


서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

Clustering

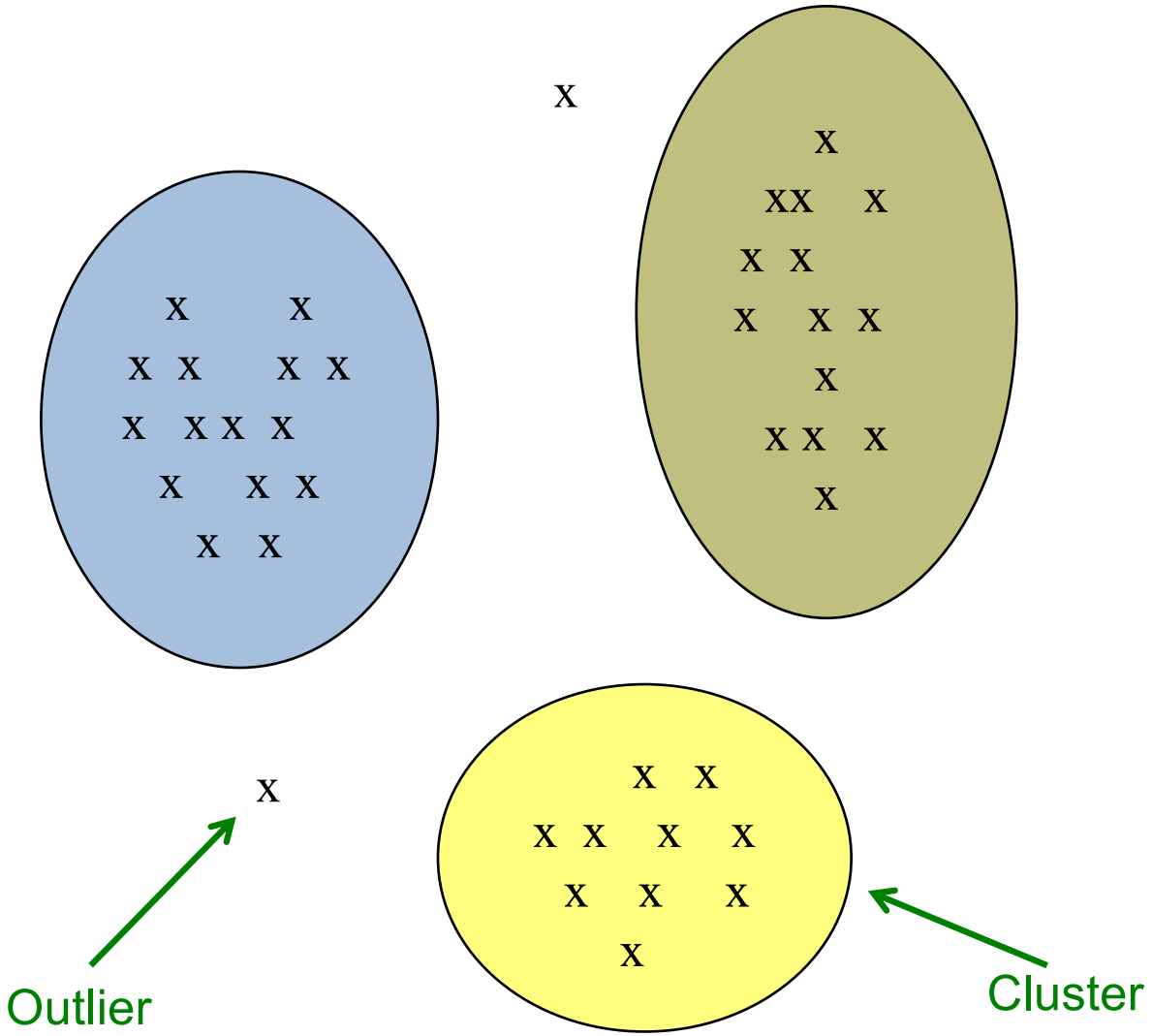
High Dimensional Data

- Given a cloud of data points we want to understand its structure

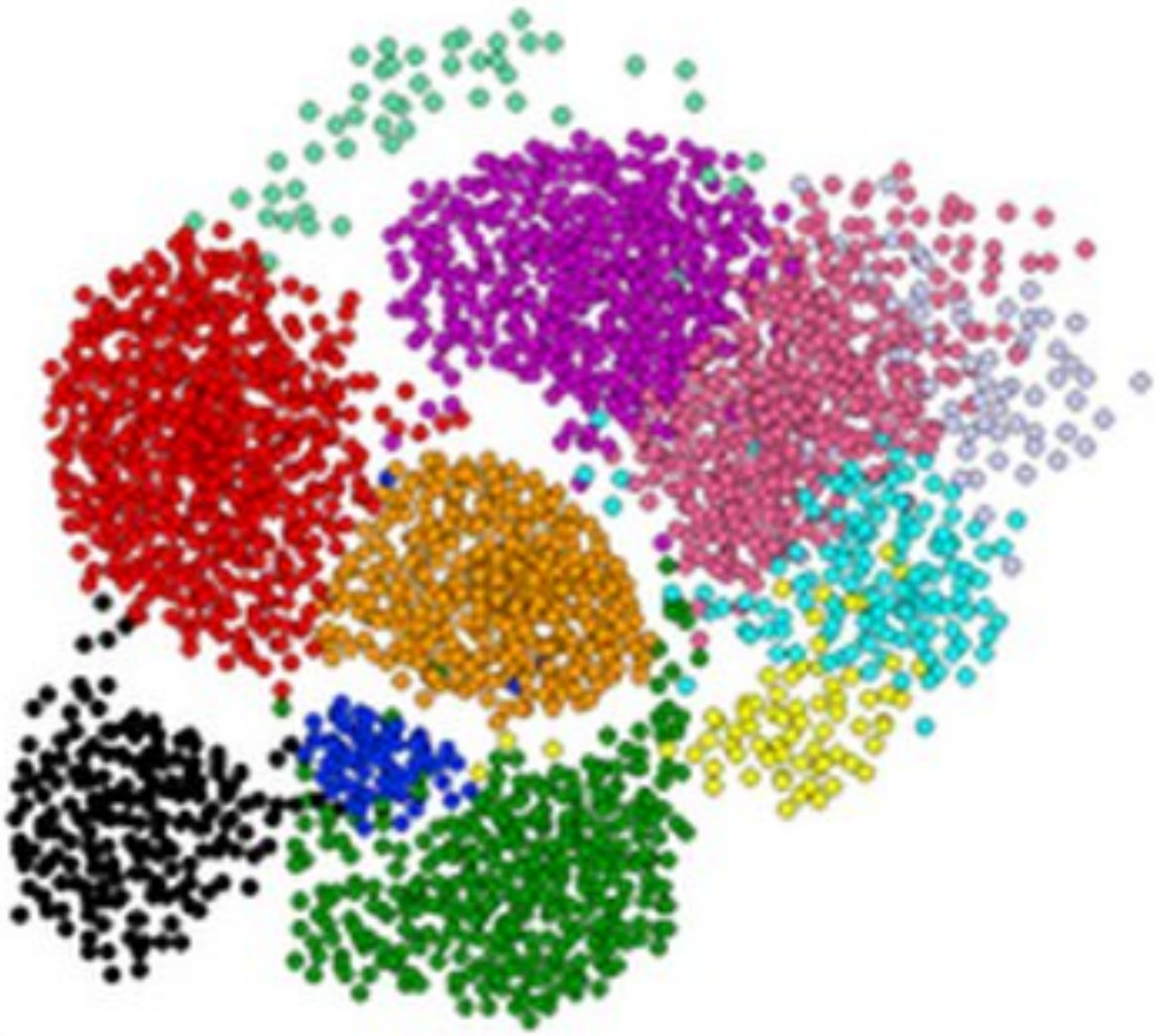
The Problem of Clustering

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of *clusters*, so that
  - Members of a cluster are close/similar to each other
  - Members of different clusters are dissimilar
- Usually:
  - Points are in a high-dimensional space
  - Similarity is defined using a distance measure
    - Euclidean, Cosine, Jaccard, edit distance, …

Example: Clusters & Outliers



Clustering is a hard problem!



Why is it hard?

- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving

- Many applications involve not 2, but 10 or 10,000 dimensions
  - Example: clustering documents by the vector of word counts (one dimension for each word).
- High-dimensional spaces look different: Almost all pairs of points are at about the same distance

Clustering Problem: Music CDs

