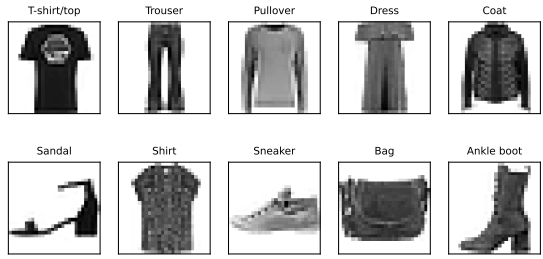
How do you *look* at 784-dimensional data?

A photo of a shirt is $28 \times 28 = \mathbf{784}$ numbers; gene expression ∼20,000; a text embedding ∼768. You cannot plot that.

Dimensionality reduction (DR): 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: Fashion-MNIST — 12,000 of its 70,000 garment photos (28×28 grayscale, 10 classes). Each image = 784 pixel values; we will squeeze those 784 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*?

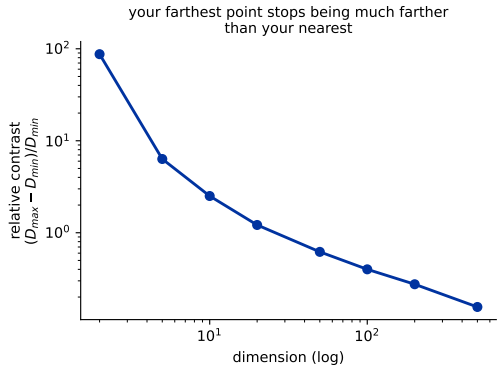
The curse of dimensionality

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

Almost the same. In 100 dimensions the farthest point is only about **40% farther** than the nearest; in 2 dimensions it is roughly **87 times** farther. Distances **concentrate**, 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`.



Relative contrast $(D_{max} - D_{min})/D_{min}$ from random query points (Beyer et al. 1999).

Outline

PCA: the idea and the math

PCA in practice

Nonlinear DR for visualization: t-SNE & UMAP

Choosing a method

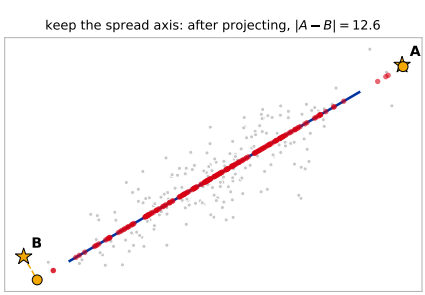
Connections and wrap-up

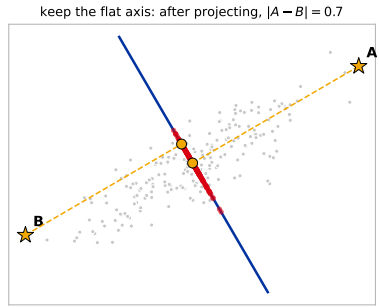
Principal Component Analysis

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

Why maximize *variance*?

Projecting always loses information: two points that differ only along a *dropped* direction land on **the same point**. So which directions can we afford to lose?

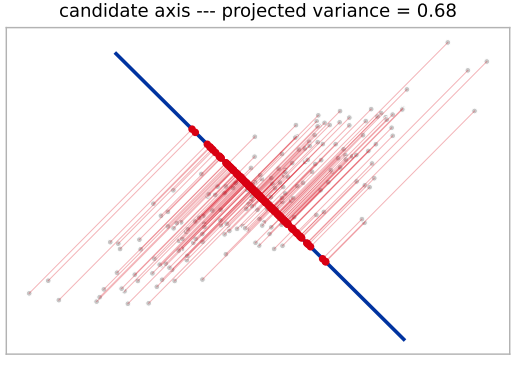




Variance measures how much the samples **disagree** along an axis. Keep the spread axis and A and B stay clearly apart; keep the flat one and they nearly collide. An axis with zero variance — the same value for every sample — can be deleted free of charge. **PCA's bet: spread = information.**

PCA: keep the directions of greatest spread

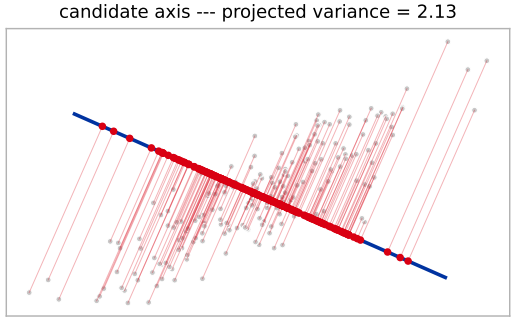
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the **top-k** such axes (here 2-D→1-D; for the 784-D garments we will keep 2).



