Applied Data Analysis (CS401)

Lecture 9
Learning from data: Unsupervised learning
15 Nov 2023

EPFL

Robert West

Announcements

- Project milestone P2 due on Fri 17 Nov 23:59 (see Ed)
  - Reminder: we won't answer questions asked in the final 24 hours before the deadline
- Homework H2 to be released on Fri 17 Nov
  - Due two weeks later, on Fri 1 Dec
- Friday's lab session:
  - Quiz 8
  - Exercises on unsupervised learning

Give us feedback on this lecture here:
https://go.epfl.ch/ada2023-lec9-feedback

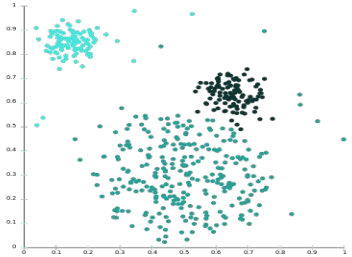
- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …

Machine learning

- **Supervised:** We are given input/output pairs (X, y) (a.k.a. "samples") that are related via a function $y = f(X)$. We would like to "learn" f, and evaluate it on new data.
  - Discrete y (class labels): **"classification"**
  - Continuous y: **"regression"** (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 (cluster labels): **"clustering"**
  - Continuous y: **"dimensionality reduction"** (e.g., matrix factorization, unsupervised neural networks)

The clustering problem

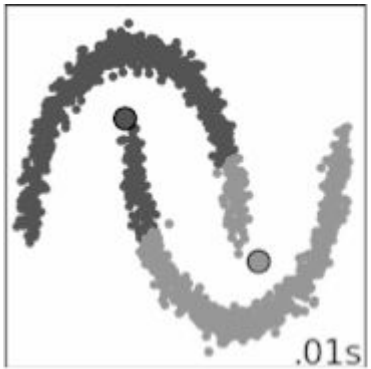
- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, such that
  - members of a cluster are close (i.e., similar) to each other
  - members of different clusters are far apart from each other
- **Usually:**
  - Points live in a high-dimensional space
  - Similarity is defined via a distance measure
    - Euclidean, cosine, Jaccard, edit distance, … (cf. lecture 7)

