AIM3 – Scalable Data Analysis and Data Mining

05 – Clustering

Sebastian Schelter, Christoph Boden, Volker Markl

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

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

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

Agenda

- Topics of the course
  - Motivation, Overview
  - MapReduce & Distributed filesystems
  - MapReduce: Joins, Patterns & Extensions
  - Stratosphere
  - Clustering
  - Dimensionality Reduction
  - Data Stream Mining
  - Graph Processing & Social Network Analysis
  - Graph Processing: Google Pregel
  - Collaborative Filtering: Neighborhood Methods
  - Collaborative Filtering: Latent Factor Models
  - Classification
  - Textmining
  - Specialized Machine Learning approaches

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, …

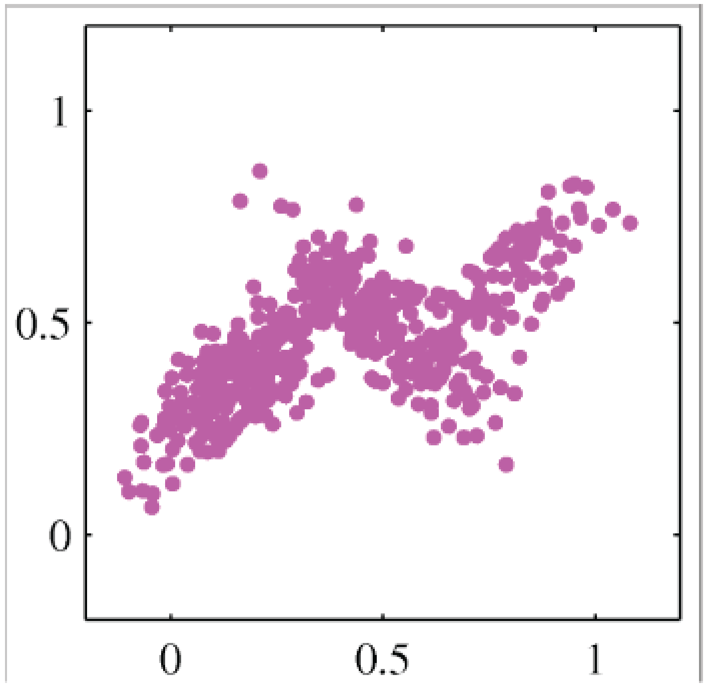
- Unsupervised learning: The data have no target attribute.
  - We want to explore the data to find some intrinsic structures in them.

Clustering: Problem definition

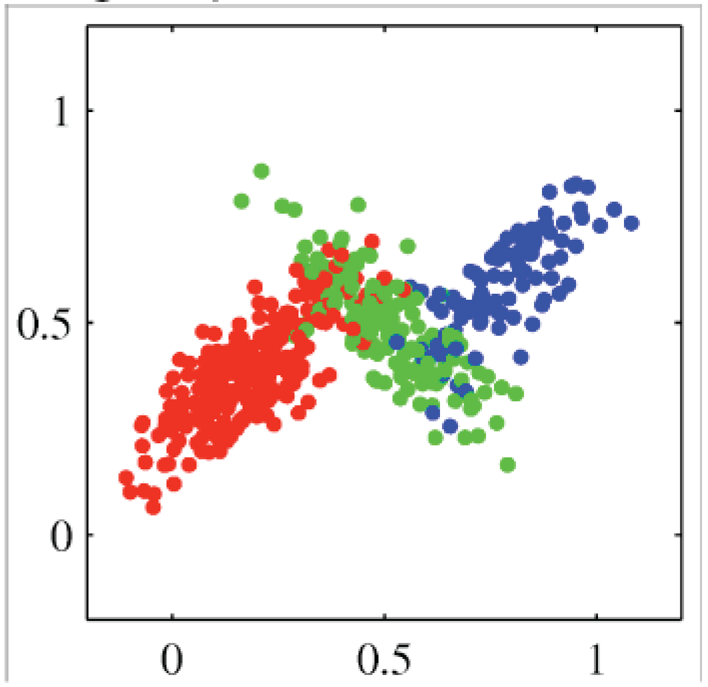
- Given a set of points, with a notion of distance between points, group the points into some number of clusters, so that
  - Members of a cluster are close/similar to each other
  - Members of different clusters are dissimilar

- Usually:
  - Points are in a high-dimensional space
  - Similarity is defined using a distance measure
  - Euclidean, Cosine, Jaccard, edit distance, …

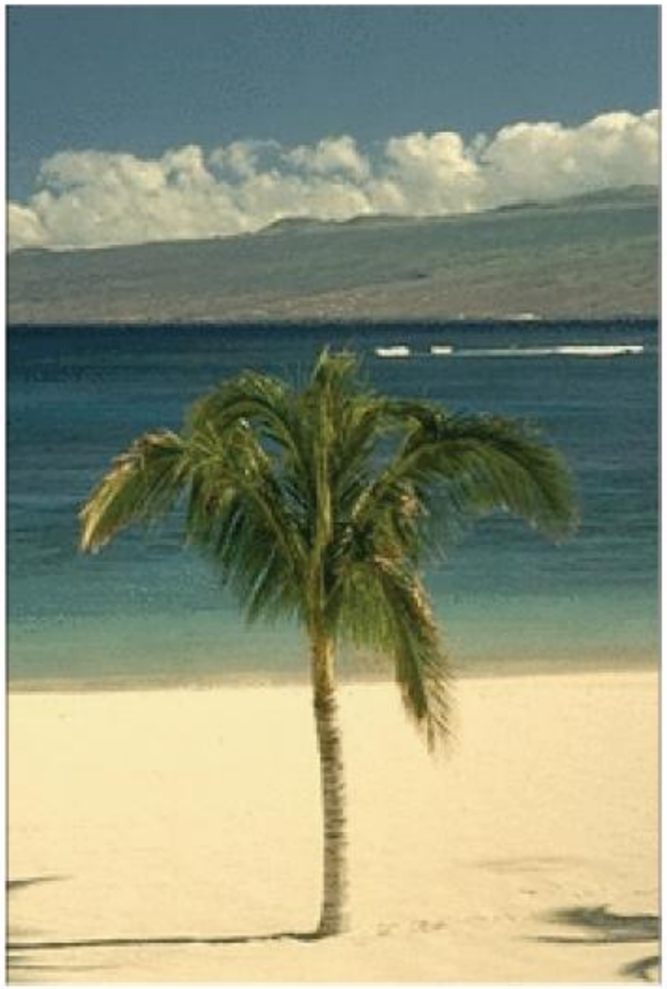
Example

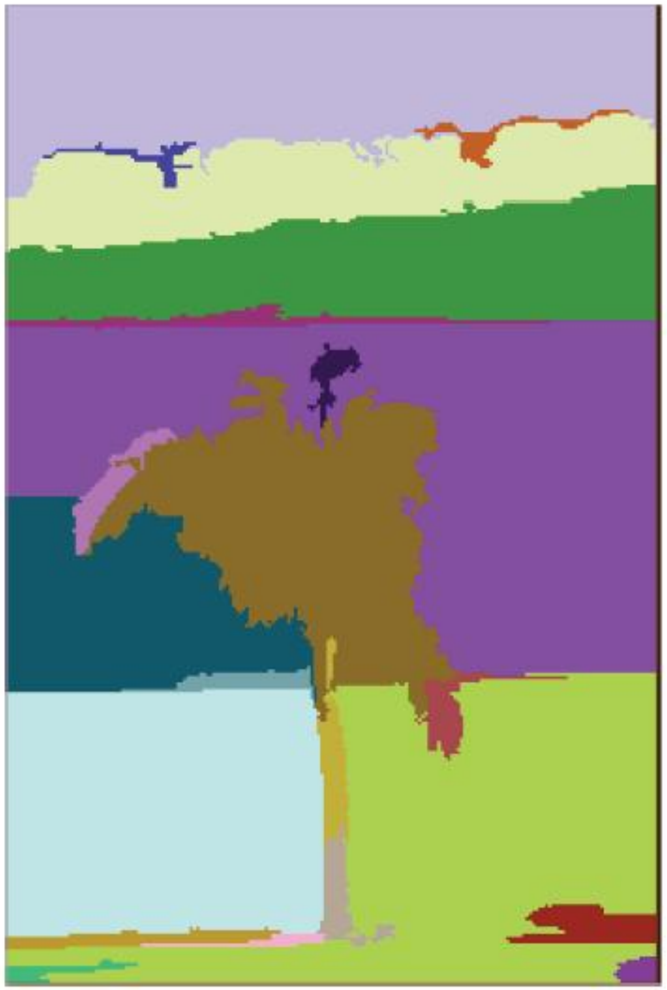


$\rightarrow$



Application: Image Segmentation





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

Problems With Clustering

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

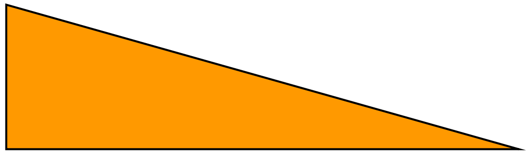
- **But:** Many applications involve not 2, but 10 or 10,000 dimensions.
- High-dimensional spaces look different: almost all pairs of points are at about the same distance.

