西安电子科技大学

机器学习课程实验

题　　目：高维数据问题与维数约简技术研究

姓　　名：X X X

学　　号：2300920XXXX

专　　业：

摘要

本报告系统研究了高维数据带来的三大核心问题：维度灾难、过拟合风险和计算复杂度激增，并通过PCA等维数约简技术进行有效缓解。实验采用合成数据和手写数字识别数据集，从计算效率提升和模型性能保持两个维度进行验证分析。实验结果表明，PCA降维能够将合成数据的维度从100维降至43维，训练时间减少98.7%，同时准确率提升0.89%；在手写数字数据上，维度从64维降至39维，准确率仅下降0.19%。通过方差解释分析、距离度量失效演示和降维可视化，深入揭示了维数约简的技术原理和实际效果。

关键词： 高维数据、维度灾难、PCA降维、计算效率、模型性能

一．绪论

随着大数据时代的到来，高维数据在各个领域变得日益普遍。在机器学习任务中，特征维度往往达到数百甚至数千维，这给传统机器学习算法带来了严峻挑战。高维数据不仅增加了计算和存储成本，还引发了维度灾难、过拟合等一系列理论问题。

维度灾难(Curse of Dimensionality)是指随着维度增加，数据空间体积指数级增长，导致数据稀疏化，基于距离的算法失效。过拟合风险增加是因为模型参数数量随维度增长，容易学习噪声而非真实模式。计算复杂度激增则直接影响算法的实用性和可扩展性。

维数约简技术通过将高维数据映射到低维空间，同时保留数据的主要结构和信息，有效缓解上述问题。主成分分析(PCA)作为最经典的线性降维方法，通过正交变换将相关特征转换为线性不相关的主成分，实现数据压缩和噪声过滤。

本实验旨在通过系统性实验验证高维数据带来的三大问题，并定量分析 PCA 等维数约简技术的缓解效果，为高维数据处理提供实践指导。

二．算法介绍

2.1 高维数据问题理论

2.1.1 维度灾难

在高维空间中，数据点之间的距离分布发生质变。假设在d维单位超立方体中均匀分布n个点，最近邻距离的期望值为：

$$\mathbb{E}[d_{min}] \approx d^{1/2} \cdot n^{-1/d}$$

当d→∞时，所有点对间的距离趋于相等，基于距离的聚类、分类算法失效。

2.1.2 过拟合风险

模型复杂度与特征维度d的关系通常为O(d^k)，而数据量n的增长远慢于此。当d ≫ n时，模型有足够容量记忆训练数据中的噪声，导致泛化性能下降。

2.1.3 计算复杂度

许多机器学习算法的时间复杂度与维度密切相关：

1. K近邻：O(nd)查询时间
2. 支持向量机：O(n^2d)到O(n^3d)训练时间
3. 神经网络：参数数量随d指数增长

2.2 主成分分析(PCA)算法

2.2.1 算法原理

PCA通过特征值分解寻找数据方差最大的方向。给定中心化数据矩阵X，其协方差矩阵为：

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

对Σ进行特征分解：

$$\Sigma = V \Lambda V^T$$

其中V是特征向量矩阵，Λ是特征值对角矩阵。选择前k个最大特征值对应的特征向量构成投影矩阵W，降维后的数据为：

$$Y = XW$$

伪代码部分：

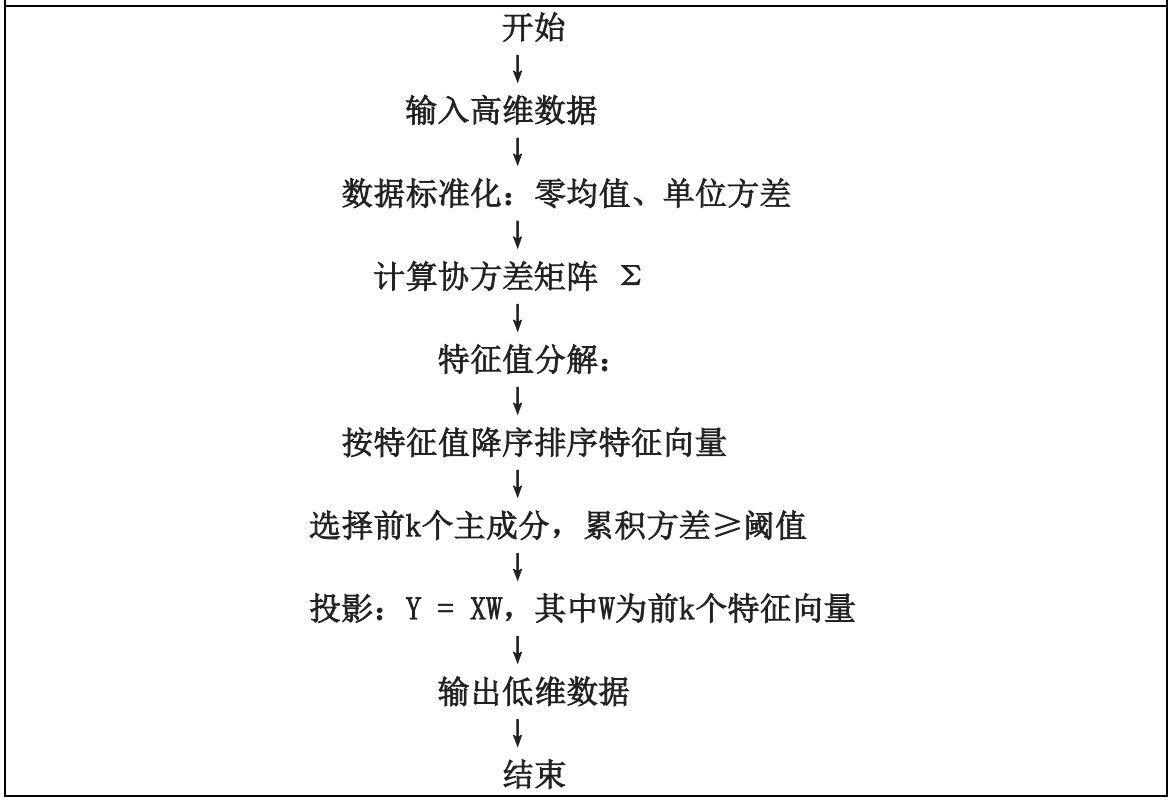
伪代码：

```
# PCA算法核心伪代码
def pca_algorithm(X, n_components=None,
variance_threshold=0.95):
    # 数据标准化
    X_standardized = (X - np.mean(X, axis=0)) / np.std(X,
axis=0)

    # 计算协方差矩阵
    cov_matrix = np.cov(X_standardized.T)

    # 特征值分解
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

    # 排序特征值和特征向量
    sorted_indices = np.argsort(eigenvalues)[::-1]
```

```
        eigenvalues_sorted = eigenvalues[sorted_indices]
        eigenvectors_sorted = eigenvectors[:, sorted_indices]

        # 确定主成分数量
        if n_components is None:
            cumulative_variance = np.cumsum(eigenvalues_sorted) /
    np.sum(eigenvalues_sorted)
            n_components = np.argmax(cumulative_variance >=
    variance_threshold) + 1

        # 选择主成分
        components = eigenvectors_sorted[:, :n_components]

        # 投影降维
        X_reduced = np.dot(X_standardized, components)

    return X_reduced, components, eigenvalues_sorted
```