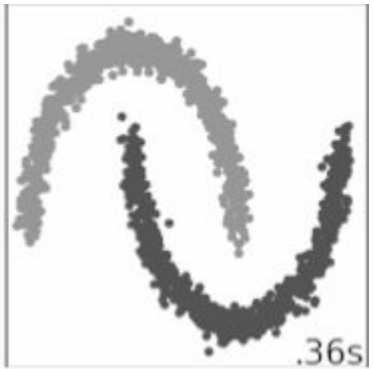


Characteristics of clustering methods

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

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



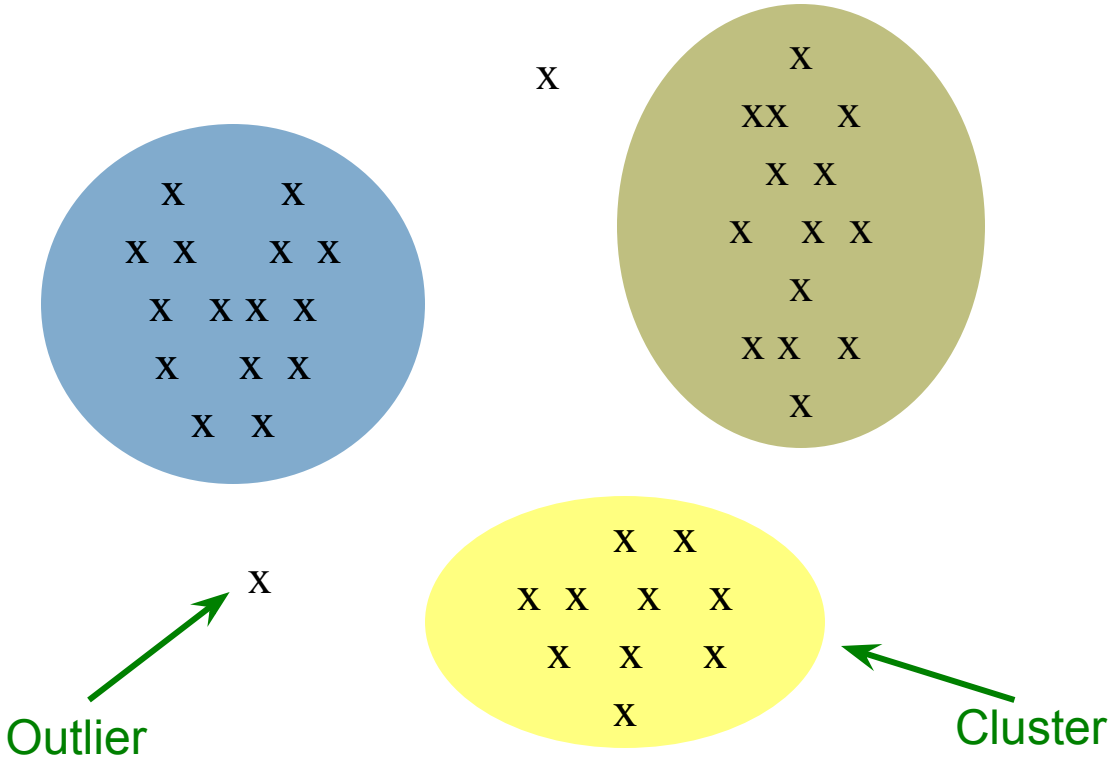


Characteristics of clustering methods

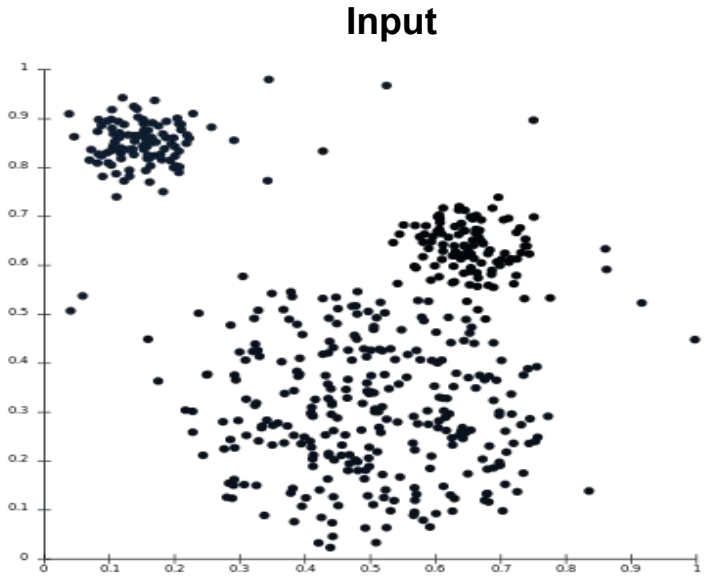
Robustness: sensitivity to noise and outliers, sensitivity to the processing order

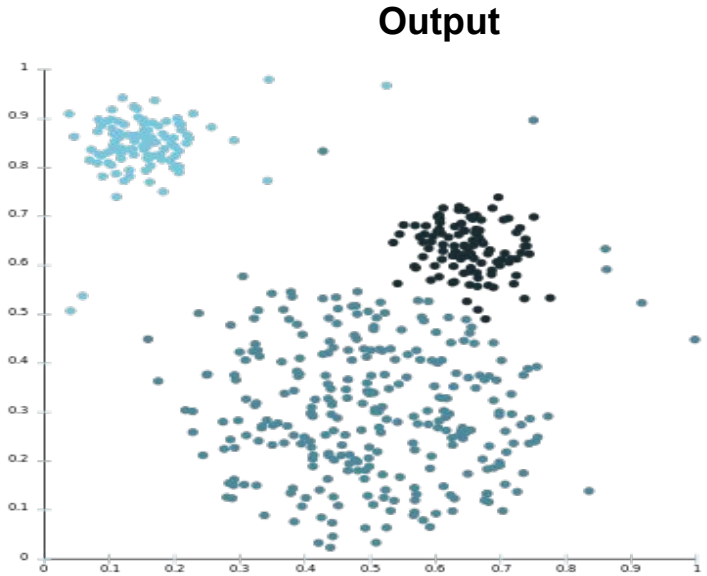
User interaction: incorporation of user constraints (e.g., number of clusters, max size of clusters), interpretability and usability

Example: clusters & outliers



A typical clustering example



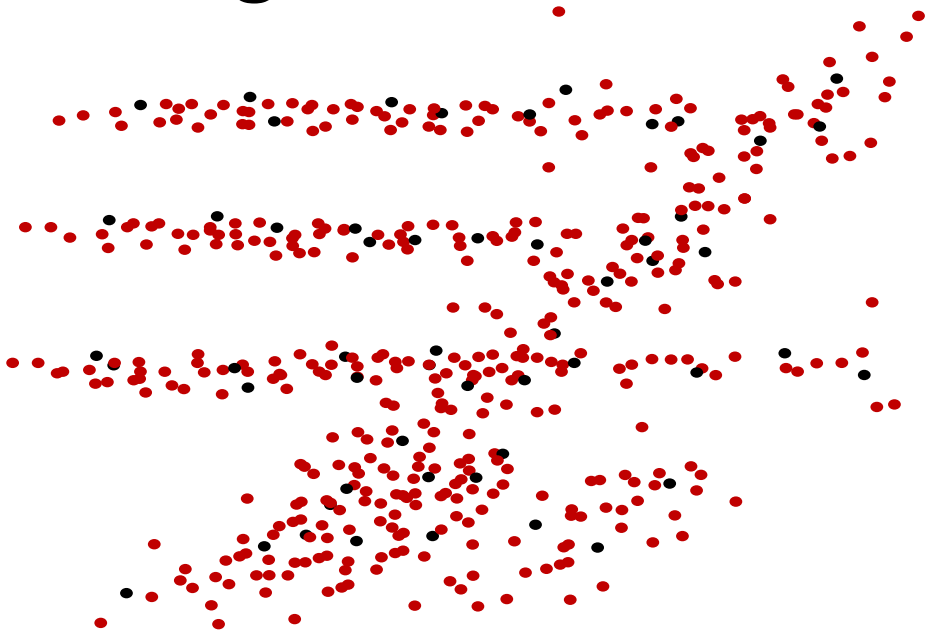


Note: Above is 2D; real scenarios often much more high-dimensional, e.g., 10,000-dimensional for 100x100 images.

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
- Supporting feature discretization for supervised learning (cf. lecture 8)
- 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.

