Student job offers:
https://www.tu.berlin/cpsme/ueber-uns/aktuelles-und-jobs

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Applied Machine Learning in Engineering

Lecture 04 summer term 2025

Prof. Merten Stender

Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

www.tu.berlin/cpsme
merten.stender@tu-berlin.de

Exam Registration

- .. Now open through **MOSES**

- If you are unable to sign up (e.g., international students), please send an email to Prof. Stender to request manual registration. Be sure to include all relevant details, such as your student ID, degree program, etc.

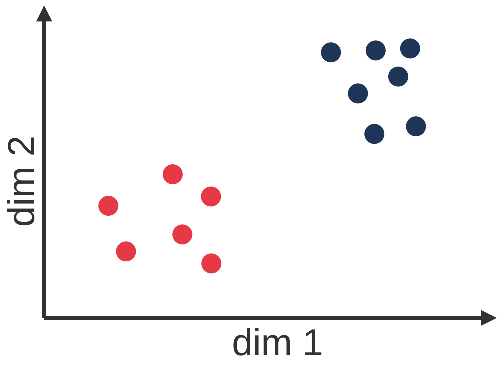
- Free to choose a date for the exam
  - If you choose the 1st date and fail, you can repeat the exam in the same period (2nd date)
  - If you choose the 2nd date and fail, you will have to wait for half a year to repeat

- You can unsubscribe from the exam registration up to 3 days before the exam through MOSES. Please check the dates and deadlines provided there. No unsubscriptions will be accepted after this period.

Recap: Lecture 03

The goodness of a clustering is in the eye of the beholder

- **Well-separated** clusters denote a situation in which any point in a cluster is closer to every other point in the cluster than to any point not belonging to the cluster



- **Objectives for cluster validity measures**:
  1. Avoid finding patterns in noise
  2. Create robust, repeatable and consistent clusterings
  3. Find a meaningful number of clusters
  4. Maximize similarity inside clusters and maximize difference between clusters

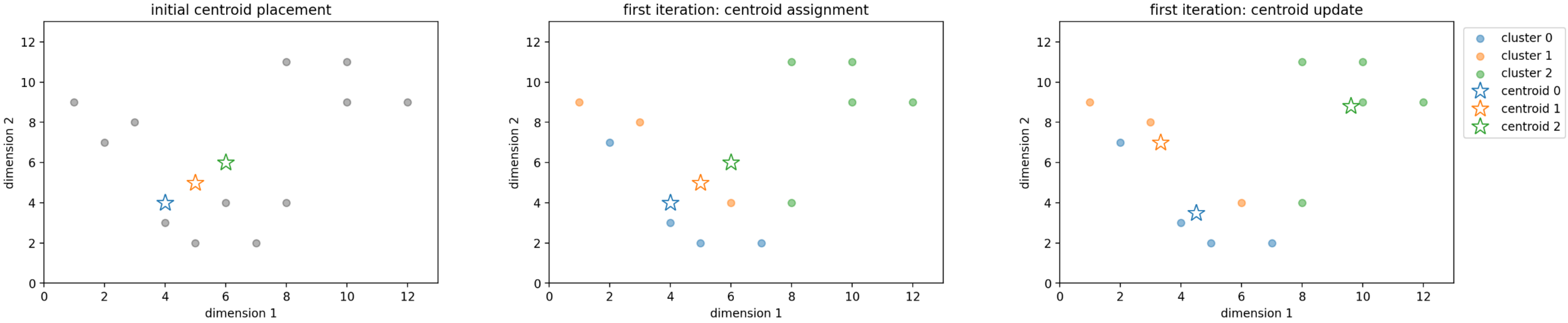
Recap: Lecture 03

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

K-means clustering algorithm: simplest and very efficient clustering algorithm

- Prototype-based, finding K (user-defined) clusters by optimal centroid placement

1. Placement of K random centroids
2. Loop until converged:
   1. Assign data points $\mathbf{x}_i$ to closest centroid $\mathbf{m}_k$ to build cluster C_k
   2. Update centroid position by averaging across $\mathbf{x}_i \in C_k$



Recap: Lecture 03

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

K-means (basic): **Pitfalls and Caveats**

- Weak convergence (trapped in local minimum)
- Strong dependence on initial centroids
- Empty clusters
- Non-deterministic results (random init. of centroids)
- User-defined selection of K
- Sensitivity to noise and outliers

Recap: Exercise 03

- Implement K-means algorithm (template provided, some lines to add)

- **assign_cluster**(x, centroids) # assigns all data points x a label given by their closest centroid
- Make **efficient use of NumPy** to keep code clean and readable
  - arg<fun> methods: return index of the desired value (i.e. minimum)
  - np.linalg. package
  - axis argument

- Basic Python:
  - any()
  - all()
  - and / or / not

```python
dists = []  # storing distances from all points to all centroids
for k in range(K):  # iterate over all K centroids

    # compute distance from all x to current centroid m_k
    dists.append(np.linalg.norm(x-centroids[k, :], ord=norm_ord, axis=1))

dists = np.vstack(dists)  # shape: [K, N]

# find the row index of minimum per column, i.e. the index of the closest centroid
cluster_labels = np.argmin(dists, axis=0)  # shape: N, 1
```

Recap: Exercise 03

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- K-means clustering using **Scikit-learn**:

```
# load csv file and plot data
data = np.genfromtxt('example_data_Kmeans.csv', delimiter=',')
plot_clusters(x=data)

from sklearn.cluster import KMeans as KMeans

# obtain the clustering
kmeans = KMeans(n_clusters=4, random_state=0, n_init="auto").fit(data)
centroids_sklearn = kmeans.cluster_centers_
labels_sklearn = kmeans.labels_
```

object attributes (see sklearn docs)

object creation and initialization

model.fit() syntax of sklearn

Questions?

Agenda

- Types of clusterings and types of clusters
- DBSCAN clustering algorithm
- Data normalization
- Python: if __name__ == "__main__":

Learning outcomes

Learn to …

- Quantify cluster properties and classify types of clusterings
- Implement a density-based clustering algorithm
- Evaluate suitable clustering techniques

Know about …

- Differences between prototype-based and density-based clustering algorithms
- Application scenarios for DBSCAN
- Importance of data normalization

Types of Clusterings and Clusters

Types of Clusterings

Clustering = the entity of clusters = the overall result of a clustering process

Three dimensions/characteristics of different clusterings (→ how to compare clusterings?)

1. **Nesting**

2. **Exclusiveness**

3. **Completeness**

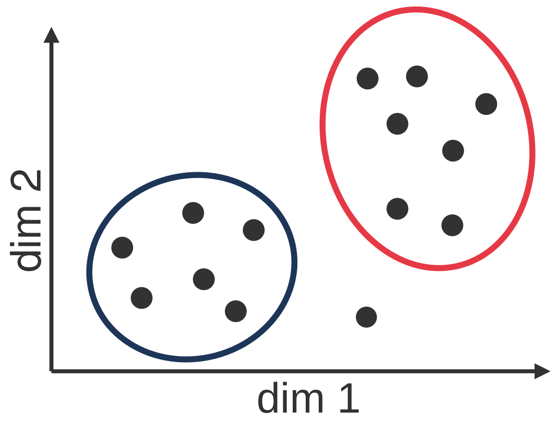
Types of Clusterings: Nesting

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Partitional clusterings**
  - Non-overlapping clusters, mutually exclusive labels
  - Each labeled data point belongs to exactly one cluster

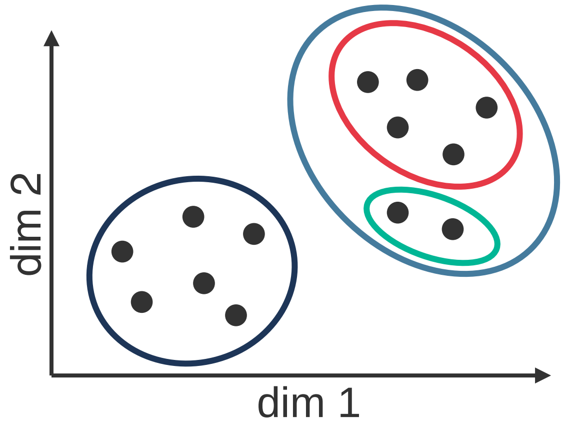
  → not every point requires a cluster assignment