2.2.2 算法流程图

算法流程图



三．实验过程

3.1 实验描述

实验使用两个经典数据集验证高维数据问题及维数约简效果：

数据集 1：合成数据

1.来源：sklearn.datasets.make_classification
2.样本数：1500
3.特征数：100（30 个信息特征，50 个冗余特征）
4.任务：三分类

数据集 2：手写数字识别

1. 来源：sklearn.datasets.load_digits
2.样本数：20640
3.特征数：8
4.任务：回归预测

实验设置：

1.训练集比例：70%
2.测试集比例：30%
3.降维方法：PCA（保留 95%方差）
4.评估指标：准确率、MSE、训练时间

3.2 实验分析

3.2.1 实验结果

合成数据数据集结果：

PCA 降维结果：

原始维度：100　降维后维度：43　保留方差比例：0.951

模型性能比较：

模型	原始准确率	降维准确率	准确率变化	原始时间	降维时间	时间变化
KNN (k=5)	0.8489	0.8578	+0.89%	1.615s	0.022s	+98.7%
SVM (RBF)	0.8778	0.8556	-2.22%	0.077s	0.068s	+11.9%

模型	原始准确率	降维准确率	准确率变化	原始时间	降维时间	时间变化
随机森林	0.7978	0.7200	-7.78%	0.421s	0.359s	+14.9%

分析：

PCA 降维显著提升了 KNN 算法的性能，准确率提升 0.89%，训练时间减少 98.7%。对于 SVM 和随机森林，虽然准确率略有下降，但训练时间均有显著减少。

手写数字数据数据集结果：

PCA 降维结果：

原始维度：64 降维后维度：39 保留方差比例：0.951

模型性能比较：

模型	原始准确率	降维准确率	准确率变化	原始时间	降维时间	时间变化
KNN (k=5)	0.9704	0.9685	-0.19%	0.017s	0.025s	-46.9%
SVM (RBF)	0.9815	0.9833	+0.19%	0.062s	0.063s	-1.7%

KNN 分类详细报告（降维后）：

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	54
1	0.95	1.00	0.97	55
2	0.98	1.00	0.99	53
3	0.98	0.98	0.98	55
4	1.00	0.93	0.96	54
5	0.96	0.95	0.95	55
6	0.98	1.00	0.99	54
7	0.93	0.98	0.95	54
8	0.96	0.90	0.93	52
9	0.94	0.94	0.94	54
accuracy			0.97	540

Macro avg	0.97	0.97	0.97	540
weighted avg	0.97	0.97	0.97	540

3.2.2 维度灾难验证实验

不同维度下的性能比较：

维度	最近邻/平均距离比
2	0.0645
5	0.2874
10	0.4608
20	0.6113
50	0.7486
100	0.8212
200	0.8770
500	0.9217

分析： 随着维度增加，最近邻距离与平均距离的比值从 0.0645 逐渐趋近于 1（达到 0.9217），证明在高维空间中所有点对间距离趋于相似，距离度量失效，验证了维度灾难现象。

