无监督学习

TA：刘尚格

Email： lshangge@smail.nju.edu.cn

1. 实验内容介绍

1. K-means算法实践
2. DBSCAN密度聚类算法实践
3. 层次聚类算法实践
4. 实际应用案例 - 客户分群

2. 实验步骤

导入相关库

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score
from sklearn.datasets import make_blobs, make_moons, make_circles
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')

# 设置中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False    # 用来正常显示负号

# 设置随机种子以确保结果可重现
np.random.seed(42)
```

2.1 第一部分：数据生成与可视化

数据可视化

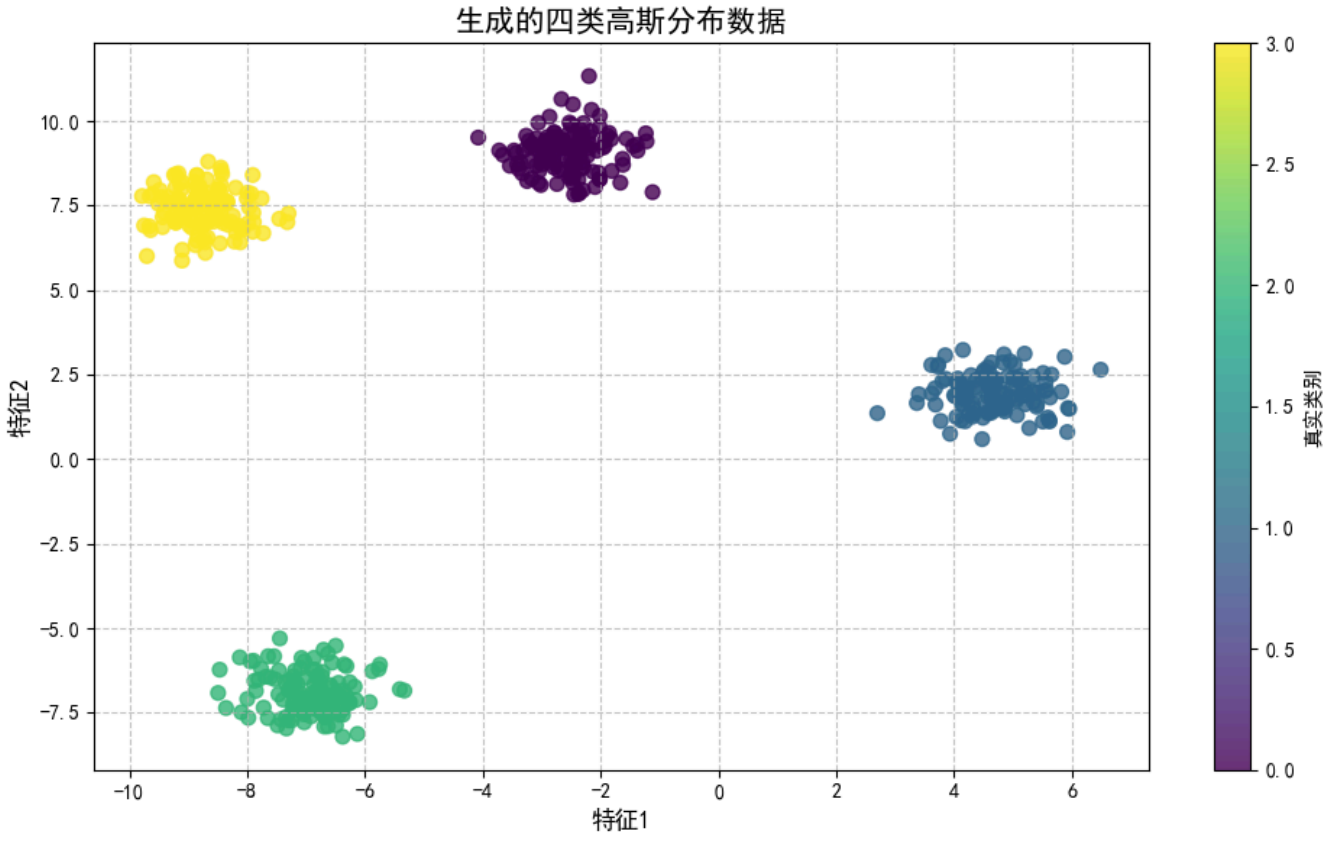
```
def plot_clusters(X, y_pred, centers=None, title="聚类结果"):
    """绘制聚类结果的辅助函数"""
```

```
    plt.figure(figsize=(10, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', marker='o', s=50,
alpha=0.8)

    if centers is not None:
        plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=200,
alpha=1, label='聚类中心')

    plt.title(title, fontsize=15)
    plt.xlabel('特征1', fontsize=12)
    plt.ylabel('特征2', fontsize=12)
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.colorbar(label='聚类标签')
    plt.tight_layout()
    plt.show()
```

生成多类高斯分布数据

```
# 1.1 生成多类高斯分布数据
print("生成四个类别的高斯分布数据...")
X_blobs, y_blobs = make_blobs(n_samples=500, centers=4, cluster_std=0.6,
random_state=42)

plt.figure(figsize=(10, 6))
plt.scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_blobs, cmap='viridis', marker='o',
s=50, alpha=0.8)
plt.title('生成的四类高斯分布数据', fontsize=15)
plt.xlabel('特征1', fontsize=12)
plt.ylabel('特征2', fontsize=12)
plt.colorbar(label='真实类别')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
生成四个类别的高斯分布数据...
```

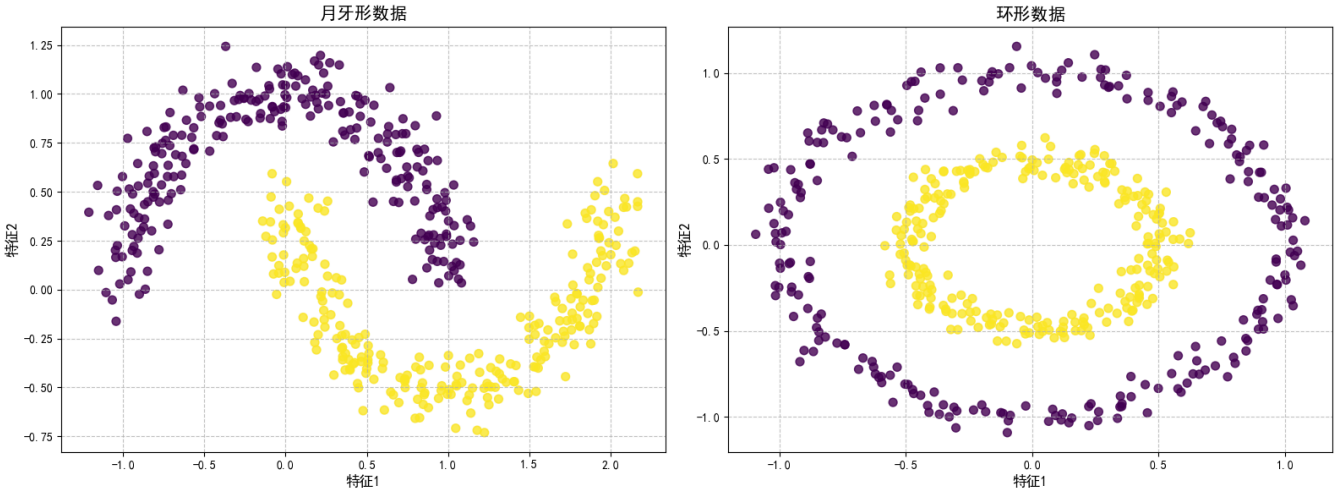


生成非球形数据

