**杭州电子科技大学计算机学院**

**数据挖掘**

**上机实验3：聚类分析**

时间：2024年12月24日，学号：22270336 姓名：成家康

**注意：**

**1）实验开始前，填写学号和姓名。**

**2）将文件名由“聚类分析”更改为“学号+姓名+聚类分析”。**

**3）作业做完之后提交至泛雅平台：上机3**

# 一、实验目的

1、理解聚类的一般过程和基本原理；

2、巩固聚类算法的算法思想，能够进行聚类操作；

3、学会聚类中的性能评估方法。

# 二、实验原理

**1、常用的聚类算法：K-Means、K-Mediods、凝聚层次聚类和 DBSCAN算法等。**

（1） K-Means(K均值)聚类

算法步骤：

(1) 首先我们选择一些类/组，并随机初始化它们各自的中心点。中心点是与每个数据点向量长度相同的位置。这需要我们提前预知类的数量(即中心点的数量)。

(2) 计算每个数据点到中心点的距离，数据点距离哪个中心点最近就划分到哪一类中。

(3) 计算每一类中中心点作为新的中心点。

(4) 重复以上步骤，直到每一类中心在每次迭代后变化不大为止。也可以多次随机初始化中心点，然后选择运行结果最好的一个。

优点：

速度快，计算简便

缺点：

必须提前知道数据有多少类/组。

（2）K-Mediods

k-均值算法采用簇的质心来代表一个簇，质心是簇中其他对象的参照点。因此，k-均值算法对孤立点是敏感的，如果具有极大值，就可能大幅度地扭曲数据的分布。

k-中心点算法是为消除这种敏感性提出的，它选择簇中位置最接近簇中心的对象（称为中心点）作为簇的代表点，目标函数仍然可以采用平方误差准则。

处理过程：首先，随机选择k个对象作为初始的k个簇的代表点，将其余对象根据其与代表点对象的距离分配到最近的簇； 然后，反复用非代表点来代替代表点，以改进聚类质量，聚类质量用一个代价函数来估计，该函数度量对象与代表点对象之间的平均相异度。

**输入:**n个对象的数据库，期望得到的簇的数目k

**输出:**使得所有对象与其最近中心点的偏差总和最小化的k个簇

**方法：**

选择k个对象作为初始的簇中心

repeat

对每个对象，计算离其最近的簇中心点，并将对象分配到该中心点代表的簇

随机选取非中心点Orandom

计算用Orandom 代替Oj 形成新集合的总代价S

如果S<0，用Orandom代替Oj，形成新的k个中心点的集合

until 不再发生变化

采用k-中心点算法有两个好处：

对属性类型没有局限性；

通过簇内主要点的位置来确定选择中心点，对孤立点的敏感性小

不足：

处理时间要比k-mean更长；

用户事先指定所需聚类簇个数k。

（3）凝聚层次聚类

层次聚类算法分为两类：自上而下和自下而上。凝聚层级聚类(HAC)是自下而上的一种聚类算法。HAC首先将每个数据点视为一个单一的簇，然后计算所有簇之间的距离来合并簇，知道所有的簇聚合成为一个簇为止。

具体步骤：

(1) 首先将每个数据点视为一个单一的簇，然后选择一个测量两个簇之间距离的度量标准。例如使用average linkage作为标准，它将两个簇之间的距离定义为第一个簇中的数据点与第二个簇中的数据点之间的平均距离。

(2) 在每次迭代中，将两个具有最小average linkage的簇合并成为一个簇。

(3) 重复步骤2知道所有的数据点合并成一个簇，然后选择需要多少个簇。

层次聚类优点：（1）不需要知道有多少个簇

（2）对于距离度量标准的选择并不敏感

缺点：效率低

（4）基于密度的聚类方法(DBSCAN)

具体步骤：

(1) 首先确定半径r和minPoints. 从一个没有被访问过的任意数据点开始，以这个点为中心，r为半径的圆内包含的点的数量是否大于或等于minPoints，如果大于或等于minPoints则改点被标记为central point,反之则会被标记为noise point。

(2) 重复1的步骤，如果一个noise point存在于某个central point为半径的圆内，则这个点被标记为边缘点，反之仍为noise point。重复步骤1，知道所有的点都被访问过。

