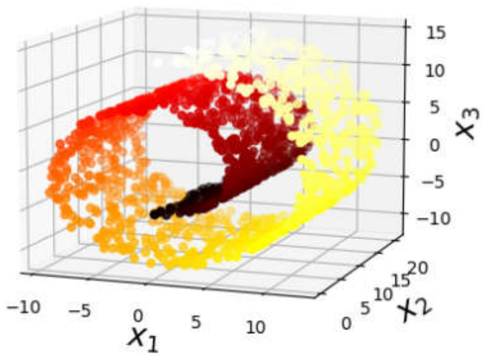
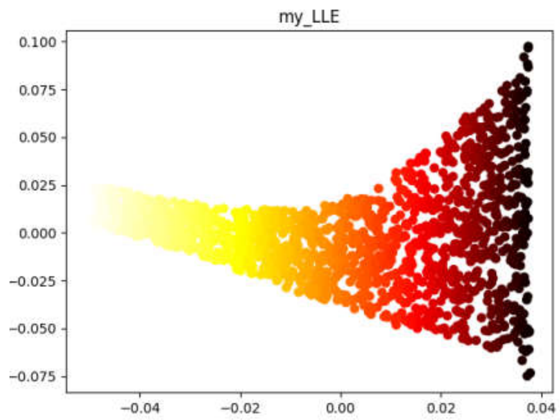
Python编程与人工智能实践

算法篇：数据降维-LLE（Locally Linear Embedding）

局部线性嵌入





于泓
鲁东大学
信息与电气工程学院
2021.10.6

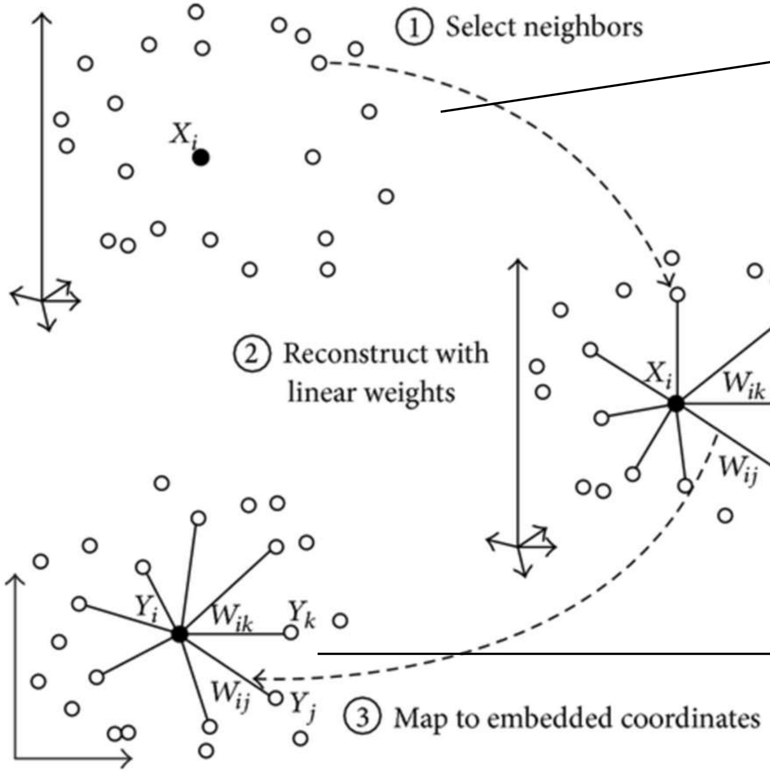
LLE（Locally Linear Embedding，局部线性嵌入）

LLE（局部线性嵌入）即"Locally Linear Embedding"的降维算法，
属于"流形"（ manifold ） 降维。在低维空间可以表示高维空间的形状

本法主要原理： 在高维空间中，局部点之间的相对位置关系，在低维空间依旧可以保持

对比： **MSD算法**：保留 点与点之间的关系（**考量单独点**）

LLE 算法：保留 局部分点簇之间的相对关系 （**考量局部多个点**）



为**中心点**，寻找**临近点**

用**临近点**表达**中心点** （**局部点簇结构**）

低维点簇依据保持高维点簇之间的**相对位置关系**（**点簇结构**）

步骤（1）高维样本点 X_i 由它的k个临近点进行近似 其中上标(k) 表示k个临近点

$$\arg\min(\mathrm{w}_i)=\frac{1}{2}\left\|X_i-\mathrm{w}_i\mathbf{X}_i^{(k)}\right\|^2$$

