安徽大学《机器学习》实验报告 5

学号：WA2214014　　专业：人工智能　　姓名：杨跃浙

实验日期：24.12.23　　教师签字：_____　　成绩：_____

[实验名称] 聚类实验

[实验目的]

1．熟悉和掌握 k-means 聚类算法

2．熟悉和掌握 DBSCAN 聚类算法

[实验要求]

1．采用 Python、Matlab 等高级语言进行编程，推荐优先选用 Python 语言

2．核心模型和算法需自主编程实现，不得直接调用 Scikit-learn、PyTorch 等成熟框架的第三方实现

3．代码可读性强：变量、函数、类等命名可读性强，包含必要的注释

4．提交实验报告要求：1）报告文件：命名为“学号-姓名-Lab5”； 2）提交时间截止：下次课

[实验原理]

(一) K-means 聚类算法

K-means 是一种经典的聚类算法，其目的是将数据分为 k 个互不重叠的簇（Cluster），使得同一个簇内的样本相似度尽可能高，不同簇之间的样本差异尽可能大。该算法采用迭代优化的方式，通过最小化簇内样本到聚类中心的距离平方和来优化结果。

1.目标函数

K-means 聚类的目标是最小化簇内样本的总误差平方和 (Within-Cluster Sum of Squares, WCSS)。目标函数定义为：

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

其中:
k 是簇的数量; C_i 是第 i 个簇;x 是样本点; μ_i 是第 i 个簇的质心。
目标函数衡量的是每个样本点到其所在簇质心的距离平方和,优化过程通过不断调整簇分配和簇中心来最小化该值。

2.距离度量
K-means 使用欧氏距离作为样本之间的相似度度量方式。两点之间的欧氏距离公式为:

$$d(x, \mu) = \sqrt{\sum_{j=1}^{m} (x_j - \mu_j)^2}$$

其中:

x_j 和 μ_j 分别是样本点和簇中心在第 j 维的值;m 是样本的特征维数。

3.算法流程
- 初始化:随机选择 k 个样本作为初始聚类中心;
- 分配样本:计算每个样本到所有聚类中心的距离,并将其分配到距离最近的聚类中心所在的簇;
- 更新聚类中心:对每个簇,计算簇内样本的均值作为新的聚类中心:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

- 迭代优化:重复样本分配和聚类中心更新步哄,直到聚类结果不再变化或达到最大迭代次数。

4. 收敛性
K-means 的优化过程保证目标函数值单调递减,并在有限步内收敛,但可能收敛到局部最优解

5.K-means 的优点和局限性
优点:
- 简单易实现;
- 收敛速度快,计算效率高;
- 对簇形状为球形、簇间间隔较大的数据效果较好。

局限性:
- 需要预先指定 k 的值;
- 对初始聚类中心敏感,可能陷入局部最优解;
- 对噪声和异常值敏感,容易导致聚类中心偏移。

本实验中,K-means 被用于对 52 周销量数据进行聚类,具体实现步骤如下:
- 归一化数据:使用 Min-Max 归一化将数据缩放到 [0,1] 区间;
- 随机初始化:随机选择 k 个聚类中心,设置 k=2,3,5 进行实验;
- 迭代聚类:重复分配样本和更新聚类中心,直至收敛或达到最大迭代次数;

• 可视化结果：通过 PCA 降维将高维数据投影到二维平面，以不同颜色区分不同簇，展示聚类效果。

(二) DBSCAN 聚类算法

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 是一种基于密度的聚类算法，能够有效发现任意形状的簇，同时可以自动识别噪声点。与 K-means 不同，DBSCAN 不需要预先指定簇的数量，而是依赖两个参数：邻域半径 ε 和密度阈值 MinPts。

1.基本概念

• 邻域：给定样本点 p ，以 ε 为半径的邻域定义为：

$$N(p) = \{q \in D | d(p, q) \leq \varepsilon\}$$

其中 $d(p, q)$ 表示 p 和 q 之间的距离。

• 核心点：若样本点的邻域中包含至少 MinPts 个样本点（包括自身），则该点是核心点。

• 边界点：不是核心点但属于某核心点邻域的点。

• 噪声点：既不是核心点，也不属于任何核心点邻域的点。

2.密度可达性

• 直接密度可达：若 $q \in N(p)$ 且 p 是核心点，则 q 直接密度可达。

• 密度可达：若存在点序列 $p_1, p_2, \ldots, p_n$ ，使得 $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n$ 是直接密度可达的，则称 p_1 和 p_n 密度可达。