- Intuitively: Music divides into categories, and customers prefer a few categories
  - But what are categories really?
- Represent a CD by a set of customers who bought it:
- Similar CDs have similar sets of customers, and vice-versa

Clustering Problem: Music CDs

Space of all CDs:

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

Clustering Problem: Documents

Finding topics:

- Represent a document by a vector $(x_1, x_2, \cdots, x_k)$, where $x_i = 1$ iff the i^{th} word (in some order) appears in the document
  - It actually doesn't matter if k is infinite; i.e., we don't limit the set of words

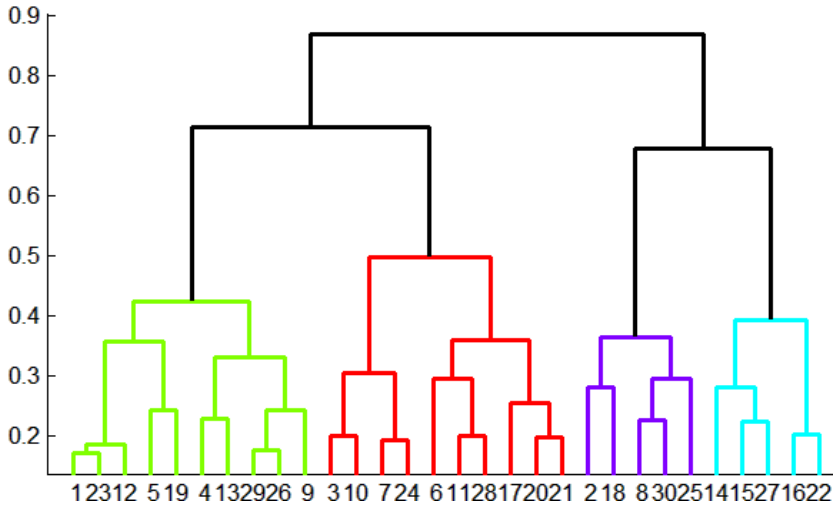
- Documents with similar sets of words may be about the same topic

Cosine, Jaccard, and Euclidean

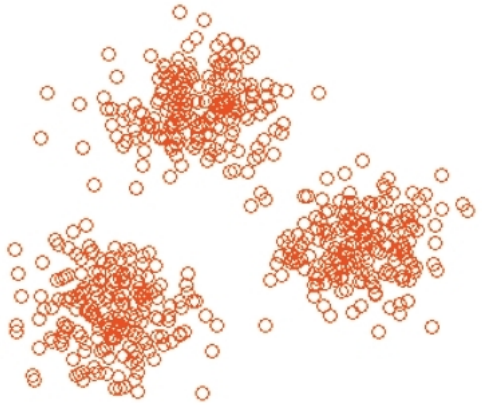
- As with CDs we have a choice when we think of documents as sets of words or shingles:
  - Sets as vectors: Measure similarity by the **Cosine distance**
  - Sets as sets: Measure similarity by the **Jaccard distance**
  - Sets as points: Measure similarity by **Euclidean distance**