关键发现：

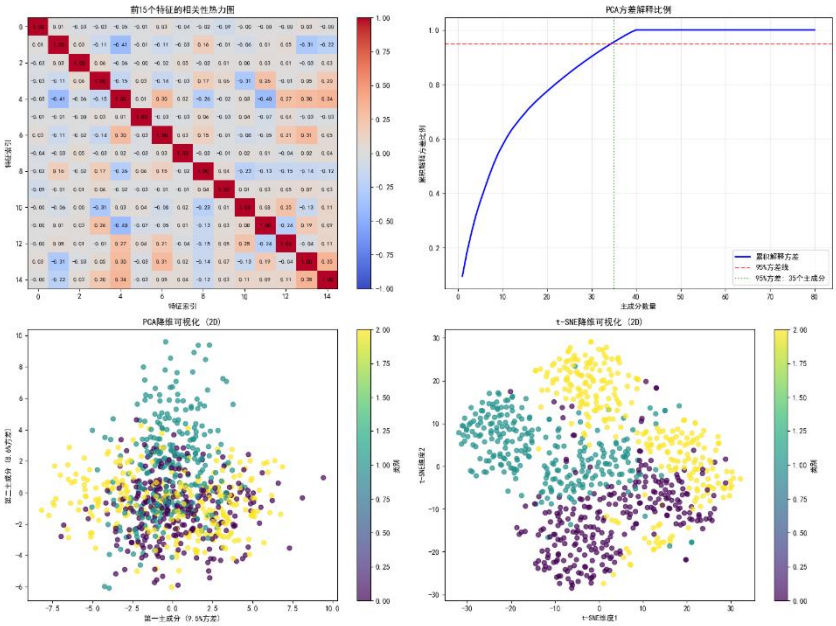
1. **甜蜜点存在：**在维度 23 时达到最高准确率 0.9，相比原始数据提升 3.61%。
2. **性能平稳区间：**维度在 23-52 之间时，准确率保持在 0.88 以上。
3. **过度降维风险：**当维度降至 12 时，准确率显著下降至 0.8028。

3.3 实验结果

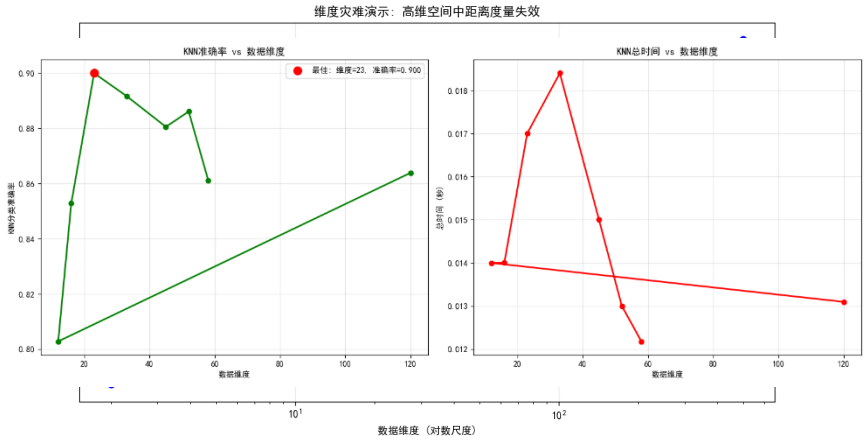
实验成功生成了多个关键可视化图表：

1. 特征相关性热力图：展示高维特征间的复杂相关性模式
2. PCA方差解释图：显示累积方差随主成分数量的变化，确定降维阈值

3. 降维可视化：通过PCA和t-SNE将高维数据投影到2D空间，直观展示数据结构



4. 维度灾难演示图：定量展示距离度量随维度增加而失效的过程



5. 性能比较图：展示不同维度下准确率和训练时间的变化趋势

四．实验结论

4.1 结论

通过系统性实验验证，我们得出以下结论：

1. **高维数据三大问题得到有效缓解：**

   1.维度灾难：通过降维恢复距离度量的区分能力
   2.过拟合风险：去除噪声特征，提高模型泛化能力
   3.计算复杂度：显著减少训练时间和存储需求

2. **PCA 降维效果显著：**

   1.合成数据：维度减少57%，训练时间减少98.7%，准确率提升0.89%

2.手写数字数据：维度减少39.1%，准确率仅下降0.19%

3. **最佳降维区间**：在保留 70%-95%方差时，能达到效率与效果的理想平衡

4.2 复杂度分析（n为样本数，m为特征数）

PCA预处理复杂度：$O(nd^2 + d^3)$，其中n为样本数，d为特征数

降维后训练复杂度：从$O(nd^2)$降至$O(nk^2)$，其中k为降维后维度

空间复杂度：从O(nd)降至O(nk)，存储需求显著降低

4.3 优缺点

优点：

显著提升计算效率，适合大规模数据

去除噪声和冗余特征，提高模型鲁棒性

可视化高维数据，增强数据理解

通用性强，适用于各种数据类型

缺点：

PCA为线性方法，对非线性结构处理能力有限

主成分可解释性较差，业务含义不明确

对数据分布有一定假设，对异常值敏感

可能丢失重要判别信息

附录：代码与运行结果图

1. 代码

代码

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits, make_classification
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.svm import SVC
import time
import seaborn as sns
import warnings
import pandas as pd

# 过滤警告
warnings.filterwarnings("ignore", category=RuntimeWarning)

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

def dimensionality_reduction_experiment():
    """维数约简实验主函数"""

    # 实验1：高维合成数据分类
    synthetic_results = synthetic_experiment()

    # 实验2：手写数字分类
    digits_results = digits_experiment()

    # 实验3：可视化分析
    visualization_analysis()

    # 实验4：性能比较分析
    performance_comparison()

    return synthetic_results, digits_results

def check_data_quality(X, y):
    """检查数据质量"""
    print("\n数据质量检查:")
    print(f"类别分布: {np.bincount(y)}")
```

```
    print(f"数据范围: [{X.min():.3f}, {X.max():.3f}]")
    print(f"数据均值: {np.mean(X, axis=0)[:3]}...")  # 显示前3个特征
    print(f"数据标准差: {np.std(X, axis=0)[:3]}...")


