维度约简

TA：刘尚格

Email： lshangge@smail.nju.edu.cn

1. 实验内容介绍

1. 基于鸢尾花数据集的PCA（主成分分析）实现
2. t-SNE可视化技术
3. 实际数据集上的应用

2. 实验步骤

导入相关库

```
# 导入相关库

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits, load_iris, fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE, LocallyLinearEmbedding
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import time
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

# 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，保证结果可复现
np.random.seed(42)
```

2.1 第一部分：PCA（主成分分析）实现

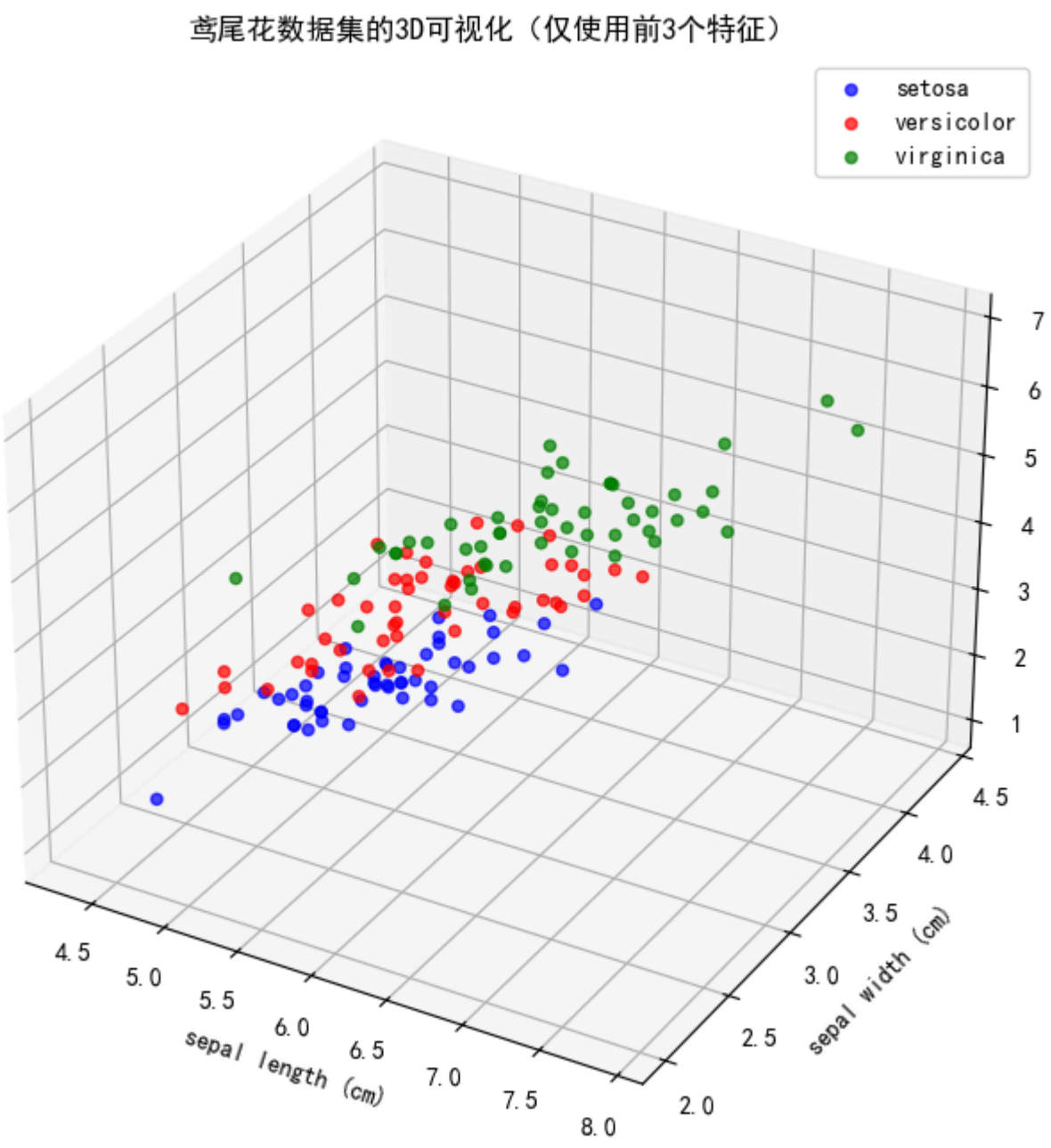
在本节中我们在鸢尾花数据集上进行实验。首先加载数据集并查看其基本信息

```
# 加载鸢尾花数据集
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names

print(f"数据集形状: {X.shape}")
print(f"特征名称: {feature_names}")
print(f"目标类别: {target_names}")
```

```
数据集形状: (150, 4)
特征名称: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
目标类别: ['setosa' 'versicolor' 'virginica']
```

