一、实验内容

问题描述：聚类分析是研究（样品或指标）分类问题的一种统计分析方法，同时也是人工智能中的一个重要算法。聚类（Cluster）分析是由若干模式（Pattern）组成的，通常，模式是一个度量（Measurement）的向量，或者是多维空间中的一个点。聚类分析以相似性为基础，在一个聚类中的模式之间比不在同一聚类中的模式之间具有更多的相似性。

内容提要：针对教师指定的两类公用数据集（纯数值型例如UCI Iris，混杂型数据例如UCI Bank Marketing），学生对给定的数据进行聚类。本次实验主要内容包括数据处理、算法实现和评价方法。

- 纯数值型数据集，UCI Iris，150个样本，4维。（参考iris.data文件）
  http://archive.ics.uci.edu/ml/datasets/Iris
- 混杂型数据集，UCI Bank Marketing，4119个样本，20维。（参考bank-additional.csv文件）
  http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

二、实验设备

1. 实验设备：台式机/笔记本等不限
2. 平台：Visual C++ / Python等不限

三、实验步骤

1. 读取数据，并做预处理
2. 至少实现一种聚类算法，选择评价方法并分析原因
3. 选择适当可视化方法显示结果
4. *扩展选做题：可以尝试多种聚类算法并比较结果

四、分析说明（包括结果图表分析说明，主要核心代码及解释）

对于任务一，使用K-means++聚类算法进行聚类并分析

1. K-means++算法

原理：K-means是一种常用的无监督聚类算法，其目的是将数据分为K个簇，使得同一簇内的数据点彼此相似，而不同簇之间的数据点差异较大。
K-means++是K-means聚类算法的一种优化版本，其核心思想是在初始化阶段 智能地选择聚类中心，从而避免了传统K-means聚类中由于随机初始化中心可能导致的较差结果。通过优化初始中心的选择，K-means++不仅能加快算法的收敛速度，还能有效提高聚类的质量。
核心步骤：
（1）选择初始聚类中心：通过根据已有聚类中心选择剩余的中心点，使得初始中心尽可能均匀地分布在数据集的空间中。
（2）分配每个数据点：将每个数据点分配给距离它最近的聚类中心。
（3）更新聚类中心：计算每个簇中所有点的均值，更新聚类中心为这些点的均值。
（4）重复迭代：直到聚类中心不再变化（或变化非常小），算法收敛。
（5）参数选择：K-means++ 需要设置一个参数 K（聚类数）。选择K的值对结果有重要影响，通常通过交叉验证或其他方法来确定最优的K值。
（6）特征归一化：在 K-means++ 中，特征归一化非常重要。因为距离计算（如欧几里得距离）对不同特征的尺度非常敏感。为了避免某些特征的值范围过大影响距离计算，常常需要对数据进行归一化或标准化处理。常用的标准化方法包括MinMaxScaler（将数据缩放到0-1范围内）和StandardScaler（将数据转化为均值为0，标准差为1的数据）。
实验过程：

（1）数据预处理

A. 加载数据

加载数据集：从 UCI 数据库获取 鸢尾花数据集，数据集包含 150 个样本，每个样本包含 4 个特征（萼片长度、萼片宽度、花瓣长度、花瓣宽度）以及类别标签。

```
# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)
```

B. 新增特征

本实验用两种方法进行聚类以观察进行分析，**一种是以面积为特征进行分析，一种通过主成分分析降维进行分析。**
以面积为特征：通过计算萼片面积 (sepal_area = sepal_length * sepal_width) 和花瓣面积 (petal_area = petal_length * petal_width) 生成新的特征，用于聚类。（使用主成分分析则跳过此步骤）

```
# 2. 特征工程：计算面积和比例特征
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']
# 计算比例特征
data['sepal_ratio'] = data['sepal_length'] / data['sepal_width']
data['petal_ratio'] = data['petal_length'] / data['petal_width']
```

（2）数据清洗

A**以面积为特征进行分析：**计算每个数据点的Z-score，对epal_area和petal_area进行异常值检测。

```
# 数据清洗：去除异常值
z_scores = data[['sepal_area', 'petal_area']].apply(zscore)
data_cleaned = data[(z_scores.abs() <= 3).all(axis=1)]
```

B**通过主成分分析降维进行分析：**利用Z-score检查数据中的异常值。

```
# 2. 数据清洗：去除异常值
z_scores = data.iloc[:, :-1].apply(zscore)
data_cleaned = data[(z_scores.abs() <= 3).all(axis=1)]
```

（3）数据标准化

使用MinMaxScaler对数据进行标准化，使得各特征的值缩放到 [0, 1] 的范围，避免特征之间的尺度差异影响聚类结果。

```
# 数据标准化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

（4）主成分分析（以面积为特征不需此步骤）

