Lecture 1: Clustering Point Clouds

Master Course — Statistical Learning and Applications

Maximilien Dreveton

Abstract

This first lecture introduces the problem of clustering a cloud of points in $\mathbb{R}^d$. We begin with a formal treatment of what a *clustering function* should satisfy, culminating in Kleinberg's impossibility theorem, which shows that no clustering function can simultaneously satisfy three (seemingly) natural axioms. We then move to practical algorithms: k-means, the Expectation–Maximisation (EM) algorithm for Gaussian mixture models, DBSCAN, and hierarchical clustering. The lecture concludes with a Python session.

Contents

Part I: Theoretical Foundations

1 The Clustering Problem

1.1 Setting and Notation

Let $\mathcal{X} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ be a finite set of n data points in dimension d. A *clustering* (or *partition*) of $\mathcal{X}$ is a family $\mathcal{C} = \{C_1, \ldots, C_K\}$ of non-empty subsets of $\mathcal{X}$ such that

$$C_i \cap C_j = \emptyset \ \ \forall\, i \neq j, \qquad \text{and} \qquad \bigcup_{k=1}^{K} C_k = \mathcal{X}.$$

Definition 1.1 (Distance function)**.** A *dissimilarity* on $\mathcal{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ satisfying, for all $x, y, z \in \mathcal{X}$:

(i) $d(x, y) = 0 \iff x = y$ (identity of indiscernibles),

(ii) $d(x, y) = d(y, x)$ (symmetry).

We do *not* necessarily require the triangle inequality in Kleinberg's framework (though most practical algorithms use metrics).

Definition 1.2 (Clustering function)**.** A *clustering function* is a map f that takes as input a finite set $\mathcal{X}$ together with a dissimilarity d on $\mathcal{X}$, and returns a partition $f(\mathcal{X}, d) = \{C_1, \ldots, C_K\}$ of $\mathcal{X}$ (where K may depend on the input).

Remark 1.3. Note that K is **not** fixed a priori in this definition. The clustering function itself decides the number of clusters.

1.2 Three Natural Axioms

Kleinberg (2002) proposed three properties that one might reasonably demand of any clustering function f.

Definition 1.4 (Scale Invariance)**.** f is *scale-invariant* if for every $(\mathcal{X}, d)$ and every $\alpha > 0$:

$$f(\mathcal{X}, \alpha \cdot d) = f(\mathcal{X}, d).$$

Why is this natural?

Multiplying all distances by a constant (e.g., converting centimetres to inches) should not change which points are grouped together.

Definition 1.5 (Richness)**.** f is *rich* if for every partition $\mathcal{C}$ of $\mathcal{X}$, there exists some distance function d such that $f(\mathcal{X}, d) = \mathcal{C}$.

Why is this natural?

No partition should be *a priori* excluded from the range of f. Given enough freedom in choosing distances, any grouping should be achievable.

Definition 1.6 (Consistency)**.** f is *consistent* if the following holds. Suppose $f(\mathcal{X}, d) = \mathcal{C}$. Let d' be a distance function on $\mathcal{X}$ such that:

- for all x, y in the *same* cluster of $\mathcal{C}$: $d'(x, y) \leq d(x, y)$,

- for all x, y in *different* clusters of $\mathcal{C}$: $d'(x, y) \geq d(x, y)$.

Then $f(\mathcal{X}, d') = \mathcal{C}$.

Why is this natural?

If we shrink intra-cluster distances and expand inter-cluster distances, the clustering should not change — it should only become "more obvious."

1.3 Kleinberg's Impossibility Theorem

Kleinberg's Impossibility Theorem (2002)

Theorem 1.7 (Kleinberg [1])**.** *For $|\mathcal{X}| \geq 2$, there is no clustering function f that simultaneously satisfies:*

(a) Scale Invariance (Definition 1.4),

(b) Richness (Definition 1.5),

(c) Consistency (Definition 1.6).

Proof sketch. Suppose for contradiction that f satisfies all three axioms. Let $|\mathcal{X}| = n \geq 2$.

Step 1. By *Richness*, there exists a distance function d_1 such that $f(\mathcal{X}, d_1)$ places all points in a single cluster: $f(\mathcal{X}, d_1) = \{\mathcal{X}\}$.

Step 2. By *Richness*, there exists a distance function d_n such that $f(\mathcal{X}, d_n)$ places each point in its own cluster (the discrete partition).

Step 3. Consider the "path" of distance functions that continuously interpolates (in a specific sense) between d_1 and d_n. Along this path, the number of clusters must jump from 1 to n. Consider the first point at which the partition changes, say from $\mathcal{C}$ (with $K < n$ clusters) to a refinement $\mathcal{C}'$ (with $K' > K$ clusters).

Step 4. Using *Scale Invariance* and *Consistency*, one can show that both $\mathcal{C}$ and $\mathcal{C}'$ should be the output for a suitably rescaled distance function, yielding a contradiction.

A full formal proof constructs an explicit one-parameter family of distance functions and uses a "critical value" argument. See [1] for the complete proof. □

1.4 Discussion and Consequences

- The theorem does **not** say that clustering is hopeless — it says we must relax at least one axiom.
- **k-means** (k fixed) violates *Richness* (it can only produce partitions into exactly k clusters).
- **Single-linkage** (with a threshold) violates *Consistency*.
- Many practitioners are happy to fix K in advance (violating Richness), which is arguably the least controversial relaxation.
- Analogies with Arrow's impossibility theorem in social choice theory are instructive: impossibility results clarify trade-offs rather than forbidding practice.

Remark 1.8. Ackerman and Ben-David (2009) [7] later proposed refined axiomatic frameworks where near-analogues of all three properties can be simultaneously satisfied by restricting the class of inputs (e.g., requiring "clusterability").

Part II: Practical Clustering Algorithms

2 The k-Means Algorithm

2.1 Objective

Given $\mathcal{X} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ and a target number of clusters K, k-means seeks to minimise the *within-cluster sum of squares* (WCSS):

$$\min_{C,\, \mu_1,\dots,\mu_K} \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2, \tag{1}$$

where $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$ is the centroid of cluster C_k.

Remark 2.1. Problem (1) is NP-hard in general (even for $K = 2$ in general dimension, or for general K in the plane [5, 6]). Lloyd's algorithm provides a practical heuristic.

2.2 Lloyd's Algorithm

Algorithm 1 Lloyd's k-Means Algorithm

Require: Data $\{x_1, \dots, x_n\} \subset \mathbb{R}^d$, number of clusters K
Ensure: Partition $\{C_1, \dots, C_K\}$ and centroids $\{\mu_1, \dots, \mu_K\}$

1: **Initialise:** Choose initial centroids $\mu_1^{(0)}, \dots, \mu_K^{(0)}$ (e.g., randomly from $\mathcal{X}$, or via k-means++)
2: **repeat**
3: **Assignment step:** For each $i = 1, \dots, n$:

$$c_i \leftarrow \arg\min_{k \in \{1,\dots,K\}} \|x_i - \mu_k^{(t)}\|^2$$

4: Set $C_k^{(t+1)} = \{x_i : c_i = k\}$ for each k.
5: **Update step:** For each $k = 1, \dots, K$:

$$\mu_k^{(t+1)} \leftarrow \frac{1}{|C_k^{(t+1)}|} \sum_{x_i \in C_k^{(t+1)}} x_i$$

6: $t \leftarrow t + 1$
7: **until** assignments do not change

Proposition 2.2 (Convergence)**.** *Lloyd's algorithm monotonically decreases the objective* (1) *at each step and converges in a finite number of iterations (since the number of partitions is finite). However, it may converge to a local minimum.*

Proof. The assignment step minimises the objective over partitions for fixed centroids. The update step minimises the objective over centroids for a fixed partition (the minimiser of $\sum_{x_i \in C_k} \|x_i - \mu\|^2$ is the mean). Hence the objective is non-increasing. Since it is bounded below by 0 and the number of distinct partitions of n points into K groups is finite ($\leq K^n$), the algorithm terminates. □

2.3 Initialisation: k-Means++

Algorithm 2 k-Means++ Initialisation [2]

1: Choose μ_1 uniformly at random from $\mathcal{X}$.
2: **for** $j = 2, \ldots, K$ **do**
3: For each x_i, compute $D(x_i) = \min_{l<j} \|x_i - \mu_l\|^2$.
4: Choose $\mu_j = x_i$ with probability proportional to $D(x_i)$.
5: **end for**

Theorem 2.3 (Arthur & Vassilvitskii, 2007 [2])**.** *The k-means++ initialisation guarantees an expected approximation ratio of* $O(\log K)$ *for the optimal k-means cost.*

2.4 Choosing K: The Elbow Method and Silhouette Score

Elbow method. Plot the WCSS as a function of K and look for an "elbow" (point of diminishing returns).

