CHAPTER 8

Dimensionality Reduction

Many modern Machine Learning problems involve extremely high-dimensional feature spaces—sometimes thousands or even millions of features per instance. This creates two major challenges: training becomes computationally expensive, and learning becomes statistically harder due to the **curse of dimensionality**. As dimensionality grows, data points tend to become far apart and the dataset becomes sparse, meaning predictions rely on larger extrapolations and models become more prone to overfitting. Dimensionality reduction aims to address this by reducing the number of features while retaining as much useful information as possible. However, this usually introduces some information loss (similar to lossy compression), may slightly reduce predictive performance, and adds complexity to the pipeline. Therefore, it is often recommended to attempt training on the original data first, and only apply dimensionality reduction when training time, memory, or visualization needs justify it.

The Curse of Dimensionality

High-dimensional space behaves in ways that conflict with everyday intuition. For example, in very high dimensions most points lie near the “border” of the space, and the average distance between two random points inside a unit hypercube grows dramatically as dimensions increase. As a result, data becomes sparse: neighbors are far away, local structure is harder to exploit, and algorithms (especially those relying on distances, like k-NN) can become less reliable. A theoretical fix is to gather exponentially more data to maintain density, but that quickly becomes infeasible in real-world settings.

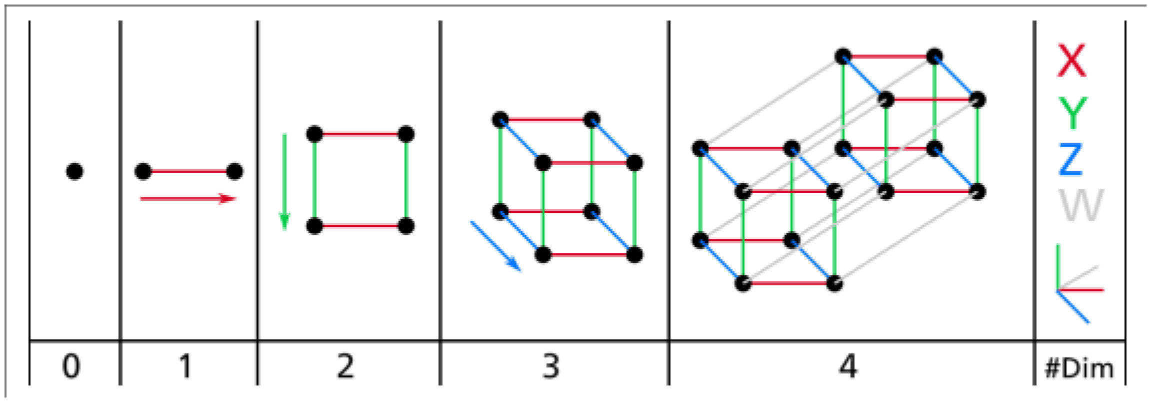

Figure 8-1. Point, segment, square, cube, and tesseract (0D to 4D hypercubes)

Main Approaches for Dimensionality Reduction

Dimensionality reduction methods typically fall into two broad categories: **projection** and **manifold learning**. Projection assumes the data lies near a lower-dimensional linear subspace and simply projects it down, while manifold learning assumes the data lies on a lower-dimensional surface that may be curved or twisted within a higher-dimensional space, requiring a nonlinear “unfolding” rather than a simple linear projection.

Projection

In many real datasets, features are redundant: some are almost constant, and others are strongly correlated. This implies the data often lies close to a lower-dimensional subspace even if it is represented in a high-dimensional space. Projection reduces dimensionality by mapping each point onto that lower-dimensional subspace. A simple example is projecting 3D data that lies near a plane down into 2D by dropping the perpendicular distance to the plane; the result preserves much of the data's structure while reducing dimensionality.

