Report of Locally Linear Embedding

Junjie Zhang
800 Dongchuan Road, Minghang District, Shanghai, 20024
Joint Institute, Shanghai Jiaotong University
bigzhang@sjtu.edu.cn

Introduction

Background Information

Data coming from real-world tends to be high dimensional, which challenges the computational efficiency and theoretical analysis. Besides, the raw data may contain irrelevant or redundant information. So, dimensionality reduction is need to solve these problems and provides bonus advantages such as helping understand and visualize the underlying structure of data[2].

Problem Statement

Traditional unsupervised methods are linear, such as principal component analysis (PCA, [10]), multidimensional scaling (MDS, [9]), independent component analysis (ICA, [4]), etc. However, its limitation on the linear underlying structure becomes a drawback since most of the cases are nonlinear.

Meaning of the Problem

Thus, nonlinear dimensionality reduction methods are developed such as generative topographic mapping (GTM, [1]), self-organising map (SOM, [8]), locally linear embedding (LLE, [6, 7]), etc. Compared with other nonlinear methods, the LLE, which we'll present in this paper, owns advantages that it's easy to be implemented and will not trap in local optimal [7]. Along with LLE, the concepts of manifold learning has been proposed. Here is an example, as you can see in 1, (B) is three dimensional data (B) sampled from two dimensional manifolds (A). The LLE algorithm can do the neighborhood-preserving mappings shown in (C). [6].

Solution

Algorithm of LLE

The key idea of LLE is to detect locally linear relationship by representing a point by its neighbors, then maintain that relation in lower dimension space. Basically, the algorithm has three steps, which are selecting neighbors, reconstructing with linear weights, mapping into embedded coordinates [6]. Let $n, D, d, i \in \mathbb{N}^*$ and $i < n$. Given a data set $X = \{x_1, x_2, ..., x_n\}$, where every $x_i \in \mathbb{R}^D$. Then, LLE is to find a low-dimension embeddings, $Y = y_1, y_2, ..., y_n \in \mathbb{R}^{n*d}$, where $d < D$, the intrinsic dimension of data.

1. Selecting neighbors

LLE uses k-nearest neighbors (KNN) to select neighbors. KNN takes use of one property of manifold, every point in a manifold has a neighborhood homeomorphic to an open subset of Euclidean space, which makes it possible to measure neighbors by Euclidean distance.

Preprint. Under review.

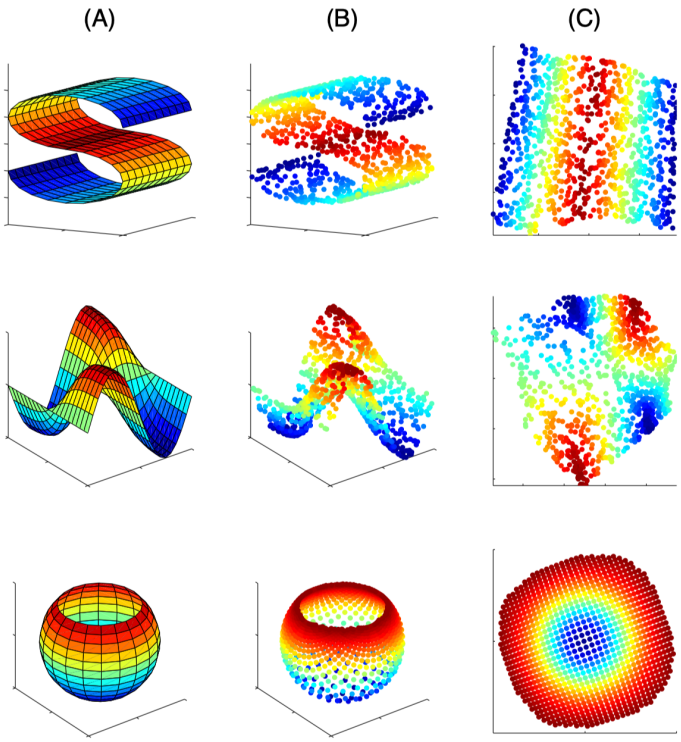

Figure 1: LLE works on the nonlinear dimensionality reduction problem [7]

For a data point x_i, we can denote the set its neighbors as $N(x_i) = \{\theta_{i1}, \theta_{i2}, ..., \theta_{ik}\}$, which contains k nearest data points within Euclidean distance.