优点：不需要知道簇的数量

缺点：需要确定距离r和minPoints

**2、 评估方法**

**（1）外在方法**

TP、True Positive   真阳性：预测为正，实际也为正

FP、False Positive  假阳性：预测为正，实际为负

FN、False Negative 假阴性：预测为负、实际为正

TN、True Negative 真阴性：预测为负、实际也为负。

例如：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | 总计 |
| TP | 2 | 2 | 1 | 5 |
| FP | 0 | 2 | 1 | 3 |
| FN | 2 | 1 | 1 | 4 |

**精确率**：

Precison=TP/(TP+FP)

**召回率**：

Recall=TP/(TP+FN)

**F1分数**（F1-Score），又称为平衡F分数（BalancedScore），它被定义为精确率和召回率的调和平均数。

**F1=（2\*** Precison\* Recall）/**（**Precison+Recall）

**Micro-F1和Macro-F1**

假设已经通过聚类算法，将待分类数据进行了聚类。对每个“类”，计算F1，然后所有类的F1合并起来考虑。这里有两种合并方式：

第一种计算出所有类别总的Precision和Recall，然后计算F1。

例如依照最上面的表格来计算: Precison=5/(5+3)=0.625,Recall=5/(5+4)=0.556，然后代入F1的公式求出F1，这种方式被称为**Micro-F1**微平均。

第二种方式是计算出每一个类的Precison和Recall后计算F1，最后将F1平均。

例如上式A类：P=2/(2+0)=1.0，R=2/(2+2)=0.5，F1=(2\*1\*0.5)/(1+0.5)=0.667。同理求出B类C类的F1，最后求平均值，这种方式叫做**Macro-F1**宏平均。

**（2）内在方法**

**轮廓系数：**

假设已经通过聚类算法，将待分类数据进行了聚类。

对于簇中的每个向量。分别计算它们的轮廓系数。对于其中的一个点 i 来说：

计算 a(i) = average(i向量到所有它属于的簇中其它点的距离)

计算 b(i) = min (i向量到与它相邻最近的一簇内的所有点的平均距离) 即：i到不包含i的所有簇的最小平均距离。

那么 i 向量轮廓系数就为：

https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/pic/item/4610b912c8fcc3ce5691cc0f9745d688d43f20bc.jpg

可见轮廓系数的值是介于 [-1,1] ，越趋近于1代表内聚度和分离度都相对较优。

**将所有点的轮廓系数求平均，就是该聚类结果总的轮廓系数**。

a(i) ：i向量到同一簇内其他点**不相似程度**的平均值

b(i) ：i向量到其他簇的平均**不相似程度**的最小值

# 三、实验内容：

从**K-means、层次聚类、DBSCAN中选用2种聚类方法对wine数据集进行聚类分析，** 对得到聚类结果采用**Micro-F1**、**Macro-F1、轮廓系数**作为评估指标进行评估。

# 四、实验步骤

（ 文字+图 分别说明步骤，配主要代码及注释）

**1.数据读取**

从文件 wine-data.txt 中读取数据，假设第一列为分类标签，其余列为特征值。真实标签需要减 1，以确保标签从 0 开始，与模型输出的聚类结果一致。

# 1. 读取数据

data = pd.read\_csv("wine-data.txt", header=None)

labels = data.iloc[:, 0] # 提取真实标签

features = data.iloc[:, 1:] # 提取特征

labels -= 1

**2.数据标准化**

使用 StandardScaler 对特征进行标准化处理，使数据均值为 0，方差为 1，有助于提高聚类模型的性能。

# 2. 数据标准化

scaler = StandardScaler()

features\_scaled = scaler.fit\_transform(features)

**3.K-Means 聚类**

设置聚类的簇数为 3（假设已知类别数量为 3），随机种子为 42。对标准化后的特征进行 K-Means 聚类，生成聚类标签。

# 3. K-Means 聚类

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans\_labels = kmeans.fit\_predict(features\_scaled)

**4.层次聚类**

使用层次聚类中的 Ward 方法，设置簇数为 3。对标准化后的数据进行聚类，生成聚类标签。

# 4. 层次聚类

hierarchical = AgglomerativeClustering(n\_clusters=3, linkage='ward')

hierarchical\_labels = hierarchical.fit\_predict(features\_scaled)

**5.评估聚类结果**

F1 分数：衡量预测结果和真实标签的一致性，使用微平均（micro）和宏平均（macro）两种指标。轮廓系数：反映数据聚类的紧密程度和分离程度。输出两个模型的评估结果，便于比较性能。

# 5. 计算评估指标

def evaluate\_clustering(true\_labels, predicted\_labels, features):