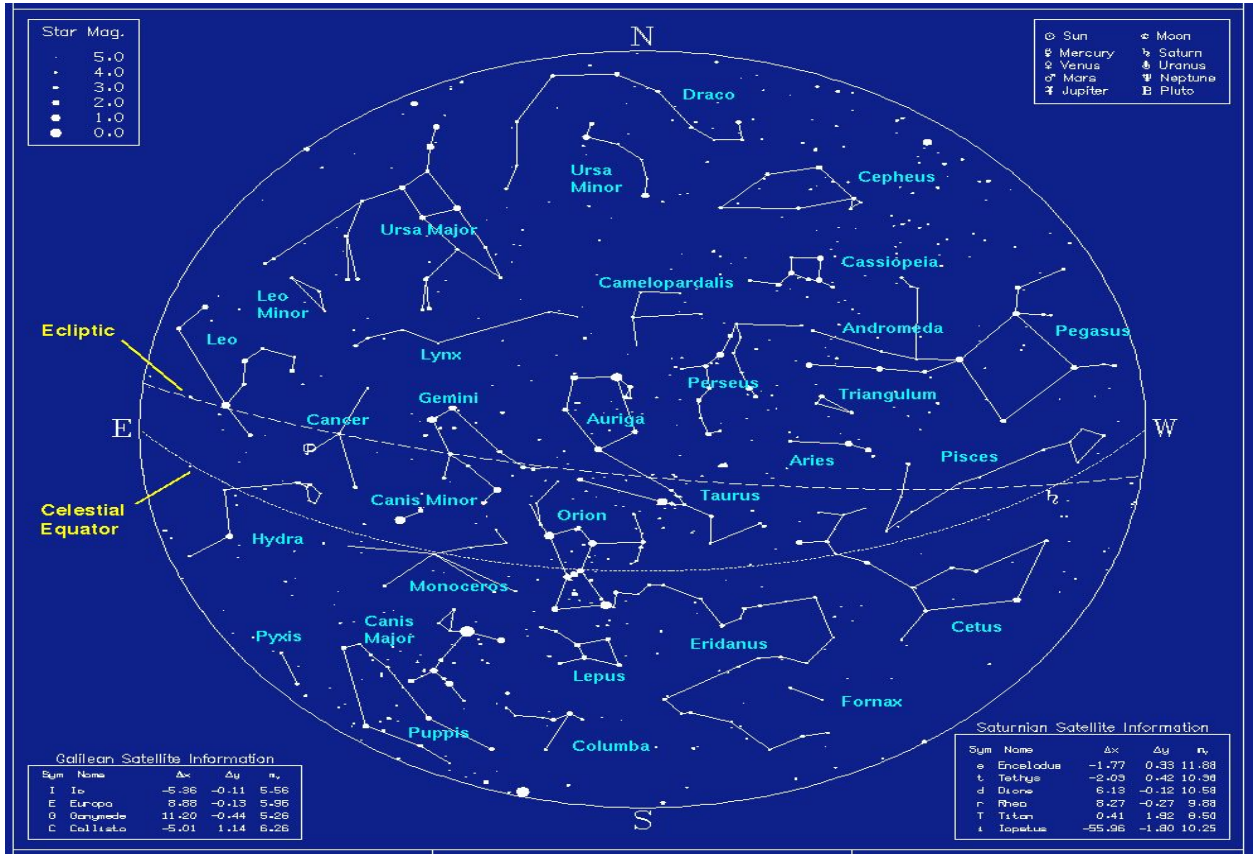
Beware of “cluster bias”!

- Human beings conceptualize the world through categories represented as exemplars or prototypes



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

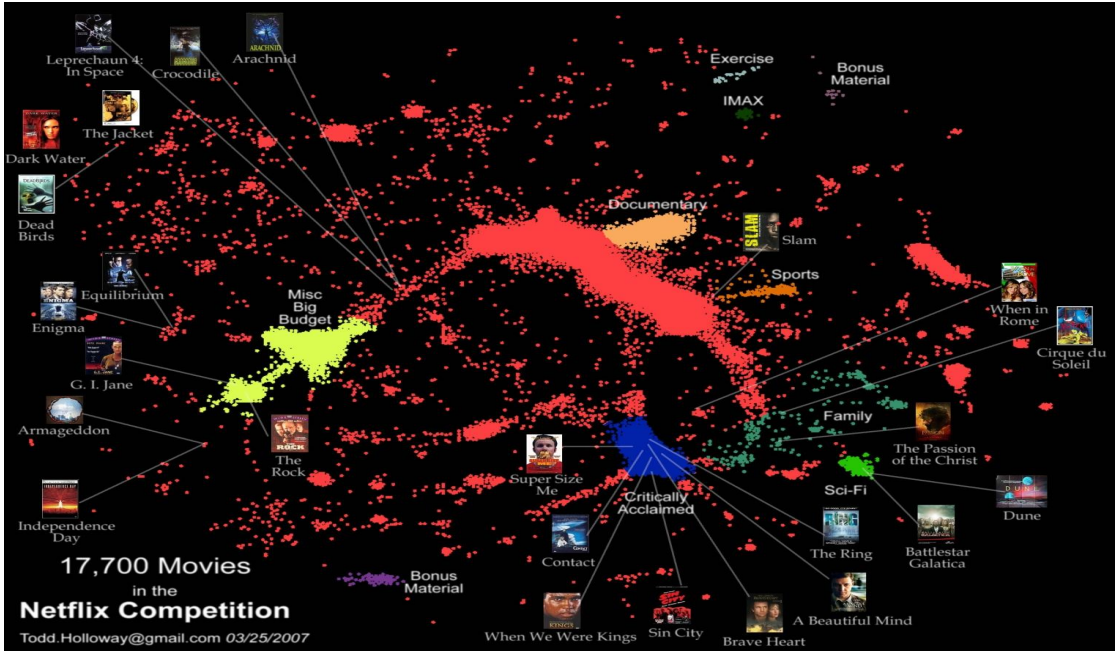
Cluster bias



Cluster bias

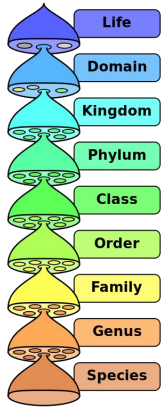
- **Clustering is used more than it should be**, because people assume an underlying domain has discrete classes in it
  - Especially true for characteristics of people, e.g., Myers-Briggs personality types like "ENTP".
- In reality the underlying data is often **continuous**.
- In such cases, continuous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

Netflix

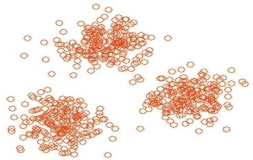


- Central portion: more of a continuum than discrete clusters
- Other methods (e.g., dimensionality reduction) may be more appropriate than discrete clustering models

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...

