# Introduction to Machine Learning (for AI)

## Unsupervised Learning

Dr. Andreea Sburlea & Dr. Matias Valdenegro & Dr. Marco Zullich

January 9, 2025

# Today's Lecture

Today we will start talking about **unsupervised learning**.

Specifically, we will talk about...

- Dimensionality reduction, and
- Clustering.

# Unsupervised Learning

## Supervised Learning

"Learning with labels": we have a dataset of pairs $(x_i, y_i)$ and a model $f$.
The goal is to produce predictions $\hat{y}_i = f(x_i)$ as *close* as possible to the true labels $y_i$.
The closeness is measured by the loss function $L$.

## Unsupervised Learning

We do not have labels, just some datapoints $x_i$.
The goal is to find some *structure* in the data, and to exploit it to solve some tasks using a model $f$:

- Dimensionality reduction
- Clustering
- Generate new data
- Estimate density
- ...

# Dimensionality Reduction

**Dimensionality reduction** indicates the act of trimming the number of features in a dataset, while retaining as much *information* as possible.

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} \rightarrow \bar{X} = \begin{pmatrix} \bar{x}_{11} & \cdots & \bar{x}_{1r} \\ \bar{x}_{21} & \cdots & \bar{x}_{2r} \\ \vdots & \ddots & \vdots \\ \bar{x}_{n1} & \cdots & \bar{x}_{nr} \end{pmatrix}$$

with $r \ll d$.

# Dimensionality Reduction – Motivation

In many cases and datasets, the number of features can be high, and they might be redundant (correlated). There are cases when one might want to reduce the number of dimensions, but keep some structure from the data.

- *Data visualization*: data needs to be 2D or 3D, and by reducing dimensions, it is possible for humans to visualize it.
- *Data preprocessing*: if the dataset has too many dimensions, dimensionality reduction could be used as pre-processsing step, in order to reduce it to a manageable number of dimensions.
- *Computational tractability*: processing data in a high dimensional space can be computationally intractable.

# Curse of Dimensionality

It is a very common problem in mathematics and computer science, where algorithms, concepts, intuitions do not scale with the number of dimensions.

- *Data sparsity*: when the number of dimensions increases, the volume represented by that space increases exponentially. Then data points become sparse.
- *Combinatorics*: if one has $d$ binary variables, the number of possible combinations is $2^d$. This is manageable in low $d$, but not at high values of $d$.

# Curse of Dimensionality

- *Sampling*: $n \propto d$ feature dimensionality also defines the number of data points that we should capture or gather, and this number is also exponential in the number of dimensions $d$. **The more features, the more data we need to sample to have a representative dataset**.

# Curse of Dimensionality

It is difficult to define clearly, but there are some practical issues that help define it.

## Intuitions

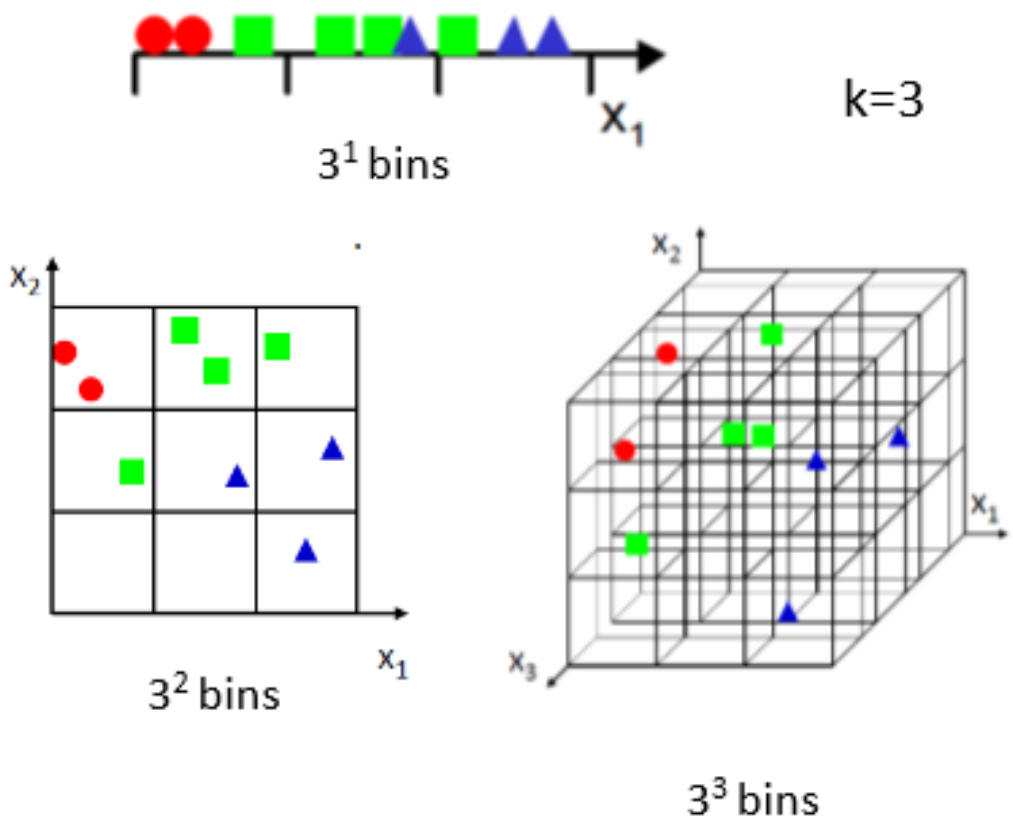
Our mathematical intuitions on low dimensional spaces do not extend/work to highly dimensional spaces.

## Performance

Machine learning model performance decreases with increasing dimensionality (of features, spaces, etc).

# Curse of Dimensionality – Sparsity

We sample $n = 9$ points into a unit hypercube. As we increase the dimension $d$ from 1 to 3, the data becomes sparser and sparser:



# Catcurse of Dogmensionality

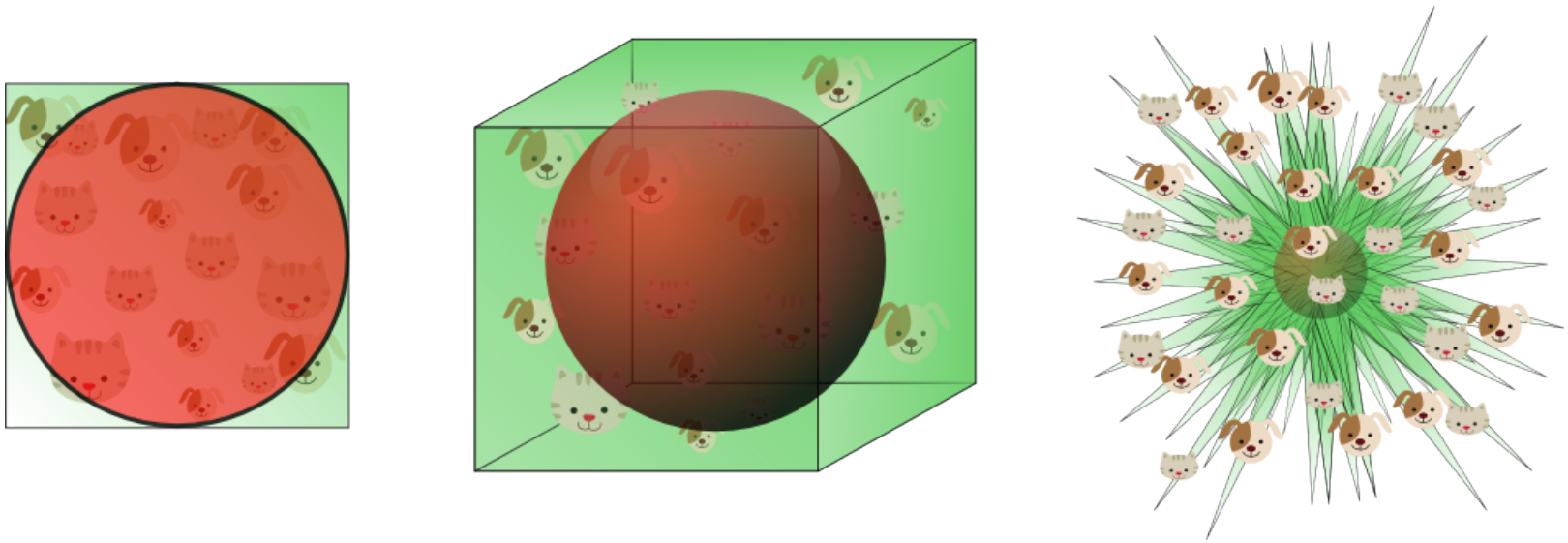
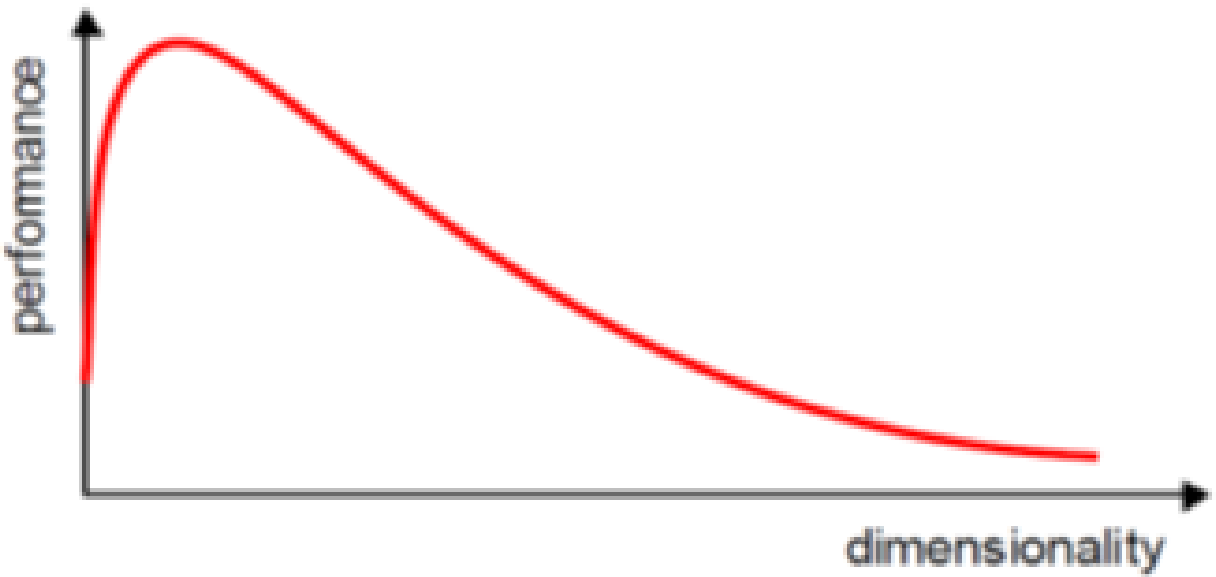


Figure: Source: `https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/`

# Curse of Dimensionality – More features

Adding features does not always result in better models.

