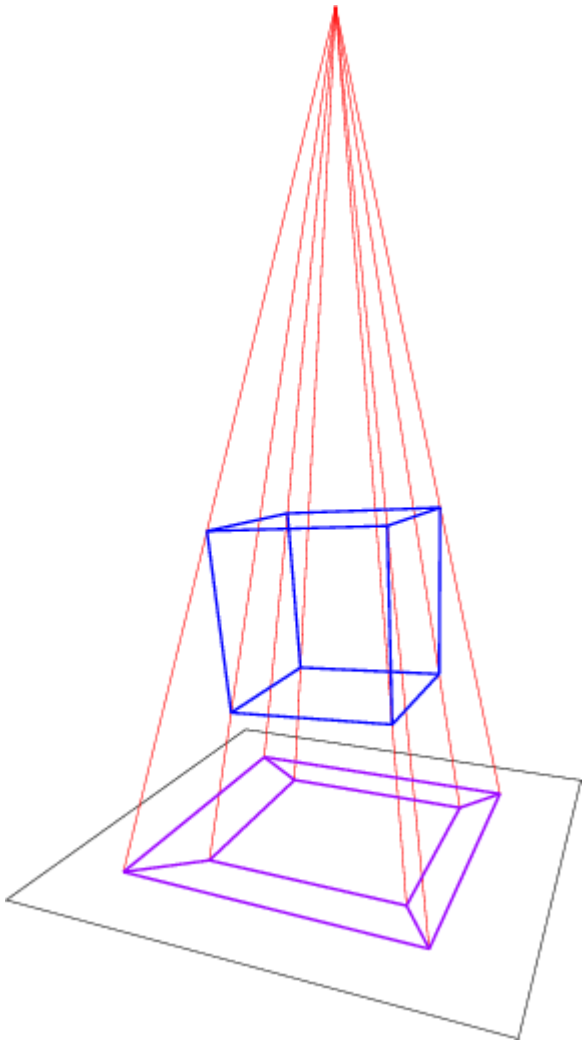
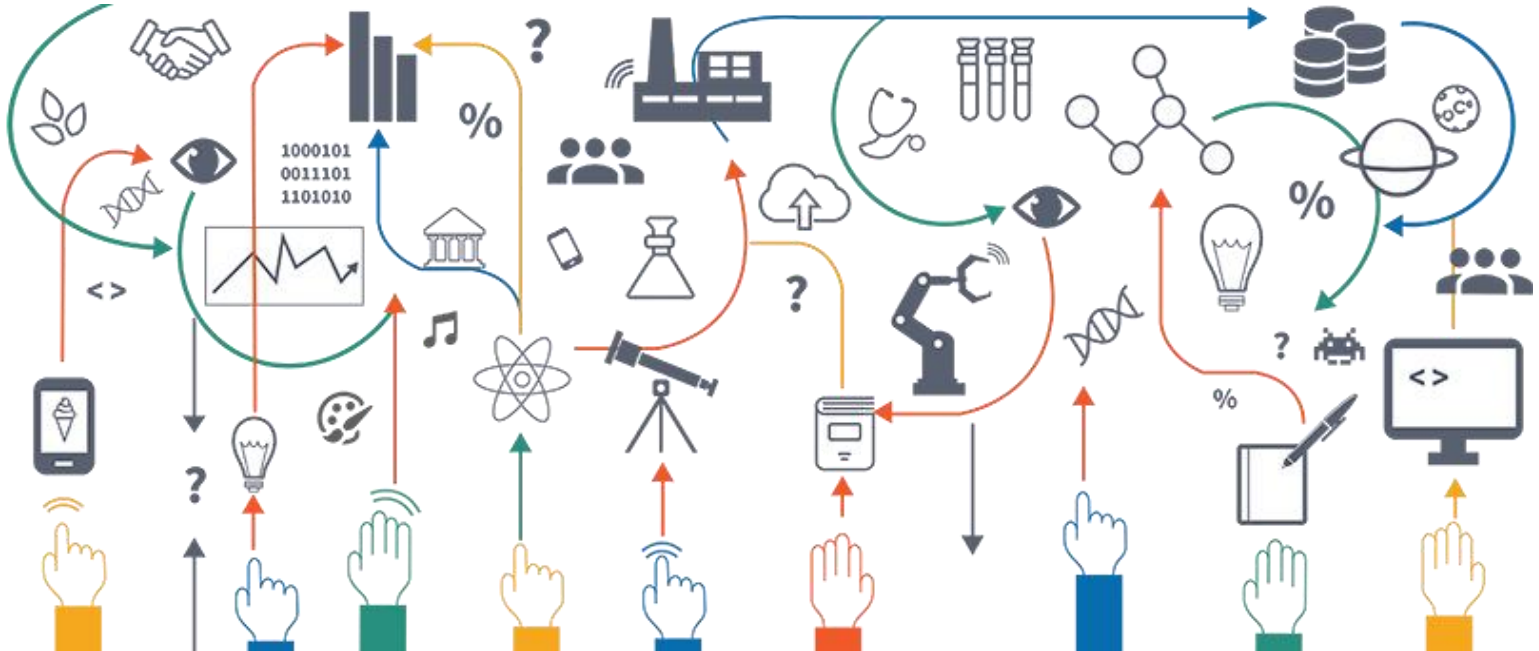
2 – Analysing High-Dimensional Data

Mathematics of Data Science



Lecturer: Dominik Dold



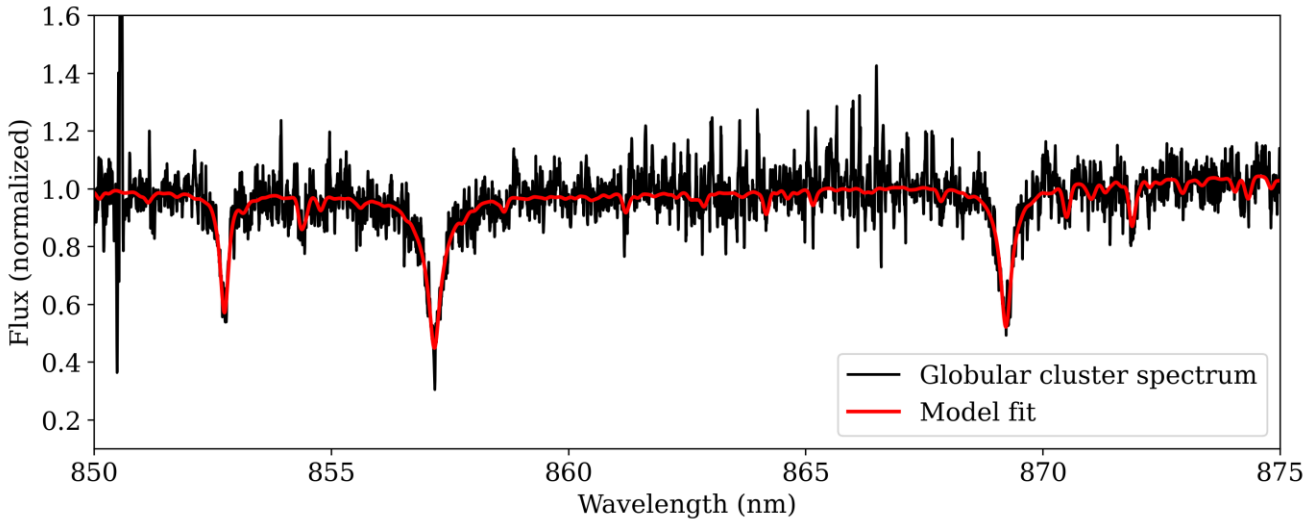
University of Vienna, WiSe 2025

Master's programme in Data Science

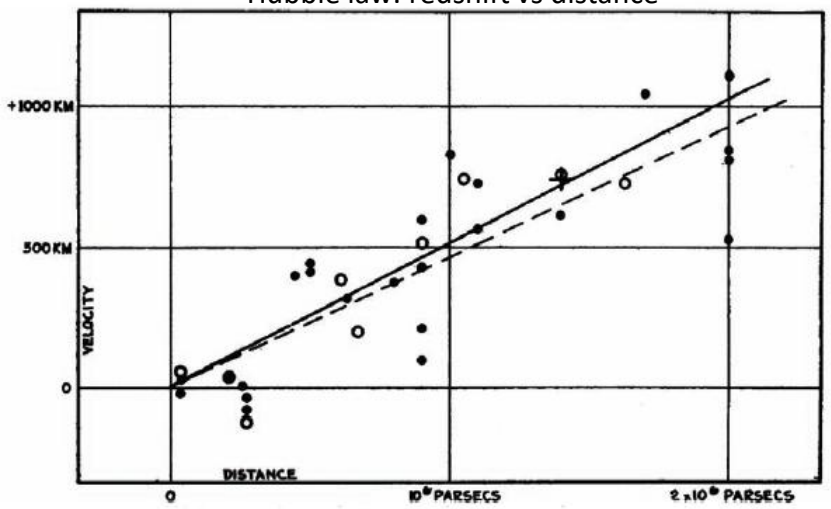
Credit: https://math.stackexchange.com/q/2286226

Why did we look so much at statistics?

One reason: data can be interpreted as samples from an underlying distribution!



Hubble law: redshift vs distance



Our data follows an underlying distribution $p(\boldsymbol{x})$,
and measurements are samples from that distribution!

Usually two cases are distinguished: $p(\boldsymbol{x})$ and $p(\boldsymbol{y}, \boldsymbol{x})$

data labels

In the following, we start with the former!

Image: Hubble, Proceedings of the National Academy of Sciences, 1929, 15, 168

Motivation: making sense of (unlabeled) data

In this lecture block, we will approach the following three challenges:

1. **The data is very high-dimensional**
   - Very unintuitive domain!
   - Hard to visualize!
   - Hard to tell which features are actually important (contain information)
2. **We only have a finite number of samples**
   - Noise vs. signal!
3. **We have no labels**
   - How can we detect useful structure in the data?

More examples of such data *(taken from https://database.eohandbook.com/)*:

1 2 3 4 5 6 7 8 9 10 ...

Instrument	Agency	Missions	Status	Type	Measurements & applications	Technical characteristics	
3MI Multi-Viewing Multi-Channel Multi-Polarisation Imaging	EUMETSAT (ESA)	Current: METOP-SG A1 Future: METOP-SG A2, METOP-SG A3 Complete: -	Being developed	Atmospheric chemistry	Measure aerosol parameters, air quality index, surface albedo, cloud information	Waveband:	VIS-SWIR: 12 channels between 0.41 µm to 2.1 µm VIS, NIR, SWIR
						Spatial resolution:	4km
						Swath width:	2200x2200 km for the VNIR channels 2200 x 1100 km for the SWIR channels
						Accuracy:	
						Data Access:	Open Access
						Data Format:	net-CDF4

1 2 3 4 5 6 7 8

Mission	Agency	Status	Launch date	EOL date	Applications	Instruments	Orbit details & URL
ALTIUS Atmospheric Limb Tracker for Investigation of the Upcoming Stratosphere (ALTIUS)	ESA	Approved	Apr 2027	Dec 2030	Altius carries a high-resolution 2D imager that observes ozone from the side, at Earth's limb or atmospheric boundary, to monitor its distribution and evolution in the Earth's atmosphere	ALTIUS Instrument	Type: Sun-synchronous Altitude: 668 km Period: Inclination: 98.07 deg Repeat cycle: 3 days LST: Fixed between 10:00 – 14:00 Longitude (if geo): Asc/desc: Descending Links: mission site | data access

Content

- The curse and blessing of high dimensions
- Eigenvalue decomposition, Determinant
- Principal component analysis
- Marcenko-Pastur distribution
- Johnson-Lindenstrauss Lemma
- K-Means clustering
- DBSCAN
- Graphs
- Spectral clustering
- Diffusion Maps

“Fun in high dimensions” quiz: question 1

Assume you are peeling an N-dimensional orange (N-dimensional unit sphere).
Which statement is true if $N \gg 3$?

A) After peeling, most of the orange is gone.
B) After peeling, most of the orange is left.

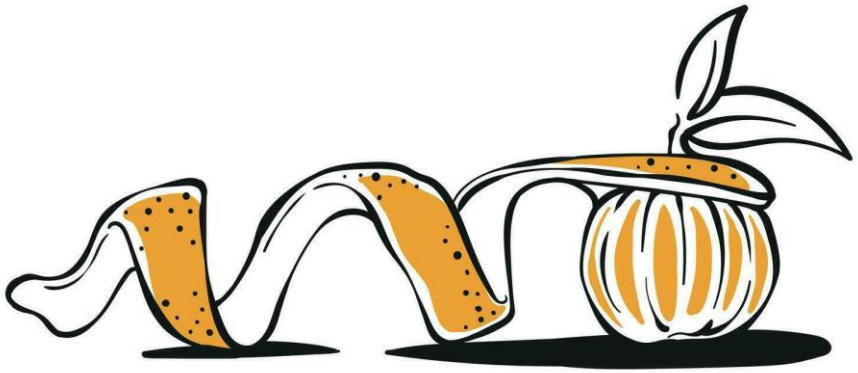
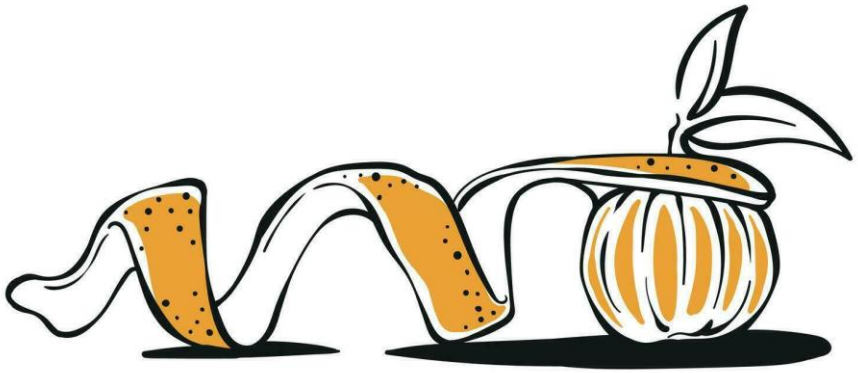


Image: https://www.vecteezy.com/vector-art/28271475-orange-fruit-peeling-skin-illustration-with-leaves-sketch-drawing-vector-design, August 2025

“Fun in high dimensions” quiz: question 1

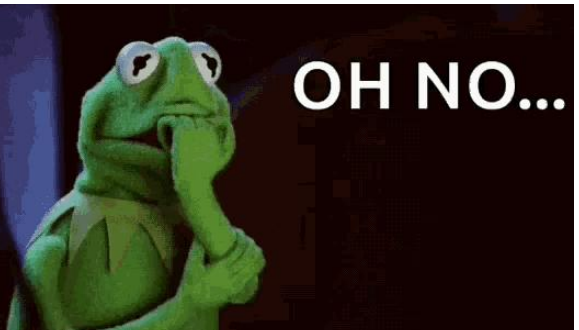
Assume you are peeling an N-dimensional orange (N-dimensional unit sphere).
Which statement is true if $N \gg 3$?

A) After peeling, most of the orange is gone.
B) After peeling, most of the orange is left.



Volume of a hypersphere: $V_N(r) = \frac{2\sqrt{\pi}^N}{N \cdot \Gamma\left(\frac{N}{2}\right)} \cdot R^N, \qquad \Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \mathrm{d}t = (z-1)!$

Gamma function



Volume ratio near surface: $\frac{V_N(1-\epsilon)}{V_N(1)} = (1-\epsilon)^N \to 0$

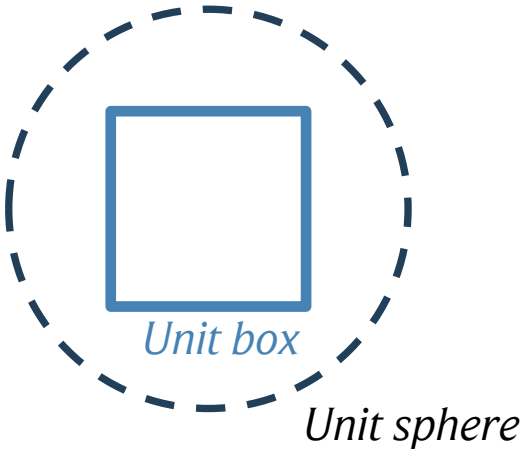
Bonus fact:
for $N \gtrsim 20,\ V_N(1) \to 0$

for $0 < \epsilon \ll 1$ and $N \to \infty$

“Fun in high dimensions” quiz: question 2

Assume a little unit square (2D box) contained in the unit circle (2D sphere).
What happens if you increase the dimension?

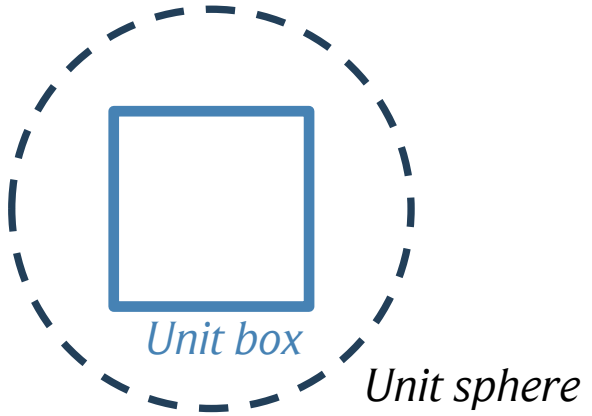
A) The box stays inside the sphere.
B) The box spills out of the sphere.
C) Both occupy the same volume.



“Fun in high dimensions” quiz: question 2

Assume a little unit square (2D box) contained in the unit circle (2D sphere).
What happens if you increase the dimension?

A) The box stays inside the sphere.
B) The box spills out of the sphere.
C) Both occupy the same volume.



We can show this using statistics!

1) Sample a random point z using a uniform distribution $U([-0.5, 0.5]^N)$.

2) We seek the probability of z to be inside the sphere: $P(\|z\|_2^2 < 1)$ in the limit of large $N \gg 12$.

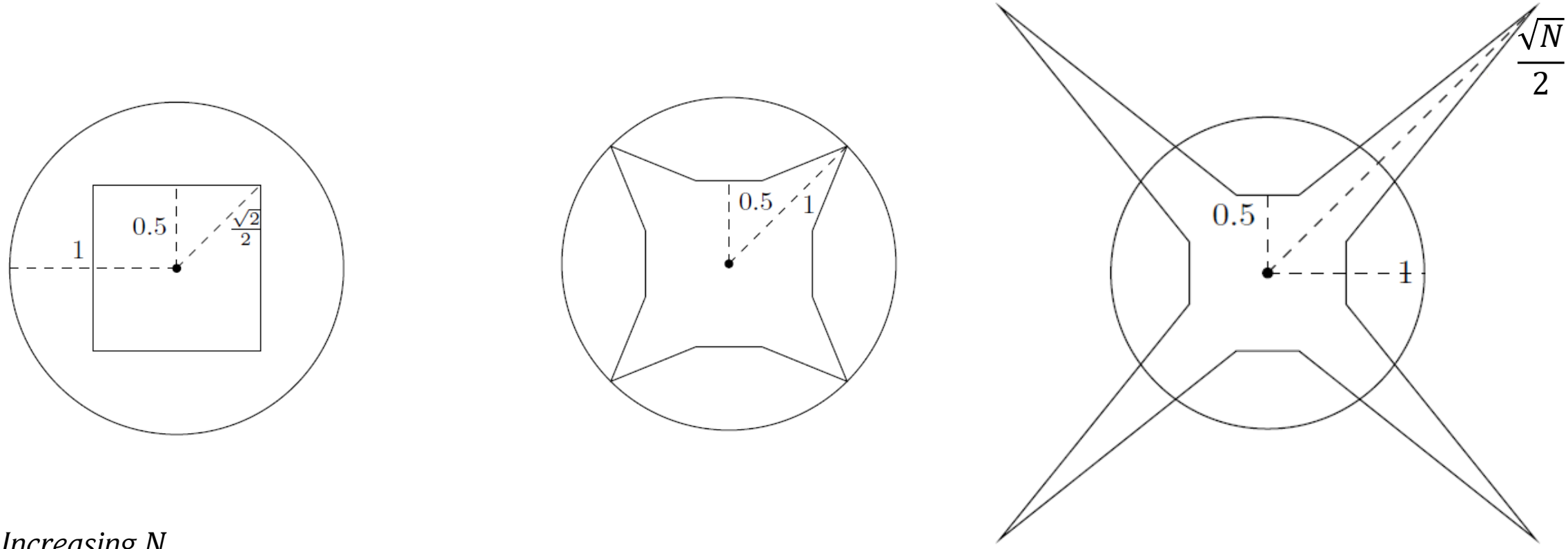
3) This is given by the one-sided **Hoeffding inequality** using $\mathrm{E}(z_i^2) = \frac{1}{12}$

$$P(\|z\|_2^2 \leq 1) = P\left(\sum_i z_i^2 \leq 1\right) = P\left(\frac{1}{N}\sum_i (z_i^2) - \mathrm{E}(z_0^2) \leq \underbrace{\frac{1}{N} - \frac{1}{12}}_{<0}\right) \leq e^{-\frac{16N}{72}}$$

“Fun in high dimensions” quiz: question 2

Thus, for large N almost all points end up **outside** of the sphere!

Why? Mid-points of sides (e.g., $(0.5,0,0,0,0)$ for $N = 5$) are within the sphere,
but corners (e.g., $(0.5,0.5,0.5,0.5,0.5)$) are outside! **Thus, most volume is in the corners!**



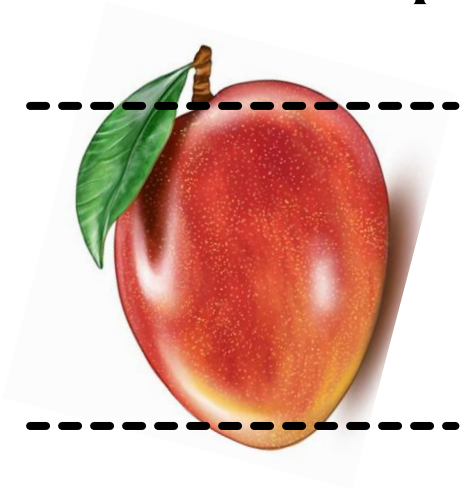
Images: Bandaira-Singer-Strohmer, Mathematics of Data Science

“Fun in high dimensions” quiz: last question

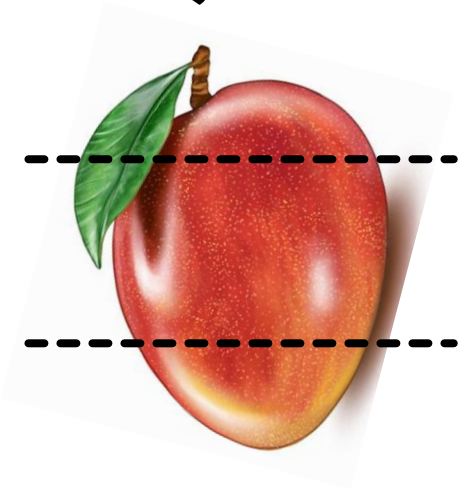
Let’s assume a d-dimensional, spherical Mango.

