Equation 8-4. LLE step 1: linearly modeling local relationships

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{m} \| \mathbf{x}^{(i)} - \sum_{j=1}^{m} w_{i,j} \mathbf{x}^{(j)} \|^2$$

$$\text{subject to} \begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^{m} w_{i,j} = 1 & \text{for } i = 1, 2, \cdots, m \end{cases}$$

After this step, the weight matrix $\widehat{\mathbf{W}}$ (containing the weights $\widehat{w}_{i,j}$) encodes the local linear relationships between the training instances. Now the second step is to map the training instances into a d-dimensional space (where $d < n$) while preserving these local relationships as much as possible. If $\mathbf{z}^{(i)}$ is the image of $\mathbf{x}^{(i)}$ in this d-dimensional space, then we want the squared distance between $\mathbf{z}^{(i)}$ and $\sum_{j=1}^{m} \widehat{w}_{i,j} \mathbf{z}^{(j)}$ to be as small as possible. This idea leads to the unconstrained optimization problem described in Equation 8-5. It looks very similar to the first step, but instead of keeping the instances fixed and finding the optimal weights, we are doing the reverse: keeping the weights fixed and finding the optimal position of the instances' images in the low-dimensional space. Note that $\mathbf{Z}$ is the matrix containing all $\mathbf{z}^{(i)}$.

Equation 8-5. LLE step 2: reducing dimensionality while preserving relationships

$$\widehat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_{i=1}^{m} \| \mathbf{z}^{(i)} - \sum_{j=1}^{m} \widehat{w}_{i,j} \mathbf{z}^{(j)} \|^2$$

Scikit-Learn's LLE implementation has the following computational complexity: $O(m \log(m) n \log(k))$ for finding the k nearest neighbors, $O(mnk^3)$ for optimizing the weights, and $O(dm^2)$ for constructing the low-dimensional representations. Unfortunately, the m^2 in the last term makes this algorithm scale poorly to very large datasets.

Other Dimensionality Reduction Techniques

There are many other dimensionality reduction techniques, several of which are available in Scikit-Learn. Here are some of the most popular:

- *Multidimensional Scaling* (MDS) reduces dimensionality while trying to preserve the distances between the instances (see Figure 8-13).

- *Isomap* creates a graph by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the *geodesic distances*[9] between the instances.
- *t-Distributed Stochastic Neighbor Embedding* (t-SNE) reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space (e.g., to visualize the MNIST images in 2D).
- *Linear Discriminant Analysis* (LDA) is actually a classification algorithm, but during training it learns the most discriminative axes between the classes, and these axes can then be used to define a hyperplane onto which to project the data. The benefit is that the projection will keep classes as far apart as possible, so LDA is a good technique to reduce dimensionality before running another classification algorithm such as an SVM classifier.

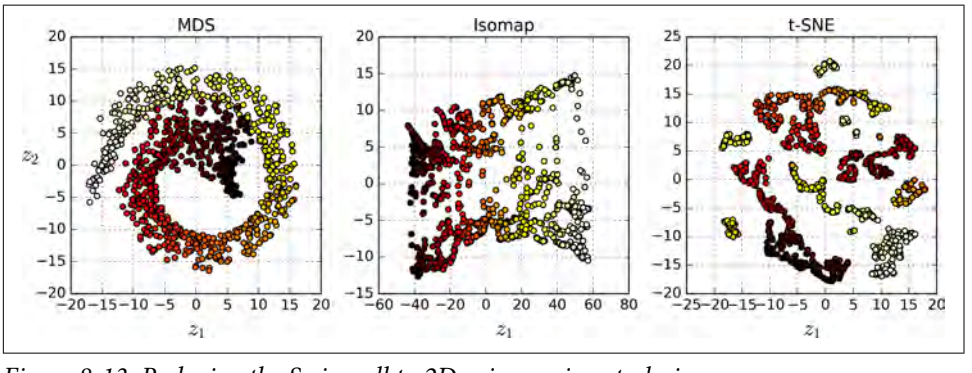

Figure 8-13. Reducing the Swiss roll to 2D using various techniques

Exercises

1. What are the main motivations for reducing a dataset's dimensionality? What are the main drawbacks?
2. What is the curse of dimensionality?
3. Once a dataset's dimensionality has been reduced, is it possible to reverse the operation? If so, how? If not, why?
4. Can PCA be used to reduce the dimensionality of a highly nonlinear dataset?
5. Suppose you perform PCA on a 1,000-dimensional dataset, setting the explained variance ratio to 95%. How many dimensions will the resulting dataset have?

9 The geodesic distance between two nodes in a graph is the number of nodes on the shortest path between these nodes.