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means.

PCA: keep the directions of greatest spread

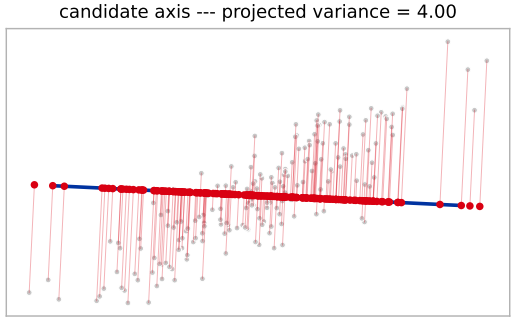
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the **top-k** such axes (here 2-D→1-D; for the 784-D garments we will keep 2).



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means.

PCA: keep the directions of greatest spread

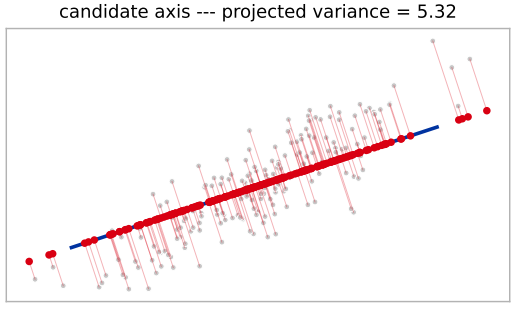
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the **top-k** such axes (here 2-D→1-D; for the 784-D garments we will keep 2).



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means.

PCA: keep the directions of greatest spread

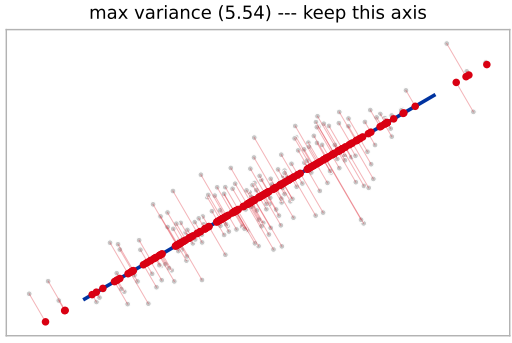
Sweep a direction through the cloud and **project** the points onto it (red dots). The **projected variance $\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}$** — how spread out those red dots are — rises and falls, peaking at the **principal axis**. Keep the **top-k** such axes (here 2-D→1-D; for the 784-D garments we will keep 2).



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means.

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-k** such axes (here 2-D→1-D; for the 784-D garments we will keep 2).



PCA does not search or iterate — the eigenvectors (next frames) give these axes directly; the sweep just shows what "maximum variance" means.

Refresher: the covariance matrix

For **centered** data (mean $= 0$):

$$\boldsymbol{\Sigma} = \tfrac{1}{n}\sum_i \mathbf{x}_i \mathbf{x}_i^\top = \tfrac{1}{n} X^\top X,$$

$$\Sigma_{jk} = \tfrac{1}{n}\sum_i x_{ij}\, x_{ik}.$$

How to read it:

- **diagonal** Σ_{jj} — the **variance** of feature j;
- **off-diagonal** Σ_{jk} — do features j and k move **together** (> 0), **opposite** (< 0), or independently (≈ 0);
- **symmetric**: $\Sigma_{jk} = \Sigma_{kj}$ — remember this for the next frame.

(Software divides by $n - 1$ instead of n; nothing below depends on which.)