# F1 分数

micro\_f1 = f1\_score(true\_labels, predicted\_labels, average='micro')

macro\_f1 = f1\_score(true\_labels, predicted\_labels, average='macro')

# 轮廓系数

silhouette = silhouette\_score(features, predicted\_labels)

return micro\_f1, macro\_f1, silhouette

# 评估 K-Means

kmeans\_micro\_f1, kmeans\_macro\_f1, kmeans\_silhouette = evaluate\_clustering(

labels, kmeans\_labels, features\_scaled

)

# 评估层次聚类

hierarchical\_micro\_f1, hierarchical\_macro\_f1, hierarchical\_silhouette = evaluate\_clustering(

labels, hierarchical\_labels, features\_scaled

)

# 6. 输出结果

print("K-Means 评估结果:")

print(f"Micro-F1: {kmeans\_micro\_f1:.4f}, Macro-F1: {kmeans\_macro\_f1:.4f}, 轮廓系数: {kmeans\_silhouette:.4f}")

print("\n层次聚类评估结果:")

print(f"Micro-F1: {hierarchical\_micro\_f1:.4f}, Macro-F1: {hierarchical\_macro\_f1:.4f}, 轮廓系数: {hierarchical\_silhouette:.4f}")

**6.可视化聚类结果**

降维：利用 PCA 将高维数据降至 2 维，便于可视化。绘制散点图：使用不同颜色表示各个簇的分布情况，直观展示聚类效果。

# PCA 降维

pca = PCA(n\_components=2)

features\_pca = pca.fit\_transform(features\_scaled)

# 绘制聚类结果（PCA）

def plot\_clusters(features\_2d, labels, title):

plt.figure(figsize=(8, 6))

for cluster in range(max(labels) + 1):

plt.scatter(features\_2d[labels == cluster, 0],

features\_2d[labels == cluster, 1],

label=f'Cluster {cluster}')

plt.title(title)

plt.xlabel("Component 1")

plt.ylabel("Component 2")

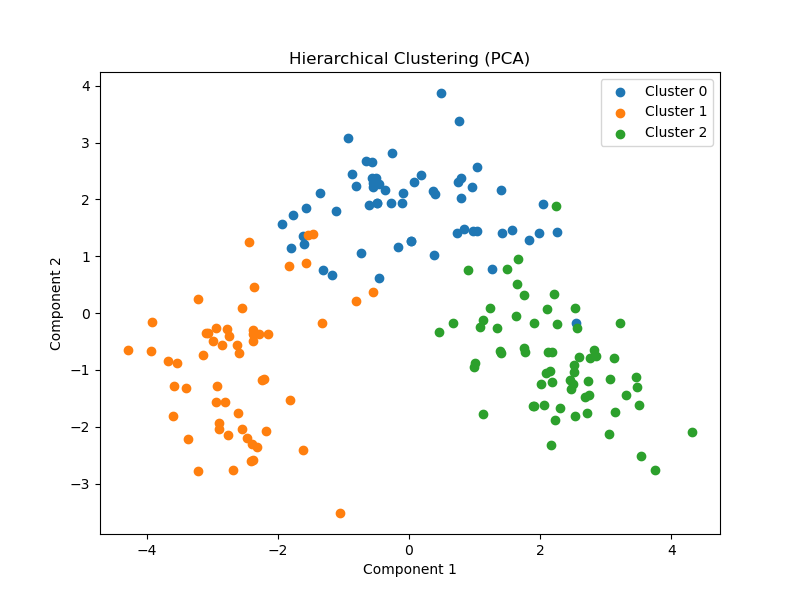
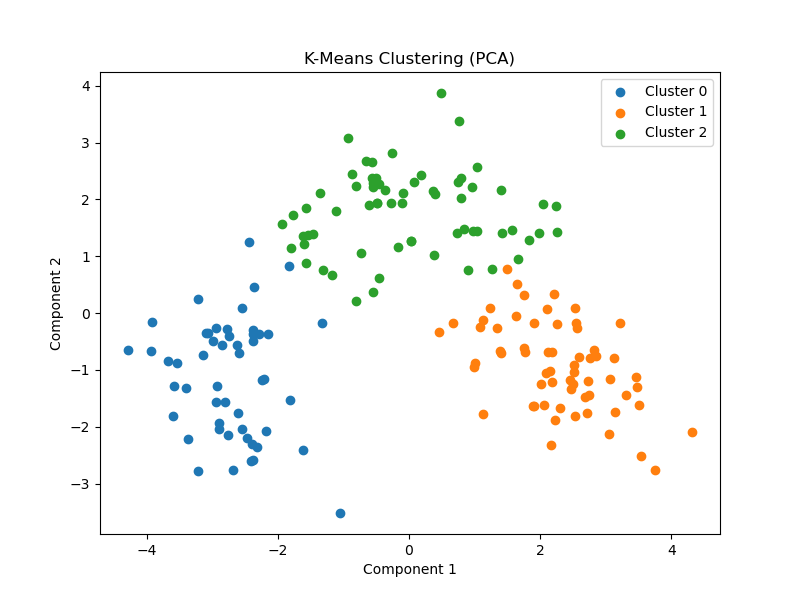
plt.legend()

