Clustering

Hierarchical /Agglomerative and Point-Assignment Approaches

Measures of "Goodness" for Clusters

BFR Algorithm

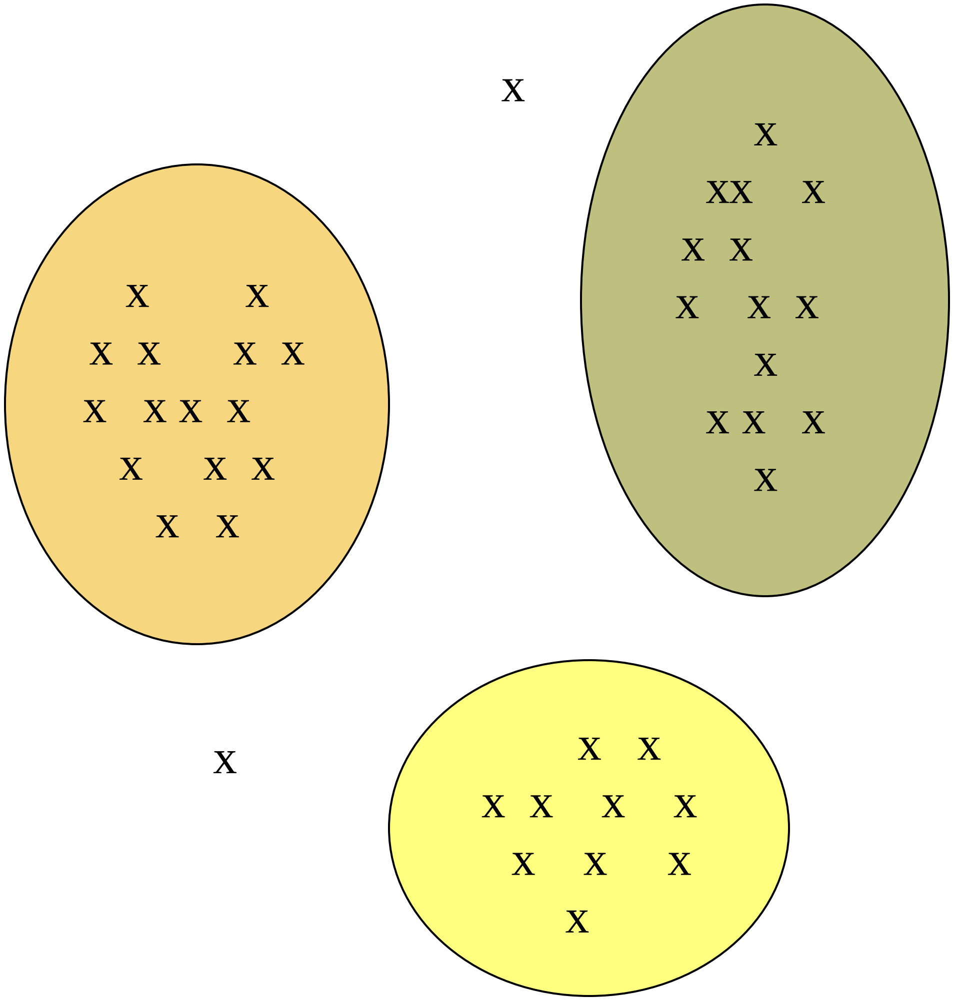
CURE Algorithm

Jeffrey D. Ullman
Stanford University

The Problem of Clustering

- Given a set of points, with a notion of distance between points, group the points into some number of *clusters*, so that members of a cluster are “close” to each other, while members of different clusters are “far.”

Example: Clusters



Problems With Clustering

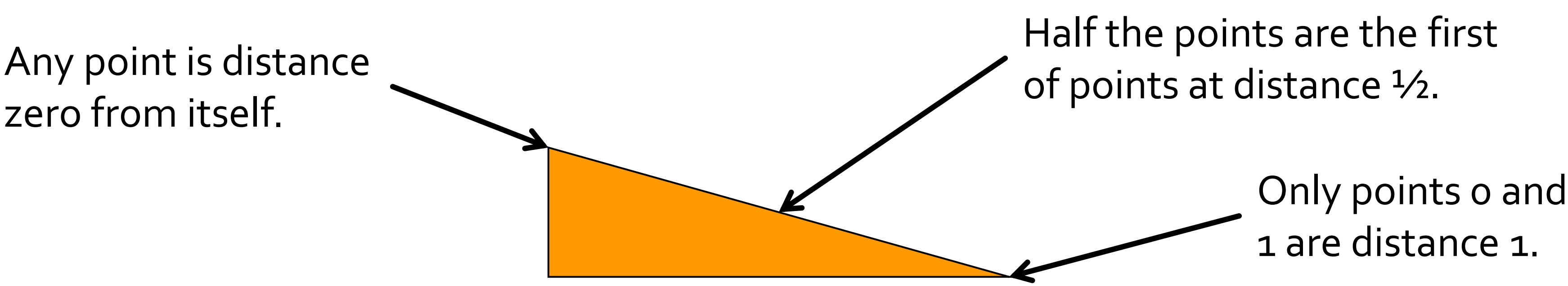
- Clustering in two dimensions looks easy.
- Clustering small amounts of data looks easy.
- And in most cases, looks are *not* deceiving.

The Curse of Dimensionality

- 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 between 0 and 1 in each dimension.
- In 2 dimensions: a variety of distances between 0 and 1.41.
- In any number of dimensions, the distance between two random points in any one dimension is distributed as a triangle.



Example – Continued

- 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.
  - I.e., “all points are the same distance apart.”

Euclidean and Non-Euclidean Distances

- Euclidean spaces have dimensions, and points have coordinates in each dimension.
- Distance between points is usually the square-root of the sum of the squares of the distances in each dimension.
- Non-Euclidean spaces have a distance measure, but points do not really have a position in the space.
  - Big problem: cannot “average” points.

Example: DNA Sequences

- Objects are sequences of {C,A,T,G}.
- Distance between sequences = *edit distance* = the minimum number of inserts and deletes needed to turn one into the other.
  - Notice: no way to "average" two strings.
- In practice, the distance for DNA sequences is more complicated: allows other operations like *mutations* (change of a symbol into another) or reversal of substrings.

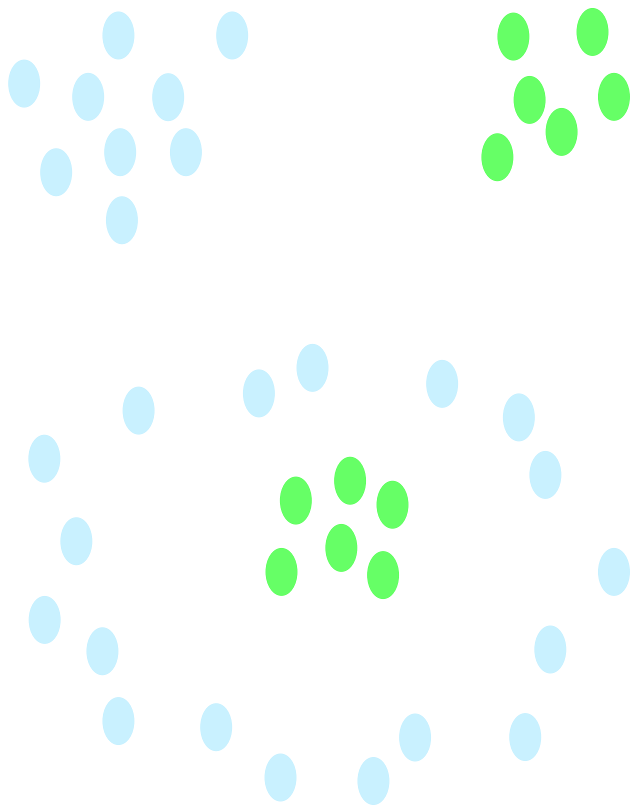
Methods of Clustering

- Hierarchical (Agglomerative):
  - Initially, each point in cluster by itself.
  - Repeatedly combine the two “nearest” clusters into one.
- Point Assignment:
  - Maintain a set of clusters.
  - Place points into their “nearest” cluster.
  - Possibly split clusters or combine clusters as we go.