```
# 1.2 生成非球形数据 - 用于测试不同算法的适用性
print("生成月牙形数据...")
X_moons, y_moons = make_moons(n_samples=500, noise=0.1, random_state=42)

print("生成环形数据...")
X_circles, y_circles = make_circles(n_samples=500, noise=0.05, factor=0.5,
random_state=42)

# 可视化非球形数据
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].scatter(X_moons[:, 0], X_moons[:, 1], c=y_moons, cmap='viridis',
marker='o', s=50, alpha=0.8)
axes[0].set_title('月牙形数据', fontsize=15)
axes[0].set_xlabel('特征1', fontsize=12)
axes[0].set_ylabel('特征2', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.7)

axes[1].scatter(X_circles[:, 0], X_circles[:, 1], c=y_circles, cmap='viridis',
marker='o', s=50, alpha=0.8)
axes[1].set_title('环形数据', fontsize=15)
axes[1].set_xlabel('特征1', fontsize=12)
axes[1].set_ylabel('特征2', fontsize=12)
```

```
axes[1].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

```
生成月牙形数据...
生成环形数据...
```



2.2 第二部分：K-means算法实践

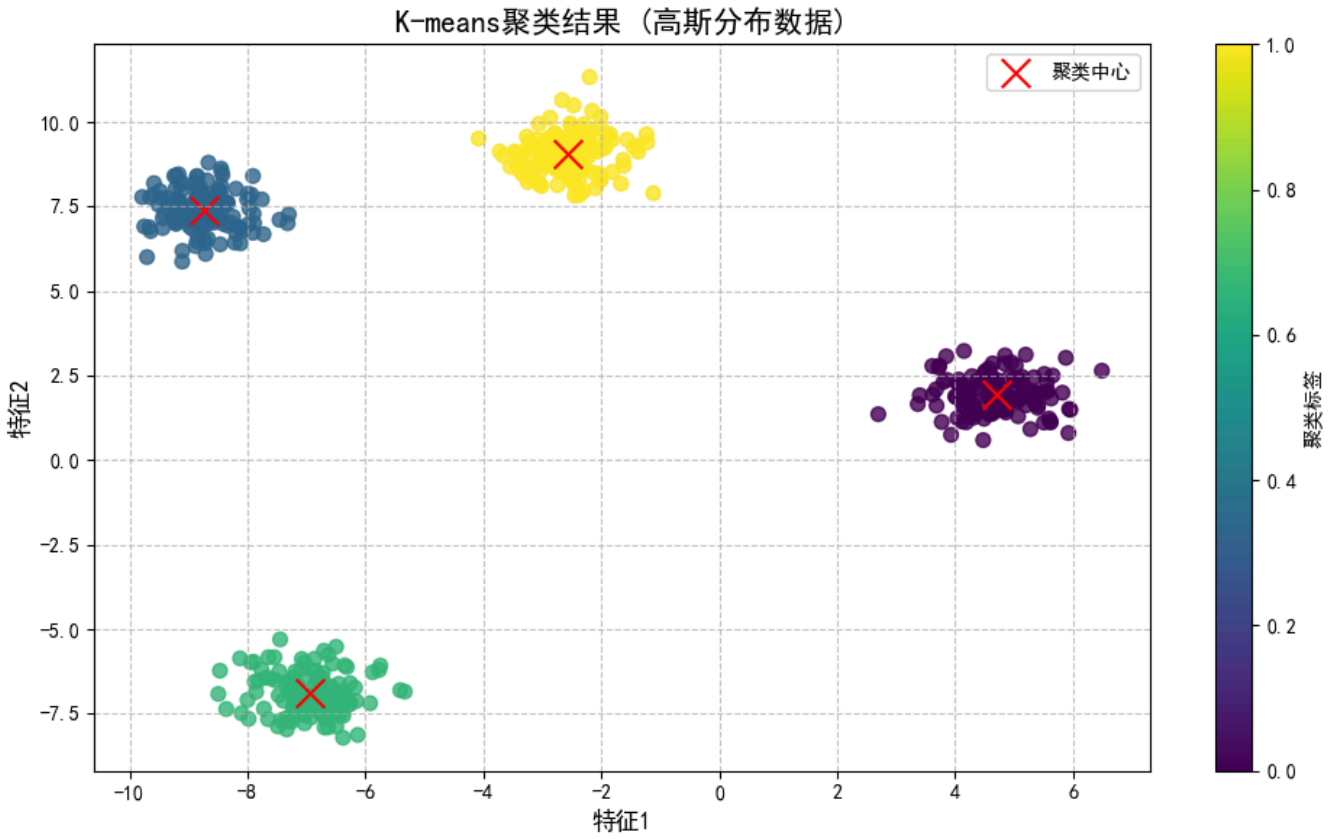
====== K-means聚类算法 ======
K-means是最常用的聚类算法之一，基于原型聚类思想。
步骤：

1. 指定聚类数k，初始化k个聚类中心
2. 将每个样本分配到最近的聚类中心
3. 重新计算每个聚类的中心（质心）
4. 重复步骤2-3直到收敛

2.1 基础实现 - 对高斯分布数据进行K-means聚类

```
# 2.1 基础实现 - 对高斯分布数据进行K-means聚类
print("\n对高斯分布数据进行K-means聚类...")
k = 4  # 聚类数量
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
y_pred_blobs = kmeans.fit_predict(X_blobs)
centers = kmeans.cluster_centers_

# 可视化聚类结果
plot_clusters(X_blobs, y_pred_blobs, centers, title="K-means聚类结果 (高斯分布数据)")
```

```
对高斯分布数据进行K-means聚类...
```