  - Example: animals → dogs, cats, horses



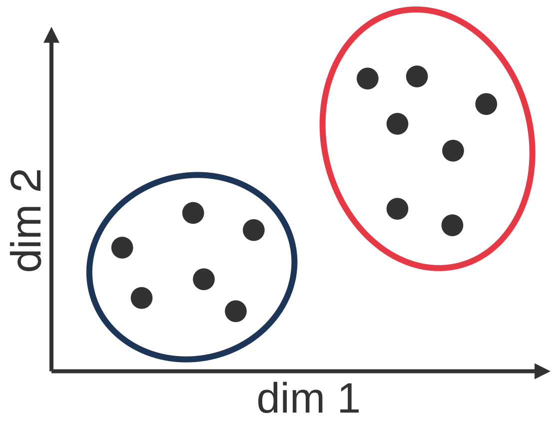
- **Hierarchical clusterings**
  - Nested clusters with subclusters
  - A data point can belong to multiple clusters across levels

  - Example: cars → sedan | SUV | sportscar

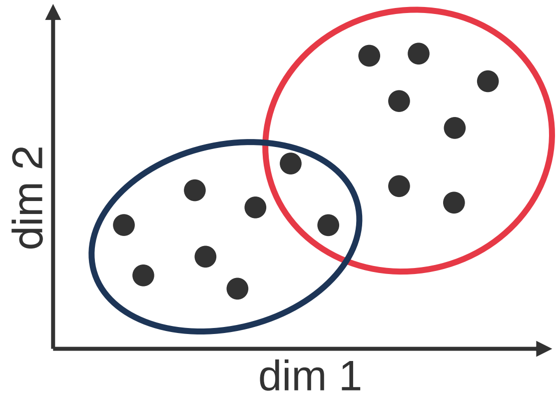


Types of Clusterings: Exclusiveness

- **Exclusive clusterings**
  - Each data point is assigned to a single cluster
  - No data point remains without a cluster label
  - Example: animals → dogs, cats, horses

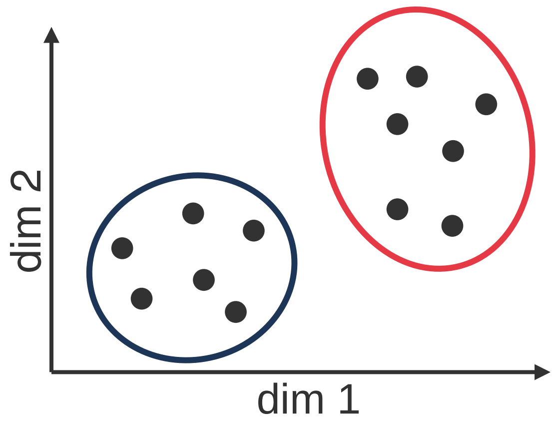


- **Overlapping (non-exclusive) clusterings**
  - Allow data points to belong to more than one cluster
  - Overlap can be hierarchical, but not necessarily
  - Example: students in double-degree programs

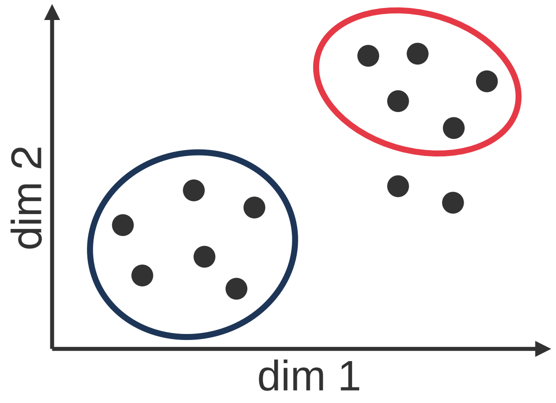


Types of Clusterings: Completeness

- **Complete clusterings**
  - Each data point is assigned to one or more cluster(s)
  - No data point remains without a cluster label



- **Incomplete clusterings**
  - Not every data point is assigned a cluster label
  - Data points without label represent outliers or noise



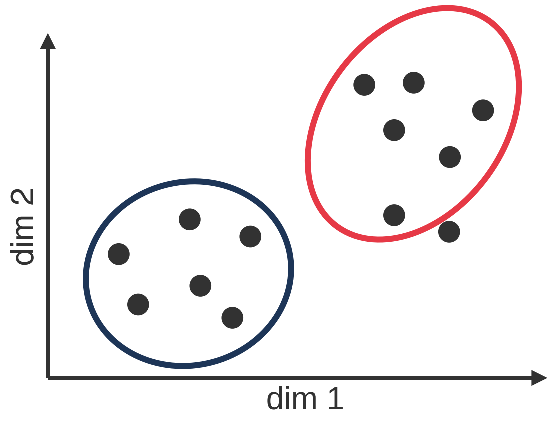
Quizz

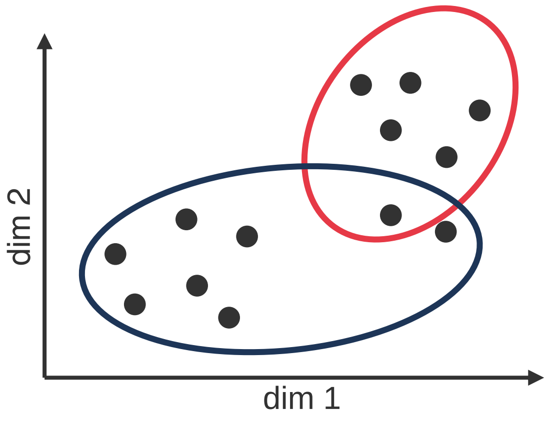
1. Illustrate a partitional, incomplete clustering

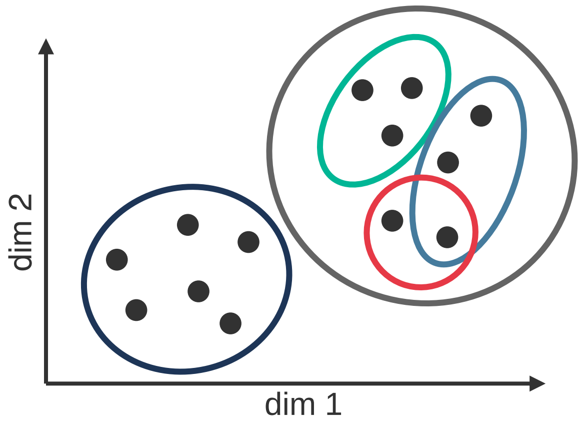
2. Illustrate a non-exclusive complete clustering

3. Illustrate a hierarchical overlapping complete clustering

Solution







Clusters

Cluster = a set of points assigned to a common group

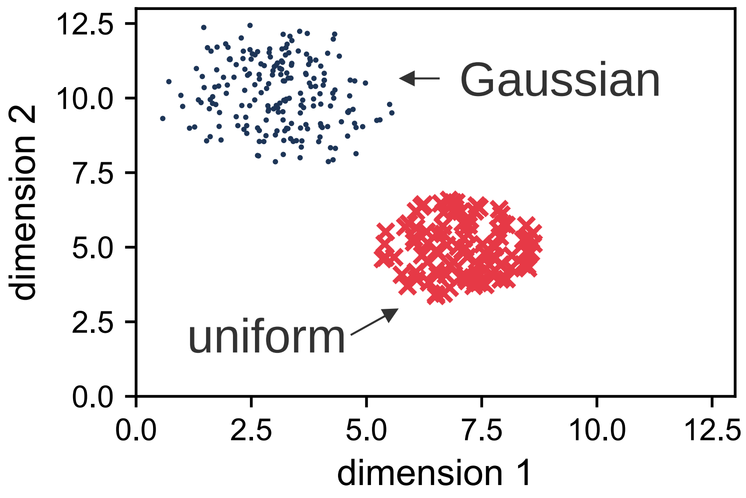
- Clusters have different characteristic properties, also called types:

  1. **Distribution**
  2. **Density**
  3. **Size or variance**

Characteristics of Clusters: Distribution

- Distribution of points within a cluster. Examples:
  - Gaussian distribution
  - Uniform distribution