The one identity PCA runs on.
Project every point onto a unit direction $\mathbf{w}$: the projected values are $\mathbf{x}_i^\top \mathbf{w}$, and their variance is

$$\begin{aligned}\mathrm{Var}(X\mathbf{w}) &= \tfrac{1}{n}\sum_i (\mathbf{x}_i^\top \mathbf{w})^2 \\ &= \mathbf{w}^\top \big(\tfrac{1}{n}\textstyle\sum_i \mathbf{x}_i \mathbf{x}_i^\top\big)\mathbf{w} = \mathbf{w}^\top \boldsymbol{\Sigma}\, \mathbf{w}.\end{aligned}$$

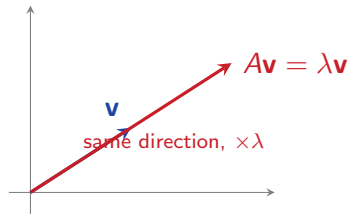
One matrix, computed *once*, gives the projected variance in *every* direction at once — which is why the sweep on the previous frame never had to touch the raw data twice.

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 $\|\mathbf{w}\| = 1$. Form the **Lagrangian** and set its gradient to zero:

$$\mathcal{L}(\mathbf{w}, \lambda) = \mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w} - \lambda(\mathbf{w}^\top \mathbf{w} - 1), \qquad \nabla_{\mathbf{w}} \mathcal{L} = 2\boldsymbol{\Sigma}\mathbf{w} - 2\lambda \mathbf{w} = 0$$

$$\implies \quad \boldsymbol{\Sigma}\mathbf{w} = \lambda\, \mathbf{w}.$$

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 (EVR)** of component i is $\lambda_i / \sum_j \lambda_j$ (first PC of the garments $\approx 29\%$; first two $\approx 47\%$).
- The scores are **uncorrelated**: $\mathrm{Cov}(Z) = W^\top \mathbf{\Sigma} W = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$. This is the precise sense in which PCA *decorrelates* — not that it "removes correlated features".

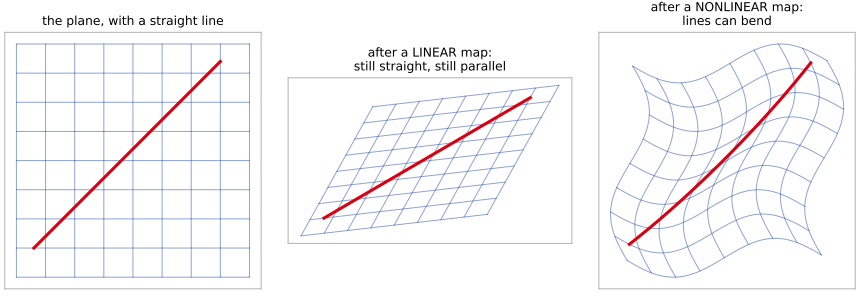
Two views, one answer. Maximizing variance **is** minimizing **reconstruction error**: projecting onto the top-k axes and lifting back loses the least. That is the basis for compression and denoising later.

What PCA does to a point — and what "linear" means

Stack the top-k eigenvectors as the columns of W_k. Every point is mapped by the **same matrix**:

$$\mathbf{z} = W_k^\top(\mathbf{x} - \bar{\mathbf{x}}), \qquad z_i = \mathbf{v}_i^\top(\mathbf{x} - \bar{\mathbf{x}}),$$

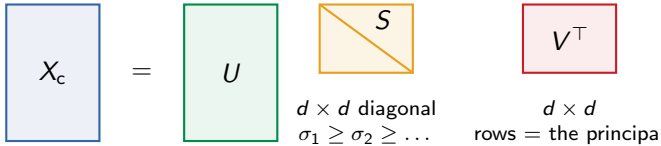
so each new feature z_i is a **fixed weighted sum** of the original features — one recipe, applied identically everywhere in space. That is what **linear** means.



A linear map can **rotate**, **stretch**, **shear** and **project** (a shadow) — grid lines stay straight, parallel stays parallel. What it can *never* do is **bend**. Hold that thought for the nonlinear section.

