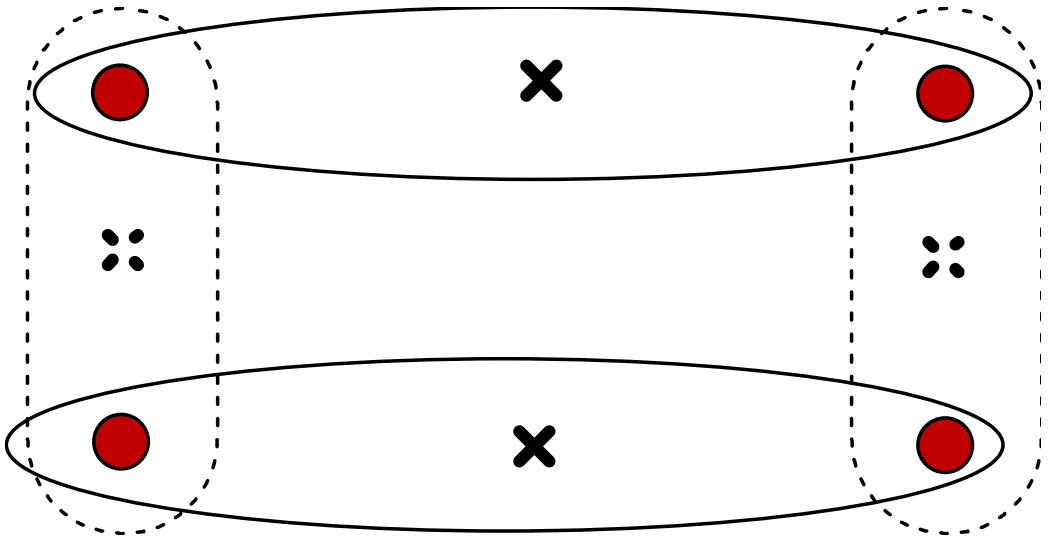
Improvements to *k*-means algorithm

Problem 1: K-Means is sensitive to initialization

- Different initializations may generate rather different clustering results (some could be far from optimal)
- Original proposal (MacQueen'67): Select K seeds randomly
  - Need to run the algorithm multiple times using different seeds
- There are many methods proposed for better initialization of *k* seeds



***K-Means*++** (Arthur & Vassilvitskii'07)

K-means++ is slower than randomly selecting centroids but gives better results, especially in reducing the Sum of Squared Errors (SSE), which measures the compactness of clusters.

Algorithm Steps

1. **First Centroid**: Choose one initial point randomly from the dataset as the first centroid.
2. **Remaining Centroids**: For each of the remaining $k - 1$ centroids:
    1. For each point in the dataset, calculate the distance to the nearest already chosen centroid. This ensures each point is somewhat close to at least one centroid.
    2. Pick the next centroid randomly, but with a probability that is proportional to the square of the distance to the nearest centroid. Points farther from any centroid are more likely to be chosen as new centroids, helping to spread them out.
3. **End**: Repeat until all k centroids are selected.

K-means++

- K-means++ is slower than randomly selecting centroids but gives better results, especially in reducing the Sum of Squared Errors (SSE), which measures the compactness of clusters.
- Choosing the centroids based on probability rather than the farthest ones makes this algorithm more balanced and robust, helps avoid outliers as centroids, and prevents deterministic patterns.

1: For the first centroid, pick one of the points at random.
2: **for** $i = 1$ to *number of trials* **do**
3: Compute the distance, $d(x)$, of each point to its closest centroid.
4: Assign each point a probability proportional to each point's $d(x)^2$.
5: Pick new centroid from the remaining points using the weighted probabilities.
6: **end for**

Src

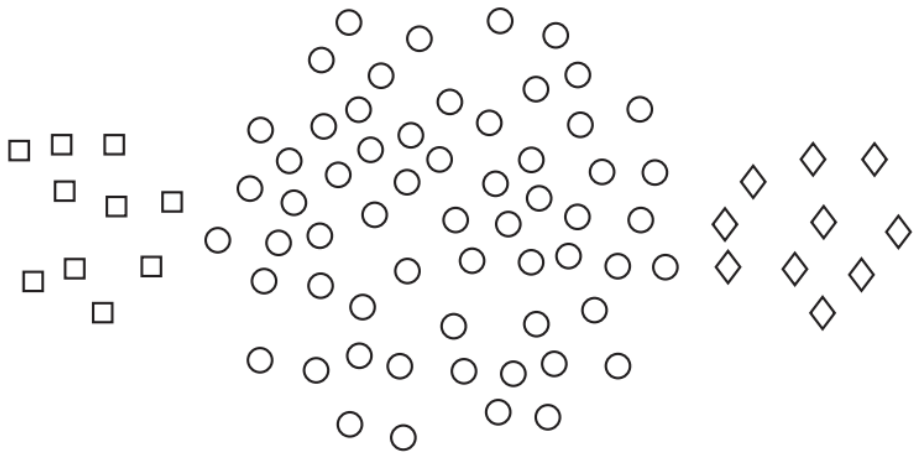
K-means vs. K-means++

- K-means++ is generally preferred over K-means
  - Even among advanced initialization strategies for K-Means (e.g., Random Partition, Maximin, among others), K-means++ is generally among the best
- K-means++ has better quality guarantee than K-means
  - For K-means, clustering can be arbitrarily worse than the optimum
  - K-means++ guarantees an approximation ratio O(log k) in expectation
    - Approximation ratio means that the solution produced by K-means++ will be within a factor of $O(\log k)$ of the optimal solution. This is a theoretical guarantee.
- In practice K-means++ often performs better than K-means in both quality and speed

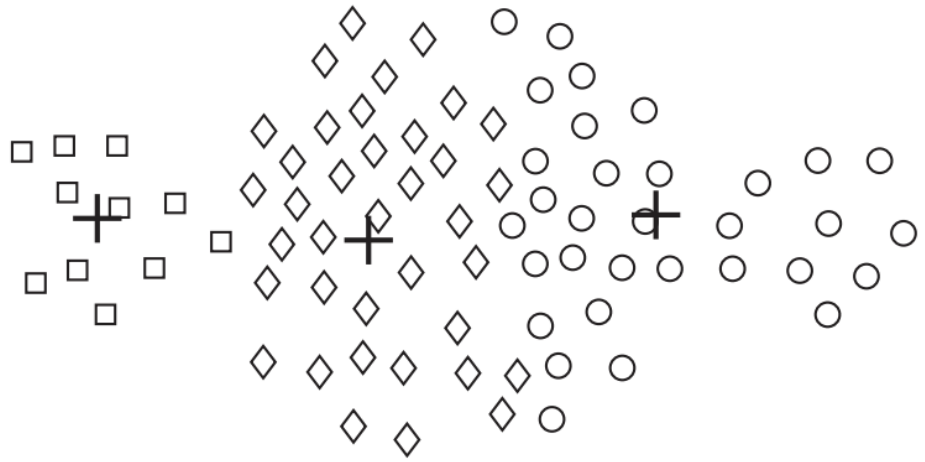
Limitations of K-means

- K-means has problems when clusters are of differing
  - Sizes
  - Densities
  - Non-globular shapes
- K-means has problems when the data contains outliers.

Limitations of K-means: Differing Size

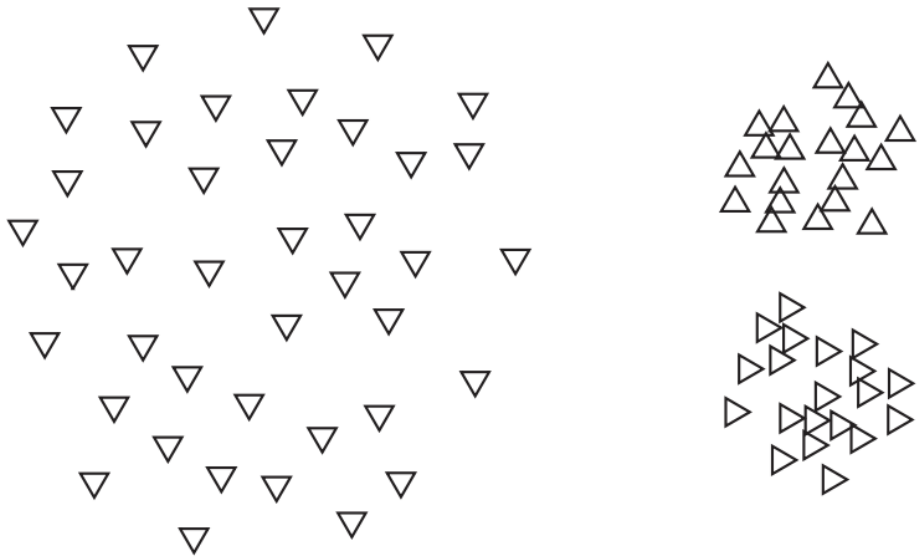


(a) Original points.

