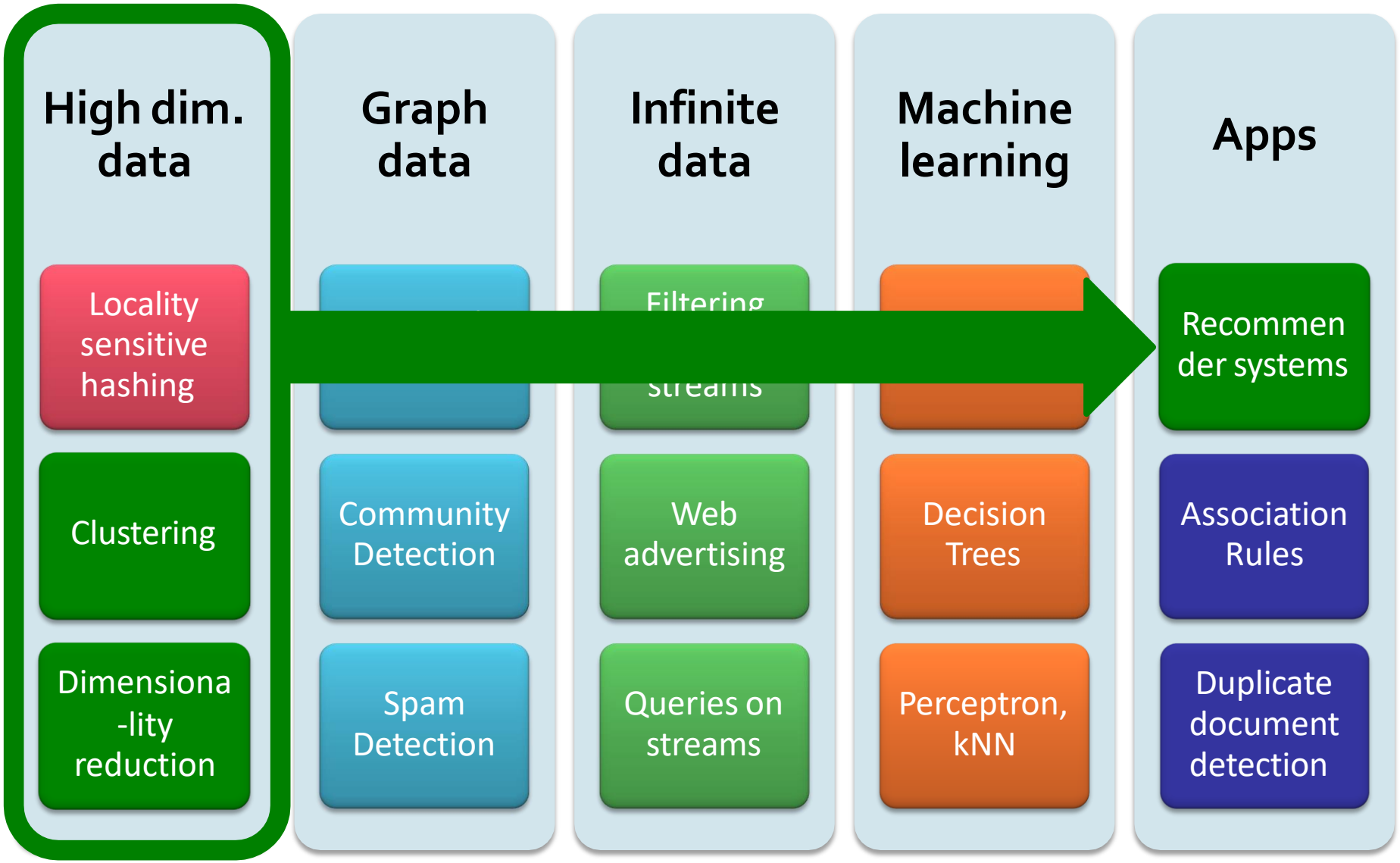
Clustering

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Stanford University
http://cs246.stanford.edu

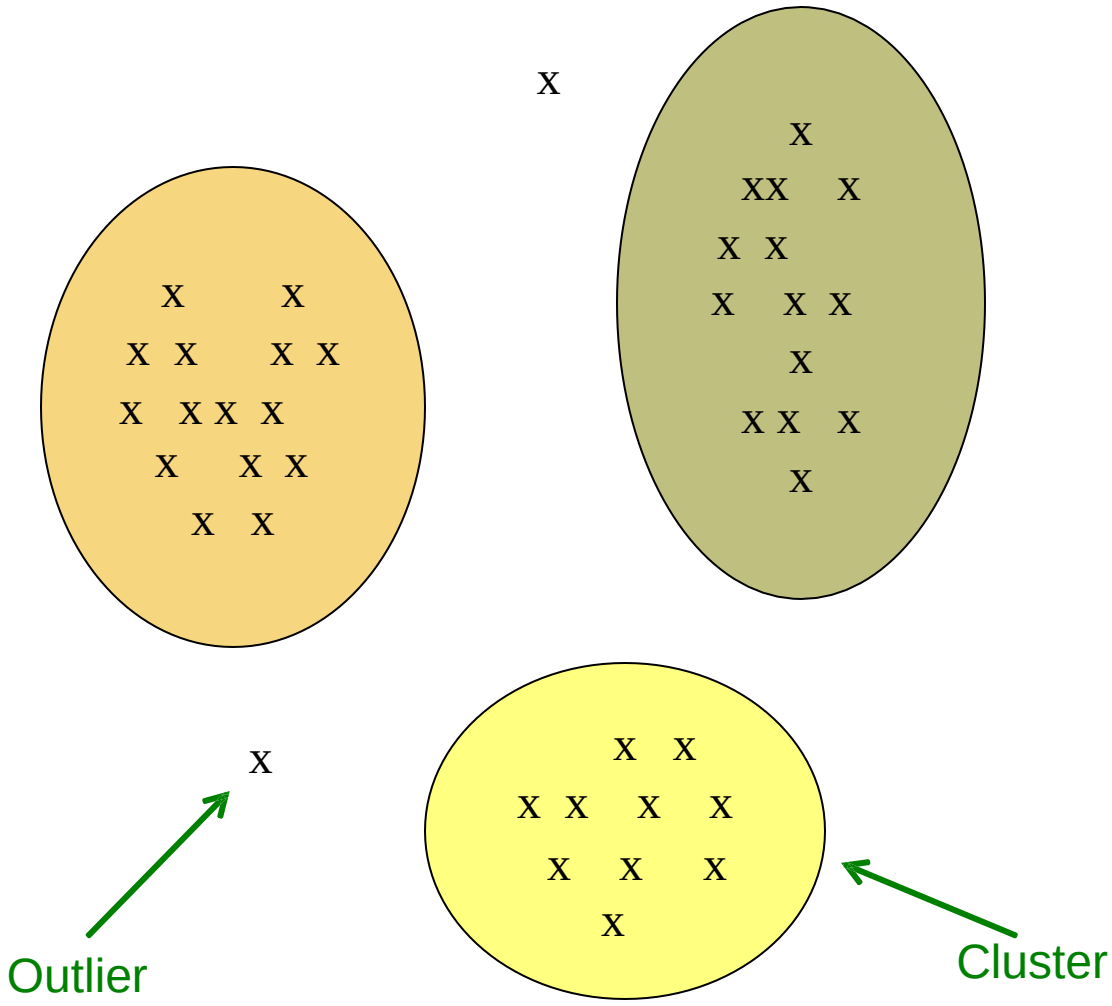
High Dimensional Data



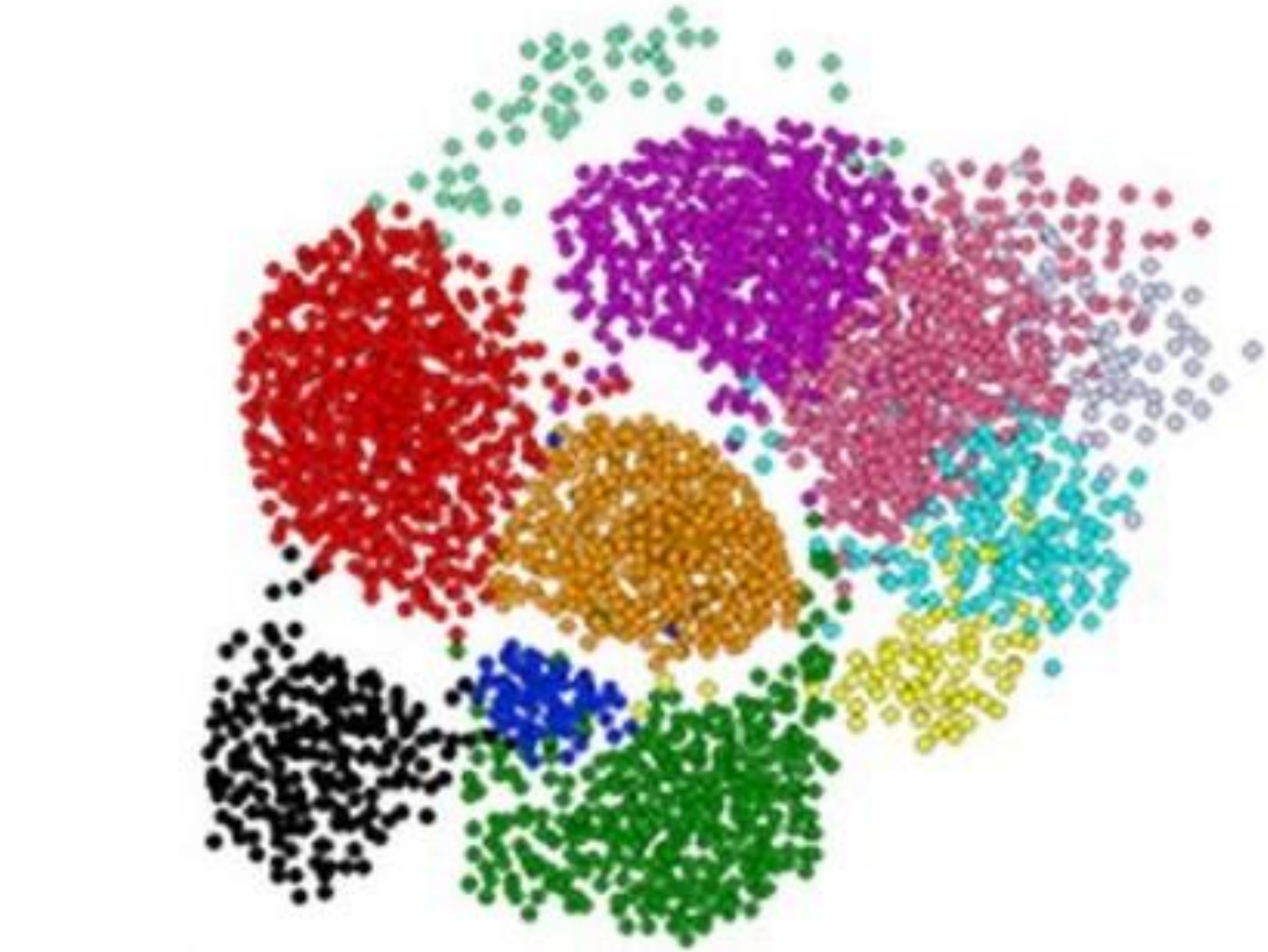
The Problem of Clustering

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, so that
  - Members of the same 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

- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:**
  Almost all pairs of points are very far from each other --> **The Curse of Dimensionality!**

Clustering Sky Objects: SkyCat

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

Clustering Problem: Music CDs

- **Intuitively: Music can be divided into categories, and customers prefer a few genres**
  - 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,…, x_d), 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 $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} word (in some order) appears in the document

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

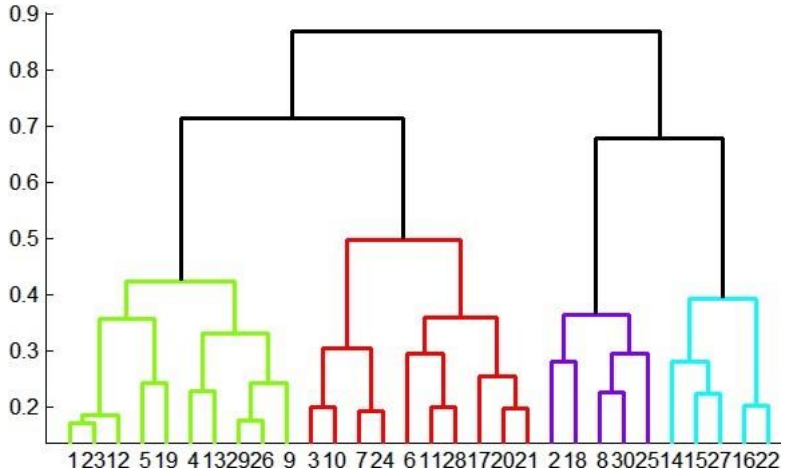
Cosine, Jaccard, and Euclidean

- **We have a choice when we think of documents as sets of words or shingles:**
  - **as vectors:** Measure similarity by the **cosine distance**
  - **as sets:** Measure similarity by the **Jaccard distance**
  - **as points:** Measure similarity by **Euclidean distance**

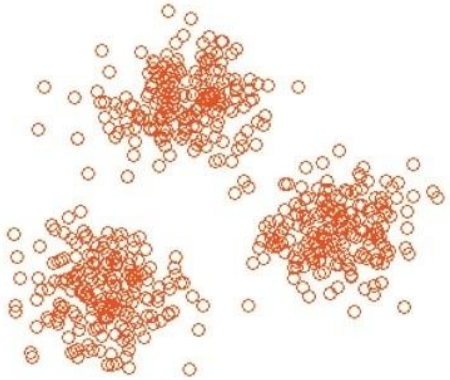
Overview: Clustering Strategies (1)

Two group of methods:

- **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 the "nearest" cluster



Overview: Clustering Strategies (2)

- Is the space Euclidean or non-Euclidean?
- In Euclidean:
  - Points are vectors of real numbers, i.e. coordinates
  - It is possible to summarize a collection of points as their average. We call it *centroid*.
  - Distance measure: L2 norm, L1 norm
- In non-Euclidean:
  - There is no notion of location, and centroid
  - We summarize a collection of points differently
  - Distance measures: Jaccard, Hamming, cosine

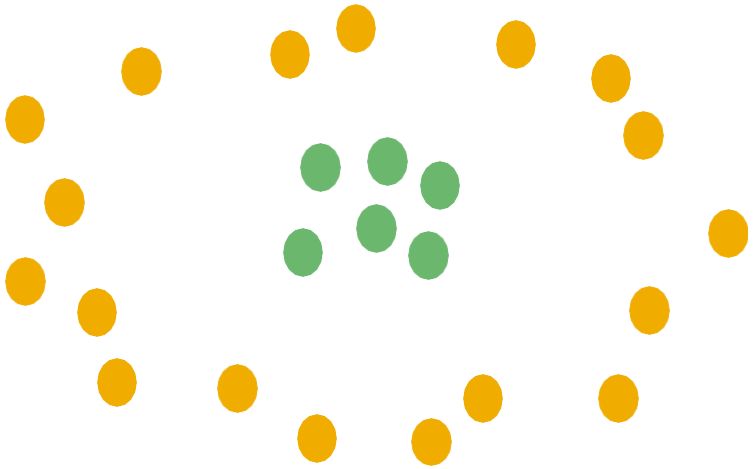
Overview: Clustering Strategies (3)

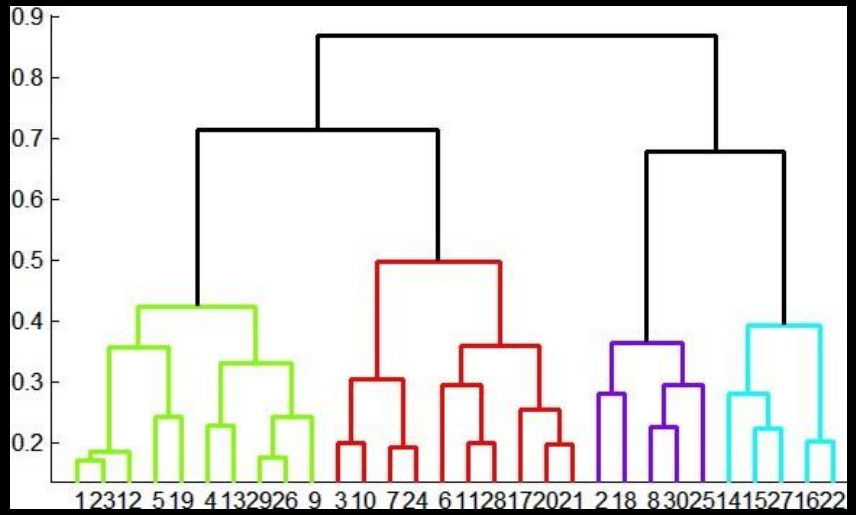
- Does the data fit in memory or does it reside on disk?
  - In-memory clustering is more straightforward
    - Example: K-means
  - Large-data clustering requires loading one batch of data at a time, cluster them in memory and keep summaries of clusters
    - Example: BFR, CURE

Hierarchical vs point-assignment

- **Point assignment good when clusters are nice, convex shapes:**

- **Hierarchical can win when shapes are weird:**
  - Note both clusters have essentially the same centroid.

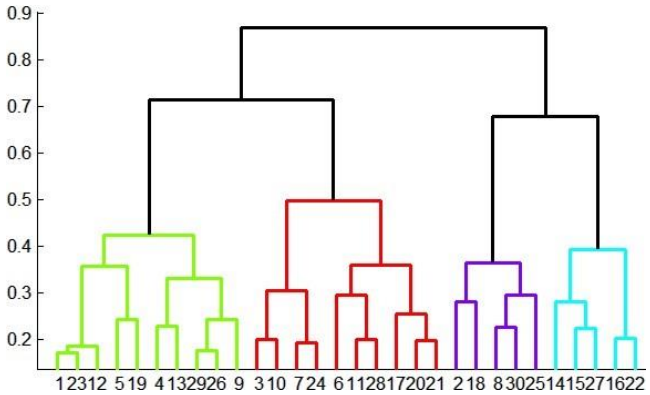




Hierarchical Clustering

Hierarchical Clustering

- **Key operation:**
  Repeatedly merge two "*nearest*" clusters



- **Three important questions:**
  - **1)** How to represent a cluster?
  - **2)** How to determine the nearness of clusters?
  - **3)** When to stop merging clusters?

Hierarchical Clustering: Euclidean

In Euclidean case:

- **(1) How to represent a cluster of many points?**
  - As we merge clusters, we represent the "location" of each cluster by its

    centroid = average of its (data)points

- **(2) How to determine the nearness of clusters?**
  - Measure cluster distances by distances of centroids
  - Merge two clusters with the shortest distance

Heirarchical Clustering: Types

Two types:

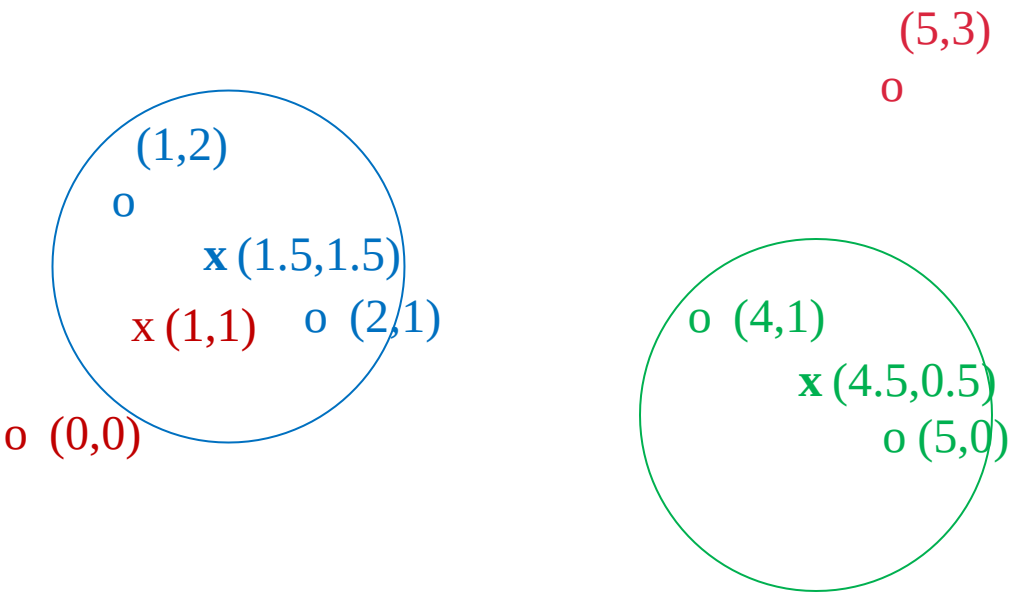
1. **Agglomerative (Bottom-up)**
   - Each object forms its own cluster
   - Iteratively merges "closest" points to form larger clusters
2. **Divisive (Top-down)**
   - Starts by placing all objects in one cluster (root)
   - Divides the root into smaller subclusters, then recursively partitions those into smaller ones