Example: Curse of Dimensionality

- Assume random points within a bounding box
  - e.g., values between 0 and 1 in each dimension.

- In 2 dimensions
  - a variety of distances between 0 and 1.41.

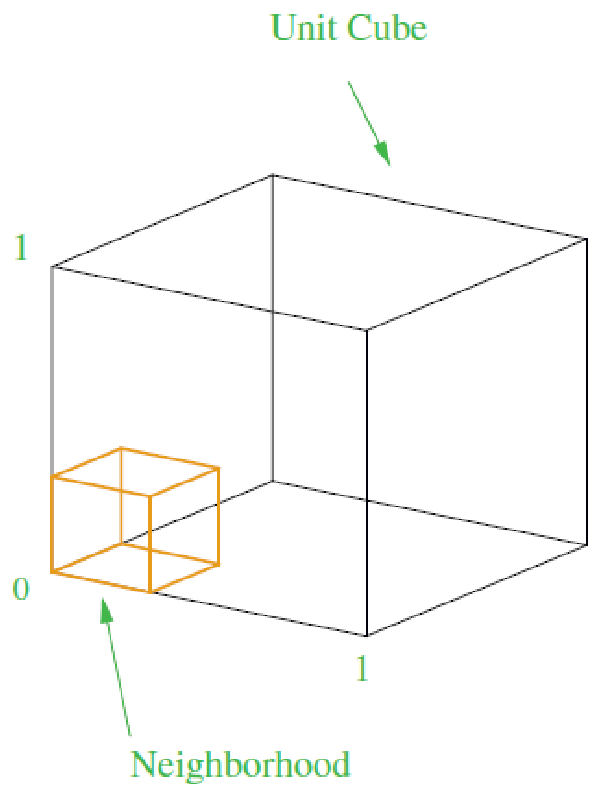
- In 10,000 dimensions
  - the difference in any one dimension is distributed as a triangle.



- The law of large numbers applies

- Actual distance between two random points is the sqrt of the sum of squares of essentially the same set of differences

The curse of dimensionality



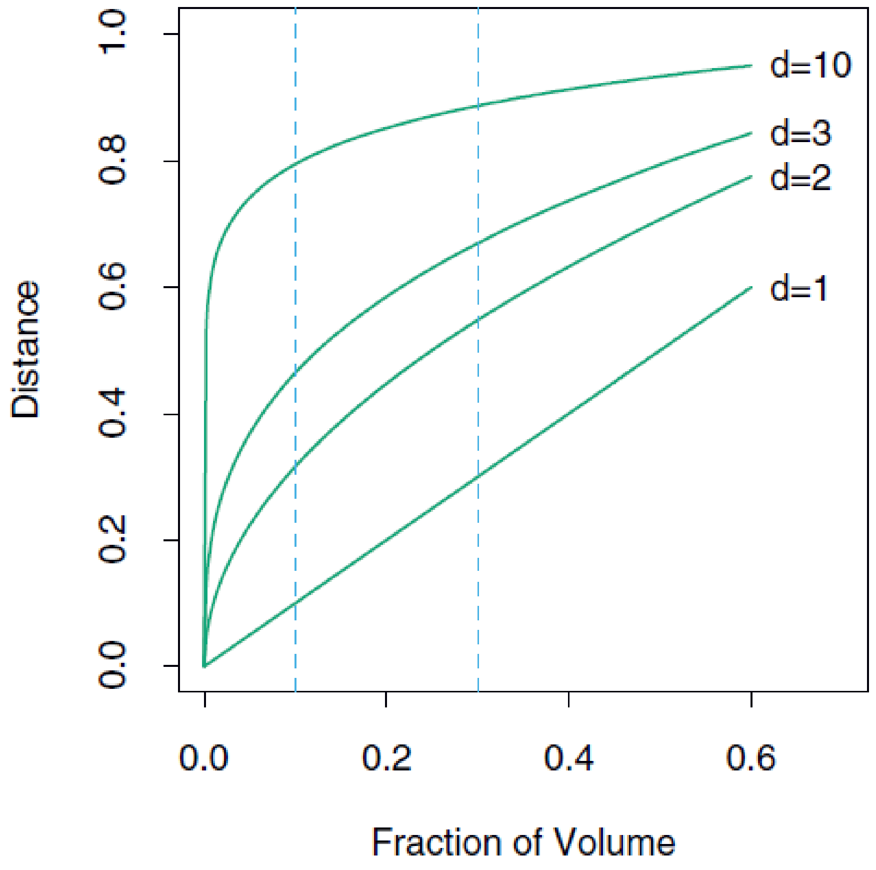


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover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

Example: SkyCat

- A catalog of 2 billion “sky objects” represents objects by their radiation in 7 dimensions (frequency bands).
- Problem: cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.
- Sloan Sky Survey is a newer, better version.

Example: Clustering CD’s

- Intuitively: music divides into categories, and customers prefer a few categories.
  - But what are categories really?

- Represent a CD by the customers who bought it.

- Similar CD’s have similar sets of customers, and vice-versa.

Example: Clustering CD‘s (II)

- Think of a space with one dimension for each customer.
  - Values in a dimension may be 0 or 1 only.

- A CD’s point in this space is $(x_1, x_2,\ldots, x_k)$, where $x_i = 1$ iff the i th customer bought the CD.
  - Compare with boolean matrix: rows = customers; cols. = CD’s.

- For Amazon, the dimension count is tens of millions.

- An alternative: use minhashing/LSH to get Jaccard similarity between “close” CD’s.

- 1 minus Jaccard similarity can serve as a (non-Euclidean) distance.

Example: Clusters of Documentsd

- 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.
    - 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.

Aside: Cosine, Jaccard, and Euclidean Distances

- As with CD's we have a choice when we think of documents as sets of words or shingles:
  - 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.

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

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)*

Example: DNA Sequences

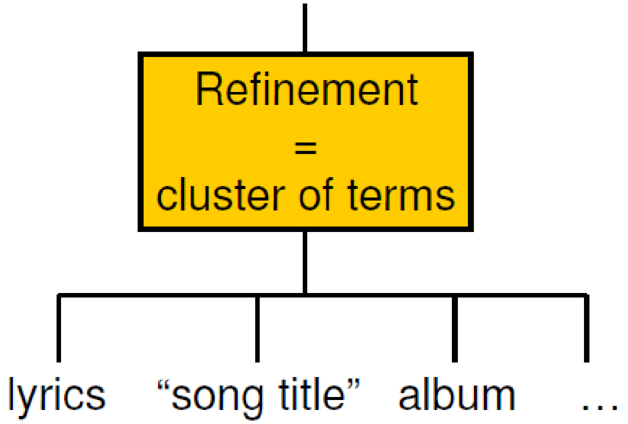
- Objects are sequences of {C,A,T,G}.
- Distance between sequences is *edit distance*, the minimum number of inserts and deletes needed to turn one into the other.
- Note there is a "distance," but no convenient space in which points "live."