Which is Better?

- Point assignment good when clusters are nice, convex shapes.
- Hierarchical can win when shapes are weird.

Aside: if you realized you had concentric clusters, you could map points based on distance from center, and turn the problem into a simple, one-dimensional case.



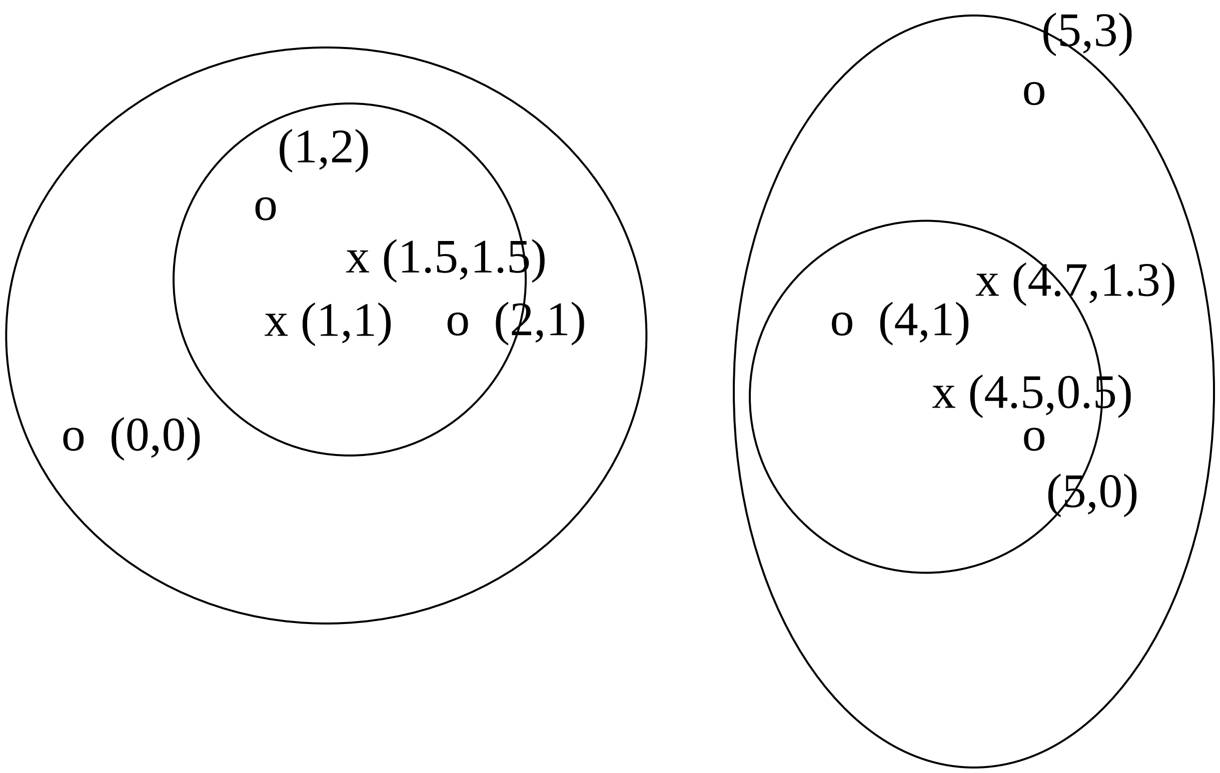
Hierarchical Clustering

- Two important questions:
  1. How do you determine the “nearness” of clusters?
  2. How do you represent a cluster of more than one point?

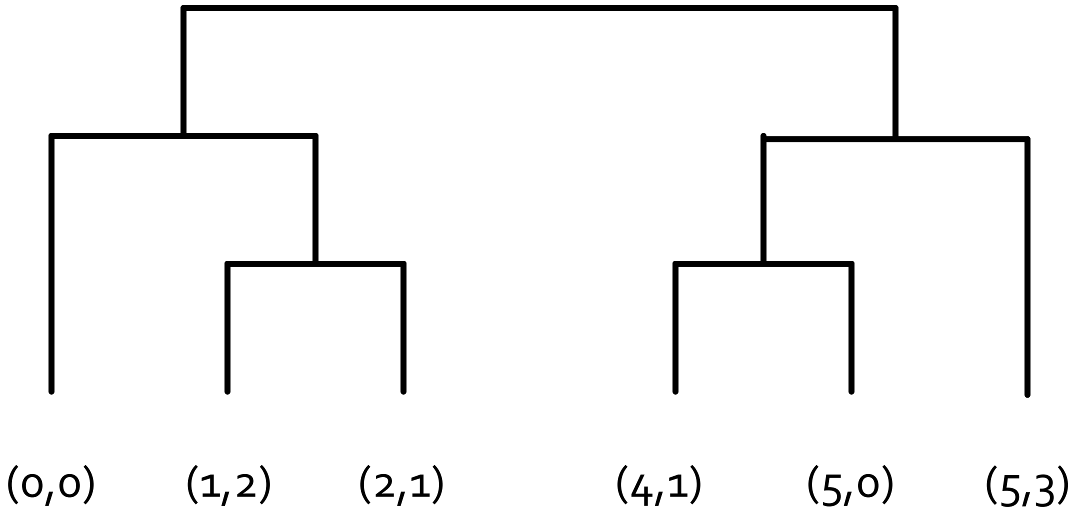
Hierarchical Clustering – (2)

