Introduction to data science

Unsupervised learning

Prof. dr. sc. Bojana Dalbelo Bašić

ak. god. 2023./2024.

Machine learning

- **Supervised:** We are given input/output samples (x, y) which we relate with a function $y = f(x)$. We would like to "learn" f, and evaluate it on new data. Types:
  - **Classification:** y is discrete (class labels).
  - **Regression:** y is continuous, e.g. linear regression.

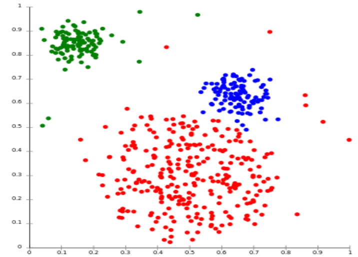
- **Unsupervised:** Given only samples x of the data, we compute a function f such that $y = f(x)$ is a "simpler" representation.
  - Discrete y: **clustering**
  - Continuous y: **dimensionality reduction** (e.g., matrix factorization, unsupervised neural networks)

Recommended book for beginners:

Ethem Alpaydin: Machine Learning: The New AI (The MIT Press Essential Knowledge series) Paperback – October 7, 2016

The clustering problem

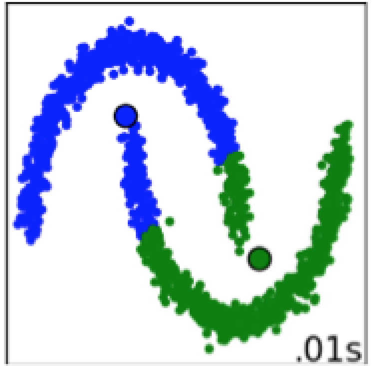
- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, such that
  - members of a cluster are close (i.e., similar) to each other
  - members of different clusters are far apart from each other
- **Usually:**
  - Points are in a high-dimensional space
  - Similarity is defined via a distance measure
    - Euclidean, cosine, Jaccard, edit distance, …

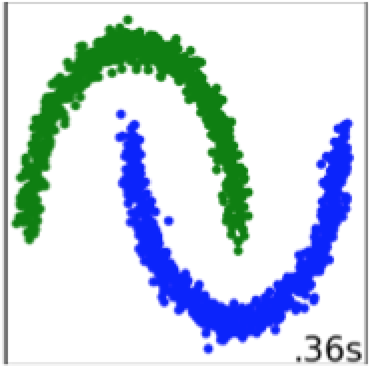


Characteristics of clustering methods

Quantitative: scalability (many samples), dimensionality (many features)

Qualitative: types of features (numerical, categorical, etc.), type of shapes (polyhedra, hyperplanes, manifolds, etc.)