Example: Hierarchical clustering

Data:
o … data point
x … centroid

Dendrogram

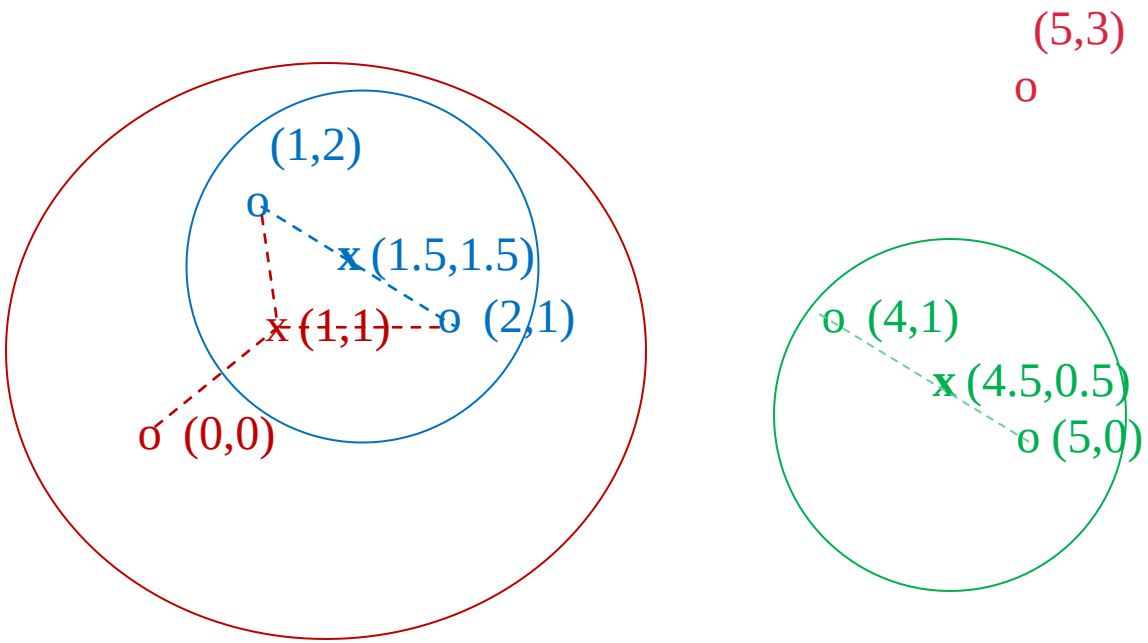
Example: Hierarchical clustering



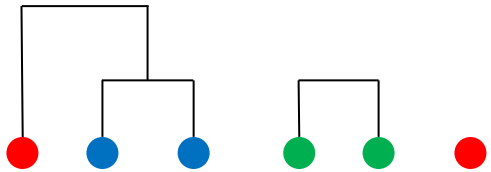
Data:
o … data point
x … centroid

Dendrogram

Example: Hierarchical clustering

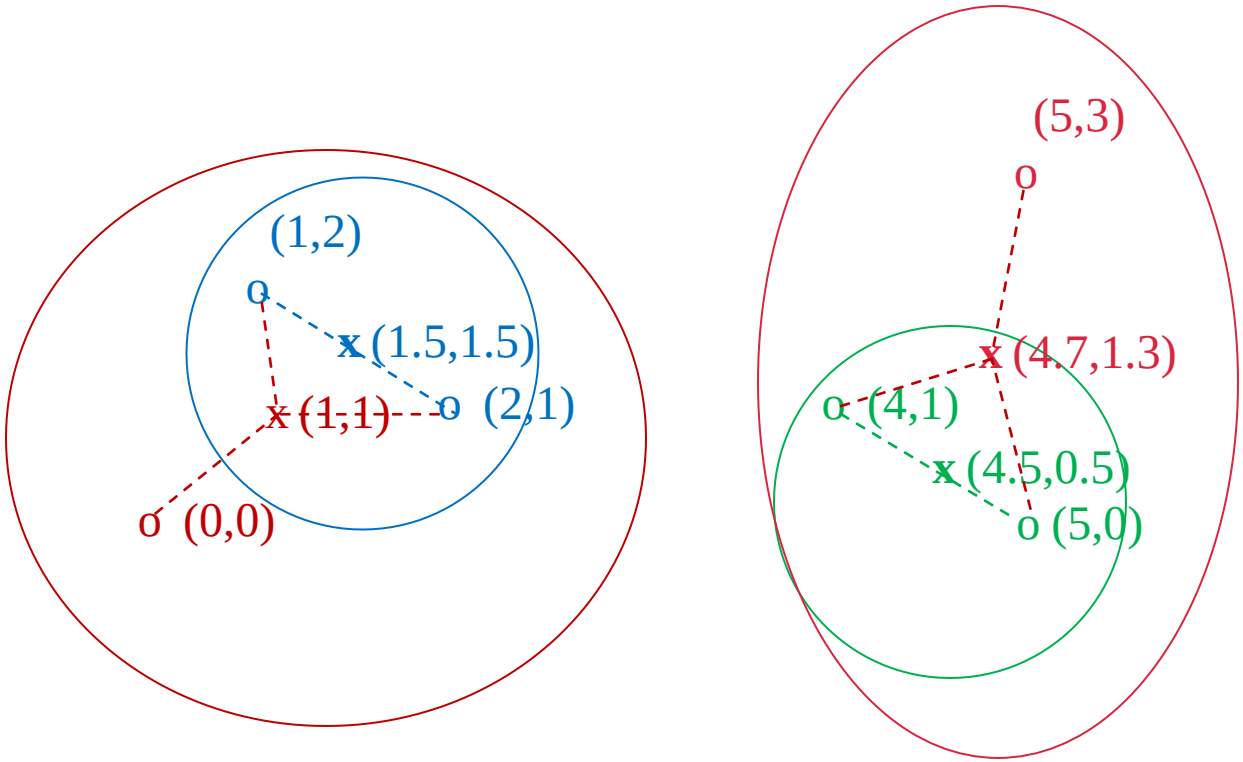


Data:
o … data point
x … centroid

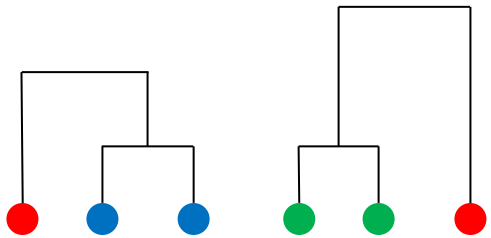


Dendrogram

Example: Hierarchical clustering

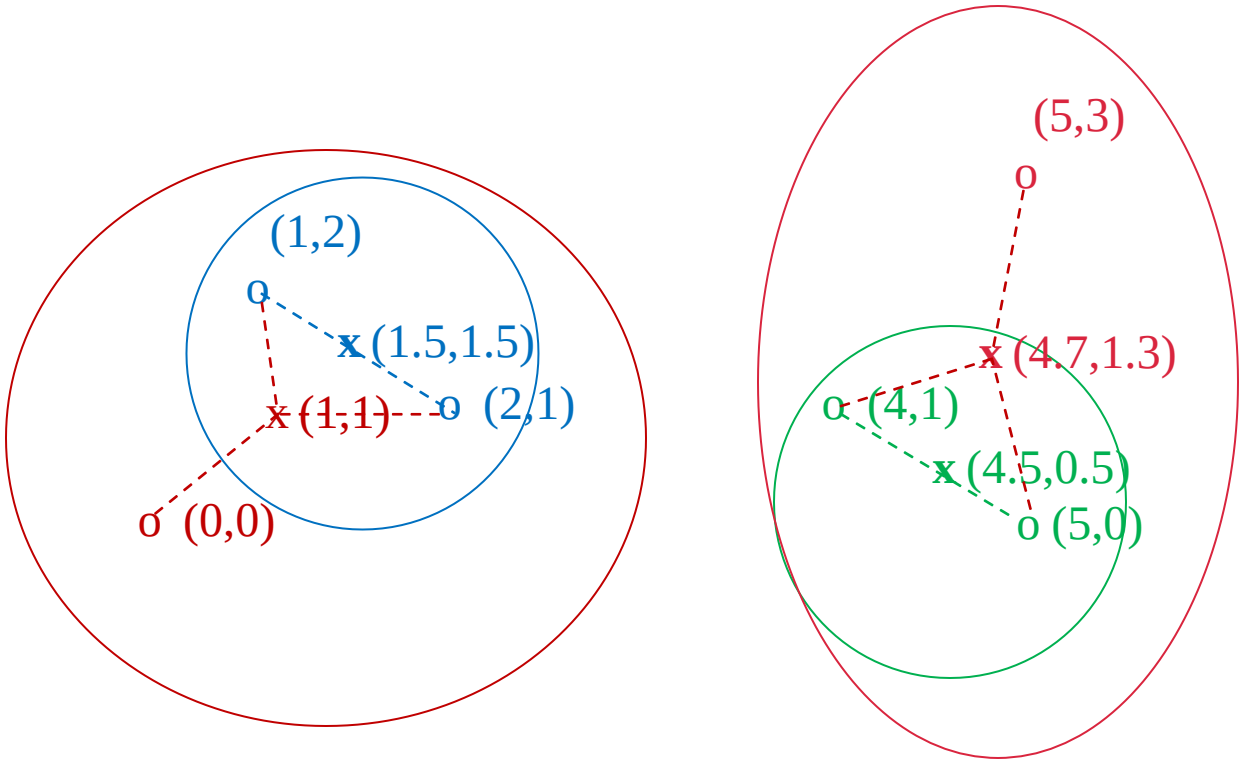


Data:
o … data point
x … centroid

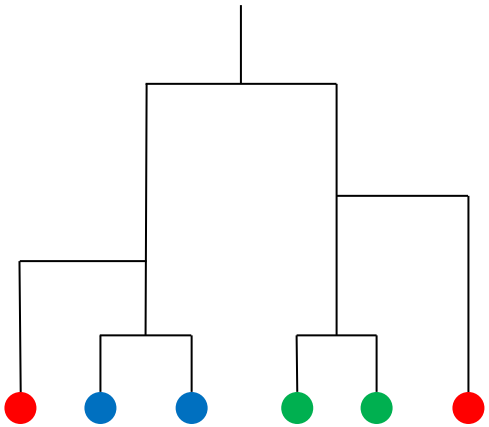


Dendrogram

Example: Hierarchical clustering



Data:
o … data point
x … centroid



Dendrogram

Heirarchical Clustering: Distance Measures

Whether using an agglomerative method or a divisive method, a core need is to measure the distance between two clusters, where each cluster is generally a set of objects.

Four widely used measures for distance between clusters are as follows, where $|\boldsymbol{p} - \boldsymbol{p}'|$ is the distance between two objects or points, $\boldsymbol{p}$ and $\boldsymbol{p}'$; $\boldsymbol{m_i}$ is the mean for cluster, C_i; and n_i is the number of objects in C_i. They are also known as *linkage measures*.

Minimum distance:
$$dist_{min}(C_i, C_j) = \min_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} \{|\boldsymbol{p} - \boldsymbol{p}'|\} \quad (10.3)$$

Maximum distance:
$$dist_{max}(C_i, C_j) = \max_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} \{|\boldsymbol{p} - \boldsymbol{p}'|\} \quad (10.4)$$

Mean distance:
$$dist_{mean}(C_i, C_j) = |\boldsymbol{m_i} - \boldsymbol{m_j}| \quad (10.5)$$

Average distance:
$$dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} |\boldsymbol{p} - \boldsymbol{p}'| \quad (10.6)$$

Hierarchical Clustering: Non-Euclidean

In non-Euclidean case:

- The only "locations" we can talk about are the points themselves. There are three main approaches:

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

1. pick a ***clustroid*** = point "***closest***" to other points

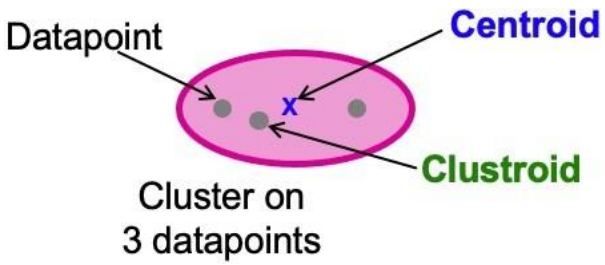
2,3. As the collection of points it is.

(2) How to determine the nearness of clusters?

1. Treat clustroid as if it were centroid
2. Various distance measures between points of two clusters
3. Various cohesion measures of the union of two clusters

Centroid vs Clusteroid

- **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.



Hierarchical Clustering: Non-Euclidean

Approach 1:

(1) How to represent a cluster:

- pick a ***clustroid*** = (data)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 $\arg\min_{c} \sum_{x \in C} d(x, c)^2$

Hierarchical Clustering: Non-Euclidean

Approach 1:

(2) How to determine the nearness of clusters?

- Treat clustroid as if it were centroid
- Define *inter-cluster distance*:
  - Minimum of the distances between any two points, one from each cluster
  - Average distance of all pairs of points, one from each cluster

Hierarchical Clustering: Non-Euclidean

Approach 2:

(1)How to represent a cluster? Pick a *clustroid*, which is closest to other points in the cluster

(2) How to determine the nearness of clusters?

Define *inter-cluster distance*:

- Minimum of the distances between any two points, one from each cluster
- Average distance of all pairs of points, one from each cluster

Hierarchical Clustering: Non-Euclidean

Approach 3:

~~**(1) How to represent a cluster?** As the collection of points it is~~

(2)How to determine the nearness of clusters?

Define a notion of *cohesion,* and merge clusters whose *union* is most cohesive

Possible notions of cohesion (the smaller, the more cohesive):

- **diameter** of the merged cluster = maximum distance between points in the cluster
- **average distance** between points in the cluster
- **Density** of the merged cluster = divide by the number of points in the cluster by some power of diameter or radius (higher density is better)

When to Stop?

(3)When do we stop merging clusters?

- When some number k of clusters are found (assumes we know the number of clusters)
- When stopping criterion is met
  - Stop if diameter or radius exceeds threshold
  - Stop if density is below some threshold
  - Stop if merging clusters yields a bad cluster
    - E.g., diameter suddenly jumps (example 7.5 in book)
- Keep merging until there is only 1 cluster left

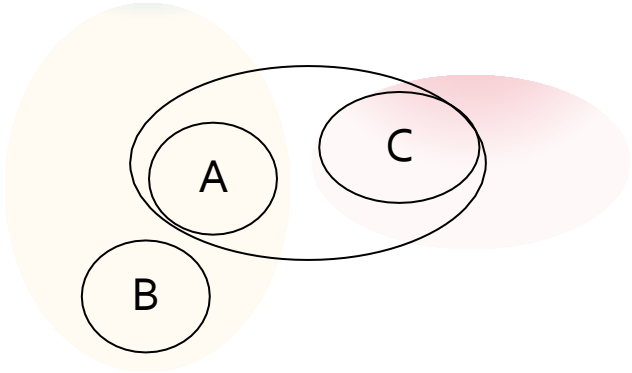
Which design choice is the best?

- 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.

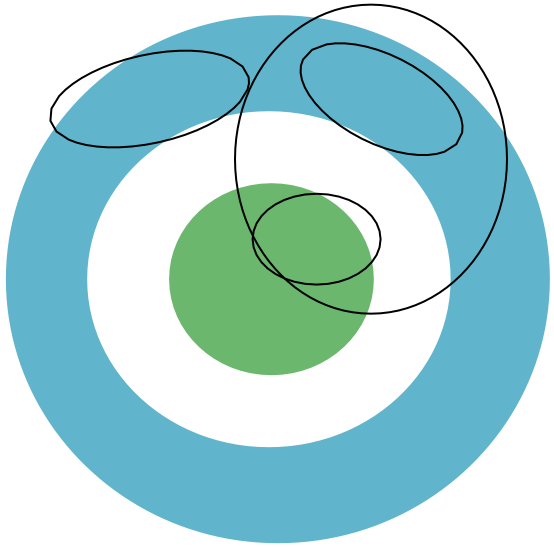


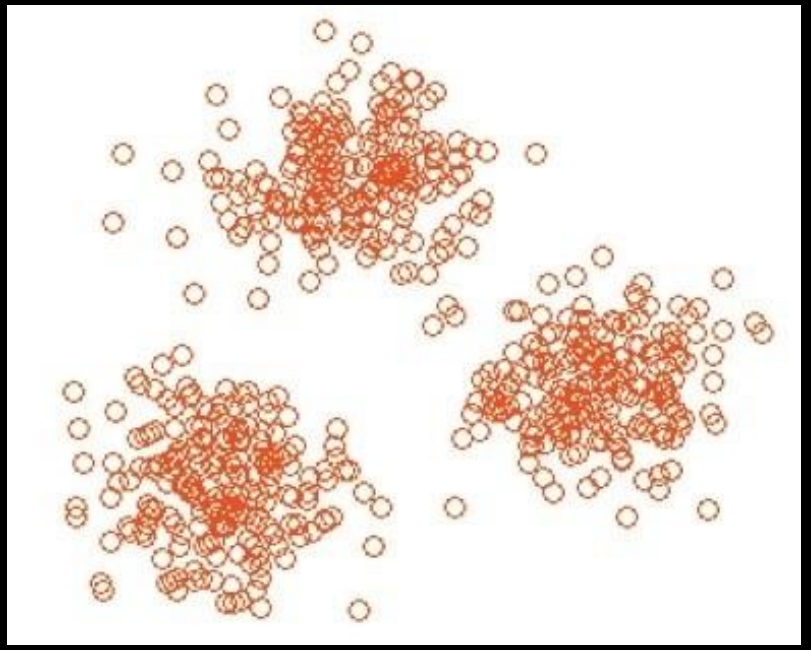
A and B have closer centroids than A and C, but closest points are from A and C.

Data density

Case 2: Concentric Clusters

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





k-means Clustering

k–means Algorithm(s)

- It is a problem formulation, not an algorithm.
- **Problem**: Given Euclidean space/distance and ***k*** **=** number of clusters, find cluster centers that minimizes sum of squared distances from each point to its cluster center
- Finding an exact solution is NP-hard.
- The approximate solution is Lloyd's algorithm or the k-means algorithm.

k–means Algorithm(s)

- Initialize clusters by picking k centers

Until convergence:

- **1)** For each point, assign it to the cluster whose current centroid is the closest
- **2)** After all points are assigned, update the centroids of the ***k*** clusters as average of datapoints within each cluster

Convergence means Points don't move between clusters and centroids stabilize

k–means Algorithm(s)

Quality of clusters:

The quality of cluster C_i can be measured by the **within-cluster variation**, which is the sum of squared error between all objects in C_i and the centroid c_i , defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist(\boldsymbol{p}, \boldsymbol{c_i})^2$$

E = sum of the squared error for all objects in the data set

p = point in space representing a given object;

ci = centroid of cluster Ci

i.e., for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed.

k–means Algorithm(s)

Properties of k-means:

- not guaranteed to converge to the global optimum and often terminates at a local optimum
- time complexity: *O(nkt)*
  - *n = number of objects*
  - *k = number of clusters*
  - *t = number of iterations until convergence*
- Usually, k << n and t << n

Shortcoming of k-means

Convergence of k-means heavily depends on the initial pick of centroids. It can perform arbitrarily bad.

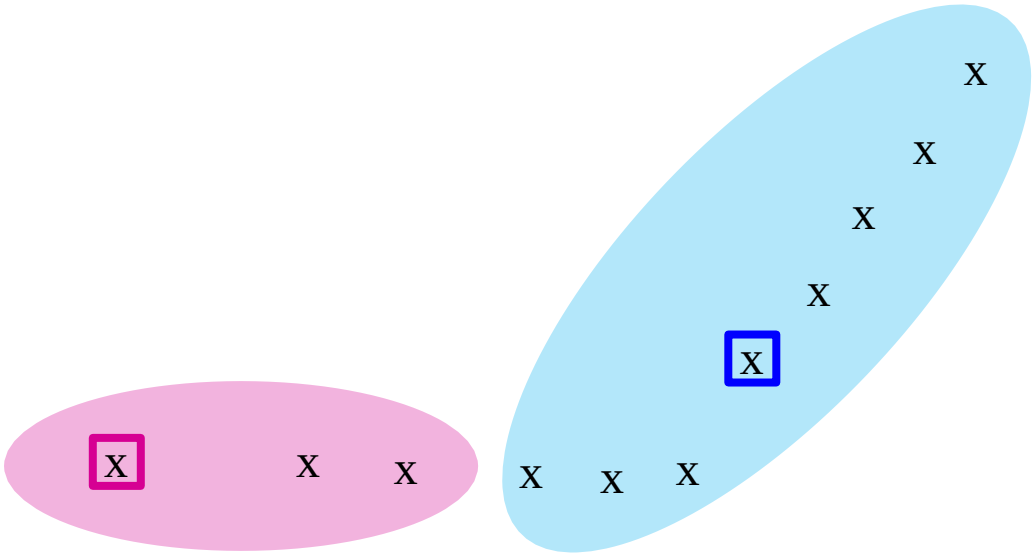
- Very sensitive to outliers
  - Alternative: k-medoids

Shortcoming of k-means

Convergence of k-means heavily depends on the initial pick of centroids. It can perform arbitrarily bad.

- Different strategies for picking k centers:
  - Pick k datapoints at random
  - K-means ++

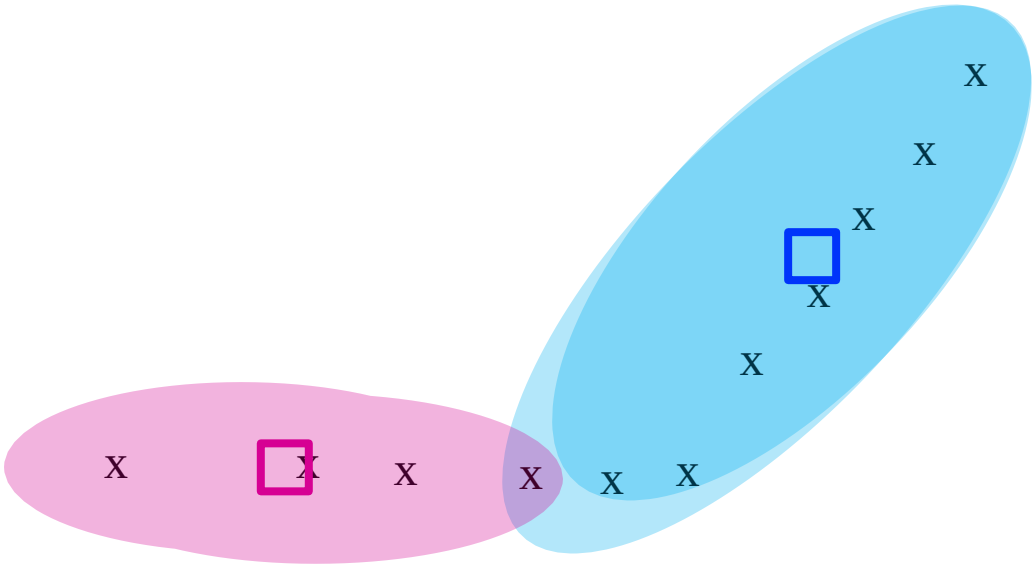
Example: Assigning Clusters



x … data point
□ … centroid

Clusters after round 1

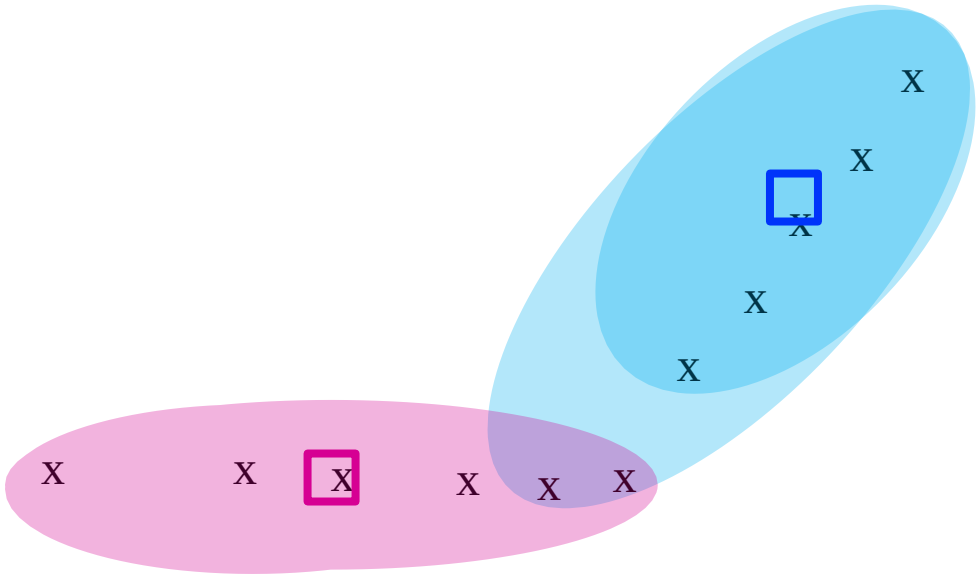
Example: Assigning Clusters



x … data point
□ … centroid

Clusters after round 2

Example: Assigning Clusters



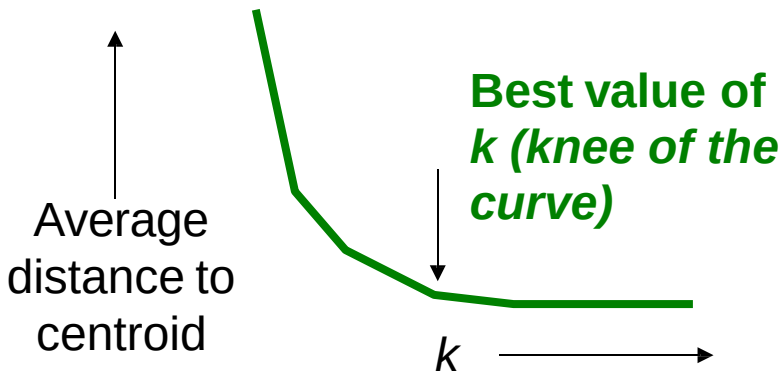
x … data point
□ … centroid

Clusters at the end

Getting the *k* right

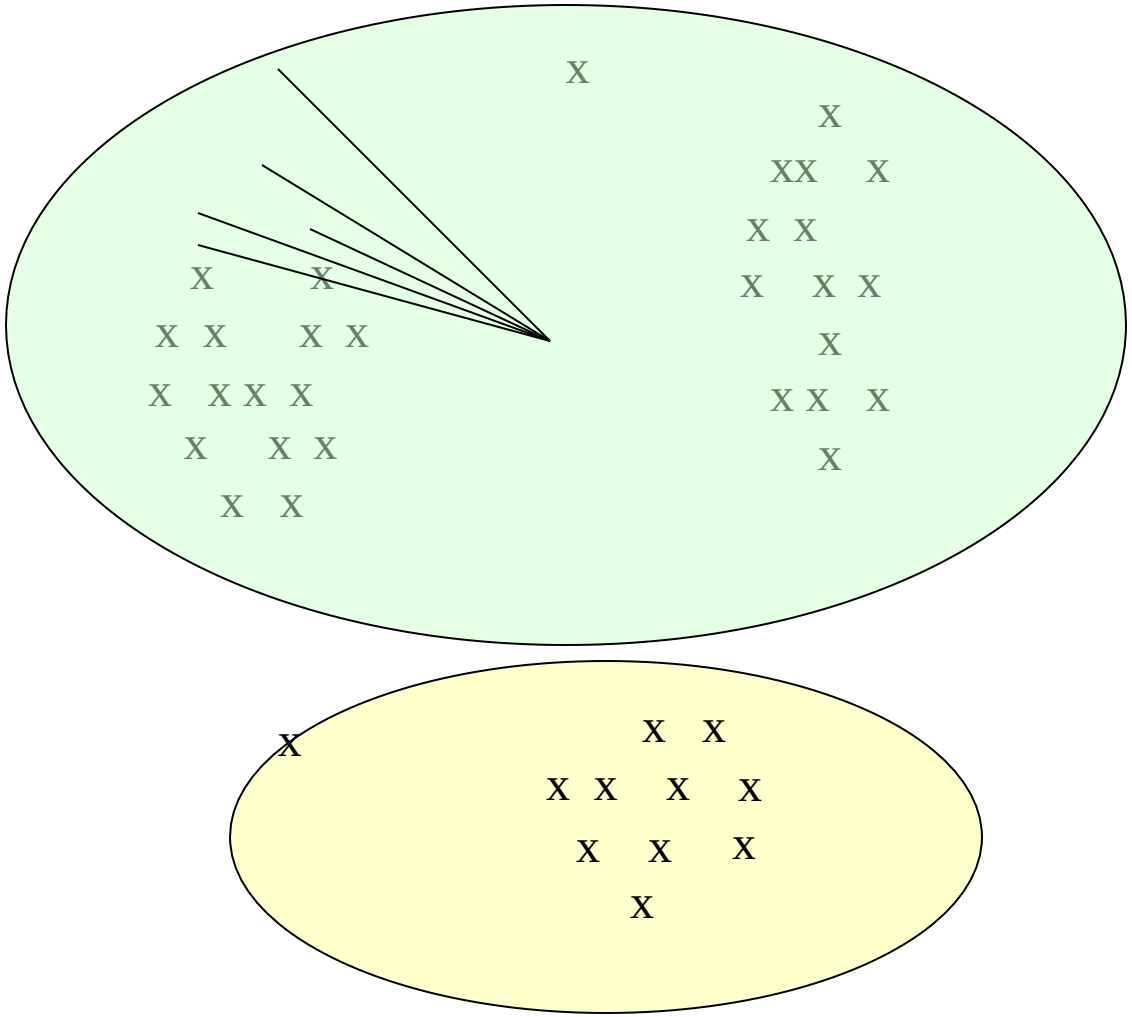
How to select *k*?

