PROF. DR.-ING. MARK SCHUTERA

_SUPERVISED LEARNING

UNFINISHED LECTURE NOTES

PUBLISHED BY UNFINISHED LECTURE NOTES

These notes are, by their very nature, unfinished, and they improve with every reader. If you spot an error, disagree with a framing, or want to add a source, a question, or a position, open a pull request at https://github.com/Quillstacks/LectureMaterial/tree/main/lecturenotes/notes_unsupervisedlearning. Every contribution is welcome.

2026-05-06 · cheerful mango Haubentaucher

"IN MY DEFENSE, I WAS LEFT UNSUPERVISED".

DER ZAUBERLEHRLING

Contents

Introduction

Supervised learning is limited to tasks where we have labeled data and clear objectives. Beyond this limited context of learning, we fall back to the pattern in the data itself.

We explore how pattern emerges, from other sources of information beyond explicit labels. We will see how pattern can be discovered in data, represented, leveraged and utilized. In that sense this course is about any form of learning that is not supervised learning.

Each lecture follows a consistent pattern. To start of the contents of the lecture will be motivated by a narrative, a theoretical primer, and some sort of hands-on experience to build intuition. Then we will dive into the, theory and algorithms, then reflection. Theory sessions are 45 minutes, followed by 45 minutes of coding practice to enforce reflection and deepen understanding of the concepts.

A Philosophy of Education

The beautiful risk of education is that it is an encounter. I am trying to model a way of thinking, and reasoning, then stepping back. The rest is yours.

- **Start with why.** A method understood only procedurally is a method you will misapply when it matters. Before learning how something works, understand why it was built, what it replaced, and where it fails. The mechanics then follow from necessity, not memorization.
- **Build taste alongside competence.** You need a sense of what beautiful is, you need to train your aesthetics before you can reliably produce it. This develops by studying what others have built, making it your own, then making your own, and honestly evaluating the gap between them. Taste is the compass. Skill is the engine. You need both, in that order.
- **Failure is the designed medium.** Each attempt teaches what a clean run cannot. The insight compounds across tries, not from a single correct execution. Those who never start failing will fail.
- **Work in the open.** Showing half-finished thinking, naming mistakes as they happen, is a professional skill. We practice it here. It is how knowledge actually moves between people. And it is how knowledge moves towards you, at unprecedented speed.
- **The fallacy of mediocracy.** Iteration and failure are the method, not the excuse for settling. Do not confuse velocity with quality. Whatever you learn, whatever you build: make it good.
- **Outcomes emerge. They are not delivered.** Judgment, taste, the capacity to sit with uncertainty: these are not course outputs. They are what you build from the encounter. I can create the conditions. The rest depends on what you do with them.

Part I

Geometry of Pattern

Similarity and Distance

2026-05-06 · cheerful mango Haubentaucher

Enter the Feature Space

The Why

In supervised learning[1], the learning process relies on data samples x paired with their ground truth labels $\tilde{y}$, which allows a model θ to learn a mapping,

$$\begin{aligned}\theta(x) &= \hat{y}, \\ &\approx \tilde{y}.\end{aligned}$$

The presence of labels provides a clear signal for learning, as it allows optimization of a loss function that measures the discrepancy between predictions and true labels. The loss then guides the learning process, enabling the model to adjust its parameters along the gradient to minimize this discrepancy:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\hat{y}, \tilde{y}).$$

Facing the absence of labels, we are left with only the data samples[2] x and no explicit signal to guide learning. The methods we will explore in this course are designed to discover and leverage the inherent structure and patterns[3] in the data itself, without relying on external supervision, thus mostly referred to as unsupervised learning.

In a first step this requires us to,

- define what we mean by pattern in the context of data.
- understand the geometry of data, which includes concepts of distance, similarity, and neighborhood.

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. https://scholar.google.com/scholar?q=Goodfellow+Bengio+Courville+2016+deep+learning+book

On the concept of truth, $\tilde{y}$, while being called ground truth, is often an approximation of the true label y. It may come from human annotation, measurements, or other sources.

[2] J. Rowley. The wisdom hierarchy: Representations of the dikw hierarchy. *Journal of Information and Communication Science*, 33(2):163–180, 2007. DOI: 10.1177/0165551506070706. https://scholar.google.com/scholar?q=Rowley+2007+wisdom+hierarchy+DIKW

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. ISBN 978-0387310732. https://scholar.google.com/scholar?q=Bishop+2006+pattern+recognition+machine+learning

Knowledge Pyramid
Data (1) raw signal
→ Information (*banana*) endowed with meaning
→ Knowledge ($\neq$ *coffee*) structured and contextualized information
→ Wisdom or *knowing and appreciating the Why*.

- learn how to engineer data representations that make pattern more accessible.

THE FUNDAMENTAL QUESTION in unsupervised learning is to find pattern, represent it, and utilize it.

Hands On Experience

CONSIDER six data points in two dimensions. Which points seem to belong together? Why do you think that? How many groups do you see? To which group would you assign new points? Draw areas around each group.

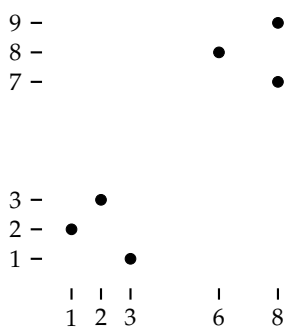


Figure 1: Six points in 2D. Can you see the two clusters? Or are there six? Or maybe one?

WITHOUT LABELS, pattern are ambiguous. With just a single additional point, the pattern can change drastically. Where does the new point belong? Does it belong to the first cluster, the second, or does it form its own cluster? Does this new point change the pattern of the original six points? How would you draw areas around the clusters now? How did that change your assumptions of the first pattern?

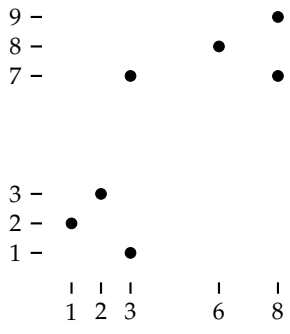


Figure 2: Seven points in 2D. The point at $(3,7)$ creates ambiguity. Is it part of the first cluster, the second, or its own cluster?

HUMANS ARE REMARABLY GOOD AT FINDING PATTERN IN DATA, even with very little information, and even if it is completely off. It

PAREIDOLY is the tendency to perceive meaningful patterns in random data. Such as animals in clouds.

is a fundamental part of our cognition, and it is what allows us to make sense of the world. However, it also motivates the need for methods to quantify and formalize pattern. On that note it is also good practice to start with a hypothesis of what you expect to find in the data, and then test that hypothesis with the data, rather than just looking for any pattern that may emerge. And on a short managerial note, if the knowledge you will gain will not make you take action, then don't waste your time measuring it in the first place.

The Learning Objectives of this lecture are that you will be able to:

- Understand the key role of distance and similarity in unsupervised learning and pattern recognition.
- Compute and interpret common distance metrics (Euclidean, Manhattan, Cosine, Mahalanobis).
- Apply appropriate normalization and scaling strategies and understand why they are necessary.

Distance and Similarity

QUITE INTUITIVELY, you already connected points and formed groups in terms of a similarity which was based on the closeness between data points. This concept will carry you very far.

A DISTANCE FUNCTION $d : X \times X \to \mathbb{R}_{\geq 0}$, which means taking two samples out of the set X, mapping them to a non-negative real number, must satisfy:

Non-negativity:	$d(x, y) \geq 0$	(1)
Identity:	$d(x, y) = 0 \iff x = y$	(2)
Symmetry:	$d(x, y) = d(y, x)$	(3)
Triangle inequality:	$d(x, z) \leq d(x, y) + d(y, z)$	(4)

EUCLIDEAN DISTANCE is probably the most familiar distance metric, measuring the straight-line distance between two points in space.

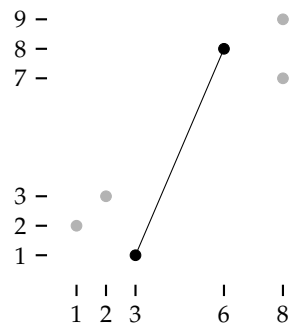


Figure 3: Euclidean distance between points (3,1) and (6,8).

And it was probably the one you implicitly used to connect the points in the previous exercise. It is defined as:

$$d_{\text{Euclidean}}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} = \|x - y\|_2$$

Calculating the Euclidean distance between points $A = (3, 1)$ and $B = (6, 8)$:

$$\begin{aligned} d(A, B) &= \sqrt{(6-3)^2 + (8-1)^2} \\ &= \sqrt{9 + 49} \\ &= \sqrt{58} \qquad \approx 7.62 \end{aligned}$$

EUCLIDEAN DISTANCE assumes all dimensions are equally important and measured in the same units. It also assumes a continuous, smooth space. It is trivial to think of scenarios where these assumptions do not hold, such as when features have different scales (e.g. age vs. income) or when the features are not linearly related or not equaly important (e.g. size vs. IQ). And then there are the cases where the units are ambiguous for the task like distances on maps.

FEATURES can be thought of as dimensions in a vector space. Each data point is a vector in this space.

MANHATTAN DISTANCE is also called L_1 distance or taxicab distance. Instead of measuring the euclidean distance, it measures the axis-aligned path along grid lines:

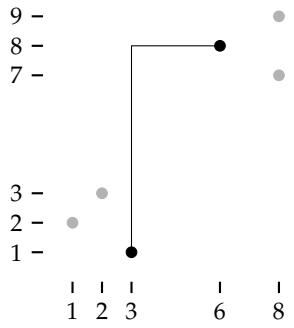


Figure 4: Manhattan (taxicab) distance between points (3,1) and (6,8): path follows grid lines.

$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^{n} |x_i - y_i| = \|x - y\|_1$$

Figure 5: Grid, 1811 (public domain) see Bridgeman Art Library v. Corel Corp.

ESPECIALLY USEFUL when movement is constrained to axes, such as for city blocks like the ones in Manhattan, pixel grids, or when you want robustness to outliers in individual dimensions.

Calculating the Manhattan distance between points $A = (3,1)$ and $B = (6,8)$:

$$\begin{aligned} d_M(A, B) &= |6-3| + |8-1| \\ &= 3 + 7 \\ &= 10 \end{aligned}$$

COSINE SIMILARITY measures the angle between two vectors, ignoring their magnitude. It is especially useful when the correlation of the features of the data matters more than its absolute values.

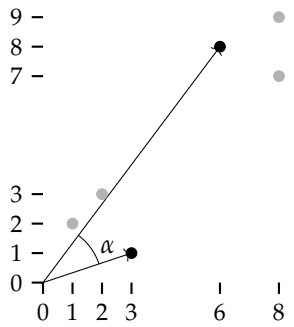


Figure 6: Cosine similarity between points (3,1) and (6,8): measuring the angle α between the position vectors.

When magnitude in the feature space does not matter only direction we use cosine similarity. It is defined as:

Calculating the cosine similarity between points $A = (3,1)$ and $B = (6,8)$:

$$\begin{aligned} \text{sim}(A, B) &= \frac{3 \cdot 6 + 1 \cdot 8}{\sqrt{3^2+1^2}\sqrt{6^2+8^2}} \\ &= \frac{26}{\sqrt{10} \cdot 10} \\ &\approx 0.82 \end{aligned}$$

$$d_{\text{cosine}}(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}$$
$$= 1 - \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}}$$

The right-hand term is the cosine similarity, which ranges from -1 (opposite) to $+1$ (identical direction), so the distance ranges from 0 to 2.

THINK of a vector multiplication as a projection of one vector onto another.

MAGNITUDE IS IRRELEVANT for example, in text analysis, documents can have vectors of very different lengths as some documents have more words than others, but cosine similarity will still focuse on how many specific words per total word count. Similarly, in recommendation systems, users might have preference vectors with different scales, as some users rate more items, some less, but cosine similarity will compare the pattern of preferences.

MAHALANOBIS DISTANCE generalizes Euclidean distance by accounting for different variances and correlations among features. It measures distance in terms of the data's own covariance structure. In other words, it transforms the space so that distances are measured in units of standard deviation along principal axes, giving ellipsoidal rather than spherical neighborhoods. The covariance matrix Σ captures the spread and correlation of the data, and is defined as:

$$\Sigma = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_i)(x_i - \mu_i)^T$$

where μ is the mean vector of the data.

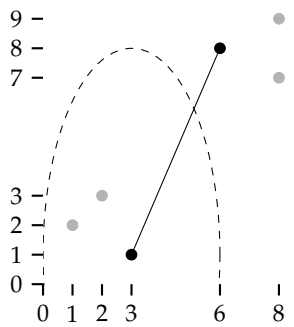


Figure 7: Mahalanobis distance between points (3,1) and (6,8): the dashed ellipse shows equidistant points under covariance Σ, replacing the Euclidean circle with an ellipse that reflects the data's spread.

Euclidean distance treats all dimensions equally, but real data often has different variances per dimension or correlations between

dimensions. The Mahalanobis distance compensates by weighting each dimension according to the data's spread. The inverse covariance matrix Σ^{-1} is used to weight the distance calculation, giving more weight to dimensions with less variance and less weight to dimensions with more variance.

$$d_{\text{Mahalanobis}}(x, y) = \sqrt{(x-y)^T \Sigma^{-1} (x-y)}$$

where Σ is the covariance matrix of the data.

IT TRANSFORMS THE SPACE so that distances are measured in units of standard deviation along principal axes, giving ellipsoidal rather than spherical neighborhoods.

Calculating the Mahalanobis distance between $A = (3, 1)$ and $B = (6, 8)$ with $\Sigma = \left(\begin{smallmatrix} 9 & 0 \\ 0 & 49 \end{smallmatrix}\right)$:

$$\begin{aligned} d_M(A, B) &= \sqrt{\frac{3^2}{9} + \frac{7^2}{49}} \\ &= \sqrt{1+1} \\ &= \sqrt{2} \\ &\approx 1.41 \end{aligned}$$

DATA PREPROCESSING, before computing distances, ensures that features are comparable. You have already seen, how the Mahalanobis distance incorporates the covariance structure of the data to account for different scales and correlations. With the Euclidean distance, if features are on different scales, the distance will be dominated by the larger-scaled feature. If one feature's measure has a different magnitude than the other, the larger-scaled feature will dominate the distance calculation. Preprocessing brings all features to a common ground by transforming the feature space into a standardized scale.

Z-SCORE NORMALIZATION transforms each feature to have mean 0 and standard deviation 1. This centers the data at the origin and scales each dimension by its own spread:

NORMALIZATION typically refers to scaling features to a specific range, such as $[0, 1]$. While standardization refers to centering features to have mean 0 and scaling to have standard deviation 1.

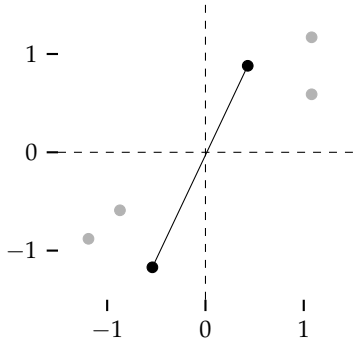


Figure 8: Our six points after z-score normalization: centered at the origin with unit variance in each dimension.

The z-score of each feature is computed as:

$$z_i = \frac{x_i - \mu_i}{\sigma_i} \tag{5}$$

where μ_i and σ_i are the mean and standard deviation of feature i.

Normalizing point $A = (3, 1)$:

$$\begin{aligned} \mu &= (4.67,\ 5.00) \\ \sigma &= (3.08,\ 3.41) \\ z_x &= \frac{3 - 4.67}{3.08} \\ &= -0.54 \\ z_y &= \frac{1 - 5.00}{3.41} \\ &= -1.17 \\ A' &= (-0.54,\ -1.17) \end{aligned}$$

USE WHEN features have different units or scales (e.g., age in years, vs. body height in milimiters). However, this is sensitive to outliers, as they influence μ and σ.

MIN-MAX SCALING transforms features to a fixed range $[0, 1]$. Each feature's minimum maps to 0 and its maximum to 1:

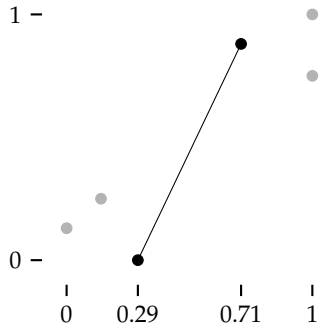


Figure 9: Our six points after min-max scaling: all values lie in $[0, 1]$.

$$x'_i = \frac{x_i - \min_i}{\max_i - \min_i} \tag{6}$$

Scaling point $A = (3, 1)$ with $x \in [1, 8]$, $y \in [1, 9]$:

$$\begin{aligned} x' &= \frac{3-1}{8-1} \\ &= \frac{2}{7} \\ &\approx 0.29 \\ y' &= \frac{1-1}{9-1} \\ &= 0 \\ A' &= (0.29,\ 0) \end{aligned}$$

USE WHEN you need bounded values. Unlike z-score, min-max guarantees a fixed range but is even more sensitive to outliers since a single extreme value stretches the scale and compresses other ranges.

CHOOSING A DISTANCE METRIC encodes your assumptions about what makes two data points similar in a given context. The same pair of points can appear close or far apart depending on the metric. The choice should be driven by the structure of your data, the context you are in, and the question you are answering.

A PRACTICAL RULE OF THUMB: try Euclidean first as a baseline, then ask whether the assumptions it makes, equal scales, independent features, magnitude matters, actually hold for your data. Each violation points toward a more suitable metric.

Distance between $A{=}(3, 1)$ and $B{=}(6, 8)$ under each metric:

$$\begin{aligned} d_{\text{Euclidean}} &= 7.62 \\ d_{\text{Manhattan}} &= 10 \\ d_{\text{Cosine}} &= 0.18 \\ d_{\text{Mahalanobis}} &= 1.41 \end{aligned}$$

Same points, four different answers. The distance defines the similarity.

Examples & Exercises

Exercises are for practice and reinforcing concepts. Try to solve them on your own first, try things, play with it, discuss, this is not a time trial. And there is no shame in not ending up at the right answer, in the same sense, that uncovering great questions and tossing them around is usually pretty fruitful on the long run.

$A = (2, 5, 1)$
$B = (6, 1, 3)$
$C = (4, 5, 2)$

Computing by hand builds the intuition that no amount of library calls can replace. Step through the arithmetic slowly.

Given points A, B, and C, compute the Euclidean distance $d_E(A, B)$, the Manhattan distance $d_M(A, C)$, and the cosine similarity $\text{sim}(A, B)$ with its corresponding cosine distance. If you feel you need more practice, go for it.

Euclidean distance sums squared coordinate differences:

$$\begin{aligned} d_E(A, B) &= \sqrt{(6-2)^2 + (1-5)^2 + (3-1)^2} \\ &= \sqrt{16 + 16 + 4} \\ &= \sqrt{36} \\ &= 6 \end{aligned}$$

Manhattan distance sums absolute differences instead, no squaring, no root:

$$\begin{aligned} d_M(A, C) &= |2-4| + |5-5| + |1-2| \\ &= 2 + 0 + 1 \\ &= 3 \end{aligned}$$

Cosine similarity measures the angle between the two vectors, independent of their length:

$$\begin{aligned} \text{sim}(A, B) &= \frac{A \cdot B}{\|A\| \, \|B\|} \\ &= \frac{2{\cdot}6 + 5{\cdot}1 + 1{\cdot}3}{\sqrt{4+25+1} \, \sqrt{36+1+9}} \\ &= \frac{20}{\sqrt{30} \, \sqrt{46}} \\ &\approx 0.54 \end{aligned}$$

The cosine distance follows as:

$$\begin{aligned} d_{\cos} &= 1 - 0.54 \\ &= 0.46 \end{aligned}$$

The metric is a modelling choice. There is no formula that tells you which distance to use, it follows from understanding your data and the context you are in. In the following scenarios, decide which distance metric is most appropriate and justify your choice. First

reason geometrically, then commit to an answer. Zoom-in so you can not see the margin notes, and try to solve it on your own first. Then read the margin notes for the answer.

A NEWS PLATFORM represents 10 000 articles as word-frequency vectors. Some articles are 100 words long, others 5 000 words. You want to find articles that cover similar topics.

COSINE SIMILARITY. Article length inflates the magnitude of the frequency vector but does not change its direction. Cosine similarity ignores magnitude, comparing only the distribution of words.

A DELIVERY COMPANY plans routes through Manhattan's grid. Pickup locations are given as (street, avenue) coordinates.

MANHATTAN DISTANCE. Movement on a grid is constrained to horizontal and vertical steps. The shortest driving path between two intersections follows the L_1 norm, not the straight line.

A COFFEE QUALITY LAB measures sourness (0 to 200) and bitterness (0 to 500) for each batch.

MAHALANOBIS DISTANCE. The features live on different scales and are correlated. Mahalanobis accounts for both by using the covariance matrix, producing ellipsoidal contours that align with the data's spread.

GIVEN THREE COFFEE BATCHES measured by sourness and bitterness:

$$P_1 = (80,\ 400)$$
$$P_2 = (200,\ 160)$$
$$P_3 = (110,\ 220)$$

compute the sample covariance matrix and use it to find the Mahalanobis distance between P_1 and P_3.

The mean vector:

$$\mu_x = 130$$
$$\mu_y = 260$$
$$\mu = (130,\ 260)$$

Deviations from the mean:

$$P_1 - \mu = (-50,\ \ 140)$$
$$P_2 - \mu = (\ \ 70,\ -100)$$
$$P_3 - \mu = (-20,\ -40)$$

The sample covariance matrix sums all three outer products:

$$\begin{aligned}\Sigma &= \frac{1}{N-1}\sum (x_i - \mu)(x_i - \mu)^T \\ &= \frac{1}{2}\begin{pmatrix} 7800 & -13200 \\ -13200 & 31200 \end{pmatrix} \\ &= \begin{pmatrix} 3900 & -6600 \\ -6600 & 15600 \end{pmatrix}\end{aligned}$$

We invert Σ analytically using the 2×2 formula:

$$\begin{aligned}\det(\Sigma) &= 3900 \cdot 15600 - 6600^2 \\ &= 17{,}280{,}000\end{aligned}$$

$$\Sigma^{-1} = \frac{1}{17{,}280{,}000}\begin{pmatrix}15600 & 6600\\ 6600 & 3900\end{pmatrix}$$

For a 2×2 matrix:

$$A = \begin{pmatrix}a & b\\ c & d\end{pmatrix}$$

$$A^{-1} = \frac{1}{ad-bc}\begin{pmatrix}d & -b\\ -c & a\end{pmatrix}$$

Swap the diagonal, negate the off-diagonal, divide by the determinant.

The difference vector is $(x-y) = P_3 - P_1 = (30,\ -180)$. Apply Σ^{-1}:

$$\Sigma^{-1}(x-y) = \frac{1}{17{,}280{,}000}\begin{pmatrix}15600 & 6600\\ 6600 & 3900\end{pmatrix}\begin{pmatrix}30\\ -180\end{pmatrix}$$

$$= \frac{1}{17{,}280{,}000}\begin{pmatrix}-720{,}000\\ -504{,}000\end{pmatrix}$$

Intuitively, A^{-1} undoes the transformation A: if A stretches and rotates space, A^{-1} maps everything back, with $AA^{-1} = I$. For Σ, the inverse rescales each dimension by its variance and removes correlations, so distances are measured in standard-deviation units.

Taking the dot product with $(x-y)$ gives the squared distance:

$$\begin{aligned}d_M^2 &= (x-y)^T(\Sigma^{-1}(x-y)) \\ &= \frac{1}{17{,}280{,}000}(30,\ -180)\begin{pmatrix}-720{,}000\\ -504{,}000\end{pmatrix} \\ &= \frac{69{,}120{,}000}{17{,}280{,}000} \\ &= 4 \\ d_M &= \sqrt{4} \\ &= 2\end{aligned}$$

The parentheses in $(x-y)^T(\Sigma^{-1}(x-y))$ highlight that $\Sigma^{-1}(x-y)$ is computed as one step. In this exercise we form Σ^{-1} explicitly because the matrix is 2×2. In higher dimensions, computing the full inverse is expensive and amplifies rounding errors. Instead, you solve $\Sigma z = (x-y)$ for z, which gives the same result as $\Sigma^{-1}(x-y)$ without ever forming the inverse — just as you can compute 5/3 by dividing directly rather than first finding 1/3 and then multiplying by 5.

Compare with the Euclidean distance, which does not account for the different scales and correlation. The raw displacement of 30 sourness units and 180 bitterness units yields an overall distance of 182.5, making P_3 seem very far from P_1, driven almost entirely by the bitterness difference:

$$\begin{aligned}d_E(P_1, P_3) &= \sqrt{(110-80)^2 + (220-400)^2} \\ &= \sqrt{900 + 32400} \\ &= \sqrt{33300} \\ &\approx 182.5\end{aligned}$$

The Mahalanobis distance is far smaller because the covariance matrix rescales for the large variance difference between features. The bitterness axis has a much larger spread (σ^2=15600 vs σ^2=3900), so a displacement of 180 bitterness units is less unusual than it appears, and contributes less to the Mahalanobis distance accordingly.

STANDARDIZATION does not change the data, it changes the expanse and spread of the space you measure it in. Units disappear; and make way for a more abstract notion of distance, in terms of how many standard deviations apart the data points are.

$C_1 = (300,\ 7)$
$C_2 = (500,\ 6)$
$C_3 = (700,\ 3)$

daily cups sold, customer rating (1 to 10)

THE EFFECT OF STANDARDIZATION on café data. Three cafés C_1, C_2, C_3 recorded as daily cups sold and customer rating.

Compute the Euclidean distance on raw features, standardize all three points, and recompute. On raw features, daily sales dominate completely; the two pairs involving C_2 look equally close:

$$\begin{aligned} d_E(C_1, C_2) &= \sqrt{(500-300)^2 + (6-7)^2} \\ &= \sqrt{40\,000 + 1} \\ &\approx 200.0 \\ d_E(C_2, C_3) &= \sqrt{(700-500)^2 + (3-6)^2} \\ &= \sqrt{40\,000 + 9} \\ &\approx 200.0 \end{aligned}$$

The rating differences (1 and 9) are drowned out by the cup counts (40,000 each). The means and standard deviations are:

$$\begin{aligned} \mu_{\text{cups}} &= 500, & \sigma_{\text{cups}} &= 200 \\ \mu_{\text{rating}} &\approx 5.33, & \sigma_{\text{rating}} &\approx 2.08 \end{aligned}$$

After z-score normalization the cafés map to:

$$C_1' \approx (-1,\ 0.80), \qquad C_2' \approx (0,\ 0.32), \qquad C_3' \approx (1,\ -1.12)$$

THE CUPS standardize cleanly because $300, 500, 700$ are evenly spaced. The ratings do not: $7, 6, 3$ give a non-integer mean and an irrational standard deviation. This is what real data looks like.

Recomputing the distances:

$$\begin{aligned} d(C_1', C_2') &= \sqrt{(0-(-1))^2 + (0.32-0.80)^2} \\ &= \sqrt{1 + 0.23} \\ &\approx 1.11 \\ d(C_2', C_3') &= \sqrt{(1-0)^2 + (-1.12-0.32)^2} \\ &= \sqrt{1 + 2.07} \\ &\approx 1.75 \end{aligned}$$

C_2 is now clearly closer to C_1 than to C_3. In raw space, sales made both pairs look identical; after standardization, the similar reviews of C_1 and C_2 pull them together while C_3's low rating pushes it away.

A TOY DATASET TO EXPLORE. You are given measurements of 20 coffee samples with three features: *acidity* (pH, 4.0 to 7.0), *bitterness*

CODE will be provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

20 coffee samples
acidity (pH, 4.0 to 7.0)
bitterness (1 to 10)
brew strength (mg/mL, 5 to 25)

four varieties: cold brew, drip, espresso, latte

(1 to 10), and *brew strength* (mg/mL, 5 to 25). The samples belong to four varieties: cold brew, drip, espresso, and latte. Load the dataset and scatter-plot acidity vs. bitterness. Then compute the Euclidean distance matrix and identify the most similar pair, is the result plausible? Apply z-score normalization, recompute the distance matrix, and observe how the nearest neighbors shift. Compute cosine similarity between all pairs and compare it to the Euclidean grouping. Finally, compute the sample covariance matrix of the normalized data and check whether any features are correlated.

WHAT TO OBSERVE. Without normalization, brew strength dominates similarity because its scale (5 to 25 mg/mL) dwarfs the others. A latte and an espresso with similar strength appear closer than two espressos with the same acidity and bitterness profile. After normalization, all three features contribute equally and the four natural clusters become visible. Cosine similarity groups samples by the *ratio* of their features, which may merge varieties with similar flavor profiles but very different intensities.

Self-Reflection and Recap

SELF-REFLECTION Questions which can guide your thoughts during the exercises and afterwards:

- How do we distinguish between supervised and unsupervised learning?
- Elaborate on the differences between $\tilde{y}$, $\hat{y}$, and y?
- What are the key properties that define a distance metric?
- How do different distance metrics capture different notions of similarity?
- What is a feature space, and how do data points relate to it?
- What do we mean by feature engineering and feature transformation?
- What can happen when features are on different scales or magnitudes?
- How can we engineer and transform feature spaces?
- Why do we transform feature spaces with respect to pattern? And what are the common transformations?

RECAP of Key Concepts:

- Unsupervised learning discovers pattern in unlabeled data
- Distance metrics define what similar means
- Feature spaces can be engineered and transformed to reveal different patterns

SO FAR, WE DID MEASURE SIMILARITY BETWEEN SINGLE DATA POINTS. How do we embed this information across data points? Is it possible to represent pattern in the data as a whole, rather than just pairwise similarities? In the next chapter we explore clusters and clustering algorithms, which are the most classic way to represent pattern in unsupervised learning.

WITHOUT KNOWING ABOUT LABELS OR SEMANTICS we can tell which samples are similar.

TEASER. How do we represent structure beyond pairwise similarities and distances?

FEEDBACK

Centroid Clustering

2026-05-06 · cheerful mango Haubentaucher

Same Same but Different.

The Why

In the previous chapter, we established that distance is a choice that is to be engineered[4]: it encodes what similar means in a given feature space. But similarity is a local, pairwise concept so far.

As a human looking at data, we immediately see clusters C of points that belong together, and we have an intuition for which new points would fit into which cluster. The process of finding them is called clustering.

A partition or a clustering assigns every point to exactly one cluster, with no cluster left empty. In this lecture we will learn about K-Means[5], which produces a *hard* partition: membership is binary, not graded.

Given that means to encode similarity over whole sets of samples, we can ask a more complex question, one that goes beyond pairs of points. Being able to answer which point belongs to which cluster is a predictive power that allows us to generalize from observed data to new instances.

In order to move from representation to leveraging pattern we have good reason to find ways to,

- compute partitions, of the data, given a distance measure.
- formalize what makes a well-defined partition by specifying an appropriate objective.
- recognize the geometric assumptions and finding ways to represent cluster characteristics.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. ISBN 978-0387310732. https://scholar.google.com/scholar?q=Bishop+2006+pattern+recognition+machine+learning

[5] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967. https://scholar.google.com/scholar?q=MacQueen+1967+methods+classification+multivariate+observations

Binary membership is a strong assumption that later clustering methods will soften.

Figure 10: Clustering allows us to move from a) and b) are similar to a) and b) are of the same. *Image is generated with NanoBanana.*

Hands On Experience

CONSIDER twelve data points in two dimensions. Which points belong together? How many clusters do you see?

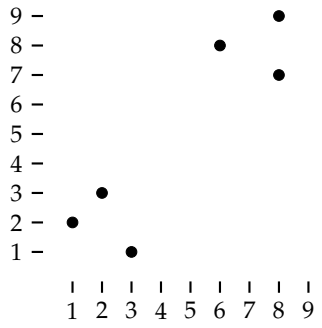


Figure 11: Six points in 2D. How many clusters do you see?

HOW MANY CLUSTERS do you see? Two seems natural, but could you argue for one? For six?

WITHOUT GROUND TRUTH, there is no single right answer, as a context for any answer could be engineered. However, some answers are better than others, and it does not come as a surprise when I tell you that most people see two clusters in this data.

THE FUNDAMENTAL QUESTION this chapter answers is how do we find the clusters mathematically? We will see how K-Means assigns each point to a cluster; and how cluster centroids function as representative points for each cluster.

WHERE WOULD YOU PLACE a representative point for each cluster? Could you think of other measures to characterize and represent clusters, beyond a single point?

A CENTROID c is the mean position of all points assigned to a cluster. It need not be an actual data point; it is the center of mass of the cluster. The centroid exists in the same space as the data. With a different distance, a different notion of center is needed. For sake of simplicity and interpretability, we tie ourselves to Euclidean distance and arithmetic means. But it is worth getting an intuition for other distances as well.

THE LEARNING OBJECTIVES of this lecture:

- Understand the K-Means objective and algorithm, including convergence.
- Apply initialization strategies and K-selection methods.
- Identify when K-Means fails and understand why.

The K-Means Algorithm

A DATA POINT is a vector $x_i \in \mathbb{R}^d$, where d is the number of features and $i = 1, \ldots, n$ indexes the observations.

