Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Applied Machine Learning in Engineering

Lecture 03 winter term 2023/24

Prof. Merten Stender

Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

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

Recap: Lecture 02

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

	Type	Description	Example	Operations
categorical / qualitative	**Nominal**	value corresponds to a set of mutually exclusive values, classes or categories	eye color, city name, brand name, class name, booleans	$=$ and $\neq$
	Ordinal	values from a set of distinct values, can be put into an order	house numbers, study semester, grades	$=$ and $\neq$ $(<, \leq, >, \geq)$
numeric / quantitative	**Interval**	values from a continuous set of equally spaced values, unit of measurement exists, no true zero	temperature °C, time on 12-hour clock, IQ test results	$=$ and $\neq$ $(<, \leq, >, \geq)$ $(+, -)$
	Ratio	values from a continuous set of equally spaced values, unit of measurement exists, true zero indicates absence	age, velocity, height	$=$ and $\neq$ $(<, \leq, >, \geq)$ $(+, -, *, /)$

Recap: Lecture 02

- Making qualitative (categorical) data readable to a computer

- **One-Hot Encoding** $y \in \{v_1, \ldots, v_k\} \mapsto y \in \mathbb{R}^k, \in \{0, 1\}$, k: number of distinct values / classes

- Traffic example:
  - 4 classes: pedestrian, crosswalk, stop sign, car

  - pedestrian → $[1 \quad 0 \quad 0 \quad 0]^\mathsf{T}$
  - crosswalk → $[0 \quad 1 \quad 0 \quad 0]^\mathsf{T}$
  - stop sign → $[0 \quad 0 \quad 1 \quad 0]^\mathsf{T}$
  - car → $[0 \quad 0 \quad 0 \quad 1]^\mathsf{T}$

  - Model prediction $y = [0.95 \quad 0 \quad 0 \quad 0.05]^\mathsf{T}$ → to 95% a pedestrian, to 5% a car

Do not use for ordinal data, as order gets lost!

(almost) no ML algorithm does create an implicit relationship or order between neighboring values in an output array such as a OHE vector

Recap: Lecture 02

Integer Encoding

- Encoding ordinal data requires **keeping an order**
- Simplistic encoding: assign an integer to each category, start with 0 for the first category

- **Example**: satisfaction rating for this class
  - 5 classes with natural rank order (*"extremely dislike", "dislike", "neutral", "like", "extremely like"*)
  - Integer encoding:

    extremely dislike → 0
    dislike → 1
    neutral → 2
    …

- **Caution!** Integer encoding keeps the order, but pretends a measure of <u>(equal) distances</u>!
  - Strictly speaking, a model prediction $\tilde{y} = 1.8$ is not meaningful, and rounding may be wrong
  - Decoding strategy is highly case-specific!

Recap: Exercise 02

- Dictionaries and mappings between key-value pairs
- One-hot encoding at set of categorical values

```
# obtain number of classes and classes themselves
self.classes = np.unique(values)
self.num_classes = len(self.classes)

# mapping between category (key) and index (value) via dictionary
self._class_map = dict(zip(self.classes, np.arange(stop=self.num_classes)))

# one-hot encode the categorical data
encoded_vals = []
for val in values:
    _enc_value = np.zeros(self.num_classes) # empty vector of zeros
    _enc_value[self._class_map[val]] = 1 # turn 'hot' (put in a one)
    encoded_vals.append(_enc_value) # stack to existing values
```

Recap: Exercise 02


```python
def decode(self, enc_vals):
    """ Invert one-hot encoding.
    expects a binary N x K binary matrix. N samples, K categories
    returns list or one-dimensional array of categories
    """
    # inverse mapping between index and category
    self._inv_class_map = {v: k for k, v in self._class_map.items()}

    # de-code one-hot encoded values
    values = []
    for enc_val in enc_vals:

        idx = np.argwhere(enc_val == 1)[0][0]
        values.append(self._inv_class_map[idx])

    return np.hstack(values)  # return a one-dim. np.ndarray of categories
```

```python
if __name__ == "__main__":

    """
    Testing the implementation
    """
    OHE = OneHotEncoder()

    values = np.array(['Berlin', 'Frankfurt', 'Munich', 'Berlin'])
    print(f'values to encode: \t{values}')

    # fit the encoder
    OHE.fit(values)

    ohe_values = OHE.encode(values)
    print(f'one-hot encoded representation: \n {ohe_values}')

    dec_values = OHE.decode(ohe_values)
    print(f'de-coded values: \t{dec_values} \n\n\n')
```