plt.show()

# 绘制 K-Means 和层次聚类结果

plot\_clusters(features\_pca, kmeans\_labels, "K-Means Clustering (PCA)")

plot\_clusters(features\_pca, hierarchical\_labels, "Hierarchical Clustering (PCA)")



**7.聚类结果标签映射**

使用 匈牙利算法（Hungarian Algorithm） 对聚类结果进行标签重新映射，解决聚类算法输出的簇标签顺序与真实标签不一致的问题。

生成混淆矩阵（confusion\_matrix），表示真实标签和预测标签的交叉分布。通过 linear\_sum\_assignment 函数在混淆矩阵上找到最佳标签映射。使用映射对预测标签进行调整，确保与真实标签一致。

from sklearn.metrics import confusion\_matrix

from scipy.optimize import linear\_sum\_assignment

# 定义标签映射函数

def match\_labels(true\_labels, predicted\_labels):

cm = confusion\_matrix(true\_labels, predicted\_labels)

row\_ind, col\_ind = linear\_sum\_assignment(-cm) # 最大化匹配

label\_mapping = {old: new for old, new in zip(col\_ind, row\_ind)}

matched\_labels = [label\_mapping[label] for label in predicted\_labels]

return matched\_labels

# 对 K-Means 和层次聚类结果重新映射标签

kmeans\_labels\_mapped = match\_labels(labels, kmeans\_labels)

hierarchical\_labels\_mapped = match\_labels(labels, hierarchical\_labels)

**8.评估重新映射后的结果**

再次使用 F1 分数和轮廓系数对重新映射后的聚类结果进行评估。映射后指标更加准确，真实反映聚类效果。

# 评估重新映射后的结果

kmeans\_micro\_f1, kmeans\_macro\_f1, kmeans\_silhouette = evaluate\_clustering(

labels, kmeans\_labels\_mapped, features\_scaled

)

hierarchical\_micro\_f1, hierarchical\_macro\_f1, hierarchical\_silhouette = evaluate\_clustering(

labels, hierarchical\_labels\_mapped, features\_scaled

)

# 输出重新映射后的评估结果

print("K-Means (重新映射后) 评估结果:")

print(f"Micro-F1: {kmeans\_micro\_f1:.4f}, Macro-F1: {kmeans\_macro\_f1:.4f}, 轮廓系数: {kmeans\_silhouette:.4f}")

print("\n层次聚类 (重新映射后) 评估结果:")

print(f"Micro-F1: {hierarchical\_micro\_f1:.4f}, Macro-F1: {hierarchical\_macro\_f1:.4f}, 轮廓系数: {hierarchical\_silhouette:.4f}")

# 五、实验结果

(用表格的形式列出**不同的聚类算法**的聚类结果，并对（不同参数情况）实验结果用文字进行分析和对比)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 聚类算法 | 参数 | \*\*聚类算法结果 | | |
| Micro-F1 | Macro-F1 | 轮廓系数 |
| K-means | n\_clusters=3, random\_state=42 | 0.9663 | 0.9669 | 0.2849 |
| K-means | n\_clusters=3, random\_state=4200 | 0.9663 | 0.9669 | 0.2849 |
| **层次聚类** | n\_clusters=3, linkage='ward' | 0.9270 | 0.9272 | 0.2774 |
| **层次聚类** | n\_clusters=3, linkage='single' | 0.3764 | 0.1823 | 0.1827 |