2. Reconstructing with linear weights,

This step is to find a convex combination of neighbors to represent each x_i. Denote the weight for neighbors as $w \in \mathbb{R}^{n \times k}$. For the first time, w should be initialized first. To get the optimal w, we should minimize the distance between convex combination of neighbors and x_i, which is that for each x_i

$$\min_w \sum_{i=0}^{n} ||x_i - \sum_{x_j \in N(x_i)} w_{ij} x_j||_2^2,$$

$$s.t. \quad \sum_{j=1}^{k} w_{ij} = 1.$$

Clearly, we can optimize w_i w.r.t. each x_i independently.
Denote $N_i = (\theta_1\ \theta_2\ ...\ \theta_k)$, $G_i = (\theta_{i1} - x_i\ \theta_{i2} - x_i\ ...\ \theta_{ik} - x_i) \in \mathbb{R}^{n \times k}$, $Q_i = G_i^T G_i$ and $\Gamma = (1\ 1\ ...1)^T \in \mathbb{R}^{k \times 1}$. Obviously, Q_i is symmetric and semi-definite. Then the optimization problem is that for each i:

$$\min_{w_i} w_i^T Q_i w_i,$$

$$s.t. w_i^T \Gamma = 1.$$

Use Lagrange multiplier here, then the optimized w_i is

$$w_i^* = \frac{Q_i^{-1}\Gamma}{\Gamma^T Q_i^{-1}\Gamma}$$

Unusually, when $k > D$, Q_i or the data points are not in general position, the regularized problem should be solved.

$$Q_i \leftarrow Q_i + \frac{\Delta^2}{k_i}\text{Tr}(Q_i)I$$

, where $\text{Tr}(Q_i)$ the trace of Q_i, $\Delta^2 \ll 1$ the regularization parameter [7]. Then, denote the weight matrix $W \in \mathbb{R}^{n\times n}$, where

$$W_{ij} = \begin{cases} w_{ij}, \text{ if } x_j \in N(x_i) \\ \\ 0, \text{ otherwise} \end{cases}$$

3. Mapping into embedding coordinates

This is to find the low-dimension embeddings which best preserve the neighbor relations. Mathematically,

$$\min_y \sum_{i=1}^{n} ||y_i - \sum_{j=1}^{k} w_{ij}\gamma_i j||,$$

$$as.t. \sum_i^n y_i = 0, \frac{1}{n}\sum_i^D y_i y_i^T = I.$$

, where γ_{ij} the corresponding low-dimension embedding of θ_{ij}. And the two constraints prevent the embedding cost function be affected by translation or scaling. Then, denote $M = (I - W)^T(i - W)$.

Then, the $d + 1$ eigenvectors associated with $d + 1$ smallest eigenvalues are computed. Furthermore, the eigenvector, which is associated with zero eigenvalue is abandoned. Then, the remaining d eigenvectors form the Y.

Implemention

The algorithm is implemented by python. The code and its usage can be seen in **Appendix**.

Tests and Results

In this section, we test our LLE algorithms within three different kinds of manifolds and compare the results with results obtained by using `sklearn` library's LLE and PCA. This three manifolds are three-dimensional and we want to reduct it in to two dimensianl space. Within the test, 1000 samples are generated with noise, and we choose the number of neighbours as 20.

1. Swiss roll shape manifold

. In this section, the samples are in a shape of Swiss roll with noise, which can be seen in Figure 2. And the result of my LLE, `sklearn`'s LLE and `sklearn`'s PCA can be seen in Figure 3. As it can be seen, my LLE and `sklearn`'s LLE reaches the same performance, which expand the roll and explain the under structure. However, with linear kernel, PCA fail to explain the under structure and still maintain the roll structure. In this case, we can see that LLE will be helpful to explain the nonlinear relationship in high-dimensional space and my LLE reaches the same performance as `sklearn`'s LLE.