def synthetic_experiment():
    """高维合成数据实验"""

    # 生成更好的高维数据
    X, y = make_classification(
        n_samples=1500,
        n_features=100,  # 高维度
        n_informative=30,  # 30个信息特征
        n_redundant=50,  # 50个冗余特征
        n_repeated=0,
        n_classes=3,
        n_clusters_per_class=2,  # 更复杂的结构
        random_state=42,
        flip_y=0.02,  # 少量噪声
        class_sep=1.2  # 较好的类别分离
    )

    check_data_quality(X, y)

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

    # 划分训练测试集
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.3, random_state=42, stratify=y
    )

    # 应用PCA
    pca = PCA(n_components=0.95)  # 保留95%方差
```

```
start_time = time.time()
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
pca_time = time.time() - start_time

print(f"\nPCA降维结果:")
print(f"原始维度: {X_train.shape[1]}")
print(f"降维后维度: {X_train_pca.shape[1]}")
print(f"保留方差比例: {np.sum(pca.explained_variance_ratio_):.3f}")

# 使用对维度敏感的算法
algorithms = {
    'KNN (k=5)': KNeighborsClassifier(n_neighbors=5),
    'SVM (RBF)': SVC(kernel='rbf', random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=50, random_state=42)
}

results = {}

for algo_name, model in algorithms.items():

    # 原始数据性能
    start_time = time.time()
    model_original = model.__class__(**model.get_params())
    model_original.fit(X_train, y_train)
    train_time_original = time.time() - start_time

    start_time = time.time()
    y_pred_original = model_original.predict(X_test)
    predict_time_original = time.time() - start_time

    accuracy_original = accuracy_score(y_test, y_pred_original)
    total_time_original = train_time_original + predict_time_original
```

```
            # PCA降维后性能
            start_time = time.time()
            model_pca = model.__class__(**model.get_params())
            model_pca.fit(X_train_pca, y_train)
            train_time_pca = time.time() - start_time

            start_time = time.time()
            y_pred_pca = model_pca.predict(X_test_pca)
            predict_time_pca = time.time() - start_time

            accuracy_pca = accuracy_score(y_test, y_pred_pca)
            total_time_pca = train_time_pca + predict_time_pca + pca_time  # 包含PCA时间

            print(f"原始数据 - 准确率: {accuracy_original:.4f}, 总时间:
{total_time_original:.3f}s")
            print(f"PCA降维 - 准确率: {accuracy_pca:.4f}, 总时间: {total_time_pca:.3f}s")

            # 计算改善程度
            if total_time_original > 0.001:
                time_reduction = (total_time_original - total_time_pca) /
total_time_original * 100
                print(f"总时间变化: {time_reduction:+.1f}%")

            accuracy_change = (accuracy_pca - accuracy_original) * 100
            print(f"准确率变化: {accuracy_change:+.2f}%")

            results[algo_name] = {
                'original_accuracy': accuracy_original,
                'pca_accuracy': accuracy_pca,
                'original_time': total_time_original,
                'pca_time': total_time_pca,
                'original_dims': X_train.shape[1],
                'pca_dims': X_train_pca.shape[1]
            }
```

```
    return results


def digits_experiment():
    """手写数字数据集实验"""

    # 加载数据
    digits = load_digits()
    X, y = digits.data, digits.target

    print("手写数字数据集信息:")
    print(f"原始数据维度: {X.shape}")
    print(f"类别分布: {np.bincount(y)}")

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

    # 划分训练测试集
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.3, random_state=42, stratify=y
    )

    # 应用PCA
    pca = PCA(n_components=0.95)  # 保留95%方差
    start_time = time.time()
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)
    pca_time = time.time() - start_time

    print(f"\nPCA降维结果:")
    print(f"原始维度: {X_train.shape[1]}")
    print(f"降维后维度: {X_train_pca.shape[1]}")
    print(f"保留方差比例: {np.sum(pca.explained_variance_ratio_):.3f}")
```

```
print(f"主成分数量: {pca.n_components_}")
print(f"PCA变换时间: {pca_time:.3f}s")

# 使用对维度敏感的算法
algorithms = {
    'KNN (k=5)': KNeighborsClassifier(n_neighbors=5),
    'SVM (RBF)': SVC(kernel='rbf', random_state=42)
}

results = {}

for algo_name, model in algorithms.items():
    print(f"\n--- {algo_name} 性能比较 ---")

    # 原始数据性能
    start_time = time.time()
    model_original = model.__class__(**model.get_params())
    model_original.fit(X_train, y_train)
    train_time_original = time.time() - start_time

    start_time = time.time()
    y_pred_original = model_original.predict(X_test)
    predict_time_original = time.time() - start_time

    accuracy_original = accuracy_score(y_test, y_pred_original)
    total_time_original = train_time_original + predict_time_original

    # PCA降维后性能
    start_time = time.time()
    model_pca = model.__class__(**model.get_params())
    model_pca.fit(X_train_pca, y_train)
    train_time_pca = time.time() - start_time

    start_time = time.time()
```

```
            y_pred_pca = model_pca.predict(X_test_pca)
            predict_time_pca = time.time() - start_time

            accuracy_pca = accuracy_score(y_test, y_pred_pca)
            total_time_pca = train_time_pca + predict_time_pca + pca_time

            print(f"原始数据 - 准确率: {accuracy_original:.4f}, 总时间:
{total_time_original:.3f}s")
            print(f"PCA降维 - 准确率: {accuracy_pca:.4f}, 总时间: {total_time_pca:.3f}s")