3. 算法流程

• 初始化：标记所有样本点为未访问；

• 识别核心点：找出所有邻域内样本数 $\geq$ MinPts 的核心点；

• 扩展簇：从某核心点开始，递归地将与其密度可达的点加入簇；

• 标记噪声：未分配到任何簇的点标记为噪声。

4.聚类系数

为衡量聚类的紧密性，定义聚类系数为：

$$\text{Coefficient} = \frac{2 \times n_{\text{odjec}}}{n_{\text{bode}} \times (n_{\text{bode}} - 1)}$$

其中：

n_{edge} 为聚类中节点之间的边总数；n_{Dode} 为聚类中节点的总数。

5.DBSCAN 的优点和局限性

优点：

• 能识别任意形状的簇；

• 对噪声点鲁棒性较强；

• 不需要预先指定簇的数量。

局限性：

• 对参数 ε 和 MinPts 敏感，参数选择不当可能影响聚类效果；

• 当簇的密度差异较大时，效果不佳。

本实验中，DBSCAN 被用于对蛋白质相互作用数据进行聚类，具体步骤如下：
• 邻域定义：使用邻接矩阵直接定义邻域，以 $\varepsilon = 1$ 表示相互作用的蛋白质之间的密度直达关系；
• 参数设置：选择 $\mathrm{MinPts} = 10,15,20$ 分别进行实验；
• 构造核心对象：识别邻域内满足密度阈值的核心点；
• 违归扩展簇：将与核心点密度可达的节点加入簇；
• 计算聚类系数：分析不同聚类的内部紧密程度；
• 输出结果：记录不同 MinPts 值下的聚类数量及其系数。

（三）PCA 降维

主成分分析（PCA，Principal Component Analysis）是一种经典的降维方法，广泛用于高维数据的特征提取、可视化和数据压缩。PCA 的核心思想是通过线性变换，将数据从原始高维空间投影到维数较低的空间，同时尽可能保留数据的全局结构和方差信息。简单来说，PCA 寻找的是数据中最主要的变化方向，并用这些方向构造新的特征。

假设有一个数据集 X，包含 n 个样本和 m 个特征，可以表示为一个 $n \times m$ 的矩阵。PCA 的目标是通过一系列步骤，将数据降到低维空间的同时保留尽可能多的信息，具体过程如下：

1.数据标准化

数据的不同特征可能具有不同的量纲和尺度（例如，有些特征值可能在 0 到 1 之间，而另一些可能在数百甚至数千范围）。因此，PCA 的第一步是对数据进行标准化，去除特征之间的量纲影响，使数据的每个特征的均值为 0，标准差为 1。标准化公式为：

$$Z = \frac{X - \mu}{\sigma}$$

其中：

Z 是标准化后的数据；μ 是每列特征的均值；σ 是每列特征的标准差。

2.计算协方差矩阵

标准化后，数据集的特征之间可能仍然具有相关性。为了量化这种相关性，需要计算数据的协方差矩阵。协方差矩阵的公式为：

$$\Sigma = \frac{1}{n-1} Z^T Z$$

其中：

Σ 是一个 $m \times m$ 的对称矩阵；矩阵中的元素 Σ_{ij} 表示第 i 个特征与第 j 个特征的协方差。

协方差矩阵揭示了特征之间的线性相关性。如果某两个特征的协方差较大，说明它们的变化模式高度相似，存在冗余信息。

3.特征分解

协方差矩阵是对称的，可以通过特征值分解（Eigen Decomposition）得到其特征值和特征向量。假设协方差矩阵的特征值为 $\lambda_1, \lambda_2, \ldots, \lambda_m$ ，对应的特征向量为 $v_1, v_2, \ldots, v_m$，满足:

$$\Sigma v = \lambda v$$

其中:

λ 表示特征值，描述了对应特征向量方向上的数据方差；v 是特征向量，表示特征值对应的方向。

特征值越大，表示该方向上的方差越大，说明该方向包含的数据信息越多。

4. 选择主成分

将特征值按从大到小的顺序排列，选择最大的 k 个特征值及其对应的特征向量作为主成分方向。具体地，选择前k个特征向量构成矩阵V_k，其中V_k是一个$m \times k$的矩阵。

在降维到二维的情况下，选择前两个特征值最大的特征向量 v_1 和 v_2 ，它们分别代表数据中变化最大的两个方向。

5.数据投影

将原始数据投影到主成分构成的低维空间中，得到降维后的数据。投影的公式为:

$$Y = ZV_k$$

其中:

Y 是降维后的数据；Z 是标准化后的数据; V_k 是选定的主成分特征向量矩阵。

投影后的数据保留了原始数据的主要变化信息，并将数据映射到低维空间。

6.PCA 的优点和局限性

优点:

- 降维和特征压缩：通过降维减少数据的复杂性，提高处理效率；
- 可视化高维数据：将高维数据映射到二维或三维空间，便于分析和理解；
- 减少冗余信息：通过协方差矩阵消除特征间的线性相关性，生成互相正交的主成分。

局限性:

- 线性假设：PCA 假设数据的特征是线性相关的，无法捕捉非线性结构；
- 信息丢失：降维过程不可避免会丢失部分信息，尤其是小特征值对应的信息；
- 依赖均值和方差：对异常值较敏感，可能导致主成分方向发生偏移。

在本实验中，PCA 被用来将高维数据降到二维，以便可视化聚类结果。具体步骤包括:

- 标准化数据：将原始数据进行标准化处理；
- 计算协方差矩阵：衡量各特征之间的相关性；
- 特征值分解：获取协方差矩阵的特征值和特征向量；
- 选择主成分：选择前两个最大特征值对应的特征向量；

• 投影到二维空间：使用降维后的数据生成二维平面图，以不同颜色标注各个聚类。

[实验内容]

(一) K—means 聚类算法

采用数据集 “data/clustering1 .csv”进行 k—means 聚类算法实验。

数据集简介：本数据集来自 UCI 数据库 Sales_Transa

2 .计算协方差矩阵

标准化后，数据集的特征之间可能仍然具有相关性。为了量化这种相关性，需要计算数据的协方差矩阵。协方差矩阵的公式为：

$$\Sigma = \frac{1}{n-1} Z^T Z$$

其中：

Σ 是一个 $m \times m$ 的对称矩阵；

矩阵中的元素 Σ_{ij} 表示第 i 个特征与第 j 个特征的协方差。

协方差矩阵揭示了特征之间的线性相关性。如果某两个特征的协方差较大，说明它们的变化模式高度相似，存在冗余信息。

ctions_Dataset_Weekly 数据，经过简单筛选后维度为 666×52，对应 666 种商品在 52 周的销量。

1. 将每一种商品看作一个样本，每一周看作一个属性，对其进行聚类。

2. 对数据进行 min—max 归一化至[0,1]区间内：

3. 随机挑选 k 个初始聚类中心，使用欧氏距离度量各样本与聚类中心的距离，划分聚类。

4. 更新聚类中心，重新划分聚类。

5. 重复第 4 步，直到各类包含的样本不再改变或达到最大迭代次数，输出结果。

6. 对原样本数据计算均值和方差，进行标准化，使用 PCA 降维方法降至二维，打印散点图至二维平面中，以不同的颜色区分不同类别的样本。

7. 更换随机聚类中心，重复上述第 3–5 步并打印散点图。

8. 参数设置：k 分别取 2、3、5，最大迭代次数 maxiter 取 100

代码输出：

- 样本二维图像，共 6 幅放在一起，三行两列，分别对应 3 个不同的 k 值，每个 k 值对应两次不同的初始聚类中心。

(二) DBSCAN 聚类算法

采用数据集“data/clustering2.csv”进行 DBSCAN 聚类算法实验。

数据集简介：本数据来自 DIP 数据库，为蛋白质相互作用网络数据，网络$G = (E, V)$，E 是网络中的节点的集合，即蛋白质，V 是网络中的边的集合，即蛋白质–蛋白质相互作用。网络保存为邻接矩阵 A 的形式，矩阵中元素A_{ij}代表第 i 个蛋白质和第 j 个蛋白质是否存在相互作用，若值为 1，则存在相互作用，并有一条边连接这两个节点。矩阵中A_{ij}和A_{ji}对应同一条边。矩阵维度为 1274×1274，即共有 1274 个蛋白质。存在相互作用的两个蛋白质之间的距离定为 1，无须计算。

