Application of MDS, kernel-PCA and ISOMAP to benchmark problem i.e. concentric circles, Swiss roll

Karan Kabbur Hanumanthappa Manjunatha

December 17, 2022

1 Problem description:

In this report, we are going to produce data to represent two concentric circles with random noise as well for the Swiss roll also and visualize them. They are considered as benchmark problems which are exploited to understand the idea behind non-linear manifold learning algorithms (Kernel PCA and ISOMAP) for finding the correct variables of high- dimensional complex systems. We apply Multidimensional Scaling(MDS), Kernel PCA(k-PCA) (using the Gaussian kernel) and ISOMAP to find an embedding of the two concentric circles with random noise as well as for the Swiss roll data.

2 Theory

2.1 Multi-dimensional scaling, (MDS)

The classical MDS is based on the fact that the coordinate matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ with n being the number of samples and m being the features/coordinates, can be obtained from the matrix $\mathbf{B} = \mathbf{X}\mathbf{X}^{\mathbf{T}}$. However, $\mathbf{X}$ is usually not known but instead proximity matrix such as *distance matrix* is known. The procedure to implement MDS is as follows[4][1]: Given a distance matrix D which is symmetric, where D_{ab} may refer to distance between city a and city b, the inverse problem is solve and obtain the position coordinates of the cities or in other words to recover the map.

1. Square the entries of the distance matrix, $D^{(2)} = [d^2]$.
2. Given n which is the number of cities, apply the double centering to squared distance matrix D^2.

$$B = -\frac{1}{2} H D^2 H^T \tag{1}$$

   with matrix $H = I - \frac{1}{n} 1.1^T$, where $1 = [1, 1, ...1]^T$ is a column vector of $1's$ whose dimension is $n \times 1$.
3. Diagonalize the matrix B and extract the corresponding eigenvalues and eigen vectors.
4. Sort the $k < m$ largest largest eigen values $\lambda_1 > \lambda_2 > ....\lambda_k$ and the corresponding eigen vectors in $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_k$.
5. A k-dimensional spatial configuration of the n objects(or cities) is derived from the coordinate matrix

$$\mathbf{X} = \mathbf{E}_k \Lambda_k^{\frac{1}{2}} \tag{2}$$

   where $\mathbf{E}_k$ is the matrix of k largest eigenvectors and Λ is the diagonal matrix of k eigenvalues of $\mathbf{B}$.

2.2 kernel-PCA

[2]

Given the data of $\mathbf{X} = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{m \times n}$ with m being the dimensionality or number of features and n being the number of samples, we would like to map the data non linearly to a feature space $\mathbf{F}$ by a non linear transformation $\phi : \mathbb{R}^m \rightarrow \mathbf{F}$ where $\mathbf{F}$ has a large dimensionality M. Mathematically, it can be represented as: $\phi(X) = \begin{pmatrix} | & | & & | \\ \phi(\mathbf{x}_1) & \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x_n}) \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{M \times n}$ where M is greater than m. In the higher dimensional space, we can then do PCA and the result will be non-linear in the original data space.

PCA in feature space: Let us suppose that for the moment that the mean of the data in feature space is 0, i.e. $\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i) = 0$. The covariance matrix is given by:

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T \tag{3}$$

The eigenvectors are:

$$\mathbf{C}\mathbf{v}_j = \lambda_j\mathbf{v}_j \tag{4}$$

where $j = 1, ..., M$. We want to avoid explicitly going to feature space - instead we want to work with kernels:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_k) \tag{5}$$

Now, we rewrite the PCA Eq.4 by substituting Eq. 3 in Eq. 4:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\mathbf{v}_j = \lambda_j\mathbf{v}_j, j = 1, ..., M \tag{6}$$

By observing the aboce equation, the eigen vectors can be written as linear combination of features:

$$\mathbf{v}_j = \sum_{i=1}^{n}a_{ji}\phi(\mathbf{x}_i) \tag{7}$$

Then finding the eigenvectors given in Eq. 7 is equaivalent to finding the coefficients $a_{ji}, j = 1, ..., M$ and $i = 1, .., n$. By substituting the Eq. 7 back into Eq. 6, we get:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\left(\sum_{i=1}^{n}a_{jl}\phi(\mathbf{x}_l)\right) = \lambda_j\sum_{i=1}^{n}a_{jl}\phi(\mathbf{x}_l) \tag{8}$$

Substituting the Kernel defined in Eq. 5 in the above equation, we can rewrite it as:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\left(\sum_{i=1}^{n}a_{jl}K(\mathbf{x}_i, \mathbf{x}_l))\right) = \lambda_j\sum_{i=1}^{n}a_{jl}\phi(\mathbf{x}_l) \tag{9}$$

Now, we do a small trick by multiplying this above equation by $\phi(\mathbf{x}_i)^T$ to the left and plug in the Kernel again.

$$\frac{1}{n}\sum_{i=1}^{n}K(\mathbf{x}_k, \mathbf{x}_i)\left(\sum_{i=1}^{n}a_{jl}K(\mathbf{x}_i, \mathbf{x}_l))\right) = \lambda_j\sum_{i=1}^{n}a_{jl}\phi(\mathbf{x}_l) \tag{10}$$

By rearranging the above equation, we get: $\mathbf{K}^2\mathbf{a}_j = m\lambda_j\mathbf{K}\mathbf{a}_j$ where $\mathbf{K} = [K(\mathbf{x}_i, \mathbf{x}_j)] \in \mathbb{R}^{n\times n}$ and $n = M$ is considered here. And $\mathbf{a}_j = [a_{j1}, a_{j2}, ..., a_{jn}]^T$. Then from the equation, we remove $\mathbf{K}$ from both sides:

$$\mathbf{K}\mathbf{a}_j = m\lambda_j\mathbf{a}_j \tag{11}$$

We have normalization condition for $\mathbf{a}_j$ vectors:

$$\begin{aligned}\mathbf{v}_j^T\mathbf{v}_j &= 1\sum_{k=1}^{n}\sum_{l=1}^{n}a_{jl}a_{jk}\phi(\mathbf{x}_l)^T\\ \phi(\mathbf{x}_k) &= 1\\ \mathbf{a}_j^T\mathbf{K}\mathbf{a}_j &= 1\end{aligned}$$