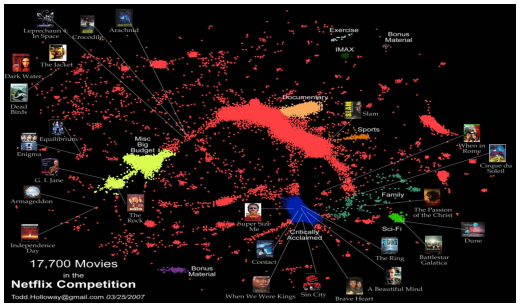
- … many applications involve not 2, but hundreds or thousands of dimensions (and large amounts of data)
- High-dimensional spaces are different ("curse of dimensionality"): volume grows exponentially w/ #dims; space is sparsely populated; clusters become less tight

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: movies

- **Intuitively: Movies divide into categories/genres, and customers prefer a few categories**
  - But what are categories really?
  - ⟶ take a data-driven approach!



- Represent a movie by a set of customers who watched it (“collaborative filtering”)
- **Idea:** Similar movies have similar sets of customers, and vice-versa

Clustering problem: movies

Space of all movies:

- Think of a space with one dimension for each customer
  - Values in a dimension may be 0 or 1 only
  - A movie is a point in this space (x_1, x_2,…, x_n), where x_i = 1 iff the i-th customer watched the movie
- For Amazon/Netflix, the dimensionality is in the millions
- **Task:** Find clusters of similar movies

Clustering problem: documents

Finding topics:

- Represent a document by a vector (x_1, x_2,…, x_n), where x_i = 1 iff the *i*-th word appears in the document (in any position)

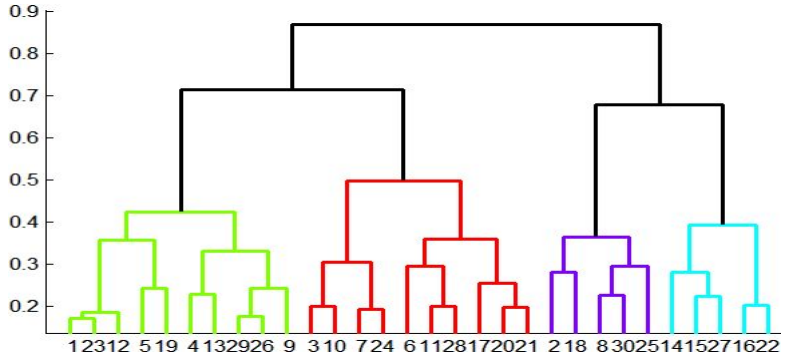
- **Idea:** Documents with similar sets of words are about same topic

Cosine, Jaccard, Euclidean distances

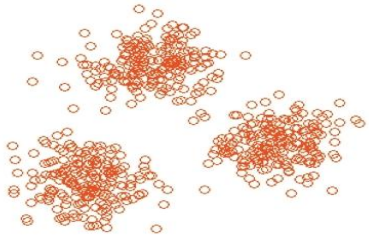
- In both examples (movies, documents) we have a choice when we thinking of data points as sets of features (users, words):
  - **Sets as vectors:**
    - Measure similarity via **Euclidean distance**
    - Measure similarity via **cosine distance**
  - **Sets as sets:**
    - Measure similarity via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

Overview: Methods of clustering

- **Hierarchical methods:**
  - **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

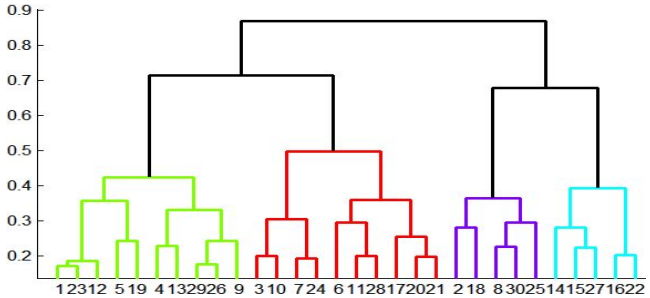


- **Flat methods (a.k.a. point-assignment methods):**
  - Maintain a set of clusters
  - Assign points to "nearest" cluster; recompute clusters; repeat



Agglomerative hierarchical clustering

- **Key operation:**
  Repeatedly combine two nearest clusters

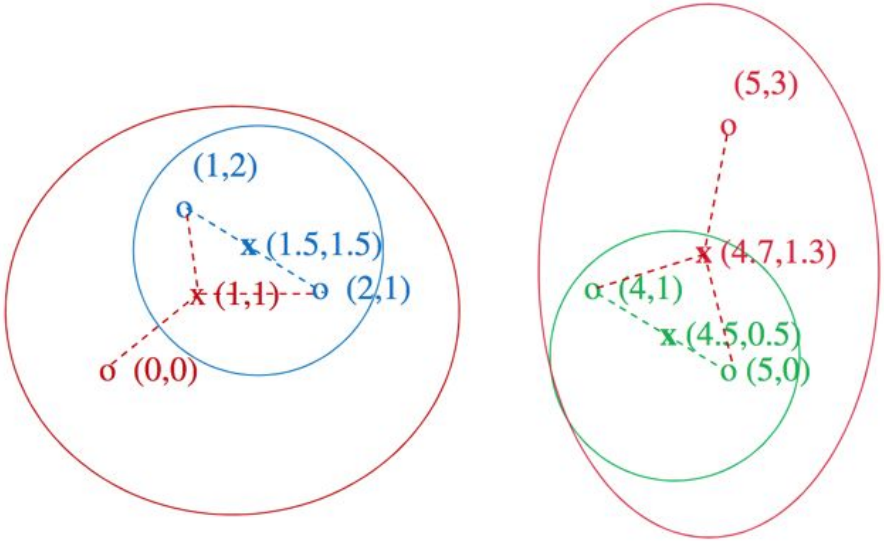


- **Three important questions:**
  - **(1)** How to represent a cluster of more than one point?
  - **(2)** How to determine the "nearness" of clusters?
  - **(3)** When to stop combining clusters?

