

Data Mining Techniques

CS 6220 - Section 3 - Fall 2016

Lecture 9

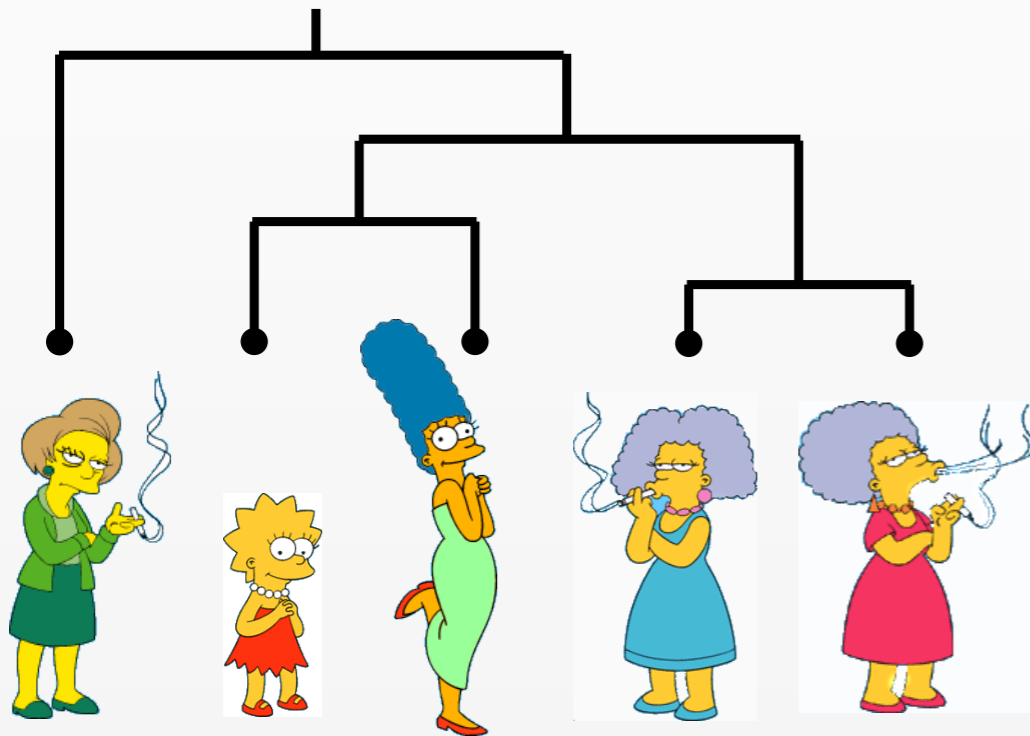
Jan-Willem van de Meent

(*credit*: Yijun Zhao, Carla Brodley, Eamon Keogh, Tan + Steinbach + Kumar)



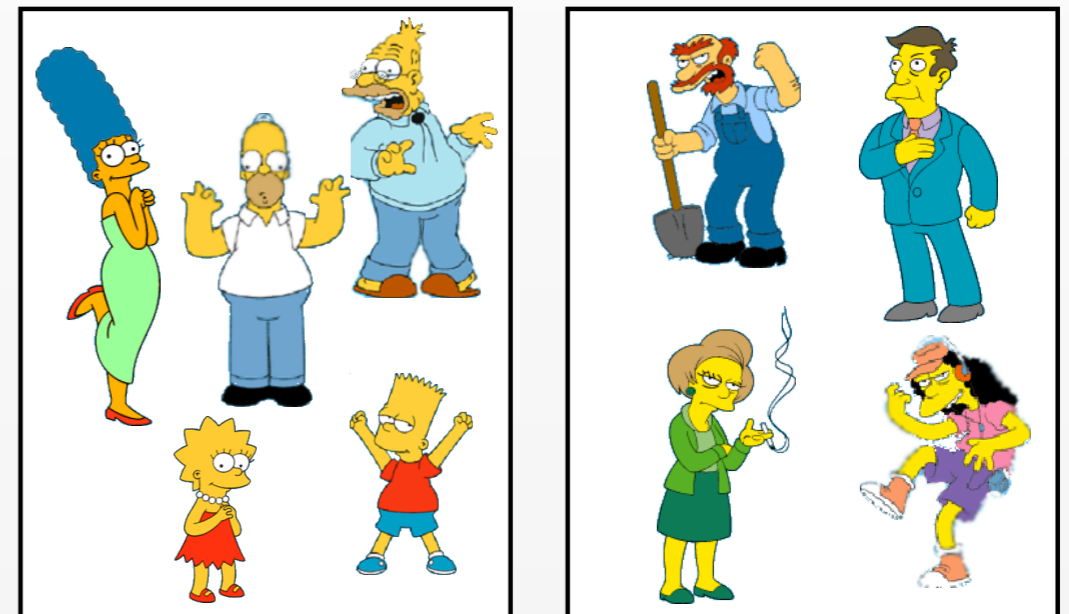
Two Types of Clustering

Hierarchical



Create a hierarchical decomposition using “*some criterion*”

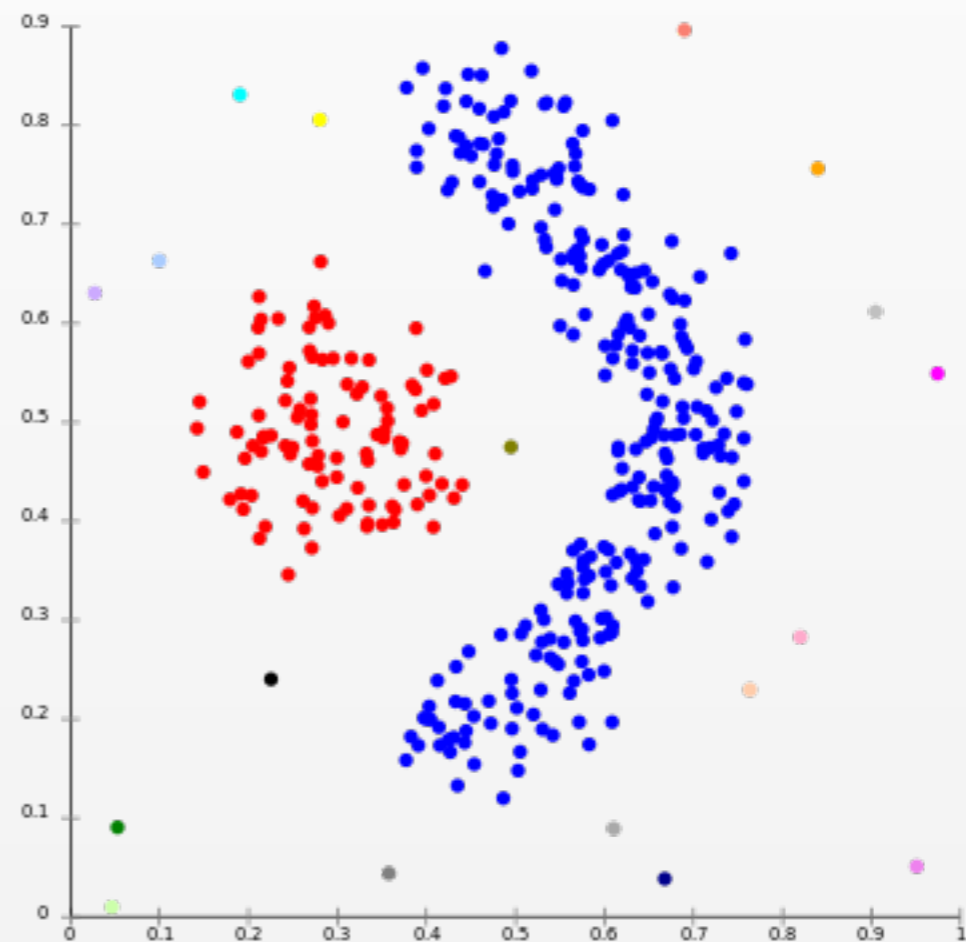
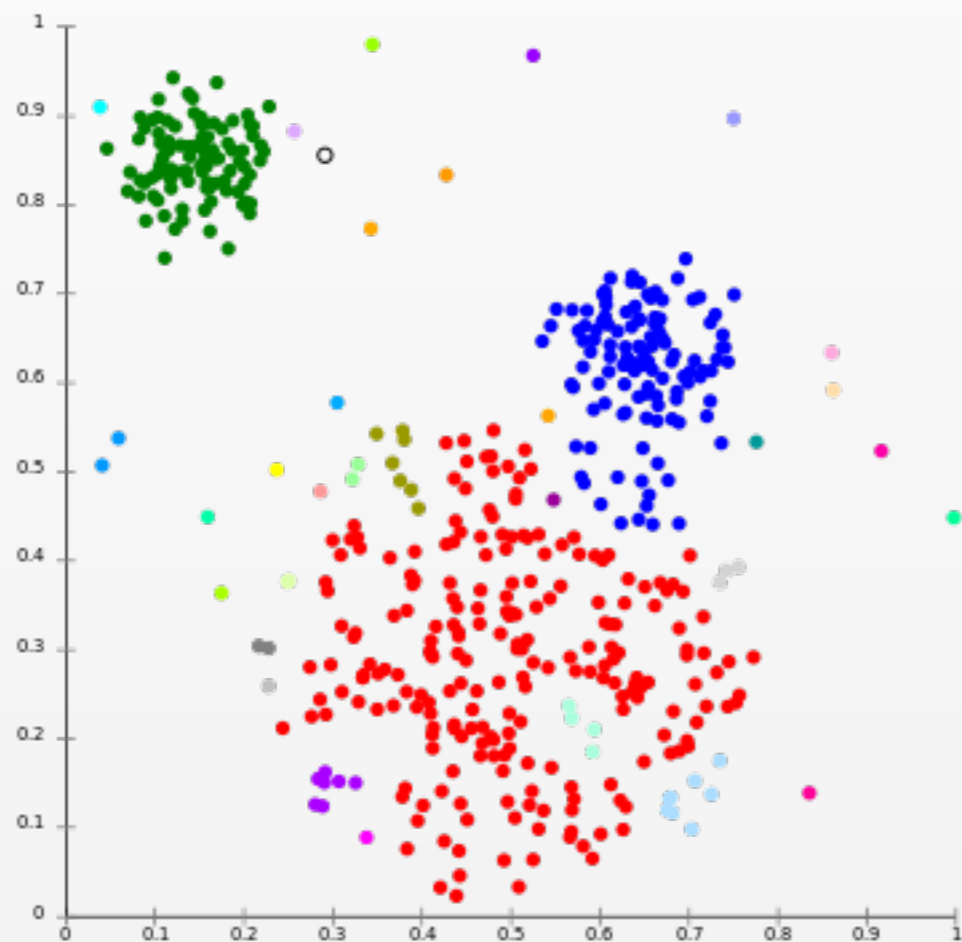
Partitional



Construct partitions and evaluate them using “*some criterion*”

~~Two~~ Four Types of Clustering

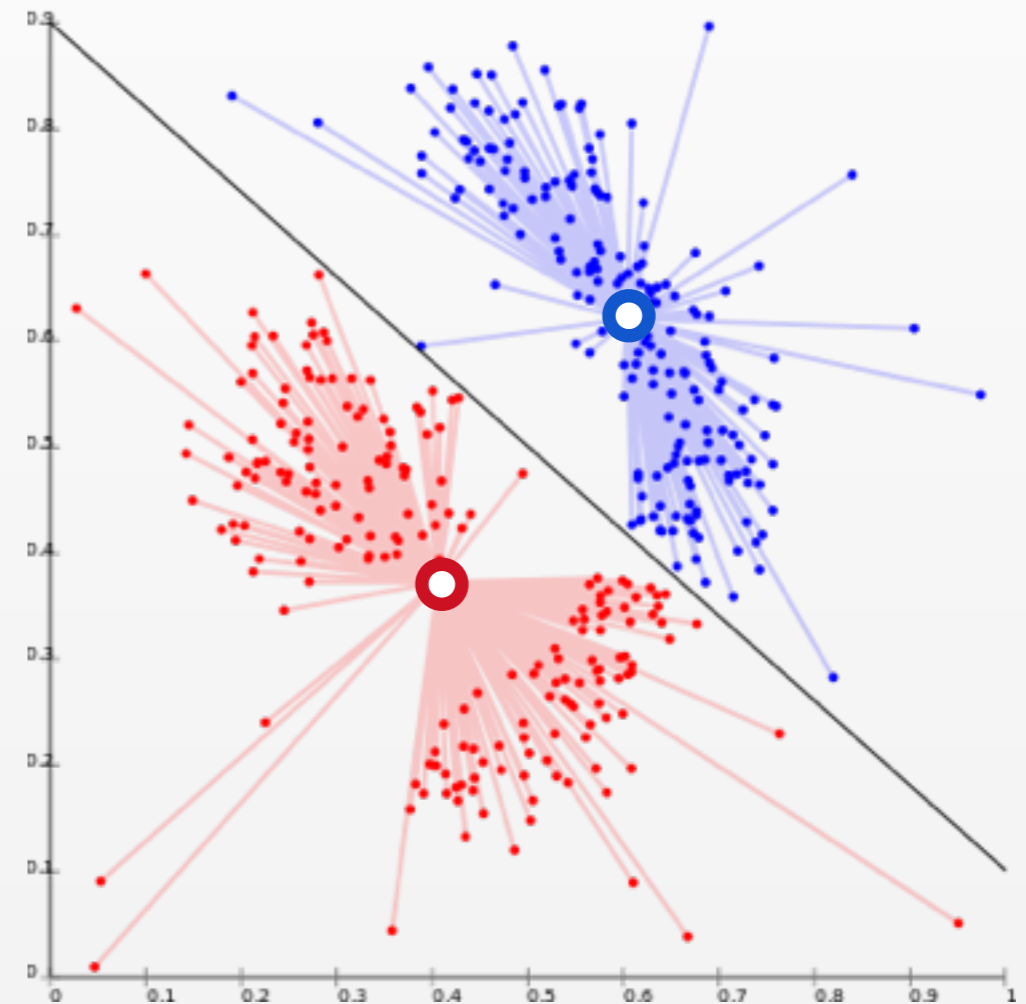
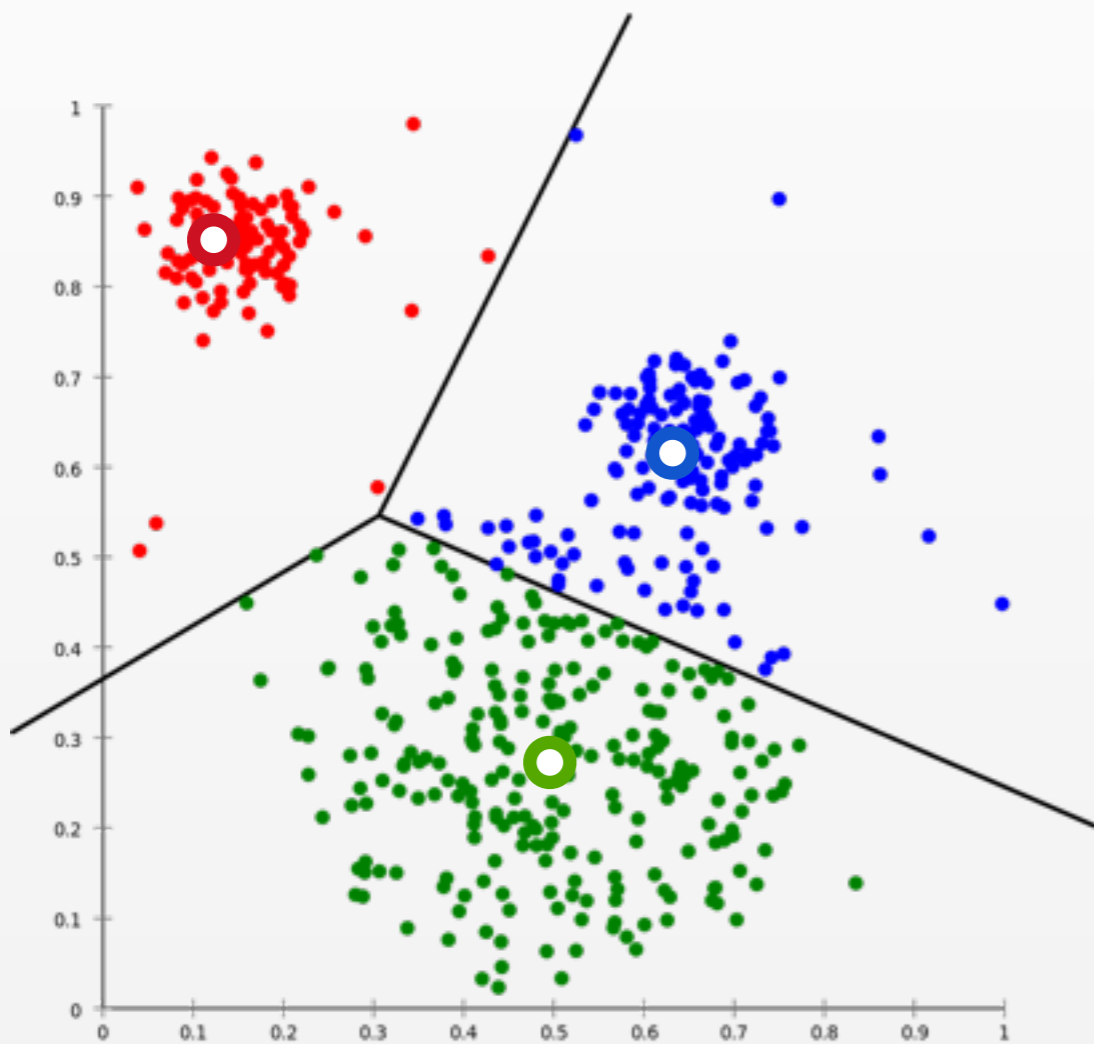
1. *Connectivity-based (Hierarchical)*



Notion of Clusters: Cut off dendrogram at some depth

~~Two~~ Four Types of Clustering

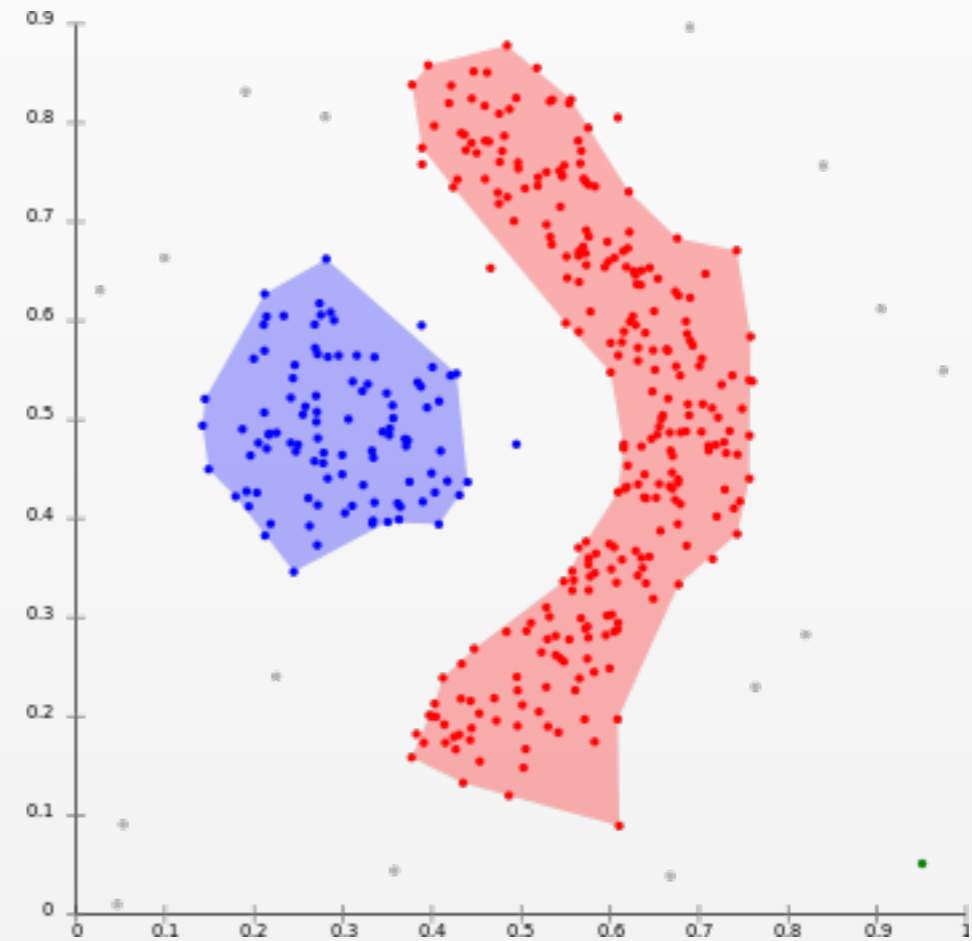
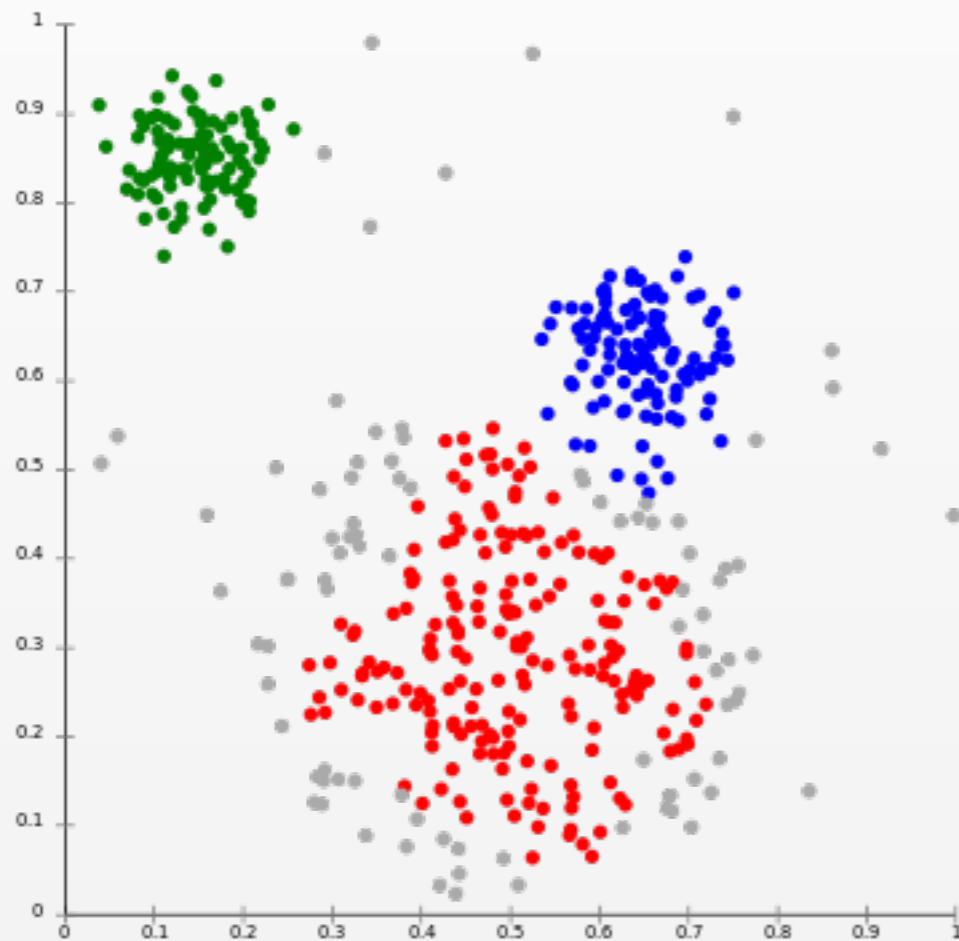
2. Centroid-based (*K*-means, *K*-medoids)