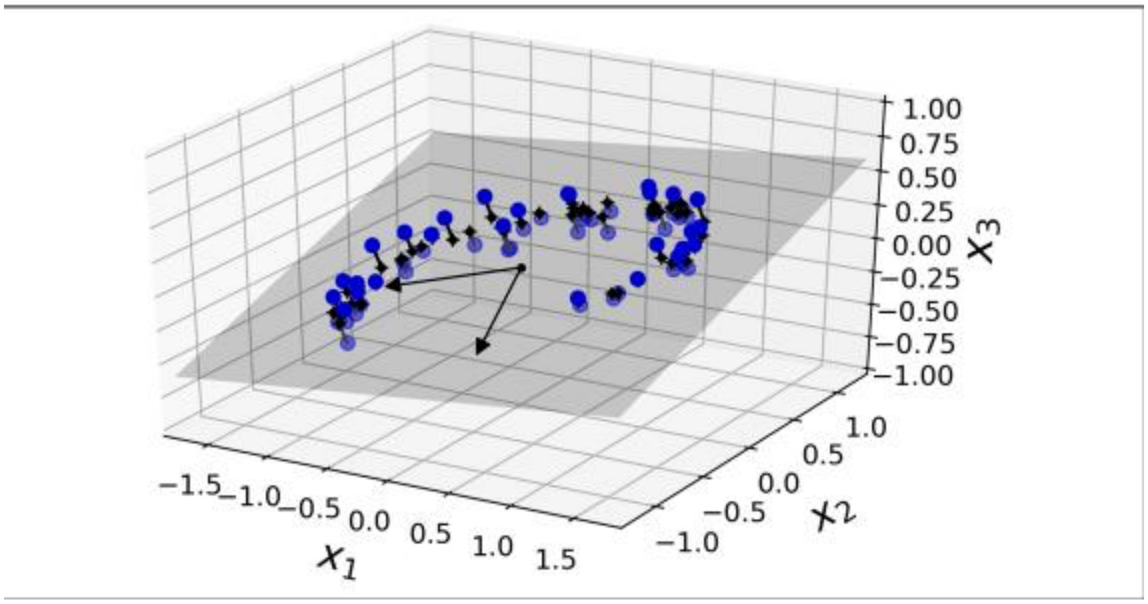

Figure 8-2. A 3D dataset lying close to a 2D subspace

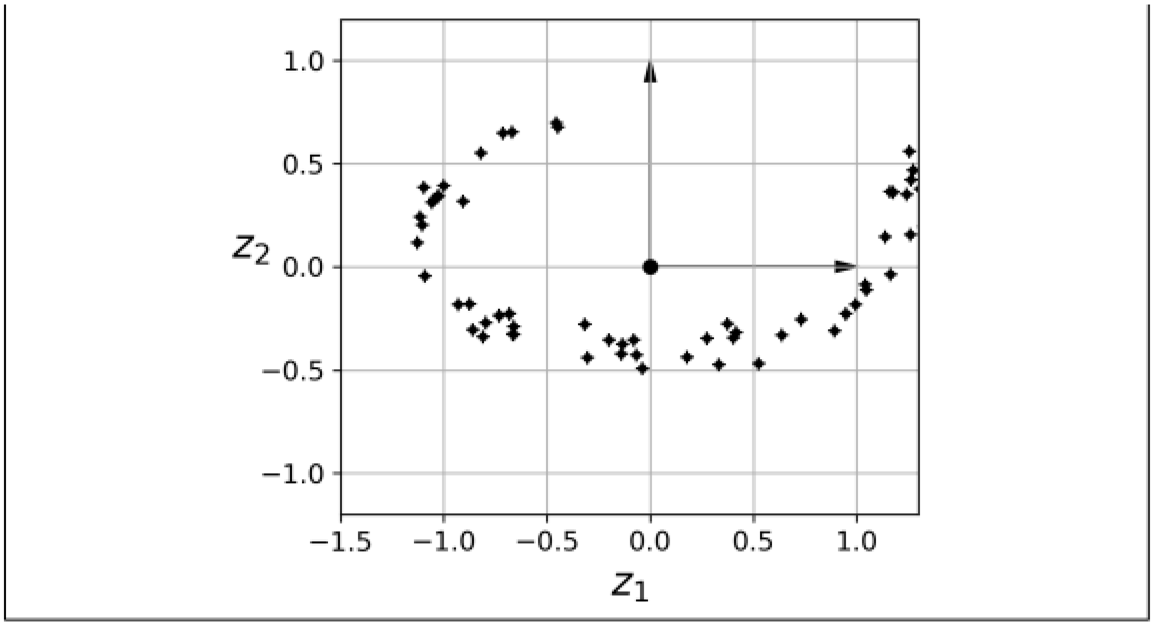

Figure 8-3. The new 2D dataset after projection

Manifold Learning
Projection can fail when the lower-dimensional structure is **nonlinear**, such as the classic **Swiss roll** dataset. A linear projection (e.g., dropping one axis) can “squash” different layers together and destroy meaningful geometry. Manifold learning addresses this by assuming the data lies on a lower-dimensional manifold that is curved in the original space, and the goal is to recover a representation that respects the manifold structure—effectively “unrolling” the data. This is motivated by the **manifold hypothesis**, which suggests that many real-world datasets (such as images) occupy only a small subset of the full high-dimensional space due to structural constraints.
A practical caution is that reducing dimensionality does not always simplify the downstream learning problem: sometimes a decision boundary that is simple in the original space becomes more complex in the reduced representation, depending on how the target concept aligns with the manifold.