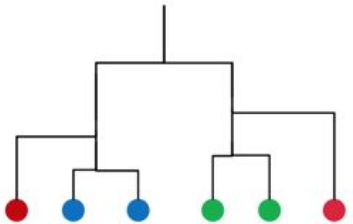
Agglomerative hierarchical clustering

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

Example: Hierarchical clustering

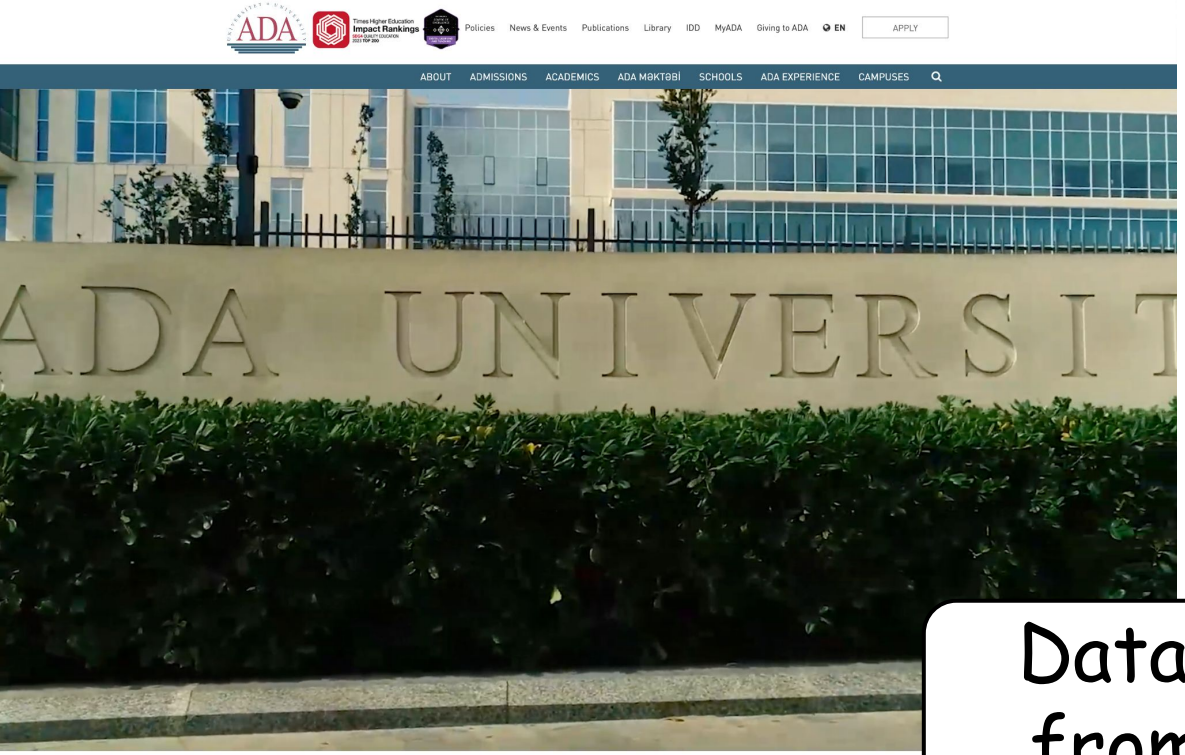


Data:
o … data point
x … centroid



Dendrogram

Commercial break

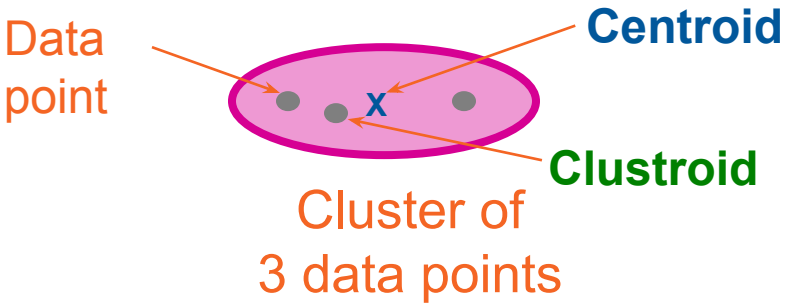


UNİVERSİTET ◦ UNIVERSITY
ADA
ada.edu.az

Data science from A to Z

Non-Euclidean case: clustroids

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

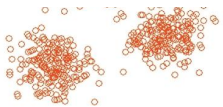


Centroid is the avg. of all data points in the cluster. This means centroid is an "artificial" point.
Clustroid is an **existing** data point that is "closest" to all other points in the cluster.

Non-Euclidean case: cluster “nearness”

- **(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 clustroids; etc.



  - **Approach 2:**

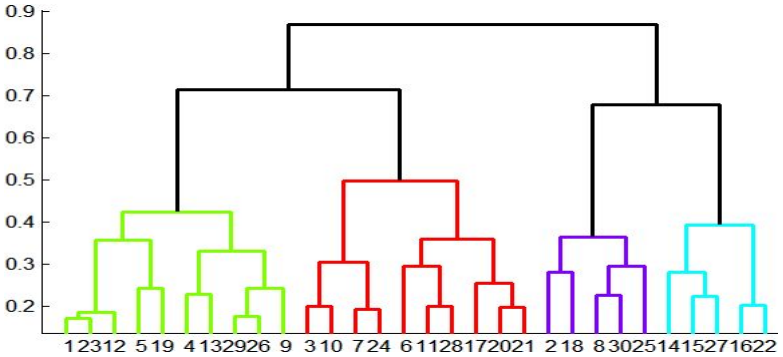
    Pick a notion of “**cohesion**” (“tightness”) of a cluster

    Then: nearness of clusters = cohesion of their *union*

Cohesion

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

How many branching points are there in a dendrogram for a dataset with *N* data points?



