CS5489 - Machine Learning

Lecture 6d - Manifold Embedding

Prof. Antoni B. Chan

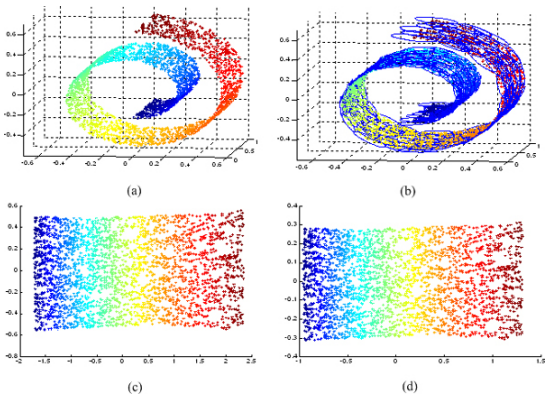
Dept. of Computer Science, City University of Hong Kong

Outline

1. Linear Dimensionality Reduction for Vectors
2. Linear Dimensionality Reduction for Text
3. Non-linear Dimensionality Reduction
4. **Manifold Embedding**

Manifold Embedding

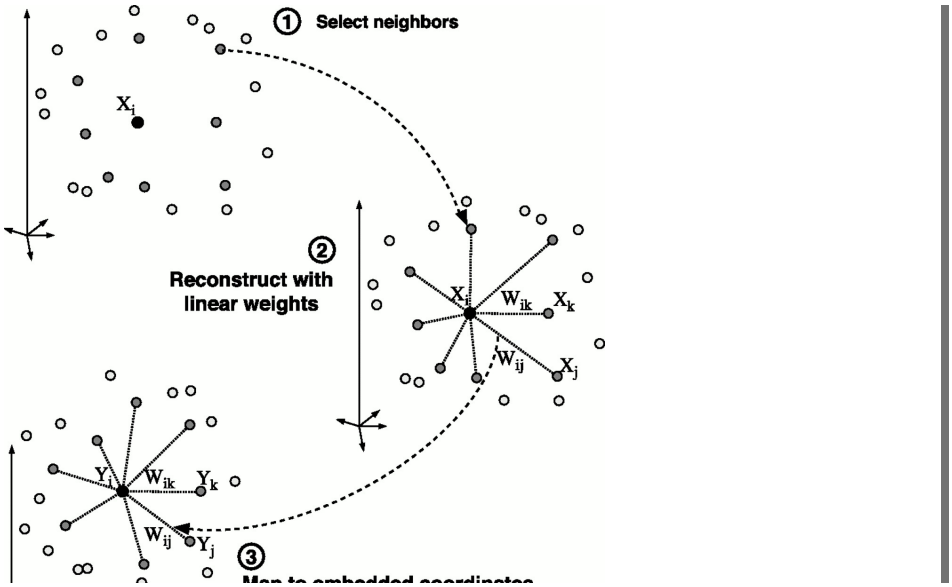
- Reduce high-dimensional data to 2 or 3 dimensions for visualization
- Try to preserve the inherent structure of the data
  - find a set of lower-dim points that optimize some criteria.
- Two types:
  - 1. preserve local neighborhood structure
    - assumes thata lies in a lower-dim manifold; unfold the manifold
  - 2. preserve pairwise distances (similarities) between points



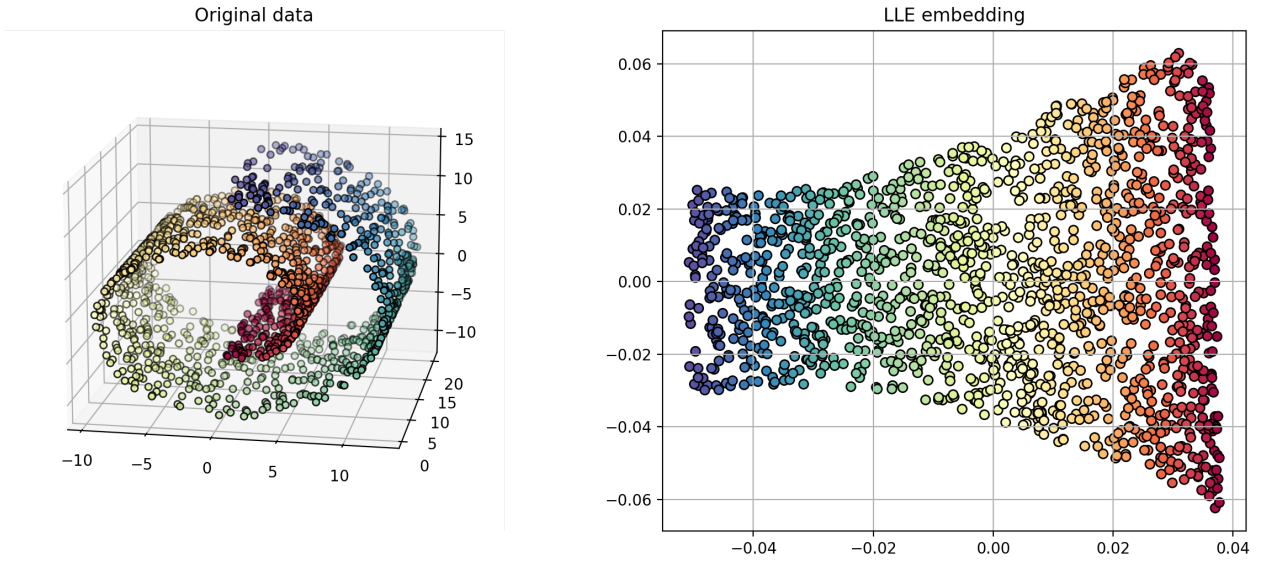
Locally-linear Embedding (LLE)

- **Idea:** preserve linearity within local neighborhoods defined by K nearest neighbors.
  - 1. a point $\mathbf{x}_i$ can be reconstructed by a linear combination of its neighbors N_i.
    - find the weights for the best reconstruction
      - $W^* = \text{argmin}_W \sum_i \|\mathbf{x}_i - \sum_{j \in N_i} w_{i,j}\mathbf{x}_j\|^2$
  - 2. the embedded point $\mathbf{y}_i$ should also have the same local linearity.
    - find the embedded points that best preserve the linearity
      - $Y^* = \text{argmin}_Y \sum_i \|\mathbf{y}_i - \sum_{j \in N_i} w_{i,j}\mathbf{y}_j\|^2$

