Kapitel 10
Klassifikationsverfahren

Skript zur Vorlesung
Geo-Informationssysteme
Wintersemester 2011/12
Ludwig-Maximilians-Universität München

10. Klassifikationsverfahren

10.1 Einführung (I)

Problem

- Gegeben ein Rasterbild, das durch ein Fernerkundungssystem geliefert wurde.
- Die Grauwerte stellen gemessene Amplituden in bestimmten Spektralbereichen dar und sind für einen Benutzer nicht direkt zu interpretieren.
- Gesucht ist eine Zuordnung aller Pixel (x,y) zu je einer von wenigen disjunkten *Klassen Ci, $1 \leq i \leq k$,* die für eine gegebene Anwendung Bedeutung besitzen.

10.1 Einführung (II)

Anwendung

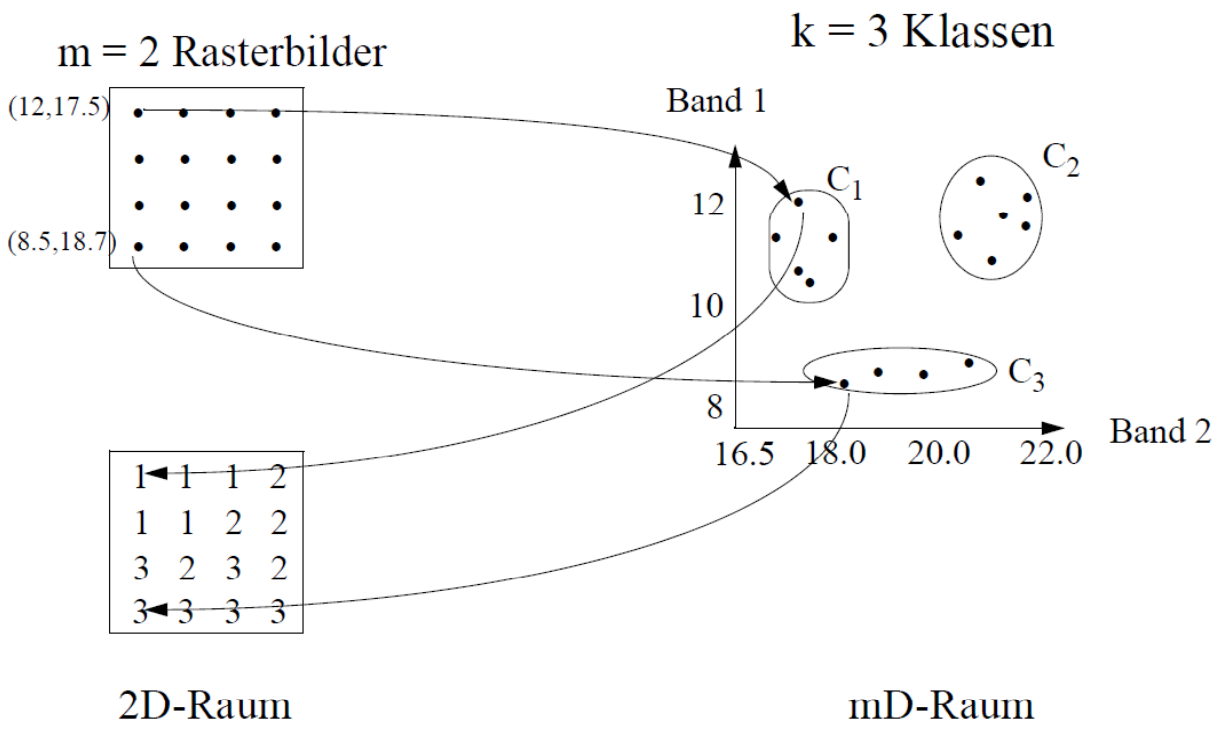
- Die Klassen können z.B. die Bodennutzung repräsentieren (*landuse).*
- Klassen z.B. water, urban, croplands, rangelands.

Lösungsansätze

- nicht überwachte Klassifikation (*unsupervised classification)*
- die Klassen C_i sind noch nicht bekannt, insbesondere ist auch die geeignete Anzahl k von zu unterscheidenden Klassen nicht vorgegeben
- überwachte Klassifikation (*supervised classification)*
  die Klassen C_i sind bereits bekannt

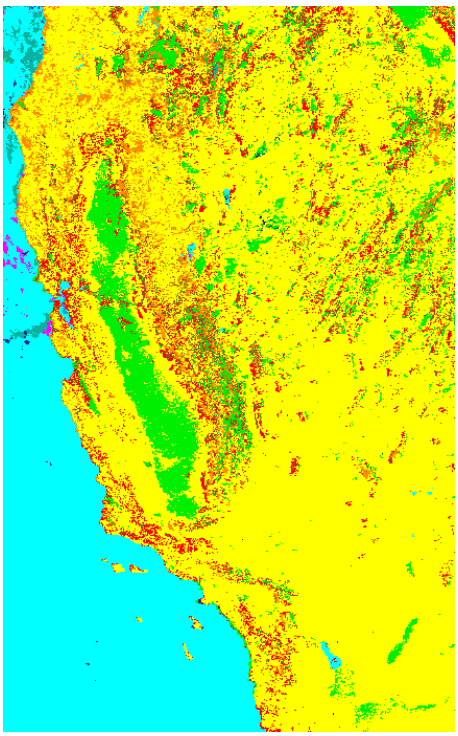
10.1 Einführung (III)

Schema einer Klassifikation



10.1 Einführung (IV)

Beispiel (Landnutzung von Kalifornien)



10.1 Einführung (V)

Überwachte Klassifikation

- Die Klassen $C_i, 1 \leq i \leq k$, *sind gegeben.*
- Gesucht ist eine Funktion f, die jedem Pixel (x,y) mit Grauwerten $d_1(x,y),\ldots, d_m(x,y)$ genau ein i, $1 \leq i \leq k$, zuordnet, d.h. eine *Klassifikationsfunktion.*
- Wir bezeichnen im folgenden mit D(x,y) den Grauwertvektor $(d_1(x,y),\ldots,d_m(x,y))$. $C_i\,(x,y)$ sei das Prädikat, das genau dann TRUE liefert, wenn $(x,y) \in C_i$.

10.1 Einführung (VI)

Überwachte Klassifikation: Vorgehen

- Festlegen der Klassen $C_i, 1 \leq i \leq k$.
- Manuelle Zuordnung der Klasse für eine kleine Zahl repräsentativer Pixel. Diese Menge T von Pixeln (x,y) mit bekannter Klasse c(x,y) bildet die *Trainingsdaten/Trainingspixel*. T enthält z.B. 1% aller Pixel des Eingangsbildes.
- Suche einer Klassifikationsfunktion f, die die Trainingsdaten möglichst korrekt klassifiziert, d.h. für möglichst viele Elemente von T soll gelten $f(x,y) = c(x,y)$.
- Automatische Zuordnung der Klasse für die restlichen Pixel mit Hilfe der gefundenen Klassifikationsfunktion *f*. Restliche Pixel z.B. *99%*.

10.1 Einführung (VII)

Nicht Überwachte Klassifikation (*Clustering)*

- Weder die Klassen (*Cluster)* C_i noch ihre Anzahl *k* sind gegeben.
- Gesucht sind sowohl die Klassen C_i , $1 \leq i \leq$ k,als auch eine geeignete Klassifikationsfunktion.

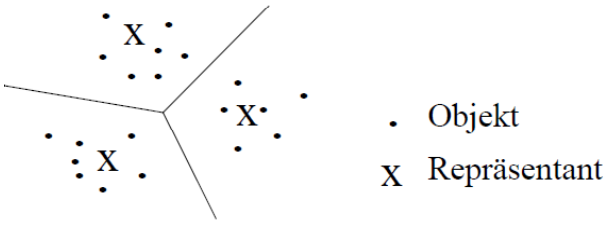
Vorgehen

- Definition der Ähnlichkeit zweier Pixel durch Definition einer Distanzfunktion *dist* basierend auf den *m* Grauwerten beider Pixel.
- Bestimmung einer Partitionierung aller Pixel in Klassen C_i so dass die Pixel einer Klasse möglichst ähnlich (kleiner Wert für dist) und die Pixel untersch. Klassen möglichst unähnlich (grosser Wert für dist) sind.
- Manuelle Interpretation der Klassen C_i mit Hilfe einer kleinen Zahl repräsentativer Pixel, für die die gesuchte Klasse bekannt ist. Z.B. stellt man fest, dass die Pixel, die C_1 bzw. C_2 zugeordnet wurden, Wasser bzw. bebautes Gebiet darstellen, d.h. C_1 = water C_2 = urban.

10.1 Typen von Clusteringalgorithmen (I)

Partitionierende Algorithmen

- Gegeben ist eine Datenbank *DB*, eine Distanzfunktion dist und ein Integer k.
- Gesucht ist eine Partitionierung (*Clustering)* in k Cluster mit minimalen Kosten.
- Zweistufiges Verfahren
  - Bestimme *k* Repräsentanten
  - ordne jedes Objekt aus *DB* dem nächstgelegenen Repräsentanten zu



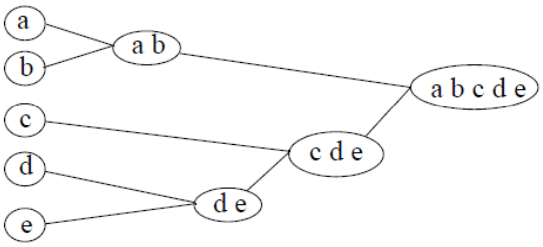
- Zwei Arten von Repräsentanten:
  - Schwerpunkt des Clusters (*k-means Algorithmen)*
  - ein Objekt des Clusters (*k-medoid Algorithmen)*

⇒Kosten einer Partitionierung = Σ *dist(o, repräsentant (o))*

10.1 Typen von Clusteringalgorithmen (II)

Hierarchische Algorithmen

- Gegeben: Datenbank *DB*, Distanzfunktion *dist* , Abbruchbedingung *Ab*
- Gesucht: hierarchische Aufteilung in Cluster, die *Ab* erfüllt.
- Das hierarchische Clustering wird als Dendrogramm dargestellt. Ein *Dendrogramm* ist ein Baum, dessen Knoten je einen Cluster repräsentieren und die folgende Eigenschaften besitzen:
  - die Wurzel repräsentiert die ganze *DB*
  - die Blätter repräsentieren die einzelnen Objekte aus *DB*
  - ein innerer Knoten repräsentiert die Vereinigung aller Objekte seiner Sohnknoten.



Dendrogramm
für DB = {a, b, c, d, e}

10.1 Typen von Clusteringalgorithmen (III)

Single Scan Algorithmen

- führen einen einzigen Durchlauf der Datenbank aus
- lokale Clusterbedingung wie z.B. minimale Dichte, passende Distanz zum Nearest Neighbor, . . .
- algorithmisches Schema wie folgt:

```
ALGORITHM SingleScanClustering (database DB, cluster condition COND)
    FOR each object o of DB DO
        IF o is not yet member of some cluster THEN
            create a new cluster C;
            WHILE neighboring objects of o satisfy COND DO
                add them to C
            ENDWHILE
        ENDIF
    ENDFOR
END SingleScanClustering
```

10.2 Partitionierendes Clustering (I)

Grundlagen

- Ziel: Partitionierung in k Cluster, so dass eine Kostenfunktion minimiert wird (Gütekriterium: Kompaktheit)
- Zentrale Annahmen:
  - Anzahl k der Cluster bekannt (Eingabeparameter)
  - Clustercharakteristik: Kompaktheit
  - Kompaktheit: Abweichung aller Objekte im Cluster von einem ausgezeichneten Cluster-Repräsentanten ist minimal
  - Kompaktheitskriterium führt meistens zu sphärisch geformten Clustern
- Erschöpfende Suche ist zu ineffizient (WARUM?)
- Daher: Lokal optimierende Verfahren
  - wähle k initiale Cluster-Repräsentanten
  - optimiere diese Repräsentanten iterativ
  - ordne jedes Objekt seinem ähnlichsten Repräsentanten zu

10.2 Partitionierendes Clustering (II)

PAM (Partitioning Around Medoids)

ALGORITHM PAM (Datenbank DB, Distanzfunktion dist, int k)

Medoide:={}; Nichtmedoide := DB; Wähle das zentralste Objekt aus DB als ersten Medoid;

FOR i FROM 2 TO k DO

wähle den bisherigen Nichtmedoid als weiteren Medoid, der die Kosten der Partitionierung am stärksten reduziert;

END FOR;

LOOP

FOR each pair (Medoid, Nichtmedoid) DO

berechne die Änderung der Kosten der Partitionierung, die durch das Ersetzen des Medoids durch den Nichtmedoid bewirkt würde;

END FOR;

Sei (M,N) das Paar mit minimaler Änderung K der Kosten;

IF K < 0 THEN

Ersetze den Medoid M durch den Nichtmedoid N;

ELSE EXIT LOOP;

END IF;

END LOOP;

10.2 Partitionierendes Clustering (III)

Komplexitätsanalyse von PAM

- Auswahl der initialen k Medoide: $O(n^2)$ für jeden Medoid, d.h. Zeitkomplexität $O(k * n^2)$
- Zahl der Durchläufe des LOOP: ???
- Zeitkomplexität eines Durchlaufs:
  - (n - k) * k Berechnungen der Änderung der Kosten
  - eine Berechnung ist $O(n) \Rightarrow O(k * n^2)$

Verbesserung 1

Nicht das Paar (Medoid, Nichtmedoid) mit minimaler Änderung (< 0) suchen sondern für das erste Paar, das die Kosten reduziert, die Ersetzung durchführen ⇒ Algorithmus CLARANS

Verbesserung 2

Die Medoide nicht mit allen Objekten der DB bestimmen sondern nur auf einer Stichprobe von DB (reduziere n) ⇒ Sampling

10.2 Partitionierendes Clustering (IV)

CLARANS (Clustering Large Applications based on RANdomized Search)

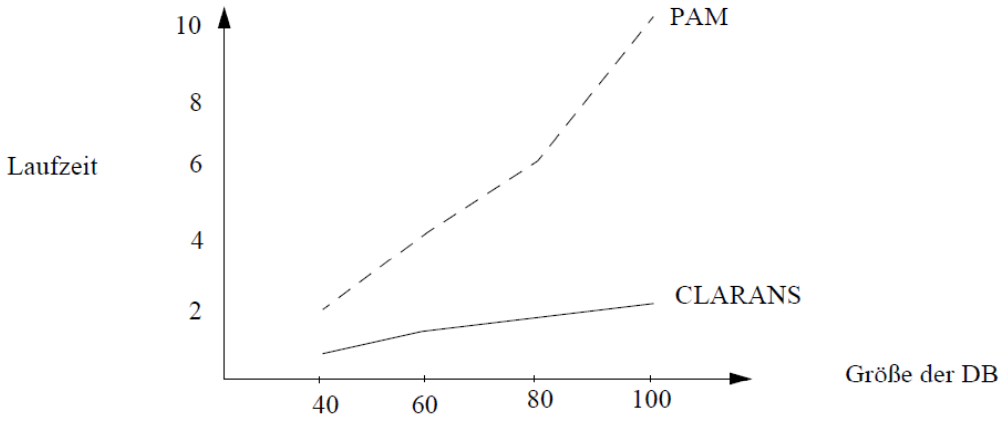
ALGORITHM CLARANS (Datenbank DB, int k, Distanzfunktion dist, int numlocal, int maxneighbor)
Medoide := {}; Nichtmedoide := DB; Kmin := MAXFLOAT; BestMedoide := {};
FOR i FROM 1 TO numlocal DO
wähle zufällig k der Objekte aus DB als Medoide;
WHILE NOT maxneighbor Paare (Medoid, Nichtmedoid) betrachtet DO
wähle zufällig einen der Medoide und einen der Nichtmedoide;
berechne die Änderung K der Kosten der Partitionierung, die durch das Ersetzen des Medoids durch den Nichtmedoid bewirkt würde;
IF K < 0 THEN
Ersetze den Medoid durch den Nichtmedoid;
Beginne bei der Zählung der Paare wieder mit 0;
END IF;
END WHILE;// Innere Schleife
berechne die Kosten K der aktuellen Partitionierung;
IF K < Kmin THEN
BestMedoide := Medoide;
Kmin := K;
END IF; END FOR;// Äußere Schleife

10.2 Partitionierendes Clustering (V)

Laufzeitkomplexität von CLARANS

- Auswahl des Paars (Medoid,Nichtmedoid): O(1)
- Berechnung der potentiellen Änderung der Kosten: O(n)
- Ersetzung des Medoids durch den Nichtmedoid: O(1)
- Anzahl der Durchläufe der Inneren Schleife: ???

Leistungsuntersuchung von CLARANS



10.2 Partitionierendes Clustering (VI)

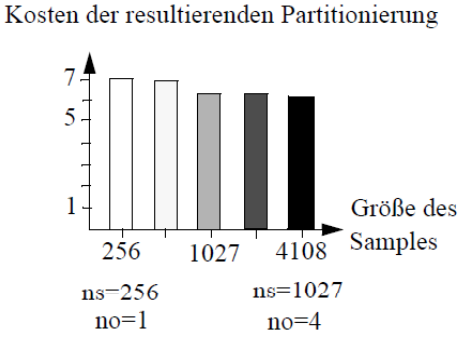
Sampling

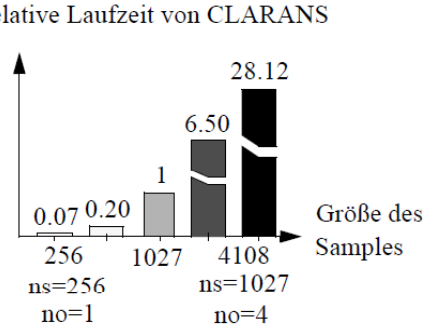
- Ein *Sample* ist eine repräsentative Stichprobe einer Menge von Daten.
- R*-Baum erhält räumliche Nachbarschaft so gut wie möglich.

⇒ wähle ein gleichverteiltes Sample von den Datenseiten des R*-Baums

Größe des Samples

- Zwei Parameter: ns = Anzahl auszuwählender Datenseiten, no = Anzahl auszuwählender Objekte pro ausgewählter Datenseite
- experimentelle Untersuchung (mit DB von 55.000 Punkten)

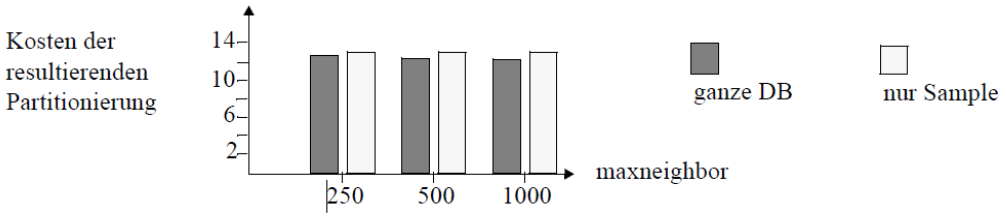




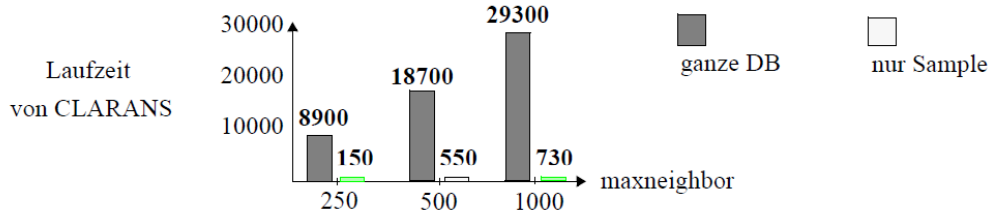
10.2 Partitionierendes Clustering (VII)

Leistungsuntersuchung des Sampling

- Vergleich der Effektivität



- Vergleich der Effizienz



=> kleiner Verlust an Effektivität und großer Gewinn an Effizienz

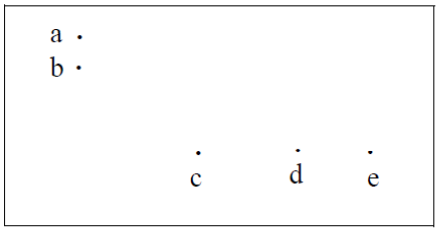
10.3 Hierarchisches Clustering (I)

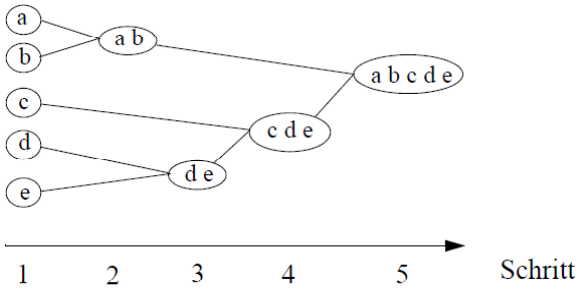
Single Link

ALGORITHMUS SingleLinkClustering (Datenbank DB, Distanzfunktion dist, Abbruchbedingung Ab)
FOR i **FROM** 1 **TO** Size(DB) **DO**
Cluster[i]:= i-tes Objekt von DB;
END ;
FOR i **FROM** 1 **TO** Size(DB) **DO**
Cluster[i].Link:= Cluster j mit minimaler dist zu Cluster i;
Cluster[i].LinkDist:= dist(Cluster[i],Cluster[i].Link);
END;
WHILE Bedingung Ab nicht erfüllt **DO**
Sei Cluster [i] das Cluster mit minimalem Wert von LinkDist;
Verschmelze Cluster[i] und Cluster[i].Link;
Aktualisiere die Werte für Cluster[i].Link und Cluster[i].LinkDist;
Aktualisiere die Werte für Link und LinkDist bei allen Clustern, deren Link Cluster[i].Link war;
Dekrementiere die aktuelle Anzahl von Clustern um 1;
END WHILE;

10.3 Hierarchisches Clustering (II)

Beispiel





Nachteile des Single Link Algorithmus

- Abbruchbedingung erfordert Bereichswissen

  ⇒ interaktive Variante: Benutzer analysiert Dendrogramm und bestimmt eine Ebene, an der das Dendrogramm abgeschnitten werden soll
- Anfälligkeit gegen Rauschen: eine Linie von Rauschpunkten kann zwei Cluster zu einem verbinden
- Ineffizienz für große Datenbanken: in jedem Schritt wird die Distanz aller Paare von aktuellen Clustern betrachtet: Laufzeitkomplexität $O(n^2)$

10.4 Single Scan Clustering (I)

DBSCAN (Density Based Spatial Clustering of Applications with Noise)

- ist ein *Single Scan Clustering* Algorithmus
- Clusterbedingung: die "Dichte" in der Eps-Umgebung eines Punkts ist höher als ein vorgegebener Grenzwert MinPts.
- DBSCAN berechnet für MinPts = 2 eine Ebene eines Dendrogramm
- DBSCAN unterstützt den Benutzer beim Bestimmen geeigneter Werte für die Parameter Eps und MinPts.
- Die Dichte in einer Umgebung ändert sich durch Einfügen weniger "Ausreißer"-Punkte weniger als die Distanz der Punkte zum nächsten Nachbarn.

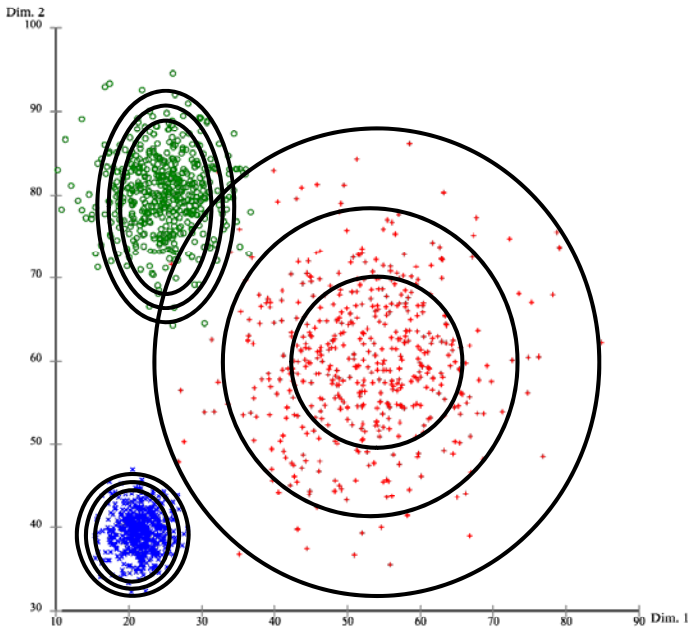
  ⇒ geringere Anfälligkeit gegen Rauschen
- Betrachte nicht die Distanz aller Paare von aktuellen Clustern, sondern betrachte nur die Distanz zum aktuellen Cluster.

  ⇒ Effizienz auch für große Datenbanken

10.4 Single Scan Clustering (II)

Intuition

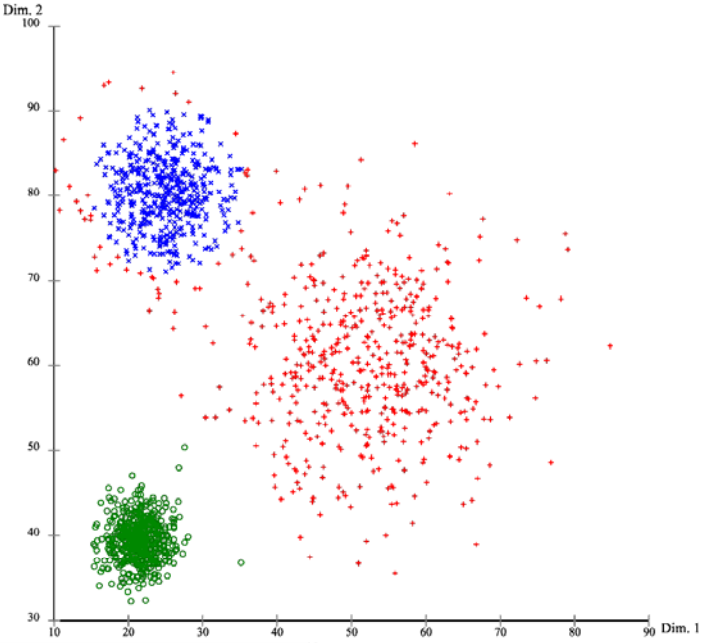
- selektiere Bereiche, die mit Mindest-Dichte verbunden sind

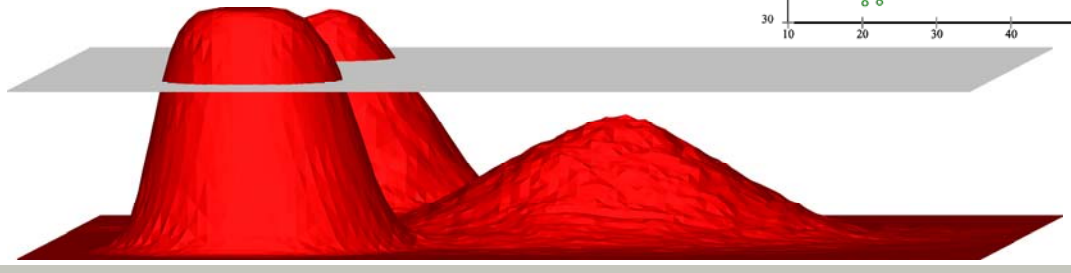


10.4 Single Scan Clustering (III)

Intuition

- hohes Dichte-Niveau: Cluster geringer Dichte gilt als Noise

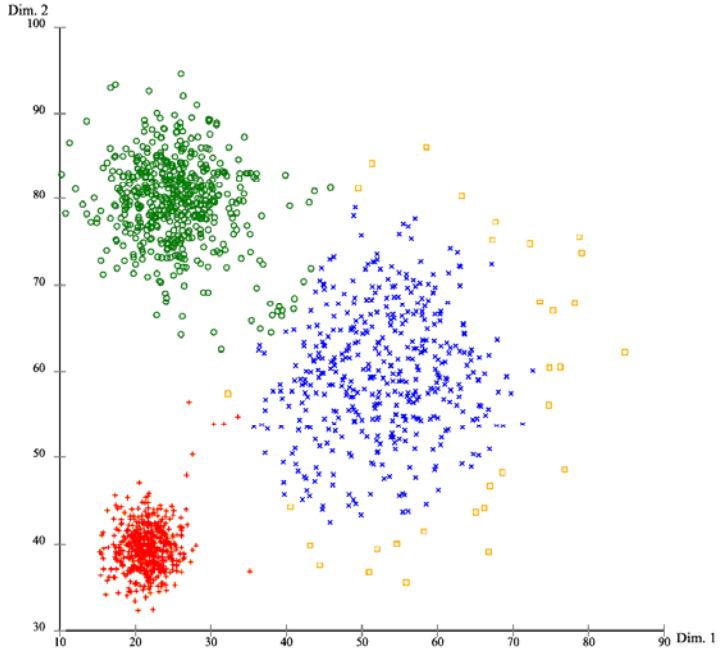


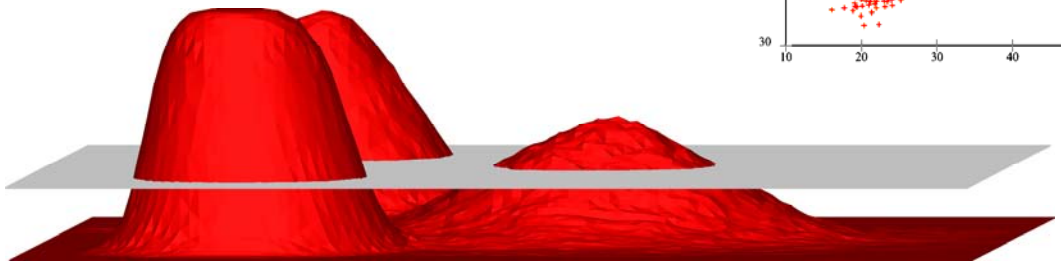


10.4 Single Scan Clustering (IV)

Intuition

- mittleres Dichte-Niveau: Identifikation von drei Clustern, einige Punkte in den Ausläufern der Verteilungen werden zu Noise

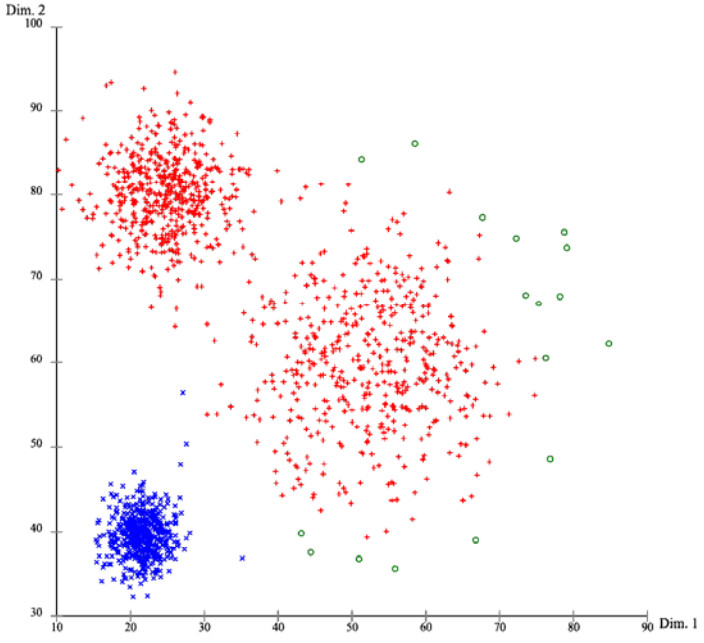


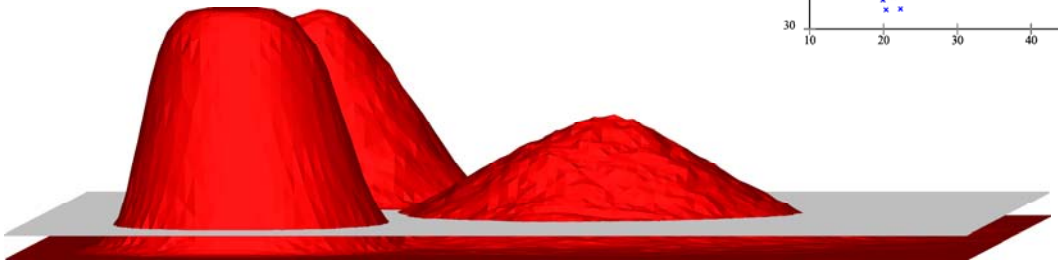


10.4 Single Scan Clustering (V)

Intuition

- geringes Dichte-Niveau: zwei Cluster verschmelzen

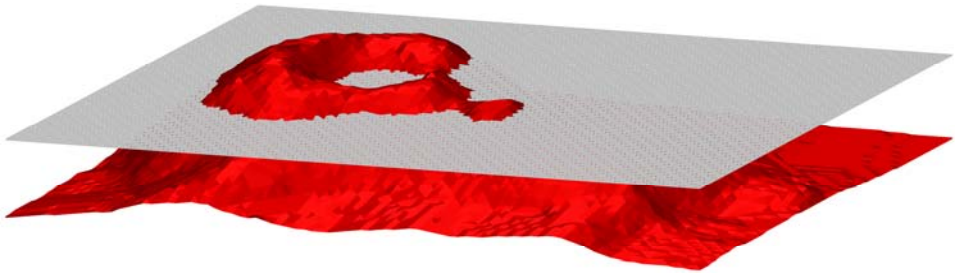




10.4 Single Scan Clustering (VI)

Intuition

- Partitionierend: Optimierungsproblem
- Dichte-basiert: „natürliche“ Strukturen (oft: geographisch/ topographisch, oder biologisch)
- Beispiel: Höhenprofil (Vulkan Mt. Eden, Auckland)



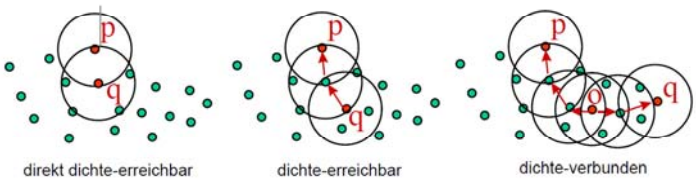
10.4 Single Scan Clustering (VII)

Notationen

- Ein Punkt *p* ist direkt dichte-erreichbar von einem Punkt *q* (bzgl. Eps und MinPts), wenn
  - 1) $|U_{Eps}(q)| \geq$ MinPts (*q* ist Kernpunkt)
  - 2) $p \in U_{Eps}(q)$

    ⇒ die direkte Dichte-Erreichbarkeit ist i.a. nicht symmetrisch
- Ein Punkt *p ist dichte-erreichbar von einem Punkt q,* wenn es eine Kette von Punkten p1, ..., pn mit p1 = q, pn = p gibt, so dass pi+1 direkt dichte-erreichbar von pi ist.
- Ein Punkt *p ist dichte-verbunden mit einem Punkt q,* wenn es einen Punkt o gibt, so dass sowohl p als auch q von o dichte-erreichbar sind.

  ⇒ die Dichte-Verbundenheit ist eine symmetrische Relation



10.4 Single Scan Clustering (VIII)

Notationen (Forts.)

- Ein *Cluster C* ist eine nichtleere Teilmenge einer Datenbank *D* mit folgenden Eigenschaften:
  - 1) $\forall\ p, q \in D: p \in C \wedge q$ dichte-erreichbar von $p \rightarrow q \in C$ (Maximalität)
  - 2) $\forall\ p, q \in C: p$ *ist* dichte-verbunden mit q (Verbundenheit)
- Seien $C_1, \ldots, C_k$ die Cluster der Datenbank D. Dann definieren wir das *Rauschen* als die Teilmenge aller Punkte von *D*, die zu keinem Cluster C_i gehören.

10.4 Single Scan Clustering (IX)

Beobachtungen

- Sei p ein Kernpunkt von *D*. Dann ist die Menge aller Punkte aus *D*, die von *p* dichte-erreichbar sind, ein Cluster.

  ⇒wenn man von einem Punkt ausgehend alle Punkte sammelt, die von ihm dichte-erreichbar sind, dann findet man einen Cluster

- Sei *C* ein Cluster und p ein beliebiger Kernpunkt dieses Clusters. Dann ist *C* gleich der Menge aller Punkte aus *D*, die von *p* dichte-erreichbar sind.

  ⇒von jedem seiner Kernpunkte ausgehend findet man den selben Cluster

10.4 Single Scan Clustering (X)

Algorithmus DBSCAN

DBSCAN (Datenbank DB, Eps real, MinPts int)

```
FOR i FROM 1 TO Size(DB) DO
  Punkt:= i-tes Objekt von DB;
  IF ClusterId von Punkt = UNCLASSIFIED THEN
        IF ExpandiereCluster(DB,Punkt,ClusterId,Eps,MinPts) THEN
       ClusterId:= nächste ClusterId;
        END IF;
  END IF;
END FOR;
```


10.4 Single Scan Clustering (XI)

Algorithmus DBSCAN (Forts.)

```
METHOD ExpandiereCluster (Datenbank DB, Punkt p, ClusterId int, Eps real, MinPts int): boolean
Seed:= RegionQuery in DB für Eps-Umgebung von p;
IF Size(Seed) < MinPts THEN  setze die ClusterId von p auf NOISE; RETURN FALSE;
ELSE
   Setze die ClusterId aller Punkte aus Seed auf ClusterId;  Lösche p aus Seed;
   WHILE Seed nichtleer DO
      AktuellerPunkt := erstes Element aus Seed;
      Ergebnis:= RegionQuery in DB für Eps-Umgebung von AktuellerPunkt;
      IF Size(Ergebnis) >= MinPts THEN
         FOR each Punkt IN Ergebnis DO
              IF ClusterId von Punkt IN {UNCLASSIFIED, NOISE} THEN
             setze ClusterId von Punkt auf ClusterId; füge Punkt in Seed ein;
         END FOR;
      END IF;
      Lösche AktuellerPunkt aus Seed;
   END WHILE;
   RETURN True;
END IF;
```

10.4 Single Scan Clustering (XII)

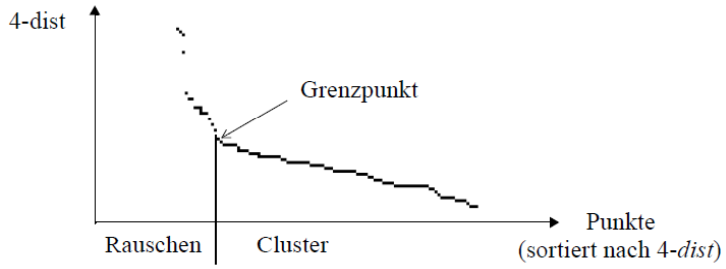
Bestimmung der Parameter

- Gesucht sind die Werte Eps und MinPts des "dünnsten" Clusters.
- Wir definieren die Funktion *k-dist,* die jedem Punkt die Distanz zu seinem k-nächsten Nachbarn zuordnet. Wenn wir die Punkte aus D nach absteigendem *k-dist* Wert sortieren erhalten wir den *sortierten k-dist Graphen.*

  [Beobachtung: die sortierten *k-dist* Graphen für $k > 4$ unterscheiden sich nicht signifikant von demjenigen für $k = 4$]
- Suche das erste "Tal" im sortierten 4-*dist* Graphen *(Grenzpunkt).*
  - Punkt liegt "links" vom Grenzpunkt: Rauschen
  - Punkt liegt "rechts" vom Grenzpunkt: Cluster

  ⇒ MinPts = 4 und

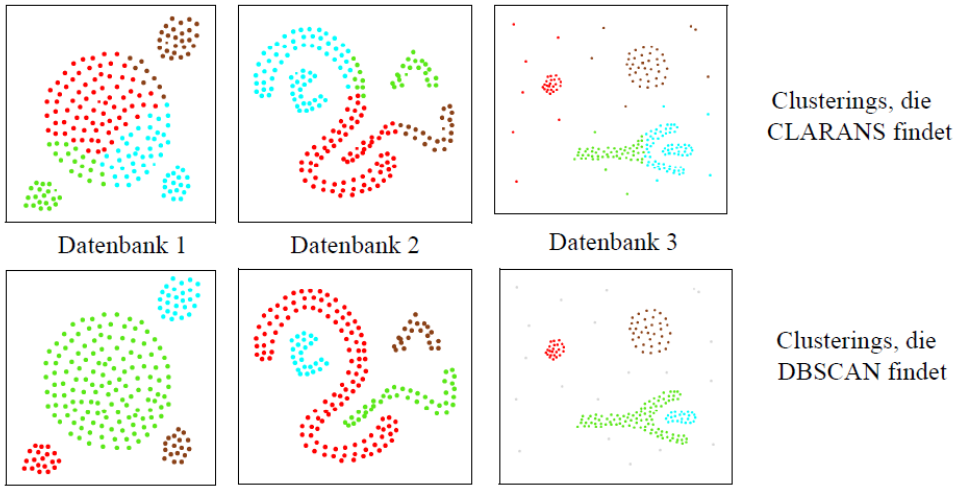
  Eps = 4-*dist des Grenzpunkts*



10.4 Single Scan Clustering (XIII)

Leistungsuntersuchung

Gefundene Clusterings



Laufzeit in sec

Anzahl der Punkte	1252	2503	3910	5213	6256	7820	8937	10426	12512
DBSCAN	3.1	6.7	11.3	16.0	17.8	24.5	28.2	32.7	41.7
CLARANS	758	3026	6845	11745	18029	29826	39265	60540	80638