In the Web Context

- Clustering, in itself, is often not the primary problem
  - Exploratory analysis is rarely needed

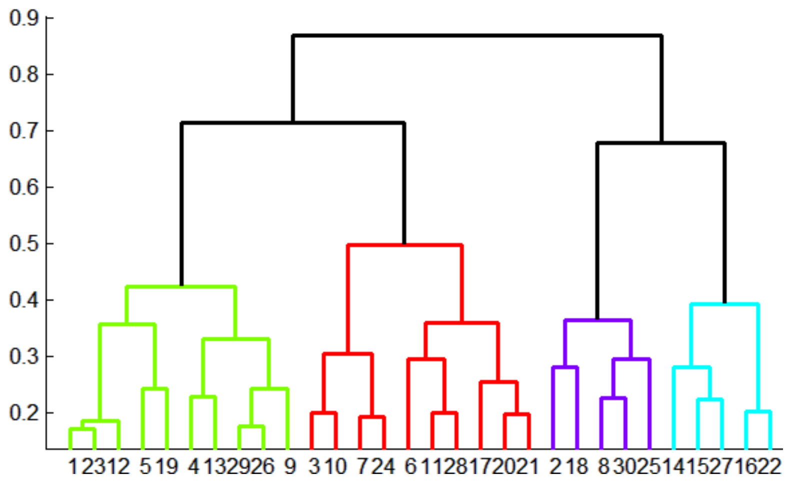
- Methods are often tied to the "final" goal

- E.g.: Query Refinement
  - User inputs ambiguous query ("*madonna*")
  - Search engine asks:
  - "*Did you mean: songs, videos, pictures?*"

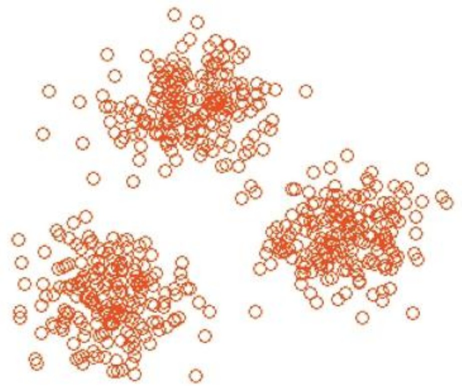


Methods of Clustering

- 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.



Hierarchical Clustering

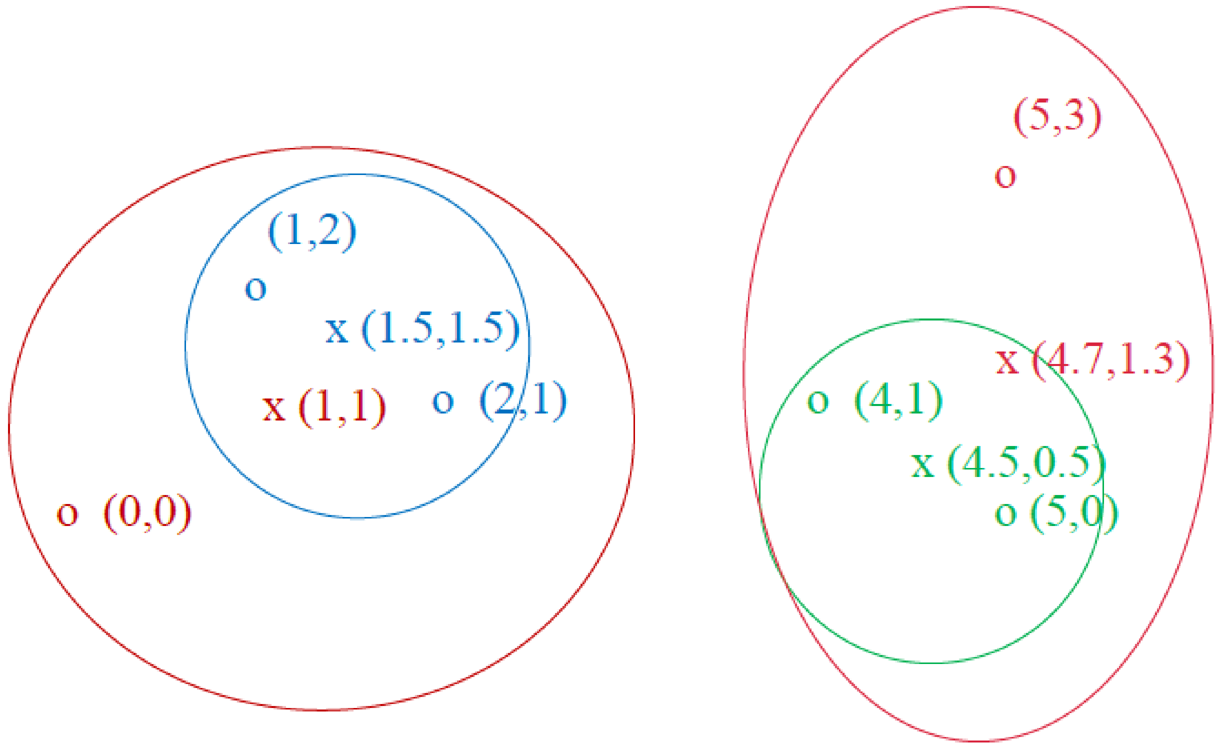
- Key operation: Repeatedly combine two nearest clusters

- 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?

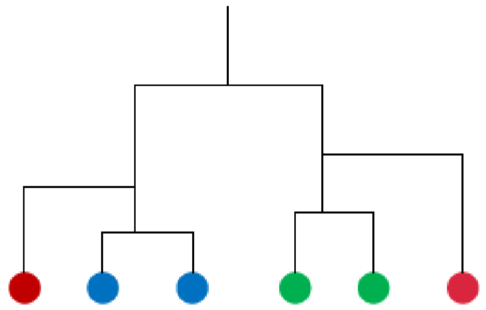
- Key problem: as you build clusters, how do you represent the location of each cluster, to tell which pair of clusters is closest?

- 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

k – Means Algorithm(s)

- Assumes Euclidean space.
- 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

- For each point, place it in the cluster whose current centroid it is nearest.
- After all points are assigned, fix the centroids of the *k* clusters.
- Optional: reassign all points to their closest centroid.
  - Sometimes moves points between clusters.

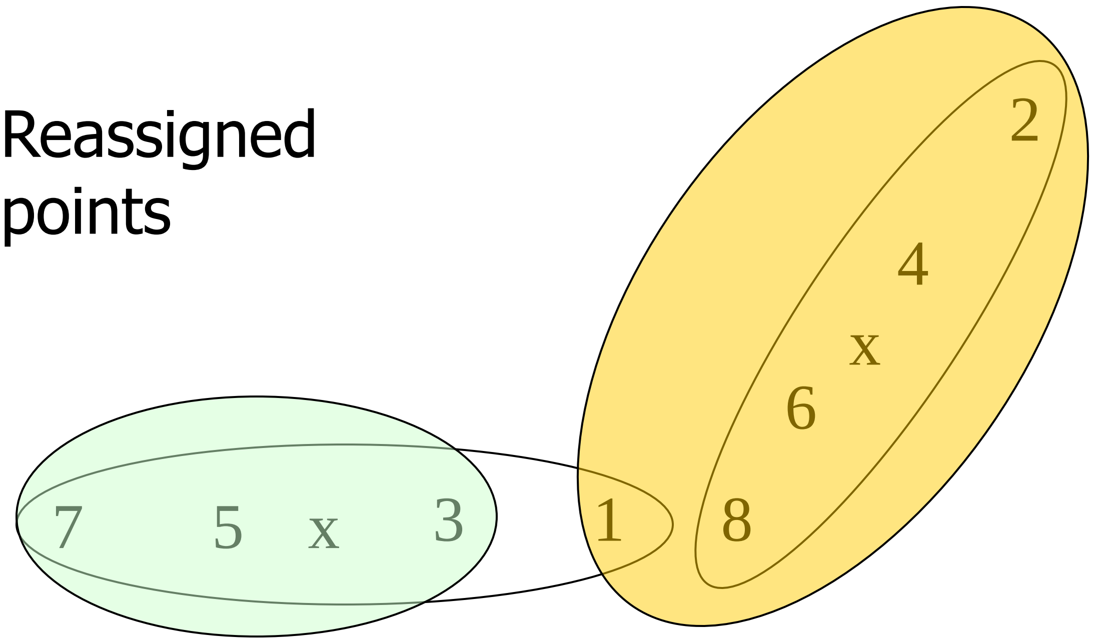
K-Means Algorithm

```
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
```