Notion of Clusters: Voronoi tessellation

~~Two~~ Four Types of Clustering

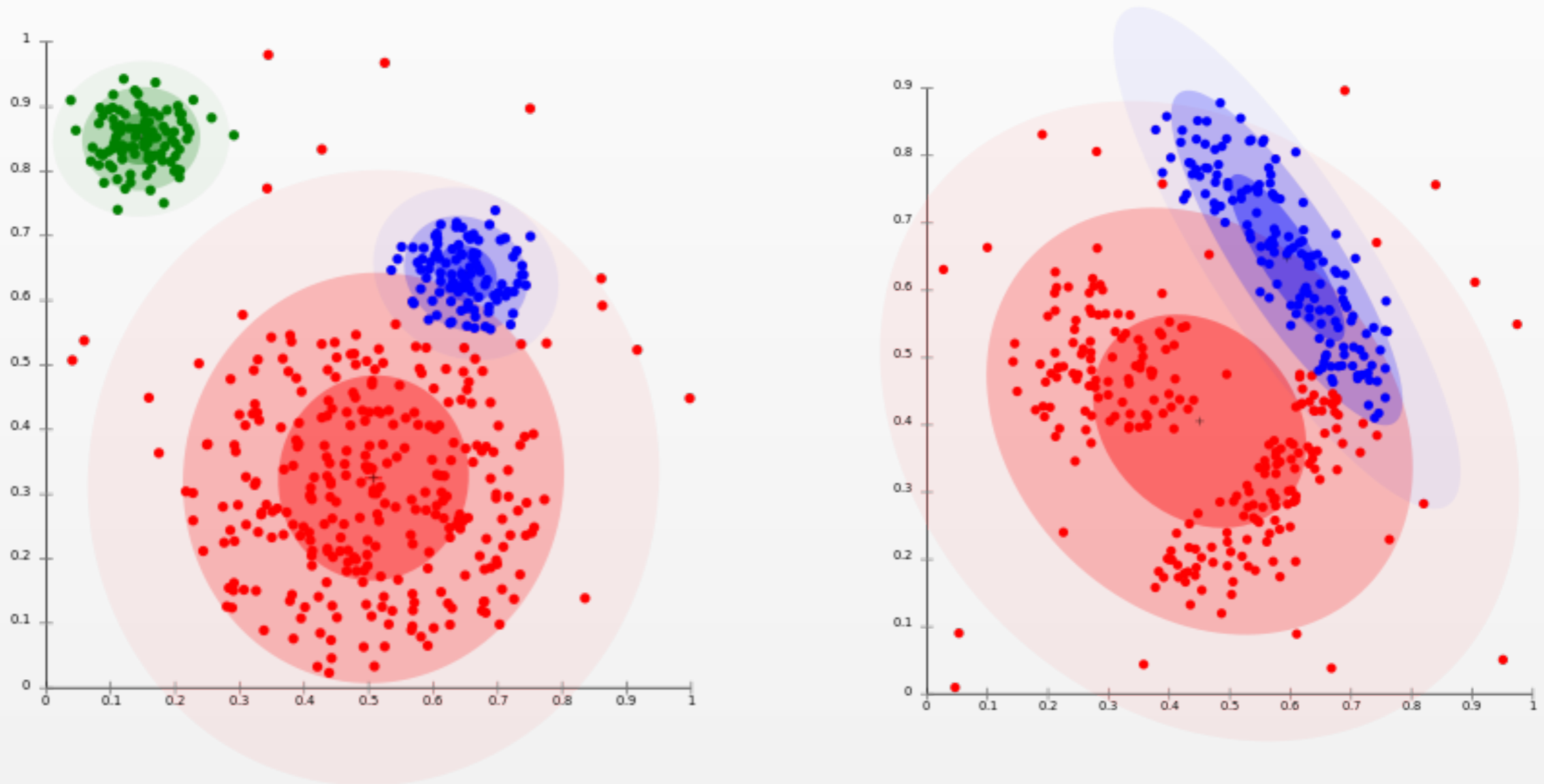
3. *Density-based (DBSCAN, OPTICS)*



Notion of Clusters: Connected regions of high density

~~Two~~ Four Types of Clustering

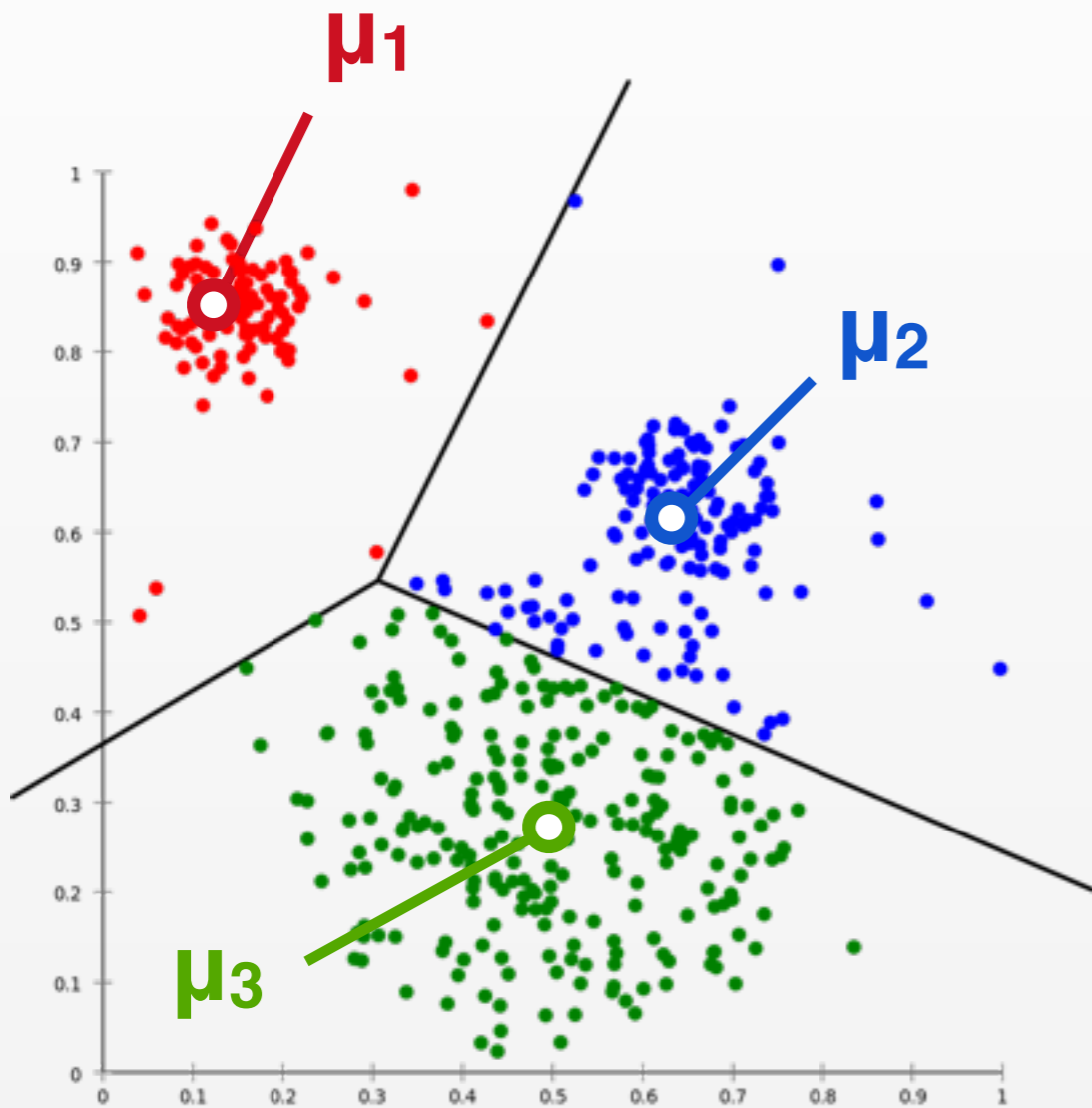
4. *Distribution-based (Mixture Models)*



Notion of Clusters: Distributions on features

K-Means Clustering

K-means Clustering



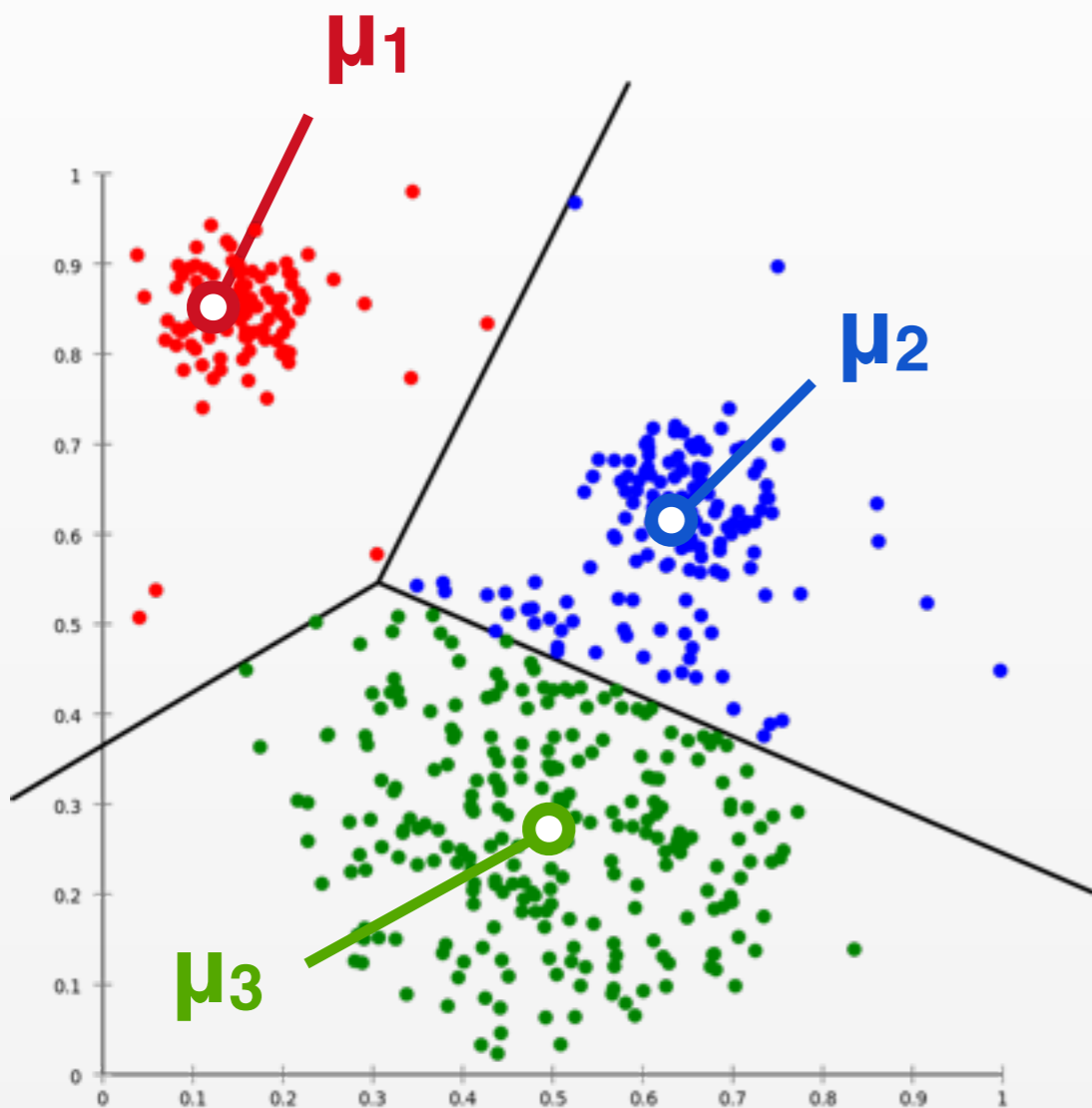
Idea: Minimize Sum of Squares

$$SSE_i = \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu_i\|^2$$

$$SSE = \sum_{j=1}^K SSE_j$$

Brute force search: NP-hard
How many partitions?

K-means Clustering



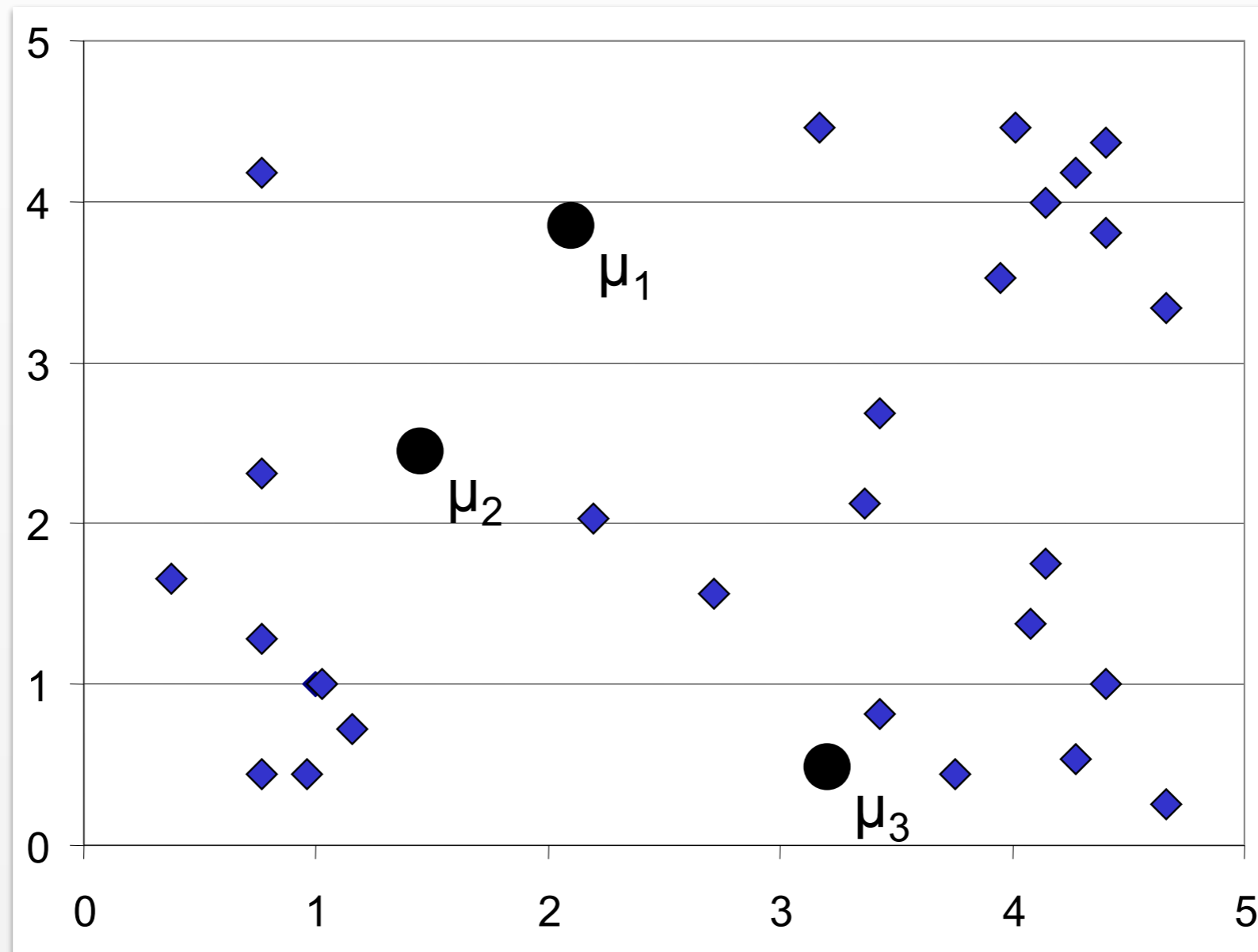
Idea: Minimize Sum of Squares

$$SSE_i = \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu_i\|^2$$

$$SSE = \sum_{j=1}^K SSE_j$$

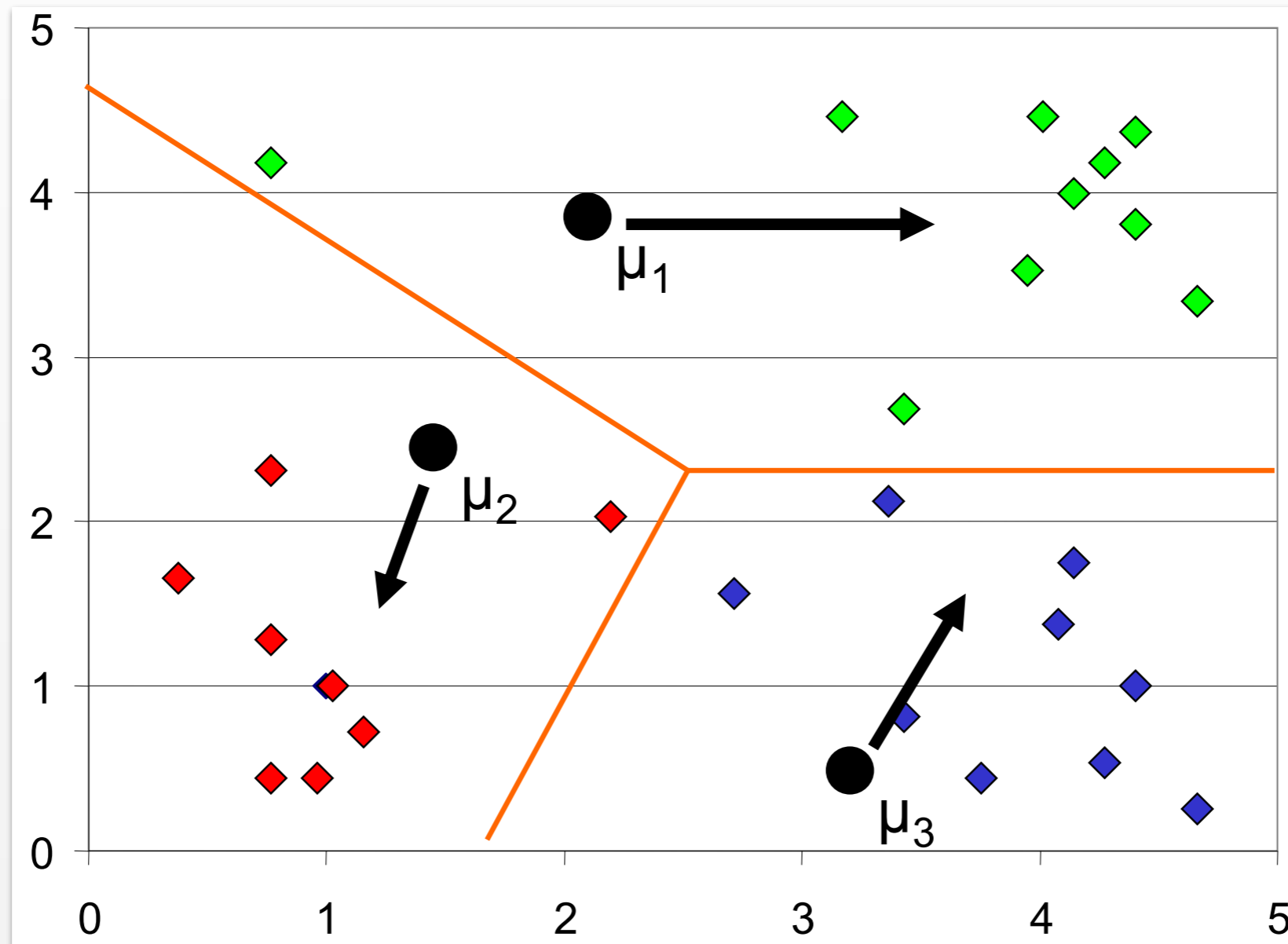
Use *heuristic* search
(as in hierarchical case)

K-means Clustering



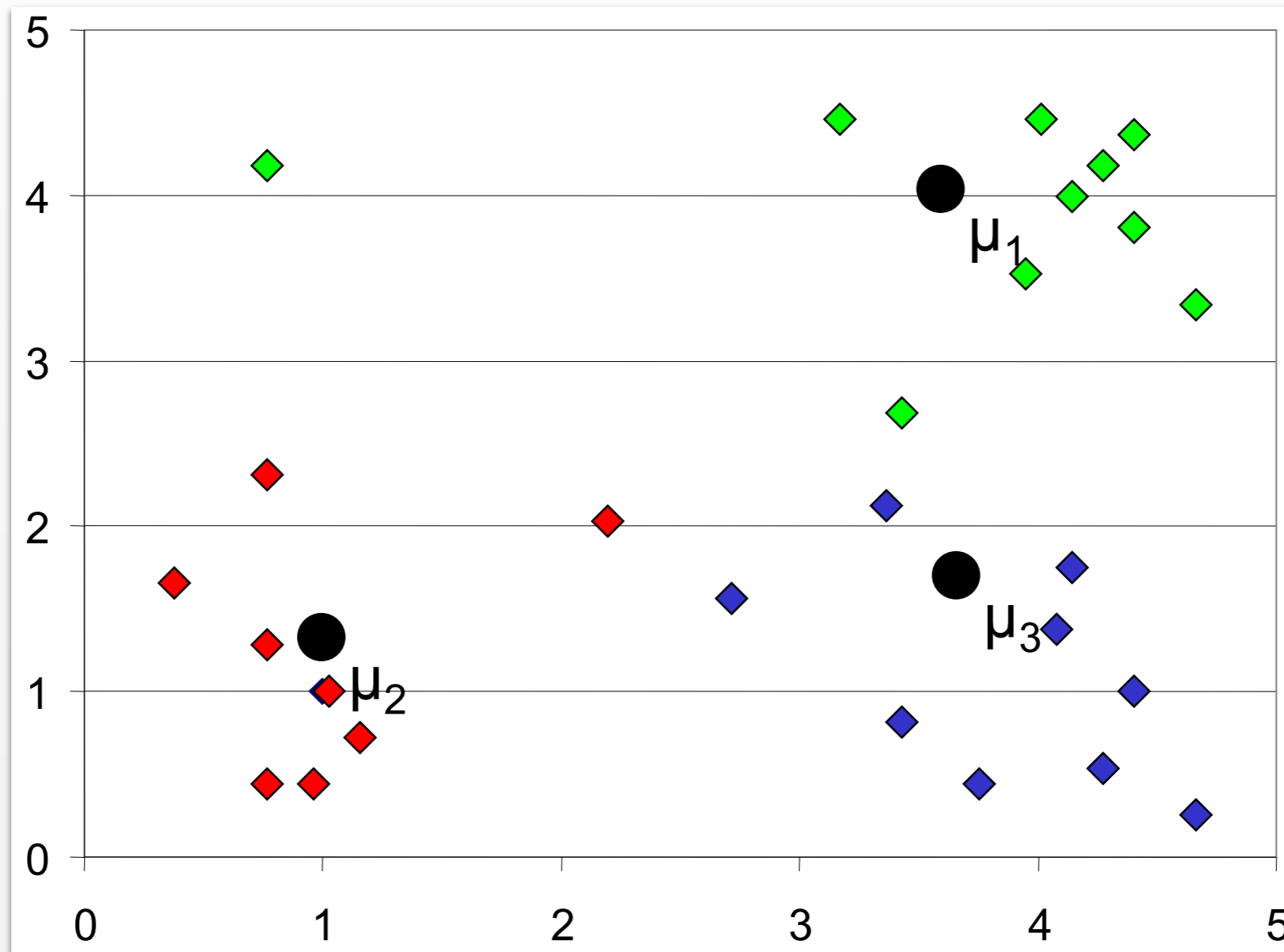
Randomly initialize K centroids μ_k

K-means Clustering



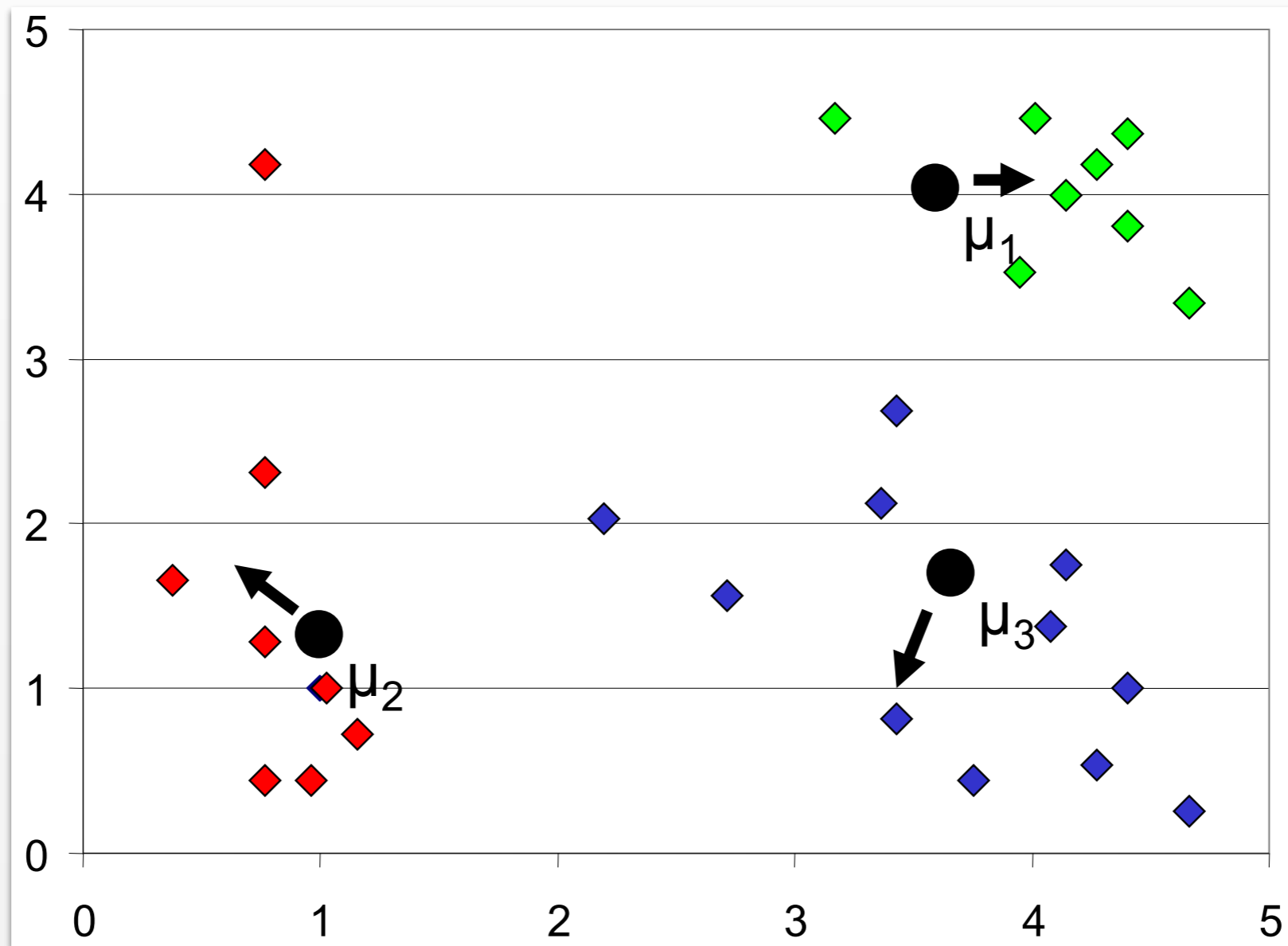
Assign each point to closest centroid,
then update centroids to average of points

K-means Clustering



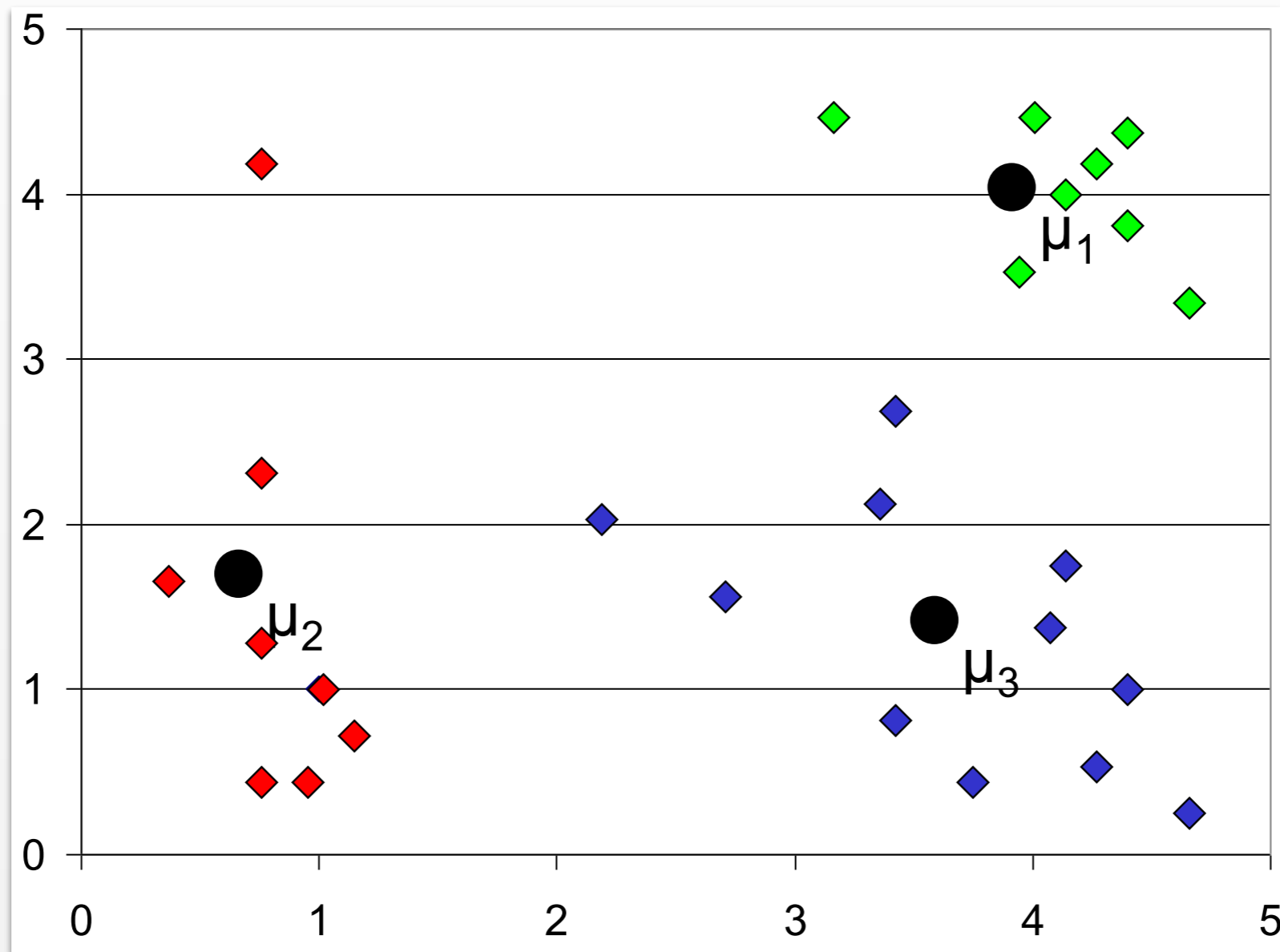
Assign each point to closest centroid,
then update centroids to average of points

K-means Clustering



Repeat until convergence
(no points reassigned, means unchanged)

K-means Clustering



Repeat until convergence
(no points reassigned, means unchanged)

K-means Algorithm

Input: $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$
Number of clusters K

Initialize: K random centroids $\mu_1, \mu_2, \dots, \mu_K$

Repeat Until Convergence

- 1 For $i = 1, \dots, K$ do
 $C_i = \{\mathbf{x} \in X \mid i = \arg \min_{1 \leq j \leq K} \|\mathbf{x} - \mu_j\|^2\}$
- 2 For $i = 1, \dots, K$ do
 $\mu_i = \arg \min_{\mathbf{z}} \sum_{\mathbf{x} \in C_i} \|\mathbf{z} - \mathbf{x}\|^2$

Output: C_1, C_2, \dots, C_K

K-means Algorithm

Input: $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$
Number of clusters K

Initialize: K random centroids $\mu_1, \mu_2, \dots, \mu_K$

Repeat Until Convergence

1 For $i = 1, \dots, K$ do
 $C_i = \{\mathbf{x} \in X \mid i = \arg \min_{1 \leq j \leq K} \|\mathbf{x} - \mu_j\|^2\}$

2 For $i = 1, \dots, K$ do
 $\mu_i = \arg \min_{\mathbf{z}} \sum_{\mathbf{x} \in C_i} \|\mathbf{z} - \mathbf{x}\|^2$

Output: C_1, C_2, \dots, C_K

- K-means: Set μ to mean of points in C
- K-medoids: Set $\mu = x$ for point in C with minimum SSE

K-means Complexity

Cost function

$$Cost = \sum_{i=1}^K \sum_{\mathbf{x} \in C_i} \| \mathbf{x} - \mu_i \|^2$$

What is the computational complexity (per iteration) for N points with D features and K clusters?

