In The Name of God

The University of Isfahan
Mathematics and Computer Science Department

Bachelor of Science Thesis
Computer Science

Subject:
Dimension Reduction Algorithms Analysis

Student:
Marzieh Ayat

Supervisor:
Dr. Davoud Mirzaei

Summer 2022

1 Acknowledgments

With all respect, I express my gratitude and appreciation to all the respected professors of the Computer Science faculty of the University of Isfahan. I bow down as a disciple to these nobles who were always my guide and teacher during these four years.

Contents

List of Figures

List of Tables

2 Abstract

Dimension reduction(DR) techniques are used to either illustrate high-dimensional data or compress data. Choosing one of these techniques is not easy and depends on many variables. Evaluating these techniques is also hard because the model affects the evaluation. Choosing one technique based on their behavior as being linear or not can help, but linear or non-linear behavior in high dimensions is not always detectable. Different methods are used on a data set to recognize the best DR technique, such as running different models and evaluating the data after fitting in the DR method. This can help us judge better.

3 Overdetermined System and Least Square Problem

definition 1. *In case the number of* ***equation*** *is greater than the* ***variables****, the system is said to be* ***overdetermined****.[1]*

This system is **inconsistent**, in other words, no x satisfies this system.[1]

example 1. *To illustrate this assume two figures 1 and 2,*

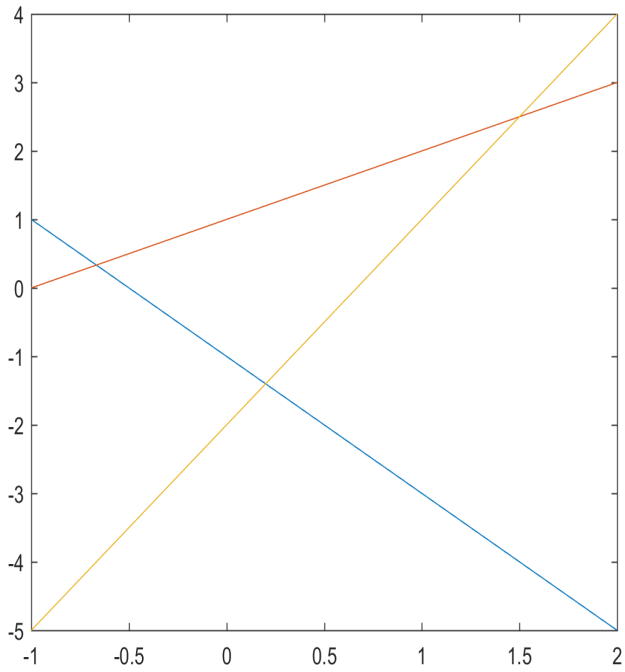

Figure 1: 3 equation 2 variable linearly independent

Figure1 shows 3 linearly independent equations with only two variables, while in figure2 we can multiply $2x+3$ *by -6 and multiply* $x+1$ *by 4, and then we see the last equation is the summation of this two equations.*

definition 2. *So instead of solving* $\mathbf{Ax} = \mathbf{b}$*,* $\min_{\mathbf{x}} \|\mathbf{Ax}-\mathbf{b}\|$ *can be solved to find an approximation answer for an overdetermined system. This approximation is called the* ***Least Square Approximation(LSA)****.*

[1] All equations should be **linearly dependent** if they were not linearly dependent, the system may have solutions.

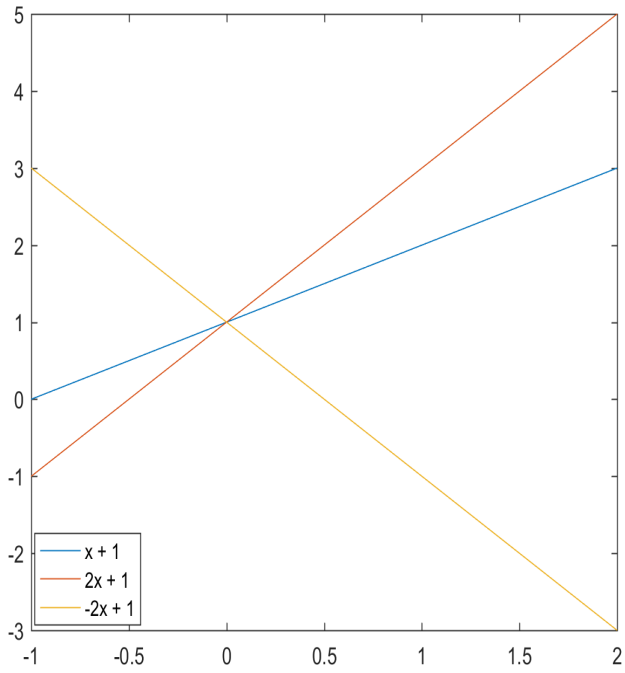

Figure 2: 3 equation 2 variable linearly independent

There are multiple ways to solve LSA. The simplest one(probably not the efficient one) is using **normal equations**, or $\mathbf{x} = \left(A^{\mathsf{T}}A\right)^{-1} A^{\mathsf{T}}\mathbf{b}$,.
Another way to solve least squares is by using decompositions like QR and SVD, which is preferred.[2]

4 QR Decomposition

Assume $A \in \mathbb{C}^{m\times n}$ with $m \geq n$, the QR decomposition is as follow:

$$A = \hat{Q}\hat{R}$$

the $\hat{Q} \in \mathbb{C}^{m\times n}$ is an orthogonal matrix and $\hat{R} \in \mathbb{C}^{n\times n}$ is an upper triangular. There are several ways for doing the decomposition the most popular algorithms are Givens Rotation, Gram-Schmidt, and Householder. Gram-Schmidt and Householder for QR will be described.

4.1 Gram-Schmidt

$$a_1 = q_1 r_{11} \rightarrow q_1 = \frac{a_1}{r_{11}}$$

$$a_2 = q_1 r_{12} + q_2 r_{22} \rightarrow q_1 = \frac{a_2 - r_{12}q_1}{r_{22}}$$

$$\vdots \quad \vdots$$

$$a_2 = q_1 r_{1n} + q_2 r_{2n} + q_n r_{nn} \cdots + \rightarrow q_1 = \frac{a_2 - \sum_{i=1}^{n} r_{in}q_i}{r_{nn}}$$

$$q_i^* q_j = \delta_{ij}$$

The R matrix is reachable with Q and A matrix. For example r_{11} and r_{22} are as follow:

$$q_1^* q_1 = \frac{a_1^* a_1}{r_{11}^2} = 1$$

$$r_{11} = \sqrt{a_1^* a_1} = \|a_1\|_2$$

$$q_1^* q_2 = \frac{q_1^* a_2 - r_{12} q_1^* q_1}{r_{22}} = 0$$

$$r_{22} = \|a_2 - (a_2 q_1^*)q_1\|_2$$

The pseudo-code of classical Gram-Schmidt for QR decomposition is in Algorithm 1.
While this algorithm is unstable so we use a modified version to solve this problem. Algorithm 2 showed this modified version This algorithm is unstable so we use a modified version to solve this problem.