Silhouette score. For each point x_i in cluster C_k, define:

$$a(i) = \frac{1}{|C_k| - 1} \sum_{\substack{x_j \in C_k \\ j \neq i}} \|x_i - x_j\|, \qquad b(i) = \min_{l \neq k} \frac{1}{|C_l|} \sum_{x_j \in C_l} \|x_i - x_j\|,$$

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

Here $a(i)$ measures the *cohesion* of point x_i: it is the average distance from x_i to the other members of its own cluster. A small value of $a(i)$ means the point sits tightly within its cluster. The quantity $b(i)$ measures *separation*: it is the average distance from x_i to the points of the nearest neighbouring cluster. A large value of $b(i)$ means the point is far from any other cluster. Consequently, $s(i) \approx +1$ when $a(i) \ll b(i)$ (point well inside its cluster), $s(i) \approx 0$ when $a(i) \approx b(i)$ (point on the boundary between two clusters), and $s(i) \approx -1$ when $a(i) \gg b(i)$ (point likely misclassified). A mean silhouette score close to 1 indicates well-separated clusters.

2.5 Limitations of k-Means

- Assumes **spherical, equally-sized** clusters (isotropic covariance).
- Sensitive to **outliers** (the mean is not robust).
- Requires K to be specified in advance.
- The **curse of dimensionality**: in high dimension, distances concentrate and all pairwise distances become nearly equal, making the algorithm less effective without preprocessing (e.g., PCA, random projections).

Remark 2.4 (k-Medoids / PAM)**.** A natural variant of k-means is **k-medoids** (Partitioning Around Medoids, [11]), which restricts each center to be an actual data point and minimizes $\sum_{k=1}^{K} \sum_{x_i \in C_k} d(x_i, m_k)$ for an *arbitrary* dissimilarity d (not necessarily Euclidean). This makes it more robust to outliers and applicable to non-vectorial data (e.g., strings with edit distance). The computational cost is higher: $O(n^2)$ per iteration instead of $O(nd)$.

3 Gaussian Mixture Models and EM

3.1 The Generative Model

A *Gaussian Mixture Model* (GMM) with K components assumes that each observation $x_i \in \mathbb{R}^d$ is drawn from:

$$x_i \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k), \tag{2}$$

where $\pi_k \geq 0$, $\sum_k \pi_k = 1$, and $\theta = \{(\pi_k, \mu_k, \Sigma_k)\}_{k=1}^{K}$ are the parameters.

Definition 3.1 (Latent variables)**.** Let $z_i \in \{1, \ldots, K\}$ denote the (unobserved) component assignment of point x_i. Then:

$$\mathbb{P}(z_i = k) = \pi_k, \qquad x_i \mid z_i = k \sim \mathcal{N}(\mu_k, \Sigma_k).$$

The *log-likelihood* of the observed data is:

$$\ell(\theta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \, \phi(x_i; \mu_k, \Sigma_k) \right), \tag{3}$$

where $\phi(\cdot; \mu, \Sigma)$ is the density of $\mathcal{N}(\mu, \Sigma)$.

3.2 The EM Algorithm

Why Direct Maximisation is Hard? The log-likelihood of the observed data is:

$$\ell(\theta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \, \phi(x_i;\, \mu_k,\, \Sigma_k) \right), \tag{4}$$

where $\phi(\cdot;\, \mu, \Sigma)$ is the density of $\mathcal{N}(\mu, \Sigma)$. Direct maximisation is intractable because of the sum over components *inside* the logarithm: we cannot decouple the contributions of the K components.

The Evidence Lower Bound (ELBO). The central idea is to replace $\ell(\theta)$ by a surrogate that is easier to optimise. For each observation x_i, introduce an auxiliary distribution $q_i = (q_{i1}, \ldots, q_{iK})$ over the latent assignment $z_i \in \{1, \ldots, K\}$, with $q_{ik} \geq 0$ and $\sum_k q_{ik} = 1$. By Jensen's inequality applied to the concave logarithm:

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \, \phi(x_i;\, \mu_k,\, \Sigma_k) \\
&= \sum_{i=1}^{n} \log \sum_{k=1}^{K} q_{ik} \, \frac{\pi_k \, \phi(x_i;\, \mu_k,\, \Sigma_k)}{q_{ik}} \\
&\geq \sum_{i=1}^{n} \sum_{k=1}^{K} q_{ik} \log \frac{\pi_k \, \phi(x_i;\, \mu_k,\, \Sigma_k)}{q_{ik}}.
\end{aligned}$$

The right-hand side is called the *evidence lower bound* (ELBO):

$$\mathcal{L}(q, \theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} q_{ik} \log \frac{\pi_k \, \phi(x_i;\, \mu_k,\, \Sigma_k)}{q_{ik}}. \tag{5}$$

The name comes from the fact that $p(x_i \mid \theta) = \sum_k \pi_k \, \phi(x_i;\, \mu_k,\, \Sigma_k)$ is sometimes called the *evidence* (the probability of the observed data after marginalising over the latent variables), and $\mathcal{L}$ is a lower bound on its logarithm. More precisely, the gap is exactly a sum of KL divergences:

$$\ell(\theta) - \mathcal{L}(q, \theta) = \sum_{i=1}^{n} \mathrm{KL}\big(q_i \,\|\, p(z_i \mid x_i, \theta)\big) \geq 0. \tag{6}$$

The key property is that, unlike $\ell(\theta)$, the ELBO $\mathcal{L}(q, \theta)$ has the logarithm *inside* the sum over components, making it amenable to closed-form optimisation over both q and θ.

E-step: Maximising the ELBO over q. For fixed $\theta = \theta^{(t)}$, we maximise $\mathcal{L}(q, \theta^{(t)})$ over the auxiliary distributions q_i. From (6), this is equivalent to setting the KL divergence to zero, which requires q_i to equal the true posterior:

$$\gamma_{ik}^{(t)} \;=\; q_{ik}^{(t)} \;=\; p(z_i = k \mid x_i,\, \theta^{(t)}) \;=\; \frac{\pi_k^{(t)} \, \phi\big(x_i;\, \mu_k^{(t)},\, \Sigma_k^{(t)}\big)}{\sum_{l=1}^{K} \pi_l^{(t)} \, \phi\big(x_i;\, \mu_l^{(t)},\, \Sigma_l^{(t)}\big)}. \tag{7}$$

The quantity $\gamma_{ik}^{(t)}$ is called the *responsibility* of component k for observation x_i: it represents the soft probability that x_i belongs to cluster k under the current parameters. After this step, the bound is *tight*: $\mathcal{L}(q^{(t)}, \theta^{(t)}) = \ell(\theta^{(t)})$.