1. 本实验中，邻域半径 ε 设为 1，即密度直达的两个节点就是存在相互作用的两个蛋白质，也就是说邻接矩阵中值为 1 的元素所在的行和列所对应的两个节点是密度直达的。

2. 密度阈值 MinPts 设为 10、15、20。

3. 依据密度聚类流程，首先根据 MinPts 构造核心对象子集。

4. 依次从中取出一个核心对象（同时从子集中删去该节点），将它的邻居加入待访问节点子集。

4.1. 依次从待访问节点子集中取出一个节点 v（同时从子集中删去该节点），若它是核心对象，将它的邻居也放入待访问节点子集。重复这一过程，直到待访问节点子集为空集，此时所有访问过的结点构成一个聚类。

5. 重复第 5 步，直到核心对象子集为空集。

6. 计算各个聚类的聚类系数：

$$coef = \frac{n\_edge}{2 \times n\_node \times (n\_node - 1)}$$

其中 n_edge 为一个聚类中所有节点两两之间存在的边的总数，n_node 为该聚类包含的节点数量。易知，若一个聚类内所有节点两两之间都有边相连（聚类内所有节点互相都是密度直达的），该聚类的聚类系数为 1，。

7. 保存所有聚类的聚类系数。

代码输出：

- 不同密度阈值 MinPts 取值时对应的聚类数量和相应各个聚类的聚类系数。

[实验代码和结果]

(一) K-means 聚类算法

实验代码：

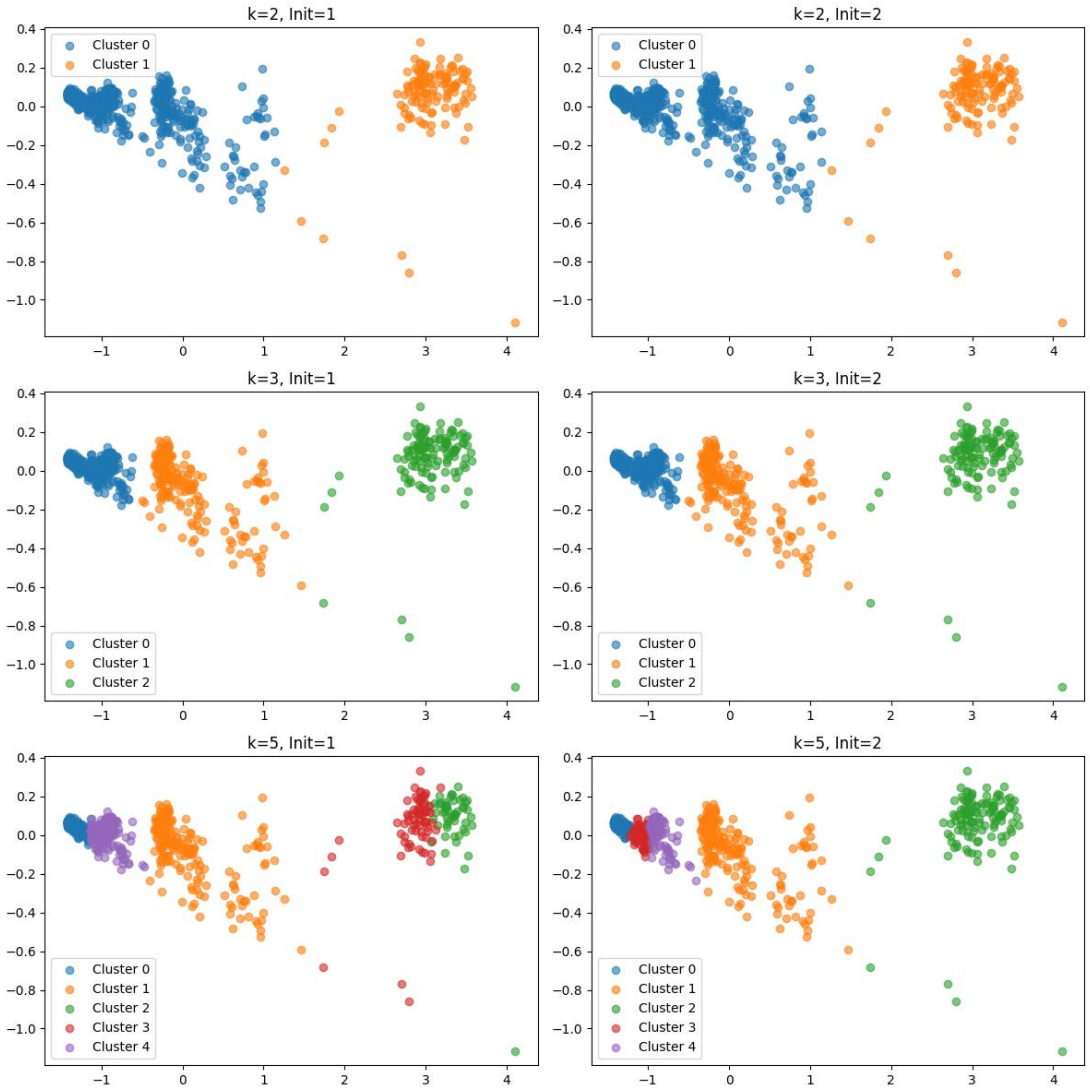
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Step 1: Load the dataset
file_path = 'data/clustering1.csv'
```