Algorithm 1 Classical Gram-Schmidt

for j = 1: n **do**
  $v_j = a_j$
  for i=1 : $j - 1$ **do**
    $r_{ij} = q_i a_j$
    $v_j = v_j - r_{ij} q_i$
  end for
  $r_{jj} = \|vj\|2$
  $q_j = v_j / r_{jj}$
end for

Algorithm 2 Modified Gram-Schmidt

for i = 1:n **do**
  $v_i = a_i$
end for
for i =1 : n **do**
  $r_{ii} = \|v_i\|2$
  $q_i = \frac{v_i}{r_{ii}}$
  for $j = (i + 1)$:n **do**
    $r_{ij} = q_i v_j$
    $v_j = v_j - r_{ij} q_i$
  end for
end for

4.2 Householder

As an informal intuition, we do as follows to implement the Householder technique:

$$A = \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \longrightarrow Q_1 A = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix}$$

$$Q_2 Q_1 A = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \\ 0 & 0 & x \end{bmatrix} \longrightarrow Q_3 Q_2 Q_1 A = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \\ 0 & 0 & 0 \end{bmatrix},$$

x is any non-zero entry. It is obvious that Q_k should apply on the last $m-k$ rows and the first rows and columns of Q_{k-1} must not get modified. Givens rotations or Householder reflectors can be used in order to define Q and R.

$$Q_k = \begin{bmatrix} I_{K-1} & O \\ O & F \end{bmatrix}$$

where I_{k-1} is an $(k-1) \times (k-1)$ identity matrix and F is:

$$F\boldsymbol{x} = \|x\| e_1$$

F is a householder reflector. By choosing $v = \|x\| e_1 x$ and $P = \frac{\boldsymbol{v}\boldsymbol{v}^*}{\boldsymbol{v}^*\boldsymbol{v}}$ we will have

$$F = I - 2P = I - 2\frac{\boldsymbol{v}\boldsymbol{v}^*}{\boldsymbol{v}^*\boldsymbol{v}} \Rightarrow F\boldsymbol{x} = \left(I - 2\frac{\boldsymbol{v}\boldsymbol{v}^*}{\boldsymbol{v}^*\boldsymbol{v}}\right)\boldsymbol{x} = \boldsymbol{x} - 2\underbrace{\frac{\boldsymbol{v}\boldsymbol{v}^*}{\boldsymbol{v}^*\boldsymbol{v}}}_{\text{matrix}}\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{v}\underbrace{\frac{\boldsymbol{v}^*\boldsymbol{x}}{\boldsymbol{v}^*\boldsymbol{v}}}_{\text{scalar}}$$

So to conclude, Algorithm 3 is the pseudo-code we discussed above. 3 Algorithm

Algorithm 3 Householder QR

for $k = 1 : n$ **do**
  $\boldsymbol{x} = A(k : m, k)$
  $\boldsymbol{v}_k = \boldsymbol{x} + \text{sign}(\boldsymbol{x}(1))\|\boldsymbol{x}\|_2 \boldsymbol{e}_1$
  $\boldsymbol{v}_k = \boldsymbol{v}_k / \|\boldsymbol{v}_k\|_2$
  $A(k : m, k : n) = A(k : m, k : n) - 2\boldsymbol{v}_k (\boldsymbol{v}_k^* A(k : m, k : n))$
end for

has the complexity of $\mathcal{O}(2mn^2 - \frac{2}{3}n^3)$ To make it clear assume the next example.

example 2. *We want to do a Householder reflector on the* $x = [2, 1, 2]^T$ *and then compute* v

$$\boldsymbol{v} = \boldsymbol{x} + \text{sign}(\boldsymbol{x}(1))\|\boldsymbol{x}\|_2 \boldsymbol{e}_1$$

$$= \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} + 3 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$$

$F\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{v}\frac{\boldsymbol{v}^*\boldsymbol{x}}{\boldsymbol{v}^*\boldsymbol{v}}$ *we know* $\boldsymbol{v}^*\boldsymbol{x} = 15$ *and* $\boldsymbol{v}^*\boldsymbol{v} = 30$
then

$$F\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{v}\frac{\boldsymbol{v}^*\boldsymbol{x}}{\boldsymbol{v}^*\boldsymbol{v}}.\boldsymbol{v}^*\boldsymbol{x} \Rightarrow \; F\boldsymbol{x} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} 23 \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix} \frac{15}{30} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$$

The Third When we confront too many features and variables in a data set, we are encouraged to reduce the size with the least loss of information and quality of the data set. These features may be highly correlated[2]. Further, the way of reducing this size will be illustrated. In this chapter, some dimension reduction algorithms will be discussed. The reason we use each and their application is important to us so we apply them to some data sets to see their accuracy.

Dimensionality reduction algorithms divide into two categories:

1. Linear Algebra Methods
   (a) Singular Value Decomposition
   (b) Principal Components Analysis
2. Manifold Learning Methods
   (a) Isomap Embedding
   (b) Locally Linear Embedding
   (c) Kernel PCA

5 Linear Algebra Methods

5.1 PCA

Remember the previous scenario, we want to reduce the size of a matrix or data set. It can be done either with choosing few components of the features, or we can combine several features to build a new feature. Principal component analysis is based on the second way. This approach has less intuitive appeal than choosing a subset, but has the advantage that, for the same amount of information loss, we can achieve a greater reduction in dimensionality. Principal components analysis (PCA) finds low dimensional approximations to the data by projecting the data onto linear subspaces.

Algorithm 4 Principal Components Analysis (PCA)

Compute $\tilde{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X}_n)(X_i - \bar{X}_n)^T$
Compute eigenvalues and eigenvectors of $\tilde{\Sigma}$
Choose a dimension k
$T_k(X) = \mu + \sum_{j=1}^{k}\beta_j e_j \ , \ \beta_j e_j = \langle X - \mu, e_j \rangle$

[2]In pre-processing, some of the features which are not correlated to the target will be omitted, so it is expected that in a cleaned data set, the features be highly correlated. However, in this chapter, this technique is discussed mathematically.