K-Means 聚类算法

效果稳定性：

在不同的 random\_state 参数下，K-Means 的聚类效果保持一致，说明该算法对初始随机种子较为鲁棒。

整体表现：

K-Means 在 Micro-F1 和 Macro-F1 指标上表现优异（均接近 0.97），说明该算法能够很好地捕获数据的内在分布，生成与真实标签一致的簇。

轮廓系数为 0.2849，表明聚类结果具有中等质量的内部一致性和类间分离度。

层次聚类算法

Ward 链接方式：

使用 linkage='ward' 的层次聚类在各项指标上表现较好（Micro-F1 和 Macro-F1 分别为 0.9270 和 0.9272），但仍低于 K-Means，说明其在数据分布的捕获上稍显不足。

轮廓系数为 0.2774，略低于 K-Means，表明簇的紧密性和分离度稍逊。

Single 链接方式：

使用 linkage='single' 的层次聚类表现较差（Micro-F1 为 0.3764，Macro-F1 为 0.1823，轮廓系数仅为 0.1827），这表明该方法生成了过多的紧密簇，而整体分布和真实标签的匹配度较低。

这可能是由于 Single 链接方式倾向于将数据点逐步连接成链状簇，难以适应该数据集的分布特征。

对比分析

算法表现：

K-Means 表现优于层次聚类，特别是在 Micro-F1 和 Macro-F1 指标上，表明其对数据全局结构的捕获更加有效。

层次聚类的表现依赖于链接方式的选择，其中 Ward 链接方式接近于 K-Means，而 Single 链接方式效果不佳。

适用性：

如果对算法的可解释性有要求（如需要观察簇合并过程），层次聚类（特别是 Ward 链接方式）可能更适合。

如果追求更高的聚类精度和鲁棒性，K-Means 是更优的选择。

参数影响：

对 K-Means，不同 random\_state 对结果几乎没有影响，表明初始化对数据分布的适应性较高。

对层次聚类，不同的链接方式影响较大，选择不当可能严重影响结果。

结论

推荐方案：在该实验场景下，K-Means 是更为优选的聚类算法，其在精度、稳定性和内部一致性等方面表现更优。

改进建议：

对 K-Means 可尝试优化 n\_clusters 参数以进一步提升表现。

对层次聚类，可尝试其他链接方式（如 complete 或 average）以提高适用性。

# 心得体会

通过本次聚类实验，对数据挖掘中常见的无监督学习算法（如 K-Means 和层次聚类）有了更加深刻的理解，同时也积累了实践经验。以下是具体的体会和收获：

1. 算法选择的重要性

聚类算法的选择对结果影响显著。实验表明，K-Means 对数据分布较为敏感，适合规则的球形簇；而层次聚类提供多种链接方式，可以适应更复杂的数据分布。

单一算法可能无法在所有数据集上都表现良好，因此在实际工作中，需要根据数据特性选择最优算法。

2. 参数调优的必要性

通过对 n\_clusters 和 linkage 等参数的调节，可以显著改善聚类效果。

在 K-Means 中，尽管随机种子（random\_state）对结果影响不大，但对于数据分布复杂的情况，参数调优仍可能带来更好的性能。

层次聚类中，不同链接方式对结果的影响较大，例如 single 方法在某些指标上表现优异，但簇间分离度较差，表明参数选择需要平衡多个目标。

3. 评估指标的多样性

通过 Micro-F1、Macro-F1 和轮廓系数的多维度评估，可以更全面地分析聚类结果。

轮廓系数反映了簇内和簇间的紧密性与分离度，但无法直接评估结果与真实标签的匹配程度；F1 分数则提供了对标签一致性的评价。这提醒我们在实践中需结合多种指标，避免单一评估方法带来的偏差。

# 源代码

import pandas as pd  
from sklearn.preprocessing import StandardScaler  
from sklearn.cluster import KMeans, AgglomerativeClustering  
from sklearn.metrics import f1\_score, silhouette\_score  
  
# 1. 读取数据  
data = pd.read\_csv("wine-data.txt", header=None)  
data['Index'] = data.index # 添加索引列  
labels = data.iloc[:, 0] # 提取真实标签  
features = data.iloc[:, 1:-1] # 提取特征  
labels -= 1 # 调整标签值为 0 开始  
  