How much of the top and bottom can you cut off until most of the Mango is gone?

A)



B)



C)

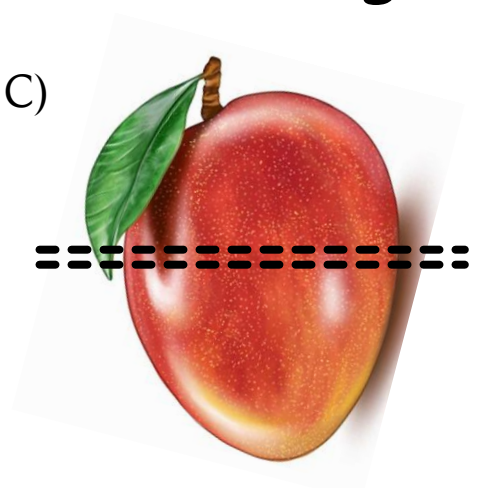
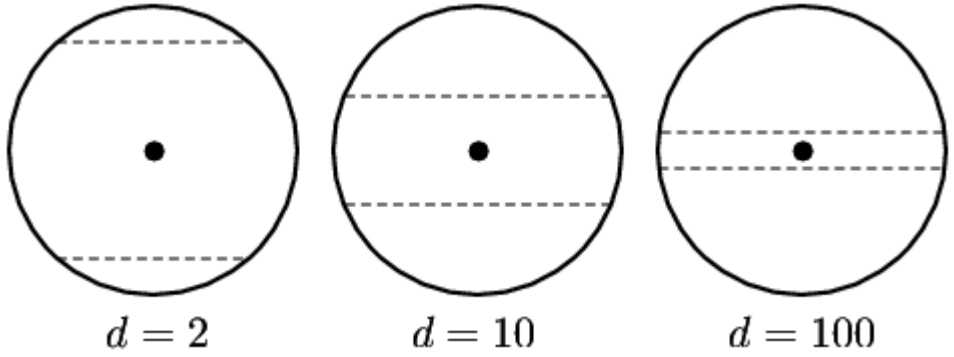


Image: https://artincontext.org/how-to-draw-a-mango/, September 2025

“Fun in high dimensions” quiz: last question

Let’s assume a d-dimensional, spherical Mango.

How much of the top and bottom can you cut off until most of the Mango is gone?



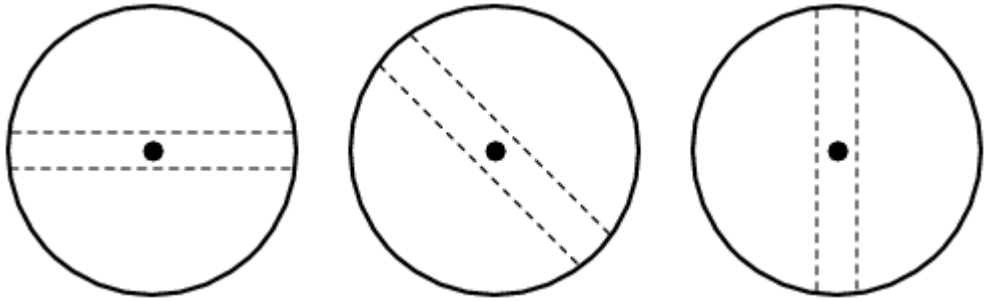
C)

It is, in fact, C)!

Most volume of a d-dimensional sphere with radius r is within distance $\frac{r}{\sqrt{d}}$ of the equator!

Extra weirdness (don’t ask me!):

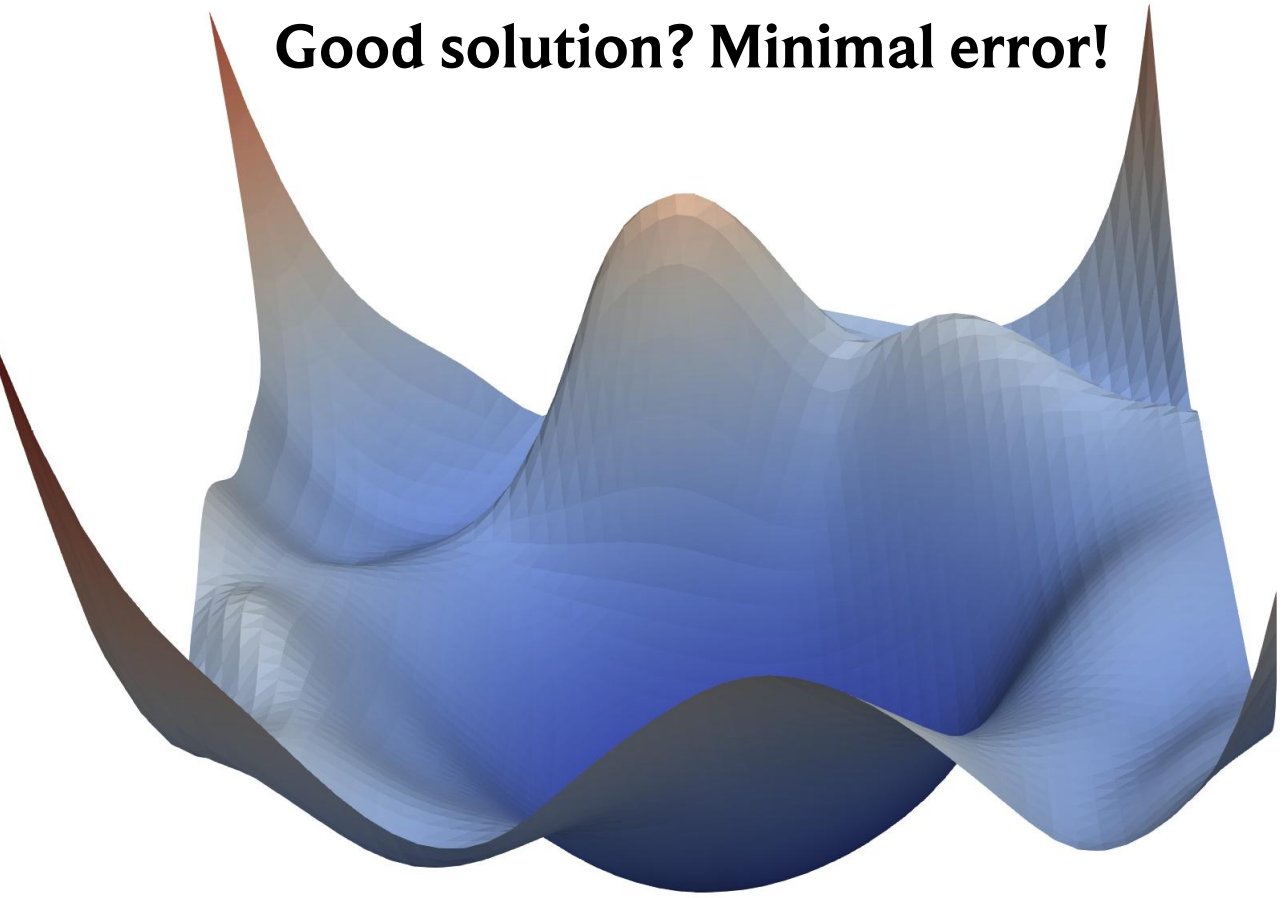
This applies to **ALL** equators…



Images: https://people.eecs.berkeley.edu/~jrs/highd/, September 2025

The blessing of dimensionality

Optimizing parameters of a neural network
= following gradients on an error landscape



Concern in low dimensions:
Getting stuck in local (non-optimal) minima.

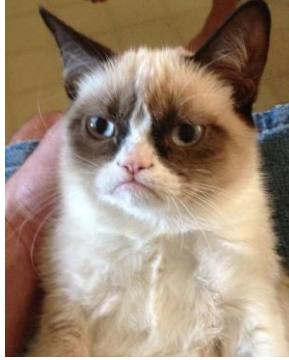
In high dimensions (case of neural nets)**:**
Almost all extrema are saddle points!

Reason:
Minimum = all N Eigenvalues of Hessian > 0,
almost impossible for high dimensions N

Intuition:
Looking in one direction, let p be the prob.
that we encounter a "wall".
Prob. to have walls in all N directions: $p^N \to 0$

Image: https://www.cs.umd.edu/~tomg/projects/landscapes/, August 2025

The curse of dimensionality (actually called like this!)



Many operations do not scale well with high dimensions!

For example: sampling a space with evenly-spaced points.

For the unit-line, if the points have distance 0.01, we need 100 points.
For the unit-square, we need $100^2 = 10^4$ points.
For the 5-dimensional hypercube, we need $100^5 = 10^{10}$ points! **Exponential explosion!**

Another example: high-dimensional data, meaning we pad together many features.

When training models, one might think having more features (*higher data dim.*) is better. However, when increasing the number of features linearly, the density of our training data reduces exponentially! Thus, our data is not representative anymore and we risk overfitting.

The manifold hypothesis

Manifold hypothesis: many high-dimensional data that occurs in the real-world populates a low-dimensional manifold.

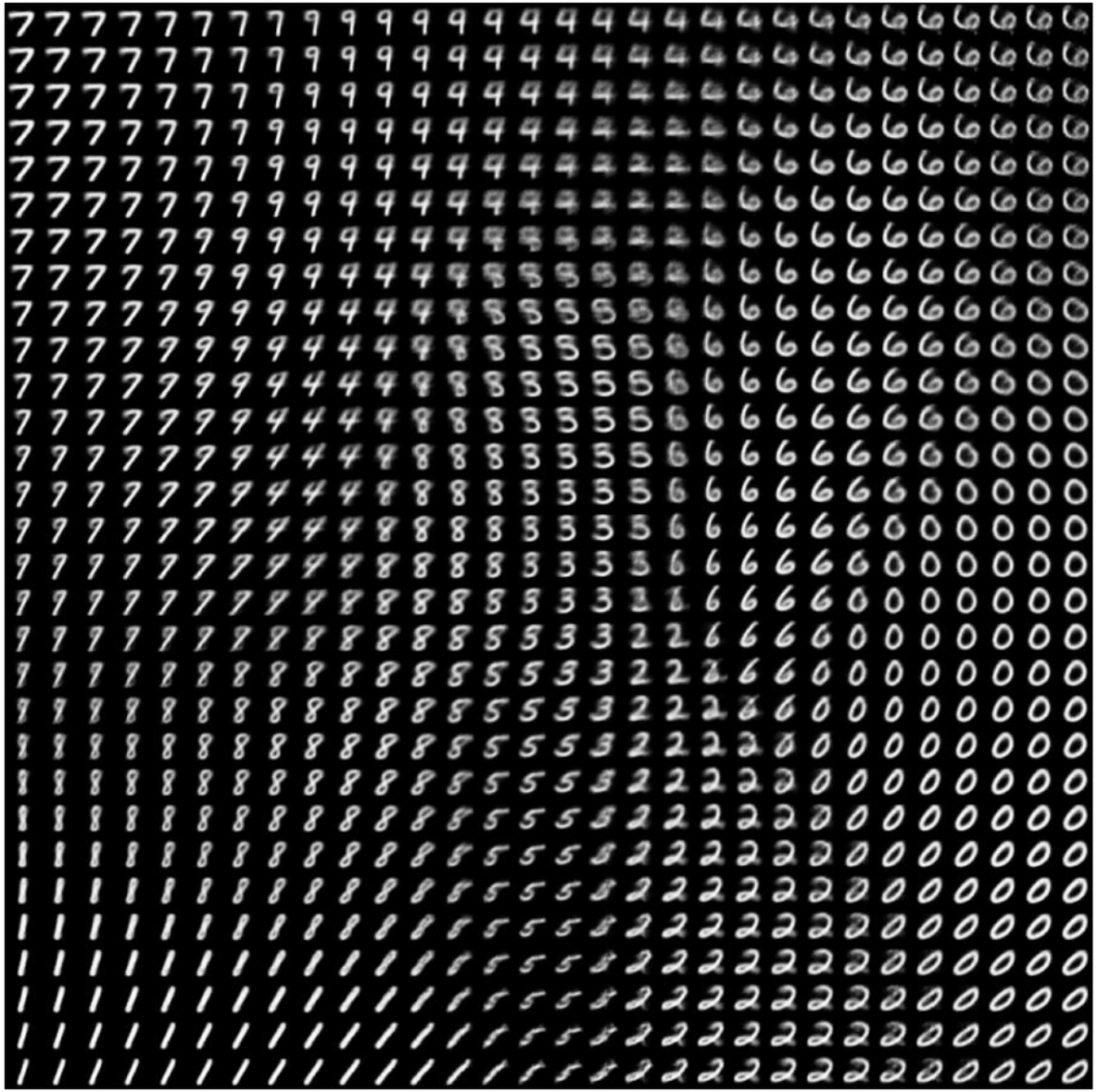
Roughly: in practice, "quality" high-dimensional data contains "(simple) structures" that we can use to answer questions about the data (*or not, then we have to collect better data*).

Or in other terms: the data can be "compressed" to lower dimensions with minimal loss.

Note: this is just a hypothesis. But without structure, we cannot do anything anyways!

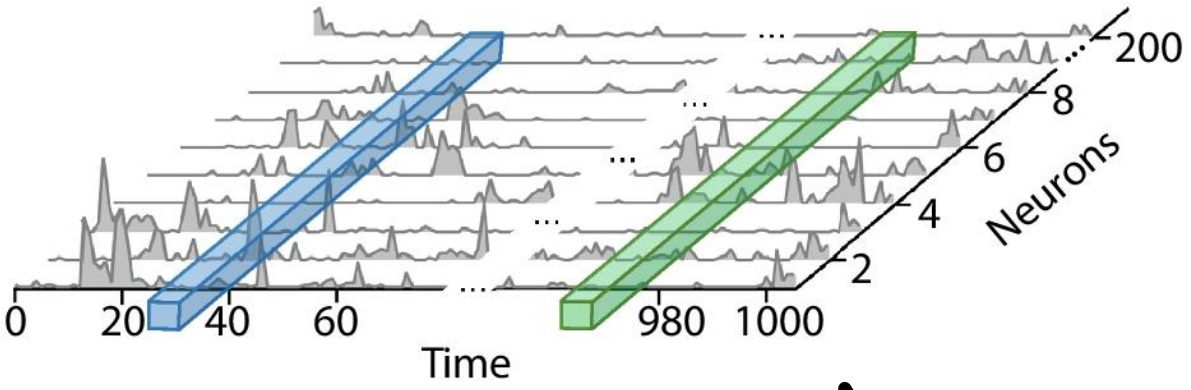
Examples

Example 1: MNIST handwritten digits dataset can be reduced to two dimensions while keeping a lot of structure (*image credit: François Chollet*).



Examples

Example 1: MNIST handwritten digits dataset can be reduced to two dimensions while keeping a lot of structure (*image credit: François Chollet*).



Example 2: neural data is incredibly high-dimensional (lots and lots of neurons), but neural activity itself often lies on a low-dimensional manifold!

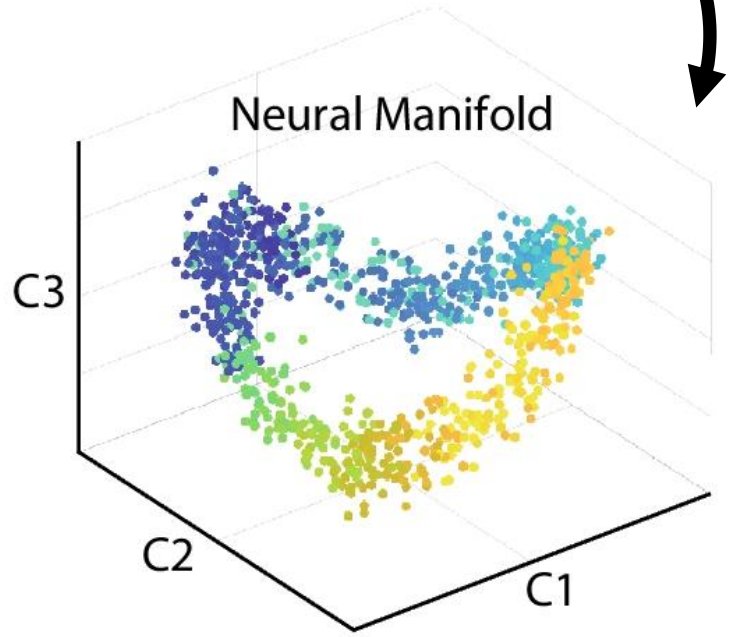


Image DOI: 10.1007/s10827-022-00839-3

Towards dimensionality reduction: a refresher

Eigenvalues and eigenvectors: Assume a square matrix $\boldsymbol{A} \in R^{n \times n}, n \in \mathbb{N}$. Then $\boldsymbol{A}$ has eigenvalue $\boldsymbol{\lambda}$ with eigenvector $\boldsymbol{v}$ if $\boldsymbol{Av} = \lambda \boldsymbol{v}$, i.e., $\boldsymbol{A}$ merely rescales $\boldsymbol{v}$ by λ.

Eigenvalue decomposition: Assume a square matrix $A \in R^{n \times n}$. Then its eigenvalue decomposition is given by $\boldsymbol{A} = \boldsymbol{V} \cdot \boldsymbol{D} \cdot \boldsymbol{V}^{-1}$ with $\boldsymbol{D} = \text{diag}(\lambda_0, \dots, \lambda_n)$ and $\boldsymbol{V} \in R^{n \times n}$. λ_i are the eigenvalues of $\boldsymbol{A}$, and the rows $\boldsymbol{V}_i$ are the eigenvectors with $\boldsymbol{A} \cdot \boldsymbol{V}_i = \lambda_i \boldsymbol{V}_i$.

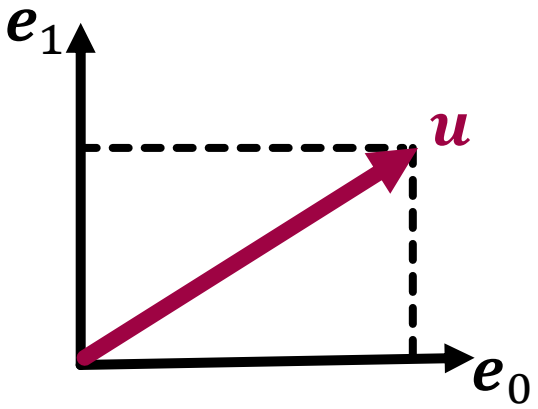
$\boldsymbol{V}$: basis change from eigenvectors to unit vectors $e_i = (0, \dots, 1, \dots, 0)$

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix}_V = a_0 \underbrace{\begin{pmatrix} V_{00} \\ V_{10} \end{pmatrix}}_{\boldsymbol{V}_0} + a_1 \underbrace{\begin{pmatrix} V_{01} \\ V_{11} \end{pmatrix}}_{\boldsymbol{V}_1} = \underbrace{\begin{pmatrix} V_{00} & V_{01} \\ V_{10} & V_{11} \end{pmatrix}}_{\boldsymbol{V}} \cdot \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = (a_0 V_{00} + a_1 V_{10}) \underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\boldsymbol{e}_0} + (a_0 V_{10} + a_1 V_{11}) \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{\boldsymbol{e}_1}$$

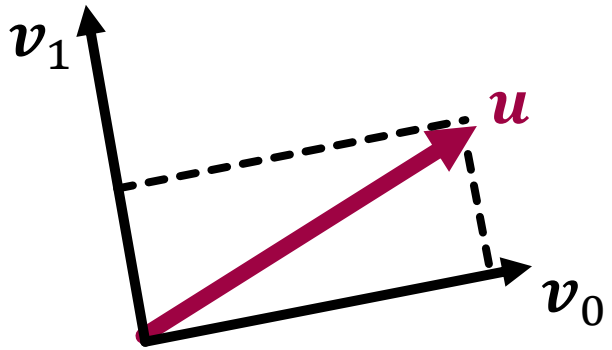
$\boldsymbol{V}^{-1}$: basis change from unit vectors to eigenvectors

Thus: $\boldsymbol{V} \cdot \boldsymbol{D} \cdot \boldsymbol{V}^{-1} \boldsymbol{u} =$ represent vector $\boldsymbol{u}$ in eigenbasis $\boldsymbol{u}' = \boldsymbol{V}^{-1} \boldsymbol{u}$, in eigenbasis $\boldsymbol{A}$ only stretches the eigenvectors $\boldsymbol{u}'' = \boldsymbol{D}\boldsymbol{u}'$, then transform back to the original basis $\boldsymbol{Au} = \boldsymbol{V}\boldsymbol{u}''$.

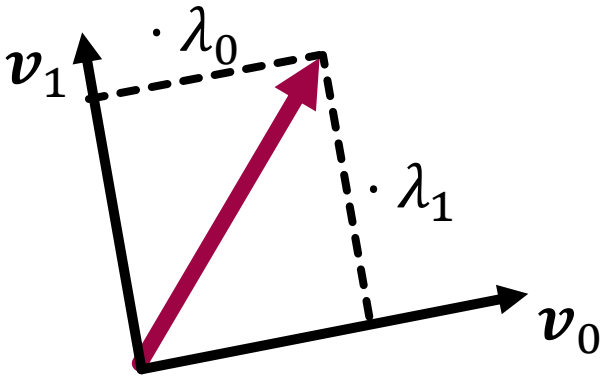
Visualisation of $A \cdot u = V \cdot D \cdot V^{-1} \cdot u$



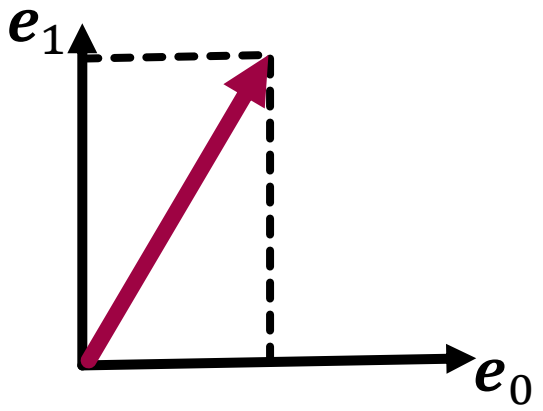
1. Vector in unit basis



2. Represent vector in Eigenbasis: $V^{-1}u$



3. In Eigenbasis, the matrix just stretches the components: $D \cdot V^{-1}u$



4. Represent new vector in unit basis: $V \cdot D \cdot V^{-1}u$

Exercise time

Show the following:

1. If $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric, then its eigenvalues are real and its eigenvectors are orthogonal.
2. If $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric, then $\boldsymbol{V}^{-1} = \boldsymbol{V}^{T}$.
3. $\max_{||\boldsymbol{u}||_2 = 1} \left(\boldsymbol{u}^{\boldsymbol{T}} \boldsymbol{A} \boldsymbol{u}\right) = \max(\lambda_i)$
4. $\min_{||\boldsymbol{u}||_2 = 1} \left(\boldsymbol{u}^{\boldsymbol{T}} \boldsymbol{A} \boldsymbol{u}\right) = \min(\lambda_i)$

Useful hints:

1. Symmetric matrix: $\boldsymbol{A}^{\boldsymbol{T}} = \boldsymbol{A}$ (or $A_{ij} = A_{ji}$)
2. Complex numbers: $c = c_0 + i\, c_1$, with $\mathrm{Re}(c) = c_0, \mathrm{Im}(c) = c_1$.
3. Complex conjugation: $\bar{c} = c_0 - i c_1$. Hence: $c + \bar{c} = 2\mathrm{Re}(c)$ and $c - \bar{c} = 2\mathrm{Im}(c)$.
4. A number λ is real-valued if $\lambda = \bar{\lambda}$.
5. Orthogonal basis: $v_i^T v_j = 0$ if $i \neq j$.