The SVD connection

In practice PCA is computed by the **singular value decomposition (SVD)** of the centered data — and every factor has a PCA meaning:

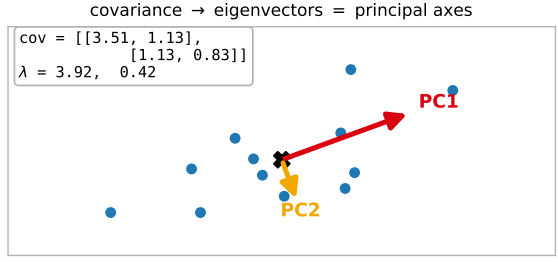


- V holds the **same eigenvectors** as the derivation, computed *without* ever forming $\mathbf{\Sigma}$ — more numerically stable.
- S is sorted, and $\lambda_i \propto \sigma_i^2$: the scree plot reads straight off the diagonal. The **scores** are $Z = US$ — what `fit_transform` returns.
- **TruncatedSVD** (a.k.a. *latent semantic analysis, 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.

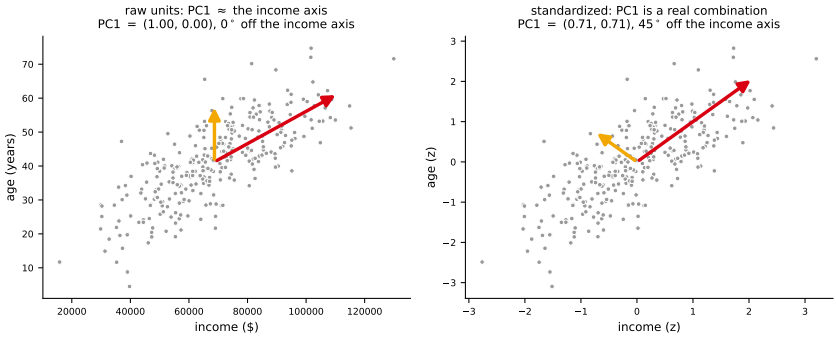


PCA in practice

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.



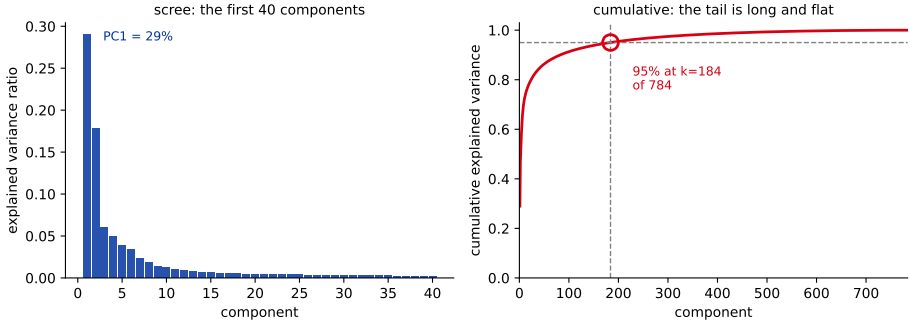
Trap: variance is scale-dependent. In dollars, PC1 is (1.00, 0.00) — literally the income axis, and age contributes nothing. Standardized, PC1 is (0.71, 0.71): the real shared direction. **Standardize** unless your features already share units (like pixel intensities, which is why we do not standardize the garments).

How many components to keep?

Predict-first: the garments are 784-D and the first PC alone is 29%. How many components do you need to reach **95%**?

How many components to keep?

Predict-first: the garments are 784-D and the first PC alone is 29%. How many components do you need to reach **95%**?



184 — almost a quarter of them, not a handful. **Scree** (left): variance per component — read its elbow like k-means' inertia elbow. **Cumulative** (right): the tail is long and flat, and that is where the detail lives.

When PCA feeds a model: do not pick k by a variance threshold — tune it by CV on the *downstream* metric, with the PCA fitted *inside* the pipeline on training folds only (the leakage rule again).

