**一、**简述**PCA及t-SNE**的原理、关键操作点（包括n\_components，主成分与原始特征的关系等；PCA与t-SNE的区别，不要超过**300字**，逻辑要通，**要有小标题**，要排版清晰，请不要网络大幅摘抄）

**PCA原理：**

PCA（Principal Component Analysis）是一种常用的降维技术，通过线性变换将高维数据映射到低维空间，保留数据集中的主要信息。其原理是通过找到数据集中的主成分（即方差最大的方向），并将数据投影到这些主成分上，实现降维。

**关键操作点：**

1. `n\_components`：指定要保留的主成分数量，决定了降维后的维度。

2. 主成分与原始特征的关系：主成分是原始特征的线性组合，保留了最大方差的方向，从而保留了数据的主要信息。

**t-SNE原理：**

t-SNE（t-Distributed Stochastic Neighbor Embedding）是一种非线性降维技术，能够将高维数据映射到二维或三维空间，同时保持数据间的局部关系。其原理是基于相似性概率建立高维空间中的概率分布，并尝试在低维空间中保持这些概率分布，从而保持数据点间的局部关系。

**关键操作点：**

1. `n\_components`：指定要降维到的维度。

2. 局部关系的保持：t-SNE重点在于保持数据点之间的局部相似性，而非全局结构。

**PCA与t-SNE的区别：**

1. 线性 vs 非线性：PCA是线性降维技术，通过线性变换将高维数据映射到低维空间；而t-SNE是非线性降维技术，能够保持数据间的非线性关系。

2. 全局 vs 局部：PCA倾向于保持全局结构，即更注重数据的整体分布；而t-SNE更关注保持局部结构，即更强调数据点之间的相对位置关系。

3. 计算复杂度：t-SNE的计算复杂度通常比PCA高，尤其在处理大规模数据时。

**二、利用算法解决问题。**

教材P187操作题3题：

1.使用sklearn内置的任意用于分离的数据集，用PCA进行降维，并将处置后的数据进行可视化展示；输出主成分中特征权重，同时利用热度图展示主成分和原始特征的相关度。

2.使用Lwf人脸数据集，多次调整PCA的n\_components参数进行特征提取，并训练分类模型，观察不同n\_components参数设置对模型准确率的影响。

3.使用sklearn内置的任意用于分离的数据集，用t-SNE进行降维，并将处置后的数据进行可视化展示.

要求：

1）针对当前数据、模型，**详叙数据预处理、模型参数调整过程及效果比对**；

2）代码要列出，重点代码加注释说明，特别是自己调试过程中的自我理解；

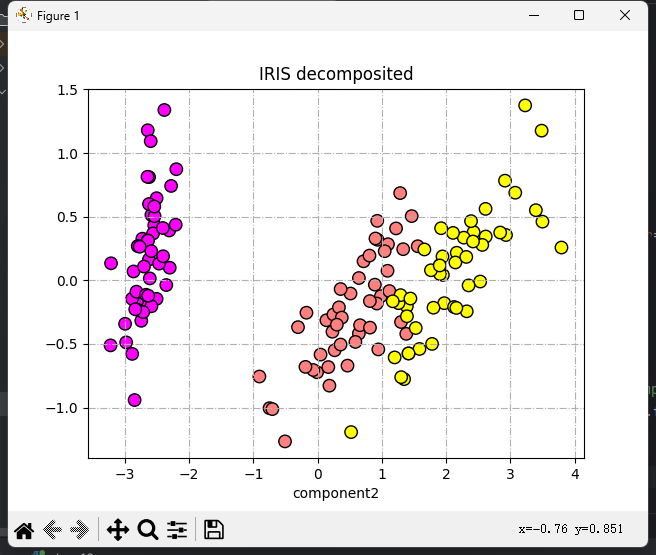
3）运行结果要截图，结果要文字说明；

4）注意排版

1.

import matplotlib.pyplot as plt  
from sklearn.decomposition import PCA  
from sklearn.datasets import load\_iris  
  
# 加载鸢尾花数据集  
iris = load\_iris()  
x = iris.data # 特征数据  
y = iris.target # 目标标签  
print(x.shape, y.shape) # 打印特征数据和目标标签的形状  
  
# 使用PCA进行降维  
pca = PCA(n\_components=2) # 指定要降到的维度为2  
x\_pca = pca.fit\_transform(x) # 对特征数据进行降维  
print(x\_pca.shape) # 打印降维后的特征数据形状  
  
# 绘制降维后的数据分布图  
plt.scatter(x\_pca[:, 0], x\_pca[:, 1], c=y, edgecolor='k', s=80, cmap='spring') # 绘制散点图  
plt.grid(linestyle='-.') # 绘制网格线  
plt.title('IRIS decomposited') # 设置标题  
plt.xlabel('component1') # 设置x轴标签  
plt.ylabel('component2') # 设置y轴标签  
plt.show() # 显示图形  
  