其中 $X_i\in[1,D]$ $\mathrm{w}_i\in[1,k]$ $\mathbf{X}_i^{(k)}\in[k,D]$

且要求 $\sum_{j=1}^{k}w_{ij}=1$ 即 $\mathrm{w}_i\,\mathrm{I}_{k\times1}=1$

$$\left\|X_i-\mathrm{w}_i\mathbf{X}_i^{(k)}\right\|^2=\left\|\mathrm{w}_i\,\mathrm{I}_{k\times1}X_i-\mathrm{w}_i\mathbf{X}_i^{(k)}\right\|^2$$

$$=\left\|\mathrm{w}_i\left(\mathrm{I}_{k\times1}X_i-\mathbf{X}_i^{(k)}\right)\right\|^2$$

$$=\mathrm{w}_i(\mathrm{I}_{k\times1}X_i-\mathbf{X}_i^{(k)})(\mathrm{I}_{k\times1}X_i-\mathbf{X}_i^{(k)})^T\mathrm{w}_i^T=\mathrm{w}_i\mathbf{S}_i\mathrm{w}_i^T$$

构建拉格朗日方程

$$\mathrm{L}(\mathrm{w}_i,\ \lambda)=\frac{1}{2}\mathrm{w}_i\mathbf{S}_i\mathrm{w}_i^T-\lambda(\mathrm{w}_i\,\mathrm{I}_{k\times1}-1)$$

求导可得 $\dfrac{\partial\,\mathrm{L}(\mathrm{w}_i)}{\partial\mathrm{w}_i}=0$

$$\mathrm{w}_i\mathbf{S}_i=\lambda\mathrm{I}_{k\times1}^T$$

$$\mathrm{w}_i=\lambda\mathrm{I}_{k\times1}^T\mathbf{S}^{-1}$$

$$\mathrm{w}_i\mathrm{I}_{k\times1}=\lambda\mathrm{I}_{k\times1}^T\mathbf{S}^{-1}\mathrm{I}_{k\times1}=1\rightarrow\lambda=\frac{1}{\mathrm{I}_{k\times1}^T\mathbf{S}^{-1}\mathrm{I}_{k\times1}}$$

带入

$$\mathrm{w}_i=\frac{\mathrm{I}_{k\times1}^T\mathbf{S}^{-1}}{\mathrm{I}_{k\times1}^T\mathbf{S}^{-1}\mathrm{I}_{k\times1}}$$

列元素的和

全部元素的和

步骤二 进行低维空间映射

映射后的低维空间点 Y_i

能够由k个临近点 $\mathbf{Y}_i^{(k)}$ → k个临近点的编号（index不变）

利用步骤一 学的的权重 w_i 进行重构

$$\arg\min(\mathbf{Y}) = \sum_{i=1}^{N}\left\|\mathrm{Y}_i - \mathrm{w}_i \mathbf{Y}_i^{(k)}\right\|^2 \qquad \mathbf{Y} \in [N, d]$$

约束条件：低维特征，**每个维度均值为0，方差为1**

可以通过偏置调节

$$\frac{1}{N}\mathbf{Y}^T\mathbf{Y} = \mathbf{I}_{d\times d}$$

单位阵

$$\sum_{i=1}^{N}\left\|\mathrm{Y}_i - \mathrm{w}_i \mathrm{Y}_i^{(k)}\right\|^2 = \sum_{i=1}^{N}\left\|\mathrm{Y}_i - \mathrm{W}_i \mathbf{Y}\right\|^2$$

$\mathrm{W}_i \in [1, N]$ 其中 k个近邻元素的位置由 w_i 中的元素填充，其他位置为0

上式可以进一步写为：

$$\arg\min(\mathbf{Y}) = \mathrm{trace}\left((\mathbf{Y} - \mathbf{WY})(\mathbf{Y} - \mathbf{WY})^T\right)$$

$$= \mathrm{trace}\left(\mathbf{Y}(\mathbf{I} - \mathbf{W})(\mathbf{I} - \mathbf{W})^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\right)$$

结合限制条件：构建拉格朗日函数

$$\mathrm{L}(\mathbf{Y}, \lambda) = \mathrm{trace}\left(\mathbf{Y}(\mathbf{I} - \mathbf{W})(\mathbf{I} - \mathbf{W})^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\right) - \lambda(\mathbf{Y}^T\mathbf{Y} - \frac{1}{N}\mathbf{I}_{d\times d})$$

$$\mathrm{L}(\mathbf{Y},\lambda)=\mathrm{trace}\left(\mathbf{Y}\left(\mathbf{I}_{N\times N}-\mathbf{W}\right)\left(\mathbf{I}_{N\times N}-\mathbf{W}\right)^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\right)-\lambda(\mathbf{Y}^{T}\mathbf{Y}-\frac{1}{N}\mathbf{I}_{d\times d})$$

