

UNIVERSITY OF WARWICK

Lecture 11

t-SNE

Gabriele Pergola and Amina Asif

Machine Learning Algorithms and Practice
CS429

November 2026

Motivation: why PCA is not enough

What a flat projection can, and cannot, show

Our example: images of handwritten digits

Our running example: **1,797** small images of handwritten digits, from 0 to 9 (a standard dataset in scikit-learn).

Each image is **8 × 8 pixels**, and each pixel is a grey level from 0 (white) to 16 (black).

Read row by row, an image becomes a list of **64 numbers**: one point in a 64-dimensional space.

We cannot plot 64 dimensions. t-SNE will draw these 1,797 points in two.



Three examples of each digit

0	0	7	15	13	1	0	0
0	8	13	6	15	4	0	0
0	2	1	13	13	0	0	0
0	0	2	15	11	1	0	0
0	0	0	1	12	12	1	0
0	0	0	0	1	10	8	0
0	0	8	4	5	14	9	0
0	0	7	13	13	9	0	0

→

x = (0, 0, 7, 15, 13, 1, 0, 0, 0, 8, …)

64 numbers: one point in 64 dimensions

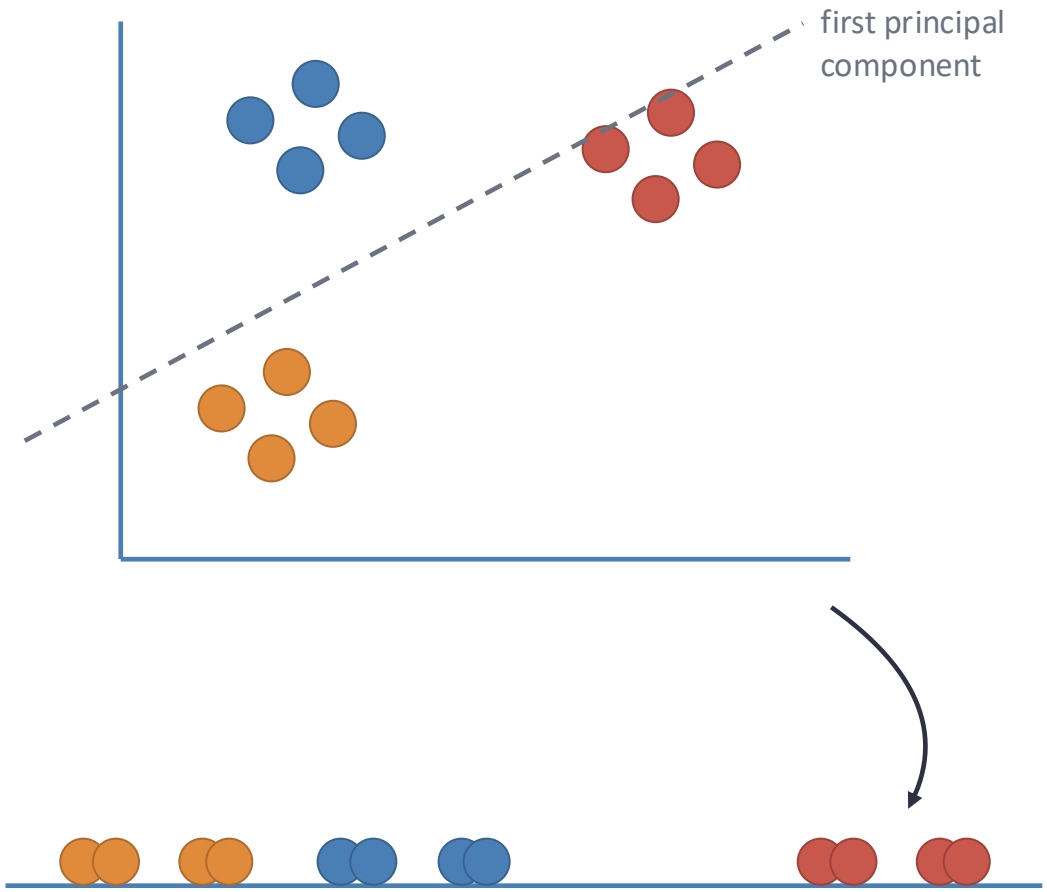
Recap: PCA finds the best flat view

In the previous lecture, **PCA** found the direction along which the data vary most, the **first principal component**, and projected the data onto it.

PCA is **linear**: it rotates the data and drops dimensions, keeping as much of the **variance** (the overall spread) as possible.

On our simple example it works well: along the first component, the three clusters stay apart.

So why do we need anything else? Here are three situations where PCA fails.



Projection onto the first component: the three clusters stay apart

PCA fails (1): most variance is not best separation

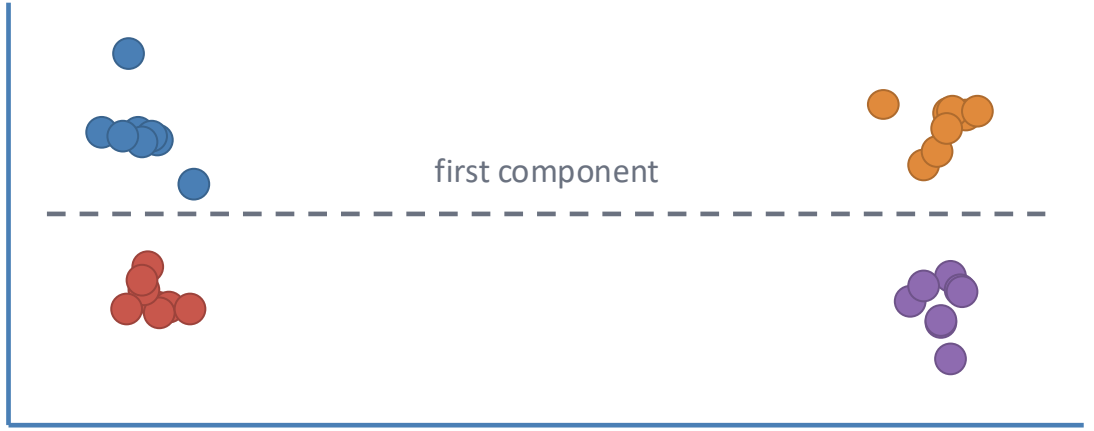
Four compact groups: two on the left, two on the right.

The data vary most from **left to right**, so that is the direction PCA keeps. The small gap between the top and bottom groups lies in the direction of **least** variance, and PCA drops it.

On the PCA line, blue falls on top of red, and orange on top of purple.

t-SNE only asks that **neighbours stay neighbours**, and all four groups stay apart.

Most variance is not the same as best separation.





PCA (first component): pairs of groups fall on top of each other



t-SNE: all four groups stay apart

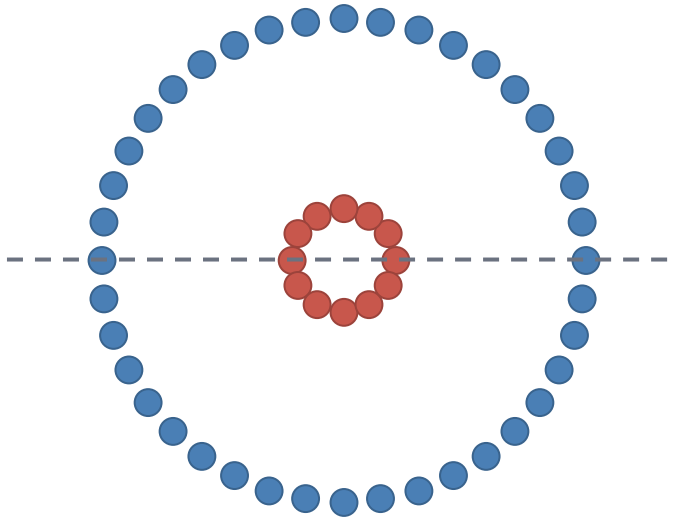
PCA fails (2): curved structure

Here the groups are **curved**: a small ring inside a larger one.

Whichever straight line PCA chooses, the large ring is squashed on top of the small one: no flat projection can keep them apart.

t-SNE does not need a straight line. Points that are close along each ring stay close, so the small ring ends up on its own. The large ring is simply cut open to lie along the line.

PCA can only rotate and flatten; it cannot unfold.





PCA: the small ring lands in the middle of the large one



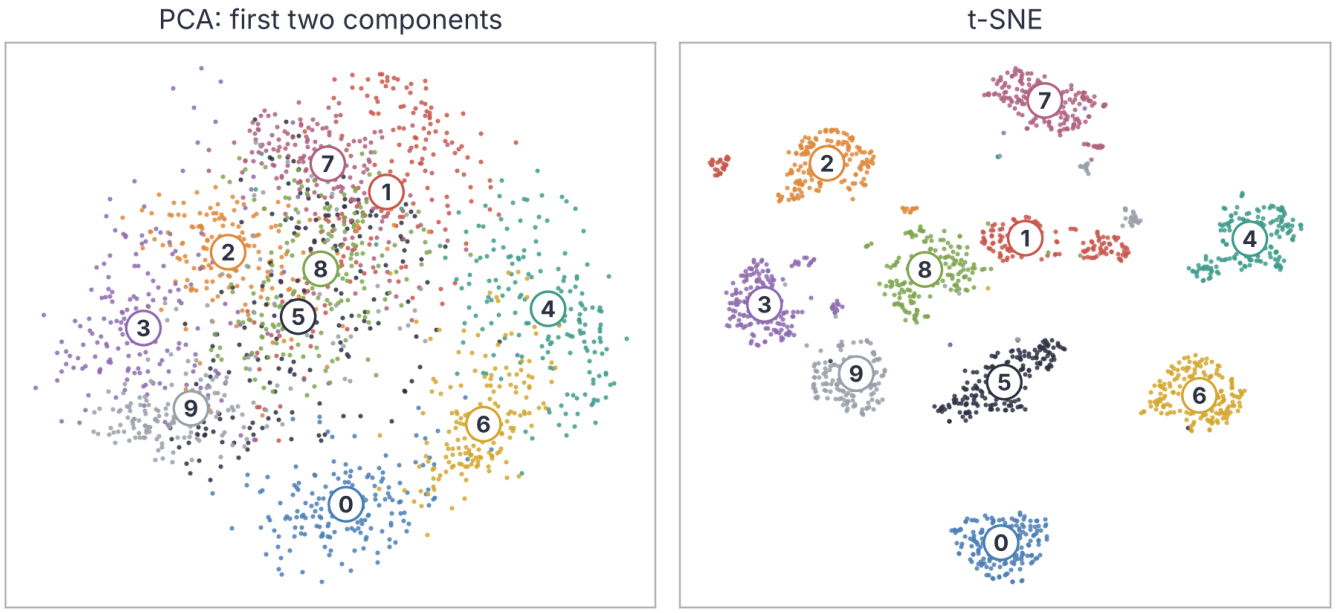
t-SNE: the small ring stays on its own

PCA fails (3): many groups, many dimensions

In the previous lecture, **PCA** found the best **flat** view of the data: the directions along which it varies most.

On the digits, the first two principal components keep only **28.5%** of the variance. Many digits overlap: no flat view can separate ten groups that live in 64 dimensions.

t-SNE asks for something different: not the best flat view, but that **neighbours stay neighbours**. Now the ten digits separate.



The same 1,797 images in both plots; colours show the true digit, which neither method was given.

What we need instead

All three failures point to the same idea.

Instead of asking which **flat view** keeps the most variance, ask where each point should go so that **its neighbours stay its neighbours**.

This is exactly what t-SNE does. The rest of the lecture shows how, step by step.

1 **PCA:** Most variance is not best separation

t-SNE: keep **neighbourhoods**, not variance

2 **PCA:** Curved structure cannot be flattened

t-SNE: allow any **non-linear** placement of the points

3 **PCA:** Many groups in many dimensions

t-SNE: only ask that **similar points end up close**, in 2-D

What t-SNE does

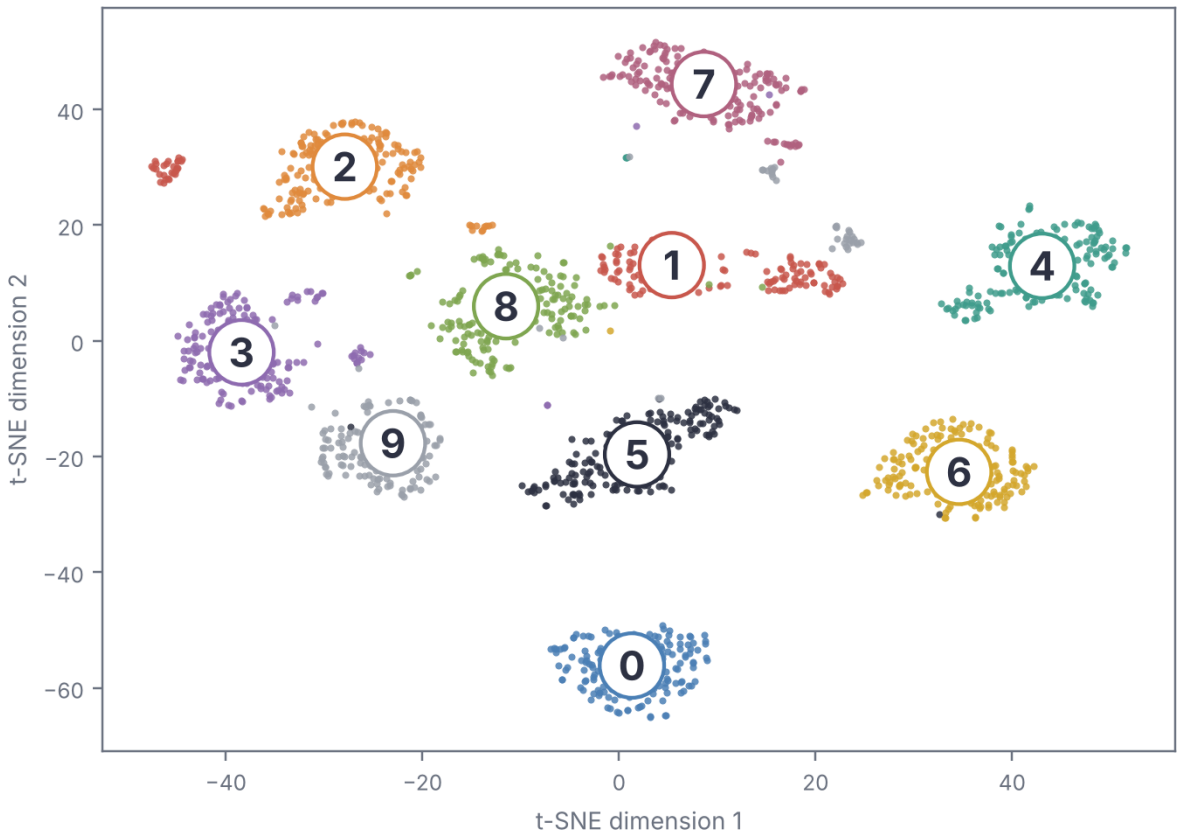
Drawing high-dimensional data in two dimensions, without losing the clusters

t-SNE in practice

t-SNE (t-distributed Stochastic Neighbour Embedding) takes data with many dimensions and draws it in two (or one), so that points that are similar end up close together.

Here, each point is one of the digit images from the previous slide: a point in **64** dimensions.

t-SNE was never told which digit each image shows. The colours are added afterwards, and the clusters it finds match the digits.

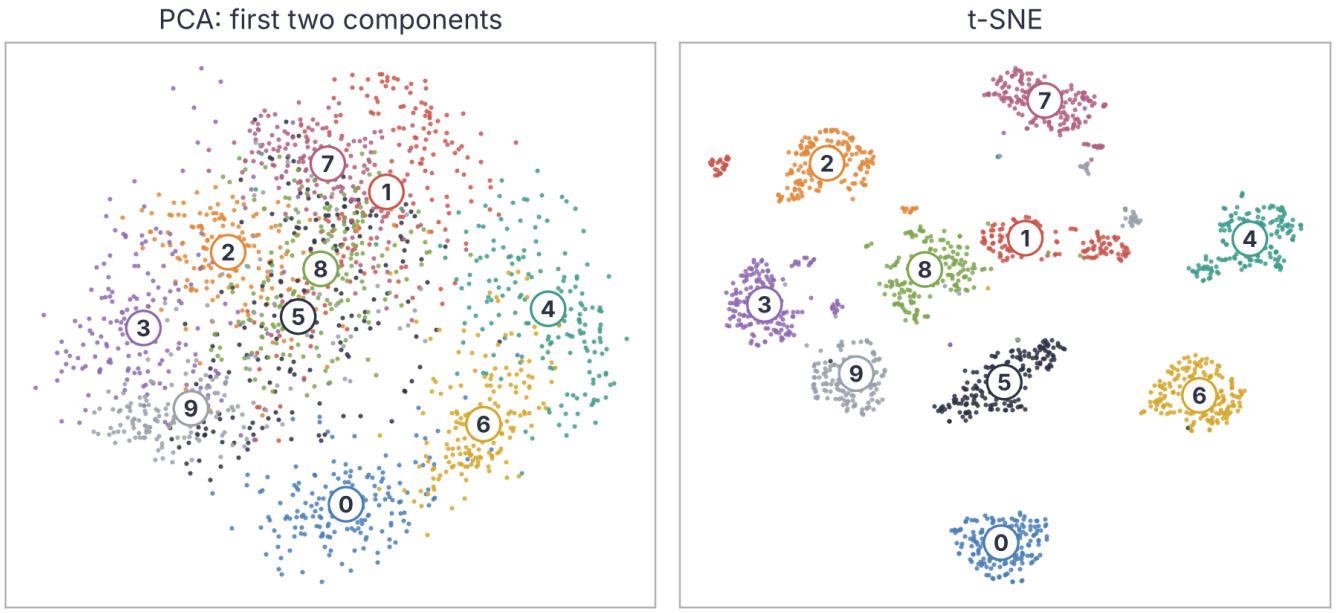


From PCA to t-SNE

In the previous lecture, **PCA** found the best **flat** view of the data: the directions along which it varies most.

On the digits, the first two principal components keep only **28.5%** of the variance. Many digits overlap: no flat view can separate ten groups that live in 64 dimensions.

t-SNE asks for something different: not the best flat view, but that **neighbours stay neighbours**. Now the ten digits separate.