K-means Complexity

Cost function

$$Cost = \sum_{1}^K \sum_{\mathbf{x} \in C_i} \| \mathbf{x} - \mu_i \|^2$$

Complexity: $O(N K D \text{ #iterations})$

Mini-batch K-means

Web-scale k-means clustering

D Sculley - [Proceedings of the 19th international conference on ..., 2010 - dl.acm.org](#)

Abstract We present two modifications to the popular k-means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. First, we propose the use of mini-batch optimization for k-means ...

[Cited by 152](#) [Related articles](#) [All 11 versions](#) [Cite](#) [Save](#)

Mini-batch K-means

WWW 2010 • Poster

April 26-30 • Raleigh • NC • USA

Web-Scale K-Means Clustering

D. Sculley
Google, Inc. Pittsburgh, PA USA
dsculley@google.com

ABSTRACT

We present two modifications to the popular k -means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. First, we propose the use of mini-batch optimization for k -means clustering. This reduces computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent. Second, we achieve sparsity with projected gradient descent, and give a fast ϵ -accurate projection onto the $L1$ -ball. Source code is freely available: <http://code.google.com/p/sofia-ml>

Categories and Subject Descriptors

I.5.3 [Computing Methodologies]: Pattern Recognition—Clustering

General Terms

Algorithms, Performance, Experimentation

Keywords

unsupervised clustering, scalability, sparse solutions

1. CLUSTERING AND THE WEB

Unsupervised clustering is an important task in a range of web-based applications, including grouping search results, near-duplicate detection, and news aggregation to name but a few. Lloyd's classic k -means algorithm remains a popular choice for real-world clustering tasks [6]. However, the standard batch algorithm is slow for large data sets. Even optimized batch k -means variants exploiting triangle inequality [3] cannot cheaply meet the latency needs of user-facing applications when clustering results on large data sets are required in a fraction of a second.

This paper proposes a mini-batch k -means variant that yields excellent clustering results with low computation cost on large data sets. We also give methods for learning sparse cluster centers that reduce storage and network cost.

2. MINI-BATCH K-MEANS

The k -means optimization problem is to find the set C of cluster centers $\mathbf{c} \in \mathbb{R}^m$, with $|C| = k$, to minimize over a set

X of examples $\mathbf{x} \in \mathbb{R}^m$ the following objective function:

$$\min \sum_{\mathbf{x} \in X} \|f(C, \mathbf{x}) - \mathbf{x}\|^2$$

Here, $f(C, \mathbf{x})$ returns the nearest cluster center $\mathbf{c} \in C$ to \mathbf{x} using Euclidean distance. It is well known that although this problem is NP-hard in general, gradient descent methods converge to a local optimum when seeded with an initial set of k examples drawn uniformly at random from X [1].

The classic batch k -means algorithm is expensive for large data sets, requiring $O(kns)$ computation time where n is the number of examples and s is the maximum number of non-zero elements in any example vector. Bottou and Bengio proposed an online, stochastic gradient descent (SGD) variant that computed a gradient descent step on one example at a time [1]. While SGD converges quickly on large data sets, it finds lower quality solutions than the batch algorithm due to stochastic noise [1].

Algorithm 1 Mini-batch k -Means.

```
1: Given:  $k$ , mini-batch size  $b$ , iterations  $t$ , data set  $X$ 
2: Initialize each  $\mathbf{c} \in C$  with an  $\mathbf{x}$  picked randomly from  $X$ 
3:  $\mathbf{v} \leftarrow 0$ 
4: for  $i = 1$  to  $t$  do
5:    $M \leftarrow b$  examples picked randomly from  $X$ 
6:   for  $\mathbf{x} \in M$  do
7:      $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$  // Cache the center nearest to  $\mathbf{x}$ 
8:   end for
9:   for  $\mathbf{x} \in M$  do
10:     $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$  // Get cached center for this  $\mathbf{x}$ 
11:     $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$  // Update per-center counts
12:     $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$  // Get per-center learning rate
13:     $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$  // Take gradient step
14:   end for
15: end for
```

We propose the use of mini-batch optimization for k -means clustering, given in Algorithm 1. The motivation behind this method is that mini-batches tend to have lower stochastic noise than individual examples in SGD (allowing convergence to better solutions) but do not suffer increased computational cost when data sets grow large with redundant examples. We use per-center learning rates for fast convergence, in the manner of [1]; convergence properties follow closely from this prior result [1].

Experiments. We tested the mini-batch k -means against both Lloyd's batch k -means [6] and the SGD variant of [1]. We used the standard RCV1 collection of documents [4] for

WWW 2010 • Poster

April 26-30 • Raleigh • NC • USA

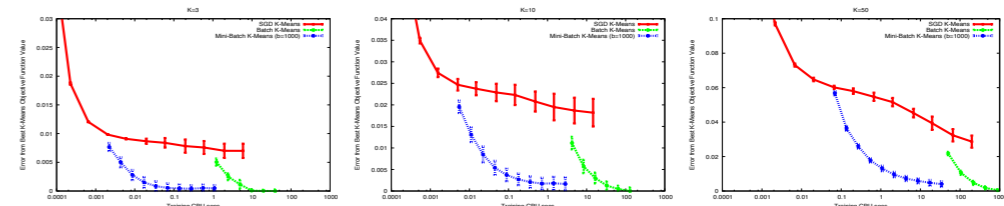


Figure 1: Convergence Speed. The mini-batch method (blue) is orders of magnitude faster than the full batch method (green), while converging to significantly better solutions than the online SGD method (red).

our experiments. To assess performance at scale, the set of 781,265 examples were used for training and the remaining 23,149 examples for testing. On each trial, the same random initial cluster centers were used for each method. We evaluated the learned cluster centers using the k -means objective function on the held-out test set; we report fractional error from the best value found by the batch algorithm run to convergence. We set the mini-batch b to 1000 based on separate initial tests; results were robust for a range of b .

The results (Fig. 1) show a clear win for mini-batch k -means. The mini-batch method converged to a near optimal value several orders of magnitude faster than the full batch method, and also achieved significantly better solutions than SGD. Additional experiments (omitted for space) showed that mini-batch k -means is several times faster on large data sets than batch k -means exploiting triangle inequality [3].

For small values of k , the mini-batch methods were able to produce near-best cluster centers for nearly a million documents in a fraction of a CPU second on a single ordinary 2.4 GHz machine. This makes real-time clustering practical for user-facing applications.

3. SPARSE CLUSTER CENTERS

We modify mini-batch k -means to find sparse cluster centers, allowing for compact storage and low network cost. The intuition for seeking sparse cluster centers for document clusters is that term frequencies follow a power-law distribution. Many terms in a given cluster will only occur in one or two documents, giving them very low weight in the cluster center. It is likely that for a locally optimal center \mathbf{c} , there is a nearby point \mathbf{c}' with many fewer non-zero values.

Sparsity may be induced in gradient descent using the projected-gradient method, projecting a given \mathbf{v} to the nearest point in an $L1$ -ball of radius λ after each update [2]. Thus, for mini-batch k -means we achieve sparsity by performing an $L1$ -ball projection on each cluster center \mathbf{c} after each mini-batch iteration.

Algorithm 2 ϵ -L1: an ϵ -Accurate Projection to $L1$ Ball.

```
1: Given:  $\epsilon$  tolerance,  $L1$ -ball radius  $\lambda$ , vector  $\mathbf{c} \in \mathbb{R}^m$ 
2: if  $\|\mathbf{c}\|_1 \leq \lambda + \epsilon$  then exit
3: upper  $\leftarrow \|\mathbf{c}\|_\infty$ ; lower  $\leftarrow 0$ ; current  $\leftarrow \|\mathbf{c}\|_1$ 
4: while current  $> \lambda(1 + \epsilon)$  or current  $< \lambda$  do
5:    $\theta \leftarrow \frac{\text{upper} + \text{lower}}{2.0}$  // Get  $L1$  value for this  $\theta$ 
6:   current  $\leftarrow \sum_{c_i \neq 0} \max(0, |c_i| - \theta)$ 
7:   if current  $\leq \lambda$  then upper  $\leftarrow \theta$  else lower  $\leftarrow \theta$ 
8: end while
9: for  $i = 1$  to  $m$  do
10:   $c_i \leftarrow \text{sign}(c_i) * \max(0, |c_i| - \theta)$  // Do the projection
11: end for
```

Fast $L1$ Projections. Applying $L1$ constraints to k -means clustering has been studied in forthcoming work by Witten and Tibshirani [5]. There, a hard $L1$ constraint was applied in the full batch setting of maximizing between-cluster distance for k -means (rather than minimizing the k -means objective function directly); the work did not discuss how to perform this projection efficiently.

The projection to the $L1$ ball can be performed effectively using, for example, the linear time $L1$ -ball projection algorithm of Duchi *et al.* [2], referred to here as LTLIP. We give an alternate method in Algorithm 2, observing that the exact $L1$ radius is not critical for sparsity. This simple approximation algorithm uses bisection to find a value θ that projects \mathbf{c} to an $L1$ ball with radius between λ and $(1 + \epsilon)\lambda$. Our method is easy to implement and is also significantly faster in practice than LTLIP due to memory concurrency.

METHOD	λ	#NON-ZERO'S	TEST OBJECTIVE	CPU'S
full batch	-	200,319	0 (baseline)	133.96
LTLIP	5.0	46,446	.004 (.002-.006)	0.51
ϵ -L1	5.0	44,060	.007 (.005-.008)	0.27
LTLIP	1.0	3,181	.018 (.016-.019)	0.48
ϵ -L1	1.0	2,547	.028 (.027-.029)	0.19

Results. Using the same set-up as above, we tested Duchi *et al.*'s linear time algorithm and our ϵ -accurate projection for mini-batch k -means, with a range of λ values. The value of ϵ was arbitrarily set to 0.01. We report values for $k = 10$, $b = 1000$, and $t = 16$ (results for other values qualitatively similar). Compared with the full batch method, we achieve much sparser solutions. The approximate projection is roughly twice as fast as LTLIP and finds sparser solutions, but gives slightly worse performance on the test set. These results show that sparse clustering may cheaply be achieved with low latency for user-facing applications.

4. REFERENCES

- [1] L. Bottou and Y. Bengio. Convergence properties of the k -means algorithm. In *Advances in Neural Information Processing Systems*, 1995.
- [2] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the $L1$ -ball for learning in high dimensions. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.
- [3] C. Elkan. Using the triangle inequality to accelerate k -means. In *ICML '03: Proceedings of the 20th international conference on Machine learning*, 2003.
- [4] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5, 2004.
- [5] D. Witten and R. Tibshirani. A framework for feature selection in clustering. To Appear: *Journal of the American Statistical Association*, 2010.
- [6] X. Wu and V. Kumar. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC, 2009.

Mini-batch K-means

WWW 2010 • Poster

April 26-30 • Raleigh • NC • USA

Web-Scale K-Means Clustering

D. Sculley
Google, Inc. Pittsburgh, PA USA
dsculley@google.com

ABSTRACT

We present two modifications to the popular k -means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. First, we propose the use of mini-batch optimization for k -means clustering. This reduces computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent. Second, we achieve sparsity with projected gradient descent, and give a fast ϵ -accurate projection onto the $L1$ -ball. Source code is freely available: <http://code.google.com/p/sofia-ml>

Categories and Subject Descriptors

I.5.3 [Computing Methodologies]: Pattern Recognition—Clustering

General Terms

Algorithms, Performance, Experimentation

Keywords

unsupervised clustering, scalability, sparse solutions

1. CLUSTERING AND THE WEB

Unsupervised clustering is an important task in a range of web-based applications, including grouping search results, near-duplicate detection, and news aggregation to name but a few. Lloyd's classic k -means algorithm remains a popular choice for real-world clustering tasks [6]. However, the standard batch algorithm is slow for large data sets. Even optimized batch k -means variants exploiting triangle inequality [3] cannot cheaply meet the latency needs of user-facing applications when clustering results on large data sets are required in a fraction of a second.

This paper proposes a mini-batch k -means variant that yields excellent clustering results with low computation cost on large data sets. We also give methods for learning sparse cluster centers that reduce storage and network cost.

2. MINI-BATCH K-MEANS

The k -means optimization problem is to find the set C of cluster centers $\mathbf{c} \in \mathbb{R}^m$, with $|C| = k$, to minimize over a set

X of examples $\mathbf{x} \in \mathbb{R}^m$ the following objective function:

$$\min_{C \subseteq X} \sum_{\mathbf{x} \in X} \|f(C, \mathbf{x}) - \mathbf{x}\|^2$$

Here, $f(C, \mathbf{x})$ returns the nearest cluster center $\mathbf{c} \in C$ to \mathbf{x} using Euclidean distance. It is well known that although this problem is NP-hard in general, gradient descent methods converge to a local optimum when seeded with an initial set of k examples drawn uniformly at random from X [1].

The classic batch k -means algorithm is expensive for large data sets, requiring $O(kns)$ computation time where n is the number of examples and s is the maximum number of non-zero elements in any example vector. Bottou and Bengio proposed an online, stochastic gradient descent (SGD) variant that computed a gradient descent step on one example at a time [1]. While SGD converges quickly on large data sets, it finds lower quality solutions than the batch algorithm due to stochastic noise [1].

Algorithm 1 Mini-batch k -Means.

```
1: Given:  $k$ , mini-batch size  $b$ , iterations  $t$ , data set  $X$ 
2: Initialize each  $\mathbf{c} \in C$  with an  $\mathbf{x}$  picked randomly from  $X$ 
3:  $\mathbf{v} \leftarrow 0$ 
4: for  $i = 1$  to  $t$  do
5:    $M \leftarrow b$  examples picked randomly from  $X$ 
6:   for  $\mathbf{x} \in M$  do
7:      $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$  // Cache the center nearest to  $\mathbf{x}$ 
8:   end for
9:   for  $\mathbf{x} \in M$  do
10:     $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$  // Get cached center for this  $\mathbf{x}$ 
11:     $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$  // Update per-center counts
12:     $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$  // Get per-center learning rate
13:     $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$  // Take gradient step
14:   end for
15: end for
```

We propose the use of mini-batch optimization for k -means clustering, given in Algorithm 1. The motivation behind this method is that mini-batches tend to have lower stochastic noise than individual examples in SGD (allowing convergence to better solutions) but do not suffer increased computational cost when data sets grow large with redundant examples. We use per-center learning rates for fast convergence, in the manner of [1]; convergence properties follow closely from this prior result [1].

Experiments. We tested the mini-batch k -means against both Lloyd's batch k -means [6] and the SGD variant of [1]. We used the standard RCV1 collection of documents [4] for

WWW 2010 • Poster

April 26-30 • Raleigh • NC • USA

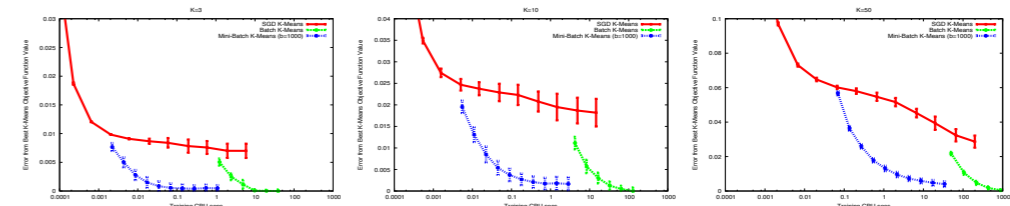


Figure 1: Convergence Speed. The mini-batch method (blue) is orders of magnitude faster than the full batch method (green), while converging to significantly better solutions than the online SGD method (red).

our experiments. To assess performance at scale, the set of 781,265 examples were used for training and the remaining 23,149 examples for testing. On each trial, the same random initial cluster centers were used for each method. We evaluated the learned cluster centers using the k -means objective function on the held-out test set; we report fractional error from the best value found by the batch algorithm run to convergence. We set the mini-batch b to 1000 based on separate initial tests; results were robust for a range of b .

The results (Fig. 1) show a clear win for mini-batch k -means. The mini-batch method converged to a near optimal value several orders of magnitude faster than the full batch method, and also achieved significantly better solutions than SGD. Additional experiments (omitted for space) showed that mini-batch k -means is several times faster on large data sets than batch k -means exploiting triangle inequality [3].

For small values of k , the mini-batch methods were able to produce near-best cluster centers for nearly a million documents in a fraction of a CPU second on a single ordinary 2.4 GHz machine. This makes real-time clustering practical for user-facing applications.

3. SPARSE CLUSTER CENTERS

We modify mini-batch k -means to find sparse cluster centers, allowing for compact storage and low network cost. The intuition for seeking sparse cluster centers for document clusters is that term frequencies follow a power-law distribution. Many terms in a given cluster will only occur in one or two documents, giving them very low weight in the cluster center. It is likely that for a locally optimal center \mathbf{c} , there is a nearby point \mathbf{c}' with many fewer non-zero values.

Sparsity may be induced in gradient descent using the projected-gradient method, projecting a given \mathbf{v} to the nearest point in an $L1$ -ball of radius λ after each update [2]. Thus, for mini-batch k -means we achieve sparsity by performing an $L1$ -ball projection on each cluster center \mathbf{c} after each mini-batch iteration.

Algorithm 2 ϵ -L1: an ϵ -Accurate Projection to $L1$ Ball.

```
1: Given:  $\epsilon$  tolerance,  $L1$ -ball radius  $\lambda$ , vector  $\mathbf{c} \in \mathbb{R}^m$ 
2: if  $\|\mathbf{c}\|_1 \leq \lambda + \epsilon$  then exit
3: upper  $\leftarrow \|\mathbf{c}\|_\infty$ ; lower  $\leftarrow 0$ ; current  $\leftarrow \|\mathbf{c}\|_1$ 
4: while current  $> \lambda(1 + \epsilon)$  or current  $< \lambda$  do
5:    $\theta \leftarrow \frac{\text{upper} + \text{lower}}{2.0}$  // Get  $L1$  value for this  $\theta$ 
6:   current  $\leftarrow \sum_{c_i \neq 0} \max(0, |c_i| - \theta)$ 
7:   if current  $\leq \lambda$  then upper  $\leftarrow \theta$  else lower  $\leftarrow \theta$ 
8: end while
9: for  $i = 1$  to  $m$  do
10:   $c_i \leftarrow \text{sign}(c_i) * \max(0, |c_i| - \theta)$  // Do the projection
11: end for
```

Fast $L1$ Projections. Applying $L1$ constraints to k -means clustering has been studied in forthcoming work by Witten and Tibshirani [5]. There, a hard $L1$ constraint was applied in the full batch setting of maximizing between-cluster distance for k -means (rather than minimizing the k -means objective function directly); the work did not discuss how to perform this projection efficiently.

The projection to the $L1$ ball can be performed effectively using, for example, the linear time $L1$ -ball projection algorithm of Duchi *et al.* [2], referred to here as LTLIP. We give an alternate method in Algorithm 2, observing that the exact $L1$ radius is not critical for sparsity. This simple approximation algorithm uses bisection to find a value θ that projects \mathbf{c} to an $L1$ ball with radius between λ and $(1 + \epsilon)\lambda$. Our method is easy to implement and is also significantly faster in practice than LTLIP due to memory concurrency.

METHOD	λ	#NON-ZERO'S	TEST OBJECTIVE	CPU'S
full batch	-	200,319	0 (baseline)	133.96
LTLIP	5.0	46,446	.004 (.002-.006)	0.51
ϵ -L1	5.0	44,060	.007 (.005-.008)	0.27
LTLIP	1.0	3,181	.018 (.016-.019)	0.48
ϵ -L1	1.0	2,547	.028 (.027-.029)	0.19

Results. Using the same set-up as above, we tested Duchi *et al.*'s linear time algorithm and our ϵ -accurate projection for mini-batch k -means, with a range of λ values. The value of ϵ was arbitrarily set to 0.01. We report values for $k = 10$, $b = 1000$, and $t = 16$ (results for other values qualitatively similar). Compared with the full batch method, we achieve much sparser solutions. The approximate projection is roughly twice as fast as LTLIP and finds sparser solutions, but gives slightly worse performance on the test set. These results show that sparse clustering may cheaply be achieved with low latency for user-facing applications.

4. REFERENCES

- [1] L. Bottou and Y. Bengio. Convergence properties of the k -means algorithm. In *Advances in Neural Information Processing Systems*, 1995.
- [2] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the $L1$ -ball for learning in high dimensions. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.
- [3] C. Elkan. Using the triangle inequality to accelerate k -means. In *ICML '03: Proceedings of the 20th international conference on Machine learning*, 2003.
- [4] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5, 2004.
- [5] D. Witten and R. Tibshirani. A framework for feature selection in clustering. To Appear: *Journal of the American Statistical Association*, 2010.
- [6] X. Wu and V. Kumar. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC, 2009.

(2-page abstract)

Mini-batch K-means

Algorithm 1 Mini-batch k -Means.