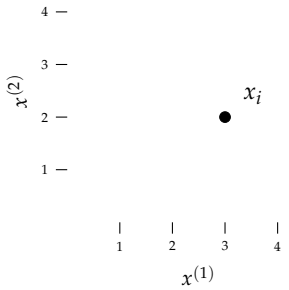


Figure 12: A single data point x_i in $\mathbb{R}^2$.

A CLUSTER is a non-empty subset $C_k \subseteq \{x_1, \ldots, x_n\}$. Each point belongs to exactly one cluster; this is the hard assignment assumption, meaning clusters are disjoint:

$$C_j \cap C_k = \emptyset \quad \text{for } j \neq k$$

A clustering of the data is a collection of K clusters $\{C_1, \ldots, C_K\}$ that is exhaustive, every point is accounted for:

$$\bigcup_{k=1}^{K} C_k = \{x_1, \ldots, x_n\}$$

K-MEANS finds a clustering of n data points into K clusters by minimizing the total squared distance from each point to its cluster's centroid. This quantity is called inertia or within-cluster sum of squares (WCSS):

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \tag{7}$$

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

MINIMIZING J is NP-hard (effort exponential in n and K) in general. K-Means finds a local optimum, not necessarily the global one. The result depends on initialization, and is thus non-deterministic. Shout out numerical methods.

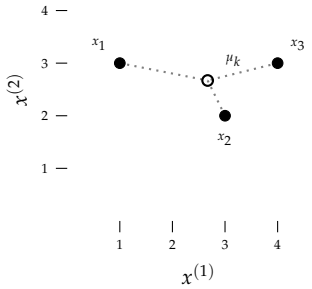


Figure 13: Cluster $C_k = \{x_1, x_2, x_3\}$ with centroid μ_k. Dotted lines show the distances whose squares sum to J.

THE ALGORITHM ALTERNATES BETWEEN TWO STEPS until the assignments no longer change. It remains one of the most cited algorithms in machine learning, in part because it is easy to implement, easy to interpret, and fast enough to run on large datasets.

Algorithm 1: K-Means

Require: Data $X = \{x_1, \ldots, x_n\}$, number of clusters K, initial centroids $\mu_1, \ldots, \mu_K$
Ensure: Clustering $\{C_1, \ldots, C_K\}$, centroids $\{\mu_1, \ldots, \mu_K\}$

1: **repeat**
2: **Assignment:** $c_i \leftarrow \arg\min_k \|x_i - \mu_k\|^2$ for each x_i
3: **Update:** $\mu_k \leftarrow \frac{1}{|C_k|} \sum_{x_i : c_i = k} x_i$ for each k
4: **until** assignments no longer change

STANDARD ALGORITHMIC FORM was described by Lloyd

S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. DOI: 10.1109/TIT.1982.1056489. https://scholar.google.com/scholar?q=Lloyd+1982+least+squares+quantization+PCM

A TRACE THROUGH THE ALGORITHM on three points with K=2 makes this concrete. With initial centroids at μ_1=x_1=$(1,3)$ and μ_2=x_3=$(4,3)$:

- For each point, the assignment step asks: which centroid is closest?
- For each cluster, the update step asks: given the current assignments, where is the new centroid?

STEP 1: INITIALIZATION. We place two centroids at $\mu_1 = (1,4)$ and $\mu_2 = (4,1)$, away from any data point. At this moment the positions of the centroids is arbitrary, and the mean variable is not meaningful.

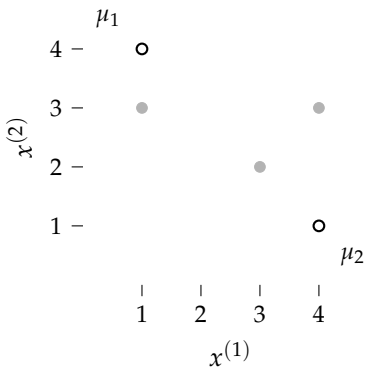


Figure 14: Initialization: two centroids placed away from the data. All points are unassigned.

STEP 2: ASSIGNMENT. Each point joins the cluster of its nearest centroid. For that, we calculate squared distances.

$$
\begin{aligned}
d^2(x_1,\mu_1) &= 0+1 = 1\\
d^2(x_1,\mu_2) &= 9+4\\
&= 13 \;\rightarrow\; C_1\\
d^2(x_2,\mu_1) &= 4+4 = 8\\
d^2(x_2,\mu_2) &= 1+1\\
&= 2 \;\rightarrow\; C_2\\
d^2(x_3,\mu_1) &= 9+1 = 10\\
d^2(x_3,\mu_2) &= 0+4\\
&= 4 \;\rightarrow\; C_2
\end{aligned}
$$

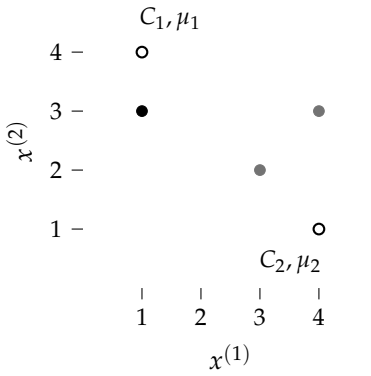


Figure 15: Assignment: $x_1 \rightarrow C_1$ (dark), x_2 and $x_3 \rightarrow C_2$ (grey).

STEP 3: UPDATE. Each centroid moves to the mean of its cluster.

$$
\begin{aligned}
\mu_1' &= x_1\\
&= (1,\ 3)\\
\mu_2' &= \tfrac{1}{2}\big((3,2)+(4,3)\big)\\
&= (3.5,\ 2.5)
\end{aligned}
$$

ON ′ NOTATION. The prime symbol (′) usually denotes updates in iterative algorithms.

CONVERGENCE is guaranteed in finite steps. The argument rests on three observations: first, each assignment step can only decrease

LOCAL OPTIMUM. Convergence does not mean the algorithm found the best clustering. It found a clustering. A different initialization may yield a better one.

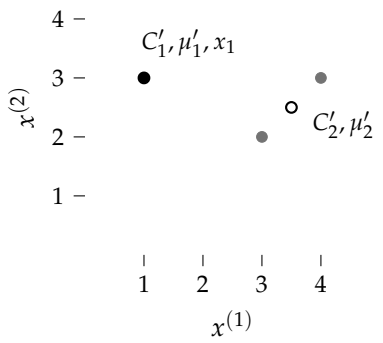


Figure 16: Update: both centroids move. μ_1 coincides with $x_1 = (1, 3)$; μ_2 jumps from $(4, 1)$ to $(3.5, 2.5)$. Re-running assignment yields the same clusters: converged at J=1.

J, since every point moves to an equal or closer centroid; second, each update step can only decrease J, since the mean minimizes squared distance to its members; third, there are at most K^n possible assignment configurations. Since J is non-increasing and the number of states is finite, the algorithm must reach a fixed assignment and stop.

Initialization Matters

K-Means is sensitive to where centroids start. The algorithm is thus non-deterministic. This is because the objective is non-convex: it has many local minima, and the algorithm may get stuck in one of them. If two initial centroids fall inside the same natural cluster, this cluster might get split. While at the same time, a different cluster might be left without a centroid, getting merged with a nearby cluster instead, which results in a high-inertia J clustering.

Running the same three points with μ_1=$(2, 2)$ and μ_2=$(4, 4)$ shows how a different start leads to a different, worse result.

Multiple restarts. Running K-Means n times with different random initializations, keep the result with the lowest J. Typical default: n=10, but checking the variance or standard deviation of the resulting Inertias and making an informed decision is recommended.

Educated InitializationK-Means++ for instance features educated initialization. Centroids seeded onto data points x_i. The first centroid is chosen uniformly at random. Each subsequent centroid is chosen with probability proportional to its squared distance to the nearest already-chosen centroid: $p(x_i) = D(x_i)^2 / \sum_j D(x_j)^2$, where $D(x_i) = \min_{j<k} \|x_i - c_j\|$. Points far from all current centroids are more likely to be chosen next. This gives a theoretical guarantee on the expected inertia relative to the global optimum.

D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007. https://scholar.google.com/scholar?q=Arthur+Vassilvitskii+2007+k-means+advantages+careful+seeding

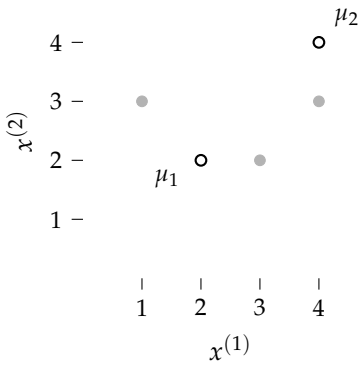


Figure 17: Initialization: centroids at μ_1=(2, 2) and μ_2=(4, 4). All points unassigned.

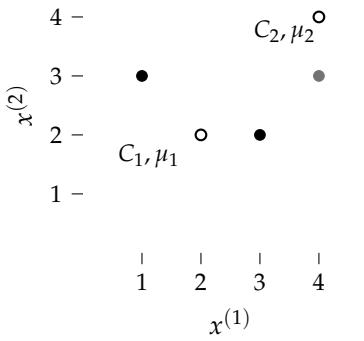


Figure 18: Assignment: x_1 and $x_2 \rightarrow C_1$ (dark), $x_3 \rightarrow C_2$ (grey).

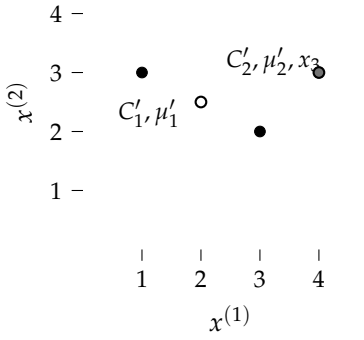


Figure 19: Update: μ_1 moves to (2, 2.5), μ_2 drops to (4, 3). Re-running assignment yields the same clusters: converged at J=2.5, worse than the earlier trace (J=1).

Choosing K

In 2D the number of clusters is often obvious, but in higher dimensions and with more data, or when facing fuzzy clusters or cluster borders, it is not. While the choice for K is often guided by domain knowledge, it is not uncommon to have no such guidance. Then we have to rely on heuristics.

The elbow method plots inertia J against K and looks for a kink: the point where adding one more cluster yields diminishing returns. For our three points, J drops from K=1 to K=2, then reaches zero at K=3.

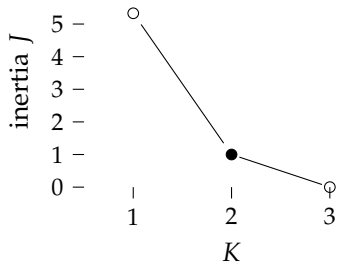


Figure 20: Inertia J as a function of K for the three trace points. The kink at K=2 (filled dot) matches the two natural clusters. K=3 trivially gives J=0 - jap, that is overfitting.

The elbow is not always clear. On real data, the curve may decrease smoothly with no obvious kink. The elbow is a heuristic, not a guarantee.

The silhouette score builds on two distances defined for each point x_i. This allows us to evaluate the clustering on a per-point scale. Think about how interpretable and beautiful that would look when colour-coded.

- **Intra-cluster distance** $a(i)$: mean distance from x_i to all other points in its own cluster. Small $a(i)$ means the point fits tightly with its neighbours.
- **Inter-cluster distance** $b(i)$: mean distance from x_i to all points in the nearest other cluster. Large $b(i)$ means the point is far from the next-best alternative.

A well-placed point has small $a(i)$ and large $b(i)$. The silhouette score combines both into a single value per point:

$$s(i) = \frac{b(i) - a(i)}{\max\big(a(i),\, b(i)\big)}$$

$s(i)$ ranges from -1; inter $<$ intra: point fits better in a neighbouring cluster. To $+1$; intra $\ll$ inter: point is tightly placed and well-separated The mean silhouette over all points is a quality score for the full clustering. Choose K that maximizes it.

Simplifications in per-cluster metrics. In practice, $a(i)$ is often computed as the distance to the cluster centroid, rather than the mean distance to all other points. And b(i) is often computed as the distance to the nearest other centroid, rather than the mean distance to all points in the nearest cluster. And even more simplifications are possible, when only calculating the silhouette for the centroids themselves.

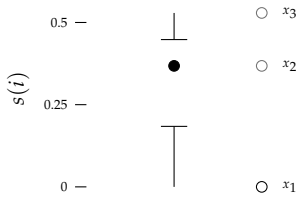
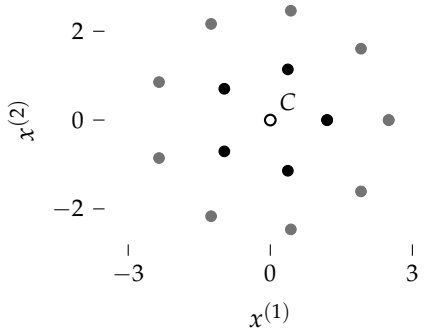


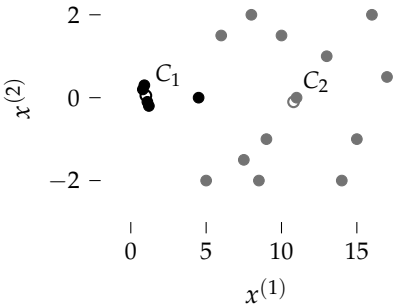
Figure 21: Silhouette scores for K=2. Box plot (left): median ≈0.37, mean ≈0.30, IQR [0.18, 0.45]. Individual scores per point (right, coloured by cluster).

When K-Means Fails

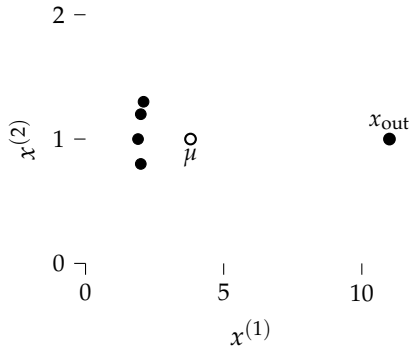
K-Means assumes clusters are roughly spherical, equally sized, and equally dense. Remember that distances measures are a choice.



Non-Convex Shapes
Figure 22: The natural clusters are concentric rings. Both share the same centroid at the origin, so K-Means cannot separate them; it will split the data along a straight boundary through the centre instead.



Cluster Density and Gravity
Figure 23: C_1 has four tightly packed points near $x = 1$; C_2 has thirteen points spread from $x = 5$ to $x = 17$, pulling its centroid to $x \approx 10.8$. The decision boundary falls near $x \approx 6$, so the point at $x = 4.5$ is assigned to C_1 even though it sits at C_2's left edge—a direct consequence of K-Means optimising inertia rather than visual proximity.



Outliers
Figure 24: Sensitivity to outliers: four tight points form a natural cluster near $(2, 1)$, but a single outlier ($\times$) pulls the centroid μ to $(3.8, 1)$, well outside the dense group. K-Means uses the mean, which is not robust to extreme points.

When K-Means struggles, the underlying issue is always the same: the Euclidean distance to a centroid is the wrong measure of belonging for that data. Can you think of another failure mode? Let me know.

K-Medoids uses actual data points as cluster centers instead of means, making it robust to outliers and compatible with non-Euclidean distances.

Mini-Batch K-Means updates centroids on random subsets of the data only, allowing us to scale to millions of points at the cost of a slightly noisier result.

Examples & Exercises

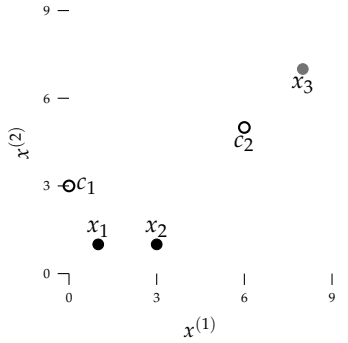


Figure 25: Three coffee samples x_1, x_2, x_3 with initial centroids $\mu_1{=}(0,3)$ and $\mu_2{=}(6,5)$ (circles).

COMPUTING BY HAND builds the intuition that no amount of function calls can replace. Step through the arithmetic slowly. Implementing from scratch comes right after, but let's anchor the mechanics first.

GIVEN THREE POINTS $x_1{=}(1,1)$, $x_2{=}(3,1)$, $x_3{=}(8,7)$, run one full K-Means iteration with $K{=}2$. The initial centroids are $c_1{=}(0,3)$ and $c_2{=}(6,5)$, neither coinciding with a data point. Your tasks: perform the assignment step, the update step, verify convergence, and evaluate the quality of the clustering by computing the inertia.

ASSIGNMENT STEP. For each point compute $d^2(x, c_k) = \sum_j (x_j - c_{kj})^2$ and assign to the nearest centroid. Here is the full trace for point $x_1{=}(1,1)$:

$$\begin{aligned} d^2(x_1,\, c_1) &= (1-0)^2 + (1-3)^2 \\ &= 1 + 4 \\ &= 5 \end{aligned}$$

$$\begin{aligned} d^2(x_1,\, c_2) &= (1-6)^2 + (1-5)^2 \\ &= 25 + 16 \\ &= 41 \end{aligned}$$

Since $5 < 41$, point x_1 is assigned to C_1. Now compute the distances for x_2 and x_3 yourself, then verify against the table below.

Point	$d^2(\cdot, \mu_1)$	$d^2(\cdot, \mu_2)$	**Cluster**
x_1 $(1,1)$	**5**	41	C_1
x_2 $(3,1)$	**13**	25	C_1
x_3 $(8,7)$	80	**8**	C_2

Table 1: Assignment step with $c_1{=}(0,3)$, $c_2{=}(6,5)$. Bold values indicate the minimum in each row; each point is assigned to the corresponding cluster.

UPDATE STEP. Recompute each centroid as the mean of its assigned points. Here is the full calculation for μ_1':

$$\begin{aligned}\mu_1' &= \frac{1}{|C_1|}\sum_{x\in C_1} x \\ &= \tfrac{1}{2}\big((1,1)+(3,1)\big) \\ &= \tfrac{1}{2}(4,2) \\ &= (2,1)\end{aligned}$$

Now derive μ_2' yourself (C_2 contains only a single point) and then check:

$$\begin{aligned}\mu_2' &= x_3 \\ &= (8,7)\end{aligned}$$

CONVERGENCE CHECK. Run the assignment step once more with μ_1'=$(2,1)$ and μ_2'=$(8,7)$: you should recover the same two clusters C_1=$\{x_1, x_2\}$, C_2=$\{x_3\}$. The algorithm has converged after a single iteration.

The inertia at convergence is:

$$\begin{aligned}J &= \|x_1-\mu_1'\|^2 + \|x_2-\mu_1'\|^2 + \|x_3-\mu_2'\|^2 \\ &= \big((1{-}2)^2+(1{-}1)^2\big) + \big((3{-}2)^2+(1{-}1)^2\big) + 0 \\ &= 1+1+0 \\ &= 2\end{aligned}$$

Calculate the inertia for $K=1$ yourself. Assume that the single centroid is at the mean of all three points, and is thus quick to compute. What is the Inertia for $K=3$, is there a need to compute it, and why?

CHOOSING K WITH THE ELBOW METHOD. The table in the margin gives the optimal inertia for $K=1,2,3$ on the same three points. Sketch the inertia curve and identify the elbow: the value of K at which adding another cluster stops yielding a large reduction. Which K do you select, and why?
Note that K=3 always reaches J=0 by placing one centroid per point; this tells us nothing about structure in the data.

Inertia values:

K	J
1	50
2	2
3	0

COMPUTE THE SILHOUETTE SCORES for the three trace points $x_1{=}(1,3)$, $x_2{=}(3,2)$, $x_3{=}(4,3)$ with $K{=}2$, where $C_1{=}\{x_1\}$ and $C_2{=}\{x_2, x_3\}$. Recall that $a(i)$ is the mean intra-cluster distance and $b(i)$ the mean nearest-cluster distance for point x_i.

For x_1 (singleton cluster C_1): a single point has no intra-cluster structure to evaluate, so by convention:

$$s(x_1) = 0$$

For a singleton, $a(x_i)$ is the mean distance to zero other points, i.e. $0/0$. Setting $a{=}0$ by convention makes the formula return $s{=}1$, as if the point were perfectly clustered. The override $s{=}0$ corrects this: without at least two points in a cluster, the silhouette score is undefined and should not inflate the mean. $s{=}0$ is the neutral midpoint of $[-1,1]$ and carries no bias.

For x_2 and x_3 (both in C_2, with $d(x_2, x_3) = \sqrt{2} \approx 1.41$):

$$\begin{aligned} a(x_2, C_2) &= d(x_2, x_3) \approx 1.41 \\ b(x_2, C_1) &= d(x_2, x_1) \\ &= \sqrt{5} \\ &\approx 2.24 \\ s(x_2) &= \frac{2.24 - 1.41}{2.24} \approx 0.37 \end{aligned}$$

$$\begin{aligned} a(x_3, C_2) &= d(x_3, x_2) \approx 1.41 \\ b(x_3, C_1) &= d(x_3, x_1) \\ &= \sqrt{9} \\ &= 3 \\ s(x_3) &= \frac{3 - 1.41}{3} \approx 0.53 \end{aligned}$$

The mean silhouette score is $\bar{s} = (0 + 0.37 + 0.53)/3 \approx 0.30$, matching the box plot in the theory section. Note that x_1 drags the mean down: singleton clusters always score $s{=}0$. A score well below 1 indicates that points are not much closer to their own cluster than to the nearest rivaling cluster.

CODE will be provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

A COFFEE AROMA DATASET TO EXPLORE. Load the dataset of 1,797 two-dimensional aroma fingerprints from ten brewing methods and scatter-plot the two aroma coordinates, coloring each sample by its true variety. Implement the assignment step, the update step, and the inertia function from scratch, then combine them into a complete K-Means run. Run K-Means with $K{=}10$ twenty times using different random seeds and compare the resulting inertia values.

Then vary K from 1 to 15, record the best inertia across three restarts for each, and plot the elbow curve.

RANDOM SEEDS. Random seeds are integers that initialize the pseudo-random number generator, ensuring reproducibility.

WHAT TO OBSERVE. Inertia varies noticeably across the twenty runs at $K{=}10$: some seeds place two starting centroids inside the same brewing group and the algorithm never recovers, producing a measurably worse partition. The elbow curve bends near $K{=}10$, though

not sharply, because several brewing varieties overlap in the two-dimensional fingerprint space. Adding clusters beyond the elbow yields diminishing inertia reduction, confirming that the algorithm is splitting existing groups rather than discovering new ones.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What does the K-Means objective actually minimize, and why is that a reasonable definition of a good clustering?
- Why does K-Means always converge, and why does convergence not guarantee the best solution?
- Which two improvements of K-means did we mention around initialization?
- How can we mitigate the effect of outliers on K-Means?
- When would you use the elbow method? When would you not?
- How does the silhouette score measure something different from inertia?
- What are the three most notorious failures of K-Means, and as a teaser how can we mitigate those?
- Given a new dataset, what would be your first three steps before running K-Means?

RECAP of Key Concepts:

- K-Means minimizes inertia by alternating assignment (points to nearest centroid) and update (centroid to cluster mean)
- Multiple random initializations is a common heuristics to mitigate local minima
- The elbow and silhouette score are complementary heuristics for choosing K
- K-Means assumes spherical, equally sized clusters; it fails on rings, elongated shapes, and density differences

K-MEANS IS FLAWED. Every failure mode in this chapter reduces to the same problem; Euclidean distance to a centroid is the wrong measure of membership for that data. But there is a more structural

limitation: K-Means must commit to a single K. With K=3 you see roast regions; with K=10 you see brewing methods. You cannot see both at once, and the elbow heuristic only helps you pick one level and drop the remaining pattern.

TEASER How can we built the full hierarchy of clusterings first, capture the complete pattern across all levels, and allow for post hoc selections?

FEEDBACK

Hierarchical Clustering

2026-05-06 · cheerful mango Haubentaucher

Ich springe von Level zu Level zu Level.

The Why

K-Means left us with three sharp problems.

The centroid assumption forces every cluster to be convex and compact. A single mean position must stand in for all members, so the algorithm breaks whenever clusters are elongated, ring-shaped, or split across multiple densities. All cluster characteristics and structure information are lost and filtered when the mean is computed. The centroid then lands somewhere no data actually lives, and every assignment made from that position is wrong by construction.

Sensitivity to initialization and outliers means results are non-deterministic. A single extreme point can pull a centroid far from the dense mass of its cluster, silently distorting every assignment downstream. Different random starts can converge to entirely different partitions of the same data.

A single scale of structure is all K-Means can see at once. With K=2 you recover two broad groups; with K=10 you recover ten fine subgroups. Both are valid descriptions of the same data at different levels of zoom, yet K-Means forces you to choose one and discard the other. This is not a flaw in the algorithm but a flaw in the question: asking "how many clusters?" presupposes that data has one natural scale.

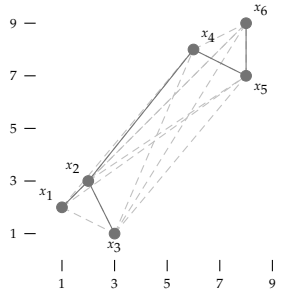


Figure 26: The canonical six points with all pairwise distances (dashed) and the full single-linkage hierarchy (solid). The last merge connects x_2 and x_4 across the two groups. No centroids needed; only pairwise distances drive the clustering.

Hierarchy is the natural answer. Structure in real data rarely lives at a single scale. Biological taxonomy nests species inside genera inside families inside orders. A document corpus organizes words into topics and topics into domains. File systems nest folders inside

folders. In each case, the right question is not “how many clusters?” but “which level of abstraction do I need right now?”.

Hierarchical clustering produces a dendrogram: a binary tree in which leaves are individual data points and each internal node records the distance at which two sub-trees were merged. The full range of partitions, from $K{=}1$ to $K{=}n$, lives in this one structure.

The same distance matrix that K-Means uses to assign points to centroids can be used to build an entire tree of clusterings. The tree is built in one pass; every value of K is then available for free.

In order to move from a flat partition to a full hierarchy of clusterings, we have good reason to find ways to,

- extend pairwise point distances to cluster distances, making the notion of closest pair well-defined at every scale.
- record each merge in a tree structure that makes every partition from $K{=}1$ to $K{=}n$ readable in a single pass.
- understand how the choice of cluster distance and linkage design shapes the resulting hierarchy and where it fails.

Hands-On Experience

How would you build a hierarchy of clusterings over these six points? Which is the cluster on the highest level of abstraction, K=1? How easy is it to build hierarchy top down? Which are the two clusters on the second highest level, K=2? Still obvious, then it gets trickier when K=3 the first non-trivial design-choices are to be made.

Divisive clustering (top-down) starts with all points in one cluster and recursively splits the cluster with the largest diameter. It is rarely used in practice: splitting is more expensive than merging, and early splits made with global information can be noisier than local merges.

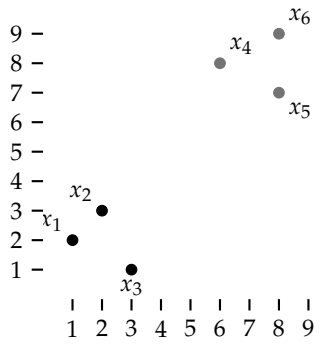


Figure 27: The canonical six points. Which two are closest? Try building a hierarchy top-down, then bottom-up, which one is easier?

Building Hierarchy Bottom Up is more intuitive. Which points are closest to each other? See how this again draws on our understanding of distance as a proxy for similarity? Connect them with a line. Now find the next closest pair and connect them. Keep going until all six points are linked into a tree. At what level of distance would you cut that tree to recover two groups? Three groups?

A horizontal cut at any chosen height yields a flat partition, as we are used to as a result of K-Means.

The fundamental question this chapter answers is how do we formalize the bottom-up merging process? We will see how agglomerative clustering builds a dendrogram from pairwise distances; and how different linkage criteria encode different assumptions about what makes two clusters close.

Linkage, is non-trivial. Two points intuitively merge into a cluster, but beyond that we need a rule for measuring the distance from that cluster to everything else. Do we use the closest pair of points across the two clusters? The furthest? The average? Each answer is a different linkage criterion, and each one tells a different story about what "nearby clusters" means.

The Learning Objectives of this lecture:

- Understand how agglomerative clustering builds a dendrogram through successive minimum-distance merges, and read merge-height gaps to identify the natural number of clusters.
- Apply single, complete, and Ward's linkage and explain how each criterion shapes the resulting hierarchy.
- Identify when hierarchical clustering succeeds where K-Means fails, recognise its failure modes, and explain why scalability requires approximations such as BIRCH.

The Hierarchical Clustering Algorithm

THE AGGLOMERATIVE STRATEGY[6] is bottom-up: given data $X = \{x_1, \ldots, x_n\}$, initialize n singleton clusters $C_i = \{x_i\}$ in an active set and precompute a distance matrix $D[i, j] = d(x_i, x_j)$. At each step the closest pair $(i^*, j^*) = \arg\min_{i \neq j} D[i, j]$ is merged into $C_m = C_{i^*} \cup C_{j^*}$; the distances to remaining active clusters are updated via linkage criterion L. Every merge is recorded as an internal node of the dendrogram T, labeled with the distance $D[i^*, j^*]$ at which it occurred.

[6] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. DOI: 10.1080/01621459.1963.10500845. https://scholar.google.com/scholar?q=Ward+1963+hierarchical+grouping+optimize+objective+function

THE RESULT T is a binary tree. Every valid flat partition from K=1 to K=n can be read off by making a horizontal cut at the desired height. The $n - 1$ merge heights span the full range from the smallest pairwise distance to the diameter of the data.

Algorithm 2: Agglomerative Hierarchical Clustering.

Require: Data $X = \{x_1, \ldots, x_n\}$, linkage criterion L
Ensure: Dendrogram T
1: Initialize: $C_i \leftarrow \{x_i\}$ for $i = 1, \ldots, n$; active $\leftarrow \{C_1, \ldots, C_n\}$
2: Compute $D[i, j] \leftarrow d(x_i, x_j)$ for all $i \neq j$
3: **while** |active| > 1 **do**
4: $(i^*, j^*) \leftarrow \arg\min_{i \neq j} D[i, j]$
5: **Record merge:** (C_{i^*}, C_{j^*}) at height $D[i^*, j^*]$ in T
6: $C_m \leftarrow C_{i^*} \cup C_{j^*}$
7: Update $D[m, k] \leftarrow L(C_m, C_k)$ for each active $C_k \neq C_{i^*}, C_{j^*}$
8: Remove C_{i^*} and C_{j^*} from active; add C_m
9: **end while**

NOTICE THAT L IS THE ONLY HYPERPARAMETER TO DETERMINE. The algorithm is otherwise fully deterministic.

INITIALIZATION IS DETERMINISTIC. Each point starts as its own singleton cluster, there is nothing to choose. This stands in sharp contrast to k-means, where a random centroid draw forces multiple restarts for optimization. Here, the starting state is uniquely determined by the data, so two runs on the same dataset always produce the same dendrogram.

Strictly, the dendrogram is unique only when all pairwise distances are distinct. When ties occur, the merge order depends on the tie-breaking rule, typically lowest index first.

TRACE THROUGH OUR SIX POINTS. Dotted edges represent pairwise distance calculations; the dashed edge marks the current merge; gray edges record prior ones.

REMOVE TWO. ADD ONE. In every iteration reduces |active| by exactly one (two clusters are removed and one merged cluster is added)

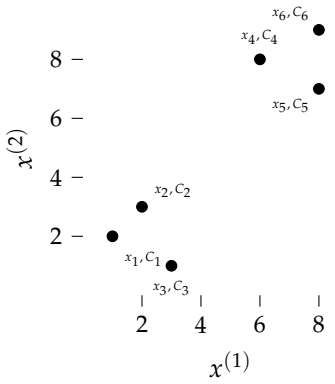


Figure 28:

INITIALIZATION: n=6 singleton clusters C_i=$\{x_i\}$ placed in the active set.

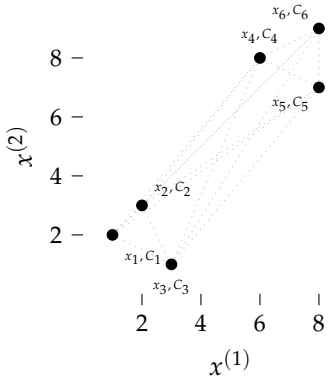


Figure 29:

COMPUTE $D[i,j]$=$d(x_i, x_j)$ for all $\binom{6}{2}$=15 pairs. Each dotted edge is one entry in the distance matrix. At every iteration the algorithm selects the globally shortest edge, $(i^*, j^*) = \arg\min_{i \neq j} D[i,j]$, where i, j range only over the current active set. Once C_{i^*} and C_{j^*} are merged into C_m, they are removed from active and replaced by C_m; past merges are never revisited.

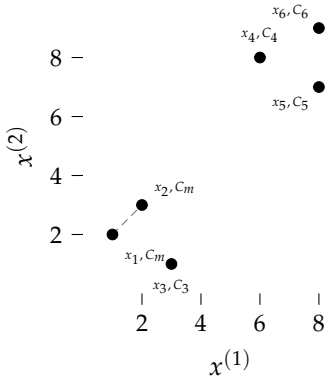


Figure 30:

MERGE 1: the globally smallest distance is $d(x_1, x_2)$. Singletons $\{x_1\}$ and $\{x_2\}$ merge into C_m=$\{x_1, x_2\}$, recorded as an internal node in T with both singletons as children, labeled by $d(x_1, x_2)$: T += $C_7 = (C_1, C_2, d(x_1, x_2))$.