求导可得

$$\mathbf{Y}\left(\mathbf{I}_{N\times N}-\mathbf{W}\right)^{T}\left(\mathbf{I}_{N\times N}-\mathbf{W}\right)=\lambda\mathbf{Y}$$

可以发现降维后的特征 $\mathbf{Y}$

就是矩阵 $\left(\mathbf{I}_{N\times N}-\mathbf{W}\right)^{T}\left(\mathbf{I}_{N\times N}-\mathbf{W}\right)$ 的 特征向量

由于要求最小值，所以应当选取d个**最小**的特征值所对应的特征向量

最小的特征值往往接近**0，**所以实际应用中，选取**除最小特征值外**最小的$\boldsymbol{d}$个特征值所对应的特征向量

作为降维后的数据

LLE算法的一般步骤

（1）为高维数据 $\mathbf{X}$ $\mathbf{X} \in [N, D]$ 中的每个样本

X_i 选取k个临近点 $\mathbf{X}_i^{(k)}$ $\mathbf{X}_i^{(k)} \in [k, D]$

（2）为每个样本点计算一组权重 $\mathrm{w}_i \in [1, k]$

可以借助这组权重，用 $\mathbf{X}_i^{(k)}$ 重构 X_i

$$\mathrm{w}_i = \frac{\mathrm{I}_{k\times1}^T \mathbf{S}_i^{-1}}{\mathrm{I}_{k\times1}^T \mathbf{S}_i^{-1} \mathrm{I}_{k\times1}}$$

$$\mathbf{S}_i = (\mathrm{I}_{k\times1} \mathrm{X}_i - \mathbf{X}_i^{(k)})(\mathrm{I}_{k\times1} \mathrm{X}_i - \mathbf{X}_i^{(k)})^T$$

（3）将 w_i $\mathrm{w}_i \in [1, k]$ 扩充为 W_i $\mathrm{W}_i \in [1, N]$

k个临近点的位置， $W_{ik} = w_{ik}$ 其他位置为0

收集N个 W_i 组成 $\mathbf{W}$

$$\mathbf{C} = (\mathbf{I}_{N\times N} - \mathbf{W})^T (\mathbf{I}_{N\times N} - \mathbf{W})$$

对 $\mathbf{C}$ 进行特征值分解

选取**除最小特征值外**，最小的d个特征值
所对应的特征向量 $\mathbf{Y}$ $\mathbf{Y} \in [N, d]$

就是降维后的特征点。

代码实现

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_s_curve
from sklearn.manifold import Isomap
from tqdm import tqdm
from sklearn.manifold import LocallyLinearEmbedding

# x 维度 [N,D]
def cal_pairwise_dist(x):

    N,D = np.shape(x)

    dist = np.zeros([N,N])

    for i in range(N):
        for j in range(N):
            dist[i,j] = np.sqrt(np.dot((x[i]-x[j]),(x[i]-x[j]).T))

    #返回任意两个点之间距离
    return dist
```

```
# 获取每个样本点的 n_neighbors个临近点的位置
def get_n_neighbors(data, n_neighbors = 10):
    dist = cal_pairwise_dist(data)
    dist[dist < 0] = 0
    N = dist.shape[0]
    Index = np.argsort(dist,axis=1)[:,1:n_neighbors+1]
    return Index
```

```
def scatter_3d(X, y):
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.hot)
    ax.view_init(10, -70)
    ax.set_xlabel("$x_1$", fontsize=18)
    ax.set_ylabel("$x_2$", fontsize=18)
    ax.set_zlabel("$x_3$", fontsize=18)
    plt.show(block=False)
```

```python
# data : N,D
def lle(data, n_dims = 2, n_neighbors = 10):
    N,D = np.shape(data)
    if n_neighbors > D:
        tol = 1e-3
    else:
        tol = 0
    # 获取 n_neighbors个临界点的位置
    Index_NN = get_n_neighbors(data,n_neighbors)

    # 计算重构权重
    w = np.zeros([N,n_neighbors])
    for i in range(N):

        X_k = data[Index_NN[i]]  #[k,D]
        X_i = [data[i]]          #[1,D]
        I = np.ones([n_neighbors,1])

        Si = np.dot((np.dot(I,X_i)-X_k), (np.dot(I,X_i)-X_k).T)

        # 为防止对角线元素过小
        Si = Si+np.eye(n_neighbors)*tol*np.trace(Si)

        Si_inv = np.linalg.pinv(Si)
        w[i] = np.dot(I.T,Si_inv)/(np.dot(np.dot(I.T,Si_inv),I))
```

$$w_i = \frac{\mathbf{I}_{k\times1}^T \mathbf{S}_i^{-1}}{\mathbf{I}_{k\times1}^T \mathbf{S}_i^{-1} \mathbf{I}_{k\times1}}$$

$$\mathbf{S}_i = (\mathrm{I}_{k\times1} \mathrm{X}_i - \mathbf{X}_i^{(k)})(\mathrm{I}_{k\times1} \mathrm{X}_i - \mathbf{X}_i^{(k)})^T$$