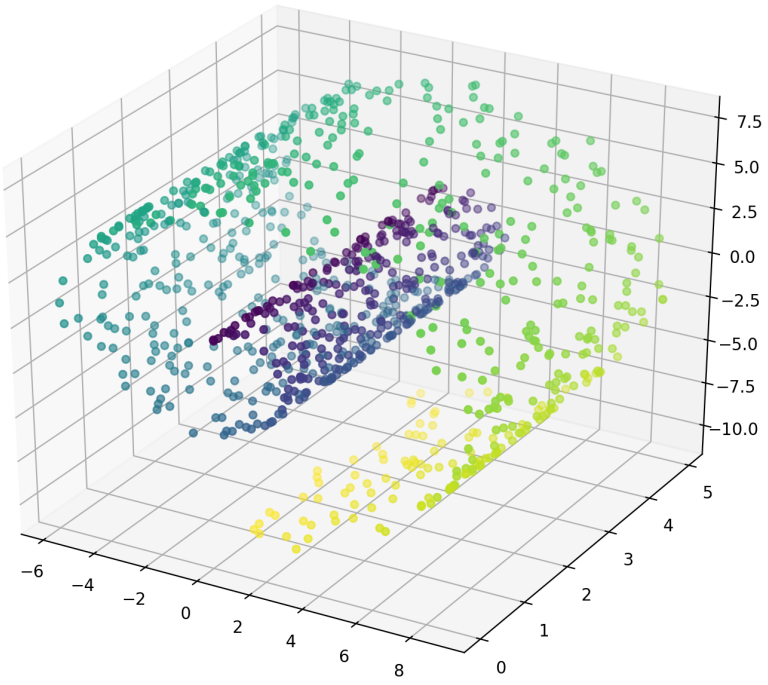

Figure 2: Generated data in shape of Swiss roll

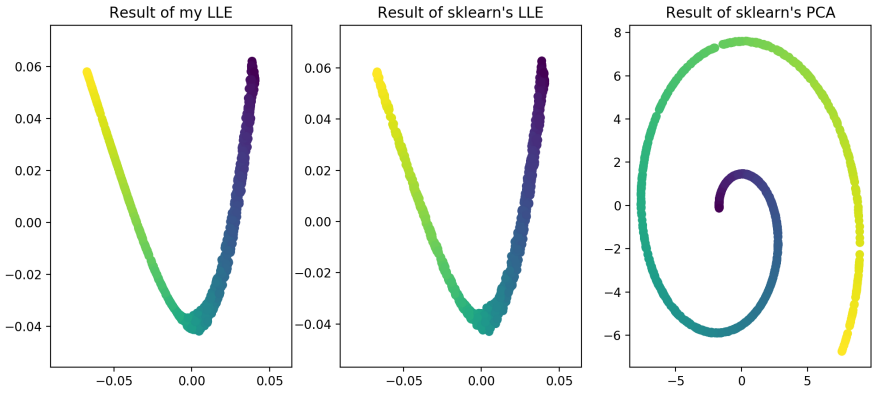

Figure 3: Results of my LLE, `sklearn`'s LLE and `sklearn`'s PCA in Swiss roll manifold

2. S shape manifold

In this section, the manifold we choose is in shape of S, which can be seen in Figure 4. And the result of LLE, `sklearn`'s LLE and `sklearn`'s PCA can be seen in Figure 5. From the result we can see that, again, my LLE algorithm has the same performance as `sklearn`'s algorithm. As for the PCA, it can barely explain the structure near the end and mix the data from end and middle-end. Thus, in this case, we can say that LLE will be helpful to explain the nonlinear relationship in high-dimensional space and my LLE reaches the same performance as `sklearn`'s LLE.

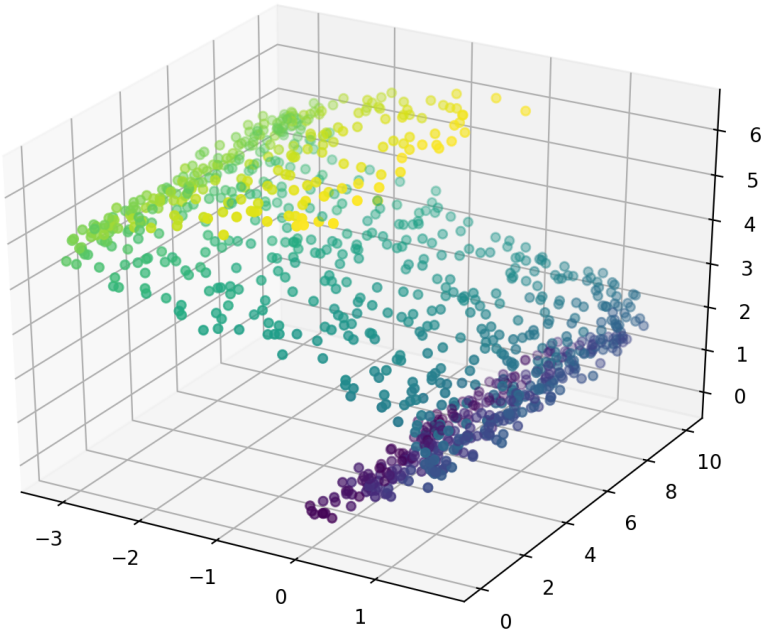

Figure 4: Generated data in shape of S

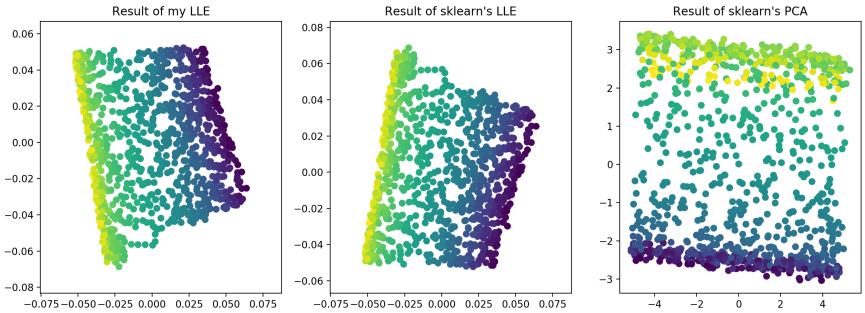

Figure 5: Results of my LLE, `sklearn`'s LLE and `sklearn`'s PCA in S shape manifold

3. Bowl shape manifold

In this section, the manifold is in shape of bowl, which can be seen in Figure 6. And the result of LLE, `sklearn`'s LLE and `sklearn`'s PCA can be seen in Figure 7. From the result, it can be seen that my LLE algorithm has the same performance as `sklearn`'s LLE. Both of them detect the under structure of the bowl shape manifold and explain it in two dimensional space. However, for PCA, it attemps to explain the structure in another point of view but fail since it mix the points in the upper end with points in the upper middle. Thus, in this case, we can say that LLE will be helpful to explain the nonlinear relationship in high-dimensional space and my LLE reaches the same performance as `sklearn`'s LLE.

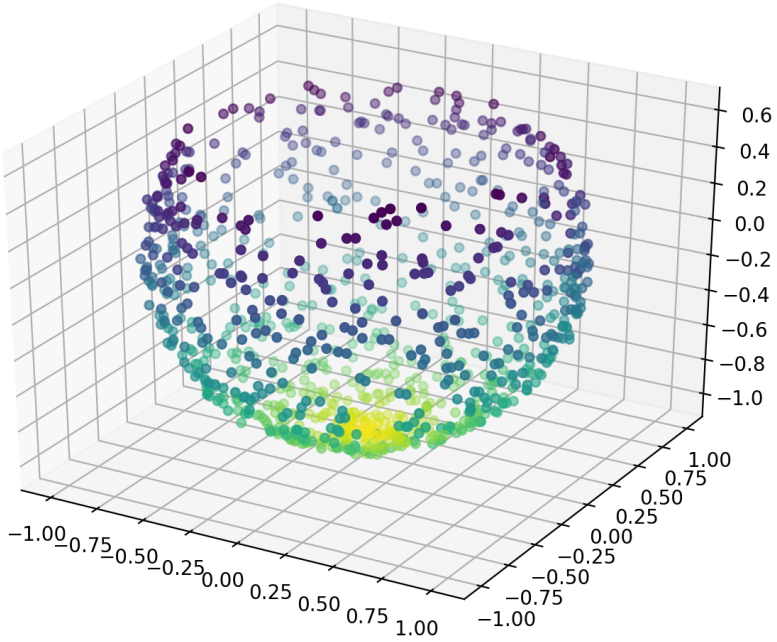

Figure 6: Generated data in shape of bowl

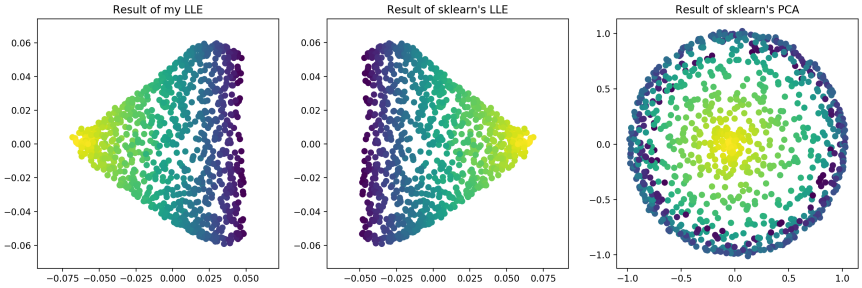

Figure 7: Results of my LLE, `sklearn`'s LLE and `sklearn`'s PCA in bowl shape manifold

Conclusion

From above tests, results and analysis, we can say that LLE will be helpful to explain the nonlinear relationship in high-dimensional space compared with linear dimension deduction method. Besides, my LLE reaches the same performance as `sklearn`'s LLE.

Drawbacks

The result of LLE is sensitive to the selection of k. If k is too small, global properties will not be represented, and if k is too large, the mapping will lose its nonlinear character and behave like traditional PCA [3]. There are two directions for the selection problem. One is to reduce the sensitivity to k, and some methods have raised, such as weighted LLE (WLLE), which uses cam weighted distance to replace the Euclidean distance [5]. However, even changes of k influence the result less, a more suitable k will leads to a better result. An automatic selection algorithm of k may help.

Appendix

Usage

To run the program, all the source file should be put in one file. And the python package `sklearn`, `numpy` and `matplotlib` should be installed in the executing environment.
To start to program, you should type the following command in your terminal.
`python main.py -n <num of the sample> -s <shape of the manifold> -k <num of neighbors to be used>`
As for the shape of manifold, it has three options. `sr` stands for Swiss roll, `s` stands for S shape, `b` stands for bowl shape.
If you need help, please run
`python main.py -h`

main.py

```
'''
* This file is the main file to start the program
** author: Junjie Zhang
'''
import numpy as np
import matplotlib.pyplot as plt
import sys
import getopt
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from data import *
from lle import *


def printHelp():
    print('Usage: python main.py -n <num of the sample> -s <shape of the
    ↪  manifold> -k <num of neighbors to be used>')
    print('<shape of the mainfold>: sr for swiss roll, s for S shape, b for bowl
    ↪  shape')
    return


def main(argv):
    # handle argument
    numSample = 500
    shape = 'sr'
    kNeibors = 30
    try:
        opts, args = getopt.getopt(
            argv, "hn:s:k:", ['help', 'numSample=', 'shape=', 'numNeighbo='])
    except getopt.GetoptError:
        printHelp()
        sys.exit(2)
    for opt, arg in opts:
        if opt in ('-h', '--help'):
            printHelp()
            sys.exit()
        elif opt in ('-n', '--numSample'):
            numSample = int(arg)
        elif opt in ('-s', '--shape'):
            if arg in ('sr', 's', 'b'):
                shape = arg
            else:
                print(
                    '<shape of the mainfold>: sr for swiss roll, s for S shape,
                    ↪  b for bowl shape')
```

```
                sys.exit(2)
        elif opt == 'k':
            kNeibors = int(arg)

    # genereate data
    if (shape == 'sr'):
        X, color = swissRoll(0, 1.5, numSample, noise=0.1)
    elif (shape == 's'):
        X, color = sShape(0, 1, numSample, noise=0.1)
    elif (shape == 'b'):
        X, color = ball(0.3, 1, numSample, noise=0.01)

    # set up solver
    lleSolver = LLE(X, 2, kNeibors)
    lowX1 = lleSolver.transform()

    # solve my sklern
    lowX2 = LocallyLinearEmbedding(
        n_components=2, n_neighbors=kNeibors).fit_transform(X)

    # solve by PCA
    pca = PCA(n_components=2)
    pca.fit(X)
    lowX3 = pca.fit_transform(X)

    X = X.T
    # plot of original data
    fig1 = plt.figure(1)
    ax3D = Axes3D(fig1)
    ax3D.scatter(X[0], X[1], X[2], c=color)

    plt.figure(2, figsize=(8, 5))
    # plot of my solver
    plt.subplot(131)
    plt.title("Result of my LLE")
    plt.scatter(lowX1[:, 0], lowX1[:, 1], c=color)

    # plot of sklearn solver
    plt.subplot(132)
    plt.title("Result of sklearn's LLE")
    plt.scatter(lowX2[:, 0], lowX2[:, 1], c=color)

    # plot of sklearn PCA solver
    plt.subplot(133)
    plt.title("Result of sklearn's PCA")
    plt.scatter(lowX3[:, 0], lowX3[:, 1], c=color)

    plt.show()


if __name__ == '__main__':
    main(sys.argv[1:])
```