The implementation of PCA on a data set is as follows. Consider A as a matrix:

$$A = \begin{bmatrix} 0 & 0 \\ 0.999 & 0.9 \\ 0.12223 & 0.312 \\ 0.175674 & 0.1499 \\ 0.245243 & 0.7499 \\ 0.265234 & 0.3599 \\ 0.3 & 0.23239 \\ 0.354373 & 0.32299 \\ 0.4423523 & 0.5 \\ 0.483 & 0.1423999 \\ 0.5 & 0.12 \\ 0.583 & 0.4999 \\ 0.6542323 & 0.1239 \\ 0.6542343 & 0.2399 \\ 0.65523 & 0.643499 \\ 0.6425473 & 0.52342299 \\ 0.683 & 0.654999 \\ 0.7 & 0.72 \\ 0.89 & 0.8899 \\ 0.76 & 0.69 \\ 0 & 0.345 \end{bmatrix}$$

Then we standardize the matrix and compute the covariance of A

$$\begin{matrix} 0.95238095 & 0.59533803 \\ 0.59533803 & 0.95238095 \end{matrix}$$

The eigenvalues of this matrix is 0.37489506,1.62510494 and eigenvectors are correspondingly [-0.70710678,-0.70710678] [0.70710678,-0.70710678]. After applying the last step we have the 4.

5.2 Connection between PCA and SVD

There is a strong connection between PCA and singular value decomposition (SVD). Let X be an $n \times d$ matrix. The SVD is

$$X = UDV^T$$

where U is an $n \times n$ matrix with orthonormal columns, V is a $d \times d$ matrix with orthonormal columns, and D is an$n \times d$ diagonal matrix with non-negative real numbers on the diagonal (called singular values). Then

$$X^T X = (VDU^T)(UDV^T) = VD^2V^T$$

and hence the singular values are the square root of the eigenvalues.

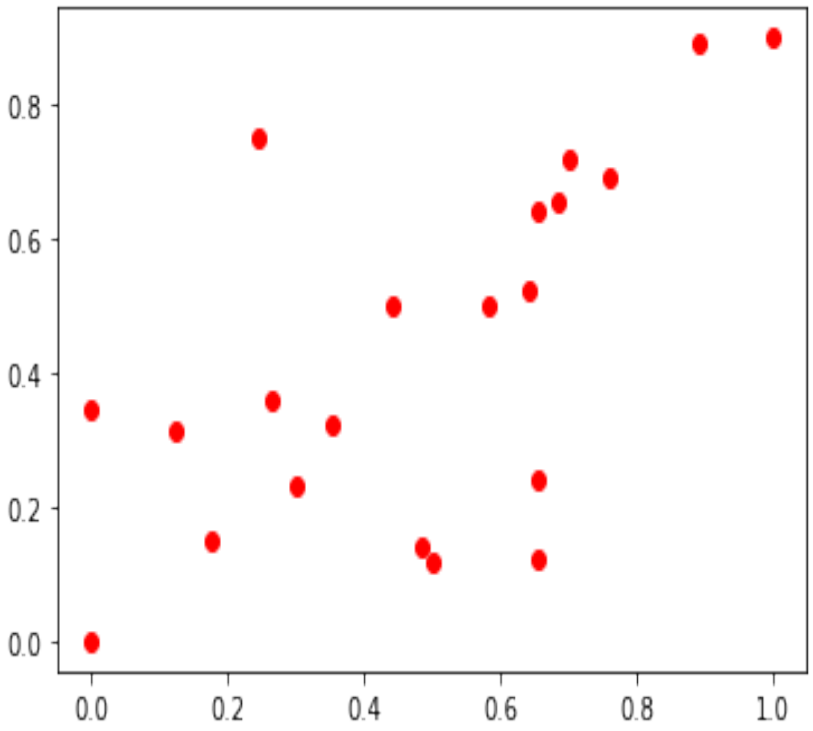

Figure 3: Data (before applying PCA).

6 Non-Linear Methods

6.1 Kernel PCA

To get a nonlinear version of PCA, we can use a kernel. We have $x \to \phi(x)$ as a "feature map" and want to transform PCA in this new feature space.

Just as for standard PCA, this selects components of high variance, but in the feature space of the kernel. In addition, the "feature functions"

$$\langle\, v_m, \Phi\,(x)\rangle = \sum_{j=1}^{n} \alpha_i^m K\Big(X_i, x\Big)$$

$$\langle\, v_m, \Phi\,(x)\rangle = \sum_{j=1}^{n} \alpha_i^m K\Big(X_i, x\Big)$$

$$f_m(x) = \sum_{j=1}^{n} \alpha_i^m K\Big(X_i, x\Big)$$

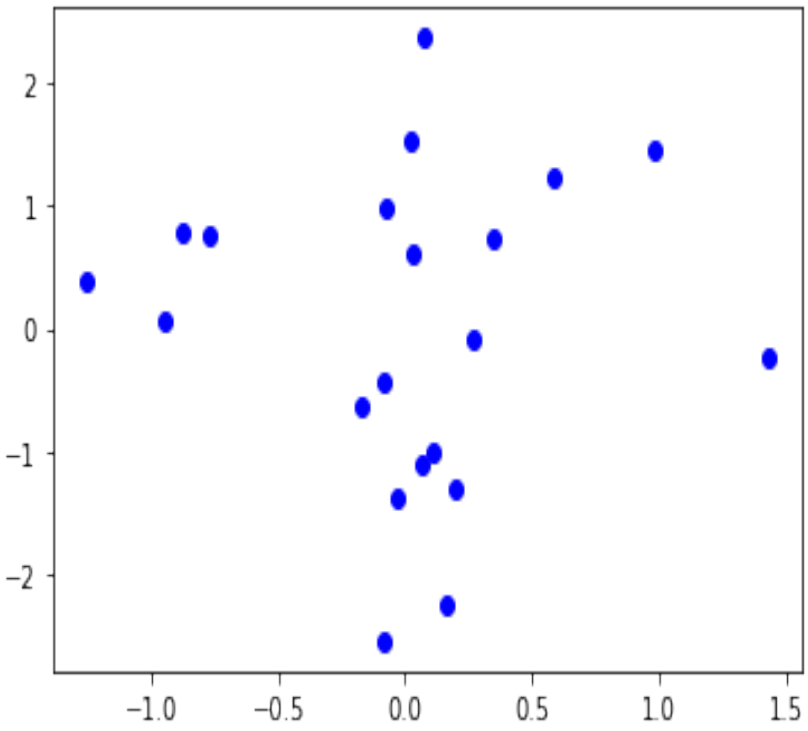

Figure 4: data (after applying PCA)

are orthogonal and act as representative feature functions in the reproducing kernel Hilbert space of the kernel, with respect to the given data.