Example: Assigning Clusters

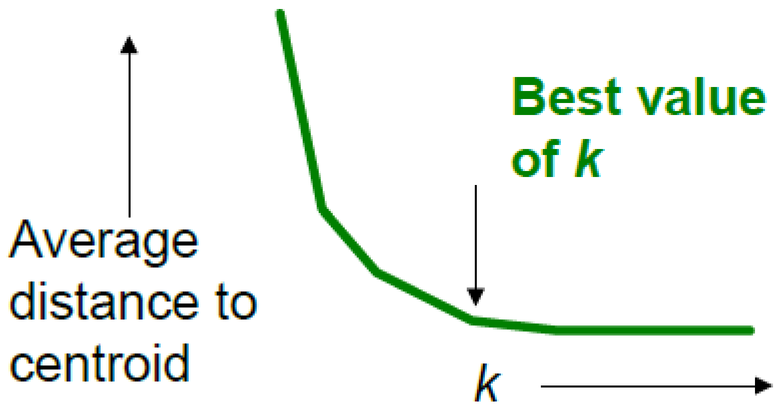
Reassigned
points



Clusters after first round

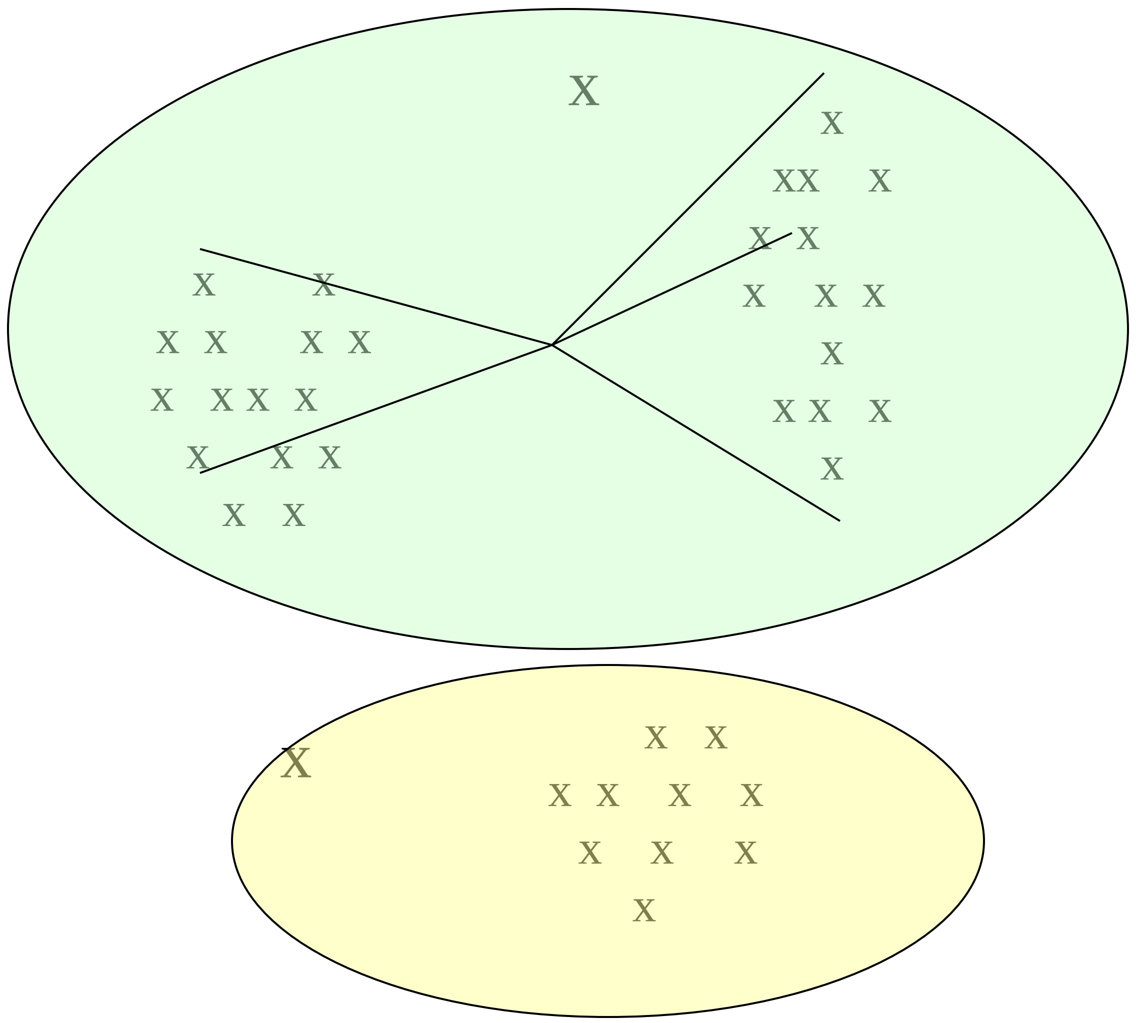
Getting the *k* right

- **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



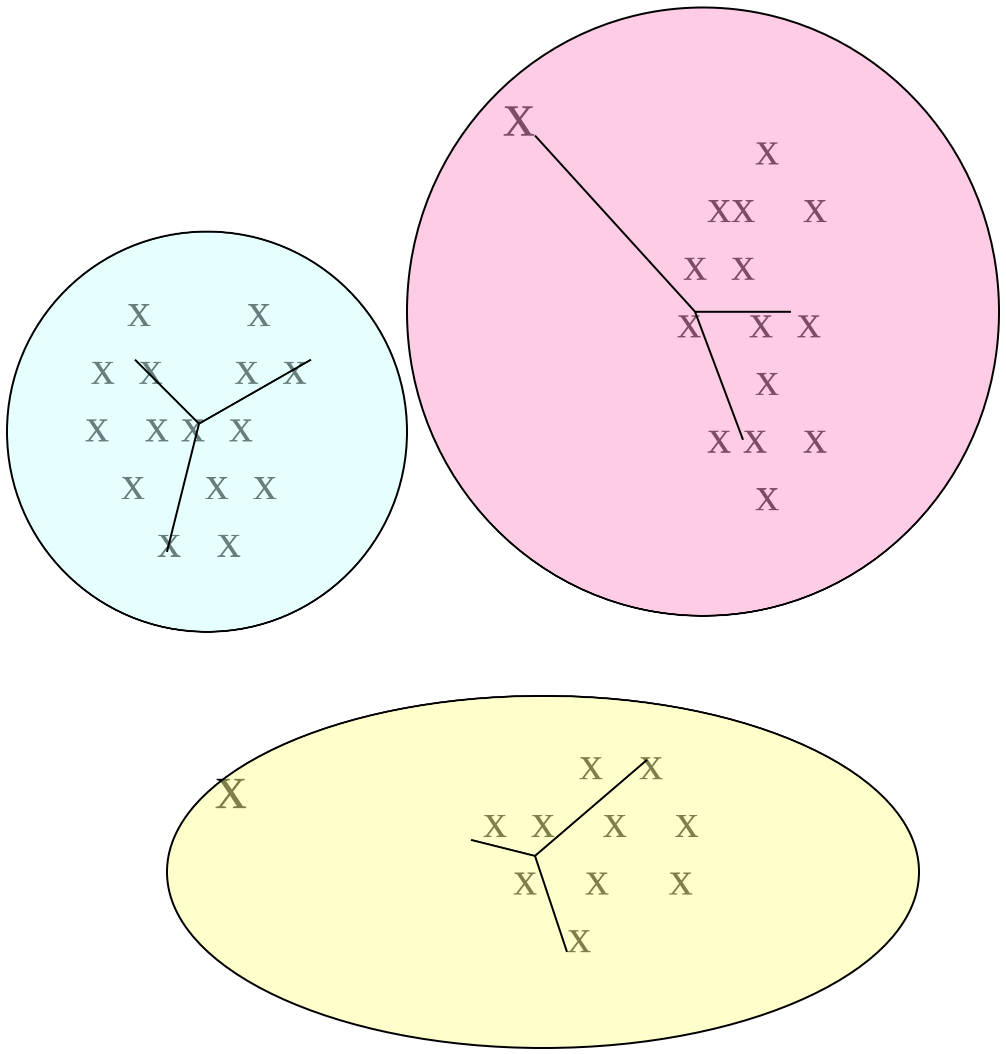
Example: Picking *k*

Too few;
many long
distances
to centroid.