data.py

```
'''
* This file is to generate different kind of data
** author: Junjie Zhang
'''
import numpy as np

# * funtion to genereate data with swiss roll shape
# ** parameter:
# *** beginCir: begin of the circle
```

```
# *** endCir: end of the circle
# *** numPoints: num of the data points
# *** a,b, parameters to adjust the radius
# *** noise: the weight on noise


def swissRoll(beginCir, endCir, numPoints, a=1.0, b=1.0, noise=0.0):
    theta = np.random.uniform(
        low=2 * beginCir*np.pi, high=2*endCir*np.pi, size=(1, numPoints))
    radius = a+b*theta
    xs = radius * np.cos(theta)
    ys = radius * np.sin(theta)
    zs = np.random.uniform(low=0, high=5, size=(1, numPoints))
    X = np.concatenate((ys, zs, xs))
    X += noise * np.random.randn(3, numPoints)
    X = X.T
    return X, np.squeeze(theta)

# * genereate data with S shape
# ** parameter:
# *** beginCir: begin of the circle
# *** endCir: end of the circle
# *** numPoints: num of the data points
# *** noise: the weight on noise


def sShape(beginCir, endCir, numPoints, noise=0.0):
    theta = np.random.uniform(
        low=2 * beginCir*np.pi, high=2*endCir*np.pi, size=(1, numPoints))
    xs = np.sin(theta)*theta**0.7
    ys = theta
    zs = np.random.uniform(low=0, high=10, size=(1, numPoints))
    X = np.concatenate((xs, zs, ys))
    X += noise * np.random.randn(3, numPoints)
    X = X.T
    return X, np.squeeze(theta)

# * genereate data with ball shape
# ** parameter:
# *** beginCir: begin of the ball
# *** endCir: end of the ball
# *** numPoints: num of the data points
# *** noise: the weight on noise


def ball(begin, end, numPoints, noise=0.0):
    R = 1
    theta = np.random.uniform(
        low=begin*np.pi, high=end*np.pi, size=(numPoints,))
    fai = np.arange(0, 2, 2/numPoints)*np.pi
    xs = np.array([R*(np.sin(theta)*np.cos(fai))])
    ys = np.array([R*(np.sin(theta)*np.sin(fai))])
    zs = np.array([R*np.cos(theta)])
    X = np.concatenate((xs, ys, zs))
    X += noise * np.random.randn(3, numPoints)
    X = X.T
    return X, theta
```

