

K-means Clustering

Aayush Bajaj

July 10, 2026

Contents

1 the objective	1
2 lloyd's algorithm	2
2.1 convergence	2
3 k-means++ initialisation	3
4 choosing k	3
5 failure modes	4
6 relation to gaussian mixtures	4
7 from scratch	4
8 see also	5

k-means is unsupervised learning's hello world: pick k prototype points, assign every datum to its nearest prototype, move each prototype to the centre of its flock, repeat. it is fast, it always terminates, and it is wrong in ways that are so instructive that every clustering course starts here anyway.

lloyd wrote it up at bell labs in 1957 for pulse-code modulation — it stayed an internal memo until 1982

1 the objective

data $x_1, \dots, x_n \in \mathbb{R}^d$. choose k centroids $\mu_1, \dots, \mu_k$ and an assignment $z_i \in \{1, \dots, k\}$ for each point, to minimise the **within-cluster sum of squares** (wcss, also “distortion”):

$$J(z, \mu) = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2. \quad (1)$$

two half-problems hide inside, each trivial with the other half frozen:

- **fix μ , optimise z** : assign each point to its nearest centroid — independent per point.
- **fix z , optimise μ** : for cluster $C_j = \{i : z_i = j\}$, minimise $\sum_{i \in C_j} \|x_i - \mu_j\|^2$. setting the gradient $-2 \sum_{i \in C_j} (x_i - \mu_j)$ to zero gives $\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i$ — the **mean**. hence the name.

jointly, though, the problem is np-hard — even for $k = 2$, even in the plane for general k . what we actually run is alternating minimisation on the two half-problems, which is lloyd’s algorithm (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009).

so every k-means run you have ever done was a heuristic. it usually doesn't matter. sometimes it does.

2 lloyd’s algorithm

- **input:** data $x_{1:n}$, cluster count k .
- **initialise:** pick starting centroids $\mu_1, \dots, \mu_k$ (see k-means++ below; naive choice is k random data points).
- **assignment step:** $z_i \leftarrow \arg \min_j \|x_i - \mu_j\|^2$ for every i .
- **update step:** $\mu_j \leftarrow \frac{1}{|C_j|} \sum_{i \in C_j} x_i$ for every j (if a cluster empties, re-seed its centroid, e.g. at the point farthest from all centroids).
- **repeat** the two steps until assignments stop changing.

each iteration costs $O(nkd)$ — one distance from every point to every centroid — which is why k-means scales to datasets that make fancier methods weep.

2.1 convergence

lloyd’s algorithm terminates in finitely many iterations, at a local optimum. the argument is two sentences:

- **monotone descent.** the assignment step minimises J over z with μ fixed; the update step minimises J over μ with z fixed (the mean is the unique minimiser of summed squared distances). so J never increases, and it is bounded below by 0.
- **finiteness.** there are at most k^n assignments. given an assignment, the optimal centroids are determined (the means), so the algorithm’s state is really just z . since J strictly decreases whenever z changes, no assignment can recur, and the algorithm halts in at most k^n steps — in practice, tens.

ties broken consistently, e.g. keep your current cluster on equal distance

what it converges *to* is only a fixed point of the two steps: no guarantee of the global minimum, and bad initialisations genuinely produce bad fixed points (two centroids sharing one blob while another blob goes unserved). standard practice: multiple random restarts, keep the run with lowest J .

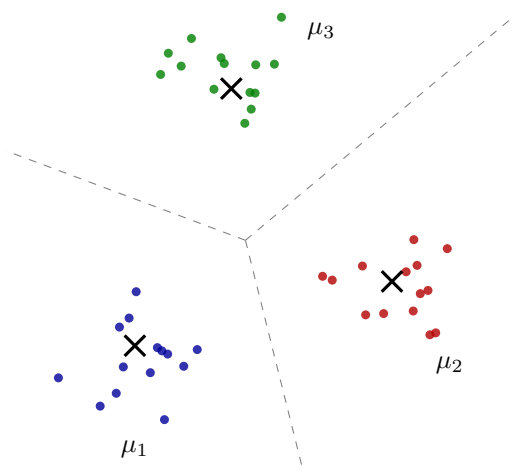


Figure 1: a lloyd fixed point: points coloured by assignment, crosses mark centroids, dashed lines are the voronoi boundaries of the centroids — every point sits on its own centroid’s side.