```
data = pd.read_csv(file_path).values

# Step 2: Min-Max Normalization
def min_max_normalize(data):
return (data - data.min(axis=0)) / (data.max(axis=0) -
data.min(axis=0))

normalized_data = min_max_normalize(data)

# Step 3: K-means clustering
def kmeans_clustering(data, k, max_iter=100,
random_state=None):
if random_state is not None:
np.random.seed(random_state)

# Randomly initialize cluster centers
n_samples, n_features = data.shape
centers = data[np.random.choice(n_samples, k,
replace=False)]

for _ in range(max_iter):
# Assign clusters based on the closest center
distances = np.linalg.norm(data[:, np.newaxis] - centers,
axis=2)
cluster_labels = np.argmin(distances, axis=1)

# Recompute cluster centers
new_centers = np.array([data[cluster_labels ==
i].mean(axis=0) for i in range(k)])
# Stop if centers do not change
if np.allclose(centers, new_centers):
break
centers = new_centers

return cluster_labels, centers

# Step 4: PCA for dimensionality reduction
def pca(data, n_components=2):
# Center the data
```

```
mean = data.mean(axis=0)
centered_data = data - mean

# Compute covariance matrix
covariance_matrix = np.cov(centered_data, rowvar=False)

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors =
np.linalg.eigh(covariance_matrix)

# Sort eigenvectors by descending eigenvalues
sorted_indices = np.argsort(eigenvalues)[::-1]
top_eigenvectors = eigenvectors[:,
sorted_indices[:n_components]]

# Project data onto top components
reduced_data = centered_data.dot(top_eigenvectors)
return reduced_data

# Step 5: Visualization and save
def kmeans_clustering_and_save(data, k_values, max_iter=100,
save_path="figure/"):
os.makedirs(save_path, exist_ok=True)

fig, axes = plt.subplots(len(k_values), 2, figsize=(12, 12))
for i, k in enumerate(k_values):
for j in range(2): # Two different random initializations
cluster_labels, _ = kmeans_clustering(data, k,
max_iter=max_iter, random_state=j)

# Reduce data to 2D using PCA
reduced_data = pca(data, n_components=2)

# Plot the clustered data in 2D
ax = axes[i, j]
for cluster in np.unique(cluster_labels):
cluster_points = reduced_data[cluster_labels == cluster]
ax.scatter(cluster_points[:, 0], cluster_points[:, 1],
label=f'Cluster {cluster}', alpha=0.6)
```

```
ax.set_title(f'k={k}, Init={j+1}')
ax.legend()
plt.tight_layout()
file_name = os.path.join(save_path, "kmeans_results.png")
plt.savefig(file_name) # Save the figure
plt.show()
print(f"Figure saved at: {file_name}")