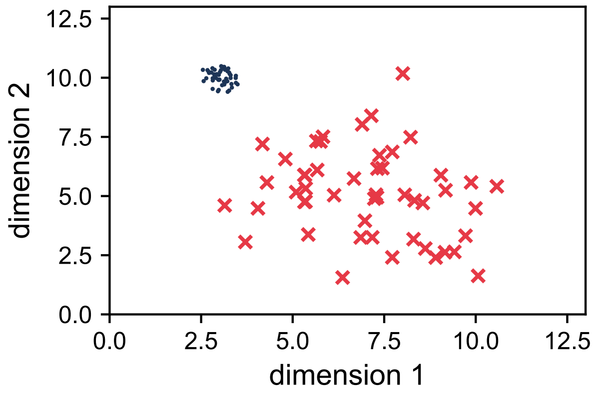
- Clusters that follow some distribution can be represented by **prototypes (‚centroids‘)** that meaningfully describe the cluster, such as the <u>average</u> of all points in a <u>normally distributed</u> cluster



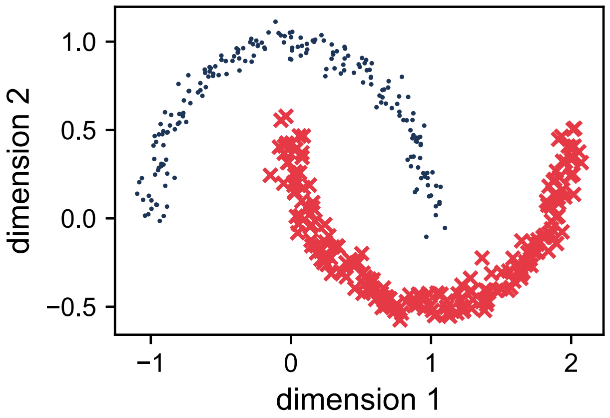
Note that any real-world data will only approximately reveal some theoretical distribution.

Characteristics of Clusters: Density

Density: the form of a cluster is given by a high-density region surrounded by a low-density region of data points



- Examples:
  - Circular / annular shapes
  - Any complex shape
  - Entangled structures

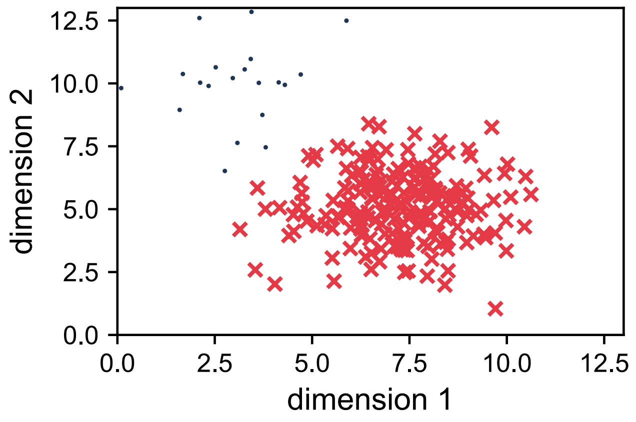


- Density-based clusters can, typically, not be represented by prototypes such as centroids!

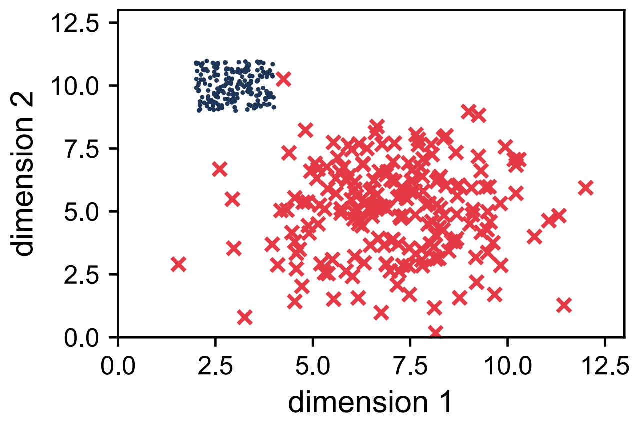
Characteristics of Clusters: Size / Variance

Size:

- Number of samples per cluster
- Expansion (hypervolume) w.r.t. to the data range and other clusters

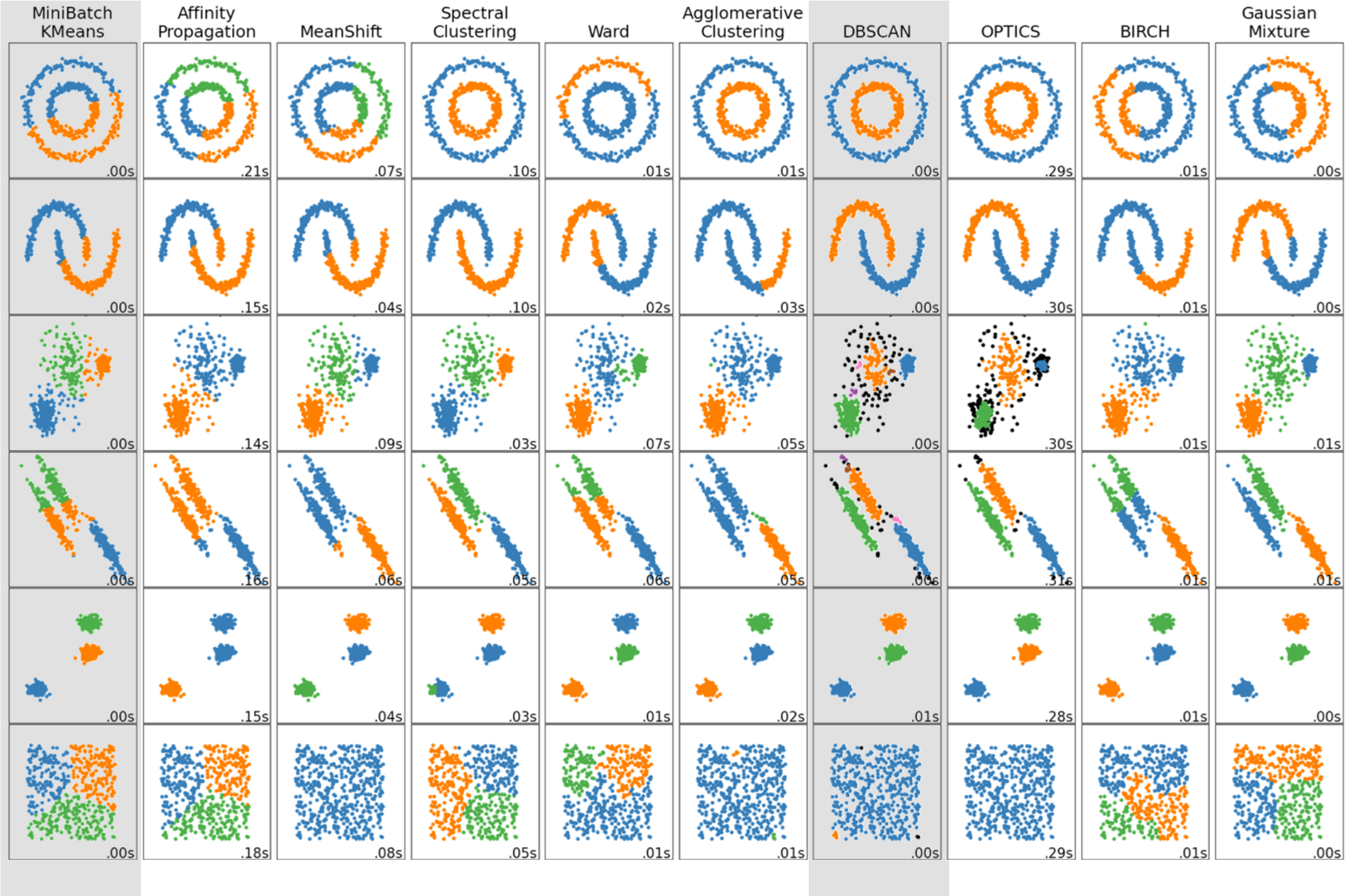


- Clusters can be large or small compared to the entity of clusters and the data range
- Small clusters may be prone to being assigned to larger clusters



Overview on Clustering Techniques

Cyber-Physical Systems
in Mechanical Engineering TU Berlin



- From scikit-learn (link)
- Many more algorithms available
- **Know your data distribution before using a clustering algorithm!**

Density-based clustering

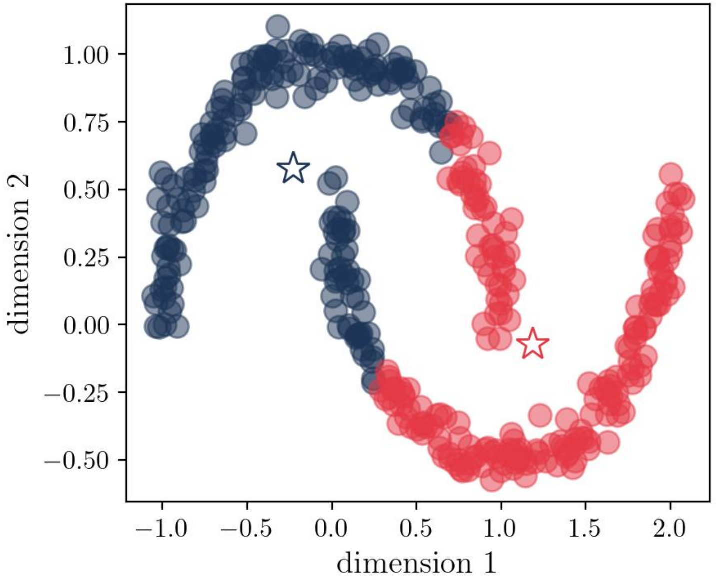
DBSCAN

Density-based Clustering

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

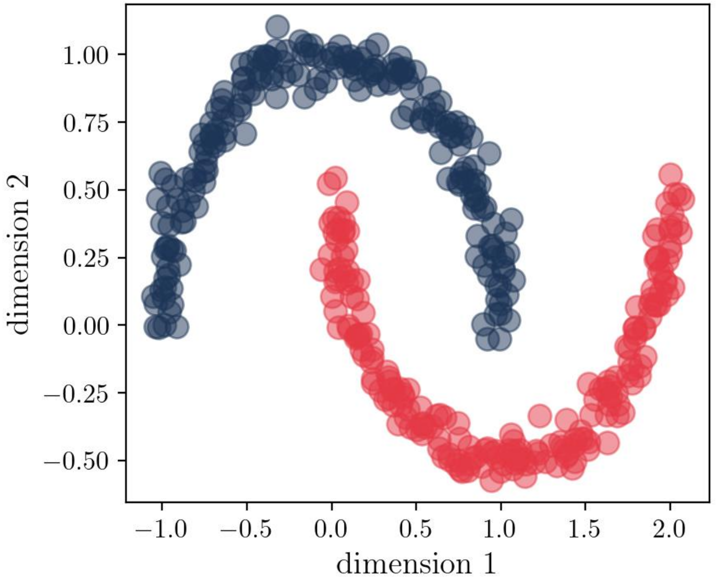
K-means

- Built for prototype-based clusters (globular shape)
- No outlier handling (exclusive and complete clustering)



DBSCAN*

- Built for density-based clusters (any shape)
- Allows for incomplete clusterings (outliers without cluster assignment)

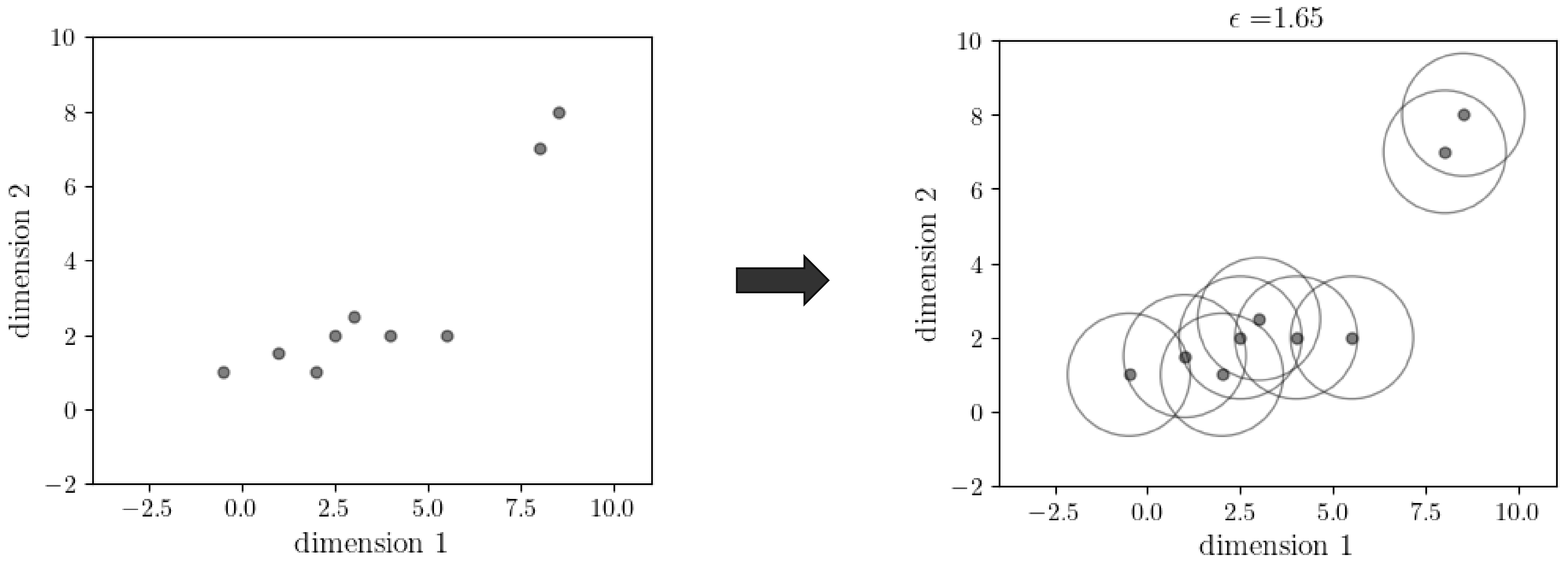


* **D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise

Encoding Density

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

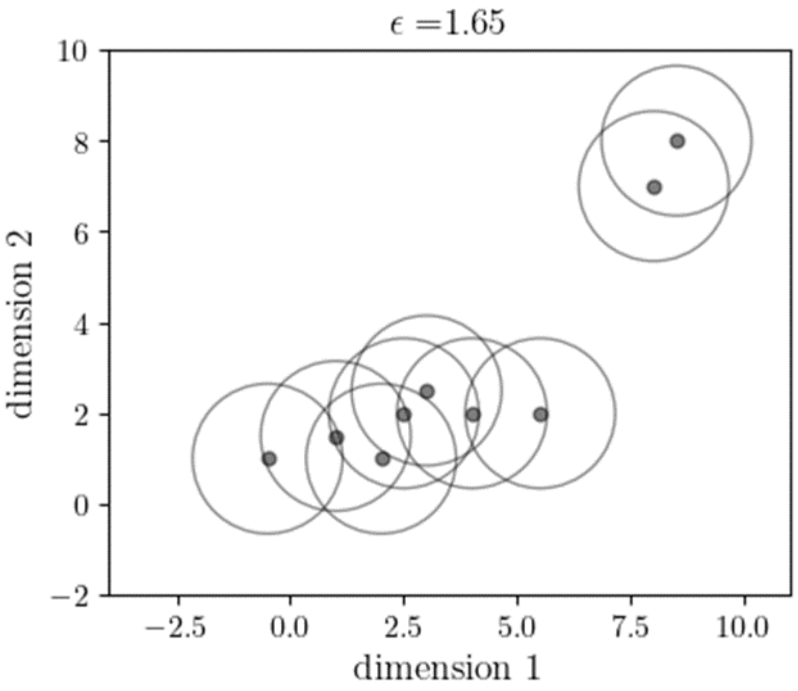
- Reachability ≈ we can jump from point to point by max. ϵ stepsize in a given norm

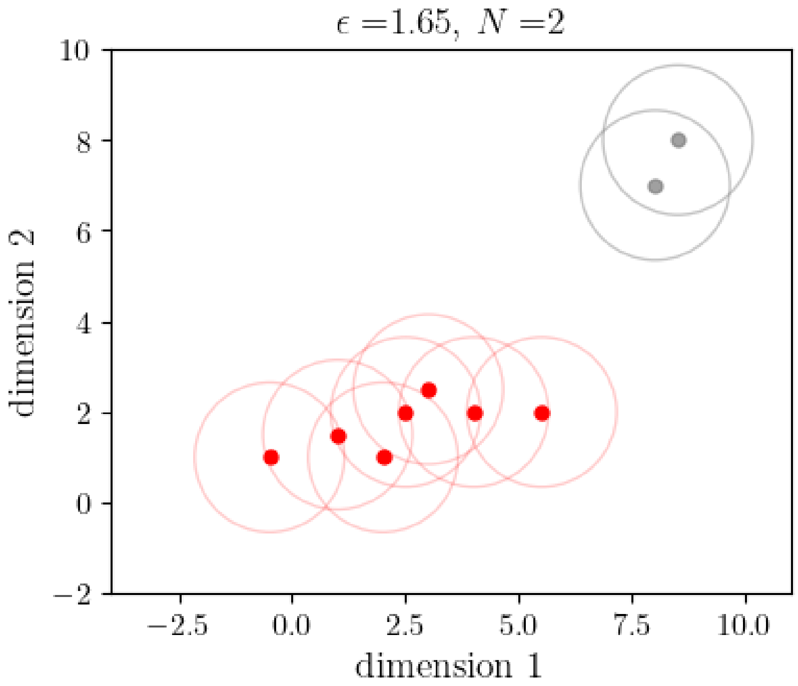


- Density-based clustering: ϵ **neighborhood**

Handling Outliers

- Outliers: single / few data points. Define a minimum number of points required per cluster $N_{\min}$

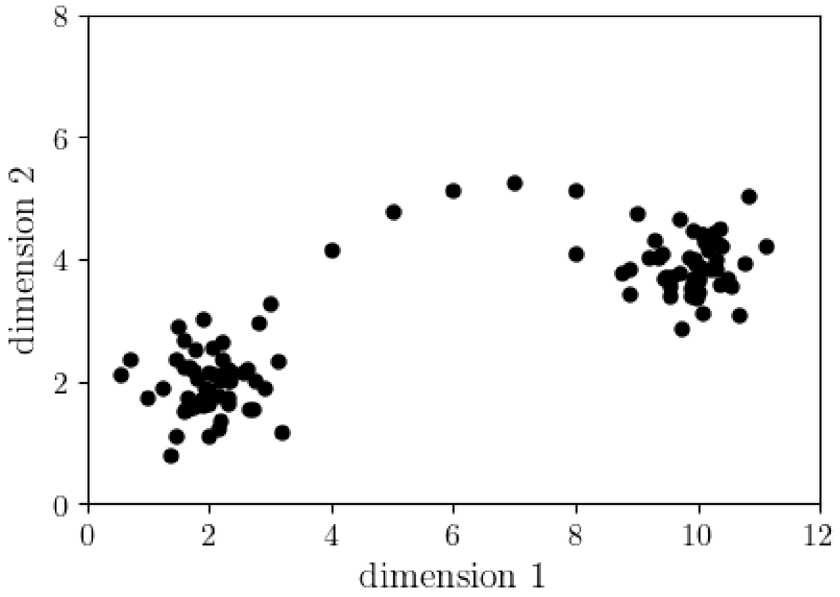


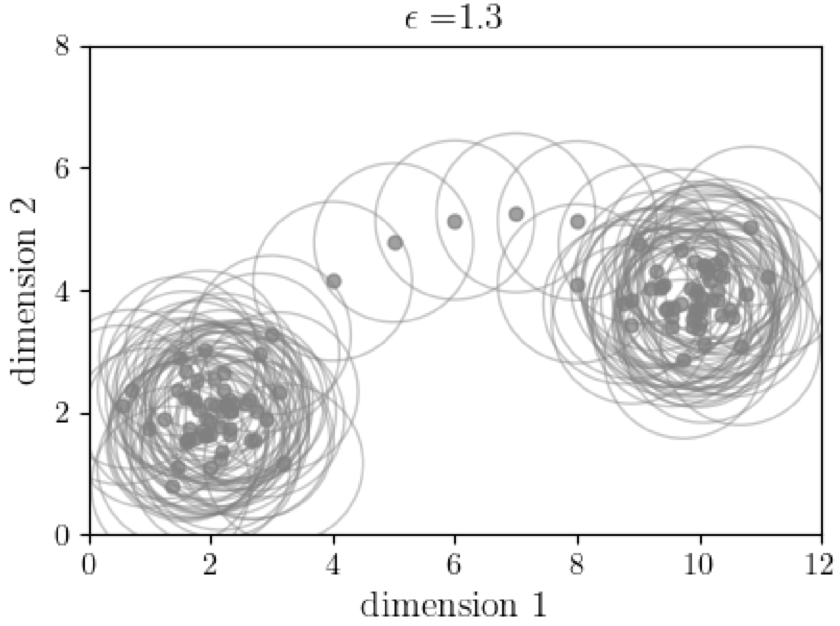


- Density-based clustering: ϵ neighborhood + $\boldsymbol{N_{\min}}$

Avoiding Single Link Effect

- For a certain ϵ value few points on a line could now link two clusters





- Density-based clustering: ϵ neighborhood + $N_{\min}$ + **minimal number of points within ϵ**

DBSCAN: Definition

- DBSCAN builds on 3 different types of data points:

- **Core point $\mathbf{x}_{\mathrm{core}}$** has at least N_{min} points within ϵ neighborhood (<u>incl. itself</u>). Interior of a cluster

- **Edge point $\mathbf{x}_{\mathrm{edge}}$** is reachable from a core point within ϵ, but is not a core point. Edge of a cluster

- **Outliers $\mathbf{x}_{\mathrm{outlier}}$** is no core point and is not ϵ -reachable from any core point. Not a cluster member

DBSCAN: Algorithm

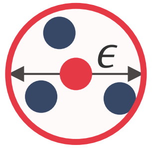
1. Find all core points in the data set
2. Start with one core point to start the cluster C_k, $k = 0$

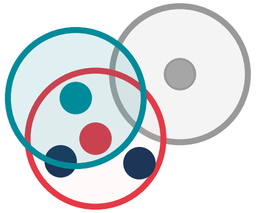
   a. Expand cluster by all ϵ-reachable core points until no core point is in reach from cluster C_k [cluster C_k contains only core points at this moment]

   b. Assign all ϵ-reachable edge points to cluster C_0

   c. Increment k and repeat a. to c.

3. Label all remaining points as outliers





DBSCAN: Pseudo-Code

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Basic implementation

Algorithm 2 DBSCAN algorithm

Find all core points within the data set, set $i = 0$
while unlabeled core points exist **do**
    Increment cluster counter $i += 1$
    Assign arbitrary unlabeled core point to cluster C_i
    while unlabeled core points are directly reachable from cluster C_i **do**
        Expand cluster C_i by directly reachable core points
    end while ▹ Cluster contains only core points up to here
    Extend cluster C_i by all directly reachable unlabeled non-core points
end while
Label unassigned points as outliers ▹ We have found i clusters now

DBSCAN: Convergence

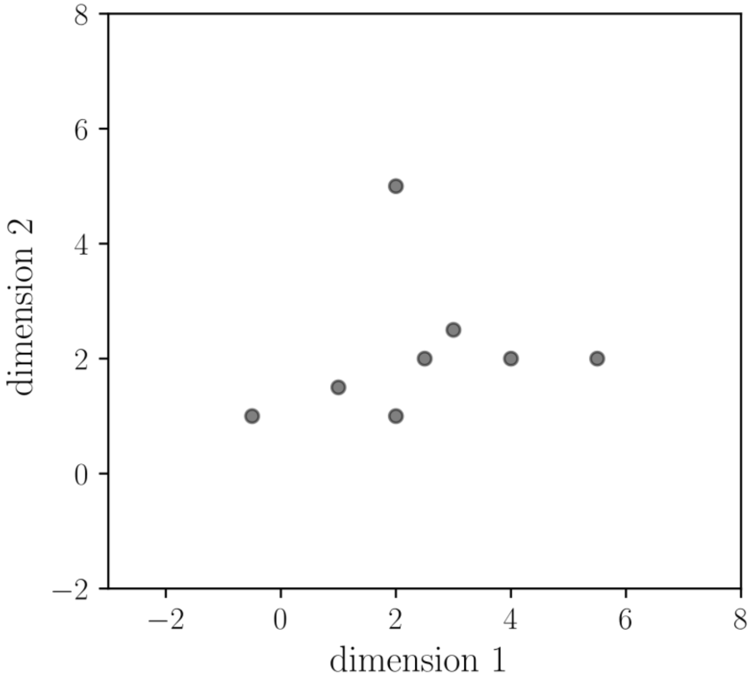
- Note: **DBSCAN is not strictly deterministic** / exactly repeatable
  - Different core point to start new cluster may result in edge points ending up in different clusters
  - Minor effect in most cases, corrections possible through extensions to basic algorithm
  - Only very weak effect of different cluster initialization

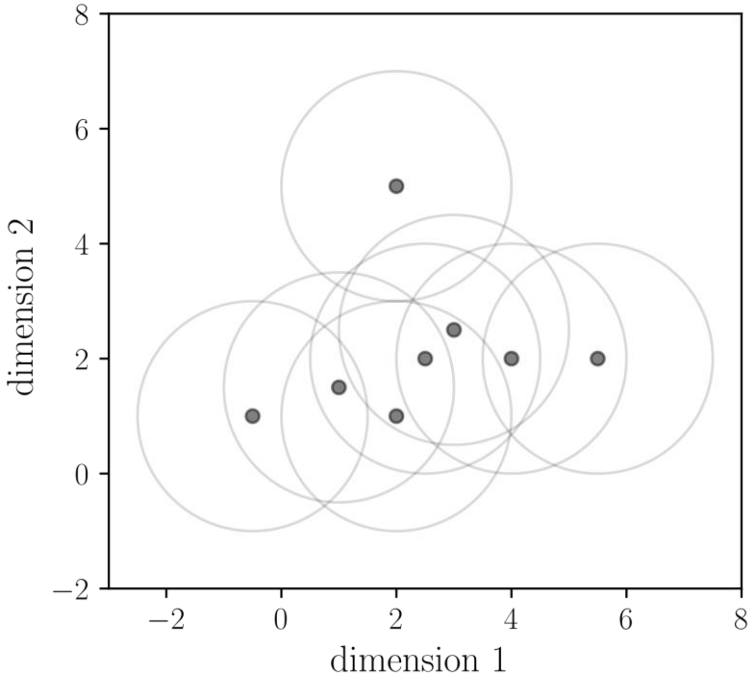
- **Computational complexity**:
  - Theoretically: $\mathrm{O}(N \cdot T)$, where T is the time to find points in ϵ neighborhood
  - Worst case: $\mathrm{O}(N^2)$ (searching the complete space for neighbors).
  - Efficient search: $\mathrm{O}(N \cdot \log N)$ (using kd-trees or other neighborhood search algorithms)

DBSCAN Example

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

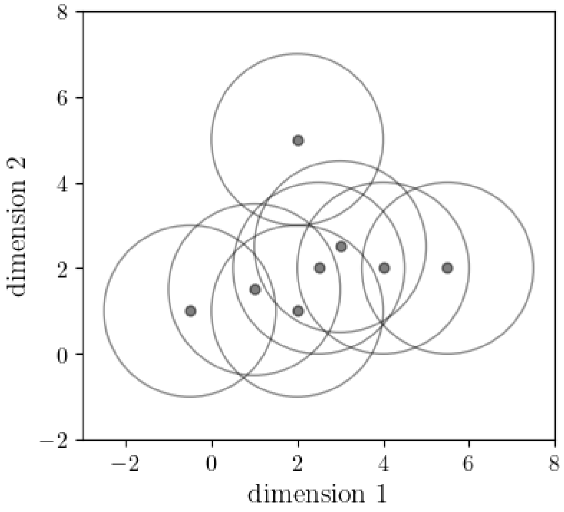
- Assign the correct type of points!
- Minimum number of points $N_{\min} = 3$, $\epsilon = 2.0$

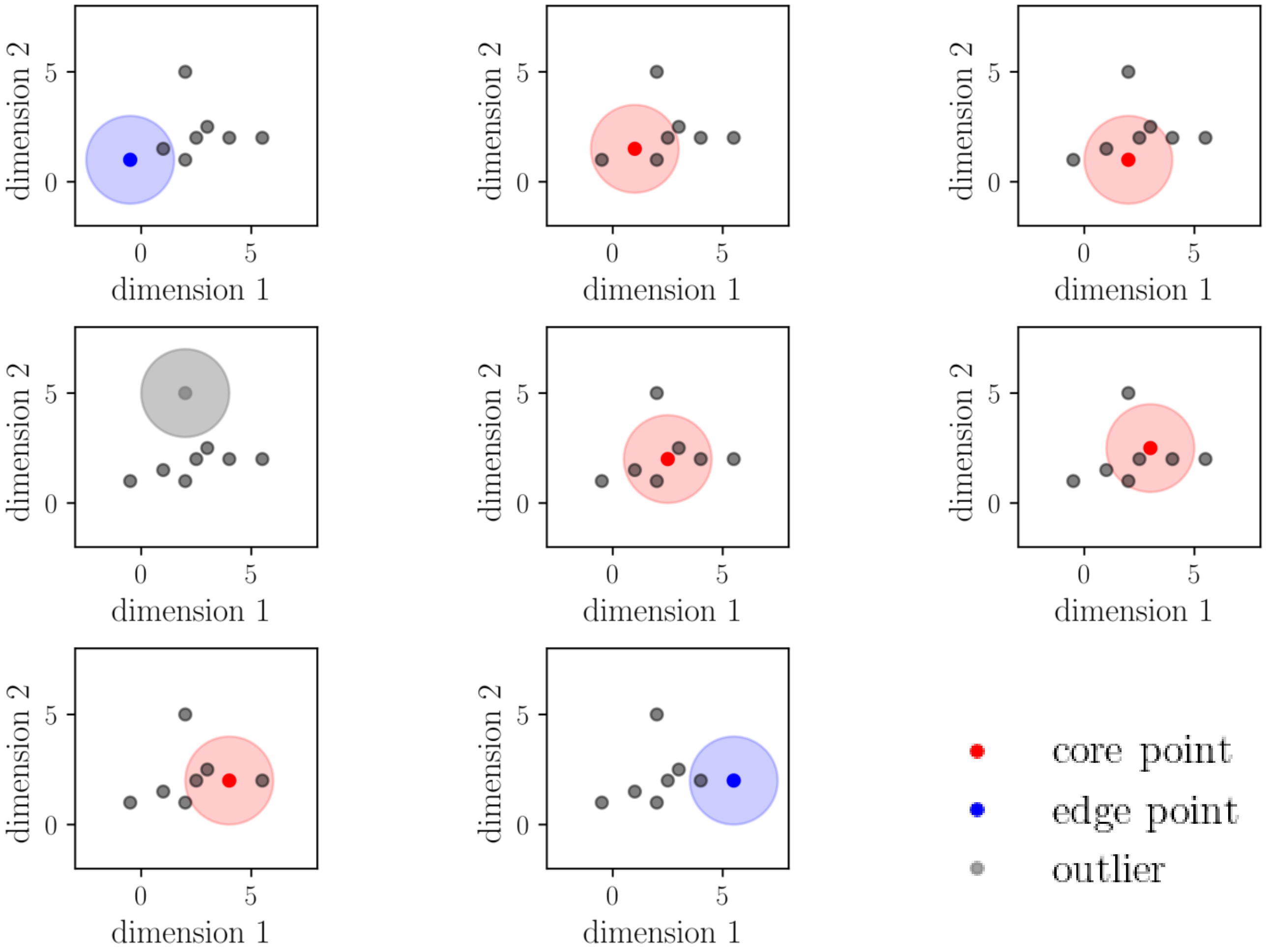




DBSCAN Example

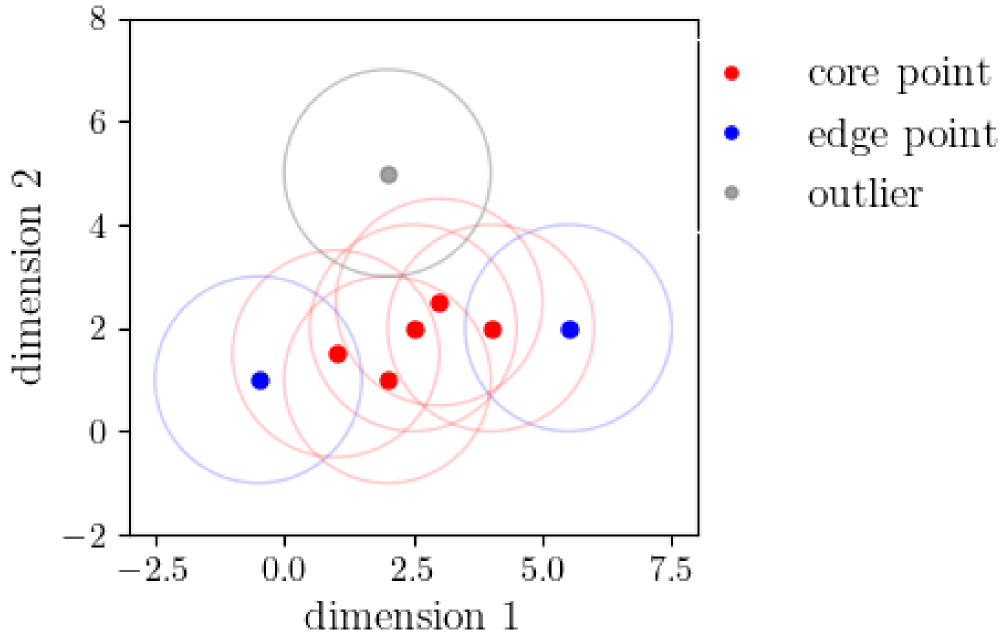
- Solution





DBSCAN: Definition

- Clusters are given by core points and their edge points
- DBSCAN determines the number of clusters itself
- Built for density-based clusters, i.e., nested or entangled clusters (globular as well)
- Incomplete clustering: outliers are identified
- Two hyperparameters: $N_{\min}$ and ϵ



Short Note: Hyperparameters

- Learnable machine learning parameters → parameters θ
- Configuration of machine learning models → **hyperparameters**

- **Examples for hyperparameters** encountered so far:
  - Choice of norm $||\cdot||$ for linear regression
  - K, centroid initialization, convergence criterion, number of repetitions for K-means clustering
  - ϵ and N_{min} for DBSCAN

- A set of learnable parameter values forms an **actual realization** of an algorithm configured by a set of hyperparameters.
- Repetitive fitting / different training data → different model parameter values for the same hyperparameters
- scikit-learn: hyperparameters returned by method get_params()

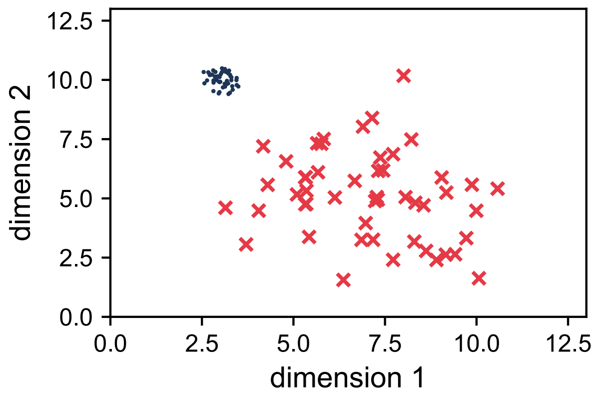
DBSCAN: Choosing Hyperparameters

- How to choose ϵ and $N_{\min}$?

- **Increasing $N_{\min}$**
  - Increases robustness against outliers
  - Creates more points labeled as outliers
  - Cuts links between clusters

- **Increasing ϵ**
  - Creates larger clusters
  - Label fewer points as outliers
  - Creates larger (and fewer) clusters

- Finding a meaningful clustering is thus a tradeoff between a) many dense and b) fewer clusters of lower density.
- DBSCAN does not perform well on clusters of very different density: fixed combination of both hyperparameters never optimal



DBSCAN: Sensitivity to Data Ranges

- Multidimensional data sets store attributes related to different quantities
- Different quantities come with different value ranges and underlying distributions
- Example: simplistic description of cars
  - Number of seats: 2 – 5 range: 3
  - Horsepower: 80 – 400 hp range: 320 hp
  - Maximum speed: 140 – 220 km/h range: 80 km/h
  - Weight: 1000 – 2000 kg range: 1000 kg

- Distance metric $\|\cdot\| < \epsilon$ in the n-dimensional feature space: one scale for all dimensions
  - Strongest weighting of **largest absolute variance / range**

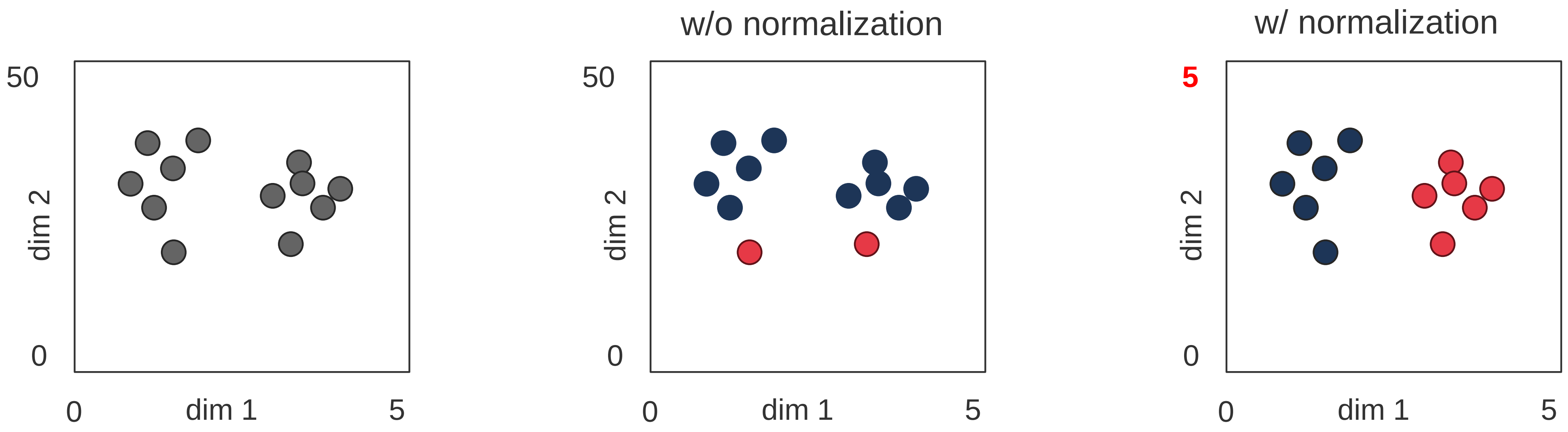
- How to select an ϵ-neighborhood for comparing $\begin{bmatrix}1600\\140\end{bmatrix}$ and $\begin{bmatrix}1200\\180\end{bmatrix}\begin{bmatrix}\mathrm{kg}\\\mathrm{km/h}\end{bmatrix}$? $\Delta = \begin{bmatrix}400\\40\end{bmatrix}$?

DBSCAN: Data Normalization Requirement

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- In order to find core and edge points in n-dimensional feature spaces w.r.t. to some distance metric $||\cdot|| < \epsilon$, the data has to have the same **dynamic range** across all feature dimensions!