```
# 主成分分析 (PCA)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

（5）K-means++聚类

A. K-means++初始化

采用 K-means++ 方法初始化聚类中心，避免传统 K-means 中随机初始化聚类中心可能导致的聚类结果差。

通过智能选择初始中心，确保每次聚类时选择的聚类中心尽可能分布均匀，从而提高聚类的效果。

```
# 定义聚类数目范围
k_values = range(1, 21)

# 存储每个 k 的误差平方和 (WCSS)
wcss = []

# 遍历不同的 k 值
for k in k_values:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=20, max_iter=300, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

B. K-means++初始化

通过肘部法则确定最佳聚类数。遍历不同的K值（从1到20），计算每 个K值对应的误差平方和（WCSS）。绘制肘部图，通过观察WCSS随K值变化的趋势，选择最佳的K值。

```
# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# 确定最佳 k 值（通过肉眼观察肘部点）
optimal_k = 3  # 根据肘部图确定
print(f"最佳簇数 k: {optimal_k}")

# 使用最佳 k 值进行聚类
kmeans_optimal = KMeans(n_clusters=optimal_k, init='k-means++', n_init=20, max_iter=300, random_state=42)
kmeans_optimal.fit(X_scaled)
labels_optimal = kmeans_optimal.labels_
```

（6）聚类结果评估

使用轮廓系数来评估聚类效果，轮廓系数值越接近1，表示聚类效果越好。

```python
# 计算轮廓系数
silhouette = silhouette_score(X_scaled, labels_optimal)
print(f"轮廓系数: {silhouette:.4f}")

# 绘制轮廓图
silhouette_vals = silhouette_samples(X_scaled, labels_optimal)
plt.figure(figsize=(10, 6))
y_lower, y_upper = 0, 0
for i in range(optimal_k):
    cluster_silhouette_vals = silhouette_vals[labels_optimal == i]
    cluster_silhouette_vals.sort()
    y_upper += len(cluster_silhouette_vals)
    plt.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none')
    y_lower += len(cluster_silhouette_vals)

plt.axvline(x=silhouette, color="red", linestyle="--")
plt.title("轮廓图")
plt.xlabel("轮廓系数")
plt.ylabel("样本")
plt.show()
```

（7）结果可视化

肘部图：绘制肘部图，展示不同K值下的WCSS，以帮助确定最佳的聚类数K。

```python
# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

轮廓图：通过绘制轮廓图，展示每个簇的轮廓系数，分析每个簇的聚类质量。

```python
# 绘制轮廓图
silhouette_vals = silhouette_samples(X_scaled, labels_optimal)
plt.figure(figsize=(10, 6))
y_lower, y_upper = 0, 0
for i in range(optimal_k):
    cluster_silhouette_vals = silhouette_vals[labels_optimal == i]
    cluster_silhouette_vals.sort()
    y_upper += len(cluster_silhouette_vals)
    plt.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none')
    y_lower += len(cluster_silhouette_vals)
```

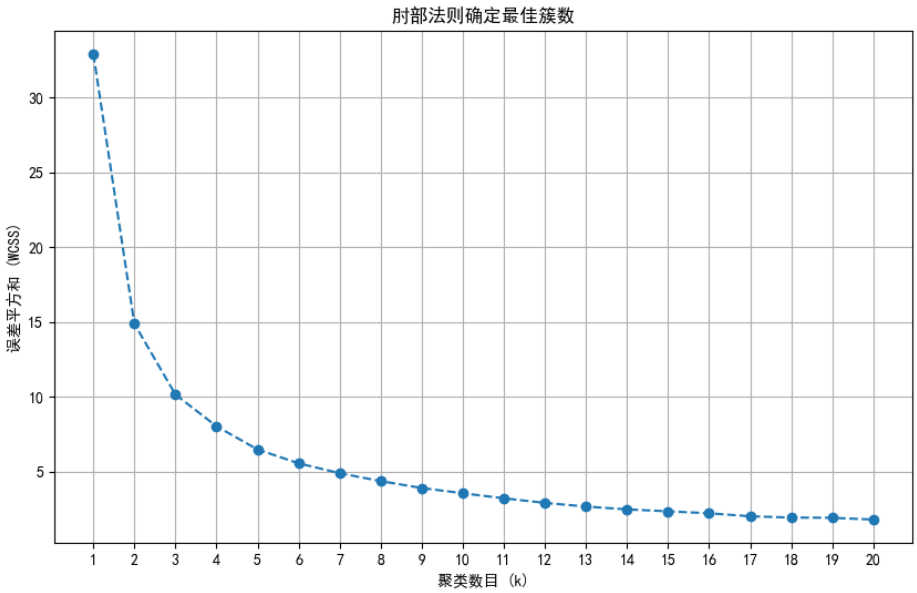
聚类散点图：根据萼片面积（sepal_area）和花瓣面积（petal_area）作为坐标，绘制聚类结果的散点图。

```python
# 可视化聚类结果
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels_optimal, cmap='viridis', s=50, label='样本点')
plt.scatter(kmeans_optimal.cluster_centers_[:, 0], kmeans_optimal.cluster_centers_[:, 1],
            c='red', s=200, marker='X', label='聚类中心')
plt.title(f'基于最佳 k={optimal_k} 的 K-means 聚类结果 (基于面积特征)')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.legend()
plt.show()
```