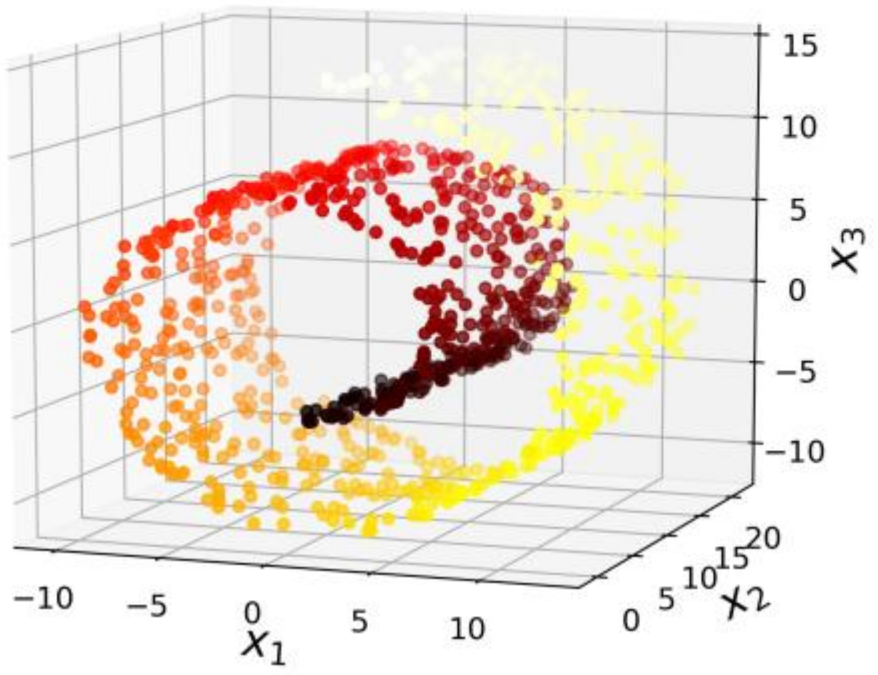

Figure 8-4. Swiss roll dataset

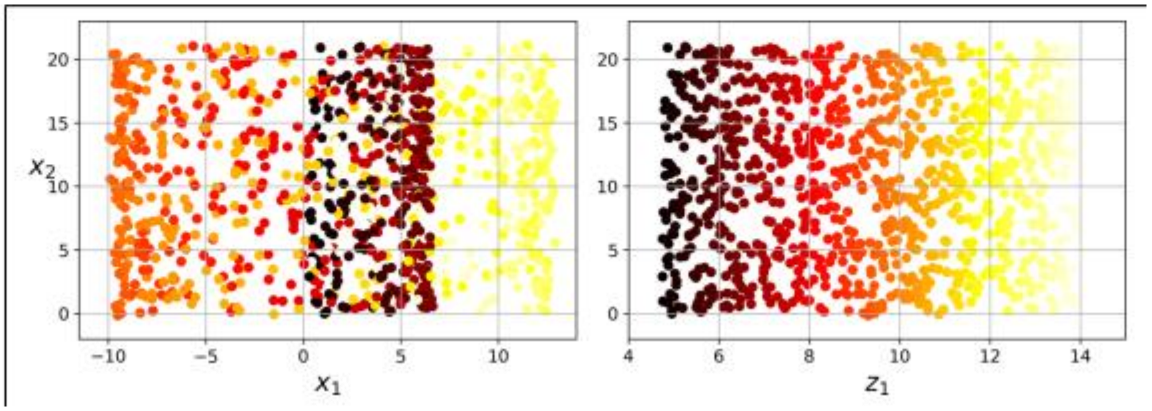

Figure 8-5. Squashing by projecting onto a plane (left) versus unrolling the Swiss roll (right)

