UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Knowledge Discovery in Databases

Erich Schubert

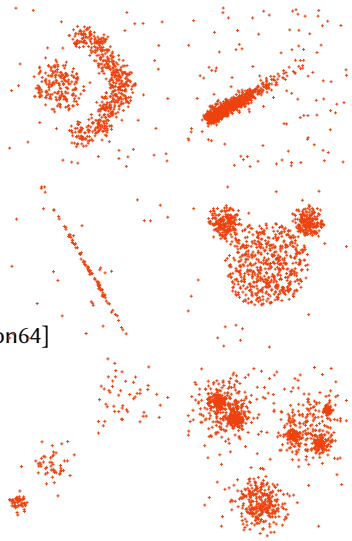
Winter Semester 2017/18

Part III

Clustering

What is Clustering?

Cluster analysis (clustering, segmentation, quantization, …) is the data mining core task to **find clusters**.



But what is a cluster? [Est02]

- cannot be precisely defined
- many different principles and models have been defined
- even more algorithms, with very different results
- when is a result "valid"?
- results are subjective "in the eye of the beholder"
- no specific definition seems "best" in the general case [Bon64]

Common themes found in definition attempts:

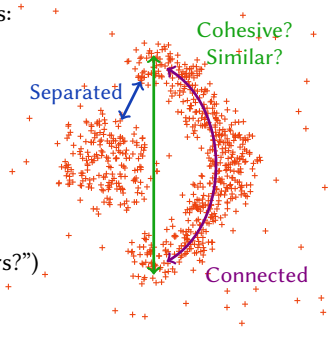
- more homogeneous
- more similar
- cohesive

Suggested reading: V. Estivill-Castro. "Why so many clustering algorithms – A Position Paper". In: *SIGKDD Explorations* 4.1 (2002), pp. 65–75

What is Clustering? /2

Cluster analysis (clustering, segmentation, quantization, …) is the data mining core task to **divide the data into clusters** such that:

- *similar* (related) objects should be in the *same* cluster
- *dissimilar* (unrelated) objects should be in *different* clusters
- clusters are *not* defined beforehand (otherwise: use classification)
- clusters have (statistical, geometric, …) properties such as:
  - connectivity
  - separation
  - least squared deviation
  - density



Clustering algorithms have different

- cluster models ("what is a cluster for this algorithm?")
- induction principles ("how does the algorithm find clusters?")

Applications of Clustering

We can use clustering **stand-alone** for data exploration,
or as **preprocessing step** for other algorithms.

Usage examples:

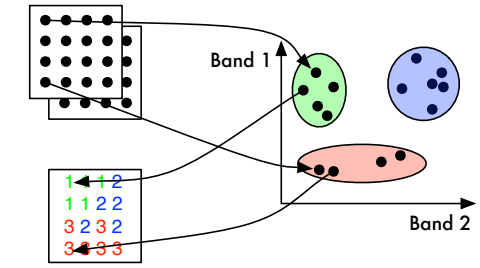
- Customer segmentation:
  Optimize ad targeting or product design for different "focus groups".
- Web visitor segmentation:
  Optimize web page navigation for different user segments.
- Data aggregation / reduction:
  Represent many data points with a single (representative) example.
  E.g., reduce color palette of an image to k colors
- Text collection organization:
  Group text documents into (previously unknown) topics.

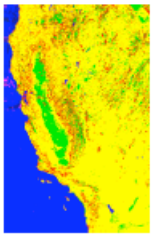
Applications of Clustering /2

- Biology: taxonomy of living things: kingdom, phylum, class, order, family, genus, and species
- Information retrieval: document clustering
- Land use: identification of areas of similar land use in an Earth observation database
- Marketing: help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- City-planning: Identifying groups of houses according to their house type, value, and geographical location
- Earthquake studies: observed Earthquake epicenters should be clustered along continent faults
- Climate: understanding Earth climate, find patterns of atmospheric and oceanic phenomena
- Economic Science: market research

Applications of Clustering /3

Construction of thematic maps from satellite imagery





Raster image of California
using 5 wave bands:
5-d feature space

Basis

- different surfaces of the Earth have different properties in terms of reflections and emissions
- cluster satellite imagery into *different land use* (forest, agriculture, desert, buildings, ...)

Basic Steps to Develop a Clustering Task

- Feature selection
  - Select information (about objects) concerning the task of interest
  - Aim at minimal information redundancy
  - Weighting of information
- Proximity measure
  - Similarity of two feature vectors
- Clustering criterion
  - Expressed via a cost function or some rules
- Clustering algorithms
  - Choice of algorithms
- Validation of the results
  - Validation test
- Interpretation of the results
  - Integration with applications

Distances, Metrics and Similarities

In *mathematics*, a metric or distance function is defined as a function $\mathrm{dist} : X \times X \to \mathbb{R}^{\geq 0}$ that satisfies the conditions

(N) $\mathrm{dist}(x, y) \geq 0$ (non-negativity)

(I) $\mathrm{dist}(x, y) = 0 \Leftrightarrow x = y$ (identity of indiscernibles)

(S) $\mathrm{dist}(x, y) = \mathrm{dist}(y, x)$ (symmetry)

(T) $\mathrm{dist}(x, y) \leq \mathrm{dist}(x, o) + \mathrm{dist}(o, y)$ (triangle inequality)

In data mining, we have a less perfect world.

- For (I), we require only $x = y \Rightarrow \mathrm{dist}(x, y) = 0$, because we often have duplicate records
- We often do not have (T).
- We often allow the distance to be ∞.

Common terminology used in data mining:

- **metric**: satisfies (N), (S), and (T).
- **distance**: satisfies (N) and (S).
- **dissimilarity**: usually (but not always) satisfies (N) and (S).

Distances, Metrics and Similarities /2

A similarity is a function $s : X \times X \to \mathbb{R} \cup \{\pm\infty\}$ such that low values indicate dissimilar objects, and large values indicate similarity.

Similarities can be

- Symmetric: $s(x, y) = s(y, x)$
- Non-negative: $s(x, y) \geq 0$
- Normalized: $s(x, y) \in [0; 1]$ and $s(x, x) = 1$

If $\mathrm{dist}(x, y)$ is a distance function, then it implies a non-negative, symmetric, normalized similarity:

$$s(x, y) := \frac{1}{1 + \mathrm{dist}(x, y)} \in [0; 1]$$

If $s(x, y)$ is a normalized and symmetric similarity, then $\mathrm{dist}(x, y) = 1 - s(x, y)$ is a distance (with at least (N) and (S), but usually not a metric).

Distance Functions

$\|x\|$ without index usually refers to the Euclidean norm.

There are *many* different distance functions.

The "standard" distance on $\mathbb{R}^d$ is Euclidean distance:

$$d_{\text{Euclidean}}(x,y) = \sqrt{\sum\nolimits_d (x_d - y_d)^2} \qquad = \|x-y\|_2$$

Manhattan distance (city block metric):

$$d_{\text{Manhattan}}(x,y) = \sum\nolimits_d |x_d - y_d| \qquad = \|x-y\|_1$$

are special cases of Minkowski norms (L_p distances, $p \geq 1$):

$$d_{L_p}(x,y) = \left(\sum\nolimits_d |x_d - y_d|^p\right)^{1/p} = \|x-y\|_p$$

In the limit $p \to \infty$ we also get the maximum norm:

$$d_{\text{Maximum}}(x,y) = \max\nolimits_d |x_d - y_d|$$

- many more distance functions – see the "Encyclopedia of Distances" [DD09]!

Distance Functions /2

Many more variants:

Squared Euclidean distance ("sum of squares"):

$$d^2_{\text{Euclidean}}(x,y) = \sum\nolimits_d (x_d - y_d)^2 \qquad = \|x-y\|_2^2$$

Minimum distance ($p \to 0$) – not metric:

$$d_{\text{Minimum}}(x,y) = \min\nolimits_d |x_d - y_d|$$

Weighted minkowski norms:

$$d_{L_p}(x,y,\omega) = \left(\sum\nolimits_d \omega_d |x_d - y_d|^p\right)^{1/p}$$

Canberra distance:

$$d_{\text{Canberra}}(x,y) = \sum\nolimits_d \frac{|x_d - y_d|}{|x_d| + |y_d|}$$

- many more distance functions – see the "Encyclopedia of Distances" [DD09]!

Similarity Functions

Instead of distances, we can often also use similarities:

Cosine similarity:

$$s_{\cos}(x,y) := \frac{x \cdot y}{\|x\| \cdot \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}}$$

on data normalized to L_2 length $\|x\| = \|y\| = 1$, this simplifies to:

$$s_{\cos}(x,y) \underset{\|x\|=\|y\|=1}{:=} x \cdot y = \sum\nolimits_i x_i y_i$$

Popular transformations to distances:

$$d_{\cos_1}(x,y) = 1 - s_{\cos}(x,y)$$
$$d_{\cos_2}(x,y) = \arccos(s_{\cos}(x,y))$$

- Careful: if we use similarities, large values are better – with distances, small values are better.

Distances for Binary Data

Binary attributes: two objects $o_1, o_2 \in \{0,1\}^d$ having only binary attributes

- Determine the following quantities:
  f_{01} = the number of attributes where o_1 has 0 and o_2 has 1
  f_{10} = the number of attributes where o_1 has 1 and o_2 has 0
  f_{00} = the number of attributes where o_1 has 0 and o_2 has 0
  f_{11} = the number of attributes where o_1 has 1 and o_2 has 1
- **Simple matching coefficient** (SMC):

$$s_{\text{SMC}}(o_1, o_2) = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{00} + f_{11}} = \frac{f_{11} + f_{00}}{d}$$

  used for *symmetric* attributes where each state (0, 1) is equally important.
- **Simple matching distance** (SMD):

$$d_{\text{SMC}}(o_1, o_2) = 1 - s_{\text{SMC}}(o_1, o_2) = \frac{f_{01} + f_{10}}{d} = \frac{d_{\text{Hamming}}(o_1, o_2)}{d}$$

Jaccard Coefficient for Sets

Jaccard coefficient for sets A and B (if $A = B = \emptyset$, we use similarity 1):

$$s_{\text{Jaccard}}(A, B) := \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad \in [0; 1]$$

Jaccard distance for sets A and B:

$$d_{\text{Jaccard}}(A, B) := 1 - s_{\text{Jaccard}}(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad \in [0; 1]$$

If we encode the sets as binary vectors, we get:

$$s_{\text{Jaccard}}(o_1, o_2) = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}$$

$$d_{\text{Jaccard}}(o_1, o_2) = \frac{f_{01} + f_{10}}{f_{01} + f_{10} + f_{11}}$$

Example Distances for Categorical Data

Categorical data: two lists $x = \{x_1, x_2, \ldots, x_d\}$ and $y = \{y_1, y_2, \ldots, y_d\}$
with each x_i and y_i being categorical (nominal) attributes

- **Hamming distance**: "count the number of differences".

$$\text{dist}_{\text{Hamming}}(x, y) = \sum_{i=1}^{d} \delta(x_i, y_i) \text{ with } \delta(x_i, y_i) = \begin{cases} 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

- **Jaccard** by taking the position into account.
- **Gower's** for mixed interval, ordinal, and categorical data.

$$\text{dist}_{\text{Gower}}(x, y) = \sum_{i=1}^{d} \begin{cases} 0 & \text{if } x_i = y_i \\ 1 & \text{if variable } X_i \text{ categorical and } x_i \neq y_i \\ \frac{|x_i - y_i|}{R_i} & \text{if variable } X_i \text{ is interval scale} \\ \frac{|\text{rank}(x_i) - \text{rank}(y_i)|}{n-1} & \text{if variable } X_i \text{ is ordinal scale} \end{cases}$$

where $R_i = \max_x x_i - \min_x x_i$ is the range, and $\text{rank}(x) \in [1, \ldots, n]$ is the rank.
Intuition: each attribute has the same maximum distance contribution 1.

Mahalanobis Distance

Given a data set with variables $X_1, \dots X_d$, and n vectors $x_1 \dots x_n$, $x_i = (x_{i1}, \dots, x_{id})^T$.
The $d \times d$ covariance matrix is computed as:

$$\Sigma_{ij} := \overline{S}_{X_i X_j} = \frac{1}{n-1} \sum_{k=1}^{n} (x_{ki} - \overline{X_i})(x_{kj} - \overline{X_j})$$

The **Mahalanobis distance** is then defined on two vectors $x, y \in \mathbb{R}^d$ as:

$$d_{Mah}(x, y) = \sqrt{(x-y)^T \Sigma^{-1} (x-y)}$$

where Σ^{-1} is the inverse of the covariance matrix of the data.

- This is a generalization of the Euclidean distance
- This takes correlation between attributes into account
- The attributes can have different ranges of values

Scaling & Normalization

Gower's distance and Mahalanobis distance scale attributes.
This can be beneficial (but also detrimental) for other distances, too!
Popular normalization approaches:

- To unit range:

$$x'_i = \frac{x_i - \min X_i}{\max X_i - \min X_i} \in [0; 1]$$

- To unit variance:

$$x'_i = \frac{x_i - \overline{X}_i}{\overline{S}_{X_i}}$$

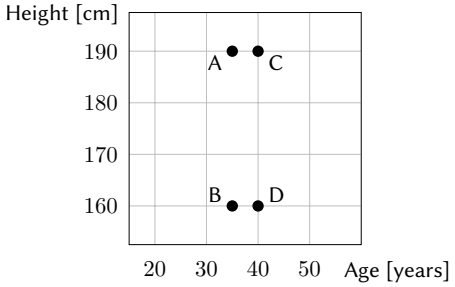
  such that $\overline{X'}_i = 0, \overline{S}_{X'_i} = 1$

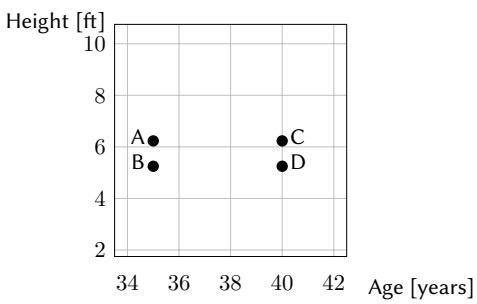
- Principal component analysis (PCA) such that $\overline{S}_{X'_i X'_j} = 0$ for $i \neq j$.
- Many more …
- Scaling $\sim$ feature weighting!

To Scale, or not to Scale?

Person	Age [years]	Height [cm]	Height [feet]
A	35	190	6.232
B	35	160	5.248
C	40	190	6.232
D	40	160	5.248

Which persons are more similar, A and B, or A and C?



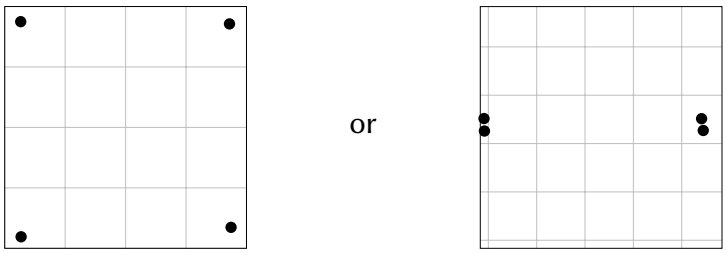


Here, scaling often improves the results!

To Scale, or not to Scale? /2

Object	Longitude	Latitude
Palermo	13.37	38.12
Venice	12.34	45.44
San Francisco	-122.42	37.78
Portland, OR	-122.68	45.52

To scale or not to scale?



Don't scale! We need to *know* what the attributes mean!

Different Kinds of Clustering Algorithms

There are *many* clustering algorithms, with different

Paradigms:

- Distance
- Similarity
- Variance
- Density
- Connectivity
- Probability
- Subgraph
- …

Properties:

- Partitions: strict, hierarchical, overlapping
- Models: Prototypes, Patterns, Gaussians, Subgraphs, …
- Outliers / noise, or complete clustering
- Hard (binary) assignment or soft (fuzzy) assignment
- Full dimensional or subspace clustering
- Rectangular, spherical, or correlated clusters
- Iterative, or single-pass, or streaming
- …

Quality: What Is Good Clustering?

A good clustering method will produce high quality clusters

- high **intra-class similarity**: cohesive within clusters
- low **inter-class similarity**: distinctive between clusters

The quality of a clustering method depends on

- the similarity measure used by the method,
- its implementation, and
- its ability to discover some or all of the hidden patterns

Quality of clustering

- There is usually a separate "quality" function that measures the "goodness" of a cluster or a clustering
- It is hard to define "similar enough" or "good enough", i.e., the answer is typically highly subjective and specific to application domain

Requirements and Challenges

- Scalability
  - Clustering all the data instead of only sample data
- Ability to deal with different types of attributes
  - Numeric, binary, categorical, ordinal, linked, and mixture of these
- Constraint-based clustering
  - User may give input as constraints
  - Use domain knowledge to determine input parameters
- Interpretability and usability
- Others
  - Discovery of clusters with arbitrary shape
  - Ability to deal with noisy data
  - Incremental clustering and insensitivity to input order
  - High dimensionality

Major Clustering Approaches

Partitioning approaches

- Construct partitions that optimize some criterion, e.g., minimizing the sum of squared errors
- Typical methods: k-means, PAM (k-medoids), CLARA, CLARANS

Density-based approaches

- Based on connectivity, density, and distance functions
- Typical methods: DBSCAN, OPTICS, DenClue, HDBSCAN*

Hierarchical approaches

- Create a hierarchical decomposition of the set of data (or objects) using some criterion
- Typical methods: DIANA, AGNES, BIRCH

Model-based

- Optimize the fit of a hypothesized model to the data
- Typical methods: Gaussian Mixture Modeling (GMM, EM), SOM, COBWEB

Major Clustering Approaches /2

Grid-based approach

- based on a multiple-level granularity structure
- Typical methods: STING, WaveCluster, CLIQUE

Frequent pattern-based

- Based on the analysis of frequent patterns
- Typical methods: p-Cluster

User-guided or constraint-based

- Clustering by considering user-specified or application-specific constraints
- Typical methods: COD (obstacles), constrained clustering

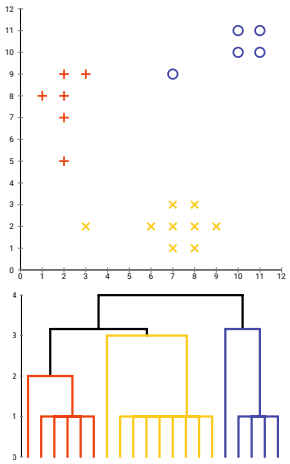
Link-based clustering

- Objects are often linked together in various ways
- Massive links can be used to cluster objects: SimRank, LinkClus

Hierarchical Agglomerative Clustering

One of the earliest clustering methods [Sne57; Sib73; Har75; KR90]:

1. Initially, every object is a cluster
2. Find two most similar clusters, and merge them
3. Repeat (2) until only one cluster remains
4. Plot tree ("dendrogram"), and choose interesting subtrees



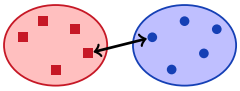
Many variations that differ by:

- Distance / similarity measure of *objects*
- Distance measure of clusters ("Linkage")
- Optimizations

Distance of Clusters

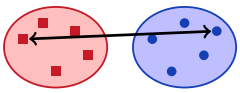
Single-linkage: minimum distance ≅ maximum similarity

$$d_{\text{single}}(A,B) := \min_{a\in A, b\in B} d(a,b) \cong \max_{a\in A, b\in B} s(a,b)$$



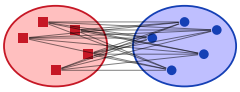
Complete-linkage: maximum distance ≅ minimum similarity

$$d_{\text{complete}}(A,B) := \max_{a\in A, b\in B} d(a,b) \cong \min_{a\in A, b\in B} s(a,b)$$



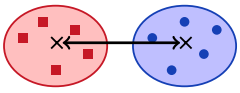
Average-linkage (UPGMA): average distance ≅ average similarity

$$d_{\text{average}}(A,B) := \frac{1}{|A|\cdot|B|} \sum_{a\in A} \sum_{b\in B} d(a,b)$$



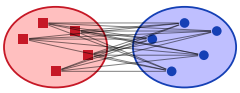
Centroid-linkage: distance of cluster centers (Euclidean only)

$$d_{\text{centroid}}(A,B) := \|\mu_A - \mu_B\|^2$$

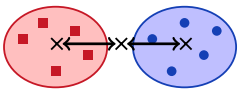


Distance of Clusters /2

McQuitty (WPGMA): average of previous sub-clusters
Defined recursively, e.g., via Lance-Williams equation.
Average distance to the previous two clusters.