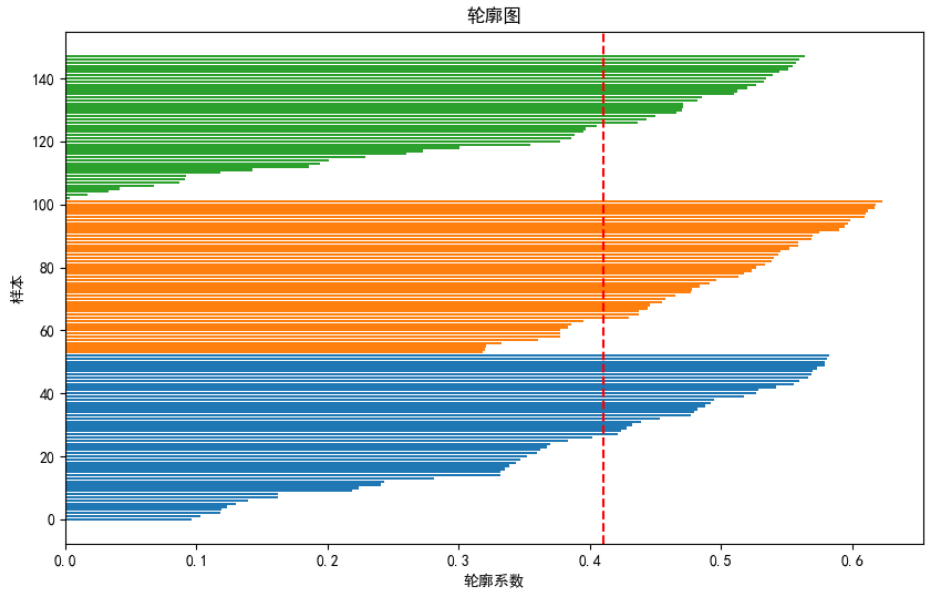
（8）运行结果截图分析

A. 以面积为特征进行分析：

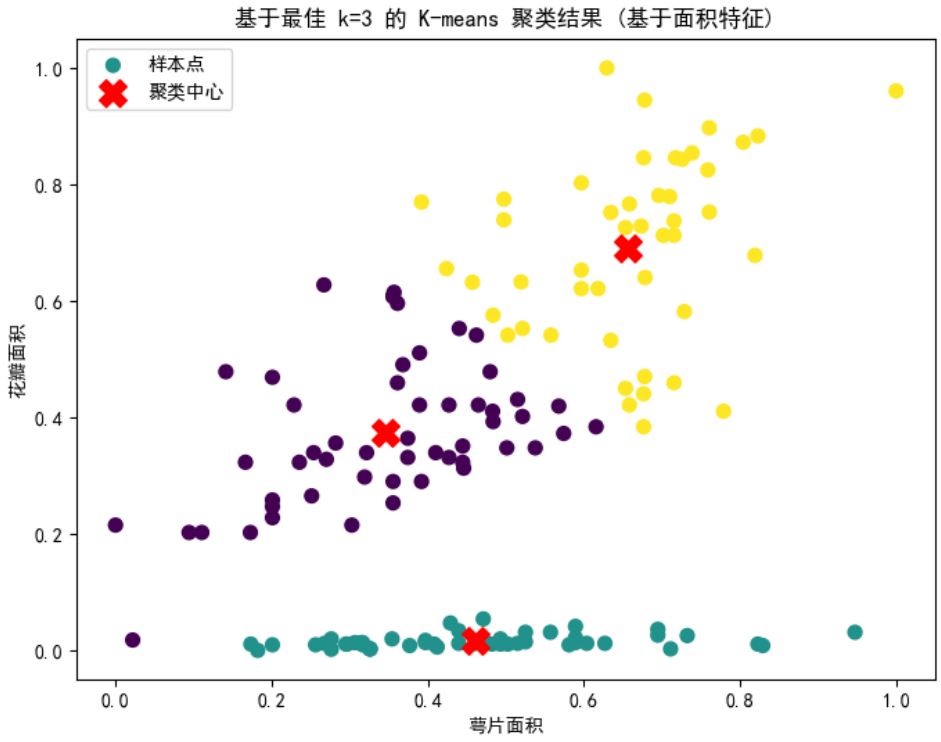
肘部图：



轮廓图：



聚类散点图:



最佳簇数和其轮廓系数:

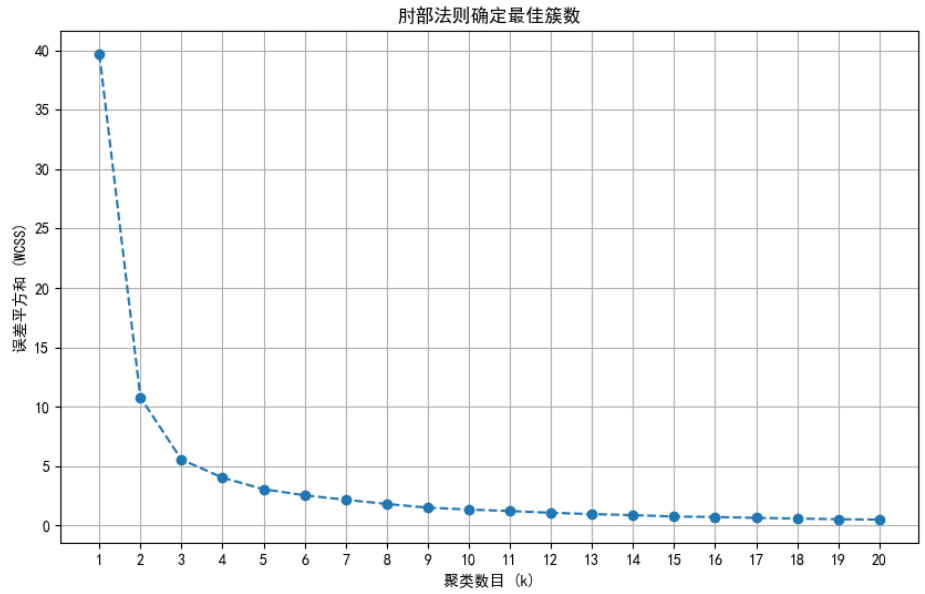
```
最佳簇数 k: 3
轮廓系数: 0.4103
```

分析:

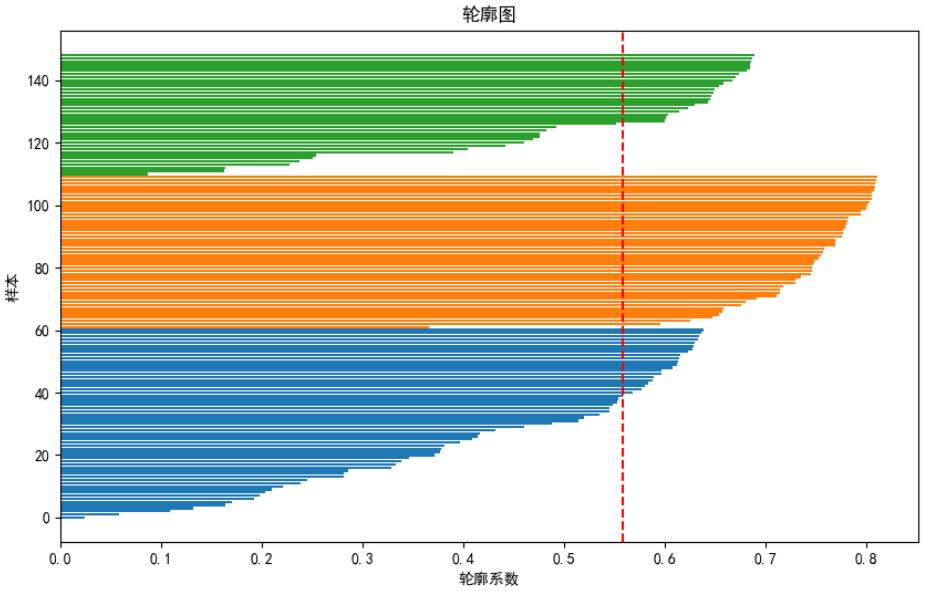
1. 面积特征（如sepal_area和petal_area）作为聚类的基础特征表现并不算太好，但能够区分出鸢尾花的不同种类。从散点图上看聚类结果较为清晰，簇的分离度较好。

B. 以主成分降维为特征进行分析：

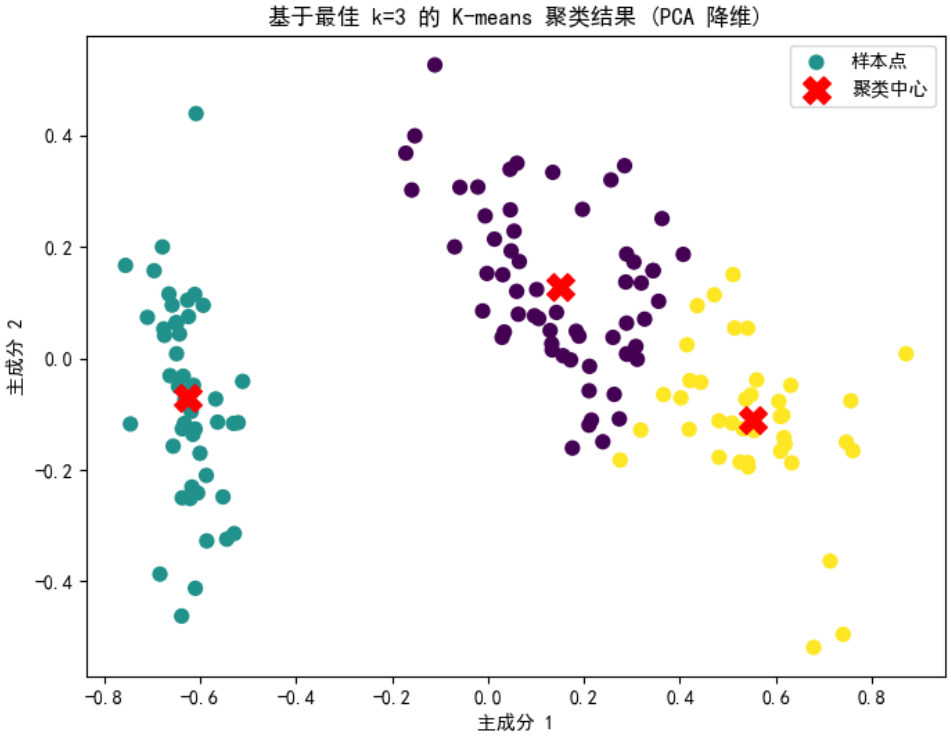
肘部图：



轮廓图：



聚类散点图：



最佳簇数和其轮廓系数:

```
最佳簇数 k: 3
轮廓系数: 0.5584
```

分析:

1. PCA降维将高维数据映射到二维空间，并保留了数据的主要变异性，能够很好地进行聚类分析。降维后的数据便于可视化和理解，聚类结果清晰且具有较好的簇分离度。
2. 与面积特征相比，PCA降维后的聚类可能会稍微损失一些信息，但由于PCA能够最大限度保留原始数据的变异性，聚类的效果依然较好。

综合分析：

综合来说，这两种方法都能有效进行聚类分析，基于面积特征的聚类可能在直接反映花卉的形态特征上略有优势，但基于PCA降维的聚类更加适合于高维数据的可视化与分析，能够简化数据维度并保持较好的聚类效果。

对于任务二，使用K-means++聚类算法进行聚类并分析

1. K-means++算法

对于实验原理和核心步骤不再赘述，只突出实验过程

实验过程:

（1）数据预处理

A. 加载数据：从CSV文件中加载银行营销数据集，使用Pandas的read_csv

函数读取数据，并对列名进行清理（去掉空格和引号）。

```
# 1. 加载数据
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)
```

B. 处理分类变量：提取数据集中的 分类变量（object 类型的列）并将其转化为虚拟变量（get_dummies），从而将分类变量转化为数值型特征（采用 one-hot 编码）。目标变量（y）的编码：将目标变量 y 中的 'yes' 和 'no' 标签编码为1和0，便于后续操作。同时将特征和目标进行分离。

```
# 提取分类变量并删除目标列
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# 将目标变量进行编码
data['y'] = data['y'].apply(lambda x: 1 if x == 'yes' else 0)  # 编码为 0 或 1

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']
```

（2）数据标准化

使用MinMaxScaler对数据进行标准化，将所有特征缩放到 [0, 1] 的范围。

```
# 数据标准化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

（3）PCA降维

使用 主成分分析（PCA）将高维数据降到二维，以便进行聚类结果的可视化。PCA 通过保留数据中的主要变异性，减少了维度，使得聚类结果更加直观。

```
# 3. PCA 降维
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

（4）K-means++ 聚类

A. K-means++初始化：智能地选择距离较远的点作为初始中心，避免了传统 K-means 中随机选择初始点可能导致的聚类效果差。

```
# 遍历不同的 k 值
for k in k_values:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=20, max_iter=300, random_state=42)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)
```

B. 肘部法则：计算不同K值下的 误差平方和（WCSS），并绘制肘部图。通过观察肘部图，确定聚类的最佳数目K。

```
# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

C. 根据肘部图确定的最佳 K 值，使用该 K 值对数据进行聚类。

```
# 确定最佳 k 值（通过肉眼观察肘部点）
optimal_k = 3  # 根据肘部图确定
print(f"最佳簇数 k: {optimal_k}")

# 使用最佳 k 值进行聚类
kmeans_optimal = KMeans(n_clusters=optimal_k, init='k-means++', n_init=20, max_iter=300, random_state=42)
kmeans_optimal.fit(X_pca)
labels_optimal = kmeans_optimal.labels_
```

（5）聚类效果评估

通过计算 轮廓系数 来评估聚类的效果。轮廓系数的值介于 [-1, 1] 之间，值越接近 1 表示聚类效果越好。计算得到的轮廓系数可以帮助判断聚类的合理性。

```
# 计算轮廓系数
silhouette = silhouette_score(X_pca, labels_optimal)
print(f"轮廓系数: {silhouette:.4f}")
```

（6）可视化聚类结果

A. 聚类结果散点图

```
# 可视化聚类结果
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels_optimal, cmap='viridis', s=50, label='样本点')
plt.scatter(kmeans_optimal.cluster_centers_[:, 0], kmeans_optimal.cluster_centers_[:, 1],
            c='red', s=200, marker='X', label='聚类中心')
plt.title(f'基于最佳 k={optimal_k} 的 K-means 聚类结果 (PCA 降维)')
plt.xlabel('主成分 1')
plt.ylabel('主成分 2')
plt.legend()
plt.show()
```

B. 肘部图

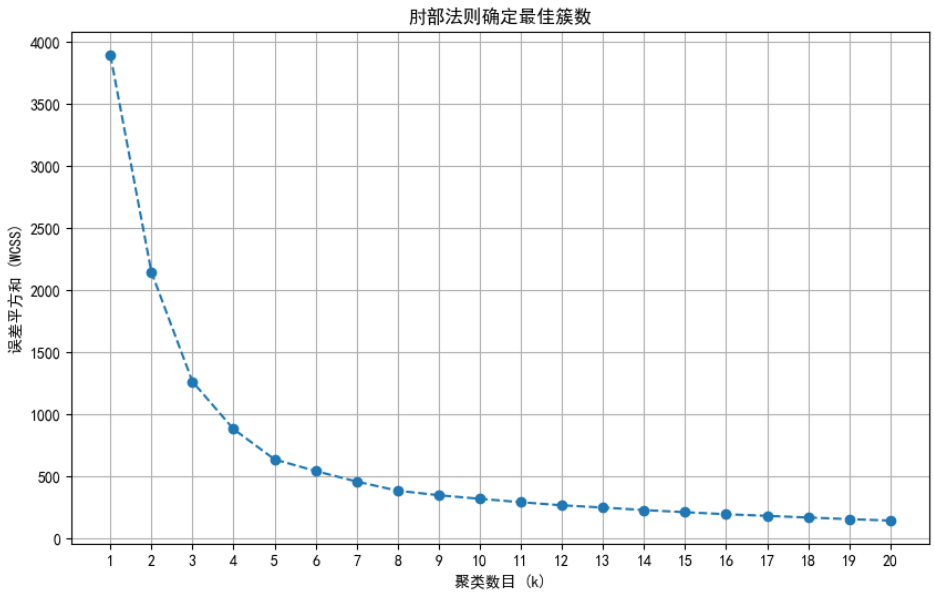
```
# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