LLE



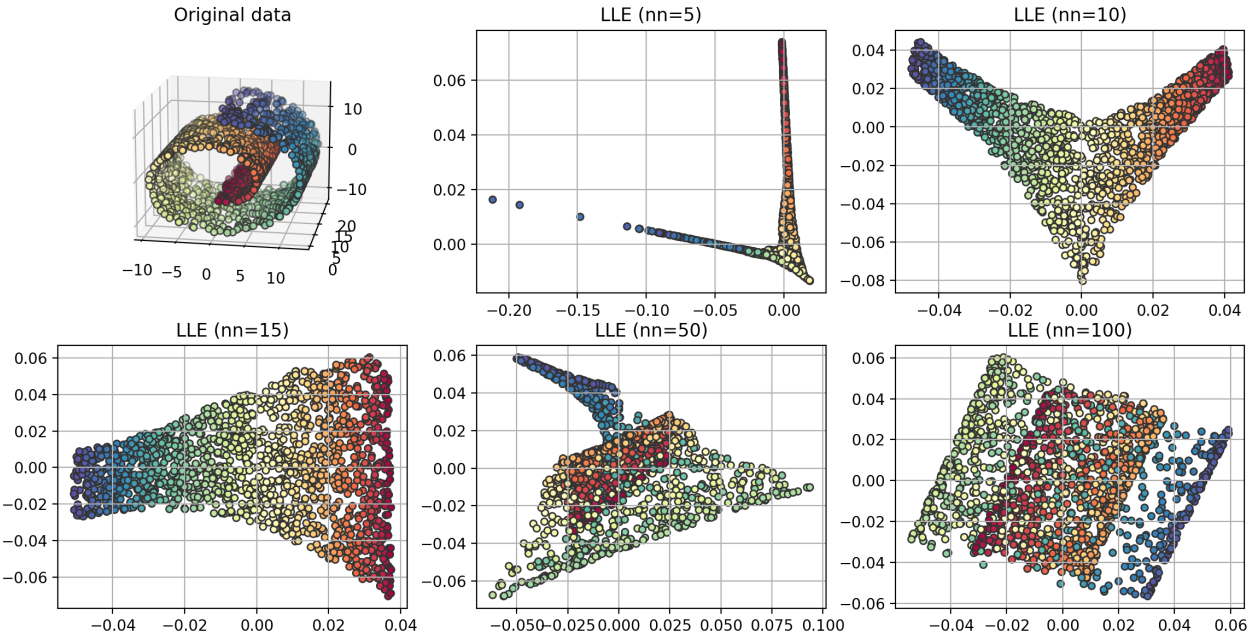
In [5]:
```
# n_neighbors = number of nearest neighbors to use for local neighborhood
# n_components = number of dimensions of manifold embedding
lle = manifold.LocallyLinearEmbedding(n_neighbors=12, n_components=2, random_state=121, n_jo
Xr = lle.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["LLE embedding"])
```



- Sensitive to the number of neighbors for defining the local region.

In [8]:
```
lfig
```

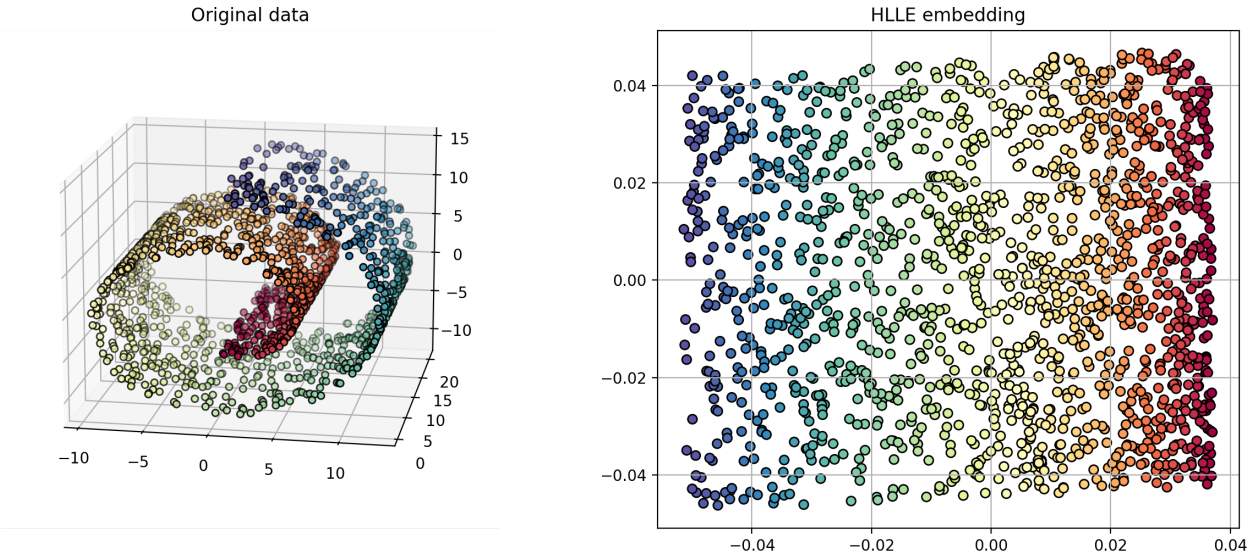
Out[8]:



Variants of LLE

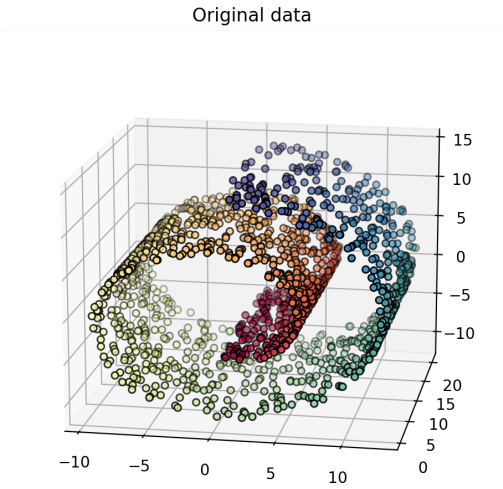
- **Hessian LLE (HLLE)**: LLE that also uses local curvature information

