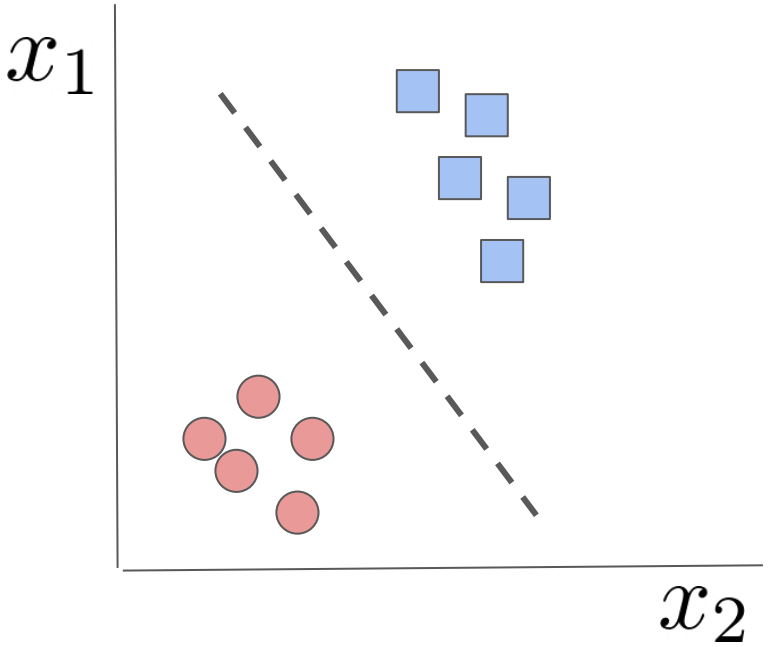
K-Means Algorithm

An introduction

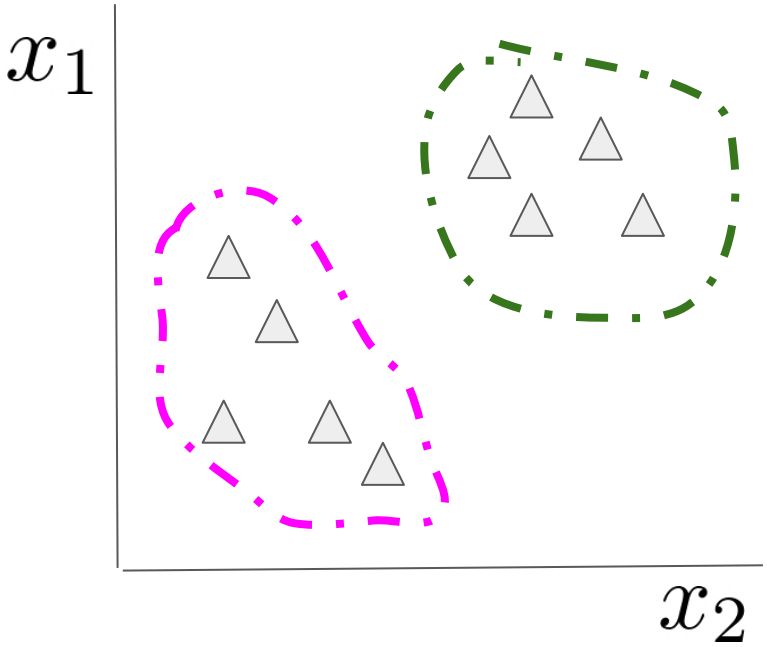
Supervised Learning

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)})$



Unsupervised Learning

Training set: $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}$



K-Means algorithm

Input:

- K (number of clusters)
- Training set $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}$

$$x^{(i)} \in R^n$$

x_0 = 1 convention is dropped

K-Means algorithm

$$c_i \neq \phi, i = 1, 2, ..., K$$
$$c_i \cap c_j = \phi, i \neq j$$
$$\cup_{i=1}^{K} c_i = T$$

K-Means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {

For $i = 1$ to m

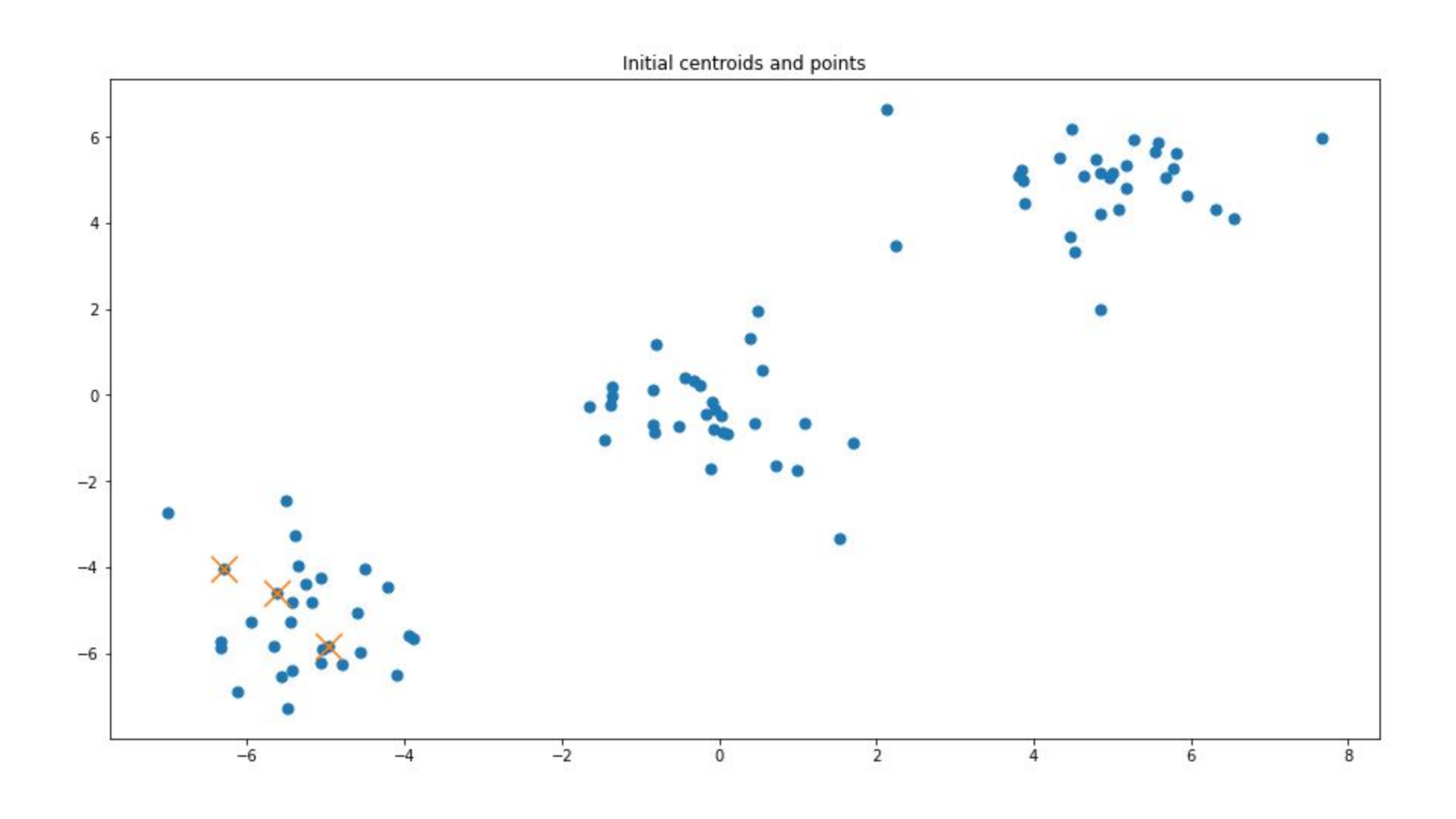
$c^{(i)}$ = index of cluster centroid closest to $x^{(i)}$ — Cluster assignment

