AIM3 – Scalable Data Analysis and Data Mining

Clustering

Christoph Boden, Sebastian Schelter, Juan Soto, Volker Markl

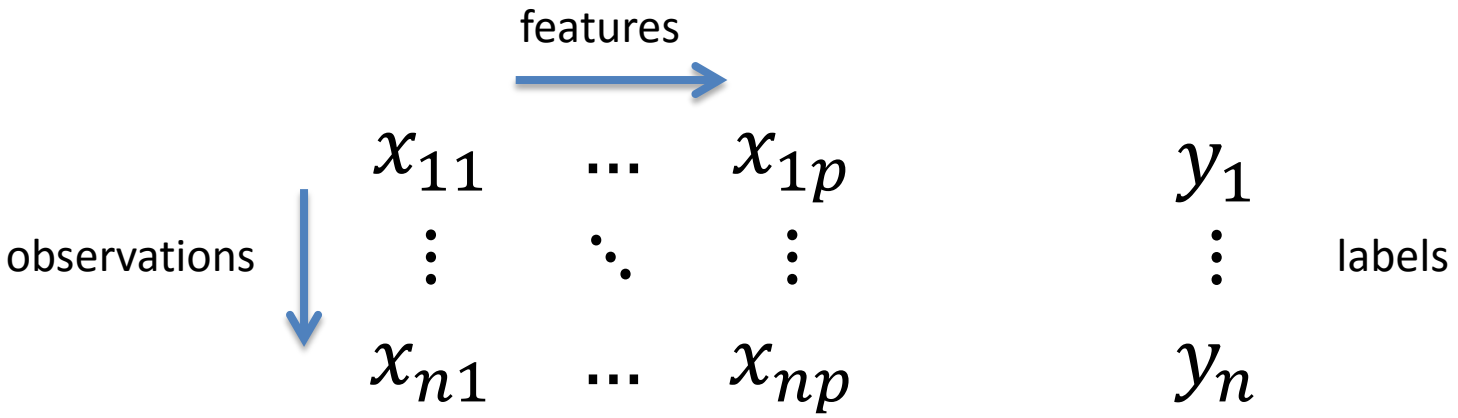
Includes Material from: Jeff Ullman, Jure Leskovec (Stanford University), Sriram Sankararaman (UC Berkeley), Junming Yin (University of Arizona)

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

http://www.dima.tu-berlin.de/

Machine Learning

- No new field per say - rather a loose confederation of „themes“ in statistical inference and decision theory
- Focus on prediction and exploratory data analysis
- Focus on computational methodolodgy and empirical evaluation
- Data Matrix: Object x Attributes (Continous, Categorical, …)



Feature Extraction

input data **features** **output**

$$\begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix}$$

“Danger”

$$\begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix}$$

Cat

Supervised learning vs. unsupervised learning

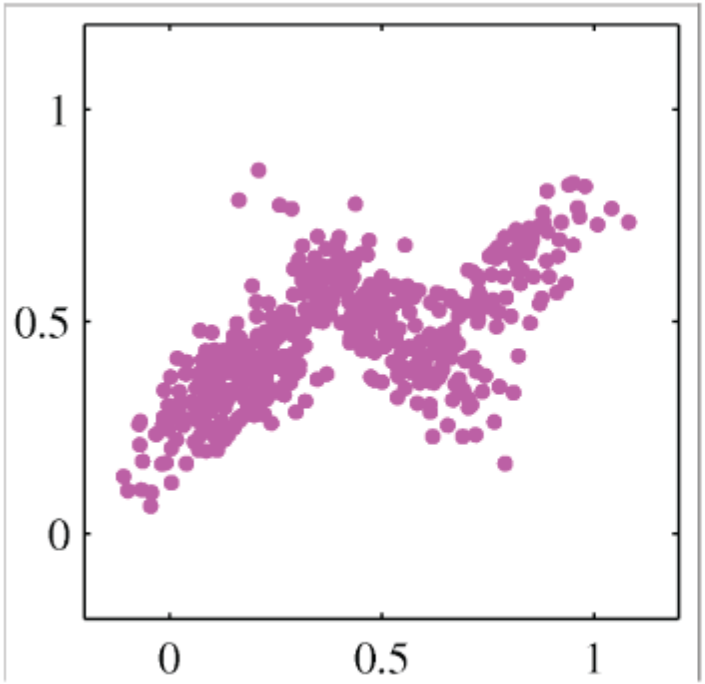
- Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.
  - These patterns are then utilized to predict the values of the target attribute in future data instances.
  - e.g.: Email marked „SPAM“, Image with Keywords, Face with Name, DNA with Genes marked, …

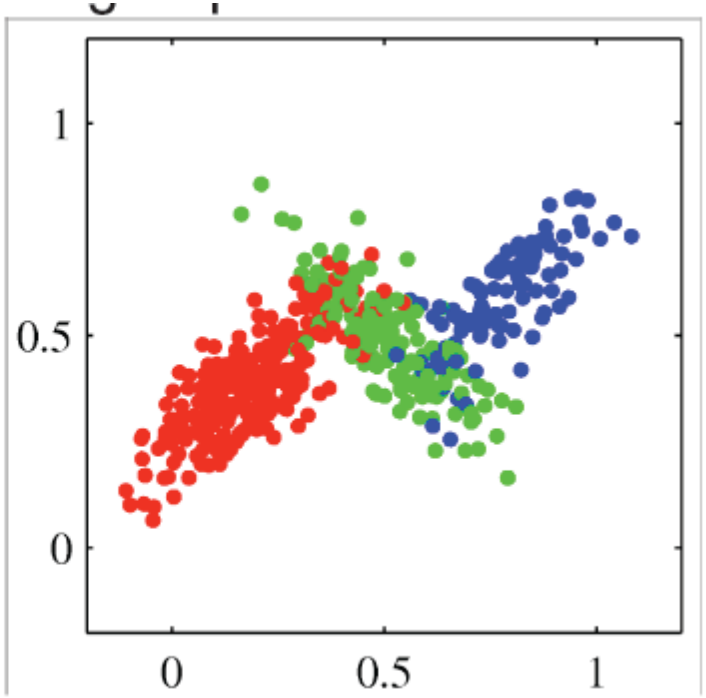
- Unsupervised learning: The data have no target attribute.
  - We want to explore the data to find some intrinsic structures in them.

Clustering

- Roughly speaking, clustering analysis aims to discover distinct clusters or groups of samples such that samples within the same group are more similar to each other than they are to the members of other groups
-
  - a *dissimilarity (similarity)* function between samples
  - a *criterion* to evaluate a groupings of samples into clusters
  - an *algorithm* that optimizes this criterion function

Example





Application: Image Segmentation

- Image segmentation: decompose the image into regions with coherent color and texture inside them

http://people.cs.uchicago.edu/ pff/segment

Application: Detecting Galaxies

Clustering Examples

- Cluster the refinements of a user search query
- Cluster Web search results
- Categorize news stories
- Web Shop: Cluster products according to categories
- DNA sequences based on edit distance

...

Issues with Clustering

- Different types of attribute types (not just numerical)
- Input parameters (e.g. how many clusters)
- Shape of Clusters
- Sensitivity to input order and Incremental Clustering (updates)
- Scalability (obviously)
  - Many applications involve not 2, but 10 or 10,000 dimensions
  - High-dimensional spaces look different!

Curse of dimensionality

→ when the dimensionality increases, the volume of the space increases so fast that the available data become sparse.

- in high dimensions, almost all pairs of points are equally far away from one another.

- in high dimensions, almost any two vectors are almost orthogonal

The curse of dimensionality

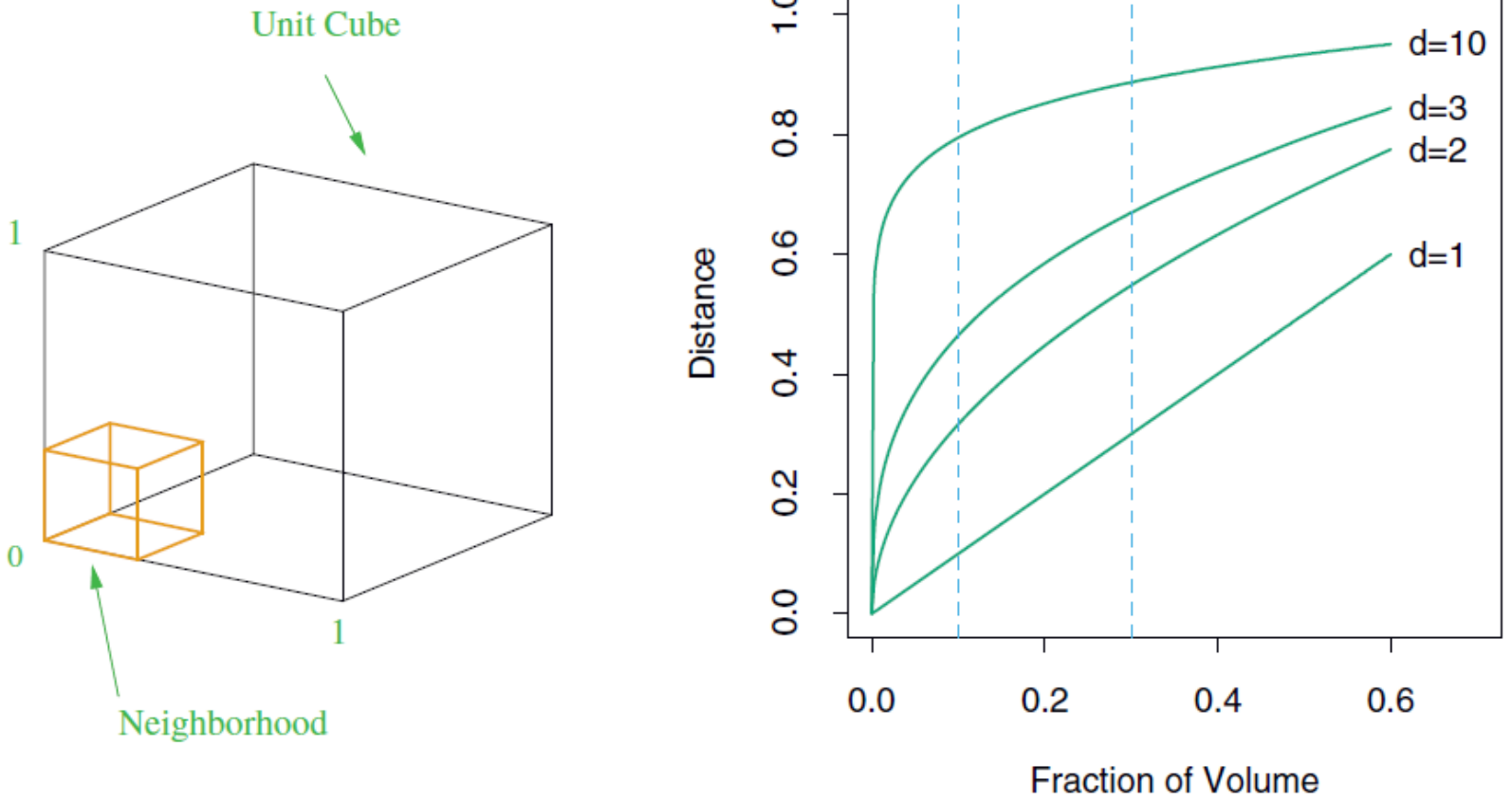


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.*

Clustering

- Roughly speaking, clustering analysis aims to discover distinct clusters or groups of samples such that samples within the same group are more similar to each other than they are to the members of other groups
  - a *dissimilarity (similarity)* function between samples
  - a *criterion* to evaluate a groupings of samples into clusters
  - an *algorithm* that optimizes this criterion function

ReFresher: Distance Meassures

- $d(x, y) \geq 0$ *(no negative distances)*
- $d(x, y) = 0$ if and only if $x = y$ *(distances are positive, except for the distance from a point to itself)*
- $d(x, y) = d(y, x)$ *(distance is symmetric)*
- $d(x, y) \leq d(x, z) + d(z, y)$ *(the triangle inequality)*

Cosine, Jaccard, and Euclidean Distances

- Different notions of similarity
  - Sets as vectors: measure similarity by the cosine distance.

$$d_{Cosine} = \frac{x \cdot y}{\sqrt{\sum_i x_i{}^2 \sum_i y_i{}^2}}$$

  - Sets as sets: measure similarity by the Jaccard distance.

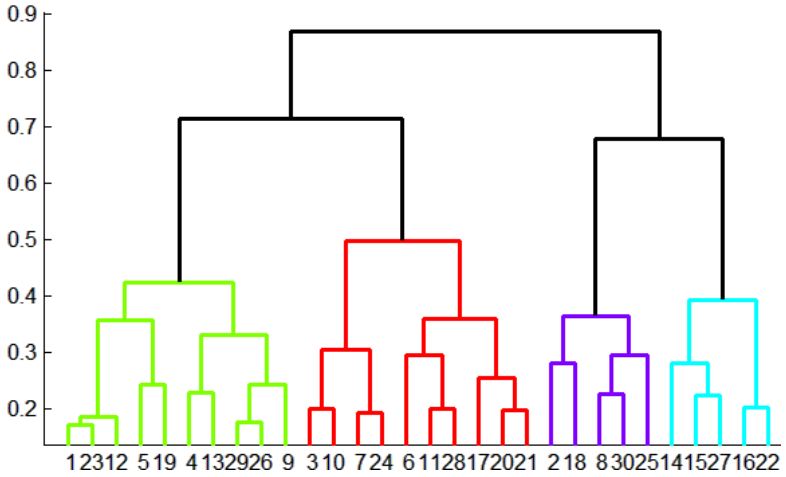
$$d_{Jaccard}(A, B) = \frac{(A \cap B)}{(A \cup B)}$$

  - Sets as points: measure similarity by Euclidean distance.

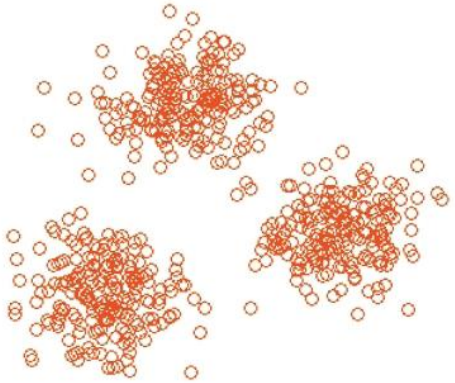
$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Methods of Clustering I

- Hierarchical (Agglomerative):
  - 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.
  - Place points into their "nearest" cluster.



Methods of Clustering (II)

- Partitioning Methods (k-means, k-mediods …)
  - Find mutually exclusie clusters of spherical shapes
  - Distance-based
  - May use mean or mediod to represent cluster center
  - Effektive for small- to medium-size data sets

- Density based methods (DBSCAN)
  - Can find arbitrary shaped cluster
  - Clusters are dense regions of objects that are seüarated by low density regisions
  - May filter out outliers

- Hierarchical methods (HAC, BIRCH …)
  - Clustering is a hierarchical decomposition
  - Cannot correct erroneous merges or splits
  - May incorporateother techniques like microclustering

Partition Based Clustering: k-means

- Assume data lives in eucleadian space
- Use centroid c of a cluster to represent a cluster
  - Potential centroids: mean, mediod …
- Define objective:

$$min \sum_{j=1}^{k} \sum_{i \in C_j} distance(x, centroid)$$

$$min \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2$$

$$O(n^{dk+1} logn)$$

K-Means Algorithm

Algorithm *k*-means(*k*, *D*)

choose *k* data points as the initial centroids (cluster centers)

repeat

for each data point **x** ∈ *D* do

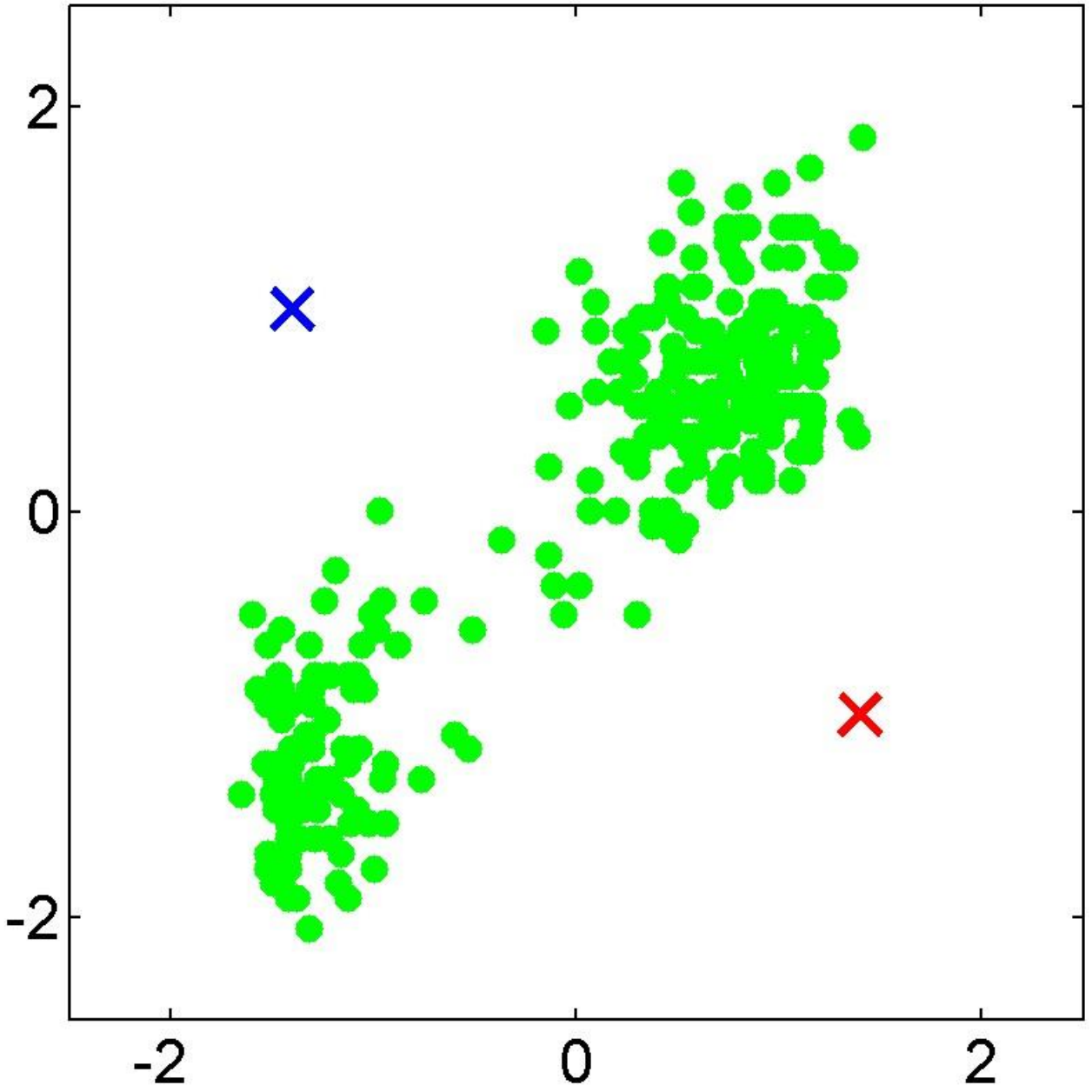
compute the distance from **x** to each centroid;

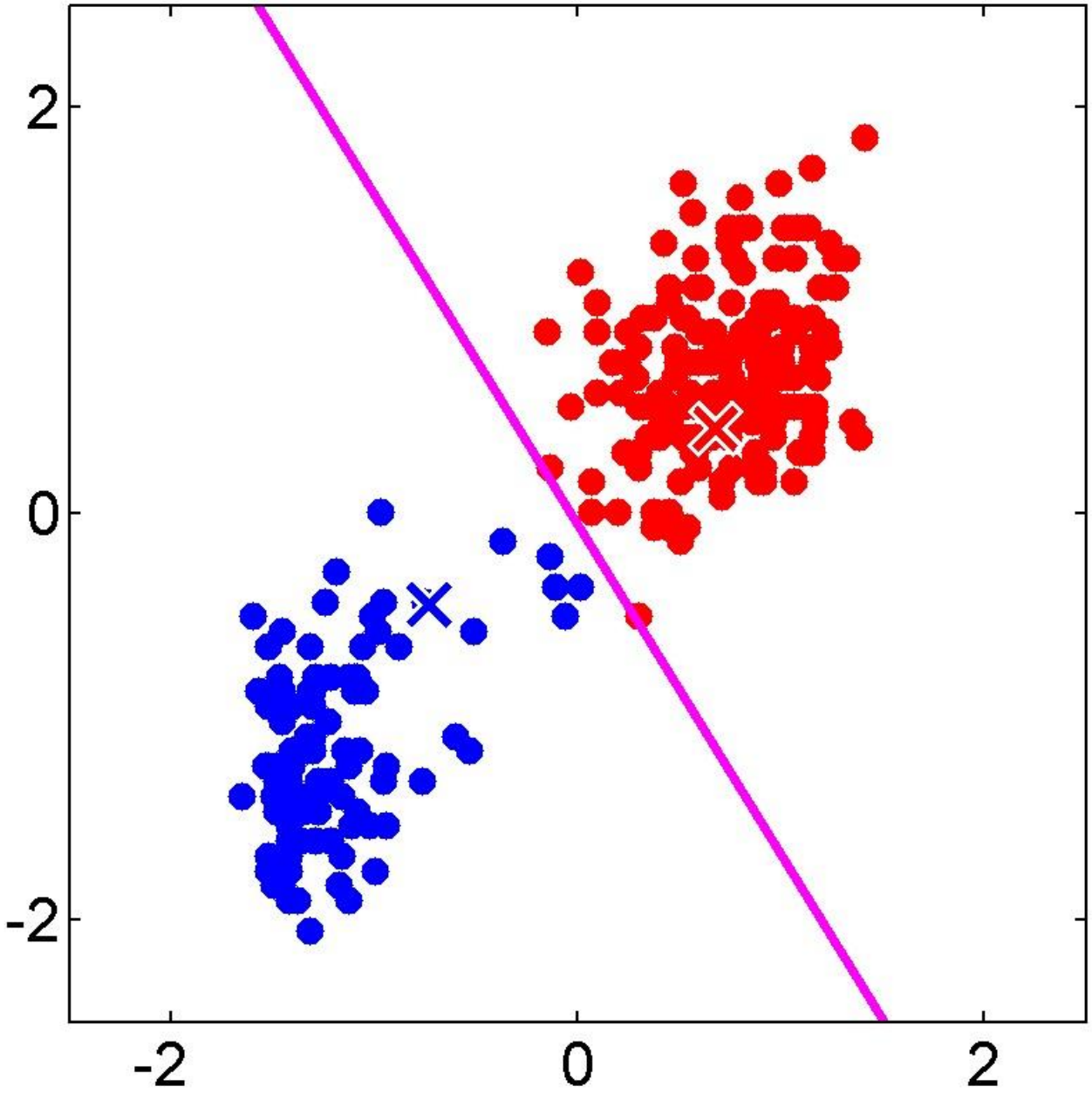
assign **x** to the closest centroid

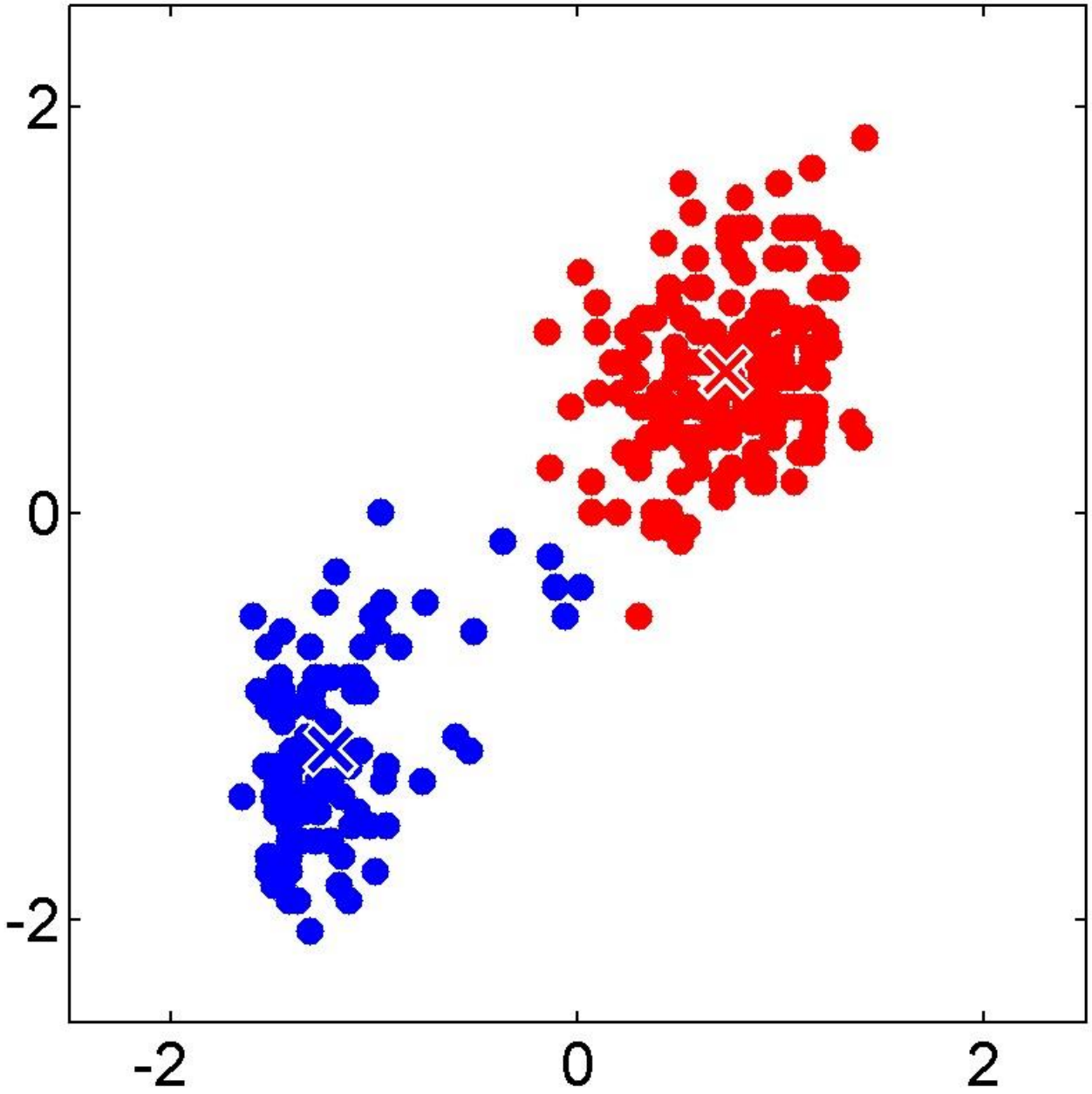
endfor

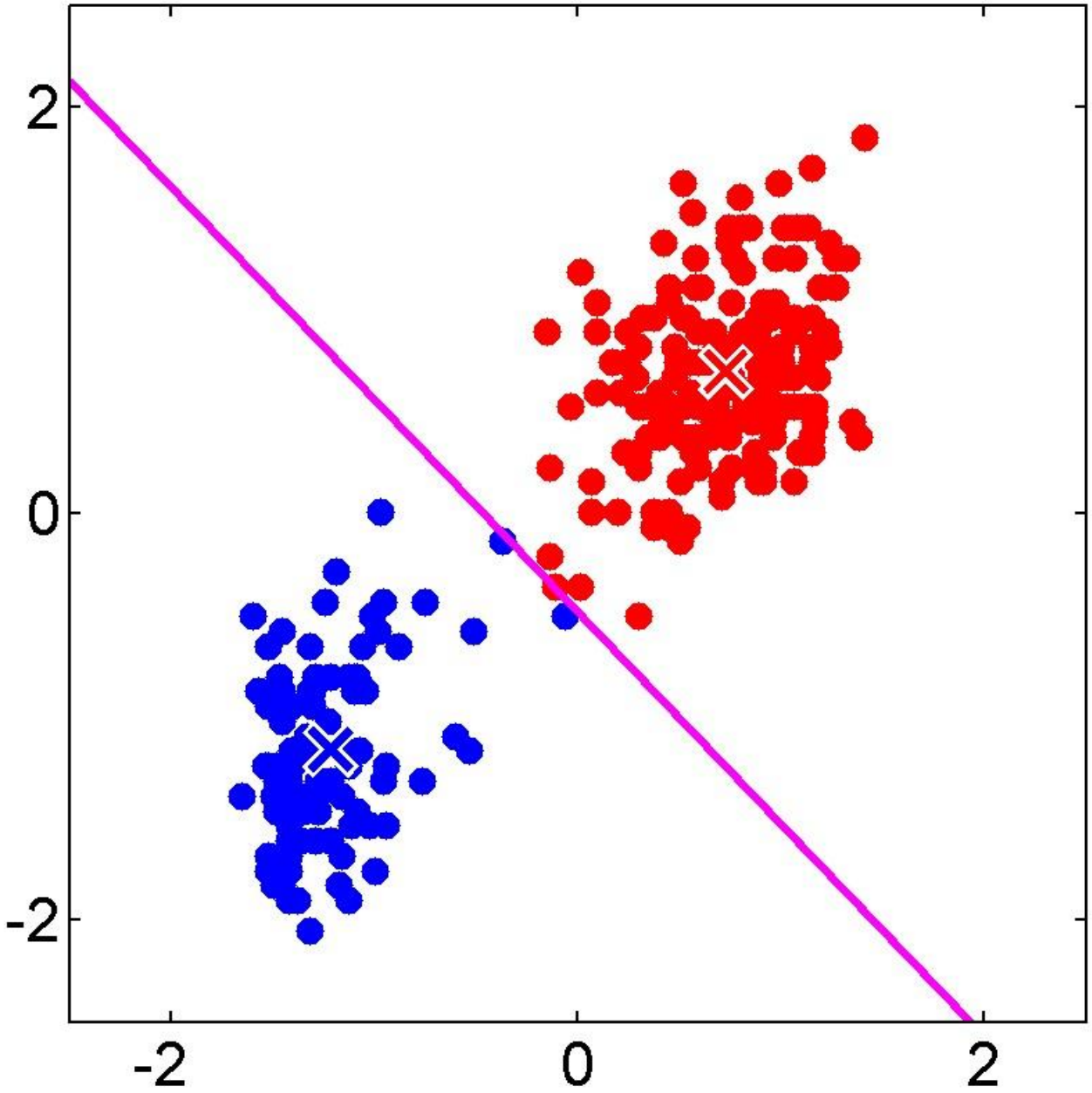
re-compute the centroid using the current cluster memberships

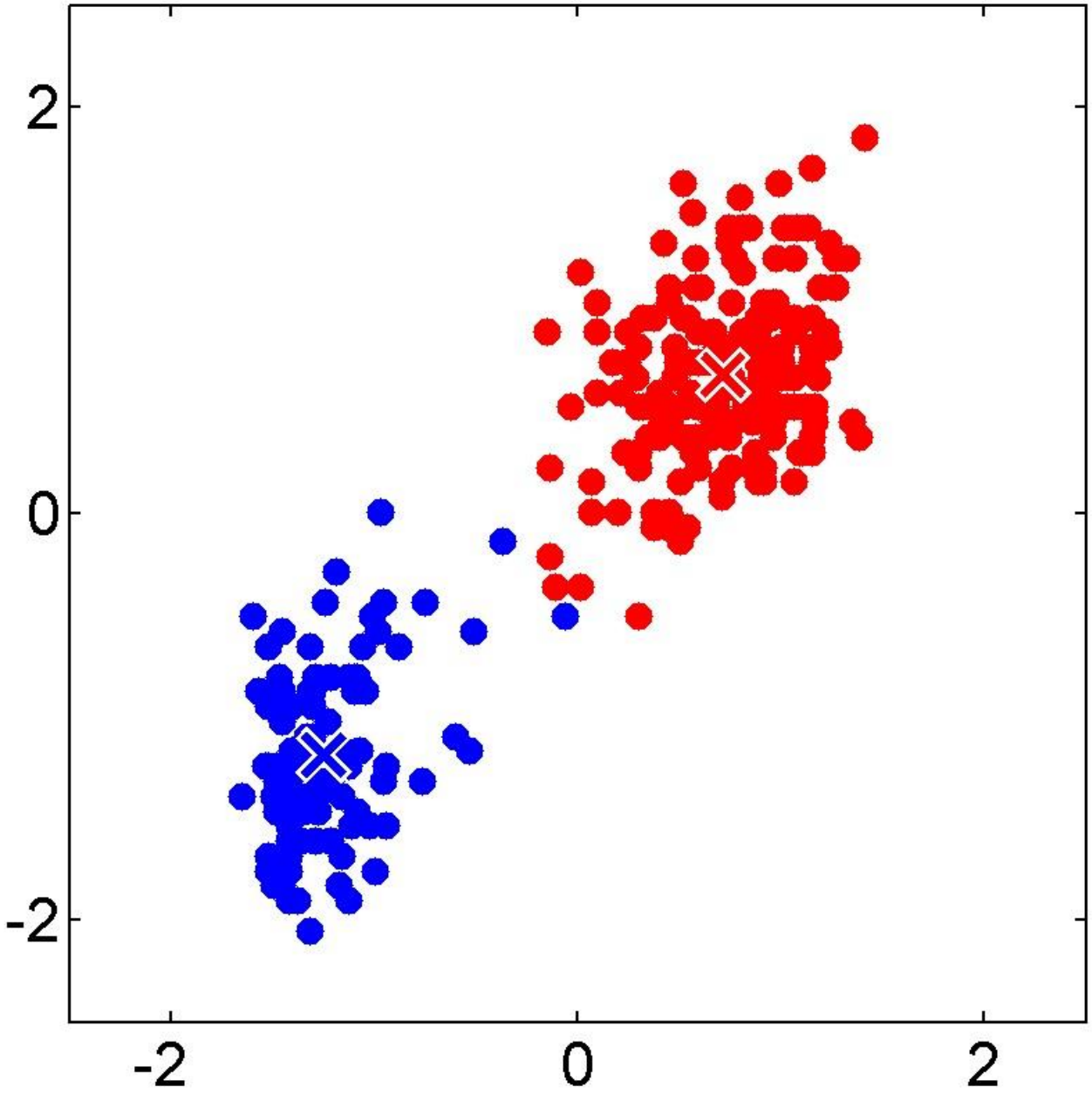
until the stopping criterion is met

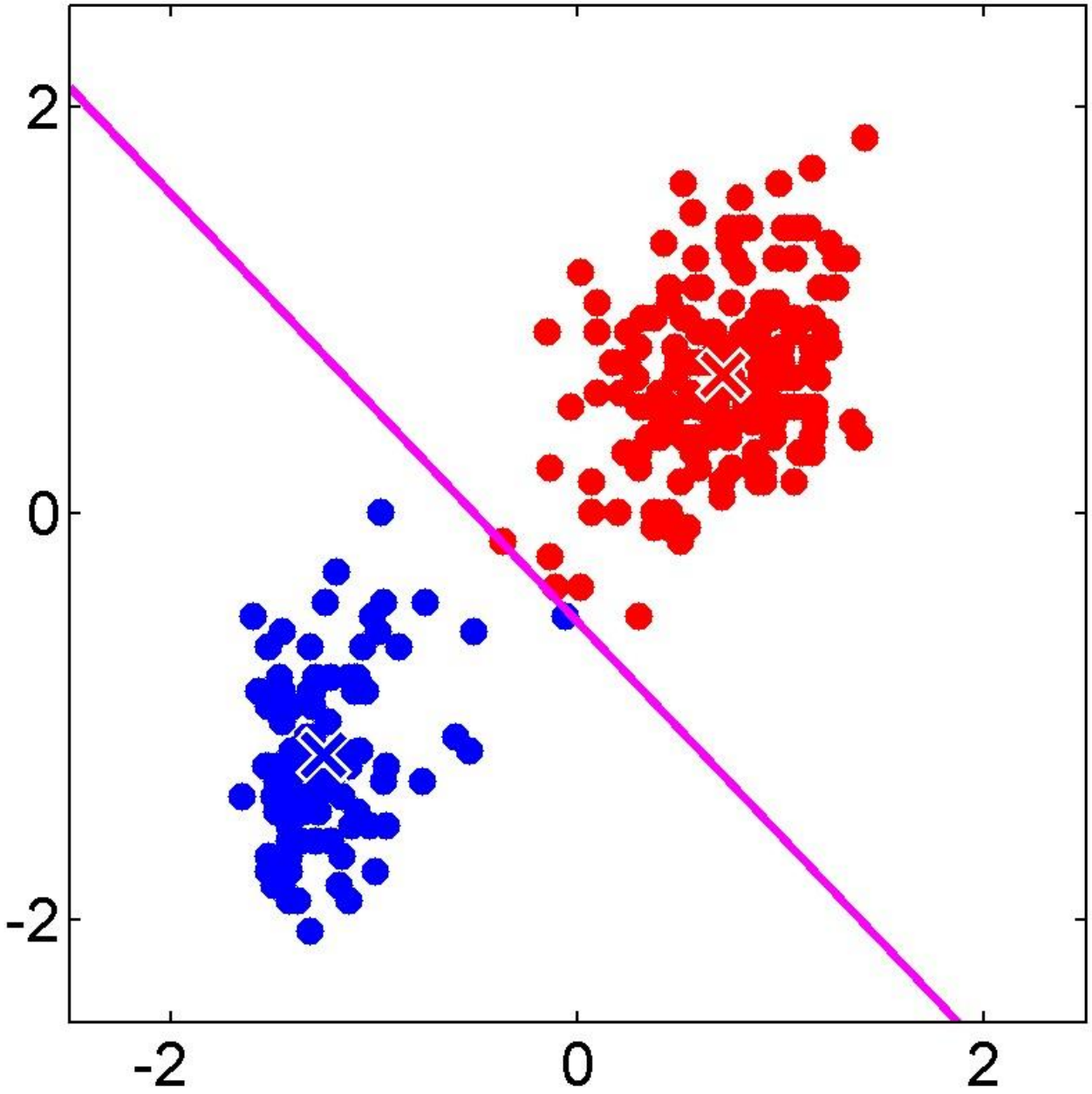

2
0
-2
-2
0
2

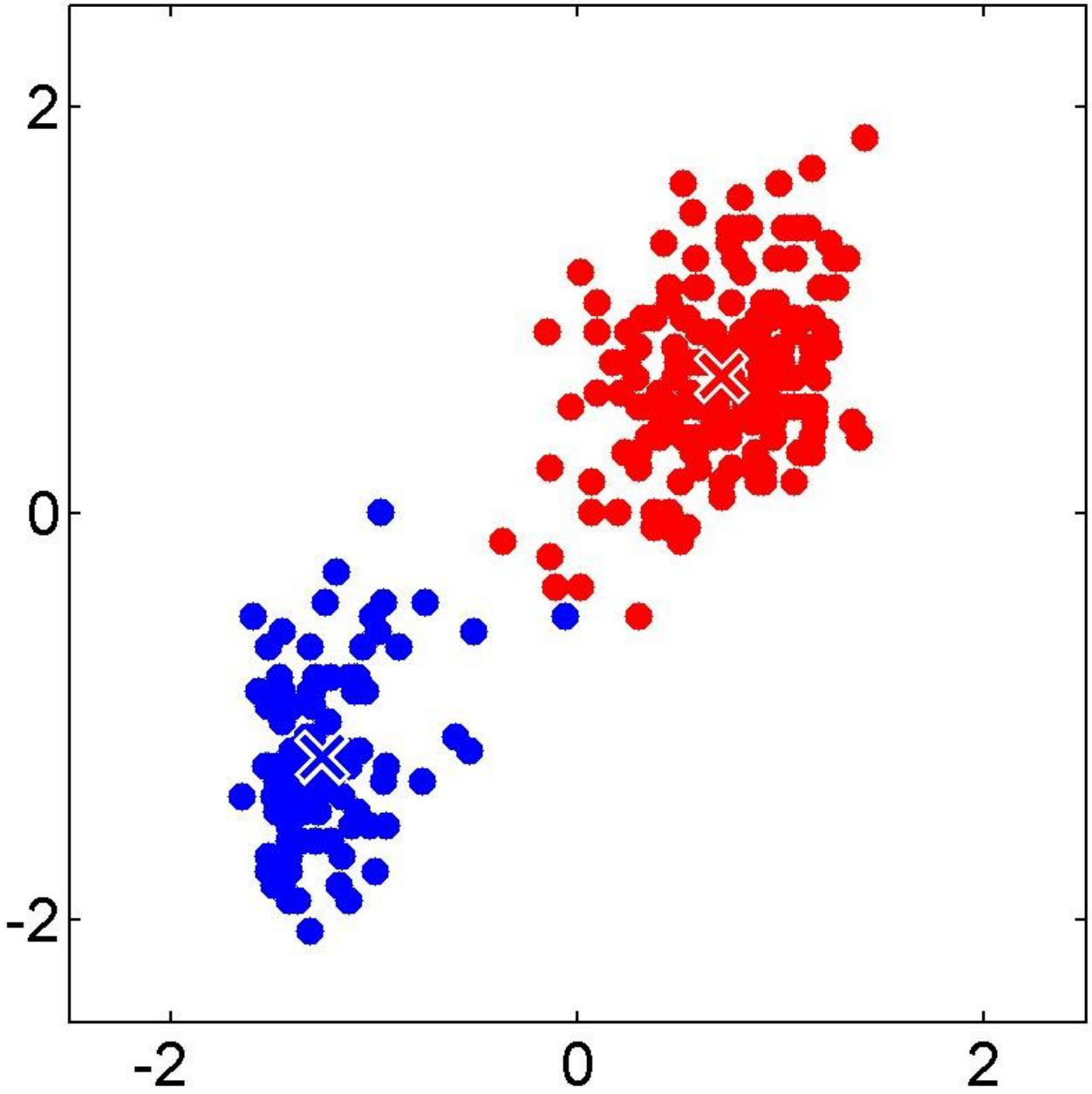

2
0
-2
-2
0
2


2
0
-2
-2
0
2


2
0
-2
-2
0
2

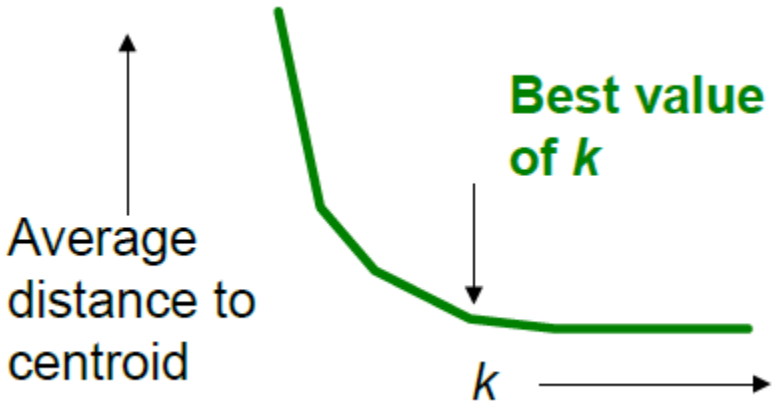

2
0
-2
-2
0
2


2
0
-2
-2
0
2


2
0
-2
-2
0
2

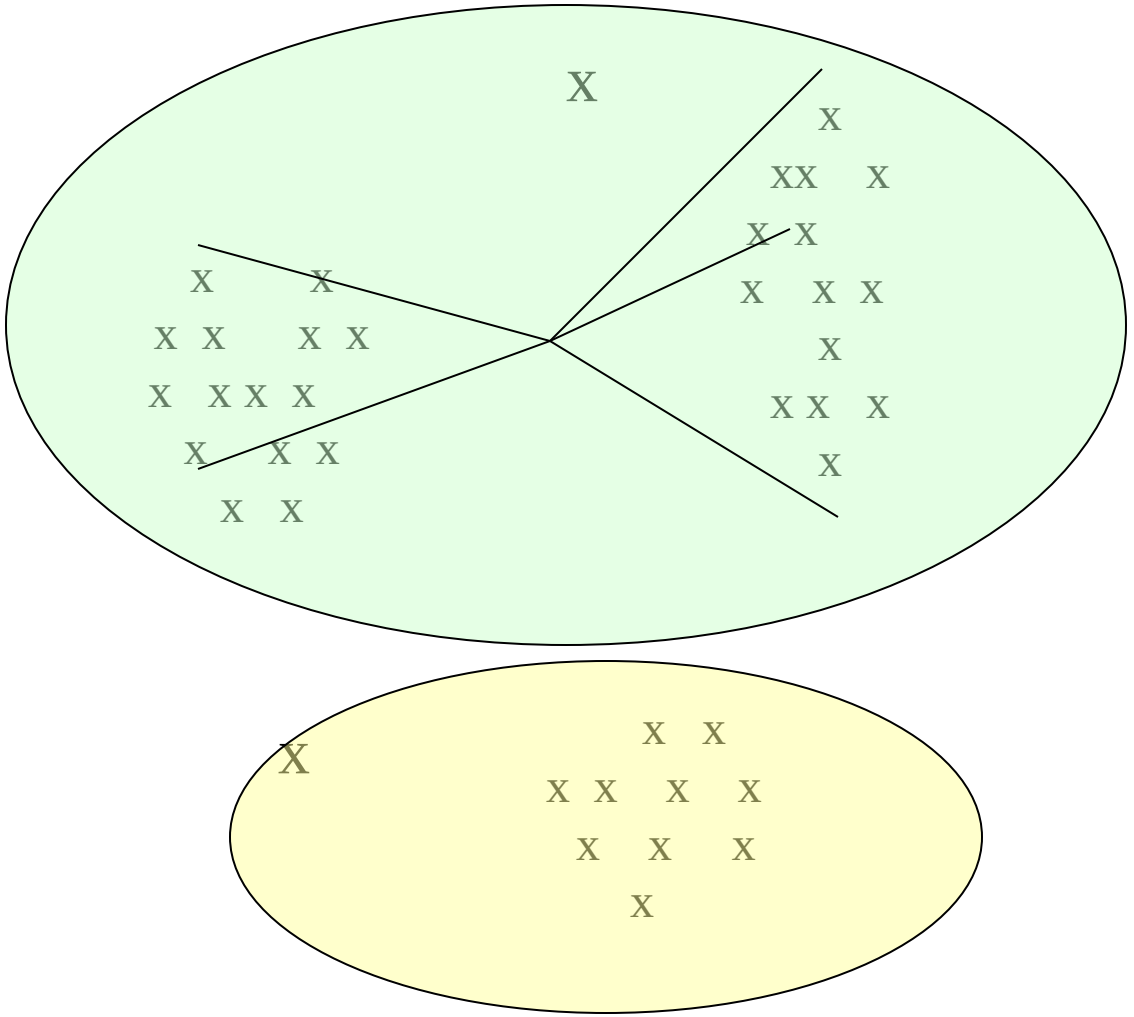
Getting the *k* right

- **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



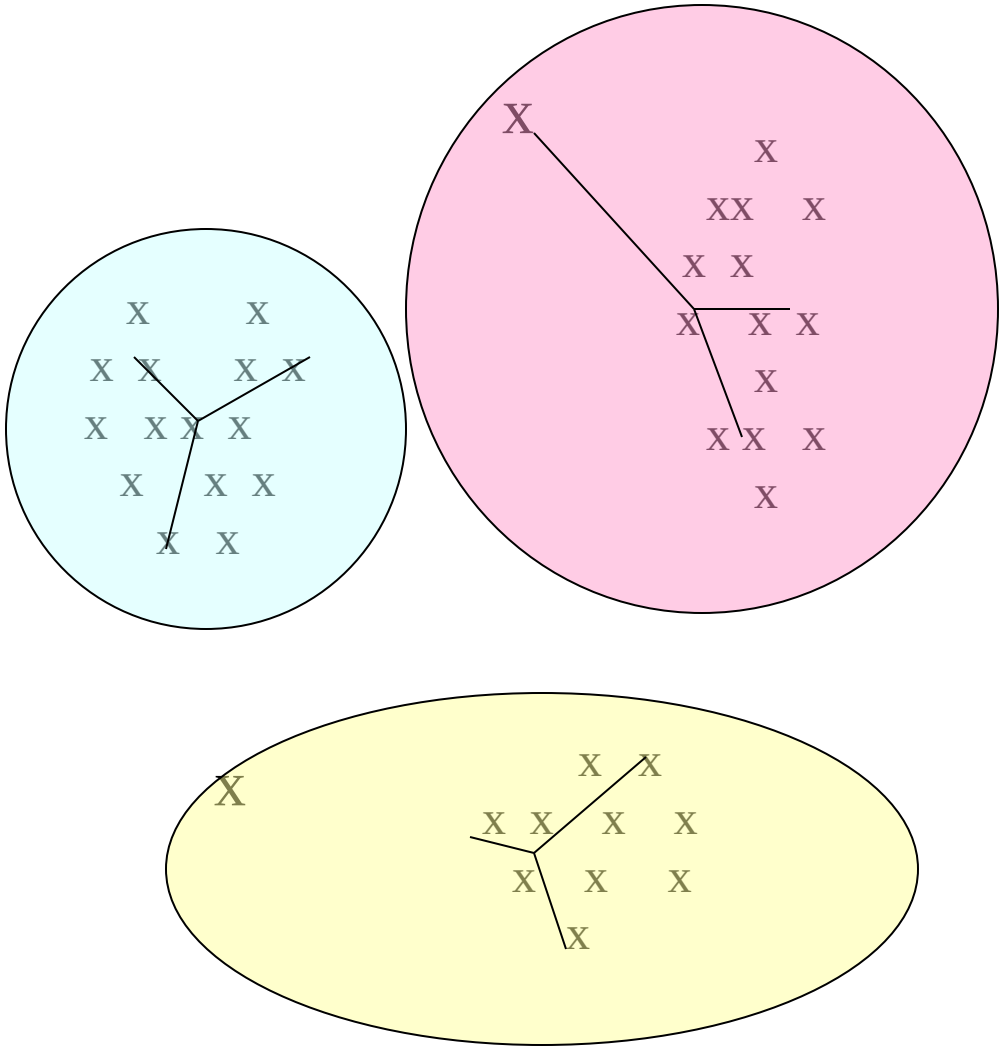
Example: Picking *k*

Too few;
many long
distances
to centroid.



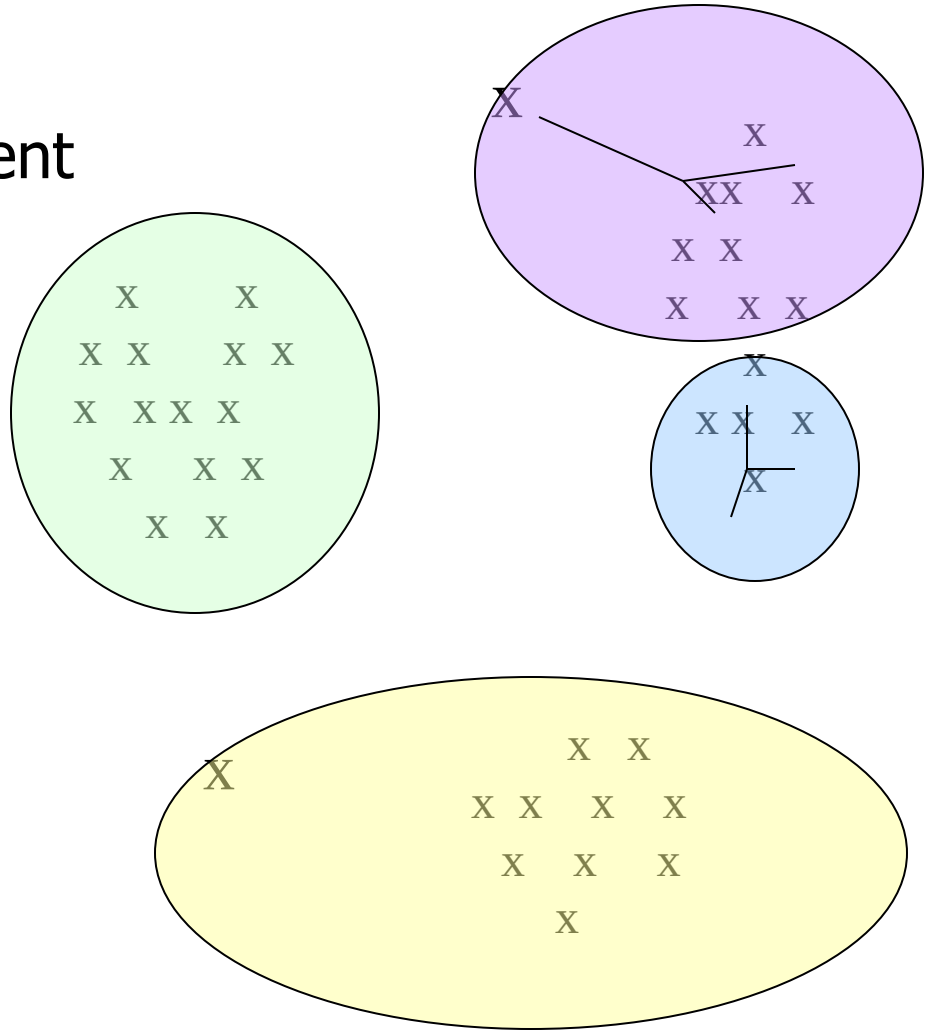
Example: Picking *k*

Just right; distances rather short.



Example: Picking *k*

Too many;
little improvement
in average
distance.



Issues with K-Means

- K needs to be predetermined
  - domain knowledge, heuristics …
- Assumes euclidean space
- Only spherical clusters
- Hard assignments of data points to clusters
  - Small shift of a data point can flip it to a different cluster
  - **Solution:** replace hard clustering of K-means with soft probabilistic assignments (GMM)

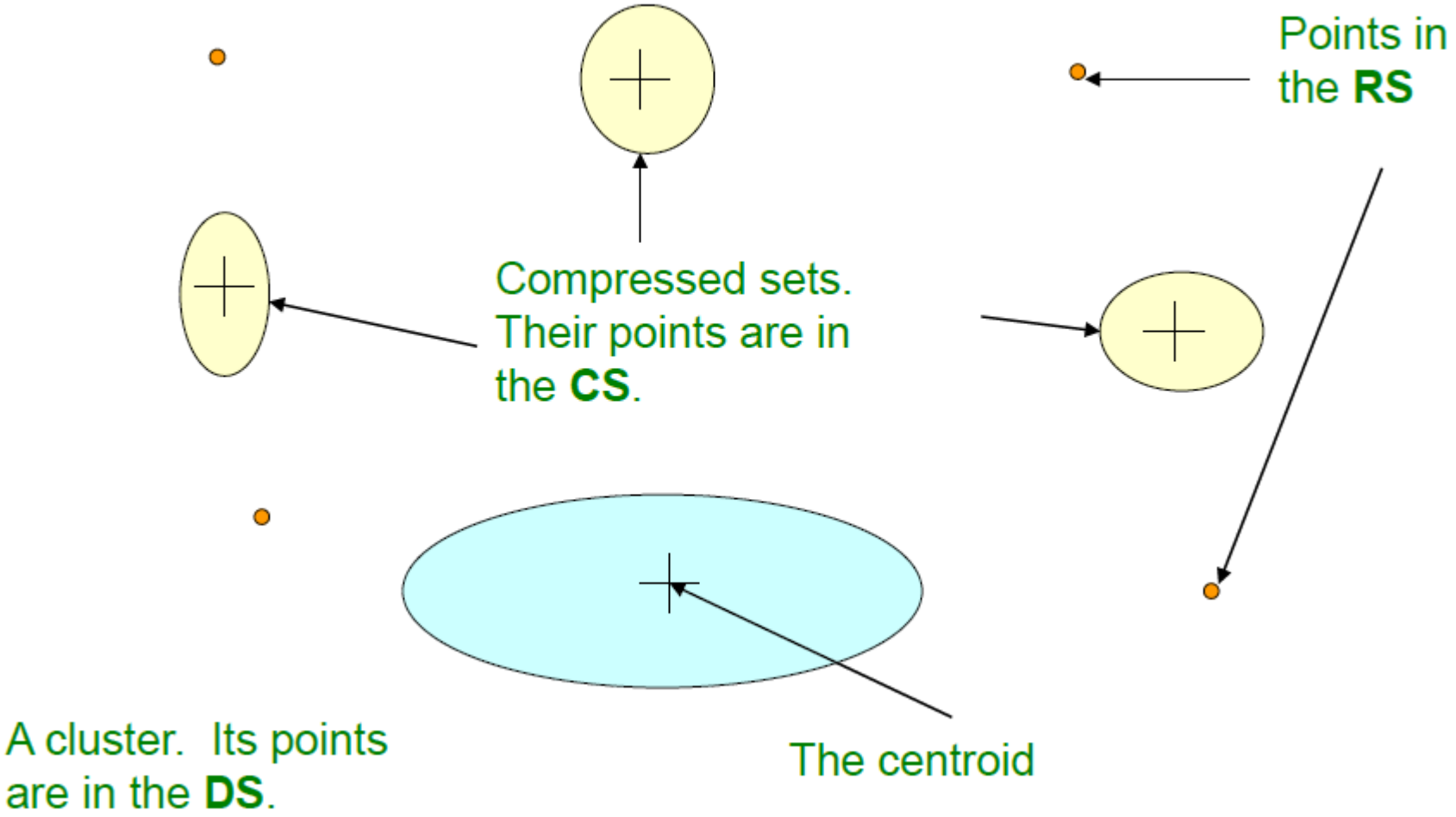
BFR Algorithm

- BFR (Bradley-Fayyad-Reina) is a variant of *k* -means designed to handle very large (disk-resident) data sets.
- It assumes that clusters are normally distributed around a centroid in a Euclidean space.
  - Standard deviations in different dimensions may vary.
- Points are read one main-memory-full at a time.
- Most points from previous memory loads are summarized by simple statistics.
- To begin, from the initial load we select the initial *k* centroids by some sensible approach.

Three Classes of Points

- *discard set (DS):*
  - points close enough to a centroid to be summarized.
- *compression set (CS):*
  - groups of points that are close together but not close to any centroid. They are summarized, but not assigned to a cluster.
- *retained set (RS):*
  - isolated points.

The „Galaxies“ Picture



Summarizing Sets of Points

- For each cluster, the discard set 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 number of points.
  - d = number of dimensions.
- Averages in each dimension (centroid coordinates) can be calculated easily as SUM_i / N.
  - SUM_i = i^{th} component of SUM.
- Variance of a cluster's discard set in dimension i can be computed by:
  - $(SUMSQ_i / N) - (SUM_i / N)^2$
  - And the standard deviation is the square root of that.
- The same statistics can represent any compression set.

The "Memory-Load" of Points

Processing the "Memory-Load" of points:

- Find those points that are "sufficiently close" to a cluster centroid; add those points to that cluster and the **DS**.
- Use any main-memory clustering algorithm to cluster the remaining points and the old **RS**.
  - Clusters go to the **CS**; outlying points to the **RS**.
- 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.

A Few Details . . .

- How do we decide if a point is “close enough” to a cluster that we will add the point to that cluster?
- How do we decide whether two compressed sets deserve to be combined into one?

How Close is Close Enough?

- We need a way to decide whether to put a new point into a cluster.

- BFR suggest two ways:
  - The *Mahalanobis distance* is less than a threshold.
  - Low likelihood of the currently nearest centroid changing.

Mahalanobis Distance

- Normalized Euclidean distance from centroid.

- For point $(x_1,\ldots,x_k)$ and centroid $(c_1,\ldots,c_k)$:
  - Normalize in each dimension: $y_i = (x_i - c_i)/\sigma_i$
  - Take sum of the squares of the y_i's.
  - Take the square root:

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

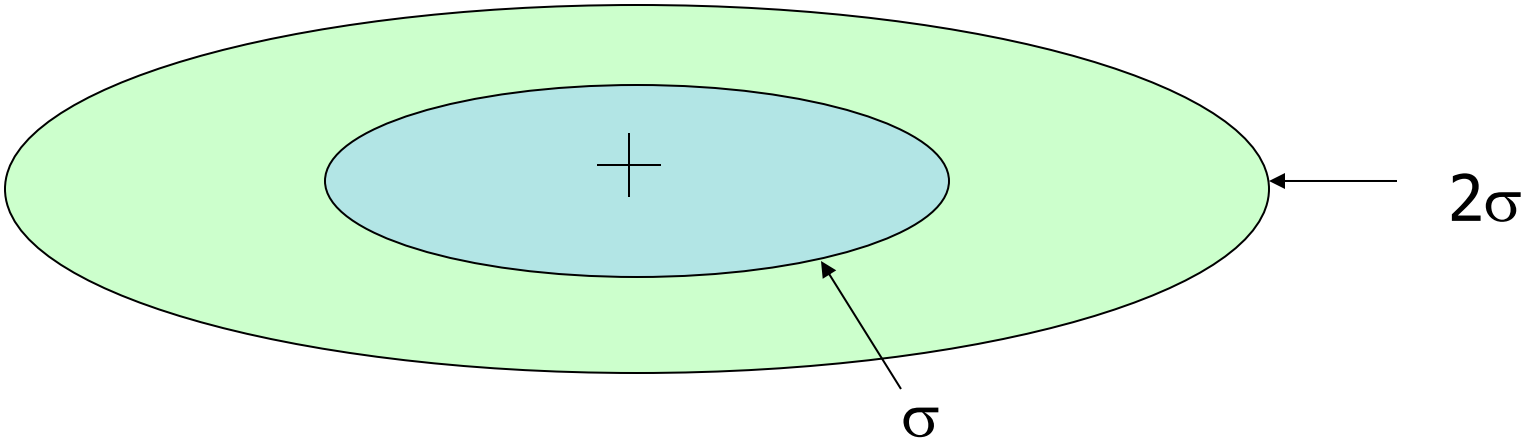
σ_i … standard deviation of points in the cluster in the i^{th} dimension

Mahalanobis Distance – (2)

- 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. 4 standard deviations.

Picture: Equal M.D. Regions



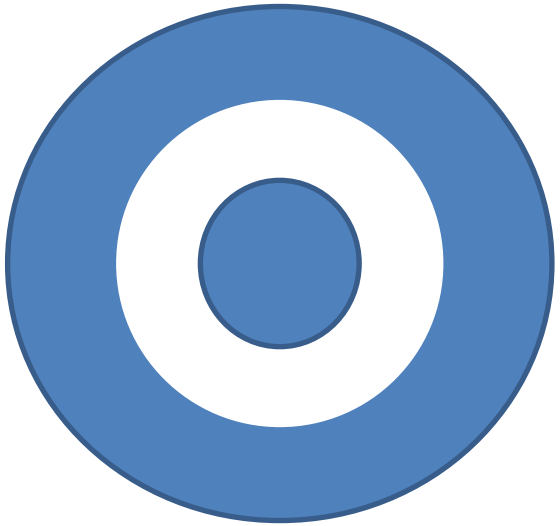
Should Two CS Subclusters Be Combined?

- Compute the variance of the combined subcluster.
  - *N*, SUM, and SUMSQ allow us to make that calculation quickly.
- Combine if the variance is below some threshold.
- Many alternatives: treat dimensions differently, consider density.

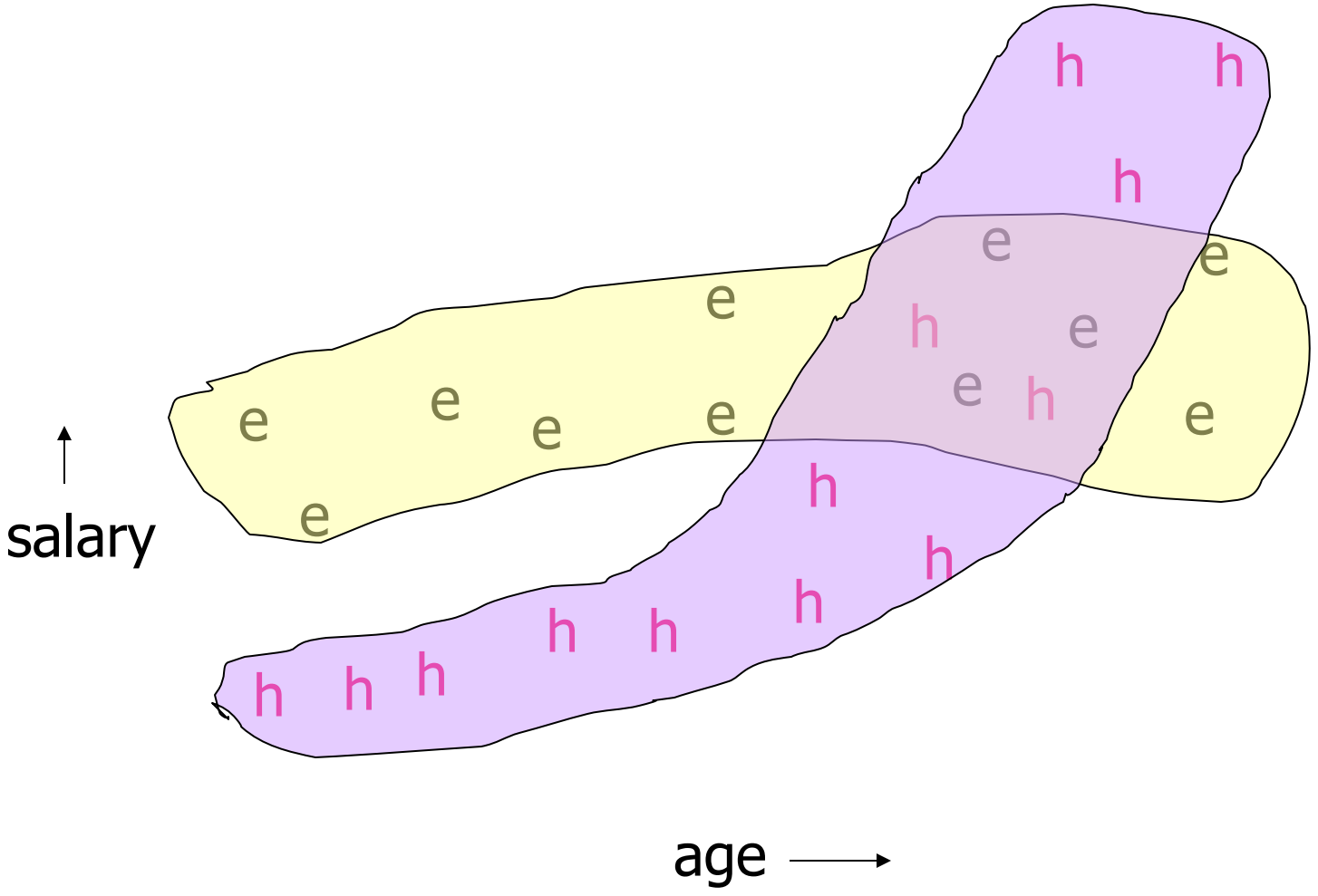
The CURE Algorithm

- Problem with BFR/*k* -means:
  - Assumes clusters are normally distributed in each dimension.
  - And axes are fixed – ellipses at an angle are *not* OK.

- CURE:
  - Assumes a Euclidean distance.
  - Allows clusters to assume any shape.



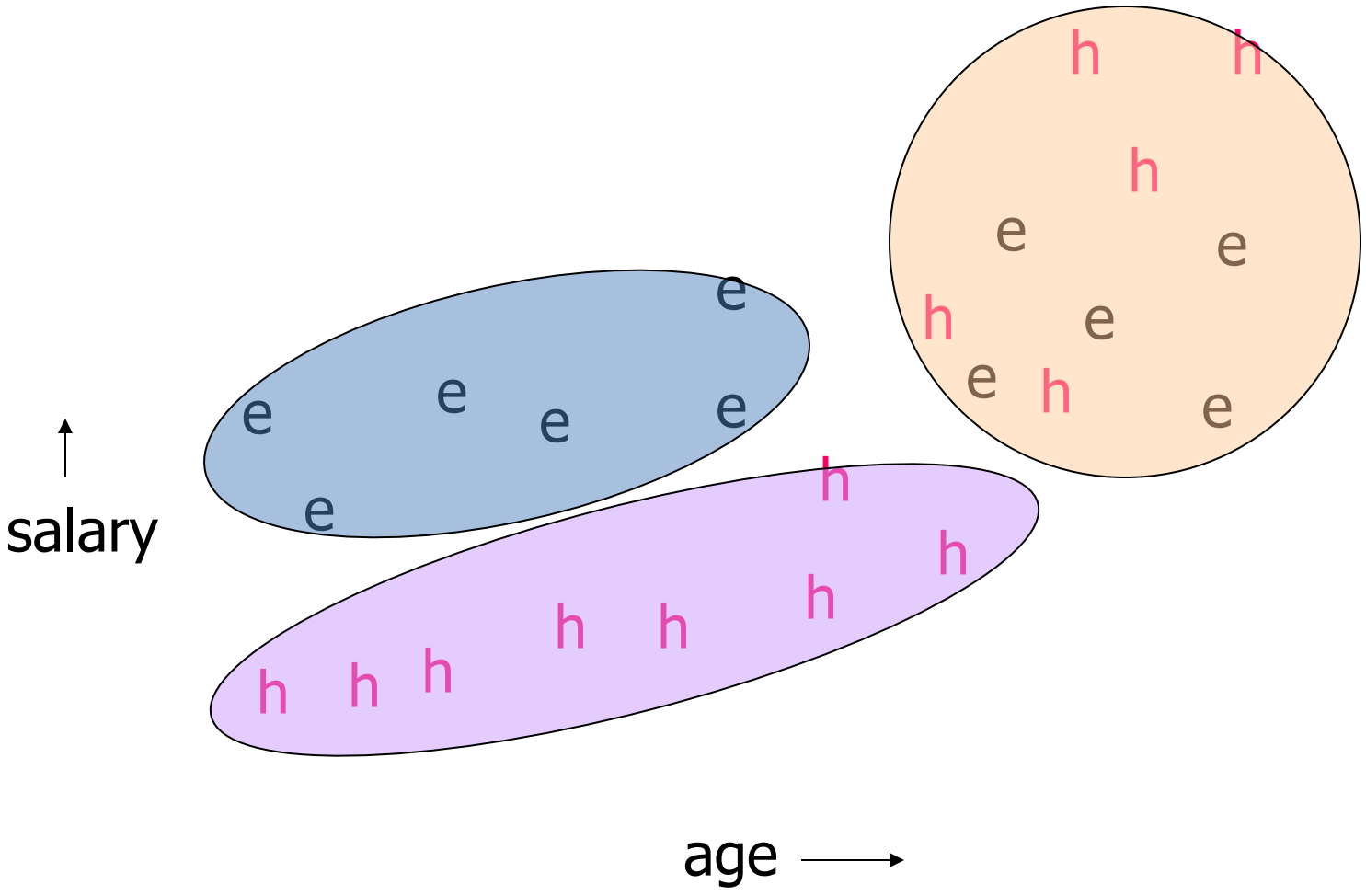
Example: Stanford Faculty Salaries



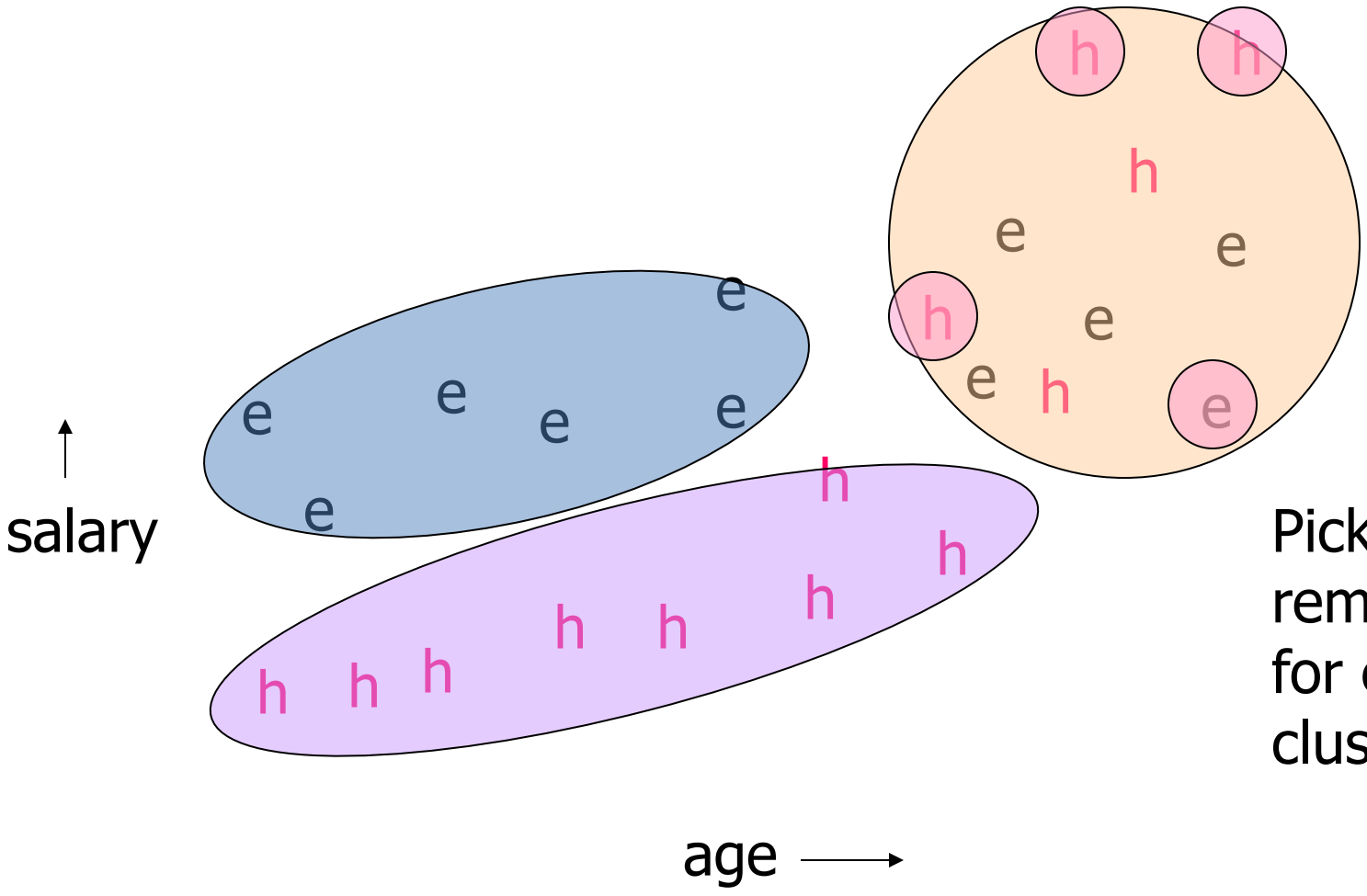
Starting CURE

- Pick a random sample of points that fit in main memory.
- Cluster these points hierarchically – group nearest points/clusters.
- For each cluster, pick a sample of points, as dispersed as possible.
- From the sample, pick representatives by moving them (say) 20% toward the centroid of the cluster.

Example: Initial Clusters

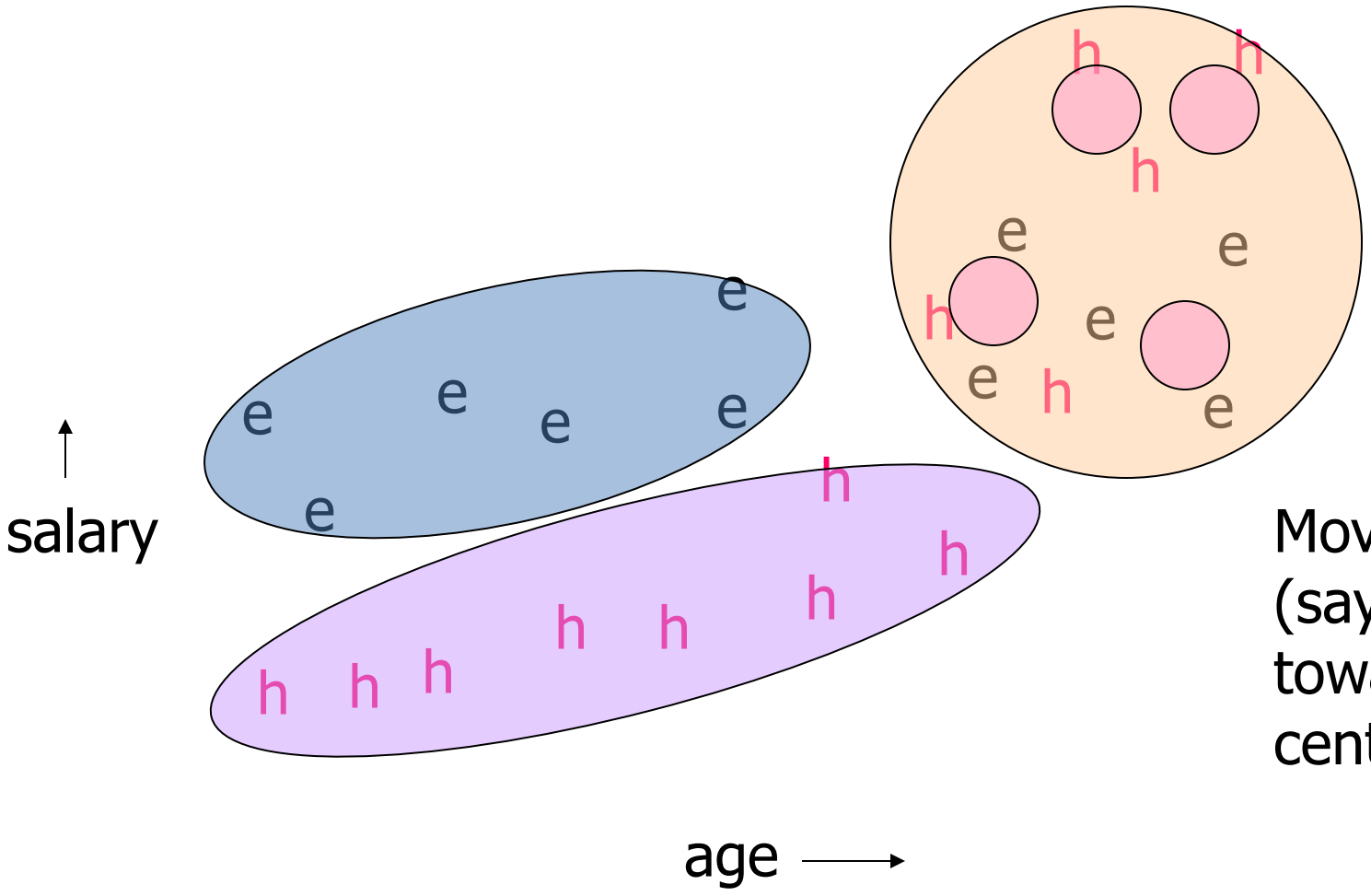


Example: Pick Dispersed Points



Pick (say) 4 remote points for each cluster.

Example: Pick Dispersed Points



Move points (say) 20% toward the centroid.

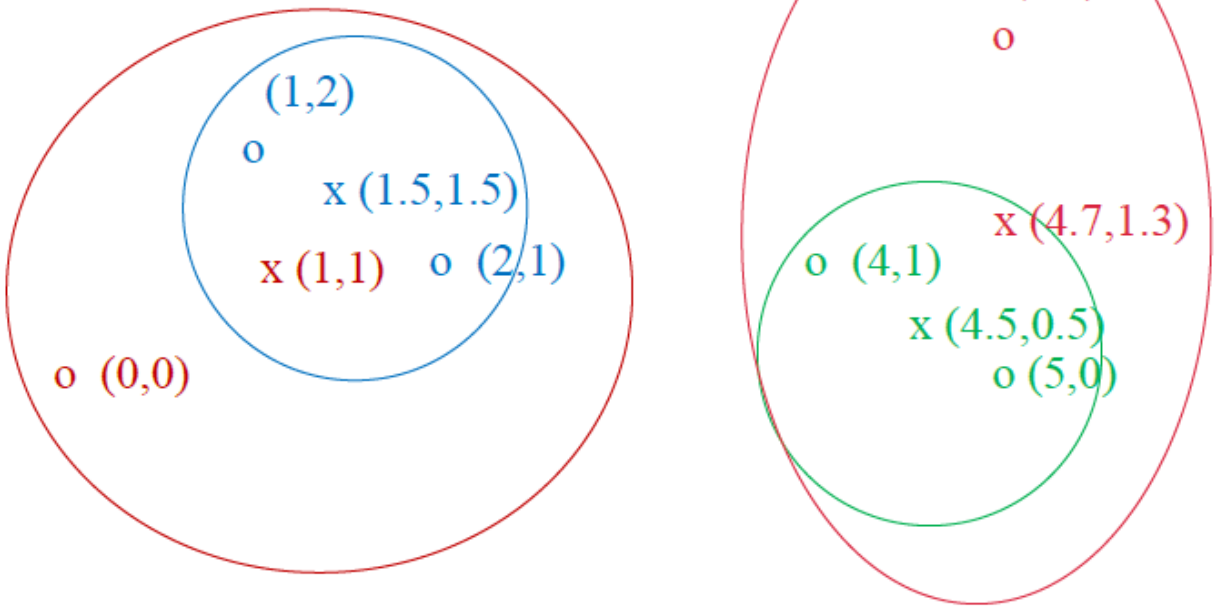
Finishing CURE

- Now, visit each point *p* in the data set.
- Place it in the "closest cluster."
  - Normal definition of "closest": that cluster with the closest (to *p*) among all the sample points of all the clusters.

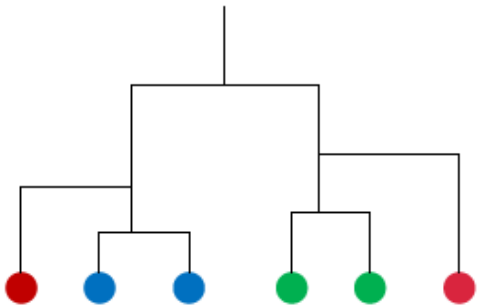
Hierarchical Clustering

- Key operation: Repeatedly combine two nearest clusters
- Three important questions:
  - How do you represent a cluster of more than one point?
  - How do you determine the "nearness" of clusters?
  - When to stop combining clusters?
- Key problem: as you build 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 points.
  - Measure intercluster distances by distances of centroids.

Example: Hierarchical Clustering



Data:
o … datapoint
x … centroid



Dendrogram

And in 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: *clustroid* = point "closest" to other points.
  - Treat clustroid as if it were centroid, when computing intercluster distances.

  - E.g.: using edit distance, we decide to merge the strings **abcd** and **aecdb**

  - edit distance = 3

  - there is no string that represents their averages

“Closest” Point?

- Possible meanings of “closest”:
  - Smallest maximum distance to the 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:

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

Defining „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 1: Use the *diameter* of the merged cluster = maximum distance between points in the cluster.
- Approach 2: Use the average distance between points in the cluster.
- Approach 3: Use a density-based approach: take the diameter or average distance, e.g., and divide by the number of points in the cluster.
  - Perhaps raise the number of points to a power first, e.g., square-root.

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

BIRCH

- Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)
- designed for clustering a large amount of numerical data
- integrates hierarchical clustering and other clustering methods
- overcomes difficulties in aglomaertive clustering
  - Scalability
  - Inability to undu what was done (clustered) before

* Zhang, Tian, Raghu Ramakrishnan, and Miron Livny. "BIRCH: an efficient data clustering method for very large databases." *ACM Sigmod Record*. Vol. 25. No. 2. ACM, 1996.

BIRCH (II)

- BIRCH is based on the notion of a *Clustering Feature* $\boldsymbol{Cf_i}$

$$CF = \langle n, LS, SS \rangle$$

$$LS = \sum_{i=1}^{n} x_i \qquad SS = \sum_{i=1}^{n} x_i^2$$

- many useful statistics can be computed based on this feature:
  - centroid
  - radius
  - diameter
  - …

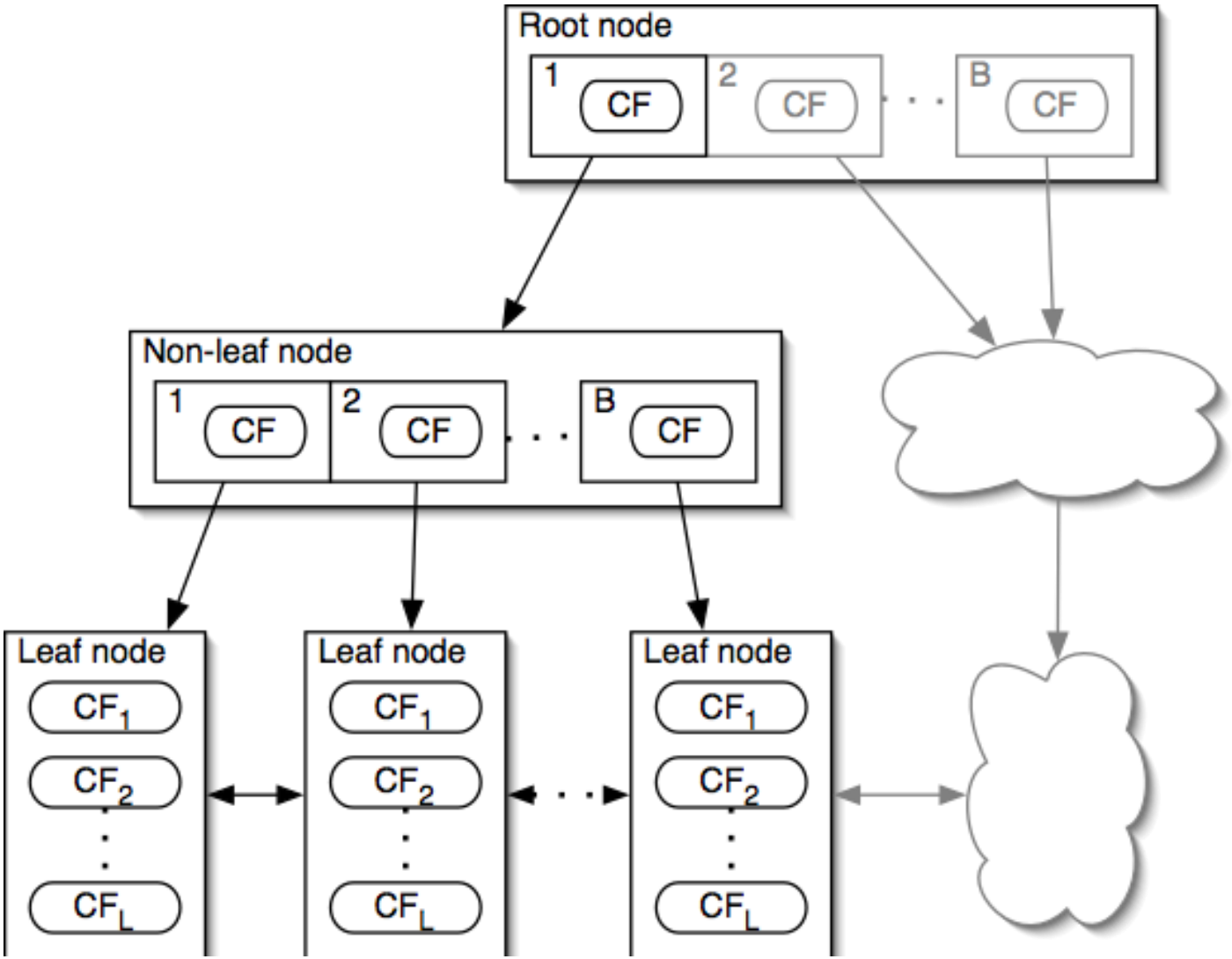
- Additive: $CF_1 + CF_2 = \langle n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2 \rangle$

BIRCH

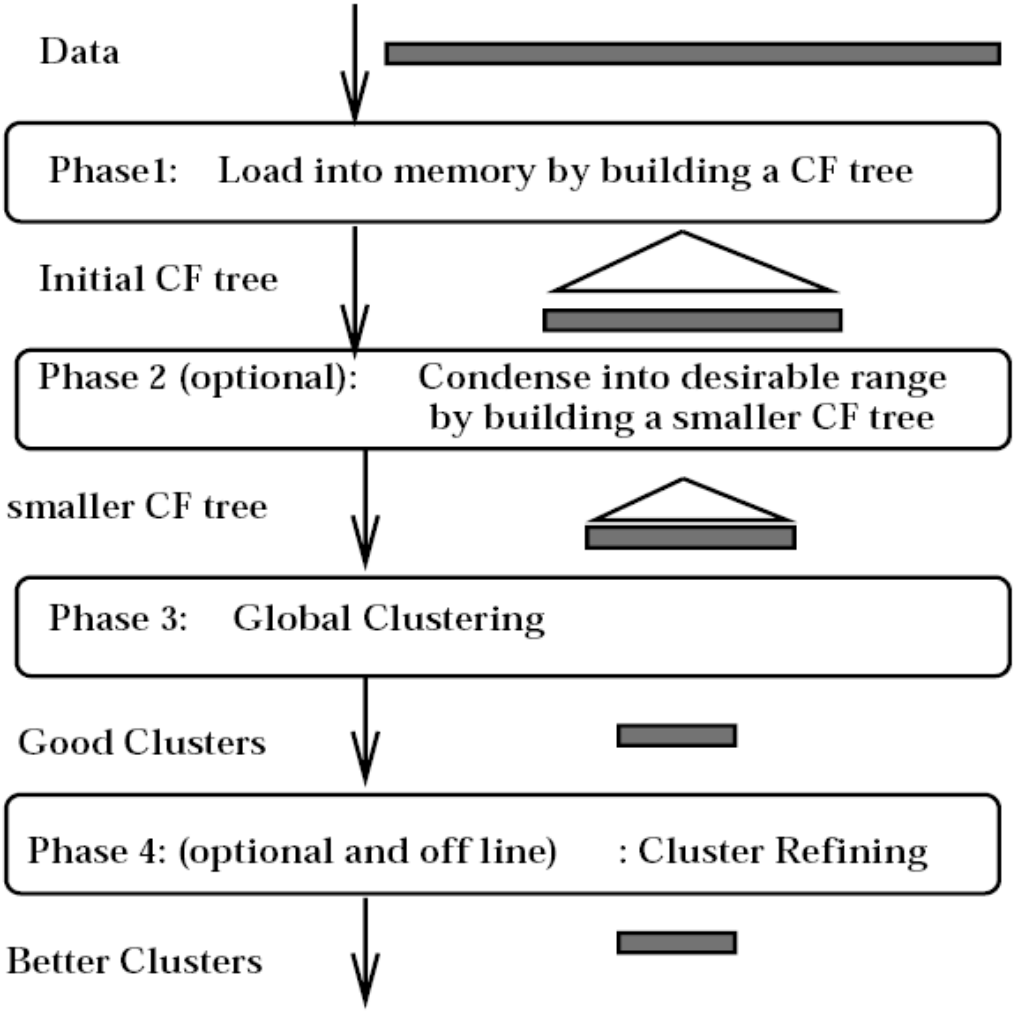
- **CF Tree** – a height-balanced tree that stores the clustering features for a hierarchical clustering with a branching factor **B** and threshold **t**

BIRCH Clustering:

- **Phase 1:** pass through data set and built initial in-memory CF-Tree
  - object is inserted to the closes leaf (subcluster)
  - new object information is passed to root
  - if diameter of leaf > threshold after insertion → split
- **Phase 2:** apply a seleced clustering algorithm to the cluster leaf nodes of the CF tree (groups sparse clusters into larger ones (outliers))
  - no need to re-read all objects!


Root node
1
CF
2
CF
B
CF
Non-leaf node
1
CF
2
CF
B
CF
Leaf node
CF1
CF2
CFL
Leaf node
CF1
CF2
CFL
Leaf node
CF1
CF2
CFL

BIRCH overview



Density Based Clustering

DBSCAN: Density-Based Clustering based on connected regions with high density

→ can find clusters of abitrary shapes

- locates regions of high density that are separated from one another by regions of low density.
  - Density = number of points within a specified radius (Eps)
- a point is a **core point** if it has more than a specified number of points (MinPts) within Eps
  - These are points that are at the interior of a cluster
- a **border point** has fewer than MinPts within Eps, but is in the neighborhood of a core point

* Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." *Kdd*. Vol. 96. No. 34. 1996.

DBSCAN

- A **noise point** is any point that is not a core point or a border point.
- Any two core points are close enough– within a distance Eps of one another – are put in the same cluster
- Any border point that is close enough to a core point is put in the same cluster as the core point
- Noise points are discarded

DBSCAN Algorithm

mark all objects as unvisited
do
  randomly select an unvisited object **p**
    mark p as **visited**
    if the ε-neighborhood of **p** has at least *MinPts* objects
      create new cluste C, and add **p** to C
      let N be the set of objects in the ε-Neighborhood of **p**
      for each point **p`** in N:
        if **p`** is *unvisited*
          mark **p`** as *visited*
          if the ε neighborhood of **p`** has at least *MinPts*
          add those points to N
        if **p`** is not yet a member of any cluster -> add **p`** to C
      end for
      output C
    else mark **p** as *noise*
  until no object is unread

Evaluation

- extrinsic methods (compare the clustering against the group truth)
  - cluster homogeneity (purity)
  - cluster completeness (
  - …
- intrinsic methods
  - evaluate goodness of a clustering by considering how well clusters are separated
  - e.g. silhouette coefficient

Summary

- Clustering
  - Clusters are often a useful summary of data that is in the form of points in some space. To cluster points, we need a **distance measure** on that space. Ideally, points in the same cluster have small distances between them, while points in different clusters have large distances between them.
- The Curse of Dimensionality
  - Points in high-dimensional Euclidean spaces, as well as points in non-Euclidean spaces often behave unintuitively. Two unexpected properties of these spaces are that random points are almost always at about the same distance, and random vectors are almost always orthogonal.
- K-Means Algorithms:
  - This family of algorithms is of the point-assignment type and assumes a Euclidean space. It is assumed that there are exactly k clusters for some known k. After picking k initial cluster centroids, the points are considered one at a time and assigned to the closest centroid. The centroid of a cluster can migrate during point assignment, and an optional last step is to reassign all the points, while holding the centroids fixed at their final values obtained during the first pass.

Summary (II)

- The BFR Algorithm
  - A version of k-means designed to handle data that is too large to fit in main memory. It assumes clusters are normally distributed about the axes
- The CURE Algorithm
  - This algorithm is of the point-assignment type. It is designed for a Euclidean space, but clusters can have any shape. It handles data that is too large to fit in main memory.
- Clustering Using Map-Reduce
  - We can divide the data into chunks and cluster each chunk in parallel, using a Map task. The clusters from each Map task can be further clustered in a single Reduce task.

K-means in MapReduce

- How would you implement K-Means in MapReduce?
- Take a set of seed centroids
  - can be generated using other algorithms e.g. Canopy Clustering
- Compute distance to centroids and determine the closest centroid for each data point in a Mapper
- Combine data points in similar clusters
- Recompute new centroids in reduce task

Map

Algorithm 1. map (key, value)

Input: centroids, the offset key, the sample value

Output: <key', value'> pair, where the key' is the index of the closest center point and value' is a string comprise of sample information

1. Construct the sample *instance* from *value*;
2. *minDis = Double.MAX VALUE*;
3. *index* = -1;
4. For i=0 to *centers*.length do
   dis= ComputeDist(*instance*, *centers*[*i*]);
   If *dis < minDis {*
   minDis = dis;
   index = i;
   }
5. End For
6. Take *index* as *key'*;
7. Construct *value'* as a string comprise of the values of different dimensions;
8. output <*key, value*> pair;

Combine

Algorithm 2. combine (*key*, *V*)
Input: *key* is the index of the cluster, *V* is the list of the samples assigned to the same cluster
Output: < *key, value*> pair, where the *key*' is the index of the cluster, *value*' is a string comprised of sum of the samples in the same cluster and the sample number

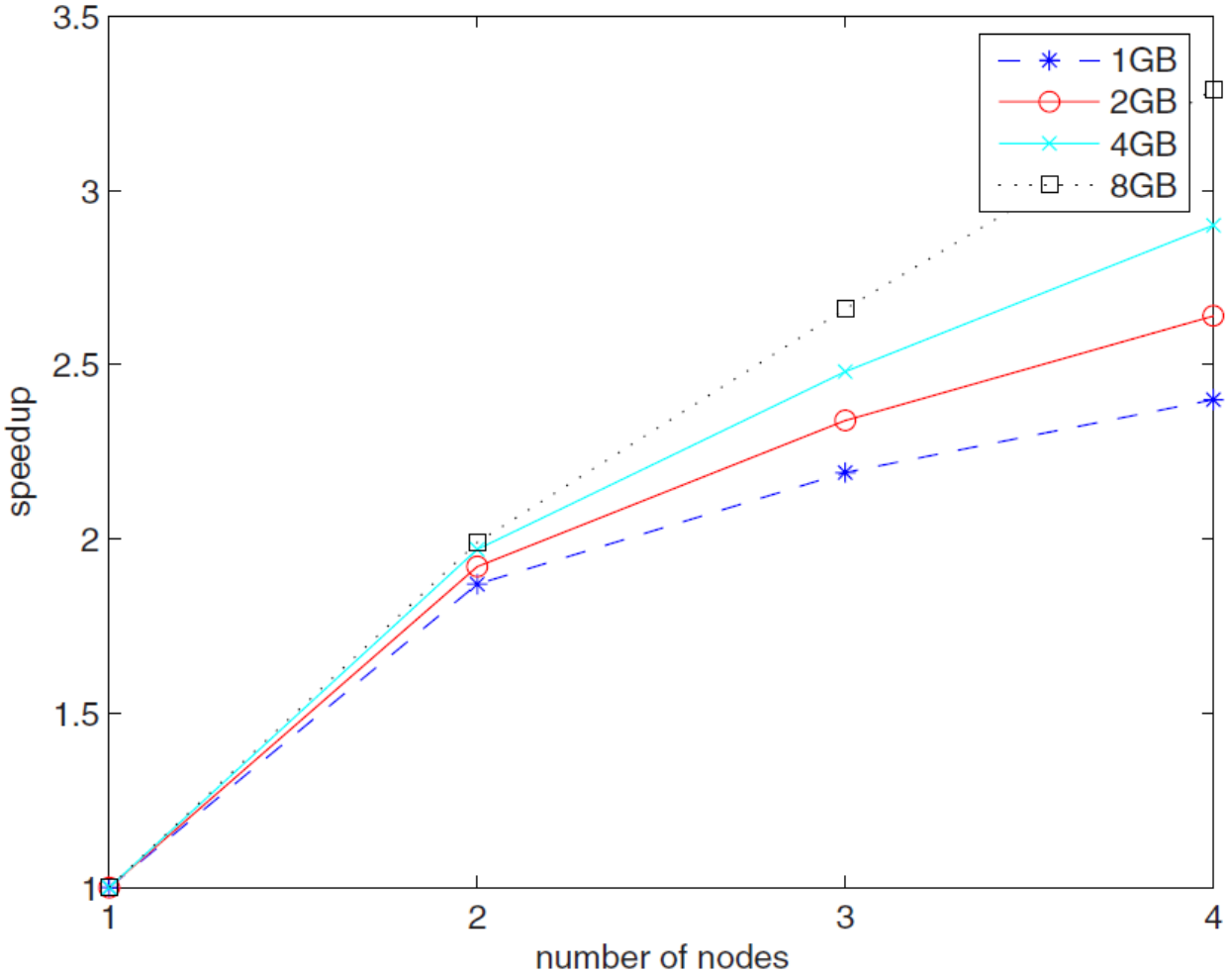
1. Initialize one array to record the sum of value of each dimensions of the samples contained in the same cluster, i.e. the samples in the list *V*;
2. Initialize a counter *num* as 0 to record the sum of sample number in the same cluster;
3. while(*V*.hasNext()){
   Construct the sample *instance* from *V*.next();
   Add the values of different dimensions of *instance* to the array
   num++;
4. }
5. Take *key* as *key*';
6. Construct *value*' as a string comprised of the sum values of different dimensions and *num*;
7. output < *key, value*> pair;

Reduce

Algorithm 3. reduce (*key*, *V*)
Input: *key* is the index of the cluster, *V* is the list of the partial sums from different host
Output: <*key, value*> pair, where the *key*' is the index of the cluster, *value*' is a string representing the new center

1. Initialize one array record the sum of value of each dimensions of the samples contained in the same cluster, e.g. the samples in the list *V*;
2. Initialize a counter *NUM* as 0 to record the sum of sample number in the same cluster;
3. while(*V*.hasNext()){
   Construct the sample *instance* from *V*.next();
   Add the values of different dimensions of *instance* to the array
   NUM += *num*;
   }
4. Divide the entries of the array by *NUM* to get the new center's coordinates;
5. Take *key* as *key*';
6. Construct *value*' as a string comprise of the *center*'s coordinates;
7. output <*key, value*> pair;

SpeedUp of parallel k-means



- keep the dataset constant and increase the number of nodes

Initialization: *k* -Means

- **Possible initialization strategies of the *k* cluster centers:**
  - 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.
  - (Canopy Clustering)

Canopy Clustering

- very simple and fast method for grouping objects into clusters
- uses a fast approximate distance metric and two distance thresholds T1 > T2 for processing.

Algorithm:

- begin with a set of points and remove one at random.
- Create a Canopy containing this point and iterate through the remainder of the point set.
- At each point, if its distance from the first point is < T1, then add the point to the cluster.
- If, in addition, the distance is < T2, then remove the point from the set.

In MapReduce:

- The data is massaged into suitable input format
- Each mapper performs canopy clustering on the points in its input set and outputs its canopies' centers
- The reducer clusters the canopy centers to produce the final canopy centers
- The points are then clustered into these final canopies