




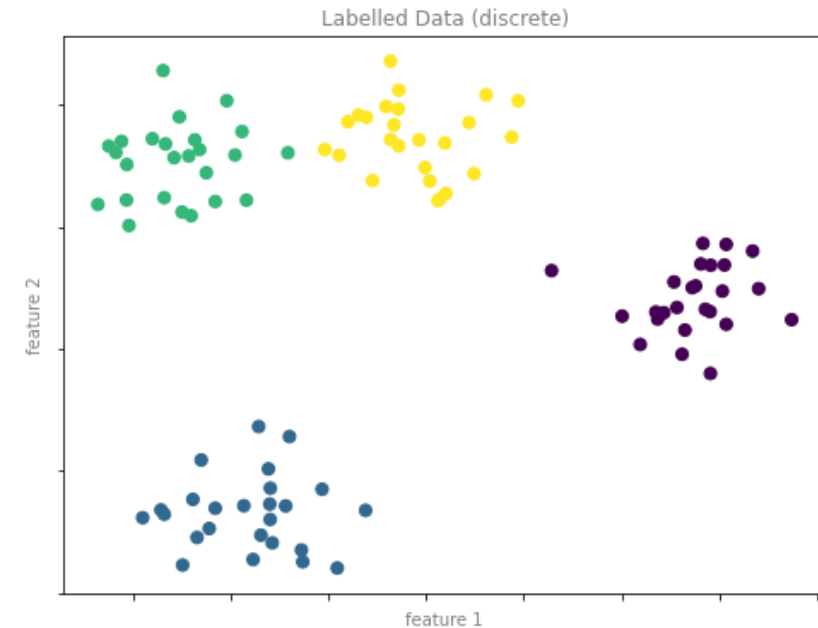
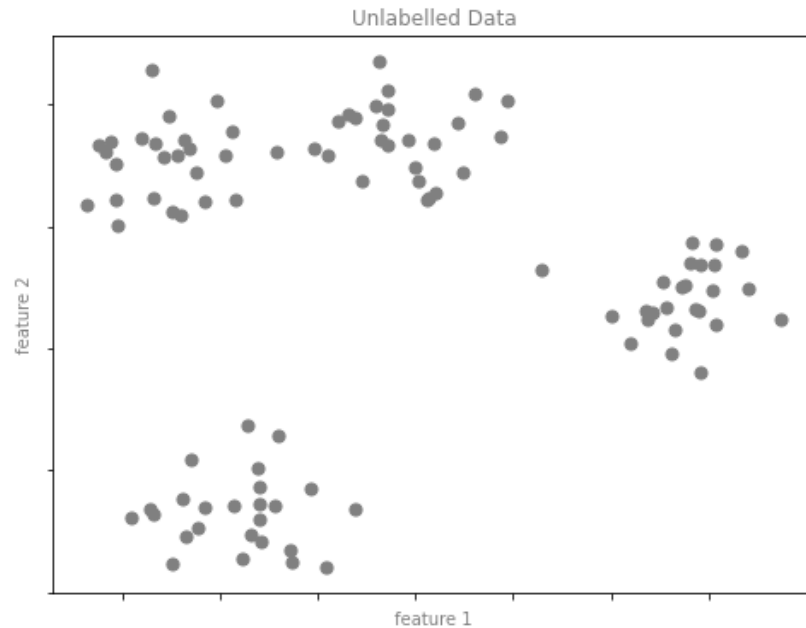
Modeling

Unsupervised Learning: Clustering

Agenda

- 
- Clustering im Allgemeinen
 - k-means Clustering: generell und in `sklearn`
 - Expectation-Maximization Algorithmus
 - k-means im Detail
 - Nachteile des EM-Algorithmus
 - Validierung von Clustering
 - Variationen k-means
 - Dichtebasiertes Clustering: DBSCAN

Was sehen Sie?

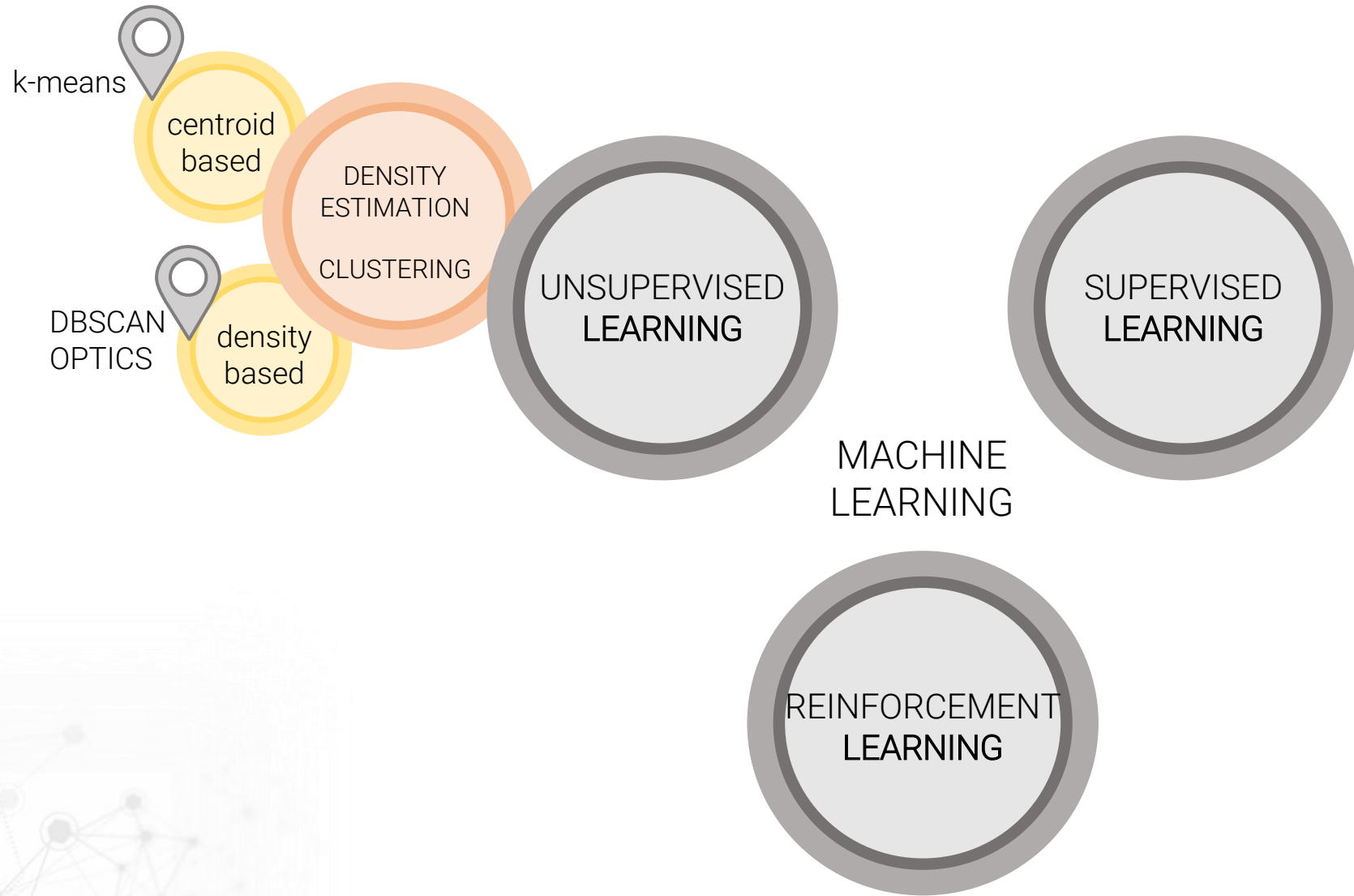


0

So what?

- Ihr visueller Apparat entdeckt automatisch und ohne Mühe Gruppierungen bzw. Struktur in Daten
- Wie bringen wir das Algorithmen bei?¹

SVM: wo befinden wir uns?



Clustering im Allgemeinen



Was denken Sie?

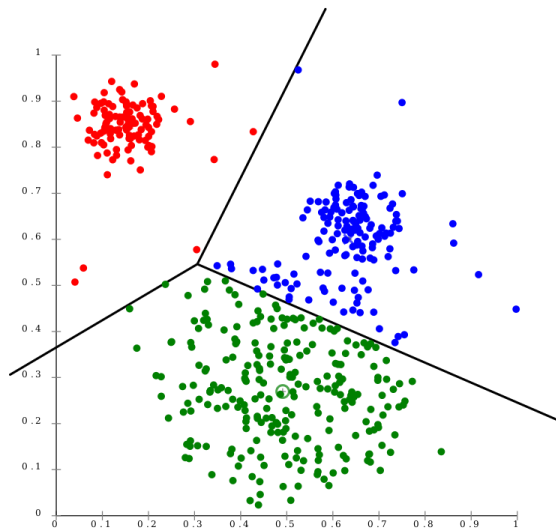
Wie würden Sie mit eigenen Worten beschreiben, was ein Clustering bewirkt bzw. durchführt?