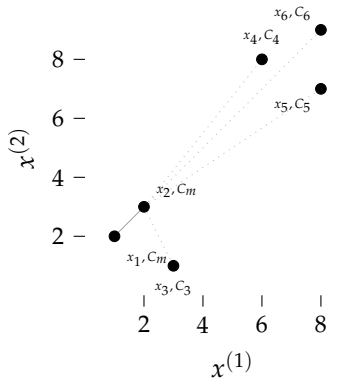


Figure 31:

UPDATE D: after forming $C_7{=}\{x_1, x_2\}$, four rows and columns of D are refreshed via $D[7,k] \leftarrow L(C_7, C_k)$. Under single linkage L takes the nearest member: $D[7,k] = \min_{x \in C_7} d(x, C_k)$. Here x_2 is the nearest member of C_7 to all remaining singletons, so every new entry anchors at x_2 (dotted edges).

DISCUSS LINKAGE

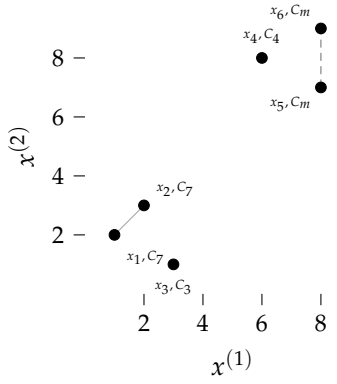


Figure 32:

MERGE 2: the next minimum in the updated D is $d(x_5, x_6)$. Singletons $\{x_5\}$ and $\{x_6\}$ merge into $C_m{=}\{x_5, x_6\}$, recorded in T as $T \mathrel{+}= C_8 = \big(C_5,\, C_6,\, d(x_5, x_6)\big)$.

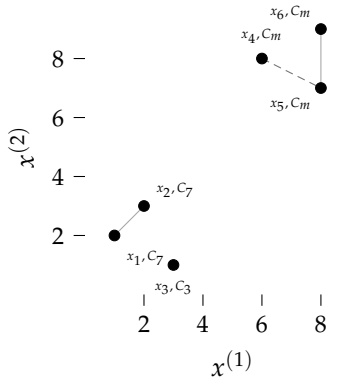


Figure 33:

MERGE 3: $D\big(C_8, x_4\big)$ is minimal under single linkage (distances to a merged cluster use the nearest member). $D\big(C_7, x_3\big)$ ties but x_4 merges first by index. Cluster C_8 absorbs x_4, linking via x_5: $T \mathrel{+}= C_9 = \big(C_8,\, C_4,\, D(C_8, C_4)\big)$.

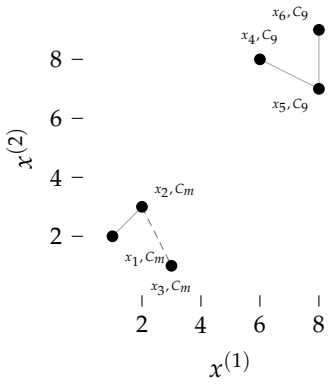


Figure 34:

MERGE 4: also at height $\sqrt{5}$, $D(C_7, x_3) = \sqrt{5}$. Cluster C_7 absorbs x_3, linking via x_2: $T \mathrel{+}= C_{10} = (C_7, C_3, D(C_7, C_3))$.

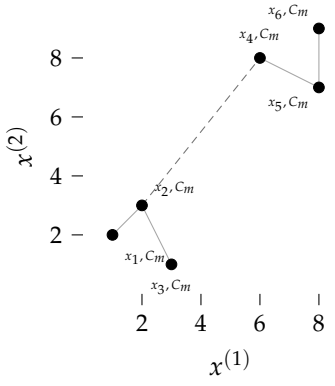


Figure 35:

MERGE 5: $C_{10}=\{x_1, x_2, x_3\}$ and $C_9=\{x_4, x_5, x_6\}$ merge at height $d(x_2, x_4)$, the nearest cross-cluster pair. All six points form one cluster; the dendrogram T is complete.

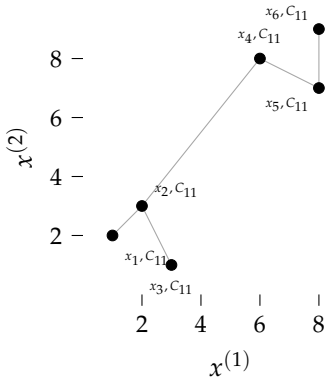


Figure 36:

TERMINATION: |active|=1. All n=6 points belong to C_{11}; the five merge edges span the complete single-linkage dendrogram. No further merges are possible.

TERMINATION. The algorithm halts when $|\text{active}| = 1$: exactly one cluster remains, containing all n points. Termination is guaranteed after exactly $n - 1$ merges. At that point T holds $n - 1$ internal nodes, one per merge, and is the complete dendrogram.

LINKAGE CRITERIA BEYOND TWO POINTS IS NON-TRIVIAL determine how the distance between two clusters is computed from the distances between their individual members. The choice of linkage profoundly shapes the dendrogram. Swap L and you get a fundamentally different clustering.

ANSWERS FROM WHERE WE MEASURE THE DISTANCE. The distance then still decides what we are going to merge.

SINGLE LINKAGE

$$D_{\text{single}}(C_i, C_j) = \min_{x \in C_i,\, y \in C_j} d(x, y)$$

The distance between two clusters is the distance between their nearest members. Single linkage admits an implementation based on Prim's minimal spanning tree (details later), which is fast. This also lets it detect elongated, non-convex clusters. This behavior is called chaining, and is also a weakness; a single close pair of points can bridge two otherwise distant clusters, pulling them into one.

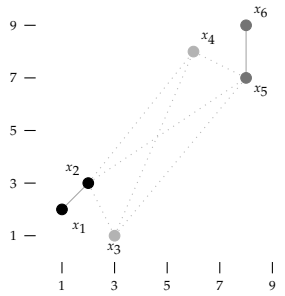


Figure 37: Single linkage: dotted lines show d_{min} for each of the six active cluster pairs. The smallest, $d(x_2, x_3)$=$d(x_4, x_5)$, determines the next merge.

COMPLETE LINKAGE

$$D_{\text{complete}}(C_i, C_j) = \max_{x \in C_i,\, y \in C_j} d(x, y)$$

The distance between two clusters is the distance between their furthest members. Complete linkage tends to produce compact, roughly equal-diameter clusters and is robust to chaining. It may, however, split naturally elongated shapes.

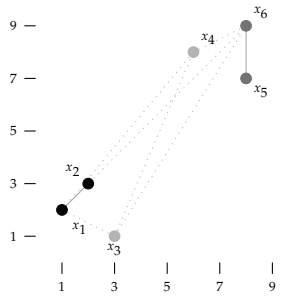


Figure 38: Complete linkage: dotted lines show d_{max} for each active cluster pair. The largest determines which merge is most expensive.

AVERAGE LINKAGE

$$D_{\text{average}}(C_i, C_j) = \frac{1}{|C_i|\,|C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$$

Also known as the Unweighted Pair Group Method with Arithmetic Mean. Is a compromise: less prone to chaining than single linkage, less aggressive at squashing elongated clusters than complete linkage. Average linkage averages all cross-cluster pairwise distances.

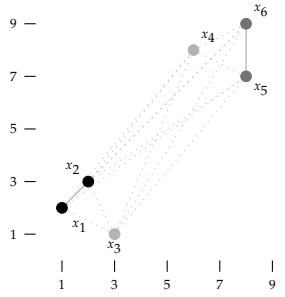


Figure 39: Average linkage: D averages all pairwise cross-cluster distances. Every dotted line contributes to the mean.

Ward's Method

Ward's criterion merges the two clusters whose union minimises the increase in total within-cluster variance:

$$D_{\text{Ward}}(C_i, C_j) = \frac{|C_i|\,|C_j|}{|C_i| + |C_j|} \left\|\mu_i - \mu_j\right\|^2$$

where μ_i and μ_j are the centroids of C_i and C_j.

$\|\cdot\|^2$ is a squared norm, not a Euclidean distance. Ward's values are variance increases; single and average linkage values are distances. The scales are not comparable.

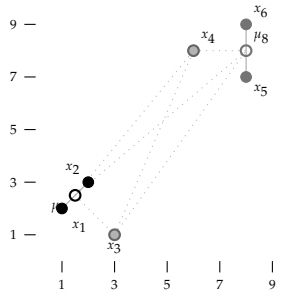
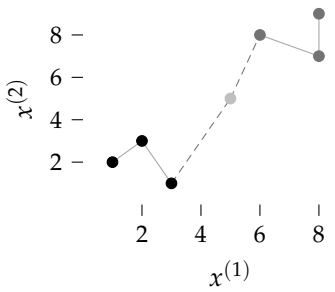


Figure 40: Ward's linkage: dotted lines connect centroids of all active cluster pairs; D is proportional to $\|\mu_i - \mu_j\|^2$ weighted by cluster size.

When Hierarchical Clustering Fails the root cause is that the linkage criterions optimises locally with no global view nor understanding for scale. Every point gets clustered at some point in the agglomeration.



Chaining and bridging

Figure 41: Single-linkage chaining: one intermediate point (light) bridges two well-separated clusters. Within-cluster merges (solid) complete first; single linkage then greedily closes through the bridge (dashed), incorrectly joining both clusters into one.

Shape Bias Ward's linkage inherits K-Means's spherical bias exactly. The difference is recovery: K-Means re-assigns points each iteration and can partially correct a bad partition. Ward's cannot; the bias compounds irreversibly.

Irreversible greedy merges and local optima. At each step the algorithm selects the globally optimal merge under the current distance matrix: no local minimum trap exists within a single iteration. However, the greedy (no-reassignment) strategy is myopic. Once C_{i^*} and C_{j^*} are merged into C_m, that decision is permanent; no subsequent step can reverse it. A suboptimal early merge propagates through all future distance updates via L, and the resulting dendrogram may differ substantially from the globally optimal hierarchy.

Irrecoverability is what makes chaining and shape bias genuinely dangerous. A bad merge silently propagates through every subsequent distance update; the only remedy is to restart from scratch.

COMPUTATIONAL COMPLEXITY Memory is the hard wall. The distance matrix $D_{i,j}$ requires $O(n^2)$ space. At n=100,000 that is already ≈40 GB for float32: you either fit it in RAM or the algorithm cannot run.

$D_{i,j}$ has $\frac{n(n-1)}{2}$ entries; $O(n^2)$ space.

FLOAT32 AT SCALE. n=10,000 needs ≈400 MB; n=100,000 needs ≈40 GB.

COMPUTE IS THE SOFT WALL. Even when the matrix fits, filling it costs $O(n^2)$. The merge steps then scan and update distances after each of the $n-1$ merges, adding $O(n^2)$ for optimised linkages or $O(n^3)$ for naive implementations.

Algorithm	Time	Notes
K-Means (baseline)	$O(n)$	K, T, d treated as constants
Naive agglomerative	$O(n^3)$	Recomputes all distances each step
Single linkage (SLINK)	$O(n^2)$	Optimal algorithm for single linkage
Average linkage	$O(n^3)$	Without acceleration

Table 2: Complexity of hierarchical clustering variants; K-Means is the linear baseline (K, T, d constant). Naive agglomerative is $O(n^3)$: it rescans the full distance matrix at each of the $n-1$ merge steps.

SINGLE LINKAGE ACHIEVES $O(n^2)$ because its update rule is a simple minimum, as the minimal distance to a merged cluster is the minimum of the distances to its two constituent, deprecated clusters:

$$d(C_i \cup C_j,\, C_k) = \min\big(d(C_i, C_k),\, d(C_j, C_k)\big)$$

SLINK[7] exploits this as each new point requires only a single scan of the current distances; $O(n)$ per update step over $n-1$ merges gives $O(n^2)$ total time.

SCALABILITY BY COMPRESSION is regained by BIRCH[8], which breaks the $O(n^2)$ memory barrier by compressing data into a compressed tree in one pass, then running agglomeration on the leaf nodes rather than individual points reducing cost. A threshold T controls compression: A small T produces many fine-grained leaves (accurate but larger tree); a large T merges more points per leaf (faster but coarser). Setting T too large can merge points from different true clusters before agglomeration begins.

[7] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973. DOI: 10.1093/comjnl/16.1.30. https://scholar.google.com/scholar?q=Sibson+1973+SLINK+optimally+efficient+single-link+cluster

[8] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, 1996. DOI: 10.1145/233269.233324. https://scholar.google.com/scholar?q=Zhang+Ramakrishnan+Livny+1996+BIRCH+efficient+data+clustering

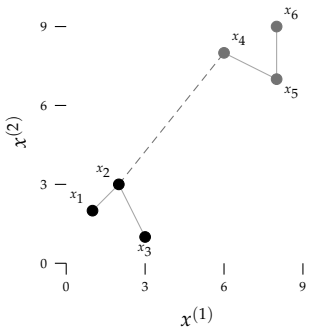
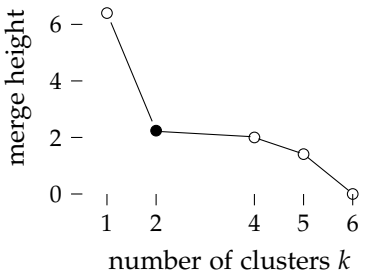


Figure 42: Six points displaying the cut cluster link executed in the dendrograms.

DENDROGRAM VOCABULARY borrows from genealogy and graph theory. The dendrogram T is a binary tree with n leaves and $n-1$ internal nodes. Each data point x_i is a leaf ℓ_i at height $h=0$. Each internal node v_k records a merge of two children v_a, v_b at height $h_k=D[i^*, j^*]$; the merged cluster is $C_k=C_a \cup C_b$. The root v_{n-1} sits at the highest merge height h_{n-1} and contains all n points. Following the tree upward from any node gives its ancestors; following downward gives its descendants.

READING A DENDROGRAM requires only one rule: a horizontal cut at height h partitions the data into as many clusters as there are vertical stems crossing that line. Cutting above $h_4=6.40$ gives one cluster; cutting between $h_3=2.24$ and $h_4=6.40$ gives two; cutting between $h_1=1.41$ and $h_2=2.00$ gives four; cutting below h_1 gives n singletons. A large gap between consecutive merge heights is the dendrogram equivalent of the elbow: it suggests a natural number of clusters. The kink at $k=2$ (filled dot) marks the jump from 2.24 to 6.40, confirming two natural clusters. Notice the missing value at $k = 3$.



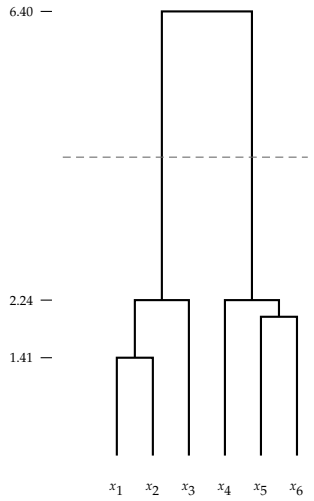


Figure 43: Single-linkage dendrogram on true scale. Heights are Euclidean distances. The jump from 2.24 to 6.40 (dashed cut) yields $C_1=\{x_1, x_2, x_3\}$, $C_2=\{x_4, x_5, x_6\}$.

WARD'S DENDROGRAM for the same six points is shown alongside. Heights are no longer Euclidean distances but variance increases: each merge height records the rise in total within-cluster variance caused by that merge.

COMPARE THE GAPS. Single linkage gave a gap from 2.24 to 6.40 (ratio $\approx$2.9$\times$). Ward's gives a gap from 3.00 to 96.6 (ratio $\approx$32$\times$). Ward's tends to amplify the gap signal, making cluster number selection more pronounced.

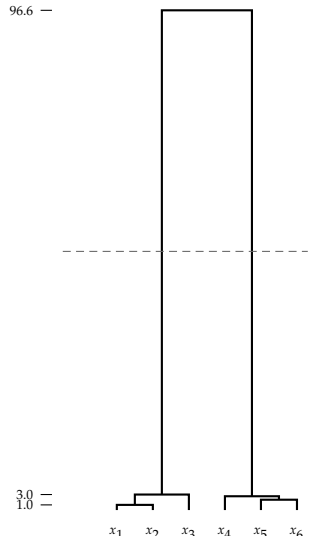


Figure 44: Ward's dendrogram on true scale (Δvar). Four merges pack below 3.0; the final merge towers at 96.6. The gap dwarfs the one in the single-linkage dendrogram above.

How do we infer new data points? The dendrogram is a static structure built on the original data; it does not directly provide a way to assign new points to clusters. But, given a cut of the dendrogram into K clusters, we can assign a new point x_{new} to the cluster whose centroid is closest under the same distance metric used for clustering. Or we assign it to a cluster based on the closest leaf or a k-nearest neighbor approach.

Examples & Exercises

GIVEN THREE POINTS x_1=(1,1), x_2=(3,1), x_3=(8,7), run single-linkage agglomerative clustering step by step. These are the same points you clustered with K-Means already, but this time there is no centroid initialization, no random seed, no choice of K. Your tasks: compute the initial pairwise distance table, identify the minimum pair at each step, record each merge, update distances using single linkage, and construct the dendrogram.

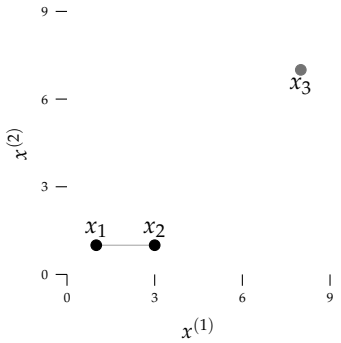


Figure 45: The same three points from the K-Means exercise. No centroids this time; the algorithm needs only pairwise distances.

REUSE BY DESIGN. Working on the same three points lets you compare the two algorithms directly: K-Means needed an initialization and an iterative loop; hierarchical clustering needs neither.

INITIAL DISTANCES. For each pair (x_i, x_j) compute the euclidean distance. With n=3 there are only $\binom{3}{2}$=3 pairs. Here is the full calculation for $d(x_1, x_2)$:

$$\begin{aligned} d(x_1, x_2) &= \sqrt{(3-1)^2 + (1-1)^2} \\ &= \sqrt{4+0} \\ &= 2 \end{aligned}$$

Compute the remaining two distances yourself, then verify against the table below. We use symmetries to reduce compute, so only the upper triangle of the distance matrix is filled; the diagonal is empty since $d(x_i, x_i) = 0$ is trivial and not needed for merges.

	x_1	x_2	x_3
x_1	–	**2.00**	8.49
x_2	–	–	7.81
x_3	–	–	–

Table 3: Initial pairwise Euclidean distances. The global minimum $d(x_1, x_2)$=2 (bold) triggers the first merge.

FIRST MERGE. The minimum entry is $d(x_1, x_2)$=2. Merge x_1 and x_2 into cluster $\{x_1, x_2\}$ at height 2. Update the distance to the remaining point using single linkage. Remember that you do not have to recompute the full distance matrix; only the distances to the new cluster $\{x_1, x_2\}$, repurposing the already computed distances to x_1 and x_2 and solving for the minimum:

$$\begin{aligned} d(\{x_1, x_2\},\, x_3) &= \min(d(x_1, x_3),\, d(x_2, x_3)) \\ &= \min(8.49,\, 7.81) \\ &= 7.81 \end{aligned}$$

SECOND MERGE. Only two active clusters remain: $\{x_1, x_2\}$ and $\{x_3\}$. Their distance is 7.81, so they merge at height 7.81. The dendrogram is complete.

Step	Clusters merged	Height	Updated distances (single linkage)
1	$x_1,\ x_2$	2.00	$d(\{x_1, x_2\},\ x_3)$=7.81
2	$\{x_1, x_2\},\ x_3$	7.81	Final merge; algorithm complete

Table 4: Single-linkage merge history for x_1, x_2, x_3. The gap from 2.00 to 7.81 confirms two natural groups, matching the K-Means result from Chapter 2.

GUARANTEED TERMINATION. How many merges did you just perform? Could there have been more? Fewer? Why?

EACH MERGE REMOVES TWO CLUSTERS from the active set and adds one, reducing |active| by exactly one. Starting from n singletons, the algorithm reaches |active|=1 after exactly $n-1$ steps.

- For n=3: exactly 2 merges.
- For n=1,000: exactly 999 merges.

This also means the dendrogram always has n leaves and $n-1$ internal nodes. Verify this on your two-merge trace: 3 leaves, 2 internal nodes, one complete binary tree T.

CONTRAST WITH K-MEANS. K-Means has no guaranteed iteration count. It can converge in 2 iterations or 200, depending on initialization. Hierarchical clustering always finishes in $n-1$ steps, with no restarts and no randomness.

READING THE DENDROGRAM. Sketch the dendrogram for the three-point trace. It has two horizontal bars: one at height 2.00 (merging x_1, x_2) and one at height 7.81 (merging everything). Now answer the following:

- Cut at h=5. How many clusters do you get? Which points are in each?
- Cut at h=1. How many clusters? What does this tell you?
- What is the largest gap between consecutive merge heights? What K does it suggest?
- How does the spread of the nodes in the dendrogram relate to an elbow plot over K?

LEAF ORDERING TRAP. A dendrogram with n leaves has 2^{n-1} valid leaf orderings: at each internal node you can swap left and right children without changing the hierarchy. Two points adjacent in the plot are not necessarily similar; only the merge height tells you when they joined.

THE GAP is $7.81 - 2.00 = 5.81$: the final merge costs nearly four times as much as the first. A cut anywhere in this gap yields K=2 with C_1=$\{x_1, x_2\}$, C_2=$\{x_3\}$, exactly the partition K-Means found.

LINKAGE COMPARISON WITH CHAINING. Add a bridge point x_4=(5, 4) to the three points. This point sits between the two natural groups in a region where no cluster really lives.

Compute the initial $\binom{4}{2}$=6 pairwise distances. Then trace single linkage to completion. Based on our previous trace, reuse the distances you already calculated, prior to the merge of x_1, x_2 at height 2.00 and calculate the missing distances to x_4. You should find the second merge of $\{x_1, x_2\}, x_4$ at height $\approx$ 3.61. Finally merge $\{x_1, x_2, x_4\}, x_3$ at height $\approx$ 4.24

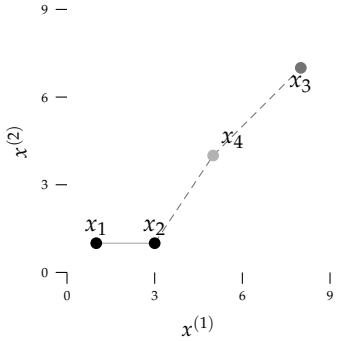


Figure 46: Bridge point x_4 in the gap between the two natural groups. Dashed lines show the chain that single linkage will follow.

THE BRIDGE POINT chains the two groups together through nearest-neighbour links: $x_2 \to x_4 \to x_3$. What happens with our gap signal? The first merge is still x_1, x_2 at height 2.00, but the second merge is now $\{x_1, x_2\}, x_4$ at height 3.61 deluting our signal, and the final merge is $\{x_1, x_2, x_4\}, x_3$ at height 4.24.

NOW REPEAT WITH COMPLETE LINKAGE. Replace each min in the update rule with max. After step 1 (still x_1, x_2 at height 2.00), the updated distance from $\{x_1, x_2\}$ to x_4 under complete linkage becomes:

$$\begin{aligned} d_{\text{comp}}(\{x_1, x_2\},\ x_4) &= \max(d(x_1, x_4),\ d(x_2, x_4)) \\ &= \max(5.00,\ 3.61) \\ &= 5.00 \end{aligned}$$

Check whether x_4 still merges with $\{x_1, x_2\}$ next, or whether a different pair is closer. Trace the remaining merges and compare the complete-linkage dendrogram to the single-linkage one. Which linkage is more vulnerable to the bridge point? Why?

WHY SINGLE LINKAGE IS FAST. Look back at the distance update you performed in the single-linkage trace. When C_i and C_j merge into C_m, the new distance to any remaining cluster C_k is:

$$D_{\text{single}}(C_m,\ C_k) = \min\big(D(C_i,\ C_k),\ D(C_j,\ C_k)\big)$$

One comparison, two table lookups, no access to the original data points. The update depends only on values already in the distance matrix. This means the entire algorithm can run on the $n \times n$ matrix alone; once initialized, the raw data is never touched again.

SLINK exploits exactly this property. Because single linkage only needs nearest-neighbour information, the merge sequence is equivalent to a minimum spanning tree. Prim's algorithm builds that tree in $O(n^2)$, matching SLINK's optimal complexity.

NOW COMPARE WITH WARD'S UPDATE. Recall that Ward's distance measures the increase in total within-cluster variance caused by a merge. For two clusters C_a and C_b with centroids μ_a, μ_b and sizes n_a, n_b:

$$D_{\text{Ward}}(C_a, C_b) = \frac{n_a \cdot n_b}{n_a + n_b} \|\mu_a - \mu_b\|^2$$

To update the distance after a merge, you must recompute the centroid of the merged cluster and then apply the definition again. This means going back to the raw data, unlike single linkage which never looks beyond the distance matrix.

SHORTCUT. The Lance-Williams formula expresses Ward's update purely in terms of existing table entries and cluster sizes, avoiding the detour through centroids. The merge loop still runs in $O(n^2 \log n)$ at best and $O(n^3)$ in the naive implementation, compared to $O(n^2)$ for single linkage.

VERIFY THE DIFFERENCE. For the three-point trace, compute the Ward distance update for $D_{\text{Ward}}(\{x_1, x_2\}, x_3)$ from the definition and confirm that it gives a different value than the single-linkage update. Ward's initial distance between two singletons reduces to $\frac{1}{2}\|x_i - x_j\|^2$. The first merge is still x_1, x_2 (at Ward height $\frac{1}{2} \cdot 4 = 2.00$). After merging, the centroid of $\{x_1, x_2\}$ is $\mu_{12} = (2, 1)$. Now apply the definition to compute the distance to the remaining singleton x_3:

$$\begin{aligned}
D_{\text{Ward}}(\{x_1, x_2\}, x_3) &= \frac{n_{12} \cdot n_3}{n_{12} + n_3} \|\mu_{12} - x_3\|^2 \\
&= \frac{2 \cdot 1}{2 + 1} \left\| \begin{pmatrix} 2 \\ 1 \end{pmatrix} - \begin{pmatrix} 8 \\ 7 \end{pmatrix} \right\|^2 \\
&= \frac{2}{3} \left\| \begin{pmatrix} -6 \\ -6 \end{pmatrix} \right\|^2 \\
&= \frac{2}{3}\left((-6)^2 + (-6)^2\right) \\
&= \frac{2}{3}(36 + 36) \\
&= \frac{2}{3} \cdot 72 \\
&= 48.00
\end{aligned}$$

Single linkage gave 7.81 (a Euclidean distance); Ward's gives 48.00 (a variance increase). The numbers are not comparable across linkage criteria, but within each criterion they determine the merge order.

BACK-OF-ENVELOPE: SCALABILITY. The distance matrix stores $\binom{n}{2}$ entries. For n=50,000 with 64-bit floats, that is $\binom{50000}{2} \times 8 \approx 9.3$ GB. For n=100,000: roughly 37 GB.

CODE will be provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

BACK AT THE COFFEETABLE. Same 1 797 coffee aroma fingerprints, ten brewing methods, two coordinates. Compute the pairwise distance matrix, implement single, complete, and average linkage from

scratch, then run all four methods and compare the truncated dendrograms. Cut each tree at $K=3$, colour the scatter plot, and compare with K-Means at $K=3$.

WHAT TO OBSERVE. Single linkage chains into one dominant cluster. Complete and Ward yield balanced partitions; the K-Means result most closely resembles Ward's, because both favour compact, spherical clusters. One hierarchical run gives you every K from 1 to n.

SELF-REFLECTION AND RECAP

SELF-REFLECTION questions to guide your thinking:

- What does the merge height in a dendrogram represent, and why does a large gap between consecutive heights signal a natural number of clusters?
- Why does single linkage produce chaining while complete linkage does not? What property of each formula causes this difference?
- In what situations would you prefer Ward's linkage over single or complete linkage? When would Ward's fail?
- You compute a dendrogram and find that the top-level merge height is only marginally larger than the second-to-top merge height. What does this tell you about the cluster structure?
- What is the computational bottleneck of hierarchical clustering, and what approximate methods exist to overcome it at scale?
- Why is a bad early merge more damaging in hierarchical clustering than a bad initial assignment in K-Means? What mechanism does K-Means have that the dendrogram lacks?

RECAP of Key Concepts:

- Agglomerative clustering greedily merges the two closest clusters at each step, producing a dendrogram that encodes all flat partitions from $K=1$ to $K=n$ in a single structure
- The algorithm terminates in exactly $n-1$ merges; no initialization, no restarts, no convergence check
- Ward's linkage minimises the same within-cluster variance as K-Means; a single Ward dendrogram encodes all partitions from $K=1$ to $K=n$, each equivalent to a K-Means run for that K but without

iterative re-assignment, so Ward's inherits the spherical bias and cannot correct it

- Single linkage detects elongated clusters but suffers from chaining; complete and Ward's linkage produce compact clusters and are more robust to sparse bridge points
- Single linkage is uniquely fast ($O(n^2)$ via SLINK) because its update rule reduces to a single min over existing table entries, equivalent to a minimum spanning tree
- A large gap between consecutive merge heights is the dendrogram signal for a natural number of clusters, analogous to the elbow in K-Means inertia curves
- Leaf adjacency in a dendrogram does not imply similarity; only the merge height matters

THE ROOT OF HIERARCHICAL FAILURE. Every linkage criterion measures distance between points or boundaries locally. No criterion has a concept of sparsity or of low-density regions. In the next Chapter we will introduce and leverage the concept of density. Starting from the single question that hierarchical clustering cannot answer; what are the characteristics of a space or neighbourhood?

TEASER How can we define a cluster without any centroid, any linkage criterion, or any choice of K, label outliers as noise automatically, and still recover clusters of arbitrary shape?

FEEDBACK

Density-Based Clustering

2026-05-06 · cheerful mango Haubentaucher

It is all about that space, about that space.

The Why

The clustering algorithms we have seen so far share a common mechanism of assigning points to clusters based on their distance to an other point; a centroid or a linkage point. Every point, no matter how isolated, is absorbed into some cluster latest at the final merge. Real data contains noise: GPS measurements with dropouts, sensor readings during interference, survey responses that are genuine outliers.

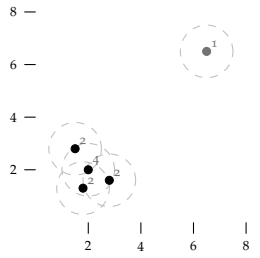


Figure 47: Each point carries its own dashed ε-neighbourhood; the small grey digit next to a point gives the number of points inside that circle, itself included. In the dense cluster at the lower left the four neighbourhoods overlap and chain the points together; the isolated p_n at the upper right has no companions within ε. Density, not distance to a centre, determines membership.

No centroid required. A density-based cluster is a space, not a mean and not a link. Its shape is implicit in the data, seeded by the points themselves; nothing forces it to be convex or spherical; detects noise, outliers and sparse bridges naturally, and avoids committing to K in advance. This geometric-assumption-free view is both the strength of density-based methods and, as we will see, the source of their own limitations.

In order to move from pairwise distances to density-aware neighbourhoods we have good reason to find ways to,

- define what it means for a region of space to be dense or sparse.
- grow clusters of arbitrary shape, including non-convex shapes, from local density, without assuming convex or spherical structure.
- separate signal from noise, so that sparse outliers are labelled as such rather than forced into a cluster.
- let the data itself decide how many clusters exist, without committing to K in advance.

Hands On Experience

CONSIDER the same six canonical points, now with a seventh point x_7=(4, 5) placed between the two natural groups.

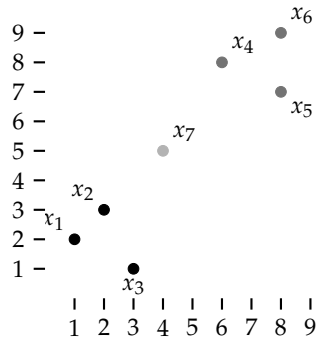


Figure 48: Six canonical points plus x_7=(4, 5) sitting between the two natural groups. Which points lie in dense neighbourhoods, and which sits in a sparse region?

DRAW A CIRCLE of radius ε=2.5 around each point and count who sits inside. The three bottom-left points find each other, the three top-right points find each other, and x_7 finds empty space; read off the data without ever choosing a number of clusters.

THE NEIGHBOURHOOD RADIUS ε is the distance within which a point looks for companions. Too small and every point is left alone; too large and everything collapses into one cluster. The right value lives where the data transitions from dense to sparse, and we will read it off the data itself using the k-distance plot.

OUR APPROACHES SO FAR WOULD REFUSE to let x_7 be lonely. With K=2 it hands the point to whichever centroid happens to sit slightly closer, an assignment driven by initialisation rather than by any structure in the data. Hierarchical clustering would fold x_7 into some merge and bury it inside a cluster it never really joined.