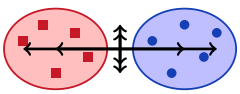


Median-linkage (Euclidean only): distance from midpoint
Defined recursively, e.g., via Lance-Williams equation.
Median is the halfway point of the previous merge.



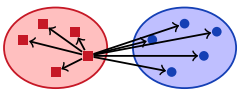
Ward-linkage (Euclidean only): Minimum increase of squared error

$$d_{\text{Ward}}(A,B) := = \frac{|A|\cdot|B|}{|A\cup B|} \|\mu_A - \mu_B\|^2$$



Mini-Max-linkage: Best maximum distance, best minimum similarity

$$d_{\text{minimax}}(A,B) := \min_{c\in A\cup B} \max_{p\in A\cup B} d(c,p) \cong \max_{c\in A\cup B} \min_{p\in A\cup B} s(c,p)$$



AGNES – Agglomerative Nesting [KR90]

AGNES, using the Lance-Williams equations [LW67]:

1. Compute the pairwise distance matrix of objects
2. Find position of the minimum distance $d(i, j)$ (similarity: maximum similarity $s(i, j)$)
3. Combine rows and columns of i and j into one using Lance-Williams update equations

$$d(A \cup B, C) = \text{LanceWilliams}\left(d(A, C), d(B, C), d(A, B)\right)$$

   using only the *stored, known* distances $d(A, C)$, $d(B, C)$, $d(A, B)$.[1]
4. Repeat from (2.) until only one entry remains
5. Return dendrogram tree

[1] *Avoid* to compute $d(A, C)$ directly, as this is expensive: $O(n^2)$.

AGNES – Agglomerative Nesting [KR90] /2

Lance-Williams update equation have the general form:

$$D(A \cup B, C) = \alpha_1 d(A, C) + \alpha_2 d(B, C) + \beta d(A, B) + \gamma |d(A, C) - d(B, C)|$$

Several (but not all) linkages can be expressed in this form (for distances):

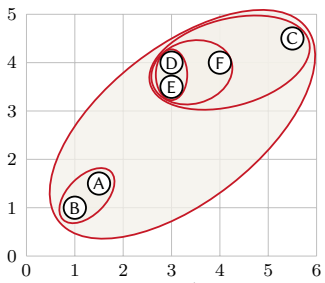
	α_1	α_2	β	γ
Single-linkage	$1/2$	$1/2$	0	$-1/2$
Complete-linkage	$1/2$	$1/2$	0	$+1/2$
Average-group-linkage (UPGMA)	$\frac{\lvert A\rvert}{\lvert A\rvert+\lvert B\rvert}$	$\frac{\lvert B\rvert}{\lvert A\rvert+\lvert B\rvert}$	0	0
McQuitty (WPGMA)	$1/2$	$1/2$	0	0
Centroid-linkage (UPGMC)	$\frac{\lvert A\rvert}{\lvert A\rvert+\lvert B\rvert}$	$\frac{\lvert B\rvert}{\lvert A\rvert+\lvert B\rvert}$	$\frac{-\lvert A\rvert\lvert B\rvert}{(\lvert A\rvert+\lvert B\rvert)^2}$	0
Median-linkage (WPGMC)	$1/2$	$1/2$	$-1/4$	0
Ward	$\frac{\lvert A\rvert+\lvert C\rvert}{\lvert A\rvert+\lvert B\rvert+\lvert C\rvert}$	$\frac{\lvert B\rvert+\lvert C\rvert}{\lvert A\rvert+\lvert B\rvert+\lvert C\rvert}$	$\frac{-\lvert C\rvert}{\lvert A\rvert+\lvert B\rvert+\lvert C\rvert}$	0

MiniMax linkage: cannot be computed with Lance-Williams updates,
but we need to find the best cluster representative (in $O(n^2)$).

Example: AGNES

Example with complete linkage (= maximum of the distances):

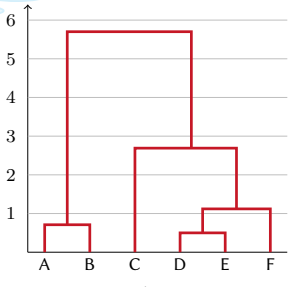
We may need to merge non-adjacent rows!



Scatter plot

	A	B	C	D	E	F
A	0	0.71	5	2.92	2.5	3.54
B	0.71	0	5.70	3.61	3.20	4.24
C	5	5.70	0	2.55	2.69	1.58
D	2.92	3.61	2.55	0	0.5	1
E	2.5	3.20	2.69	0.5	0	1.12
F	3.54	4.24	1.58	1	1.12	0

Distance matrix



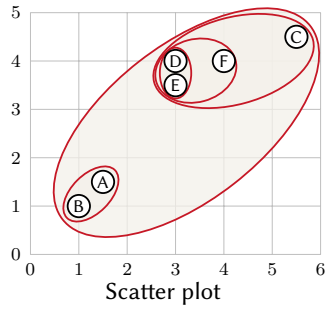
Dendrogram

We don't know the optimum label positions in advance

Example: AGNES /2

In this very simple example, single and complete linkage are very similar

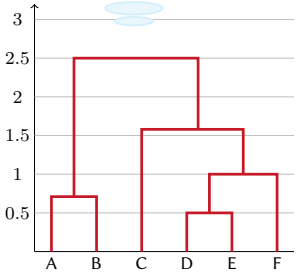
Example with single linkage (= minimum of the distances):



Scatter plot

	A	B	C	D	E	F
A	0	0.71	5	2.92	2.5	3.54
B	0.71	0	5.70	3.61	3.20	4.24
C	5	5.70	0	2.55	2.69	1.58
D	2.92	3.61	2.55	0	0.5	1
E	2.5	3.20	2.69	0.5	0	1.12
F	3.54	4.24	1.58	1	1.12	0

Distance matrix



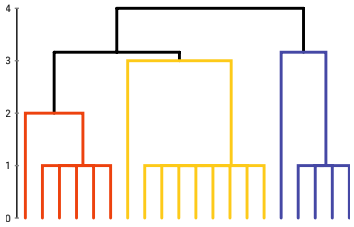
Dendrogram

Same clusters in this example, but this is usually *not* the case.

Extracting Clusters from a Dendrogram

At this point, we have the dendrogram – but not yet "clusters".
Various strategies have been discussed:

- Visually inspect the dendrogram, choose interesting branches
- Stop when k clusters remain (may be necessary to ignore noise [Sch+17b])
- Stop at a certain distance (e.g., maximum cluster diameter with complete-link)
- Change in cluster sizes or density [Cam+13]
- Constraints satisfied (semi-supervised) [Pou+14]:
  Certain objects are labeled as "must" or "should not" be in the same clusters.



Complexity of Hierarchical Clustering

Complexity analysis of AGNES:

1. Computing the distance matrix: $\mathcal{O}(n^2)$ time and memory.
2. Finding the minimum distance / maximum similarity: $\mathcal{O}(n^2) \cdot i$
3. Updating the matrix: $\mathcal{O}(n) \cdot i$ (with Lance-Williams) or $\mathcal{O}(n^2) \cdot i$
4. Number of iterations: $i = \mathcal{O}(n)$

Total: $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ memory!

Better algorithms can run in guaranteed $O(n^2)$ time [Sib73; Def77], with priority queues we get $O(n^2 \log n)$ [Mur83; DE84; Epp98; Mül11], or "usually n^2" time with caching [And73].

Instead of *agglomerative* (bottom-up), we can also begin with one cluster, and divide it (DIANA). But there are $O(2^n)$ splits – need heuristics (i.e., other clustering algorithms) to split.

- Hierarchical clustering does not scale to large data, code optimization matters [KSZ16].

Benefits and Limitations of HAC

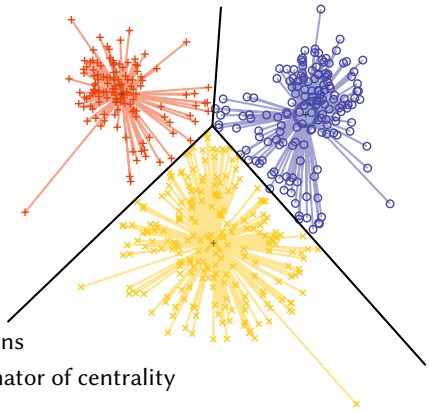
Benefits:

- Very general: any distance / similarity (for text: cosine!)
- Easy to understand and interpret
- Hierarchical result
- Dendrogram visualization often useful (for small data)
- Number of clusters does not need to be fixed beforehand
- Many variants

Limitations:

- Scalability is the main problem (in particular, $\mathcal{O}(n^2)$ memory)
- In many cases, users want a flat partitioning
- Unbalanced cluster sizes (i.e., number of points)
- Outliers

k-means Clustering



The k-means problem:

- Divide data into k subsets (k is a parameter)
- Subsets represented by their *arithmetic mean* μ_C
- Optimize the *least squared* error

$$\text{SSQ} := \sum_C \sum_d \sum_{x_i \in C} (x_{i,d} - \mu_{C,d})^2$$

Important properties:

- Squared errors put more weight on larger deviations
- Arithmetic mean is the maximum likelihood estimator of centrality
- Data is split into Voronoi cells

- k-means is a good choice, if we have k signals and normal distributed measurement error

Suggested reading: the history of least squares estimation (Legendre, Gauss):
`https://en.wikipedia.org/wiki/Least_squares#History`

The Sum of Squares Objective

We can rearrange these sums because of communtativity

The sum-of-squares objective:

$$\text{SSQ} := \underbrace{\sum\nolimits_C}_{\text{every cluster}} \times \underbrace{\sum\nolimits_d}_{\text{every dimension}} \times \underbrace{\sum\nolimits_{x_i \in C}}_{\text{every point}} \underbrace{(x_{i,d} - \mu_{C,d})^2}_{\text{squared deviation from mean}}$$

For every cluster C and dimension d, the arithmetic mean minimizes

$$\sum\nolimits_{x_i \in C} (x_{i,d} - \mu_{C,d})^2 \quad \text{is minimized by} \quad \mu_{C,d} = \tfrac{1}{|C|} \sum\nolimits_{x_i \in C} x_{i,d}$$

Assigning every point x_i to its least-squares closest cluster C usually[2] reduces SSQ, too.

Note: sum of squares $\equiv$ squared Euclidean distance:

$$\sum\nolimits_d (x_{i,d} - \mu_{C,d})^2 \equiv \|x_i - \mu_C\|^2 \equiv d^2_{\text{Euclidean}}(x_i, \mu_C)$$

We can therefore say that every point is assigned the "closest" cluster, *but* we cannot use arbitrary other distance functions in k-means (because the arithmetic mean only minimizes SSQ).

[2] This is *not* always optimal: the change in mean can increase the SSQ of the new cluster. But this difference is commonly ignored in algorithms and textbooks.

The Standard Algorithm (Lloyd's Algorithm)

The standard algorithm for k-means [Ste56; For65; Llo82]:

Algorithm: Lloyd-Forgy algorithm for k-means

1 Choose k points randomly as initial centers
2 **repeat**
3 Assign every point to the least-squares closest center
4 **stop if** no cluster assignment changed
5 Update the centers with the arithmetic mean

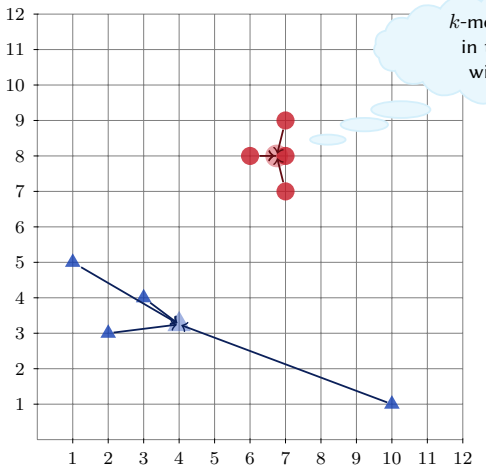
Lines 3 and 5 both improve SSQ

This is *not* the most efficient algorithm (despite everybody teaching this variant).
ELKI [Sch+15] contains ≈ 10 variants (e.g., Sort-Means [Phi02]; benchmarks in [KSZ16]).
There is little reason to still use this variant in practise!

The name k-means was first used by MacQueen for a slightly different algorithm [Mac67].

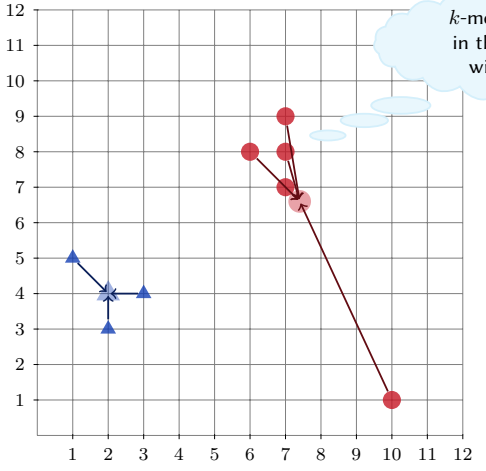
k-means was invented several times, and has an interesting history [Boc07].

Example: k-means Clustering with Lloyd's algorithm /1



k-means has converged in the third iteration with SSQ = 61.5

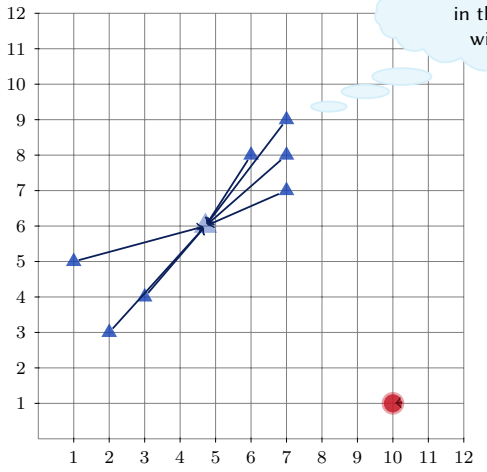
Example: k-means Clustering with Lloyd's algorithm /2



k-means has converged in the second iteration with SSQ = 54.4

Result with different starting centroids.

Example: k-means Clustering with Lloyd's algorithm



Result with different starting centroids.

Non-determinism & non-optimality

Most k-means algorithms

- do not guarantee to find the *global optimum* (would be NP-hard – too expensive)
- give different local optima,[3] depending on the starting point

In practical use:

- data is never exact, or complete
- the "optimum"[4] result is not necessarily the most *useful*

- Usually, we gain little by finding the true optimum
- It is usually good enough to **try multiple random initializations** and keep the "best"[4]

[3] More precisely: static point. The standard algorithm may fail to even find a local minimum [HW79].
[4] Least squares, i.e., lowest SSQ – this does not mean it will actually give the most insight.

Initialization

Ideally, each initial center is from a different cluster.

But the chance of randomly drawing one centroid from each cluster is small:

- Assuming (for simplification) that all clusters are the same size n, then

$$p = \frac{\text{number of ways to select one centroid from each cluster}}{\text{number of ways to select } k \text{ centroids}} = \frac{k!n^k}{(kn)^k} = \frac{k!}{k^k}$$

- For example, if $k = 10$, then probability $= 10!/10^{10} = 0.00036$
- We can run k-means multiple times, and keep the best solution.
  But we still have a low chance of getting one center from each cluster!
  For $k = 10$ and 100 tries: $p' = 0.0353$. Need ≈ 2000 attempts for 50% chance.

On the other hand, even if we choose suboptimal initial centroids,
the algorithm still *can* converge to a good solution.

And even with one initial centroid from each cluster, it can converge to a bad solution!

Initialization /2

Several strategies for initializing k-means exist:

- Initial centers given by domain expert
- Randomly assign points to partitions $1 \dots k$ (*not* very good) [For65]
- Randomly generate k centers from a uniform distribution (*not* very good)
- Randomly choose k data points as centers (uniform from the data) [For65]
- First k data points [Mac67, incremental k-means]
- Choose a point randomly, then use always the farthest point to get k initial points
  (often quite well; initial centers are often outliers, and gives similar results when run often)
- Weighting points by their distance [AV07, K-means++]
  Points are chosen randomly with $p \propto \min_c \|x_i - c\|^2$ (c = all current seeds)
- Run a few k-means iterations on a sample, then use centroids from the sample result.
- ...

Complexity of k-means Clustering

In the standard algorithm:

1. Initialization is usually cheap, $O(k)$ (k-means++: $O(N \cdot k \cdot d)$ [AV07])
2. Reassignment is $O(N \cdot k \cdot d) \cdot i$
3. Mean computation is $O(N \cdot d) \cdot i$
4. Number of iterations $i \in 2^{\Omega(\sqrt{N})}$ [AV06] (but fortunately, usually $i \ll N$)
5. Total: $O(N \cdot k \cdot d \cdot i)$

Worst case is superpolynomial, but in practice the method will usually run much better than n^2.

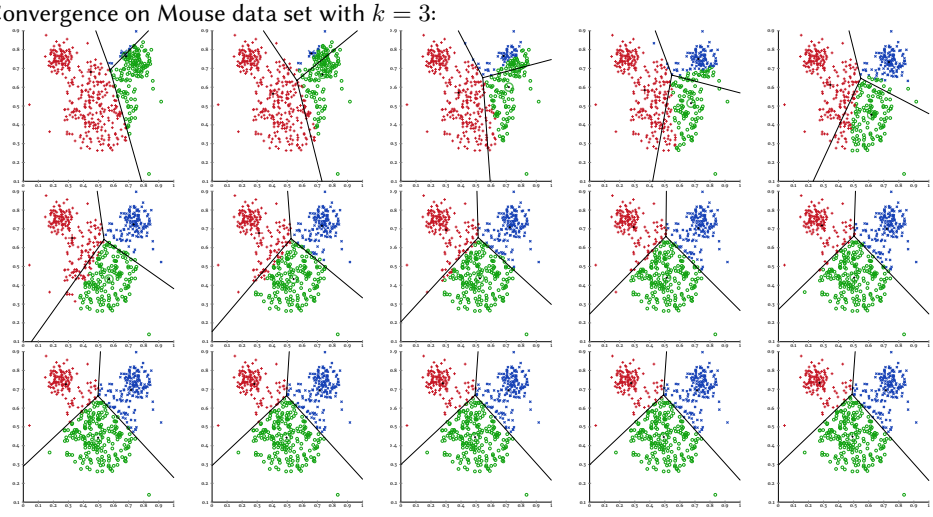
We can force a limit on the number of iterations, e.g., $i = 100$, with little loss in quality usually.

In practice, often the fastest clustering algorithm we use.

Improved algorithms primarily reduce the number of computations for reassignment,
but usually with no theoretical guarantees (so still $O(N \cdot k \cdot d) \cdot i$ worst-case)

Clusters Changes are Increasingly Incremental

Convergence on Mouse data set with $k = 3$:



Benefits and Drawbacks of k-means

Benefits:

- Very fast algorithm ($O(k \cdot d \cdot N)$, if we limit the number of iterations)
- Convenient centroid vector for every cluster
  (We can analyze this vector to get a "topic")
- Can be run multiple times to get different results

Limitations:

- Difficult to choose the number of clusters, k
- Cannot be used with arbitrary distances
- Sensitive to scaling – requires careful preprocessing
- Does not produce the same result every time
- Sensitive to outliers (squared errors emphasize outliers)
- Cluster sizes can be quite unbalanced (e.g., one-element outlier clusters)

Choosing the "Optimum" k for k-means

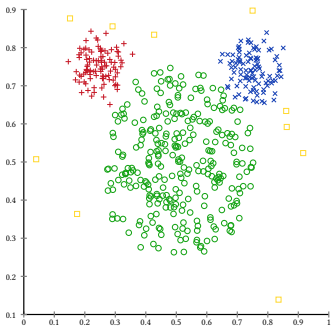
A key challenge of k-means is choosing k:

- Trivial to prove: $\mathrm{SSQ}_{\text{optimum},k} \geq \mathrm{SSQ}_{\text{optimum},k+1}$.
  - Avoid comparing SSQ for different k or different data (including normalization).
- $\mathrm{SSQ}_{k=N} = 0$ — "perfect" solution? No: useless.
- SSQ_k may exhibit an "elbow" or "knee": initially it improves fast, then much slower.
- Use alternate criteria such as Silhouette [Rou87], AIC [Aka77], BIC [Sch78; ZXF08].
  - Computing silhouette is $\mathcal{O}(n^2)$ – more expensive than k-means.
  - AIC, BIC try to reduce overfitting by penalizing model complexity (= high k).

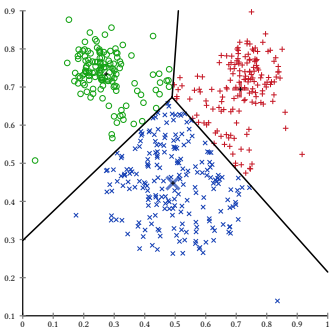
  More details will come in evaluation section.
- Nevertheless, these measures are *heuristics* – other k can be better in practice!
- Methods such as X-means [PM00] split clusters as long as a quality criterion improves.

Example: Choosing the "Optimum" k /2

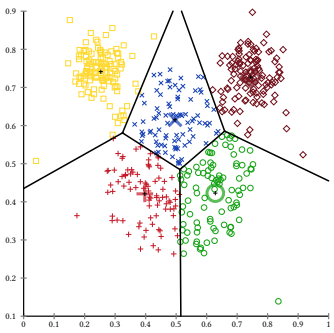
Toy "mouse" data set:



Best with $k = 3$:



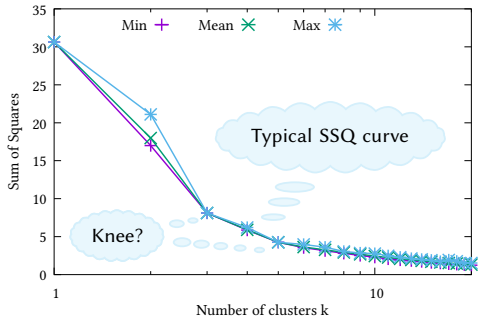
Best with $k = 5$:

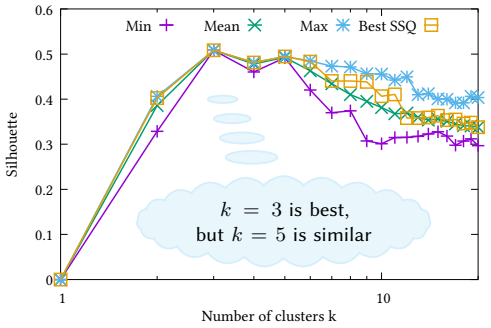


Example: Choosing the "Optimum" k /2

0.5 is not considered to be a good Silhouette

On the toy (mouse) data set:
Best results for 25 initializations, $k = 1 \ldots 20$:





All tested measures either prefer 3 or 5 clusters.

k-means and Distances

Could we use a different distance in the k-means?

$$\mathrm{SSQ}' := \sum_C \sum_{x_i \in C} \mathrm{dist}(x_i - \mu_C)$$

We can still assign every point to the closest center.

But: is the *mean* the optimum cluster center?

The mean is a least-squares estimator in $\mathbb{R}$, i.e., it minimizes $\|x_{i,d} - \mu_{C,d}\|^2$.

That does not imply the mean minimizes *other* distances!

Counter example: $1, 2, 3, 4, 10 \in \mathbb{R}$; Euclidean distance becomes $|x_i - m|$ (*no* square!)
The mean is $m = 4$. Mean distance: 2.4 The median is $m = 3$. Mean distance: 2.2
➧ the mean does not minimize *linear* Euclidean distance, Manhattan distance, …

The k-means algorithm minimizes Bregman divergences [Ban+05].
For other distances, we need to replace the "mean" in k-means.

k-means Variations for Other Distances

k-medians: instead of the mean, we use the median in each dimension. [BMS96]
For use with Manhattan norm $\|x_i - m_j\|_1$.

k-modes: use the mode instead of the mean. [Hua98]
For categorical data, using Hamming distance.

k-prototypes: mean on continuous variables, mode on categorical. [Hua98]
For mixed data, using squared Eucldiean respectively Hamming distance.

k-medoids: using the medoid (element with smallest distance sum).
For arbitrary distance functions.

Spherical k-means: use the mean, but normalized to unit length.
For cosine distance.

Gaussian Mixture Modeling: using mean and covariance.
For use with Mahalanobis distance.

k-means for Text Clustering

Cosine similarity is closely connected to squared Euclidean distance.

Spherical k-means [DM01] uses:

- Input data is normalized to have $\|x_i\| = 1$
- At each iteration, the new centers are normalized to $\mu'_C := \|\mu_C\| = 1$
- μ'_C minimizes average cosine similarity [DM01]

$$\sum_{x_i \in C} \langle x_i, \mu'_C \rangle \cong |C| - \sum_{x_i \in C} \|x_i, \mu'_C\|^2$$

- Sparse nearest-centroid computations in $O(d')$ where d' is the number of non-zero values
- Result is similar to a SVD factorization of the document-term-matrix [DM01]

Pre-processing and Post-processing

Pre-processing and post-processing commonly used with k-means:

Pre-processing:

- Scale / normalize continuous attributes to $[0; 1]$ or unit variance.
- Encode categorical attributes as binary attributes.
- Eliminate outliers

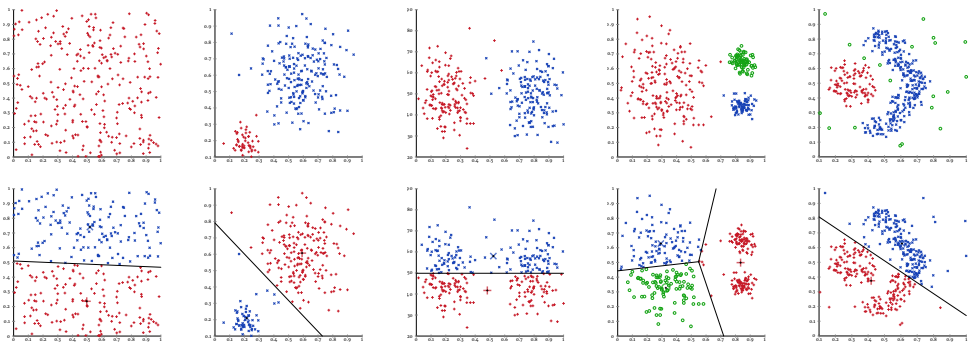
Post-processing

- Eliminate clusters with few elements (probably outliers)
- Split "loose" clusters, i.e., clusters with relatively high SSE
- Merge clusters that are "close" and that have relatively low SSE
- Can use these steps during the clustering process
  E.g., ISODATA algorithm [BH65], X-means [PM00], G-means [HE03]

Limitations of k-means

k-means has problems when clusters are of differing sizes, densities, or have non-spherical shape
k-means has problems when the data contains outliers.

Badly chosen k: Diameter: Scaling: Different densities: Cluster shape:



k-medoids Clustering

This approach tries to improve over two weaknesses of k-means:

- k-means is sensitive to outliers (because of the squared errors)
- k-means cannot be used with arbitrary distances.

Idea: The **medoid** of a set is the object with the least distance to all others.
- The most central, most "representative" object

k-medoids objective function: **absolute-error criterion**

$$\mathrm{TD} = \sum_{i=1}^{k} \sum_{x_j \in C_i} \mathrm{dist}(x_j, m_i)$$

where m_i is the medoid of cluster C_i.

As with k-means, the k-medoid problem is NP-hard.
The algorithm **Partitioning Around Medoids** (PAM) guesses a result, then uses iterative refinement, similar to k-means.

Partitioning Around Medoids

Partitioning Around Medoids (PAM, [KR90])

- choose a good initial set of medoids
- iteratively improve the clustering by doing the best (m_i, o_j) swap, where m_i is a medoid, and o_j is a non-medoid.
- if we cannot find a swap that improves TD, the algorithm has converged.
- good for small data, but not very scalable

Algorithm: Partitioning Around Medoids

PAM consists of two parts:

PAM BUILD, to find an initial clustering:

Algorithm: PAM BUILD: Find initial cluster centers

1 $m_1 \leftarrow$ point with the smallest *distance sum* TD to all other points
2 **for** i=2...k **do**
3 $\quad m_i \leftarrow$ point which *reduces* TD most
4 **return** TD, $\{m_1, \ldots, m_k\}$

This needs $O(n^2 k)$ time, and is best implemented with $O(n^2)$ memory.

We could use this to seed k-means, too. But it is usually slower than a complete k-means run.

Algorithm: Partitioning Around Medoids /2

PAM SWAP, to improve the clustering:

Algorithm: PAM SWAP: Improve the clustering

repeat
  for $m_i \in \{m_1, \ldots, m_k\}$ **do**
    for $c_j \notin \{m_1, \ldots, m_k\}$ **do**
      $\mathrm{TD}' \leftarrow \mathrm{TD}$ with c_j medoid instead of m_i
      Remember (m_i, c_j, TD') for the best TD'
  stop if the best TD' did not improve the clustering
  swap (m_i, c_j) of the best TD'
return TD, M, C

This needs $O(k(n-k)^2)$ time for each iteration i.

The authors of PAM assumed, that only few iterations will be needed, because the initial centers are supposed to be good already.

Algorithm: CLARA (Clustering Large Applications) [KR90]

Algorithm: CLARA: Clustering Large Applications

$\mathrm{TD}_{\text{best}}, M_{\text{best}}, C_{\text{best}} \leftarrow \infty, \emptyset, \emptyset$
for $i = 1 \ldots 5$ **do**
  $S_i \leftarrow$ sample of $40 + 2k$ objects
  $M_i \leftarrow$ medoids from running PAM(S_i)
  $\mathrm{TD}_i, C_i \leftarrow$ compute total deviation and cluster assignment using M_i
  if $\mathrm{TD}_i < \mathrm{TD}_{\text{best}}$ **then**
    $\mathrm{TD}_{\text{best}}, M_{\text{best}}, C_{\text{best}} \leftarrow \mathrm{TD}_i, M_i, C_i$
return $TD_{\text{best}}, M_{\text{best}}, C_{\text{best}}$

- sampling-based method: apply PAM to a random sample of whole dataset
- builds clustering from multiple random samples and returns best clustering as output
- runtime complexity: $O(ks^2 + k(n-k))$ with sample size $s \approx 40 + 2k$
- applicable to larger data sets but the result is only based on a small sample
- samples need to be "representative", medoids are only chosen from the best sample

Algorithm: CLARANS [NH02]

Algorithm: CLARANS: Clustering Large Applications based on RANdomized Search

$\mathrm{TD}_{\text{best}}, M_{\text{best}}, C_{\text{best}} \leftarrow \infty, \emptyset, \emptyset$
for $l = 1 \ldots \mathrm{numlocal}$ **do** `// Number of times to restart`
  $M_l \leftarrow$ random medoids
  $\mathrm{TD}_l, C_l \leftarrow$ compute total deviation and cluster assignment using M_l
  for $i = 1 \ldots \mathrm{maxneighbor}$ **do** `// Attempt to improve the medoids`
    $m_j, o_k \leftarrow$ randomly select a medoid m_j, and a non-medoid o_k
    $\mathrm{TD}'_l, C'_l \leftarrow$ total deviation when replacing m_j with o_k
    if $\mathrm{TD}'_l < \mathrm{TD}_l$ **then** `// swap improves the result`
      $m_j, \mathrm{TD}_l, C_l \leftarrow o_k, \mathrm{TD}'_l, C'_l$ `// accept the improvement`
      $i \leftarrow 1$ `// restart inner loop`
  if $\mathrm{TD}_l < \mathrm{TD}_{\text{best}}$ **then** `// keep overall best result`
    $\mathrm{TD}_{\text{best}}, M_{\text{best}}, C_{\text{best}} \leftarrow \mathrm{TD}_l, M_l, C_l$
return $\mathrm{TD}_{\text{best}}, M_{\text{best}}, C_{\text{best}}$

Algorithm: CLARANS [NH02] /2

Clustering Large Applications based on RANdomized Search (CLARANS) [NH02]

- considers at most *maxneighbor* many randomly selected pairs to find *one* improvement
- replace medoids as soon as we found a better candidate, rather than testing all $k \cdot (n-k)$ alternatives for the best improvement possible
- search for k "optimal" medoids is repeated *numlocal* times (c.f., restarting multiple times with k-means)
- complexity: $O(\text{numlocal} \cdot \text{maxneighbor} \cdot \text{swap} \cdot n)$
- good results only if we choose maxneighbor large enough
- in practice the typical runtime complexity of CLARANS is similar to $O(n^2)$

k-medoids, Lloyd style

We can adopt Lloyd's approach also for k-medoids:

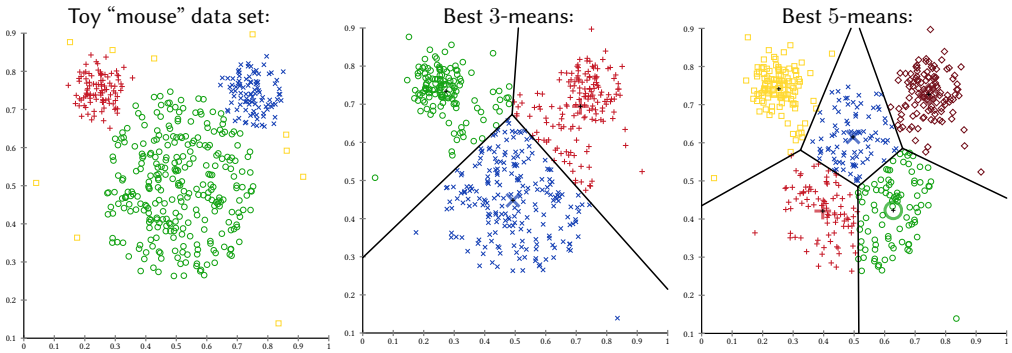
Algorithm: k-medoids with alternating optimization [Ach+12]

```
1 {m_1, ..., m_k} ← choose k random initial medoids
2 repeat
3     {C_1, ..., C_k} ← assign every point to the nearest medoid's cluster
4     stop if no cluster assignment has changed
5     foreach cluster C_i do
6         m_i ← object with the smallest distance sum to all others within C_i
7 return TD, M, C
```

- similar to k-means, we alternate between (1) optimizing the cluster assignment, and (2) choosing the best medoids.
- can update all medoids in each iteration.
- complexity: $O(k \cdot n + n^2)$ per iteration, comparable to PAM

From k-means to Gaussian EM Clustering

k-means can not handle clusters with different "radius" well.



- could we estimate mean and radius?
- model the data with multivariate Gaussian distributions

Expectation-Maximization Clustering

EM (Expectation-Maximization) is the underlying principle in Lloyd's k-means:

1. Choose initial model parameters θ
2. Expect latent variables (e.g., cluster assignment) from θ and the data.
3. Update θ to maximize the likelihood of observing the data
4. Repeat (2.)–(3.) until a stopping condition holds

Recall Lloyd's k-means:

1. Choose k centers randomly (θ: random centers)
2. Expect cluster labels by choosing the nearest center as label
3. Update cluster centers with maximum-likelihood estimation of centrality
4. Repeat (2.)–(3.) until change = 0

Objective: optimize SSQ.

Clustering Model Optimization Methods 3: 62 / 153

Expectation-Maximization Clustering

EM (Expectation-Maximization) is the underlying principle in Lloyd's k-means:

1. Choose initial model parameters θ
2. Expect latent variables (e.g., cluster assignment) from θ and the data.
3. Update θ to maximize the likelihood of observing the data
4. Repeat (2.)–(3.) until a stopping condition holds

Gaussian Mixture Modeling (GMM): [DLR77]

1. Choose k centers randomly, unit covariance, and uniform weight:
$$\theta = (\mu_1, \Sigma_1, w_1, \mu_2, \Sigma_2, w_2, \dots \mu_k, \Sigma_k, w_k)$$
2. Expect cluster labels based on Gaussian distribution density
3. Update Gaussians with mean and covariance matrix
4. Repeat (2.)–(3.) until change $< \epsilon$

Objective: Optimize log-likelihood: $\log \mathcal{L}(\theta) := \sum_x \log P(x \mid \theta)$

Gaussian Mixture Modeling & EM

The multivariate Gaussian density with center μ_i and covariance matrix Σ_i is:
$$P(x \mid C_i) = \mathrm{pdf}(x, \mu_i, \Sigma_i) := \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} \cdot e^{-\frac{1}{2}\left((x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)\right)}$$

For the **Expectation Step**, we use Bayes' rule:
$$P(C_i \mid x) = \frac{\mathrm{pdf}(x, \mu_i, \Sigma_i) P(C_i)}{P(x)}$$

$P(C_i \mid x)$ is the relative "responsibility" of C_i for point x
$P(C_i \mid x)$ proportional to $w_i \cdot \mathrm{pdf}_i$
$P(C_i \mid x) \propto w_i \mathrm{pdf}(p, \mu_i, \Sigma_i)$

Using $P(C_i) = w_i$ and the law of total probability:
$$P(C_i \mid x) = \frac{w_i \mathrm{pdf}(p, \mu_i, \Sigma_i)}{\sum_j w_j \mathrm{pdf}(p, \mu_j, \Sigma_j)}$$

For the **Maximization Step**:
Use weighted mean and weighted covariance to recompute cluster model.

weighted $\overline{X}$ / $\overline{S}_{XY}$
i cluster, j, k dimensions

$$\begin{aligned}
\mu_{i,j} &= \frac{1}{\sum_x P(C_i \mid x)} \sum_x P(C_i \mid x) x_j \\
\Sigma_{i,j,k} &= \frac{1}{\sum_x P(C_i \mid x)} \sum_x P(C_i \mid x)(x_j - \mu_j)(x_k - \mu_k) \\
w_i &= P(C_i) = \frac{1}{|X|} \sum_x P(C_i \mid x)
\end{aligned}$$

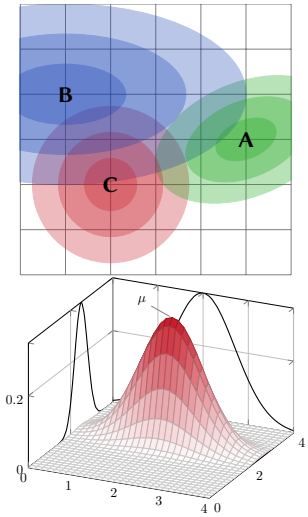
Fitting Multiple Gaussian Distributions

Probability density function of a multivariate Gaussian:

$$\mathrm{pdf}(x, \mu, \Sigma) := \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \cdot e^{-\frac{1}{2}\left((x-\mu)^T \Sigma^{-1} (x-\mu)\right)}$$

If we constrain Σ we can control the cluster shape:

- We always want symmetric and positive semi-definite
- Σ covariance matrix: rotated ellipsoid (A)
- Σ diagonal ("variance matrix"): ellipsoid (B)
- Σ scaled unit matrix: spherical (C)
- Same Σ for all clusters, or different Σ_i each



Understanding the Gaussian Density

Multivariate normal distribution:

$$\mathrm{pdf}(x, \mu, \Sigma) := \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \cdot e^{-\frac{1}{2}\left((x-\mu)^T \Sigma^{-1} (x-\mu)\right)}$$

1-dimensional normal distribution:

$$\mathrm{pdf}(x, \mu, \sigma) := \frac{1}{\sqrt{(2\pi)\sigma^2}} \cdot e^{-\frac{1}{2}\left((x-\mu)\sigma^{-2}(x-\mu)\right)}$$

Normalization (to a total volume of 1) and squared deviation from center
Compare this to Mahalanobis distance:

$$d_{\mathit{Mahalanobis}}(x, \mu, \Sigma)^2 := (x-\mu)^T \Sigma^{-1} (x-\mu)$$

- Σ/Σ^{-1} plays a central role here, the remainder is squared Euclidean distance!

Decomposing the Inverse Covariance Matrix – Σ^{-1}

Covariance matrixes are symmetric, non-negative on the diagonal, and can be inverted.
(This may need a robust numerical implementation; ignore constant attributes)

We can use the eigendecomposition to factorize Σ into:

$$\Sigma = V \Lambda V^{-1} \quad \text{and, therefore,} \quad \Sigma^{-1} = V \Lambda^{-1} V^{-1}$$

where V is *orthonormal* and contains the eigenvectors $\cong$ rotation
and Λ is *diagonal* and contains the eigenvalues $\cong$ squared scaling.

(You may want to refresh your linear algebra knowledge.)

Decomposing the Inverse Covariance Matrix – Σ^{-1} /2

Based on this decomposition $\Sigma = V\Lambda V^{-1}$, let λ_i be the diagonal entries of Λ.

Build Ω using $\omega_i = \sqrt{\lambda_i}^{-1} = \lambda_i^{-\frac{1}{2}}$. Then $\Omega^T = \Omega$ (because it is diagonal) and $\Lambda^{-1} = \Omega^T\Omega$.

$$\begin{aligned}\Sigma^{-1} = V\Lambda^{-1}V^{-1} &= V\Omega^T\Omega V^T = (\Omega V^T)^T\Omega V^T \\ d^2_{\text{Mahalanobis}} &= (x-\mu)^T\Sigma^{-1}(x-\mu) \\ &= (x-\mu)^T(\Omega V^T)^T\Omega V^T(x-\mu) \\ &= \left\langle \Omega V^T(x-\mu), \Omega V^T(x-\mu)\right\rangle \\ &= \left\|\Omega V^T(x-\mu)\right\|^2\end{aligned}$$

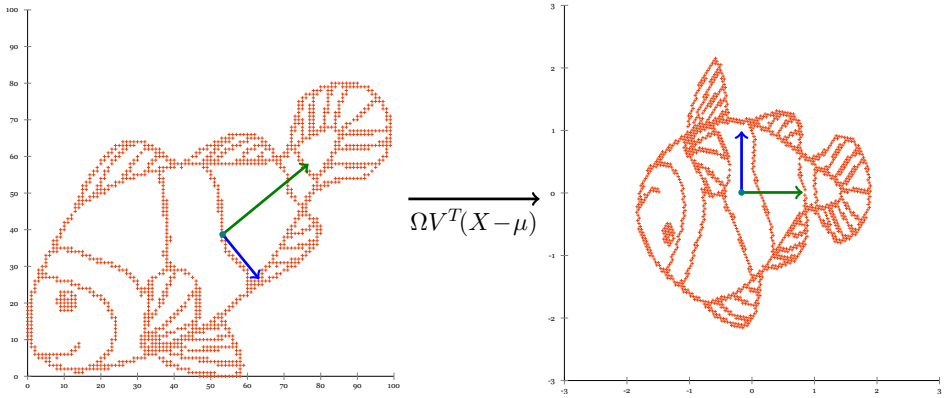
- Mahalanobis $\approx$ Euclidean distance in a rotated, scaled space.

V^T: rotation that rotates eigenvectors to the coordinate axes
Ω: scale coordinates such that the eigenvalues become unit

After this transformation, the new covariance matrix is the unit matrix!

Decomposing the Inverse Covariance Matrix – Σ^{-1} /2

Example of projecting the data with ΩV^T:



Algorithm: EM Clustering

The generic EM clustering algorithm is:

Algorithm: EM Clustering

1 $\theta \leftarrow$ choose initial model
2 $\log \mathcal{L} \leftarrow$ log likelihood of initial model
3 **repeat**
4 $\quad P(C_i \mid x) \leftarrow$ expect probabilities for all x `// Expectation Step`
5 $\quad \theta \leftarrow$ maximize new model parameters `// Maximization Step`
6 $\quad \log \mathcal{L}_{\text{previous}} \leftarrow \log \mathcal{L}$
7 $\quad \log \mathcal{L} \leftarrow$ log likelihood of new model θ `// Model Evaluation`
8 $\quad$ **stop if** $|\log \mathcal{L}_{\text{previous}} - \log \mathcal{L}| < \varepsilon$

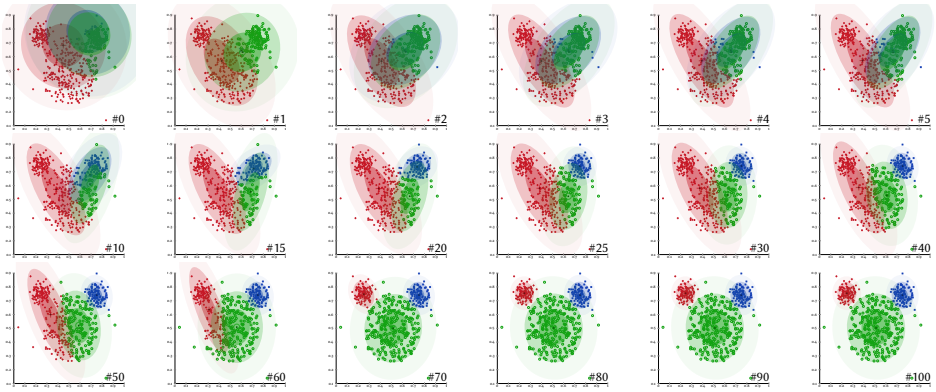
The log likelihood $\log \mathcal{L} := \sum_x \log P(x \mid \theta) = \sum_x \log\left(\sum_i w_i P(x \mid C_i)\right)$
is improved by both the E-step, and the M-step.

For Gaussian Mixture modeling, $\theta = (\mu_1, \Sigma_1, w_1, \mu_2, \Sigma_2, w_2, \ldots, \mu_k, \Sigma_k, w_k)$

In contrast to k-means, we *need* to use a stopping threshold, because the change will not become 0.

Gaussian Mixture Modeling Example

Clustering mouse data set with $k = 3$:



GMM with the EM Algorithm: Discussion

Complexity of Gaussian Mixture Modeling:

- $O(n \cdot k \cdot d + k \cdot d^3)$ for each iteration
  (with a simpler diagonal Σ model: $O(n \cdot k \cdot d)$)
- in general, number of iterations is quite high
- numerical issues require a careful implementation
- for covariance estimation, we need a lot of data. Should be $n > kd^2$.

As it is the case for k-means and k-medoid, result and runtime very much depend on

- initial choice of model parameters θ
- in particular, a good choice of parameter k
- data needs to contain Gaussians

Modification for a partitioning of the data points into k disjoint clusters:
every point is only assigned to the cluster to which it belongs with the highest probability

Clustering Other Data with EM

We cannot use *Gaussian* EM on text:

- text is not Gaussian distributed.
- text is discrete and sparse, Gaussians are continuous.
- covariance matrixes have $\mathcal{O}(d^2)$ entries:
  - memory requirements (text has a very high dimensionality d)
  - data requirements (to reliably estimate the parameters, we need very many data points)
  - matrix inversion is even $\mathcal{O}(d^3)$

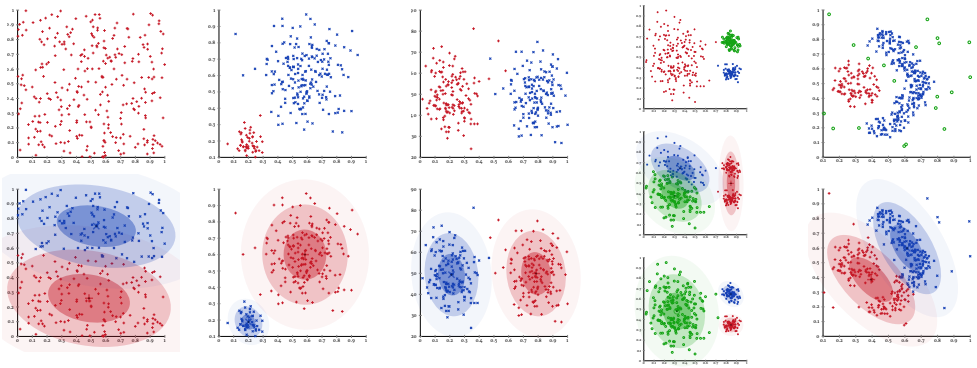
But the general EM principle can be used with *other distributions*.
For example, we can use a mixture of Bernoulli or multinomial distributions.

- PLSI/PLSA uses multinomial distributions [Hof99].
- clickstreams can be modeled with a mixture of Markov models [Cad+03; YH02]
- fuzzy c-means is a "soft" k-means (but without a parametric model) [Dun73; Bez81]
- many more

Limitations of Gaussian Mixture Modeling

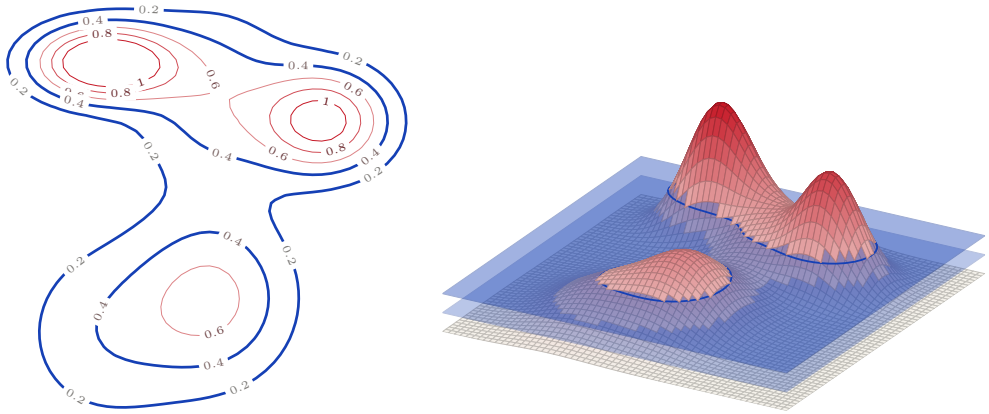
Gaussian Mixture Modeling works much better than k means *if* the data consists of Gaussians.
But it otherwise shares many of the limitations.

Badly chosen k: Diameter: Scaling: Different densities: Cluster shape:



Density-based Clustering: Core Idea

Density connected regions form clusters:



Density-based Clustering: Foundations

Idea

- clusters are regions *with high object density*
- clusters are *separated* by regions with lower density
- clusters can be of arbitrary shape (not just spherical)

Core assumptions for density-based clusters

- objects of the cluster have *"high" density* (e.g., above a specific threshold)
- a cluster consists of a *connected* region of high density

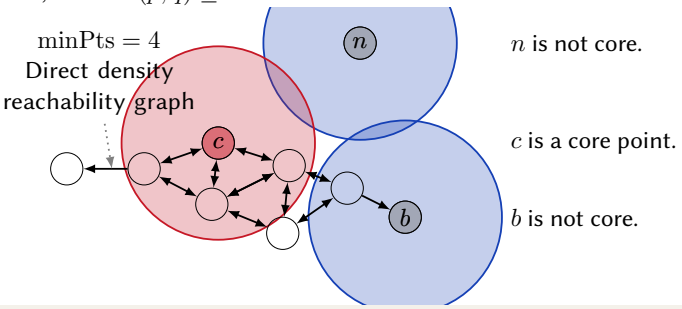
Density-based Clustering: Foundations /2

Dense objects ("core objects")

- parameters $\varepsilon > 0$ and $\mathrm{minPts} \geq 1$ specify a "density threshold"
- an object $p \in O$ (O being set of objects) is a **core object** if

$$|N_\varepsilon(p)| \geq \mathrm{minPts} \quad \text{where } N_\varepsilon(p) = \{q \in O | \mathrm{dist}(p,q) \leq \varepsilon\}$$

Direct density reachability:
An object $q \in O$ is *direct-density-reachable* from p (denote as $q \triangleleft p$ or $p \triangleright q$) if and only if p is a *core* point, and $\mathrm{dist}(p,q) \leq \varepsilon$.



Density-reachability and Density-connectivity

Based on direct-density-reachability for a given ε and minPts, we define:

Density-reachability: q is *density reachable* from p ($q \triangleleft p$ or $p \triangleright q$) if and only if there exists a chain of *direct-density-reachable* objects

$$p = o_1 \triangleright o_2 \triangleright \ldots \triangleright o_k = q$$

$\triangleright$ is the transitive closure of $\triangleright$

Density-connectivity: p and q are *density connected* ($p \mathrel{<\!\triangleright} q$) if and only if there exists an object r such that both p and q are *density reachable* from r.

$$p \triangleleft r \triangleright q$$

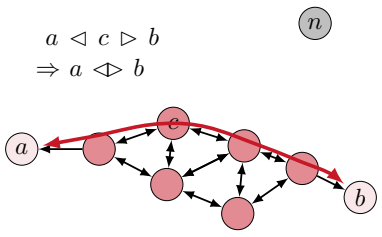
p and q can both be border points, all others must be core

We use the following notation:

$p \trianglerighteq q$	q is *direct* density reachable from p
$p \triangleright q$	q is density reachable from p
$p \bowtie q$	p and q are mutually *direct*-density-reachable
$p \bowtie q$	p and q are mutually density-reachable
$p \mathrel{<\!\triangleright} q$	p and q are density-connected

Density-reachability

$$a \triangleleft c \triangleright b$$
$$\Rightarrow a \mathrel{<\!\triangleright} b$$



Density-based cluster:
A non-empty subset $C \subseteq O$ is a *density-based cluster*, given ε and minPts, if:

(1) Maximality: $\forall p, q \in O$: if $p \in C$ and q is density-reachable from p, then $q \in C$

$$\forall p, q \in O: \quad p \in C \wedge p \triangleright q \Rightarrow q \in C$$

(2) Connectivity: $\forall p, q \in C$: p and q are density-connected

$$\forall p, q \in C: \quad p \mathrel{<\!\triangleright} q$$

Clustering Approach

Border and noise points:

- A point b that is *not* a core point, but that is density-reachable is called a *border point*.
- A point n that is not density-reachable is called a *noise point*.

$$\text{Type}(p) = \begin{cases} \text{Core} & \text{if } |N(p,\varepsilon)| \geq \text{minPts} \\ \text{Border} & \text{if } |N(p,\varepsilon)| < \text{minPts} \ \wedge \ \exists_q \, q \rhd p \\ \text{Noise} & \text{if } |N(p,\varepsilon)| < \text{minPts} \ \wedge \ \nexists_q \, q \rhd p \end{cases}$$

Density-based clustering:

- a *density-based clustering* $\mathbf{C}$ of a set of objects O, given parameters ε and minPts, is a *complete* set of density-based clusters
- $\text{Noise}_{\mathbf{C}}$ is the set of all objects in O that do not belong to any density-based cluster $C_i \in \mathbf{C}$, i.e., $\text{Noise}_{\mathbf{C}} = O \setminus (C_1 \cup \cdots \cup C_k)$

Abstract DBSCAN Algorithm

Density-Based Clustering of Applications with Noise:

Algorithm: Abstract DBSCAN algorithm [Sch+17a]

```
1 Compute neighbors of each point and identify core points   // Identify core points
2 Join neighboring core points into clusters                 // Assign core points
3 foreach non-core point do
4     Add to a neighboring core point if possible            // Assign border points
5     Otherwise, add to noise                                // Assign noise points
```

This abstract algorithm finds

- connected components of core points such that $\forall_{p,q} p \bowtie q$ (lines 1 + 2)
- + their "border point" neighbors
- remaining points are noise

The original DBSCAN algorithm [Est+96] is a database oriented, *iterative* algorithm to solve this.

The successor algorithm HDBSCAN* [CMS13] only uses $p \bowtie q$, all non-core points are noise.

DBSCAN Algorithm

DBSCAN KDD'96 high-level view:

1. begin with any yet unlabeled *core* point p (if it is not core, label it as "noise" and repeat)
2. begin a new cluster C_i, and continue with its neighbors $N = N(p, \varepsilon)$
3. for each neighbor $n \in N$:
   - 3.1 if n is already labeled by some cluster,[5] continue with the next n
   - 3.2 label n as cluster C_i
   - 3.3 if n *core*, $N \leftarrow N \cup N(n, \varepsilon)$
4. when we have processed all (transitive) neighbors N, the cluster is complete
5. continue until there is no unlabeled point

Since the initial point is core, for all its neighbors n we have $p \trianglerighteq n$.

For each n that is core, and each neighbor n', we have $n \trianglerighteq n'$, and therefore $p \rhd n'$.

- it is easy to see that this constructs density-connected clusters according to the definition.

[5] Non-core points can be "noise", or can be reachable from more than one cluster. In this case we only keep one label, although we could allow multi-assignment.

DBSCAN Algorithm /2

Algorithm: Pseudocode of original sequential DBSCAN algorithm [Est+96; Sch+17a]

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 Algorithm /3

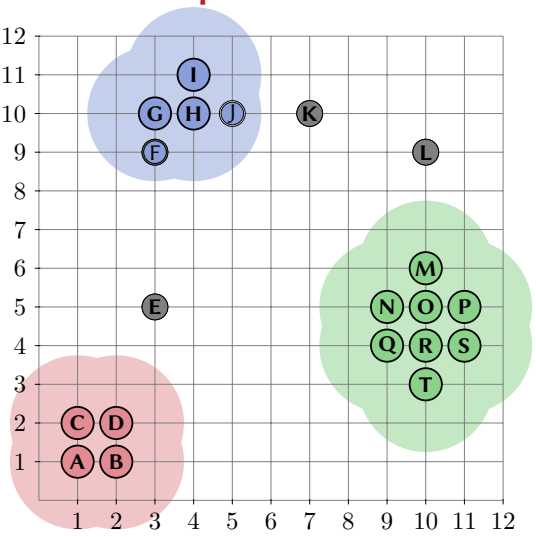
Properties of the algorithm

- on core points and noise points, the result is exactly according to the definitions
- border points are only in one cluster, from which they are reachable
  (This could trivially be allowed, but is not desirable in practise!)
- complexity: $O(n \cdot T)$ with T the complexity to find the ε-neighbors
  In many applications $T \approx \log n$ with indexing, but this cannot be guaranteed.
  Typical indexed behavior then is $n \log n$, but worst-case is $O(n^2)$ distance computations.

Challenges

- how to choose ε and minPts?
- which distance function to use?
- which index for acceleration?
- how to evaluate the result?

DBSCAN example



$\varepsilon = 1.75$

$\mathrm{minPts} = 4$

Seeds

Choosing DBSCAN parameters

Choosing minPts:

- usually easier to choose
- heuristic: $2\times$dimensionality [Est+96]
- choose larger values for large and noisy data sets [Sch+17a]

Choosing ε:

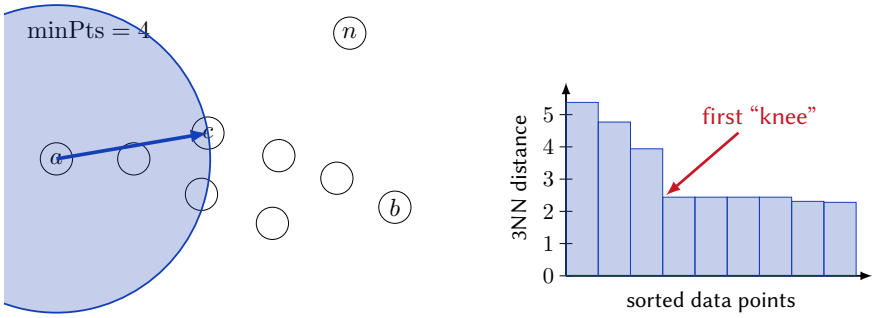
- too large clusters: reduce radius ε [Sch+17a]
- too much noise: increase radius ε [Sch+17a]
- first guess: based on distances to the $(\mathrm{minPts}-1)$-nearest neighbor [Est+96]

Detecting bad parameters: [Sch+17a]

- ε range query results are too large $\Rightarrow$ slow execution
- almost all points are in the same cluster (largest cluster should be $< 20\%$ to $< 50\%$ usually)
- too much noise (should usually be $\ll 1\%$ to $< 30\%$, depending on the application)

Choosing DBSCAN parameters /2

Example: $\mathrm{minPts} = 4$, therefore plot 3-nearest-neighbor distances:



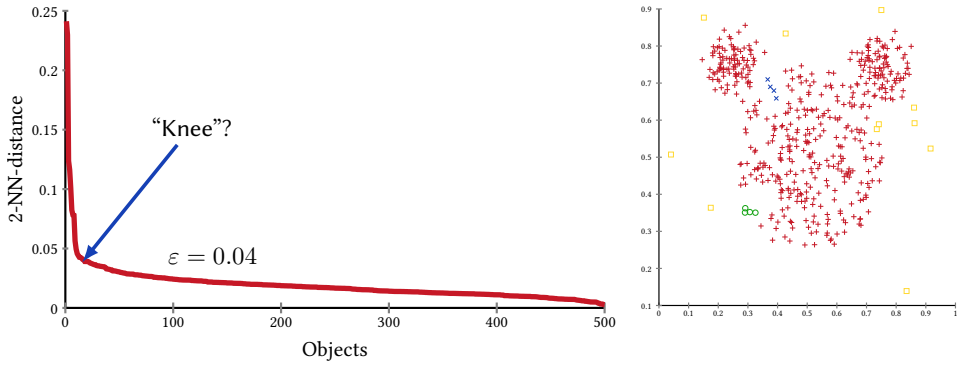
Sort the distances, look for a knee, or choose a small quantile.
$\Rightarrow$ Suggested parameter for $\mathrm{minPts} = 4$ is $\varepsilon \approx 2.5$

With an index, this takes about $n \cdot \log n$ time, $O(n^2)$ without.

On large data, we can query only a subset to make this faster.

Choosing DBSCAN parameters /3

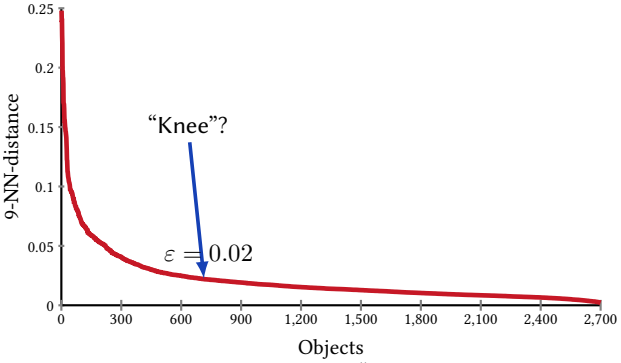
Heuristics on the mouse data set.

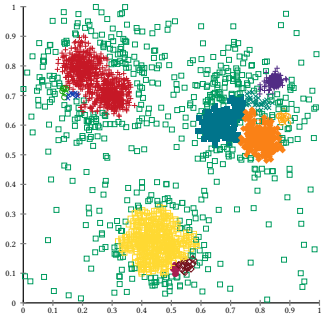


- only a *heuristic* — there are no "correct" parameters (in particular with hierarchies)
- on Gaussian distributions with different densities, we often get a smooth curve
- sometimes, we see multiple "knees", sometimes none

Choosing DBSCAN parameters /3

Heuristics on the nested clusters data set.





- only a *heuristic* — there are no "correct" parameters (in particular with hierarchies)
- on Gaussian distributions with different densities, we often get a smooth curve
- sometimes, we see multiple "knees", sometimes none

Generalized Density-based Clustering [San+98]

Motivation

- traditional clustering approaches are designed for point coordinate objects
- apply clustering to other types of data
  - geospatial data such as polygons
  - data with a notion of similarity, but not a distance
  - graph and network data
  - pixel data
  - …

Generalize the notion of density in DBSCAN:

ε threshold	$d(p,q) \leq \varepsilon$	$\rightarrow$ Neighbor predicate	$\mathrm{NPred}(p,q)$
ε range query	$N(p) := \{q \mid d(p,q) \leq \varepsilon\}$	$\rightarrow$ Neighborhood	$N(p) := \{q \mid \mathrm{NPred}(p,q)\}$
minPts threshold	$\lvert N(o)\rvert \geq \mathrm{minPts}$	$\rightarrow$ Core predicate	$\mathrm{IsCore}(N(o))$

Example neighbor predicates: polygon overlap, pixel adjacency, similarity, …
Example core predicates: count, total polygon area, homogenity, …

Accelerated DBSCAN Variations

Methods for finding the same result, but faster:

Grid-based Accelerated DBSCAN [MM08]

- Partition the data into grid cells, overlapping by ε
- Run DBSCAN on each cell
- Merge the results into a single clustering
- For Minkowski L_p distances

Anytime Density-Based Clustering (AnyDBC) [MAS16]

- avoids doing a ε-range-query for every point
- cover the data with a non-transitive set of "primitive clusters"
- infer "core" property also by counting how often a point is found
- query only points that may cause primitive clusters to merge
- some points will remain "maybe-core" (border or core)

Improved DBSCAN Variations

Methods that aim at improving the results or usability of DBSCAN:

Locally Scaled Density Based Clustering (LSDBC) [BY07]

- estimate density of each point
- always continue DBSCAN with the most dense unlabeled point
- stop expanding when the density drops below $\alpha\cdot$ density of the first point
- $\Rightarrow$ can find clusters of different densities

Hierarchical DBSCAN* (HDBSCAN*) [CMS13]

- clusters consists only of core points, no border points ($\Rightarrow$ nicer theory; the star * in the name)
- perform a hierarchical agglomerative clustering based on density reachability
- runtime $O(n^2)$, adds the challenge of extracting clusters from the dendrogram
- inspired by OPTICS / continuation of OPTICS (next slides ...)

Many, many, more ... in 2017, people still publish new DBSCAN variations

Density-based Hierarchical Clustering

The ***k*-nearest neighbor distance** $k\text{-dist}(p)$ of a point p is defined as the unique distance with:

(i) for at least k objects $o \in D, o \neq p$: $\text{dist}(p,o) \leq k\text{-dist}(p)$

(ii) for at most $k-1$ objects $o \in D, o \neq p$: $\text{dist}(p,o) < k\text{-dist}(p)$

The ***k*-nearest neighbors** $k\text{NN}(p)$ of a point p are defined as:

$$k\text{NN}(p) = \{o \in D \mid o \neq p \wedge \text{dist}(o,p) \leq k\text{-dist}(p)\}$$

Note: A point p is a DBSCAN core point if and only if its $(\text{minPts}-1)\text{-dist}(p) \leq \varepsilon$[6]

Note: A point p is a DBSCAN core point for all $\varepsilon \geq (\text{minPts}-1)\text{-dist}(p)$
$\Rightarrow$ monotonicity: any DBSCAN cluster at ε is a subset of some DBSCAN cluster at $\varepsilon' > \varepsilon$.

- Idea: process points based on their $(\text{minPts}-1)\text{-dist}$.

Can we get DBSCAN results for all ε at once?

[6] $(\text{minPts}-1)\text{-dist}$ is k-dist with $k = \text{minPts}-1$

Density-based Hierarchical Clustering /2

For performance reasons, OPTICS uses ε-range queries to find neighbors (discussed later).

Core-distance of an object *p*

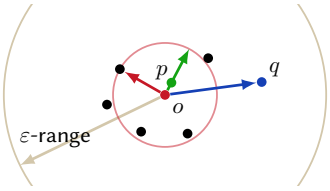
$$\text{CoreDist}_{\varepsilon,\text{minPts}}(p) = \begin{cases} (\text{minPts}-1)\text{-dist}(p) & \text{if} |N_\varepsilon(p)| \geq \text{minPts} \\ \infty & \text{otherwise} \end{cases}$$

with $(\text{minPts}-1)\text{-dist}(p)$ being the smallest distance for which p is a core object.

Reachability-distance[7] of object *p* from object *o*

$$\text{ReachDist}_{\varepsilon,\text{minPts}}(p \leftarrow o) = \max\{\text{CoreDist}_{\varepsilon,\text{minPts}}(o), \text{dist}(o,p)\}$$

$\text{minPts} = 5$



ε-query radius
$\text{CoreDist}(o)$
$\text{ReachDist}(q \leftarrow o)$
$\text{ReachDist}(p \leftarrow o)$

[7] This is not a distance by our definitions, because it is *not symmetric*! We use $p \leftarrow o$ to emphasize this.

OPTICS Clustering [Ank+99]

OPTICS high-level view: Process points in a priority queue ordered by lowest reachability.

1. get the minimum-reachability point p from the priority queue
   (if the queue is empty, choose any unprocessed point with reachability set to ∞.)
2. query the neighbors $N_\varepsilon(p)$ and determine $\mathrm{CoreDist}_{\varepsilon,\mathrm{minPts}}$
3. add p to the output, with its current minimum reachability and its core distance
4. mark p as processed
5. add each unprocessed neighbor n to the queue, using $\mathrm{ReachDist}_{\varepsilon,\mathrm{minPts}}(n \leftarrow p)$,
   (but only improving the reachability if the point is already in the queue)
6. repeat, until all points are processed

Note: the reachability of a *processed* point is never updated.

Complexity: assuming queries are in $\log n$, we get $n \log n$. Worst case: $O(n^2)$

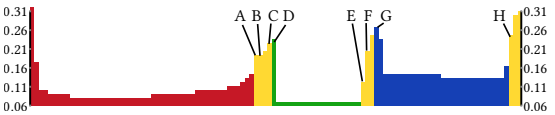
Cluster Order

OPTICS does *not* directly produce a (hierarchical) clustering but a "cluster ordering".

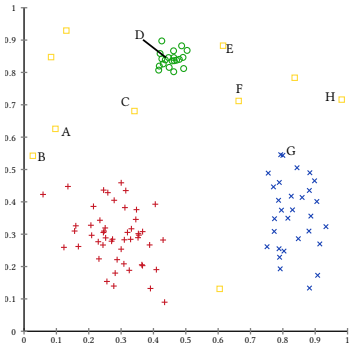
The **cluster order** with respect to ε and minPts

- begins with an arbitrary object
- the next object has the *minimum reachability distance* among all unprocessed objects discovered so far

Reachability plot:



→ cluster order →



The core distance can also be plotted similar, and can be used to identify noise.

OPTICS Algorithm [Ank+99]

Algorithm: OPTICS Clustering Algorithm [Ank+99]

```
SeedList ← empty priority queue
ClusterOrder ← empty list
repeat
    if SeedList.empty() then
        if no unprocessed points then stop
        (r, o) ← (∞, next unprocessed point)        // Get an unprocessed point
    else (r, o) ← SeedList.deleteMin()              // next point from seed list
    N ← RangeQuery(o, dist, ε)                      // get neighborhood
    ClusterOrder.add( (r, o, CoreDist(o, N)) )      // add point to cluster order
    Mark o as processed
    if p is Core then foreach n ∈ N do              // Explore neighborhood
        if n is processed then continue
        p ← ReachDist(n ← o)                        // compute reachability
        if n ∈ SeedList then SeedList.decreaseKey( (p, n) ) // update priority queue
                        else SeedList.insert( (p, n) )
```

Priority Queues

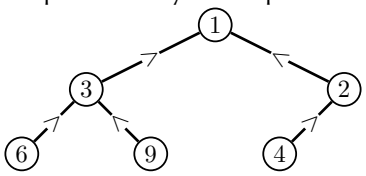
A *priority queue* is a data structure storing object associated with a priority, designed for finding the element with the lowest priority.

Several operations can be defined on priority queues, we need the following:

- Operation DeleteMin: remove and return the smallest element by priority.
- Operation Insert: add an element to the heap.
- Operation DecreaseKey: decrease the key of an object on the heap

Trivial implementation: unsorted list & linear scanning (insert in $O(1)$, all other in $O(n)$)

Better implementation: binary heap $\Rightarrow$ refresh your HeapSort knowledge



Priority Queues /2

DeleteMin:
remove the first element and replace by last, use Heapify-Down to repair, and return the removed element.

Insert:
append new element,
use Heapify-Up to repair the heap.

DecreaseKey:
update existing element,
use Heapify-Up to repair the heap.

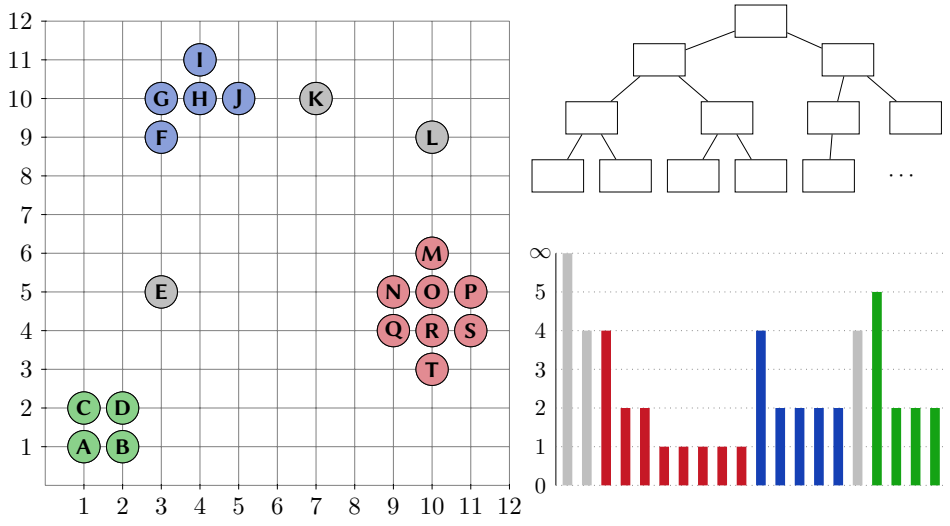
Algorithm: Heapify-Down(array, i, x)

```
1 while 2i + 1 < len(array) do
2     c ← 2i + 1                    // Left  child index
3     if c + 1 < len(array) and array[c + 1] ≤ array[c]
        then c ← c + 1              // Right child index
4     if x < array[c] then break
5     array[i] ← array[c]           // Pull child up
6     i ← c                         // Move down
7 array[i] ← x
```

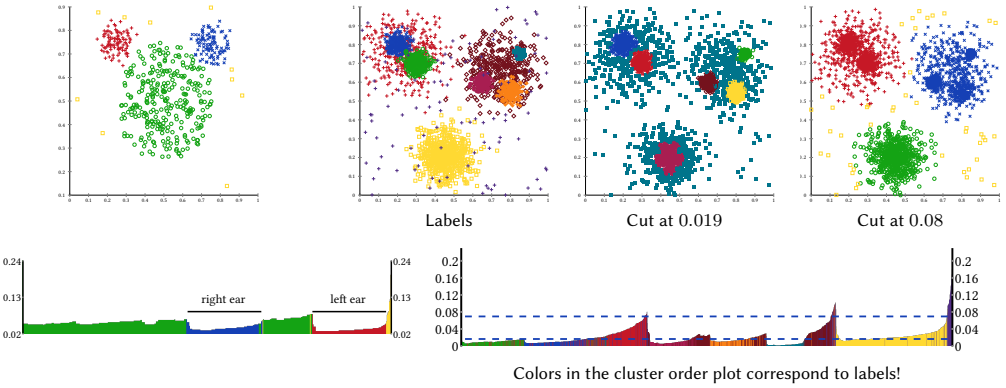
Algorithm: Heapify-Up(array, i, x)

```
1 while i > 0 do
2     j ← ⌊(i − 1)/2⌋               // Parent position
3     if array[j] < x then break
4     array[i] ← array[j]           // Pull parent down
5     i ← j                         // Move upward
6 array[i] ← x
```


OPTICS Example $\mathrm{minPts} = 4$, $\varepsilon = 5$, Manhattan distance



OPTICS Reachability Plots



Colors in the cluster order plot correspond to labels!

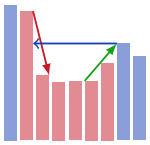
Extracting Clusters from OPTICS Reachability Plots

Horizontal cut:

- merge points with reachability $\leq \varepsilon$ to their predecessor
- result is like DBSCAN with $\varepsilon =$ height of the cut (but only needs $O(n)$ to extract)

ξ method for hierarchical clusters: [Ank+99]

- identify "steep" points where the reachability changes by a factor of $1 - \xi$
- merge neighboring steep points into a steep area.
- find matching pairs of "steep down" and "steep up" points
- in particular the end of the cluster needs to be refined carefully
- must be at least minPts points apart
- can be nested in longer intervals to get a *hierarchical* clustering



Role of the Parameters ε and minPts

Properties of OPTICS and its cluster order

- the cluster order is not unique, but very much order dependent
  because of starting point(s) and many identical reachability distances
- reachabilities are $r_p = \min\{\mathrm{ReachDist}(p \leftarrow n) \mid n \text{ comes before } p \wedge \mathrm{dist}(p,n) \leq \varepsilon\}$.
- ε serves as a cut-off, and improves performance from $O(n^2)$ to possibly $n \log n$ with indexes
- too small ε causes many points with reachability ∞ = disconnected components
- heuristic for ε: the largest minPts-distance based on a sample of the data
- larger minPts produce a smoother plot, because $\mathrm{ReachDist} = \mathrm{CoreDist}$ more often
- $\mathrm{minPts} \leq 2$ – result is equivalent to single-link clustering
- too small minPts can cause "chaining" effects similar to single-link
- rule of thumb: minPts between 10 and 50 often works

Other Density-based Clustering Algorithms

Mean-Shift: [FH75]

- Replace each point with the weighted average of its ε neighbors.
- Points that move to the same location ($\pm$ a small threshold) form a cluster
- Runtime: $O(n^2)$ per iteration, many iterations

DenClue: [HK98]

- Use kernel density estimation
- Use gradients to find density maxima
- Approximate the data using a grid [HG07]

Density-Peak Clustering: [RL14]

- Estimate density of each point
- Every point connects to the nearest neighbor of higher density
- Runtime: $O(n^2)$; needs user input to select clusters

Biclustering & Subspace Clustering

Popular in gene expression analysis.

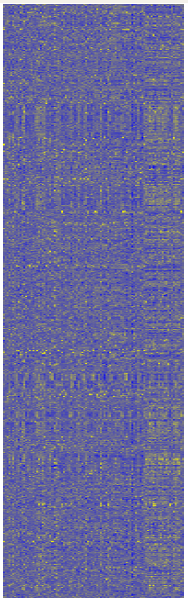
- Every row is a gene
- Every column is a sample

} or transposed

- Only a few genes are relevant
- No semantic ordering of rows or columns
- Some samples may be contaminated
- Numerical value may be unreliable, only "high" or "low"

- Key idea of biclustering: Cheng and Church [CC00]

Find a subset of rows and columns (submatrix, after permutation), such that all values are high/low or exhibit some pattern.



Bicluster Patterns [CC00]

Some examples for bicluster patterns:

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

constant

1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

rows

1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6

columns

1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10
6	7	8	9	10	11

additive

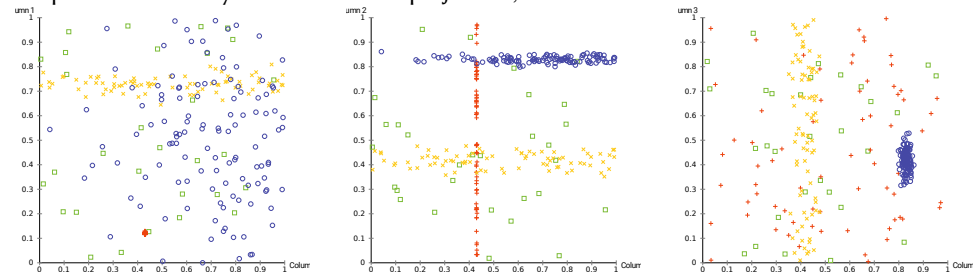
1	2	4	0	4	3
3	6	12	0	12	9
2	4	8	0	8	6
4	8	16	0	16	12
1.5	3	6	0	6	4.5
0.5	1	2	0	2	1.5

multiplicative

- clusters *may* overlap in rows and columns
- patterns will never be this ideal, but noisy!
- many algorithms focus on the constant pattern type only, as there are $\mathcal{O}(2^{N \cdot d})$ possibilities.

Density-based Subspace Clustering

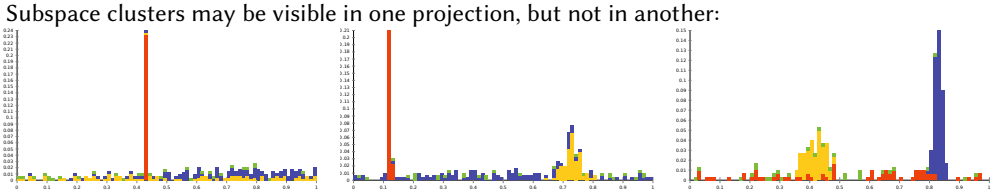
Subspace clusters may be visible in one projection, but not in another:



Clustering Biclustering Methods 3: 105 / 153

Density-based Subspace Clustering

Subspace clusters may be visible in one projection, but not in another:



Popular key idea:

- Find dense areas in 1-dimensional projections
- Combine subspaces as long as the cluster remains dense

Examples: CLIQUE [Agr+98], PROCLUS [Agg+99], SUBCLU [KKK04]
There also exist “correlation clustering”, for rotated subspaces.

BIRCH Clustering [ZRL96]

Cluster Features and the **CF-tree**:

- idea: summarize/compress the data into a tree structure
- phase 1: build a tree, which for every node, stores a cluster feature $\mathrm{CF} = (n, \vec{\mathrm{LS}}, \mathrm{SS})$:
  number of points n; linear sum $\vec{\mathrm{LS}} := \sum_i \vec{x_i}$; sum of squares $\mathrm{SS} := \sum_i \|x_i\|^2$
- points within a threshold criterion are added to an existing node, otherwise add a new node
- when running out of memory, rebuild the tree with a larger threshold
  insert all *leaf cluster features* (not the raw data) into a new tree
- phase 2: build a more compact tree
- phase 3: run a clustering algorithm on the aggregated cluster features
  e.g., k-means, hierarchical clustering, CLARANS
- phase 4: re-scan the data, and map all data points to the nearest cluster
- extensions: data bubbles for DBSCAN [Bre+01],
  two-step clustering for categorical data [Chi+01]

BIRCH Clustering [ZRL96] /2

Cluster Features properties:

- **CF additivity theorem**: we can merge two cluster features $\mathrm{CF}_1 + \mathrm{CF}_2$: $\mathrm{CF}_1 + \mathrm{CF}_2 := (n_1 + n_2, \vec{\mathrm{LS}}_1 + \vec{\mathrm{LS}}_2, \mathrm{SS}_1 + \mathrm{SS}_2)$
- we can derive the following aggregates from a cluster feature CF:[8]
  - Centroid: $\vec{\mu} = \frac{\sum_i \vec{x_i}}{n} = \frac{\vec{\mathrm{LS}}}{n}$
  - "Radius": $R = \sqrt{\frac{1}{n}\sum_i (\vec{x_i} - \vec{\mu})^2} = \sqrt{\frac{1}{n}\mathrm{SS} - \|\mu\|^2}$
  - "Diameter": $D = \sqrt{\sum_{i,j} \|x_i - x_j\|^2 / (n(n-1))} = \sqrt{(n \cdot \mathrm{SS} - \|\vec{\mathrm{LS}}\|^2)/\binom{n}{2}}$
- we can compute several distances of two CF as:
  - centroid Euclidean $D_0(\mathrm{CF}_1, \mathrm{CF}_2) := \|\vec{\mu_1} - \vec{\mu_2}\|$
  - centroid Manhattan $D_1(\mathrm{CF}_1, \mathrm{CF}_2) := \sum_d |\mu_{1,d} - \mu_{2,d}|$
  - average inter-cluster distance $D_2(\mathrm{CF}_1, \mathrm{CF}_2) := \sqrt{\frac{n_1 \mathrm{SS}_2 + n_2 \mathrm{SS}_1 - 2\sum \vec{\mathrm{LS}}_1 \vec{\mathrm{LS}}_2}{n_1 n_2}}$
  - average intra-cluster distance $D_3(\mathrm{CF}_1, \mathrm{CF}_2) := \ldots$
  - variance increase distance $D_4(\mathrm{CF}_1, \mathrm{CF}_2) := \ldots$

[8] Note that the "radius" is rather the average distance from the mean, and the "diameter" is the average pairwise distance. Beware that this suffers from numerical instability with floating point math!

CURE Clustering [GRS98]

Clustering Using REpresentatives (CURE):

- draw a random sample of $s \ll n$ points
- split the sample s into p partitions of size $\frac{s}{p}$ (default $p = 1$)
- cluster each partition into $\frac{s}{pq}$ clusters (default $q = 3$)
- cluster the resulting $\frac{s}{q}$ representative points of all partitions
- label all n data points with the cluster of the nearest representative point

using a modified hierarchical clustering algorithm

- use a k-d-tree to accelerate neighbor search, and keep candidates in a priority queue
- always merge the closest two clusters
- for the merged cluster, find c farthest points, but move them by α to the mean

ROCK Clustering [GRS99]

RObust Clustering using linKs (ROCK) for categorical data:

- uses Jaccard a primary similarity: $J(x, y) := |x \cap y| / |x \cup y|$
- a link is an object $z \neq x, y$ with $J(x, z) < \theta$ and $J(z, y) < \theta$
- let $\mathrm{link}(x, y) = |\{z \mid J(x, z) < \theta \wedge J(z, y) < \theta\}|$ be the number of links
- try to maximize
$$E_l = \sum_i n_i \sum_{x,y \in C_i} \frac{\mathrm{link}(x,y)}{n_i^{1+2f(\theta)}}$$
- merge clusters with best "goodness"
$$g(C_i, C_j) = \frac{\sum_{x \in C_i} \sum_{y \in C_j} \mathrm{link}(x,y)}{(n_i+n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$
- $f(\theta)$ is application and data dependent.
  Example in market basket analysis: $f(\theta) = \frac{1-\theta}{1+\theta}$

CHAMELEON [KHK99]

CHAMELEON: Clustering Using Dynamic Modeling

1. generate a sparse graph, e.g., using the k-nearest neighbors only
2. try to minimize the edge cut EC, the weight of edges cut when partitioning the graph
3. partition the graph using hMETIS[9] into "relatively small" sub-clusters
4. merge sub-clusters back into clusters based on "relative interconnectivity" (EC of merging the cluster, normalized by the average of a fictional partition of the two child clusters) and "relative closeness" (the average length of a edge between the clusters, compared to the weighted average length of an edge within the clusters)

[9] Not published, how this really works. This is proprietary for circuit design.
Implementations of Chameleon usually require installing the original hMETIS binary to do the partioning.

Spectral Clustering [SM00; MS00; NJW01]

Based on graph theory, use the spectrum (eigenvalues) of the similarity matrix for clustering.

- compute the similarity matrix $S = (s_{ij})$ (with $s_{ij} \geq 0$)
  Popular similarities include $s_{ij} = 1$ if i, j are k nearest neighbors, or ε neighbors
- compute the diagonal matrix $D = (d_{ij})$ with $d_{ii} := \sum_j s_{ij}$
- compute a graph Laplacian matrix such as:
  - non-normalized: $L = D - S$
  - Shi-Malik normalized cut: $L_{\text{shi-malik}} = D^{-1}L = I - D^{-1}S$ [SM00]
  - random walk normalized: $L_{\text{rw}} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}SD^{-1/2}$ [NJW01]
- compute *smallest* eigenvectors and eigenvalues
- each 0 eigenvalue indicates a connected component
- we can project onto the eigenvectors of the smallest eigenvalues:
  well connected points will be close, disconnected points will be far
- finally, cluster in the projected space, e.g., using k-means

Affinity Propagation Clustering [FD07]

Based on the idea of message passing: each object "communicates" to others which object they would prefer to be a cluster representative, and how much responsibility they received.

- compute a similarity matrix, e.g., $s(x, y) = -\|x - y\|^2$
- set the diagonal to the median of $s(x, y)$ (adjust to control the number of clusters)
- iteratively send responsibility $r(i, k)$ and availability $a(i, k)$ messages:

$$r(i,k) \leftarrow s(i,k) - \max_{j \neq k}\{a(i,j) + s(i,j)\}$$

$$a(i,k) \leftarrow \begin{cases} \min\{0, r(k,k) + \sum_{j \notin \{i,k\}} \max(0, r(j,k))\} & i \neq k \\ \sum_{j \notin \{k\}} \max(0, r(j,k)) & i = k \end{cases}$$

- objects i are assigned to object k which maximizes $a(i, k) + r(i, k)$;
  if $i = k$ (and the value is positive), then they are cluster representatives and form a cluster
- the number of clusters is decided indirectly, by $s(i, i)$.
- runtime complexity: $O(n^2)$, and needs $3 \cdot n^2$ memory to store (s_{ij}), (r_{ij}), (a_{ij}).

Further Clustering Approaches

We only briefly touched many subdomains, such as:

- biclustering
  e.g., Cheng and Church, …
- projected and subspace clustering
  e.g., CLIQUE, PROCLUS, PreDeCon, SUBCLU, P3C, StatPC, …

Some notable subtypes not covered at all in this class:

- correlation clustering
  e.g., ORCLUS, LMCLUS, COPAC, 4C, ERiC, CASH, …
- online and streaming algorithms
  e.g., StreamKM++, CluStream, DenStream, D-Stream, …
- graph clustering
- topic modeling
  e.g., pLSI, LDA, …

Evaluation of Clustering

We can distinguish between four different kinds of evaluation:

- Unsupervised evaluation (usually based on distance / deviation)
  Statistics such as SSQ, Silhouette, Davies-Bouldin, …
  Used as: heuristics for choosing model parameters such as k
- Supervised evaluation based on labels
  Indexes such as Adjusted Rand Index, Normalized Mutual Information, Purity, …
  Used as: similarity with a reference solution
- Indirect evaluation
  Based on the performance of some other (usually supervised) algorithm in a later stage
  E.g., how much does classification improve, if we use the clustering as feature
- Expert evaluation
  Manual evaluation by a domain expert

Unsupervised Evaluation of Clusterings

Recall the basic idea of distance-based clustering algorithms:

- Items in the same cluster should be more similar
- Items in other clusters should be less similar

- compare the distance within the same cluster to distances to other clusters

Some simple approaches (N points, d dimension):

$$\text{MeanDistance}(C) := \frac{1}{N}\sum_{C_i}\sum_{x\in C_i}\text{dist}(x,\mu_{C_i})$$

$$\text{MeanSquaredDistance}(C) := \frac{1}{N}\sum_{C_i}\sum_{x\in C_i}\text{dist}(x,\mu_{C_i})^2$$

$$\text{RMSSTD}(C) := \sqrt{\frac{1}{\sum_{C_i} d\cdot(|C_i|-1)}\sum_{C_i}\sum_{x\in C_i}\|x-\mu_{C_i}\|^2}$$

How can we combine this with the distance to other clusters?

R^2: Coefficient of Determination

The "total sum of squares" (TTS) of a data set X is:

$$\mathrm{TTS} = \sum_x \left\|x - \overline{X}\right\|^2 = \mathrm{SSE}(X)$$

this can be decomposed into:

$$\begin{aligned}\mathrm{TTS} &= \sum_{C_i} \left(\sum_{x \in C_i} \left\|x - \overline{C_i}\right\|^2 + |C_i| \left\|\overline{X} - \overline{C_i}\right\|^2 \right) \\ &= \underbrace{\sum_{C_i} \mathrm{SSE}(C_i)}_{\mathrm{WCSS}} + \underbrace{\sum_{C_i} |C_i| \left\|\overline{X} - \overline{C_i}\right\|^2}_{\mathrm{BCSS}} \\ \mathrm{TTS} &= \mathrm{WCSS} + \mathrm{BCSS}\end{aligned}$$

TSS	Total Sum of Squares
WCSS	Within-Cluster Sum of Squares
BCSS	Between-Cluster Sum of Squares

- Because total sum of squares TSS is *constant*, minimizing WCSS = maximizing BCSS

Explained variance $R^2 := \mathrm{BCSS}/\mathrm{TSS} = (\mathrm{TSS} - \mathrm{WCSS})/\mathrm{TSS} \in [0, 1]$

Variance-Ratio-Criterion [CH74]

The **Calinski-Harabasz Variance-Ratio-Criterion** (VRC) is defined as:

$$\mathrm{VRC} := \frac{\mathrm{BCSS}/(k-1)}{\mathrm{WCSS}/(N-k)} = \frac{N-k}{k-1}\frac{\mathrm{BCSS}}{\mathrm{WCSS}}$$

Increasing k increases BCSS and decreases WCSS.
Here, both terms get a penalty for increasing k!

Connected to statistics: one-way analysis of variance (ANOVA) test ("F-test").

Null hypothesis: all samples are drawn from distributions with the same mean.
A large F-value indicates that the difference in means is significant.

Beware: you *must* not use the same data to cluster and to do this test!
The proper test scenario is to, e.g., test if a categorical attribute exhibits a significant difference in means of a numerical attribute (and which is *only* used in the test).

Silhouette [Rou87]

Define the mean distance of x to its own cluster, and to the closest other cluster:

$$\begin{aligned} a(x \in C_i) &:= \mathrm{mean}_{y \in C_i, y \neq x}\ d(x, y) \\ b(x \in C_i) &:= \min_{C_j \neq C_i} \mathrm{mean}_{y \in C_j}\ d(x, y)\end{aligned}$$

The silhouette width of a point x (can be used for plotting) then is:

$$s(x \in C_i) := \begin{cases} \frac{b(x)-a(x)}{\max\{a(x),b(x)\}} & \text{if } |C_i| > 1 \\ 0 & \text{otherwise} \end{cases}$$

The silhouette of a clustering C then is:

$$\mathrm{Silhouette}(C) := \mathrm{mean}_x\, s(x)$$

Silhouette of a point: 1 if $a \ll b$ (well assigned point)
0 if $a = b$ (point is between the two clusters), and
-1 if $a \gg b$ (poorly assigned point).
An average Silhouette of > 0.25 is considered "weak", > 0.5 is "reasonable", > 0.7 is "strong".