Useful case: Assume a **real-valued symmetric** matrix $\boldsymbol{A} \in R^{n \times n}$. Then $\boldsymbol{A}$ has an eigenvalue decomposition with real-valued eigenvalues: $\boldsymbol{A} = \boldsymbol{V} \cdot \boldsymbol{D} \cdot \boldsymbol{V}^{T}$ and $\boldsymbol{V}^{T}\boldsymbol{V} = \boldsymbol{1}$.

To get Eigenvalues, we need… the determinant

The determinant is always a bit hard to digest, but it has a surprisingly simple origin!

Let's try to solve the following system of linear equations: $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$

Rewrite: $\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$

This is solved by:
(see written notes)

$$x_0 = \frac{b_0 A_{11} - b_1 A_{01}}{A_{11} A_{00} - A_{01} A_{10}}$$

$$x_1 = \frac{b_1 A_{00} - b_0 A_{10}}{A_{11} A_{00} - A_{01} A_{10}}$$

We define $A_{11}A_{00} - A_{01}A_{10}$ as the determinant of $\boldsymbol{A}$, $\det(\boldsymbol{A})$, for $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$

Interpretation? If $\det(\boldsymbol{A}) \neq \boldsymbol{0}$, then

- $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$ has a unique solution for $\boldsymbol{x}$
- $\boldsymbol{A}$ has full rank
- Columns/rows of $\boldsymbol{A}$ are linearly independent
- Columns/rows form a basis of $\mathbb{R}^2$

The determinant in higher dimensions

The same "derivation" is true for higher dimensions as well!

For $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, the determinant is given by (*Leibniz formula*): $\det(\boldsymbol{A}) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \prod_{i=0}^{n-1} A_{i\sigma_i}$

S_n: permutations of n elements, e.g., $S_n = ((0,1),(1,0))$

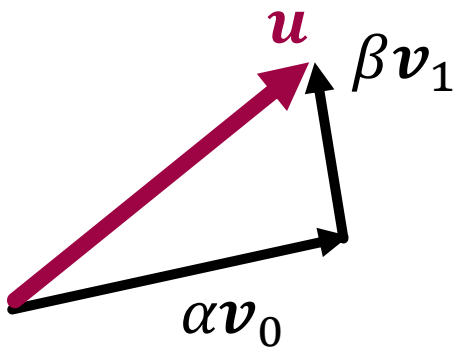
$\operatorname{sign}(\sigma)$: 1 for even number of permutations, -1 otherwise, e.g., $\operatorname{sign}\big((1,0)\big) = -1$

Exercise:

1. Using the Leibniz formula, calculate the determinant for $n = 2$ and $n = 3$.
2. Using the property $\det(\boldsymbol{A} \cdot \boldsymbol{B}) = \det(\boldsymbol{A}) \cdot \det(\boldsymbol{B})$, show that the determinant of a matrix is the product of its eigenvalues.

Determinant = change in volume through linear map!

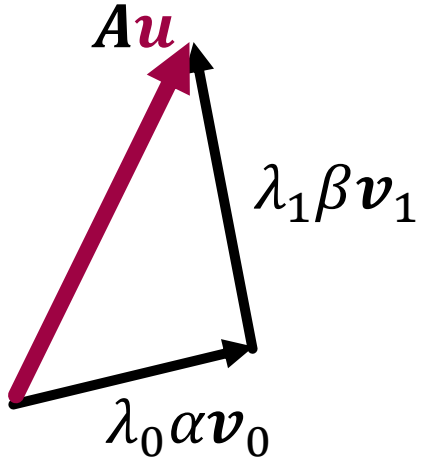
This is especially nice to see using the eigenbasis of matrix A



u represented in orthonormal eigenbasis,
i.e., $u = \alpha v_0 + \beta v_1$

Surface area of triangle:

$$\propto |\alpha||\beta| \cdot |v_0|\,|v_1|$$



Au = rescaling in direction of eigenvectors

Surface area of triangle:

$$\propto |\lambda_0||\lambda_1|\,|\alpha|\,|\beta| \cdot |v_0|\,|v_1|$$

Changed by factor $|\lambda_0||\lambda_1|$!

Last step: calculating eigenvalues and eigenvectors

Reminder: We want $\boldsymbol{Av} = \lambda \boldsymbol{v}$, or in other terms: $(\boldsymbol{A} - \boldsymbol{1}\lambda)\boldsymbol{v} = 0$

This is a system of linear equations!
Obviously, this is solved by $\boldsymbol{v} = 0$ if $\det(\boldsymbol{A} - \boldsymbol{1}\lambda) \neq 0$

Find non-trivial solution: **require** $\det(\boldsymbol{A} - \boldsymbol{1}\lambda) = 0$

Thus, there are two steps:

1. Get eigenvalues by solving $\det(\boldsymbol{A} - \boldsymbol{1}\lambda) = 0$
2. Use eigenvalue λ_i to solve for $\boldsymbol{v}_i \neq \boldsymbol{0}, (\boldsymbol{A} - \boldsymbol{1}\lambda_i)\boldsymbol{v}_i = 0$

Exercises: Calculate the eigenvalues and eigenvectors of the following matrices

1. $\begin{pmatrix} 3 & 0 \\ 8 & -1 \end{pmatrix}$

2. $\begin{pmatrix} 0 & 3 \\ 4 & 0 \end{pmatrix}$

3. $\begin{pmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

Dimensionality reduction

Setting: Assume we are faced with high-dimensional data. A common approach is to reduce the number of dimensions to d, i.e., to find a representation that uses only a small number of meaningful features. This is often done as a preprocessing step before using other methods such as clustering, which work better in low dimensions.

Principal Component Analysis (PCA) is the *goat (greatest of all time)* of dimensionality reduction. It is principled, simple, and especially useful for visualizing data in 2D and 3D.

The goal of PCA is to **project** the data into a d −dimensional subspace **such that directions with highest feature variance are kept**.

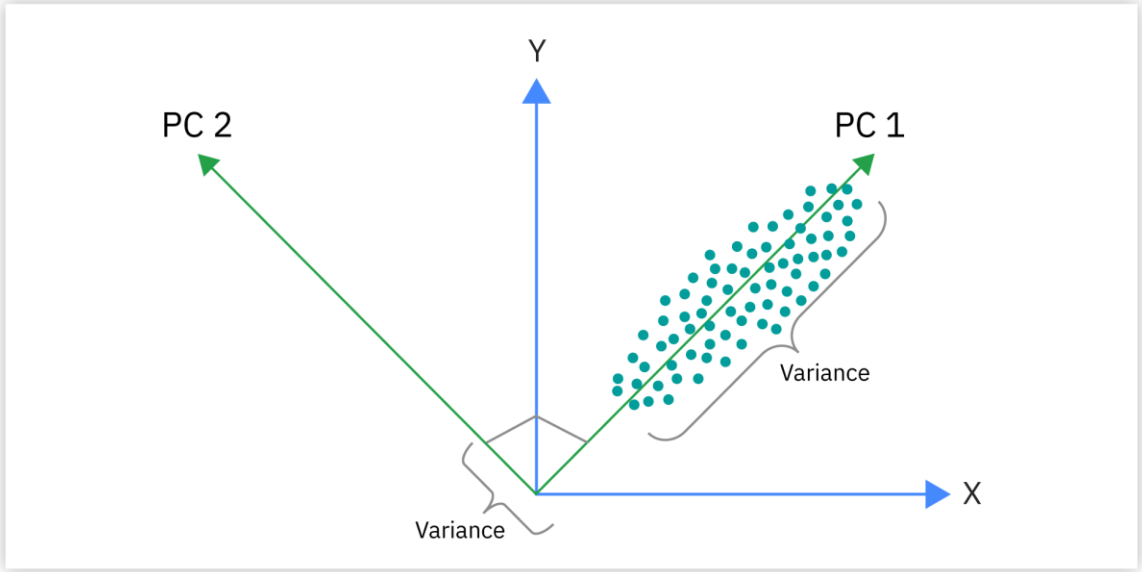
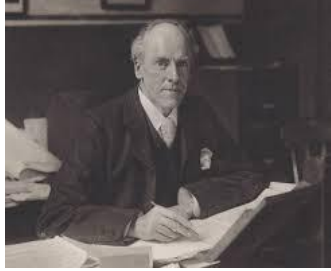


Image: https://www.ibm.com/think/topics/principal-component-analysis, September 2025

Principal component analysis (PCA)



Karl Pearson (1857 - 1936)

PCA: Given data samples $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_N \in \mathbb{R}^n$, the columns of the projection $\boldsymbol{V} \in \mathbb{R}^{n \times d}$ are given by the d eigenvectors of the covariance matrix

$$\boldsymbol{\Sigma} = \frac{1}{n-1} \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T, \qquad \boldsymbol{\mu} = \frac{1}{n} \sum_{i}^{n} \boldsymbol{x}_i$$

with the highest eigenvalues. The projected data is given by $\boldsymbol{y}_i = \boldsymbol{V}^T \boldsymbol{x}_i \in \mathbb{R}^d$ $(d < n)$.

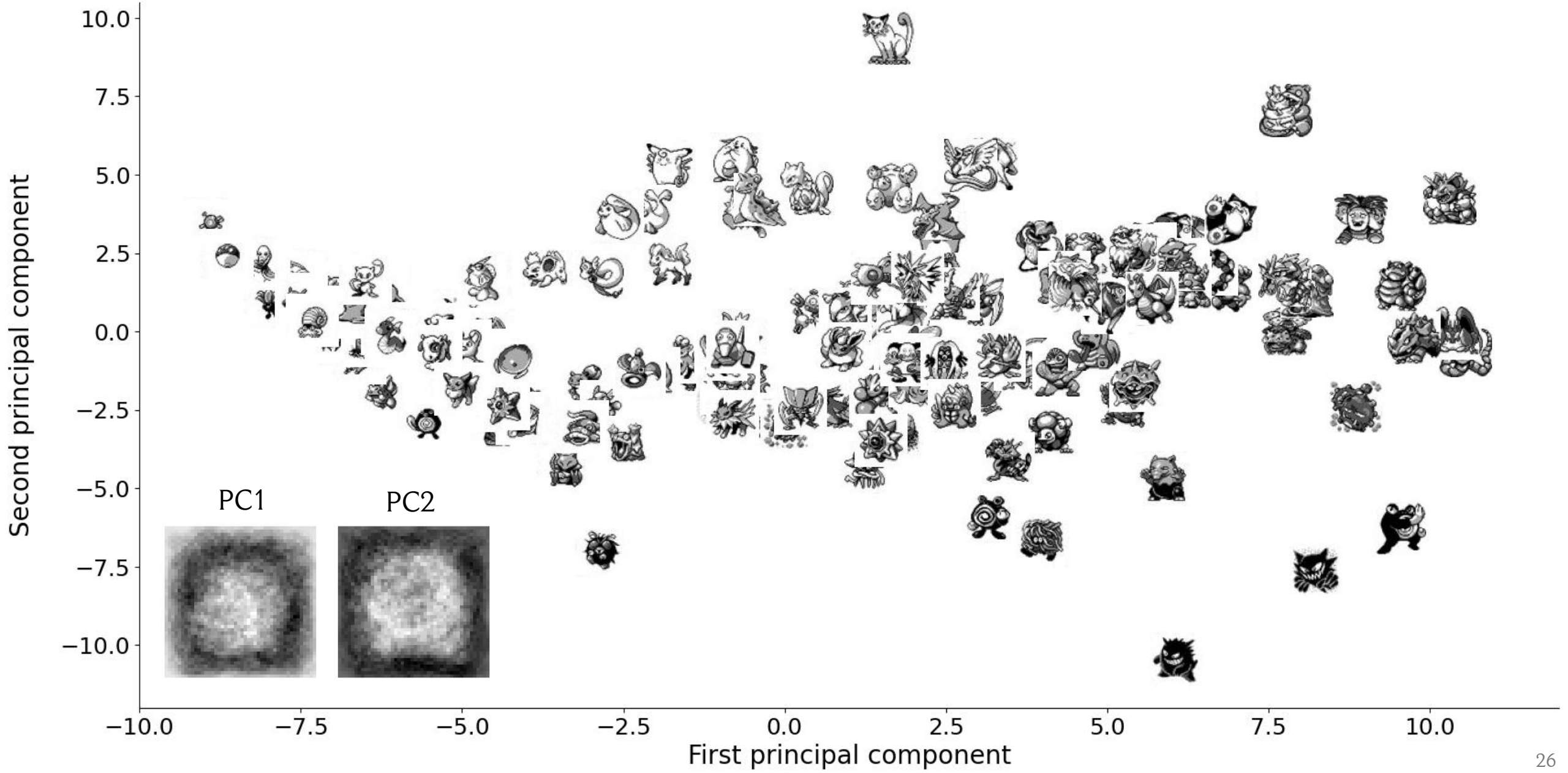
The objective that PCA optimizes can be formulated in two different ways:

1. As we just said: finding a projection that conserves as much variance in the data as possible.
2. Finding a d-dimensional affine subspace for which the projected points best resemble the original ones, $x_i \approx \mu + \boldsymbol{V} y_i$, with constant $\mu \in \mathbb{R}^{\boldsymbol{n}}$.

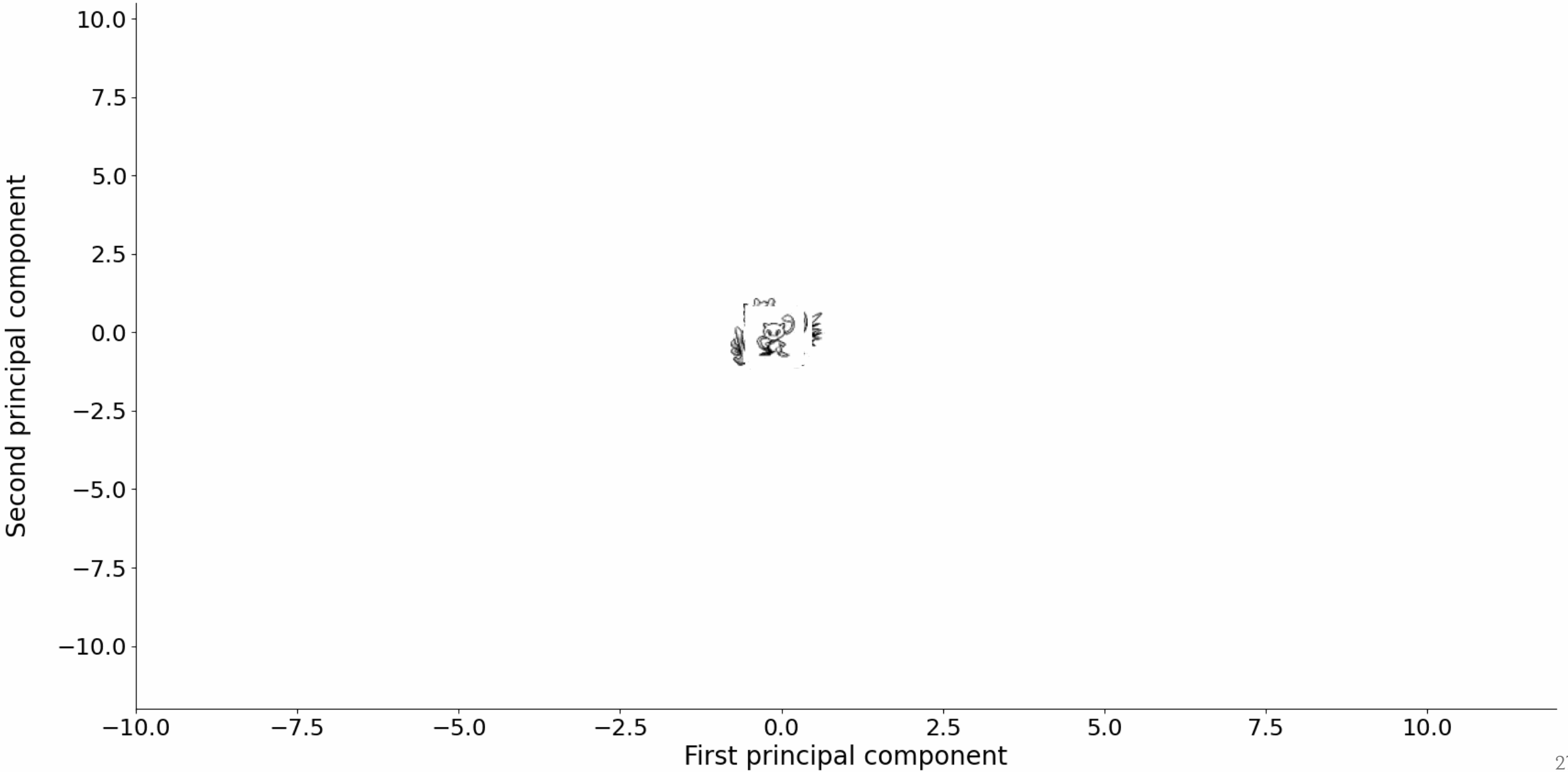
The solutions are identical up to a constant offset $\boldsymbol{V}^T \mu$.

Proof: see handwritten notes

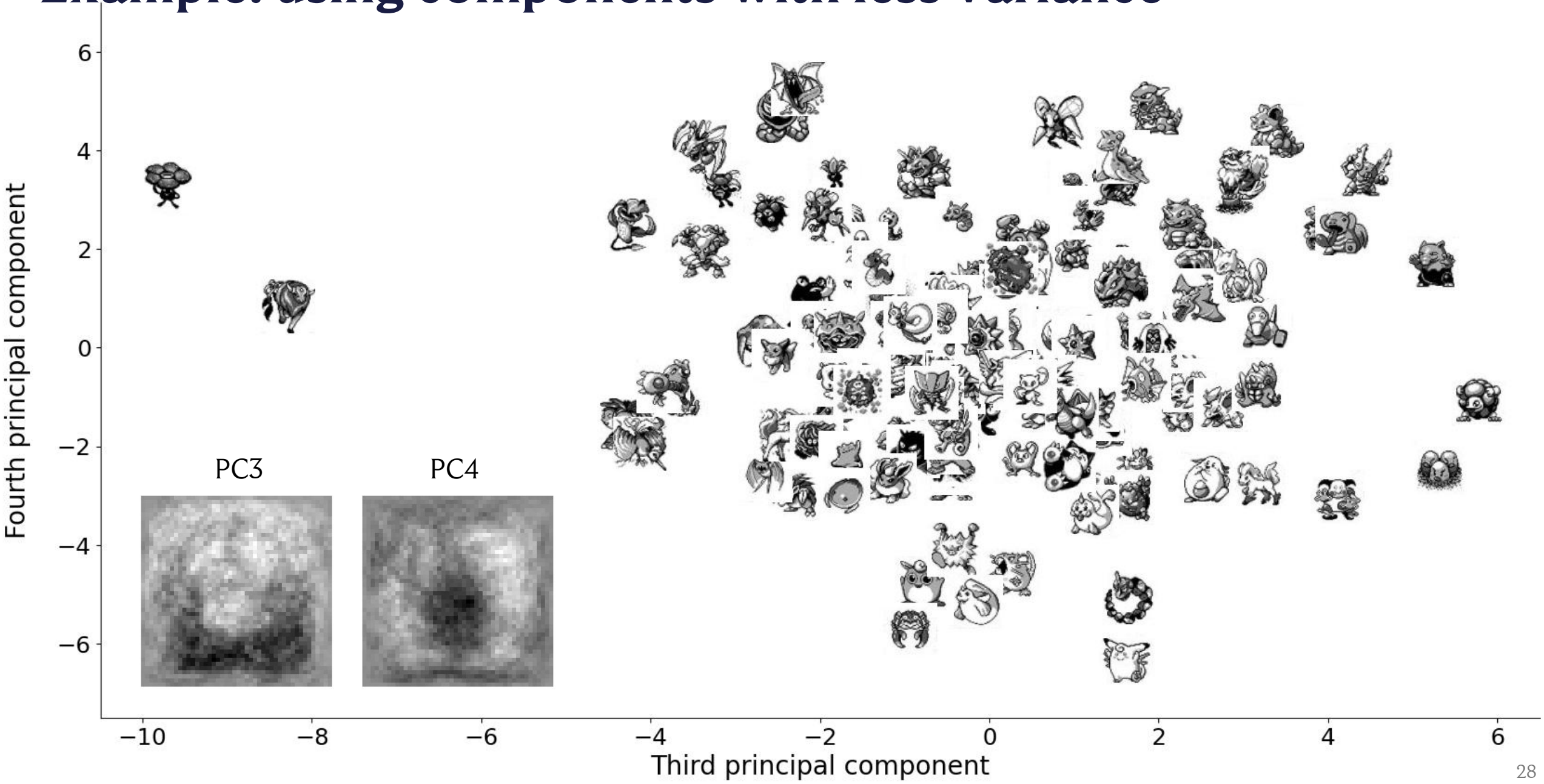
Example: using PCA on the original Pokemon



Example: randomly perturb y_i until $x_i \approx \mu + Vy_i$

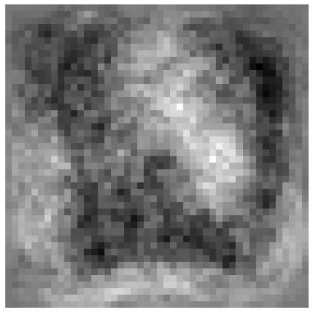


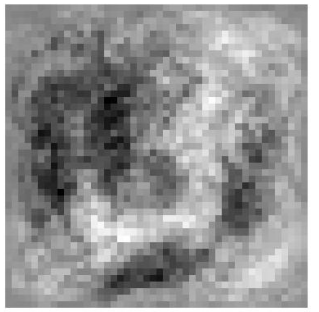
Example: using components with less variance



Example: using components with less variance

Note: *The first and second eigenvalue are by far the largest!*





Sixth principal component

8
6
4
2
0
−2
−4
−6

−6 −4 −2 0 2 4 6 8

Fifth principal component

Which target dimension to pick for PCA?