可视化数据集分布

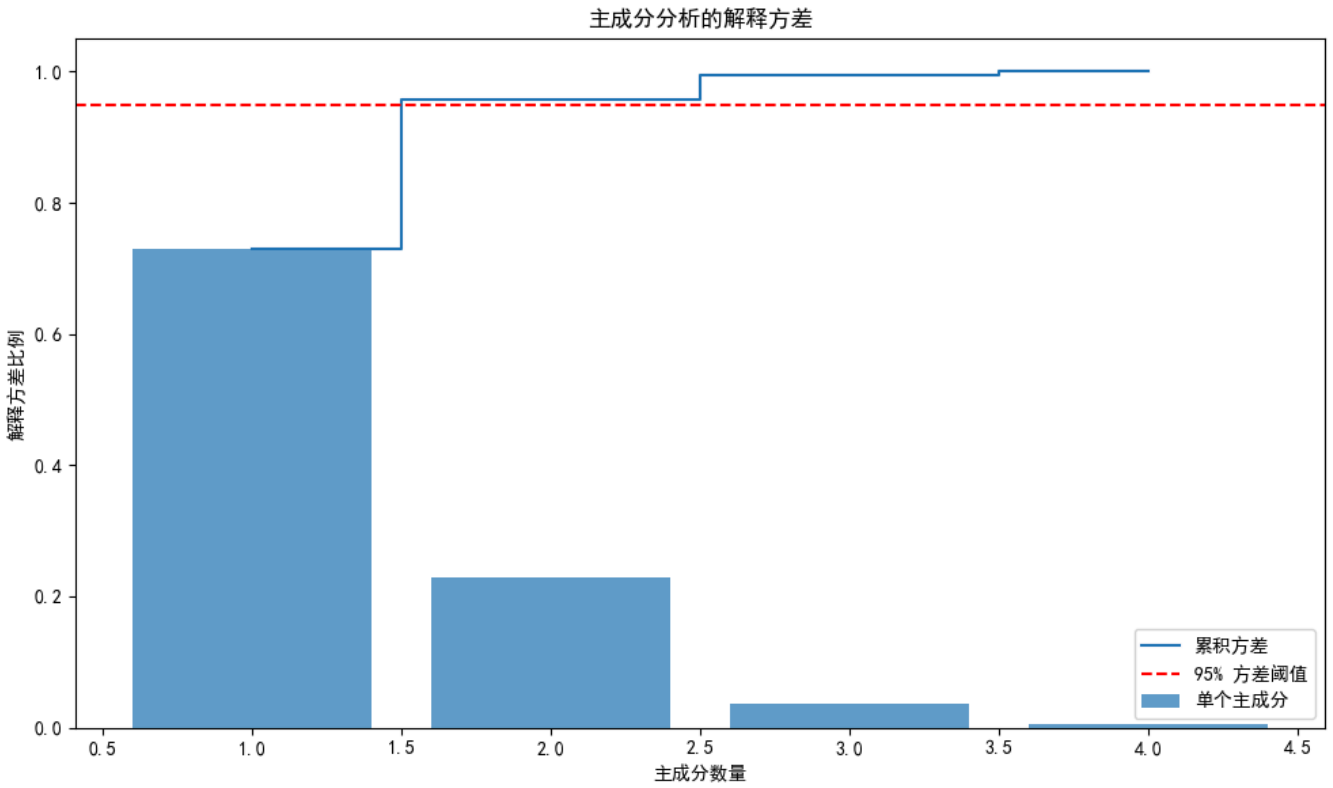
```
# 创建3D数据可视化（选择前3个特征）
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# 为每个类别绘制不同颜色的点
colors = ['blue', 'red', 'green']
for i, color in enumerate(colors):
    ax.scatter(X[y==i, 0], X[y==i, 1], X[y==i, 2],
               color=color, label=target_names[i], alpha=0.7)

ax.set_xlabel(feature_names[0])
ax.set_ylabel(feature_names[1])
ax.set_zlabel(feature_names[2])
ax.set_title('鸢尾花数据集的3D可视化（仅使用前3个特征）')
ax.legend()
plt.show()
```




```
# 使用PCA进行维度约简

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

# 应用PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# 查看各主成分解释的方差比例
```