POLLING TIME

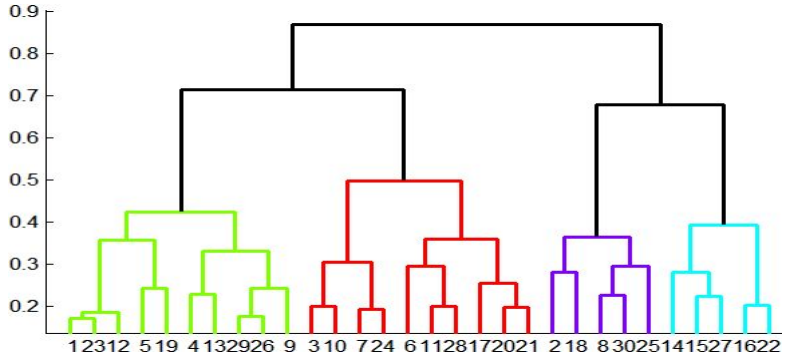
- Scan QR code or go to https://web.speakup.info/room/join/66626

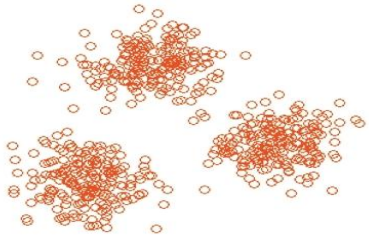
Implementation

- **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

Overview: Methods of clustering

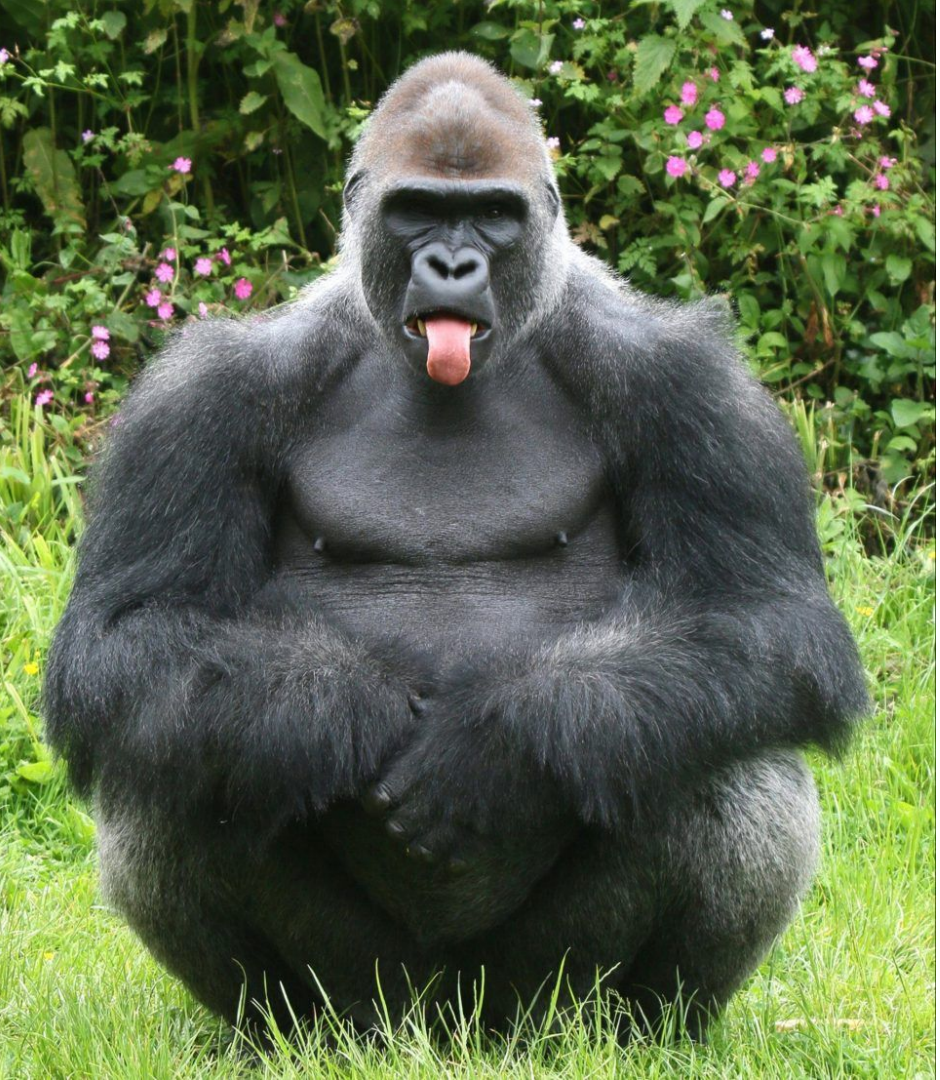
- **Hierarchical methods:**
  - **Agglomerative** (bot
    - Initially, each cluster
    - epeatedly e the two
    - est” rs into one
  - **Div** own):
    - Start one cluster and recursively split it





- **Flat methods (a.k.a. point-assignment me ods):**
  - Maintain a set of clusters
  - Assign points to “nearest” cluster; recompute clusters; repeat

