Comparison and Application of Different Algorithms in Manifold Learning

Dehao Dai

16 March, 2021

Abstract

Before analyzing large data sets in the machine learning, one significant problem is how to simplify these data sets, namely how to discover the problem of dimension reduction. We introduce the notation $Manifold$ to construct a low-dimensional manifold for data embedding into a high-dimensional space. In this paper, we discuss several algorithms for manifold learning and applications to the synthetic data and the real Image problem.

1 Introduction

In the lecture, we have discussed about the Dimensionality Reduction. The problem of reducing the dimension of the data is an important problem in statistical learning. High-dimensional data can be reduced to only a few significant features (or variables) including almost all important information. Dimension reduction usually constructs a mapping from an high-dimensional space to a lower-dimensional one. There are two kinds of dimension reduction methods, (i) Classical (linear) methods and (ii) Manifold (nonlinear) methods.

Classical methods for dimension reduction, like Principal Components Analysis (PCA) and Multidimensional Scaling (MDS), are simple to implement, efficiently computable, and available to discover the true structure of data lying on a linear subspace of the high-dimensional data space. PCA finds a low-dimensional embedding of the data points which aims at preserving the pairwise variance as much as possible. Classical MDS finds an embedding that preserves the pairwise Euclidean distances. Thus, it is equivalent to PCA whose distances are Euclidean. However, many data sets contain essential nonlinear structures that cannot be displayed in the usual space ($\mathbb{R}^2$ and $\mathbb{R}^3$). PCA and MDS usually use the projection method so that the intrinsic structure may be broken. Thus, we need to discover a new kind of method to avoid ignoring the intrinsic structure.

Here we introduce a new kind of methods of dimension reduction - Manifold methods. Manifold methods are used by notation *Manifold*. In mathematics, a manifold is a topological space that is locally homeomorphic to Euclidean space. In other words, it has the local properties of Euclidean space and can be calculated using Euclidean distance. For example, a point (x, y, z) on the sphere can be denoted by

$$x = x_0 + r\sin\theta\cos\psi, \quad y = y_0 + r\sin\theta\sin\psi, \quad z = z_0 + r\cos\theta$$

where $0 \leq \psi \leq 2\pi$ and $0 \leq \theta \leq \pi$. In fact, the point can be expressed by two variables (θ, ψ), so the sphere is 2-d manifold embedded in 3-d space.

In this paper, we introduce 6 different Manifold learning algorithms in the Section Three. In the Section Four, we use the example of faces' images to illustrate the application of Manifold Methods to detect the true degrees of freedom of data set.

2 Algorithms of Manifold Learning

In this section, we introduce several algorithms for manifold learning: (i) Isometric Mapping (Isomap), (ii) Locally Linear Embedding (LLE), (iii) Local Eigenmap (LE), (iv) (t-distributed) Stochastic Neighbor Embedding (SNE and tSNE). And we note that some algorithms require nearest neighborhood parameter k. We usually runs one algorithm over different choices of k and compares the outcome to choose the suitable k. But for the practical problems, we can fix the neighborhood parameter k first.

2.1 Isometric Mapping (Isomap)

Isometric Mapping (Tenenbaum et al. 2000) builds on classical MDS but preserve the intrinsic structure of data. The geodesic distances in Isomap Method are distances along a manifold. It can reflect the true low-dimensional geometry of the manifold, while PCA and MDS just discuss the Euclidean distance and fail to detect the actual intrinsic structure. Then these approximations can be computed efficiently by finding shortest paths in a graph with edges connecting neighboring data points. Thus, the complete isometric mapping, or Isomap algorithm has four steps.

1. Generate a neigborhood graph
   Define the graph G over the data points by connecting points i and j as measured by $\|\mathbf{x}_i - \mathbf{x}_j\|$ if i is one of the K nearest neighbors of j. Set edge lengths equal to $\|\mathbf{x}_i - \mathbf{x}_j\|$.

2. Denote w_{ij} by the weight on the edge between nodes i and j. Define w_{ij} as

$$w_{ij} = \begin{cases} \|\mathbf{x}_i - \mathbf{x}_j\|, & \text{if } i,j \text{ are linked by an edge,} \\ \infty, & \text{otherwise.} \end{cases}$$

3. Compute the shortest distances
   For each value of $k = 1, 2, \cdots, n$, replace all w_{ij} by $\min\{w_{ij}, w_{ik} + w_{kj}\}$. The matrix of final values $D_w = (w_{ij})$ will contain the shortest distances between each pair of nodes. We need $O(n^3)$ operations by Floyd's algorithm or $O(kn^2 \log n)$ by Dijkstra's algorithm.