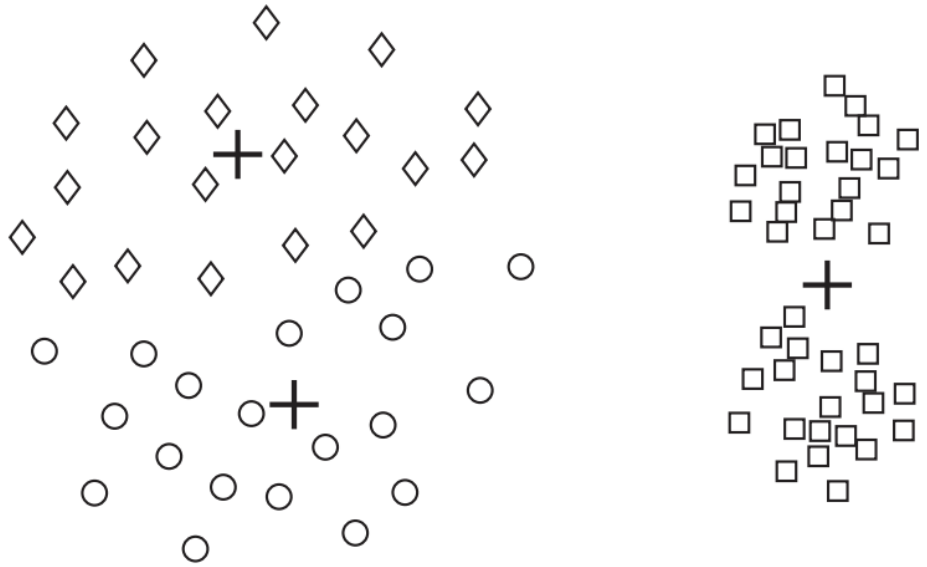


(b) Three K-means clusters.

Limitations of K-means: Differing Density

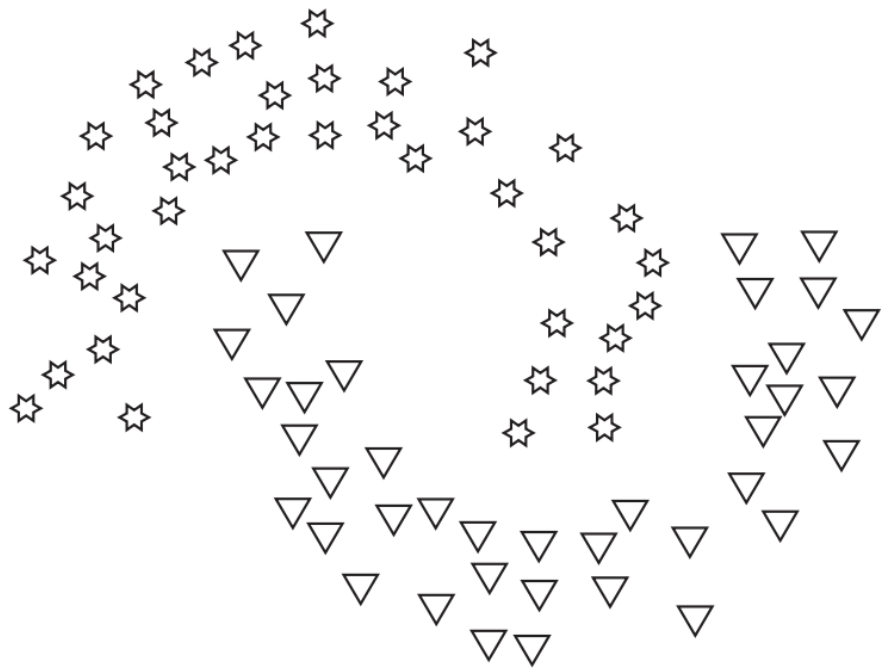


(a) Original points.

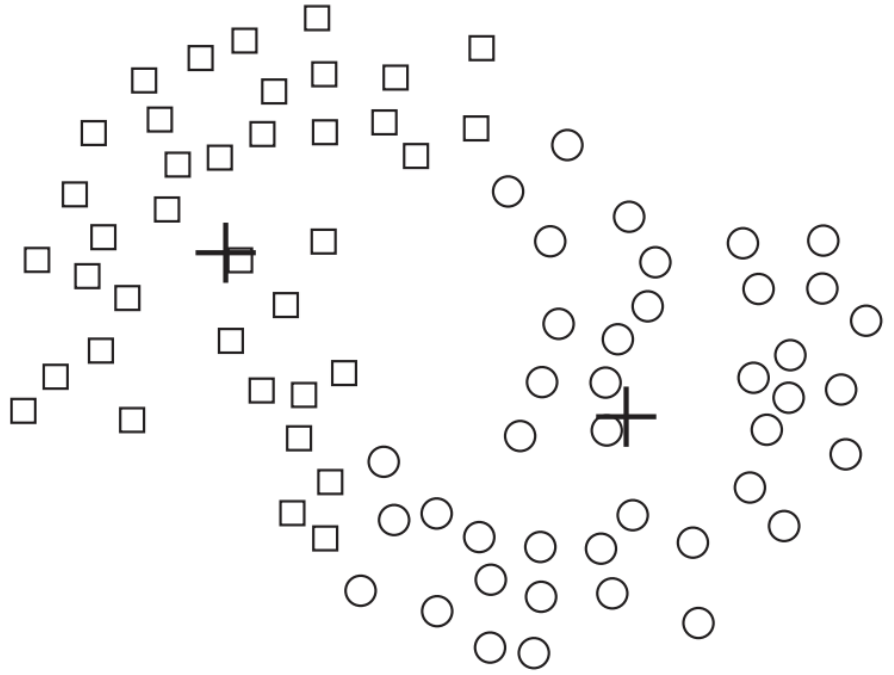


(b) Three K-means clusters.

Limitations of K-means: Non-globular Clusters

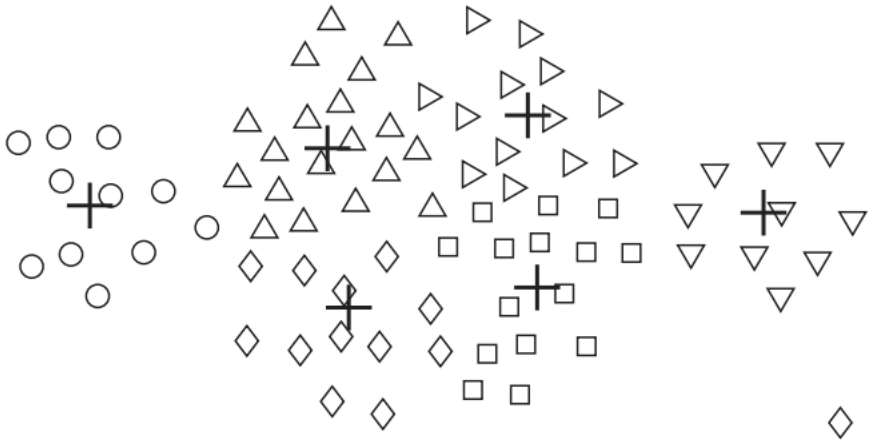


(a) Original points.

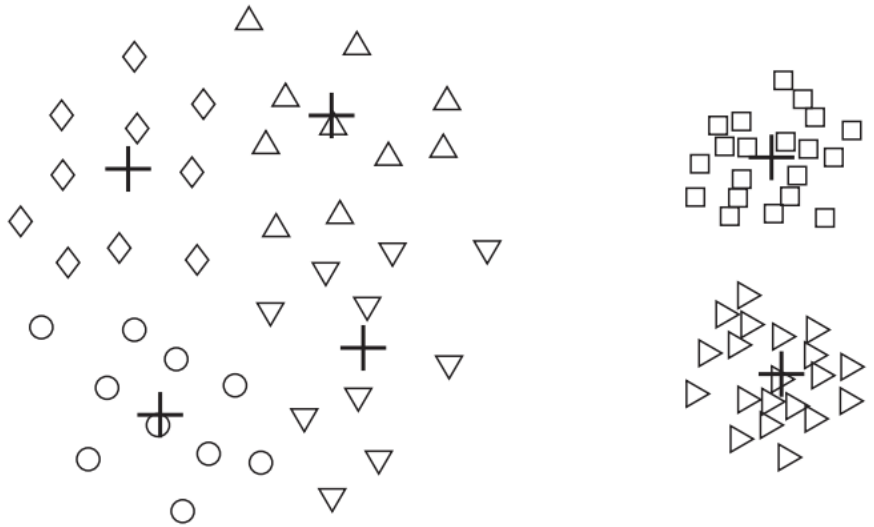


(b) Two K-means clusters.

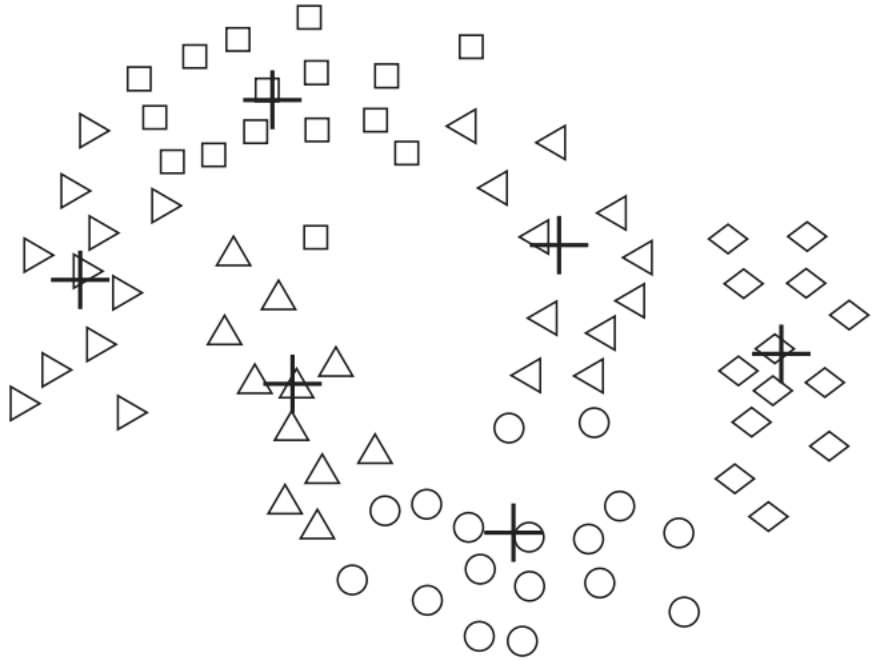
Using K-means to find clusters that are subclusters of the natural clusters



(a) Unequal sizes.



(b) Unequal densities.



(c) Non-spherical shapes.

Problem 2: *k*-Means is susceptible to outliers

- The **k-means algorithm** calculates the center of clusters by averaging points in each cluster.
- This approach is **sensitive to outliers** (points far from most other points) because they pull the average toward them, which can distort the clusters.

Example

- There are six points on a line: 1, 2, 3, 8, 9, 10, and 25.
- k = 2 (two clusters), and point 25 is an outlier
- **k-means Clustering**:
  - When using k-means, the points are split into two clusters: {1, 2, 3} and {8, 9, 10, 25}.
  - The average of {8, 9, 10, 25} becomes 14.67, which doesn't represent the main part of this cluster well because 25 is pulling it higher.
- This results in **high within-cluster variation** due to the outlier.

k-Medoids as an alternative

- k-Medoids is a variation of k-means, designed to reduce sensitivity to outliers.
- Instead of using the mean (average), k-medoids selects an actual point from the dataset as the center (or “medoid”) of each cluster.
- Each data point is assigned to the cluster with the closest medoid, based on minimizing the absolute-error (total distance from each point to its closest medoid).

The *k*-medoids method

"How can we modify the k-means algorithm to diminish such sensitivity to outliers?" Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is assigned to the cluster of which the representative object is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object $\boldsymbol{p}$ and its corresponding representative object. That is, an **absolute-error criterion** is used, defined as

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{p} \in C_i} dist(\boldsymbol{p}, \boldsymbol{o_i}), \tag{8.2}$$

where E is the sum of the absolute error for all objects $\boldsymbol{p}$ in the data set, and $\boldsymbol{o_i}$ is the representative object of C_i. This is the basis for the ***k*-medoids method**, which groups n objects into k clusters by minimizing the absolute error (Eq. (8.2)).

When $k = 1$, we can find the exact median in $O(n^2)$ time. However, when k is a general positive number, the k-medoid problem is NP-hard.

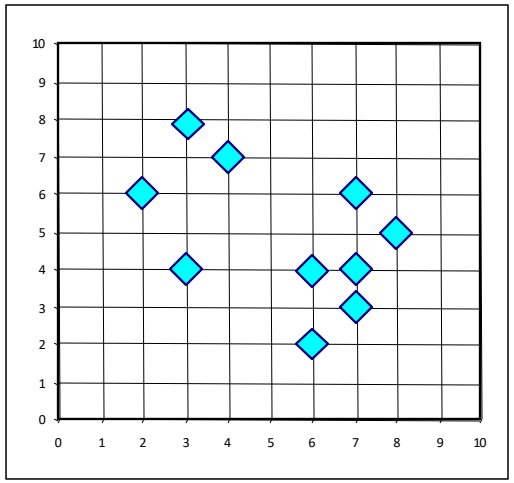
Partitioning Around Medoids (PAM)

- Partitioning Around Medoids (PAM) algorithm is a popular way to perform k-medoids clustering
- It uses actual data points (medoids) as the cluster centers.
- This makes it more robust to noise and outliers.
- PAM's decision-making process is based on immediate, local cost reduction. Each swap is chosen based on its immediate impact on clustering cost, not on long-term or global optimality. Hence PAM is a greedy, iterative algorithm.
- k-medoids clustering is NP-hard, so finding the true optimal solution is computationally prohibitive for large k and large datasets.
- PAM provides an approximate solution to the k-medoids problem by using a greedy algorithm that iteratively improves the selection of medoids.
- PAM's greedy nature means it doesn't guarantee an optimal solution, but it is much faster than an exact approach would be.

Steps in the PAM Algorithm:

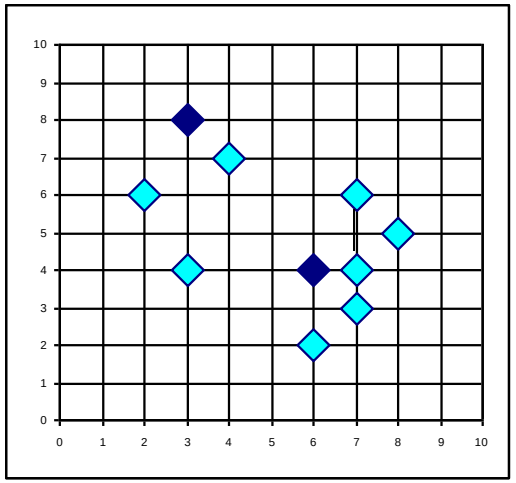
- **Initialization**: Randomly pick k objects as the initial medoids.
- **Assignment**: Assign each non-medoid object to the nearest medoid to form clusters.
- **Swap Evaluation**:
  - Randomly pick a non-medoid object, *o_random*, and calculate the cost of swapping each medoid *o_j* with *o_random*.
  - If swapping reduces the overall clustering cost (measured by a change in absolute error, E), the swap is performed.
- **Repeat**: Steps 2 and 3 continue until no further swaps improve the cost.