Agenda

- Unsupervised learning
- Cluster validity metrics
- K-means clustering
- Python: scikit-learn library

Learning outcomes

Learn to …

- Identify unsupervised learning tasks
- Quantify properties of clusters
- Implement a basic clustering method

Know about …

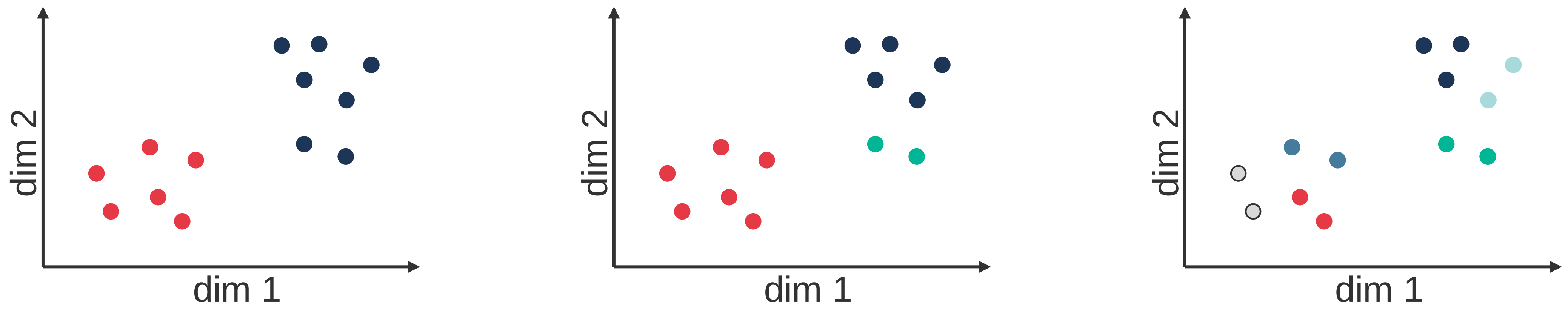
- Time complexity of K-means
- Advantages and disadvantages of K-means

Clustering

Clustering Data

Clustering denotes the process of dividing a set of data points into distinct groups (clusters), thereby **maximizing intra-group similarity** and **minimizing inter-group similarity.**

- What to consider a good clustering?

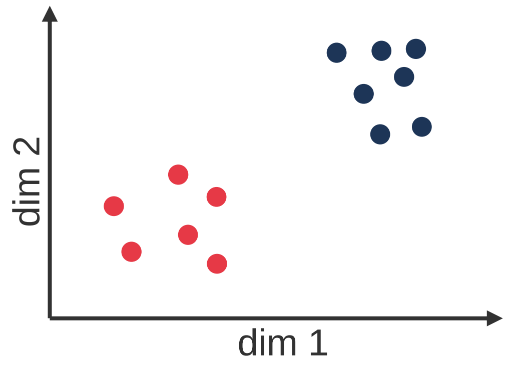


→ Measures of cluster validity

Measures of Cluster Validity

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



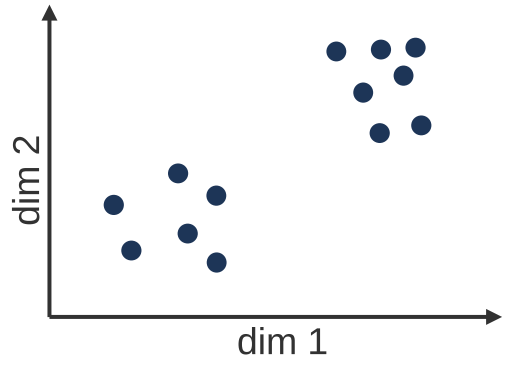
- **Objectives for cluster validity measures**:
  1. Avoid finding patterns in noise
  2. Create robust, repeatable and consistent clusterings
  3. Find a meaningful number of clusters
  4. Maximize similarity inside clusters and maximize difference between clusters