# Step 6: Apply the function with k=2, 3, 5
kmeans_clustering_and_save(normalized_data, k_values=[2, 3,
5])
```

实验结果：



（二）　　DBSCAN 聚类算法

实验代码：

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = 'data/clustering2.csv'
adj_matrix = pd.read_csv(file_path, header=None).values

# Step 1: DBSCAN implementation with ε=1
def dbscan_custom(adj_matrix, min_pts_values):
n = adj_matrix.shape[0]
results = {}

for min_pts in min_pts_values:
visited = np.zeros(n, dtype=bool) # To mark visited nodes
cluster_labels = -np.ones(n, dtype=int) # To assign clusters
cluster_id = 0 # Cluster ID starts from 0
core_objects = []

# Step 2: Identify core objects
for i in range(n):
neighbors = np.where(adj_matrix[i] == 1)[0]
if len(neighbors) >= min_pts:
core_objects.append(i)

# Step 3: Process each core object
for core in core_objects:
if visited[core]:
continue
visited[core] = True
cluster_labels[core] = cluster_id

# Initialize the neighbor set
neighbors = set(np.where(adj_matrix[core] == 1)[0])
neighbors_to_visit = neighbors.copy()

while neighbors_to_visit:
neighbor = neighbors_to_visit.pop()
```

```
if not visited[neighbor]:
visited[neighbor] = True
cluster_labels[neighbor] = cluster_id
# Check if the neighbor is also a core object
neighbor_neighbors = np.where(adj_matrix[neighbor] == 1)[0]
if len(neighbor_neighbors) >= min_pts:
neighbors_to_visit.update(neighbor_neighbors)

cluster_id += 1

# Step 4: Compute clustering coefficients
cluster_coefficients = {}
for cluster in range(cluster_id):
cluster_nodes = np.where(cluster_labels == cluster)[0]
if len(cluster_nodes) > 1:
subgraph = adj_matrix[np.ix_(cluster_nodes, cluster_nodes)]
n_edges = np.sum(subgraph) / 2 # Count undirected edges
n_nodes = len(cluster_nodes)
cluster_coefficient = (2 * n_edges) / (n_nodes * (n_nodes -
1))
else:
cluster_coefficient = 0
cluster_coefficients[cluster] = cluster_coefficient

results[min_pts] = {
"num_clusters": cluster_id,
"cluster_coefficients": cluster_coefficients
}

return results

# Run the custom DBSCAN algorithm
min_pts_values = [10, 15, 20]
results = dbscan_custom(adj_matrix, min_pts_values)

# Print results
for min_pts, result in results.items():
print(f"MinPts = {min_pts}")
print(f"Number of clusters: {result['num_clusters']}")
```

```
print("Cluster coefficients:")
for cluster, coefficient in
result["cluster_coefficients"].items():
print(f" Cluster {cluster}: {coefficient:.4f}")
```

实验结果：

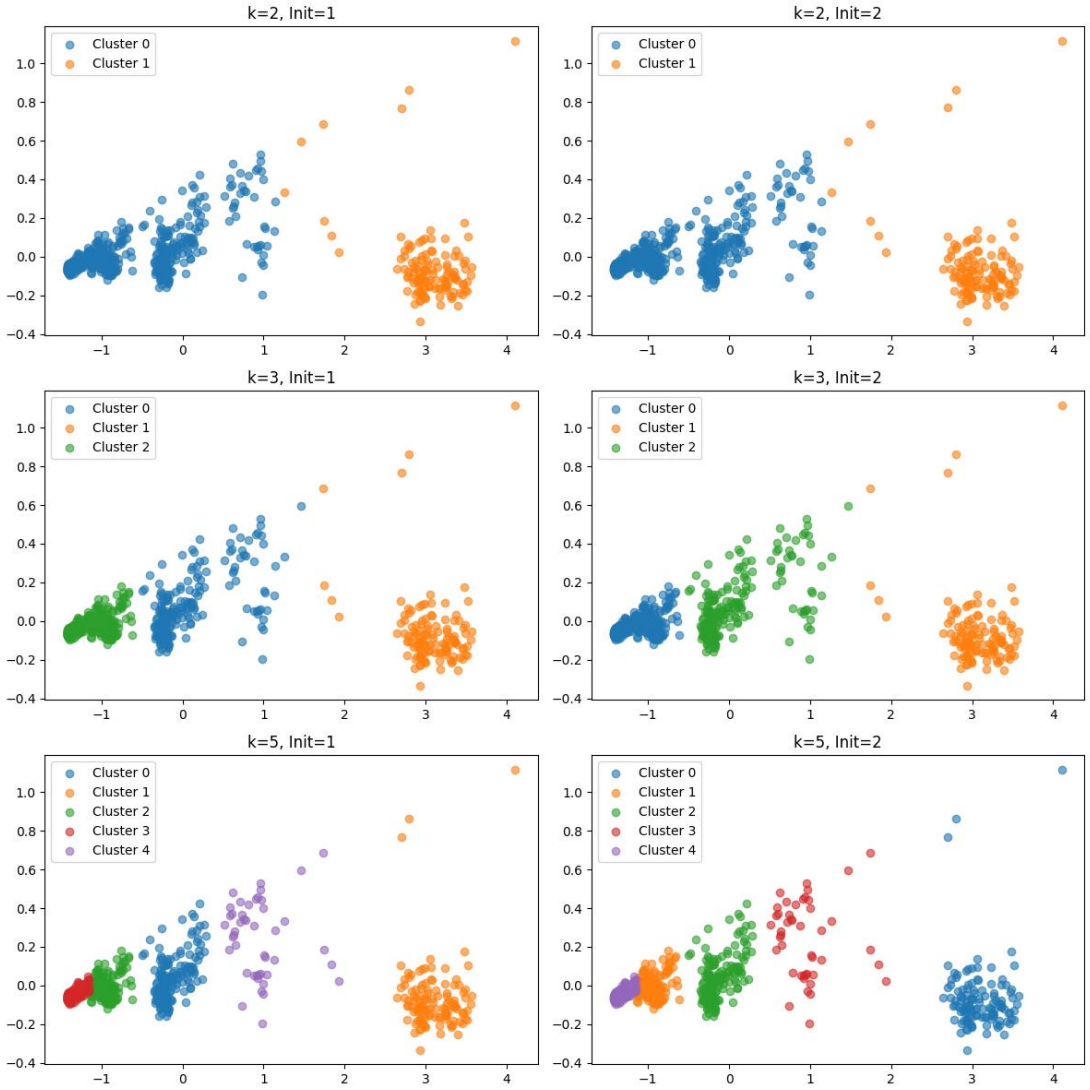
```
MinPts = 10
Number of clusters: 10
Cluster coefficients:
  Cluster 0: 0.0070
  Cluster 1: 0.0426
  Cluster 2: 0.1053
  Cluster 3: 0.1333
  Cluster 4: 0.1333
  Cluster 5: 0.1250
  Cluster 6: 0.1176
  Cluster 7: 0.0833
  Cluster 8: 0.0952
  Cluster 9: 0.1111

MinPts = 15
Number of clusters: 10
Cluster coefficients:
  Cluster 0: 0.0070
  Cluster 1: 0.0426
  Cluster 2: 0.1053
  Cluster 3: 0.1333
  Cluster 4: 0.1333
  Cluster 5: 0.1250
  Cluster 6: 0.1176
  Cluster 7: 0.0833
  Cluster 8: 0.0952
  Cluster 9: 0.1111

MinPts = 20
Number of clusters: 7
Cluster coefficients:
  Cluster 0: 0.0070
  Cluster 1: 0.0426
  Cluster 2: 0.1176
  Cluster 3: 0.1053
  Cluster 4: 0.0833
  Cluster 5: 0.0952
  Cluster 6: 0.1111
```

[小结或讨论]

通过本次实验，我深入理解并掌握了 K-means 和 DBSCAN 两种聚类算法的原理、实现方法以及各自的优缺点。实验过程中，针对不同的数据集和任务场景，我分别运用了这两种算法，并对结果进行了详细分析和比较，得出了以下重要认识。

在 K-means 聚类实验 中，我首先使用了销量数据，通过 Min-Max 归一化将数据标准化到 [0,1] 区间，降低了特征尺度对距离计算的影响。在聚类的核心流程中，我随机初始化了聚类中心，并通过欧氏距离的度量方法完成了样本分配和中心更新。实验结果表明，当 k 值不同（例如 k=2, 3, 5）时，聚类结果存在显著差异：较小的 k 值倾向于合并更多相似但可能具有细微差异的样本，而较大的 k 值则进一步细分了数据，使得每一类的内部差异更加显著。通过 PCA 降维和散点图的可视化，我观察到不同初始聚类中心对结果的影响，这表明 K-means 对初始条件较为敏感，可能陷入局部最优解。