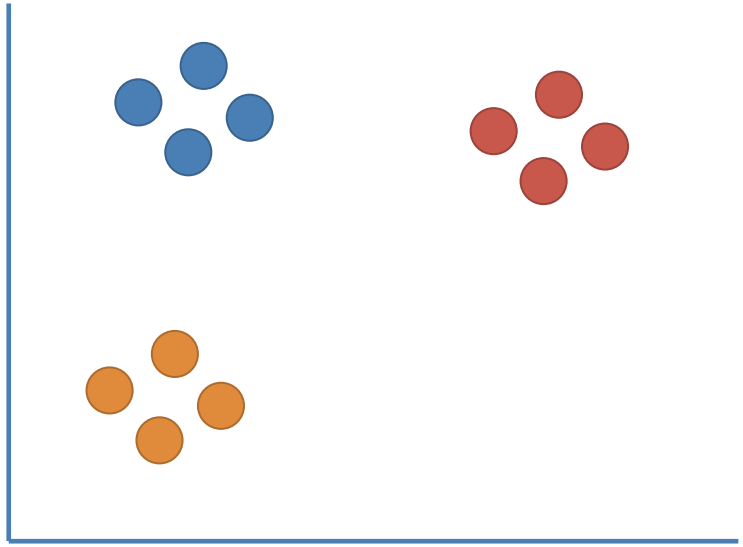


The same 1,797 images in both plots; colours show the true digit, which neither method was given.

A simple example

Here is a basic 2-D scatter plot, with three clusters of four points each.

We will walk through how t-SNE would transform this graph…

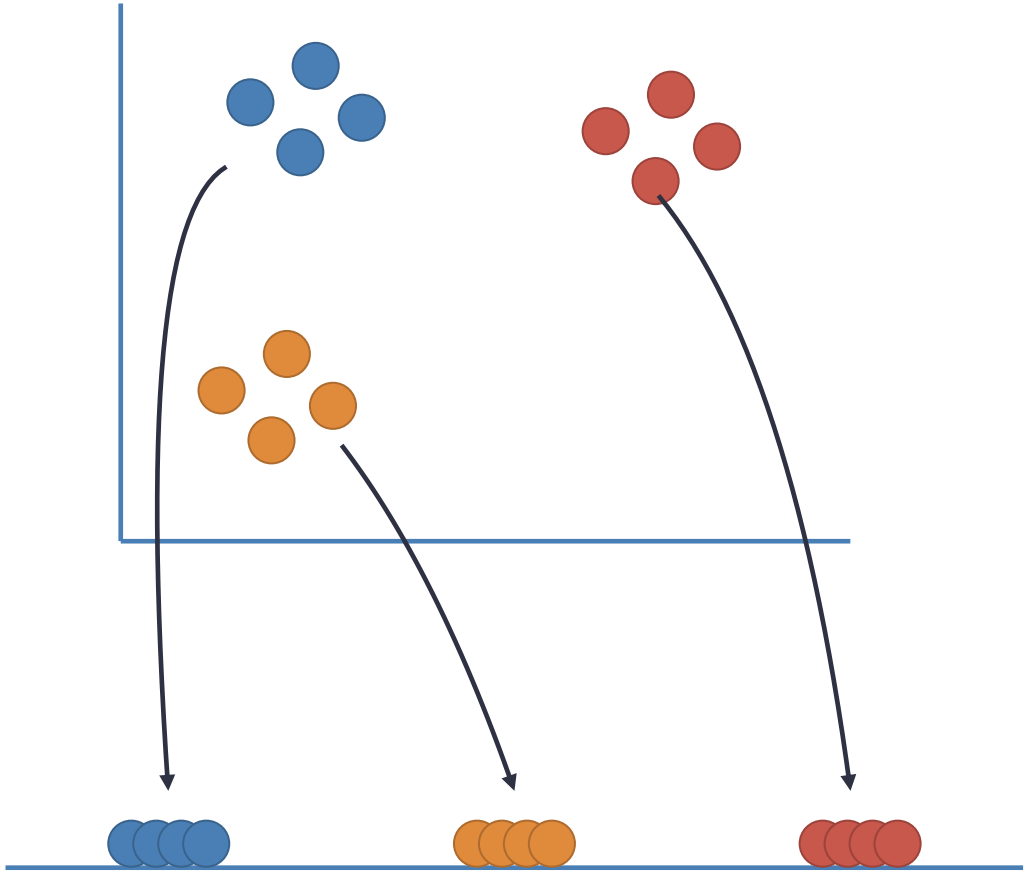


A simple example

Here is a basic 2-D scatter plot, with three clusters of four points each.

We will walk through how t-SNE would transform this graph…

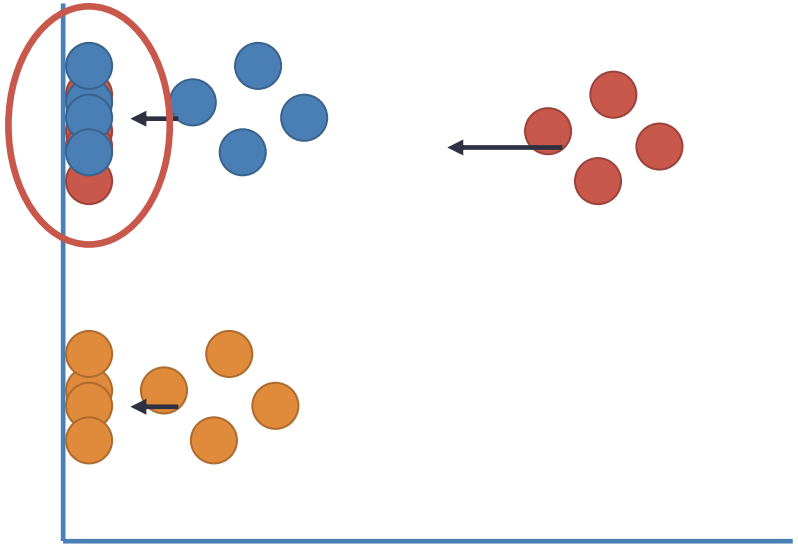
…into a flat, 1-D plot on a number line.



Why not simply project onto an axis?

If we just projected the data onto one of the axes, we would get a mess that does not preserve the original clustering.

Projecting onto the **y-axis**: instead of two distinct clusters, the blue and red points end up mixed together.



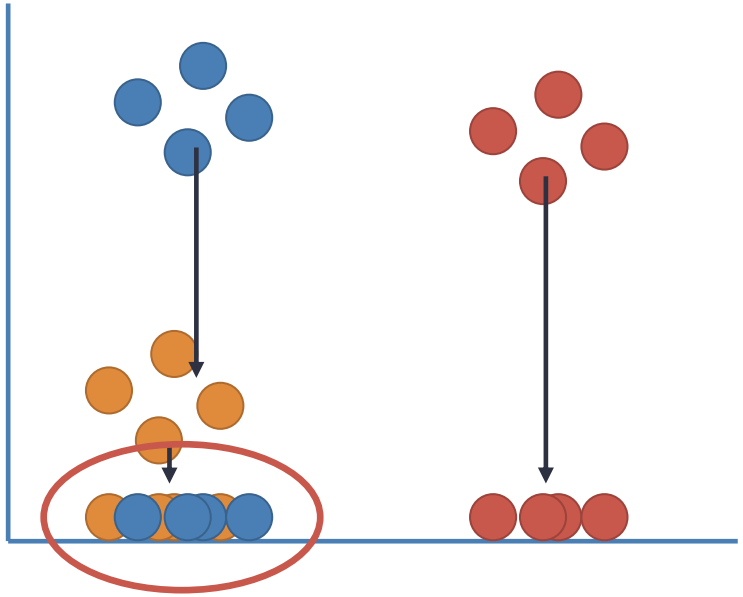
Why not simply project onto an axis?

If we just projected the data onto one of the axes, we would get a mess that does not preserve the original clustering.

Projecting onto the **y-axis**: instead of two distinct clusters, the blue and red points end up mixed together.

Projecting onto the **x-axis** has the same problem, this time with the orange and blue points.

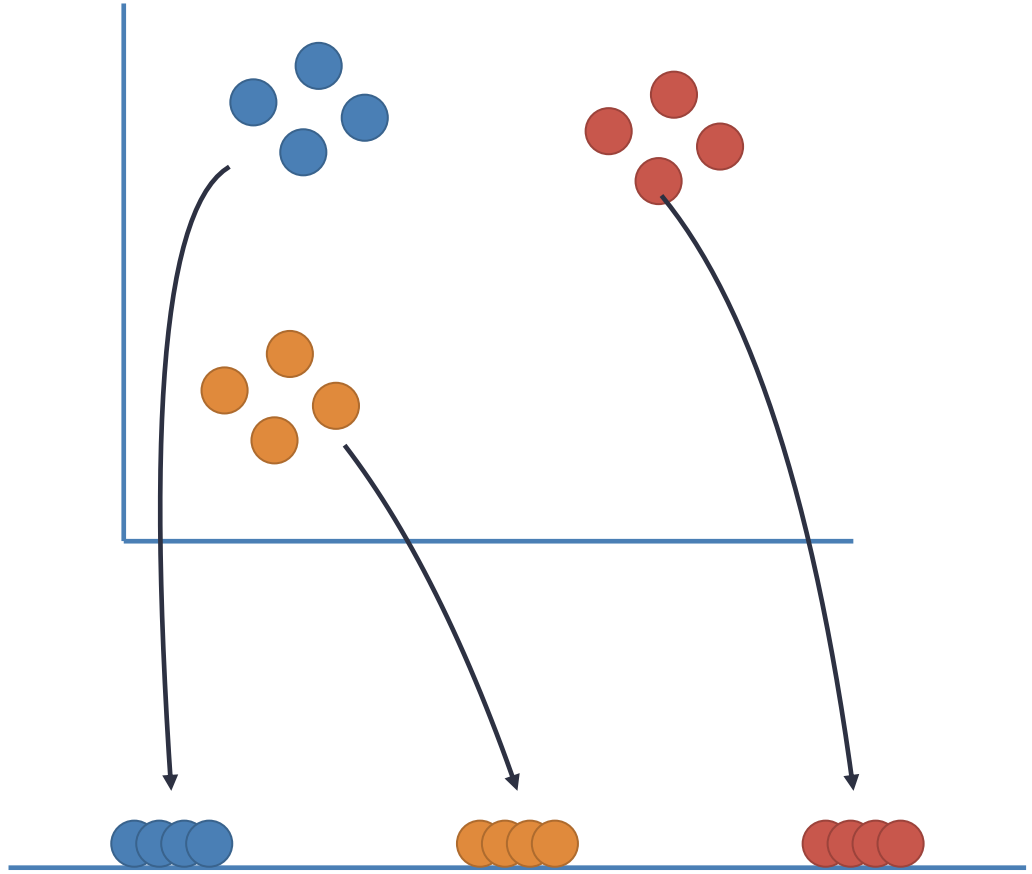
PCA would choose the **best line** rather than an axis, and in this simple case it would separate all three clusters. But real data, with many clusters or curved structure, defeat **any** flat projection.



What t-SNE does

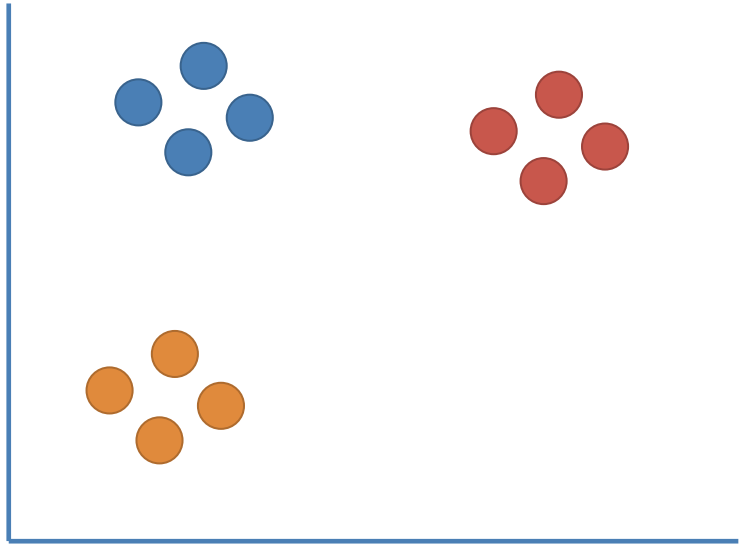
t-SNE finds a way to project data into a **low-dimensional** space (here, the 1-D number line) so that the clustering in the **high-dimensional** space (here, the 2-D scatter plot) is preserved.

In practice the original space has many dimensions (64 for the digits) and the target, a dimension **you choose**, is usually 2-D. The idea is exactly the same.



The starting point: a random placement

We start with the original scatter plot…

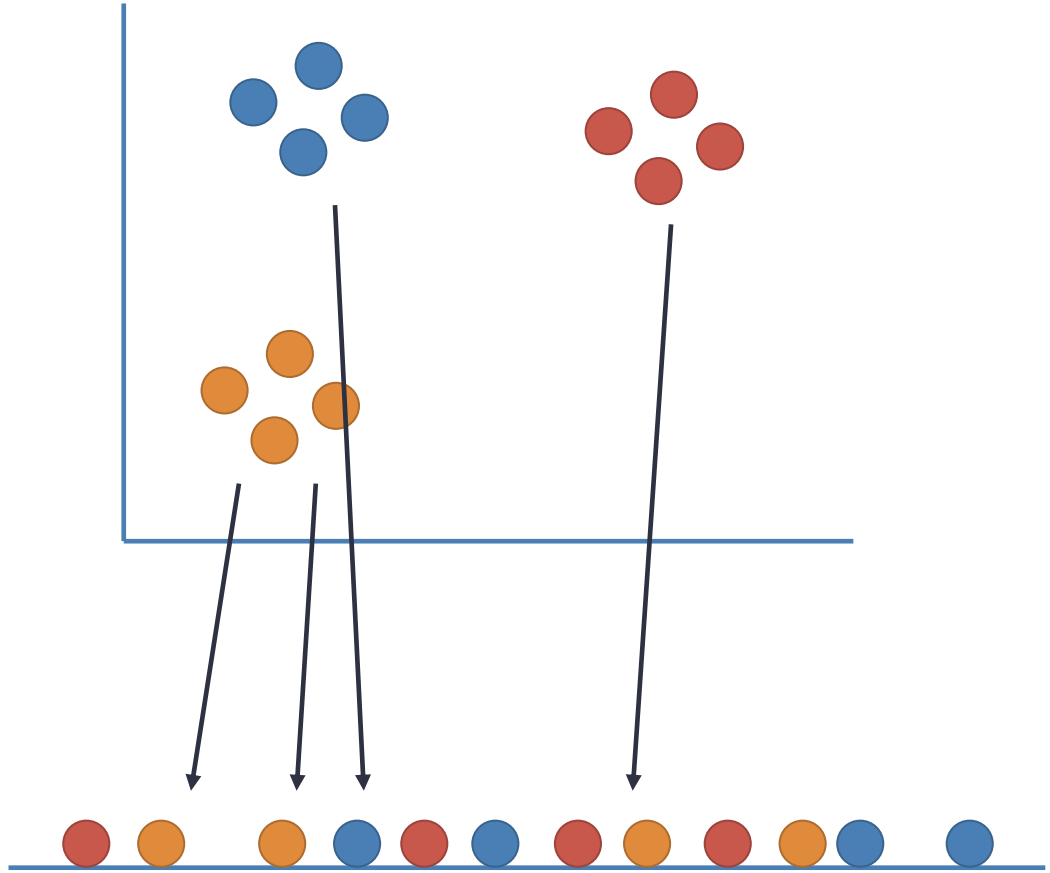


The starting point: a random placement

We start with the original scatter plot…

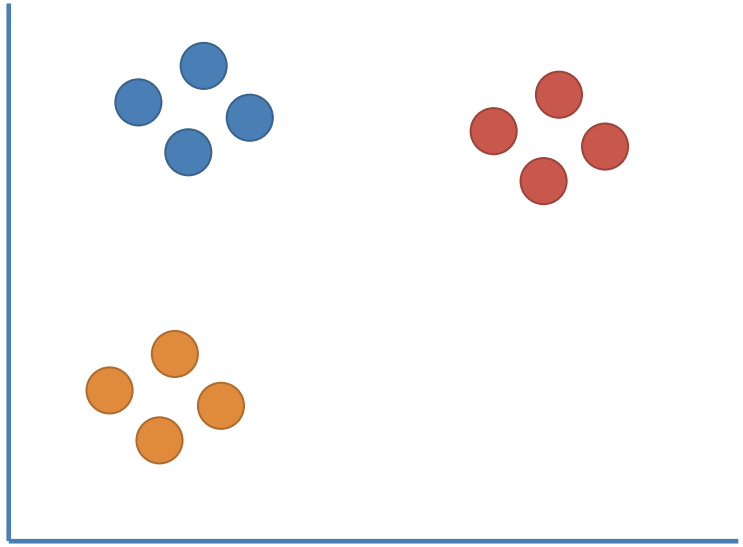
…then we put the points on the number line in a **random order**.

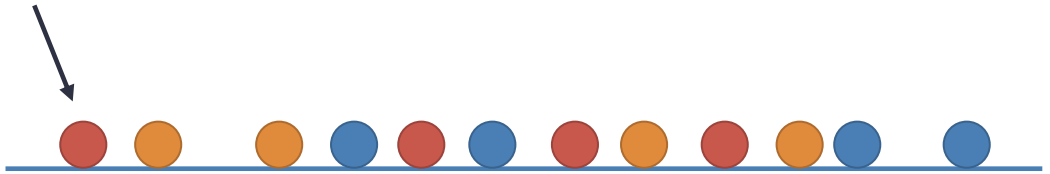
From here, t-SNE moves the points, a little at a time, until the clusters reappear. The random start depends on a **random seed, which you choose**.



Attraction and repulsion

Consider the first point on the number line, which is **red**.