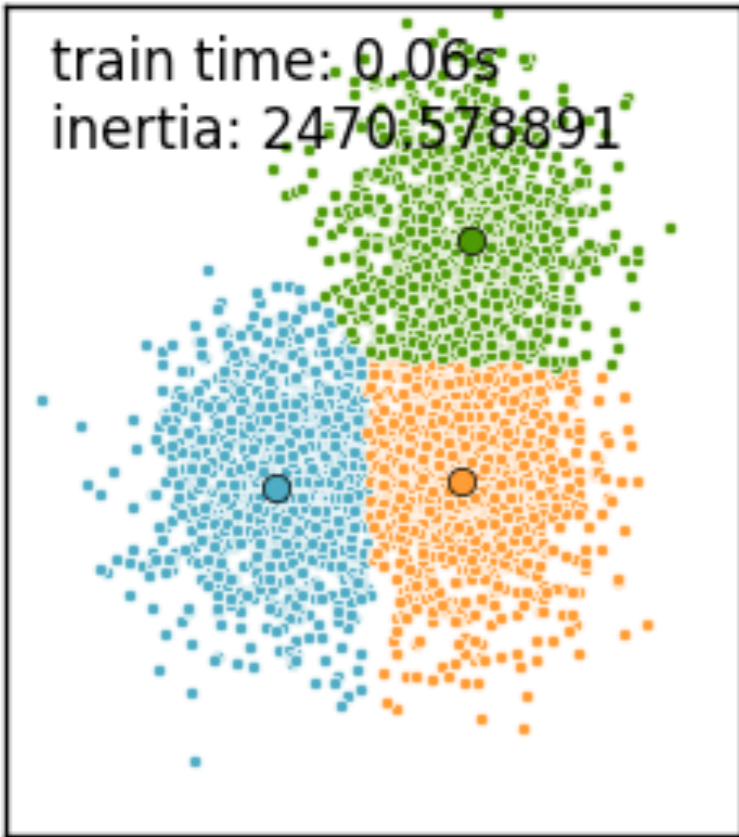
```
1: Given:  $k$ , mini-batch size  $b$ , iterations  $t$ , data set  $X$ 
2: Initialize each  $\mathbf{c} \in C$  with an  $\mathbf{x}$  picked randomly from  $X$ 
3:  $\mathbf{v} \leftarrow 0$ 
4: for  $i = 1$  to  $t$  do
5:    $M \leftarrow b$  examples picked randomly from  $X$ 
6:   for  $\mathbf{x} \in M$  do
7:      $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$  // Cache the center nearest to  $\mathbf{x}$ 
8:   end for
9:   for  $\mathbf{x} \in M$  do
10:     $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$  // Get cached center for this  $\mathbf{x}$ 
11:     $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$  // Update per-center counts
12:     $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$  // Get per-center learning rate
13:     $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$  // Take gradient step
14:   end for
15: end for
```

Complexity: $O(N M K D t)$

Mini-batch K-means

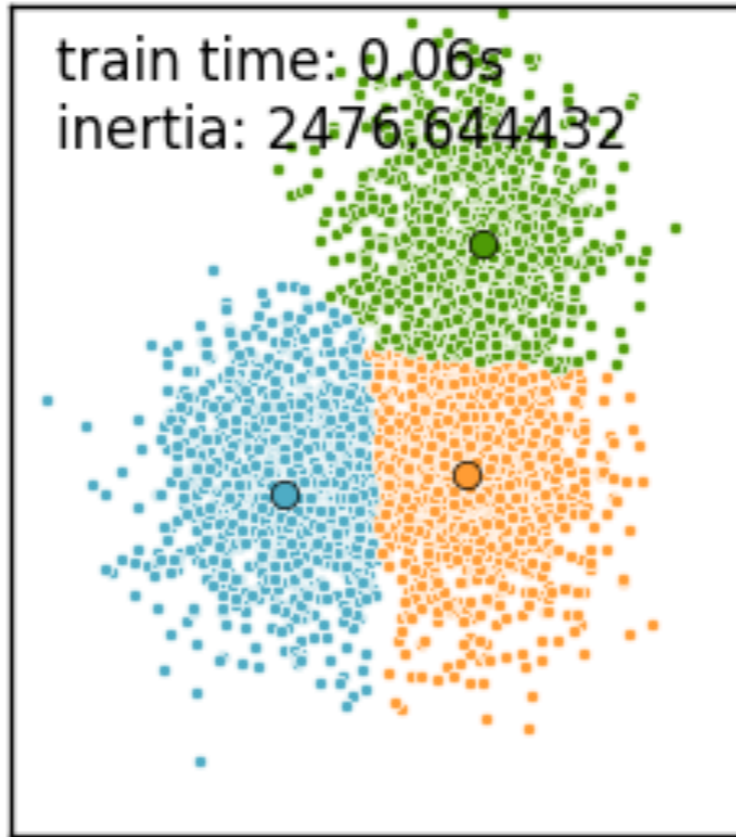
KMeans

train time: 0.06s
inertia: 2470.578891

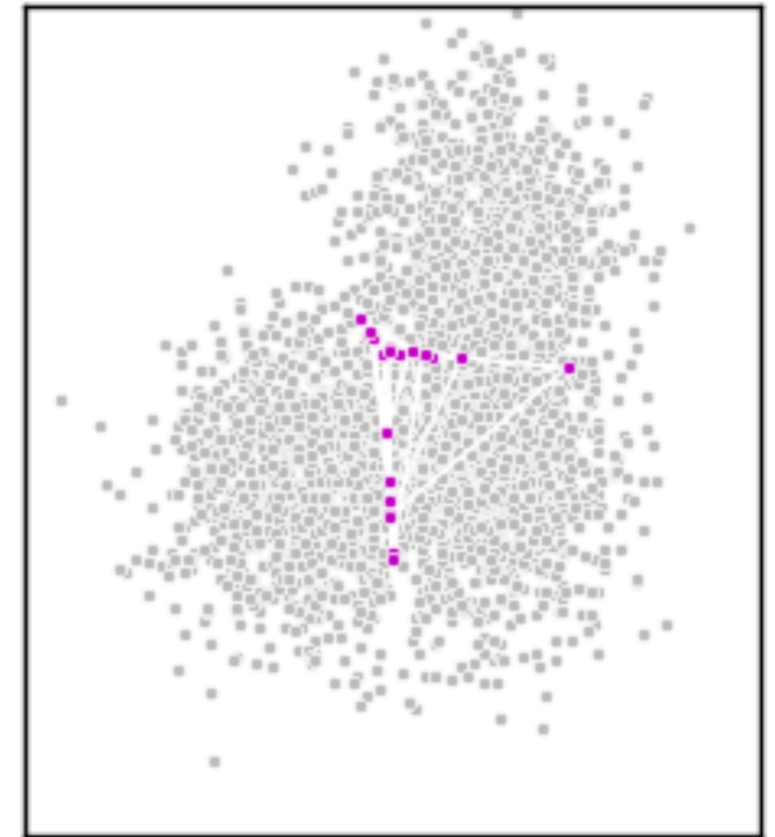


MiniBatchKMeans

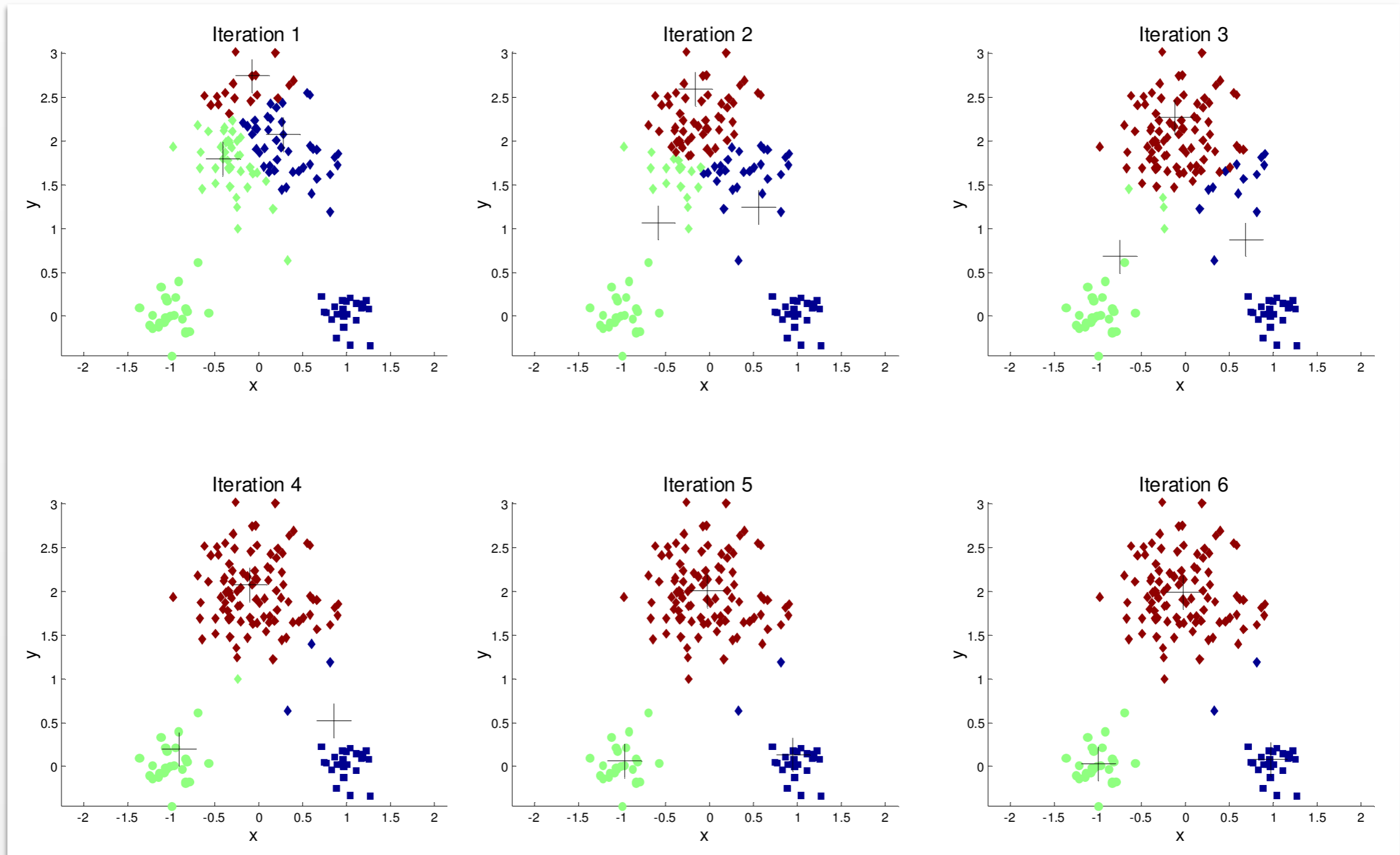
train time: 0.06s
inertia: 2476.644432



Difference

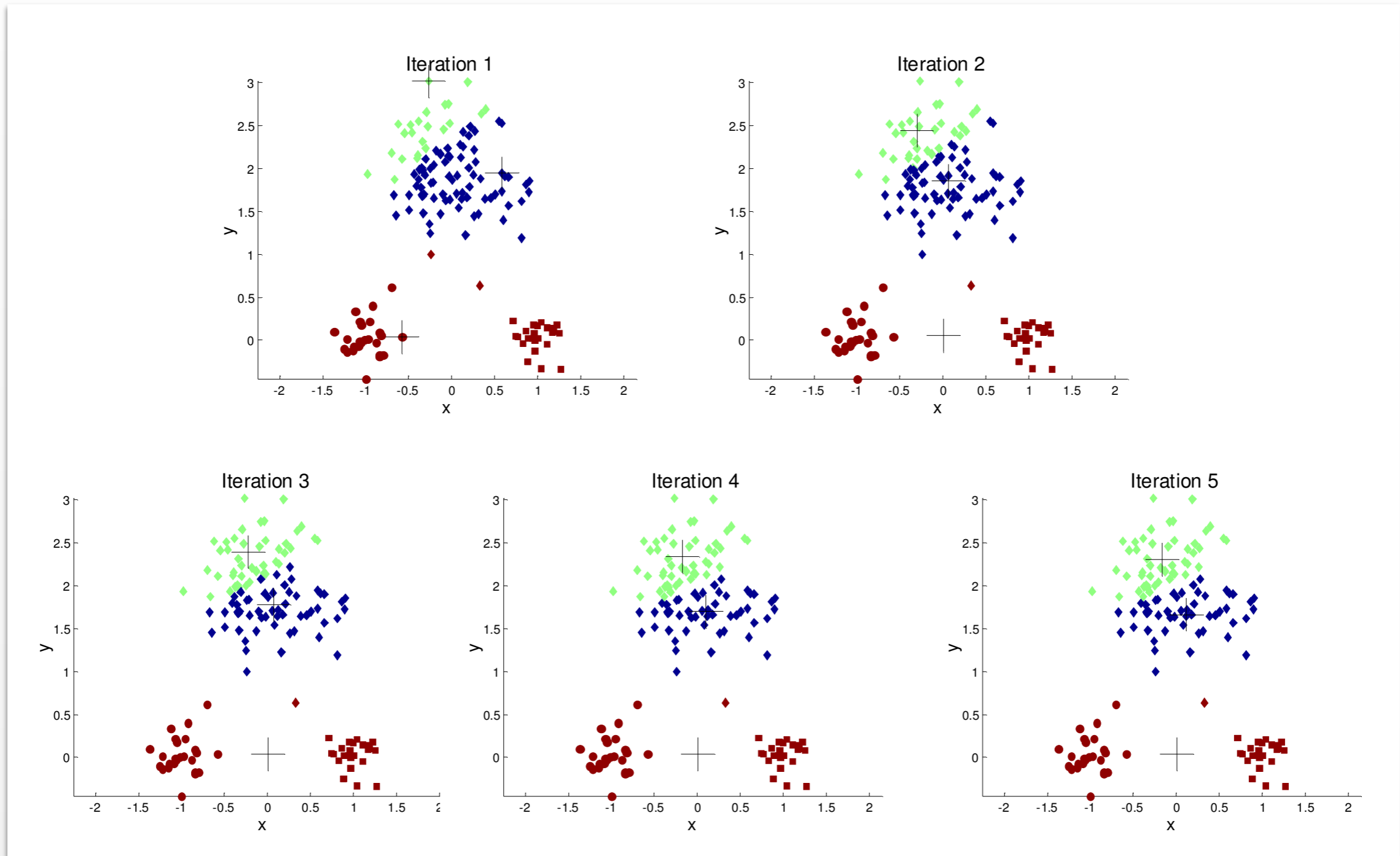


Importance of Initial Centroids



“Good” initial choice

Importance of Initial Centroids



“Bad” initial choice

Importance of Initial Centroids

What is the chance of randomly selecting one point from each of K clusters?

(assume each cluster has size $n = N/K$)

Importance of Initial Centroids

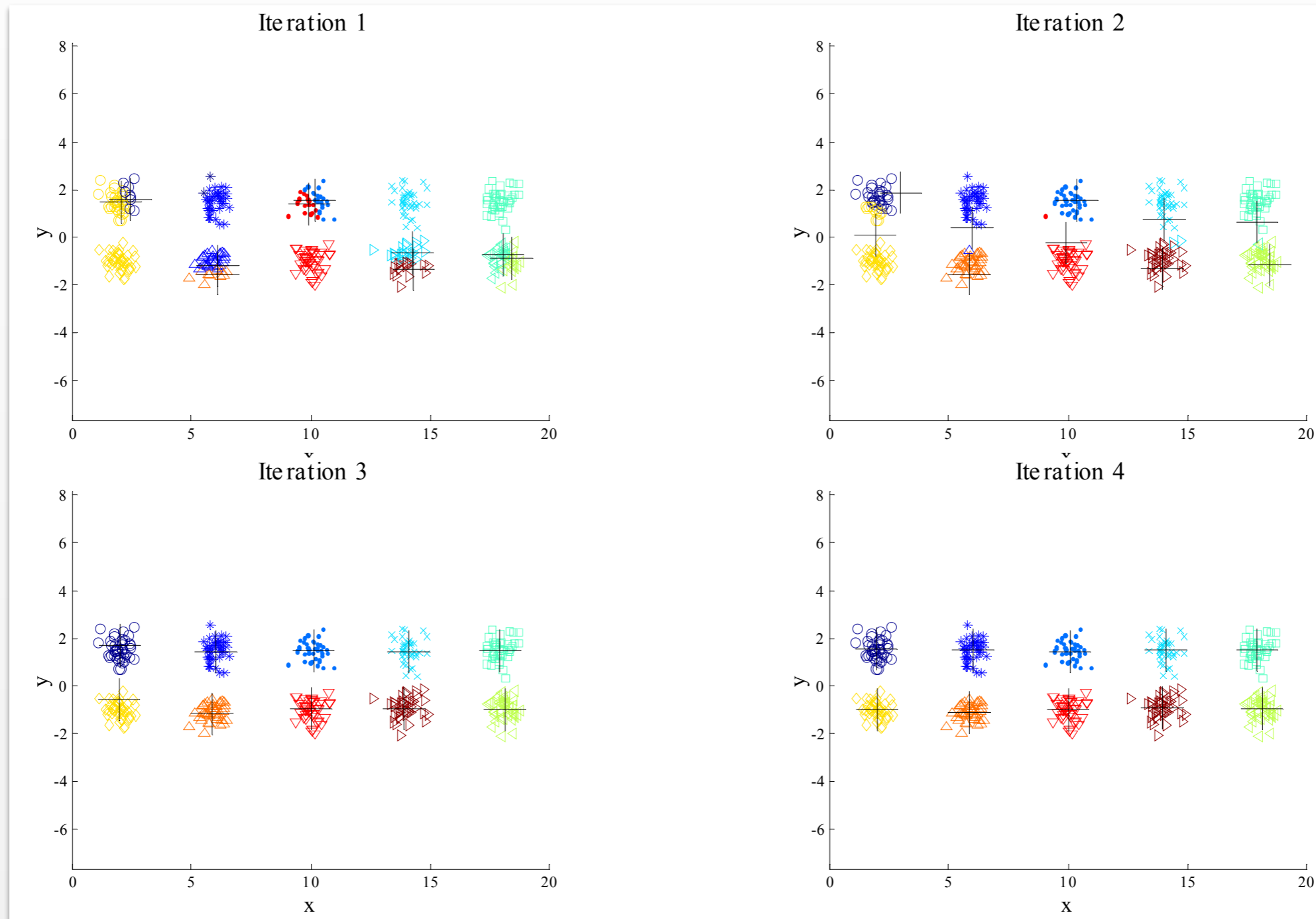
What is the chance of randomly selecting one point from each of K clusters?

(assume each cluster has size $n = N/K$)

$$\frac{\text{ways to select one from each cluster}}{\text{ways to select } K \text{ centroids}} = \frac{K!n^K}{(Kn)^K} = \frac{K!}{K^K}$$

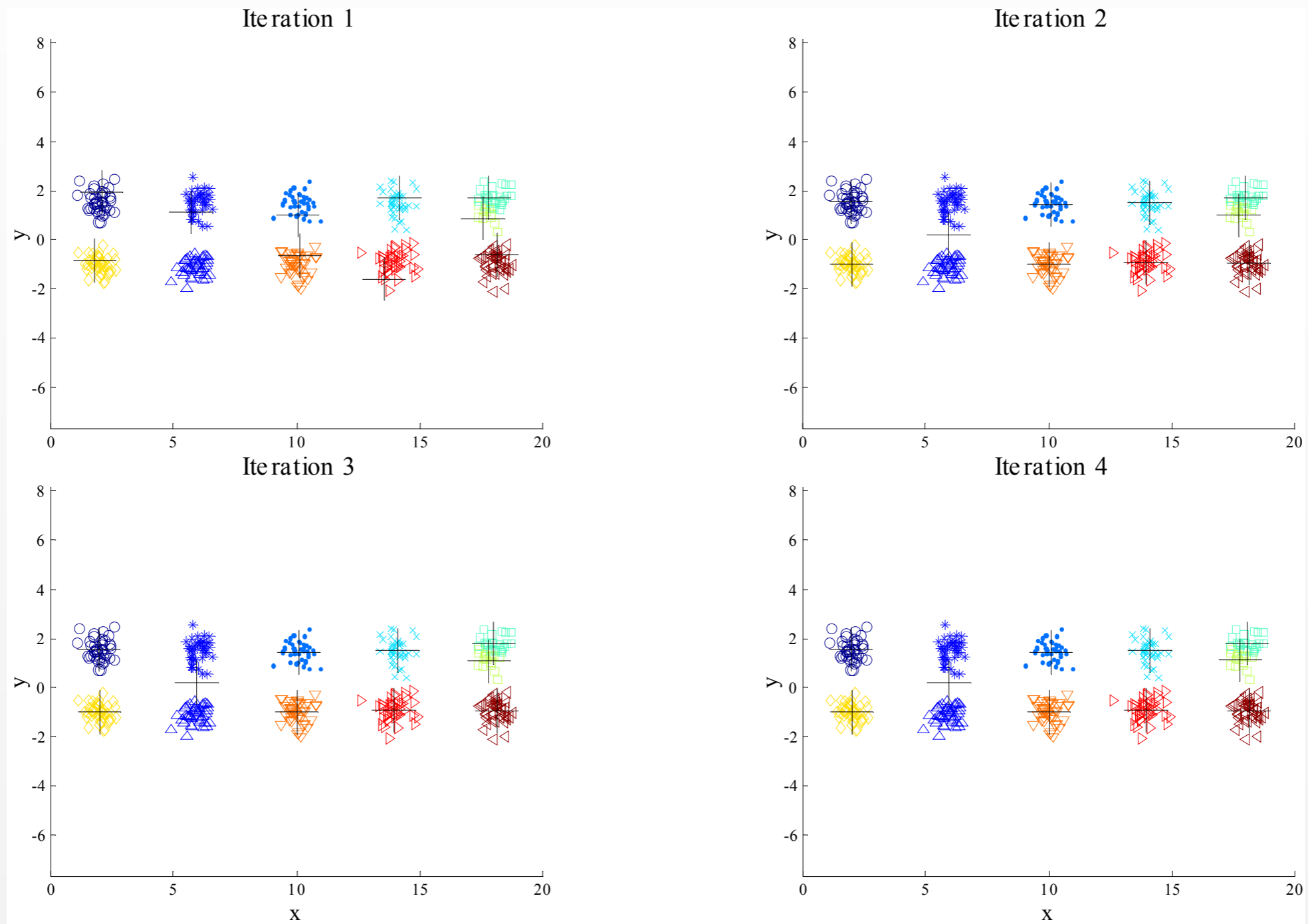
Implication: We will almost always have multiple initial centroids in same cluster.

Importance of Initial Centroids



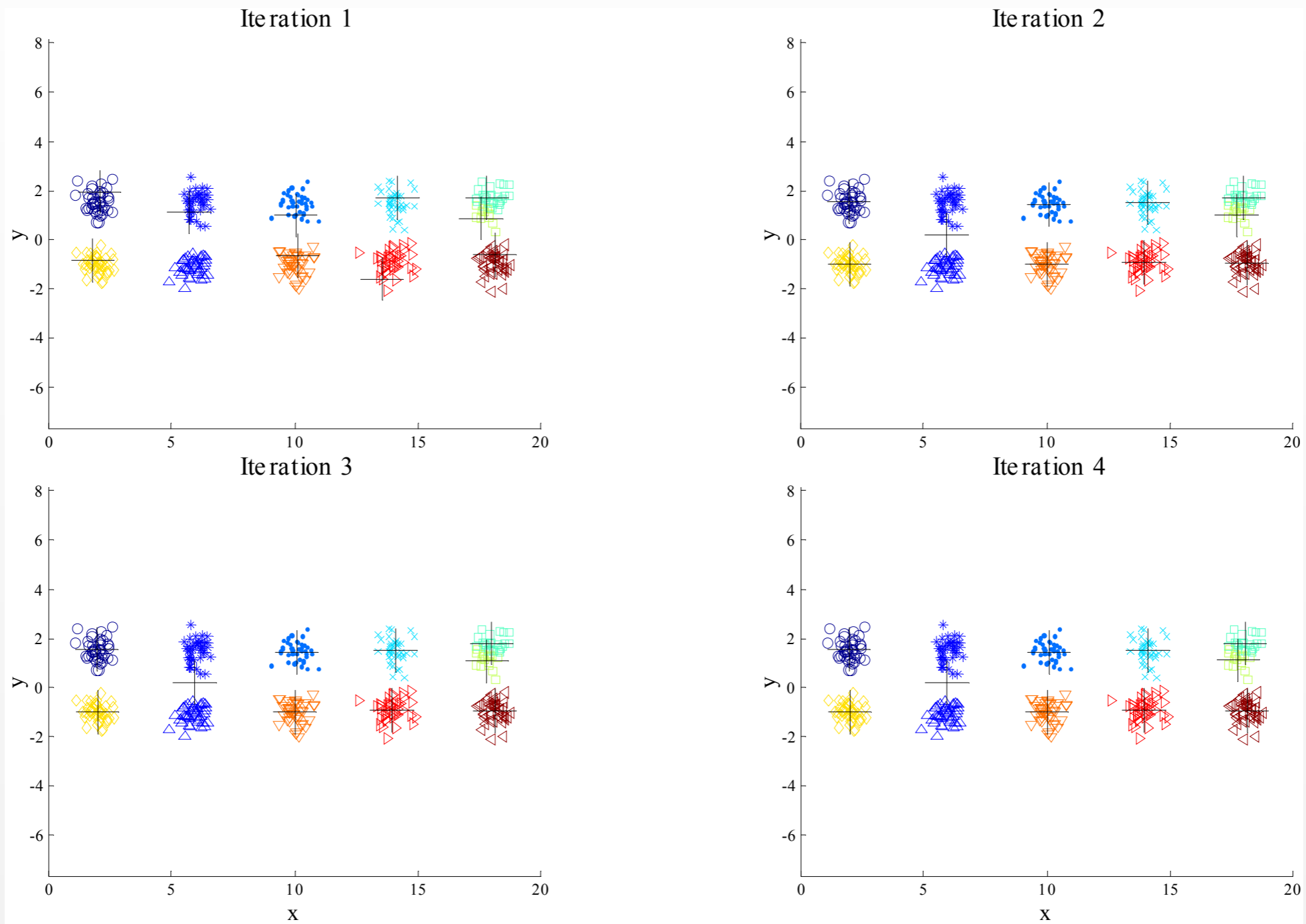
5 pairs of clusters, two initial points in each pair