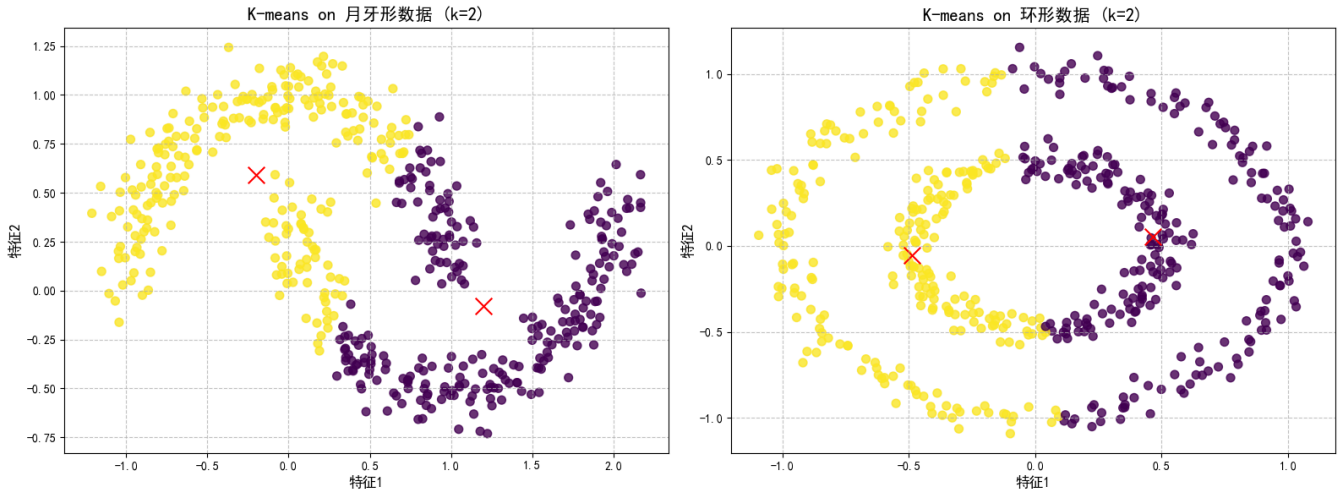


2.2 探索K-means的局限性

对非球形数据使用K-means

```
# 2.2 探索K-means的局限性

# 对月牙形数据使用K-means
kmeans_moons = KMeans(n_clusters=2, random_state=42, n_init=10)
y_pred_moons = kmeans_moons.fit_predict(X_moons)
centers_moons = kmeans_moons.cluster_centers_
```

```
# 对环形数据使用K-means
kmeans_circles = KMeans(n_clusters=2, random_state=42, n_init=10)
y_pred_circles = kmeans_circles.fit_predict(X_circles)
centers_circles = kmeans_circles.cluster_centers_

# 可视化结果
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred_moons, cmap='viridis',
marker='o', s=50, alpha=0.8)
axes[0].scatter(centers_moons[:, 0], centers_moons[:, 1], c='red', marker='x',
s=200, alpha=1)
axes[0].set_title('K-means on 月牙形数据 (k=2)', fontsize=15)
axes[0].set_xlabel('特征1', fontsize=12)
axes[0].set_ylabel('特征2', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.7)

axes[1].scatter(X_circles[:, 0], X_circles[:, 1], c=y_pred_circles,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[1].scatter(centers_circles[:, 0], centers_circles[:, 1], c='red',
marker='x', s=200, alpha=1)
axes[1].set_title('K-means on 环形数据 (k=2)', fontsize=15)
axes[1].set_xlabel('特征1', fontsize=12)
axes[1].set_ylabel('特征2', fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



结论：K-means对非球形分布的数据效果不佳，因为它假设数据呈球形分布。

2.3 探索不同的K值

========== K值选择方法 ==========

1. 肘部法则 (Elbow Method):

   肘部法则是一种确定最佳聚类数量的直观方法。其基本原理是：

   • 计算不同K值下的惯性值(Inertia)或总平方和误差(SSE, Sum of Squared Errors)

   • 惯性值代表样本到其最近聚类中心的距离平方和: Σ(样本点到其聚类中心的距离)^2

   • 随着K值增加，惯性值通常会持续下降（聚类越多，样本点与聚类中心越近）

   • 当图像呈现"肘部"形状时（下降速率突然变缓），对应的K值可能是最佳选择

   • 肘部位置表示增加更多聚类带来的边际收益已经开始显著降低

   • 数学解释：在肘部位置，聚类内方差与聚类数量达到了较好的平衡点

   肘部法则的优缺点：

   • 优点：简单直观，计算开销小

   • 缺点：在某些数据集上肘部不明显，可能产生主观判断

2. 轮廓系数 (Silhouette Score):

   轮廓系数是评估聚类质量的更为复杂的指标，它同时考虑了内聚度和分离度：

   • 轮廓值计算公式：$s(i) = \frac{b_i - a_i}{\max(a_i, b_i)}$

   其中:

   • a_i: 样本i与同一簇中其他样本的平均距离（簇内距离）- 衡量内聚度

   • b_i: 样本i与最近邻簇中所有样本的平均距离（簇间距离）- 衡量分离度

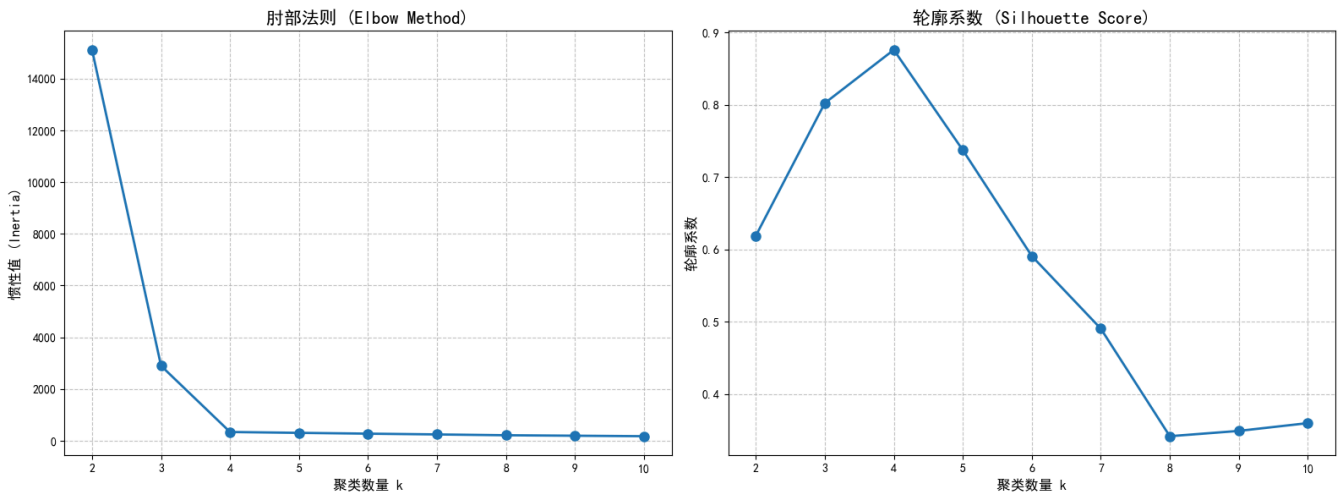
   • 轮廓系数是所有样本轮廓值的平均值

   • 取值范围：[-1, 1]

     - 接近+1: 样本远离相邻簇，聚类良好

```
# 2.3 探索不同的K值

inertia_values = []
silhouette_scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    y_pred = kmeans.fit_predict(X_blobs)
    inertia_values.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_blobs, y_pred))