Plugging this into Eq. 11, we get:

$$\lambda_j m\mathbf{a}_j^T\mathbf{a}_j = 1 \tag{12}$$

For a new point $\mathbf{x}$, its projection onto the principal components is:

$$\phi(\mathbf{x})^T \mathbf{v}_j = \sum_{i=1}^{n} a_{ji} \phi(\mathbf{x})^T \phi(\mathbf{x}_i) = \sum_{i=1}^{n} a_{ji} K(\mathbf{x}, \mathbf{x}_i) \tag{13}$$

Normalizing the feature space: In general, $\phi(\mathbf{x}_i)$ may not be centered at high dimensional space. And so, we want to work with:

$$\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \frac{1}{n} \sum_{k=1}^{n} \phi(\mathbf{x}_k) \tag{14}$$

The corresponding kernel matrix entries are given by:

$$\tilde{K}(\mathbf{x}_k, \mathbf{x}_l) = \tilde{\phi}(\mathbf{x}_k)^T \tilde{\phi}(\mathbf{x}_l) \tag{15}$$

After some algebra, we finally get the centered Kernal matrix:

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbb{1}_{1/n}\mathbf{K} - \mathbf{K}\mathbb{1}_{1/n} + \mathbb{1}_{1/n}\mathbf{K}\mathbb{1}_{1/n} \tag{16}$$

where $\mathbb{1}_{1/n}$ is matrix with all elements equal to $\frac{1}{n}$.

The algorithm of the Gaussian RBF Kernel PCA given the dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$ with m features and n samples, is as follows:

1. Consider the Gaussian RBF Kernel defined by: $K(\mathbf{x}, \mathbf{y}) = exp\left(-\gamma ||\mathbf{x} - \mathbf{y}||_2^2\right)$ where γ is the hyperparameter that has to be tuned based on the problem given.
2. Construct the Kernel matrix $\mathbf{K}$ according to the Eq. 5.
3. Center the Kernel to obtain $\tilde{\mathbf{K}}$ given by Eq. 16.
4. Solve for eigen values and eigen vectors of this centered Kernel $\tilde{\mathbf{K}}$ and sort the eigen values λ_j in descending order along with its corresponding eigen vectors, $\mathbf{a}_j$.
5. Project in a low dimensional subspace $d << n$ i.e. PCA i.e. d leading eigen vectors.
6. Embedding - For any data point (new or old), we can represent it as the following set of features

$$y_j = \sum_{i=1}^{n} a_{ji} K(\mathbf{x}, \mathbf{x}_i), j = 1, ..., n \tag{17}$$

2.3 Isometric Mapping(ISOMAP)

ISOMAP is an approach to solve dimensionality reduction problems that uses easily measured local metric information to learn the underlying global geometry of a data set. Unlike classical techniques such as principal component analysis (PCA) and multidimensional scaling (MDS), this approach is capable of discovering the nonlinear degrees of freedom that underlie com- plex natural observations, such as human handwriting or images of a face under different viewing conditions[3]. There are many data sets which contain nonlinear structures that are invisi ble to PCA and MDS for eg. Swiss roll. For examples such as this the PCA and MDS effectively see just the Euclidean structure and they fail to detect the intrinsic two-dimensionality, whereas geodesic distances reflect the true low-dimensional geometry of the manifold.

The complete Isometric feature mapping has three main steps[3]:

1. **Construct neighborhood graph** - Given data points in high dimensional space, create a weighted graph matrix connecting all these points such that the graph is **connected**. When we say the graph is connected, it means there exists a path from any one point to any other point and so there are no breakages in the graph. In the *sklearn* library of Python there is *sklearn.neighbors* module by which takes the input:
   (a) the dataset $\mathbf{X} \in n$ samples $\times m$ features
   (b) k-nearest neighbors to be considered for each sample or data point
   (c) specify that we require euclidean distance as the weight of the edges between the nodes(i.e. samples)

   Let's call it the graph G which connects all the data points i and j and the weighted distance between them as is $d_X(i, j)$.
2. **Compute shortest paths** - Consider the above matrix of weighted distances between of the k nearest neighbors of each node(or data point) as input to the *dijkstra()* function of the *scipy.sparse.csgraph* module of Python. The

Dijkstra algorithm, provided such a matrix with entries initialized as $d_G(i,j) = d_X(i,j)$ as input, it computes the shortest distance between any point i and any point j in the graph. Mathematically, it is denoted as, for each value of $k = 1, 2, ..N$ we replace the entries $d_G(i,j)$ by $min\{d_G(i,j), d_G(i,k) + d_G(k,j)\}$. Then in the end, we obtain finally Geodesic distance matrix $D_G = \{d_G(i,j)\}$ which contains shortest path distances between all pairs of points in G.

3. **Construct d-dimensional embedding** - Apply the mMDS algorithm mentioned in the previous section, but not on the euclidean distance matrix as we usually do for MDS, but instead use the computed Geodesic matrix D_G as an input to it. All the process remains the same as in MDS.

2.4 Performance evaluation of the methods:

The evaluation of these methods(k-PCA, MDS, Isomap) on comparable grounds are done by computing and comparing the residual variances. A generalized formula for computing the residual variance is given by[3], *Ref.42*:

$$Residual \quad Variance = 1 - R^2(\tilde{D}_M, D_Y) \tag{18}$$

where the terms are:

1. D_Y is the matrix of Euclidean distances in the low-dimensional embedding recovered by each algorithm.
2. $\tilde{D}_M$ is each algorithm's best estimate of the intrinsic manifold distances
   (a) $\tilde{D}_M$, for Isomap is the graph distance matrix or geodesic distance matrix computed after Steps 1, 2 of the procedure given in Isomap in the previous subsection.
   (b) $\tilde{D}_M$, for MDS is Euclidean input-space distance matrix D_X given the dataset $\mathbf{X} \in n$ samples $\times m$ coordinates/features.
   (c) $\tilde{D}_M$, for kernel-PCA it is the Euclidean distance in kernel space between the points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$. Since we don't know the transformation $\phi()$, we write in terms of Kernel matrix as:

$$||\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)||_2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j) \tag{19}$$

3. $R()$ is the standard linear correlation coefficient, taken over all entries of $\tilde{D}_M$ and D_Y.