Training set performance may increase, but usually generalization decreases:

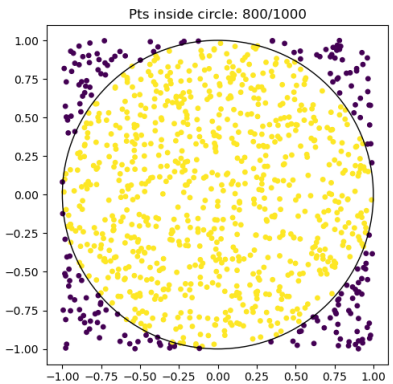


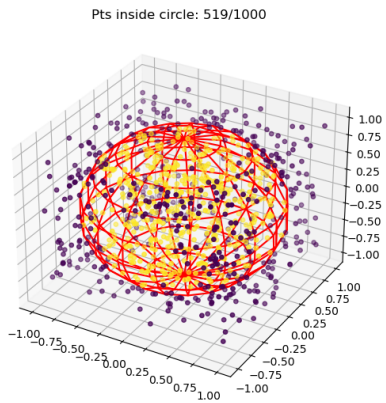
# Curse of Dimensionality – Sampling

Sampling points inside a given distribution is harder and harder as $d$ increases.

We want to sample points inside a unit sphere. **Q:** *How can we do it?*

We sample inside a unit hypercube (very simple to do), then we pick points s.t. $||x||_2 \leq 1$.



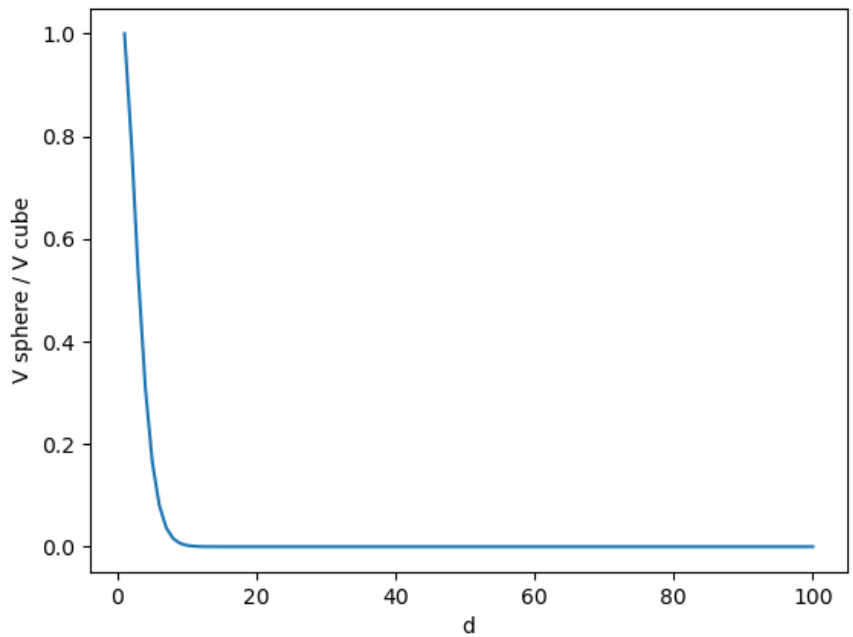




Proportion of points inside unit sphere given dimension $d$

| $d$ | prop. |
|---|---|
| 1 | 1.00 |
| 2 | 0.78 |
| 3 | 0.52 |
| 4 | 0.34 |
| 10 | 0.03 |
| 100 | 0.00 |

# Curse of Dimensionality - Cubes over Spheres!

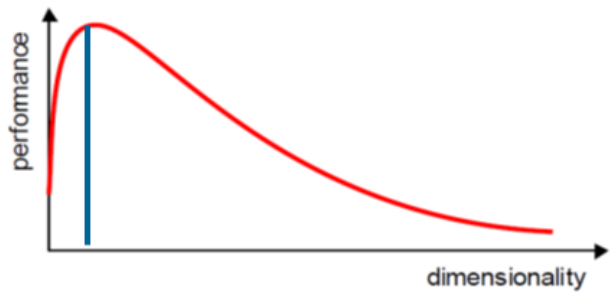
Volume of a $d$-dimensional Hyper-sphere relative to a unit Hyper-cube (inscribed) is $V(d) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} 0.5^d$.




The volume ratio between sphere and cube goes to zero, showing sparsity with increasing number of dimensions $d$.

# Dimensionality Reduction

- What is the objective?
  - Choose an optimum set of features of lower dimensionality to improve classification accuracy.




- Different methods can be used to reduce dimensionality:
  - Feature extraction
  - Feature selection

# Dimensionality Reduction

**Feature extraction**: finds a set of new features (i.e., through some mapping f()) from the existing features.

**Feature selection**: chooses a subset of the original features.

The mapping f() could be linear or non-linear

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

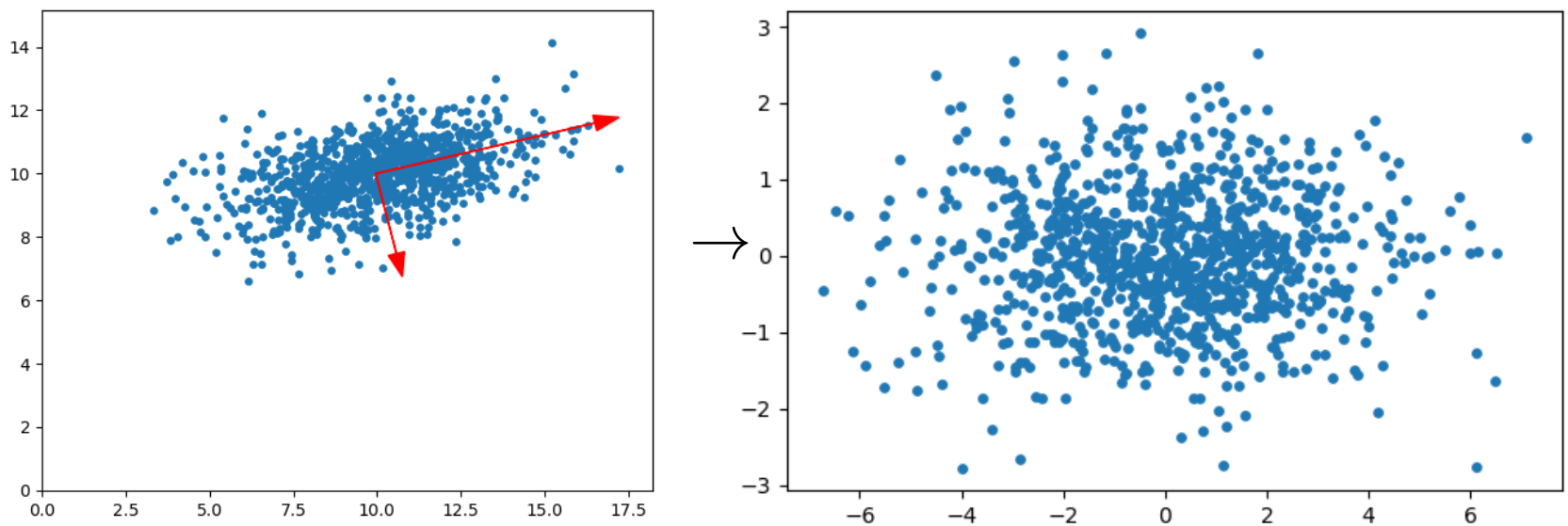
K<<N

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \rightarrow \mathbf{y} = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \cdot \\ \cdot \\ x_{i_K} \end{bmatrix}$$

K<<N

# Principal Components Analysis

- The classic method for dimensionality reduction.
- Underlying idea: find *rototranslation* (rotation + translation) of the axes (dimension) to capture the maximum variance in the data:



# Principal Components Analysis – how to

1. *Translation*: center the data w.r.t. its mean:

$$\bar{X} = X - \mathbb{E}[X]$$

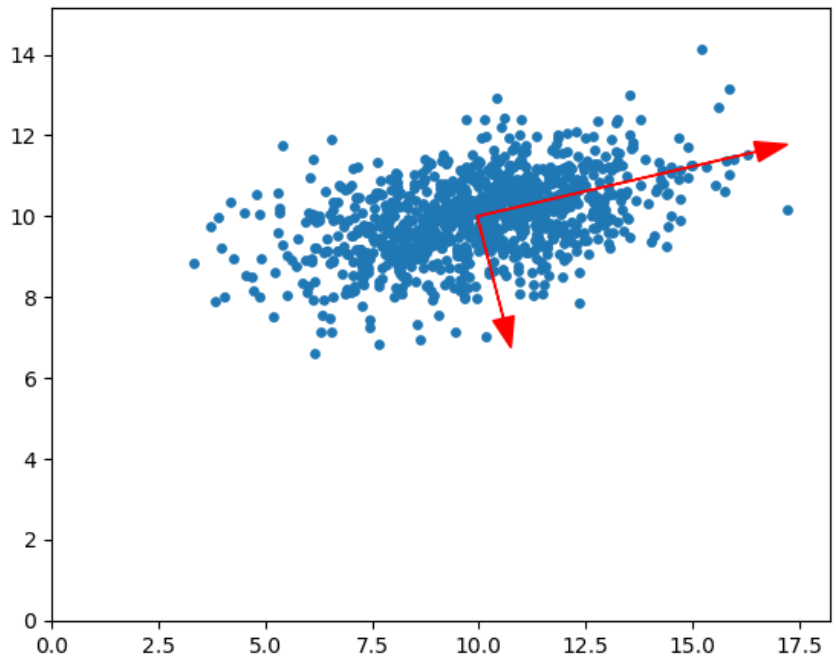
2. Compute the *covariance matrix*:

$$\Sigma = \mathrm{COV}(X) = \mathrm{COV}(\bar{X}) = \bar{X}^\top \bar{X}$$

$\Sigma$ tells us how the data is spread out and how the features *interact* with each other:

$$\Sigma(X) = \begin{pmatrix} \sigma_{x_1}^2 & \sigma_{x_1,x_2} & \cdots & \sigma_{x_1,x_d} \\ \sigma_{x_1,x_2} & \sigma_{x_2}^2 & \cdots & \sigma_{x_2,x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{x_1,x_d} & \sigma_{x_2,x_d} & \cdots & \sigma_{x_d}^2 \end{pmatrix}$$

# Principal Components Analysis – Covariance



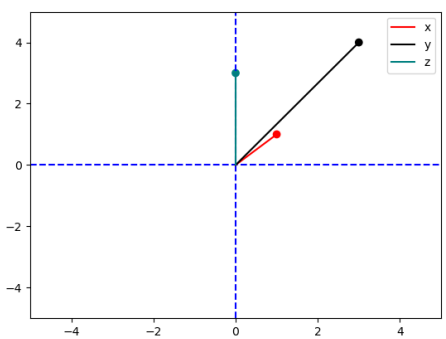

$$\Sigma = \begin{pmatrix} 5 & 1 \\ 1 & 1 \end{pmatrix}$$

# Principal Components Analysis – Matrices as transformations

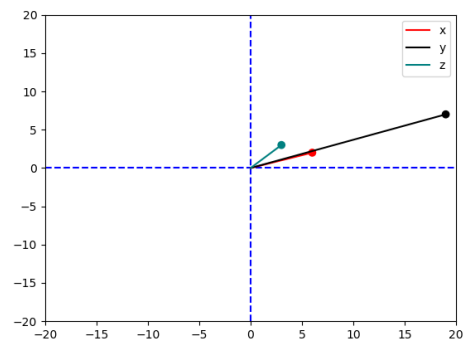
All matrices can be seen as transformations of the space. $A \in \mathbb{R}^{p \times q}$ transforms any vector $v$ of $R^q$ into a vector of $R^p$:

$$\mathbb{R}^p \ni \mathbf{w} = A\mathbf{v}$$

Specifically, a square matrix $A \in \mathbb{R}^{d \times d}$ represents an isomorphism, since it does not change the dimensionality of the space, but act by rotating and reflecting the space.

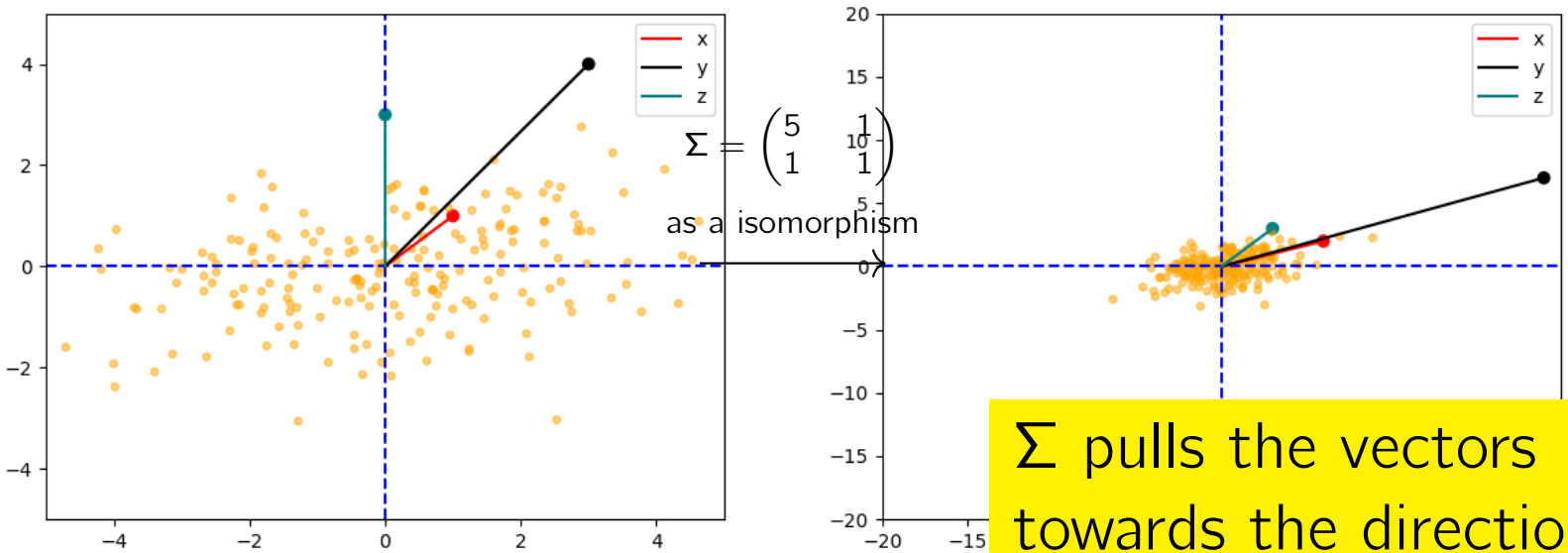



$$\Sigma = \begin{pmatrix} 5 & 1 \\ 1 & 1 \end{pmatrix}$$ as a isomorphism



# Principal Components Analysis – Matrices as transformations

We can overlay a sample from a Gaussian with mean $(0, 0)$ and covariance matrix $\Sigma$ to showcase the effect of $\Sigma$ as a transformation.




$\Sigma$ pulls the vectors towards the direction of maximum variance

# Principal Components Analysis – Eigendecomposition

The *pull* directions are given by the *eigenvectors* of $\Sigma$.
The *pull* magnitude is given by the *eigenvalues* of $\Sigma$.
To recover them we can use the equation

$$\Sigma \underbrace{\mathbf{v}}_{\textit{eigenvector}} = \underbrace{\lambda}_{\textit{eigenvalue}} \mathbf{v}$$

The equation allows for $d$ solutions, we can gather the eigenvectors in a matrix $V \in \mathbb{R}^{d \times d}$ and the eigenvalues in a matrix $\Lambda \in \mathbb{R}^{d \times d}$, with $\Lambda = \text{diag}(\lambda_1, \ldots \lambda_d)$.

$\Sigma$ can be decomposed as $\Sigma = V \Lambda V^\top$.

# Principal Components Analysis – how to

Now that we have discovered a way to recover the directions of maximum variance, we can successfully continue with the recipe for PCA:

3. Eigendecompose $\Sigma = V \Lambda V^\top$ to get eigenvectors and eigenvalues.
4. The data can be *projected* onto the eigenvectors $V$ to obtain the new representation:

$$Y = \bar{X} V$$

# Principal Components Analysis – Dimensionality reduction

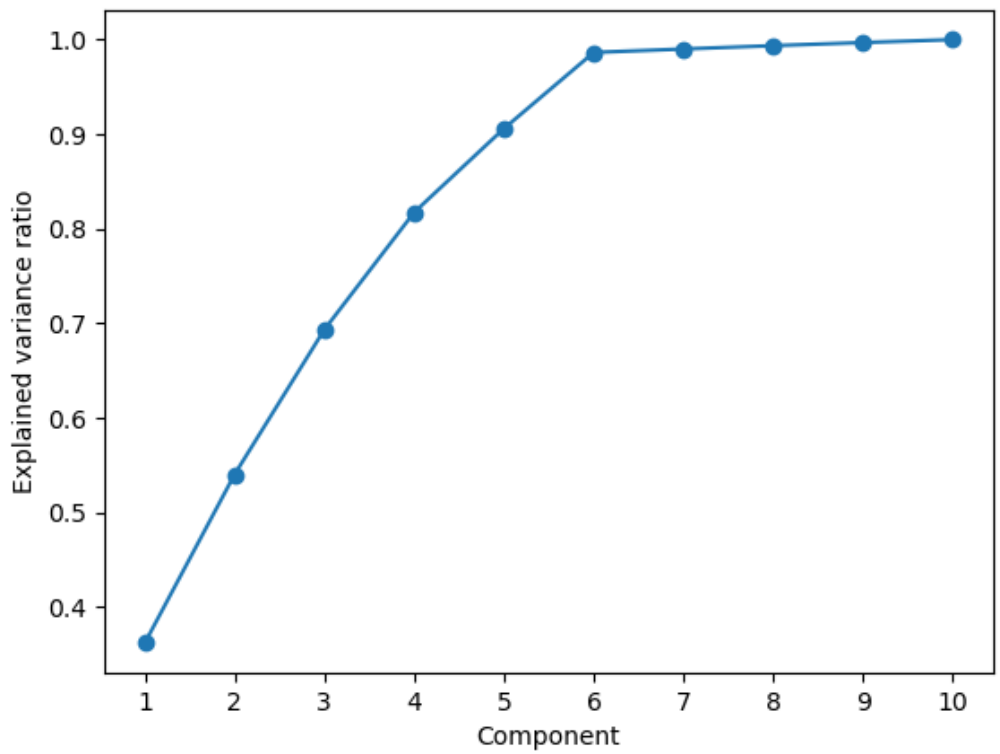
The eigenvalues represent the variance captured by each eigenvector (equivalent to "magnitude" of pull in the geometric interpretation).