Reading a biplot

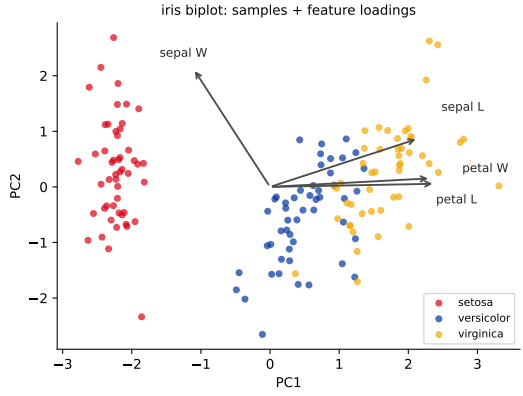
A **biplot** shows two things on the same PC1–PC2 axes:

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

How to read an arrow:

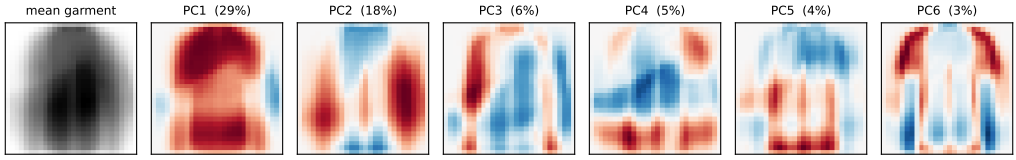
- **direction** = the way that feature increases;
- **length** = how strongly it drives these two PCs;
- arrows pointing the **same way** = correlated features.

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



The principal directions, drawn as images

Each $\mathbf{v}_i$ lives in the *same* 784-dimensional pixel space as the garments — so every component **is** a 28×28 image, and we can simply look at it.



Red and blue = pixels the component pushes in *opposite* directions; a sample's score z_i says how much of the pattern to mix in, with sign. PC1 (29%) is roughly "how much of the frame the garment fills"; PC2 contrasts trouser-like verticals with broad tops; the later, smaller components draw ever finer shape details.

Reconstruction (next frame) is literally: start from the mean garment and **mix in** z_i units of each of these pictures. In Project 1 you will draw these for faces — the famous *eigenfaces*.

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**:

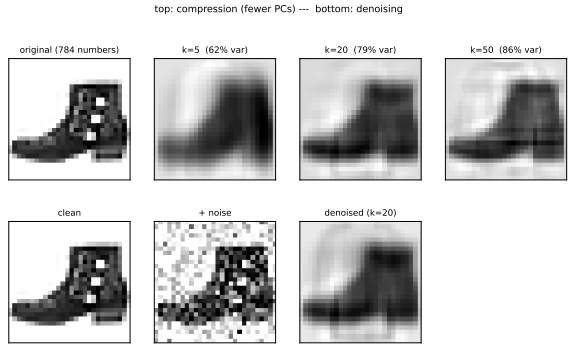
$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \textstyle\sum_{i=1}^{k} z_i\,\mathbf{v}_i$$

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

784 numbers → **50** and the garment is still clearly itself.

Callback: the blurriness at k=5 is exactly the variance the scree plot said we threw away. The squared reconstruction error is $\sum_{i>k}\lambda_i$ (the dropped eigenvalues); as a *fraction* of the total, that is 1− cumulative EVR.

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. DR is feature **extraction** (build new features); the other family, feature **selection**, keeps a subset of the originals (drop low-variance or correlated columns, L1, . . .). Extraction gives you better axes; selection gives you back your original, readable ones.
- **Linear only** — it cannot unfold a curved manifold (next section).
- **Sensitive** to outliers and to feature scale.
- Components can be **hard to interpret**.

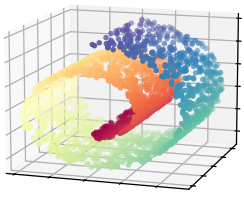
Nonlinear methods

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

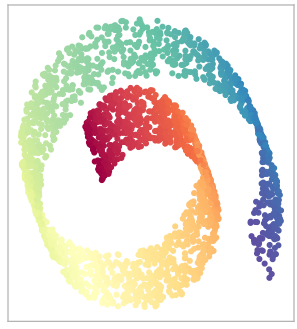
Why nonlinear?