- 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



Dendrogram



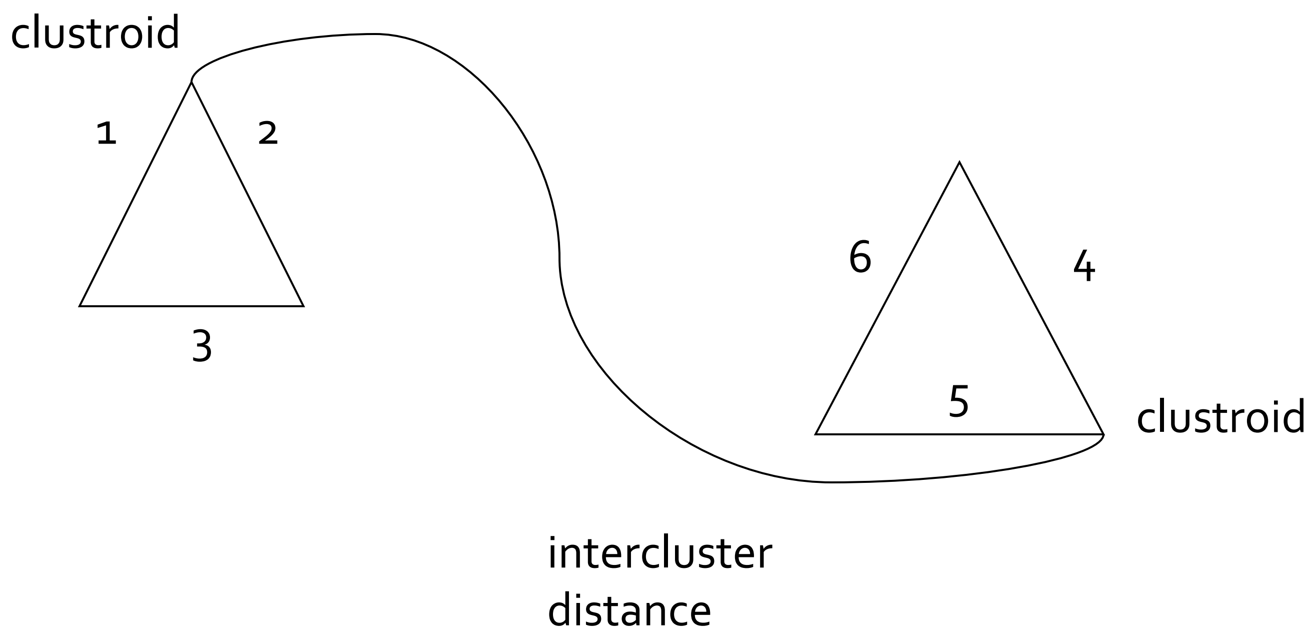
And in the Non-Euclidean Case?

- 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.

“Closest” Point?

- Possible meanings:
  1. Smallest maximum distance to the other points.
  2. Smallest average distance to other points.
  3. Smallest sum of squares of distances to other points.
  4. Etc., etc.

Example: Intercluster Distance



Other Approaches to 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 centroid or 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: 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.

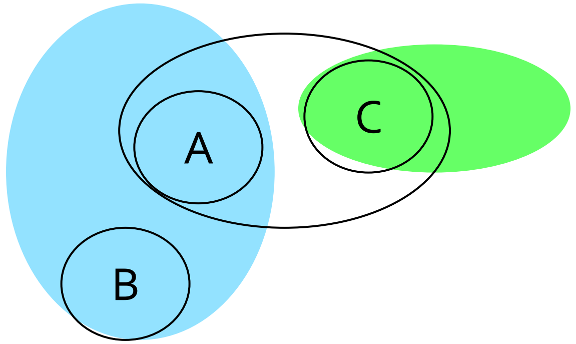
Which is Best

- It really depends on the shape of clusters.
  - Which you may not know in advance.
- Example: we'll compare two approaches:

1. Merge clusters with smallest distance between centroids (or clustroids for non-Euclidean).
2. Merge clusters with the smallest distance between two points, one from each cluster.

Case 1: Convex Clusters

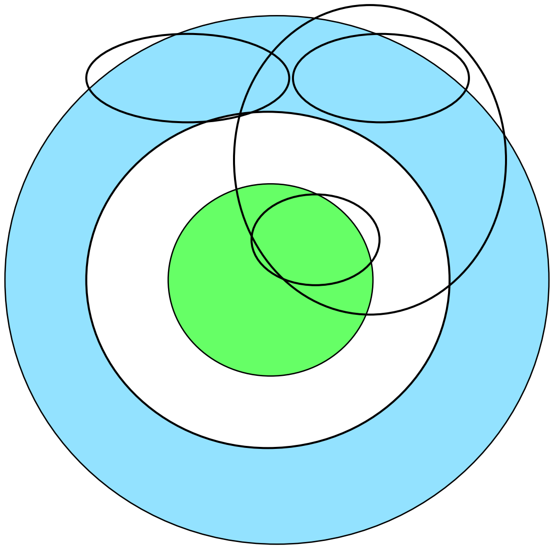
- Centroid-based merging works well.
- But merger based on closest members might accidentally merge incorrectly.



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

Case 2: Concentric Clusters

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



k–Means Algorithm(s)

- An example of point-assignment.
- Assumes Euclidean space.
- Start by picking *k*, the number of clusters.
- Initialize clusters with a *seed* (= 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.
    - OK, as long as there are no *outliers* (points that are far from any reasonable cluster).

k-Means++

- Basic idea: pick a small sample of points, cluster them by any algorithm, and use the centroids as a seed.
- In k-means++, sample size = k times a factor that is logarithmic in the total number of points.
- Sequentially pick sample points randomly, 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 picked point.

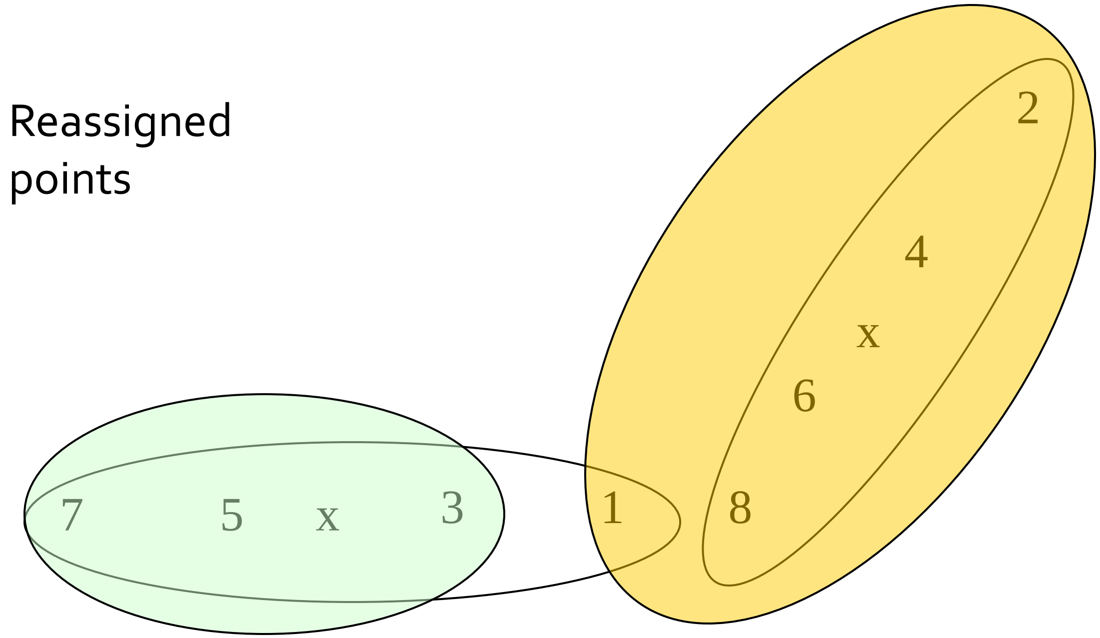
k-Means||

- k-means++, like other seed methods, is sequential.
  - You need to update D(p) for each unpicked p due to new point.
- Naturally parallel: many compute nodes can each handle a small set of points.
  - Each picks a few new sample points using same D(p).
- Really important and common trick: don't update after every selection; rather make many selections at one round.
  - Suboptimal picks don't really matter.