3 k-means++ initialisation

random initialisation fails in a predictable way: two seeds land in the same dense blob. k-means++ spreads seeds out *probabilistically*:

- **first centre:** pick a data point uniformly at random.
- **subsequent centres:** pick data point x with probability proportional to $D(x)^2$, where $D(x)$ is the distance from x to the nearest centre chosen so far.
- **then** run lloyd as usual.

squaring the distance biases hard towards points far from every existing seed — likely members of unclaimed clusters — while still giving outliers only proportional (not certain) weight. the guarantee, stated honestly:¹ immediately after seeding (before a single lloyd iteration), the expected cost satisfies

$$\mathbb{E}[J] \leq 8(\ln k + 2) J_{\text{opt}}, \quad (2)$$

an $O(\log k)$ -competitive bound *in expectation over the seeding randomness* — not a guarantee for any single run, and lloyd afterwards only improves it. in practice k-means++ both speeds convergence (our run below converges in three iterations) and largely de-risks the restarts ritual.

4 choosing k

J decreases monotonically in k (more centroids never hurt the objective — at $k = n$ it is zero), so you cannot pick k by minimising J . standard instruments:

- **elbow:** plot $J(k)$ and pick the kink where the marginal gain collapses. cheap, subjective, and real data frequently declines to supply an elbow.
- **silhouette:** for each point let a_i be its mean distance to its own cluster and b_i its mean distance to the nearest other cluster; score $s(i) = \frac{b_i - a_i}{\max(a_i, b_i)} \in [-1, 1]$ and maximise the average over k . interpretable point-by-point (negative $s(i)$ flags misassignments), but the pairwise distances cost $O(n^2)$.
- **gap statistic:** compare $\log J(k)$ against its expectation under a structureless reference distribution (uniform over the data’s bounding box), estimated by monte carlo; choose the smallest k whose gap beats the next one’s within a standard error, $\text{Gap}(k) \geq \text{Gap}(k + 1) - s_{k+1}$. the most principled of the three — it has

¹arthur & vassilvitskii (2007), *k-means++: the advantages of careful seeding*, soda 2007. the $8(\ln k + 2)$ factor is their theorem 1.1.

references

Deisenroth, Marc Peter and Faisal, A. Aldo and Ong, Cheng Soon (2020). *Mathematics for Machine Learning*, Cambridge University Press. Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome (2009). *The Elements of Statistical Learning*, Springer.

a null model, so it can honestly answer $k = 1$, “there are no clusters here” — and the most expensive (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009).

all three are advisory. if the downstream use is vector quantisation or indexing, k is a budget, not a truth to be discovered.

5 failure modes

the wcss objective hard-codes assumptions; violate them and k-means confidently returns nonsense:

- **anisotropy.** euclidean distance to a single centroid models clusters as round. elongated or tilted clusters get sliced across their long axis, because the voronoi boundary is always a straight perpendicular bisector. whitening the data first (pca) helps only if all clusters share the same elongation.
- **unequal sizes and densities.** a broad sparse cluster next to a tight dense one loses its periphery to the tight cluster’s centroid — wcss prefers balanced distortion, not correct membership.
- **non-convex shapes.** concentric rings, half-moons: no assignment of points to nearest-centroid regions can trace them, since voronoi cells are convex. kernelise the distances or switch to spectral clustering.
- **it never says no.** run k-means on uniform noise and it returns k tidy clusters with a straight face. clustering structure must be tested (gap statistic), never assumed from the output.