DENSITY ALLOWS US TO DO SO. A point whose ε-neighbourhood is empty is not handed to the nearest cluster; it is labelled p_n. Cluster shape is read off local neighbourhoods with a sense for p_b's, the number of clusters emerges from where density breaks instead of from a chosen K, and sparse points are flagged instead of absorbed.

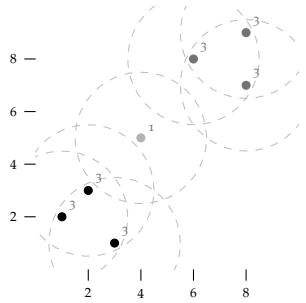


Figure 49: The seven canonical points with their dashed ε=2.5 neighbourhoods. The small grey digit next to each point gives $|N_\varepsilon|$, itself included. Each natural group forms a three-point neighbourhood; x_7 sits in a sparse region and finds only itself within ε.

THE LEARNING OBJECTIVES of this lecture:

- Trace a density clustering by hand on a small dataset and learn how to set the neighbourhood radius ε, and the minimum number of points $n_{\min}$ from the data itself.
- Understand the concepts of core (p_c), border (p_b), and noise (p_n) points by formalising neighbourhoods.
- Recognise when density-based clustering succeeds where K-Means and hierarchical fail, where they share characteristics, and where it fails in turn under varying density, and in high dimensionality.

The Density-Based Algorithm

DB SCAN[9] or Density-Based Spatial Clustering of Applications with Noise, builds clusters from local density rather than from a chosen number of centroids or a sequence of merges. Two parameters govern the algorithm: ε, the neighbourhood radius, and $n_{\min}$, the minimum number of points required to form a dense region.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231. AAAI Press, 1996. https://scholar.google.com/scholar?q=Ester+Kriegel+Sander+Xu+1996+density-based+algorithm+discovering+clusters

GIVEN DATA $X = \{x_1, \ldots, x_n\}$ and parameters $(\varepsilon, n_{\min})$, DB Scan returns a labelling

$$\tilde{y} : X \longrightarrow \{1, 2, \ldots, K, p_n\}$$

where K is determined by the data rather than fixed in advance. The output is a flat partition plus a p_n set N.

EVERY POINT IS CLASSIFIED into one of three roles before any cluster is formed. The ε-neighbourhood of a point p is the set of all points within distance ε:

$$N_\varepsilon(p) = \{q \in X : d(p, q) \leq \varepsilon\}.$$

- A point p_c is a core point if $|N_\varepsilon(p_c)| \geq n_{\min}$.
- A point p_b is a border point if $|N_\varepsilon(p_b)| < n_{\min}$ but $p_b \in N_\varepsilon(p_c)$ for some core point p_c.
- A point p_n is noise if it is neither core nor border.

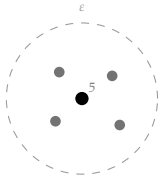


Figure 50:

CORE POINT. The black point has $|N_\varepsilon|$=5 $\geq n_{\min}$=4. Four companions sit inside its dashed ε-neighbourhood. The small grey digit at the upper right gives $|N_\varepsilon|$, the point itself included.

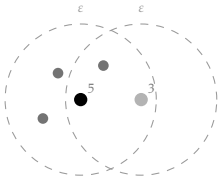


Figure 51:

BORDER POINT p_b. The light grey point on the right has $|N_\varepsilon|$=3 $< n_{\min}$=4. It is not a p_c itself, but it falls inside the ε-neighbourhood of the black p_c on the left and is absorbed into that cluster.

DENSITY REACHABILITY is not symmetric. A p_b is reachable from a p_c whenever $p_b \in N_\varepsilon(p_c)$, but p_c is not reachable from p_b because p_b is not a p_c. Cluster membership propagates from p_c's outward, never in reverse.

DENSITY CONNECTIVITY extends reachability to full cluster membership. Point q is density-reachable from p if there exists a chain of points $p{=}p_1, p_2, \ldots, p_m{=}q$ such that

$$p_{i+1} \in N_\varepsilon(p_i) \quad \text{and} \quad |N_\varepsilon(p_i)| \geq n_{\min} \quad \text{for } i = 1, \ldots, m-1.$$

Two points p, q are density-connected if there exists a third point o_c from which both are density-reachable; a cluster is then defined as a maximal set of pairwise density-connected points.

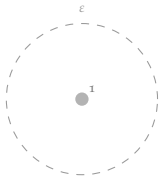


Figure 52:

NOISE POINT p_n. With $|N_\varepsilon|$=1 and no p_c within ε, the point belongs to no cluster. The dashed circle is otherwise empty.

TRACE THROUGH OUR SEVEN POINTS. Dashed circles are ε=2.5 neighbourhoods; dark points belong to C_1, grey to C_2, and an open circle marks p_n.

INITIALIZATION is trivial as the set of clusters and their core-memberships is the same regardless of the order in which points are processed. Any unvisited p_c triggers a new cluster; absorbing surrounding p_b's immediately; and growing the cluster by absorbing other p_c's and then their p_b's. Only p_b's that happen to lie in the ε-neighbourhood of several p_c's can be assigned to different clusters depending on visit order, but keeping their point class p_c, p_b, or p_n.

Algorithm 3: Naive Density-Based Clustering (DBSCAN)

Require: Data X, neighbourhood radius ε, density threshold $n_{\min}$
Ensure: Cluster label for each point (or p_n)

Mark all points as unvisited; $C \leftarrow 0$
for each unvisited point $p_i \in X$ **do**
  Mark p_i as visited
  Compute $N \leftarrow N_\varepsilon(p_i)$
  if $|N| < n_{\min}$ **then**
    Label p_i as p_n
  else
    Label p_i as p_c; $C \leftarrow C + 1$; assign p_i to cluster C
    for each $p_j \in N \setminus \{p_i\}$ **do**
      if p_j is unvisited **then**
        Mark p_j as visited
        Compute $N' \leftarrow N_\varepsilon(p_j)$
        if $|N'| \geq n_{\min}$ **then** label p_j as p_c; $N \leftarrow N \cup N'$
        end if
      end if
      if p_j has no cluster **then** assign p_j to C (as p_c or p_b)
      end if
    end for
  end if
end for

TERMINATION AND DETERMINISM. The outer loop visits every point at most once and the inner loop only grows by adding previously unvisited points, so the algorithm halts after $O(n)$ neighbourhood queries. The resulting cluster set is determined uniquely by the data and the pair $(\varepsilon, n_{\min})$: p_c's are fixed by the $|N_\varepsilon|$ threshold.

COMPLEXITY. Each of the n points triggers one ε-neighbourhood query costing $O(n)$, giving $O(n^2)$ overall. A spatial index (k-d or ball tree) reduces each query to $O(\log n)$, yielding $O(n \log n)$ in low dimensions.

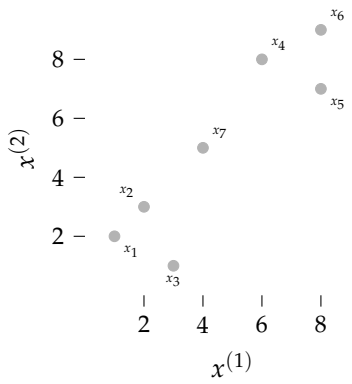


Figure 53:

INITIALIZATION: all n=7 points unvisited. Parameters ε=2.5, $n_{\min}$=2 are fixed. No seed necessary. Randomly picking x_1 as the first point to process.

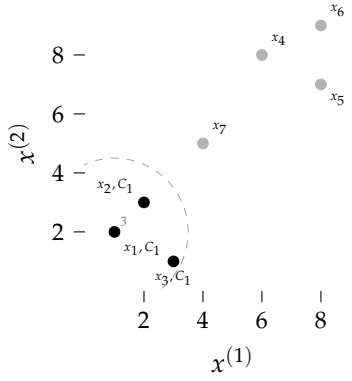


Figure 54:

VISIT x_1: $N_{2.5}(x_1)=\{x_1, x_2, x_3\}$, $|N|=3 \geq 2$, so x_1 is a p_c.

Seed cluster C_1; assign x_1 to C_1 and add x_2 and x_3 to the expansion queue. Points outside the dashed circle remain unvisited.

Distances: $d(x_1, x_2)=\sqrt{2}\approx 1.41$ ✓, $d(x_1, x_3)=\sqrt{5}\approx 2.24$ ✓.

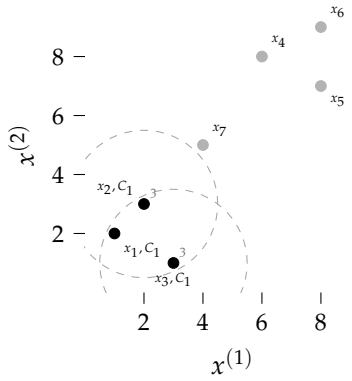


Figure 55:

EXPAND C_1: dequeue x_2: $N_{2.5}(x_2)=\{x_1, x_2, x_3\}$, all already assigned to C_1, no new points added.

Dequeue x_3: $N_{2.5}(x_3)=\{x_1, x_2, x_3\}$, same result.

Expansion queue empty: $C_1=\{x_1, x_2, x_3\}$ is sealed.

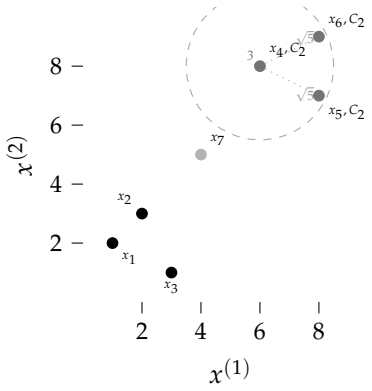


Figure 56:

VISIT x_4: $N_{2.5}(x_4)=\{x_4, x_5, x_6\}$, $|N|=3 \geq 2$, so x_4 is a p_c.

Seed C_2; expand from x_5 and x_6: both are p_c's and their neighbourhoods cover only $\{x_4, x_5, x_6\}$.

Distances: $d(x_4, x_5)=\sqrt{5}\approx 2.24$ ✓, $d(x_4, x_6)=\sqrt{5}\approx 2.24$ ✓, $d(x_4, x_7)=\sqrt{13}\approx 3.61$ ×.

$C_2=\{x_4, x_5, x_6\}$ sealed. Only x_7 remains unvisited.

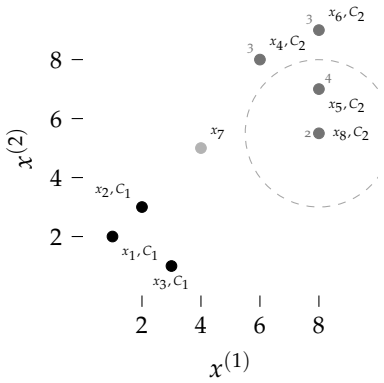


Figure 57:

HYPOTHETICAL POINT $x_8=(8, 5.5)$: $d(x_5, x_8)=1.5 \leq \varepsilon$, so x_5 absorbs x_8 into its neighbourhood.

$N_{2.5}(x_8)=\{x_8, x_5\}$, so $|N|=2 < n_{\min}=3$; x_8 is a border point. Because x_5 is already a core point of C_2, x_8 is classified as C_2.

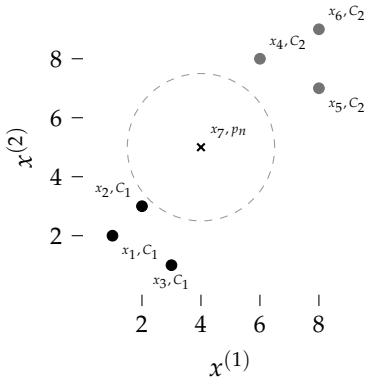


Figure 58:

VISIT x_7: its two nearest neighbours are x_2 at $d\approx 2.83$ and x_4 at $d\approx 3.61$, both beyond $\varepsilon=2.5$. So $N_{2.5}(x_7)=\{x_7\}$, $|N|=1 < 2$, and x_7 is labelled p_n.

All points visited. Final result: $C_1=\{x_1, x_2, x_3\}$, $C_2=\{x_4, x_5, x_6\}$, and x_7 left as noise. No K was chosen; the noise point was never forced into a cluster.

Hyperparameter Selection

CHOOSING $n_{\min}$ follows a practical rule of thumb: set $n_{\min} > d+1$ where d is the dimension of the feature space. For two-dimensional data, $n_{\min}=3$ or 4 is typical; higher dimensions require larger values to keep sparse neighbourhoods from being treated as dense.

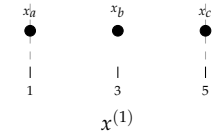


Figure 59: How many points do you need at least to fence in a point in 2D? 3D?

CHOOSING ε uses the k-distance plot: for each point, compute the distance $d_k(x_i)$ to its k-th nearest neighbour at $k=n_{\min}-1$. The $(n_{\min}-1)$-th nearest neighbour distance is the tightest radius that still qualifies.

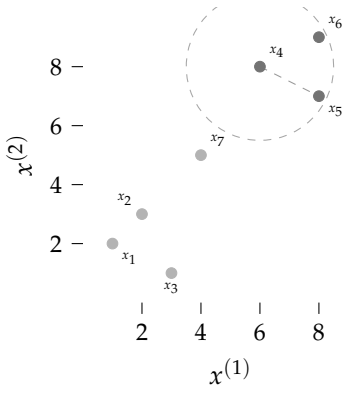


Figure 60:

k-TH NEAREST NEIGHBOUR OF x_4: $n_{\min}=2$, $k=n_{\min}-1=1$. The dashed circle shows the $\varepsilon=2.5$ neighbourhood; the nearest neighbour of x_4 is x_5 at distance $d_k(x_4)=\sqrt{5}\approx 2.24$, well inside ε.

and sort these distances in ascending order,

$$d_k^{(1)} \leq d_k^{(2)} \leq \ldots \leq d_k^{(n)}.$$

The curve rises slowly in dense regions and sharply in sparse ones; the elbow marks the transition and is the recommended ε. Notice, that the sorting does not reflect cluster membership: points from different clusters can be interleaved in the k-distance plot.

SENSITIVITY SWEEP. Try a few $n_{\min}$ values ($d+1$, $2d$, etc.) and compare the resulting k-distance plots. The value that gives the sharpest, most unambiguous elbow is usually the right one.

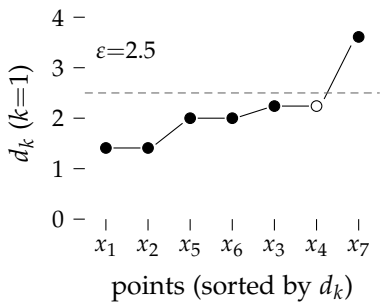
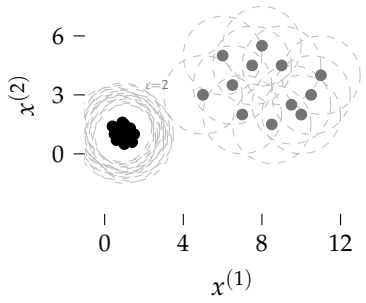


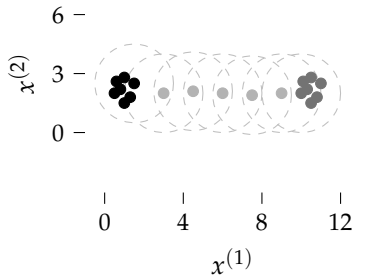
Figure 61: k-distance plot for the seven canonical points with $k{=}n_{\min}{-}1{=}1$. Each $d_k^{(i)}$ is the distance from one point to its nearest neighbour, sorted ascending. The first six values sit at $\sqrt{2}{\approx}1.41$, 2.00, or $\sqrt{5}{\approx}2.24$; $d_k^{(7)}$ jumps to $\sqrt{13}{\approx}3.61$ (the lone x_7). The unfilled dot marks the elbow, and any ε in the gap, such as $\varepsilon{=}2.5$ (dashed), separates the dense six from the sparse outlier.

When DB Scan Fails

DB Scan has its own pathologies. Every density-based method sets a single threshold. The most common arise from the single global ε, which filters or bridges. Then we have the curse of dimensionality, which makes all points look far apart and destroys the notion of density.



Varying Density
Figure 62: Varying cluster density. An ε tight enough to resolve the left cluster leaves the right cluster disconnected; an ε large enough to connect the right cluster merges everything on the left into one blob, loosing expressiveness of the clustering. No single global threshold works for both.



Bridging and Chaining
Figure 63: Bridging: a thin chain of p_c-eligible points (light) spans the gap between two well-separated clusters. The overlapping ε-circles show how density reachability chains through p_c's, fusing both into one cluster. Increasing $n_{\min}$ or decreasing ε breaks the bridge at the cost of shrinking the clusters themselves.

OPTICS or Ordering Points To Identify the Clustering Structure [10] introduces two per-point distances. The core distance ε_c is the per-

[10] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999. DOI: 10.1145/304182.304187. https://scholar.google.com/scholar?q=Ankerst+Breunig+Kriegel+Sander+1999+OPTICS+ordering+points

point analogue of the global ε, the smallest radius that captures $n_{\min}$ neighbours:

$$\varepsilon_c(x_i) = \min\{r : |N_r(x_i)| \geq n_{\min}\}.$$

The reachability distance $r(x_i, x_j)$ is defined for every pair where x_i is an already-visited p_c and x_j is an unvisited point. It measures the cost for x_i to absorb x_j, and is at least $\varepsilon_c(x_i)$, even if x_j sits closer:

$$r(x_i, x_j) = \max\{\varepsilon_c(x_i),\ d(x_i, x_j)\}.$$

Dense regions have small ε_c, sparse regions large ones. The max ensures that a point is never reached at a distance shorter than its ε_c, so reachability adapts to local density automatically.

HDBSCAN or Hierarchical DBSCAN [11] replaces the Euclidean distance with a mutual reachability distance:

$$d_{\text{mreach}}(x_i, x_j) = \max\{\varepsilon_c(x_i),\ \varepsilon_c(x_j),\ d(x_i, x_j)\}.$$

This spreads apart points in sparse regions; exactly those that should not be merged early. From here HDBSCAN builds a full density hierarchy by applying single-linkage clustering to the mutual reachability distance, and potentially condenses it into a flat clustering by selecting the most persistent clusters across all density levels.

[11] R. J. G. B. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 160–172, 2013. https://scholar.google.com/scholar?q=Campello+Moulavi+Sander+2013+density-based+clustering+hierarchical+density+estimates

THE ONLY PARAMETER the user must set is $n_{\min}$, and the result is often robust to its choice.

DENSITY HIERARCHIES AND DENDROGRAMS. The density language and the hierarchical-clustering language describe the same structure from different angles.

THE CURSE OF DIMENSIONALITY: All distance-based methods share a deeper vulnerability as the number of features d grows, the volume of the space increases so fast that a fixed number of points N becomes sparse everywhere.

CONSIDER N POINTS drawn uniformly in the unit hypercube $[0,1]^d$. The mean nearest-neighbour distance scales as

$$d_{\text{NN}} \ \propto\ N^{-1/d}.$$

Fix N=10 and watch the spacing grow with d:

$$\begin{aligned} d = 1: &\quad 10^{-1/1} = 0.10 \\ d = 2: &\quad 10^{-1/2} \approx 0.32 \\ d = 3: &\quad 10^{-1/3} \approx 0.46 \\ d = 10: &\quad 10^{-1/10} \approx 0.79 \\ d = 50: &\quad 10^{-1/50} \approx 0.96 \end{aligned}$$

By d=50 every point is nearly as far from its neighbour as from the boundary of the cube. The points have not moved apart; the space has grown underneath them.

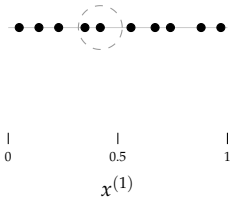


Figure 64: N=10 points in $[0,1]^1$: mean NN spacing $\approx 10^{-1}$=0.1. The line is well covered.

$\propto$ means "proportional to": $a \propto b$ says $a = c \cdot b$ for some constant c independent of the variables of interest.

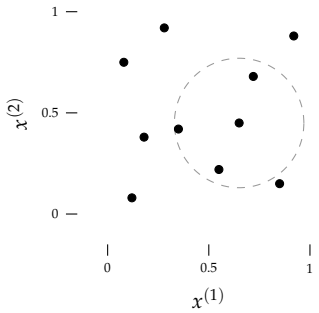


Figure 65: Same N=10 points in $[0,1]^2$: the space has grown from a line to a square, and the mean NN spacing rises to $10^{-1/2} \approx 0.32$.

FOR DISTANCE-BASED METHODS THIS IS FATAL. The ε-neighbourhood of a point captures a hypersphere whose volume shrinks relative to the cube: $V_{\text{sphere}} / V_{\text{cube}} \to 0$ as d grows. A radius that includes many neighbours in two dimensions encloses almost nothing in fifty. Either ε must grow until every point is a neighbour, collapsing all structure into one cluster, or it stays small and every point is labelled p_n.

Examples & Exercises

COMPUTING BY HAND reveals what the algorithm actually does at each step; run it slowly and feel where density propagates and where it stops.

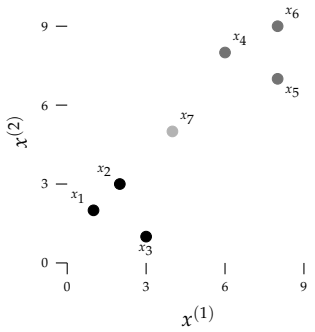


Figure 66: Seven points for the DBSCAN trace. x_7=(4,5) sits between the two natural groups.

GIVEN SEVEN POINTS x_1=(1,2), x_2=(2,3), x_3=(3,1), x_4=(6,8), x_5=(8,7), x_6=(8,9), x_7=(4,5), run DBSCAN with ε=2.5 and $n_{\min}$=2. Your tasks: compute the neighbourhood size $|N_{2.5}(x_i)|$ for each point, classify each point as p_c, p_b, or p_n, and identify the clusters.

NEIGHBOURHOOD SIZES. For each point x_i count all points within distance 2.5 (including x_i itself). Here is the full calculation for x_1:

$$
\begin{aligned}
d(x_1, x_2) &= \sqrt{(2-1)^2 + (3-2)^2} \\
&= \sqrt{1+1} \\
&= \sqrt{2} \\
&\approx 1.41 \quad (\leq 2.5, \text{ counted}) \\
d(x_1, x_3) &= \sqrt{(3-1)^2 + (1-2)^2} \\
&= \sqrt{4+1} \\
&= \sqrt{5} \\
&\approx 2.24 \quad (\leq 2.5, \text{ counted}) \\
d(x_1, x_7) &= \sqrt{(4-1)^2 + (5-2)^2} \\
&= \sqrt{9+9} \\
&= \sqrt{18} \\
&\approx 4.24 \quad (> 2.5, \text{ not counted})
\end{aligned}
$$

So $N_{2.5}(x_1) = \{x_1, x_2, x_3\}$ and $|N_{2.5}(x_1)| = 3 \geq 2$: x_1 is a p_c. Compute the remaining neighbourhood sizes yourself, then verify against the table below.

VERIFY x_7 IS p_n. Compute $d(x_7, x_3)$ to confirm that x_7 lies outside

Point	Coords	$\lvert N_{2.5}\rvert$	Type	Neighbours in $N_{2.5}$	Cluster
x_1	$(1,2)$	3	p_c	$\{x_1, x_2, x_3\}$	C_1
x_2	$(2,3)$	3	p_c	$\{x_1, x_2, x_3\}$	C_1
x_3	$(3,1)$	3	p_c	$\{x_1, x_2, x_3\}$	C_1
x_4	$(6,8)$	3	p_c	$\{x_4, x_5, x_6\}$	C_2
x_5	$(8,7)$	3	p_c	$\{x_4, x_5, x_6\}$	C_2
x_6	$(8,9)$	3	p_c	$\{x_4, x_5, x_6\}$	C_2
x_7	$(4,5)$	1	p_n	$\{x_7\}$ only	–

Table 5: DBSCAN trace for ε=2.5, $n_{\min}$=2. All six canonical points are p_c's; x_7 has no neighbour within radius 2.5 and is labelled p_n. The two natural clusters are recovered without specifying K and without assigning p_n to either group.

the ε-neighbourhood of its nearest candidate:

$$\begin{aligned} d(x_7, x_3) &= \sqrt{(4-3)^2 + (5-1)^2} \\ &= \sqrt{1+16} \\ &= \sqrt{17} \\ &\approx 4.12 \quad (> 2.5) \end{aligned}$$

Compute $d(x_7, x_4)$ yourself. Why is x_7 correctly identified as p_n rather than being assigned to whichever cluster is nearer?

ADDING A BORDER POINT. Place an eighth point x_8=(8, 5.5) into the dataset and raise the density threshold to $n_{\min}$=3. Compute $\lvert N_{2.5}(x_8)\rvert$ and classify x_8 as p_c, p_b, or p_n. Which cluster, if any, does x_8 join, and why? Verify that the six original cluster members remain p_c's under the new threshold.

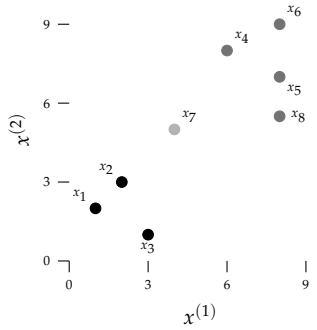


Figure 67: Eight points for the border-point exercise. x_8=(8, 5.5) sits below C_2.

DENSITY REACHABILITY FROM x_8. Still using the eight-point dataset with ε=2.5 and $n_{\min}$=3: is x_8 density-reachable from x_4? If so, write down the chain of points that connects them. Now reverse the question: is x_4 density-reachable from x_8? Why or why not?

PARAMETER SENSITIVITY. Re-run the trace with ε=4.5 and $n_{\min}$=2. Does x_7 change classification? At what minimum value of ε does x_7 first enter the neighbourhood of some p_c?

BUILDING THE k-DISTANCE PLOT. Return to the original seven points with $n_{\min}$=2, so k=$n_{\min}$−1=1. For each point x_i, compute

the distance to its nearest neighbour $d_1(x_i)$. Here is x_5:

$$\begin{aligned} d(x_5,\, x_4) &= \sqrt{(8-6)^2 + (7-8)^2} \\ &= \sqrt{5} \\ &\approx 2.24 \\ d(x_5,\, x_6) &= \sqrt{(8-8)^2 + (9-7)^2} \\ &= \sqrt{4} \\ &= 2.00 \\ d_1(x_5) &= 2.00 \quad \text{(nearest neighbour is } x_6\text{)} \end{aligned}$$

Compute $d_1(x_i)$ for the remaining six points. Sort all seven values in ascending order and sketch the curve. Where is the elbow? Does the ε=2.5 from the earlier trace sit in the gap?

Comparison with K-Means. Run K-Means with K=2 on the same seven points. Which cluster does K-Means assign x_7 to? Is that assignment more or less informative than DBSCAN's p_n label, and why?

K-Means baseline result. After convergence with K=2, the centroids settle at $\mu_1 \approx (2.0,\, 2.0)$ and $\mu_2 \approx (7.3,\, 8.0)$. Point x_7=$(4, 5)$ is closer to μ_1:

$$\begin{aligned} d(x_7,\, \mu_1) &= \sqrt{(4-2)^2 + (5-2)^2} \\ &= \sqrt{4+9} \\ &= \sqrt{13} \approx 3.61 \\ d(x_7,\, \mu_2) &= \sqrt{(4-7.3)^2 + (5-8.0)^2} \\ &= \sqrt{10.89+9} \\ &= \sqrt{19.89} \approx 4.46 \end{aligned}$$

K-Means assigns x_7 to C_1, as shown below. The assignment is not wrong in a strict sense; it is simply forced. K-Means has no p_n concept and cannot abstain.

From global to local density. Return to the eight-point dataset ($x_1, \ldots, x_7$ plus x_8=(8, 5.5)) with $n_{\min}$=3. For each point x_i, compute the core distance $\varepsilon_c(x_i)$: the smallest radius that captures at least $n_{\min}$ points, itself included. Here is x_5:

The core distance $\varepsilon_c(x_5)$=2.00 is the radius needed to capture $\{x_8,\, x_6,\, x_5\}$: exactly three points.

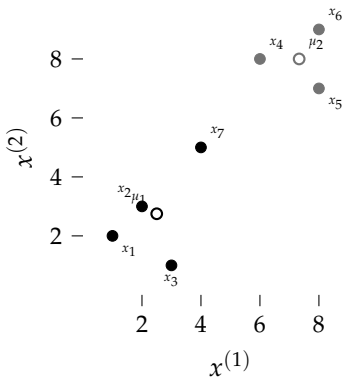


Figure 68: K-Means baseline with K=2. o-marks for centroids μ_1 and μ_2. Because K-Means must assign every point, x_7 is forced into C_1 (dark). Compare with Figure 58: DBSCAN labels x_7 as p_n and makes no forced assignment.

$$
\begin{aligned}
d(x_5,\, x_8) &= 1.50 \\
d(x_5,\, x_6) &= 2.00 \\
d(x_5,\, x_4) &= \sqrt{5} \\
&\approx 2.24 \\
\varepsilon_c(x_5) &= 2.00
\end{aligned}
$$

Compute ε_c for x_1 through x_4, x_6, x_7, and x_8. Which points have the largest core distance, and what does that tell you about their local density?

Now compute the mutual reachability distance for two pairs:

$$
\begin{aligned}
d_{\text{mreach}}(x_4,\, x_5) &= \max\{\varepsilon_c(x_4),\, \varepsilon_c(x_5),\, d(x_4,\, x_5)\} \\
d_{\text{mreach}}(x_7,\, x_2) &= \max\{\varepsilon_c(x_7),\, \varepsilon_c(x_2),\, d(x_7,\, x_2)\}
\end{aligned}
$$

Compare each to the raw Euclidean distance. For which pair does d_{mreach} differ from d, and why? Why does HDBSCAN not need a global ε?

THE CURSE OF DIMENSIONALITY BY THE NUMBERS. The mean nearest-neighbour spacing scales as $d_{\text{NN}} \propto N^{-1/d}$. Compute $N^{-1/d}$ for N=100 at $d = 2,\ 10,\ 50,\ 100$. At what dimension d does the spacing first exceed 0.9? If you tried to run DBSCAN at that dimension, what would happen to the ε-neighbourhoods?

BACK AT THE COFFEETABLE CODE. Same 1 797 aroma fingerprints, two coordinates. Implement epsilon-neighborhoods and the core-point classifier, use the k-distance plot to pick ε, then compare DBSCAN with K-Means and Ward at the same cluster count.

CODE will be provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

WHAT TO OBSERVE. Core points fill the dense hearts of each variety;

noise points land in the gaps no method should claim. DBSCAN follows density contours rather than drawing spherical cuts, so it agrees with K-Means in the cores but diverges at irregular boundaries. A single ε cannot capture clusters of very different densities. In next steps, learn about sensitivity sweeps and HDBSCAN to address this.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What does density reachability gain over distance-to-centroid? Which K-Means failures disappear, which new ones appear?
- Why is density reachability asymmetric, and what does that imply for the determinism of p_b labels?
- A p_b sits inside two p_c's from different clusters. Is its label unique? What does that say about what a DB Scan cluster really is?
- Why is bridging the density-based dual of single-linkage chaining, and why does every remedy cost something?
- The k-distance plot shows a straight line with no elbow. What does this tell you, and which method would you try next?
- What is the essential difference between an OPTICS reachability plot and an HDBSCAN density hierarchy?
- When would you prefer DB Scan over Ward's method even if both produce a sensible partition?

RECAP of Key Concepts:

- DB Scan clusters are maximal sets of density-connected points; no centroid, no pre-specified K
- Every point is p_c when $|N_\varepsilon| \geq n_{\min}$, p_b when inside a p_c's neighbourhood without being p_c itself, or p_n
- Density reachability is transitive along p_c's but not symmetric; p_b labels depend on visit order
- The partition is determined by $(X, \varepsilon, n_{\min})$ up to shared p_b's; $O(n)$ queries, no restarts
- The k-distance plot with $k{=}n_{\min}{-}1$, sorted ascending, chooses ε at the elbow

- Failure modes: varying density, where no global ε fits two scales, and high dimensionality, where distances concentrate
- Bridging fuses clusters through chains of p_c's, the density dual of single-linkage chaining
- OPTICS exposes all density scales via a reachability plot; HDBSCAN extracts the most persistent clusters from a full density hierarchy

TWO WAYS OUT OF THE CURSE OF DIMENSIONALITY. Either we collect exponentially more data [12], which is rarely possible, or we reduce the dimension so that the samples we already have sit densely in the space we actually care about. The observation that makes the second route tractable is that real datasets almost never use their ambient dimensions fully: the meaningful variation lives on a much lower-dimensional subset, and the rest is redundancy, correlation, or noise. Finding that signal-carrying subset is what the next part of these notes is about.

UP NEXT. Part II, Representing Pattern, leaves the original feature space behind and learns to construct new coordinates. Chapter 5 opens with principal component analysis as the linear baseline. Only once a dataset is described in the dimensions that carry signal do the clustering methods of Part I regain their footing.

