



uni

Universität Augsburg
Wirtschaftswissenschaftliche
Fakultät

Clusteranalyse

Projekstudium: Data Mining

Burak Özkan, Simon Rudenko

31.01.2022

Agenda

- 1 Unsupervised Learning - Clusteranalyse
- 2 Fuzzy Algorithmus
- 3 DBSCAN
- 4 Clustering von Songs auf Spotify
 - 4.1 Datenvorbereitung
 - 4.2 Verwendung der Fuzzy und DBSCAN Algorithmen
 - 4.3 Hauptkomponentenanalyse (PCA)
 - 4.4 Silhouette
- 5 Fazit



01

UNSUPERVISED LEARNING – CLUSTERANALYSE

Unsupervised learning - Clusteranalyse

Ansätze des Machine Learnings, (vgl. Roland Schwaiger, Jochim Steinwendner, S.38, 2019)

Unsupervised Learning

- Unüberwachte Lernen
- Ermittlung unbekannter Strukturen und Mustern in den Daten ohne Zielvariable
- Modell: Clustering, Assoziationen,...

Supervised Learning

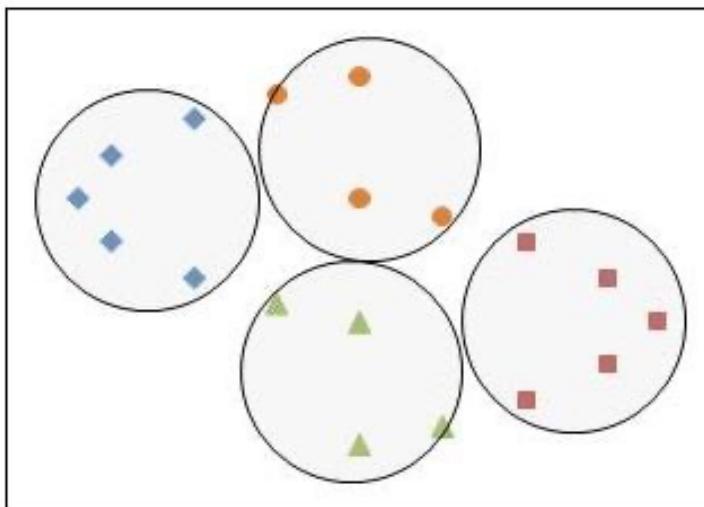
- Überwachte Lernen
- Ermittlung von Funktionen auf Basis von Trainingsdaten, deren Output bekannt ist
- Modell: Neuronale Netze, Klassifikation,...

Unsupervised learning - Clusteranalyse

Allgemeines zum Clustering

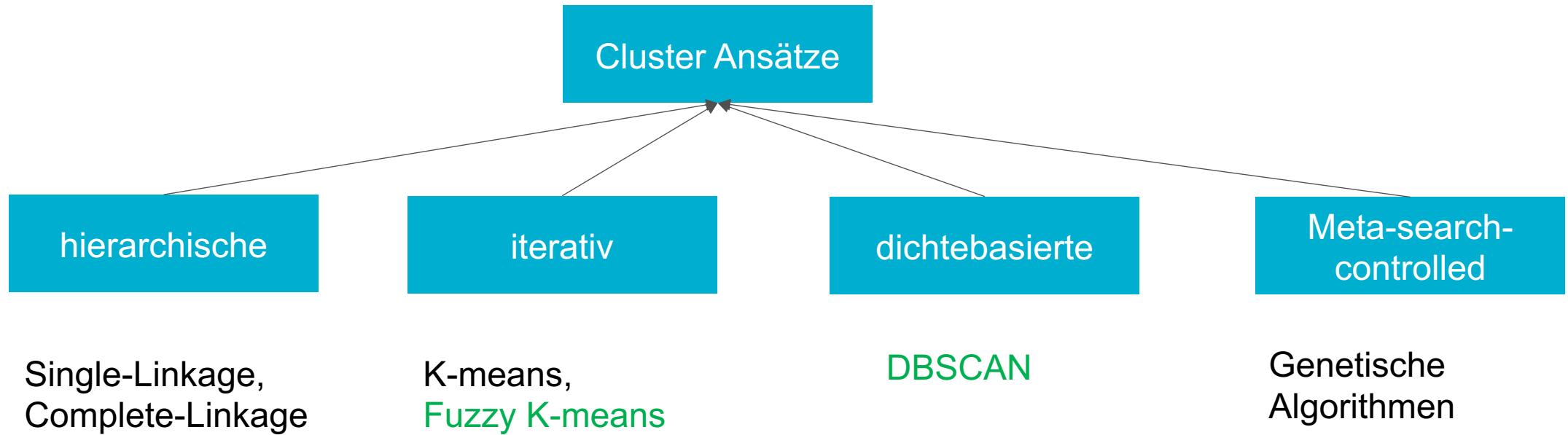
Methode zur Klassenbildung d.h. Einteilung einer Menge von Objekten in kleinere Teilmengen

- Objekte derselben Klasse möglichst ähnlich innerhalb
- Möglichst deutliche Unterscheidung der Objekte, die verschiedenen Klassen zugewiesen werden



Unsupervised learning - Clusteranalyse

Cluster Arten (Benno Stein, Michael Busch: Paper 2005)



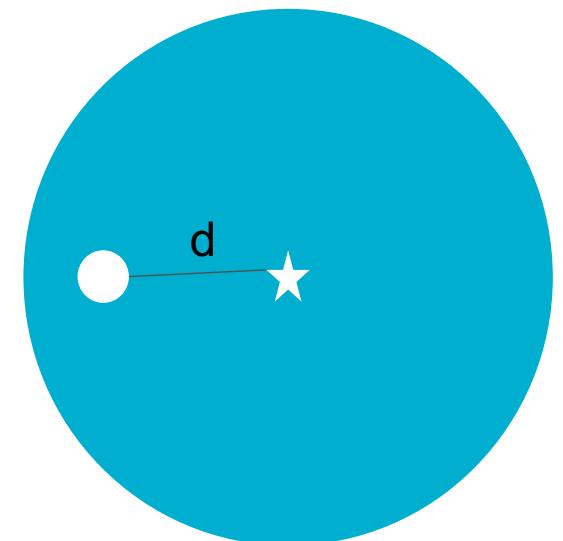
Unsupervised learning - Clusteranalyse

Distanzmaß

- Vorgehensweise: Bestimmung der Ähnlichkeiten/Distanzen zwischen Objekten

Seien x_1, \dots, x_n & y_1, \dots, y_n die J-dimensionalen Merkmalswerte

- Manhattan-Distanz: $d_1(X, Y) = \sum_{j=1}^J |x_j - y_j|$
- Euklidische Distanz: : $d_2(X, Y) = \sqrt{\sum_{j=1}^J (x_j - y_j)^2}$
- Mahalanobis-Distanz: $d_3(X, Y) = (x - y)' \Sigma^{-1} (x - y)$





02

FUZZY ALGORITHMUS

Fuzzy Clustering

Wiederholung: K-means Algorithmus (vgl. Jörg Frochte, 2020, S.309-310)

Idee:

- Die Zahl der gewünschten Gruppen ist vorgegeben
- Clusterbildung minimiert die Summe der Distanzen von den einzelnen Objekten zu den Clusterzentren
- n Objekte auf k Klassen zu verteilen

Distanzmaß:

- Euklidische Distanz: $d_2(x_m, x_l) = \sqrt{\sum_{j=1}^J (x_{mj} - x_{lj})^2}$

Fuzzy Clustering

Wiederholung: K-means Algorithmus (vgl. Jörg Frochte, 2020, S.309)

Ziel:

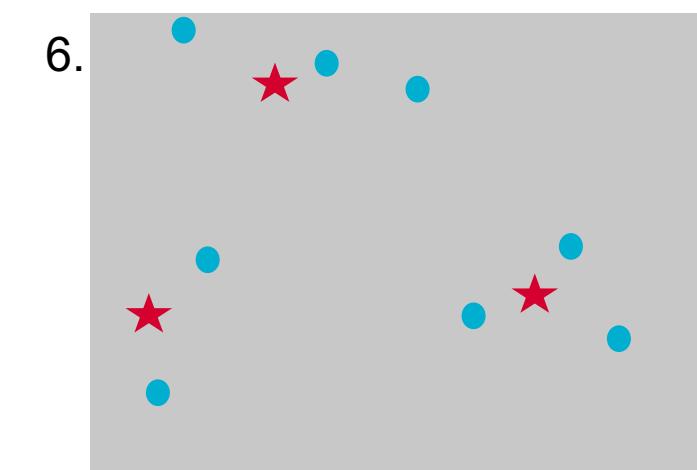
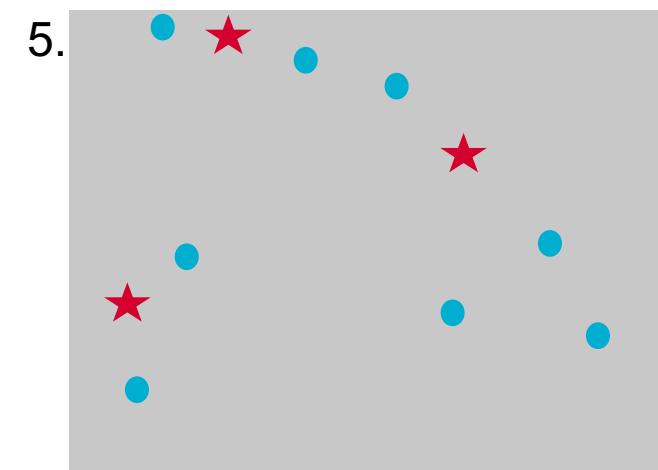
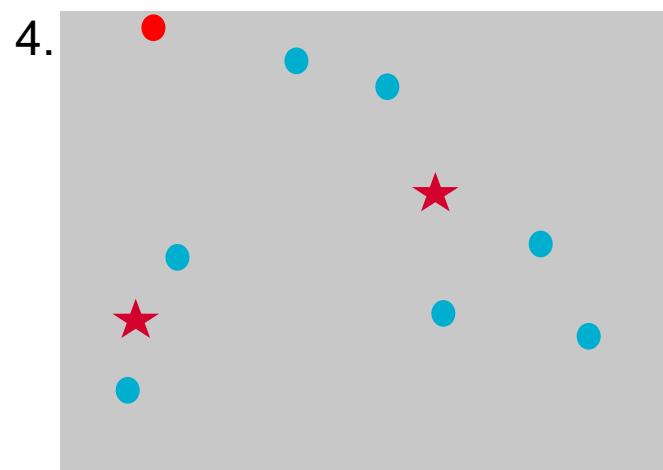
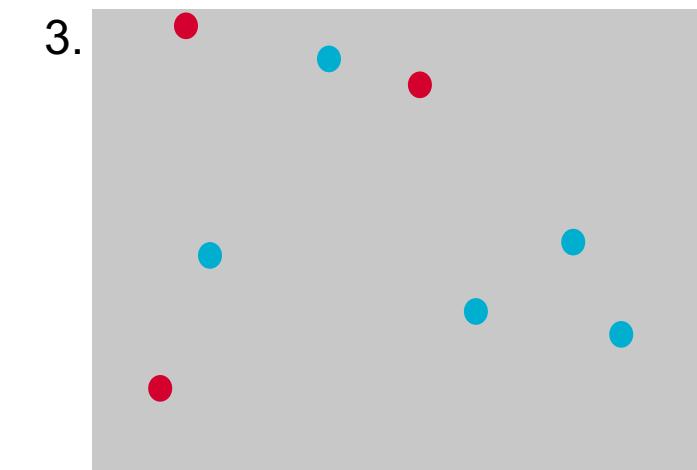
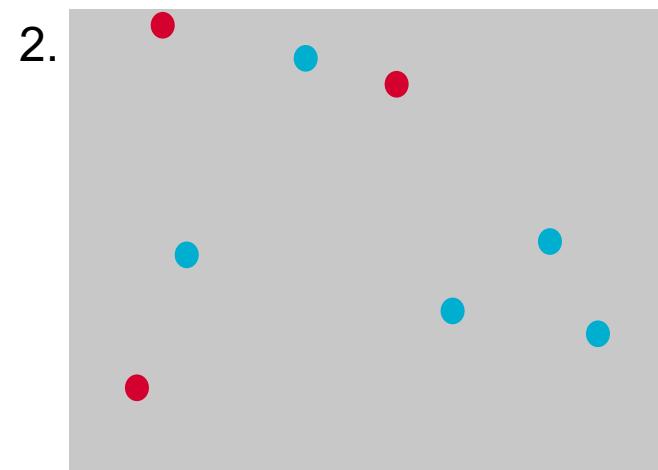
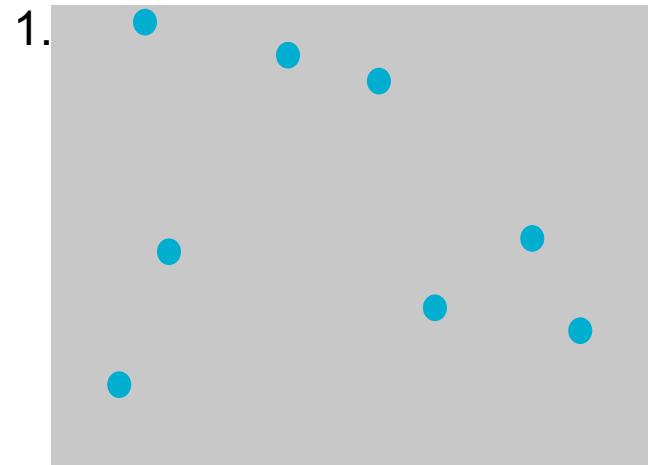
- Optimale Clusterzugehörigkeit mit den besten Zentren
- Zielfunktion: $J = \min \sum_{i=1}^K \sum_{x_i \in C_j} d_2(x_i, m_j)$

Vorgehen:

1. Initialisierung: Setze k Clusterzentren auf zufällige Werte aus dem Datensatz
2. (Neu-)Zuordnung: Ordne jedes Objekt dem nächsten Clusterzentrum zu
3. Neuberechnung: Clusterzentren als Mittelwert der ihm zugeordneten Objekt neu berechnen
4. Wiederholen von Neuzuordnung und Neuberechnung bis Zuordnung unverändert bleibt

Fuzzy Clustering

Wiederholung: K-means Algorithmus mit $k = 3$



Fuzzy Clustering

Einführung in den Algorithmus

k-Means

- Zuordnung des Objekts einem eindeutigen Clusterzentrum
 - Klassische Logik in der Clusterzugehörigkeit: True (1) und False (0)

Fuzzy k-Means

- Keine eindeutige Zuordnung eines Clusterzentrums, sondern Zuordnung des Objekts mit einem bestimmten Gewicht (Fuzzy Logik).
 - Wahrscheinlichkeit  stärke der Zugehörigkeit eines bestimmten Clusters
 - Fuzzy Logik Aussage zwischen 0 und 1
 - Übertragung der Fuzzy Ansätze in das Clustering

Fuzzy Clustering

Annahme (vgl. Jörg Frochte, 2020, S.314)

- n Datensätze und K Cluster sollen gebildet werden
- Fuzzy Ansatz benötigt für jeden der n Datensätze somit K Werte
- Damit ergibt sich in welchem Maße der Datensatz zu einem Cluster zugehörig ist eine Matrix $W \in \mathbb{R}^{c \times n}$
 - Matrix enthält Werte zwischen 0 und 1
 - Grad der Zugehörigkeit w_{ij} , dem sich der Datensatz j dem Cluster i zugehörig fühlt

Alle Basisideen des k-Means bleiben erhalten: $\min \sum_{i=1}^K \sum_{j \in C_i} d_2(x_i, m_j)$

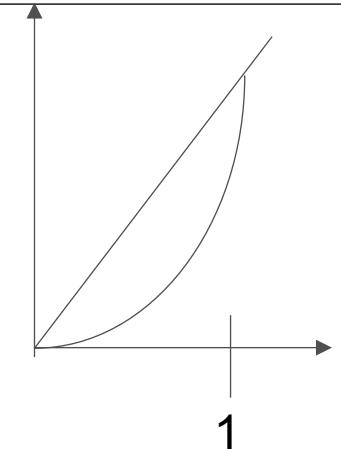
- Zielfunktion: $J = \min \sum_{i=1}^K \sum_{j=1}^n (w_{ij})^m d_2(x_i, m_j)$
 - Fuzzy-Zugehörigkeit fließt in das Funktional ein
 - Komplexe Rolle von m (Fuzzifier)

Fuzzy Clustering

Fuzzifier (vgl. Jörg Frochte, 2020, S.314)

$$J = \min \sum_{i=1}^K \sum_{j=1}^n (\mathbf{w}_{ij})^m d_2(\mathbf{x}_i, \mathbf{m}_j)$$

- Fuzzifier $m \geq 1$
- Wahl von m verändert, wie scharf die Zugehörigkeit zu Clustern gewertet wird
- Je größer m , desto stärker werden die Werte kleiner 1 reduziert
- Großes m führt zu unschärferen Clustern
- $m = 1$ entspricht dem k-Means
- $m > 3$ unüblich und wenig erfolgsversprechend
- In der Regel Wahl bei $m = 2$, solange kein spezieller Grund vorliegt



Fuzzy Clustering

Nebenbedingungen des Algorithmus (vgl. Jörg Frochte, 2020, S.314)

- Beim k-Means: Cluster dürfen nicht leer sein (bleibt beim Fuzzy erhalten)
$$(1) \sum_{j=1}^n w_{ij} > 0 \text{ für alle } i = 1, \dots, K$$

Neue Nebenbedingung (2)

- Summe der Zugehörigkeiten zu den Clustern für jeden Datensatz
$$\sum_{i=1}^c w_{ij} = 1 \text{ für alle } j = 1, \dots, n$$
- Damit bezieht sich Nebenbedingung 1 auf die Spalten und Nebenbedingung 2 auf die Zeilen der Matrix W

Fuzzy Clustering

Vorgehensweise (vgl. Jörg Frochte, 2020, S.315-316)

- 1. Initialisiere k Repräsentanten μ_i für die Cluster
- 2. Berechne für jedes Element bzgl. jedes Cluster ein Maß der Zugehörigkeit mittels

$$w_{ij} = \frac{1}{\sum_{k=1}^K \left(\frac{\|x_i - \mu_k\|}{\|x_i - \mu_j\|} \right)^{\frac{2}{m-1}}}$$

- 3. Berechne durch gewichtete Mittelwertbildung die neuen Repräsentanten μ_i der Cluster

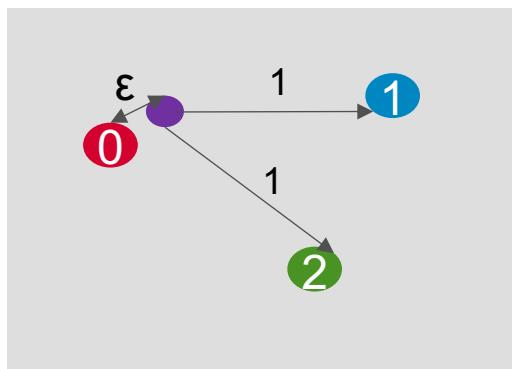
$$\mu_i = \sum_{j=1}^n \frac{(w_{ij})^m}{\sum_{j=1}^n (w_{ij})^m} x_j$$

Merke:

- Großer Abstand -> kleines Gewicht
- Kleiner Abstand -> großes Gewicht

Fuzzy Clustering

Beispiel 1: $w_{lila,0}$; $m = 2$



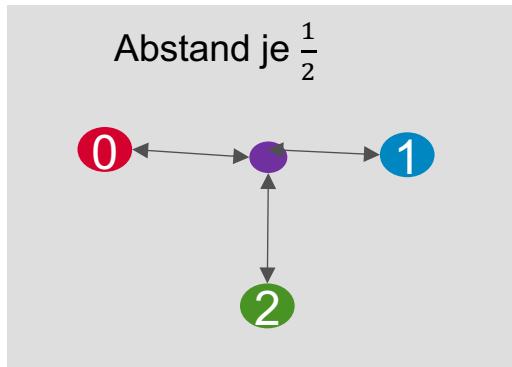
$$\frac{1}{\left(\frac{\varepsilon}{\varepsilon}\right)^2 + \left(\frac{\varepsilon}{1}\right)^2 + \left(\frac{\varepsilon}{1}\right)^2} \approx 1$$

$$w_{ij} = \frac{1}{\sum_{k=1}^K \left(\frac{\|x_i - \mu_k\|}{\|x_i - \mu_j\|} \right)^{m-1}}$$

Wahrscheinlichkeit:

[$w_{lila,0} = 0.989$, $w_{lila,1} = 0.01$, $w_{lila,2} = 0.001$]

Beispiel 2: $w_{lila,0}$; $m = 2$



$$\frac{1}{\left(\frac{0.5}{0.5}\right)^2 + \left(\frac{0.5}{0.5}\right)^2 + \left(\frac{0.5}{0.5}\right)^2} \approx \frac{1}{3}$$

Wahrscheinlichkeit:

[$w_{lila,0} = \frac{1}{3}$, $w_{lila,1} = \frac{1}{3}$, $w_{lila,2} = \frac{1}{3}$]

Fuzzy Clustering

Vorgehensweise

- 1. Initialisiere k Repräsentanten μ_i für die Cluster
- 2. Berechne für jedes Element bzgl. jedes Cluster ein Maß der Zugehörigkeit mittels

$$w_{ij} = \frac{1}{\sum_{k=1}^K \left(\frac{\|x_i - \mu_j\|}{\|x_i - \mu_k\|} \right)^{\frac{2}{m-1}}}$$

- 3. Berechne durch gewichtete Mittelwertbildung die neuen Repräsentanten μ_i der Cluster

Merke:

- Großer Abstand -> kleines Gewicht
- Kleiner Abstand -> großes Gewicht

Fuzzy Clustering

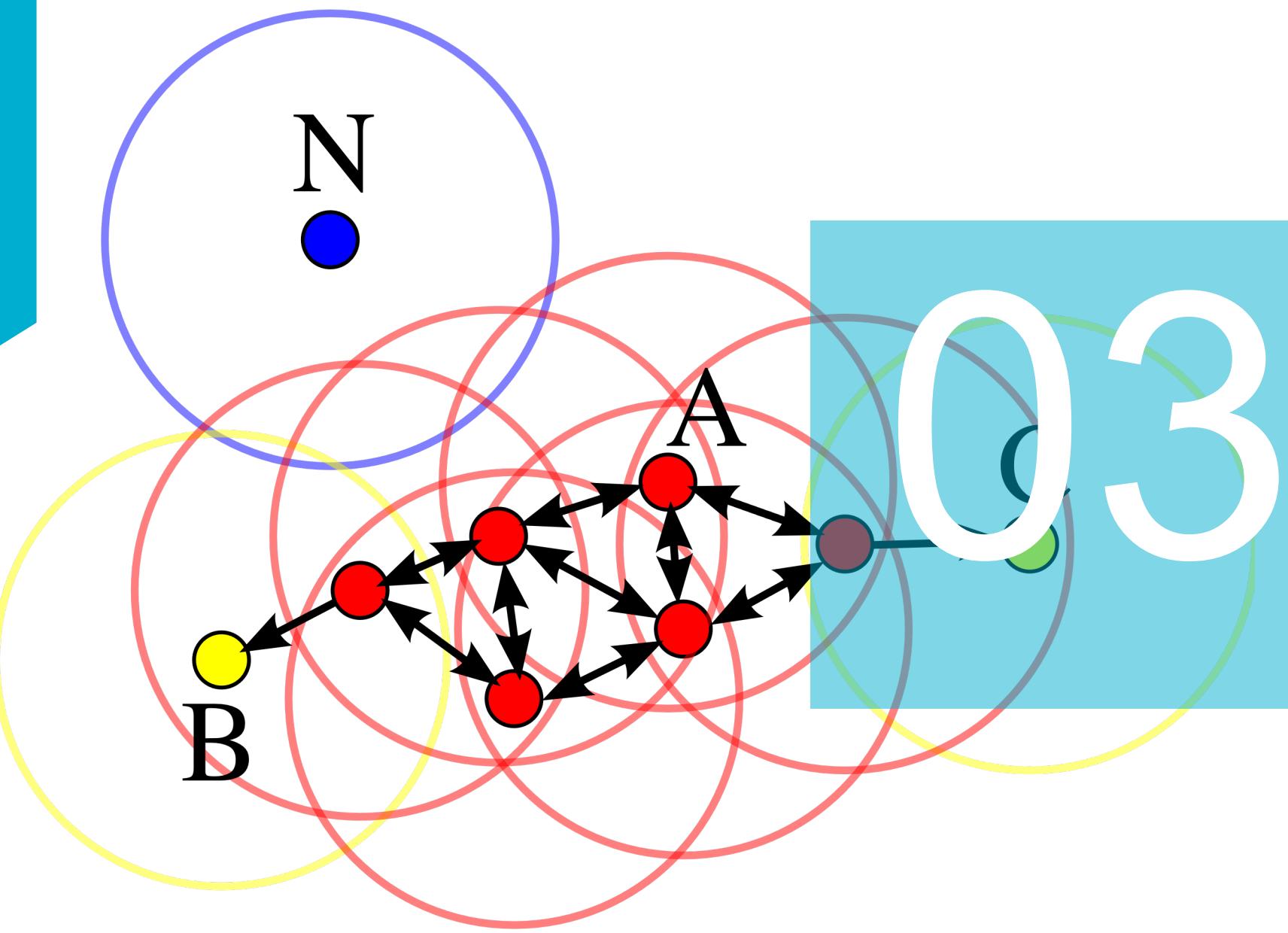
Vor- und Nachteile



- Relativ einfach zu implementieren
- Ermöglicht einen Datenpunkt, sich in mehreren Clustern zu befinden
- Erkennung einer Struktur von komplexen Datensätzen



- Probleme bei der Verwendung von kategorialen Merkmalen
- Abhängig von Anfangsdaten
- Wahl der Anzahl der Cluster

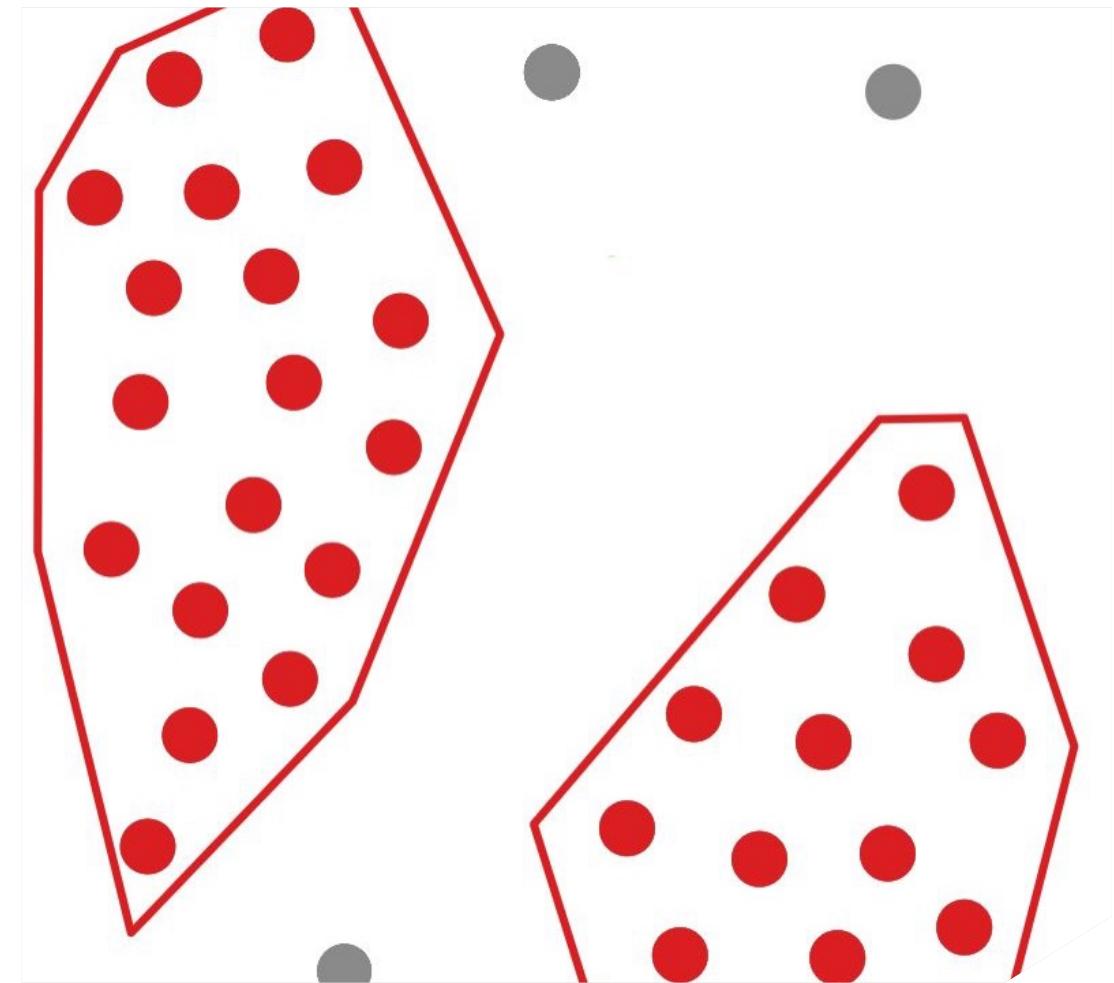


DBSCAN
ALGORITHMUS

DBSCAN

Eigenschaften (Ester et al, 1996)

- dichtebasierter Ansatz
 - Punkte innerhalb eines Clusters sind dichter als außerhalb
→ bestimmte **Anzahl an Punkten** müssen innerhalb eines **Radius** sein
- kann Rauschen (Noise) identifizieren



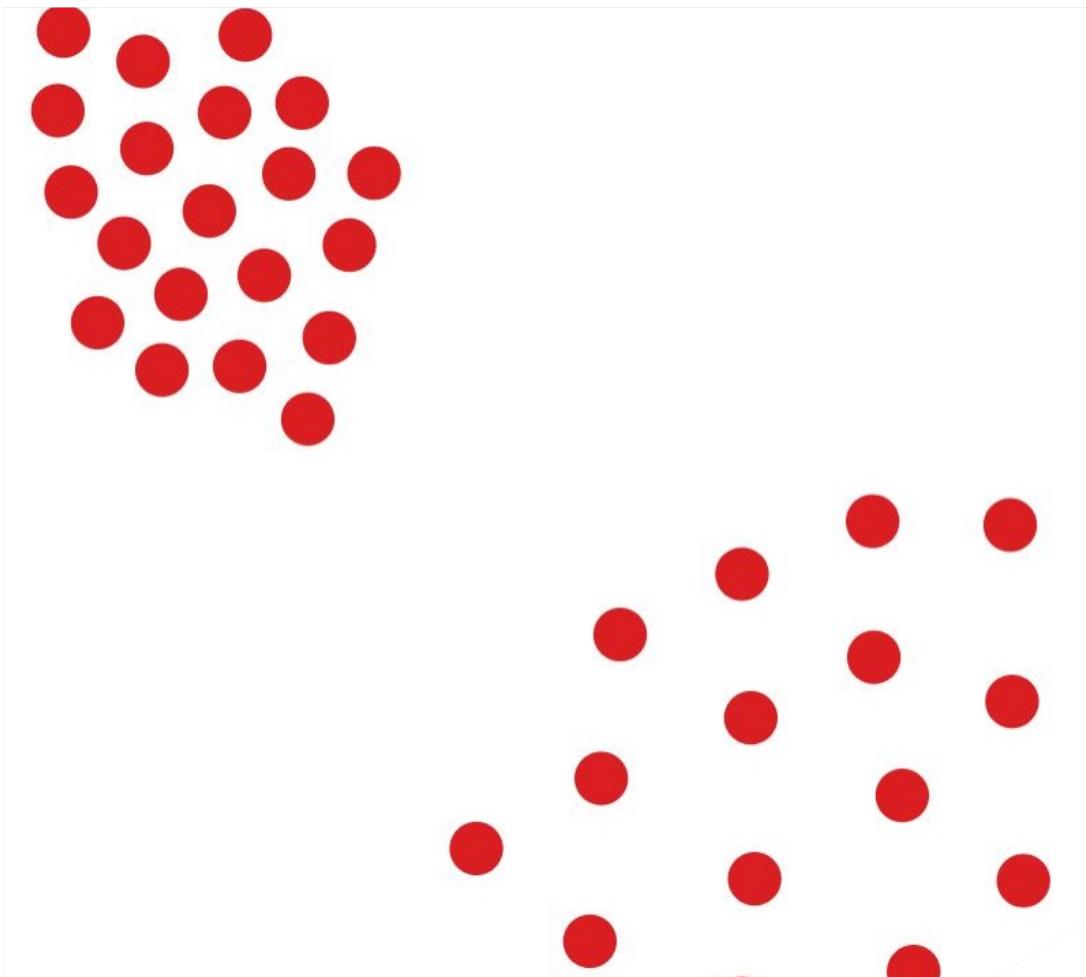
DBSCAN

Eigenschaften (Ester et al, 1996)

- dichtebasierter Ansatz
 - Punkte innerhalb eines Clusters sind dichter als außerhalb
→ bestimmte **Anzahl an Punkten** müssen innerhalb eines **Radius** sein
- kann Rauschen (Noise) identifizieren

Problem: für jedes Cluster werden die selben Parameter verwendet

→ wähle Parameter aus dem „thinnest“ Cluster



DBSCAN

Eigenschaften (Ester et al, 1996)

- dichtebasierter Ansatz
 - Punkte innerhalb eines Clusters sind dichter als außerhalb
→ bestimmte **Anzahl an Punkten** müssen innerhalb eines **Radius** sein
- kann Rauschen (Noise) identifizieren

Problem: für jedes Cluster werden die selben Parameter verwendet

→ wähle Parameter aus dem „thinnest“ Cluster

auch geeignet für Cluster mit nicht-konvexer Form



k-Means

DBSCAN

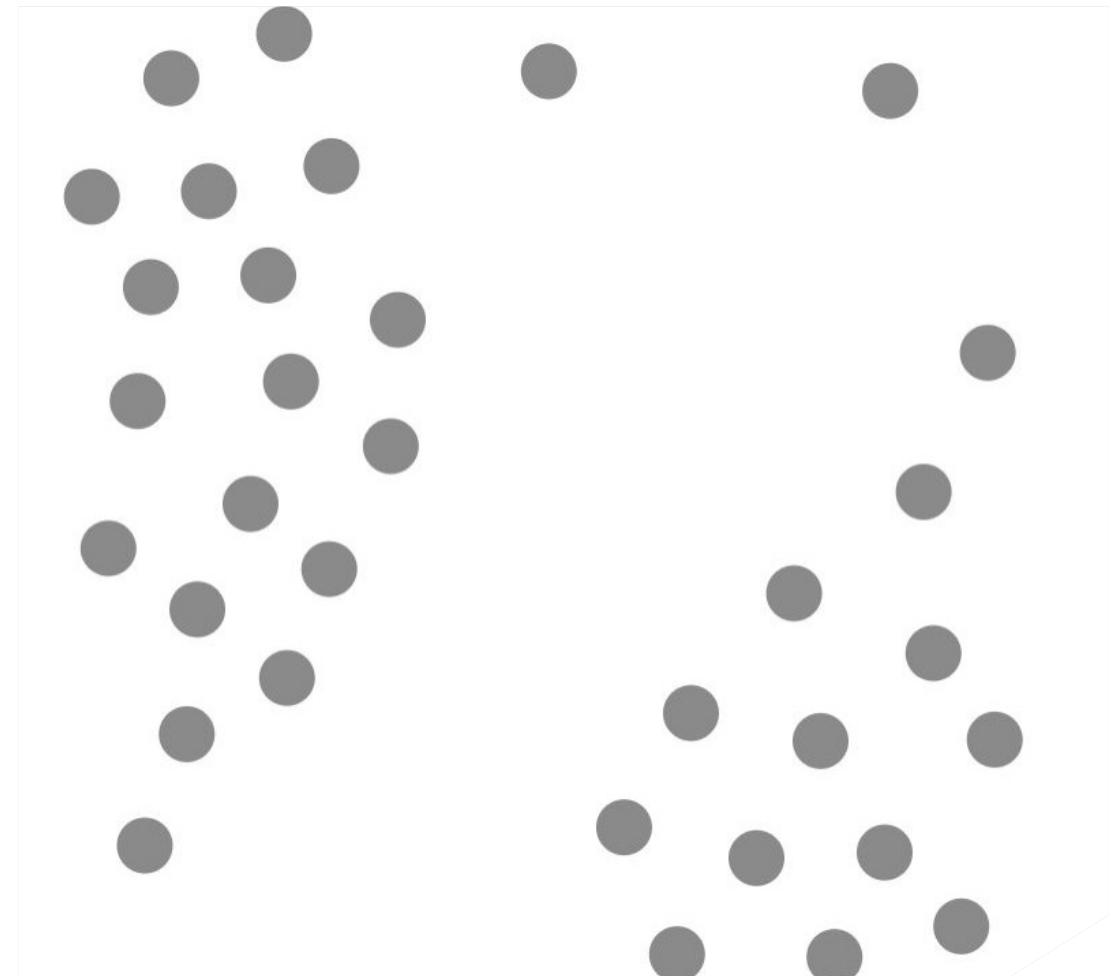


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                      // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                    // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                      // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue           // Core-point check
16    S ← S ∪ N
```

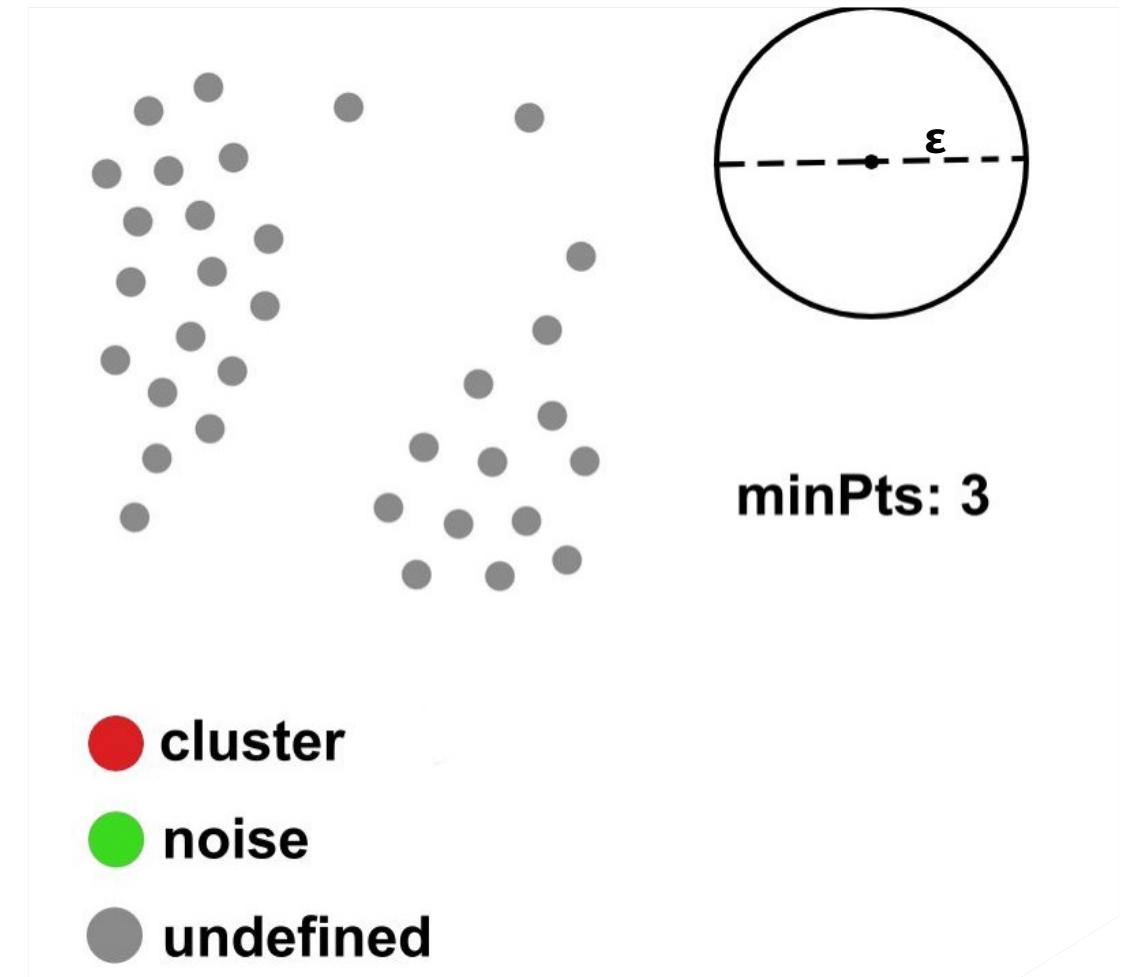


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                      // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label
8   label(p) ← c
9   Seed set S ← N \ {p}                       // Start a new cluster
10  foreach q in S do                         // Expand neighborhood
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue           // Core-point check
16    S ← S ∪ N
```

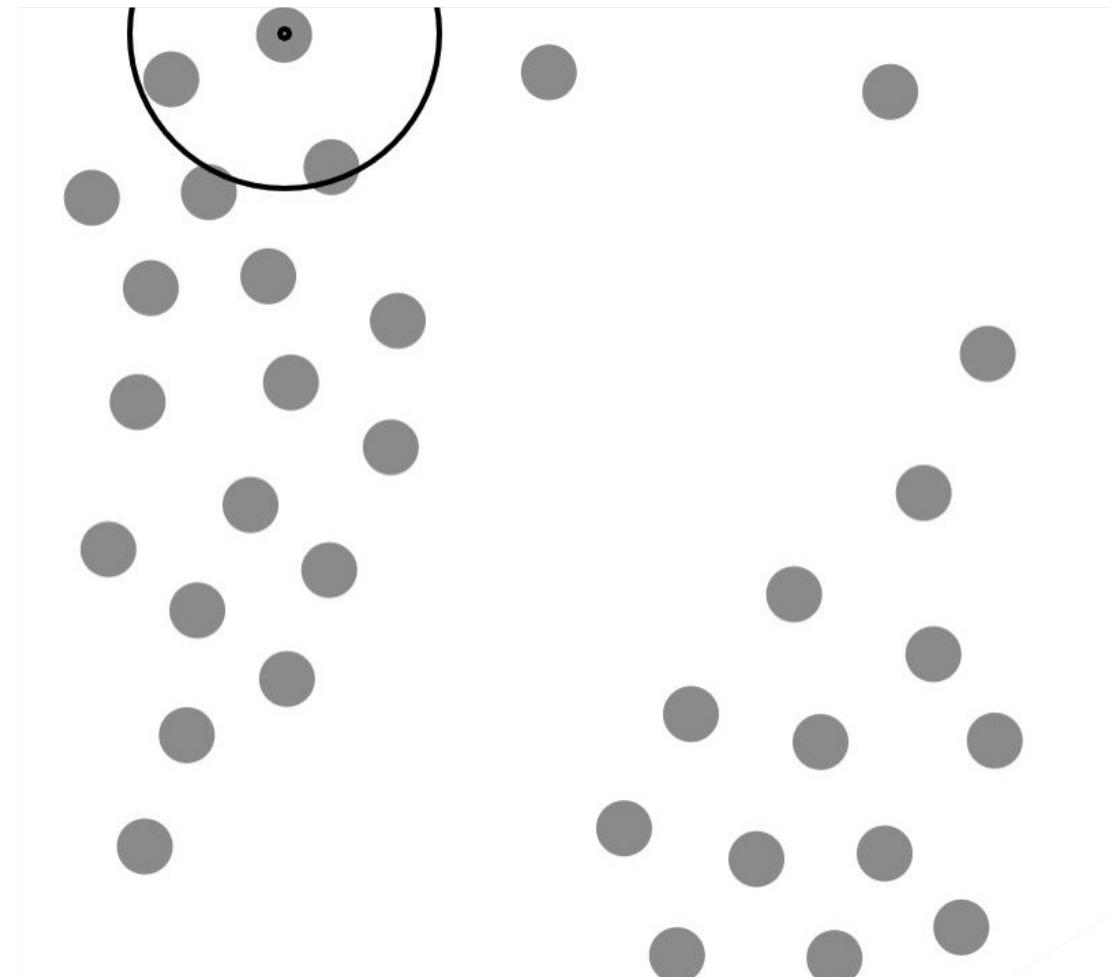


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                      // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                    // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                      // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue           // Core-point check
16    S ← S ∪ N
```

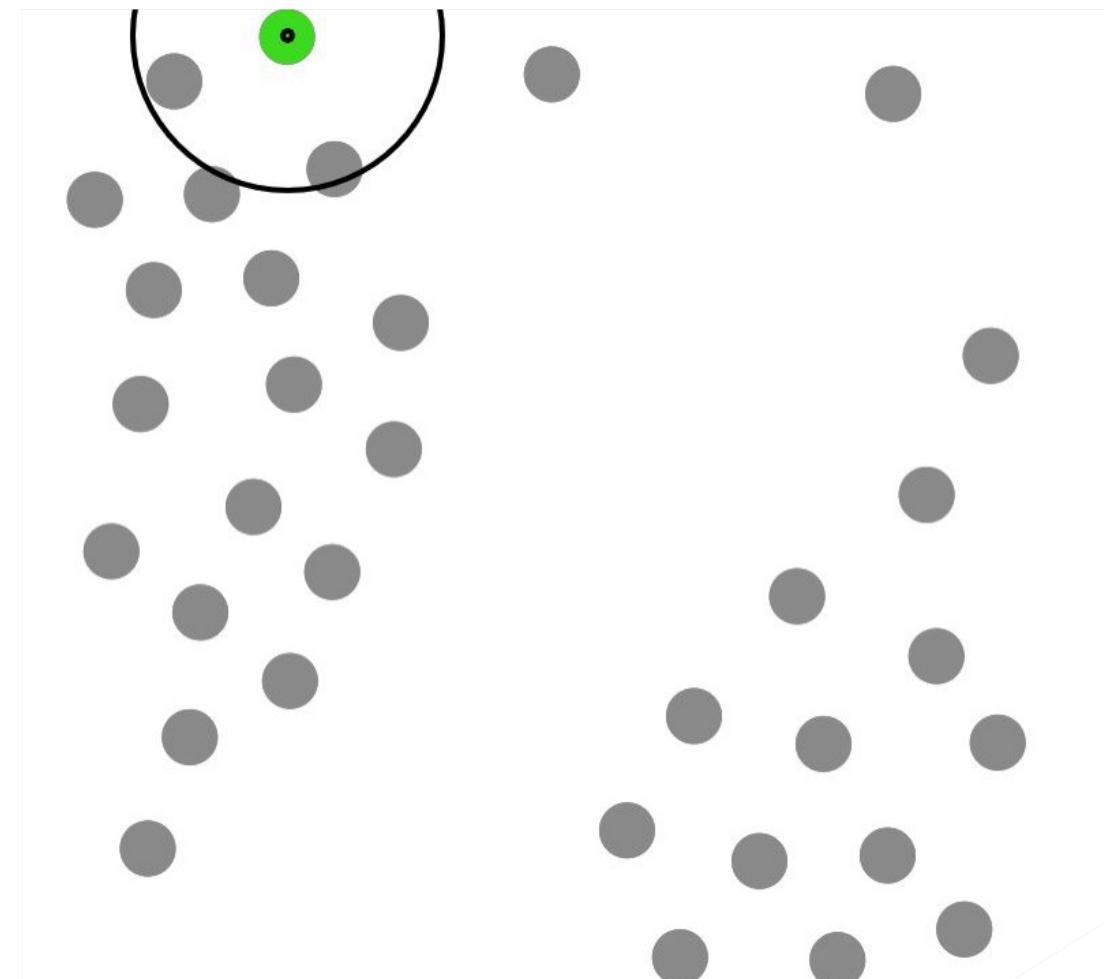


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                      // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                    // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                      // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue           // Core-point check
16    S ← S ∪ N
```

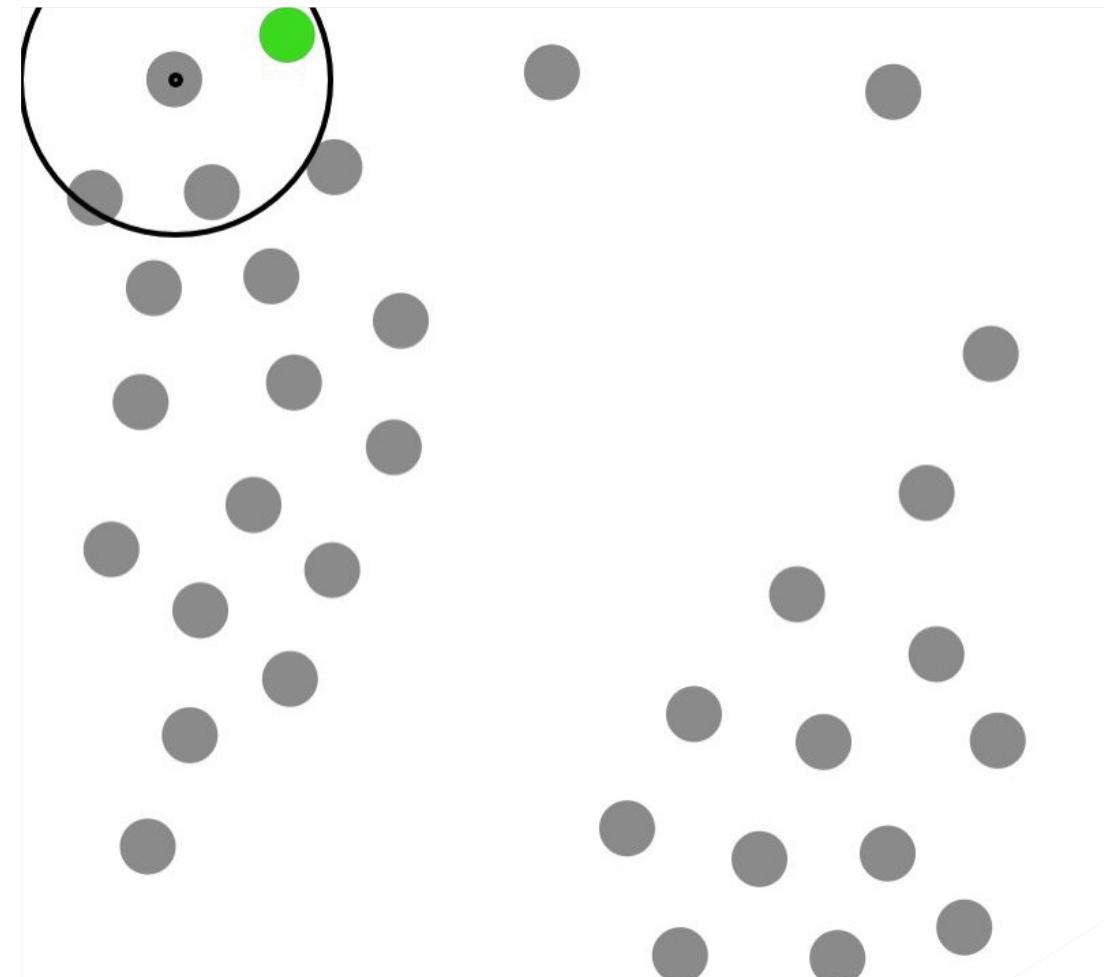


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4)

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                     // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                   // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                      // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue          // Core-point check
16    S ← S ∪ N
```

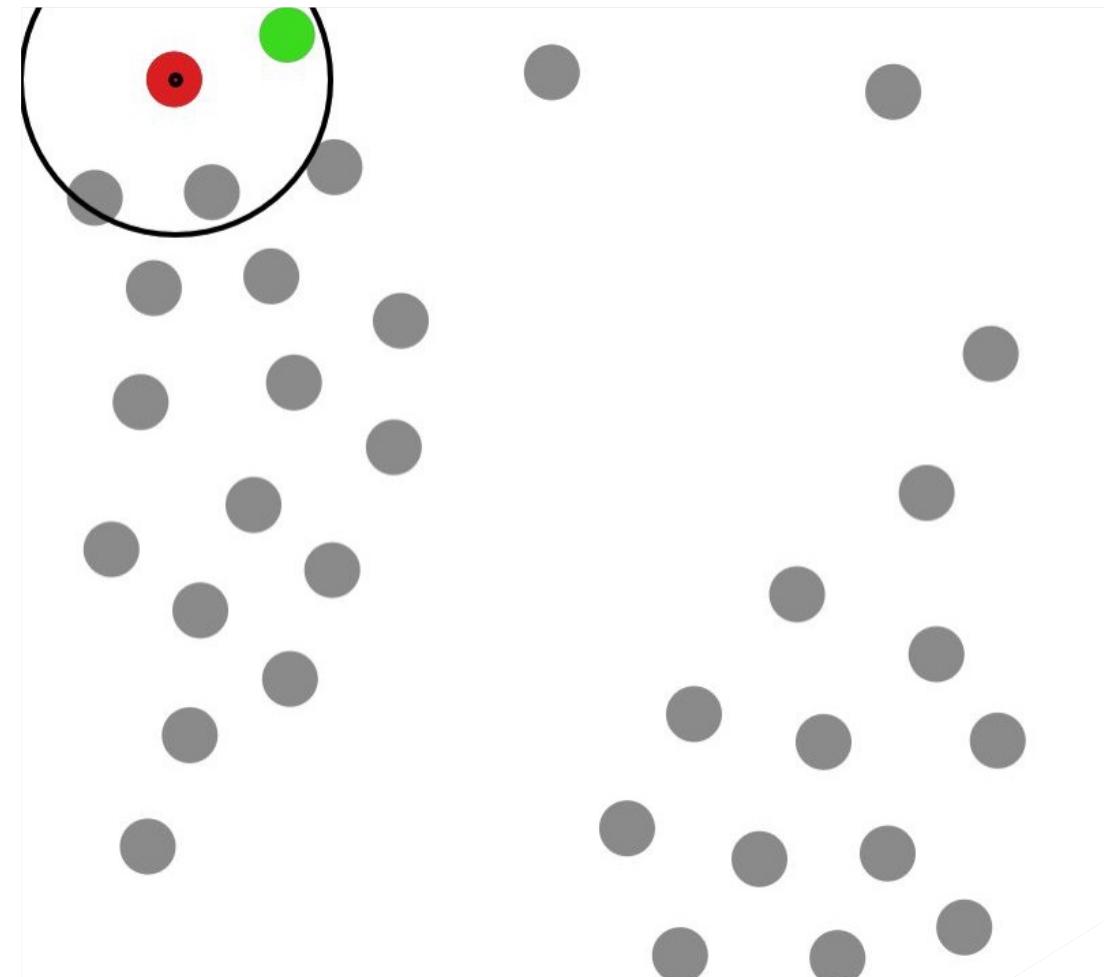


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4)

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                     // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                   // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                      // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue          // Core-point check
16    S ← S ∪ N
```

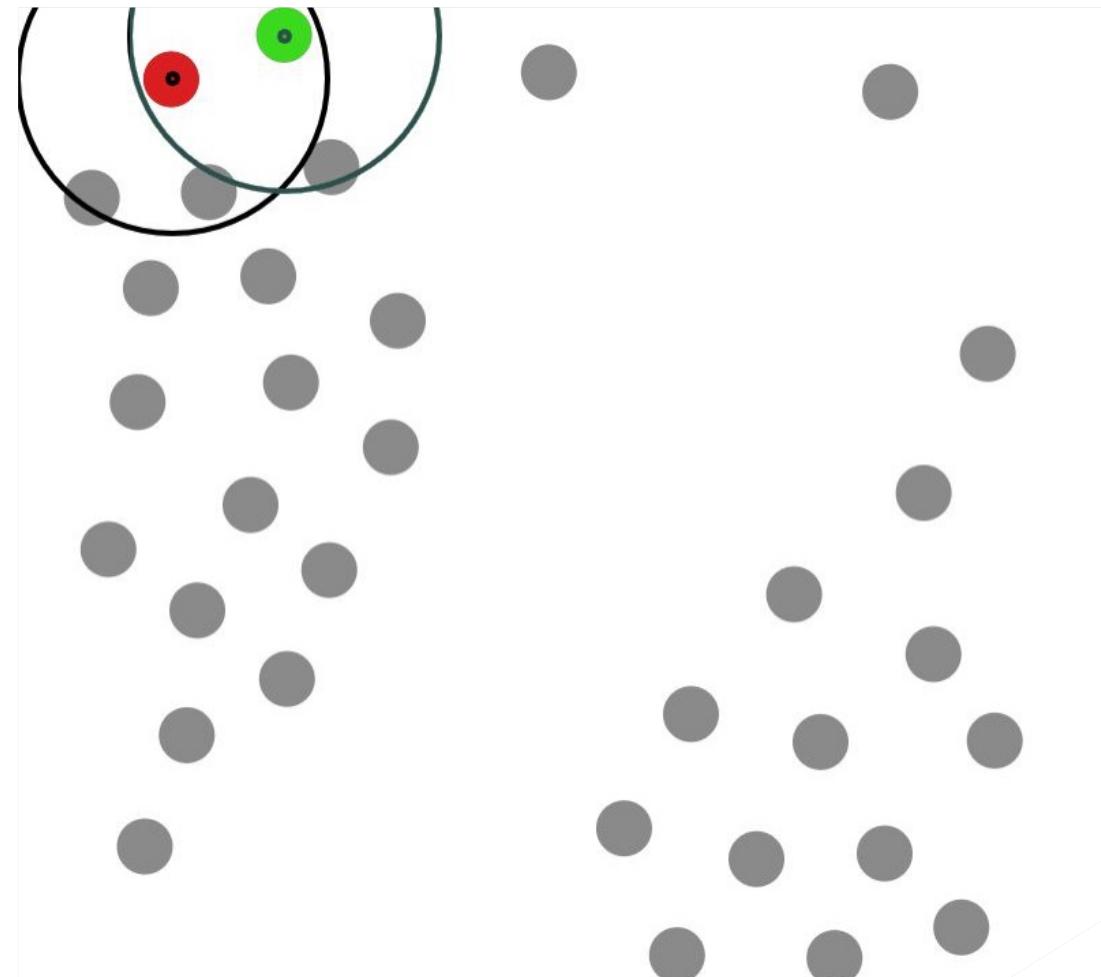


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do                                // Iterate over every point
2   if label(p) ≠ undefined then continue                           // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)                      // Find initial neighbors
4   if |N| < minPts then                                           // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                                         // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                                            // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue                                // Core-point check
16    S ← S ∪ N
```

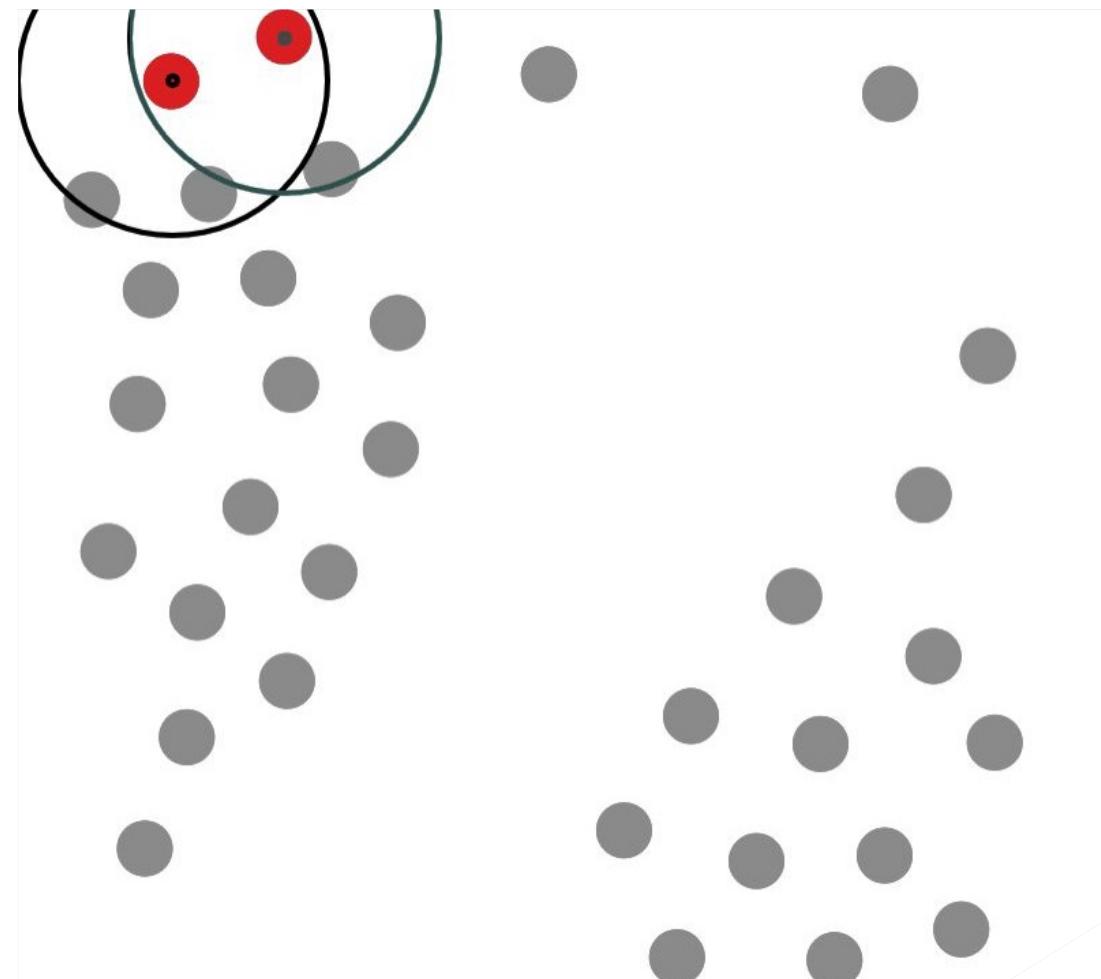


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                      // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                   // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                     // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue          // Core-point check
16    S ← S ∪ N
```

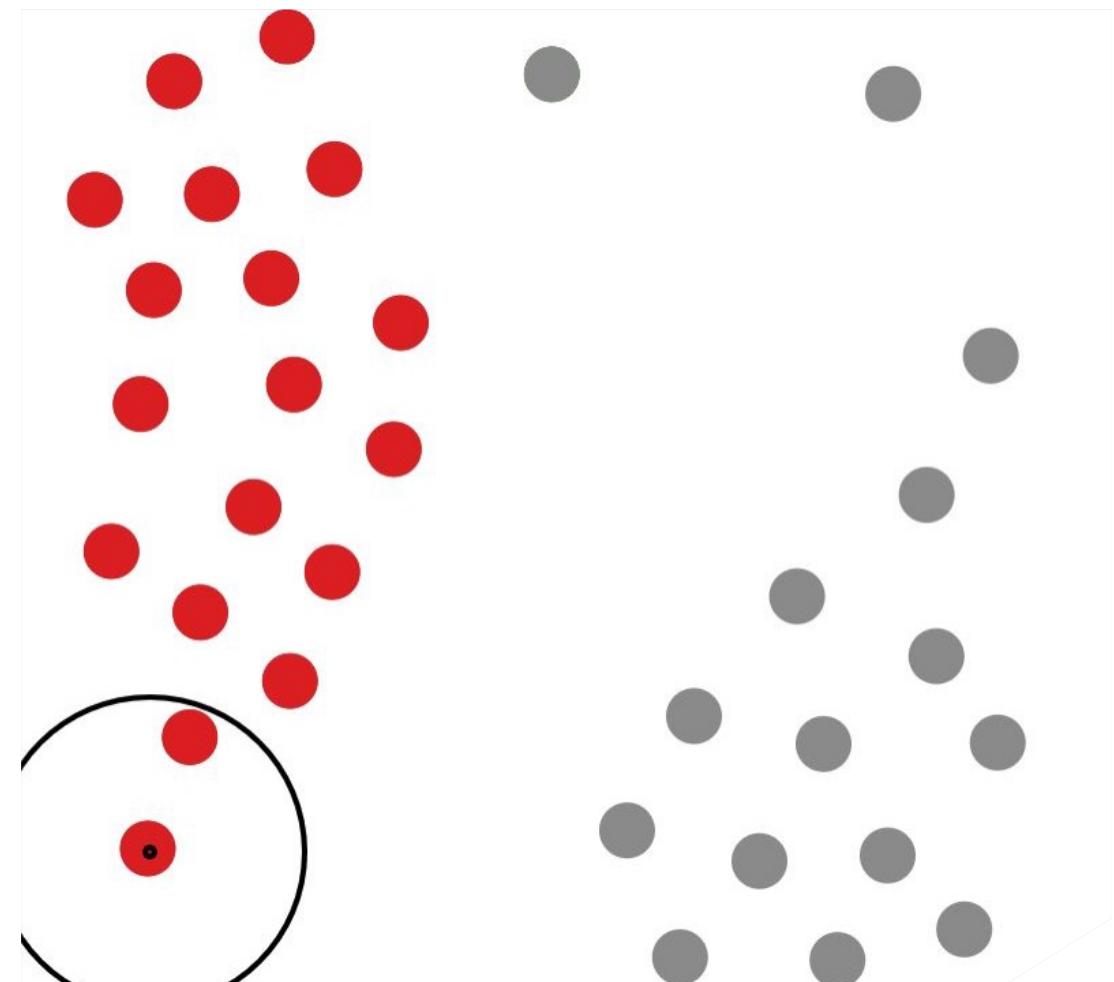


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do           // Iterate over every point
2   if label(p) ≠ undefined then continue      // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)  // Find initial neighbors
4   if |N| < minPts then                      // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                   // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                     // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue          // Core-point check
16    S ← S ∪ N
```

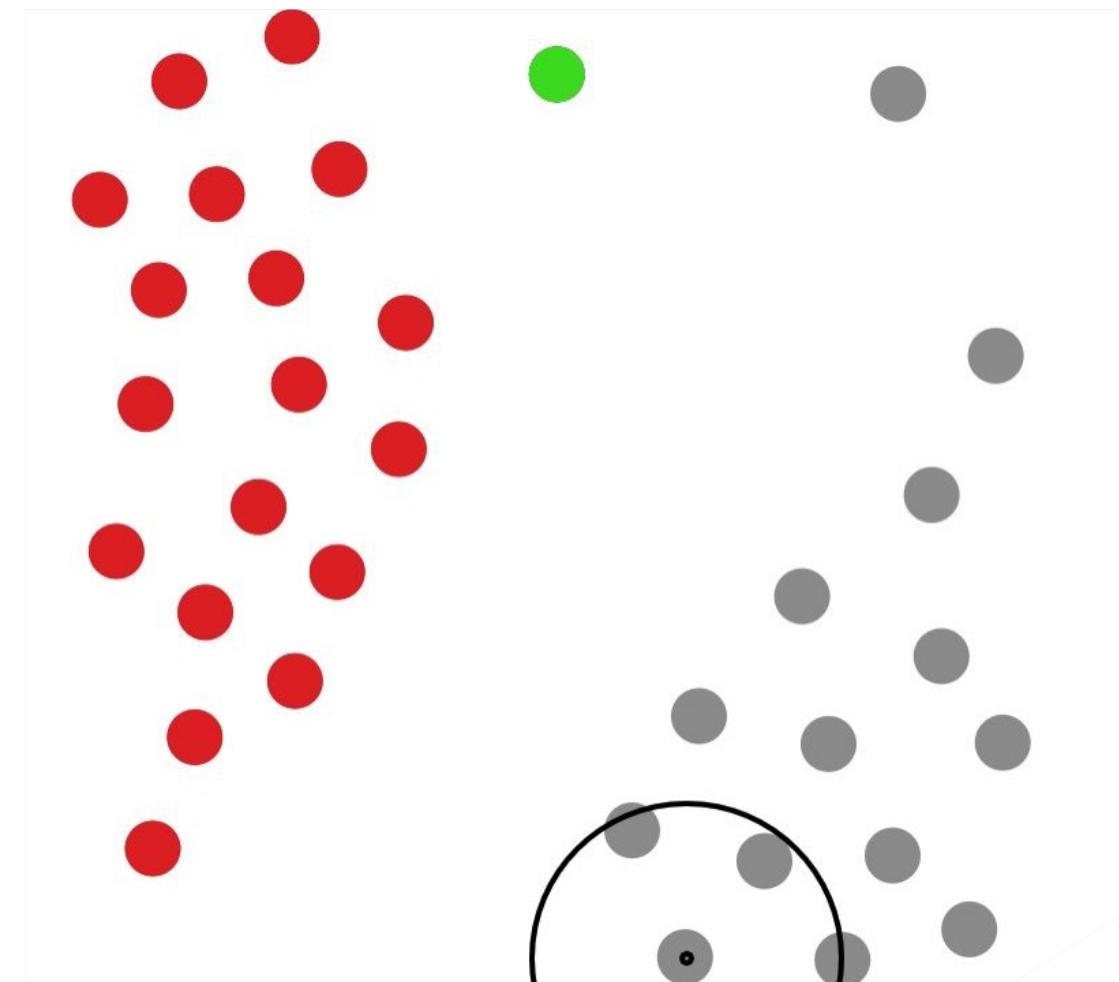


DBSCAN

Algorithmus (E. Schubert et al. (2017), S.19:4

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1 foreach point p in database DB do                                // Iterate over every point
2   if label(p) ≠ undefined then continue                           // Skip processed points
3   Neighbors N ← RANGEQUERY(DB, dist, p, ε)                      // Find initial neighbors
4   if |N| < minPts then                                            // Non-core points are noise
5     label(p) ← Noise
6     continue
7   c ← next cluster label                                         // Start a new cluster
8   label(p) ← c
9   Seed set S ← N \ {p}                                           // Expand neighborhood
10  foreach q in S do
11    if label(q) = Noise then label(q) ← c
12    if label(q) ≠ undefined then continue
13    Neighbors N ← RANGEQUERY(DB, dist, q, ε)
14    label(q) ← c
15    if |N| < minPts then continue                                 // Core-point check
16    S ← S ∪ N
```

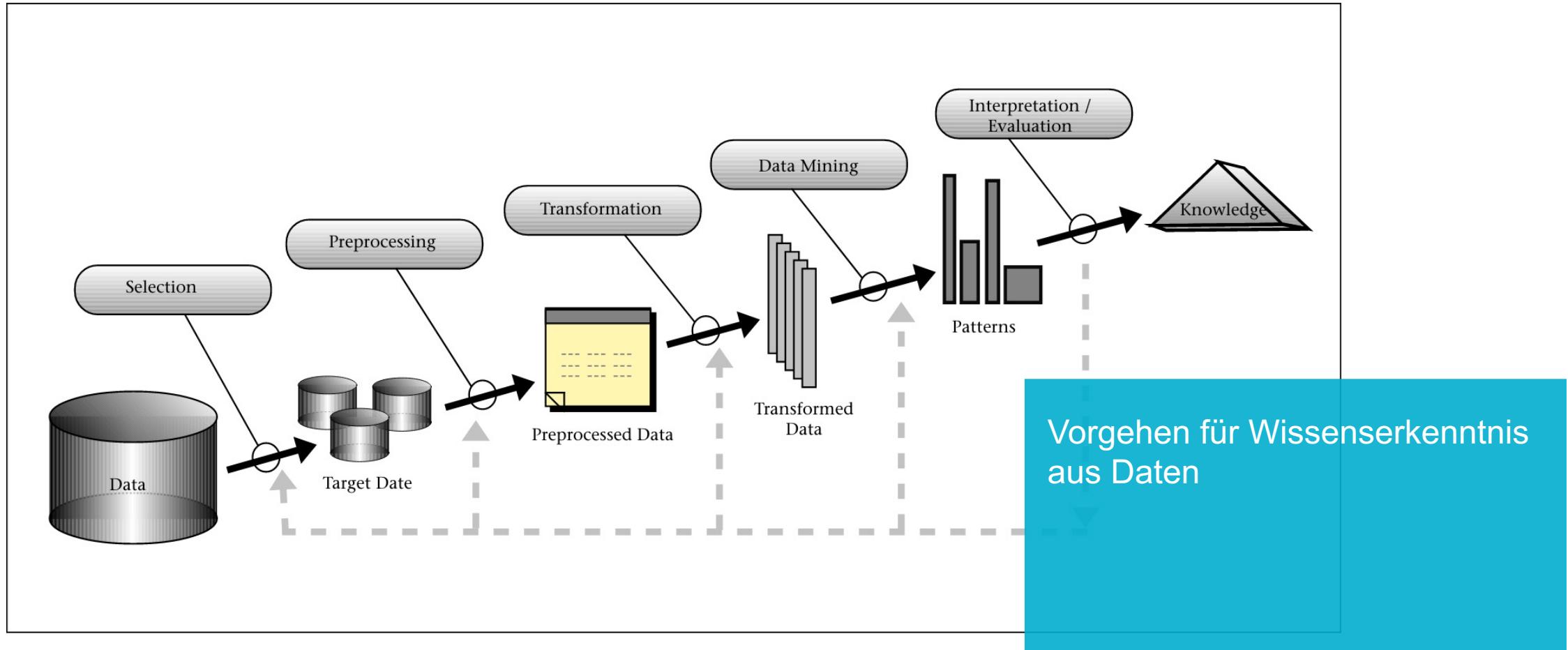


A large, blurred collage of various album covers from the Spotify library, serving as a background for the title.

04

CLUSTERING VON
SONGS AUF
SPOTIFY

Clusteranalyse von Songs auf Spotify



Bildquelle: Fayyad, U.M.; Piatetsky-Shapiro, G.; Smyth, P.: From data mining to knowledge discovery: An overview. In: Fayyad, U.M.; Piatetsky-Shapiro, G.; Smyth, P.; Uthurusamy, R. (Hrsg.): Advances in knowledge discovery and data mining. Menlo Park et al.: AAAI Press, 1996, S. 1-34

Clusteranalyse von Songs auf Spotify

Der Datensatz

- Quelle: https://www.kaggle.com/sashankpillai/spotify-top-200-charts-20202021?select=spotify_dataset.csv (Zugriff: 14.11.2021)
- „The dataset include all the songs that have been on the Top 200 Weekly (Global) charts of Spotify in 2020 & 2021“
- 23 features

Audio features	Consumer features
Danceability	Number of times charted
Acousticness	Artist followers
Energy	Weeks charted
Instrumentalness	...
Loudness	
Genre	
...	
- 1545 Datentupel

Clusteranalyse von Songs auf Spotify

Der Datensatz im „Rohzustand“

Song Name	Streams	Artist	Artist Followers	Song ID	Genre	Release Date	...	Danceability	Energy	Loudness	Speechiness	Acousticness
Beggin'	48,633,449	Måneskin	3377762	3Wrjm47oTz2sjlgck11l5e	['indie rock italiano', 'italian pop']	2017-12-08	...	0.714	0.8	-4.808	0.0504	0.127
STAY (with Justin Bieber)	47,248,719	The Kid LAROI	2230022	5HCyWIXZPP0y6Gqq8TgA20	['australian hip hop']	2021-07-09	...	0.591	0.764	-5.484	0.0483	0.0383
good 4 u	40,162,559	Olivia Rodrigo	6266514	4ZtFanR9U6ndgddUvNjcjG	['pop']	2021-05-21	...	0.563	0.664	-5.044	0.154	0.335
Bad Habits	37,799,456	Ed Sheeran	83293380	6PQ88X9TkUIAUIZJHW2upE	['pop', 'uk pop']	2021-06-25	...	0.808	0.897	-3.712	0.0348	0.0469
INDUSTRY BABY (feat. Jack Harlow)	33,948,454	Lil Nas X	5473565	27NovPIUIRrOZoCHxABJwK	['lgbtq+ hip hop', 'pop rap']	2021-07-23	...	0.736	0.704	-7.409	0.0615	0.0203

Clusteranalyse von Songs auf Spotify

Datenumwandlung

Im „Rohzustand“ kann der vorhandene Datensatz nicht verwertet werden

- Alle Datenzellen sind im Datentyp String gespeichert
- Leere Felder (NA) enthalten
- numerische und kategoriale Merkmale vorhanden
- mehrere Daten in einem Feld

Clusteranalyse von Songs auf Spotify

Der Datensatz nach der ersten Umwandlung

Song Name	Streams	Artist	Artist Followers	Song ID	Genre	Release Date	...	Speechiness	Acousticness
Beggin'	48633449	Måneskin	3377762	3Wrjm47oTz2sjlgck11l5e	'indie rock italiano', 'italian pop'	15126912000000000000	...	0.0504	0.1270
STAY (with Justin Bieber)	47248719	The Kid LAROI	2230022	5HCyWIXZPP0y6Gqq8TgA20	'australian hip hop'	16257888000000000000	...	0.0483	0.0383
good 4 u	40162559	Olivia Rodrigo	6266514	4ZtFanR9U6ndgddUvNcjcG	'pop'	16215552000000000000	...	0.1540	0.3350
Bad Habits	37799456	Ed Sheeran	83293380	6PQ88X9TkUIAUIZJHW2upE	'pop', 'uk pop'	16245792000000000000	...	0.0348	0.0469
INDUSTRY BABY (feat. Jack Harlow)	33948454	Lil Nas X	5473565	27NovPIUIRrOZOCHxABJwK	'lgbtq+ hip hop', 'pop rap'	16269984000000000000	...	0.0615	0.0203

Clusteranalyse von Songs auf Spotify

Umwandlung der Spalte Genre

Ausgangslage: mehrere Genre in einem Datenfeld

Folge: einzelnes Genre eines Songs bleibt unberücksichtigt

Ziel: Jedes Genre eines Songs soll erkannt werden und in die Clusterberechnung einfließen

Vorgehen:

1. Strings umwandeln (Klammern, Leerzeichen entfernen)
2. Spalte Genre in mehrere Spalten aufspalten
3. Genre-Namen in numerische Werte umwandeln
4. Spalten mit gleichen Genres zusammenfassen

Clusteranalyse von Songs auf Spotify

Umwandlung der Spalte Genre

```
import category_encoders as ce

df['Genre'] = df['Genre'].str.strip("[]")
genres_splitted = df['Genre'].str.split(',', expand=True)
genres_splitted = genres_splitted.fillna("NA")
genres_splitted[genres_splitted.columns] = genres_splitted[genres_splitted.columns].replace("", "", regex=True)
genres_splitted[genres_splitted.columns] = genres_splitted[genres_splitted.columns].replace(" ", "", regex=True)
enc = ce.OneHotEncoder(return_df=True, use_cat_names=True, drop_invariant=True)
genres_splitted_expand = enc.fit_transform(genres_splitted)
```

```
display(genres_splitted_expand)
print(np.unique(genres_splitted.values))
```

incepop	0_puertoricanpop	0_latin	0_k-pop	0_canadianpop	0_canadiancontemporary&b#250;	...	8_NA	8_slaphouse	8_yachtrock	8_post-teenpop	8_trap	9_NA	9_trance	9_
0	0	0	0	0		0 ...	1	0	0	0	0	1	0	
0	0	0	0	0		0 ...	1	0	0	0	0	1	0	
0	0	0	0	0		0 ...	1	0	0	0	0	1	0	
0	0	0	0	0		0 ...	1	0	0	0	0	1	0	
0	0	0	0	0		0 ...	1	0	0	0	0	1	0	
...
1	0	0	0	0		0 ...	1	0	0	0	0	1	0	

Clusteranalyse von Songs auf Spotify

Umwandlung der Spalte Genre

```
unq = np.unique(genres_splitted.values)
```

```
genre_df = pd.DataFrame()
for idx, genre in enumerate(unq):
    filter_genre = genres_splitted_expand.filter(regex=f'_{genre}$')
    sum_columns = filter_genre.sum(axis=1)
    sum_columns.name = genre
    genre_df.insert(loc=idx, column=genre, value=sum_columns)
genre_df = genre_df.drop(columns=['NA'])
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/IPython/core/interactiveshell.py:336
4: PerformanceWarning: DataFrame is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
    if (await self.run_code(code, result, async_=asy)):
```

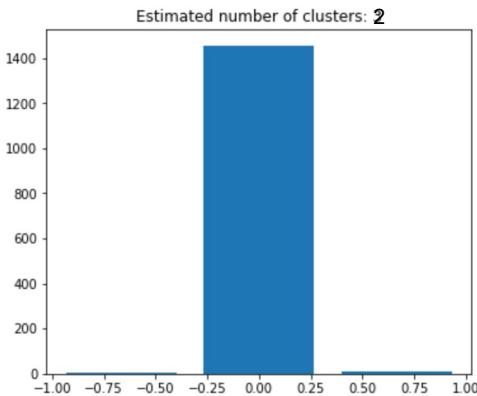
```
display(genre_df.head())
```

albumrock	alternativemetal	alternativepoprock	alternativer&b	...	vancouverindie	vaportrap	vegasindie	venezuelanhiphop	viralpop	viralrap	vocaljazz	weird
0	0	0	0	...	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	0	0	0	0	0	0

Clusteranalyse von Songs auf Spotify

erster Versuch fürs Clustering

nachdem die Datentypen im Datensatz umgewandelt und die Spalte Genre aufgeteilt wurde, starten wir das erste Mal ein Clustering. Dafür wurden die Daten normiert (dazu später mehr)



```
from sklearn.cluster import DBSCAN  
  
db = DBSCAN(eps=0.02113777, metric = "precomputed", n_jobs=-1).fit(dist_matrix)
```

Beobachtung:

- zwei Cluster, ein Ausreißer
- ein Cluster enthält 98.98% aller Daten
- Performance (Silhouette-Score): 0.36
- Folgerung: Bearbeite den Datensatz
 - Wir haben als nächstes die Genres angeschaut

Analyse der Labels und Cluster-Qualität

```
print(f'number of values: {len(_labels)}')  
print(f'all labels: {np.unique(_labels)}')  
display(np.histogram(_labels, bins=number_clusters))  
  
number of values: 1470  
all labels: [-1  0  1]  
  
(array([ 7, 1455,     8]),  
 array([-1.          , -0.33333333,  0.33333333,  1.          ]))
```

Clusteranalyse von Songs auf Spotify

erster Versuch fürs Clustering

Fuzzy

```
from fcmmeans import FCM
from sklearn.metrics import silhouette_score

def Silhouette(x, kmax):
    sil = [None, None]
    for k in range(2, kmax+1):
        fcm = FCM(n_clusters = k)
        fcm.fit(x)
        labels = fcm.u.argmax(axis=1)
        sil.append(silhouette_score(x, labels, metric = 'euclidean'))
    return sil

Silhouette(dist_matrix, 10)
```

[None,
None,
0.15693144530319653,
0.15693144530319653,
0.15693144530319653,
0.15693144530319653,
0.15693144530319653,
0.15693144530319653,
-0.040791244371648,
0.027396885562283566,
0.012045200949085922,
0.15693144530319653]

Optimale Clusteranzahl
zwischen 2 und 6, 10

```
print("Anzahl der Beobachtungen für Cluster 0: ", len(df_final[df_final["Cluster"] == 0]))
print("Anzahl der Beobachtungen für Cluster 1: ", len(df_final[df_final["Cluster"] == 1]))
print("Anzahl der Wahrscheinlichkeiten für Cluster 1 & 2, die zwischen 0.49 und 0.51: ",
    ((fcm.u[np.where((fcm.u > 0.49) & (fcm.u < 0.51))].size)/2)
```

Anzahl der Beobachtungen für Cluster 0: 713
Anzahl der Beobachtungen für Cluster 1: 757
Anzahl der Wahrscheinlichkeiten für Cluster 1 & 2, die zwischen 0.49 und 0.51: 1470.0

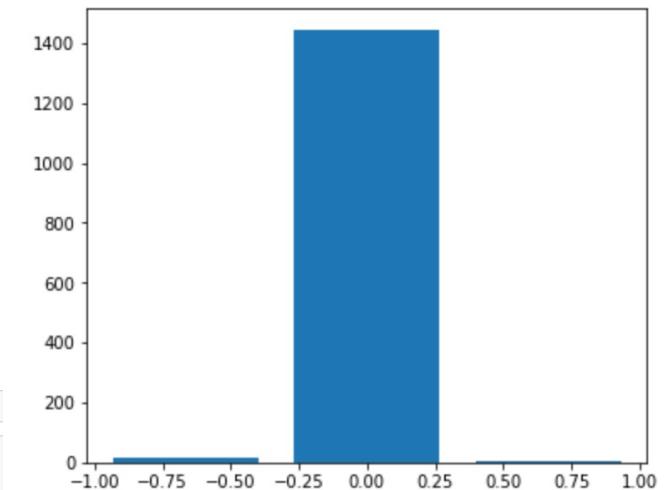
Die Implementierung des Fuzzy-Algorithmus lässt keine kategorialen Daten zu. Daher wurde für nur hier für den Vergleich auch beim DBSCAN kategoriale Merkmale entfernt. In den weiteren Versuchen verbleiben kategoriale Merkmale beim DBSCAN

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.61307393, metric = "precomputed")
```

Cluster Plotting

```
_labels = np.array(db.labels_)
plt.figure(figsize=(6, 5))
number_clusters = len(np.unique(_labels))
plt.hist(_labels, bins=number_clusters, rwidth=0.8)
plt.title(f"Estimated number of clusters: { number_clusters }")
plt.show()
```

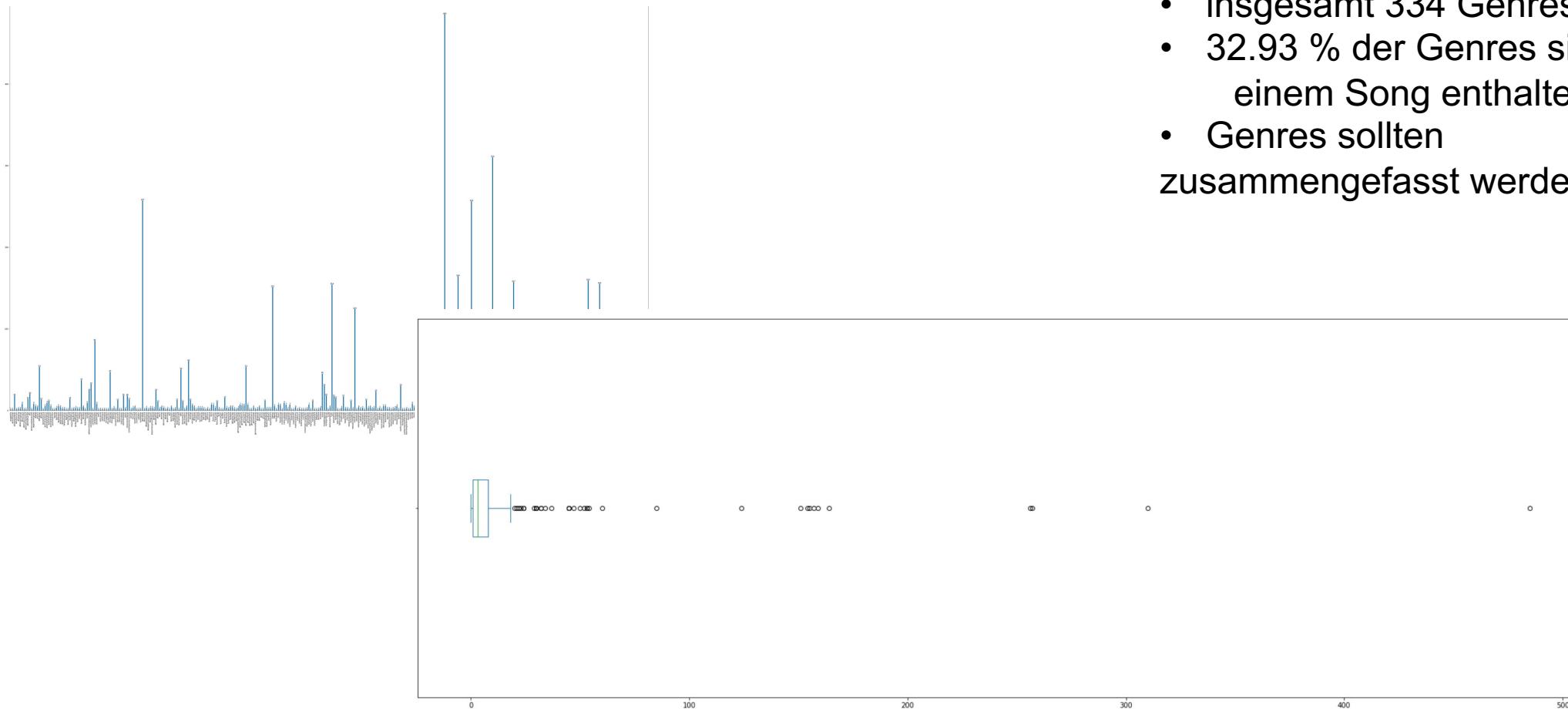
Estimated number of clusters: 2



```
from sklearn.metrics import silhouette_score
print(f'number of values: {len(_labels)}')
print(f'all labels: {np.unique(_labels)}')
display(np.histogram(_labels, bins=number_clusters))
print(f'silhouette score: {silhouette_score(X=dist_matrix, labels=db.labels_, metric="euclidean")}')
number of values: 1470
all labels: [-1  0  1]
(array([ 18, 1445,     7]), array([-1.          , -0.33333333,  0.33333333,  1.          ]))
silhouette score: 0.3284784251521221
```

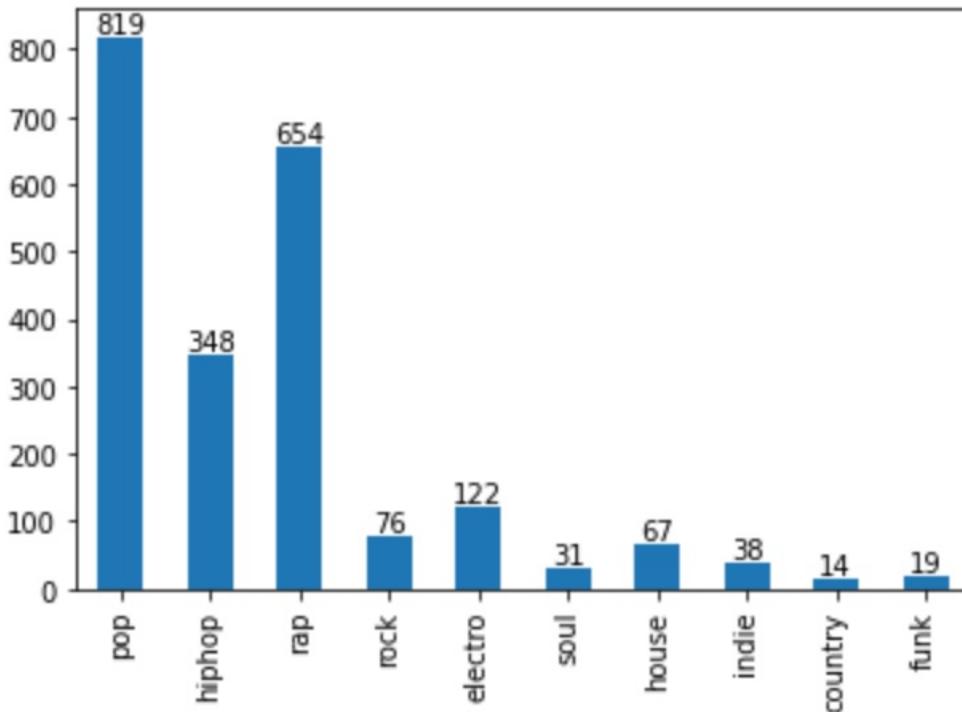
Clusteranalyse von Songs auf Spotify

Untersuchung der Genres



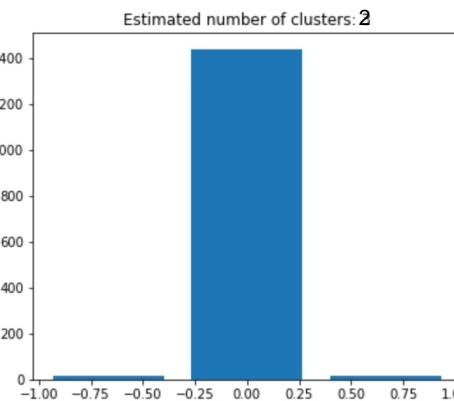
Clusteranalyse von Songs auf Spotify

Gruppierung der Genres



Mit den gruppierten Genres
wird erneut eine Clusteranalyse durchgeführt

```
db = DBSCAN(eps=0.11623452, metric = "precomputed", n_jobs=-1).fit(dist_matrix)
```



Analyse der Labels und Cluster-Qualität

```
print(f'number of values: {len(_labels)}')
print(f'all labels: {np.unique(_labels)}')
display(np.histogram(_labels, bins=number_clusters))
```

```
number of values: 1470
all labels: [-1  0  1]
(array([ 16, 1437,   17]), array([-1.          , -0.33333333,  0.33333333,  1.          ]))
```

- 97,76 % der Songs sind in einem Cluster → 1.22% weniger als vorher
- Performance (Silhouette-Score): 0.23 → Verschlechterung um 0.13

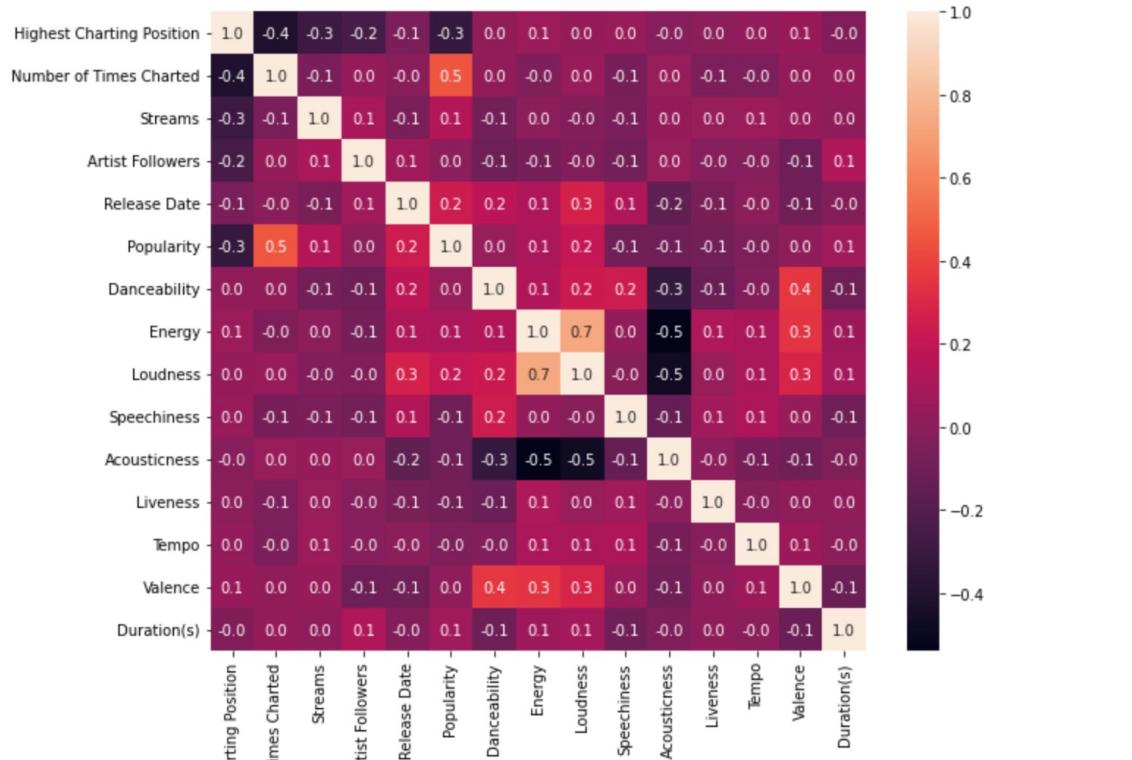
Clusteranalyse von Songs auf Spotify

Analyse der Korrelation zwischen den Variablen

```
import seaborn as sns
%matplotlib inline

#Korrelation's plot
df_o = df_final.copy()

df_o.drop(["pop", "hiphop", "rap", "rock", "electro", "soul", "house", "indie", "country", "funk"], inplace=True, axis=0)
sns.heatmap(df_o.corr(), annot = True, fmt = ".1f", ax = ax)
plt.show()
```



Bisher sind alle Songs in einer Gruppe. Ändert sich das, wenn wir bestimmte Variablen auslassen? Wir untersuchen die Variablen auf Multikollinearität

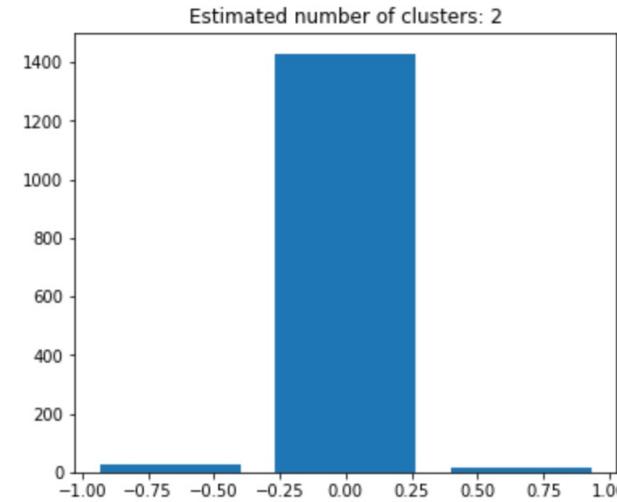
Die stärkste Korrelation liegt bei 0.7 (< 0.95 → keine Multikollinearität). Bei Entfernung des Genres „Energy“ ändert sich die Cluster-Performance nicht.

Clusteranalyse von Songs auf Spotify

Entfernen von Bewegungsdaten: DBSCAN

Variablen mit einzigartigen Ausprägungen ohne „Informationsgehalt“ (SongID, Song-Name) wurden von Anfang an entfernt.

Man könnte zusätzlich Daten entfernen, die sich mit der Zeit ändern (Streams, Highest Charting Position, Number of Times Charted, Artist-Followers, Popularity).



Die Clusterstruktur und Clusterqualität hat sich kaum geändert

Analyse der Labels und Cluster-Qualität

```
print(f'number of values: {len(_labels)}')
print(f'all labels: {np.unique(_labels)}')
display(np.histogram(_labels, bins=number_clusters))

number of values: 1470
all labels: [-1  0  1]
(array([ 28, 1429,   13]),
 array([-1.          , -0.33333333,  0.33333333,  1.          ]))
```

Clusteranalyse von Songs auf Spotify

Entfernen von Bewegungsdaten: Fuzzy

Variablen mit einzigartigen Ausprägungen ohne „Informationsgehalt“ (SongID, Song-Name) wurden von Anfang an entfernt.

Man könnte zusätzlich Daten entfernen, die sich mit der Zeit ändern (Streams, Highest Charting Position, Number of Times Charted, Artist-Followers, Popularity).

```
from fcmeans import FCM
from sklearn.metrics import silhouette_score

def Silhouette(x, kmax):
    sil = [None, None]
    for k in range(2, kmax+1):
        fcm = FCM(n_clusters = k)
        fcm.fit(x)
        labels = fcm.u.argmax(axis=1)
        sil.append(silhouette_score(x, labels, metric = 'euclidean'))
    return sil

Silhouette(dist_matrix, 10)

[None,
 None,
 0.20732809121769868, ← Optimale Clusteranzahl bei k = 2
 0.12281328718295265,
 0.11474230764706296,
 0.12137638494910714,
 0.10886487609852523,
 0.010869988662217736,
 0.030784418432915458,
 0.04398171970151576,
 -0.004119419241819058]
```

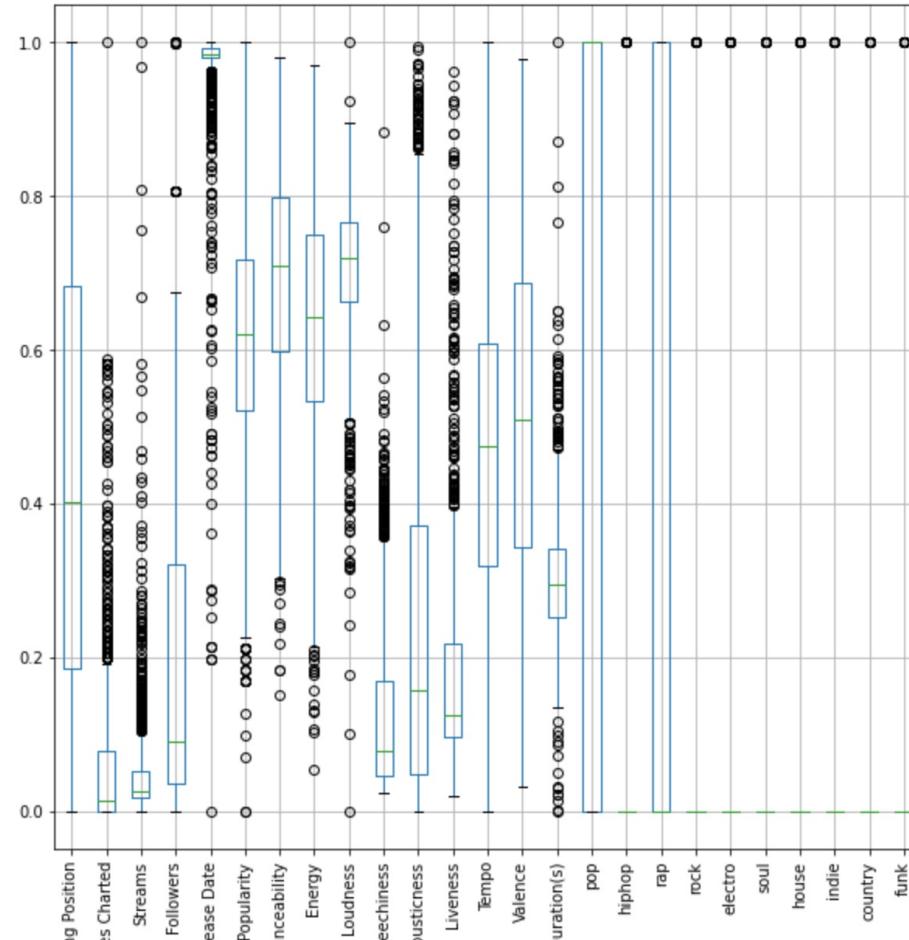
Cluster	Cluster Probability
0	0.670281
0	0.644017
0	0.666522
0	0.629763
0	0.614729
...	...
1	0.725727
1	0.636028
1	0.715500
1	0.645205
1	0.580747

Clusteranalyse von Songs auf Spotify

Genauere Untersuchung des Datensatzes

Durch eine nähere Betrachtung der einzelnen Variablen könnten weitere Möglichkeiten geben, wie man den Datensatz umwandeln kann

```
choose_columns = df_final.drop(['Chord'], axis=1).columns.tolist()
boxplot = df_final.boxplot(column=choose_columns, figsize=(10, 10), vert=True, rot=90)
```



- 10 von 15 metrisch skalierten Variablen zeigen eine starke Schiefe auf.
- 7 (metr.) Variablen rechtsschief, 5 linksschief
- viele und starke Ausreißer in mehreren Variablen

Weiterer Ansatz: Daten anders skalieren

Clusteranalyse von Songs auf Spotify

Datenskalierung

Bisher wurden auf die Daten eine Min-Max-Skalierung angewandt.

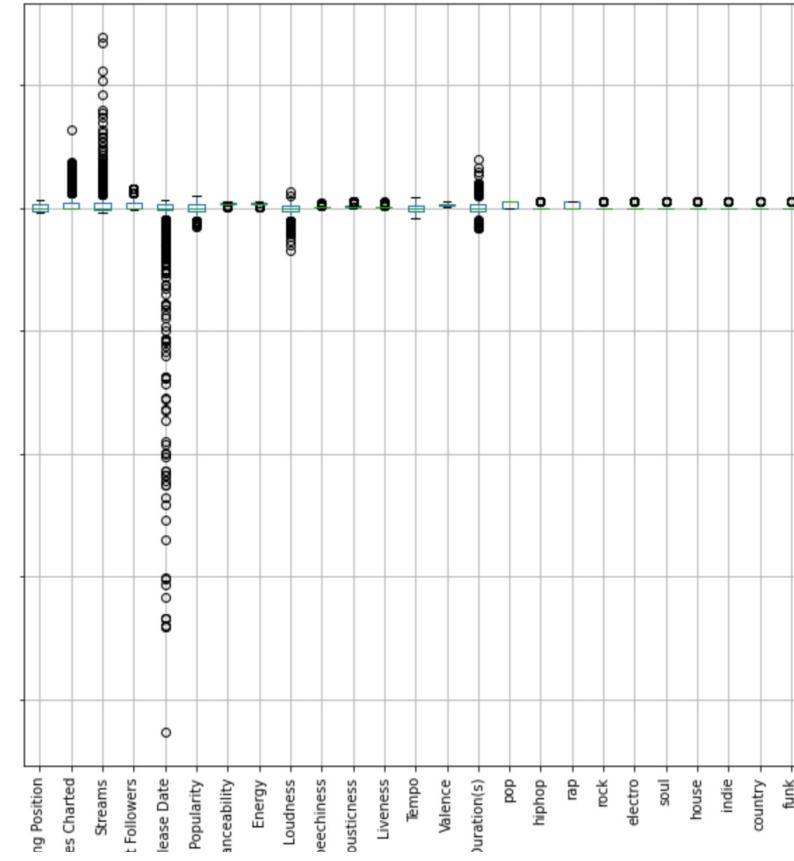
$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Nun wenden wir den RobustScaler an, der stabiler bei Ausreißern ist.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler

# scaler = MinMaxScaler()
scaler = RobustScaler()

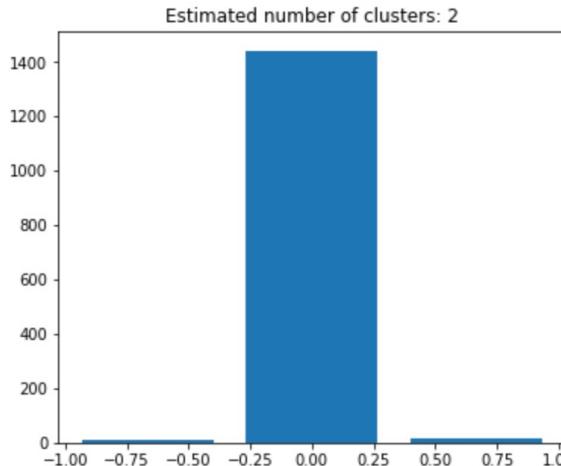
scaler.fit(df_final[['Tempo', 'Loudness','Duration(s)', 'Release Date', *movement_data]])
df_final[['Tempo', 'Loudness','Duration(s)', 'Release Date', *movement_data]] = scaler.transform(df_final[['Tempo', 'Loudness','Duration(s)', 'Release Date', *movement_data]])
```



Für den nächsten Clustering-Versuch entfernen wir zusätzlich drei Variablen mit den größten Ausreißern (Streams, Release Date, Times Charted)

Clusteranalyse von Songs auf Spotify

Datenskalierung



Analyse der Labels und Cluster-Qualität

```
from sklearn.metrics import silhouette_score

print(f'number of values: {len(_labels)}')
print(f'all labels: {np.unique(_labels)}')
display(np.histogram(_labels, bins=number_clusters))
print(f'silhouette score: {silhouette_score(X=dist_matrix, lat
```

number of values: 1470
all labels: [-1 0 1]
(array([11, 1441, 18]),
 array([-1. , -0.33333333, 0.33333333, 1.]))
silhouette score: 0.23126494884490967

Auch nach einer anderen Skalierung ändert sich das Clustering nicht

Clusterqualität

```
from fcmmeans import FCM
from sklearn.metrics import silhouette_score

def Silhouette(x, kmax):
    sil = [None, None]
    for k in range(2, kmax+1):
        fcm = FCM(n_clusters = k)
        fcm.fit(x)
        labels = fcm.u.argmax(axis=1)
        sil.append(silhouette_score(x, labels, metric = 'euclidean'))
    return sil

Silhouette(dist_matrix, 10)
[None,
 None,
 0.17144927676054478, ← Optimale Clusteranzahl bei k = 2
 -0.00829538468555254,
 -0.044211043443541764,
 -0.06342021460163105,
 -0.10933897123212856,
 -0.17127103963169193,
 -0.12123774113090868,
 -0.11619497485652458,
 -0.13920873952315688]
```

```
print("Anzahl der Beobachtungen für Cluster 0: ", len(df_final[df_final["Cluster"] == 0]))
print("Anzahl der Beobachtungen für Cluster 1: ", len(df_final[df_final["Cluster"] == 1]))
print("Anzahl der Wahrscheinlichkeiten für Cluster 1 & 2, die zwischen 0.49 und 0.51: ",
      ((fcm.u[np.where((fcm.u > 0.49) & (fcm.u < 0.51))].size)/2)

Anzahl der Beobachtungen für Cluster 0:  808
Anzahl der Beobachtungen für Cluster 1:  662
Anzahl der Wahrscheinlichkeiten für Cluster 1 & 2, die zwischen 0.49 und 0.51:  1470.0
```

Clusteranalyse von Songs auf Spotify

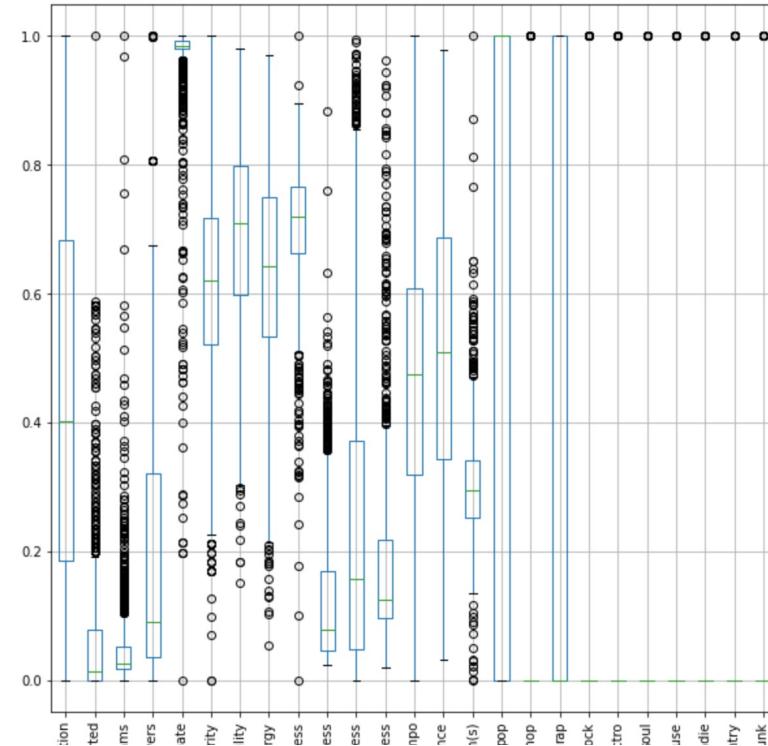
Normierung

Als letztes wird versucht die Schiefe aus den Daten zu korrigieren. Dazu werden die Daten vor der Skalierung logarithmiert. Anschließend wird der Min-Max-Scaler angewandt.

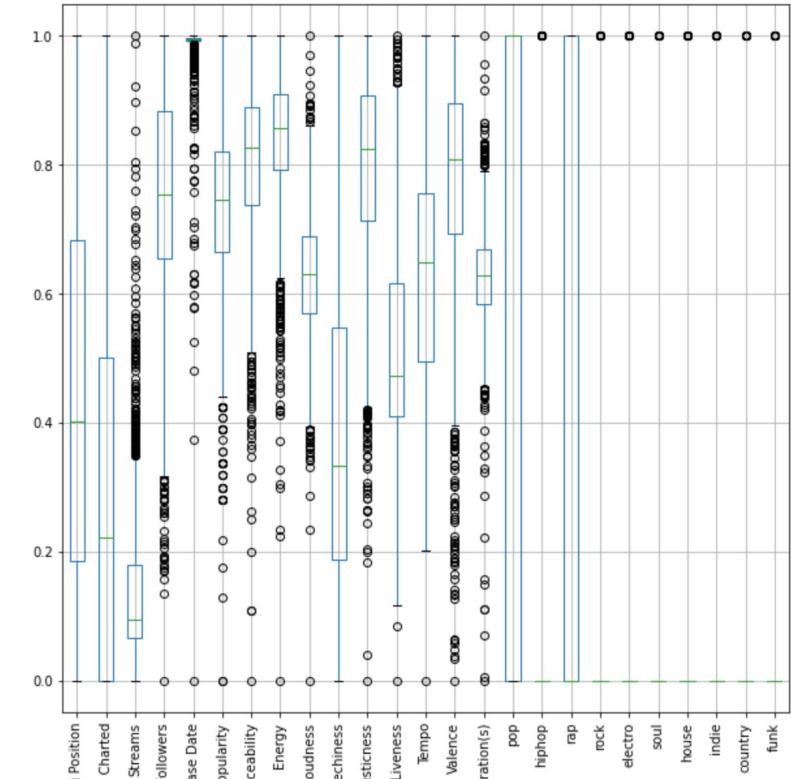
Speechiness	Acousticness	Liveness	Tempo	Duration (ms)	Valence	Chord	Duration(s)
-1.297569	-0.896196	-0.444906	2.127111	211560	-0.229885	B	2.325434
-1.316053	-1.416801	-0.987163	2.230265	141806	-0.320572	C#/Db	2.151695

Speechiness	Acousticness	Liveness	Tempo	Duration (ms)	Valence	Chord	Duration(s)
0.0504	0.1270	0.3590	134.002	211560	0.589	B	211.560
0.0483	0.0383	0.1030	169.928	141806	0.478	C#/Db	141.806

ohne Logarithmieren



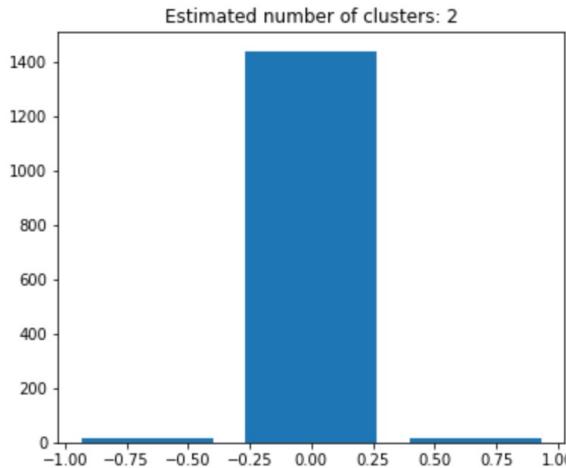
mit Logarithmieren



auch nach dem Logarithmieren bleibt eine Schiefe bei den meisten Variablen vorhanden

Clusteranalyse von Songs auf Spotify

Clustering nach Logarithmierung der Daten



auch nach dem
Logarithmieren
gibt es keine
wesentliche
Änderung
beim Clustering.

Analyse der Labels und Cluster-Qualität

```
from sklearn.metrics import silhouette_score

print(f'number of values: {len(_labels)}')
print(f'all labels: {np.unique(_labels)}')
display(np.histogram(_labels, bins=number_clusters))
print(f'silhouette score: {silhouette_score(X=dist_ma1
```

number of values: 1470
all labels: [-1 0 1]
(array([16, 1437, 17]),
 array([-1. , -0.33333333, 0.33333333, 1.
silhouette score: 0.22990764677524567



HAUPT- KOMPONENTEN- ANALYSE

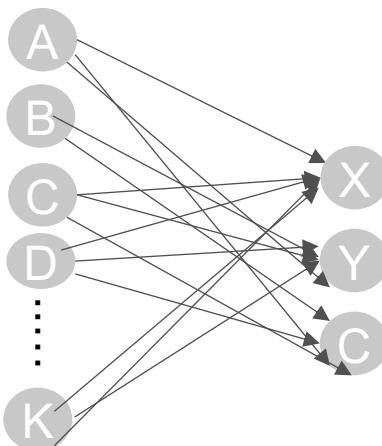
Clustering von Songs auf Spotify

Hauptkomponentenanalyse (vgl. Analytik Vidhya, 2020)

- Principal Component Analysis (PCA)
- Verfahren der multivariaten Statistik für der Datenreduktion

Ziel: Bündeln von Informationen aus vielen Variablen in wenige Hauptkomponente

- Hauptkomponenten: Künstlich erzeugte Variablen, die alle voneinander stochastisch unabhängig sind



Clustering von Songs auf Spotify

PCA: Vorgehensweise (vgl. Analytik Vidhya, 2020)

1. Standardisierung der Daten (*data_stand*: 5x4)
2. Berechnung der Kovarianz mit dem gegebenen Datensatz, um die Korrelation zu erkennen
(*data_cov*: quadratische 4x4 Matrix)
3. Berechnen der Eigenwerte und Eigenvektoren der Kovarianzmatrix, um die Hauptkomponenten zu identifizieren
 1. Es gilt: $A\vec{v} - \lambda\vec{v} = 0$
 2. Bestimmung des Eigenwerts λ : $\det(A - \lambda I) = 0$
 3. Bestimmung des Eigenvektors: $(A - \lambda I)\vec{v} = 0$

Clustering von Songs auf Spotify

PCA: Vorgehensweise (vgl. Analytik Vidhya, 2020)

4. Sortieren der Eigenwerte und die entsprechenden Eigenvektoren
5. K Eigenwerte wählen und Matrix von Eigenvektoren
6. Transformation der ursprünglichen Matrix

Data_stand * k Eigenvektoren = Transformierten Daten

Beispiel für Schritt 5 und 6:

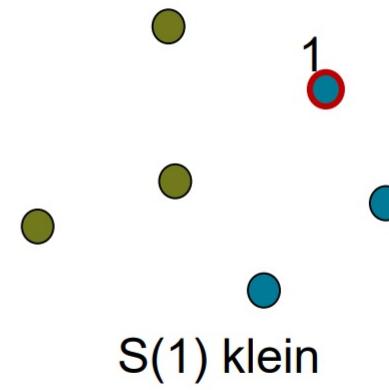
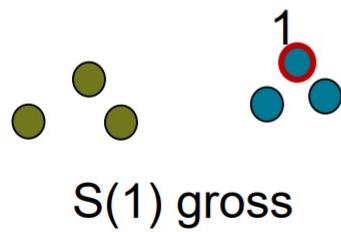
- Schritt 5: K = 2, entspricht 2 Eigenvektor
- Schritt 6: data_stand * matrix (Eigenvektor_1 Eigenvektor_2) = Transformierte Daten
$$(5, 4) \quad * \quad (4, 2) \quad = \quad (5, 2)$$



SILHOUETTE

Silhouette

Beispiel



Silhouette

Silhouettenkoeffizient (vgl. Alind Gupta, 2019)

- Definiert als der arithmetische Mittel aller n_c Silhouetten des Clusters C
- Es gilt $s_c = \frac{1}{n_c} \sum_{i \in C} s(i)$
- Verwendung im Spotify Datensatz für die Wahl des optimalen k Werts beim Fuzzy Algorithmus

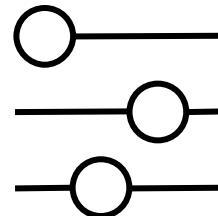
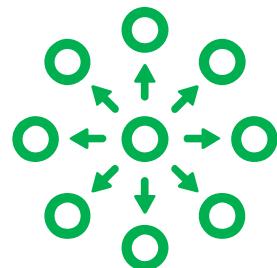


05

FAZIT

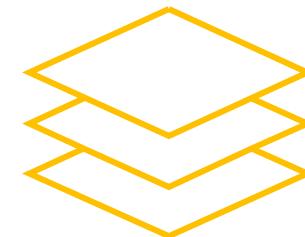
Clustering von Songs auf Spotify

Wir wollten die meistgehörten Songs auf Spotify in mehrere sinnvolle Gruppen einteilen und verstehen, worin sich diese Songs unterscheiden. Dabei sollte die Qualität dieser Gruppen möglichst groß sein.



Obwohl wir auf verschiedene Weisen den Datensatz transformiert haben, hat sich die Clusterstruktur kaum geändert. Welche Erkenntnis ziehen wir daraus?

Die beliebtesten Songs sind sich so ähnlich, dass sie sich nicht in Gruppen aufteilen lassen. Daraus lässt sich folgern, dass sich der Musikgeschmack bei den Songs nicht unterscheidet.



WAS HABEN WIR AUS DEM PROJEKT GELERNT ?

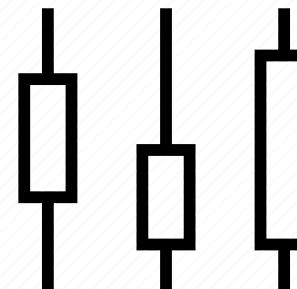


1

Datenvorbereitung und Transformation benötigen die meiste Zeit (ca. 80 %) in Gegensatz zum Data-Mining-Prozess

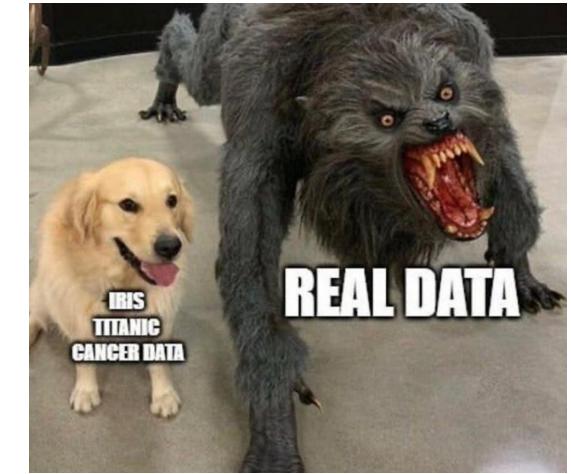
2

Clusteranalyse geht nicht ohne statistische Analyse



3

Die Realität kann zu chaotisch sein, um in ihr Ordnung zu finden oder sie ist zu ähnlich, dass dass man keine Unterschiede findet





06

QUELLEN

Unsupervised learning - Clusteranalyse

Literaturverzeichnis

Kapitel 1:

- Roland Schwaiger, Joachim Steinwendner, Neuronale Netze programmieren mit Python, 1.Auflage, 2019 S. 36-39
- Paper von Benno Stein, Michael Busch *Density-based Cluster Algorithms in Low-dimensional and High-dimensional Applications*, Universität Weimar, 2005

Kapitel 2:

- Buch von Prof. Dr. Jörg Frochte, *Maschinelles lernen*, 3.Auflage, 20 November 2020, (S.308 -317)
- Kapitel 3: Buch von Prof. Dr. Jörg Frochte, *Maschinelles lernen*, 3.Auflage, 20 November 2020, (S.308 -317)

Unsupervised learning - Clusteranalyse

Literaturverzeichnis

Kapitel 4:

- Analytik Vidhya, <https://medium.com/analytics-vidhya/understanding-principle-component-analysis-pca-step-by-step-e7a4bb4031d9>, 7 Januar 2020
- Alind Gupta, <https://www.geeksforgeeks.org/silhouette-algorithm-to-determine-the-optimal-value-of-k/>, 6 Januar 2019