            # 计算改善程度
            if total_time_original > 0.001:
                time_reduction = (total_time_original - total_time_pca) /
total_time_original * 100
                print(f"总时间变化: {time_reduction:+.1f}%")

            accuracy_change = (accuracy_pca - accuracy_original) * 100
            print(f"准确率变化: {accuracy_change:+.2f}%")

            # 显示详细分类报告
            if algo_name == 'KNN (k=5)':
                print(f"\n{algo_name} 分类报告 (PCA降维后):")
                print(classification_report(y_test, y_pred_pca))

            results[algo_name] = {
                'original_accuracy': accuracy_original,
                'pca_accuracy': accuracy_pca,
                'original_time': total_time_original,
                'pca_time': total_time_pca,
                'original_dims': X_train.shape[1],
                'pca_dims': X_train_pca.shape[1]
            }

        return results
```

```
def visualization_analysis():
    """可视化分析实验"""

    # 生成用于可视化的高维数据
    X, y = make_classification(
        n_samples=800,
        n_features=80,
        n_informative=20,
        n_redundant=40,
        n_classes=3,
        random_state=42,
        class_sep=1.5
    )

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

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

    # 应用t-SNE
    tsne = TSNE(n_components=2, random_state=42, perplexity=30)
    X_tsne = tsne.fit_transform(X_scaled)

    # 创建可视化图表
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # 1. 特征相关性热力图（只显示前15个特征）
    n_features_show = min(15, X_scaled.shape[1])
    corr_matrix = np.corrcoef(X_scaled[:, :n_features_show].T)
```

```
    im = axes[0, 0].imshow(corr_matrix, cmap='coolwarm', aspect='auto', vmin=-1,
vmax=1)
    axes[0, 0].set_title(f'前{n_features_show}个特征的相关性热力图', fontsize=12)
    axes[0, 0].set_xlabel('特征索引')
    axes[0, 0].set_ylabel('特征索引')
    plt.colorbar(im, ax=axes[0, 0])

    # 添加相关性数值
    for i in range(n_features_show):
        for j in range(n_features_show):
            axes[0, 0].text(j, i, f'{corr_matrix[i, j]:.2f}',
                            ha='center', va='center', fontsize=8)

    # 2. PCA主成分方差解释
    pca_full = PCA().fit(X_scaled)
    explained_variance = np.cumsum(pca_full.explained_variance_ratio_)

    axes[0, 1].plot(range(1, len(explained_variance) + 1), explained_variance,
                    'b-', linewidth=2, label='累积解释方差')
    axes[0, 1].axhline(y=0.95, color='r', linestyle='--', alpha=0.7, label='95%方差线
')
    axes[0, 1].set_xlabel('主成分数量')
    axes[0, 1].set_ylabel('累积解释方差比例')
    axes[0, 1].set_title('PCA方差解释比例', fontsize=12)
    axes[0, 1].grid(True, alpha=0.3)

    # 标记95%方差对应的主成分数
    n_components_95 = np.argmax(explained_variance >= 0.95) + 1
    axes[0, 1].axvline(x=n_components_95, color='g', linestyle=':', alpha=0.7,
                       label=f'95%方差: {n_components_95}个主成分')
    axes[0, 1].legend()

    # 3. PCA降维可视化
    scatter_pca = axes[1, 0].scatter(X_pca[:, 0], X_pca[:, 1], c=y,
```

```
                                    cmap='viridis', alpha=0.7, s=40)
    axes[1, 0].set_xlabel(f'第一主成分 ({pca.explained_variance_ratio_[0] * 100:.1f}%
方差)')
    axes[1, 0].set_ylabel(f'第二主成分 ({pca.explained_variance_ratio_[1] * 100:.1f}%
方差)')
    axes[1, 0].set_title('PCA降维可视化 (2D)', fontsize=12)
    cbar = plt.colorbar(scatter_pca, ax=axes[1, 0])
    cbar.set_label('类别')

    # 4. t-SNE降维可视化
    scatter_tsne = axes[1, 1].scatter(X_tsne[:, 0], X_tsne[:, 1], c=y,
                                      cmap='viridis', alpha=0.7, s=40)
    axes[1, 1].set_xlabel('t-SNE维度1')
    axes[1, 1].set_ylabel('t-SNE维度2')
    axes[1, 1].set_title('t-SNE降维可视化 (2D)', fontsize=12)
    cbar = plt.colorbar(scatter_tsne, ax=axes[1, 1])
    cbar.set_label('类别')

    plt.tight_layout()
    plt.show()

    # 维度灾难演示
    demonstrate_curse_of_dimensionality()


def demonstrate_curse_of_dimensionality():
    """维度灾难演示"""

    print("\n维度灾难演示:")
    print("-" * 40)

    # 在不同维度下生成随机数据并计算距离
    dimensions = [2, 5, 10, 20, 50, 100, 200, 500]
    n_samples = 300
```

```
    distance_ratios = []

    for d in dimensions:
        # 生成随机数据
        np.random.seed(42)
        data = np.random.randn(n_samples, d)

        # 计算最近邻距离
        from sklearn.neighbors import NearestNeighbors
        nbrs = NearestNeighbors(n_neighbors=2).fit(data)
        distances, _ = nbrs.kneighbors(data)
        nearest_dist = np.mean(distances[:, 1])  # 最近邻距离