3 Results

3.1 Case study for concentric circles

The plot of concentric circles with gaussian noise = 0.1 as scattered data points is shown in Fig. 1. The samples from each class represented in the colors of red and blue cannot be linearly separated, i.e. there is no straight line that can split the samples of the inner set from the outer set. As observed from Fig. 3, using a gaussian kernel allows to make a non-linear projection. Here, by using an RBF kernel, we observe that the projection unfold the dataset while keeping approximately preserving the relative distances of pairs of data points that are close to one another in the original space. It is the same case for Isomap as seen in Fig. 4 where the data both in 1D projection as well as 2D projection is seperated into two clusters. From these two methods, we see that the samples of a given class are closer to each other than the samples from the opposite class, untangling both sample sets. Now, we can use a linear classifier to separate the samples from the two classes. However, the same cannot be said to metric MDS result as seen from result Fig. 2. In the 1D projection of MDS, the two clusters overlap and in the 2D projection we get back the result similar to original, so we see that such a projection would not help if define a linear classifier to distinguish samples from both classes. Again looking at the Fig. 2 on the right, the projection made using MDS, we see that there is no change regarding the scaling; indeed the data being two concentric circles centered in zero, the original data is already isotropic. However, we can see that the data have been rotated. Thus we can conclude that such a projection would not help if we define a linear classifier to distinguish samples from both classes. Thus, we can say that MDS fails for the this dataset.

To compare the performance of the methods for the case of concentric circles, we have computed the residual variances as given in Eq. 18 and it is depicted in Fig. 9a. In all caseses, residual variance decreases as the dimensionality is increased. The intrinsic dimensionality of the data can be estimated by looking for the "elbow" at which this curve ceases to decrease significantly with added dimensions. Surprisingly, we observe that residual variance is high for initial few dimensions unlike the MDS or Isomap which bottoms out at dimension=2.

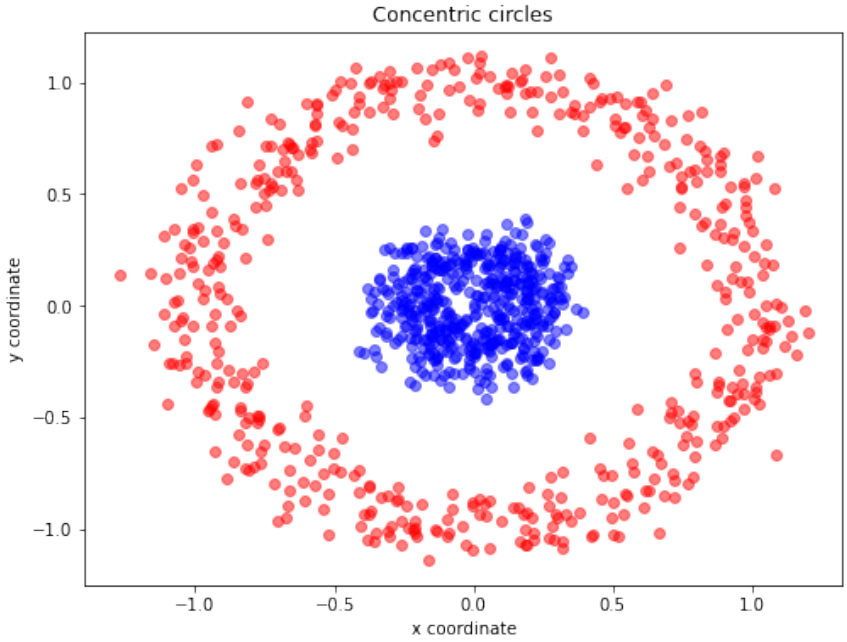

Figure 1. 2D representation of concentric circles with gaussian noise = 0.1. The inner circle and outer circle clusters are distinguished by their colors, blue and red respectively.

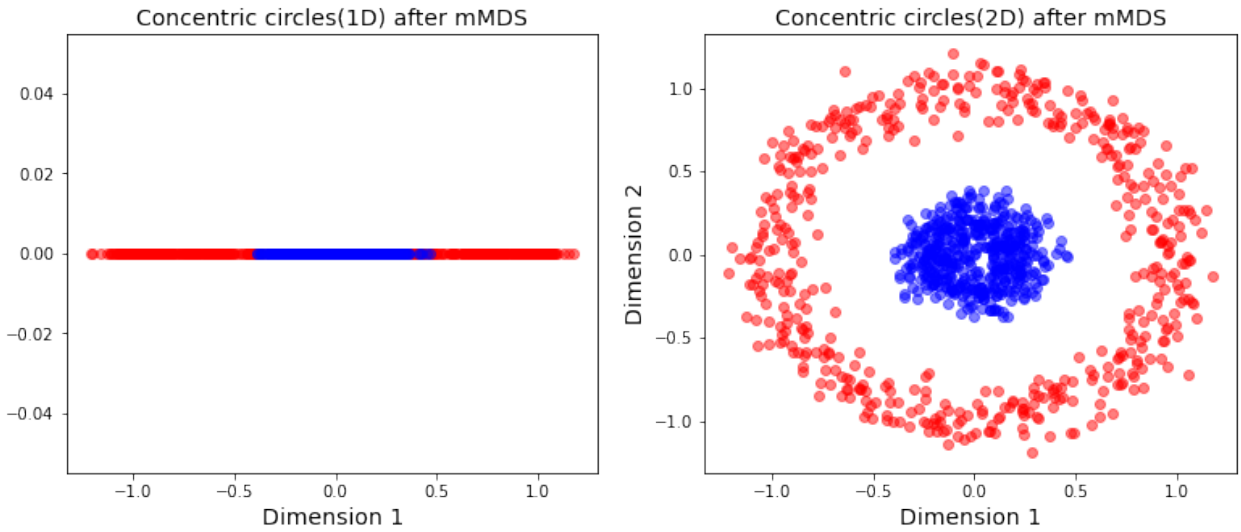

Figure 2. Result of embedding by applying metric MDS. Only Dimension 1 plotted on the left, while on the right the we have Dimension 1 vs Dimension 2 of the projected and scaled feature space.