PCA is **linear**: remember the grid — it can rotate, stretch and project, but it can never *bend*, so it cannot *unfold* a curved manifold. Nonlinear methods instead preserve **local neighborhoods** — who is near whom.

the data: a 2-D sheet
rolled up in 3-D



PCA (linear): a shadow ---
the layers stay overlapped



UMAP: one sweep of colour,
end to end, nothing overlapping

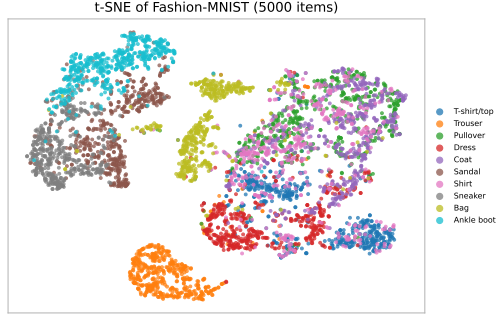
Color = position along the roll. **PCA** gives you a *shadow*: the roll stays rolled, so points on *adjacent turns* sit side by side even though they are far apart along the sheet. **UMAP** sweeps the colors once, end to end. It did not lay the sheet out straight — it had no reason to. Topology preserved, geometry not.

t-SNE: t-distributed Stochastic Neighbor Embedding

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; its heavy tail fixes *crowding* — 2-D lacks room for all the mid-distance neighbors, so they may sit farther out penalty-free.
- Minimize $\mathrm{KL}(P \,\|\, Q)$ by gradient descent (iterative, *stochastic*).

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



Sandal, sneaker, ankle boot are each other's nearest classes here — so are pullover, coat, shirt. **t-SNE recovered the semantics from raw pixels**, no labels.

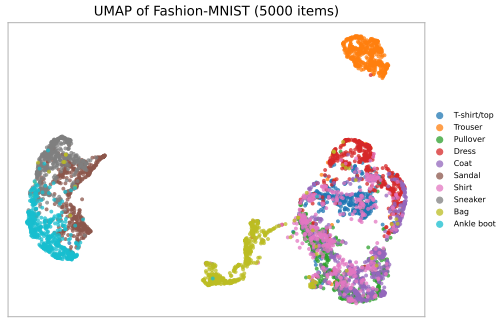
van der Maaten & Hinton (2008).

UMAP

Uniform Manifold Approximation and Projection (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.

This frame is *what UMAP gives you*. **How it works** — the fuzzy graph, ρ_i and σ_i, the cross-entropy — is the whole of the next lecture.



McInnes et al. (2018).

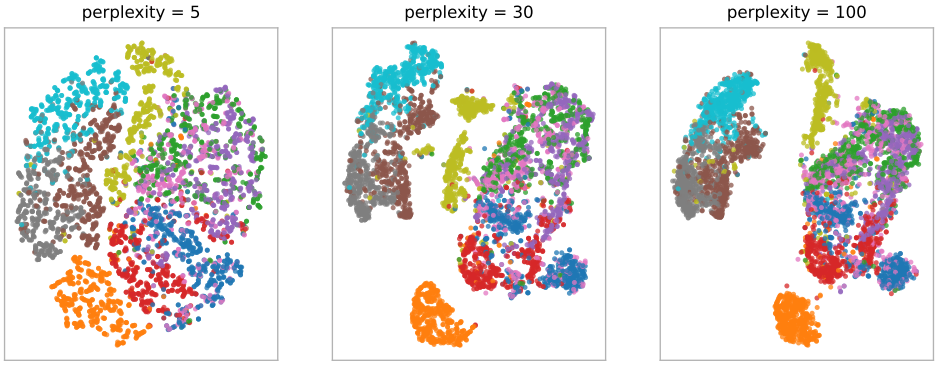
Caveats: t-SNE / UMAP can mislead

Predict-first: same data, same algorithm, one hyperparameter moved. Will the picture be the same?