Populating Clusters

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

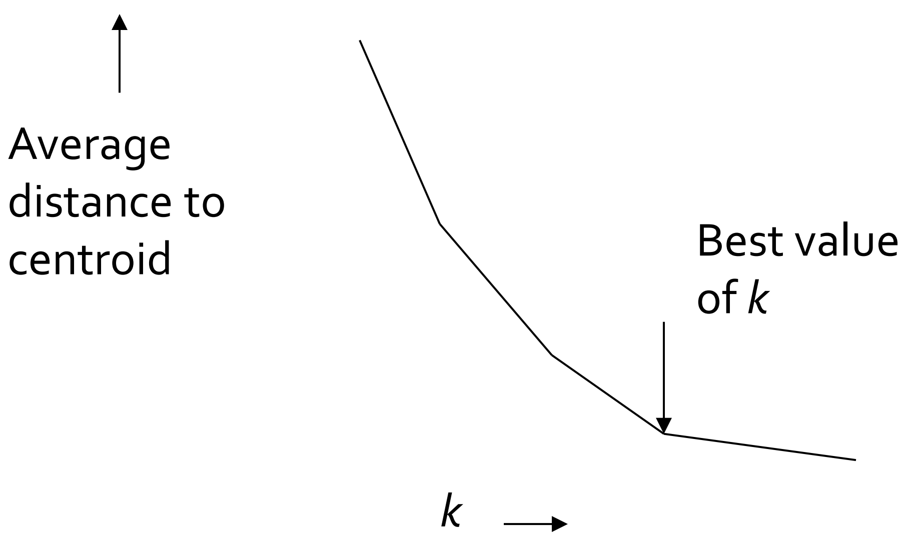
Example: Assigning Clusters



Clusters after first round

Getting *k* Right

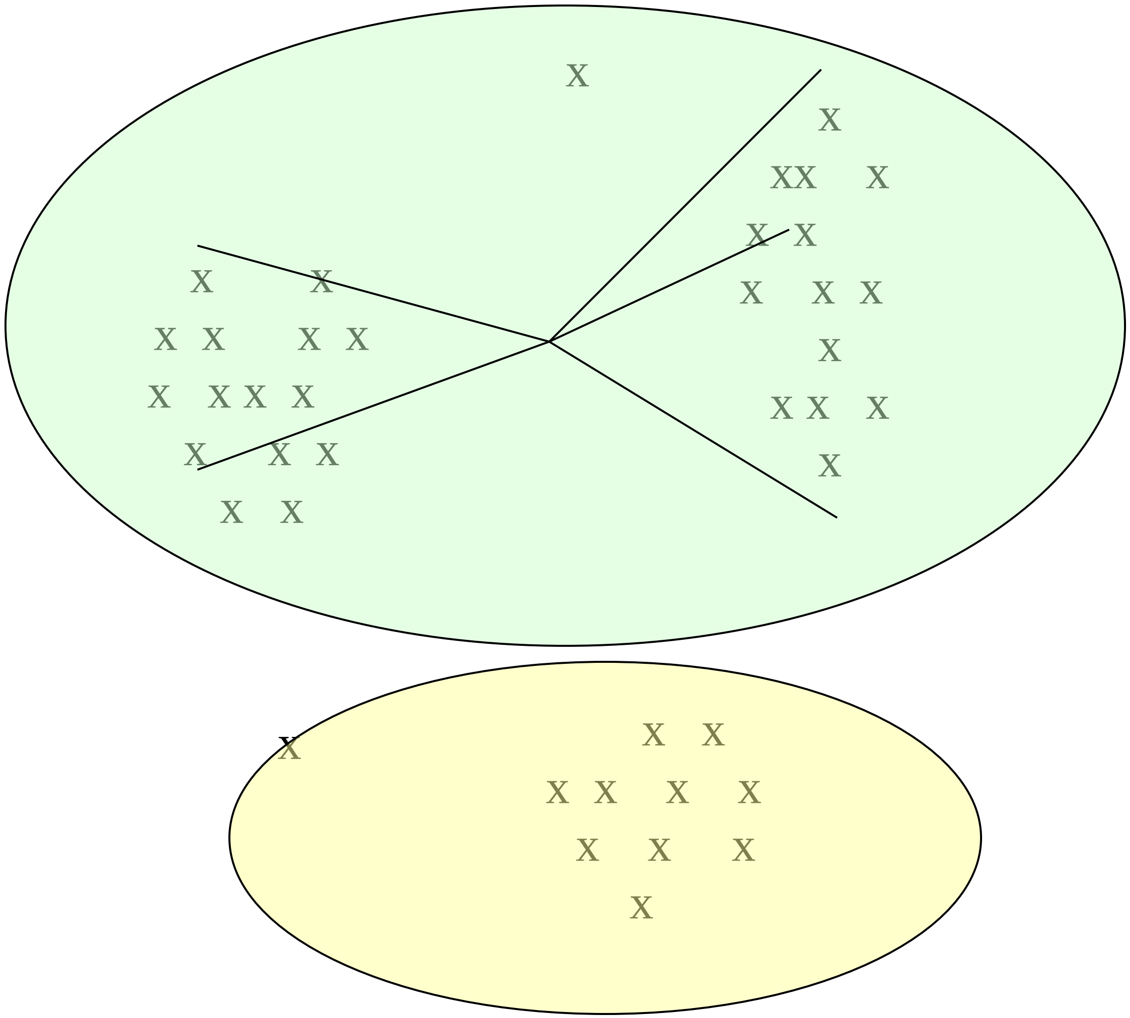
- Try different *k*, looking at the change in the average distance to centroid, as *k* increases.
- Average falls rapidly until right *k*, then changes little.



Note: binary search for k is possible.