Importance of Initial Centroids



Some pairs have 3 initial centroids, some have 1

Importance of Initial Centroids



Conclusion: Heuristic search leads to local optima

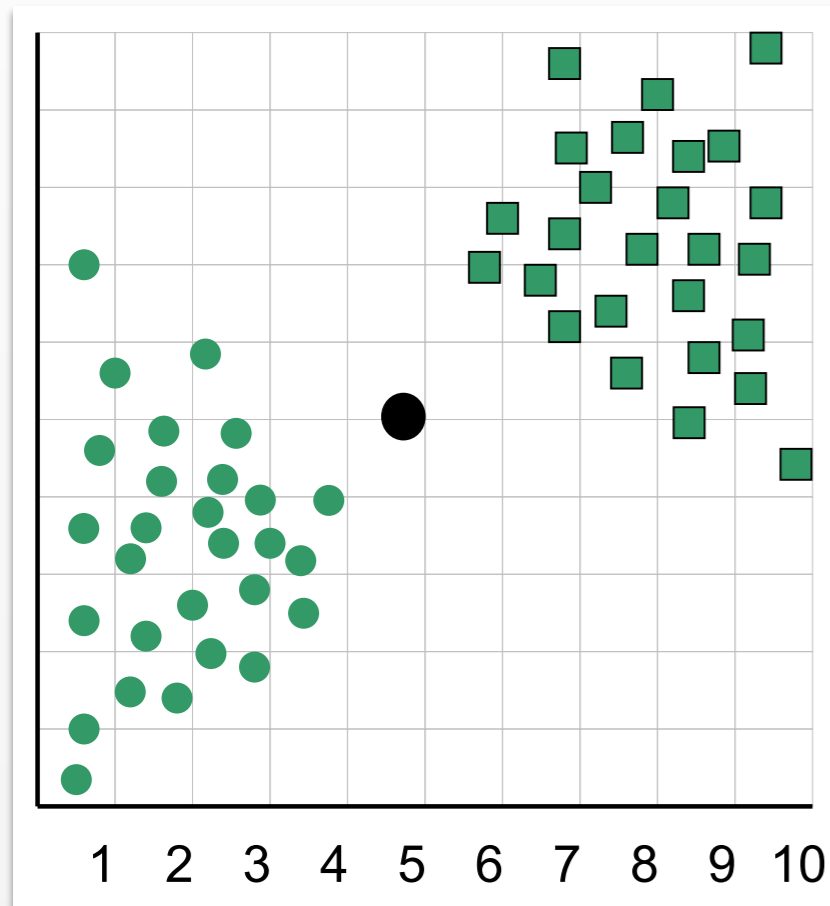
Importance of Initial Centroids

Initialization tricks

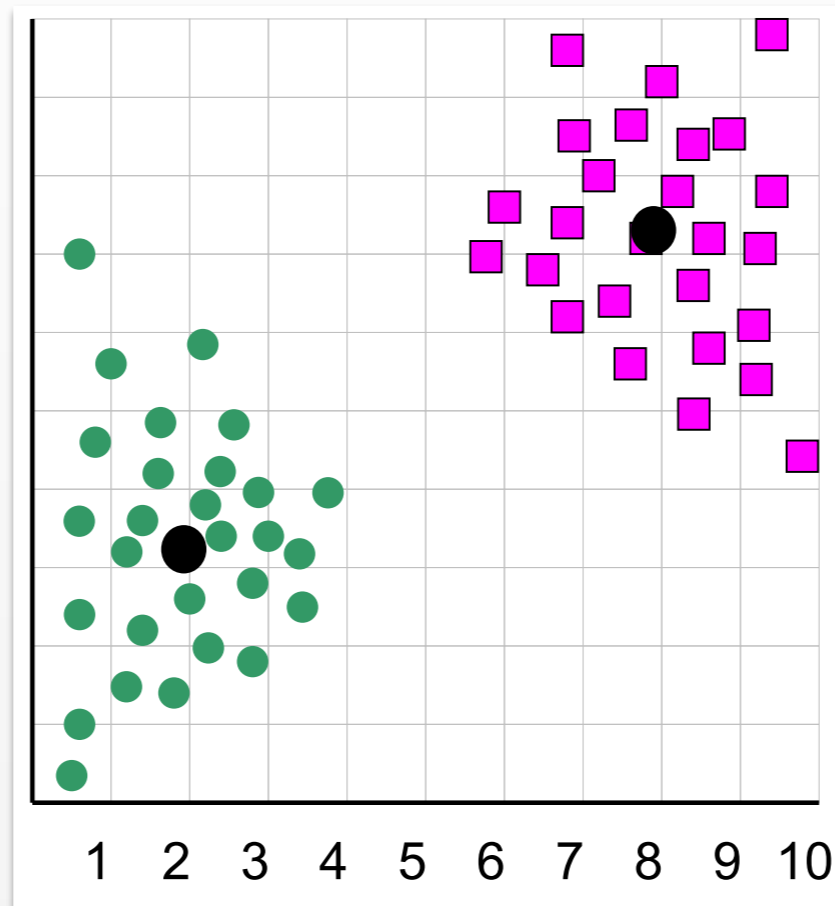
- Use multiple restarts
- Initialize with hierarchical clustering
- Select more than K points,
keep most widely separated points

Choosing K

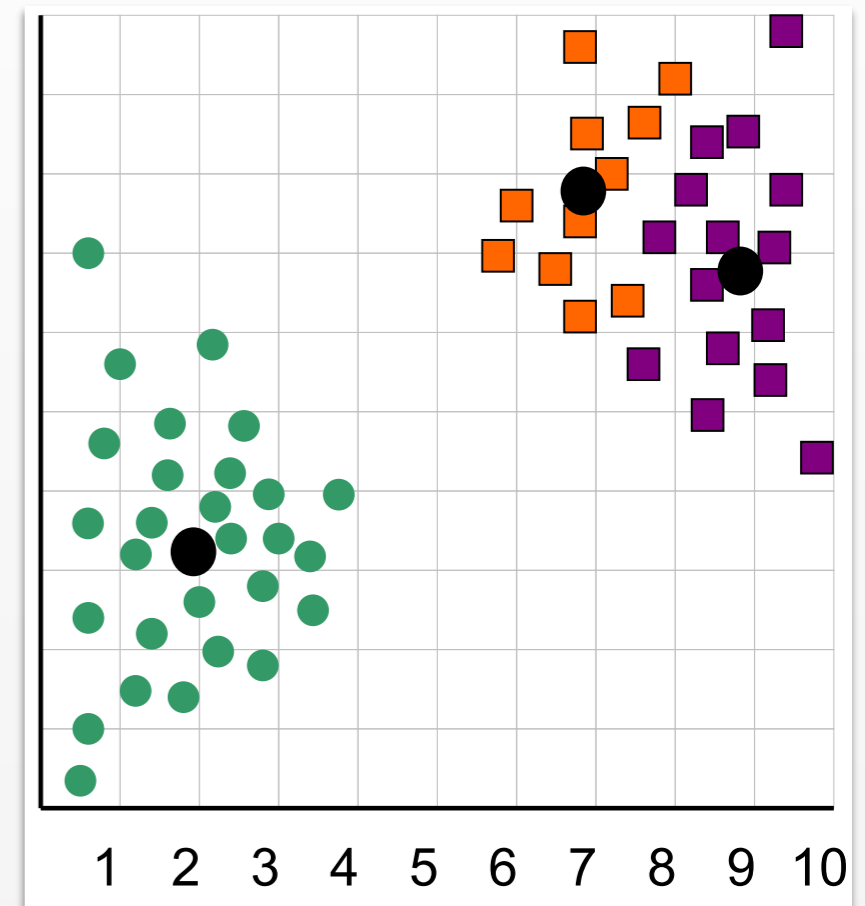
K=1, SSE=873



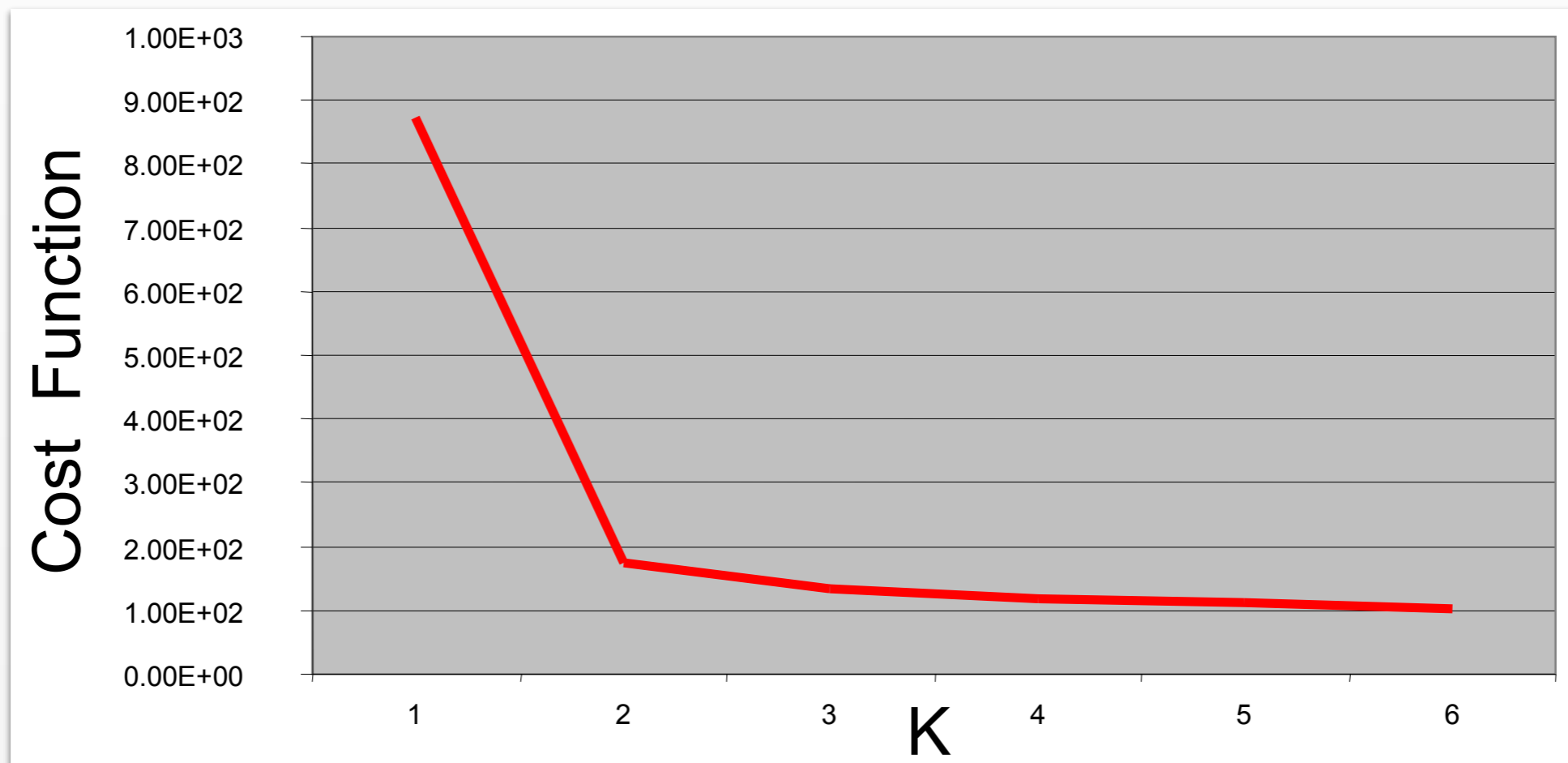
K=2, SSE=173



K=3, SSE=134

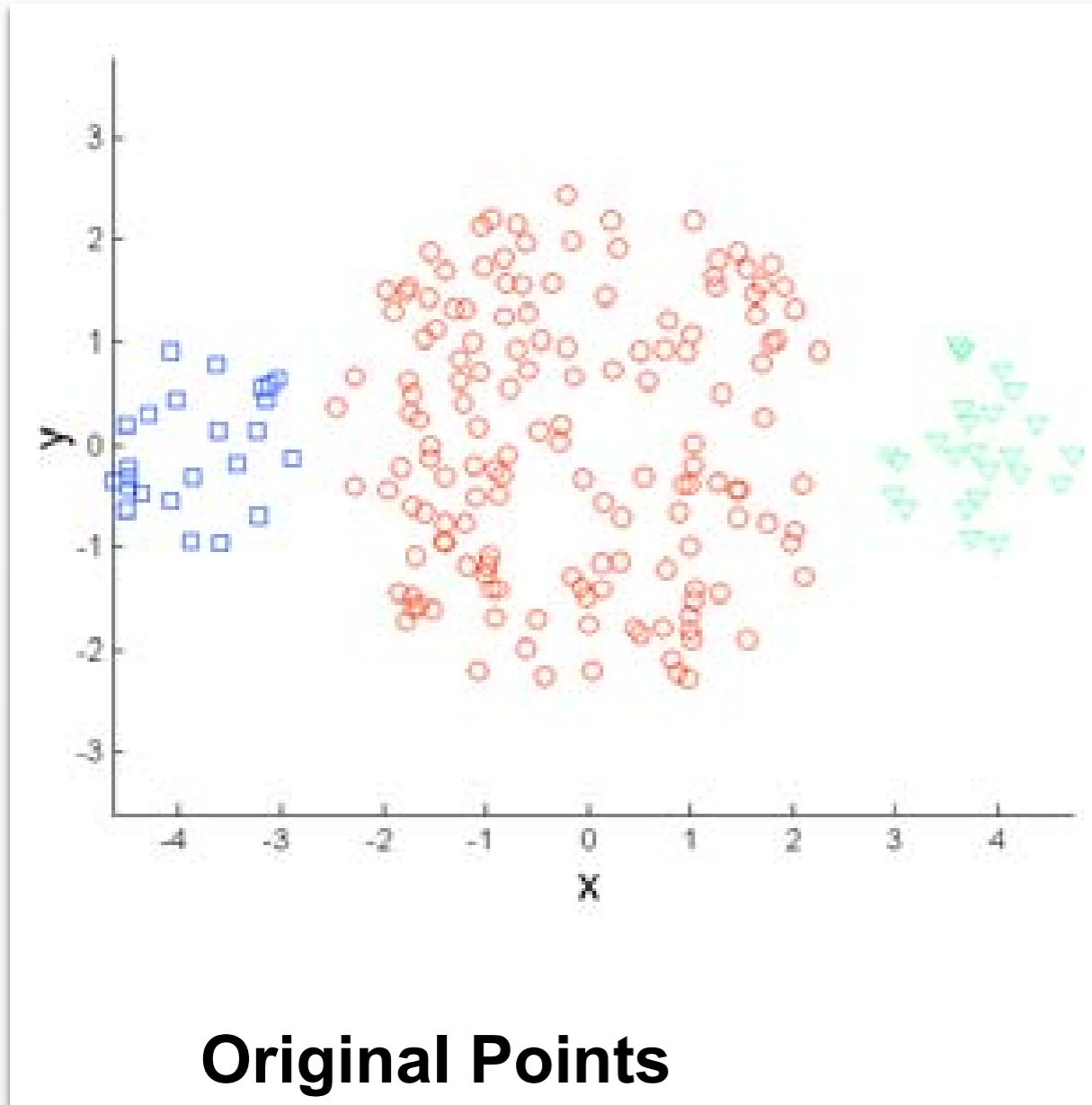


Choosing K

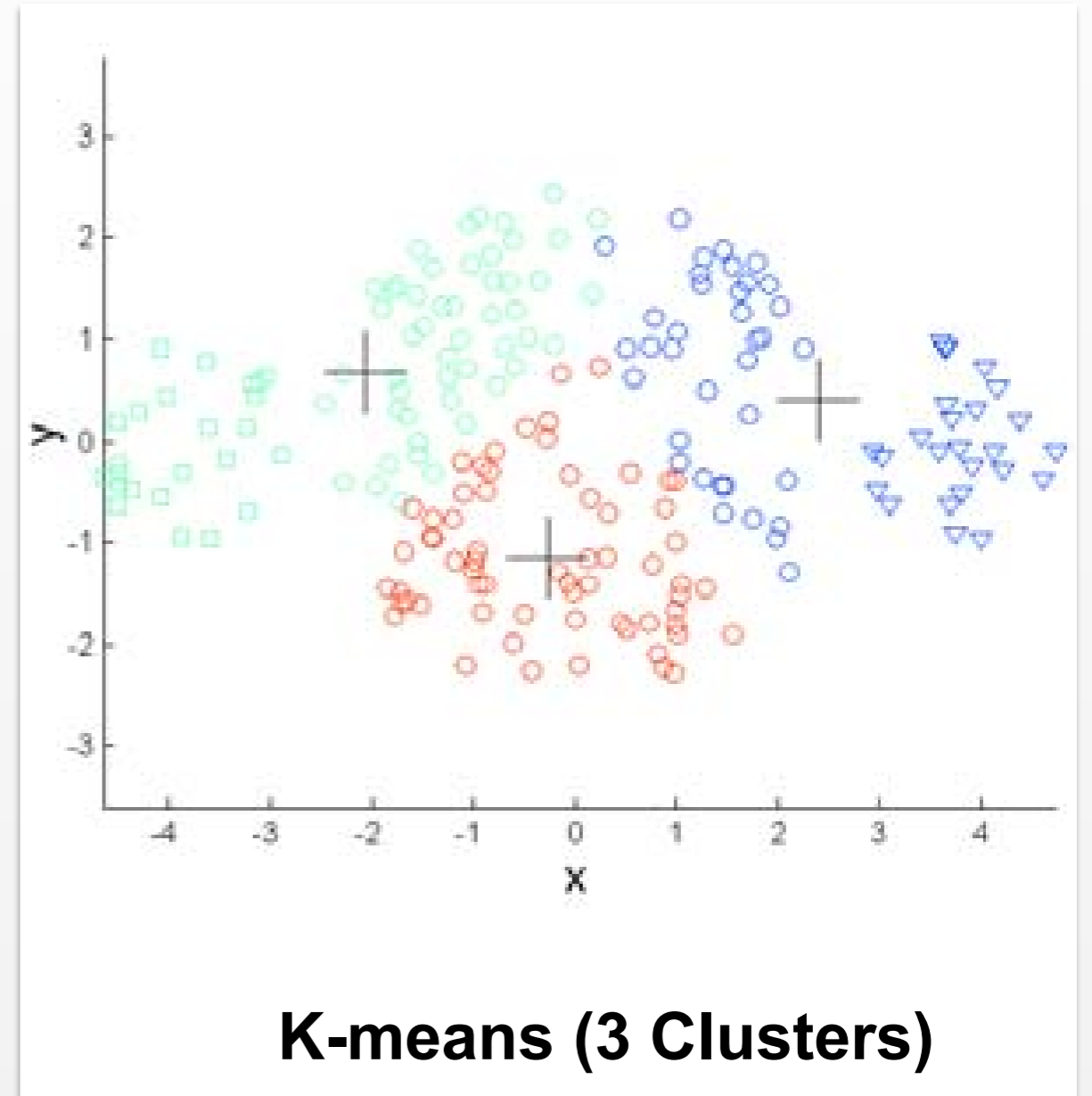
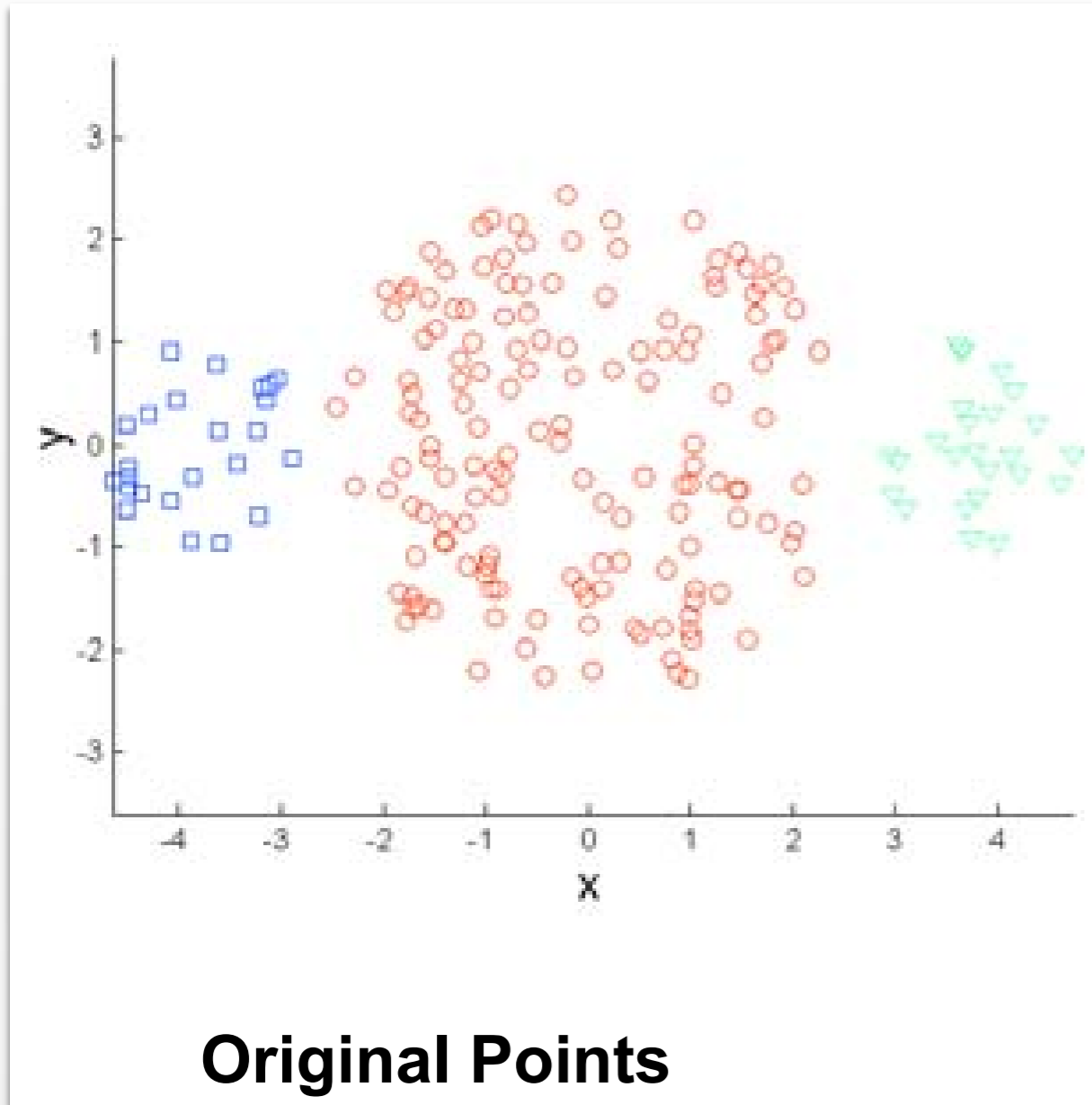


“Elbow finding” (a.k.a. “knee finding”)
Set K to value just above “abrupt” increase

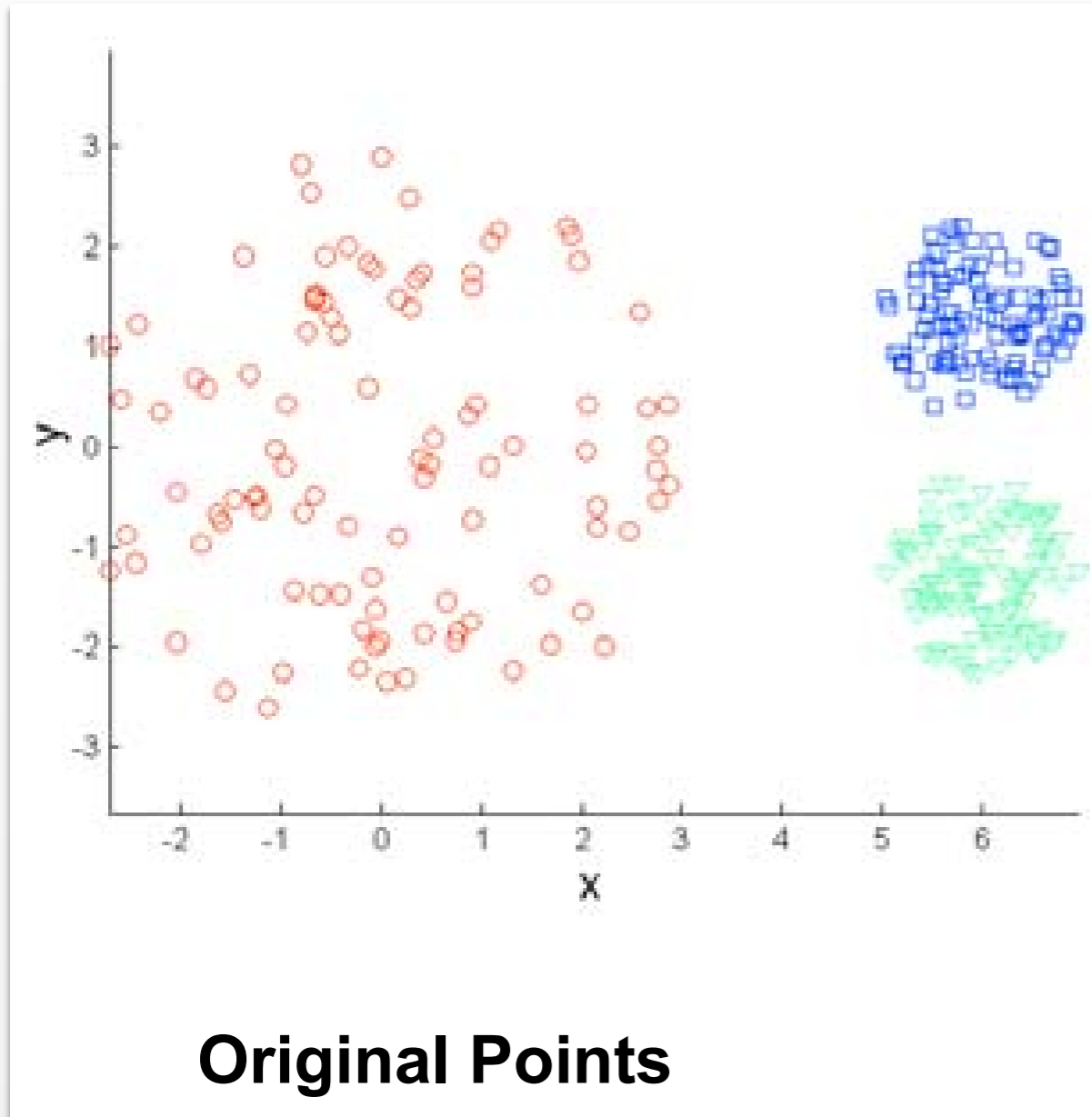
K-means Limitations: Differing Sizes



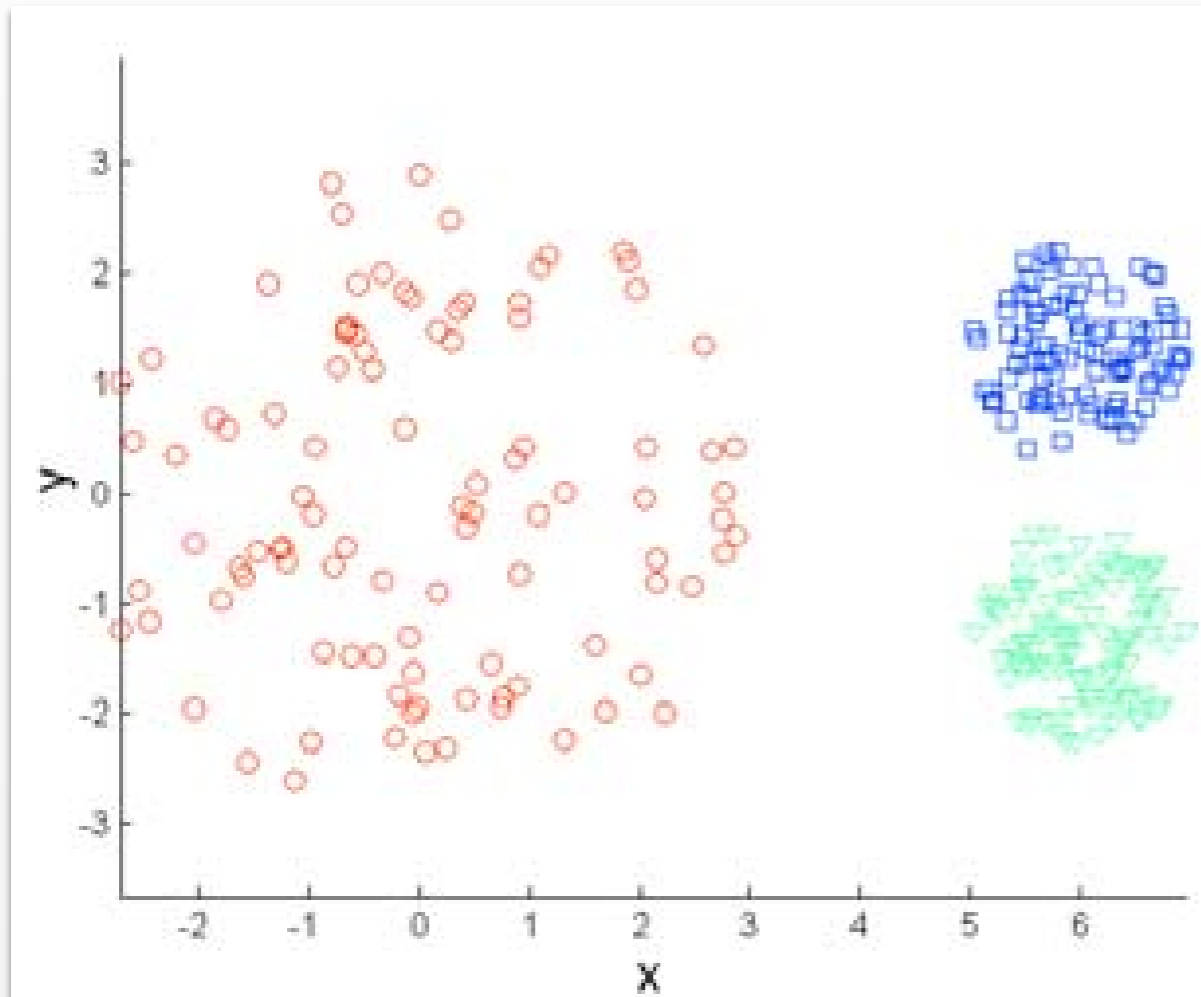
K-means Limitations: Differing Sizes



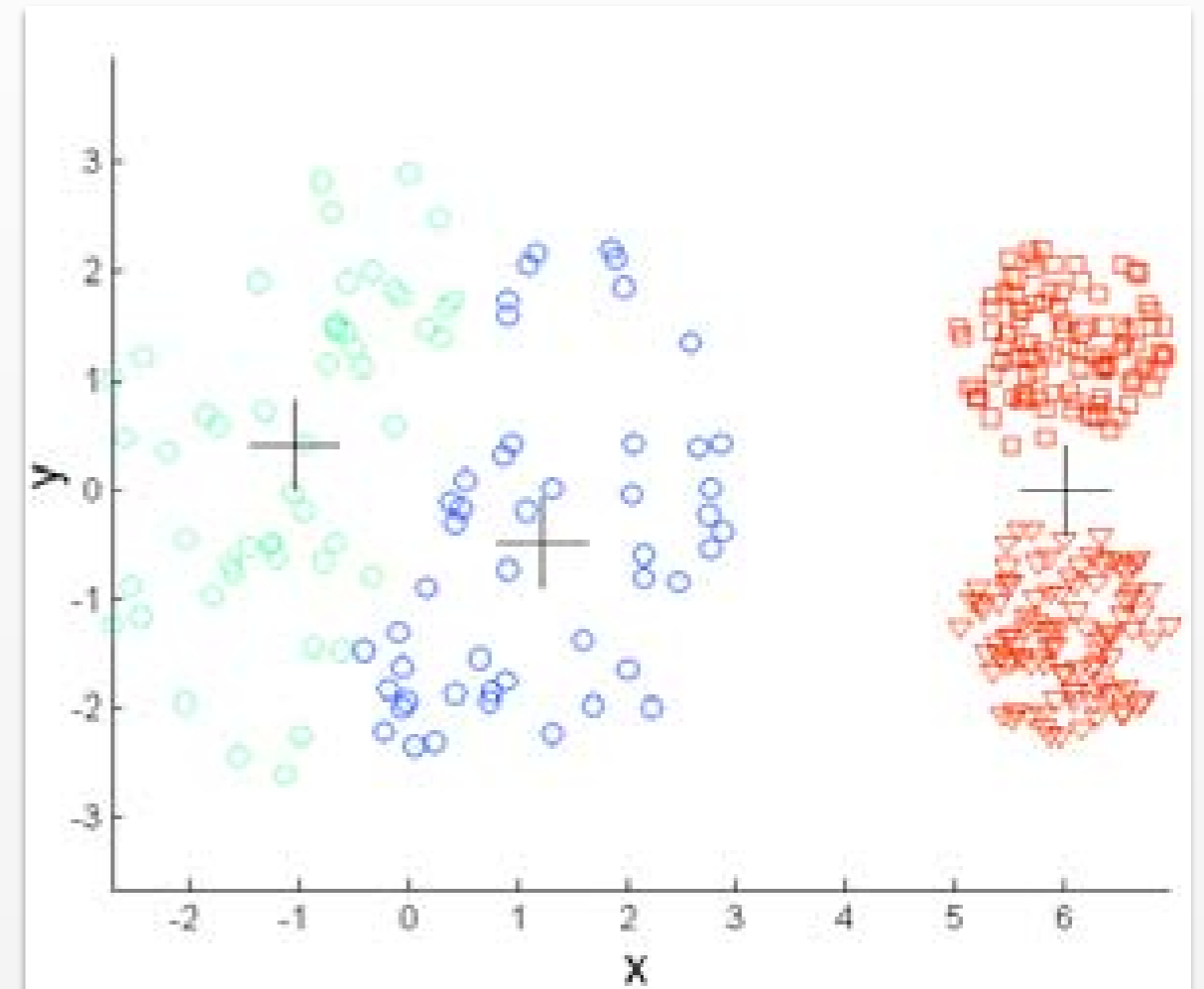
K-means Limitations: Differing Densities



K-means Limitations: Differing Densities

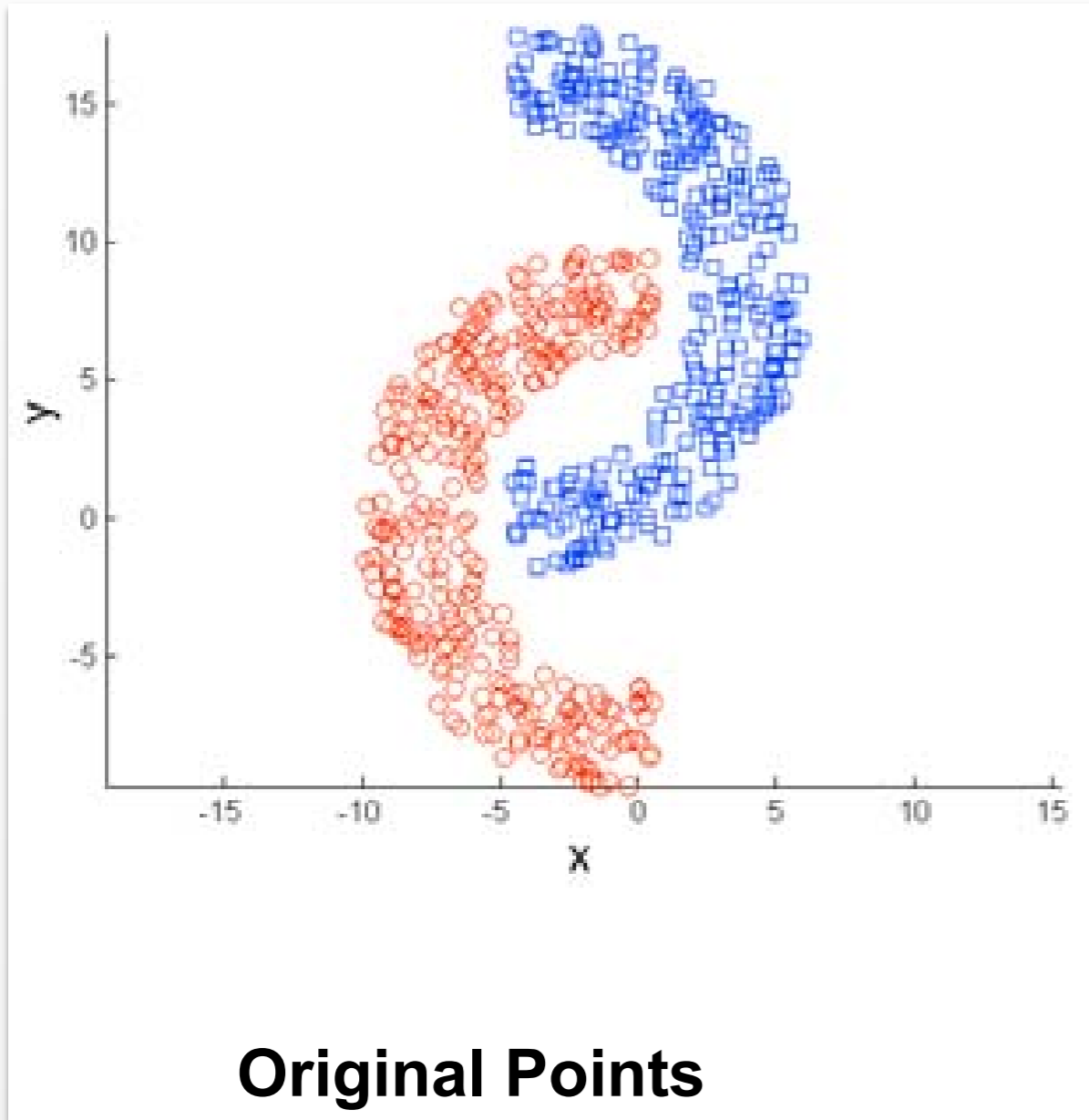


Original Points

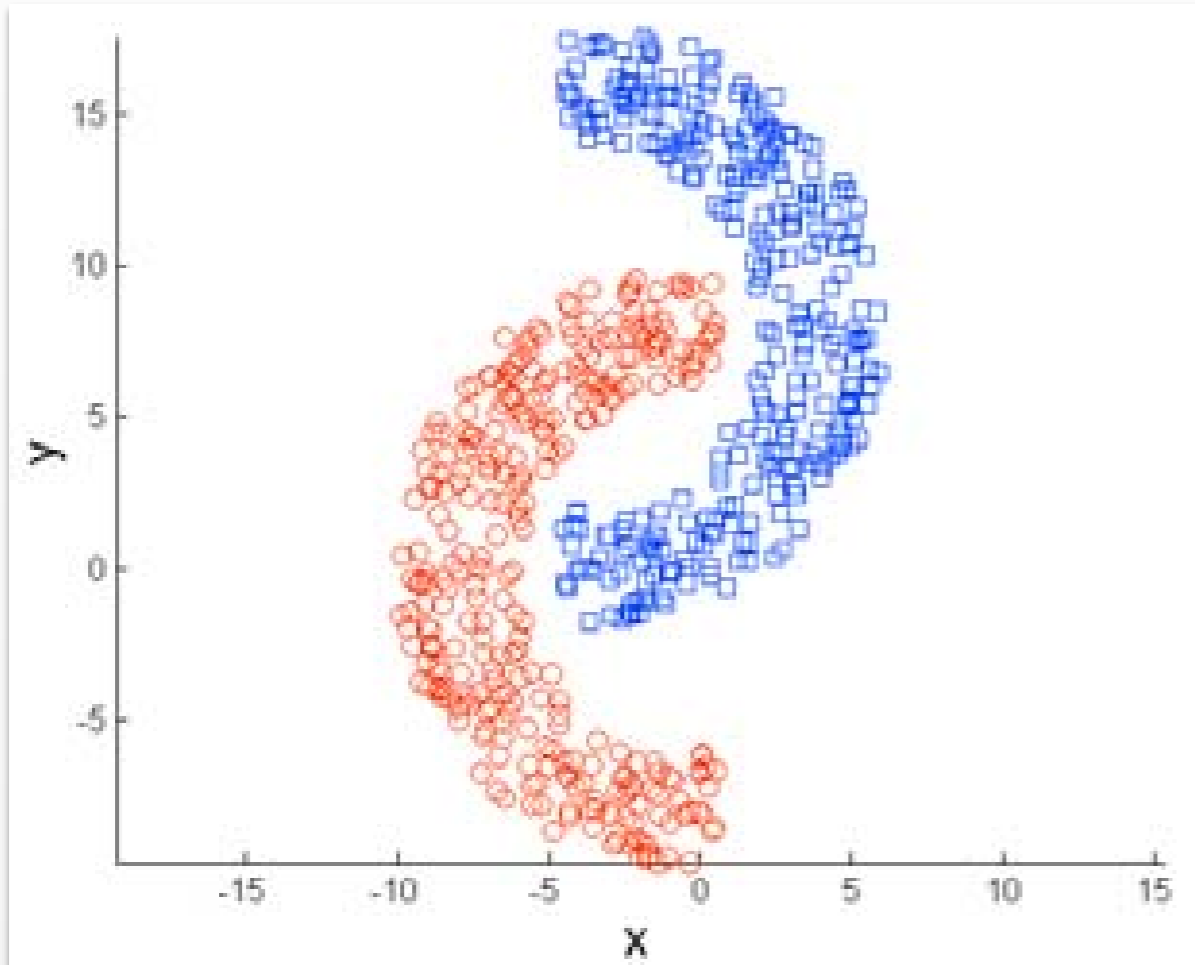


K-means (3 Clusters)

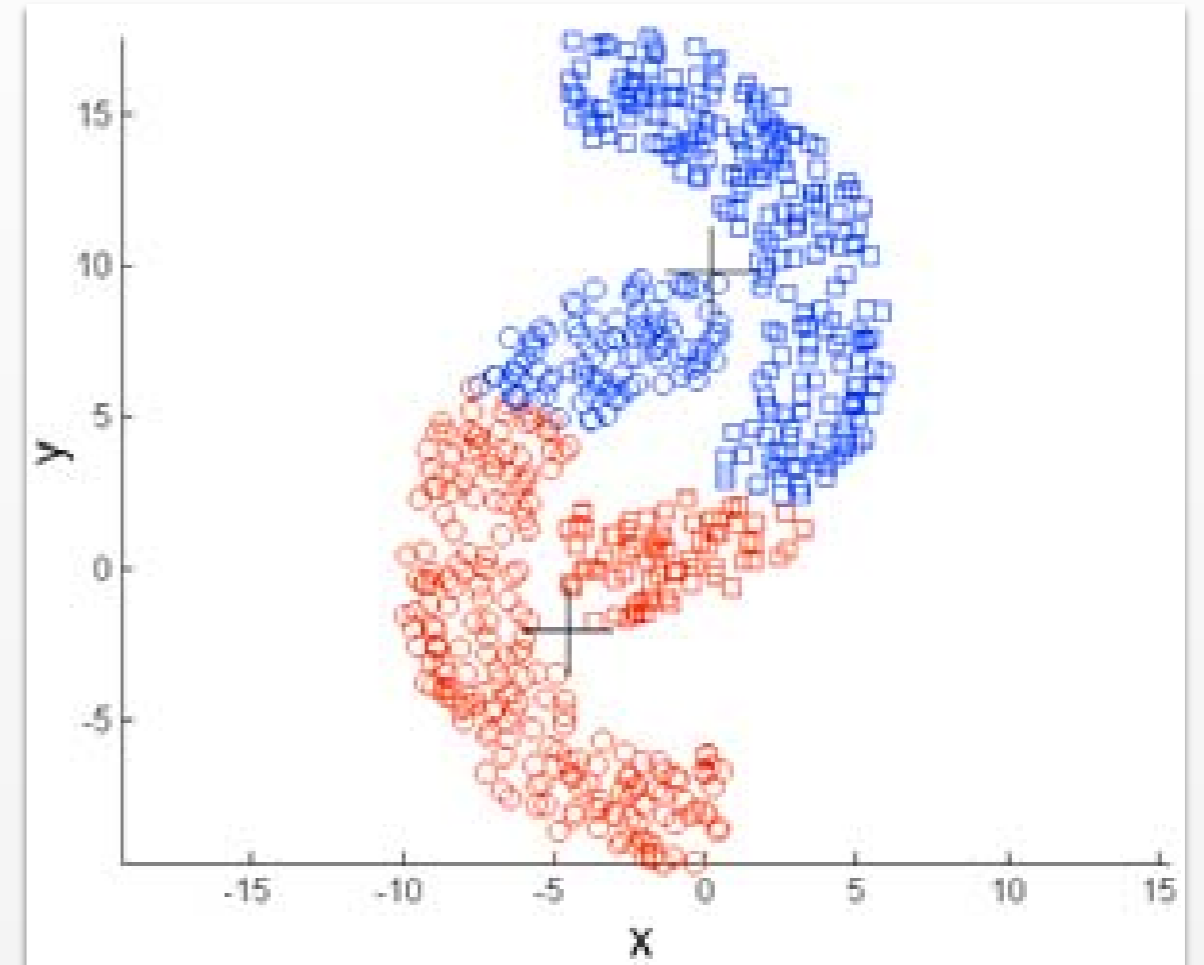
K-means Limitations: Non-globular Shapes



K-means Limitations: Non-globular Shapes

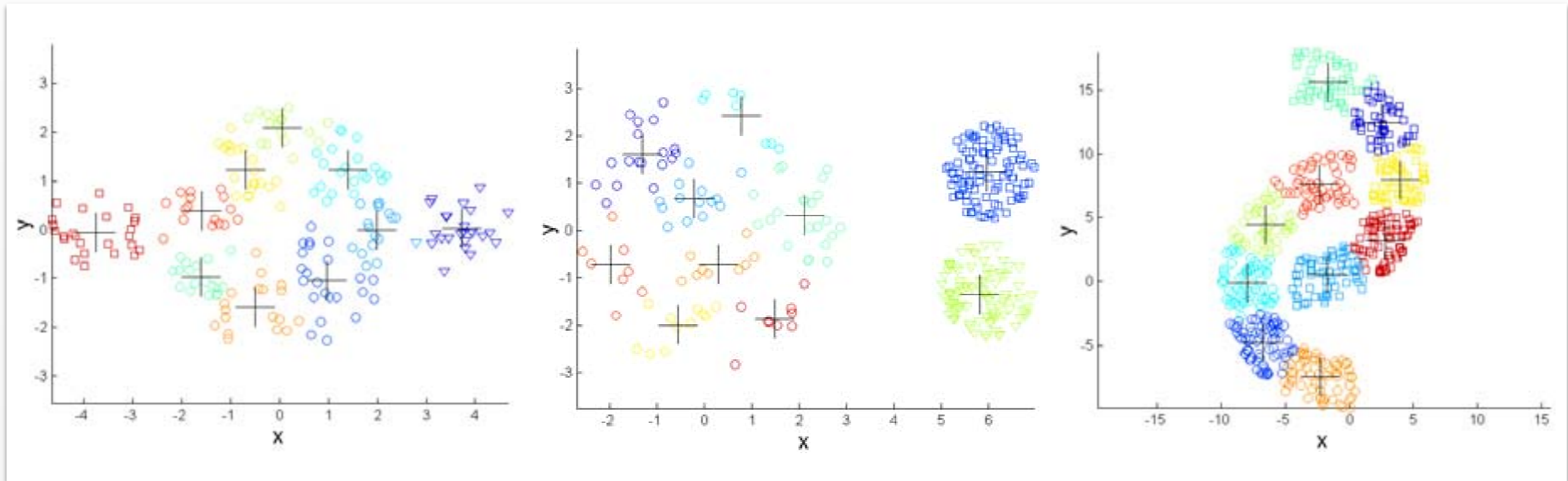


Original Points



K-means (2 Clusters)

Overcoming K-means Limitations

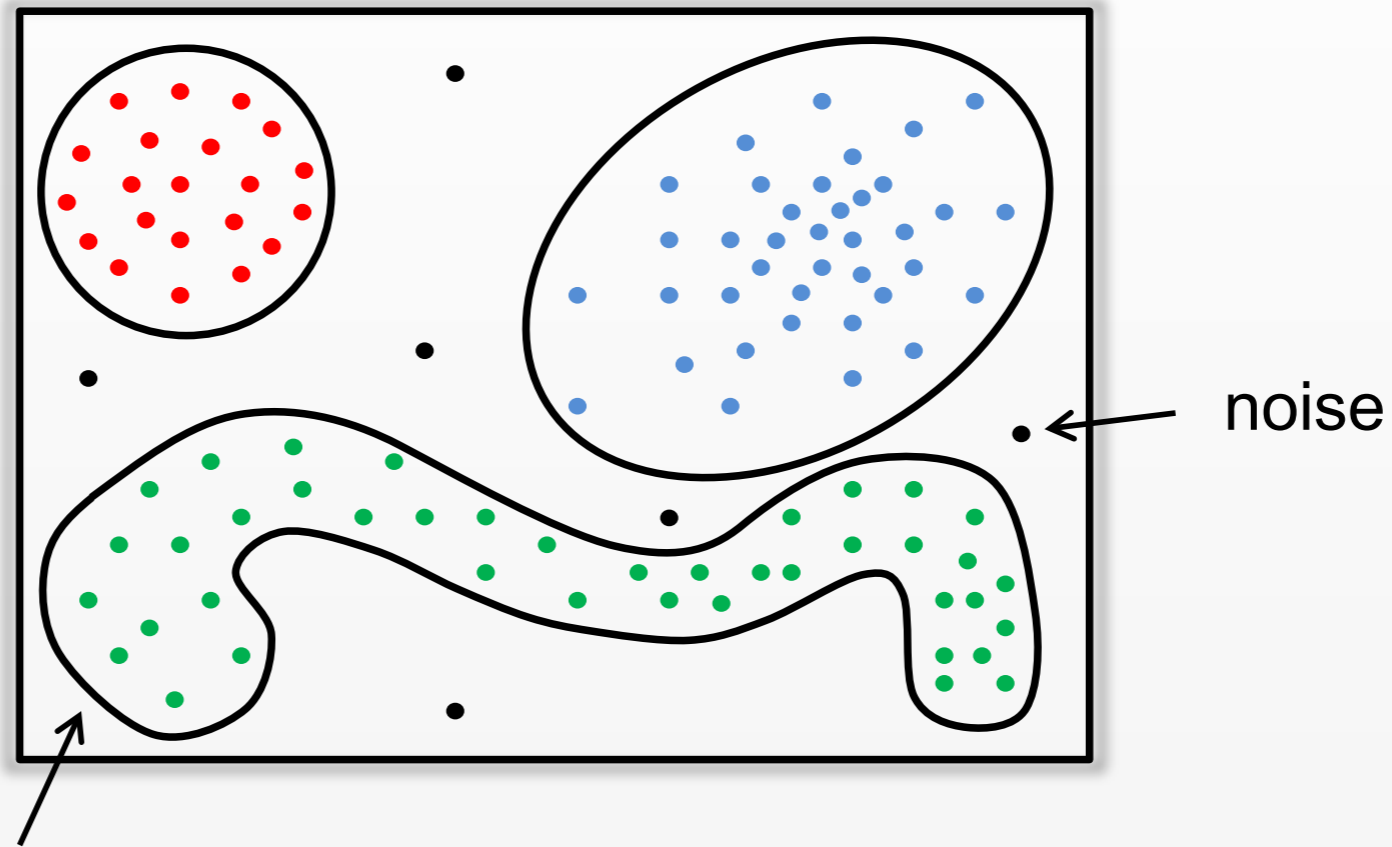


Intuition: “Combine” smaller clusters into larger clusters

- *One Solution:* Hierarchical Clustering
- *Another Solution:* Density-based Clustering

Density-based Clustering

DBSCAN



arbitrarily shaped clusters

[\[PDF\] A density-based algorithm for discovering clusters in large spatial databases with noise.](#)

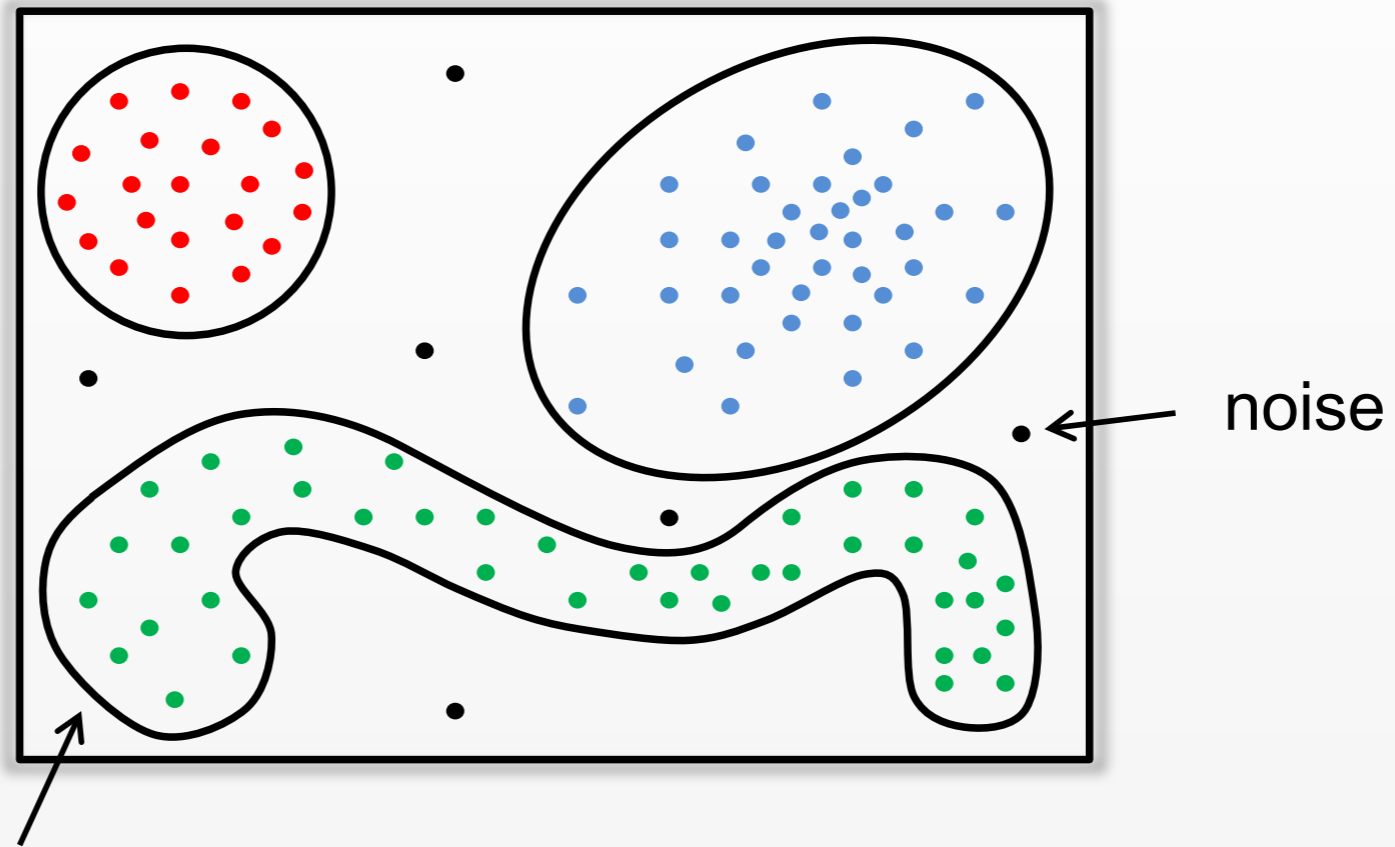
[M Ester, HP Kriegel, J Sander, X Xu - Kdd, 1996 - aaii.org](#)

Abstract Clustering algorithms are attractive for the task of class identification in spatial databases. However, the application to large spatial databases rises the following requirements for clustering algorithms: minimal requirements of domain knowledge to ...

