Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2022/23

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) *NumPy, SciPy*
6. Datenverarbeitung und -visualisierung
   a) *Pandas, matplotlib, NLTK*
7. Machine Learning (scikit-learn)
   a) **Grundlegende Ansätze (Datensätze, Auswertung)**
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers

Themen

I. Projektaufgabe 4

    1. Herangehensweise & Tipps

II. Begrifflichkeiten

    1. „Künstliche Intelligenz“, Data Science & Machine Learning

    2. Agenten

III. Clustering

Projektaufgabe 4

„Datenverarbeitung und -darstellung“

1. Hellinger-Distanz zwischen zwei Matrizen P, Q

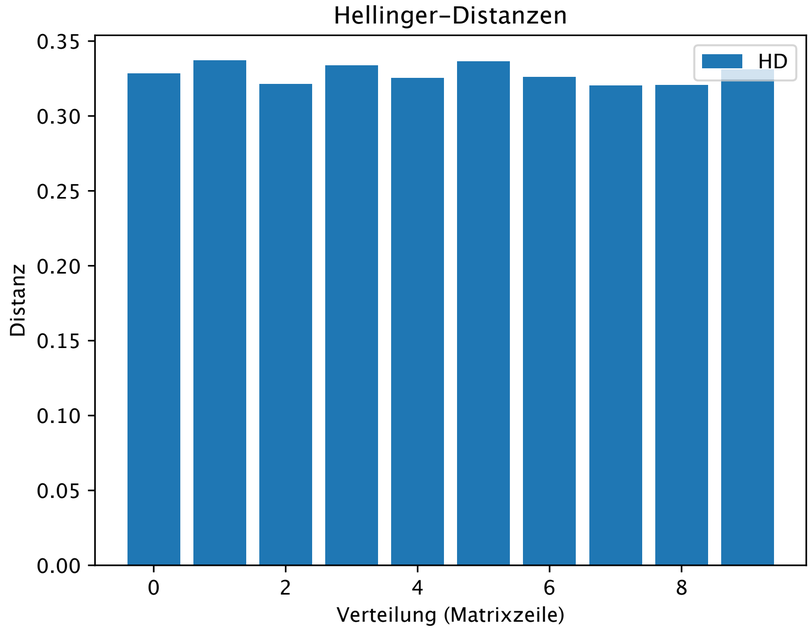
   - Distanz jeweils zeilenweise in Ergebnisvektor H

2. Minimale Distanz

   - Matrix mit Zeile aus P und Zeile aus Q

3. Säulendiagramm der Distanzen

$$h_i = \frac{1}{\sqrt{2}} \sqrt{\sum_{j=1}^{k} \left(\sqrt{p_{i,j}} - \sqrt{q_{i,j}} \right)^2}$$



Herangehensweise & Tipps

- NumPy-Arrays als Eingabe und Ausgabe
- Keine Schleifen erlaubt
- Funktionen nur mittels Python (ohne NumPy) implementiert im Moodle verfügbar
- Für die minimale Distanz eine passende NumPy-Funktion selbst raussuchen/ bestimmen
- Beschriftungen bei Säulendiagramm exakt identisch
- Rückgabe des `plt`-Moduls

II.
Begrifflichkeiten

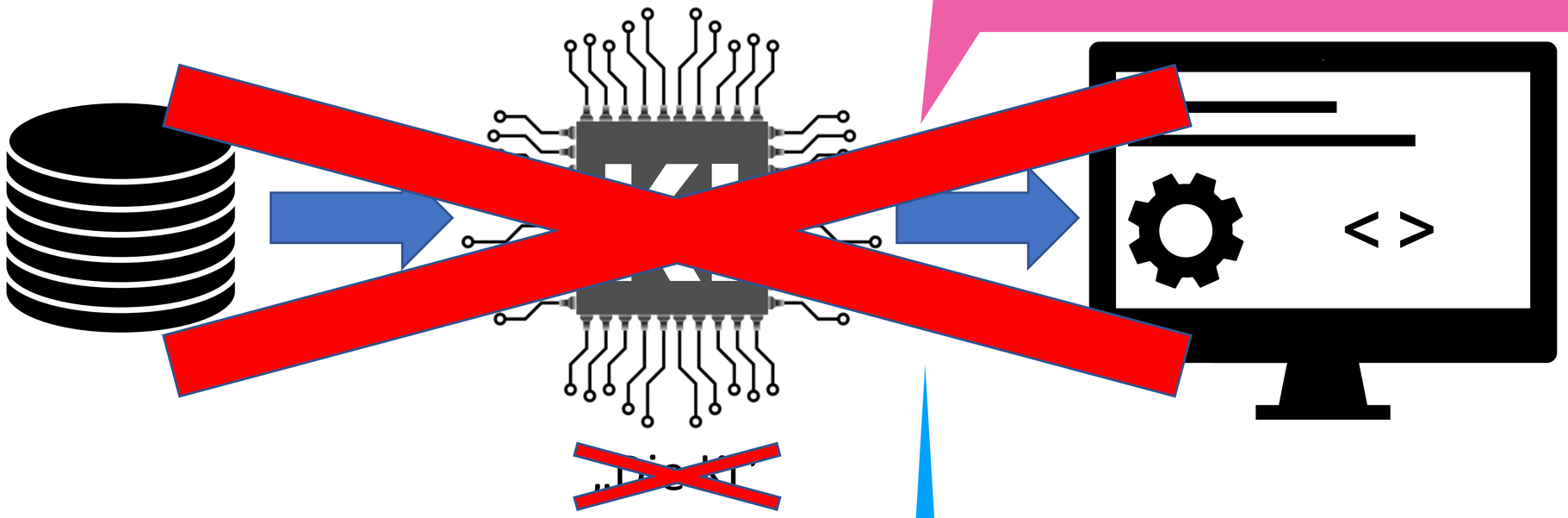
1. „Künstliche Intelligenz“, Maschinelles Lernen & Data Science

Danksagung

- Nachfolgende Folien sind teilweise übernommen aus folgenden Vorlesungen und Vorträgen
  - Prof. Ralf Möller: „Non-Standard Datenbanken und Data-Mining“
  - Dr. Marcel Gehrke, Prof. Ralf Möller: „Einführung in Web und Data Science“
  - Prof. Ralf Möller: „Von AlphaZero zur Mars-Rover-Autonomie“

„Künstliche Intelligenz“ Märchen

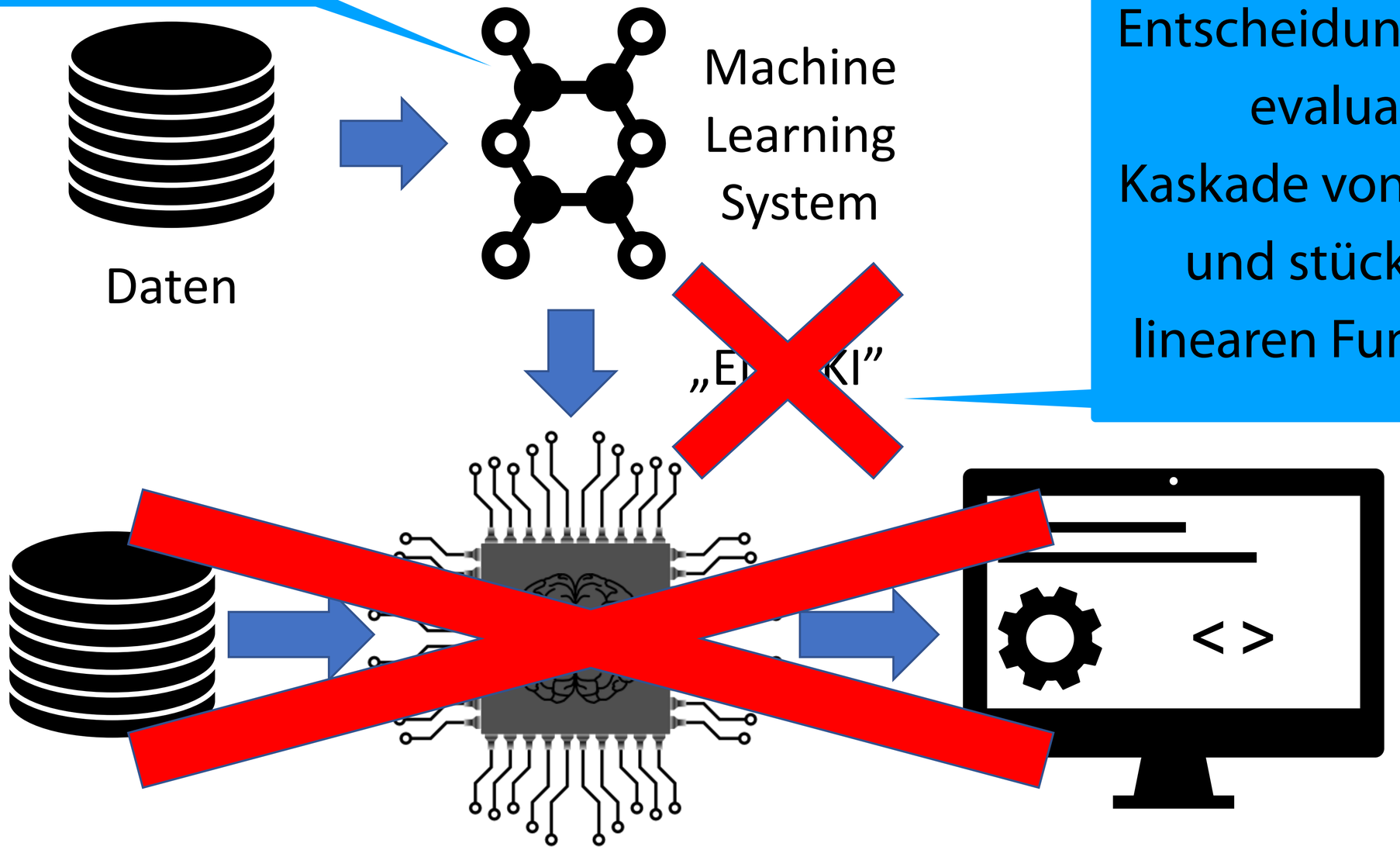
„KI“ als ein Baustein, der wie im Kopf funktioniert, und dann die „Intelligenz“ auf einen Computer bringt.