Algorithm 5 kernel PCA

Center the kernel.
Compute K_{ij}.
Diagonalize K
Normalize eigenvectors $\alpha(m)$ so that $\langle \alpha(m); \alpha(m) \rangle = \frac{1}{\lambda_m}$
Compute the projection of a test point x onto an eigenvector v_m by

$$\langle\, v_m, \Phi\,(x) \rangle = \sum_{j=1}^{n} \alpha_i^m K\Big(X_i, x\Big)$$

Consider A as a matrix:

$$A = \begin{bmatrix} 0 & 0 \\ 0.999 & 0.9 \\ 0.12223 & 0.312 \\ 0.175674 & 0.1499 \\ 0.245243 & 0.7499 \\ 0.265234 & 0.3599 \\ 0.3 & 0.23239 \\ 0.354373 & 0.32299 \\ 0.4423523 & 0.5 \\ 0.483 & 0.1423999 \\ 0.5 & 0.12 \\ 0.583 & 0.4999 \\ 0.6542323 & 0.1239 \\ 0.6542343 & 0.2399 \\ 0.65523 & 0.643499 \\ 0.6425473 & 0.52342299 \\ 0.683 & 0.654999 \\ 0.7 & 0.72 \\ 0.89 & 0.8899 \\ 0.76 & 0.69 \\ 0 & 0.345 \end{bmatrix}$$

After applying the last step we have the 5.

6.2 Local Linear Embedding

Local Linear Embedding (LLE) is another nonlinear dimension reduction method. The LLE algorithm is comprised of three steps.

- Step 1:K Nearest Neighbors. For each data point, a set of k nearest neighbors is constructed. Brute-force search requires $\mathcal{O}(n^2 d)$ operations.

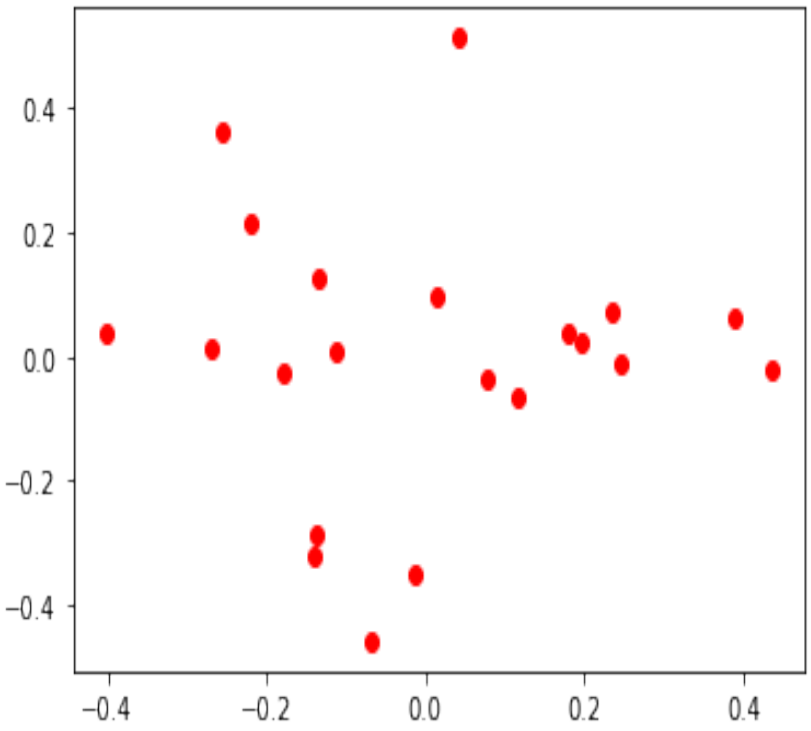

Figure 5: Data after applying kernel pca.

If *approximate* nearest neighbors are calculated, more efficient algorithms are possible. The number of neighbors K is only parameter needed by LLE.

- Step 2: Local weights. a set of weights ω_{ij} is used in this step, which characterize the local geometry of each point. The weights is base on linear combination of neighbors of each input, this is done by solving the least squares problem. When X_i is not one of the k nearest neighbors of X_j so their wights would be zero $\omega_{ij} = 0$. Neighboring is a symmetric relation and weights are mutual.

  The weights ω_{ij} are constrained so that $\omega_{ij} = 0$if X_j is not one of the K nearest neighbors of X_j. Moreover, the weights are normalized to sum to one: $\sum_j \omega_{ij} = 1$, for $i = 1 \cdots n$. This normalization ensures that the optimal weights are invariant to rotation, translation, and scaling.

- Step 3: Linearization. In this step the points $X_i \in \mathbb{R}^d$ are mapped to $YY_i \in \mathbb{R}^m$, where m is the final dimension that the data reduce to it. The vectors are chosen to minimize the reconstruction error under the local

linear mappings constructed in the previous step. That is, the goal is to optimize the functional Y_i

$$\Psi(y) = \sum_{i=1}^{n} \|Y_i - \sum_j \omega_{ij} Y_j\|_2^2 \tag{1}$$

where the weights ω_{ij} are calculated in Step 2. To obtain a unique solution, the vectors are " to have mean zero and unit covariance:

$$\sum_i \|Y_i = 0, \frac{1}{n} \sum_i Y_i Y_i^T = I_m \tag{2}$$

Carrying out this optimization is equivalent to finding eigenvectors by minimizing the quadratic form

$$\Psi(y) = y^T G y \tag{3}$$

$$= y^T (I - W)^T (I - W) y \tag{4}$$

Each minimization gives one of the lower $(m+1)$ eigenvectors of the $n \times n$ matrix $G = (I - W)^T (I - W)$ $(1, 1 \cdots, 1)^T$.

