Dimensionality Reduction

Simplifying Data Without Losing Essence

What is Dimensionality Reduction?

Definition: The process of reducing the number of features (dimensions) in a dataset.

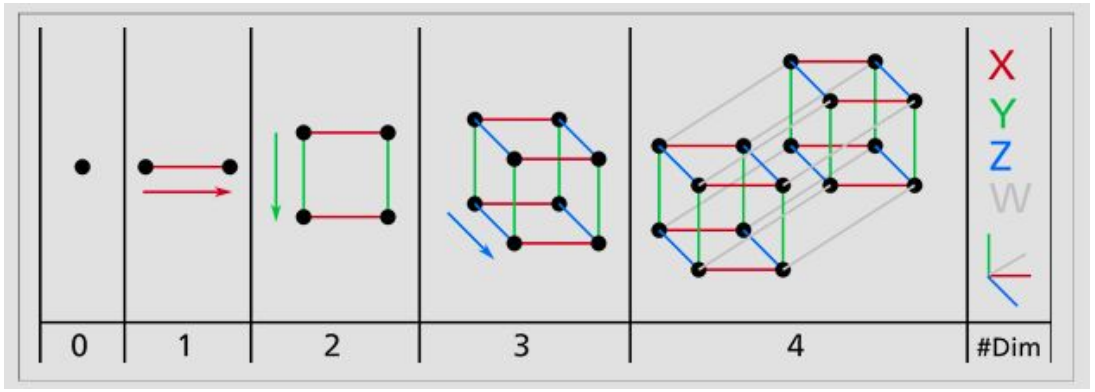
Goal: Simplify data while preserving as much meaningful information as possible.

Benefits:

- Reduces computational cost
- Visualizes high-dimensional data
- Avoids overfitting (curse of dimensionality)
- Removes redundant or noisy features

The Curse of Dimensionality

- As the number of features grows, data becomes sparse.
- Distances between points become less meaningful.
- Models become more complex and prone to overfitting.
- Visualization becomes nearly impossible beyond 3D.



Dimension	Distance between two points
Unit Square	0.52
3D Cube	0.66
1,000,000-dimensional Hypercube	