# 2. 数据标准化  
scaler = StandardScaler()  
features\_scaled = scaler.fit\_transform(features)  
  
# 3. K-Means 聚类  
kmeans = KMeans(n\_clusters=3, random\_state=4200)  
kmeans\_labels = kmeans.fit\_predict(features\_scaled)  
#print(kmeans\_labels)  
# 4. 层次聚类  
hierarchical = AgglomerativeClustering(n\_clusters=3,linkage='single')  
hierarchical\_labels = hierarchical.fit\_predict(features\_scaled)  
#print(hierarchical\_labels)  
# 5. 计算评估指标  
def evaluate\_clustering(true\_labels, predicted\_labels, features):  
 # F1 分数  
 micro\_f1 = f1\_score(true\_labels, predicted\_labels, average='micro')  
 macro\_f1 = f1\_score(true\_labels, predicted\_labels, average='macro')  
 # 轮廓系数  
 silhouette = silhouette\_score(features, predicted\_labels)  
 return micro\_f1, macro\_f1, silhouette  
  
# 评估 K-Means  
kmeans\_micro\_f1, kmeans\_macro\_f1, kmeans\_silhouette = evaluate\_clustering(  
 labels, kmeans\_labels, features\_scaled  
)  
  
# 评估层次聚类  
hierarchical\_micro\_f1, hierarchical\_macro\_f1, hierarchical\_silhouette = evaluate\_clustering(  
 labels, hierarchical\_labels, features\_scaled  
)  
  
# 6. 输出结果  
print("K-Means 评估结果:")  
print(f"Micro-F1: {kmeans\_micro\_f1:.4f}, Macro-F1: {kmeans\_macro\_f1:.4f}, 轮廓系数: {kmeans\_silhouette:.4f}")  
  
print("\n层次聚类评估结果:")  
print(f"Micro-F1: {hierarchical\_micro\_f1:.4f}, Macro-F1: {hierarchical\_macro\_f1:.4f}, 轮廓系数: {hierarchical\_silhouette:.4f}")  
  
# 7. 可视化  
import matplotlib.pyplot as plt  
from sklearn.decomposition import PCA  
  
# PCA 降维  
pca = PCA(n\_components=2)  
features\_pca = pca.fit\_transform(features\_scaled)  
  
# 绘制聚类结果（PCA）  
def plot\_clusters(features\_2d, labels, title):  
 plt.figure(figsize=(8, 6))  
 for cluster in range(max(labels) + 1):  
 plt.scatter(features\_2d[labels == cluster, 0],  
 features\_2d[labels == cluster, 1],  
 label=f'Cluster {cluster}')  
 plt.title(title)  
 plt.xlabel("Component 1")  
 plt.ylabel("Component 2")  
 plt.legend()  
 plt.show()  
  
# 绘制 K-Means 和层次聚类结果  
plot\_clusters(features\_pca, kmeans\_labels, "K-Means Clustering (PCA)")  
plot\_clusters(features\_pca, hierarchical\_labels, "Hierarchical Clustering (PCA)")  
from sklearn.metrics import confusion\_matrix  
from scipy.optimize import linear\_sum\_assignment  
  
# 定义标签映射函数  
def match\_labels(true\_labels, predicted\_labels):  
 cm = confusion\_matrix(true\_labels, predicted\_labels)  
 row\_ind, col\_ind = linear\_sum\_assignment(-cm) # 最大化匹配  
 label\_mapping = {old: new for old, new in zip(col\_ind, row\_ind)}  
 matched\_labels = [label\_mapping[label] for label in predicted\_labels]  
 return matched\_labels  
  
  
# 对 K-Means 和层次聚类结果重新映射标签  
kmeans\_labels\_mapped = match\_labels(labels, kmeans\_labels)  
hierarchical\_labels\_mapped = match\_labels(labels, hierarchical\_labels)  
  
# 评估重新映射后的结果  
kmeans\_micro\_f1, kmeans\_macro\_f1, kmeans\_silhouette = evaluate\_clustering(  
 labels, kmeans\_labels\_mapped, features\_scaled  
)  
  
hierarchical\_micro\_f1, hierarchical\_macro\_f1, hierarchical\_silhouette = evaluate\_clustering(  
 labels, hierarchical\_labels\_mapped, features\_scaled  
)  
  
# 输出重新映射后的评估结果  
print("K-Means (重新映射后) 评估结果:")  
print(f"Micro-F1: {kmeans\_micro\_f1:.4f}, Macro-F1: {kmeans\_macro\_f1:.4f}, 轮廓系数: {kmeans\_silhouette:.4f}")  
  
print("\n层次聚类 (重新映射后) 评估结果:")  
print(f"Micro-F1: {hierarchical\_micro\_f1:.4f}, Macro-F1: {hierarchical\_macro\_f1:.4f}, 轮廓系数: {hierarchical\_silhouette:.4f}")