Zum Beispiel:

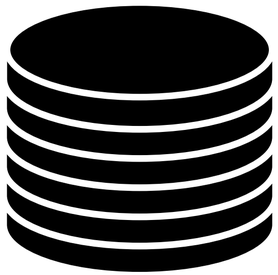
Klassifikation, Regression, Prädiktion, Diagnose, …

TensorFlow, PyTorch

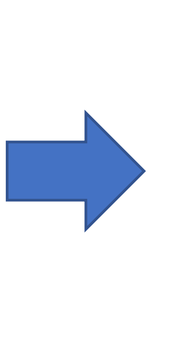


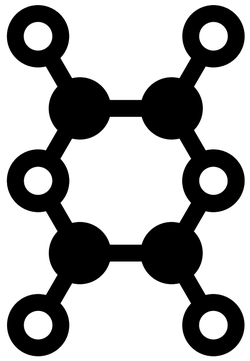
Programm:
Entscheidungsbaum-evaluator,
Kaskade von linearen und stückweise linearen Funktionen

Maschinelles Lernen

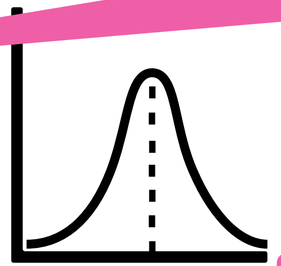


Trainingsdaten





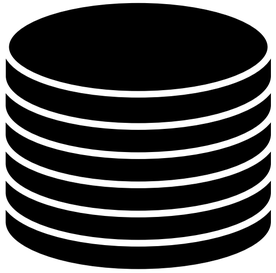
Training



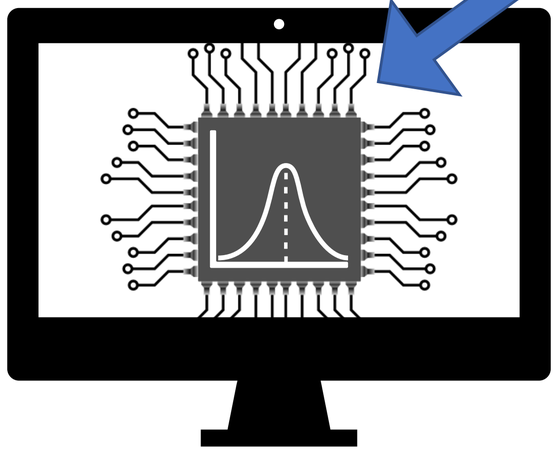
Modell (P

Ein Modell aus den Trainigsdaten „lernen“ (berechnen).

Modell (Programm) mit anderen Daten nutzen.



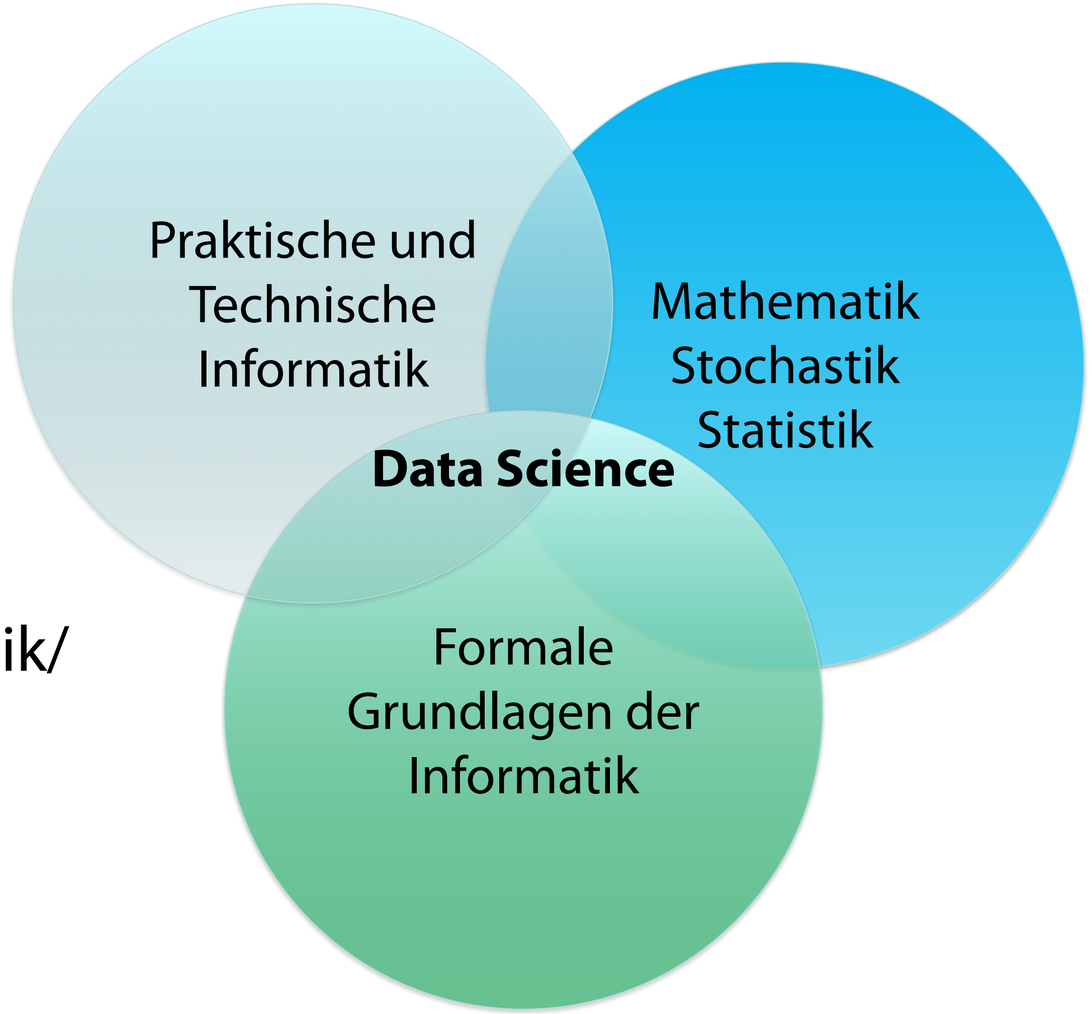
(Test-)Daten



Ergebnis
(Klassfikation etc.)

