

- *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.

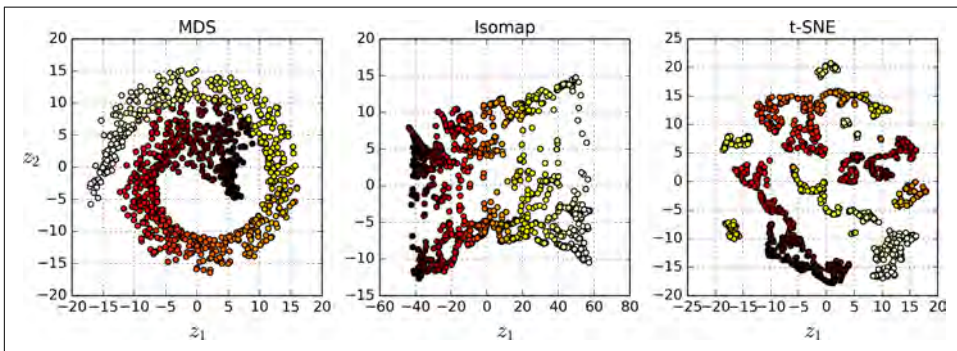


Figure 8-13. Reducing the Swiss roll to 2D using various techniques

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.

6. In what cases would you use vanilla PCA, Incremental PCA, Randomized PCA, or Kernel PCA?
7. How can you evaluate the performance of a dimensionality reduction algorithm on your dataset?
8. Does it make any sense to chain two different dimensionality reduction algorithms?
9. Load the MNIST dataset (introduced in [Chapter 3](#)) and split it into a training set and a test set (take the first 60,000 instances for training, and the remaining 10,000 for testing). Train a Random Forest classifier on the dataset and time how long it takes, then evaluate the resulting model on the test set. Next, use PCA to reduce the dataset's dimensionality, with an explained variance ratio of 95%. Train a new Random Forest classifier on the reduced dataset and see how long it takes. Was training much faster? Next evaluate the classifier on the test set: how does it compare to the previous classifier?
10. Use t-SNE to reduce the MNIST dataset down to two dimensions and plot the result using Matplotlib. You can use a scatterplot using 10 different colors to represent each image's target class. Alternatively, you can write colored digits at the location of each instance, or even plot scaled-down versions of the digit images themselves (if you plot all digits, the visualization will be too cluttered, so you should either draw a random sample or plot an instance only if no other instance has already been plotted at a close distance). You should get a nice visualization with well-separated clusters of digits. Try using other dimensionality reduction algorithms such as PCA, LLE, or MDS and compare the resulting visualizations.

Solutions to these exercises are available in [Appendix A](#).

PART II

Neural Networks and Deep Learning