Data Science 2

Vorlesung 4 - Clustering

Prof. Dr. Christian Bockermann

Hochschule Bochum

Wintersemester 2024/2025

Inhalt

Clustering sucht Aufteilung von Daten in ähnliche Gruppen

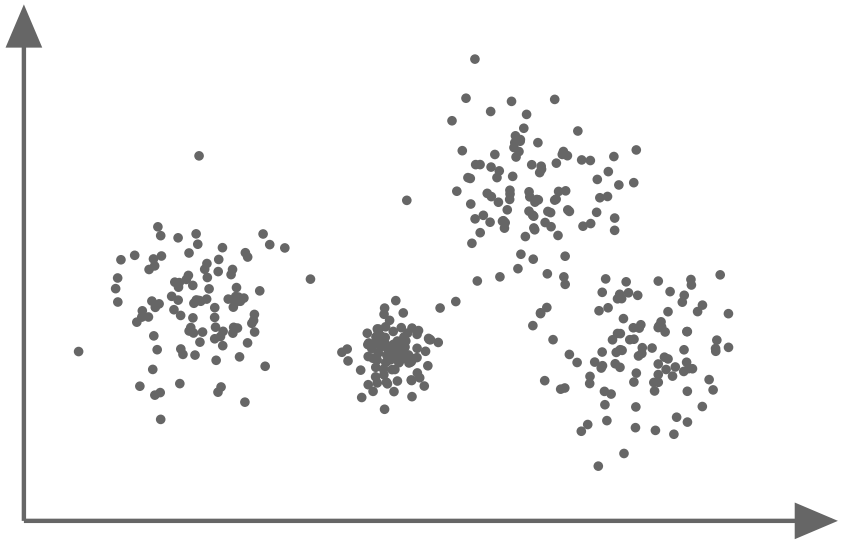
- Datenmenge **X** von Beispielen (keine Klassen gegeben!)
- Parameter k zu findender Gruppen
- Abstandsmaß $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- Qualitätsfunktion q

Ziel:

- Abstand *innerhalb* der Gruppen soll minimiert, Abstand *zwischen* den Gruppen soll maximiert werden

Beispiel: Clustering

Sei $\mathbf{C} = C_1, \ldots, C_k$ eine Aufteilung der Daten X (ein *Clustering*)

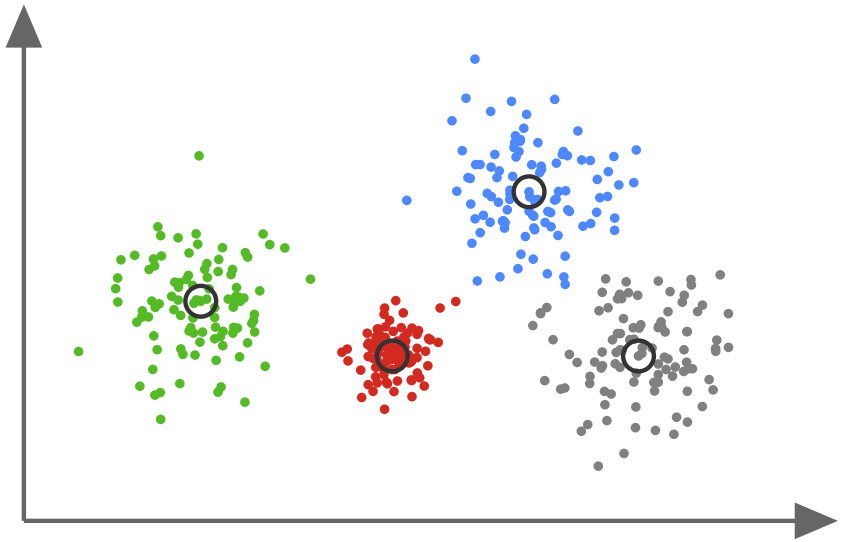


Qualitätsfunktion: (Innere Abstände)

$$q_{inner}(\mathbf{C}) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \bar{\mathbf{c}}_i) \quad \text{, mit } \bar{\mathbf{c}}_i \text{ Zentrum von } C_i$$

Beispiel: Clustering

Sei $\mathbf{C} = C_1, \ldots, C_k$ eine Aufteilung der Daten X (ein *Clustering*)



Clustering auf Datenpunkten mit $k = 4$. Die schwarzen Kreise markieren jeweils das Zentrum $\bar{\mathbf{c}}_i$ des jeweiligen Cluster C_i.

Qualitätsfunktion: (Innere Abstände)

$$q_{inner}(\mathbf{C}) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \bar{\mathbf{c}}_i) \quad \text{, mit } \bar{\mathbf{c}}_i \text{ Zentrum von } C_i$$

Beispiel: Clustering

- Clustering unter mehreren Qualitätsaspekten:

$$q_{inner}(\mathbf{C}) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \bar{\mathbf{c}}_i) \quad \rightarrow \text{Minimieren}$$

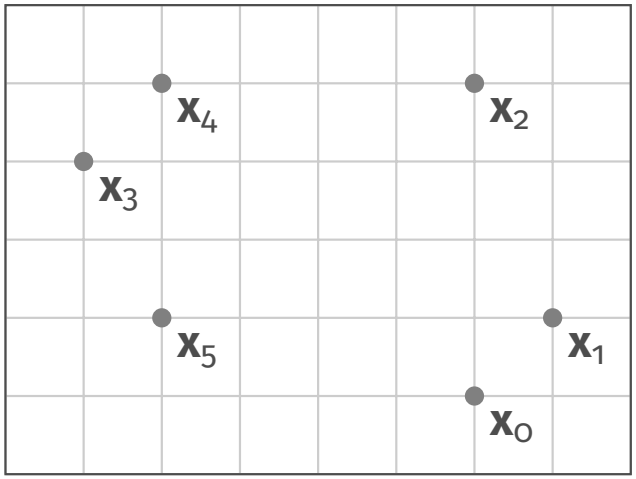
$$q_{outer}(\mathbf{C}) = \sum_{i=1}^{k} \sum_{x \in C_j, j \neq i} d(x, \bar{\mathbf{c}}_j) \quad \rightarrow \text{Maximieren}$$

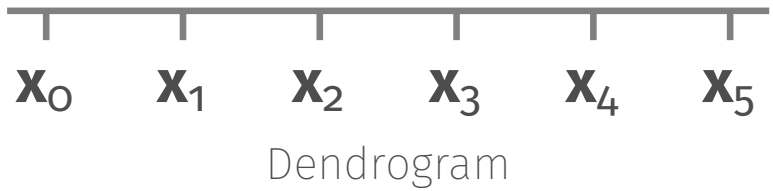
Überblick Clustering-Verfahren

Einordnung von Clustering-Verfahren

- Hierarchisches Clustering, agglomerativ/divisiv
- Iterative Verfahren (z.B. k-Means)
- Dichte-basiertes Clustering (z.B. DBScan)
- Stochastische Verfahren
- Meta-Daten Basierte Verfahren

Hierarchisches Clustering (agglomerativ)

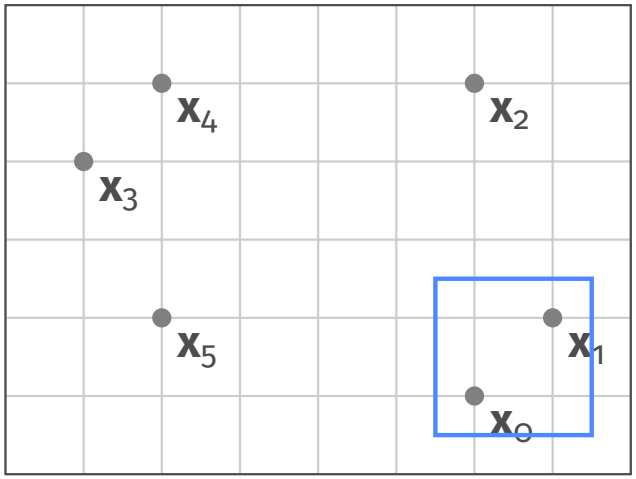


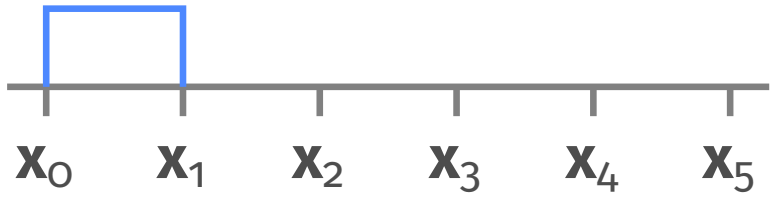


Dendrogram

- Jeder Datenpunkt ist anfangs ein Cluster
- In jedem Schritt werden die nächstgelegenen Cluster zusammengefasst
- Erzeugt Hierarchie von Aufteilungen der Daten

Hierarchisches Clustering (agglomerativ)

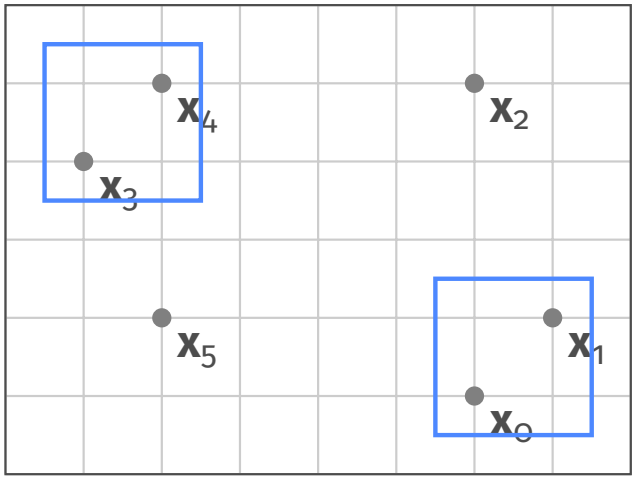


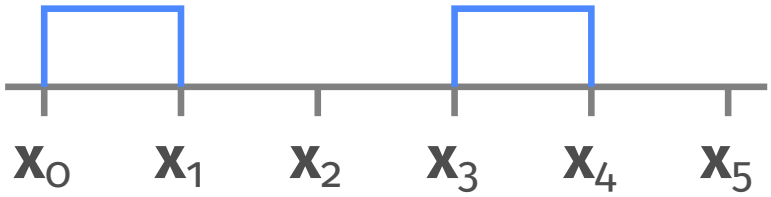


Dendrogram

- Jeder Datenpunkt ist anfangs ein Cluster
- In jedem Schritt werden die nächstgelegenen Cluster zusammengefasst
- Erzeugt Hierarchie von Aufteilungen der Daten

Hierarchisches Clustering (agglomerativ)

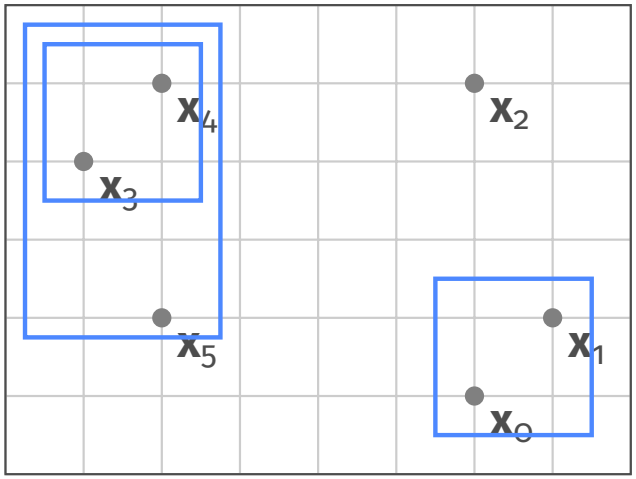


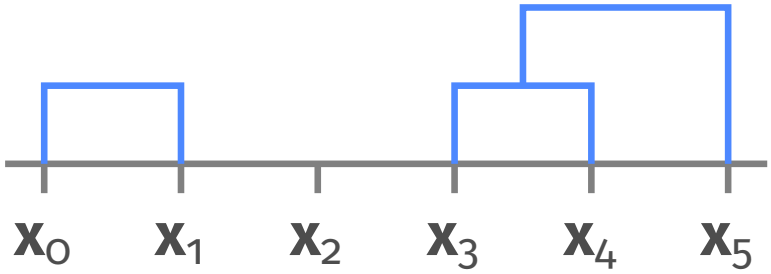


Dendrogram

- Jeder Datenpunkt ist anfangs ein Cluster
- In jedem Schritt werden die nächstgelegenen Cluster zusammengefasst
- Erzeugt Hierarchie von Aufteilungen der Daten

Hierarchisches Clustering (agglomerativ)

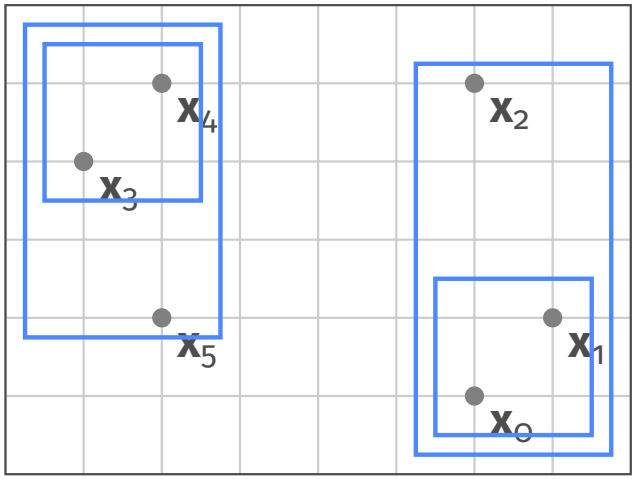


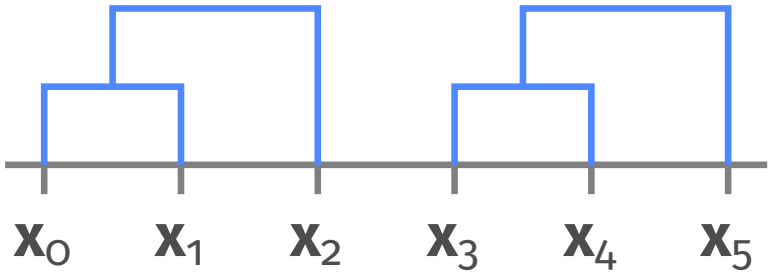


Dendrogram

- Jeder Datenpunkt ist anfangs ein Cluster
- In jedem Schritt werden die nächstgelegenen Cluster zusammengefasst
- Erzeugt Hierarchie von Aufteilungen der Daten

Hierarchisches Clustering (agglomerativ)

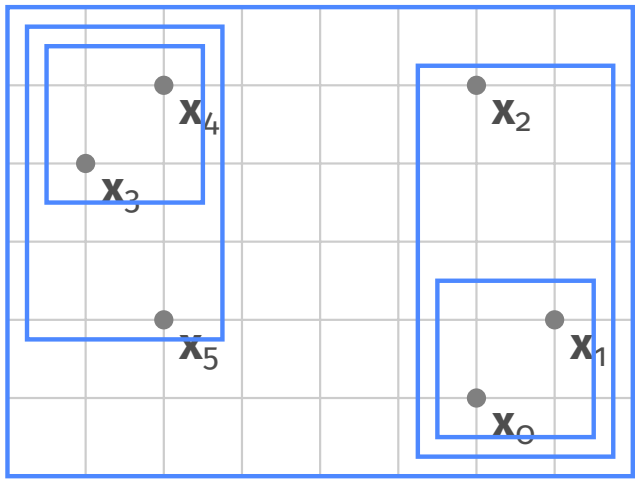


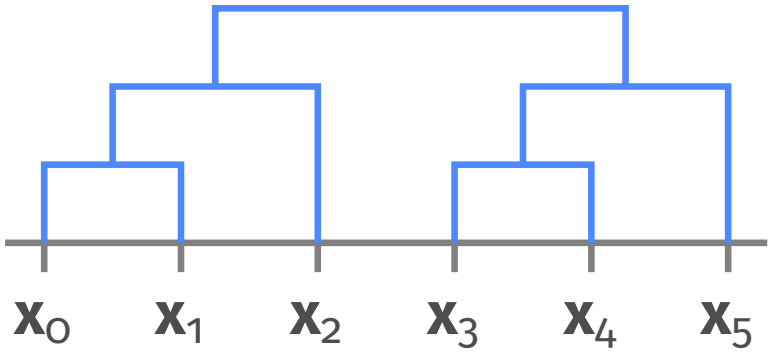


Dendrogram

- Jeder Datenpunkt ist anfangs ein Cluster
- In jedem Schritt werden die nächstgelegenen Cluster zusammengefasst
- Erzeugt Hierarchie von Aufteilungen der Daten

Hierarchisches Clustering (agglomerativ)





Dendrogram

- Jeder Datenpunkt ist anfangs ein Cluster
- In jedem Schritt werden die nächstgelegenen Cluster zusammengefasst
- Erzeugt Hierarchie von Aufteilungen der Daten

Hierarchisches Clustering - nächstgelegene Cluster

Wir müssen in jedem Schritt die beiden nächstgelegenen Cluster **G** und **H** bestimmen, die zusammengefasst werden.

Hierarchisches Clustering - nächstgelegene Cluster

Wir müssen in jedem Schritt die beiden nächstgelegenen Cluster **G** und **H** bestimmen, die zusammengefasst werden.

Beim single linkage wird der Abstand von **G** und **H** definiert als

$$D(\mathbf{G}, \mathbf{H}) = \min_{\mathbf{p} \in G, \mathbf{q} \in \mathbf{H}} d(\mathbf{p}, \mathbf{q})$$

$\Rightarrow$ minimaler Abstand zweier Punkte aus **G** und **H**.

Hierarchisches Clustering - nächstgelegene Cluster

Beim complete linkage wird der **maximale** Abstand zweier Punkte aus **G** und **H** als Abstand der Cluster benutzt:

$$D(\mathbf{G}, \mathbf{H}) = \max_{\mathbf{p} \in G, \mathbf{q} \in \mathbf{H}} d(\mathbf{p}, \mathbf{q})$$

Die Cluster $\mathbf{G}, \mathbf{H}$ mit kleinstem $D(\mathbf{G}, \mathbf{H})$ werden zusammengefasst.

Der k-Means Algorithmus

k-Means Algorithmus

- Distanz-basiertes, iteratives Clustering-Verfahren
- Erzeugt ***k*** disjunkte Teilmengen von **X**
- ***k*** ist Benutzer-Parameter
- Distanzfunktion wird auch von Benutzer gewählt

Algorithmus: k-Means

1. Wähle k zufällige Clusterpunkte $c_1, \ldots, c_k$ aus $\mathbf{X}$
2. Ordne jedes $\mathbf{x} \in \mathbf{X}$ dem nächstgelegenen c_i zu, d.h.

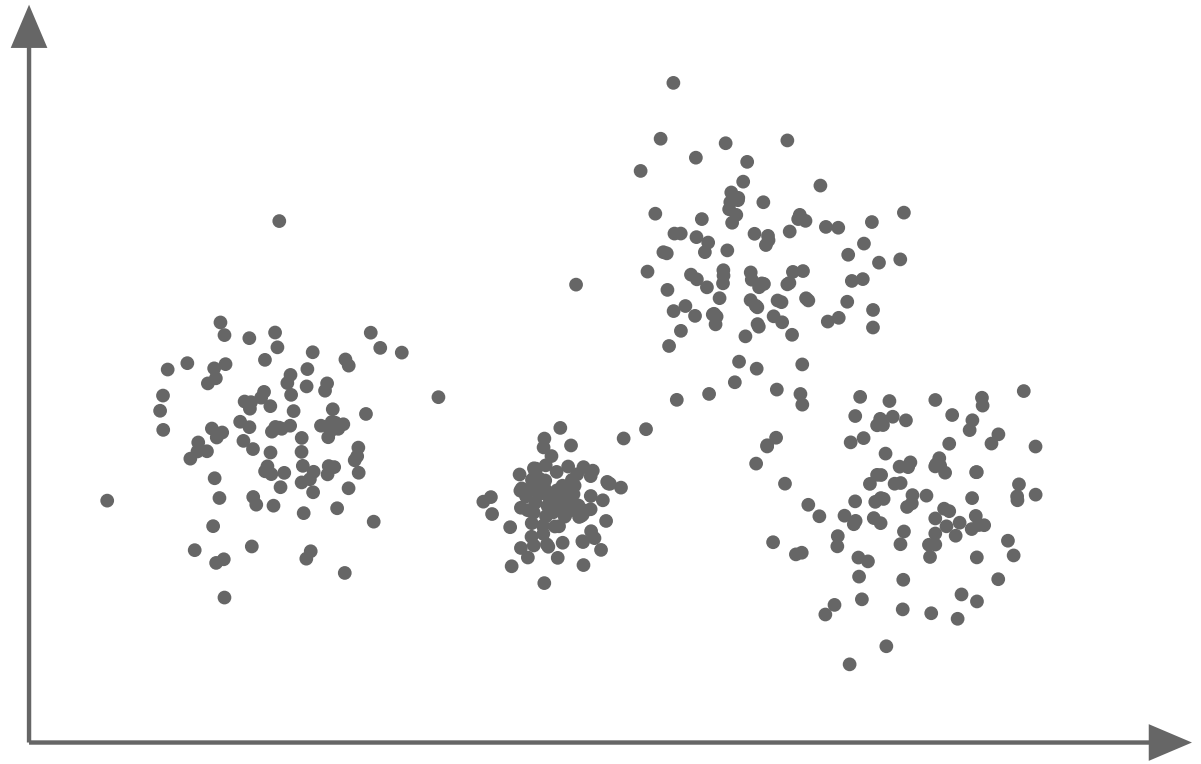
$$\mathbf{C}_i = \{\, \mathbf{x} \in \mathbf{X} \mid \mathbf{x} \text{ am nächsten an } \mathbf{c}_i \,\}$$

3. Berechne neue Clusterpunkte

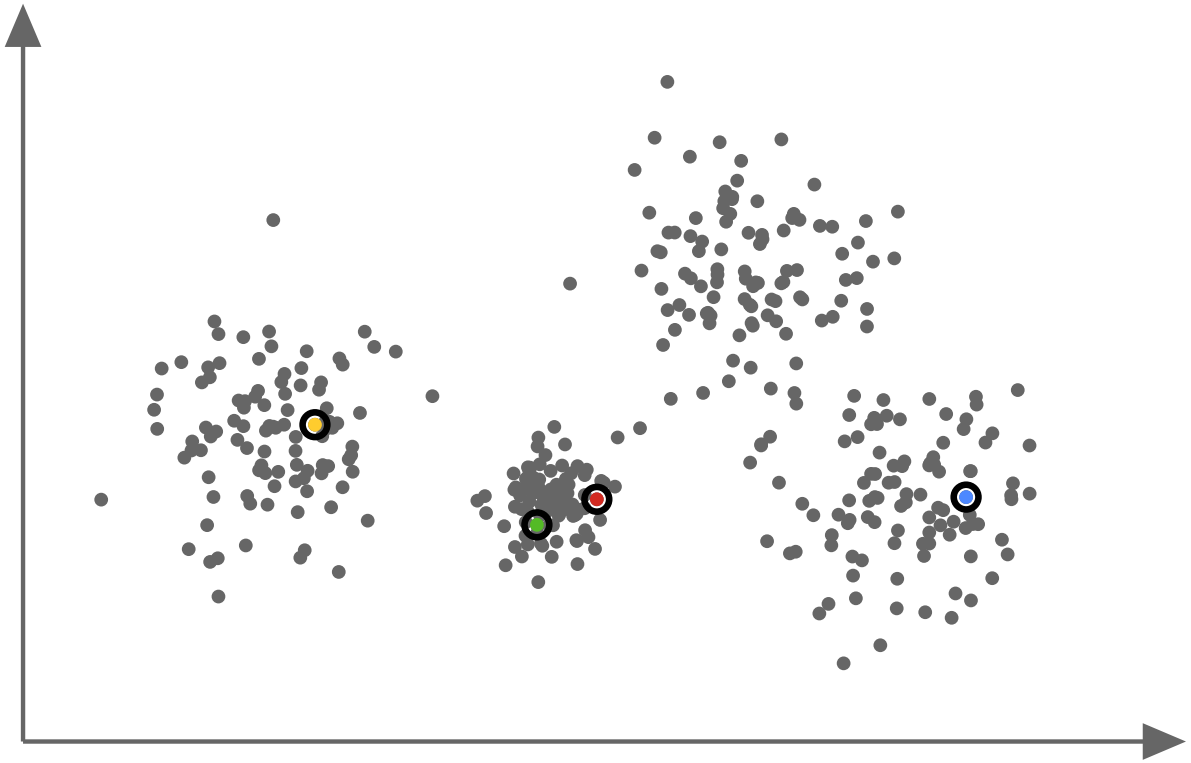
$$\bar{\mathbf{c}}_i = \frac{1}{|\mathbf{C}_i|} \sum_{\mathbf{x_j} \in \mathbf{C}_i} \mathbf{x_j}$$

Wenn $\bar{\mathbf{c}}_1, \ldots, \bar{\mathbf{c}}_k \simeq \mathbf{c}_1, \ldots, \mathbf{c}_k$, dann STOP, sonst springe zu 2. mit den neuen Punkten $\bar{\mathbf{c}}_1, \ldots, \bar{\mathbf{c}}_k$

Beispiel: k-Means

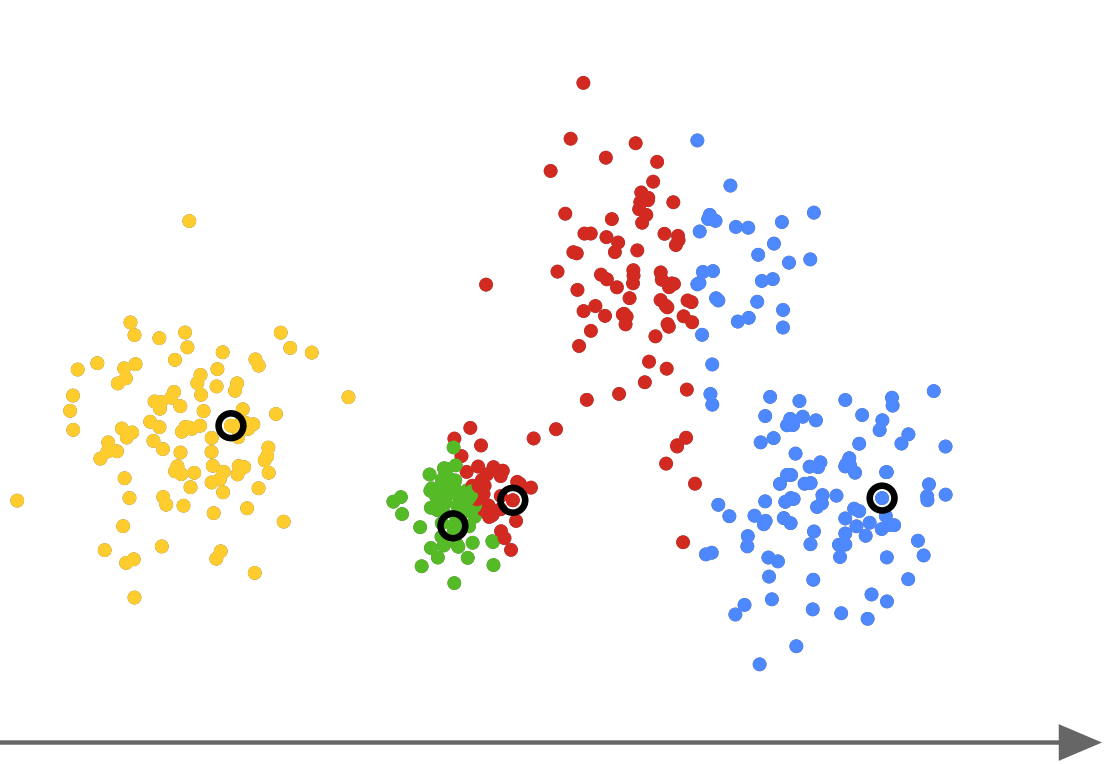


Beispiel: k-Means



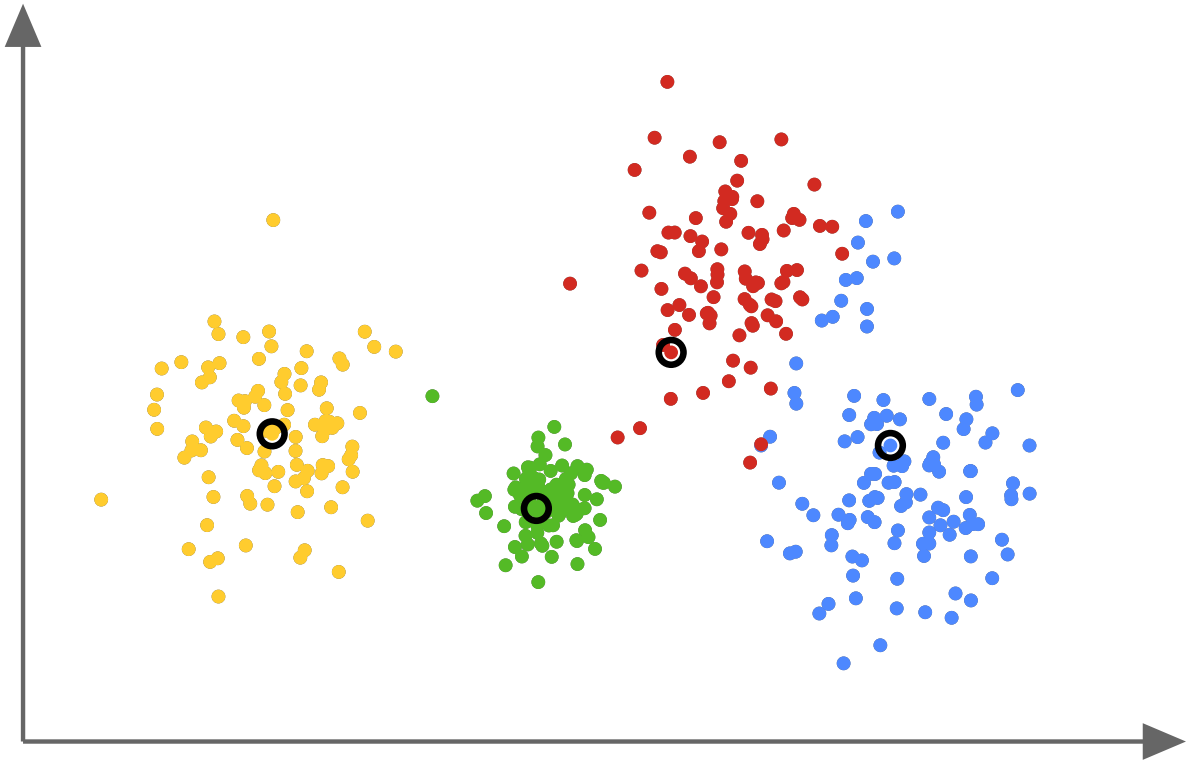
1. Wähle k zufällige Cluster-Mittelpunkte $\mathbf{c}_1, \ldots, \mathbf{c}_k$

Beispiel: k-Means



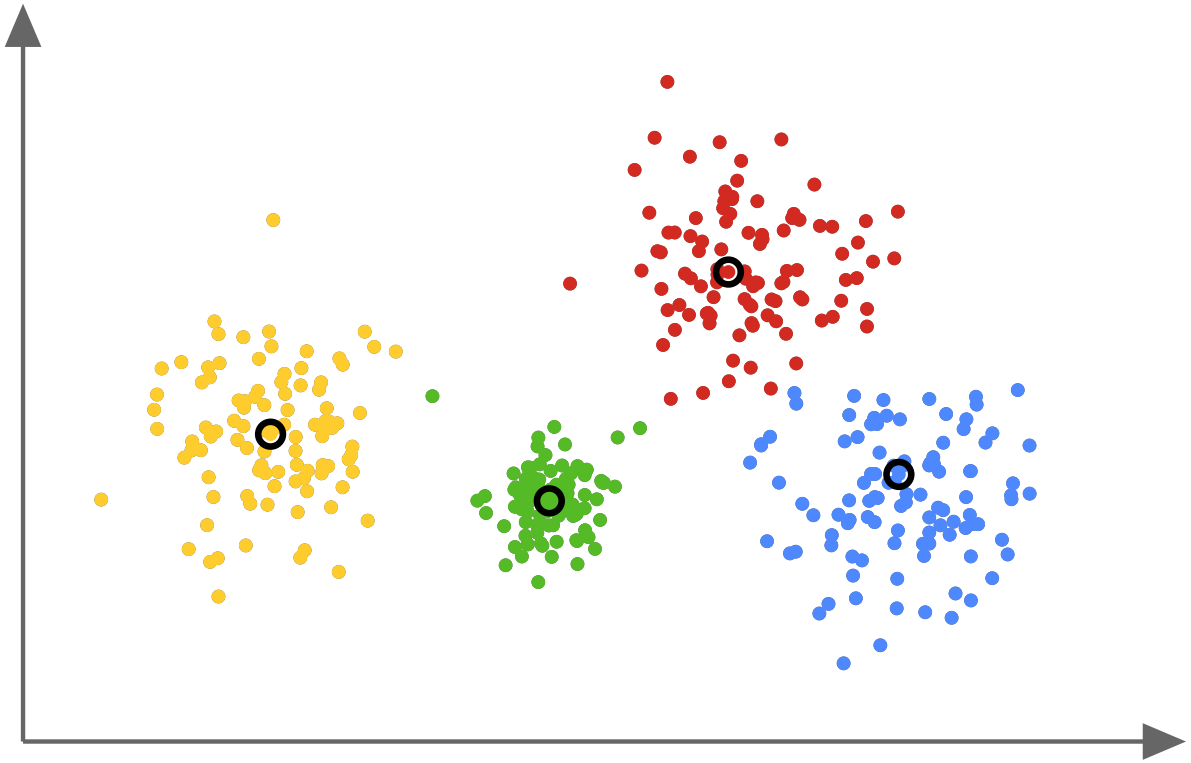
2. Ordne jedem Punkt seinen nächsten Cluster-Mittelpunkt zu

Beispiel: k-Means



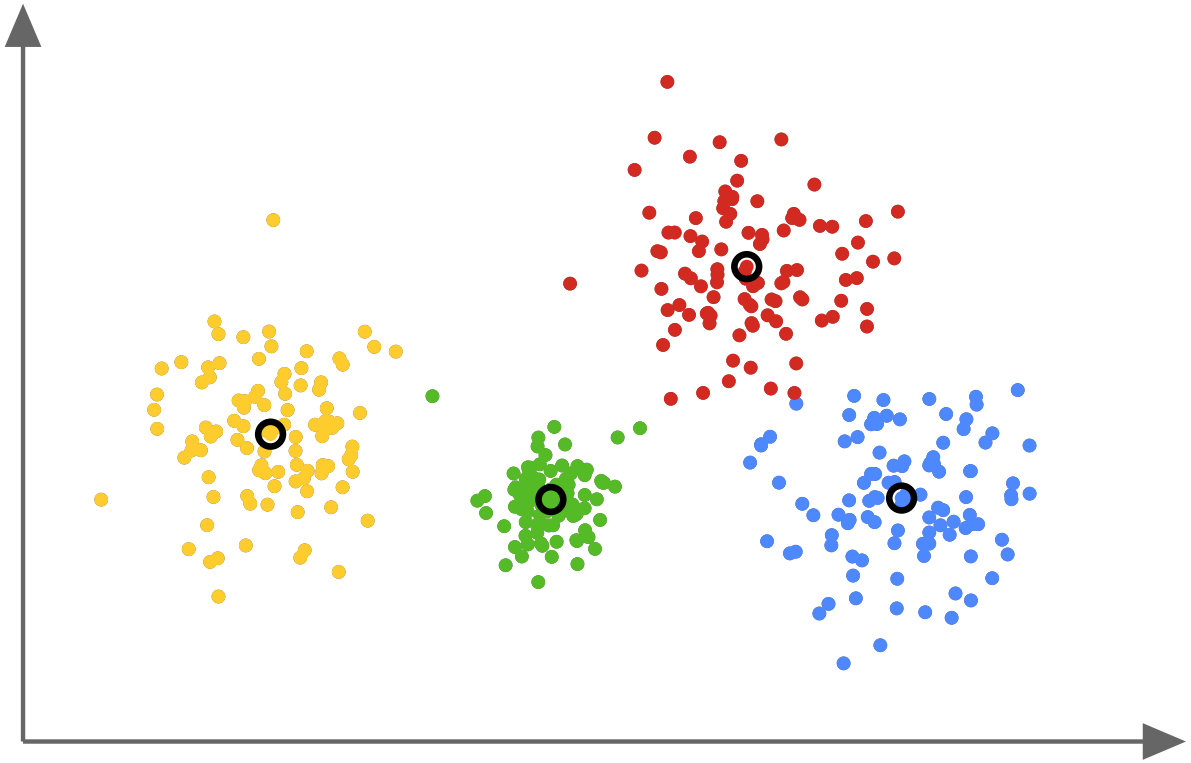
3. Für die Cluster neue Mittelpunkte berechnen, Punkte zuordnen

Beispiel: k-Means



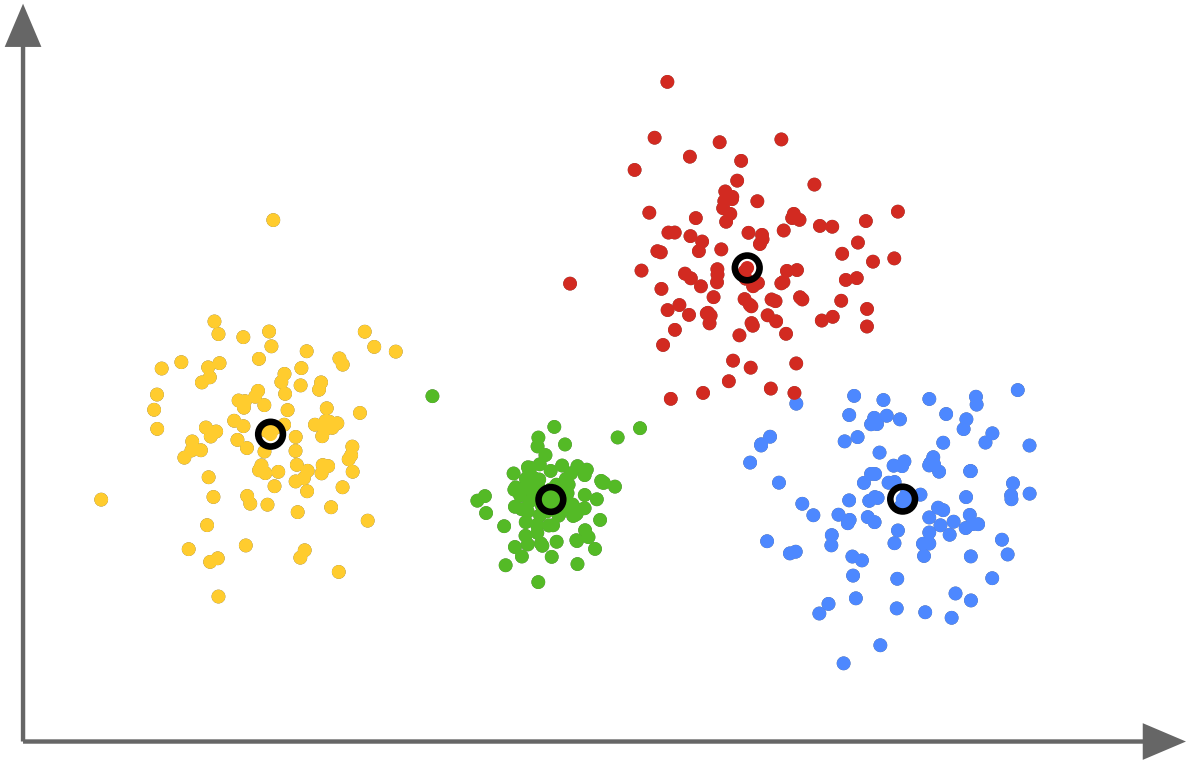
Schritte 2. und 3. wiederholen bis Cluster-Mittelpunkte stabil

Beispiel: k-Means



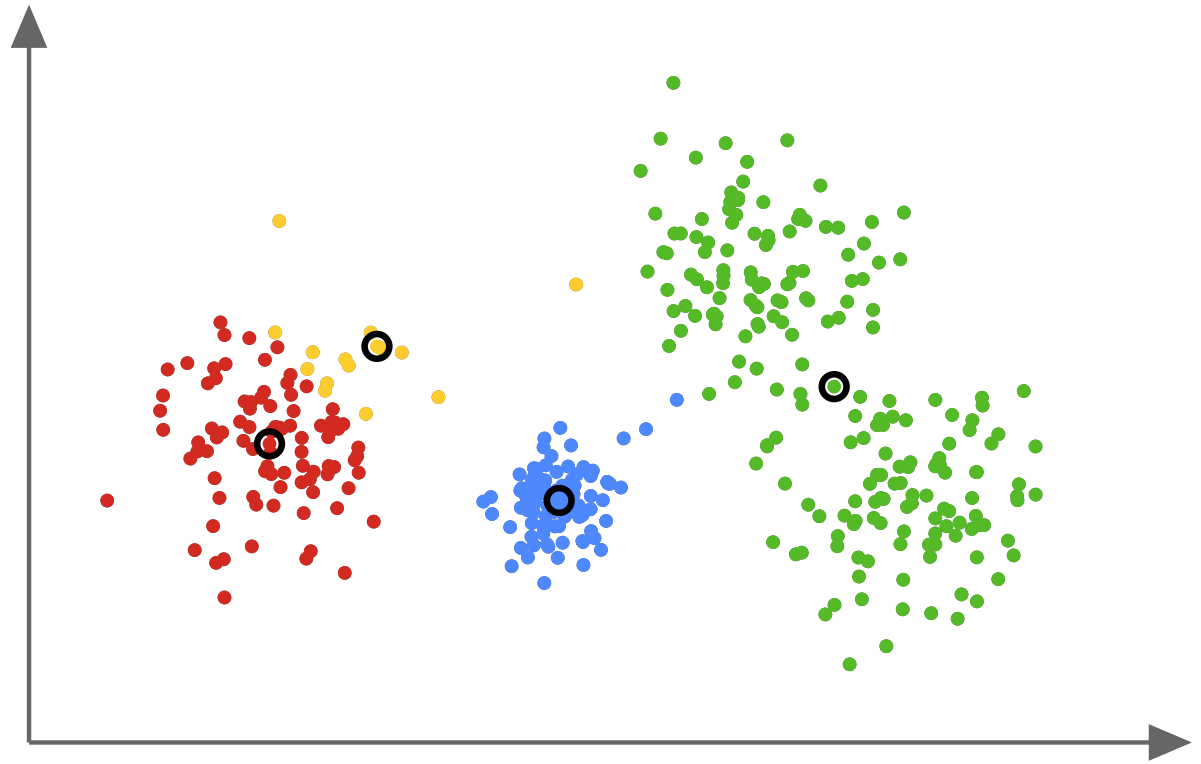
Schritte 2. und 3. wiederholen bis Cluster-Mittelpunkte stabil

Beispiel: k-Means



Schritte 2. und 3. wiederholen bis Cluster-Mittelpunkte stabil

Beispiel: Ergebnis hängt von zufälligen Startpunkten ab



k-Means Parameter

- Unterschiedliche Heuristiken bzgl. der Wahl der Startpunkte
- ggf. mehrfach Starten und Ergebnisse vergleichen

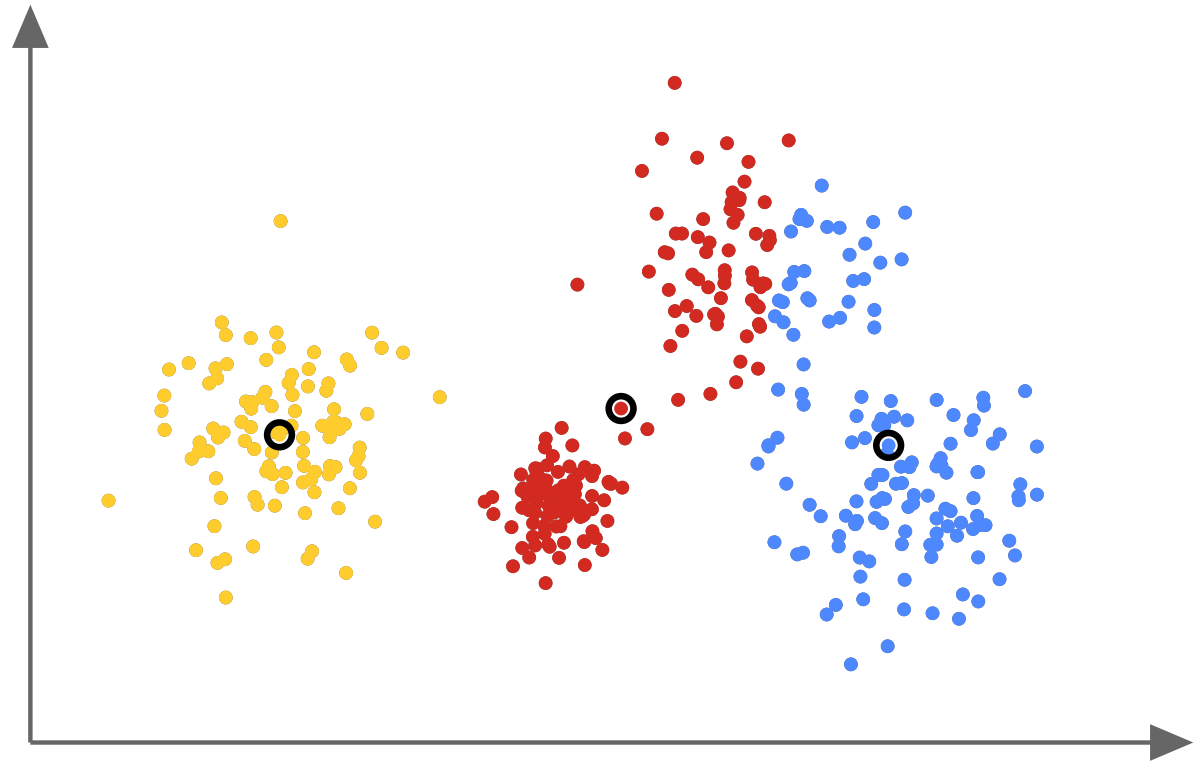
k-Means Parameter

- Unterschiedliche Heuristiken bzgl. der Wahl der Startpunkte
- ggf. mehrfach Starten und Ergebnisse vergleichen

Wie wählen wir Parameter *k*?

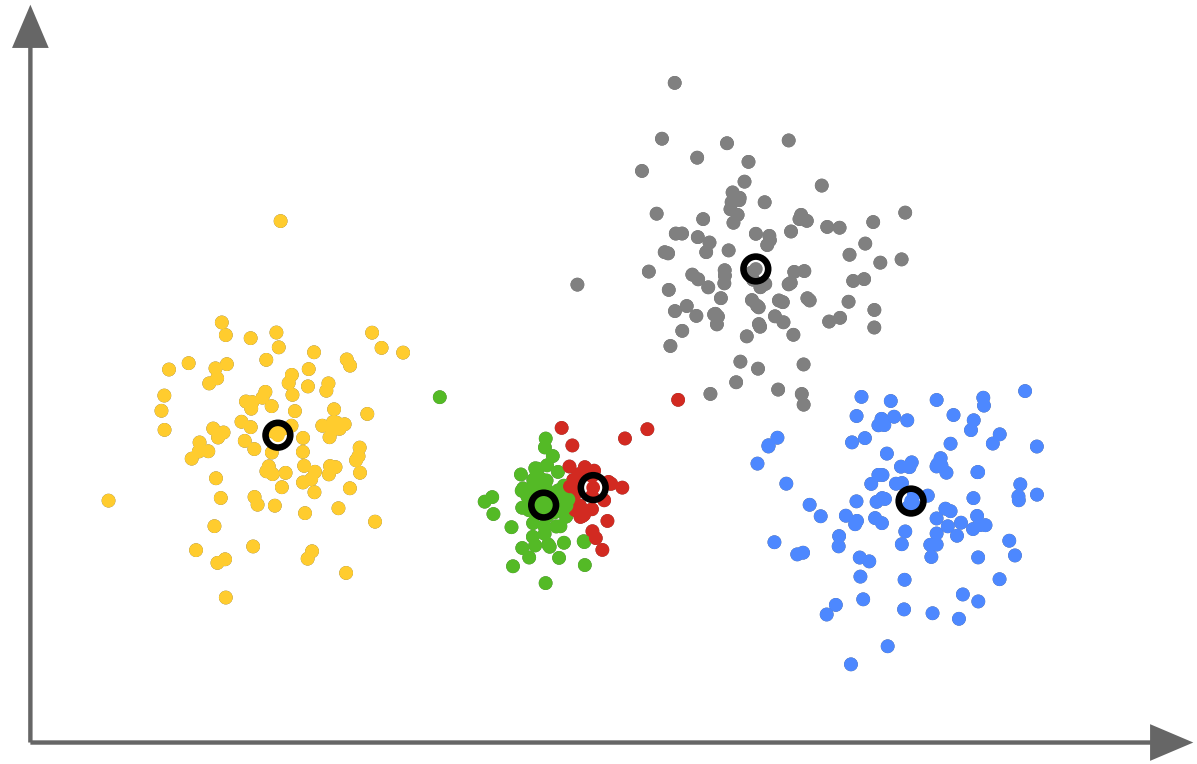
- Keine “So wird’s gemacht Lösung”
- Vorwissen über Daten nutzen
- ***k*** aus der Fragestellung ableiten?

Beispiel: k-Means mit verschiedenem k, gleiche Daten



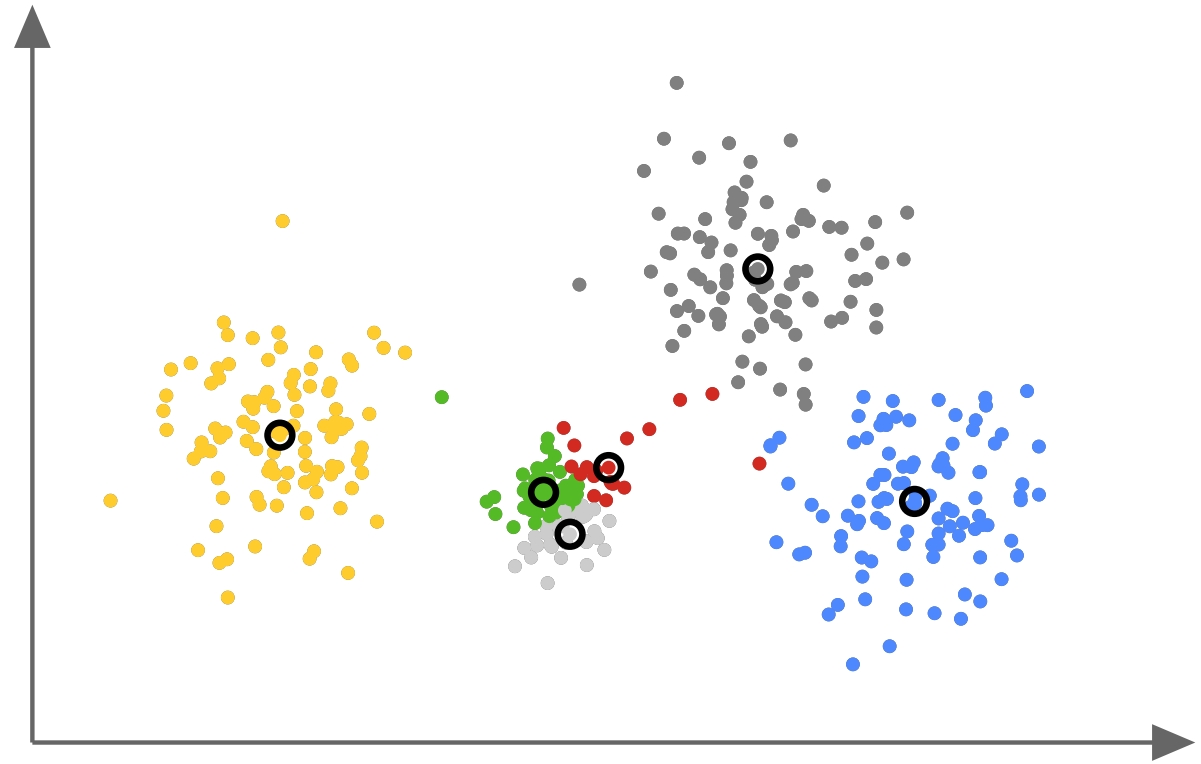
k-Means mit $k = 3$

Beispiel: k-Means mit verschiedenem k, gleiche Daten



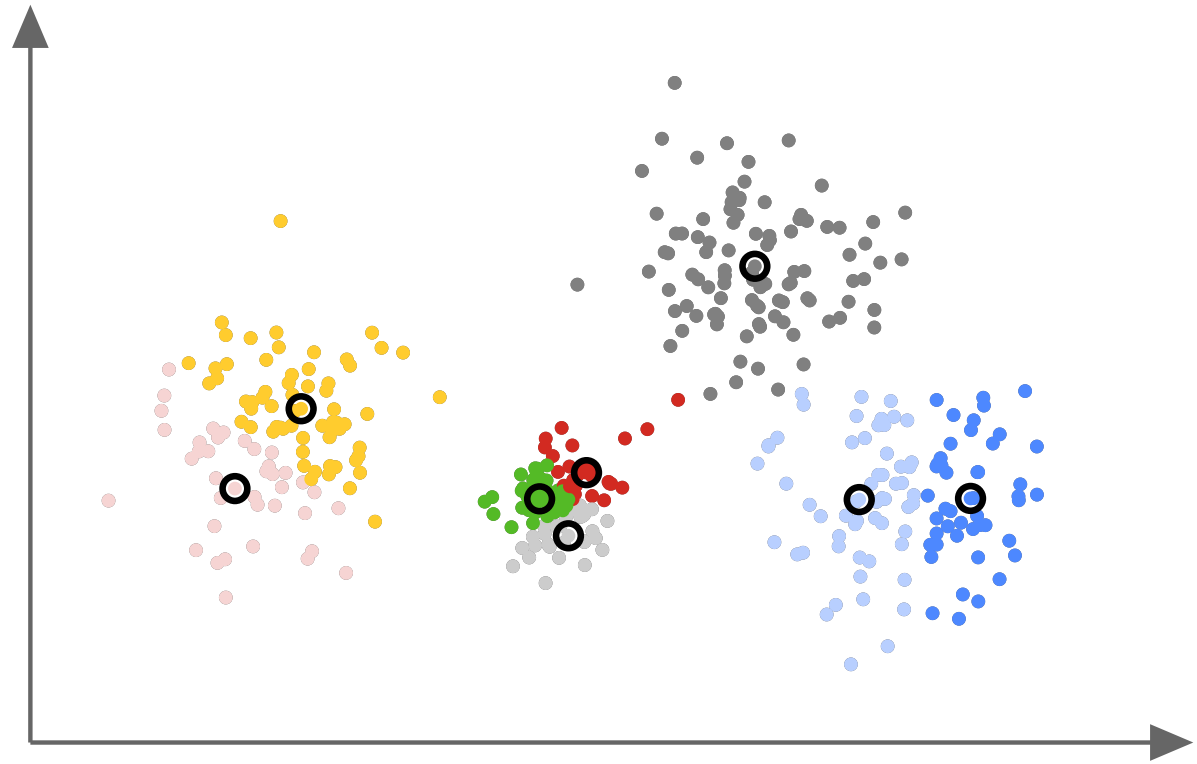
k-Means mit $k = 5$

Beispiel: k-Means mit verschiedenem k, gleiche Daten



k-Means mit $k = 6$

Beispiel: k-Means mit verschiedenem k, gleiche Daten



k-Means mit $k = 8$

k-Means Ergebnis

- ***k***-Means liefert disjunkte Aufteilung der Daten **X**
- Keine “Interpretation” durch Algorithmus

k-Means Ergebnis

- *k*-Means liefert disjunkte Aufteilung der Daten **X**
- Keine “Interpretation” durch Algorithmus
- Cluster nachfolgend manuell inspizieren?

a1	a2	cluster
3.750	**3.500**	**3**
4.075	**3.411**	**5**
3.628	**3.745**	**3**
3.644	**3.581**	**3**

k-Means basiert auf Distanz-Funktion

- Standard: Euklidische Distanz
  (vgl. Data Science 1, Foliensatz 6, Folie 6+)
- Funktioniert nur auf numerischen Attributen

Daten einlesen, numerische Spalten auswählen

Wähle `number` Spalten mit `select_dtypes(..)`:

```
df = pd.read_csv('data/cluster-data.csv')

# Selektiere alle numerischen Spalten:
X = df.select_dtypes(include=['number'])
```

kMeans in Python

```python
from sklearn.cluster import KMeans

# Parameter 'k' = 4
kmeans = KMeans(n_clusters=4)

# Clustering berechnen:
kmeans.fit(X)

# pro Datenpunkt nr des Clusters
clusters = kmeans.predict(X)
```

Clustering Ergebnis ist Liste der Cluster-Nummern:

```
clusters = kmeans.predict(X)

# als Spalte in den DataFrame einfuegen:
df['cluster'] = clusters
```

Bei $k = 4$ enthält `df['cluster']` die Werte $0, \ldots, 3$

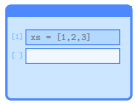
Clustering plotten

Berechne für jedes $i = 0, \ldots, 3$ eine Farbe zum Plotten

```
# Farb-Zuordnung:
mapping = { 0: 'red', 1: 'blue', 2: 'green', 3: '
                                    yellow'}

# Neue Spalte 'color' mit den Farben berechnen:
df['color'] = df['cluster'].replace(mapping)

# Plotten von 'a1', 'a2' mit Farbe 'color'
df.plot.scatter(x='a1', y='a2', c='color')
```



◀ Probieren Sie es im Notebook aus!

Notebook: Kurse/DataScience2/V11-kMeans.ipynb

Clustering von Dokumenten

Beispiel: Gruppierung von Produkten

A bloomy Eau de Parfume for men by ProfumoInCasa.

A earthy Eau de Parfume for men by HappyScent, now vegan!

A intense Eau de Toilette for women by Flavair. Organic and vegan!

Beispiel: Gruppierung von Produkten

A bloomy Eau de Parfume for men by ProfumoInCasa.

A earthy Eau de Parfume for men by HappyScent, now vegan!

A intense Eau de Toilette for women by Flavair. Organic and vegan!

Als Text-Spalten im DataFrame:

ID	Produkt-Beschreibung
0	A bloomy Eau de Parfume for men by ProfumoInCasa.
1	A earthy Eau de Parfume for men by HappyScent, now vegan!
2	A intense Eau de Toilette for women by Flavair. Organic and vegan!

Repräsentation von Texten als Bag of Words

- Ein Attribut für jedes Wort
- Ein Dokument ist Vektor mit Anzahl der enthaltenen Worte

Beispiel:

A bloomy Eau de Parfume for women.

A	bloomy	earthy	Eau	de	Parfume	Toilette	for	men	women
1	1	0	1	1	1	0	1	0	1

Repräsentation von Texten als Bag of Words

- Ein Attribut für jedes Wort
- Ein Dokument ist Vektor mit Anzahl der enthaltenen Worte

Beispiel:

A bloomy Eau de Parfume for women.

A earthy Eau de Toilette for men.

A	bloomy	earthy	Eau	de	Parfume	Toilette	for	men	women
1	1	0	1	1	1	0	1	0	1
1	0	1	1	1	0	1	1	1	0

Wort-Vektoren mit Python/sklearn

```
from sklearn.feature_extraction.text import
                                    CountVectorizer

vectorizer = CountVectorizer()

# Lerne Worte und Anzahlen pro Dokument
X = vectorizer.fit(df['text'])

# Die Liste der Woerter:
words = vectorizer.get_feature_names()

# Erzeuge einen DataFrame mit Wort-Spalten
wf = pd.DataFrame(X.toarray(), columns=words)
```

Bag of Words: Jedes Wort gleich wichtig?

Term Frequency (TF) – Anzahl der Vorkommen eines Wortes (*term*) in Dokument ***d***:

$$tf(t, d) = \text{Häufigkeit von } t \text{ in } d$$

A	bloomy	earthy	Eau	de	Parfume	Toilette	for	men	women
1	1	0	1	1	1	0	1	0	1
1	0	1	1	1	0	1	1	1	0

Bag of Words: Jedes Wort gleich wichtig?

Term Frequency (TF) – Anzahl der Vorkommen eines Wortes (*term*) in Dokument ***d***:

$$tf(t, d) = \text{Häufigkeit von } \boldsymbol{t} \text{ in } \boldsymbol{d}$$

A	bloomy	earthy	Eau	de	Parfume	Toilette	for	men	women
1	1	0	1	1	1	0	1	0	1
1	0	1	1	1	0	1	1	1	0

Idee: Ein Wort, das in nahezu allen Dokumenten vorkommt hat kaum Aussagekraft!

Bag of Words – TF/IDF Darstellung

Sei $\mathcal{D}$ die Menge aller Texte aus dem Datensatz:

- document frequency – in wie vielen Dokumenten ist das Wort t enthalten?

$$df(t) = |\{d \in \mathcal{D} \mid t \in d\}|$$

- inverse document frequency – logarithmierter Anteil der Dokumente, die t enthalten:

$$idf(t) = -\log \frac{df(t)}{|\mathcal{D}|} = \log \frac{|\mathcal{D}|}{df(t)}$$

TF/IDF Darstellung

TF/IDF Wert für Wort ***t*** in Dokument ***d***:

$$tfidf(t,d) = tf(t,d) \cdot idf(t)$$

TF/IDF nimmt Gewichtung der vorherigen Bag of Words Darstellung vor:

A	bloomy	earthy	Eau	de	Parfume	Toilette	for	men	women
0.334	0.470	0	0.334	0.334	0.470	0	0.334	0	0.470
0.334	0	0.470	0.334	0.334	0	0.470	0.334	0.470	0

(Obiger Datensatz besteht nur aus 2 Dokumenten)

TF/IDF mit Python

```
from sklearn.feature_extraction.text
       import CountVectorizer, TfidfTransformer

vec = CountVectorizer()
tfidf = TfidfTransformer()

# erst zaehlen, dann tf/idf gewichten:
x = vec.fit_transform(df['text'])
x = tfidf.fit_transform(x)

words = vec.get_feature_names()

# DataFrame mit Wort-Spalten nach tf/idf
X = pd.DataFrame(x, columns=words)
```

Clustering von Dokumenten

- Vektor-Darstellung von Dokumenten (TF/IDF)
- Clustering z.B. mit k-Means
- Wie bewerten wir die Cluster?

Clustering von Dokumenten

- Vektor-Darstellung von Dokumenten (TF/IDF)
- Clustering z.B. mit k-Means
- Wie bewerten wir die Cluster?
- Exploration z.B. über WordCloud

WordCloud Erzeugung mit Python

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
%matplotlib inline

# cluster auswaehlen:
cluster0 = X[X['cluster'] == 0].sum()

wc = WordCloud()
wc.generate_from_weights( cluster0 )

plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

WordCloud Darstellung für Text-Cluster