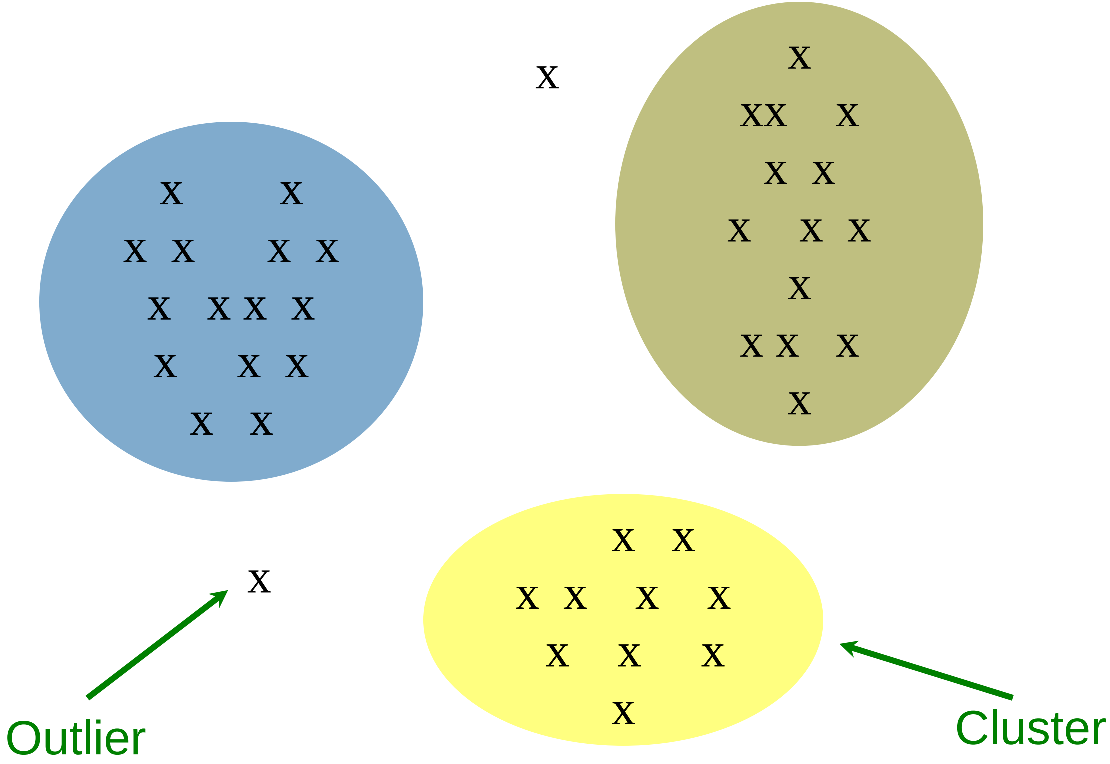


Characteristics of clustering methods

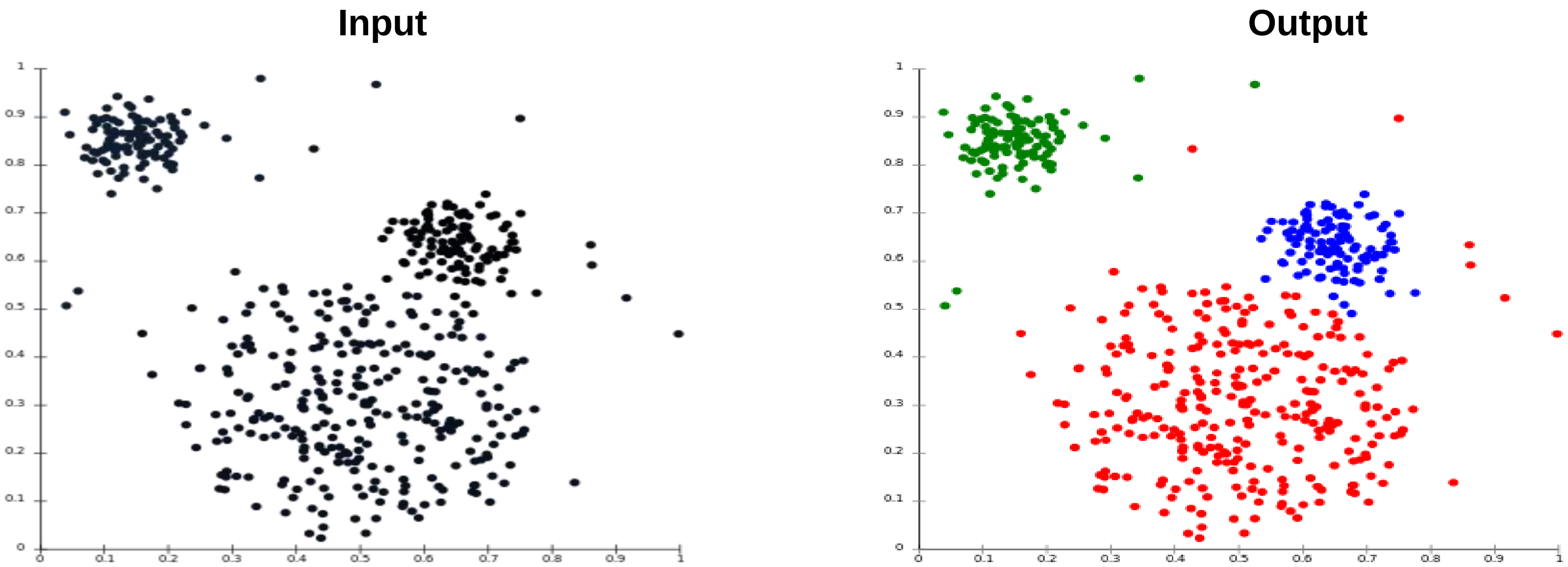
Robustness: sensitivity to noise and outliers, sensitivity to the processing order

User interaction: incorporation of user constraints (e.g., number of clusters, max size of clusters), interpretability and usability

Example: clusters & outliers



A typical clustering example

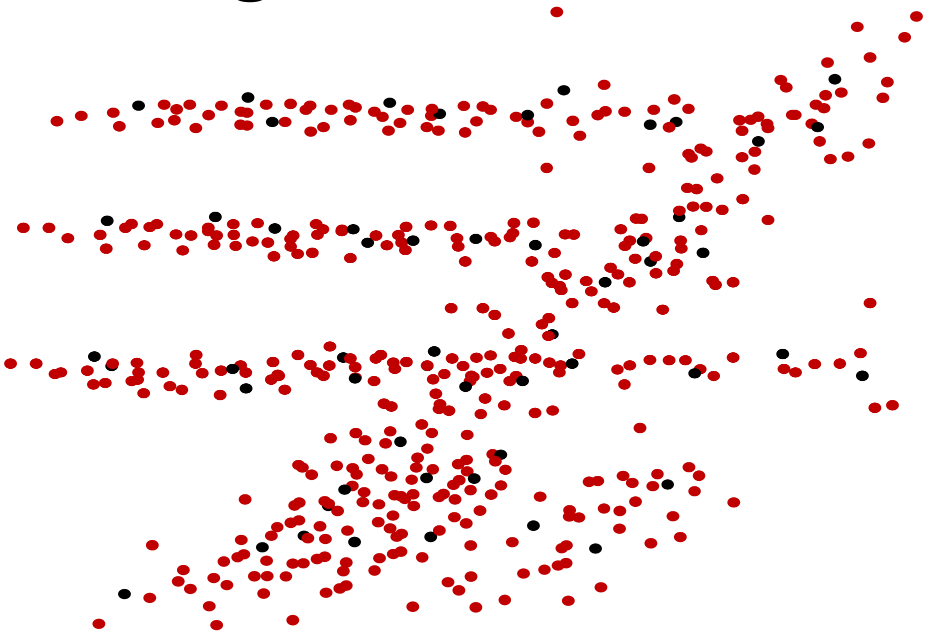


Note: Above is 2D; real scenarios often much more high-dimensional, e.g., 10,000-dimensional for 100x100 images.

Some use cases for clustering

- Data exploration (especially for high-dimensional data, where visualization fails)
- Partitioning of data for more fine-grained subsequent analysis
- Marketing: building personas
- Supporting data labeling for supervised learning
- Data compression (next slide)
- …

Clustering for condensation/compression



Here we don't require that clusters extract meaningful structure, but that they give a coarse-grained version of the data.

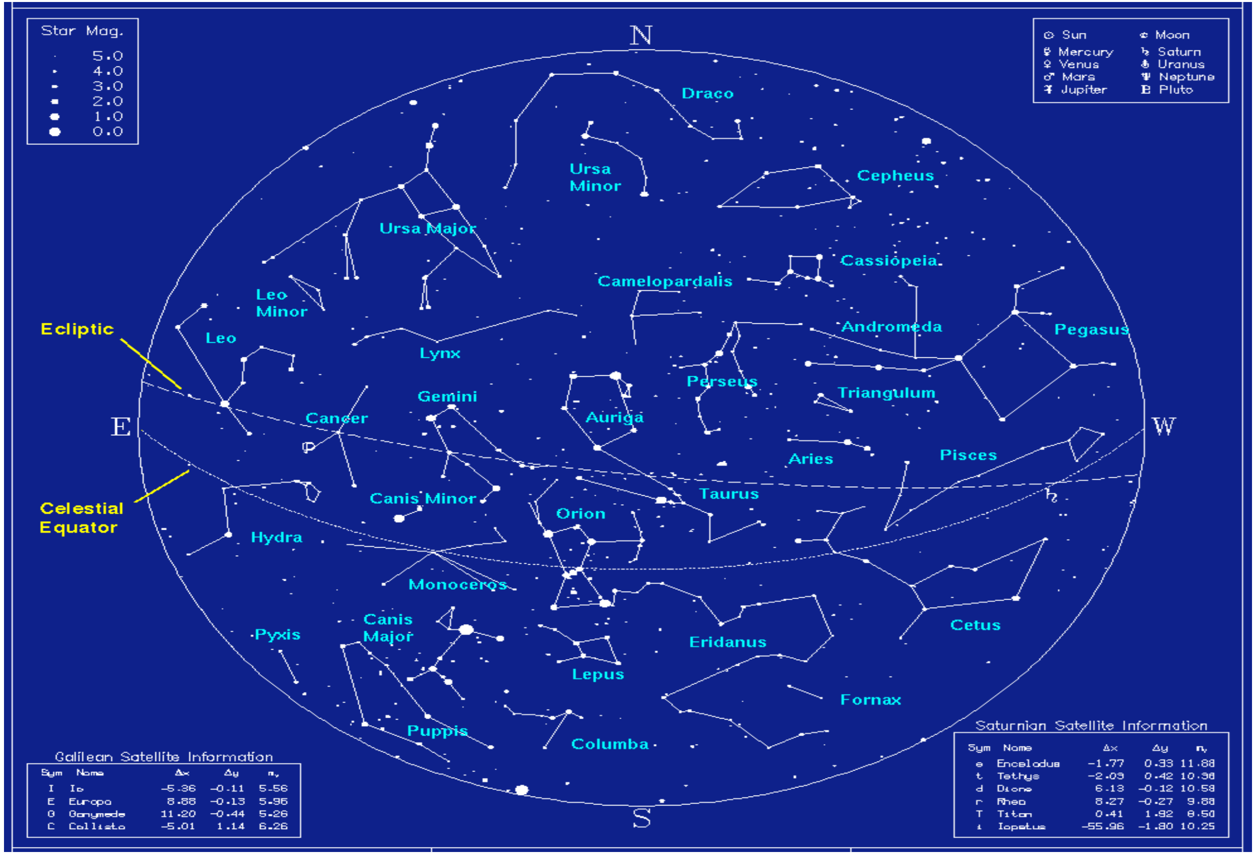
Beware of "cluster bias"!