C. 轮廓图

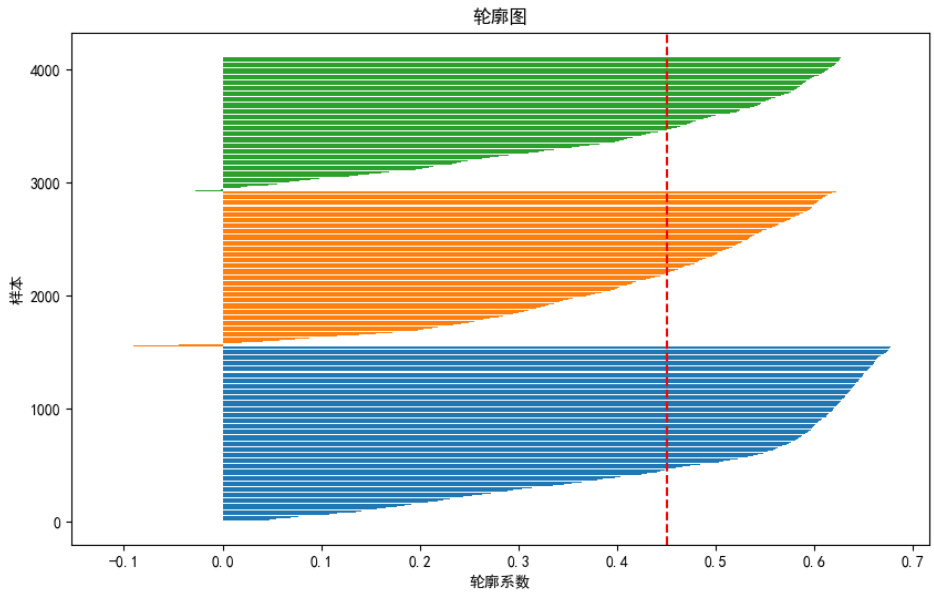
```
# 绘制轮廓图
silhouette_vals = silhouette_samples(X_pca, labels_optimal)
plt.figure(figsize=(10, 6))
y_lower, y_upper = 0, 0
for i in range(optimal_k):
    cluster_silhouette_vals = silhouette_vals[labels_optimal == i]
    cluster_silhouette_vals.sort()
    y_upper += len(cluster_silhouette_vals)
    plt.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none')
    y_lower += len(cluster_silhouette_vals)

plt.axvline(x=silhouette, color="red", linestyle="--")
plt.title("轮廓图")
plt.xlabel("轮廓系数")
plt.ylabel("样本")
plt.show()
```

（7）运行结果截图分析

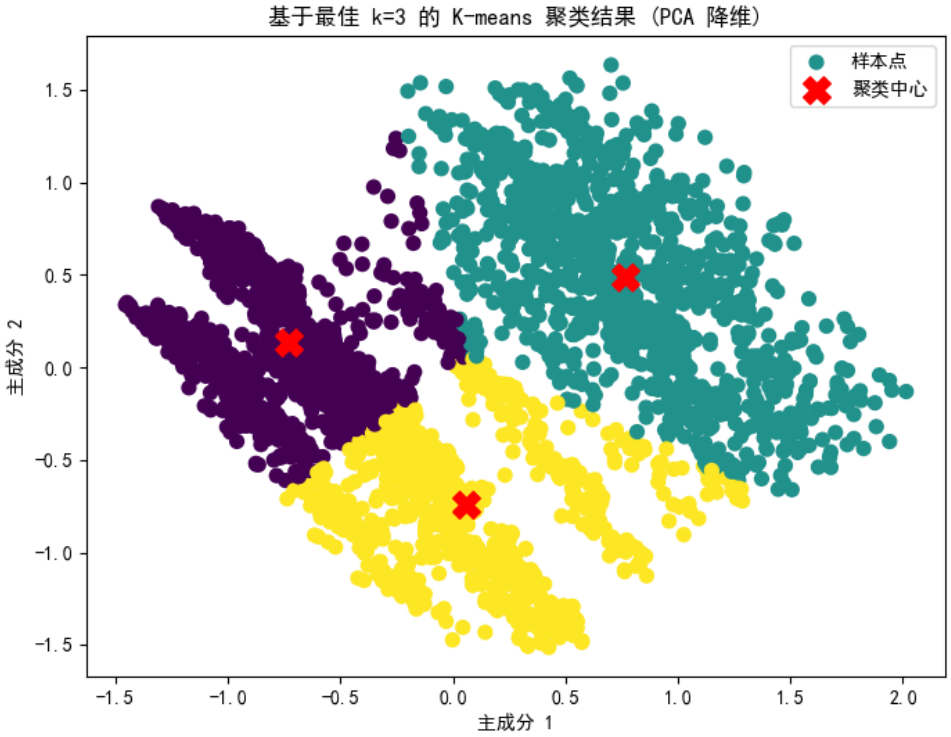
肘部图:


肘部法则确定最佳簇数
误差平方和（WCSS）
聚类数目（k）
0
500
1000
1500
2000
2500
3000
3500
4000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

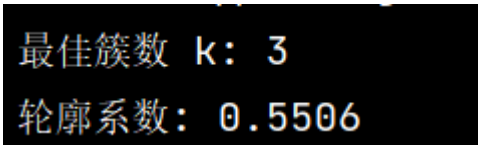
轮廓图：


轮廓图
样本
轮廓系数
0
1000
2000
3000
4000
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

聚类散点图：



最佳簇数和轮廓系数:



分析:

1. 聚类质量：轮廓系数为0.5506，说明聚类效果良好，簇之间有较好的分离，簇内的紧密度较高，聚类结果是可以接受的。
2. 聚类分析：聚类结果在PCA降维后的二维平面中分布较为清晰，聚类中心的标识明确，表明聚类算法在数据上表现稳定。

五、总结心得

本次实验我通过K-means++聚类算法 对给定数据集进行聚类，使用了PCA降维和肘部法则来确定最佳聚类数，并通过轮廓系数评估聚类质量。最终，通过聚类结果的可视化，展示了数据集的簇结构和分布情况，得到了理想的聚类效果。这个实验不仅加深了我对聚类算法的理解，还让我更加熟悉如何通过评估指标和可视化方法来分析聚类效果。

不足之处:

1. 没有尝试其他聚类算法，如DBSCAN或层次聚类进行比较分析，这类算法可能能更好地处理数据的噪声和复杂结构。
2. 虽然肘部法则给出了合理的k值（k=3），但也可以通过交叉验证等方式来进一步验证聚类数目，确保最佳聚类数的选择。

附录（所有代码）

Question1_K-Means.py

```python
# 导入必要的库
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import zscore
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.preprocessing import MinMaxScaler

# 配置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)

# 2. 特征工程：计算面积和比例特征
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']
# 计算比例特征
data['sepal_ratio'] = data['sepal_length'] / data['sepal_width']
data['petal_ratio'] = data['petal_length'] / data['petal_width']

# 数据清洗：去除异常值
z_scores = data[['sepal_area', 'petal_area']].apply(zscore)
data_cleaned = data[(z_scores.abs() <= 3).all(axis=1)]

# 构建特征矩阵
X = data_cleaned[['sepal_area', 'petal_area', 'sepal_ratio', 'petal_ratio']].values

# 数据标准化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# 定义聚类数目范围
k_values = range(1, 21)

# 存储每个 k 的误差平方和 (WCSS)
wcss = []

# 遍历不同的 k 值
```

```
for k in k_values:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=20, max_iter=300, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# 确定最佳 k 值（通过肉眼观察肘部点）
optimal_k = 3  # 根据肘部图确定
print(f"最佳簇数 k: {optimal_k}")

# 使用最佳 k 值进行聚类
kmeans_optimal = KMeans(n_clusters=optimal_k, init='k-means++', n_init=20, max_iter=300,
random_state=42)
kmeans_optimal.fit(X_scaled)
labels_optimal = kmeans_optimal.labels_

# 计算轮廓系数
silhouette = silhouette_score(X_scaled, labels_optimal)
print(f"轮廓系数: {silhouette:.4f}")

# 绘制轮廓图
silhouette_vals = silhouette_samples(X_scaled, labels_optimal)
plt.figure(figsize=(10, 6))
y_lower, y_upper = 0, 0
for i in range(optimal_k):
    cluster_silhouette_vals = silhouette_vals[labels_optimal == i]
    cluster_silhouette_vals.sort()
    y_upper += len(cluster_silhouette_vals)
    plt.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none')
    y_lower += len(cluster_silhouette_vals)

plt.axvline(x=silhouette, color="red", linestyle="--")
plt.title("轮廓图")
plt.xlabel("轮廓系数")
plt.ylabel("样本")
```

```
plt.show()

# 可视化聚类结果
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels_optimal, cmap='viridis', s=50, label='样本点')
plt.scatter(kmeans_optimal.cluster_centers_[:, 0], kmeans_optimal.cluster_centers_[:, 1],
                c='red', s=200, marker='X', label='聚类中心')
plt.title(f'基于最佳 k={optimal_k} 的 K-means 聚类结果 (基于面积特征)')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.legend()
plt.show()
```