Silhouette [Rou87] /2

Challenges with using the Silhouette criterion:

- The complexity of Silhouette is: $\mathcal{O}(n^2)$
  $\Rightarrow$ does not scale to large data sets.
- **Simplified Silhouette**:
  use the distances to cluster centers instead of average distances, $\mathcal{O}(n \cdot k)$
- When a cluster has a single point, $a(x)$ is not defined.
  Rousseeuw [Rou87] suggests to use $s(x) = 0$ then (see the definition of $s(x)$)
- Which distance should we use, e.g., with k-means – Euclidean, or squared Euclidean?
- In high-dimensional data, $a(x) \to b(x)$ due to the curse of dimensionality. Then $s(x) \to 0$

Davies-Bouldin Index [DB79]

Power mean (with power p) of $(\|x-y\|_p)^p = \sum_i (x_i - y_i)^p$

Let the scatter of a cluster C_i be (recall L_p-norms):

$$S_i := \left(\frac{1}{|C_i|}\sum_{x \in C_i} \|x - \mu_{C_i}\|_p^p\right)^{1/p}$$

Let the separation of clusters be:

$$M_{ij} := \left\|\mu_{C_i} - \mu_{C_j}\right\|_p$$

For $p = 2$, S_i is standard deviation, M_{ij} is the Euclidean distance!

The similarity of two clusters then is defined as:

$$R_{ij} := \frac{S_i + S_j}{M_{ij}}$$

Clustering quality is the average maximum similarity:

$$\mathrm{DB}(C) := \mathrm{mean}_{C_i}\ \max_{C_j \neq C_i}\ R_{ij}$$

A small Davies-Bouldin index is better, i.e., $S_i + S_j \ll M_{ij}$, scatter $\ll$ distance

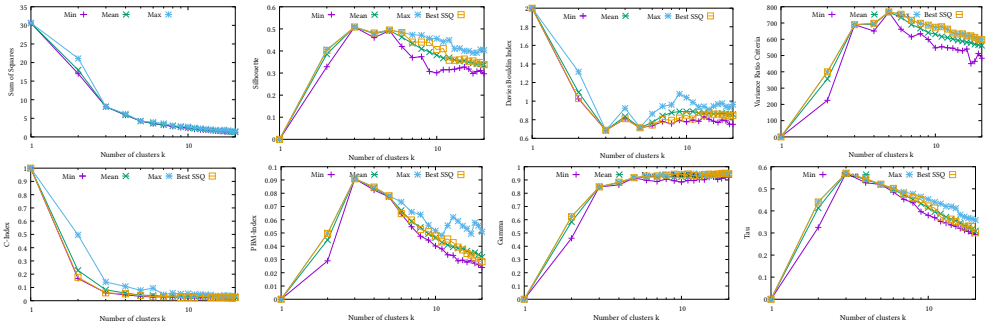
Other Internal Clustering Indexes

There have been many more indexes proposed over time:

- Dunn index: cluster distance / maximum cluster diameter [Dun73; Dun74]
- Gamma and Tau: P(within-cluster distance $<$ between-cluster distance) [BH75]
- C-Index: sum of within-cluster distances / same number of smallest distances [HL76]
- Xie-Beni index for fuzzy clustering [XB91]
- Gap statistic: compare to results on generated random data [TWH01]
- I-Index: maximum between cluster centers / sum of distances to the cluster center [MB02]
- S_Dbw: separation and density based [HV01; HBV02]
- PBM Index: distance to cluster center / distance to total center [PBM04]
- DBCV: density based cluster validation [Mou+14]
- ...

Examples: Mouse data

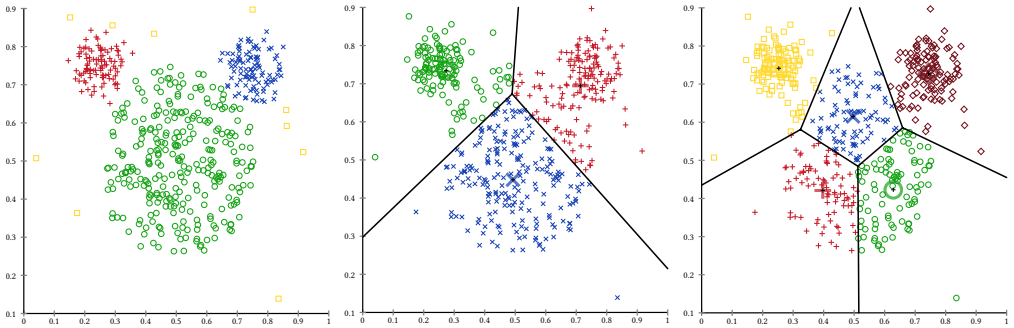
Revisiting k-means on the mouse data:



Examples: Mouse data

Revisiting k-means on the mouse data:

Toy "mouse" data set: Least SSQ with $k = 3$: Least SSQ with $k = 5$:



Supervised Cluster Evaluation

External evaluation measures assume we know the "true" clustering.

In the following, every point has a cluster $C(x)$ and a true class $K(x)$.

The "raw data" (e.g., vectors) will *not* be used by these measures.

In literature this is popular to compare the potential of algorithms.

Often, classification data is used, and it is assumed that good clusters = the classes.

On real data this will often not be possible: no labels available.

There may be more than one meaningful clustering of the same data!

Sometimes, we can at least label some data, or treat some properties as potential labels, then choose the clustering that has the best agreement on the labeled part of the data.

Supervised Cluster Evaluation /2

The matching problem:

- Clusters C are usually enumerated $1, 2, 3, \ldots, k$
- True classes K are usually labeled with meaningful classes
- Which C is which class K?
  - Clustering is *not* classification, we cannot evaluate it the same way
- What if there are more clusters than classes?
- What if a cluster contains two classes?
- What if a class contains two clusters?

To overcome this

- Choose the best (C, K) matching with, e.g., the Hungarian algorithm (uncommon)
- Compare every cluster C to every class K

Purity, Precision and Recall

A simple measure popular in text clustering (but not much in "other" clustering):

$$\mathrm{Purity}(C_i, K) := \max_{K_j} \frac{|C_i \cap K_j|}{|C_i|}$$

$$\mathrm{Purity}(C, K) := \frac{1}{N} \sum_i |C_i| \mathrm{Purity}(C_i, K) = \frac{1}{N} \sum_i \max_{K_j} |C_i \cap K_j|$$

$\Rightarrow$ A cluster, which *only* contains elements of class K_j has purity 1

- similar to "precision" in classification and information retrieval

But: every document is its own cluster has purity 1, is this really optimal?

We could also define a "recall" equivalent:

$$\mathrm{Recall}(C, K_j) := \max_{C_i} \frac{|C_i \cap K_j|}{|K_j|}$$

$$\mathrm{Recall}(C, K) := \frac{1}{N} \sum_j |K_j| \mathrm{Recall}(C, K_j) = \frac{1}{N} \sum_j \max_{C_i} |C_i \cap K_j|$$

But this is even more misleading: if we put all objects into one cluster, we get recall 1!

Pair-counting Evaluation

Many cluster evaluation measures are based on pair-counting.

If (and only if) i and j are in the same cluster, then (i, j) is a pair.

This gives us a *binary, classification-like* problem:

		$K(i) = K(j)$: pair	$K(i) \neq K(j)$: no pair
$C(i) = C(j)$	pair	true positive (a)	false positive (b)
$C(i) \neq C(j)$	no pair	false negative (c)	true negative (d)

which can be computed using

$$a = \sum_{i,j} \binom{|C_i \cap K_j|}{2} \qquad b = \sum_i \binom{|C_i|}{2} - a$$

$$c = \sum_i \binom{|K_i|}{2} - a \qquad d = \binom{N}{2} - a - b - c$$

Objects are a "pair" iff they are related ("should be together")

- we are predicting which objects are related (pairs), and which are not

Pair-counting Cluster Evaluation Measures

		$K(i) = K(j)$: pair	$K(i) \neq K(j)$: no pair
$C(i) = C(j)$	pair	true positive (a)	false positive (b)
$C(i) \neq C(j)$	no pair	false negative (c)	true negative (d)

$$\text{Precision} = \frac{a}{a+b} \qquad \text{Recall} = \frac{a}{a+c}$$

$$\text{Rand index [Ran71]} = \frac{a+d}{a+b+c+d} = (a+d)/\binom{N}{2} = \text{Accuracy}$$

$$\text{Fowlkes-Mallows [FM83]} = \sqrt{\text{Precision} \cdot \text{Recall}} = a/\sqrt{(a+b)\cdot(a+c)}$$

$$\text{Jaccard} = \frac{a}{a+b+c}$$

$$\text{F}_1\text{-Measure} = \frac{2\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}} = \frac{2a}{2a+b+c}$$

$$\text{F}_\beta\text{-Measure} = \frac{(1+\beta^2)\cdot\text{Precision}\cdot\text{Recall}}{\beta^2\cdot\text{Precision}+\text{Recall}}$$

$$\text{Adjusted Rand Index [HA85]} = \frac{\text{Rand index} - E[\text{Rand index}]}{\text{optimal Rand index} - E[\text{Rand index}]}$$

$$= \frac{\binom{N}{2}(a+d) - ((a+b)(a+c)+(c+d)(b+d))}{\binom{N}{2}^2 - ((a+b)(a+c)+(c+d)(b+d))}$$

Mutual Information [Mei03; Mei05; Mei12]

Evaluation by Mutual Information:

$$I(C,K) = \sum_i \sum_j P(C_i \cap K_j) \log \frac{P(C_i \cap K_j)}{P(C_i)\cdot P(K_j)}$$

$$= \sum_i \sum_j \frac{|C_i \cap K_j|}{N} \log \frac{N|C_i \cap K_j|}{|C_i|\cdot|K_j|}$$

Using $\log \frac{a}{b} = -\log \frac{b}{a}$

Entropy:

$$H(C) = -\sum_i P(C_i) \log P(C_i) = -\sum_i \frac{|C_i|}{N} \log \frac{|C_i|}{N} = I(C,C)$$

Normalized Mutual Information (NMI):[10]

$$\text{NMI}(C,K) = \frac{I(C,K)}{(H(C)+H(K))/2} = \frac{I(C,K)+I(K,C)}{I(C,C)+I(K,K)}$$

I and H are the usual Shannon entropy.

[10] There exist ≥ 5 variations of normalized mutual information [VEB10].

Further Supervised Cluster Evaluation Measures

Some further evaluation measures:

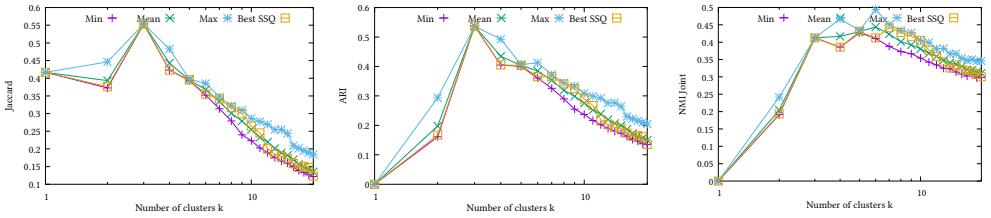
- Adjustment for chance is general principle,

  $$\text{Adjusted Index} = \frac{\text{Index} - E[\text{Index}]}{\text{optimal Index} - E[\text{Index}]}$$

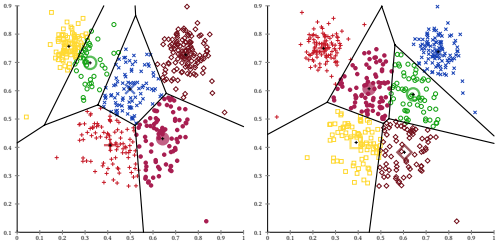
  For example Adjusted Rand Index [HA85] or Adjusted Mutual Information [VEB10]
- B-Cubed evaluation [BB98]
- Set matching purity [ZK01] and F1 [SKK00]
- Edit distance [PL02]
- Visual comparison of multiple clusterings [Ach+12]
- Gini-based evaluation [Sch+15]
- ...

Supervised Evaluation of the Mouse Data Set

Revisiting k-means on the mouse data:



On this toy data set, most unsupervised methods prefer $k = 3$.
NMI prefers the right solution with $k = 6$.
SSQ prefers the left (worse?) solution.



Spatial Index Structures

Key idea of spatial search:

- aggregate the data into *spatial* partitions
- identify partitions that are relevant for our search
- search within relevant partitions only
- do this recursively

Requirements for *efficient* search:

- the search result is only a small subset
- we have a compact summary of the partition
- filtering rules can prune non-relevant partitions
- partitions do not overlap too much
- partitions do not vary in the number of objects too much

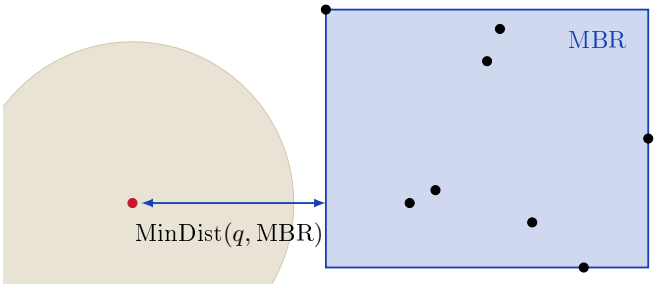
Spatial Index Structures /2

Observations

- index structures already realize some (very rough) “clustering”
- similar objects are usually nearby in the index
- can usually be constructed fairly fast (with bulk-loading techniques etc.)
- can be re-used and shared for multiple applications
- can store complex objects such as polygons, often using bounding rectangles
- objects and partitions overlap
- no universal index – needs to be carefully set up (distances, queries, …)
- if they work, speedups are often substantial

Spatial Data Search: ε-Range Queries

Example: summarize partitions using **Minimum Bounding Rectangles** (MBR):



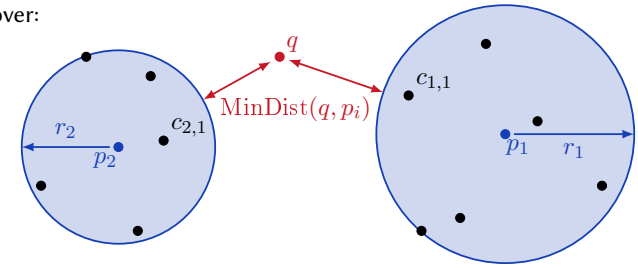
Instead of computing the distances to each point, compute the minimum distance to the MBR.

If $\text{MinDist}(q, \text{MBR}) > \varepsilon$, then no point in the MBR *can* be closer than ε.

If $\text{MinDist}(q, \text{MBR}) \leq \varepsilon$, we need to process the contents of the MBR.

Spatial Data Search: k-Nearest Neighbors

Example: ball cover:



Bounding sphere: for all child points c_{ij} of p_i, we have $\text{dist}(p_i, c_{ij}) \leq r_i$.

By the *triangle inequality*, we have $\text{dist}(q, c_{ij}) + \text{dist}(c_{ij}, p_i) \geq \text{dist}(q, p_i)$.
by using r_i, we get the bound $\text{dist}(q, c_{ij}) \geq \text{dist}(q, p_i) - \text{dist}(c_{ij}, p_i) \geq \text{dist}(q, p_i) - r_i$

Put nodes p_i into a priority queue, ordered by $\text{MinDist}(q, p_i) := \text{dist}(q, p_i) - r_i$.
If we have k results that are better than the remaining queue entries, we can stop.

Balls will usually contain nested balls in a tree structure.

Spatial Index Structures

Many different strategies for partitioning data exist:

- grid — Quadtree [FB74], Gridfile [NHS84], …
- binary splits — k-d-tree [Ben75], …
- minimal bounding boxes — R-tree [Gut84], R$^+$-tree [SRF87], R*-tree [Bec+90], …
- ball covers — Ball-tree [Omo89], M-tree [CPZ97], iDistance[Yu+01], Cover-tree [BKL06], …
- ball and outside — VP-tree [Uhl91; Yia93], …
- …

Most use either rectangles (coordinate ranges), or balls (triangle inequality).

Simpler partition description = less data to read / process / update / ….

R-trees [Gut84]

Objective: efficient management and querying of spatial objects (points, lines, polygons, …)

Common index queries:

- point query: all objects that contain a given point
- region query: all objects that are fully contained in (overlap with) a given query rectangle
- range query: all object within a certain radius
- kNN query: find the k nearest neighbors

Approach

- management, storage, and search of axis-parallel rectangles
- objects are represented by their minimum bounding rectangles in $\mathbb{R}^d$
- search queries operate on minimum bounding rectangles

R-trees [Gut84] /2

R-tree: generalization of B$^+$-tree to multiple dimensions (balanced tree, page fill …)

- stores d-dimensional minimum bounding rectangles (MBR)
  objects (e.g., polygons) are represented by their MBR
- objects stored in leaf nodes only
- objects and nodes are not disjoint, but *may* overlap
- inner nodes store the MBRs of their child nodes
- the root node is the MBR of the entire data set

R$^+$-tree [SRF87]

- no overlap allowed
- requires "splitting" and "clipping" of spatial objects; leads to some redundancy
- guarantees unique path from root to leaf node

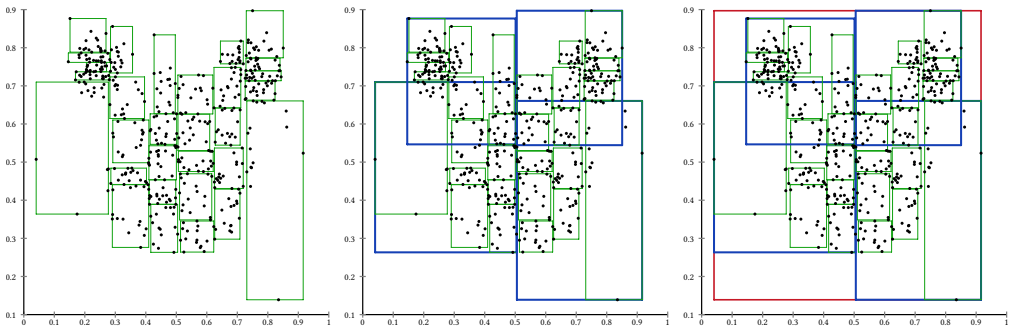
R*-tree [Bec+90]

- improved split heuristic for pages, aimed at minimization of both coverage and overlap

R-trees [Gut84] /3

Example: R-tree for the mouse data set, bulk-loaded:

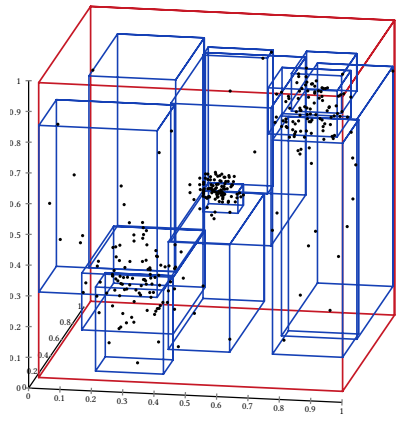
Leaves + inner nodes + root node



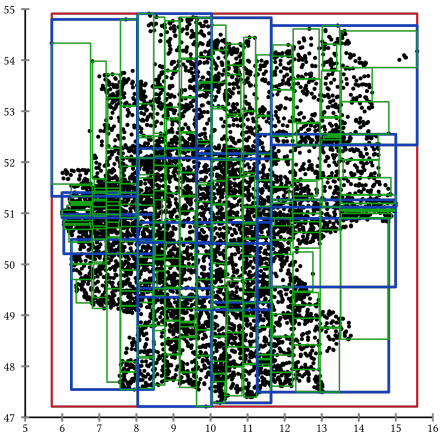
R-trees [Gut84] /4

Further examples:

Gaussians in 3D:

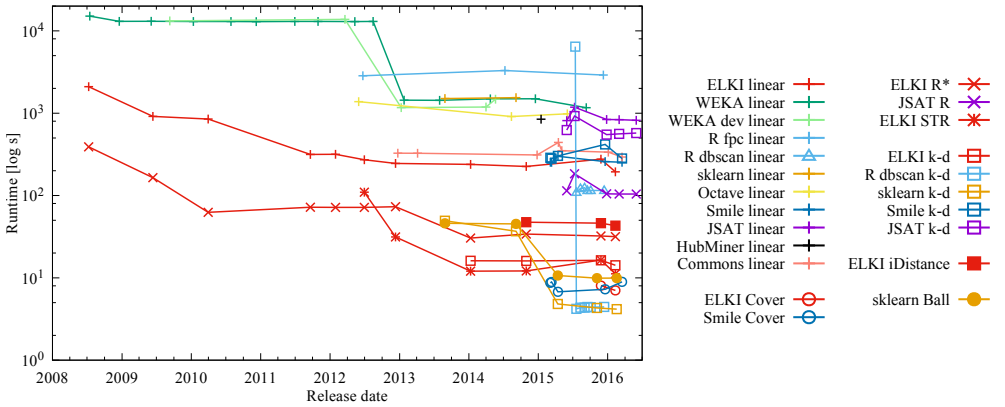


10.000 Wikidata coordinates for Germany:



Speedups Using Index-based Queries [KSZ16]

Index-accelerated DBSCAN implementations are substantially faster:



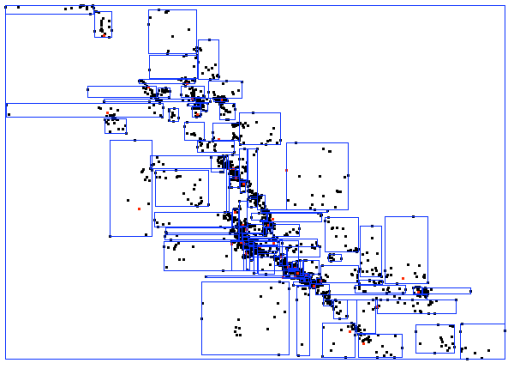
Best non-indexed: > 100 seconds. Best indexed: < 5 seconds.

(But implementation details matter, too. Good code will easily be $10\times$ faster than bad code.)

Index-based Sampling

Approach

- construct R-tree from the data to be clustered
- select one or more representatives from the data pages (leaf nodes) of the R-tree
- apply clustering only to these representatives



Bibliography I

[Ach+12] Elke Achtert, Sascha Goldhofer, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. "Evaluation of Clusterings - Metrics and Visual Support". In: *IEEE ICDE*. 2012, pp. 1285–1288.

[Agg+99] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. "Fast Algorithms for Projected Clustering". In: *Proc. ACM SIGMOD International Conference on Management of Data*. 1999, pp. 61–72.

[Agr+98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". In: *SIGMOD*. 1998, pp. 94–105.

[Aka77] H. Akaike. "On entropy maximization principle". In: *Applications of Statistics*. 1977, pp. 27–41.

[And73] M. R. Anderberg. "Cluster analysis for applications". In: Probability and mathematical statistics. Academic Press, 1973. Chap. Hierarchical Clustering Methods, pp. 131–155. ISBN: 0120576503.

[Ank+99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. "OPTICS: Ordering Points To Identify the Clustering Structure". In: *SIGMOD 1999*. 1999, pp. 49–60.

[AV06] David Arthur and Sergei Vassilvitskii. "How slow is the k-means method?" In: *Symposium on Computational Geometry*. 2006, pp. 144–153.

[AV07] D. Arthur and S. Vassilvitskii. "k-means++: the advantages of careful seeding". In: *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2007, pp. 1027–1035.

[Ban+05] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. "Clustering with Bregman Divergences". In: *J. Machine Learning Research* 6 (2005), pp. 1705–1749.

Bibliography II

[BB98] A. Bagga and B. Baldwin. "Entity-Based Cross-Document Coreferencing Using the Vector Space Model". In: *COLING-ACL*. 1998, pp. 79–85.

[Bec+90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In: *SIGMOD 1990*. 1990, pp. 322–331.

[Ben75] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9 (1975), pp. 509–517.

[Bez81] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Springer, 1981.

[BH65] G. H. Ball and D. J. Hall. *ISODATA, a novel method of data analysis and pattern classification*. Tech. rep. Stanford Research Institute, 1965.

[BH75] F. B. Baker and L. J. Hubert. "Measuring the Power of Hierarchical Cluster Analysis". In: *Journal American Statistical Association* 70.349 (1975), pp. 31–38.

[BKL06] Alina Beygelzimer, Sham Kakade, and John Langford. "Cover trees for nearest neighbor". In: *ICML*. 2006, pp. 97–104.

[BMS96] Paul S. Bradley, Olvi L. Mangasarian, and W. Nick Street. "Clustering via Concave Minimization". In: *NIPS*. 1996, pp. 368–374.

[Boc07] H.-H. Bock. "Clustering Methods: A History of k-Means Algorithms". In: *Selected Contributions in Data Analysis and Classification*. Ed. by P. Brito, G. Cucumel, P. Bertrand, and F. Carvalho. Springer, 2007, pp. 161–172.

[Bon64] Raymond E. Bonner. "On Some Clustering Techniques". In: *IBM Journal of Research and Development* 8.1 (1964), pp. 22–32.

Bibliography III

[Bre+01] M. M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. "Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering". In: *SIGMOD*. 2001, pp. 79–90.

[BY07] E. Biçici and D. Yuret. "Locally Scaled Density Based Clustering". In: *ICANNGA 2007*. 2007, pp. 739–748.

[Cad+03] Igor V. Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. "Model-Based Clustering and Visualization of Navigation Patterns on a Web Site". In: *Data Min. Knowl. Discov.* 7.4 (2003), pp. 399–424.

[Cam+13] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander. "A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies". In: *Data Min. Knowl. Discov.* 27.3 (2013), pp. 344–371.

[CC00] Y. Cheng and G. M. Church. "Biclustering of Expression Data". In: *ISMB*. 2000, pp. 93–103.

[CH74] T. Caliński and J. Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics* 3.1 (1974), pp. 1–27.

[Chi+01] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris. "A robust and scalable clustering algorithm for mixed type attributes in large database environment". In: *SIGKDD*. 2001, pp. 263–268.

[CMS13] R. J. G. B. Campello, D. Moulavi, and J. Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: *PAKDD*. 2013, pp. 160–172.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces". In: *VLDB*. 1997, pp. 426–435.

[DB79] D. Davies and D. Bouldin. "A cluster separation measure". In: *Pattern Analysis and Machine Intelligence* 1 (1979), pp. 224–227.

Bibliography IV

[DD09] M. M. Deza and E. Deza. *Encyclopedia of Distances.* 3rd. Springer, 2009. ISBN: 9783662443415.

[DE84] William H. E. Day and Herbert Edelsbrunner. "Efficient algorithms for agglomerative hierarchical clustering methods". In: *Journal of Classification* 1.1 (1984), pp. 7–24.

[Def77] D. Defays. "An Efficient Algorithm for the Complete Link Cluster Method". In: *The Computer Journal* 20.4 (1977), pp. 364–366.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 39.1 (1977), pp. 1–31.

[DM01] Inderjit S. Dhillon and Dharmendra S. Modha. "Concept Decompositions for Large Sparse Text Data Using Clustering". In: *Machine Learning* 42.1/2 (2001), pp. 143–175.

[Dun73] J. C. Dunn. "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57.

[Dun74] J. C. Dunn. "Well separated clusters and optimal fuzzy partitions". In: *Journal of Cybernetics* 4 (1974), pp. 95–104.

[Epp98] David Eppstein. "Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs". In: *Proc. ACM-SIAM Symposium on Discrete Algorithms.* 1998, pp. 619–628.

[Est+96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *KDD'96.* 1996, pp. 226–231.

[Est02] V. Estivill-Castro. "Why so many clustering algorithms – A Position Paper". In: *SIGKDD Explorations* 4.1 (2002), pp. 65–75.

Bibliography V

[FB74] Raphael A. Finkel and Jon Louis Bentley. "Quad Trees: A Data Structure for Retrieval on Composite Keys". In: *Acta Inf.* 4 (1974), pp. 1–9.

[FD07] B. J. Frey and D. Dueck. "Clustering by passing messages between data points". In: *Science* 315.5814 (2007), pp. 972–976.

[FH75] K. Fukunaga and L. D. Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Trans. Information Theory* 21.1 (1975), pp. 32–40.

[FM83] E. B. Fowlkes and C. L. Mallows. "A Method for Comparing Two Hierarchical Clusterings". In: *Journal American Statistical Association* 78.383 (1983), pp. 553–569.

[For65] E. W. Forgy. "Cluster analysis of multivariate data: efficiency versus interpretability of classifications". In: *Biometrics* 21 (1965), pp. 768–769.

[GRS98] S. Guha, R. Rastogi, and K. Shim. "CURE: An Efficient Clustering Algorithm for Large Databases". In: *SIGMOD.* 1998, pp. 73–84.

[GRS99] S. Guha, R. Rastogi, and K. Shim. "ROCK: A Robust Clustering Algorithm for Categorical Attributes". In: *ICDE.* 1999, pp. 512–521.

[Gut84] Antonin Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching". In: *SIGMOD'84.* 1984, pp. 47–57.

[HA85] L. Hubert and P. Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (1985), pp. 193–218.

[Har75] J. A. Hartigan. *Clustering Algorithms.* New York, London, Sydney, Toronto: John Wiley&Sons, 1975.

[HBV02] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. "Clustering Validity Checking Methods: Part II". In: *SIGMOD Record* 31.3 (2002), pp. 19–27.

Bibliography VI

[HE03] Greg Hamerly and Charles Elkan. "Learning the k in k-means". In: *NIPS.* 2003, pp. 281–288.

[HG07] A. Hinneburg and H.-H. Gabriel. "DENCLUE 2.0: Fast Clustering Based on Kernel Density Estimation". In: *Advances in Intelligent Data Analysis VII.* 2007, pp. 70–80.

[HK98] A. Hinneburg and D. A. Keim. "An Efficient Approach to Clustering in Large Multimedia Databases with Noise". In: *KDD'98.* 1998, pp. 58–65.

[HL76] L. J. Hubert and J. R. Levin. "A general statistical framework for assessing categorical clustering in free recall." In: *Psychological Bulletin* 83.6 (1976), pp. 1072–1080.

[Hof99] Thomas Hofmann. "Learning the Similarity of Documents: An Information-Geometric Approach to Document Retrieval and Categorization". In: *Neural Information Processing Systems, NIPS.* 1999, pp. 914–920.

[Hua98] Zhexue Huang. "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values". In: *Data Min. Knowl. Discov.* 2.3 (1998), pp. 283–304.

[HV01] M. Halkidi and M. Vazirgiannis. "Clustering Validity Assessment: Finding the Optimal Partitioning of a Data Set". In: *ICDM.* 2001, pp. 187–194.

[HW79] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* (1979), pp. 100–108.

[KHK99] G. Karypis, E.-H. Han, and V. Kumar. "Chameleon: Hierarchical Clustering Using Dynamic Modeling". In: *IEEE Computer* 32.8 (1999), pp. 68–75.

Bibliography VII

[KKK04] P. Kröger, H.-P. Kriegel, and K. Kailing. "Density-Connected Subspace Clustering for High-Dimensional Data". In: *Proc. of the Fourth SIAM International Conference on Data Mining*. 2004, pp. 246–256.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analyis*. John Wiley&Sons, 1990. ISBN: 9780471878766.

[KSZ16] H.-P. Kriegel, E. Schubert, and A. Zimek. "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?" In: *Knowledge and Information Systems (KAIS)* (2016), pp. 1–38.

[Llo82] S. P. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–136.

[LW67] G. N. Lance and W. T. Williams. "A General Theory of Classificatory Sorting Strategies. 1. Hierarchical Systems". In: *The Computer Journal* 9.4 (1967), pp. 373–380.

[Mac67] J. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In: *5th Berkeley Symposium on Mathematics, Statistics, and Probabilistics*. Vol. 1. 1967, pp. 281–297.

[MAS16] S. T. Mai, I. Assent, and M. Storgaard. "AnyDBC: An Efficient Anytime Density-based Clustering Algorithm for Very Large Complex Datasets". In: *KDD 2016*. 2016, pp. 1025–1034.

[MB02] Ujjwal Maulik and Sanghamitra Bandyopadhyay. "Performance Evaluation of Some Clustering Algorithms and Validity Indices". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.12 (2002), pp. 1650–1654.

[Mei03] M. Meila. "Comparing Clusterings by the Variation of Information". In: *Computational Learning Theory (COLT)*. 2003, pp. 173–187.

[Mei05] M. Meila. "Comparing Clusterings – An Axiomatic View". In: *Int. Conf. Machine Learning (ICML)*. 2005, pp. 577–584.

Bibliography VIII

[Mei12] M. Meilă. "Local equivalences of distances between clusterings–a geometric perspective". In: *Machine Learning* 86.3 (2012), pp. 369–389.

[MM08] S. Mahran and K. Mahar. "Using grid for accelerating density-based clustering". In: *CIT 2008*. 2008, pp. 35–40.

[Mou+14] D. Moulavi, P. A. Jaskowiak, R. J. G. B. Campello, A. Zimek, and J. Sander. "Density-based Clustering Validation". In: *SIAM SDM*. 2014, pp. 839–847.

[MS00] M. Meila and J. Shi. "Learning Segmentation by Random Walks". In: *NIPS*. MIT Press, 2000, pp. 873–879.

[Mül11] D. Müllner. "Modern hierarchical, agglomerative clustering algorithms". In: *arXiv preprint* arXiv:1207.0016 (2011).

[Mur83] F. Murtagh. "A Survey of Recent Advances in Hierarchical Clustering Algorithms". In: *Comput. J.* 26.4 (1983), pp. 354–359.

[NH02] R. T. Ng and J. Han. "CLARANS: A method for clustering objects for spatial data mining". In: *IEEE Transactions on Knowledge and Data Engineering* 14.5 (2002), pp. 1003–1016.

[NHS84] Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. "The Grid File: An Adaptable, Symmetric Multikey File Structure". In: *ACM Trans. Database Syst.* 9.1 (1984), pp. 38–71.

[NJW01] A. Y. Ng, M. I. Jordan, and Y. Weiss. "On Spectral Clustering: Analysis and an algorithm". In: *NIPS*. 2001, pp. 849–856.

[Omo89] Stephen M. Omohundro. *Five Balltree Construction Algorithms*. Tech. rep. TR-89-063. International Computer Science Institute, Berkeley, 1989.

[PBM04] M. K. Pakhira, S. Bandyopadhyay, and U. Maulik. "Validity index for crisp and fuzzy clusters". In: *Pattern Recognition* 37.3 (2004), pp. 487–501.

Bibliography IX

[Phi02] S. J. Phillips. "Acceleration of K-Means and Related Clustering Algorithms". In: *ALENEX'02*. 2002, pp. 166–177.

[PL02] P. Pantel and D. Lin. "Document clustering with committees". In: *ACM SIGIR*. 2002, pp. 199–206.

[PM00] D. Pelleg and A. Moore. "X-means: Extending k-means with efficient estimation of the number of clusters". In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*. Vol. 1. 2000, pp. 727–734.

[Pou+14] M. Pourrajabi, D. Moulavi, R. J. G. B. Campello, A. Zimek, J. Sander, and R. Goebel. "Model Selection for Semi-Supervised Clustering". In: *EDBT 2014*. 2014, pp. 331–342.

[Ran71] W. M. Rand. "Objective criteria for the evaluation of clustering methods". In: *Journal American Statistical Association* 66.336 (1971), pp. 846–850.

[RL14] A. Rodriguez and A. Laio. "Clustering by fast search and find of density peaks". In: *Science* 344.6191 (2014), pp. 1492–1496.

[Rou87] P. J. Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[San+98] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications". In: *Data Min. Knowl. Discov.* 2.2 (1998), pp. 169–194.

[Sch+15] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek. "A Framework for Clustering Uncertain Data". In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1976–1979.

[Sch+17a] E. Schubert, J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". In: *ACM Trans. Database Syst.* 42.3 (2017), 19:1–19:21.

Bibliography X

[Sch+17b] E. Schubert, A. Spitz, M. Weiler, J. Geiß, and M. Gertz. "Semantic Word Clouds with Background Corpus Normalization and t-distributed Stochastic Neighbor Embedding". In: *CoRR* abs/1708.03569 (2017). arXiv: 1708.03569.

[Sch78] G. Schwarz. "Estimating the dimension of a model". In: *The Annals of Statistics* 6.2 (1978), pp. 461–464.

[Sib73] R. Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". In: *The Computer Journal* 16.1 (1973), pp. 30–34.

[SKK00] M. Steinbach, G. Karypis, and V. Kumar. "A comparison of document clustering techniques". In: *KDD workshop on text mining*. Vol. 400. 2000, pp. 525–526.

[SM00] J. Shi and J. Malik. "Normalized Cuts and Image Segmentation". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22.8 (2000), pp. 888–905.

[Sne57] P. H. A. Sneath. "The Application of Computers to Taxonomy". In: *Journal of General Microbiology* 17 (1957), pp. 201–226.

[SRF87] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects". In: *VLDB'87*. 1987, pp. 507–518.

[Ste56] H. Steinhaus. "Sur la division des corp materiels en parties". In: *Bull. Acad. Polon. Sci* 1 (1956), pp. 801–804.

[TWH01] Robert Tibshirani, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423.

[Uhl91] Jeffrey K. Uhlmann. "Satisfying General Proximity/Similarity Queries with Metric Trees". In: *Inf. Process. Lett.* 40.4 (1991), pp. 175–179.

Bibliography XI

[VEB10] N. X. Vinh, J. Epps, and J. Bailey. "Information Theoretic Measures for Clustering Comparison: Variants, Properties, Normalization and Correction for Chance". In: *J. Machine Learning Research* 11 (2010), pp. 2837–2854.

[XB91] Xuanli Lisa Xie and Gerardo Beni. "A Validity Measure for Fuzzy Clustering". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 13.8 (1991), pp. 841–847.

[YH02] Alexander Ypma and Tom Heskes. "Automatic Categorization of Web Pages and User Clustering with Mixtures of Hidden Markov Models". In: *Workshop WEBKDD*. 2002, pp. 35–49.

[Yia93] Peter N. Yianilos. "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces". In: *SODA 1993*. 1993, pp. 311–321.

[Yu+01] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. "Indexing the Distance: An Efficient Method to KNN Processing". In: *VLDB*. 2001, pp. 421–430.

[ZK01] Y. Zhao and G. Karypis. *Criterion Functions for Document Clustering: Experiments and Analysis*. Tech. rep. 01-40. University of Minnesota, Department of Computer Science, 2001.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In: *SIGMOD 1996*. 1996, pp. 103–114.

[ZXF08] Q. Zhao, M. Xu, and P. Fränti. "Knee Point Detection on Bayesian Information Criterion". In: *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. 2008, pp. 431–438.