- Human beings conceptualize the world through categories represented as *exemplars* (Rosch 1973, Estes 1994).



- We tend to see cluster structure whether it is there or not.
- Works well for dogs, but…

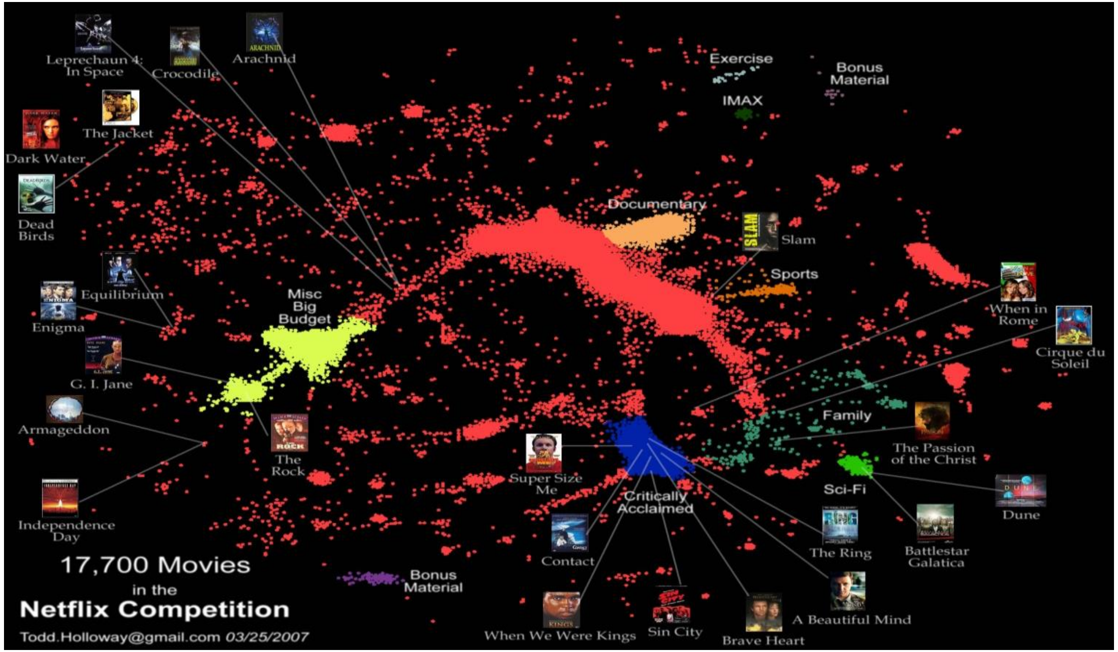
Cluster bias



“Cluster bias”

- **Clustering is used more than it should be**, because people assume an underlying domain has discrete classes in it
  - Especially true for characteristics of people, e.g., Myers-Briggs personality types like “ENTP”.
- In reality the underlying data is usually **continuous**.
- In such cases, continuous models (e.g., matrix factorization, “soft” clustering, k-NN) tend to do better (cf. next slide)

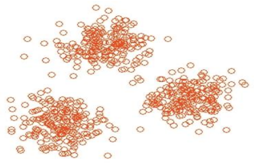
Netflix



- More of a continuum than discrete clusters
- Other methods (e.g., matrix factorization, k-NN) may do better than discrete cluster models