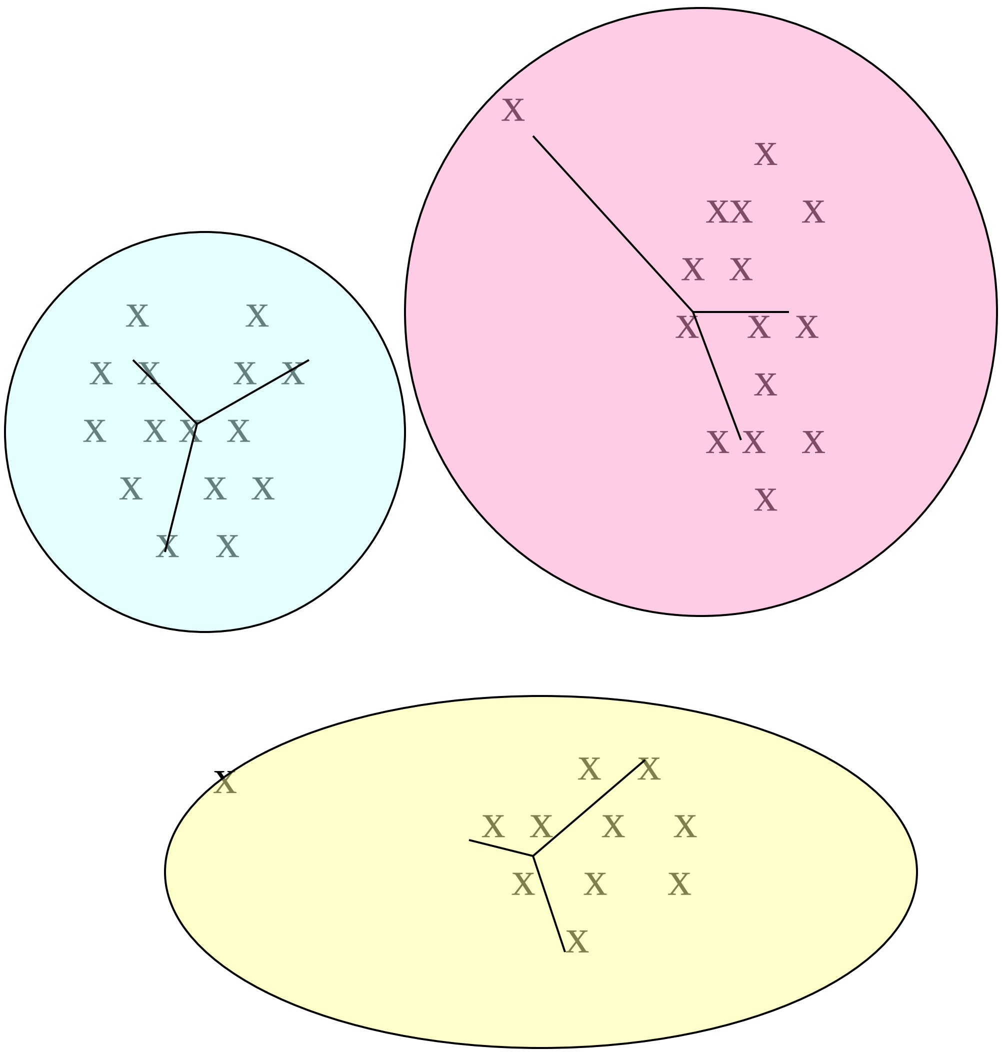
Example: Picking *k*

Too few;
many long
distances
to centroid.



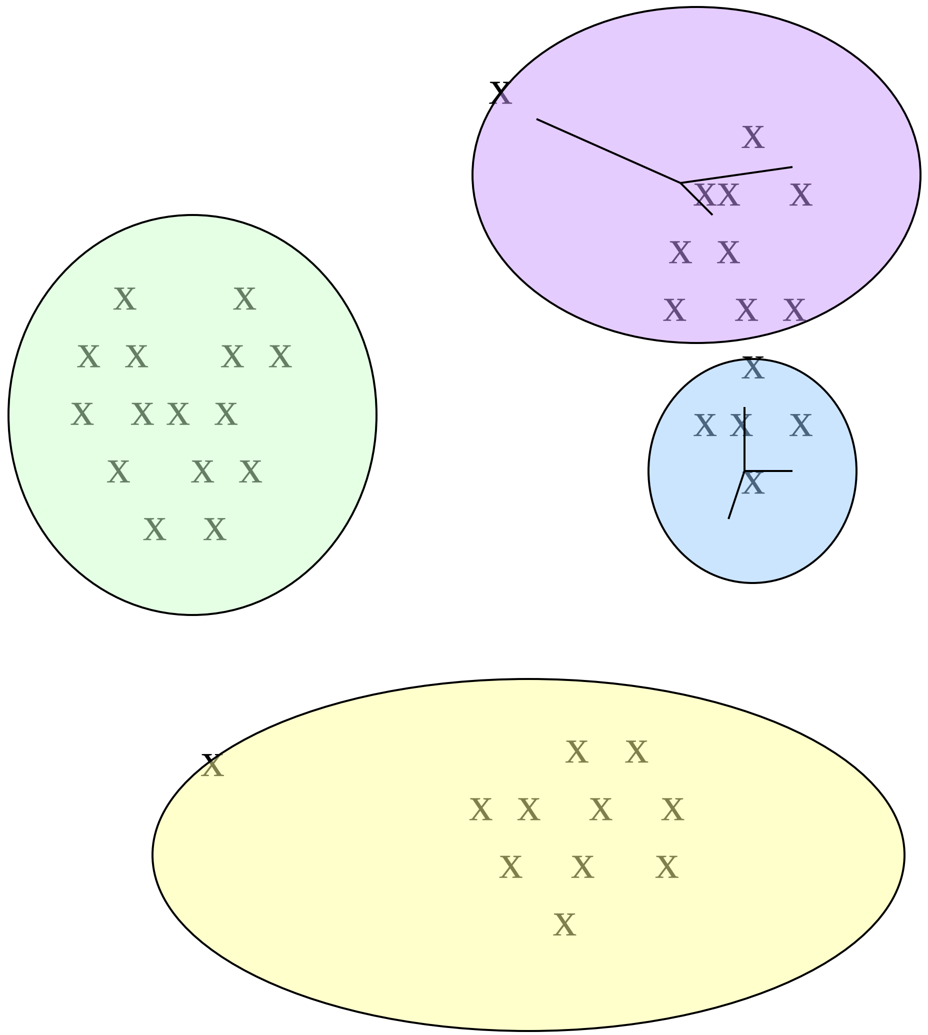
Example: Picking *k*

Just right;
distances
rather short.



Example: Picking *k*

Too many;
little improvement
in average
distance.



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.

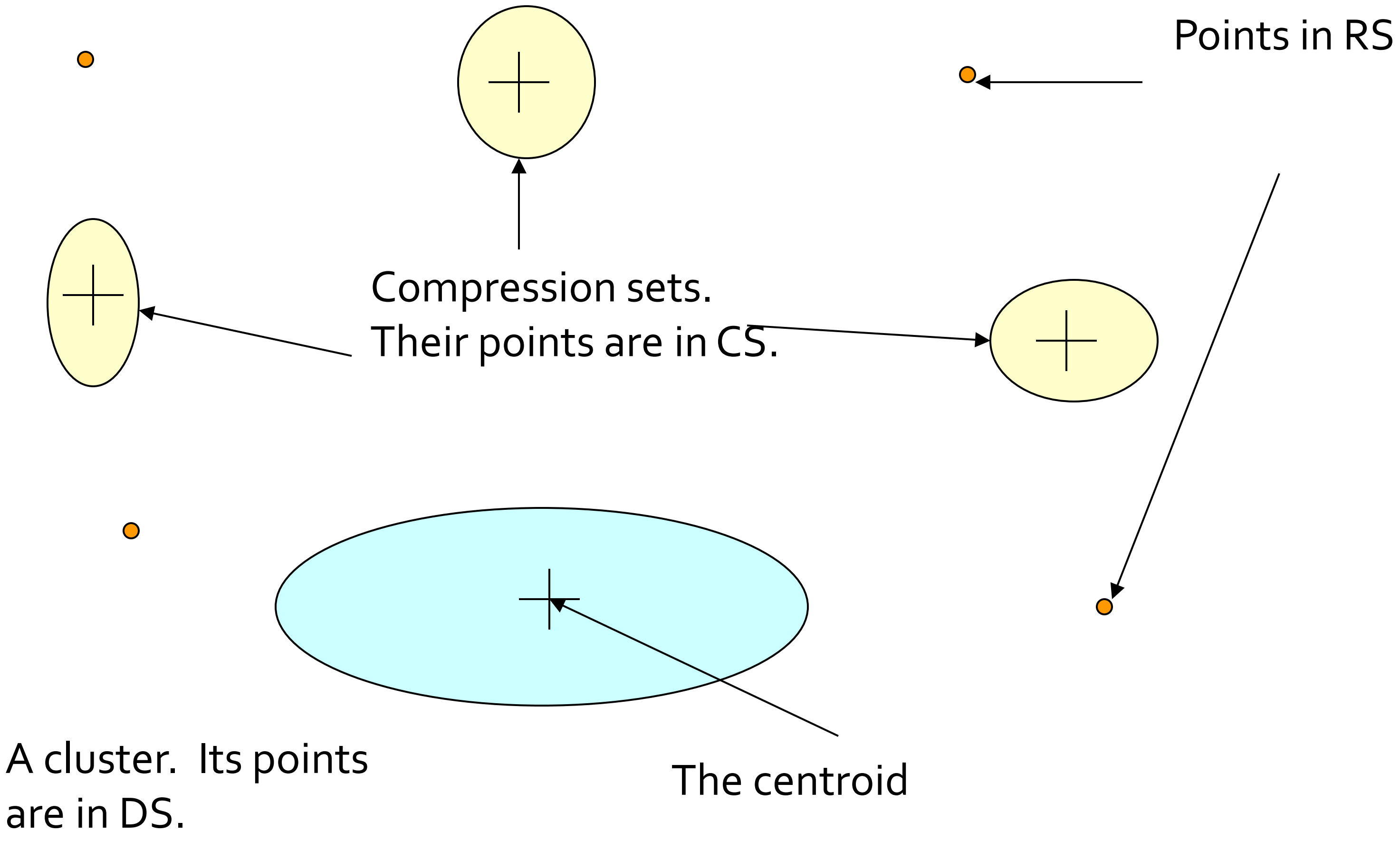
BFR – (2)

- Points are read one main-memory-full at a time.
- Most points from previous memory loads are summarized by simple statistics.
  - Also kept in main memory, which limits how many points can be read in one "memory load."
- To begin, from the initial load we select the initial *k* centroids by some sensible approach.

Three Classes of Points

1. The *discard set* (*DS*): points close enough to a centroid to be summarized.
2. The *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.
3. The *retained set* (*RS*): isolated points.

“Galaxies” Picture



Summarizing Sets of Points

- Each cluster in the discard set and each compression set is summarized by:
  1. The number of points, *N*.
  2. The vector SUM, whose i th component is the sum of the coordinates of the points in the i th dimension.
  3. The vector SUMSQ: i th component = sum of squares of coordinates in i th dimension.

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 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.

Processing a "Memory-Load" of Points

1. Find those points that are "sufficiently close" to a cluster centroid; add those points to that cluster and the DS.
2. 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.

Processing – (2)

3. Adjust statistics of the clusters to account for the new points.
   - Consider merging compressed sets in the CS.
4. 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:
  1. The *Mahalanobis distance* is less than a threshold.
  2. Low likelihood of the currently nearest centroid changing.

Mahalanobis Distance

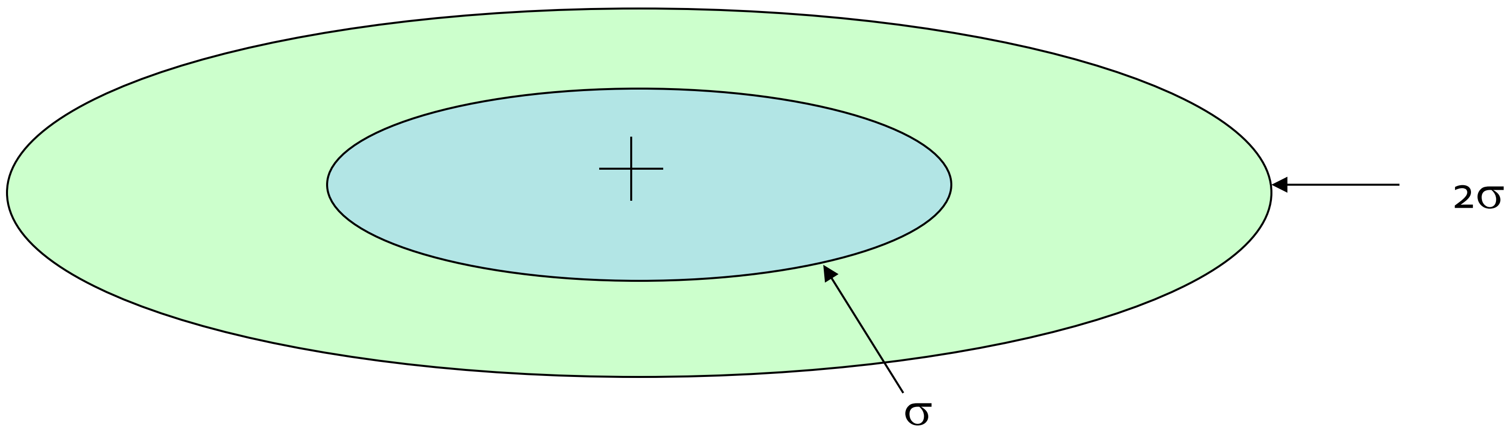
- Normalized Euclidean distance from centroid.
- For point $(x_1,\dots, x_k)$ and centroid $(c_1,\dots, c_k)$:

1. Normalize in each dimension: $y_i = (x_i - c_i)/\sigma_i$
    - σ_i = standard deviation in i^{th} dimension for this cluster.
2. Take sum of the squares of the y_i's.
3. Take the square root.

Mahalanobis Distance – (2)

- If clusters are normally distributed in d dimensions, then after transformation, one standard deviation = $\sqrt{d}$.
  - I.e., 70% 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



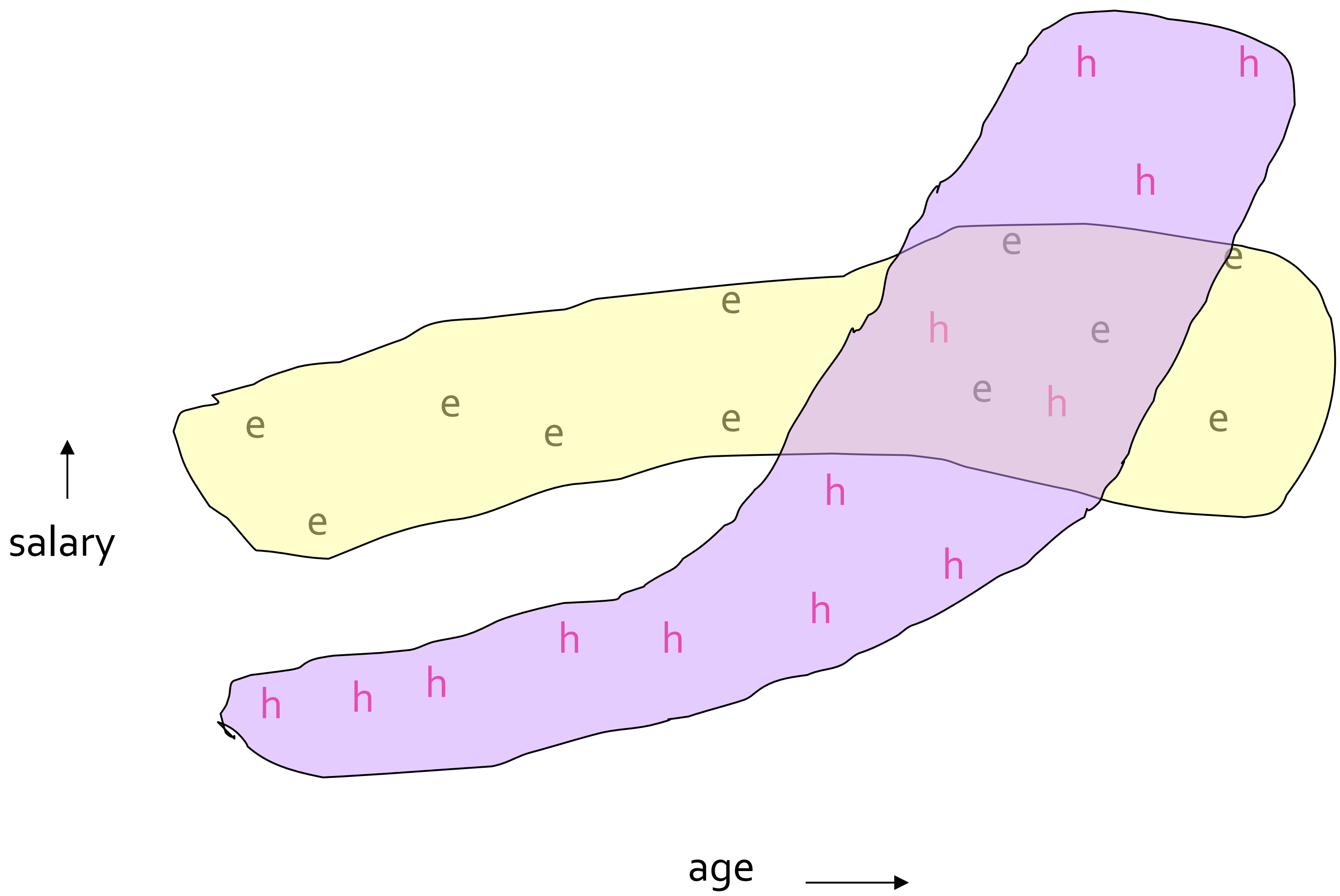
Should Two CS Subclusters Be Combined?

- Similar to measuring cohesion. For example:
- Compute the variance of the combined subcluster, in each dimension.
  - *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.

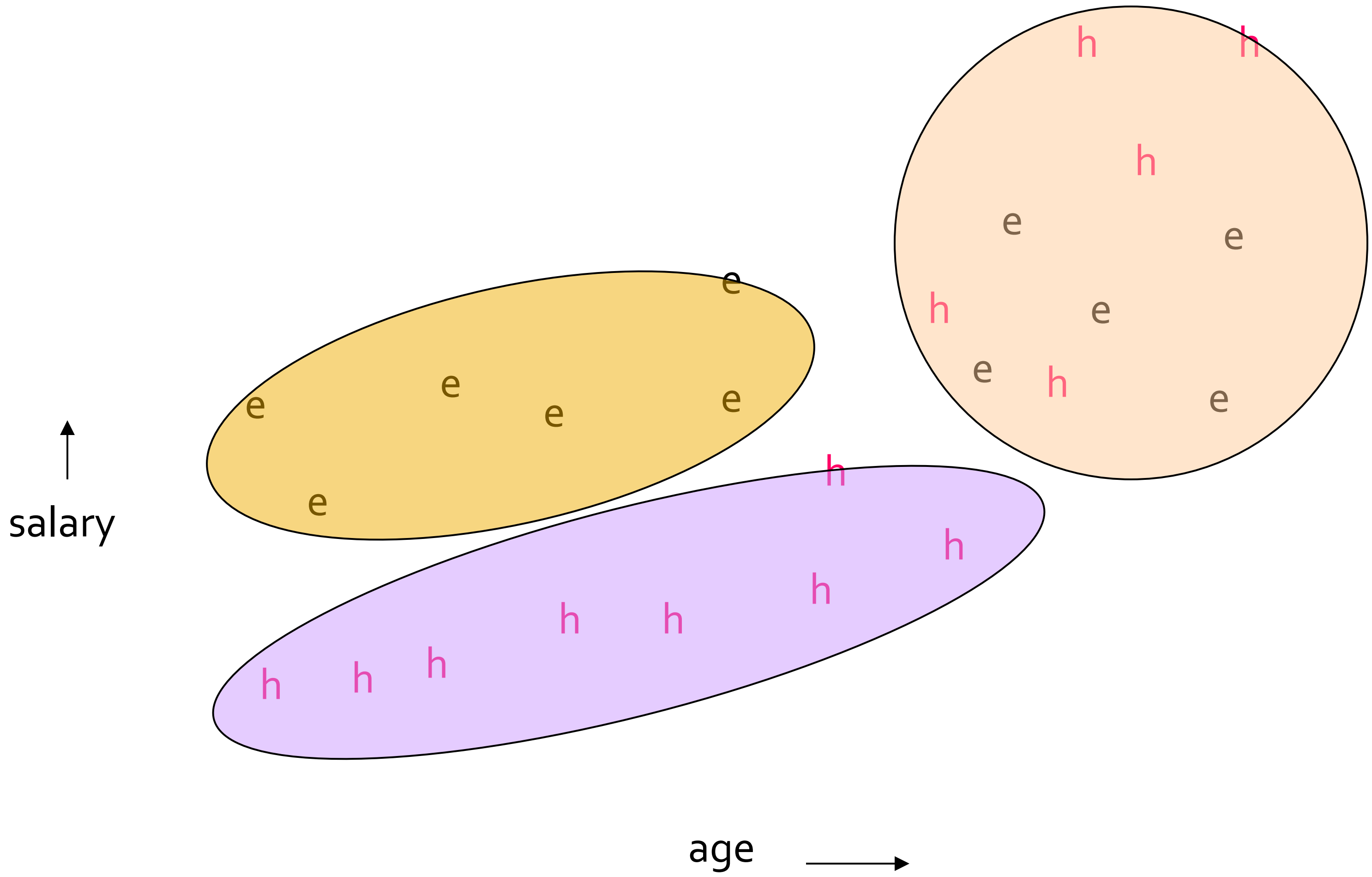
Example: Stanford Faculty Salaries



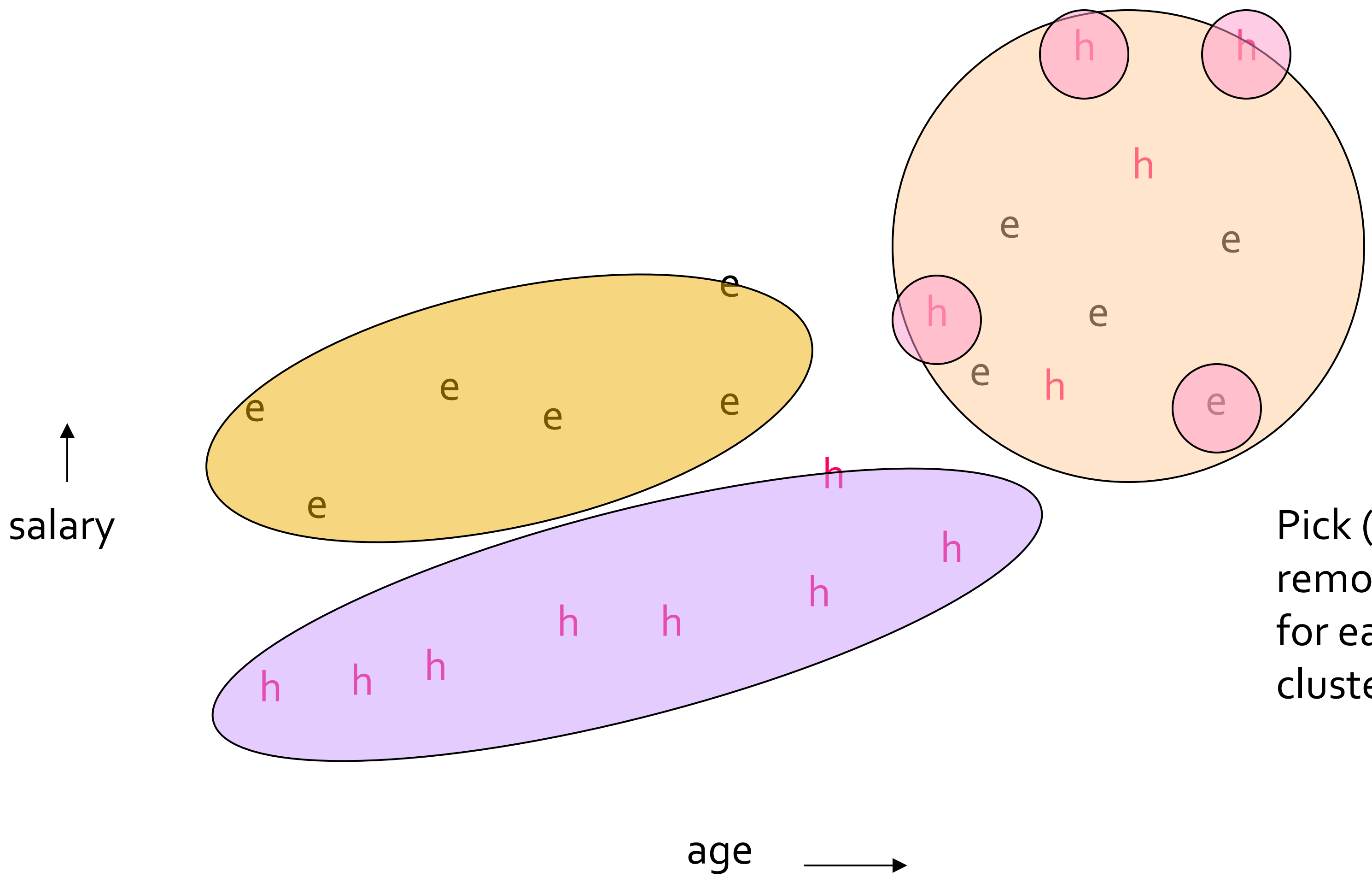
Starting CURE

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

Example: Initial Clusters

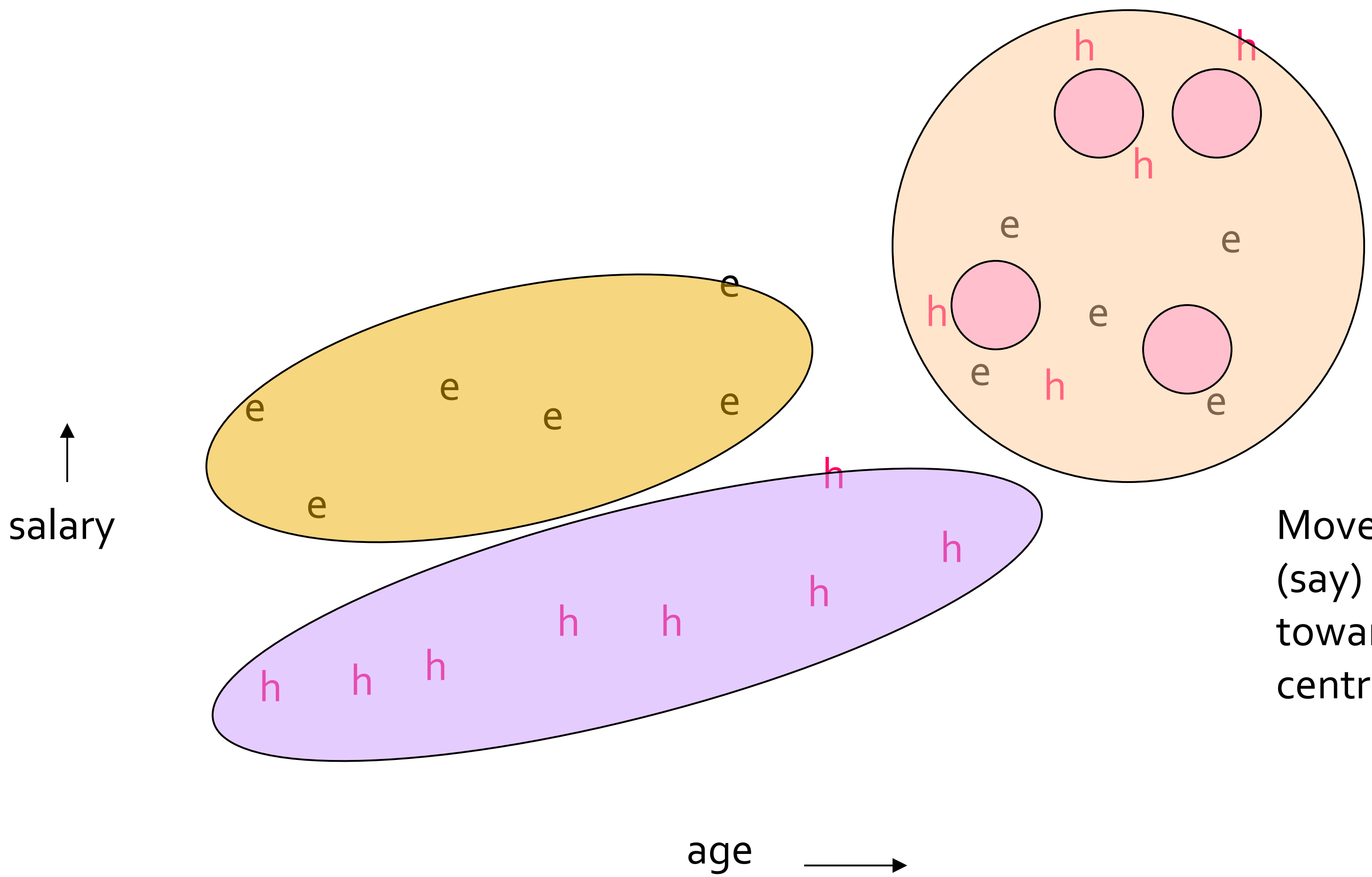


Example: Pick Dispersed Points



Pick (say) 4 remote points for each cluster.

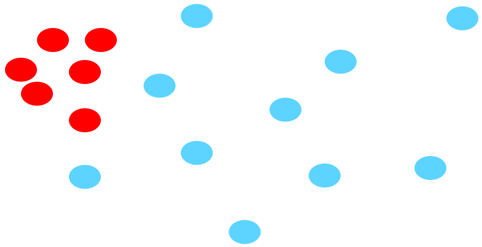
Example: Pick Dispersed Points



Move points (say) 20% toward the centroid.

Why the 20% Move Inward?

- A large, dispersed cluster will have large moves from its boundary.
- A small, dense cluster will have little move.
- Favors a small, dense cluster that is near a larger dispersed cluster.



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.