→ Data normalization is key to any (especially density-based) clustering algorithm

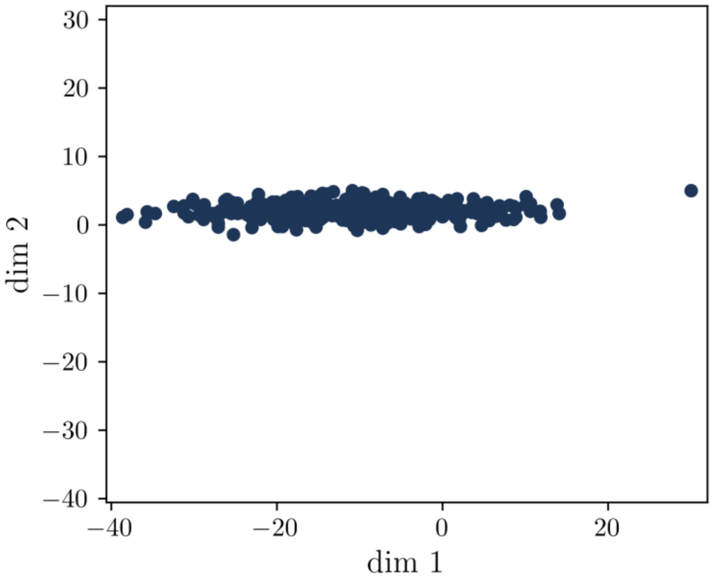


Data Normalization

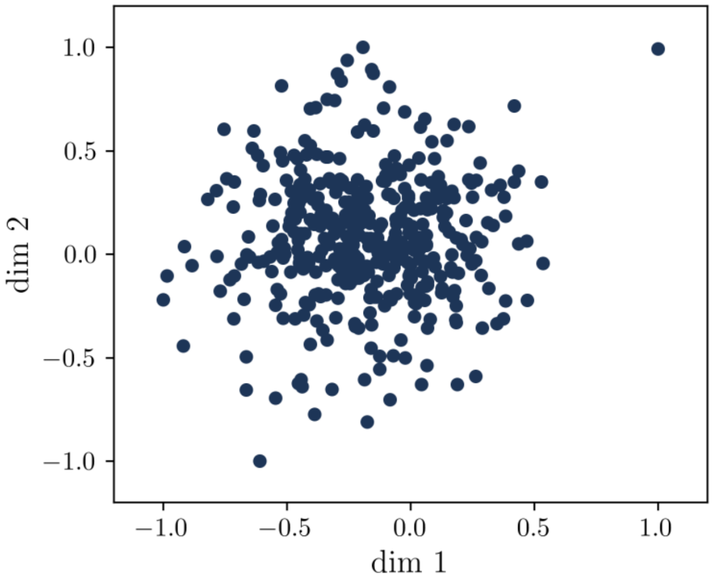
Data Normalization

- **Linear rescaling** values from a range $[\min(x), \max(x)]$ to $[a^*, b^*]$, typically $[0, 1]$ or $[-1, 1]$
  - Sensitive to outliers and extreme values
  - No outlier removal or assumption about underlying distribution

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \cdot (b^* - a^*) + a^*$$



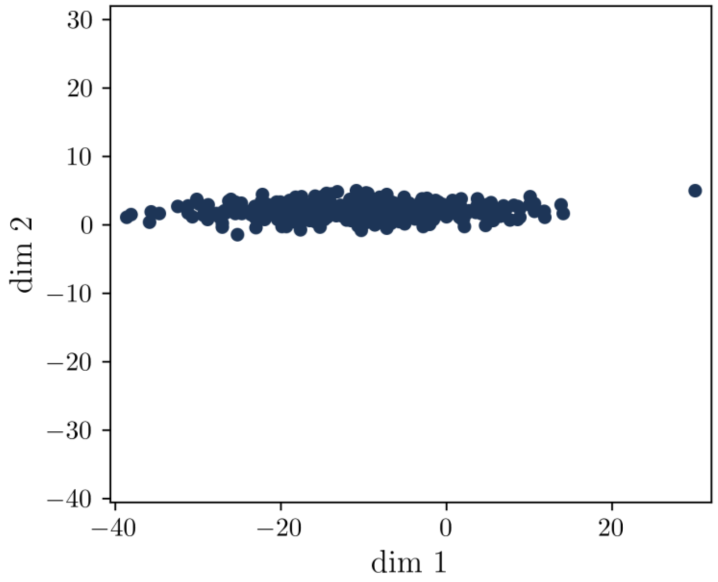
range $[-1, 1]$



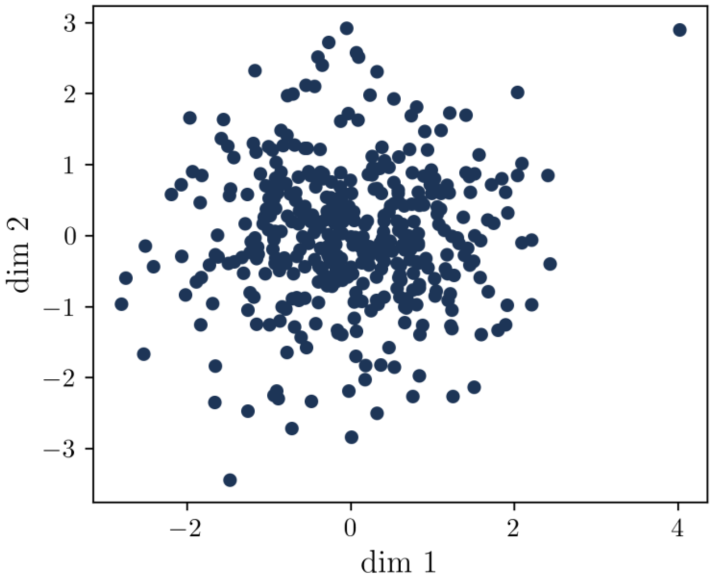
Data Normalization

- **Z-scoring** (scikit-learn: StandardScaler())
  - Centering the data (removing the mean)
  - Some robustness against outliers
  - No matching value range guaranteed across dimensions

$$\tilde{\mathbf{x}} = \underbrace{\frac{1}{\sigma(\mathbf{x})}}_{\text{unit standard deviation}} \cdot \underbrace{\left(\mathbf{x} - \frac{1}{N}\sum_{i=1}^{N} x_i\right)}_{\text{zero mean}}$$



z-scoring



Data Normalization

- Many more ways to normalize or scale data, particularly for **noisy data**
- **Noisy data** (having outliers) can skew the normalization, as the maximum range value is corrupted

- **Robust scaling:** variant of min-max scaler with offset removal. Using the interquartile range Q as reference value for the normalization. Implicit assumption about normal-like distribution.

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - Q2}{(Q3 - Q1)}, \quad \mathbf{x} \in \mathbb{R}^{N,1}$$

- **Log scaling**: rescaling for data with outliers, rescaling for data spanning multiple orders of magnitude. Note that this transformation changes the underlying data distribution, not working for negative values.

$$\tilde{\mathbf{x}} = \log(\mathbf{x} + \gamma), \quad \gamma = \min([\min(\mathbf{x}), 0]), \quad \mathbf{x} \in \mathbb{R}^{N,1}$$

Python: main idiom

if __name__ == "__main__„:

The main function

- The core body of a program (the main) should be a function
  - to store code that should only run when your file is executed as a script
  - to allow importing functions in a different module without running the main code
  - to allow for testing your functions

- **if __name__ == "__main__"** idiom

echo.py

```
def echo(text: str, repetitions: int = 5) -> str:
    """Imitate a real-world echo."""
    echoed_text = ""
    for i in range(repetitions, 0, -1):
        echoed_text += f"{text[-i:]}\n"
    return f"{echoed_text.lower()}."


if __name__ == "__main__":
    text = 'hello'
    print(echo(text))
```

running this will echo the world „hello“

some_module.py

```
from echo import echo

print(echo('echo'))
```

running this will echo the world „echo“.

without the __main__ idiom in echo.py:
this module would echo „hello“ **and** „echo“

Exercise 04

Exercise 04: Task 1 (at home!)

1. **Implement a z-scoring class** with the following methods

   - Zscorer.fit(x) → estimating mean and std. deviation from data x
   - Zscorer.transform(x) → z-score data x
   - Zscorer.inverse_transform(x) → undo the z-scoring
   - Should work for any data dimension n
   - Validate against scikit-learn using a synthetic data set

Exercise 04: Task 2

- Find clusters in second-hand car sales data (from UK, 2000-2024)
- Load data, extract information on year, mileage and price
- Find clusters using DBSCAN
- Find optimal clusters using a grid search for the hyperparameter ϵ
- Visualize the clustering results
- Interpret the results

Questions?