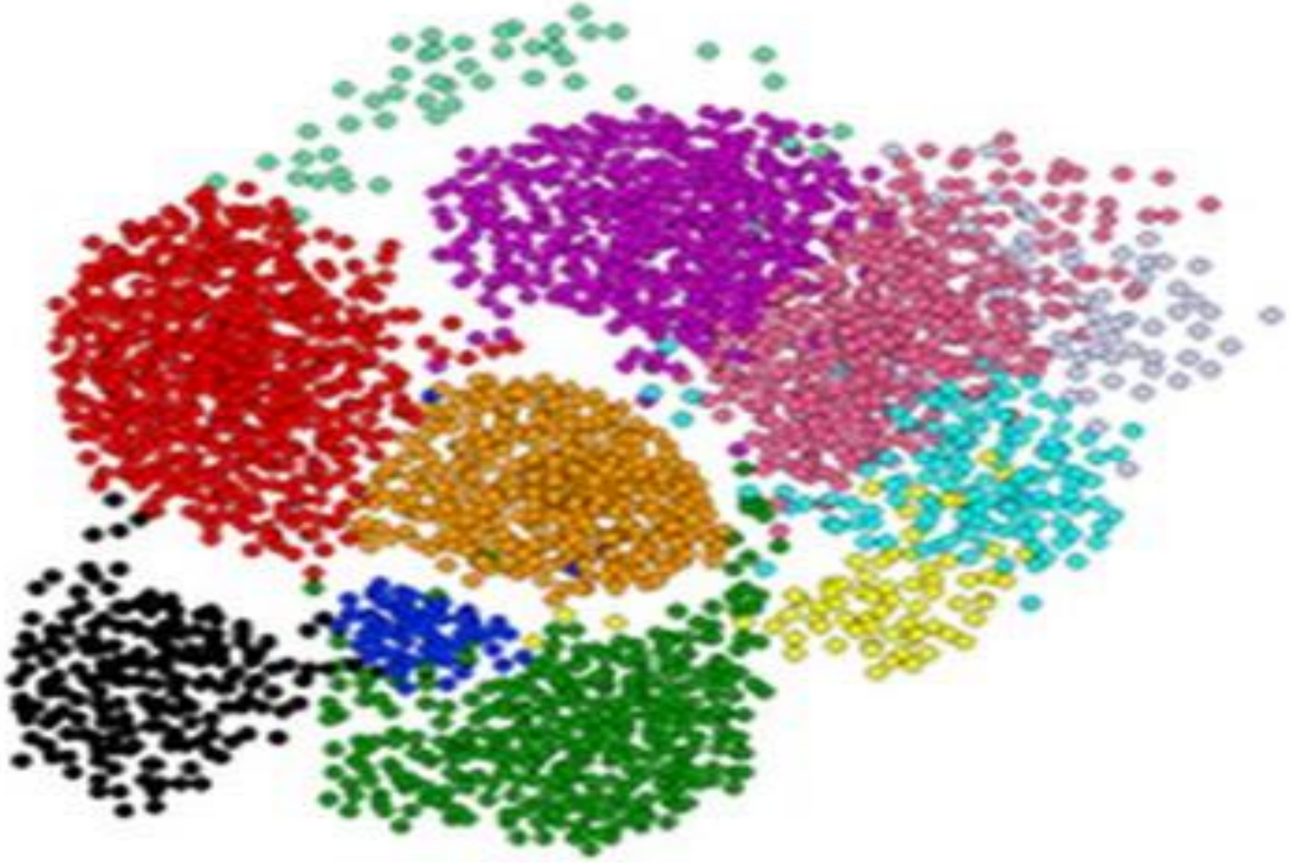
Terminology

- **Hierarchical clustering:** clusters form a tree-shaped hierarchy. Can be computed bottom-up or top-down.
- **Flat clustering:** no inter-cluster structure.



- **Hard clustering:** items assigned to a unique cluster.
- **Soft clustering:** cluster membership is a probability distribution over all clusters

Clustering is a hard problem!



Why is it hard?

- Clustering in 2 dimensions looks easy
- Clustering small amounts of data looks easy
- And in these special cases, it actually is often easy, but...

- … many applications involve not 2, but 10 or 10,000 dimensions (and large amounts of data)
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance (“curse of dimensionality”)

Clustering problem: galaxies

- A catalog of 2 billion "sky objects" represents objects by their radiation in 7 dimensions (frequency bands)
- Problem: Cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.
- Sloan Digital Sky Survey [link]

Clustering problem: music CDs

- **Intuitively: Music divides into categories, and customers prefer a few categories**
  - But what are categories really?
  - —> take a data-driven approach!

- Represent a CD by a set of customers who bought it ("collaborative filtering")
- Similar CDs have similar sets of customers, and vice-versa

Clustering problem: music CDs

Space of all CDs:

- Think of a space with one dimension for each customer
  - Values in a dimension may be 0 or 1 only
  - A CD is a point in this space $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i-th customer bought the CD
- For Amazon, the dimensionality is tens of millions
- **Task:** Find clusters of similar CDs

Clustering problem: documents

Finding topics:

- Represent a document by a vector $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i-th word appears in the document (in any position)

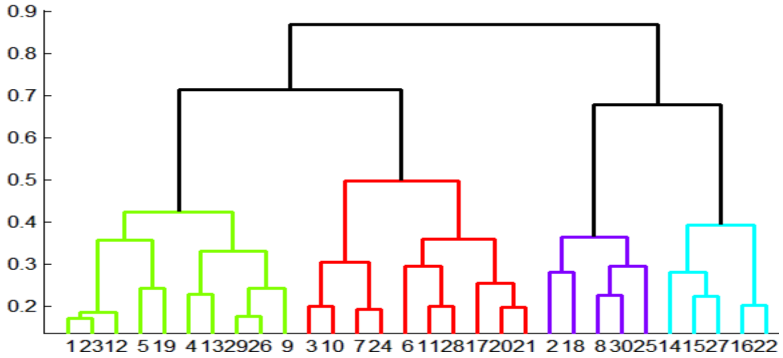
- **Idea: documents with similar sets of words are about same topic**

Cosine, Jaccard, Euclidean distances

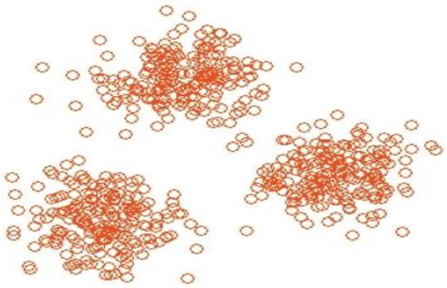
- In both examples (CDs, documents) we have a choice when we thinking of data points as sets of features (users, words):
  - **Sets as vectors:**
    - Measure similarity by **Euclidean distance**
    - Measure similarity by the **cosine distance**
  - **Sets as sets:** Measure similarity by the **Jaccard distance**

Overview: Methods of clustering

- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it



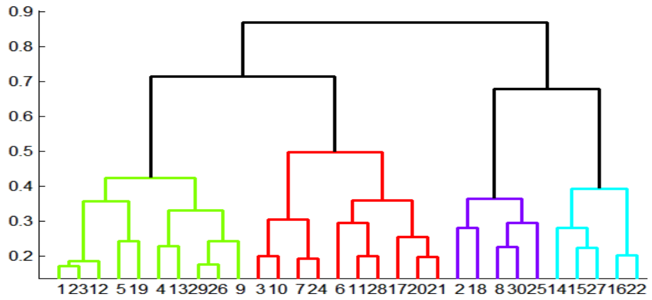
- **Point assignment:**
  - Maintain a set of clusters
  - Points belong to "nearest" cluster