3.2 Case study of Swiss roll

The 3D plot of Swiss roll with gaussian noise = 0.2 and number of samples = 2048 is shown in Fig. 5. As seen from the Fig. 6 on th eright, the shape of a 2D object in MDS looks like a picture taken of the same 3D object but from a specific angle. Computing the Isomap embeddings, we find that it seems to unroll the Swiss Roll pretty effectively even in 1D as well as in 2D as shown in Fig. 8 as compared to methods such as k-PCA and MDS shown by Fig. 6, 8. Thus, the methods MDS and k-PCA fails to unroll the swiss roll or in other words capture the its inherent structure in the low dimensional space.

To compare the performance of the methods for the case of Swiss roll, we have computed the residual variances as given in Eq. 18 and it is depicted in Fig. 9b. The methods k-PCA and MDS which is a classical method fails but in contrast the residual variance of Isomap bottoms out at dimension = 2.

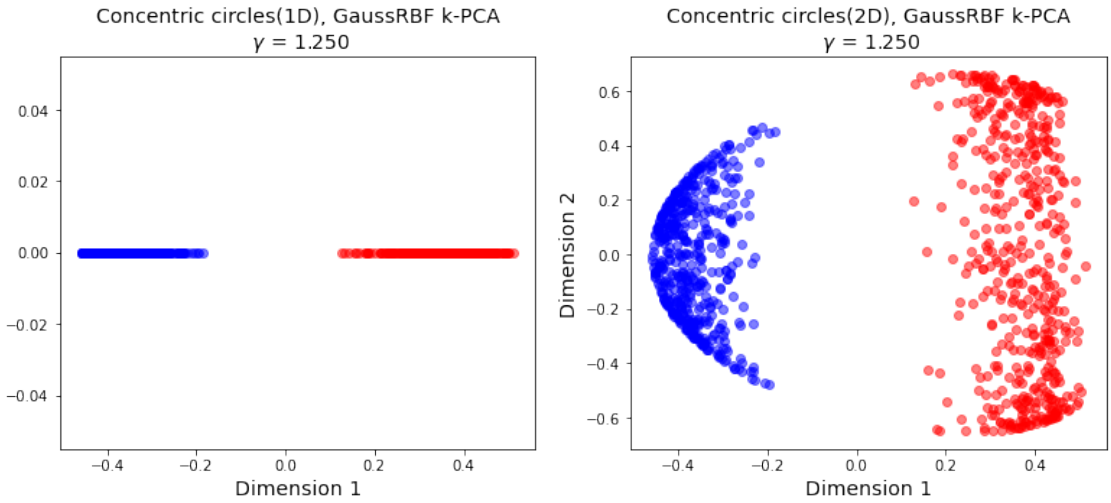

Figure 3. Results of concentric dataset projected onto first two principal components after applying the method Gaussian RBF kernel-PCA, with the hyperparameter $\gamma = 0.1$ chosen.

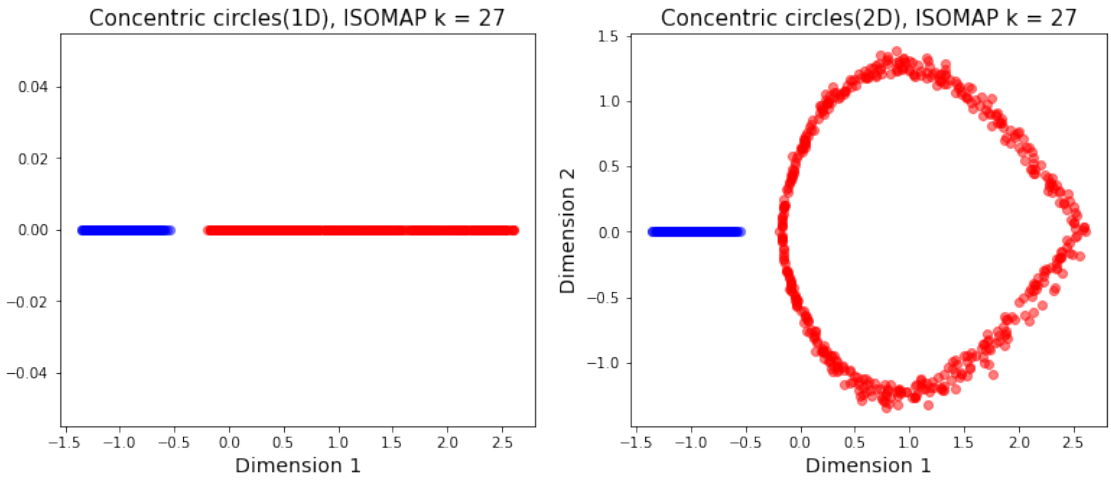

Figure 4. Results of concentric circles dataset embedded/projected into first 2 dimensions after applying the ISOMAP, with the hyperparameter k-nearest neighbors chosen to be 15.

4 Conclusions

Applying the three dimensionality reduction methods on the concentric circles and the Swiss roll, we can conclude the following:

1. For the case on concentric circles, the classical metric MDS which is dependent on the euclidean distance between the data points, fails to differentiate the two sets of data points(red and blue) (both in 1D and 2D projections) into two seperate groups/clusters but non-linear manifold algorithms such as k-PCA and Isomap succeeds for a given certain values of hyperparameters.
2. Only Isomap succeeds in learning the non-linear manifold and so it unfolds nicely in the case of Swiss roll dataset. It is done such that Geodesic distance between any two points in the original space is almost equivalent to euclidean distance between those same points in the projected embedded space.
3. In the case of Swiss roll, Isomap performs well as it bottoms out at dimension = 2, when residual variance vs dimsnions are plotted, unlike the other two methods which fails.
4. The residual variance of k-PCA for the first few dimensions are observed to be higher than MDS even though it separates out the clusters distinctively.