- 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



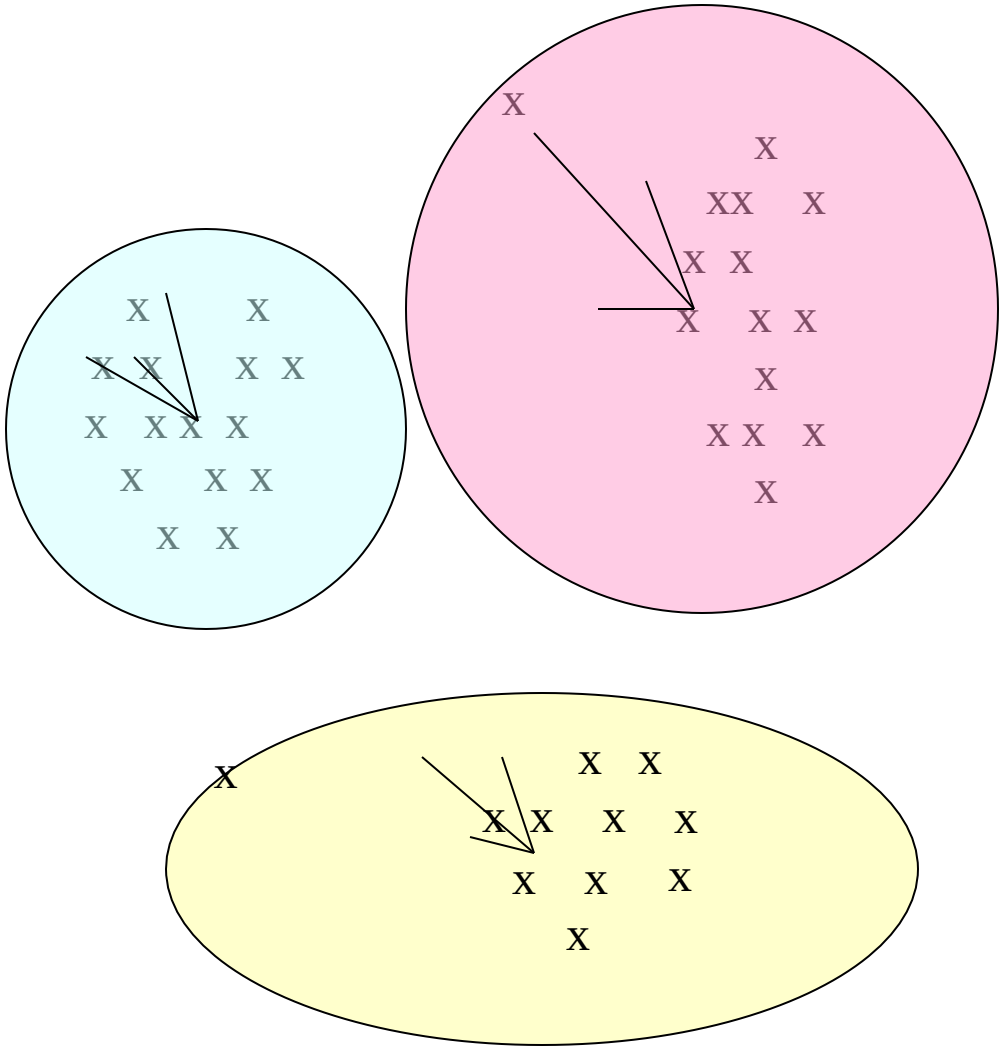
Example: Picking *k*

Too few;
many long
distances
to centroid



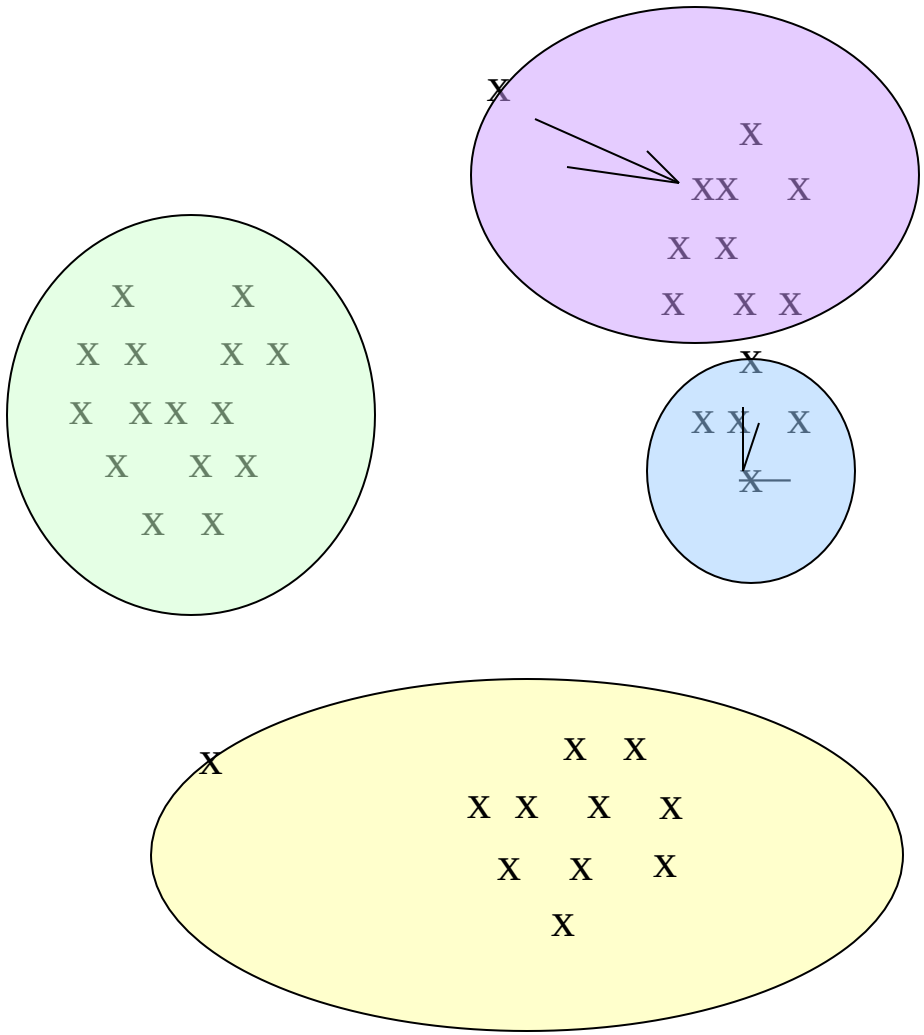
Example: Picking *k*

Just right;
distances
rather short



Example: Picking *k*

Too many;
little improvement
in average
distance



Picking the initial *k* points

- **Approach 1: Sampling**
  - Cluster a sample of the data using hierarchical clustering, to obtain k clusters.
  - Pick a point from each cluster (e.g., point closest to centroid)
  - Sample fits in main memory
- **Approach 2: Pick "dispersed" set of points (k-Means++)**
  - Pick first point at random
  - Pick the next point to be the one whose minimum distance from the selected points is as large as possible
  - Repeat until we have *k* points

k-Means++

- **Basic idea:** Pick a small sample of points S, cluster them by any algorithm, and use the centroids as a seed
- In **k-means++,** sample size $|S| = k$ times a factor that is logarithmic in the total number of points

- **How to pick sample points:** Visit points in random order, but the probability of adding a point p to the sample is proportional to $D(p)^2$.
  - $D(p)$ = distance between p and the nearest already picked point.

Complexity of k-means clustering

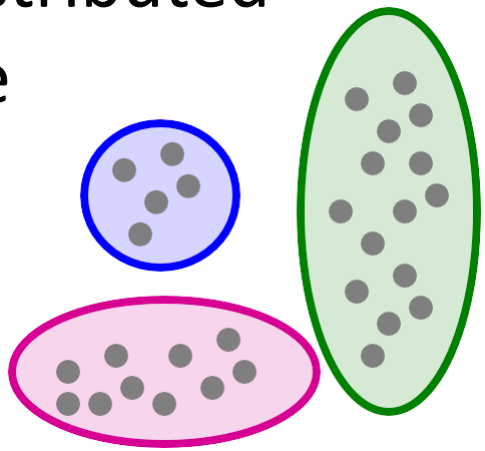
- In each round, we have to examine each input point exactly once to find closest centroid
- Each round is $O(Nk)$ for N points, k clusters
- If there are t rounds, then the complexity is $O(Nkt)$
- Problem: Number of rounds to convergence can be very large i.e., t can be very large
- Can we cluster in a single pass over the data?

The BFR Algorithm

Extension of *k*-means to large data

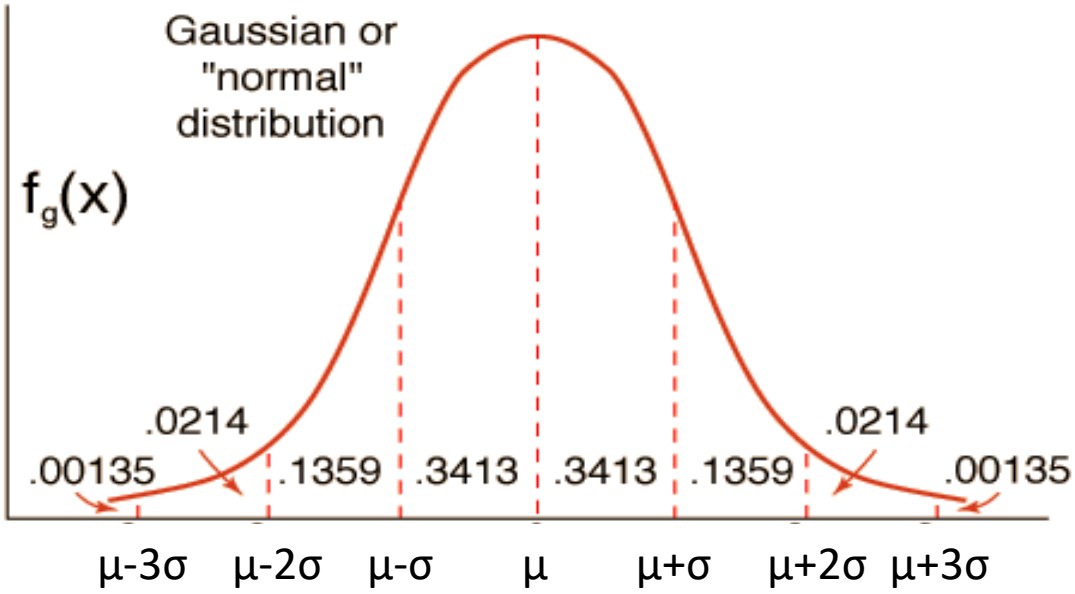
BFR Algorithm

- **BFR** [Bradley-Fayyad-Reina] is a variant of *k*-means designed to handle **very large** (disk-resident) data sets
- **Assumes** that clusters are normally distributed around a centroid in a Euclidean space
  - Standard deviations in different dimensions may vary
    - Clusters are axis-aligned ellipses



- Goal is to find cluster centroids; point assignment can be done in a second pass through the data.

Normal Distribution Overview



- Can quantify the likelihood of finding a point in the cluster at a given distance from the centroid along each dimension. (Remember the 68-94-99 rule?)

- Standard deviations in different dimensions may vary.

BFR Overview

- **Efficient way to summarize clusters:** Want memory required O(clusters) and not O(data)
- **IDEA: Rather than keeping points, BFR keeps summary statistics of groups of points**
  - **3 sets: Discard set, Compressed set, Retained set**
- **Overview of the algorithm:**
  - **1.** Initialize *K* clusters/centroids
  - **2.** Load in a bag of points from disk
  - **3.** Assign new points to one of the *K* original clusters, if they are ***"sufficiently close"*** to one of the clusters
  - **4.** Cluster the remaining points, and create new clusters
  - **5.** Try to merge new clusters from step 4 with any of the existing clusters
  - **6.** Repeat steps 2-5 until all points are examined

BFR Algorithm

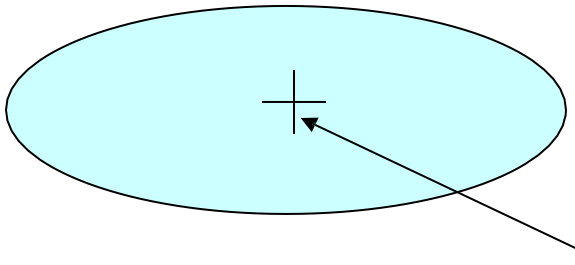
- **Points are read from disk one main-memory-full at a time**
- **Most points from previous memory loads are summarized by simple statistics**
- **Step 1)** From the initial load we select the initial ***k*** centroids by some sensible approach:
  - Take ***k*** random points
  - Take a small random sample and cluster optimally
  - Take a sample; pick a random point, and then ***k–1*** more points, each as far from the previously selected points as possible

Three Classes of Points

3 sets of points which we keep track of:

- **Discard set (DS):**
  - Points close enough to a centroid to be summarized
- **Compressed set (CS):**
  - Groups of points that are close together but not close to any existing centroid
  - These points are summarized, but not assigned to a cluster
- **Retained set (RS):**
  - Isolated points waiting to be assigned to a compressed set

BFR: "Galaxies" Picture

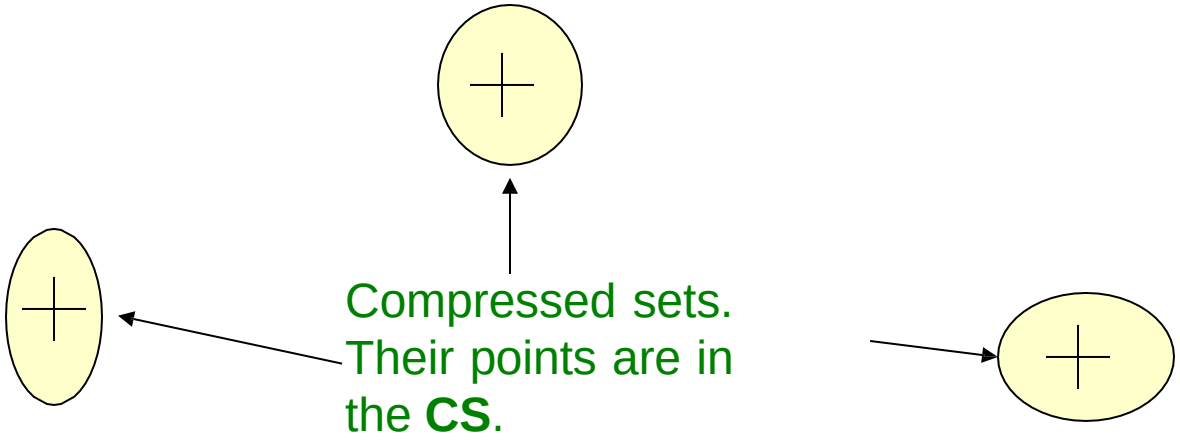


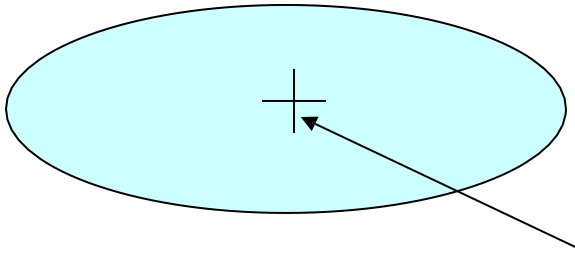
A cluster. Its points are in the **DS**.

The centroid

Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points, we store them as they are

BFR: "Galaxies" Picture



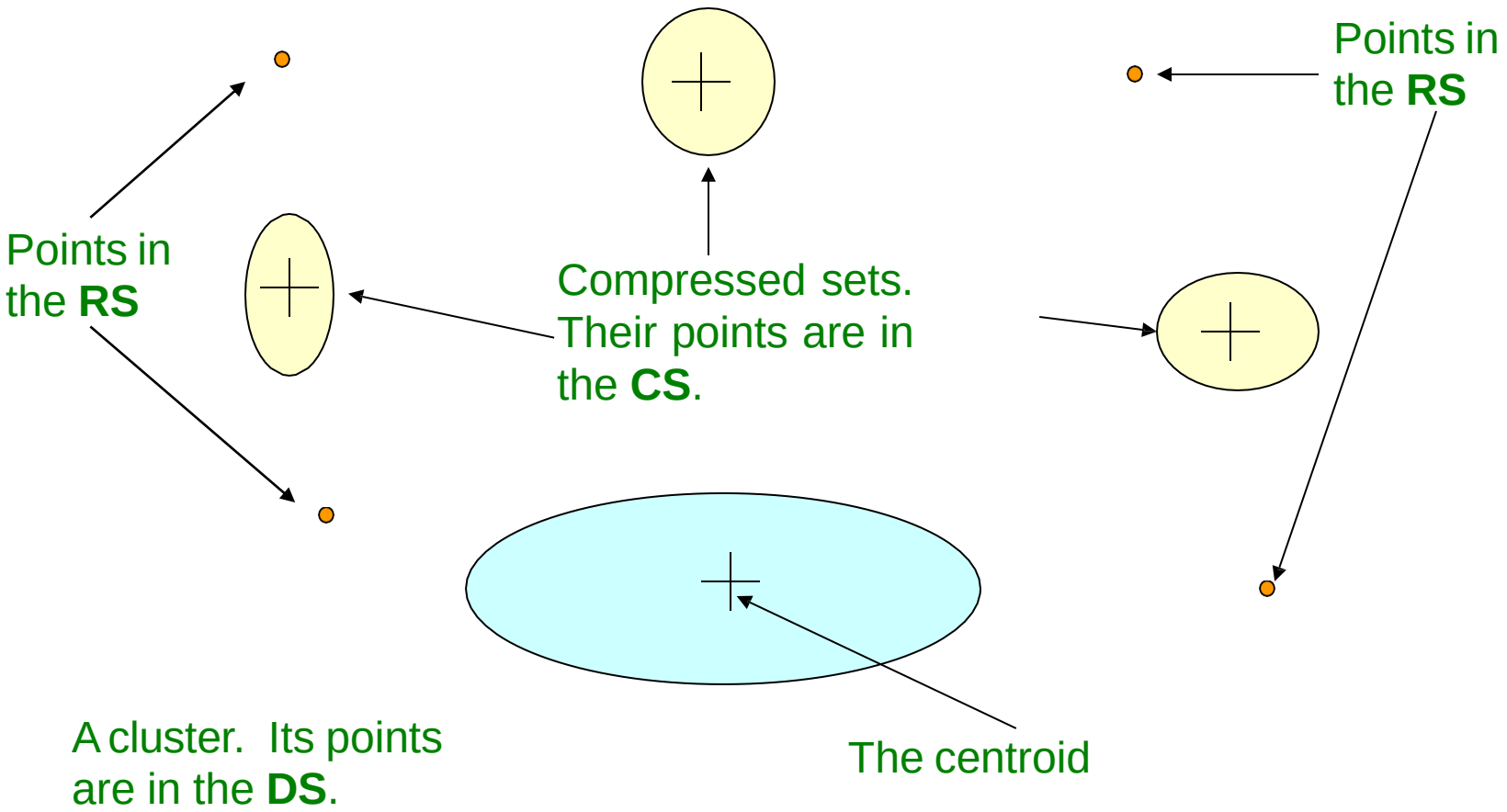


A cluster. Its points are in the **DS**.

The centroid

Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points, we store them as they are

BFR: "Galaxies" Picture

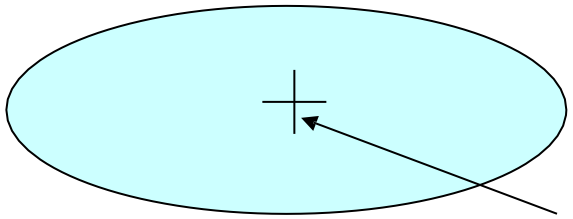


Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points, we store them as they are

Summarizing Sets of Points

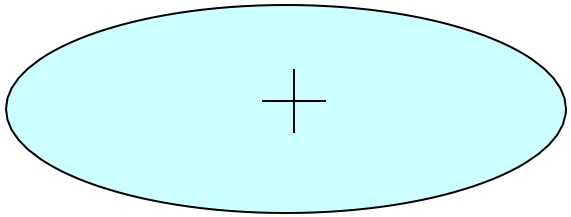
For each cluster, the discard set (DS) is summarized by:

- The number of points, ***N***
- The vector ***SUM***, whose i^{th} component is the sum of the coordinates of the points in the i^{th} dimension
- The vector ***SUMSQ***: i^{th} component = sum of squares of coordinates in i^{th} dimension



Summarizing Points: Comments

- **$2d + 1$** values represent any size cluster
  - ***d*** = number of dimensions
- Average in **each dimension** (**the centroid**) can be calculated as **SUM_i / N**
  - **SUM_i** = i^{th} component of SUM
- Variance of a cluster's discard set in dimension *i* is: **$(SUMSQ_i / N) – (SUM_i / N)^2$**
  - And standard deviation is the square root of that
- **Next step: Actual clustering**



Note: Dropping the "axis-aligned" clusters assumption would require storing full covariance matrix to summarize the cluster. So, instead of **SUMSQ** being a ***d***-dim vector, it would be a ***d x d*** matrix, which is too big!

The "Memory-Load" of Points

Steps 3-5) Processing "Memory-Load" of points:

- **Step 3)** Find those points that are "**sufficiently close**" to a cluster centroid and add those points to that cluster and the **DS**
  - These points are so close to the centroid that they can be summarized and then discarded
- **Step 4)** Use any in-memory clustering algorithm to cluster the remaining points and the old **RS**
  - Clusters go to the **CS**; outlying points to the **RS**

Discard set (DS): Close enough to a centroid to be summarized.
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

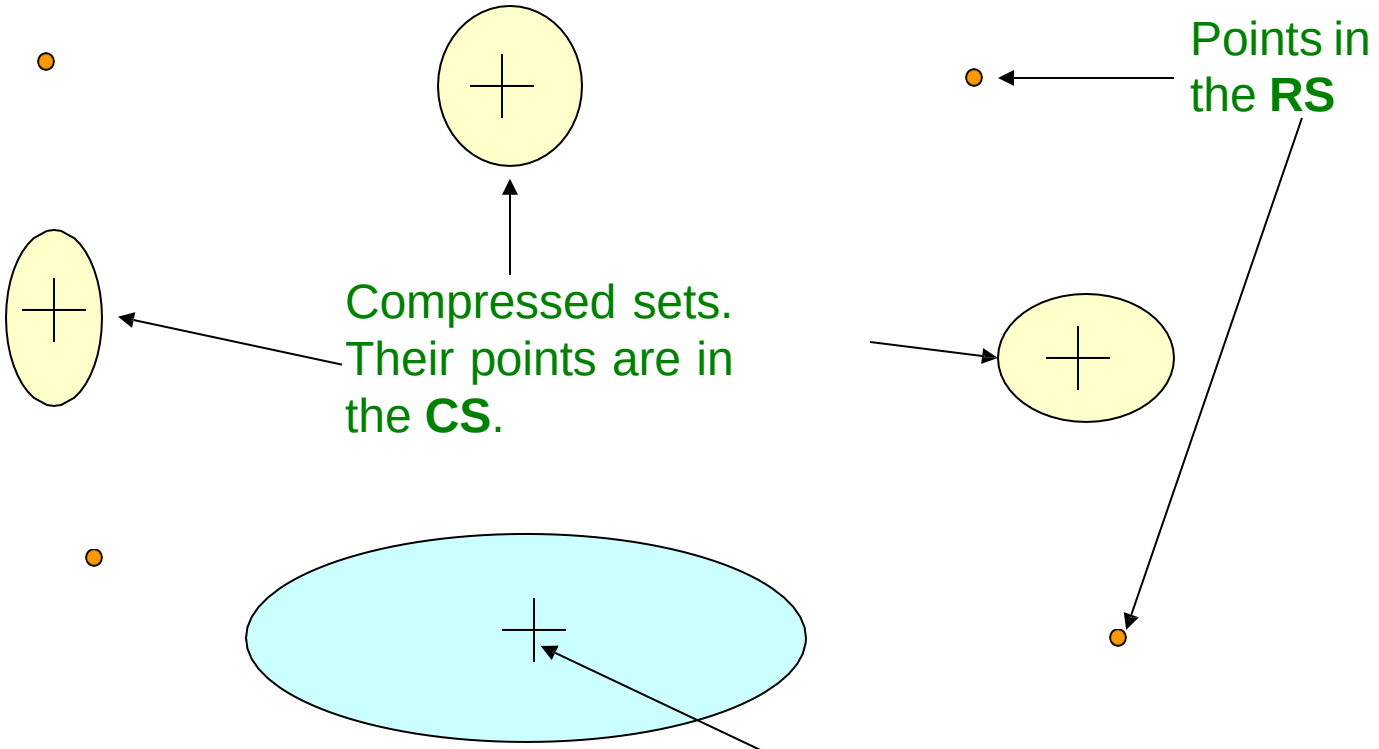
The "Memory-Load" of Points

Steps 3-5) Processing "Memory-Load" of points:

- **Step 5) DS set:** Adjust statistics of the clusters to account for the new points
  - Add ***N***s, ***SUM***s, ***SUMSQ***s
  - Consider merging compressed sets in the **DS**
- **If this is the last round**, merge all compressed sets in the **CS** and all **RS** points into their nearest cluster

Discard set (DS): Close enough to a centroid to be summarized.
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

Summary: BFR



Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

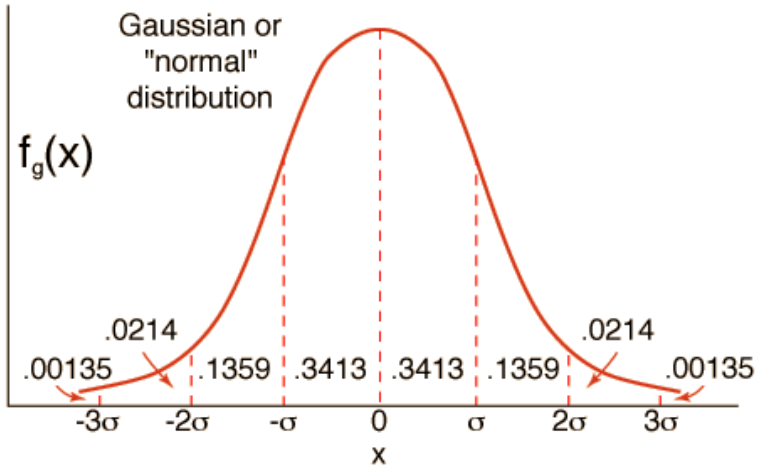
A Few Details…

- **Q1) How do we decide if a point is "close enough" to a cluster that we will add the point to that cluster?**
- **Q2) How do we decide whether two compressed sets (CS) deserve to be combined into one?**

How Close is Close Enough?

- **Q1) We need a way to decide whether to put a new point into a cluster (and discard)**

- **BFR suggests two ways:**
  - The **Mahalanobis distance** is less than a threshold
  - **High likelihood of the point belonging to currently nearest centroid**



Mahalanobis Distance

- **Normalized Euclidean distance from centroid**

- For a given point $(x_1, \ldots, x_d)$ and a given centroid $(c_1, \ldots, c_d)$
  1. Normalize in each dimension: $y_i = (x_i - c_i) / \sigma_i$
  2. Take sum of the squares of the y_i
  3. Take the square root

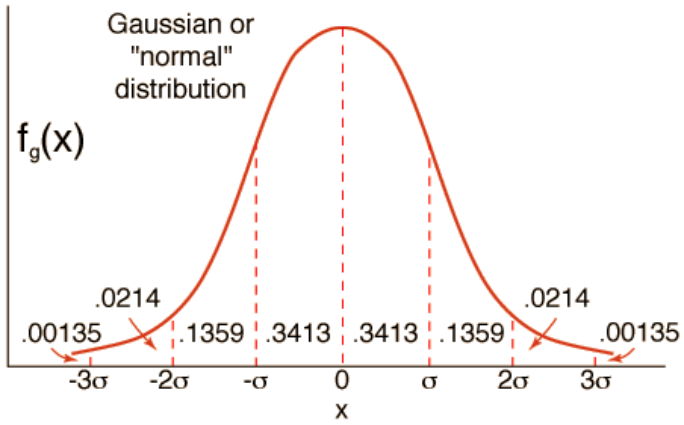
$$d(x, c) = \sqrt{\sum_{i=1}^{d} \left(\frac{x_i - c_i}{\sigma_i}\right)^2}$$

σ_i … standard deviation of points in the cluster in the i^{th} dimension

Mahalanobis Distance

- If clusters are normally distributed in d dimensions, then after transformation, one standard deviation = $\sqrt{d}$
  - i.e., 68% of the points of the cluster will have a Mahalanobis distance $< \sqrt{d}$

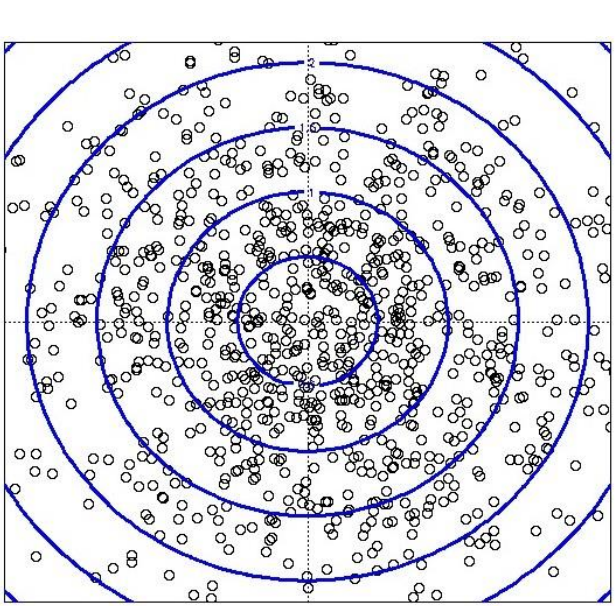
- Accept a point for a cluster if its M.D. is **<** some threshold, e.g. **2** standard deviations



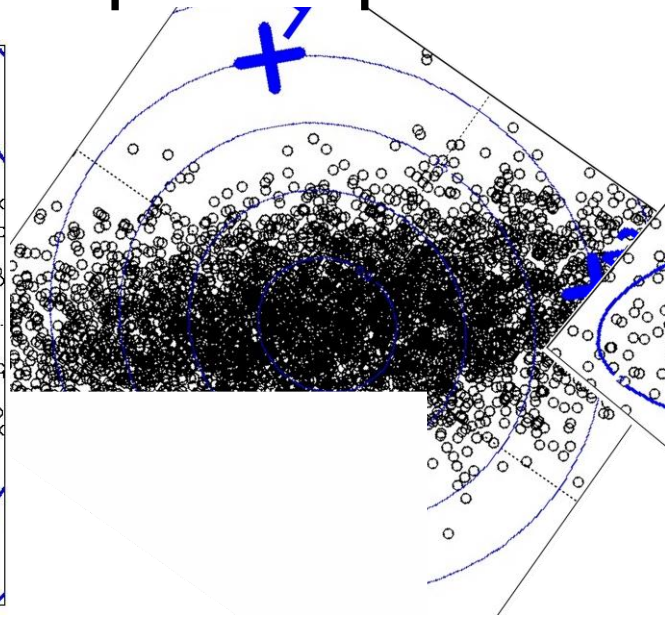
Picture: Equal M.D. Regions

- **Euclidean vs. Mahalanobis distance**

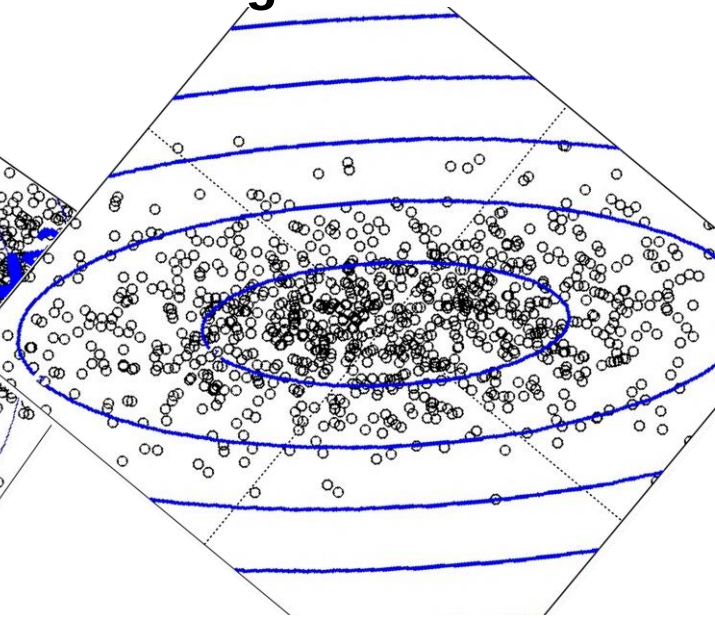
Contours of equidistant points from the origin



Uniformly distributed points, Euclidean distance



Normally distributed points, Euclidean distance

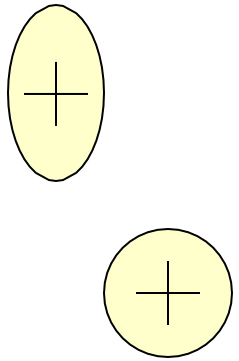


Normally distributed points, Mahalanobis distance

Should 2 CS clusters be combined?

Q2) Should 2 CS clusters be combined?

- Compute the variance of the combined subcluster
  - ***N***, ***SUM***, and ***SUMSQ*** allow us to make that calculation quickly
- Combine if the combined variance is below some threshold



- **Many alternatives:** Treat dimensions differently, consider density

Measuring Cluster Quality

1. **Internal cluster validation :** The clustering result is evaluated based on the data clustered itself (internal information) without reference to external information.
2. **External cluster validation :** Clustering results are evaluated based on some externally known result, such as externally provided class labels.
3. **Relative cluster validation :** The clustering results are evaluated by varying different parameters for the same algorithm (e.g., changing the number of clusters).

https://www.geeksforgeeks.org/dunn-index-and-db-index-cluster-validity-indices-set-1/

Measuring Cluster Quality

Internal Validation techniques:

1. Silhouette coefficient
2. Dunn Index -> Self-study
3. DBI (Davies–Bouldin index) -> Self-study

Silhouette Coefficient:

$$s_i = \frac{b_i - a_i}{max(b_i, a_i)}$$

where,

a_i = average distance of point *i* to other points in the same cluster (intra-cluster distance)

b_i = average distance of point *i* to other points belonging to the closest cluster, which point *i* is not a part of (inter-cluster distance)

Measuring Cluster Quality

$$\text{Dunn index}(U) = \min_{1 \le i \le c} \left\{ \min_{1 \le j \le c,\, j \ne i} \left\{ \frac{\delta(X_i, X_j)}{\max_{1 \le k \le c} [\Delta(X_k)]} \right\} \right\}$$

$\delta(X_i, X_j)$ is the intercluster distance i.e. the distance between cluster X_i and X_j

$\Delta(X_k)$ is the intracluster distance of cluster X_k i.e. distance within the cluster X_k

Measuring Cluster Quality

$$\text{DB index}(U) = 1/k \sum_{i=1}^{k} \max_{i \neq j} \left\{ \frac{\Delta(Xi) + \Delta(Xj)}{\delta(Xi, Xj)} \right\}$$

δ (Xi, Xj) is the intercluster distance i.e. the distance between cluster Xi and Xj

Δ(Xk) is the intracluster distance of cluster Xk i.e. distance within the cluster Xk

Summary

- **Clustering:** Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***
- **Algorithms:**
  - Agglomerative **hierarchical clustering**:
    - Centroid and clustroid
  - ***k*-means:**
    - Initialization, picking *k*
  - **BFR**
- **Cluster quality:** Silhouette Coefficient, Dunn Index, DBI