MIE524 Data Mining

Clustering

Slides Credits:
Slides from Leskovec, Rajaraman, Ullman (http://www.mmds.org), Leskovec & Ghashami
Alexandra Chouldechova, Roger Grosse, Amir-massoud Farahmand, and Juan Carrasquilla

MIE524: Course Topics (Tentative)

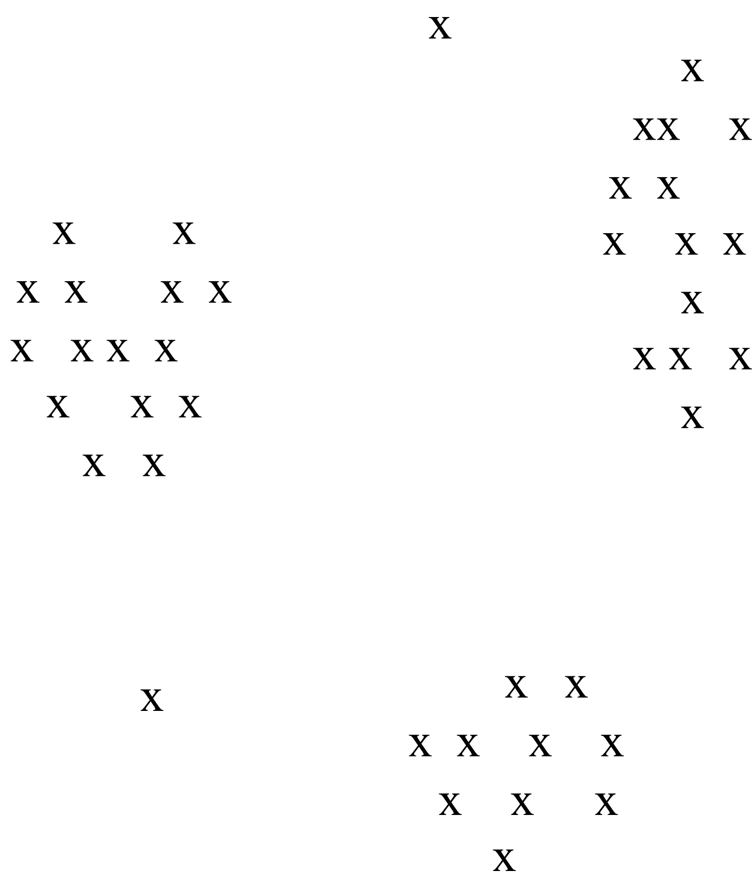
Large-scale Machine Learning	High-dimensional Data	Graph Data	Applications
Learning Embedding (NN / AE)	Locality sensitive hashing	Processing Massive Graphs	Recommender systems
Decision Trees	Clustering	PageRank, SimRank	Association Rules
Ensemble Models (GBTs)	Dimensionality reduction	Graph Representation Learning	Neural Language Models

