Clustering

Distance Measures

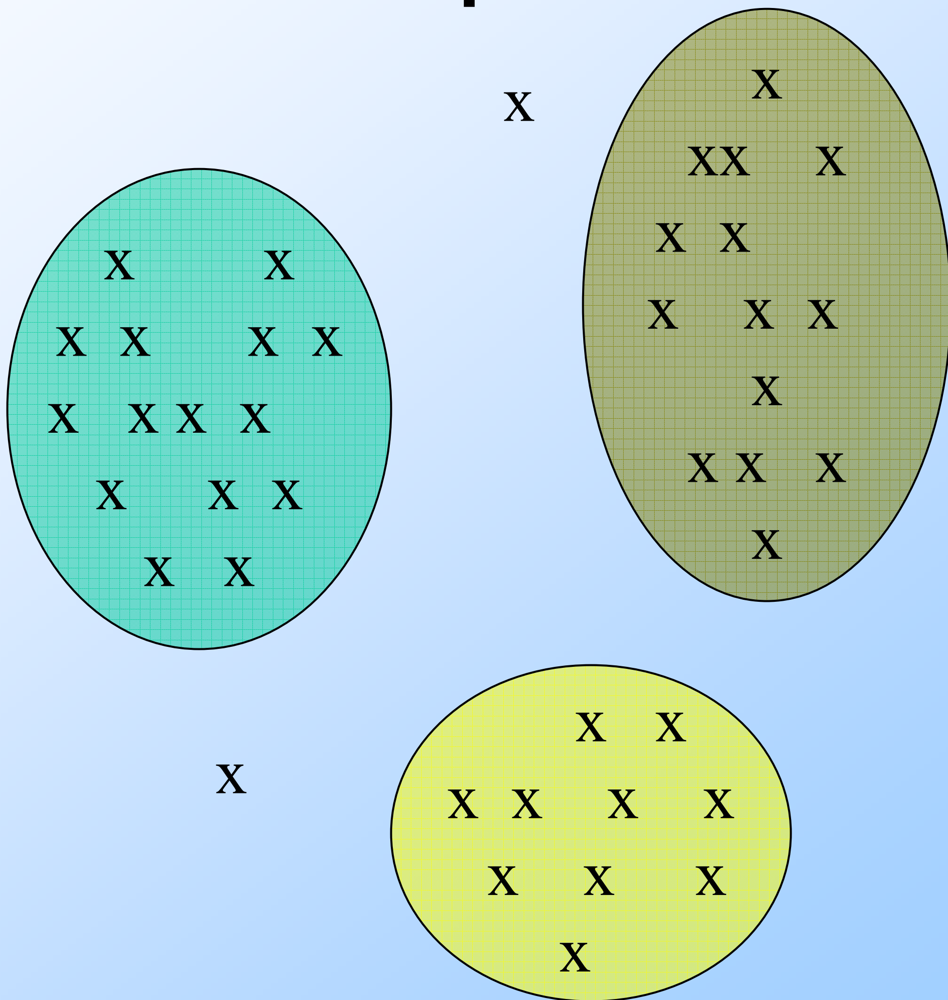
Hierarchical Clustering

k -Means Algorithms

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 in some sense as close to each other as possible.

Example



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

Example: SkyCat

- A catalog of 2 billion "sky objects" represented objects by their radiation in 9 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 (Collaborative Filtering)

- 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.

The Space of CD's

- 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 the "correlated items" matrix: rows = customers; cols. = CD's.

Example: Clustering Documents

- 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.

Example: Protein 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."

Distance Measures

- Each clustering problem is based on some kind of “distance” between points.
- Two major classes of distance measure:
  1. *Euclidean*
  2. *Non-Euclidean*

Euclidean Vs. Non-Euclidean

- A *Euclidean space* has some number of real-valued dimensions and "dense" points.
  - There is a notion of "average" of two points.
  - A *Euclidean distance* is based on the locations of points in such a space.
- A *Non-Euclidean distance* is based on properties of points, but not their "location" in a space.

Axioms of a Distance Measure

- *d* is a *distance measure* if it is a function from pairs of points to reals such that:
  1. $d(x,y) \geq 0$.
  2. $d(x,y) = 0$ iff $x = y$.
  3. $d(x,y) = d(y,x)$.
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).

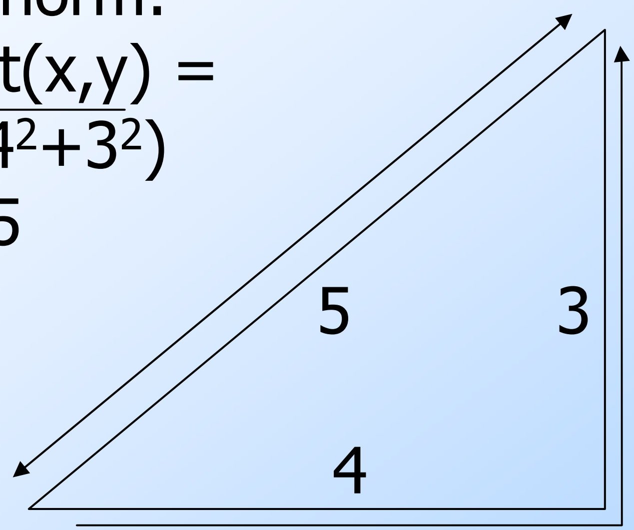
Some Euclidean Distances

- *L_2 norm* : d(x,y) = square root of the sum of the squares of the differences between *x* and *y* in each dimension.
  - The most common notion of "distance."
- *L_1 norm* : sum of the differences in each dimension.
  - *Manhattan distance* = distance if you had to travel along coordinates only.

Examples of Euclidean Distances

$y = (9,8)$

L_2-norm:
$dist(x,y) =$
$\sqrt{(4^2+3^2)}$
$= 5$



$x = (5,5)$

L_1-norm:
$dist(x,y) =$
$4+3 = 7$

Another Euclidean Distance

- *L_∞ norm* : d(x,y) = the maximum of the differences between *x* and *y* in any dimension.
- Note: the maximum is the limit as *n* goes to ∞ of what you get by taking the n^{th} power of the differences, summing and taking the n^{th} root.

Non-Euclidean Distances

- *Jaccard distance* for sets = 1 minus ratio of sizes of intersection and union.
- *Cosine distance* = angle between vectors from the origin to the points in question.
- *Edit distance* = number of inserts and deletes to change one string into another.

Jaccard Distance

- Example: p_1 = 10111; p_2 = 10011.
  - Size of intersection = 3; size of union = 4, Jaccard measure (not distance) = 3/4.
- Need to make a distance function satisfying triangle inequality and other laws.
- d(x,y) = 1 – (Jaccard measure) works.

Why J.D. Is a Distance Measure

- $d(x,x) = 0$ because $x \cap x = x \cup x$.
- $d(x,y) = d(y,x)$ because union and intersection are symmetric.
- $d(x,y) \geq 0$ because $|x \cap y| \leq |x \cup y|$.
- $d(x,y) \leq d(x,z) + d(z,y)$ trickier --- next slide.

Triangle Inequality for J.D.

$$1 - \frac{|x \cap z|}{|x \cup z|} + 1 - \frac{|y \cap z|}{|y \cup z|} \geq 1 - \frac{|x \cap y|}{|x \cup y|}$$

- Remember: $|a \cap b|/|a \cup b|$ = probability that minhash(a) = minhash(b).
- Thus, $1 - |a \cap b|/|a \cup b|$ = probability that minhash(a) $\neq$ minhash(b).

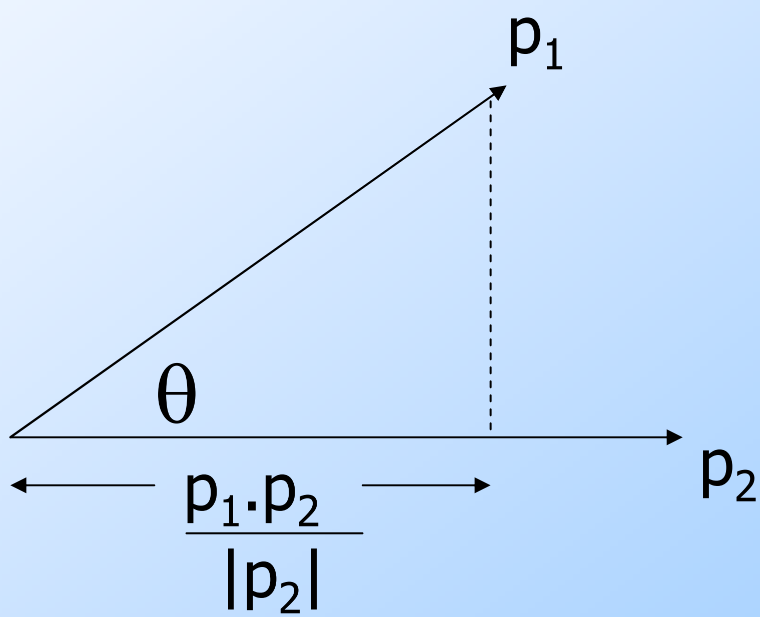
Triangle Inequality --- (2)

- So we need to observe that prob[minhash(x) $\neq$ minhash(y)] $\leq$ prob[minhash(x) $\neq$ minhash(z)] + prob[minhash(z) $\neq$ minhash(y)]
- Clincher: whenever minhash(x) $\neq$ minhash(y), one of minhash(x) $\neq$ minhash(z) and minhash(z) $\neq$ minhash(y) must be true.

Cosine Distance

- Think of a point as a vector from the origin (0,0,…,0) to its location.
- Two points' vectors make an angle, whose cosine is the normalized dot-product of the vectors: $p_1.p_2/|p_2||p_1|$.
  - Example $p_1 = 00111$; $p_2 = 10011$.
  - $p_1.p_2 = 2$; $|p_1| = |p_2| = \sqrt{3}$.
  - $\cos(\theta) = 2/3$; θ is about 48 degrees.

Cosine-Measure Diagram

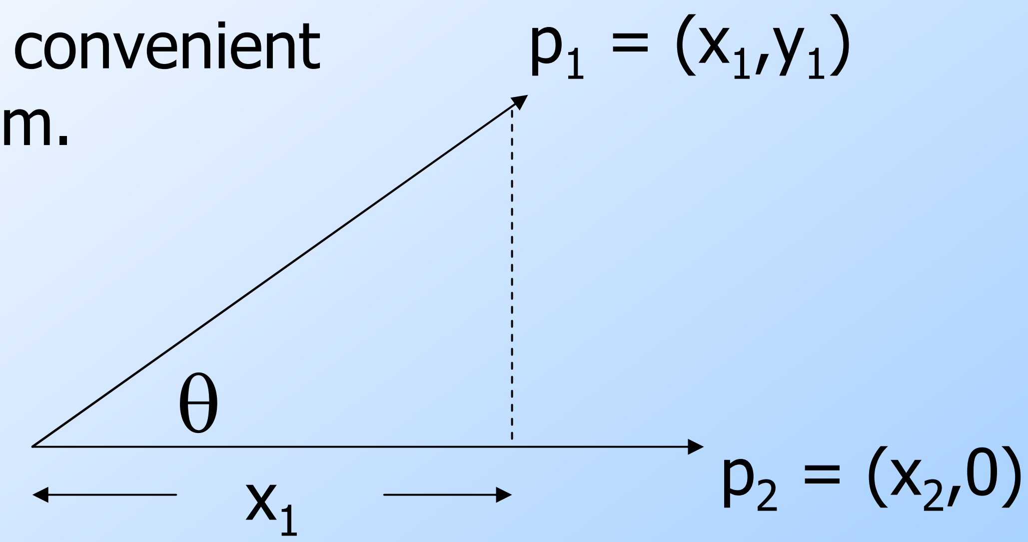


$$\text{dist}(p_1, p_2) = \theta = \arccos(p_1.p_2/|p_2||p_1|)$$

Why?

Dot product is invariant under rotation, so pick convenient coordinate system.

$p_1.p_2 = x_1 * x_2.$
$|p_2| = x_2.$



$x_1 = p_1.p_2/|p_2|$

Why C.D. Is a Distance Measure

- d(x,x) = 0 because arccos(1) = 0.
- d(x,y) = d(y,x) by symmetry.
- d(x,y) $\geq$ 0 because angles are chosen to be in the range 0 to 180 degrees.
- Triangle inequality: physical reasoning. If I rotate an angle from *x* to *z* and then from *z* to *y*, I can't rotate less than from *x* to *y*.

Edit Distance

- The edit distance of two strings is the number of inserts and deletes of characters needed to turn one into the other.
- Equivalently, $d(x,y) =$ $|x| + |y| -2|LCS(x,y)|$.
  - LCS = *longest common subsequence* = longest string obtained both by deleting from *x* and deleting from *y*.

Example

- x = *abcde* ; y = *bcduve*.
- Turn x into y by deleting *a*, then inserting *u* and *v* after *d*.
  - Edit-distance = 3.
- Or, LCS(x,y) = *bcde*.
- |x| + |y| - 2|LCS(x,y)| = 5 + 6 –2*4 = 3.

Why E.D. Is a Distance Measure

- $d(x,x) = 0$ because 0 edits suffice.
- $d(x,y) = d(y,x)$ because insert/delete are inverses of each other.
- $d(x,y) \geq 0$: no notion of negative edits.
- Triangle inequality: changing x to z and then to y is one way to change x to y.

Variant Edit Distance

- Allow insert, delete, and *mutate*.
  - Change one character into another.
- Minimum number of inserts, deletes, and mutates also forms a distance measure.

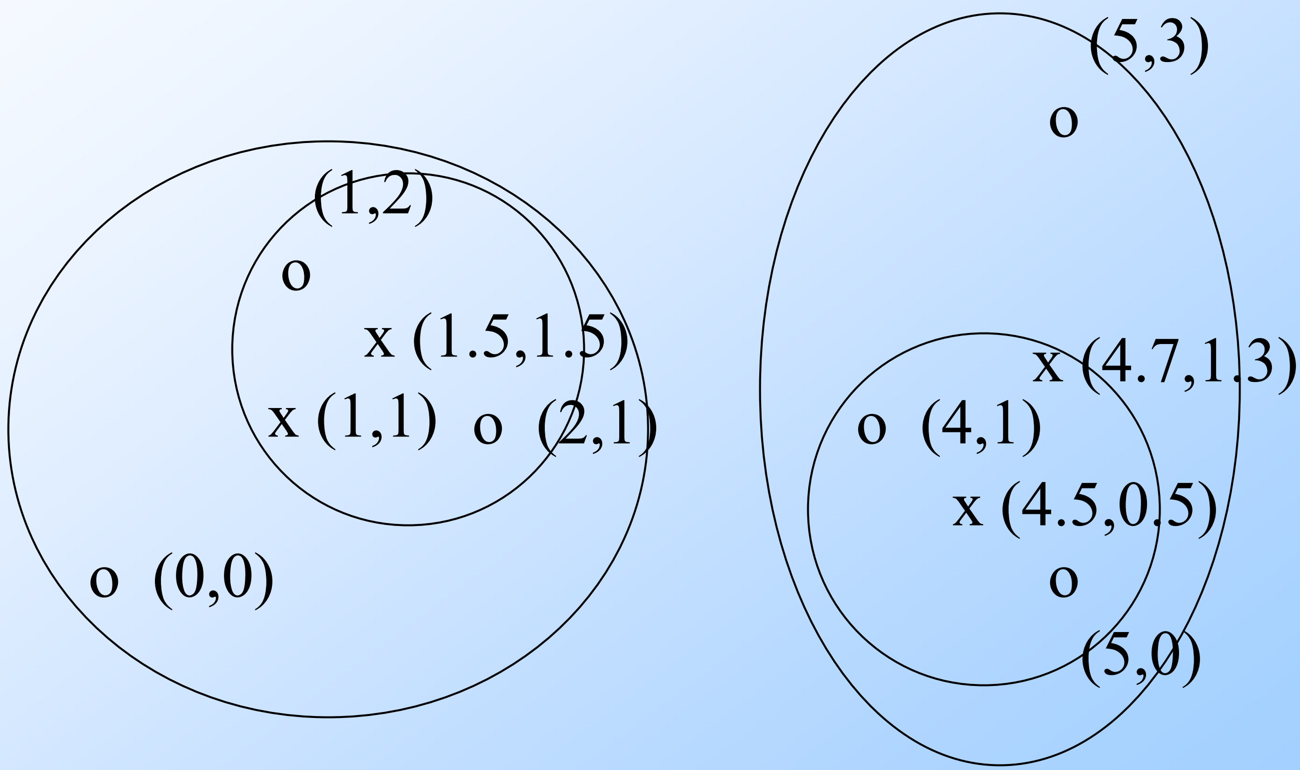
Methods of Clustering

- Hierarchical:
  - Initially, each point in cluster by itself.
  - Repeatedly combine the two “closest” clusters into one.
- Point Assignment:
  - Maintain a set of clusters.
  - Place points into “closest” cluster.

Hierarchical Clustering

- 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



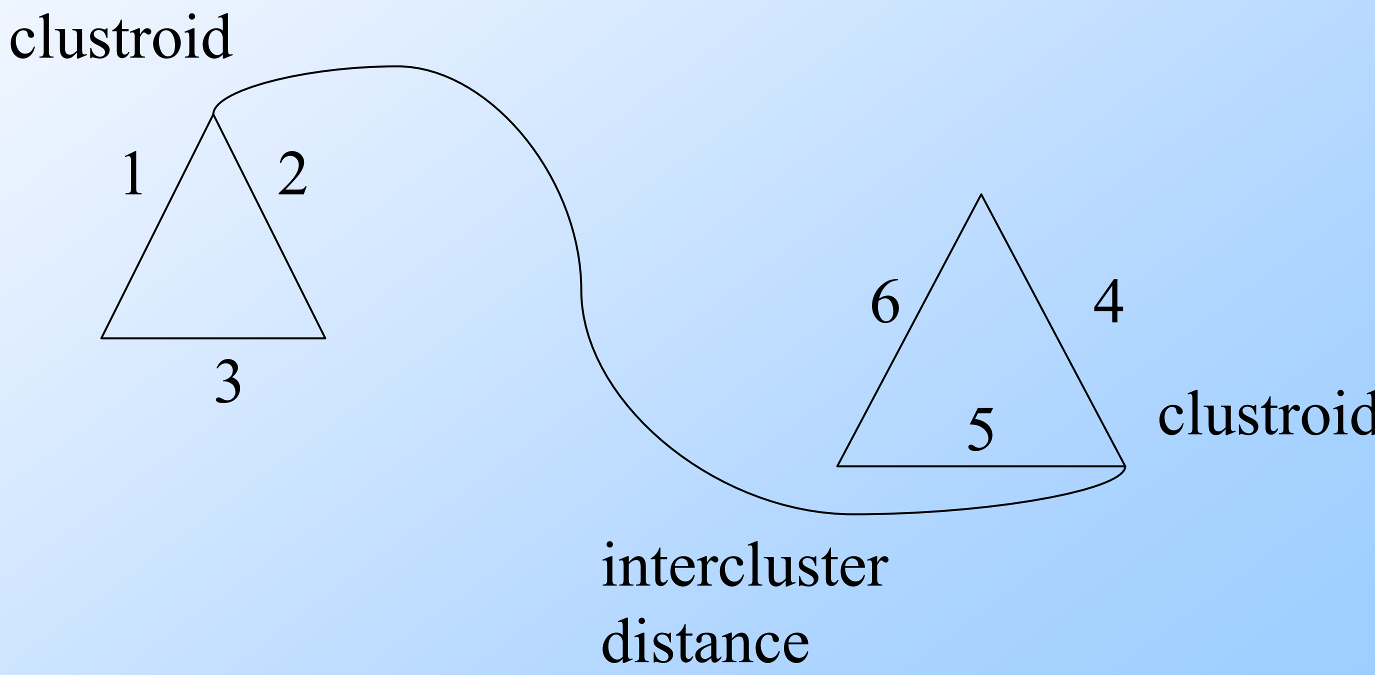
And in the Non-Euclidean Case?

- 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”?

- 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.

Example



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 clustroid.
  - Merge clusters whose union is most cohesive.

k –Means Algorithm(s)

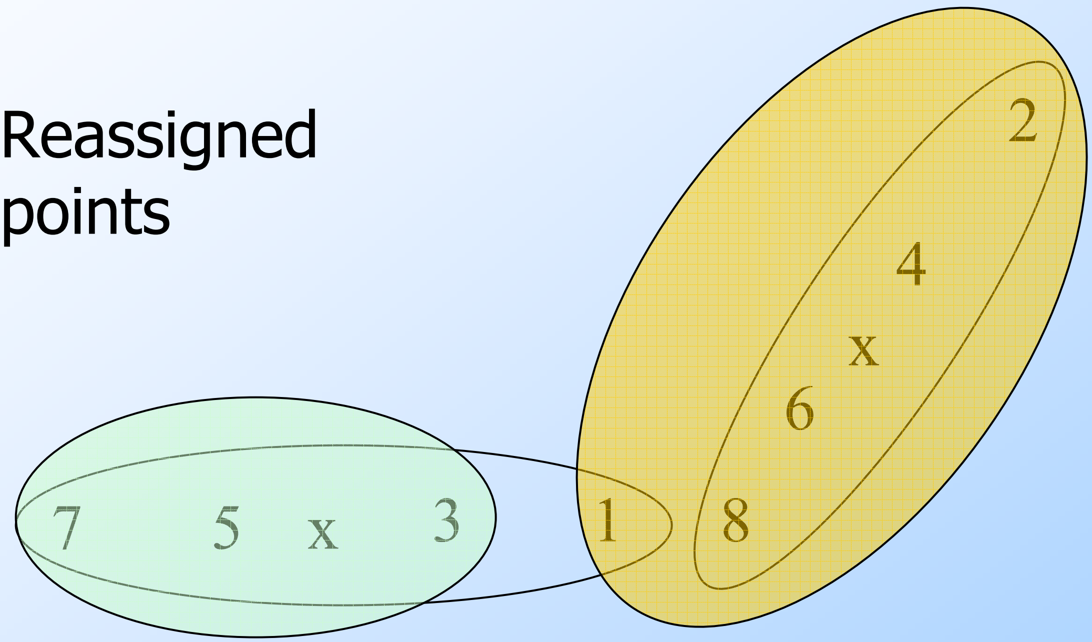
- Assumes Euclidean space.
- Start by picking k, the number of clusters.
- Initialize clusters by picking one point per cluster.
  - For instance, pick one point at random, then k -1 other points, each as far away as possible from the previous points.

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. Reassign all points to their closest centroid.
   - Sometimes moves points between clusters.

Example

Reassigned points

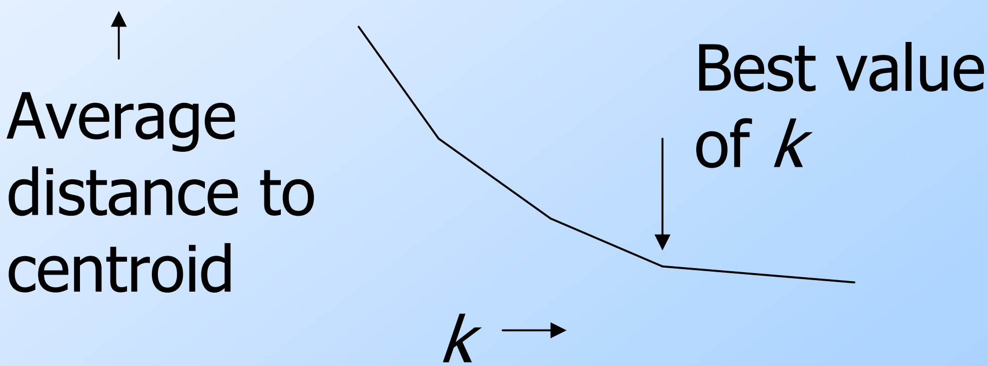


Clusters after first round

Getting *k* Right

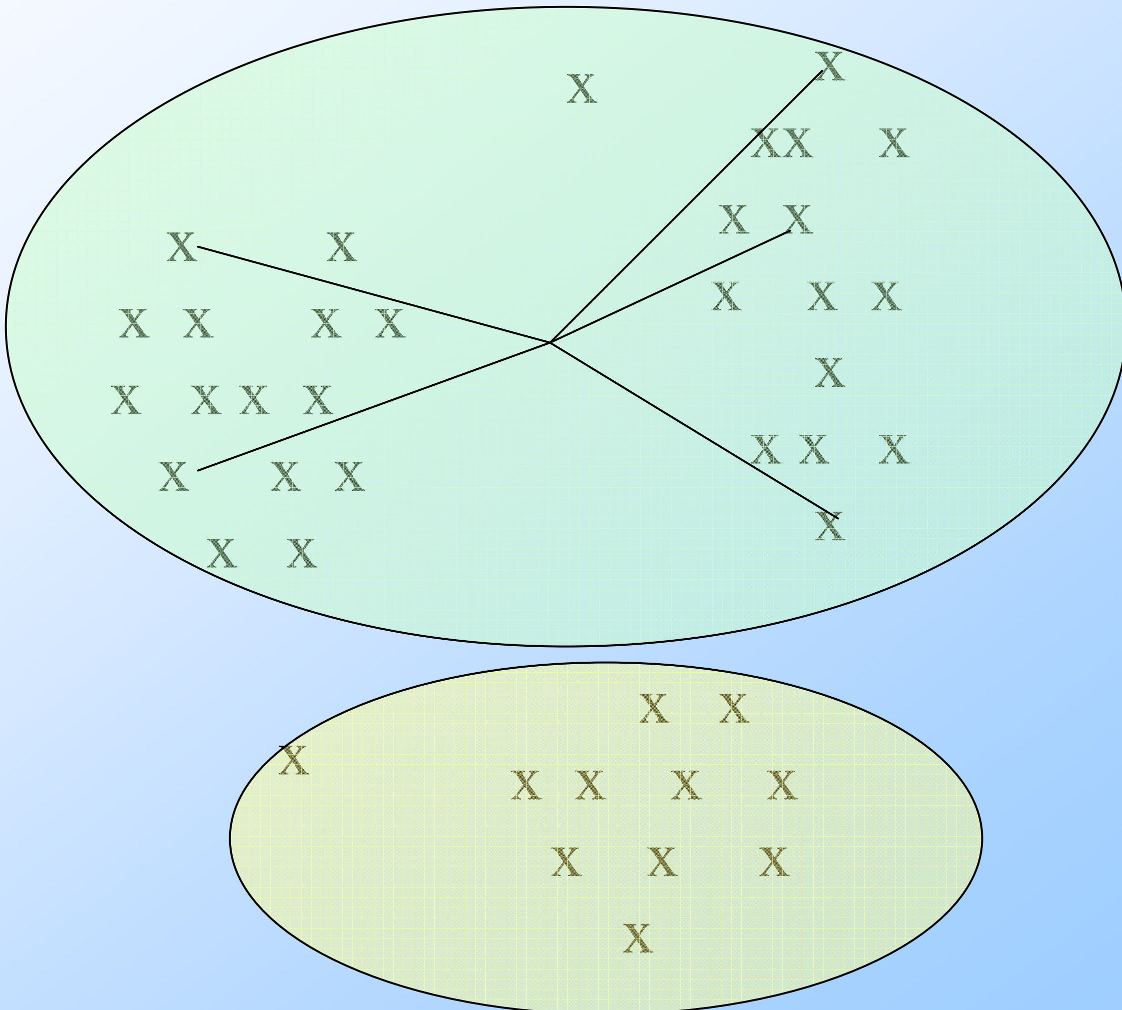
- Try different *k*, looking at the change in the average distance to centroid, as *k* increases.

- Average falls rapidly until right *k*, then changes little.



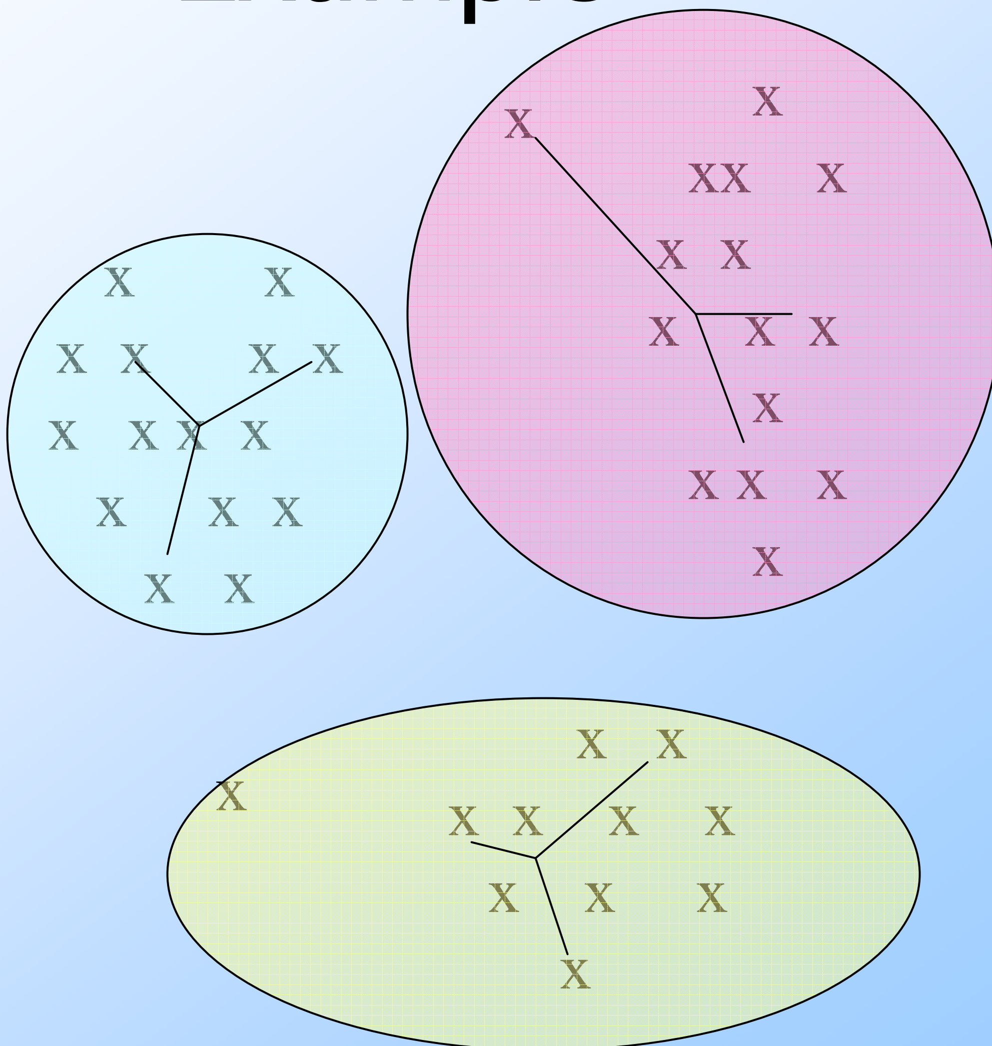
Example

Too few;
many long
distances
to centroid.



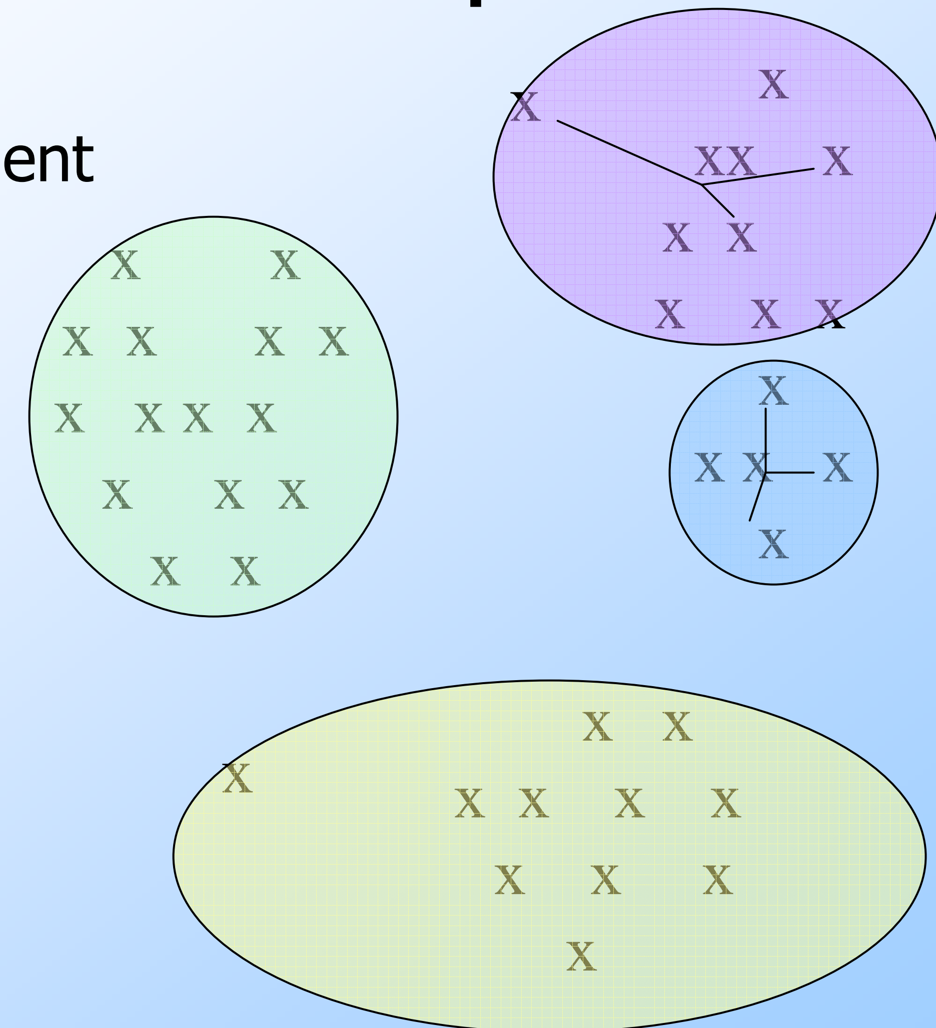
Example

Just right;
distances
rather short.



Example

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 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* : points close enough to a centroid to be represented statistically.
2. The *compression set* : groups of points that are close together but not close to any centroid. They are represented statistically, but not assigned to a cluster.
3. The *retained set* : isolated points.

Representing Sets of Points

- For each cluster, the discard set is represented 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.

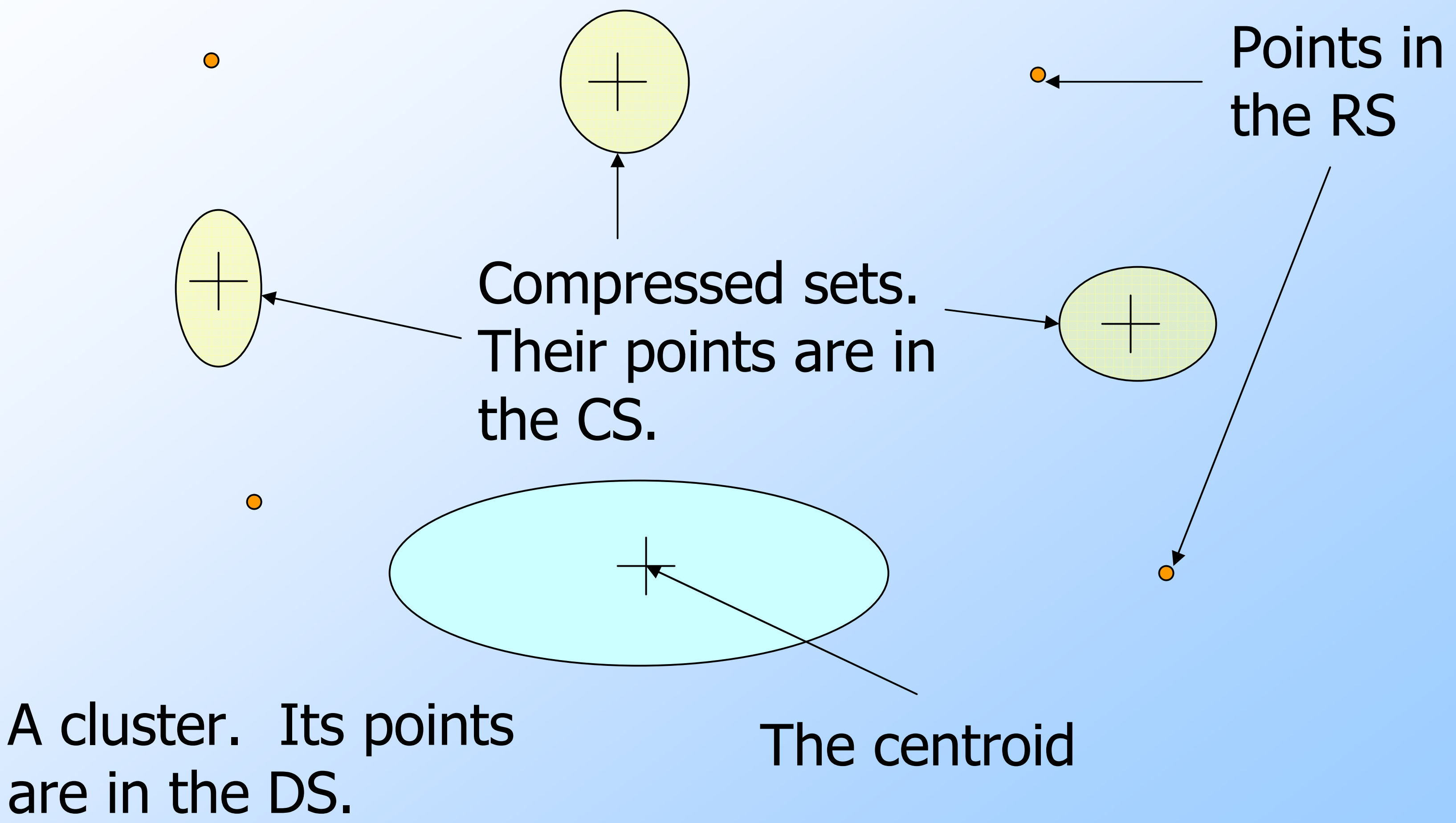
Comments

- $2d + 1$ values represent any number of points.
  - d = number of dimensions.
- Averages in each dimension (centroid coordinates) can be calculated easily as SUM_i / N.
  - $\mathrm{SUM}_i = i^{\text{th}}$ component of SUM.

Comments --- (2)

- Variance of a cluster's discard set in dimension i can be computed by:
  $$(\mathrm{SUMSQ}_i / N) - (\mathrm{SUM}_i / N)^2$$
- And the standard deviation is the square root of that.
- The same statistics can represent any compression set.

"Galaxies" Picture



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.
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.

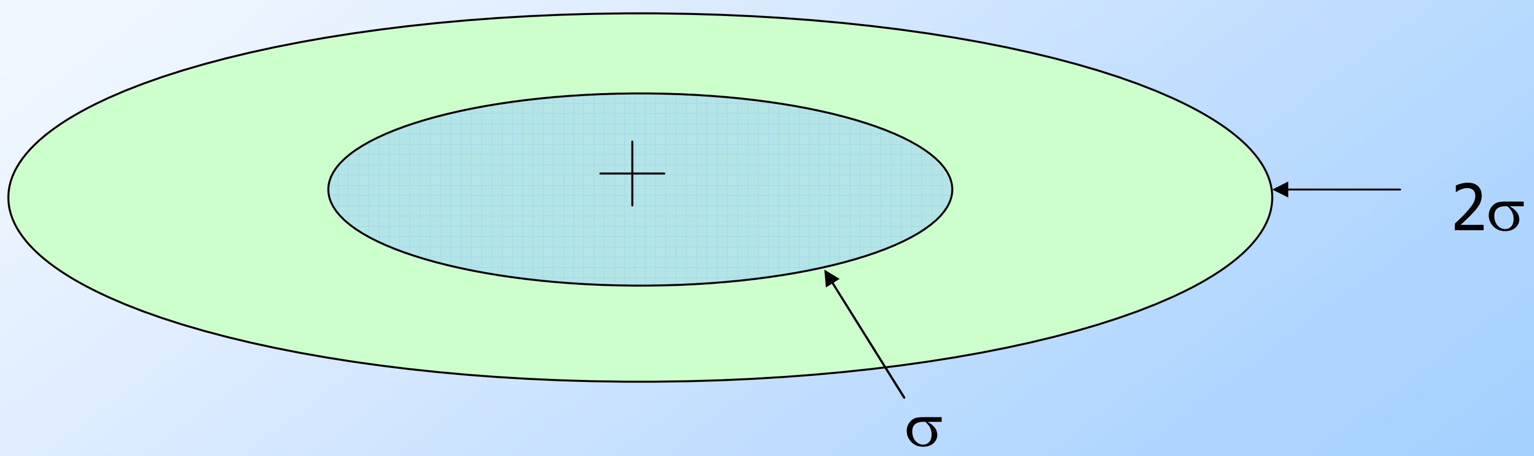
Mahalanobis Distance

- Normalized Euclidean distance.
- 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 one standard deviation corresponds to a distance $\sqrt{d}$.
  - 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?

- Compute the variance of the combined subcluster.
  - *N*, SUM, and SUMSQ allow us to make that calculation.
- Combine if the variance is below some threshold.