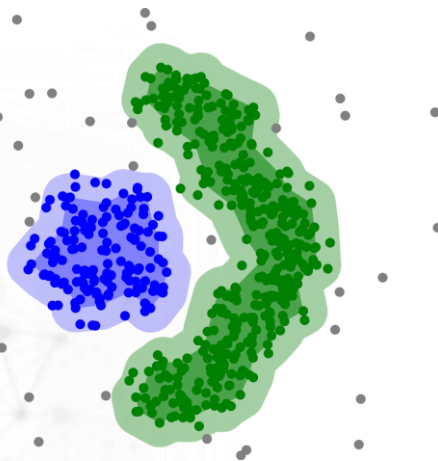
Was denken Sie?

Was heißt hier optimal?

Partitionierend



Dichtebasiert

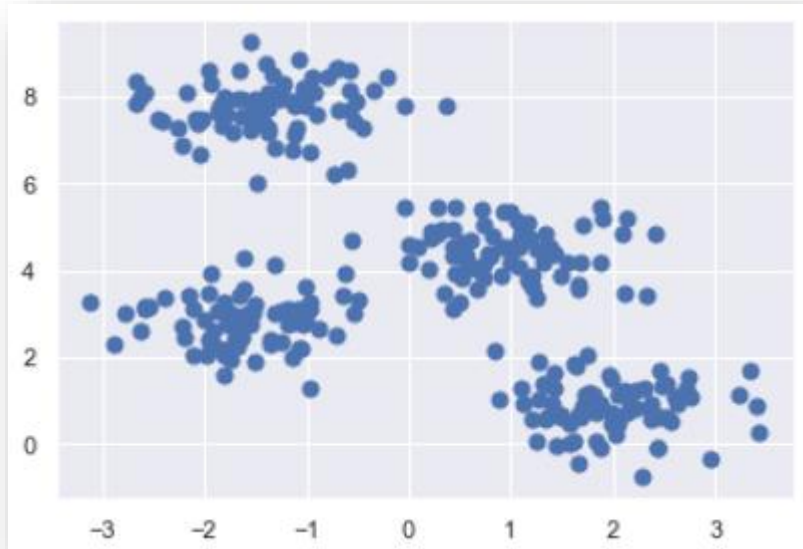


- Clustering Algorithmen versuchen aus den Eigenschaften der Daten eine **optimale Unterteilung in Gruppen** zu erreichen → diskrete Labels
- Es gibt **verschiedene Arten des Clusterings**:
 - Partitionierendes Clustering (z.B. k-means und Varianten)
 - Dichtebasiertes Clustering (z.B. DBSCAN, OPTICS)
 - Hierarchisches Clustering
 - Gitterbasierte Verfahren

k-means: im Allgemeinen

?

Was denken Sie?
Wie wird ein Cluster definiert?



```
1 X, y_true = make_blobs(n_samples=300, centers=4,  
2                       cluster_std=0.60, random_state=0)  
3 plt.scatter(X[:, 0], X[:, 1], s=50);
```

- Wir beginnen mit dem *k-means* Algorithmus
- Der k-means Algorithmus sucht nach einer vorabbestimmten Anzahl an Clustern in einem nicht gelabelten Datensatz
- Heuristik des k-means Algorithmus:
 - Der Cluster-Mittelpunkt (== **Centroid**) ist das arithmetische Mittel aller Punkte des jeweiligen Clusters
 - Jeder Datenpunkt ist **näher** an seinem eigenen Centroid als an anderen

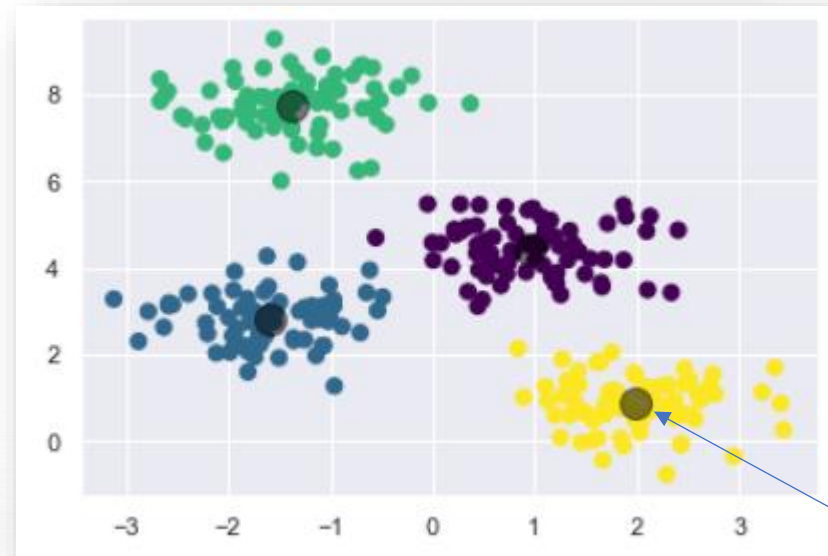
0

So what?

Unsere Leitfrage: Wie macht der k-means Algorithmus das, was wir „mit dem Auge“ mühelos können?

k-means: sklearn

```
1 from sklearn.cluster import KMeans
2 kmeans = KMeans(n_clusters=4)
3 kmeans.fit(X)
4 y_kmeans = kmeans.predict(X)
```



