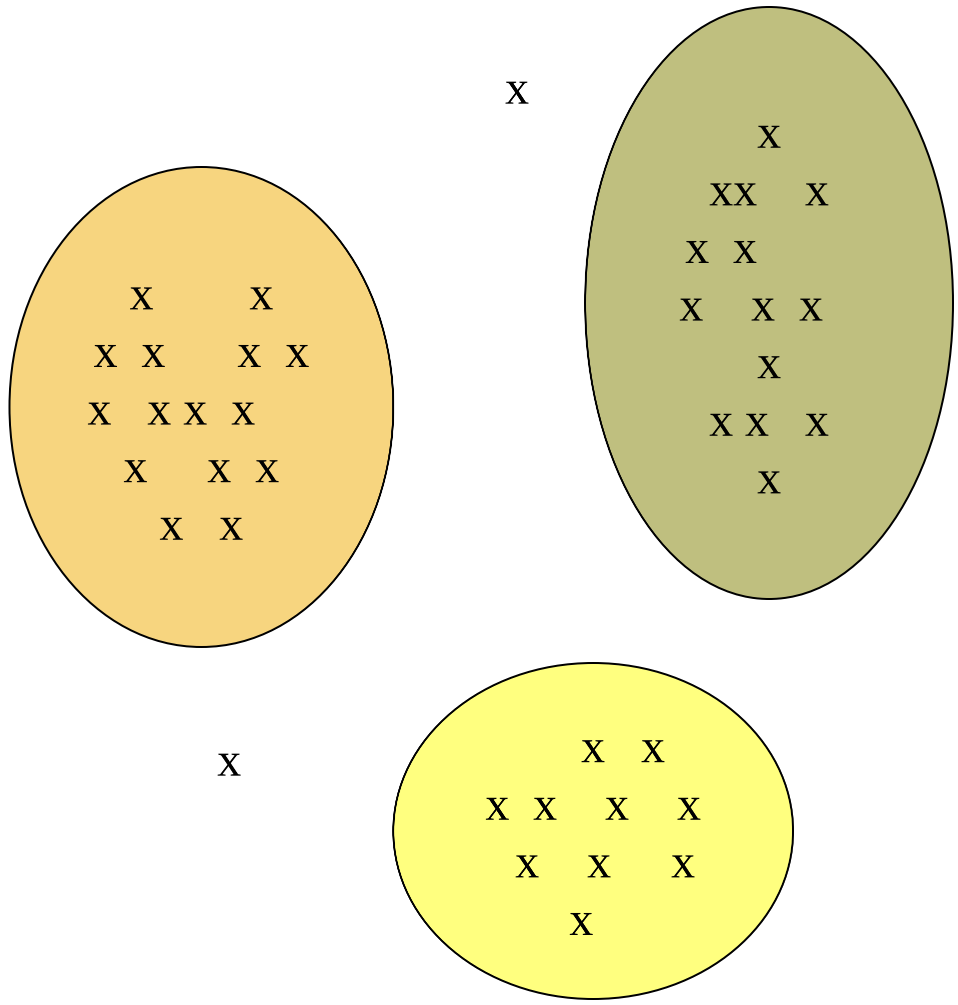
Clustering

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

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/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



Application: 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.

Clustering Movies

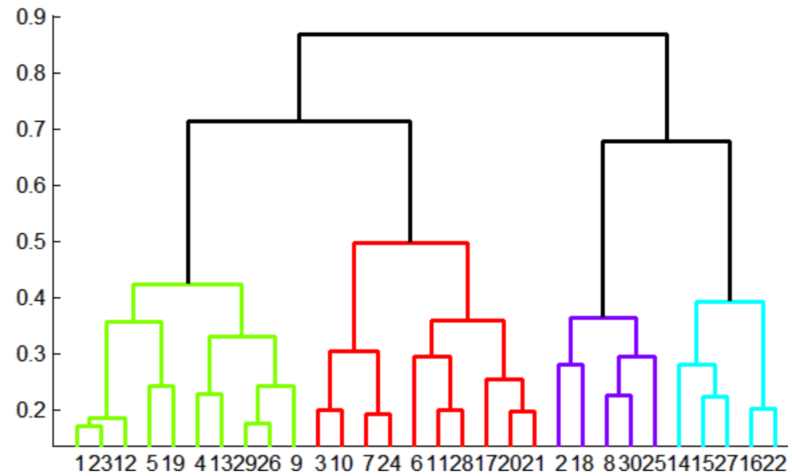
- Intuitively: Movies divide into categories, and customers prefer a few categories.
  - But what are categories really?
- Represent a movie by the customers who bought/rated it
- Similar movies's have similar sets of customers, and vice-versa
- Space of all movies:
  - Think of a space with one dimension for each customer.
    - Values in a dimension may be 0 or 1 only.
    - A movies's point in this space is $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} customer bought the movie.

Cosine, Jaccard, and Euclidean

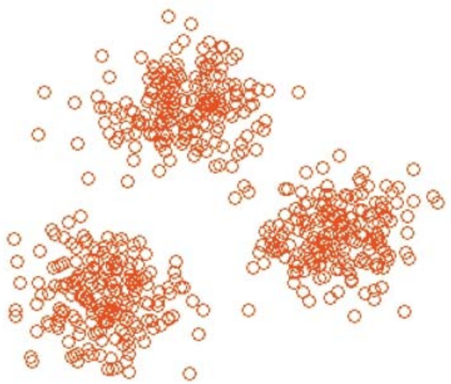
- We have a choice:
  1. Sets as vectors: measure similarity by the cosine distance.
  2. Sets as sets: measure similarity by the Jaccard distance.
  3. Sets as points: measure similarity by Euclidean distance.

Methods of Clustering

- Hierarchical:
  - Agglomerative (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one.
  - Divisive (top down):
    - Start with one cluster and recursively split it



- Point Assignment:
  - Maintain a set of clusters
  - Points belong to "nearest" cluster



Hierarchical Clustering

- Key operation:
  Repeatedly combine two nearest clusters

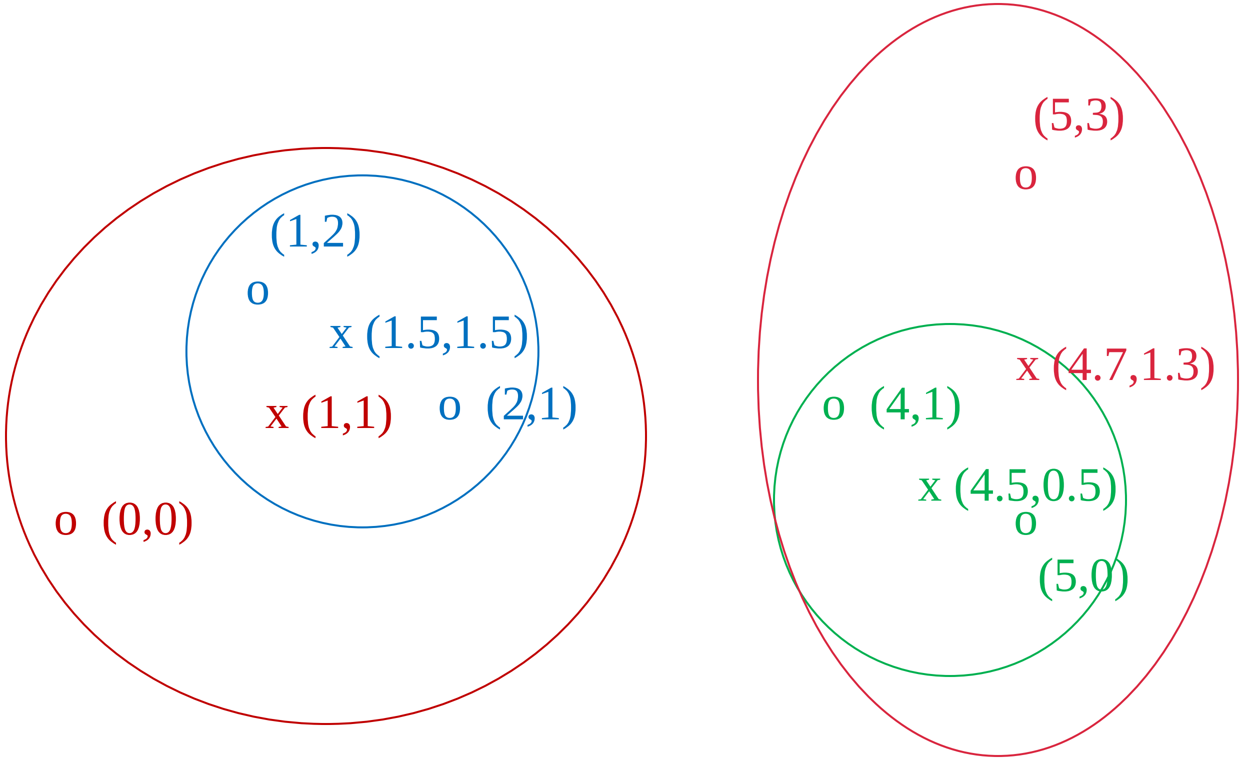
- Three important questions:
  1. How do you represent a cluster of more than one point?
  2. How do you determine the "nearness" of clusters?
  3. When to stop combining clusters?

Hierarchical Clustering – (2)

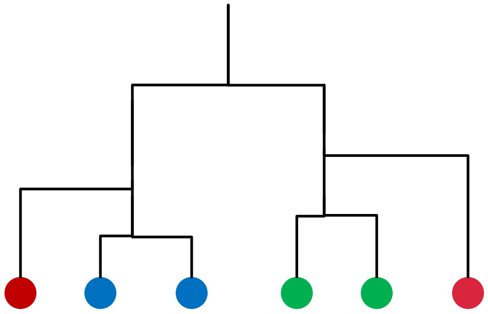
- Key problem: as you build clusters, how do you represent the location of each cluster, to tell which pair of clusters is closest?

- Euclidean case: each cluster has a *centroid* = average of its points
  - Measure cluster distances by distances of centroids

Example: Hierarchical clustering



Data



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.
     - For distance d centroid 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 avg. 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.

Implementing hierarchical clustering

- Naïve implementation:
  - At each step, compute pairwise distances between all pairs of clusters
  - $O(N^3)$
- 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/distance.
- 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

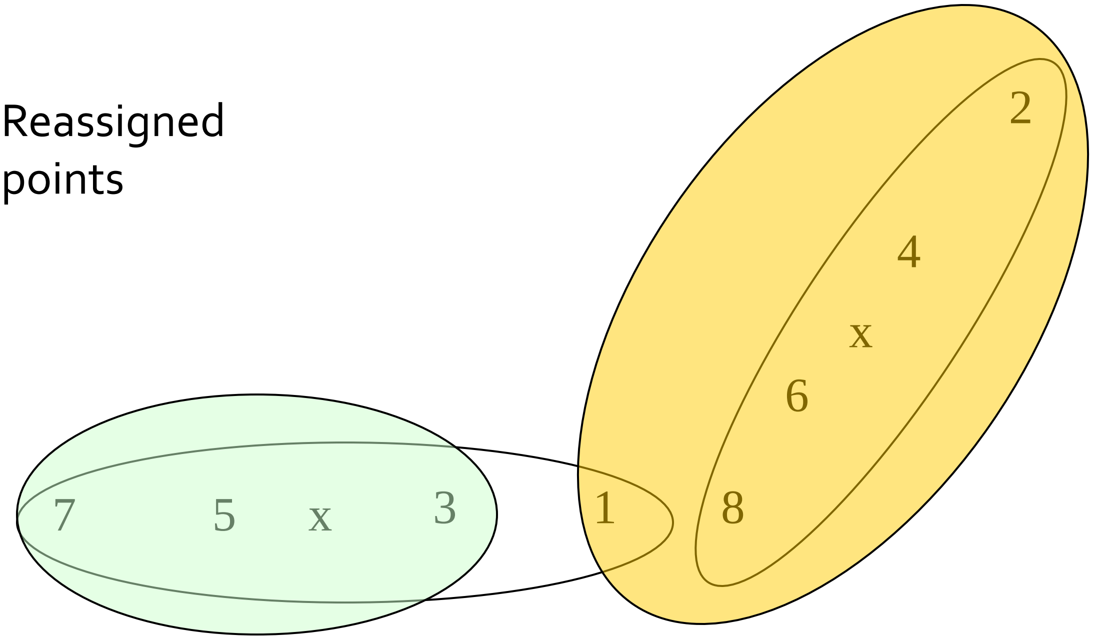
1. For each point, place it in the cluster whose current centroid it is nearest.

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

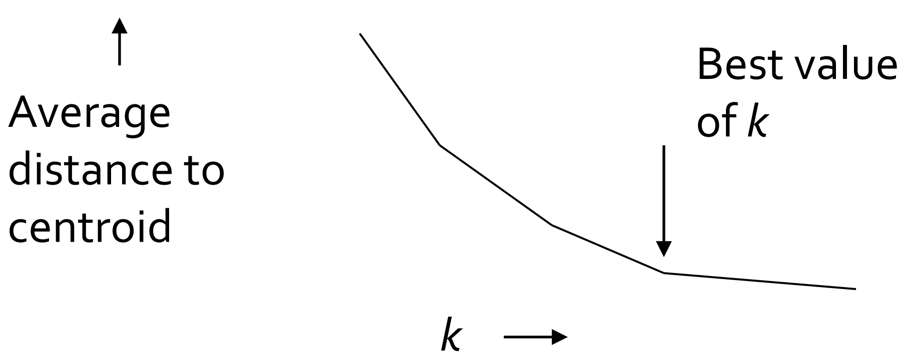
Reassigned points



Clusters after first round

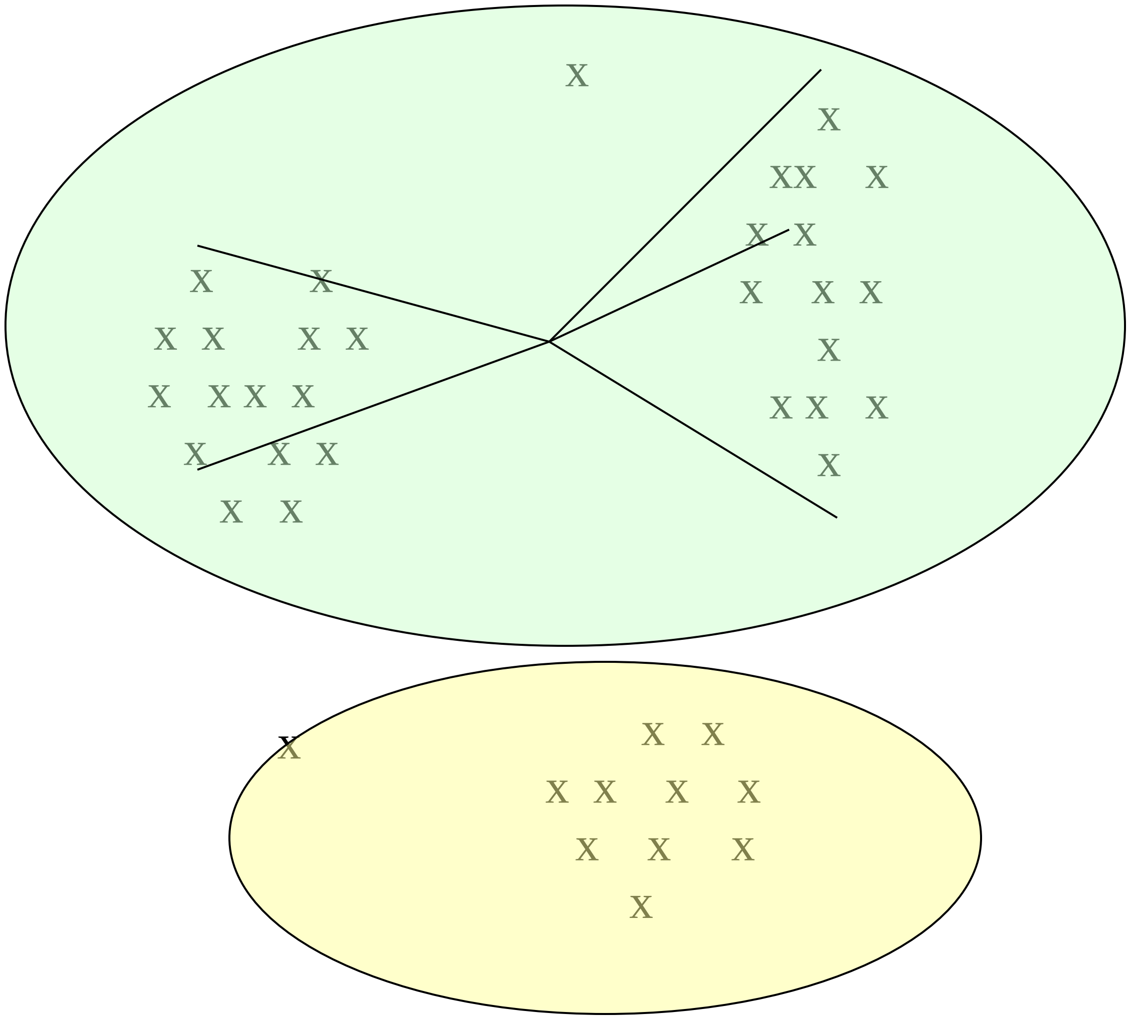
Getting *k* Right

- 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.



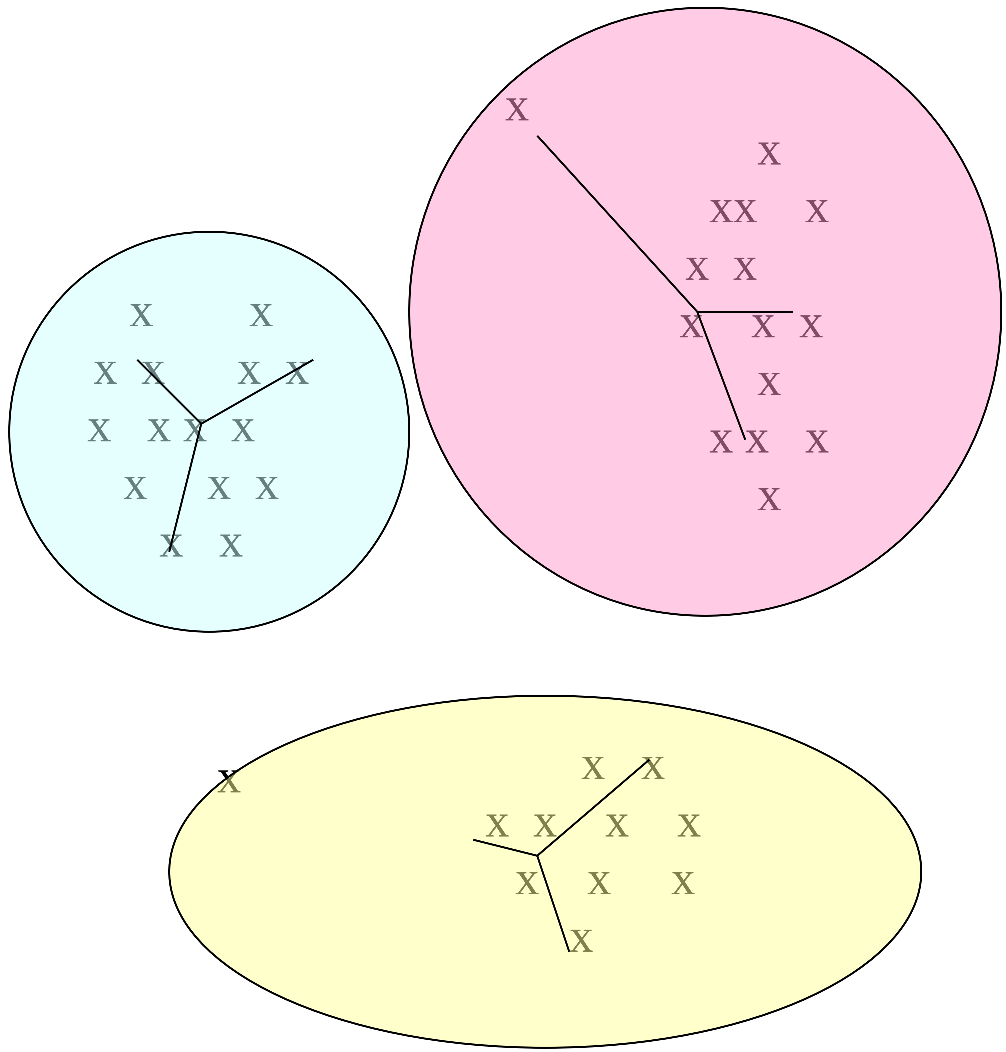
Example: Picking *k*

Too few;
many long
distances
to centroid.



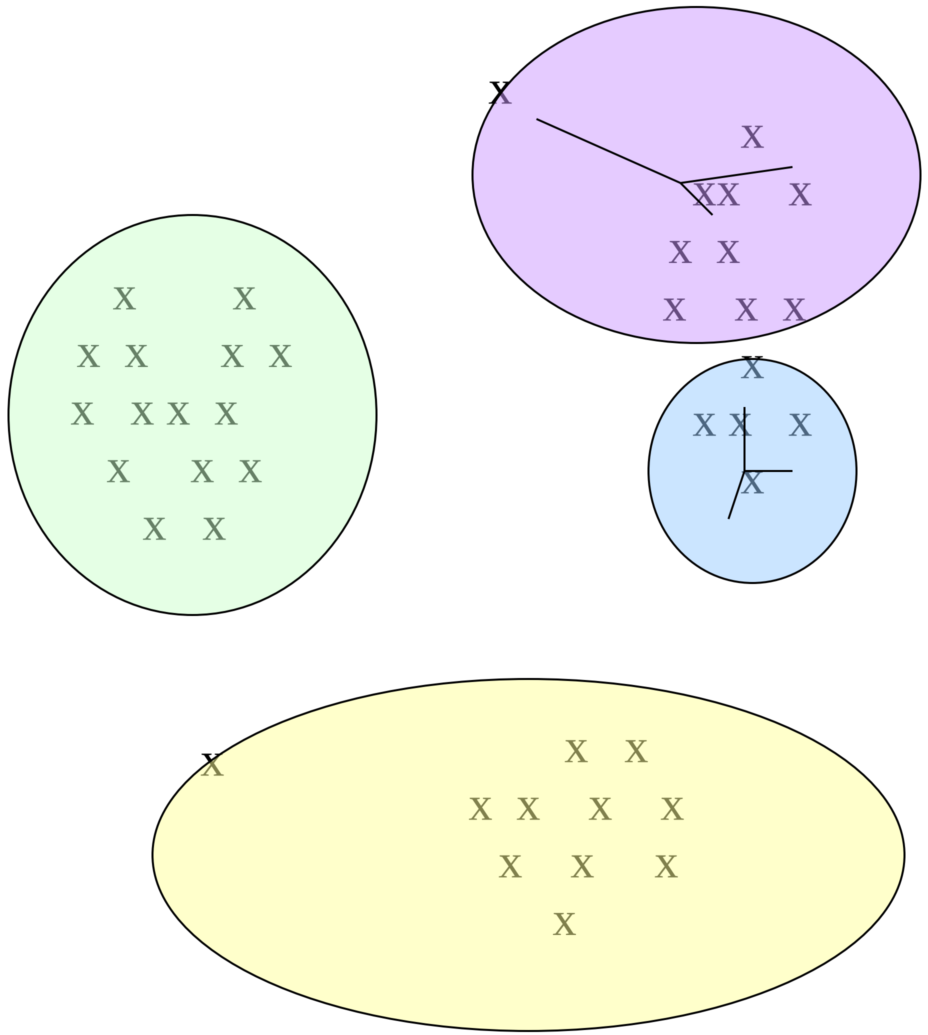
Example: Picking *k*

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.
- To begin, from the initial load we select the initial *k* centroids by some sensible approach.

Initialization: *k* -Means

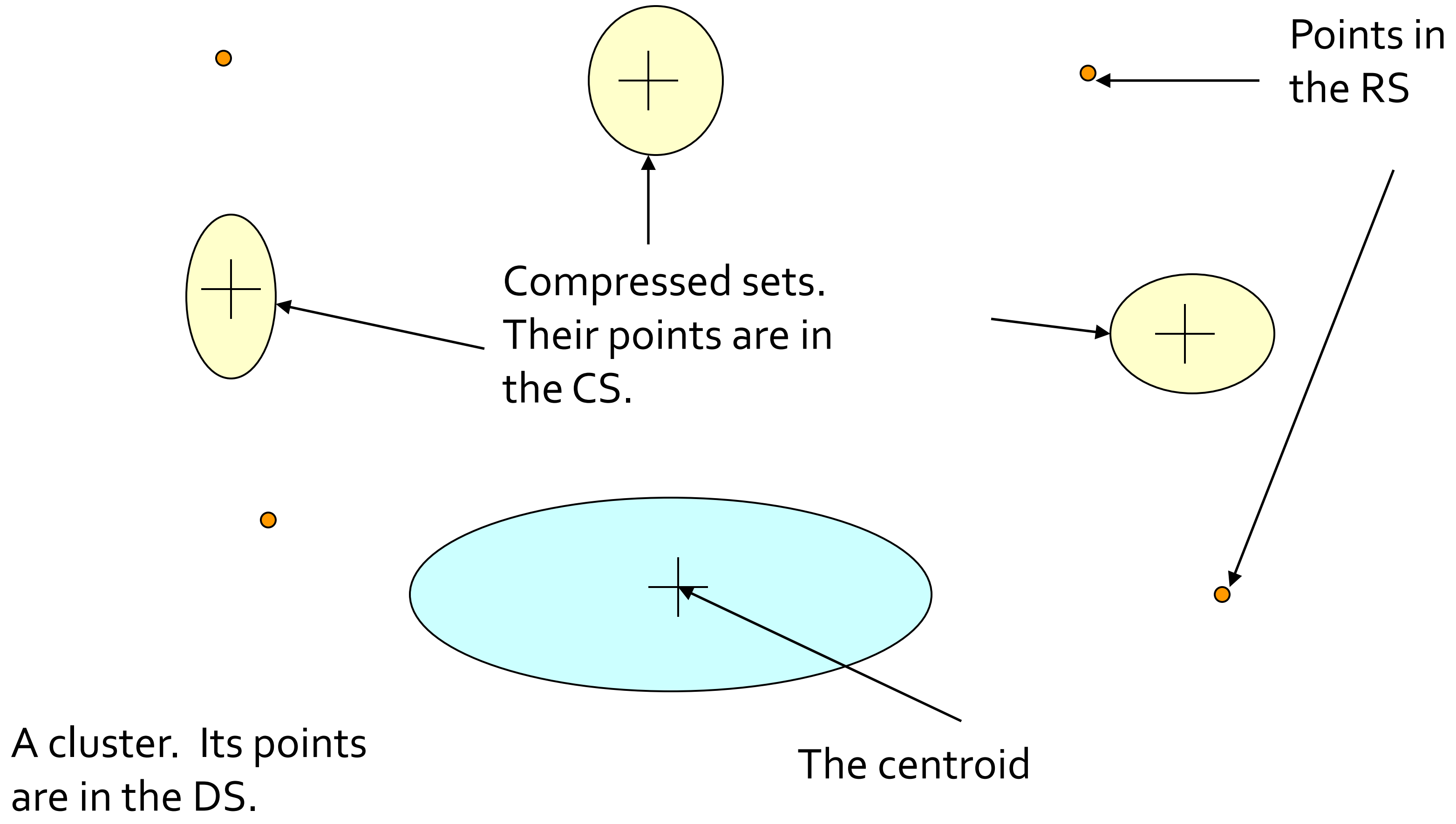
- Possibilities include:
  1. Take a small random sample and cluster optimally

  2. 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

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

- For each cluster, the discard 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 size cluster.
  - d = number of dimensions.
- Averages in each dimension (centroid) can be calculated as SUM_i / N.
  - $\text{SUM}_i = i^{\text{th}}$ component of SUM.
- Variance of a cluster's discard set in dimension i is: $(\text{SUMSQ}_i / N) - (\text{SUM}_i / N)^2$
  - And standard deviation is the square root of that.
- Q: Why use this representation of clusters?

Processing "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.
   - Add N's, SUM's, SUMSQ's.
4. Consider merging compressed sets in the CS.
5. 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

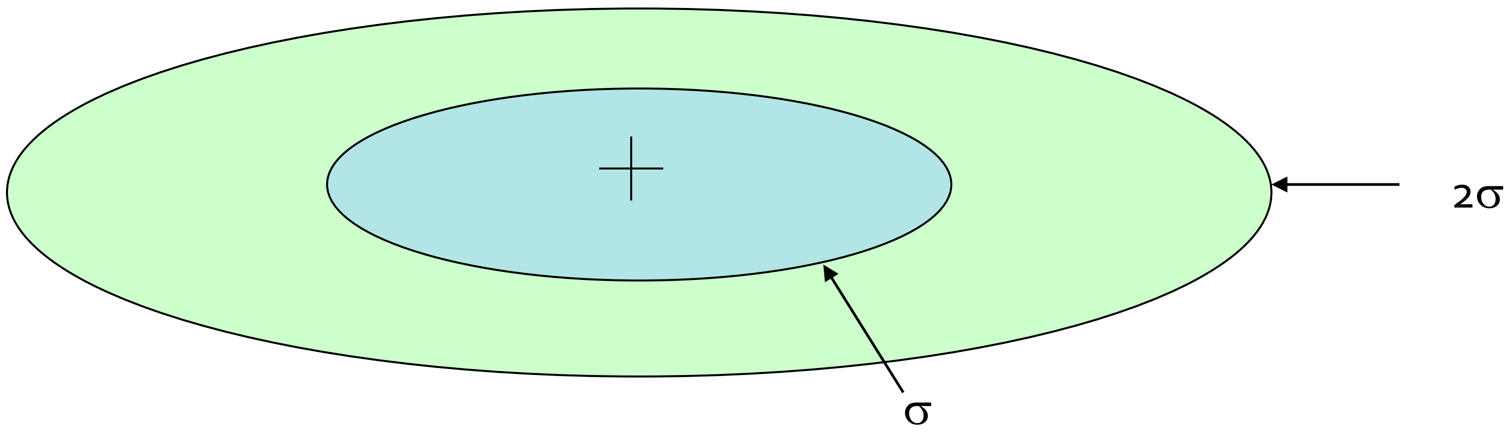
- Normalized Euclidean distance from centroid.

- For point $(x_1,\ldots,x_k)$ and centroid $(c_1,\ldots,c_k)$:
  1. Normalize in each dimension: $y_i = (x_i - c_i)/\sigma_i$
  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 2 CS subclusters be combined?

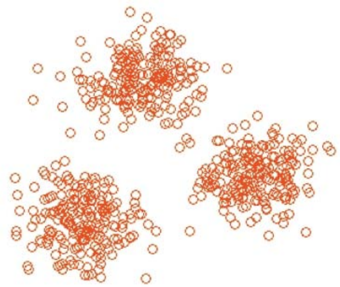
- Compute the variance of the combined subcluster.
  - *N*, SUM, and SUMSQ allow us to make that calculation quickly.

- Combine if the variance is below some threshold.

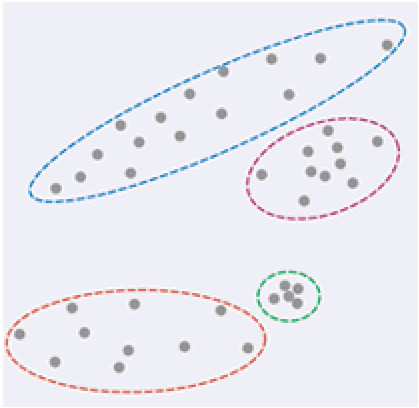
- Many alternatives: treat dimensions differently, consider density.

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.

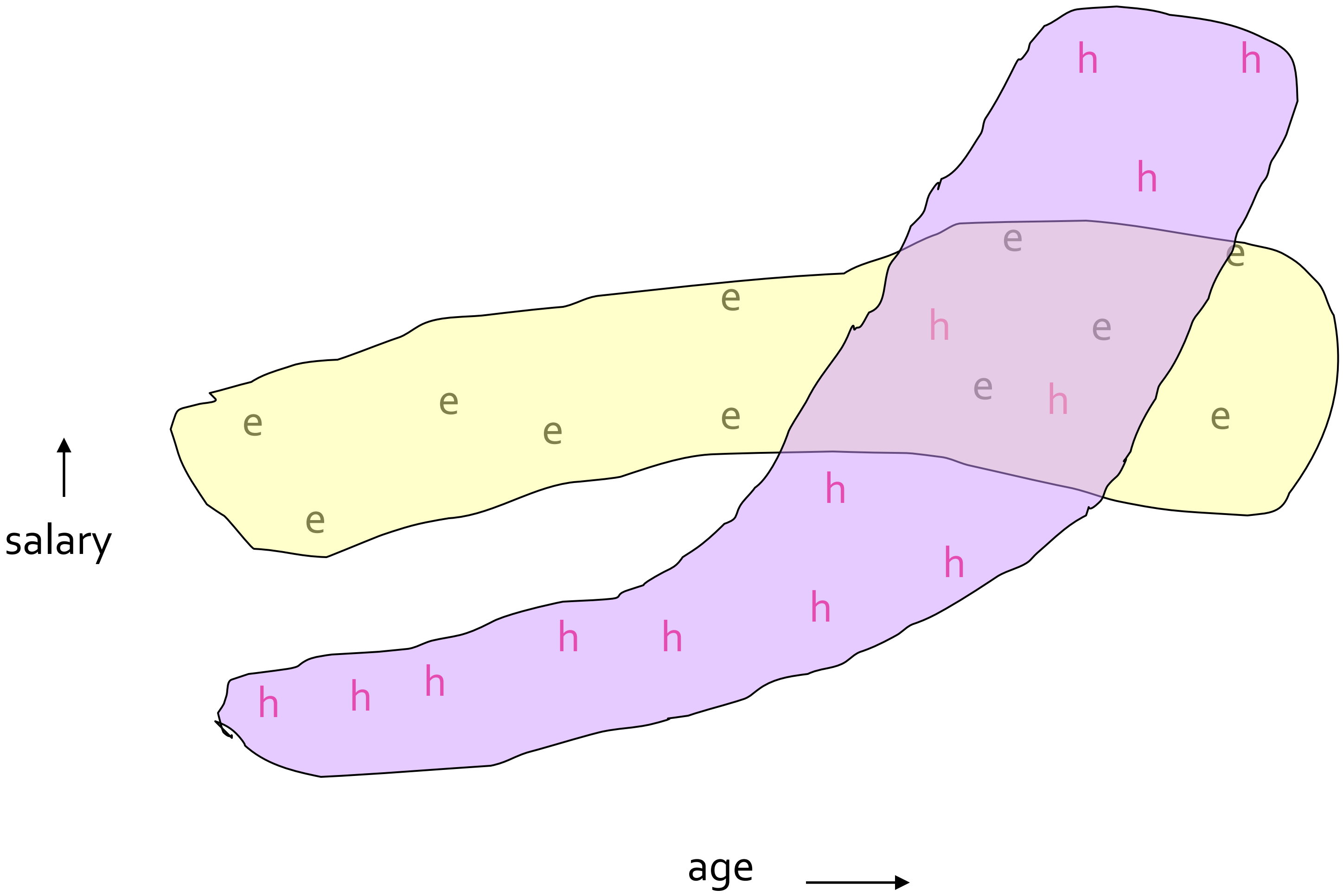


Vs.



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

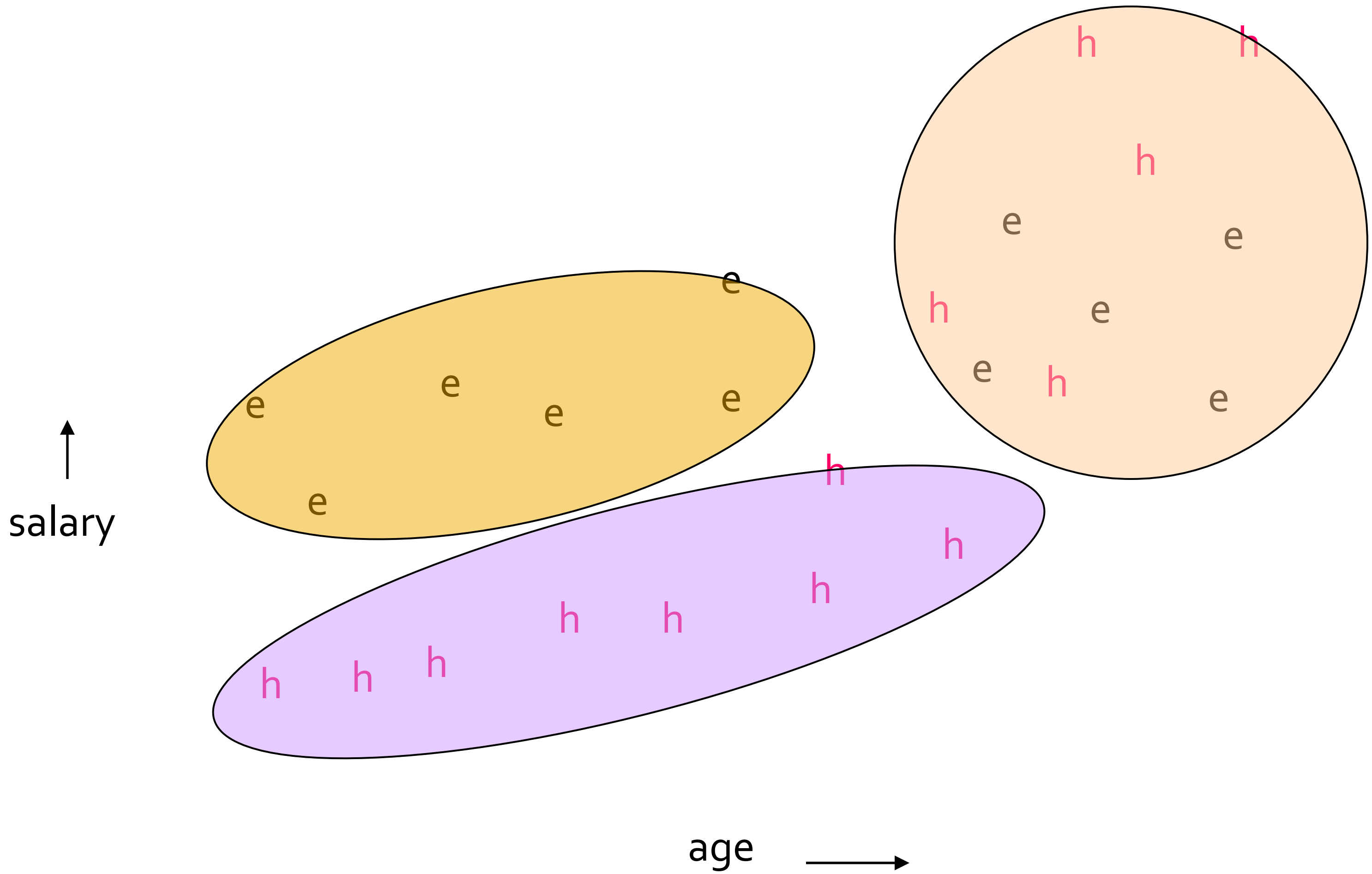
Example: Stanford Faculty Salaries



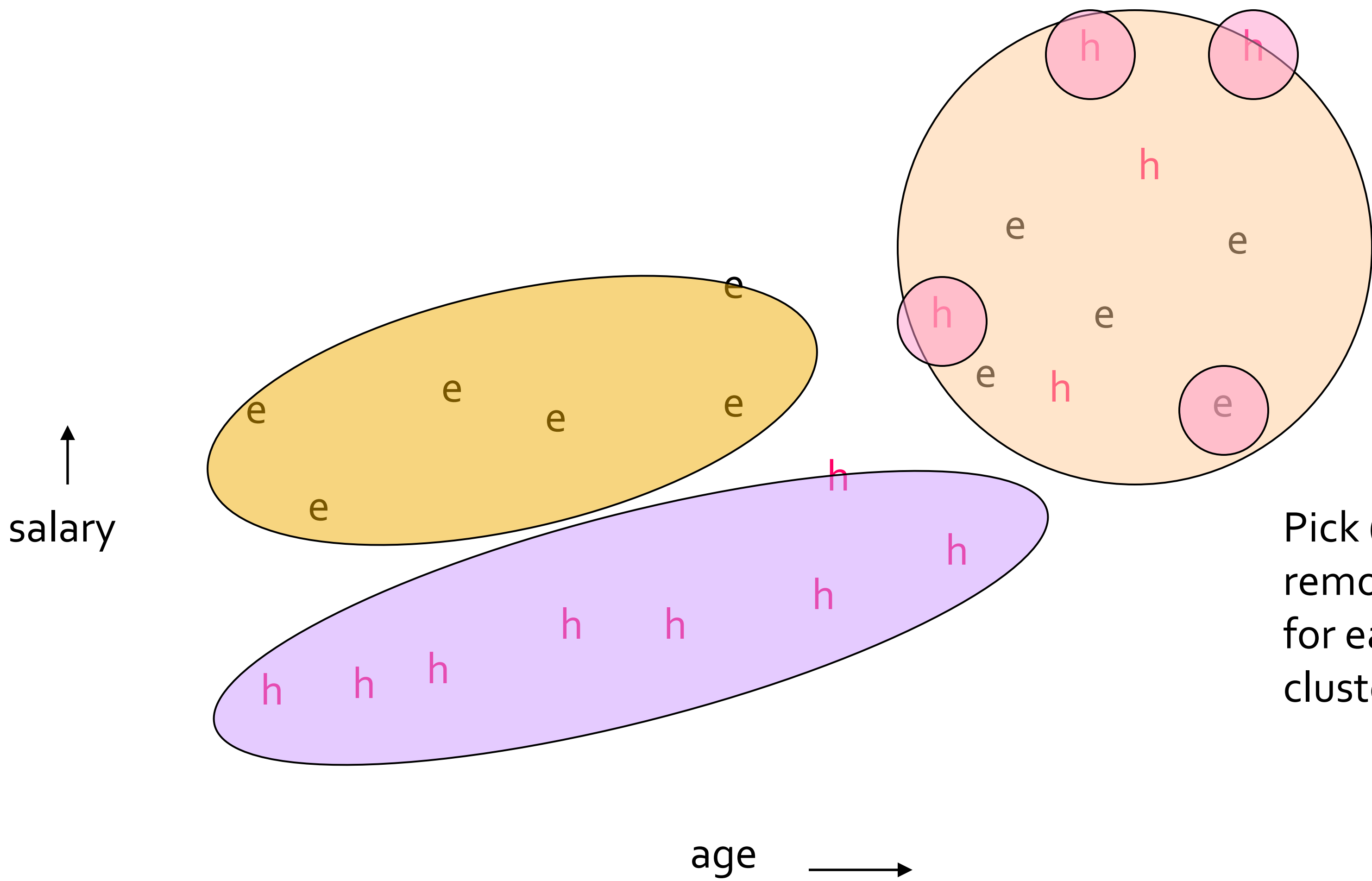
Starting CURE

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