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

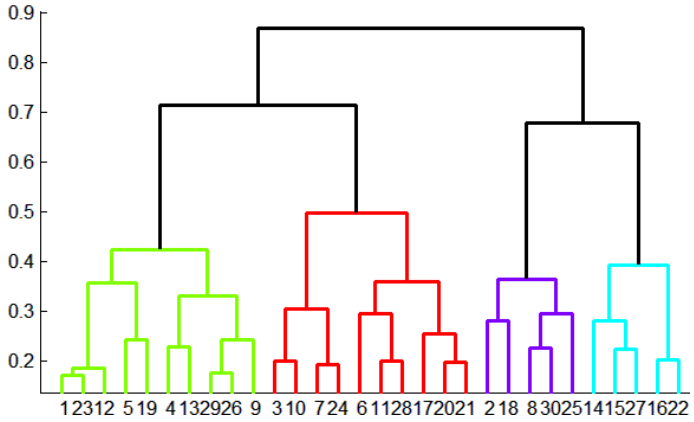


Hierarchical Clustering

- Key operation:

  Repeatedly combine

  two nearest clusters



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

Hierarchical Clustering

- Key operation: Repeatedly combine two nearest clusters

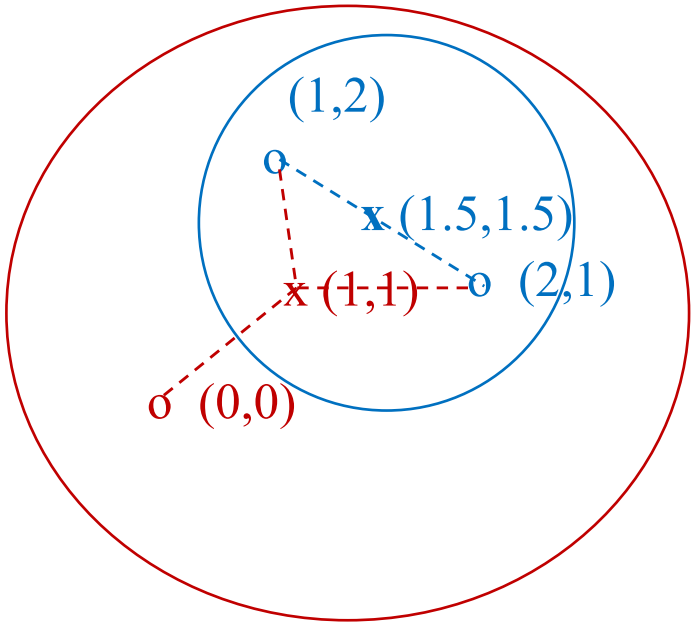
(1) How to represent a cluster of many points?

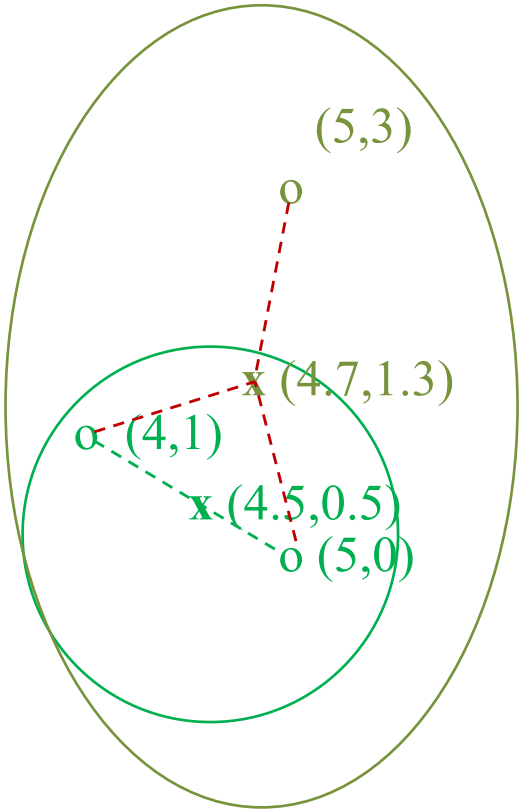
- Key problem: As you merge clusters, how do you represent the "location" of each cluster, to tell which pair of clusters is closest?
- Euclidean case: each cluster has a *centroid*
- average of its (data)points

(2) How to determine "nearness" of clusters?

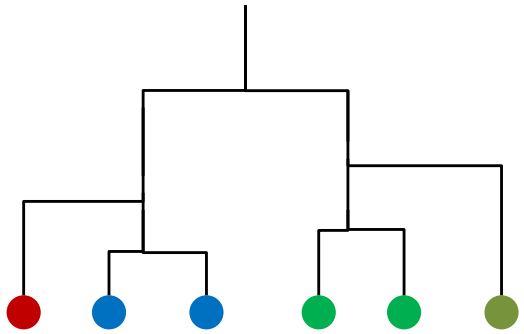
- Measure cluster distances by distances of centroids

Example: Hierarchical clustering



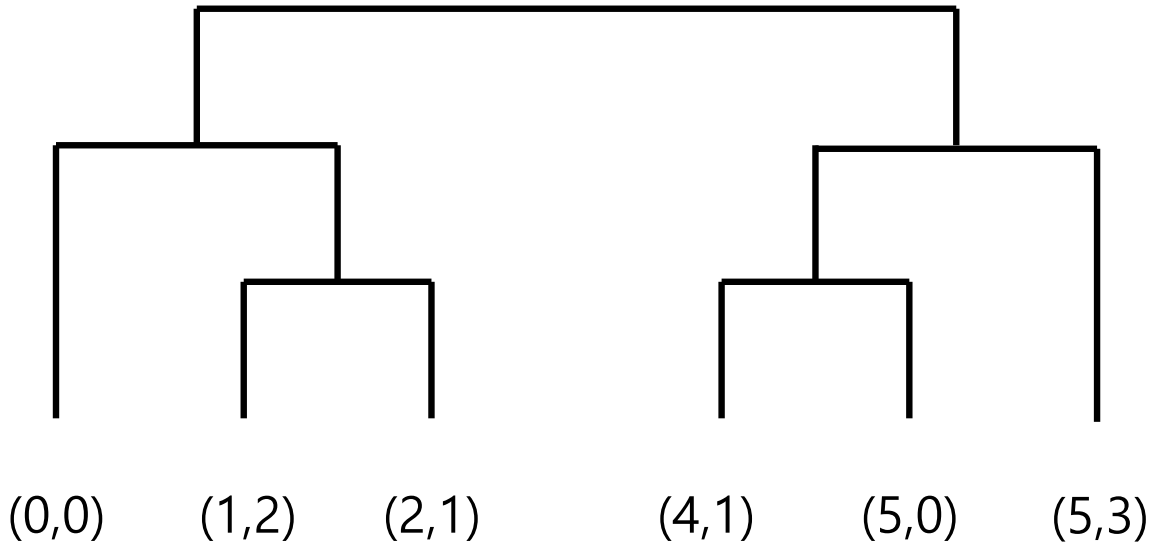


Data:
o … data point
x … centroid



Dendrogram

Dendrogram



And in the Non-Euclidean Case?

What about the Non-Euclidean case?

- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points

- Approach 1:

  (1) How to represent a cluster of many points?

  clustroid = (data)point "***closest***" to other points

  (2) How do you determine the "nearness" of clusters?

  Treat clustroid as if it were centroid,

  when computing inter-cluster distances

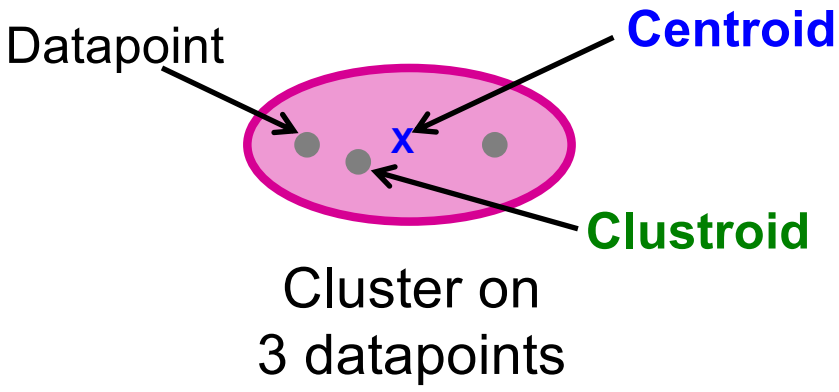
"Closest" Point?

- (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
  - Smallest sum of squares of distances to other points
    - For distance metric d clustroid c of cluster C is:

$$\min_{c} \sum_{x \in C} d(x,c)^2$$



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

Defining "Nearness" of Clusters

(2) How do you determine the "nearness" of clusters?

- Approach 2:

  Intercluster distance = minimum of the distances between any two points, one from each cluster

- Approach 3:

  Pick a notion of "cohesion" of clusters,

  e.g., maximum distance from the clustroid

  - Merge clusters whose *union* is most cohesive

Cohesion

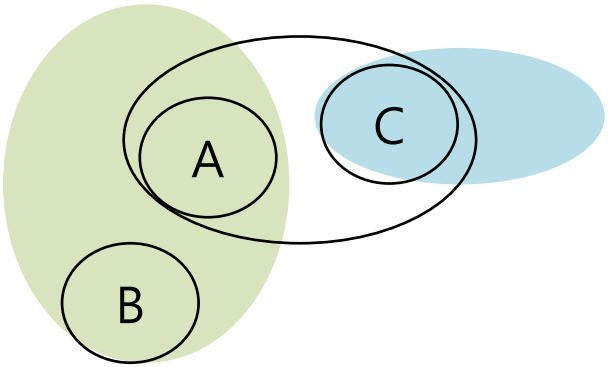
- Approach 3.1: Use the diameter of the merged cluster = maximum distance between points in the cluster
- Approach 3.2: Use the average distance between points in the cluster
- Approach 3.3: Use a density-based approach
  - Take the diameter or avg. distance, e.g., and divide by the number of points in the cluster

Which is Best?

- It really depends on the shape of clusters.
  - Which you may not know in advance.
- Example: we'll compare two approaches:
  1. Merge clusters with smallest distance between centroids (or clustroids for non-Euclidean).
  2. Merge clusters with the smallest distance between two points, one from each cluster.