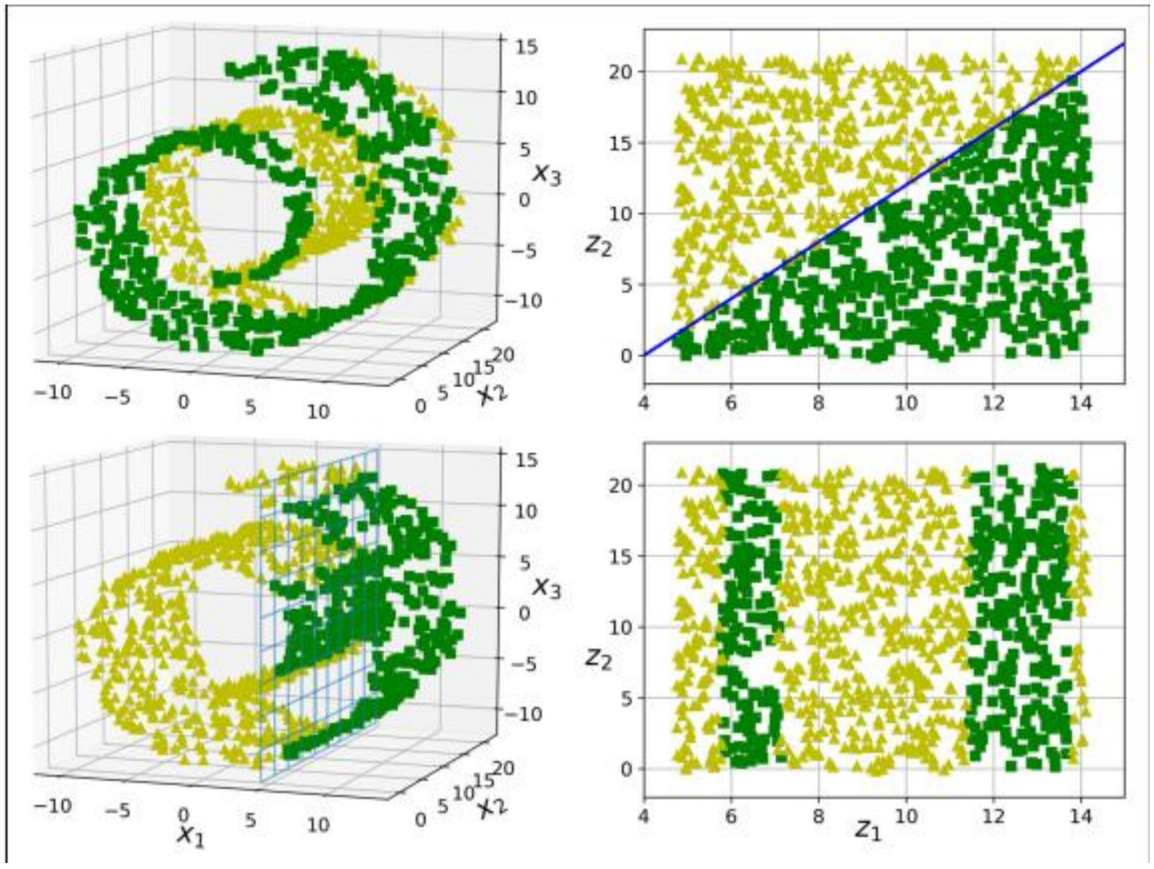

Figure 8-6. The decision boundary may not always be simpler with lower dimensions

PCA

Principal Component Analysis (PCA) is the most widely used dimensionality reduction technique. PCA performs a linear projection: it finds the hyperplane that best fits the data and projects the data onto it, choosing directions that preserve as much variance as possible. The intuition is that variance often correlates with signal: selecting the directions that preserve maximum variance tends to minimize information loss and also minimizes reconstruction error under a mean squared error interpretation. PCA identifies **principal components (PCs)**, which are orthogonal axes ordered by the amount of variance they explain—PC1 explains the most variance, PC2 explains the largest remaining variance orthogonal to PC1, and so on.

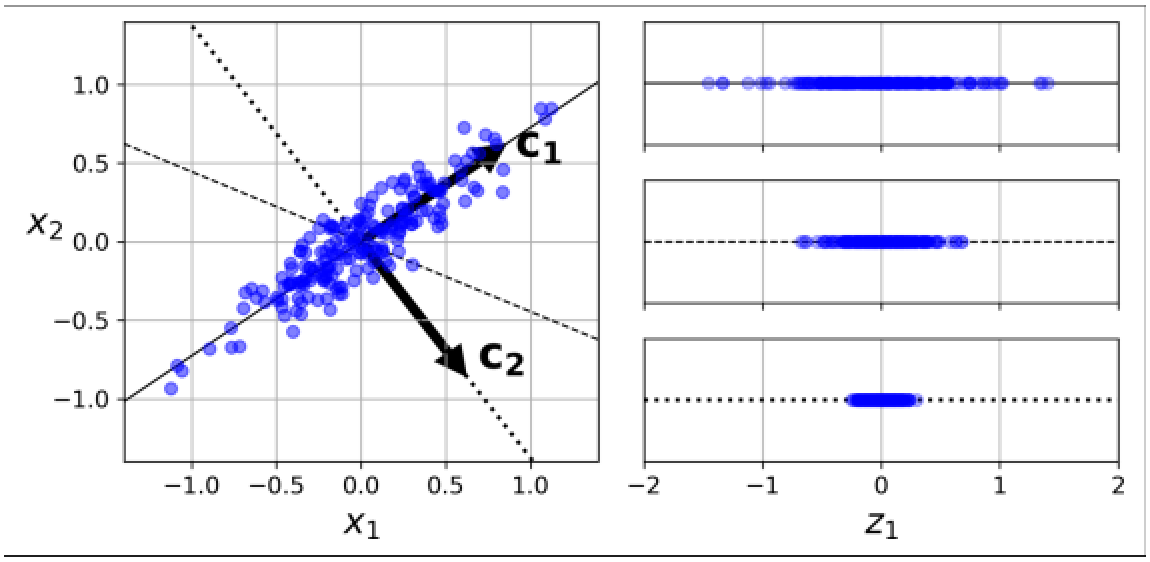

Figure 8-7. Selecting the subspace to project on

Principal Components via SVD
PCA is commonly computed using **Singular Value Decomposition (SVD)**. After centering the data, SVD decomposes the data matrix and yields principal component directions. One practical note is that the direction (sign) of principal component vectors is not stable: a component can flip direction without changing the axis it represents, and when variances are very close, components can even swap/rotate slightly—yet the subspace they span generally stays stable.

Projecting to d Dimensions
Once PCs are computed, dimensionality reduction to **d dimensions** is done by projecting the dataset onto the subspace spanned by the first d PCs. This preserves the most variance possible among all linear projections to d dimensions. Scikit-Learn's PCA automates centering and provides a convenient fit_transform() interface.

Explained Variance Ratio and Choosing d
PCA provides an **explained variance ratio** per component, showing how much variance each PC captures. This makes it easy to choose an appropriate number of dimensions: rather than guessing d, you can keep enough PCs to preserve a target fraction of total variance (e.g., 95%). Another common diagnostic is plotting cumulative explained variance versus number of components and choosing an "elbow point," where marginal gains diminish.

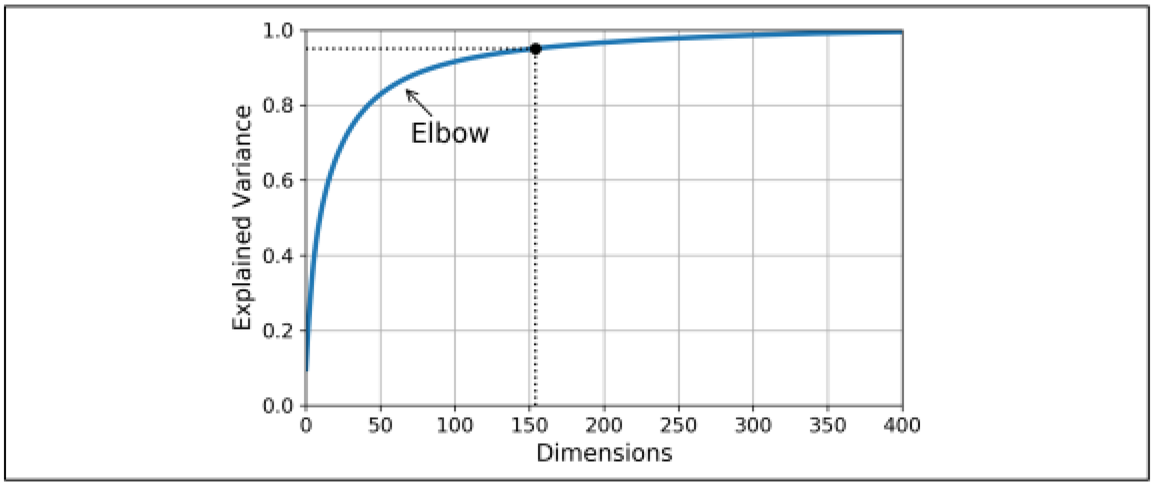

Figure 8-8. Explained variance as a function of the number of dimensions

PCA for Compression and Reconstruction Error
PCA can be used as a lossy compression technique. After reducing dimensionality, the dataset requires far less storage and training may become significantly faster. PCA also supports approximate reconstruction via inverse_transform(), producing a decompressed approximation of the original inputs. The difference between original data and reconstructed data is measured via **reconstruction error** (often MSE).

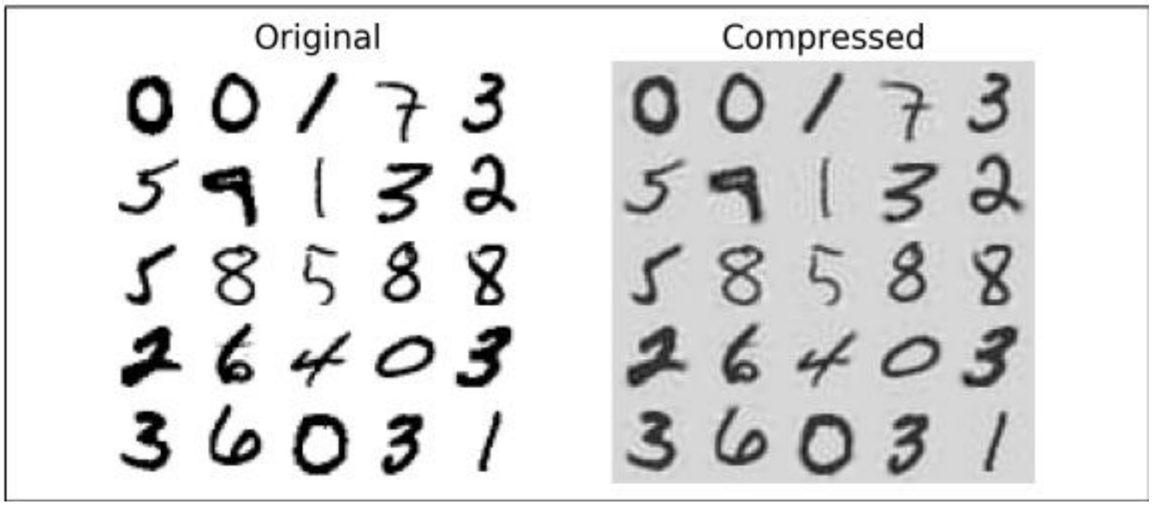

Figure 8-9. MNIST compression that preserves 95% of the variance

Randomized PCA and Incremental PCA
Standard PCA via full SVD can be expensive when the dataset is large. **Randomized PCA** uses a stochastic method to approximate the first d PCs much faster when d is far smaller than the original dimensionality. **Incremental PCA (IPCA)** addresses memory constraints by processing data in mini-batches, allowing PCA to run on datasets that do not fit in RAM and supporting streaming/online scenarios. IPCA can also be combined with memory-mapped arrays (memmap) for disk-backed workflows.

Kernel PCA
PCA is linear, so it cannot capture nonlinear manifolds like the Swiss roll. **Kernel PCA (kPCA)** extends PCA using the **kernel trick**, implicitly mapping data into a high-dimensional feature space where a linear PCA corresponds to a nonlinear projection in the original space. kPCA can preserve nonlinear cluster structure and sometimes “unroll” complex manifolds more effectively than linear PCA. In Scikit-Learn, KernelPCA supports kernels such as linear, RBF, and sigmoid.

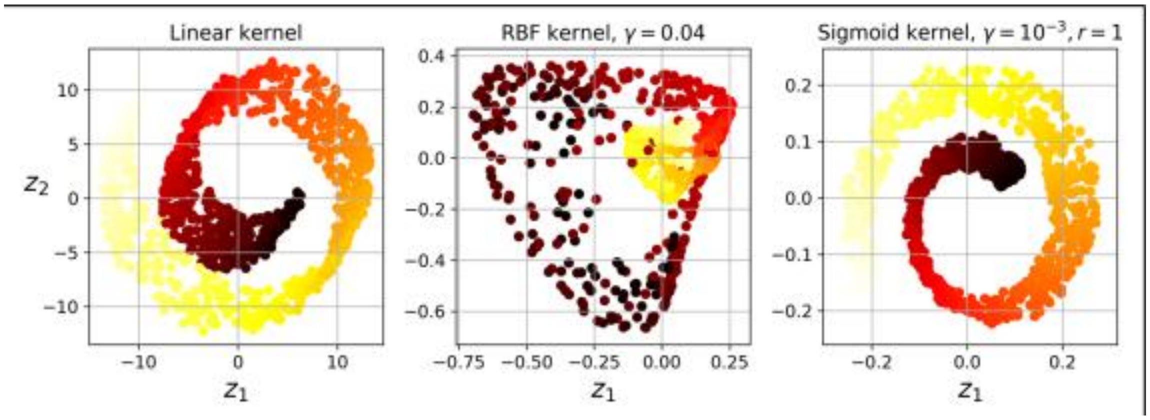

Figure 8-10. Swiss roll reduced to 2D using kPCA with various kernels

Selecting Kernel and Hyperparameters
Because kPCA is unsupervised, selecting kernels and hyperparameters is less direct than in supervised learning. A common practical strategy is to treat dimensionality reduction as part of a supervised pipeline and tune kPCA using grid search based on downstream performance (e.g., classification accuracy after Logistic Regression). Another approach is to choose settings that minimize a reconstruction-style error. Since kPCA reconstruction is not straightforward (the inverse mapping is not naturally available due to the implicit infinite-dimensional feature space), the concept of a **reconstruction pre-image** is used: find a point in the original input space that maps close to the reconstructed point in feature space, then measure error in the input space. Scikit-Learn can approximate this when fit_inverse_transform=True, enabling inverse_transform() for KernelPCA.

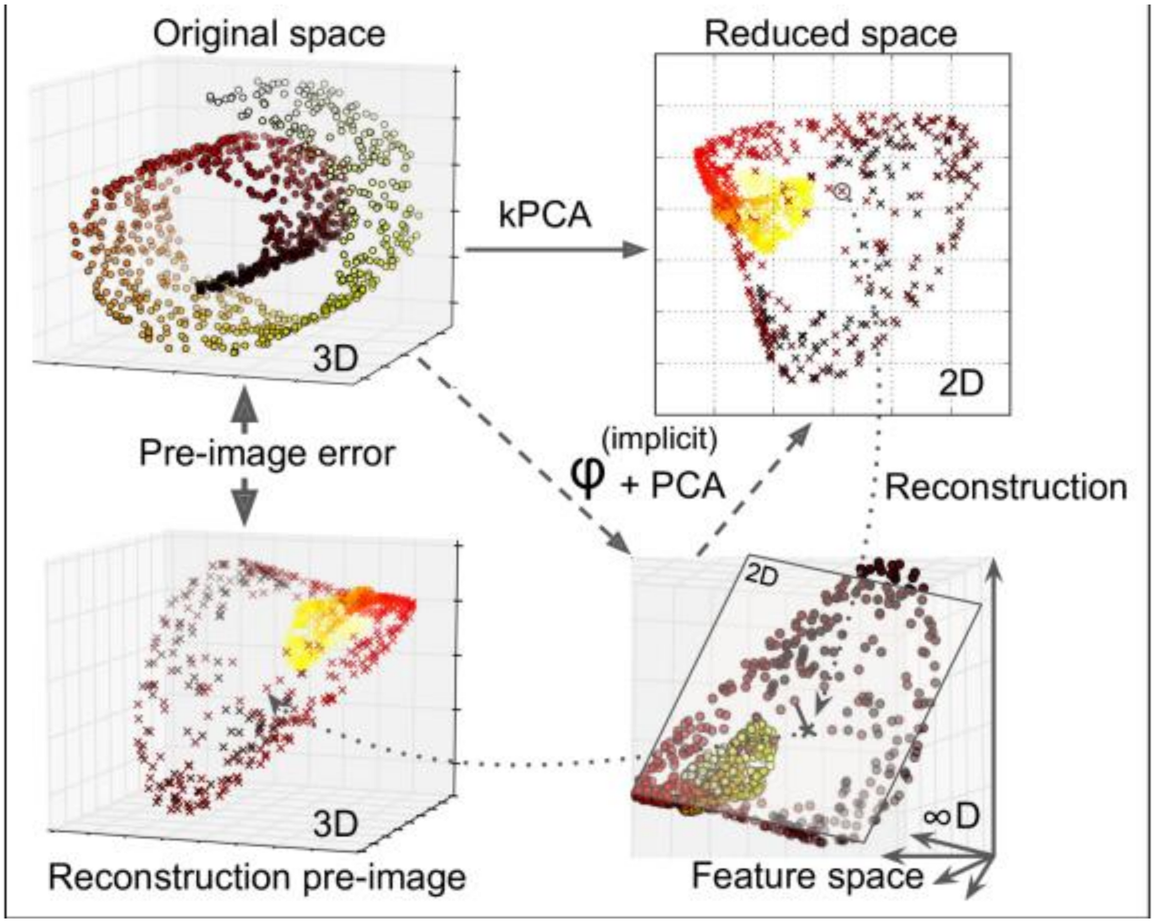

Figure 8-11. Kernel PCA and the reconstruction pre-image error

LLE

Locally Linear Embedding (LLE) is a manifold learning method designed to preserve local neighborhood structure. The algorithm first finds each instance's k nearest neighbors, then represents each point as a weighted linear combination of its neighbors. In the second step, it finds a low-dimensional embedding that preserves these reconstruction weights as well as possible. LLE is effective at "unrolling" manifolds (including the Swiss roll), but it scales poorly to very large datasets due to computational complexity terms involving m^2.

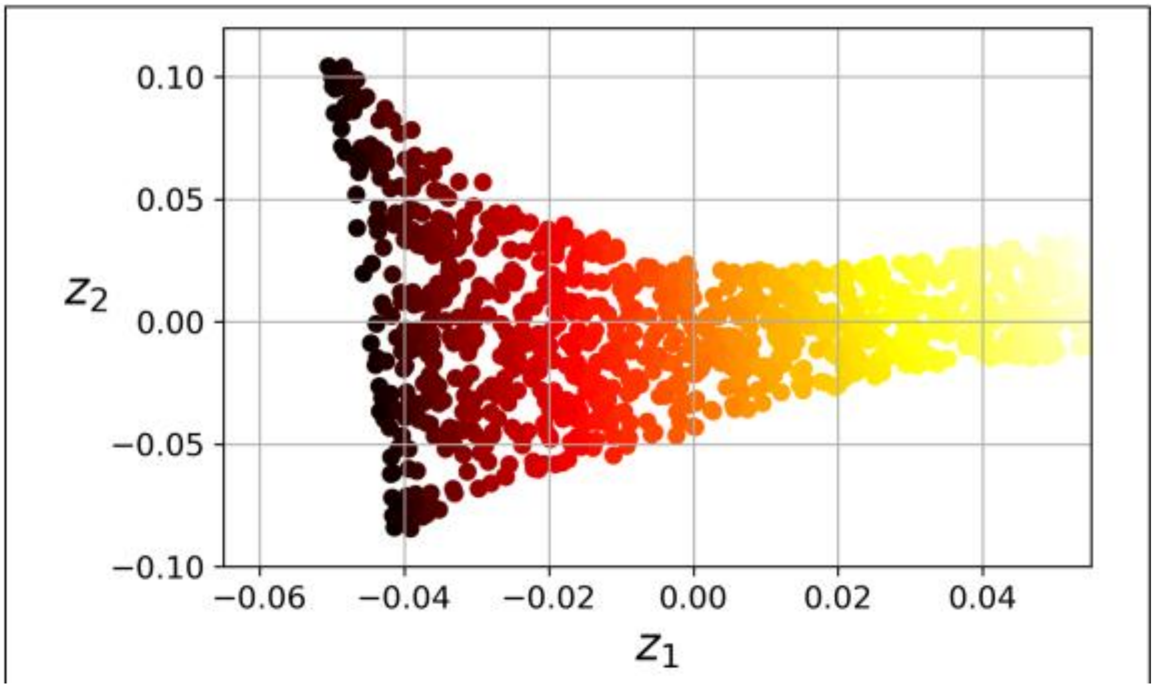

Figure 8-12. Unrolled Swiss roll using LLE

Other Dimensionality Reduction Techniques

Beyond PCA/kPCA/LLE, many techniques exist with different optimization goals. **Random Projections** use random linear maps that often preserve distances surprisingly well (Johnson–Lindenstrauss lemma). **MDS** tries to preserve pairwise distances. **Isomap** preserves geodesic distances over a neighbor graph. **t-SNE** focuses on preserving local similarity structure and is heavily used for visualization and cluster inspection. **LDA**, while primarily a classification method, learns discriminative axes that can be used to reduce dimensionality in a way that separates classes well—often useful as a preprocessing step before another classifier.

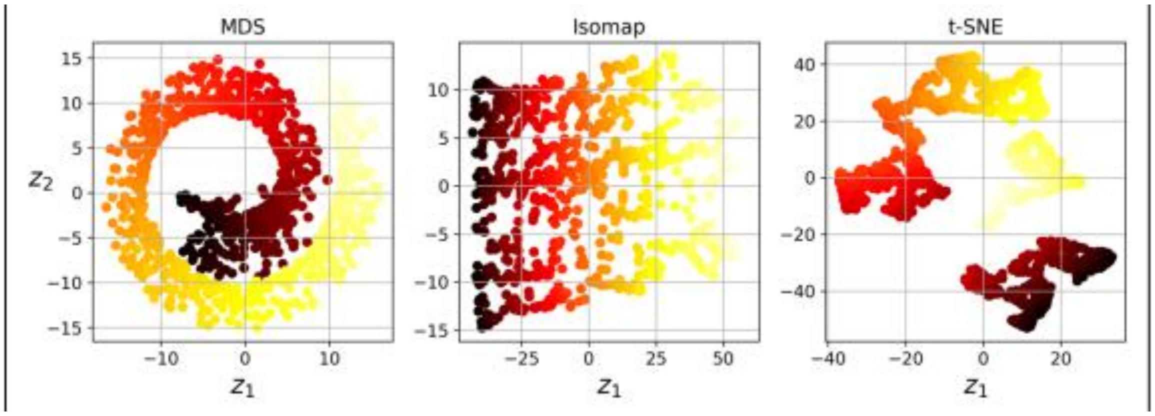

Figure 8-13. Using various techniques to reduce the Swiss roll to 2D