Dimensionality Reduction

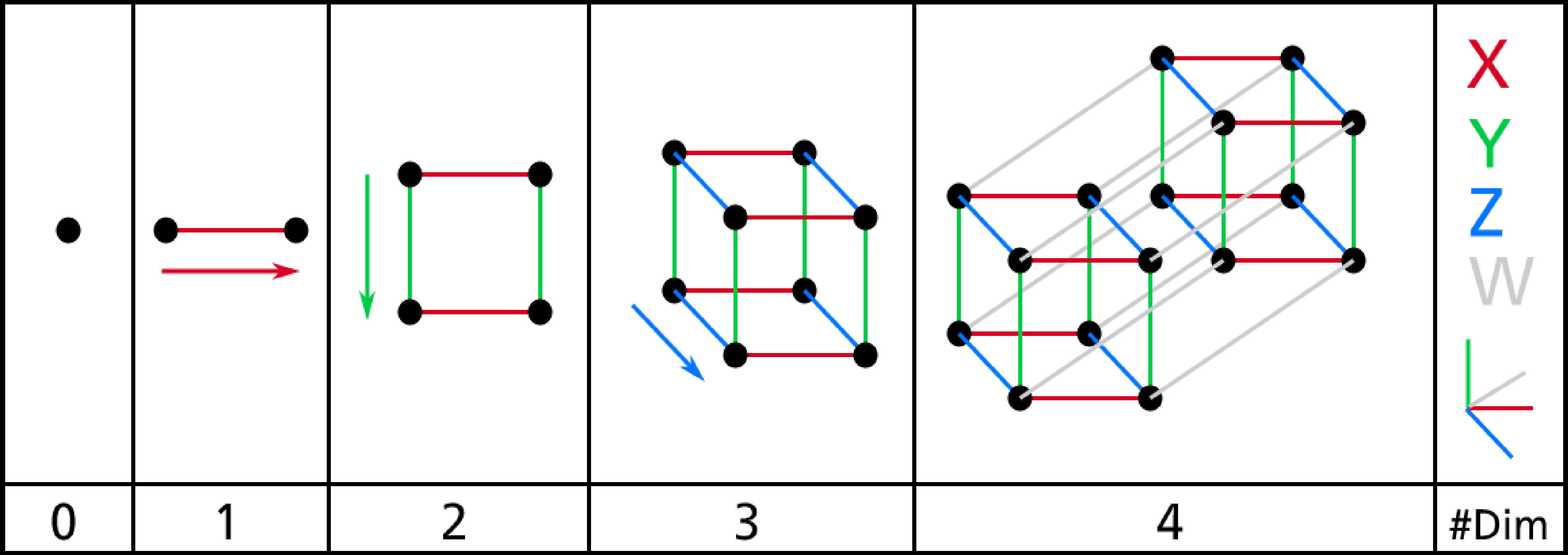
Chapter 8: pp 213 – 234

Why Dimensionality Reduction?

- Have to deal with thousands or millions of features → need to reduce the number of dimensions
  - Turn intractable to tractable problem
  - Speed up training
  - Good for data visualization
  - Mitigate the *curse of dimensionality*
- Reducing dimensionality causes some loss of information
- Main approaches: projection and manifold learning

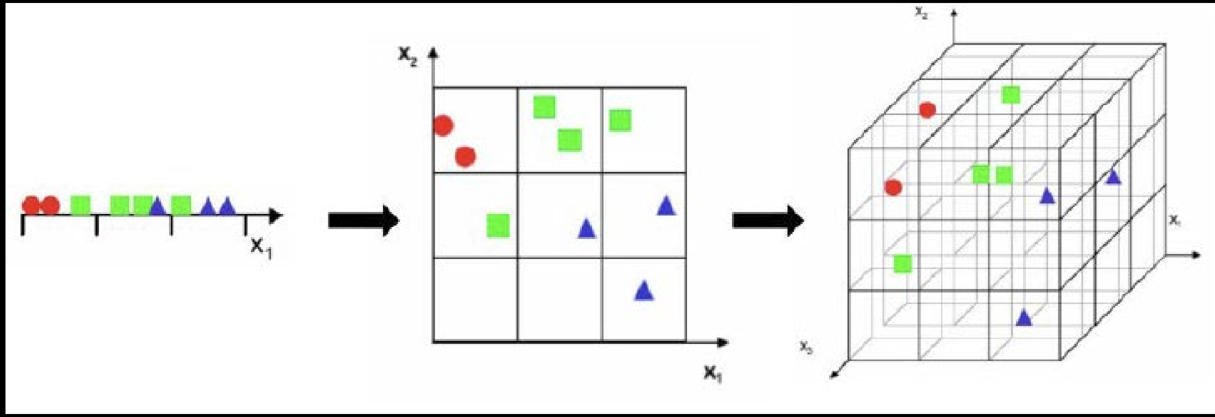
0D ot 4D hypercubes

Figure 8-1: point, segment, square, cube, tessaract



Curse of Dimensionality

- Case 1: average distance between 2 points
  - In a 1x1 square = 0.52
  - In a 1x1x1 cube = 0.66
  - In a 1M dimensional hypercube = 408.25
- Case 2:



- High-dimensional datasets tend to be very sparse
- The number of samples required to maintain statistical significance grows exponentially with the number of dimensions
- *Hughes phenomenon* – the size of training data tends to remain fixed as dimensionality increases causing loss of classifiability

Projection

- Training data in most real-world problems are not spread out uniformly across all dimensions

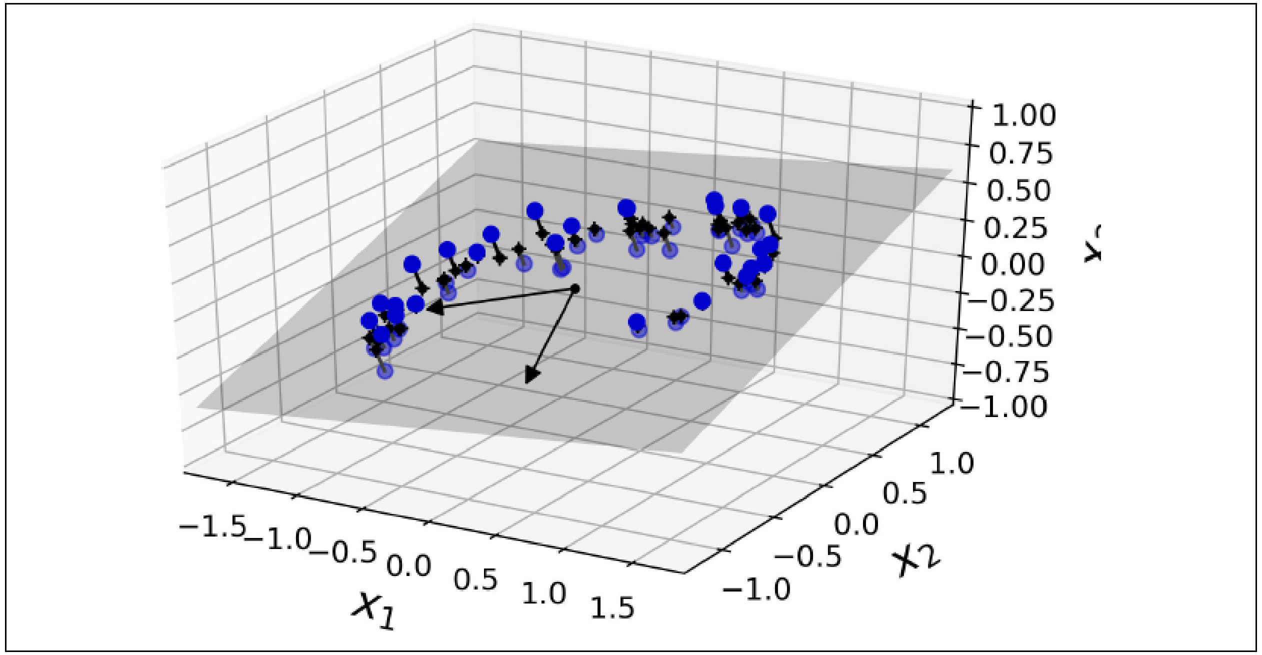


Figure 8-2. A 3D dataset lying close to a 2D subspace

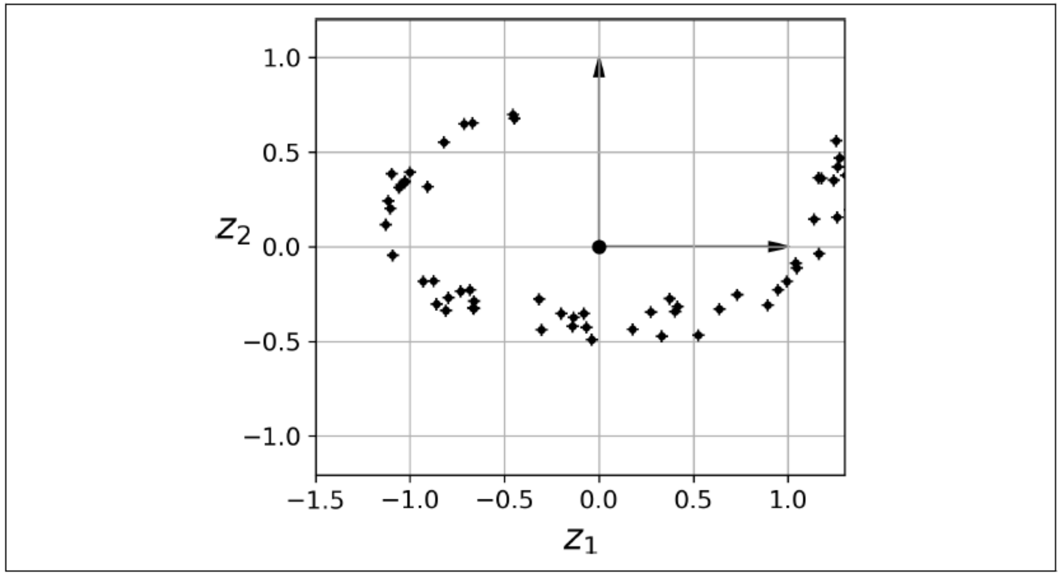


Figure 8-3. The new 2D dataset after projection

- Projection may not always be the best approach to dimensionality reduction

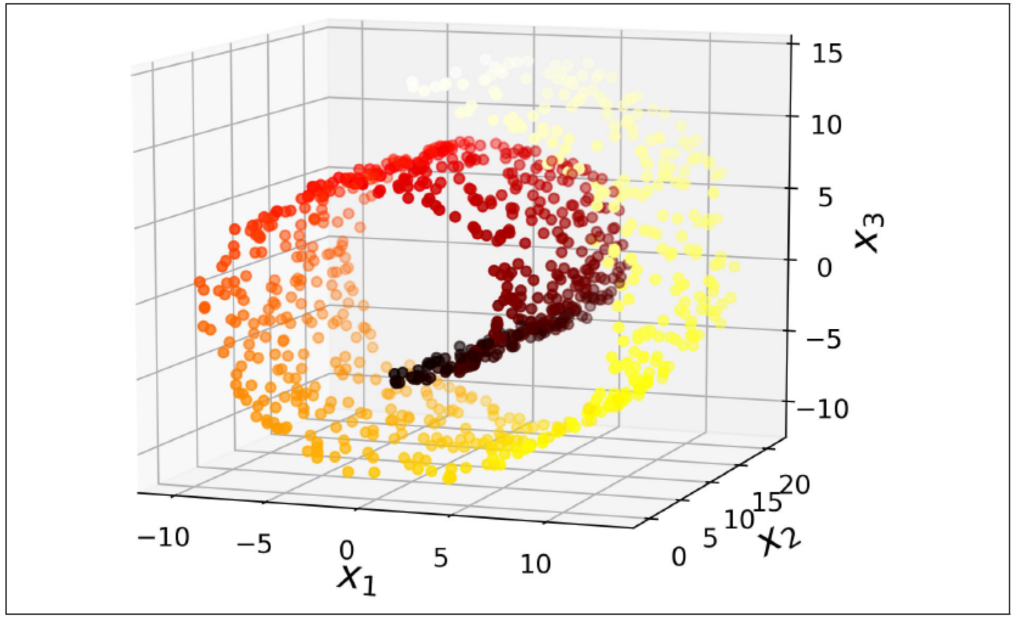


Figure 8-4. Swiss roll dataset

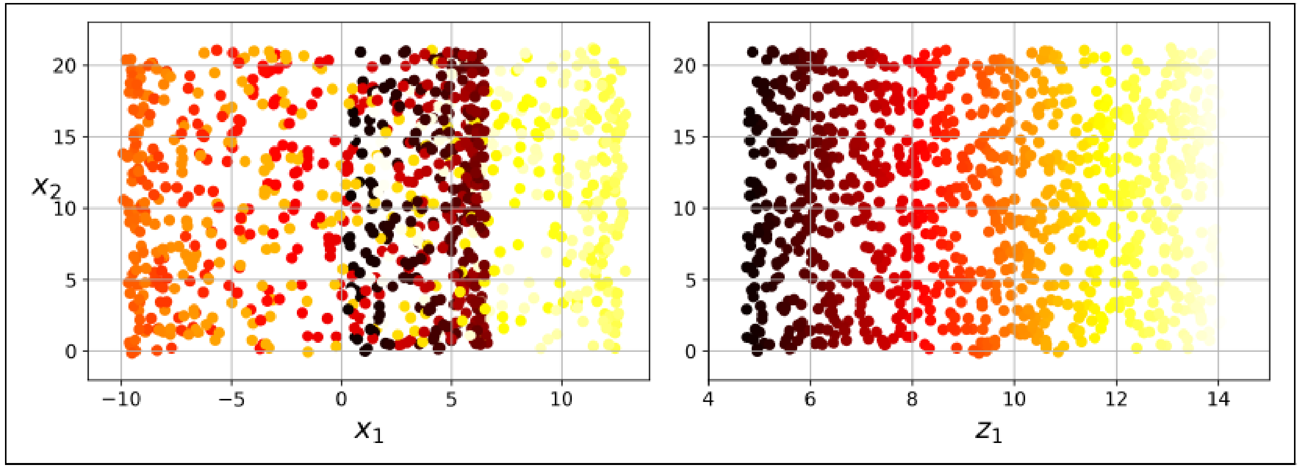


Figure 8-5. Squashing by projecting onto a plane (left) versus unrolling the Swiss roll (right)

Manifold Learning

- *d*-dimensional manifold is a *d*-dimensional shape that can be bent and twisted in an *n*-dimensional space ($d < n$)
- Manifold Learning is a dimensionality-reduction technique by modeling the manifold on which training data lies based on the *manifold assumption* (*manifold hypothesis*)
  - High-dimensional datasets tend to lie close to a much lower-dimensional manifold

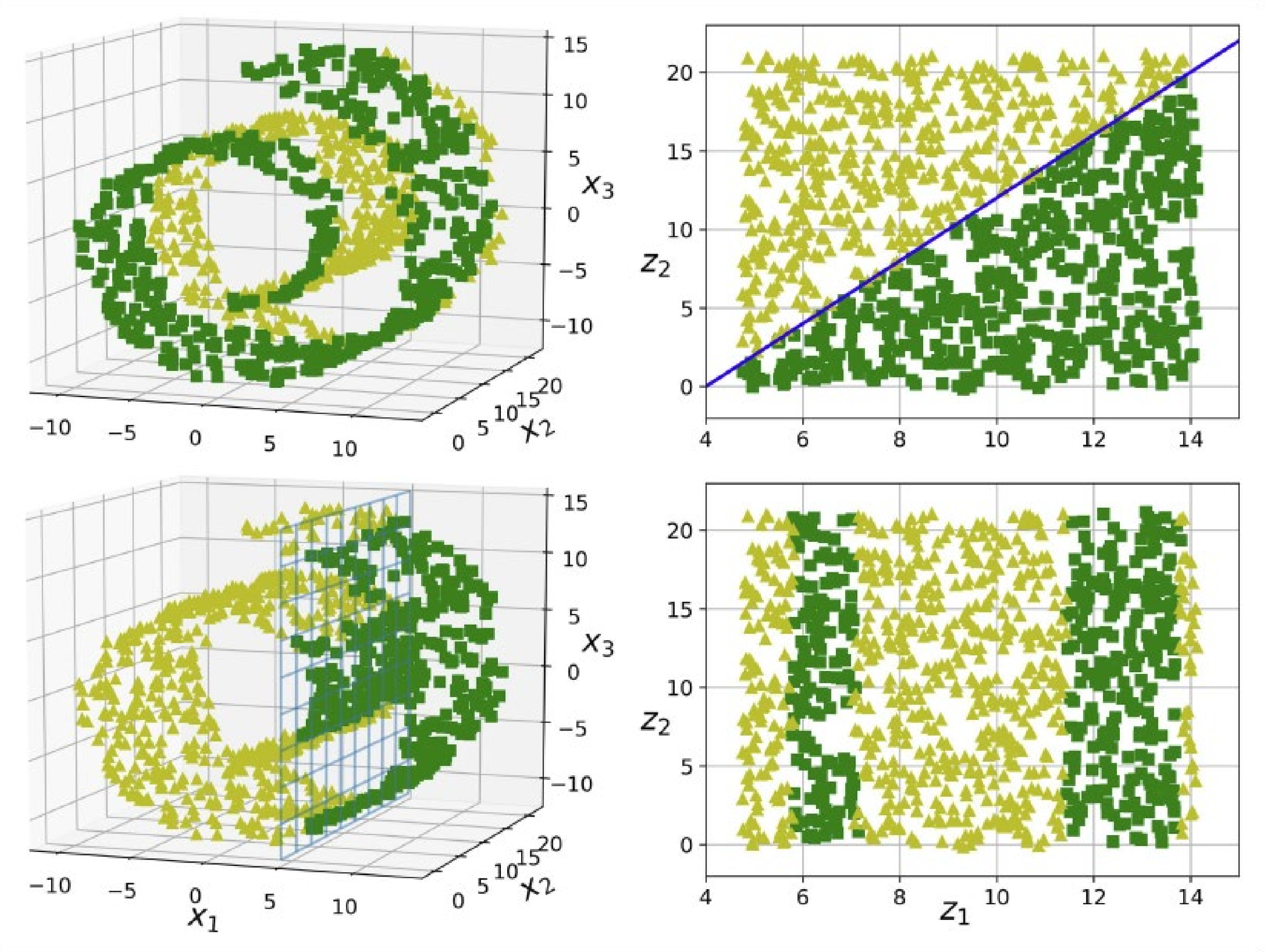


Figure 8-6. The decision boundary may not always be simpler with lower dimensions

Principal Component Analysis

- Preserving the Variance
  - Choose the axis that preserves the maximum amount of variance

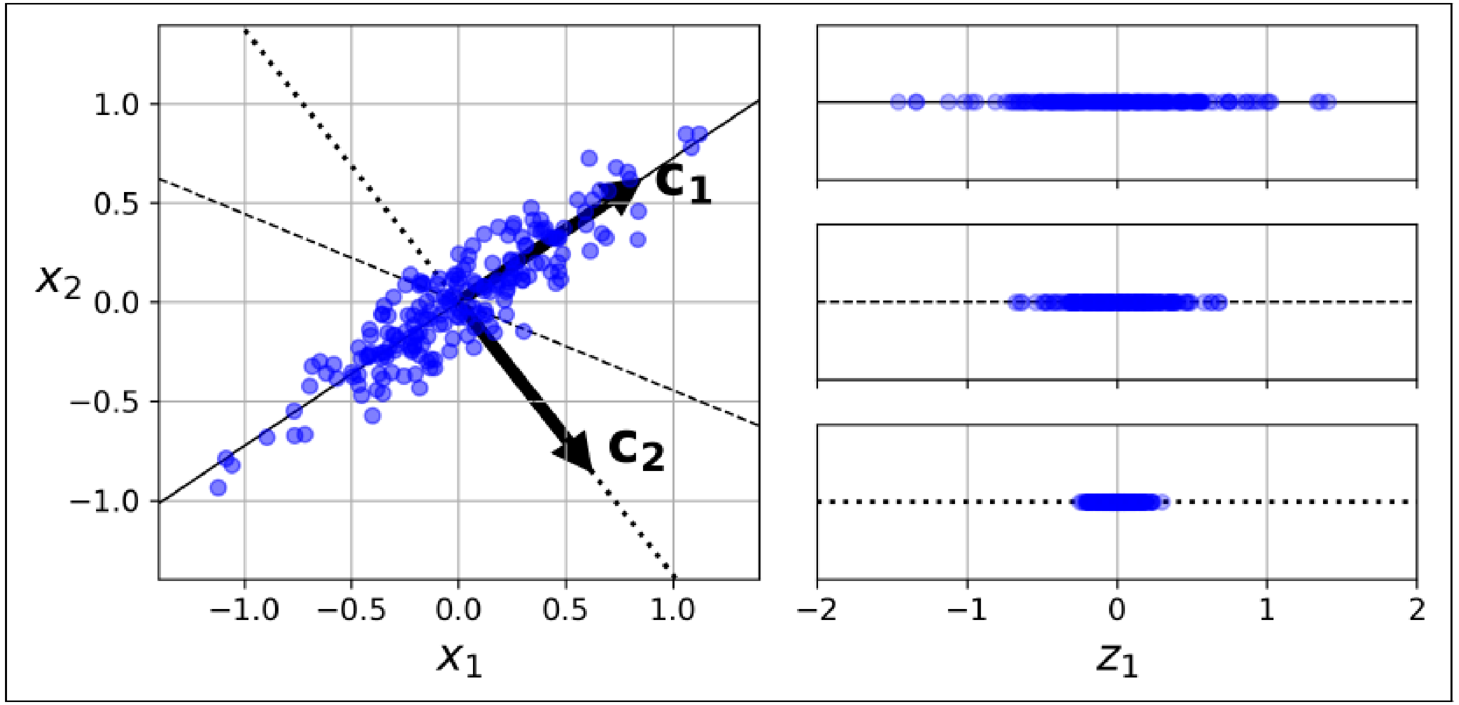


Figure 8-7. Selecting the subspace onto which to project

- Principal Components
  - i-th PC is the unit vector that defines the i-th axis
  - 1st PC is c_1, 2nd PC is c_2
  - Use SVD or Eigen-decomposition to compute PCs
  - Assume dataset is centered around the origin

Equation 8-1. Principal components matrix

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \mathbf{c_n} \\ | & | & & | \end{pmatrix}$$

Example: Iris Dataset Classification Accuracy

	Using all 4 features	Using 1st feature	Using 1st Principal Component
10-fold cross validation	0.9333	0.8000	1.0000
	1.0000	0.7333	0.8667
	1.0000	0.6000	0.9333
	1.0000	0.5333	0.8000
	1.0000	0.6000	1.0000
	1.0000	0.6667	0.8667
	0.9333	0.9333	1.0000
	0.9333	0.5333	0.9333
	1.0000	0.8667	0.9333
	0.9333	1.0000	0.9333
Average	**0.9733**	**0.7267**	**0.9267**

Projecting Down

Equation 8-2. Projecting the training set down to d dimensions

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

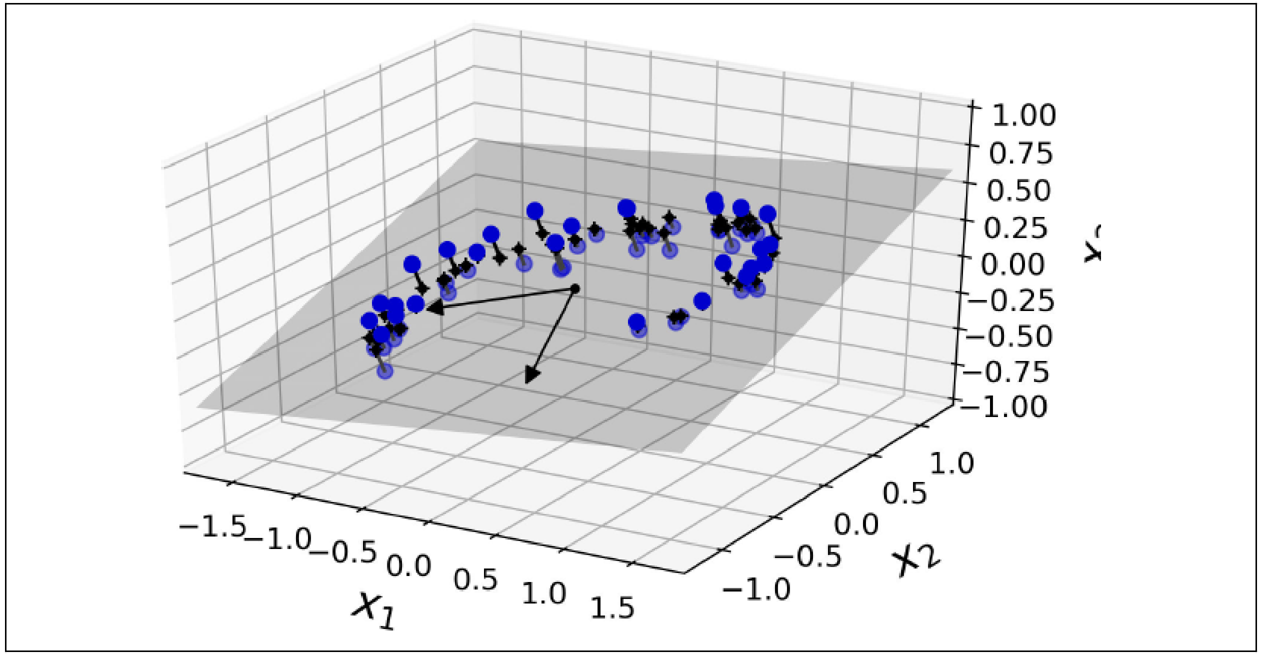


Figure 8-2. A 3D dataset lying close to a 2D subspace

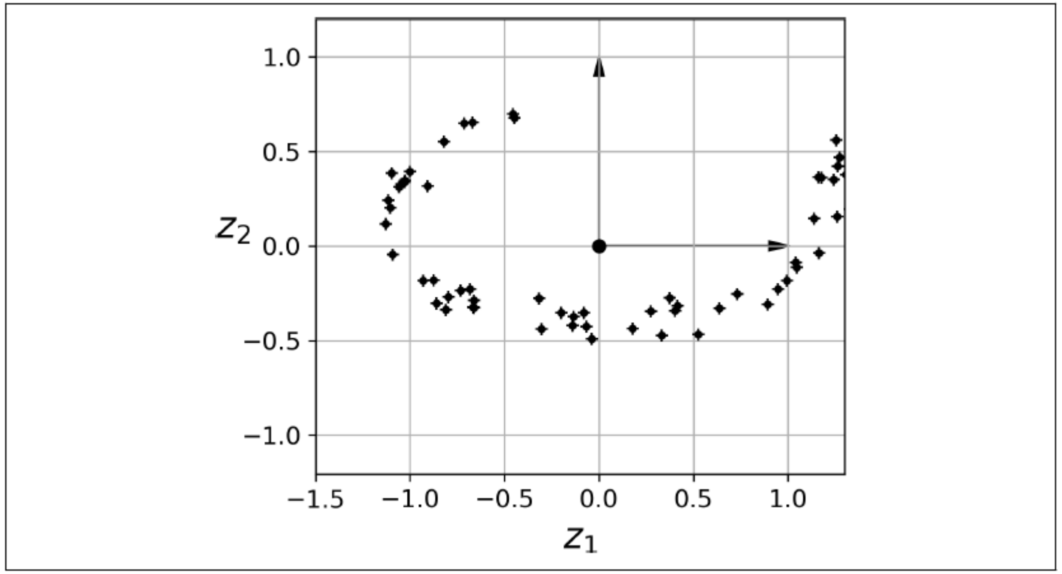


Figure 8-3. The new 2D dataset after projection

Explained Variance Ratio

- Proportion of the dataset's variance that lies along the axis of each PC
- Projecting 3D dataset onto 2D space:

Axis	Variance
1st axis	84.2 %
2nd axis	14.6 %
3rd axis	1.2 %

- Explaned Variance: measures the discrepancy between a model and actual data

Choosing the Right Number of Dimensions

- Rule of thumb: preserve approx. 95% of the variance
- Plot the explained variance and find the “elbow”

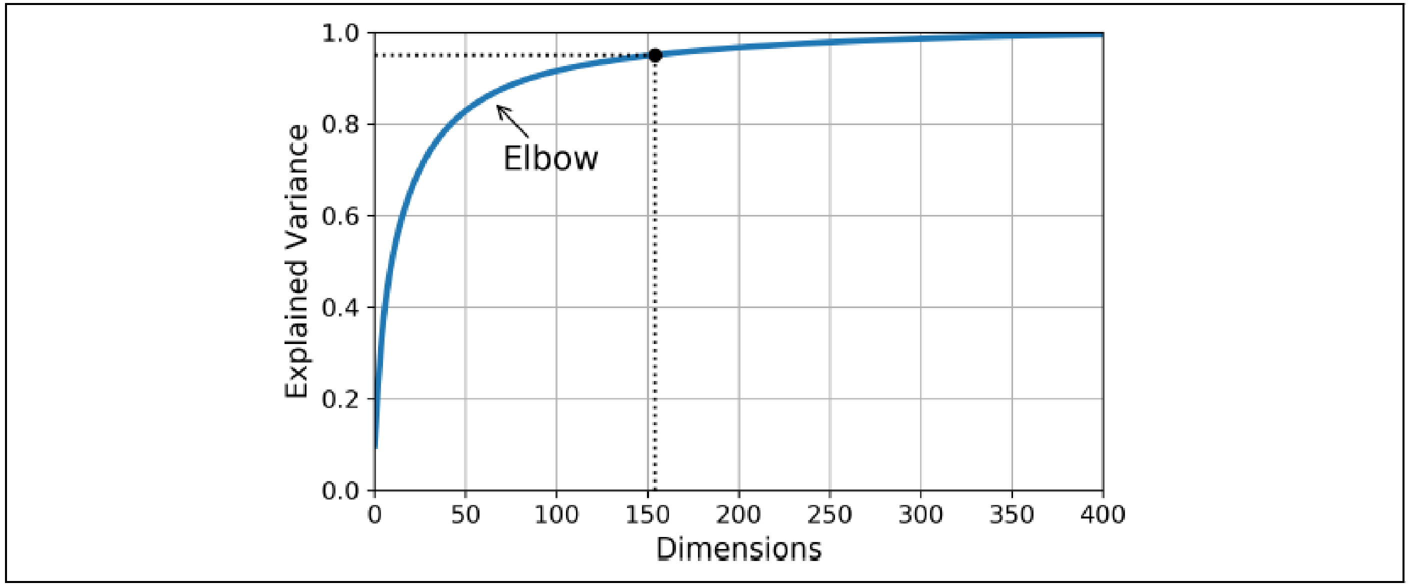


Figure 8-8. Explained variance as a function of the number of dimensions

PCA for Compression

- Recover the image from reduced MNIST dataset (154 dimensions, instead of the original 784 dimensions)

- Reconstruction error: mean squared distance between original and reconstructed data

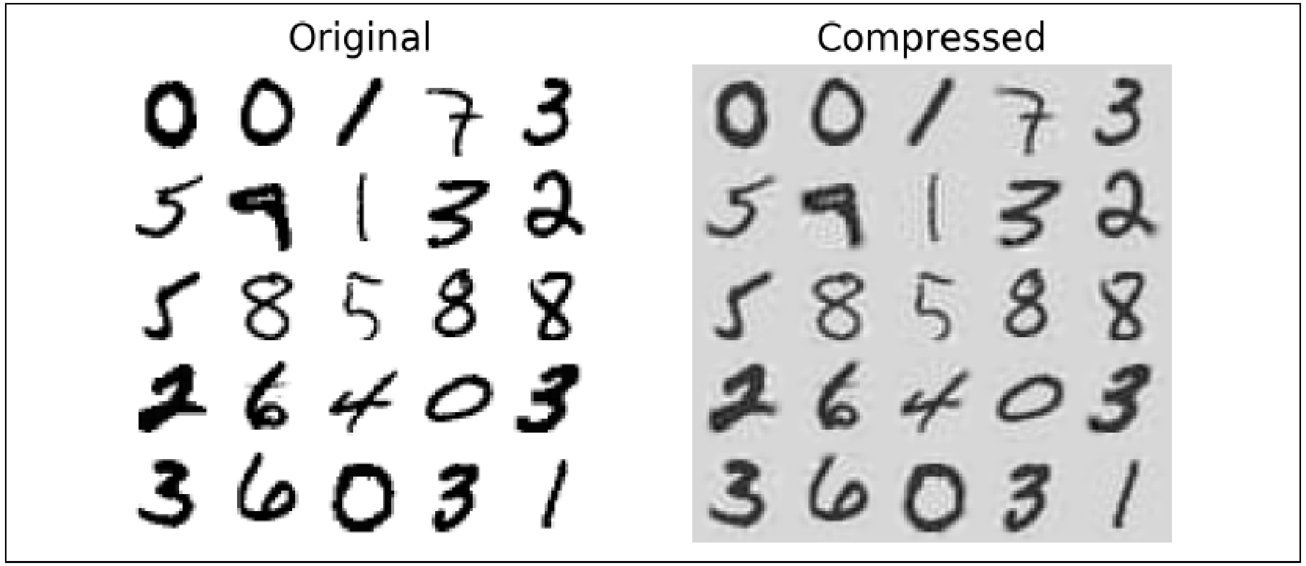


Figure 8-9. MNIST compression preserving 95% of the variance

Equation 8-3. PCA inverse transformation, back to the original number of dimensions

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}} \mathbf{W}_d^{\;T}$$

Randomized PCA

- Finds an *approximation* of the first *d* PCs
- Computational complexity
  - Full SVD: $O\left(m \times n^2\right) + O(n^3)$
  - Randomized PCA: $O\left(m \times d^2\right) + O(d^3)$ → much faster if $d < n$

    - m: number of instances
    - n: number of dimensions

Incremental PCA

- Split training set into mini-batches
- Feed one mini-batch at a time
- Useful for large training set and PCA online

Kernel PCA

- Kernel trick – maps instances into a very high-dimensional space (feature space)

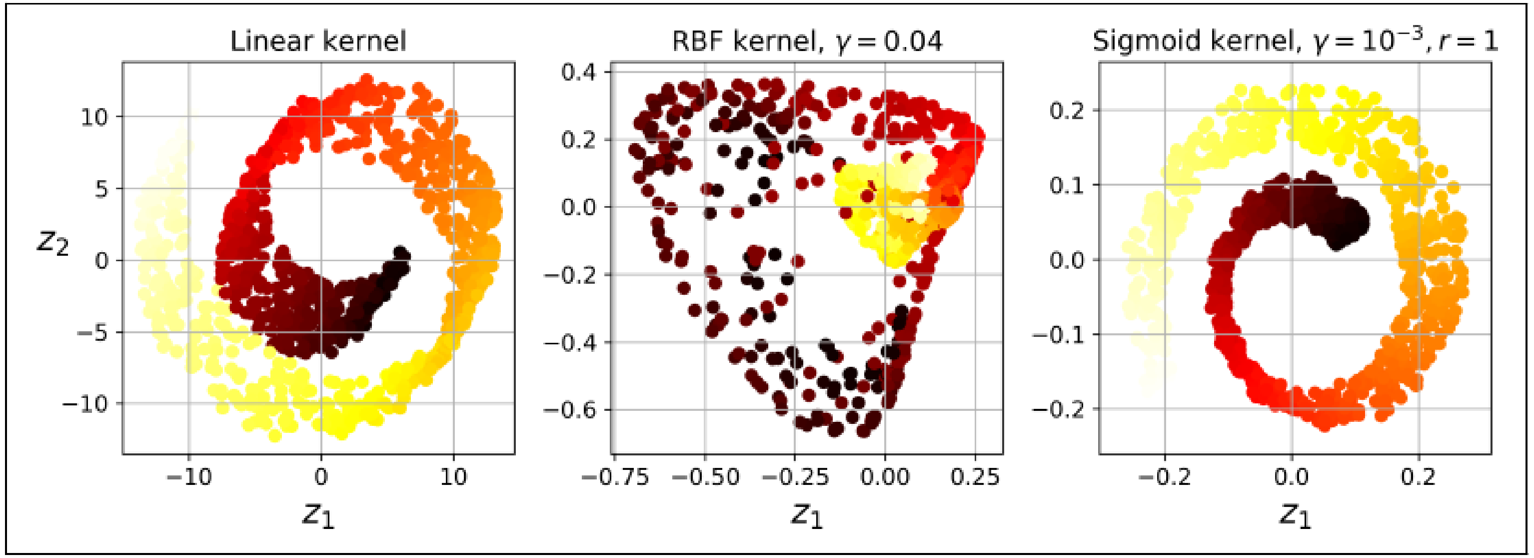


Figure 8-10. Swiss roll reduced to 2D using kPCA with various kernels

How to Select a Kernel

- If DR is a preprocessing step for a supervised learning task, use grid search to select the kernel that lead to the best performance on that task
- Find the kernel that yields the lowest reconstruction error
  - Apply kPCA to transform original space into *reduced space*
  - Perform inverse PCA in reduced space to find the reconstructed point in the high-dimensional feature space → can't compute reconstruction error in *feature space*
  - Form reconstruction pre-image then measure the distance to the original instance

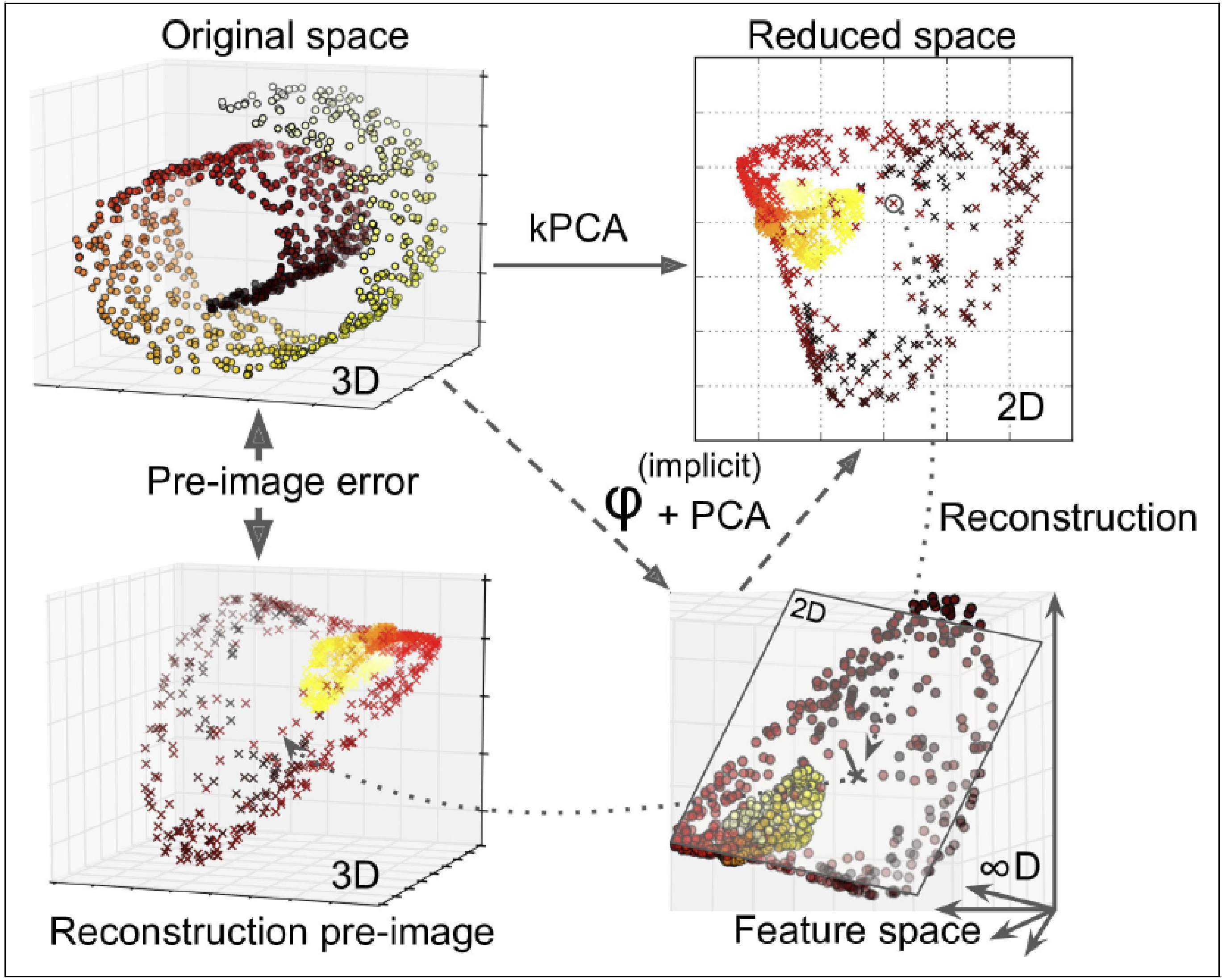


Figure 8-11. Kernel PCA and the reconstruction pre-image error

Reconstruction Pre-Image

- Train a supervised regression model with projected instances as the training set and the original instances as the targets
- Compute reconstruction pre-image error
- Use grid search to find the kernel that minimizes this error

Locally Linear Embedding (LLE)

- Non-Linear Dimensionality Reduction (NLDR)
- Measures how each training instance linearly relates to its closest neighbors
- Find low-dimensional representation that best preserves these relationships

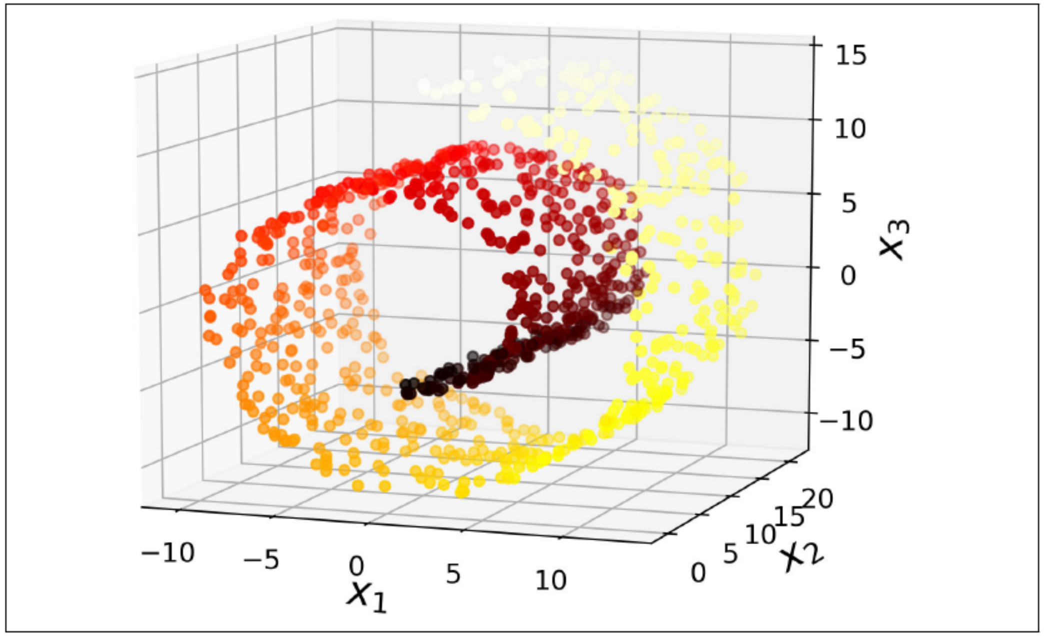


Figure 8-4. Swiss roll dataset

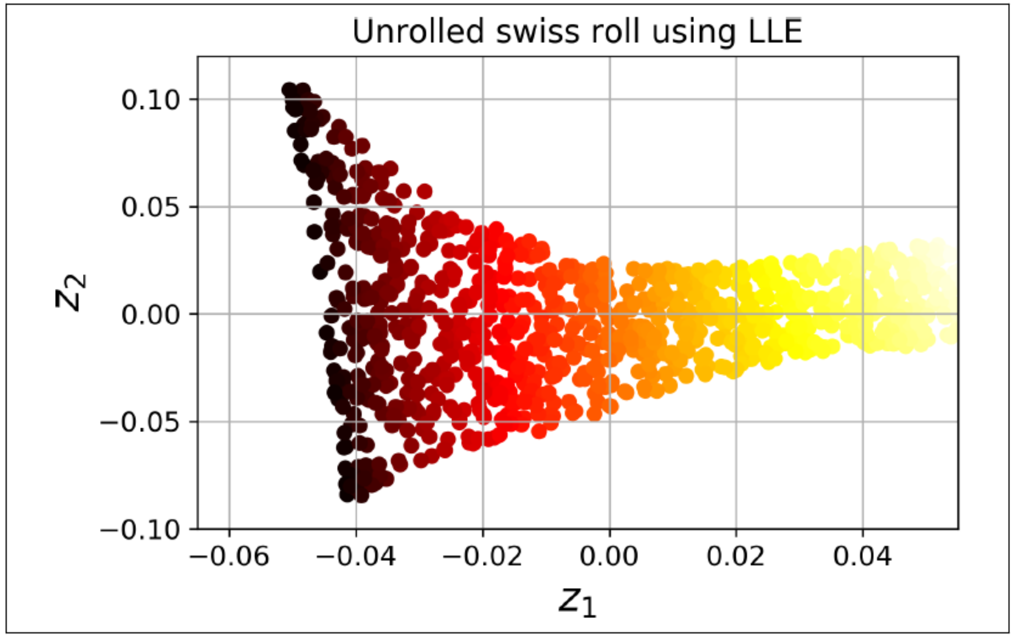


Figure 8-12. Unrolled Swiss roll using LLE

- Multi-Dimensional Scaling (MDS)
  - Preserves the distances between instances
- Isomap
  - Preserves the geodesic distances between instances
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
  - Keeps similar instances close and dissimilar instances apart

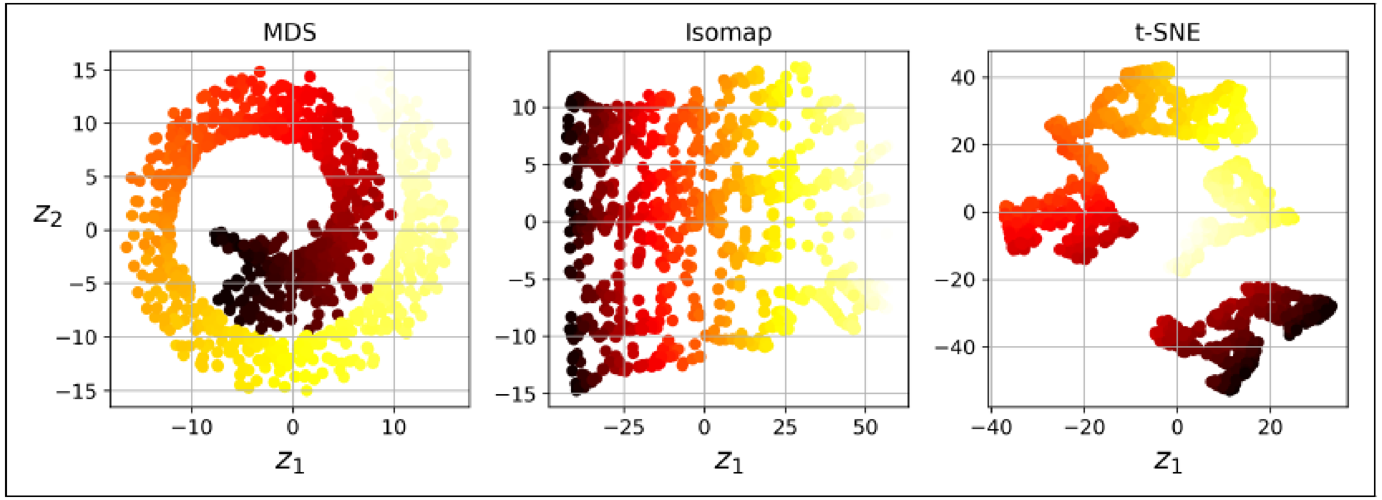


Figure 8-13. Reducing the Swiss roll to 2D using various techniques

- Linear Discriminant Analysis / Fisher's Discriminant
  - Learns the most discriminative axis between classes
  - Uses these axis to define a hyperplane onto which to project data
  - Keeps classes as far apart as possible
  - Maximize the ratio of *between-class variance* to *within-class variance*