4. Construct d-dimensional embedding using Theorem 1
   Let λ_p be p-th eigenvalue of the matrix $\tau(D_w) = -HSH/2$, where S is the matrix of squared distances $S_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, and H is the matrix $H_{ij} = \mathbb{1}_{\{i=j\}} - 1/n$ (statement and proof from *Algorithms for manifold learning*) . Let $v_p^{(i)}$ be the i-th component of the p-th eigenvector. Then set the p-th component of the d-dimensional coordinate vector y_i equal to $\sqrt{\lambda_p} v_p^{(i)}$.

2.2 Locally Linear Embedding (LLE)

Locally Linear Embedding (LLE) (Roweis and Saul 2000) is similar to Isomap. Suppose the data consist of N real-valued vectors $\mathbf{x}_i \in \mathbb{R}^d$. If the manifold is well-sampled, each data point and its neighbors are expected to be close to a locally linear part (bounded and approximated as a plane) of the manifold. The LLE algorithm has steps:

1. Build a neighborhood graph by using the k nearest neighbors of each data point $\mathbf{x}_i$
2. Compute the weights $W = (w_{ij})$ that best reconstruct $\mathbf{x}_i$ from its neighbors, solving

$$\arg\min_W \mathcal{E}(W) = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \|\mathbf{x}_i - \sum_j w_{ij}\mathbf{x}_j\|^2, \text{with} \sum_j w_{ij} = 1$$

   with $w_{ij} = 0$ if $\mathbf{x}_j$ does not belong to k nearest neighborhood of $\mathbf{x}_i$. The solution of $\arg\min_W \mathcal{E}(W)$ is $\mathbf{w} = (\mathbf{1}'C^{-1}\mathbf{1})^{-1}C^{-1}\mathbf{1}$ with $C = (c_{jk})$ and $c_{jk} = (\mathbf{x} - \mathbf{x}_j)'(\mathbf{x} - \mathbf{x}_k)$.
3. Compute the vectors $\mathbf{Y} \in \mathbb{R}^{n\times q}$ best reconstructed by the weights, minimizing:

$$\Phi(\mathbf{Y}) = \sum_{i=1}^{n} \|\mathbf{y}_i - \sum_j w_{ij}\mathbf{y}_j\|^2$$

   by finding the smallest eigenmodes of the sparse symmetric matrix $\mathbf{M}$ in $m_{ij} = \mathbb{1}_{\{i=j\}} - w_{ij} - w_{ji} + \sum_k w_{ik}w_{kj}$

2.3 Laplacian Eigenmap (LE)

Laplacian Eigenmap (Belkin and Niyogi 2003) assumes that the images of close points in the high-dimensional space are also close. And Belkin used the Laplacian Operator to transform the original problem to general eigenvalue problem. Thus, LE has four main steps:

1. Constructing the adjacency graph with n nodes
2. Compute the edge weights $\mathbf{W} = (w_{ij})$. There are two kinds of form to weight the edges.
   - Heat kernel (parameter $t \in \mathbb{R}$):

$$w_{ij} = \begin{cases} \exp\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\}, & \text{if } i,j \text{ are linked by an edge,} \\ 0, & \text{otherwise.} \end{cases}$$

   - Flat kernel: $w_{ij} = \mathbb{1}\{i, j \text{ are linked by an edge}\}$
3. Compute the bottom generalized eigenvectors: $L\mathbf{y} = \lambda D\mathbf{y}$, where $D = (d_{ii})$ with $d_{ii} = \sum_j w_{ij}$ and $L = D - W$ is Laplacian matrix and m-order positive semi-definite.
4. Define the embedding vectors: $\mathbf{x}_i \to (y_{i1}, y_{i2}, \cdots y_{im})$

2.4 Stochastic Neighbor Embedding

All of these previous methods require each high-dimensional observation to be limited to only one single linear (or non-linear) location in the low-dimensional space. For "many-to-one" mappings, it may be difficult to determine an accurate locations of the original data in the low-dimensional space. There are two kinds of similar algorithms to solve this problem based on Probability. Our algorithm, Stochastic Neighbor Embedding (Hinton and Roweis 2002) tries to place the objects in a low-dimensional space to optimally preserve neighborhood information. So it is appropriate to apply these algorithms into processing image-related data.

