Machine Learning

Imperial College London

Nuri Cingillioglu

Part 6: Unsupervised Learning

The art of latent

Where are we?

In supervised learning, every training sample came with an explicit teacher target:

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$$

- **Regression:** Predict continuous target $y \in \mathbb{R}$ (e.g. server power, house price).
- **Classification:** Predict discrete class $y \in \{1, \ldots, C\}$ (e.g. benign vs malignant).

Human labels are expensive, noisy, and rare. Unlabelled data is everywhere: millions of unlabelled images, streaming sensor telemetry, genomes, and raw web text.

> What about self-supervised learning?

Enter Unsupervised Learning

In **unsupervised learning**, we are handed *only* feature vectors $\mathbf{x} \in \mathbb{R}^d$ with no labels y:

$$\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$

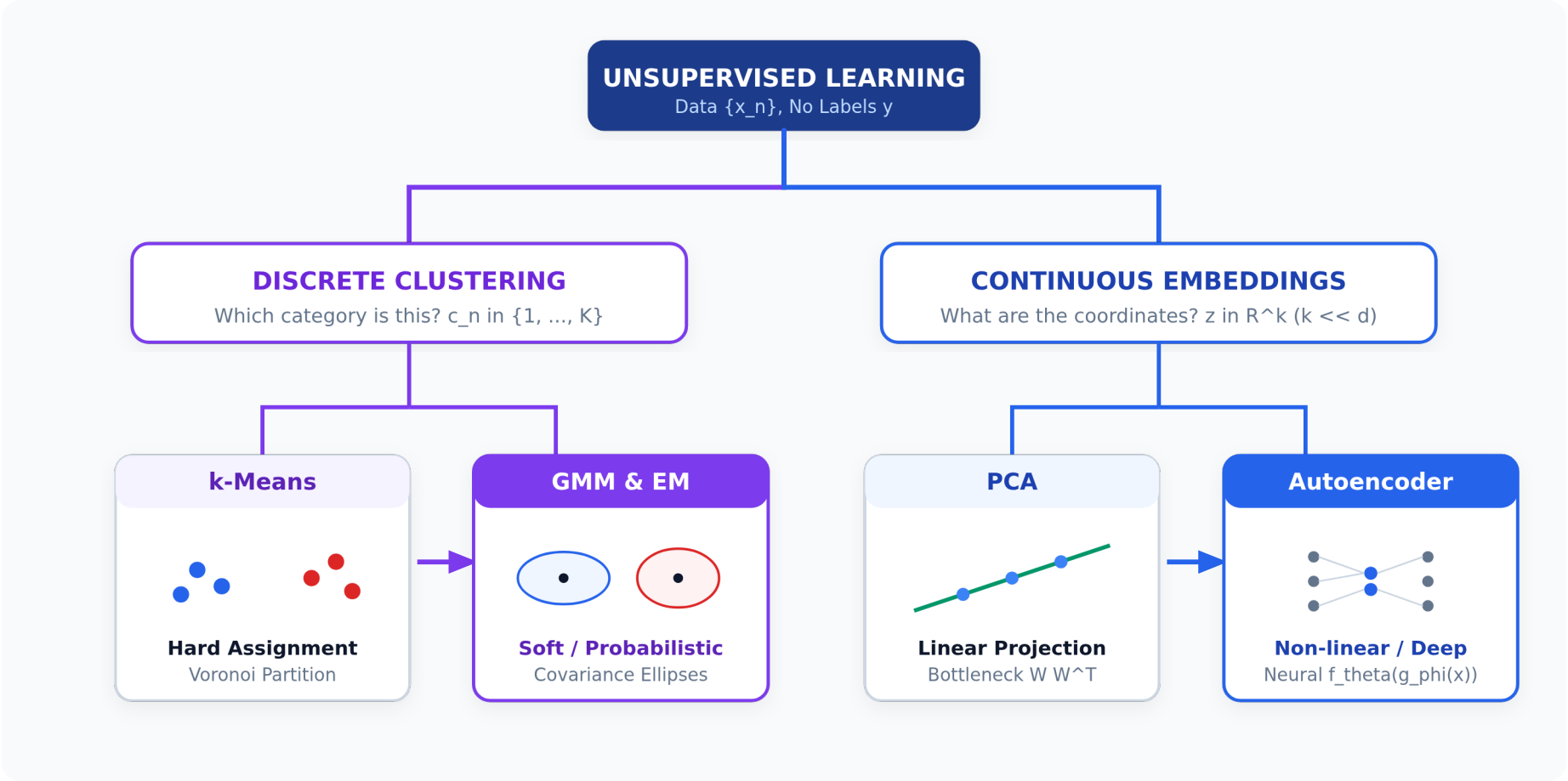
There are no ground-truth answers and no $(y - \hat{y})$ loss to minimise.

What Can We Discover?

1. **Natural Groupings:** Do the data points naturally fall into discrete families or clusters?
2. **Compact Representations:** Can we capture 95% of the information with a few primary axes?
3. **Complex Manifolds:** Can we unroll high-dimensional data onto a 2D map to visualise it?
4. **Density Estimation:** What probability distribution $p(\mathbf{x})$ generated the data?

What we'll cover

Two clean pairs of **Linear Baseline → Non-linear / Probabilistic Generalisation**:



Grouping Without Labels

Imagine you are given customer purchase records, hospital patient metrics, or galaxy spectral emissions.

You are asked: **"Are there distinct types or groups here?"**

The goal:

Partition N unlabelled observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$ into K disjoint groups $\{C_1, \ldots, C_K\}$ such that:

- Points in the *same* group are as close **(similar)** as possible.
- Points in *different* groups are as far apart as possible.

Mathematical Formulation of k-Means

Let each cluster $k \in \{1, \ldots, K\}$ be represented by a prototype centre:

$$\boldsymbol{\mu}_k \in \mathbb{R}^d \quad (\text{the cluster centroid})$$

For each data point $\mathbf{x}_n$, define a categorical cluster assignment:

$$c_n \in \{1, \ldots, K\}$$

where $c_n = k$ denotes that data point $\mathbf{x}_n$ is assigned to cluster k:

$$\mathbf{x}_n \mapsto c_n \in \{1, \ldots, K\}$$

The objective function

We want to minimise the **Within-Cluster Sum of Squares (WCSS)**, also called **Inertia** or **Distortion**:

$$J(c, \boldsymbol{\mu}) = \sum_{n=1}^{N} \|\mathbf{x}_n - \boldsymbol{\mu}_{c_n}\|_2^2$$

In my words, this means *bundle the points up*. The more bundled the better.

The Great British Bundle Off.

> Surprise! Finding the global minimum of J is **NP-hard**.

Lloyd's Coordinate Descent Algorithm

Stuart Lloyd (1957) proposed an elegant alternating algorithm:

Step 1: Assignment Step

Hold centroids $\boldsymbol{\mu}$ fixed. Assign each point to its *nearest centroid*:

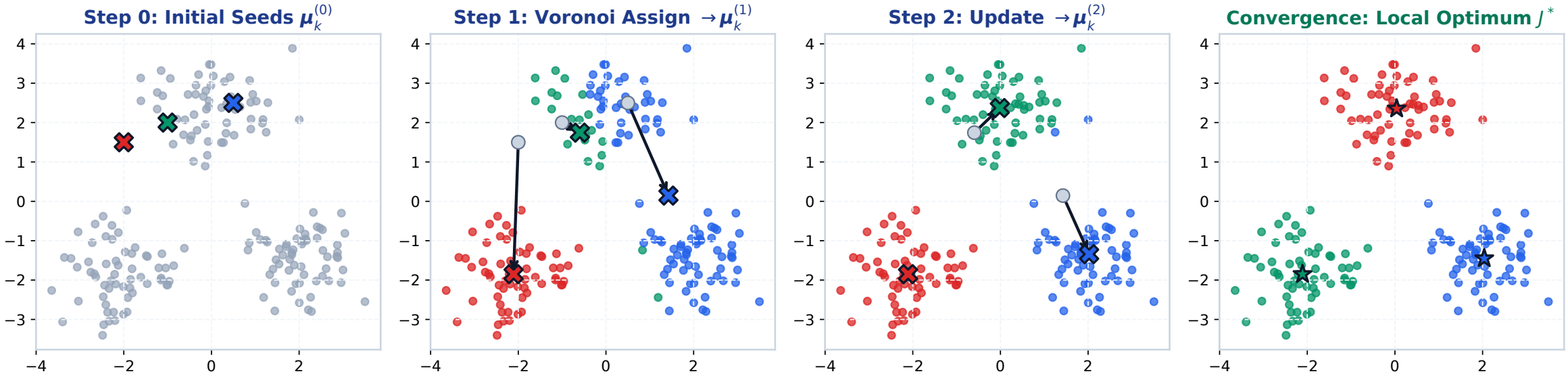
$$c_n \leftarrow \arg\min_{k \in \{1,\ldots,K\}} \left\| \mathbf{x}_n - \boldsymbol{\mu}_k \right\|_2^2$$

Step 2: Update Step

Hold assignments c fixed. Recompute each centroid as the *mean* of its assigned points:

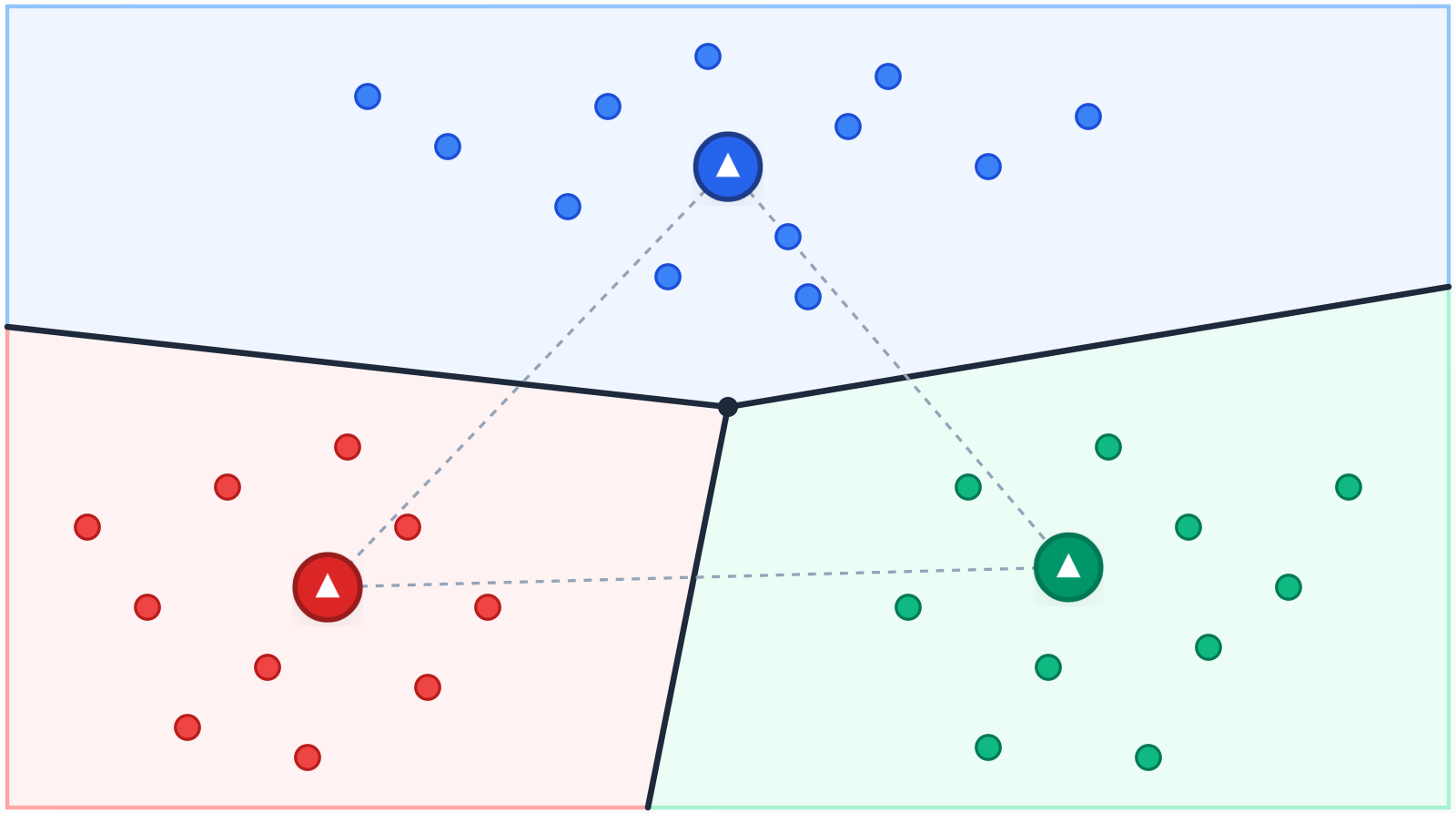
$$\boldsymbol{\mu}_k \leftarrow \frac{1}{\left| \{ n : c_n = k \} \right|} \sum_{n : c_n = k} \mathbf{x}_n$$

Iterations Progress in 2D



Notice how the centroids shift towards the centres of mass at each step until the objective stops decreasing.

We got clusters Steve



Convergence & Stopping Criteria

Monotonic Convergence: Each step is guaranteed to decrease (or leave unchanged) the objective J. Because there are only finitely many assignments, Lloyd's algorithm is guaranteed to converge in a finite number of iterations!

Practical Stopping Criteria:

1. **Centroids stop moving:** $\|\boldsymbol{\mu}_k^{(t)} - \boldsymbol{\mu}_k^{(t-1)}\|_2 < \epsilon$ for all k.
2. **Assignments stabilise:** $c_n^{(t)} = c_n^{(t-1)}$ for all n (no points switch clusters).
3. **Objective threshold:** Relative decrease threshold $\frac{J^{(t-1)} - J^{(t)}}{J^{(t-1)}} < \text{tol}$.
4. **Iteration budget:** A fixed maximum number of iterations is reached (e.g. `max_iter = 300`).

The Initialisation Trap & k-Means++

Because k-Means is iterative, it converges to a **local minimum**. Bad initial seeds can lead to terrible clusterings.

Arthur & Vassilvitskii (2007) introduced **k-Means++**:

1. Pick the first centroid $\boldsymbol{\mu}_1$ uniformly at random from the data.
2. For each point $\mathbf{x}$, compute its distance to the closest chosen centroid:

$$D(\mathbf{x}) = \min_j \|\mathbf{x} - \boldsymbol{\mu}_j\|_2$$

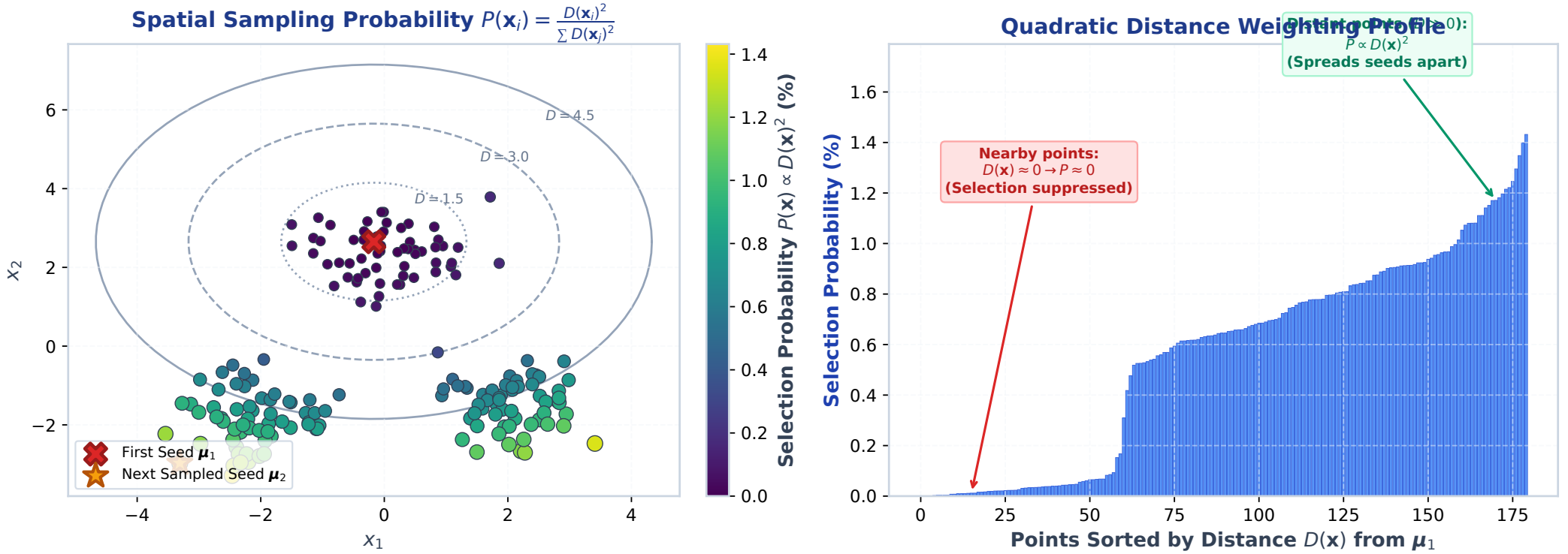
3. Pick the next centroid with probability proportional to **squared distance**:

$$P(\mathbf{x}) = \frac{D(\mathbf{x})^2}{\sum_{\mathbf{x}'} D(\mathbf{x}')^2}$$

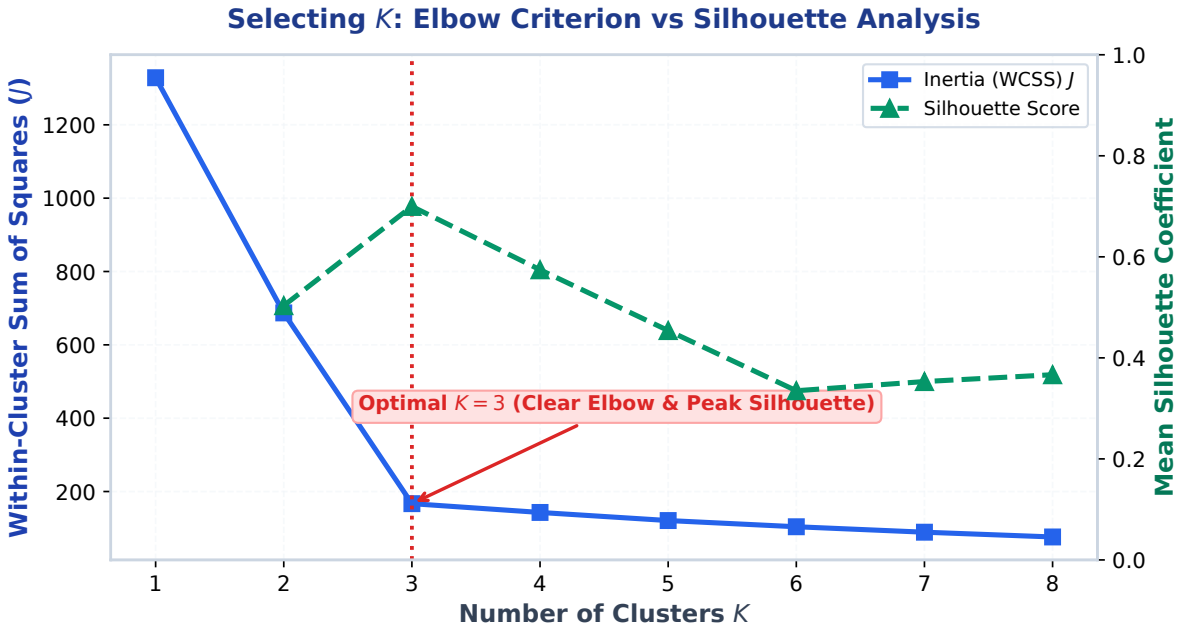
4. Repeat until K centroids are chosen, then run standard Lloyd's.

k-Means++ Sampling Geometry

Points far from existing centroids have high probability $P(\mathbf{x}) \propto D(\mathbf{x})^2$ of becoming the next seed, spreading initial centres across clusters:



Selecting K: The Elbow Method



- As K increases, J always decreases (at $K = N$, $J = 0$).
- Look for the **"Elbow"**: the point where adding another cluster gives diminishing returns.
- Supplement with the **Silhouette Score** measuring cluster cohesion vs separation.