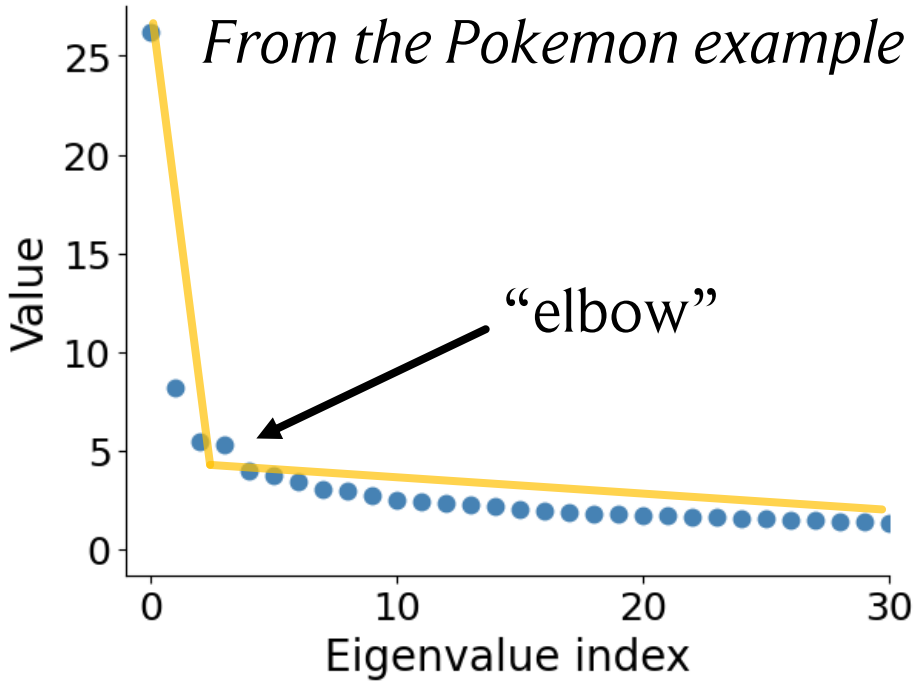
For visualization, we obviously pick $d = 2$ or $d = 3$.

But PCA is also useful in other applications!

- **Denoising**
- **Downstream analysis:** use lower-dim. representation as input for another method

Any ideas how to choose the dimension?

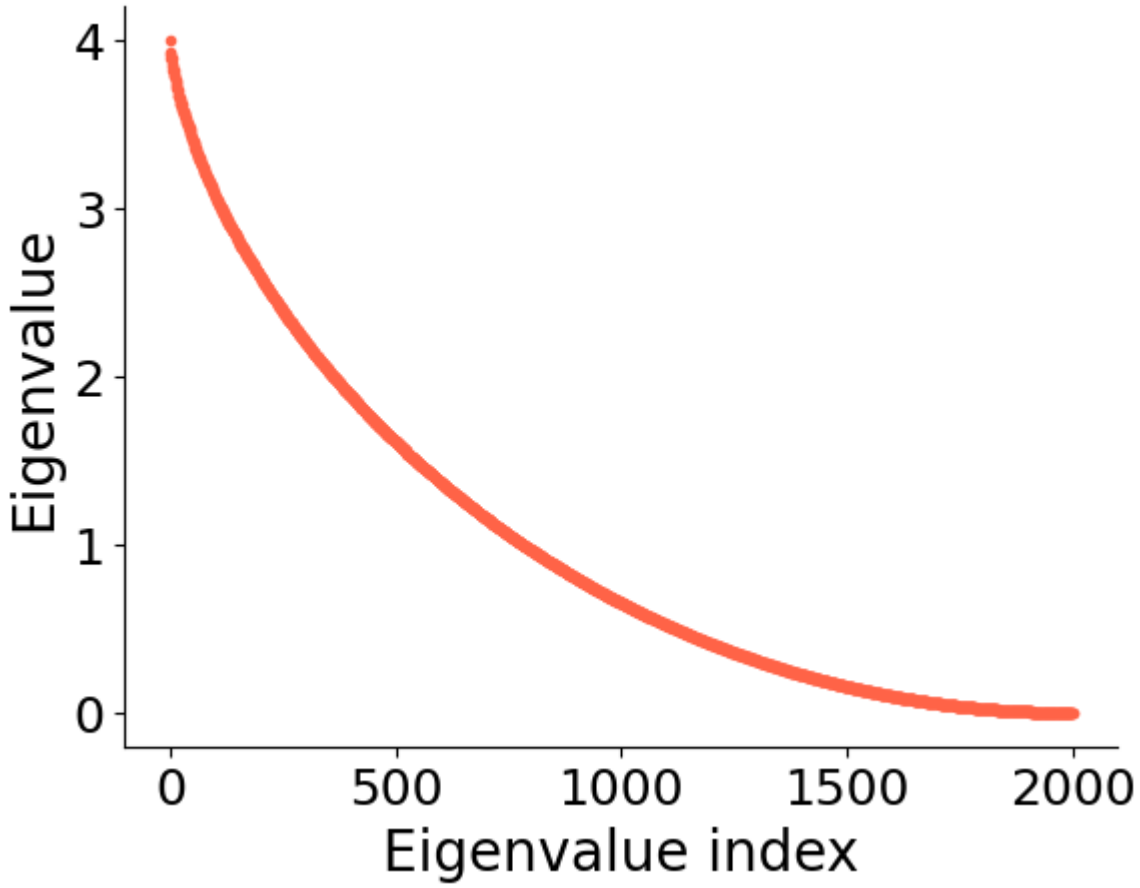
Popular heuristic: look at eigenvalues, is there an "elbow" point?



Question!

I have 2000 data samples, with each sample having a dimension of 2000.

Now I calculate the covariance matrix and look at its eigenvalues.



Does this data have a specific low-dimensional structure?

Spectrum of covariance matrices for finite sample sizes

Assume our data $\boldsymbol{x}$ is generated from a d-dimensional Gaussian distribution:

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

$\boldsymbol{\mu}$: mean
$\boldsymbol{\Sigma}$: covariance matrix

If we have n samples $\boldsymbol{x}_n$, we can approximate the covariance matrix via:

$$\boldsymbol{\Sigma}_n = \frac{1}{n-1}\sum_{k=1}^{n}(\boldsymbol{x}_k - \boldsymbol{\mu})\,(\boldsymbol{x}_k - \boldsymbol{\mu})^T$$

And we know that for $n \to \infty$ and fixed d, $\Sigma_n \to \Sigma$ *(law of large numbers)*.

But what if d and p are similar?!

The Marčenko-Pastur distribution

Volodymir Marchenko (1922 -)
and Leonid Pastur (1937 -)

We can get an (surprising) insight for the simple case $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$ and $\boldsymbol{\mu} = \mathbf{0}$

Unit matrix with σ^2 on the diagonal.
Thus, dimensions are independent normal random variables!

Marčenko-Pastur distribution: Assume we have n samples $x_k \in \mathbb{R}^d$ sampled from the normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. If both d and n go to ∞ with fixed ratio $\gamma = \frac{d}{n} \leq 1$, then the eigenvalues of $\boldsymbol{\Sigma}_n$ have the following distribution:

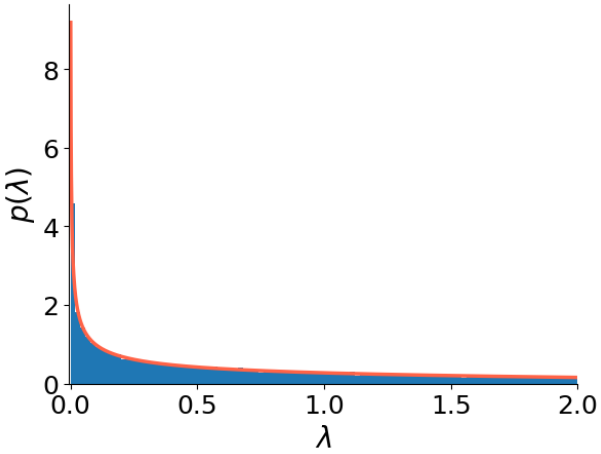
$$p(\lambda) = \frac{1}{2\pi\sigma^2}\frac{\sqrt{(\gamma_+ - \lambda)(\lambda - \gamma_-)}}{\gamma\lambda}1_{[\gamma_-,\gamma_+]}(\lambda) \quad \text{with}$$

$$\gamma_\pm = \sigma^2(1 \pm \sqrt{\gamma})^2$$

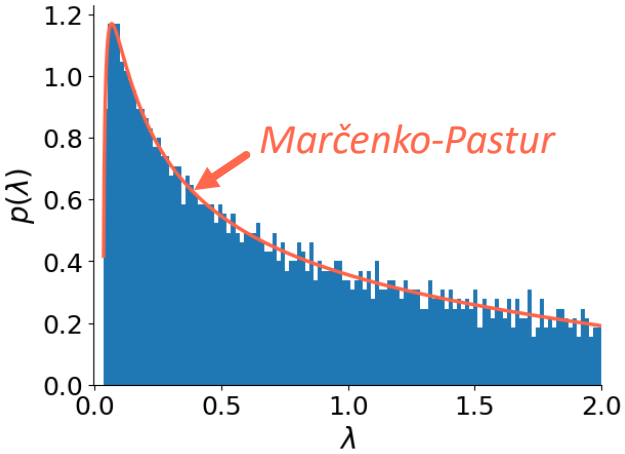
$$1_{[\gamma_-,\gamma_+]}(\lambda) = \begin{cases} 1 & \text{if } \gamma_- \leq \lambda \leq \gamma_+ \\ 0 & \text{otherwise} \end{cases}$$

High-dimensions hit again!

$d = 2000, n = 2000$

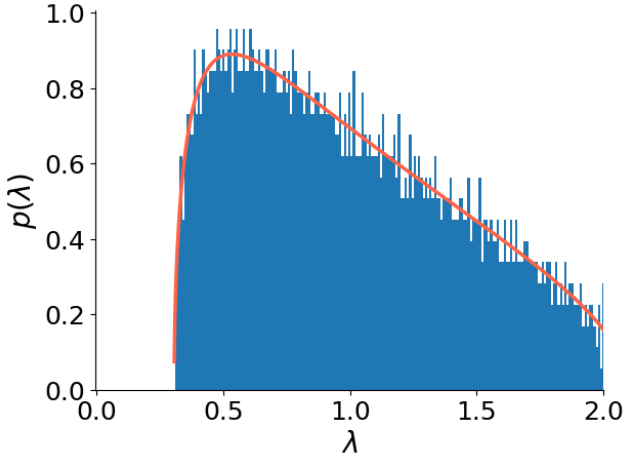


$d = 2000, n = 3000$

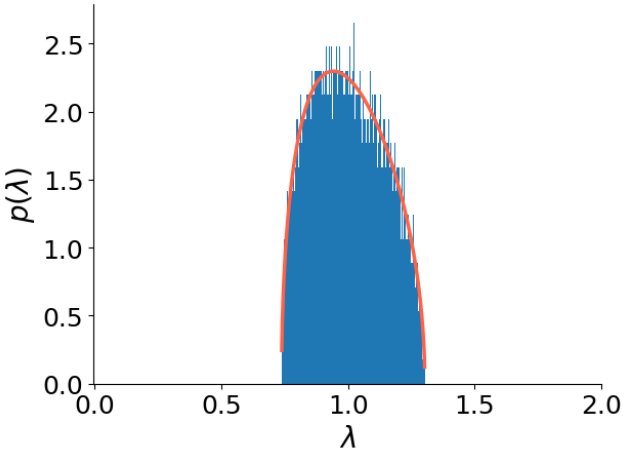


Even though the data was generated from a distribution without low-dimensional substructure, for low sample sizes it looks like it did!

$d = 2000, n = 10000$



$d = 2000, n = 100000$



Is there anything we can do though…?

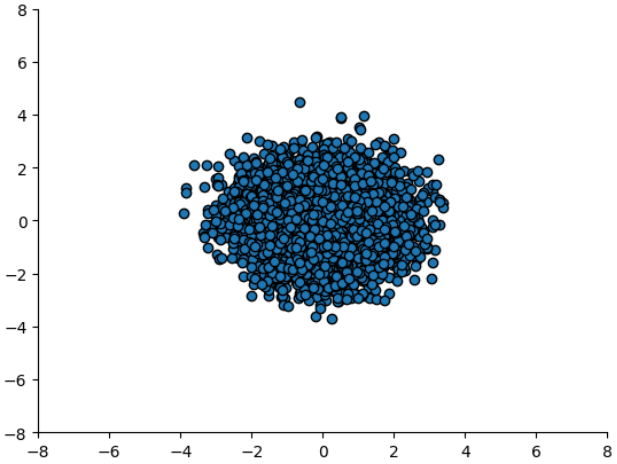
Under our current assumptions: **yes**!

Let's artificially add a principal component ourselves: $\boldsymbol{x_k} = \boldsymbol{g} + \sqrt{\beta}\, g_0 \boldsymbol{u}$

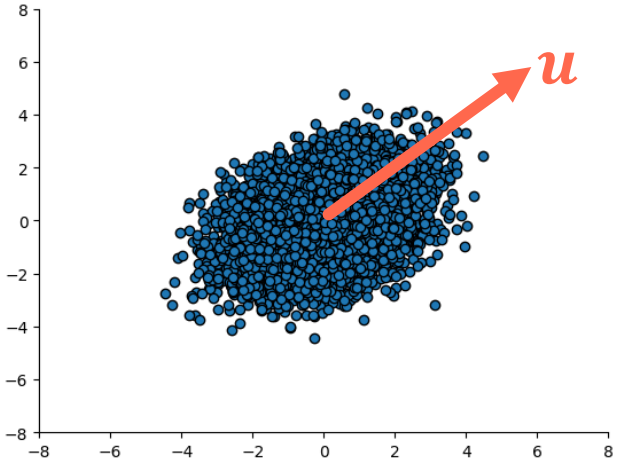
$\boldsymbol{g} \sim \mathcal{N}(\mathbf{0},\ \mathbf{I}), g_0 \sim \mathcal{N}(0,\ 1)$,
$\boldsymbol{u}$ is a normed vector (our signal)
β is the signal-to-noise ratio (SNR)

Thus, we actually have *(see handwritten notes)*: $\boldsymbol{x_k} \sim \mathcal{N}(\mathbf{0}, \mathbf{I} + \beta \boldsymbol{u}\boldsymbol{u}^T)$

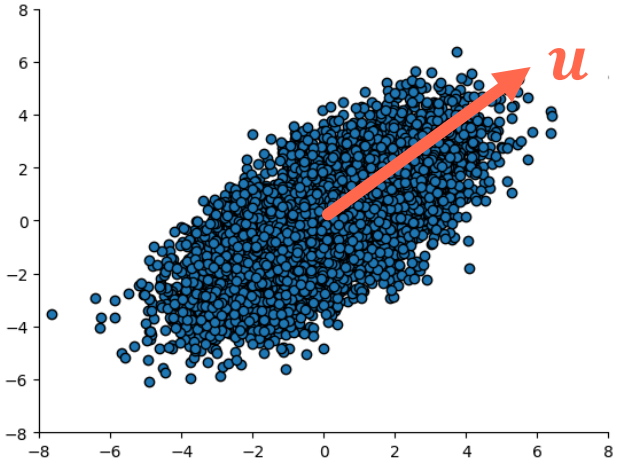
$\beta = 0$



$\beta = 1$



$\beta = 4$

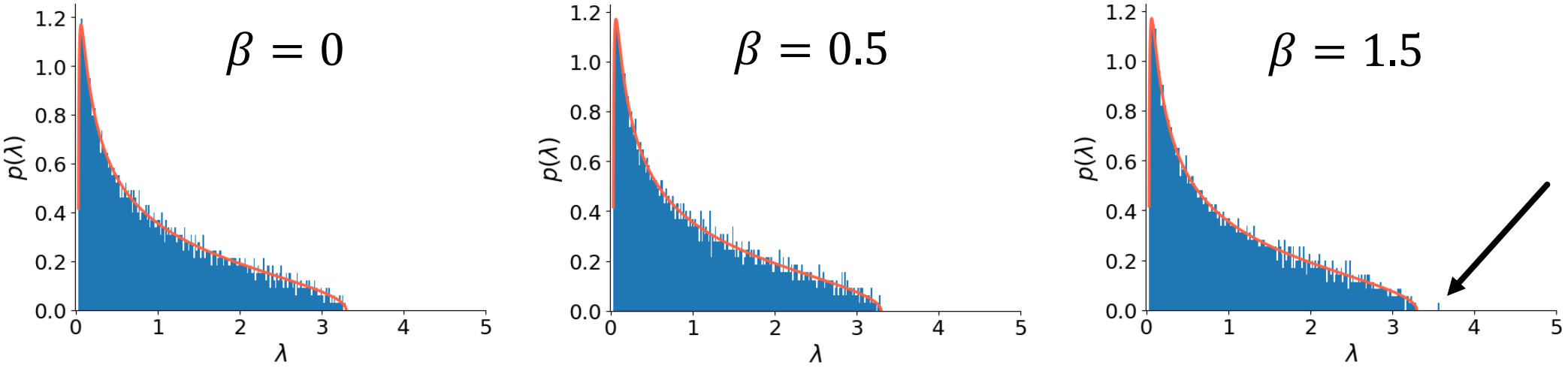


Is there anything we can do though…?

Curious: If $\beta \geq \sqrt{\gamma} = \sqrt{\frac{d}{n}}$, the added component appears as a separate eigenvalue where the Marčenko-Pastur distribution would be 0!

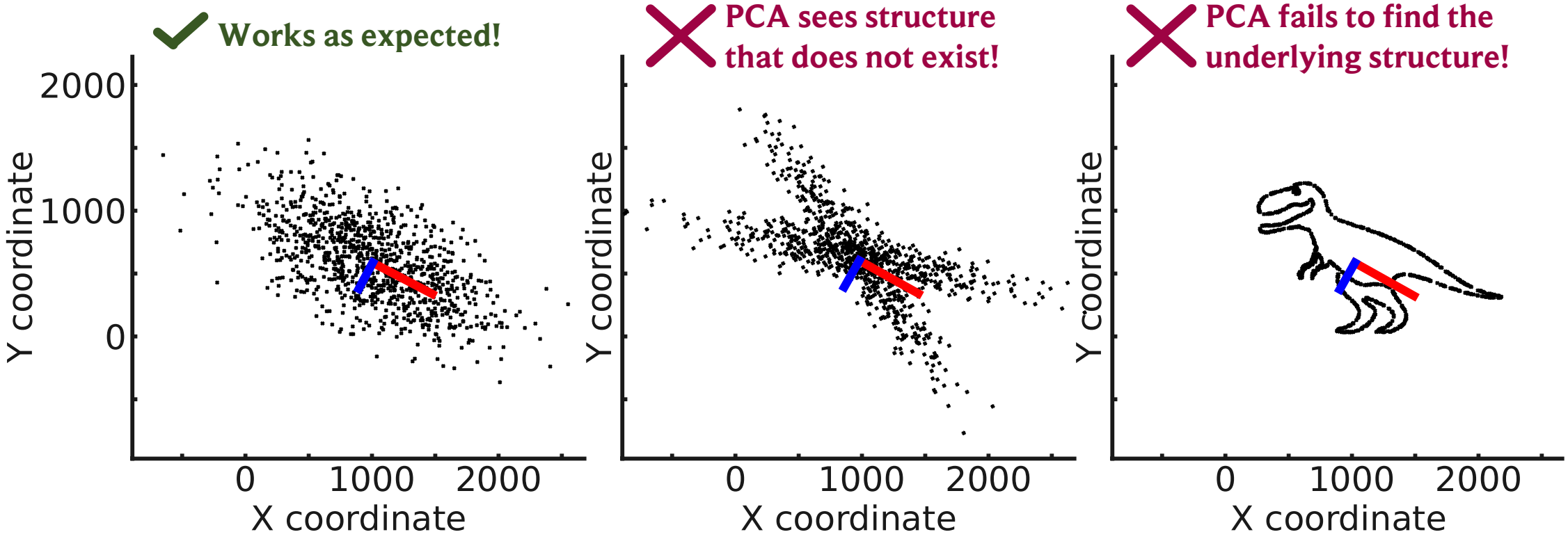
Note: basically, the signal has to be strong enough to surpass the highest eigenvalue caused by noise!

Time to try this in action: we use again $n = 3000, d = 2000$, thus $\sqrt{\gamma} \approx 0.8$



You can use the # of values on the right of the Marčenko-Pastur distribution as the # of principal components.
***BUT** do not forget that this analysis is only valid under certain conditions (Gaussian noise)!!*

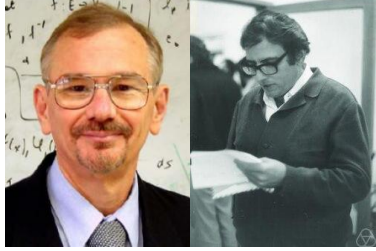
Beware: pitfalls of PCA…



Vastly different data, but the principal components are identical!

Eva Dyer & Konrad Kording. *Why the simplest explanation isn't always the best.* PNAS (2023).