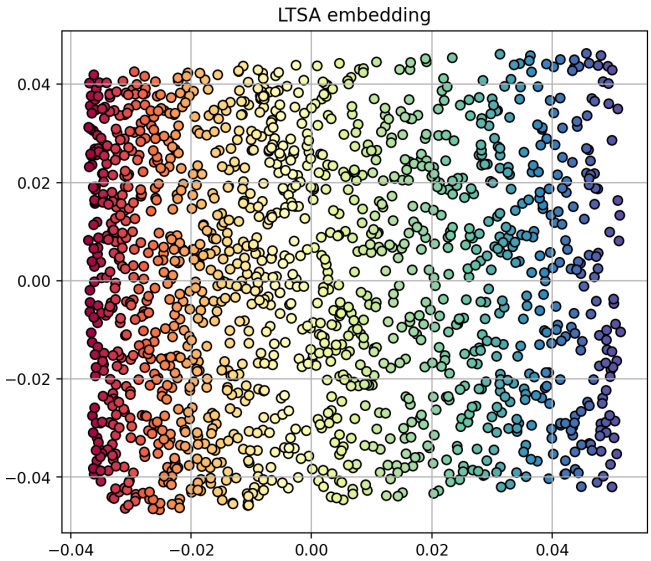
In [9]:
```
# set method to 'hessian'
hlle = manifold.LocallyLinearEmbedding(method='hessian', n_neighbors=12, n_components=2, ran
Xr = hlle.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["HLLE embedding"])
```



- **Local tangent space alignment (LTSA)**: rather than preserve distances, align local tangent spaces of the neighborhoods.

In [11]:
```
# set method to 'ltsa'
ltsa = manifold.LocallyLinearEmbedding(method='ltsa', n_neighbors=12, n_components=2, random
Xr = ltsa.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["LTSA embedding"])
```

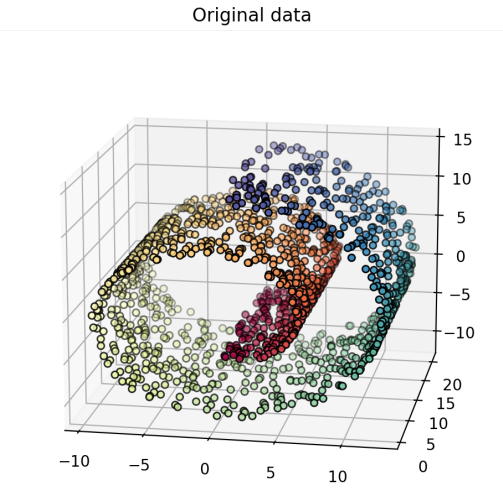


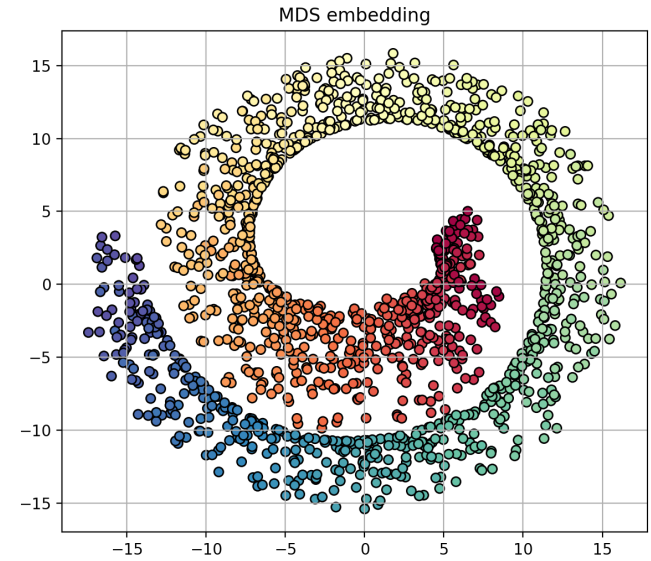


Multidimensional Scaling

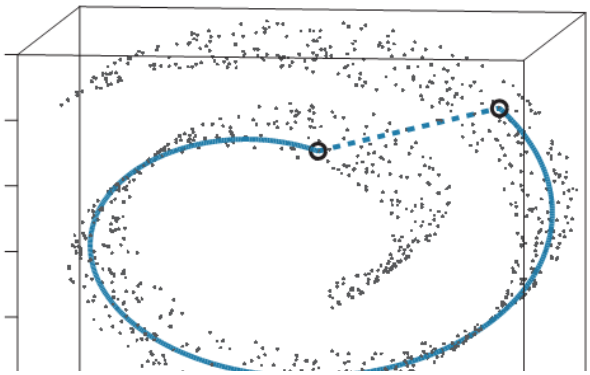
- **Idea:** find a low-dimensional embedding that preserves the pairwise distances between points in original high-dim space.
  - $Y^* = \text{argmin}_Y \sum_{i,j}(d(\mathbf{x}_i, \mathbf{x}_j) - d(\mathbf{y}_i, \mathbf{y}_j)))^2$
    - $\mathbf{x}_i$ are points in original space
    - $\mathbf{y}_i$ are points in the embedding space

In [13]:
```
mds = manifold.MDS(n_components=2, random_state=1234, n_jobs=-1)
Xr = mds.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["MDS embedding"])
```



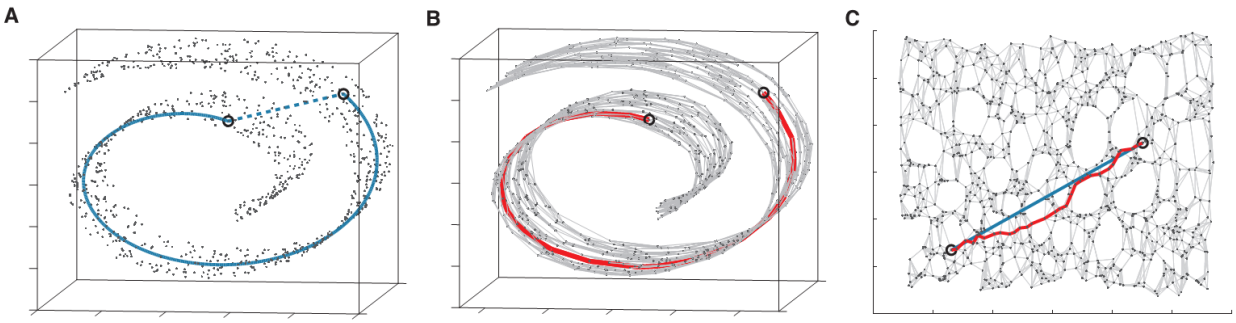


- **Problem**
  - MDS tries to preserve the Euclidean distance between the points.
  - For a manifold, two points may be far away along the manifold (geodesic distance), but close in Euclidean distance.
    - dashed line = Euclidean distance
    - solid line = Geodesic distance