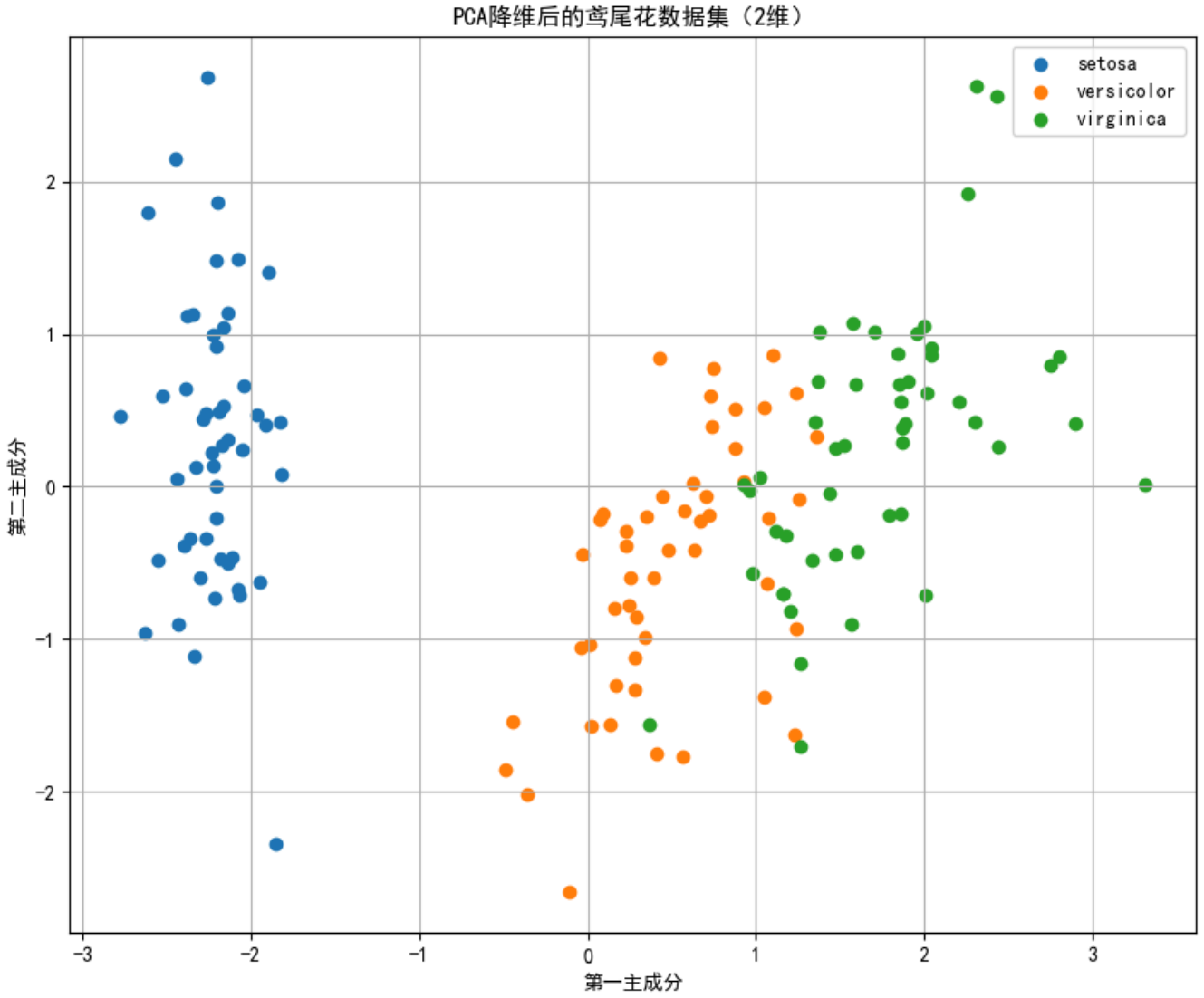
```
print("各主成分解释的方差比例:")
print(pca.explained_variance_ratio_)

# 累积方差比例
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
print("\n累积方差比例:")
print(cumulative_variance)

# 可视化解释方差比例
plt.figure(figsize=(10, 6))
plt.bar(range(1, len(pca.explained_variance_ratio_) + 1),
pca.explained_variance_ratio_, alpha=0.7, label='单个主成分')
plt.step(range(1, len(cumulative_variance) + 1), cumulative_variance,
where='mid', label='累积方差')
plt.axhline(y=0.95, color='r', linestyle='--', label='95% 方差阈值')
plt.xlabel('主成分数量')
plt.ylabel('解释方差比例')
plt.title('主成分分析的解释方差')
plt.legend()
plt.tight_layout()
plt.show()
```

```
各主成分解释的方差比例:
[0.72962445 0.22850762 0.03668922 0.00517871]

累积方差比例:
[0.72962445 0.95813207 0.99482129 1.        ]
```