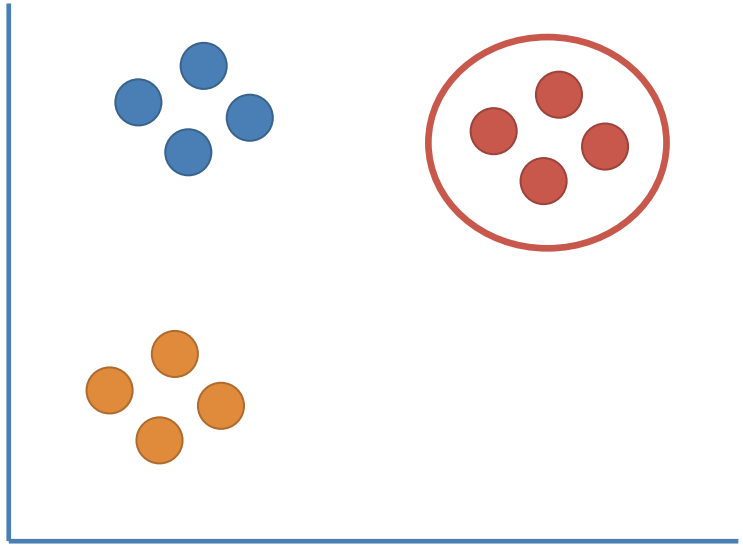




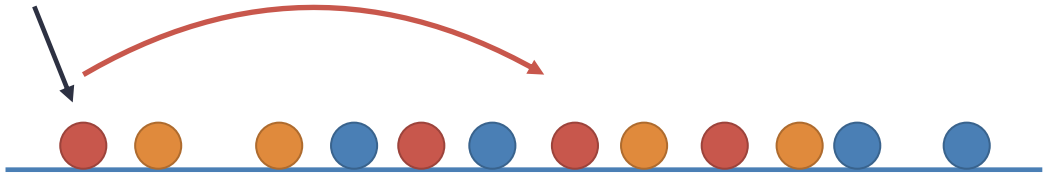
Attraction and repulsion

Consider the first point on the number line, which is **red**.

It belongs to the red cluster in the scatter plot, so it is **attracted** towards the other red points.



attraction

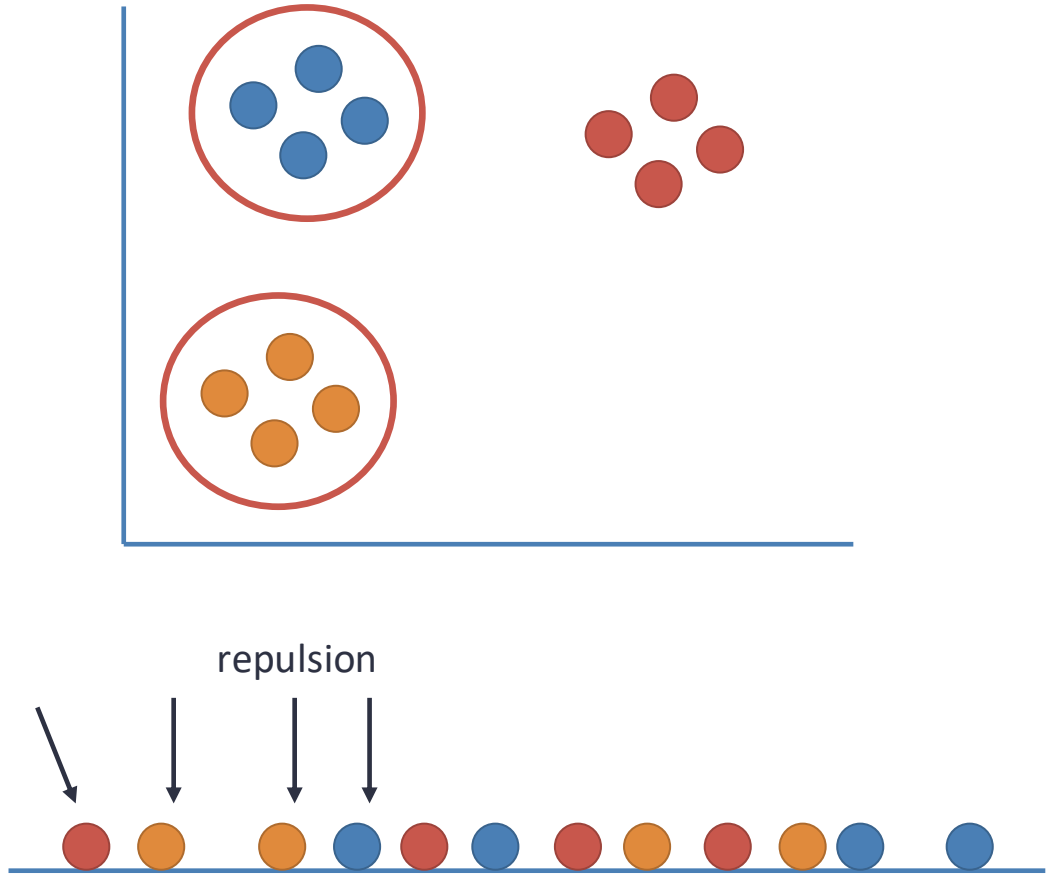


Attraction and repulsion

Consider the first point on the number line, which is **red**.

It belongs to the red cluster in the scatter plot, so it is **attracted** towards the other red points.

At the same time, its neighbours on the number line belong to clusters that are far away in the scatter plot, so they **repel** it.



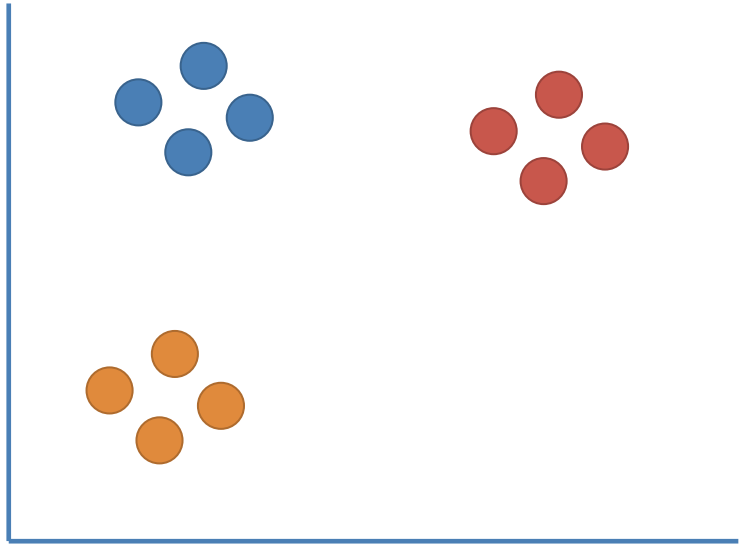
Attraction and repulsion

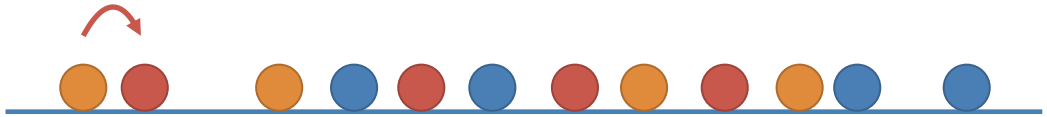
Consider the first point on the number line, which is **red**.

It belongs to the red cluster in the scatter plot, so it is **attracted** towards the other red points.

At the same time, its neighbours on the number line belong to clusters that are far away in the scatter plot, so they **repel** it.

So the red point moves a little to the right, closer to the other red points. Every point is moved in the same way.





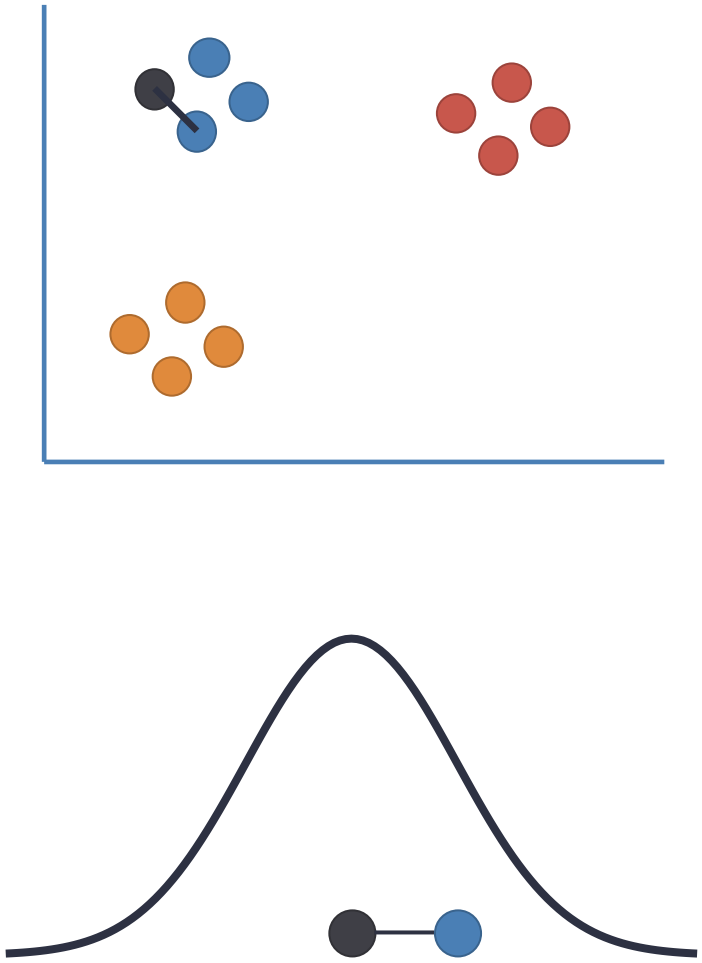
Similarity in the original data

Turning distances in the scatter plot into similarity scores

Measuring similarity in the scatter plot

First, pick a point of interest (black) and measure the distance between it and another point.

Then plot that distance on a **normal curve** centred on the point of interest.

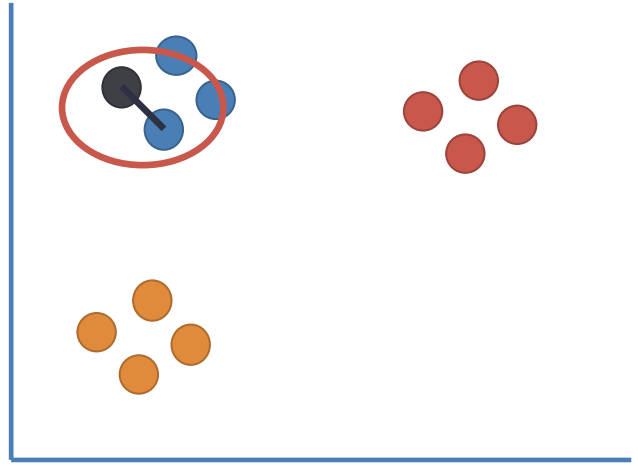


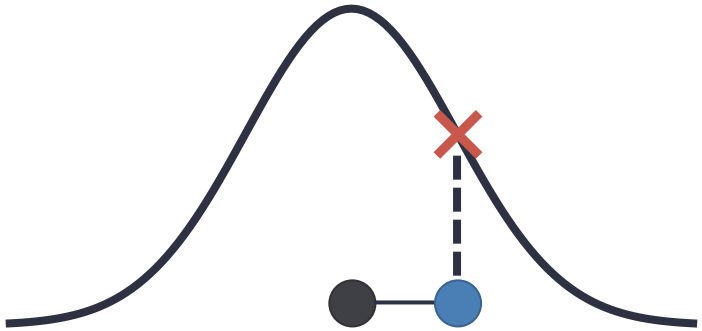
Measuring similarity in the scatter plot

First, pick a point of interest (black) and measure the distance between it and another point.

Then plot that distance on a **normal curve** centred on the point of interest.

The height of the curve at that distance is the **unscaled similarity** for this pair of points.

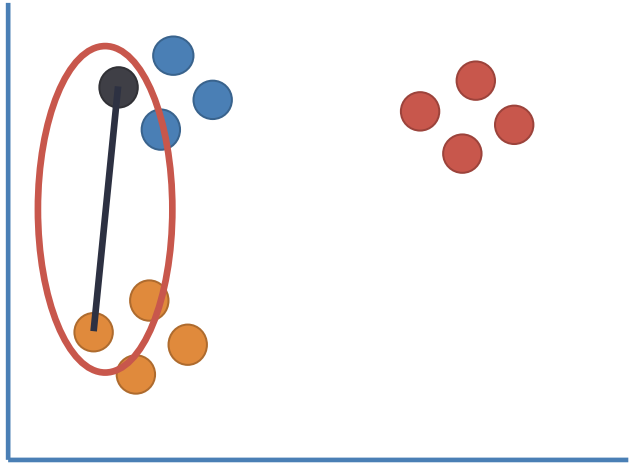


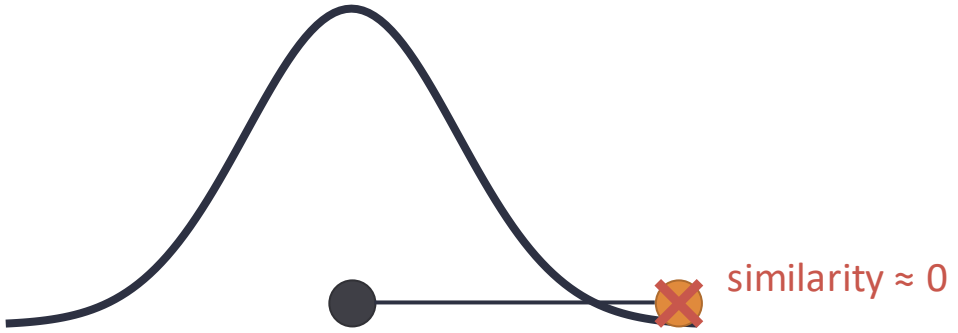


Close points and distant points

Using a normal curve means that **distant** points have very low similarity values…

…and **close** points have high similarity values.

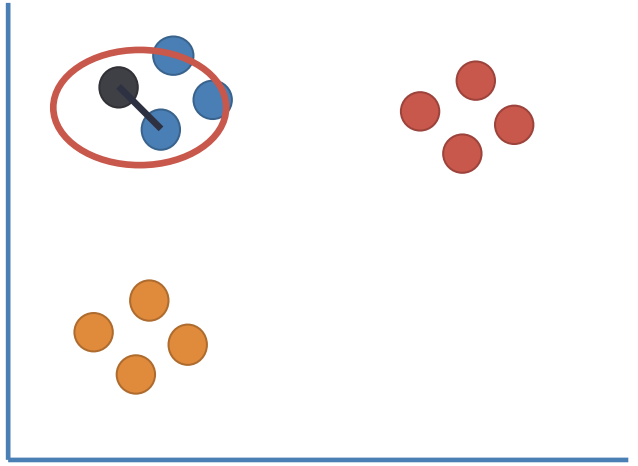


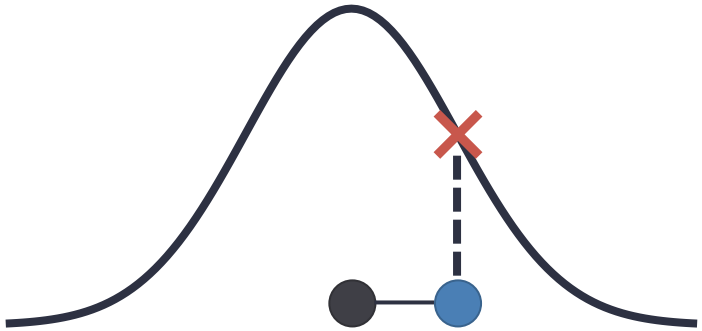


Close points and distant points

Using a normal curve means that **distant** points have very low similarity values…

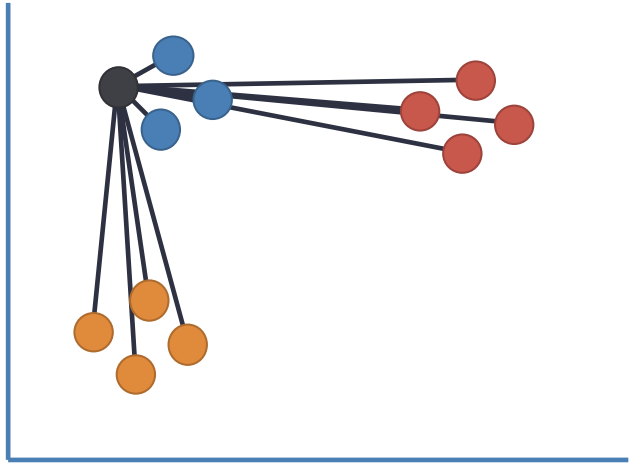
…and **close** points have high similarity values.





Similarity to every other point

Ultimately, we measure the distances between the point of interest and **all** of the other points…

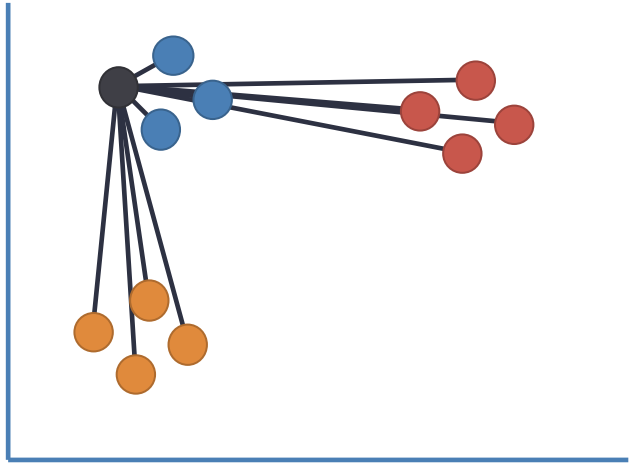


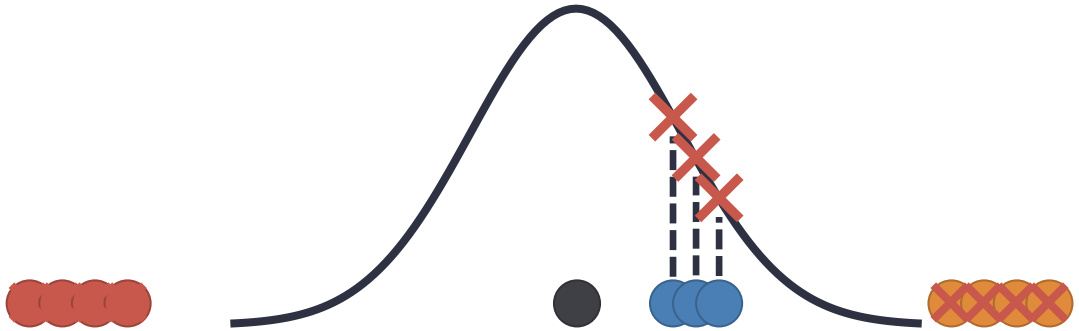
Similarity to every other point

Ultimately, we measure the distances between the point of interest and **all** of the other points…

…plot them on the normal curve…

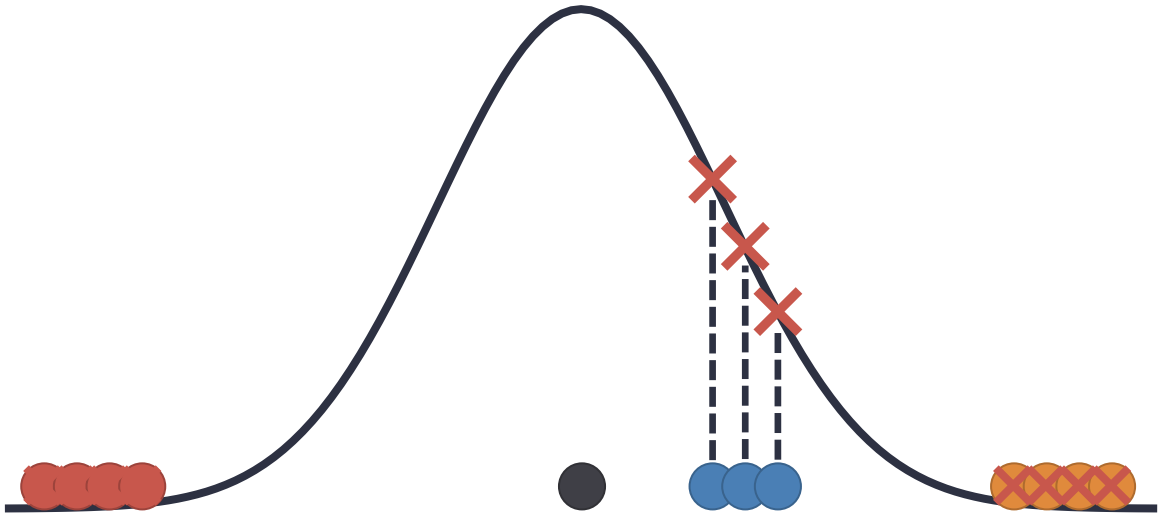
…and read off the height of the curve at each one. These are the **unscaled similarity scores** with respect to the point of interest.