Algorithm 6 LLE

Compute K nearest neighbors for each point;
Compute local reconstruction weights w_{ij} by minimizing
$\Psi(y) = \sum_{i=1}^{n} \|Y_i - \sum_j \omega_{ij} Y_j\|_2^2 \quad subject \quad to \qquad \sum_j \omega ij = 1$
Compute outputs $Y_i \in \mathbb{R}^m$ by computing the first m eigenvectors with nonzero eigenvalues for the $n \times n$ matrix $G = (I - W)^T (I - W)$ The reduced data matrix is $[u_1 \cdots u_m]$ where u_j are the eigenvectors corresponding to the first (smallest) nonzero eigenvalues of$G$.

In the last step, if the graph encoded by the nearest neighbor is not connected, there may be more than one eigenvector with an eigenvalue zero. Another solution for this problem when the graph is disconnected is to run the algorithm separately on each connected component. However, the recommended procedure is to choose K so that the graph is connected. Using the simplest algorithms, the first step has time complexity $\mathcal{O}(dn^2)$, the second step requires $\mathcal{O}(nK^3)$ operations, and the third step, using routines for computing eigenvalues for sparse matrices, requires$\mathcal{O}(mn^2)$ operations (and $\mathcal{O}(n^3)$ operations in the worse case if sparsity is not exploited and the full spectrum is computed). Thus, for high-dimensional problems, the first step is the most expensive. Since the third step computes eigenvectors, it shares the property with PCA that as more dimensions are added to the embedding, the previously computed coordinates do not change.[4]

6.3 Isomap

The difference of this technique with LLE is in mapping and how it assesses similarity between objects, otherwise they both intend to provide a low dimensional manifold representation of a higher dimensional data set. The first step in Isomap is to construct a graph with the nodes representing instances $X_i \in \mathbb{R}^d$ to be embedded in a low dimensional space. Standard choices are a k-nearest neighbors, and ϵ-neighborhoods. In the k-nearest neighborhood graph, each point X_i is connected to its closest k neighbors $\mathcal{N}_k(X_i)$, where distance is measured using Euclidean distance in the ambient space $\mathbb{R}^d$. In the ϵ-neighborhood graph, each point X_i is connected to all points $N_\epsilon(X_i)$ within a Euclidean ball(circle) of radius ϵ centered at X_i . The graph $G = (V, E)$ by taking edge set $V = \{x_1, \cdots, x_n\}$ and edge set

$$(u, v) \in E \quad if \quad v \in \mathcal{N}(u) \quad or \quad u \in \mathcal{N}(v) \tag{5}$$

The next step is to form a distance between points by taking the path distance

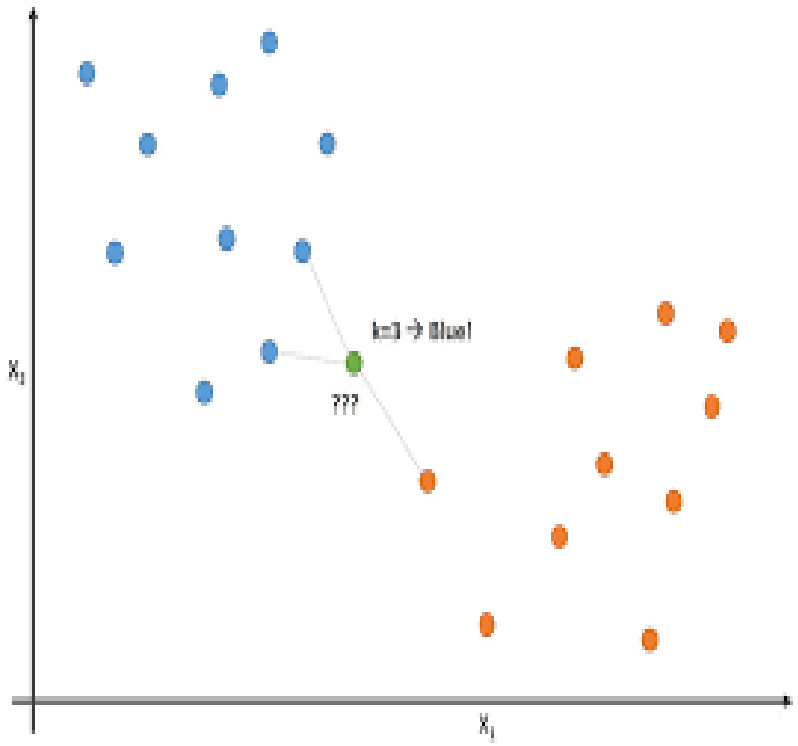

Figure 6: K nearest neighbors.[6]

in the graph. That is $d(X_iX_j)$ is the shortest path between node X_i and X_j.

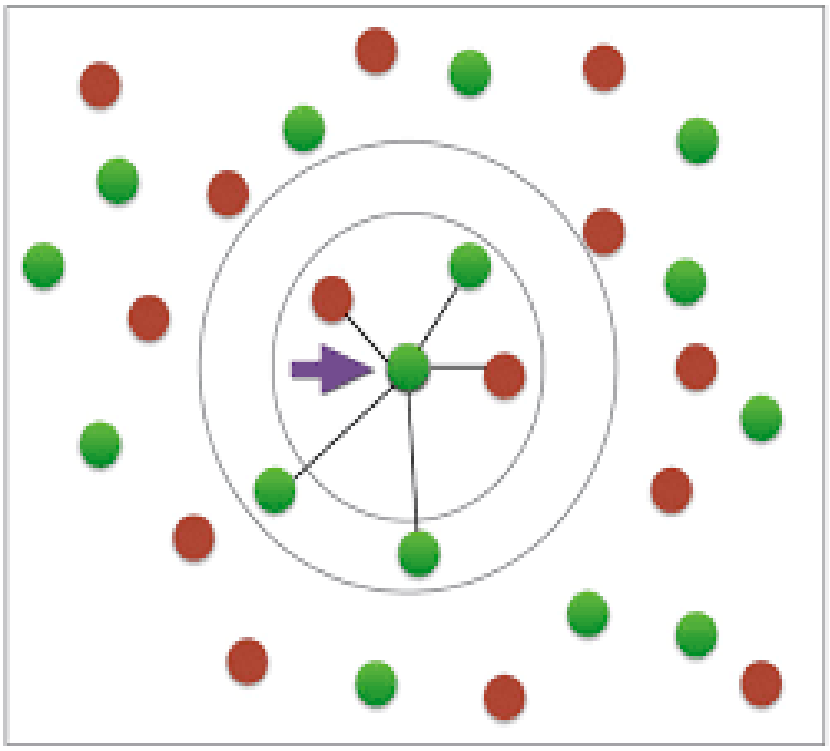

Figure 7: ϵ-neighborhoods.[7]