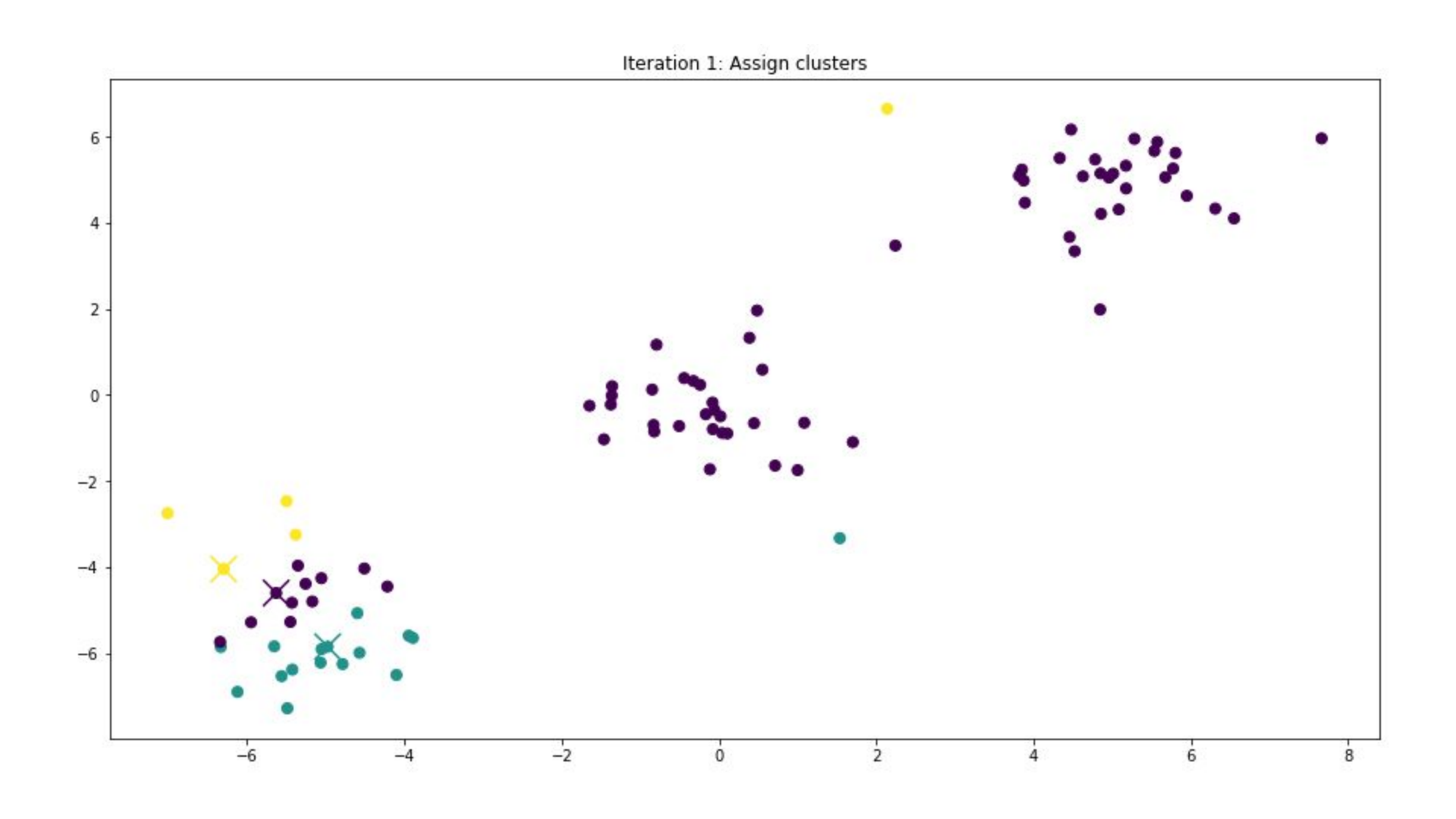
For $k = 1$ to K

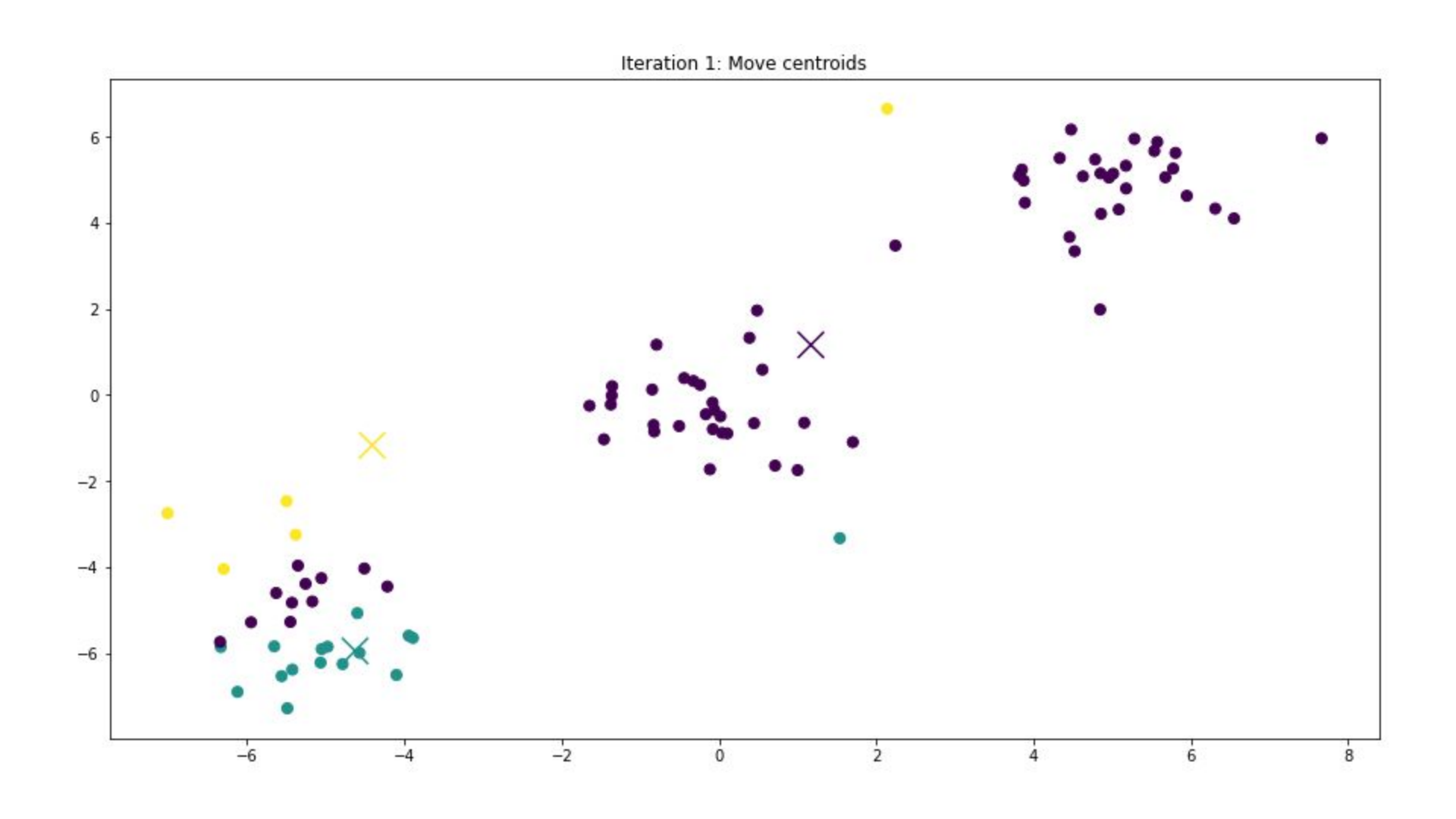
μ_k = mean of points assigned to cluster k — Move centroids

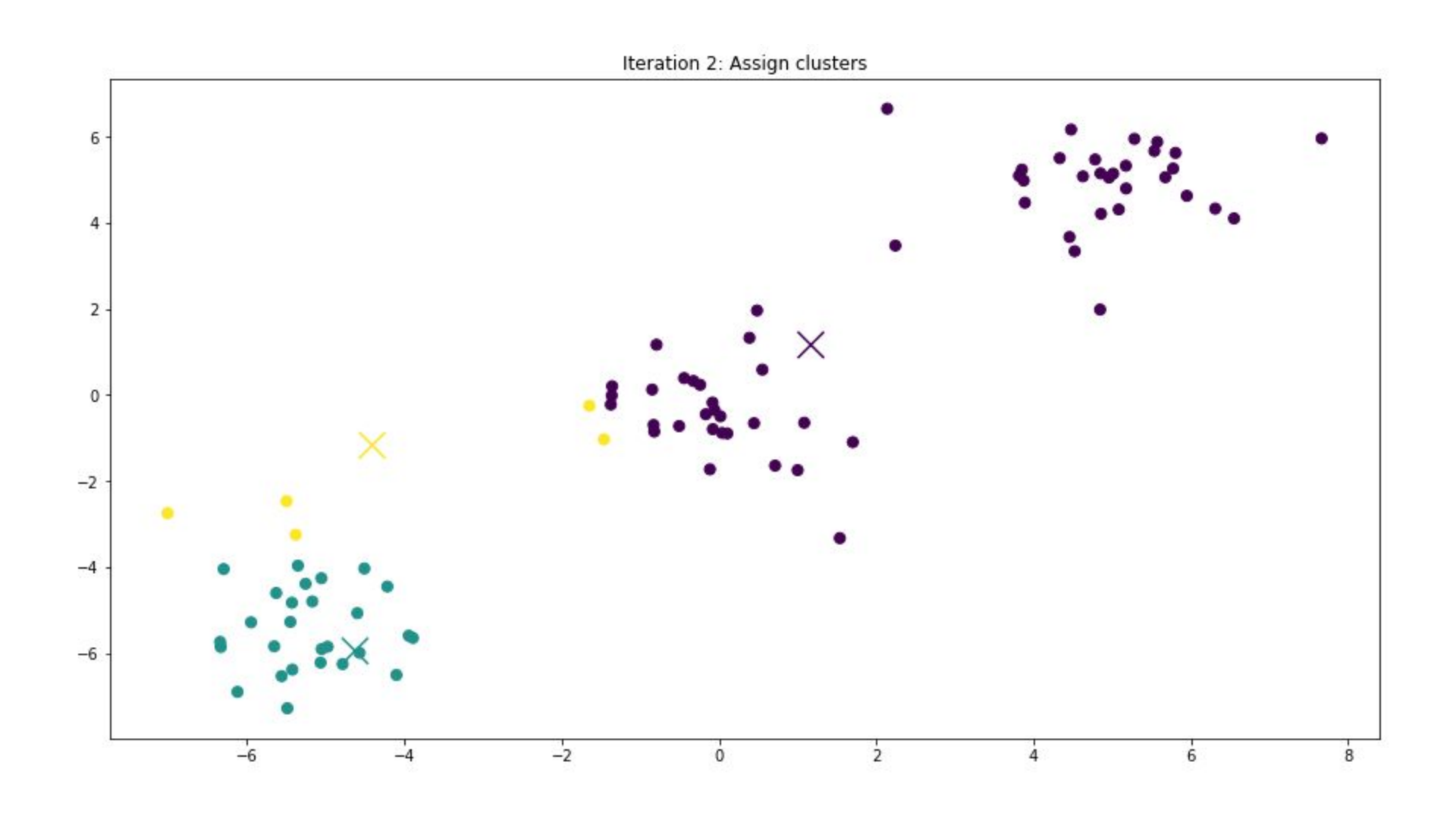
}

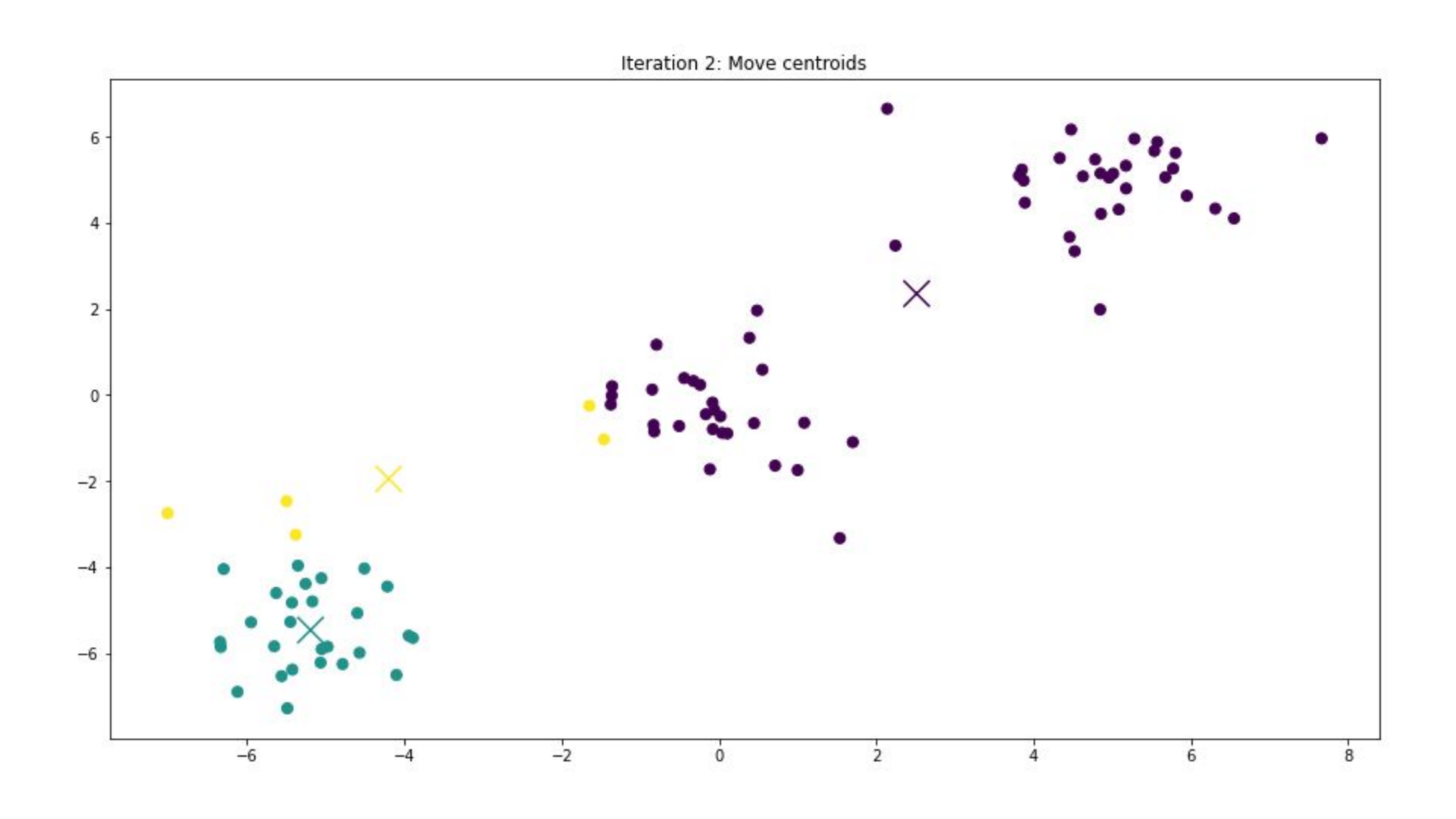
Intuition: points in same cluster are tightly packed (minimal intra-class variance) while points in different clusters are far apart (maximal inter-class variance)

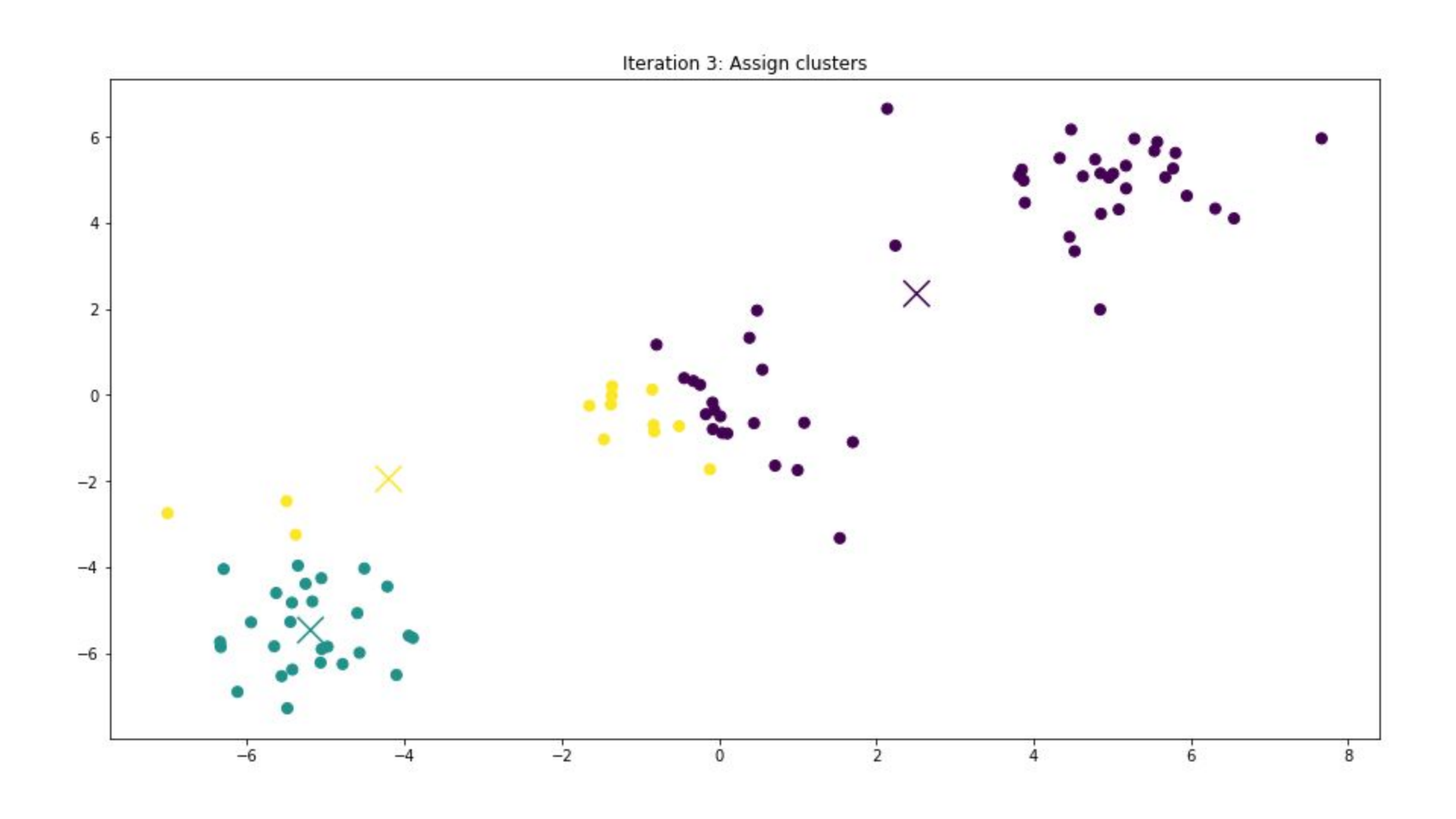

Initial centroids and points
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