A simple approach: Johnson-Lindenstrauss Lemma



William Johnson (1944 -) and
Joram Lindenstrauss (1936 - 2012)

There is a surprisingly simple way of mapping data to lower dimensions:

Let $x_1, x_2, \ldots, x_m \in \mathbb{R}^n$ be arbitrary. Then for $d = \frac{4\ln(m)}{\epsilon^2}$, there exists a linear map $y_i = L(x_i)$ such that for $0 < \epsilon < 1$ and with probability at least $1 - \frac{2}{m}$:

$$(1-\epsilon)\|x_j\|_2 \leq \|y_j\|_2 \leq (1+\epsilon)\|x_j\|_2 \quad \forall j$$

$$(1-\epsilon)\|x_j - x_i\|_2 \leq \|y_j - y_i\|_2 \leq (1+\epsilon)\|x_j - x_i\|_2 \quad \forall j, i$$

Proof: see handwritten notes

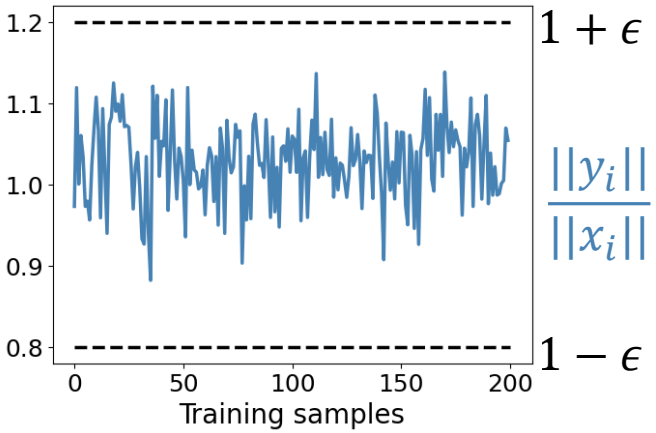
We can prove this by construction for a very simple map: $L(x_i) = \frac{1}{\sqrt{d}} G x_i$, where $G \in \mathbb{R}^{d \times n}$ is a matrix with entries G_{ij} sampled from a normal distribution with $\mu = 0$ and $\sigma^2 = 1$

Thus, for random maps, we have guarantees for preserving lengths and distances!

Example: using random projections to reduce dimensions

We use $m = 200$ random samples from the MNIST handwritten digits dataset ($n = 28 \cdot 28$).

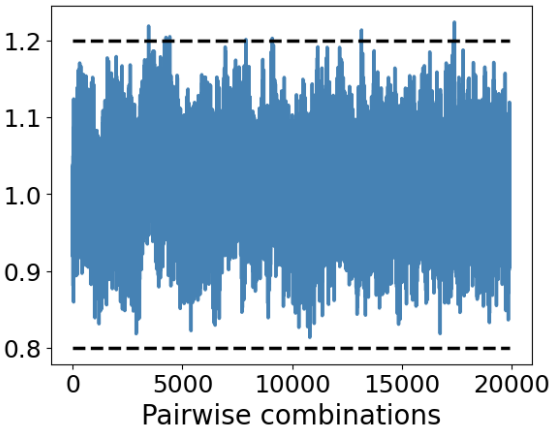
For $\epsilon = 0.2$, we get $d = 132$:



$1 + \epsilon$

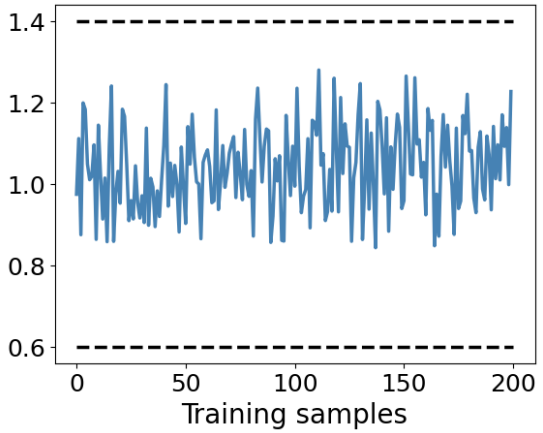
$\frac{||y_i||}{||x_i||}$

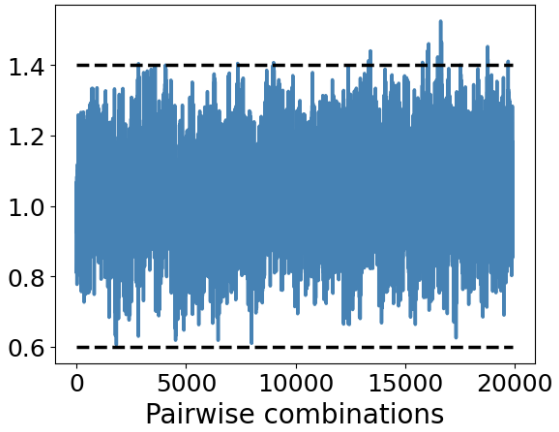
$1 - \epsilon$



$\frac{||y_i - y_j||}{||x_i - x_j||}$

For $\epsilon = 0.4$, we get $d = 33$:

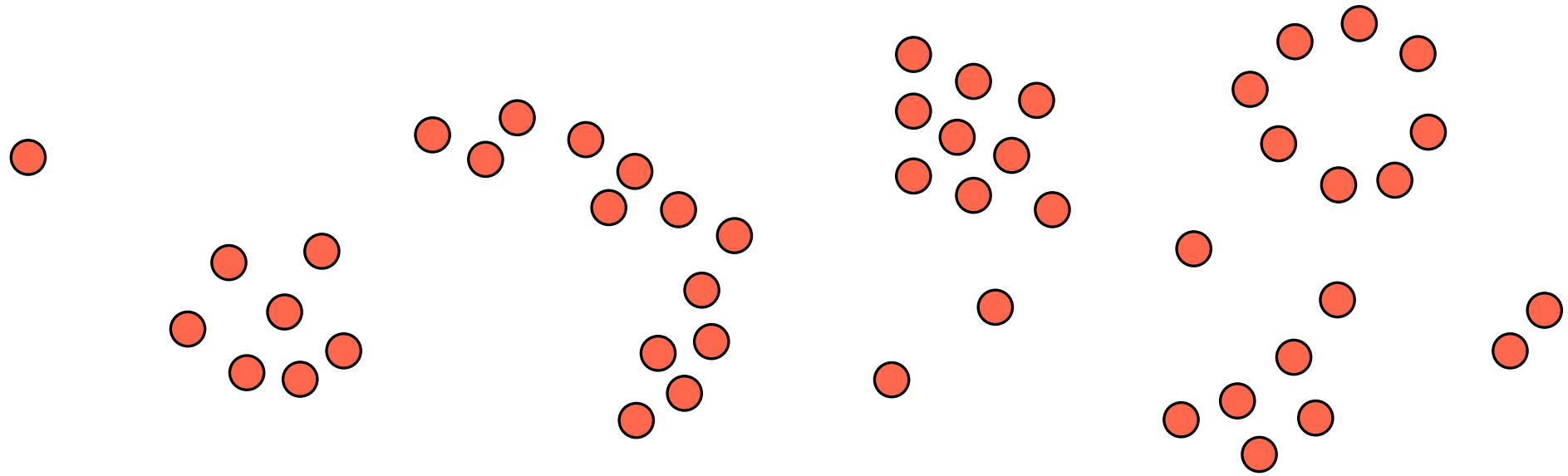




Note: good for very high dimensions, as too low values of d yield large values of ϵ!

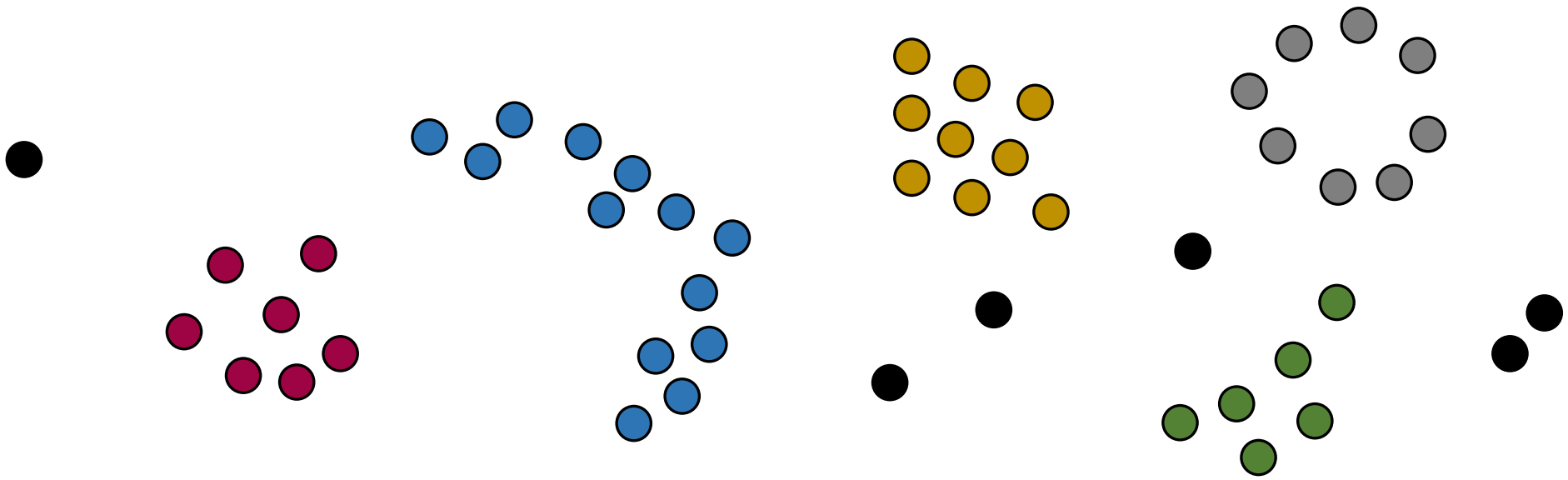
Making sense of unlabeled data: clustering

Assume we have high-dimensional data. One step to deal with it is, as discussed, to reduce its dimension. **But what then?**



Making sense of unlabeled data: clustering

Assume we have high-dimensional data. One step to deal with it is, as discussed, to reduce its dimension. **But what then?**



→ We can group data points that are “similar” to each other: **clustering**

One of the most basic methods: k-means

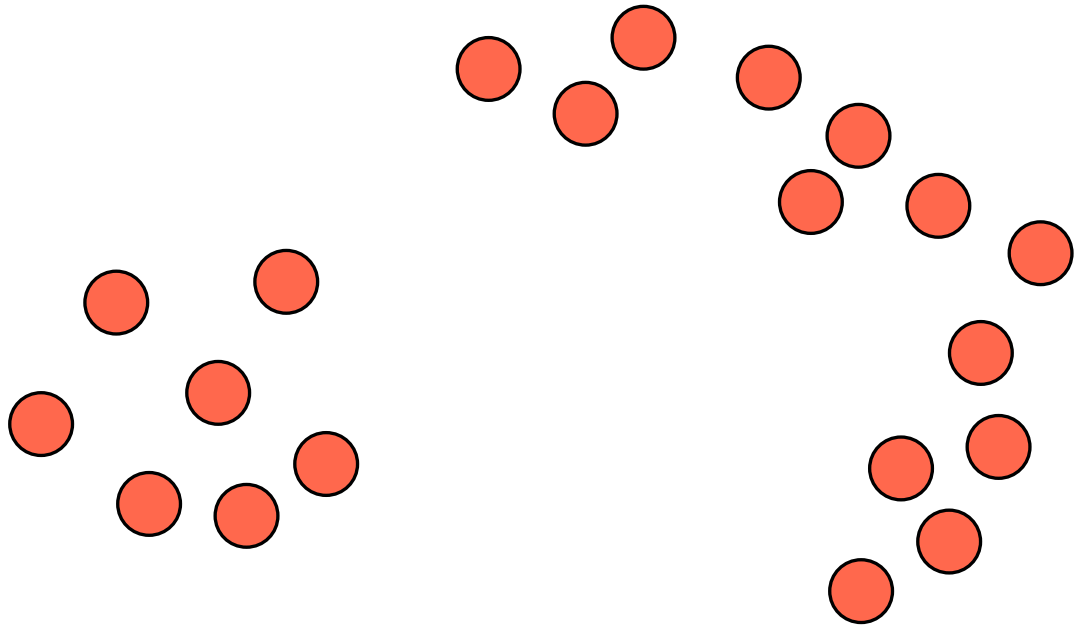
***k*-means:** Assume we have n data samples x_i, $i \in [1, n]$. Choose the number of clusters k which we want to find, as well as three initial centroids of the clusters c_j, $c_j \in [1,2,3]$. l_i is the cluster label that we assigned data sample i, and $A_j = \{i: l_i = j\}$.

Then iterate as follows:

1. Assign each data sample to the cluster with the closest centroid, i.e., $l_i = \min_j(d(x_i, c_j))$, where d is a distance metric like the Euclidean metric.
2. Recalculate the centroids: $c_j = \frac{1}{|A_j|}\sum_{k \in A_j} x_j$
3. Go back to 1. and repeat until the clusters are stable.

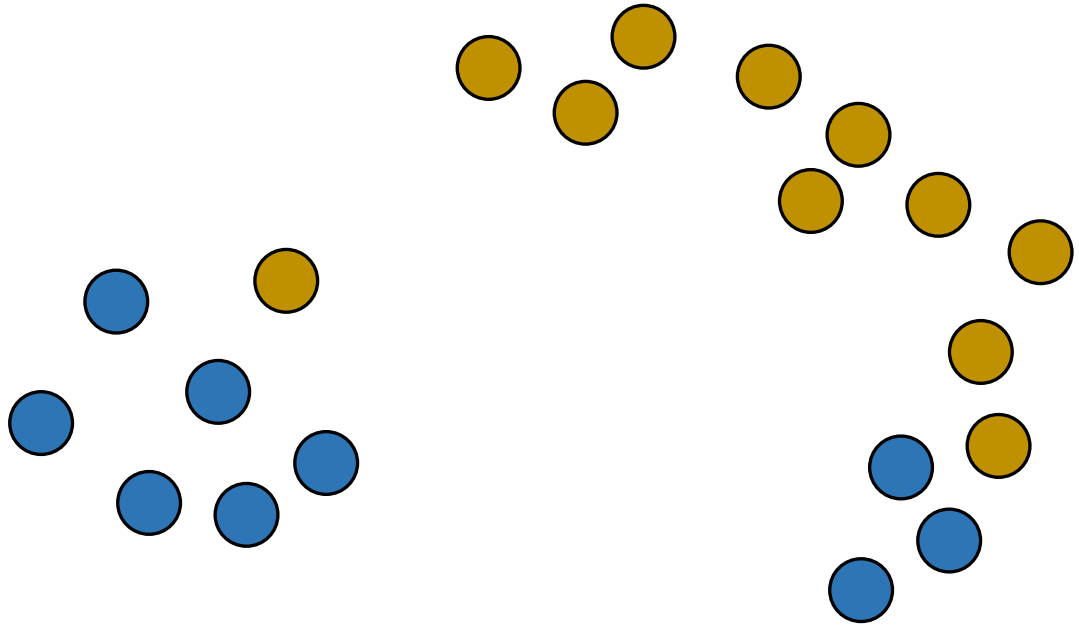
How do we initialize the centroids?
A common approach is to randomly assign labels in the beginning and calculate the centroids from these clusters.

k-means step by step



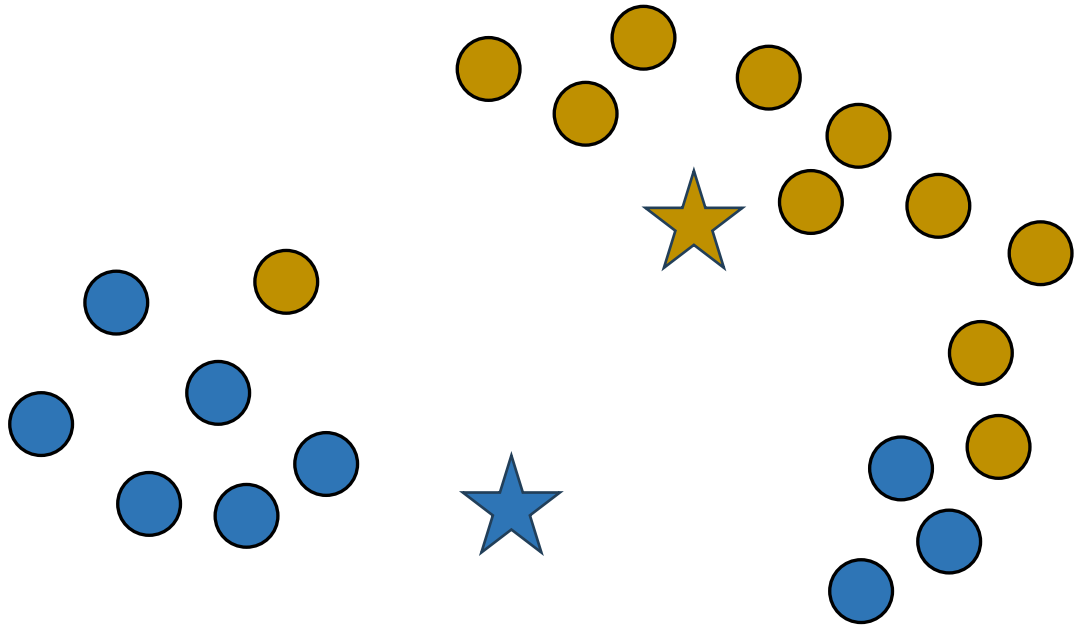
k-means step by step

Initial step: randomly assign classes!



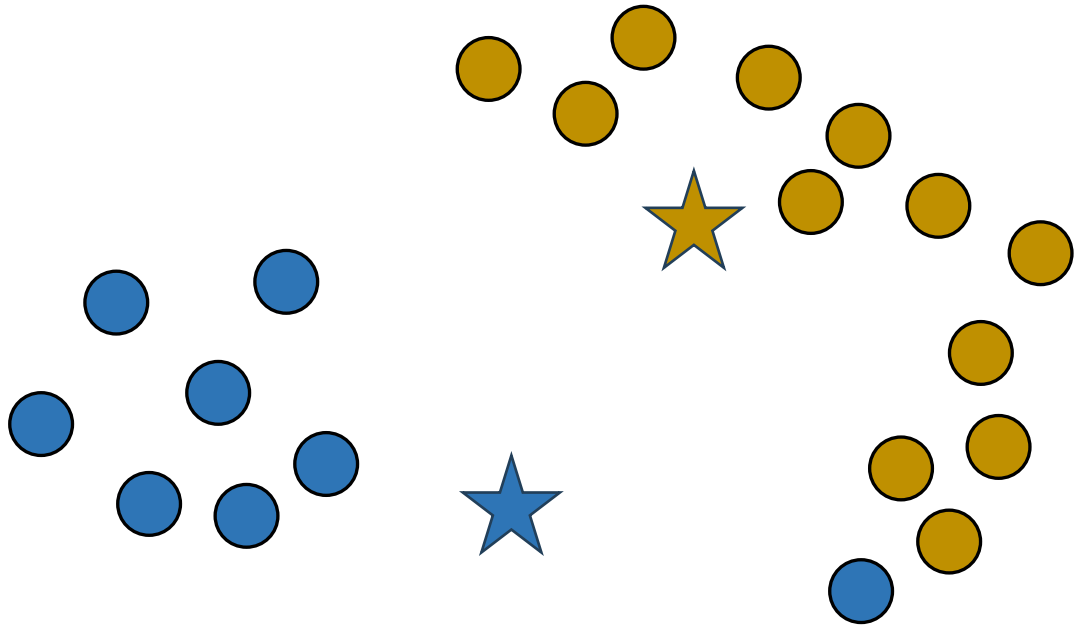
k-means step by step

Then: calculate centroids = “centre of mass” per class!



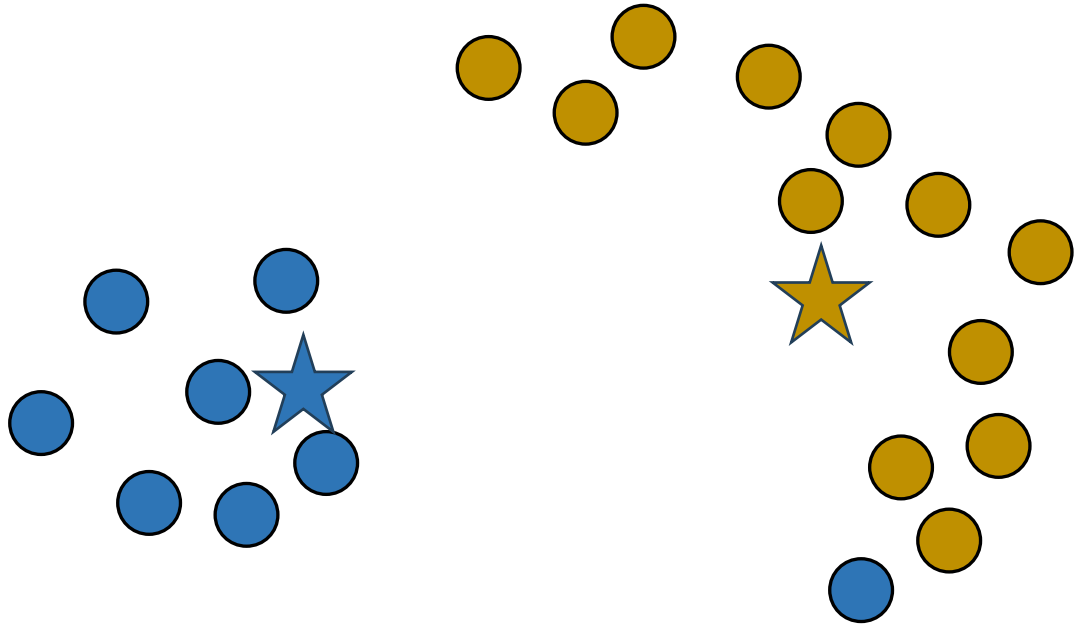
k-means step by step

Then: reassign classes — for each point, its class is given by the closer centroid!



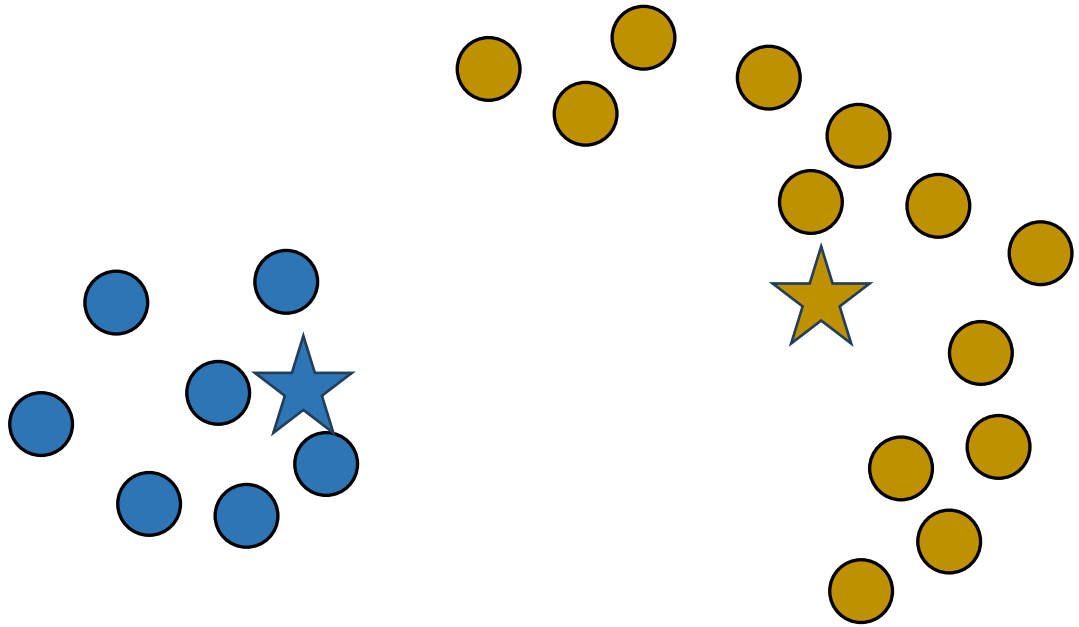
k-means step by step

Then: recalculate centroids!