2.4.1 The basic SNE algorithm

Suppose we have n observation $\mathbf{x}_1, \cdots, \mathbf{x}_n$. For each data point $\mathbf{x}_i$, it has k potential neighbors. Now we compute the probability p_{ij} that data point $\mathbf{x}_i$ chooses data point $\mathbf{x}_j$ as its neighbor:

$$p_{j|i} = \frac{\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_k\|^2}{2\sigma_i^2}\right)}$$

by the scaled squared Euclidean distance between two high-dimensional points, $\mathbf{x}_i, \mathbf{x}_j$. In the low-dimensional space we also use *Gaussian* neighborhoods but with a fixed variance so the probability q_{ij} that data point $\mathbf{x}_i$ chooses data point $\mathbf{x}_j$ as its neighbor is a function of the low-dimensional images $\mathbf{y}_i$ of the original data points:

$$q_{j|i} = \frac{\exp\left(-\|\mathbf{y}_i - \mathbf{y}_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|\mathbf{y}_i - \mathbf{y}_k\|^2\right)}$$

To minimize a cost function or solve the problem:

$$\arg\min_{\mathbf{y}} C(\mathbf{y}) = \arg\min_{\mathbf{y}} \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

It can be differentiated by $\mathbf{y}$: $\frac{\partial C}{\partial \mathbf{y}_i} = 2\sum_j (\mathbf{y}_i - \mathbf{y}_j)(p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i})$.

2.4.2 The t-distributed SNE algorithm

In the basic SNE algorithm, there are two kinds of problems: (i) the cost function (i.e. Kullback-Leibler divergence) is not symmetric so that SNE algorithm may focus on the local structure rather than the whole one; (ii) different clusters may be too crowded to be separated easily. Therefore, Maaten modified the basic SNE algorithm (Van der Maaten and Hinton 2008).

First, Maaten replaced the conditional distribution $p_{j|i}, q_{j|i}$ by joint distribution p_{ij}, q_{ij}. However, the data point $\mathbf{x}_i$ may be so far away from the cluster that $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ is great and p_{ij} is small. Thus, the data point is not significant for the cost function. Maaten introduced the symmetric conditional distribution. For n data points, we have joint distibution $p_{ij} = \frac{1}{2n}(p_{j|i} + p_{i|j})$. Hence, the cost function is updated as

$$C(\mathbf{y}) = \arg\min_{\mathbf{y}} \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

and it also can be differentiated by $\mathbf{y}$: $\frac{\partial C}{\partial \mathbf{y}_i} = 4\sum_j (\mathbf{y}_i - \mathbf{y}_j)(p_{ij} - q_{ij})$.

Second, since t-distribution is long-tailed, the algorithm use t-distribution to redefine q_{ij}

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

and

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_j \left(p_{ij} - q_{ij}\right)\left(y_i - y_j\right)\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

3 Application to the Data

3.1 Application to the synthetic data

3.1.1 Preparation

First, the data is generated from multivariate normal distribution, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ following the same covariance $\Sigma = \begin{pmatrix} 20 & 2 & 0.5 \\ 2 & 10 & 1.5 \\ 0.5 & 1.5 & 5 \end{pmatrix}$. and means $\begin{pmatrix} 10 \\ 0 \\ 20 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 5 \\ 10 \end{pmatrix}$, $\begin{pmatrix} 10 \\ 5 \\ 0 \end{pmatrix}$ respectively. And their responses are 1, 2, 3 respectively.

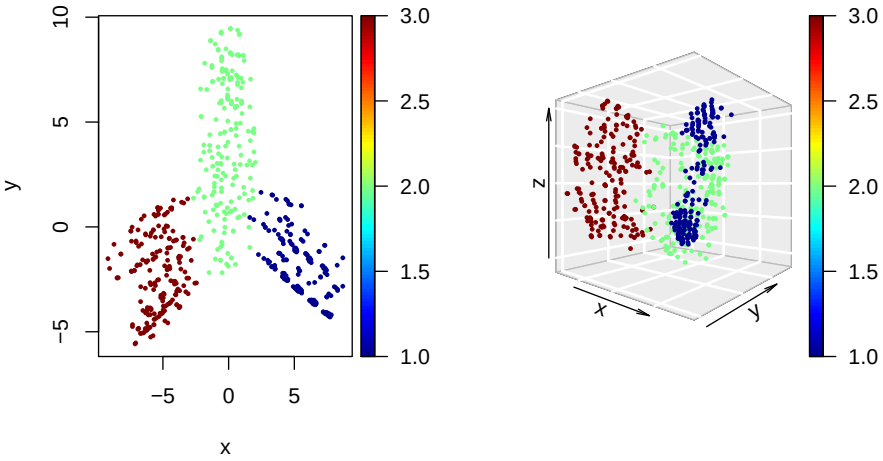
```
library(MASS)
n = 200
set.seed(1)
x1 = mvrnorm(n, c(10, 0, 20), matrix(c(20, 2, 0.5, 2, 10, 1.5, 0.5, 1.5, 5), 3, 3))
set.seed(2)
x2 = mvrnorm(n, c(0, 5, 10), matrix(c(20, 2, 0.5, 2, 10, 1.5, 0.5, 1.5, 5), 3, 3))
set.seed(3)
x3 = mvrnorm(n, c(10, 5, 0), matrix(c(20, 2, 0.5, 2, 10, 1.5, 0.5, 1.5, 5), 3, 3))

y = c(rep(1, n), rep(2, n), rep(3, n))
data = cbind(y, rbind(x1,x2,x3))
```