# 绘制肘部图和轮廓系数图
```

```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].plot(k_values, inertia_values, 'o-', linewidth=2, markersize=8)
axes[0].set_title('肘部法则 (Elbow Method)', fontsize=15)
axes[0].set_xlabel('聚类数量 k', fontsize=12)
axes[0].set_ylabel('惯性值 (Inertia)', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.7)

axes[1].plot(k_values, silhouette_scores, 'o-', linewidth=2, markersize=8)
axes[1].set_title('轮廓系数 (Silhouette Score)', fontsize=15)
axes[1].set_xlabel('聚类数量 k', fontsize=12)
axes[1].set_ylabel('轮廓系数', fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



在本例中，我们可以观察：

- 肘部法则显示在K=5附近有一个明显的'肘部'
- 轮廓系数在K=4时达到最大值0.8757
- 两种方法给出了不同的最优K值，这时需要结合业务需求或进一步分析

2.3 第三部分：DBSCAN密度聚类算法实践

===== DBSCAN密度聚类算法 =====

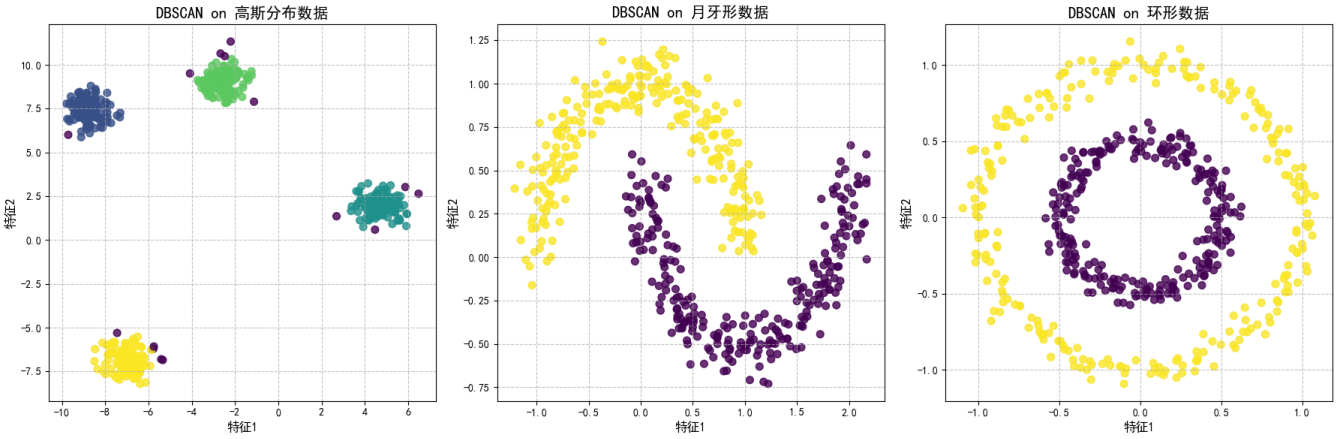
DBSCAN是一种基于密度的聚类算法，能够发现任意形状的聚类。
关键参数：

- eps：邻域半径

- min_samples：成为核心点所需的最小样本数

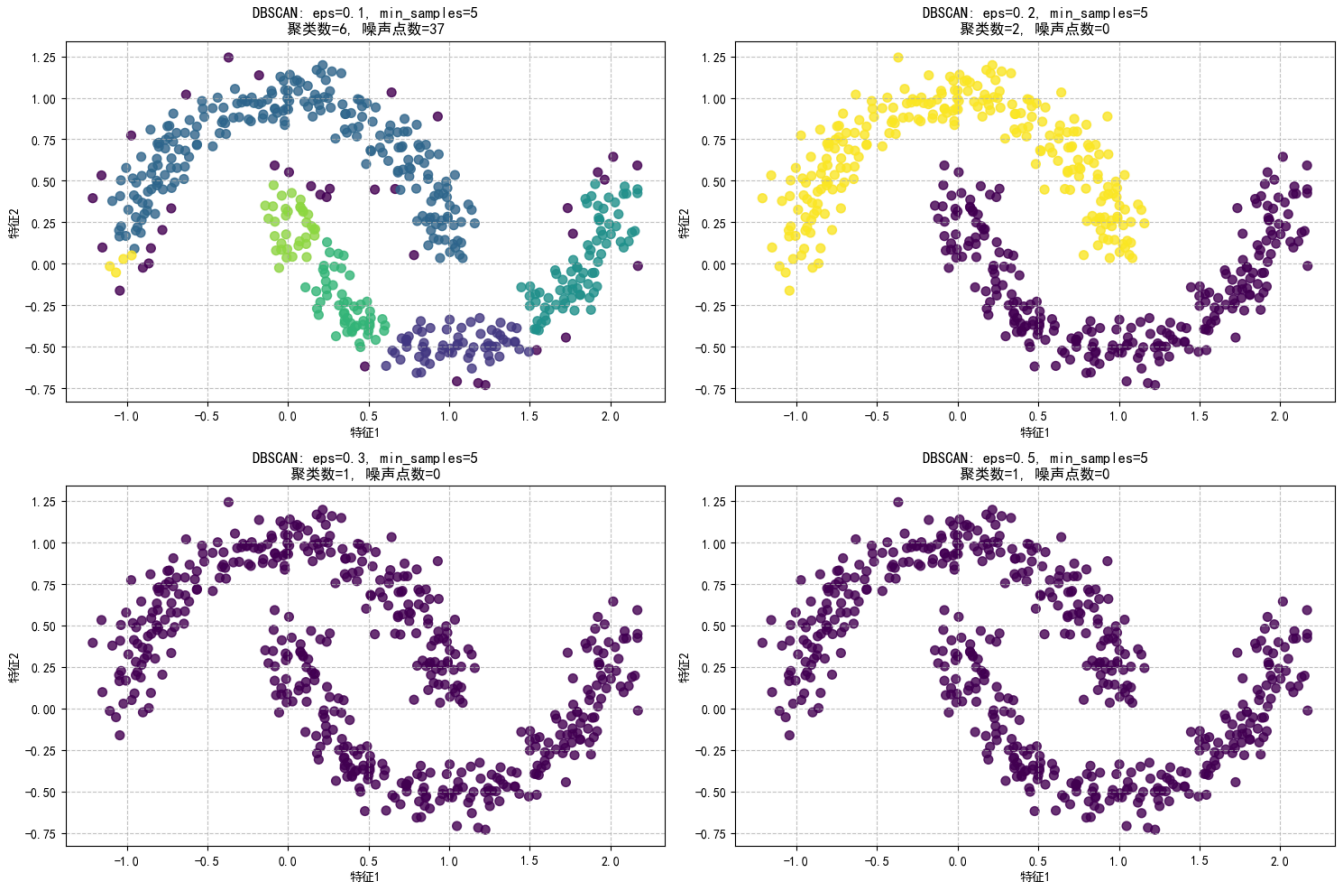
3.1 DBSCAN对不同形状数据的表现

```
# 对高斯分布数据使用DBSCAN
dbscan_blobs = DBSCAN(eps=0.5, min_samples=5)
y_pred_dbscan_blobs = dbscan_blobs.fit_predict(X_blobs)

# 对月牙形数据使用DBSCAN
dbscan_moons = DBSCAN(eps=0.2, min_samples=5)
y_pred_dbscan_moons = dbscan_moons.fit_predict(X_moons)

# 对环形数据使用DBSCAN
dbscan_circles = DBSCAN(eps=0.2, min_samples=5)
y_pred_dbscan_circles = dbscan_circles.fit_predict(X_circles)

# 可视化结果
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

axes[0].scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_pred_dbscan_blobs,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[0].set_title('DBSCAN on 高斯分布数据', fontsize=15)
axes[0].set_xlabel('特征1', fontsize=12)
axes[0].set_ylabel('特征2', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.7)

axes[1].scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred_dbscan_moons,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[1].set_title('DBSCAN on 月牙形数据', fontsize=15)
axes[1].set_xlabel('特征1', fontsize=12)
axes[1].set_ylabel('特征2', fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.7)

axes[2].scatter(X_circles[:, 0], X_circles[:, 1], c=y_pred_dbscan_circles,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[2].set_title('DBSCAN on 环形数据', fontsize=15)
axes[2].set_xlabel('特征1', fontsize=12)
axes[2].set_ylabel('特征2', fontsize=12)
axes[2].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



3.2 探索DBSCAN参数的影响