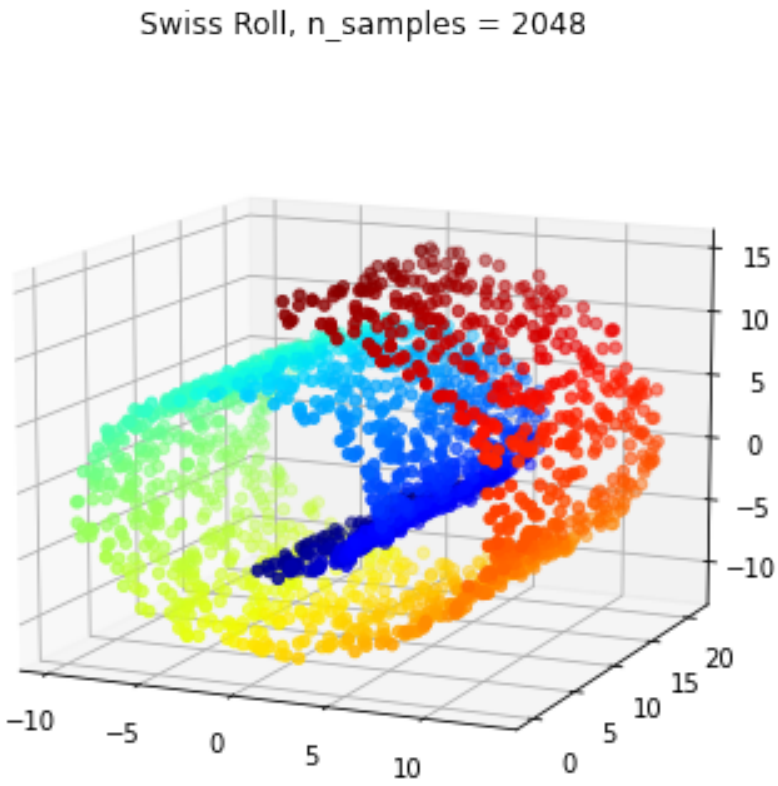

Figure 5. 3D representation of Swiss roll with of number of data points = 2048 and gaussian noise = 0.2.

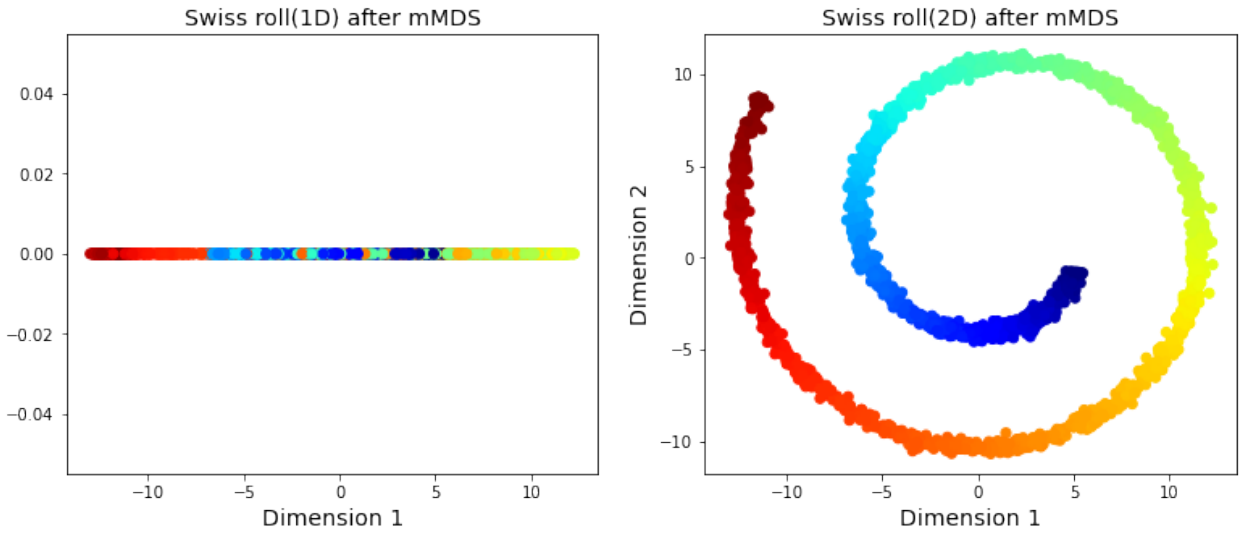

Figure 6. Result of embedding Swiss roll dataset by applying metric MDS. Only Dimension 1 plotted on the left, while on the right the we have Dimension 1 vs Dimension 2 of the projected and scaled feature space.

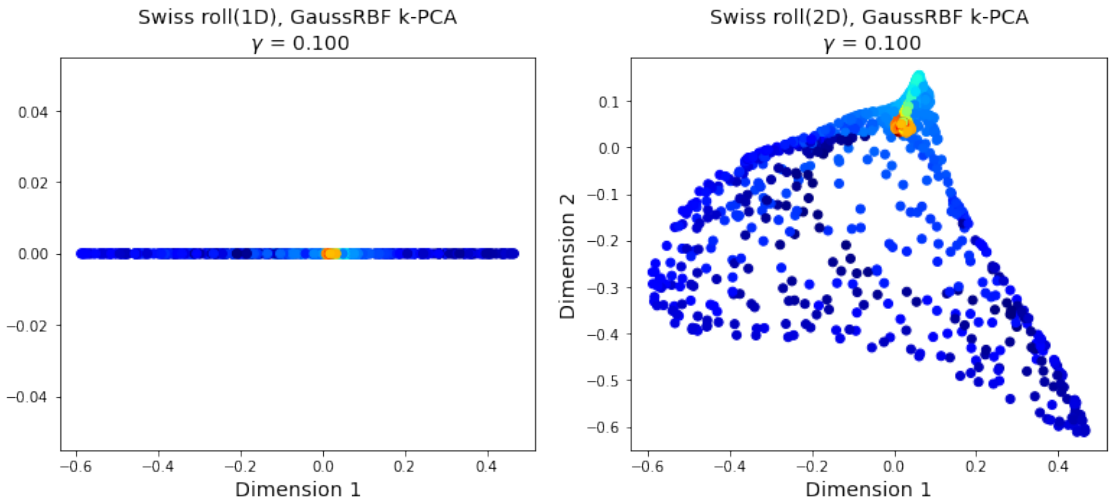

Figure 7. Results of Swiss roll dataset projected onto first two principal components after applying the method Gaussian RBF kernel-PCA, with the hyperparameter $\gamma = 15$ chosen.

