主成分分析

1. 原理概述

1. 主成分分析（Principal Component Analysis，简称 PCA），其主要目标是**将高维数据映射到低维空间中**，并且希望在该低维空间中**原数据的信息量尽可能不丢失**。其中该低维空间中的特征是**全新的正交特征**（即线性无关），又称为**主成分**。
2. 为了在低维空间中尽量保留高维数据的信息，则对原始特征降维时，需要尽可能**在原始特征具有最大投影信息量的维度上进行投影**，从而使得降维后的信息量损失最小。
3. 为了实现上述要求，PCA 的工作内容就是**从原始的空间中顺序地找出一组相互正交的坐标轴（特征）**
   - 第一个坐标轴（特征）选取**原始数据中方差最大的方向**，对应于**第一主成分（PC1）**。
   - 第二个坐标轴选取**与第一个坐标轴正交的平面中原始数据方差最大的方向**，对应于**第二主成分（PC2）**。
   - 第三个坐标轴选取**与第一、第二个坐标轴正交的平面中原始数据方差最大的方向**，对应**第三主成分（PC3）**。
   - 以此类推
4. 通过数学分析，可以发现：**原始数据在 PC1 上的投影上方差最大，代表了绝大部分信息**，新的主成分所包含的信息量依次递减。因此**大部分方差包含在前 k 个坐标轴（主成分）中，后边的坐标轴（主成分）所含的方差几乎为 0**，通过此我们可以只保留包含绝大部分方差的特征维度，而忽略包含方差几乎为 0 的特征维度，从而实现对数据特征的降维处理。

> i.e. **（全新的正交）特征、主成分、坐标轴在这里为统一概念**，PCA 主要功能就是将高纬特征映射到低维的正交特征（又称之为主成分）

2. 最大方差理论（PCA 的实现步骤）

1. 在信号处理领域，我们认为**信号具有较大方差，而噪声具有较小方差**。主成分分析（Principal Component Analysis，PCA）是一种常用于数据降维的技术，其**核心思想是寻找数据中变异性最大的方向（主成分）**，以便通过选择少量主成分来近似表示原始数据。因此我们不难引出 PCA 的目标即**最大化投影方差**，也就是让**数据在主轴上投影的方差最大**。
2. **最大方差理论**的关键观点是，通过选择方差最大的方向，我们能够最大限度地保留原始数据的信息。这是因为**方差衡量了数据在某个方向上的变异性**，而选择方差最大的方向相当于选择了数据中最主要的变化方向。

> 主成分分析（PCA）的实现步骤如下，

1. 对所有**样本进行中心化**：对原始数据进行中心化，可以移除数据的均值，使得数据围绕原点分布。从而有利于 PCA 基于数据的相对位置进行降维。
2. **计算样本的协方差矩阵 XX^T**：样本 x_i 在投影方向 W （单位向量，unit vector）上的投影坐标为 x_i^TW，因此所有样本点投影后的方差可以表示为 $D(x)=\sum_i W^Tx_ix_i^TW$（等价于 $\max\limits_W W^TXX^TW$），且 $W^TW=I$。因此优化问题变为，

$$\begin{aligned} &\max_W W^TXX^TW \\ &s.t.\ W^TW=I \end{aligned} \tag{1}$$

3. **对协方差矩阵 XX^T 做特征值分解**，解得特征值和特征向量：对式 (1) 使用拉格朗日乘子法有 $XX^TW=\lambda W$，将其带入 $D(x)$ 得到 $D(x)=W^TXX^TW=W^T\lambda W=\lambda$，即**样本点投影后的方差即协方差矩阵 XX^T 的特征值**，且最大方差即协方差矩阵最大的特征值。
4. **选择主成分**：按照特征值**降序排列**特征向量，选择前几个特征向量作为主成分。这些主成分构成了一个新的坐标系，称之为主成分空间。关于主成分数量选择有以下两种方式：
   - **预先指定**低维空间的维数为 d'，则取最大的 d' 个特征值对应的特征向量 $w_1,w_2,w_3,..,w_d'$ 即可
   - **设置重构阈值**（可看作累计解释方差），记为 t，然后取使得下式成立的最小的 d' 值，再取对应数量的特征向量即可

$$\frac{\sum_{i=1}^{d'}\lambda_i}{\sum_{i=1}^{d}\lambda_i}\geq t \tag{2}$$

5. **数据投影**：将原始数据投影到选择的主成分空间中，得到降维后的数据。

3. python 实现 PCA

PCA.py

```
import numpy as np
import matplotlib.pyplot as plt


class PCA:
    """ 主成分分析类，实现维数约简 """
    """ PCA 类的使用说明
        1. pca = PCA(data, threshold=0.85, dimension=None)  首先传入高维数据，指定重构阈值[可选]，指定主成分数量[可选]
```

