人工智能实验：Topic 4——降维技术

Part 3：局部保持投影

3190102060　黄嘉欣

一、LPP 概述

作为一种常用的降维技术，局部保持投影能够在降维的同时保留空间中样本的局部邻域结构，避免样本集的发散，保持原来的近邻结构。因此，LPP 的目标是给定在R^n空间的$\boldsymbol{x}_1$，$\boldsymbol{x}_2$，…，$\boldsymbol{x}_m$的情况下，通过映射矩阵 $\boldsymbol{A}$，将 m 个点映射到$R^{n'}$空间的$\boldsymbol{y}_1$，$\boldsymbol{y}_2$，…，$\boldsymbol{y}_m$，即$\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{x}_i$。满足$n' < n$且降维的同时保留空间中样本的局部邻域结构。令目标函数为：$\sum_{ij}(\boldsymbol{y}_i - \boldsymbol{y}_j)^2 W_{i,j}$，其中$W_{i,j} = e^{-\frac{||x_i - x_j||^2}{t}}$。当$\boldsymbol{x}_i$与$\boldsymbol{x}_j$离得比较远时，可直接将$W_{i,j}$设为 0，则目标函数为 0；否则$W_{i,j}$比较大，故需要保证$\boldsymbol{y}_i$，$\boldsymbol{y}_j$也要近，才能使目标函数变小。假设$\boldsymbol{a}$为变换矩阵，即$\boldsymbol{y}^T = \boldsymbol{a}^T\boldsymbol{X}$，于是可将上述目标函数转化为：$\boldsymbol{a}^T\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^T\boldsymbol{a}$，其中$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$，$\boldsymbol{D}$是对角矩阵，且满足$D_{i,i} = \sum_j W_{i,j}$。由于$D_{i,i}$越大，其对应的$y_i$所占比重（重要性）越高，因此我们设定一个约束条件：$\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y} = \boldsymbol{1}$，即$\boldsymbol{a}^T\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^T\boldsymbol{a} = \boldsymbol{1}$，最后使目标函数转变为 arg min $\boldsymbol{a}^T\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^T\boldsymbol{a}$。利用拉格朗日乘子法，我们可以计算出$\boldsymbol{a}$的解为$(\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^T$的$n'$个最小非零特征值所对应的特征向量组成的矩阵；$\boldsymbol{X}^T\boldsymbol{a}$即为降维后的数据。

二、实验 4-4

① 实验题目

利用 sklearn.datasets.load_digits 函数，导入手写数字数据集作为$\boldsymbol{X}^T$，通过 LPP 对生成的随机数据进行降维（n_dim=2），并可视化降维后的数据。具体的实现步骤为：① 定义函数——计算邻接矩阵$\boldsymbol{W}$，输入变量为 data，n_neighbour，t；② 定义函数——LPP，输入变量为 data，n_dim，n_neighbour，t，计算$\boldsymbol{D}$矩阵、$\boldsymbol{L}$矩阵以及$\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^T$、$\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^T$，在此基础上对$(\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^T$进行特征值分解，提取其 n_dim 个最小非零特征值所对应的特征向量，计算降维后的数据。

② 实验结果与分析

如图 2.1，为热核参数$t = 0.001$时 LPP 降维效果与 PCA 降维后数据的比较。可以发现，通过 LPP 降维后得到的数据比较紧凑，样本空间中的局部邻域都有所保留；而 PCA

得到的结果相对而言更加发散，与预期的结果相一致。除此之外，改变热核参数t，LPP的降维输出发生了较大的变化。当其从0.001增大到0.01时，降维后的数据变得发散，这表明当t较小时，可以改善算法表现的质量。

