

Download from finelybook www.finelybook.com

You may be wondering how to perform this reconstruction. One solution is to train a supervised regression model, with the projected instances as the training set and the original instances as the targets. Scikit-Learn will do this automatically if you set `fit_inverse_transform=True`, as shown in the following code:⁷

```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.0433,  
                    fit_inverse_transform=True)  
X_reduced = rbf_pca.fit_transform(X)  
X_preimage = rbf_pca.inverse_transform(X_reduced)
```



By default, `fit_inverse_transform=False` and `KernelPCA` has no `inverse_transform()` method. This method only gets created when you set `fit_inverse_transform=True`.

You can then compute the reconstruction pre-image error:

```
>>> from sklearn.metrics import mean_squared_error  
>>> mean_squared_error(X, X_preimage)  
32.786308795766132
```

Now you can use grid search with cross-validation to find the kernel and hyperparameters that minimize this pre-image reconstruction error.

LLE

Locally Linear Embedding (LLE)⁸ is another very powerful *nonlinear dimensionality reduction* (NLDR) technique. It is a Manifold Learning technique that does not rely on projections like the previous algorithms. In a nutshell, LLE works by first measuring how each training instance linearly relates to its closest neighbors (c.n.), and then looking for a low-dimensional representation of the training set where these local relationships are best preserved (more details shortly). This makes it particularly good at unrolling twisted manifolds, especially when there is not too much noise.

For example, the following code uses Scikit-Learn's `LocallyLinearEmbedding` class to unroll the Swiss roll. The resulting 2D dataset is shown in [Figure 8-12](#). As you can see, the Swiss roll is completely unrolled and the distances between instances are locally well preserved. However, distances are not preserved on a larger scale: the left part of the unrolled Swiss roll is squeezed, while the right part is stretched. Nevertheless, LLE did a pretty good job at modeling the manifold.

⁷ Scikit-Learn uses the algorithm based on Kernel Ridge Regression described in Gokhan H. Bakır, Jason Weston, and Bernhard Scholkopf, “*Learning to Find Pre-images*” (Tubingen, Germany: Max Planck Institute for Biological Cybernetics, 2004).

⁸ “Nonlinear Dimensionality Reduction by Locally Linear Embedding,” S. Roweis, L. Saul (2000).

```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)  
X_reduced = lle.fit_transform(X)
```

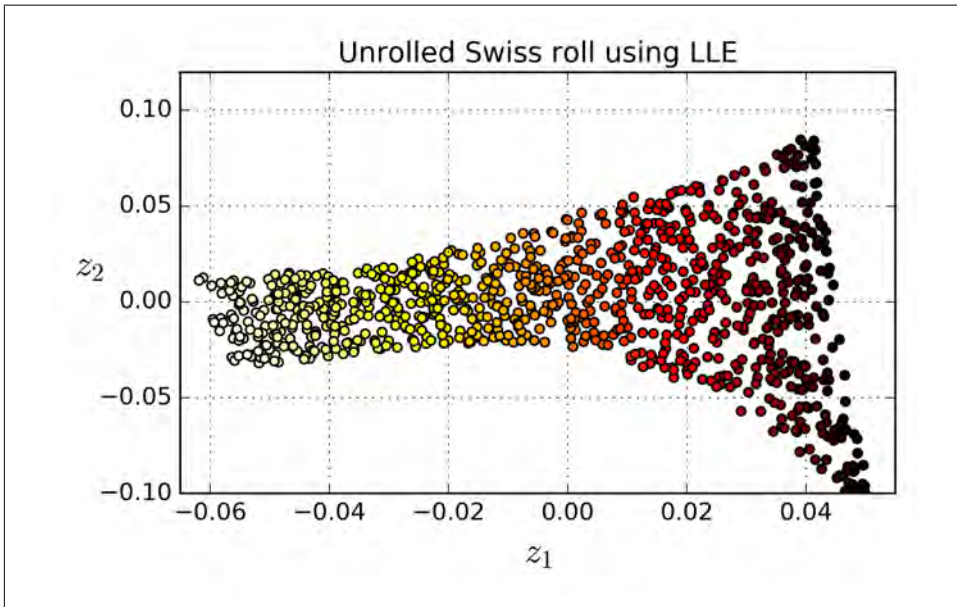


Figure 8-12. Unrolled Swiss roll using LLE

Here's how LLE works: first, for each training instance $\mathbf{x}^{(i)}$, the algorithm identifies its k closest neighbors (in the preceding code $k = 10$), then tries to reconstruct $\mathbf{x}^{(i)}$ as a linear function of these neighbors. More specifically, it finds the weights $w_{i,j}$ such that the squared distance between $\mathbf{x}^{(i)}$ and $\sum_{j=1}^m w_{i,j} \mathbf{x}^{(j)}$ is as small as possible, assuming $w_{i,j} = 0$ if $\mathbf{x}^{(j)}$ is not one of the k closest neighbors of $\mathbf{x}^{(i)}$. Thus the first step of LLE is the constrained optimization problem described in Equation 8-4, where \mathbf{W} is the weight matrix containing all the weights $w_{i,j}$. The second constraint simply normalizes the weights for each training instance $\mathbf{x}^{(i)}$.

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

$$\begin{aligned} \widehat{\mathbf{W}} = \operatorname{argmin}_{\mathbf{W}} \sum_{i=1}^m \left\| \mathbf{x}^{(i)} - \sum_{j=1}^m w_{i,j} \mathbf{x}^{(j)} \right\|^2 \\ \text{subject to } \begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^m w_{i,j} = 1 & \text{for } i = 1, 2, \dots, m \end{cases} \end{aligned}$$

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

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

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](#)).