```
# 为月牙形数据测试不同的eps参数
eps_values = [0.1, 0.2, 0.3, 0.5]
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.flatten()

for i, eps in enumerate(eps_values):
    dbscan = DBSCAN(eps=eps, min_samples=5)
    y_pred = dbscan.fit_predict(X_moons)

    # 计算噪声点的数量
    n_noise = np.sum(y_pred == -1)
    n_clusters = len(set(y_pred)) - (1 if -1 in y_pred else 0)

    axes[i].scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred, cmap='viridis',
marker='o', s=50, alpha=0.8)
    axes[i].set_title(f'DBSCAN: eps={eps}, min_samples=5\n聚类数={n_clusters}, 噪
声点数={n_noise}', fontsize=12)
    axes[i].set_xlabel('特征1', fontsize=10)
    axes[i].set_ylabel('特征2', fontsize=10)
    axes[i].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



DBSCAN的优势：

1. 能发现任意形状的聚类
2. 能自动识别噪声点
3. 不需要预先指定聚类数量

DBSCAN的局限：

1. 对参数敏感
2. 处理不同密度的聚类效果不佳
3. 在高维空间效果可能不佳

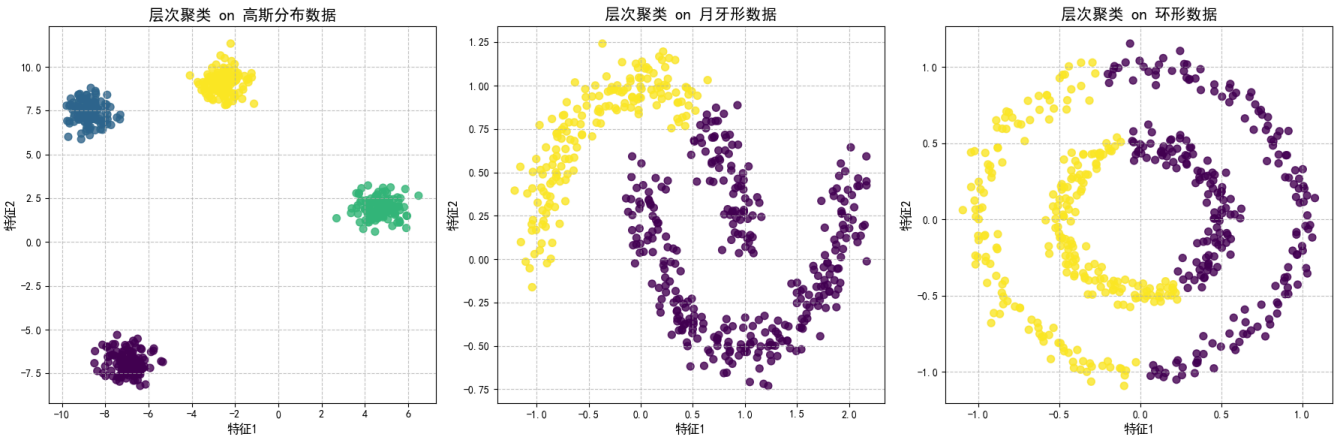
2.4 第四部分：层次聚类算法实践

===== 层次聚类算法 =====
层次聚类通过构建聚类的层次结构来分析数据。
两种主要类型：

- 凝聚式（自下而上）：从单个样本开始，逐步合并最相似的聚类
- 分裂式（自上而下）：从一个大聚类开始，逐步分裂

4.1 凝聚式层次聚类

```
# 对高斯分布数据应用层次聚类
agg_clustering = AgglomerativeClustering(n_clusters=4)
y_pred_agg_blobs = agg_clustering.fit_predict(X_blobs)

# 对月牙形数据应用层次聚类
agg_clustering_moons = AgglomerativeClustering(n_clusters=2)
y_pred_agg_moons = agg_clustering_moons.fit_predict(X_moons)

# 对环形数据应用层次聚类
agg_clustering_circles = AgglomerativeClustering(n_clusters=2)
y_pred_agg_circles = agg_clustering_circles.fit_predict(X_circles)

# 可视化结果
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

axes[0].scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_pred_agg_blobs,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[0].set_title('层次聚类 on 高斯分布数据', fontsize=15)