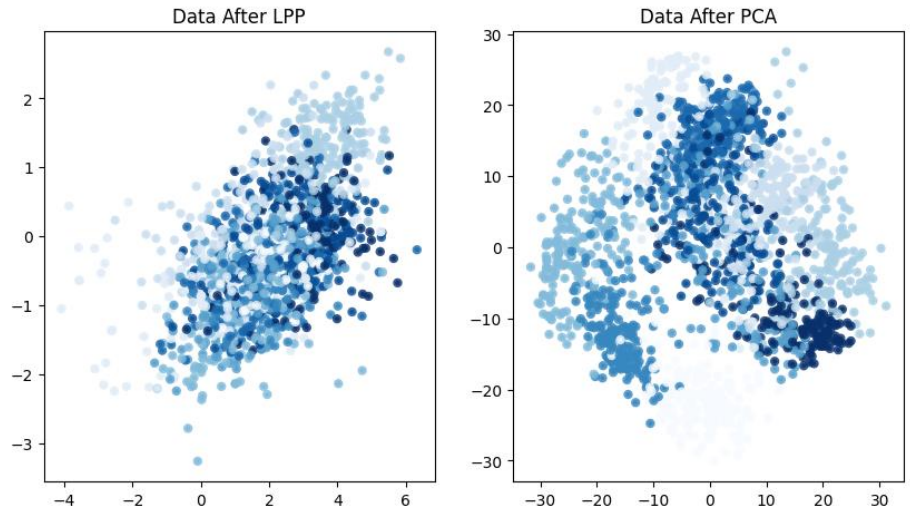

图 2.1 $t = 0.001$时降维后数据比较

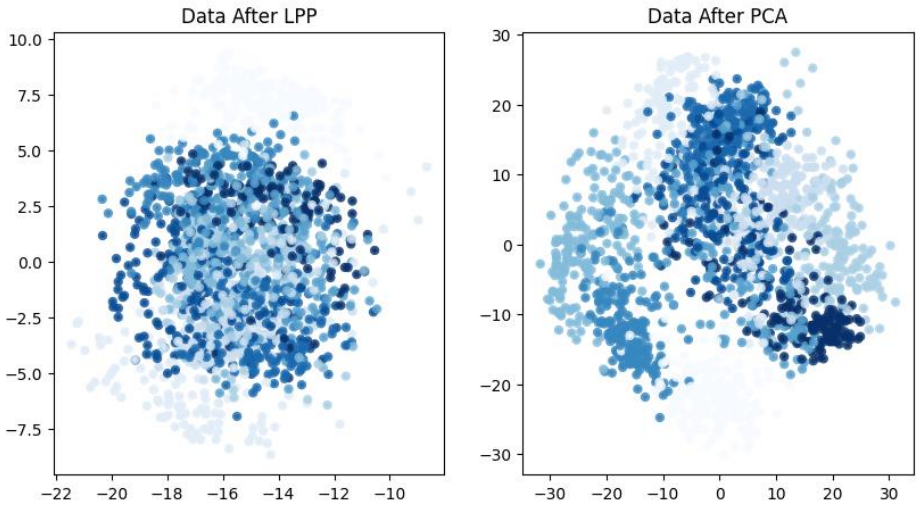

图 2.2 $t = 0.01$时降维后数据比较

三、附录：实验 Python 代码——*lpp.py*

```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from pca import *
```

```
def compute_W(data,n_neighbour,t):
    """计算邻接矩阵
    Args:
      data: 数据集
      n_neighbour: 邻接点个数
      t: 热核参数
    Returns:
      W: 邻接矩阵
    """
    sum = np.sum(np.square(data),1)   # 计算任意两个样本之间的距离平方
    sqrDist = np.add(np.add(-2*np.dot(data, data.T), sum).T,sum)
    oriW = np.exp(-sqrDist/t)         # 初步计算 W，未区分近邻关系
    dimen = oriW.shape[0]             # W 的维度
    W = np.zeros((dimen,dimen))       # 初始化 W
    for i in range(dimen):        # 找到与每个样本最近的 n_neighbour 个样本
        index = np.argsort(sqrDist[i])[1:1+n_neighbour]
        W[i,index] = oriW[i,index]    # 保存 W_{i,j}
        W[index,i] = oriW[index,i]    # 对称性
    return W


def LPP(data,n_dim,n_neighbour,t):
    """使用 LPP 降维
    Args:
      data: 数据集
      n_dim: 希望降到的维度
      n_neighbour: 邻接点个数
      t: 热核参数
    Returns:
      lowDData: 降维后的矩阵
    """
    W = compute_W(data,n_neighbour,t)  # 建立邻接矩阵
    dimen = data.shape[0]
    D = np.zeros((dimen,dimen))
    for i in range(dimen):            # 计算 D 矩阵
        D[i,i] = np.sum(W[i])
    L = D - W                         # 计算 L 矩阵
    XDXT = np.dot(np.dot(data.T,D),data)       # 计算 XDXT
    XLXT = np.dot(np.dot(data.T,L),data)       # 计算 XLXT
    matrix = np.dot(np.linalg.pinv(XDXT),XLXT) # (XDXT)^(-1)XLXT
    eigValues,eigVecs = np.linalg.eig(matrix)  # 特征值分解
    sortedIndex = np.argsort(eigValues)        # 对特征值从小到大排序
    sortedEigValues = eigValues[sortedIndex]   # 按下标排序
    start = 0
    while sortedEigValues[start] < 1e-6:
```

```
        start += 1
    # 不接近 0 的前 n_dim 个特征值的索引
    remEigenIndex = sortedIndex[start:start+n_dim]
    remEigVecs = eigVecs[:,remEigenIndex]          # 特征值对应的特征向量
    lowDData = np.dot(data,remEigVecs)             # 降维
    return lowDData


def main():
    data = datasets.load_digits().data  # 导入数据集
    label = datasets.load_digits().target
    sum = np.sum(np.square(data),1)     # 计算任意两个样本之间的距离平方
    sqrDist = np.add(np.add(-2*np.dot(data, data.T), sum).T,sum)
    t = 0.001*np.max(sqrDist)           # 设定热核参数
    lowDData = LPP(data,2,5,t)          # lpp 降维
    pca_lowDData,reconsMat = pca(data,2)  # pca 降维
    plt.subplot(1,2,1)                  # LPP 降维后数据可视化
    l1 = plt.scatter(lowDData[:,0],lowDData[:,1],c=label,marker='.',
                     cmap='Blues',alpha=0.8,linewidths=3)
    plt.title("Data After LPP")
    plt.subplot(1,2,2)                  # PCA 降维后数据可视化
    l2 = plt.scatter(pca_lowDData[:,0].tolist(),
                     pca_lowDData[:,1].tolist(),c=label,marker='.',
                     cmap='Blues',alpha=0.8,linewidths=3)
    plt.title("Data After PCA")
    plt.show()


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

注：代码中 pca()方法由 lab4 实现。