NEXT



K-means

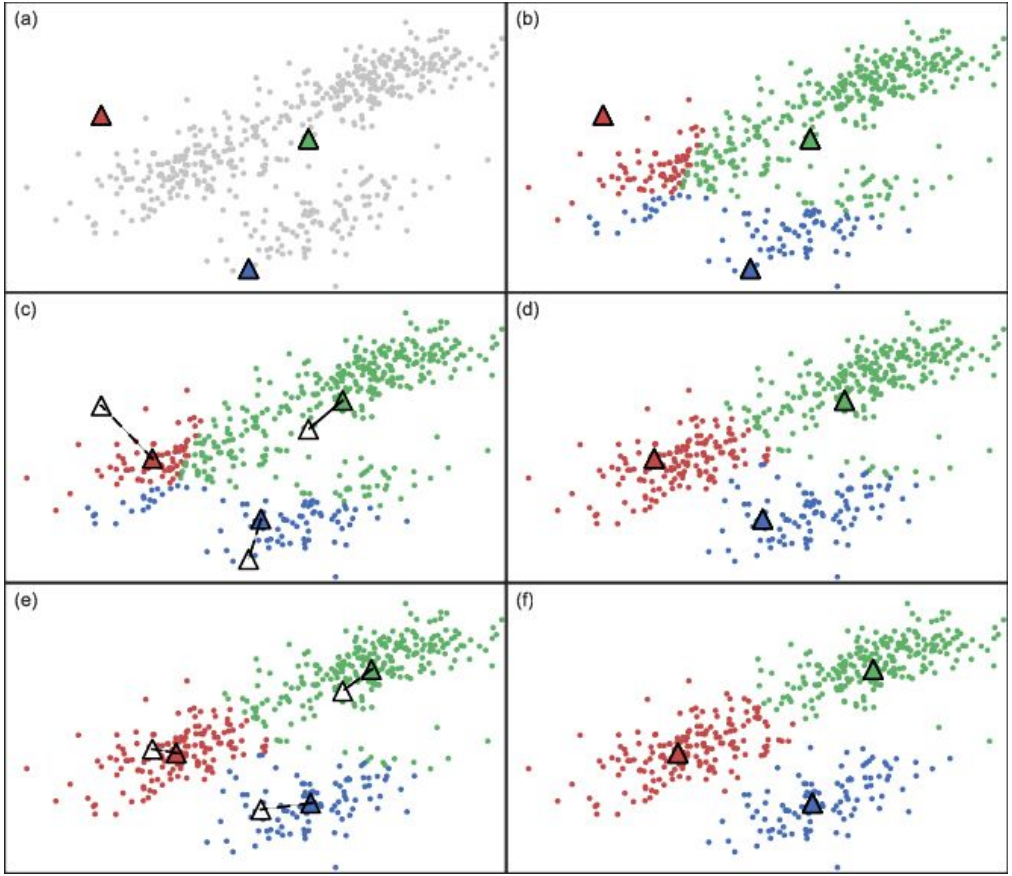
The gorilla among the point-assignment clustering algorithms

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 data point, and assign the point to that cluster.
  - **Recompute the cluster centroid** (the mean of data points 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 (guaranteed to happen)
- 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

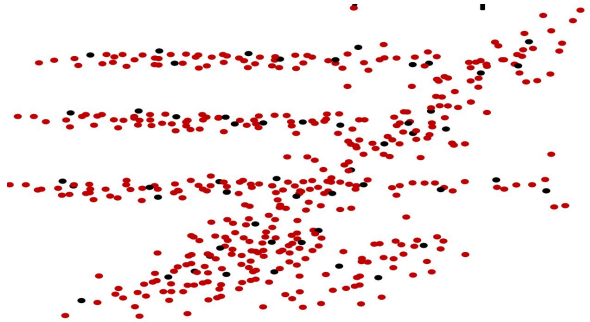
Iterate:

- 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

- Greedy algorithm with random initialization – **solution may be suboptimal** & vary 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

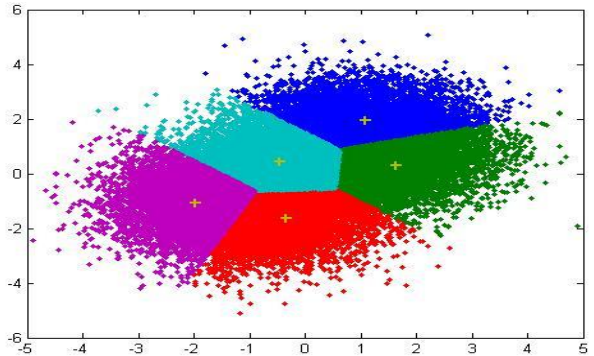
Often terminates at a **local but non-global optimum** (mitigated by smart initialization such k-means++, or by re-running multiple times with different initializations)

Requires the notion of a **mean**

Requires specifying ***k*** (number of clusters) in advance

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

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



How to choose *k*?

For *k* = 1, 2, 3, …

Run k-means with *k* clusters

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

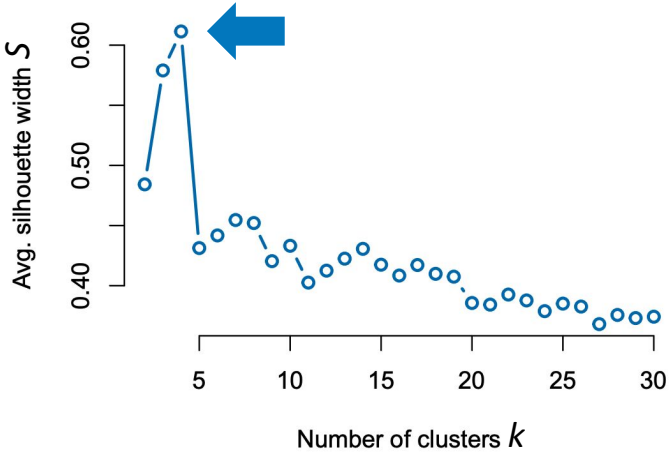
S = average of *s*(*i*) over all *i*

Plot *S* against *k*

Pick *k* for which *S* is greatest

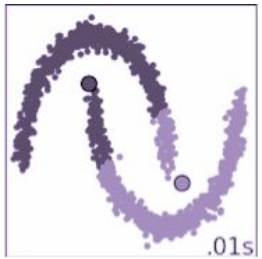
b(*i*): avg. distance to points in closest other cluster

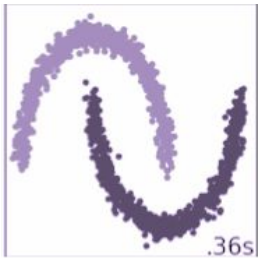
a(*i*): avg. distance to points in own cluster