axes[0].set_xlabel('特征1', fontsize=12)
axes[0].set_ylabel('特征2', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.7)

axes[1].scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred_agg_moons,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[1].set_title('层次聚类 on 月牙形数据', fontsize=15)
axes[1].set_xlabel('特征1', fontsize=12)
axes[1].set_ylabel('特征2', fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.7)

axes[2].scatter(X_circles[:, 0], X_circles[:, 1], c=y_pred_agg_circles,
cmap='viridis', marker='o', s=50, alpha=0.8)
axes[2].set_title('层次聚类 on 环形数据', fontsize=15)
axes[2].set_xlabel('特征1', fontsize=12)
axes[2].set_ylabel('特征2', fontsize=12)
axes[2].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

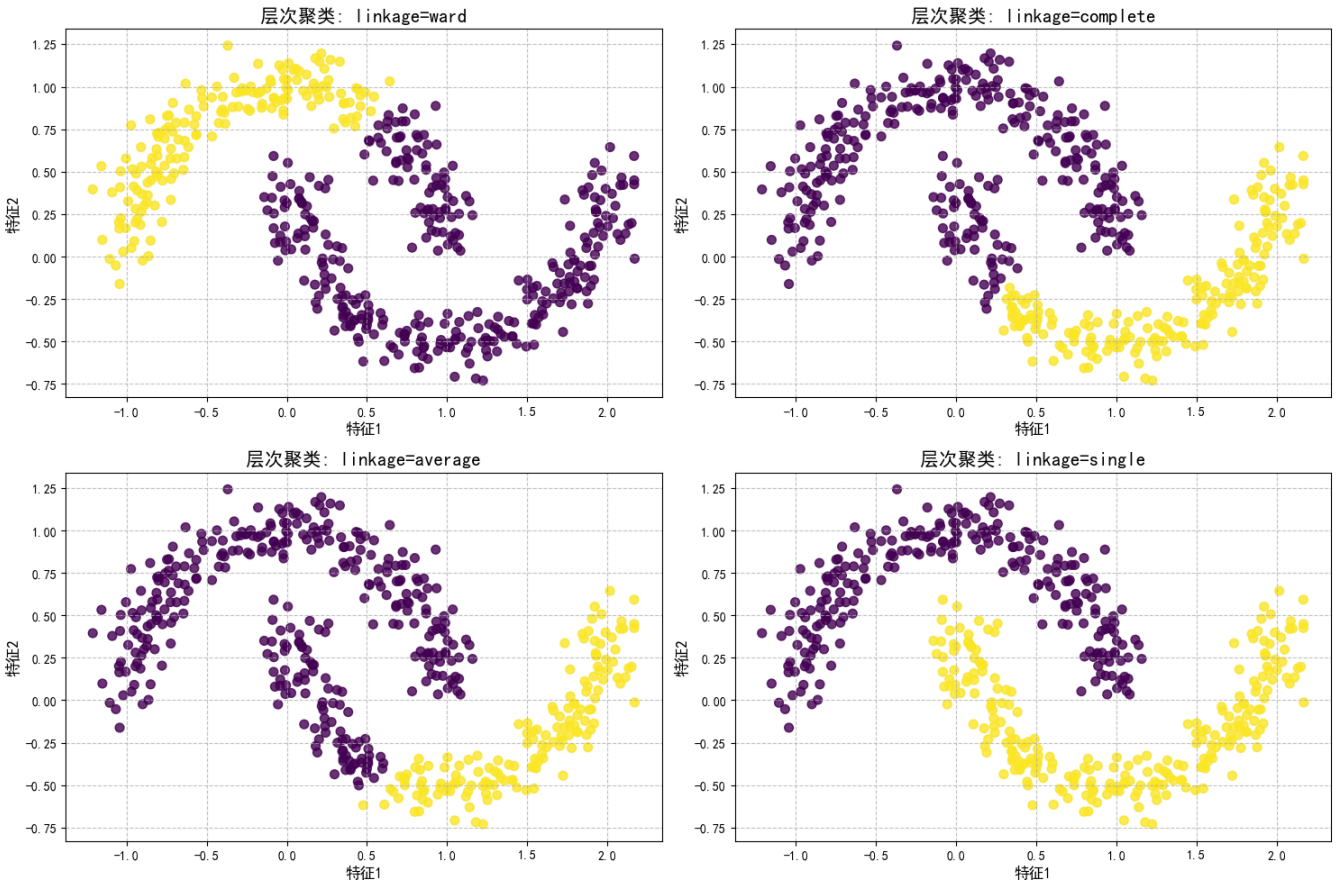


4.2 探索不同的链接方法

不同链接方法的说明：

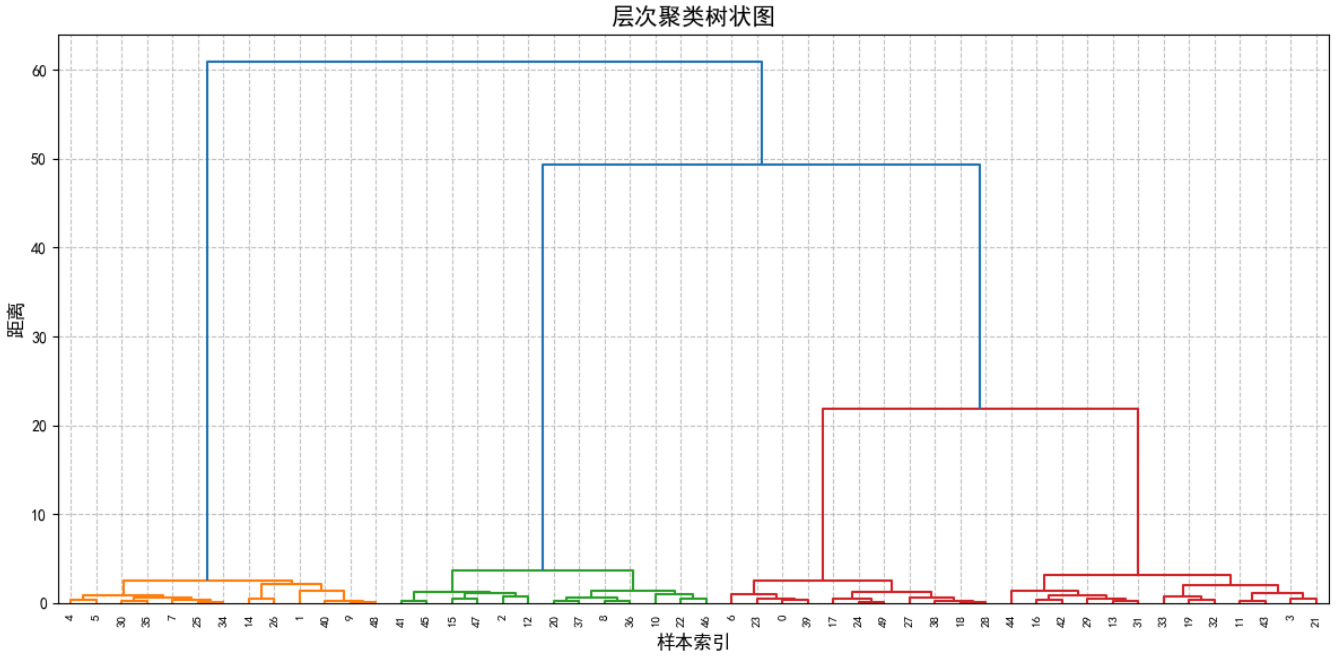
1. ward：最小化类内方差（默认），倾向于创建大小相等的聚类
2. complete：基于最远点的距离（完全链接）
3. average：基于所有点对之间的平均距离
4. single：基于最近点的距离（单链接）

```
linkage_types = ['ward', 'complete', 'average', 'single']
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.flatten()

for i, linkage in enumerate(linkage_types):
    agg_clustering = AgglomerativeClustering(n_clusters=2, linkage=linkage)
    y_pred = agg_clustering.fit_predict(X_moons)

    axes[i].scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred, cmap='viridis',
marker='o', s=50, alpha=0.8)
    axes[i].set_title(f'层次聚类: linkage={linkage}', fontsize=15)
    axes[i].set_xlabel('特征1', fontsize=12)
    axes[i].set_ylabel('特征2', fontsize=12)
    axes[i].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