```python
    # 计算 W
    W = np.zeros([N,N])
    for i in range(N):
        W[i,Index_NN[i]] = w[i]

    I_N = np.eye(N)
    C = np.dot((I_N-W).T,(I_N-W))

    # 进行特征值的分解
    eig_val, eig_vector = np.linalg.eig(C)

    index_ = np.argsort(eig_val)[1:n_dims+1]

    y = eig_vector[:,index_]
    return y
```

$$\mathbf{C} = (\mathbf{I}_{N\times N} - \mathbf{W})^T (\mathbf{I}_{N\times N} - \mathbf{W})$$

实验结果

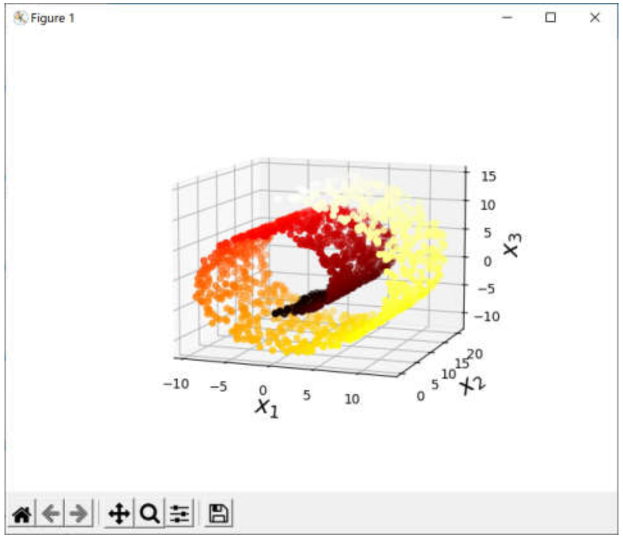


Figure 1
15
10
5
0
−5
−10
x_3
−10
−5
0
5
10
x_1
0
5
10
15
20
x_2

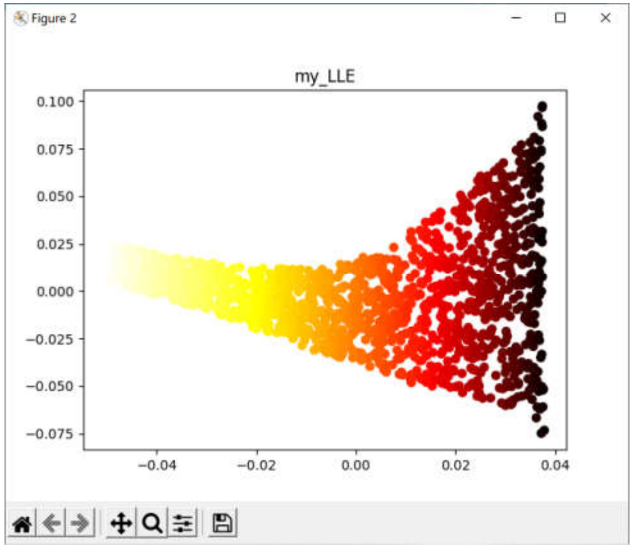


Figure 2
my_LLE
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.04
−0.02
0.00
0.02
0.04

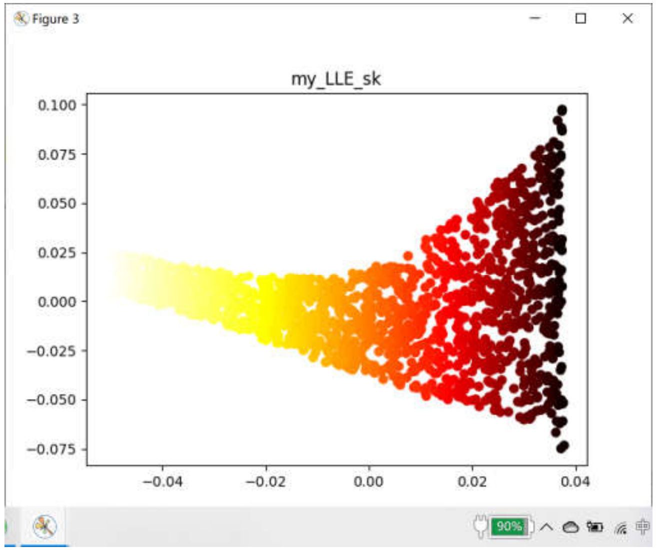


Figure 3
my_LLE_sk
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.04
−0.02
0.00
0.02
0.04
90%