If we want to reduce the dimensionality, we can select the top $r$ eigenvectors and the corresponding eigenvalues to obtain a representation of dimension $r$ that maximally captures the variance in the data:

$$Y = \bar{X} V_{[:,1:r]}$$

# Principal Component Analysis – in practice

We can sort the eigenvalues and plot them in an *eigenvalue spectrum* (a.k.a. *elbow plot*—the cumulative sum of each eigenvalue) to see how much variance in % is captured by each eigenvector:



# Principal Component Analysis – Reprojection

We can reproject the data back into the original space by inverting the transformation:

$$\tilde{X} = YV^\top + \mathbb{E}[X]$$

It can be shown that PCA minimizes the reprojection error

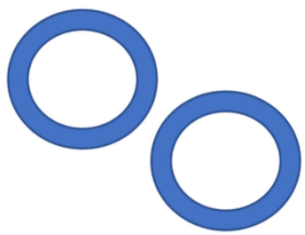
$$L = ||X - V^T XV||^2$$

.

# Principal Component Analysis – Drawbacks

- One of the major drawbacks of PCA is that it does not retain non-linear variance. This means PCA will not be able to get results for figures like this.

High-dim Space (3D)



Low-dim Space (2D)

- In simple terms, PCA works on retaining only global variance, and thus retaining local variance was the motivation behind t-SNE.

# t-Stochastic Neighbor Embedding (t-SNE)

- t-SNE is a very popular nonlinear dimensionality reduction method that is tailored for 2D/3D visualization.
- In a nutshell, t-SNE works by first transforming distances in the input feature space into a probability distribution. Another probability distribution with a low-dimensional support is defined and optimized in order to minimize the KL divergence between distributions.
- Then the low-dimensional support distribution is sampled to obtain the embedding representation. This puts the stochastic part of t-SNE.

# t-Stochastic Neighbor Embedding (t-SNE)

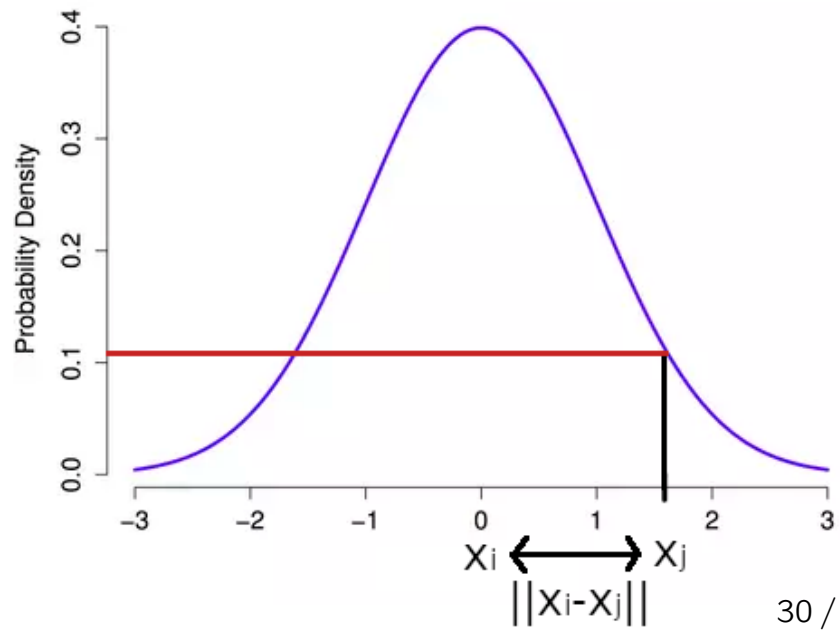
- Stochastic $\rightarrow$ not definite but random probability
- Neighbor $\rightarrow$ concerned only about retaining the variance of neighbor points
- Embedding $\rightarrow$ plotting data into lower dimensions
- t-SNE is a machine learning algorithm that generates slightly different results each time on the same data set, focusing on retaining the structure of neighbor points.
- It is an *iterative procedure*, i.e., the projection happens in multiple steps, each time trying to improve the embedding.

# t-Stochastic Neighbor Embedding (t-SNE)

We begin by obtaining the distances between all of the points in the dataset, to define the neighborhoods of our data points. We do not use a norm to compute the distance, but rather a Gaussian kernel:

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

Intuition: we measure the distance as the probability density that the point $x_j$ would have under a Gaussian centered in $x_i$, then we normalize the probabilities.



# t-Stochastic Neighbor Embedding (t-SNE)

The actual similarity scores $p_{ji}$ in the high-dimensional space are actually adjusted slightly from $p_{j|i}$:
since $p_{j|i}$ is not symmetric ($p_{i|j} \gtreqless p_{j|i}$)[1], we force the equality by defining $p_{ij}$ as the *average* of the other two:

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

$p_{ij}$ is then the actual *similarity score* between $i$ and $j$, the *probability* that $x_i$ and $x_j$ are neighbors in the high-dimensional space.

[1] This is because the two values are computed based on Gaussian distributions with possibly different values of $\sigma^2$

# t-Stochastic Neighbor Embedding (t-SNE)

The data is projected onto a low-dimensional space $\mathcal{Y}$ of dimension 2 or 3.
Here, the similarity between datapoints is measured using a t-distribution kernel:

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

The choice of the t-distribution is to allow for robustness to outliers.

# t-Stochastic Neighbor Embedding (t-SNE)

The low-dimensional embedding is learned by minimizing the KL divergence between the two distributions:

$$KL(P||Q) = \sum_{i,j|i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Training is operated via gradient descent on the projections.

We can view the training process as neighbors slowly "attracting" while the non-neighbors are "repelling" each other.

# t-Stochastic Neighbor Embedding (t-SNE)

This algorithm has 2 hyperparameters:

## Number of iterations

The number of iterations of gradient descent that the algorithm will run.

## Perplexity

The *perplexity* is related to the variance $\sigma_i^2$ that each Gaussian distribution will have for each point $x_i$. The higher the perplexity, the more variance each Gaussian will have, and the more points will be considered as neighbors.
A high perplexity means more points will contribute to "attracting" each other in the embedding space.

# t-Stochastic Neighbor Embedding (t-SNE)

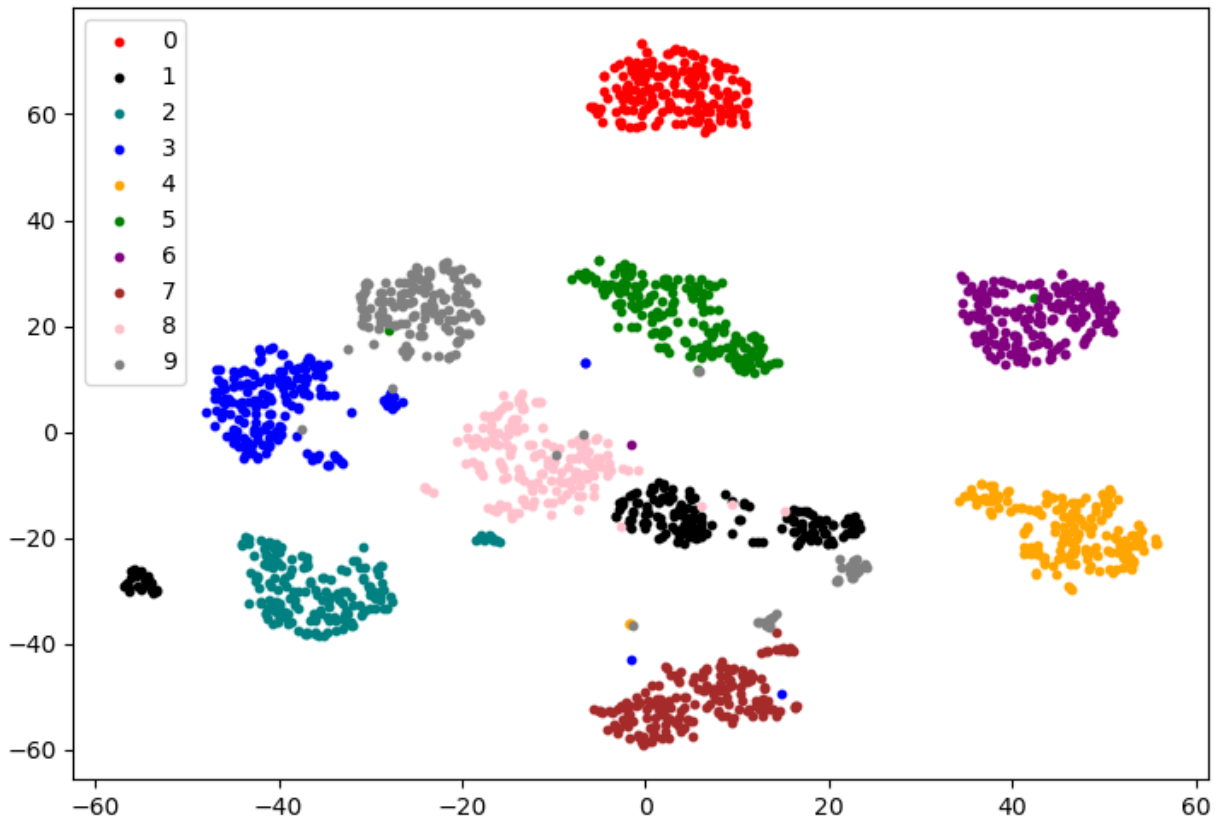



Figure: t-SNE on MNIST with perplexity=30

# Multi-Dimensional Scaling

- Another dimensionality reduction method, similar to t-SNE, but not stochastic.
- t-SNE does not try to preserve distances in the high-dimensional space, but MDS does at least approximate distances from the high to the low-dimensional manifold.
- MDS first computes pair-wise distances between points $x$ in the high-dimensional input space:

$$d_{ij} = ||x_i - x_j||$$

- The idea is that we learn an embedding $y$ that has a different number of dimension (usually lower), then using an optimization algorithm, we learn the values of $y$ that mimic the distances $d_{ij}$.

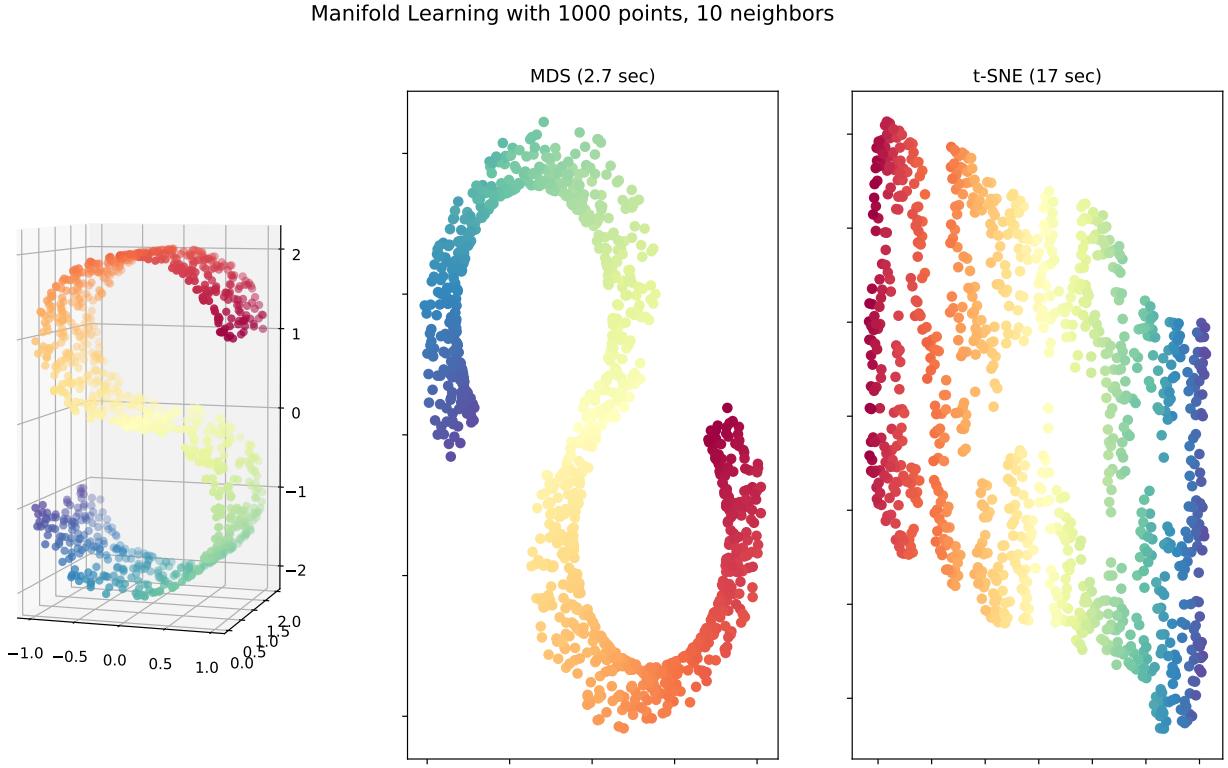
# Multi-Dimensional Scaling

- Then a low-dimensional embedding $y$ is optimized to closely mimic the pair-wise distances:

$$L(y) = \sum_{i \neq j} (d_{ij} - ||y_i - y_j||)^2$$

- The minimization of loss $L$ can be performed using gradient descent. Generally different solutions are obtained on each run, but as distances are approximated, they usually correspond to rotations of the low-dimensional space.

# t-SNE vs MDS


Manifold Learning with 1000 points, 10 neighbors
MDS (2.7 sec)
t-SNE (17 sec)
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
0.0
0.5
1.0
1.5
2.0

# What is Clustering?

**Clustering** is an unsupervised learning technique, where the task is to find patterns common to specific groups of data points. We call these groups **clusters**.

The idea is that elements in a cluster $C_i$ are more similar among themselves, than they are similar to a different cluster $C_j$.

This means that a measure of **similarity** among elements in a dataset is required.

# Clustering

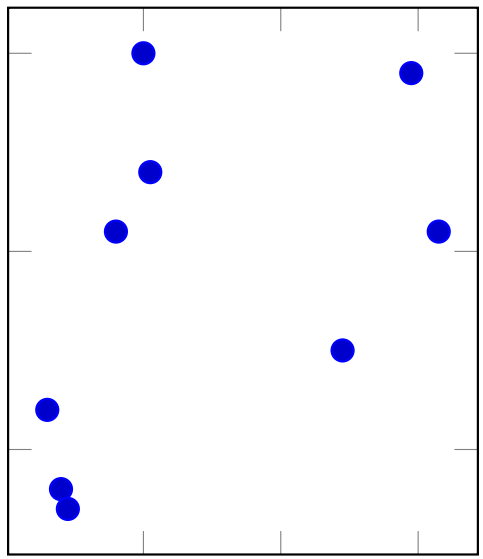


Figure: Raw Data

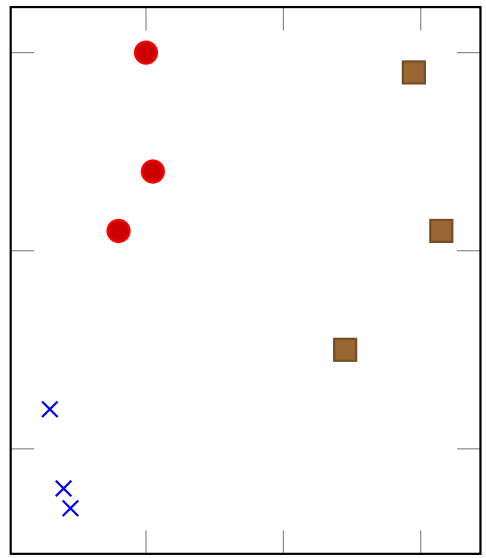


Figure: Clustered Data with $k = 3$ clusters

# Applications of Clustering

- User or customer segmentation/identification.
- Community identification (in social networks for example).
- Identifying different types of tissue in medical imaging.
- Recommendation systems.
- Anomaly detection.
- Image segmentation.
- NLP for lexical word ambiguity.
- ...

# Metrics for Clustering

## Within Cluster Sum of Squared Errors

Given cluster centers or centroids $c_j$ and associated data points $x_i$, the WSS metric computes the sum of squared distances of each point to its cluster *centroid*, across all clusters $C$:

$$WSS = \sum_{j}^{C} \sum_{i} d(c_j, x_i)^2 \tag{1}$$

Where $d$ can be any distance metric, not necessarily the euclidean distance.

A cluster centroid is any measure of *centrality* in the cluster (mean, median, mode, *etc.*).

# Metrics for Clustering

## Silhouette Value

First we define the mean distance between point $i$ and all other data points in the same cluster $C_i$.

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(x_i, x_j) \tag{2}$$

Then we define the smallest mean distance of point $i$ to all other points in other clusters where $i$ is not a member:

$$b(i) = \min_{k \neq i} \sum_{j \in C_k} d(x_i, x_j) \tag{3}$$

Finally, the Silhouette value is defined for $|C_i| > 1$ as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4}$$

# Metrics for Clustering

## Silhouette Value

Note that the $s(i)$ is computed for each sample, and its possible values are $-1 \geq s(i) \geq 1$.

- $s(i) \approx 1$ indicates that the point is well clustered.
- $s(i) \approx 0$ indicates that the point is on the border of two clusters ($a(i) \approx b(i)$).
- $s(i) \approx -1$ indicates that the point might be misclustered.

To obtain a **metric for the whole dataset**, the mean of $s(i)$ can be computed as the Silhouette coefficient:

$$SC(X) = \frac{\sum_i s(i)}{n} \tag{5}$$

# Selecting the Number of Clusters

As part of the clustering process, we assume that the data can be divided into $k$ groups or clusters, but how do we select the value of $k$? There are multiple ways:

**Elbow Method** Perform clustering using a variety of values of $k$, and then plot a clustering metric vs $k$, selecting the value of $k$ with diminishing returns (called the elbow). *Already seen for PCAs.*

# Selecting the Number of Clusters

**Silhouette Method** Try a selection of values of $k$, and select the one with the highest Silhouette coefficient (mean Silhouette value). Discrete optimization algorithms can be used to maximize this value too. Analysis of individual Silhouette values might also indicate how clusters can be improved.

**Other methods** Read this if interested.

# Elbow Method

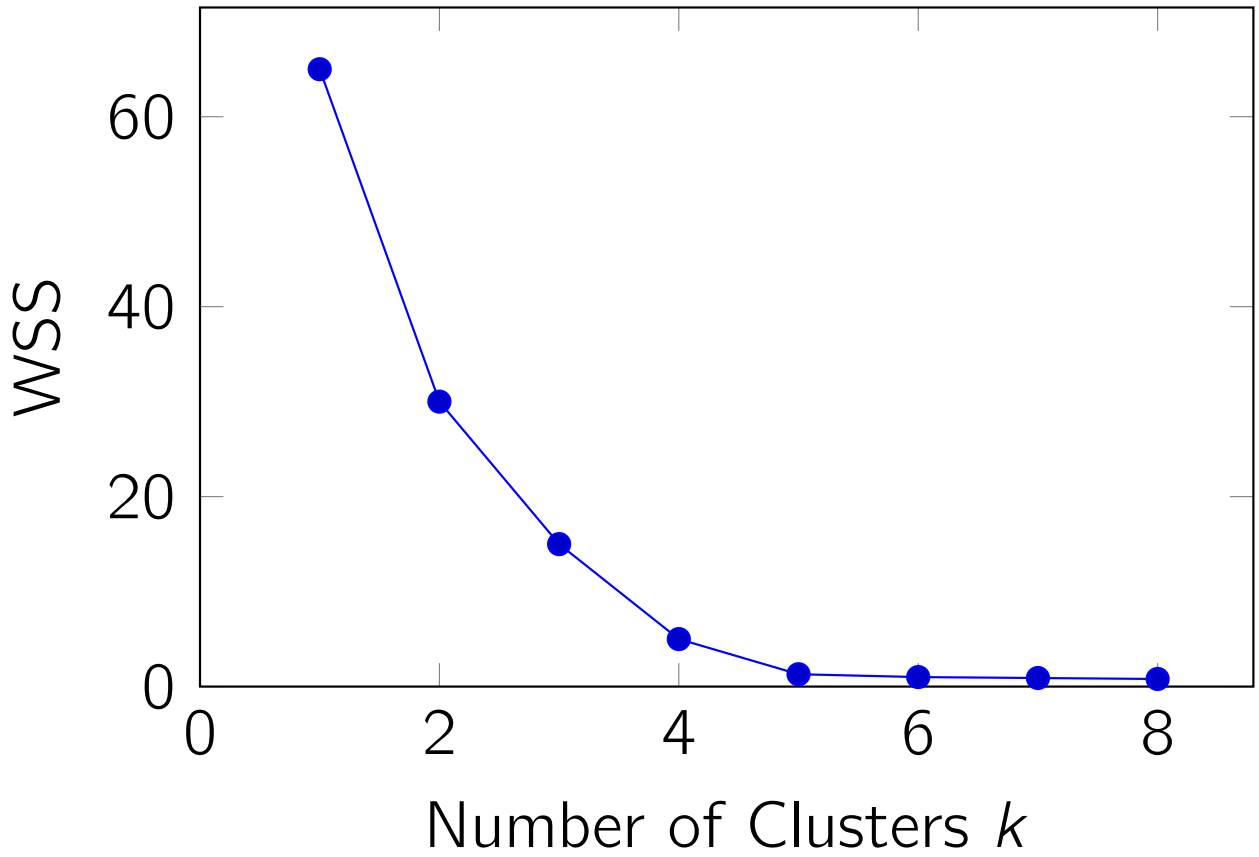
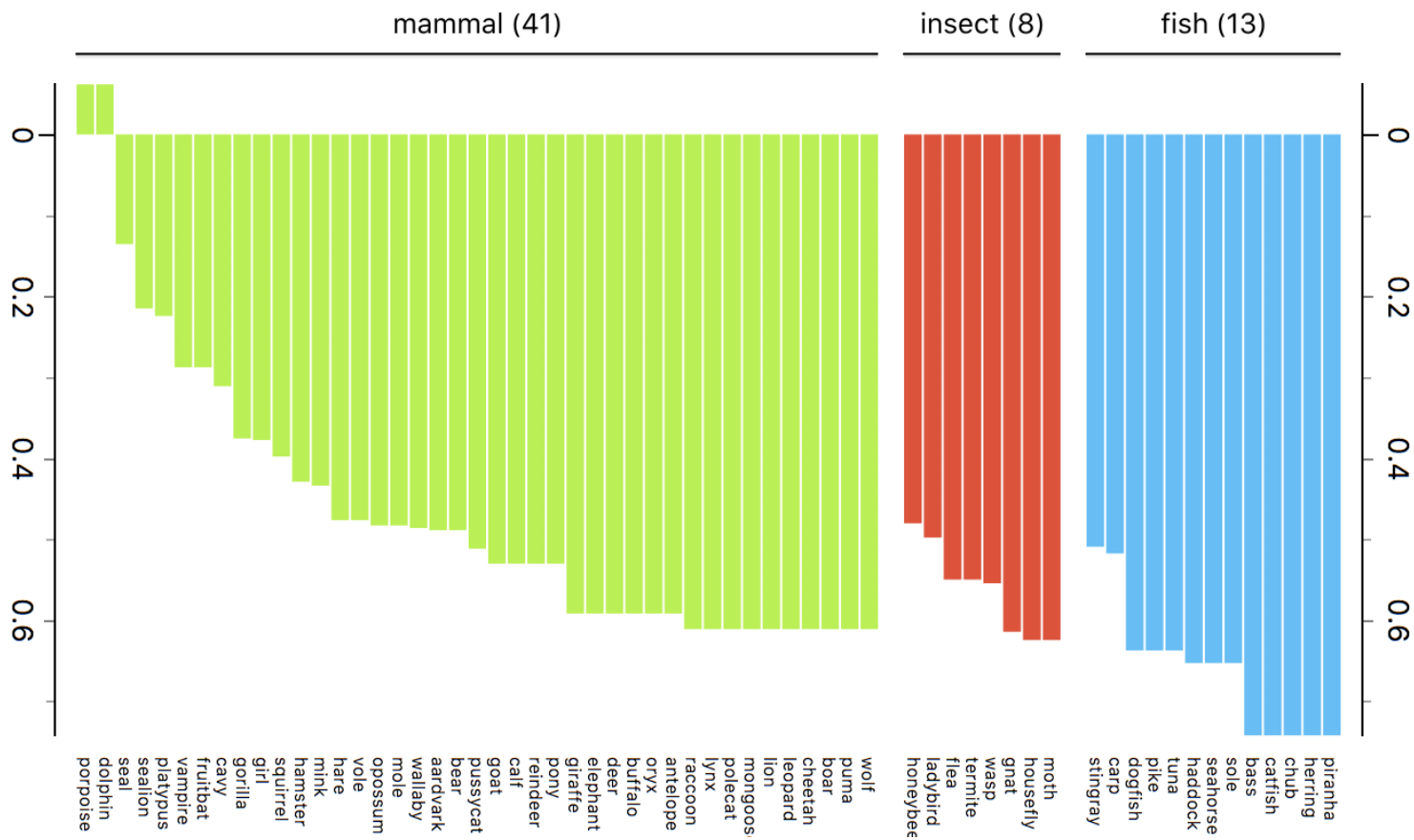



Figure: Example Elbow Plot of WSS vs $k$, here the elbow is at $k = 5$.

# Silhouette Method



# K-Means Clustering

It is a simple and intuitive clustering algorithm, using the following steps:

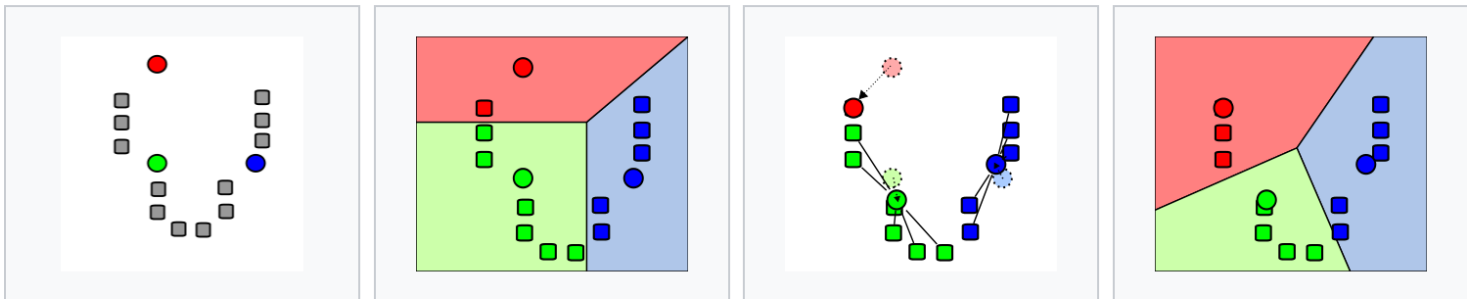
1. Initialize $k$ cluster centroids $c_k$ with random values.
2. Assign each data point $x_i$ to the closest (minimum distance metric) cluster centroids, producing $i \in c_k$.
3. For each cluster centroids $c_k$, recompute the centroids by taking the mean of data points assigned to that cluster:

$$c_k = \sum_{i \in c_k} x_i / n \tag{6}$$

4. Repeat steps 2 and 3 until convergence is reached (cluster centers do not change, up to some maximum tolerance, or max number of iterations reached).

# K-Means Clustering

**Demonstration of the standard algorithm**



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Figure: Source: Wikipedia Article on K-Means Clustering.

# K-Means Clustering - Assumptions

K-Means makes a lot of assumptions on data and cluster composition:

- Cluster centroids/centers are inside each cluster (not true if one has a half-moon shape data).
- Divides the feature space $x$ into voronoi regions, which might not be optimal.
- K-Means fails if the data has complex shapes, only works when clusters can be defined as distances from a centroid (similar to a $d$-dimensional sphere).