Case 1: Convex Clusters

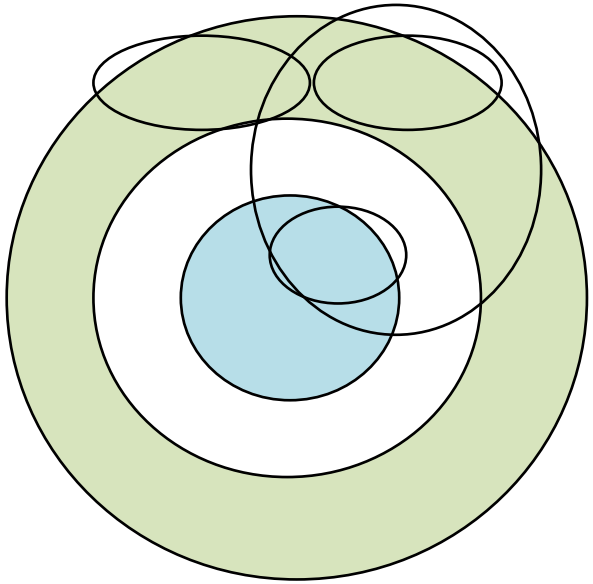
- Centroid-based merging works well.
- But merger based on closest members might accidentally merge incorrectly.



A and B have closer centroids than A and C, but closest points are from A and C.

Case 2: Concentric Clusters

- Linking based on closest members works well.
- But Centroid-based linking might cause errors.



Implementation

- Naïve implementation of hierarchical clustering:
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - $O(N^3)$
- 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

k–means clustering

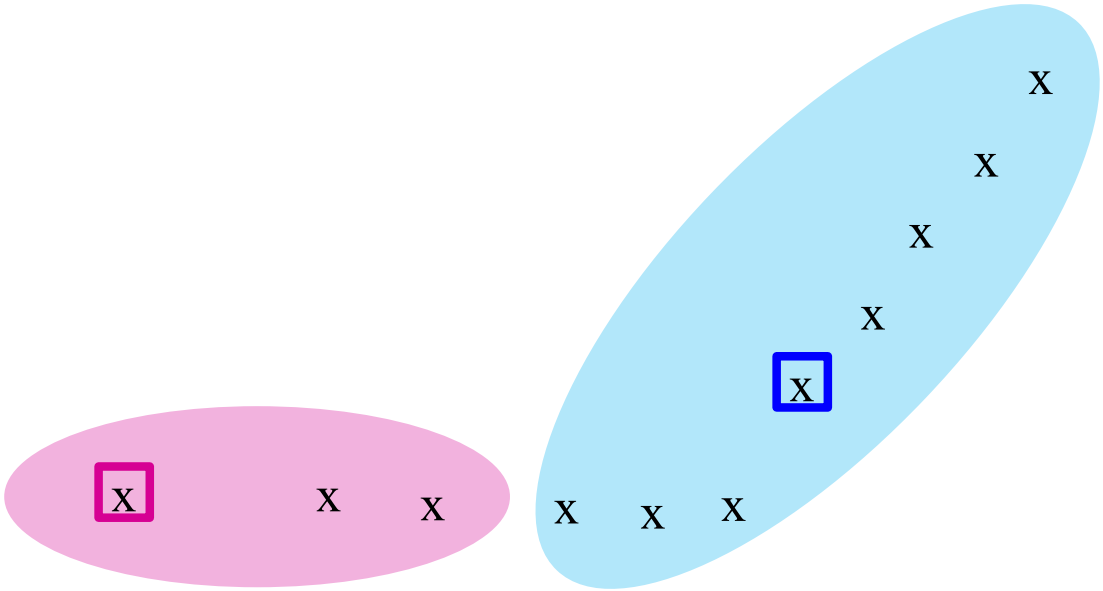
$k-$means Algorithm(s)

- Assumes Euclidean space/distance
- Start by picking k, the number of clusters
- Initialize clusters by picking one point per cluster
  - Example: Pick one point at random, then $k-1$ other points, each as far away as possible from the previous points

Populating Clusters

- 1) For each point, place it in the cluster whose current centroid it is nearest
- 2) After all points are assigned, update the locations of centroids of the *k* clusters
- 3) Reassign all points to their closest centroid
  - Sometimes moves points between clusters
- Repeat 2 and 3 until convergence
  - **Convergence:** Points don't move between clusters and centroids stabilize

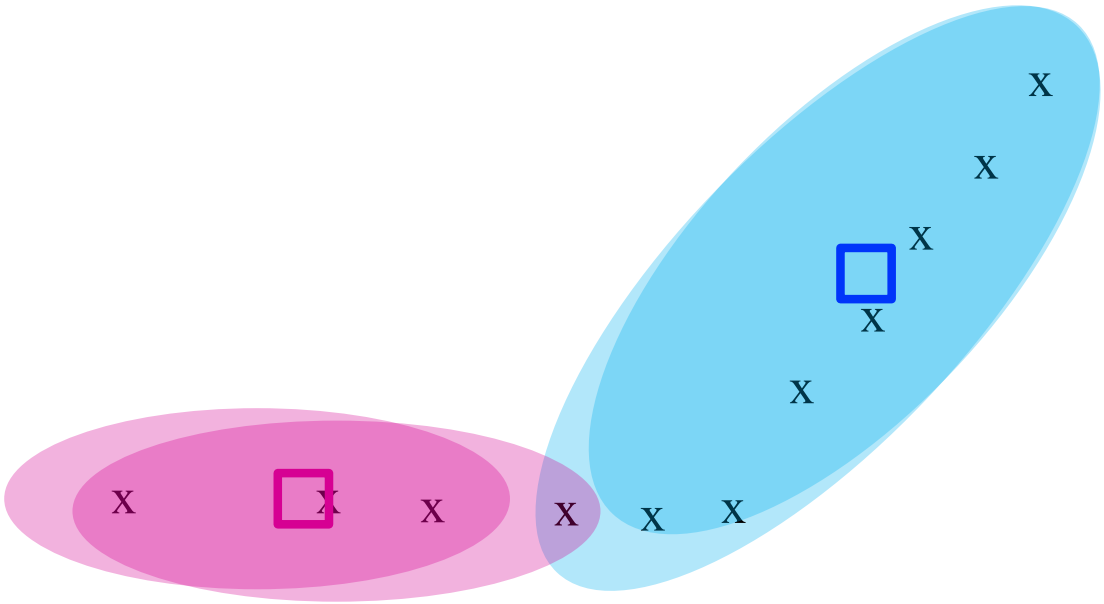
Example: Assigning Clusters



x … data point
□ … centroid

Clusters after round 1

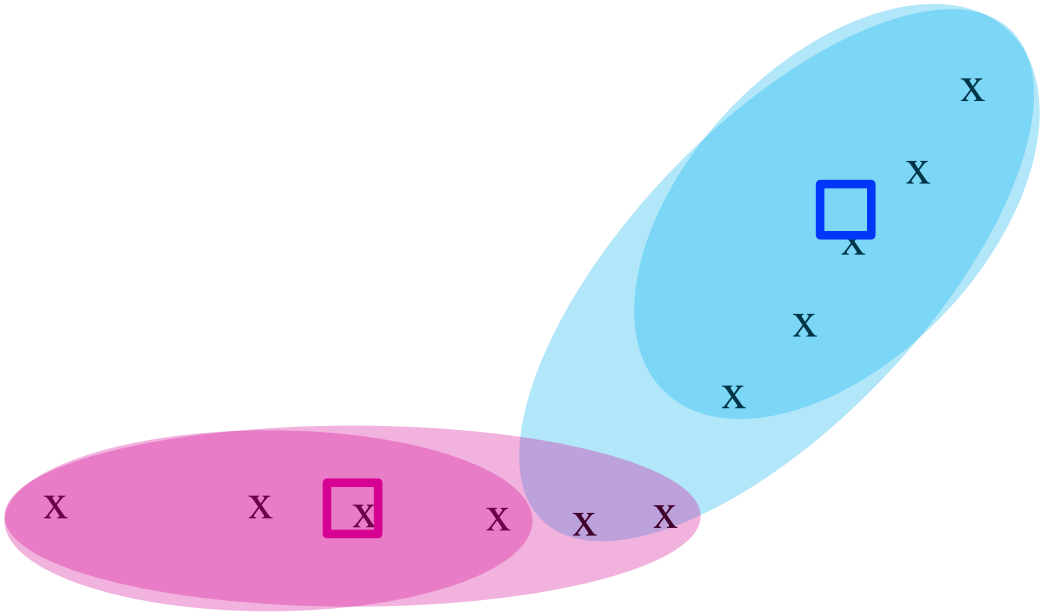
Example: Assigning Clusters



x … data point
□ … centroid

Clusters after round 2