[12] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2): 8–12, 2009. DOI: 10.1109/MIS.2009.36. https://scholar.google.com/scholar?q=Halevy+Norvig+Pereira+2009+unreasonable+effectiveness+of+data

TEASER. The curse of dimensionality breaks every method in this chapter, yet real data rarely uses all its ambient dimensions. Can we find the directions that carry the most signal and build a lower-dimensional representation around them, so that the clustering methods of Part I regain their footing?

FEEDBACK.

Part II

Representing Pattern

Dimensionality Reduction

2026-05-06 · cheerful mango Haubentaucher

Folding and Unfolding Space.

The Why

Dimensionality reduction finds those factors and discards the rest.

MNIST handwritten digits , for example, live in $\mathbb{R}^{784}$ (28×28 pixels), but vary along a handful of latent factors such as shapes or geometric primitives. The intrinsic dimensionality is estimated at roughly $k \approx 12$ to 14.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. DOI: 10.1109/5.726791. https://scholar.google.com/scholar?q=LeCun+Bottou+Bengio+Haffner+1998+gradient-based+learning+document+recognition

Real-world data $\mathbf{X} \in \mathbb{R}^{n \times d}$ is often high-dimensional: images have millions of pixels, genomes have billions of base pairs, text embeddings have thousands of dimensions. Every clustering method in Part I takes such a feature space as given and works within it. But while dimensionality comes at a cost and degrades our approaches, dimensionality carries information. The question then becomes whether a more compact representation could carry the same information.

Dimensionality has a price. In low dimensions, some points are close and others are far away; that contrast is what makes clustering work. As d grows, the contrast disappears: all pairwise distances converge toward the same value.

Computational cost grows with d, too. Distance computations scale as $O(nd)$, covariance matrices require $O(d^2)$ storage, and eigendecompositions cost $O(d^3)$. Reducing from d to k before fitting a downstream model can turn an intractable pipeline into a practical one.

Expressiveness of the feature space. Original features are often redundant or entangled. If two features $x^{(i)}$ and $x^{(j)}$ are highly correlated ($|\rho_{ij}| \approx 1$), they carry nearly the same information; a good representation replaces them with a single component that captures their shared variance.

Interpretability benefits directly. Humans usually usually do

not think and reason in high-dimensional vector spaces; we prefer low-dimensional, disentangled representations that align with our concepts of space.

IN ORDER TO MOVE FROM HIGH-DIMENSIONAL OBSERVATIONS TO COMPACT, SIGNAL-CARRYING REPRESENTATIONS we have good reason to find ways to,

- find a mapping that retains the pattern that carries the signal.
- construct proxy dimensions that are not limited to the original features, so that hidden patterns become apparent.
- understand what different notions of preserving pattern imply, and choose the right one for a given task.

Hands-On Experience

CONSIDER the canonical six points, but now arranged so that $x^{(1)}$ varies only slightly while $x^{(2)}$ carries nearly all the spread.

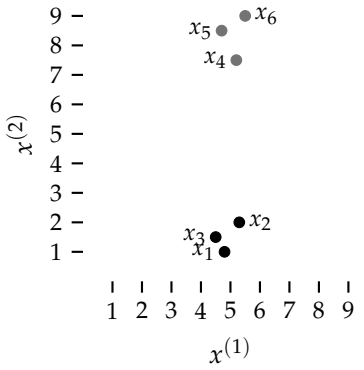


Figure 69: The canonical six points, repositioned so that $x^{(1)}$ barely varies while $x^{(2)}$ carries both the cluster separation and the within-cluster spread. Two features describe each point, but the data is essentially one-dimensional.

- If you had to keep only one of the two features, which would you choose?
- How much information would you lose?
- And could you recover the two clusters from that single feature alone?

THE TWO CLUSTERS are clearly visible, and one feature carries almost all the signal. The first feature $x^{(1)}$ hovers around 5 for every point; dropping it loses almost nothing. The second feature $x^{(2)}$ alone separates the two groups and captures the within-cluster variation.

THE METHODS IN THIS CHAPTER AUTOMATE THAT JUDGMENT OF DIMENSIONALITY REDUCTION. What is the most efficient mapping you can find for the MNIST dataset, using only geometric primitives such as circles, lines so that each class is well-separated?

HAND-CRAFTED FEATURE ENGINEERING is dimensionality reduction by another name. When a domain expert selects "age" and "income" from a database with hundreds of columns, or computes edge gradients from raw pixels, they are projecting from $\mathbb{R}^d$ to $\mathbb{R}^k$ using human genius.

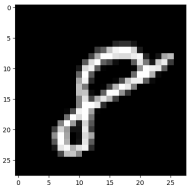


Figure 70: A single MNIST digit its meaningful variation spans few dimensions.

A WELL-CHOSEN MAPPING $f\colon \mathbb{R}^d \to \mathbb{R}^k$ disentangles latent factors, yielding coordinates that are more discriminative for downstream tasks than the originals.

THE LEARNING OBJECTIVES of this lecture:

- Understand naive feature selection based on variance and its limitations.
- Understand the PCA objective and its connection to variance maximization and the singular value decomposition.
- Apply t-SNE and UMAP for visualization and avoid common misinterpretations of their output.
- Choose appropriate dimensionality reduction methods based on linearity, dataset scale, and whether the goal is preprocessing or visualization.

The Curse of Dimensionality, Revisited

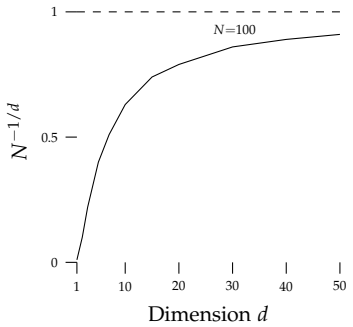


Figure 71: Mean nearest-neighbour spacing $d_{\mathrm{NN}} \propto N^{-1/d}$ for $N{=}100$ points. By $d{=}50$ every point is nearly as far from its neighbour as from the boundary. But if the data only occupies $k \ll d$ effective dimensions, the curse applies to k, not d.

THE KEY OBSERVATION is that high d is only fatal when all d dimensions carry independent information. In practice they rarely do. Features are correlated, redundant, or simply noisy. The effective degrees of freedom, the intrinsic dimensionality k, is often far smaller than d.

REDUCING d TO k REVERSES THE CURSE. The spacing, the distance concentration, the density collapse all depend on the dimension of the space the data actually lives in. If we project from $\mathbb{R}^d$ down to $\mathbb{R}^k$, the clustering methods from Part I regain their footing: distances become discriminative again, ε-neighbourhoods fill up, and centroids sit among their members.

THE QUESTION IS HOW TO FIND THE RIGHT k DIRECTIONS. Picking original features by hand (as in variance-based selection) works when the axes happen to align with the signal. When they do not, we need methods that construct new directions from the data itself. That is what PCA, and later t-SNE and UMAP, will do.

THE BLESSING OF DIMENSIONALITY. The Johnson-Lindenstrauss lemma shows that n points in high dimensions can be projected to $O(\log n/\epsilon^2)$ dimensions while preserving pairwise distances within $(1\pm\epsilon)$. Even random projections work.

W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984. https://scholar.google.com/scholar?q=Johnson+Lindenstrauss+1984+extensions+lipschitz+mappings+hilbert+space

A Naïve Baseline: Variance-Based Feature Selection

WHEN AXES ALIGN WITH THE SIGNAL consider the most straightforward way to reduce dimensions: measure the variance of each original feature independently and keep only those with the highest variance.
Given data $X \in \mathbb{R}^{n\times d}$, compute the variance along each coordinate axis:

$$\mathrm{Var}(x^{(j)}) = \frac{1}{n}\sum_{i=1}^{n}\big(x_i^{(j)} - \bar{x}^{(j)}\big)^2, \quad j = 1,\dots,d$$

Sort the features by decreasing variance, keep the top k, and discard the rest. This is pure feature selection: no new axes are created, only existing ones are retained or dropped.

FEATURE SELECTION falls into three families: *filter* methods score features independently of any model (e.g. mutual information, variance thresholding), *wrapper* methods evaluate subsets by training a predictor, and *embedded* methods perform selection inside the learning objective (e.g. ℓ_1 regularisation). Wrapper and embedded methods require labels; filters may or may not. Without labels we are limited to unsupervised proxies such as variance.

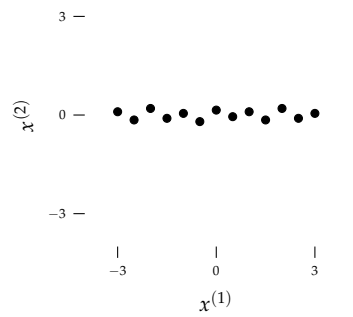


Figure 72: Data spread along $x^{(1)}$ but nearly constant in $x^{(2)}$. Dropping $x^{(2)}$ loses almost nothing.

WHEN DOES THIS WORK? If the coordinate axes already align with the directions of meaningful variation, variance-based selection is fast, interpretable, and surprisingly effective. A near-constant feature carries almost no information, and removing it loses little.

When does it fail? When the signal does not happen to align with the axis, for example when it runs diagonally across the coordinate axes. In that case, no single original feature captures it well.

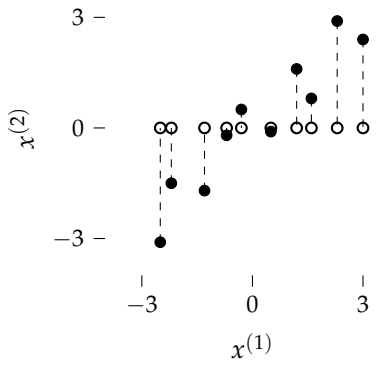


Figure 73: Naïve feature selection: dropping $x^{(2)}$ projects onto the horizontal axis. The dashed errors are large because the data varies along $y{=}x$, not along a coordinate axis. Both features have nearly equal variance, so the naïve ranking cannot decide which to keep. Notice, how in this case both features would still preserver the signal.

Principal Component Analysis

Here both features have nearly the same variance, so variance-based selection cannot decide which to keep. Dropping either axis incurs large reconstruction error. The real pattern lies along the diagonal $y{=}x$, a direction that no original feature represents. When the interesting variation runs between features, we need to construct and map to new dimensions, meaning features. That is exactly what PCA does.

The manifold hypothesis holds that high-dimensional data concentrates on a k-dimensional manifold $\mathcal{M} \subset \mathbb{R}^d$ with $k \ll d$. The goal of dimensionality reduction is to find a coordinate system for $\mathcal{M}$.

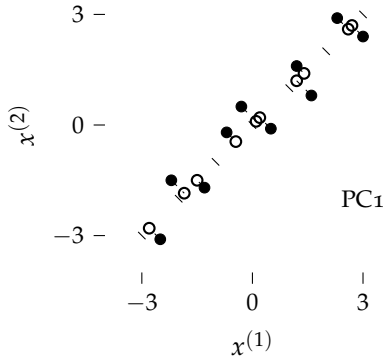


Figure 74: PCA projection onto PC1: The direction of maximum variance runs along $y{=}x$. The reconstruction errors (dashed) are tiny compared to the naïve projection.

Given centred data $X \in \mathbb{R}^{n \times d}$, where rows are samples, PCA finds a k-dimensional subspace, $k < d$, that best represents the data.

PCA finds orthogonal directions of maximum variance in the data.

K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572, 1901. DOI: 10.1080/14786440109462720. https://scholar.google.com/scholar?q=Pearson+1901+lines+planes+closest+fit+systems+points; and H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933. DOI: 10.1037/h0071325. https://scholar.

Algorithm 4: PCA via Eigendecomposition

Require: Centred data $X \in \mathbb{R}^{n \times d}$ (each column has zero mean, so that $X^T X/n$ is the covariance matrix; if features have different scales, standardise first by dividing each column by its standard deviation), target dimension k
Ensure: Principal components $W \in \mathbb{R}^{d \times k}$, projected data $Z \in \mathbb{R}^{n \times k}$
1: Compute covariance matrix $\Sigma = X^T X/n$
2: Compute eigendecomposition $\Sigma = V \Lambda V^T$
3: $W \leftarrow$ first k columns of V (sorted by decreasing eigenvalue)
4: $Z \leftarrow XW$ ▷ Project data onto k-dimensional subspace

THE VARIANCE MAXIMIZATION VIEW. Find the direction $w_1 \in \mathbb{R}^d$ with $\|w_1\|=1$ that maximises the variance of projected data:

$$\begin{aligned} w_1 &= \arg \max_{\|w\|=1} \operatorname{Var}(Xw) \\ &= \arg \max_{\|w\|=1} w^T \Sigma\, w, \\ \Sigma &= \tfrac{1}{n} X^T X. \end{aligned}$$

LAGRANGE MULTIPLIERS. Maximising $w^T \Sigma w$ subject to $w^T w = 1$ is a constrained optimisation problem. Introduce a multiplier λ and form the Lagrangian:

$$\mathcal{L}(w, \lambda) = w^T \Sigma\, w \; - \; \lambda\, (w^T w - 1)$$

Differentiating with respect to w and setting the gradient to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= 2\, \Sigma\, w \; - \; 2\, \lambda\, w \\ &\stackrel{!}{=} \; 0 \end{aligned}$$

which gives the eigenvalue equation:

$$\Sigma\, w = \lambda\, w$$

So w must be an eigenvector of Σ. The projected variance along any such eigenvector equals its eigenvalue:

$$\begin{aligned} w^T \Sigma\, w &= w^T (\lambda\, w) \\ &= \lambda\, w^T w \\ &= \lambda \end{aligned}$$

Maximising the variance therefore means choosing w_1 as the eigenvector with the largest eigenvalue λ_1. The second principal component is the eigenvector with the second-largest eigenvalue, orthogonal to the first, and so on. The first k principal components together define the optimal k-dimensional subspace. $k \leq d$ is a free choice.

ORTHOGONALITY of principal components follows directly: eigenvectors of a symmetric matrix corresponding to distinct eigenvalues are orthogonal.

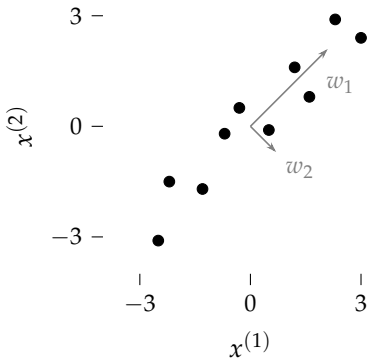


Figure 75: The two eigenvectors of Σ. w_1 points along the direction of maximum variance; w_2 is orthogonal and captures the residual spread.

COMPUTING EIGENVALUES. Eigenvalues satisfy the characteristic equation $\det(\Sigma - \lambda I) = 0$. For a 2×2 covariance matrix $\Sigma = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$:

$$\begin{aligned} \det(\Sigma - \lambda I) &= \det \begin{pmatrix} a-\lambda & b \\ b & c-\lambda \end{pmatrix} \\ &= (a-\lambda)(c-\lambda) - b^2 \\ &= \lambda^2 - (a+c)\,\lambda + (ac - b^2) \\ &\stackrel{!}{=} 0 \end{aligned}$$

GEOMETRIC INTUITION. Σ stretches space along its eigenvector directions. Subtracting λI weakens every direction by λ. When λ exactly matches the stretch along some eigenvector, $\Sigma - \lambda I$ collapses that direction to zero, the matrix becomes singular, and $\det = 0$. So the characteristic equation asks: at which scaling factors λ does Σ lose a dimension?

Applying the quadratic formula:

$$\begin{aligned} \lambda &= \frac{(a+c) \pm \sqrt{(a+c)^2 - 4(ac - b^2)}}{2} \\ &= \frac{(a+c) \pm \sqrt{(a-c)^2 + 4b^2}}{2} \end{aligned}$$

COMPUTING EIGENVECTORS. For each eigenvalue λ_i, solve $(\Sigma - \lambda_i I)\, w = 0$. From the first row of the 2×2 system:

$$\begin{aligned} (a - \lambda_i)\, w^{(1)} + b\, w^{(2)} &= 0 \\ w^{(2)} &= -\frac{a - \lambda_i}{b}\, w^{(1)} \end{aligned}$$

Any non-zero solution gives the direction; normalise to $\|w_i\|=1$ to obtain the principal component.

A SIMPLE DIAGNOSTIC reveals whether the data is compressible. Inspect the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$ of Σ. If most are near zero, the data varies in only a few directions; the remaining dimensions add volume but no information.

The scree plot (Figure 76) visualises exactly this: a steep drop tells you how many components matter.

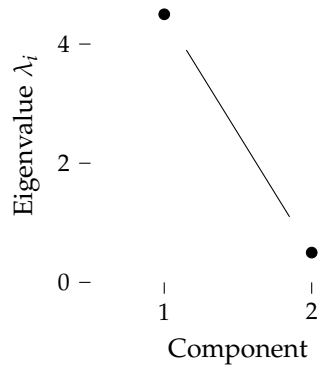


Figure 76: Scree plot for the two-component example (λ_1=4.5, λ_2=0.5). The sharp drop confirms that one direction captures most of the variance.

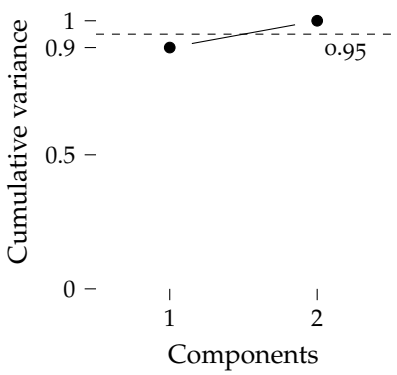


Figure 77: Cumulative explained variance: the fraction of total variance captured by the first k components is $\sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{d} \lambda_i$. Here PC1 alone explains 90%; both components are needed to exceed the rule of thumb 95% threshold (dashed).

Reconstruction Error

So far we derived PCA from variance maximisation: find the directions along which the projected data spreads the most. There is a dual view that asks the opposite question: if we compress from d dimensions to k and then decompress back to d, how much signal do we lose?

Given centred data $X \in \mathbb{R}^{n \times d}$ and an orthonormal basis $W \in \mathbb{R}^{d \times k}$ ($W^T W = I_k$), each point x_i is projected to $z_i = W^T x_i \in \mathbb{R}^k$ and reconstructed as $\hat{x}_i = W z_i = W W^T x_i$. The reconstruction error is the squared distance between each point and its reconstruction:

$$\begin{aligned} \mathcal{L}(R &= \sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2 \\ &= \sum_{i=1}^{n} \|x_i - W W^T x_i\|^2 \end{aligned}$$

The dashed error segments are short because PC1 captures the dominant direction of variation, making x and its reconstruction similar.

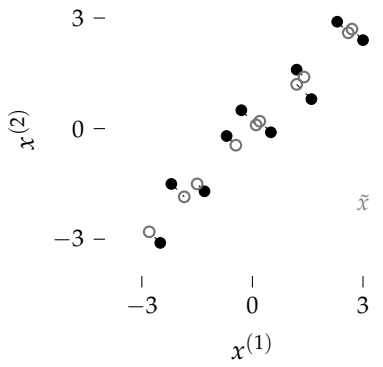


Figure 78: Reconstruction error of PCA with k=1. Each point (filled) is projected onto PC1 (the line y=x) to get its reconstruction (open circle). The dashed segments are the reconstruction errors: the perpendicular distances that PCA minimises. Compare with Figure 73, where projection onto a coordinate axis leaves much larger errors.

For $d > 2$, closed-form eigendecompositions do not exist. In practice, use the SVD: $X = U \Sigma V^T$, where the columns of V are the eigenvectors of $X^T X$ with $\lambda_i = \sigma_i^2 / n$. This avoids forming $X^T X$ explicitly and is numerically stable.

Independent Component Analysis

PCA decorrelates but does not separate sources. Uncorrelated ($\mathrm{Cov}(s_i, s_j)$=0) is weaker than independent ($p(s_i, s_j) = p(s_i)\, p(s_j)$). Given a linear mixture $X = AS$ of independent signals S, ICA recovers $S = A^{-1} X$ by maximising non-Gaussianity of the components.

t-SNE[13]

[13] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. https://scholar.google.com/scholar?q=van+der+Maaten+Hinton+2008+visualizing+data+t-SNE

T-SNE. stands for t-distributed Stochastic Neighbour Embedding. It is optimised for one-, two- or three-dimensional visualization. t-SNE preserves local pattern; nearby points in high dimensions stay nearby in the embedding. It does this in four steps.

Algorithm 5: t-SNE. All points are updated simultaneously (batch gradient descent). The KL objective is non-convex, so there is no convergence guarantee; different random initialisations can yield different layouts.

Require: High-dimensional data X, perplexity, target dimension k

1: Compute pairwise similarities in high-D (Gaussian kernel, maps distances to $(0, 1]$):

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{l\neq i}\exp(-\|x_i - x_l\|^2/2\sigma_i^2)}$$

Symmetrize: $p_{ij} = (p_{j|i} + p_{i|j})/2n$

2: Initialize low-D embedding Y

3: **repeat**

4: Compute low-D similarities with Student-t kernel:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

5: Gradient step: minimise

$$\mathrm{KL}(P\|Q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

6: **until** convergence

PCA AND ICA ARE LINEAR. No rotation of the axes can untangle classes that live on a curved manifold. When the goal is visualization rather than preprocessing, we need methods that can bend the coordinate system to follow the data. This is what t-SNE and UMAP provide.

PERPLEXITY controls the effective number of neighbours each point considers. It sets the bandwidth σ_i of the Gaussian kernel. Typical values: 5 to 50. Higher perplexity means more global pattern is considered.

PERPLEXITY is defined as $2^{H(P_i)}$ where H is the Shannon entropy of the conditional distribution over neighbours. Think of it as the effective number of neighbours.

THE GRADIENT of $\mathrm{KL}(P\|Q)$ with respect to each embedding coordinate is

$$\frac{\partial\,\mathrm{KL}}{\partial\, y_i} = 4\sum_{j\neq i}(p_{ij} - q_{ij})\,(y_i - y_j)\,\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

THE KL DIVERGENCE $\mathrm{KL}(P\|Q) = \sum_{ij} p_{ij}\log(p_{ij}/q_{ij})$ measures how much the low-dimensional distribution Q diverges from the high-dimensional distribution P. It is zero when $Q = P$ and asymmetric: misrepresenting a large p_{ij} with a small q_{ij} is penalised heavily, so t-SNE prioritises preserving local neighbourhoods.

Each summand acts as a pairwise force: where $p_{ij} > q_{ij}$ the force is attractive, where $p_{ij} < q_{ij}$ it is repulsive. The heavy tails of the Student-t kernel give distant pairs in Y a larger q_{ij} than a Gaussian would, so the repulsive force saturates. This prevents the crowding

problem where all points collapse into the centre of the map. Gradient descent moves the embedding until local neighbourhoods in Y match those in X.

TRACE the four steps on our canonical six points, mapping from $\mathbb{R}^2$ to $\mathbb{R}^1$ with perplexity=2.

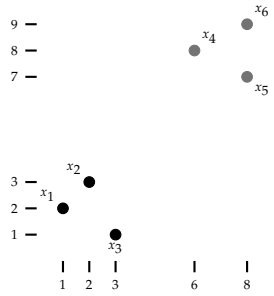
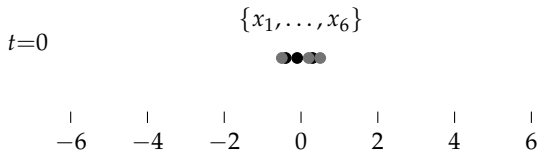
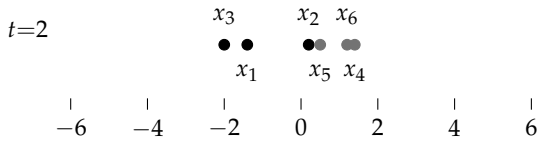


Figure 79: The canonical six points (again)). Cluster A: x_1=(1, 2), x_2=(2, 3), x_3=(3, 1). Cluster B: x_4=(6, 8), x_5=(8, 7), x_6=(8, 9).

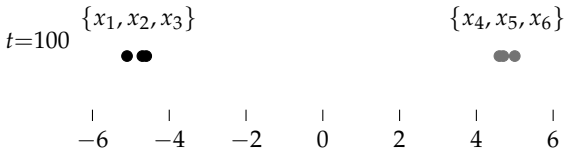
STEP 1: INITIALISE THE EMBEDDING. Each y_i is drawn independently from $\mathcal{N}(0, 10^{-4})$. At t=0 the six points land in near-random order: cluster A (black) and cluster B (grey) are completely intermixed.



STEP 2: COMPUTE HIGH-DIMENSIONAL SIMILARITIES AND BEGIN GRADIENT DESCENT. Here perplexity $=$ 2. Same-cluster pairs get large similarities, cross-cluster pairs near zero; the resulting matrix P is computed once and stays fixed as the target. Gradient descent then pulls high-similarity neighbours together and pushes low-similarity points apart. By t=2 most points drift toward their cluster partners, but x_2; initialised near x_4 (see t=0), needed a few steps for its atractive forces to move through the repulsive forces of the other cluster's points.



STEP 3: CONVERGENCE. At every iteration, q_{ij} is recomputed from the current embedding distances. The forces balance and the embedding stabilises. Within each cluster the ordering mirrors similarity: x_1/x_2 (highest p_{12}=0.098) land closest in A, x_5/x_6 (p_{56}=0.089) in B.



INTERPRETING T-SNE. Cluster sizes and inter-cluster distances are meaningless. t-SNE equalises local densities and preserves only neighbourhood structure. Results vary with the random seed; always run multiple times. Use t-SNE for visualization only, never as preprocessing for downstream models.

UMAP

UMAP stands for Uniform Manifold Approximation and Projection . It has theoretical foundations in Riemannian geometry and algebraic topology.

L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018. https://arxiv.org/abs/1802.03426

UMAP BRINGS THREE PRACTICAL STRENGTHS.

SCALE. UMAP operates on a sparse k-nearest-neighbour graph instead of all n^2 pairs, giving $O(n \log n)$ complexity in practice, large enough for million-point datasets.

GLOBAL STRUCTURE. UMAP optimises a cross-entropy loss that penalises both pulling true neighbours apart *and* pushing true non-neighbours together, so the embedding preserves large-scale arrangement alongside local clusters.

REPRODUCIBILITY. UMAP initialises the embedding with a spectral embedding of the graph Laplacian, so repeated runs on the same data converge to similar layouts rather than producing arbitrary rotations each time.

Algorithm 6: UMAP. Like t-SNE, all points are updated simultaneously and the objective is non-convex. The spectral initialisation makes results more reproducible than t-SNE's random start, but does not guarantee a global optimum. Key parameters: `n_neighbors` (typically 15–200) controls local versus global structure, analogous to perplexity in t-SNE; `min_dist` (typically 0.1–0.5) controls how tightly points cluster in the embedding.

Require: High-dimensional data X, number of neighbours k, target dimension d, minimum distance δ

1: Build a weighted k-nearest-neighbour graph. For each x_i, set $\rho_i = \min_{j \in \mathrm{NN}(i)} \|x_i - x_j\|$ and choose σ_i so that

$$\sum_{j \in \mathrm{NN}(i)} \exp\big(-(\|x_i - x_j\| - \rho_i)/\sigma_i\big) = \log_2 k$$

2: Compute directed edge weights:

$$w_{j|i} = \exp\big(-\max(0,\, \|x_i - x_j\| - \rho_i)/\sigma_i\big)$$

Symmetrise: $w_{ij} = w_{j|i} + w_{i|j} - w_{j|i} \cdot w_{i|j}$

3: Initialise low-D coordinates Y via spectral embedding

4: **repeat**

5: Compute low-D weights:

$$v_{ij} = \big(1 + a\,\|y_i - y_j\|^{2b}\big)^{-1}$$

6: Gradient step: minimise

$$\mathcal{L} = \sum_{ij} \Big[w_{ij} \log \frac{w_{ij}}{v_{ij}} + (1 - w_{ij}) \log \frac{1 - w_{ij}}{1 - v_{ij}} \Big]$$

7: **until** convergence

Examples & Exercises

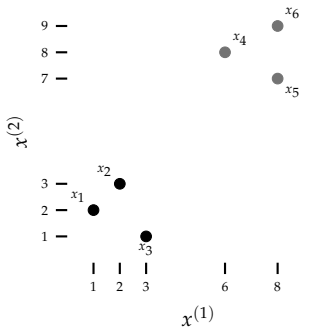


Figure 80: The canonical six points from Chapters 3 and 4. Cluster A: $x_1{=}(1,2)$, $x_2{=}(2,3)$, $x_3{=}(3,1)$. Cluster B: $x_4{=}(6,8)$, $x_5{=}(8,7)$, $x_6{=}(8,9)$.

COMPUTING BY HAND builds the geometric intuition that plotting the output never can. Step through the arithmetic slowly.

GIVEN THE CANONICAL SIX POINTS $x_1{=}(1,2)$, $x_2{=}(2,3)$, $x_3{=}(3,1)$, $x_4{=}(6,8)$, $x_5{=}(8,7)$, $x_6{=}(8,9)$, centre the data, compute the covariance matrix Σ, find its eigenvalues and eigenvectors, and determine what fraction of variance PC1 explains.

CENTERING. PCA requires zero-mean data. Compute the sample mean:

$$\begin{aligned}\bar{x} &= \frac{1}{6}\sum_{i=1}^{6} x_i \\ &= \left(\frac{28}{6},\ \frac{30}{6}\right) \\ &= \left(\frac{14}{3},\ 5\right)\end{aligned}$$

Subtract $\bar{x}$ from each point. Here is the full calculation for x_1:

$$\begin{aligned}\tilde{x}_1 &= x_1 - \bar{x} \\ &= \left(1 - \tfrac{14}{3},\ 2 - 5\right) \\ &= \left(-\tfrac{11}{3},\ -3\right)\end{aligned}$$

Compute the remaining five yourself, then verify against the table below.

Point	**Original**	**Centred $\tilde{x}_i$**
x_1	$(1,\ 2)$	$(-11/3,\ -3)$
x_2	$(2,\ 3)$	$(-8/3,\ -2)$
x_3	$(3,\ 1)$	$(-5/3,\ -4)$
x_4	$(6,\ 8)$	$(4/3,\ 3)$
x_5	$(8,\ 7)$	$(10/3,\ 2)$
x_6	$(8,\ 9)$	$(10/3,\ 4)$

Table 6: Centred data. The second coordinate centres cleanly ($\bar{x}^{(2)}{=}5$); the first introduces thirds ($\bar{x}^{(1)}{=}14/3$).

THE COVARIANCE MATRIX. $\Sigma = \tilde{X}^T\tilde{X}/n$. Here is the full calculation for Σ_{11}:

$$\begin{aligned}\Sigma_{11} &= \frac{1}{6}\sum_{i=1}^{6}\left(\tilde{x}_i^{(1)}\right)^2 \\ &= \frac{1}{6}\left[\frac{121}{9}+\frac{64}{9}+\frac{25}{9}+\frac{16}{9}+\frac{100}{9}+\frac{100}{9}\right] \\ &= \frac{1}{6}\cdot\frac{426}{9} \\ &= \frac{71}{9} \\ &\approx 7.89\end{aligned}$$

Compute Σ_{12} and Σ_{22} yourself. The full matrix:

$$\Sigma = \begin{pmatrix}71/9 & 47/6\\ 47/6 & 29/3\end{pmatrix} \approx \begin{pmatrix}7.89 & 7.83\\ 7.83 & 9.67\end{pmatrix}$$

$\Sigma_{12} = 47/6 \approx 7.83$: both features are strongly positively correlated, as expected from the scatter plot where both coordinates increase together from cluster A to cluster B.

EIGENVALUES. Apply the formula from the theory section with $a = 71/9$, $b = 47/6$, $c = 29/3$:

$$\begin{aligned}a + c &= \frac{71}{9}+\frac{29}{3} \\ &= \frac{158}{9} \\ &\approx 17.56\end{aligned}$$

$$\begin{aligned}(a-c)^2 + 4b^2 &= \left(\frac{-16}{9}\right)^2 + 4\left(\frac{47}{6}\right)^2 \\ &= \frac{256}{81}+\frac{2209}{9} \\ &= \frac{20137}{81} \\ &\approx 248.6\end{aligned}$$

So:

$$\begin{aligned}\lambda &= \frac{17.56 \pm \sqrt{248.6}}{2} \\ &\approx \frac{17.56 \pm 15.77}{2}\end{aligned}$$

$\lambda_1 \approx 16.66$ and $\lambda_2 \approx 0.89$.

EIGENVECTOR FOR $\lambda_1 \approx 16.66$. Solve $(a - \lambda_1)\, w^{(1)} + b\, w^{(2)} = 0$:

$$\begin{aligned}(7.89 - 16.66)\, w^{(1)} + 7.83\, w^{(2)} &= 0 \\ -8.77\, w^{(1)} + 7.83\, w^{(2)} &= 0 \\ w^{(2)} &= \frac{8.77}{7.83}\, w^{(1)} \\ &\approx 1.12\, w^{(1)}\end{aligned}$$

$$\begin{aligned}\|w_1\| &= \sqrt{1^2 + 1.12^2} \\ &= \sqrt{2.25} \\ &= 1.50, \\ w_1 &\approx \frac{1}{1.50}\begin{pmatrix}1\\1.12\end{pmatrix} \\ &\approx \begin{pmatrix}0.67\\0.75\end{pmatrix}.\end{aligned}$$

EIGENVECTOR FOR $\lambda_2 \approx 0.89$. Solve $(a - \lambda_2)\, w^{(1)} + b\, w^{(2)} = 0$:

$$\begin{aligned}7.00\, w^{(1)} + 7.83\, w^{(2)} &= 0 \\ w^{(2)} &\approx -0.89\, w^{(1)}\end{aligned}$$

Normalising: $w_2 \approx \begin{pmatrix}0.75\\-0.67\end{pmatrix}$.

