic)
print(comparison)
```

```
     true  kmeans  hierarchical  dbscan
0       1       1             1       1
1       1       1             1       1
```

```
2        1        1               1        1
3        1        1               1        1
4        1        1               1        1
..     ...      ...             ...      ...
393      1        3               3        3
394      2        3               3        1
395      1        3               3        3
396      1        3               3        3
397      1        3               3        3

[392 rows x 4 columns]
```

When we compare these result with supervised models, we can see our predictions are not precise at all in unsupervised learning methods. Here are some possible reason.

1. Labeled Data: Supervised learning relies on labeled data, which provides explicit guidance to the model, allowing it to learn patterns and relationships more effectively.

2. Exploiting Known Patterns: Supervised learning models are explicitly trained to recognize and exploit the patterns present in the labeled data, leading to more accurate predictions.

3. Evaluating Model Performance: Supervised learning methods can be evaluated using metrics that provide quantitative measures of performance, facilitating iterative improvement.

4. Task-Specific Optimization: Supervised learning methods are designed to solve specific tasks, allowing for fine-tuning and better performance optimization.

5. Bias Reduction: Supervised learning with diverse labeled data helps mitigate biases and improves generalization across different instances.

I think the reason 1 and 2 are the most important reasons for the result we observed.