从图中可以看出，前两个主成分就能解释约95%的方差。所以我们使用PCA将数据降到2维并可视化

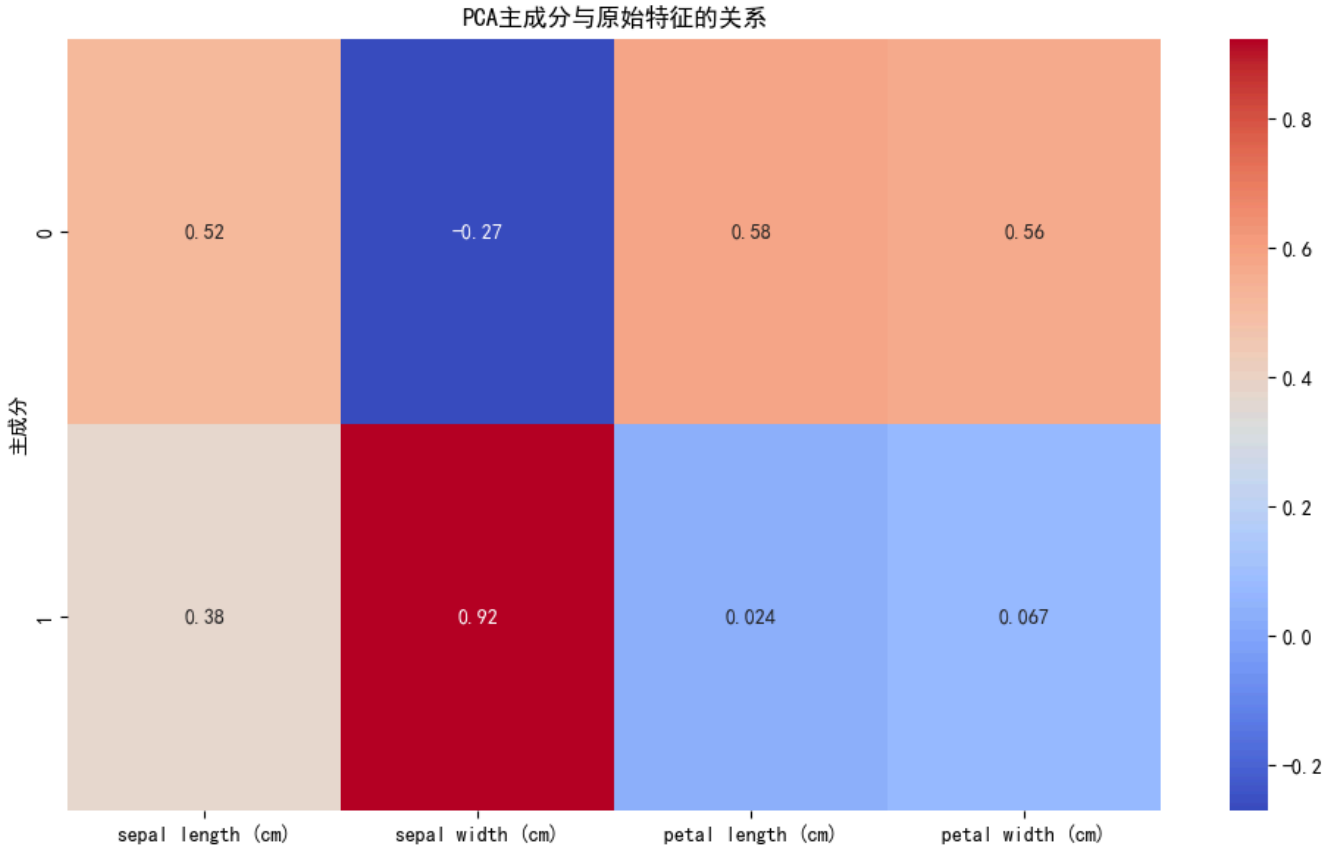
```
# 使用PCA降至2维
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X_scaled)

# 可视化降维后的数据
plt.figure(figsize=(10, 8))
for i, target_name in enumerate(target_names):
    plt.scatter(X_pca_2d[y==i, 0], X_pca_2d[y==i, 1], label=target_name)
plt.xlabel('第一主成分')
plt.ylabel('第二主成分')
plt.title('PCA降维后的鸢尾花数据集（2维）')
plt.legend()
plt.grid(True)
plt.show()
```



探究主成分特征与原始特征的关系

```
# 查看PCA的特征向量（主成分方向）
print("\nPCA的特征向量（主成分方向）:")
print(pca_2d.components_)

# 可视化主成分与原始特征的关系
plt.figure(figsize=(10, 6))
components = pd.DataFrame(pca_2d.components_, columns=feature_names)
sns.heatmap(components, annot=True, cmap='coolwarm')
plt.title('PCA主成分与原始特征的关系')
plt.ylabel('主成分')
plt.tight_layout()
plt.show()
```

```
PCA的特征向量（主成分方向）：
[[ 0.52106591 -0.26934744  0.5804131   0.56485654]
 [ 0.37741762  0.92329566  0.02449161  0.06694199]]
```