Scaling the similarity scores

The next step is to **scale** the unscaled similarities so that they **add up to 1**.

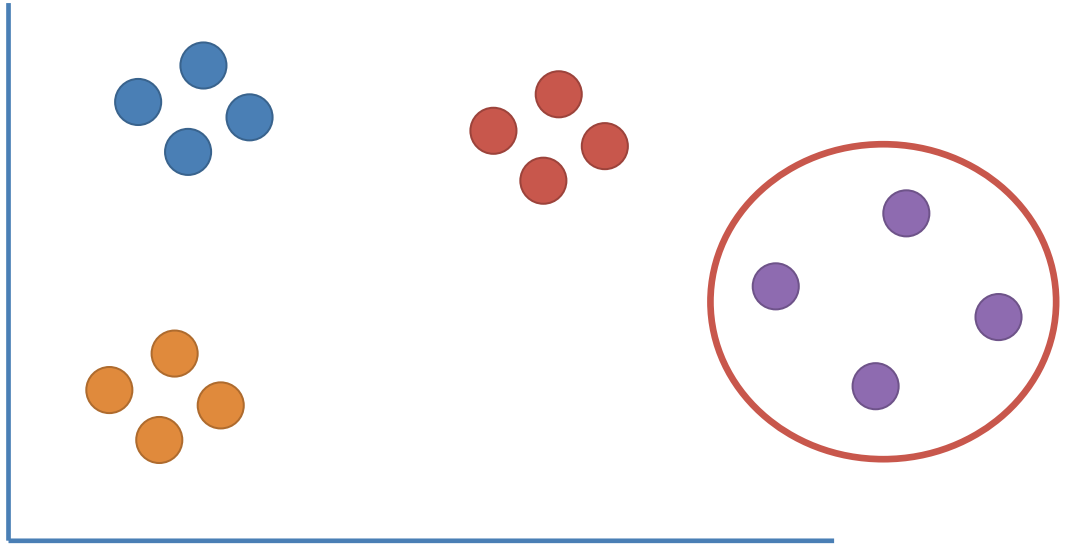


Scaling the similarity scores

The next step is to **scale** the unscaled similarities so that they **add up to 1**.

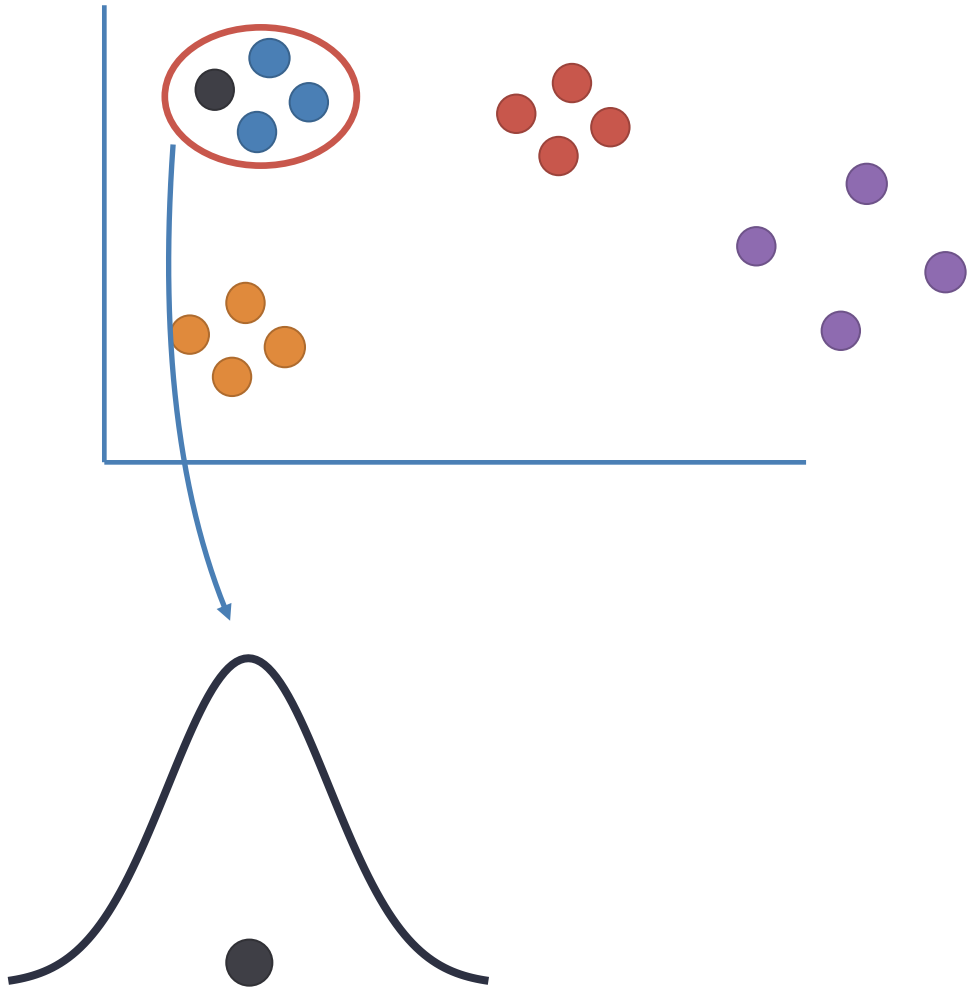
Why? It has to do with something we have not mentioned yet: clusters can have different **densities**.

To illustrate the idea, we add a **purple** cluster that is half as dense as the others: its points are twice as far apart.



The width of the curve depends on density

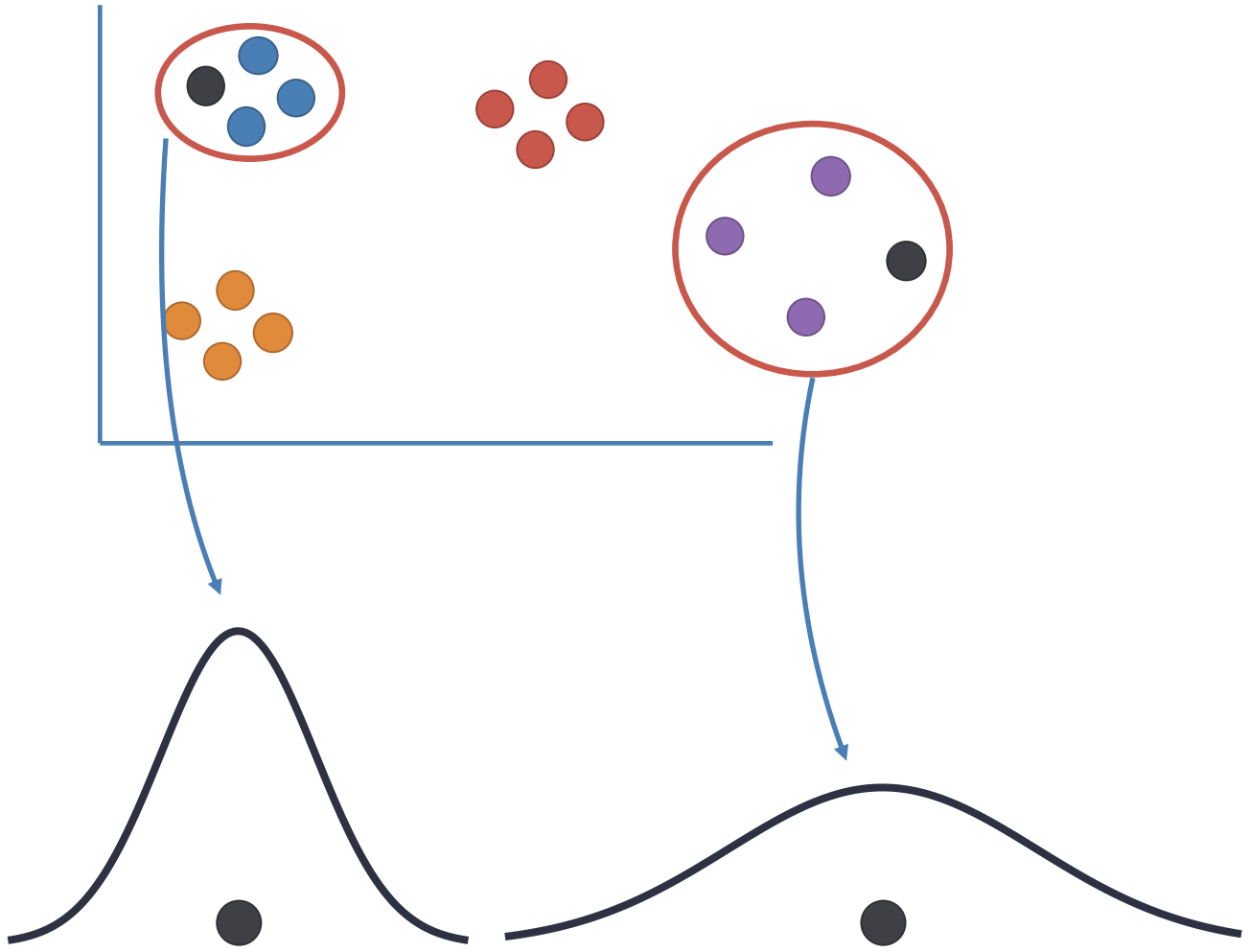
The **width** of the normal curve depends on the density of data near the point of interest. **t-SNE sets it automatically**, for each point.



The width of the curve depends on density

The **width** of the normal curve depends on the density of data near the point of interest. **t-SNE sets it automatically**, for each point.

Less dense regions get **wider** curves: the purple curve is twice as wide as the blue one, and only half as tall.

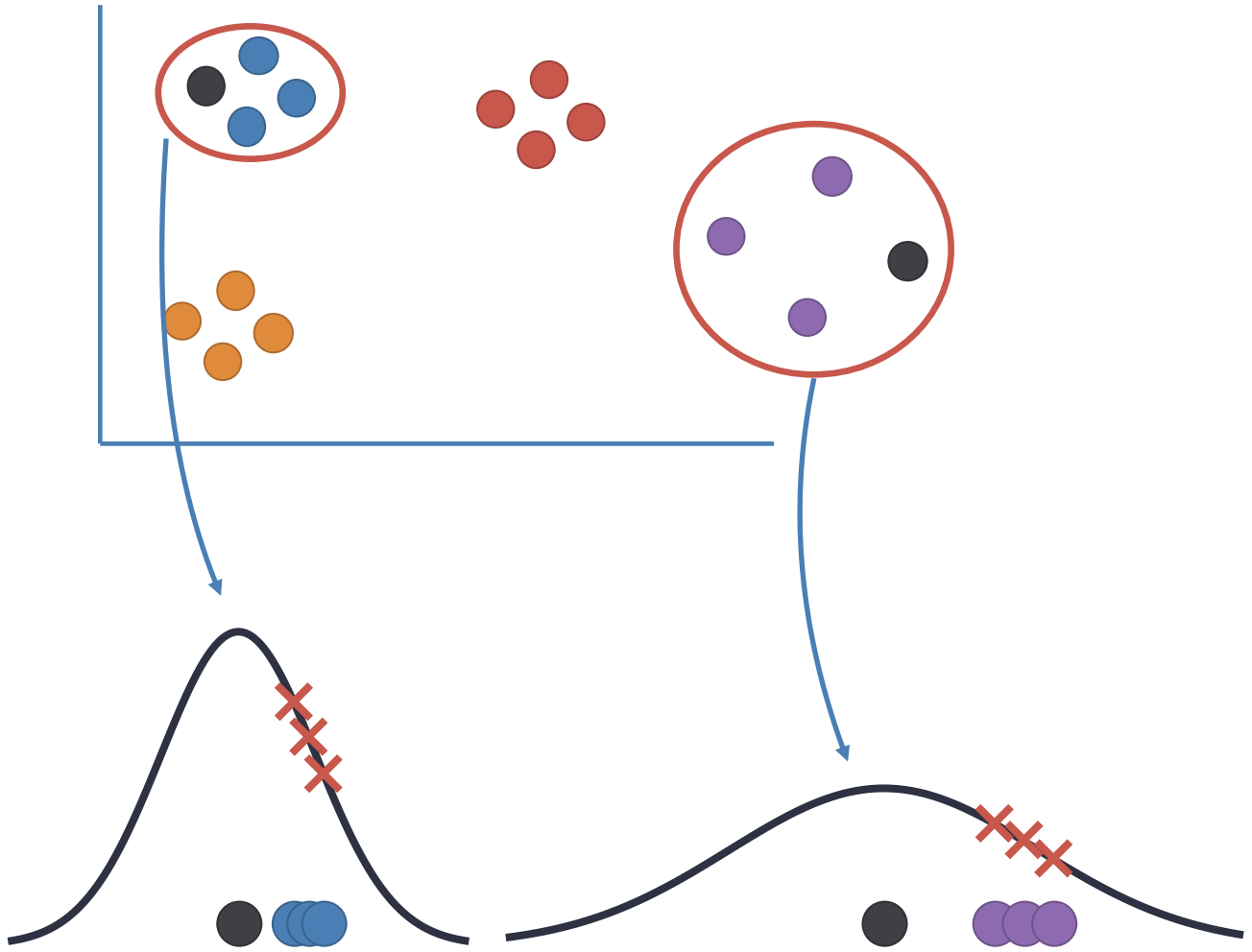


The width of the curve depends on density

The **width** of the normal curve depends on the density of data near the point of interest. **t-SNE sets it automatically**, for each point.

Less dense regions get **wider** curves: the purple curve is twice as wide as the blue one, and only half as tall.

The purple points are twice as far away, but the purple curve is also twice as wide, so after scaling the similarity scores are **the same** for both clusters.

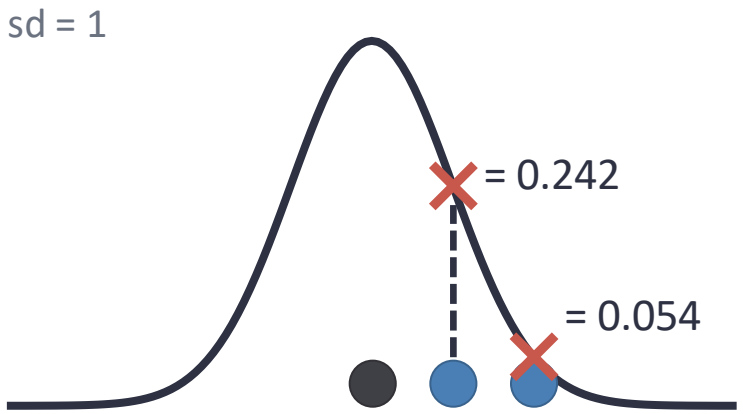


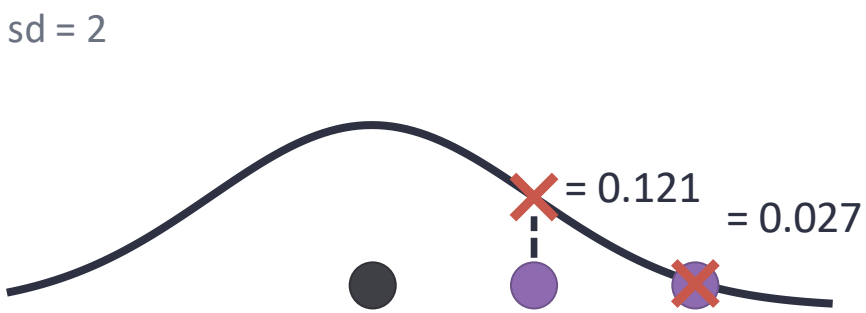
A worked example

The top curve has standard deviation **1**. Its two neighbours are at distances 1 and 2.

The bottom curve has standard deviation **2**, and its neighbours are twice as far away: at distances 2 and 4.

The heights of the curves at those distances are the **unscaled** similarity values.