Main Approaches to Dimensionality Reduction

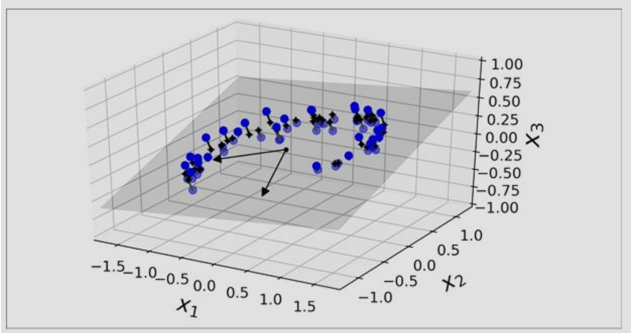


Figure 8-2. A 3D dataset lying close to a 2D subspace

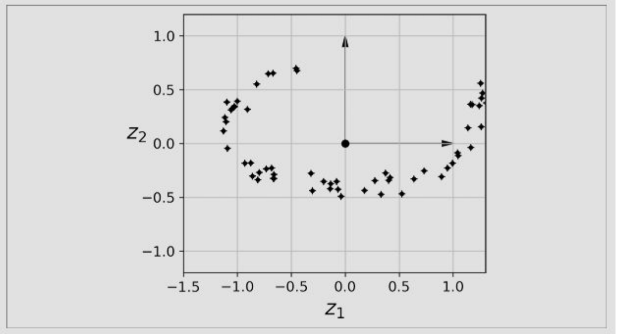


Figure 8-3. The new 2D dataset after projection

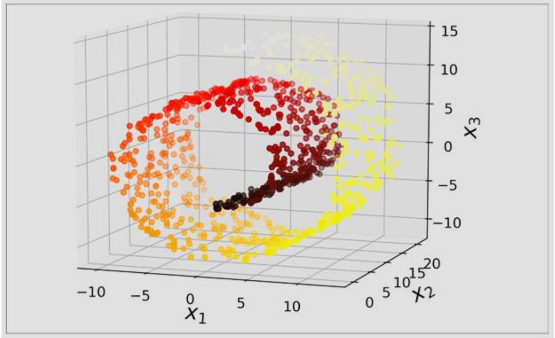


Figure 8-4. Swiss roll dataset

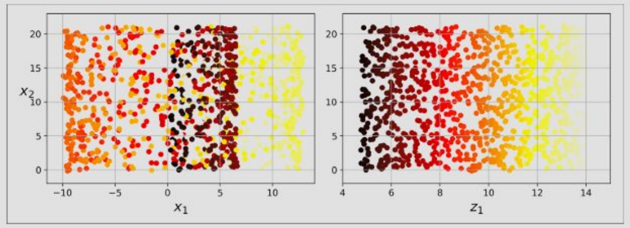


Figure 8-5. Squashing by projecting onto a plane (left) versus unrolling the Swiss roll (right)

- **Projection**
  - Projects data into a lower-dimensional subspace.
  - Example: PCA (Principal Component Analysis)

Main Approaches to Dimensionality Reduction

- **Manifold Learning**
  - Unfolds twisted data into a lower-dimensional space.
  - **Manifold hypothesis:**
    - Most real-world high-dimensional datasets lie close to a much lower-dimensional manifold.
  - A d-dimensional manifold is a part of an n-dimensional space (where $d < n$) that locally resembles a d-dimensional hyperplane.
    - In the case of the Swiss roll, $d = 2$ and $n = 3$
    - The Swiss roll is an example of a 2D manifold.
    - It locally resembles a 2D plane, but it is rolled in the third dimension
  - Example: LLE (Locally Linear Embedding)

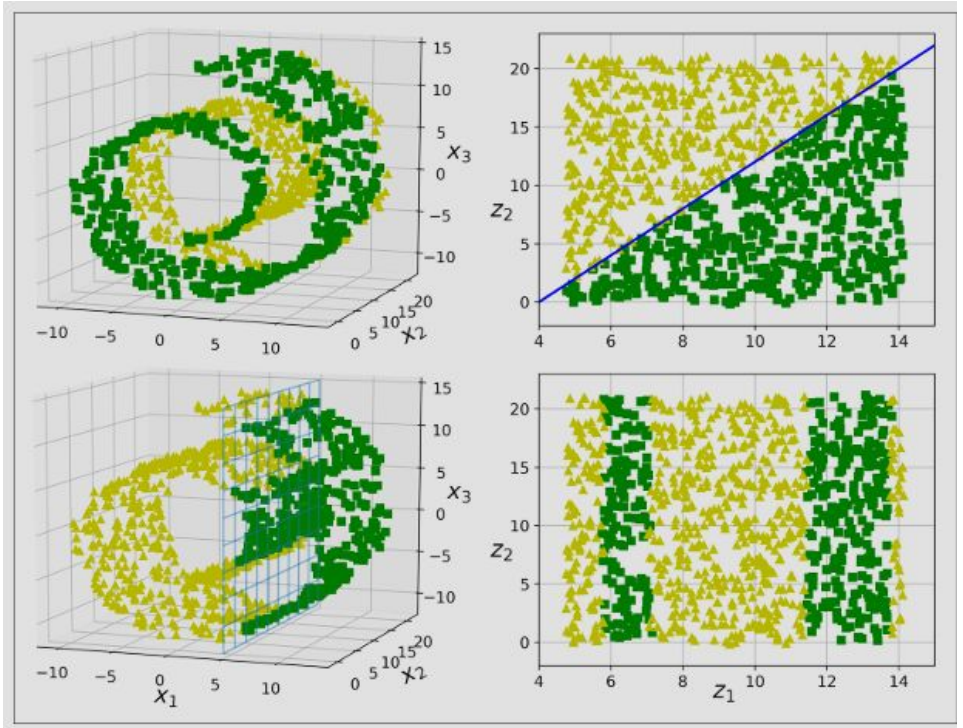
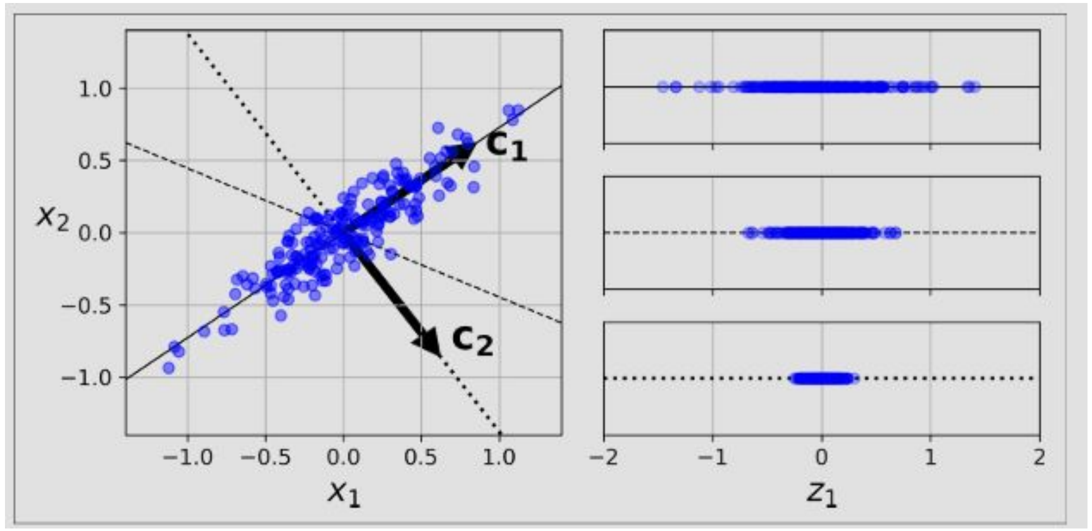