Measures of Cluster Validity

Internal measures

No a-priori information exists about clusters or class labels

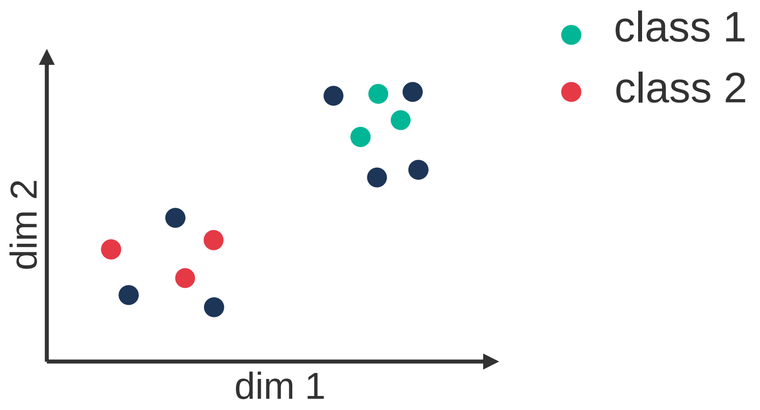
- Momentum (SSE/compactness/cohesion)
- Cluster separation
- Silhouette Coefficient
- Dunn-indx, correlation, …



External measures

Some external knowledge about clusters exist, such as class labels for some instances.

- Entropy
- Adjusted Mutual Information (AMI)
- Adjusted Rand Index (ARI), …

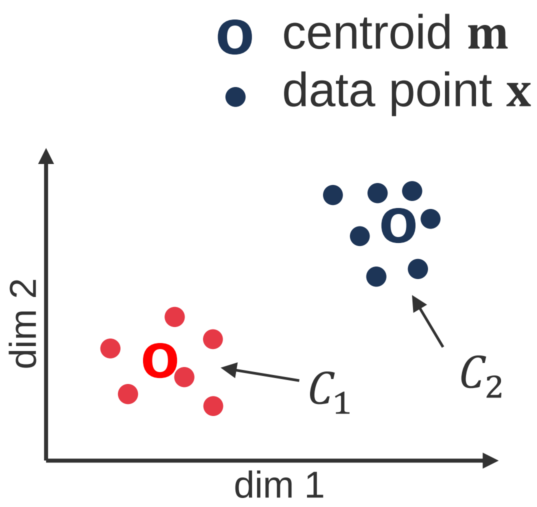


Momentum (Compactness / Cohesion)

- Internal cluster validity metric based on **within-cluster sum-of-squares**

$$\mathrm{SSE} = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{m}_k - \mathbf{x}_i\|^2$$

centroid $\mathbf{m}_k \in \mathbb{R}^n$



- Measure of the **variability** and **density** of the observations within each cluster

- **Question:** what would the clustering look like if we optimized for minimal SSE?
  - w/o constraints on the number of clusters: $SSE = 0$ for $K = N$ → undesired behavior
  - Simplest / trivial approach: set user-defined number of clusters K → ***K*-means algorithm**

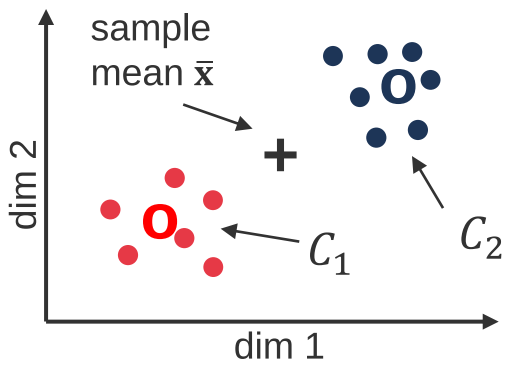
Cluster Separation

- Internal cluster validity metric based on **between cluster sum of squares**

O centroid $\mathbf{m}$
● data point $\mathbf{x}$

$$\mathrm{BSS} = \sum_{k=1}^{K} \|\mathbf{m}_k - \bar{\mathbf{x}}\|^2$$

centroid $\mathbf{m}_k \in \mathbb{R}^n$



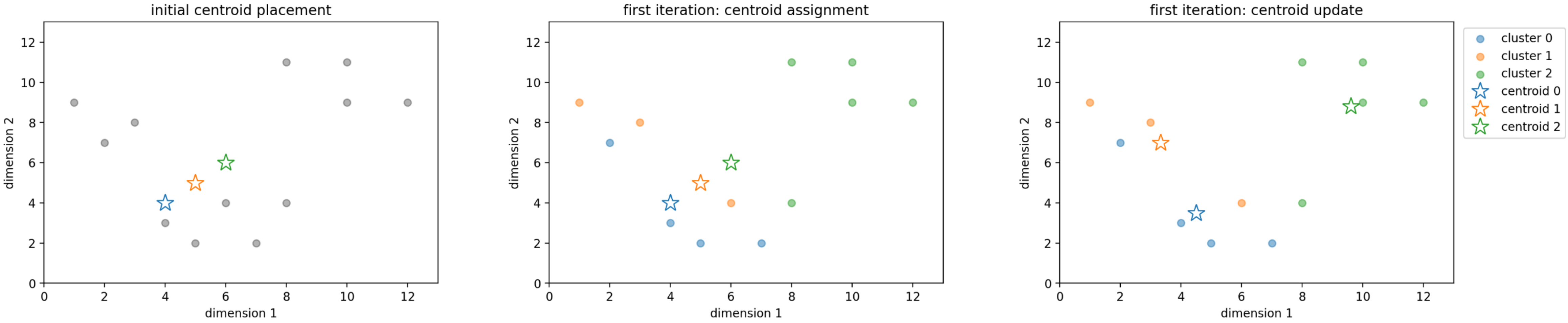
- Measure for how far centroids are spread out w.r.t. sample mean $\bar{\mathbf{x}}$
- BSS does not account for cluster size, cluster density, or well-separated clusters
- Larger BSS is better

K-means clustering

K-means: Basic 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 $\mathbf{x}_i \in C_k$



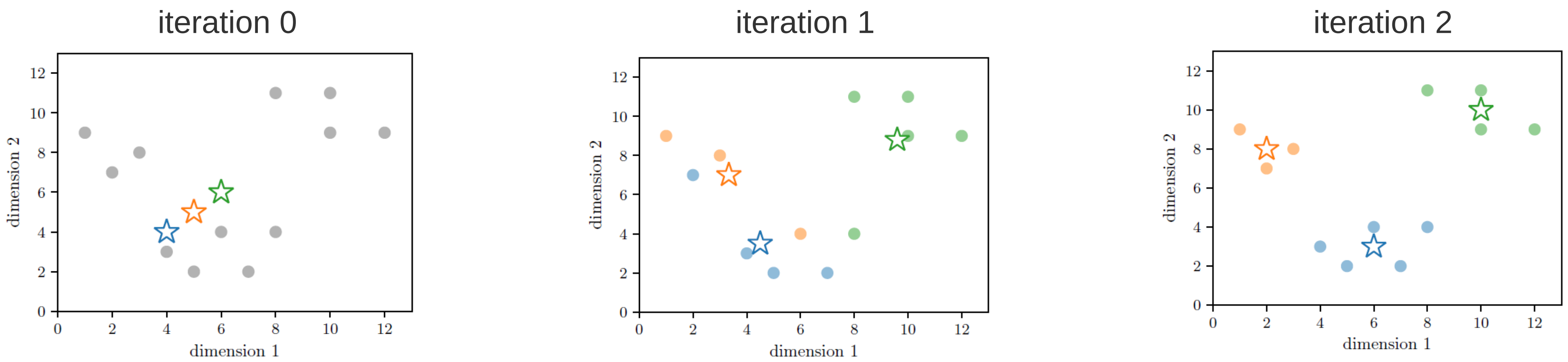
K-means: Convergence Criterion

- **When to stop updating centroid assignments and centroid updates?**

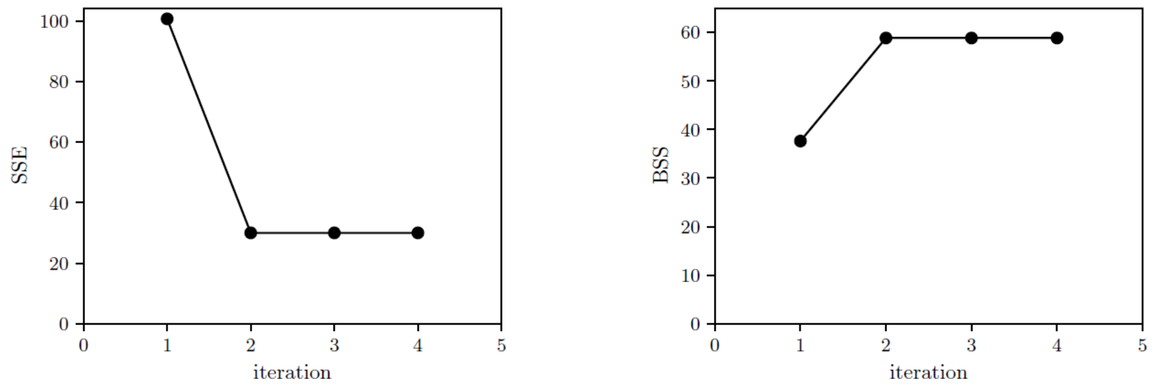
- Consider some consecutive iterations of K-means
  - How many points changed clusters?
  - How much did centroid positions change? → Frobenius norm of consecutive $\mathbf{m} = [\mathbf{m}_1, \dots, \mathbf{m}_K]$

- Typical convergence criterion: <1% points change clusters during last 2 iterations

K-means: Example



- Tracking cluster validity metrics



K-means: Complexity

- Simple and efficient algorithm

Algorithm 1 K-means algorithm (basic form)

1: Select K initial centroids $\mathbf{m}_k$
2: **repeat**
3:     **for all** $i = 1, \ldots, N$ **do** ▹ First phase: cluster assignment
4:         assign point $\mathbf{x}_i$ to closest centroid $\mathbf{m}$ and update assignment vector r_{ik}
5:     **end for**
6:     **for all** $k = 1, \ldots, K$ **do** ▹ Second phase: centroid update
7:         update centroid positions $\mathbf{m}_k$ by averaging across all $\mathbf{x}$ in cluster C_k
8:     **end for**
9: **until** clusterings converged

- Complexity: $\mathcal{O}(K \cdot N \cdot n \cdot j)$ (scales **linearly** with number of data points)
  (j: number of iterations until convergence)

K-means (basic): Pitfalls and Caveats

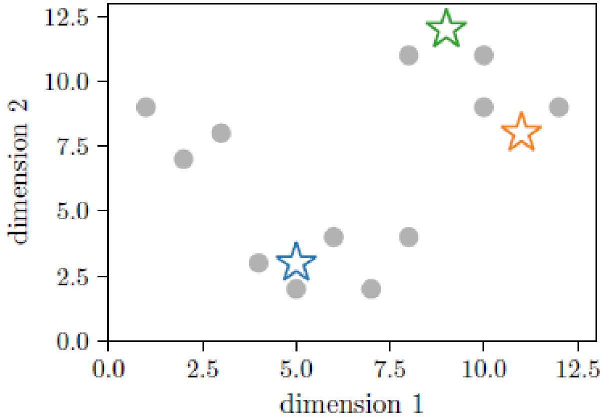
- Weak convergence (trapped in local minimum)
  - Strong dependence on initial centroids
- Empty clusters
  - Algorithm stops when a cluster is empty
- Non-deterministic results
  - For stochastic centroid initialization
- User-defined selection of K
  - Generally unknown, requires parameter studies
- Sensitivity to noise and outliers
  - Requires a-priori outlier detection or more advanced K-means algorithms

Dependence on Initial Centroids

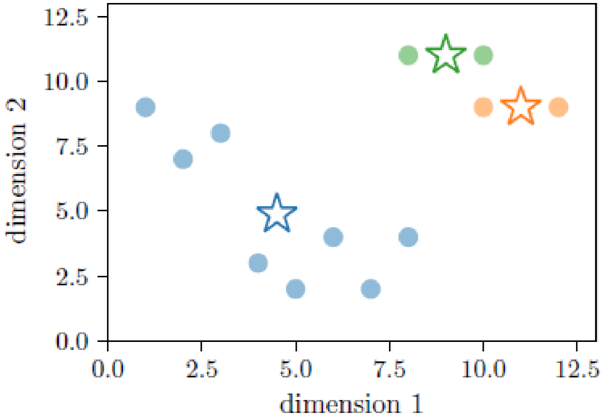
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Weak convergence for poor centroid initialization
  - Centroids placed randomly in data range
  - Centroids picked randomly from data points

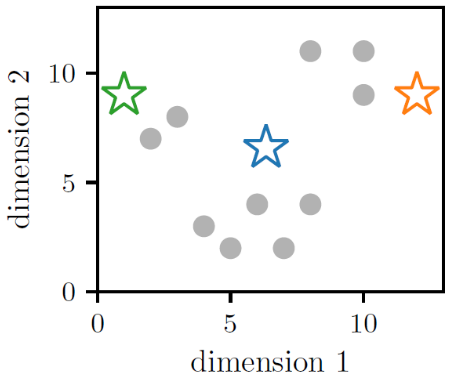
initialization



final result



- Solution strategies
  1. **Repetitive clustering**, each time selecting different centroids, selecting minimal
  2. **Iterative centroid placement**: 1st centroid into data sample center, 2nd into data point farthest aways, 3rd centroid into data point farthest aways from 1st and 2nd centroid, …
  3. **User-defined centroid placement** (leveraging a-priori knowledge)



Handling Empty Clusters

- Clusters C can end up empty after the cluster assignment step
- No point will ever by assigned to that cluster again
- Basic algorithm would stop once an empty cluster is met

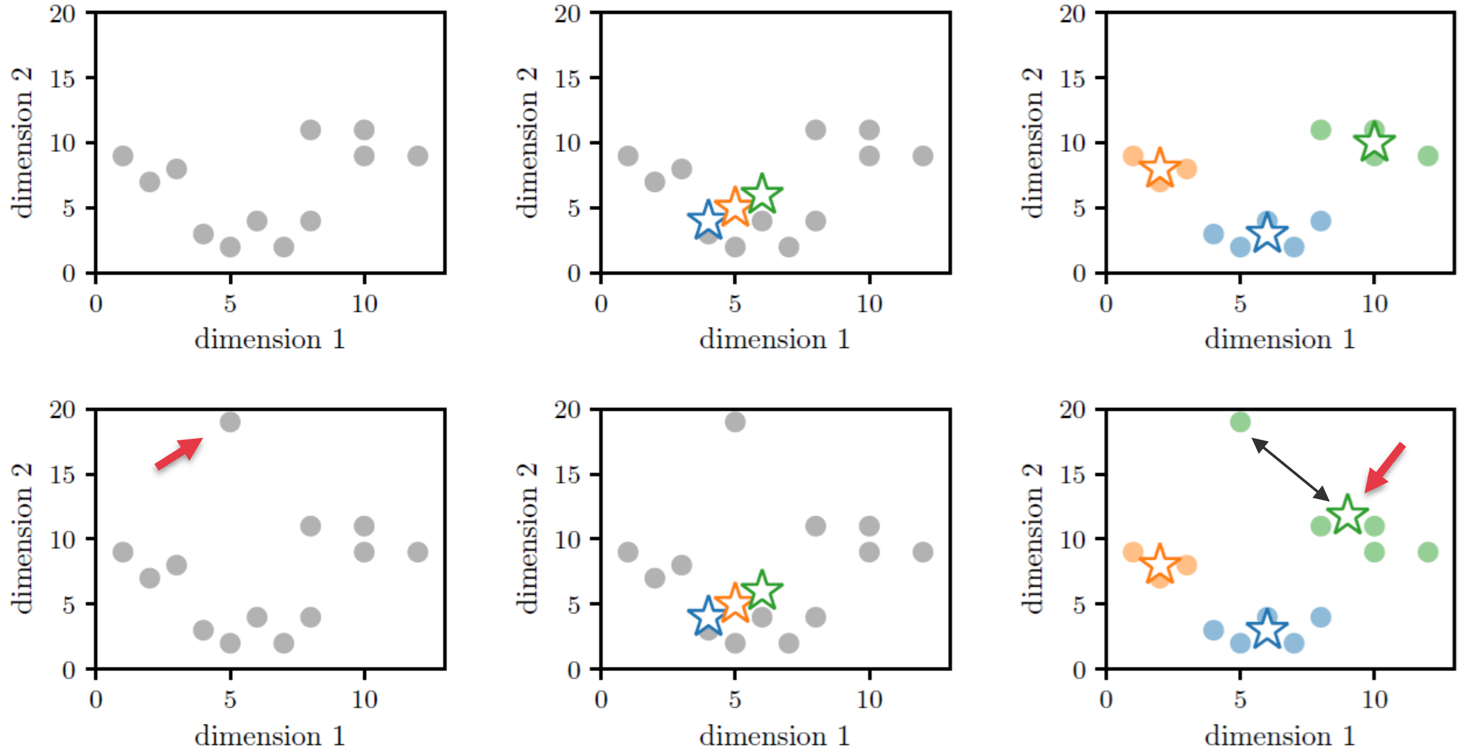
- Solution strategy: centroid re-placement

1. Place centroid at position of data point the farthest away from any centroid
   → Maximal reduction of total SSE, prone to selecting outliers

2. Place centroid into the cluster that is the less compact (largest SSE value)
   → Splits large clusters, potentially creating artificial sub-clusters

Sensitivity to Noise and Outliers

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

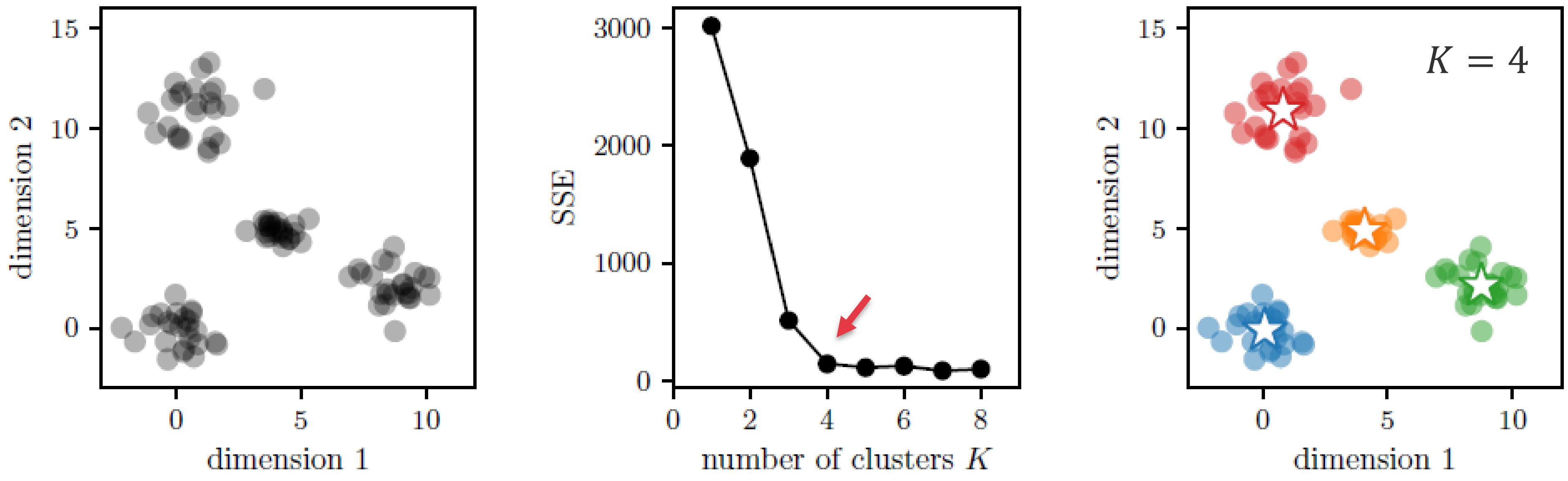


- Noise and/or outliers can heavily distort the final centroid positions
- Outliers: largest contribution to cluster validity metrics (SSE)
- Strategies
  - Outlier removal before clustering
  - Advanced K-means methods: remove extraordinarily strongly contributing data points
  - Post-processing by data point removal and re-running K-means from there

Selecting an Optimal K

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Hyperparameter study for K while tracking cluster validity metrics
- Selection of final K: elbow method



Variants of K-means

- Today: **Lloyd‘s algorithm** (simplest and basic form)

Other approaches:

- **MacQueen’s algorithm**: updates centroid positions with each new data point assignment (incremental updating)
  - Better convergence behavior
  - Introduces an order-dependency (no determinism can be installed), non-repetitive results

- **Hartingan-Wong’s algorithm**: initial centroids placed in vicinity of data center; centroid assignment based on SSE or other cluster validity metric instead of (Euclidean) distance metric
  - Designed to build more compact clusters
  - Initialization prone to generating artificial subclusters

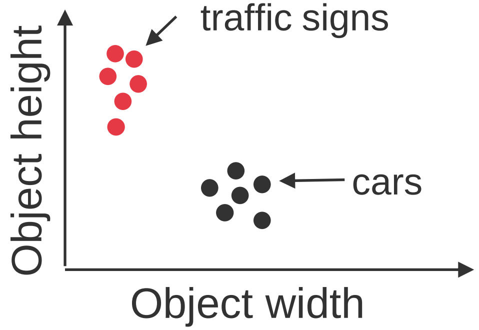
- **Many more**: bisecting K-means (better convergence), …

Unsupervised Learning: Applications

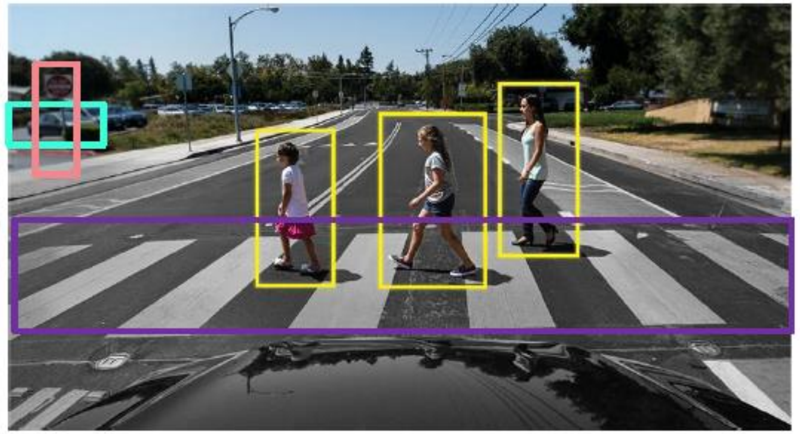
- Find similar customer behavior: shopping carts at the supermarket
  - Potential clusters: students, young family, retiree

- Data (image) compression and color quantization (link)

- Engineering applications:
  - Finding similar customer behavior, such as driving conditions
  - Finding patterns in high-dimensional measurement data
  - Data pre-processing for ML modeling

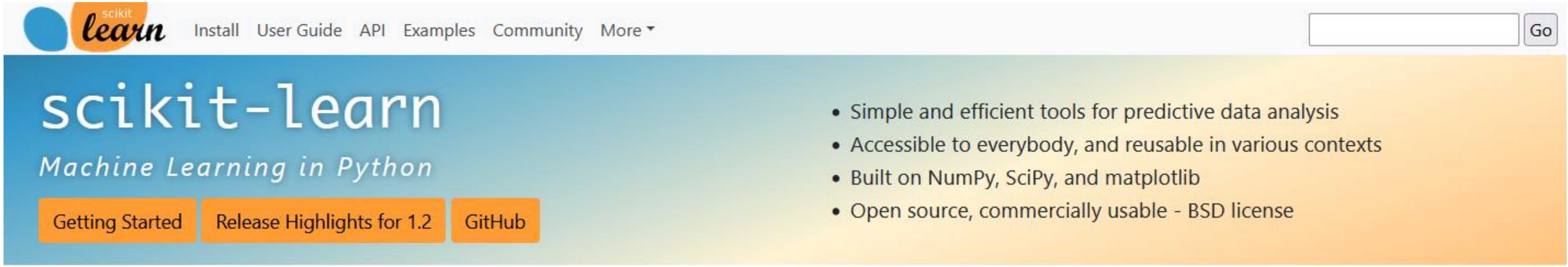


Anker definition for object detection methods



Python: scikit-learn

scikit-learn



- Most-used Python-based package for classical (statistical) machine learning
- Well-documented with underlying mathematics and literature

- Common structure across all ML methods: `.fit()`, `.predict()` methods

- K-means clustering: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

K-means clustering in scikit-learn

- sklearn.cluster.KMeans (link)

scikit learn   Install   User Guide   API   Examples   Community   More   Go

Prev   Up   Next

scikit-learn 1.2.2
Other versions

Please **cite us** if you use the software.

sklearn.cluster.KMeans
KMeans
Examples using sklearn.cluster.KMeans

sklearn.cluster.KMeans

class `sklearn.cluster.KMeans`(*n_clusters=8, *, init='k-means++', n_init='warn', max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd'*) [source]

K-Means clustering.

Read more in the User Guide.

Parameters:

n_clusters : *int, default=8*
The number of clusters to form as well as the number of centroids to generate.

init : *{'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++'*
Method for initialization:

'k-means++' : selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia. This technique speeds up convergence. The algorithm implemented is "greedy k-means++". It differs from the vanilla k-means++ by making several trials at each sampling step and choosing the best centroid among them.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n_clusters and a random state and return an initialization.

Exercise 03

November 8th, 2023

Prof. M. Stender | merten.stender@tu-berlin.de | tu.berlin/cpsme

Exercise 03

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Implement K-means algorithm (template provided, some lines to add)
- Evaluate on a small sample data set
- Compare against scikit-learn implementation
- [Extra]: Implement the same functionality as object-oriented code

Questions?