This distance can be computed for sparse graphs in time $\mathcal{O}(|E_j + |V|log|V|)$. The final step is to embed the points into a low dimensional space using metric multi-dimensional scaling(MDS), which is finding a linear map T to minimize loss function L defines as follows.

$$L = \sum_{i,j} (\|X_i - X_j\|^2 - \|Z_i - Z_j\|^2)$$

Algorithm 7 Isomap

k Compute k nearest neighbors for each point, forming the nearest neighbor graph $G = (V, E)$with vertices X_i.
Compute graph distances $d(X_i, X_j)$ using Dijkstras algorithm
Embed the points into low dimensions using metric multidimensional scaling

Isomap and LLE both obtain nonlinear dimensionality reduction by mapping points into a low-dimensional space, in a manner that preserves the local

geometry. This local geometry will not be preserved by classical PCA or MDS, since far away points on the manifold will be, typically, be mapped to nearby points in the lower dimensional space.[4]

example 3. *The data set can be considered as a matrix, the data set in this example is Heart Disease Dataset[8]. Figures8, 9, 10, and 11 is this dataset after implementing algorithms PCA, SVD, LLE, and Isomap, respectively. Logistic Regression is used to model the records.*
The accuracy of this algorithm can be checked easily. Two models, Logistic Regression(LR) and K-nearest neighborhood are used to evaluate the accuracy of this implementation. KNN is the first step of both Isomap and LLE, so taking it as a model makes sense.[4] The figure is the accuracy of Logistic Regression and Figure is the accuracy of KNN.

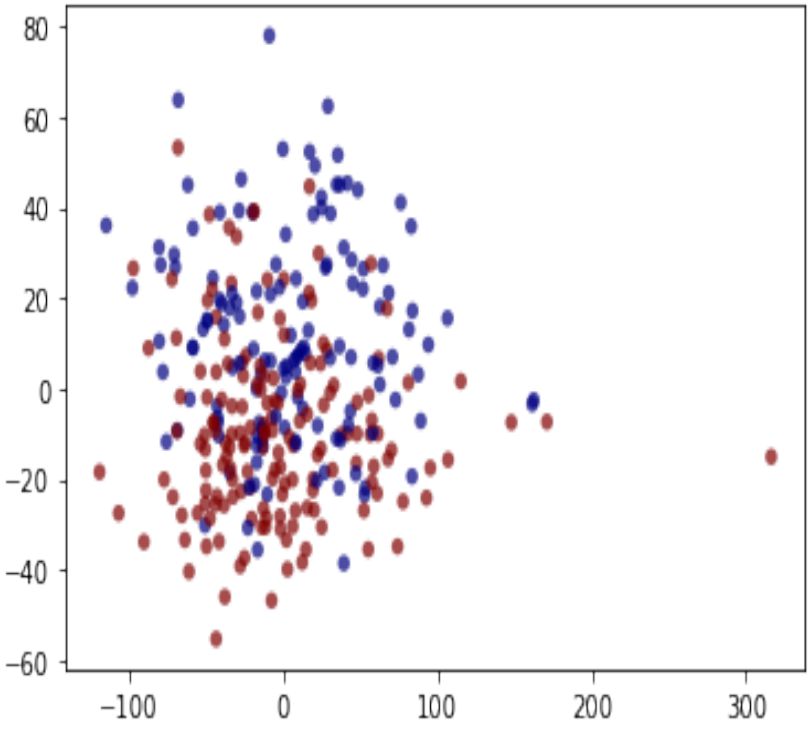

Figure 8: Heart data set after applying PCA.

In LR model, PCA has the highest accuracy. SVD, Isomap, LLE, and Kernel PCA are at the next level in a decreasing way of accuracy. In KNN model K=4 (neighbors) is selected. This model increased the accuracy of the Isomap and LLE and placed them higher than SVD. But they are still less than PCA. KNN model has overall less accuracy than LR. This shows that the model used can

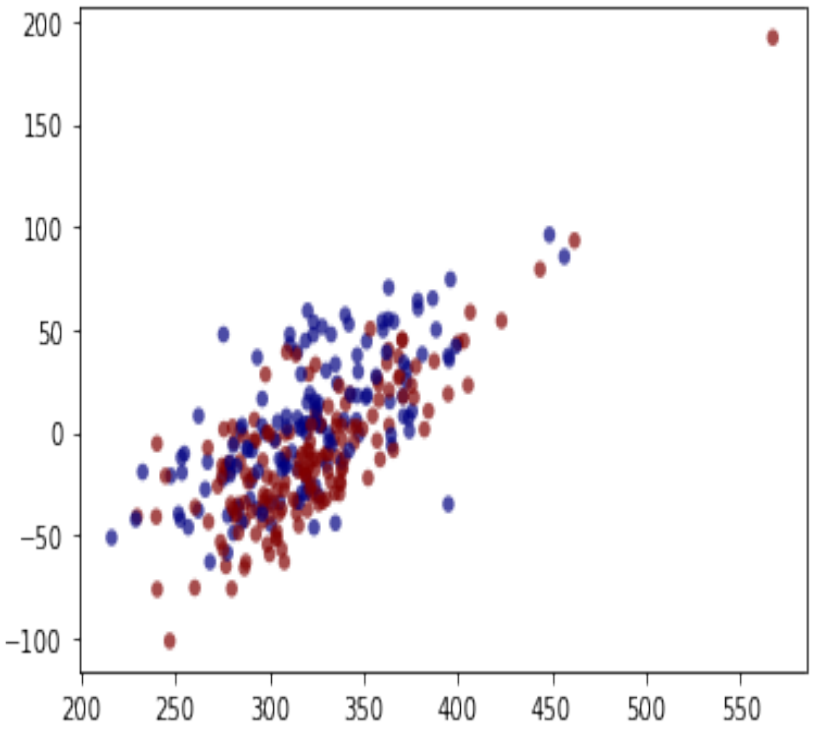

Figure 9: Heart data set after applying SVD.

affect the decision of choosing an algorithm if accuracy was the evaluation tool. However, changing K itself can modify these numbers as well. Consider choosing k=10, then Isomap accuracy will be greater than PCA.

If KMeans was the model, for k=1 all the algorithms have the same accuracy. Figure 15 shows their accuracy. By increasing the number of clusters, unlike another algorithm, the accuracy of kernel PCA will be increased.