Verify orthogonality: $w_1^T w_2 = (0.67)(0.75) + (0.75)(-0.67) = 0$. ✓

EXPLAINED VARIANCE BY PC1:

$$\begin{aligned}\frac{\lambda_1}{\lambda_1 + \lambda_2} &\approx \frac{16.66}{17.56} \\ &\approx 95\%\end{aligned}$$

GEOMETRIC INTERPRETATION. PC1 points roughly along the direction from cluster A (bottom left) to cluster B (top right). The two clusters that are visible in two dimensions can be separated using a single principal component.

PROJECT AND RECONSTRUCT. Using the eigenvectors from the previous exercise, project each of the six centred points onto PC1, reconstruct them in the original space, and compute the reconstruction error.

PROJECTION ONTO PC1. For each centred point $\tilde{x}_i$, the coordinate on PC1 is $z_i = \tilde{x}_i^T w_1$. Here is the full calculation for x_1:

$$\begin{aligned} z_1 &= \tilde{x}_1^T w_1 \\ &= (-3.67)(0.67) + (-3)(0.75) \\ &= -2.46 - 2.25 \\ &= -4.71 \end{aligned}$$

Compute z_2 through z_6 yourself, then verify against the table below.

RECONSTRUCTION. Each projected point maps back to 2D via $\hat{x}_i = z_i \cdot w_1$. For x_1:

$$\begin{aligned} \hat{x}_1 &= -4.71 \cdot \begin{pmatrix} 0.67 \\ 0.75 \end{pmatrix} \\ &= \begin{pmatrix} -3.16 \\ -3.53 \end{pmatrix} \end{aligned}$$

RECONSTRUCTION ERROR:

$$\begin{aligned} \|\tilde{x}_1 - \hat{x}_1\|^2 &= (-3.67 + 3.16)^2 + (-3.00 + 3.53)^2 \\ &= (-0.51)^2 + (0.53)^2 \\ &= 0.26 + 0.28 \\ &= 0.54 \end{aligned}$$

Point	**Centred** $\tilde{x}_i$	z_i	**Reconstructed** $\hat{x}_i$	$\|\tilde{x}_i - \hat{x}_i\|^2$
x_1	$(-3.67, -3)$	-4.71	$(-3.16, -3.53)$	0.54
x_2	$(-2.67, -2)$	-3.29	$(-2.20, -2.47)$	0.44
x_3	$(-1.67, -4)$	-4.12	$(-2.76, -3.09)$	2.02
x_4	$(1.33, 3)$	3.14	$(2.10, 2.36)$	1.01
x_5	$(3.33, 2)$	3.73	$(2.50, 2.80)$	1.33
x_6	$(3.33, 4)$	5.23	$(3.50, 3.92)$	0.04
			Total	**5.38**

Table 7: PCA projection onto PC1 and reconstruction for the canonical six points. The total error $\approx n \cdot \lambda_2 = 6 \times 0.89 = 5.34$ (the small difference is rounding in intermediate steps). Point x_3 has the largest error because it sits furthest from the PC1 direction; x_6 has almost none because it lies nearly on PC1.

NAÏVE FEATURE SELECTION VS PCA. Using the same six centred points, compare the reconstruction error of naïve feature selection with PCA.

DROP $x^{(2)}$ (KEEP $x^{(1)}$). The reconstruction is $\hat{x}_i = (\tilde{x}_i^{(1)}, 0)$. For x_1:

$$\begin{aligned}\|\tilde{x}_1 - \hat{x}_1\|^2 &= (-3.67 - (-3.67))^2 + (-3 - 0)^2 \\ &= 0 + 9 \\ &= 9\end{aligned}$$

DROP $x^{(1)}$ (KEEP $x^{(2)}$). The reconstruction is $\hat{x}_i = (0, \tilde{x}_i^{(2)})$. For x_1:

$$\begin{aligned}\|\tilde{x}_1 - \hat{x}_1\|^2 &= (-3.67 - 0)^2 + (-3 - (-3))^2 \\ &= 13.47 + 0 \\ &= 13.47\end{aligned}$$

Compute the remaining errors yourself, then verify against the table.

Point	**Drop $x^{(2)}$**	**Drop $x^{(1)}$**	**PCA (PC1)**
x_1	9.00	13.44	0.54
x_2	4.00	7.11	0.44
x_3	16.00	2.78	2.02
x_4	9.00	1.78	1.01
x_5	4.00	11.11	1.33
x_6	16.00	11.11	0.04
Total	**58.00**	**47.33**	**5.38**

Table 8: Reconstruction error comparison. The naïve variance ranking correctly picks $x^{(2)}$ ($\mathrm{Var}(x^{(2)}) \approx 9.67 > \mathrm{Var}(x^{(1)}) \approx 7.89$), yet PCA still reduces the error by a factor of ≈ 9.

The naïve approach gets the ranking right: $x^{(2)}$ does carry more variance than $x^{(1)}$. But neither coordinate axis aligns with the direction the data actually varies along. PCA constructs that direction from the data and achieves roughly nine times less error.

CODE is provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

THE CODE EXCERCISES LOAD 1 000 handwritten digits from MNIST: 784 pixels each, ten classes. In code, you will apply three preprocessing stages: discard uninformative pixels via a variance threshold, centre the data by subtracting the per-feature mean (without centring, PC1 aligns with the mean intensity rather than with the dominant direction of variation), and eigendecompose the covariance matrix to project onto the top two components. Then compare the PCA projection with t-SNE (sweeping perplexity) and UMAP (sweeping neighbourhood size).

WHAT TO OBSERVE. PCA preserves global geometry: The ten digit classes form broad, overlapping regions whose distances reflect the original space. Because PCA is linear, it cannot untangle classes that

live on a curved manifold; overlap in the 2D plot is a direct consequence. t-SNE compresses each class into a tight ball, but distances between balls are arbitrary and change with the random seed. UMAP produces similarly tight clusters while retaining more of the global arrangement. Both nonlinear methods are sensitive to their main hyperparameter; the sweep makes this visible.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What does it mean for data to have low intrinsic dimensionality?
- Why does PCA correspond to an eigenvector decomposition of the covariance matrix?
- What is the difference between an uncorrelated and an independent component?
- Why does t-SNE use a Student-t kernel in the low-dimensional space but a Gaussian in the high-dimensional space?
- When would you prefer UMAP over t-SNE, and when would you use PCA instead of either?
- What does the perplexity parameter actually control, and what happens when it is too low or too high?
- Why should you never use t-SNE or UMAP embeddings as features for a downstream classifier?
- How does the curse of dimensionality motivate dimensionality reduction before running K-Means?

RECAP of Key Concepts:

- PCA finds the orthogonal directions of maximum variance via eigendecomposition of the covariance matrix; SVD is the numerically preferred implementation.
- The explained-variance ratio guides the choice of k; retain enough components to exceed a threshold such as 95%.
- t-SNE and UMAP preserve local neighbourhood patterns for visualization; cluster sizes and between-cluster distances in their output are not interpretable.

- ICA finds statistically independent sources by maximising non-Gaussianity, going beyond the uncorrelated directions found by PCA.
- No non-linear method should be used as a preprocessing step for downstream learning; use PCA or SVD for that purpose.

PCA AND ICA CAN ONLY ROTATE AND SCALE AXES. When the underlying structure is nonlinear, no linear map can recover it. t-SNE and UMAP solve this for visualization, but neither learns a parametric encoder that can be applied to new data or reused as a preprocessing step.

AUTOENCODERS learn both the compression and the reconstruction from data. The encoder is a neural network that maps x_i to a low-dimensional code z_i; the decoder maps z_i back to $\hat{x}_i$. Variational Autoencoders go one step further: the encoder outputs not a single point but a mean μ and a variance σ^2, turning the code into a distribution from which new data can be sampled.

THE FUNDAMENTAL LIMIT of PCA and ICA is linearity; they cannot capture curved structure. t-SNE and UMAP learn nonlinear embeddings, but are designed for visualization, not for learning a reusable mapping.

TEASER. If we replace the fixed eigenvector projection with a learned, nonlinear encoder, can we get a compact representation that generalises to unseen inputs?

FEEDBACK

Variational Inference

2026-05-06 · cheerful mango Haubentaucher

EMBEDDINGS AND LATENT FEATURE SPACES.

The Why

THE CENTRAL PROBLEM OF UNSUPERVISED LEARNING is representation learning: finding a mapping from raw observations to a latent feature space in which the underlying factors of variation become explicit. Bengio, Courville, and Vincent[14] argue that the quality of a representation determines the success of any downstream task, and that a good representation disentangles the factors that explain the data.

PCA gave us a first answer: project onto the directions of maximum variance. But PCA is linear; it cannot recover structure that lies on a curved manifold. What we lack is a parametric encoder that can be applied to new, unseen data or reused as a feature extractor on nonlinear feature spaces.

AUTOENCODERS close that gap. Hinton and Salakhutdinov[15] showed that a neural network can learn a nonlinear, low-dimensional embedding z from which a second network reconstructs $\hat{x}$, outperforming PCA on complex data. The encoder $\theta_E \colon \mathbb{R}^d \to \mathbb{R}^k$ and the decoder $\theta_D \colon \mathbb{R}^k \to \mathbb{R}^d$ are trained jointly to minimise reconstruction error $\|x - \theta_D(\theta_E(x))\|^2$. The bottleneck forces the network to discover a compact representation.

STRUCTURE REQUIRES A PROBABILISTIC FORMULATION.

If we replace the deterministic bottleneck with a distribution, and regularise that distribution toward a known prior, the latent space becomes smooth and complete: every region maps to a plausible output, and nearby embeddings correspond to similar inputs. Kingma and Welling[16] formalised this idea as the Variational Autoencoder,

PLATO'S CAVE ALLEGORY describes prisoners who see only shadows on a wall, never the objects that cast them. The shadows are high dimensional and entangled; the true forms behind them are few and independent. Dimensionality reduction, as we have seen, tries to recover those forms from the shadows. This chapter asks two follow-up questions: can we organise the recovered forms into a structured, reusable feature space, and once we have that space, can we cast new, plausible shadows we have never observed?

Plato. *The Republic*. Penguin Classics, 2007. Book VII, 514a–520a. Originally written c. 380 BCE

[14] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. DOI: 10.1109/TPAMI.2013.50. https://scholar.google.com/scholar?q=Bengio+Courville+Vincent+2013+representation+learning+review+new+perspectives

[15] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. DOI: 10.1126/science.1127647. https://scholar.google.com/scholar?q=Hinton+Salakhutdinov+2006+reducing+dimensionality+neural+networks

A PRIOR is the distribution we assume over the latent space before seeing any data.

[16] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2014. https://scholar.google.com/scholar?q=Kingma+Welling+2014+Auto-Encoding+Variational+Bayes

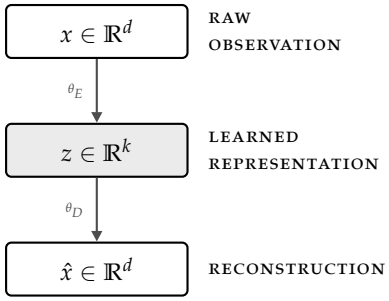


Figure 81: Autoencoder architecture. The encoder θ_E maps the input to a low-dimensional representation; the decoder θ_D reconstructs the input from it. The bottleneck forces the network to discover a compact latent feature space.

deriving a principled training objective from variational inference that balances faithful reconstruction against latent space regularity.

ALL OF THIS MUST HAPPEN WITHOUT LABELS. Supervised learning can organise a feature space by asking "which class does this input belong to?" and adjusting representations until the answer is easy to read off. An unsupervised model has no such signal. Instead, it must derive structure from the data alone, using two complementary pressures: the requirement to reconstruct the input faithfully, and the requirement to keep the latent space close to a known prior. Reconstruction forces the embedding to retain information; the prior forces the embedding to be organised. Neither pressure requires a human to annotate a single example.

IN ORDER TO MOVE FROM RECONSTRUCTION-ONLY EMBEDDINGS TO WELL-ORGANISED LATENT FEATURE SPACES we have good reason to find ways to,

- formulate a learning objective that rewards both faithful reconstruction and a smooth, regular latent space.
- train a model that outputs distributions rather than points, so that the latent space has no undefined gaps.
- control how much structure the latent space carries, and understand how that choice affects the quality of the learned representation.

Hands-On Experience

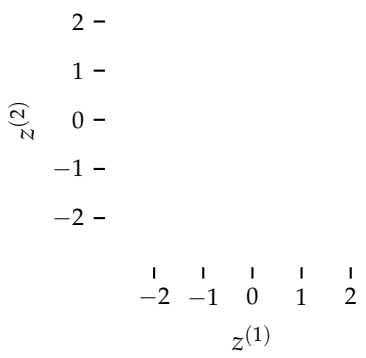
FORM GROUPS OF TWO: an encoder and a decoder. Prepare five digit cards showing **0**, **1**, **3**, **7**, and **8**. Shuffle them face down on the table.

STEP 1: DESIGN YOUR FEATURES. The encoder picks two properties of digit shapes and writes them down. These are your two latent features $z^{(1)}$ and $z^{(2)}$. For example: $z^{(1)}$ = "number of closed loops" (0 has one, 8 has two, 1 has none) and $z^{(2)}$ = "has a straight vertical stroke" (1 yes, 7 yes, 0 no); make sure to come up with your own The decoder does not see these definitions.

STEP 2: PLAY. Each round, the encoder flips one card, looks at the digit, and writes two numbers $(z^{(1)}, z^{(2)})$ on a slip of paper. The decoder receives only the slip and guesses which digit it is. Then reveal the card. The decoder now knows: "the embedding $(1, 0)$ meant the digit 3." Round by round, the decoder builds a mental map of what the embeddings mean. When all five cards are done, shuffle and keep going.

STEP 3: OBSERVE. In early rounds, the decoder guesses blindly. With each reveal, the decoder learns what the embeddings mean. At some point the reconstruction loss drops to zero: the decoder has learned the encoder's representation from examples alone.

STEP 4: RECORD AND COMPARE. Plot your five embeddings in the coordinate system below.



STEP 5: SWAP DECODERS. Find another group. Send your decoder to them, take theirs. Play one more pass with the swapped decoder.

SCORING. After the decoder guesses, flip the card. Correct digit = 0, wrong digit = 1. Sum over all rounds: that is your reconstruction loss.

Figure 82: Your latent space. Plot the five digit embeddings at the coordinates $(z^{(1)}, z^{(2)})$ your encoder chose.

- How many rounds did the decoder need before reconstructions became reliable?
- Which digits did the decoder confuse most often? What does that tell you about your feature choice?

The swapped decoder has learned a different codebook: in their group, $(1, 0)$ might have meant "7," in yours it means "3." The reconstruction loss jumps back up. The problem is not that either representation is wrong; both worked within their own group. The problem is that nothing forced the two groups to organise their embeddings the same way.

STEP 6: SHRINK THE BOTTLENECK. Drop to a single latent feature $z^{(1)}$. The encoder must pick the one property that matters most; the decoder has only one number to work with. Play a full pass and record the new reconstruction loss.

STEP 7: DISCRETISE. Go back to two features, but restrict each to binary values (0 or 1). The encoder can no longer spread digits across a continuous range and must find features that cleanly partition the digit set with just two bits.

STEP 8: ADD DIGITS. Keep two binary features, but add the cards **2**, **4**, **5**, and **9**. A representation that separated five digits may collapse when it must also distinguish eight. But the extra classes also expose ad hoc feature choices: properties that happened to work for a small set get overwhelmed, and the encoder is pressured toward genuinely structural features (loops, strokes, intersections) that scale.

STEP 9: SWAP DECODERS AGAIN. Find a different group than in Step 5. Exchange decoders and play one pass with the expanded digit set. Compare the reconstruction loss to the Step 5 swap: did the stress tests in Steps 6 to 8 push both groups toward more compatible representations.

After each decoder swap, collect the chosen latent features from all groups on the board. Are the feature sets converging across the room? Which features survived the stress tests and which were abandoned?

EACH STEP HURTS RECONSTRUCTION but forces the encoder to find features that generalise across inputs rather than overfit to a narrow set. This tension between reconstruction fidelity and generalisation through compression or broader coverage is exactly the trade-off that the VAE formalises. And a trade-off you know from supervised learning as well, overfitting versus generalization.

THIS IS THE REPRESENTATION LEARNING PROBLEM. Every choice of features defines a different geometry: different neighbours, different distances. Within one encoder-decoder pair, reconstruction works. Across pairs, the embeddings are incompatible.

THE FUNDAMENTAL QUESTION is: can we add a shared convention to the latent space, so that the embeddings are not only good for

TWO OBJECTIVES, ONE GAME. Steps 2 to 4 trained the reconstruction part: the decoder learned to recover digits from embeddings. Steps 5 and 9 exposed the missing regularisation: without a shared convention for what the embeddings mean, a new decoder cannot read them. Steps 6 to 8 showed that compression and broader coverage force better features. A VAE adds exactly this second pressure: it forces every encoder to arrange its embeddings according to a shared prior, so that any decoder can use them.

reconstruction but also follow a common structure that any decoder could interpret?

The Learning Objectives of this lecture:

- The Autoencoder Training Cycle. Architecture, forward pass, reconstruction loss, backpropagation, and what the bottleneck forces the network to learn.
- From Autoencoders to VAEs. Why reconstruction alone leaves the latent space unstructured, and what a probabilistic formulation buys us.
- The Evidence Lower Bound. The training objective that balances faithful reconstruction against latent space regularity.
- The Reparameterisation Trick. How to backpropagate through a stochastic sampling step.
- VAE Training and Bottleneck Design. Practical choices that shape the latent space: loss functions, latent dimensionality, and numerical stability.
- Exploring the Latent Space. Interpolation and generation as tests of representation quality.
- VAE Variants. How they each change one structural assumption.
- Feature Extraction, Transfer, and Distillation. Using the trained encoder as a backbone for downstream tasks.

The Autoencoder

Follows the notion of a bottleneck architecture: an encoder that compresses the input into a low-dimensional embedding, and a decoder that reconstructs the input from that embedding. Due to the bottleneck, the encoder must learn to extract the most salient features of the input, and the decoder must learn to use those features to reconstruct the input as losslessly as possible.

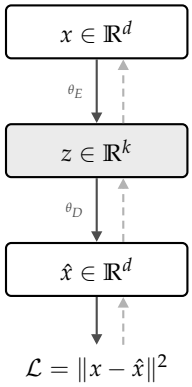


Figure 83: Autoencoder training cycle. Solid arrows: forward pass ($x \to z \to \hat{x} \to \mathcal{L}$). Dashed arrows: gradient flow back through decoder and encoder.

THE ENCODER $\theta_E \colon \mathbb{R}^d \to \mathbb{R}^k$ compresses a d-dimensional input x into a k-dimensional embedding z, with $k \ll d$.

NOTATION. θ_E denotes the encoder; θ_D denotes the decoder. Both are neural networks trained jointly.

THE DECODER $\theta_D \colon \mathbb{R}^k \to \mathbb{R}^d$ takes z and attempts to reconstruct the original input as $\hat{x}$.

THE FORWARD PASS maps a single training example through both networks. Given an input x, the encoder produces the embedding:

$$z = \theta_E(x)$$

The decoder then reconstructs:

$$\begin{aligned} \hat{x} &= \theta_D(z) \\ &= \theta_D(\theta_E(x)) \end{aligned}$$

THE RECONSTRUCTION LOSS measures how accurate the decoder reconstructs the input. The standard choice is the mean squared error over all dimensions:

$$\begin{aligned} \mathcal{L}(\theta_E, \theta_D;\, x) &= \|x - \hat{x}\|^2 \\ &= \sum_{j=1}^{d} (x_j - \hat{x}_j)^2 \end{aligned}$$

BACKPROPAGATION is the backbone of supervised learning, and it applies here unchanged. The loss $\mathcal{L}$ depends on θ_D (through $\hat{x}$) and on θ_E (through z, which feeds into $\hat{x}$). The chain rule gives:

$$\frac{\partial \mathcal{L}}{\partial \theta_D} = \frac{\partial \mathcal{L}}{\partial \hat{x}} \cdot \frac{\partial \hat{x}}{\partial \theta_D}$$

for the decoder, and

$$\frac{\partial \mathcal{L}}{\partial \theta_E} = \frac{\partial \mathcal{L}}{\partial \hat{x}} \cdot \frac{\partial \hat{x}}{\partial z} \cdot \frac{\partial z}{\partial \theta_E}$$

for the encoder. The gradient signal flows from the loss, through the decoder, through z, and into the encoder. Both networks are updated from the same single loss.

THE PARAMETER UPDATE applies standard gradient descent (or a variant such as Adam):

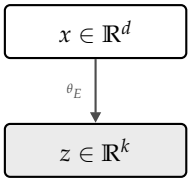
$$\theta_E \leftarrow \theta_E - \eta \frac{\partial \mathcal{L}}{\partial \theta_E}$$

$$\theta_D \leftarrow \theta_D - \eta \frac{\partial \mathcal{L}}{\partial \theta_D}$$

where η is the learning rate. One forward pass, one backward pass, one update: that is a single training step. Repeating this over minibatches and epochs, the encoder and decoder co-adapt until the reconstruction error converges.

THE BOTTLENECK. Because $k \ll d$, the embedding z cannot simply copy x. The encoder must decide which aspects of the input to preserve and which to discard. This forced compression is what generalizes the encoder into a feature extractor.[17]

THE TRAINED ENCODER IS A FEATURE EXTRACTOR, OR DIMENSIONALITY REDUCER. Once training converges, the decoder can be discarded. The encoder alone maps any new input x to an embedding z that captures the factors of variation learned during training. This embedding can serve as input to a classifier, a clustering algorithm, or a retrieval system. The autoencoder has learned a representation; the question is whether that representation is generalized.



CONNECTION TO PCA. If the encoder and decoder are both linear (no activation functions) and the loss is MSE, the autoencoder recovers the same subspace as PCA[18]. The k embedding dimensions span the top-k principal component directions. Adding nonlinear activations allows the network to capture structure that PCA cannot: curved manifolds, discrete clusters, hierarchical features.

[17] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. DOI: 10.1126/science.1127647. https://scholar.google.com/scholar?q=Hinton+Salakhutdinov+2006+reducing+dimensionality+neural+networks

WHY DEEP (LEARNING)? Each layer applies a linear transformation followed by a nonlinear activation. A single hidden layer can in principle approximate any function, but may need impractically many neurons and is hard to learn. Deeper networks compose simple nonlinearities into increasingly abstract features: the first layer detects edges, the second combines edges into textures, the third recognises parts. Each layer reuses the representations below it, so the network expresses complex structure with far fewer parameters than a shallow one. Hinton and Salakhutdinov showed that a deep autoencoder with the same total capacity as a shallow one produces qualitatively better embeddings, because hierarchical composition is a more efficient way to represent the kind of structure found in real data.

[18] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989. DOI: 10.1016/0893-6080(89)90014-2. https://scholar.google.com/scholar?q=Baldi+Hornik+1989+neural+networks+principal+component+analysis

From Autoencoders to Variational Autoencoders

STANDARD AUTOENCODERS learn $x \to z \to \hat{x}$, minimising $\|x - \hat{x}\|^2$.

THE LATENT SPACE HAS NO GEOMETRIC GUARANTEES. The objective only rewards reconstruction; nothing constrains how embeddings are arranged. Nearby points may decode to unrelated outputs, gaps between embeddings are undefined, and the layout can change arbitrarily across training runs. The representation captures enough to reconstruct, but it does not organise the factors of variation in a way that supports interpolation, sampling, or transfer.

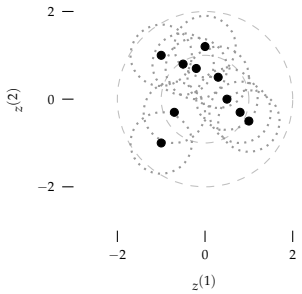
TO FIX THIS, WE ADD A CONSTRAINT. In a plain autoencoder, the encoder's output layer produces a single vector $z \in \mathbb{R}^k$: one fixed point per input. In a Variational Autoencoder, the output layer produces two vectors instead: a mean $\mu \in \mathbb{R}^k$ and a variance $\sigma^2 \in \mathbb{R}^k$. Together they define a Gaussian distribution over the latent space:

$$q_{\theta_E}(z|x) = \mathcal{N}(\mu, \sigma^2)$$

During training, we do not pass μ directly to the decoder. Instead we sample from that distribution:

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

So the decoder sees a slightly different z every time, even for the same input.



VAE latent space. Each small circle is one input's encoder distribution $q_{\theta_E}(z|x_i)$; the dot marks its mean μ. The large dashed circles show the prior $p(z) = \mathcal{N}(0, I)$ at 1σ and 2σ. The KL term pushes each small circle toward unit variance and its mean toward the origin, so that the mixture of all encoder distributions approximates the prior.

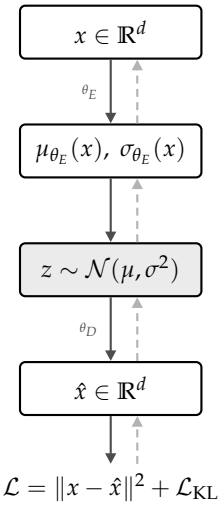
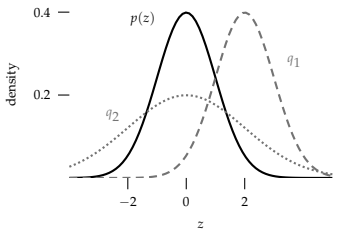


Figure 84: VAE architecture. Compare with Figure 83: the deterministic bottleneck is replaced by distribution parameters μ, σ and a stochastic sample. The loss combines reconstruction fidelity with KL regularisation toward the prior.

WE ENFORCE THAT BY ADDING A LOSS a KL divergence $\mathcal{L}_{\text{KL}}(q_{\theta_E}(z|x) \| p(z))$, computed for each input separately. It measures how far that input's

encoder distribution $\mathcal{N}(\mu, \sigma^2)$ is from the prior $p(z) = \mathcal{N}(0, I)$. The KL between two Gaussians penalises two things: a mean μ that is far from the origin, and a variance σ^2 that deviates from one. So for each input, the KL term pulls the mean toward zero and the variance toward unity.



KL PENALTY. $p(z) = \mathcal{N}(0,1)$ is the prior (solid). $q_1 = \mathcal{N}(2,1)$ (dashed) is shifted; $q_2 = \mathcal{N}(0,4)$ (dotted) is widened. Spreading mass (q_2, $\mathcal{L}_{\text{KL}}$=0.81) costs more than translating it (q_1, $\mathcal{L}_{\text{KL}}$=0.50, evaluated in Exercise 2).

IF THE KL TERM ACTED ALONE, every input would collapse onto the same distribution $\mathcal{N}(0, I)$, and the decoder could not tell any two inputs apart. The reconstruction term fights back: it needs different inputs to have distinguishable embeddings, so it pushes their means apart. Training finds a compromise. Each input gets a distribution that is close to the prior but not identical to it: the means spread out just enough for the decoder to reconstruct, and the variances stay close to one. When all these slightly shifted, roughly unit-variance Gaussians are added together, their aggregate approximates the prior. That is how the prior fills in the gaps between embeddings and gives the latent space its geometric structure. Together they turn the encoder into a feature extractor whose embeddings are both informative and geometrically well behaved: nearby embeddings correspond to similar inputs, and every region of the space is meaningful.

THE RESULT is a latent space where geometric proximity reflects semantic similarity.

VAE Variants

β-VAE[19] scales up the KL term to pressure the encoder to use the latent space efficiently. The loss reweights the with a single hyperparameter:

$$\mathcal{L}_\beta = \text{Reconstruction} + \beta \cdot \mathcal{L}_{\text{KL}}$$

VQ-VAE[20] replaces continuous latents with discrete codes from a learned codebook. Each encoder output z_e is snapped to the nearest codebook entry, giving reconstructional structure:

$$z_q = \arg \min_{e_k \in \text{codebook}} \|z_e - e_k\|$$

CONDITIONAL VAE[21] feeds a conditioning variable c, such as a class label, into both encoder and decoder. The model now learns conditional distributions, which enables directed generation:

$$q_{\theta_E}(z|x,c), \quad p_{\theta_D}(x|z,c)$$

[19] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. β-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations (ICLR)*, 2017. https://scholar.google.com/scholar?q=Higgins+2017+beta-VAE+learning+basic+visual+concepts

[20] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. https://scholar.google.com/scholar?q=van+den+Oord+2017+neural+discrete+representation+learning+VQ-VAE

[21] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, 2015. https://scholar.google.com/scholar?q=Sohn+Lee+Yan+2015+learning+structured+output+conditional+generative

The Reparameterisation Trick

How do we now the distribution of the current embedding space? Our model now reconstructs data in two steps: draw z from the prior, then decode. To measure how well the model explains an observation x, we would need to sum over all possible embeddings z and check which ones decode close to x:

$$p_{\theta_D}(x) = \int p_{\theta_D}(x|z)\, p(z)\, dz$$

In words. The probability that the model generates x equals, summed over every possible latent z, the probability that the decoder produces x given that z, weighted by the prior probability of drawing that z in the first place. Each z is one possible explanation; the integral combines all of them.

Spoiler. We can't. This integral has no closed form and the latent space is too high-dimensional to sum over by brute force. Instead of summing over all of latent space, the encoder $q_{\theta_E}(z|x)$ output is used as an estimate, for each input x, where useful embeddings z for x live. We sample only from that region[22], turning an impossible integral into a practical expectation over a handful of samples.

[22] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2014. https://scholar.google.com/scholar?q=Kingma+Welling+2014+Auto-Encoding+Variational+Bayes

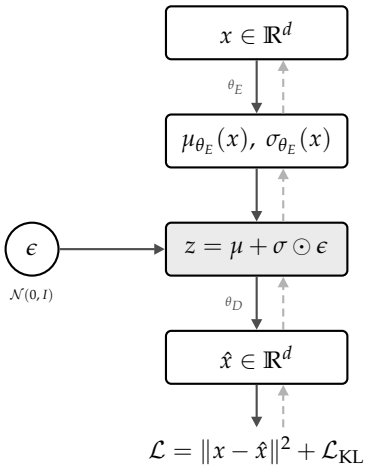


Figure 85: Reparameterised VAE or **Kingma's view**. Compare with Figure 84: the stochastic sample is replaced by $z = \mu + \sigma \odot \epsilon$ where ϵ carries the randomness and ,is drawn from a fixed distribution. Gradients (dashed) flow through μ and σ without passing through the random sample.

Designing the Bottleneck

THE LATENT DIMENSIONALITY k is the most consequential hyperparameter in a VAE. It controls the capacity of the information channel between encoder and decoder: how many bits of information about x can pass through z.

$$z \in \mathbb{R}^k$$

The bottleneck embedding. The dimensionality k controls how much information passes from encoder to decoder.

TOO FEW DIMENSIONS and the encoder cannot represent the factors of variation in the data. Reconstruction quality degrades because the embedding lacks the capacity to distinguish meaningfully different inputs. Two inputs that differ in important ways may map to nearly identical latent vectors, and the decoder produces a blurred average of what those embeddings could mean.

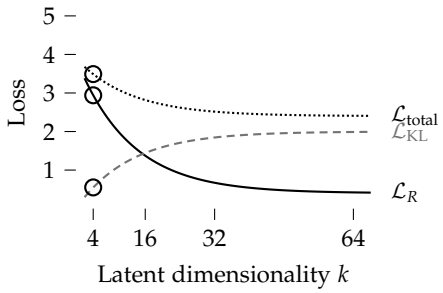


Figure 86: Bottleneck too small. At k=4, reconstruction loss is large ($\mathcal{L}_R \approx 2.9$) because the few active dimensions cannot encode all factors of variation. Each active dimension carries a lot of signal, but only a handful of them are paying KL ($\mathcal{L}_{\text{KL}} \approx 0.55$). The total ($\circ$ on the dotted curve) sits high because reconstruction error dominates.

TOO MANY DIMENSIONS and the model faces the opposite problem. With excess capacity, the encoder can encode each training point precisely without needing to discover shared structure. The KL term penalises unused dimensions toward the prior, but is blended out by the reconstruction success.

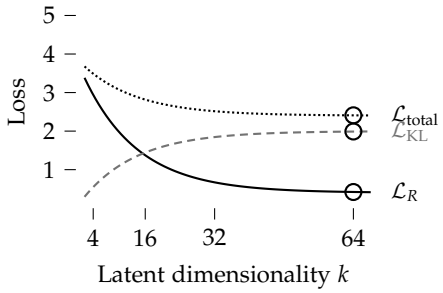


Figure 87: Bottleneck too large. At k=64, reconstruction loss has saturated near its floor ($\mathcal{L}_R \approx 0.42$) and the KL cost has saturated near its ceiling ($\mathcal{L}_{\text{KL}} \approx 1.99$): adding more dimensions changes neither term, because extra dimensions collapse to the prior and pay no KL. The total ($\circ$ on the dotted curve) sits at the plateau. The cost of k too large is therefore wasted parameters and slower training, not a higher loss.