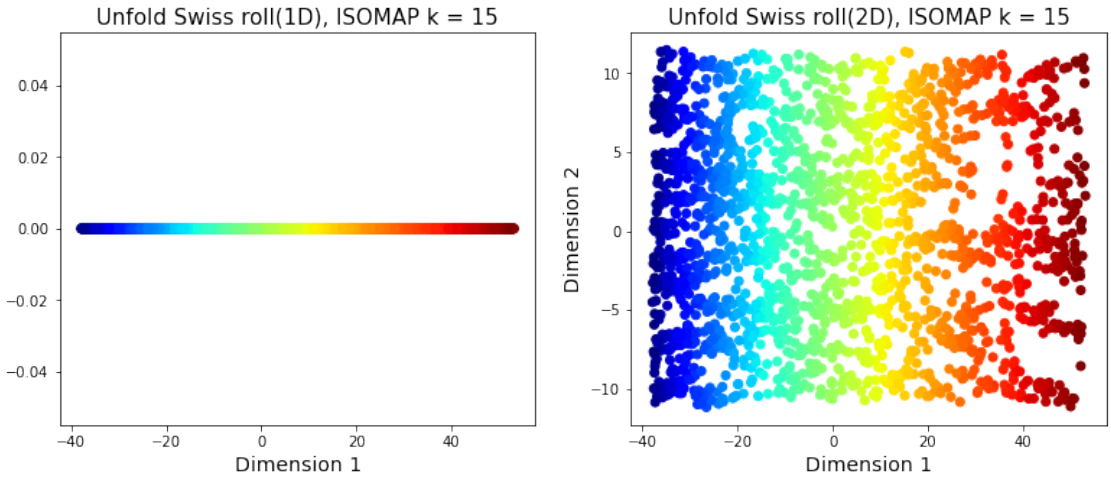

Figure 8. Results of Swiss roll dataset embedded/projected into first 2 dimensions after applying the ISOMAP, with the hyperparameter k-nearest neighbors chosen to be 27.

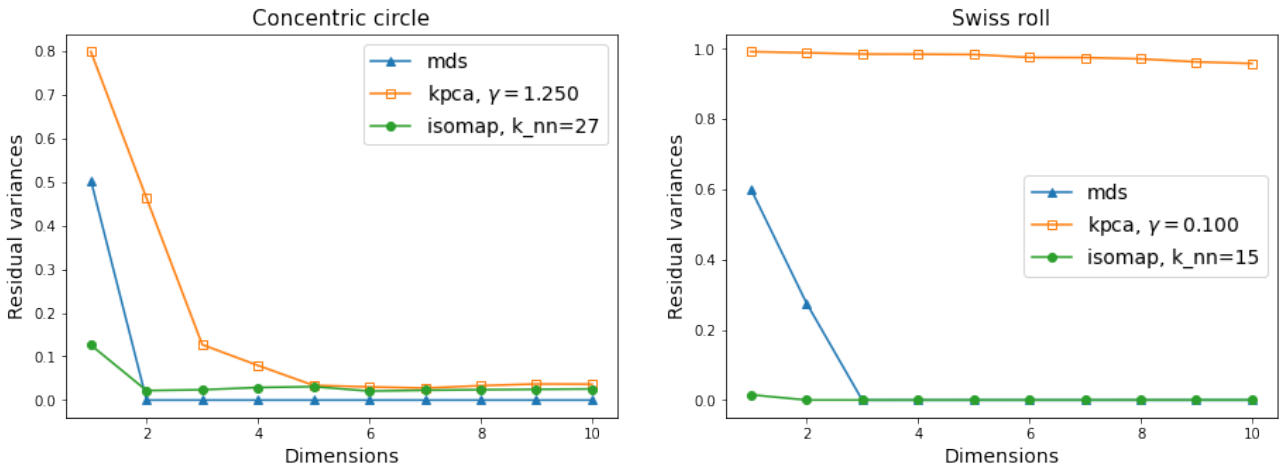

Figure 9. Performance of the methods. Residual variances computed according to Eq. 18 for **a).** Concentric cirlces **b).** Swiss roll

References

[1] Ingwer Borg and Patrick J F Groenen. *Modern multidimensional scaling*. en. Springer Series in Statistics. New York, NY: Springer, May 2000.

[2] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". In: *Neural Computation* 10.5 (July 1998), pp. 1299–1319. DOI: 10.1162/089976698300017467. URL: https://doi.org/10.1162/089976698300017467.

[3] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (Dec. 2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319. URL: https://doi.org/10.1126/science.290.5500.2319.

[4] Florian Wickelmaier. "An introduction to MDS". In: (Apr. 2003).

5 Appendix

```python
from sklearn.datasets import make_circles
# create data points for plotting 2D concentric circles
X, y = make_circles(n_samples=1000, random_state=123, noise=0.1, factor=0.2)

# Plot the data points for visualization
plt.figure(figsize=(8,6))

plt.scatter(X[y==0, 0], X[y==0, 1], color='red', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', alpha=0.5)
plt.title('Concentric circles')
plt.ylabel('y coordinate')
plt.xlabel('x coordinate')
plt.show()
```

Listing 1. Code for creating and visualizing Concentric circles with noise

```python
# Create Swiss roll dataset:
from sklearn.datasets import make_swiss_roll
X, t = make_swiss_roll(n_samples=2048, noise=0.2, random_state=42)

# visualize Swiss roll:
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=t, cmap=plt.cm.jet)#, s=50, alpha=0.8)
ax.view_init(azim=-66, elev=12)

ax.set_title("Swiss Roll, n_samples = %d"%2048)
plt.show()
```

Listing 2. Code for creating and visualizing Swiss roll with noise

```python
def apply_MDS(D, k=2):
    # dimension of the distance matrix
    n = D.shape[0]

    I = np.eye(n)

    # column vector of ones of dim = (n,1)
    one = np.ones(n).reshape(-1,1)

    # operator
    H = I - (1./n)*np.dot(one, one.T)

    B = -0.5*np.matmul(H, np.matmul(D**2, H.T))

    # find evals, and eig vectors
```

```
    eig_vals, eig_vecs = np.linalg.eig(B)
    srt = np.argsort(eig_vals)[::-1] # indices of eigenvalues from largest to smallest
    eig_vals_srtd = eig_vals[srt] # sorted eigenvalues
    eig_vecs_srtd = eig_vecs[:, srt] # sorted eigenvectors

    # select first k largest evals and evectors
    Eval_2 = eig_vals_srtd[:k]
    Evec_2 = eig_vecs_srtd[:,:k]

    # coordinates of dimension, n x k
    X = np.matmul(Evec_2, np.diag(np.sqrt(Eval_2)))
    return X
```

