Dimensionality Reduction

2026-05-06 · cheerful mango Haubentaucher

Folding and Unfolding Space.

The Why

Dimensionality reduction finds those factors and discards the rest.

MNIST handwritten digits , for example, live in $\mathbb{R}^{784}$ (28×28 pixels), but vary along a handful of latent factors such as shapes or geometric primitives. The intrinsic dimensionality is estimated at roughly $k \approx 12$ to 14.

Real-world data $\mathbf{X} \in \mathbb{R}^{n \times d}$ is often high-dimensional: images have millions of pixels, genomes have billions of base pairs, text embeddings have thousands of dimensions. Every clustering method in Part I takes such a feature space as given and works within it. But while dimensionality comes at a cost and degrades our approaches, dimensionality carries information. The question then becomes whether a more compact representation could carry the same information.

Dimensionality has a price. In low dimensions, some points are close and others are far away; that contrast is what makes clustering work. As d grows, the contrast disappears: all pairwise distances converge toward the same value.

Computational cost grows with d, too. Distance computations scale as $O(nd)$, covariance matrices require $O(d^2)$ storage, and eigendecompositions cost $O(d^3)$. Reducing from d to k before fitting a downstream model can turn an intractable pipeline into a practical one.

Expressiveness of the feature space. Original features are often redundant or entangled. If two features $x^{(i)}$ and $x^{(j)}$ are highly correlated ($|\rho_{ij}| \approx 1$), they carry nearly the same information; a good representation replaces them with a single component that captures their shared variance.

INTERPRETABILITY benefits directly. Humans usually usually do not think and reason in high-dimensional vector spaces; we prefer low-dimensional, disentangled representations that align with our concepts of space.

IN ORDER TO MOVE FROM HIGH-DIMENSIONAL OBSERVATIONS TO COMPACT, SIGNAL-CARRYING REPRESENTATIONS we have good reason to find ways to,

- find a mapping that retains the pattern that carries the signal.
- construct proxy dimensions that are not limited to the original features, so that hidden patterns become apparent.
- understand what different notions of preserving pattern imply, and choose the right one for a given task.

Hands-On Experience

CONSIDER the canonical six points, but now arranged so that $x^{(1)}$ varies only slightly while $x^{(2)}$ carries nearly all the spread.

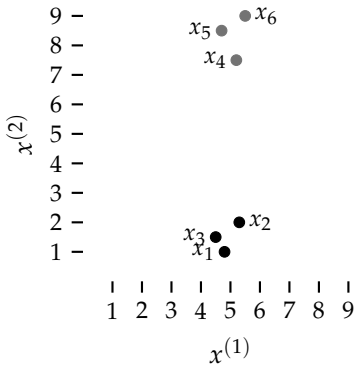


Figure 1: The canonical six points, repositioned so that $x^{(1)}$ barely varies while $x^{(2)}$ carries both the cluster separation and the within-cluster spread. Two features describe each point, but the data is essentially one-dimensional.

THE TWO CLUSTERS are clearly visible, and one feature carries almost all the signal. The first feature $x^{(1)}$ hovers around 5 for every point; dropping it loses almost nothing. The second feature $x^{(2)}$ alone separates the two groups and captures the within-cluster variation.

- If you had to keep only one of the two features, which would you choose?
- How much information would you lose?
- And could you recover the two clusters from that single feature alone?

THE METHODS IN THIS CHAPTER AUTOMATE THAT JUDGMENT OF DIMENSIONALITY REDUCTION. What is the most efficient mapping you can find for the MNIST dataset, using only geometric primitives such as circles, lines so that each class is well-separated?

A WELL-CHOSEN MAPPING $f\colon \mathbb{R}^d \to \mathbb{R}^k$ disentangles latent factors, yielding coordinates that are more discriminative for downstream tasks than the originals.

HAND-CRAFTED FEATURE ENGINEERING is dimensionality reduction by another name. When a domain expert selects "age" and "income" from a database with hundreds of columns, or computes edge gradients from raw pixels, they are projecting from $\mathbb{R}^d$ to $\mathbb{R}^k$ using human genius.

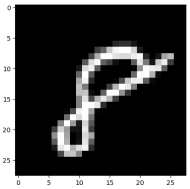


Figure 2: A single MNIST digit its meaningful variation spans few dimensions.

THE LEARNING OBJECTIVES of this lecture:

- Understand naive feature selection based on variance and its limitations.
- Understand the PCA objective and its connection to variance maximization and the singular value decomposition.
- Apply t-SNE and UMAP for visualization and avoid common misinterpretations of their output.
- Choose appropriate dimensionality reduction methods based on linearity, dataset scale, and whether the goal is preprocessing or visualization.

The Curse of Dimensionality, Revisited

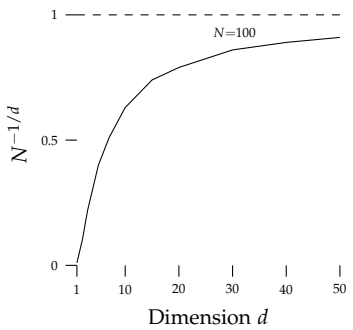


Figure 3: Mean nearest-neighbour spacing $d_{\mathrm{NN}} \propto N^{-1/d}$ for N=100 points. By d=50 every point is nearly as far from its neighbour as from the boundary. But if the data only occupies $k \ll d$ effective dimensions, the curse applies to k, not d.