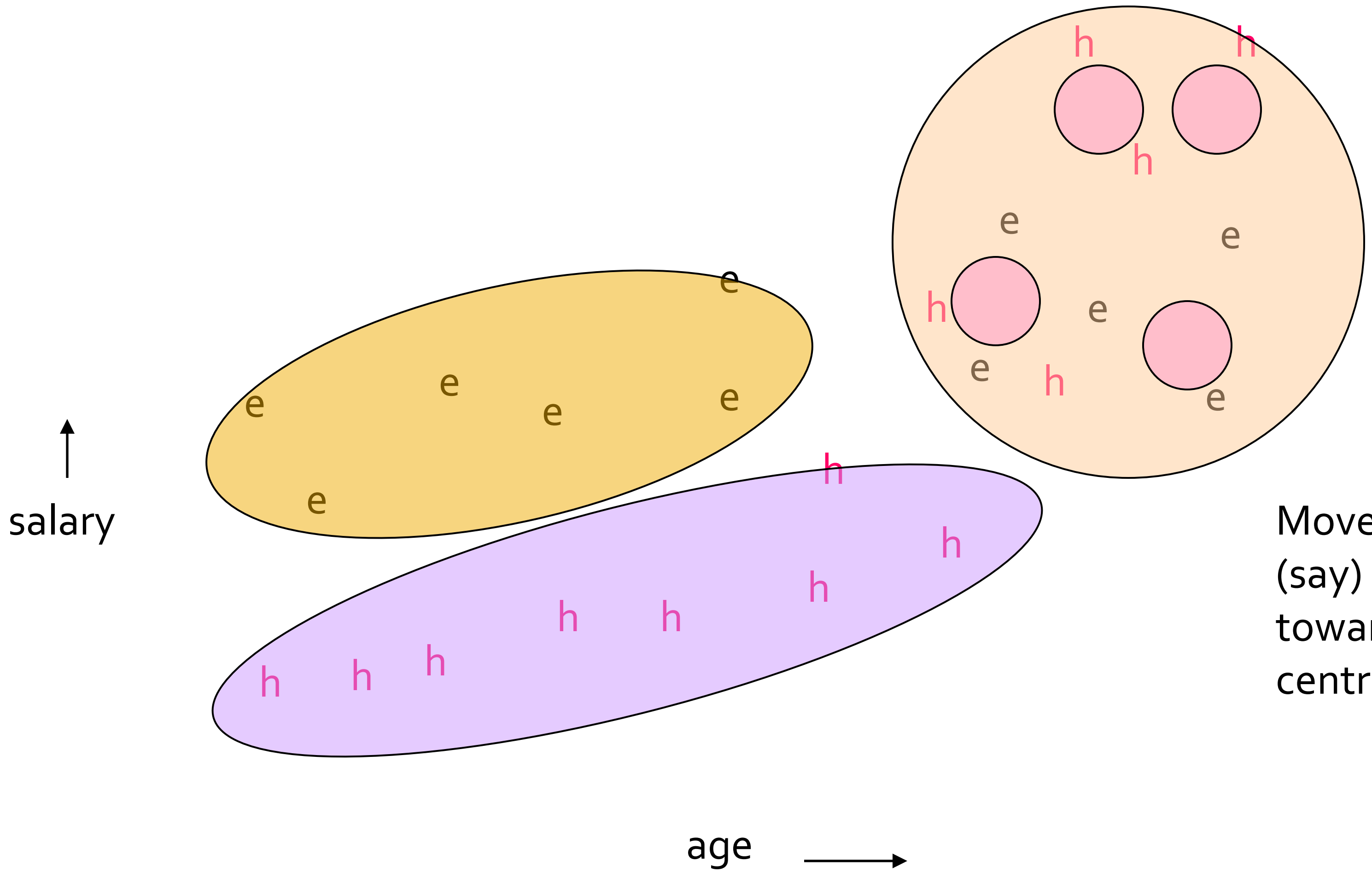


Example: Pick Dispersed Points



Pick (say) 4 remote points for each cluster.

Example: Pick Dispersed Points



Move points (say) 20% toward the centroid.

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.