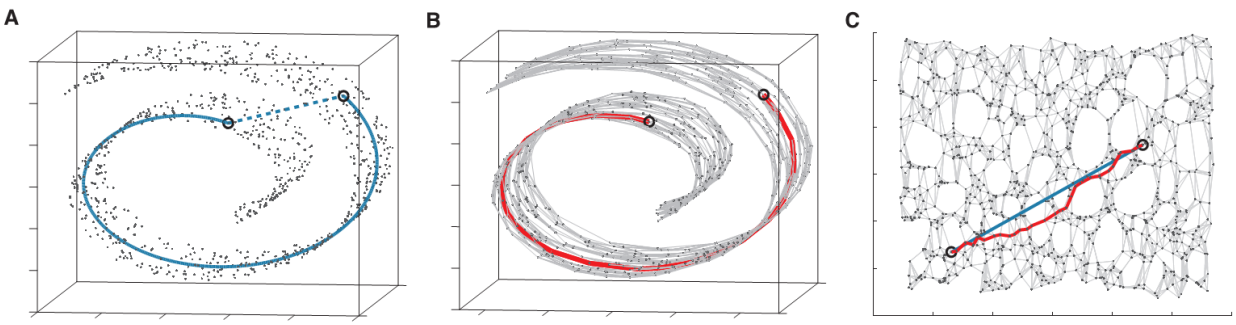


Isomap (Isometric Mapping)

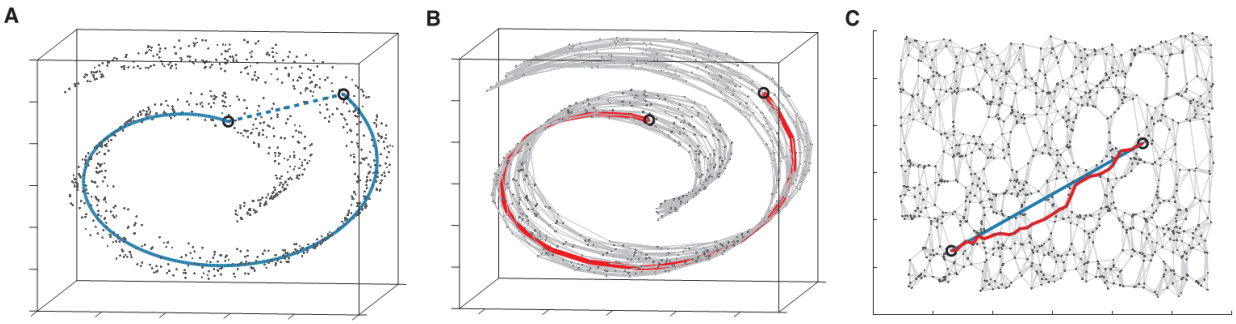
- Find embedding that preserves geodesic distances between points
  - geodesic distance = distances moving on the manifold (figure **A**)



- Approximate manifold by forming a graph (figure **B**)
  - each data point is a node
  - edge is added between nodes if they are K-nearest-neighbors.
  - calculate geodesic distance between 2 points using shortest-paths algorithm (Dijkstra's algorithm).

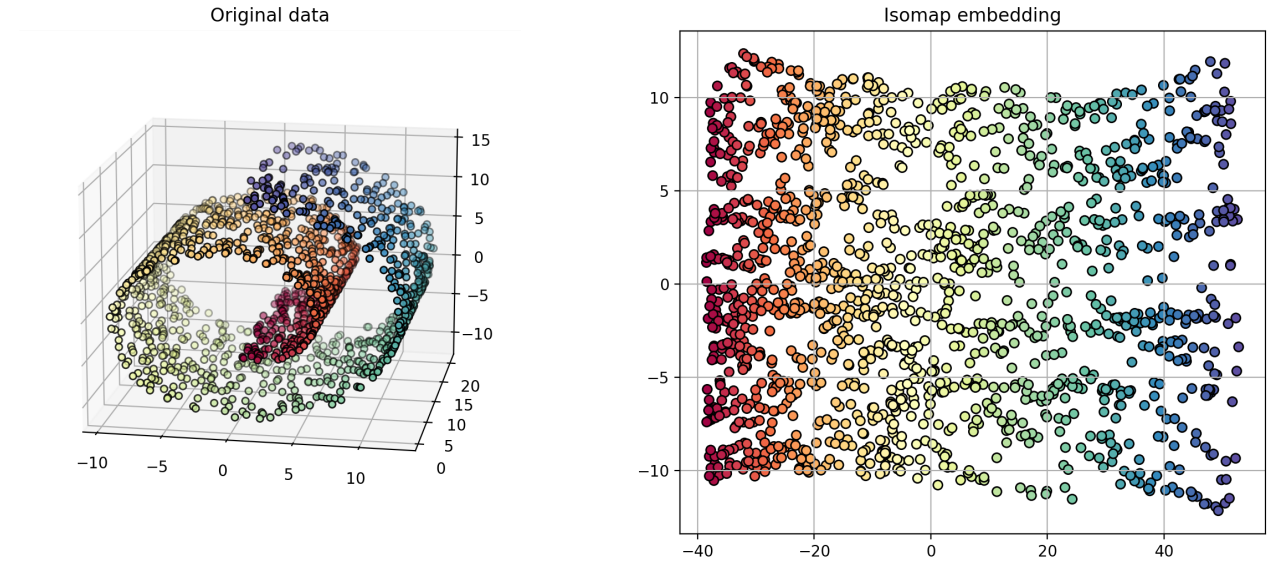


- Use MDS on the geodesic distances to calculate the embedding (figure **C**).