观察上面的热图，我们可以理解每个主成分如何与原始特征相关：

- 第一主成分主要由花瓣长度和花瓣宽度组成
- 第二主成分主要涉及花萼长度和花萼宽度

这种分析帮助我们理解降维后的特征在原始数据中的含义。

2.2 第二部分：t-SNE可视化技术

t-SNE(t-Distributed Stochastic Neighbor Embedding)是一种非线性降维技术，特别适合数据可视化。

t-SNE的特点：

1. 擅长保持数据的局部结构
2. 对于可视化高维数据聚类效果很好
3. 非线性方法，能捕捉复杂的数据关系
4. 计算成本较高，不适合大型数据集

本节我们使用一个更复杂的数据集—MNIST手写数字数据集的一个子集进行实验

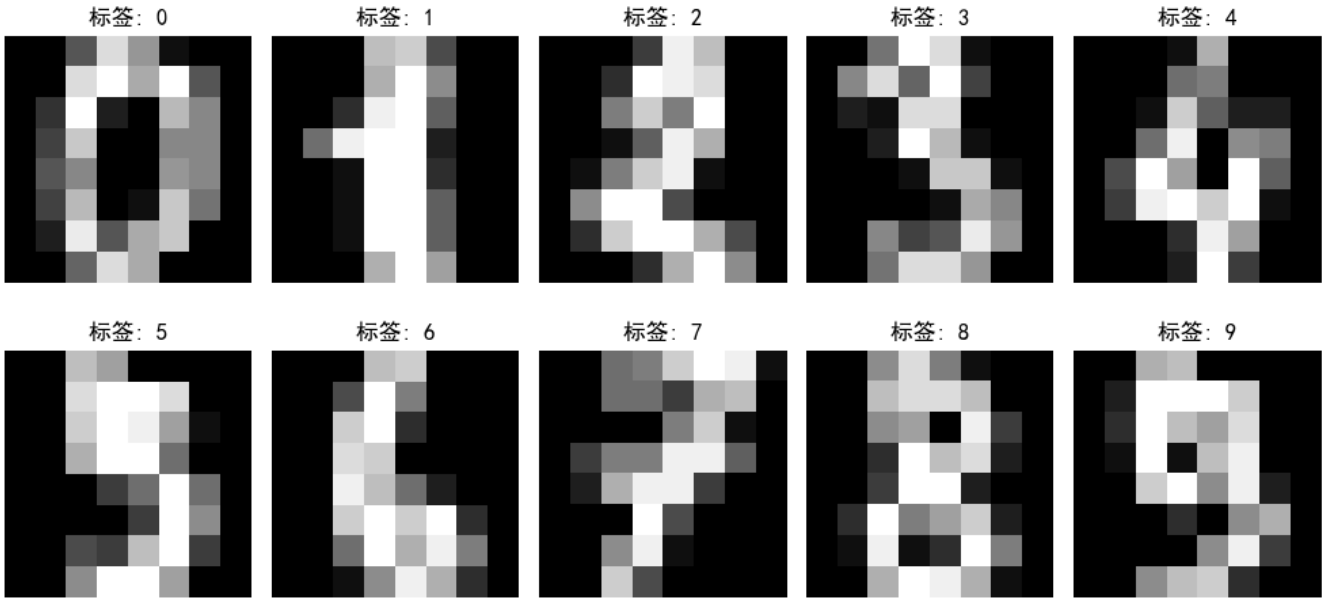
加载数据集并查看基本信息

```
# 加载手写数字数据集
digits = load_digits()
X_digits = digits.data
y_digits = digits.target

print(f"数据集形状: {X_digits.shape}")
print(f"目标类别: {np.unique(y_digits)}")
```

```
数据集形状: (1797, 64)
目标类别: [0 1 2 3 4 5 6 7 8 9]
```

可视化部分数据

```
# 显示一些数字示例
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(digits.images[i], cmap='gray')
    plt.title(f'标签: {y_digits[i]}')
    plt.axis('off')
plt.tight_layout()
plt.show()
```



接下来应用t-SNE将64维的手写数字数据降到2维

数据预处理

```
# 为了演示，我们选取部分数据（t-SNE计算较慢）
n_samples = 1000
indices = np.random.choice(len(X_digits), n_samples, replace=False)
X_digits_subset = X_digits[indices]
y_digits_subset = y_digits[indices]

# 数据标准化
X_digits_scaled = StandardScaler().fit_transform(X_digits_subset)
```