Data Science

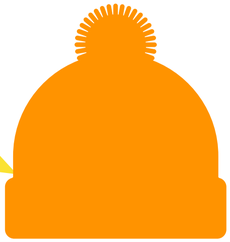
- Extraktion von Wissen aus Daten (u.a. Graphdaten)
- Begriff schon vor 60 Jahren für Informatik vorgeschlagen
- Entwicklung innovativer Konzepte in den Bereichen Logik, Datenbanken und Stochastik/ Statistik (Datenanalyse und Wissensentdeckung)
- Verwendung von LADS und Analysis



II.
Begrifflichkeiten

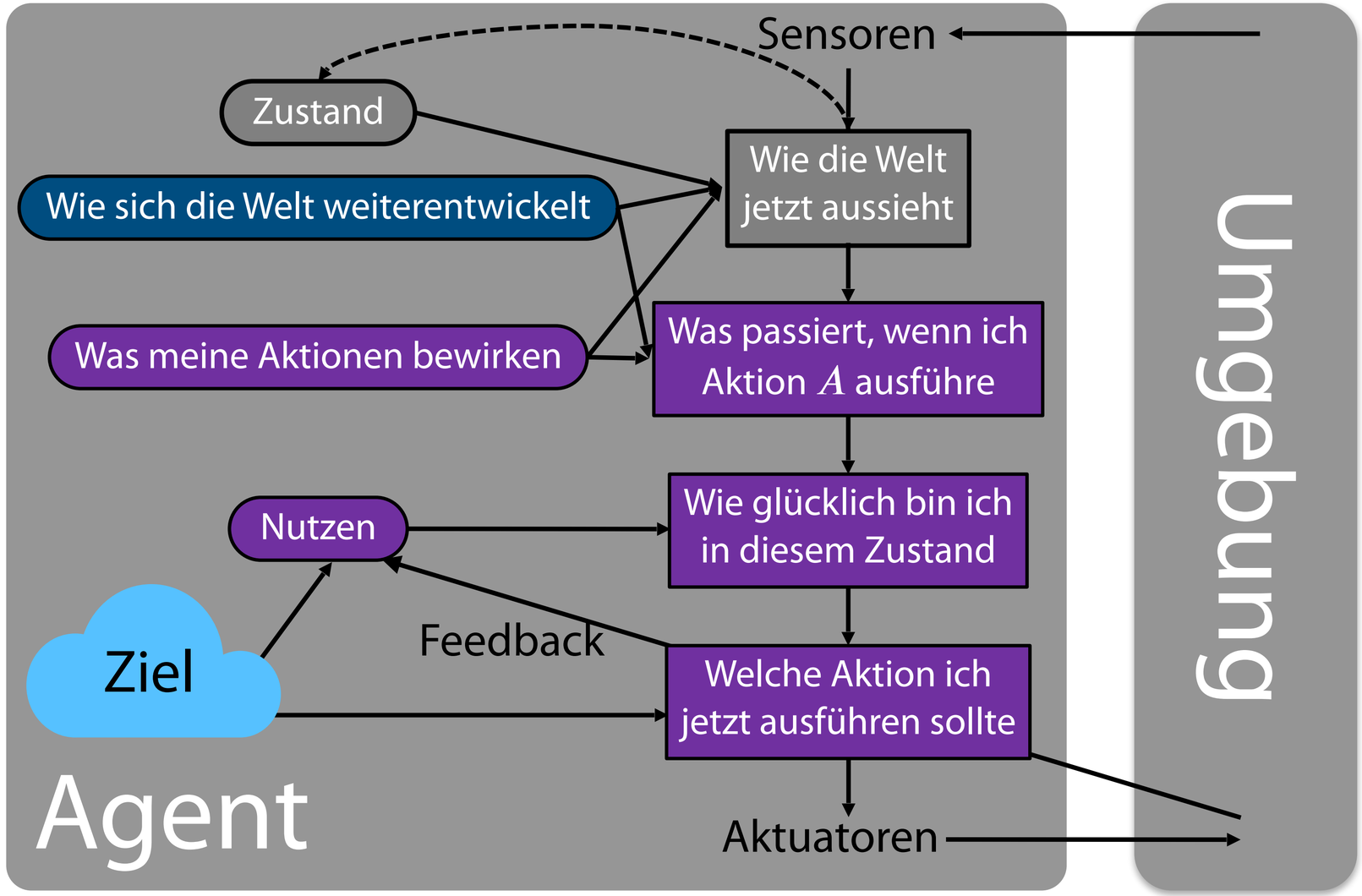
2. Agenten

Und wo ist jetzt das, was man häufig unter „KI“ versteht?

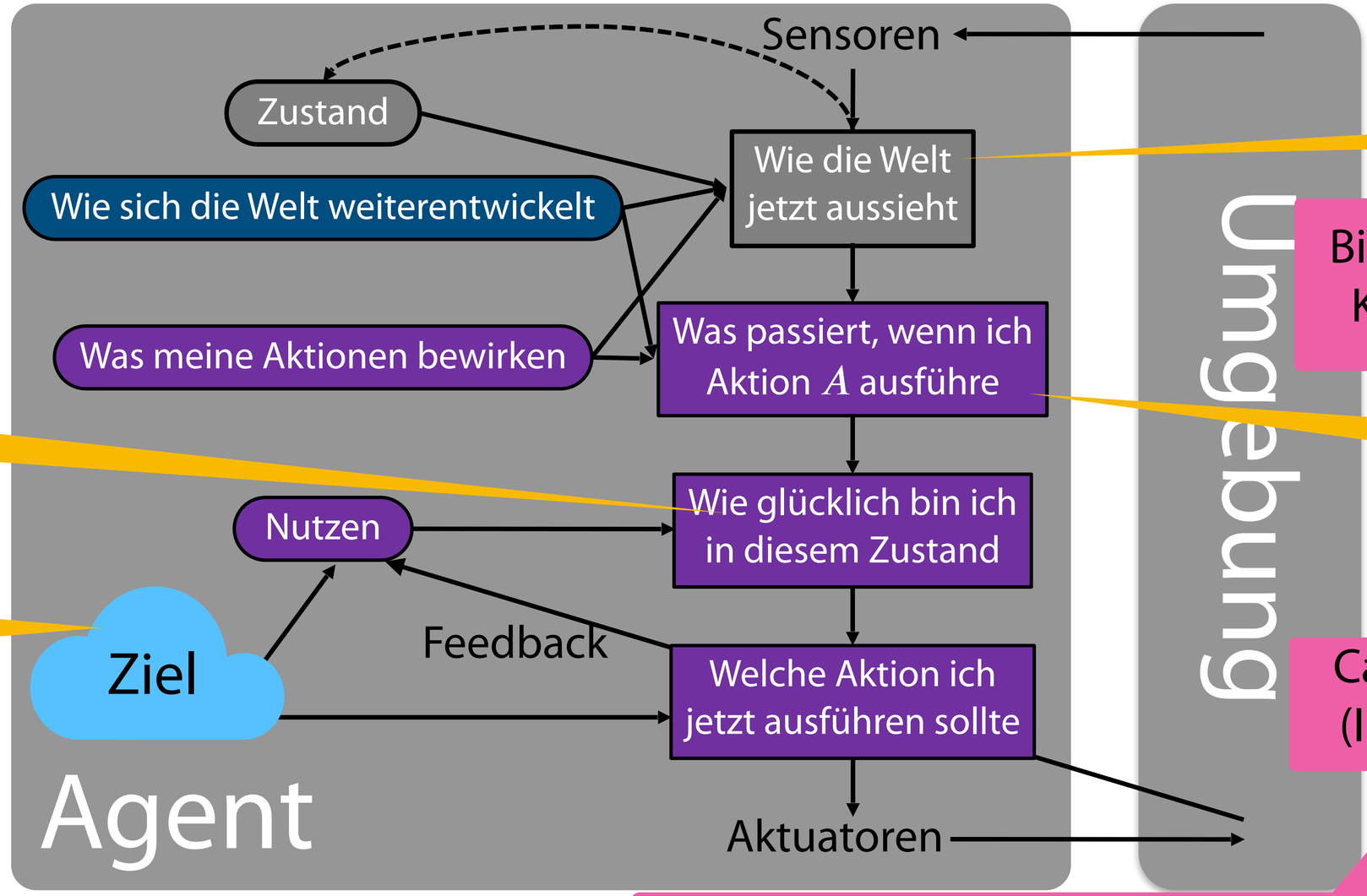


Agenten

- Wissenschaft der intelligenten Systeme
- Agenten
  - Haben/bilden Ziele
  - Sensoren/Aktoren
  - Handlungsplanung
  - Lernen zur Laufzeit
- Agenten interagieren mit Menschen (and anderen Agenten)
  - Ziele der Agenten beeinflussbar



Agent: Maschinelles Lernen



Regression

Bilderkennung, Klassifikation

Causal Models (Intervention)

Reinforcement Learning (Ziel und Nutzen aus Feedback „lernen“)

Causal Models (Counterfactuals) „Was wäre, wenn ich x getan hätte?“

Konzept

- „Werkzeuge“ für das wissenschaftliche Arbeiten
  - Fokus auf Werkzeug *Python*
  - Anwendung *Machine Learning* und *Data Science*
- Vorstellung einer Auswahl
- Verfahren des *Machine Learning* und *Data Science*
  1. Problematik
  2. Idee der Theorie
  3. Lösung mittels Python & Paketen

Vorlesungen zu den Themen

Kleine Teilmenge von IFIS-Modulen

- Bachelor
  - **Einführung in Web und Data Science** (CS1800)
  - **Non-Standard Datenbanken und Data Mining** (CS3130)
  - Logikprogrammierung (CS3055)
- Master
  - Aktuelle Themen Data Science und KI (CS 5070) (z.B. zum Thema „Probabilistic and Differential Programming“)
  - **Intelligente Agenten** (CS 4514)
  - Informationssysteme (CS4130)

III.
Clustering

Clustering

Problem	Clustering, Partitionierung
Verfahren	*k*-Means
Art	Unüberwacht
Hyperparameter	Initiale Zentroiden, Anzahl Zentroiden
Python-Paket	SKLearn
Klasse	`sklearn.cluster.KMeans`

Unüberwacht:
Daten haben keine Label, es wird hier nach einer „sinnvollen“ Ordnung/ Gruppierung der Daten gesucht.