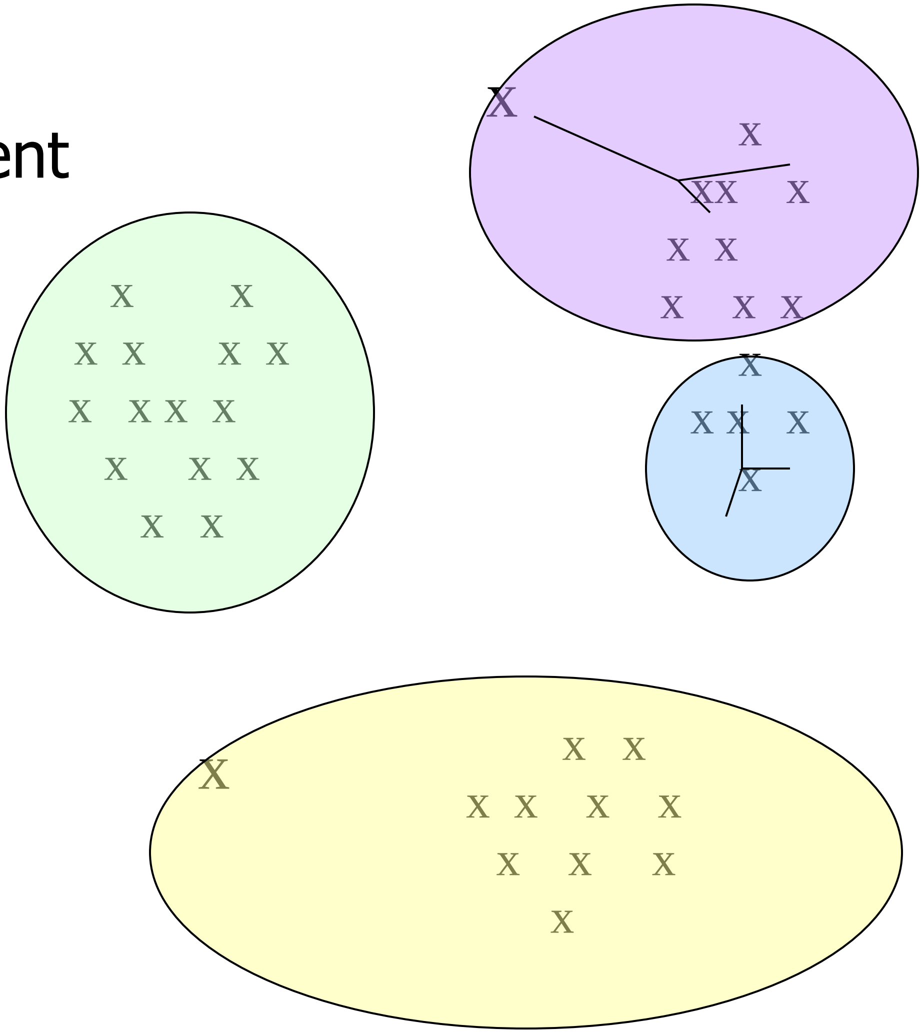
Example: Picking *k*

Just right;
distances
rather short.



Example: Picking *k*

Too many;
little improvement
in average
distance.