3.1.2 Isomap

Now, we use `do.isomap()` to reduce the dimension to 3 (though the data is 3-d) by setting the neighborhood $k = 5$. Then we observe the graph later.

```
library(Rdimtools)
library(plot3D)
result = do.isomap(data, ndim = 3, type = c("knn", 5))
result = result$Y
par(mfrow = c(1,2))
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = y)
scatter3D(result[,1], result[,2], result[,3],
          pch = 16, cex = 0.5, colvar = y, phi = 0, bty ="g",
          thicktype = "detailed")
```

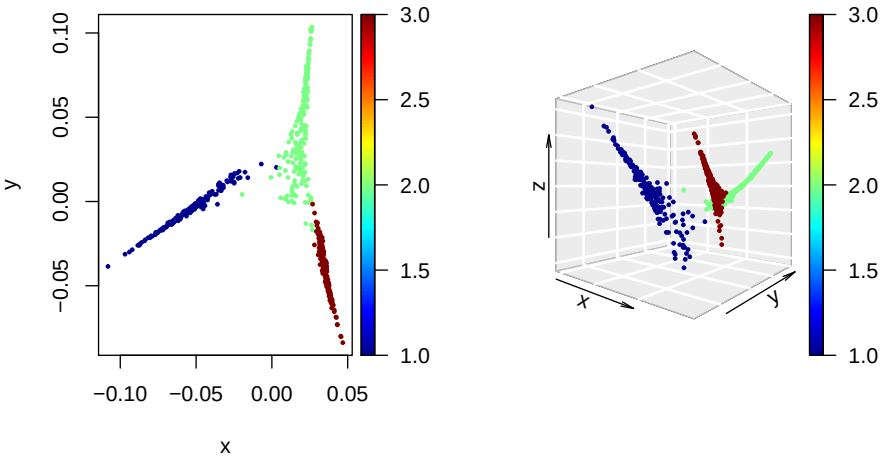


From the 3-d graph and 2-d graph, we find that the classification of the data is clear because of the multivariate normal distribution.

3.1.3 LLE

Now, we use `do.lle()` to reduce the dimension to 3 (though the data is 3-d) by setting the neighborhood $k = 5$ and observe the graph later.

```
result = do.lle(data, ndim = 3, type = c("knn", 5))
result = result$Y
par(mfrow = c(1,2))
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = y)
scatter3D(result[,1], result[,2], result[,3], pch = 16, cex = 0.5, colvar = y,
          phi = 0, bty ="g", thicktype = "detailed")
```

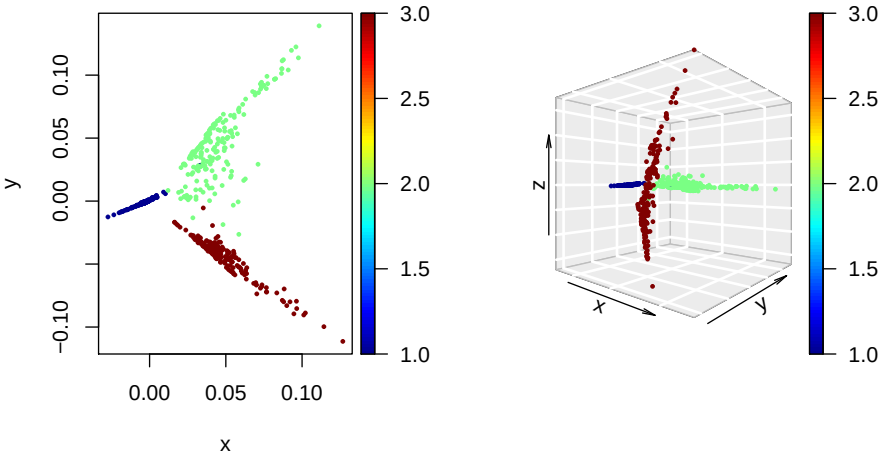


From the 3-d graph and 2-d graph, we find that the classification of the data is still clear because of the multivariate normal distribution. And we find the three planes are almost parallel in the 3-d graph since the LLE algorithm use the localized linear embedding.

