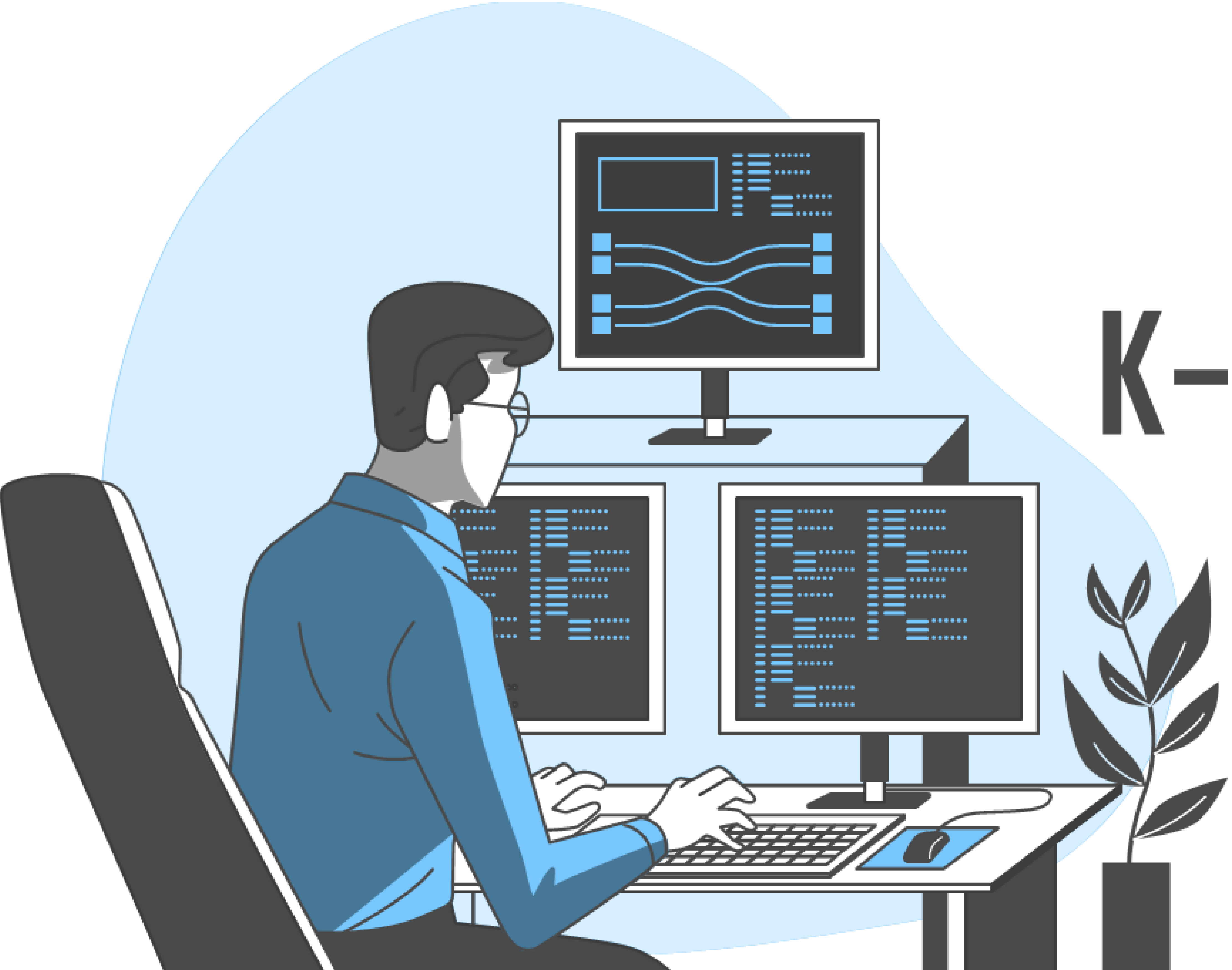


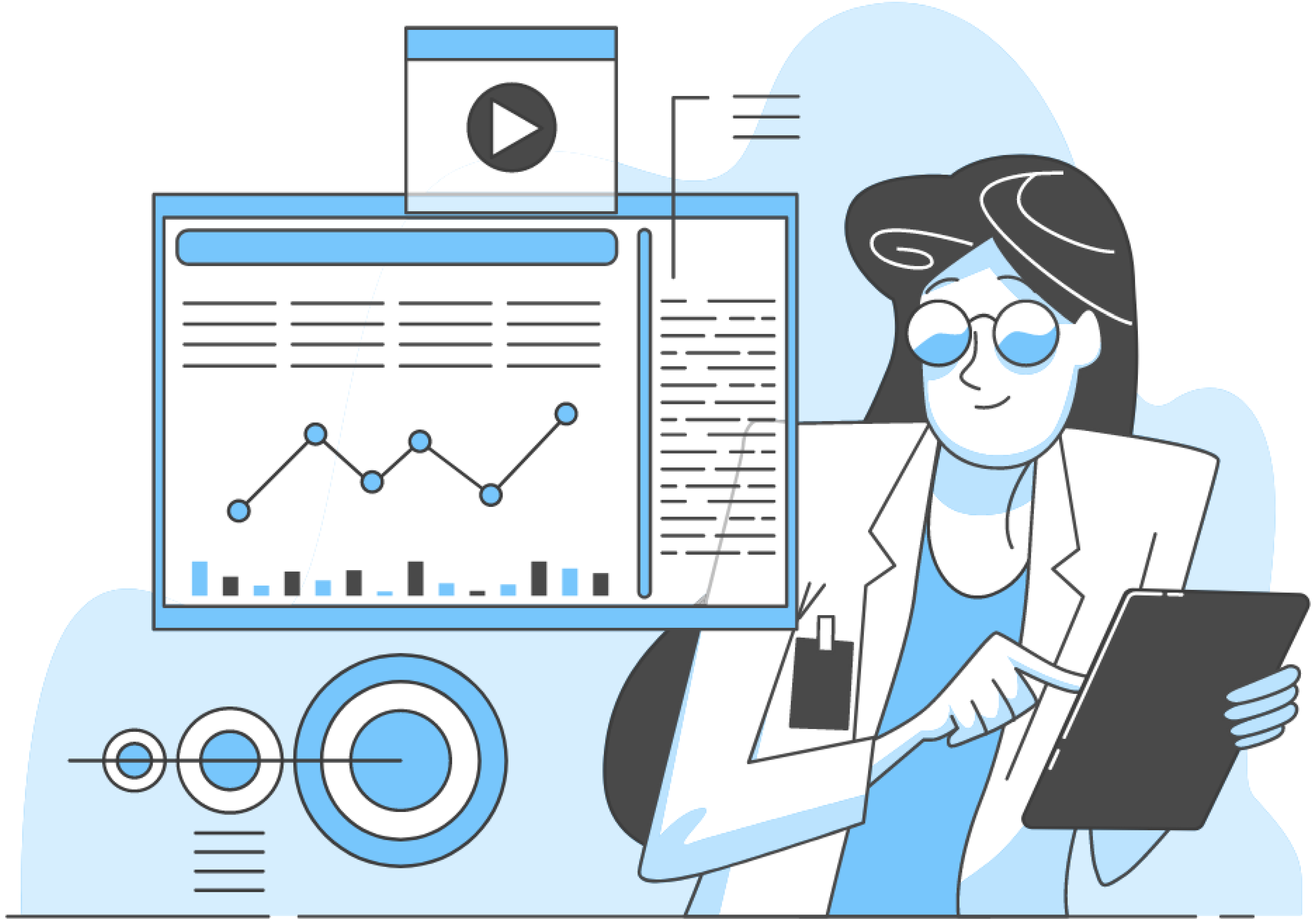
K-MEANS BASED METHOD

Group 6 - CC01

Group members

Nguyễn Mỹ Khanh	Lê Khánh Duy	Huỳnh Thanh Phúc
2052525	2052003	2052211

Table of Contents



01

INTRODUCTION

Clustering

- **Clustering:** the process of partitioning a set of data objects (or observations) into subsets.

- Each subset is a **cluster**. Data in a cluster are similar to one another, yet dissimilar to objects in other clusters.

- Different clustering methods may generate different clusterings on the same data set.

Problems for Cluster Analysis

- **Scalability:** big data requires methods to scale.

- **Variety of attributes:** different type of data: nominal, numerical, ...

–> methods should work well with complex data types.

- **Cluster shape**: methods have assumptions about cluster shapes they work with – No free lunch theorem.

Problems for Cluster Analysis

- **Noisy data:** Clustering algorithms can be sensitive to noise, outliers.
- **High-dimensional data:** high-dimensional data can be very sparse and highly skewed.

Methods

Grid-based

Density-based

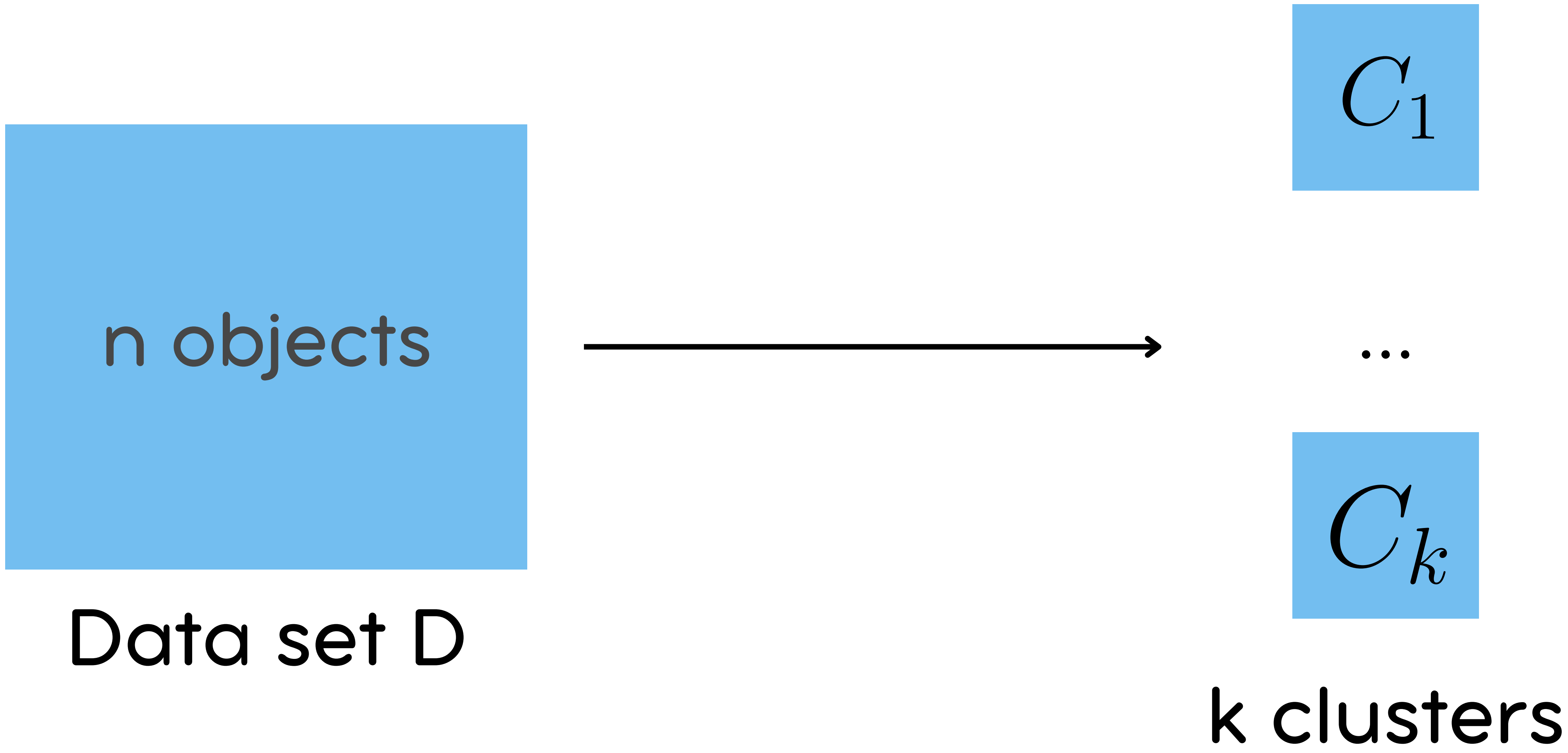
Partitioning

Hierarchical

02

K-means Clustering

Problems



- $k < n$
- Each object belongs to only one cluster.

- We want points close to each other in the same cluster.
- Use Euclidean distance to define "close"
- Define centroid on each cluster and try to minimize cluster points distance to them.

$$\phi = \sum_{i=1}^{k} \sum_{p \in C_i} \|p - c_i\|_2^2$$

Where:

- Φ is the sum of the squared error/distance
- p is ta data point
- c_i is the centroid of cluster C_i

Algorithm

Input:

- k: the number of clusters,
- D: a data set containing n objects.

Output:

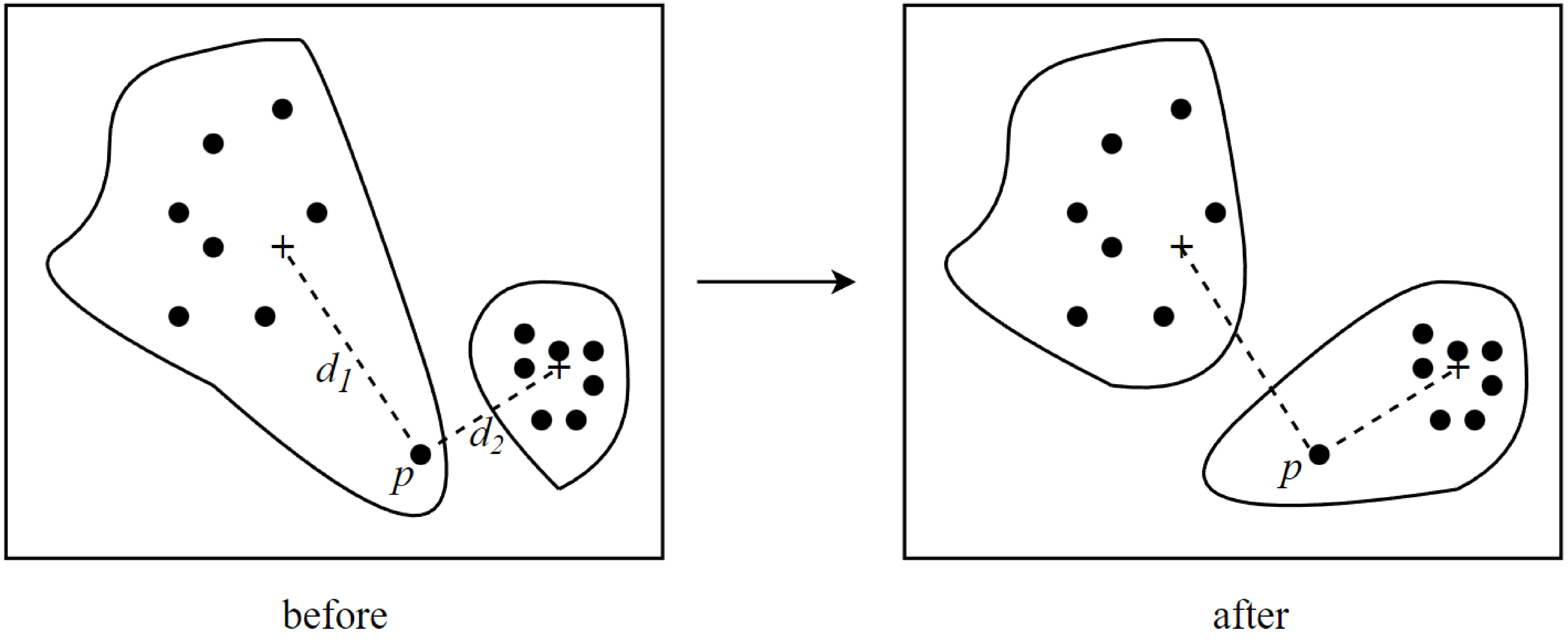
- A set of k clusters.

Lloyd algorithm:

(1) arbitrarily choose k objects from D as the initial centroids
(2) repeat until convergence:
(3) (re)assign each object to its closest cluster/centroid
(4) update each cluster centroid to its center of mass

Reasoning

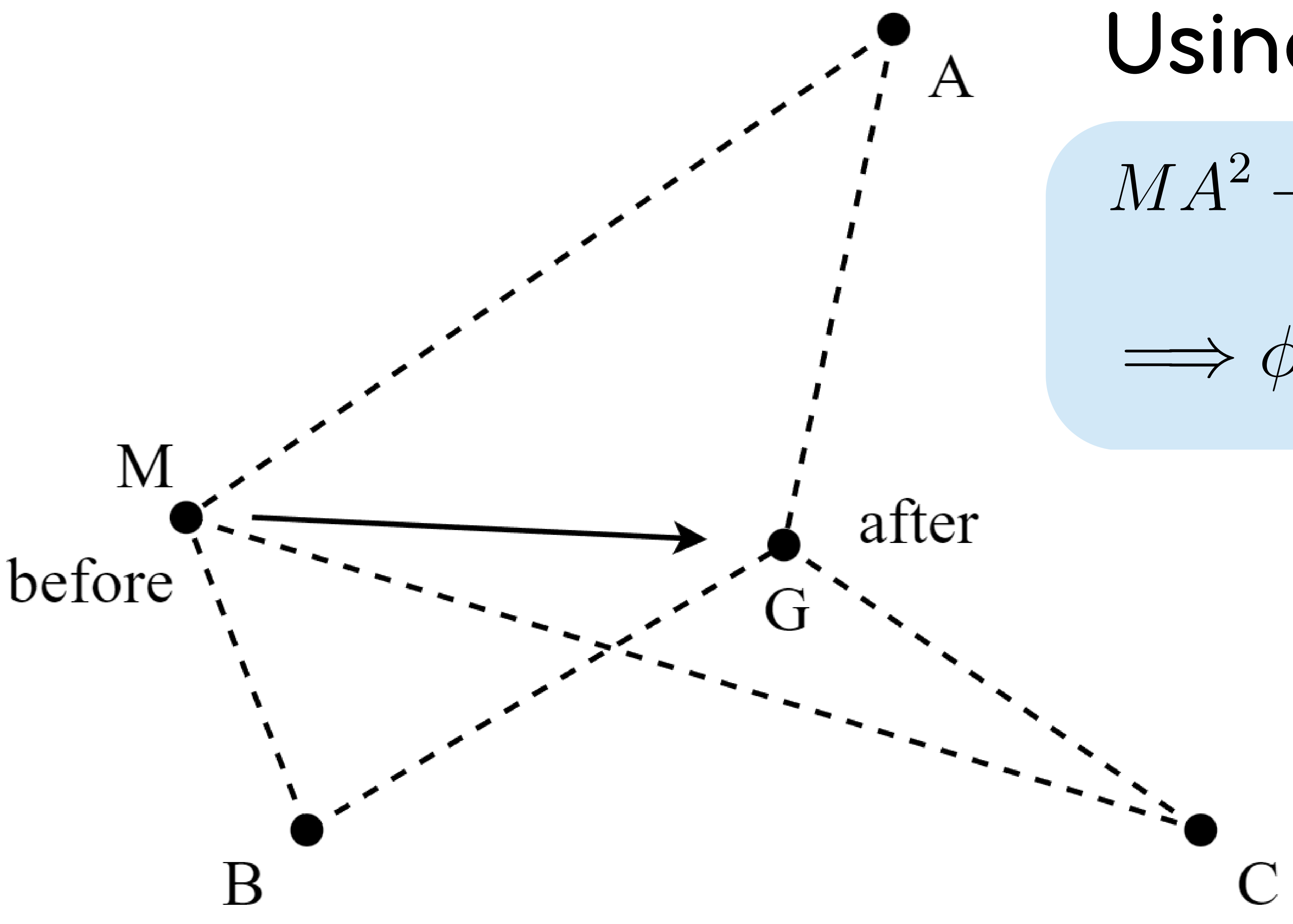
Step 3: assign



$$\phi_{\text{before}} - \phi_{\text{after}} = d_1 - d_2 \geq 0$$

Reasoning

Step 4: update centroid



Using linear algebra or geometry:

$$MA^2 + MB^2 + MC^2 = 3MG^2 + GA^2 + GB^2 + GC^2$$

$$\Longrightarrow \phi_{\text{before}} = \phi_{\text{after}} + 3MG^2 \geq \phi_{\text{after}}$$

Reasoning

Both steps 3 and 4 reduce the loss value

-> The algorithm converges

Example

Example 1 : K-means algorithm implementation using k = 2

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Example

Step 1: Initialization: Randomly we choose following two centroids (k=2) for two clusters.
-> In this case the 2 centroids are: m1 = (1.0, 1.0) and m2 = (5.0,7.0).

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Example

Step 2: Calculate distance between centroid and data point

$$d(m_1, 2) = \sqrt{(1. - 1.5)^2 + (1. - 2.)^2} = 1.12$$
$$d(m_2, 2) = \sqrt{(5. - 1.5)^2 + (7. - 2.)^2} = 6.10$$

Individual	Centroid m1	Centroid m2
1(1.0,1.0)	0	7.21
2(1.5,2.0)	1.12	6.1
3(3.0,4.0)	3.61	3.61
4(5.0,7.0)	7.21	0
5(3.5,5.0)	4.72	2.5
6(4.5,5.0)	5.31	2.06
7(3.5,4.5)	4.3	2.92

Example

Step 2:

New clusters:

{1,2,3} and {4,5,6,7}.

New centroids:

m1=((1.0+1.5 +3.0)/3 , (1.0+ 2.0+4.0)/3) = (1.83,2.3)

m2=((5.0+3.5+4.5+3.5)/4, (7.0+5.0+5.0+4.5))/4 = (4.12,5.38)

Individual	Centroid m1	Centroid m2
1(1.0,1.0)	0	7.21
2(1.5,2.0)	1.12	6.1
3(3.0,4.0)	3.61	3.61
4(5.0,7.0)	7.21	0
5(3.5,5.0)	4.72	2.5
6(4.5,5.0)	5.31	2.06
7(3.5,4.5)	4.3	2.92

Example

Step 3:

New clusters:

{1,2} and {3,4,5,6,7}.

Next centroids:

m1=(1.25,1.5)

m2=(3.9,5.1)

Individual	Centroid m1	Centroid m2
1(1.0,1.0)	1.57	5.38
2(1.5,2.0)	0.47	4.28
3(3.0,4.0)	2.04	1.78
4(5.0,7.0)	5.64	1.84
5(3.5,5.0)	3.15	0.73
6(4.5,5.0)	3.78	0.54
7(3.5,4.5)	2.74	1.08

Example

Step 4: Repeat new clusters: {1,2} and {3,4,5,6,7}

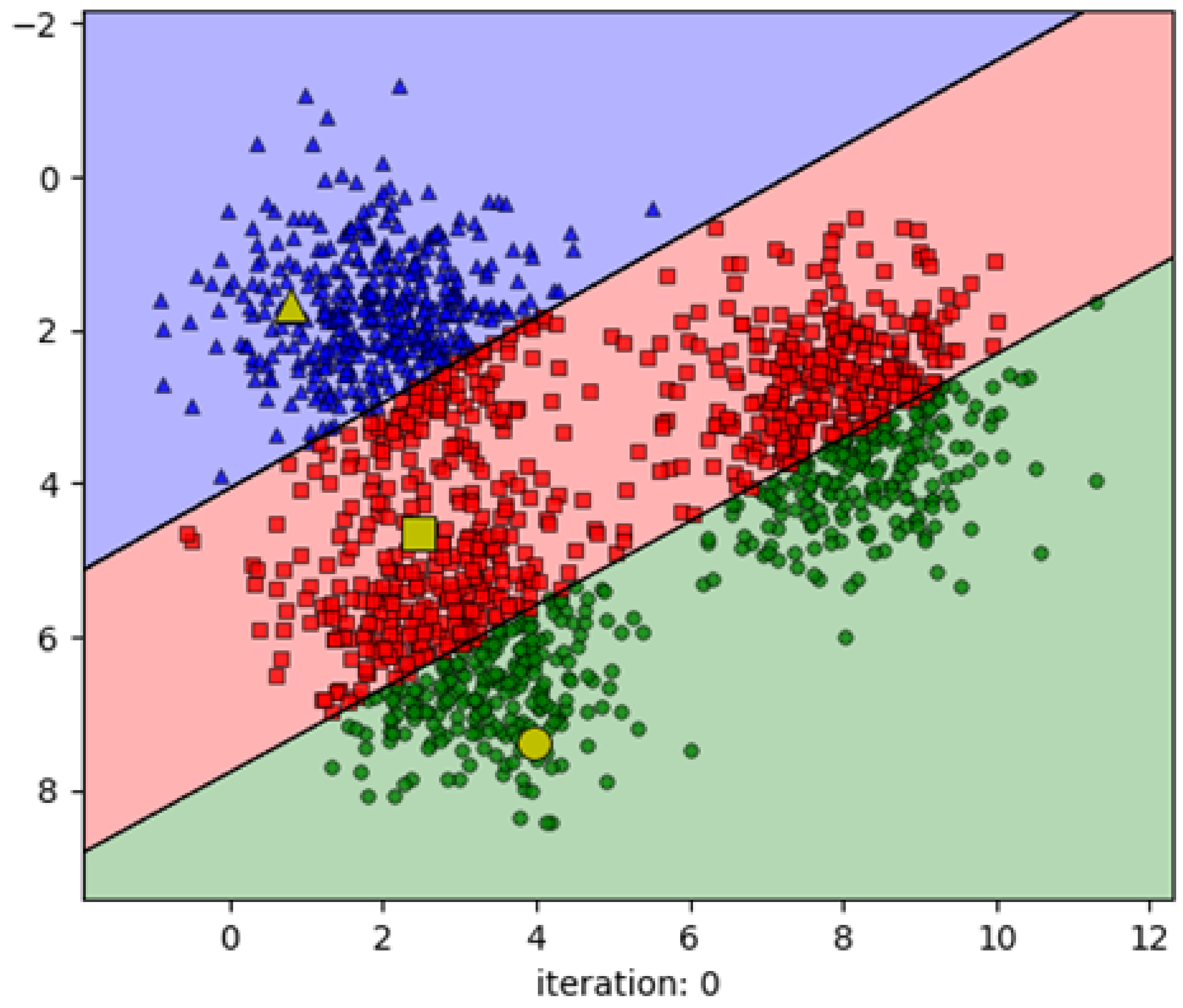
The algorithm halts at the above solution.

Individual	Centroid m1	Centroid m2
1(1.0,1.0)	1.57	5.38
2(1.5,2.0)	0.47	4.28
3(3.0,4.0)	2.04	1.78
4(5.0,7.0)	5.64	1.84
5(3.5,5.0)	3.15	0.73
6(4.5,5.0)	3.78	0.54
7(3.5,4.5)	2.74	1.08

Example

Example 2: K-means algorithm implementation using k = 3 (plot)

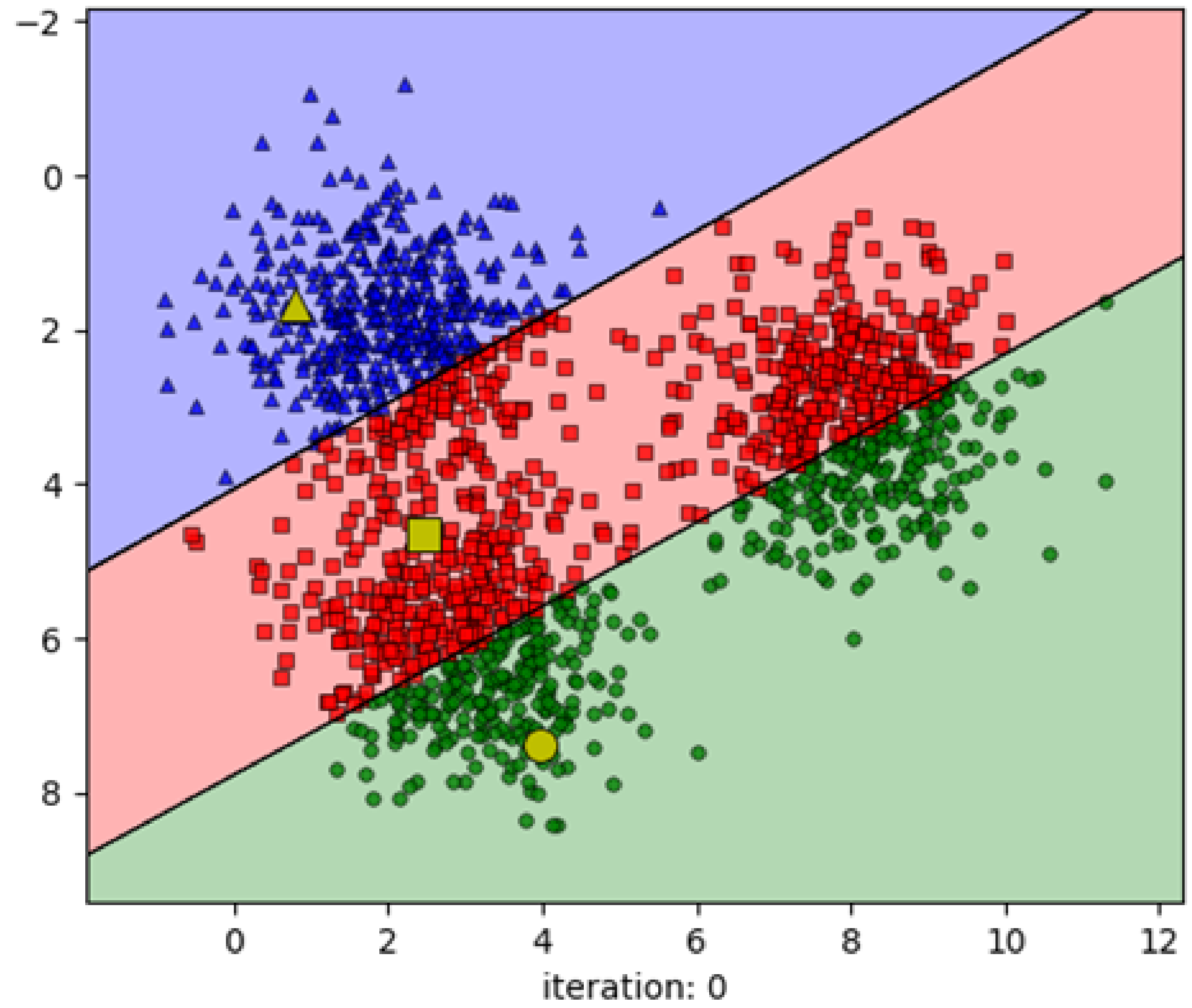
Initiation



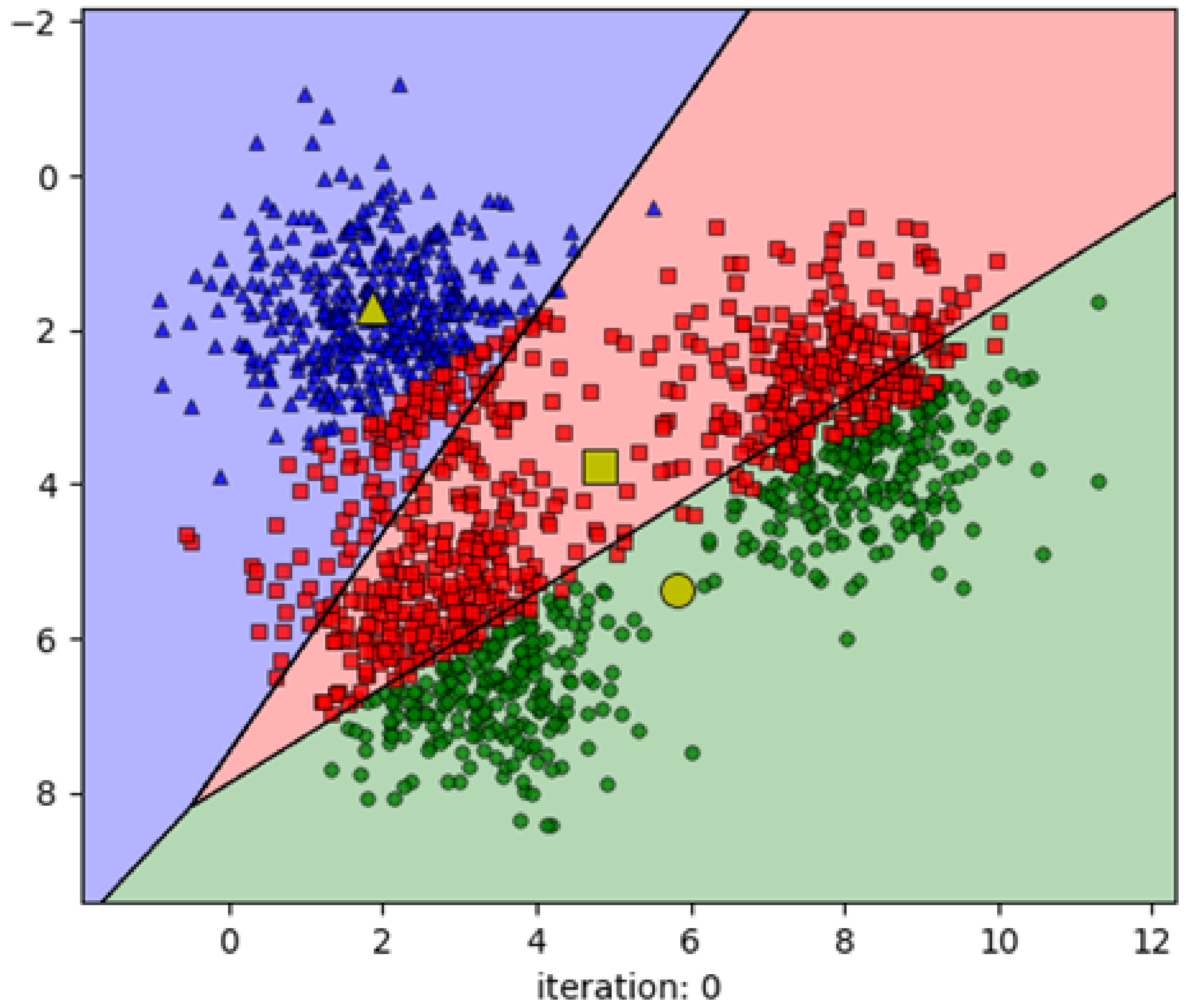
Example

Example 2: K-means algorithm implementation using k = 3 (plot)

Assign



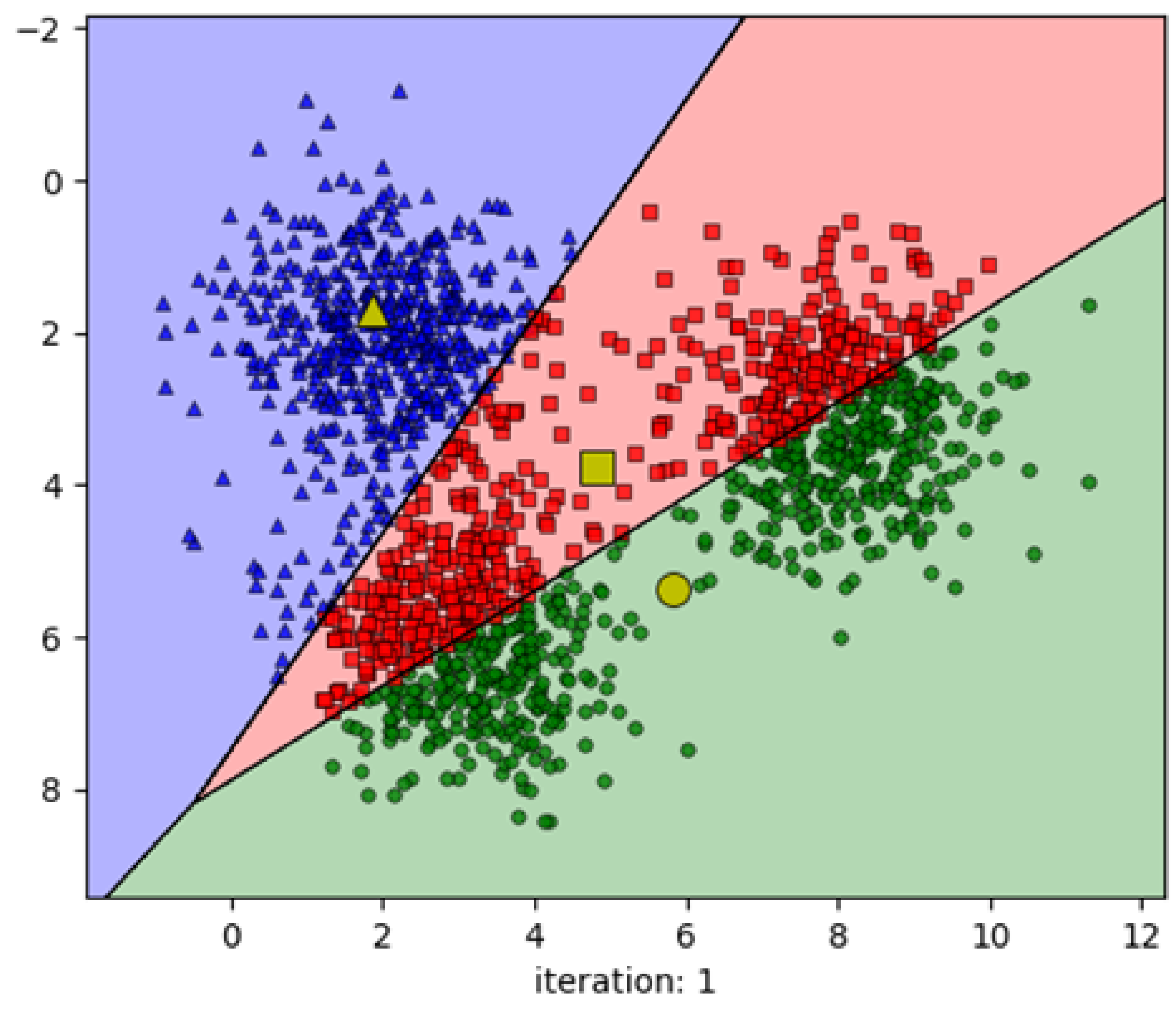
Update



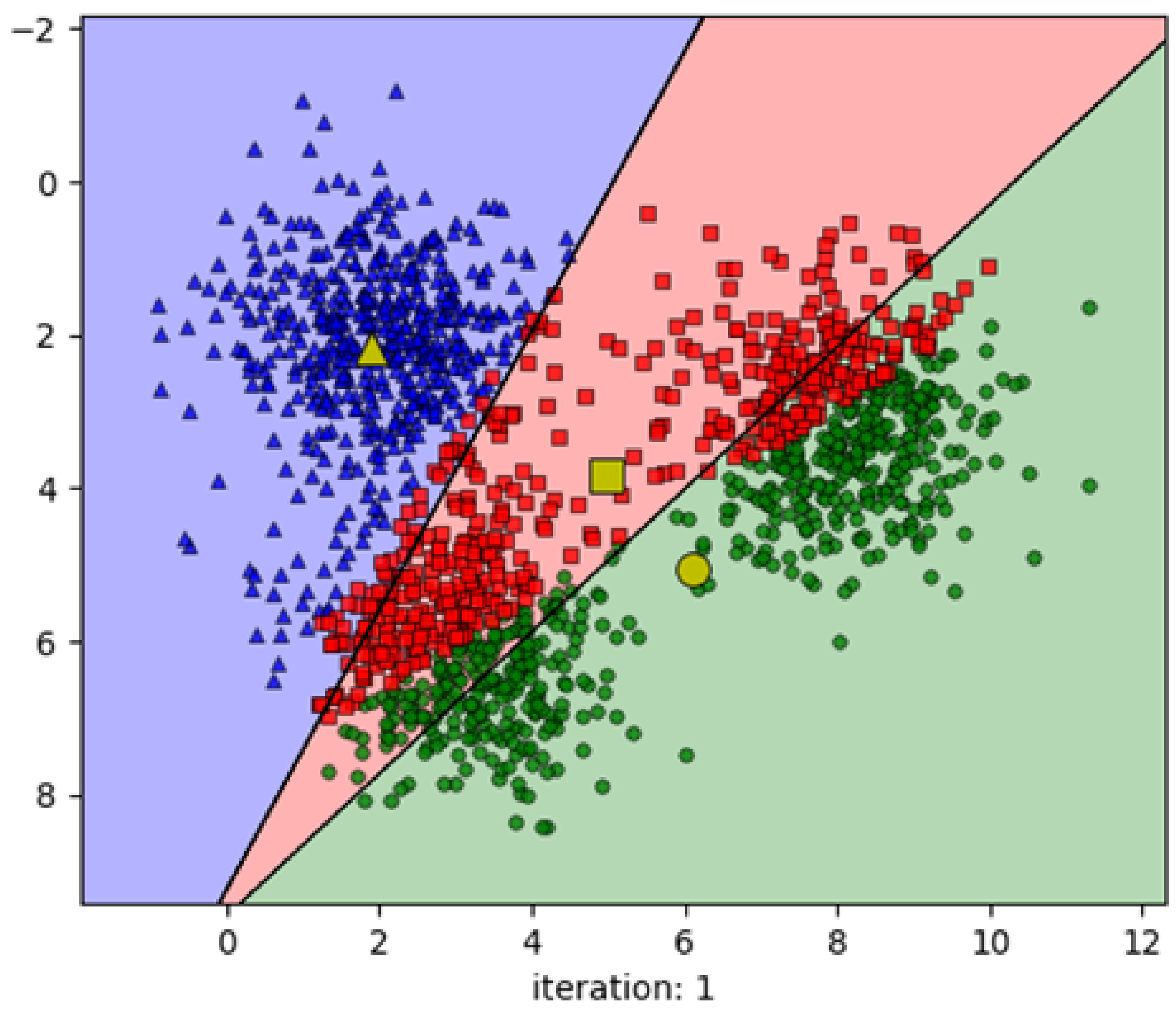
Example

Example 2: K-means algorithm implementation using k = 3 (plot)

Assign



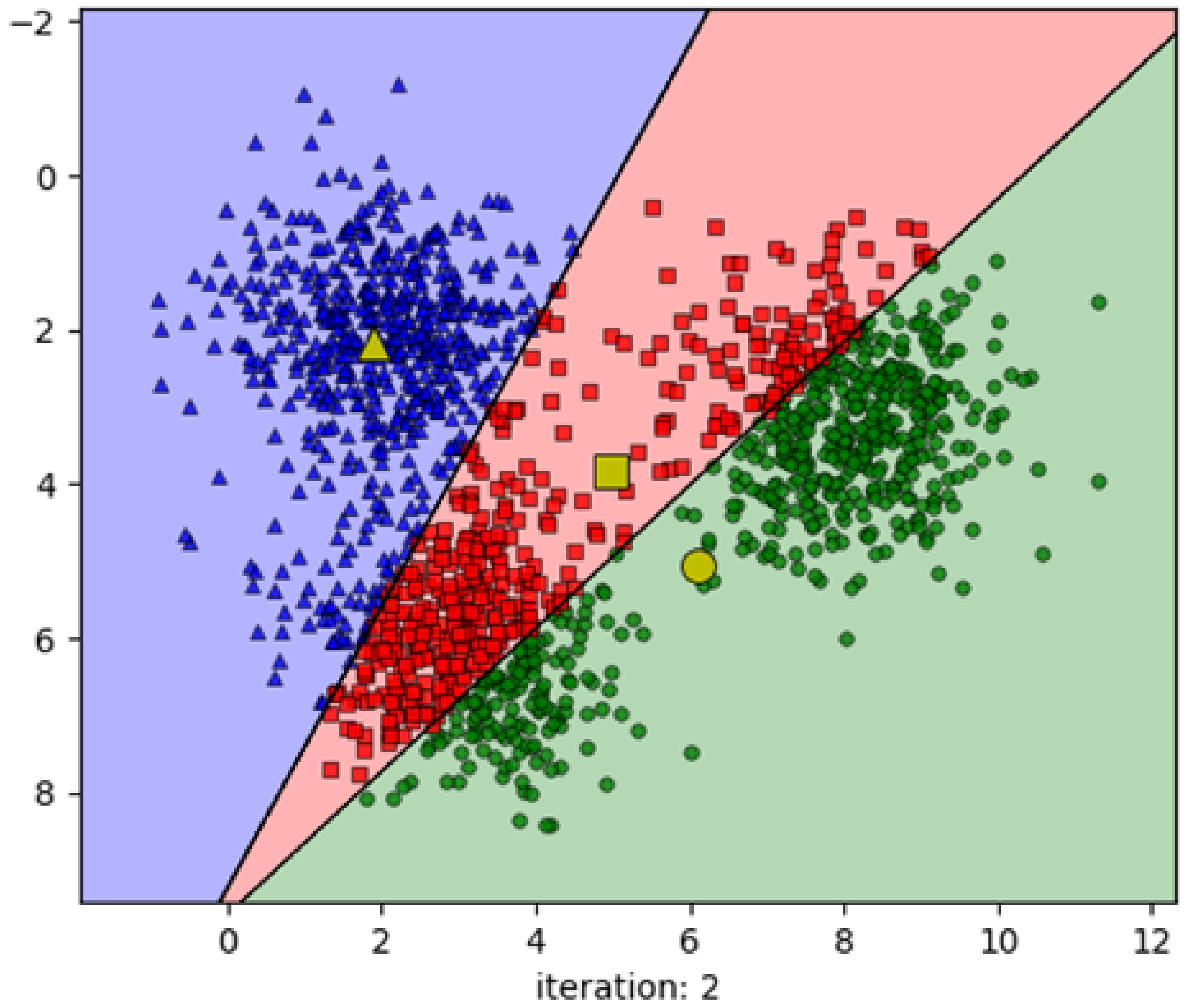
Update



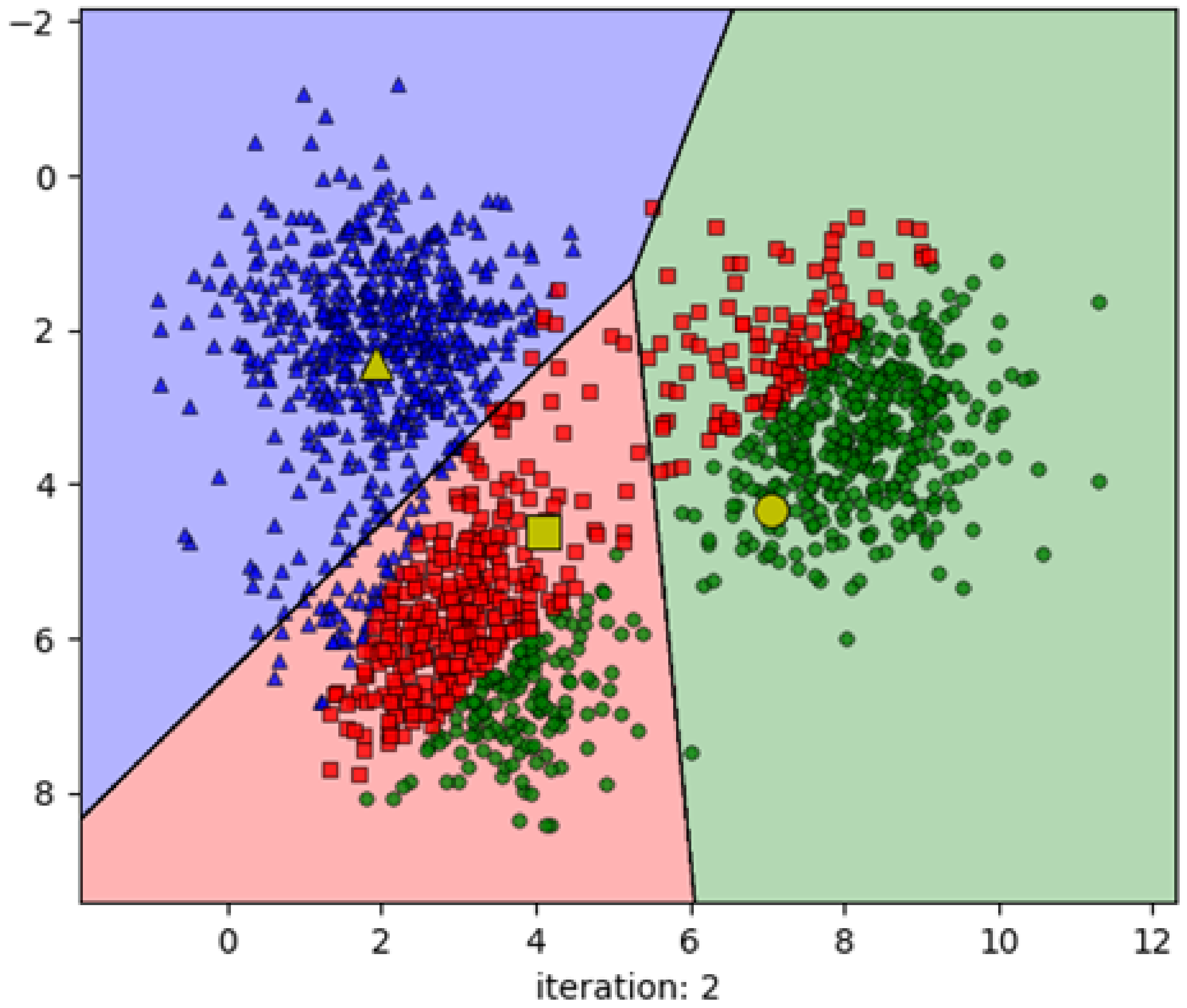
Example

Example 2: K-means algorithm implementation using k = 3 (plot)

Assign



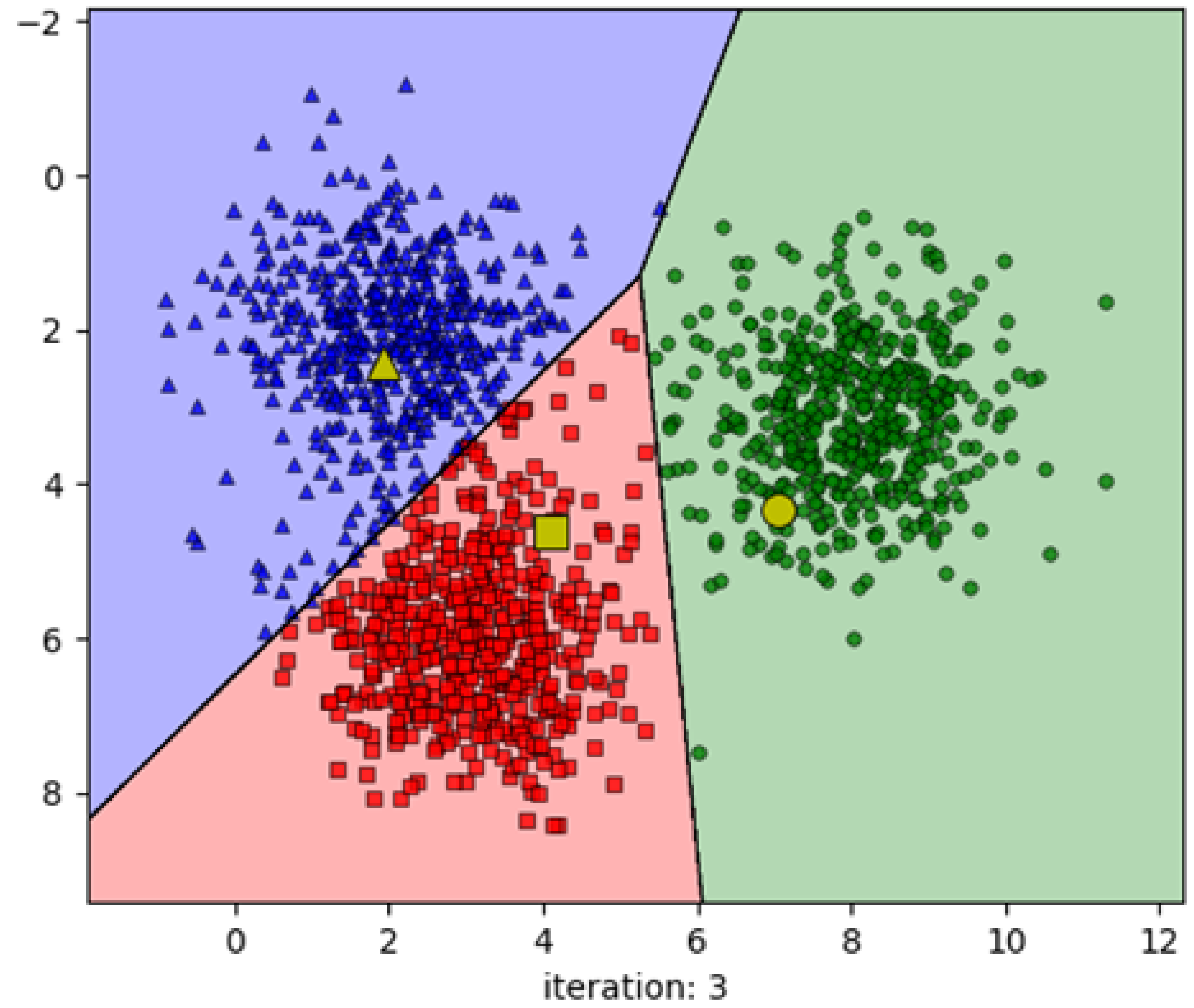
Update



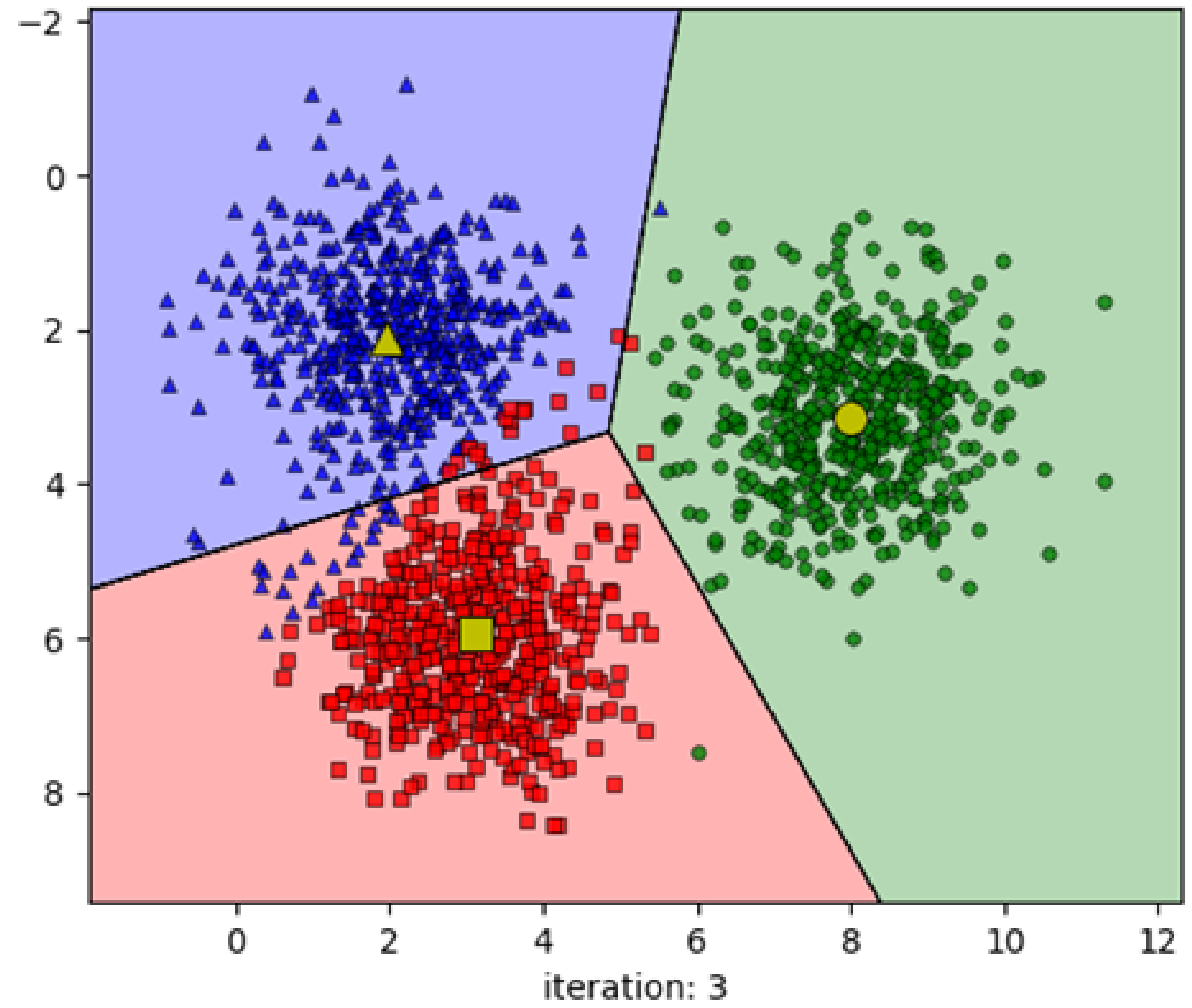
Example

Example 2: K-means algorithm implementation using k = 3 (plot)

Assign



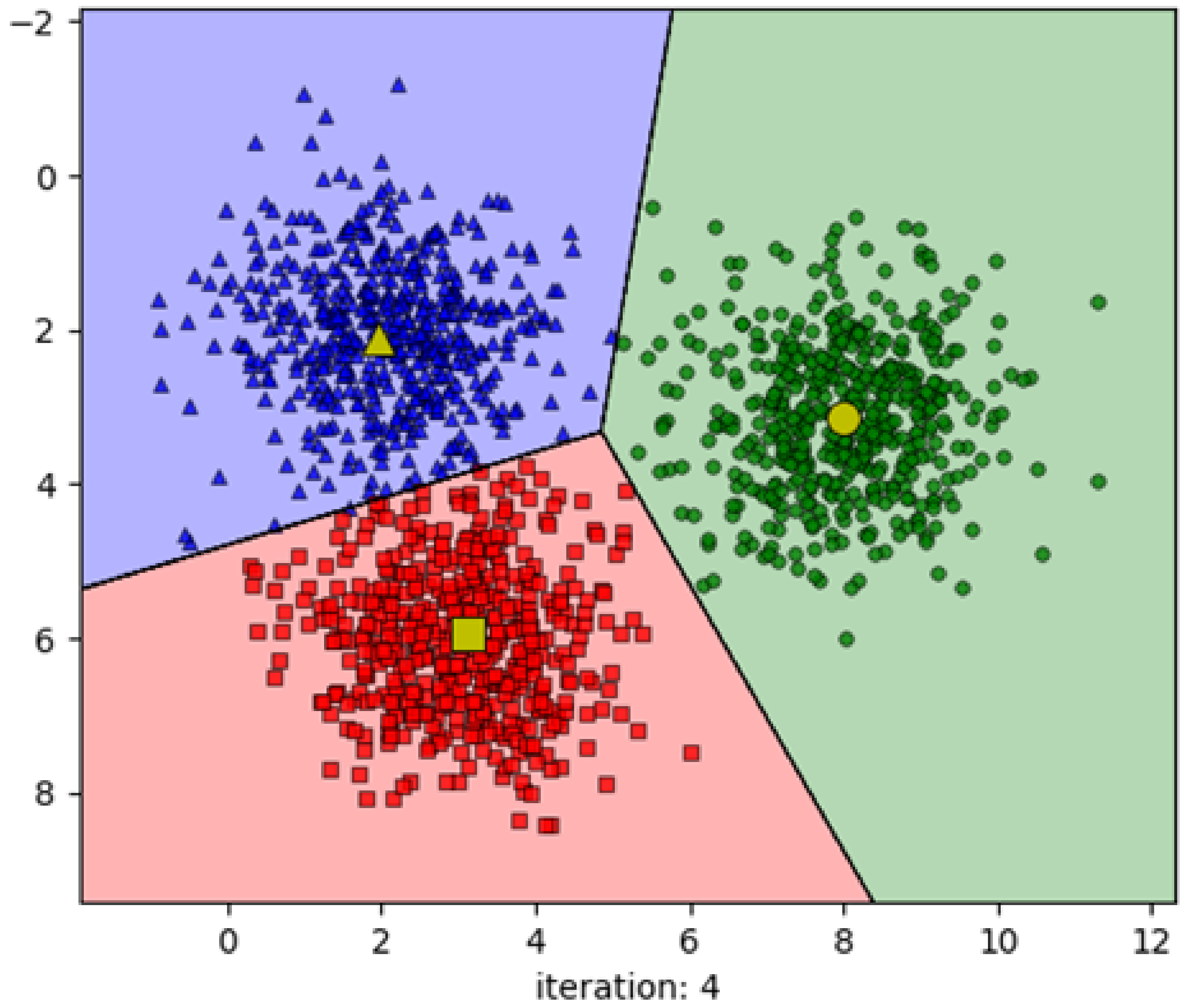
Update



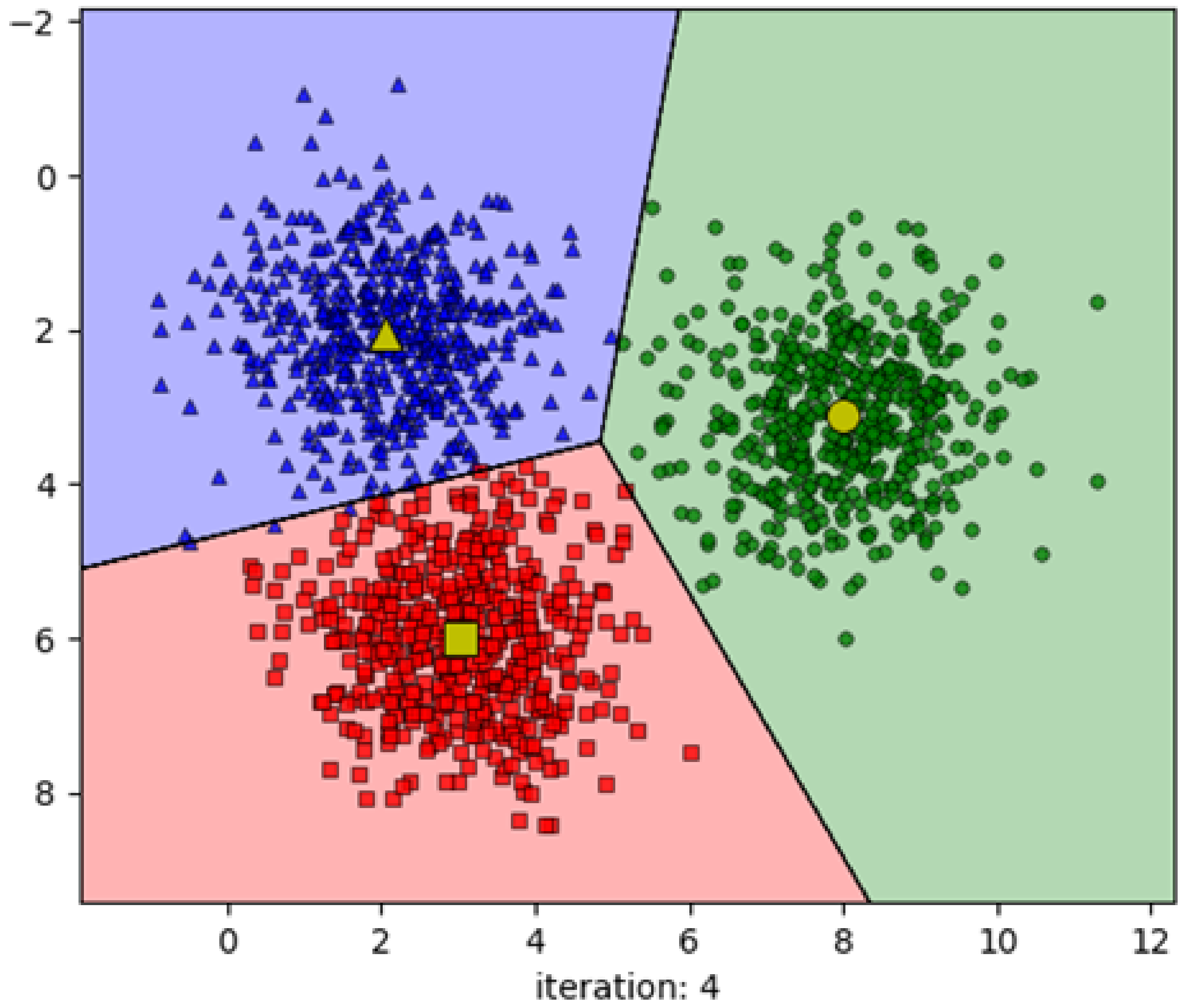
Example

Example 2: K-means algorithm implementation using k = 3 (plot)

Assign



Update

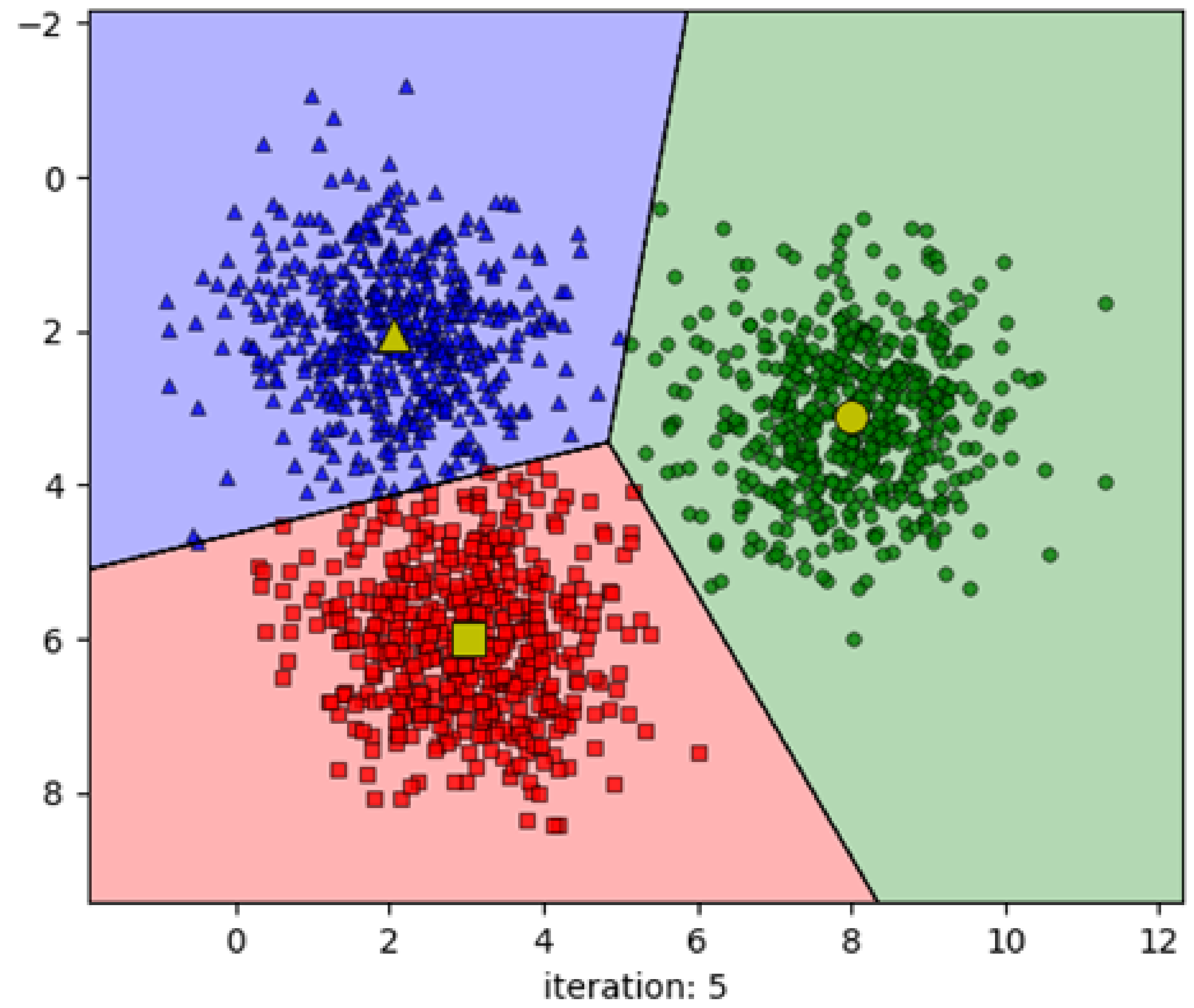


Example

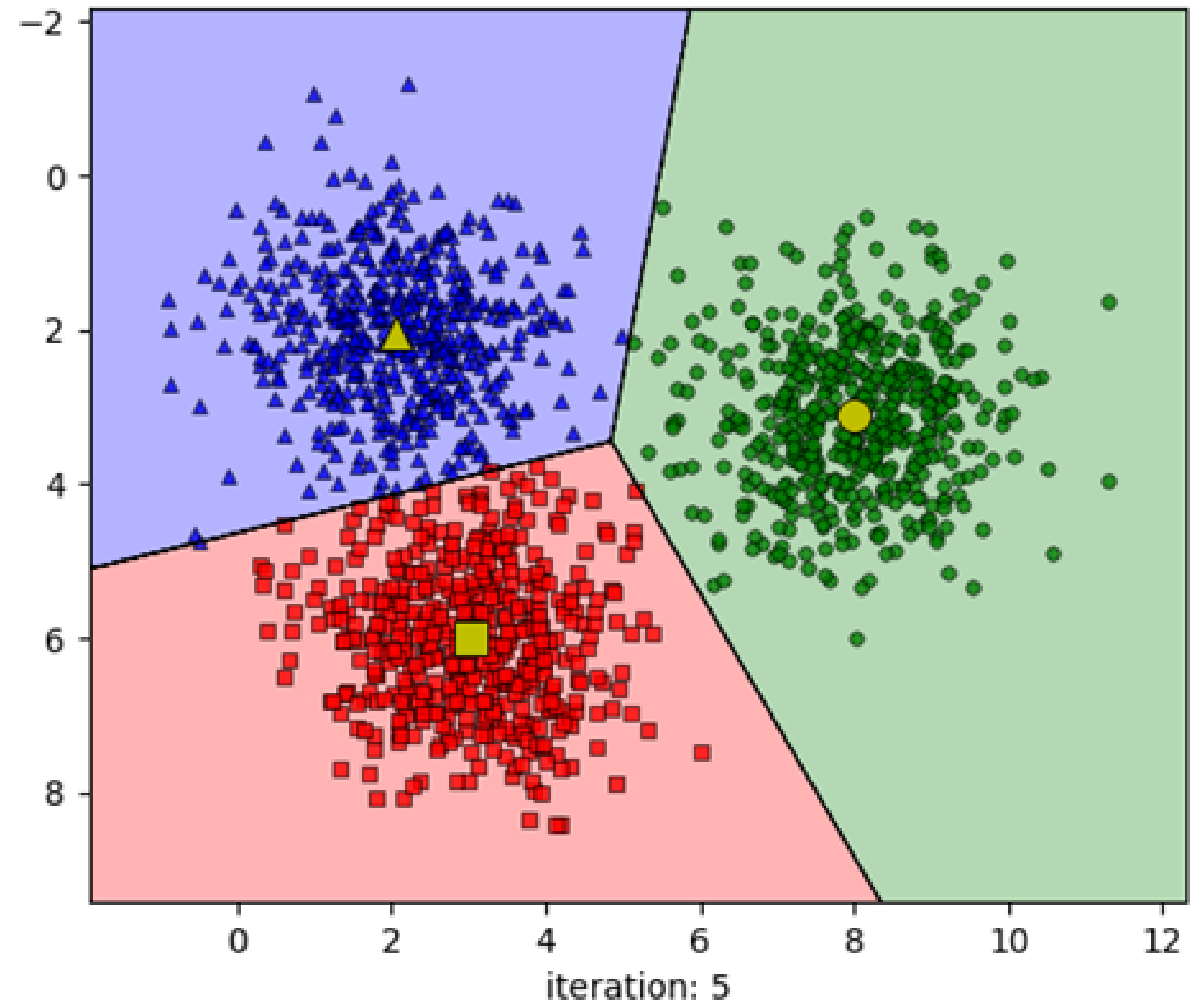
Example 2: K-means algorithm implementation using k = 3 (plot)

Terminate

Assign



Update



Comments on basic K-means

Strength

- Very easy to implement.
- Relatively efficient: O(kni), where n is # objects, k is # clusters, and i is # iterations. Normally, k, i<<n.

03.

Shortcoming of Lloyd k-means

Initialization

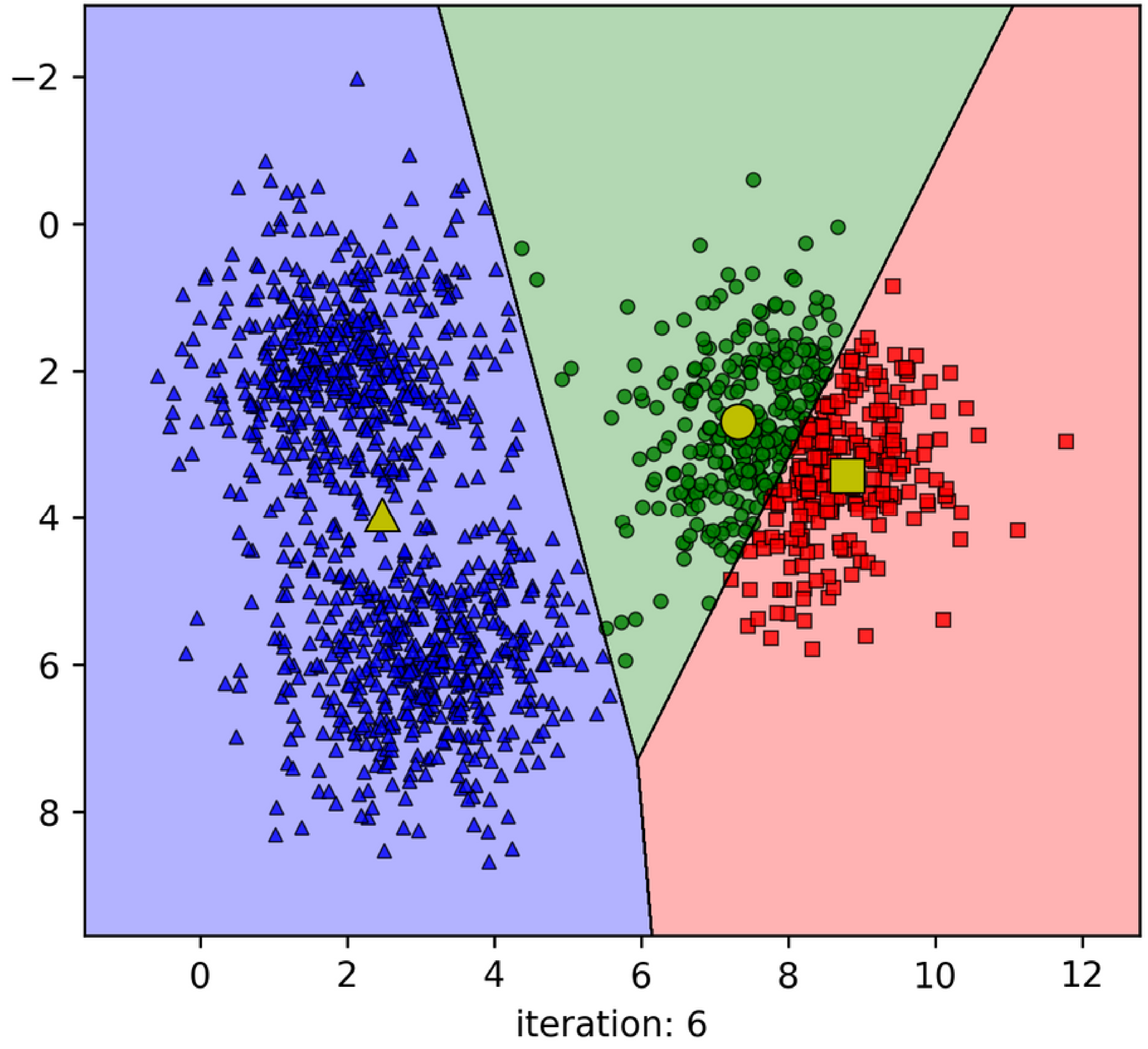
Minimizing **within-cluster-variation** is NP-hard

$$\phi = \sum_{i=1}^{k} \sum_{p \in C_i} \|p - c_i\|_2^2$$

-> Convergence of basic k-means depends on initialization

Basic k-means can perform arbitrarily bad

Initialization



Bad clusters are as likely as good ones

simple fix: try the algorithm several times and choose one with the smallest loss

K-means++

intuition: spreading out centroids produces better clusters

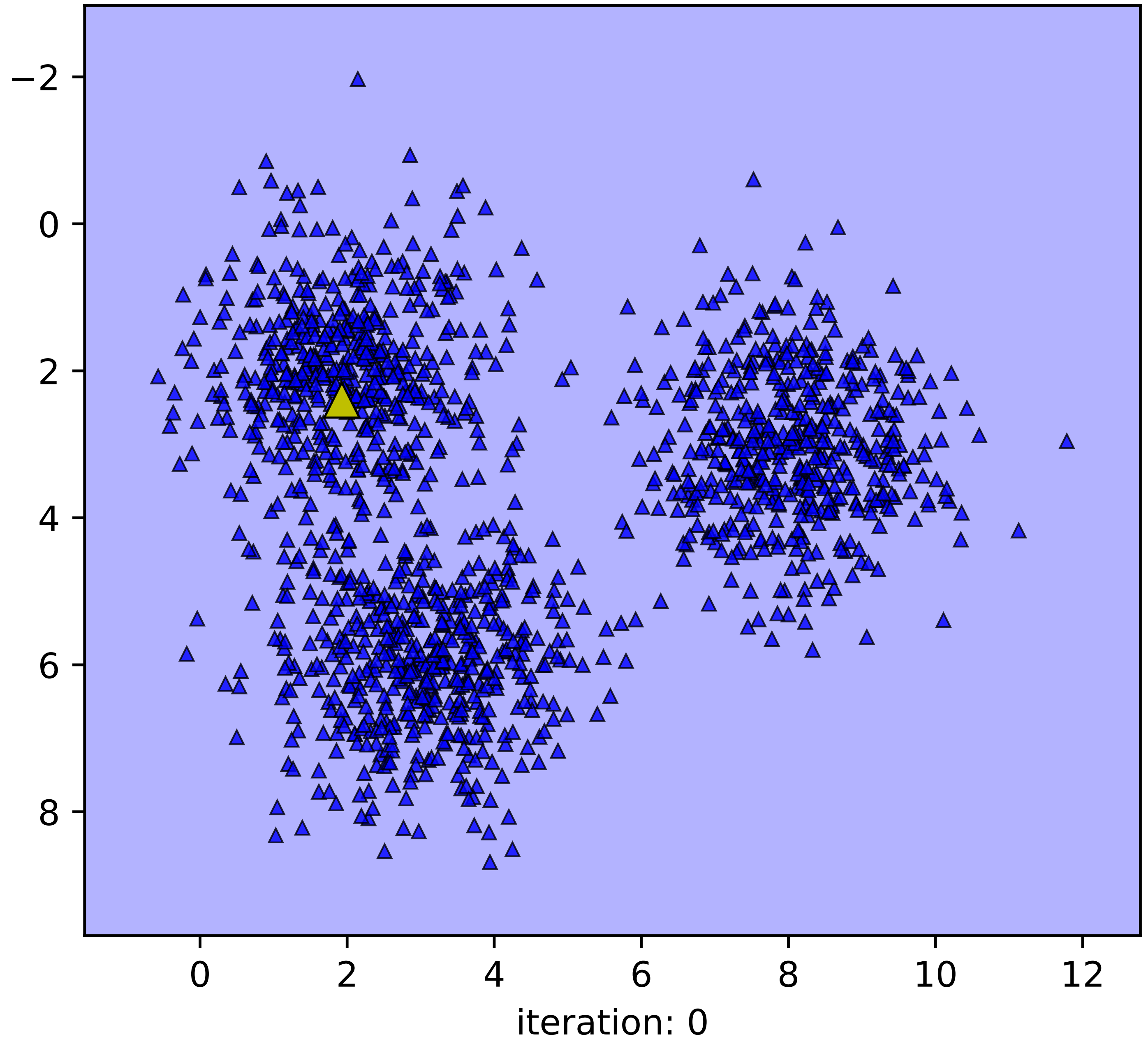
keep the iteration steps from normal k-means

K-means++

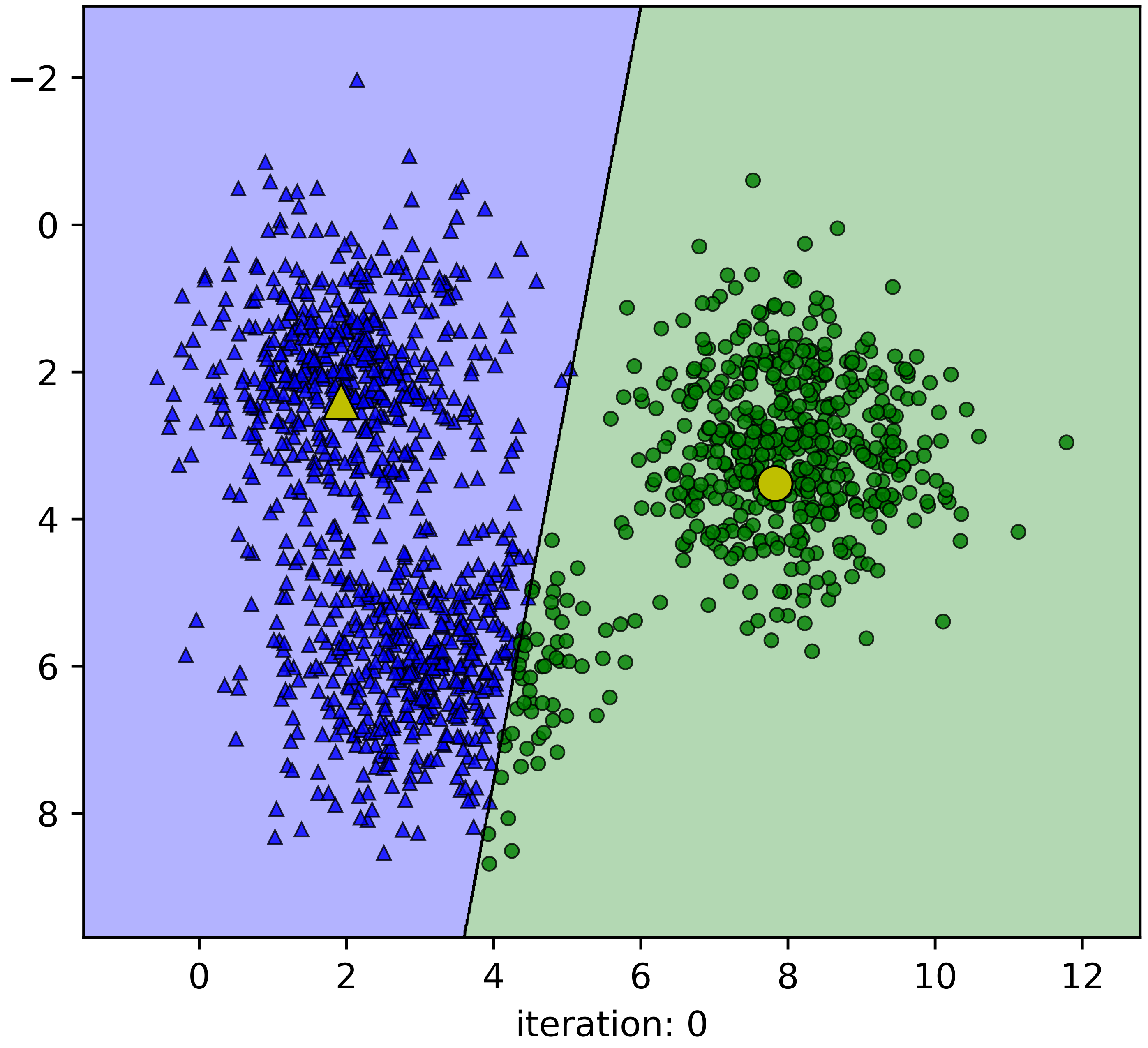
Initialization:

1. uniformly choose the first centroid
2. for each data point **x**, compute distance to its closest centroid $D(\mathbf{x})$
3. select each **x** with probability: $\frac{D(\mathbf{x})^2}{\sum_{\mathbf{y} \in X} D(\mathbf{y})^2}$
4. repeat steps 2, 3 for k – 1 times

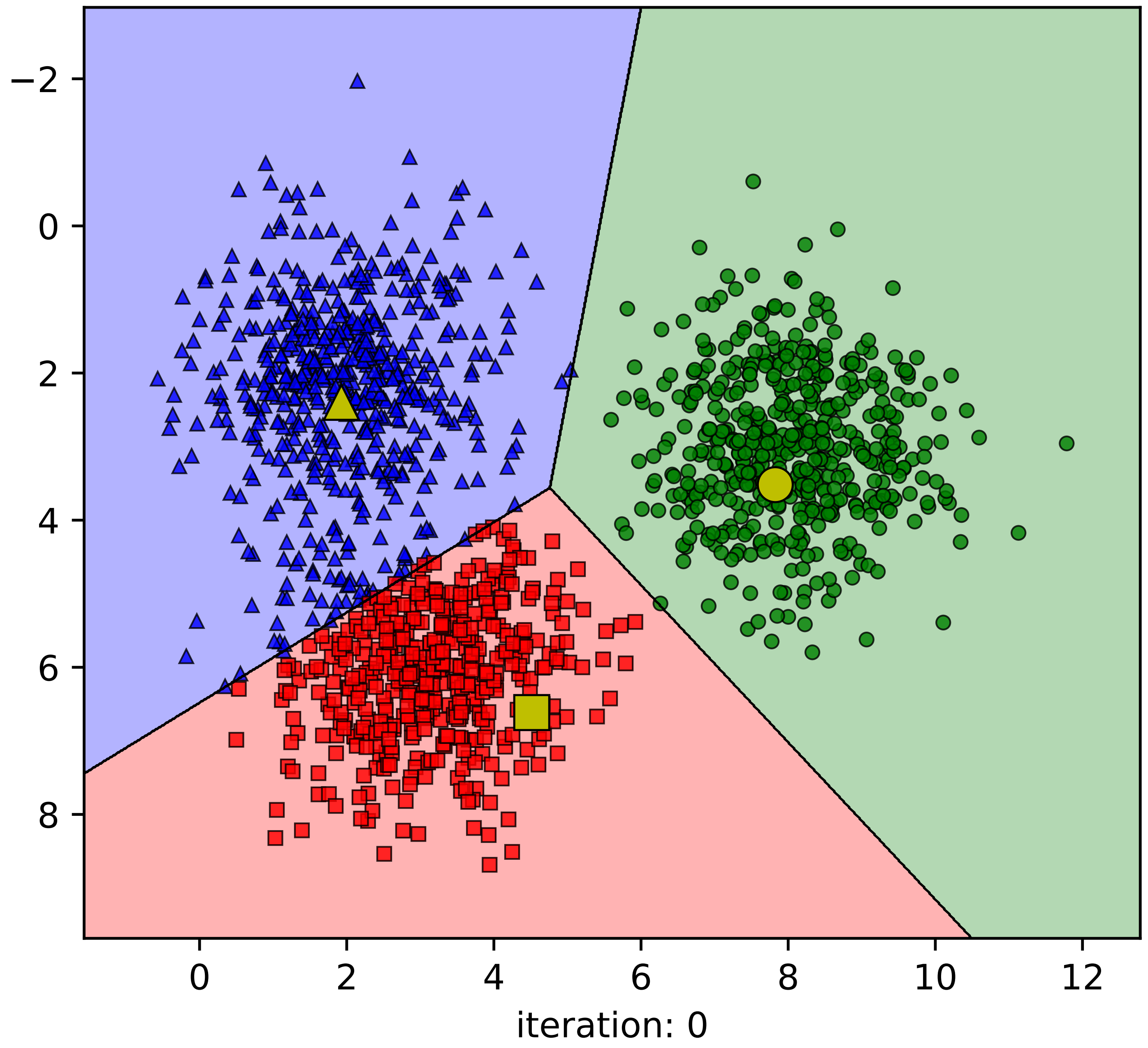
K-means++



K-means++



K-means++



K-means++

O(log k)-approximate. Specifically, let ϕ denote the loss value obtained by k-means++, then

$$E(\phi) \leq 8(\ln k + 2)\phi_{\text{OPT}}$$ [1]

Initialization time complexity: $O(k^2n)$

overall time complexity: $O(k^2n + kni)$

[1]: Arthur, D., & Vassilvitskii, S. (2006). k-means++: The advantages of careful seeding. Stanford.

K-means++

Greedy initialization: in every step, we sample ℓ candidate centers, then pick the one with smallest loss

Scikit-learn implements this

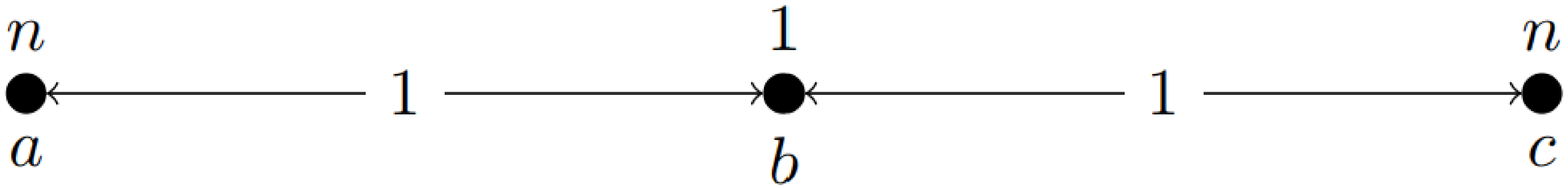
Fun fact: it has worse worst-case guarantee than k-means++

K-means++

$\ell = 1$: K-means++

$\ell = n \Longrightarrow$ deterministic, Maxmin initialization

$k = 2$:



$$\text{Step } 1 : \mathcal{C} \longleftarrow \{b\},\ \phi = 2n \qquad \phi_{\text{OPT}} = 2$$

$$\text{Step } 2 : \mathcal{C} \longleftarrow \mathcal{C} \cup \{a\},\ \phi = n \qquad \mathcal{C}_{\text{OPT}} = \{a, c\}$$

$\frac{\phi}{\phi_{\text{OPT}}} = \frac{n}{2} = \Theta(n)$: fails to obtain O(log k)-approximate

K-means++

Greedy initialization: $\Omega(\ell \log k)$-approximate in expectation for certain dataset [2]

Sklearn implementation, pick $\ell = \Theta(\log k)$

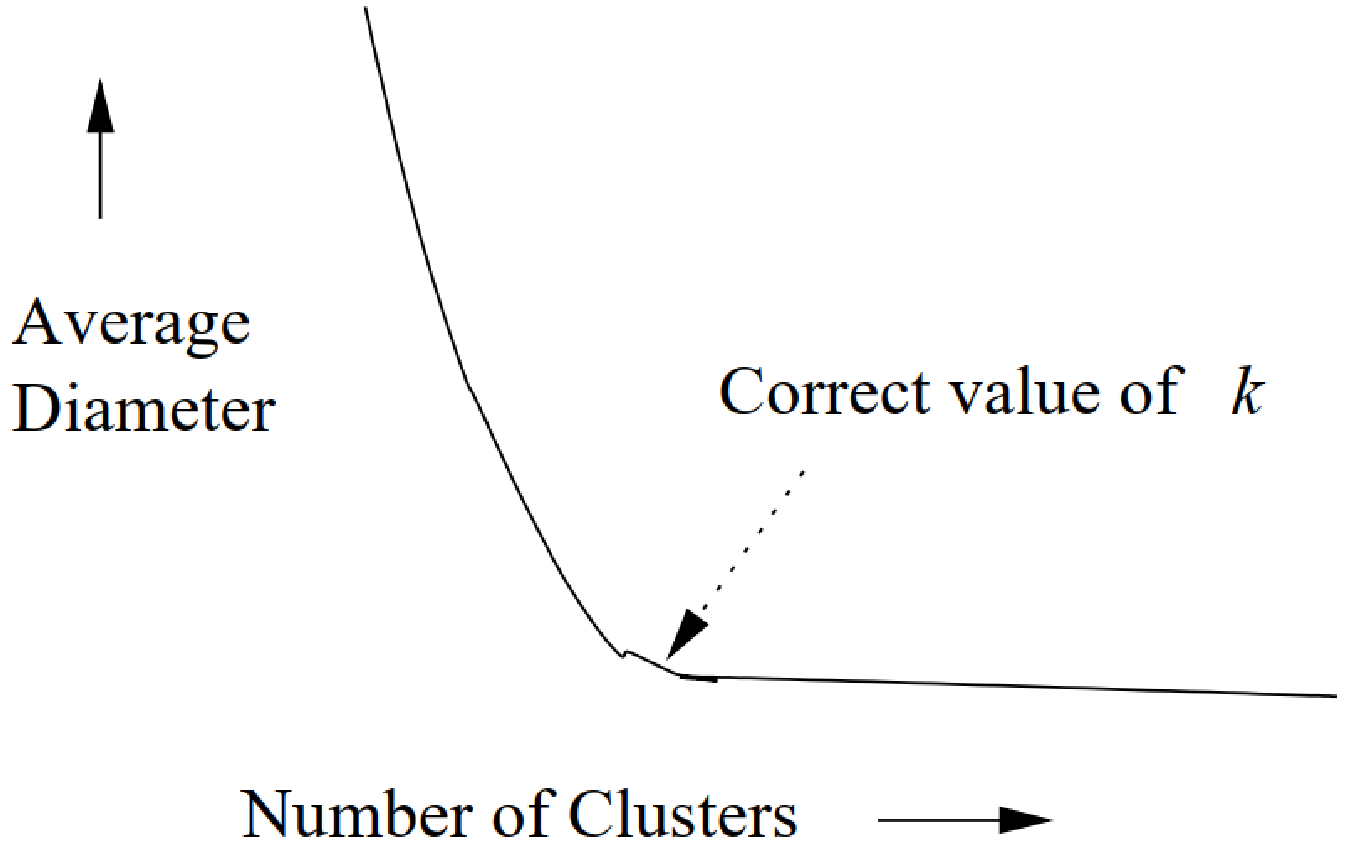
-> even worse

[2]: Bhattacharya, A., Eube, J., Röglin, H., & Schmidt, M. (2019). Noisy, greedy and not so greedy k-means++. arXiv preprint arXiv:1912.00653.

Getting k right

Try different k, observe change in the average cluster radius or diameter as k increases

Intuition: average falls rapidly until the right k, then changes little



Getting k right

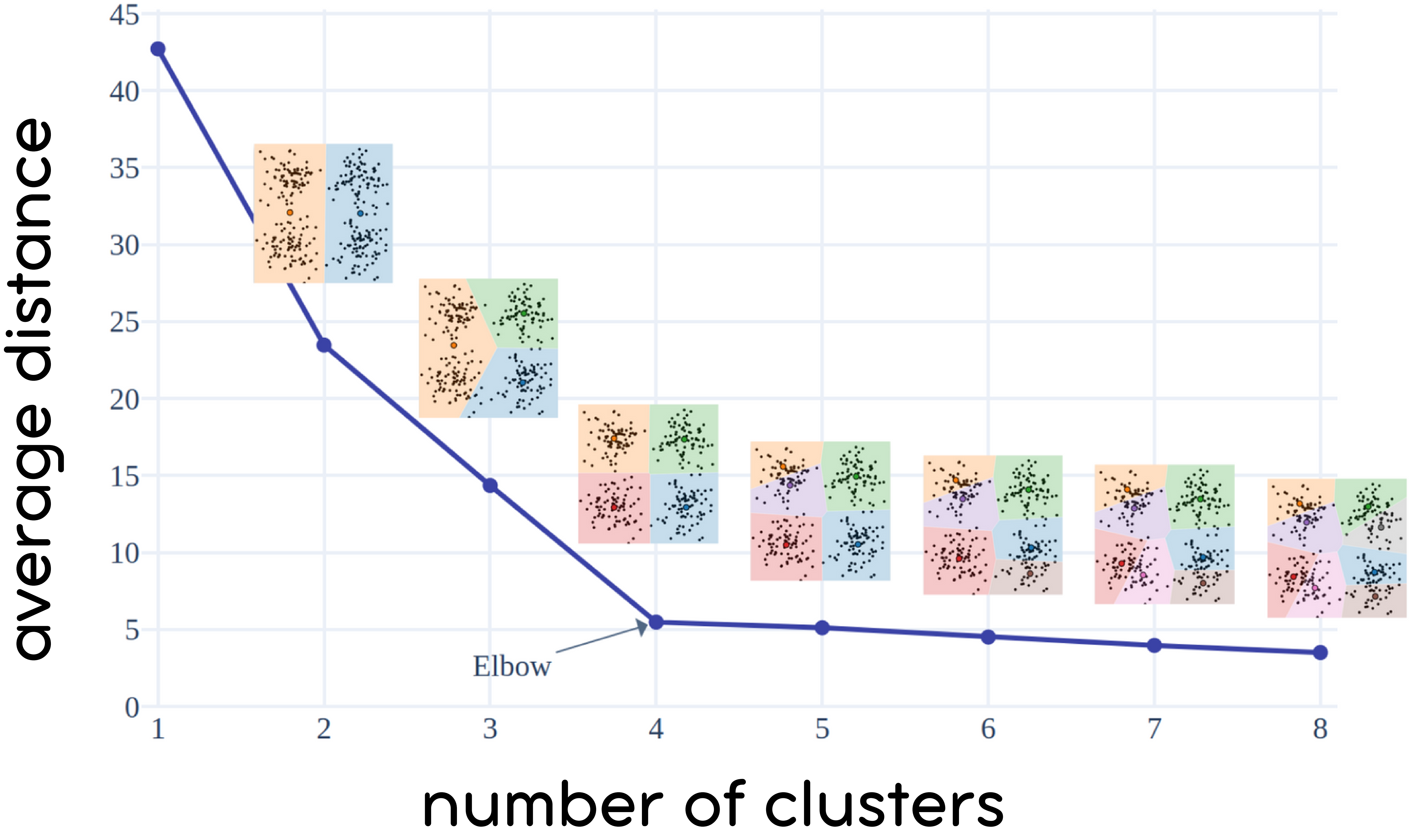
Algorithm:

1. run k-means with k = 1,2,4,8,... Stop at the first v that has small* decrease
2. do binary search for best k in [v/2, v]

Achieve O(in k logk)

*: How small -> set a threshold

Getting k right



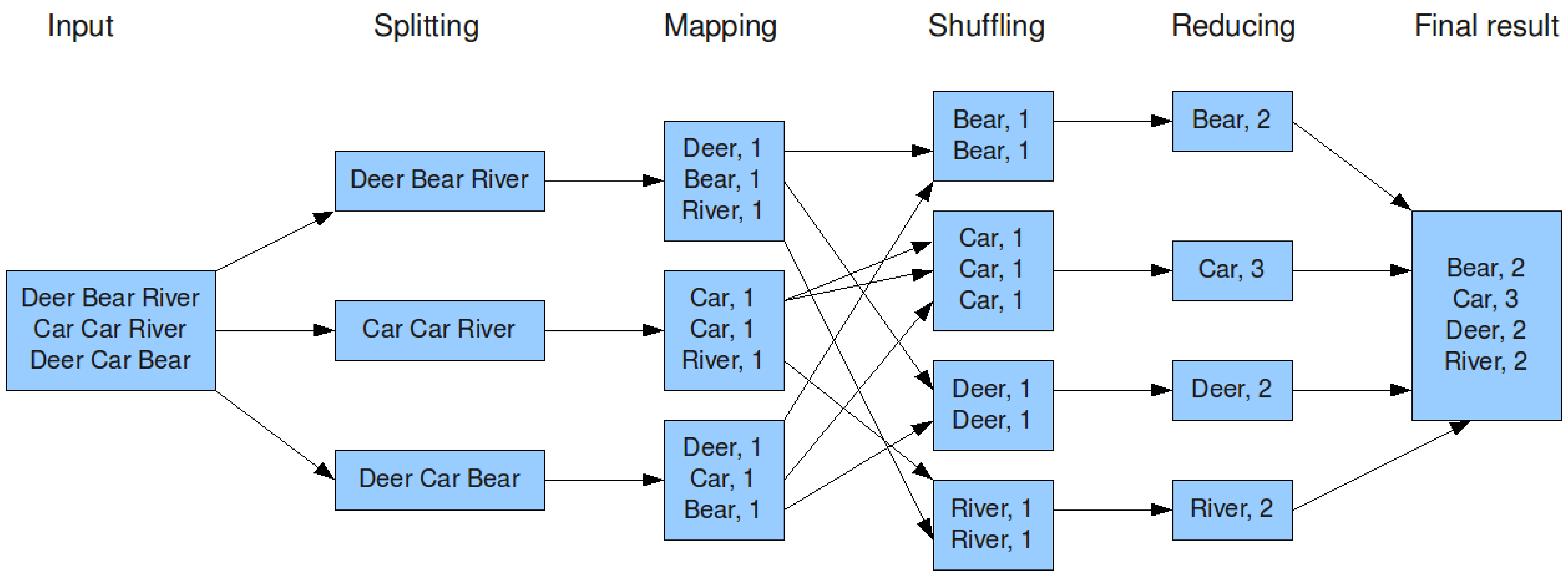
Scalability

Parallelize k-means

Use MapReduce paradigm

Scalability

MapReduce Overview



Scalability

assign:

$$\texttt{kmeansMap}(x) :$$
$$\qquad \texttt{return}\ (\mathrm{argmin}_i \|x - c_i\|^2, (x, 1))$$

update:

$$\texttt{kmeansReduce}(i, [(x, s), (y, t)]) :$$
$$\qquad \texttt{return}\ (i, (x + y, s + t))$$

$$\Big(i, \Big(\sum_{x \in C_i} x, |C_i|\Big)\Big)$$

$$c_i \longleftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Scalability

k-means++ initialization can't be parallelized well

Bahmani et al. (2012) proposes k-means||

	Random	K-means++	K-means||
sample per iteration	k	1	$\ell = \mathcal{O}(k)$
number of iteration	1	k	$t = \mathcal{O}(1)$
worst-case bound	inf	$\mathcal{O}(\log k \cdot \phi_{\mathrm{OPT}})$	$\mathcal{O}(\epsilon + \log k \cdot \phi_{\mathrm{OPT}})$ [3]

K-means||

Initialization:

1. uniformly choose the first centroid
2. for each data point **x**, compute distance to its closest centroid $D(\mathbf{x})$
3. select each **x** with probability: $\frac{\ell D(\mathbf{x})^2}{\sum_{\mathbf{y}\in X} D(\mathbf{y})^2}$
4. repeat steps 2, 3 for t times
5. recluster $E(\ell t + 1)$ points to k centroids using k-means++

K-means||

Time complexity: $\mathcal{O}(n\ell t)$

Spark.mllib suggests $\ell = 2k,\ t = 5 \to 8$

Step 2: $\texttt{updateCost}(x):$

$$\texttt{return}\ (x, \min_{c\in\mathcal{C}} \|x - c\|^2)$$

Bottle neck, may use multiprocessing here

Step 3: $\texttt{sample}(x, D_x):$

$$\texttt{with probability}\ \frac{\ell D_x}{\phi}:$$

$$\texttt{return}\ x$$

$$\texttt{else}:$$

$$\texttt{return None}$$

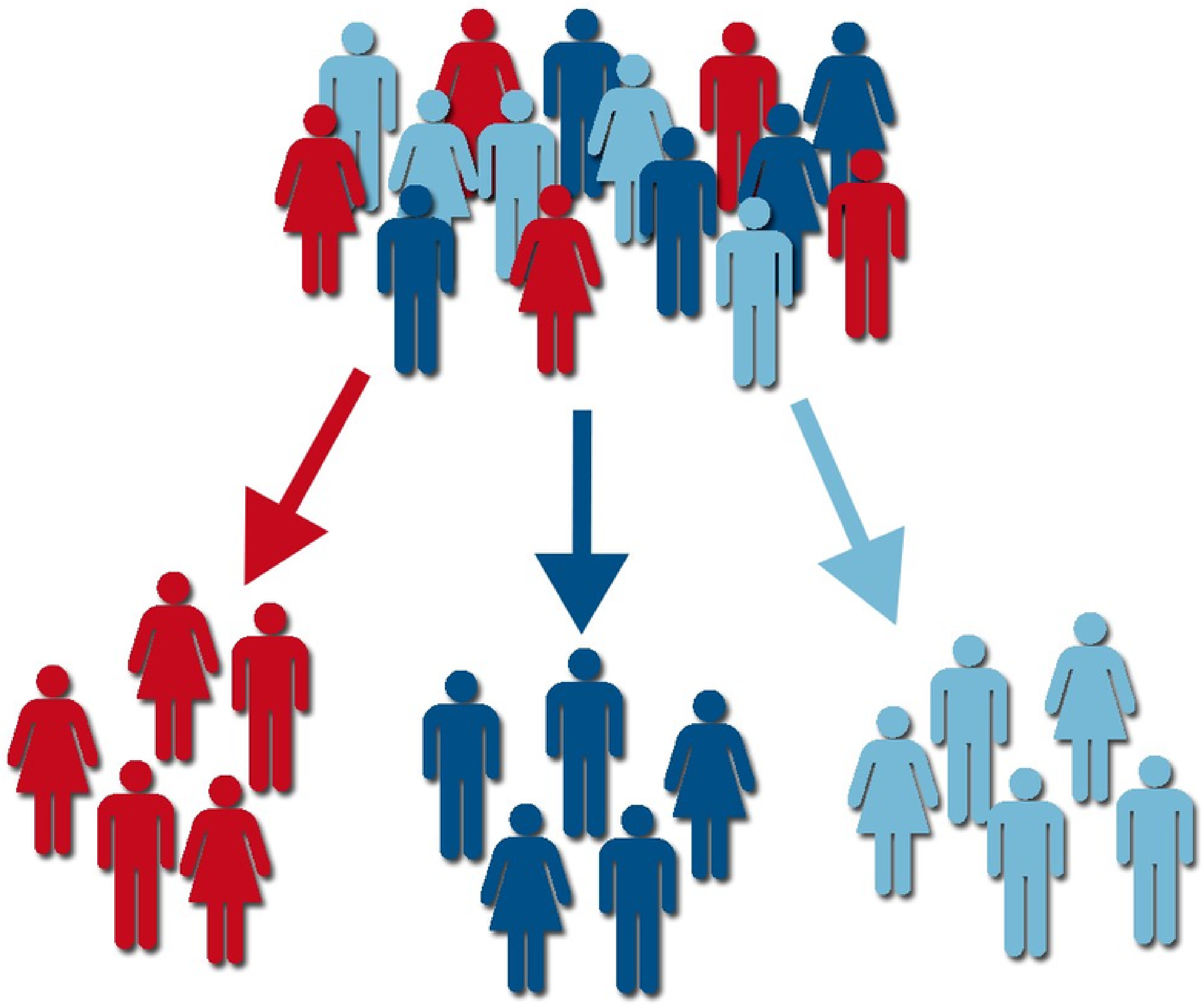
04

Application & Summary

Application

1. Customer Segmentation

- helps marketers segment customers based on purchase history, interests, or activity monitoring.



Application

2. Document classification

3. Delivery store optimization

4. Call record detail analysis

5. Image segmentation

Summary

Clustering: Given a set of points, with a notion of distance between points, group the points into some number of ***clusters***

Algorithms:

K-means

Initialization, picking k, scalability