Agglomerative hierarchical clustering

- **Key operation:**

  Repeatedly combine two nearest clusters

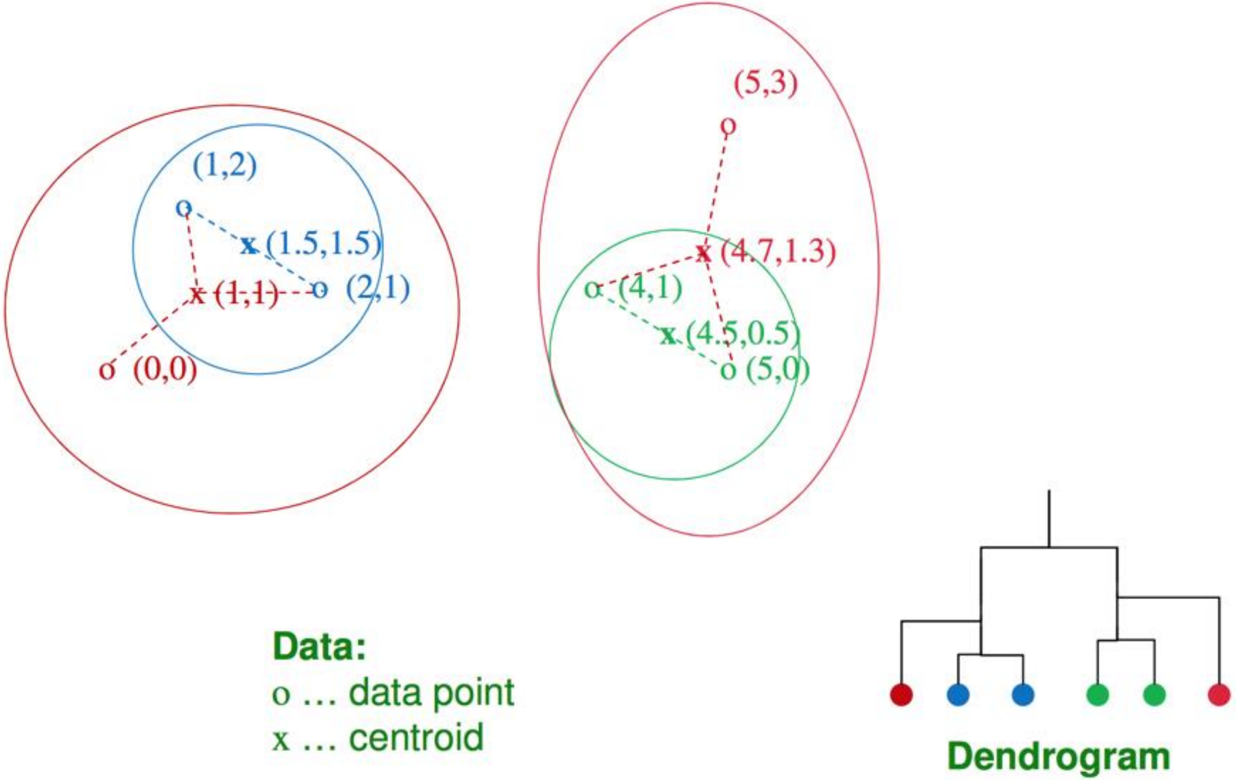


- **Three important questions:**
  - **1)** How to represent a cluster of more than one point?
  - **2)** How to determine the “nearness” of clusters?
  - **3)** When to stop combining clusters?

Agglomerative hierarchical clustering

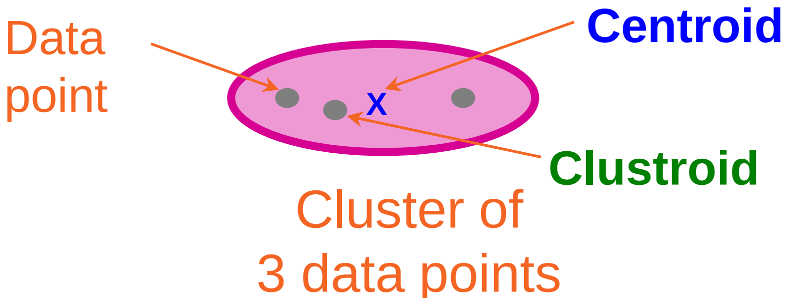
- **Key operation: Repeatedly combine two nearest clusters**
- **(1) How to represent a cluster of many points?**
  - Euclidean case: each cluster has a **centroid** = average of its points
  - What about non-Euclidean case?
- **(2) How to determine "nearness" of clusters?**
  - Euclidean case: measure cluster distances by distances of centroids
  - What about non-Euclidean case?

Example: Hierarchical clustering



Non-Euclidean case: clustroids

- **(1) How to represent a cluster of many points?**
  clustroid = point **"closest"** to other points
- **Possible meanings of "closest":**
  - Smallest maximum distance to other points
  - Smallest average distance to other points (a.k.a. **medoid**)
  - Smallest sum of squares of distances to other points



Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.
Clustroid is an **existing** (data)point that is "closest" to all other points in the cluster.

Defining "nearness" of clusters

- **(2) How do you determine the "nearness" of clusters?**
  - **Approach 1:**
    Intercluster distance = minimum of the distances between any two points, one from each cluster; or average of distances; or distance between centroids/clustroids; etc.
  - **Approach 2:**
    Pick a notion of "**cohesion**" ("tightness") of clusters
    - Nearness of clusters = cohesion of their *union*

Cohesion

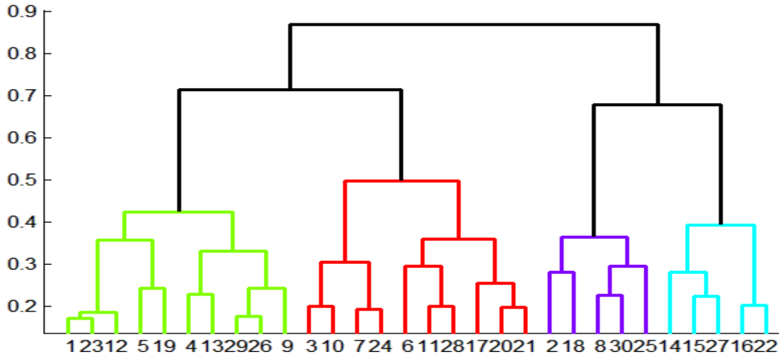
- **Approach 2.1:** Use the **diameter** of the merged cluster = maximum distance between points in the cluster
- **Approach 2.2:** Use the **average distance** between points in the cluster

Implementation

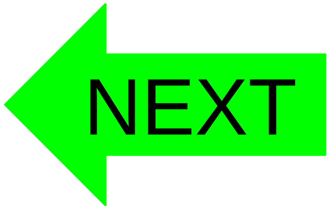
- **Naïve implementation of hierarchical clustering:**
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - $O(N^3)$, where N is the number of data points
- **Careful implementation using priority queue** can reduce time to $O(N^2 \log N)$
  - Still too expensive for really big datasets that do not fit in memory