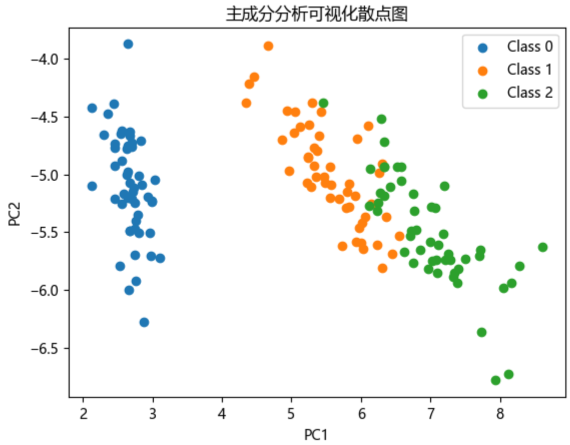
```
        2. projectedData = pca.project() 调用 project() 方法，返回投影数据
        3. pca.visualize() 对降维后的数据进行二维或三维可视化
    """

    def __init__(self, data, threshold=0.85, dimension=None):
        self.data = data  # 原始数据矩阵，n × m，样本行向量，共计 n 个样本，每个样本 m 维特征
        self.threshold = threshold  # 重构阈值（or 最小累计解释方差），用于选择主成分的数量（低维空间的维数）
        self.dimension = dimension  # 指定主成分的数量，if None，则通过默认的重构阈值选择主成分，如果该属性有值，则忽略重构阈值

        self._projectMatrix = None  # 投影矩阵，PCA 的目标
        self._cov = None  # 数据样本中心化处理之后的协方差矩阵
        self._sorted_eigenvalues = None  # 协方差矩阵的升序排列的特征值
        self._sorted_eigenvectors = None  # 协方差矩阵的特征值对应的特征向量
        self._cev = None  # 协方差矩阵的前 i 个特征值对应的累计解释方差

    def _fit(self):
        """ 核心函数，根据传入数据得到投影矩阵 """
        # 0.数据预处理（数据标准化）
        # 标准化：即一列的每个元素，减去该列的均值后，除以该列的方差
        data = np.array(self.data, dtype=float)  # 确保数据矩阵是 ndarray 类型，其中数据是浮点数
        mean = np.mean(data, axis=0)  # 求解每列均值
        std_dev = np.std(data, axis=0)  # 求每列标准差
        normalized_data = ((data - mean) / std_dev)  # 标准化后的数据矩阵

        # 1.样本中心化：即将每列的数据减去每列均值，数据预处理阶段已经完成该内容

        # 2.计算协方差矩阵 XX^T (shape=m×m)
        # p.s. 一般而言，数据矩阵 X 中一个列向量为一个样本，因此其协方差矩阵计算是 XX^T，
        # 但是此处的数据矩阵要求一个行向量为一个样本，因此其协方差矩阵计算是 X^TX
        cov = np.dot(normalized_data.T, normalized_data)
        self._cov = cov

        # 3.对协方差矩阵做特征值分解
        eigenvalues, eigenvectors = np.linalg.eig(cov)  # 特征值和对应的特征向量（列向量）

        # 4.选择主成分
        # 基于特征值对特征值和特征向量降序排列
        sort_indices = np.argsort(eigenvalues)[::-1]  # 获取特征值降序排序索引
        sorted_eigenvalues = eigenvalues[sort_indices]  # 使用索引值对特征值和特征向量进行排序
        sorted_eigenvectors = eigenvectors[:, sort_indices]
```