4.3 绘制层次聚类树状图

```
from scipy.cluster.hierarchy import dendrogram, linkage

# 为了演示目的，使用较少的样本
X_sample = X_blobs[:50]

# 计算链接矩阵
Z = linkage(X_sample, method='ward')

plt.figure(figsize=(12, 6))
dendrogram(Z)
plt.title('层次聚类树状图', fontsize=15)
plt.xlabel('样本索引', fontsize=12)
plt.ylabel('距离', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



树状图解释：

1. 纵轴表示距离（或相似度）
2. 每次合并在树状图中由横线表示
3. 通过在特定高度切割树状图，可以得到不同数量的聚类

2.5 第五部分：实际应用案例

5.1 加载数据

我们将使用模拟的客户数据

特征：年龄、收入、消费频率、消费金额、忠诚度

```
np.random.seed(42)
n_customers = 500

age = np.random.normal(35, 10, n_customers)
income = np.random.normal(60000, 15000, n_customers)
frequency = np.random.normal(10, 5, n_customers)
spending = np.random.normal(500, 200, n_customers)
loyalty = np.random.normal(3, 1, n_customers)

# 添加一些相关性
income = income + age * 1000
spending = spending + income * 0.01 + frequency * 20
loyalty = loyalty + frequency * 0.1 - age * 0.01
```

```
# 将数据组合成一个数组
X_customers = np.column_stack([age, income, frequency, spending, loyalty])
df_customers = pd.DataFrame(X_customers, columns=['年龄', '收入', '消费频率', '消费金额', '忠诚度'])

print(df_customers.head())
print("\n数据统计信息：")
print(df_customers.describe())
```

```
          年龄             收入        消费频率          消费金额       忠诚度
0  39.967142  113859.804743  16.996777  2134.205806  3.624828
1  33.617357  122258.606595  14.623168  1904.812291  3.981625
2  41.476885   80498.371774  10.298152  1347.306978  2.822626
3  50.230299  118674.837114   6.765316  1821.379802  2.866267
4  32.658466   82898.827716  13.491117  1564.773684  2.128912

数据统计信息：
              年龄             收入        消费频率          消费金额       忠诚度
count  500.000000     500.000000  500.000000   500.000000  500.000000
mean    35.068380   95545.771702   10.542423  1672.943759    3.692042
std      9.812532   17020.808896    5.051232   281.734842    1.072720
min      2.587327   42177.193247   -4.481277   668.594118    0.571043
25%     27.996926   84815.560017    6.987852  1505.891263    2.991539
50%     35.127971   94912.368134   10.599029  1675.285171    3.697926
75%     41.367833  105766.954117   13.773692  1838.722514    4.422065
max     73.527315  158974.160431   23.008416  2485.666412    7.318738
```

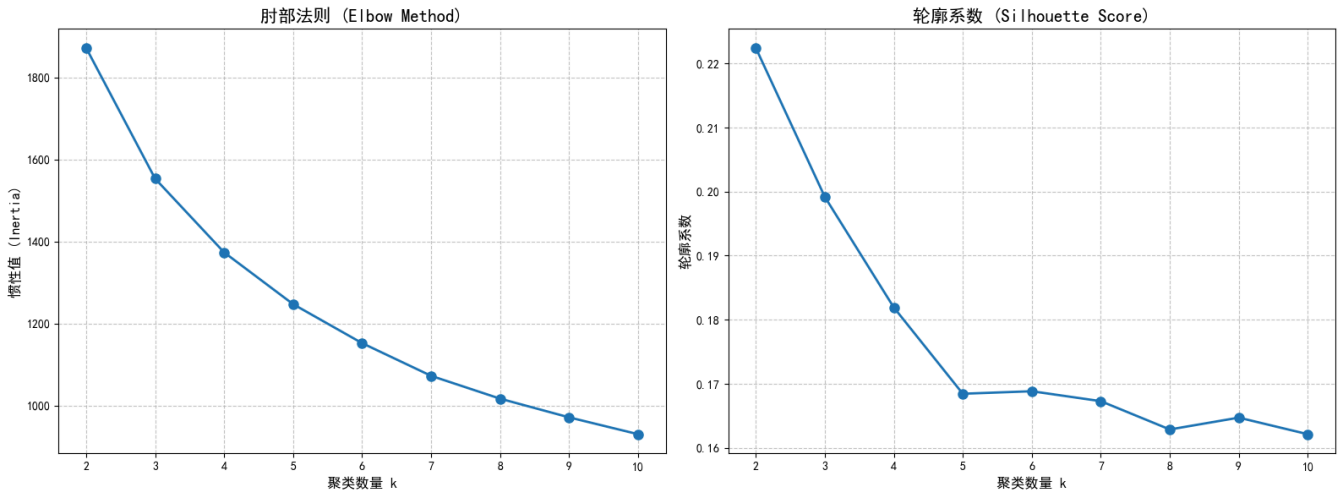
5.2 数据预处理

```
# 标准化数据
scaler = StandardScaler()
X_customers_scaled = scaler.fit_transform(X_customers)
df_customers_scaled = pd.DataFrame(X_customers_scaled, columns=['年龄', '收入', '消费频率', '消费金额', '忠诚度'])

print("\n标准化后的数据：")
print(df_customers_scaled.head())
```

```
标准化后的数据：
         年龄        收入        消费频率        消费金额       忠诚度
0  0.499735  1.077057  1.279058  1.638860 -0.062720
1 -0.148023  1.570994  0.808681  0.823827  0.270222
2  0.653748 -0.884944 -0.048407 -1.156985 -0.811290
3  1.546706  1.360231 -0.748508  0.527392 -0.770567
4 -0.245841 -0.743773  0.584342 -0.384327 -1.458625
```

5.3 确定最佳聚类数

```
inertia_values = []
silhouette_scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_customers_scaled)
    inertia_values.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_customers_scaled,
kmeans.labels_))

# 绘制肘部图和轮廓系数图
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

axes[0].plot(k_values, inertia_values, 'o-', linewidth=2, markersize=8)
axes[0].set_title('肘部法则 (Elbow Method)', fontsize=15)
axes[0].set_xlabel('聚类数量 k', fontsize=12)
axes[0].set_ylabel('惯性值 (Inertia)', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.7)

axes[1].plot(k_values, silhouette_scores, 'o-', linewidth=2, markersize=8)
axes[1].set_title('轮廓系数 (Silhouette Score)', fontsize=15)
axes[1].set_xlabel('聚类数量 k', fontsize=12)
axes[1].set_ylabel('轮廓系数', fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