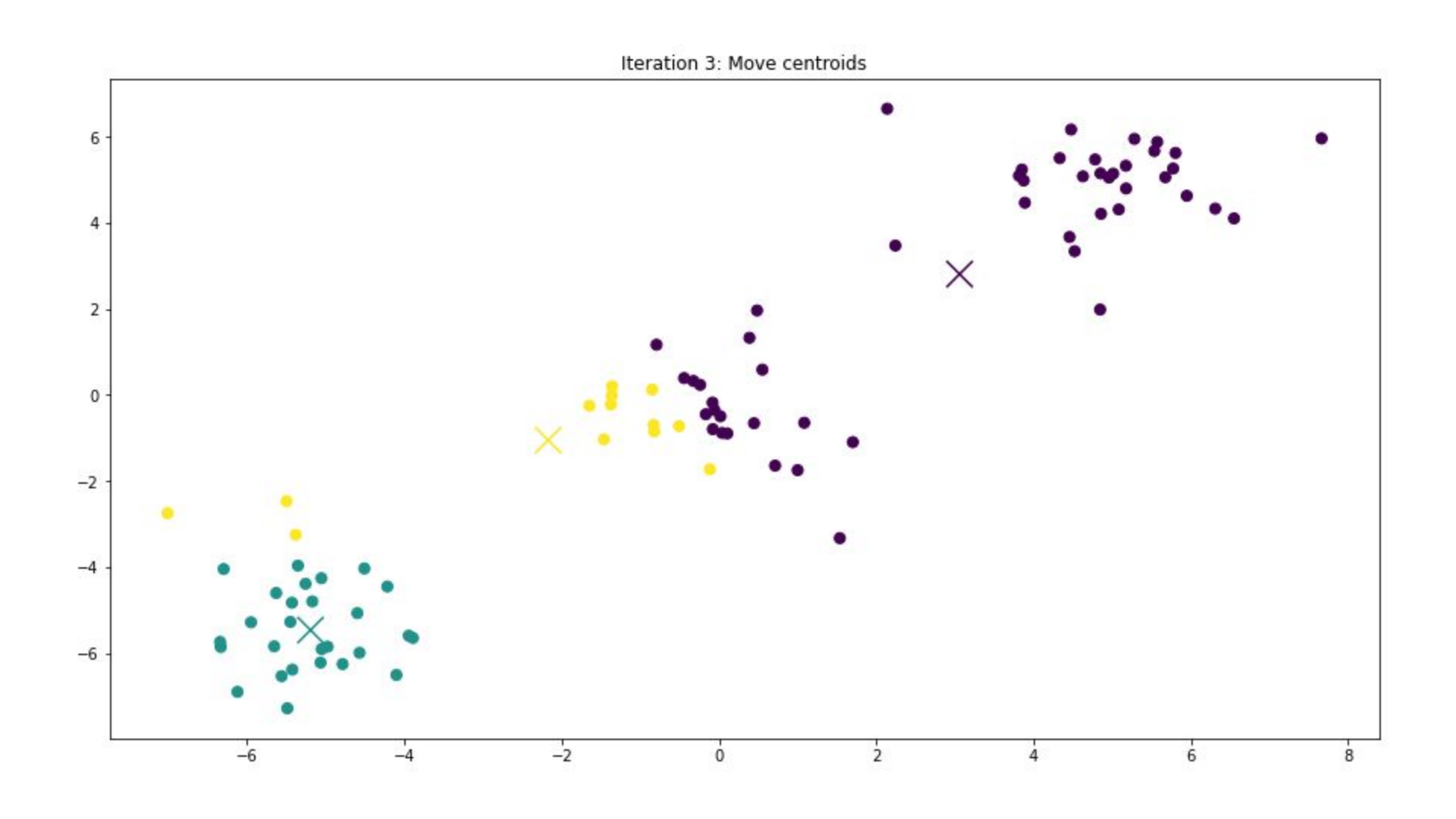

Iteration 1: Assign clusters
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

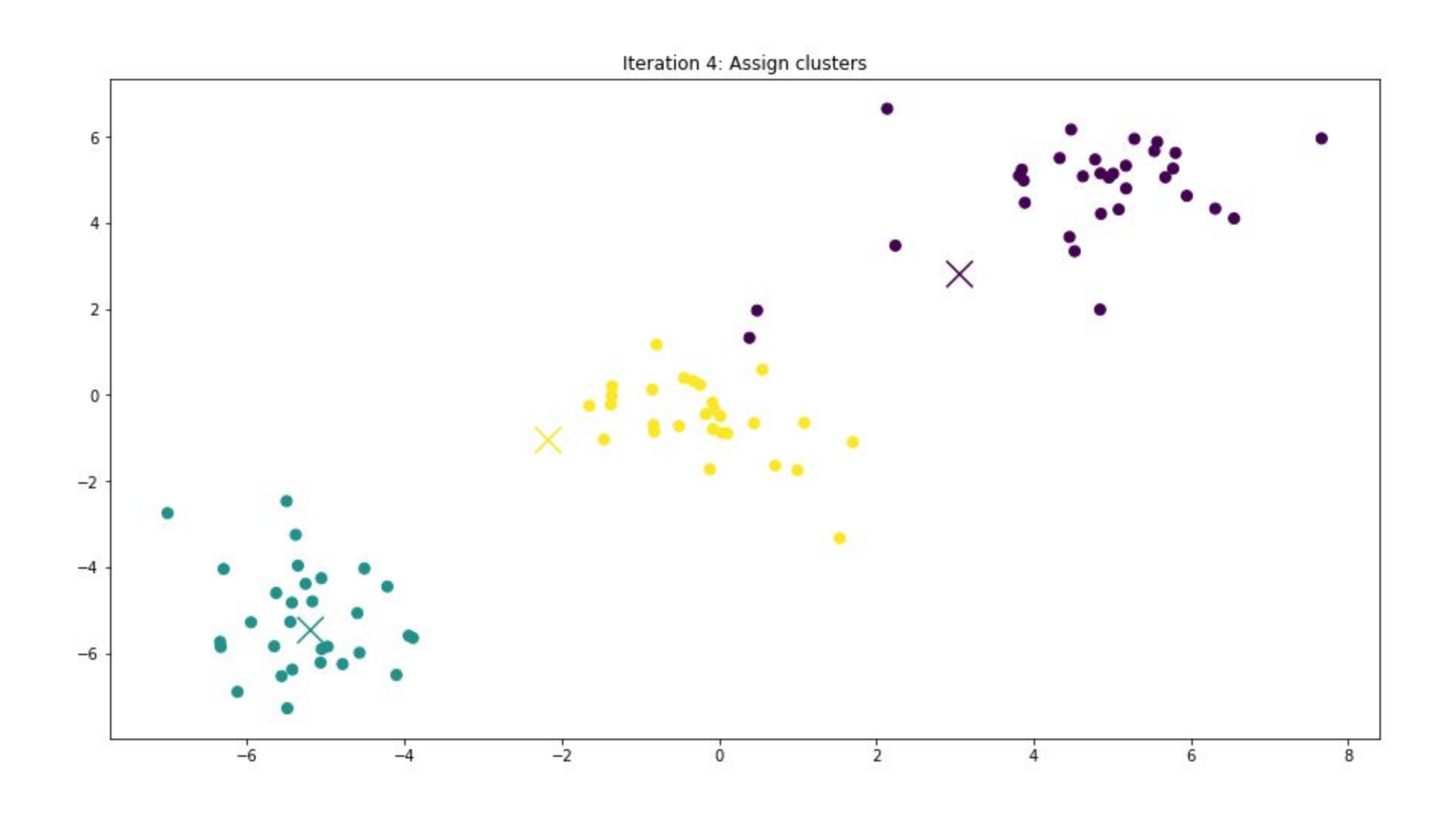

Iteration 1: Move centroids
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

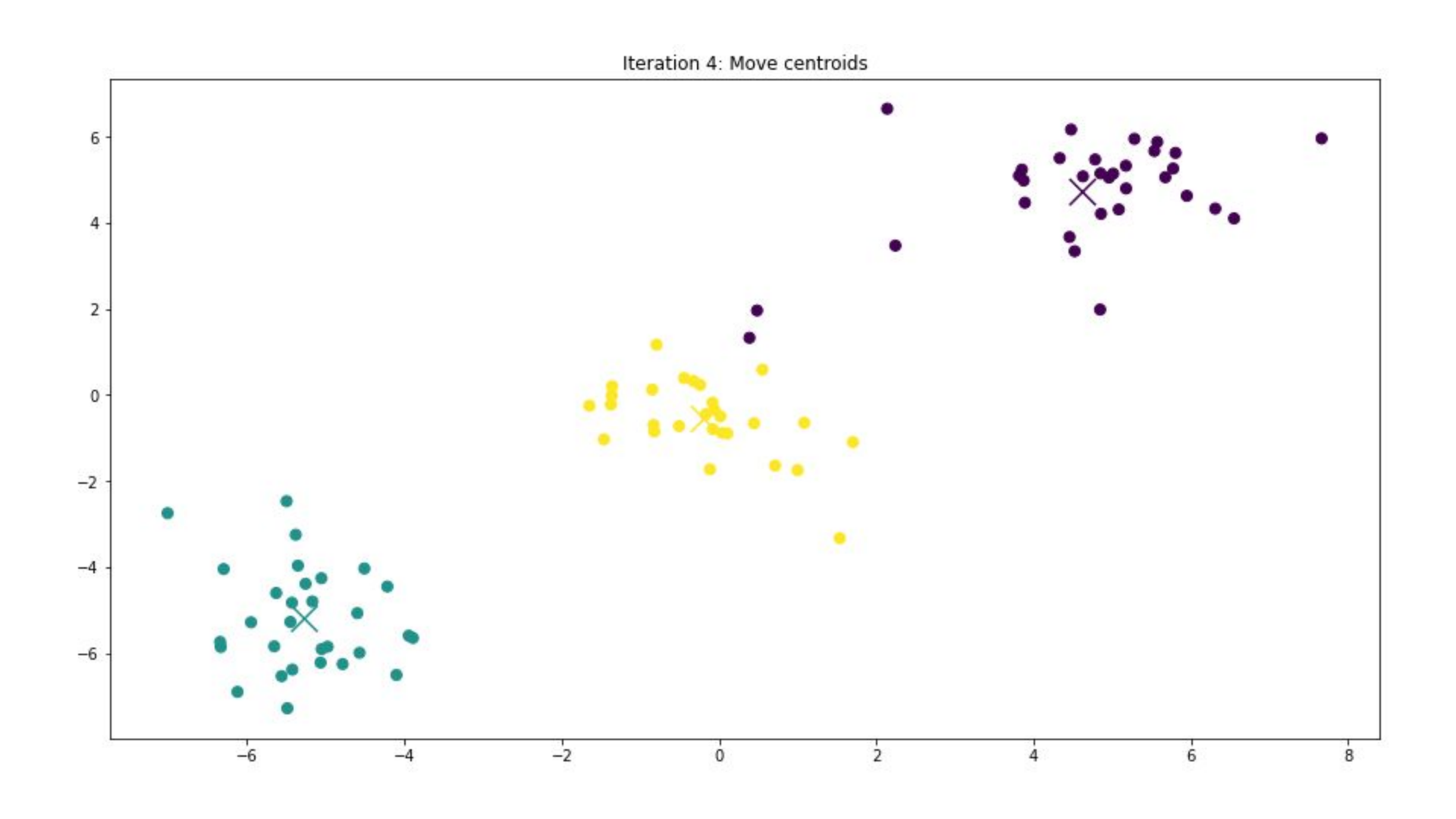

Iteration 2: Assign clusters
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

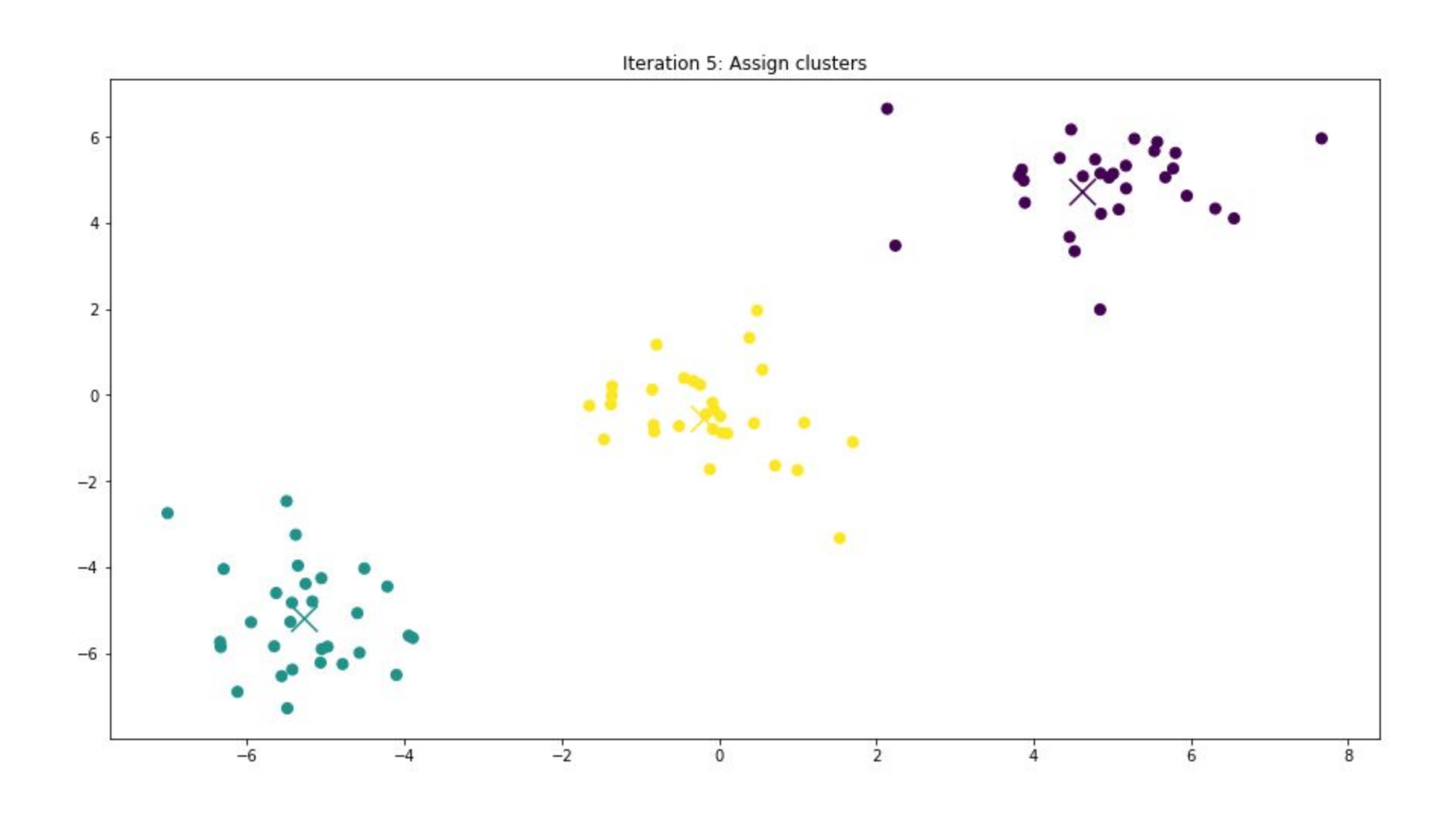

Iteration 2: Move centroids
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

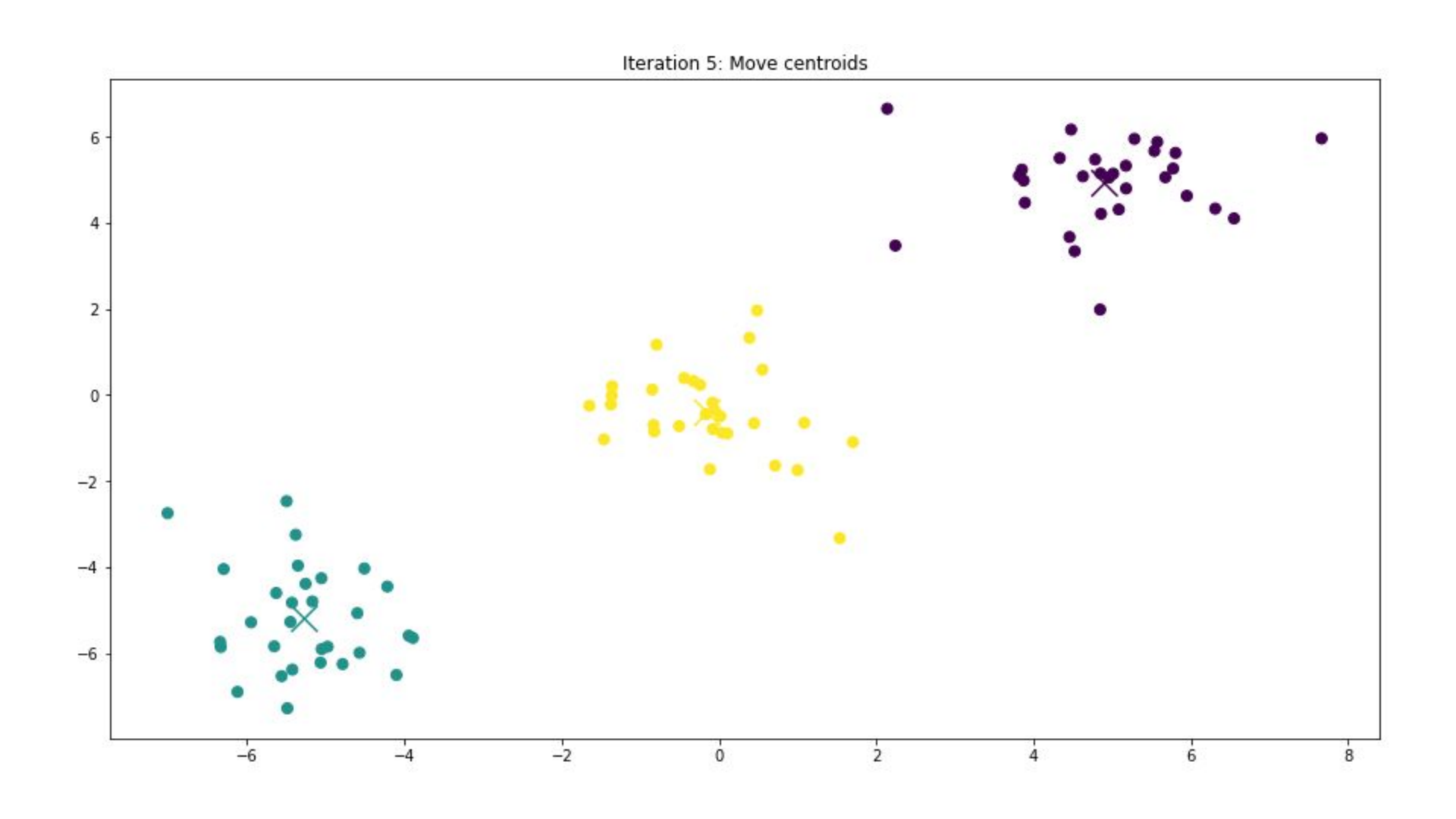

Iteration 3: Assign clusters
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8


Iteration 3: Move centroids
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8


Iteration 4: Assign clusters
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8


Iteration 4: Move centroids
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8


Iteration 5: Assign clusters
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8


Iteration 5: Move centroids
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

K-Means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {

For $i = 1$ to m

$c^{(i)}$ = index of cluster centroid closest to $x^{(i)}$

Cluster assignment

For $k = 1$ to K

μ_k = mean of points assigned to cluster k

Move centroids

}

K-Means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {

For $i = 1$ to m

$c^{(i)}$ = index of cluster centroid closest to $x^{(i)}$ — Cluster assignment

For $k = 1$ to K

μ_k = mean of points assigned to cluster k — Move centroids

}

1. Random initialization may affect performance
2. Need a metric for clustering performance to decide when to stop
3. Also need to judge the performance of random initializations

K-Means optimization objective

$$J(c^{(1)}, c^{(2)}, ..., c^{(m)}, \mu_1, \mu_2, ..., \mu_k) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

- It is the average squared error between the training samples and their representative prototypes (cluster centroids)
- The above value is also called **distortion**
- The goal is to minimize the optimization objective with respect to the parameters

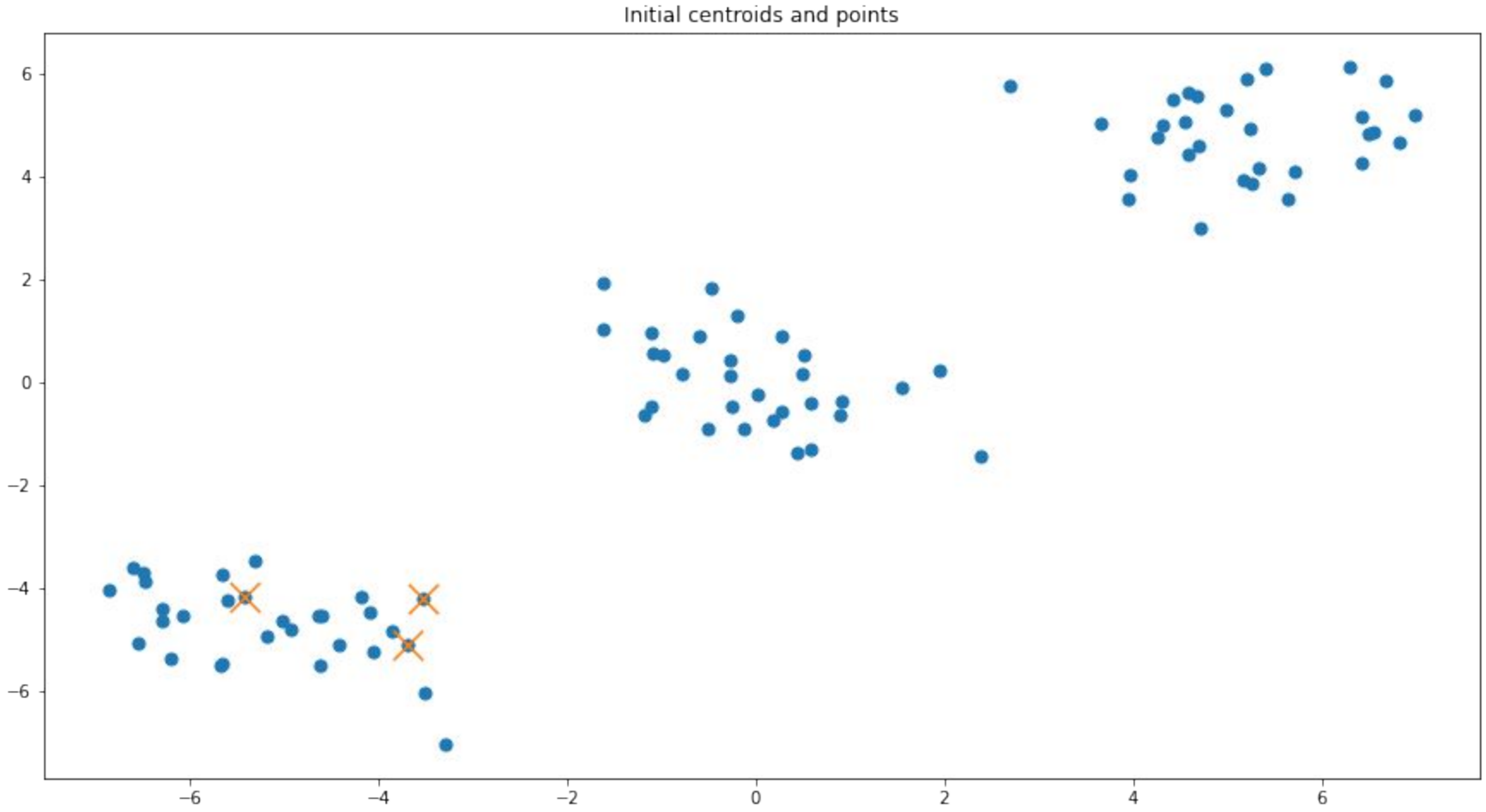
Random initialization

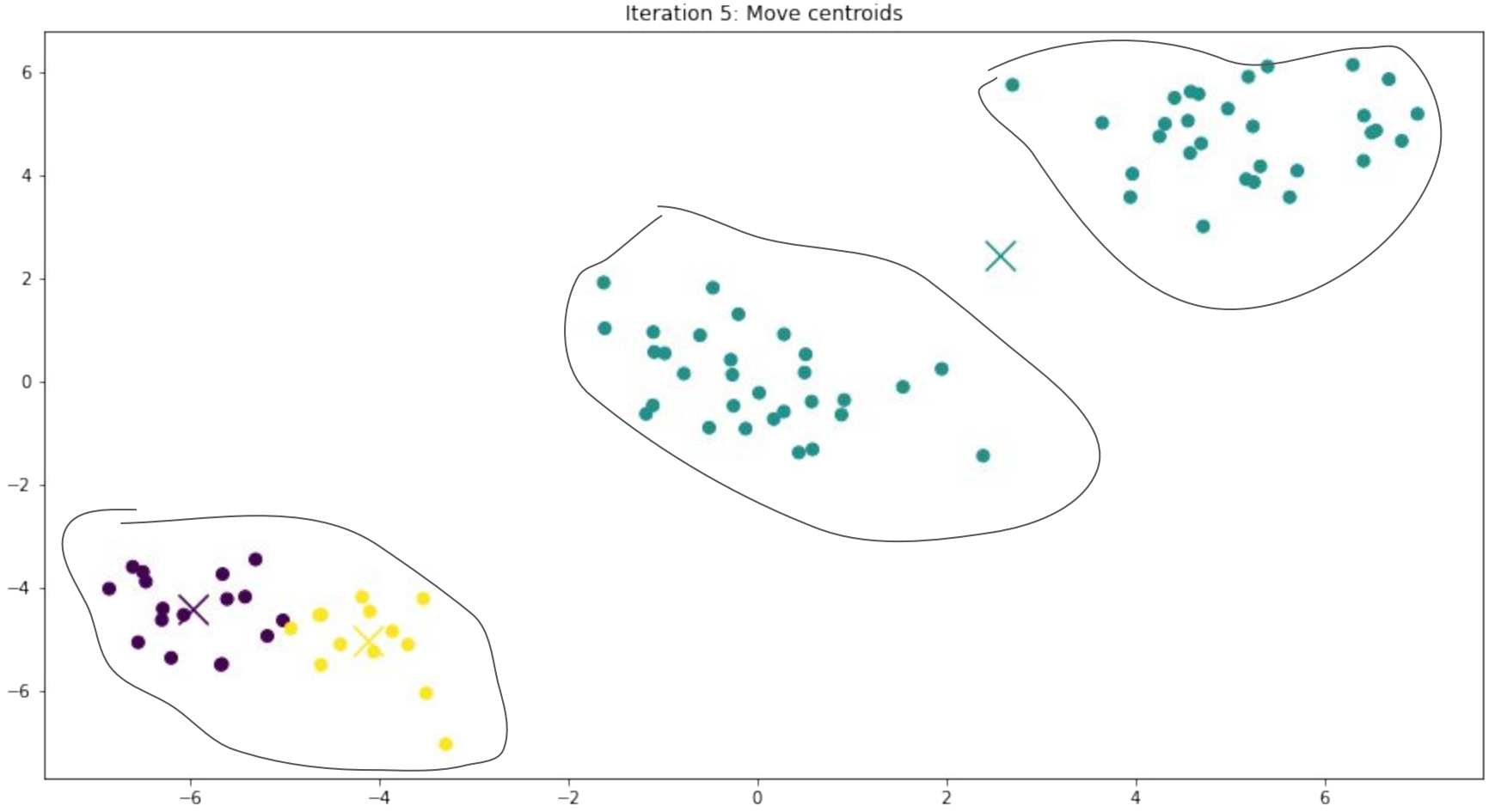
- Randomly pick *K* training examples
- Set $\mu_1, \mu_2, \ldots, \mu_k$ equal to these *K* examples

Results may depend on the random selection

Some iterations may converge to **local optima**

Solution is to run some fixed number of iterations and pick the clustering with the least distortion


Initial centroids and points
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


Iteration 5: Move centroids
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

K-Means++

It is a way to choose the initial seed points in an efficient way

Guarantees an $O(\log k)$ competitive solution as compared to the optimal solution

K-Means++

The algorithm:

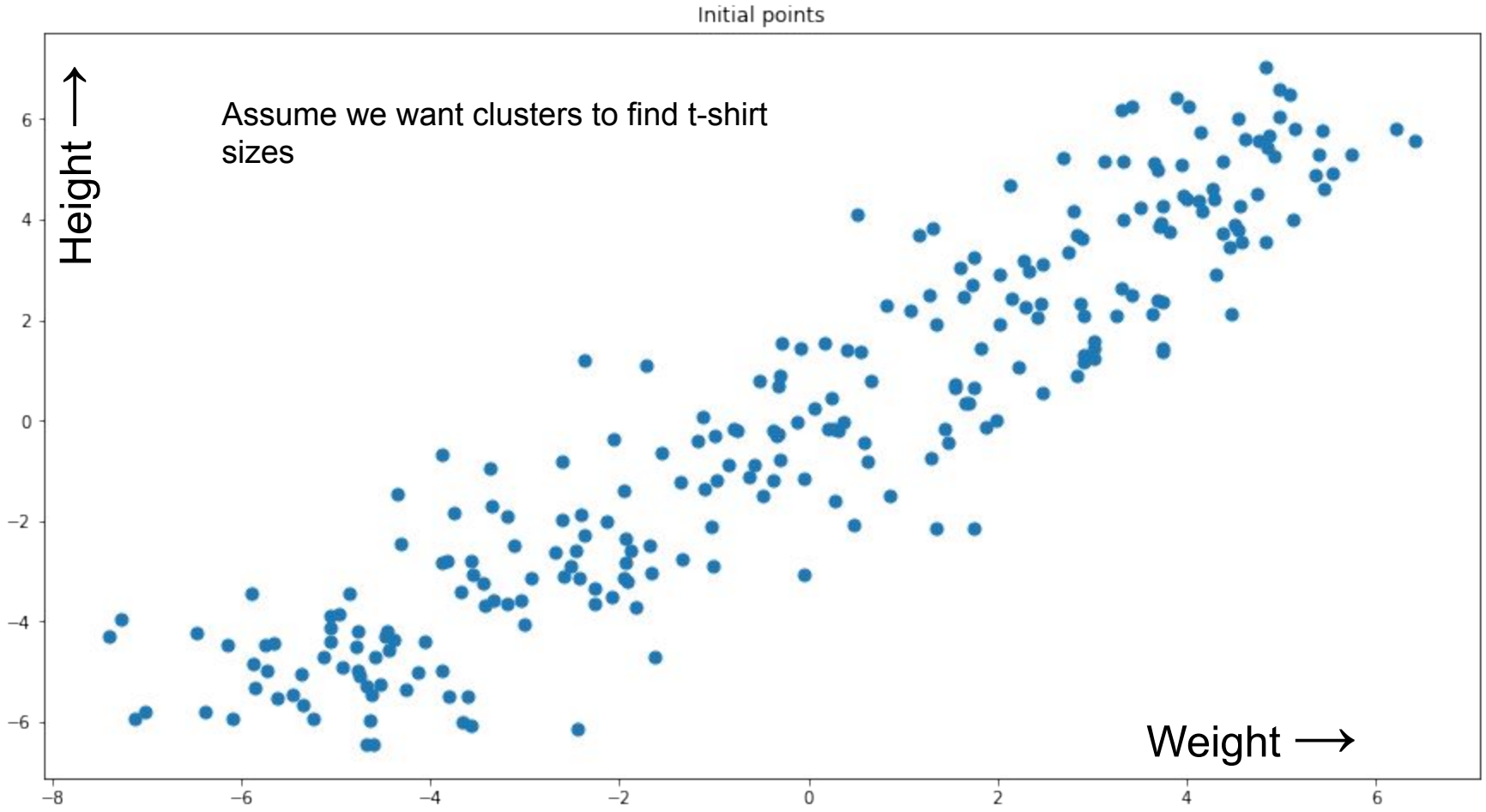
1. Choose one center uniformly at random among the data points
2. For each data point x not chosen yet, compute $D(x)$, the distance between x and the nearest center that has already been chosen
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$
4. Repeat Steps 2 and 3 until k centers have been chosen
5. Now that the initial centers have been chosen, proceed using standard K-Means

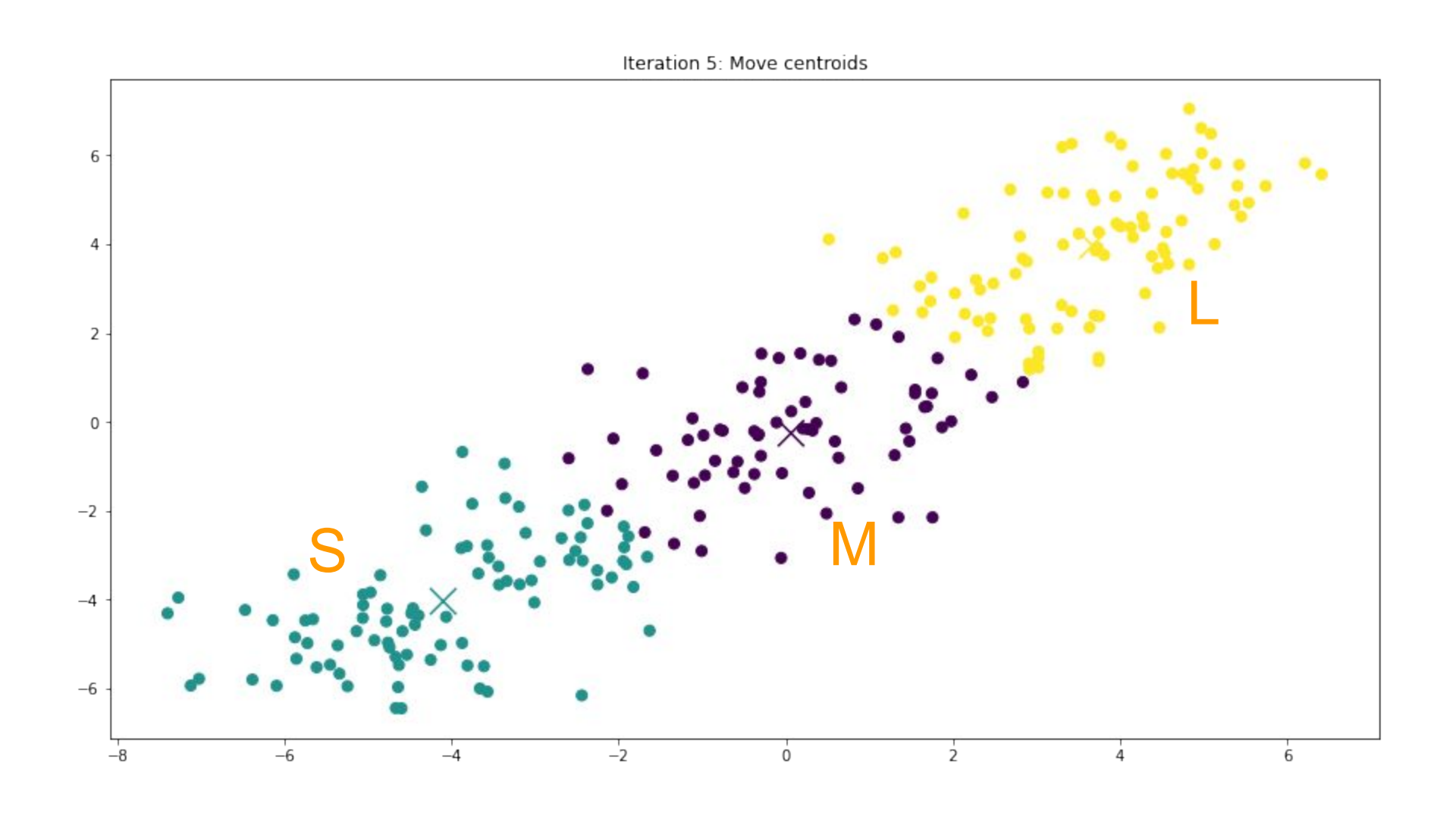
Choosing the value of k

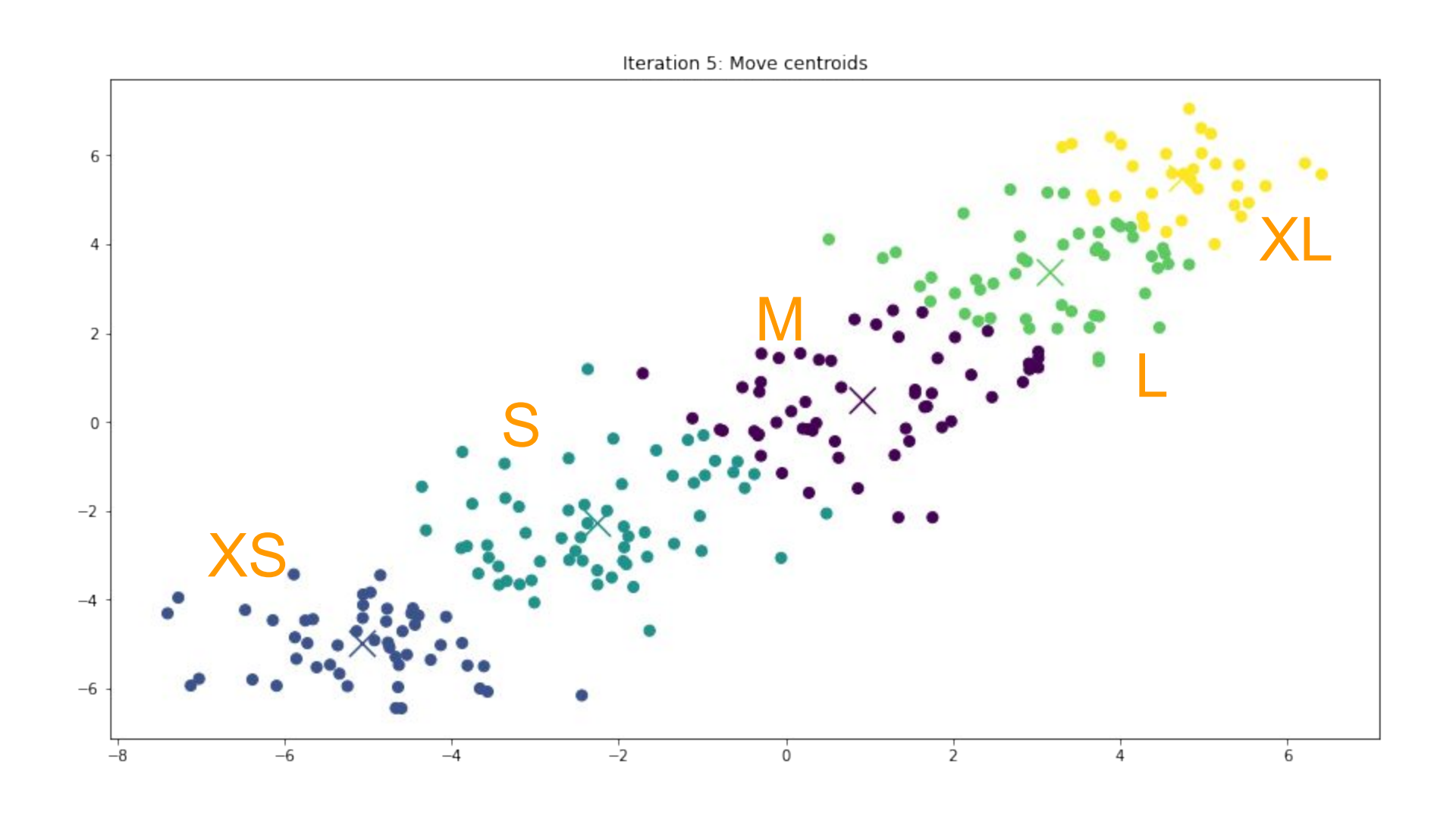
Most of the times we are using K-Means for a downstream task in a pipeline

We should evaluate the algorithm (i.e. the value of K) depending on the performance in that task (example in next slides)

However, there are also some methods for helping us choose a suitable K

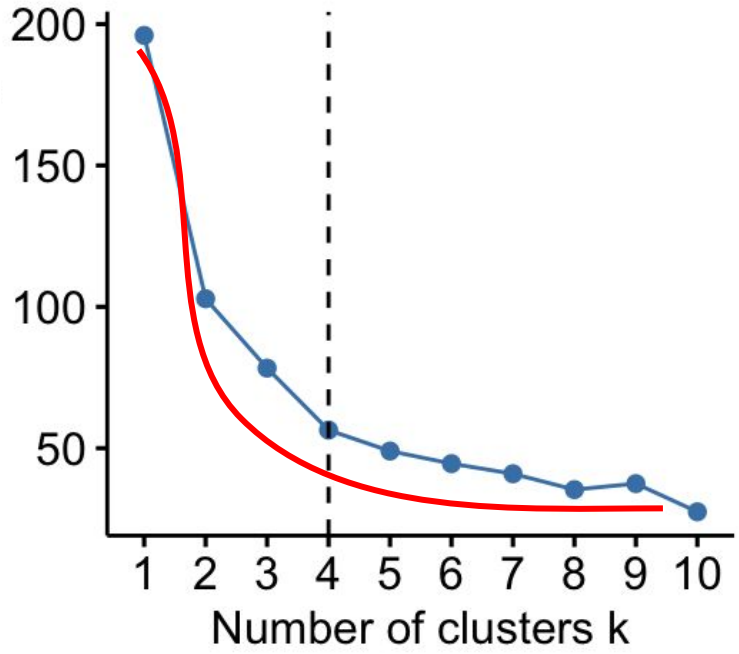

Initial points
Assume we want clusters to find t-shirt sizes
Height ⟶
Weight ⟶
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


Iteration 5: Move centroids
S
M
L
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6


Iteration 5: Move centroids
XS
S
M
L
XL
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

Elbow method

- Plot the objective function vs different values of k
- The “elbow” in the corresponding graph provides the suitable value of k



Silhouette score

The Silhouette Coefficient is calculated using

- the mean intra-cluster distance (a) and
- the mean nearest-cluster distance (b)

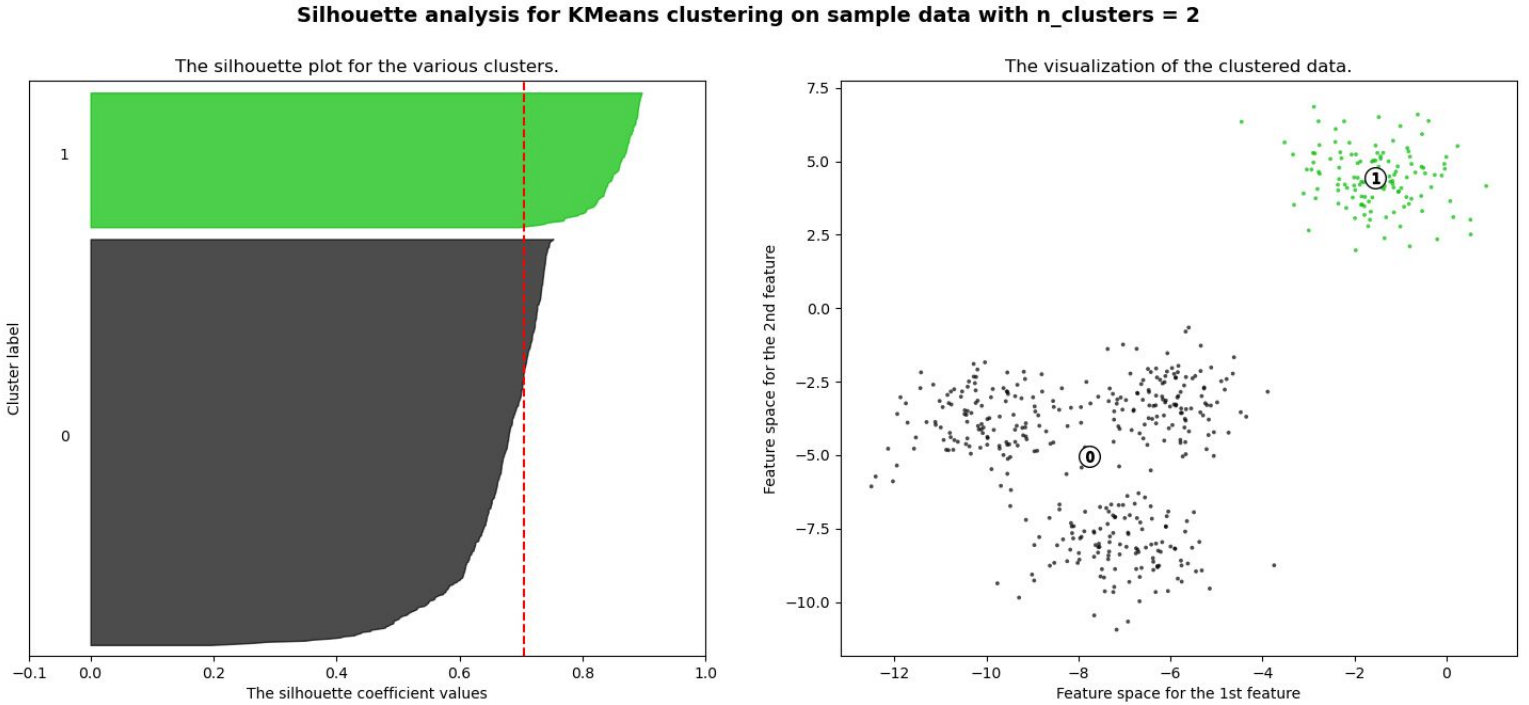
for each sample

The Silhouette Coefficient for a sample is *(b - a) / max(a, b)*

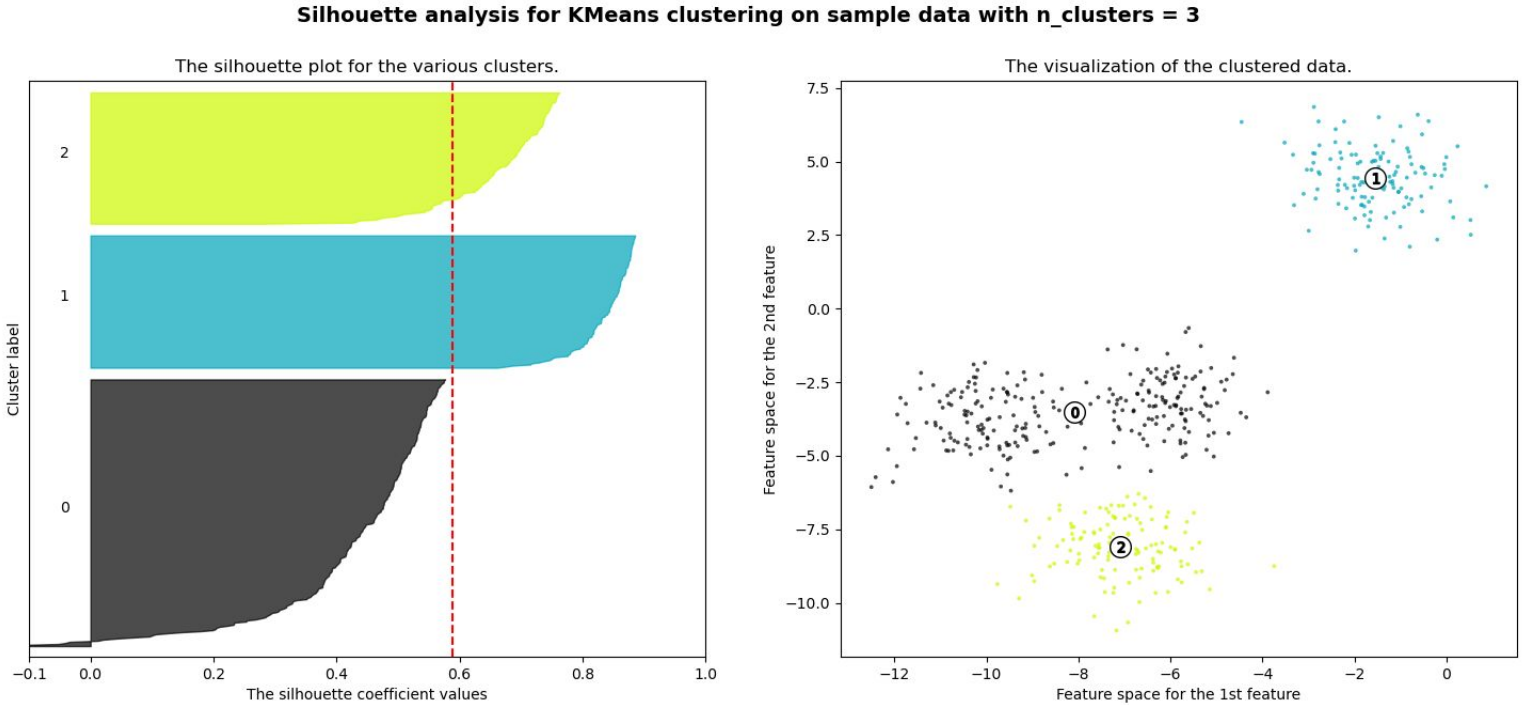
To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of

A value near to 1 represents good clustering while a value near to -1 represents bad clustering

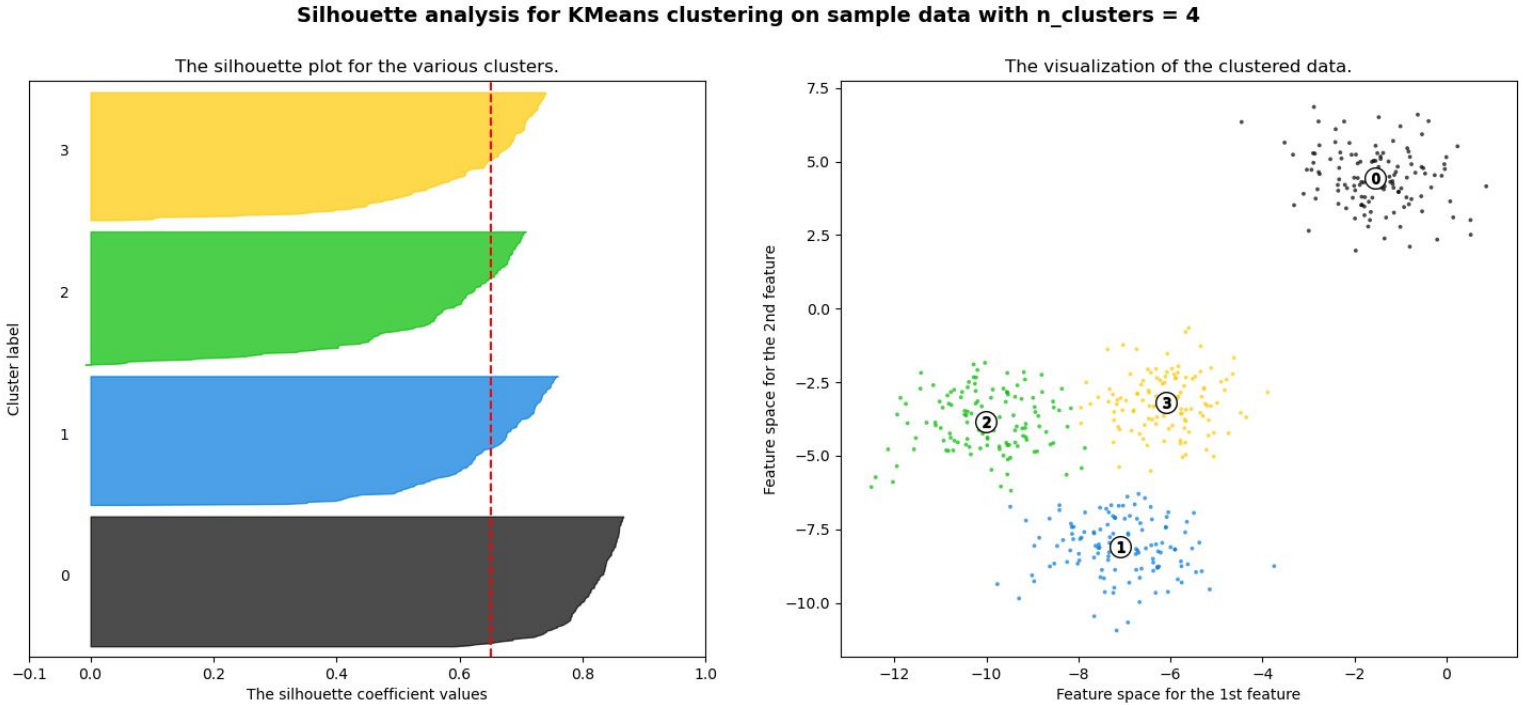
Silhouette score



Silhouette score



Silhouette score

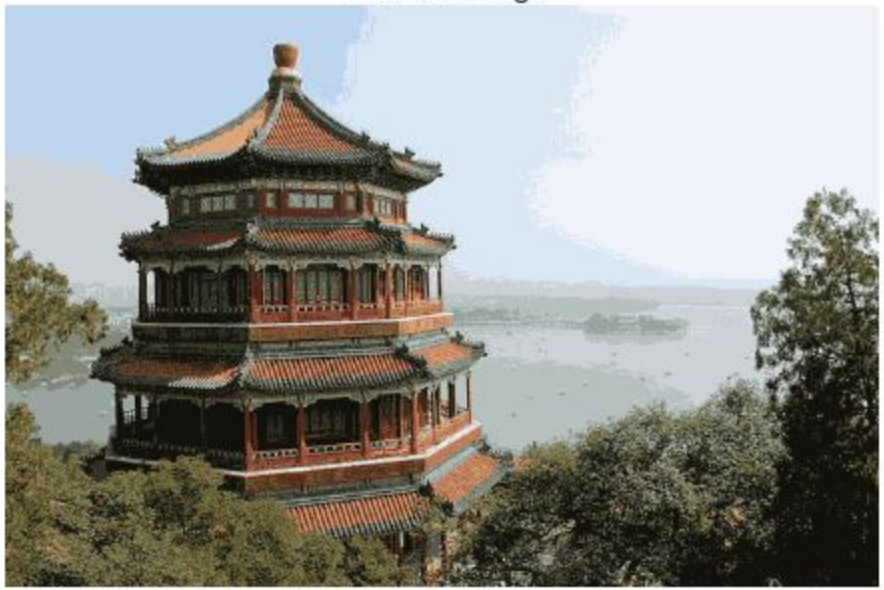


Applications

- Cluster Analysis
  - Market segmentation
  - Social network analysis
  - Organizing computing clusters
  - Astronomical data analysis
- Vector quantization
  - Image compression
  - Image segmentation
  - Non-random sampling

Original Image

16-color Image

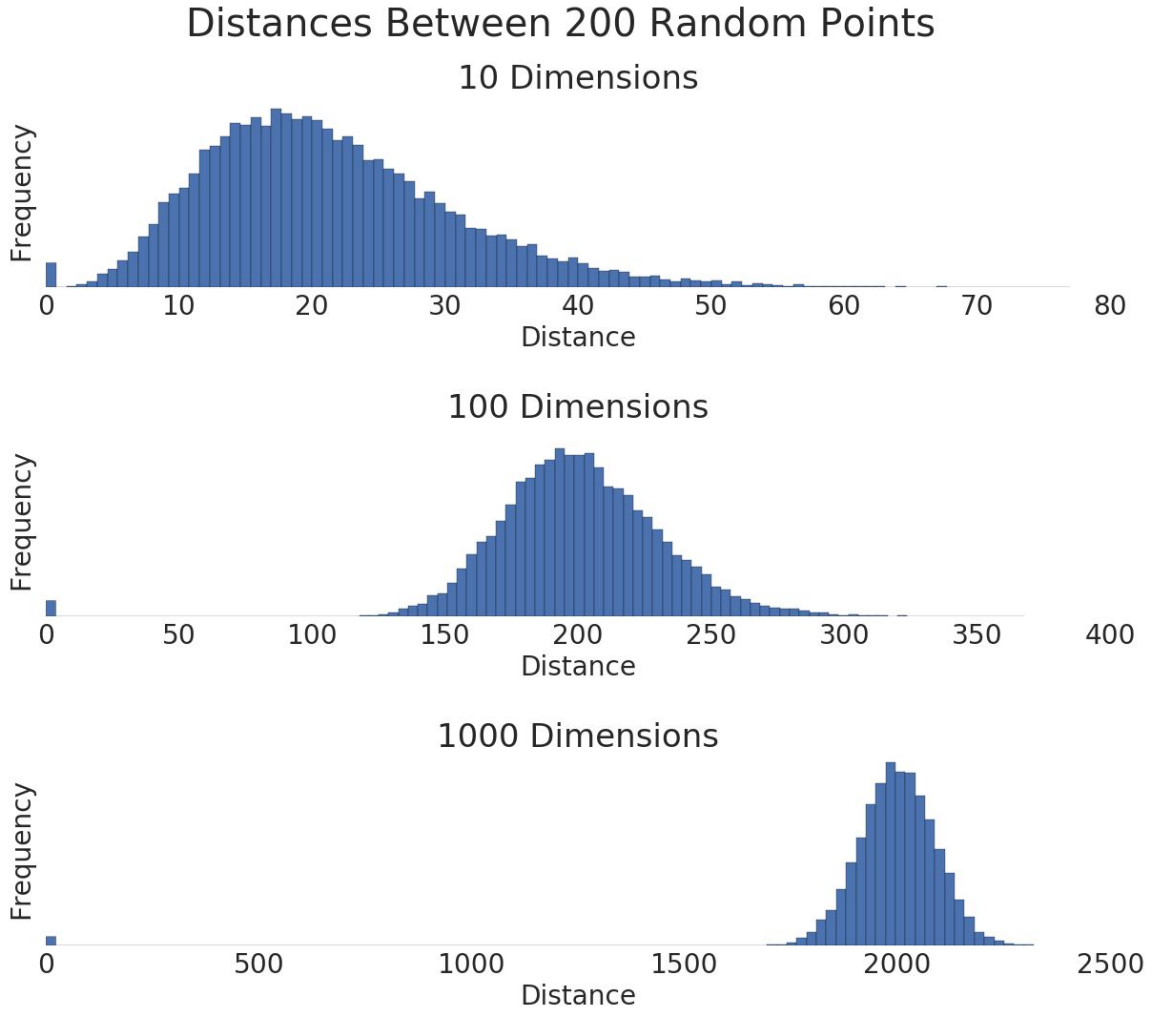


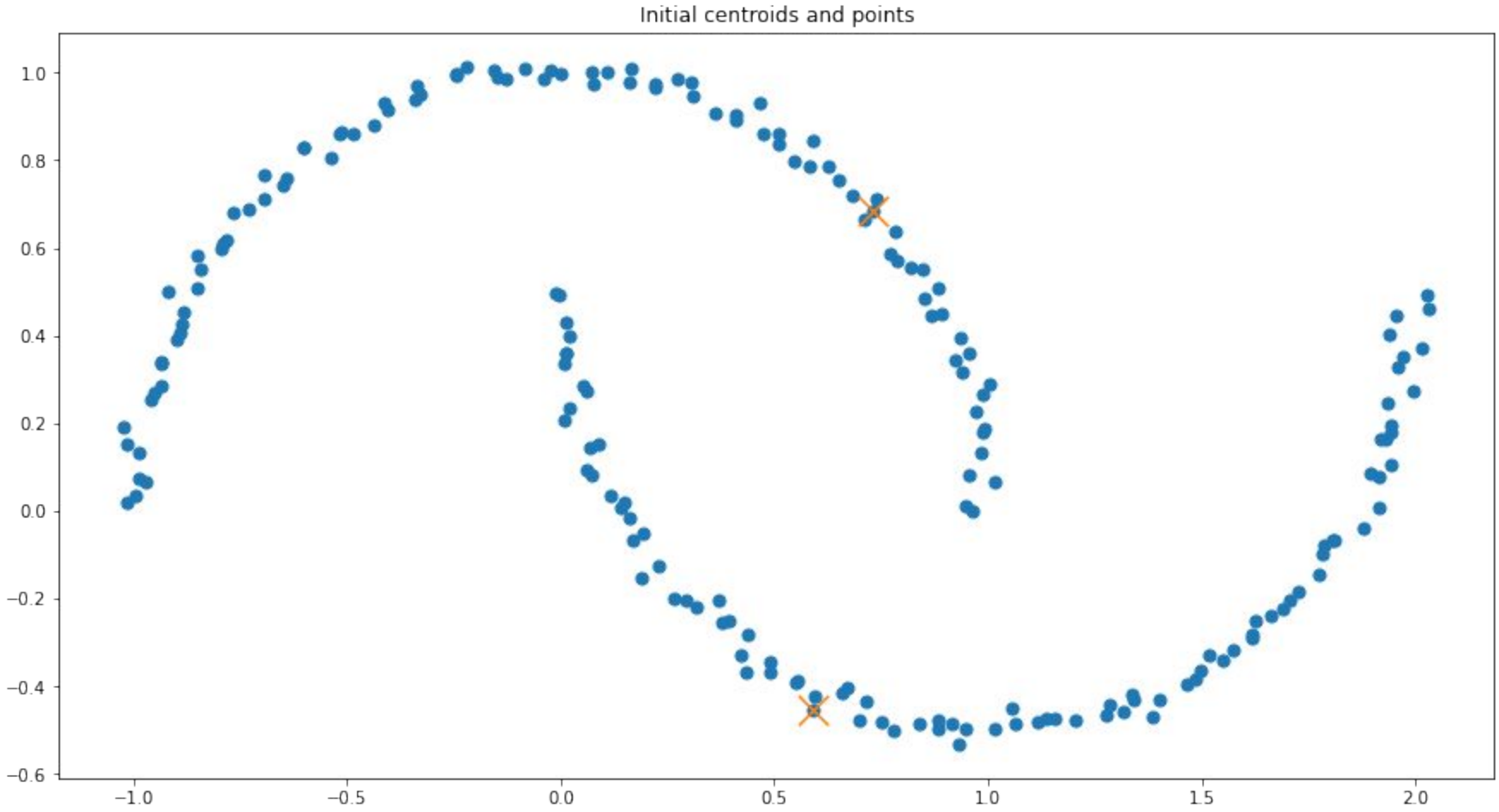
Advantages

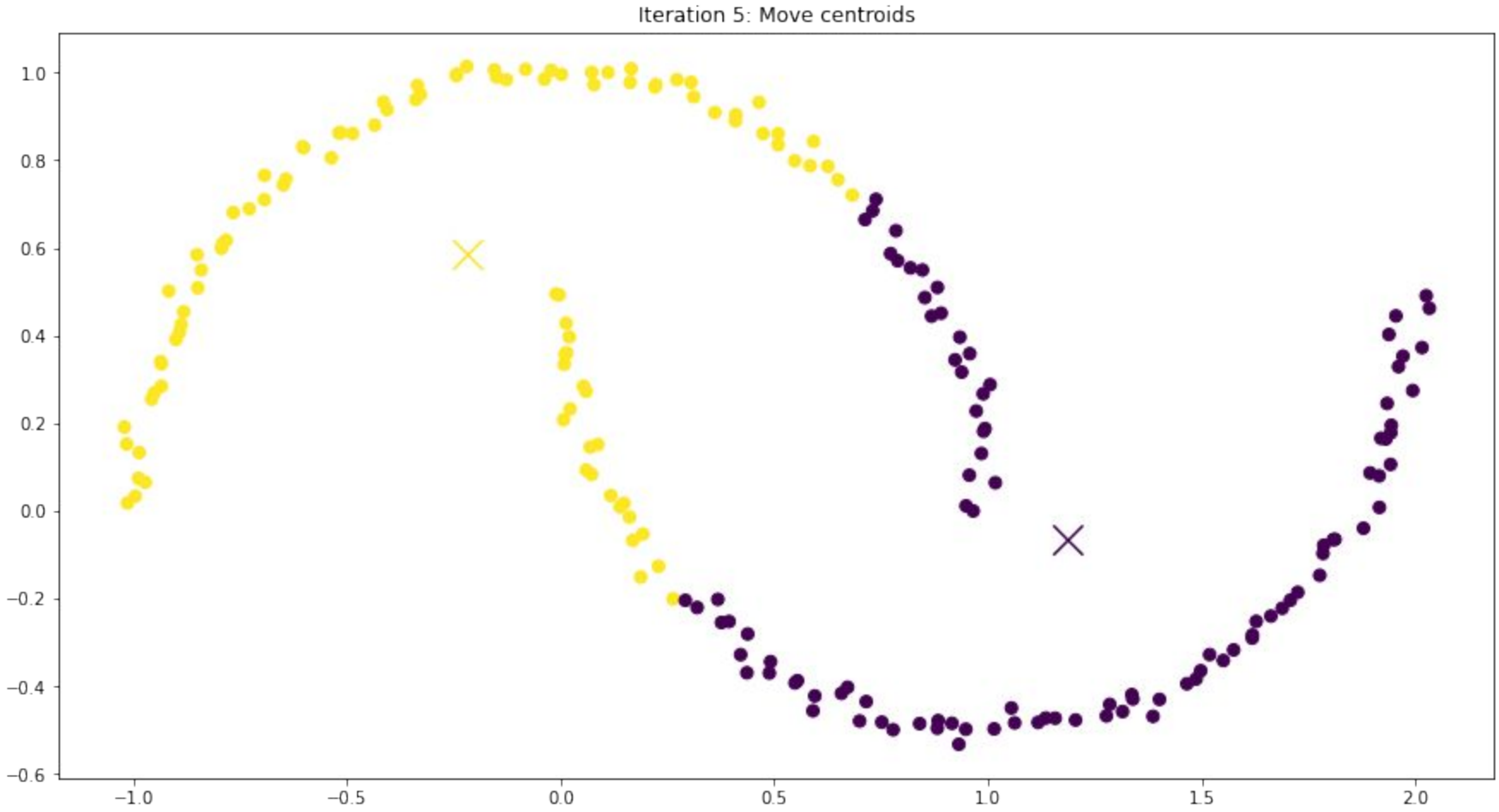
- Easy to implement
- Works quite well in practice
- Scales to large datasets (provided k is reasonably small)
- Produces tighter clusters
- Guarantees convergence
- Can warm-start the position of centroids
- Adapts to new data points
- Instances can change clusters as the algorithm proceeds

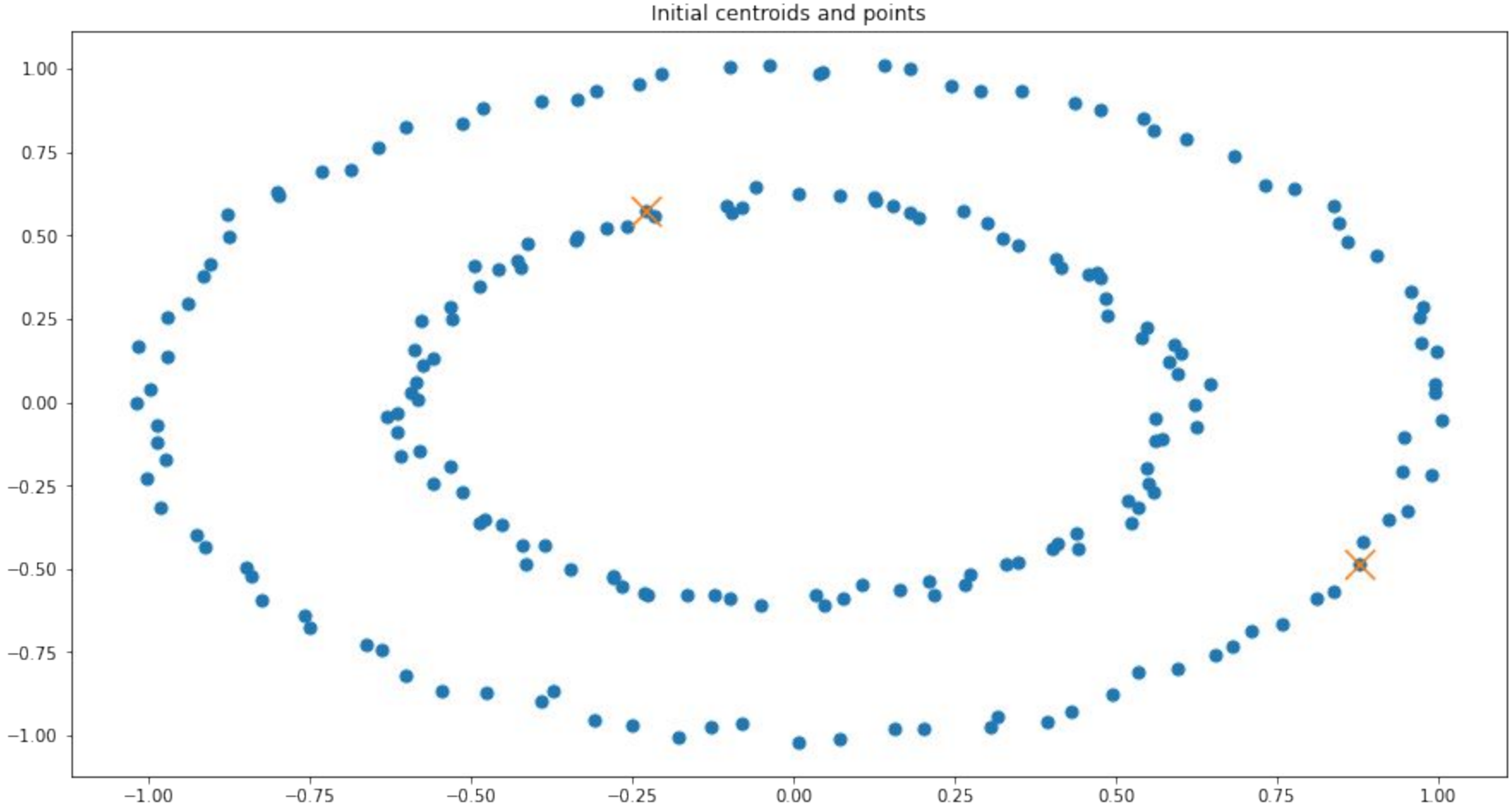
Disadvantages

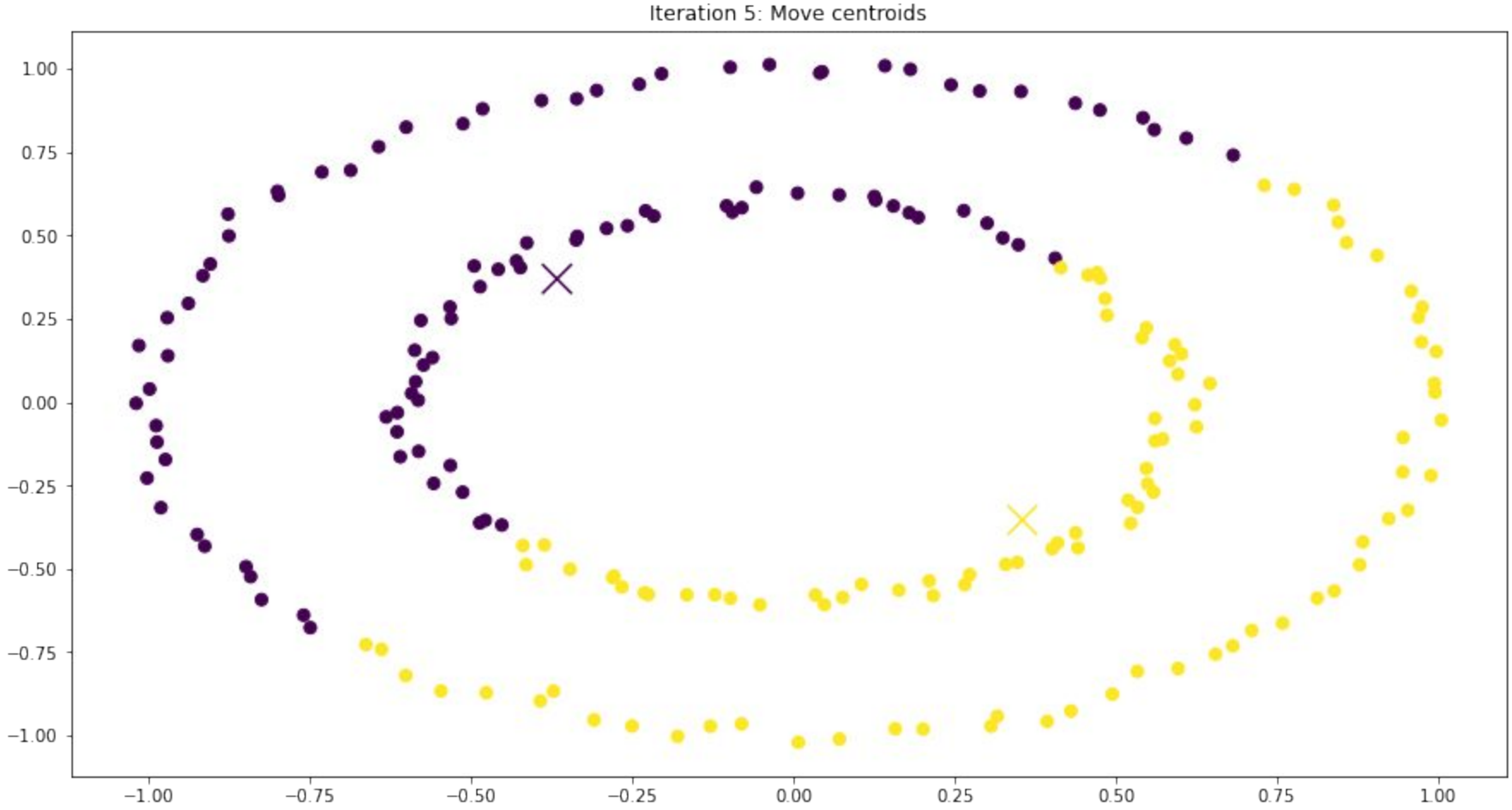
- Difficult to predict the number of clusters
- Depends heavily on the initial seed points (can get stuck at local optima)
- Sensitive to scaling (normalization/standardization)
- Sensitive to outliers
- Sensitive to the order of the data points
- Good for roughly spherical clusters, but worse for some geometric patterns
- Does not scale with many dimensions i.e. as the number of dimensions increases, a distance based similarity measure converges to a constant (**curse of dimensionality**)
- Doesn’t tell whether clustering is possible or not


Distances Between 200 Random Points
10 Dimensions
Frequency
0
10
20
30
40
50
60
70
80
Distance
100 Dimensions
Frequency
0
50
100
150
200
250
300
350
400
Distance
1000 Dimensions
Frequency
0
500
1000
1500
2000
2500
Distance


Initial centroids and points
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Iteration 5: Move centroids
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Initial centroids and points
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

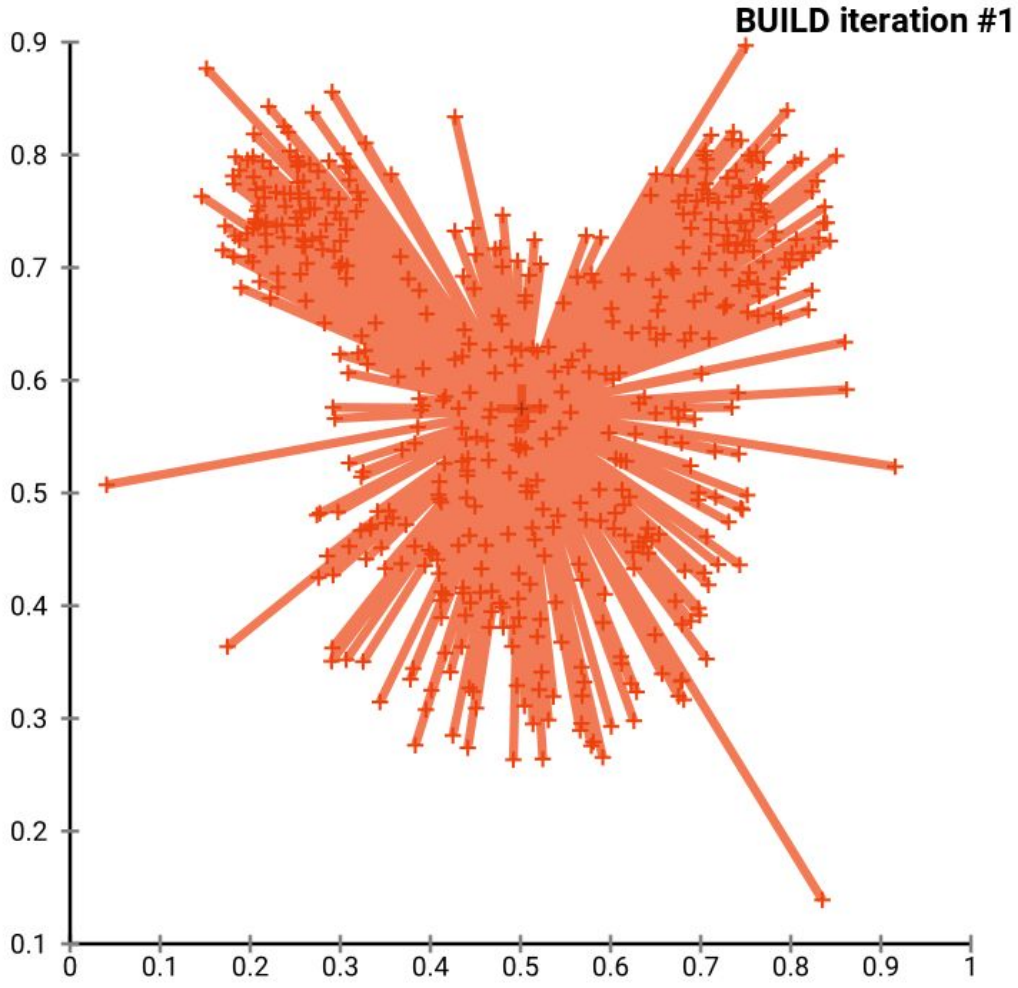

Iteration 5: Move centroids
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Variations

K Medoids

1. **(BUILD)** Initialize: greedily select k of the n data points as the medoids to minimize the cost
2. Associate each data point to the closest medoid.
3. **(SWAP)** While the cost of the configuration decreases:
    1. For each medoid m, and for each non-medoid data point o:
        1. Consider the swap of m and o, and compute the cost change
        2. If the cost change is the current best, remember this m and o combination
    2. Perform the best swap of m_{best} and o_{best}, if it decreases the cost function. Otherwise, the algorithm terminates.

K Medoids


BUILD iteration #1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Fuzzy Clustering

Each data point can belong to more than one cluster.

Each point has a membership score with respect to a particular cluster index. Clusters are assigned based on the maximum membership score.

One of the most widely used fuzzy clustering algorithms is the Fuzzy C-means clustering (FCM) Algorithm.

Fuzzy C-means clustering

- Choose a number of clusters.
- Assign coefficients randomly to each data point for being in the clusters.
- Repeat until the algorithm has converged (that is, the coefficients' change between two iterations is no more than ϵ, the given sensitivity threshold) :
  - Compute the centroid for each cluster.
  - For each data point, compute its coefficients of being in the clusters.

Fuzzy C-means clustering

Any point x has a set of coefficients giving the degree of being in the k^{th} cluster $w_k(x)$.

The centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster, or, mathematically,

$$c_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m}$$

where m is the hyperparameter that controls how fuzzy the cluster will be. The higher it is, the fuzzier the cluster will be in the end.

Thank you