Figure 8-6. The decision boundary may not always be simpler with lower dimensions

Principal Component Analysis (PCA)

- Identifies the hyperplane that lies closest to the data, and then it projects the data onto it.
- **Idea:** Find the axis that preserves the maximum variance.
- **Steps:**
  - Center the data.
  - Compute covariance matrix.
  - Perform eigenvalue decomposition.
  - Select top *k* eigenvectors (principal components).
- **Result:** Data projected onto a lower-dimensional subspace.



Principal Components

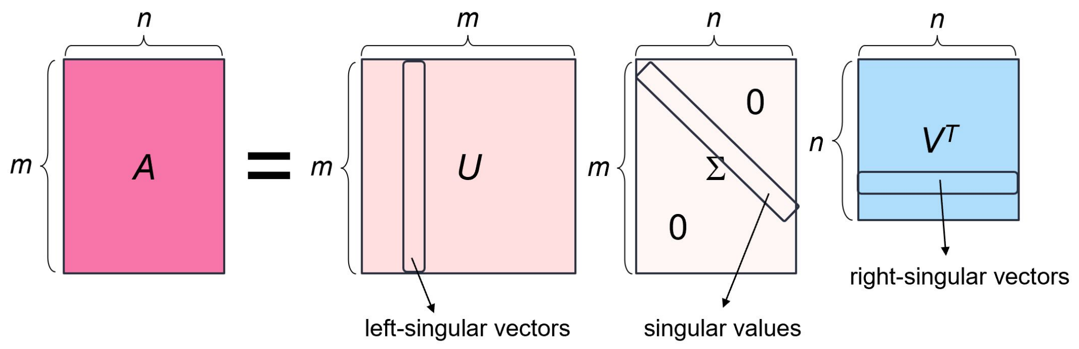
- The ith axis is called the ith Principal Component (PC) of the data.
  - The first PC is the axis on which vector c1 lies, and the second PC is the axis on which vector c2 lies.
- How to find the principal components of a training set?
  - **Singular Value Decomposition (SVD)**
    - Decompose the training set matrix X into the matrix multiplication of three matrices $U\,\Sigma\,V^\intercal$, where V contains the unit vectors that define all the principal components.

Introduction to SVD

- Singular Value Decomposition (SVD) is a factorization method in linear algebra.
- It decomposes a matrix into three other matrices: $A = U\Sigma V^T$
- Where V contains the unit vectors that define all the principal components.
- Widely used in:
  - Dimensionality reduction
  - Data compression
  - Noise reduction
  - Latent Semantic Analysis (LSA)

Components of SVD

-
- U ($m \times m$ matrix):
  - Orthogonal matrix
  - Columns = left singular vectors
- Σ ($m \times n$ matrix):
  - Diagonal matrix (singular values)
  - Values arranged in decreasing order
- V^T ($n \times n$ matrix):
  - Orthogonal matrix
  - Rows = right singular vectors



Geometric Intuition

-

- SVD decomposes a linear transformation into:
  - Rotation/Reflection (V^T)
  - Scaling (Σ)
  - Rotation/Reflection (U)
- It helps understand data structure in reduced dimensions.