[Cited by 8901](#) [Related articles](#) [All 70 versions](#) [Cite](#) [Save](#) [More](#)

(one of the most-cited clustering methods)

DBSCAN



arbitrarily shaped clusters

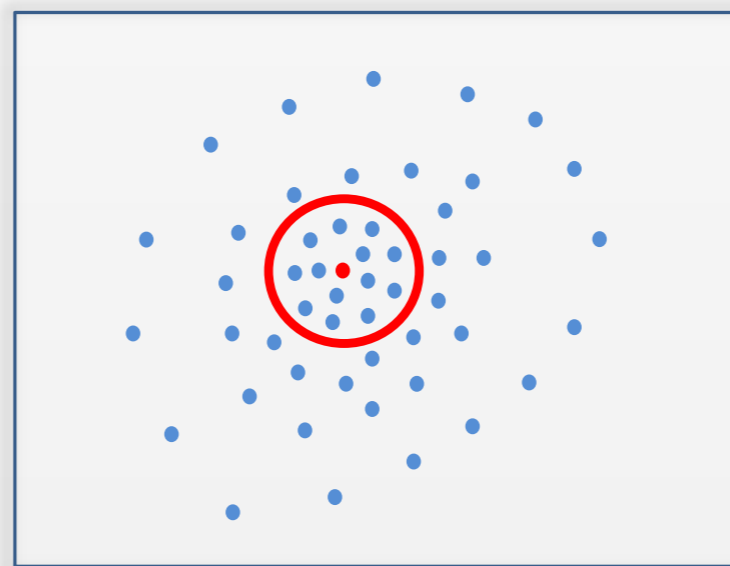
Intuition

- *A cluster* is a region of *high* density
- *Noise* points lie in regions of *low* density

Defining “High Density”

Naïve approach

For each point in a cluster there are **at least** a **minimum number (MinPts)** of points in an **Eps-neighborhood** of that point.

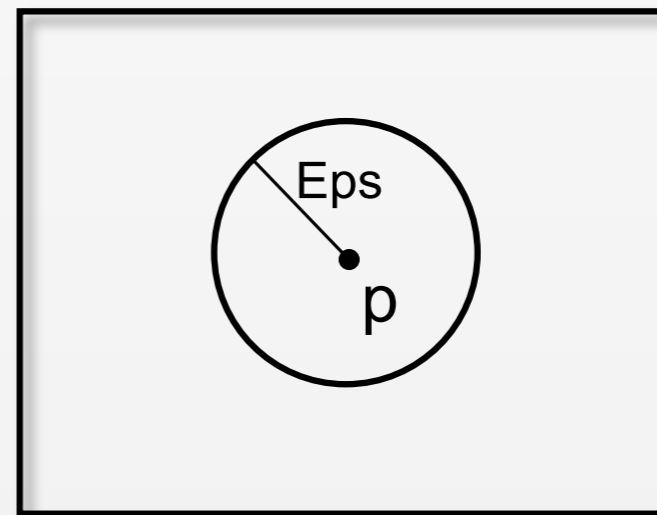


cluster

Defining “High Density”

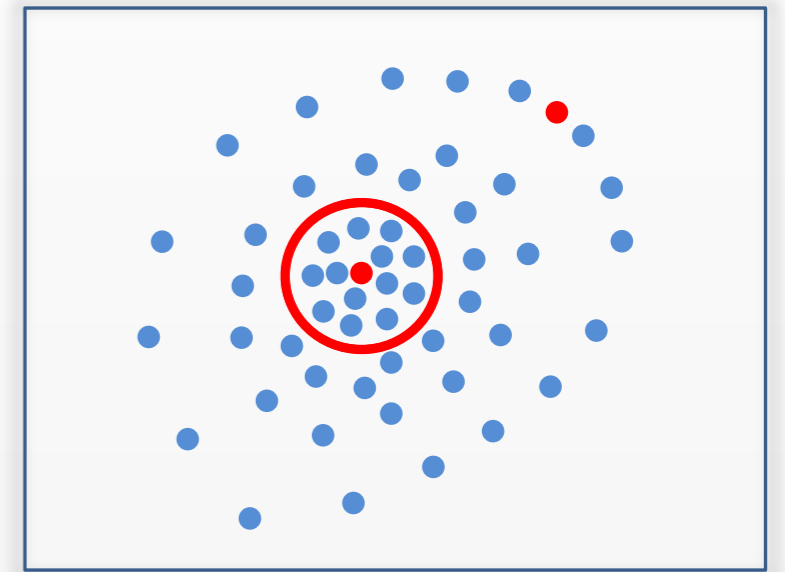
Eps-neighborhood of a point p

$$N_{\text{Eps}}(p) = \{ q \in D \mid \text{dist}(p, q) \leq \text{Eps} \}$$



Defining “High Density”

- In each cluster there are two kinds of points:
 - points inside the cluster (core points)
 - points on the border (border points)



cluster

An **Eps-neighborhood of a border point** contains **significantly less points than** an **Eps-neighborhood of a core point**.

Defining “High Density”

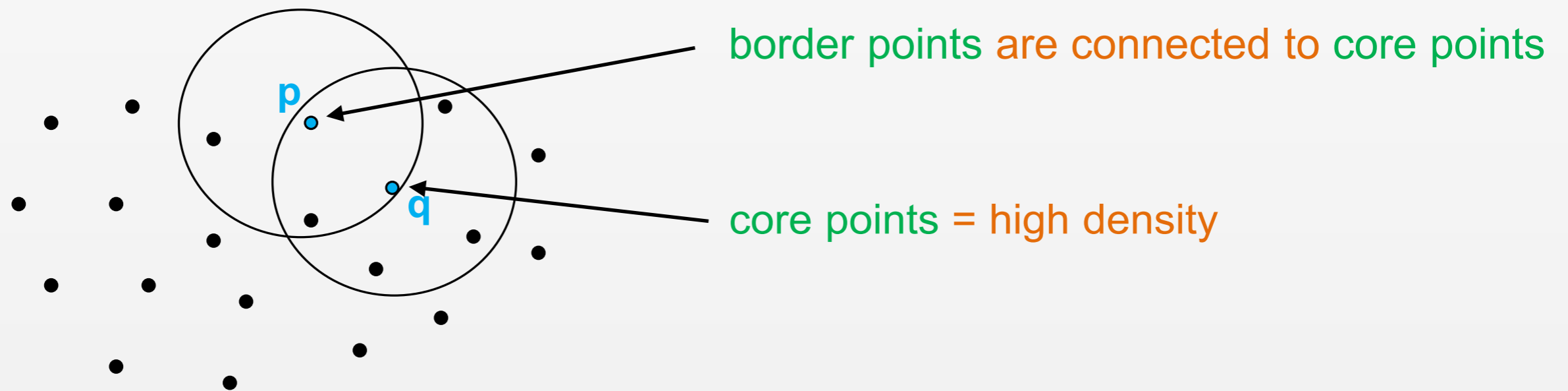
Better notion of cluster

For every point p in a cluster C there is a point $q \in C$, so that

(1) p is inside of the Eps-neighborhood of q

and

(2) $N_{Eps}(q)$ contains at least $MinPts$ points.

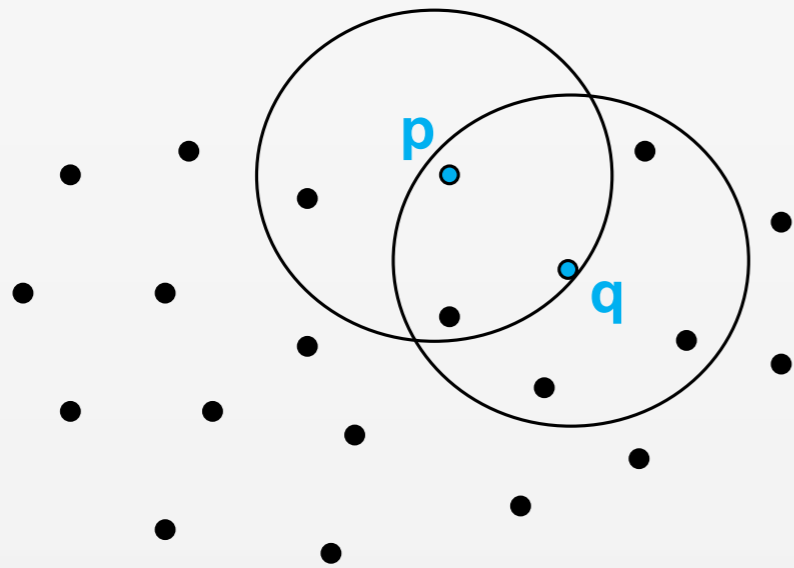


Density Reachability

Definition

A point p is **directly density-reachable** from a point q with regard to the parameters Eps and $MinPts$, if

- 1) $p \in N_{Eps}(q)$ (reachability)
- 2) $|N_{Eps}(q)| \geq MinPts$ (core point condition)



Parameter: $MinPts = 5$

p directly density reachable from q

$$p \in N_{Eps}(q)$$

$$|N_{Eps}(q)| = 6 \geq 5 = MinPts \quad (\text{core point condition})$$

q **not** directly density reachable from p

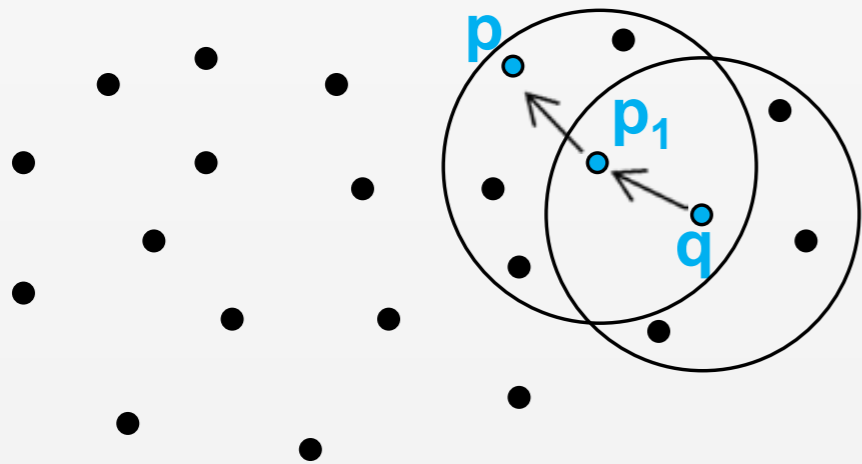
$$|N_{Eps}(p)| = 4 < 5 = MinPts \quad (\text{core point condition})$$

Note: This is an asymmetric relationship

Density Reachability

Definition

A point p is **density-reachable** from a point q with regard to the parameters Eps and MinPts if there is a **chain of points** p_1, p_2, \dots, p_s with $p_1 = q$ and $p_s = p$ such that p_{i+1} is **directly density-reachable** from p_i for all $1 < i < s-1$.



$$\text{MinPts} = 5$$

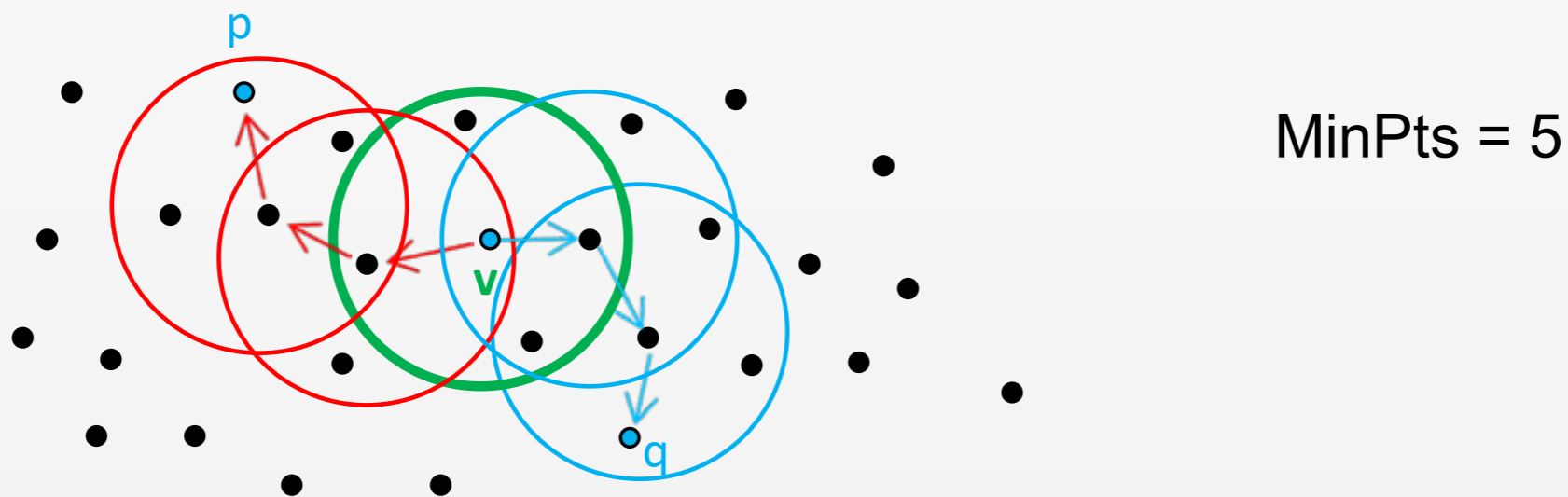
$$|N_{\text{Eps}}(q)| = 5 = \text{MinPts} \quad (\text{core point condition})$$

$$|N_{\text{Eps}}(p_1)| = 6 \geq 5 = \text{MinPts} \quad (\text{core point condition})$$

Density Connectivity

Definition (density-connected)

A point p is **density-connected** to a point q with regard to the parameters Eps and $MinPts$ if there is a point v such that both p and q are density-reachable from v .



Note: This is a symmetric relationship

Definition of a Cluster

A **cluster** with regard to the parameters ϵ and MinPts is a non-empty subset C of the database D with

1) For all $p, q \in D$:

(Maximality)

If $p \in C$ and q is density-reachable from p with regard to the parameters ϵ and MinPts , then $q \in C$.

2) For all $p, q \in C$:

(Connectivity)

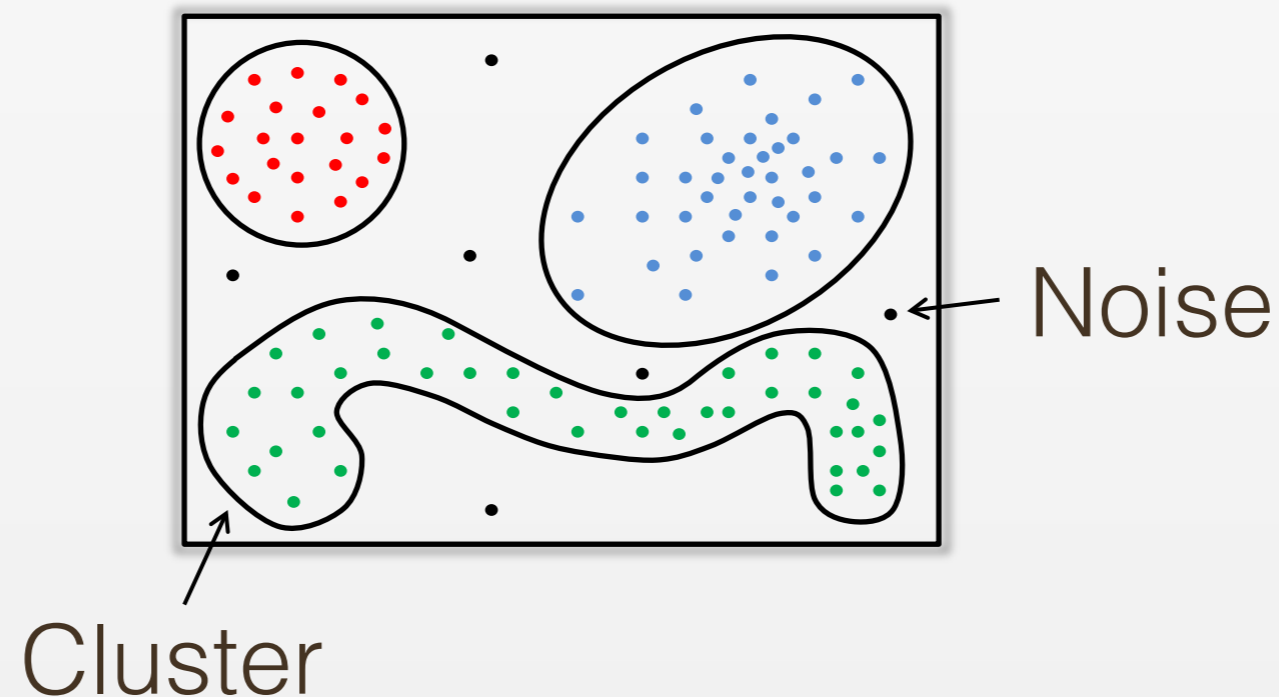
The point p is density-connected to q with regard to the parameters ϵ and MinPts .

Definition of Noise

Let C_1, \dots, C_k be the clusters of the database D with regard to the parameters Eps_i and $MinPts_i$ ($i=1, \dots, k$).

The set of points in the database D not belonging to any cluster C_1, \dots, C_k is called **noise**:

$$\text{Noise} = \{ p \in D \mid p \notin C_i \text{ for all } i = 1, \dots, k \}$$



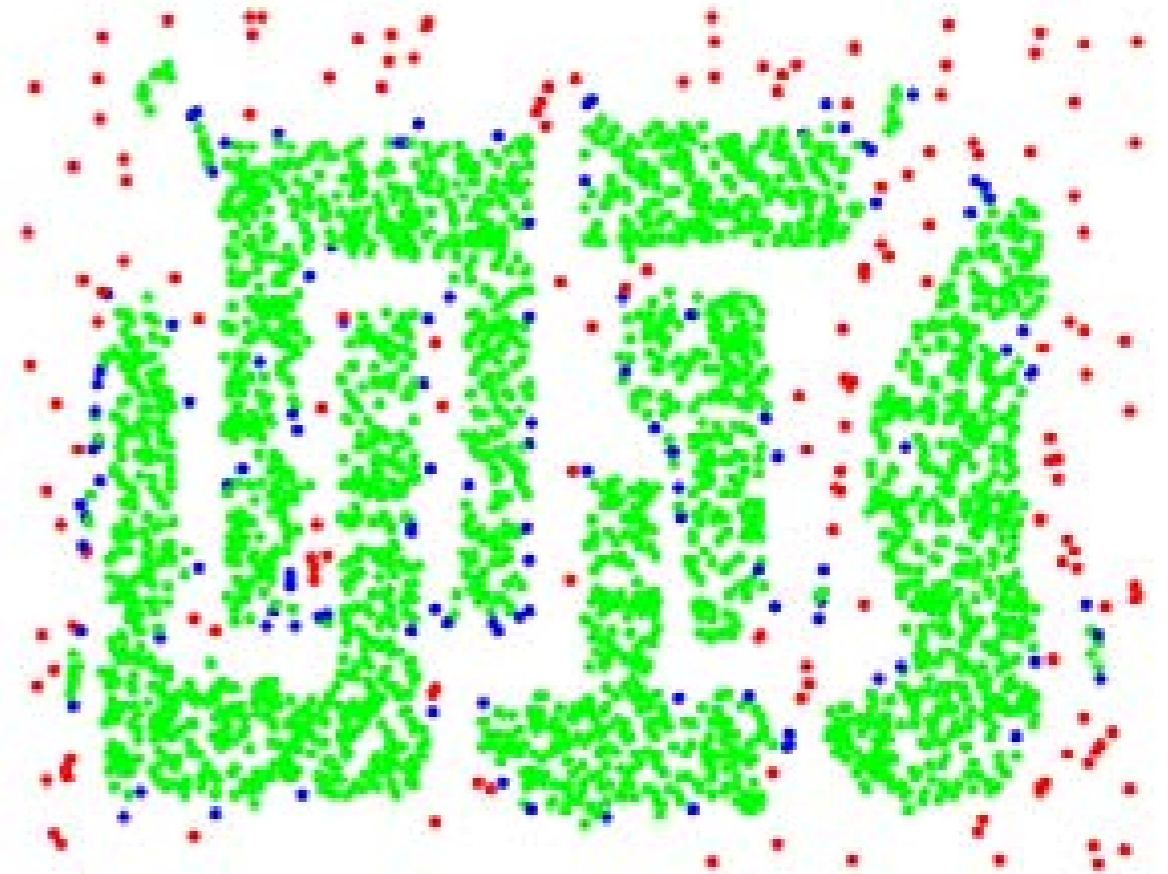
DBSCAN Algorithm

- (1) Start with an **arbitrary point p** from the database and **retrieve all points density-reachable from p** with regard to Eps and $MinPts$.
- (2) If p is a **core point**, the procedure yields a **cluster** with regard to Eps and $MinPts$ and all points in the **cluster** are classified.
- (3) If p is a **border point**, no points are density-reachable from p and DBSCAN visits the **next unclassified point in the database**.

DBSCAN Algorithm



Original Points



Point types: **core**,
border and **noise**

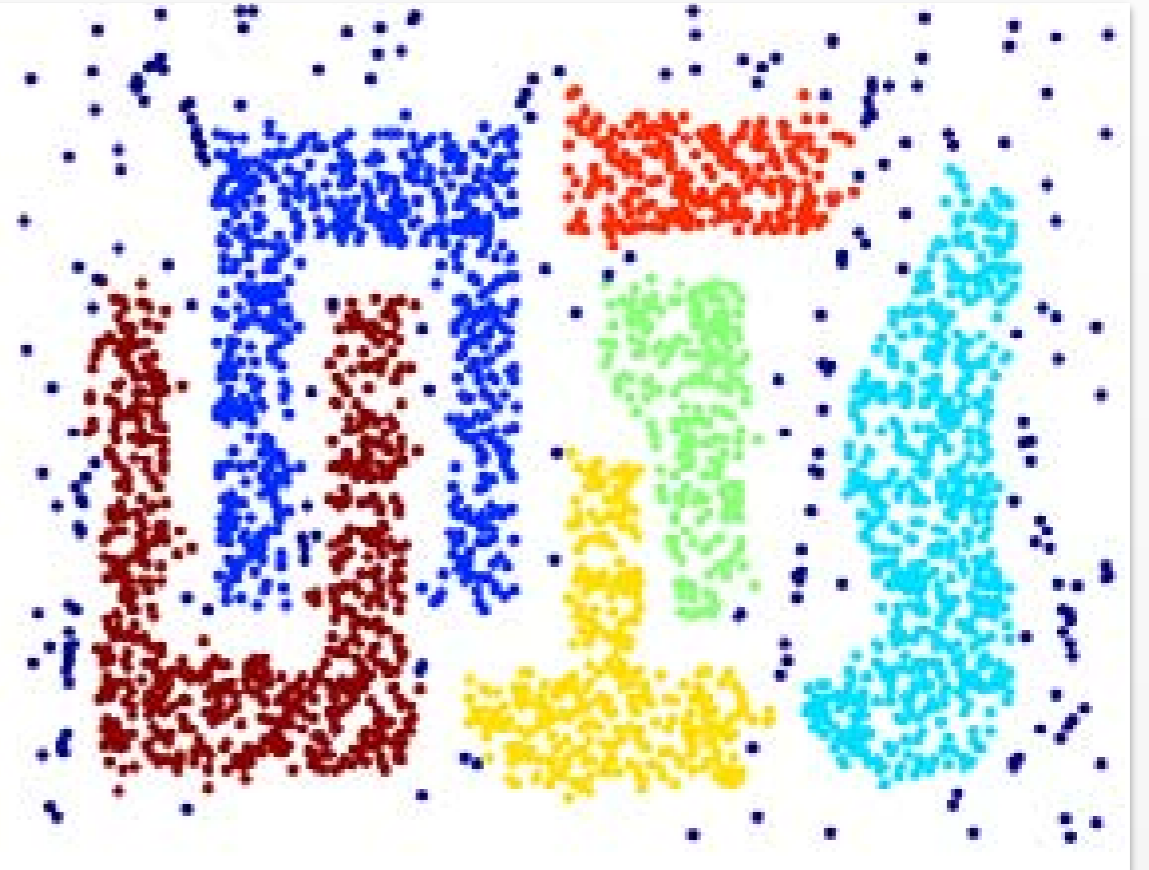
DBSCAN Complexity

- *Time complexity:* $O(N^2)$ if done naively, $O(N \log N)$ when using a spatial index (*works in relatively low dimensions*)
- *Space complexity:* $O(N)$

DBSCAN strengths



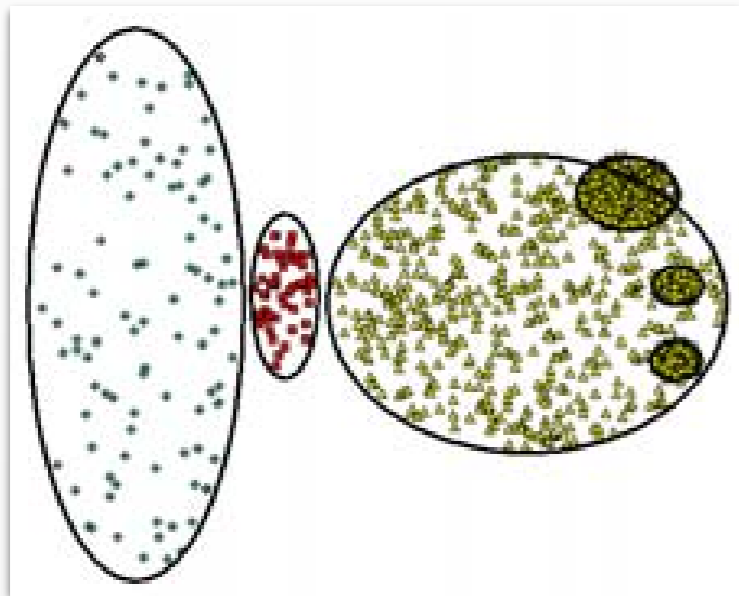
Original Points



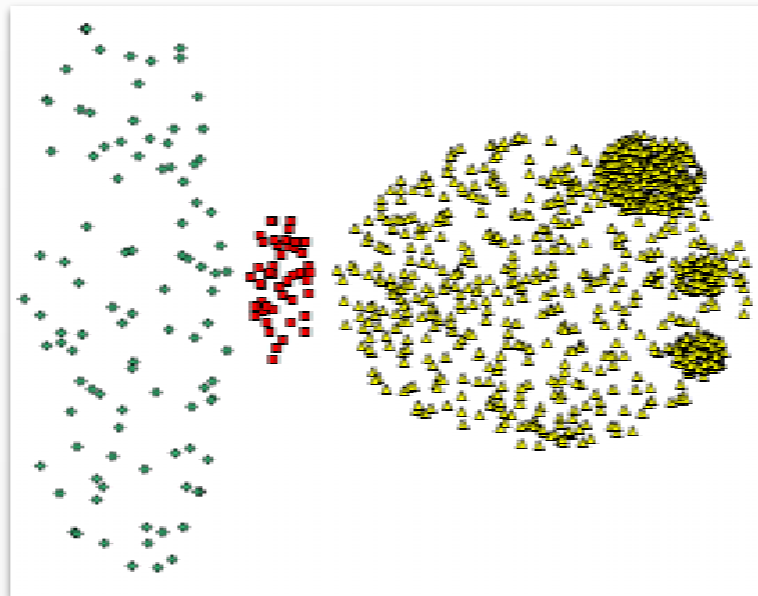
Clusters

- + Resistant to noise
- + Can handle arbitrary shapes

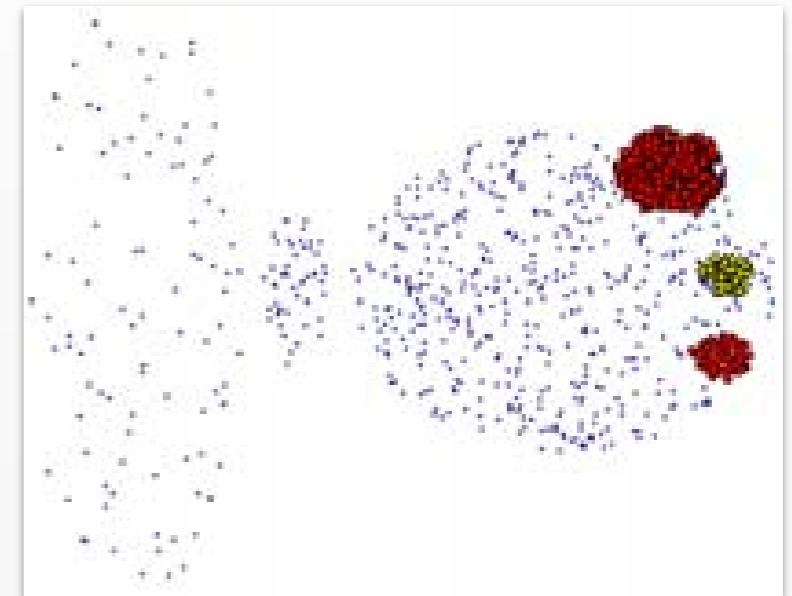
DBSCAN Weaknesses



Ground Truth



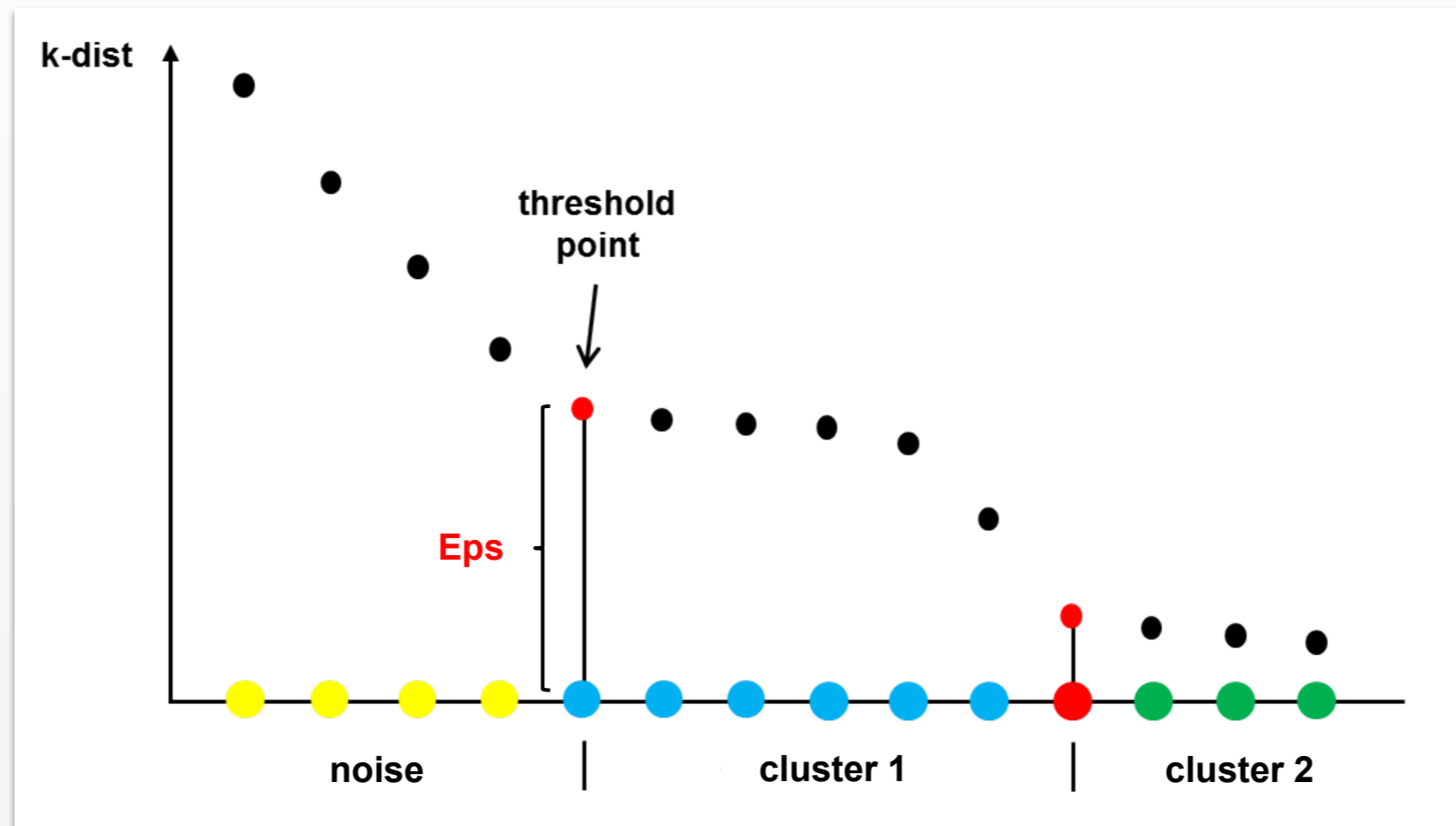
MinPts = 4, Eps=9.92



MinPts = 4, Eps=9.75

- Varying densities
- High dimensional data
- Overlapping clusters

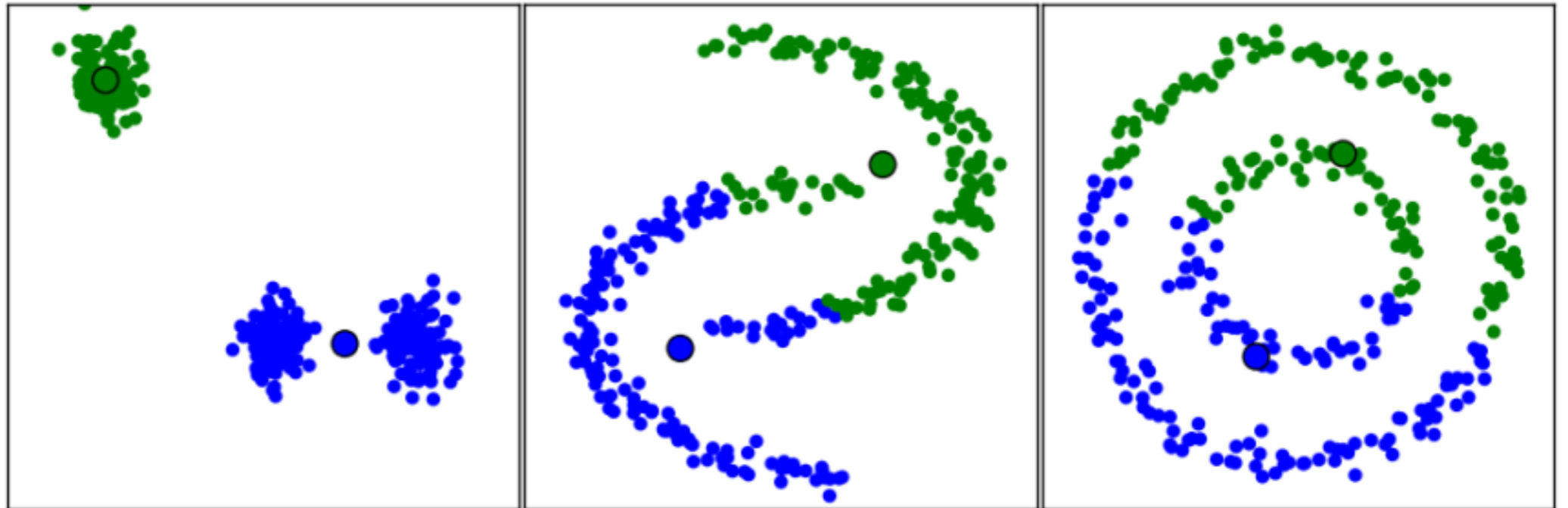
Determining EPS and MINPTS



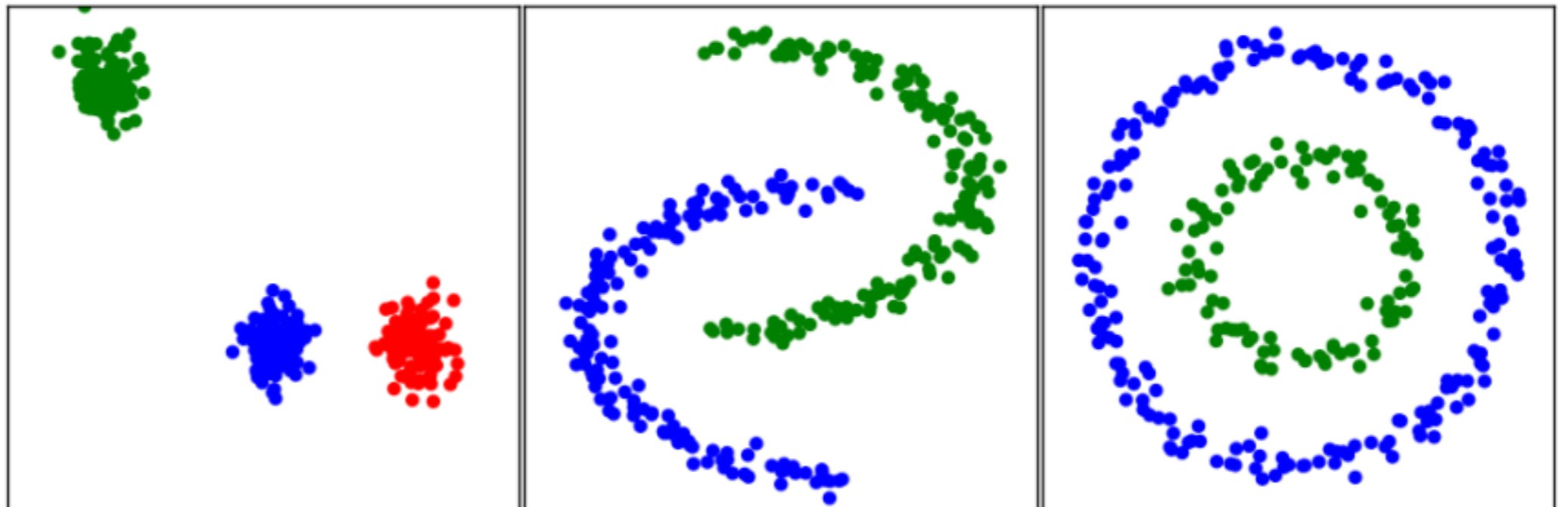
- Calculate distance of k -th nearest neighbor for each point
- Plot in ascending / descending order
- Set EPS to max distance before “jump”

K-means vs DBSCAN

K-means



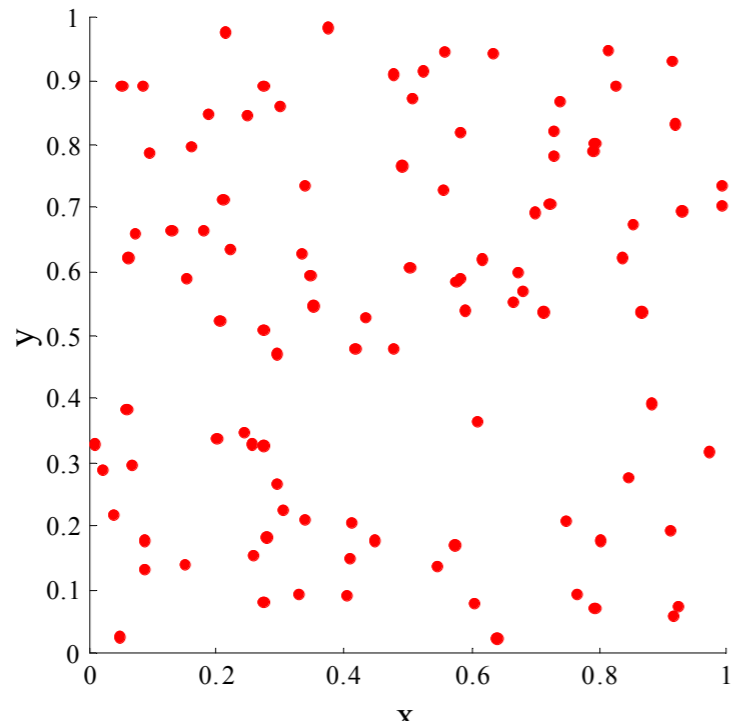
DBSCAN



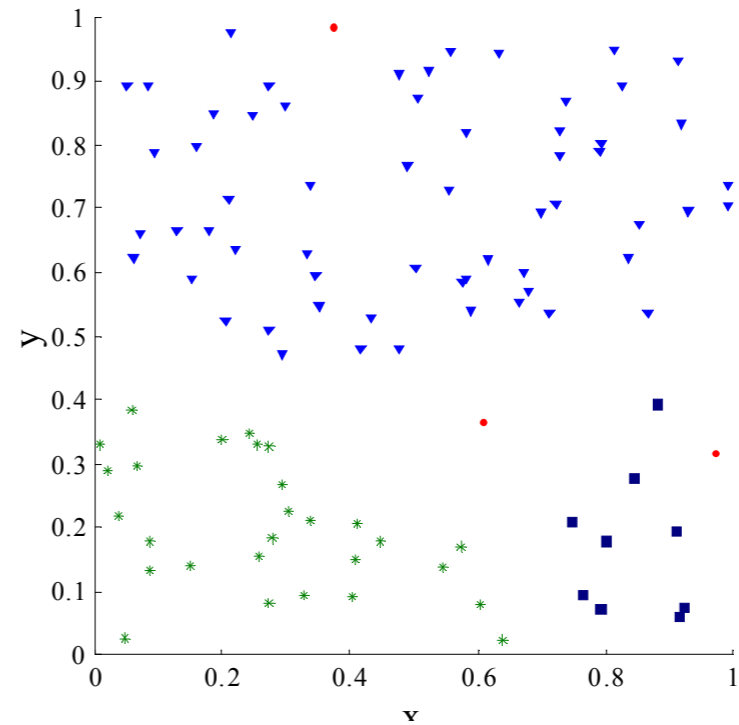
Evaluation of Clustering

Clusters in Random Data

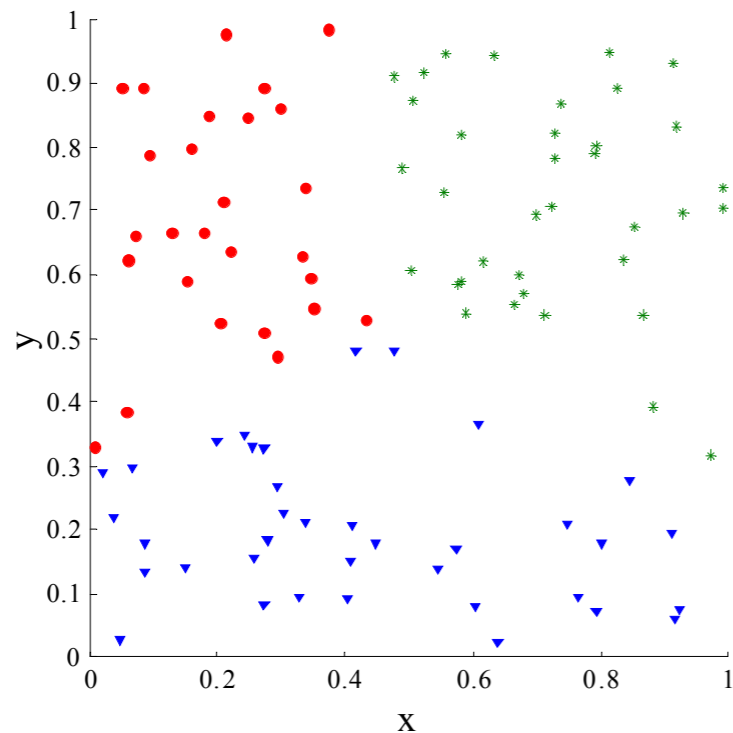
**Random
Points**



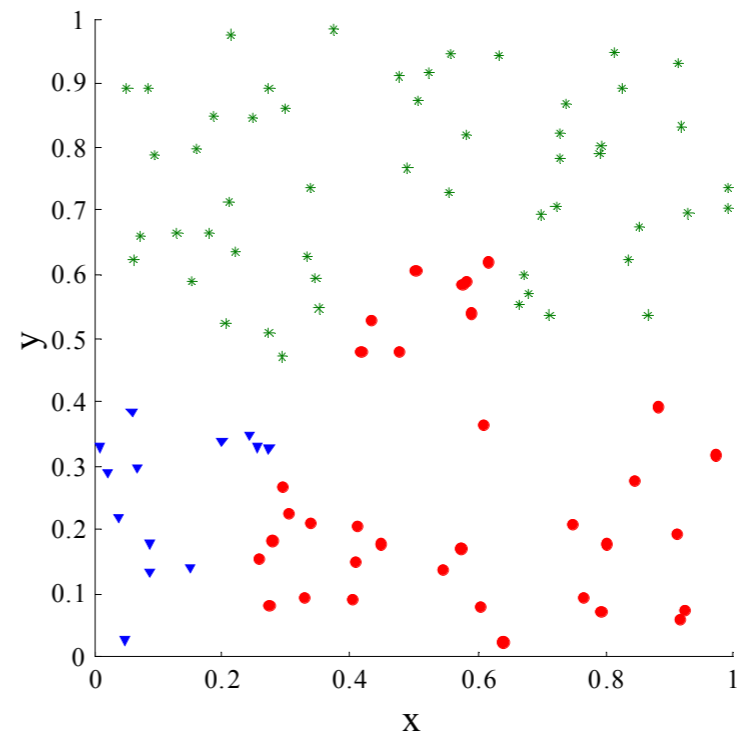
DBSCAN



K-means



**Complete
Link**



Clustering Criteria

- ***External Quality Criteria***
 - Precision-Recall Measure
 - Mutual Information
- ***Internal Quality Criteria***

Measure compactness of clusters

 - Sum of Squared Error (SSE)
 - Scatter Criteria

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Uncorrelated Variables

$$p(a, b) = p(a)p(b)$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Uncorrelated Variables

$$p(a, b) = p(a)p(b)$$

$$I(A; B) = \sum_{a \in A, b \in B} p(a)p(b) \log \frac{p(a)p(b)}{p(a)p(b)} = 0$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Perfectly Correlated Variables

$$p(A = a, B = b) = \delta(a, b)p(B = b)$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Perfectly Correlated Variables

$$p(A = a, B = b) = \delta(a, b)p(B = b)$$

$$I(A; B) = \sum_{a \in A, B \in B} p(A = a, B = b) \log \frac{p(A = a, B = b)}{p(A = a)p(B = b)}$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Perfectly Correlated Variables

$$p(A = a, B = b) = \delta(a, b)p(B = b)$$

$$I(A; B) = \sum_{b \in B} p(B = b) \log \frac{p(B = b)}{p(A = b)p(B = b)}$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Perfectly Correlated Variables

$$P(A = k) = \sum_{l \in B} \delta(k, l) p(B = l) = p(B = k)$$

$$I(A; B) = \sum_{b \in B} p(B = b) \log \frac{p(B = b)}{p(A = b)p(B = b)}$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Perfectly Correlated Variables

$$P(A = k) = \sum_{l \in B} \delta(k, l) p(B = l) = p(B = k)$$

$$I(A; B) = \sum_{b \in B} p(B = b) \log \frac{p(B = b)}{p(B = b)p(B = b)}$$

Mutual Information (External)

$$I(A, B) = \sum_{a \in A, b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

Perfectly Correlated Variables

$$P(A = k) = \sum_{l \in B} \delta(k, l) p(B = l) = p(B = k)$$

$$I(A; B) = - \sum_{b \in B} p(b) \log p(b) = H(B)$$

Mutual Information (External)

$$I(Y; Z) = \sum_{y,z} p(y,z) \log \frac{p(y,z)}{p(y)p(z)}$$

y_n : True class label for example n

z_n : Clustering label for example n

Mutual Information (External)

$$I(Y; Z) = \sum_{y,z} p(y,z) \log \frac{p(y,z)}{p(y)p(z)}$$

y_n : True class label for example n

z_n : Clustering label for example n

$$p(Y = k) = \frac{1}{N} \sum_n I(y_n = k) \quad p(Z = l) = \frac{1}{N} \sum_n I(z_n = l)$$

$$p(Y = k, Z = l) = \frac{1}{N} \sum_n I(y_n = k \wedge z_n = l)$$

Mutual Information (External)

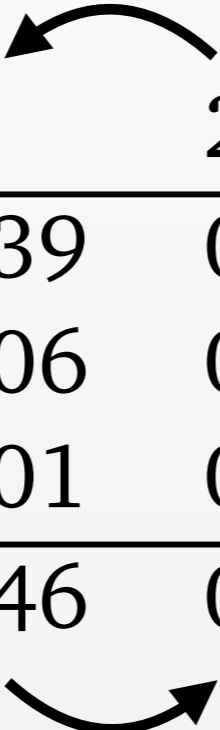
$$I(Y; Z) = \sum_{y,z} p(y,z) \log \frac{p(y,z)}{p(y)p(z)}$$

$p(y, z)$	1	2	3	$p(y)$
cat	0.39	0.08	0.02	0.49
dog	0.06	0.31	0.01	0.38
parrot	0.01	0.01	0.11	0.13
$p(z)$	0.46	0.40	0.04	

Mutual Information (External)

$$I(Y; Z) = \sum_{y,z} p(y,z) \log \frac{p(y,z)}{p(y)p(z)}$$

$p(y,z)$	1	2	3	$p(y)$
cat	0.39	0.08	0.02	0.49
dog	0.06	0.31	0.01	0.38
parrot	0.01	0.01	0.11	0.13
$p(z)$	0.46	0.40	0.04	



What happens to $I(Y; Z)$ if we swap cluster labels?

Mutual Information (External)

$$I(Y; Z) = \sum_{y,z} p(y,z) \log \frac{p(y,z)}{p(y)p(z)}$$

$p(y,z)$	1	2	3	$p(y)$
cat	0.08	0.39	0.02	0.49
dog	0.31	0.06	0.01	0.38
parrot	0.01	0.01	0.11	0.13
$p(z)$	0.40	0.46	0.04	

What happens to $I(Y;Z)$ if we swap cluster labels?

Mutual Information (External)

$$I(Y; Z) = \sum_{y,z} p(y,z) \log \frac{p(y,z)}{p(y)p(z)}$$

$p(y,z)$	1	2	3	$p(y)$
cat	0.08	0.39	0.02	0.49
dog	0.31	0.06	0.01	0.38
parrot	0.01	0.01	0.11	0.13
$p(z)$	0.40	0.46	0.04	

Mutual Information is ***invariant*** under label permutations

Scatter Criteria (Internal)

Let $\mathbf{x} = (x_1, \dots, x_d)^T$
 C_1, \dots, C_K be a clustering of $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$

Define

- Size of each cluster:

$$N_i = |C_i| \quad i = 1, 2, \dots, K$$

- Mean for each cluster:

$$\mu_i = \frac{1}{N_i} \sum_{\mathbf{x} \in C_i} \mathbf{x} \quad i = 1, 2, \dots, K$$

- Total mean :

$$\mu = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \quad \text{OR} \quad \mu = \frac{1}{N} \sum_{i=1}^K N_i \mu_i$$

Scatter Criteria (Internal)

- Scatter matrix for the i^{th} cluster:

$$S_i = \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^T \quad (\text{outer product})$$

- Within cluster scatter matrix :

$$S_W = \sum_{i=1}^K S_i$$

- Between cluster scatter matrix :

$$S_B = \sum_{i=1}^K N_i(\mu_i - \mu)(\mu_i - \mu)^T \quad (\text{outer product})$$

Scatter Criteria (Internal)

- The trace criteria: sum of the diagonal elements of a matrix
- A good partition of the data should have:
 - Low $tr(S_W)$: similar to minimizing SSE
 - High $tr(S_B)$
 - High $\frac{tr(S_B)}{tr(S_W)}$