We also define the effective number of points assigned to component k:

$$N_k^{(t)} = \sum_{i=1}^{n} \gamma_{ik}^{(t)}.$$

M-step: Maximising the ELBO over θ For fixed $q = q^{(t)}$ (i.e. the responsibilities just computed), we maximise $\mathcal{L}(q^{(t)}, \theta)$ over $\theta = (\pi_k, \mu_k, \Sigma_k)_{k=1}^{K}$. Since the logarithm in (5) now sits inside the sum, the optimisation decouples across components and yields closed-form updates. Concretely, expanding $\mathcal{L}$ and differentiating:

- **Mixing weights** (maximising subject to $\sum_k \pi_k = 1$, via a Lagrange multiplier):

$$\pi_k^{(t+1)} = \frac{N_k^{(t)}}{n}. \tag{8}$$

  This is simply the fraction of (soft) points assigned to component k.

- **Means** (setting $\partial \mathcal{L} / \partial \mu_k = 0$):

$$\mu_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{i=1}^{n} \gamma_{ik}^{(t)} \, x_i. \tag{9}$$

  This is the weighted average of all observations, with weights given by the responsibilities.

- **Covariances** (setting $\partial \mathcal{L} / \partial \Sigma_k = 0$):

$$\Sigma_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{i=1}^{n} \gamma_{ik}^{(t)} \big(x_i - \mu_k^{(t+1)}\big)\big(x_i - \mu_k^{(t+1)}\big)^{\top}. \tag{10}$$

  This is the weighted covariance matrix for component k.

Why It Works: Monotonicity

Theorem 3.2 (Monotonicity of EM)**.** *The EM algorithm satisfies* $\ell(\theta^{(t+1)}) \geq \ell(\theta^{(t)})$ *for all* t*.*

Proof sketch. Chaining the two steps:

$$\ell(\theta^{(t+1)}) \;\geq\; \mathcal{L}(q^{(t)}, \theta^{(t+1)}) \;\geq\; \mathcal{L}(q^{(t)}, \theta^{(t)}) \;=\; \ell(\theta^{(t)}).$$

The first inequality holds because the ELBO always lower-bounds the log-likelihood (6). The second holds because the M-step maximises $\mathcal{L}$ over θ. Equality on the right holds because the E-step made the bound tight. □

Full Algorithm. See Algorithm 3. EM provides a natural way to obtain hard cluster assignments from a fitted GMM: simply assign each observation to the component with the highest responsibility,

$$\hat{z}_i = \arg\max_{k \in \{1,\ldots,K\}} \gamma_{ik}.$$

This is the Bayes-optimal assignment under zero-one loss. Unlike k-means, however, the soft responsibilities γ_{ik} remain available and convey useful information about the uncertainty of each assignment: for instance, a point with $\gamma_{i1} = 0.52$ and $\gamma_{i2} = 0.48$ sits near the decision boundary between two clusters, whereas $\gamma_{i1} \approx 1$ indicates a confident assignment.

Algorithm 3 Expectation–Maximisation for GMMs

Require: Data $\{x_1, \ldots, x_n\}$, number of components K
Ensure: Parameters $\hat{\theta} = \{(\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k)\}_{k=1}^{K}$

1: **Initialise** $\theta^{(0)}$ (e.g., from k-means output)
2: **repeat**
3: **E-step:** Compute responsibilities

$$\gamma_{ik}^{(t)} = \frac{\pi_k^{(t)}\, \phi(x_i; \mu_k^{(t)}, \Sigma_k^{(t)})}{\sum_{l=1}^{K} \pi_l^{(t)}\, \phi(x_i; \mu_l^{(t)}, \Sigma_l^{(t)})}, \qquad N_k^{(t)} = \sum_{i=1}^{n} \gamma_{ik}^{(t)} \tag{11}$$

4: **M-step:** Update parameters

$$\pi_k^{(t+1)} = \frac{N_k^{(t)}}{n}, \tag{12}$$

$$\mu_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{i=1}^{n} \gamma_{ik}^{(t)}\, x_i, \tag{13}$$

$$\Sigma_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{i=1}^{n} \gamma_{ik}^{(t)} (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^\top. \tag{14}$$

5: $t \leftarrow t + 1$
6: **until** convergence of $\ell(\theta^{(t)})$

3.3 Relation Between k-Means and EM

Proposition 3.3. *k-means can be viewed as a limiting case of EM for GMMs where all covariance matrices are $\Sigma_k = \sigma^2 I_d$ with $\sigma^2 \to 0$. In this limit, the responsibilities γ_{ik} become hard assignments (0 or 1), and the M-step reduces to computing centroids.*

Proof. With $\Sigma_k = \sigma^2 I_d$ and $\pi_k = 1/K$, the Gaussian density of component k evaluated at x_i is:

$$\phi(x_i;\, \mu_k,\, \sigma^2 I_d) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\!\left(-\frac{\|x_i - \mu_k\|^2}{2\sigma^2}\right).$$

The responsibility of component k for observation x_i is:

$$\gamma_{ik} = \frac{\pi_k\, \phi(x_i;\, \mu_k,\, \sigma^2 I_d)}{\sum_{j=1}^{K} \pi_j\, \phi(x_i;\, \mu_j,\, \sigma^2 I_d)} = \frac{\exp\!\big(-\|x_i - \mu_k\|^2/(2\sigma^2)\big)}{\sum_{j=1}^{K} \exp\!\big(-\|x_i - \mu_j\|^2/(2\sigma^2)\big)},$$

where the prefactors $(2\pi\sigma^2)^{-d/2}$ and the equal weights $1/K$ cancel between numerator and denominator. Dividing numerator and denominator by $\exp\!\big(-\min_j \|x_i - \mu_j\|^2/(2\sigma^2)\big)$, we obtain:

$$\gamma_{ik} = \frac{\exp\!\big(-(\|x_i - \mu_k\|^2 - \min_j \|x_i - \mu_j\|^2)/(2\sigma^2)\big)}{\sum_{j=1}^{K} \exp\!\big(-(\|x_i - \mu_j\|^2 - \min_j \|x_i - \mu_j\|^2)/(2\sigma^2)\big)}.$$

Each exponent is non-positive, and equals zero if and only if component j is the nearest to x_i. As $\sigma^2 \to 0$, the terms with strictly negative exponents vanish, so (assuming a unique nearest centroid):

$$\gamma_{ik} \xrightarrow{\sigma^2 \to 0} \begin{cases} 1 & \text{if } k = \arg\min_j \|x_i - \mu_j\|^2, \\ 0 & \text{otherwise.} \end{cases}$$

This is exactly the hard assignment step of k-means.

For the M-step, the general mean update is:

$$\mu_k^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ik}^{(t)}\, x_i}{\sum_{i=1}^{n} \gamma_{ik}^{(t)}}.$$

Under hard assignments ($\gamma_{ik} \in \{0, 1\}$), this becomes:

$$\mu_k^{(t+1)} = \frac{1}{|C_k^{(t)}|} \sum_{i \in C_k^{(t)}} x_i,$$

which is the centroid update of k-means. Since both the assignment and update steps coincide, the EM iterates reduce to the k-means iterates. □

Soft vs. Hard Clustering

	k-Means	EM (GMM)
Assignment	Hard ($x_i \in C_k$)	Soft ($\gamma_{ik} \in [0, 1]$)
Cluster shape	Spherical	Ellipsoidal
Cluster size	Equal	Varying (via π_k)
Objective	WCSS	Log-likelihood

3.4 Model Selection: BIC and AIC

To choose K in a principled way, one can use information criteria:

$$\text{AIC} = -2\ell(\hat{\theta}) + 2p, \tag{15}$$

$$\text{BIC} = -2\ell(\hat{\theta}) + p \log n, \tag{16}$$

where $p = K\big(1 + d + \frac{d(d+1)}{2}\big) - 1$ is the number of free parameters. Select the K that minimises the criterion.

3.5 High-Dimensional Considerations

- In high dimension, Σ_k has $O(d^2)$ parameters per component $\Rightarrow$ estimation becomes unreliable unless $n \gg d^2$.
- Common remedies: **diagonal** covariance ($\Sigma_k = \mathrm{diag}(\sigma_{k1}^2, \ldots, \sigma_{kd}^2)$), **tied** covariance ($\Sigma_k = \Sigma$ for all k), or **regularisation**.
- **Dimension reduction** (PCA, random projections) before clustering is standard practice.

4 DBSCAN

4.1 Motivation

k-means and GMMs assume a fixed number of convex/ellipsoidal clusters. Many real datasets contain clusters of **arbitrary shape**, varying density, and **outliers**. DBSCAN (Density-Based Spatial Clustering of Applications with Noise, Ester et al., 1996 [3]) addresses these issues.

4.2 Key Definitions

Fix parameters $\varepsilon > 0$ (neighbourhood radius) and $\texttt{minPts} \in \mathbb{N}$ (minimum number of points).

Definition 4.1 (ε-neighbourhood)**.** $N_\varepsilon(x_i) = \{x_j \in \mathcal{X} : \|x_i - x_j\| \leq \varepsilon\}$.

Definition 4.2 (Core, border, noise points)**.**

- x_i is a **core point** if $|N_\varepsilon(x_i)| \geq \texttt{minPts}$.
- x_i is a **border point** if it is not a core point but belongs to $N_\varepsilon(x_j)$ for some core point x_j.
- x_i is a **noise point** otherwise.

Definition 4.3 (Density-reachability)**.** Point x_j is *directly density-reachable* from x_i if x_i is a core point and $x_j \in N_\varepsilon(x_i)$.

Point x_j is *density-reachable* from x_i if there exists a chain $x_i = p_1, p_2, \ldots, p_m = x_j$ where each p_{l+1} is directly density-reachable from p_l.

Points x_i and x_j are *density-connected* if there exists a point x_q such that both x_i and x_j are density-reachable from x_q.

A cluster in DBSCAN is a maximal set of density-connected points. Points that are not density-reachable from any core point are classified as noise.

4.3 Algorithm

Algorithm 4 DBSCAN

Require: Data $\{x_1, \ldots, x_n\}$, parameters ε, `minPts`
Ensure: Cluster labels $\ell_1, \ldots, \ell_n$ ($\ell_i = -1$ for noise)

Mark all points as UNVISITED
$C \leftarrow 0$ ▷ cluster counter
for each UNVISITED point x_i **do**
    Mark x_i as VISITED
    $\mathcal{N} \leftarrow N_\varepsilon(x_i)$
    if $|\mathcal{N}| <$ `minPts` **then**
        Mark x_i as NOISE
    else
        $C \leftarrow C + 1$
        EXPANDCLUSTER($x_i, \mathcal{N}, C, \varepsilon,$ `minPts`)
    end if
end for

function EXPANDCLUSTER($x_i, \mathcal{N}, C, \varepsilon,$ `minPts`)
    Add x_i to cluster C
    for each $x_j \in \mathcal{N}$ **do**
        if x_j is UNVISITED **then**
            Mark x_j as VISITED
            $\mathcal{N}' \leftarrow N_\varepsilon(x_j)$
            if $|\mathcal{N}'| \geq$ `minPts` **then**
                $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}'$
            end if
        end if
        if x_j is not yet in any cluster **then**
            Add x_j to cluster C
        end if
    end for
end function

4.4 Properties

- **Complexity:** $O(n^2)$ in general; $O(n \log n)$ with spatial indexing (e.g., KD-tree, ball tree) when d is moderate.
- **Does not require** K to be specified in advance.
- **Identifies outliers** explicitly.
- Struggles with clusters of **varying densities**.
- The **curse of dimensionality** severely affects the ε-ball: in high dimensions, the volume of the ball grows as ε^d while data becomes sparse, making parameter selection very difficult.

Choosing ε. A common heuristic: compute the distance to the `minPts`-th nearest neighbour for each point, sort these distances, and look for a "knee" in the plot.

5 Hierarchical Clustering

5.1 Agglomerative (Bottom-Up) Approach

Algorithm 5 Agglomerative Hierarchical Clustering

Require: Pairwise distance matrix $D \in \mathbb{R}^{n \times n}$, linkage criterion
Ensure: Dendrogram (nested sequence of partitions)
1: Initialise: each point is its own cluster: $\mathcal{C} = \{\{x_1\}, \ldots, \{x_n\}\}$
2: **while** $|\mathcal{C}| > 1$ **do**
3: Find the pair (C_a, C_b) with smallest inter-cluster distance (according to the linkage criterion)
4: Merge: $C_{ab} \leftarrow C_a \cup C_b, \quad \mathcal{C} \leftarrow (\mathcal{C} \setminus \{C_a, C_b\}) \cup \{C_{ab}\}$
5: Record the merge and the distance at which it occurred
6: **end while**

5.2 Linkage Criteria

Let C_a, C_b be two clusters. Common inter-cluster distances:

$$d_{\text{single}}(C_a, C_b) = \min_{x \in C_a,\, y \in C_b} \|x - y\| \qquad \text{(single linkage)}, \tag{17}$$

$$d_{\text{complete}}(C_a, C_b) = \max_{x \in C_a,\, y \in C_b} \|x - y\| \qquad \text{(complete linkage)}, \tag{18}$$

$$d_{\text{average}}(C_a, C_b) = \frac{1}{|C_a||C_b|} \sum_{x \in C_a} \sum_{y \in C_b} \|x - y\| \qquad \text{(average linkage)}, \tag{19}$$

$$d_{\text{Ward}}(C_a, C_b) = \frac{|C_a||C_b|}{|C_a| + |C_b|} \|\bar{x}_a - \bar{x}_b\|^2 \qquad \text{(Ward's method)}. \tag{20}$$

Linkage comparison

- **Single linkage:** can find non-convex clusters but suffers from the *chaining effect*.
- **Complete linkage:** produces compact clusters but is sensitive to outliers.
- **Ward's method:** minimises variance (equivalent to k-means objective at each merge step); tends to produce balanced, spherical clusters.

5.3 Dendrograms and Cutting

The output of agglomerative clustering is a **dendrogram**: a tree whose leaves are the data points and whose internal nodes represent merges. Cutting the dendrogram at a given height h yields a flat partition.

- Cut at a fixed number of clusters K: take the partition obtained just before the $(n-K)$-th merge.
- Cut at a fixed distance threshold h: merge all clusters connected below height h.

5.4 Complexity and Scalability

- **Naive implementation:** $O(n^3)$ time, $O(n^2)$ space: at each of the $n-1$ merge steps, scan the full distance matrix to find the closest pair, then update it.

- **Single linkage** can be computed in $O(n^2)$ time by reducing the problem to a minimum spanning tree (Kruskal/Prim), since single-linkage merges correspond exactly to MST edges in increasing order of weight.

- **Nearest-neighbour chain algorithm.** A linkage criterion d is called *reducible* if, for any three clusters A, B, C:

$$d(A,B) \le \min\bigl\{d(A,C),\ d(B,C)\bigr\} \implies d(A \cup B,\ C) \le \min\bigl\{d(A,C),\ d(B,C)\bigr\}.$$

  Intuitively, merging two clusters that are mutually nearest neighbours cannot create a pair that should have been merged earlier. Ward's method, complete linkage, and average linkage (weighted and unweighted) are all reducible; single linkage is also reducible but is handled more efficiently by the MST approach.

  The algorithm maintains a stack and repeatedly follows *nearest-neighbour chains*: starting from any cluster, push its nearest neighbour, then that cluster's nearest neighbour, and so on. For reducible linkages, such a chain must terminate in a **mutual nearest-neighbour pair** (A, B) (that is, A's nearest neighbour is B and vice versa) which is then guaranteed to be a valid next merge. After merging, the chain is unwound and the process continues.

  Theorem 5.1 (Nearest-neighbour chain complexity)**.** *For any reducible linkage criterion, the nearest-neighbour chain algorithm computes the complete hierarchy in $O(n^2)$ time and $O(n^2)$ space.*

  Proof sketch. Each cluster is pushed onto the stack at most once before being merged, so the total number of nearest-neighbour queries across all chains is $O(n)$. Each query scans the current set of active clusters, which is at most n, giving $O(n)$ per query. The total work for all queries is therefore $O(n^2)$. Initialising the distance matrix also costs $O(n^2)$, and updating distances after each merge (using the Lance–Williams formula) takes $O(n)$ per merge, for $O(n^2)$ in total over all $n-1$ merges. □

- **Summary of complexities:**

Linkage	Best known time	Algorithm
Single	$O(n^2)$	MST (Prim/Kruskal)
Complete	$O(n^2)$	NN-chain
Average	$O(n^2)$	NN-chain
Ward	$O(n^2)$	NN-chain

  All methods require $O(n^2)$ space for the distance matrix. For large n, this remains expensive; consider **BIRCH** or mini-batch approaches.

5.5 Complexity and Scalability

- Naive implementation: $O(n^3)$ time, $O(n^2)$ space.
- Single linkage can be computed in $O(n^2)$ time using a minimum spanning tree (Kruskal/Prim).
- For large n, hierarchical clustering becomes expensive; consider **BIRCH** or mini-batch approaches.

6 Comparison and Summary

Method	K required?	Cluster shape	Outliers	Complexity	High-d
k-means	Yes	Spherical	Sensitive	$O(nKdt)$	Moderate
EM (GMM)	Yes	Ellipsoidal	Sensitive	$O(nKd^2t)$	Difficult
DBSCAN	No	Arbitrary	Robust	$O(n^2)/O(n \log n)$	Difficult
Hierarchical	No*	Depends on linkage	Depends	$O(n^2)$–$O(n^3)$	Moderate

*A cut level or target K is chosen *a posteriori*.

7 Evaluating Clustering Quality

A fundamental difficulty in clustering is *assessment*: how do we know whether a clustering is good? We distinguish two families of criteria.

7.1 External Metrics (Ground Truth Available)

Suppose we have access to a ground-truth partition $\mathcal{C}^* = \{C_1^*, \dots, C_K^*\}$ and a predicted partition $\hat{\mathcal{C}} = \{\hat{C}_1, \dots, \hat{C}_{\hat{K}}\}$.

Definition 7.1 (Rand Index)**.** Consider all $\binom{n}{2}$ pairs of points. Let a be the number of pairs that are in the *same* cluster in both $\mathcal{C}^*$ and $\hat{\mathcal{C}}$, and b the number of pairs that are in *different* clusters in both. The **Rand Index** is

$$\mathrm{RI} = \frac{a+b}{\binom{n}{2}}.$$

The Rand Index lies in $[0,1]$ but does not have a meaningful zero baseline (a random partition can achieve a high RI). This motivates the adjusted version.

Definition 7.2 (Adjusted Rand Index)**.** The **Adjusted Rand Index** corrects for chance:

$$\mathrm{ARI} = \frac{\mathrm{RI} - \mathbb{E}[\mathrm{RI}]}{\max(\mathrm{RI}) - \mathbb{E}[\mathrm{RI}]},$$

where the expectation is taken over random partitions with the same cluster sizes. ARI $= 1$ for perfect agreement, ARI ≈ 0 for random partitions, and ARI < 0 is possible.

Definition 7.3 (Normalized Mutual Information)**.** Let U and V be the random variables corresponding to the cluster assignment of a randomly chosen point under $\mathcal{C}^*$ and $\hat{\mathcal{C}}$ respectively. The **Normalized Mutual Information** is

$$\mathrm{NMI} = \frac{I(U;V)}{\frac{1}{2}(H(U) + H(V))},$$

where $I(U;V)$ is the mutual information and $H(\cdot)$ the entropy. NMI $\in [0,1]$, with 1 indicating perfect agreement.

Remark 7.4. A common mistake is to evaluate clustering with *accuracy*. Since cluster labels are arbitrary (there is no canonical mapping between predicted labels and true labels), one would need to solve an optimal assignment problem first. ARI and NMI are permutation-invariant by construction and are therefore preferred.

7.2 Internal Metrics (No Ground Truth)

When no ground truth exists (the common case in practice) we rely on internal criteria, such as the Silhouette Score (already discussed in Section 2.4).

Definition 7.5 (Silhouette Score)**.** For a point x_i in cluster C_k, let

$$a(i) = \frac{1}{|C_k| - 1} \sum_{j \in C_k, j \neq i} d(x_i, x_j), \qquad b(i) = \min_{\ell \neq k} \frac{1}{|C_\ell|} \sum_{j \in C_\ell} d(x_i, x_j).$$

The silhouette of point i is $s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, 1]$. The **mean silhouette score** is $\bar{s} = \frac{1}{n} \sum_{i=1}^{n} s(i)$.

For model-based approaches (GMM), the **BIC** and **AIC** (already discussed in Section 3) provide principled model-selection criteria.

Practical Clustering Workflow

1. **Visualize** the data in 2D (PCA or t-SNE/UMAP) to build intuition about cluster shapes, density, and outliers.
2. **Standardize** features (zero mean, unit variance) unless the original scales carry meaning.
3. **Choose an algorithm** based on prior knowledge:
   - Convex, roughly equal-size clusters → k-means.
   - Elliptical, varying-size clusters → GMM/EM.
   - Arbitrary shapes, noise expected → DBSCAN/HDBSCAN.
   - Need a hierarchy / don't know K → agglomerative.
4. **Select hyperparameters** (K, ε, linkage, ...) using internal criteria (silhouette, BIC, dendrogram, k-distance plot).
5. **Validate**: if ground truth exists, compute ARI/NMI. If not, check stability (does the result change significantly with different initializations or subsamples?).
6. **Iterate**: clustering is exploratory. Try multiple approaches and compare.

From point clouds to graphs — preview of Lecture 2

All algorithms in this lecture assume the data are points in $\mathbb{R}^d$ (or at least that pairwise distances are meaningful). But many datasets are naturally represented as *graphs*: social networks, biological interaction networks, the internet. Clustering *nodes* of a graph — called **community detection** — requires fundamentally different tools. In Lecture 2, we will see how the **Louvain** algorithm directly optimizes a graph-based objective (modularity), and how **spectral clustering** builds a beautiful bridge between the two worlds: it constructs a similarity graph from point-cloud data, then uses eigenvectors of the graph Laplacian to embed nodes into $\mathbb{R}^K$ before applying k-means. Spectral clustering thus serves as the natural transition from today's lecture to graph-based methods.

Part III: Hands-On Python Exercises

Setup

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons, make_circles
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
np.random.seed(10)
```

Listing 1: Imports and configuration

Exercise 1. [k-Means Clustering] **Useful tools:** `sklearn.datasets.make_blobs`, `sklearn.cluster.KMeans`, `sklearn.metrics.silhouette_score`.

1. Generate a 2D dataset with $n = 300$ points and $K = 4$ isotropic Gaussian clusters using `make_blobs`. Visualise the data with a scatter plot.
2. Run k-means with $K = 4$. Plot the resulting cluster assignments and the centroids. Do you recover the true clusters?
3. **Sensitivity to initialisation.** Run k-means 5 times with `n_init=1` and `init='random'`. Plot each result side by side. Do you always obtain the same partition? What does the parameter `n_init` do in the default setting?
4. **Choosing K: the elbow method.** For $K = 1, 2, \ldots, 8$, run k-means and record the inertia (`model.inertia_`). Plot inertia vs. K. Where is the "elbow"?
5. **Choosing K: silhouette analysis.** For the same values of K, compute the mean silhouette score. Which K maximises it? Does it agree with the elbow method?
6. **A failure case.** Generate a dataset of two half-moons using `sklearn.datasets.make_moons` ($n = 500$, `noise=0.1`). Run k-means with $K = 2$. Is the result satisfactory? Explain why k-means fails here.

Exercise 2. [Gaussian Mixture Models] **Useful tools:** `sklearn.mixture.GaussianMixture`, `model.bic()`, `model.aic()`, `model.predict_proba()`.

1. Generate an *anisotropic* dataset: start from `make_blobs` ($K = 3$, $n = 500$), then apply a linear transformation (e.g. multiply X by a 2×2 matrix such as $\begin{pmatrix} 0.8 & -0.6 \\ 0.5 & 0.9 \end{pmatrix}$) so that the clusters become elongated. Visualise the result.
2. Fit a GMM with $K = 3$ and `covariance_type='full'`. Plot the cluster assignments. Compare visually with k-means ($K = 3$) on the same data. Which method handles the elongated clusters better?
3. **Soft assignments.** Use `predict_proba` to obtain the responsibility matrix. Pick a few points near the boundary between two clusters and display their responsibilities. What does this tell you that a hard assignment does not?
4. **Effect of covariance type.** Fit GMMs with `covariance_type` set to `'spherical'`, `'diag'`, `'tied'`, and `'full'`. For each, plot the resulting assignments. Which variant is equivalent to k-means (up to soft assignments)?

5. **Model selection with BIC.** For $K = 1, 2, \ldots, 8$ and `covariance_type='full'`, compute the BIC (`model.bic(X)`). Plot BIC vs. K and select the best model. Does BIC recover the correct K?

6. **Bonus.** Repeat the BIC analysis for all four covariance types on the same plot. Which (type, K) pair achieves the lowest BIC overall?

Exercise 3. [DBSCAN] **Useful tools:** `sklearn.cluster.DBSCAN`, `sklearn.neighbors.NearestNeighbors`, `sklearn.metrics.adjusted_rand_score`.

1. Generate the two-moons dataset (`make_moons`, $n = 500$, `noise=0.1`). Run DBSCAN with $\varepsilon = 0.2$ and `min_samples`$= 5$. Plot the assignments, showing noise points (label -1) in a distinct colour. Does DBSCAN succeed where k-means failed?

2. **Sensitivity to ε.** Try $\varepsilon \in \{0.05, 0.1, 0.2, 0.5, 1.0\}$ with `min_samples`$= 5$. For each value, plot the assignments and record the number of clusters found and the percentage of noise points. What happens when ε is too small? Too large?

3. **The k-distance plot.** For each point, compute the distance to its k-th nearest neighbour ($k =$ `min_samples` $- 1 = 4$) using `NearestNeighbors`. Sort these distances and plot them. Identify the "elbow" in the curve. How does the elbow relate to a good choice of ε?

4. **Effect of `min_samples`.** Fix ε at the value suggested by the k-distance plot. Vary `min_samples` in $\{3, 5, 10, 20, 50\}$. How does it affect the number of clusters and the fraction of noise points?

5. **A harder dataset.** Generate two concentric circles using `sklearn.datasets.make_circles` ($n = 500$, `noise=0.05`, `factor=0.5`). Compare k-means ($K = 2$), GMM ($K = 2$), and DBSCAN (choose ε with the k-distance plot). Which method recovers the two circles?

6. **Bonus.** On the moons dataset, compute the Adjusted Rand Index (ARI) for each ε tested in question 2. Plot ARI vs. ε. Is the ARI-optimal ε consistent with the k-distance plot?

Exercise 4. [Hierarchical Clustering] **Useful tools:** `scipy.cluster.hierarchy.linkage`, `scipy.cluster.hierarchy.dendrogram`, `scipy.cluster.hierarchy.fcluster`, `sklearn.cluster.AgglomerativeClustering`.

1. Generate a small dataset with $n = 20$ points and $K = 3$ clusters (`make_blobs`). Compute the linkage matrix with Ward's method (`linkage(X, method='ward')`) and plot the dendrogram. At what height would you cut to obtain 3 clusters? Use `fcluster` with the appropriate threshold and verify.

2. **Comparing linkage criteria.** Generate a larger dataset ($n = 300$, $K = 4$). Compute and display the dendrograms for `'single'`, `'complete'`, `'average'`, and `'ward'` linkage. How do the tree shapes differ? Which linkage produces the most balanced dendrogram?

3. **Chaining effect.** On the two-moons dataset ($n = 300$, `noise=0.1`), run `AgglomerativeClustering` with $K = 2$ for each of the four linkages. Plot the results. Which linkage recovers the moons? Which one suffers from the *chaining effect*, and what does that mean?

4. **Comparison with other methods.** On the blobs dataset ($n = 300$, $K = 4$), compare agglomerative clustering (Ward, $K = 4$), k-means ($K = 4$), GMM ($K = 4$), and DBSCAN (tune ε). Compute the ARI for each. Are the results similar? Which algorithm would you recommend for this data geometry?

5. **Bonus: choosing K from the dendrogram.** On the blobs dataset, plot the dendrogram (Ward) and extract the last 10 merge distances. Plot them as a function of the merge step (a "gap" plot). The largest jump suggests the number of clusters. Does this heuristic recover $K = 4$?

   Hint: the merge distances are stored in the third column of the linkage matrix.

Exercise 5. [Clustering Handwritten Digits] The `load_digits` dataset contains 1797 images of handwritten digits (0–9), each represented as a 64-dimensional vector. The code below loads and visualises the data.

```
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import (adjusted_rand_score,
    normalized_mutual_info_score, silhouette_score)
import matplotlib.pyplot as plt
import numpy as np

digits = load_digits()
X, y_true = digits.data, digits.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2, random_state=42)
X_2d = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_2d[:,0], X_2d[:,1], c=y_true, s=5,
                      cmap='tab10', alpha=0.6)
plt.colorbar(scatter, label='True digit')
plt.title('PCA projection (2D) of digits dataset')
plt.show()
```

1. **k-Means.** Fit k-means with $K = 10$ on `X_scaled` (not `X_2d`). Compute the ARI, NMI, and silhouette score.

2. **GMM.** Fit a Gaussian mixture with 10 components on `X_scaled`. Try the four `covariance_type` options (`'full'`, `'tied'`, `'diag'`, `'spherical'`). Which gives the best ARI?

3. **Dimension reduction before clustering.** Use PCA to reduce `X_scaled` to 20 dimensions, then re-run k-means and GMM. Do the ARI scores improve?

4. **Hierarchical and DBSCAN (bonus).** On the PCA-reduced data (20d), try Ward linkage (cut at 10 clusters) and DBSCAN. Compare ARIs with the previous methods.

Hint: Reducing dimension before clustering removes noise from low-variance directions and helps distance-based methods.

Part IV: Hands-On Python Session Correction

Setup

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons, make_circles
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
np.random.seed(10)
```

Listing 2: Imports and configuration

8 Exercise 1: k-Means on Synthetic Data (15 min)

Exercise 1: Exploring k-Means

Goal: Understand the behaviour of k-means, the effect of initialisation, and the choice of K.

1.1 Generate and visualise data

```python
# Generate 3 well-separated blobs in 2D
X_blobs, y_blobs = make_blobs(n_samples=300, centers=3,
                              cluster_std=1.0, random_state=42)

plt.figure(figsize=(6, 5))
plt.scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_blobs, cmap='viridis',
            s=20, alpha=0.7)
plt.title("Ground truth (3 blobs)")
plt.xlabel("$x_1$"); plt.ylabel("$x_2$")
plt.show()
```

Listing 3: Generate isotropic Gaussian blobs

1.2 Run k-means and visualise

```python
# Run k-means with K=3
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10,
                random_state=42)
labels_km = kmeans.fit_predict(X_blobs)

# Visualise
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].scatter(X_blobs[:, 0], X_blobs[:, 1], c=y_blobs,
                cmap='viridis', s=20)
axes[0].set_title("Ground truth")

axes[1].scatter(X_blobs[:, 0], X_blobs[:, 1], c=labels_km,
                cmap='viridis', s=20)
```

```
axes[1].scatter(kmeans.cluster_centers_[:, 0],
                kmeans.cluster_centers_[:, 1],
                c='red', marker='X', s=200, edgecolors='black')
axes[1].set_title(f"k-Means (K=3), ARI={adjusted_rand_score(y_blobs,
    labels_km):.2f}")
plt.tight_layout(); plt.show()
```

Listing 4: k-Means clustering

1.3 Elbow plot and silhouette

```
Ks = range(2, 10)
inertias = []
silhouettes = []

for K in Ks:
    km = KMeans(n_clusters=K, n_init=10, random_state=42)
    km.fit(X_blobs)
    inertias.append(km.inertia_)
    silhouettes.append(silhouette_score(X_blobs, km.labels_))

fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].plot(Ks, inertias, 'o-'); axes[0].set_xlabel("K")
axes[0].set_ylabel("WCSS (inertia)"); axes[0].set_title("Elbow plot")

axes[1].plot(Ks, silhouettes, 's-', color='orange'); axes[1].set_xlabel(
    "K")
axes[1].set_ylabel("Silhouette score"); axes[1].set_title("Silhouette")
plt.tight_layout(); plt.show()
```

Listing 5: Choosing K

1.4 When k-means fails

```
# Generate non-convex clusters
X_moons, y_moons = make_moons(n_samples=300, noise=0.05,
                              random_state=42)

km_moons = KMeans(n_clusters=2, random_state=42).fit(X_moons)

plt.figure(figsize=(6, 4))
plt.scatter(X_moons[:, 0], X_moons[:, 1], c=km_moons.labels_,
            cmap='viridis', s=20)
plt.title(f"k-Means on moons (ARI={adjusted_rand_score(y_moons, km_moons
    .labels_):.2f})")
plt.show()

# Question: Why does k-means fail here?
```

Listing 6: Non-convex data

9 Exercise 2: Gaussian Mixture Models (15 min)

Exercise 2: EM for Gaussian Mixtures

Goal: Fit a GMM, compare with k-means, visualise covariance ellipses, and use BIC for model selection.

2.1 Generate anisotropic data

```
# Stretch one of the blobs
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = X_blobs.copy()
mask = (y_blobs == 1)
X_aniso[mask] = X_aniso[mask] @ transformation

plt.figure(figsize=(6, 5))
plt.scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_blobs,
            cmap='viridis', s=20)
plt.title("Anisotropic blobs (ground truth)")
plt.show()
```

Listing 7: Anisotropic blobs

2.2 Fit GMM and compare

```
gmm = GaussianMixture(n_components=3, covariance_type='full',
                      random_state=42)
labels_gmm = gmm.fit_predict(X_aniso)

km_aniso = KMeans(n_clusters=3, random_state=42).fit_predict(X_aniso)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].scatter(X_aniso[:, 0], X_aniso[:, 1], c=km_aniso,
                cmap='viridis', s=20)
axes[0].set_title(f"k-Means, ARI={adjusted_rand_score(y_blobs, km_aniso)
    :.2f}")

axes[1].scatter(X_aniso[:, 0], X_aniso[:, 1], c=labels_gmm,
                cmap='viridis', s=20)
axes[1].set_title(f"GMM, ARI={adjusted_rand_score(y_blobs, labels_gmm)
    :.2f}")
plt.tight_layout(); plt.show()
```

Listing 8: GMM vs. k-Means on anisotropic data

2.3 BIC for model selection

```
bics = []
Ks = range(1, 8)
for K in Ks:
    g = GaussianMixture(n_components=K, covariance_type='full',
                        random_state=42)
    g.fit(X_aniso)
    bics.append(g.bic(X_aniso))

```

```
plt.figure(figsize=(6, 4))
plt.plot(Ks, bics, 'o-')
plt.xlabel("Number of components K")
plt.ylabel("BIC")
plt.title("BIC for GMM model selection")
plt.show()
print(f"Optimal K by BIC: {list(Ks)[np.argmin(bics)]}")
```

Listing 9: BIC curve

10 Exercise 3: DBSCAN (15 min)

Exercise 3: DBSCAN on Non-Convex Clusters

Goal: Apply DBSCAN where k-means fails, explore the effect of ε and `minPts`, and identify noise.

3.1 DBSCAN on moons

```
db = DBSCAN(eps=0.2, min_samples=5)
labels_db = db.fit_predict(X_moons)

n_clusters = len(set(labels_db) - {-1})
n_noise = (labels_db == -1).sum()

plt.figure(figsize=(6, 4))
plt.scatter(X_moons[:, 0], X_moons[:, 1], c=labels_db, cmap='viridis',
            s=20)
plt.title(f"DBSCAN: {n_clusters} clusters, {n_noise} noise points, "
          f"ARI={adjusted_rand_score(y_moons, labels_db):.2f}")
plt.show()
```

Listing 10: DBSCAN on two moons

3.2 Sensitivity to ε

```
fig, axes = plt.subplots(1, 4, figsize=(20, 4))
for ax, eps in zip(axes, [0.05, 0.1, 0.2, 0.5]):
    db = DBSCAN(eps=eps, min_samples=5)
    labels = db.fit_predict(X_moons)
    n_cl = len(set(labels) - {-1})
    ax.scatter(X_moons[:, 0], X_moons[:, 1], c=labels, cmap='viridis',
               s=15)
    ax.set_title(f"eps={eps}, {n_cl} clusters")
plt.tight_layout(); plt.show()
```

Listing 11: Varying ε

3.3 The k-distance plot

```
from sklearn.neighbors import NearestNeighbors

k = 5  # minPts
nn = NearestNeighbors(n_neighbors=k)
```

```
nn.fit(X_moons)
distances, _ = nn.kneighbors(X_moons)
k_distances = np.sort(distances[:, -1])

plt.figure(figsize=(6, 4))
plt.plot(k_distances)
plt.xlabel("Points (sorted)")
plt.ylabel(f"Distance to {k}-th nearest neighbour")
plt.title("k-distance graph (knee $\\approx$ good $\\varepsilon$)")
plt.axhline(y=0.2, color='r', linestyle='--', label='$\\varepsilon=0.2$')
plt.legend(); plt.show()
```

Listing 12: Choosing ε via k-distance graph

Remark 10.1 (HDBSCAN)**.** A modern extension, **HDBSCAN** [12], eliminates the sensitive ε parameter by running DBSCAN across all ε values and extracting a hierarchy. It then selects the most *stable* clusters from this hierarchy automatically. In practice, HDBSCAN requires essentially only `min_samples` and tends to be significantly more robust than DBSCAN. It is available via the `hdbscan` method (in scikit-learn version ≥ 1.3).

11 Exercise 4: Hierarchical Clustering (15 min)

Exercise 4: Hierarchical Clustering and Dendrograms

Goal: Build dendrograms with different linkage criteria and compare the resulting partitions.

4.1 Dendrograms with different linkages

```
# Use the blob data for clearer dendrograms (subsample for visibility)
idx = np.random.choice(len(X_blobs), 50, replace=False)
X_sub = X_blobs[idx]
y_sub = y_blobs[idx]

fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, method in zip(axes, ['single', 'complete', 'ward']):
    Z = linkage(X_sub, method=method)
    dendrogram(Z, ax=ax, truncate_mode='lastp', p=20,
               leaf_rotation=90, leaf_font_size=8)
    ax.set_title(f"Linkage: {method}")
    ax.set_ylabel("Distance")
plt.tight_layout(); plt.show()
```

Listing 13: Computing and plotting dendrograms

4.2 Flat clustering from hierarchical

```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, link in zip(axes, ['single', 'complete', 'ward']):
    agg = AgglomerativeClustering(n_clusters=3, linkage=link)
    labels_agg = agg.fit_predict(X_blobs)
    ari = adjusted_rand_score(y_blobs, labels_agg)
    ax.scatter(X_blobs[:, 0], X_blobs[:, 1], c=labels_agg,
```

```
                cmap='viridis', s=20)
    ax.set_title(f"{link} linkage, ARI={ari:.2f}")
plt.tight_layout(); plt.show()
```

Listing 14: Agglomerative clustering with scikit-learn

4.3 Single linkage on moons

```
agg_moons = AgglomerativeClustering(n_clusters=2, linkage='single')
labels_agg_moons = agg_moons.fit_predict(X_moons)
ari = adjusted_rand_score(y_moons, labels_agg_moons)

plt.figure(figsize=(6, 4))
plt.scatter(X_moons[:, 0], X_moons[:, 1], c=labels_agg_moons,
            cmap='viridis', s=20)
plt.title(f"Single-linkage on moons, ARI={ari:.2f}")
plt.show()

# Question: Why does single linkage work here?
# What happens with complete or Ward linkage?
```

Listing 15: Single linkage captures non-convex shapes

Exercice 5 : Clustering in High Dimensions

We generate data from a Gaussian mixture model where clusters *partially overlap*, and study how each algorithm behaves as the dimension d grows.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import adjusted_rand_score
from sklearn.neighbors import NearestNeighbors
from sklearn.datasets import make_classification

def generate_gmm_data(n_samples, d, K, separation=1.0, seed=42):
    """
    Generate a K-cluster dataset in dimension d using
    sklearn's make_classification.

    Parameters
    ----------
    n_samples : int, total number of points
    d : int, ambient dimension
    K : int, number of clusters
    separation : float, controls cluster separation
                 (larger = easier)
    seed : int

    Returns
    -------
    X : ndarray (n_samples, d)
    y : ndarray (n_samples,)
    """
    X, y = make_classification(
```

```
        n_samples=n_samples,
        n_features=d,
        n_informative=min(d, max(K, 5)),
        n_redundant=0,
        n_clusters_per_class=1,
        n_classes=K,
        class_sep=separation,
        flip_y=0.0,
        random_state=seed
    )
    return X, y
```

```
# ----- Experiment: ARI vs dimension for all algorithms -----
dims = [2, 5, 10, 20, 50, 100, 200, 500]
K = 4
n_samples = 4 * 100  # 100 per cluster on average
separation = 1.0     # Try also 0.5 (hard) or 2.0 (easy)

results = {name: [] for name in
           ['KMeans', 'GMM', 'Agglom. (Ward)', 'DBSCAN']}

for d in dims:
    X, y_true = generate_gmm_data(n_samples, d, K,
                                  separation=separation, seed=42)

    # k-means
    km = KMeans(n_clusters=K, n_init=10, random_state=0).fit(X)
    results['KMeans'].append(
        adjusted_rand_score(y_true, km.labels_))

    # GMM (EM)
    cov_type = 'diag' if d >= 50 else 'full'
    gmm = GaussianMixture(n_components=K,
                          covariance_type=cov_type,
                          n_init=5, random_state=0).fit(X)
    results['GMM'].append(
        adjusted_rand_score(y_true, gmm.predict(X)))

    # Agglomerative (Ward)
    agg = AgglomerativeClustering(
        n_clusters=K, linkage='ward').fit(X)
    results['Agglom. (Ward)'].append(
        adjusted_rand_score(y_true, agg.labels_))

    # DBSCAN -- adaptive eps using k-distance heuristic
    nn = NearestNeighbors(n_neighbors=10).fit(X)
    dists, _ = nn.kneighbors(X)
    eps = np.percentile(dists[:, -1], 90)
    db = DBSCAN(eps=eps, min_samples=5).fit(X)
    results['DBSCAN'].append(
        adjusted_rand_score(y_true, db.labels_))

# ----- Plot -----
plt.figure(figsize=(9, 5))
markers = ['o-', 's--', 'D-.', 'v:']
for (name, ari_list), marker in zip(results.items(), markers):
    plt.plot(dims, ari_list, marker, label=name,
             linewidth=2, markersize=6)
```

```
47
48 plt.xlabel('Dimension $d$', fontsize=12)
49 plt.ylabel('Adjusted Rand Index', fontsize=12)
50 plt.title(f'Clustering performance vs. dimension '
51           f'($K={K}$, $n={n_samples}$, '
52           f'class\\_sep$={separation}$)', fontsize=13)
53 plt.xscale('log')
54 plt.ylim(-0.05, 1.05)
55 plt.legend(fontsize=11)
56 plt.grid(True, alpha=0.3)
57 plt.tight_layout()
58 plt.show()
59
60 # ----- Print summary table -----
61 print(f"{'Dim':>5}", end="")
62 for name in results:
63     print(f"  {name:>15}", end="")
64 print()
65 print("-" * (5 + 17 * len(results)))
66 for i, d in enumerate(dims):
67     print(f"{d:>5}", end="")
68     for name in results:
69         print(f"  {results[name][i]:>15.3f}", end="")
70     print()
```

Questions:

1. Run the experiment with `separation=2.0`. Which algorithm degrades fastest as d increases? Which is most robust? Explain.

2. Now try different values for `separation=1.5`. How do the relative rankings change?

3. Why do we switch to `covariance_type='diag'` for the GMM when $d \geq 50$? What would happen with `'full'`? (Hint: how many parameters does a full covariance matrix have?)

4. DBSCAN requires careful tuning of ε. Here we used an adaptive heuristic. Does it work well? Is DBSCAN a natural choice for this generative model? Why or why not?

5. **Bonus:** Add a PCA step that projects X to $K-1=3$ dimensions before clustering. Does this help? For which algorithms? (*from sklearn.decomposition import PCA*)

12 Exercise 6: Clustering on Real Data

Exercise 5: Clustering on Real Data — Handwritten Digits

We now apply the tools from this lecture to a real, higher-dimensional dataset.

```
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import adjusted_rand_score,
    normalized_mutual_info_score, silhouette_score
import matplotlib.pyplot as plt
import numpy as np

# Load data: 1797 images of 8x8 pixels -> d=64
digits = load_digits()
X, y_true = digits.data, digits.target
print(f"Shape: {X.shape}, Classes: {np.unique(y_true)}")

# Step 1: Standardize and reduce dimension for visualization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2, random_state=42)
X_2d = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_2d[:,0], X_2d[:,1], c=y_true, s=5,
                      cmap='tab10', alpha=0.6)
plt.colorbar(scatter, label='True digit')
plt.title('PCA projection (2D) of digits dataset')
plt.show()
```

1. Try out some clustering algorithms on the digit data set (with and without the dimnesion reduction step), and compare the results.

References

[1] J. Kleinberg, *An impossibility theorem for clustering*, Advances in Neural Information Processing Systems (NeurIPS), 2002.

[2] D. Arthur and S. Vassilvitskii, *k-means++: The advantages of careful seeding*, Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1027–1035, 2007.

[3] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Proceedings of KDD, pp. 226–231, 1996.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm*, Journal of the Royal Statistical Society, Series B, 39(1):1–38, 1977.

[5] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, *NP-hardness of Euclidean sum-of-squares clustering*, Machine Learning, 75(2):245–248, 2009.

[6] S. Dasgupta, *The hardness of k-means clustering*, Technical Report CS2008-0916, UC San Diego, 2008.

[7] M. Ackerman and S. Ben-David, *Clusterability: A theoretical study*, Proceedings of AISTATS, pp. 1–8, 2009.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd edition, Springer, 2009.

[10] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022.

[11] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, 1990.

[12] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-Based Clustering Based on Hierarchical Density Estimates," *Proc. PAKDD*, pp. 160–172, 2013.