THE INFORMATION BOTTLENECK PRINCIPLE[23] provides a theoretical lens. The optimal embedding retains all information in x that is relevant to reconstruction. The latent dimensionality k sets the ceiling

[23] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000. https://arxiv.org/abs/physics/0004057

on how much can be stored; the KL weight determines how aggressively the model compresses toward that ceiling to discarding weak representations.

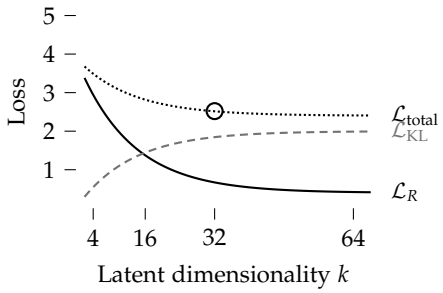


Figure 88: Bottleneck sizing trade-off. Reconstruction loss (solid) falls as k grows; the KL cost (dashed) rises until the data's information has been encoded, then plateaus as additional dimensions collapse to the prior. The total (dotted) is L-shaped: it drops sharply, then flattens. A good k sits near the elbow (○ at k=32), where the marginal gain per added dimension has fallen below what the extra parameters and training time cost.

ACTIVE UNITS. Inactive dimensions have collapsed to the prior, as there is no potential reconstruction gain contributed by their information. Monitoring the number of active units during training reveals how much of the latent capacity the model actually uses[24].

ACTIVE-UNIT CRITERION. A latent dimension j is called active if

$$\mathcal{L}_{\text{KL}}(q(z_j|x) \,\|\, p(z_j)) > \delta$$

for a meaningful fraction of the training data.

[24] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. In *International Conference on Learning Representations (ICLR)*, 2016. https://arxiv.org/abs/1509.00519

THIS READING ONLY HOLDS IN A PARTICULAR REGIME. The active-unit count is meaningful when k is rather large that some dimensions could afford to drop, and when the reconstruction loss is competitive with the KL pull. The diagnostic is really a measure of how many dimensions the reconstruction loss can keep alive against the KL pull. So cross-dimension activity comparisons are needed.

A PRACTICAL PROTOCOL for choosing k:

- Start with a generously large k. Adding capacity rarely hurts validation reconstruction; it just stops helping past the elbow.
- Train once and read off the effective capacity. The number of active dimensions is a direct estimate of the data's information ceiling.
- Use validation reconstruction, not training loss, as the model-selection signal to be sensitive to overfitting.

RULE OF THUMB. For input dimensionality d, a reasonable starting point is k on the order of $\sqrt{d}$ for tabular and pixel data. Text is a special case: vocabulary size makes d huge (30k+ tokens), but semantic information for sentence-level text VAEs, start at $k \approx 32-128$.

TWO EXCEPTIONS where the protocol does not apply. For visualisation, set k=2 or k=3 outright and accept the fidelity loss; the goal is a readable scatter, not optimal reconstruction. For very large models, where a single training run is expensive, pick a k from a comparable published architecture and skip the sweep entirely.

BEYOND DIMENSIONALITY, the structure of the prior itself shapes what the latent space can express. The standard choice $p(z) = \mathcal{N}(0, I)$ assumes the latent factors are independent and identically scaled. When the true factors of variation have different scales, correlations, or discrete structure, a mismatched prior forces the encoder to waste capacity compensating for the mismatch.

KL ANNEALING ramps the KL weight from zero over the first N epochs, letting the encoder learn useful representations before regularisation kicks in and prevents posterior collapse.

S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 10–21, 2016. https://arxiv.org/abs/1511.06349

Exploring the Latent Space

LATENT INTERPOLATION reveals the structure that the VAE has learned. Given two data points x_1 and x_2, encode, interpolate in the latent space, then decode each step:

$$
\begin{aligned}
z_1 &= \mu_{\theta_E}(x_1) \\
z_2 &= \mu_{\theta_E}(x_2) \\
z_t &= (1-t)\, z_1 + t\, z_2 \\
\hat{x}_t &= \theta_D(z_t)
\end{aligned}
$$

Because the latent space is regularised, the decoded path transitions smoothly between the two inputs[25]. Smooth interpolation is the simplest test of whether the encoder has learned a genuinely continuous feature space rather than a scattered collection of embeddings.

[25] T. White. Sampling generative networks. *arXiv preprint arXiv:1609.04468*, 2016. https://scholar.google.com/scholar?q=White+2016+Sampling+Generative+Networks

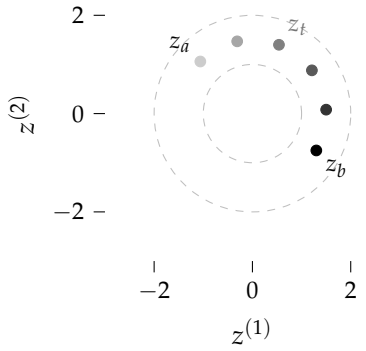


Figure 89: Latent interpolation. Two inputs encode to z_a and z_b. Linear interpolation (lerp, dashed) cuts through the low-density middle of the prior; in high dimensions this region is empty under $\mathcal{N}(0, I)$ and decoded midpoints look unnatural. Spherical linear interpolation (slerp, solid arc) follows the high-density shell at roughly $\|z\| \approx \sqrt{d}$, keeping every z_t on-distribution. Each waypoint is decoded to produce a smooth transition.

GENERATION is equally straightforward: sample from the prior and decode.

$$
\begin{aligned}
z &\sim \mathcal{N}(0, I) \\
\hat{x} &= \theta_D(z)
\end{aligned}
$$

The regularisation ensures that every region of the prior generates plausible data. The same property that makes the encoder a reliable feature extractor, a latent space with no dead zones, is what makes generation possible.

WELCOME TO GENERATIVE AI. A trained decoder paired with a tractable prior is, in essence, a generative model: draw a vector from a Gaussian, run it through the network, and a new sample falls out the other side. Everything that followed in the field, from diffusion models to large image and video generators, builds on the same trick of learning a mapping from a simple prior to a complex data distribution.

Examples & Exercises

THE EXERCISES START WITH A FEW NUMERICAL PASSES to fix the mechanics, then a set of concept and diagnostic prompts that target the key ideas of the chapter one at a time, and finally a hands-on notebook on MNIST. Worked solutions for the compute exercises are in the body; concept exercises carry the answer in a margin note. Try it yourself first, then check.

VAE LOSS BY THE NUMBERS. For a 1D VAE with encoder output μ=1, σ=1 on input x=3 with reconstruction $\hat{x}$=2.7, compute the total loss $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_{\text{KL}}$ using squared error and the Gaussian KL closed form $\frac{1}{2}(\mu^2 + \sigma^2 - \log\sigma^2 - 1)$.

RECONSTRUCTION LOSS.

$$\begin{aligned}\mathcal{L}_R &= (x - \hat{x})^2 \\ &= (3 - 2.7)^2 \\ &= 0.3^2 \\ &= 0.09\end{aligned}$$

KL TERM.

$$\begin{aligned}\mathcal{L}_{\text{KL}} &= \tfrac{1}{2}(\mu^2 + \sigma^2 - \log\sigma^2 - 1) \\ &= \tfrac{1}{2}(1 + 1 - \log 1 - 1) \\ &= \tfrac{1}{2}(1 + 1 - 0 - 1) \\ &= \tfrac{1}{2}(1) \\ &= 0.50\end{aligned}$$

TOTAL LOSS.

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_R + \mathcal{L}_{\text{KL}} \\ &= 0.09 + 0.50 \\ &= 0.59\end{aligned}$$

SECOND CONFIGURATION: μ=0, σ=1, x=3, $\hat{x}$=2.0.

$$\begin{aligned}\mathcal{L}_R &= (3-2.0)^2 \\ &= 1.00 \\ \mathcal{L}_{\text{KL}} &= \tfrac{1}{2}(0+1-0-1) \\ &= 0 \\ \mathcal{L} &= 1.00+0 \\ &= 1.00\end{aligned}$$

The encoder matches the prior exactly so the KL drops to zero, but reconstruction more than triples. The first configuration achieves a lower total loss by spending some KL for a much better fit; that trade-off is what training resolves.

WHERE THE TRADE-OFF LIVES. Reconstruction is small in configuration one but the KL penalty is five times larger; in configuration two the KL is free but reconstruction dominates. Training finds the encoder that minimises the sum, not either term in isolation.

KL DIVERGENCE BY HAND. Compute $\mathcal{L}_{\text{KL}}\big(\mathcal{N}(\mu,\sigma^2)\,\|\,\mathcal{N}(0,1)\big) = \tfrac{1}{2}(\mu^2+\sigma^2-\log\sigma^2-1)$ for three settings, and split each result into a shift cost $\tfrac{1}{2}\mu^2$ and a spread cost $\tfrac{1}{2}(\sigma^2-\log\sigma^2-1)$.

SETTING (I): μ=0, σ=1 (posterior matches the prior).

$$\begin{aligned}\tfrac{1}{2}\mu^2 &= 0 \\ \tfrac{1}{2}(\sigma^2-\log\sigma^2-1) &= \tfrac{1}{2}(1-0-1) \\ &= 0 \\ \mathcal{L}_{\text{KL}} &= 0+0 \\ &= 0\end{aligned}$$

SETTING (II): μ=1, σ=1 (mean shifted, same spread).

$$\begin{aligned}\tfrac{1}{2}\mu^2 &= \tfrac{1}{2}\times 1 \\ &= 0.50 \\ \tfrac{1}{2}(\sigma^2-\log\sigma^2-1) &= \tfrac{1}{2}(1-0-1) \\ &= 0 \\ \mathcal{L}_{\text{KL}} &= 0.50+0 \\ &= 0.50\end{aligned}$$

SETTING (III): $\mu=0$, $\sigma=2$ (correct mean, too wide).

$$\begin{aligned}\tfrac{1}{2}\mu^2 &= 0\\ \tfrac{1}{2}(\sigma^2-\log\sigma^2-1) &= \tfrac{1}{2}(4-\log 4-1)\\ &= \tfrac{1}{2}(4-1.386-1)\\ &\approx 0.81\\ \mathcal{L}_{\mathrm{KL}} &\approx 0+0.81\\ &\approx 0.81\end{aligned}$$

WHY THE ASYMMETRY. The shift cost is quadratic in μ. The spread cost is asymmetric: it grows roughly linearly as σ exceeds 1, but the $-\log\sigma^2$ term blows up as $\sigma \to 0$, so narrow posteriors are the most expensive of all. Setting (iii) costs more than (ii) because spreading mass away from the prior is harder than translating it.

β-VAE BALANCE PASS. Reuse the first configuration from *VAE loss by the numbers* ($\mathcal{L}_R$=0.09, $\mathcal{L}_{\mathrm{KL}}$=0.50) and compute the β-VAE loss $\mathcal{L}_\beta = \mathcal{L}_R + \beta\cdot\mathcal{L}_{\mathrm{KL}}$ for $\beta\in\{0.5,\,1,\,4\}$.

β=0.5.

$$\begin{aligned}\mathcal{L}_{\beta=0.5} &= 0.09+0.5\times 0.50\\ &= 0.09+0.25\\ &= 0.34\end{aligned}$$

β=1 (STANDARD ELBO).

$$\begin{aligned}\mathcal{L}_{\beta=1} &= 0.09+1\times 0.50\\ &= 0.09+0.50\\ &= 0.59\end{aligned}$$

β=4.

$$\begin{aligned}\mathcal{L}_{\beta=4} &= 0.09+4\times 0.50\\ &= 0.09+2.00\\ &= 2.09\end{aligned}$$

The KL contribution to the total grows from 73% (β=0.5) to 96% (β=4). At β=4 the encoder has every incentive to shrink μ toward zero and σ toward one, at the cost of reconstruction. At β=0.5 the prior pull is relaxed and the encoder can spread out for sharper reconstructions. β=1 is the standard ELBO balance.

WHY β MATTERS. Training does not just evaluate this loss once; it follows the gradient. Whichever term dominates the loss dominates the gradient, and that is what shapes the final encoder.

REPARAMETERISATION. For μ=0.5, σ=1.2, and a sampled $\epsilon=-0.3$, compute the reparameterised sample $z=\mu+\sigma\epsilon$ and verify the gradients with respect to μ and σ.

FORWARD PASS.

$$\begin{aligned} z &= \mu + \sigma \cdot \epsilon \\ &= 0.5 + 1.2 \times (-0.3) \\ &= 0.5 - 0.36 \\ &= 0.14 \end{aligned}$$

GRADIENTS.

$$\begin{aligned} \frac{\partial z}{\partial \mu} &= \frac{\partial}{\partial \mu}(\mu + \sigma\epsilon) \\ &= 1 \\ \frac{\partial z}{\partial \sigma} &= \frac{\partial}{\partial \sigma}(\mu + \sigma\epsilon) \\ &= \epsilon \\ &= -0.3 \end{aligned}$$

Both gradients are finite scalars, so μ and σ get a clean signal from backpropagation; the randomness lives in ϵ, off the gradient path.

KL COST, BY INSPECTION. Without computing, rank the following encoder distributions by KL cost against the prior $\mathcal{N}(0, 1)$, lowest first.

- (i) μ=0, σ=1.
- (ii) μ=0, σ=0.3.
- (iii) μ=1.5, σ=1.
- (iv) μ=0, σ=3.

ANSWER. (i) < (iii) < (iv) < (ii). (i) matches the prior exactly so the KL is zero. (iii) only shifts the mean; the cost grows quadratically with μ. (iv) is too wide but the spread cost is roughly linear in σ^2. (ii) is too narrow, and the $-\log\sigma^2$ term blows up as $\sigma \to 0$, making narrow posteriors the most expensive.

AE VERSUS VAE: PREDICT THE EMBEDDING. You train a vanilla autoencoder and a VAE with k=2 on MNIST, both to convergence, and plot the validation embeddings.

- Which space concentrates encodings near the origin, and which spreads them arbitrarily across the plane?
- Does proximity in the latent space imply the inputs are visually similar? Is the answer the same for both models?

- If you pick a random point from a sparsely populated region and decode it, in which space is the result a plausible digit?
- Why k=2 and not k=10, given that MNIST has ten digit classes?

ANSWER. The AE places encodings wherever reconstruction loss is lowest; the layout is arbitrary, with no global structure and large unused regions. The VAE pulls every encoding toward the prior, so the cloud concentrates near the origin and fills the unit disc. Proximity reflects visual similarity in both, because the decoder is smooth, but only the VAE gives geometric position a global meaning: any z sampled from the prior decodes to something plausible, while a random point in an AE's empty region decodes to noise. k=2 is for visualisation; you plot $z^{(1)}$ against $z^{(2)}$ directly. At k=10 you would need t-SNE or UMAP to see the layout, and the differences you observe would be artefacts of the projection rather than of the AE-vs-VAE distinction.

PER-DIMENSION KL. For a k=4 VAE on a single input, the encoder outputs the parameters below (independent dimensions, prior $\mathcal{N}(0,1)$ per dim). Compute $\mathcal{L}_{\mathrm{KL}}^{(j)} = \frac{1}{2}(\mu_j^2 + \sigma_j^2 - \log\sigma_j^2 - 1)$ for each dimension and the total $\mathcal{L}_{\mathrm{KL}} = \sum_j \mathcal{L}_{\mathrm{KL}}^{(j)}$, then label each dimension active or collapsed.

DIM 1: μ_1=1.5, σ_1=0.5.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(1)} &= \tfrac{1}{2}(1.5^2 + 0.5^2 - \log 0.5^2 - 1)\\ &= \tfrac{1}{2}(2.25 + 0.25 + 1.386 - 1)\\ &= \tfrac{1}{2}(2.886)\\ &\approx 1.44\end{aligned}$$

DIM 2: μ_2=0, σ_2=1.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(2)} &= \tfrac{1}{2}(0 + 1 - 0 - 1)\\ &= 0\end{aligned}$$

DIM 3: μ_3=0.2, σ_3=0.9.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(3)} &= \tfrac{1}{2}(0.2^2 + 0.9^2 - \log 0.9^2 - 1)\\ &= \tfrac{1}{2}(0.04 + 0.81 + 0.211 - 1)\\ &= \tfrac{1}{2}(0.061)\\ &\approx 0.03\end{aligned}$$

DIM 4: $\mu_4 = -0.8$, σ_4=0.7.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}}^{(4)} &= \tfrac{1}{2}((-0.8)^2 + 0.7^2 - \log 0.7^2 - 1)\\ &= \tfrac{1}{2}(0.64 + 0.49 + 0.713 - 1)\\ &= \tfrac{1}{2}(0.843)\\ &\approx 0.42\end{aligned}$$

TOTAL AND VERDICT.

$$\begin{aligned}\mathcal{L}_{\mathrm{KL}} &= 1.44 + 0 + 0.03 + 0.42\\ &\approx 1.89\end{aligned}$$

Dimensions 1 and 4 are active: large per-dim KL, σ well below 1, mean clearly away from zero. Dimension 2 has collapsed to the prior. Dimension 3 is borderline: nearly at the prior, contributing almost no information.

WHAT THE PER-DIM KL TELLS YOU. Posterior collapse is a per-dimension phenomenon: a single VAE can have some dimensions active and others collapsed in the same training run. Summing the KL hides that detail; reading it per dimension exposes which axes the encoder uses and which it has given up on.

COUNTING ACTIVE UNITS. A VAE with k=8 reports the following per-dimension posterior statistics, averaged over the validation set: the squared mean $\overline{\mu_j^2}$, the variance σ_j^2, and the resulting per-dimension KL $\mathcal{L}_{\mathrm{KL}}^{(j)} = \frac{1}{2}(\overline{\mu_j^2} + \sigma_j^2 - \log \sigma_j^2 - 1)$.

Dim j	1	2	3	4	5	6	7	8
$\overline{\mu_j^2}$	1.50	0.01	0.80	0.00	1.20	0.00	0.60	0.70
σ_j^2	0.12	0.97	0.31	0.99	0.18	1.00	0.41	0.95
$\mathcal{L}_{\mathrm{KL}}^{(j)}$	1.37	0.01	0.64	0.00	1.05	0.00	0.45	0.35

Table 9: Per-dimension posterior statistics for a k=8 VAE on the validation set. Inactive dimensions have $\overline{\mu_j^2} \approx 0$ and $\sigma_j^2 \approx 1$, so their KL contribution falls to zero and they have collapsed back to the prior. Active dimensions either spread the means (large $\overline{\mu_j^2}$), tighten the variance (small σ_j^2), or both, and pay a positive KL for the information they carry.

How many dimensions are doing real work? Which ones can you remove without losing reconstruction quality? If you re-trained at k=4, would you expect the same number of active dimensions?

ANSWER. Five active dimensions: 1, 3, 5, 7, 8, identifiable by $\mathcal{L}_{\mathrm{KL}}^{(j)}$ well above zero. Dimensions 2, 4, 6 have $\mathcal{L}_{\mathrm{KL}}^{(j)} \approx 0$ and have collapsed to the prior; dropping them costs nothing. Note that dim 8 looks borderline by variance alone ($\sigma_8^2 = 0.95$), but its mean spread $\overline{\mu_8^2} = 0.70$ keeps it carrying information, which is why the KL is the cleaner diagnostic. Re-training at k=4 should still yield about four active units, because the active count is set by the data's information content, not by k once k is past the elbow.

DIAGNOSE THE TRAINING RUN. Three runs of the same VAE on the same data give different end-of-training behaviour. For each, name what is happening and what you would change.

Run	$\mathcal{L}_R$	$\mathcal{L}_{\mathbf{KL}}$	**Reconstructions**
A	high	≈ 0	blurry mean digit
B	low	high	sharp but unrelated to input
C	low	moderate	sharp and semantically sound

Table 10: Three end-of-training profiles for the same VAE on the same data. One is healthy training, one is posterior collapse, and one is a memorising encoder that bypasses the prior.

ANSWER. **A** is posterior collapse: KL near zero means the encoder ignores x, so the decoder outputs the data mean. Lower β or anneal it from zero. **B** has the encoder bypassing the prior to memorise inputs (high KL, low reconstruction, but sharp outputs that do not generalise to sampled z). Raise β or check for an over-parameterised encoder. **C** is what you want: both terms contribute and reconstructions track the inputs.

KL ANNEALING BY THE NUMBERS. A linear-ramp schedule sets

$$\beta(t) = \min\left(\tfrac{t}{10}, 1\right)$$

so β rises from 0 at epoch 0 to 1 at epoch 10 and stays at 1 afterwards. Assume that on a fixed validation batch the encoder reports $\mathcal{L}_R$=0.4 and $\mathcal{L}_{\mathrm{KL}}$=1.2 throughout training (the values themselves do not change, only the weight). Compute the total loss $\mathcal{L}_\beta(t) = \mathcal{L}_R + \beta(t) \cdot \mathcal{L}_{\mathrm{KL}}$ at epochs $t \in \{1, 5, 10, 20\}$.

EPOCH 1: β=0.1.

$$\begin{aligned}\mathcal{L}_\beta(1) &= 0.4 + 0.1 \times 1.2 \\ &= 0.4 + 0.12 \\ &= 0.52\end{aligned}$$

EPOCH 5: β=0.5.

$$\begin{aligned}\mathcal{L}_\beta(5) &= 0.4 + 0.5 \times 1.2 \\ &= 0.4 + 0.60 \\ &= 1.00\end{aligned}$$

EPOCH 10: β=1.

$$\begin{aligned}\mathcal{L}_\beta(10) &= 0.4 + 1 \times 1.2 \\ &= 1.60\end{aligned}$$

EPOCH 20: β=1 (CLAMPED).

$$\mathcal{L}_\beta(20) = 1.60$$

At which epoch does the KL contribution first match the reconstruction term, and what would $\mathcal{L}(t)$ look like if you set β=1 from the start?

WHEN THE TERMS MATCH. $\beta(t) \cdot \mathcal{L}_{\mathrm{KL}} = \mathcal{L}_R$ when $\beta(t) = 0.4/1.2 \approx 0.33$, i.e. around epoch 3 on this schedule. Without the ramp, the KL would dominate from epoch 1, giving the encoder every incentive to ignore x before it has learned a useful representation; the warm-up window buys the encoder time to first encode information, then take the regularisation pressure.

READ THE OBSERVATIONS FROM THE EMBEDDING SPACE EXPLORATION. You linearly interpolate between two encoded digits in latent space and decode each step. For each described path, name what the latent space tells you.

- Path 1: a smooth morph from a 3 into an 8, passing through plausible intermediate shapes.
- Path 2: the 3 stays fixed for half the path, then suddenly jumps to the 8.
- Path 3: the path passes through several unrelated digit shapes (a 2, then a 5) before reaching the 8.

ANSWER. Path 1 indicates a well-organised latent space: nearby points decode to similar digits and the regulariser has filled the region between the two encodings. Path 2 means the encoder placed the two digits far apart with empty space in between; the prior has not pulled them close enough. Path 3 means the line crosses other class regions; that is geometrically honest but suggests the chosen direction is not a meaningful axis.

AN MNIST SUBSET TO EXPLORE. You are given the same thousand handwritten digits as in the previous chapter. Implement three pieces from scratch in numpy: a linear autoencoder, the closed-form KL divergence to a standard normal prior, and the reparameterisation trick $z = \mu + \sigma \odot \varepsilon$. Sweep k on the linear AE and inspect the reconstruction curve, then compare precomputed AE and VAE embeddings at k=2 and at $k \in \{2, 8, 32\}$. Optionally lift a k=16 VAE: bar-chart its per-dimension KL using your own implementation, and decode an interpolation between an encoded 0 and an encoded 1.

CODE is provided as a Python notebook. Use it as a starting point, break things, and observe what changes.

1 000 MNIST digits
784 pixels each
ten classes

implement: linear AE, KL divergence, reparameterisation trick

WHAT TO OBSERVE. The linear AE elbow matches the PCA reconstruction curve, confirming that a tied-weight linear AE is PCA in disguise. The AE and VAE scatters at k=2 differ in geometry: AE encodings spread arbitrarily with large empty regions, VAE encodings concentrate near the origin and fill the unit disc. Active units saturate well below the chosen k. The interpolation passes through plausible intermediate digits, showing that the latent space is genuinely continuous.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What does the autoencoder bottleneck force the encoder to discover, and how does it relate to the PCA subspace?
- Why does a deterministic autoencoder fail as a generative model?
- In the Gaussian KL, why is a narrow posterior more expensive than a shifted or widened one?
- When would you reach for β-VAE, VQ-VAE, or a conditional VAE over the plain VAE?
- Why does the reparameterisation trick work, and what does it exploit about the Gaussian?
- What is posterior collapse, and which design lever brings it on most often?
- How does the active-unit count tell you whether k is set correctly?
- Why does sampling $z \sim \mathcal{N}(0, I)$ and decoding produce plausible new data in a VAE but not in a plain autoencoder?

RECAP of Key Concepts:

- Autoencoders learn a nonlinear θ_E, θ_D through a bottleneck; linear AE recovers PCA, nonlinear goes beyond it.
- VAEs replace the point embedding with $q(z|x)$ regularised toward a prior, giving a continuous, generative latent space.
- The Gaussian KL trades a quadratic shift cost against an asymmetric spread cost; narrow posteriors cost the most.
- β-VAE buys disentanglement, VQ-VAE buys discrete codes, conditional VAE buys controlled generation.
- The reparameterisation trick $z = \mu + \sigma \odot \varepsilon$ keeps the sample differentiable in μ and σ.
- Active-unit count diagnoses bottleneck use; KL warm-up is the standard fix for posterior collapse.
- Sampling $z \sim p(z)$ and decoding generates new data; smooth interpolation tests latent continuity.

THE ROOT OF MODEL BLINDNESS. Any model can tell you what it has learned, but not how much to trust what it says about inputs it has never seen.

Unsupervised deep learning is at root the project of modelling what counts as familiar, and the VAE does this directly: low reconstruction error and high prior likelihood under $p(z)$ mean an input resembles the training data. Yet every model trained on finite data, generative or not, has a blind spot for inputs outside its experience: a VAE's error rises on unfamiliar inputs and a classifier's softmax stays peaked on adversarial ones, but neither flags the gap in any principled way.

Uncertainty estimation turns familiarity into a principled tool, and it is itself an unsupervised problem: knowing where a model's knowledge ends depends on the learned distribution of inputs, not on a held-out label. The next chapter develops model-agnostic methods for separating irreducible noise from reducible ignorance, and uses them for anomaly detection, out-of-distribution flagging, and graceful degradation in deployment.

Teaser. How does a model trained without labels know when a new input falls outside the structure it has learned?

Feedback.

Uncertainty Estimation

2026-05-06 · cheerful mango Haubentaucher

I know that I know nothing.
The Why

In the previous chapter we learned to compress high-dimensional data into structured latent spaces and embeddings and even to draw new samples from those spaces. Closing we even explored and stepped through the latent feature space itself. Neither capability told us how much we should trust those embeddings or outputs for any particular input. A reconstruction looks the same whether the embedding lay in the heart of the training distribution or far from any encoded neighbour in feature space; the model has no built-in way to communicate its own uncertainty.

Neural networks are trained to predict, and they do so with impressive accuracy on familiar data, which makes them particularly misleading on unfamiliar data.

The Confidence Value and Prediction Probabilities are ambiguous. Neural networks calibrate poorly on out-of-distribution inputs by design: the softmax normalises to sum to one. Confidence collapses to a measure of similarity within the known domain itself and fails out-of-domain distributions. As a consequence we need to establish ways to determine the actual confidence we have in a model.

Adversarial inputs lift the problem to a sharper edge, where the model is high-confidence wrong on imperceptible perturbations.

Calibration[26]. In a supervised setting the accuracy of the model is known, and using that information the confidence can be calibrated by the actual accuracy in a specific confidence bin.

[26] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, 2017. https://arxiv.org/abs/1706.04599

Confidence Head. A complementary route is to train a small head whose only job is to predict whether the main model is actually confident to be right on the current input, turning the model's own likelihood of being correct into a prediction target.

Confidence by Reconstruction. In the spirit of the Variational

Autoencoder Architecture, the confidence in an embedding gets varified when its reconstruction gets compared with its decoding. Even when out of domain samples fall within the latent feature space of our training data, the reconstruction collapses onto the training domain, highlighting out of domain samples.

EPISTEMIC UNCERTAINTY. While uncertainty within the data, which is called aleatoric, is a concern the models own uncertainty is the focus when brought to application. Introducting noise into our learned parameters during inference, the variance in prediction result is measured and used as an indicator for uncertainty. This approach is the basis for Monte Carlo Dropout.

APPLICATION AND ACTING UNDER UNCERTAINTY. Uncertainty estimations turn confidence into a signal that informs our downstream modules.

IN ORDER TO MOVE FROM OVERCONFIDENT PREDICTORS TO UNCERTAINTY ESTIMATIONS we have good reason to find ways to,

- separate aleatoric from epistemic uncertainty, so that each source informs the predictive confidence of our model.
- measure or predict model uncertainty in out of domain distributions.
- act on uncertainty estimates in practice, routing unreliable predictions to human reviewers, falling back to fail-safe defaults, and detecting inputs the model was never trained to handle.

THE HEAVY TAIL OF ALEATORIC UNCERTAINTY. Real-world data is heavy-tailed: A small number of common situations cover most of the volume, but a long tail of rare configurations refuses to vanish no matter how much data is collected. Autonomous vehicles thus have no means to have learned those rare events, so at least they need to realize that they move out of domain.

P. Koopman. The heavy tail safety ceiling. In *Automated and Connected Vehicle Systems Testing Symposium*, 2018. https://users.ece.cmu.edu/~koopman/pubs/koopman18_heavy_tail_ceiling.pdf

ODD AND DRIFT. The operational design domain names the conditions under which a system was designed to work, namely the slice of the world the training data covered. ODD detection asks at runtime whether the current input falls inside that slice. Data drift is the slow change in input statistics over time; model drift is the resulting decoupling of stated confidence from empirical accuracy.

ANOMALY DETECTION AND ON-DOMAIN LEARNING. Making a virtue of necessity. We can use uncertainty estimates for anomaly detection on one-class embeddings: Deliberately fit only normal data, then flag any input the model finds unusual without ever showing it a labelled anomaly.

SELF-REFLECTION questions to guide your thinking:

- Why does softmax turn confidence into a similarity score within the training distribution rather than an absolute correctness estimate?
- Why is high softmax confidence on an adversarial perturbation the same failure mode as overconfidence out of domain, not a separate one?
- In a reliability diagram, why does the gap at the heaviest bin determine the user-visible failure mode of a deployed system?
- Why does histogram binning close the in-domain ECE almost completely while leaving the out-of-domain ECE almost untouched?
- How does histogram binning differ in assumption and applicability from a separately trained confidence head?
- How does MC Dropout turn a single trained network into an estimator of predictive variance, and what assumption links that variance to epistemic uncertainty?
- Which kind of out-of-domain input is structurally invisible to MC Dropout, and why?
- In a DeVries-Taylor confidence head, every hint towards the label costs $-\log c$. Why does that loss alone tend to saturate at $c \to 1$, and what does outlier exposure add that closes the loophole?
- Why can the reconstruction error of a one-class VAE flag an out-of-domain input even when its latent embedding falls inside the training cloud, and what kind of input slips through that signal?
- Why do VAE reconstruction error and VAE latent geometry tend to confuse the same out-of-domain inputs, and what does that say about geometric vs semantic OOD detection?
- What separates aleatoric from epistemic uncertainty, and which of the two is reduced by collecting more training data?
- Why does the heavy tail of real-world data imply that the epistemic component cannot be driven to zero, and what does that mean for safety-critical deployment?
- How do data drift and model drift, defined relative to the operational design domain, decouple stated confidence from empirical accuracy at runtime?

- Why can a one-class generator serve as an anomaly detector without ever observing a labelled anomaly, and what does that imply for the choice of objective?
- On what axes would you compare softmax, calibration, MC Dropout, a confidence head, and a VAE OOD score when picking one for a concrete deployment?

RECAP of Key Concepts:

- Softmax encodes similarity within the training distribution, not correctness, and so fails out of domain and on adversarial inputs.
- ECE measures the gap between stated confidence and empirical accuracy; histogram binning closes it on the distribution it was fit on but does not transfer.
- Uncertainty splits into aleatoric (irreducible noise) and epistemic (model ignorance); MC Dropout and a confidence head estimate the latter, and only the latter shrinks under more training data.
- A one-class VAE turns reconstruction error and latent geometry into OOD signals, and inherits a blind spot whenever the unknown looks geometrically like the known.
- The operational design domain delimits the input slice the model was trained for; data drift and model drift decouple stated confidence from empirical accuracy at runtime.

A NOTION OF DOMAIN. Every method in this chapter inherits its notion of domain from a narrow training distribution, and the long tail of novel inputs is structurally invisible to a narrowly-trained detector. The data covers only an increment of the input space the deployed model will encounter.