Compare figures 16 and 17 to see this change. All being said, PCA works better for this data set. Although using Kmeans with big k fitting with Isomap will work better, the accuracy of PCA in other models is higher numerically. But if the model is unsupervised learning especially Kmeans with greater clusters, it is beneficial to use Isomap or LLE. In this data set, it seems that the Kmeans model is not the best choice as well.

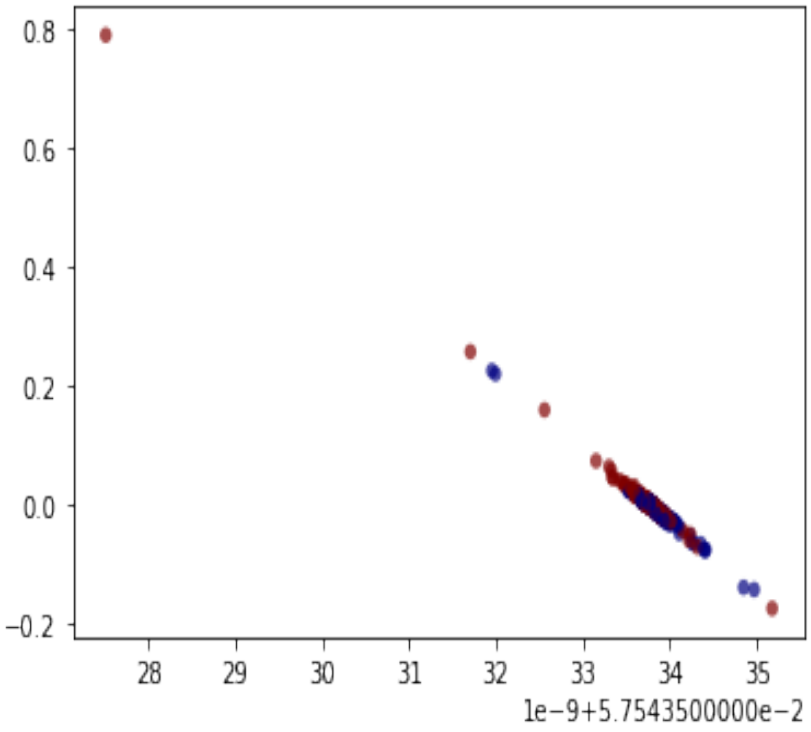

Figure 10: Heart data set after applying LLE.

model	K	PCA	SVD	iso	LLE	KPCA
LR	-	0.704	0.634	0.674	0.554	0.543
KNN	5	0.603	0.533	0.609	0.552	0.457
	10	0.648	0.593	0.666	0.564	0.476
Kmeans	2	0.469	0.490	0.503	0.475	0.537
	3	0.289	0.345	0.292	0.403	0.525

Table 1: DR algorithm comparison

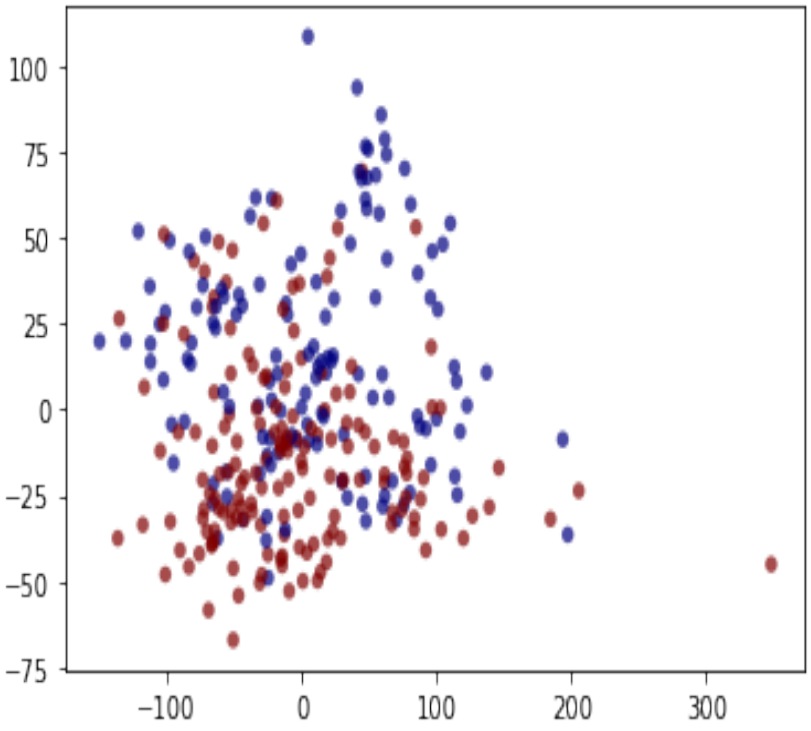

Figure 11: Heart data set after applying Isomap.

```
('pca', PCA(n_components=2)) Accuracy: 0.704 (0.085)
('svd', TruncatedSVD()) Accuracy: 0.634 (0.092)
('iso', Isomap()) Accuracy: 0.674 (0.082)
('lle', LocallyLinearEmbedding()) Accuracy: 0.554 (0.025)
('kpca', KernelPCA(gamma=15, kernel='rbf')) Accuracy: 0.543 (0.013)
```

Figure 12: Accuracy of different DR algorithms on Heart data set using Logistic Regression.

```
('pca', PCA(n_components=2)) Accuracy: 0.603 (0.087)
('svd', TruncatedSVD()) Accuracy: 0.533 (0.092)
('iso', Isomap()) Accuracy: 0.609 (0.081)
('lle', LocallyLinearEmbedding()) Accuracy: 0.552 (0.078)
('kpca', KernelPCA(gamma=15, kernel='rbf')) Accuracy: 0.457 (0.013)
```

Figure 13: Accuracy of different DR algorithms on Heart data set using KNN.

```
('pca', PCA(n_components=2)) Accuracy: 0.648 (0.099)
('svd', TruncatedSVD()) Accuracy: 0.593 (0.097)
('iso', Isomap()) Accuracy: 0.666 (0.093)
('lle', LocallyLinearEmbedding()) Accuracy: 0.564 (0.079)
('kpca', KernelPCA(gamma=15, kernel='rbf')) Accuracy: 0.476 (0.038)
```