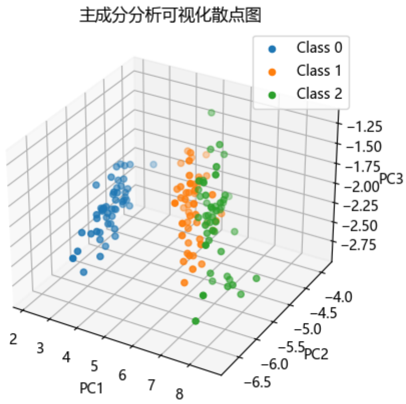
```python
        self._sorted_eigenvalues = sorted_eigenvalues
        self._sorted_eigenvectors = sorted_eigenvectors

        if not self.dimension:  # 未指定维数，则基于重构阈值选择主成分
            # 计算累计解释方差，cev[i] 表示前 i 个特征值的累计解释方差
            cev = []
            for i in range(len(sorted_eigenvalues)):
                cev_i = sum(sorted_eigenvalues[:i + 1]) /
sum(sorted_eigenvalues)
                cev.append(cev_i)
            self._cev = cev
            # 基于重构阈值，选择主成分
            for i in range(len(cev)):
                if cev[i] ≥ self.threshold:  # 满足重构阈值，则确定投影矩阵
                    self._projectMatrix = sorted_eigenvectors[:, :i + 1]
                    break
        else:  # 基于指定维数选择主成分
            if self.dimension > self.data.shape[1]:
                raise Exception("illegal dimension")
            self._projectMatrix = sorted_eigenvectors[:, :self.dimension + 1]

        # 5.数据投影：将原始数据投影到主成分空间中
        # 需调用 PCA 的方法，project

    def project(self):
        """ 获取投影后的数据（低维数据） """
        # self.data n 行样本，m 维特征；self.projectMatrix m 维载荷因子，k 个主成分
（PCA）
        self._fit()

        projectedData = np.dot(self.data, self._projectMatrix)

        return projectedData

    def visualize(self, target, dimension=2):
        """ 对降维后的数据进行二维的可视化操作（即只选择两个主成分） """
        # target 是每个样本的分类，基于此可以对相同的样本点以相同的颜色表示
        self.dimension = dimension
        # 获取样本的类别情况
        unique_labels = np.unique(target)
        # 将数据和标签拼接
        target = target.reshape(-1, 1)
        data = np.concatenate((self.project(), target), axis=1)
        # 替换为系统中存在的中文字体
        plt.rcParams['font.sans-serif'] = 'Microsoft YaHei'
```

```python
        if self.dimension == 2:  # 二维可视化
            # 可视化
            for label in unique_labels:
                class_points = data[data[:, -1] == label]  # 获取类别为 label 的样本点
                plt.scatter(class_points[:, 0], class_points[:, 1], label=f"Class {label}")

            plt.xlabel("PC1")
            plt.ylabel("PC2")
            plt.title("主成分分析可视化散点图")
            plt.legend()
            plt.show()
        elif self.dimension == 3:  # 三维可视化
            figure = plt.figure()
            ax = figure.add_subplot(111, projection="3d")

            # 可视化
            for label in unique_labels:
                class_points = data[data[:, -1] == label]  # 获取类别为 label 的样本点
                ax.scatter(class_points[:, 0], class_points[:, 1], class_points[:, 2], label=f'Class {label}')

            ax.set_xlabel("PC1")
            ax.set_ylabel("PC2")
            ax.set_zlabel("PC3")
            ax.set_title("主成分分析可视化散点图")
            ax.legend()
            plt.show()


if __name__ == "__main__":
    # 载入数据
    from sklearn.datasets import load_iris

    iris_dataset = load_iris()
    iris_target = iris_dataset['target']  # 数据标签
    iris_target_names = iris_dataset['target_names']  # 数据标签名
    iris_data = iris_dataset['data']  # 数据集
    iris_feature_names = iris_dataset['feature_names']  # 数据集特征名

    # 主成分分析
    pca = PCA(iris_data)

    # 可视化主成分分析后的结果
```

```
    pca.visualize(iris_target, 2)  # 二维可视化
    pca.visualize(iris_target, 3)  # 三维可视化
```

散点图可视化





4. sklearn 中的 PCA

PCA_sklearn.py

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np

iris_dataset = load_iris()
iris_target = iris_dataset['target']  # 数据标签
iris_target_names = iris_dataset['target_names']  # 数据标签名
iris_data = iris_dataset['data']  # 数据集
iris_feature_names = iris_dataset['feature_names']  # 数据集特征名
```

```python
pca = PCA()
iris_data_projected = pca.fit_transform(iris_data)

# 二维可视化
# 获取样本的类别情况
unique_labels = np.unique(iris_target)
# 将数据和标签拼接
target = iris_target.reshape(-1, 1)
data = np.concatenate((iris_data_projected, target), axis=1)
# 替换为系统中存在的中文字体
plt.rcParams['font.sans-serif'] = 'Microsoft YaHei'

for label in unique_labels:
    class_points = data[data[:, -1] == label]  # 获取类别为 label 的样本点
    plt.scatter(class_points[:, 0], class_points[:, 1], label=f"Class {label}")

plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("主成分分析可视化散点图")
plt.legend()
plt.show()
```

散点图可视化