同时，我也注意到，K-means 的目标函数，即簇内误差平方和（WCSS），在迭代过程中单调递减，并最终收敛。这与理论一致，证明了算法的正确性和稳定性。然而，由于数据簇间形状并非完全球形，我发现 K-means 在某些情况下的表现并不理想，这主要是因为其假设数据满足欧几里得空间的球形分布，而忽略了复杂形状的可能性。
同时，我发现了采用 sklearn 库中现成的 K-means 聚类，将会得到不一样的可视化结果：



这种现象主要来源于算法实现细节的差异。首先，sklearn 的实现使用了改进的初始化方法，例如 K-means++，能够在初始阶段选择彼此距离较远的聚类中心，这样的策略通常能更好地避免陷入局部最优，从而提高聚类的质量。而我的手动实现中，聚类中心的选择是完全随机的，可能导致结果的稳定性和优化效果略逊一筹。此外，sklearn 在迭代过程中可能对目标函数的优化采用了更加严格的收敛条件，并且对数值运算的稳定性进行了细致的处理，例如使用高效的线性代数库或优化的矩阵运算，而我的手动实现较为简单，可能存在精度不足或对特殊情况的处理不够完善的问题。另一方面，在数据预处理中，我使用了 Min-Max 归一化，而 sklearn 中可能默认进行了 Z-score 标准化或允许用户灵活选择归一化方式，这种预处理的差异直接影响了样本点间的距离计算，进而导致聚类结果的不同。此外，降维过程中，sklearn 的 PCA 实现可能采用了更高效的奇异值分解（SVD），这与我手动计算协方差矩阵的方式有所不同，可能会对数据的投影方向产生微妙的影响，最终在可视化时表现为样本点的分布变化或分类边界的不同。这些差异反映了一个高度优化的库函数在实践中能够提供更稳定、更高效结果的优势，同时也提示我，在实际工作中要充分理解工具的具体实现方式和优化策略，以便更全面地解释结果和选择适合问题需求的解决方案。

在 DBSCAN 聚类实验 中，我选择了蛋白质相互作用网络数据作为实验对象。与 K-means 不同，DBSCAN 不需要预设簇的数量，而是通过密度阈值（MinPts）和邻域半径（ε）来定义簇的形状和大小。在实验中，我固定了 ε=1，并对 MinPts 设定了不同的值（如 10、15、20）。结果表明，随着 MinPts 的增加，聚类数量逐渐减少，单个聚类的规模也随之增大。这表明，较高的 MinPts 值会筛选掉更多的低密度区域，保留密度更高的核心区域作为聚类。

从算法结果来看，DBSCAN 成功识别出了数据中非球形的簇，并能够对噪声点（未分配到任何聚类的点）进行标记。通过计算聚类系数，我进一步分析了不同聚类的内部紧密性（公式 $C = \frac{2n_{edge}}{n_{node} \cdot (n_{node}-1)}$）。实验显示，密度更高的聚类其聚类系数更接近 1，这与理论结果一致。但我也发现，当簇的密度差异较大时，DBSCAN 对某些低密度区域的聚类效果欠佳。此外，DBSCAN 对参数 ε 和 MinPts 的选择较为敏感，适当的参数调优对结果的质量至关重要。

综合来看，这两种聚类算法各有其适用场景和局限性。K-means 适用于球形分布、簇间间隔较大的数据集，但对初始条件敏感，且对噪声点和异常值的鲁棒性较弱。DBSCAN 则在任意形状的簇以及噪声点处理方面更具优势，但其参数选择复杂且对不同密度分布的簇表现有限。在实际应用中，我会根据数据特性和任务需求，选择合适的聚类算法或结合两者，以期获得更优的结果。

本次实验不仅让我在理论上加深了对聚类方法的理解，也通过实践掌握了如何从数据预处理到结果分析的全流程操作，为后续的机器学习研究和实际应用奠定了坚实的基础。