Question1_K-means_PCA.py

```
# 导入必要的库
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import zscore
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.preprocessing import MinMaxScaler

# 配置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)

# 2. 数据清洗：去除异常值
z_scores = data.iloc[:, :-1].apply(zscore)
data_cleaned = data[(z_scores.abs() <= 3).all(axis=1)]

# 构建特征矩阵
X = data_cleaned.iloc[:, :-1].values

# 数据标准化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# 主成分分析 (PCA)
pca = PCA(n_components=2)
```

```
X_pca = pca.fit_transform(X_scaled)

# 定义聚类数目范围
k_values = range(1, 21)

# 存储每个 k 的误差平方和 (WCSS)
wcss = []

# 遍历不同的 k 值
for k in k_values:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=20, max_iter=300, random_state=42)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)

# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# 确定最佳 k 值（通过肉眼观察肘部点）
optimal_k = 3  # 根据肘部图确定
print(f"最佳簇数 k: {optimal_k}")

# 使用最佳 k 值进行聚类
kmeans_optimal = KMeans(n_clusters=optimal_k, init='k-means++', n_init=20, max_iter=300,
random_state=42)
kmeans_optimal.fit(X_pca)
labels_optimal = kmeans_optimal.labels_

# 计算轮廓系数
silhouette = silhouette_score(X_pca, labels_optimal)
print(f"轮廓系数: {silhouette:.4f}")

# 绘制轮廓图
silhouette_vals = silhouette_samples(X_pca, labels_optimal)
plt.figure(figsize=(10, 6))
y_lower, y_upper = 0, 0
for i in range(optimal_k):
    cluster_silhouette_vals = silhouette_vals[labels_optimal == i]
```

```
        cluster_silhouette_vals.sort()
        y_upper += len(cluster_silhouette_vals)
        plt.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none')
        y_lower += len(cluster_silhouette_vals)

plt.axvline(x=silhouette, color="red", linestyle="--")
plt.title("轮廓图")
plt.xlabel("轮廓系数")
plt.ylabel("样本")
plt.show()

# 可视化聚类结果
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels_optimal, cmap='viridis', s=50, label='样本点')
plt.scatter(kmeans_optimal.cluster_centers_[:, 0], kmeans_optimal.cluster_centers_[:, 1],
                c='red', s=200, marker='X', label='聚类中心')
plt.title(f'基于最佳 k={optimal_k} 的 K-means 聚类结果 (PCA 降维)')
plt.xlabel('主成分 1')
plt.ylabel('主成分 2')
plt.legend()
plt.show()
```

Question2_K-means.py

```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.preprocessing import MinMaxScaler
# 配置中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
matplotlib.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)

# 2. 数据预处理
# 提取分类变量并删除目标列
```

```
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# 将目标变量进行编码
data['y'] = data['y'].apply(lambda x: 1 if x == 'yes' else 0)  # 编码为 0 或 1

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']

# 数据标准化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# 3. PCA 降维
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 4. 定义聚类数目范围
k_values = range(1, 21)

# 存储每个 k 的误差平方和 (WCSS)
wcss = []

# 遍历不同的 k 值
for k in k_values:
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=20, max_iter=300, random_state=42)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)

# 绘制肘部图
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss, marker='o', linestyle='--')
plt.title('肘部法则确定最佳簇数')
plt.xlabel('聚类数目 (k)')
plt.ylabel('误差平方和 (WCSS)')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# 确定最佳 k 值（通过肉眼观察肘部点）
optimal_k = 3  # 根据肘部图确定
print(f"最佳簇数 k: {optimal_k}")
```

```
# 使用最佳 k 值进行聚类
kmeans_optimal = KMeans(n_clusters=optimal_k, init='k-means++', n_init=20, max_iter=300,
random_state=42)
kmeans_optimal.fit(X_pca)
labels_optimal = kmeans_optimal.labels_

# 计算轮廓系数
silhouette = silhouette_score(X_pca, labels_optimal)
print(f"轮廓系数: {silhouette:.4f}")

# 绘制轮廓图
silhouette_vals = silhouette_samples(X_pca, labels_optimal)
plt.figure(figsize=(10, 6))
y_lower, y_upper = 0, 0
for i in range(optimal_k):
    cluster_silhouette_vals = silhouette_vals[labels_optimal == i]
    cluster_silhouette_vals.sort()
    y_upper += len(cluster_silhouette_vals)
    plt.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none')
    y_lower += len(cluster_silhouette_vals)

plt.axvline(x=silhouette, color="red", linestyle="--")
plt.title("轮廓图")
plt.xlabel("轮廓系数")
plt.ylabel("样本")
plt.show()

# 可视化聚类结果
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels_optimal, cmap='viridis', s=50, label='样本点')
plt.scatter(kmeans_optimal.cluster_centers_[:, 0], kmeans_optimal.cluster_centers_[:, 1],
            c='red', s=200, marker='X', label='聚类中心')
plt.title(f'基于最佳 k={optimal_k} 的 K-means 聚类结果 (PCA 降维)')
plt.xlabel('主成分 1')
plt.ylabel('主成分 2')
plt.legend()
plt.show()
```