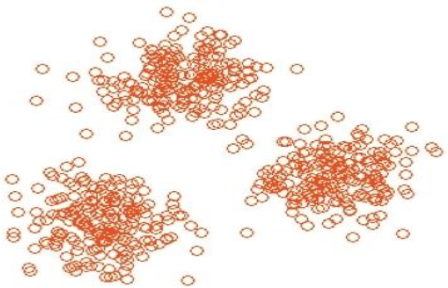
Overview: Methods of clustering

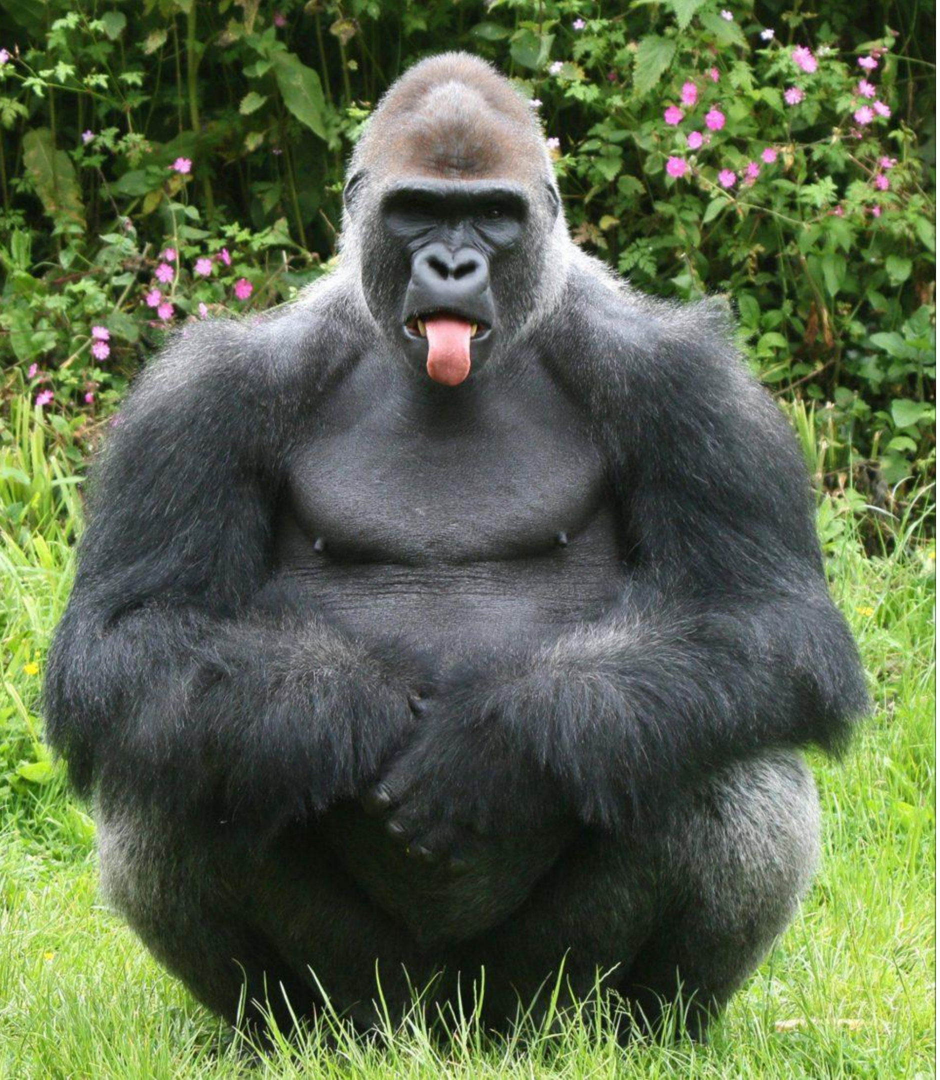
- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it



- **Point assignment:**
  - Maintain a set of clusters
  - Points belong to "nearest" cluster







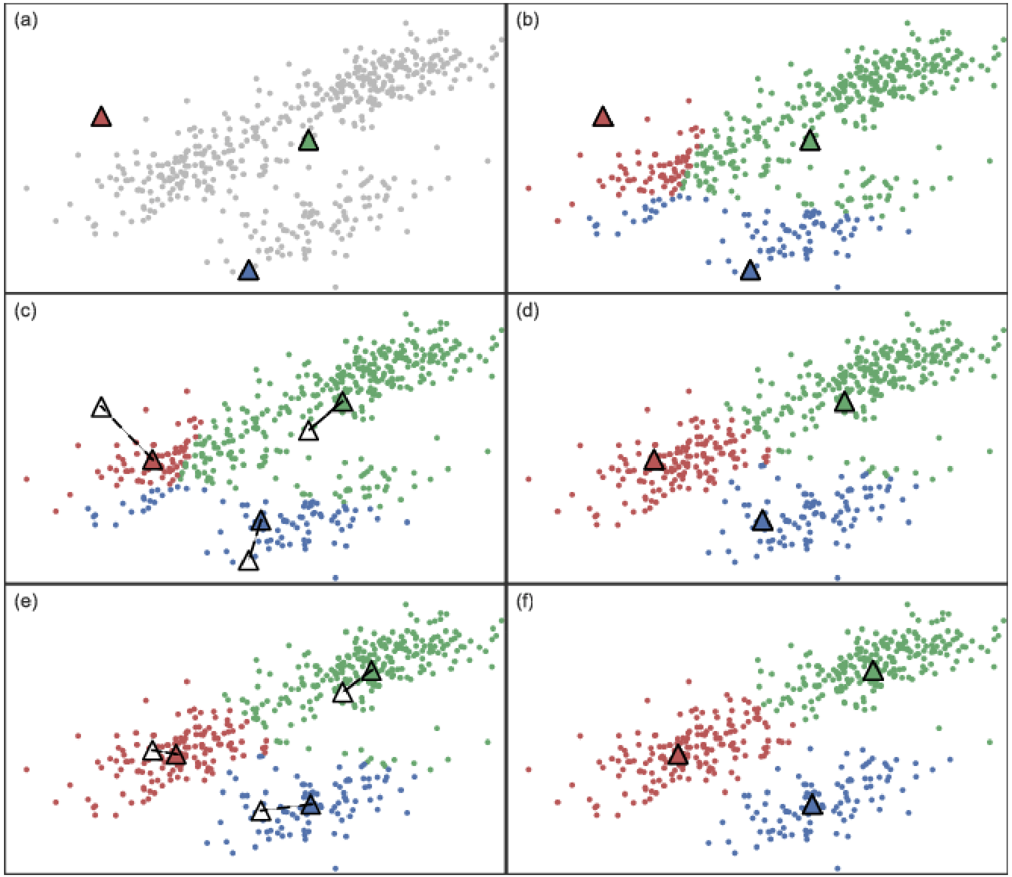
K-means

The gorilla among the point-assignment clustering algorithms

K-means clustering

- Goal: assign each data point to one of *k* clusters such that the total distance of points to their centroids is minimized
- Solved by a simple greedy algorithm (Lloyd's algorithm):
- Locally minimize the "distance" (usually squared Euclidean distance) from data points to their respective centroids:
  - **Find the closest cluster centroid** for each item, and assign it to that cluster.
  - **Recompute the cluster centroid** (the mean of items in the cluster) for each cluster.

K-means clustering



K-means clustering

How long to iterate?

- For fixed number of iterations
- or until no change in assignments
- or until only small change in cluster “tightness” (sum of [squared] distances from points to centroids)

K-means initialization

We need to pick some points for the first round of the algorithm:

- **Random sample:** Pick a random subset of k points from the dataset.
- **K-Means++:** Iteratively construct a random sample with good spacing across the dataset.