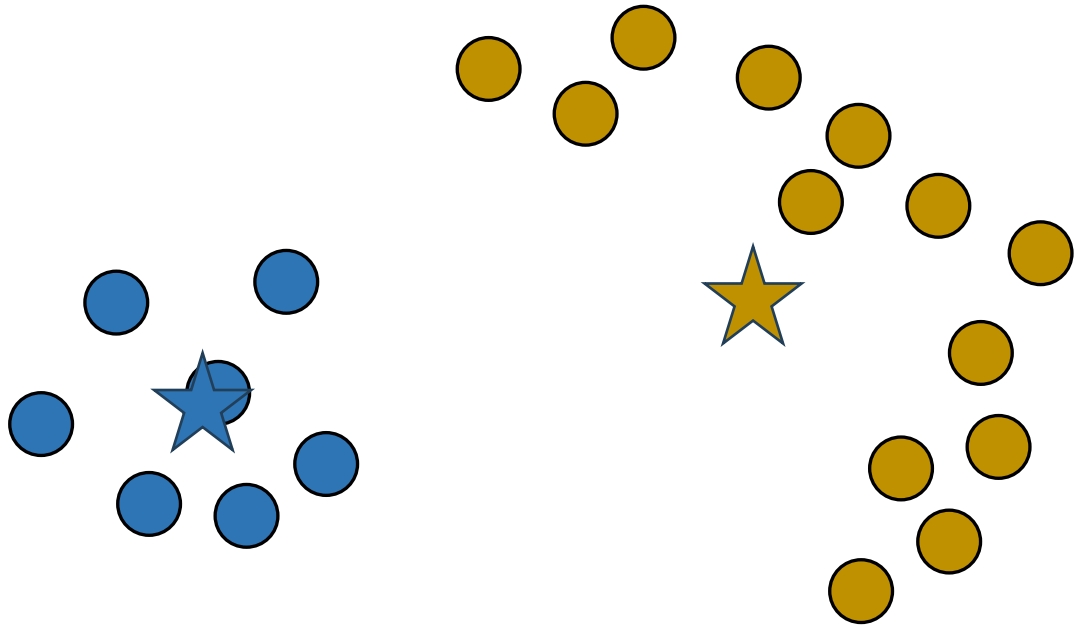
k-means step by step

Then: reassign labels (you see where this is going!)



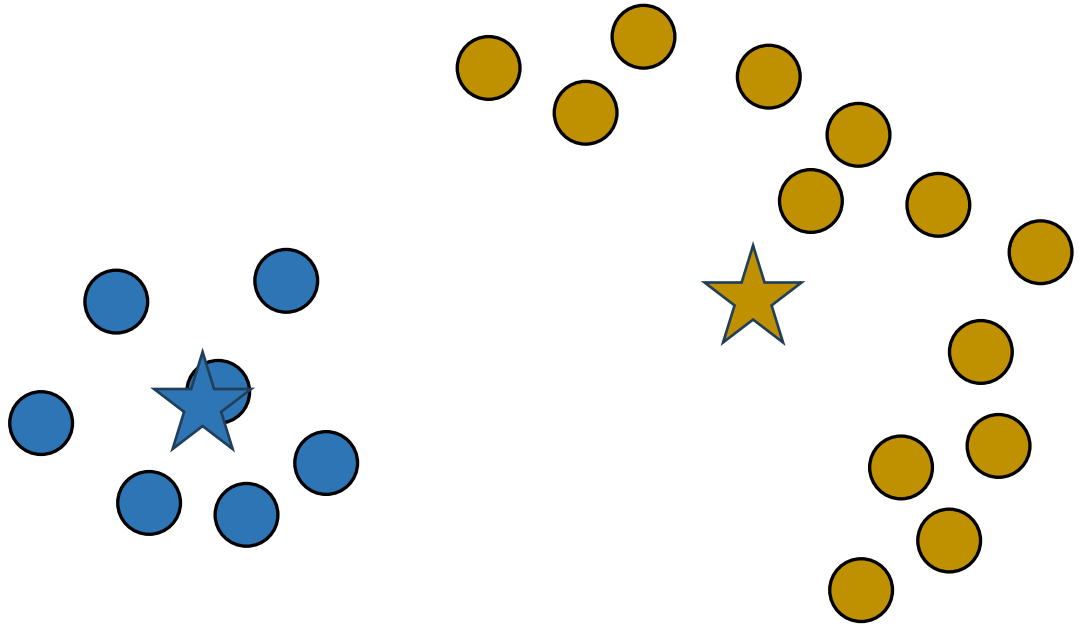
k-means step by step

Then: recalculate centroids!

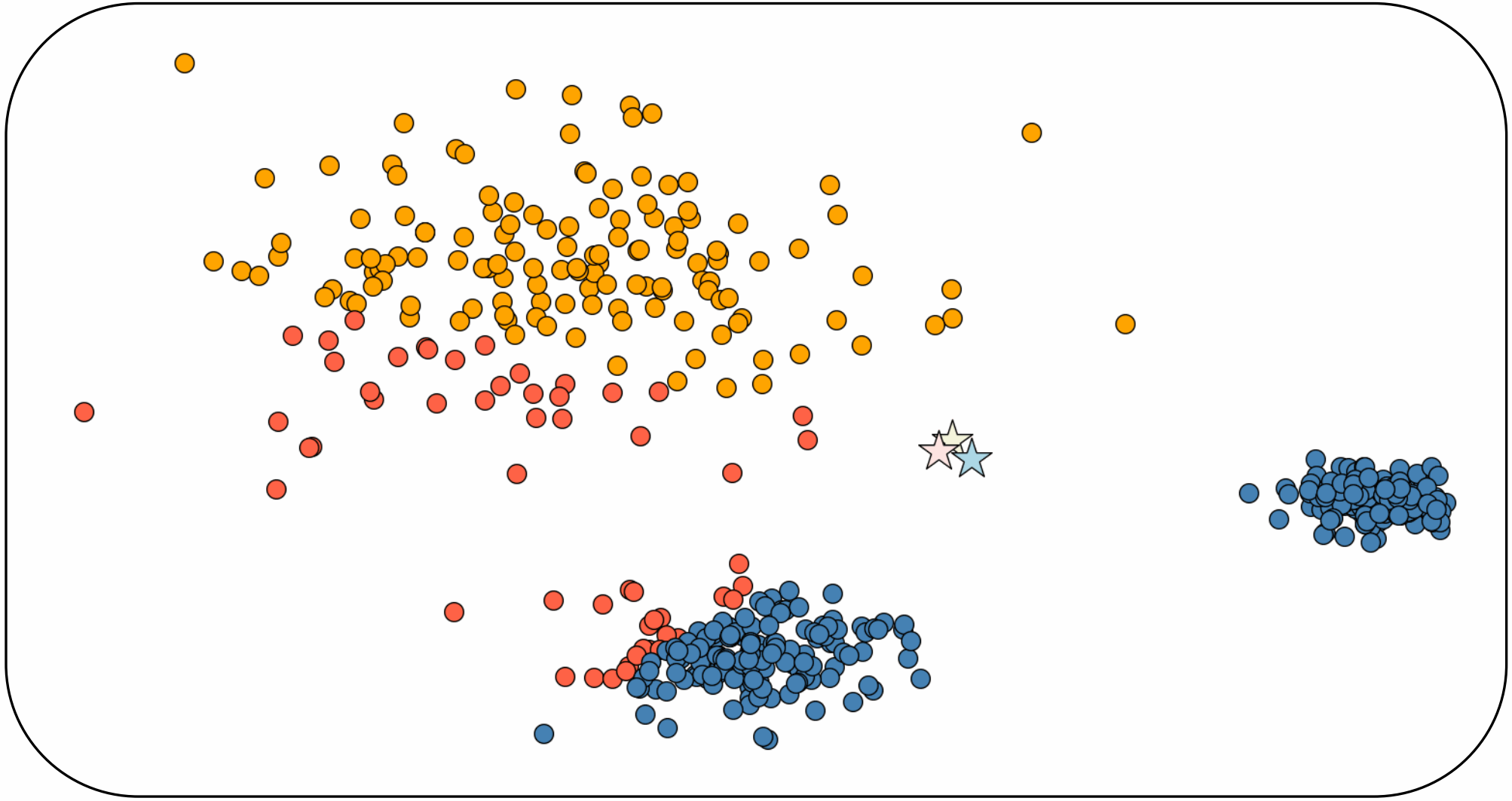


k-means step by step

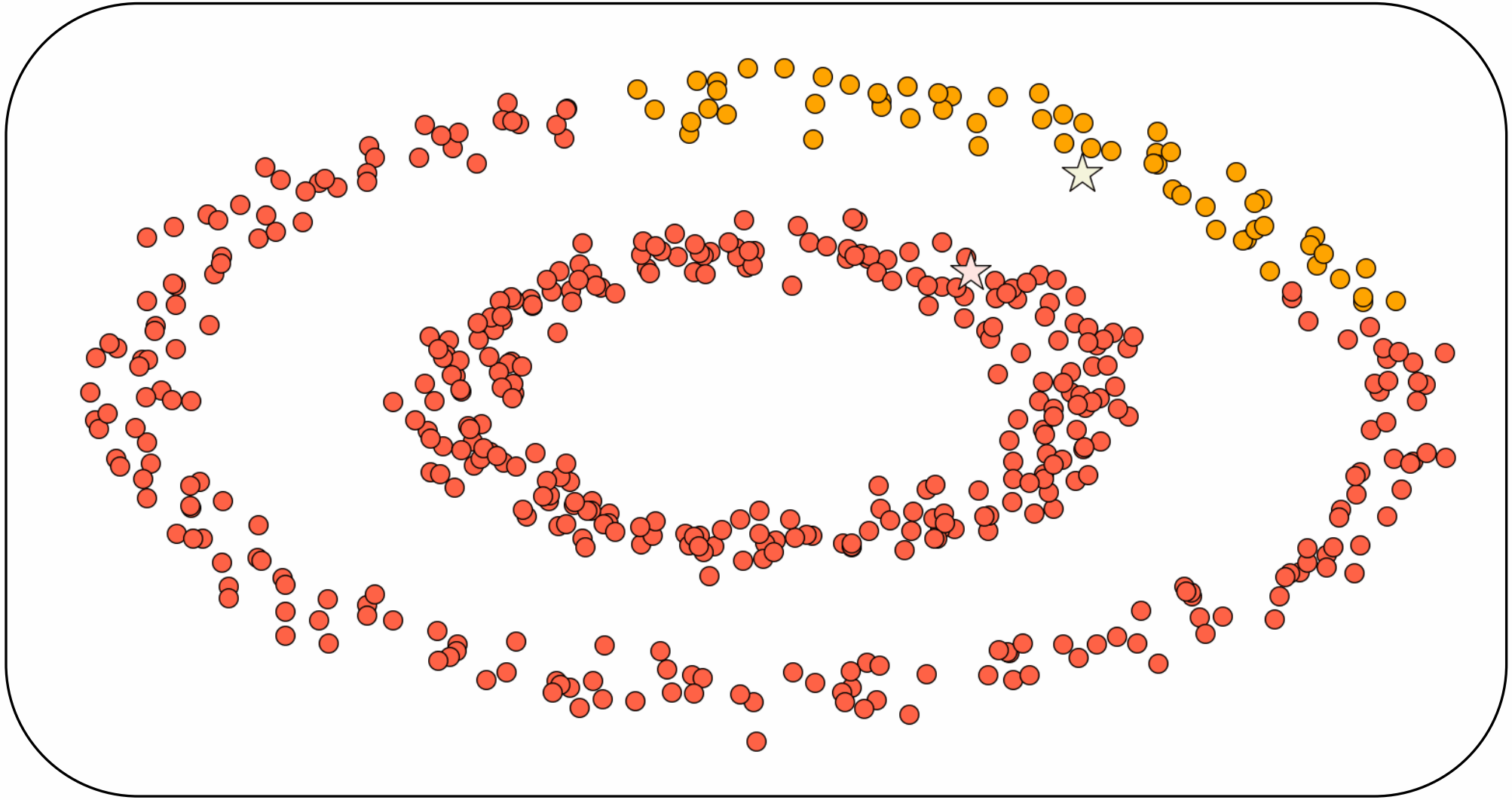
Then: reassign labels! Nothing changed, so we are done :)



k-means: first example



k-means: second example



k-means: advantages and downsides

Advantages:

1. Simple algorithm, very intuitive
2. Easily enhanced by combining it with other preprocessing methods *(we investigate one later!)*
3. Almost no hyperparameters

Downsides:

1. We have to set the number of clusters
2. Does not detect outliers, i.e., every point is assigned a cluster
3. Only finds *convex* clusters *(see the "circle in circle" example, which is not convex)*

An alternative: DBSCAN

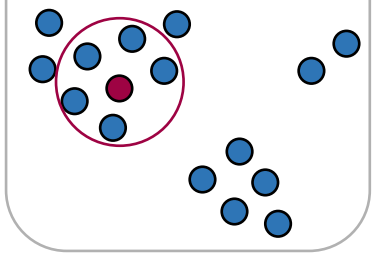
DBSCAN: We have n data samples x_i, $i \in [1, n]$. Choose the radius r used to search for neighbours around data points, and the number of neighbours N required to form a cluster. Do the following:

1. Pick one point without label assigned. Detect all points within radius r of it. If the number of these points is equal or larger to N, form a new cluster and add all these points to it.
2. Check for each neighbour whether they also have more than N neighbours within radius r. If so, add all neighbours that do not belong to another cluster.
   Repeat for all new points detected like that until no neighbours remain to be visited.
3. Start again at 1. for a new cluster. Stop when all points are assigned to a cluster.

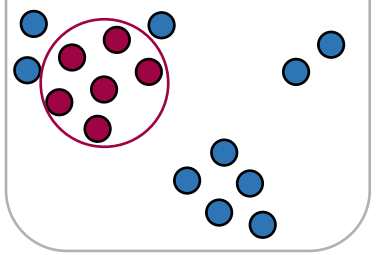
DBSCAN step by step

Example with $\mathrm{N} = 4$

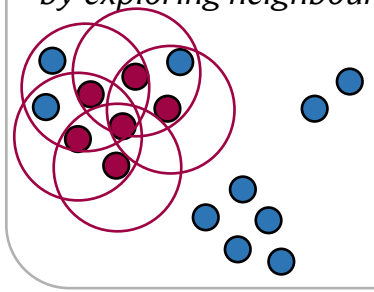
1. Select point, check neighbourhood



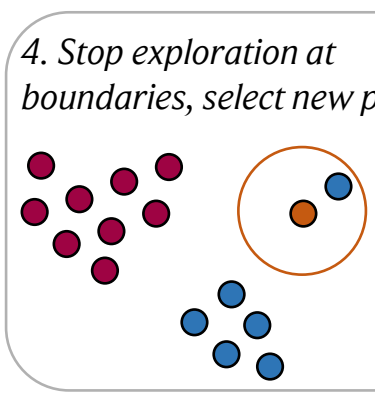
2. Form cluster if more than N neighbours



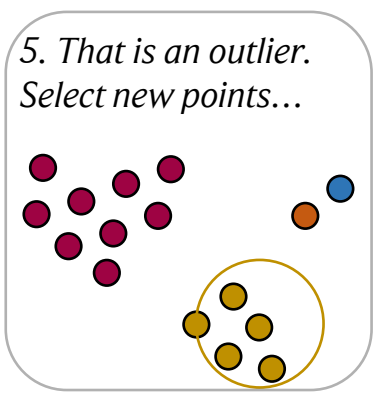
3. Increase cluster recursively by exploring neighbourhood



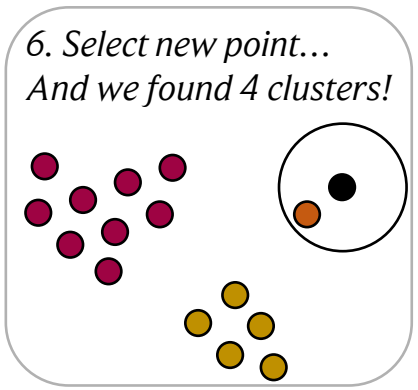
4. Stop exploration at boundaries, select new point



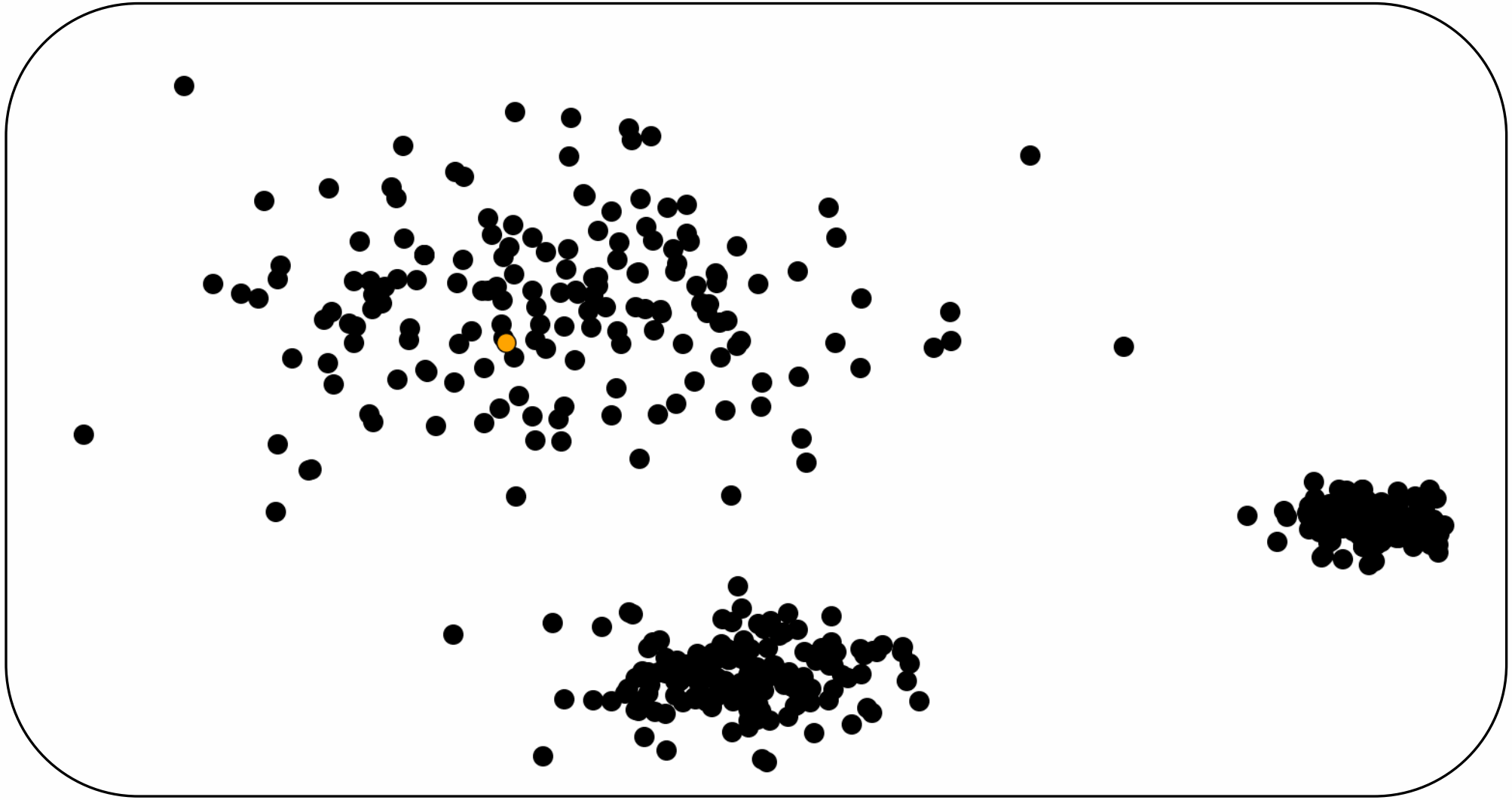
5. That is an outlier. Select new points…



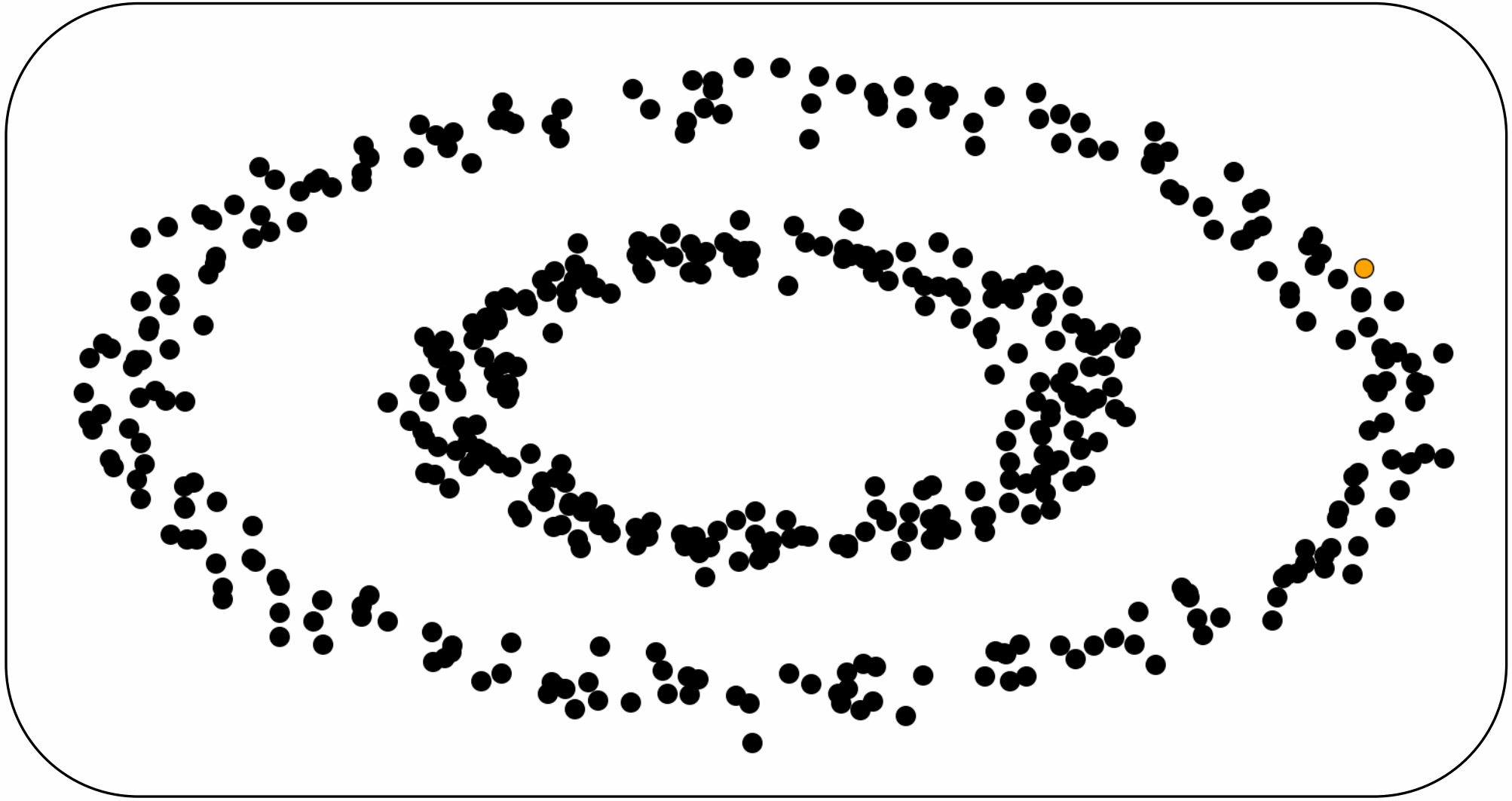
6. Select new point… And we found 4 clusters!



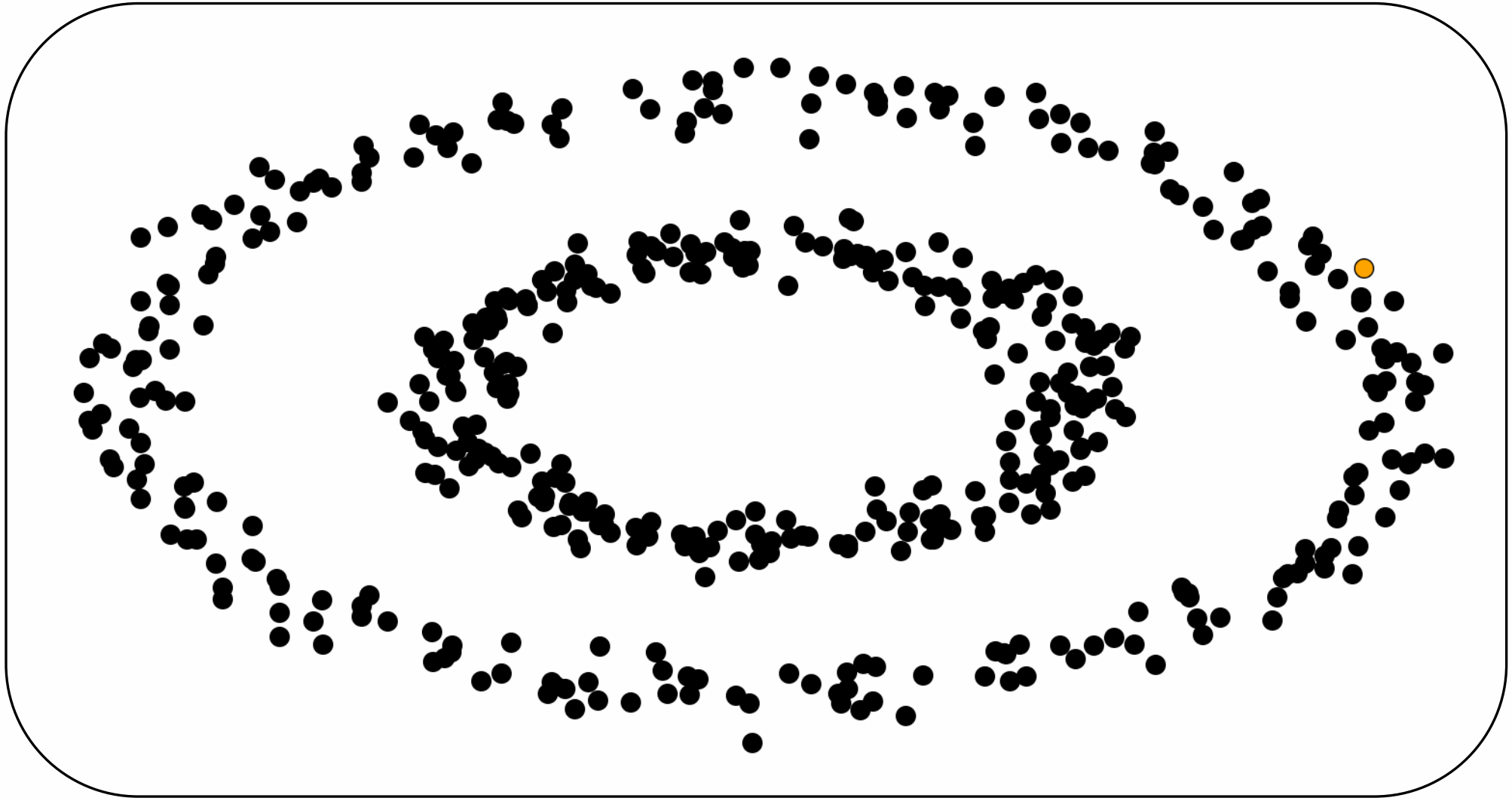
DBSCAN: first example



DBSCAN: second example



DBSCAN: second example gone wrong…



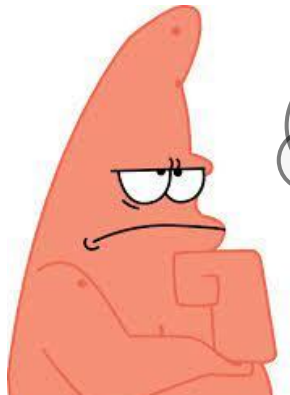
DBSCAN: advantages and downsides

Advantages:

1. It is also simple and intuitive
2. Finds clusters of any shape
3. No need to set a number of clusters
4. Detects outliers / is robust to noise

Downsides:

1. Has two hyperparameters, which heavily influence the quality of the found solution



Spectral clustering

The previous algorithms are quite intuitive, but is there a more grounded method?

Yes! Spectral clustering is a method grounded in graph theory!

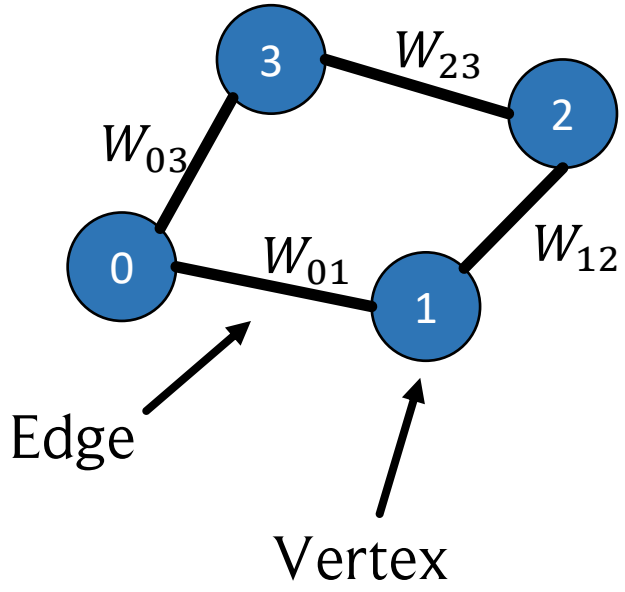
There is a variety of spectral clustering algorithms, but they are all very similar:

Spectral clustering: To find k clusters in n data samples x_i, do the following steps.

1. Transform your data to a graph, with edges encoding the similarity of data points.
2. Calculate the graph Laplacian.
3. Perform a spectral decomposition (i.e., eigenvectors). of the graph Laplacian.
4. Take the eigenvectors v_j corresponding to the k smallest eigenvalues. Create a matrix V, where the columns are these eigenvectors, i.e., $V = (v_0, v_1, \dots, v_k)$.
5. For every data point x_i, the i^{th} row V_i in this matrix is its new representation.
6. Perform k-means on this new data representation to find the clusters.

1. Transforming our data points into a graph

A **graph** is a collection of nodes/vertices V and edges E, $G = (V, E)$.
In our case, edges are weighted, i.e., have a value, and the graph is undirected ($W_{ij} = W_{ji}$)



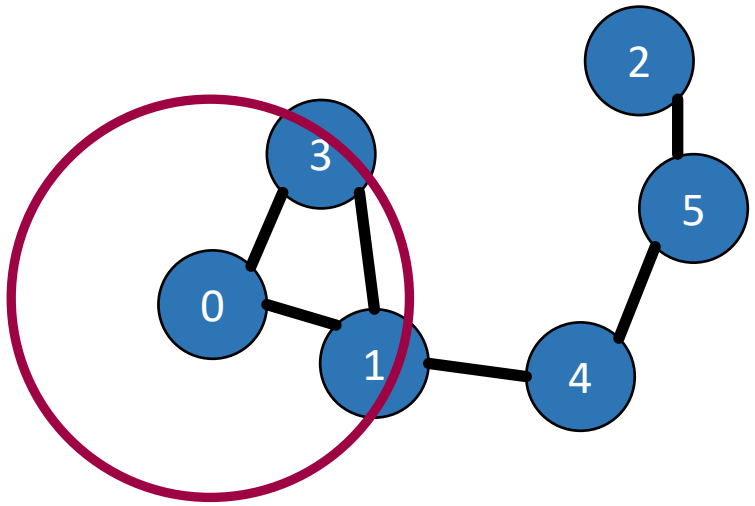
Super useful! Lots of problems can be phrased in terms of graphs!

1. Transforming our data points into a graph

Turning data into graph ⟶ **Adding edges** between data points that are similar.

There are several options on how to do this, for example:

1. Add edge between nodes within a certain radius ϵ



2. Connect the m nearest neighbours.

3. Connect all nodes and add a similarity metric to each edge, $W_{ij} = f(x_i, x_j)$ with $w_{ij} \geq 0$.
If $W_{ij} = 0$, the nodes are not connected.

Example for f: $f(x_i, x_j) = e^{-\frac{\|x_i - x_j\|_2^2}{2\,\sigma^2}}$
σ is a scale parameter indicating how far points can be apart to be considered "similar".

Note: *In case 1. and 2., we set $w_{ij} = 1$ if there is an edge, $w_{ij} = 0$ otherwise.*

2. Calculate the graph Laplacian

Graph Laplacian: We define the (unnormalized) graph Laplacian as

$$\boldsymbol{L = D - W},$$ with

W_{ij}: similarity matrix / adjacency matrix

D: degree matrix, $D_{ij} = 0$ if $i \neq j$

$D_{ii} = \sum_j W_{ij}$

Exercise: Proof the following properties.

1. For every vector $f \in \mathbb{R}^n$, we have $f^{\mathrm{T}} L f = \frac{1}{2}\sum_{i,j=1}^{n} W_{ij}\left(f_i - f_j\right)^2$
2. L is symmetric and positive definite (i.e., $f^{\mathrm{T}} L f \geq 0$ for all $f \in \mathbb{R}^n$)
3. The smallest eigenvalue of L is 0 with the constant one vector $(1,1,\ldots,1)$ as eigenvector.
4. L has non-negative, real-valued eigenvalues.

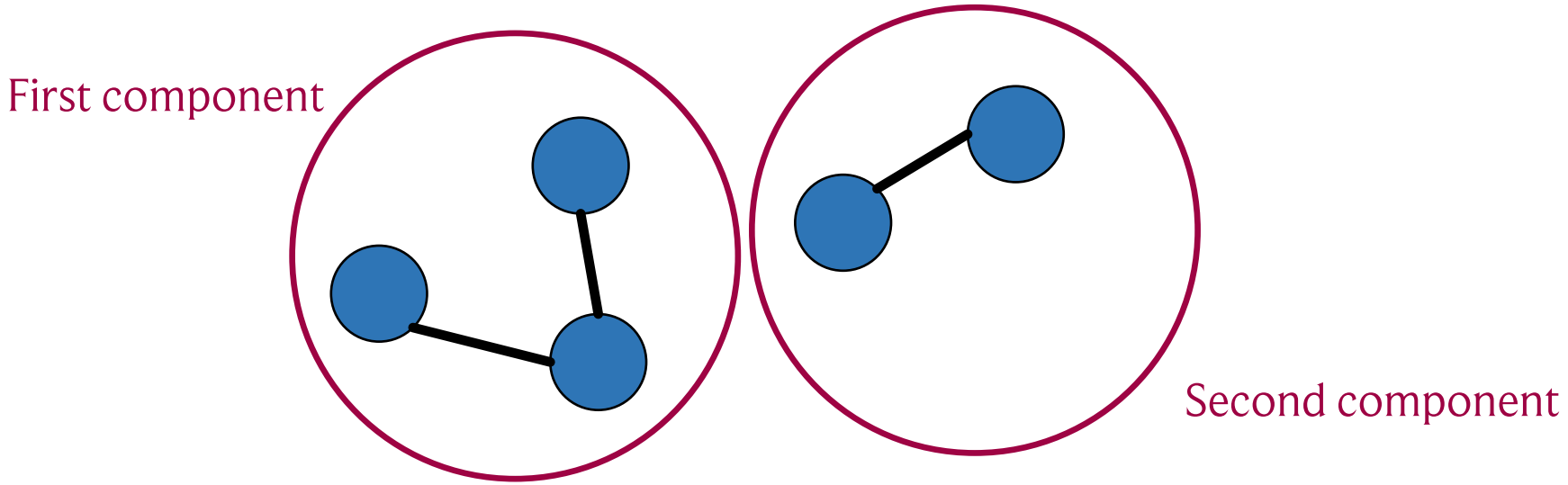
3. – 6. Spectral decomposition + k-means

We motivate this step via the following observation:

Let G be an undirected graph with non-negative weights. The number of **connected components** of G is given by the **multiplicity** of the 0 eigenvalue of the Laplacian L (i.e., the number of distinct eigenvectors with eigenvalue 0).

Connected component: set of vertices that are reachable from each other.

Proof: see handwritten notes



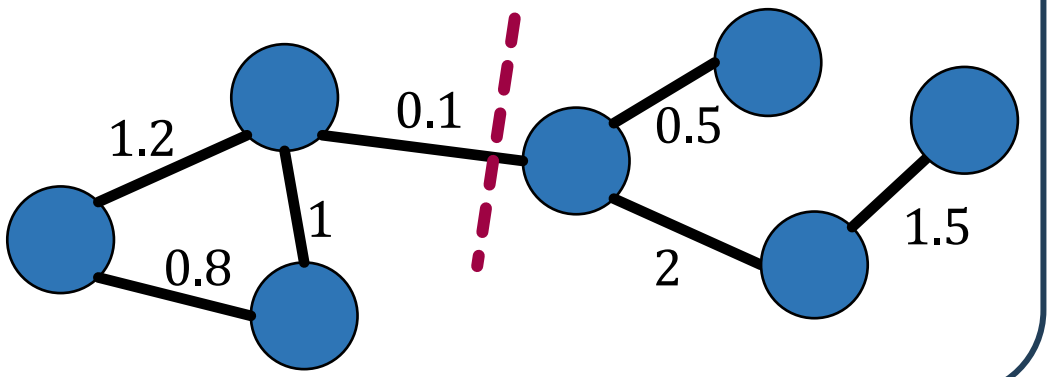
Thus: Spectrum of Laplacian describes connectedness. Useful for clustering!

3. – 6. Spectral decomposition + k-means

In fact, the described algorithm can be connected to a well-known graph problem

RatioCut: Split up the vertices of a graph into k sets A_n, such that:

1. $W\left(A_n, C(A_n)\right) = \sum_{i \in A_n, j \in C(A_n)} W_{ij}$ is minimal,
2. the sets are balanced, $|A_0| = |A_1| = \cdots = |A_k|$



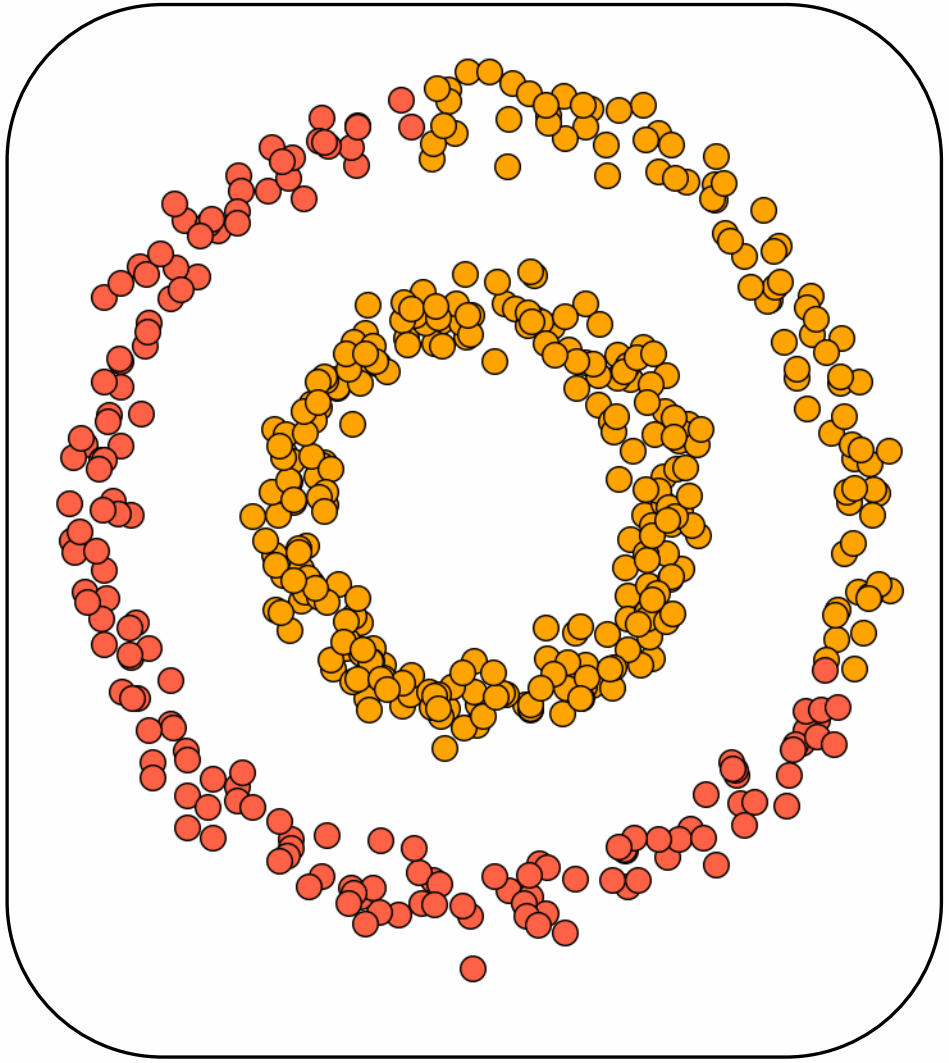
Surprisingly, the following holds:

$\min_{V^T V = I} \mathrm{Tr}(V^T L V)$, together with k-means on the rows of V, approximately solves RatioCut!

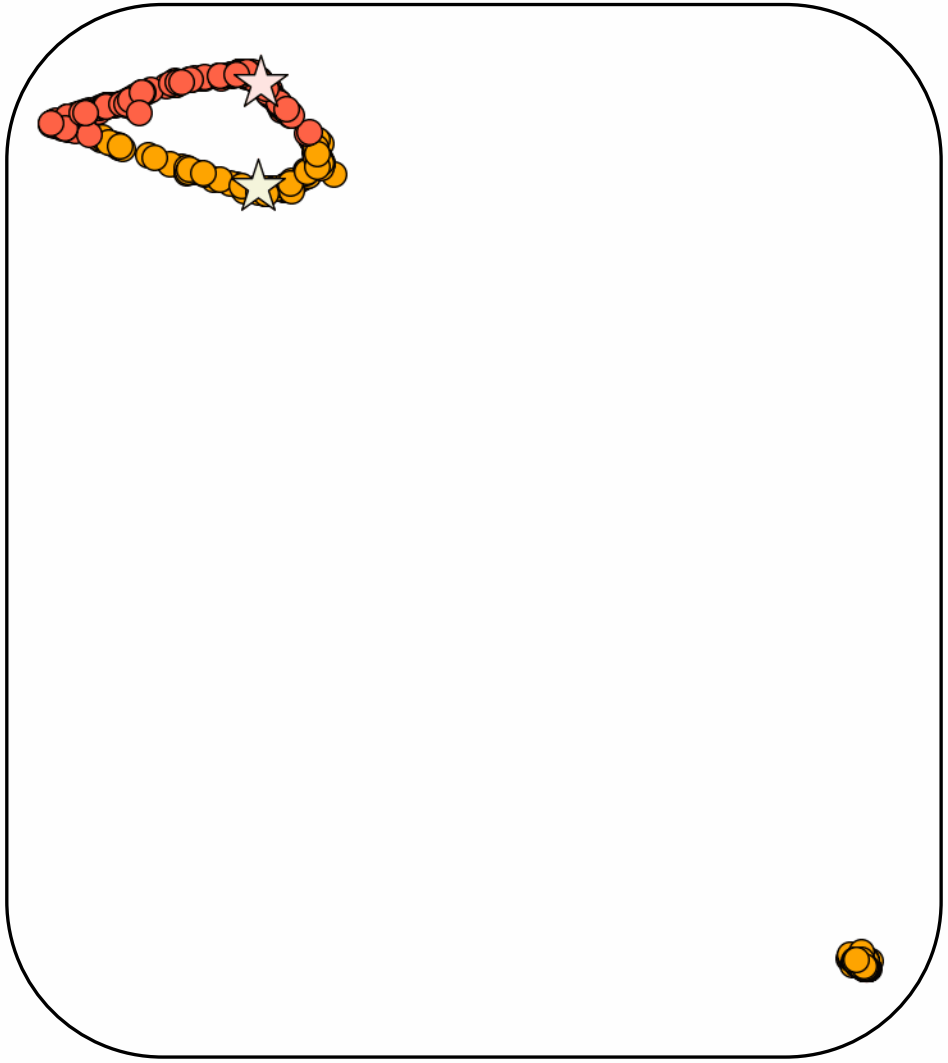
Proof: see handwritten notes

We know from earlier:
this minimization problem is solved by the eigenvectors with the k lowest eigenvalues!