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

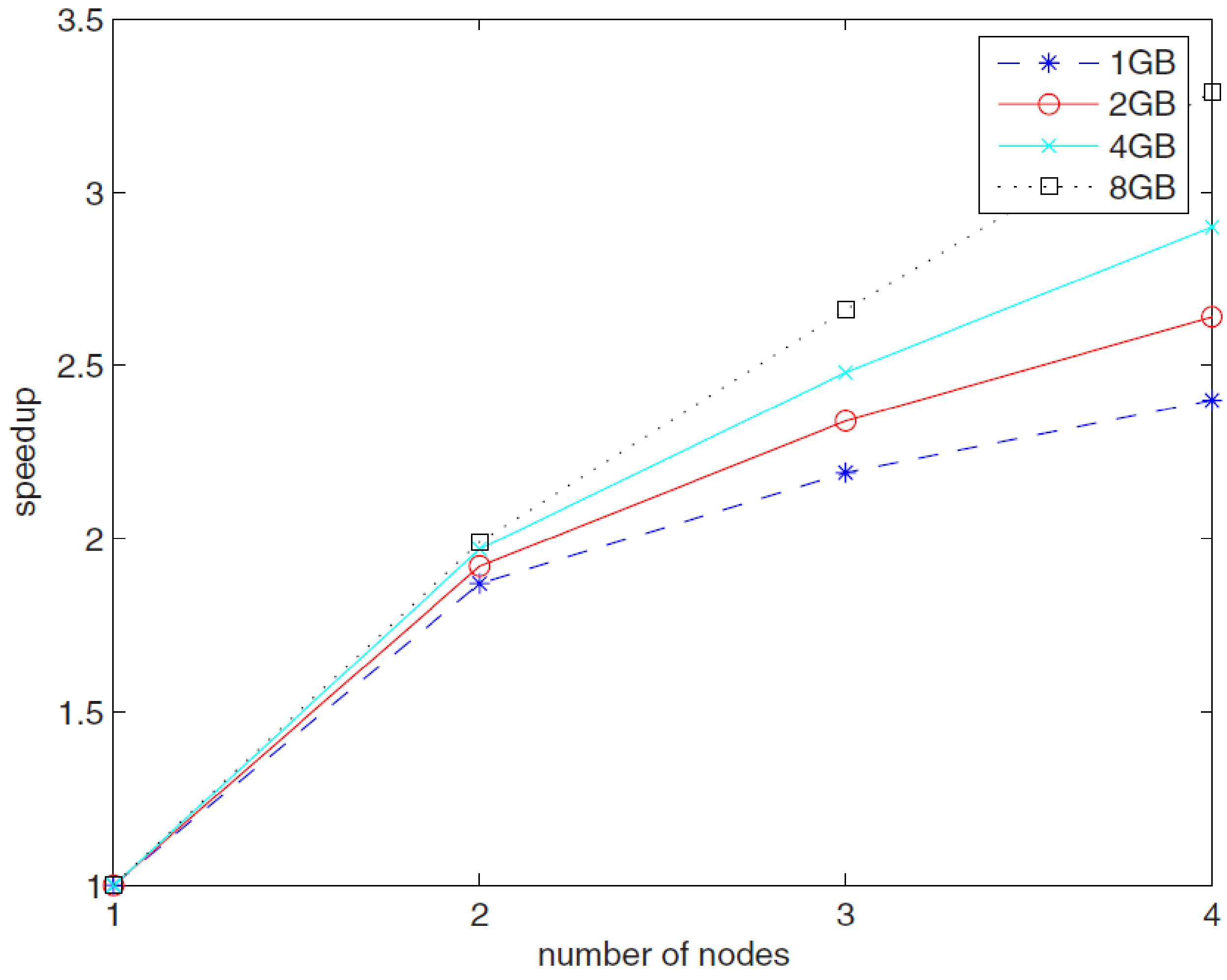
```
1. Initialize one array to record the sum of value of each dimensions of the samples
   contained in the same cluster, i.e. the samples in the list V;
2. Initialize a counter num as 0 to record the sum of sample number in the same
   cluster;
3. while(V.hasNext()){
       Construct the sample instance 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

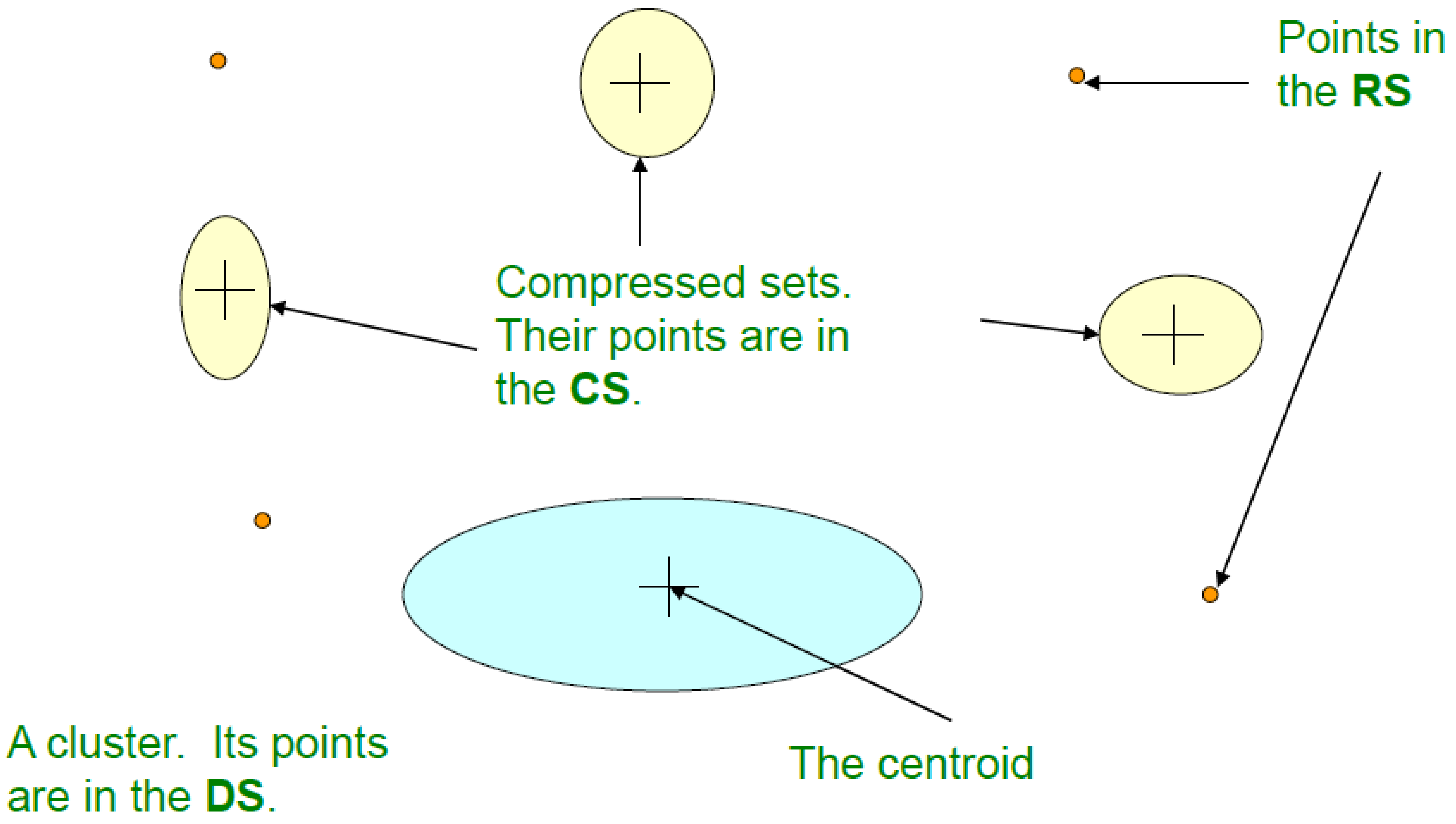
BFR Algorithm

- 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


Points in the **RS**
Compressed sets. Their points are in the **CS**.
A cluster. Its points are in the **DS**.
The centroid

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:
  - $(\text{SUMSQ}_i / N) - (\text{SUM}_i / N)^2$
  - 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:

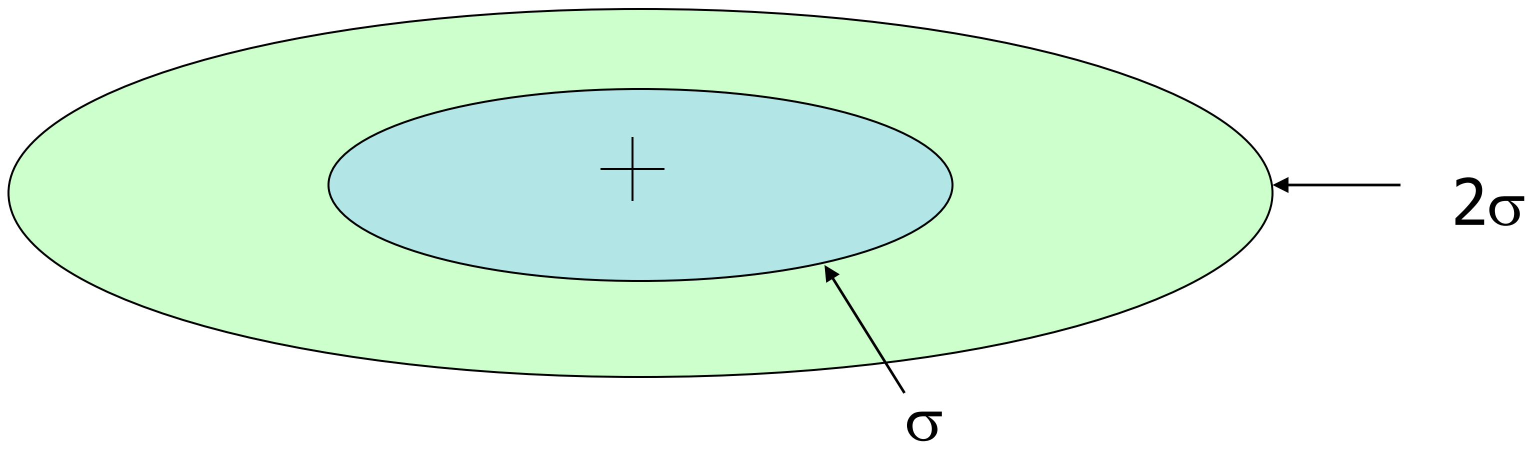