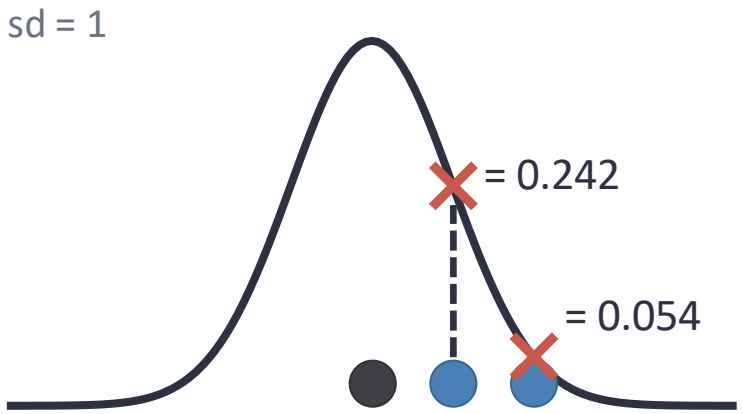
A worked example

To scale the similarity scores so that they sum to 1, divide each score by the sum of all the scores:

$$\frac{\text{Score}}{\text{Sum of all scores}} = \text{Scaled score}$$

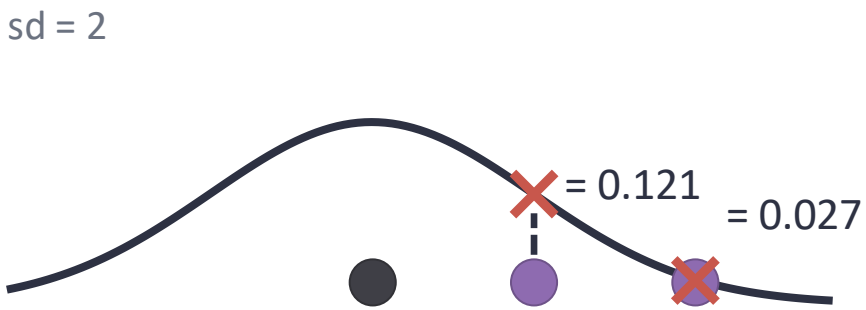
The scaled scores for the two curves are **the same**, even though one cluster is half as dense as the other.

This is exactly why we scale: it makes the similarity scores independent of the density of each cluster.



$$\frac{0.242}{0.242 + 0.054} = 0.82$$

$$\frac{0.054}{0.242 + 0.054} = 0.18$$



$$\frac{0.121}{0.121 + 0.027} = 0.82$$

$$\frac{0.027}{0.121 + 0.027} = 0.18$$

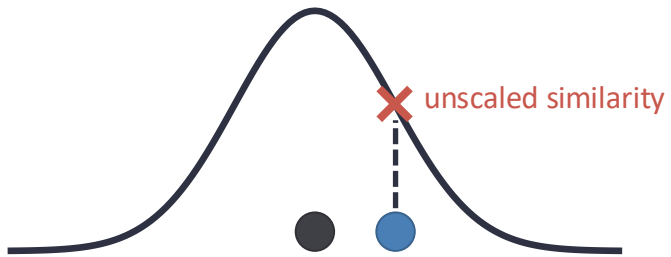
In formulas: similarity in the original data

Everything since the normal curve fits in one formula: the **height of the curve**, divided by **the sum of all the heights**.

We call the result $p_{j|i}$: the similarity of point j to the point of interest i.

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

The picture we are formalising:



where

$\|x_i - x_j\|$ the distance between the point of interest x_i and another point x_j

σ_i the width of the normal curve around x_i, **set automatically by t-SNE** (next slides)

$\exp(\ldots)$ the height of the curve: the unscaled similarity (the constant in front cancels when we scale)

$\sum_{k \neq i}$ the sum of all the scores, so that the $p_{j|i}$ add up to 1

Reading the formula: similarity in the original data

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

Step by step

1. Measure the distance $\|x_i - x_j\|$ to the point of interest.
2. Square it and divide by $2\sigma_i^2$: distance measured in units of the curve's width.
3. Take exp(− …): 1 at distance 0, falling quickly towards 0. This is the normal curve's height.
4. Divide by the sum over all other points, so the values add up to 1.
5. Result: a distribution over the possible neighbours of x_i.

Why the exponential?

It is the shape of the **normal curve**. $\exp(-d^2/2\sigma^2)$ equals 1 at d = 0 and decays very fast, so near points get large values and distant points almost nothing.

The constant $1/(\sigma\sqrt{2\pi})$ is dropped because it cancels in the division.

Is it really a probability?

Mathematically, yes: each $p_{j|i} \geq 0$ and they add up to 1.

We can interprete as: if x_i picked one neighbour at random, in proportion to its normal curve, $p_{j|i}$ is the chance of picking x_j.

Notice: it is conditional: $p_{j|i}$ need not equal $p_{i|j}$.

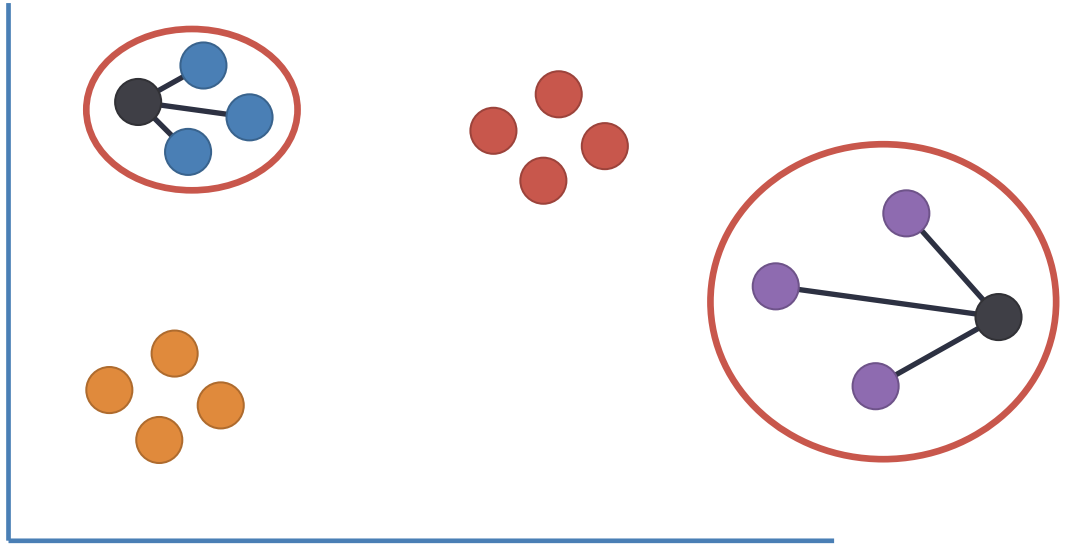
t-SNE: the Perplexity parameter

The reality is a little more complicated, but only slightly.

t-SNE has a **perplexity** parameter, which sets the expected density: roughly, how many close neighbours each point should have.

You choose it; t-SNE then **adjusts each σ_i automatically** to match it.

This comes into play, but these two clusters still end up more "similar" than you might expect from the scatter plot.



In formulas: perplexity

The **perplexity** is the one number **you choose** (typically 5 to 50; scikit-learn uses 30).

t-SNE then sets each width σ_i automatically, so that every point has that perplexity: roughly, the same **number of close neighbours**.

Dense region: narrow curve. Sparse region: wide curve.



$$\mathrm{Perp}(P_i) = 2^{H(P_i)}, \qquad H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

where

$H(P_i)$ the entropy: how spread out point i's similarities are

If point i shares its similarity equally among k neighbours, its perplexity is exactly k.

In a dense region a narrow curve already reaches k neighbours, so σ_i is small; in a sparse region σ_i is large.

A tiny example: perplexity for one width

One point i has three neighbours, A, B and C, at distances **1, 2 and 3**.

You choose a perplexity of **2**: i should behave as if it had about two neighbours.

t-SNE starts by trying a width, say $\sigma_i = 1$, and works out what perplexity that width gives.

Result: about **1.7** effective neighbours. A dominates, B counts a little, C hardly at all. That is fewer than the 2 we asked for.

Step 1. Height of the normal curve, then scale so the values add up to 1

Neighbour	Distance	Height $\exp(-d^2/2\sigma^2)$	Scaled: $p_{j\|i}$
A	1	0.607	0.81
B	2	0.135	0.18
C	3	0.011	0.01
sum		0.753	**1.00**

Step 2. Entropy: how spread out the three values are

$H = -(0.81 \log_2 0.81 + 0.18 \log_2 0.18 + 0.01 \log_2 0.01) \approx$ **0.79 bits**

Step 3. Perplexity: turn the entropy back into a number of neighbours

$\text{Perp} = 2^H = 2^{0.79} \approx$ **1.7** (target: 2, so the curve is too narrow)

Reading the formula: why entropy?

$$\mathrm{Perp}(P_i) = 2^{H(P_i)}, \quad H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

Step by step

1. We want every point to have roughly the same number of neighbours, whatever the local density.
2. So we need to measure "how many neighbours" from the similarities $p_{j|i}$.
3. Entropy measures how spread out those similarities are, in bits.
4. 2^H turns bits back into a count: an effective number of neighbours.
5. t-SNE adjusts σ_i until that count equals the perplexity **you chose**.

Why not simply count the k nearest neighbours?

A hard count ignores the actual distances and jumps abruptly when a point enters or leaves the list. The $p_{j|i}$ change smoothly with σ_i, so we want an equally smooth "number of neighbours", computed from them.

What does entropy measure?

How uncertain the choice of neighbour is.
All the similarity on one neighbour: no uncertainty, H = 0.
Spread equally over k neighbours: the most uncertainty possible for k, $H = \log_2 k$ bits.

Why 2 to the power H?

It undoes the logarithm, turning bits back into a count: equal over k neighbours gives $2^{\log_2 k} = k$.

$$H = 0.5(1) + 0.25(2) + 0.25(2) = 1.5 \text{ bits}, \quad \mathrm{Perp} = 2^{1.5} \approx 2.8$$

How does this set σ_i?

A wider curve spreads the similarity over more points, so the entropy, and the perplexity, grow with σ_i. A binary search finds the σ_i that hits the chosen value.
Small perplexity: a very local view;
large perplexity: each point takes more of the data into account.

A tiny example: adjusting σ_i to match the perplexity

A **wider** curve shares the similarity more evenly, so the entropy and the perplexity go **up**; a narrower curve gives everything to the nearest point.

Because perplexity always grows with σ_i, t-SNE finds the right width with a simple **binary search** (not gradient descent): too low, widen; too high, narrow; each step halves the gap.

Here it stops at $\sigma_i \approx 1.16$, after about a dozen steps. **t-SNE does this automatically**, once for every point, before any point moves.

σ_i	Scaled similarities (A, B, C)	Entropy H	Perplexity 2^H
0.5	1.00, 0.00, 0.00	0.03	1.0
1.0	0.81, 0.18, 0.01	0.79	1.7
1.16	**0.73, 0.24, 0.04**	**1.00**	**2.0 ✓**
1.5	0.59, 0.31, 0.10	1.30	2.5
2.0	0.49, 0.33, 0.18	1.48	2.8

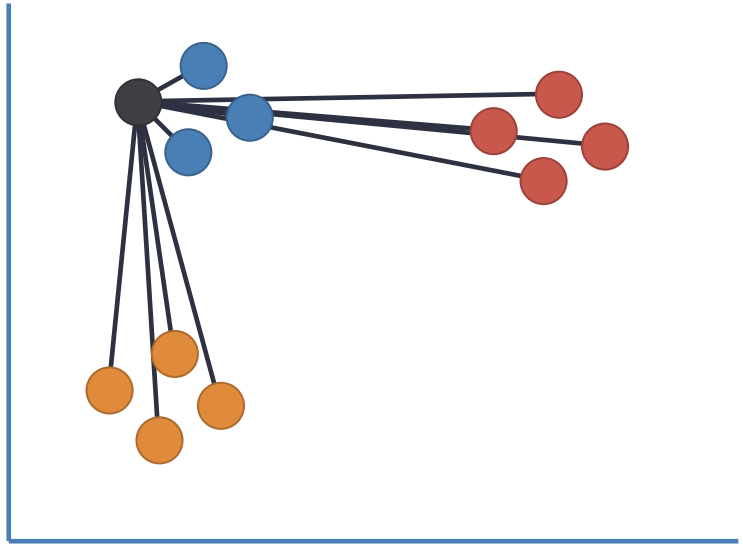
The same target for a point j in a sparser region

Neighbours twice as far away: distances **2, 4 and 6**. The search ends at $\sigma_j \approx 2.31$, twice as wide, and the scaled similarities are **identical**: 0.73, 0.24, 0.04.

Every point ends up with about the same number of effective neighbours, whatever the density.

Repeating for every point

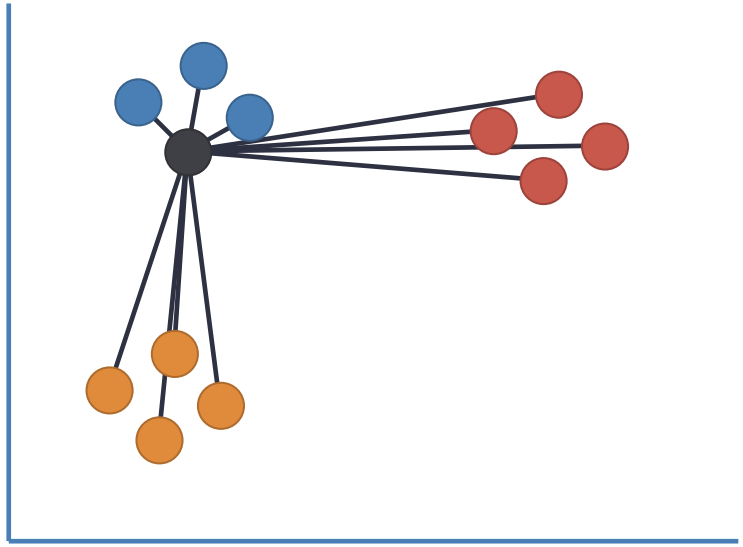
We have now calculated the similarity scores for this point.



Repeating for every point

We have now calculated the similarity scores for this point.

Next, we do the same for this point… and then for **every** point in the data.

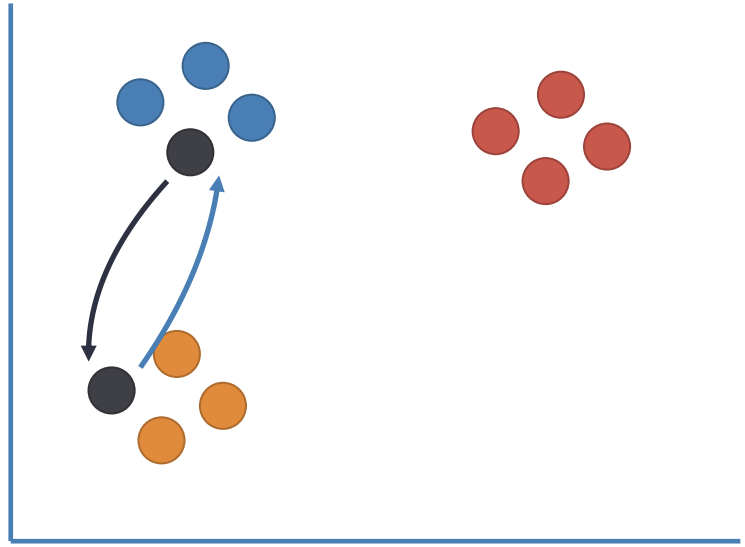


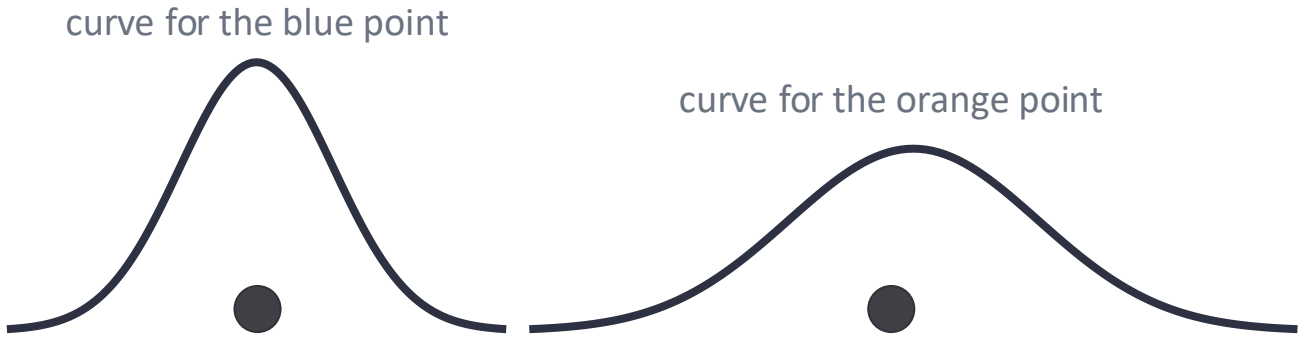
Repeating for every point

We have now calculated the similarity scores for this point.

Next, we do the same for this point… and then for **every** point in the data.

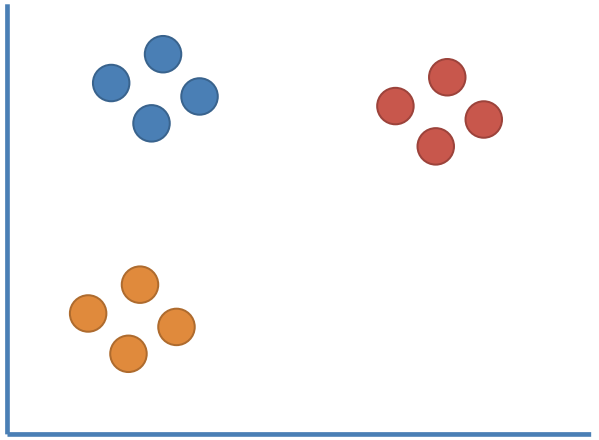
Each point has its own curve width, so the similarity from a blue point to an orange point might not equal the similarity from the orange point to the blue point. t-SNE **averages** the two, so each pair gets a single score.

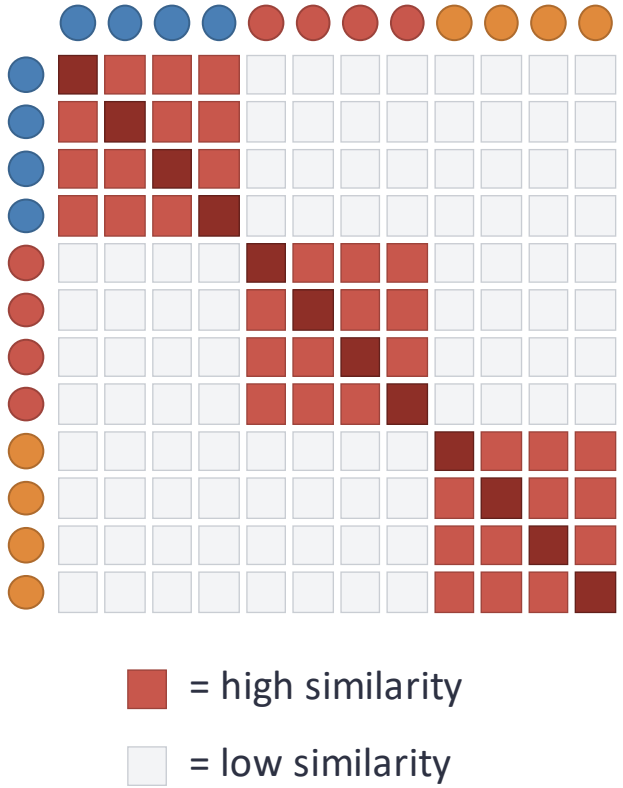




The similarity matrix

Ultimately, we end up with a **matrix** of similarity scores: one row and one column for each point.