Note: Finding an optimal k-means clustering is NP-hard. The above help avoid bad configurations.

K-means++ [link]

Start: Choose first cluster center at random from the data points

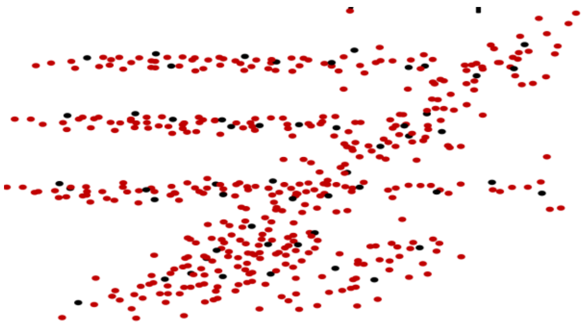
Iterate:

- For every remaining data point x, compute the distance D(x) from x to the closest previously selected cluster center.
- Choose a remaining point x randomly with probability proportional to $D(x)^2$, and make it a new cluster center.

Intuitively, this finds a sample of widely-spaced points from dense regions of the data space, avoiding “collapsing” of the clustering into a few internal centers.

K-means properties

- It's a greedy algorithm with random setup – **solution isn't optimal** and varies significantly with different initial points.
- Very simple convergence proofs.
- **Performance is O(nk) per iteration** — not bad, and can be heuristically improved.
  n = number of points in the dataset, k = number clusters
- Many variants, e.g.
  - Fixed-size clusters
  - Soft clustering
- Works well for data condensation/compression.



K-means drawbacks

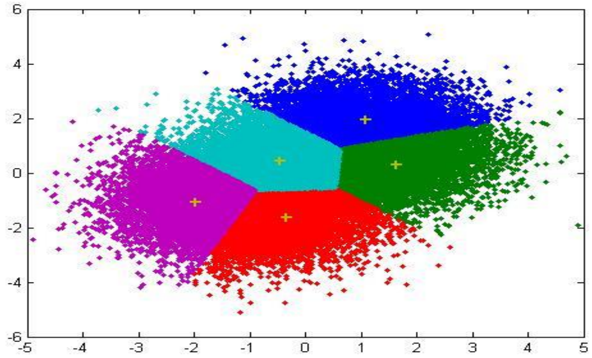
Often terminates at a **local optimum** (mitigated by smart initialization such k-means++, or by re-running multiple times with different initializations)

Need a notion of **mean**

Need to specify **k** (number of clusters) in advance

Doesn’t handle **noisy data and outliers** well

Clusters **only** have **convex shapes**



How to choose k?

Run k-means for k = 1, 2, 3, …

For each data point i, compute "silhouette" $s(i) = \dfrac{b(i) - a(i)}{\max\{a(i), b(i)\}}$

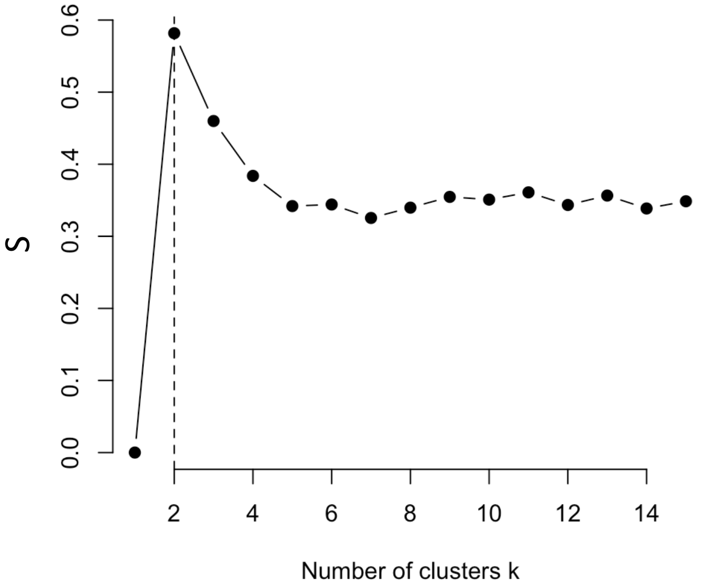
b(i): avg. distance to points in closest other cluster

a(i): avg. distance to points in own cluster

S = average of s(i) over all i

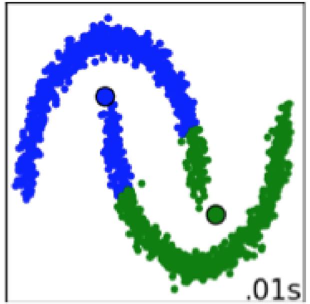
Plot S against k

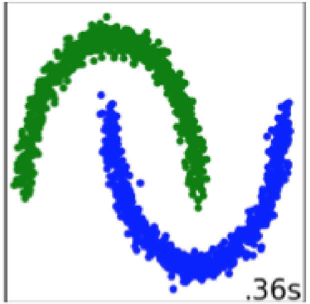
Pick k for which S is greatest



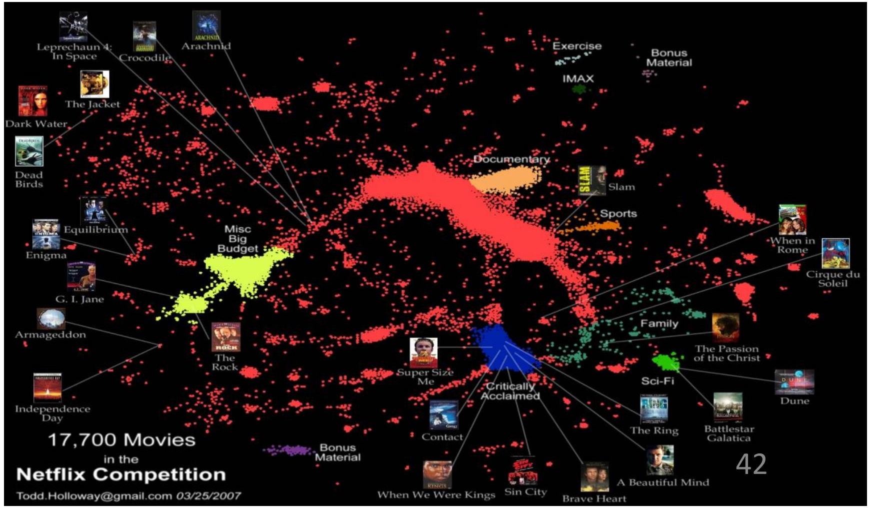
DBSCAN

- "**D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise"
- Motivation: Centroid-based clustering methods like k-means favor clusters that are spherical, and have great difficulty with anything else