TRANSFER LEARNING. In order to build ever more capable models and expand beyond known domains, our target is to widen the feature space before a new domain or a new task even arrives.

TEASER. How do our embeddings need to be formalized and trained in order to generalize and transfer between tasks and domains?

FEEDBACK

Part III

Distilling Pattern

Transfer Learning

2026-05-06 · cheerful mango Haubentaucher

The Flywheel of Generalized Embeddings.
The Why

Machine Learning Engineer

In the Chapter on Variational Auto Encoders we trained an Encoder that mapped inputs into a structured latent space, where nearby points were similar and the bottleneck, our embedding, distilled what mattered. The space was useful, but it belonged to one model trained for one task. By the same logic, every new task would mean training that encoder again, from scratch, on a freshly and sufficiently labelled dataset.

Deep Learning Engineer

Let's reuse or transfer embeddings. Almost no production model is trained from scratch. They start from a backbone someone else pretrained at scale, freeze or adapt it, and bolt a small task-specific head on top. The encoder is the asset, the head is a thin wrapper, for a narrow task or domain, and the same backbone is shared across many downstream tasks.

Train once, reuse forever. Almost every modern vision and language system in production starts from someone else's pretrained encoder. The encoder is the expensive part, the task-specific head on top is cheap, and the choice of encoder determines almost everything that follows.

Training an encoder per task is not viable. Labels are expensive to collect, domains splinter into modalities that each deserve their own representation, pretraining a competitive backbone costs hundreds of GPU years, and entire classes of inputs have no labelled examples to begin with. The encoder therefore has to come from somewhere else and the cost of obtaining it has to be amortised across many downstream tasks.

Scarce labels. A medical imaging team may need months to label five hundred chest X-rays, while a competitive vision encoder is trained on millions.

Reuse is the mechanism. A backbone trained at scale on a broad task supplies features that work, often surprisingly well, for tasks it was never trained on. Push the scale far enough and the backbone becomes a foundation model that handles new tasks without any task-specific training at all, because its embeddings already cover the

Splintered domains. Satellite imagery, manufacturing defects, document layouts, and X-ray modalities each have their own vocabulary of inputs and in principle each deserves its own encoder.

Enormous compute. Pretraining a modern vision transformer or large language model costs hundreds of GPU years, which few teams can afford once and none can afford per task.

world that the new task lives in. Once we found that the long tail of any one task already sits inside the backbones embedding [27].

On Modalities. Foundation models such as CLIP, GPT, and SAM push this idea to the extreme: A single representation, learned once on hundreds of millions of image-text pairs or tokens, can be reused on classification, retrieval, or segmentation tasks it was never explicitly trained for, with zero or only a handful of labelled examples. Natural language is specifically applicable to task transfer as the input itself is adaptable and directs the task. The same property that makes the input the task specification is also the surface on which prompt injection attacks operate, since adversarial text inserted into the input can redirect the model to a task its operator never intended [28].

In order to move from one-task-one-model to many-tasks-one-backbone we have good reason to find ways to,

- reuse features learned at scale for downstream tasks with little or no labelled data.
- adapt pretrained representations efficiently, without paying full retraining cost.
- bridge modalities so that text descriptions can serve as classifiers for visual inputs.
- diagnose when transfer fails because source and target distributions are too far apart.

The fundamental question this chapter answers is how a model trained on one corpus can serve a task it never saw, with little or no additional supervision, and how to pick the right adaptation strategy for the data and compute budget at hand.

[27] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 8748–8763, 2021. https://arxiv.org/abs/2103.00020

[28] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 79–90, 2023. https://arxiv.org/abs/2302.12173

LoRA factors the correction as two thin matrices that slot onto a frozen layer. With W frozen, $\partial L/\partial W$ is never formed and Adam's optimiser state collapses onto the rank-r adapters. Fine-tuning just got cheap.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. https://arxiv.org/abs/2106.09685

SELF-REFLECTION questions to guide your thinking:

- Why does one shot or few shot learning on top of a frozen backbone work, when training the same network from scratch on the same handful of examples does not?
- What role do class centroids play as anchor points in the embedding, and why does averaging many embeddings into one prototype still represent the class faithfully?
- In Lampert zero shot, the bridge from attributes to centroids is fitted only on the seen classes, yet it locates unseen classes correctly. What does the bridge actually learn that lets it generalise, and what role does source breadth play in making that work?
- Why does the Lampert bridge stall when two source classes share an identical attribute row, and what does that aliasing tell you about where the ceiling on attribute based zero shot lives?
- What changes when the hand built bridge is replaced by a contrastively trained text encoder, and why is the resulting structure of two encoders meeting in a shared space the same shape as Lampert?
- Why does cosine similarity work cleanly as the decision rule in both Lampert and CLIP, and what does it imply about how the two sides of the comparison have to be normalised?
- Why does composing pretrained pieces at inference time, a frozen backbone with a learned bridge and a retrieval or clustering step, produce useful behaviour on tasks that none of the pieces was specifically trained for?
- The CLIP contrastive loss encourages the diagonal of the image caption similarity matrix to dominate each row and each column. Why does that single objective produce both the alignment of matched pairs and the separation of unmatched ones at the same time, without a separate negative term?
- Why is the classification head discarded the moment you transfer to a new task, and what does the act of throwing it away tell you about where the value of pretraining actually accumulates inside the network?
- Why does the choice between freezing the backbone and fine tuning it end up depending on both how much labelled target data is available and how far the target distribution sits from the source, rather than being a fixed best practice?

RECAP of Key Concepts:

- A backbone is a pretrained encoder reused across tasks while the head supplies only the decision, so pretraining cost is amortised across many downstream uses rather than paid each time.
- Linear probes succeed at one and few shot because the frozen embedding has already separated the classes, so the probe only has to pick a side of a hyperplane that the backbone effectively pre-drew.
- Class centroids act as anchor points in the embedding: averaging the embeddings of a class gives a single prototype, and classification reduces to nearest centroid under cosine similarity once both sides are unit normalised.
- Zero shot means reaching unseen classes through a bridge between class semantics and image features, hand built and fitted by least squares in Lampert, learned contrastively from natural language pairs in CLIP; the bridge structure of two encoders meeting in a shared space is the same in both, only the fitting criterion changes.
- Foundation models are backbones whose scale and breadth push the embedding to cover downstream input distributions, so composition of frozen pieces at inference time can replace task specific training, and transfer only fails when source and target distributions diverge beyond what the embedding describes.

THE BACKBONE ARRIVES READY-MADE. Throughout this chapter we have taken pretrained encoders as given and asked how to reuse them. ImageNet supervised pretraining works for natural images because someone else once labelled fourteen million examples, but no second ImageNet is being built for every new domain. Foundation models need to find a source of supervision that hides in plain sight.

SELF-SUPERVISED LEARNING generalises this idea. Instead of asking humans for labels, the model is given a task that can be solved using only the structure of the data itself: predict an image from itself, predict the next token from the previous ones. The labels come from the data, and the representations that emerge are the very backbones this chapter has taught us to reuse.

TEASER. Where in the data does implicit labels, cues and supervision hide?

FEEDBACK

Self-Supervised Learning

2026-05-06 · cheerful mango Haubentaucher

There is free lunch.

The Why

In the previous chapter we learned to reuse a pretrained backbone across tasks, and that foundation models extend reuse to tasks they were never explicitly trained on. This chapter generalises that idea, learning representations from raw data through a family of objectives that scale to the corpora behind today's foundation models.

Corpus. A dataset is curated and usually labelled, assembled with a specific task in mind. A corpus is the body of raw data itself, often scraped or harvested at web scale, with no task attached. Self-supervised learning is what lets a corpus take the place of a dataset.

Labels $\tilde{y}$ are the bottleneck. The internet, sensor networks, and scientific instruments produce orders of magnitude more raw data than any annotation budget can keep up with. Annotation is expensive, biased by the curator, slow to scale with model size, and tied to the specific domain that worked them out.

ImageNet absorbed years of human effort to assemble fourteen million labelled images. The open web holds a hundred billion images that no curator will ever touch. The asymmetry between labelled and unlabelled data is the structural fact that motivates this chapter.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848. https://scholar.google.com/scholar?q=Deng+Dong+Socher+Li+Li+Fei-Fei+2009+ImageNet+large-scale+hierarchical+image+database

Self-supervised learning extracts supervision from the data itself. Instead of asking a human what an image contains, we design a pretext task whose label can be read off directly from the data structure.

- Pretext tasks. The pretext task is scaffolding; the representations the network builds while solving it are the real product, and they transfer to downstream tasks the network was never trained for. The first generation of self-supervised methods built auxiliary objectives by hand: Rotate the image and predict the rotation, scramble the patches and reassemble them, fill in the colour of a grayscale photo.

- Contrastive learning. A second generation drops the hand-designed task and asks the network to recognise the same image

Chapter 6 showed one route to such a backbone without labels: The variational autoencoder distilled a representation from a reconstruction objective alone.

under different views. Two random crops of one photo should land near each other in the embedding space, while crops from different photos should land apart.

MASKED AND NEXT-TOKEN PREDICTION. The dominant pretext task at scale is to mask part of the input and ask the model to reconstruct or denoise it. This can be masked tokens in BERT [29], masked patches in MAE, or the next token in autoregressive language models.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 4171–4186, 2019. https://arxiv.org/abs/1810.04805

SPAN MASKING. Single-token masking is often trivial, since twenty visible neighbours give away the missing token. Span masking removes contiguous chunks instead, forcing the model to reconstruct multi-token structure. SpanBERT and T5 made this the default in modern encoder-decoder pretraining.

EXAMPLE. Single-token: `She slipped on a yellow [?] peel and fell over`. Span: `She slipped on a [? ? ?] and fell over`, where the contiguous chunk is the three adjacent tokens `yellow banana peel` the model has to recover together.

CAUSAL NEXT-TOKEN PREDICTION. The decoder-only language model takes the same denoising principle and applies it strictly left to right. At every position, the model sees only the prefix and predicts the next token. GPT scales this objective to internet-sized corpora, and the resulting model is a foundation model whose representations transfer to almost every language task by prompting alone.

IN THE BEGINNING WAS THE WORD. Prompt injection, reasoning, and agentic tool use.

PREDICTING IN LATENT SPACE. Reconstructing pixels, or tokens, wastes capacity on texture and lighting the downstream task does not care about. Joint-embedding predictive architectures (JEPA) move the target into feature space instead. The teacher and the student share the same encoder, the teacher's weights are an exponential moving average of the student's to stabilize targets over time, and no labels need to enter the pipeline at any point. On every step the teacher runs a forward pass on the unmasked input and produces target representations of the masked region. The student sees a masked view of the same input sample, predicts those target representations in the latent space, and the loss is the distance between predicted and target features. There is no decoder and pure prediction happens in the encoder's feature space, so the method is efficient and scales well to large models and large corpora.

CHOOSE BY REGIME. Architecture tracks the downstream task: Encoder-only for understanding, decoder-only for generation, encoder-decoder such as T5 and BART for both at doubled cost.

IN ORDER TO MOVE FROM EXPENSIVE MANUAL ANNOTATION TO LEARNING FROM RAW DATA we have good reason to find ways to,

- design auxiliary tasks whose supervision signal is inherent in the data itself, so that scale is bounded by storage rather than by human effort.
- learn representations that capture semantic structure without any

DINO predates and inspires the JEPA family. A momentum teacher produces soft targets in feature space, the student matches them with cross entropy, and no negatives or labels are needed. Self-supervised attention maps emerge that segment objects without supervision.

M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9650–9660, 2021. https://arxiv.org/abs/2104.14294

human labels, transferable across downstream tasks the model was never explicitly trained for.

- recognise denoising as the unifying objective behind masked-token, masked-patch, and next-token prediction.
- match the masking pattern to the downstream regime, choosing between encoder-only, decoder-only, and encoder-decoder recipes when the task is understanding, generation, or both.
- predict in feature space rather than pixel space when raw reconstruction would spend capacity on signal the downstream task does not care about.

THE FUNDAMENTAL QUESTION this chapter answers is whether the structure of unlabelled data carries enough signal to teach a model what to look at, without any human ever telling it what to value.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- What goes wrong when a pretext task is too easy, and how can the network exploit shortcuts that solve the task without producing useful representations?
- Why does the contrastive recipe of treating different views of the same input as similar, and views of different inputs as dissimilar, give a stronger signal than handcrafted pretext tasks such as rotation prediction?
- Why is the choice of what counts as a different view the design decision that determines which features the contrastive representation will treat as invariant?
- Why is denoising a useful frame for thinking about masked-token, masked-patch, and next-token prediction together rather than as separate methods?
- Where does the supervision signal come from in masked prediction?
- How does the architectural choice between encoder-only, decoder-only, and encoder-decoder follow from whether the downstream task is understanding, generation, or both?
- Why does the encoder-decoder recipe end up paying for both halves of the bargain in architecture cost?
- Why does masking a contiguous span force the model to learn more than masking the same number of scattered single tokens?
- Why is moving the prediction target from pixel space to feature space a useful trade-off when raw reconstruction would spend capacity on signal the downstream task does not care about?
- What role does the slow-moving averaged teacher play in producing a stable target the student can predict against, and why is the same loss not workable when the teacher equals the student at every step?
- Why is the latent-space prediction recipe still considered self-supervised even though there is a teacher network feeding the student?
- Across all the methods in this chapter, what is the common idea that lets each of them produce useful representations from raw data with no human labels in the loop?

Recap of key concepts:

- Self-supervised learning derives its supervision from the structure of the data itself, with the pretext task as scaffolding and the learned representation as the product.
- Pretext tasks evolved from handcrafted recipes such as rotation prediction and patch scrambling toward principled relational objectives such as contrastive learning over augmented views.
- Denoising unifies the dominant masked-prediction family: Hide part of the input as masked tokens, masked patches, contiguous spans, or the next token in a sequence, and train the model to reconstruct what was hidden.
- The architectural choice between encoder-only, decoder-only, and encoder-decoder tracks the downstream regime, with encoder-only for understanding, decoder-only for generation, and encoder-decoder for both at the cost of doubled architecture.
- Predicting in latent space sidesteps the irrelevant signal that raw pixel reconstruction would spend capacity on, with an exponential moving average teacher providing a stable target the student can predict against in feature space.

Teaser. A self-supervised backbone gives you a head start, not a finished classifier; the downstream task still wants labels. What signals and cues can you think of to put in the place of supervision.

Self-supervised learning softened the annotation bottleneck from one side. Instead of asking humans to label more data, the chapter designed objectives and tasks that read supervision off the data structure itself, and the resulting representations thus can be transferred to downstream tasks the model was never explicitly trained for. The bottleneck has another side, where supervision for dedicated downstream tasks still arrives, from humans, from auxiliary sensors coupled to the input, or from larger models standing in as labellers, but in the form of rough rules, heuristics, proxy signals, or noisy stand-in labels rather than clean per-example annotations.

The next chapter turns to weak supervision, where a label model learns to combine the noisy and conflicting rule outputs into a single training signal. Self-supervised learning trusted the structure of the data; weak supervision trusts the structure of the rules; both share the conviction that some signal is a signal.

feedback

Weak Supervision

2026-05-06 · cheerful mango Haubentaucher

A weak signal, is still a signal.
The Why

In the previous chapters we have seen how to represent generalizable structure off the data, and defaulted to small clean head bolted on the embedding. This chapter turns to the head itself, asking what to do in the absence of clean labels.

Weak labels are drawn from heterogeneous sources, each with its own coverage and reliability profile. Crowd annotations exhibit substantial inter-annotator disagreement. Heuristic rules attain high precision on cases anticipated by their author and provide low coverage elsewhere. Large Pretrained classifier models afford broad coverage but are systematically overconfident on inputs lying outside the training distribution. Physical sensor measurements and behavioural proxies serve as auxiliary signals, correlating with downstream events of interest without themselves constituting labels, yet both can be folded into the pipeline as labelling-function outputs. The composition of these sources varies by modality, with text, image, sequence and tabular settings each exhibiting characteristic patterns. A unifying abstraction subsumes all of them: A labelling function is a programmatic mapping from input to a label. Under this representation, disparate sources reduce to a single object class amenable to downstream or bidirectional aggregation.

A judge is a labelling function that exploits the verifier-generator asymmetry. Discriminating a good answer from a poor one is structurally easier than producing a good one from scratch, the same gap that powers GAN discriminators and RLHF reward models. A language model judge consumes a candidate output and returns a scalar or categorical verdict [30], sitting alongside symbolic

Labels are random variables $\tilde{y}$. Not ground truth facts, just noisy realisations of an underlying truth y that we never observe directly. The annotation pipeline is a measurement device, and like every measurement device it has a bias and a variance.

Heuristics as benchmarks. Cheap to author, interpretable in their failures, and immune to training-distribution shift, heuristic baselines establish the performance floor a learned method must clear. The gap between a heuristic and a learned model quantifies whether the additional complexity is justified.

Sensors and proxies. Sensors such as vibration, touch, torque, current draw, lidar and you name it serve as proxies for machine failure, anomaly onset, or obstacle presence. Behavioural signals such as clicks, dwell times, completion rates, and purchase records serve as proxies for relevance, engagement, or user preference.

Die Ode. Try writing a four-line ode to a banana in the manner of Schiller. Then judge whether this one is any good:
Sei mir gegrüßt, du krumme Frucht,
in Gold und Schweigen eingehüllt;
es trägt die Reife stille Wucht,
bis süß dein Schicksal sich erfüllt.

[30] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks*, 2023. https://arxiv.org/abs/2306.05685

and statistical labelling functions under the same uniform interface. Judges offer lower cost than human evaluation and broader coverage than regular-expression rules, while inheriting the calibration and stylistic biases of the underlying model. Process reward models generalise the construction to intermediate reasoning steps, scoring each step of a chain rather than only the final answer and so spreading supervision along the whole trajectory instead of concentrating it at the end.

AUXILIARY TASKS construct intermediate and spreaded supervision. If the main task is hard to label, find an easier auxiliary task that shares structure with it and train on that instead.

GENERATORS ARE LARGE MODELS THAT PRODUCE LABELLED EXAMPLES DIRECTLY. Where a judge discriminates, a generator synthesises both input and label. The canonical case is knowledge distillation: A teacher's softmax supplies richer supervision than a one-hot target, and a smaller student is trained to match it. The same recipe underlies synthetic-data and instruction-tuning pipelines at scale.

SNORKEL. Three moving parts: Engineers write labelling functions in Python that each vote on a data point or abstain, a label model learns the accuracies and correlations of those functions to turn their votes into a probability distribution over the true label, and a discriminative end model is trained on those probabilistic labels and generalises beyond the rules.

A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment*, volume 11, pages 269–282, 2017. https://arxiv.org/abs/1711.10160

SOFT LABELS ARE THE NATURAL OUTPUT OF WEAK SUPERVISION. A soft label assigns a probability to each candidate class, preserving inter-class structure that an argmax would discard. In Snorkel-style pipelines this distribution is the posterior of the label model, which sees only the labelling-function outputs and never the inputs themselves. The end model, a high-capacity discriminator, trains on those soft posteriors with full access to the inputs but never sees the labelling functions, and so generalises to inputs on which no labelling function ever fired. The same soft-label construction recurs across judges, distillation, and label smoothing, where the source of the distribution differs but the training objective is identical.

ACTIVE SUPERVISION. The supervised complement to weak supervision: A small budget of clean labels is spent on the inputs from which the model stands to gain the most . Strictly supervised learning, but with the selection strategy as the lever. Acquisition criteria such as uncertainty sampling, query-by-committee, and expected model change rank candidates by anticipated information gain, and an annotator is queried only for the highest-ranked.

B. Settles. *Active Learning*. Morgan & Claypool Publishers, 2012. https://scholar.google.com/scholar?q=Settles+2012+active+learning+synthesis+lectures

IN ORDER TO MOVE FROM THE FICTION OF GROUND TRUTH LABELS TO PRINCIPLED LEARNING UNDER NOISE we have good reason to find ways to,

- collect rules, crowd annotations, pretrained classifiers, and sensor proxies behind one labelling-function interface.
- exploit the verifier-generator gap with language-model judges as discriminators over candidate outputs.
- use larger models as generators, distilling a teacher's softmax into a smaller student.
- aggregate noisy votes through a label model and train an end model on the resulting soft posteriors.

THE FUNDAMENTAL QUESTION this chapter answers is how to compose imperfect sources into supervision a model can trust.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- Why are labels best treated as random variables $\tilde{y}$ rather than as ground truth, and what does that reframing buy you when you design a training pipeline?
- Why is it useful to treat heterogeneous label sources, from heuristic rules and crowd annotations to pretrained classifiers and sensor proxies, behind a single labelling-function interface, and what does the abstraction hide?
- When does a sensor measurement or a behavioural signal qualify as a labelling function, and when does the correlation it relies on fall apart?
- What is the verifier-generator gap, and how do GAN discriminators, RLHF reward models, and LLM-as-judge all sit on the same asymmetry?
- What calibration and stylistic biases does an LLM-as-judge inherit from its base model, and what does that mean for using it as a labelling function at scale?
- How does a process reward model spread supervision along a reasoning chain rather than concentrating it on the final answer, and why does that matter for training?
- Why does knowledge distillation transfer more information through a teacher's softmax than through one-hot labels, and where does the same recipe show up at scale?
- How does the label model in Snorkel recover each labelling function's accuracy and correlation structure without ever observing the true label?
- Why does the end model generalise to inputs on which no labelling function ever fired, and what would break this property?
- What do label smoothing, distillation, and label-model posteriors share, and where do they differ?
- Across rules, judges, distilled teachers, and label models, what is the common move that turns noisy partial signal into supervision a model can trust?

RECAP of key concepts:

- Labels are random variables, not ground truth; any noisy source, heuristic rule, crowd worker, pretrained classifier, sensor measurement, or behavioural proxy, is unified behind one labelling-function interface.
- Judges exploit the verifier-generator gap: discriminating a good output from a poor one is structurally easier than producing a good one, and LLM judges, GAN discriminators, and RLHF reward models all sit on the same asymmetry. Process reward models extend the construction to intermediate steps, scoring each step of a reasoning chain so that supervision is spread along the whole trajectory rather than concentrated on the final answer.
- Generators turn larger models into labellers; knowledge distillation transfers a teacher's softmax to a smaller student, and the same recipe powers synthetic-data and instruction-tuning pipelines.
- The label model aggregates labelling-function votes into a probabilistic posterior over the true label, sees only the votes and never the inputs, and a high-capacity end model trains on those soft posteriors with full input access and so generalises beyond the rules.
- Soft labels recur across label-model posteriors, distillation, and label smoothing; the source of the distribution differs, the training objective is identical.

TEASER. How does the supervision problem change when model interacts with an environment and receives supervision not as per-example labels but as action-level signals such as demonstrations, feedback, corrections, or rewards rather than per-example labels?

WEAK SUPERVISION INTERFACED THE ANNOTATION BOTTLENECK FROM NOISY PARTIAL SUPERVISION. Instead of deriving supervision from the data structure as self-supervised learning did, the chapter accepted noisy labels from rules, crowds, sensors, judges, and larger models, and combined it through a label model into soft posteriors that an end model could train against. Throughout, the auxiliary model was a labelling model: It answered what class an input belonged to and nothing else.

THE NEXT CHAPTER TURNS TO INTERACTIVE AND REINFORCEMENT LEARNING, where the model's predictions turn into actions which interact with an environment, which proactively influences the supervision cues it receives. The supervision is no longer a static label per input, but a dynamic signal that depends on the model's behaviour and the environment's response.

FEEDBACK

Part IV

Interaction Pattern

Reinforcement Learning Fundamentals

2026-05-06 · cheerful mango Haubentaucher

Playing Atari.
The Why

In the previous chapter supervision still arrived as labels, but a label on a single input cannot transmit a sequence of decisions or the consequences of one action on the next. Reinforcement learning answers a complementary question: What if the supervision is a scalar reward and depends on the sequence of actions a model takes?

Markov decision processes. The setting is formalised [31] as a tuple of a state s, an action a, a transition probability $P(s' \mid s, a)$, a reward $r(s, a)$, and a discount factor $\gamma \in [0, 1)$. The Markov property requires that the next state depends only on the current state and action, which lets the agent reason forward without dragging the past along, while γ downweights rewards t steps in the future by γ^t, keeping the sum of returns finite over infinite horizons.

Bellman equations. The value of a state under policy π is the expected discounted return when starting in s and following π from then on. Bellman's insight is that this value satisfies a local recursion: The value of a state equals the expected immediate reward plus the discounted expected value of the next state. This turns long-horizon optimisation into a fixed-point problem which suits Markovian dynamics.

Value-based methods learn what each action is worth. A Q-learning agent maintains an action-value table and bootstraps each estimate against the next, converging to the optimum even when behaviour is off-policy. The Deep Q-Network, DQN, scales the table to a neural network for high-dimensional inputs, which means a single set of weights now produces the value for every state-action

Reward. A scalar number returned by the environment after an action, carrying no information about which action would have been better, no gradient, no class identity, only a magnitude and a sign. Let's extract signal yet again.

Simulation as substrate. Most reinforcement learning happens inside a simulator, since collecting millions of transitions on real hardware is prohibitive. The simulator's reward and transition function are perfect inside the sim and biased outside it, and the resulting sim-to-real gap is the central practical concern.

[31] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961. https://scholar.google.com/scholar?q=Bellman+1961+adaptive+control+processes+guided+tour

Pressure One: Credit assignment. The reward arrives long after the action that helped earned it. Bellman decomposition turns the long-horizon problem into a local fixed-point one.

Model and policy. A model θ is the bag of parameters that gradient descent updates. A policy π is the behaviour rule the agent acts under, mapping states to actions. The two coincide in policy-gradient methods, where θ directly parameterises π_θ, but separate in value-based methods, where θ parameterises a value function Q_θ and the policy is derived from it as greedy or ε-greedy.

pair, and updating the weights for one state changes the values for all of them. Two instabilities follow.

Replay Buffer First, the agent's experience is a stream of correlated transitions: Each new state is the consequence of the previous one, and the agent often spends long stretches in similar regions of the environment. A network trained directly on this stream overfits to whatever region the agent is currently in and forgets what it learned about earlier ones. The experience replay buffer stores past transitions in a large pool and samples minibatches uniformly from it, so each gradient step sees a mix of transitions from many different points in time. Notice that this quickly becomes a compute bottleneck and does not scale.

Target Network Second, the bootstrapped target uses the same network whose weights we are updating, so each update changes both the predictions and the targets they are regressing against, and convergence is unstable because the network is chasing a target that moves with it. A target network is a slow-updating copy used only to compute that target, kept frozen for thousands of steps so the online network has a stable regression problem to converge to before the target shifts.

Pressure Two: Stability. Bootstrapped value estimates and policy gradients can amplify their own noise. Replay buffers, target networks, and trust regions stop the loop from running away.

Policy-based methods learn the action rule directly. A value-based method learns what each action is worth and then acts greedily, deriving the policy from the value estimates. A policy-based method skips the value function and parameterises a stochastic policy $\pi_\theta(a \mid s)$ that outputs the probability of each action directly. This also allows for continuous action spaces, where a value-based method regresses. The policy gradient theorem turns the long-run expected return into a quantity we can backpropagate against, and applies it by rolling out a trajectory, observing the return, and nudging π_θ to make the actions that led to high return more likely and those that led to low return less likely.

Off-policy vs on-policy. Q-learning and DQN learn from data collected under any behaviour, including a replay buffer of past transitions. Policy-gradient methods need data collected under the current policy, which is why trust regions matter: They control how far the new policy can drift before the old data stops applying.

Trust regions and clipping. Policy-gradient methods are on-policy: The gradient is computed under the old policy, so if a single update moves the new policy too far the data no longer applies and performance collapses. Trust-region methods address this by explicitly constraining the size of the policy update, which requires a second-order optimisation step that is expensive to compute. Proximal Policy Optimisation, PPO [32], replaces the explicit constraint with a much simpler clipping trick. Each action in a trajectory has a probability under the old policy and a probability under the new policy,

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. https://arxiv.org/abs/1707.06347

and their ratio measures how much the policy has shifted on that action since the last update. PPO clips this ratio to a narrow band around one, typically $[0.8, 1.2]$, before plugging it into the policy-gradient update. The width of the band opts for stability over sample efficiency.

Pressure Three: Exploration. Without enough variety in the data, the agent never sees the actions whose value it has yet to estimate. Exploration is the open subfield that supplies that variety.

Trade-off Exploration and Exploitation. Every method above needs a behaviour rule for collecting data, and that rule has to balance two pressures. Exploit what is currently known to be good so reward is collected, explore actions that are uncertain so the value estimates can improve. ε-greedy, and entropy bonuses on the policyare the instruments that show up everywhere in practice, and the design of better exploration is an open subfield in its own right.

Atari and Go. A DQN agent learned to play forty-nine Atari games from raw pixels and a single score signal . The score arrives at the end of a life or an episode, not after each frame, and the agent has to figure out by itself which of the thousands of intermediate joystick presses earned it.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. DOI: 10.1038/nature14236. https://scholar.google.com/scholar?q=Mnih+2015+human+level+control+deep+reinforcement+learning

In order to move to learning from a scalar, model dependant reward we have good reason to find ways to,

- formalise sequential decision problems as Markov Decision Processes and decompose long-horizon value into a local Bellman recursion.
- learn action values by bootstrapping each estimate against the next, and stabilise the neural-network case with a replay buffer for decorrelated samples and a target network for a fixed regression target.
- parameterise a stochastic policy directly and update it by rolling out trajectories and reweighting actions according to the returns they led to.
- constrain how far each policy update is allowed to move from the previous one, so that the data collected under the old policy still informs the new one.
- collect informative experience under uncertainty, balancing exploitation of known reward against exploration of unfamiliar actions.

The fundamental question this chapter answers is whether a single scalar reward, delayed and sparse, carries enough signal to teach an agent how to act, without any teacher ever specifying what the right action would have been.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- Why does the Markov property let the agent reason about the future without dragging the past along, and what would change if the next state depended on the full history?
- Why does the discount factor γ exist at all, and how does $\gamma < 1$ keep the sum of returns finite over infinite horizons?
- How does the Bellman recursion turn a long-horizon optimisation into a local fixed-point problem?
- What does it mean for Q-learning to be off-policy, and why does that property let DQN learn from a replay buffer of past transitions?
- Why does scaling the Q-table to a neural network introduce instabilities that the tabular case did not have, given that one set of weights now produces values for every state-action pair?
- What problem does the experience replay buffer solve, how does uniformly sampling past transitions from a large pool address it, and where does the approach run into a compute bottleneck?
- Why is a target network needed on top of the replay buffer, and what goes wrong when the target moves with the network being trained?
- How is a policy-based method different from a value-based method, why does parameterising $\pi_\theta(a \mid s)$ directly let you skip learning the value function, and how does this open the door to continuous action spaces?
- Why is the policy gradient theorem the key technical move that lets us treat the long-run expected return as a quantity we can backpropagate against?
- Why are trust regions and PPO clipping a policy-gradient concern only, and not relevant for Q-learning?
- Why does PPO clip the ratio between new and old policy probabilities to a narrow band such as $[0.8, 1.2]$, and how does the width of that band trade stability against sample efficiency?
- Why does every method in this chapter need a behaviour rule for exploration, and what role do ε-greedy, entropy bonuses on the policy, and stochastic policies play as the three blunt instruments?

RECAP of key concepts:

- Markov decision processes formalise sequential decision making as the tuple (s, a, P, r, γ) with the Markov property: the next state depends only on the current state and action, and γ downweights rewards t steps in the future by γ^t to keep returns finite over infinite horizons.
- The Bellman equations decompose the value of a state into immediate reward plus discounted value of the next state, and almost every method in this chapter is a way to solve that local fixed-point problem under different practical constraints.
- Value-based methods learn what each action is worth and act greedily; Q-learning is off-policy and converges in the tabular case, and DQN scales it to neural networks with an experience replay buffer that mixes correlated transitions and a target network that holds the regression target still, though replay sampling becomes a compute bottleneck at scale.
- Policy-based methods skip the value function and parameterise a stochastic policy $\pi_\theta(a \mid s)$ directly, which also unlocks continuous action spaces where value-based regression struggles; the policy gradient theorem provides a backpropagatable surrogate, and PPO bounds the on-policy update by clipping the probability ratio between new and old policy to a narrow band around one.
- Exploration supplies the data variety the methods above consume; ε-greedy, entropy bonuses on the policy, and stochastic policies are the three blunt instruments, and better exploration is an open subfield in its own right.

TEASER. Bellman decomposition makes long-horizon credit assignment tractable in principle. In practice, sparse rewards mean infeasible many steps before any informative signal arrives. What signal would shortcut all that?

REINFORCEMENT LEARNING'S HARDEST PROBLEM IS LONG-HORIZON CREDIT ASSIGNMENT. Bellman decomposition reduces it to local updates, but across sparse, long horizons it arrives too rarely to learn from on any practical budget.

THE NEXT CHAPTER TURNS TO INTERACTIVE AND IMITATION LEARNING, where we compress the long-horizon credit-assignment problem into a single demonstration, correction, or preference. Reinforcement learning derived its supervision from environmental reward; interactive learning derives it from few-shot demonstrations, corrections, or preferences from an annotator.

FEEDBACK

Index