# K-Means Clustering - Assumptions

- Fails when: clusters have different variance, or one dimension dominates over the other (anisotropy), or when the number of data points is too skewed in each cluster.
- Of course, if $k$ is not "correct", the clusters will be very strange.
- Large values of $k$ can lead to overclustering.

# K-Means Clustering - Failure Cases

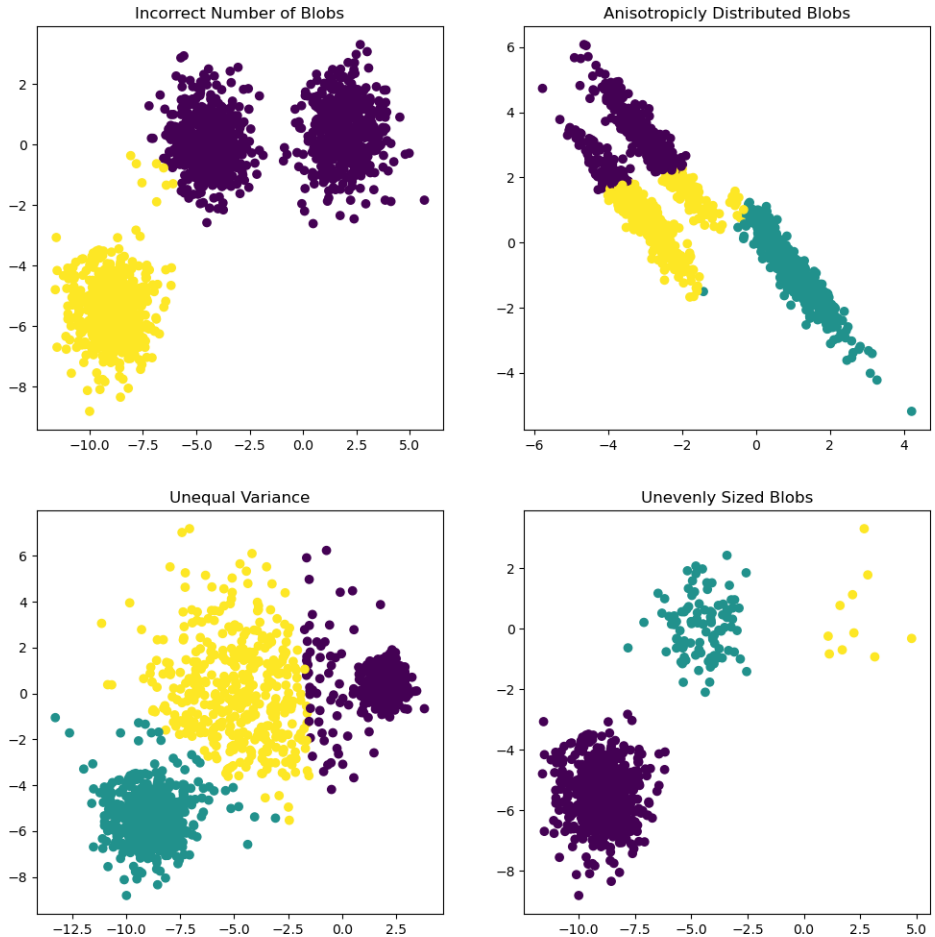



Figure: Source: `https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html`

# Introduction

- K-Means clustering fails to cluster data that shows non-convex or non-linear clusters, since it is based only on distance from centroids.
- DBSCAN in contrast makes clusters based on **density**, and can make clusters with shapes that K-Means cannot do (arbitrary shapes to be precise).
- It is a good algorithm to be aware of, and the concepts are quite simple.
- DBSCAN means Density-Based Spatial Clustering of Applications with Noise.

# Basic Concepts

## Hyperparameters

- $\epsilon$ – Distance threshold to define *connectedness* between points.
- `minPts` – Minimum number of points to define a *core point*.

## Core Point

A point $p$ is called core point if at least `minPts` points are within $\epsilon$ distance of it (including $p$).

## Direct Reachability

A point $q$ is directly reachable from point $p$ if $q$ is within distance $\epsilon$ of core point $p$ ($q$ is in the *neighborhood* of $p$). Points are only directly reachable from core points.

# Basic Concepts

## Reachability

A point $p$ is reachable from $p$ if there is a path $p_1, p_2, ..., p_n$ where each consecutive point is reachable, and the sequence of points starts at $p$ and ends in $p$. Note that all $p_i$'s should be core points, while $q$ does not have to be a core point.

Note that reachability is not symmetric.

## Outliers and Noise

Any point that is not a core point and not reachable from a core point is called an outlier or noise point.

# Basic Concepts

## Density Connection

Points $p$ and $q$ are density-connected if there is a point $p$ such that both $p$ and $q$ are reachable from $o$.

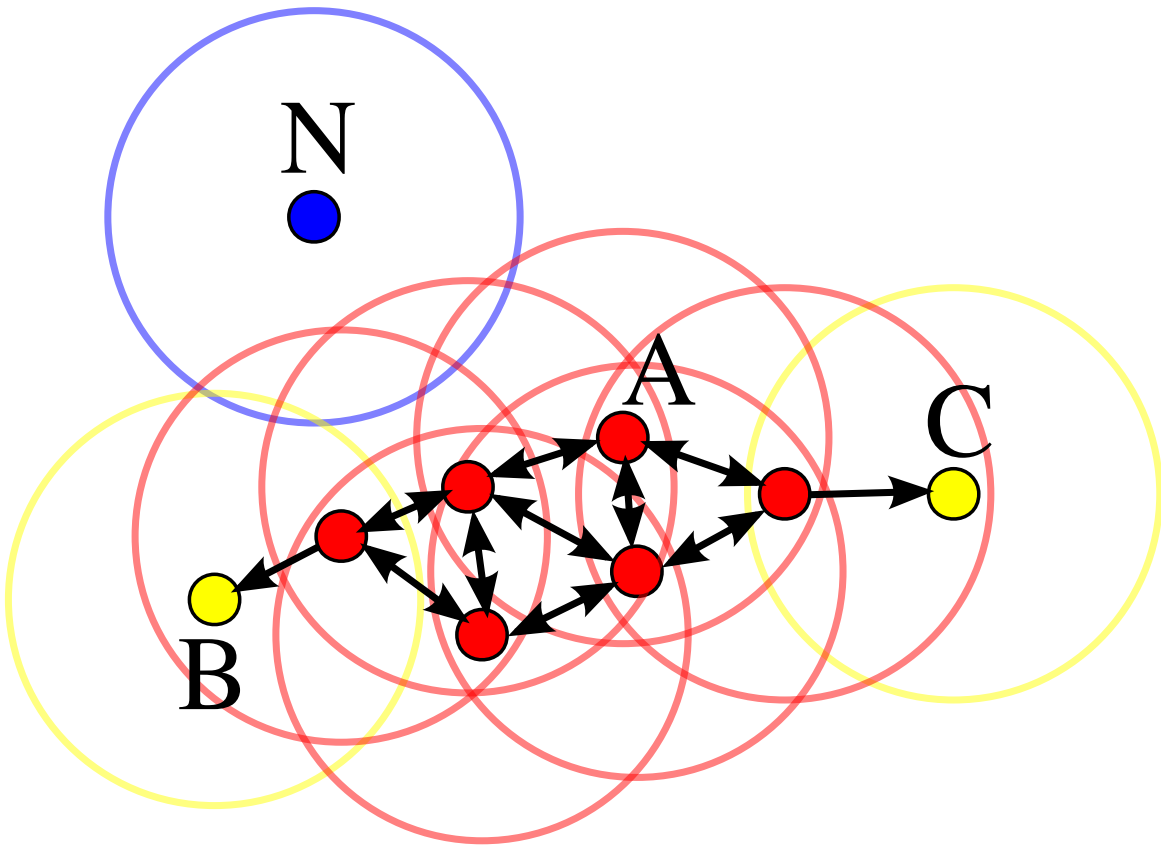
By definition only core points can reach non-core points, and this makes the relationship non-symmetric. Density-connectedness is symmetric.

## Cluster

A cluster in DBSCAN has two properties:

- All points in a cluster are mutually density-connected.
- If a point is density connected from some point in the cluster, it is part of the cluster.

# Basic Concepts

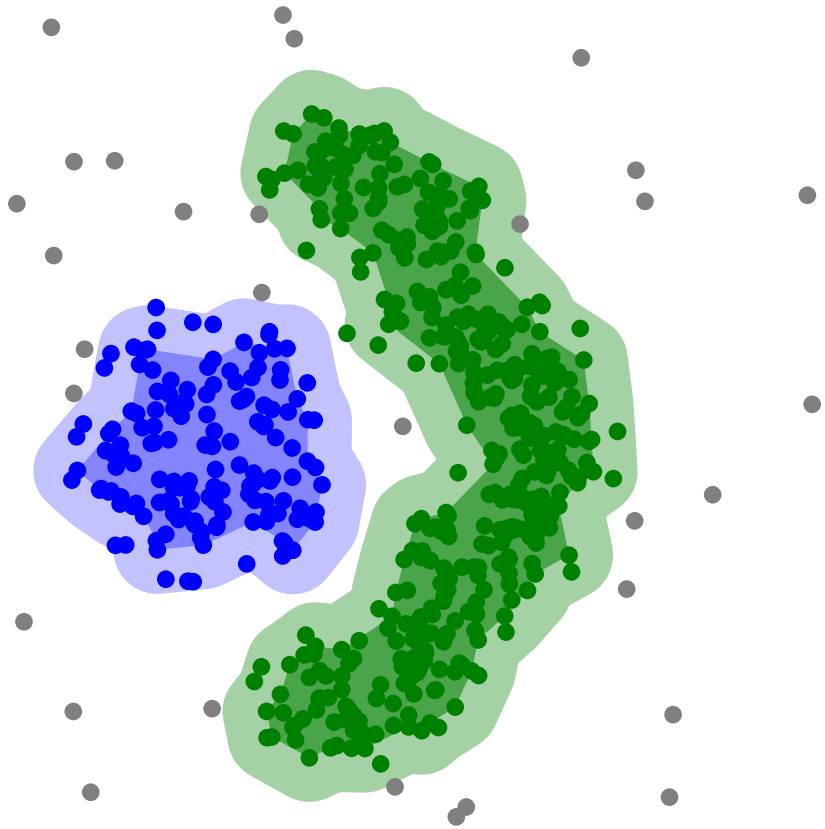



Example of connectivity in DBSCAN.
Blue is a outlier or noise point, Red are core points, and yellow are reachable points from point A.
Here `minPts` = 4

# Basic Algorithm

1. Find points in the $\epsilon$ neighborhood of every points, and identify core points with more than minPts neighbors.
2. Find the connected components of core points in the neighborhood graph, ignoring non-core points.
3. Assign each non-core point to a nearby cluster at $\epsilon$ distance. Any point not assigned in this step is assumed to be a noise point (outliers).