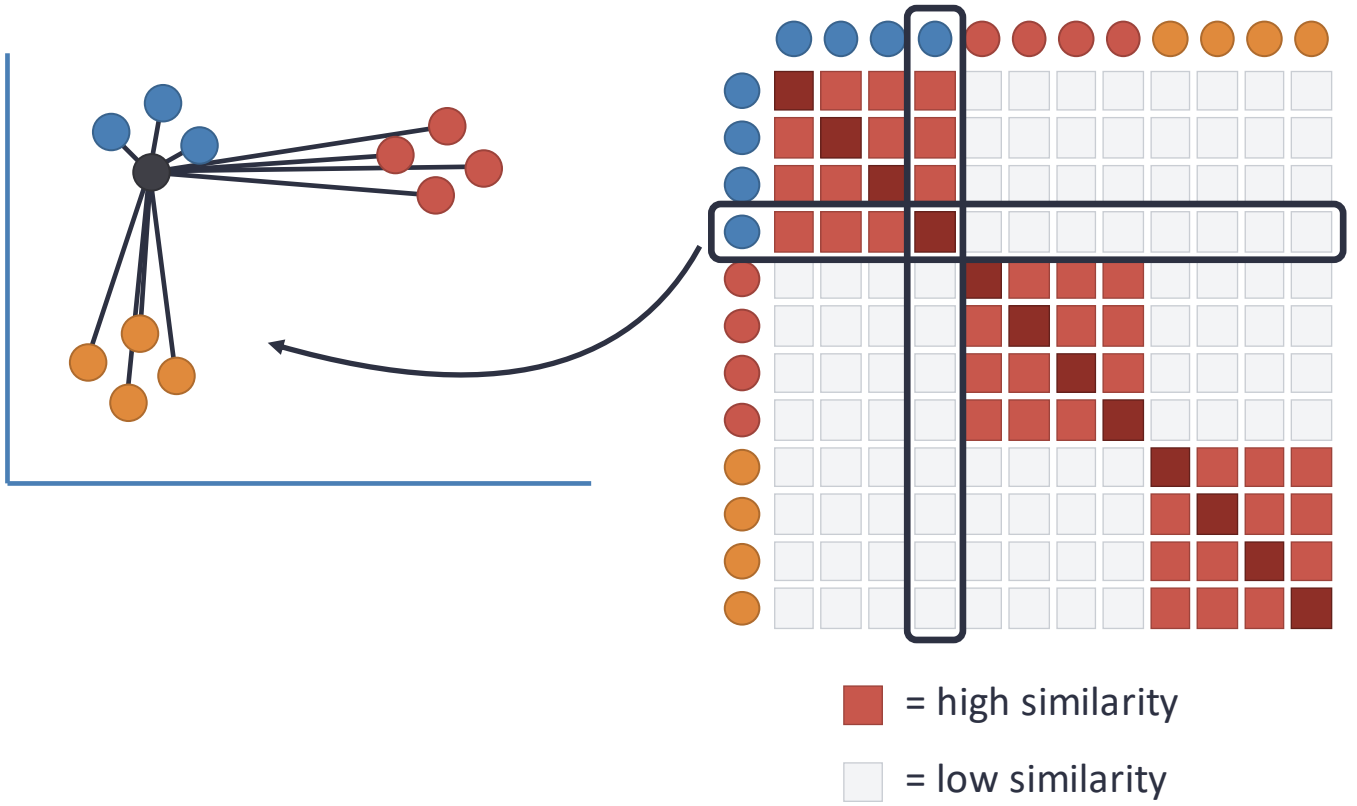
The similarity matrix

Ultimately, we end up with a **matrix** of similarity scores: one row and one column for each point.

Each row holds the scores from one point to all the others. Points in the same cluster have high similarity, so the matrix shows three red blocks.

In formulas, each entry averages the two directions; dividing by 2n makes all the entries add up to 1:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$



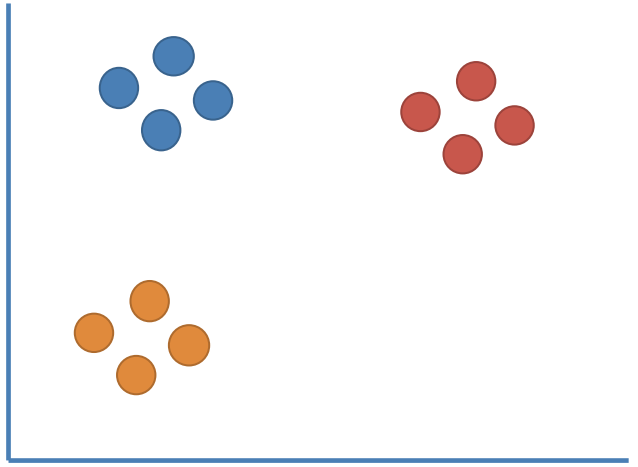
Similarity on the number line

A second matrix, built with a t-distribution

Similarities on the number line

Now we randomly project the data onto the number line…

…and calculate similarity scores for the points on the number line.

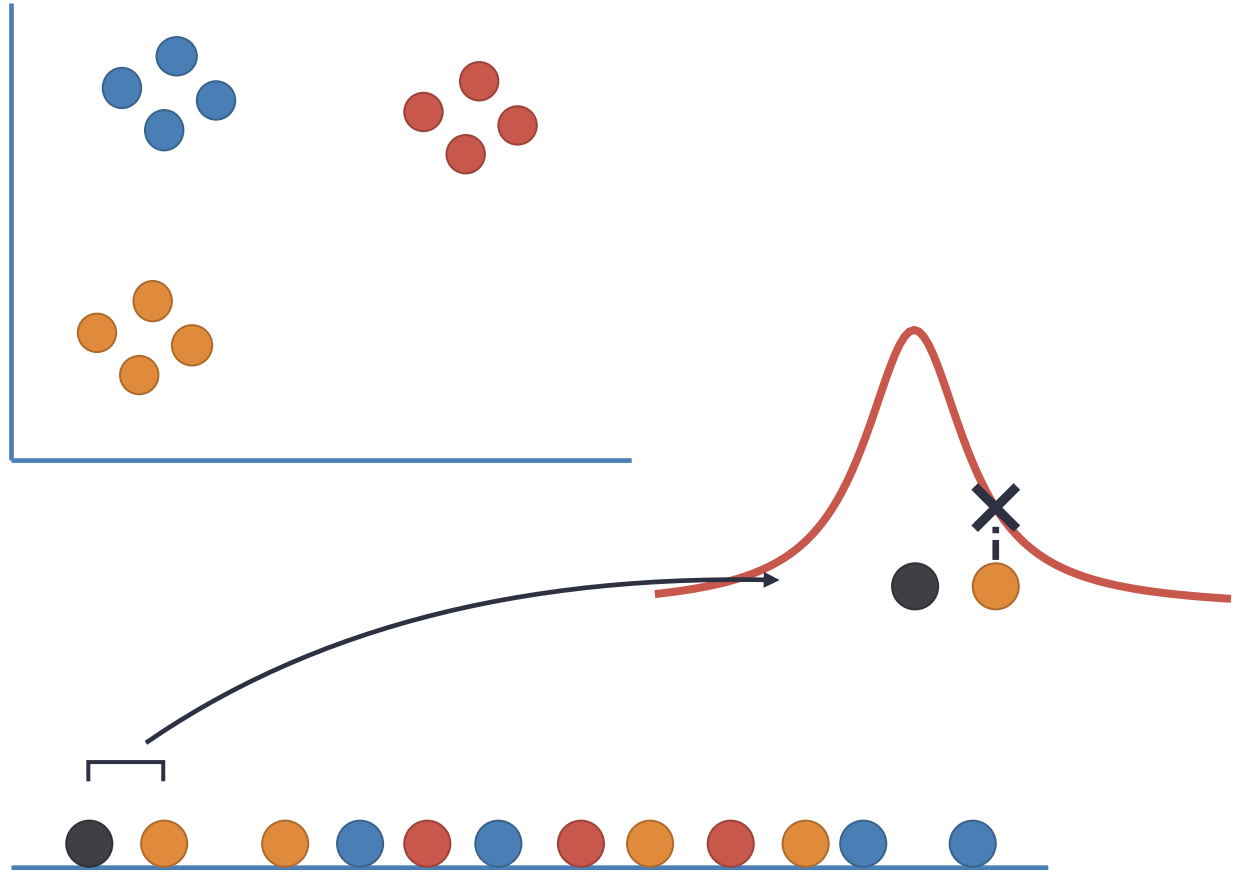




Similarities on the number line

Just like before, that means picking a point, measuring a distance, and reading off the height of a curve.

However, this time we use a **t-distribution** (red) instead of a normal curve.



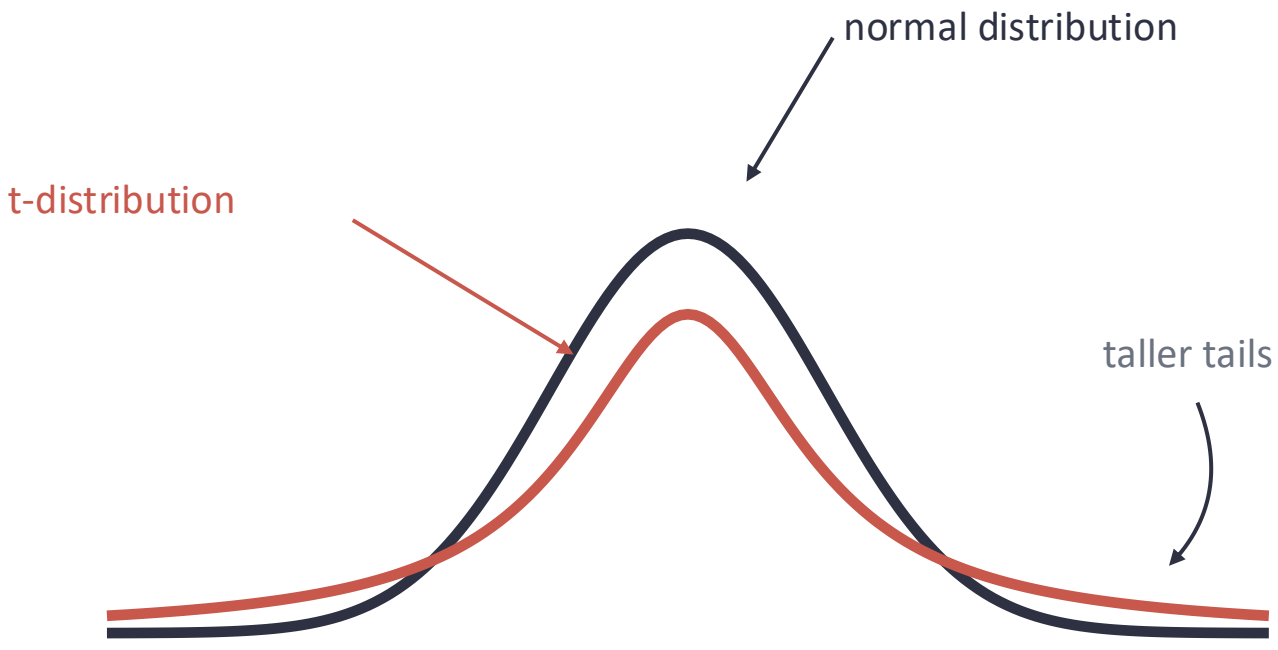
The t-distribution

A t-distribution is a lot like a normal distribution…

…except that the t-distribution is **not as tall** in the middle…

…and its **tails are taller** at the ends.

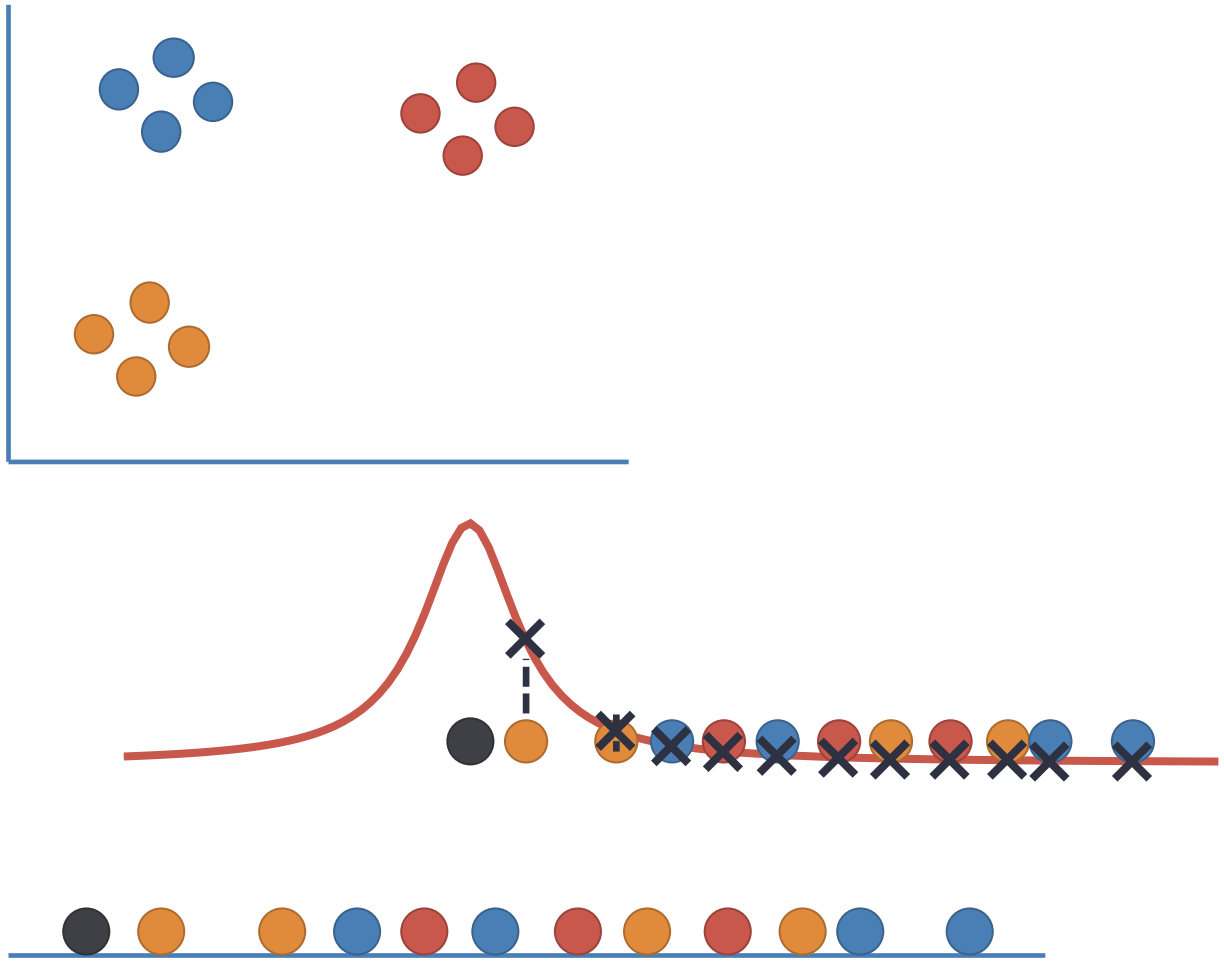
We will see why the taller tails matter once the points start to move.



Unscaled similarities with the t-distribution

Using the t-distribution, we calculate the **unscaled** similarity scores for all the points on the number line…

…and then **scale** them so that they add up to 1, exactly as before.



In formulas: similarity on the number line

The recipe is the same, with two changes:

1. a **t-distribution** instead of a normal curve, and
2. **the same curve** for every point.

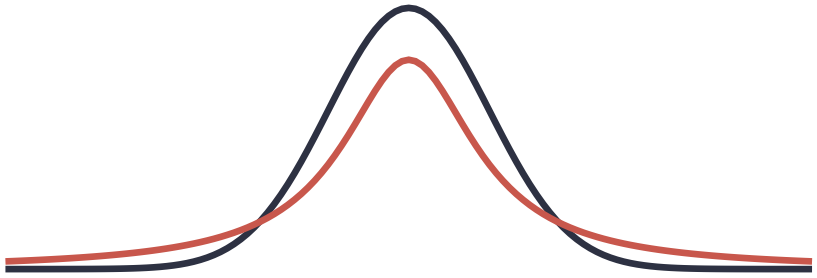
We call the result q_{ij}.

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

where:

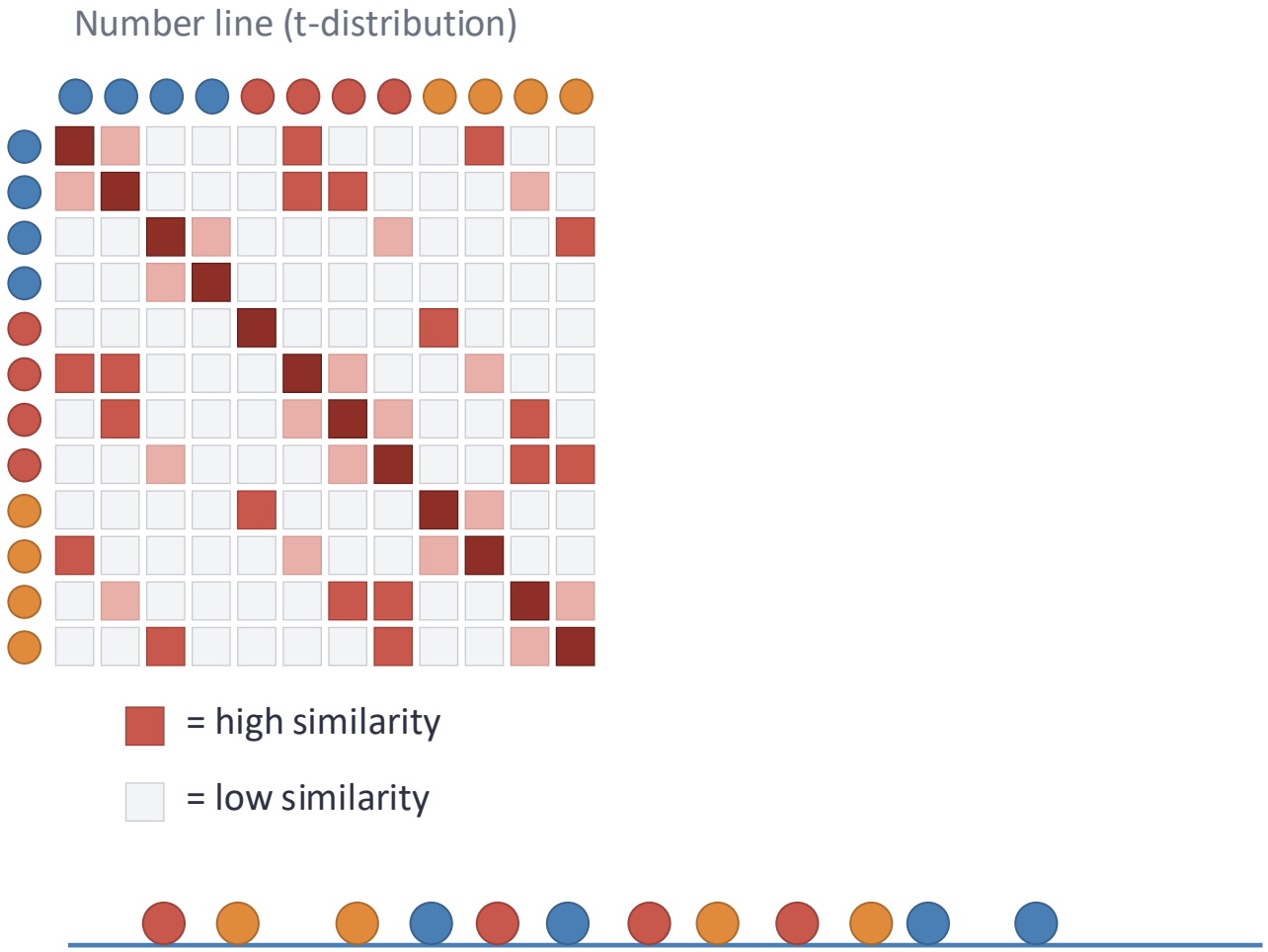
$y_i,\ y_j$	the positions of the two points on the number line
$(1 + d^2)^{-1}$	the height of the t-distribution (one degree of freedom) at distance d; **fixed**: there is no σ to choose
$\sum_{k \neq l}$	the sum over all pairs, so that all the q_{ij} add up to 1

Normal curve (black) and t-distribution (red):



Comparing the two matrices

Like before, we end up with a matrix of similarity scores, but this matrix is "**sparse**"…

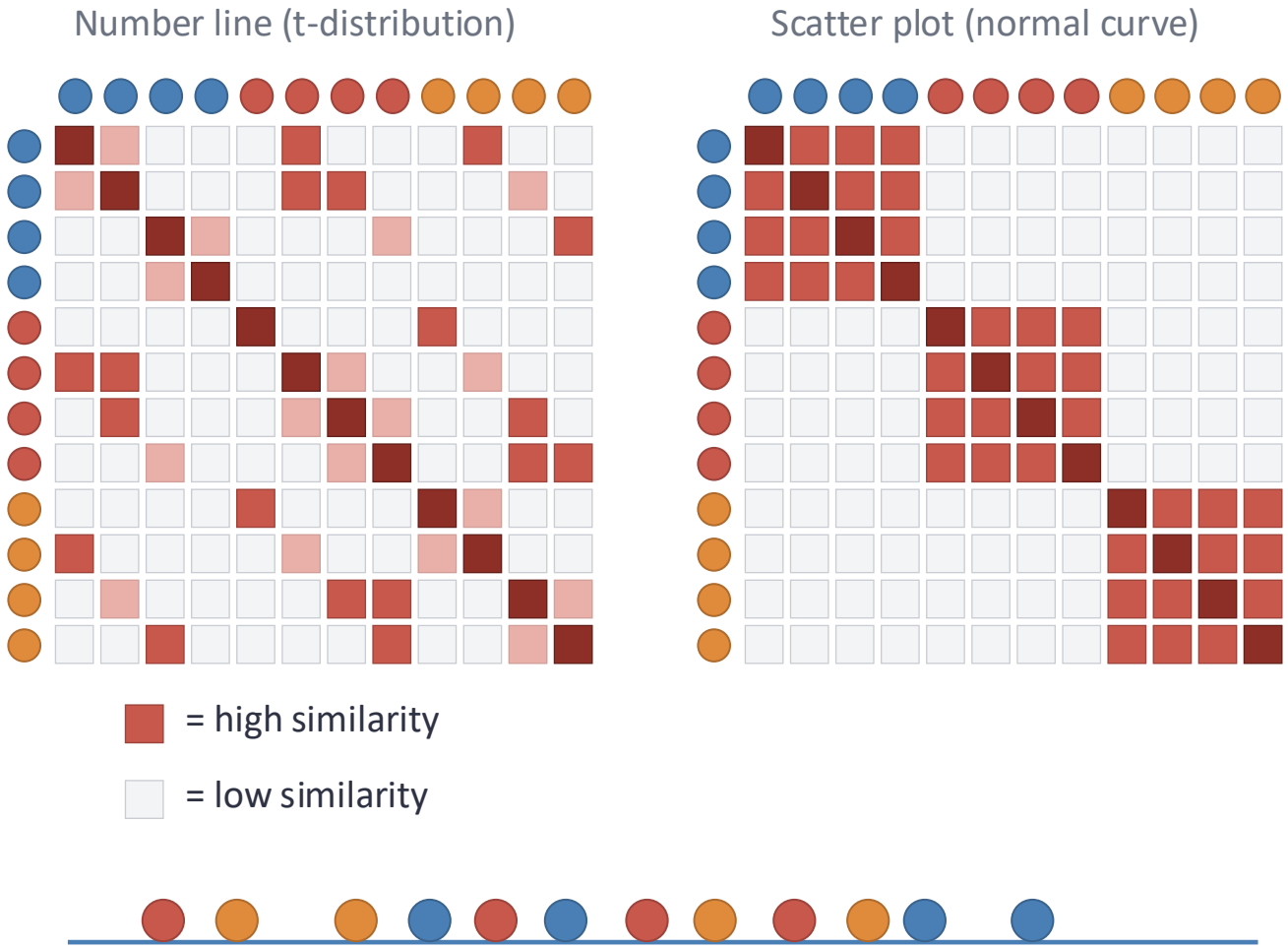


Comparing the two matrices

Like before, we end up with a matrix of similarity scores, but this matrix is "**sparse**"…

…compared with the original matrix from the scatter plot.

The left matrix comes from the current number line; the right one is the target we want to reproduce.

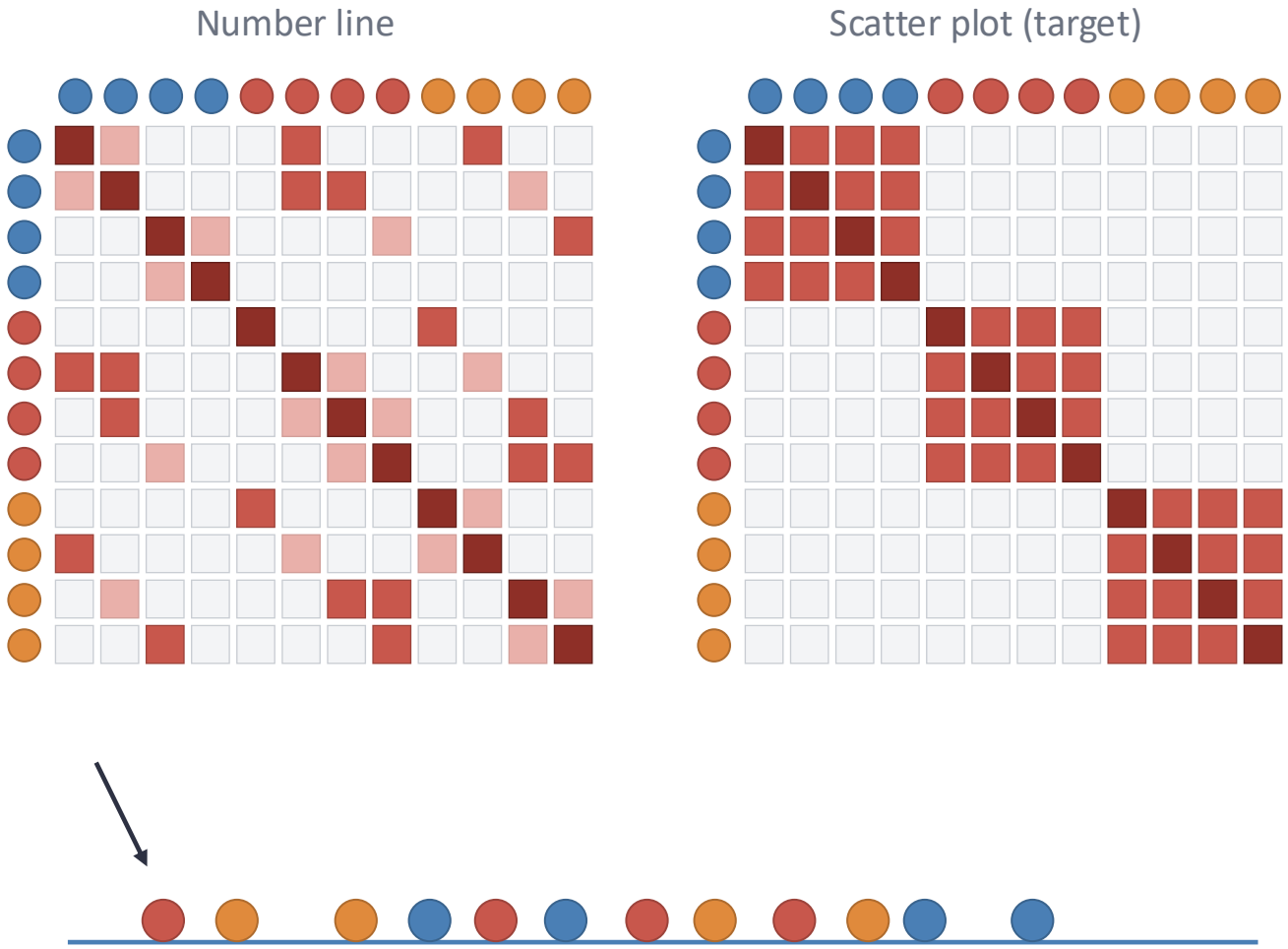


Moving the points

Making the number line agree with the scatter plot

The goal of moving a point

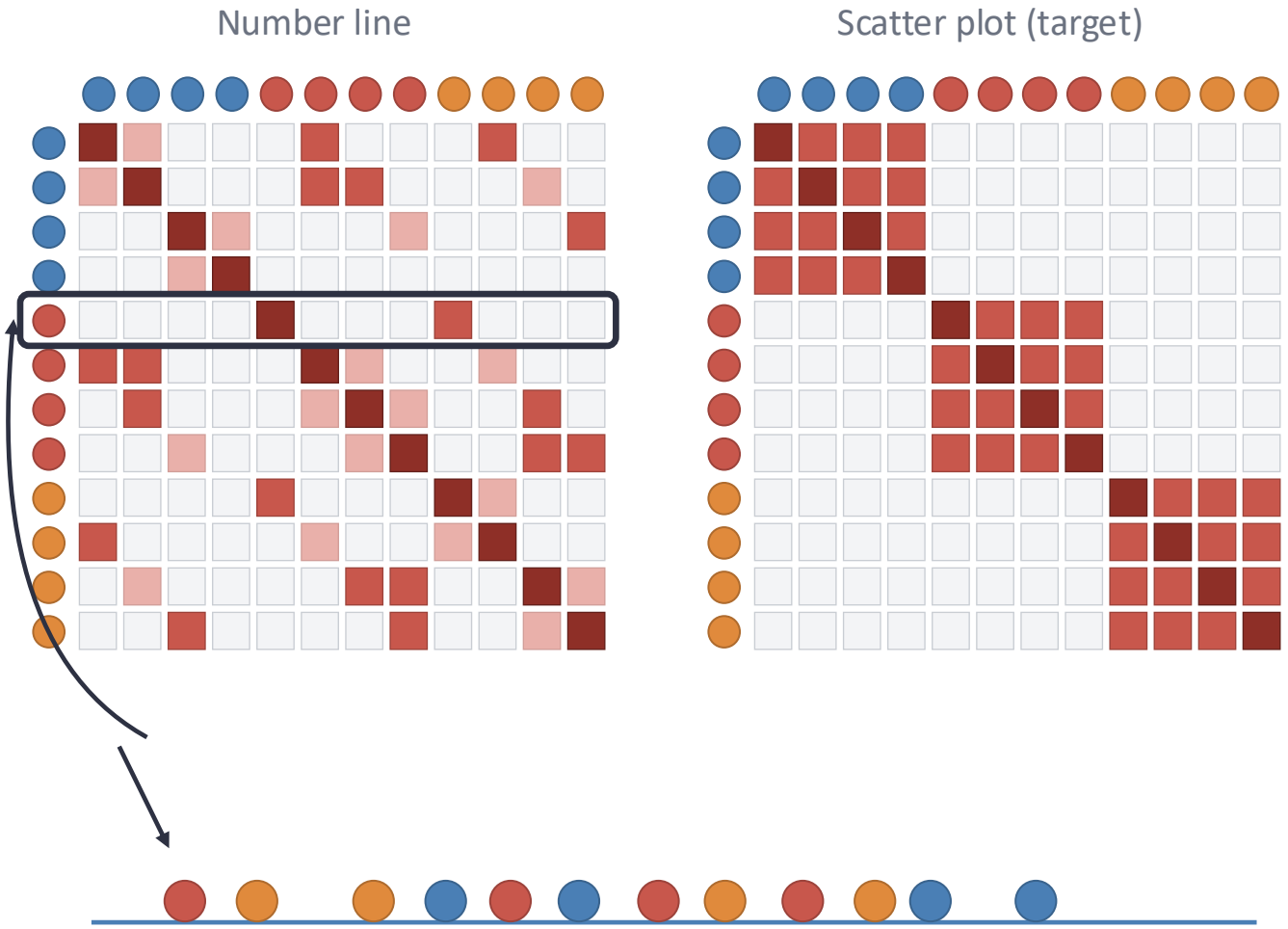
The goal of moving this point (the first red point on the number line)…



The goal of moving a point

The goal of moving this point (the first red point on the number line)…

…is to make **its row** in the number-line matrix…

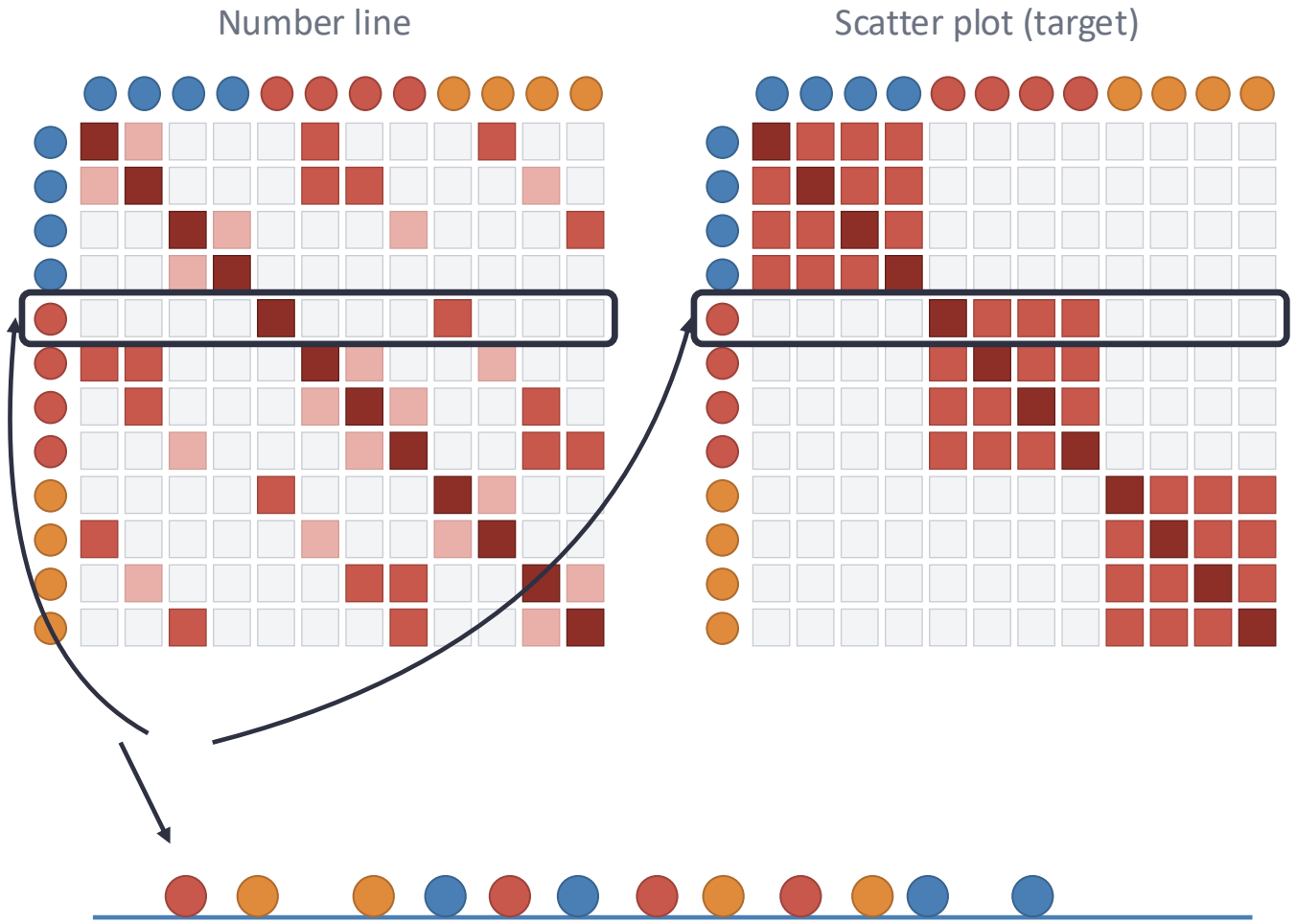


The goal of moving a point

The goal of moving this point (the first red point on the number line)…

…is to make **its row** in the number-line matrix…

…look like **its row** in the scatter-plot matrix.