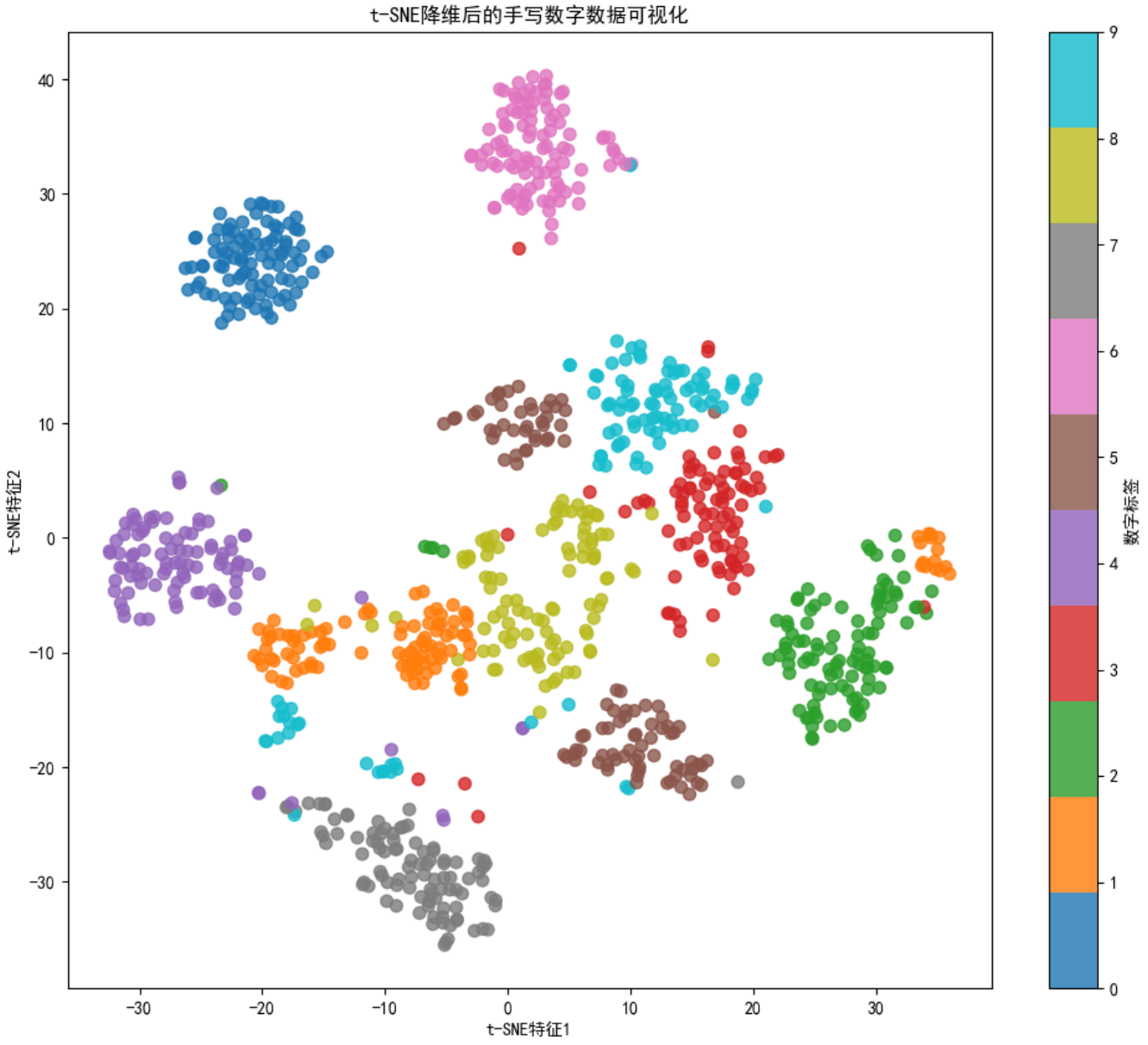
使用t-SNE降维至二维

```
# 应用t-SNE
time_start = time.time()
tsne = TSNE(n_components=2, random_state=42, perplexity=40)
X_tsne = tsne.fit_transform(X_digits_scaled)
time_end = time.time()
print(f"t-SNE计算耗时: {time_end - time_start:.2f}秒")
```

```
t-SNE计算耗时: 1.33秒
```

可视化t-SNE结果

```

plt.figure(figsize=(12, 10))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits_subset,
                      cmap='tab10', alpha=0.8, s=50)
plt.colorbar(scatter, label='数字标签')
plt.title('t-SNE降维后的手写数字数据可视化')
plt.xlabel('t-SNE特征1')
plt.ylabel('t-SNE特征2')
plt.show()
```



下面我们可以对比PCA和t-SNE的结果

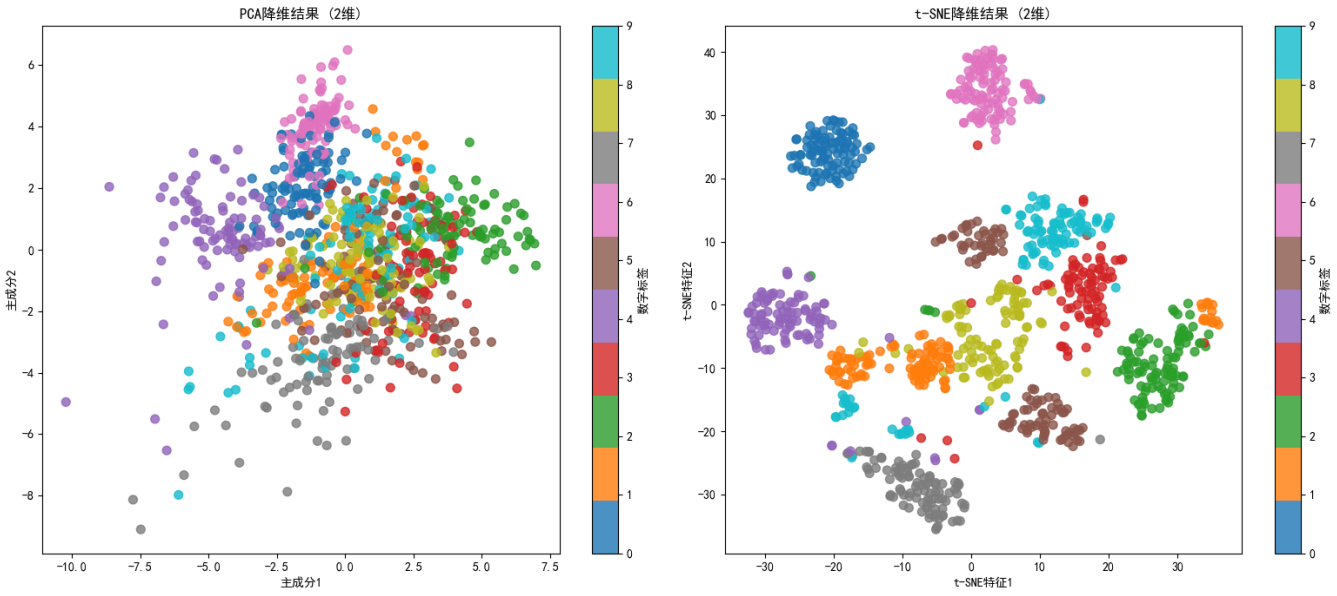
使用PCA进行降维

```
# 使用PCA降至2维
pca_2d_digits = PCA(n_components=2)
X_pca_2d_digits = pca_2d_digits.fit_transform(X_digits_scaled)
```