# Example Non-Linear Clusters



# DBSCAN Hyper-parameters

DBSCAN requires two hyper-parameters to be set:

Distance Threshold $\epsilon$ Distance to define connectedness between points. It can be obtained from a k-distance graph, plotting the distance to $k = \texttt{minPts} - 1$ nearest neighbors, ordered by largest distance. If $\epsilon$ is too small, many points will be considered noise, and if $\epsilon$ is large, then DBSCAN will produce a single large cluster.

# DBSCAN Hyper-parameters

**Minimum Number of Points `minPts`** Minimum number of points to define a core point. This value should be chosen to be at least 3, and a good heuristic is that $\texttt{minPts} \geq d + 1$, where $d$ is the dimensionality of the input features. Larger values of `minPts` are preferred as it produces stable clusters, and another heuristic is to set $\texttt{minPts} = 2d$.

# Sorted K-Distance Graph/Plot

To determine a good value for $\epsilon$, the authors of DBSCAN proposed the following algorithm. Define a function k-dist that maps each point to the distance from its k-nearest neighbor. Sorting points by descending order of their k-distance, the following is obtained:

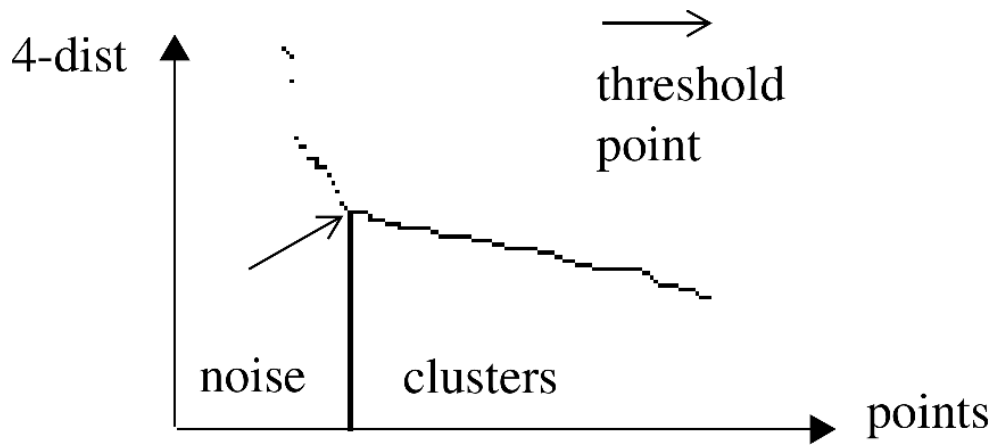



Figure: From the original DBSCAN paper, a good value of $\epsilon$ is marked with an arrow, generally corresponding to the *elbow* in the k-distance plot.

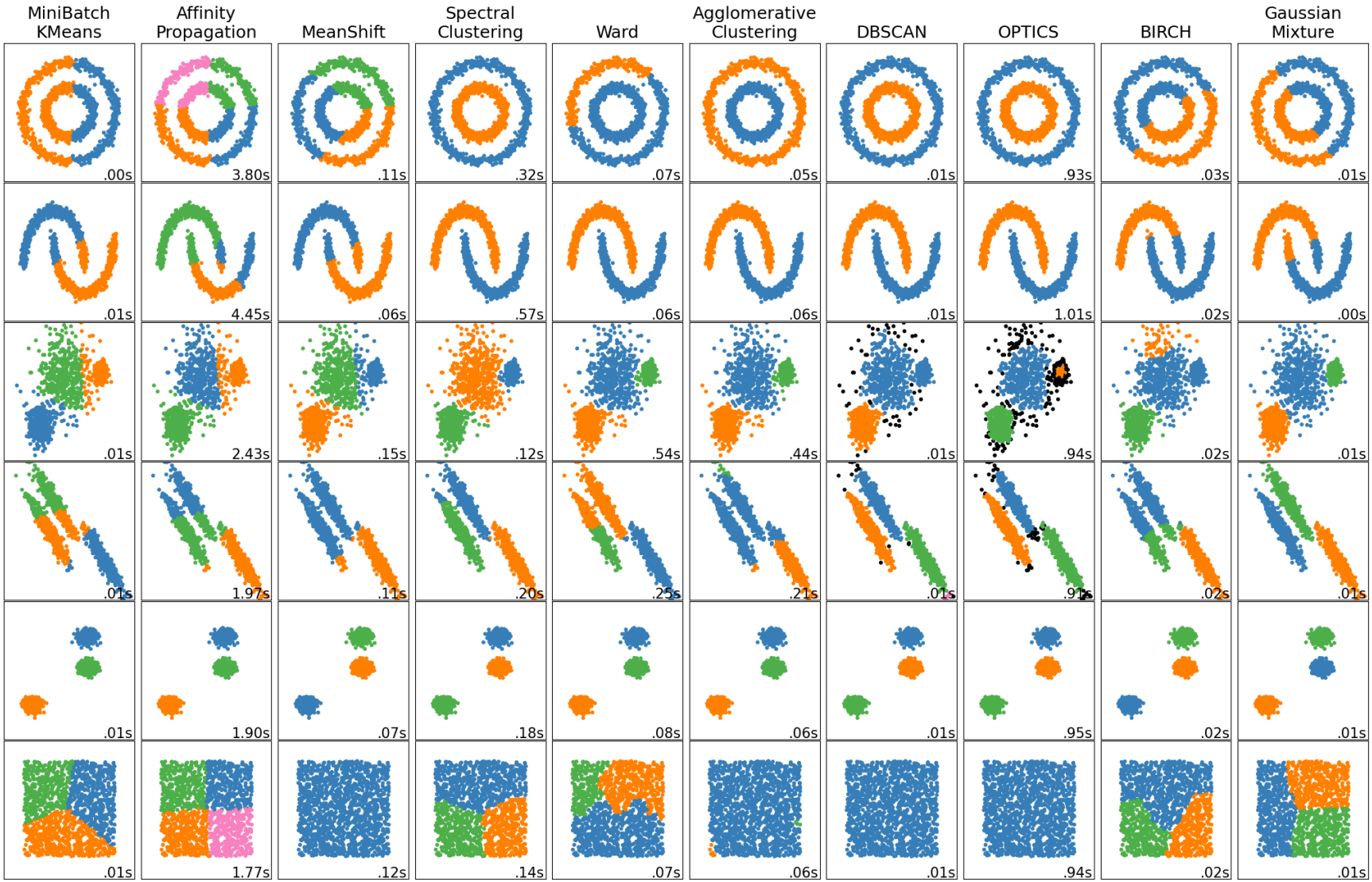
# DBSCAN Advantages

1. Does not need to specify the number of clusters, but it needs two other parameters that implicitly define the number of clusters (but are more intuitive).
2. Can find clusters with arbitrary shapes.
3. Incorporates explicit modeling of noise and outliers.
4. Can be accelerated with distance databases and pre-computer distances, for example, a KD-Tree, Octrees, or Binary Space Partitioning.

# DBSCAN Disadvantages

1. Two hyperparameters to tune, which are not interpretable from a human perspective.
2. Makes different assumptions compared to k-means, which maybe good or bad, depending on data.
3. Has an explicit definition of an outlier, which can make mistakes (classify as outlier what is not an outlier).

# Clustering Algorithms Comparison



# What you Must Learn

- Dimensionality reduction: what it is and why to use it.
  - PCA: know well; t-SNE, MDS: know the basics & intuitions
  - PCA vs t-SNE vs MDS?
- Clustering: what it is and why to use it.
  - K-Means and DBSCAN
  - (Dis)advantages of the two methods

# Book Chapter Readings

If you want to dive **deeper** in this topic, we recommend:

- Bishop Book: Chapter 1.3 for curse of dimensionality, Chapter 9.1 for k-means clustering.

# Questions to Think About

1. What is the difference between PCA and t-SNE?
2. How to determine the target dimension for PCA?
3. How do you decide how many clusters are optimal for your application?
4. On what type of data distribution is DBSCAN outperforming k-means?
5. Does changing the distance function in a clustering method change the clusters? (Assuming the same number of clusters).

# Take Home Messages

- Increasing the dimensions does not always lead to a higher performance!
- Unsupervised learning is hard and often hyperparameters and choice of methods must be done by hand:
  - Some problems are solved through trial and error. Attempt to cluster using different distance metrics and inspect the results.