Figure 14: Accuracy of different DR algorithms with KNN on Heart dataset using k=10.

```
('pca', PCA(n_components=2)) Accuracy: 0.457 (0.013)
('svd', TruncatedSVD()) Accuracy: 0.457 (0.013)
('iso', Isomap()) Accuracy: 0.457 (0.013)
('lle', LocallyLinearEmbedding()) Accuracy: 0.457 (0.013)
('kpca', KernelPCA(gamma=15, kernel='rbf')) Accuracy: 0.457 (0.013)
```

Figure 15: Accuracy of different DR algorithms with Kmeans on Heart data set using k=1.

```
('pca', PCA(n_components=2)) Accuracy: 0.469 (0.103)
('svd', TruncatedSVD()) Accuracy: 0.490 (0.109)
('iso', Isomap()) Accuracy: 0.503 (0.121)
('lle', LocallyLinearEmbedding()) Accuracy: 0.475 (0.075)
('kpca', KernelPCA(gamma=15, kernel='rbf')) Accuracy: 0.537 (0.026)
```

Figure 16: Accuracy of different DR algorithms with Kmeans on Heart data set using k=2.

```
('pca', PCA(n_components=2)) Accuracy: 0.289 (0.098)
('svd', TruncatedSVD()) Accuracy: 0.345 (0.088)
('iso', Isomap()) Accuracy: 0.292 (0.106)
('lle', LocallyLinearEmbedding()) Accuracy: 0.403 (0.121)
('kpca', KernelPCA(gamma=15, kernel='rbf')) Accuracy: 0.525 (0.037)
```

Figure 17: Accuracy of different DR algorithms with Kmeans on Heart data set using k=2.

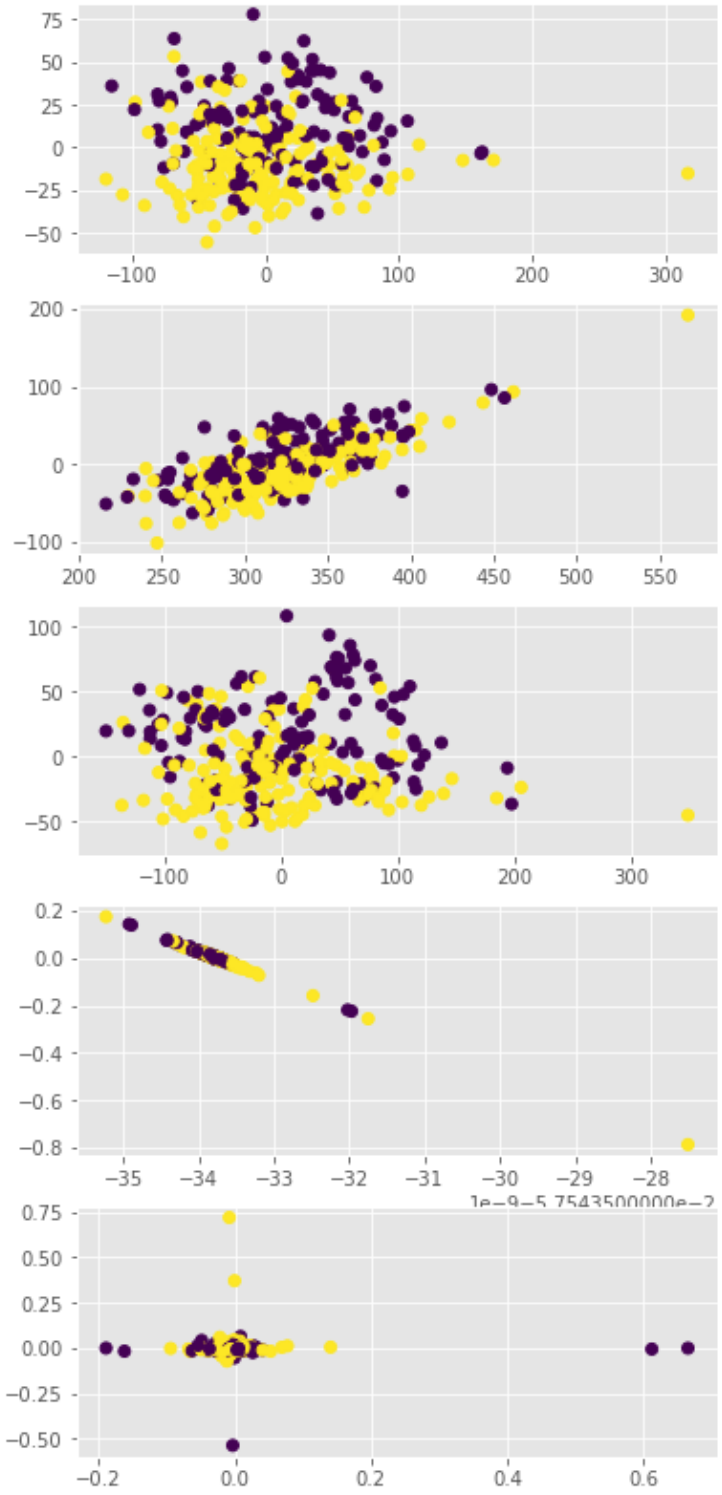

Figure 18: All DR algorithms on Heart Disease data set.

7 Conclusion

In conclusion, there are many variables that effect DR's efficiency[3]. The Data set itself can change the decision. Overall, if the data set needs an unsupervised learning model, especially Kmeans, Isomap and LLE are better choices. In other cases, PCA (or probably Kernel PCA) is the best option.

[3]by efficiency, we mean the less lose the quality of original data. This can give a better sense of data

References

[1] Gentle, James E. (2012-12-06).*Numerical Linear Algebra for Applications in Statistics.* ISBN 9781461206231.

[2] Anton, Howard; Rorres, Chris (2005).*Elementary Linear Algebra (9th ed.).* John Wiley and Sons, Inc. ISBN 978-0-471-66959-3.

[3] Trefethen, Lloyd; Bau, III, David (1997). *Numerical Linear Algebra*. ISBN 978-0898713619.

[4] https://www.stat.cmu.edu/ larry/=sml/DimRed.pdf

[5] I A N T. JOLLIFFE.*PRINCIPAL COMPONENT ANALYSIS: A BEGINNERS GUIDE- I. Introductionand application*. Institute ofMathematics, University ofKent, Canterbury DOI: 10.1002/j.1477-8696.1990.tb05558.x.

[6] https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a

[7] https://rapidminer.com/blog/k-nearest-neighbors-laziest-machine-learning-technique/

[8] https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset