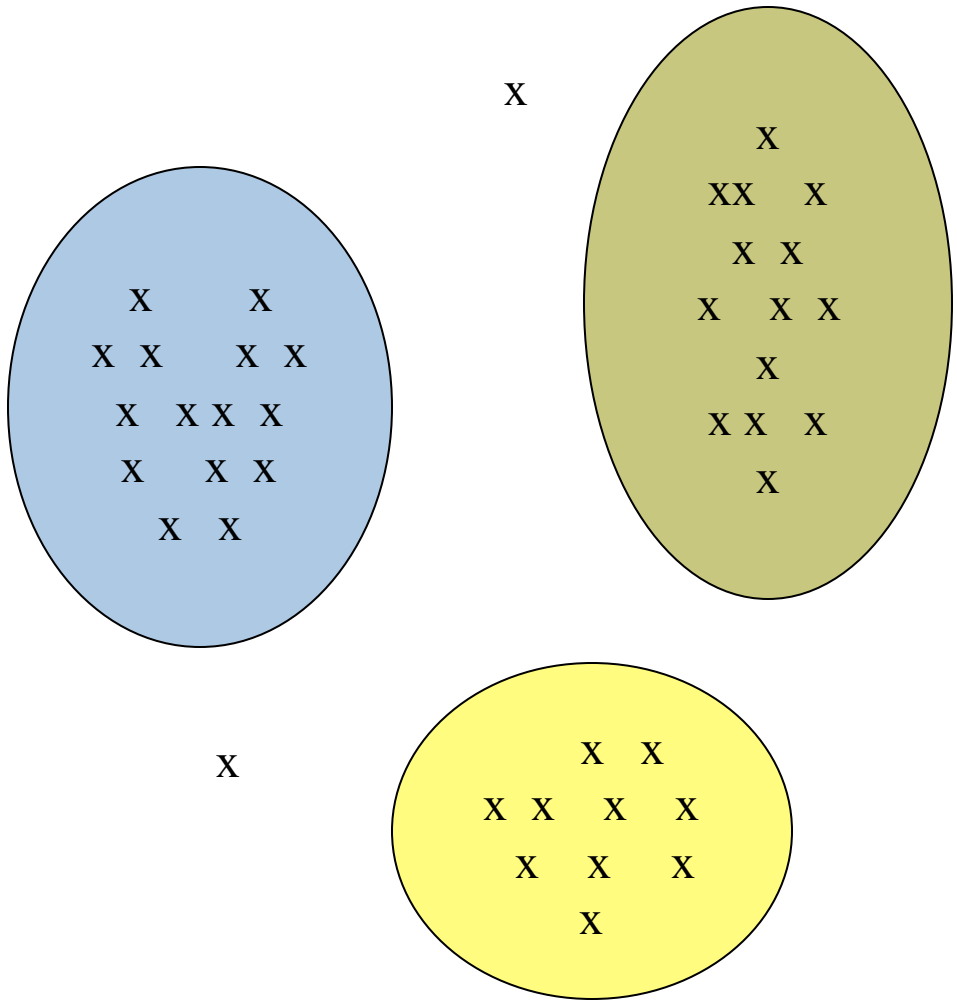
Clustering

Barna Saha

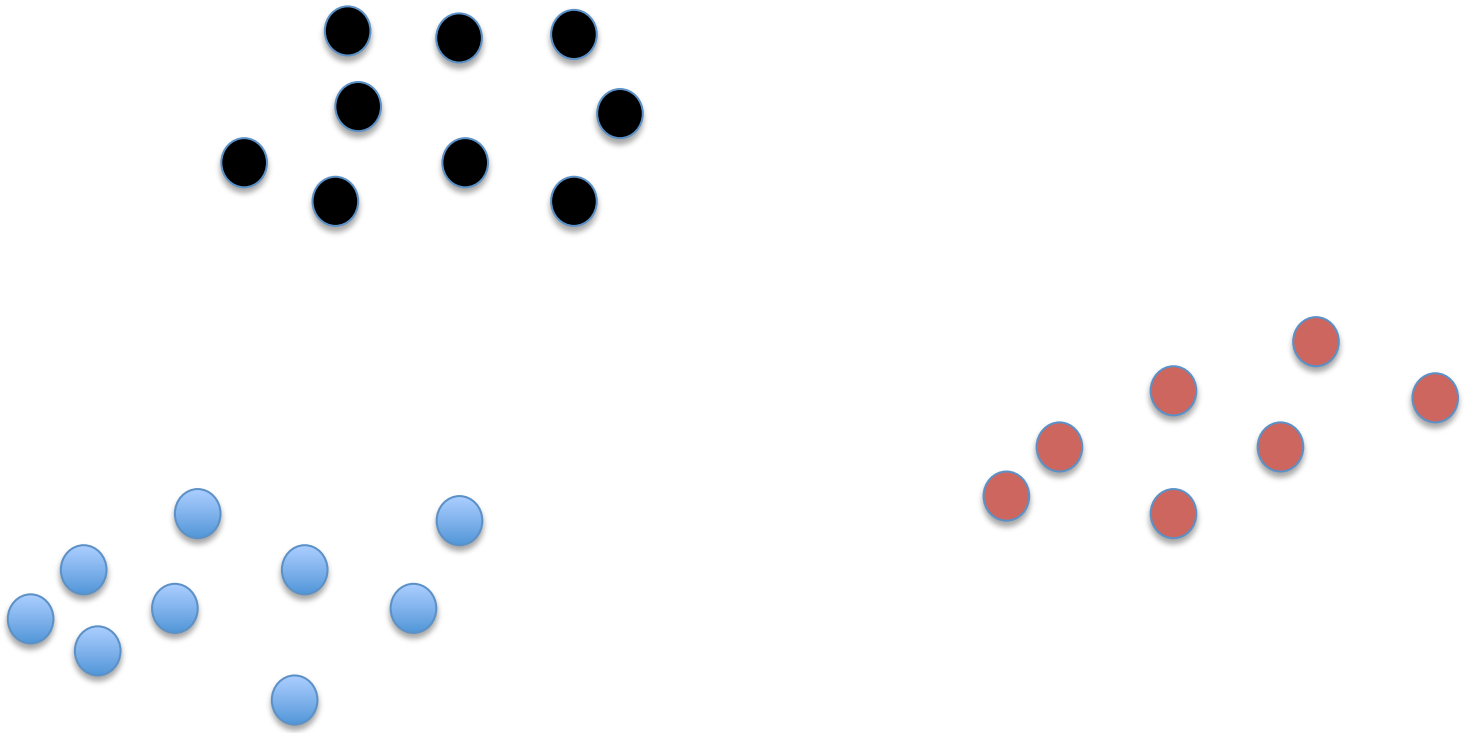
The Problem of Clustering

- 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



Clustering in Low Dimensional Euclidean Space is Easy



Modern Clustering Problem

- May involve Euclidean spaces of very high dimension.
- Non Euclidean space: Jaccard distance, Cosine Distance, Hamming Distance, Edit Distance etc.
- Example:
  - Cluster documents by topics based on occurrences of unusual words
  - Cluster moviegoers by the type or types of movies they like
  - Cluster genes by their sequence similarity

A Popular Clustering Algorithm: K Means

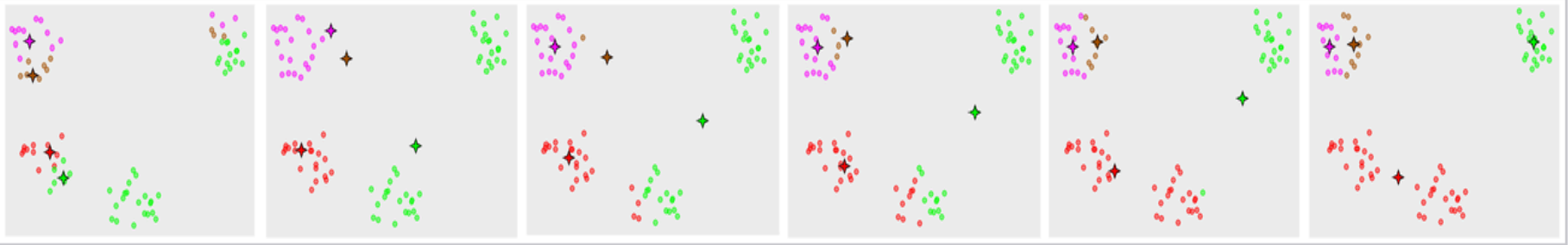
- k=number of clusters
- Given k and a set of data points in the Euclidean space select k centers so as the the sum of squared distance between each point and its nearest center is minimized.p
- Solving this problem exactly in NP Hard
- 25 years ago Llyod proposed a simple local seacrh based algorithm that is still very widely used---has polynomial time smoothed complexity.
  - However Llyod's algorithm may get stuck at a local optima
    - K Means ++

Llyod's Local Search Algorithm

1. Begin with k arbitrary centers, typically chosen uniformly at random from the data points.
2. Assign each point to its nearest cluster center
3. Recompute the new cluster centers as the center of mass of the points assigned to the clusters
4. Repeat Steps 1-3 until the process converges.

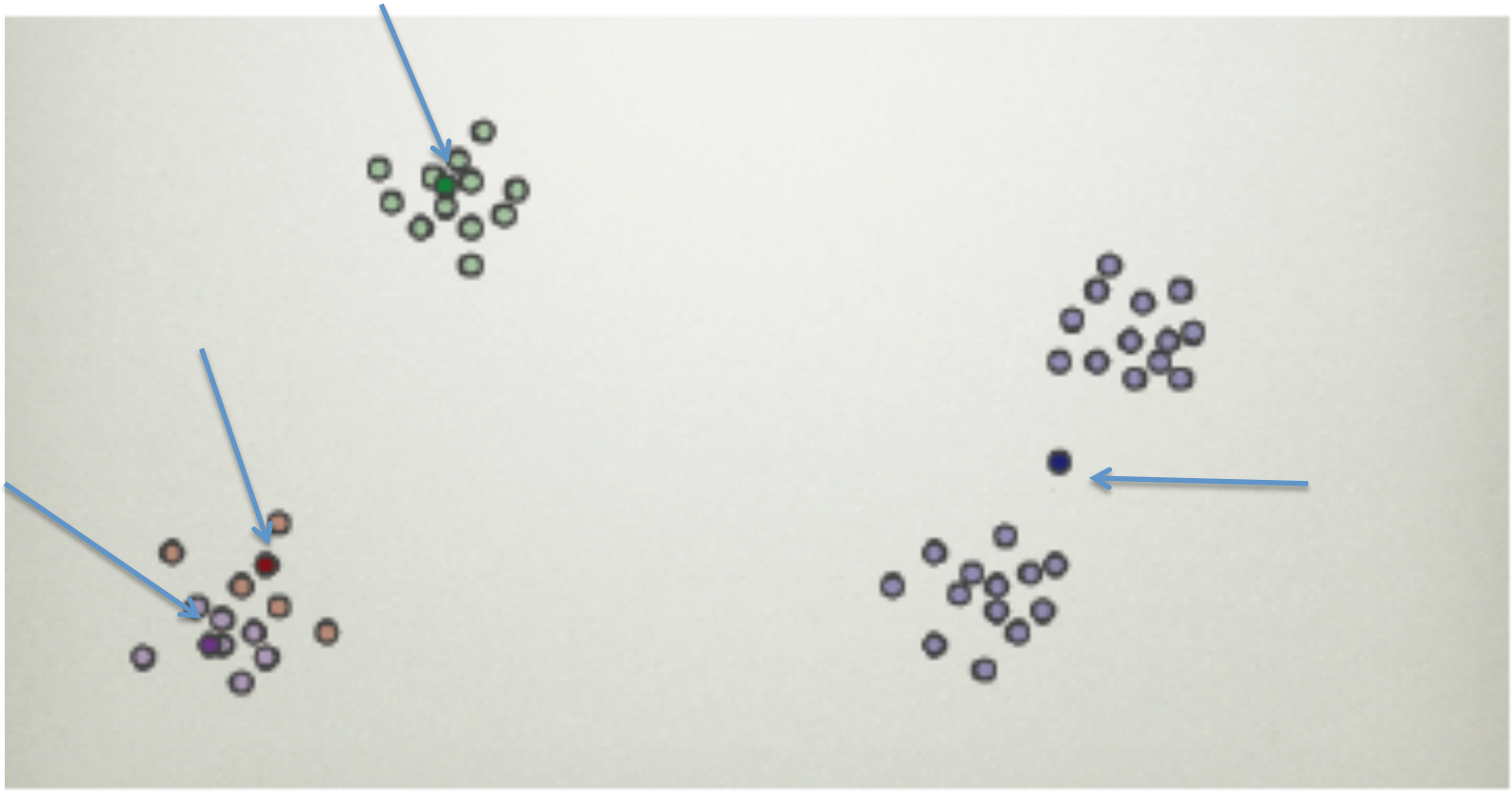
Illustration (taken from Wiki)

Convergence to local optima

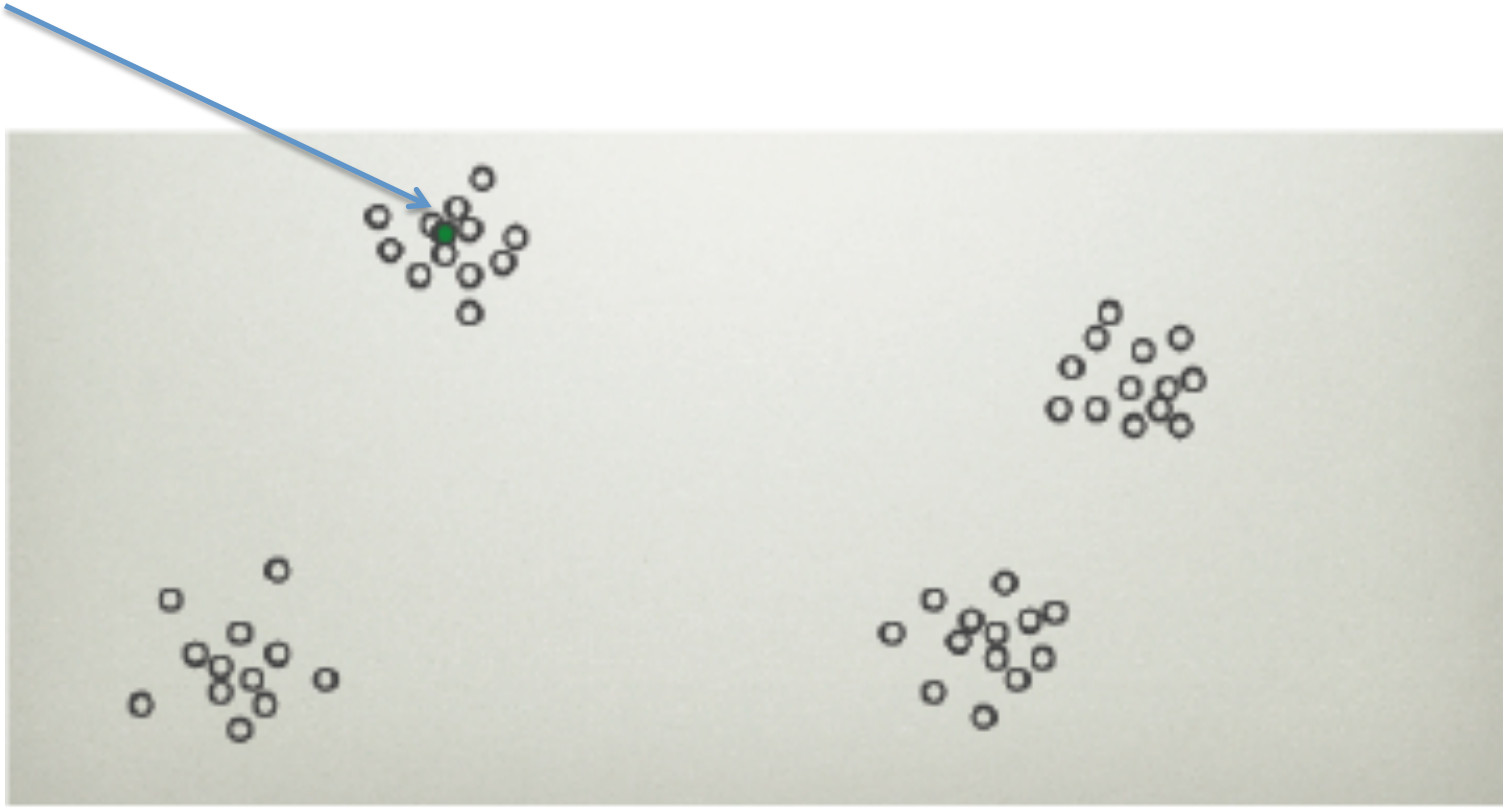


A typical example of the k-means convergence to a local minimum. In this example, the result of k-means clustering (the right figure) contradicts the obvious cluster structure of the data set. The small circles are the data points, the four ray stars are the centroids (means). The initial configuration is on the left figure. The algorithm converges after five iterations presented on the figures, from the left to the right. The illustration was prepared with the Mirkes Java applet.[36]

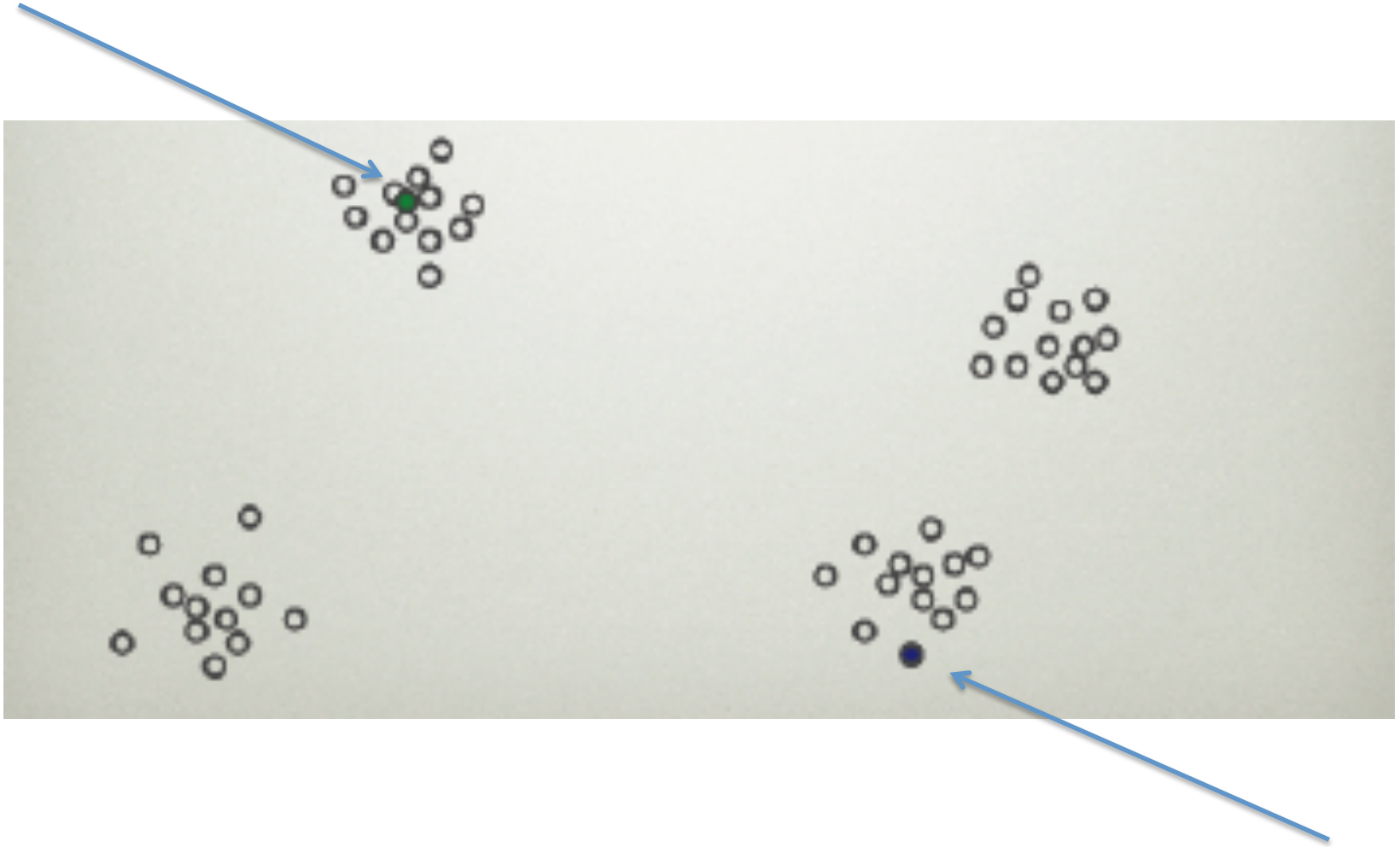
Convergence to Local Optima



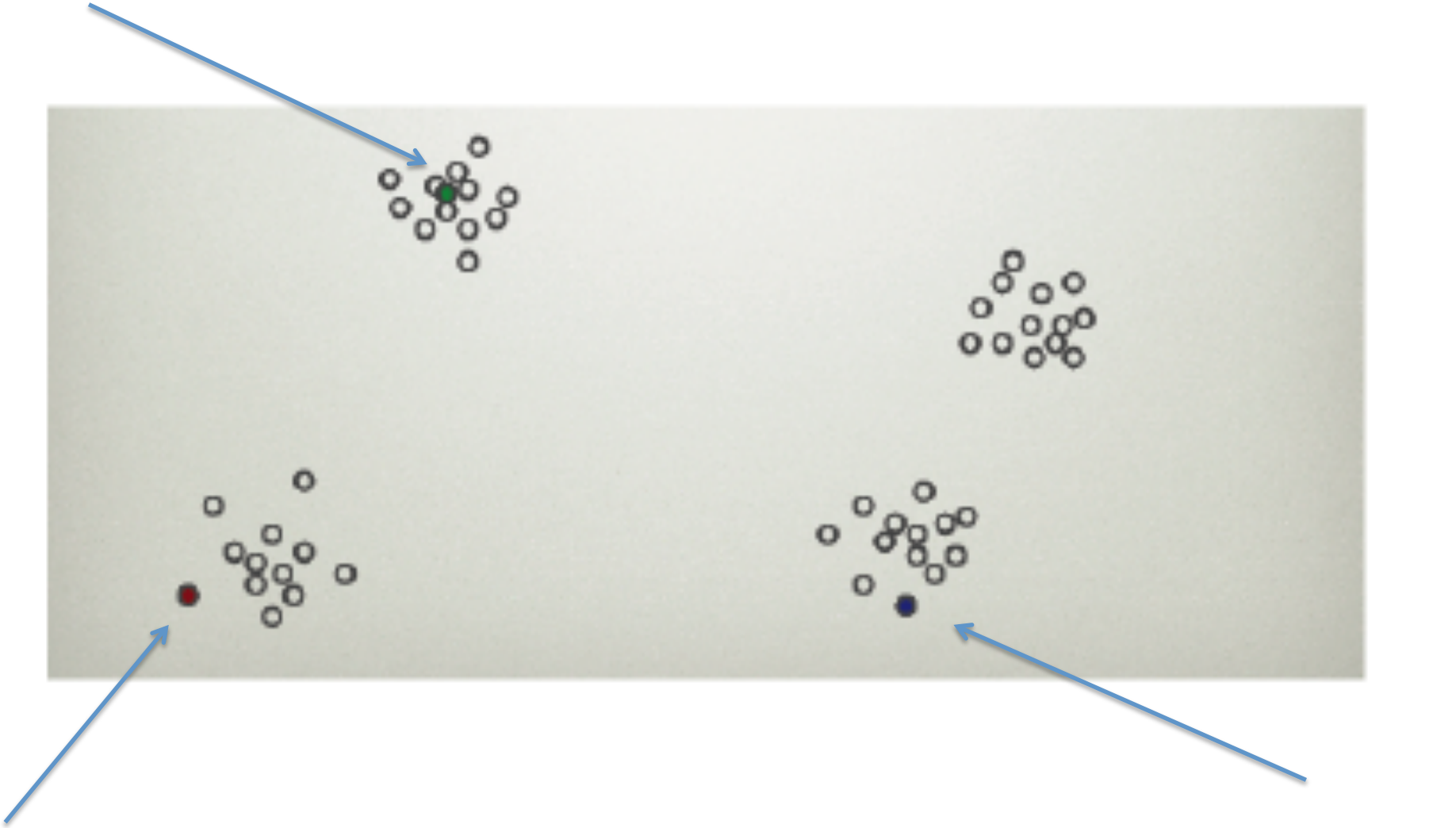
Selecting centers by distance works some time



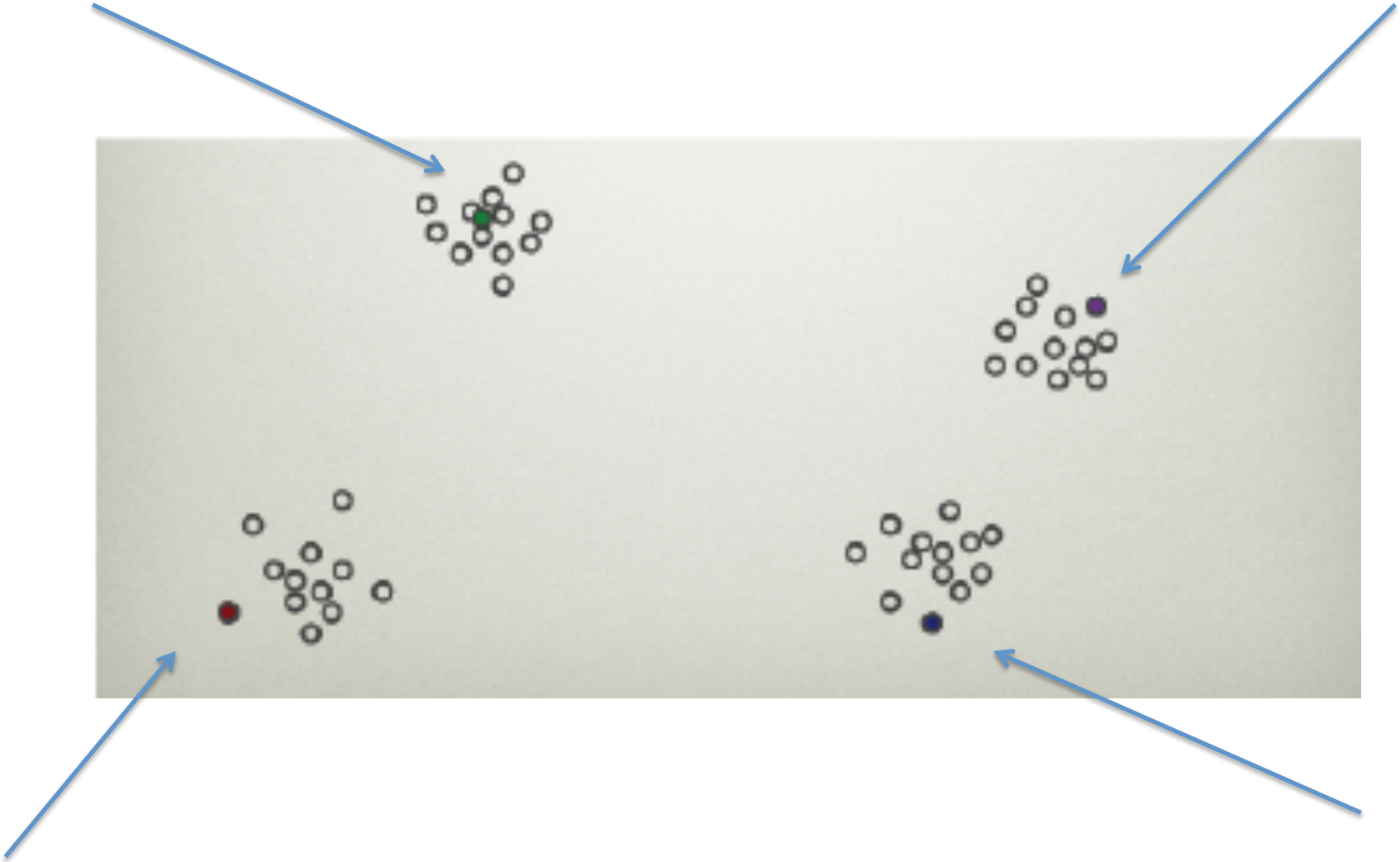
Selecting centers by distance works some time



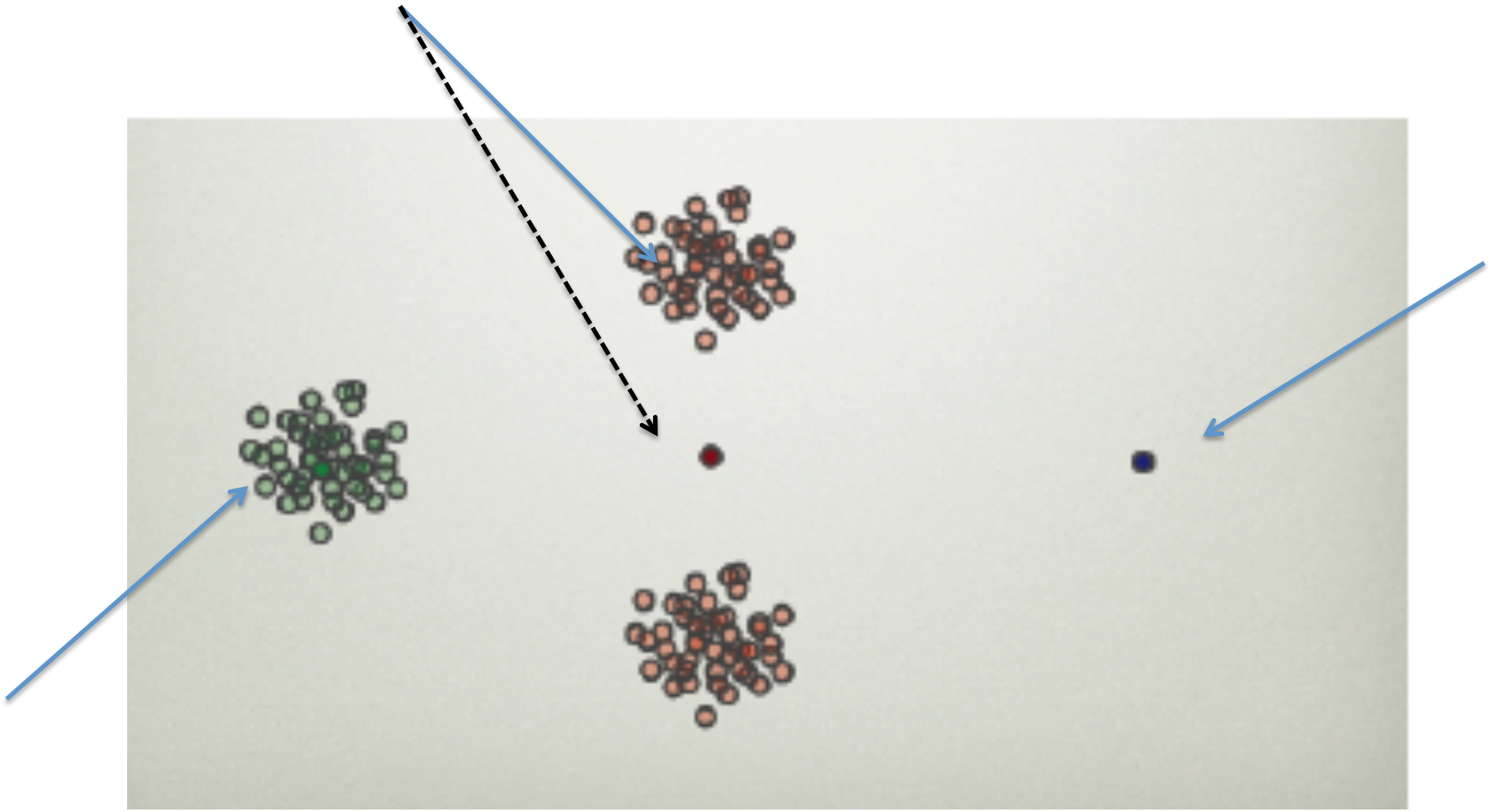
Selecting centers by distance works some time



Selecting centers by distance works some time



Sensitive to Outlier



K-Means++

- Interpolate between the two methods:

  Let $D(x)$ be the distance between x and the nearest cluster center. Sample x as a cluster center proportionately to $(D(x))^2$.

k-means++ returns clustering $\mathcal{C}$ which is log k-competitive.

Just the initialization in Llyod's algorithm changes—everything else remains the same.

Objective based clustering

- K-means: $\arg\min_S \sum_{i=1}^{k} \sum_{x_i \in S} ||x - \mu_i||^2$
- K-median $\arg\min_S \sum_{i=1}^{k} \sum_{x_i \in S} ||x - \mu_i||$
- K-center $\arg\min_S \max_{i=1}^{k} \max_{x_i \in S} ||x - \mu_i||$

K-median

- In Llyod's algorithm use the next cluster center as the median of the elements in the cluster.

Another simple local search algorithm

- Start with arbitrary k centers: C
- Assign points to the nearest center and compute the objective value
- If swapping a vertex x outside of C with a vertex v in C, decreases the objective value, *swap*

The algorithm converges and gives a 5-approximation.
Better approximation bound known.

K-center

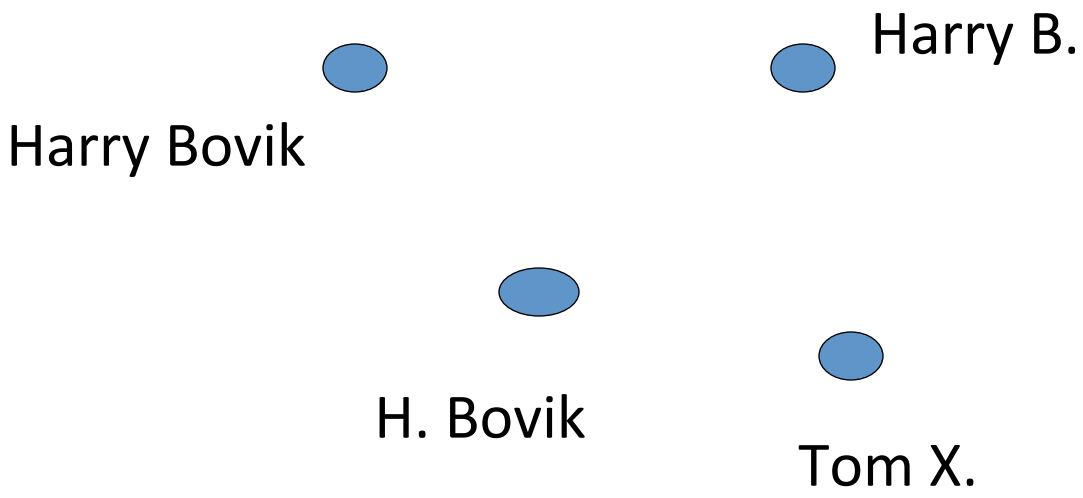
- Minimizes the maximum distance
- A simple greedy algorithm gives a 2-approximation

- Pick any vertex v arbitrarily and declare it as the first center
- For i=2 to k
  - Select the vertex in V that is farthest from the already chosen centers and make it the new *i*-th center.

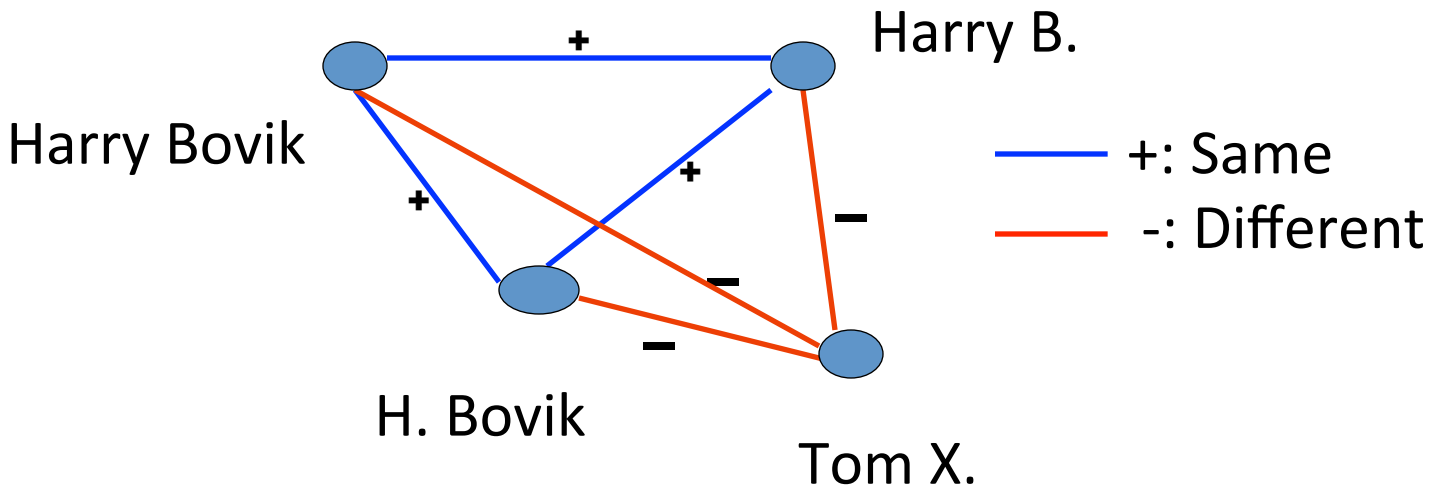
Clustering with unknown k

- Say we want to cluster *n* objects of some kind (documents, images, text strings)
- But we don't have a meaningful way to project into Euclidean space.
- Using past data train up some classifier ***f(x,y)=same/different***.
- Then run ***f*** on all pairs and try to find most consistent clustering.

The problem



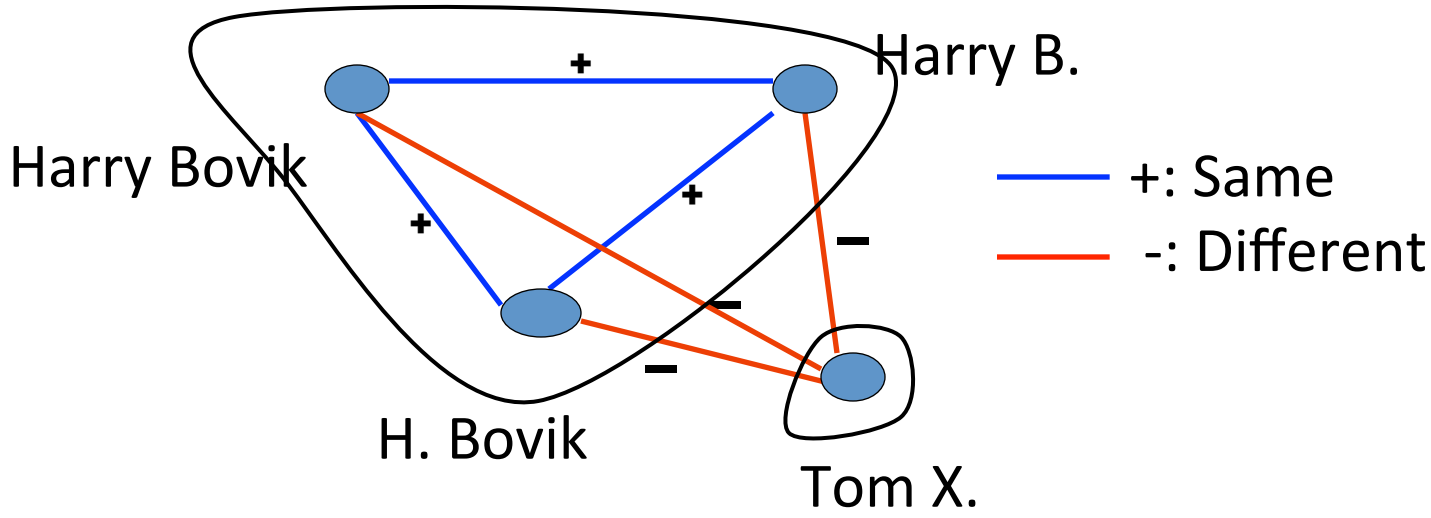
The problem



Train up f(x)= same/different

Run f on all pairs

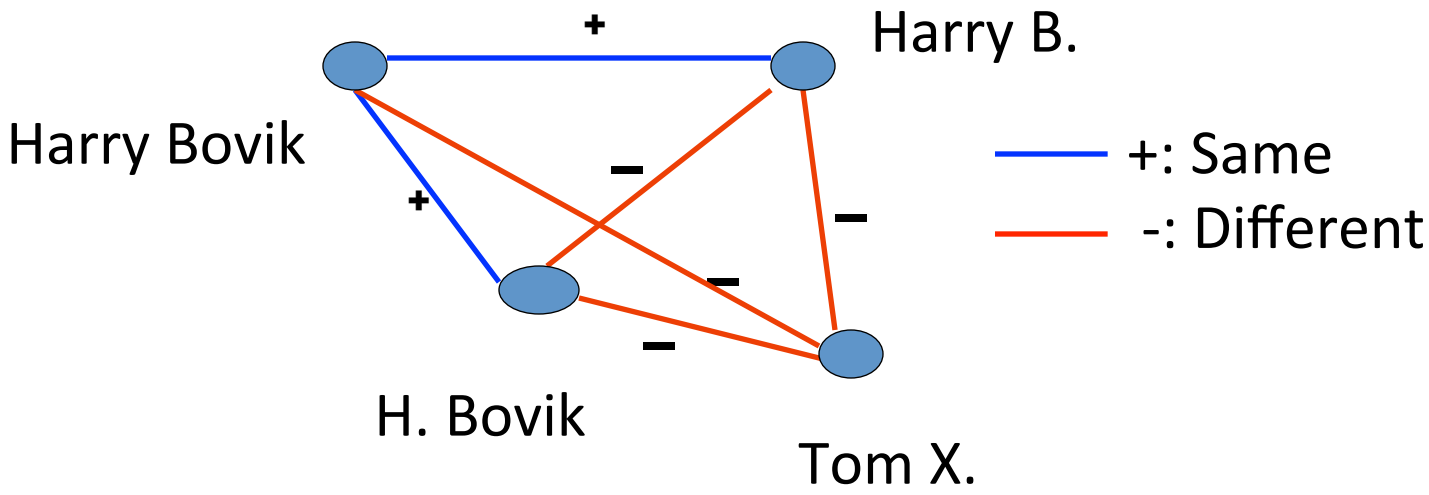
The problem



Totally consistent:

1. + edges inside clusters
2. - edges outside clusters

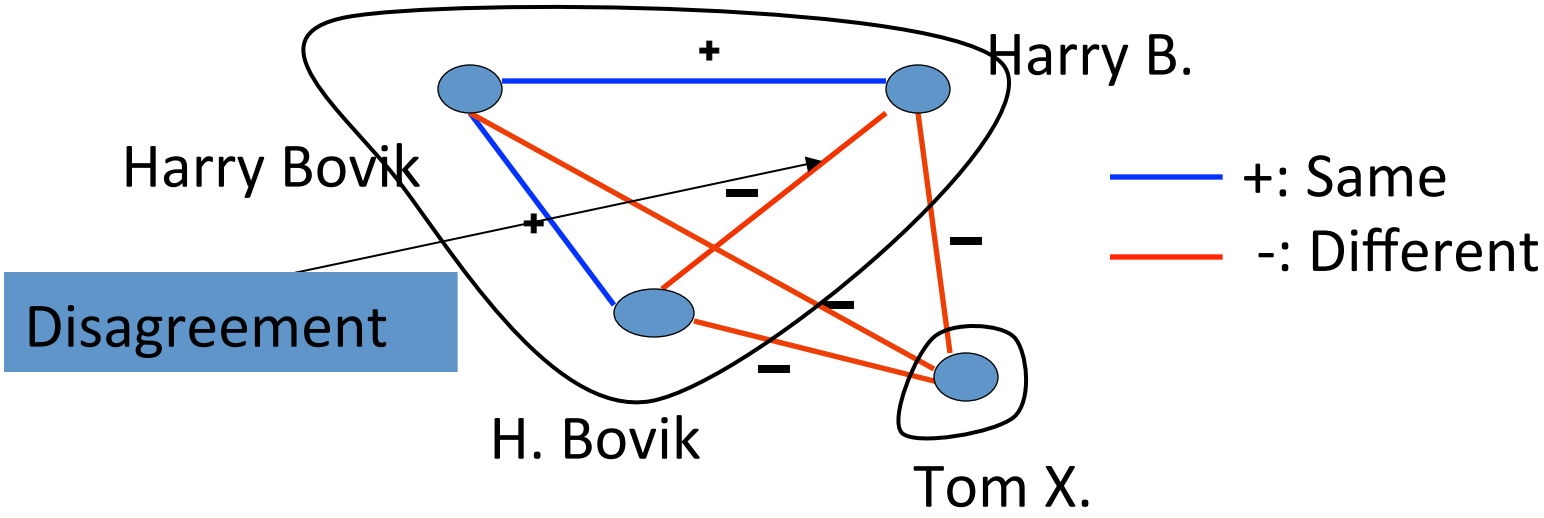
The problem



Train up f(x)= same/different

Run f on all pairs

The problem

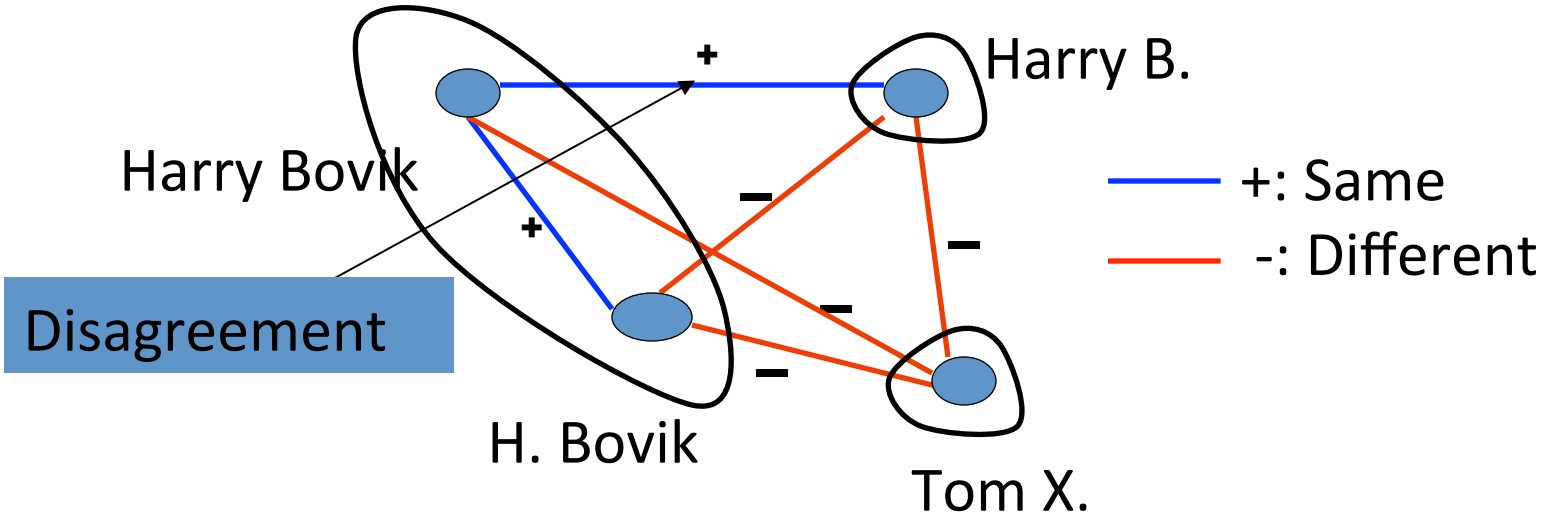


Train up f(x)= same/different

Run f on all pairs

Find most consistent clustering

The problem

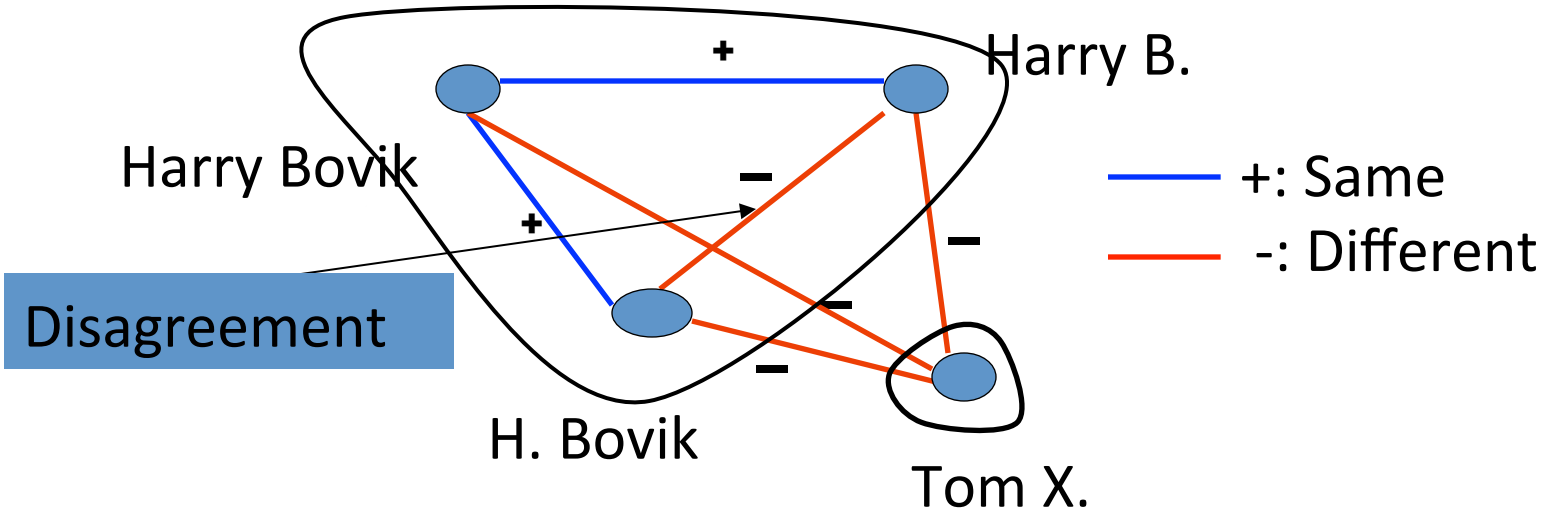


Train up f(x)= same/different

Run f on all pairs

Find most consistent clustering

The problem



Problem: Given a complete graph on n vertices.

Each edge labeled + or -.

Goal is to find partition of vertices as consistent as possible with edge labels.

Max #(agreements) or Min #(disagreements)

There is no k : # of clusters could be anything

The Problem

Noise Removal:

There is some true clustering. However some edges incorrect. Still want to do well.

Agnostic Learning:

No inherent clustering.

Try to find the best representation using hypothesis

Eg: Research communities via collaboration graph

Nice features of formulation

- There's no *k*. (OPT can have anywhere from 1 to n clusters)
- If a perfect solution exists, then it's easy to find: *$C(v) = N^+(v)$. [Why?]*
- Easy to get agreement on ½ of edges. *[Why?]*

Minimizing Disagreements

Goal: Get a constant factor approx.

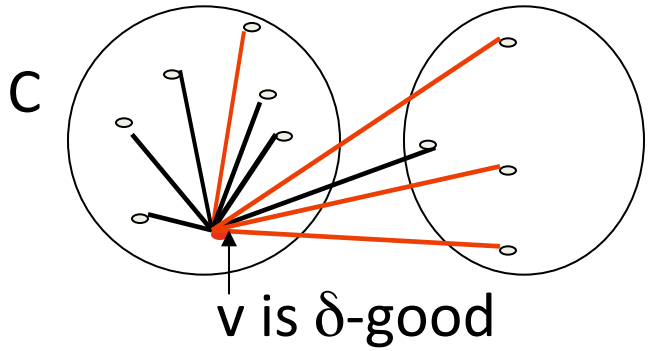
Minimizing Disagreement

- Pick a random permutation of vertices
- Select v from the random order and create a cluster with all its positive neighbors
- Remove that cluster with all associated edges
- Repeat

Gives a 3-approximation

δ-clean Clusters

Given a clustering, vertex δ-good if few disagreements



$N^-(v)$ Within C $< \delta|C|$

$N^+(v)$ Outside C $< \delta|C|$

+: Similar

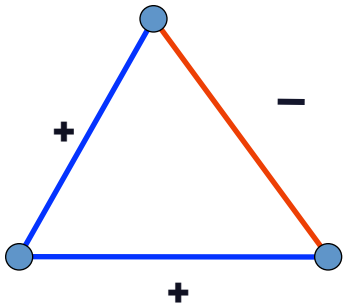
-: Dissimilar

Algorithm

1. Pick vertex v. Let $C(v) = N^+(v)$
2. Modify C(v)
   (a) Remove 3δ-bad vertices from C(v).
   (b) Add 7δ good vertices into C(v).
3. Delete C(v). Repeat until done, or above always makes empty clusters.
4. Output nodes left as singletons.

Lower bounding idea: bad triangles

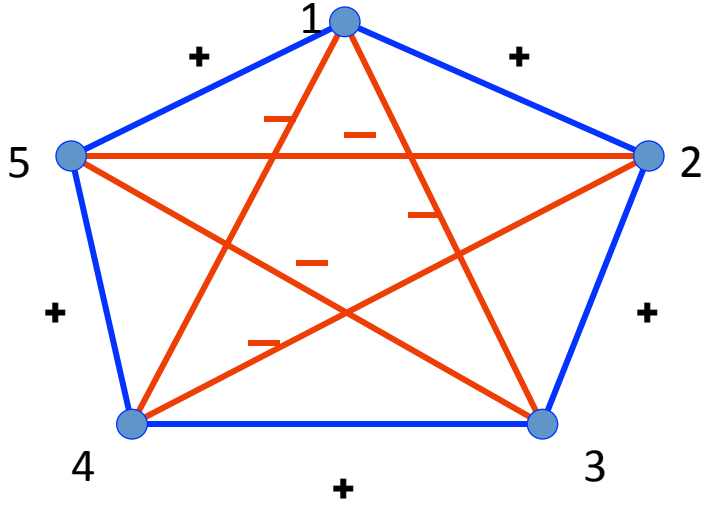
Consider



We know any clustering has to disagree with at least one of these edges.

Lower bounding idea: bad triangles

If several such disjoint, then mistake on each one



2 Edge disjoint Bad Triangles
(1,2,3), (3,4,5)

D_{opt} >= #{Edge disjoint bad triangles}