In [14]:
```
iso = manifold.Isomap(n_neighbors=12, n_components=2, n_jobs=-1)
Xr = iso.fit_transform(X)
```

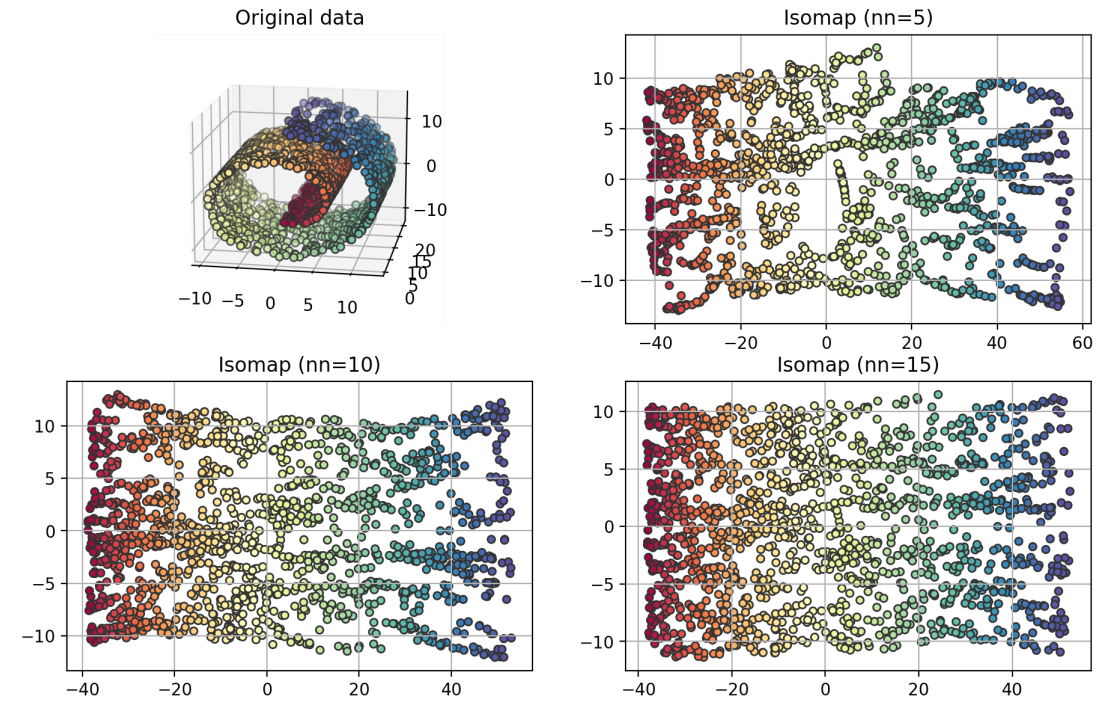
```
plt.figure(figsize=(15,6))
```



- Sensitive to number of neighbors
  - too few neighbors will put "holes" in the manifold

In [17]:
```
ifig
```

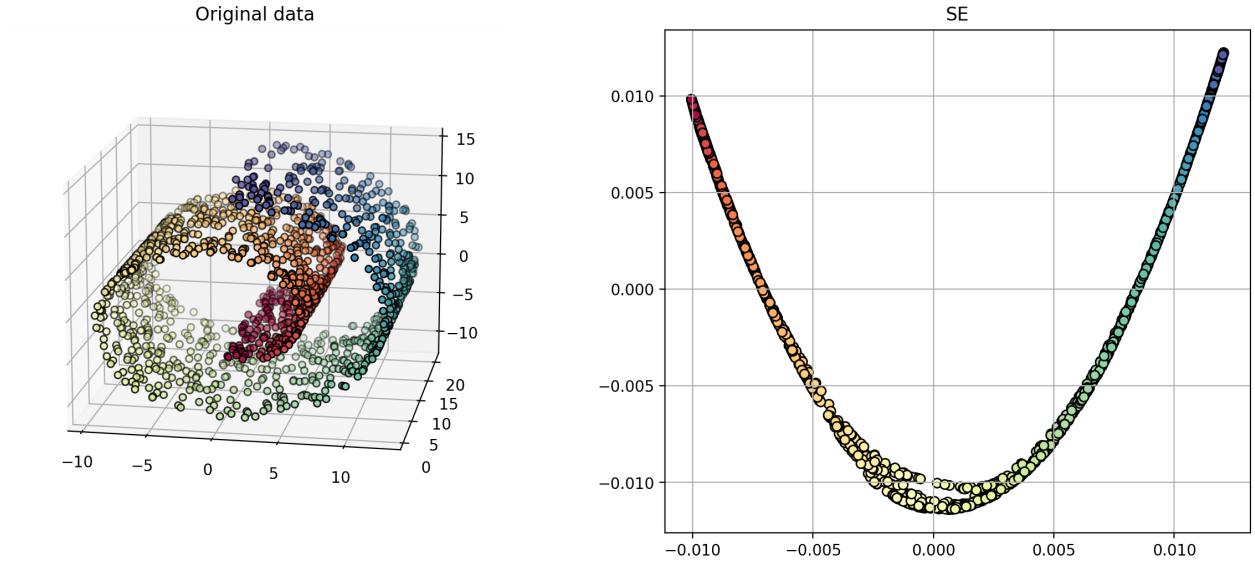
Out[17]:



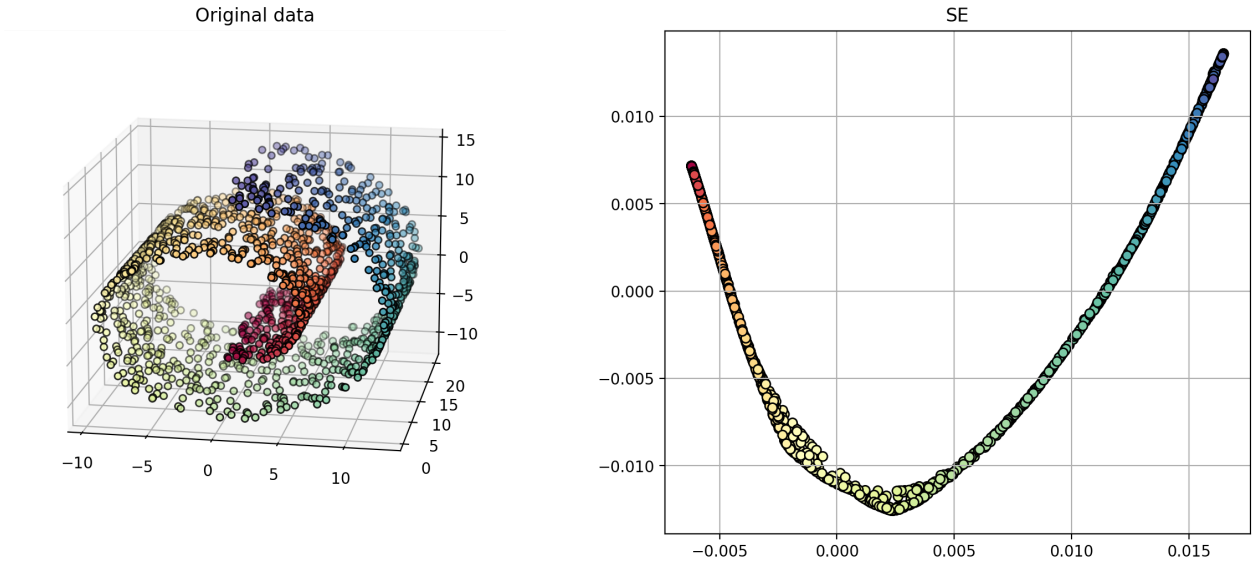
Spectral Embedding (Laplacian Eigenmaps)