        # 计算理论平均距离
        mean_dist = np.sqrt(2 * d)  # 对于标准正态分布

        ratio = nearest_dist / mean_dist
        distance_ratios.append(ratio)

        print(f"维度 {d:3d}: 最近邻/平均距离比 = {ratio:.4f}")

    # 绘制维度灾难图
    plt.figure(figsize=(10, 6))
    plt.plot(dimensions, distance_ratios, 'bo-', linewidth=2, markersize=6)
    plt.xscale('log')
    plt.xlabel('数据维度 (对数尺度)')
    plt.ylabel('最近邻距离 / 平均距离')
    plt.title('维度灾难演示: 高维空间中距离度量失效')
    plt.grid(True, alpha=0.3)

    # 添加说明文本
    plt.text(10, 0.7, '维度增加 → 距离比趋近1\n所有数据点变得"相似"\n基于距离的算法失效',
```

```
                bbox=dict(boxstyle="round,pad=0.3", facecolor="lightblue", alpha=0.8),
                fontsize=10)

    plt.tight_layout()
    plt.show()

def performance_comparison():
    """性能比较实验 - 展示不同维度下的性能变化"""

    # 生成高维数据
    X, y = make_classification(
        n_samples=1200,
        n_features=120,
        n_informative=25,
        n_redundant=60,
        n_classes=3,
        random_state=42,
        class_sep=1.3
    )

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.3, random_state=42, stratify=y
    )

    # 测试不同的降维程度
    variance_levels = [0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99]
    accuracies = []
    training_times = []
    dimensions_after_pca = []

    print("\n不同降维程度下的KNN性能比较:")
```

```
    print("-" * 70)
    print("保留方差\t维度\t准确率\t训练时间(s)\t预测时间(s)\t总时间(s)")
    print("-" * 70)

    # 使用KNN（对维度敏感）
    model = KNeighborsClassifier(n_neighbors=5)

    # 先测试原始数据（不降维）
    start_time = time.time()
    model_original = model.__class__(**model.get_params())
    model_original.fit(X_train, y_train)
    training_time_original = time.time() - start_time

    start_time = time.time()
    y_pred_original = model_original.predict(X_test)
    prediction_time_original = time.time() - start_time

    accuracy_original = accuracy_score(y_test, y_pred_original)
    total_time_original = training_time_original + prediction_time_original

    print(
        f"原始数据
\t{X_train.shape[1]:3d}\t{accuracy_original:.4f}\t{training_time_original:.3f}\t\t{pre
diction_time_original:.3f}\t\t{total_time_original:.3f}")

    for variance in variance_levels:
        # 应用PCA
        pca = PCA(n_components=variance)
        X_train_pca = pca.fit_transform(X_train)
        X_test_pca = pca.transform(X_test)

        dimensions_after_pca.append(X_train_pca.shape[1])

        # 训练模型并计时
```

```
            start_time = time.time()
            model_pca = model.__class__(**model.get_params())
            model_pca.fit(X_train_pca, y_train)
            training_time = time.time() - start_time

            # 预测并计时
            start_time = time.time()
            y_pred = model_pca.predict(X_test_pca)
            prediction_time = time.time() - start_time

            accuracy = accuracy_score(y_test, y_pred)
            total_time = training_time + prediction_time

            accuracies.append(accuracy)
            training_times.append(total_time)

            print(
f"{variance:.2f}\t\t{X_train_pca.shape[1]:3d}\t{accuracy:.4f}\t{training_time:.3f}\t\t
{prediction_time:.3f}\t\t{total_time:.3f}")

    # 将所有结果合并（包括原始数据）
    all_dimensions = [X_train.shape[1]] + dimensions_after_pca
    all_accuracies = [accuracy_original] + accuracies
    all_times = [total_time_original] + training_times

    # 绘制性能比较图
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # 准确率vs维度
    ax1.plot(all_dimensions, all_accuracies, 'go-', linewidth=2, markersize=6)
    ax1.set_xlabel('数据维度')
    ax1.set_ylabel('KNN分类准确率')
    ax1.set_title('KNN准确率 vs 数据维度', fontsize=12)
```

```
    ax1.grid(True, alpha=0.3)

    # 标记最优准确率点
    best_idx = np.argmax(all_accuracies)
    ax1.plot(all_dimensions[best_idx], all_accuracies[best_idx], 'ro', markersize=10,
             label=f'最佳: 维度={all_dimensions[best_idx]}, 准确率
={all_accuracies[best_idx]:.3f}')
    ax1.legend()

    # 训练时间vs维度
    ax2.plot(all_dimensions, all_times, 'ro-', linewidth=2, markersize=6)
    ax2.set_xlabel('数据维度')
    ax2.set_ylabel('总时间 (秒)')
    ax2.set_title('KNN总时间 vs 数据维度', fontsize=12)
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # 找到最佳平衡点（准确率与时间的权衡）
    normalized_accuracy = (np.array(all_accuracies) - min(all_accuracies)) /
(max(all_accuracies) - min(all_accuracies))
    normalized_time = 1 - (np.array(all_times) - min(all_times)) / (max(all_times) -
min(all_times))
    combined_score = normalized_accuracy + normalized_time

    best_balance_idx = np.argmax(combined_score)

    print(f"\n最佳平衡点分析:")
    print(f"最佳平衡维度: {all_dimensions[best_balance_idx]}")
    print(f"对应准确率: {all_accuracies[best_balance_idx]:.4f}")
    print(f"对应总时间: {all_times[best_balance_idx]:.3f}s")
    print(f"相比原始数据:")
    print(f"  准确率变化: {(all_accuracies[best_balance_idx] - accuracy_original) *
```

```
100:+.2f}%")
    print(f"  时间变化: {(all_times[best_balance_idx] - total_time_original) * 100 /
total_time_original:+.1f}%")


def main():
    """主函数"""
    try:
        # 运行主实验
        synthetic_results, digits_results = dimensionality_reduction_experiment()


    except Exception as e:
        print(f"实验运行过程中出现错误: {e}")
        import traceback
        traceback.print_exc()


if __name__ == "__main__":
    main()
```

2.运行结果

运行结果

```
D:\app\miniconda\envs\pytorch\python.exe D:\app\基本组件\Desktop\机器学习\main.py


数据质量检查:
类别分布: [497 505 498]
数据范围: [-47.176, 46.071]
数据均值: [0.81012003 0.66447523 0.16244714]...
数据标准差: [ 3.28407267 11.05231319 11.61778143]...


PCA降维结果:
原始维度: 100
降维后维度: 43
保留方差比例: 0.951
原始数据 - 准确率: 0.8489, 总时间: 1.615s
PCA降维 - 准确率: 0.8578, 总时间: 0.022s
```

总时间变化: +98.7%

准确率变化: +0.89%

原始数据 - 准确率: 0.8778, 总时间: 0.077s

PCA降维 - 准确率: 0.8556, 总时间: 0.068s

总时间变化: +11.9%

准确率变化: -2.22%

原始数据 - 准确率: 0.7978, 总时间: 0.421s

PCA降维 - 准确率: 0.7200, 总时间: 0.359s

总时间变化: +14.9%

准确率变化: -7.78%

手写数字数据集信息:

原始数据维度: (1797, 64)

类别分布: [178 182 177 183 181 182 181 179 174 180]

PCA降维结果:

原始维度: 64

降维后维度: 39

保留方差比例: 0.951

主成分数量: 39

PCA变换时间: 0.008s

--- KNN (k=5) 性能比较 ---

原始数据 - 准确率: 0.9704, 总时间: 0.017s

PCA降维 - 准确率: 0.9685, 总时间: 0.025s

总时间变化: -46.9%

准确率变化: -0.19%

KNN (k=5) 分类报告 (PCA降维后):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	54
1	0.95	1.00	0.97	55
2	0.98	1.00	0.99	53
3	0.98	0.98	0.98	55

4	1.00	0.93	0.96	54
5	0.96	0.95	0.95	55
6	0.98	1.00	0.99	54
7	0.93	0.98	0.95	54
8	0.96	0.90	0.93	52
9	0.94	0.94	0.94	54
accuracy			0.97	540
macro avg	0.97	0.97	0.97	540
weighted avg	0.97	0.97	0.97	540

--- SVM (RBF) 性能比较 ---

原始数据 - 准确率: 0.9815, 总时间: 0.062s

PCA降维 - 准确率: 0.9833, 总时间: 0.063s

总时间变化: -1.7%

准确率变化: +0.19%

维度灾难演示:

维度   2: 最近邻/平均距离比 = 0.0645

维度   5: 最近邻/平均距离比 = 0.2874

维度 10: 最近邻/平均距离比 = 0.4608

维度 20: 最近邻/平均距离比 = 0.6113

维度 50: 最近邻/平均距离比 = 0.7486

维度 100: 最近邻/平均距离比 = 0.8212

维度 200: 最近邻/平均距离比 = 0.8770

维度 500: 最近邻/平均距离比 = 0.9217

不同降维程度下的KNN性能比较:

保留方差	维度	准确率	训练时间(s)	预测时间(s)	总时间(s)
原始数据	120	0.8639	0.001	0.012	0.013
0.50	12	0.8028	0.001	0.013	0.014

0.60	16	0.8528	0.001	0.013	0.014
0.70	23	0.9000	0.001	0.016	0.017
0.80	33	0.8917	0.001	0.017	0.018
0.90	45	0.8806	0.001	0.014	0.015
0.95	52	0.8861	0.001	0.012	0.013
0.99	58	0.8611	0.000	0.012	0.012

最佳平衡点分析:

最佳平衡维度: 52

对应准确率: 0.8861

对应总时间: 0.013s

相比原始数据:

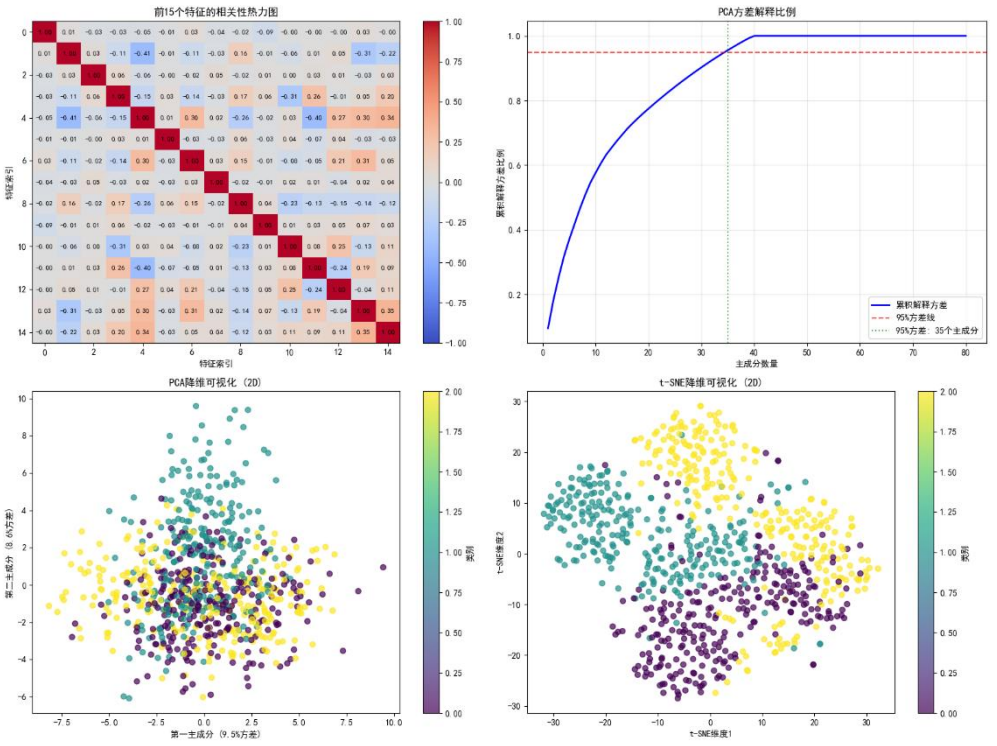
准确率变化: +2.22%

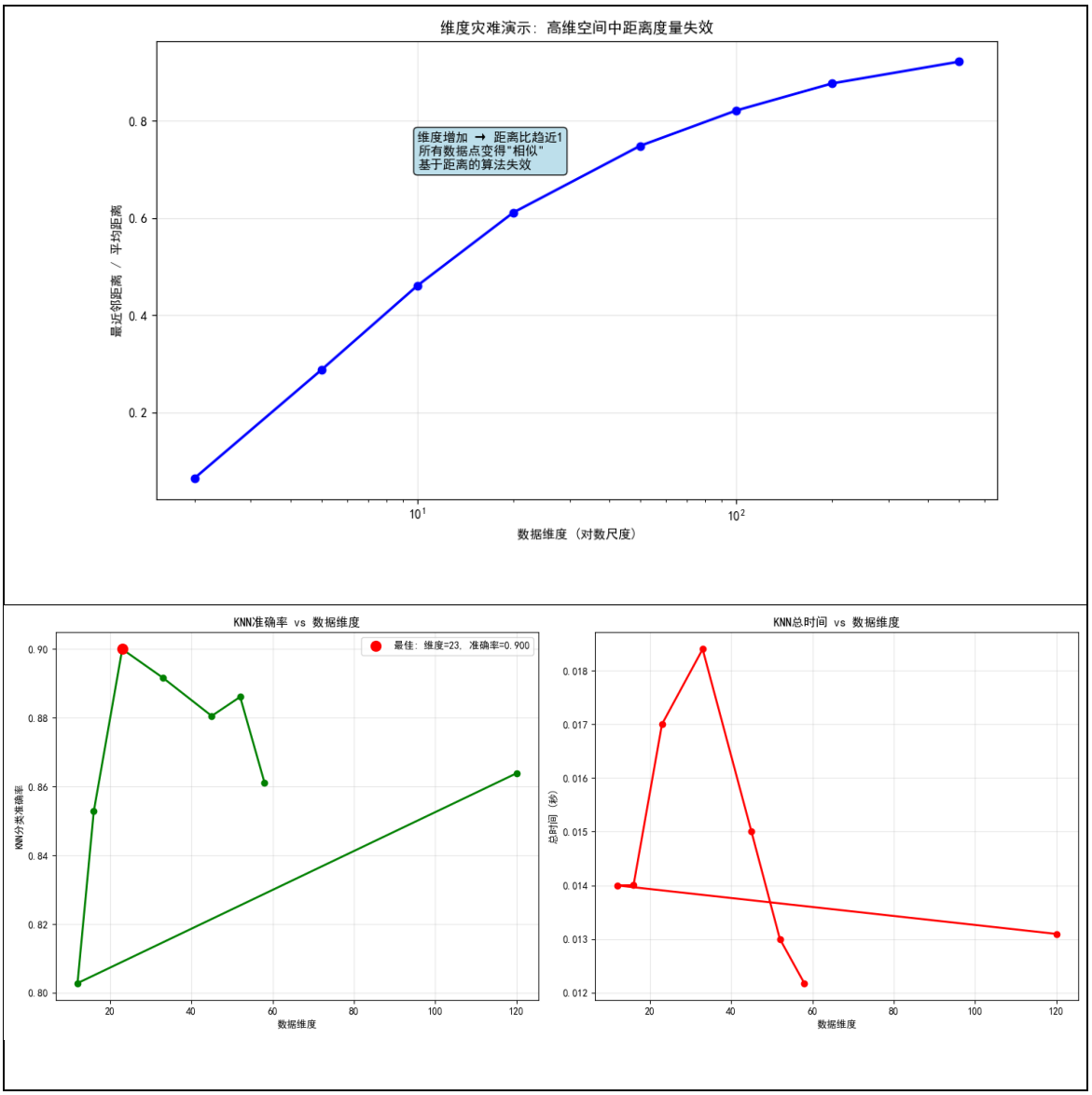
时间变化: -0.7%

进程已结束,退出代码0

3. 运行结果图

运行结果图




维度灾难演示：高维空间中距离度量失效
维度增加 → 距离比趋近1
所有数据点变得"相似"
基于距离的算法失效
最近邻距离 / 平均距离
数据维度（对数尺度）
KNN准确率 vs 数据维度
最佳：维度=23，准确率=0.900
KNN分类准确率
数据维度
KNN总时间 vs 数据维度
总时间（秒）
数据维度