THE KEY OBSERVATION is that high d is only fatal when all d dimensions carry independent information. In practice they rarely do. Features are correlated, redundant, or simply noisy. The effective degrees of freedom, the intrinsic dimensionality k, is often far smaller than d.

REDUCING d TO k REVERSES THE CURSE. The spacing, the distance concentration, the density collapse all depend on the dimension of the space the data actually lives in. If we project from $\mathbb{R}^d$ down to $\mathbb{R}^k$, the clustering methods from Part I regain their footing: distances become discriminative again, ε-neighbourhoods fill up, and centroids sit among their members.

THE BLESSING OF DIMENSIONALITY. The Johnson-Lindenstrauss lemma shows that n points in high dimensions can be projected to $O(\log n/\epsilon^2)$ dimensions while preserving pairwise distances within $(1\pm\epsilon)$. Even random projections work.

THE QUESTION IS HOW TO FIND THE RIGHT k DIRECTIONS. Picking original features by hand (as in variance-based selection) works when the axes happen to align with the signal. When they do not, we need methods that construct new directions from the data itself. That is what PCA, and later t-SNE and UMAP, will do.

A Naïve Baseline: Variance-Based Feature Selection

FEATURE SELECTION falls into three families: *filter* methods score features independently of any model (e.g. mutual information, variance thresholding), *wrapper* methods evaluate subsets by training a predictor, and *embedded* methods perform selection inside the learning objective (e.g. ℓ_1 regularisation). Wrapper and embedded methods require labels; filters may or may not. Without labels we are limited to unsupervised proxies such as variance.

WHEN AXES ALIGN WITH THE SIGNAL consider the most straightforward way to reduce dimensions: measure the variance of each original feature independently and keep only those with the highest variance.

Given data $X \in \mathbb{R}^{n\times d}$, compute the variance along each coordinate axis:

$$\mathrm{Var}(x^{(j)}) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^{(j)} - \bar{x}^{(j)}\right)^2, \quad j = 1,\ldots,d$$

Sort the features by decreasing variance, keep the top k, and discard the rest. This is pure feature selection: no new axes are created, only existing ones are retained or dropped.

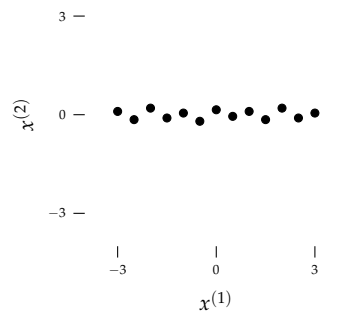


Figure 4: Data spread along $x^{(1)}$ but nearly constant in $x^{(2)}$. Dropping $x^{(2)}$ loses almost nothing.

WHEN DOES THIS WORK? If the coordinate axes already align with the directions of meaningful variation, variance-based selection is fast, interpretable, and surprisingly effective. A near-constant feature carries almost no information, and removing it loses little.

When does it fail? When the signal does not happen to align with the axis, for example when it runs diagonally across the coordinate axes. In that case, no single original feature captures it well.

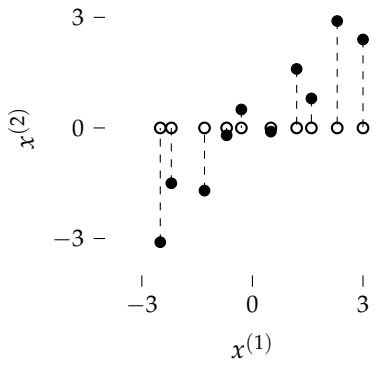


Figure 5: Naïve feature selection: dropping $x^{(2)}$ projects onto the horizontal axis. The dashed errors are large because the data varies along $y{=}x$, not along a coordinate axis. Both features have nearly equal variance, so the naïve ranking cannot decide which to keep. Notice, how in this case both features would still preserver the signal.

Principal Component Analysis

Here both features have nearly the same variance, so variance-based selection cannot decide which to keep. Dropping either axis incurs large reconstruction error. The real pattern lies along the diagonal $y{=}x$, a direction that no original feature represents. When the interesting variation runs between features, we need to construct and map to new dimensions, meaning features. That is exactly what PCA does.

The manifold hypothesis holds that high-dimensional data concentrates on a k-dimensional manifold $\mathcal{M} \subset \mathbb{R}^d$ with $k \ll d$. The goal of dimensionality reduction is to find a coordinate system for $\mathcal{M}$.

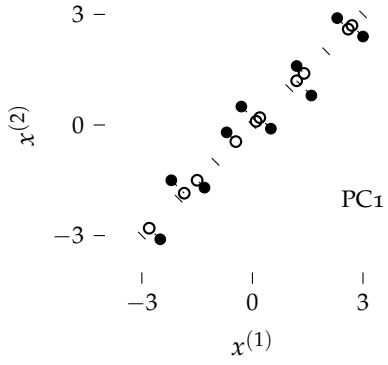


Figure 6: PCA projection onto PC1: The direction of maximum variance runs along $y{=}x$. The reconstruction errors (dashed) are tiny compared to the naïve projection.

Given centred data $X \in \mathbb{R}^{n \times d}$, where rows are samples, PCA finds a k-dimensional subspace, $k < d$, that best represents the data.

PCA finds orthogonal directions of maximum variance in the data.
; and

Algorithm 1: PCA via Eigendecomposition

Require: Centred data $X \in \mathbb{R}^{n \times d}$ (each column has zero mean, so that $X^T X/n$ is the covariance matrix; if features have different scales, standardise first by dividing each column by its standard deviation), target dimension k
Ensure: Principal components $W \in \mathbb{R}^{d \times k}$, projected data $Z \in \mathbb{R}^{n \times k}$
1: Compute covariance matrix $\Sigma = X^T X/n$
2: Compute eigendecomposition $\Sigma = V \Lambda V^T$
3: $W \leftarrow$ first k columns of V (sorted by decreasing eigenvalue)
4: $Z \leftarrow XW$ ▷ Project data onto k-dimensional subspace

THE VARIANCE MAXIMIZATION VIEW. Find the direction $w_1 \in \mathbb{R}^d$ with $\|w_1\|=1$ that maximises the variance of projected data:

$$\begin{aligned} w_1 &= \arg \max_{\|w\|=1} \operatorname{Var}(Xw) \\ &= \arg \max_{\|w\|=1} w^T \Sigma\, w, \\ \Sigma &= \tfrac{1}{n} X^T X. \end{aligned}$$

LAGRANGE MULTIPLIERS. Maximising $w^T \Sigma w$ subject to $w^T w = 1$ is a constrained optimisation problem. Introduce a multiplier λ and form the Lagrangian:

$$\mathcal{L}(w, \lambda) = w^T \Sigma\, w \; - \; \lambda\, (w^T w - 1)$$

Differentiating with respect to w and setting the gradient to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= 2\, \Sigma\, w \; - \; 2\, \lambda\, w \\ &\stackrel{!}{=} \; 0 \end{aligned}$$

which gives the eigenvalue equation:

$$\Sigma\, w = \lambda\, w$$

So w must be an eigenvector of Σ. The projected variance along any such eigenvector equals its eigenvalue:

$$\begin{aligned} w^T \Sigma\, w &= w^T (\lambda\, w) \\ &= \lambda\, w^T w \\ &= \lambda \end{aligned}$$

Maximising the variance therefore means choosing w_1 as the eigenvector with the largest eigenvalue λ_1. The second principal component is the eigenvector with the second-largest eigenvalue, orthogonal to the first, and so on. The first k principal components together define the optimal k-dimensional subspace. $k \leq d$ is a free choice.

ORTHOGONALITY of principal components follows directly: eigenvectors of a symmetric matrix corresponding to distinct eigenvalues are orthogonal.

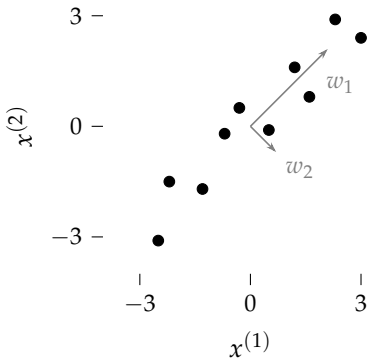


Figure 7: The two eigenvectors of Σ. w_1 points along the direction of maximum variance; w_2 is orthogonal and captures the residual spread.

COMPUTING EIGENVALUES. Eigenvalues satisfy the characteristic equation $\det(\Sigma - \lambda I) = 0$. For a 2×2 covariance matrix $\Sigma = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$:

$$\begin{aligned} \det(\Sigma - \lambda I) &= \det \begin{pmatrix} a-\lambda & b \\ b & c-\lambda \end{pmatrix} \\ &= (a-\lambda)(c-\lambda) - b^2 \\ &= \lambda^2 - (a+c)\,\lambda + (ac - b^2) \\ &\stackrel{!}{=} 0 \end{aligned}$$

GEOMETRIC INTUITION. Σ stretches space along its eigenvector directions. Subtracting λI weakens every direction by λ. When λ exactly matches the stretch along some eigenvector, $\Sigma - \lambda I$ collapses that direction to zero, the matrix becomes singular, and $\det = 0$. So the characteristic equation asks: at which scaling factors λ does Σ lose a dimension?

Applying the quadratic formula:

$$\begin{aligned} \lambda &= \frac{(a+c) \pm \sqrt{(a+c)^2 - 4(ac - b^2)}}{2} \\ &= \frac{(a+c) \pm \sqrt{(a-c)^2 + 4b^2}}{2} \end{aligned}$$

COMPUTING EIGENVECTORS. For each eigenvalue λ_i, solve $(\Sigma - \lambda_i I)\, w = 0$. From the first row of the 2×2 system:

$$\begin{aligned} (a - \lambda_i)\, w^{(1)} + b\, w^{(2)} &= 0 \\ w^{(2)} &= -\frac{a - \lambda_i}{b}\, w^{(1)} \end{aligned}$$

Any non-zero solution gives the direction; normalise to $\|w_i\|=1$ to obtain the principal component.

A SIMPLE DIAGNOSTIC reveals whether the data is compressible. Inspect the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$ of Σ. If most are near zero, the data varies in only a few directions; the remaining dimensions add volume but no information.

The scree plot (Figure 8) visualises exactly this: a steep drop tells you how many components matter.

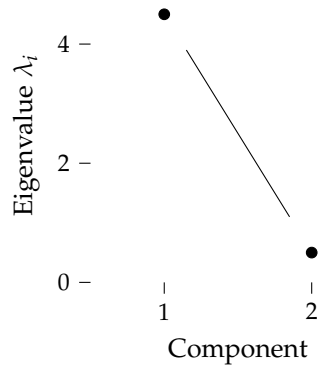


Figure 8: Scree plot for the two-component example (λ_1=4.5, λ_2=0.5). The sharp drop confirms that one direction captures most of the variance.

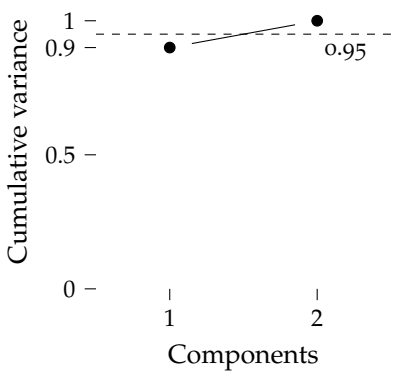


Figure 9: Cumulative explained variance: the fraction of total variance captured by the first k components is $\sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{d} \lambda_i$. Here PC1 alone explains 90%; both components are needed to exceed the rule of thumb 95% threshold (dashed).

Reconstruction Error

So far we derived PCA from variance maximisation: find the directions along which the projected data spreads the most. There is a dual view that asks the opposite question: if we compress from d dimensions to k and then decompress back to d, how much signal do we lose?

Given centred data $X \in \mathbb{R}^{n \times d}$ and an orthonormal basis $W \in \mathbb{R}^{d \times k}$ ($W^T W = I_k$), each point x_i is projected to $z_i = W^T x_i \in \mathbb{R}^k$ and reconstructed as $\hat{x}_i = W z_i = W W^T x_i$. The reconstruction error is the squared distance between each point and its reconstruction:

$$\begin{aligned} \mathcal{L}(R &= \sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2 \\ &= \sum_{i=1}^{n} \|x_i - W W^T x_i\|^2 \end{aligned}$$

The dashed error segments are short because PC1 captures the dominant direction of variation, making x and its reconstruction similar.

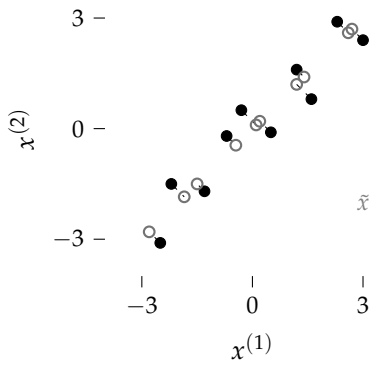


Figure 10: Reconstruction error of PCA with k=1. Each point (filled) is projected onto PC1 (the line y=x) to get its reconstruction (open circle). The dashed segments are the reconstruction errors: the perpendicular distances that PCA minimises. Compare with Figure 5, where projection onto a coordinate axis leaves much larger errors.

For $d > 2$, closed-form eigendecompositions do not exist. In practice, use the SVD: $X = U \Sigma V^T$, where the columns of V are the eigenvectors of $X^T X$ with $\lambda_i = \sigma_i^2 / n$. This avoids forming $X^T X$ explicitly and is numerically stable.

Independent Component Analysis

PCA decorrelates but does not separate sources. Uncorrelated ($\mathrm{Cov}(s_i, s_j)$=0) is weaker than independent ($p(s_i, s_j) = p(s_i)\, p(s_j)$). Given a linear mixture $X = AS$ of independent signals S, ICA recovers $S = A^{-1} X$ by maximising non-Gaussianity of the components.

t-SNE[1]

[1]

T-SNE. stands for t-distributed Stochastic Neighbour Embedding. It is optimised for one-, two- or three-dimensional visualization. t-SNE preserves local pattern; nearby points in high dimensions stay nearby in the embedding. It does this in four steps.

Algorithm 2: t-SNE. All points are updated simultaneously (batch gradient descent). The KL objective is non-convex, so there is no convergence guarantee; different random initialisations can yield different layouts.

Require: High-dimensional data X, perplexity, target dimension k

1: Compute pairwise similarities in high-D (Gaussian kernel, maps distances to $(0, 1]$):

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{l\neq i}\exp(-\|x_i - x_l\|^2/2\sigma_i^2)}$$

Symmetrize: $p_{ij} = (p_{j|i} + p_{i|j})/2n$

2: Initialize low-D embedding Y

3: **repeat**

4: Compute low-D similarities with Student-t kernel:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

5: Gradient step: minimise

$$\mathrm{KL}(P\|Q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

6: **until** convergence

PCA AND ICA ARE LINEAR. No rotation of the axes can untangle classes that live on a curved manifold. When the goal is visualization rather than preprocessing, we need methods that can bend the coordinate system to follow the data. This is what t-SNE and UMAP provide.

PERPLEXITY controls the effective number of neighbours each point considers. It sets the bandwidth σ_i of the Gaussian kernel. Typical values: 5 to 50. Higher perplexity means more global pattern is considered.

PERPLEXITY is defined as $2^{H(P_i)}$ where H is the Shannon entropy of the conditional distribution over neighbours. Think of it as the effective number of neighbours.

THE GRADIENT of $\mathrm{KL}(P\|Q)$ with respect to each embedding coordinate is

$$\frac{\partial\,\mathrm{KL}}{\partial\, y_i} = 4\sum_{j\neq i}(p_{ij} - q_{ij})\,(y_i - y_j)\,\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

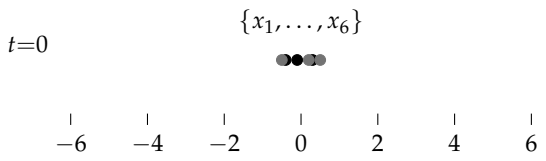
THE KL DIVERGENCE $\mathrm{KL}(P\|Q) = \sum_{ij} p_{ij}\log(p_{ij}/q_{ij})$ measures how much the low-dimensional distribution Q diverges from the high-dimensional distribution P. It is zero when $Q = P$ and asymmetric: misrepresenting a large p_{ij} with a small q_{ij} is penalised heavily, so t-SNE prioritises preserving local neighbourhoods.

Each summand acts as a pairwise force: where $p_{ij} > q_{ij}$ the force is attractive, where $p_{ij} < q_{ij}$ it is repulsive. The heavy tails of the Student-t kernel give distant pairs in Y a larger q_{ij} than a Gaussian would, so the repulsive force saturates. This prevents the crowding

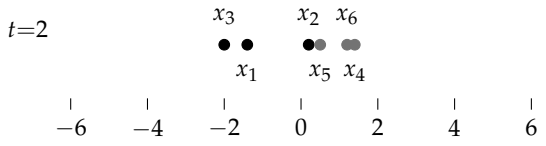
problem where all points collapse into the centre of the map. Gradient descent moves the embedding until local neighbourhoods in Y match those in X.

TRACE the four steps on our canonical six points, mapping from $\mathbb{R}^2$ to $\mathbb{R}^1$ with perplexity=2.

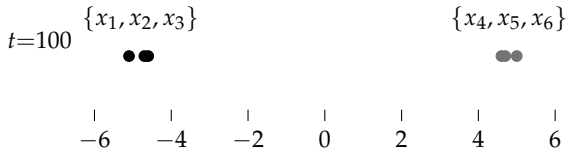
STEP 1: INITIALISE THE EMBEDDING. Each y_i is drawn independently from $\mathcal{N}(0, 10^{-4})$. At t=0 the six points land in near-random order: cluster A (black) and cluster B (grey) are completely intermixed.



STEP 2: COMPUTE HIGH-DIMENSIONAL SIMILARITIES AND BEGIN GRADIENT DESCENT. Here perplexity $= 2$. Same-cluster pairs get large similarities, cross-cluster pairs near zero; the resulting matrix P is computed once and stays fixed as the target. Gradient descent then pulls high-similarity neighbours together and pushes low-similarity points apart. By t=2 most points drift toward their cluster partners, but x_2; initialised near x_4 (see t=0), needed a few steps for its atractive forces to move through the repulsive forces of the other cluster's points.



STEP 3: CONVERGENCE. At every iteration, q_{ij} is recomputed from the current embedding distances. The forces balance and the embedding stabilises. Within each cluster the ordering mirrors similarity: x_1/x_2 (highest p_{12}=0.098) land closest in A, x_5/x_6 (p_{56}=0.089) in B.



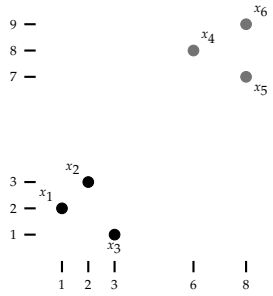


Figure 11: The canonical six points (again)). Cluster A: x_1=(1, 2), x_2=(2, 3), x_3=(3, 1). Cluster B: x_4=(6, 8), x_5=(8, 7), x_6=(8, 9).

INTERPRETING T-SNE. Cluster sizes and inter-cluster distances are meaningless. t-SNE equalises local densities and preserves only neighbourhood structure. Results vary with the random seed; always run multiple times. Use t-SNE for visualization only, never as preprocessing for downstream models.

UMAP

UMAP stands for Uniform Manifold Approximation and Projection . It has theoretical foundations in Riemannian geometry and algebraic topology.

UMAP BRINGS THREE PRACTICAL STRENGTHS.

SCALE. UMAP operates on a sparse k-nearest-neighbour graph instead of all n^2 pairs, giving $O(n \log n)$ complexity in practice, large enough for million-point datasets.

GLOBAL STRUCTURE. UMAP optimises a cross-entropy loss that penalises both pulling true neighbours apart *and* pushing true non-neighbours together, so the embedding preserves large-scale arrangement alongside local clusters.

REPRODUCIBILITY. UMAP initialises the embedding with a spectral embedding of the graph Laplacian, so repeated runs on the same data converge to similar layouts rather than producing arbitrary rotations each time.

Algorithm 3: UMAP. Like t-SNE, all points are updated simultaneously and the objective is non-convex. The spectral initialisation makes results more reproducible than t-SNE's random start, but does not guarantee a global optimum. Key parameters: `n_neighbors` (typically 15–200) controls local versus global structure, analogous to perplexity in t-SNE; `min_dist` (typically 0.1–0.5) controls how tightly points cluster in the embedding.

Require: High-dimensional data X, number of neighbours k, target dimension d, minimum distance δ

1: Build a weighted k-nearest-neighbour graph. For each x_i, set $\rho_i = \min_{j \in \mathrm{NN}(i)} \|x_i - x_j\|$ and choose σ_i so that

$$\sum_{j \in \mathrm{NN}(i)} \exp\big(-(\|x_i - x_j\| - \rho_i)/\sigma_i\big) = \log_2 k$$

2: Compute directed edge weights:

$$w_{j|i} = \exp\big(-\max(0,\, \|x_i - x_j\| - \rho_i)/\sigma_i\big)$$

Symmetrise: $w_{ij} = w_{j|i} + w_{i|j} - w_{j|i} \cdot w_{i|j}$

3: Initialise low-D coordinates Y via spectral embedding

4: **repeat**

5: Compute low-D weights:

$$v_{ij} = \big(1 + a\,\|y_i - y_j\|^{2b}\big)^{-1}$$

6: Gradient step: minimise

$$\mathcal{L} = \sum_{ij} \Big[w_{ij} \log \frac{w_{ij}}{v_{ij}} + (1 - w_{ij}) \log \frac{1 - w_{ij}}{1 - v_{ij}} \Big]$$

7: **until** convergence

Examples & Exercises

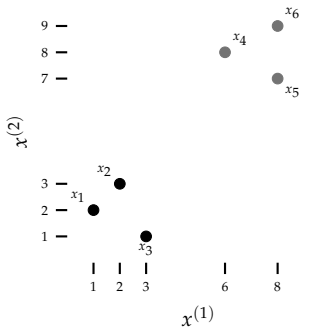


Figure 12: The canonical six points from Chapters 3 and 4. Cluster A: $x_1{=}(1,2)$, $x_2{=}(2,3)$, $x_3{=}(3,1)$. Cluster B: $x_4{=}(6,8)$, $x_5{=}(8,7)$, $x_6{=}(8,9)$.

Computing by hand builds the geometric intuition that plotting the output never can. Step through the arithmetic slowly.

Given the canonical six points $x_1{=}(1,2)$, $x_2{=}(2,3)$, $x_3{=}(3,1)$, $x_4{=}(6,8)$, $x_5{=}(8,7)$, $x_6{=}(8,9)$, centre the data, compute the covariance matrix Σ, find its eigenvalues and eigenvectors, and determine what fraction of variance PC1 explains.

Centering. PCA requires zero-mean data. Compute the sample mean:

$$\begin{aligned}\bar{x} &= \frac{1}{6}\sum_{i=1}^{6} x_i \\ &= \left(\frac{28}{6},\ \frac{30}{6}\right) \\ &= \left(\frac{14}{3},\ 5\right)\end{aligned}$$

Subtract $\bar{x}$ from each point. Here is the full calculation for x_1:

$$\begin{aligned}\tilde{x}_1 &= x_1 - \bar{x} \\ &= \left(1 - \tfrac{14}{3},\ 2 - 5\right) \\ &= \left(-\tfrac{11}{3},\ -3\right)\end{aligned}$$

Compute the remaining five yourself, then verify against the table below.

Point	**Original**	**Centred $\tilde{x}_i$**
x_1	$(1,\ 2)$	$(-11/3,\ -3)$
x_2	$(2,\ 3)$	$(-8/3,\ -2)$
x_3	$(3,\ 1)$	$(-5/3,\ -4)$
x_4	$(6,\ 8)$	$(4/3,\ 3)$
x_5	$(8,\ 7)$	$(10/3,\ 2)$
x_6	$(8,\ 9)$	$(10/3,\ 4)$

Table 1: Centred data. The second coordinate centres cleanly ($\bar{x}^{(2)}{=}5$); the first introduces thirds ($\bar{x}^{(1)}{=}14/3$).

THE COVARIANCE MATRIX. $\Sigma = \tilde{X}^T\tilde{X}/n$. Here is the full calculation for Σ_{11}:

$$\begin{aligned}\Sigma_{11} &= \frac{1}{6}\sum_{i=1}^{6}\left(\tilde{x}_i^{(1)}\right)^2 \\ &= \frac{1}{6}\left[\frac{121}{9}+\frac{64}{9}+\frac{25}{9}+\frac{16}{9}+\frac{100}{9}+\frac{100}{9}\right] \\ &= \frac{1}{6}\cdot\frac{426}{9} \\ &= \frac{71}{9} \\ &\approx 7.89\end{aligned}$$

Compute Σ_{12} and Σ_{22} yourself. The full matrix:

$$\Sigma = \begin{pmatrix}71/9 & 47/6\\ 47/6 & 29/3\end{pmatrix} \approx \begin{pmatrix}7.89 & 7.83\\ 7.83 & 9.67\end{pmatrix}$$

$\Sigma_{12} = 47/6 \approx 7.83$: both features are strongly positively correlated, as expected from the scatter plot where both coordinates increase together from cluster A to cluster B.

EIGENVALUES. Apply the formula from the theory section with $a = 71/9$, $b = 47/6$, $c = 29/3$:

$$\begin{aligned}a + c &= \frac{71}{9}+\frac{29}{3} \\ &= \frac{158}{9} \\ &\approx 17.56\end{aligned}$$

$$\begin{aligned}(a-c)^2 + 4b^2 &= \left(\frac{-16}{9}\right)^2 + 4\left(\frac{47}{6}\right)^2 \\ &= \frac{256}{81}+\frac{2209}{9} \\ &= \frac{20137}{81} \\ &\approx 248.6\end{aligned}$$

So:

$$\begin{aligned}\lambda &= \frac{17.56 \pm \sqrt{248.6}}{2} \\ &\approx \frac{17.56 \pm 15.77}{2}\end{aligned}$$

$\lambda_1 \approx 16.66$ and $\lambda_2 \approx 0.89$.

EIGENVECTOR FOR $\lambda_1 \approx 16.66$. Solve $(a - \lambda_1)\, w^{(1)} + b\, w^{(2)} = 0$:

$$\begin{aligned}(7.89 - 16.66)\, w^{(1)} + 7.83\, w^{(2)} &= 0 \\ -8.77\, w^{(1)} + 7.83\, w^{(2)} &= 0 \\ w^{(2)} &= \frac{8.77}{7.83}\, w^{(1)} \\ &\approx 1.12\, w^{(1)}\end{aligned}$$

$$\begin{aligned}\|w_1\| &= \sqrt{1^2 + 1.12^2} \\ &= \sqrt{2.25} \\ &= 1.50, \\ w_1 &\approx \frac{1}{1.50}\begin{pmatrix}1\\1.12\end{pmatrix} \\ &\approx \begin{pmatrix}0.67\\0.75\end{pmatrix}.\end{aligned}$$

EIGENVECTOR FOR $\lambda_2 \approx 0.89$. Solve $(a - \lambda_2)\, w^{(1)} + b\, w^{(2)} = 0$:

$$\begin{aligned}7.00\, w^{(1)} + 7.83\, w^{(2)} &= 0 \\ w^{(2)} &\approx -0.89\, w^{(1)}\end{aligned}$$

Normalising: $w_2 \approx \begin{pmatrix}0.75\\-0.67\end{pmatrix}$.

Verify orthogonality: $w_1^T w_2 = (0.67)(0.75) + (0.75)(-0.67) = 0$. ✓

EXPLAINED VARIANCE BY PC1:

$$\begin{aligned}\frac{\lambda_1}{\lambda_1 + \lambda_2} &\approx \frac{16.66}{17.56} \\ &\approx 95\%\end{aligned}$$

GEOMETRIC INTERPRETATION. PC1 points roughly along the direction from cluster A (bottom left) to cluster B (top right). The two clusters that are visible in two dimensions can be separated using a single principal component.

PROJECT AND RECONSTRUCT. Using the eigenvectors from the previous exercise, project each of the six centred points onto PC1, reconstruct them in the original space, and compute the reconstruction error.

PROJECTION ONTO PC1. For each centred point $\tilde{x}_i$, the coordinate on PC1 is $z_i = \tilde{x}_i^T w_1$. Here is the full calculation for x_1:

$$\begin{aligned} z_1 &= \tilde{x}_1^T w_1 \\ &= (-3.67)(0.67) + (-3)(0.75) \\ &= -2.46 - 2.25 \\ &= -4.71 \end{aligned}$$

Compute z_2 through z_6 yourself, then verify against the table below.

RECONSTRUCTION. Each projected point maps back to 2D via $\hat{x}_i = z_i \cdot w_1$. For x_1:

$$\begin{aligned} \hat{x}_1 &= -4.71 \cdot \begin{pmatrix} 0.67 \\ 0.75 \end{pmatrix} \\ &= \begin{pmatrix} -3.16 \\ -3.53 \end{pmatrix} \end{aligned}$$

RECONSTRUCTION ERROR:

$$\begin{aligned} \|\tilde{x}_1 - \hat{x}_1\|^2 &= (-3.67 + 3.16)^2 + (-3.00 + 3.53)^2 \\ &= (-0.51)^2 + (0.53)^2 \\ &= 0.26 + 0.28 \\ &= 0.54 \end{aligned}$$

Point	**Centred** $\tilde{x}_i$	z_i	**Reconstructed** $\hat{x}_i$	$\|\tilde{x}_i - \hat{x}_i\|^2$
x_1	$(-3.67, -3)$	-4.71	$(-3.16, -3.53)$	0.54
x_2	$(-2.67, -2)$	-3.29	$(-2.20, -2.47)$	0.44
x_3	$(-1.67, -4)$	-4.12	$(-2.76, -3.09)$	2.02
x_4	$(1.33, 3)$	3.14	$(2.10, 2.36)$	1.01
x_5	$(3.33, 2)$	3.73	$(2.50, 2.80)$	1.33
x_6	$(3.33, 4)$	5.23	$(3.50, 3.92)$	0.04
			Total	**5.38**

Table 2: PCA projection onto PC1 and reconstruction for the canonical six points. The total error $\approx n \cdot \lambda_2 = 6 \times 0.89 = 5.34$ (the small difference is rounding in intermediate steps). Point x_3 has the largest error because it sits furthest from the PC1 direction; x_6 has almost none because it lies nearly on PC1.

NAÏVE FEATURE SELECTION VS PCA. Using the same six centred points, compare the reconstruction error of naïve feature selection with PCA.

DROP $x^{(2)}$ (KEEP $x^{(1)}$). The reconstruction is $\hat{x}_i = (\tilde{x}_i^{(1)}, 0)$. For x_1:

$$\begin{aligned}\|\tilde{x}_1 - \hat{x}_1\|^2 &= (-3.67 - (-3.67))^2 + (-3 - 0)^2 \\ &= 0 + 9 \\ &= 9\end{aligned}$$

DROP $x^{(1)}$ (KEEP $x^{(2)}$). The reconstruction is $\hat{x}_i = (0, \tilde{x}_i^{(2)})$. For x_1:

$$\begin{aligned}\|\tilde{x}_1 - \hat{x}_1\|^2 &= (-3.67 - 0)^2 + (-3 - (-3))^2 \\ &= 13.47 + 0 \\ &= 13.47\end{aligned}$$

Compute the remaining errors yourself, then verify against the table.

Point	**Drop $x^{(2)}$**	**Drop $x^{(1)}$**	**PCA (PC1)**
x_1	9.00	13.44	0.54
x_2	4.00	7.11	0.44
x_3	16.00	2.78	2.02
x_4	9.00	1.78	1.01
x_5	4.00	11.11	1.33
x_6	16.00	11.11	0.04
Total	**58.00**	**47.33**	**5.38**

Table 3: Reconstruction error comparison. The naïve variance ranking correctly picks $x^{(2)}$ ($\mathrm{Var}(x^{(2)}) \approx 9.67 > \mathrm{Var}(x^{(1)}) \approx 7.89$), yet PCA still reduces the error by a factor of ≈ 9.

The naïve approach gets the ranking right: $x^{(2)}$ does carry more variance than $x^{(1)}$. But neither coordinate axis aligns with the direction the data actually varies along. PCA constructs that direction from the data and achieves roughly nine times less error.

THE CODE EXCERCISES LOAD 1 000 handwritten digits from MNIST: 784 pixels each, ten classes. In code, you will apply three preprocessing stages: discard uninformative pixels via a variance threshold, centre the data by subtracting the per-feature mean (without centring, PC1 aligns with the mean intensity rather than with the dominant direction of variation), and eigendecompose the covariance matrix to project onto the top two components. Then compare the PCA projection with t-SNE (sweeping perplexity) and UMAP (sweeping neighbourhood size).

CODE is provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

WHAT TO OBSERVE. PCA preserves global geometry: The ten digit classes form broad, overlapping regions whose distances reflect the original space. Because PCA is linear, it cannot untangle classes that

live on a curved manifold; overlap in the 2D plot is a direct consequence. t-SNE compresses each class into a tight ball, but distances between balls are arbitrary and change with the random seed. UMAP produces similarly tight clusters while retaining more of the global arrangement. Both nonlinear methods are sensitive to their main hyperparameter; the sweep makes this visible.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What does it mean for data to have low intrinsic dimensionality?
- Why does PCA correspond to an eigenvector decomposition of the covariance matrix?
- What is the difference between an uncorrelated and an independent component?
- Why does t-SNE use a Student-t kernel in the low-dimensional space but a Gaussian in the high-dimensional space?
- When would you prefer UMAP over t-SNE, and when would you use PCA instead of either?
- What does the perplexity parameter actually control, and what happens when it is too low or too high?
- Why should you never use t-SNE or UMAP embeddings as features for a downstream classifier?
- How does the curse of dimensionality motivate dimensionality reduction before running K-Means?

RECAP of Key Concepts:

- PCA finds the orthogonal directions of maximum variance via eigendecomposition of the covariance matrix; SVD is the numerically preferred implementation.
- The explained-variance ratio guides the choice of k; retain enough components to exceed a threshold such as 95%.
- t-SNE and UMAP preserve local neighbourhood patterns for visualization; cluster sizes and between-cluster distances in their output are not interpretable.

- ICA finds statistically independent sources by maximising non-Gaussianity, going beyond the uncorrelated directions found by PCA.
- No non-linear method should be used as a preprocessing step for downstream learning; use PCA or SVD for that purpose.

PCA AND ICA CAN ONLY ROTATE AND SCALE AXES. When the underlying structure is nonlinear, no linear map can recover it. t-SNE and UMAP solve this for visualization, but neither learns a parametric encoder that can be applied to new data or reused as a preprocessing step.

AUTOENCODERS learn both the compression and the reconstruction from data. The encoder is a neural network that maps x_i to a low-dimensional code z_i; the decoder maps z_i back to $\hat{x}_i$. Variational Autoencoders go one step further: the encoder outputs not a single point but a mean μ and a variance σ^2, turning the code into a distribution from which new data can be sampled.

THE FUNDAMENTAL LIMIT of PCA and ICA is linearity; they cannot capture curved structure. t-SNE and UMAP learn nonlinear embeddings, but are designed for visualization, not for learning a reusable mapping.

TEASER. If we replace the fixed eigenvector projection with a learned, nonlinear encoder, can we get a compact representation that generalises to unseen inputs?

FEEDBACK