$$d(x,c) = \sqrt{\sum_{i}^{d} \left(\frac{x_i - c_i}{\sigma_i}\right)^2}$$

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

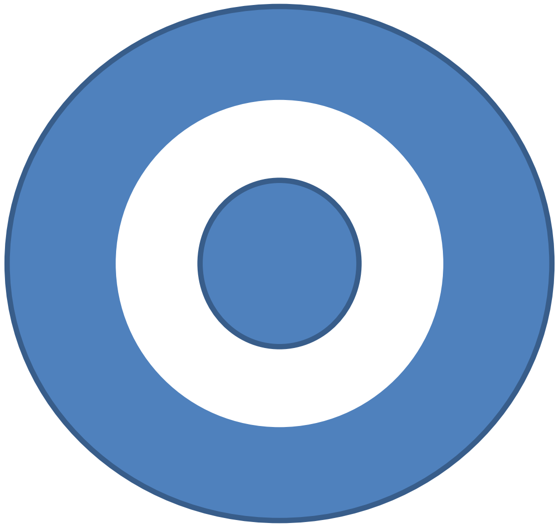


Should Two CS Subclusters Be Combined?

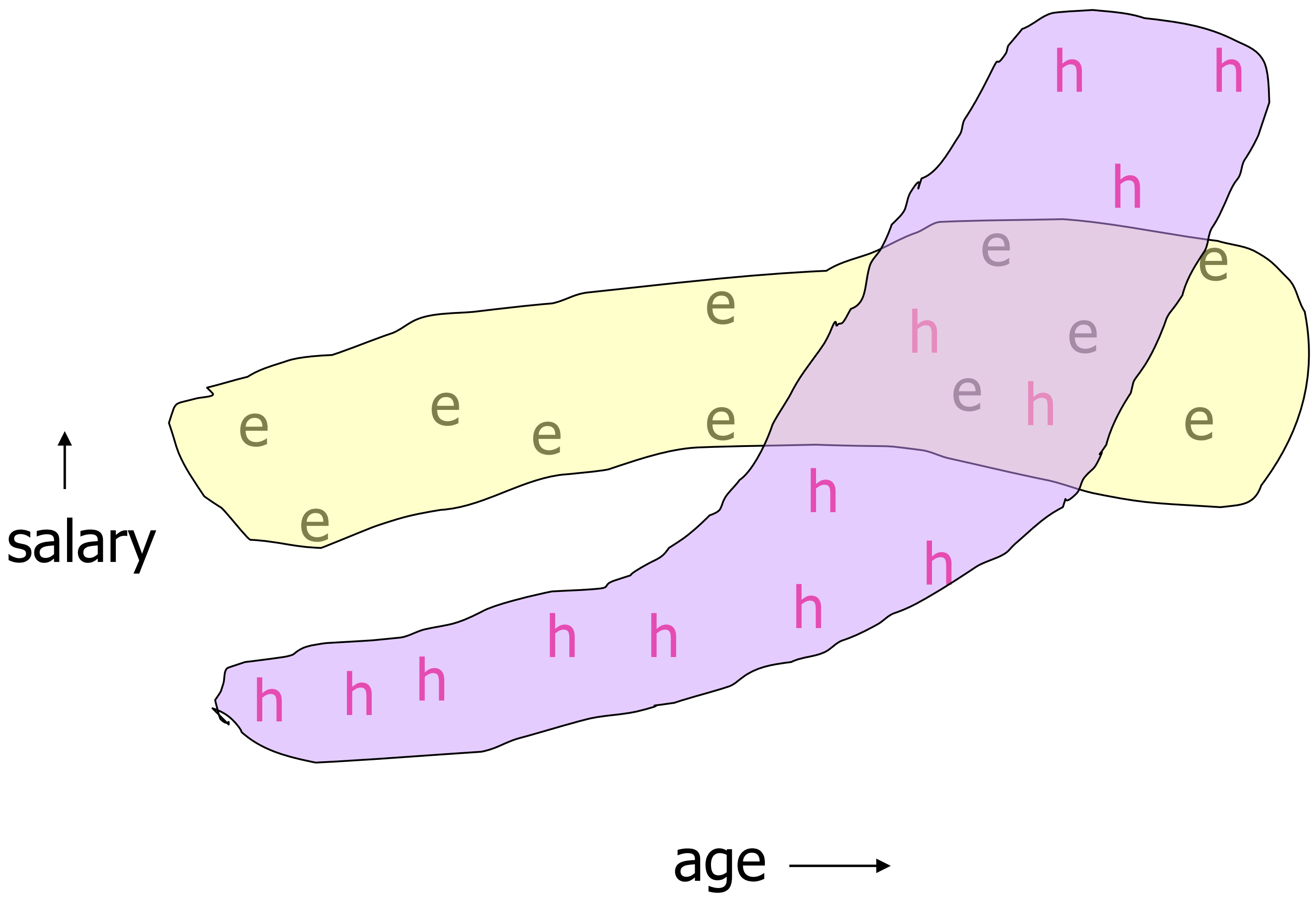
- Compute the variance of the combined subcluster.
  - *N*, SUM, and SUMSQ allow us to make that calculation quickly.
- Combine if the variance is below some threshold.
- Many alternatives: treat dimensions differently, consider density.

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.



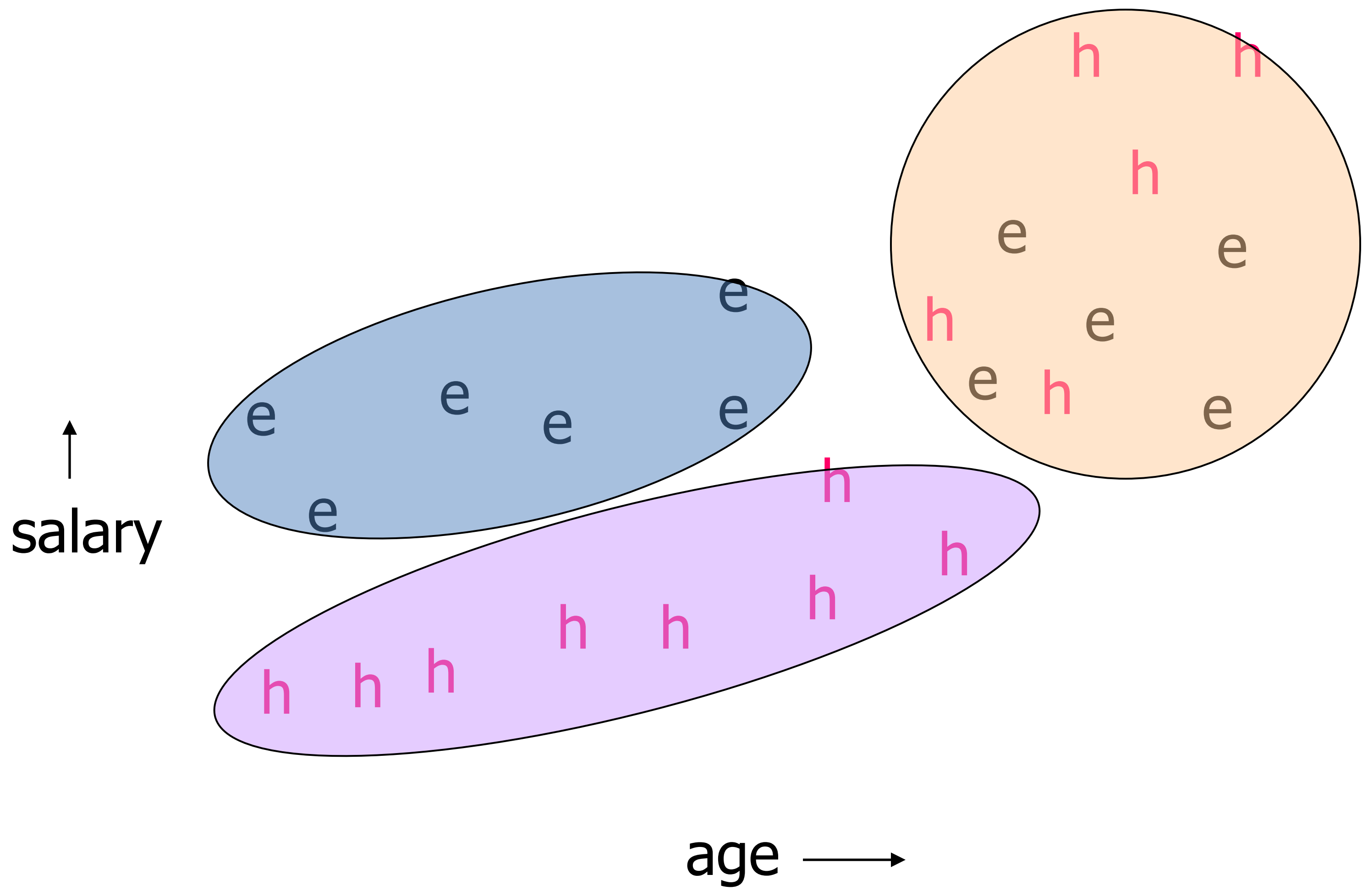
Example: Stanford Faculty Salaries



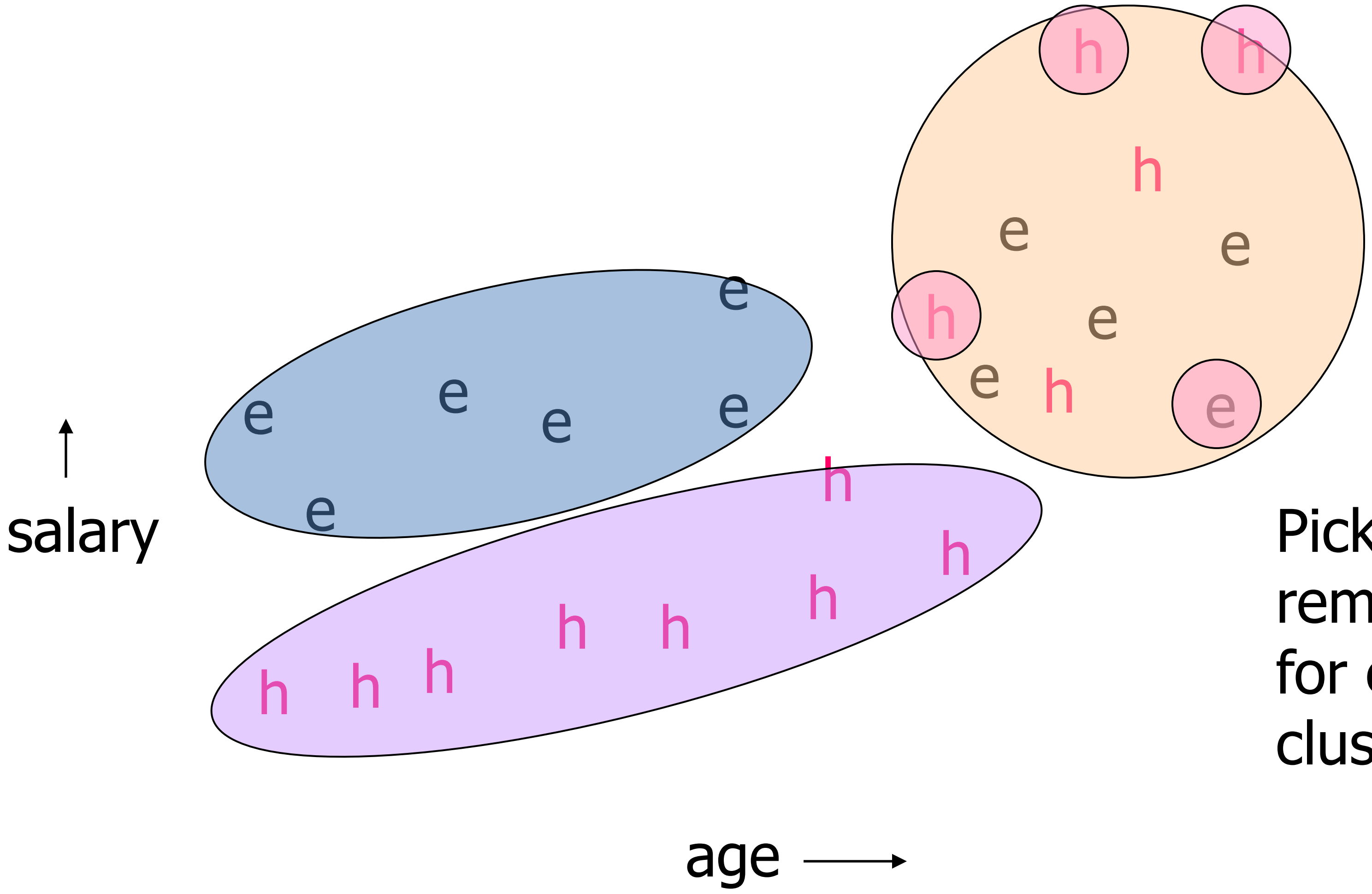
Starting CURE

- 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

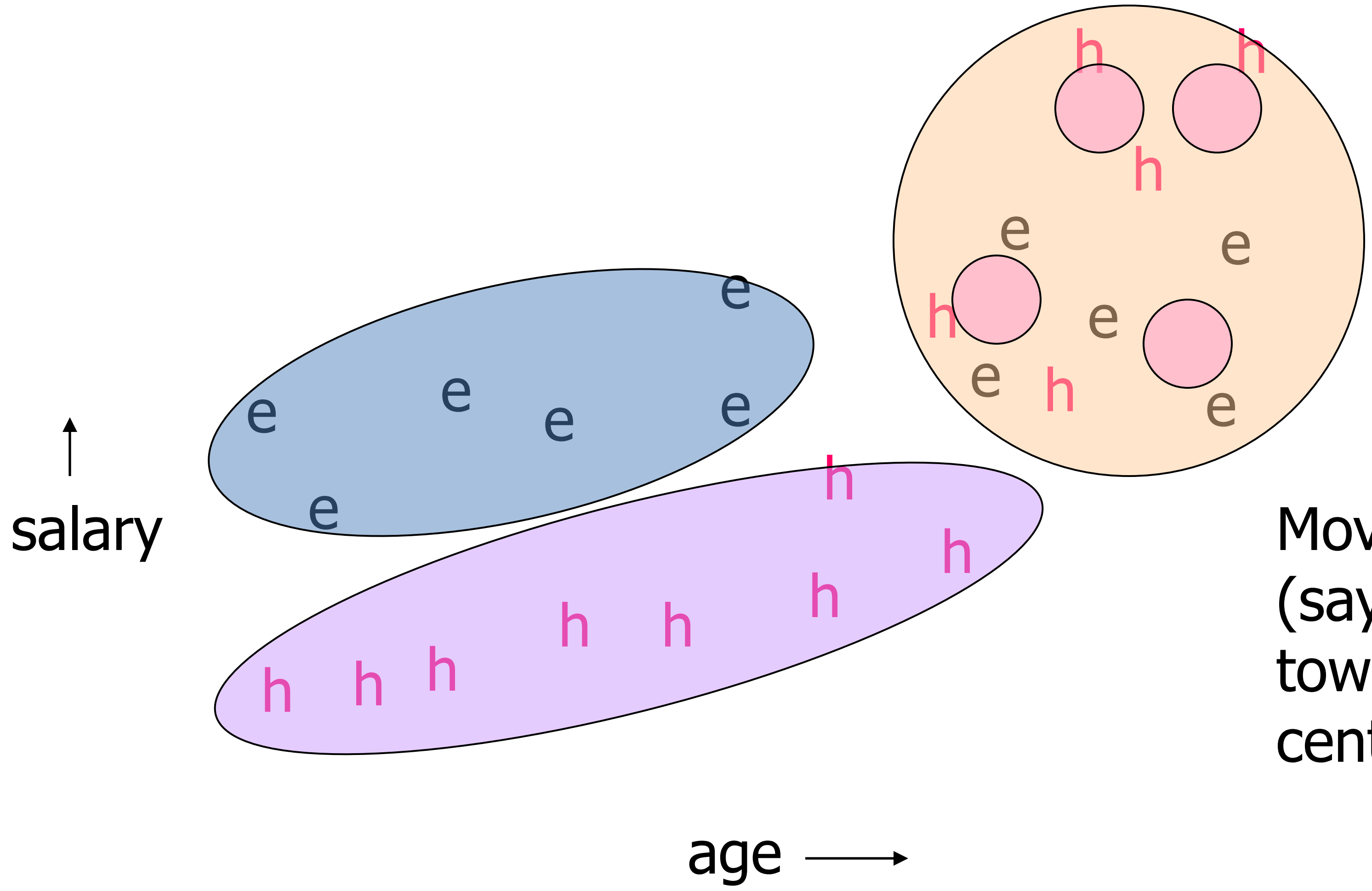


Example: Pick Dispersed Points



Pick (say) 4 remote points for each cluster.

Example: Pick Dispersed Points