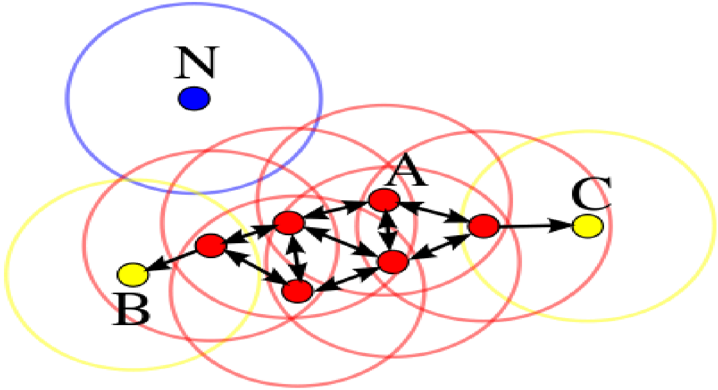


- But with real data we have:



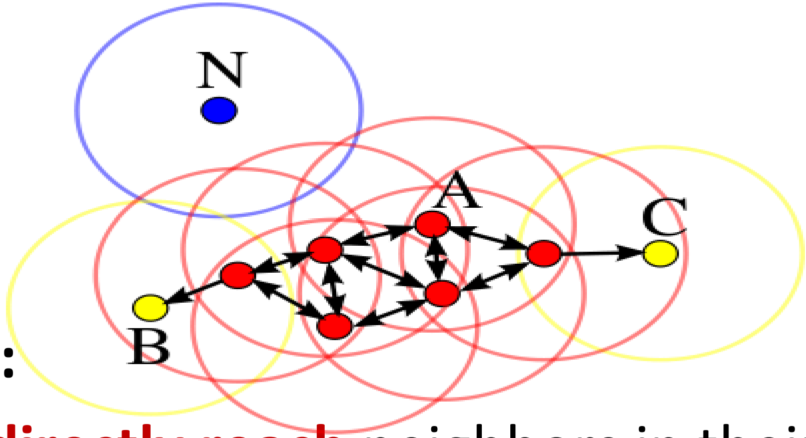
DBSCAN

- DBSCAN performs density-based clustering, and follows the shape of dense neighborhoods of points.



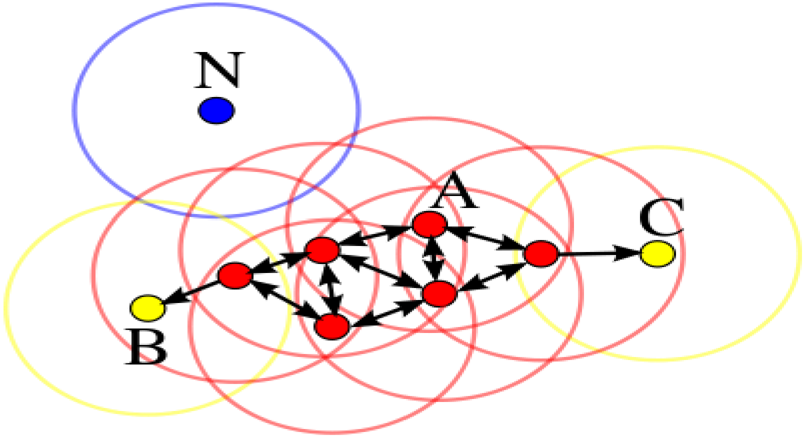
- **Def.:** **core points** have at least *minPts* neighbors in a sphere of diameter ε around them.
- The red points here are core points with at least *minPts* = 3 neighbors in an ε-sphere around them.

DBSCAN



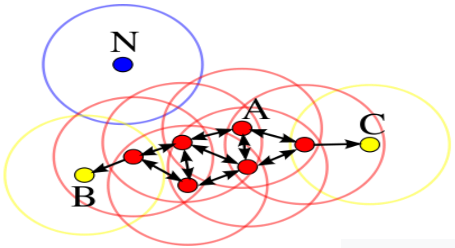
- More **definitions (!):**
  - Core points can **directly reach** neighbors in their ε-sphere
  - From non-core points, no other points can be reached
  - Point q is **density-reachable** from p if there is a series of points $p = p_1, \ldots, p_n = q$ such that p_{i+1} is directly reachable from p_i
  - All points not density-reachable from any other points are

DBSCAN clusters

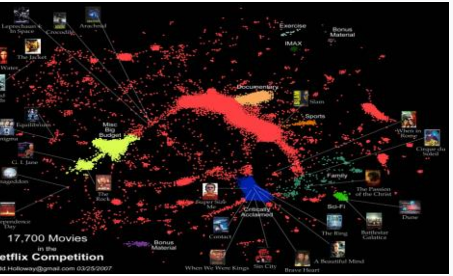


- Even more **definitions:**
  - Points *p*, *q* are **density-connected** if there is a point *o* such that both *p* and *q* are density-reachable from *o*.
  - A **cluster** is a set of points which are **mutually density-connected**.
  - That is, if a point is density-reachable from a cluster point, it is part of the cluster as well.
  - In the above figure, red points are mutually density-reachable; B and C are density-connected; N is an outlier.

DBSCAN algorithm




```
DBSCAN(DB, dist, eps, minPts) {
   C = 0                                                 /* Cluster counter */
   for each point P in database DB {
      if label(P) ≠ undefined then continue              /* Previously processed in inner loop *
      Neighbors N = RangeQuery(DB, dist, P, eps)         /* Find neighbors */
      if |N| < minPts then {                             /* Density check */
         label(P) = Noise                                /* Label as Noise */
         continue
      }
      C = C + 1                                          /* next cluster label */
      label(P) = C                                       /* Label initial point */
      Seed set S = N \ {P}                               /* Neighbors to expand */
      for each point Q in S {                            /* Process every seed point */
         if label(Q) = Noise then label(Q) = C           /* Change Noise to border point */
         if label(Q) ≠ undefined then continue           /* Previously processed */
         label(Q) = C                                    /* Label neighbor */
         Neighbors N = RangeQuery(DB, dist, Q, eps)      /* Find neighbors */
         if |N| ≥ minPts then {                          /* Density check */
            S = S ∪ N                                    /* Add new neighbors to seed set */
         }
      }
   }
}
```



DBSCAN performance

- DBSCAN uses all-pairs point distances, but using an efficient indexing structure, each RangeQuery (for finding neighbors within ε-sphere) takes only O(log n) time
- The algorithm overall can be made to run in **O(n log n)**
- Fast neighbor search becomes progressively harder (higher constants) in higher dimensions

This lecture is based on the course *Applied Data Analysis* (ADA), EPFL. The author is Robert West.