Caveats: t-SNE / UMAP can mislead

Predict-first: same data, same algorithm, one hyperparameter moved. Will the picture be the same?

same data, three perplexities --- the picture changes

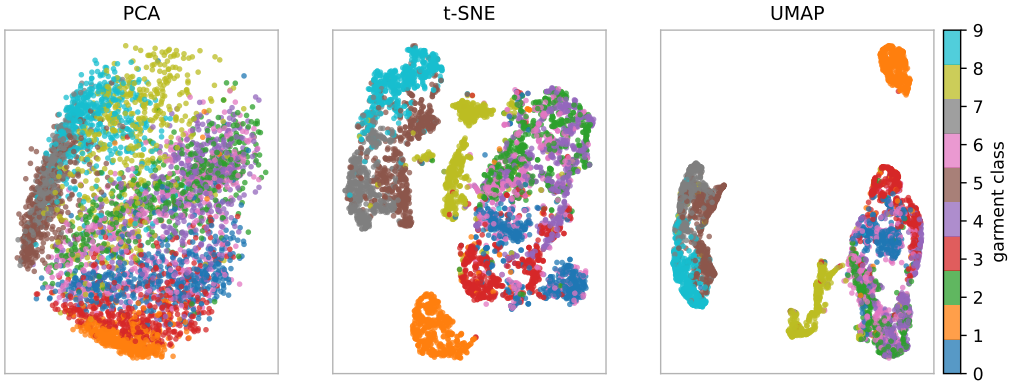


So: **cluster sizes and between-cluster distances are not meaningful**; results are **stochastic**; **do not feed the 2-D embedding to a downstream model** — it is for the eyes. (Wattenberg et al., distill.pub 2016.)

Which one?

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

PCA vs t-SNE vs UMAP on the garments



PCA (linear) piles the classes on top of each other; t-SNE and UMAP pull them apart. PCA is for preprocessing; t-SNE / UMAP are for *looking*.

In practice the nonlinear methods here were run on the top 50 PCs, not the raw 784 pixels: PCA first as a denoiser and accelerator, then t-SNE / UMAP for the picture. That stack is the normal recipe, not a shortcut.

Decision table

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

New points? — can a fitted model embed data it has never seen? **PCA**: `transform`, exact. **UMAP**: `transform`, approximate. **t-SNE**: no — it learns positions, not a map, so a new point means refitting everything. One more reason the 2-D picture cannot be a feature for a model.

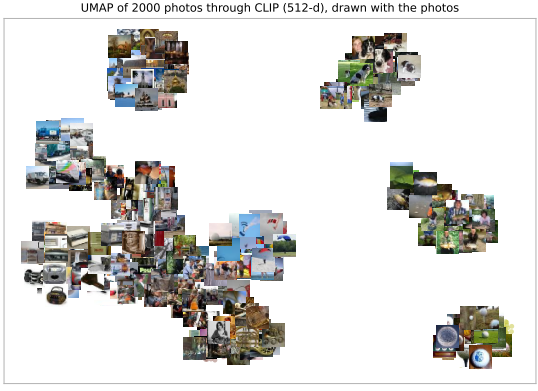
Rule of thumb: reach for **PCA** first (fast, cheap, reversible); use **t-SNE / UMAP** when you specifically want a 2-D *picture* of the structure.

Connections

Where DR meets the rest of the course.

What this is actually used for

The garments were a teaching dataset. The real job: 2000 photos through **CLIP** into **512-D** vectors, then UMAP to 2-D, each point drawn as its own photo.



Nobody labelled this, and there is no legend because you do not need one. Similar *meanings* landed together — the same picture you will build for retrieval in ch17 (RAG).

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.
- And if the goal is separating *classes*: there is a supervised cousin, **Linear Discriminant Analysis (LDA)**, which picks directions by class separation rather than variance. You will meet it as *Fisherfaces* in Project 2.

Recap

- **Why:** visualize, compress, denoise — and beat the curse of dimensionality (in 100-D your farthest point is only $\sim$40% farther than your nearest).
- **PCA** — center, then take the top eigenvectors of the covariance (via SVD); the scores come out uncorrelated and ordered by variance; linear, fast, reversible. Great for preprocessing. On the garments, 95% of the variance needs **184 of 784** components — but 50 already rebuilds a recognizable shirt.
- **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).

Next (lecture 36): UMAP, opened up — the fuzzy neighbor graph, why each point gets its own distance scale, and what the cross-entropy is really doing. After that, unsupervised learning is closed and we move to a new family of models.