Listing 3. Algorithm for MDS which takes Euclidean distance matrix and number of components for the data to be projected into, as inputs

```
def gaussRBF_kernel_pca(X, gamma, n_components):
    """
    RBF kernel PCA implementation.

    Parameters
    ------------
    X: {NumPy ndarray}, shape = [n_samples, n_features]
    gamma: float

    Tuning parameter of the RBF kernel
    Number of principal components to return
    n_components: int

    Returns
    ------------
    Projected dataset
    X_pc: {NumPy ndarray}, shape = [n_examples, k_features]
    """
    from sklearn.metrics.pairwise import euclidean_distances

    mat_sq_dists = euclidean_distances(X, squared=True)

    # Compute the symmetric kernel matrix.
    K = np.exp(-gamma * mat_sq_dists)

    # Center the kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

    # Set the Kernel upto 8 decimal places
    K = np.around(K, decimals=8)

    # Check for symmetry
    #print("Is symmetric?", np.all(K==K.T))

    # 3. Compute eigenvectors and values of the kernel
    eig_vals , eig_vecs = np.linalg.eig( K )

    # 4. Sort and choose the first 3 largest eigenvectors
    srt = np.argsort( eig_vals )[:: -1]

    eig_vals_srtd = eig_vals[srt] # sorted eigenvalues
    eig_vecs_srtd = eig_vecs[:, srt]
    #n_components = 2


```

```
    # 5. Project to a low dimensional subspace d(=n_components)
    X_pc = eig_vecs_srtd [:,0:n_components]

    # 6. Scale the projected components
    X_pc = X_pc * np.sqrt(eig_vals_srtd[0:n_components])#np.diag( np.sqrt(eig_vals_srtd[0:
    n_components])) )

    return X_pc, eig_vals_srtd
```

Listing 4. Function implmented for applying Kernel PCA using Gauss RBF as Kernel

```

from sklearn.manifold import Isomap
# Input k nearest neighbors to form a weighted graph
k_opt=27

# path method refers to Dikshitra's algorithm
# p=2 represents computing of euclidean distance matrix between k nearest neighbors
isomap = Isomap(n_components=2, n_neighbors=k_opt, path_method = "D", p=2)
X_reduced_isomap = isomap.fit_transform(X)
```

Listing 5. Applying Isomap to dataset

```
X, y = make_circles(n_samples=1000, random_state=123, noise=0.1, factor=0.2)
from sklearn.metrics.pairwise import euclidean_distances
D = euclidean_distances(X)
# Set the distances values to 8 decimals to avoid asymmetric
D = np.around(D, decimals=8)
print("Is symmetric?", np.all(D==D.T))


# MDS result
X_mds = apply_MDS(D, k=2)
#the concentric circles, 2D
fig = plt.figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.scatter(X_mds[y==0, 0], np.zeros((500,1)), color='red', alpha=0.5)
plt.scatter(X_mds[y==1, 0], np.zeros((500,1)), color='blue', alpha=0.5)
plt.title('Concentric circles(1D) after mMDS', size=14)
plt.xlabel('Dimension 1', size=14)

plt.subplot(1,2,2)
plt.scatter(X_mds[y==0, 0], X_mds[y==0, 1], color='red', alpha=0.5)
plt.scatter(X_mds[y==1, 0], X_mds[y==1, 1], color='blue', alpha=0.5)

plt.title('Concentric circles(2D) after mMDS', size=14)
plt.xlabel('Dimension 1', size=14)
plt.ylabel('Dimension 2', size=14)
plt.show()


# k-PCA result
n_components = 2
gamma=1.25
X_pc, _= gaussRBF_kernel_pca(X, gamma, n_components)
#the concentric circles, 2D
fig = plt.figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.scatter(X_pc[y==0, 0], np.zeros((500,1)), color='red', alpha=0.5)
plt.scatter(X_pc[y==1, 0], np.zeros((500,1)), color='blue', alpha=0.5)
plt.title('Concentric circles(1D), GaussRBF k-PCA\n $\gamma$ = %.3f'%gamma, size=14)
```

```
plt.xlabel('Dimension 1', size=14)

plt.subplot(1,2,2)
plt.scatter(X_pc[y==0, 0], X_pc[y==0, 1], color='red', alpha=0.5)
plt.scatter(X_pc[y==1, 0], X_pc[y==1, 1], color='blue', alpha=0.5)

plt.title('Concentric circles(2D), GaussRBF k-PCA\n $\gamma$ = %.3f'%gamma, size=14)
plt.xlabel('Dimension 1', size=14)
plt.ylabel('Dimension 2', size=14)

plt.show()



# ISOMAP result
#k_opt = k_nn[np.argmin(error_list)]
k_opt=27

isomap = Isomap(n_components=2, n_neighbors=k_opt, path_method = "D", p=2)
X_reduced_isomap = isomap.fit_transform(X)
# the Swiss Roll, 2D
fig = plt.figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.scatter(X_reduced_isomap[y==0, 0], np.zeros((500,1)), color='red', alpha=0.5)
plt.scatter(X_reduced_isomap[y==1, 0], np.zeros((500,1)), color='blue', alpha=0.5)

plt.title('Concentric circles(1D), ISOMAP k = %d'%k_opt, size=15)
plt.xlabel('Dimension 1', size=14)

plt.subplot(1,2,2)
plt.scatter(X_reduced_isomap[y==0, 0], X_reduced_isomap[y==0, 1], color='red', alpha=0.5)
plt.scatter(X_reduced_isomap[y==1, 0], X_reduced_isomap[y==1, 1], color='blue', alpha=0.5)

plt.title('Concentric circles(2D), ISOMAP k = %d'%k_opt, size=15)
plt.xlabel('Dimension 1', size=14)
plt.ylabel('Dimension 2', size=14)

plt.show()
```

