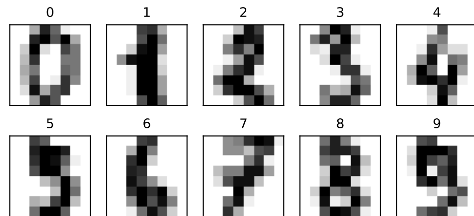
How do you *look* at 64-dimensional data?

A handwritten digit is $8 \times 8 =$ **64** numbers; gene expression ~20,000; a text embedding ~768. You cannot plot that.

Dimensionality reduction: replace many features with a *few* new ones that keep most of the structure.

Bridge from clustering — both are **unsupervised** (no y): clustering groups the **rows** (samples), DR compresses the **columns** (features).



Running dataset: sklearn's 1797 digits (8×8 grayscale, labels 0–9). Each image = 64 pixel values; we will squeeze those 64 down to 2 to *see* the data.

Why reduce dimensions?

- **Visualization** — see high-dimensional data in 2-D.
- **Preprocessing / compression** — faster models, less storage, fewer features before clustering or a classifier.
- **Denoising** — drop tiny-variance directions that are *often* just noise.

And a deeper reason, next: in high dimensions, distance itself stops working.

The curse of dimensionality

Predict-first: in 100 dimensions, how much closer is your *nearest* point than your *farthest*?

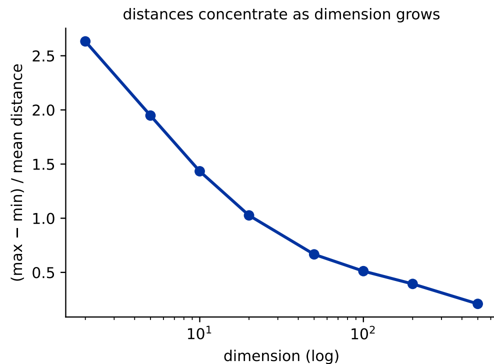
The curse of dimensionality

Predict-first: in 100 dimensions, how much closer is your *nearest* point than your *farthest*?

Almost the same. As dimension grows, **distances concentrate** — near and far blur together, so distance-based methods (k-means, kNN, DBSCAN...) degrade.

Reduce first, and distances mean something again.

Full treatment in our homework: `math/30_curse_of_dimensionality.qmd`.



Random points; relative contrast vanishes (Beyer et al. 1999).

Outline

PCA: the idea and the math

PCA in practice

Nonlinear DR for visualization: t-SNE & UMAP

Choosing a method

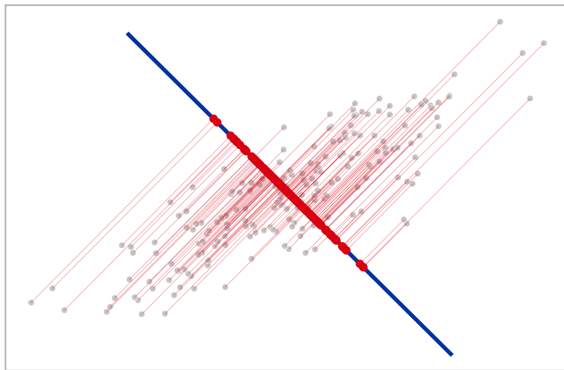
Connections and wrap-up

Principal Component Analysis

The linear workhorse: new axes along the directions of greatest variance.

PCA: keep the directions of greatest spread

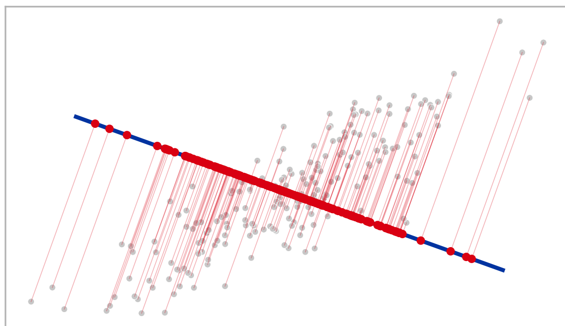
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the top few such axes.



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means. Keep the **top**-k (here 2-D→1-D; for 64-D digits we keep

PCA: keep the directions of greatest spread

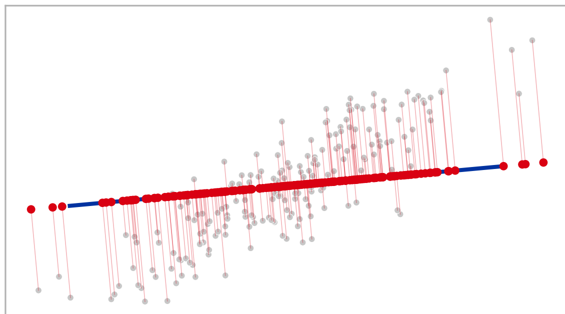
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance** $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$ — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the top few such axes.



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means. Keep the **top**-k (here 2-D→1-D; for 64-D digits we keep 2).

PCA: keep the directions of greatest spread

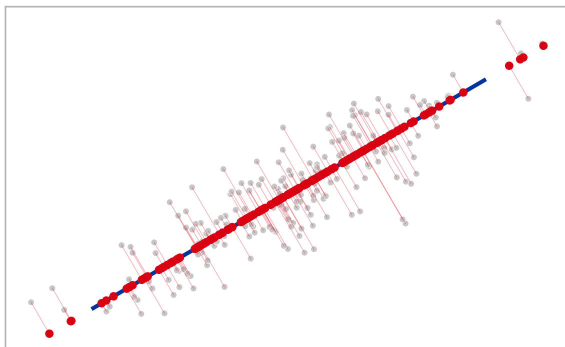
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the top few such axes.



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means. Keep the **top**-k (here 2-D→1-D; for 64-D digits we keep 2).

PCA: keep the directions of greatest spread

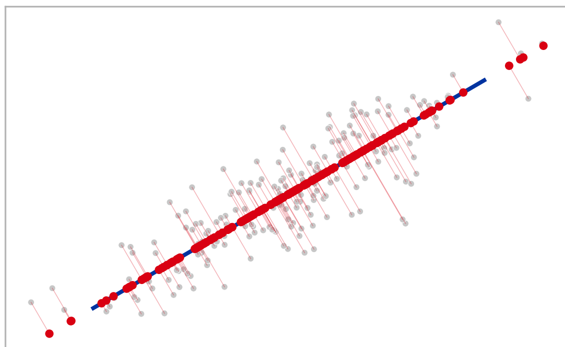
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \mathbf{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the top few such axes.



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means. Keep the **top**-k (here 2-D→1-D; for 64-D digits we keep 2).

PCA: keep the directions of greatest spread

Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the top few such axes.



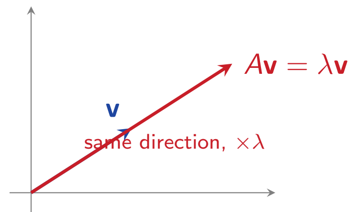
PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means. Keep the **top**-k (here 2-D→1-D; for 64-D digits we keep 2).

Refresher: eigenvectors and eigenvalues

An **eigenvector v** of a matrix A keeps its *direction* when A acts on it; it only gets scaled by the **eigenvalue** λ:

$$A\mathbf{v} = \lambda\mathbf{v}.$$

A **symmetric** matrix — like a covariance — has real eigenvalues and **orthogonal** eigenvectors. *That is why the principal axes come out perpendicular.*



Derivation (1): maximize variance

Step 0 — center the data (subtract each feature's mean). Without it, $X^\top X$ measures spread about the *origin*, not about the data.

Variance of the data projected onto a unit direction $\mathbf{w}$ is $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$ ($\boldsymbol{\Sigma}$ = covariance of the centered data). Maximize it subject to $\|\mathbf{w}\| = 1$; a **Lagrange multiplier** gives:

$$\max_{\mathbf{w}} \ \mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w} \ \text{ s.t. } \|\mathbf{w}\| = 1 \implies \boldsymbol{\Sigma} \mathbf{w} = \lambda \mathbf{w}.$$

The principal directions are the **eigenvectors of the covariance matrix**; each eigenvalue λ is the **variance captured** along that direction.

Derivation (2): explained variance

- Eigenvalues sorted **descending** $\Rightarrow$ principal components come out **ordered by variance**; take the top k eigenvectors.
- **Explained-variance ratio** of component i is $\lambda_i / \sum_j \lambda_j$ (e.g. first PC of the digits $\approx 12\%$; first two $\approx 22\%$).

Two views, one answer. Maximizing variance **is** minimizing **reconstruction error**: projecting onto the top-k axes and lifting back loses the least. That is the basis for compression and denoising later.

The SVD connection

In practice PCA is computed by the **singular value decomposition** of the centered data:

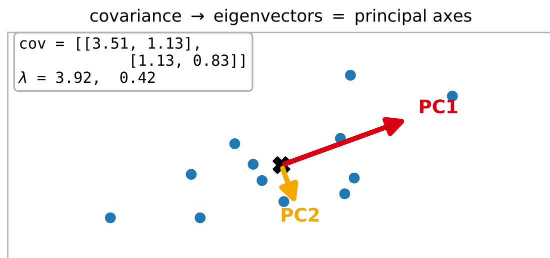
$$X_{\text{centered}} = USV^\top, \qquad \text{columns of } V = \text{principal components.}$$

- More numerically stable than forming the covariance matrix explicitly.
- **TruncatedSVD** (a.k.a. *LSA* for text) skips centering, so it works on **sparse** data — centering would destroy the sparsity.

PCA by hand (2-D)

1. Center the points (subtract the mean, the **X**).
2. Form the 2×2 **covariance** matrix.
3. Its **eigenvectors** are the principal axes; **eigenvalues** λ are the variances (arrow length $\propto \sqrt{\lambda}$).

PC1 captures most of the spread; PC2 is perpendicular and small.



PCA in practice

Scaling, how many components to keep, what they mean, and where PCA bites.

Centering vs scaling

- PCA **always centers** (scikit-learn does it for you) — mandatory.
- PCA does **not standardize** — that is your choice.

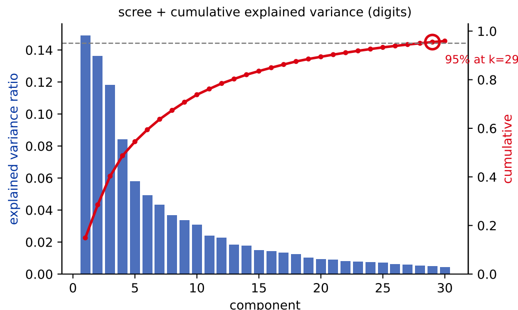
Trap: variance is scale-dependent, so an unscaled large-range feature (income in 10,000s) hijacks the first component. **Standardize** unless your features already share units (like pixel intensities).

How many components to keep?

Scree plot: explained variance per component (it drops fast).

Cumulative explained variance: keep enough components to reach, say, **95%**.

On the digits, ∼29 of 64 components already cover 95% of the variance.



Reading a biplot

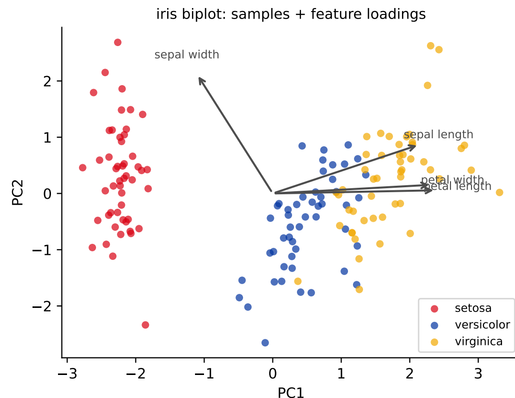
A **biplot** shows two things on the same PC1–PC2 axes:

- **dots** = the samples projected onto PC1/PC2 (colored by species here);
- **arrows** = the original features (their *loadings*).

How to read an arrow:

- **direction** = the way that feature increases;
- **length** = how strongly it drives these two PCs;
- arrows pointing the **same way** = correlated features.

So petal length & width point together (correlated) and separate the species along PC1; sepal width points elsewhere. *Components are combinations — only partly interpretable.*



Reconstruction: compression and denoising

Project to k scores, then **lift back** to the original space:

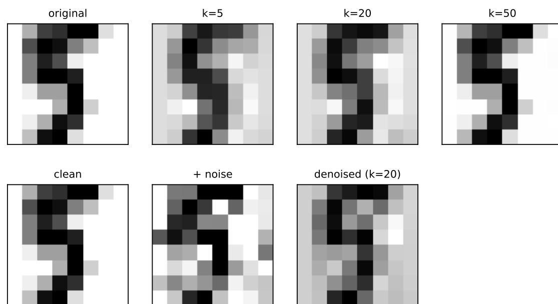
$$\mathbf{z} = W_k^\top (\mathbf{x} - \bar{\mathbf{x}}), \qquad \hat{\mathbf{x}} = \bar{\mathbf{x}} + W_k\, \mathbf{z}.$$

"Lift back" $=$ the **mean plus a weighted sum of the kept axes**:

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \sum_{i=1}^{k} z_i\, \mathbf{v}_i$$

($\mathbf{v}_i$ the principal directions, z_i the scores). Fewer k $=$ more compression (blurrier).

Noise lives in the small-variance directions you dropped, so the rebuild is also **denoised**.



PCA pitfalls

- **Not feature selection** — components are *combinations* of all features, not a chosen subset.
- **Linear only** — it cannot unfold a curved manifold (next section).
- **Sensitive** to outliers and to feature scale.
- Components can be **hard to interpret**.

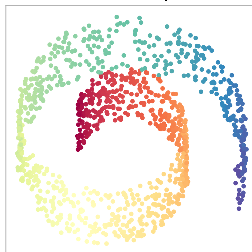
Nonlinear methods

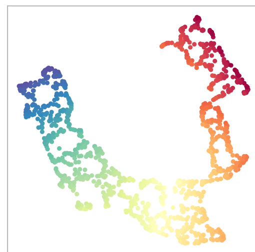
When the structure is a curved manifold, linear PCA is not enough — t-SNE and UMAP preserve *local* neighborhoods.

Why nonlinear?

PCA is **linear**: it can rotate and project, but it cannot *unfold* a curved manifold. On a “swiss roll”, PCA keeps the layers folded together.

Nonlinear methods instead preserve **local neighborhoods** — who is near whom.





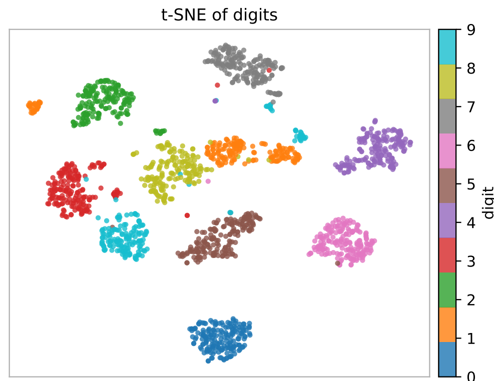
color = position along the roll.

t-SNE

Turn distances into **similarity probabilities**, high-D and low-D, then match them.

- High-D: $p_{ij} = \frac{1}{2n}(p_{j|i} + p_{i|j})$ (Gaussian neighborhoods).
- Low-D: q_{ij} from a **Student-*t*** kernel (heavy tail = fixes the "crowding" problem).
- Minimize $\mathrm{KL}(P \,\|\, Q)$ by gradient descent (iterative, *stochastic*).

Perplexity $\approx$ effective #neighbors (typ. 5–50).

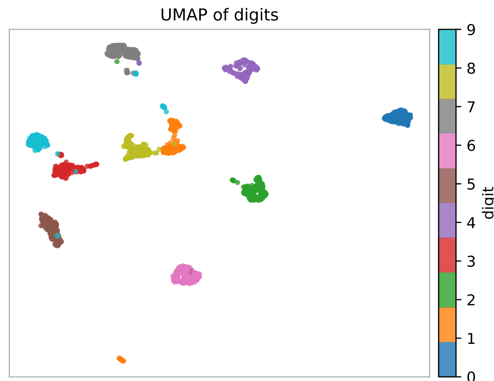


van der Maaten & Hinton (2008).

UMAP

Build a **fuzzy nearest-neighbor graph**, then find a low-D layout whose graph matches it (minimize a graph **cross-entropy**).

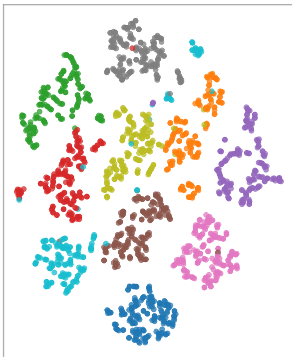
- `n_neighbors`: small = local detail, large = global shape.
- `min_dist`: how tightly points may pack.
- **Faster** than t-SNE, scales better, keeps *more* global structure.

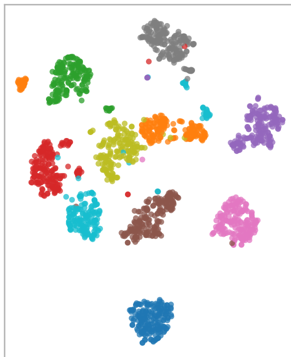


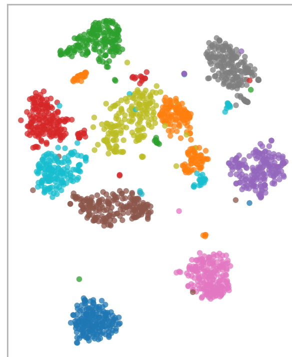
McInnes et al. (2018).

Caveats: t-SNE / UMAP can mislead

same digits, three perplexities --- the picture changes





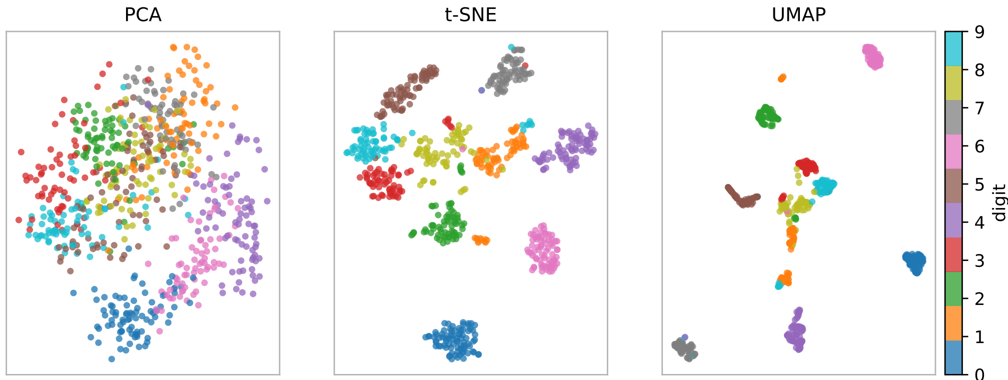


Same data, different settings → different pictures. So: **cluster sizes and between-cluster distances are not meaningful**; results are **stochastic**; they are sensitive to perplexity / `n_neighbors`; **do not feed the 2-D embedding to a downstream model** — it is for the eyes. (Wattenberg et al., distill.pub 2016.)

Which one?

PCA, t-SNE, UMAP — side by side, then a decision table.

PCA vs t-SNE vs UMAP on the digits



PCA (linear) overlaps the digits; t-SNE and UMAP separate them cleanly. PCA is for preprocessing; t-SNE / UMAP are for *looking*.

Decision table

	Linear?	Global?	Determin.?	Use it for
PCA	yes	yes	yes	preprocessing, compression, denoising
t-SNE	no	no	no	visualizing local cluster structure
UMAP	no	some	no (seeded)	visualization, faster / larger data

Rule of thumb: reach for **PCA** first (fast, cheap, reversible); use **t-SNE / UMAP** when you specifically want a 2-D *picture* of the structure.

Connections

Where DR meets the rest of the course.

A wider view (one line each)

- **Kernel PCA** — PCA in a kernel feature space: nonlinear components without leaving the PCA framework.
- **Autoencoders** — a neural network that learns a low-D "bottleneck": nonlinear, learned DR. *(Neural-networks chapter.)*
- **DR as preprocessing** — standardize $\rightarrow$ PCA $\rightarrow$ then cluster or train; this is the cluster-after-PCA pipeline from the previous deck, and it denoises too.

When NOT to reduce

DR throws information away — so do not reduce reflexively:

- PCA is **unsupervised**: it ranks directions by variance *with no knowledge of y*. The direction that separates your labels can sit in a **low-variance** component PCA discards (a thin signal axis vs. a fat nuisance axis).
- **Tree models** (forests, boosting) usually do not need it.
- Always try the model *without* DR first; add it only if it helps.

Recap

- **Why:** visualize, compress, denoise — and beat the curse of dimensionality.
- **PCA** — center, then take the top eigenvectors of the covariance (via SVD); keep enough for $\sim$95% variance; linear, fast, reversible. Great for preprocessing.
- **t-SNE / UMAP** — nonlinear, preserve local neighborhoods, excellent for 2-D *pictures* — but sizes/distances are not meaningful and they are for the eyes only.
- Reach for PCA first; don't reduce reflexively (it is unsupervised, and it loses information).

That closes unsupervised learning (clustering + DR). Next chapter: a new family of models.