lle.py

```
'''
* This file is to implement the algorithem of linear locally embedding.
** author: Junjie Zhang
'''
```

```python
import numpy as np


class LLE():

    # * initial function
    # ** parameter:
    # *** kNeighbor: number of neighborhood
    # *** targetDim: target dimension after dimension deduction
    # *** X: matrix of samples , where each row is a sample, has dimension of (numSample, originDim)
    def __init__(self, X, targetDim, kNeighbor=5):
        self.kNeighbor = kNeighbor
        self.targetDim = targetDim
        self.X = X
        self.numSample, self.originDim = X.shape

    # * function to calculate the pairwise distance (Euclidean distance)
    def calPairwiseDist(self):
        squreSumMat = np.sum(np.square(self.X), axis=1)
        self.distMat = np.add(
            np.add(-2 * np.dot(self.X, self.X.T), squreSumMat).T, squreSumMat)
        self.distMat[self.distMat < 0] = 0
        self.distMat = self.distMat**0.5

    # * function to select the neigbors
    def selectNeighbor(self):
        # calculate the distance matrix
        self.calPairwiseDist()
        # the first one is itself, skip it
        self.neighborIndexMat = np.argsort(self.distMat, axis=1)[
            :, 1:self.kNeighbor+1]

    # * function to choose regulation parameter according to number of neighbors and the dimension of the data
    def chooseTol(self):
        if self.kNeighbor > self.originDim:
            self.tol = 1e-3
        else:
            self.tol = 0

    # function to reconstruct with linear weights
    def reconstruct(self):
        w = np.zeros((self.kNeighbor, self.numSample))
        I = np.ones((self.kNeighbor, 1))
        for ii in range(self.numSample):
            Xi = np.tile(self.X[ii], (self.kNeighbor, 1)).T
            Neigbori = self.X[self.neighborIndexMat[ii]].T
            Qi = np.dot((Xi-Neigbori).T, (Xi-Neigbori))
            # in case of kNeighbor > originDim
            Qi = Qi + np.eye(self.kNeighbor)*self.tol*np.trace(Qi)
            QiInv = np.linalg.pinv(Qi)
            wi = np.dot(QiInv, I)/(np.dot(np.dot(I.T, QiInv), I)[0, 0])
            w[:, ii] = wi[:, 0]

        # get W matrix based on w matrix
        self.W = np.zeros((self.numSample, self.numSample))
        for ii in range(self.numSample):
            for jj in range(self.kNeighbor):
                self.W[self.neighborIndexMat[ii][jj], ii] = w[jj, ii]

    # function to map the original data into embedding coordinates
    def embed(self):
        IW = np.eye(self.numSample)
        M = np.dot((IW-self.W), (IW-self.W).T)
```

```
            eigenVal, eigenVec = np.linalg.eig(M)
            vecIndex = np.argsort(eigenVal)[1: self.targetDim + 1]
            self.lowX = eigenVec[:, vecIndex]

        # * function to transform X to low-dimension
        def transform(self):
            # select neighbor
            self.selectNeighbor()

            # choose toleration
            self.chooseTol()

            # Reconstructing with linear weights
            self.reconstruct()

            # Mapping into embedding coordinates
            self.embed()
            return self.lowX
```

References

[1] Christopher M Bishop, Markus Svensén, and Christopher KI Williams. Gtm: The generative topographic mapping. *Neural computation*, 10(1):215–234, 1998.

[2] Miguel A Carreira-Perpinán. A review of dimension reduction techniques. *Department of Computer Science. University of Sheffield. Tech. Rep. CS-96-09*, 9:1–69, 1997.

[3] Dick De Ridder and Robert PW Duin. Locally linear embedding for classification. *Pattern Recognition Group, Dept. of Imaging Science & Technology, Delft University of Technology, Delft, The Netherlands, Tech. Rep. PH-2002-01*, pages 1–12, 2002.

[4] Bogdan Mijovic, Maarten De Vos, Ivan Gligorijevic, Joachim Taelman, and Sabine Van Huffel. Source separation from single-channel recordings by combining empirical-mode decomposition and independent component analysis. *IEEE transactions on biomedical engineering*, 57(9):2188–2196, 2010.

[5] Yaozhang Pan, Shuzhi Sam Ge, and Abdullah Al Mamun. Weighted locally linear embedding for dimension reduction. *Pattern Recognition*, 42(5):798–811, 2009.

[6] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[7] Lawrence K Saul and Sam T Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of machine learning research*, 4(Jun):119–155, 2003.

[8] Teuvo and Kohonen. The self-organizing map. *Neurocomputing*, 1998.

[9] He-Wen Wei, Rong Peng, Qun Wan, Zhang-Xin Chen, and Shang-Fu Ye. Multidimensional scaling analysis for passive moving target localization with tdoa and fdoa measurements. *IEEE Transactions on Signal Processing*, 58(3):1677–1688, 2009.

[10] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.