- Preserve the neighborhood structure
  - 1. form an affinity (adjacency) matrix W
    - using K nearest neighbors
      - $W_{i,j} = \begin{cases} 1, & \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are neighbors} \\ 0, & \text{they are not neighbors} \end{cases}$.
    - using RBF kernel
      - $k(\mathbf{x}_i, \mathbf{x}_j)$ represents how close a neighbor.
  - 2. keep neighboring points close together in the embedding.
    - higher weight $W_{i,j}$ on points that are close together in original space.
      - $Y^* = \text{argmin}_Y \sum_{i,j} W_{i,j}(y_i - y_j)^2$

In [18]:
```
# using nearest-neighbors for affinity
spe = manifold.SpectralEmbedding(n_components=2, affinity='nearest_neighbors',
                                 random_state=1234, n_neighbors=12, n_jobs=-1)
Xr = spe.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["SE"])
```



In [20]:
```
# using RBF kernel for affinity
# gamma is the inverse bandwidth
spe = manifold.SpectralEmbedding(n_components=2, affinity='rbf',
                                 gamma=0.2, random_state=1234)
Xr = spe.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["SE"])
```



t-Distributed Stochastic Neighbor Embedding (t-SNE)

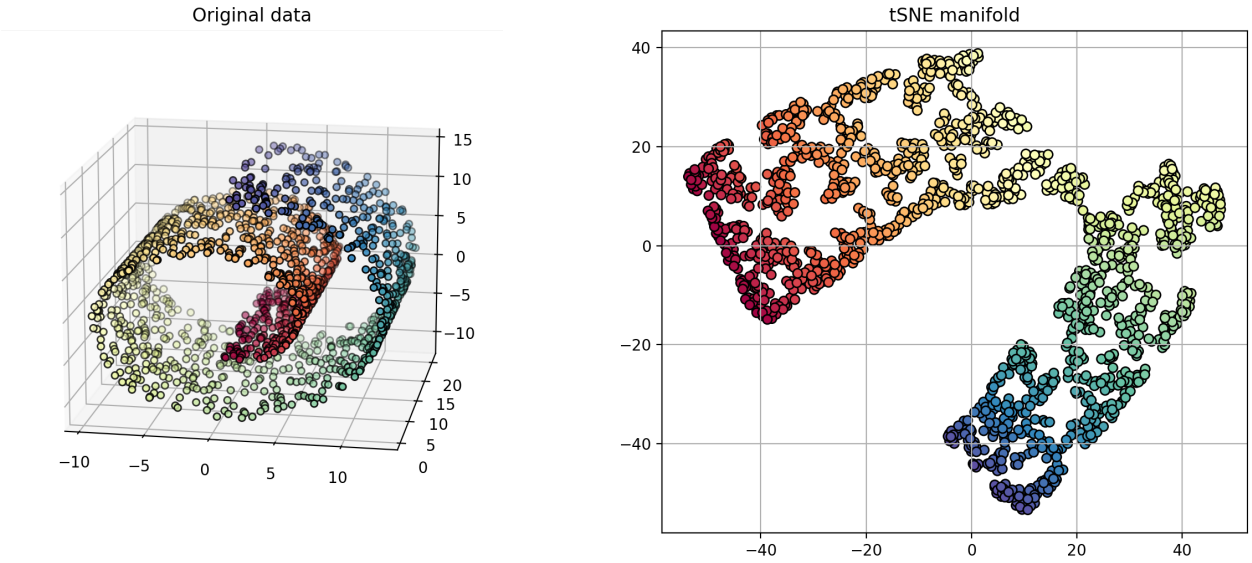
- Preserve pairwise similarities
  - 1. treat similarities in original space as probabilities
    - the probability that j is a neighbor of i: $p_{j|i} = \frac{k_i(\mathbf{x}_i,\mathbf{x}_j)}{\sum_{k\neq i} k_i(\mathbf{x}_i,\mathbf{x}_k)}$
      - $k_i(\mathbf{x}_i, \mathbf{x}_j)$ is the RBF kernel with inverse bandwidth γ_i.
    - the similarity between i and j: $p_{i,j} = \frac{p_{j|i}+p_{i|j}}{2N}$
  - 2. similarities in the embedding space (student-t distribution)

    - the probability that i and j are neighbors: $q_{i,j} = \frac{(1+\|\mathbf{y}_i - \mathbf{y_j}\|^2)^{-1}}{\sum_{k \neq l}(1+\|\mathbf{y}_l - \mathbf{y_k}\|^2)^{-1}}$
  - 3. find the embedding points that preserve the probability structure:
    - $Y = \text{argmin}_Y \sum_{i \neq j} p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}$
    - the objective is the Kullback-Leibler (KL) divergence, which is a measure of similarity between two

- Tends to group together similar items
  - student-t distribution is heavy-tailed
  - "moderate" distances in original space are converted to "large" distances in embedding space

In [22]:
```
# perplexity = similar to number of neighbors
tsne = manifold.TSNE(n_components=2, perplexity=30.0, random_state=11)
Xr = tsne.fit_transform(X)

plt.figure(figsize=(15,6))
plot_manifolds(X, Y, [Xr], ["tSNE manifold"])
```