Example: the circles again, but now k-means doesn't fail!



Real data points



Laplacian eigenvector representation

Spectral clustering: advantages and downsides

Advantages:

1. Theoretical grounded
2. Simple to implement:
   Laplace + Eigenvectors + k-means
3. We can look at the eigenvectors used for clustering!
4. Finds clusters of any shape

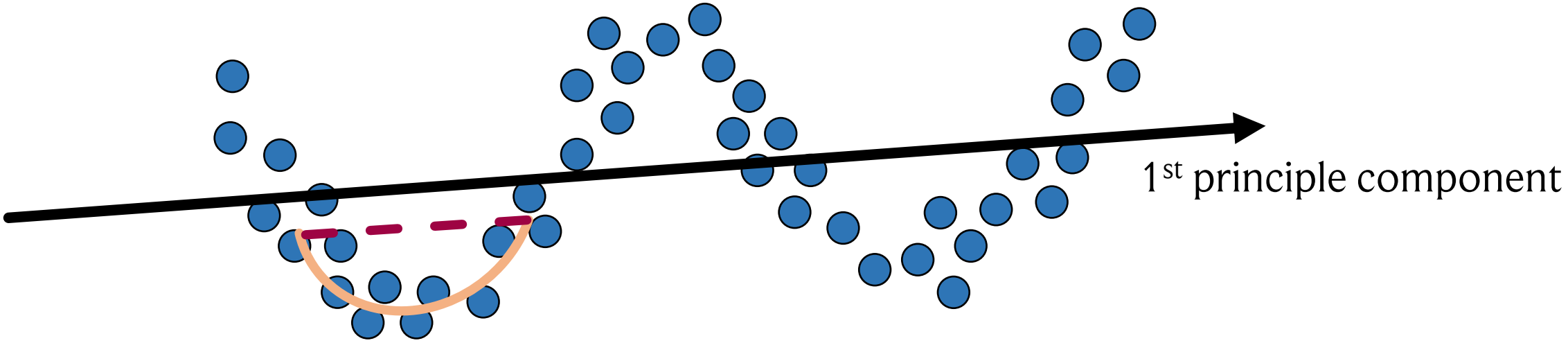
Downsides:

1. Number of clusters has to be chosen in advance
2. Distance function used to construct the Laplacian influences the quality of found clusters
3. If the constructed graph contains more connected components than clusters we are looking for, it will just return those components

Note: Spectral Clustering changes depending on how the Laplacian is constructed!

Can we use a similar approach for dimensionality reduction?

There is one limitation of PCA we haven't discussed so far: it is a **linear** method *(we project using a matrix multiplication)*.



PCA ignores that the data can lie on a submanifold, and distances should be measured **along that manifold**!

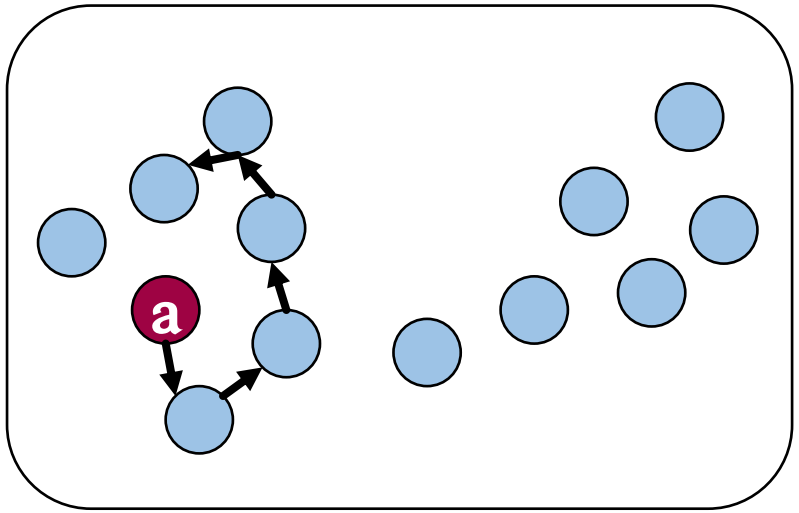
→ We need a non-linear method!

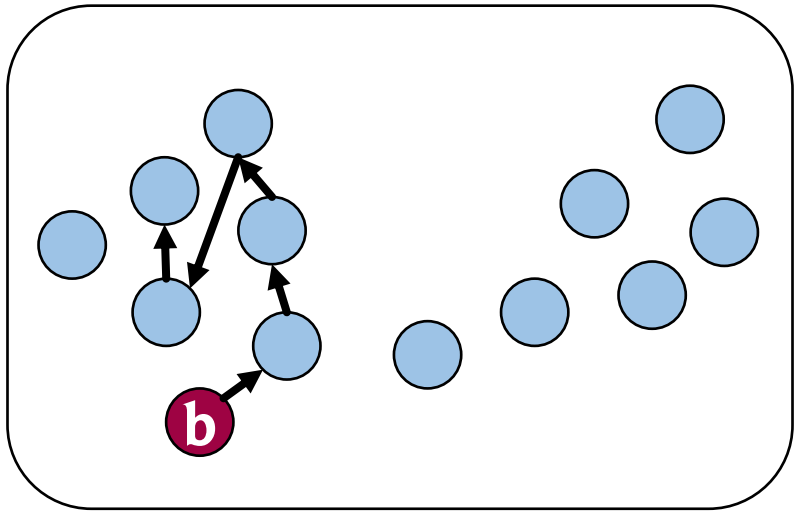
A new picture: performing random walks on data

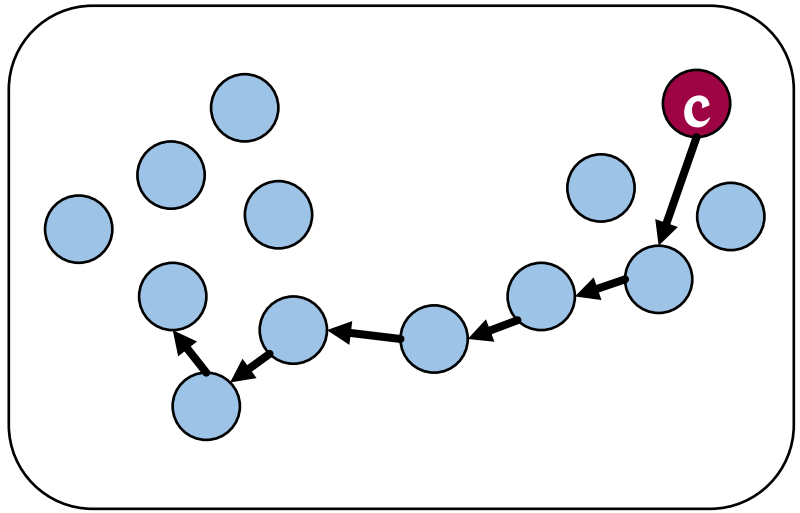
Let's assume again that our data is a fully connected graph.

The idea: Similarity is obtained through random walks on the graph. If random walks started from two different nodes are, on average, highly similar, this means that both nodes should have similar low-dimensional representations.

Example: Random walks starting from **a** and **b** are similar, while those from **c** are very different.





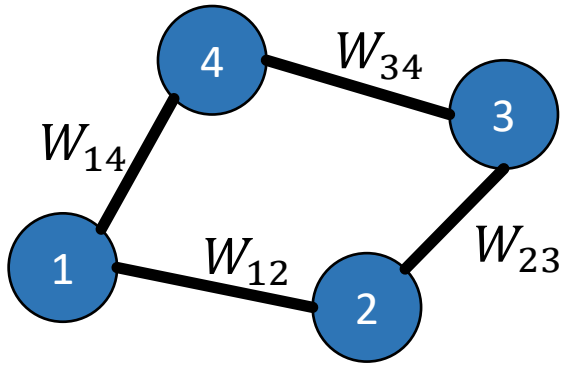


Constructing the random walk

1. As for spectral clustering, we turn our data into a graph by defining edge weights that measure similarity between data points:

$$W_{ij} = e^{-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}} \text{ for } n \text{ data samples } x_i$$

And $\boldsymbol{D}$ with: $D_{ij} = 0$ if $i \neq j$, $D_{ii} = \sum_j W_{ij}$



2. Transition matrix: We then define a matrix that contains the probabilities to hop from any node to any other in a single step:

$$\boldsymbol{M} = \boldsymbol{D}^{-1}\boldsymbol{W}$$

3. Random walk: Assume we perform a random walk using this matrix, and at time t our walker is visiting node $Z(t)$. Then:

$$p(Z(t) = j \mid Z(0) = i) = (\boldsymbol{M}^t)_{ij}$$

Details: see handwritten notes

Metric for embedding: similarity of random walks!

4. The probabilities where we can end up are given by

$$\boldsymbol{e}_i^T \boldsymbol{M}^t = [p(Z(t) = 1 \mid Z(0) = i), \ldots, p(Z(t) = n \mid Z(0) = i)]$$

i^{th} row of $\boldsymbol{M}^t$

Probability of ending up at node n when starting from i.

5. Define: data samples i and j are **highly similar if their random walks are similar**:

$$\boldsymbol{e}_i^T \boldsymbol{M}^t \simeq \boldsymbol{e}_j^T \boldsymbol{M}^t$$

Details: see handwritten notes

Diffusion maps: spectral decomposition on random walks

$\boldsymbol{e}_i^T \boldsymbol{M}^t$ has n elements, so it is not necessarily good for obtaining low-dimensional representations of the data. But we can extract the most important components instead by spectrally decomposing $\boldsymbol{M}$!

$$\boldsymbol{M} = \sum_{k=1}^{n} \lambda_k \boldsymbol{\phi}_k \boldsymbol{\psi}_k^T$$

Where $\boldsymbol{\phi}_k$ are right-eigenvectors and $\boldsymbol{\psi}_k^T$ left-eigenvectors of $\boldsymbol{M}$

$$\boldsymbol{\psi}_k^T \boldsymbol{M} = \lambda_k \boldsymbol{\psi}_k^T , \qquad \boldsymbol{M}\phi_k = \lambda_k \phi_k , \qquad \boldsymbol{\phi}_i^T \boldsymbol{\psi}_j = \delta_{ij}$$

From this, we get:

$$\boldsymbol{e}_i^T \boldsymbol{M}^t = \sum_{k=1}^{n} \lambda_k^t \phi_{k,i} \boldsymbol{\psi}_k^T$$

Exercises: Show

1. $\boldsymbol{M}^t = \sum_{k=1}^{n} \lambda_k^t \boldsymbol{\phi}_k \boldsymbol{\psi}_k^T$
2. The 1-vector is an eigenvector of $\boldsymbol{M}$ with eigenvalue 1.
3. All eigenvalues of $\boldsymbol{M}$ satisfy $|\lambda_k| \leq 1$

Details: see handwritten notes

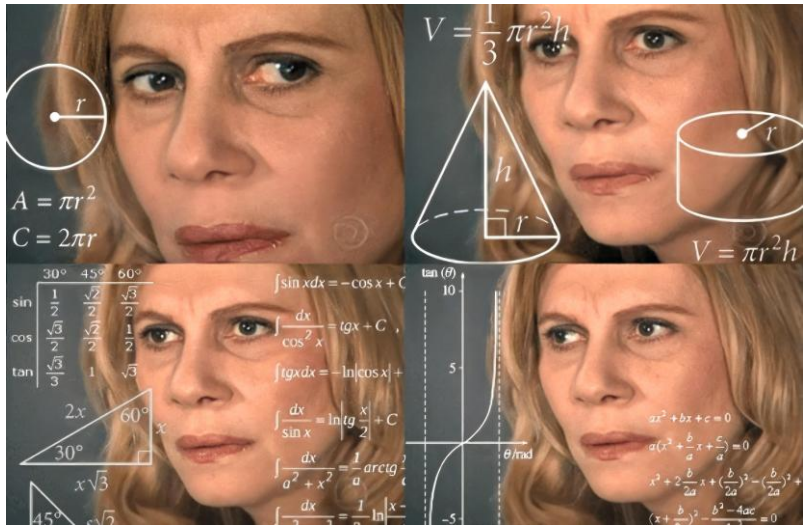
Intermezzo: what does this mean?!

$$\boldsymbol{M} = \sum_{k=1}^{n} \lambda_k \boldsymbol{\phi}_k \boldsymbol{\psi}_k^T$$

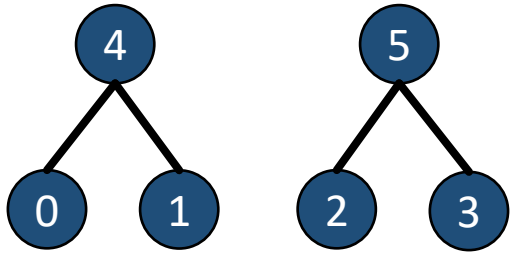
…from which we typically jump to…

$\boldsymbol{\phi}_k$ — Set of nodes

$\boldsymbol{\psi}_k^T$ — Set of nodes



Example (made up):



$\boldsymbol{\phi}_0 = (1,1,0,0,0,0)$ $\quad \boldsymbol{\psi}_0 = (0,0,0,0,1,0)$

$\boldsymbol{\phi}_1 = (0,0,1,1,0,0)$ $\quad \boldsymbol{\psi}_1 = (0,0,0,0,0,1)$

Diffusion maps: spectral decomposition on random walks

Using the results from the exercises, we know that

1. ϕ_0 is the 1-vector and thus always the same, meaning we can neglect it.
2. For large enough t, the smallest eigenvalues will quickly go to 0.

Thus, we truncate our representation, keeping only the $d < n$ components with the largest (non-trivial) eigenvalues:

$$\boldsymbol{e}_i^T \boldsymbol{M}^t \approx \sum_{k=2}^{d+1} \lambda_k^t \phi_{k,i} \boldsymbol{\psi}_k^T$$

From this, we can immediately extract the low-dimensional representations of our data:

For data sample $\boldsymbol{x}_i$, its **diffusion map (Laplacian Eigenmap) representation** is given by:

$$\boldsymbol{y}_i^{t,d} = \left(\lambda_2^t \phi_{2,i}, \dots, \lambda_{d+1}^t \phi_{d+1,i}\right)$$

Details: see handwritten notes

Recap: from random walks to dimensionality reduction

The most important steps were:

1. Create a graph from the data, with edge weights encoding how similar data points are.
2. Construct a random walk on the graph. We did this by defining a transition matrix $\boldsymbol{M}$.
3. Similarity of data points is measured by how similar their random walks are!
4. Finally: an embedding is found through the eigen-decomposition of $\boldsymbol{M}$.

The following theorem formalizes this:

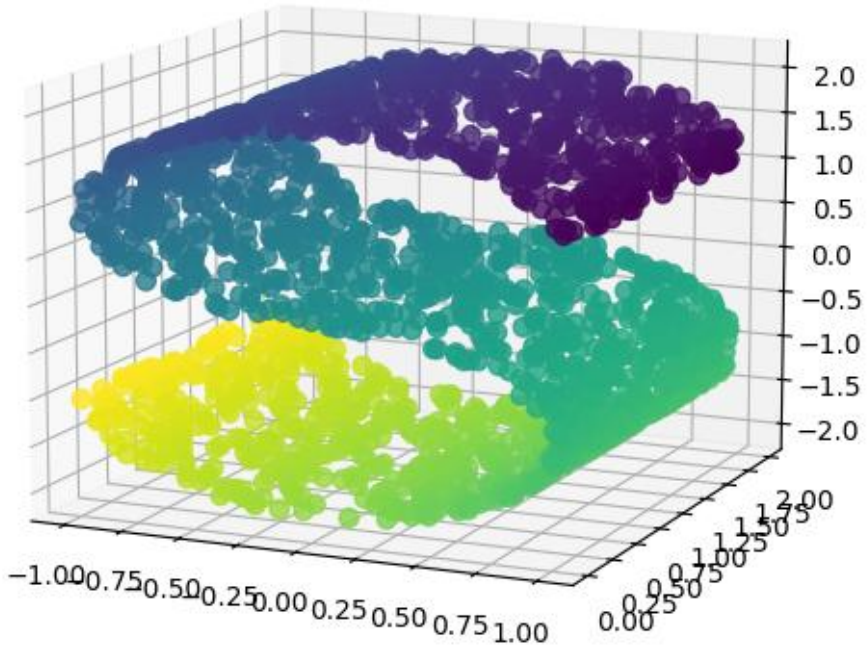
For any pair of nodes i and j, we have:

$$\|\boldsymbol{y}_i^{t,n} - \boldsymbol{y}_j^{t,n}\|^2 = \sum_l^n \frac{1}{D_{ll}} (p(Z(t) = l \mid Z(0) = i) - p(Z(t) = l \mid Z(0) = j))$$

Details: see handwritten notes

Example: embedding the 3D S-curve in 2D

Dataset



PCA

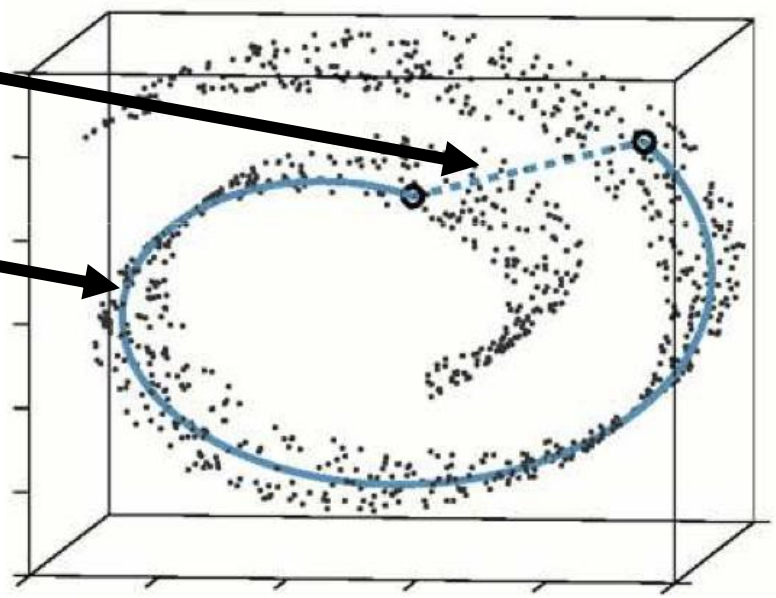
Diffusion map

Note: Here, it is not about the specific algorithms but about the ideas on how to embed data such that certain features are conserved! E.g., by measuring distances differently, you will arrive at different algorithms!

Example of alternative methods: ISOMAP

Euclidean distance (in sub-space)

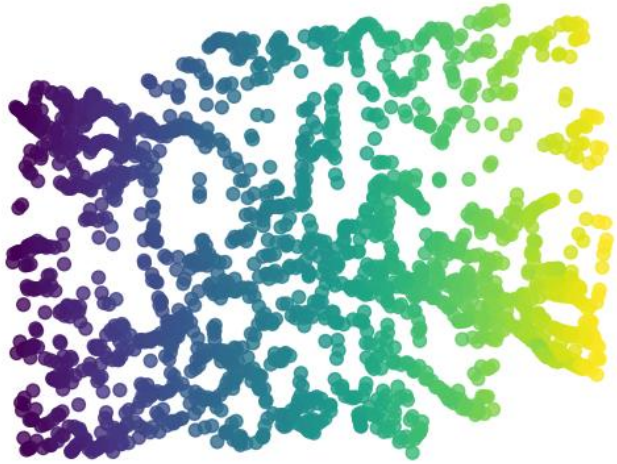
Geodesic distance (along sub-manifold)

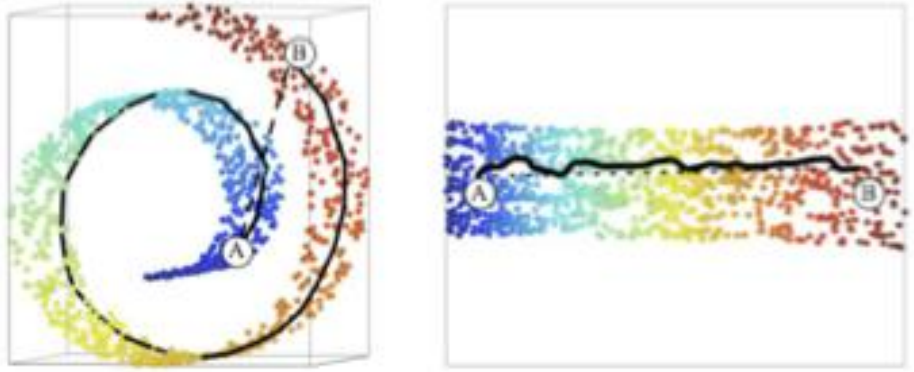


ISOMAP (briefly…)

1. Construct neighbourhood graph again
2. As distances, get the geodesic distance (e.g. find shortest paths using Dijkstra's algorithm)
3. Find vector representations that preserve geodesic distances

S-Curve using **ISOMAP**





Images: Mihai Cucuringu, University of Oxford, 2023

Outro: Spectral Clustering with normalized Laplacian

We can use diffusion maps to arrive at yet another clustering algorithm!

Basically: calculate the diffusion maps (with $d = k$) and perform k-means using them.

In fact, this way we recover spectral clustering using a normalized Laplacian (setting $t = 0$)!

Spectral clustering: Perform the same method, but:

1. Use the random walk Laplacian: $\boldsymbol{L_{rw}} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W} = \boldsymbol{I} - \boldsymbol{M}$
   Note that $\boldsymbol{L_{rw}}$ has the same left and right eigenvectors as $\boldsymbol{M}$ with eigenvalues $1 - \lambda_l$
2. We take the right-eigenvectors $\phi_l$corresponding to the k lowest eigenvalues.
3. The rest is identical to the spectral clustering algorithm we introduced for $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$

Note: we usually get the eigenvectors v_l of the normalized Laplacian $\boldsymbol{L_N} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$, from which the right-eigenvectors of $\boldsymbol{L_{rw}}$ are obtained by rescaling: $\phi_l = D^{-1/2}v_l$.