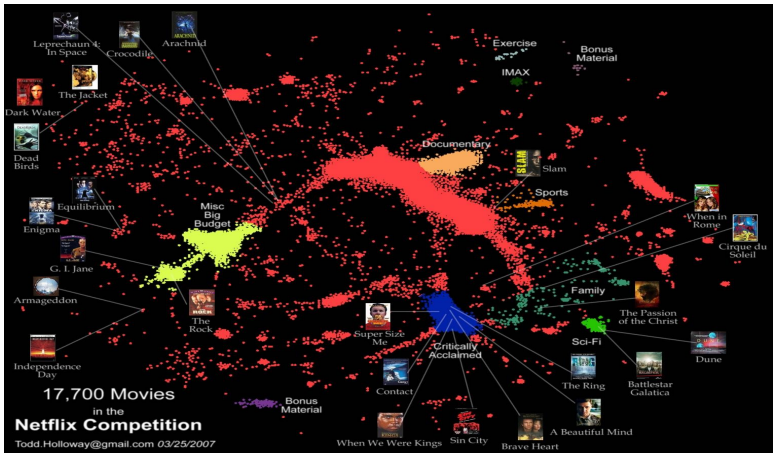
DBSCAN

- "**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



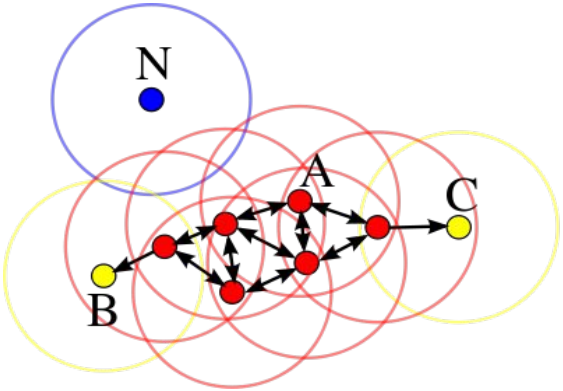


- But with real data we often have:



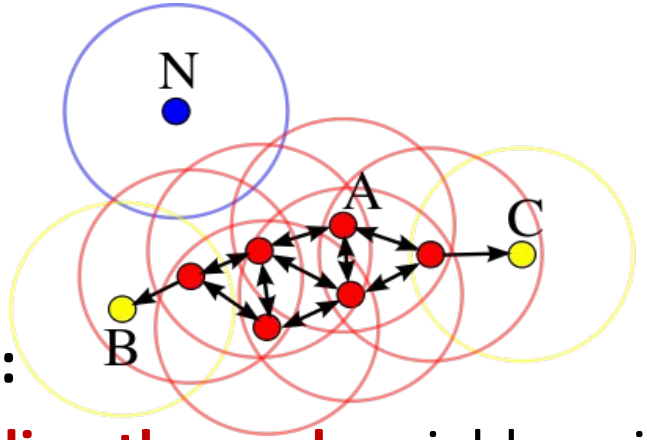
DBSCAN

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



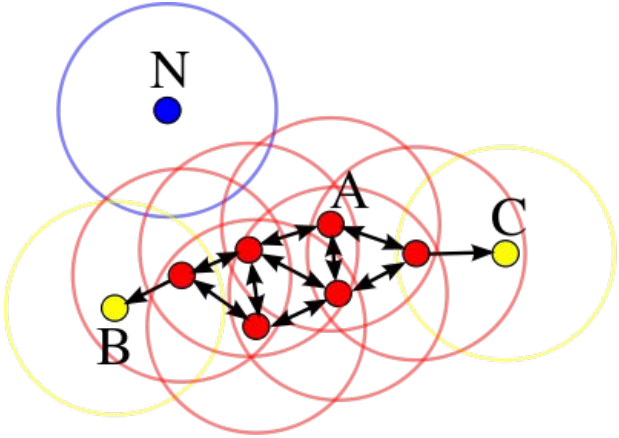
- **Def.: core points** have at least *minPts* neighbors in a sphere of diameter ε around them.
- The red points here are core points with at least *minPts* = 3 neighbors in an ε-sphere around them.

DBSCAN

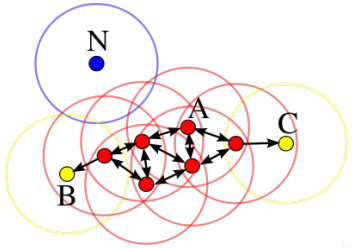


- More **definitions (!):**
  - Core points can **directly reach** neighbors in their ε-sphere
  - From non-core points, no other points can be reached
  - Point *q* is **density-reachable** from *p* if there is a series of points $p = p_1, \ldots, p_n = q$ such that p_{i+1} is directly reachable from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

DBSCAN clusters

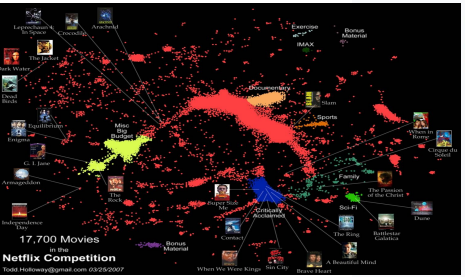


- Even more **definitions:**
  - Points *p*, *q* are **density-connected** if there is a point *o* such that both *p* and *q* are density-reachable from *o*.
  - A **cluster** is a set of points that are **mutually density-connected**.
  - That is, if a point is density-reachable from a cluster point, 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 U 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, due to the

Give us feedback on this lecture here:
https://go.epfl.ch/ada2023-lec9-feedback

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …

What are the most important statistical ideas of the past 50 years?*

Andrew Gelman† and Aki Vehtari‡

3 June 2021

Abstract

We review the most important statistical ideas of the past half century, which we categorize as: counterfactual causal inference, bootstrapping and simulation-based inference, overparameterized models and regularization, Bayesian multilevel models, generic computation algorithms, adaptive decision analysis, robust inference, and exploratory data analysis. We discuss key contributions in these subfields, how they relate to modern computing and big data, and how they might be developed and extended in future decades. The goal of this article is to provoke thought and discussion regarding the larger themes of research in statistics and data science.

https://arxiv.org/abs/2012.00174