Example: Assigning Clusters

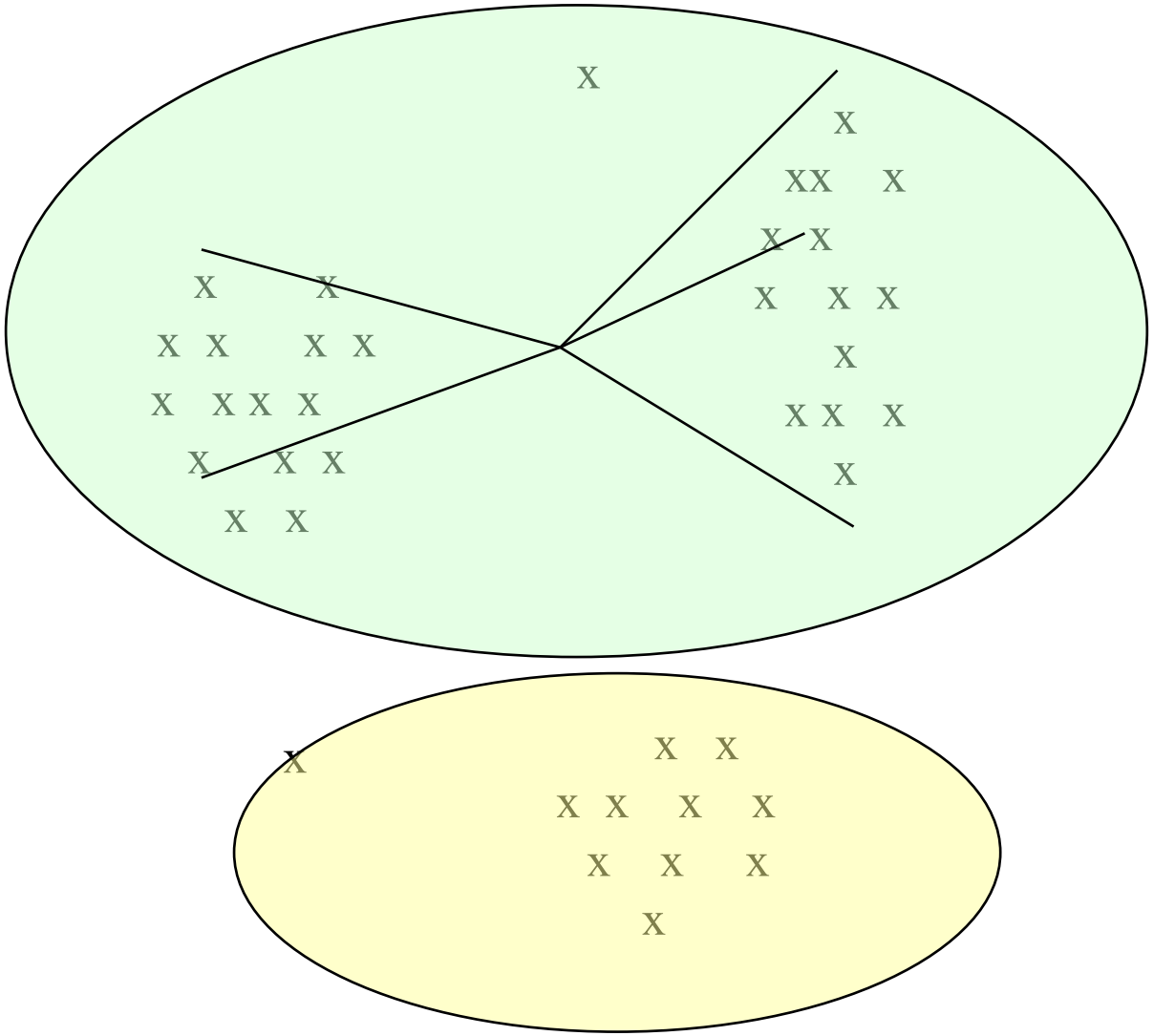


x … data point
□ … centroid

Clusters at the end

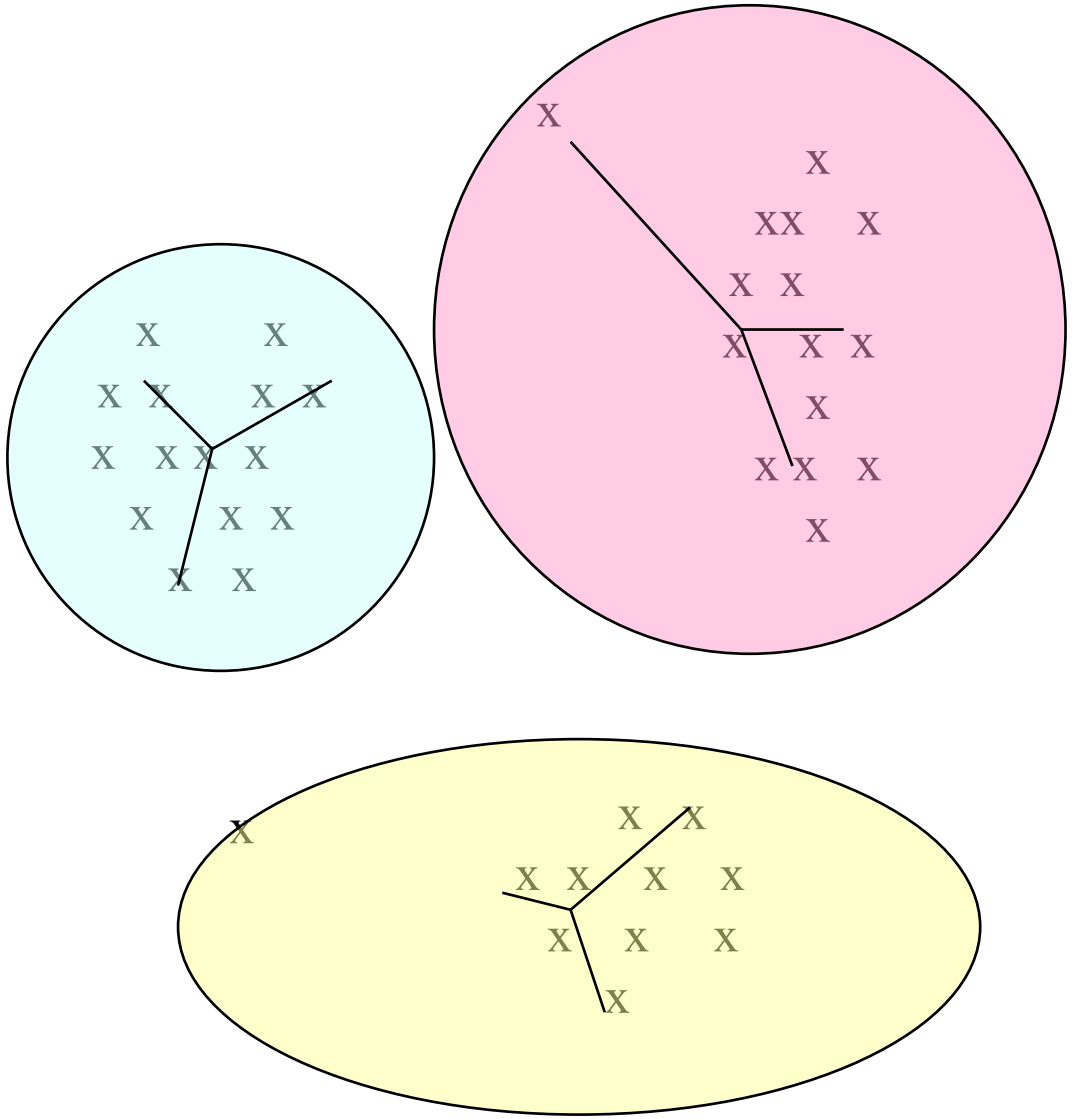
Example: Picking k

Too few;
many long
distances
to centroid.



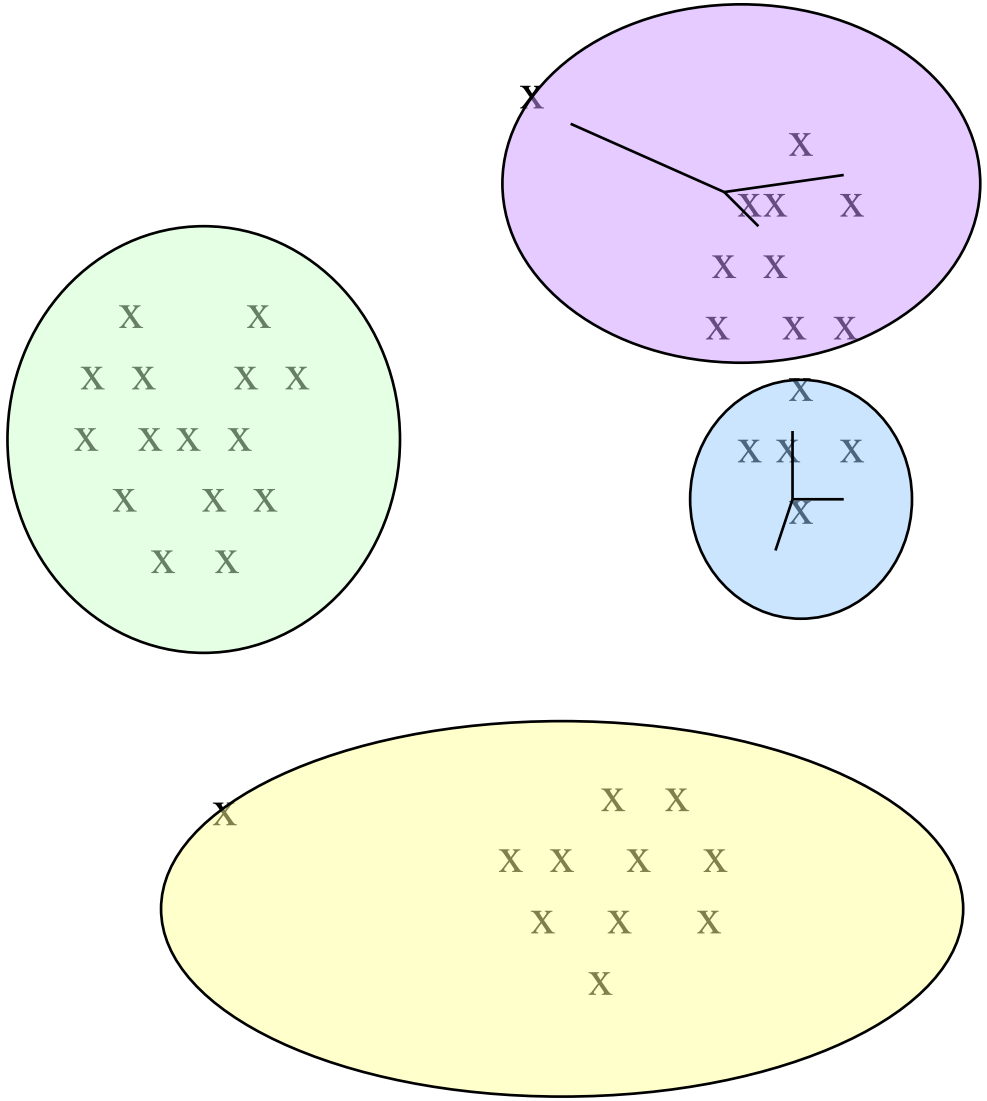
Example: Picking k

Just right;
distances
rather short.



Example: Picking k

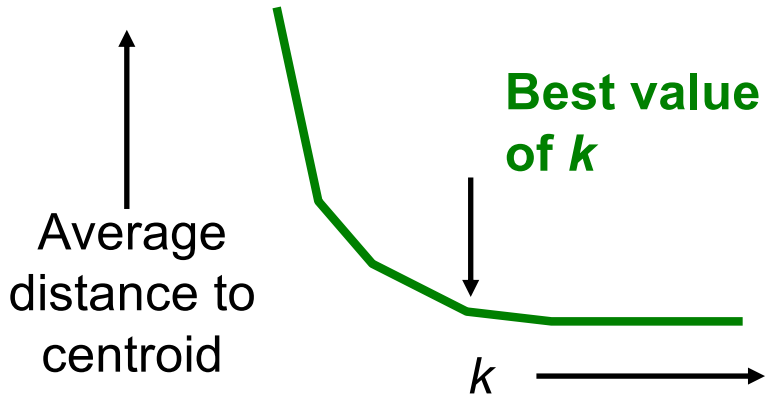
Too many;
little improvement
in average
distance.



Getting the *k* right

How to select *k*?

- Try different **k**, looking at the change in the average distance to centroid as **k** increases
- Average falls rapidly until right **k**, then changes little



k-means on MapReduce

- Initial Centroid
  - ⟨ClusterID, RandomCentroid⟩ or ⟨RandomCentroid⟩
- Map: Classify
  - setup(): Read cluster centers from a file
  - ⟨Point⟩ → ⟨NearestClusterID, Point⟩
- Reduce: Recenter
  - ⟨ClusterID, ⟨$Point_1$, $Point_2$, …, $Point_N$⟩ ⟩
  - Calculate a new center for the cluster
  - ⟨ClusterID, CentroidPoint⟩ or ⟨Centroid⟩

Job Chaining

```
var inputDir
var outputDir
var centroidDir

for i in no-of-iterations (
        Configure job here with all params
        Set job input directory = inputDir
        Set job output directory = outputDir + i
        Run job
        centroidDir = outputDir + i
)
```