PAM: A Typical K-Medoids Algorithm

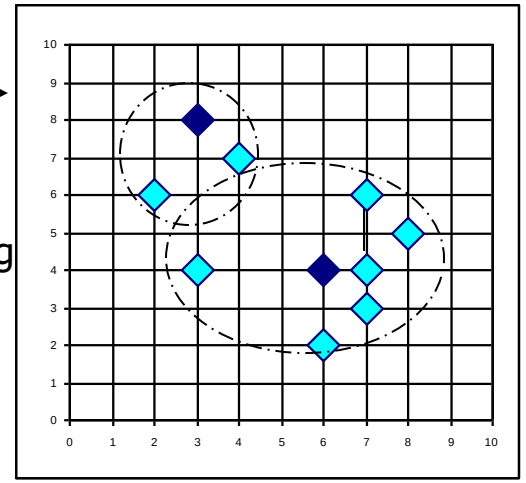


$K = 2$

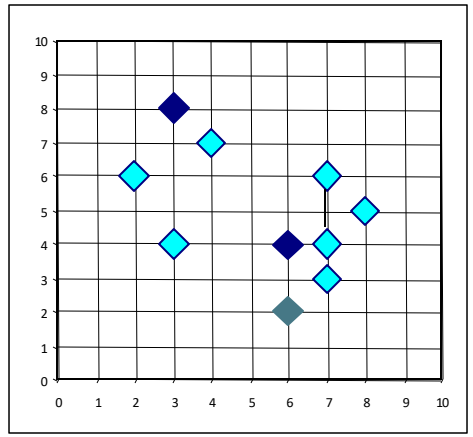
Arbitrary choose K object as initial medoids



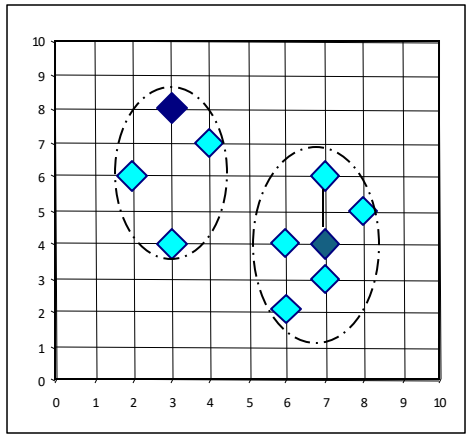
Assign each remaining object to nearest medoids



Randomly select a non-medoid object, O_{ramdom}



Compute total cost of swapping



Swapping O and O_{ramdom}

If quality is improved

Select initial *K medoids* randomly

Repeat

Object re-assignment

Swap medoid *m* with o_i if it improves the clustering quality

Until convergence criterion is satisfied

Discussion on K-Medoids Clustering

- K-Medoids Clustering: Find representative objects (medoids) in clusters
- PAM (Partitioning Around Medoids: Kaufmann & Rousseeuw 1987)
  - Starts from an initial set of medoids, and
  - Iteratively replaces one of the medoids by one of the non-medoids if it improves the total sum of the squared errors (SSE) of the resulting clustering
  - PAM works effectively for small data sets but does not scale well for large data sets (due to the computational complexity)
  - Computational complexity: PAM: $O(K(n - K)2)$ (quite expensive!)
- Efficiency improvements on PAM
  - CLARA (Kaufmann & Rousseeuw, 1990):
    - PAM on samples; $O(Ks2 + K(n - K))$, s is the sample size
  - CLARANS (Ng & Han, 1994): Randomized re-sampling, ensuring efficiency + quality

K-Medians: Handling Outliers by Computing Medians

- Medians are less sensitive to outliers than means
  - Think of the median salary vs. mean salary of a large firm when adding a few top executives!
- K-Medians: Instead of taking the mean value of the object in a cluster as a reference point, medians are used (L1-norm as the distance measure)
- The criterion function for the K-Medians algorithm

$$S = \sum_{k=1}^{K} \sum_{x_i \in C_k} |x_{ij} - med_{kj}|$$

- The K-Medians clustering algorithm:
  - Select K points as the initial representative objects (i.e., as initial K medians)
  - Repeat
    - Assign every point to its nearest median
    - Re-compute the median using the median of each individual feature
  - Until convergence criterion is satisfied

K-Modes: Clustering Categorical Data

- K-Means cannot handle non-numerical (categorical) data
  - Mapping categorical value to 1/0 cannot generate quality clusters for high-dimensional data
- K-Modes: An extension to K-Means by replacing means of clusters with modes
- Dissimilarity measure between object X and the center of a cluster Z
  - $\Phi(x_j, z_j) = 1$ when $x_j \neq z_j$; and 0 otherwise
- This dissimilarity measure (distance function) is frequency-based
- Algorithm is still based on iterative object cluster assignment and centroid update
- A mixture of categorical and numerical data: Using a K-Prototype method