可视化两种降维方法的结果

```
# 创建对比图
plt.figure(figsize=(16, 7))

plt.subplot(1, 2, 1)
scatter1 = plt.scatter(X_pca_2d_digits[:, 0], X_pca_2d_digits[:, 1],
                       c=y_digits_subset, cmap='tab10', alpha=0.8, s=50)
```

```
plt.colorbar(scatter1, label='数字标签')
plt.title('PCA降维结果 (2维)')
plt.xlabel('主成分1')
plt.ylabel('主成分2')

plt.subplot(1, 2, 2)
scatter2 = plt.scatter(X_tsne[:, 0], X_tsne[:, 1],
                       c=y_digits_subset, cmap='tab10', alpha=0.8, s=50)
plt.colorbar(scatter2, label='数字标签')
plt.title('t-SNE降维结果 (2维)')
plt.xlabel('t-SNE特征1')
plt.ylabel('t-SNE特征2')

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



观察比较:

- PCA是线性方法，保留全局结构，但类别间的分离不明显
- t-SNE是非线性方法，能更好地将不同类别分离开，形成清晰的簇
- t-SNE特别适合数据可视化和聚类分析

t-SNE的关键参数:

- perplexity: 控制近邻数量，通常在5到50之间
- early_exaggeration: 控制聚类分离程度
- learning_rate: 影响优化过程

2.3 第三部分：实际数据集上的应用

在实际应用中，维度约简常用于：

1. 数据预处理：在训练模型前降低特征数量
2. 可视化：帮助理解数据结构
3. 噪声消除：去除数据中的噪声维度

下面我们将演示如何将维度约简作为数据预处理步骤，并评估其对分类性能的影响

本节我们使用完整的手写数字数据集与KNN分类器进行试验

```
# 加载MNIST数据集
print("数据集：MNIST手写数字")
print(f"原始特征数量：{X_digits.shape[1]}")
```

```
数据集：MNIST手写数字
原始特征数量：64
```

```
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_digits, y_digits,
test_size=0.3, random_state=42)
```

```
# 不使用降维直接分类
scaler_full = StandardScaler()
X_train_scaled = scaler_full.fit_transform(X_train)
X_test_scaled = scaler_full.transform(X_test)
```

```
# 使用KNN分类器
start_time = time.time()
knn_full = KNeighborsClassifier(n_neighbors=5)
knn_full.fit(X_train_scaled, y_train)
y_pred_full = knn_full.predict(X_test_scaled)
end_time = time.time()

accuracy_full = accuracy_score(y_test, y_pred_full)
print("1. 不使用降维的分类性能：")
print(f"分类准确率：{accuracy_full:.4f}")
print(f"训练和预测时间：{end_time - start_time:.2f}秒")
```

```
1. 不使用降维的分类性能：
分类准确率：0.9759
训练和预测时间：0.02秒
```

使用PCA降维后分类

```
# 确定保留95%方差所需的主成分数量
scaler_pca = StandardScaler()
X_train_scaled_pca = scaler_pca.fit_transform(X_train)
X_test_scaled_pca = scaler_pca.transform(X_test)

pca_var = PCA()
pca_var.fit(X_train_scaled_pca)
cumulative_variance = np.cumsum(pca_var.explained_variance_ratio_)
n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1
print(f"保留95%方差需要的主成分数量: {n_components_95}")
```