Computational Models: Single Machine | MapReduce/Spark | GPU

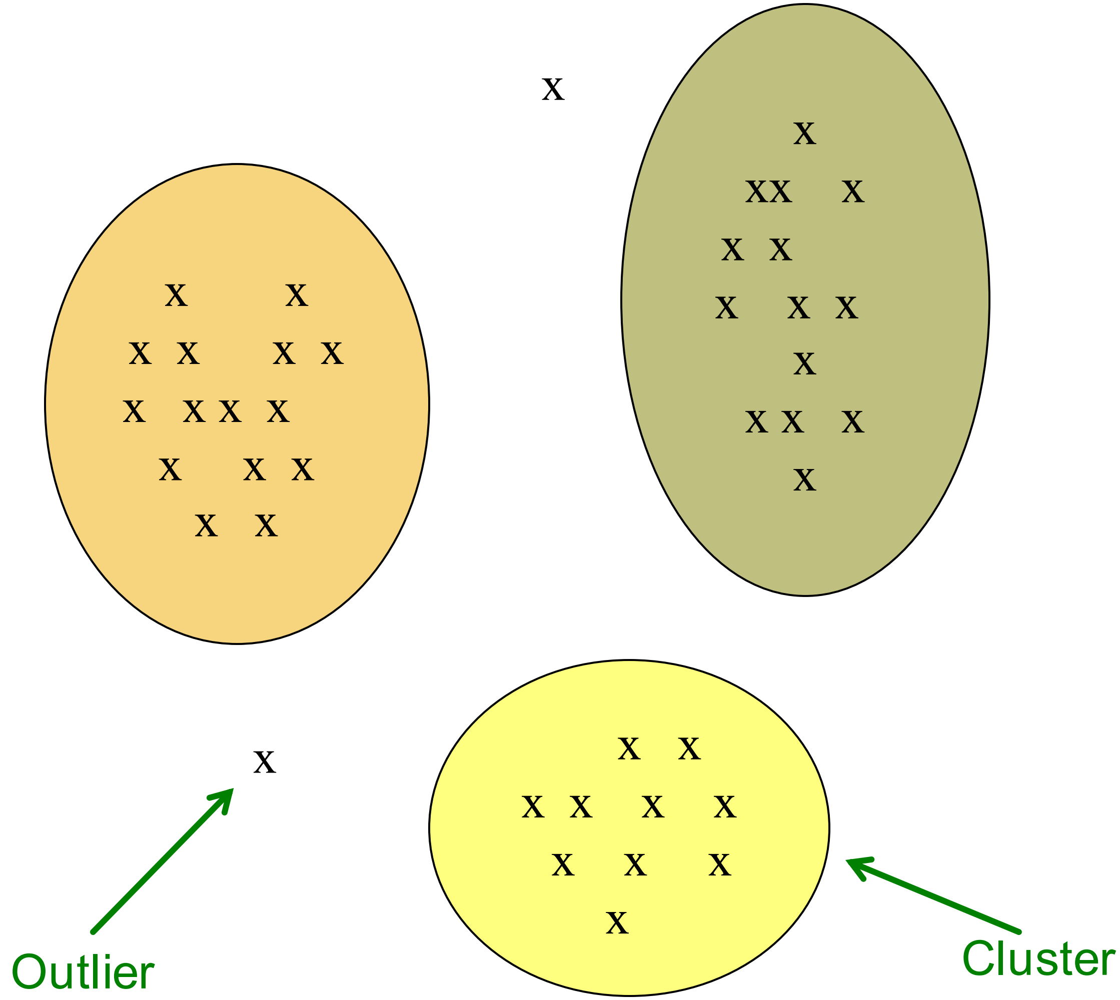
The Problem of Clustering

- 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



Example: Clusters & Outliers



Examples of clustering tasks

- Identify similar groups of online shoppers based on their browsing and purchasing history

- Identify similar groups of music listeners or movie viewers based on their ratings or recent listening/viewing patterns

- Cluster input variables based on their correlations to remove redundant predictors from consideration

- Cluster hospital patients based on their medical histories

- Determine how to place sensors, broadcasting towers, law enforcement, or emergency-care centers to guarantee that desired *coverage criteria* are met

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 $(x_1, x_2,\ldots, x_d)$, 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

Why is it hard?

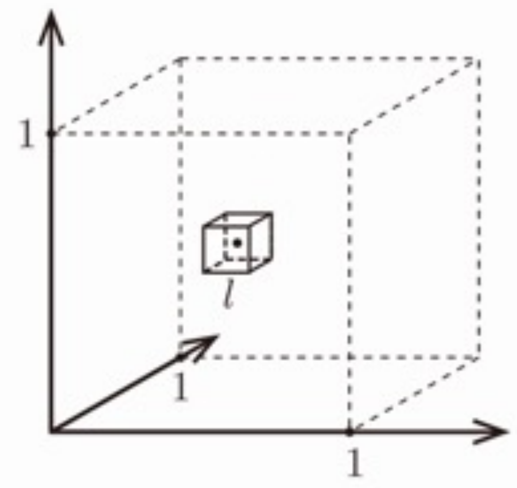
- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving

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

Example: Curse of Dimensionality

- **Take 10,000 uniform random points on [0,1] line. Assume query point is at the origin**
- **What fraction of "space" do we need to cover to get 0.1% of data** (10 nearest neighbors)
- In 1-dim to get 10 neighbors we must go to distance 10/10,000=0.001 on the average
- In 2-dim we must go $\sqrt{0.001}$=0.032 to get a square that contains 0.001 volume
- In general, in d-dim we must go $(0.001)^{\frac{1}{d}}$
- So, in 10-dim to capture 0.1% of the data we need 50% of the range.

Curse of dimensionality



Formally, imagine the unit cube $[0,1]^d$. All training data is sampled *uniformly* within this cube, i.e. $\forall i, x_i \in [0,1]^d$, and we are considering the $k = 10$ nearest neighbors of such a test point.

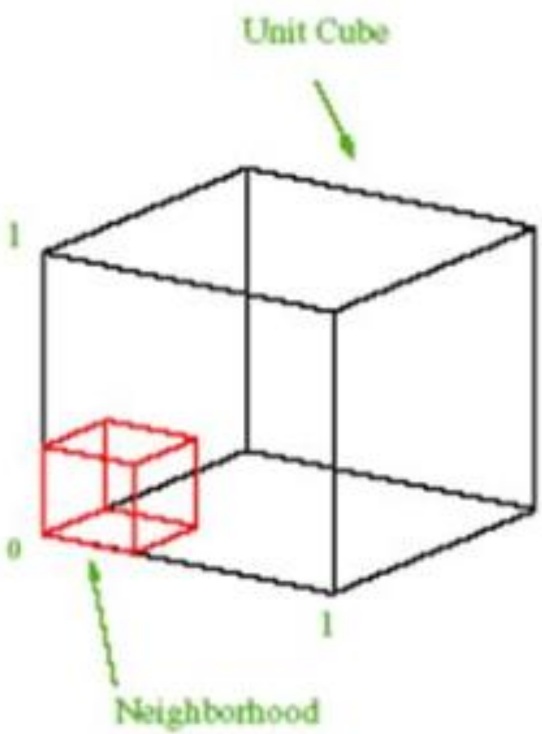
Let ℓ be the edge length of the smallest hyper-cube that contains all k-nearest neighbor of a test point. Then $\ell^d \approx \frac{k}{n}$ and $\ell \approx \left(\frac{k}{n}\right)^{1/d}$. If $n = 1000$, how big is ℓ?

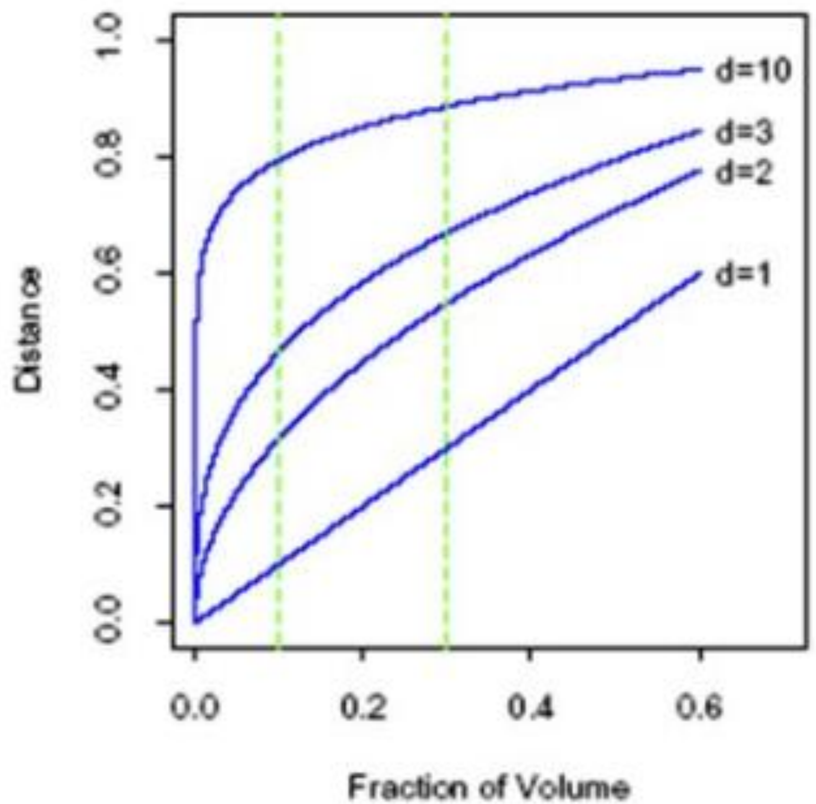
d	ℓ
2	0.1
10	0.63
100	0.955
1000	0.9954

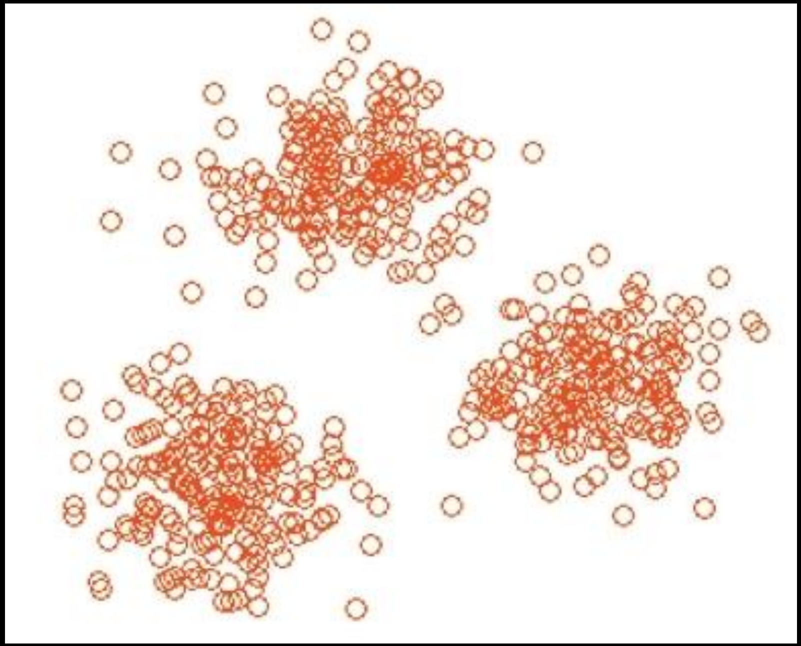
https://www.cs.cornell.edu/courses/cs4780/2022fa/lectures/lecturenote02_kNN.html

Example: Curse of Dimensionality

Curse of Dimensionality: All points are very far from each other







k-means Clustering

Method: K-mean clustering

- Main idea: A good clustering is one for which the within-cluster variation is as small as possible.

- The within-cluster variation for cluster C_k is some measure of the amount by which the observations within each class differ from one another

- We'll denote it by $WCV(C_k)$

- Goal: Find $C_1, \ldots, C_K$ that minimize

$$\sum_{k=1}^{K} \mathrm{WCV}(C_k)$$

- This says: Partition the observations into K clusters such that the WCV summed up over all K clusters is as small as possible

How to define within-cluster variation?

- Goal: Find $C_1, \ldots, C_K$ that minimize

$$\sum_{k=1}^{K} \mathrm{WCV}(C_k)$$

- Typically, we use squared Euclidean distance:

$$\mathrm{WCV}(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

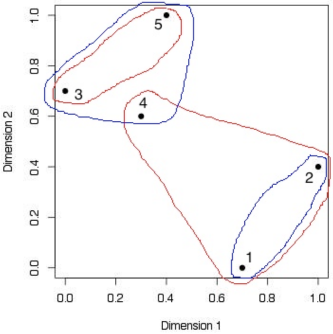
where $|C_k|$ denotes the number of observations in cluster k

- To be clear: We're treating K as fixed ahead of time. We are *not* optimizing K as part of this objective.

Simple example

Here $n = 5$ and $K = 2$,
The full *distance matrix* for all 5 observations is shown below.

	1	2	3	4	5
1	0	0.25	0.98	0.52	1.09
2	0.25	0	1.09	0.53	0.72
3	0.98	1.09	0	0.10	0.25
4	0.52	0.53	0.10	0	0.17
5	1.09	0.72	0.25	0.17	0



- Red clustering: $\sum \mathrm{WCV}_k = (0.25 + 0.53 + 0.52)/3 + 0.25/2 = 0.56$
- Blue clustering: $\sum \mathrm{WCV}_k = 0.25/2 + (0.10 + 0.17 + 0.25)/3 = 0.30$
- It's easy to see that the Blue clustering minimizes the within-cluster variation among all possible partitions of the data into $K = 2$ clusters

How do we minimize WCV?

$$\sum_{k=1}^{K} \mathrm{WCV}(C_k) = \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$
$$= \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|_2^2$$

- It's *computationally infeasible* to actually minimize this criterion
- We essentially have to try all possible partitions of n points into K sets.
- When $n = 10$, $K = 4$, there are $34{,}105$ possible partitions
- When $n = 25$, $K = 4$, there are $5 \times 10^{13}\ldots$
- We're going to have to settle for an approximate solution

K-means algorithm

- It turns out that we can rewrite WCV_k more conveniently:

$$\mathrm{WCV}_k = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|_2^2 = 2 \sum_{i \in C_k} \|x_i - \bar{x}_k\|^2$$

where $\bar{x}_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$ is just the average of all the points in cluster C_k

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

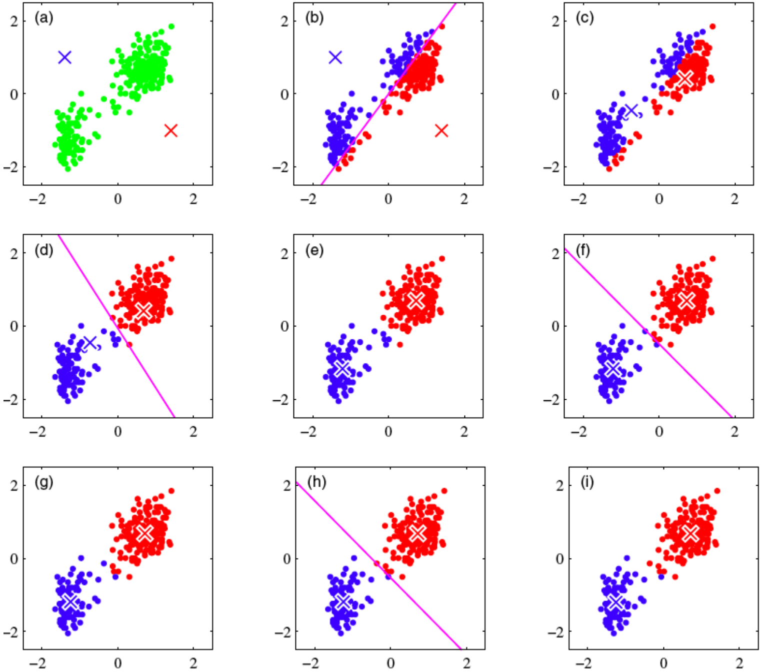
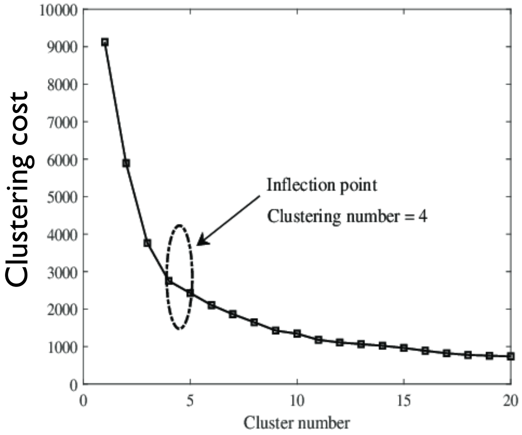


Figure from Bishop

Simple demo: http://syskall.com/kmeans.js/

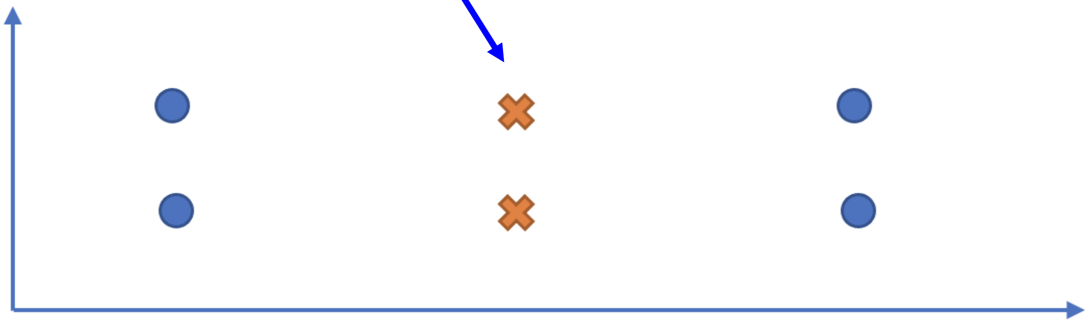
How to choose K?



[Source]

Shortcoming of k-means

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



Summary of K-means

We'd love to minimize

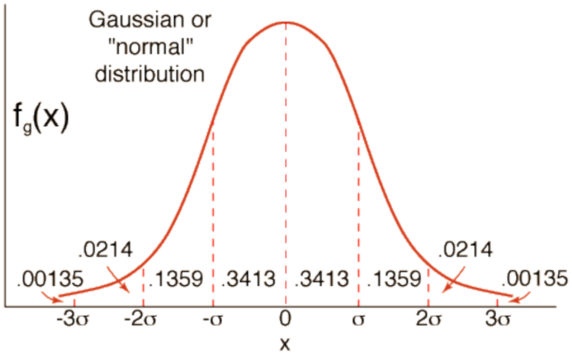
$$\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|_2^2$$

- It's *infeasible* to actually optimize this in practice, but K-means at least gives us a so-called local optimum of this objective
- The result we get depends both on K, and also on the *random initialization* that we wind up with
- It's a good idea to try different random starts and pick the best result among them
- There's a method called K-means++ that improves how the clusters are initialized

The BFR Algorithm

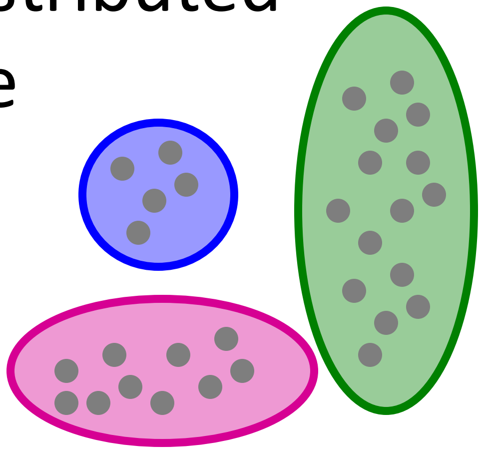
Extension of *k*-means to large data

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.

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 within some distance threshold of the cluster
  - **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

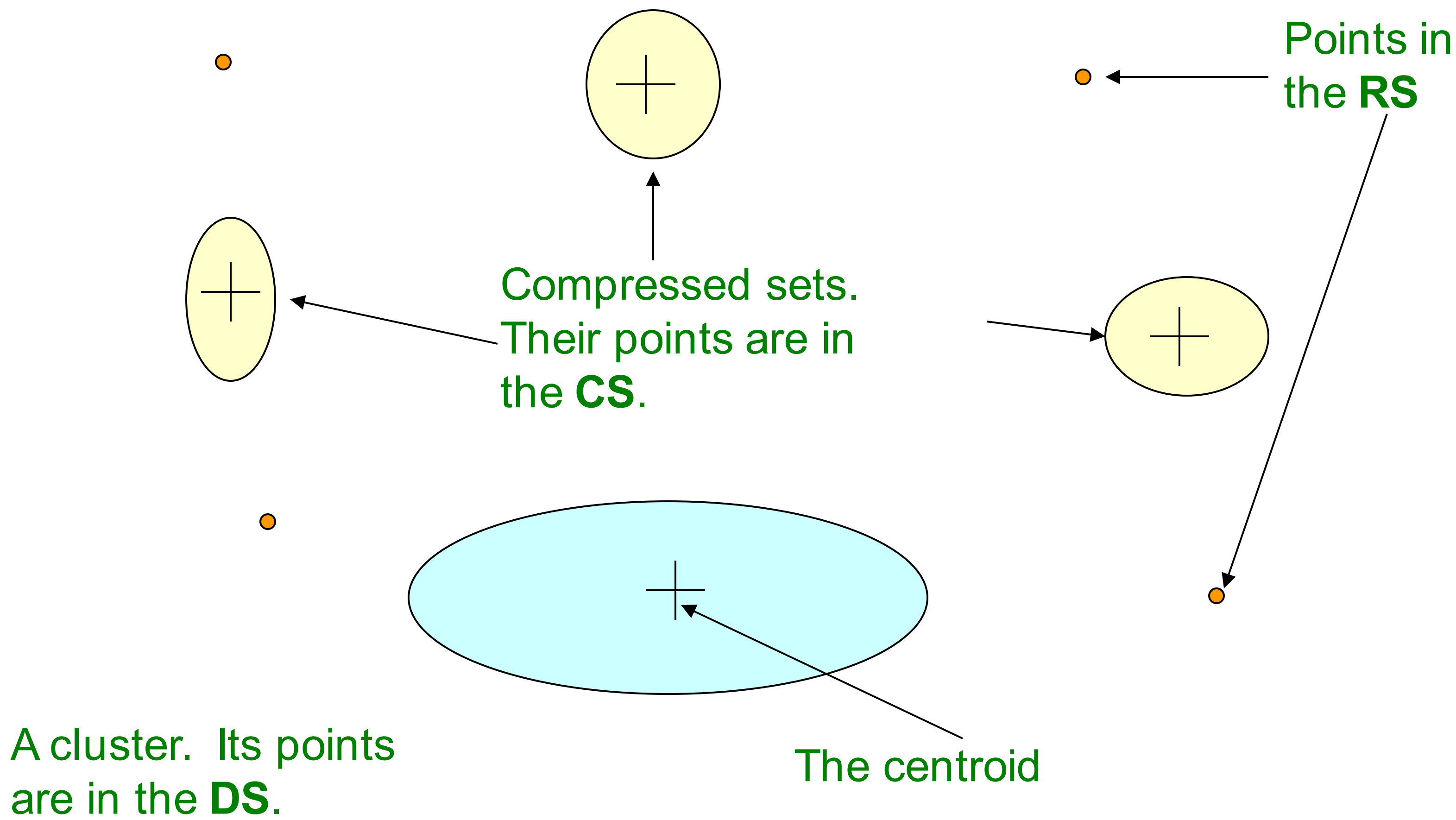
- **Points are read from disk one main-memory-full at a time**
- **Most points from previous memory loads are summarized by simple statistics**
- **Step 1)** From the initial load we select the initial ***k*** centroids by some sensible approach:
  - Take ***k*** random points
  - Take a small random 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 compression set

BFR: “Galaxies” Picture

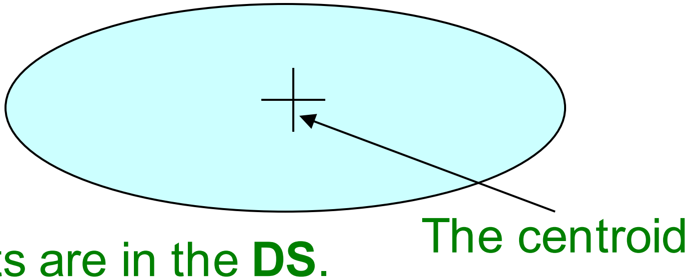


Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points, we store them as they are

Summarizing Sets of Points

For each cluster, the discard set (DS) is <u>summarized</u> 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

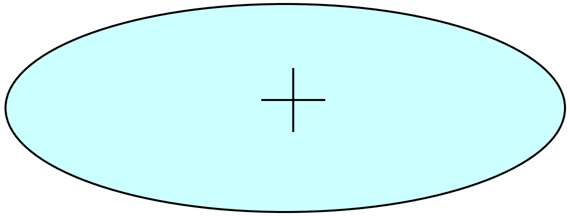


A cluster.
All its points are in the **DS**.

Summarizing Points: Comments

- **2*d* + 1** values represent any size cluster
  - ***d*** = number of dimensions
- Average in **each dimension** (**the centroid**) can be calculated as $\mathbf{SUM}_i\,/\,N$
  - $\mathbf{SUM}_i$ = i^{th} component of SUM
- Variance of a cluster's discard set in dimension *i* is: $(\mathbf{SUMSQ}_i\,/\,N) - (\mathbf{SUM}_i\,/\,N)^2$
  - 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

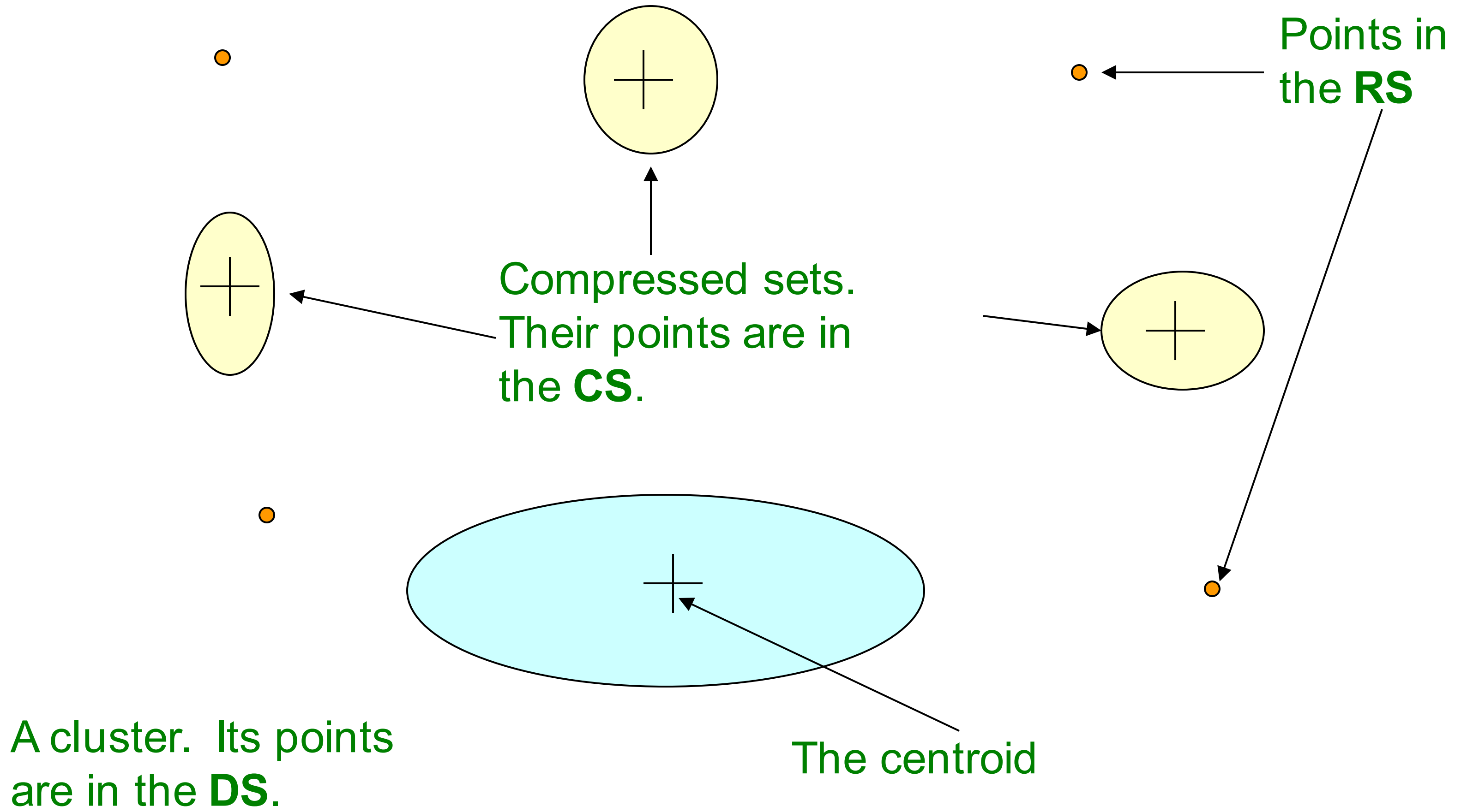
The "Memory-Load" of Points

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 CS
- **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)**

  - 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

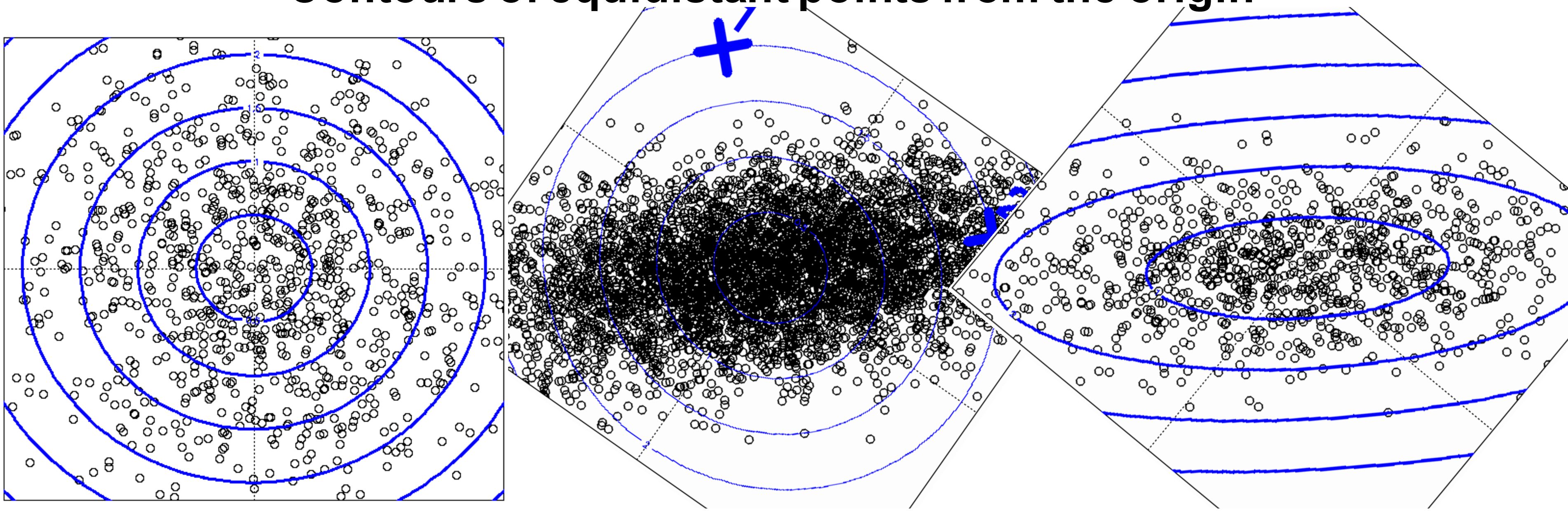
$$d(x,c) = \sqrt{\sum_{i=1}^{d} \left(\frac{x_i - c_i}{\sigma_i} \right)^2}$$

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

Picture: Equal M.D. Regions

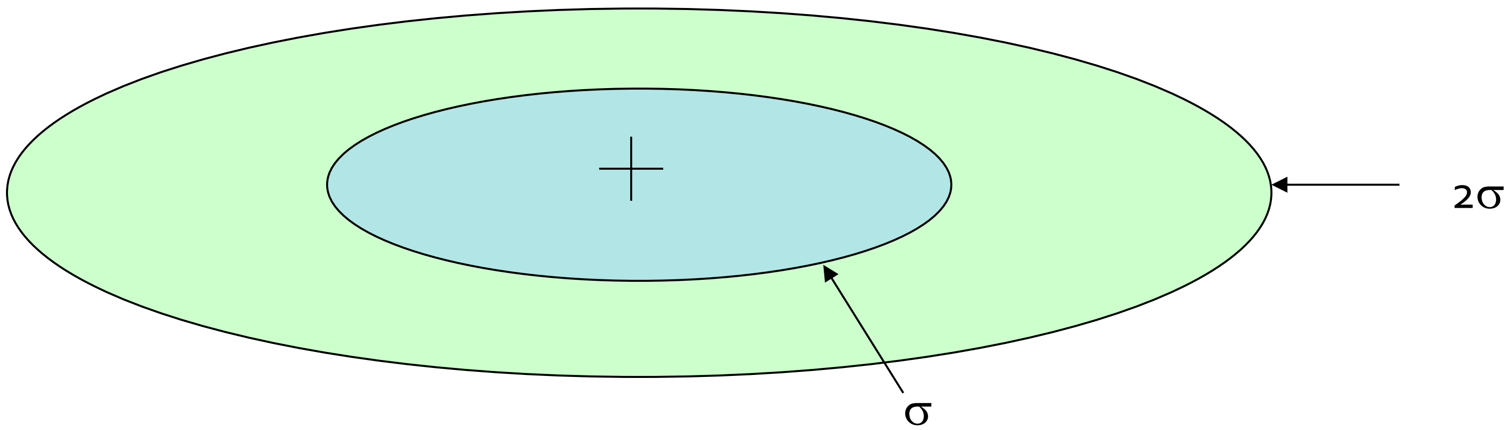
- **Euclidean vs. Mahalanobis distance**

Contours of equidistant points from the origin



Uniformly distributed points, Euclidean distance | **Normally distributed points, Euclidean distance** | **Normally distributed points, Mahalanobis distance**

Picture: Equal M.D. Regions

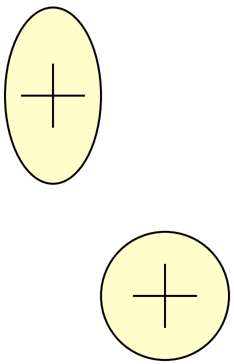


Accept a point for a cluster if its M.D. is < some threshold, e.g., 2 standard deviations

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