5.4 执行聚类

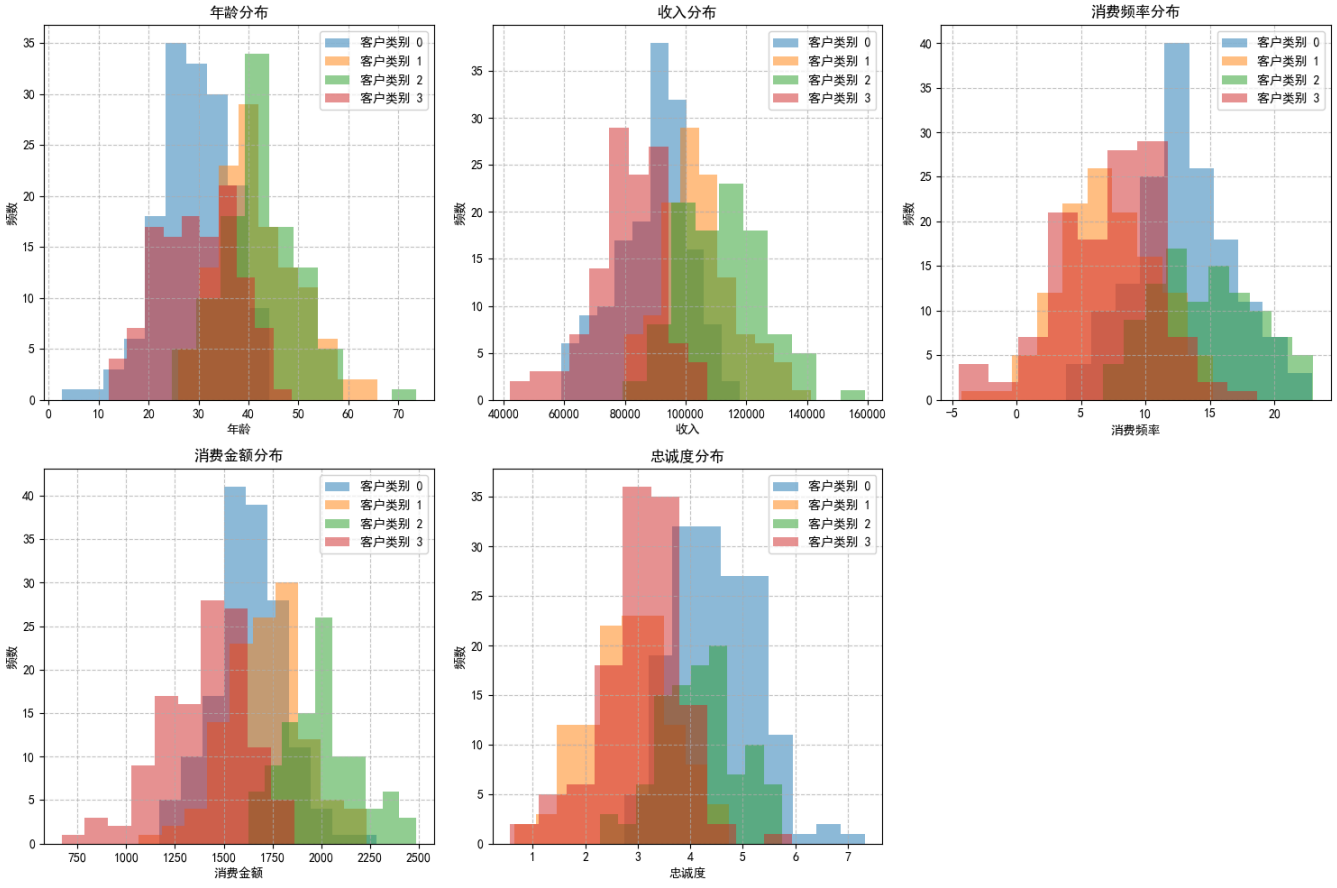
```
optimal_k = 4  # 根据上面的分析确定
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_customers_scaled)

# 添加聚类标签到原始数据
df_customers['客户类别'] = cluster_labels
```

5.5 分析聚类结果

```
# 计算每个聚类的中心
cluster_centers = kmeans.cluster_centers_

# 将聚类中心转换回原始数据范围
cluster_centers_original = scaler.inverse_transform(cluster_centers)
df_centers = pd.DataFrame(cluster_centers_original, columns=['年龄', '收入', '消费
频率', '消费金额', '忠诚度'])
df_centers.index = ['客户类别 ' + str(i) for i in range(optimal_k)]

print("\n聚类中心：")
print(df_centers)

# 统计每个聚类的大小
cluster_sizes = df_customers['客户类别'].value_counts().sort_index()
print("\n每个客户类别的人数：")
print(cluster_sizes)

# 可视化每个特征在不同聚类中的分布
plt.figure(figsize=(15, 10))

features = ['年龄', '收入', '消费频率', '消费金额', '忠诚度']
for i, feature in enumerate(features):
    plt.subplot(2, 3, i+1)
    for cluster in range(optimal_k):
        plt.hist(df_customers[df_customers['客户类别'] == cluster][feature],
                 alpha=0.5, label=f'客户类别 {cluster}')
    plt.title(f'{feature}分布', fontsize=12)
    plt.xlabel(feature, fontsize=10)
    plt.ylabel('频数', fontsize=10)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()

plt.tight_layout()
plt.show()
```

```
聚类中心：
                年龄          收入        消费频率          消费金额        忠诚度
```

```
客户类别 0   29.224442    89590.616295  12.965585  1629.791636  4.493678
客户类别 1   41.931358   104348.076548   7.132181  1715.363134  2.821339
客户类别 2   42.232302   111430.738275  14.586247  1999.697893  4.186408
客户类别 3   29.599422    80703.157883   7.312917  1403.922439  3.091862

每个客户类别的人数：
客户类别
0    157
1    121
2    103
3    119
Name: count, dtype: int64
```



思考：使用聚类算法分析真实数据集

数据集选择

可以选择以下数据集之一（或自选其他合适的数据集）：

1. 鸢尾花数据集 (Iris Dataset) - 简单经典
2. 葡萄酒数据集 (Wine Dataset) - 多特征分类
3. 电商客户数据集 - 市场细分应用

4. 信用卡交易数据集 - 异常检测应用

实施步骤

1. **数据加载与探索**：
   - 加载选择的数据集
   - 进行初步的数据分析和可视化
   - 处理缺失值和异常值
2. **数据预处理**：
   - 特征选择和/或特征工程
   - 数据标准化/归一化
   - 降维可视化（可选）
3. **聚类算法应用**：
   - 使用至少3种不同的聚类算法（如K-means、DBSCAN、层次聚类等）
   - 确定最优的参数（如聚类数k）
   - 可视化聚类结果
4. **聚类评估**：
   - 使用内部评价指标评估聚类结果
   - 如果有真实标签，也使用外部评价指标
   - 比较不同算法的性能
5. **结果分析与解释**：
   - 分析每个聚类的特点
   - 结合领域知识解释聚类结果的意义
   - 提出基于聚类结果的实际应用建议