In formulas: comparing the two matrices

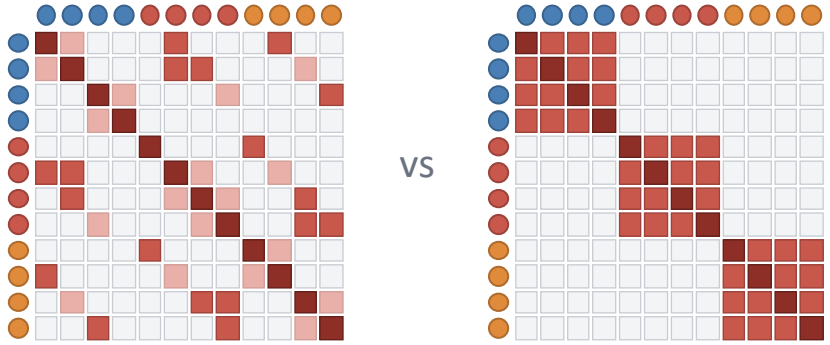
- To make the two matrices agree, we need one number that says **how different** they are.
- t-SNE uses the: **Kullback–Leibler (KL) divergence**: **C**,
- and moves the points to make C as **small** as possible.

$$C = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

where:

$p_{ij},\ q_{ij}$	similarities in the original data and on the number line
$q_{ij} = p_{ij}$	the pair is placed correctly: log 1 = 0, so it adds nothing to C
$p_{ij} \gg q_{ij}$	neighbours placed far apart: **a large cost**. So t-SNE works hardest at keeping neighbours together.

Q (number line) P (scatter plot)



Reading the formula: why the KL divergence?

$$C = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Step by step

1. P and Q each add up to 1, so both are probability distributions over the same pairs.
2. Compare them pair by pair through the ratio p_{ij} / q_{ij}.
3. Take the logarithm: a perfect match gives log 1 = 0.
4. Weight by p_{ij}: errors count in proportion to how similar the pair really is.
5. Add up over all pairs: one number, C, to make as small as possible.

Why can we use a divergence at all?

Because we scaled both matrices. P and Q are two probability distributions over the same pairs, and KL is the standard measure of how far one distribution is from another.

Why the logarithm of a ratio?

The ratio says how far off q_{ij} is, as a multiple. The logarithm turns a perfect match into 0, a q that is too small into a positive number, and a q that is too large into a negative one. Summed over all pairs, $C \geq 0$, and $C = 0$ only when $Q = P$.

Why weight by p_{ij}?

Mistakes matter in proportion to how similar the pair really is. **Asymmetric "weights".**
So t-SNE keeps neighbours together, but does not insist that distant points stay distant.

$$0.1 \log \frac{0.1}{0.01} \approx 0.23 \qquad \text{vs} \qquad 0.01 \log \frac{0.01}{0.1} \approx -0.023$$

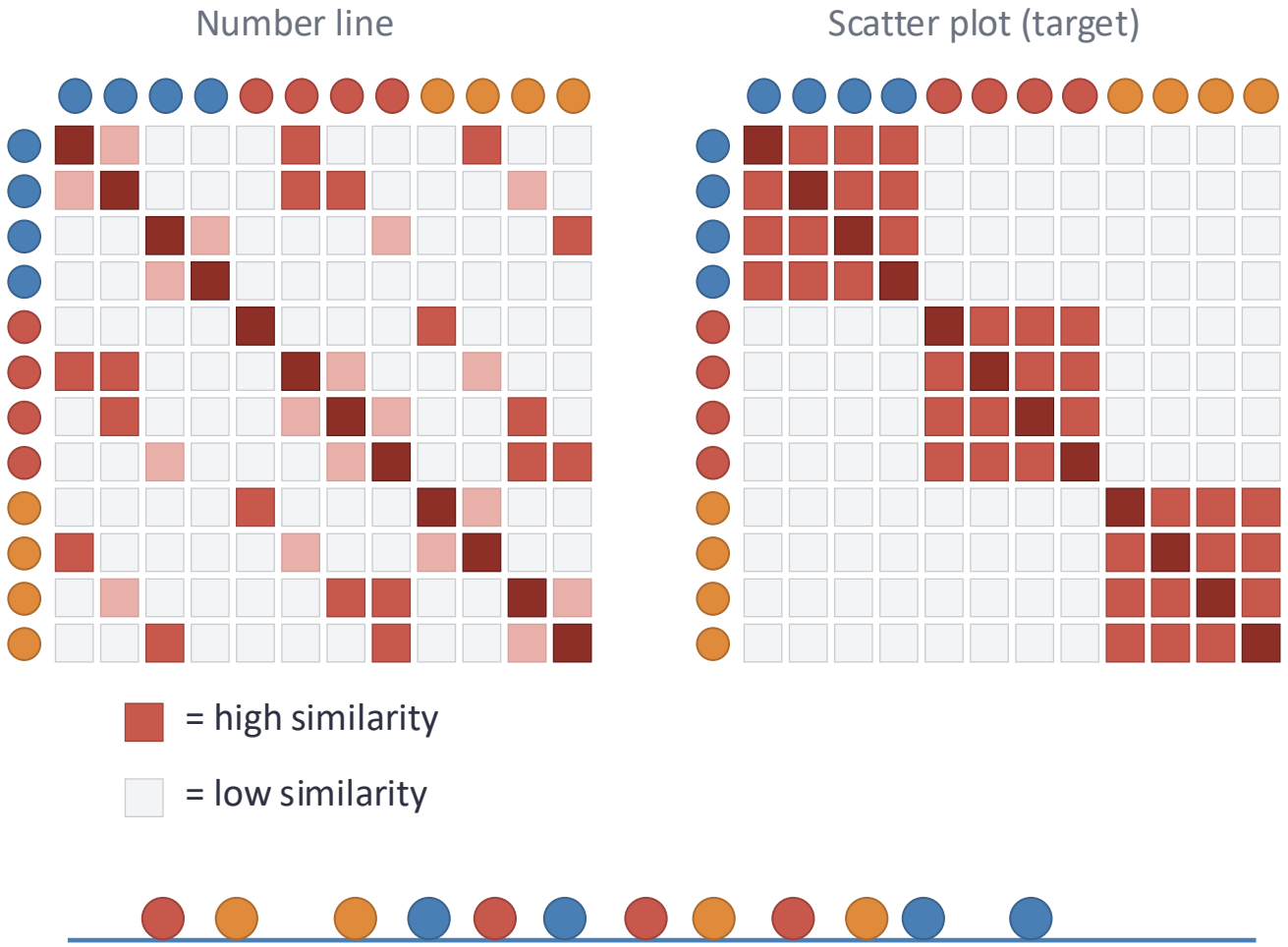
What does this mean for the picture?

t-SNE works hard to keep true neighbours together, but cares little whether distant points stay distant.
This is why the **neighbourhoods** in a t-SNE plot are reliable, but the **distances between clusters** are not.

Moving the points, a little at a time

t-SNE moves the points **a little at a time**. At each step it chooses a direction that makes the matrix on the left more like the matrix on the right. The step size and the number of steps are **settings you choose**.

It uses small steps because, a little like a game of chess, the problem cannot be solved all at once. Instead, it goes one move at a time.

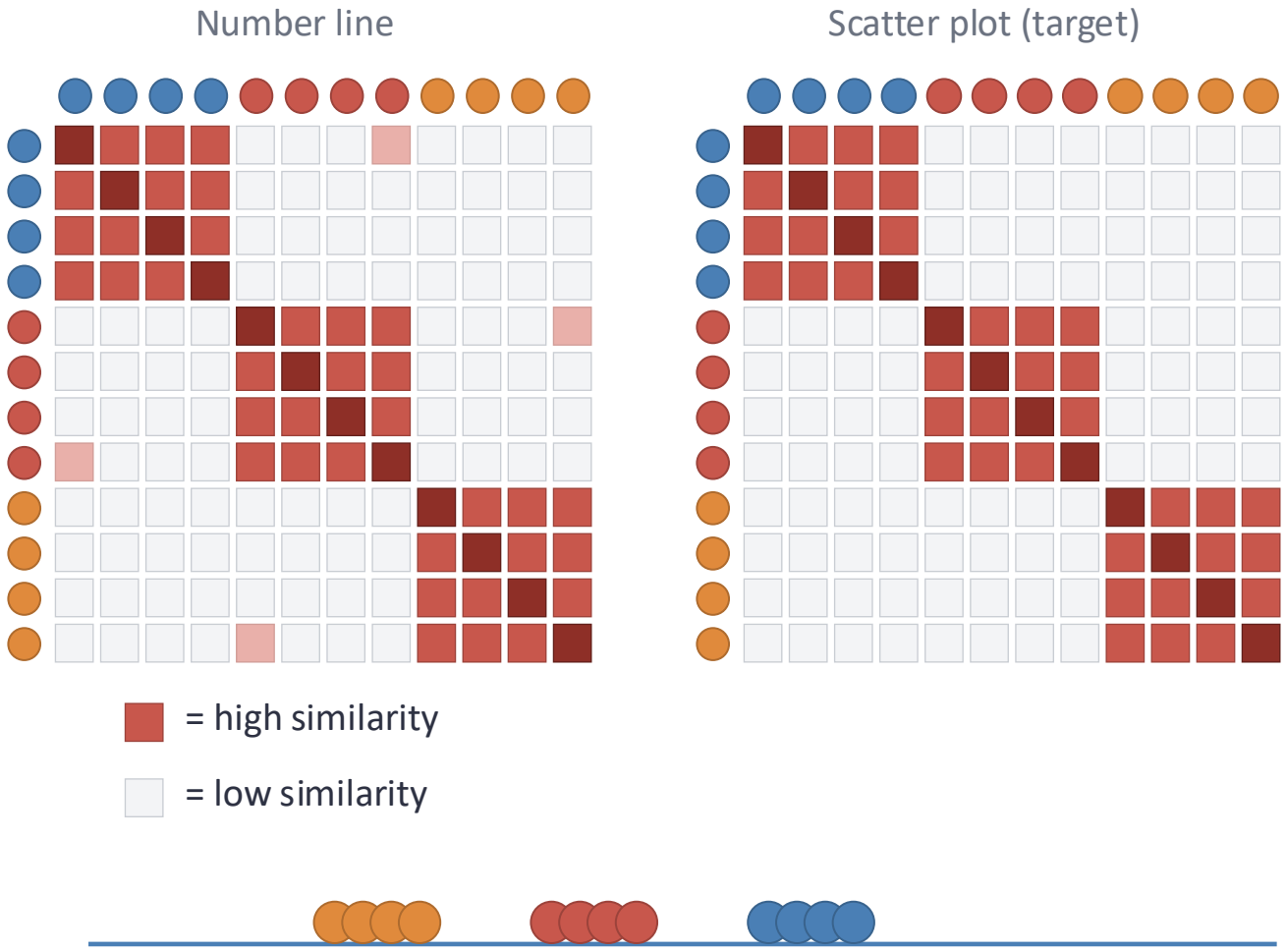


Moving the points, a little at a time

t-SNE moves the points **a little at a time**. At each step it chooses a direction that makes the matrix on the left more like the matrix on the right. The step size and the number of steps are **settings you choose**.

It uses small steps because, a little like a game of chess, the problem cannot be solved all at once. Instead, it goes one move at a time.

When the two matrices agree, the clusters from the scatter plot have reappeared on the number line.



In formulas: moving the points

Each small step moves every point in the direction that **reduces C** the most: this is **gradient descent**.

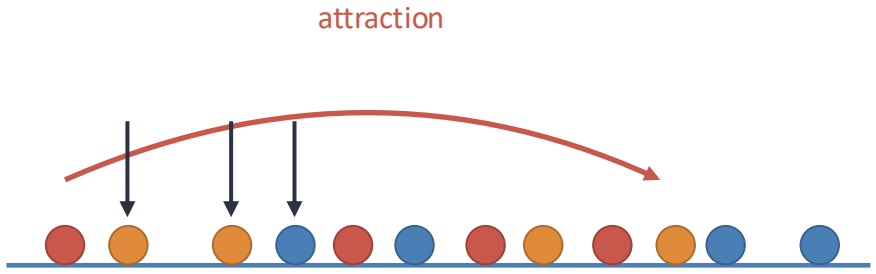
That direction is a sum of "pulls and pushes" from all the other points.

$$y_i \leftarrow y_i - \eta \frac{\partial C}{\partial y_i}$$

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij}) (y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

intesity — direction

The attraction and repulsion from the start:



where

η — the step size (learning rate), **which you choose**

$p_{ij} > q_{ij}$ — too far apart: y_i moves towards y_j (**attraction**)

$p_{ij} < q_{ij}$ — too close: y_i moves away from y_j (**repulsion**)

Why use a t-distribution?

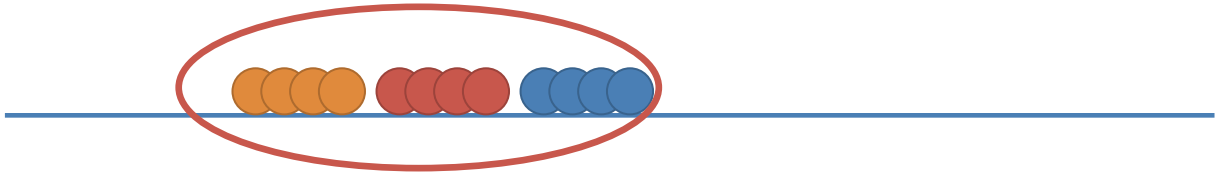
Why use a t-distribution on the number line, rather than a normal curve?

Its tails are taller, so two points only reach a low similarity score when they are placed **further apart** than a normal curve would need.

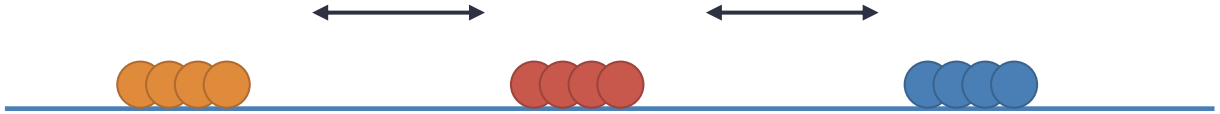
This pushes different clusters apart. Without it, the clusters would all **clump up in the middle** and be harder to see.

In numbers: to bring a similarity down to 0.01, a normal curve needs a distance of about **3**; the t-distribution needs about **10**.

With a normal curve on the number line



With a t-distribution on the number line



Same clusters, but now clearly separated.

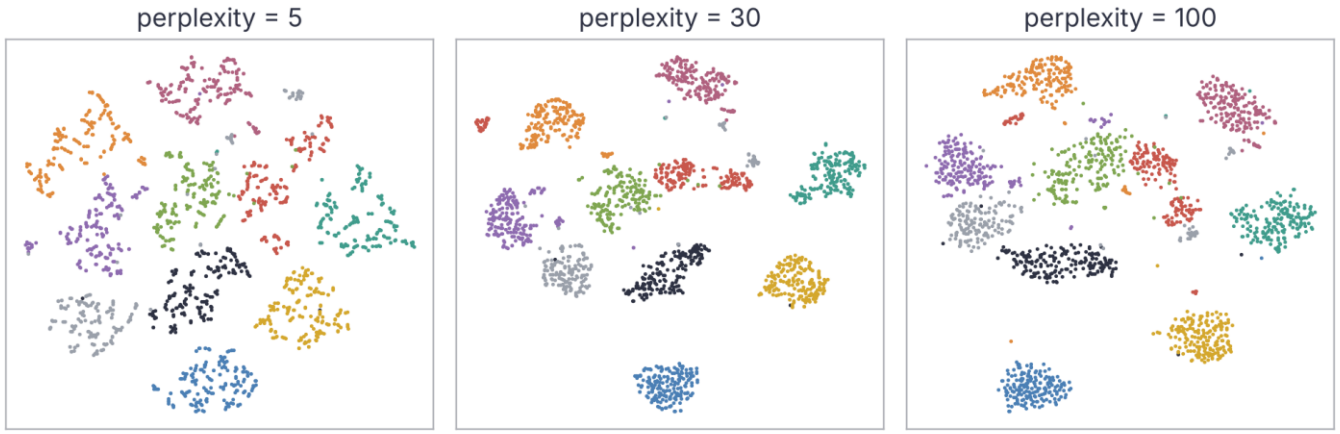
Reading t-SNE plots with care

The picture depends on the **perplexity you choose**: try a few values (here: the digits again).

Cluster sizes mean little: each σ_i adapts to the local density, so dense and sparse clusters end up looking alike.

Distances between clusters are not reliable either; only the neighbourhoods are.

Different random seeds give different layouts, and there is no formula for placing a new point: t-SNE is for **visualisation**, not a general transform like PCA.




```
from sklearn.manifold import TSNE

Y = TSNE(n_components=2, perplexity=30,
         init="pca", random_state=0).fit_transform(X)
```

PCA and t-SNE

	PCA	t-SNE
Type of map	Linear: a projection	Non-linear: no formula
What it keeps	Global variance, large distances	Local neighbourhoods
How it is computed	Eigenvectors, solved exactly	Gradient descent on KL
Same result every run?	Yes	No: depends on the seed
Axes and distances	Meaningful	Only the neighbourhoods
New data points	Project with the same directions	Rerun t-SNE
Used for	Visualisation, compression, features	Visualisation only

Used together in practice

1 Reduce first

Run PCA down to about 50 dimensions, then run t-SNE on those: less noise, and much faster.

2 Start from PCA

scikit-learn's default starting picture for t-SNE is the PCA projection, which t-SNE then refines locally.

In short: PCA gives the best **flat view** and keeps the large-scale layout; t-SNE gives up flatness to keep **who is next to whom**.

Reading

- Hastie, Tibshirani and Friedman (2009). "***The Elements of Statistical Learning (ESL)***", 2nd edition.
  §14.8, multidimensional scaling, and §14.9, nonlinear dimension reduction: background on preserving distances and neighbourhoods.
- Murphy (2022). "***Probabilistic Machine Learning: An Introduction***", MIT Press, at probml.github.io.
  §20.4.10, t-SNE; also §20.4.4, multidimensional scaling.

Further reading

- van der Maaten and Hinton (2008). "***Visualizing Data using t-SNE***". Journal of Machine Learning Research 9, 2579–2605.
  The original paper: Sections 2 and 3 cover SNE and t-SNE, including the crowding problem.
- Wattenberg, Viégas and Johnson (2016). "***How to Use t-SNE Effectively***". Distill, at distill.pub/2016/misread-tsne.
  Interactive examples of how perplexity and the number of steps change the picture, and how plots can mislead.