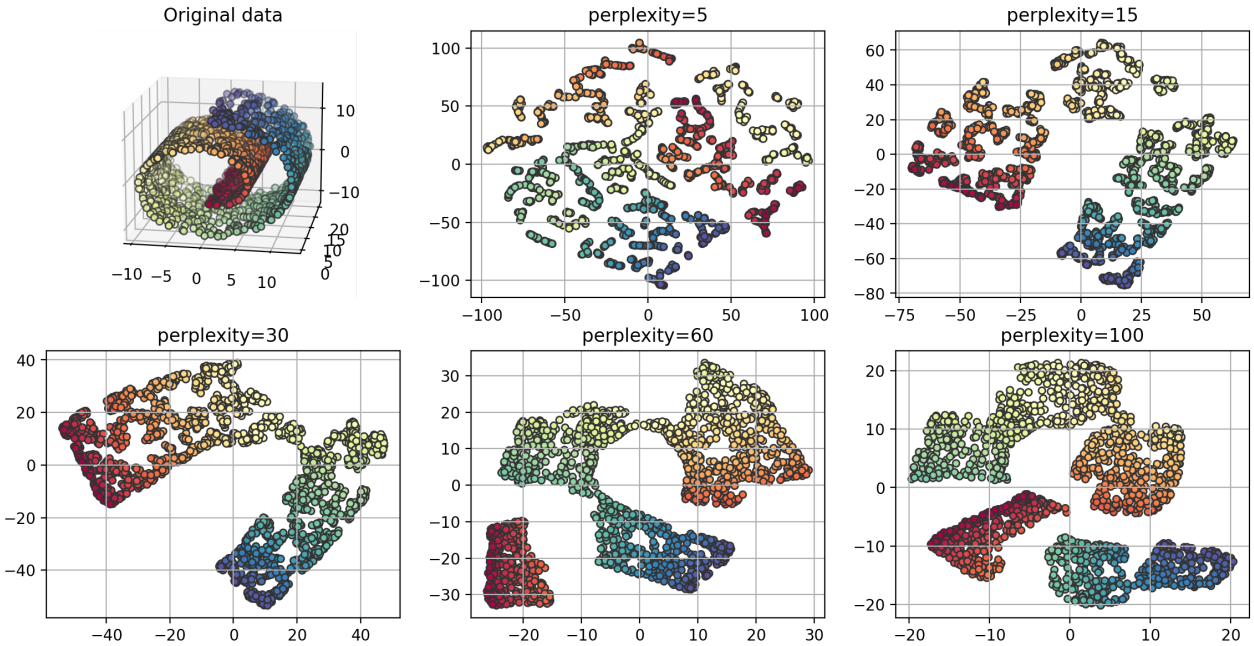
```
/Users/abc/miniforge3/envs/py39np/lib/python3.9/site-packages/sklearn/manifold/_t_
sne.py:800: FutureWarning: The default initialization in TSNE will change from 'ra
ndom' to 'pca' in 1.2.
  warnings.warn(
/Users/abc/miniforge3/envs/py39np/lib/python3.9/site-packages/sklearn/manifold/_t_
sne.py:810: FutureWarning: The default learning rate in TSNE will change from 200.
0 to 'auto' in 1.2.
  warnings.warn(
```



- The `perplexity` parameter controls the number of groups (size of groups).
  - small perplexity: form more groups
  - large perplexity: form less groups

In [25]:
```
tsnefig
```

Out[25]:

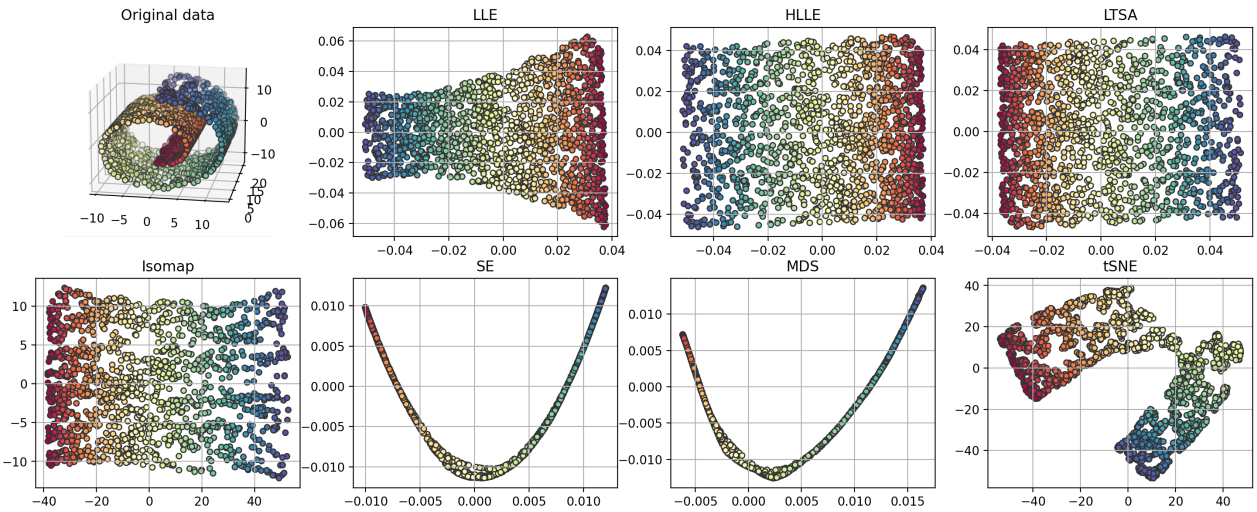


Comparison of Methods

- Preserves local neighborhoods: LLE, HLLE, TLSA
- Preserves geodesic distance: Isomap, SE
- Preserves distances/similarities: MDS, t-SNE

In [27]:
```
fig
```

Out[27]:

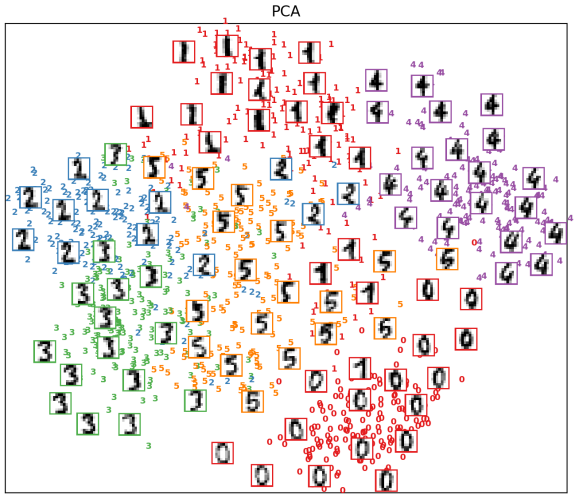


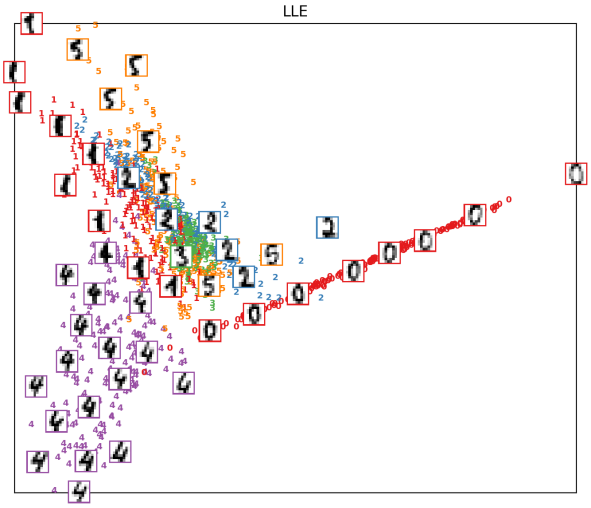
Digit Images

- Digits 0-5
- Embed 8x8 image (64D vector) into 2-dimensional space for visualization

In [31]:
```
dfig[0]
```

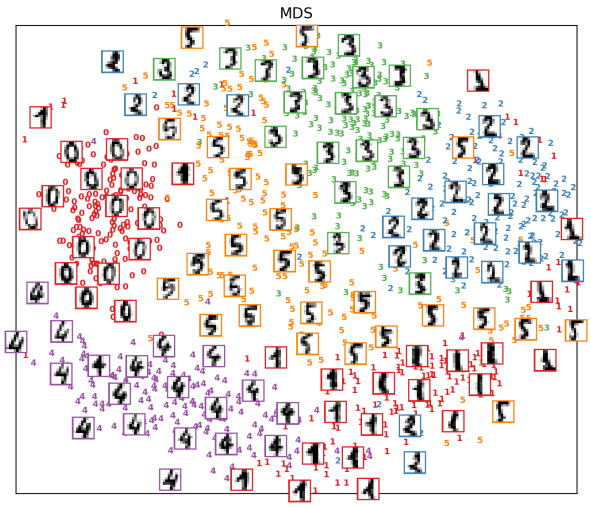
Out[31]:

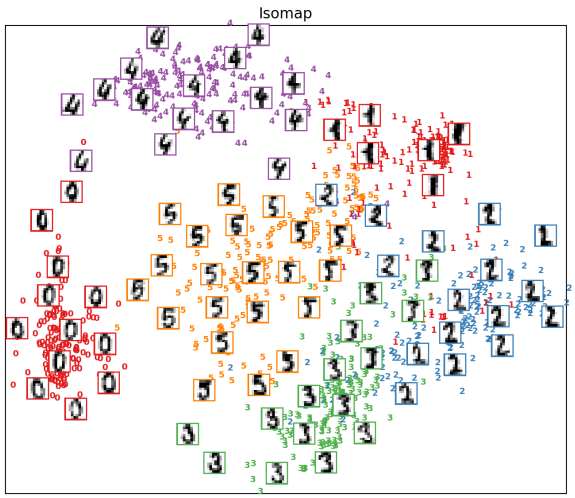




In [32]:
```
dfig[1]
```

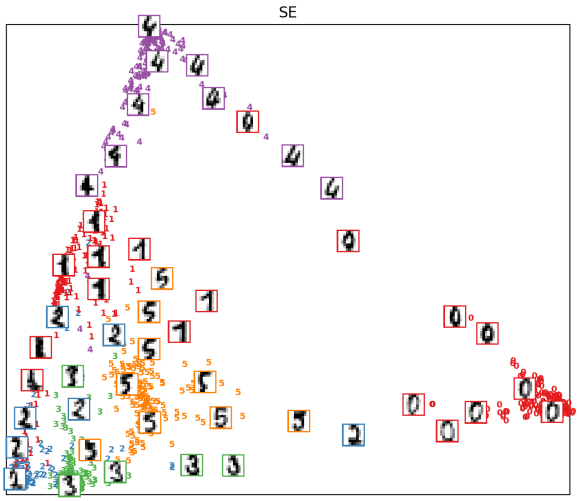
Out[32]:

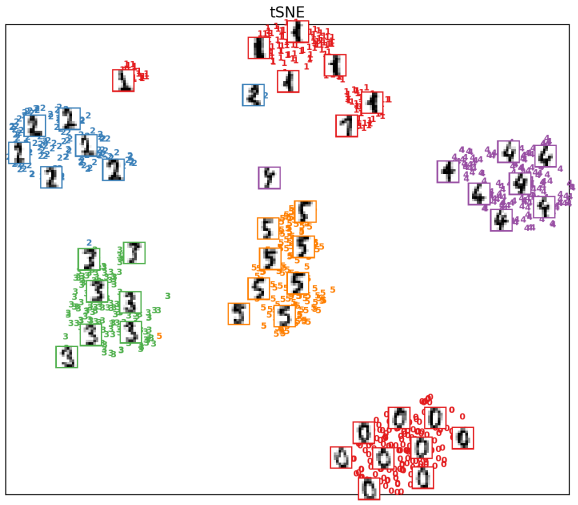




In [33]:
```
dfig[2]
```

Out[33]:





Non-Linear Dimensionality Reduction - Summary

- **Goal:** given high-dim data, find a low-dim representation.
  - try to preserve inherent structure of data
- Two types of approaches:
  - *Non-linear dimensionality reduction*
    - calculate low-dim coefficients using non-linear projections (kernel).
    - learn mapping from input space to low-dim space.

- *Manifold embedding*
  - Optimize over the embedding (low-dim) points to preserve some property from the high-dim points.
  - After training, manifold embedding methods cannot transform a novel (new) point.
    - need to re-train the whole thing

Name	Objective	Advantages	Disadvantages
Kernel principal component analysis (KPCA)	PCA in high-dim feature space (kernel trick)	- can transform novel data	- need to calculate kernel matrix; - need to keep training data around for embedding new points.
Locally-Linear Embedding (LLE)	preserve local neighborhood distances	- good for single low-dim manifold	- cannot embed a novel point.
Hessian LLE	w/ local curvature information.	- good for single low-dim manifold	- cannot embed a novel point.
Local tangent space alignment (LTSA)	align tangent spaces	- good for single low-dim manifold	- cannot embed a novel point.
Multi-dimensional scaling (MDS)	preserve pairwise distances	- good for viewing the space.	- cannot embed a novel point.
Isometric mapping (Isomap)	preserve geodesic distances	- good for single low-dim manifold	- cannot embed a novel point.
Spectral embedding (SE)	"PCA" on graph representation	- good for single low-dim manifold	- cannot embed a novel point.
t-distributed Stochastic Neighbor Embedding (t-SNE)	preserve pairwise similarities	- can discover similar items	- cannot embed a novel point.