Silhouette Analysis

For each sample $\mathbf{x}_i$ in cluster C_I:

- **Mean intra-cluster distance (cohesion):**

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, j \neq i} \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

- **Mean nearest-cluster distance (separation):**

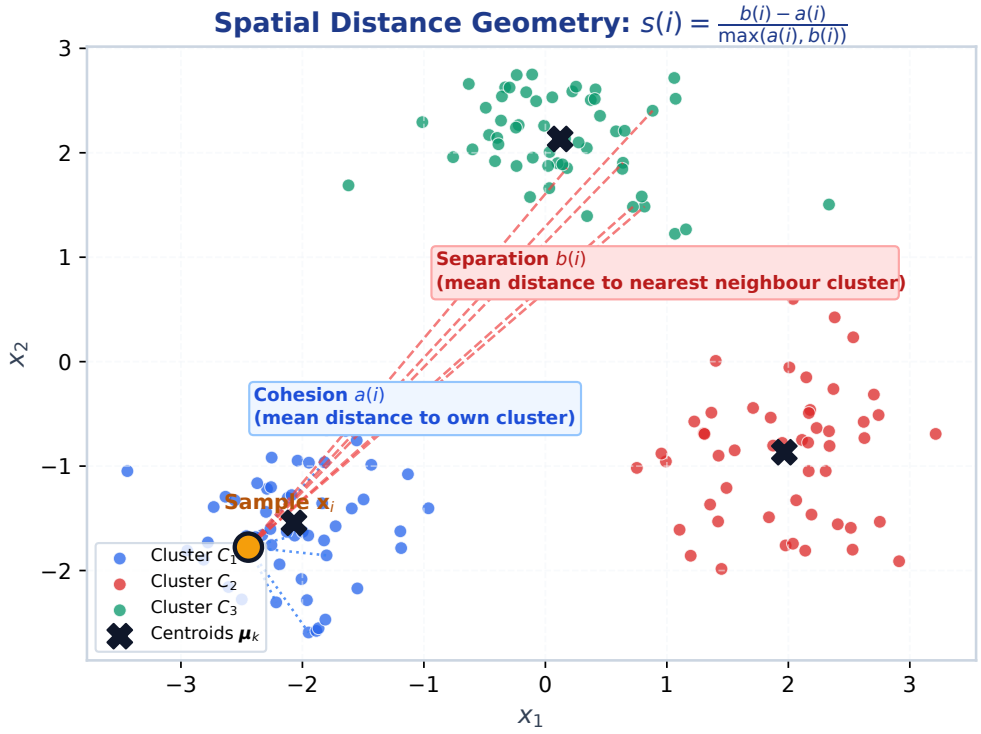
$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

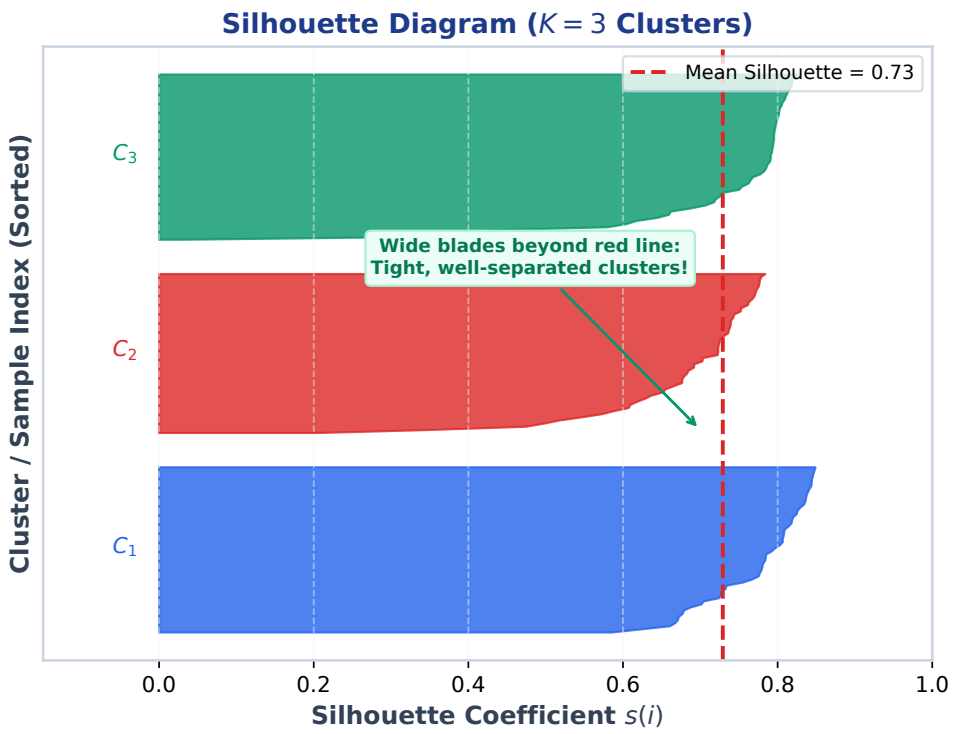
The **Silhouette coefficient** of sample i is:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, +1]$$

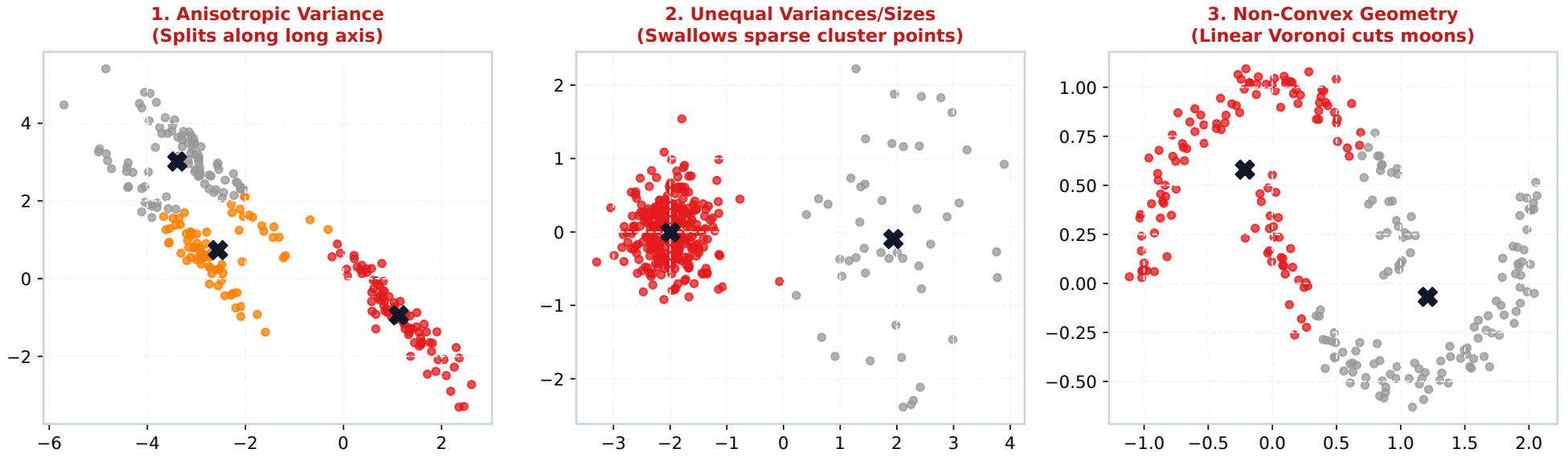
Silhouette Analysis

- $s(i) \approx +1$: Point is tightly inside its own cluster and well separated from neighbours.
- $s(i) \approx 0$: Point sits right on the boundary between two clusters.
- $s(i) < 0$: Point is closer to a neighbouring cluster than its own (probably misclustered).





Where Does k-Means Struggle?



k-Means makes three assumptions:

1. **Spherical Clusters:** Assumes isotropic variance in all directions.
2. **Equal Cluster Sizes & Densities:** Struggles when clusters have different spreads.
3. **Hard Boundaries:** Forces every point to belong 100% to a single cluster, with zero uncertainty.

Soft Clustering

A patient whose blood markers are on the borderline between "Healthy" and "Pre-diabetic".

- In k-Means, they are forced into one cluster ($c_n = 1$ or 2 with 100% certainty).
- In reality, we want a **probability** expressing confidence:

$$\gamma_{nk} = P(\text{Cluster } k \mid \mathbf{x}_n) = 0.52 \quad \text{vs} \quad 0.48$$

where γ_{nk} is the probability of assigning data point n to cluster k.

Gaussian Mixture Model (GMM)

For every data point $\mathbf{x}_n$, imagine nature generating the sample in two steps:

1. **Pick a cluster** $z_n \in \{1, \ldots, K\}$:

$$P(z_n = k) = \pi_k \qquad \left(\sum_{k=1}^{K} \pi_k = 1,\ \pi_k \geq 0 \right)$$

2. **Sample the observation** $\mathbf{x}_n \in \mathbb{R}^d$:

$$\mathbf{x}_n \mid (z_n = k) \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Marginal Density (The Resulting Data Distribution)

By the law of total probability:

$$p(\mathbf{x}_n) = \sum_{k=1}^{K} P(z_n = k)\, p(\mathbf{x}_n \mid z_n = k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

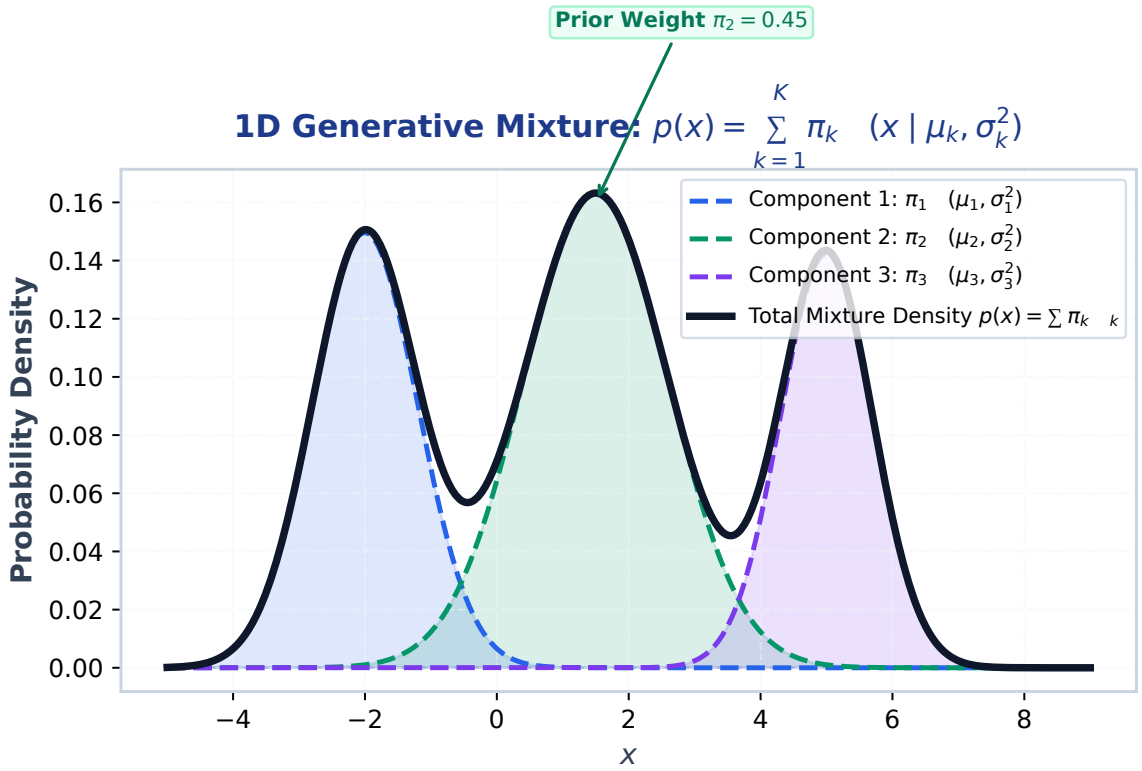
where the multivariate Gaussian density is:

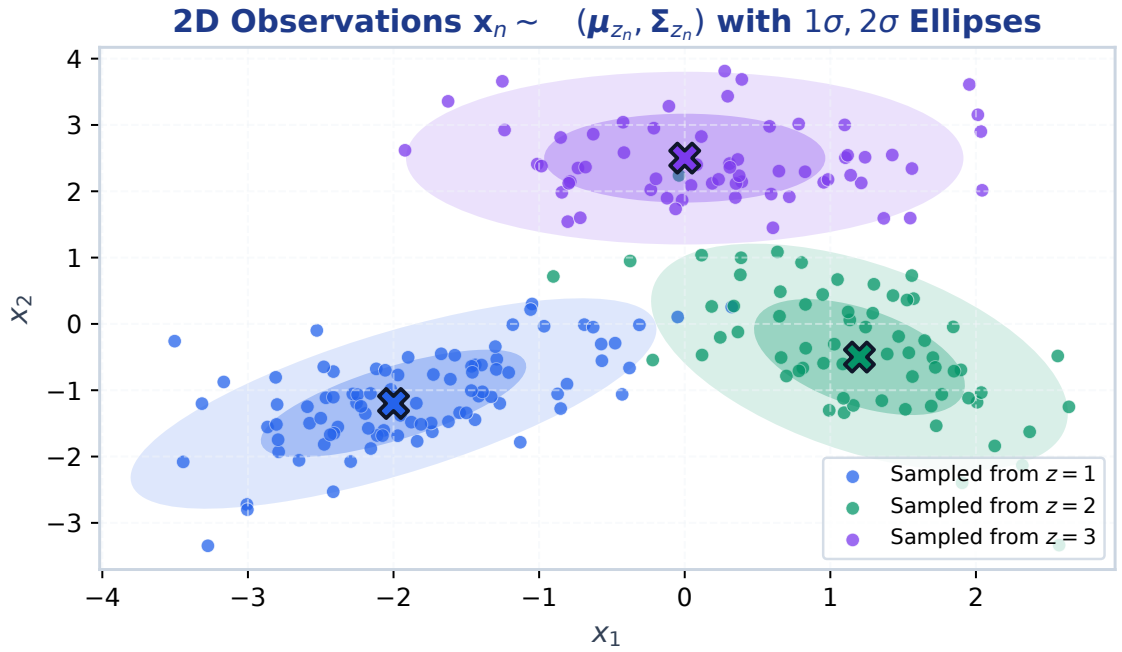
$$\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k)\right)$$

> This is a bit ugly, especially the multivariate Gaussian but don't panic.

GMM Visually

z_n is a **latent (unobserved)** variable; $\mathbf{x}_n$ is the **observed** data:





The Marginal Likelihood

For N i.i.d. observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the log-likelihood is:

$$\ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

Notice the **sum inside the logarithm**! We cannot distribute the log. Setting derivatives to zero does not yield a closed-form solution. All cluster parameters are coupled together!

> Plan B $\longrightarrow$ quit and build an AI startup without GMMs, wait for GPT-X to solve it. 😉

Expectation-Maximisation (EM)

Dempster, Laird, and Rubin (1977) proposed the **EM algorithm** to solve this:

1. E-Step (Expectation / Soft Responsibilities)

Bayes' theorem to compute the posterior probability (responsibility γ_{nk}) that cluster k produced point $\mathbf{x}_n$:

$$\gamma_{nk} = P(z_n = k \mid \mathbf{x}_n) = \frac{P(z_n = k)\, p(\mathbf{x}_n \mid z_n = k)}{\sum_{j=1}^{K} P(z_n = j)\, p(\mathbf{x}_n \mid z_n = j)} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

> Here $\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is a *number*, you plug in $\mathbf{x}_n, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ and get a number.

2. M-Step (Maximisation / Update Parameters)

Re-estimate parameters using the soft responsibilities as weights:

$$N_k = \sum_{n=1}^{N} \gamma_{nk} \quad \text{(Effective number of points assigned to cluster } k\text{)}$$

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T$$

$$\pi_k^{\text{new}} = \frac{N_k}{N}$$

GMM EM: A Concrete 1D Walkthrough

Imagine we record $x = \text{Daily Revision Hours}$ for 4 students preparing for exams:

$$\mathcal{D} = \{x_1 = 1\text{h},\ x_2 = 2\text{h},\ x_3 = 7\text{h},\ x_4 = 8\text{h}\}$$

Two natural groups exist (Casual Revision vs Intensive Cramming).
We fit $K = 2$ Gaussians with initial priors $\pi_1 = \pi_2 = 0.5$ and initial means
$\mu_1 = 2.0\text{h},\ \mu_2 = 6.0\text{h}$.

E-Step: Evaluating Responsibilities

Given Gaussian likelihood $\mathcal{N}(x \mid \mu, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2}}$:

Student x_n (Hours)	Density $\mathcal{N}(x_n \mid \mu_1)$	Density $\mathcal{N}(x_n \mid \mu_2)$	Responsibility γ_{n1}	Responsibility γ_{n2}
$x_1 = 1\text{h}$	0.2420	0.0000	**1.000**	**0.000**
$x_2 = 2\text{h}$	0.3989	0.0001	**0.999**	**0.001**
$x_3 = 7\text{h}$	0.0000	0.2420	**0.000**	**1.000**
$x_4 = 8\text{h}$	0.0000	0.0540	**0.000**	**1.000**

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n \mid \mu_k, 1)}{\sum_j \pi_j \mathcal{N}(x_n \mid \mu_j, 1)}$$

M-Step Parameter Updates

Effective Cluster Sizes

$$N_1 = 1.000 + 0.999 + 0.000 + 0.000 \approx \mathbf{2.0}, \quad N_2 \approx \mathbf{2.0}$$

Updated Priors & Means

$$\pi_1^{\text{new}} = \frac{2.0}{4} = 0.5, \quad \pi_2^{\text{new}} = 0.5$$

$$\mu_1^{\text{new}} = \frac{1.0(1) + 1.0(2)}{2.0} = \mathbf{1.5}\text{h}, \quad \mu_2^{\text{new}} = \frac{1.0(7) + 1.0(8)}{2.0} = \mathbf{7.5}\text{h}$$

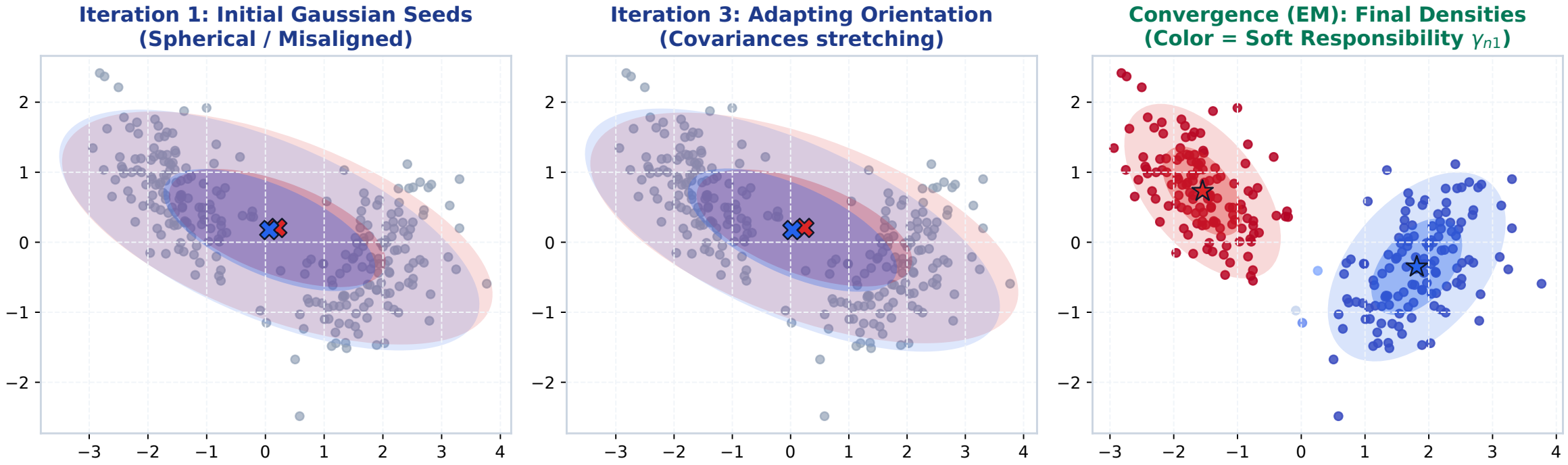
Updated Variances (General Case)

$$\sigma_1^{2,\text{new}} = \frac{1.0(1 - 1.5)^2 + 1.0(2 - 1.5)^2}{2.0} = \mathbf{0.25}, \quad \sigma_2^{2,\text{new}} = \mathbf{0.25}$$

In a single EM step, the means snap to the cluster averages $\{1.5\text{h}, 7.5\text{h}\}$ and variance shrinks to capture the tight spread!

GMM EM Progress in 2D

The elliptical contours adapt their orientation and scale to fit the data:



k-Means is GMM

What happens to GMM if we fix equal priors $\pi_k = \frac{1}{K}$ and isotropic spherical covariance $\boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I}$, and let the variance shrink to zero?

$$\gamma_{nk} = \frac{\exp\left(-\frac{\|\mathbf{x}_n - \boldsymbol{\mu}_k\|_2^2}{2\sigma^2}\right)}{\sum_{j=1}^{K} \exp\left(-\frac{\|\mathbf{x}_n - \boldsymbol{\mu}_j\|_2^2}{2\sigma^2}\right)}$$

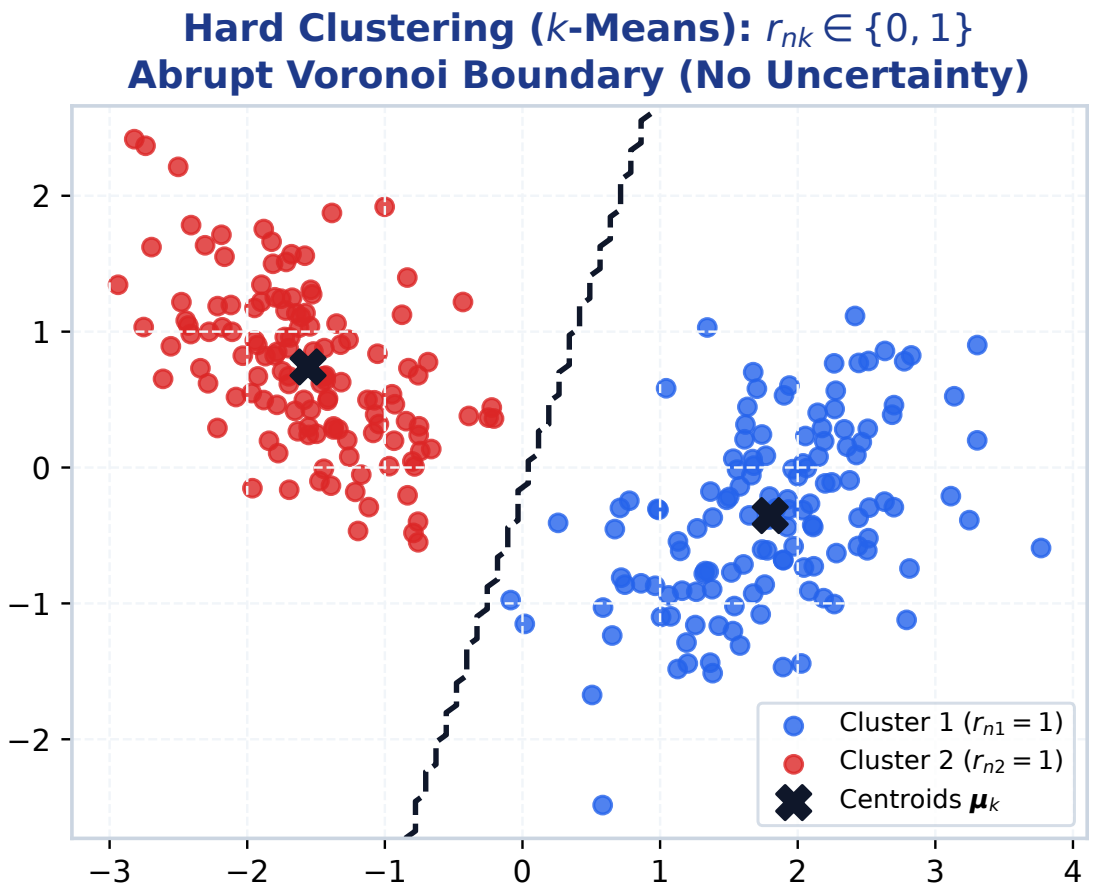
As $\sigma^2 \to 0$:

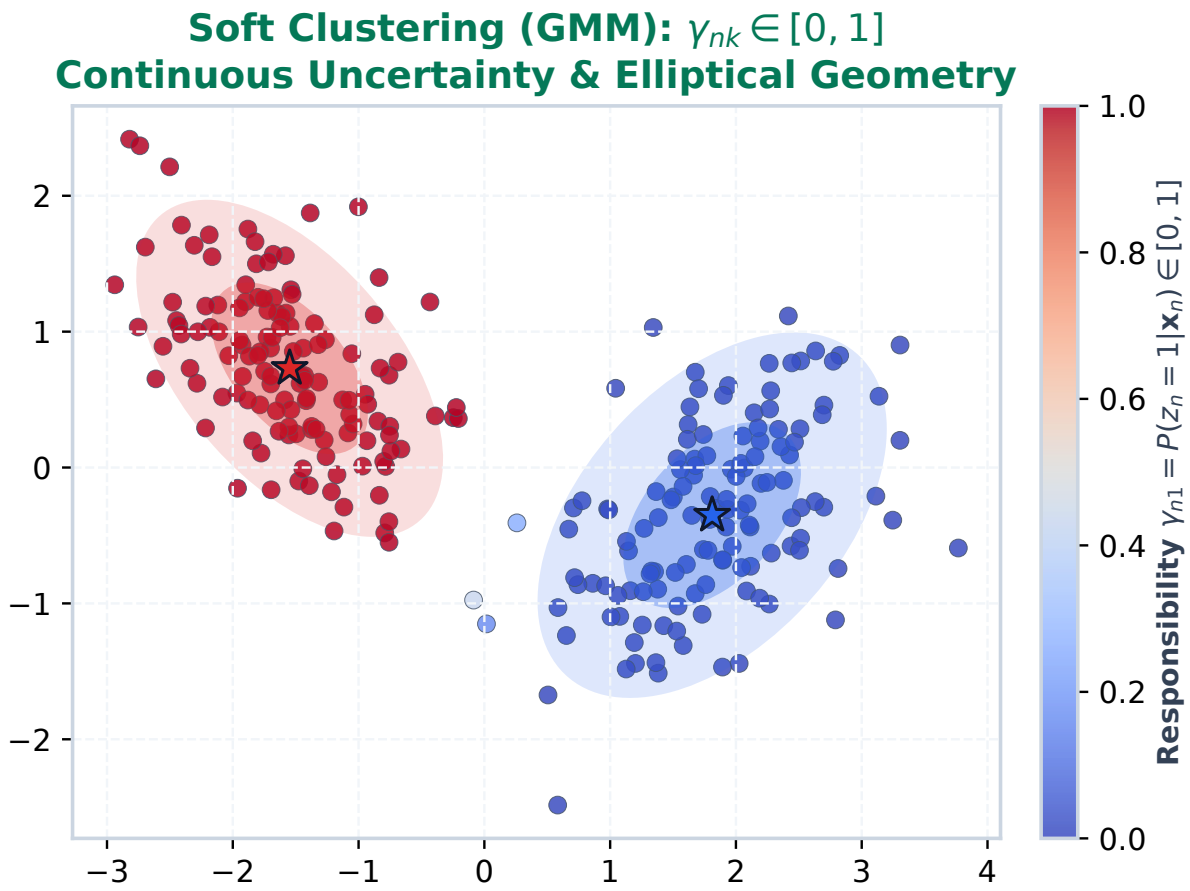
- The component closest to $\mathbf{x}_n$ dominates exponentially.
- The soft responsibility collapses to a 0 or 1 hard step function:

$$\lim_{\sigma^2 \to 0} \gamma_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|_2^2 \\ 0 & \text{otherwise} \end{cases} = \mathbb{I}(c_n = k)$$

> There is a connection here with LLM temperatures, can you spot it?

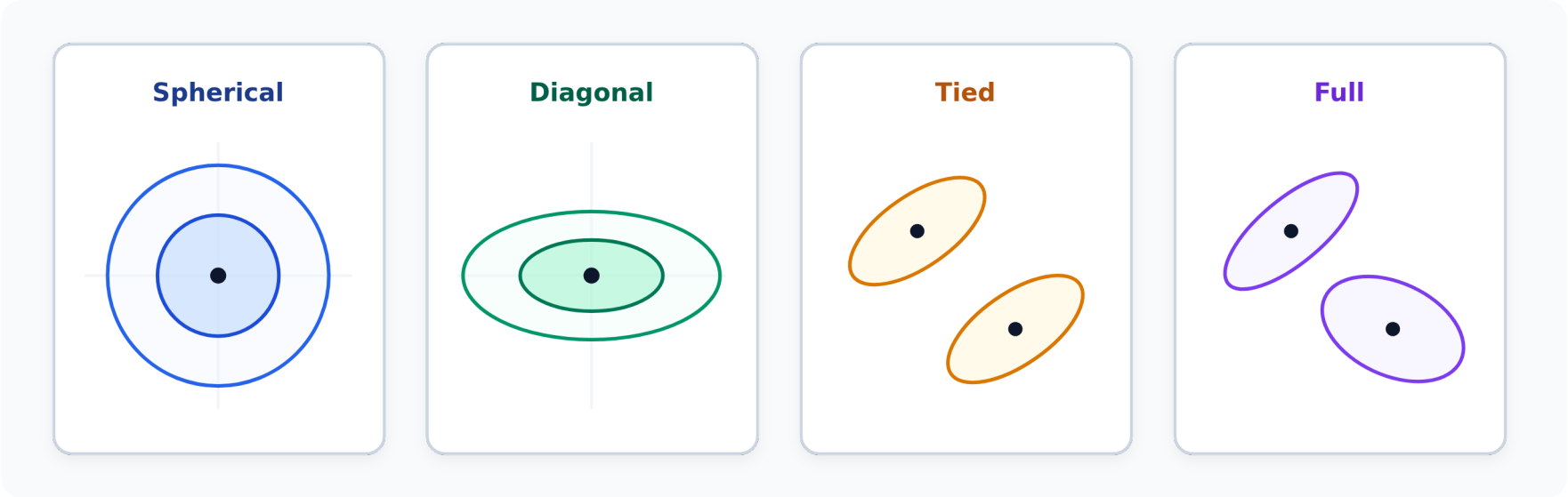
Hard vs Soft Clustering





Covariance Matrix Structures

We can control the geometric complexity of the Gaussian ellipses through $\boldsymbol{\Sigma}_k$:



- **Spherical:** Isotropic circles (1 variance parameter σ_k^2 per cluster).
- **Diagonal:** Axis-aligned ellipses (no feature correlations, d parameters per cluster).
- **Tied:** All clusters share the exact same orientation and shape ($\frac{d(d+1)}{2}$ parameters).
- **Full:** Freely oriented, unconstrained ellipses ($K \times \frac{d(d+1)}{2}$ parameters).

Model Selection: Why Likelihood Fails

In k-Means, we used inertia and the elbow heuristic to choose K. Can we choose K and the covariance structure simply by maximising training log-likelihood?

$$\ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

- Adding more Gaussian components ($K \uparrow$) or choosing more flexible covariance matrices (`'full'` vs `'spherical'`) **always increases** (or maintains) the training log-likelihood.
- In the extreme limit $K \to N$, every single point can have its own zero-variance Gaussian spike, sending likelihood to ∞.

Bayesian Information Criterion (BIC)

Introduced by **Gideon Schwarz (1978)**, BIC balances data likelihood against parameter count:

$$\mathrm{BIC} = -2 \ln \hat{L} + p \ln N$$

where:

- $\hat{L} = p(\mathbf{X} \mid \hat{\boldsymbol{\theta}})$ is the maximised likelihood of the fitted model.
- p is the total number of **free parameters** estimated by EM.
- N is the number of observed data points.

Regularisation:

- **Goodness-of-Fit Error ($-2 \ln \hat{L}$):** Decreases as the model fits the data closer.
- **Complexity Penalty ($+p \ln N$):** Increases with more clusters and parameters.

We select the model $(K, \mathrm{covariance})$ that **minimises BIC**!

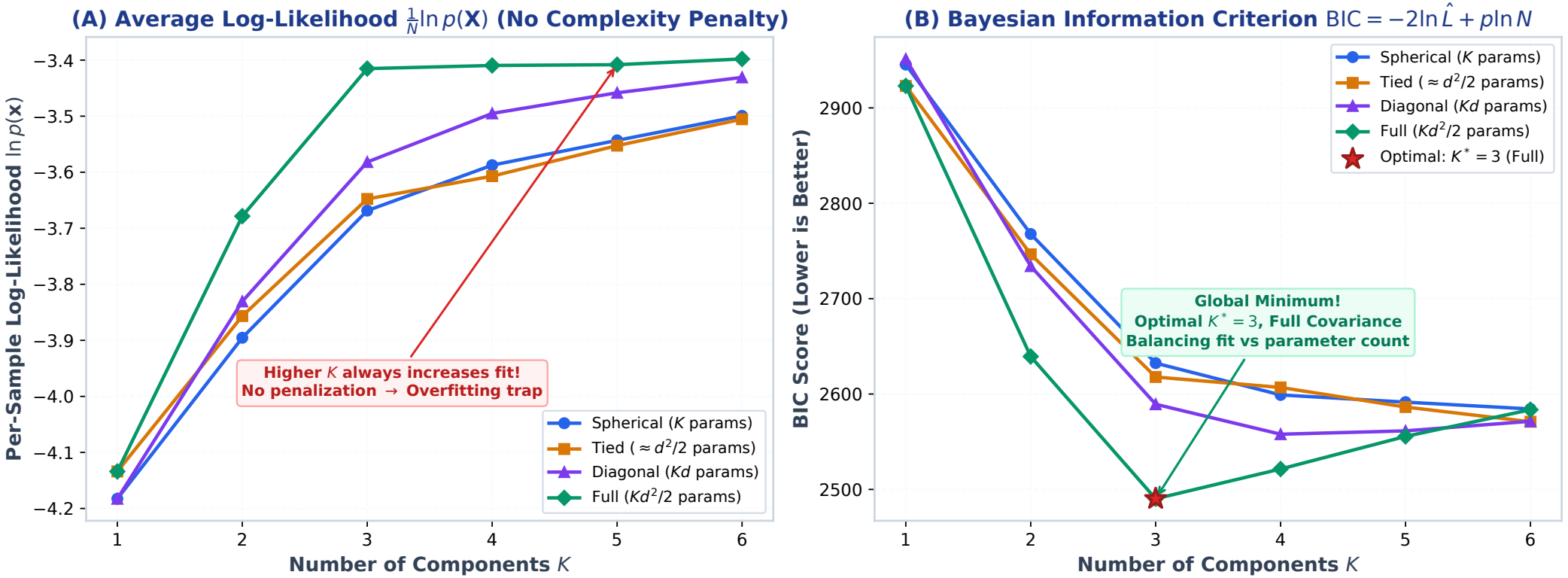
Counting Parameters p

Why does the choice of covariance type drastically alter BIC?

For a mixture of K Gaussians in d dimensions:

- **Mixing weights $\boldsymbol{\pi}$:** $K - 1$ parameters (since $\sum_{k=1}^{K} \pi_k = 1$).
- **Cluster centres $\boldsymbol{\mu}$:** $K \times d$ parameters.
- **Covariances $\boldsymbol{\Sigma}$:**
  - **Spherical:** $K \times 1$ parameters.
  - **Diagonal:** $K \times d$ parameters.
  - **Tied:** $d + \frac{d^2 - d}{2} = \frac{d(d+1)}{2}$ shared parameters.
  - **Full:** $K \times \frac{d(d+1)}{2}$ parameters.

Model Selection with BIC in Action



- **Log-Likelihood (A):** Monotonically rises as $K \to \infty$ — blind to overfitting.
- **BIC Score (B):** Penalises extra parameters, producing a clear minimum at $K^* = 3$ (`full`).

Remember likelihood

To evaluate how well a fitted GMM $\hat{\boldsymbol{\theta}} = \{\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}\}$ explains data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$:

1. **Per-Cluster Density:** For point $\mathbf{x}_n$, evaluate each Gaussian component:

$$\mathcal{N}(\mathbf{x}_n \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$$

2. **Mixture Density (Per Sample):** Weight each Gaussian by its mixing proportion $\hat{\pi}_k$:

$$p(\mathbf{x}_n \mid \hat{\boldsymbol{\theta}}) = \sum_{k=1}^{K} \hat{\pi}_k \mathcal{N}(\mathbf{x}_n \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)$$

3. **Total Log-Likelihood (All Samples):** Sum the logs assuming i.i.d. observations:

$$\ln \hat{L} = \ln p(\mathbf{X} \mid \hat{\boldsymbol{\theta}}) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n \mid \hat{\boldsymbol{\theta}}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \hat{\pi}_k \mathcal{N}(\mathbf{x}_n \mid \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k) \right)$$

Toy Walkthrough: BIC Calculation

Consider fitting GMMs with **full covariance** to $N = 100$ samples in $d = 2$ dimensions ($\ln N \approx 4.61$).
From our parameter formulas: $p = \underbrace{(K-1)}_{\boldsymbol{\pi}} + \underbrace{2K}_{\boldsymbol{\mu}} + \underbrace{3K}_{\boldsymbol{\Sigma}} = 6K - 1.$

Model	p	$\ln \hat{L}$	**Fit Error ($-2 \ln \hat{L}$)**	**Penalty ($p \ln N$)**	**BIC**	**Decision**
$K = 2$	11	-450.0	900.0	$11 \times 4.61 = 50.7$	950.7	Underfitting
$K = 3$	17	-420.0	840.0	$17 \times 4.61 = 78.4$	$\mathbf{918.4}$	**Optimal (Minimum)**
$K = 4$	23	-415.0	830.0	$23 \times 4.61 = 106.0$	936.0	Overfitting (Penalised)

Fitting a GMM in scikit-learn

```
import numpy as np
from sklearn.mixture import GaussianMixture

# 1. Instantiate GMM with chosen K and covariance structure
gmm = GaussianMixture(
    n_components=3,
    covariance_type='full',   # 'spherical', 'diag', 'tied', or 'full'
    n_init=10,                # Run 10 EM restarts to avoid local minima
    random_state=42
)

# 2. Fit parameters via EM
gmm.fit(X)

# 3. Model inspection
print("Mixing weights:", gmm.weights_)          # pi_k
print("Cluster means:\n", gmm.means_)           # mu_k
print("Log-likelihood:", gmm.score(X) * len(X)) # ln p(X)
print("Model BIC:", gmm.bic(X))                 # -2 ln L + p ln N

# 4. Inference on new points
labels = gmm.predict(X_new)           # Hard assignment: argmax_k gamma_nk
probs  = gmm.predict_proba(X_new)     # Soft responsibilities: gamma_nk in [0, 1]
```

GMM Model Selection in Python

We can automate finding the optimal $(K, \text{covariance})$ pair in a few lines of code:

```
lowest_bic = np.inf
best_gmm = None

# Grid search across candidate cluster counts and covariance structures
for cov_type in ['spherical', 'diag', 'tied', 'full']:
    for k in range(1, 8):
        candidate = GaussianMixture(n_components=k, covariance_type=cov_type, n_init=5, random_state=42)
        candidate.fit(X)
        bic = candidate.bic(X)
        if bic < lowest_bic:
            lowest_bic, best_gmm = bic, candidate

print(f"Optimal Model: K = {best_gmm.n_components}, Covariance = '{best_gmm.covariance_type}'")
print(f"Minimum BIC:   {lowest_bic:.2f}")
```

Interactive Clustering Demo

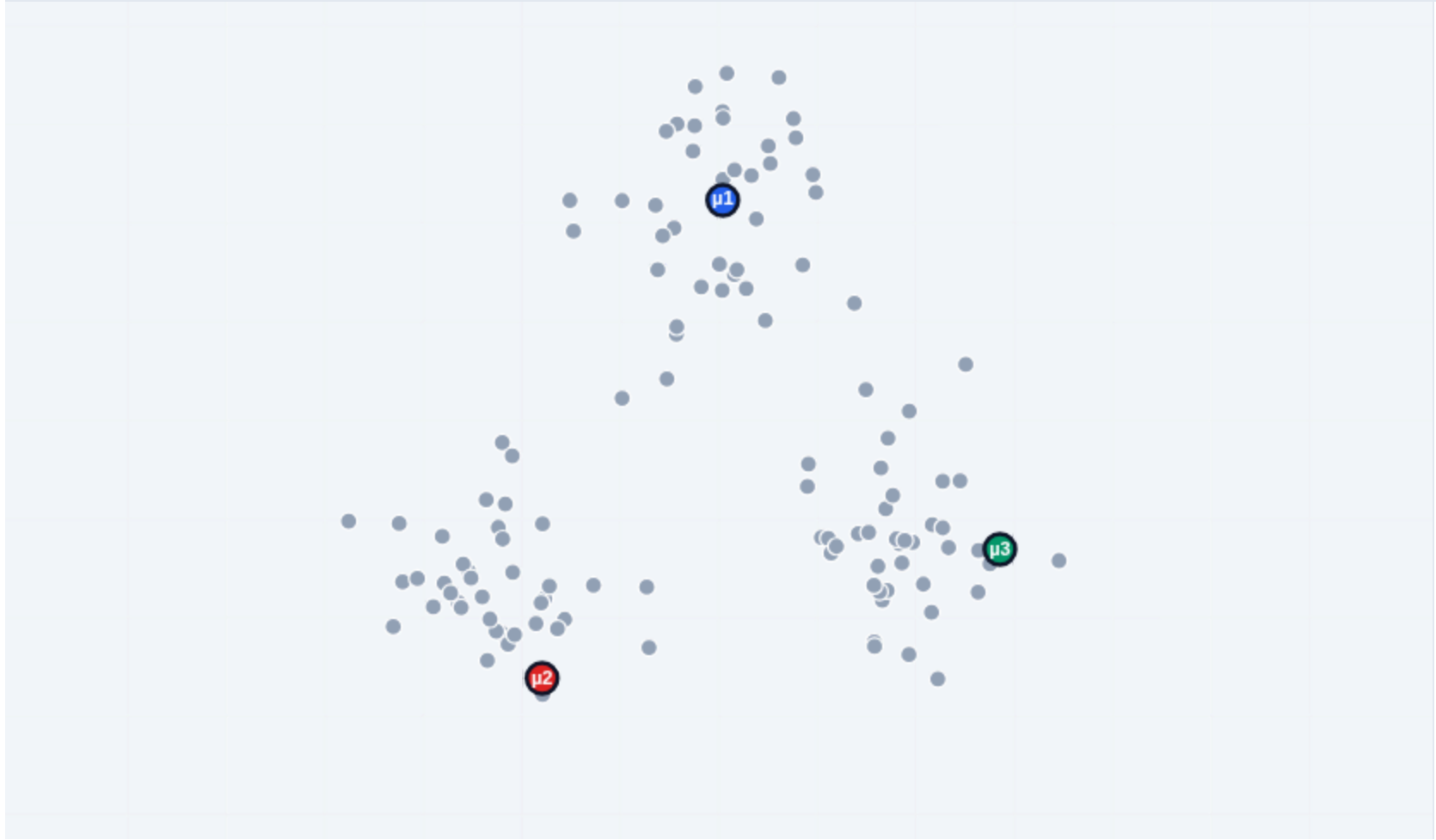
Clustering Algorithms Playground

k-Means (Lloyd's) | GMM (EM Algorithm)

Data: 3 Gaussian Blobs

K: K = 3

Step Algorithm | Auto Run | Reset / Re-seed



CURRENT STATE

Step 0: Initial Seeds

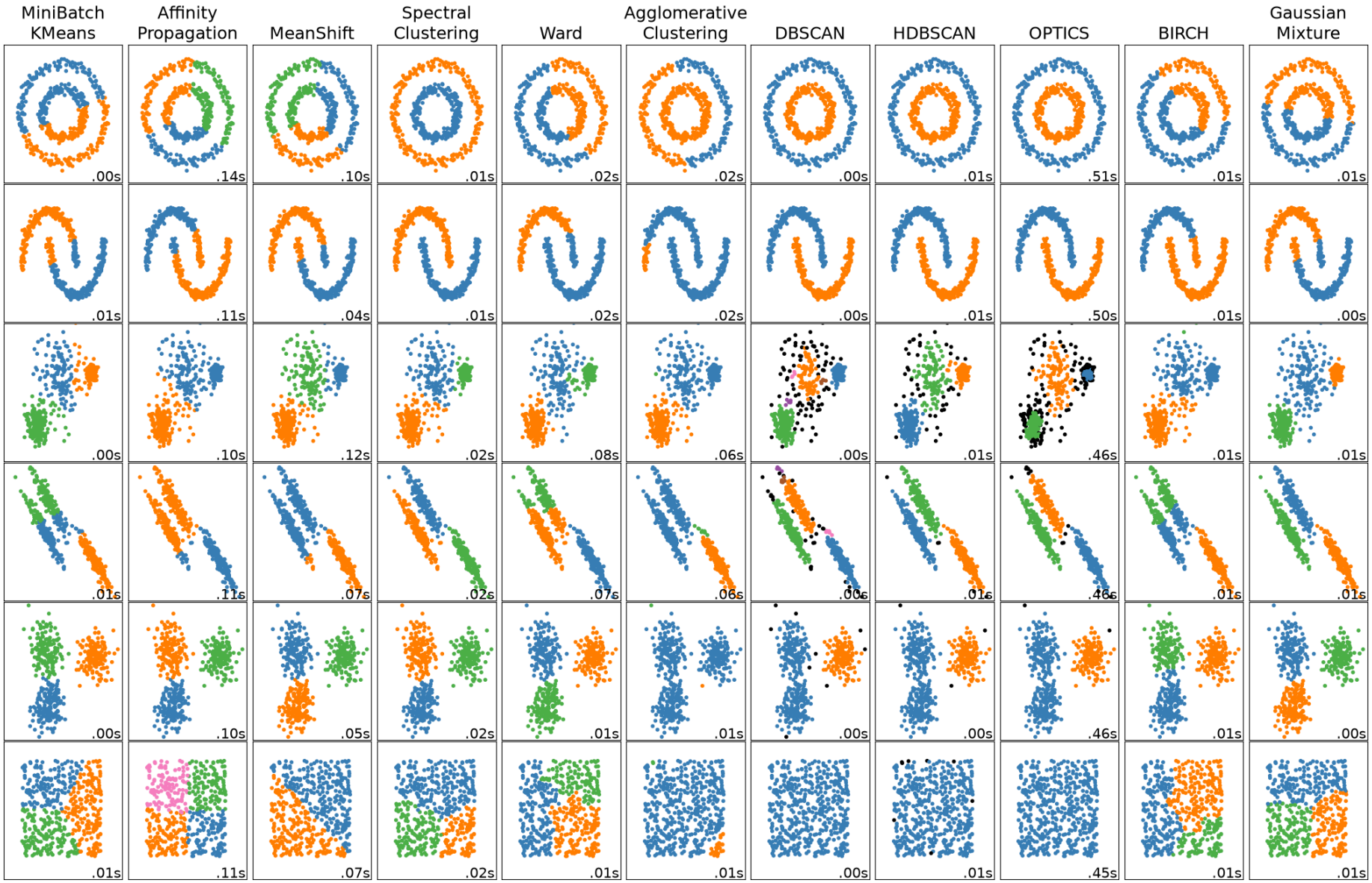
Ready to begin iteration.

CONVERGENCE METRICS

Iteration:	0
Inertia (WCSS J):	--
Converged:	No

Lloyd's k-Means Step:
1. *Assignment Step*: Point x_n assigned to nearest centroid mu_k (Voronoi cell).
2. *Update Step*: Centroid mu_k moves to mean of its assigned points.

Compare: Notice on anisotropic data how k-Means cuts across ellipses due to spherical Voronoi boundaries, whereas GMM adapts its covariance orientation!



Source: https://scikit-learn.org/stable/modules/clustering.html

Dimensionality: The Redundancy Problem

Now we turn from grouping data points to **finding compact representations**.

Suppose we record 100 features for every Imperial student: revision hours, mock exam score, problem sheet grades, library visits, tutorial attendance…

- **Correlation:** E.g. `Revision Hours` and `Mock Exam Score` vary together.
- **Useless:** Most features are utterly useless, e.g. `Study Room`

Core Question: Can we find a **compact representation** of data?

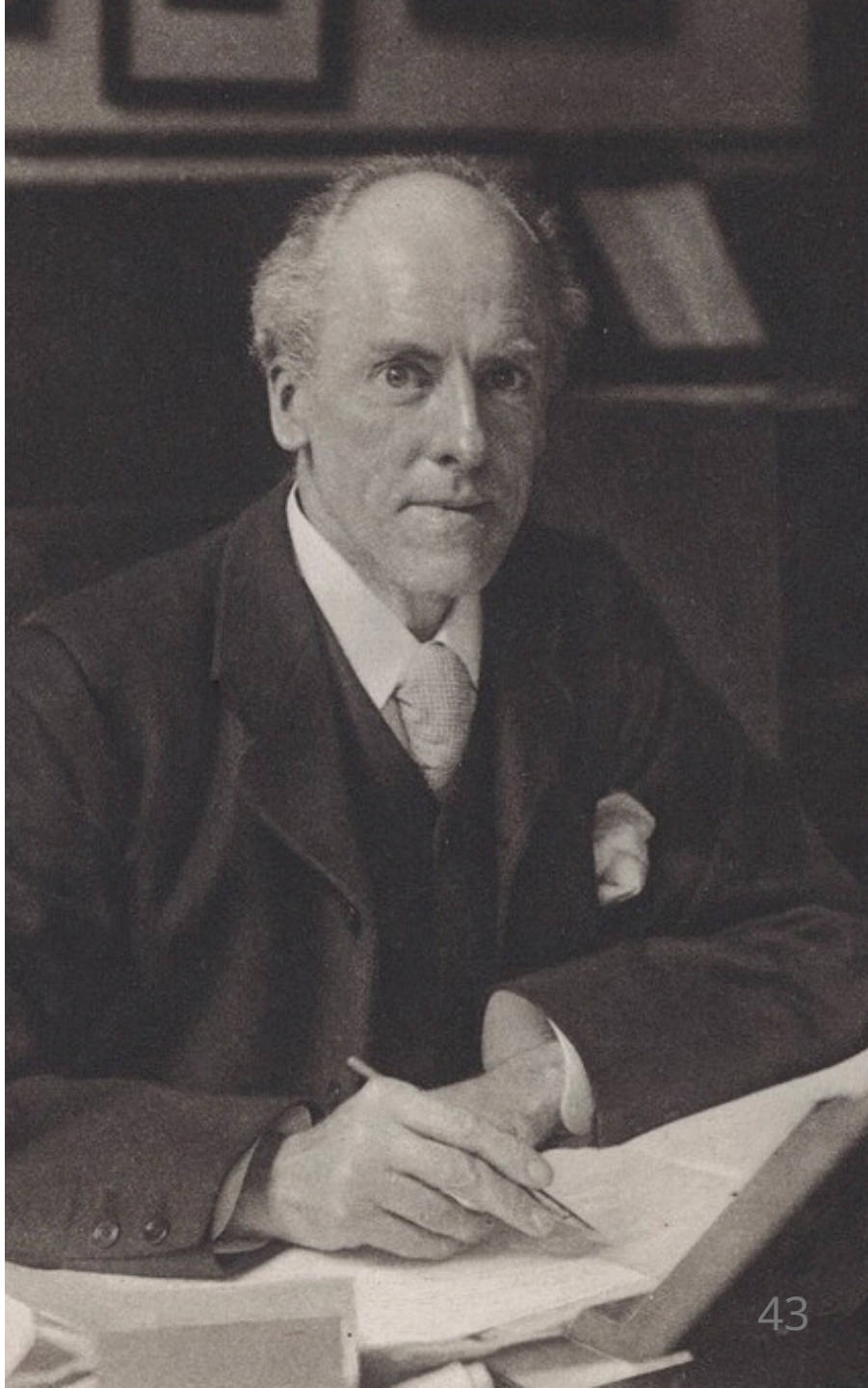
Principal Component Analysis (PCA)

Karl Pearson (1901)

"On lines and planes of closest fit to systems of points in space."

The Intuitive Idea:

- If data points are spread out widely along a direction, that direction carries **real information**.
- If points barely vary along a direction, that direction carries mostly **redundant noise**.
- Find the direction of **maximum variance**.



Step 1: Zero Mean (Centring the Data)

Given N samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$:

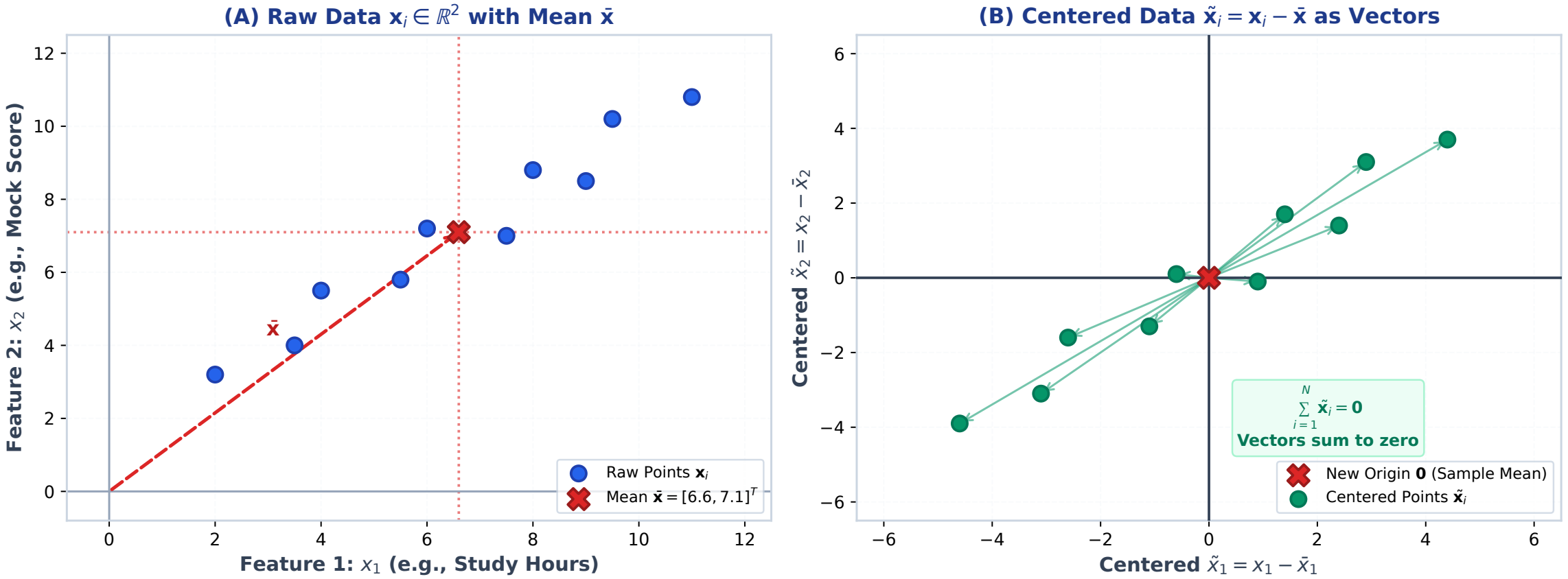
1. Compute the sample mean vector: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \in \mathbb{R}^d$
2. Centre every point by subtracting the mean:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}} \implies \sum_{i=1}^{N} \tilde{\mathbf{x}}_i = \mathbf{0}$$

3. Stack the centred observations as rows into the **design matrix** $\tilde{\mathbf{X}}$:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T \end{bmatrix} = \begin{bmatrix} \tilde{x}_{11} & \tilde{x}_{12} & \cdots & \tilde{x}_{1d} \\ \tilde{x}_{21} & \tilde{x}_{22} & \cdots & \tilde{x}_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{x}_{N1} & \tilde{x}_{N2} & \cdots & \tilde{x}_{Nd} \end{bmatrix} \in \mathbb{R}^{N \times d}$$

Centring Data & Origin Shift



- **Centred Data (B):** Coordinate origin $(0, 0)$ translated directly to the sample mean $\bar{\mathbf{x}}$.
- Every observation $\tilde{\mathbf{x}}_i$ is now a **displacement vector** emanating from the centre of mass.

Deriving the Covariance Matrix $\mathbf{S}$

Multiplying $\tilde{\mathbf{X}}^T$ by $\tilde{\mathbf{X}}$ computes all pairwise column dot products simultaneously:

$$\mathbf{S} = \frac{1}{N} \underbrace{\tilde{\mathbf{X}}^T}_{(d \times N)} \underbrace{\tilde{\mathbf{X}}}_{(N \times d)} = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\tilde{\mathbf{x}}_i}_{(d \times 1)} \underbrace{\tilde{\mathbf{x}}_i^T}_{(1 \times d)} \in \mathbb{R}^{d \times d}$$

- $\mathbf{S}$ is symmetric ($\mathbf{S}^T = \mathbf{S}$) and positive semi-definite ($\mathbf{u}^T \mathbf{S} \mathbf{u} \geq 0$).
- Diagonal entries $S_{jj} = \mathrm{Var}(x_j)$; off-diagonal entries $S_{jk} = \mathrm{Cov}(x_j, x_k)$.

> This tells us how individual features increase or decrease together.

The Maximum Variance Principle

Let $\mathbf{u}_1 \in \mathbb{R}^d$ be a unit direction vector ($\|\mathbf{u}_1\|_2 = 1$).

The scalar projection of every point $\tilde{\mathbf{x}}_i$ onto $\mathbf{u}_1$ is:

$$\mathbf{z}_1 = \tilde{\mathbf{X}}\mathbf{u}_1 \in \mathbb{R}^N \quad \text{where } z_{1_i} = \tilde{\mathbf{x}}_i^T \mathbf{u}_1 = \mathbf{u}_1 \cdot \tilde{\mathbf{x}}_i$$

The sample variance of these projected coordinates along $\mathbf{u}_1$ is:

$$\text{Var}(z_1) = \frac{1}{N}\sum_{i=1}^{N}(z_{1_i} - 0)^2 = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}_1^T\tilde{\mathbf{x}}_i)(\tilde{\mathbf{x}}_i^T\mathbf{u}_1) = \mathbf{u}_1^T\left(\frac{1}{N}\sum_{i=1}^{N}\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T\right)\mathbf{u}_1 = \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1$$

Maximising Variance

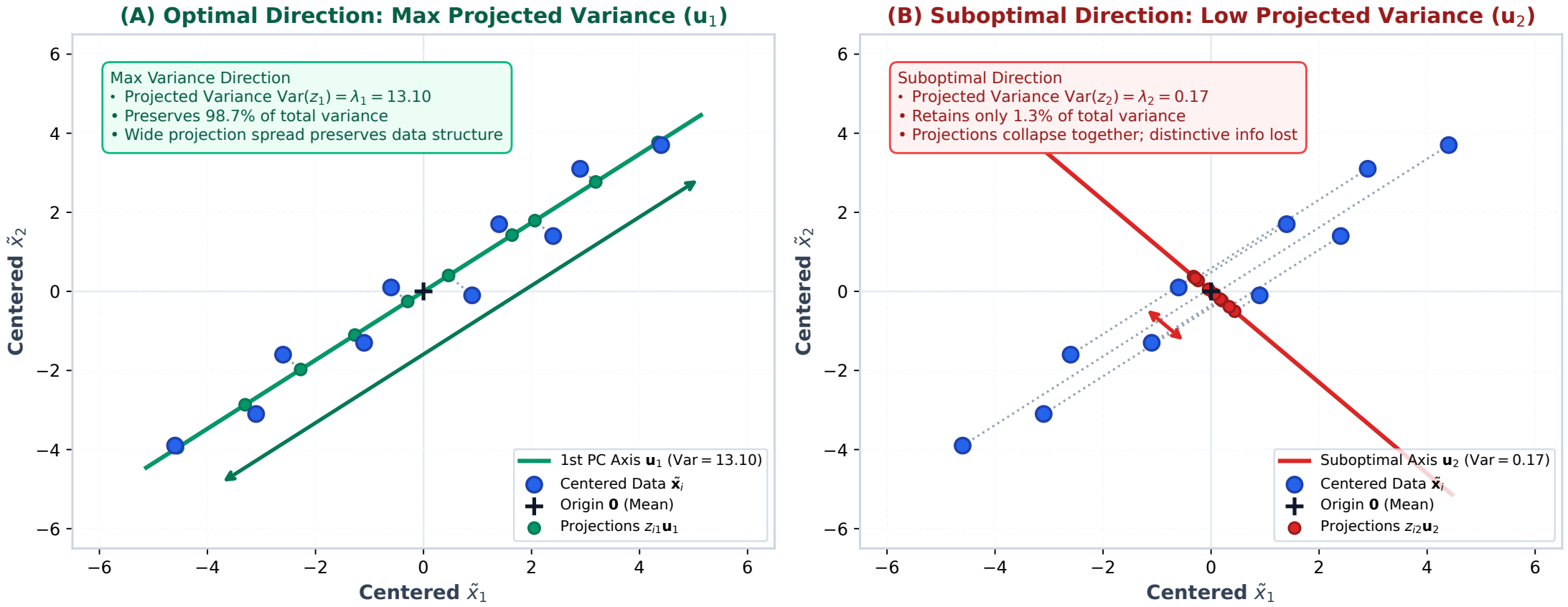
We want to find the direction $\mathbf{u}_1$ that maximises the projected variance:

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \quad \text{subject to} \quad \mathbf{u}_1^T \mathbf{u}_1 = 1$$

We constrain $\|\mathbf{u}_1\|_2 = 1$ because we only care about the **direction of the axis**, so we restrict $\mathbf{u}_1$ to lie on the **unit sphere** ($\mathbf{u}_1^T \mathbf{u}_1 = 1$).

> In other words, Steve, give me the **direction** that maximises the variance.

Visualising Projected Variance



- **Max Variance Direction ($\mathbf{u}_1$):** Projections are widely spread out, preserving maximal information about the data.

Eigenvalue Magic

$$\max_{\mathbf{u}_1} \underbrace{\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1}_{f(\mathbf{u}_1)} \quad \text{subject to} \quad \underbrace{\mathbf{u}_1^T \mathbf{u}_1}_{g(\mathbf{u}_1)} = 1$$

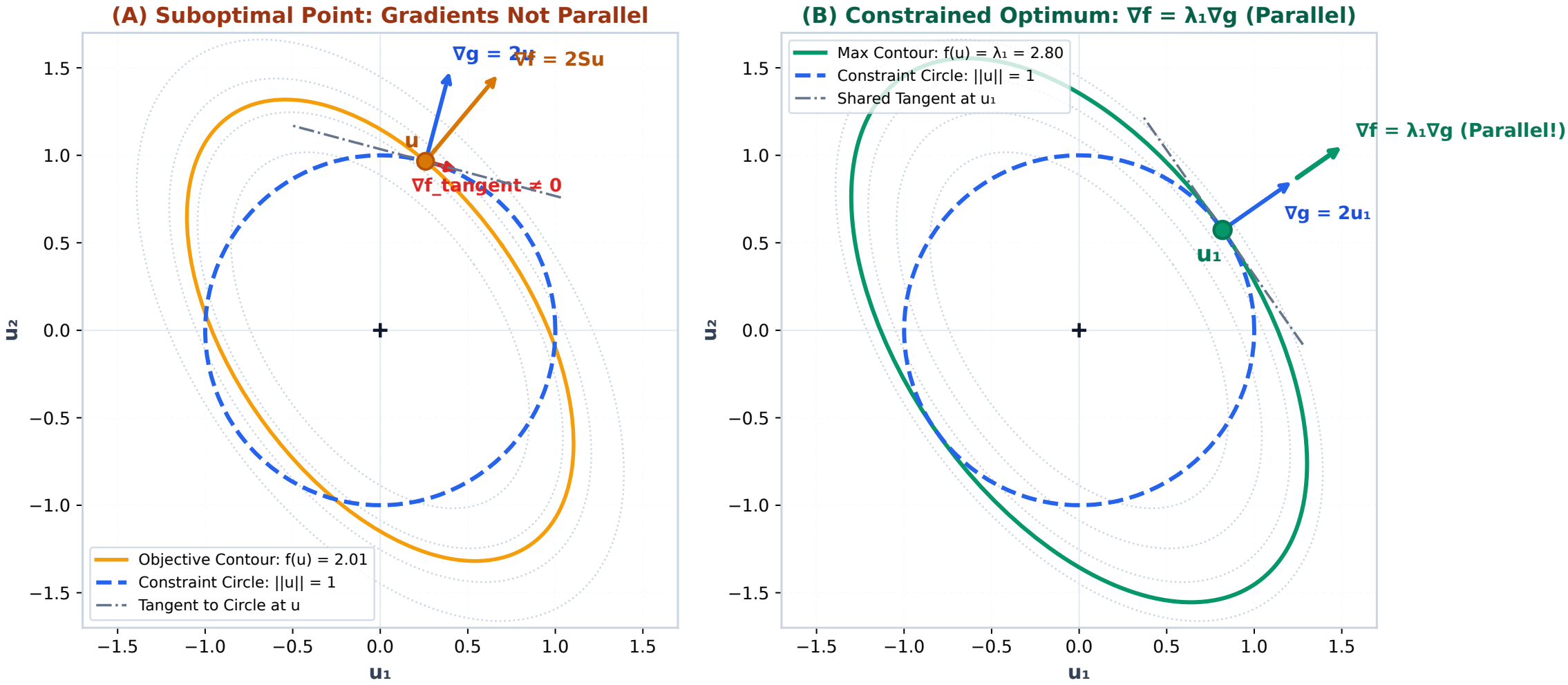
- Objective function: $f(\mathbf{u}_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \implies \nabla_{\mathbf{u}_1} f(\mathbf{u}_1) = 2\mathbf{S}\mathbf{u}_1$
- Constraint function: $g(\mathbf{u}_1) = \mathbf{u}_1^T \mathbf{u}_1 \implies \nabla_{\mathbf{u}_1} g(\mathbf{u}_1) = 2\mathbf{u}_1$

At the optimum, any move along the sphere must not increase the objective, so the gradient can only point straight out from the sphere, parallel to $\mathbf{u}_1$.

$$\begin{aligned} \nabla_{\mathbf{u}_1} f(\mathbf{u}_1) &= \lambda_1 \nabla_{\mathbf{u}_1} g(\mathbf{u}_1) \\ 2\mathbf{S}\mathbf{u}_1 &= \lambda_1 2\mathbf{u}_1 \\ \mathbf{S}\mathbf{u}_1 &= \lambda_1 \mathbf{u}_1 \end{aligned}$$

> This is constrained optimisation, we don't really care so don't worry about the math too much. It is called Lagrange multipliers and is out of scope.

Geometric Intuition: Gradient Alignment



Eigenvalue Decomposition

$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

This is the standard **eigenvalue equation**! The projected variance is simply:

$$\text{Projected Variance} = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \mathbf{u}_1^T (\lambda_1 \mathbf{u}_1) = \lambda_1 (\mathbf{u}_1^T \mathbf{u}_1) = \lambda_1$$

> $\mathbf{u}_1$ must be an **eigenvector** of $\mathbf{S}$, and the eigenvalue λ_1 **is the projected variance**! To maximise variance, we simply choose the eigenvector with the **largest eigenvalue** $\lambda_{\max}$.

Subsequent Principal Components

What about the second direction $\mathbf{u}_2$?

We maximise variance $\mathbf{u}_2^T \mathbf{S} \mathbf{u}_2$ subject to:

1. Unit length: $\mathbf{u}_2^T \mathbf{u}_2 = 1$
2. **Orthogonal to $\mathbf{u}_1$:** $\mathbf{u}_1^T \mathbf{u}_2 = 0$

- $\mathbf{u}_2$ is the eigenvector with the **second largest eigenvalue** λ_2.
- In general, the j-th principal component is the j-th eigenvector of $\mathbf{S}$, capturing variance λ_j .

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$$

$$\text{Total Variance} = \sum_{j=1}^{d} \lambda_j = \text{Tr}(\mathbf{S})$$

PCA by Hand: A 2D Toy Walkthrough

Consider 4 students with 2 features ($x_1 = \text{Study Hours}, x_2 = \text{Mock Score}$):

Student	Study Hours (x_1)	Mock Exam Score (x_2)
1	2	1
2	4	5
3	6	5
4	8	9

Step 1: Means

$$\bar{x}_1 = \frac{2+4+6+8}{4} = 5, \quad \bar{x}_2 = \frac{1+5+5+9}{4} = 5 \implies \bar{\mathbf{x}} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

Toy Walkthrough: Covariance Matrix

Step 2: Centred Data $\tilde{\mathbf{X}}$

$$\tilde{\mathbf{X}} = \begin{bmatrix} 2-5 & 1-5 \\ 4-5 & 5-5 \\ 6-5 & 5-5 \\ 8-5 & 9-5 \end{bmatrix} = \begin{bmatrix} -3 & -4 \\ -1 & 0 \\ 1 & 0 \\ 3 & 4 \end{bmatrix}$$

Step 3: Covariance Matrix $\mathbf{S} = \frac{1}{4}\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}$

$$\mathbf{S} = \frac{1}{4}\begin{bmatrix} -3 & -1 & 1 & 3 \\ -4 & 0 & 0 & 4 \end{bmatrix}\begin{bmatrix} -3 & -4 \\ -1 & 0 \\ 1 & 0 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 6 \\ 6 & 8 \end{bmatrix}$$

- $\mathrm{Var}(x_1) = 5, \quad \mathrm{Var}(x_2) = 8, \quad \mathrm{Cov}(x_1, x_2) = 6$

Toy Walkthrough: Eigenvalues

Step 4: Characteristic Equation $\det(\mathbf{S} - \lambda \mathbf{I}) = 0$

$$\det \begin{bmatrix} 5-\lambda & 6 \\ 6 & 8-\lambda \end{bmatrix} = (5-\lambda)(8-\lambda) - 36 = \lambda^2 - 13\lambda + 4 = 0$$

Using the quadratic formula:

$$\lambda = \frac{13 \pm \sqrt{169 - 16}}{2} = \frac{13 \pm \sqrt{153}}{2} \approx \frac{13 \pm 12.369}{2}$$

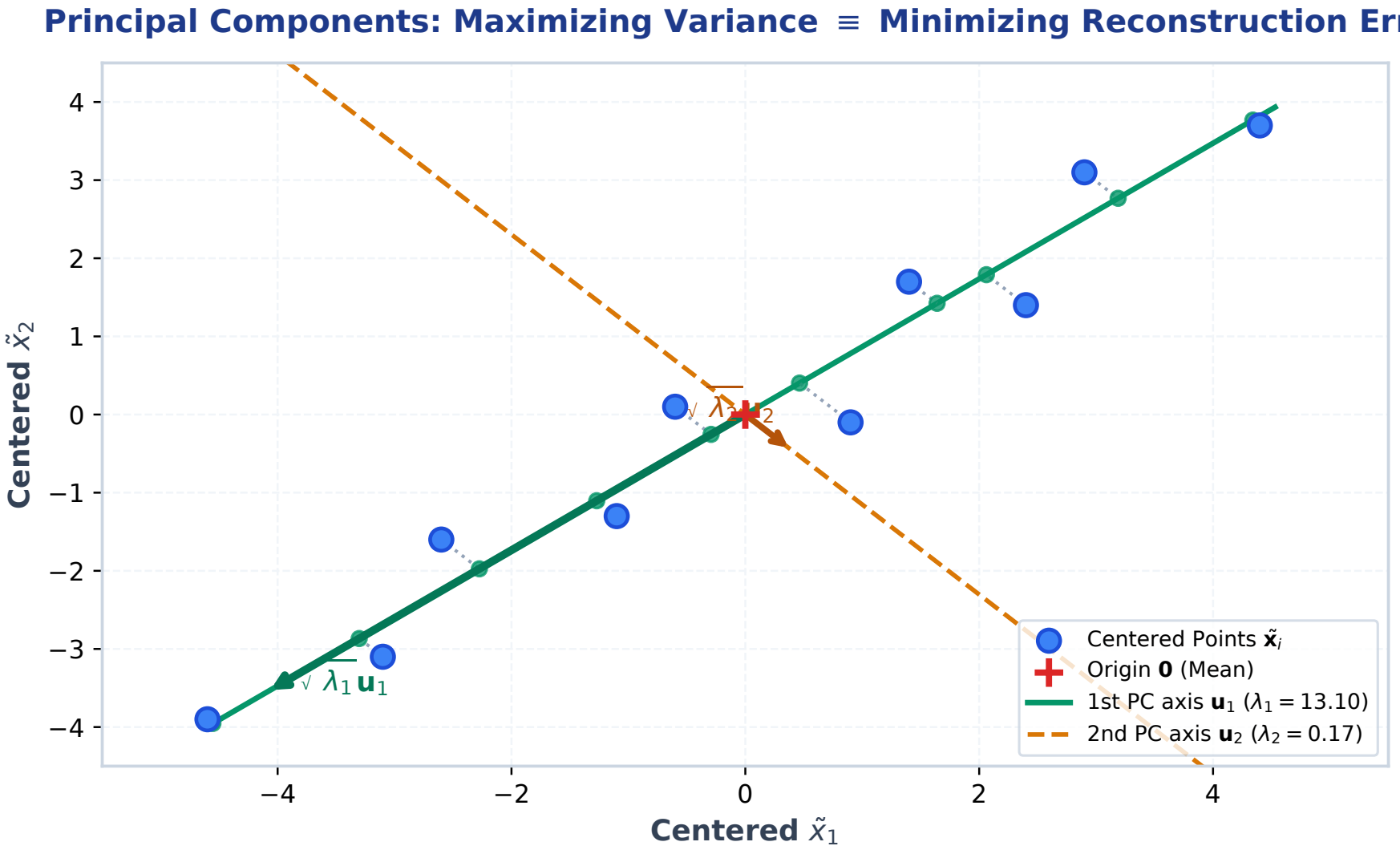
$$\lambda_1 \approx 12.685, \quad \lambda_2 \approx 0.315$$

Notice: $\lambda_1 + \lambda_2 = 13.000 = \text{Total Variance}$.

The first principal component alone captures:

$$\frac{12.685}{5+8} = \mathbf{97.6\%} \text{ of the total variance!}$$

Principal Axes & Orthogonal Projections



Singular Value Decomposition (SVD)

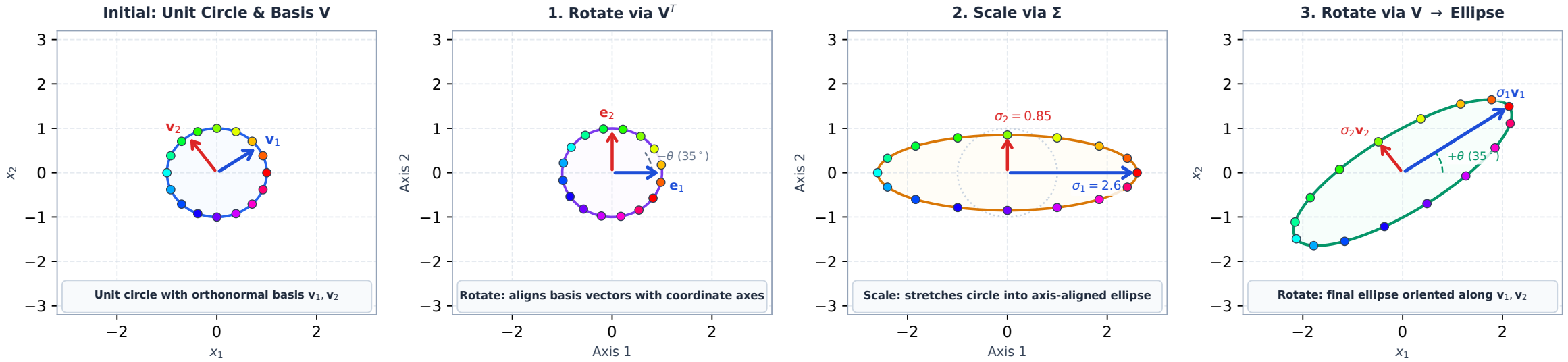
In practice, PCA is computed via the **(SVD)** of centred data $\tilde{\mathbf{X}} \in \mathbb{R}^{N \times d}$:

$$\tilde{\mathbf{X}} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$

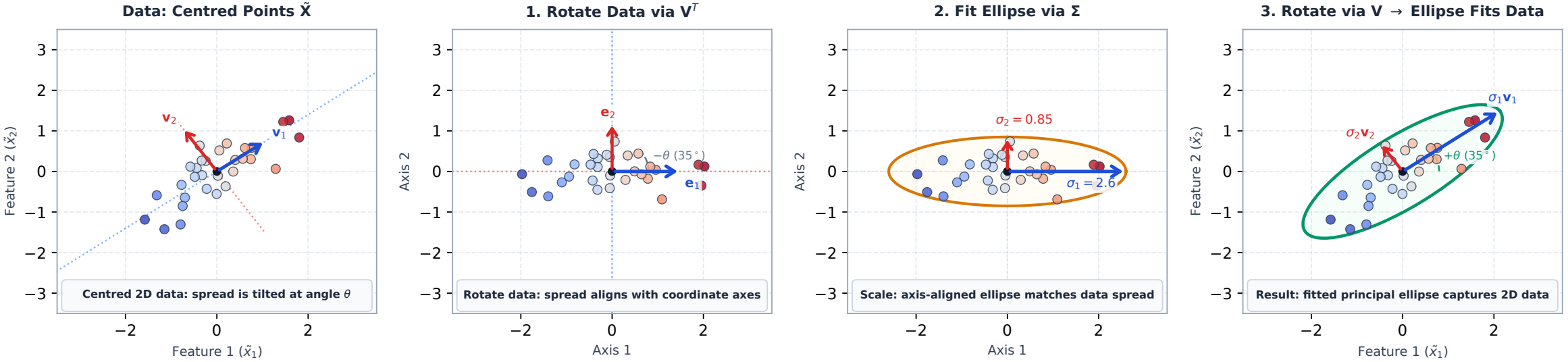
where $\mathbf{U} \in \mathbb{R}^{N \times N}$ and $\mathbf{V} \in \mathbb{R}^{d \times d}$ are orthogonal ($\mathbf{U}^T\mathbf{U} = \mathbf{I}_N, \mathbf{V}^T\mathbf{V} = \mathbf{I}_d$), and $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times d}$ contains singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$.

1. **Numerical Stability:**
   Forming $\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}$ is problematic. SVD operates directly on $\tilde{\mathbf{X}}$.
2. **Efficiency When $N \ll d$:**
   If we have $N = 100$ samples and $d = 20,000$ features (e.g. genomics), $\mathbf{S}$ is a massive $20,000 \times 20,000$ matrix ($\approx 3.2\ \text{GB}$). Truncated SVD computes the top k components without ever forming the $d \times d$ matrix.

SVD Geometry (Mapping Unit Circle to Ellipse)



PCA Application (Ellipse to 2D Data Cloud)



Why Does SVD Solve PCA?

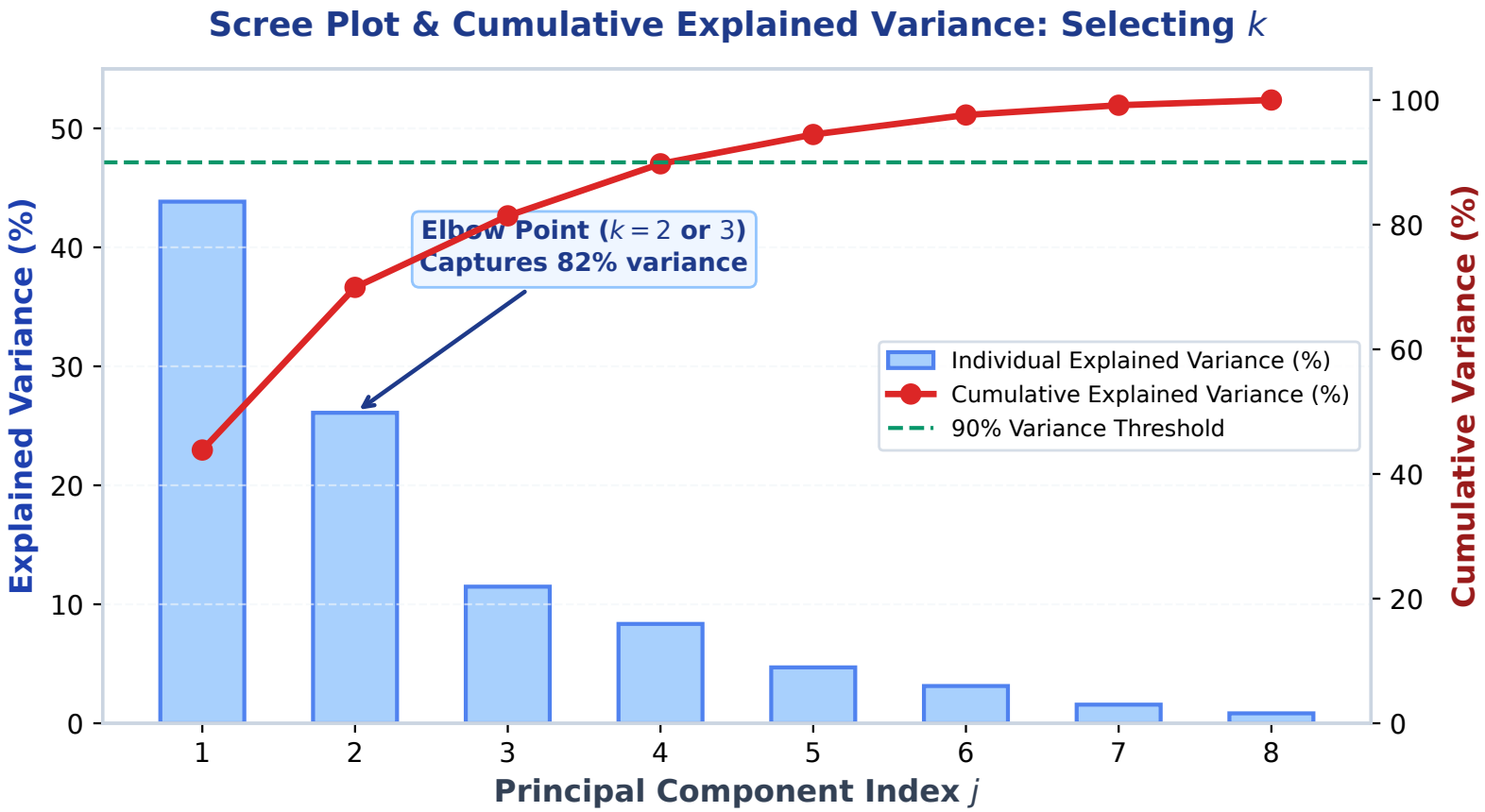
Substitute the SVD of $\tilde{\mathbf{X}}$ into the definition of the covariance matrix $\mathbf{S}$:

$$\mathbf{S} = \frac{1}{N}\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} = \frac{1}{N}(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \frac{1}{N}\mathbf{V}\mathbf{\Sigma}^T(\underbrace{\mathbf{U}^T\mathbf{U}}_{\mathbf{I}_N})\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\left(\frac{\mathbf{\Sigma}^2}{N}\right)\mathbf{V}^T$$

This becomes the **exact spectral eigendecomposition** $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$.

- The right vectors $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$ are the **eigenvectors** of $\mathbf{S}$.
- The covariance eigenvalues are directly: $\lambda_j = \frac{\sigma_j^2}{N}$.

Selecting k



- **Elbow Rule:** Retain components up to the "elbow" where eigenvalue drop-off flattens.
- **Threshold Rule:** Retain components to explain 90% or 95% of cumulative variance.

Interactive PCA Demo

Rotate the projection axis to see the projected variance bar peak at the first eigenvector:



MNIST PCA

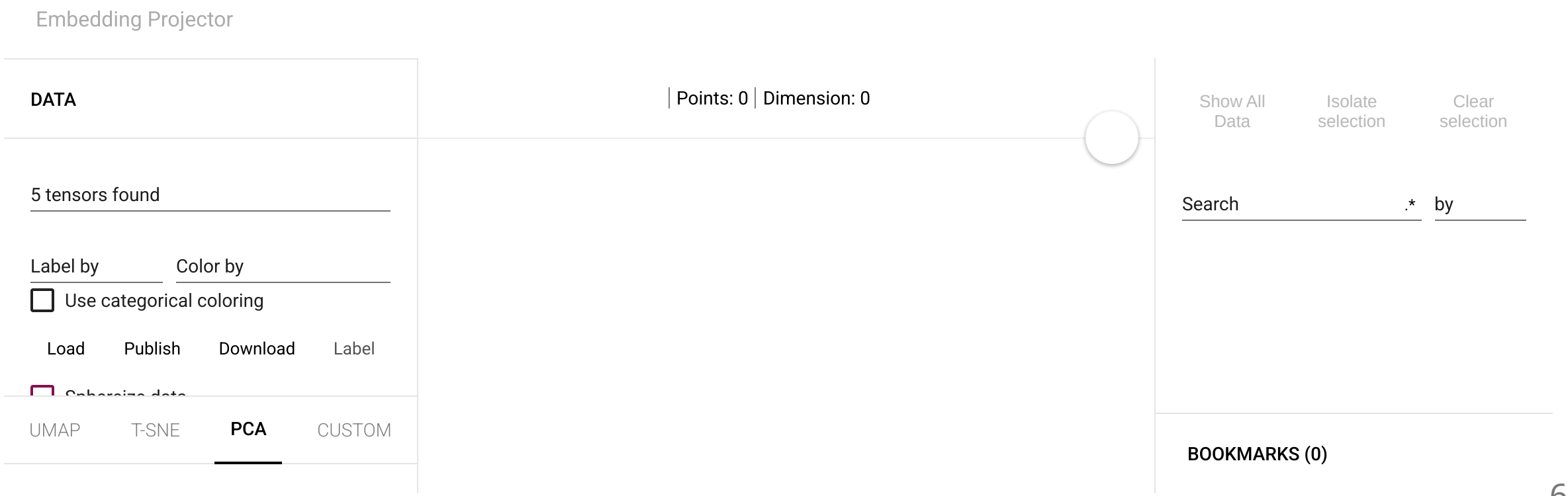
```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA

# 1. Fetch MNIST (70,000 samples, 784 features)
X, y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False)

# 2. Fit PCA and project down to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X / 255.0)  # scale pixel values to [0, 1]

# 3. Plot the projection
plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    X_pca[:, 0], X_pca[:, 1], c=y.astype(int), cmap='tab10', s=1, alpha=0.5
)
plt.colorbar(scatter, label='Digit Class')
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%} variance)')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%} variance)')
plt.title('MNIST Projected onto First 2 Principal Components')
plt.tight_layout()
plt.show()
```

MNIST PCA


Embedding Projector
DATA
Points: 0 | Dimension: 0
Show All Data
Isolate selection
Clear selection
5 tensors found
Search
.*
by
Label by
Color by
Use categorical coloring
Load
Publish
Download
Label
UMAP
T-SNE
PCA
CUSTOM
BOOKMARKS (0)

The Dual Perspective: Reconstruction Error

Pearson's original 1901 formulation pursued an identical goal from a different angle: finding the subspace of **minimum reconstruction error**!

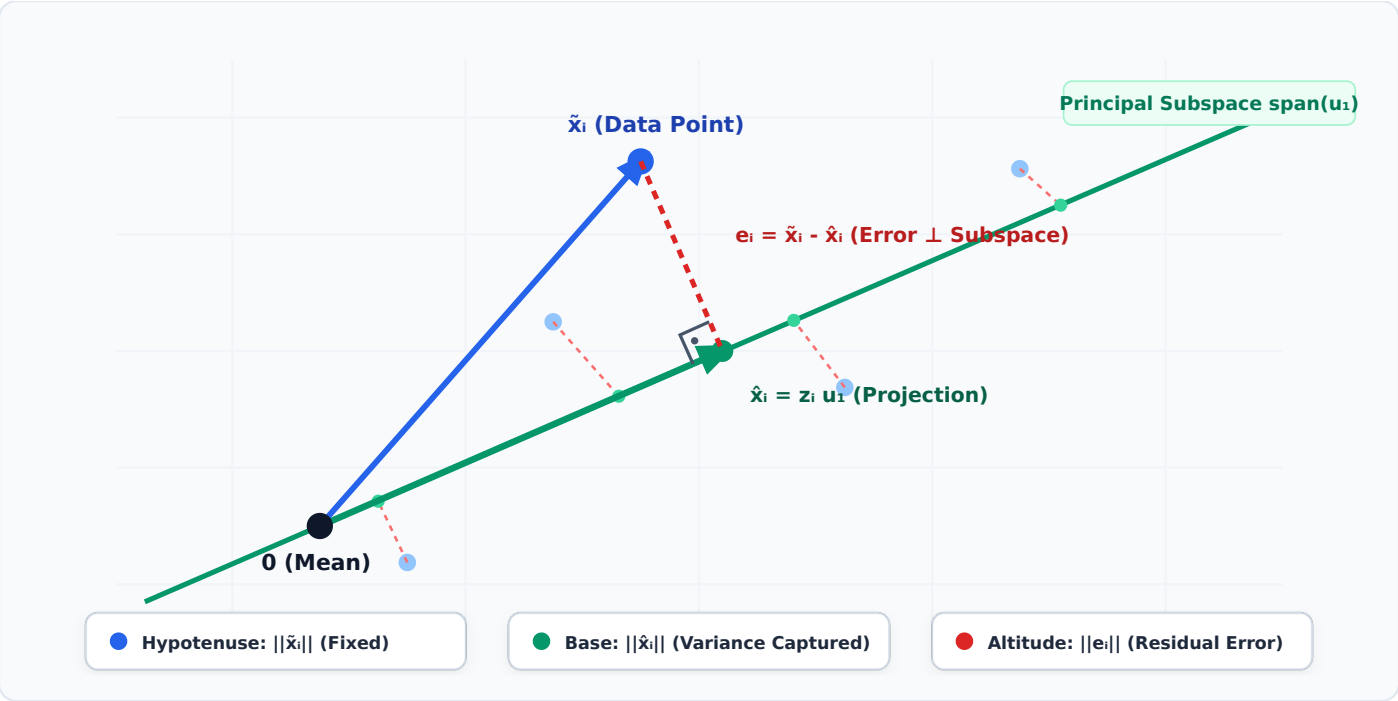
Given centred data $\tilde{\mathbf{x}}_i \in \mathbb{R}^d$, find an orthonormal linear bottleneck basis matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$ ($\mathbf{W}^T\mathbf{W} = \mathbf{I}_k$) minimising the average squared reconstruction error:

$$\mathcal{L}_{\text{PCA}}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} \|\tilde{\mathbf{x}}_i - \hat{\mathbf{x}}_i\|_2^2 = \frac{1}{N}\sum_{i=1}^{N} \|\tilde{\mathbf{x}}_i - \mathbf{W}\mathbf{W}^T\tilde{\mathbf{x}}_i\|_2^2$$

Pythagoras strikes back

For any data point $\tilde{\mathbf{x}}_i$, its squared distance from the origin $\|\tilde{\mathbf{x}}_i\|_2^2$ is **fixed and constant**, that's where the data point lives.

$$\underbrace{\|\tilde{\mathbf{x}}_i\|_2^2}_{\text{Hypotenuse}^2\ (\text{Fixed})} = \underbrace{\|\mathbf{W}\mathbf{W}^T\tilde{\mathbf{x}}_i\|_2^2}_{\text{Base}^2\ (\text{Variance Captured})} + \underbrace{\|\tilde{\mathbf{x}}_i - \mathbf{W}\mathbf{W}^T\tilde{\mathbf{x}}_i\|_2^2}_{\text{Altitude}^2\ (\text{Reconstruction Error})}$$



The Zero-Sum Game

Averaging the Pythagorean identity over all N observations:

$$\underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}}_i\|_2^2}_{\text{Total Variance (Constant)}} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{W}^T\tilde{\mathbf{x}}_i\|_2^2}_{\text{Projected Variance}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}}_i - \hat{\mathbf{x}}_i\|_2^2}_{\text{Reconstruction Error}}$$

Total variance (remember mean is 0) is a fixed property of the dataset. Therefore, every bit of variance captured along the subspace **directly subtracts** from the reconstruction error:

$$\max_{\mathbf{W}} \text{Projected Variance} \iff \min_{\mathbf{W}} \text{Reconstruction Error}$$

Maximising the shadow along the axis forces the perpendicular drop to be as short as possible.

> Cool...

Where Does PCA Fail?

PCA is fast, global, and computed in closed form via SVD. But it has one fundamental limitation:

Linear
PCA can only project onto **flat linear hyperplanes**.

If data lives on a curved, non-linear manifold (e.g. an S-curve, a Swiss roll, or natural images), a linear projection fails...

So what?

In PCA, our bottleneck projection is strictly linear:

$$\mathbf{x} \mapsto \mathbf{z} = \mathbf{W}^T \tilde{\mathbf{x}}, \quad \mathbf{z} \mapsto \hat{\mathbf{x}} = \mathbf{W}\mathbf{z}$$

What if we replace this linear projection with **deep neural networks**?

> It's happening again, we are going from some linear transformation to a non-linear one in the name of AI!

From PCA to Autoencoders

Instead of a linear matrix $\mathbf{W}$, decompose dimensionality reduction into:

The Encoder Network g_ϕ:
Compresses high-dimensional input $\mathbf{x} \in \mathbb{R}^d$ into a low-dimensional bottleneck code $\mathbf{z} \in \mathbb{R}^k$ ($k \ll d$):

$$\mathbf{z} = g_\phi(\mathbf{x})$$

The Decoder Network f_θ:
Reconstructs the original input from the latent code:

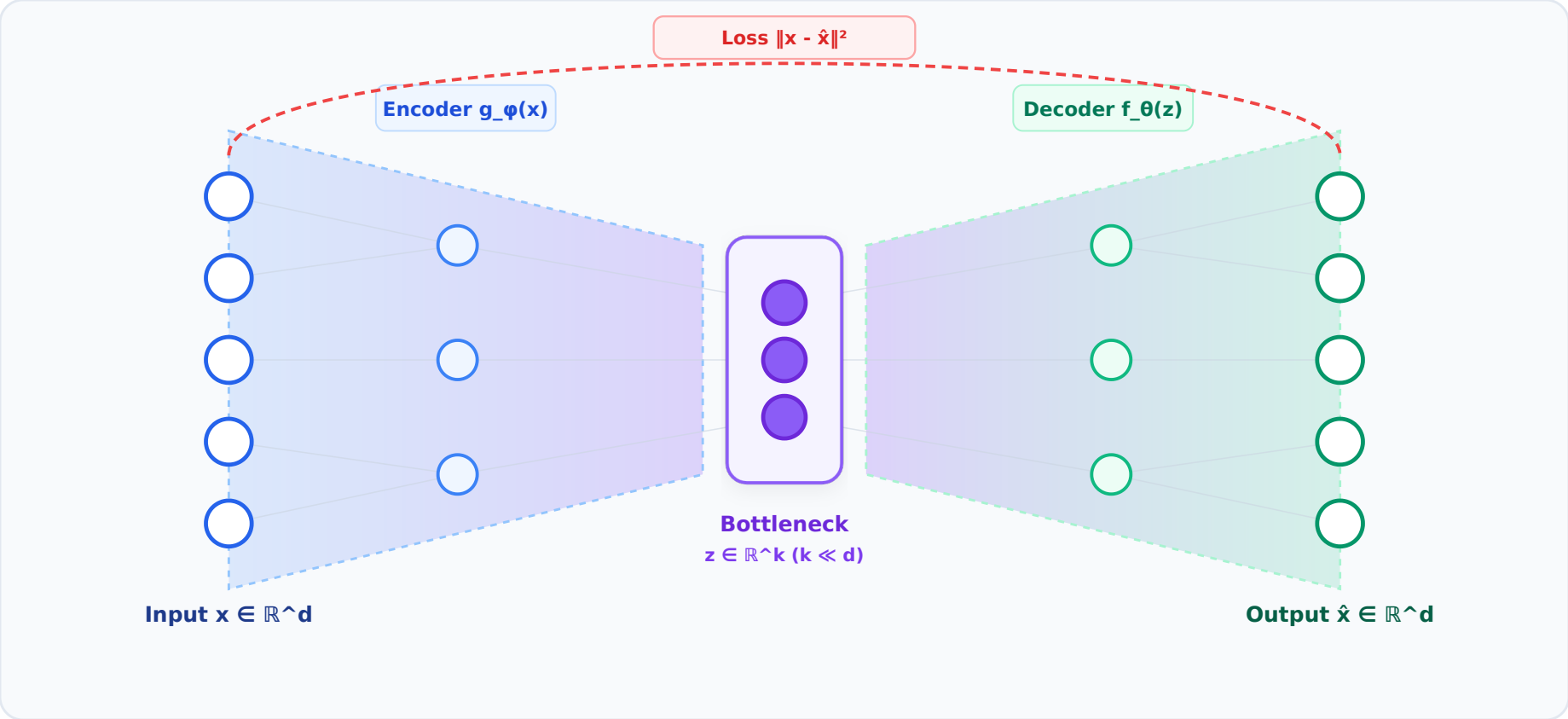
$$\hat{\mathbf{x}} = f_\theta(\mathbf{z}) = f_\theta(g_\phi(\mathbf{x})) \in \mathbb{R}^d$$

The Objective Function:

$$\mathcal{L}_{\text{AE}}(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - f_\theta(g_\phi(\mathbf{x}_i))\|_2^2$$

The Autoencoder Architecture

An Autoencoder learns an **identity mapping through an informational bottleneck**:



- Trained end-to-end via **stochastic gradient descent (SGD)** and **backpropagation**.

Linear AE is PCA

What happens if the encoder and decoder are **strictly linear** (no activation functions)?

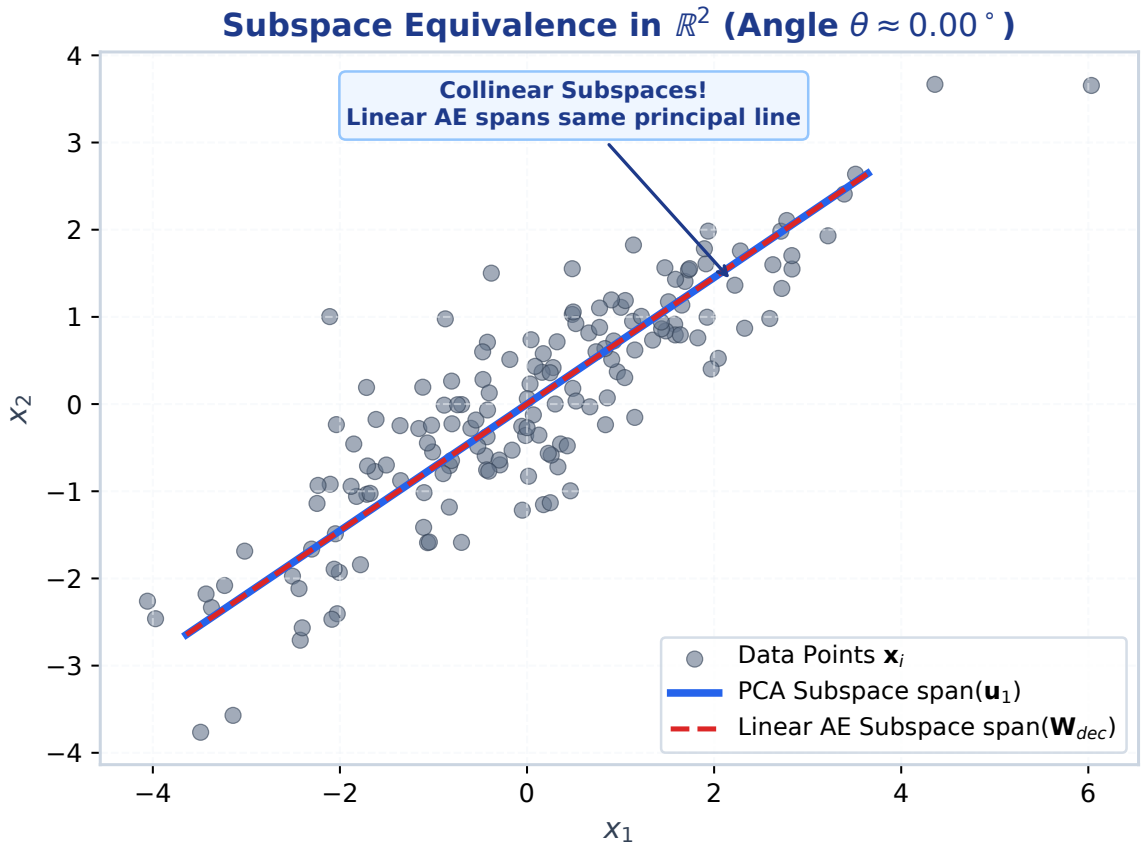
$$g(\mathbf{x}) = \mathbf{W}_1\mathbf{x}, \quad f(\mathbf{z}) = \mathbf{W}_2\mathbf{z} \implies \hat{\mathbf{x}} = \mathbf{W}_2\mathbf{W}_1\mathbf{x}$$

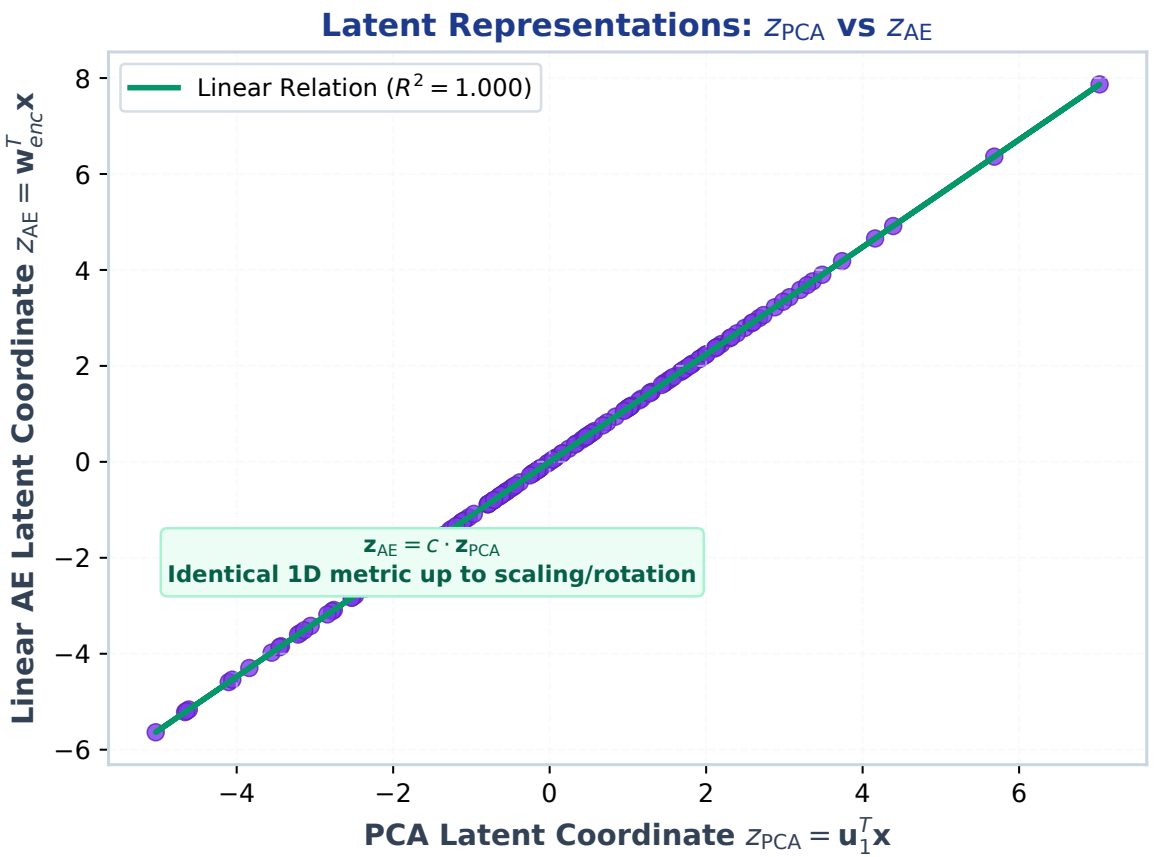
where $\mathbf{W}_1 \in \mathbb{R}^{k\times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d\times k}$.

- **linear algebra (spectral decomposition)** and **connectionist learning (gradient descent)** converge to the exact same geometric solution...

> Ouch... Where is my AI? AGI, SI?

Linear AE is PCA

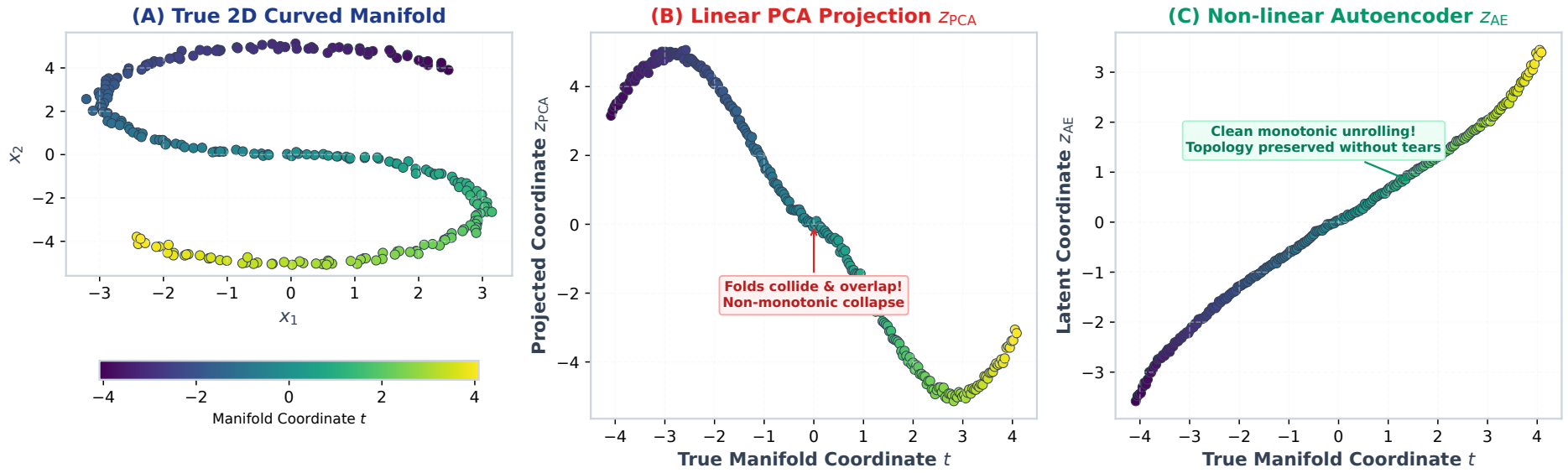




> I don't know what I was expecting when I asked AI to generate a plot, they are equal so we have 2 overlapping lines with some fancy callout boxes 🫠

Non-linear Autoencoders: Manifold Learning

By introducing non-linear activations (ReLU, GELU, $\tanh$), the network can parametrise **curved, non-linear manifolds**:



- **PCA Fails:** Folded branches collide and overlap in linear projection z_{PCA}.
- **Autoencoder Succeeds:** A 1D latent code z_{AE} cleanly **unrolls** the non-linear manifold, discovering the true intrinsic coordinate!

Bottleneck Capacity

What controls what an autoencoder learns? The bottleneck dimension k relative to d:

Undercomplete Autoencoder ($k < d$):

- Latent dimension is smaller than input dimension.
- The network is forced to compress: it must discard noise and capture the dominant manifold structure.

Overcomplete Autoencoder ($k \geq d$):

- What if $k > d$? Without restrictions, the network can cheat by learning the **trivial identity function**:

$$g_\phi(\mathbf{x}) = \mathbf{x}, \quad f_\theta(\mathbf{z}) = \mathbf{z} \implies \hat{\mathbf{x}} = \mathbf{x} \quad (\text{Loss } \mathcal{L} = 0!)$$

- You can enforce regularisation like sparsity $\rightarrow$ Sparse Auto Encoders.

Representation Learning

I want to take this moment to highlight few connecting points:

1. We said Neural Networks can learn intermediate representations, e.g. to solve XOR.
2. These representations are hopefully some compact version of the input data.
3. They are learned through some goal and optimisation: **reconstruction** in the case of autoencoders, **next-token** prediction in the case of language models.

> In the grand scheme, it is about compressing and processing information in a compact and meaningful way.

Denoising Autoencoders

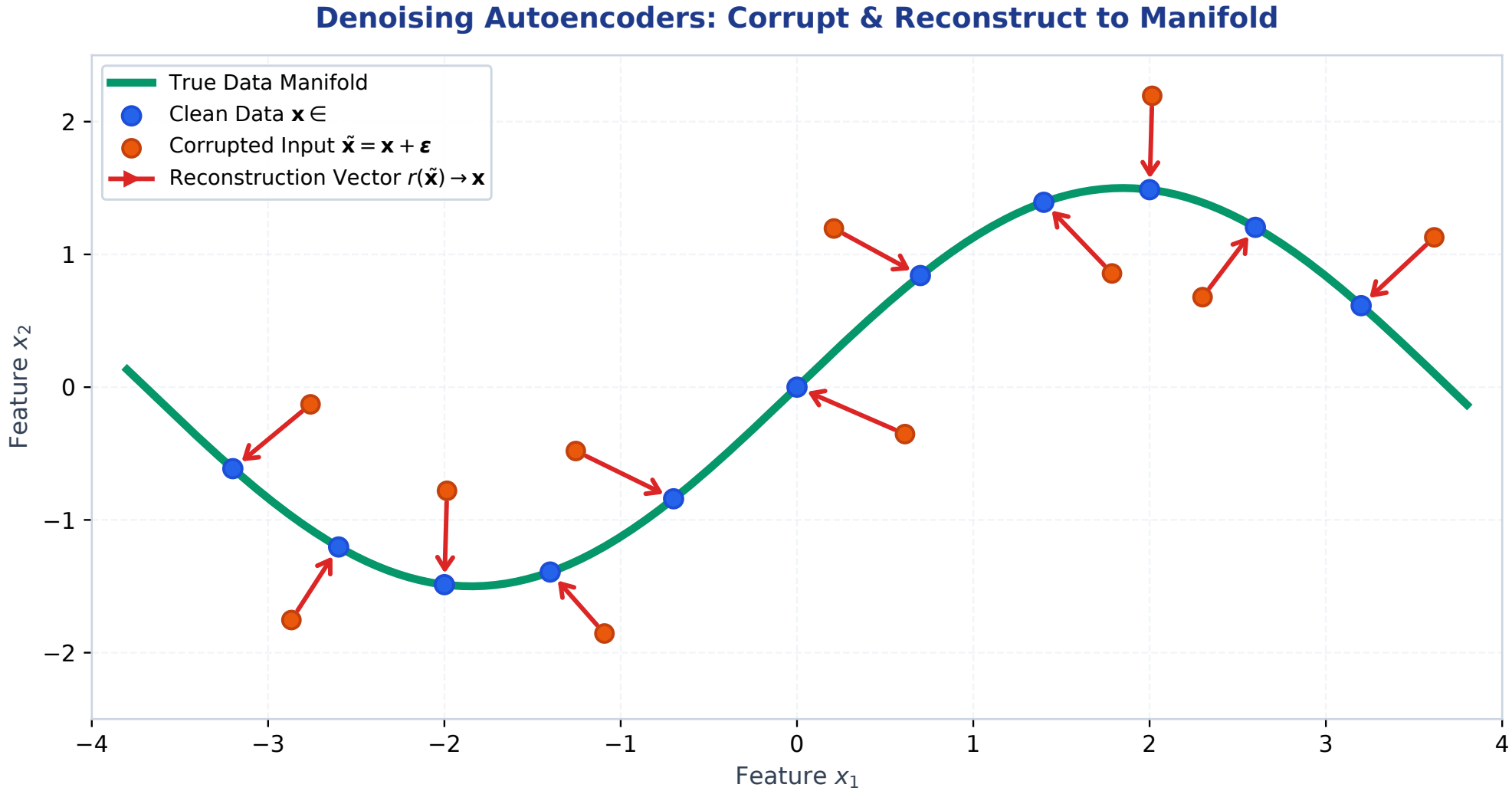
Introduced by **Pascal Vincent et al. (2008)**:

1. Take a clean data point $\mathbf{x} \in \mathbb{R}^d$.
2. Inject stochastic noise to create a corrupted version $\tilde{\mathbf{x}} \sim q(\tilde{\mathbf{x}} \mid \mathbf{x})$:

$$\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$$

3. Feed corrupted $\tilde{\mathbf{x}}$ into the encoder, but force the decoder to reconstruct the **original clean** $\mathbf{x}$:

$$\mathcal{L}_{\text{DAE}}(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} \| \mathbf{x}_i - f_\theta(g_\phi(\tilde{\mathbf{x}}_i)) \|_2^2$$



Diffusion models generate images by chaining this idea.

Interactive Autoencoder Demo

Autoencoder vs PCA Explorer

Non-linear Manifold Unrolling

Dataset: Non-linear S-Curve

Architecture: Non-linear MLP AE (k=1)

▶ Train | Step | Reset

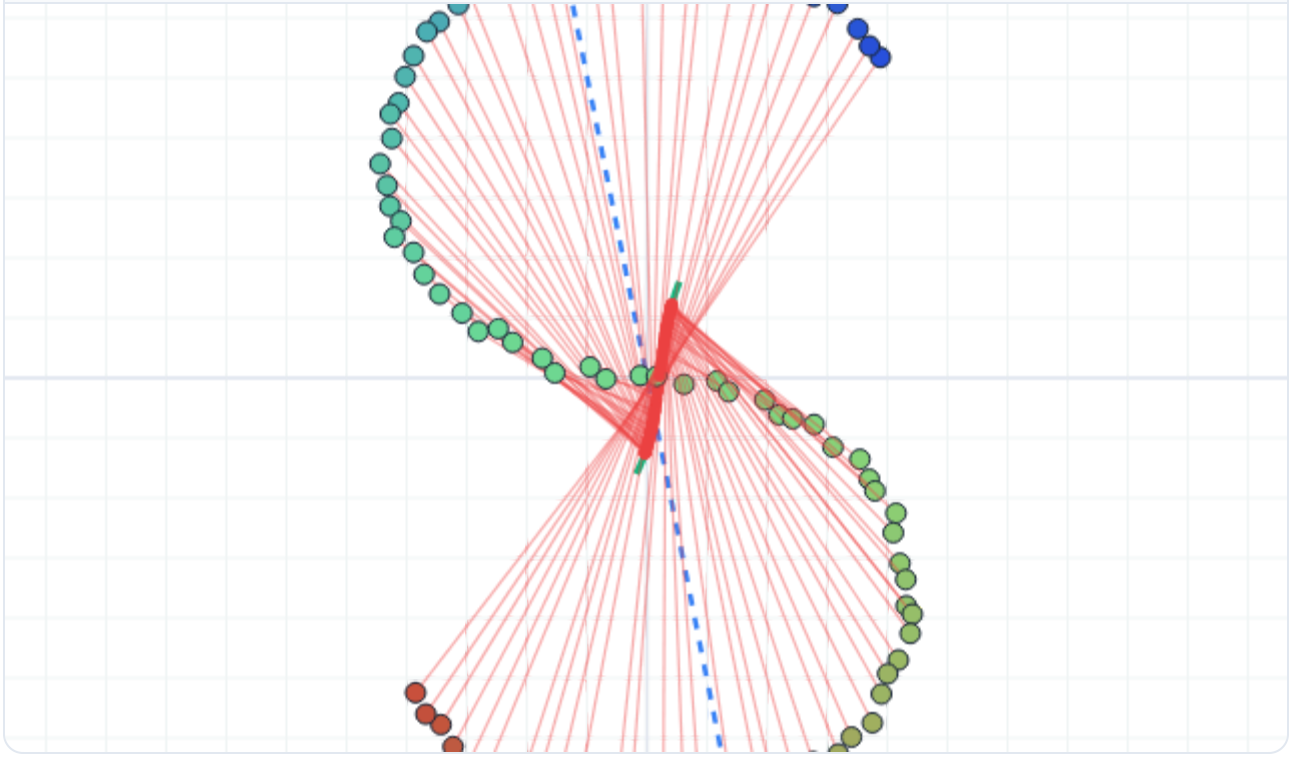
Overlay PCA Line

(A) 2D Data Space & Reconstruction $\hat{\mathbf{x}}$

Original $\mathbf{x}$

Decoded $\hat{\mathbf{x}}$

Manifold $\hat{\mathbf{x}}(z)$



(B) 1D Bottleneck Latent Space $z \in \mathbb{R}^1$

Clean Non-linear Unrolling

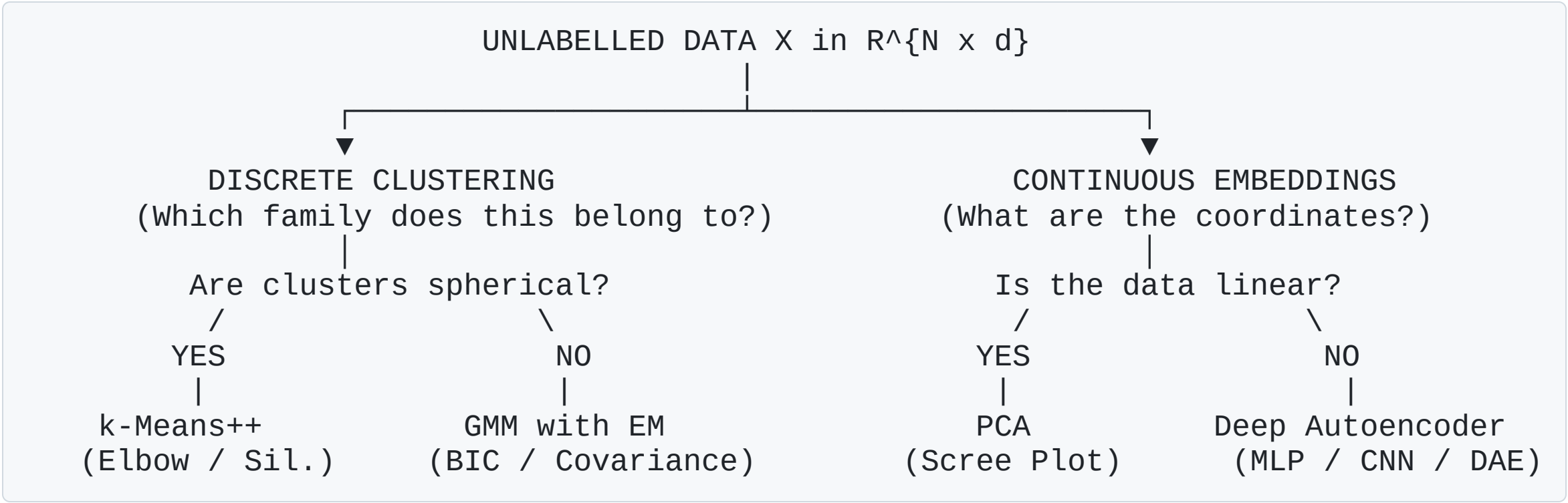


Epoch: 0 **Reconstruction MSE:** 6.2873 **PCA Subspace Collinearity:** N/A (Non-linear Manifold)

Interactive paired learning: Gradient Descent vs Spectral Projection

Method	Target Output	Optimisation	Key Strength	Main Limitation
k-Means	Hard clusters $c_n \in \{1, \ldots, K\}$	Lloyd's coordinate descent	Extremely fast, scales to millions of points	Assumes spherical, equal-variance clusters
GMM	Soft probabilities $\gamma_{nk} \in [0, 1]$	Expectation-Maximisation (EM)	Models uncertainty	Sensitive to initialisation
PCA	Continuous linear axes $\mathbf{z} \in \mathbb{R}^k$	Exact SVD / Eigendecomposition	Closed-form, global optimum, captures max variance	Restricted to flat linear subspaces
Autoencoders	Continuous non-linear coordinates $\mathbf{z} \in \mathbb{R}^k$	Gradient descent (Backprop) on MSE	Parametric, unrolls curved manifolds, reusable	Non-convex loss, requires architecture tuning

The Practitioner's Workflow



Where is my artificial brain?

Don't underestimate representation learning as a core concept. I think we faired better than expected but still:

AI $\rightarrow$ ML $\rightarrow$ Numbers game

Next Part: We'll let machines take actions and evolve.