Centroid

```
1 plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
2
3 centers = kmeans.cluster_centers_
4 plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

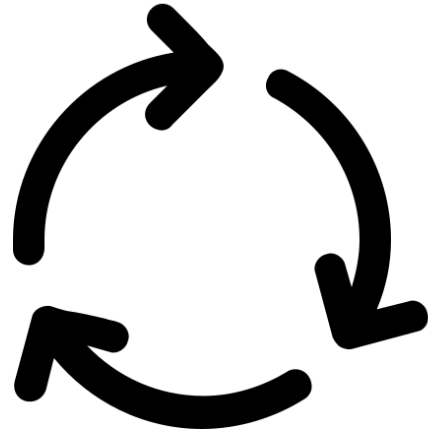
- In sklearn weist der k-means Algorithmus die gleiche Struktur wie andere Modelle auf
- Als wichtiger Zusatz bei der **Instanziierung** des Modells müssen wir die **Anzahl an Clustern** angeben
- Die **Zugehörigkeit** bzw. **Labels** der Datenpunkte (auch neuer bzw. ungesehener Datenpunkte) zu den jeweiligen Centroids erhalten wir über die `.predict()`-Methode
- Die erlernten Centroids stecken im `.cluster_centers_`-Attribut des Modells



Was denken Sie?

Wie findet der Algorithmus die Centroids ohne alle möglichen Kombinationen explizit zu evaluieren?

Einschub: Expectation-Maximization



- Die Heuristik der vorletzten Folie wird umgesetzt – ein „Rezept“ entsteht → die allgemeine Form dieses „Rezeptes“ ist der sog. **Expectation-Maximization-Algorithmus**
- Der EM-Algorithmus findet in vielen Machine Learning Modellen Anwendung
- Es handelt sich um einen **iterativen** Algorithmus
- **Generelle Idee:**
 1. Initialisiere eine **zufällige** Konfiguration des Modells
 2. Führe folgende zwei Schritte abwechselnd bis **Konvergenz** aus:
 - i. **E(xpectation)-Schritt:** Zuordnung der Daten zu den einzelnen Teilen des Modells
 - ii. **M(aximization)-Schritt:** Parameter an die neueste Zuordnung anpassen
- Konvergenz: findet keine **wesentliche** Verbesserung mehr statt → Abbruch

Einschub: Expectation-Maximization



Was denken Sie?

Was sind die analogen Schritte beim k-means?

Initialisiere Centroids zufällig

Weise Datenpunkte zum nächsten Centroid zu

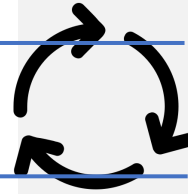
Berechne die neuen Centroids als arithmetische Mittel der neuen Datenpunkte

0

So what?

- *Expectation*: bedeutet beim k-means also „unsere Erwartung“ updaten, zu welchem Cluster die Datenpunkte gehören
- *Maximization*: ein Gütemaß wird maximiert (in unserem Fall durch die Berechnung des Mittelwertes)

- Die Heuristik der vorletzten Folie wird umgesetzt – ein „Rezept“ entsteht → die allgemeine Form dieses „Rezeptes“ ist der sog. *Expectation-Maximization-Algorithmus*
- Der EM-Algorithmus findet in vielen Machine Learning Modellen Anwendung
- Es handelt sich um einen **iterativen** Algorithmus
- **Generelle Idee:**
 1. Initialisiere eine **zufällige** Konfiguration des Modells
 2. Führe folgende zwei Schritte abwechselnd *bis Konvergenz* aus:
 - i. **E(xpectation)-Schritt**: Zuordnung der Daten zu den einzelnen Teilen des Modells
 - ii. **M(aximization)-Schritt**: Parameter an die neueste Zuordnung anpassen
- Konvergenz: findet keine **wesentliche** Verbesserung mehr statt → Abbruch



k-means: Input und Output

Input

\mathbf{X} : Data matrix with dimensions $N \times T$

N : number of observations

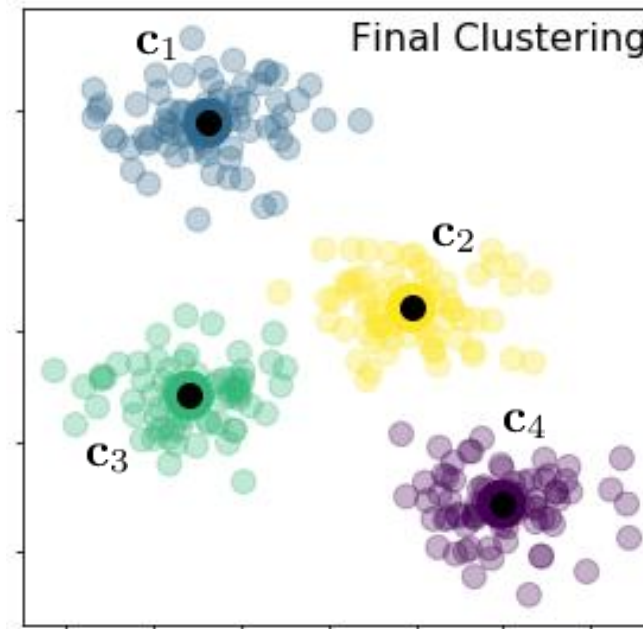
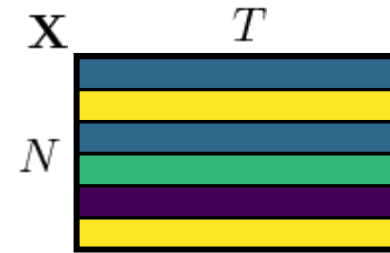
T : number of features (e.g. \bar{x} , σ^2 , ...)

k : number of clusters to be extracted

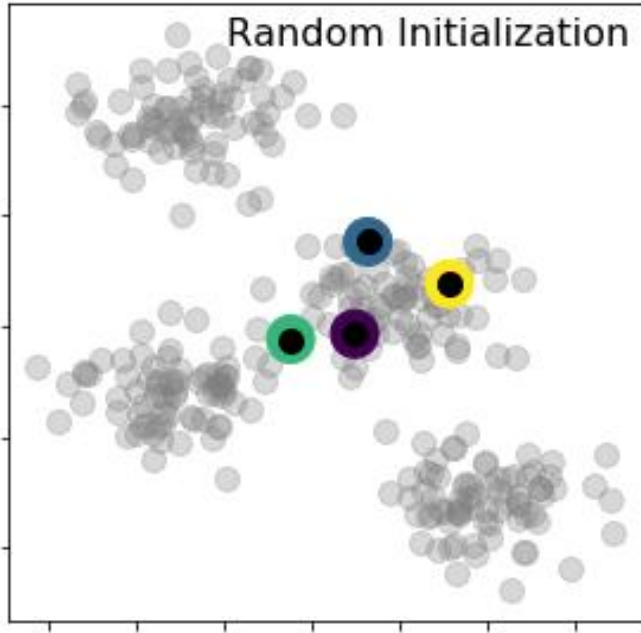
Output

\mathbf{c}_z : cluster centroids with $z = 1, \dots, k$

\hat{z}_t : assigned cluster index of data point \mathbf{x}_t



k-means: Ausführung



1. Initialization:
Initialize centroids \mathbf{c}_z
randomly

2. Expectation step:
Assign to nearest centroid

$$\hat{z}_t = \arg \min_z (d(\mathbf{c}_z, \mathbf{x}_t))$$

3. Maximization step:
Calculate new centroid

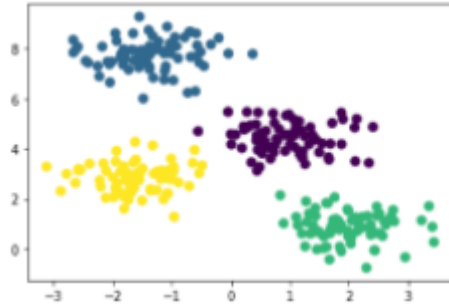
$$\mathbf{c}_z \leftarrow \frac{\sum_{t=1}^T u(t, z, \hat{z}_t) \mathbf{x}_t}{\sum_{t=1}^T u(t, z, \hat{z}_t)}$$

with

$$u(t, z, \hat{z}_t) = \begin{cases} 1, & \text{if } \hat{z}_t = z \\ 0, & \text{else.} \end{cases}$$

4. Repeat
2. and 3. until
convergence

Beispiel: k-means from Scratch



Der k-means Algorithmus ist so anschaulich, dass wir diesen Algorithmus "from Scratch" - also im Detail - nachprogrammieren können. Unsere Aufgabe in diesem Beispiel ist es also, den k-means Algorithmus Schritt für Schritt zu implementieren. Schreiben Sie also eine Funktion, die als Eingabeargumente die Daten und die Anzahl zu extrahierender Cluster aufnimmt - und uns die Centroids und Labels der Daten zurückgibt.

Nachteile des EM-Algorithmus



Was denken Sie?
Was könnten Nachteile
dieses Algorithmus sein?



Was denken Sie?
Was tun wir dagegen?



Was denken Sie?
Wie könnten wir das in unserer
Implementierung einbringen?

Das *globale Optimum* wird möglicherweise nicht gefunden

- Der EM-Algorithmus „verbessert sich“ zwar von Iteration zu Iteration, jedoch ist eine Konvergenz ins globale Optimum nicht garantiert
→ Abhängigkeit von (u.a.) Initialbedingungen
- Daher startet man den Algorithmus **häufig** für **unterschiedliche** Startbedingungen
- und **wählt** dann das **beste Ergebnis** aus (oder aggregiert die Ergebnisse)
- `sklearn` führt diese mehrfache Initialisierung mit unterschiedlichen Anfangsbedingungen per default aus – einstellbar mittels des Eingabearguments `n_init`



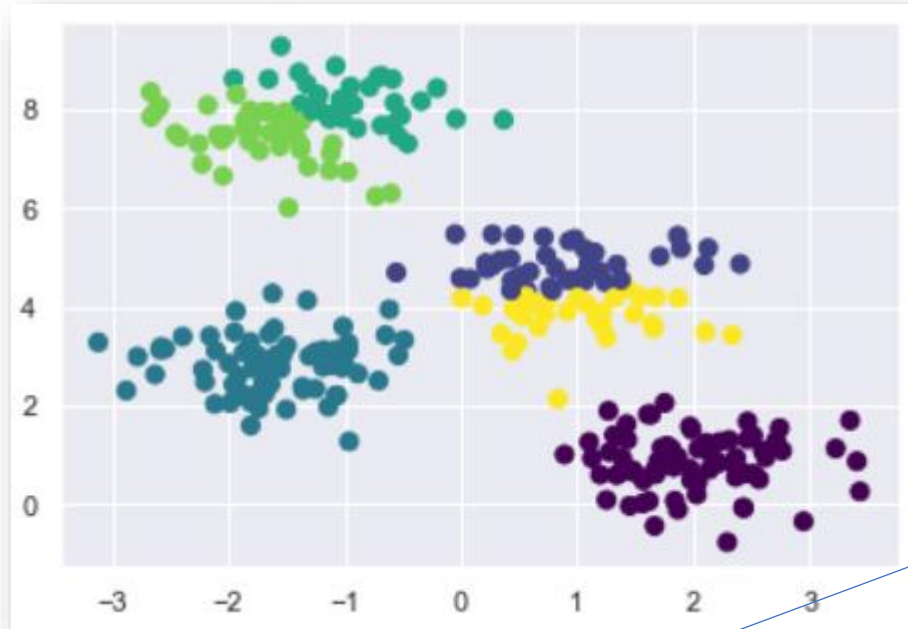
Was denken Sie?
Was machen wir mit
den Ergebnissen?

Nachteile des EM-Algorithmus



Was denken Sie?

Was könnte man tun um eine geeignete Anzahl an Clustern zu bestimmen?



```
1 kmeans = KMeans(6, random_state=0)
2 labels = kmeans.fit_predict(X)
3 plt.scatter(X[:, 0], X[:, 1], c=labels,
4             s=50, cmap='viridis');
```

Die Anzahl an Clustern muss vorgegeben werden – der k-means lernt diese nicht eigenständig aus den Daten

- Hierbei handelt es sich um ein Problem, das viele *Unsupervised* Algorithmen betrifft
- Konkret können wir unserem k-means Modell aus `sklearn` eine beliebige Anzahl an zu extrahierenden Clustern vorgeben
- Man benötigt ein *Validierungskriterium*, um entscheiden zu können, welche Anzahl an extrahierten Clustern geeignet ist

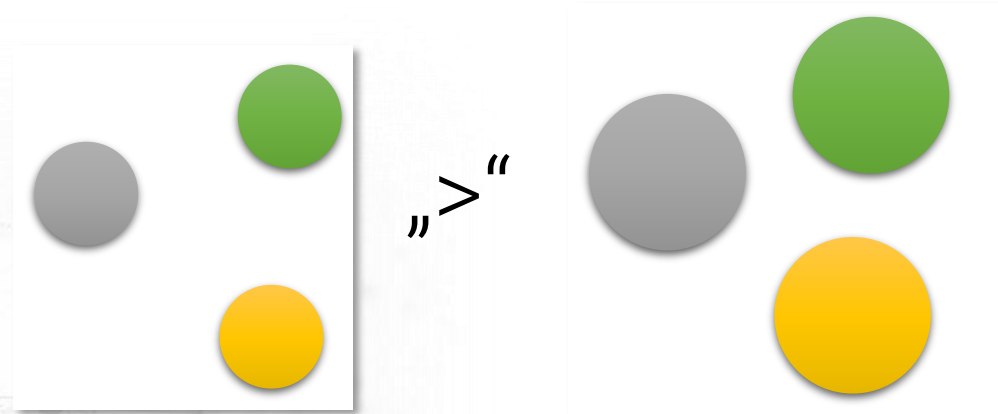
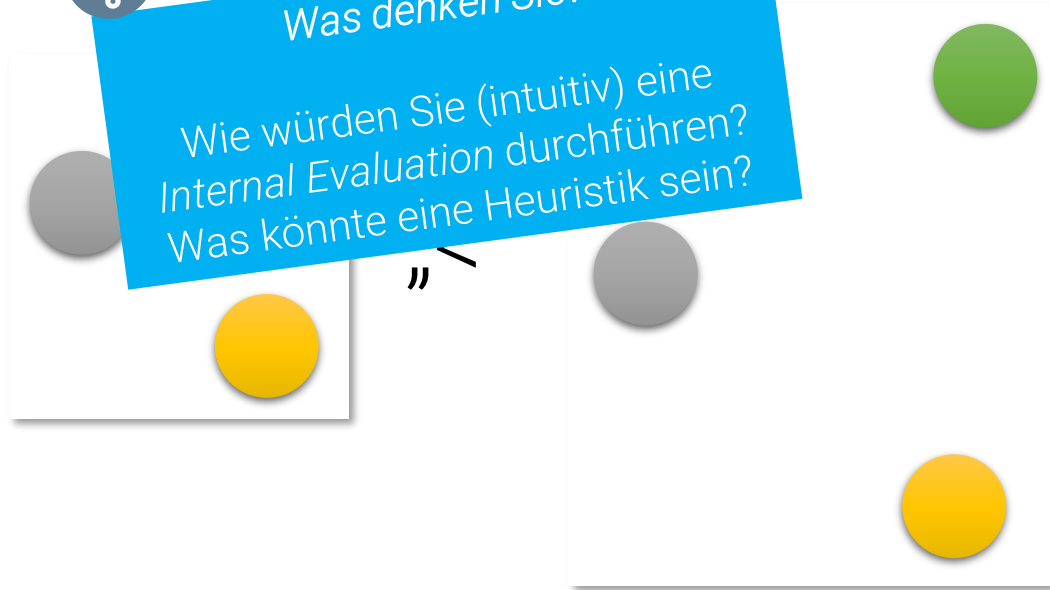
Validierung von Clustering



Was denken Sie?

Wie würden Sie (intuitiv) eine Internal Evaluation durchführen?
Was könnte eine Heuristik sein?

„>“



„>“



Was denken Sie?

Wie würden Sie (intuitiv) bewerten, ob Ihr Clustering „gut“ war?

Verschiedene, grundlegende Möglichkeiten der Cluster-Validierung:

- **Internal Evaluation:** das Clustering wird durch ein Gütemaß repräsentiert und bewertet
- **External Evaluation:** das Clustering wird mit einer „Ground Truth“ an Labels verglichen
- **Manual Evaluation:** manuelle Evaluation durch einen Fachexperten
- **Indirect Evaluation:** Evaluation durch Anwendung des Clustering-Ergebnisses

0

So what?

- Cluster-Validitätskriterien bewerten nach Heuristiken wie
- „inter-Cluster-Ähnlichkeit“ so gering wie möglich bzw. „-Distanz“ so groß wie möglich – und zugleich
 - „intra-Cluster-Ähnlichkeit“ so hoch wie möglich bzw. „-Streuung“ so niedrig wie möglich

Validierung von Clustering: Maße



Dunn-Index:

- Intuitives Maß zur Bewertung eines Clusterings basierend auf dem Verhältnis aus der **Inter-Cluster-Distanz** $\delta(C_i, C_j)$ zur **Intra-Cluster-Distanz** Δ_i

$$DI_m = \frac{\min_{1 \leq i < j \leq m} \delta(C_i, C_j)}{\max_{1 \leq k \leq m} \Delta_k}$$

wobei m der Anzahl an Clustern und C_i den Centroids entspricht

- Für die Inter-Cluster-Distanz könnte man die Abstände zwischen den Centroids definieren
- Für die Intra-Cluster-Distanz das Maximum der Abstände zwischen Datenpunkte innerhalb eines Clusters



Demo
Gemeinsam am
Whiteboard herleiten



Was denken Sie?

Können Sie sich das min
und max erklären?

Davies-Bouldin Index:

- Weniger intuitives Maß als der Dunn-Index
- Bewertet die Trennung zwischen den Clustern und die Streuung innerhalb der Cluster
- Er ist definiert als:

$$DB = \frac{1}{m} \sum_{i=1}^m \max_{i \neq j} \left(\frac{s_i + s_j}{d(C_i, C_j)} \right)$$

- Wobei s_i die Streuung der Abstände aller Datenpunkte zum jeweiligen Centroid innerhalb des Clusters i darstellt
- In `sklearn.metrics` gibt es eine Funktion hierfür
`davies_bouldin_score(X, y)`

Validierung von Clustering: Maße

Silhouette Score: Beschreibung

- Bewertet, wie **ähnlich** ein Datenpunkt zu seinem **eigenen** und wie **unähnlich** zu den **übrigen Clustern** ist
- Er ist so definiert, dass sein **Wertebereich** zwischen -1 und 1 liegt
- Interpretation der Wertebereiche:
 - **Nahe 1**: Datenpunkt liegt in einem Cluster
 - **Um 0**: Datenpunkt liegt **zwischen** zwei Clustern
 - **< 0**: Datenpunkte eines nahegelegenen Clusters liegen näher am betrachteten Datenpunkt → Clustering kann verbessert werden
- Weisen also viele Datenpunkte einen **hohen Silhouette-Score** auf, dann liegt ein **geeignetes Clustering** vor
- Typisch sind auch Plots des Silhouette Scores

Silhouette Score: Definition

- Wir benötigen den mittleren Abstand eines Datenpunktes zu allen anderen eines Clusters A

$$\text{dist}(A, o) = \frac{1}{n_A - 1} \sum_{a \in A, a \neq o} \text{dist}(a, o)$$

- Distanz zum nächstgelegenen Cluster B : den kleinsten, mittleren Abstand eines Datenpunktes zu allen anderen Datenpunkten der nicht-eigenen Cluster C

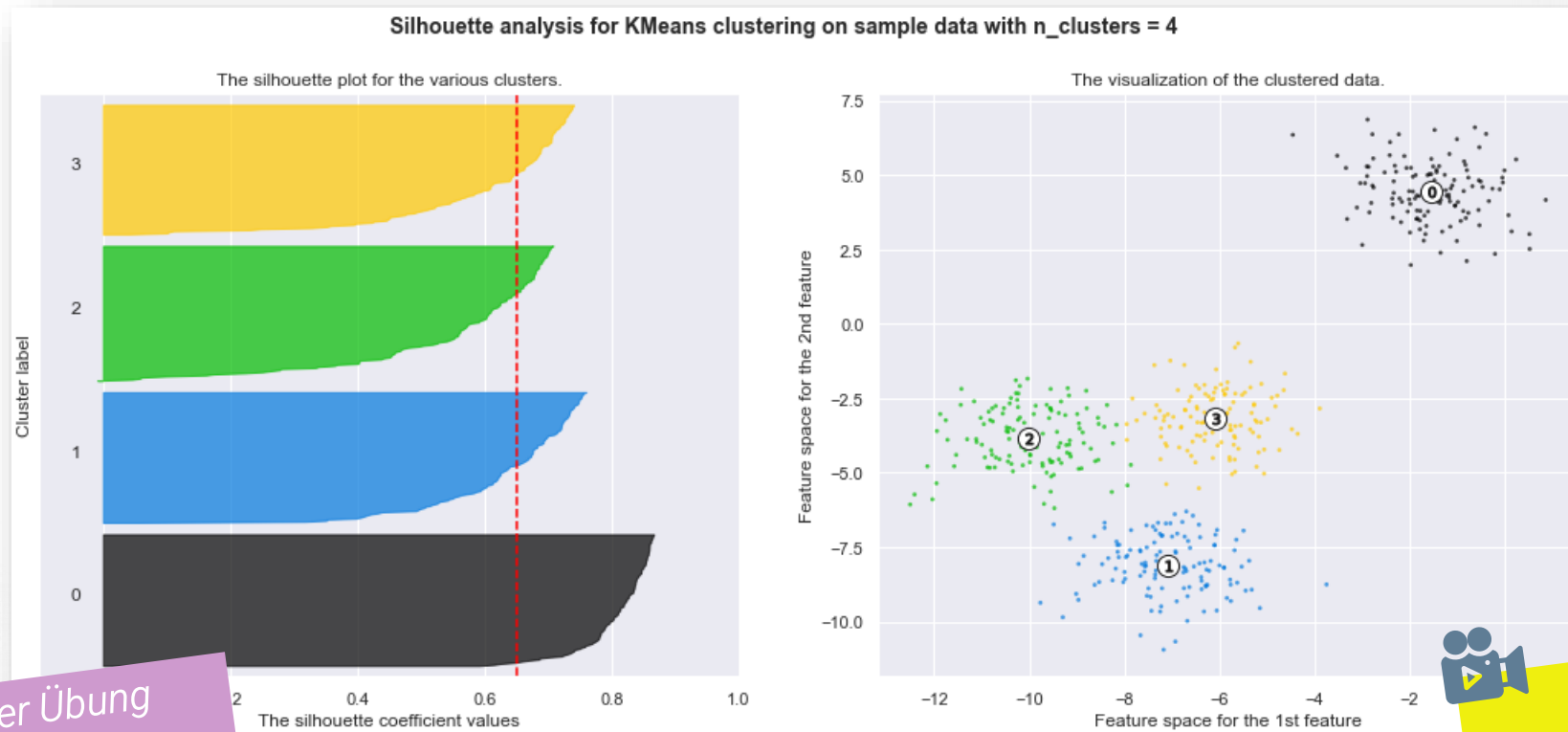
$$\text{dist}(B, o) = \min_{C \neq A} \underbrace{\left(\frac{1}{n_C} \sum_{c \in C} \text{dist}(c, o) \right)}_{=\text{dist}(C, o)}$$

- Damit definieren wir den Silhouette Score für einen Datenpunkt

$$S(o) = \begin{cases} 0 & \text{wenn } o \text{ einziges Element von } A \text{ ist} \\ \frac{\text{dist}(B, o) - \text{dist}(A, o)}{\max\{\text{dist}(A, o), \text{dist}(B, o)\}} & \text{sonst} \end{cases}$$

- Und der Mittelwert hiervon über alle Datenpunkte ist ein *Gütemaß* für einen Clustering-Vorgang

Validierung von Clustering: Silhouette-Plot



Möglichkeit einer Übung
Cluster-Validierung kann
problematisch sein: sehen wir
uns in einer Übungsaufgabe
noch genauer an



Demo
Ausführung im
Jupyter Notebook

Wie lesen und interpretieren wir einen Silhouette-Plot?



Interpretationshilfen/-heuristiken

- Gibt es viele Cluster/Datenpunkte mit Silhouette-Scores **kleiner** als der **durchschnittliche Score**, so spricht das für ein schlechtes Clustering
- **Negative Silhouette-Scores** sind immer kritisch zu betrachten und deuten häufig auf ein schlechtes Clustering hin
- Allgemein: eine heuristische – aber nicht pauschal zutreffende – Annahme: **natürliche Cluster haben oft vergleichbare Größen**. Gibt es starke Unterschiede → evtl. schlechtes Clustering

Validierung von Clustering: Maße

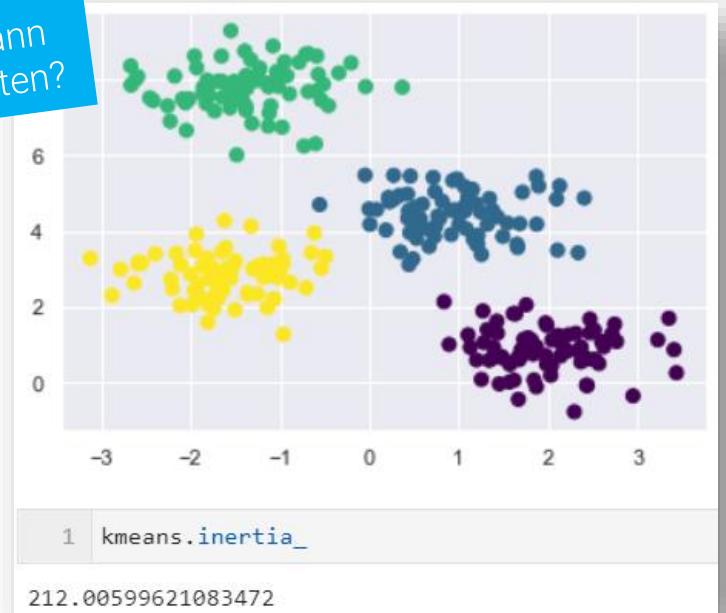
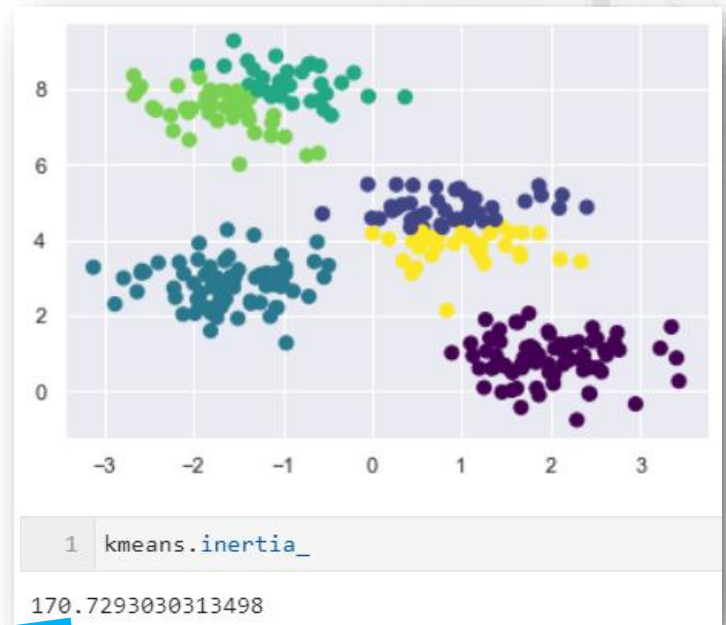
SSE und Elbow Plots:

- Eine weitere, einfachere Möglichkeit zur Validierung bietet die Betrachtung der Sum of Squared Errors bzw. Sum of Squared Distances
- Hierzu berechnet man die Summe der quadratischen Abstände zwischen allen Datenpunkten und ihren jeweilig zugehörigen Centroids
- In `sklearn` liegt dieser Wert unter dem Attribut `.inertia_` des Modells
- Dieses Maß trägt man dann in einem sog. *Elbow Plot* bzw. *Knee Plot* bzw. *Scree Plot* auf
- Die optimale Anzahl an Clustern liegt vor, wenn dieser Graph einen „Knick macht“
- Grundgedanke: ab diesem Punkt erklären weitere Cluster zu wenig Varianz der Daten

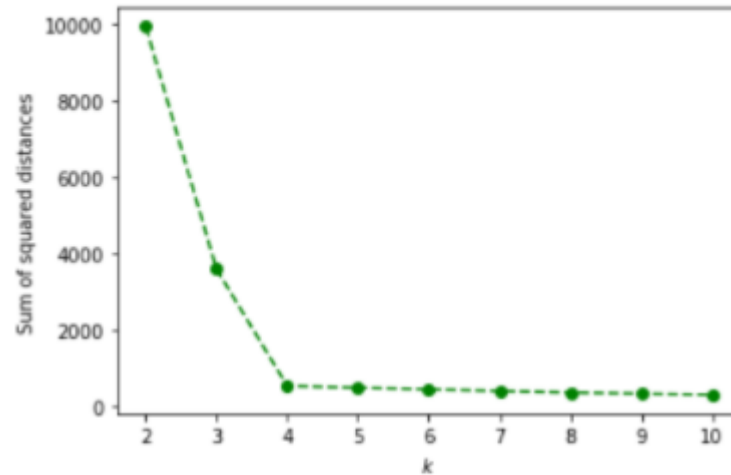


Was denken Sie?

Was tun wir dann mit diesen Werten?

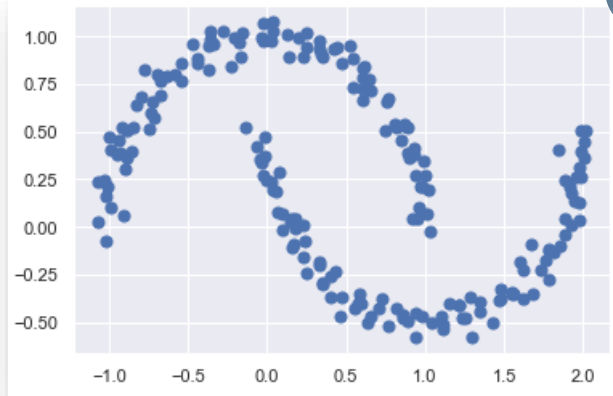


Beispiel: Cluster-Validierung mittels Elbow-Plot



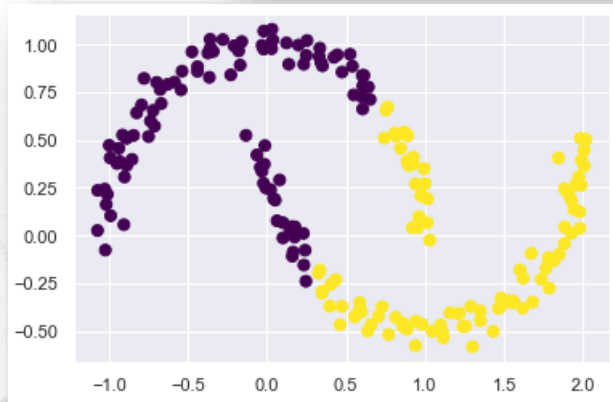
In unserem selbst erzeugten Beispiel stecken vier Cluster. Wir wollen nun mittels der Elbow-Plot-Methode validieren, dass auch wirklich ein Clustering, das vier Cluster extrahiert, die Struktur der Daten am besten repräsentiert. Unsere Aufgabe ist also mittels eines `for`-Loops k-means mit ansteigendem k an unseren Daten zu trainieren und die Sum of Squared Distances eines jeden Trainingsdurchgangs in einem Linienplot darzustellen.

Nachteile des EM-Algorithmus



Was denken Sie?
Wie wird k-means
hier performen?

```
1 labels = KMeans(2, random_state=0).fit_predict(X)
2 plt.scatter(X[:, 0], X[:, 1], c=labels,
3             s=50, cmap='viridis');
```

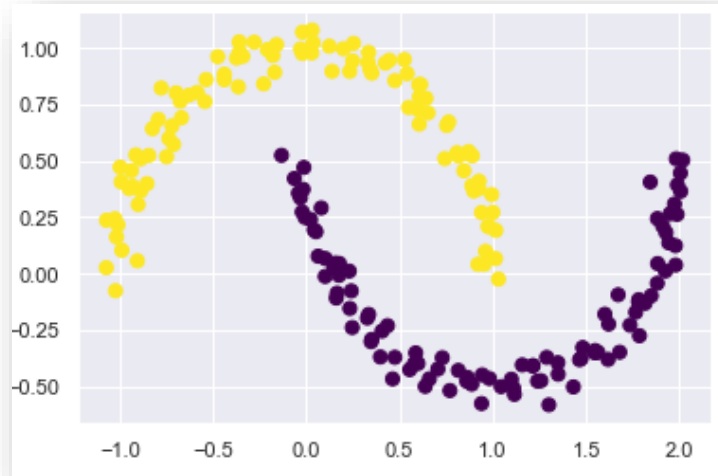


Was denken Sie?
Welche Herangehensweise
könnten wir hierzu aus
anderen Modellen entleihen?

k-means ist nur für linear-separierbare Probleme geeignet

- Die grundlegende Annahme des *k-means* „Datenpunkte sind näher am eigenen Cluster als an anderen“ führt zu Problemen bei „komplizierten Geometrien“
- Die **Grenzen** zwischen Clustern, die aus dem *k-means* resultieren, sind immer **linear**
- Mittels des **Kernel-Tricks**, den wir bei SVMs kennengelernt hatten, kann man *k-means* auch zur Lösung **nicht-linear** separierbarer Probleme befähigen

Variationen des k -means Algorithmus



Spectral Clustering

- Mittels graphentheoretischer Maße werden die Daten in einen **höherdimensionalen** Raum projiziert
- In diesem **höherdimensionalen** Raum kann k -means auch mit **linearen** Grenzen Cluster **separieren**, die im niedrigdimensionalen Raum nur nicht-linear separierbar wären

```
1 from sklearn.cluster import SpectralClustering
2 model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
3                             assign_labels='kmeans')
4 labels = model.fit_predict(X)
5 plt.scatter(X[:, 0], X[:, 1], c=labels,
6             s=50, cmap='viridis');
```

Variationen des k-means Algorithmus



Was denken Sie?

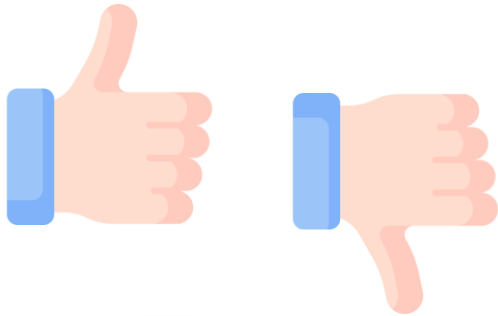
Was würden Sie intuitiv am k-means Algorithmus (noch) verbessern?



Weitere Varianten des k-means Clustering:

- **k-means++:**
 - Centroids werden nach bestimmter **Heuristik initialisiert** (Wahrscheinlichkeit der Initialisierung höher, je weiter von einem schon initialisierten Centroid entfernt)
 - **Konvergiert schneller** – gleiche Ergebnisse wie k-means
- **k-Median:**
 - **anderes Distanzmaß** als die euklidische Distanz → Manhattan Distanz
 - **Median** anstatt Mittelwert im M-Schritt
- Und viele weitere...

Vorteile, Nachteile, Besonderheiten: k-means



Pros:

- Schnell und effizient
- Leicht verständlich und implementierbar
- Anwendbar auf viele verschiedene Datenarten

Cons:

- Stark abhängig von den Initialisierungsbedingungen
→ Annäherung an globales Optimum schwierig
- Mehrere Initialisierungen nötig, um Obigem Rechnung zu tragen
- Ohne Trick: nur linear separierbare Probleme lösbar
– „natürlichere“/realistischere Cluster schwierig zu entdecken
- Ausreißer haben großen Einfluss (auf *k-means*)
- Probleme bei unterschiedlich großen/dichten Clustern

Beispiel: Bestimmung des Betriebszustandes von alten Brownfield-Maschinen



Im Maschinenbau gibt es den sog. *Weihenstephaner Standard*, in dem verschiedene Maschinenzustände kodiert sind. Beispielsweise gibt es die Zuordnungen:

- 4 : Prepared
- 8 : Mangel
- 16 : Stau
- 128 : Produktion
- 1024 : Eigenstörung
- 4096 : Fremdstörung

Bei neueren Maschinen gibt es in der Maschine Schnittstellen, die diese Zustände ausgeben können. Bei älteren Maschinen, die schon lange im Feld sind und bei denen Neuerungen zugekauft werden müssen (sog. *Retrofits* im *Brownfield*), könnte es passieren, dass diese keine Betriebszustandsangaben machen können. Für moderne Funktionalitäten sind diese jedoch essentiell. Daher könnte es für ein Maschinenbauunternehmen von Interesse sein auch bei älteren Maschinen datengetriebenen Betriebszustände zu bestimmen.

Hierzu wollen wir das k -means Clustering nutzen. Uns liegen Maschinendaten in Form von kurzzeitigen (1-minütigen) *Snapshots* eines Vibrations- und eines Temperatursensors am Motor einer Maschine vor. Wie können wir diese Daten und unser Clustering nutzen, um nach Betriebszuständen zu suchen? Nach was würden wir Ausschau halten? Haben wir einen Anhaltspunkt für k ?

0

So what?

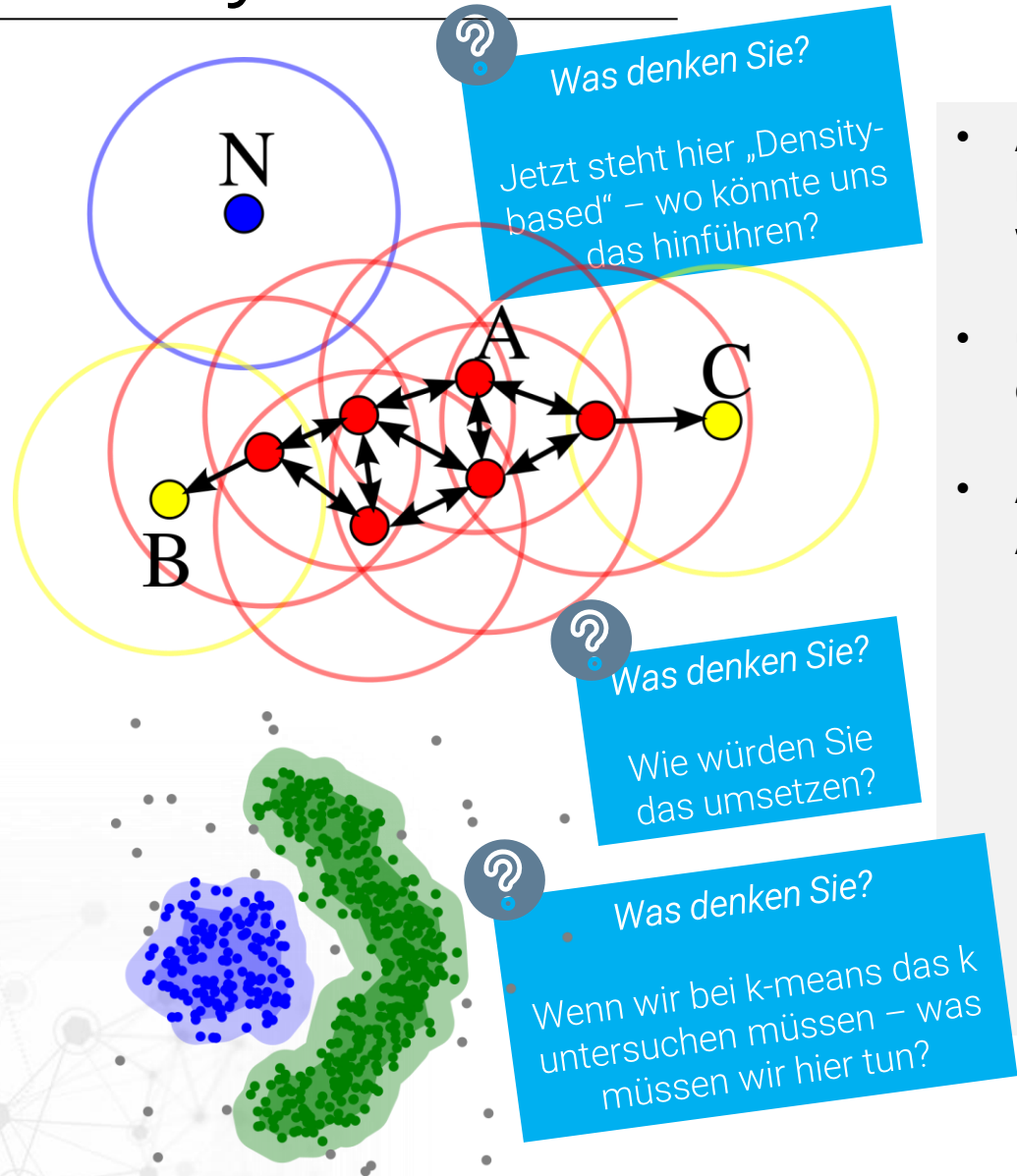
- Wir sehen: Scaling kann helfen – aber wir können nicht pauschal behaupten, dass es immer angewendet werden muss/kann!
- Kein einfaches Unterfangen hier eine geeignete Anzahl an Clustern zu entdecken – wir sollten eine neue Clustering-Methode kennenlernen!

Noch ein paar Worte zu: Scaling



- Wenn wir uns die Frage stellen, ob wir skalieren sollen oder nicht, dann kann man diese Frage auch umformulieren in: **in welcher Situation liegt mir ein geeignetes Distanzmaß vor?**
- Wenn uns z.B. zwei Features vorliegen – bei einem sind die Differenzen größer, beim anderen kleiner: ist es gewollt, dass das **Feature mit den größeren Differenzen** auch **größeres Gewicht** bekommt?
- Wenn wir die letzte Frage mit ja beantworten, dann sollten wir kein Scaling durchführen – andernfalls schon!

Density-based Clustering



- Anstatt Cluster explizit anhand eines Abstandsmaßes zu definieren bzw. extrahieren, kann man dies durch die **Abgrenzung dichter von weniger dichten** Regionen durchführen → *Density-based*
- Großer Vorteil: **beliebige** Cluster-Formen können erkannt bzw. extrahiert werden
- Algorithmus **DBSCAN** (Density-based Spatial Clustering of Applications with **Noise**):
 - Zähle, für **jeden Datenpunkt**, wie viele andere Datenpunkte sich **innerhalb** einer **kleinen Distanz** ϵ_{ps} befinden
→ Hyperparameter *Epsilon-Nachbarschaftsgröße*
 - Falls ein Datenpunkt mindestens min_samples Datenpunkte in seiner Epsilon-Nachbarschaft aufweist, dann gehört dieser zu einer sog. *Core Region (A)*
 - *Density-reachable (B und C)* Datenpunkte beinhalten einen Core Punkt, sind jedoch selbst nicht dicht
 - Alle anderen Datenpunkte werden als *Noise (N)* deklariert

Density-based Clustering: sklearn

```
1 # Import
2 from sklearn.datasets import make_moons
3 from sklearn.cluster import DBSCAN
4
5 # Make data
6 X, y = make_moons(n_samples=1000, noise=0.05, random_state=42)
7
8 # Fit DBSCAN with certain hyperparameter values
9 dbscan = DBSCAN(eps=0.05, min_samples=5)
10 dbscan.fit(X)
```

DBSCAN(eps=0.05)

```
1 dbscan.labels_[:10]
array([ 0,  2, -1, -1,  1,  0,  0,  0,  2,  5], dtype=int64)
```

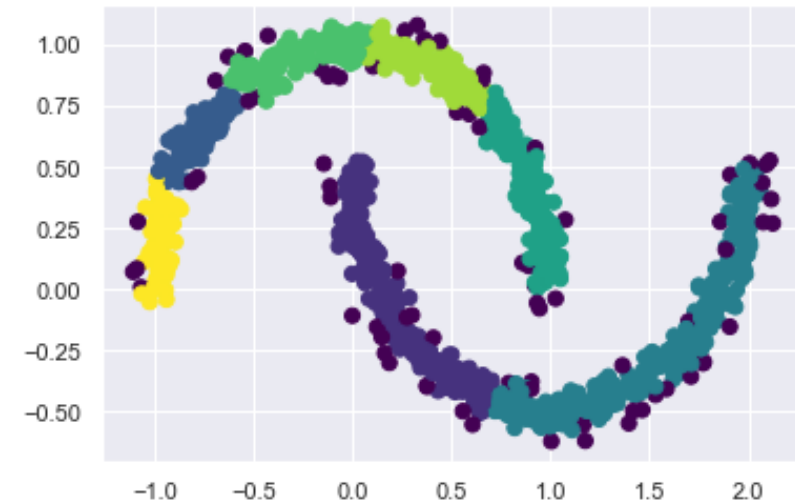
Labels mit Werten von -1?

```
1 np.unique(dbscan.labels_)
array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64)
```

Wtf? 7 Cluster Labels

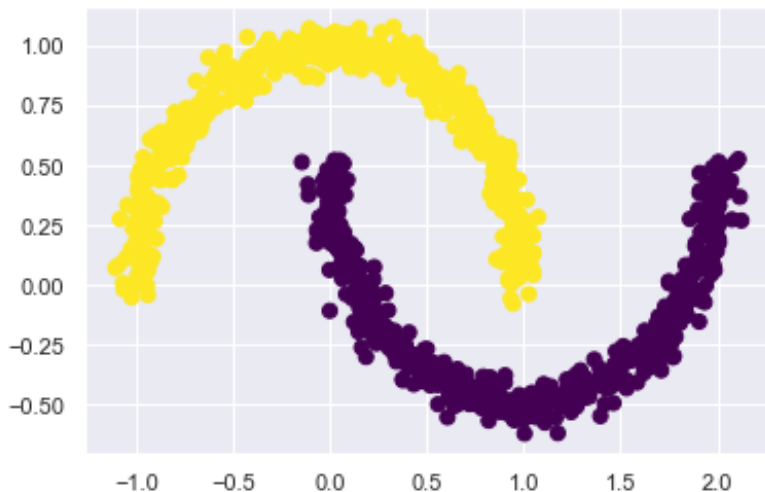
- sklearn hat auch zu DBSCAN eine einfach zu bedienende Klasse
- Wir versuchen das nicht-linear separierbare Problem aus den vorhergehenden Folien mittels DBSCAN zu lösen

```
1 plt.scatter(X[:, 0], X[:, 1], c=dbscan.labels_,
2             s=50, cmap='viridis');
```



Density-based Clustering: sklearn

```
1 # Fit DBSCAN with certain hyperparameter values
2 dbscan = DBSCAN(eps=0.20, min_samples=5)
3 dbscan.fit(X)
4
5 # Plot again
6 plt.scatter(X[:, 0], X[:, 1], c=dbscan.labels_,
7             s=50, cmap='viridis');
```



- Wir sehen hier wieder wie wichtig es ist nach geeigneten Hyperparameter-Einstellungen zu suchen
- Man würde hier einen Optimierungsvorgang in einem zweidimensionalen Hyperparameterraum durchführen – `min_samples` und `eps`
- Die DBSCAN-Klasse aus `sklearn` hat **keine** `.predict()`-Methode – diese wurde bewusst ausgespart
→ User soll frei wählen können, wie neue Datenpunkte zu den jeweiligen Clustern **zugeordnet** werden
- Dies kann z.B. mittels eines *k*-Nearest-Neighbors Classifier geschehen



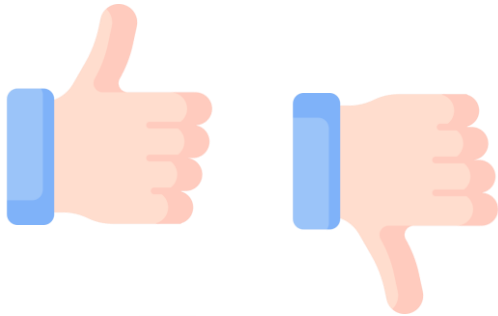
Was denken Sie?

Wie würden Sie das umsetzen?

Beispiel: Assign new points to DBSCAN Clusters

In diesem Beispiel sehen wir uns an wie unbekannte Datenpunkte zu einem, mittels DBSCAN geclusterten, Datensatz hinzugefügt werden. Hierzu nutzen wir den k -Nearest-Neighbors Klassifikator. Dies ist ein intuitiver Klassifikator, den wir hier *on-the-fly* betrachten. Er befindet sich in `sklearn.neighbors` und heißt `KNeighborsClassifier`. Die Klassifizierung eines neuen Datenpunktes geschieht einfach über ein *Majoritätsvotum* der k nächsten Nachbarn. D.h. das Lernen dieses Klassifikators besteht nur aus dem Abspeichern des Datensatzes und dessen Labels! Die Labels erhalten wir also aus dem Training des DBSCAN-Algorithmus auf den vorliegenden Datensatz.

Vorteile, Nachteile, Besonderheiten: density-based



Was denken Sie?
Welche Vor- und Nachteile sehen Sie bei DBSCAN?

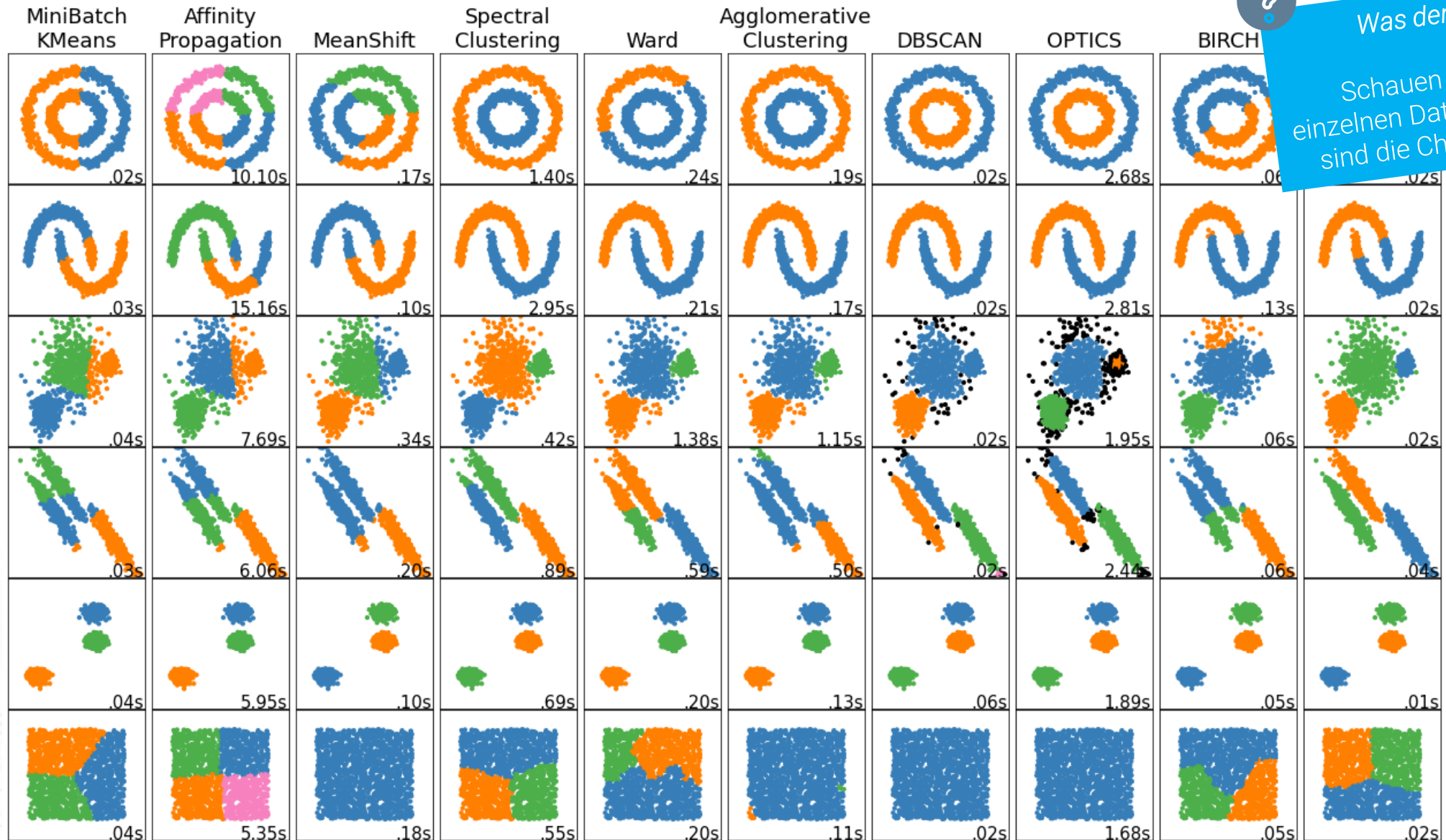
Pros:

- Anzahl an Clustern müssen nicht vorab bestimmt werden
- Cluster beliebiger Form können entdeckt werden
- Beinhaltet Anomalien/Rauschen als „Klasse“ und ist robust gegenüber diesen
- Nur zwei Hyperparameter
- Hyperparameter können gut domänengetrieben gesetzt werden

Cons:

- Nicht vollständig deterministisch → Border-Points
- Auch hier: abhängig vom Distanzmaß
- Probleme bei Clustern mit starken Unterschieden in Dichten → `min_samples`, `eps` Kombination kann nicht für alle Cluster konsistent bestimmt werden

Vergleich verschiedener Clustering-Verfahren



Was denken Sie?

Schauen Sie sich die einzelnen Datensätze an: was sind die Charakteristiken?