3.1.4 LE algorithm

Now, we use `do.lapeig()` to reduce the dimension to 3 (though the data is 3-d) by setting the neighborhood $k = 5$ and observe the graph later.

```
result = do.lapeig(data, ndim = 3, type = c("knn", 5))
result = result$Y
par(mfrow = c(1,2))
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = y)
scatter3D(result[,1], result[,2], result[,3], pch = 16, cex = 0.5, colvar = y,
          phi = 0, bty ="g", thicktype = "detailed")
```



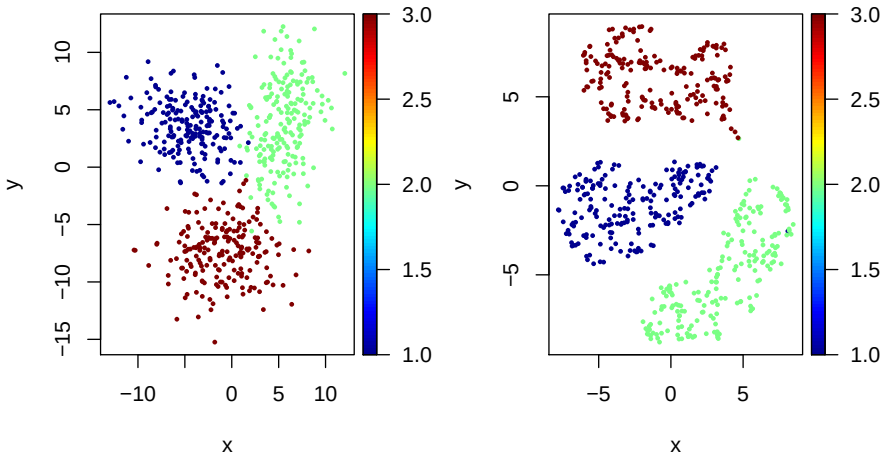
It is similar to the LLE algorithm. But the classification of the data is less clear than the previous algorithm since the blue points and the green points are in the same plane from the 3-d graph. And we find that some green points are in the brown and blue region which may affect the actual classification.

3.1.5 SNE algorithm

We use two SNE algorithm to make the experiment. Note that both the two algorithm only choose the dimension d reduced is smaller than the original one m. We use the `do.sne()` and `do.tsne()` to reduce the dimension.

```
# Classical One
par(mfrow = c(1,2))
result = do.sne(data, ndim = 2)
result = result$Y
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = y)

# Modified one based on t-distribution
result = do.tsne(data, ndim = 2)
result = result$Y
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = y)
```



The results are similar but the tsne can solve the problem of crowding. And the classification is more clear than the previous algorithm.

3.2 Applications to the Image Data

3.2.1 Preparation

The dataset has been processed from https://cs.nyu.edu/~roweis/data/frey_rawface.mat by Matlab. They are 1965 images of Brendan's face, taken from sequential frames of a small video and their sizes are all 20 pixel by 28 pixel. Within the 560-dimensional input space, all of the images lie on an intrinsically three-dimensional manifold. Our goal is to discover, given the high-dimensional inputs, low-dimensional representations that capture the intrinsic degrees of data. Since the entries in this matrix are greyscale of image, we use function `showMatrix(matrix)` to display these images by `image()`.

```
library(RColorBrewer)
showMatrix = function(x,...){
image(x[,ncol(x):1], xaxt = 'none', yaxt = 'none',
      col = rev(colorRampPalette(brewer.pal(9,'Greys'))(100)),...)
}
faces = as.matrix(read.table("~/Desktop/faces.txt"))
# display the 111-th to 120-th images
par(mfrow=c(10,10),mar=rep(0,4))
for (i in 111:120){
  showMatrix(matrix(faces[,i],20,28))
}
```

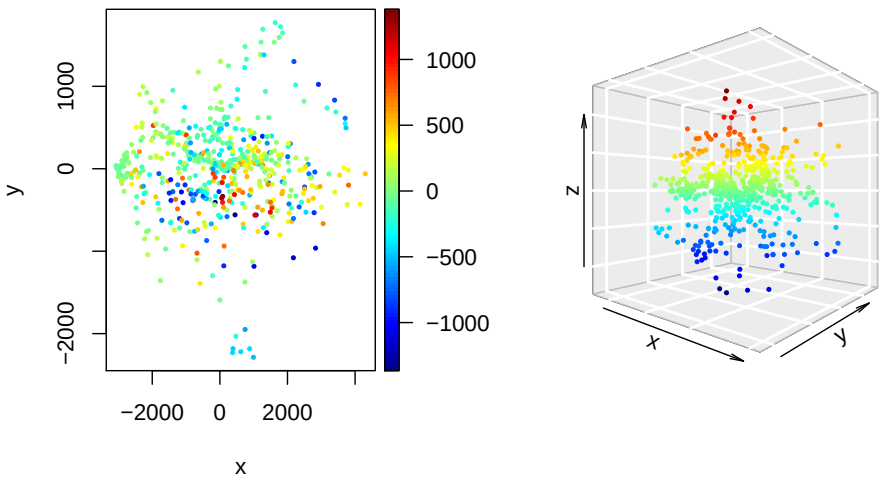
From the graph, we find that the 111-th to 120-th images sampled from the whole data are similar. The pixels in every column (all images) are 154 on average. In fact, different angles can make different shadows or face expressions. So, we expect the angle or degree of freedom as the components. Now we introduce package `Rdimtools` and `umap` to analysis image-related data. `Rdimtools` has several dimension reduction methods like Isomap, LLE, Laplacian Eigenmap, SNE and so on. And

`umap` is a particular package for algorithm UMAP. We observe the result from both 2-d and 3-d to determine the possible manifold the algorithms provide. And we judge whether the manifold help us analysis the image-related data.

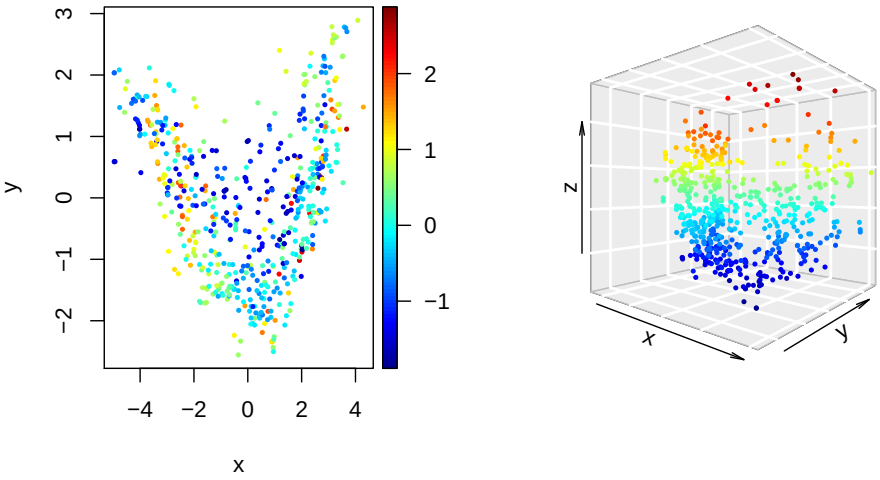
3.2.2 Isomap algorithm

First, we use `do.pca(faces, ndim = 3)` to reduce the dimension of the data set and have a direct observation on the result. We can extract the first two dimension data as 2-d graphs and the three dimension data as 3-d graph.

```
library(Rdimtools)
library(plot3D)
par(mfrow = c(1,2))
# Using PCA to reduce the dimension of the data set
faces_pca = do.pca(faces, ndim = 3)
faces_pca = faces_pca$Y
scatter2D(faces_pca[,1], faces_pca[,2], pch = 16, cex = 0.5,
          colvar = faces_pca[,3])
scatter3D(faces_pca[,1], faces_pca[,2], faces_pca[,3],
          pch = 16, cex = 0.5, colkey = FALSE, phi = 0, bty ="g",
          thicktype = "detailed")
```




```
# Using Isomap to reduce the dimension of the data set
result = do.isomap(faces, ndim = 3, type = c("knn", 11))
result = result$Y
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5,
          colvar = result[,3])
scatter3D(result[,1], result[,2], result[,3],
          pch = 16, cex = 0.5, colkey = FALSE, phi = 0, bty ="g",
          thicktype = "detailed")
```

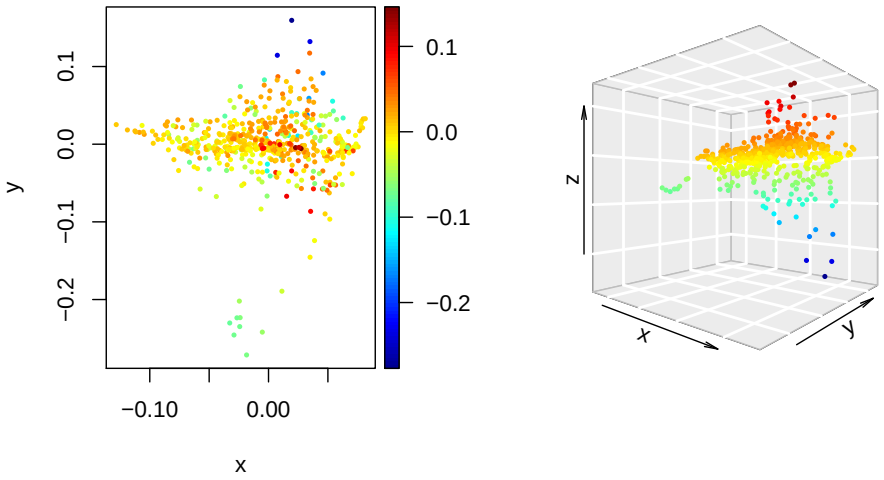


Now we use `do.isomap(faces, ndim = 3, type = c("knn", 11))` to compute the $k = 11$ nearest neighborhood distance to reduce the dimensionality by Isomap. And we plot the 2-d and 3-d graphs from the result. It is obvious that the result provided by PCA is so unclear that we may not interpret what is the two or three components. In fact, the pictures of faces are at different angles. So we expect the two components should be directions. However, from the graphs provide by Isomap, the manifold is like a "V"-type curve. So Isomap's global coordinates provide a simpler way to analyze and interpret high-dimensional observation based on their intrinsic nonlinear degrees. Just like a set of face images, known to have at least two degrees of freedom, Isomap can detect the dimension and separate to true potential components.

3.2.3 LLE algorithm

First, we use `do.lle(faces, ndim = 3)` to reduce the dimension of the data set by setting the nearest neighborhood $k = 11$ and have a direct observation on the result. We can extract the first two dimension data as 2-d graphs and the three dimension data as 3-d graph.

```
result = do.lle(faces, ndim = 3, type = c("knn", 11))
result = result$Y
par(mfrow = c(1,2))
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = result[,3])
scatter3D(result[,1], result[,2], result[,3],
          pch = 16, cex = 0.5, colkey = FALSE, phi = 0, bty ="g",
          thicktype = "detailed")
```

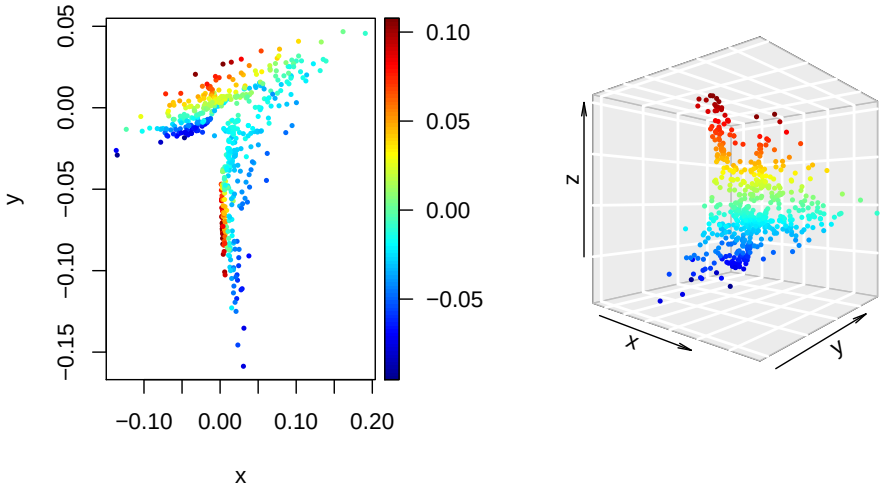


From the 2-d and 3-d graphs, the LLE provide less clear classification than the Isomap. In fact the local features may not represent the whole manifold. Thus, the 3-d graph implies that the manifold is like some plane, which may cause unclear classification. Just like the face images, the local feature in this high-dimensional data may be not clear so that this algorithm is less good than the previous one.

3.2.4 LE algorithm

First, we use `do.lapeig(faces, ndim = 3)` to reduce the dimension of the data set by setting the nearest neighborhood $k = 11$. And we also can scatterplot the first two dimension data and three dimension data.

```
result = do.lapeig(faces, ndim = 3, type = c("knn", 11))
result = result$Y
par(mfrow = c(1,2))
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, colvar = result[,3])
scatter3D(result[,1], result[,2], result[,3],
          pch = 16, cex = 0.5, colkey = FALSE, phi = 0, bty ="g",
          thicktype = "detailed")
```

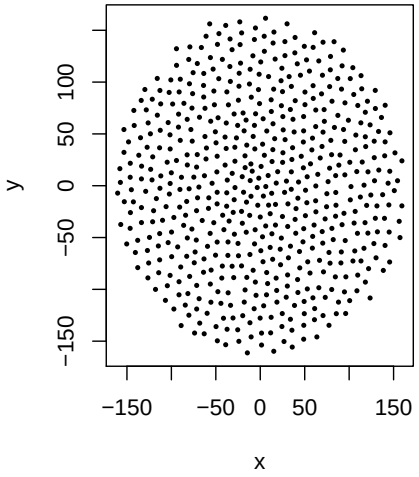


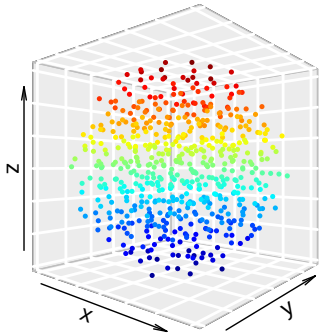
The result is clearer than previous algorithm. We can find that the manifold constructed by this

algorithm is crescent and two components can be regarded as the angles. It is similar to the result provided by the Isomap since they are similar algorithms.

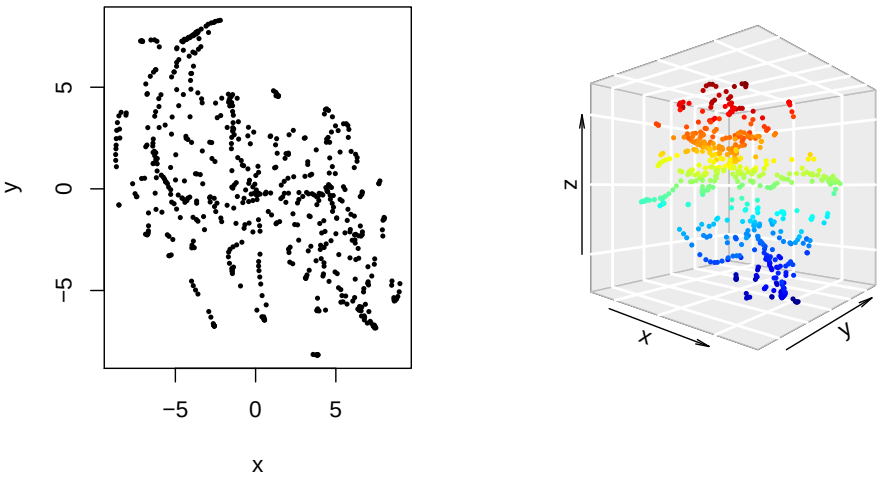
3.2.5 SNE algorithm

```
# Classical One
par(mfrow = c(1,2))
result = do.sne(faces, ndim = 2)
result = result$Y
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5,col = 1)
result = do.sne(faces, ndim = 3)
result = result$Y
scatter3D(result[,1], result[,2], result[,3],
          pch = 16, cex = 0.5, colkey = FALSE, phi = 0, bty ="g",
          thicktype = "detailed")
```






```
# Modified one based on t-distribution
result = do.tsne(faces, ndim = 2)
result = result$Y
scatter2D(result[,1], result[,2], pch = 16, cex = 0.5, col = 1)
result = do.tsne(faces, ndim = 3)
result = result$Y
scatter3D(result[,1], result[,2], result[,3],
          pch = 16, cex = 0.5, colkey = FALSE, phi = 0, bty ="g",
          thicktype = "detailed")
```



The result is clearer than previous algorithm, but the time of applying function is far longer than previous all algorithms. And the result by t-sne algorithm could avoid the problem of crowding. It is same as the result of the synthetic data.

References

Belkin, M., and Niyogi, P. (2003), "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, MIT Press, 15, 1373–1396.

Hinton, G., and Roweis, S. T. (2002), "Stochastic neighbor embedding," in *NIPS*, Citeseer, pp. 833–840.

Roweis, S., and Saul, L. (2000), "Nonlinear dimensionality reduction by locally linear embedding," *Science*, 290, 2323–2326.

Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000), "A global geometric framework for nonlinear dimensionality reduction," *Science*, American Association for the Advancement of Science, 290, 2319–2323.

Van der Maaten, L., and Hinton, G. (2008), "Visualizing data using t-sne." *Journal of machine learning research*, 9.