Move points (say) 20% toward the centroid.

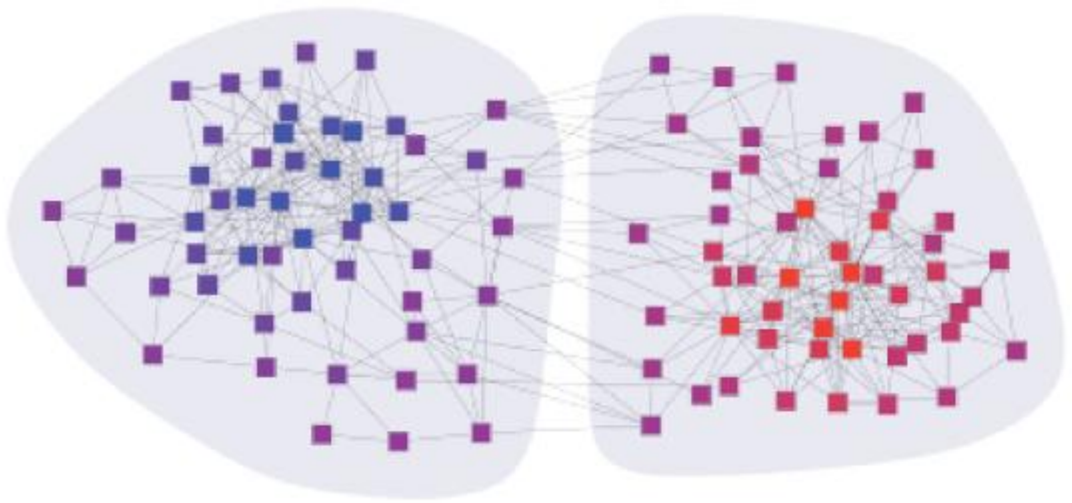
Finishing CURE

- 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.

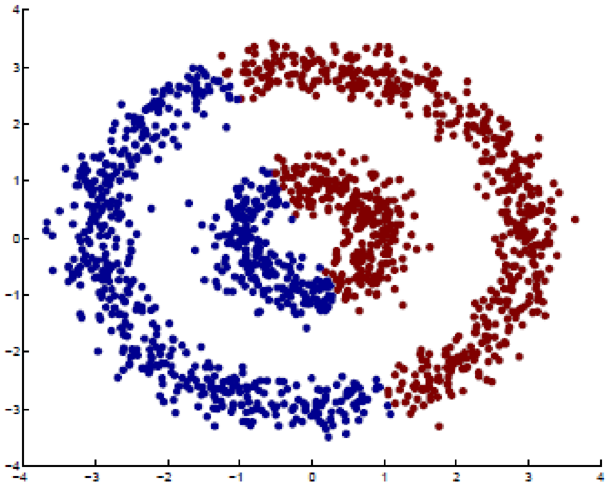
Spectral Clustering

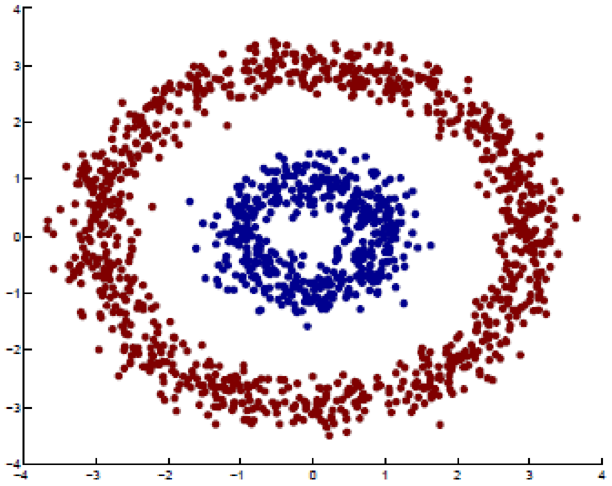
- Represent datapoints as vertices V of a graph G.
- Each pair of vertices is connected by an edge.
- Edges have weights W. Large weights mean that adjacent vertices are similar.
- The graph construction depends on the application.



Graph Partitioning

- Build a weighted graph G = (V,E,W).
- Construct a matrix L = f (W) (different variants of spectral clustering result from different functions f .
- Compute the eigenvectors of the k smallest eigenvalues of L. These provide a new representation of the original data points.
- Cluster the points in this new representation (e.g. using K-means).





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.