```
保留95%方差需要的主成分数量: 40
```

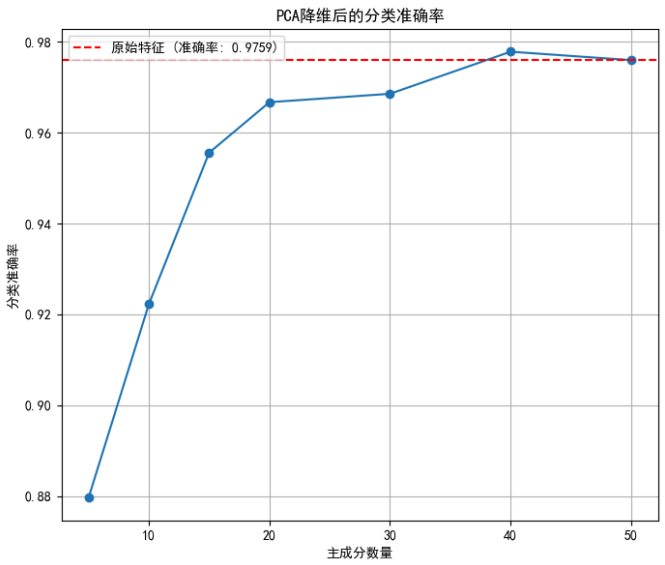
```
# 应用PCA降维
start_time = time.time()
pca_model = PCA(n_components=n_components_95)
X_train_pca = pca_model.fit_transform(X_train_scaled_pca)
X_test_pca = pca_model.transform(X_test_scaled_pca)

# 使用KNN分类器
knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_train_pca, y_train)
y_pred_pca = knn_pca.predict(X_test_pca)
end_time = time.time()

accuracy_pca = accuracy_score(y_test, y_pred_pca)
print("2. 使用PCA降维后的分类性能:")
print(f"分类准确率: {accuracy_pca:.4f}")
print(f"训练和预测时间: {end_time - start_time:.2f}秒")
print(f"降维后特征数量: {X_train_pca.shape[1]}")
```

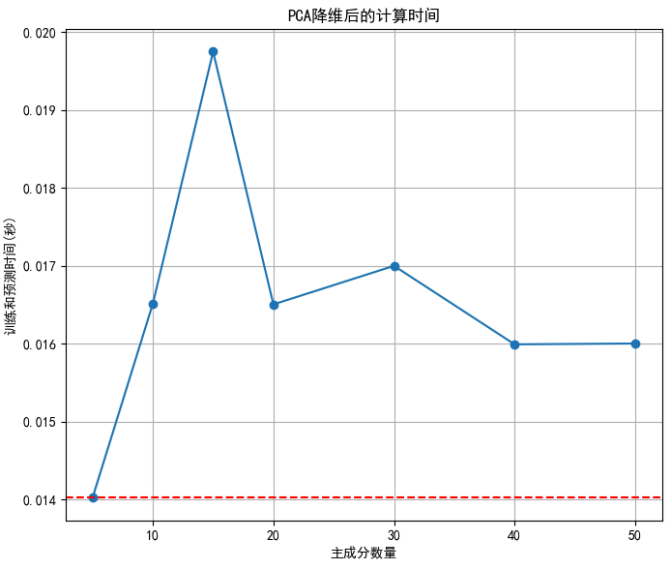
```
2. 使用PCA降维后的分类性能:
分类准确率: 0.9778
训练和预测时间: 0.03秒
降维后特征数量: 40
```

比较各种维度下的性能

```
n_components_range = [5, 10, 15, 20, 30, 40, 50] # 各种主成分数量
accuracies = []
training_times = []

for n_comp in n_components_range:
    start_time = time.time()
```

```
    # PCA降维
    pca_model = PCA(n_components=n_comp)
    X_train_pca = pca_model.fit_transform(X_train_scaled_pca)
    X_test_pca = pca_model.transform(X_test_scaled_pca)

    # 分类
    knn_model = KNeighborsClassifier(n_neighbors=5)
    knn_model.fit(X_train_pca, y_train)
    y_pred = knn_model.predict(X_test_pca)

    end_time = time.time()

    accuracies.append(accuracy_score(y_test, y_pred))
    training_times.append(end_time - start_time)

# 可视化结果
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(n_components_range, accuracies, marker='o', linestyle='-')
plt.axhline(y=accuracy_full, color='r', linestyle='--', label=f'原始特征 (准确率:
{accuracy_full:.4f})')
plt.xlabel('主成分数量')
plt.ylabel('分类准确率')
plt.title('PCA降维后的分类准确率')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(n_components_range, training_times, marker='o', linestyle='-')
plt.axhline(y=training_times[0], color='r', linestyle='--')
plt.xlabel('主成分数量')
plt.ylabel('训练和预测时间(秒)')
plt.title('PCA降维后的计算时间')
plt.grid(True)

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





从上面的分析可以看出：

1. 使用PCA降维后，我们可以大幅减少特征数量（从64维降至约40维）
2. 降维后的分类性能接近原始特征的性能
3. 随着维度减少，计算时间显著降低
4. 当主成分数量太少时，性能会明显下降

这说明维度约简在保留大部分有用信息的同时，能显著提高计算效率。

3. 思考

探索非线性降维技术的优势和应用场景

- 实现核PCA (Kernel PCA)并与标准PCA、t-SNE进行比较
- 在人造数据集（如瑞士卷或S曲线）上对比不同算法的效果
- 分析各算法在保留数据结构方面的优劣