Dafür gilt es z.B. eine Fehlerfunktion zu minimieren.

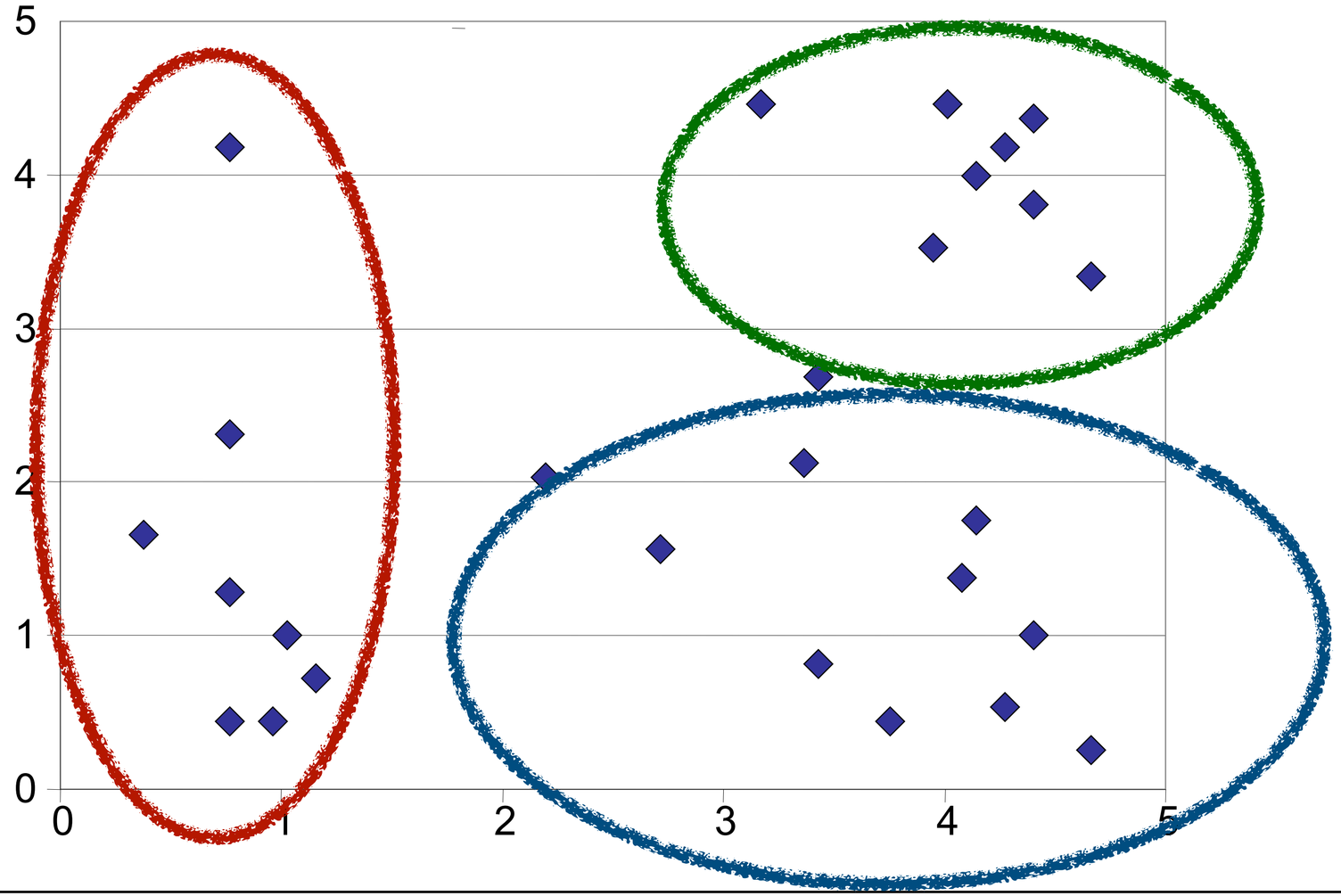
k-Means

- Form des unüberwachten Lernens
- Suche nach natürlicher Gruppierungen von Objekten
  - Klassen direkt aus Daten bestimmen
    - Hohe Intra-Klassen-Ähnlichkeit
    - Kleine Inter-Klassen-Ähnlichkeit

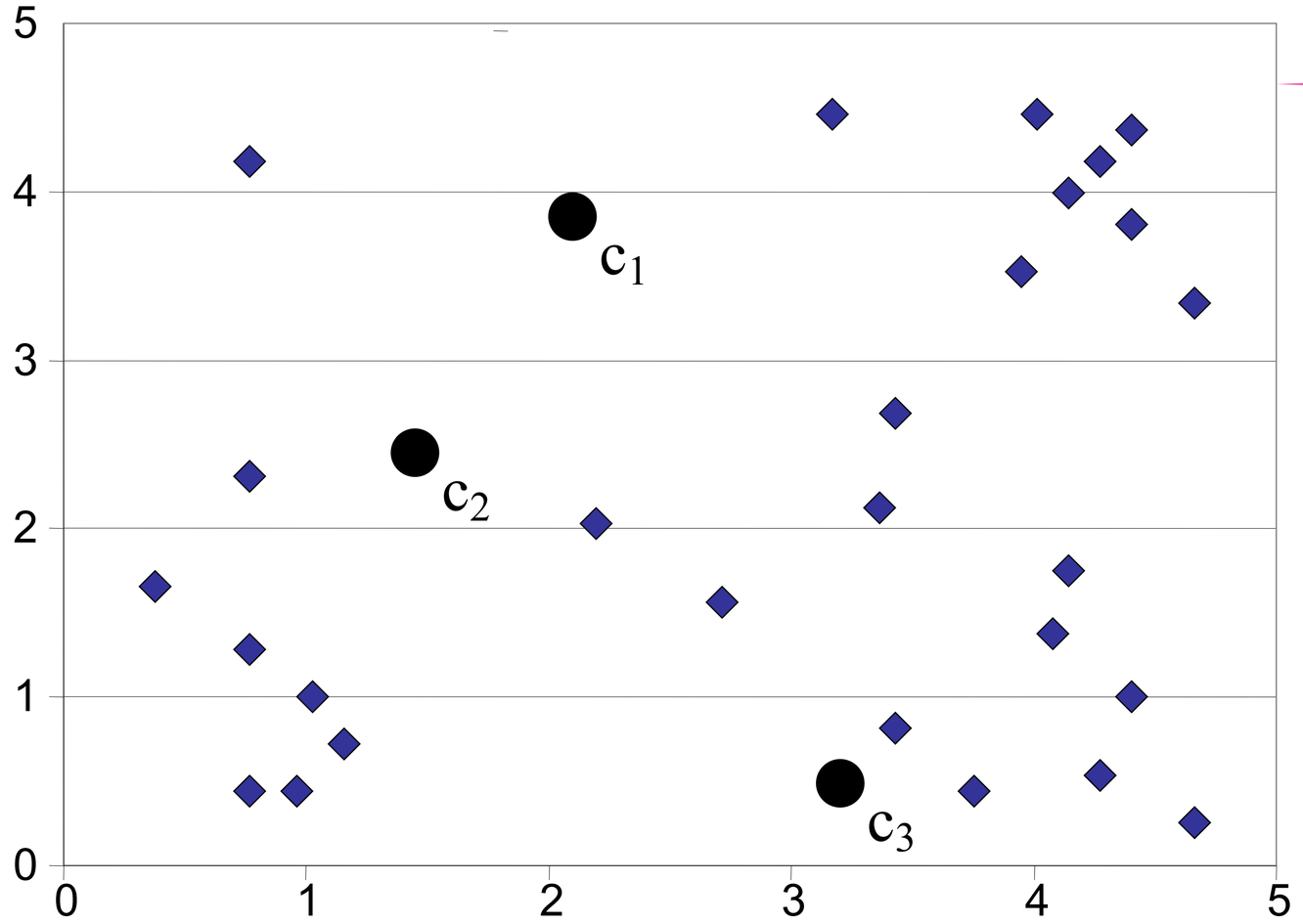
$$\|x - y\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Distanzmaß,
z.B. euklidische Distanz

Gruppierungen gesucht!



k-Means; $t = 0$



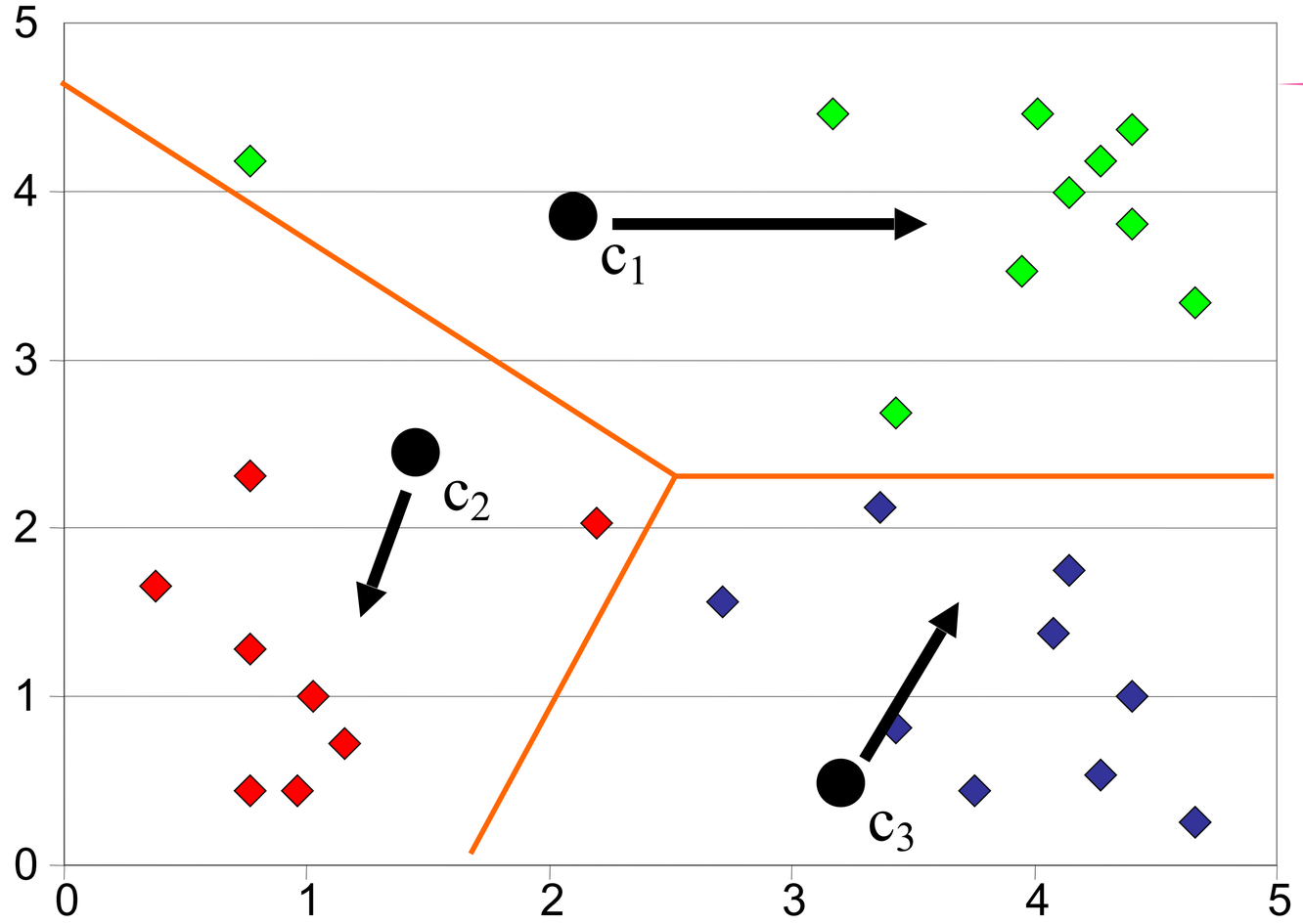
Drei initiale Zentroiden in den Datenpunkten

Zuordnung jedes Datenpunkts zum dichtesten Zentroiden, bilden Cluster *C*.

$$C_i^t = \left\{ x_j : \left\| x_j - c_i^t \right\|_2 \leq \left\| x_j - c_r^t \right\|_2 \text{ for all } r = 1 \ldots k,\ r \neq i \right\}$$

Zentroid $i = 1, \ldots, k$ und Iteration beginnend mit $t = 0$

k-Means; $t = 0 + 1$



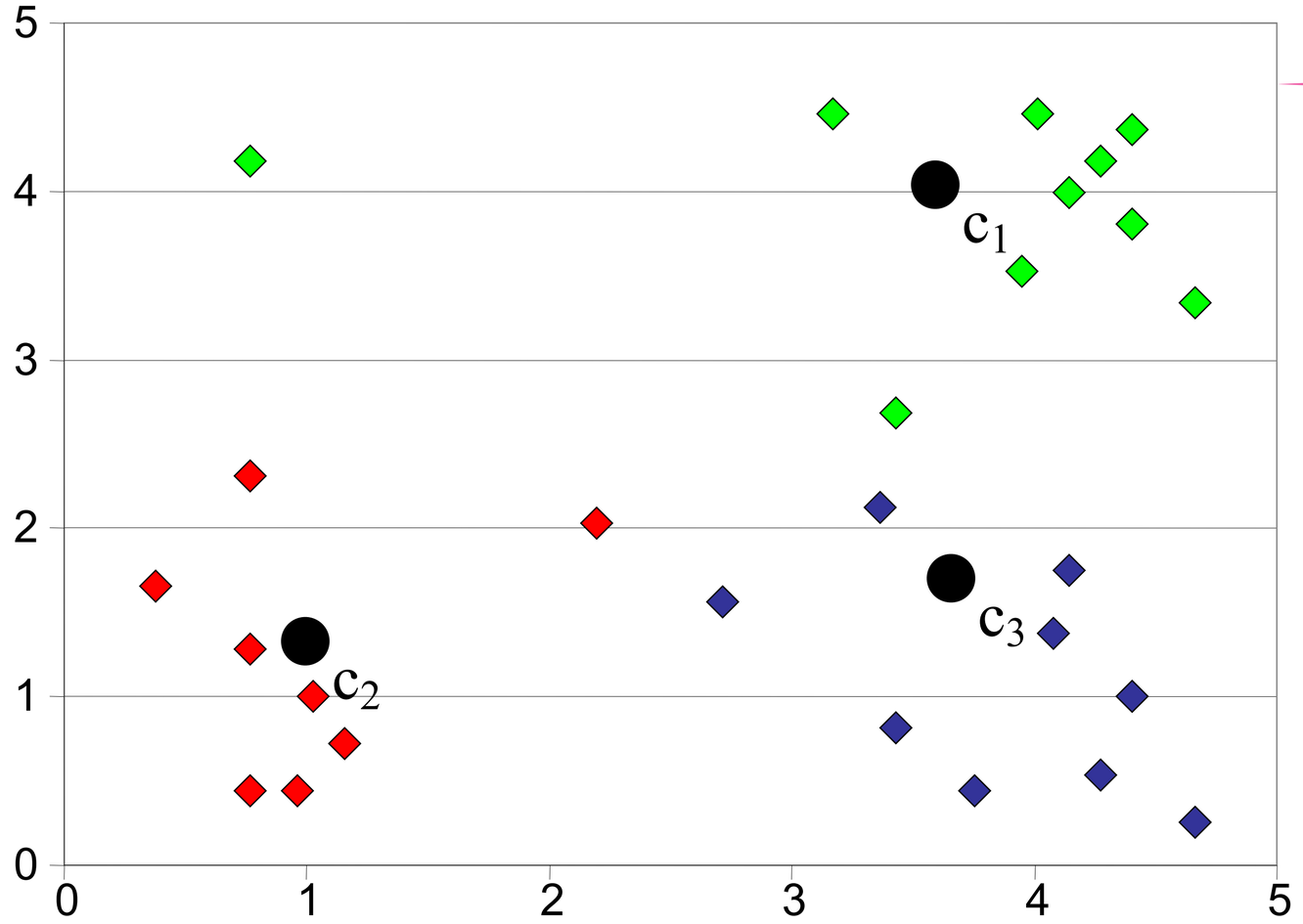
Drei Cluster farblich markiert.

Zentroiden in „Mitte“ jedes Cluster bewegen.

$$c_i^{t+1} = \frac{1}{\left|C_i^t\right|} \sum_{x_j \in C_i^t} x_j$$

Weiter iterieren.

k-Means; *t* = 1



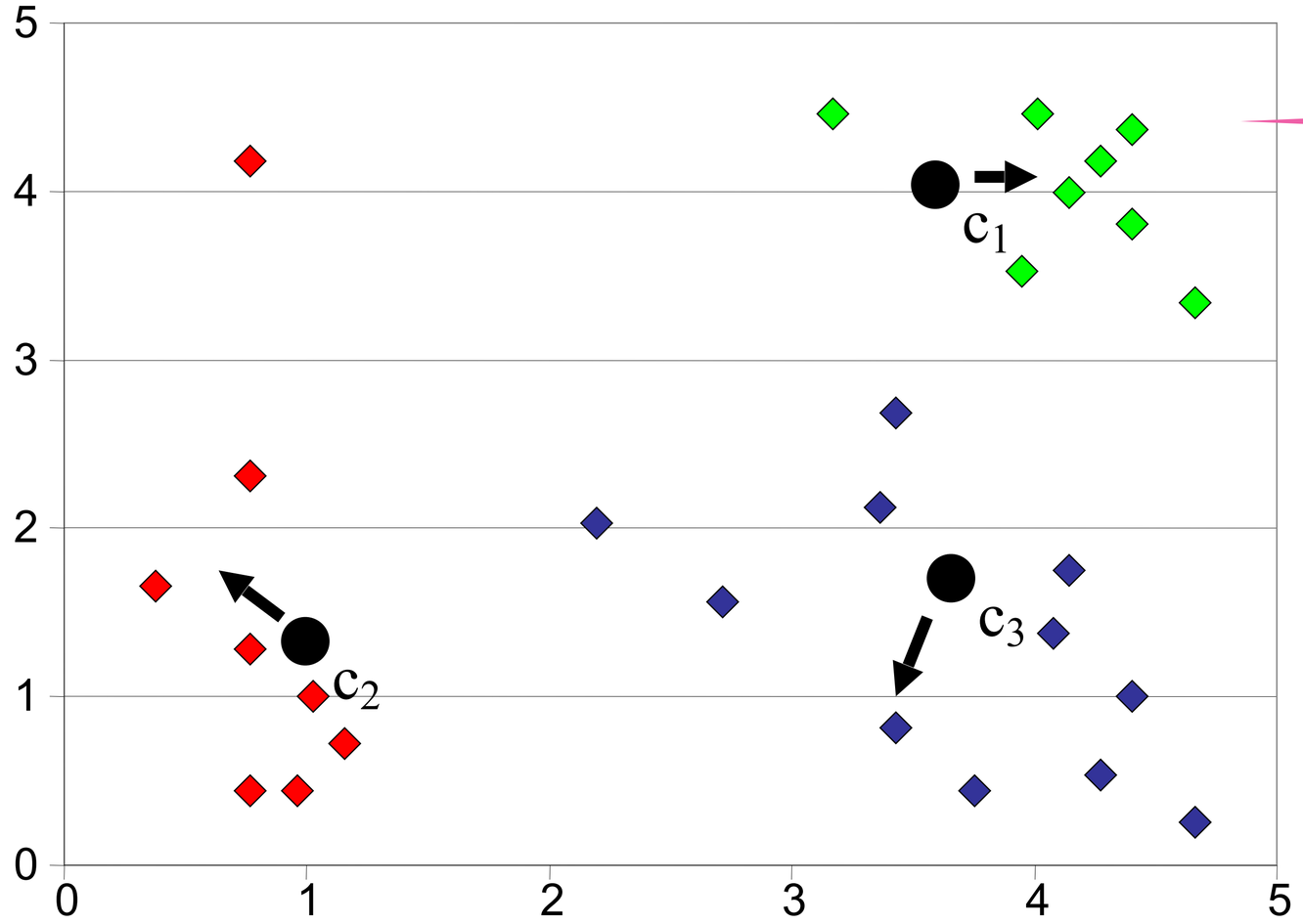
Zentroiden an neuen Positionen

Datenpunkte den Zentroiden neu zuordnen, Cluster verändern sich.

$$C_i^t = \left\{ x_j : \left\| x_j - c_i^t \right\|_2 \leq \left\| x_j - c_r^t \right\|_2 \text{ for all } r = 1 \ldots k,\ r \neq i \right\}$$

Zentroid $i = 1, \ldots, k$ und Iteration beginnend mit $t = 0$

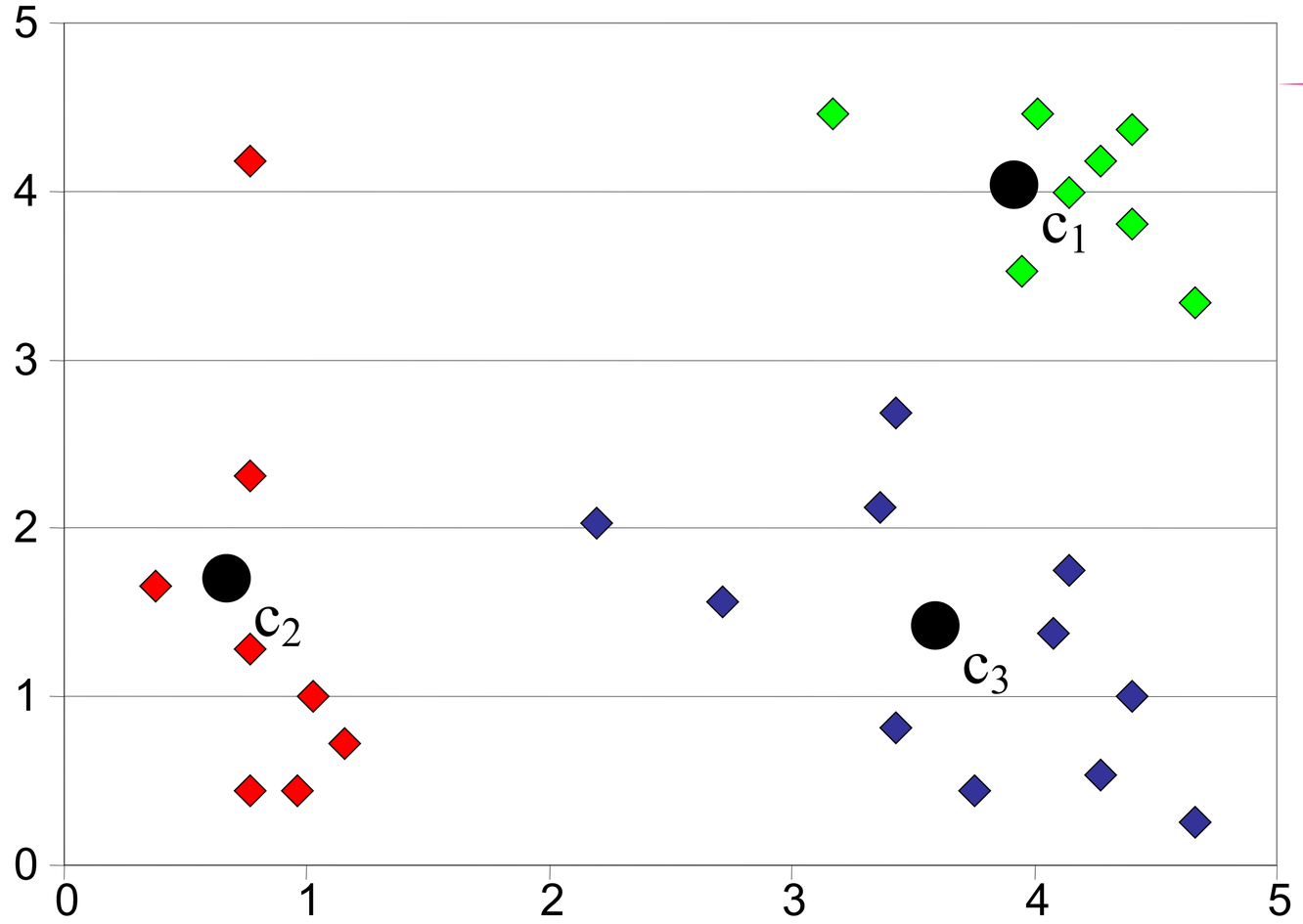
k-Means; $t = 1 + 1$



Erneut Zentroiden in die Mitte schieben.

$$c_i^{t+1} = \frac{1}{\left|C_i^t\right|} \sum_{x_j \in C_i^t} x_j$$

k-Means; $t = 2$



Zentroiden wieder an neuen Positionen

Keine Änderungen den Zugehörigkeiten mehr

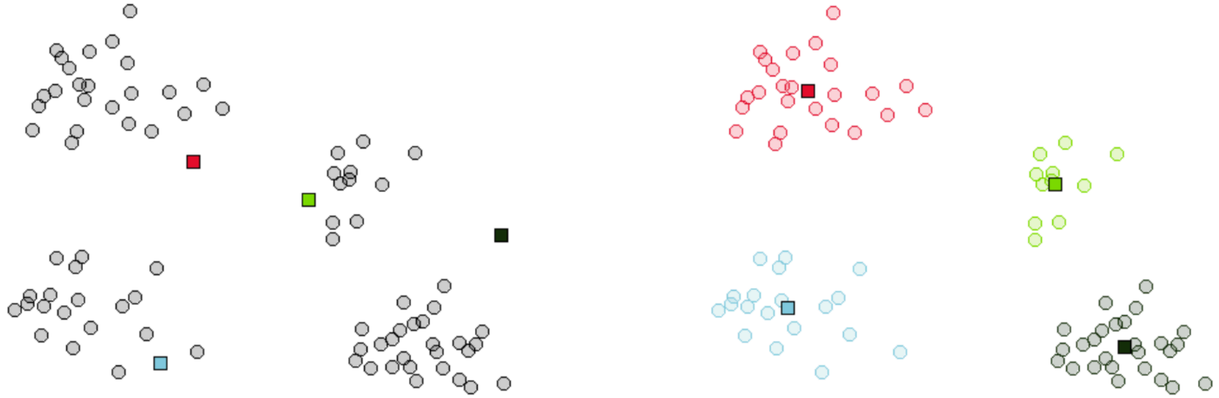
→

Algorithmus beendet

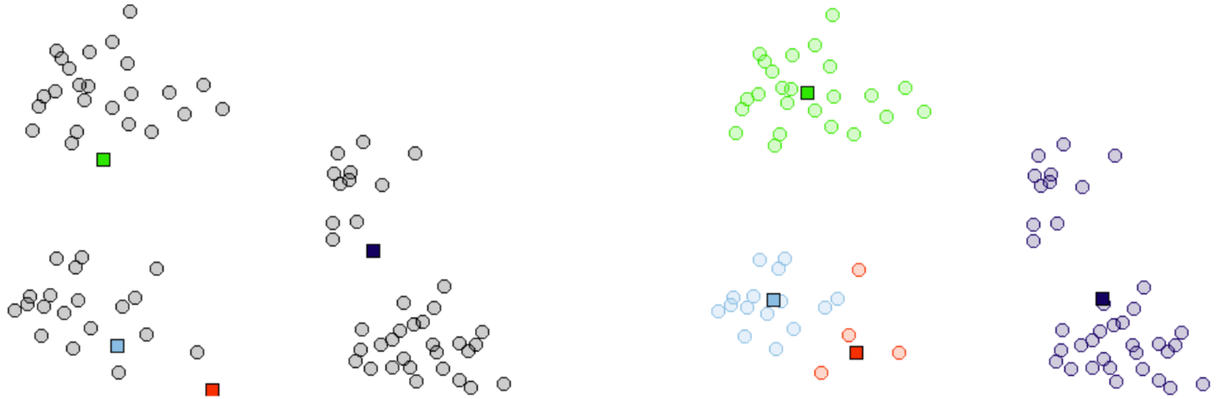
J. B. MacQueen: "Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*", Berkeley, University of California Press, 1:281-297, **1967**

k-Means: Ergebnis hängt vom Startwert ab

Gutes Ergebnis



Schlechtes Ergebnis



Installation

- Python Paket
  - https://scikit-learn.org/
- Installation z.B. mit `pip3 install scikit-learn`
- Import, z.B.

Paket `sklearn` und Installation als `scikit-learn`

Intern Nutzung von NumPy und C

Integration mit z.B. Matplotlib und Pandas

```
from sklearn.cluster import KMeans
```

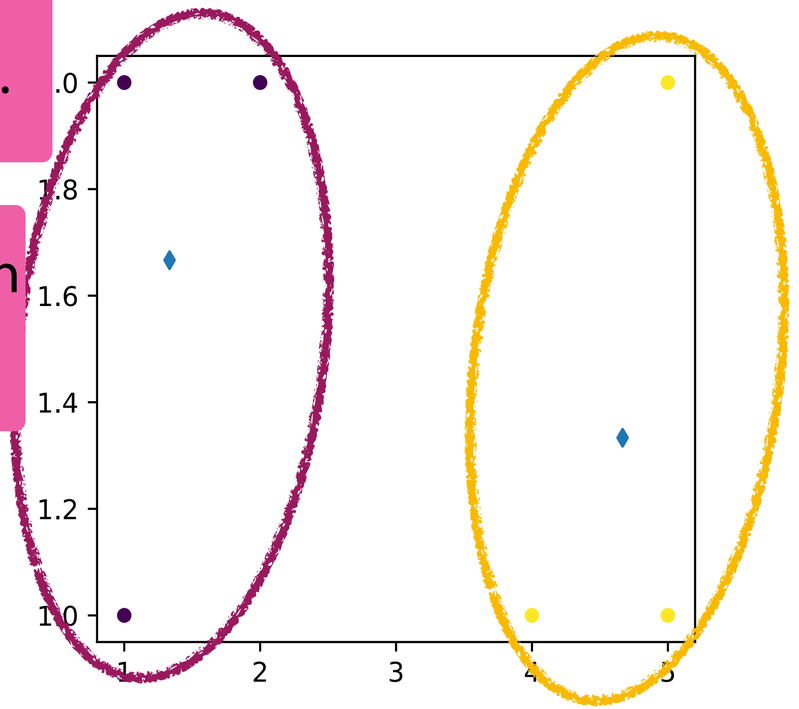
KMeans-Klasse

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X = np.array([
   [1, 1], [1, 2], [2, 2],
   [4, 1], [5, 1], [5, 2]
])

plt.scatter(X[:, 0], X[:, 1]
plt.show()

km = KMeans(n_clusters=2, init='random', tol=1e-4)
km.fit(X)

print(km.cluster_centers_)

print(km.labels_)


print(km.predict([[1, 1.5]]))
```

```
[[1.33333333 1.66666667]
 [4.66666667 1.33333333]]
```

```
[0 0 0 1 1 1]
```

```
[0]
```



Klasse mit gewählten Hyperparametern erzeugen.

Modell mittels *k*-Means auf den Daten *X* trainieren.

Einige Werte können direkt den Attributen der Klasse entnommen werden.

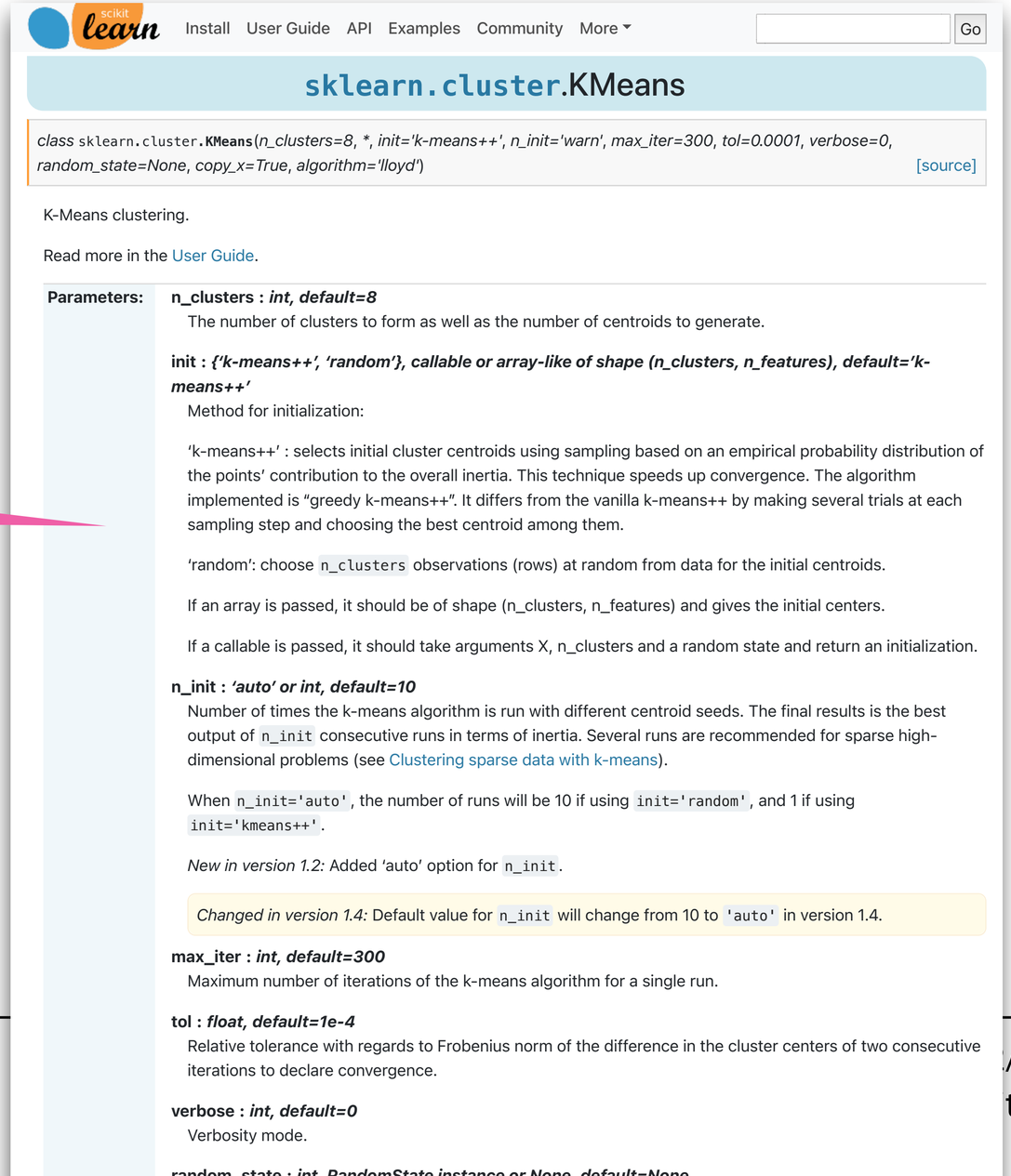
Datenpunkte und Plot mittels Matplotlib.

Vorhersage für neues Datum bestimmen.

Methodenstruktur SKLearn

```
km = 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
)

# Attributes
km.cluster_centers_
km.labels_
km.n_iter_

# Methods
fit(X[, y, sample_weight])
fit_predict(X[, y, sample_weight])
fit_transform(X[, y, sample_weight])
get_feature_names_out([input_features])
predict(X[, sample_weight])
transform(X)
```

Ausführliche Dokumentation für jede Klasse, mit ähnlichem Aufbau und ähnlichen Methoden, z.B. `fit()`, …



scikit-learn Install User Guide API Examples Community More Go

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.

n_init : *'auto' or int, default=10*
Number of times the k-means algorithm is run with different centroid seeds. The final results is the best output of `n_init` consecutive runs in terms of inertia. Several runs are recommended for sparse high-dimensional problems (see Clustering sparse data with k-means).

When `n_init='auto'`, the number of runs will be 10 if using `init='random'`, and 1 if using `init='kmeans++'`.

New in version 1.2: Added 'auto' option for `n_init`.

Changed in version 1.4: Default value for `n_init` will change from 10 to `'auto'` in version 1.4.

max_iter : *int, default=300*
Maximum number of iterations of the k-means algorithm for a single run.

tol : *float, default=1e-4*
Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence.

verbose : *int, default=0*
Verbosity mode.

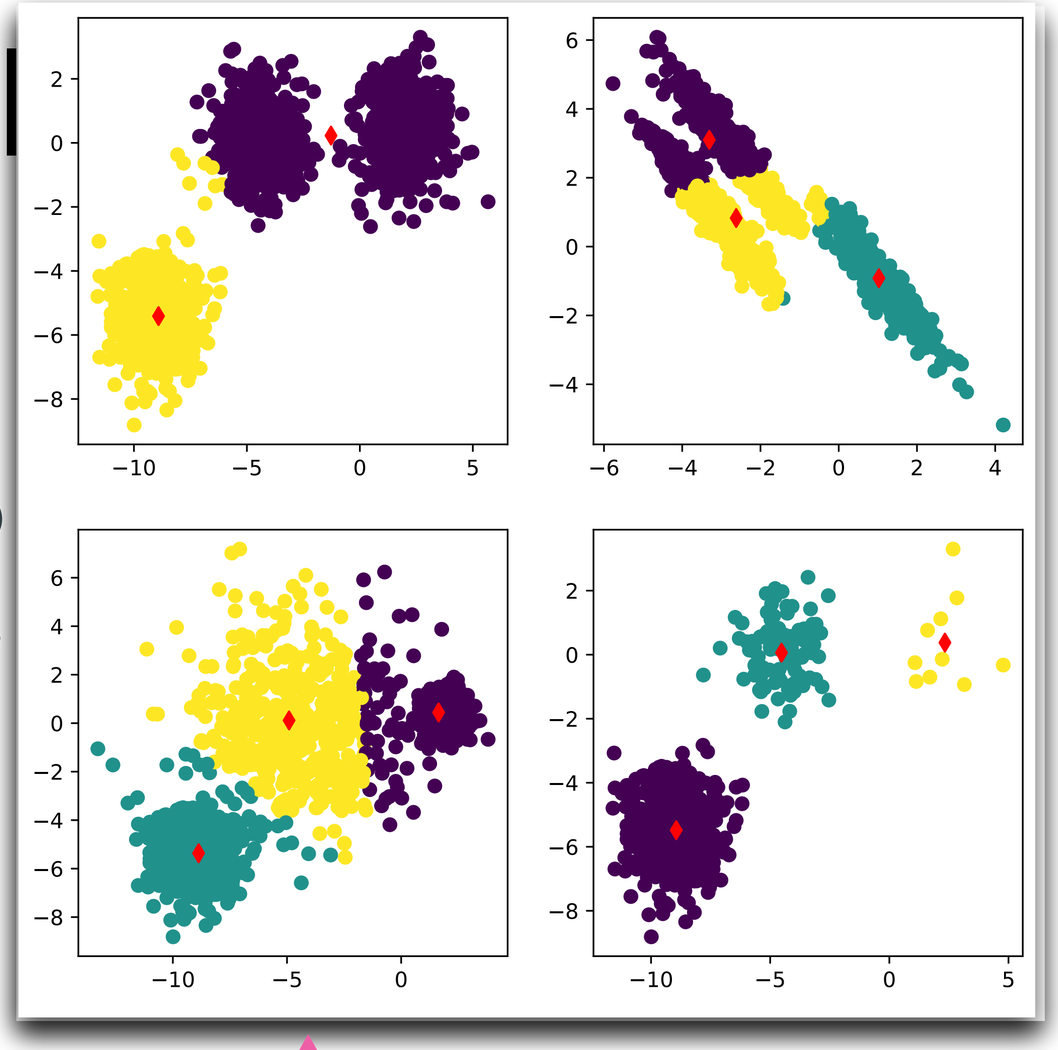
random_state : *int, RandomState instance or None, default=None*

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

Beispiel

Beispieldaten erzeugen und in verschiede „spannende" Instanzen transformieren.

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

X_a, y_a = make_blobs(n_samples=1500, random_state=170)
X_b, y_b = np.dot(X_a, [[0.60834549,… ]]), y_a
X_c, y_c = make_blobs(n_samples=1500, cluster_std=[1.0,
X_d, y_d = np.vstack(…), … * 10

# Plot Data

km_a = KMeans(n_clusters=2, random_state=170).fit(X_a)
km_b = KMeans(n_clusters=3, random_state=170).fit(X_b)
km_c = KMeans(n_clusters=3, random_state=170).fit(X_c)
km_d = KMeans(n_clusters=3, random_state=170).fit(X_d)

# Plot Cluster
```



Jeweils ein Modell mittels k-Means berechnen.

Rote Rauten sind die Zentroiden, man sieht, dass das Ergebnis teilweise suboptimal ist.

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

Diskussion *k*-Means

- Meist relativ wenige Schritte notwendig
  - Findet aber ggf. nur lokales Optimum
- Nur anwendbar, wenn Mittel definiert
- Basiert auf vorgegebener Clusteranzahl *k*
- Cluster haben meist gleiche Größe
- Probleme bei nichtkonvexen Formen

Ergebnis

Wunsch

Zusammenfassung

I. Projektaufgabe 4

II. Begrifflichkeiten

    1. „Künstliche Intelligenz“
    2. Data Science
    3. Machine Learning
    4. Agenten

III. Clustering

Nächste Woche gucken wir uns weitere Beispiele für ML Verfahren an.

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) *NumPy, SciPy*
6. Datenverarbeitung und -visualisierung
   a) *Pandas, matplotlib, NLTK*
7. Machine Learning (scikit-learn)
   a) *Grundlegende Ansätze (Datensätze, Auswertung)*
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers