

Data Warehousing und Data Mining

Clustering

Ulf Leser

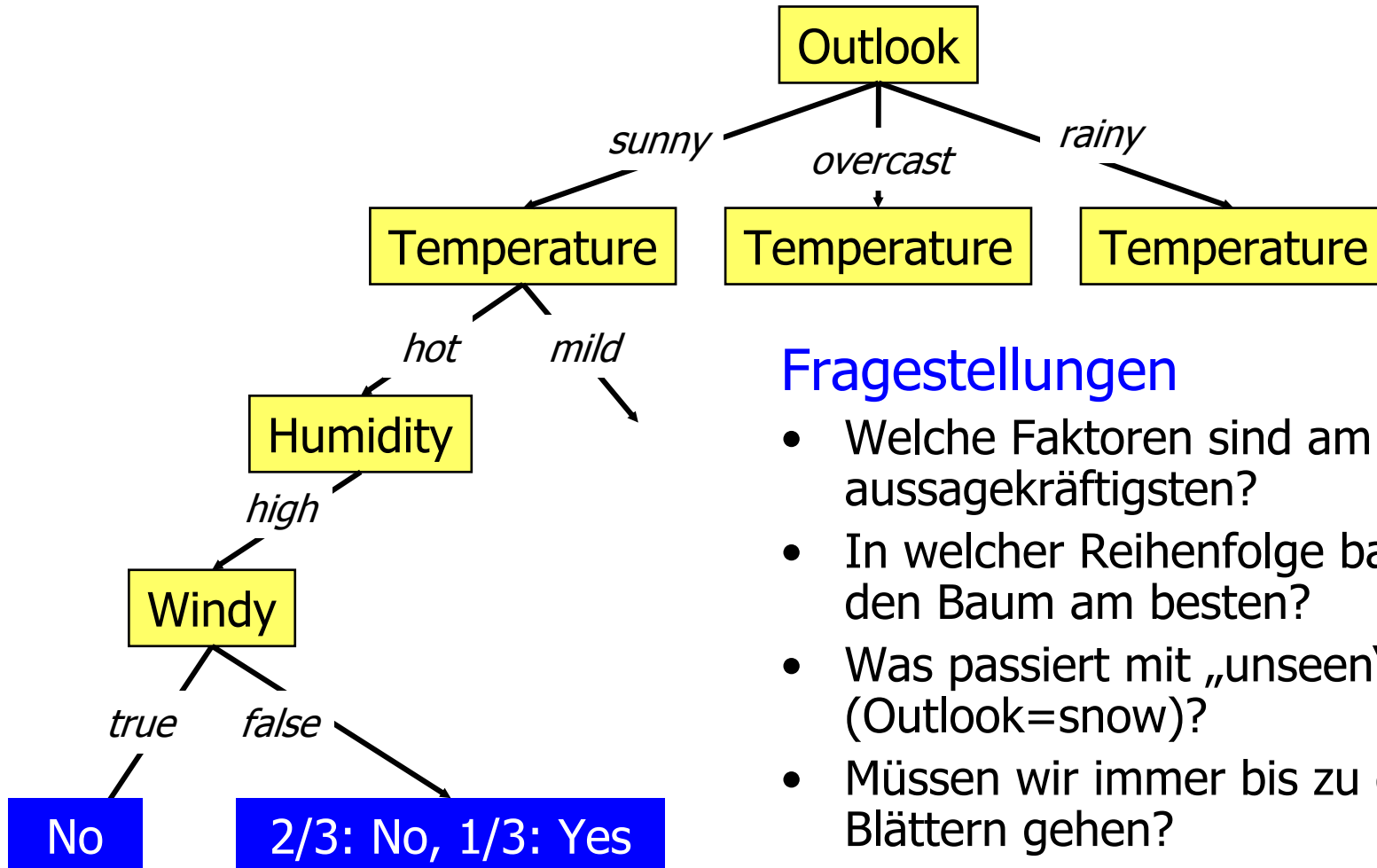
Wissensmanagement in der
Bioinformatik



Mehr Beobachtungen

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	Normal	False	Yes
...
Sunny	Hot	High	False	Yes
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
...

Entscheidungsbäume



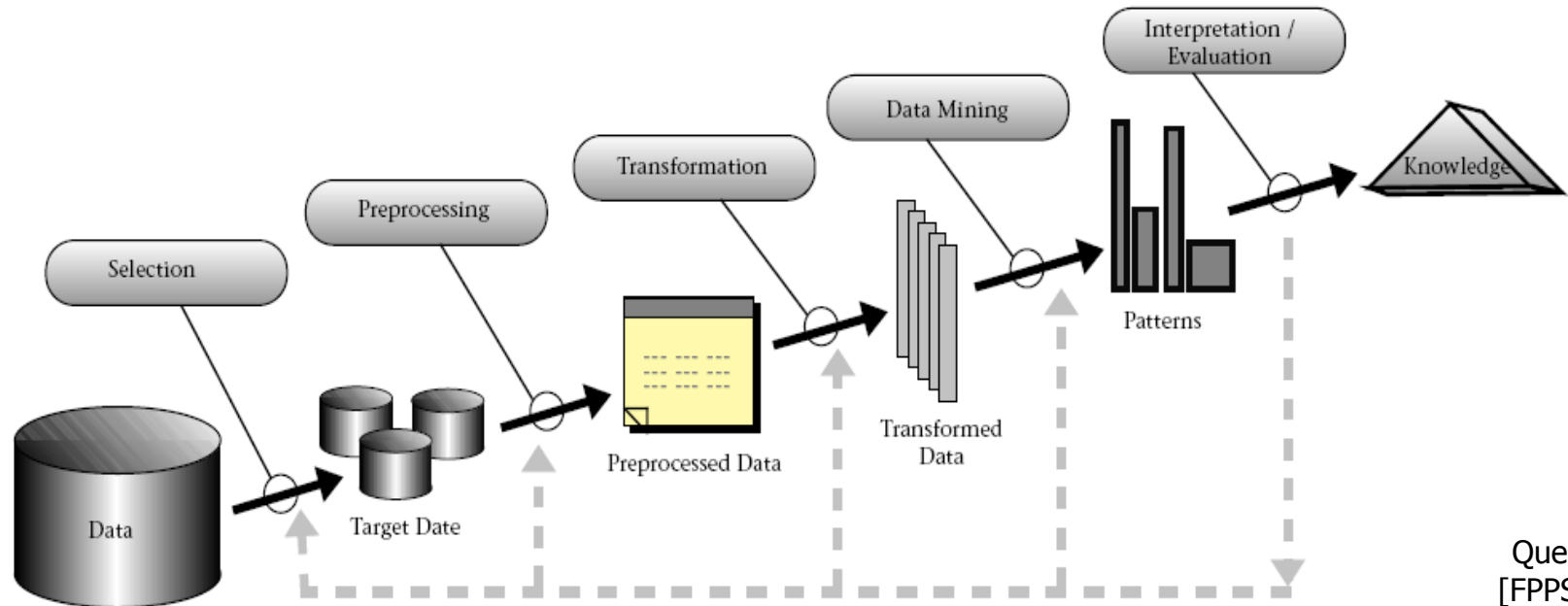
Fragestellungen

- Welche Faktoren sind am aussagekräftigsten?
- In welcher Reihenfolge bauen wir den Baum am besten?
- Was passiert mit „unseen“ Werten (Outlook=snow)?
- Müssen wir immer bis zu den Blättern gehen?

Heute: Data Mining

- Riesige **Datenberge**
 - Business: Weblogs, e-commerce, Telekoms, Transaktionen, Börsendaten, Kundendaten, ...
 - Forschung: Astronomie, Teilchenphysik, Bioinformatik, ...
 - Jeder: Nachrichten, Blogs, Webseiten, Fotos, ...
- Manuelle Analyse ausgeschlossen
 - Millionen oder **Milliarden von Datensätzen**
 - Hochdimensionale Daten mit Hunderten von Attributen
 - „**Data-Driven**“: Data Mining generiert die Hypothese und soll sie auch gleich testen
 - Vorsicht: Irgendwas findet man immer
- „We are drowning in data and starving for knowledge“
 - „Was machen Kunden eigentlich auf meiner Webseite?“

KDD als Prozess



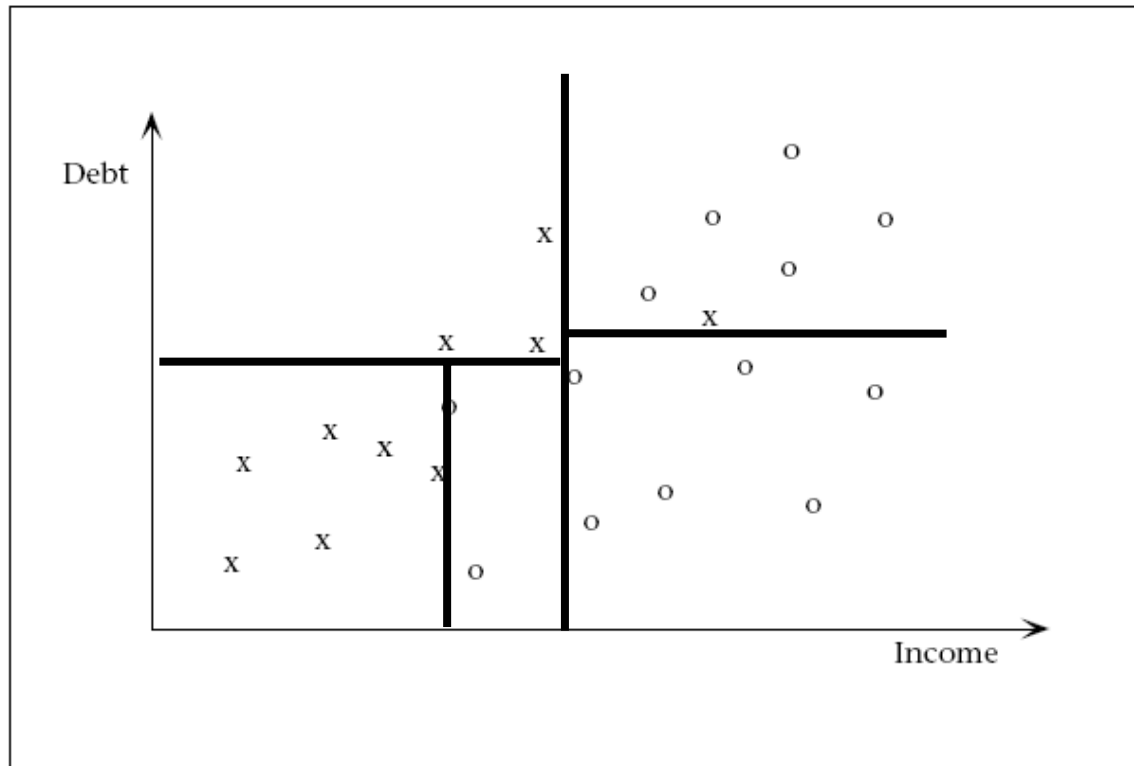
Quelle:
[FPPS96]

- Anwendung verstehen
- Daten auswählen
- Data cleaning und preprozessieren
- Datenreduktion (Dimensionen, Gruppierung, ...)
- Explorative Datenanalyse
- Data Mining
- Interpretation und Anwendung der Ergebnisse

Die wichtigsten drei Fragestellungen

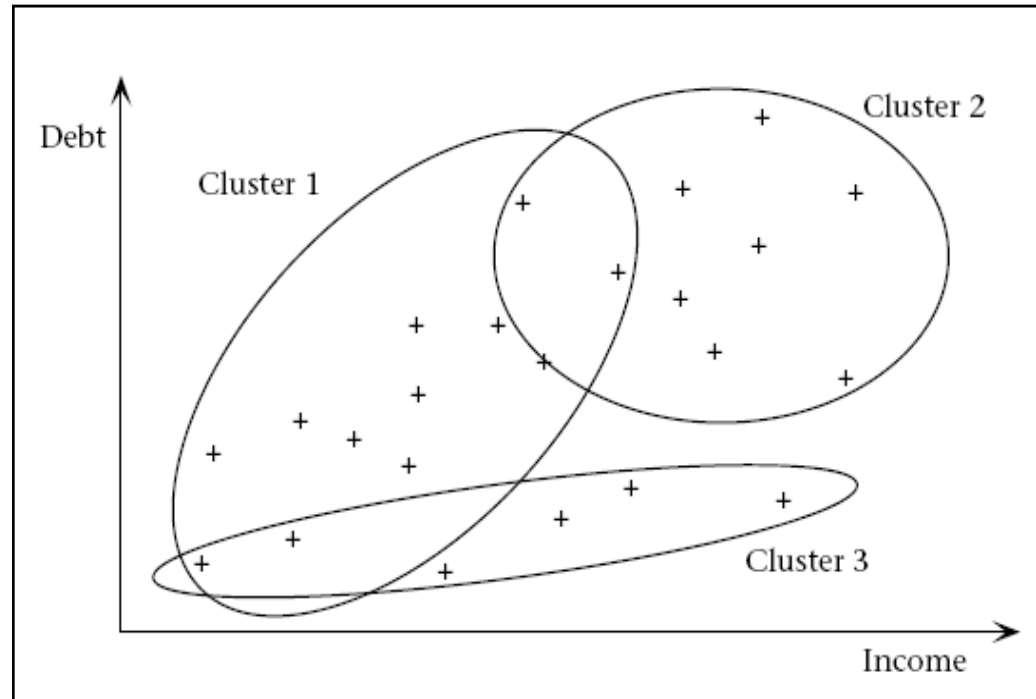
- Klassifikation
 - Gegeben eine Menge von Objekten mit Klassenzugehörigkeit (Trainingsdaten) und eine Menge von Objekten ohne (Testdaten)
 - Frage: **Welcher Klasse** gehören die unklassifizierten Objekte an?
 - Beispiel: Fraud-Detection bei Kreditkarten
- Clustering
 - Gegeben eine Menge von Objekten
 - Frage: Wie kann man diese Objekte in möglichst **homogene Cluster** einteilen?
 - Beispiel: Segmentierung von Kunden
- Assoziationsregeln
 - Gegeben eine Menge von jeweils gemeinsam durchgeführten Aktionen
 - Frage: Welche Aktionen kommen **besonders häufig zusammen** vor?
 - Beispiel: Welche Produkte werden besonders häufig gemeinsam gekauft?

Hierarchische Aufteilung



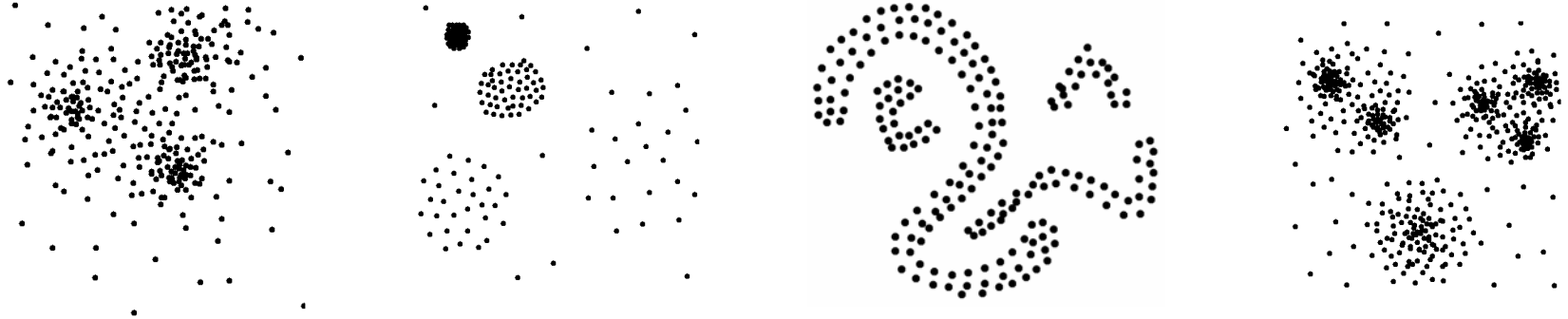
- **Verwendung lokaler Trennfunktionen**
 - Lokal für einen Subraum der Daten
- Ähnlichkeit zu mehrdimensionalen Indexen ist kein Zufall

Clustering



- Finde Gruppen zusammengehörender Objekte
- Basiert auf einem **Abstandsmaß** für Objekte
 - Zusammengehörend = „nahe“ im Raum

Nicht immer so einfach



- Problem ist schillernder und deutlich weniger gut definiert als Klassifikation
 - Wie groß sollen die Cluster ein?
 - Welche Form dürfen die Cluster haben?
 - Wie viele Cluster erwartet man?
 - Müssen alle Punkte geclustert werden?
 - Dürfen sich Cluster überlappen?

Quelle:
[ES00]

Association Rule Mining

- Welche Ereignisse (Items) geschehen häufig zusammen?

Transaction	Items
t_1	Bread, Jelly, Peanut Butter
t_2	Bread, Peanut Butter
t_3	Bread, Milk, Peanut Butter
t_4	Beer, Bread
t_5	Beer, Milk

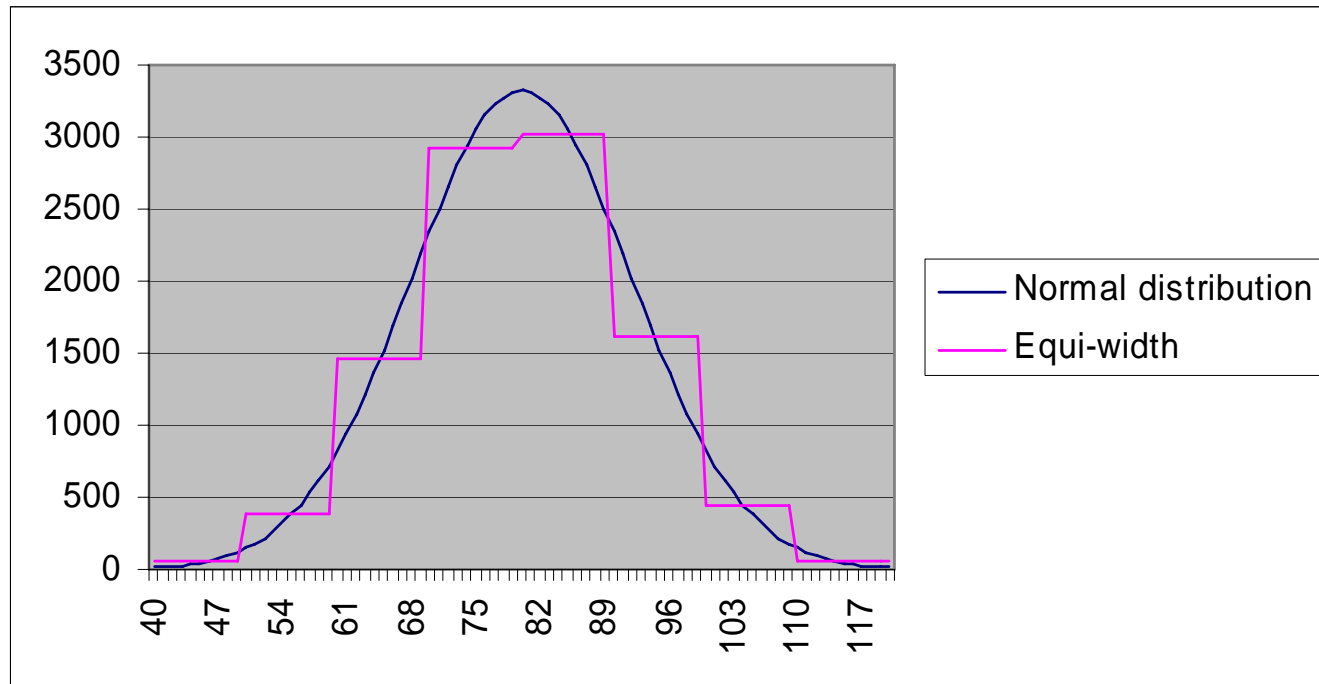
Quelle:
[Dun02]

- Problem: Es gibt so viele mögliche Itemsets!
 - Wie viele?

Datenaufbereitung

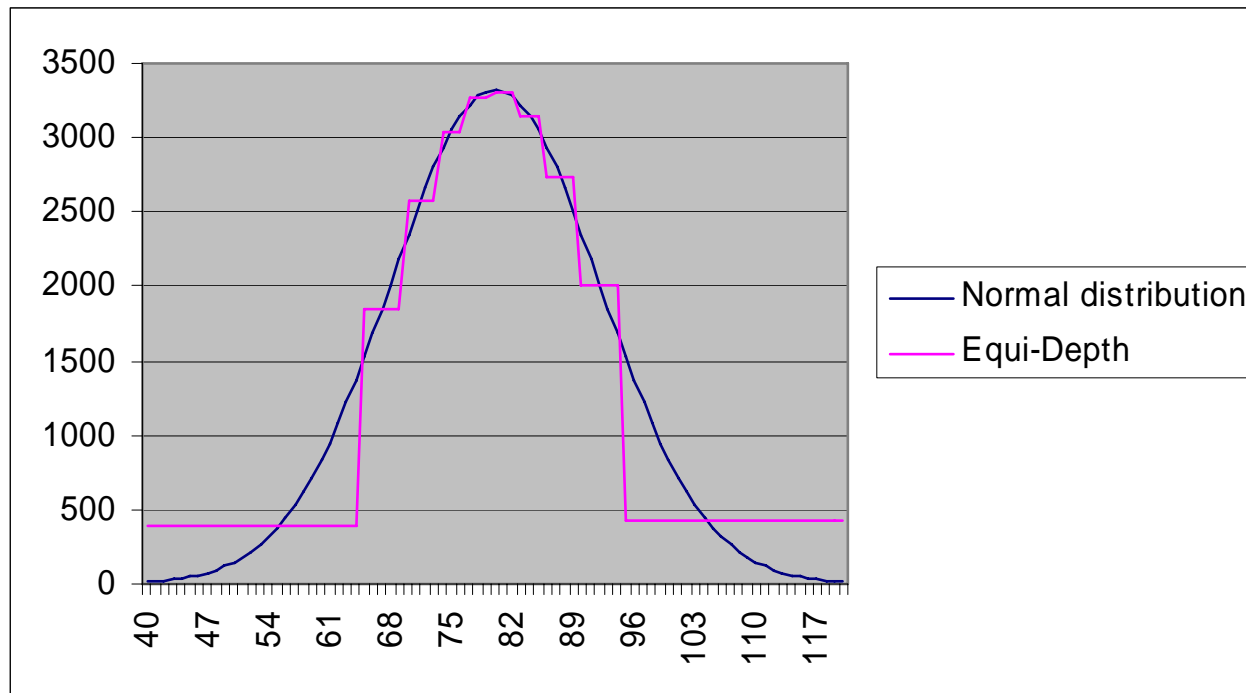
- Viele DM Verfahren reagieren empfindlich auf **Ausreißer, fehlende Werte, Datenfehler** etc.
- Preprocessing: Herstellung einer homogenen und bereinigten Datenbasis
 - Das ist unser ETL Prozess: **Erhöhung der Datenqualität**
- Beispiele
 - Ersetzung von fehlenden Werten durch Schätzen, Extrapolation
 - Aber Datenverteilung muss erhalten bleiben
 - Diskretisierung von Werten (Binning)
 - Z.B. Einteilung des Einkommens von Kunden in 5 Bereiche
 - Glättet Ausreißer, reduziert die Zahl verschiedener Werte
 - Ranking von Werten
 - Statt absoluten Einkommen wird der Rang benutzt
 - Glättet Ausreißern
 - Lässt aber auch ev. wichtige Unterschiede verschwinden

Equi-Width Histograms



- Zahl der Bins festlegen und **Raum äquidistant aufteilen**
- Bins enthalten unterschiedlich viele Werte und decken den ganzen Raum gleichmäßig ab
- Berechnung durch Sortierung und einen Scan

Equi-Depth



- Zahl der Bins festlegen, dann Raum so aufteilen, dass alle **Bins gleich viele Tupel** enthalten
- Führt zu gleichgroßen Bins mit unterschiedlicher Breite

ODM Algorithmen

- Klassifikationsalgorithmen
 - Decision Tree
 - Naive Bayes und Adaptive Bayes Networks
 - SVM mit linearen Kernel
- Regression
- Clustering
 - K-Means
 - Euklidischer- oder Cosinus-Abstand
 - Cluster werden durch Centroide repräsentiert
 - Orthogonal Partitioning Clustering
 - Hierarchisches Clustering mit achsenparallelen Schnitten
- Association Rule Mining
 - A-Priori Algorithmus mit Min-Support/Confidence Filtering
- Text Mining
 - Clustering mit k-Means
 - Klassifikation mit SVM

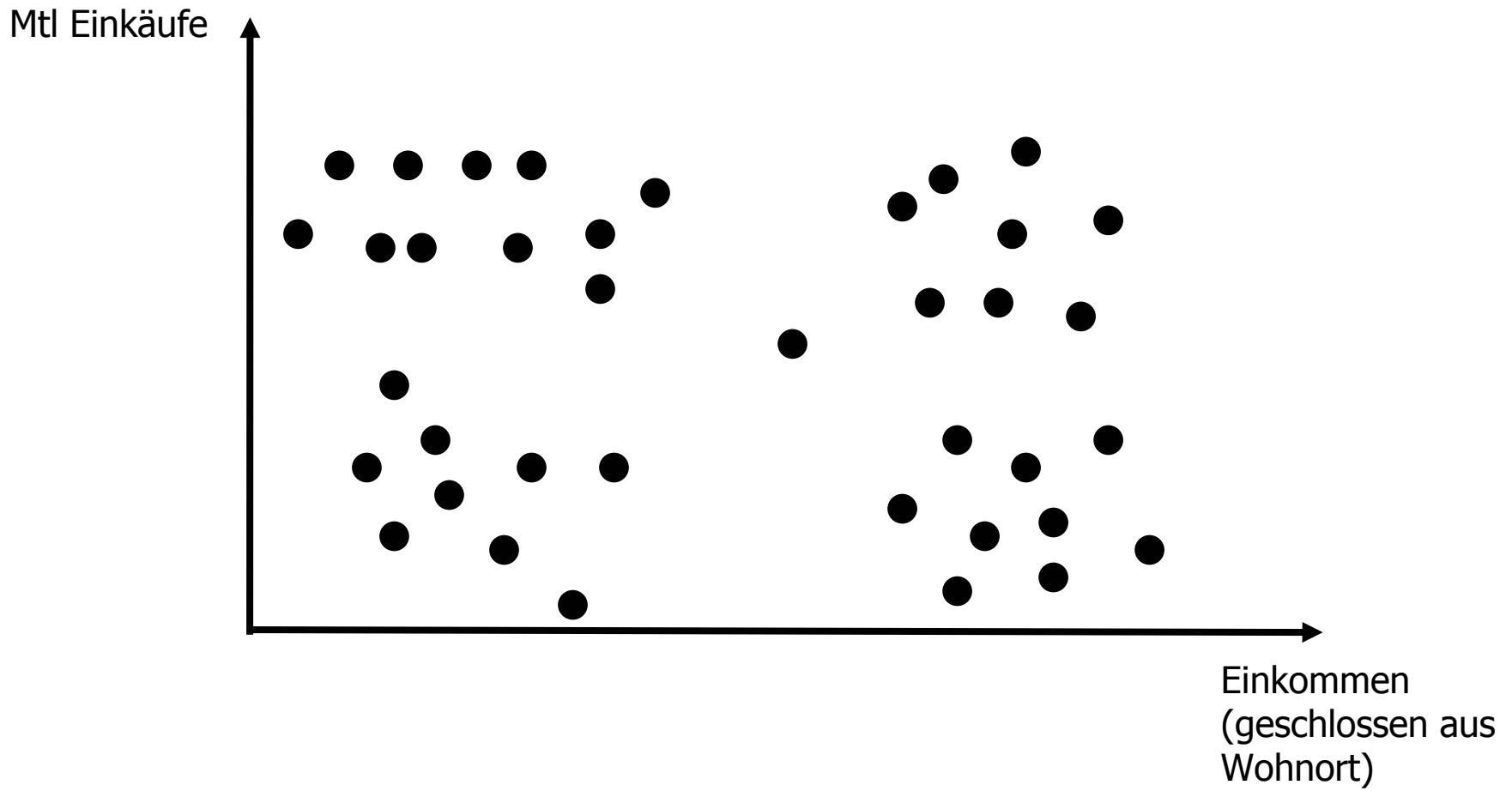
Inhalt dieser Vorlesung

- Einführung
 - Clustergüte
 - Ähnlichkeiten
 - Clustermitte
- Hierarchisches Clustering
- Partitionierendes Clustering
- Dichte-basiertes Clustering

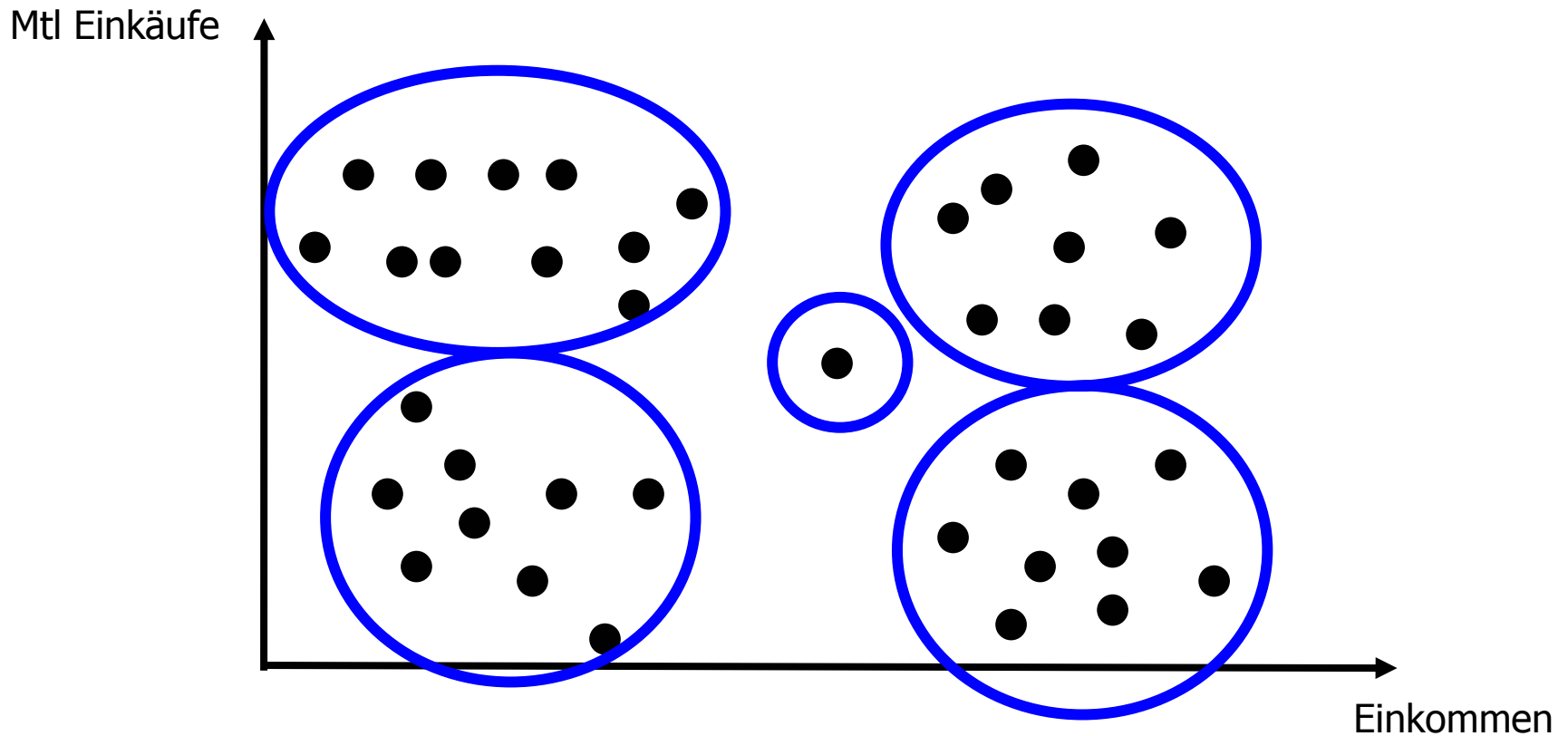
Clustering

- Finde **Gruppen ähnlicher Objekte**
 - Ohne zu wissen wie viele Gruppen es geben soll
 - „**Unsupervised learning**“
 - (Mit der Gruppierung haben wir meist auch eine Klassifikation)
- **Anwendungen**
 - Segmentiere Kunden in Gruppen
 - Clustere Patienten in verschiedene Therapieanleitungen
 - Finde Typen von Sternen in astronomischen Karten
 - Welche Cluster von Genen reagieren ähnlich auf ein bestimmtes Medikament?
 - Welche Ergebnisse einer Websuche kommen aus dem selben Thema(encluster)
 - ...

Beispiel 1

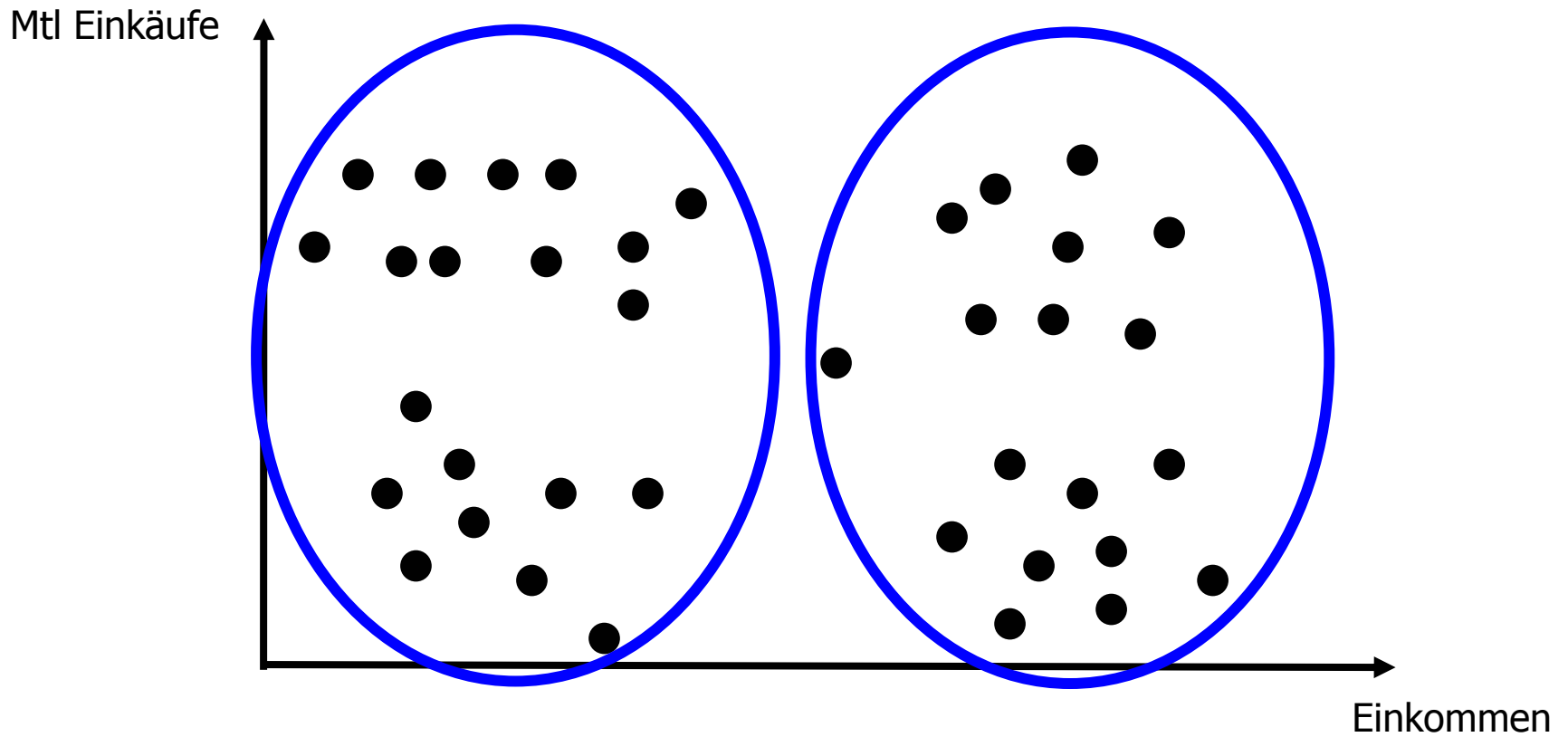


Beispiel 1



- Vier Cluster und ein **Ausreißer**(-Cluster)
- Überlappungsfreie, konvexe Cluster

Beispiel 2



- Zwei Cluster
- Besser?

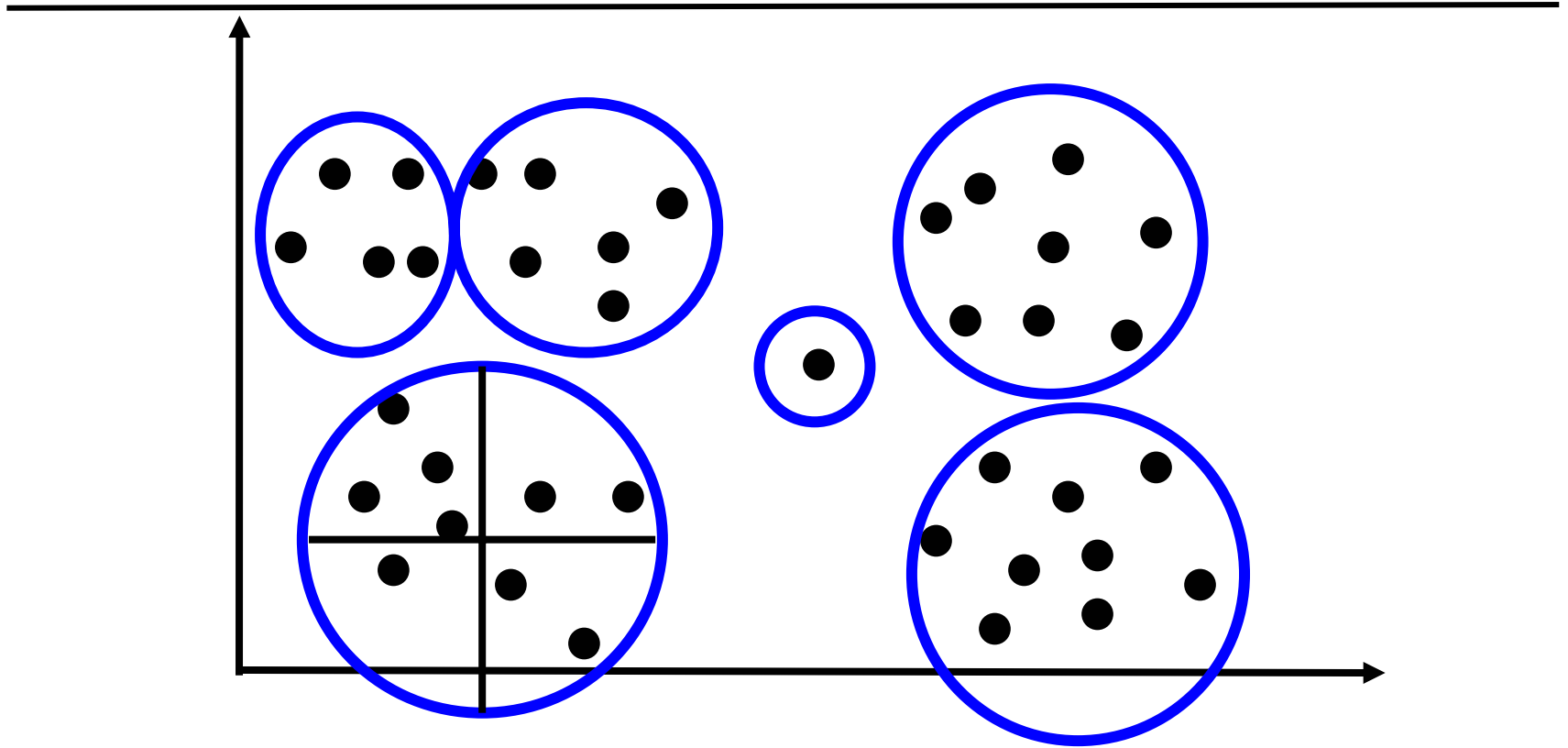
Güte eines Clusterings

- Intuitiv ist eine Gruppierung gut, wenn innerhalb **jedes Clusters alle Punkte nahe beieinander** liegen
- Definition
*Sei f eine Klassenzuordnung für Objekte O zu einer Menge von Klassen C mit $|C|=k$. Sei m_c der Mittelpunkt aller Objekte der Klasse $c \in C$, und sei $d(o, o')$ der Abstand zwischen zwei beliebigen Punkten. Dann ist die **k -Güte von f** :*

$$q_k(f) = \sum_{c \in C} \sum_{f(o)=c} d(o, m_c)$$

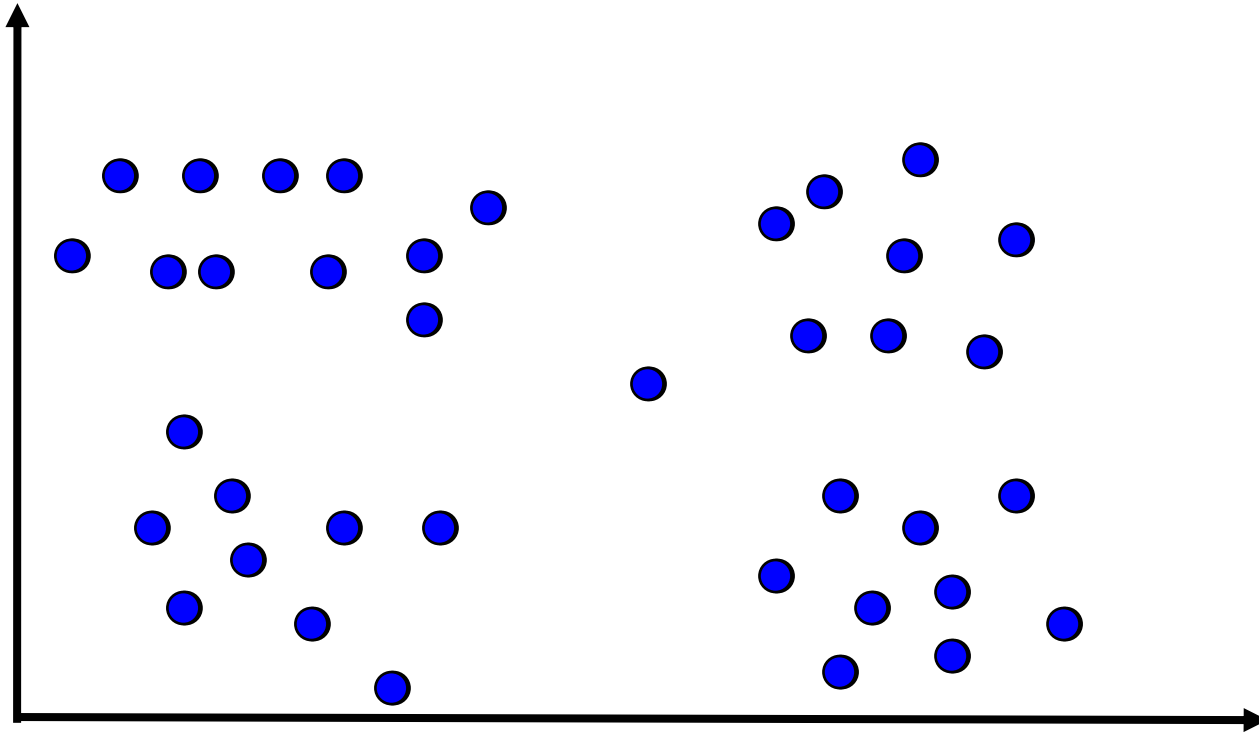
- Bemerkung
 - Zur Bestimmung von Mittelpunkten kommen wir gleich
 - Auch die **Einschränkung auf k -Güte** erklärt sich gleich

6-Güte



- In jedem Cluster Mittelpunkte bestimmen
- **Abstand aller Punkte** im Cluster zum Mittelpunkt summieren
- Summe über alle Cluster bilden

Nachteil



- Optimales Clustering ohne Einschränkung auf k ?
 - Trivial mit $k=|O|$

Güte bei fester Anzahl von Clustern

- k-Güte ist als Maß nur dann sinnvoll, wenn die **Anzahl an Clustern vorab feststeht**
- Nur damit: Clustering als Optimierung
 - Finde für eine Menge O von Objekten eine Zuordnung f in k Cluster so, dass $q_k(f)$ minimal ist
- Varianten
 - Festlegung von k ist sehr sensitiv bei Ausreißern
 - Ausreißer vorab löschen (aber wie findet man die? Clustering!)
 - Cluster werden immer **kreisförmig** sein
 - Oft verlangt man, dass die Intra-Cluster Ähnlichkeit hoch und gleichzeitig die **Inter-Cluster Ähnlichkeit gering** ist
 - Formel ändern: Maximiere die Summe von $q_k(f)$ minus die Summe über den durchschnittlichen Abstand jedes Punkts zu allen Clusterzentren außer seinem eigenen

Clustergüte zweiter Versuch

- Definition

*Sei $f: O \rightarrow C$ mit $|C|$ beliebig. Sei $\text{dist}(o, C_i)$ der **mittlere Abstand** von o zu allen Punkten eines Clusters C_i .*

Damit definieren wir

- $a(o) = \text{dist}(o, f(o))$

- $b(o) = \min(\text{dist}(o, C_i))$ mit $C_i \neq f(o)$

- Die **Silhouette eines Punktes** $s(o)$ ist $s(o) = \frac{b(o) - a(o)}{\max(a(o), b(o))}$

- Die **Silhouette von f** ist $\sum s(o)$

- Bemerkung

- $b(o)$: Verschlechterung, wenn es Cluster $f(o)$ nicht gäbe (und o dem nächsten Cluster zugeordnet werden würde)

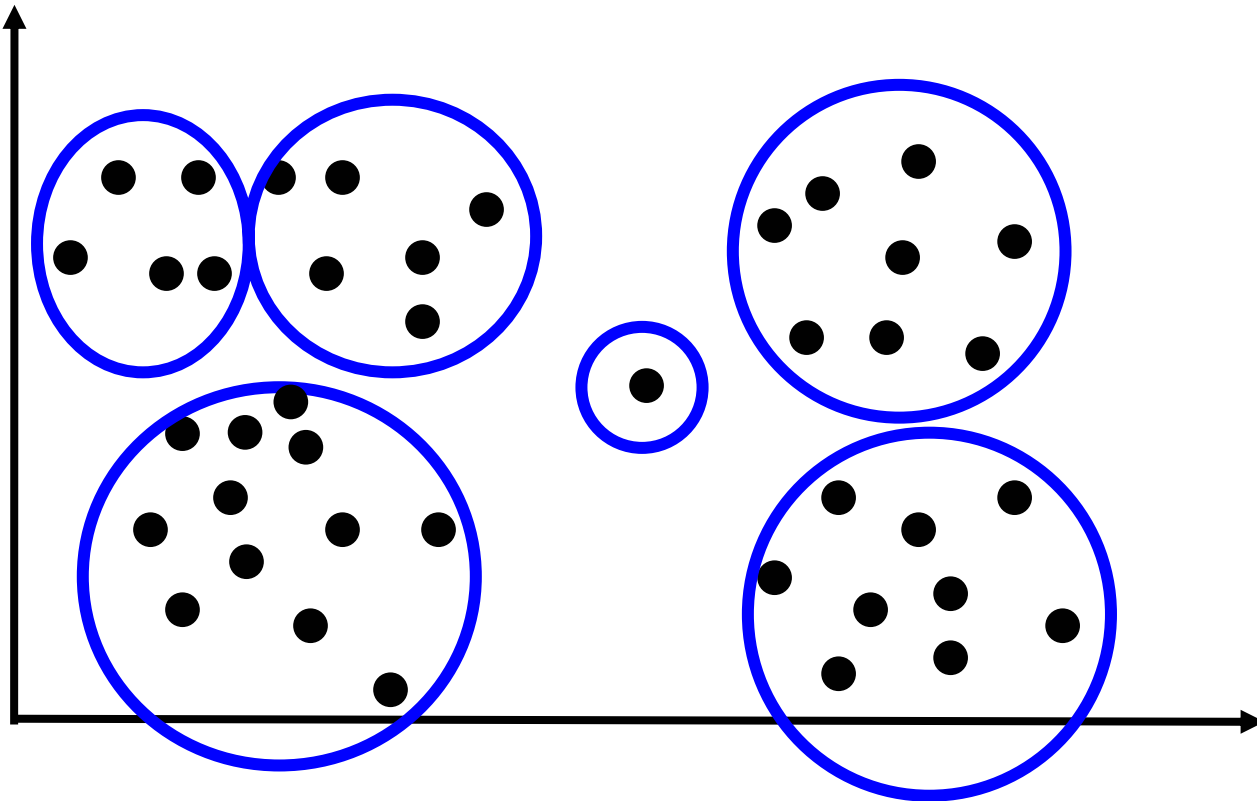
- Das heißt

- $s(o) \approx 0$: Punkt liegt zwischen zwei Clustern

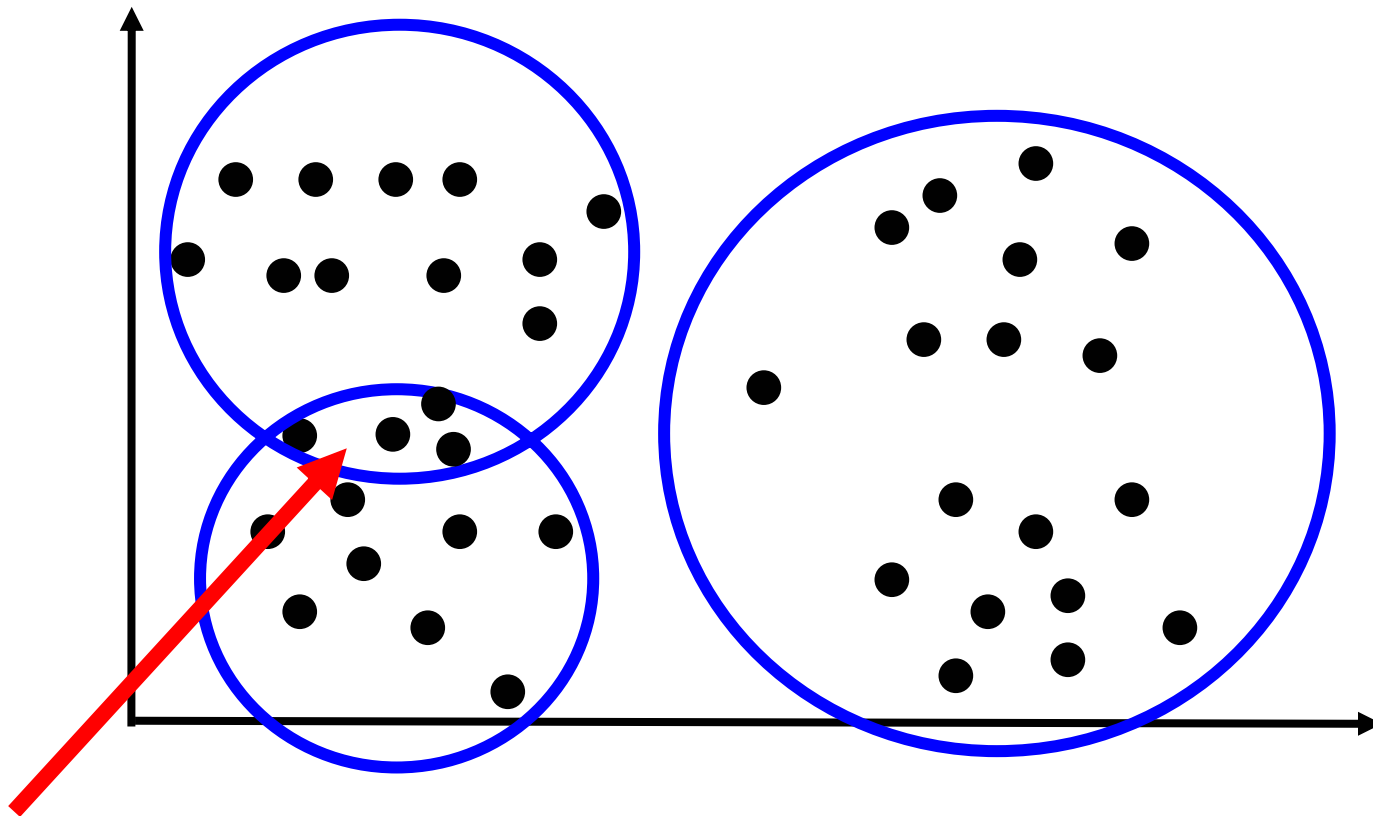
- $s(o) \rightarrow 1$: Punkt liegt näher am eigenen als am nächsten Cluster

- $s(o) \rightarrow -1$: Punkt liegt näher am nächsten Cluster als am eigenen

Silhouette gut



Silhouette schlecht



Zu welchem Cluster sollen diese Punkte gehören?

Ähnlichkeit

- Alle Verfahren müssen die **Ähnlichkeit** oder Nähe zweier (m-dimensionaler) Objekte berechnen können
 - Es gibt eine unendliche Menge von Ähnlichkeitsfunktionen
- Numerische Werte
 - **Euklidischer Abstand**
 - Betont große Abstände in einzelnen Dimensionen sehr stark
 - Standard für metrische Werte
 - Cosinus-Abstand: Differenz der Winkel der Featurevektoren zum Nullpunkt
 - Ausreißer in einzelnen Dimensionen zählen weniger
 - Standard z.B. beim Text-Mining
- Kategorielle Werte
 - Anwendungsabhängiges Maß notwendig
- Mengenwertige Werte
 - Z.B. Schlagwörter zu einem Dokument
 - Beispiel: **Anzahl gleicher Werte** normiert auf die Vereinigung der Wertemengen

Die Mitte eines Clusters

- Was ist der Mittelpunkt eines Clusters?
- Numerische Werte
 - Meistens: Der **Mittelwert** aller Featurevektoren der Punkte des Clusters
 - Centroid
 - Manchmal: Der **Median** aller Featurevektoren der Punkte des Clusters
 - **Medoid**: Der „mittlerste“ Punkt von C , der auch in O ist
 - Nachteil: **Berechnung eines Medoids ist teuer**
 - Vorteil: Weniger sensitiv bei Ausreißern
- Kategoriale Werte
 - Mittelwerte sind i.A. nicht definiert
 - Ein Abstandsmaß braucht man natürlich trotzdem
 - Also muss man den Mediod nehmen

Inhalt dieser Vorlesung

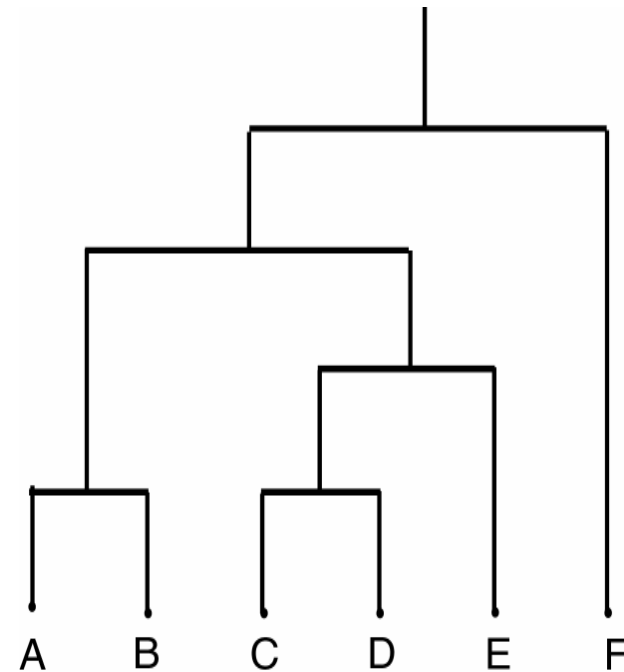
- Einführung
- Hierarchisches Clustering
- Partitionierendes Clustering
- Dichte-basiertes Clustering

Clusteralgorithmen

- **Hierarchische Verfahren:** hierarchisch geschachtelte Cluster
 - Bottom-Up: Ausgehend von $|C|=|O|$ werden sukzessive Cluster verschmolzen bis nur noch ein Cluster da ist
 - Top-Down: Ausgehend von $C_0=O$ werden sukzessive Cluster geteilt bis jedes Objekt sein eigener Cluster ist
 - Benötigen kein k , aber sind eher teuer
 - Berechnen nicht unmittelbar Cluster (weiterer Schritt notwendig)
- **Partitionierende Verfahren:** Zerlegung der Punktmenge in k Cluster
 - Benötigen die Anzahl k der Cluster als Parameter
- **Dichte-basierte Verfahren**
 - Suchen nach Subräumen mit hoher Punktdichte und fassen die zu Clustern zusammen
 - Tendenziell langsam
 - Können sehr gut durch multidimensionale Indexstrukturen unterstützt werden

Hierarchisches Clustering

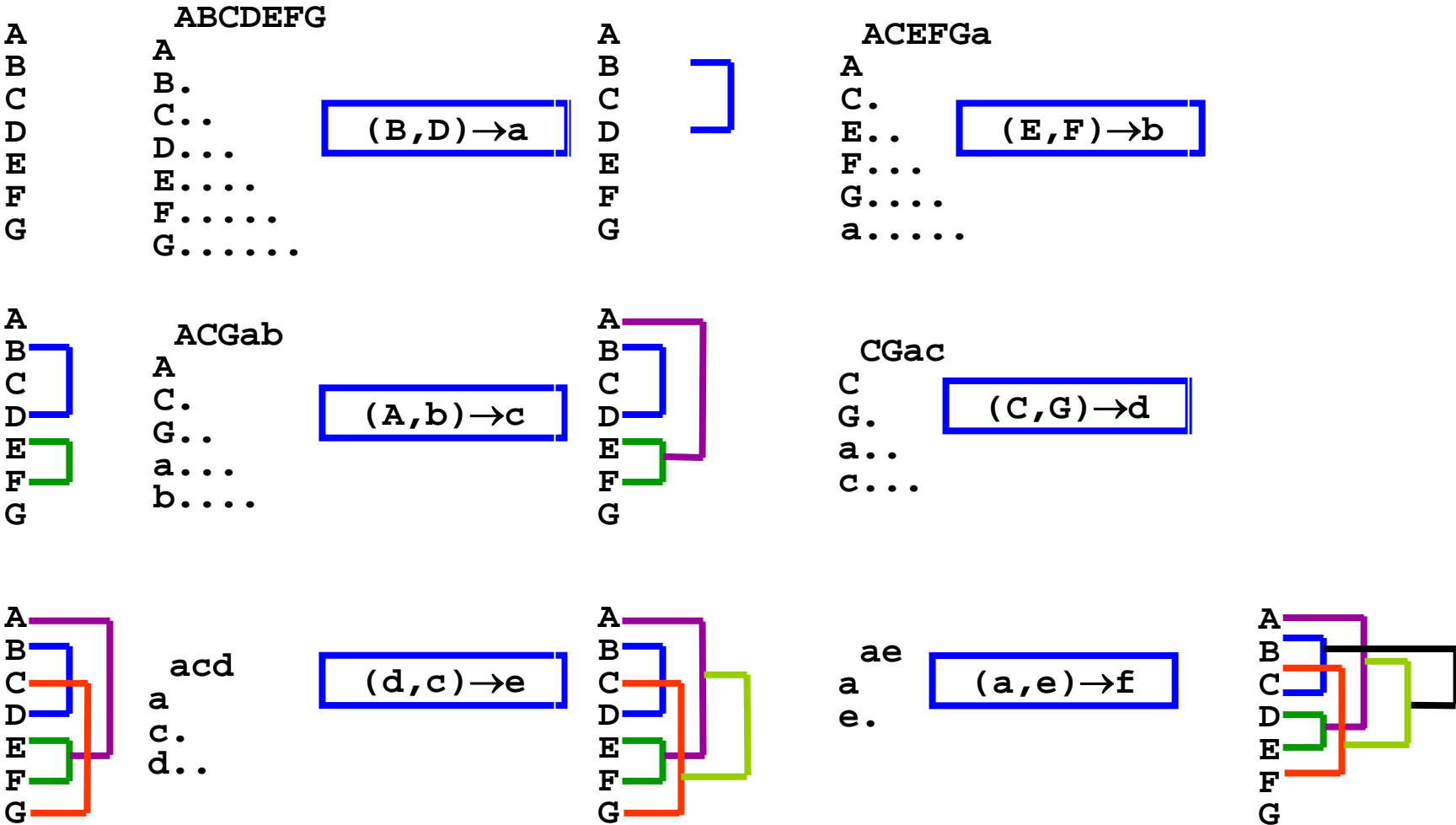
- Auch: UPGMA
 - Unweighted Pair-group method with arithmetic mean
- Bottom-Up Berechnung eines binären Baums (**Dendogramm**)
- Algorithmus
 - Berechne **Abstandsmatrix M**
 - Ähnlichkeit aller Objekte zu allen anderen Objekten
 - Wähle Paar o_1, o_2 mit dem kleinsten Abstand
 - Berechne $x = m(o_1, o_2)$ (den Mittelpunkt)
 - Lösche o_1, o_2 aus M; füge x ein
 - Berechne Abstand von x zu allen verbleibenden Objekten/Clustern in M
 - Als Mittelwert der Abstände von o_1 und o_2
 - Iteriere, bis M leer ist



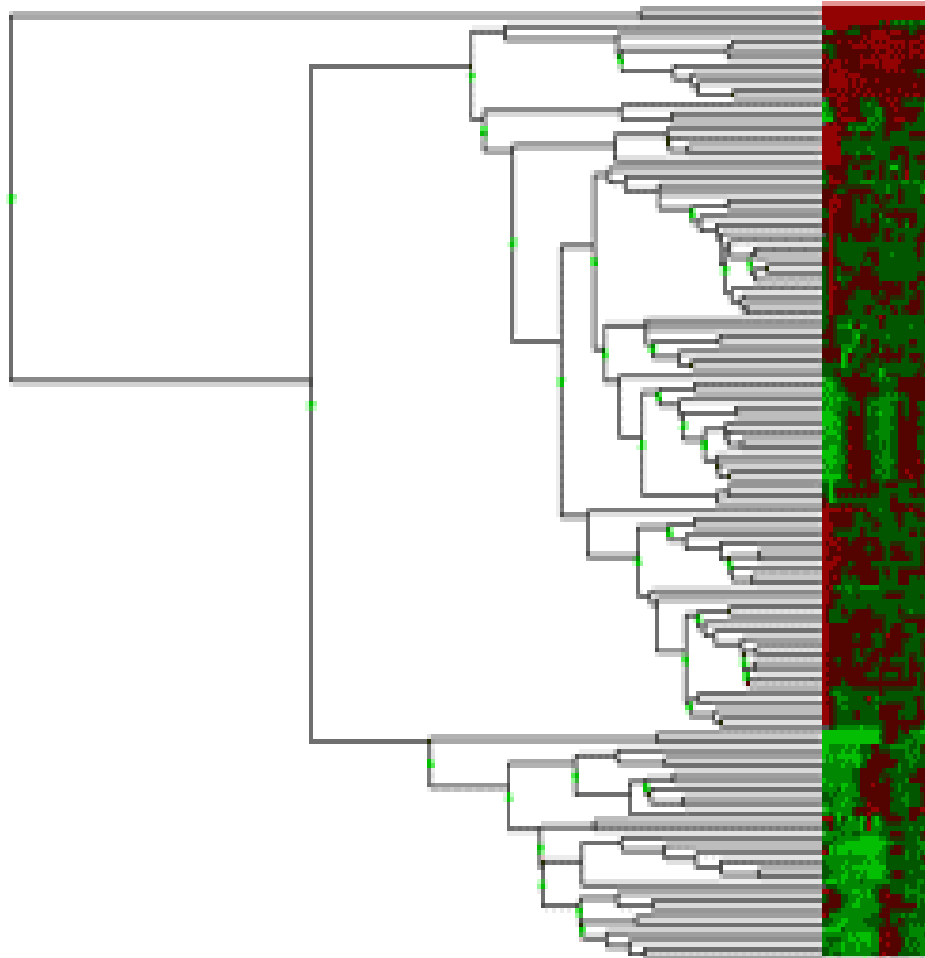
Beispiel: Ähnlichkeitsmatrix

	A	B	C	D	E	F	..
A		90	84	91	81	82	..
B			43	99	37	36	..
C				14	35	21	87
D					28	34	..
E						95	..
F							..
..	

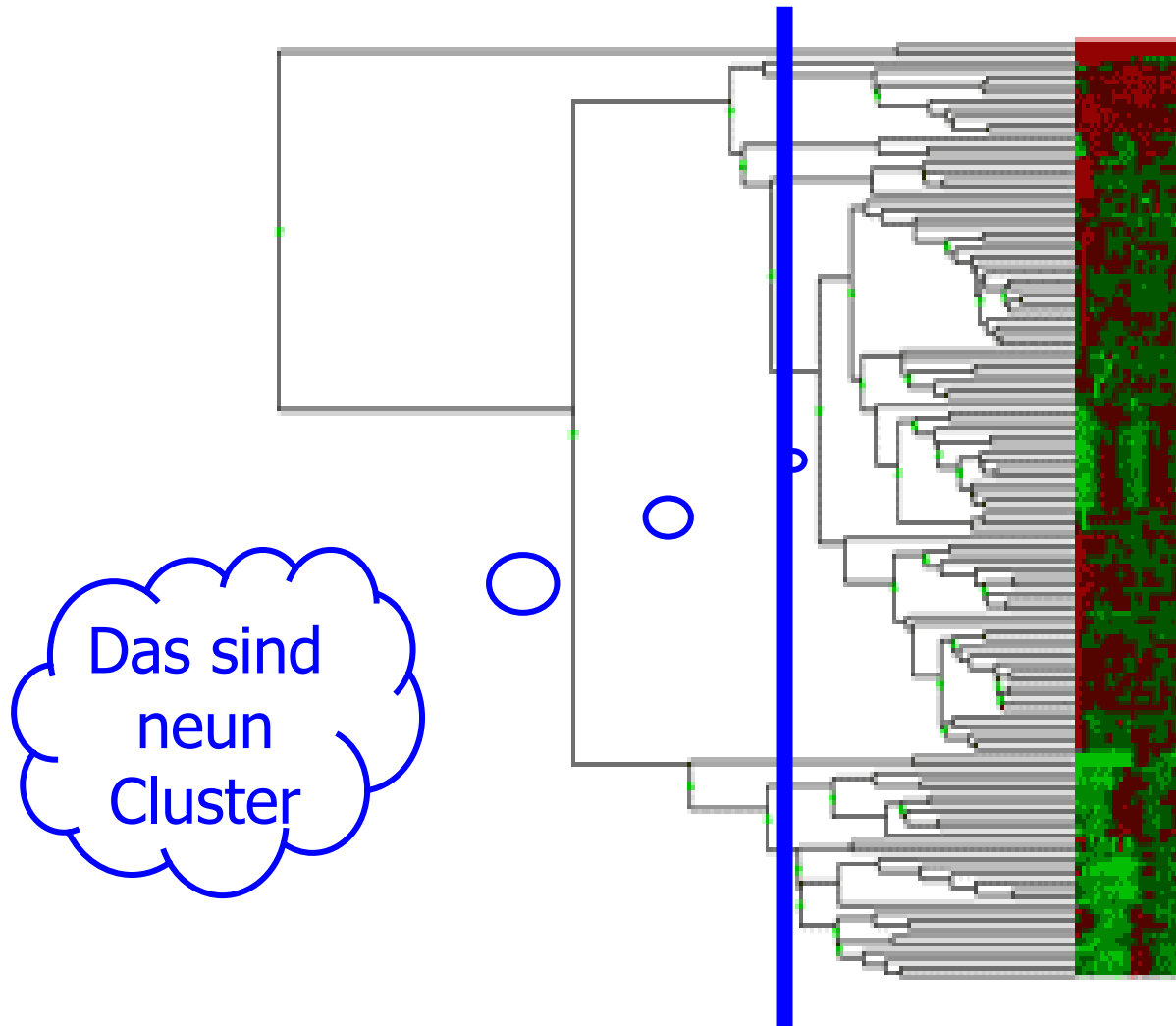
Iteration



Bespiel Genexpressionsdaten

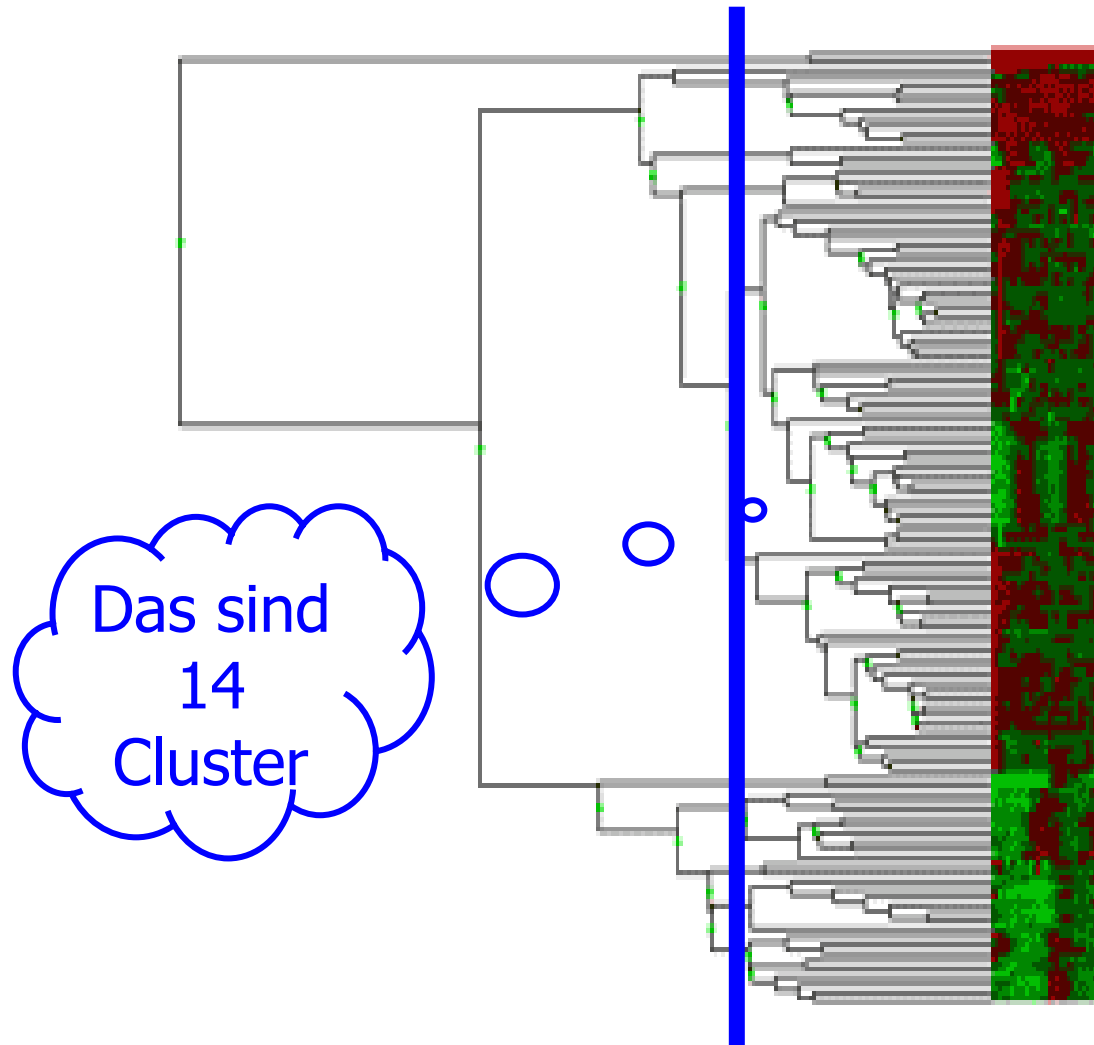


Festlegung von Clustern



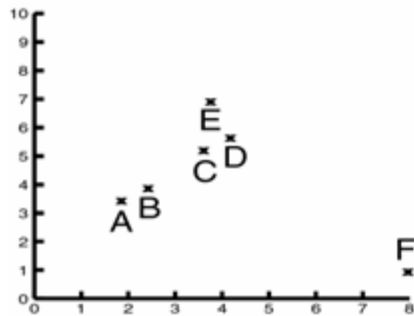
Das sind
neun
Cluster

Alternative

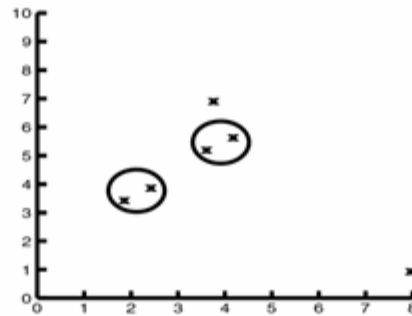


Das sind
14
Cluster

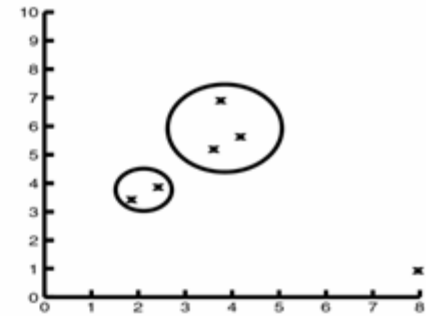
Geschachtelte Cluster



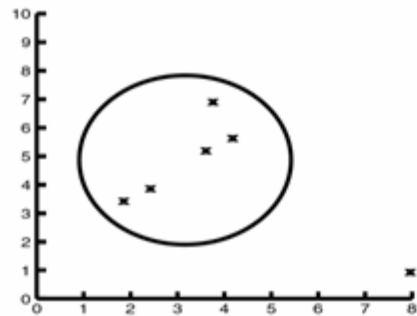
a) Six Clusters



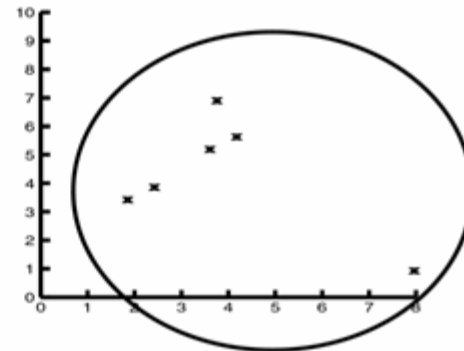
b) Four Clusters



c) Three Clusters



d) Two Clusters



e) One Cluster

Eigenschaften

- Vorteile
 - Konzeptionell **einfach**, hübsche, irgendwie intuitive Grafiken
 - Keine Festlegung auf Anzahl Cluster notwendig
- Nachteile
 - Benötigt die **Abstandsmatrix** als Eingabe
 - $|O|=n$: $O(n^2)$ Platz und $O(n^2)$ Zeit
 - Plus Zeit zur Berechnung der Cluster
 - Berechnet keine Cluster
 - Weiterer Schritt (wie?) notwendig
- Fazit
 - Kaum anwendbar für viele (>10000) Objekte
 - Eher ein Umweg als Clustering

SQL - Distanzmatrix

- Annahmen

- Alle Objekte und ihre Attribute a, b, \dots stehen in einer Tabelle `objects`
 - Z.B: $(o_1, x_1, y_1, z_1), (o_2, x_2, y_2, z_2), (o_3, x_3, y_3, z_3), \dots$
- Numerische Attribute
- Euklidischer Abstand

- Berechnung der Distanzmatrix M ?

```
SELECT  t1.oid, t2.oid,  
        sqrt(sqr(t1.a-t2.a)+sqr(t1.b-t2.b)+...)   
FROM    objects t1, objects t2  
WHERE   t1.oid>t2.oid;
```

SQL - Iteration

- Distanzmatrix materialisieren (teuer!)
 - Tabelle `distance`
- Wir erweitern `objects` um ein Flag `c`, das anzeigt, ob ein Tuple ein Objekt oder ein Cluster ist
- Iteratives Berechnung
 - Geht nicht mit einer Query
 - PL-SQL Programm mit $n=|O|$ Durchläufen
 - Mit Index auf Abstandspalte in `distance` ein $\log(n)$ Zugriff aus kleinstes Paar `P`
 - Löschen aller Einträge des Paares in `distance`
 - Berechne Mittelwert von `P` und speichere mit `c=true` und neuer, bisher größter OID in `objects`
 - Berechne neue Abstände und Einfügen in `distance`

Neue Abstände

```

INSERT INTO distance
SELECT t1.oid, t2.oid, dist(t1,t2)
FROM   objects t1, objects t2
WHERE  t1.oid = $newid AND
       t2.oid in (SELECT DISTINCT oid1
                  FROM distance);
    
```

Nur für die brauchen wir noch neue Abstände

Beispiel

	1	2	3	4	5	6	7
2							
3							
4							
...							

Distanzmatrix

o1	o2	d
2	1	?
3	1	?
3	2	?
4	1	?
4	2	?
4	3	?
5	1	?
5	2	?
...

Distanz-
tabelle

o1	o2	d
2	1	?
3	1	?
3	2	?
4	1	?
4	2	?
4	3	?
5	1	?
5	2	?
...

Z.B: (2,3) als
kleinstes
Element gelöscht

o1	o2	d
4	1	?
5	1	?
5	4	?
6	1	?
6	4	?
6	5	?
7	1	?
...

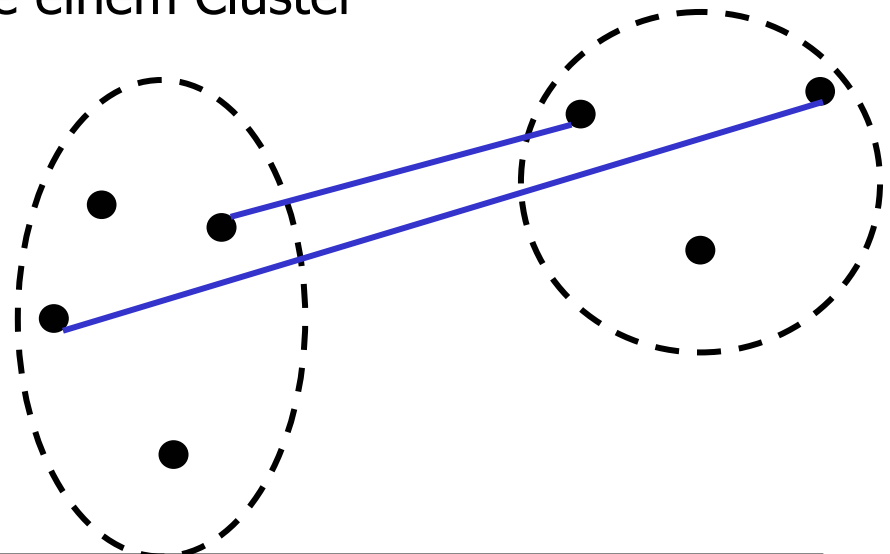
o1	o2	d
4	1	?
5	1	?
5	4	?
6	1	?
6	4	?
6	5	?
7	1	?
...
8	1	?
8	4	?
...

Neuer Knoten 8
eingefügt

Kleinstes
Element;
Spezial-
behandlung
notwendig

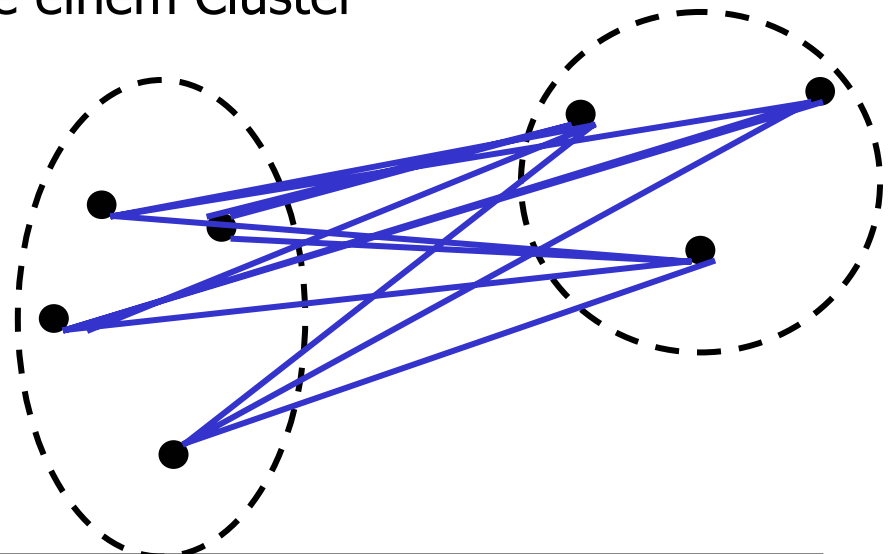
Varianten: Abstandsmaße

- Beim hierarchischen Clustern berechnen wir oft den **Abstand zweier Cluster** bzw. eines Punktes zu einem Cluster
- Bisher: Clustermittelpunkt verwenden
- Viele Alternativen
 - **Single Link**: Minimum aller Abstände zwischen zwei Objekten aus je einem Cluster
 - **Complete Link**: Maximum aller Abstände ...
 - **Average Link**: Durchschnittlicher Abstand ...
 - **Centroid**: Abstand der Mittelpunkte



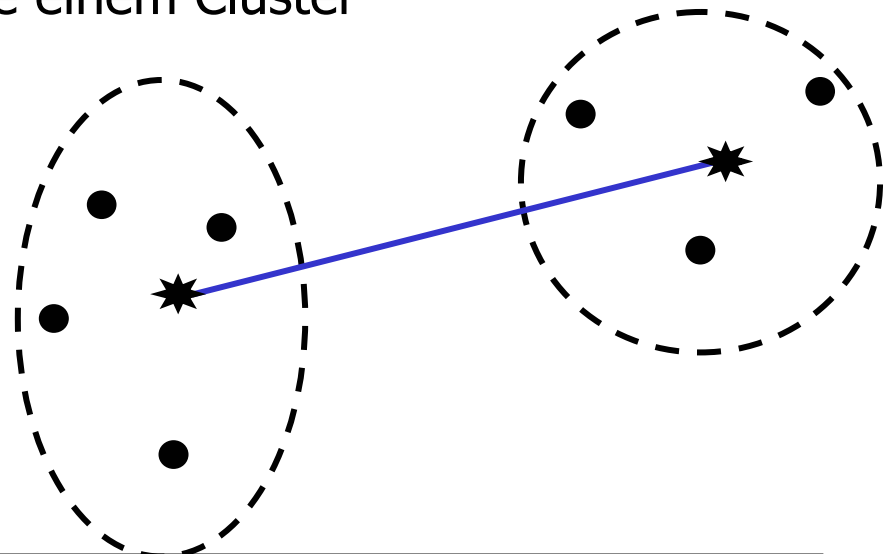
Varianten: Abstandsmaße

- Beim hierarchischen Clustern berechnen wir oft den Abstand zweier Cluster bzw. eines Punktes zu einem Cluster
- Bisher: Clustermittelpunkt verwenden
- Viele Alternativen
 - Single Link: Minimum aller Abstände zwischen zwei Objekten aus je einem Cluster
 - Complete Link: Maximum aller Abstände ...
 - **Average Link:** Durchschnittlicher Abstand ...
 - Centroid: Abstand der Mittelpunkte



Varianten: Abstandsmaße

- Beim hierarchischen Clustern berechnen wir oft den Abstand zweier Cluster bzw. eines Punktes zu einem Cluster
- Bisher: Clustermittelpunkt verwenden
- Viele Alternativen
 - Single Link: Minimum aller Abstände zwischen zwei Objekten aus je einem Cluster
 - Complete Link: Maximum aller Abstände ...
 - Average Link: Durchschnittlicher Abstand ...
 - **Centroid**: Abstand der Mittelpunkte



Inhalt dieser Vorlesung

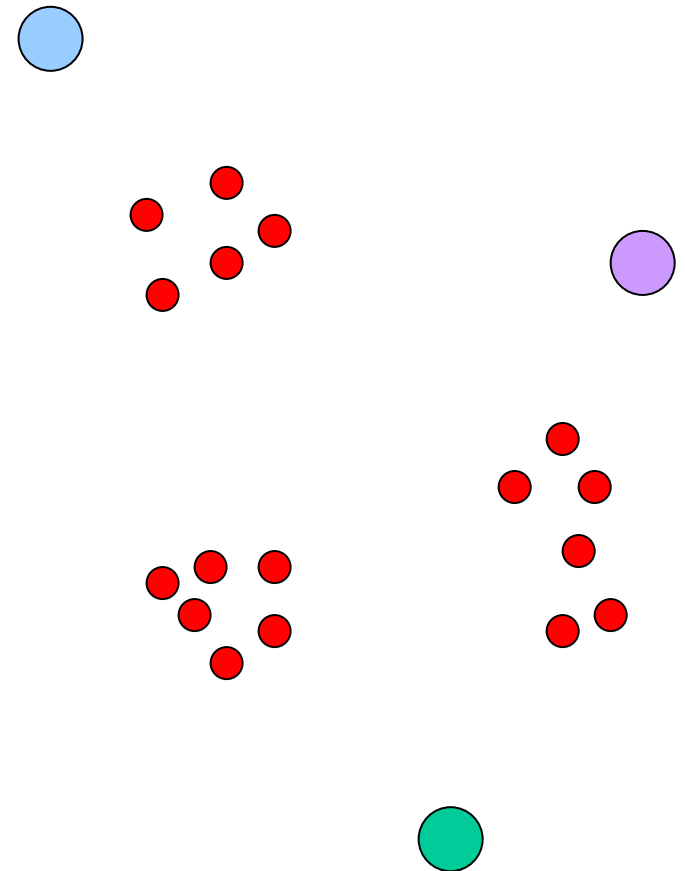
- Einführung
- Hierarchisches Clustering
- Partitionierendes Clustering
 - k-Means
 - k-Medoid und CLARANS
 - Sampling: BIRCH
- Dichte-basiertes Clustering

K-Means

- K-Means ist wahrscheinlich die bekannteste Clusteringmethode
 - Mit sehr vielen Varianten
- Benötigt die **Anzahl k von Clustern als Parameter**
- Berechnet ein lokales Optimum bezüglich k -Güte
 - Keine Garantien für globale Optimalität bzgl. k -Güte
- Algorithmus
 - Wähle zufällig k Clustermittelpunkte
 - Iteriere
 - Für alle Objekte
 - Berechne Abstand jedes Objekts zu jedem Clustermittelpunkt
 - Weise Objekt seinem **nächsten Clustermittelpunkt** zu
 - Wenn sich keine Objektzuordnung mehr geändert hat, dann STOP
 - Für alle Cluster
 - Berechne **neues Clusterzentrum**

Beispiel 1

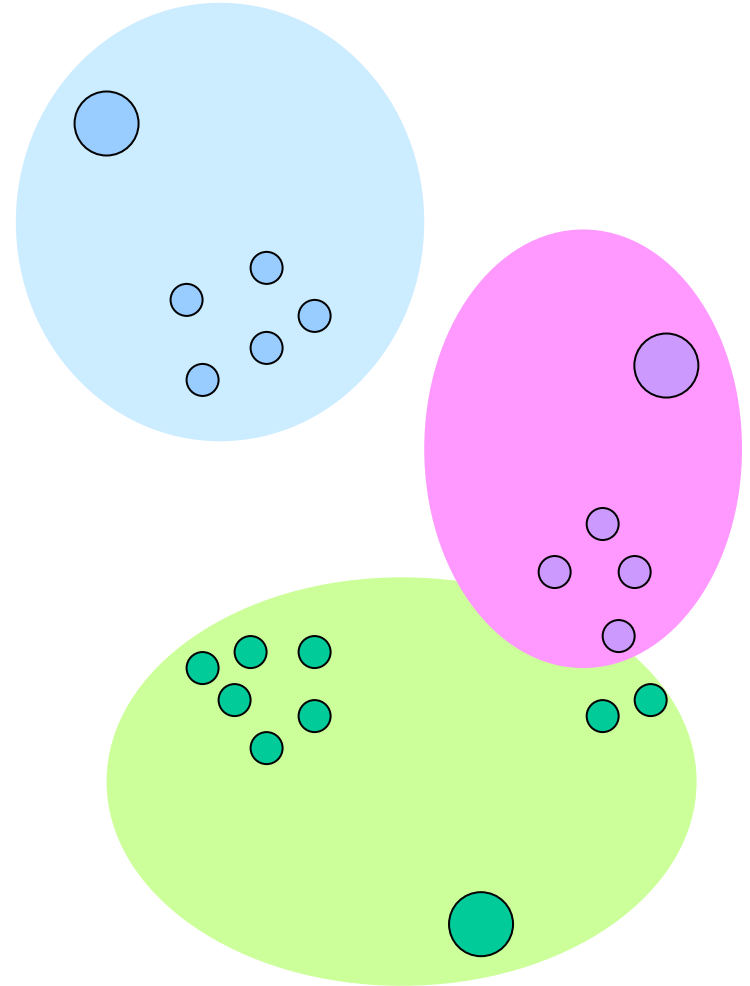
- $k=3$ zufällige Startwerte auswählen



Quelle: Stanford, CS 262
Computational Genomics

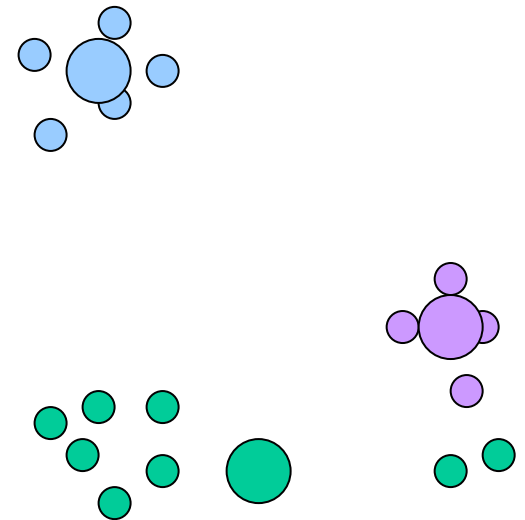
Beispiel 2

- Objekte dem nächsten Clusterzentrum zuordnen



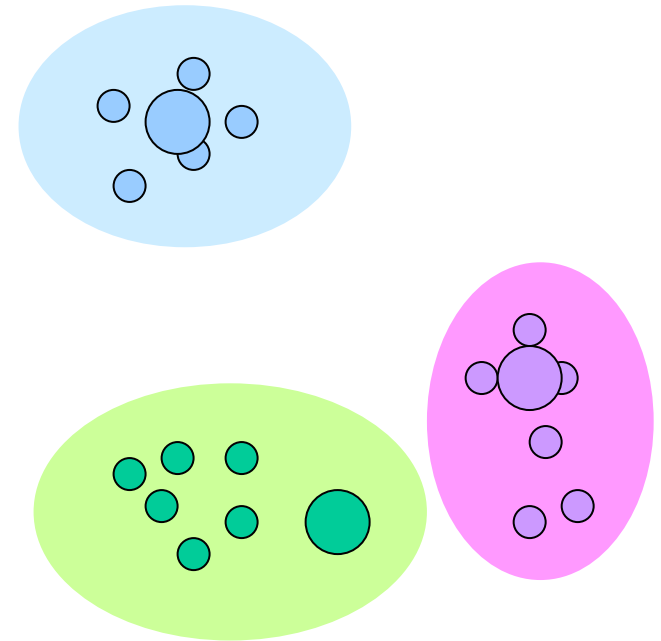
Beispiel 3

- Clustermittelpunkte neu berechnen



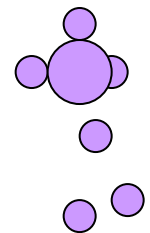
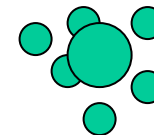
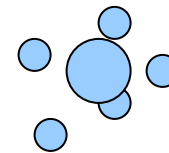
Beispiel 4

- Objekte neu zuordnen



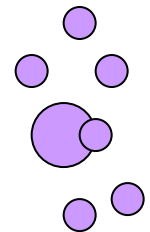
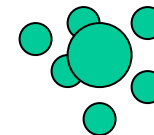
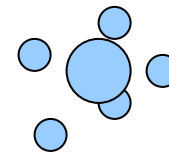
Beispiel 5

- Mittelpunkte anpassen



Beispiel 6

- Fertig, keine neuen Zuordnungen mehr



Varianten

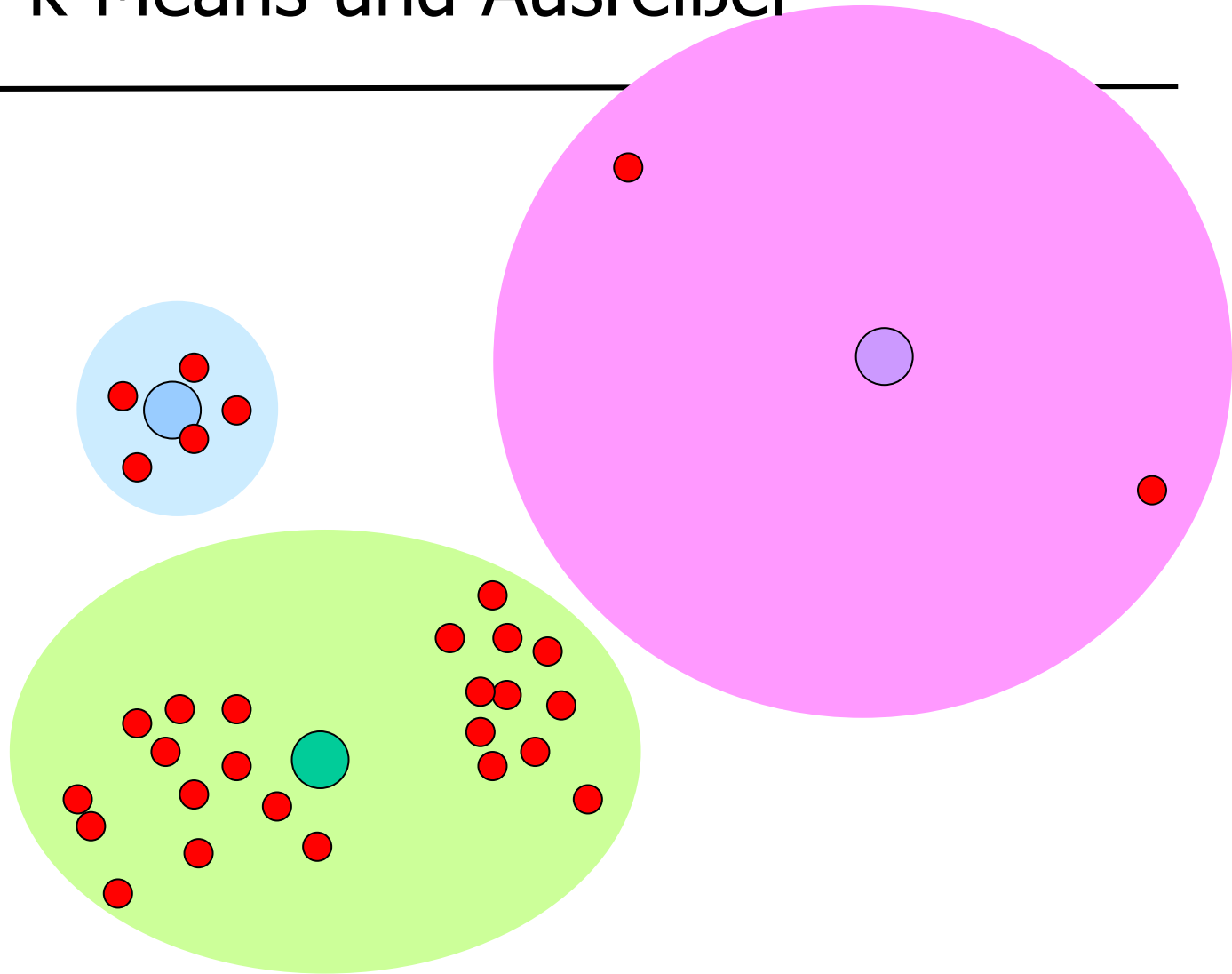
- Wähle **initiale Clusterzentren gleichmäßig verteilt** im Raum statt beliebige Datenpunkte zu nehmen
 - Schlecht für stark geclusterte Daten, da Mittelpunkte erst einen weiten Weg zurücklegen müssen
 - Gut für relativ gleichmäßig verteilte Punkte (mit lokalen Clustern)
 - Wie verteilt man Punkte gleichmäßig bei 1000-dimensionalen Daten?
- Stop, wenn nur noch **wenige (Schwellwert) Objekte ihre Zugehörigkeit** geändert haben
 - Wichtig! Konvergenz von k-Means ist nicht garantiert
 - Schneller, leichte Ungenauigkeiten
- Berechne neue Clustermittelpunkte nach jedem Wechsel
 - Mittelpunkte können **inkrementell aktualisiert werden**
 - Spart die zweite innere Schleife, aber Reihenfolge der Punkte wird wichtig
- Starte k-Means mehrmals mit **unterschiedlichen Startpunkten** und nimm das beste Ergebnis
 - Standardmethode, um zufällig schlechte Startkonstellationen zu verhindern



Eigenschaften

- In der Praxis konvergiert der Algorithmus relativ schnell (5-10 Läufe)
- Wenn l die Zahl der Durchläufe ist, brauchen wir
 - l Iterationen
 - Neuzuordnung: $n \cdot k$ Vergleiche Objekte-Zentren
 - Clusterbestimmung: n Vektoradditionen, verteilt auf k Cluster
 - Zusammen: $O(n \cdot k \cdot l)$
 - Das ist schnell
 - Insbesondere benötigen wir **keine Distanzmatrix**
- Nachteil: Welches k nehmen wir?
 - Alternative: Verschiedene k probieren
 - Silhouette zur Güteabschätzung verwenden
- Sehr empfindlich gegenüber **Ausreißern**
 - Verzerren die Struktur der Cluster
 - Es werden immer alle Punkte geclustered

k-Means und Ausreißer



SQL

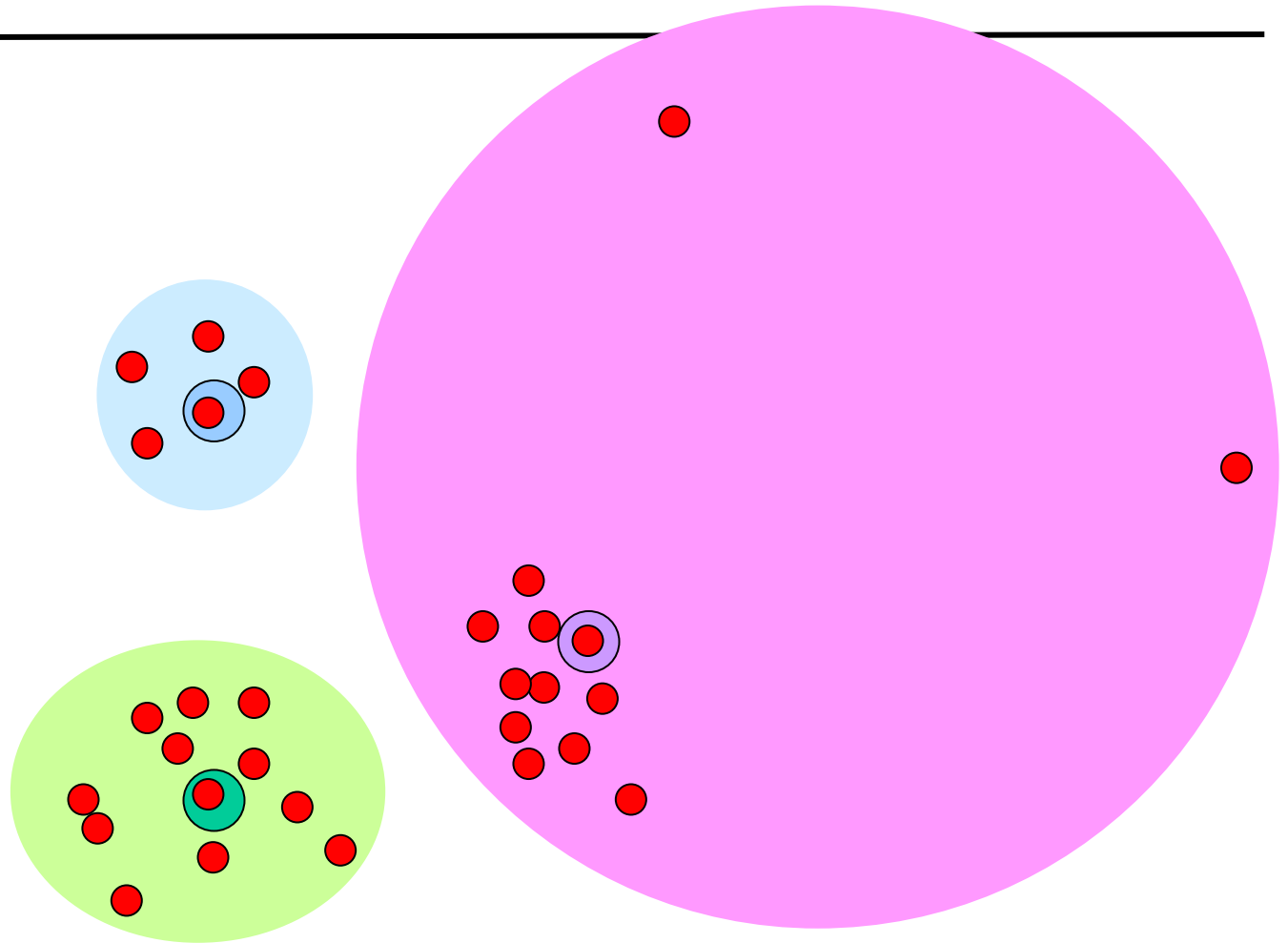
- Tabelle `objects` mit Objekten und Attributen
- Tabelle `cluster` speichert Koordinaten des Zentrums
- Erstes Update
 - Zuweisung aller Clusterzentren
- Zweites Update
 - Berechnung aller neuer Clustermittelpunkte
- **Probleme und Potential**
 - Auch nicht geänderte Cluster werden neu berechnet
 - Was passiert bei vielen Dimensionen?

```
REPEAT
  UPDATE objects
  SET cluster=
    (SELECT cid
     FROM (SELECT dist(o.a,a,...) d
           FROM cluster
           ORDER BY d)
     WHERE ROWNUM=1);
  IF %SQLCOUNT% != 0
    UPDATE cluster
    SET (a,b,...)=
      (SELECT sum(a)/n,sum(b)/n, ...
       FROM objects o
       WHERE o.cluster=cid);
  ELSE
    BREAK;
  ENDIF;
UNTIL FALSE;
```

K-Medoid

- K-Means arbeitet nicht mit kategoriellen Werten
 - Mittelpunkte sind nicht definiert
- Andere Methode: K-Mediod
 - Wahl des mittleren Punktes eines Clusters statt des Mittelpunkts
 - „Repräsentativer Punkt“ des Clusters
 - PAM: Partitioning around Medoids
- Problem?
 - Die Medoide können wir nicht einfach linear aus allen Objekten eines Clusters berechnen
 - Statt dessen: Berechne Güte für alle möglichen Medoide in jedem Cluster und wähle jeweils das Beste
 - Sehr teuer ($O(n^3)$)
 - Entspricht fast einem kompletten Test aller möglichen Clusterzentren
 - Reale Laufzeiten sind aber schneller
- Vorteil: Weniger sensitiv bzgl. Ausreißern

k-Medoid und Ausreißer



CLARANS

- Idee: Teste nur **manche Vertauschungen**

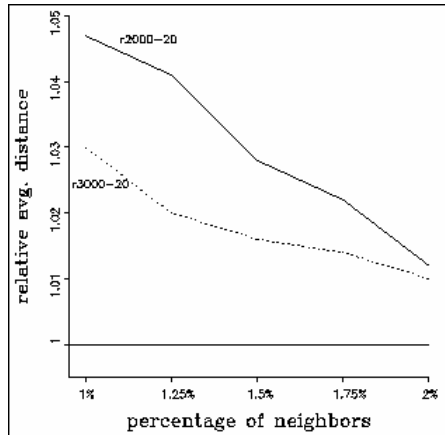
- maxneighbor
viele
- dafür starte
öfter (numlocal)

- CLARANS: A
Clustering
Algorithm
based on
Randomized
Search

```
for r from 1 to numlocal do
  wähle zufällig k Objekte als Medoide;
  i := 0;
  while i < maxneighbor do
    Wähle zufällig (Medoid M, Nicht-Medoid N);
    TD_Änderung := TDN↔M - TD;
    if TD_Änderung < 0 then
      ersetze M durch N;
      TD := TDN↔M;
      i := 0;
    else
      i := i + 1;
    end do;
  if TD < TD_best then
    TD_best := TD; Merke die aktuell besten Medoide;
  end if;
end for;
return Medoide;
```

Vergleich [ES00]

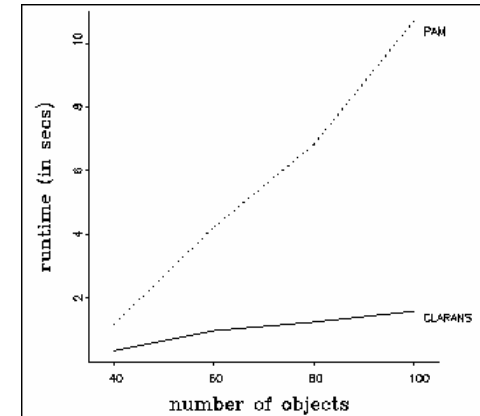
Qualität



TD(CLARANS)

TD(PAM)

Laufzeit



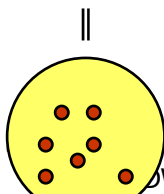
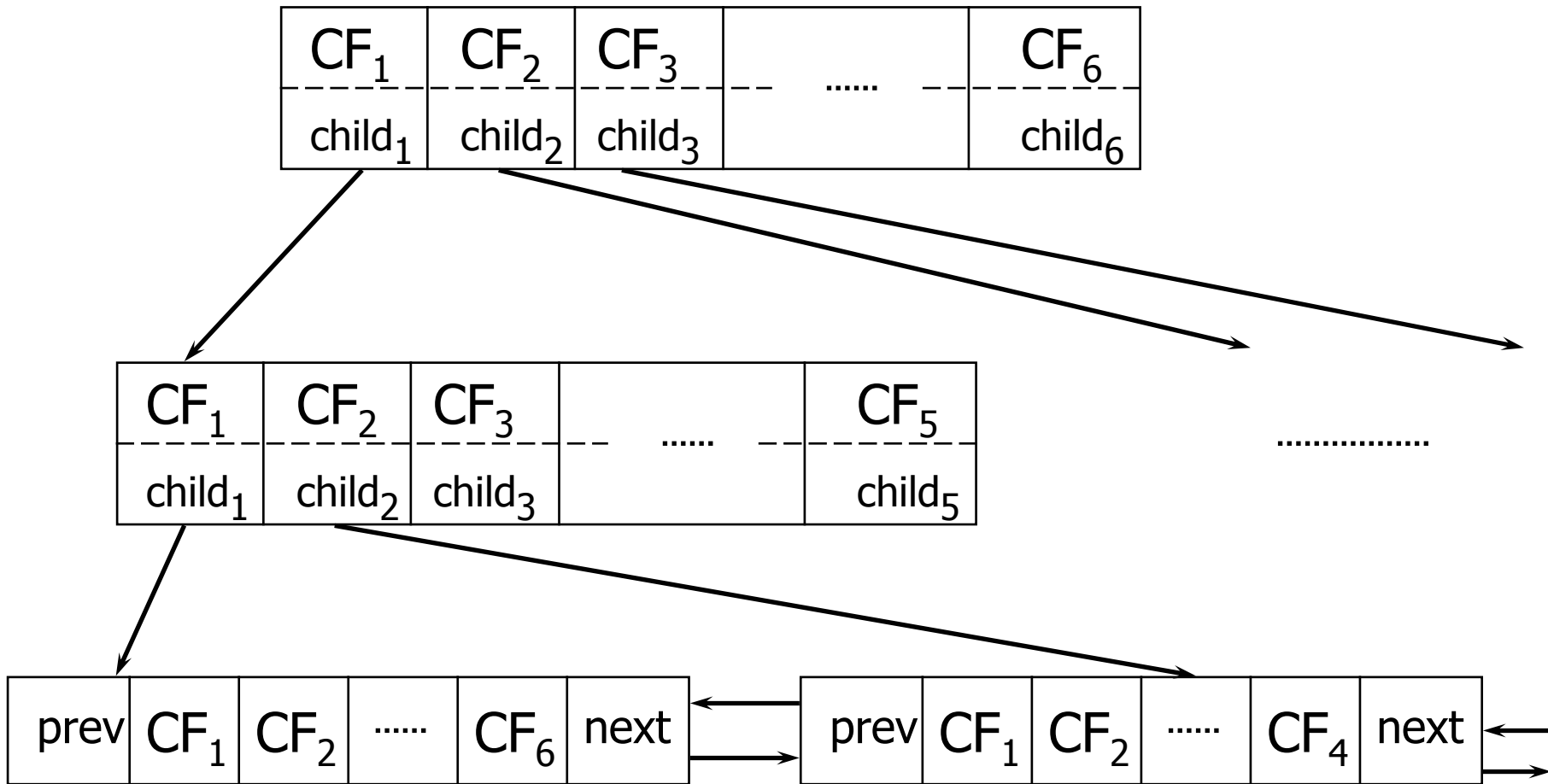
Quelle:
[ES00]

- Unwesentlich schlechtere Ergebnisse (1-5%)
- Viel bessere Laufzeit (nahezu linear)
- Nicht untypisch: Wenn die Daten „gut“ clustern, dann findet man diese Cluster sehr schnell
 - Zuordnung der wenigen problematischen Objekte benötigt dann viel Zeit, bringt aber nur wenig Verbesserung

Skalierbares k-Means: BIRCH

- K-Means und CLARANS greifen häufig und zufällig auf Objekte der Datenbank zu
 - Ineffizient und langsam bei sehr großen Datenbanken
- **BIRCH**: Balanced iterative reducing and clustering using hierarchies
 - Macht nur wenige Scans durch die Datenbank (minimal 2)
 - Phase 1: Im Speicher wird ein **Cluster Feature Tree** aufgebaut
 - Ähnlich einem B* Baum, repräsentiert hierarchische Cluster
 - Parameter sind der Branching Factor und ein Schwellwert t
 - T bestimmt die **minimale Güte** eines Knoten (Cluster)
 - Blattknoten speichern nur **Cluster Feature**: Zentrum, Radius und Güte
 - Neues Objekt: Finde nächsten Cluster und passe CF an
 - Das geht inkrementell und wegen der Baumstruktur sehr schnell
 - Bei „Überlauf“ (Cluster zu schlecht) teile Blattknoten
 - Wenn dabei der BF des Elternknoten überschritten wird – teile den auch (rekursiv den Baum rauf)
 - Wenn der Speicher voll ist: Erneuter Lauf mit größerem t

Beispiel [HK06]



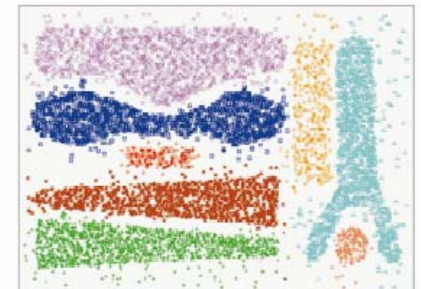
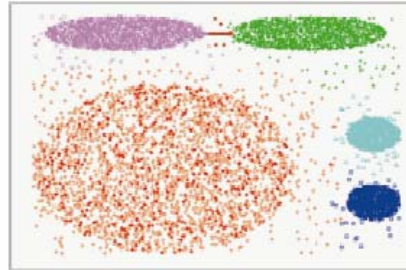
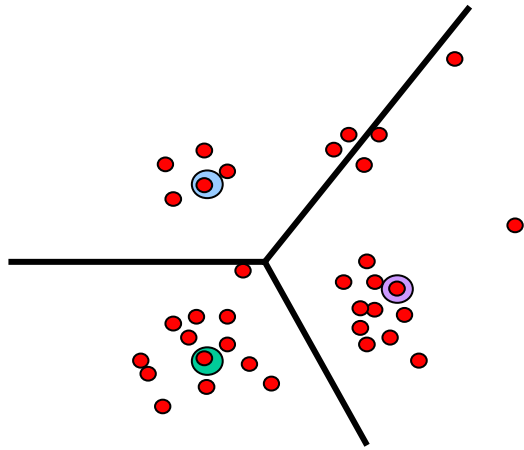
BIRCH Phase 2

- Nach Phase 1 repräsentiert jedes Blatt einen Cluster
 - Aber Objekte sind noch nicht zugeordnet
- Einfachste Möglichkeit
 - Scanne einmal durch die Daten und weise jedes Objekt seinem nächsten Cluster zu
- Verbesserungen
 - Merge/splitte Cluster mit ihren Nachbarn
 - Die findet man über den Baum
 - Viel Raum für Ideen

Inhalt dieser Vorlesung

- Einführung
- Hierarchisches Clustering
- Partitionierendes Clustering
- Dichte-basiertes Clustering
 - DBSCAN
 - Clustering und multidimensionale Indexstrukturen

Aber ...



Quelle:
[FPPS96]

- K-Means (und CLARANS und k-Medoid) finden nur **konvexe Clusters**
 - Das ergibt sich auf der Nähe zu **einem** Mittelpunkt

Dichtebasiertes Clustering [EKSX96]

- Sucht nach **Regionen hoher Dichte**
 - Anzahl Cluster ist nicht vorbestimmt
 - Findet auch **nicht konvexe** Cluster
- Bekanntester Vertreter: **DBSCAN**
- Wie definiert man „dichte“ Bereiche?
 - Jeder Punkt eines Clusters hat **viele nahe Nachbarn**
 - Alle Punkte eines Clusters sind über nahe Nachbarn voneinander erreichbar

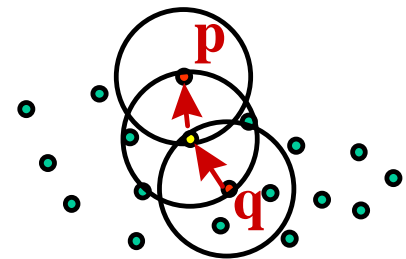
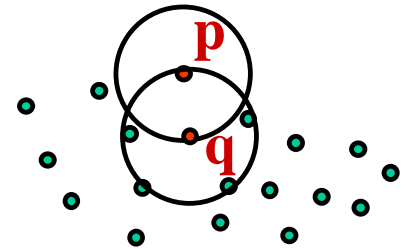


Grundbegriffe

- Definition

Geg. Parameter ε („Nachbar“) und minpts („viele“). Sei $N_\varepsilon(o)$ die ε -Nachbarschaft von Punkt o .

- Ein Objekt o heißt **Kernobjekt**, wenn $|N_\varepsilon(o)| \geq \text{minpts}$
- Ein Objekt p ist **direkt dichte-erreichbar** von einem Objekt q , wenn q ein Kernobjekt ist und $p \in N_\varepsilon(q)$
 - p muss kein Kernobjekt sein (Rand)
- p ist **dichte-erreichbar** von q , wenn es eine Kette von direkt dichte-erreichbaren Objekten zwischen p und q gibt.



- Bemerkung

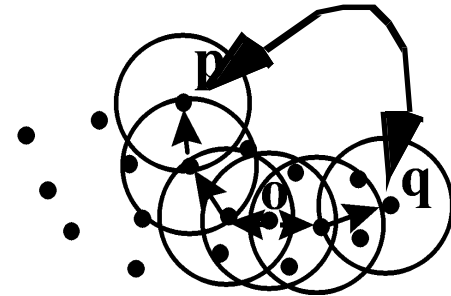
- Dichte-Erreichbarkeit erzeugt einen Kernbereich und einen Rand

Weitere Grundbegriffe

- Definition

Voraussetzungen wie eben.

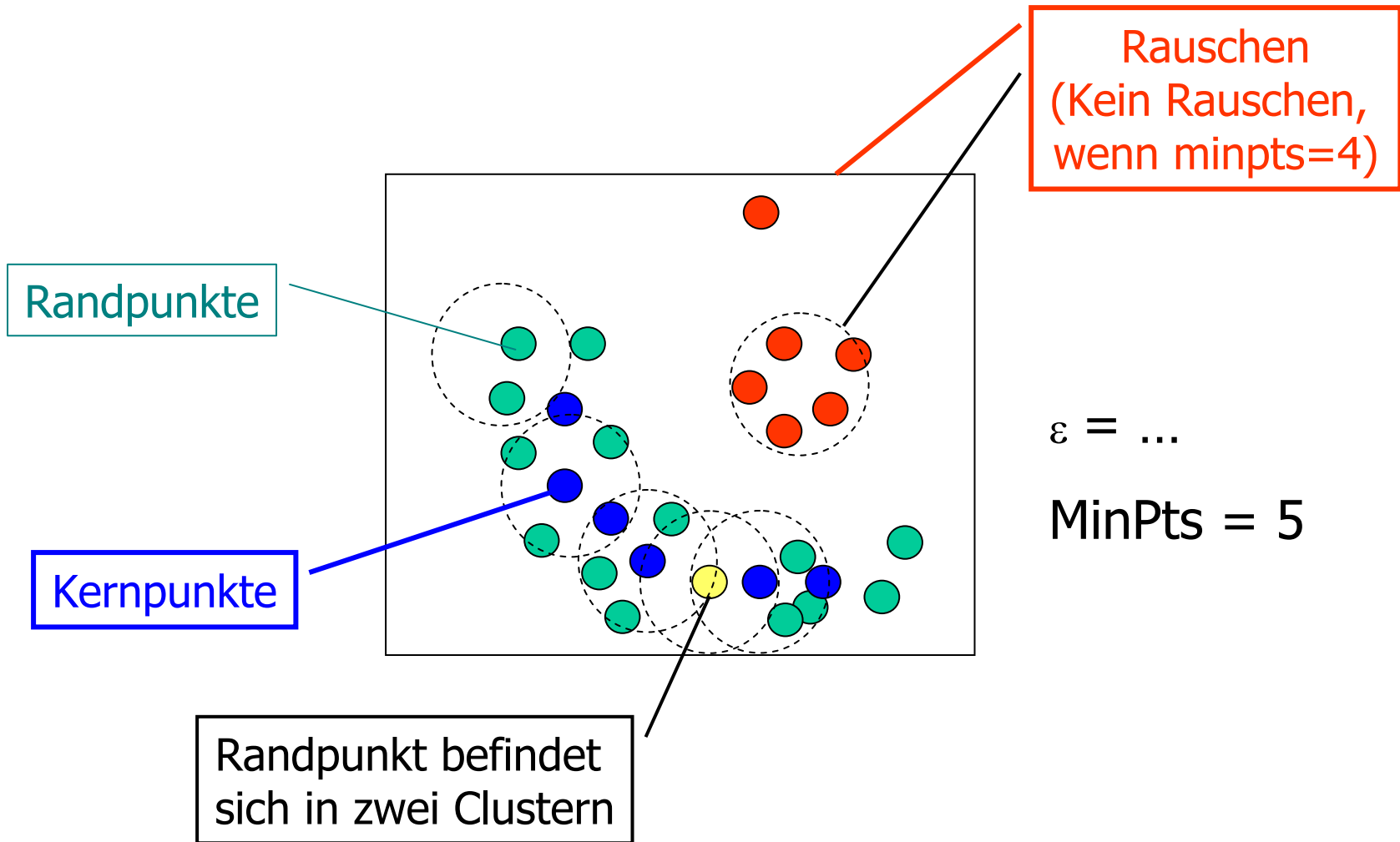
- Zwei Objekte p und q sind **dichte-verbunden**, wenn es ein mindestens Objekt o gibt, von dem p und q dichte-erreichbar sind.
 - Auch Randpunkte sind also dichte-verbunden
- Ein **Cluster** ist eine Teilmenge $C \subseteq O$ für die gilt
 - **Maximalität**: $\forall p, q \in O$: wenn $p \in C$ und q dichte-erreichbar von p ist, dann ist auch $q \in C$
 - **Verbundenheit**: $\forall p, q \in C$: p ist dichte-verbunden mit q
- Ein **Clustering** von O ist die Menge aller Cluster von O , die mindestens ein Kernobjekt enthalten
- Alle Punkte, die nicht in einem Cluster sind, heißen **Rauschen**



- Bemerkung

- Es gilt: Cluster C und $p \in C$ ein Kernobjekt: $C = \{o \in O \mid o \text{ dichte-erreichbar von } p\}$
- Cluster sind **nicht notwendigerweise disjunkt**
 - Aber überlappen höchstens sehr wenig

Beispiel



Algorithmus

- Aus der Definition ergibt sich unmittelbar ein Algorithmus zum Finden **des dichte-basierten Clusterings** einer Objektmenge O
 - Das Clustering ist eindeutig

```
clusterCount := 1;
for i from 1 to |O| do // Alle Punkte ansehen
  o := O.get(i);
  if o.clusterID = NULL and // Punkt in keinem Cluster
    kernobjekt(o) then // ... und im Kernbereich
    // eines (neuen) Clusters

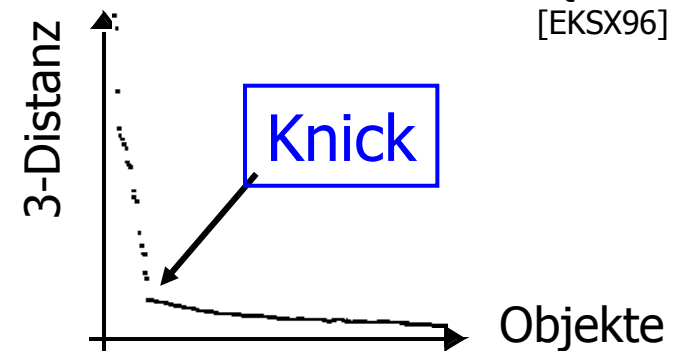
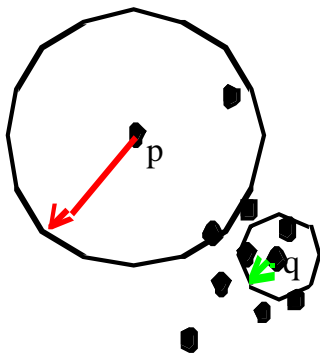
  if expCl( o, O, clusterCount ) != 0 // Cluster berechnen
  then clusterCount++; // Nächster Cluster
```

Analyse

- Benötigt – oberflächlich gesehen – nur einen Lauf durch die Daten
- Aber: **ExpandiereCluster ist teuer**
 - Sucht wiederholt nach allen Punkten in der ε -Nachbarschaft von Punkten o
 - Ohne multidimensionalen Index
 - Alle paarweisen Distanzen vorberechnen (Distanzmatrix – teuer)
 - Bei Anfrage für o : Alle Objekte p verwerfen mit $p \notin N_\varepsilon(o)$
 - Benötigt $O(n^2)$ Zeit und Platz - schlecht
 - **Mit multidimensionalem Index**
 - Multidimensionaler Index muss Nachbarschaftsqueries unterstützen
 - Damit: $O(n^*)$ Aufwand für eine ε -Query)
- Gleiches Problem beim Test `kernobjekt()`

Wie findet man die Parameter?

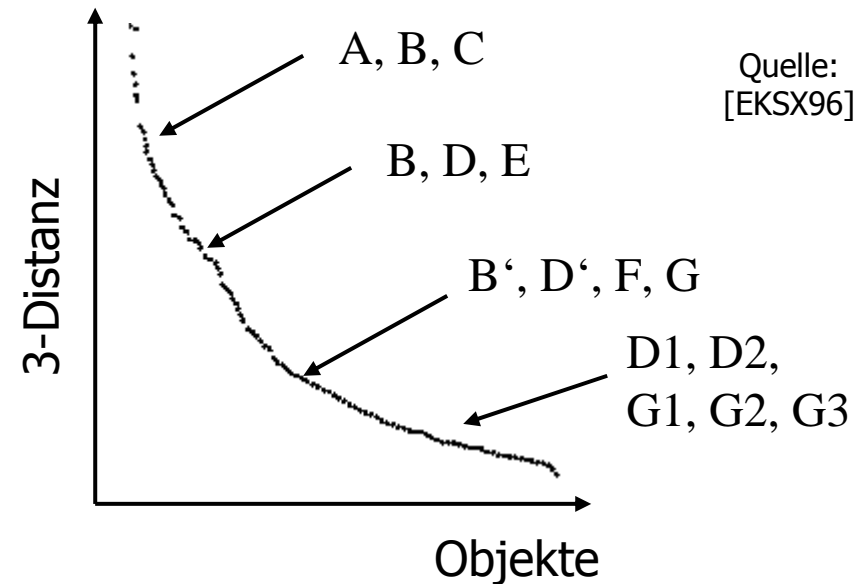
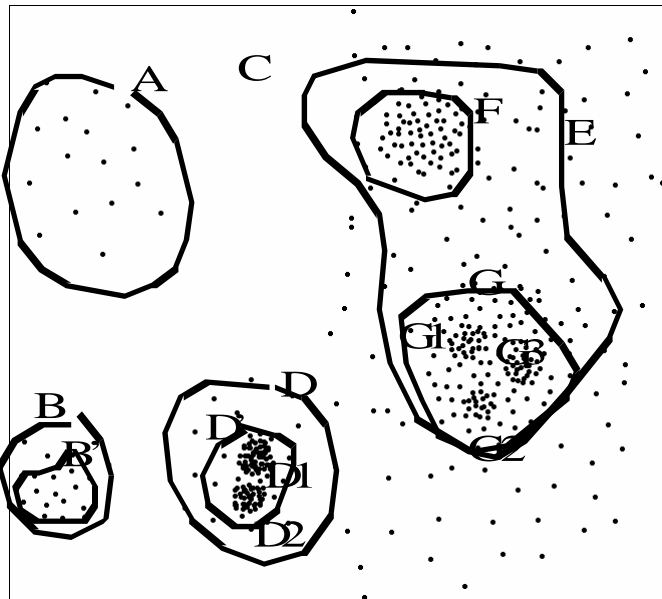
- Idee: Finde den am **wenigsten dichten Cluster** in den Daten
 - Der aber trotzdem ein Cluster ist – Definitionssache
- Für ein Objekt o ist seine **k-Distanz** die Entfernung des k -nächsten Objekt
- Damit können wir ein k-Distanz-Diagramm bauen
- Wähle den „Knick“ in der Verteilung



Quelle:
[EK SX96]

Probleme

- Und wenn es keinen Knick gibt?
- Schwierig: **Stark unterschiedlicher Dichte** in verschiedenen Bereichen des Raumes
 - Viele (kleine) Knicks
 - Mit einem Parameterpaar \minpts, ε kann man das nicht beschreiben
- Auch schwierig: Cluster und Rauschen sind nicht gut getrennt
 - Kein Knick

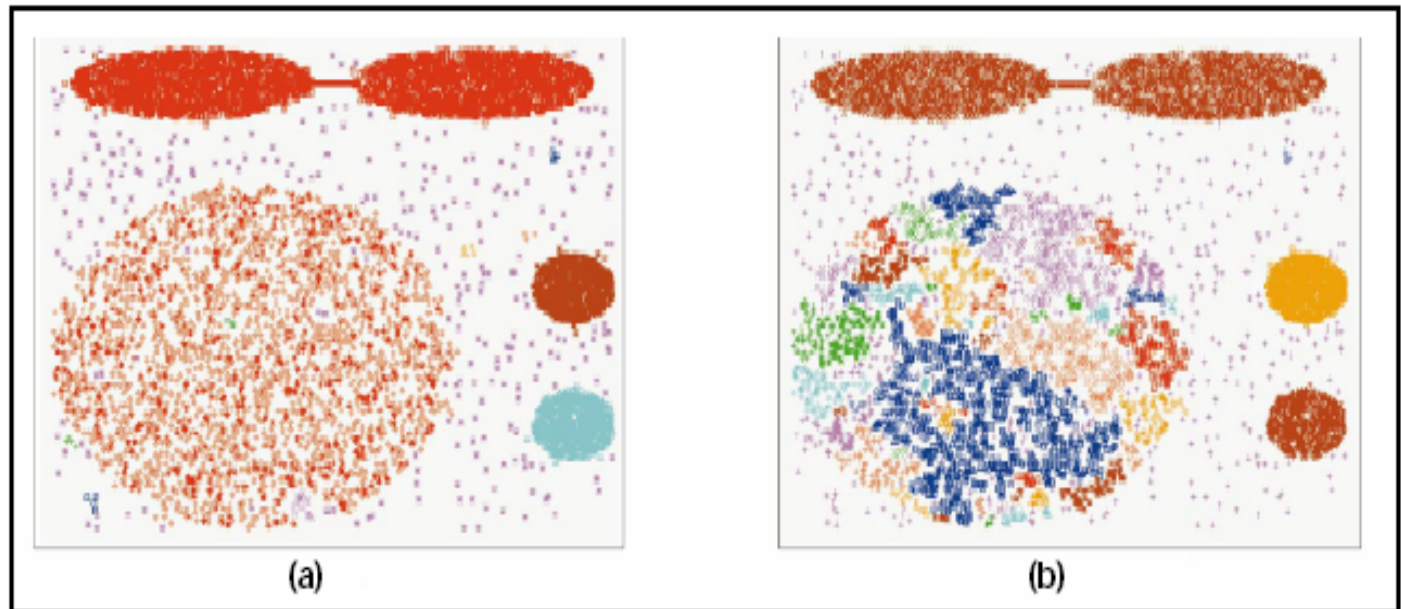


Quelle:
[EK SX96]

Sensitivität

- Wählen des „richtigen“ ε ist aber leider absolut essentiell

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.



Clustering und Indexstrukturen

- Recall: Multidimensionale Indexstrukturen basieren auf der Idee, im hochdimensionalen Raum **benachbarte Objekte** auch auf der ein/zweidimensionalen Disk benachbart zu speichern
 - Grid-File, kd-Baum, R-Baum
- Damit ist jeder MDI auch ein Clustering – mit Problemen
 - An den **Rändern** der Blöcke
 - Cluster sind alle gleich groß
 - Mancher Block ist fast leer
- Ein MDI kann aber wunderbar verwendet werden
 - Zur Unterstützung der ε -Anfragen von DBSCAN
 - Zur Auswahl guter Startpunkte bei k-Means oder CLARANS
 - Zur Auswahl guter Samples für BIRCH
 - Als Vor-Cluster (mit Nachbearbeitung): Grid-Cluster

Literatur

- Ester, M. and Sander, J. (2000). "Knowledge Discovery in Databases". Berlin, Springer.
- Han, J. and Kamber, M. (2006). "Data Mining. Concepts and Techniques", Morgan Kaufmann.
- Ester, M., Kriegel, H. P., Sander, J. and Xu, X. (1996). "A density-based algorithm for discovering clusters in large spatial databases". Conference on Knowledge Discovery in Databases.