6 relation to gaussian mixtures

k-means is the hard-assignment limit of em on a gaussian mixture model. take a gmm with equal weights and spherical covariances $\Sigma_j = \epsilon I$, and let $\epsilon \rightarrow 0$: the responsibility of component j for point x_i ,

$$r_{ij} = \frac{\exp(-\|x_i - \mu_j\|^2/2\epsilon)}{\sum_l \exp(-\|x_i - \mu_l\|^2/2\epsilon)} \longrightarrow \mathbf{1}\{j = \arg \min_l \|x_i - \mu_l\|^2\}, \quad (3)$$

collapses onto the nearest centroid (the softmax freezes into a max), the e-step becomes lloyd’s assignment step, and the m-step’s weighted means become plain means (Deisenroth, Marc Peter and Faisal, A. Aldo and Ong, Cheng Soon, 2020). read backwards: gmms are k-means with soft memberships, learned covariances (fixing anisotropy) and learned weights (fixing unequal sizes) — at the price of many more parameters per cluster.

7 from scratch

lloyd’s algorithm with k-means++ seeding, on three gaussian blobs:

```
1 import numpy as np
2
3 rng = np.random.default_rng(3)
4
5 # three blobs in 2d, 100 points each
```

```

6 centres_true = np.array([[0.0, 0.0], [4.0, 1.0], [1.5, 4.0]])
7 X = np.vstack([c + rng.normal(0, 0.7, size=(100, 2)) for c in centres_true])
8
9 def wcss(X, mu, z):
10     return sum(np.sum((X[z == j] - mu[j])**2) for j in range(len(mu)))
11
12 def kmeans_pp_init(X, k, rng):
13     mu = [X[rng.integers(len(X))]] # first centre uniform
14     for _ in range(k - 1):
15         d2 = np.min([np.sum((X - m)**2, axis=1) for m in mu], axis=0)
16         mu.append(X[rng.choice(len(X), p=d2 / d2.sum())])
17     return np.array(mu)
18
19 def lloyd(X, k, rng, init="pp", verbose=True):
20     mu = kmeans_pp_init(X, k, rng) if init == "pp" \
21         else X[rng.choice(len(X), k, replace=False)]
22     for it in range(100):
23         # assignment step: nearest centroid
24         z = np.argmin((X[:, None, :] - mu[None])**2).sum(-1), axis=1)
25         # update step: mean of each cluster
26         mu_new = np.array([X[z == j].mean(axis=0) if np.any(z == j) else mu[j]
27                             for j in range(k)])
28         if verbose:
29             print(f"iter {it}: wcss = {wcss(X, mu_new, z):.2f}")
30         if np.allclose(mu_new, mu):
31             break
32         mu = mu_new
33     return mu, z
34
35 mu, z = lloyd(X, 3, rng)
36 print("final centroids:")
37 print(np.round(mu[np.argsort(mu[:, 0])], 3))
38 print("true centres:")
39 print(centres_true[np.argsort(centres_true[:, 0])])
40 print("cluster sizes:", np.bincount(z))

```

```

iter 0: wcss = 283.72
iter 1: wcss = 280.65
iter 2: wcss = 280.65
final centroids:
[[0.015 0.068]
 [1.499 4.225]
 [4.011 1.016]]
true centres:
[[0.  0. ]
 [1.5 4. ]
 [4.  1. ]]
cluster sizes: [101 100  99]

```

three iterations to convergence — k-means++ seeding starts so close to the answer that lloyd barely has anything left to do. the recovered centroids match the generating centres to within sampling noise, and the two boundary points that hopped blobs during generation (101/100/99 rather than 100/100/100) are assigned to whichever centroid they actually sit nearest, which is the most k-means can ever promise.

8 see also

- [gaussian mixture models](#) — k-means with soft assignments and covariances
- [principal component analysis](#) — decorrelate/whiten before clustering, or com-

press after

- **k-nearest neighbours** — the other famous k , supervised and unrelated
- **curse of dimensionality** — in high dimensions all distances concentrate, and “nearest centroid” quietly loses meaning