Listing 6. Methods applied to Concentric circles and along with thier embedding and visualization in first two dimensions

```

# MDS
from sklearn.metrics.pairwise import euclidean_distances
D = euclidean_distances(X)
# Set the distances values to 8 decimals to avoid asymmetric
D = np.around(D, decimals=8)
print("Is symmetric?", np.all(D==D.T))
X_mds = apply_MDS(D, k=2)
# the Swiss Roll, 2D
fig = plt.figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.scatter(X_mds[:, 0],np.zeros(X.shape[0]), c=t, cmap=plt.cm.jet)

plt.title('Swiss roll(1D) after mMDS', size=14)
plt.xlabel('Dimension 1', size=14)

plt.subplot(1,2,2)
plt.scatter(X_mds[:, 0], X_mds[:, 1], c=t, cmap=plt.cm.jet)

plt.title('Swiss roll(2D) after mMDS', size=14)
```

```
plt.xlabel('Dimension 1', size=14)
plt.ylabel('Dimension 2', size=14)
plt.show()



# Kernel-PCA
gamma=0.1
n_components = 2
X_pc, eig_vals_srtd = gaussRBF_kernel_pca(X, gamma, n_components)

# the Swiss Roll, 2D
fig = plt.figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.scatter(X_pc[:, 0],np.zeros(X.shape[0]), c=t, cmap=plt.cm.jet)

plt.title('Swiss roll(1D), GaussRBF k-PCA\n $\gamma$ = %.3f'%gamma , size=14)
plt.xlabel('Dimension 1', size=14)

plt.subplot(1,2,2)
plt.scatter(X_pc[:, 0], X_pc[:, 1], c=t, cmap=plt.cm.jet)

plt.title('Swiss roll(2D), GaussRBF k-PCA\n $\gamma$ = %.3f'%gamma , size=14)
plt.xlabel('Dimension 1', size=14)
plt.ylabel('Dimension 2', size=14)

plt.show()


# ISOMAP
from sklearn.manifold import Isomap

#k_opt = k_nn[np.argmin(error_list)]
k_opt = 15
isomap = Isomap(n_components=2, n_neighbors=k_opt, path_method = "D", p=2)
X_reduced_isomap = isomap.fit_transform(X)

# the Swiss Roll, 2D
fig = plt.figure(figsize=(13,5))
plt.subplot(1,2,1)
plt.scatter(X_reduced_isomap[:, 0], np.zeros(X.shape[0]), c=t, cmap=plt.cm.jet)

plt.title('Unfold Swiss roll(1D), ISOMAP k = %d'%k_opt, size=15)
plt.xlabel('Dimension 1', size=14)

plt.subplot(1,2,2)
plt.scatter(X_reduced_isomap[:, 0], X_reduced_isomap[:, 1], c=t, cmap=plt.cm.jet)
plt.title('Unfold Swiss roll(2D), ISOMAP k = %d'%k_opt, size=15)
plt.xlabel('Dimension 1', size=14)
plt.ylabel('Dimension 2', size=14)

plt.show()
```

Listing 7. Methods applied to Swiss roll and along with thier embedding and visualization in first two dimensions

```

# algorithms best estimate of the manifold
def compute_DM_kpca(X, gamma):
    from sklearn.metrics.pairwise import euclidean_distances

    # squared euclidean distance matrix
    mat_sq_dists = euclidean_distances(X, squared=True)
```

```
    # Compute the symmetric kernel matrix.
    K = np.exp(-gamma * mat_sq_dists)

    # ref: https://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture13.pdf
    # Euclidean distance in kernel space btw points phi(xi) and phi(xj)
    # ||phi(xi)-phi(xj)|| = K(xi,xi) +K(xj,xj) - 2K(xi,xj)
    D = np.zeros((K.shape[0], K.shape[1]))
    for i in range(K.shape[0]):
        for j in range(K.shape[1]):
            D[i,j] = K[i,i]+K[j,j]-2*K[i,j]

    return D

def compute_DM_mds(X):
    from sklearn.metrics.pairwise import euclidean_distances
    D = euclidean_distances(X)
    # Set the distances values to 8 decimals to avoid asymmetric
    D = np.around(D, decimals=4)
    return D


def compute_DM_isomap(X, n_neighbors):
    from scipy.sparse.csgraph import dijkstra
    from sklearn.neighbors import kneighbors_graph

    # Construct the weighted nearest neighbor graph
    D = kneighbors_graph(X, n_neighbors=n_neighbors, mode="distance").toarray()

    # Compute the shortest path betwen any given two nodes and output it as a distance matrix
    Dijk_D = dijkstra(D, directed=False, indices=None, unweighted=False)

    Dijk_D = np.around(Dijk_D, decimals=4)
    return Dijk_D


def compute_DX(X):
    from sklearn.metrics.pairwise import euclidean_distances
    D = euclidean_distances(X)
    # Set the distances values to 8 decimals to avoid asymmetric
    D = np.around(D, decimals=4)
    return D

def residual_variance(DM, DX):

    return 1-(np.corrcoef(DM.flatten(), DX.flatten())[0,1])**2
```

Listing 8. Computation of Residual Variance

```
resVar_mds = []
resVar_kpca = []
resVar_isomap = []

n_dimension = range(1,11)

for i in n_dimension:

    resVar_mds.append( residual_variance(  compute_DM_mds(X), compute_DX(X_mds[:,0:i])    ) )
    resVar_kpca.append( residual_variance(compute_DM_kpca(X, gamma), compute_DX(X_pc[:,0:i])) )
    resVar_isomap.append(  residual_variance(compute_DM_isomap(X,n_neighbors=k_opt),
                                compute_DX(X_reduced_isomap[:,0:i]) ) )

plt.figure(figsize=(7,5))
```

```
plt.plot(n_dimension, resVar_mds, marker="^",label="mds")
plt.plot(n_dimension, resVar_kpca, marker="s", mfc="None",label="kpca, $\gamma=%.3f$"%gamma)
plt.plot(n_dimension, resVar_isomap, marker="o", label="isomap, k_nn=%d"%k_opt, )
plt.legend(loc="best", prop={"size":14})
plt.ylabel("Residual variances", size=14)
plt.xlabel("Dimensions", size=14)
plt.title("Concentric circle", size=15)
plt.show()
```

Listing 9. Visualization of residual variances comparing the three methods