# 绘制主成分向量图  
plt.matshow(pca.components\_, cmap='summer') # 绘制热力图  
plt.yticks([0, 1], ['component1', 'component2']) # 设置y轴刻度标签  
plt.xticks(range(len(iris.feature\_names)), iris.feature\_names, rotation=90, ha='left') # 设置x轴刻度标签  
plt.colorbar() # 添加颜色条  
plt.show() # 显示图形  
  
# 打印主成分向量  
print(pca.components\_)

**结果：**



图表

低可信度描述已自动生成

颜色越黄，相关度越大；颜色越绿，相关度越小。

主成分1与花瓣长度最相关，与萼片宽度最不相关主成分2与萼片宽度最相关，与花瓣长度最不相关。

文本

描述已自动生成

花瓣长度和花瓣宽度在component1中具有较大的权重，而萼片长度和萼片宽度在component2中具有较大的权重。花瓣长度和花瓣宽度可能在数据集中具有较强的相关性，因为它们在同一个主成分中具有相似的权重。

2.

from sklearn.datasets import fetch\_lfw\_people  
from sklearn.model\_selection import train\_test\_split  
from sklearn.decomposition import PCA  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score  
  
# 加载LWF数据集  
lfw\_dataset = fetch\_lfw\_people(min\_faces\_per\_person=70, resize=0.4)  
  
# 划分数据集  
X = lfw\_dataset.data  
y = lfw\_dataset.target  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  
  
# 定义不同的n\_components参数值  
n\_components\_list = [10, 50, 100, 150, 200]  
  
# 记录准确率  
accuracies = []  
  
for n\_components in n\_components\_list:  
 # 初始化PCA  
 pca = PCA(n\_components=n\_components, whiten=True, random\_state=42)  
 # 应用PCA到训练数据上  
 X\_train\_pca = pca.fit\_transform(X\_train)  
 # 应用相同的PCA变换到测试数据上  
 X\_test\_pca = pca.transform(X\_test)  
  
 # 训练SVM分类器  
 clf = SVC(kernel='rbf', class\_weight='balanced', random\_state=42)  
 clf.fit(X\_train\_pca, y\_train)  
  
 # 预测并计算准确率  
 y\_pred = clf.predict(X\_test\_pca)  
 accuracy = accuracy\_score(y\_test, y\_pred)  
 accuracies.append(accuracy)  
 print(f"n\_components = {n\_components}, Accuracy = {accuracy:.2f}")  
  
# 绘制准确率随n\_components变化的曲线  
import matplotlib.pyplot as plt  
  
plt.figure()  
plt.plot(n\_components\_list, accuracies, marker='o')  
plt.title('Accuracy vs n\_components')  
plt.xlabel('n\_components')  
plt.ylabel('Accuracy')  
plt.grid(True)  
plt.show()

**结果：**

从结果可以看出，随着主成分数量的增加，准确率整体上呈现先增加后减少的趋势。随着主成分数量的增加，模型能够更好地捕捉数据的方差，从而提高了分类的准确性。

图表, 折线图

描述已自动生成

当主成分数量较小时，模型可能会欠拟合，导致准确率较低；而当主成分数量较多时，模型可能会过拟合，导致在测试集上的准确率下降。

可看出当n\_components=100时准确率最佳。

3.

from sklearn.datasets import load\_digits  
import matplotlib.pyplot as plt  
  
# 加载手写数字数据集  
digits = load\_digits()  
print("数据集样本数量及特征数量:", digits.data.shape)  
  
# 绘制部分手写数字样本图像  
fig, axes = plt.subplots(5, 2, figsize=(3, 6), subplot\_kw={'xticks': (), 'yticks': ()})  
for ax, img in zip(axes.ravel(), digits.images):  
 ax.imshow(img, cmap='Greys') # 绘制灰度图像  
plt.show()  
  
from sklearn.manifold import TSNE  
  
# 使用t-SNE进行降维  
tsne = TSNE(random\_state=42)  
digits\_tsne = tsne.fit\_transform(digits.data)  
  
# 绘制降维后的数据分布图  
colors = ["#476E2B", "#7851A8", "#BD3440", "#5A2D4E", "#879525",  
 "#A83793", "#4B655E", "#852642", "#3A3111", "#535B8A"]  
plt.figure(figsize=(9, 9))  
plt.xlim(digits\_tsne[:, 0].min(), digits\_tsne[:, 0].max() + 1)  
plt.ylim(digits\_tsne[:, 1].min(), digits\_tsne[:, 1].max() + 1)  
  
for i in range(len(digits.data)):  
 plt.text(digits\_tsne[i, 0], digits\_tsne[i, 1], str(digits.target[i]),  
 color=colors[digits.target[i]],  
 fontdict={'weight': 'bold', 'size': 9})  
plt.xlabel("t-SNE feature 0")  
plt.ylabel("t-SNE feature 1")  
plt.show()

**结果：**

文本

中度可信度描述已自动生成

图表, 散点图

描述已自动生成

采用t-SNE降维的方法，将原始的64维手写数字数据转换成了二维数据。通过标注每个数据点的数字类别，可以观察到在二维空间中不同数字之间的分布情况。t-SNE降维后的数据点更加分散，数字之间的边界更加清晰，这有助于更好地区分不同数字之间的关系。

游戏画面

低可信度描述已自动生成