Numerical Example

Let’s compute SVD for a 2×2 matrix: $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$

Steps:

1. Compute eigenvalues of $A^T A$
   - $A^T A = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix}$
   - $Eigenvalues: \lambda_1 = 16, \lambda_2 = 4$
2. Singular values ($\sqrt{eigenvalues}$):

$\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$

3. Compute eigenvectors of $A^T A$
   - Eigenvectors must be unit vectors
   - Vector V should be orthonormal in SVD
   - Eigenvector for eigenvalue 16: $[1, 1]/\sqrt{2}$
   - Eigenvector for eigenvalue 4:$[1, -1]/\sqrt{2}$
   - $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

4. Compute $U = A\, V\, \Sigma^{-1}$
   - After computation: $U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

Final Decomposition

-

$A = U\Sigma V^T$

Where:

- $U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

- $\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$

- $V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

Low-rank approximation (dimensionality reduction)

- Keep only the top singular value $\sigma_1 = 4$. The **rank-1** approximation is:

$$A_1 = \sigma_1 u_1 v_1^T$$

Where,

$$u_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

The rank-1 approximation is

$$A_1 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

Reconstruction error (Frobenius norm)

- $$A - A_1 = \begin{bmatrix} 3-2 & 1-2 \\ 1-2 & 3-2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Frobenius norm:

$$||A - A_1|| = \sqrt{1^2 + (-1)^2 + (-1)^2 + 1^2} = 2.$$

- If you keep both singular values ($k = 2$), the reconstruction is exact and the error is 0.

How much energy (variance) does the top singular value capture?

- $Energy \propto \sigma_i^2$

$$\sigma_1^2 = 4^2 = 16, \sigma_2^2 = 2^2 = 4$$

Fraction explained by σ_1:

$$\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} = \frac{16}{16+4} = \frac{16}{20} = 0.8 = 80\%$$

- The top singular value captures **80%** of the matrix energy/variance; keeping only that singular value yields a reasonable approximation (rank-1) that captures most structure but not all.

Projecting Down to d Dimensions

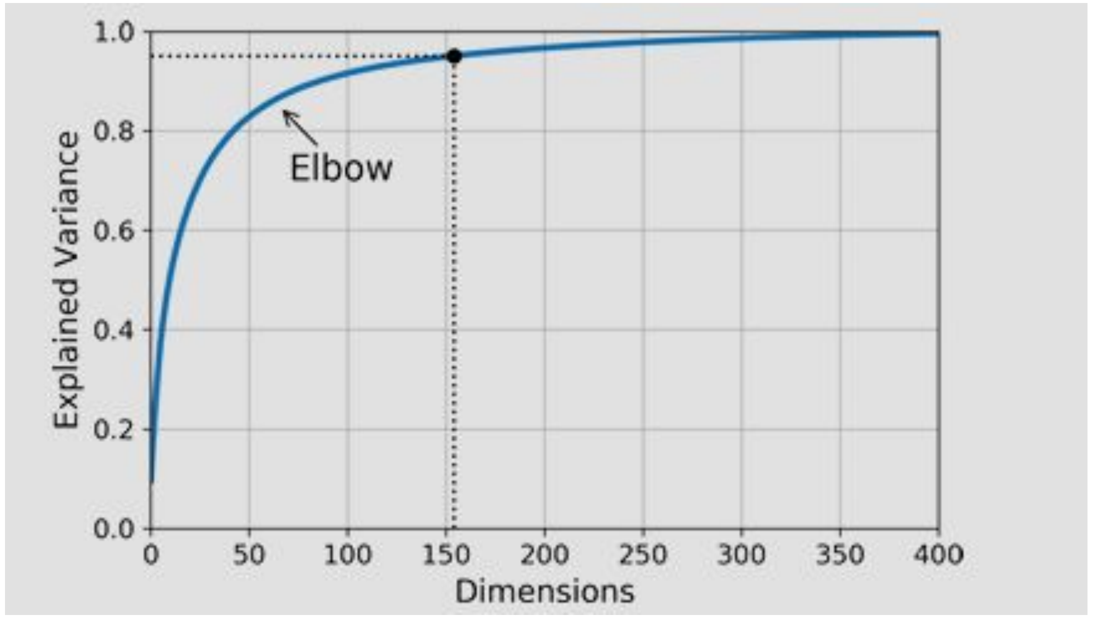
- Projecting dataset onto the hyperplane defined by the first d principal components.
- Projecting the training set down to d dimensions

$$X_{d\text{-}proj} = X \cdot W_d$$

- W_d is the matrix containing the first d columns of V

Choosing the Right Number of Dimensions

- Choose *k* such that ~95% of variance is retained.



PCA for Compression

- After dimensionality reduction, data takes less space.
- Can reconstruct original data (with loss) using inverse transform.
- PCA inverse transformation, back to the original number of dimensions

$$X_{recovered} = X_{d\text{-}proj} W_d^T$$

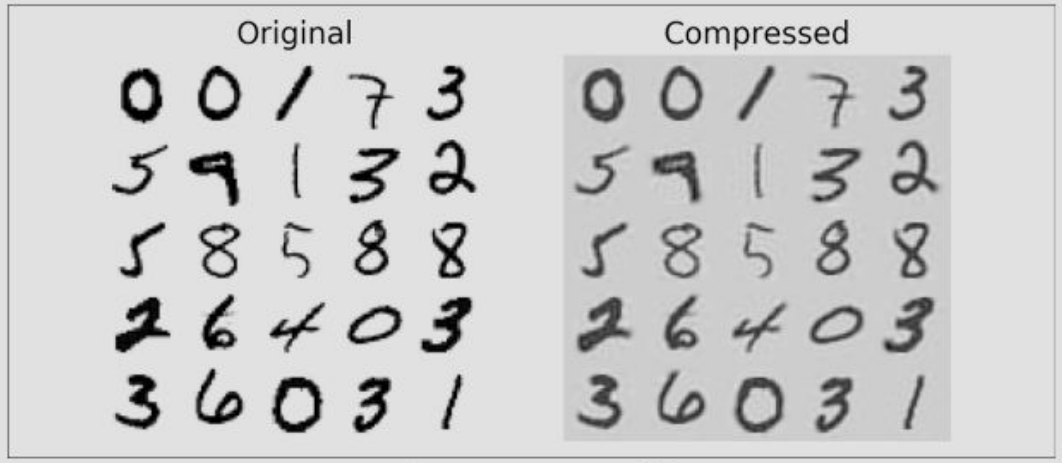


Figure 8-9. MNIST compression that preserves 95% of the variance

Locally Linear Embedding (LLE)

- **Idea:** Each point is a linear combination of its neighbors.
- Preserves local relationships.
- Good for unrolling twisted manifolds.
- **How LLE works:** For each training instance x^i, the algorithm identifies its k closest neighbors, then tries to reconstruct x^i as a linear function of these neighbors.

Locally Linear Embedding (LLE)

- LLE step one: Linearly modeling local relationships

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

$$subject\ to \begin{cases} w_{i,j} = 0, if\ x^j\ is\ not\ neighbor\ of\ x^i \\ \sum_{j=1}^{m} w_{i,j} = 1, \qquad for\ i = 1,2,\ldots,m \end{cases}$$

- LLE step two: Reducing dimensionality while preserving relationships

$$\hat{Z} = \underset{Z}{argmin} \sum_{i=1}^{m} \left(z^i - \sum_{j=1}^{m} w_{i,j} z^j \right)^2$$

Other Dimensionality Reduction Techniques

- **t-Distributed Stochastic Neighbor Embedding (t-SNE):** Great for visualization, preserves local structure.
- **Isomap:** Good for geodesic distance preservation.
- **UMAP:** Modern, fast, and effective for high-dimensional data.
- **Autoencoders:** Neural networks for nonlinear dimensionality reduction.
- **Multidimensional Scaling (MDS):** Reduces dimensionality while trying to preserve the distances between the instances.
- **Linear Discriminant Analysis (LDA):** 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 projection will keep classes as far apart as possible.