Aprendizado de Máquina e Reconhecimento de Padrões

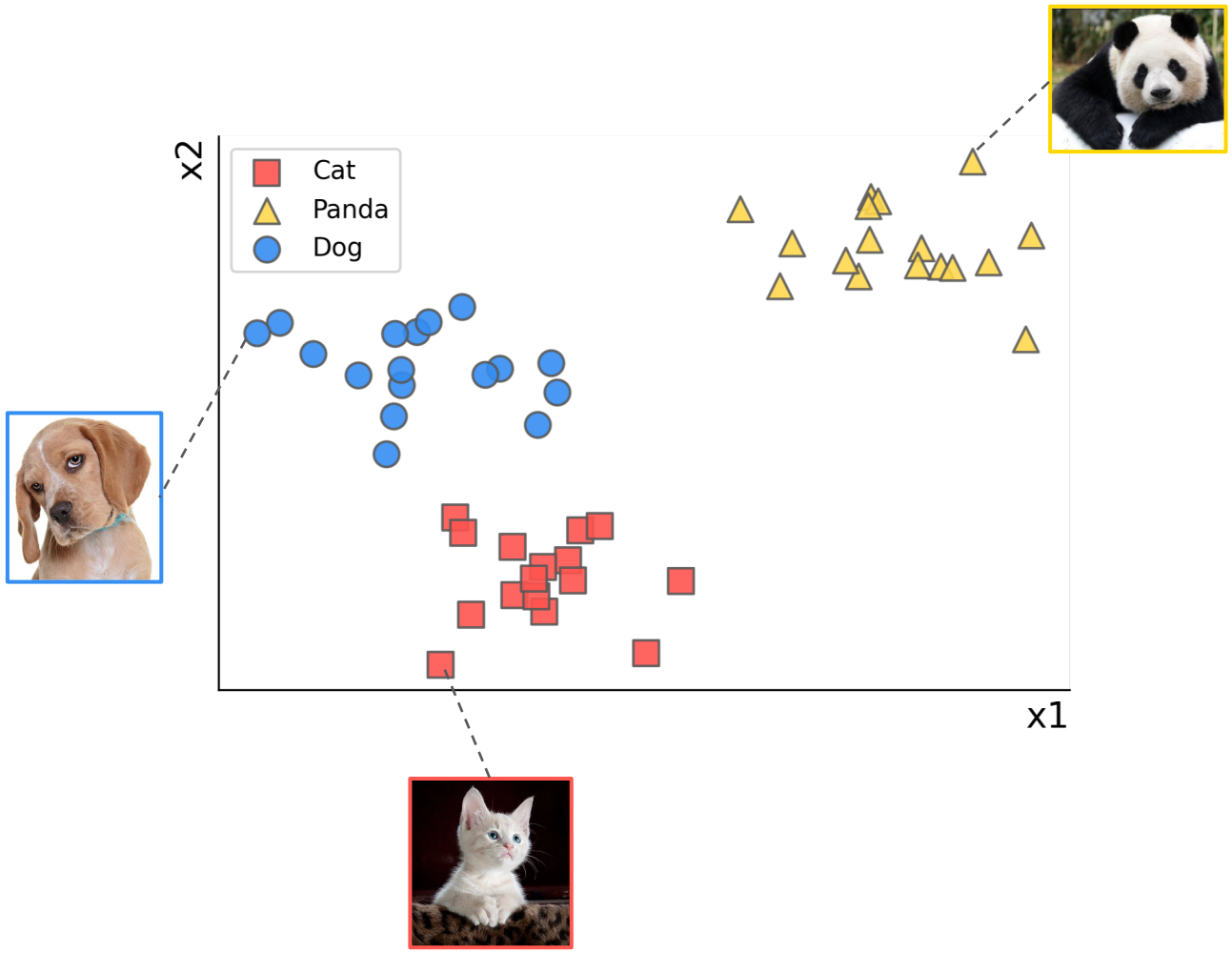
Kmeans

Prof. Dr. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas

Supervised Learning (e.g., classification)

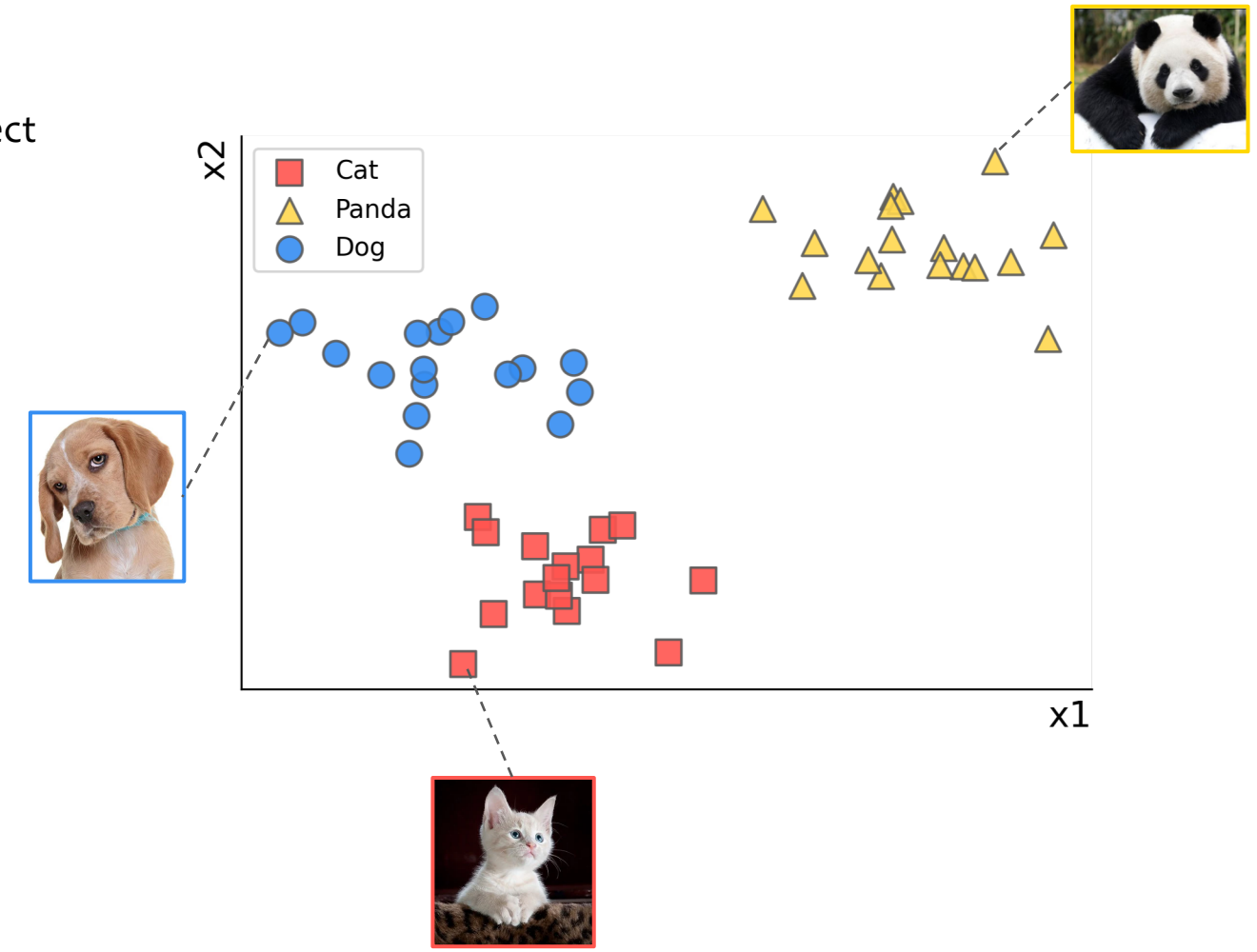
Previously **annotated/labeled** **training data**.



Supervised Learning (e.g., classification)

Previously **annotated/labeled** **training data**.

Mostly, **manual annotation**: human experts **manually** inspect and tag each sample (label assigment / data annotation).

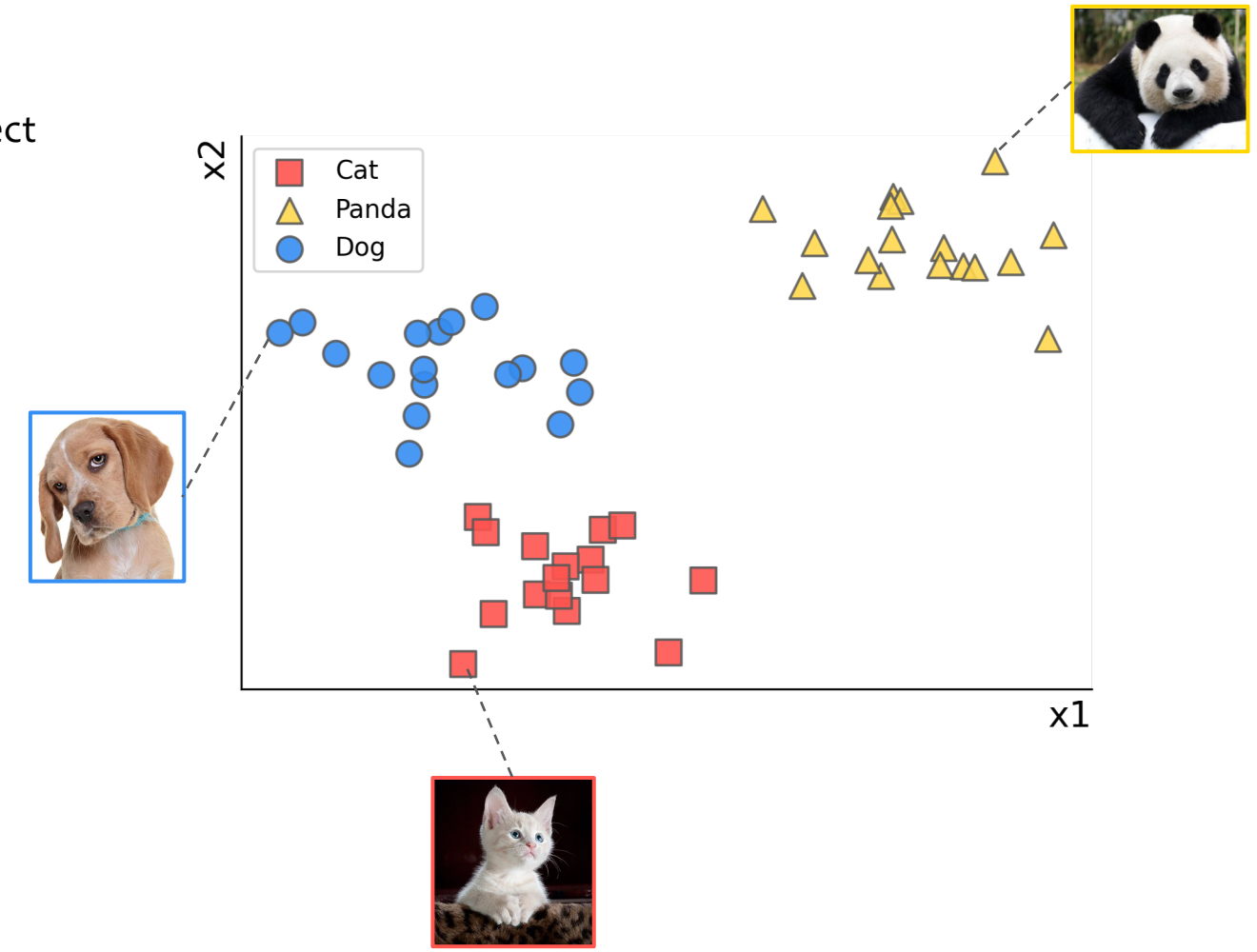


Supervised Learning (e.g., classification)

Previously **annotated/labeled** **training data**.

Mostly, **manual annotation**: human experts **manually** inspect and tag each sample (label assigment / data annotation).

Long, costly, error-prone, and tedious task, so it will usually only be done on a small subset of the available samples.



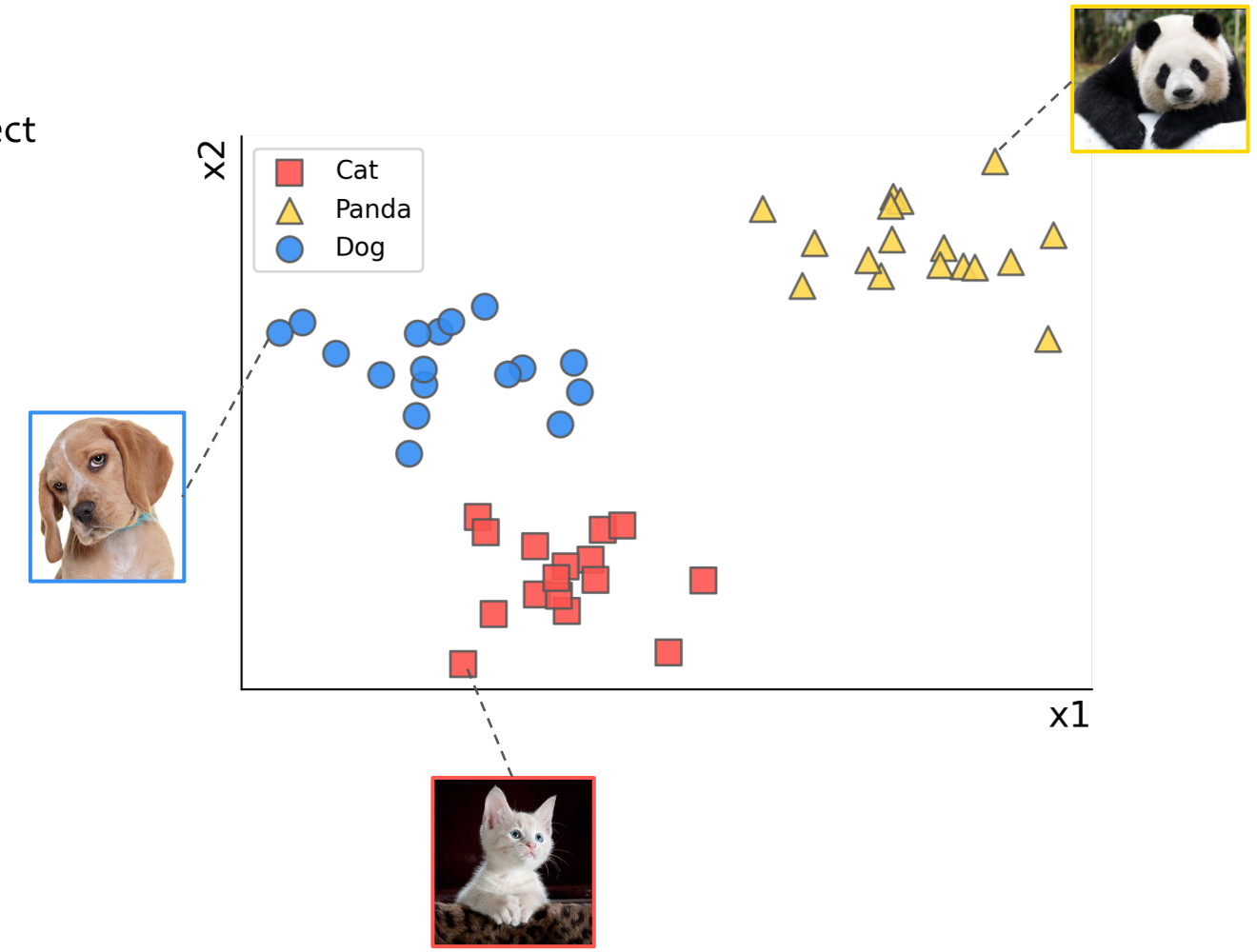
Supervised Learning (e.g., classification)

Previously **annotated/labeled** **training data**.

Mostly, **manual annotation**: human experts **manually** inspect and tag each sample (label assigment / data annotation).

Long, costly, error-prone, and tedious task, so it will usually only be done on a small subset of the available samples.

As a result, the **labeled dataset** will be **quite small**, and the classifier's performance will be disappointing.



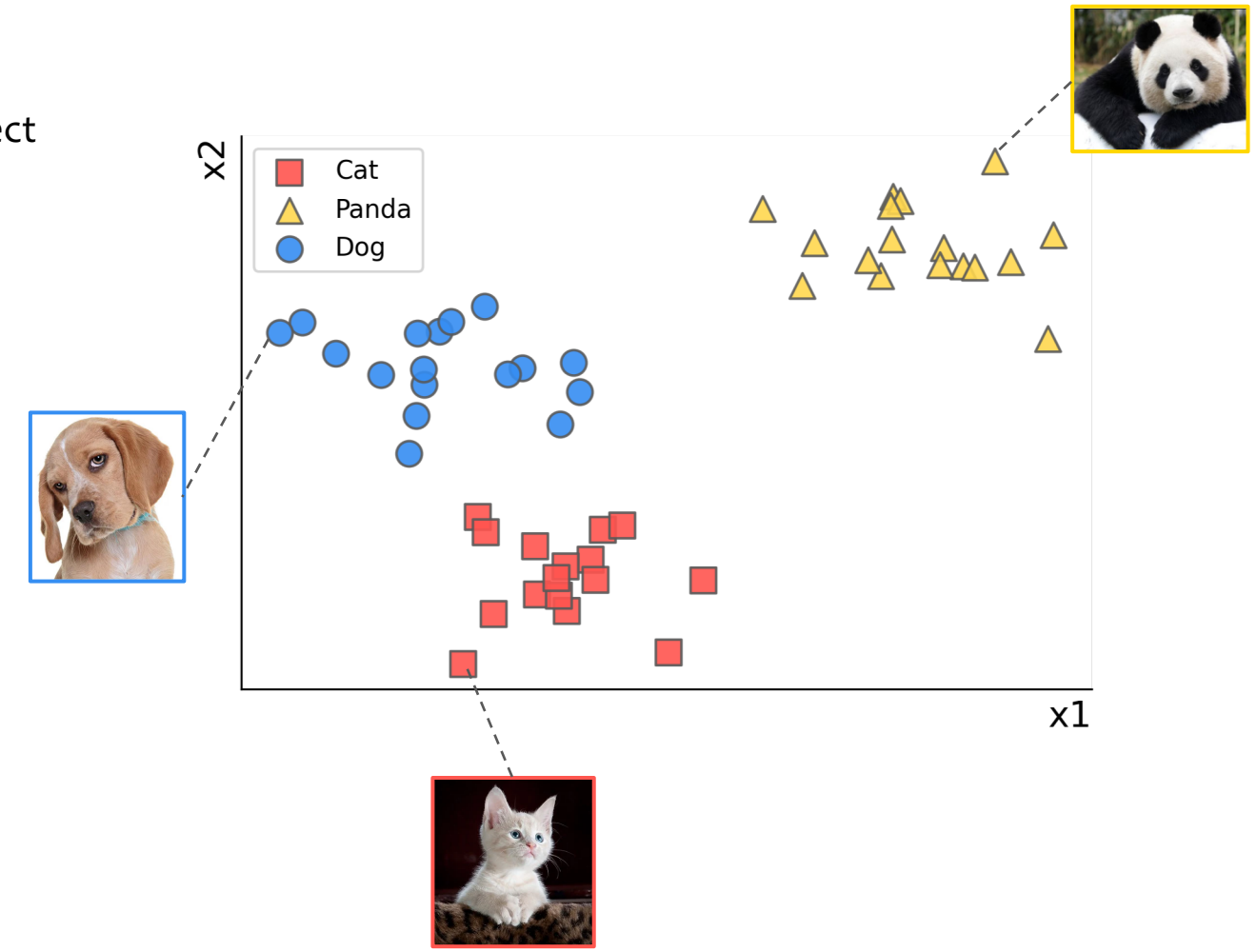
Supervised Learning (e.g., classification)

Previously **annotated/labeled** **training data**.

Mostly, **manual annotation**: human experts **manually** inspect and tag each sample (label assigment / data annotation).

Long, costly, error-prone, and tedious task, so it will usually only be done on a small subset of the available samples.

As a result, the **labeled dataset** will be **quite small**, and the classifier's performance will be disappointing.



⚠ These limitations are more accentuated in some applications, *e.g.*, medical diagnosis, for demaning specific experts often unavailable for data annotation.

Supervised Learning (e.g., classification)

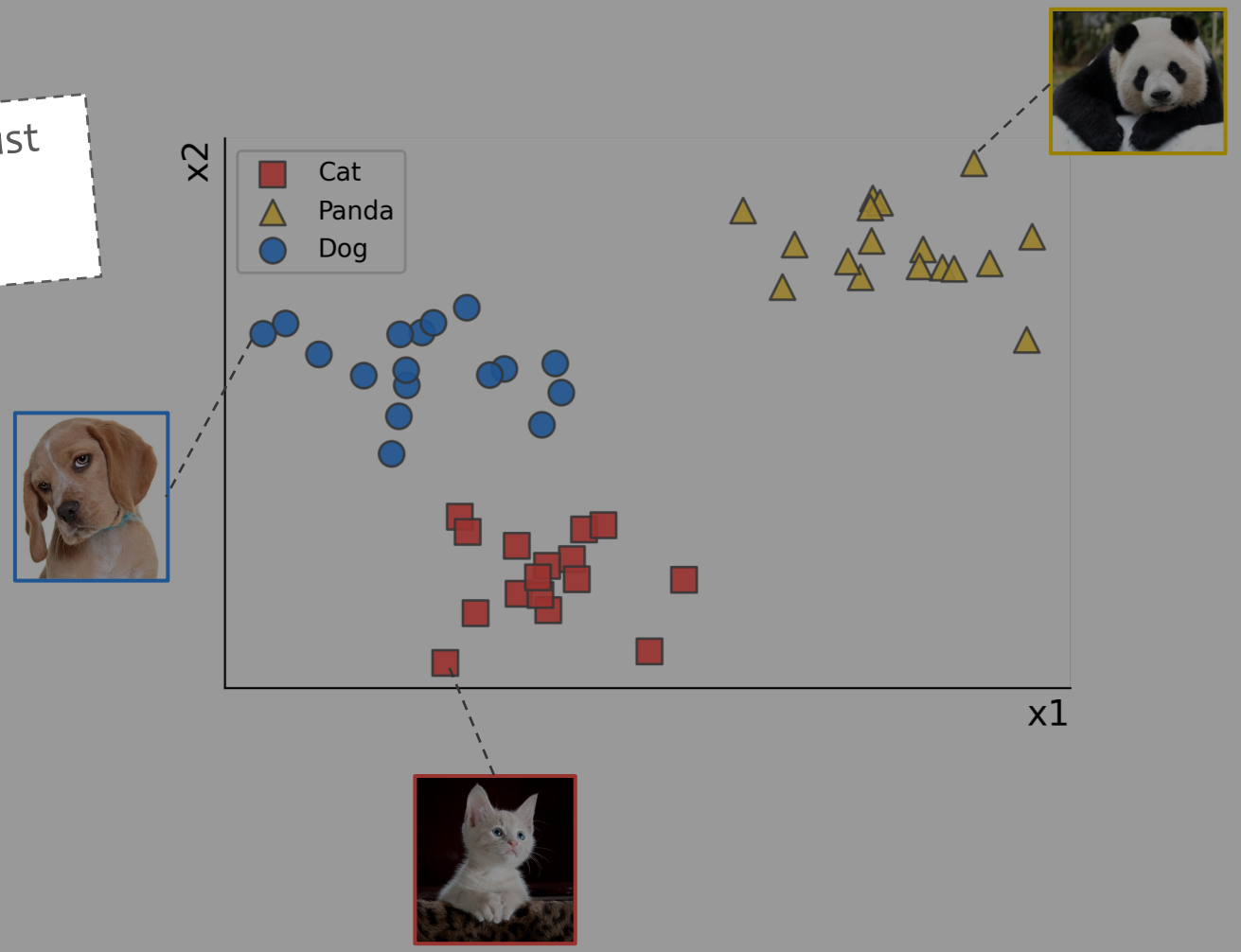
Previously **annotated/labeled** **training data**.

Mostly, **manual annotation**: human expert
and tag e

Wouldn't it be great if the algorithm could just **exploit the unlabeled data** without needing humans to label every sample?

Long, cost
tedious task, so it will **usually only be done on a small subset** of the available samples.

As a result, the **labeled dataset** will be **quite small**, and the classifier's performance will be disappointing.



⚠ These limitations are more accentuated in some applications, *e.g.*, medical diagnosis, for demaning specific experts often unavailable for data annotation.

Supervised Learning (e.g., classification)

Previously **annotated/labeled** **training data**.

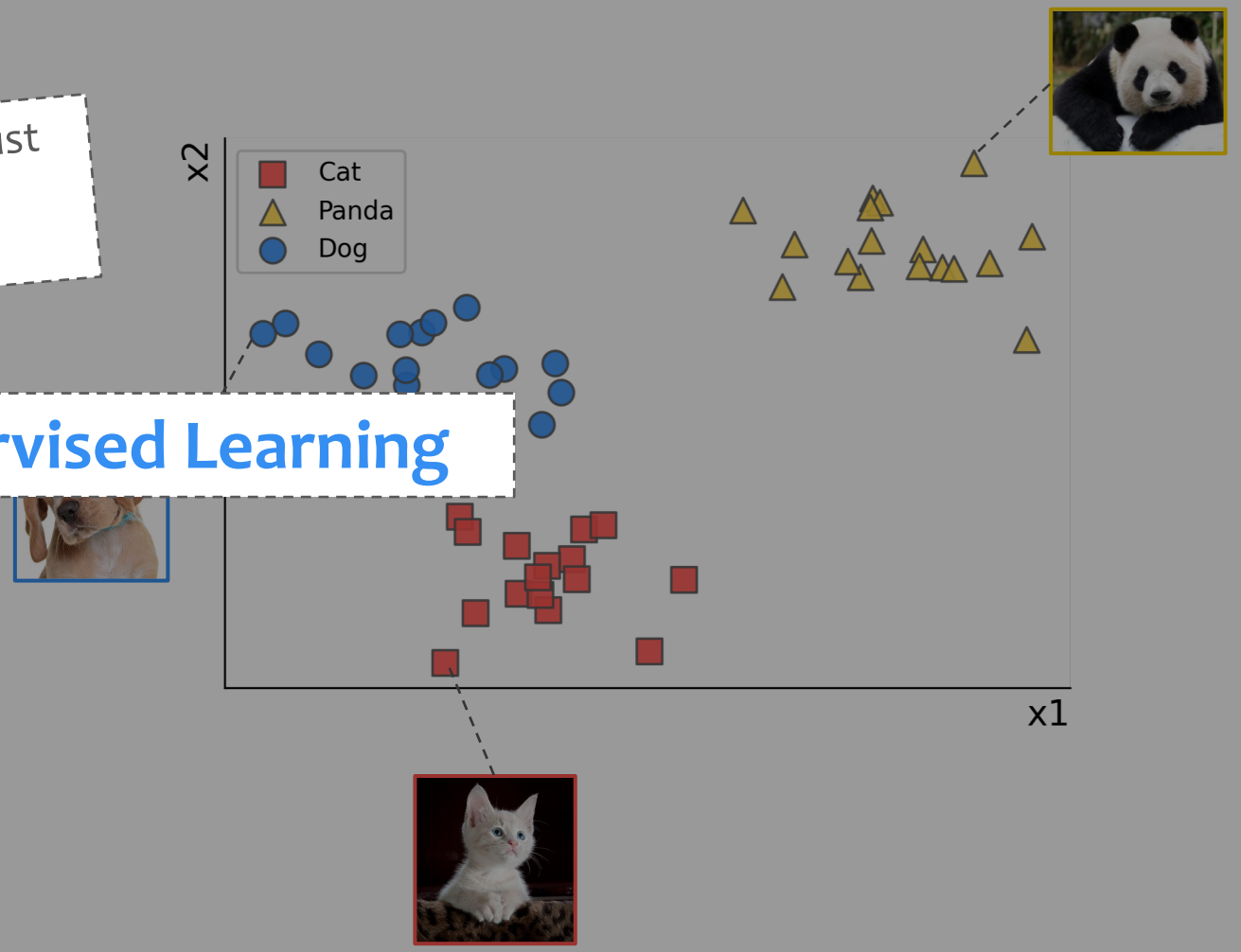
Mostly, **manual annotation**: human expert
and tag e

Long, cos tedious task, so it will
usually only be done on a small subset of the available
samples.

As a result, the **labeled dataset** will be **quite small**, and
the classifier's performance will be disappointing.

Wouldn't it be great if the algorithm could just **exploit the unlabeled data** without needing humans to label every sample?

Unsupervised Learning

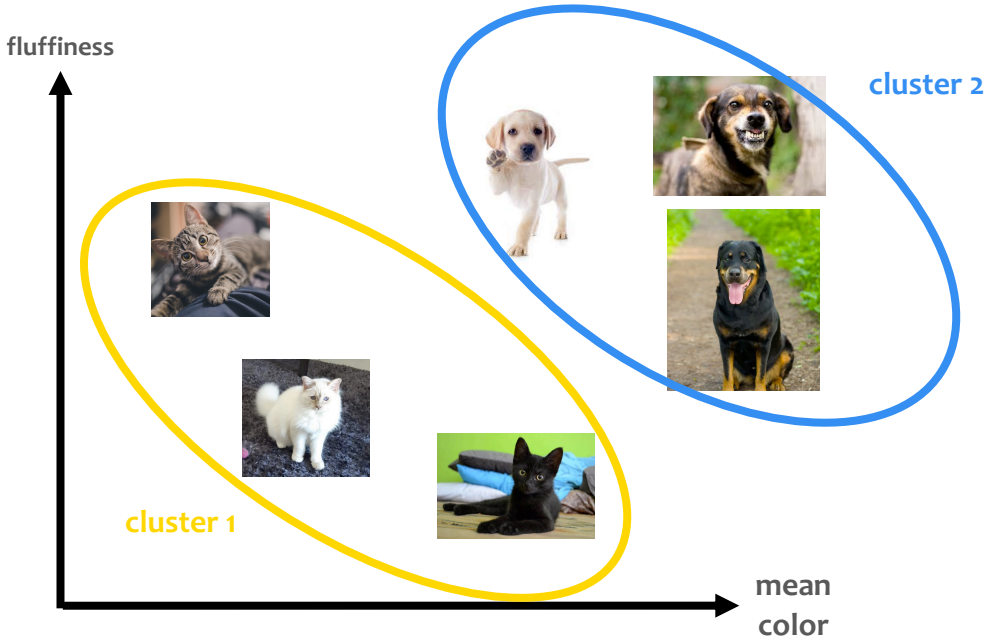


These limitations are more accentuated in some applications, *e.g.*, medical diagnosis, for demaning specific experts often unavailable for data annotation.

Unsupervised Learning

no supervision → **no** labels / annotations

Clustering

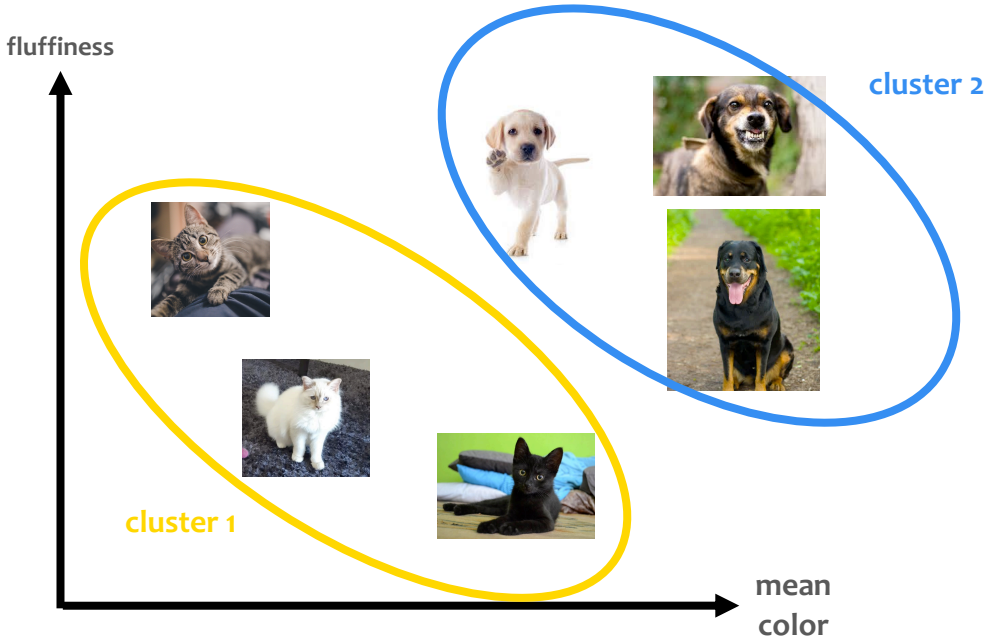


- k-Means
- DBSCAN
- Mean-shift
- OPF Clustering
- Hierarchical Clustering
- ...

Unsupervised Learning

no supervision → no labels / annotations

Clustering

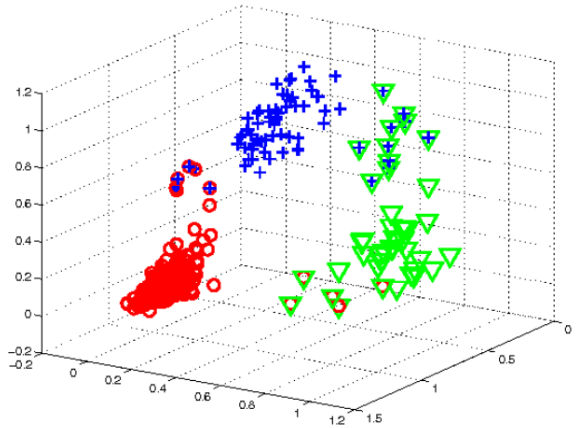


- k-Means
- DBSCAN
- Mean-shift
- OPF Clustering
- Hierarchical Clustering
- ...

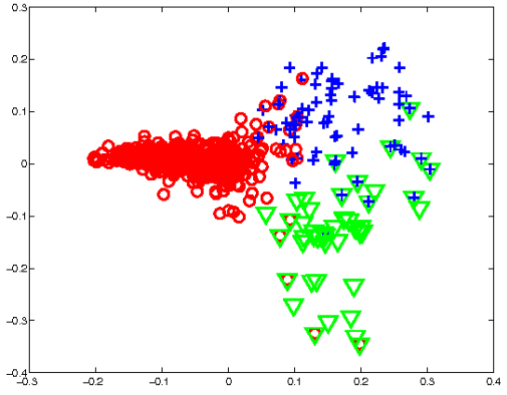
Dimensionality Reduction / Projection

Project a **high-dimensional data** on to a **lower-dimensional feature space** without losing much information.

Projection on to a **2D** or **3D** space is used to **visualize and understand** the data.



high-dimensional data



projection

- Principal Component Analysis (PCA)
- Locally Linear Embedding (LLE)
- t-SNE
- UMAP
- ...

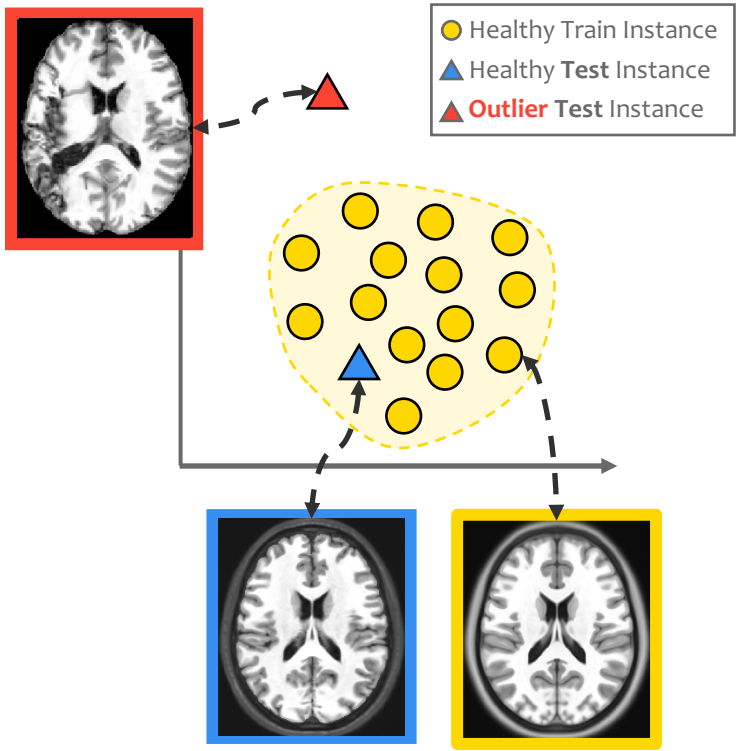
Unsupervised Learning

Anomaly/Novelty/Outlier Detection

The algorithm is fed with **training data** containing **only** a **single class** (normal, control, healthy).

New unseen instances (test) that **DOES NOT** look a **normal one** is an **outlier**, being considered as an **anomaly**.

The **training set** must be very **clean** and **representative**.



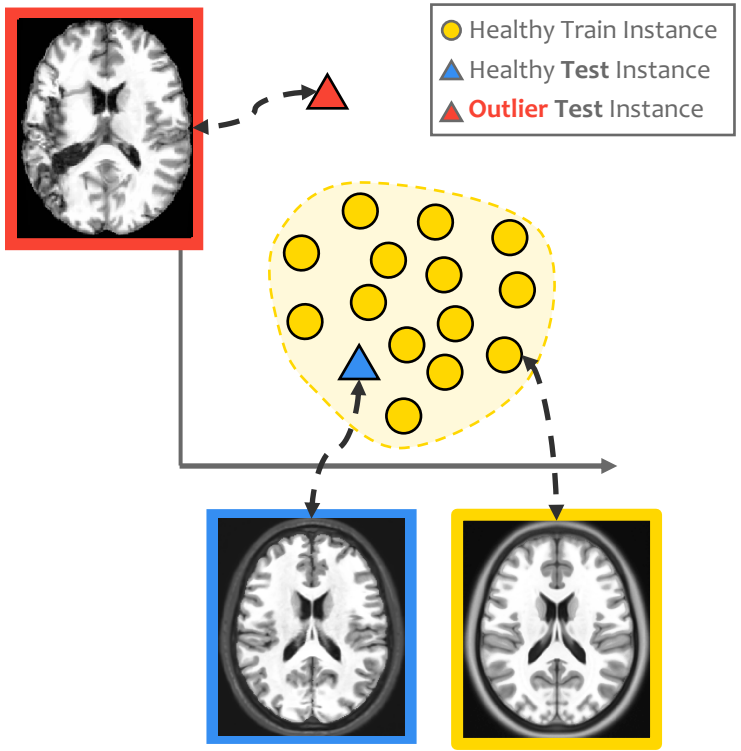
Unsupervised Learning

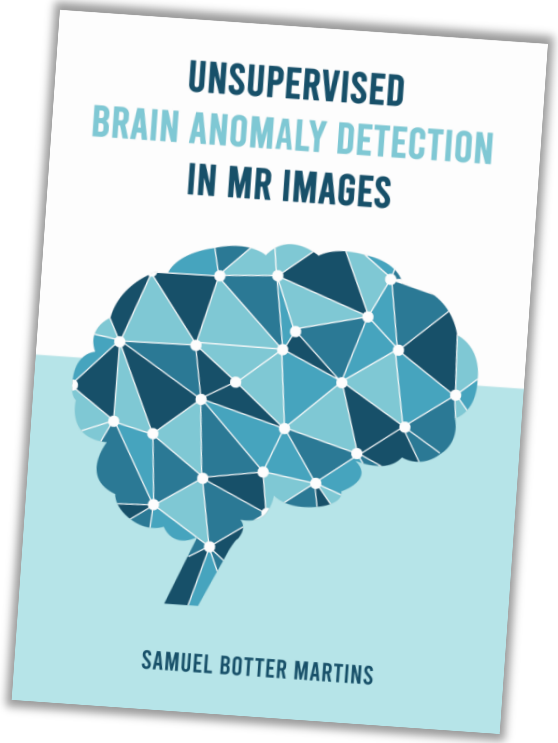
Anomaly/Novelty/Outlier Detection

The algorithm is fed with **training data** containing **only** a **single class** (normal, control, healthy).

New unseen instances (test) that **DOES NOT** look a **normal one** is an **outlier**, being considered as an **anomaly**.

The **training set** must be very **clean** and **representative**.

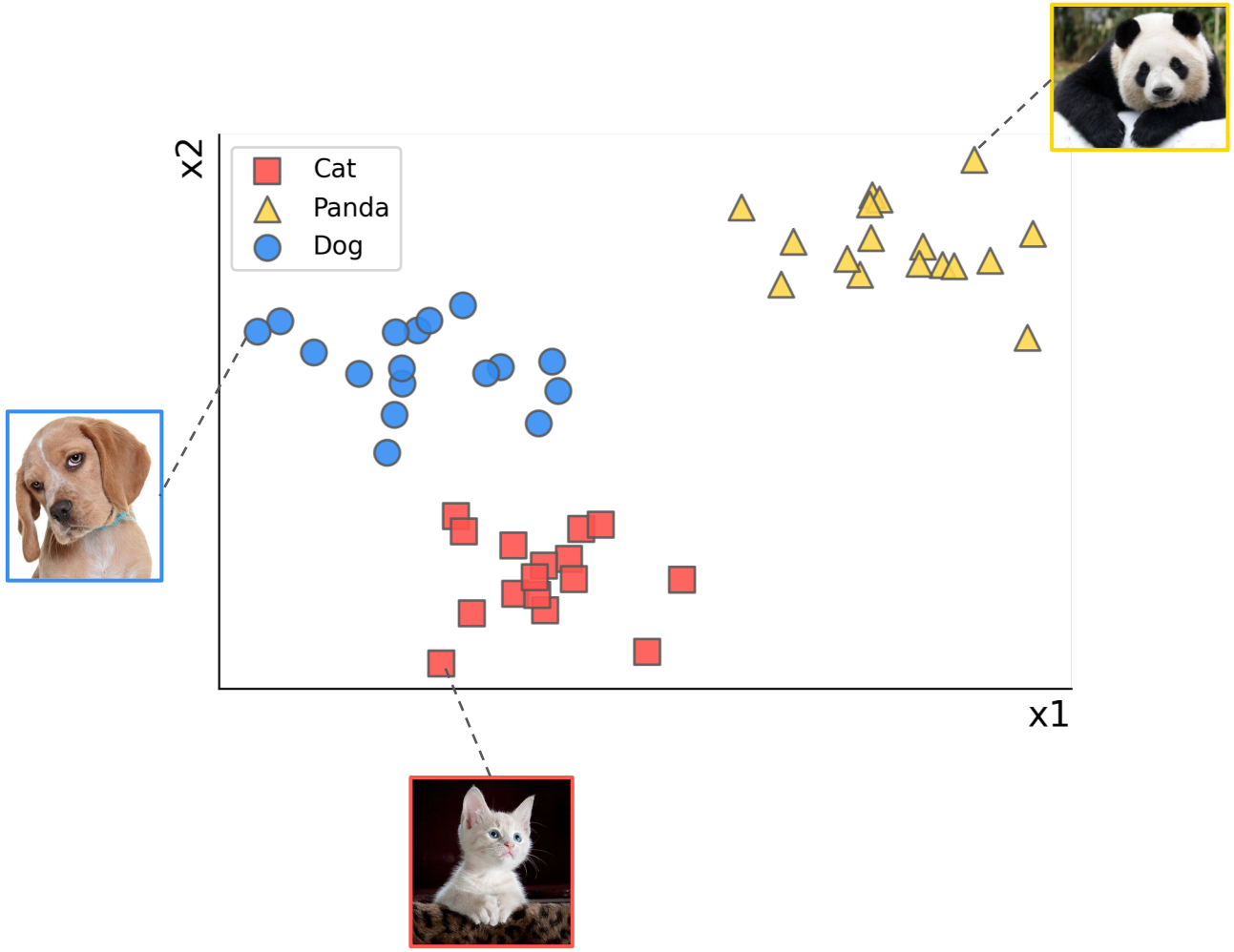




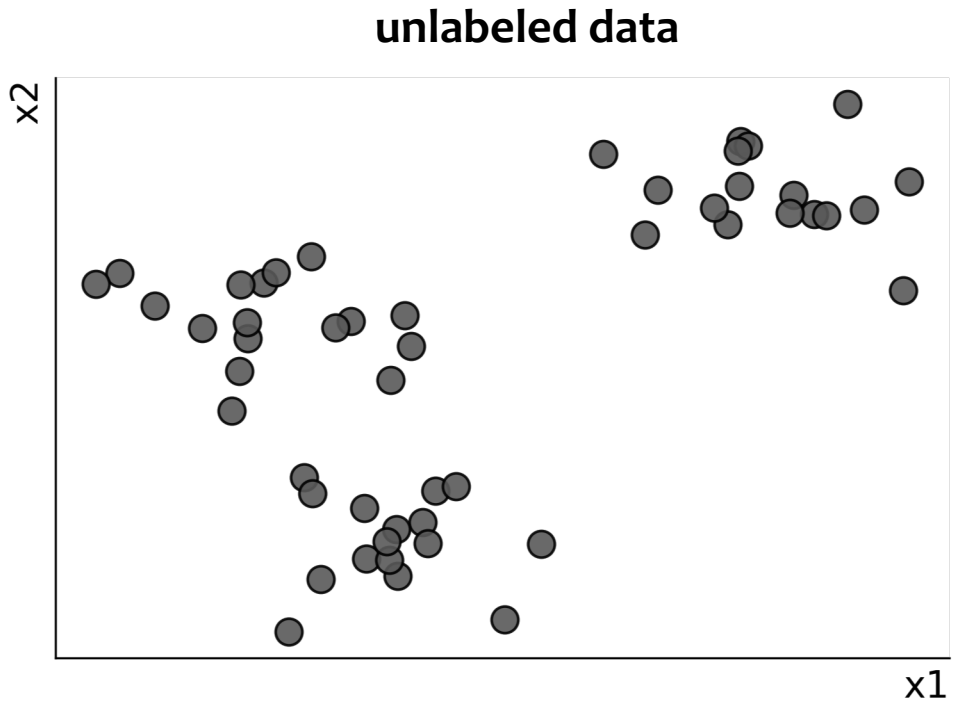
https://research.rug.nl/en/publications/unsupervised-brain-anomaly-detection-in-mr-images

Let’s start with **Clustering!**

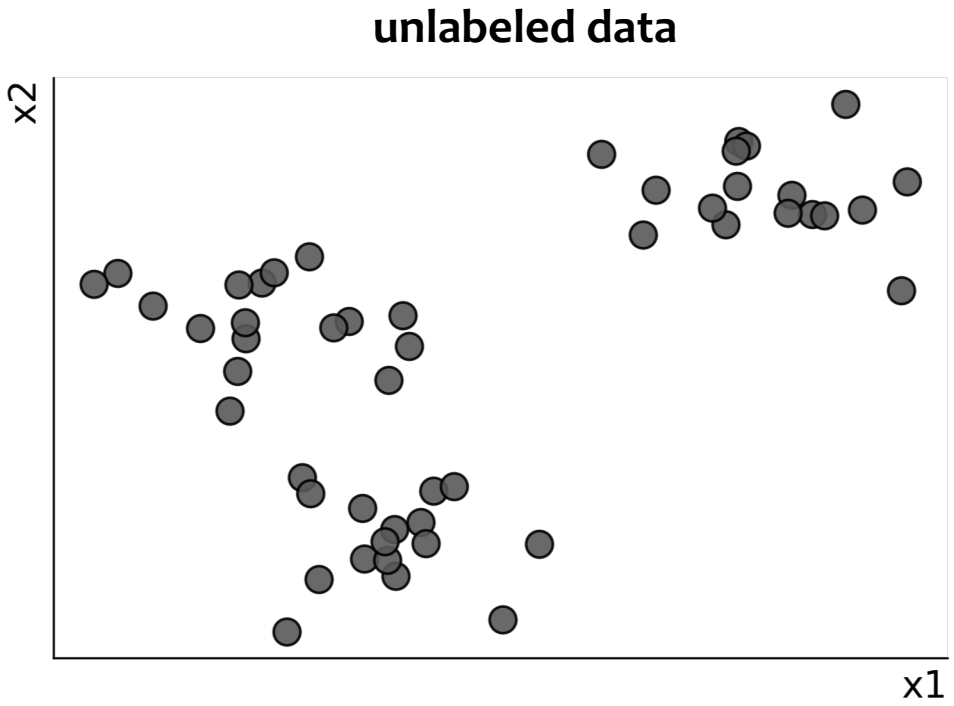
What if instead of having a set of previously labeled data...



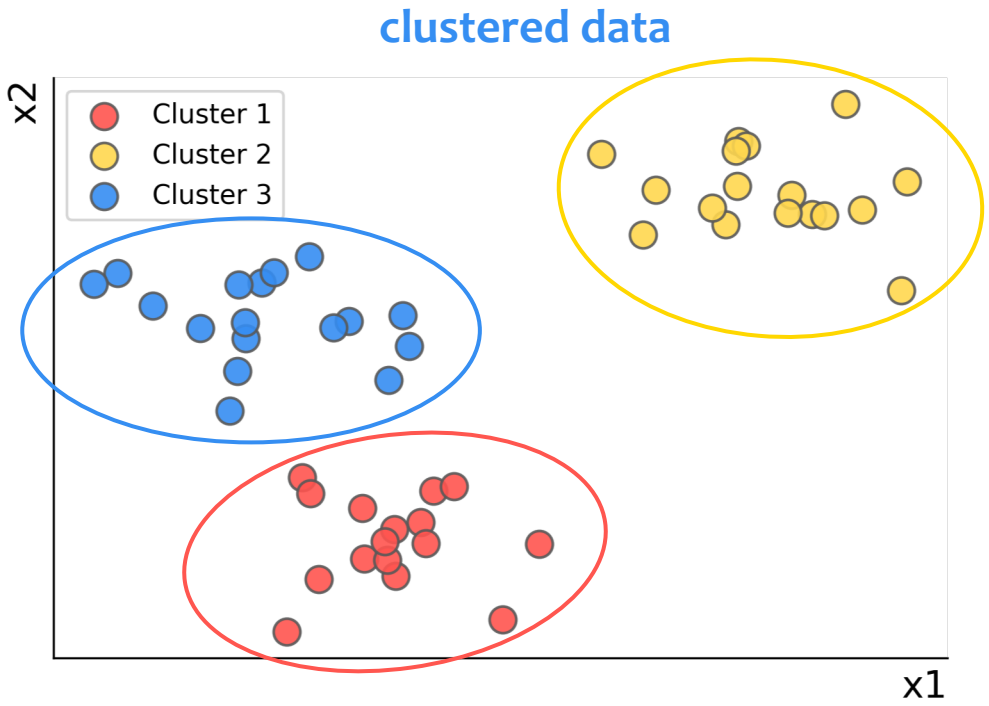
What if instead of having a set of previously labeled data, the **data were unlabeled**?



One could use some **clustering algorithm** **to group similar instances** together into *clusters/groups.*

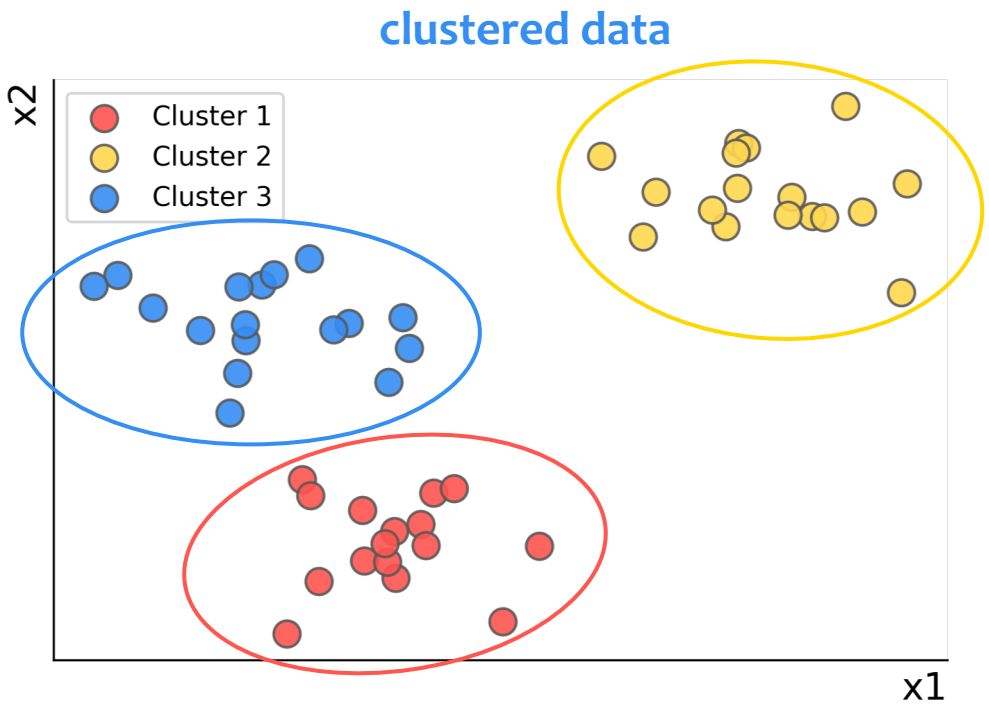


One could use some **clustering algorithm** **to group similar instances** together into *clusters/groups.*



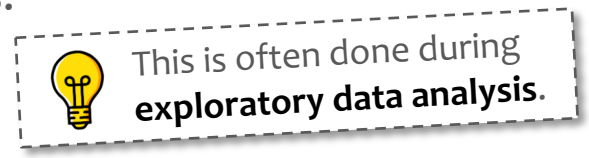
One could use some **clustering algorithm** **to group similar instances** together into *clusters/groups.*

One may then **analyze each cluster individually** in order to find some specific pattern/behaviour/characteristic between their instances.

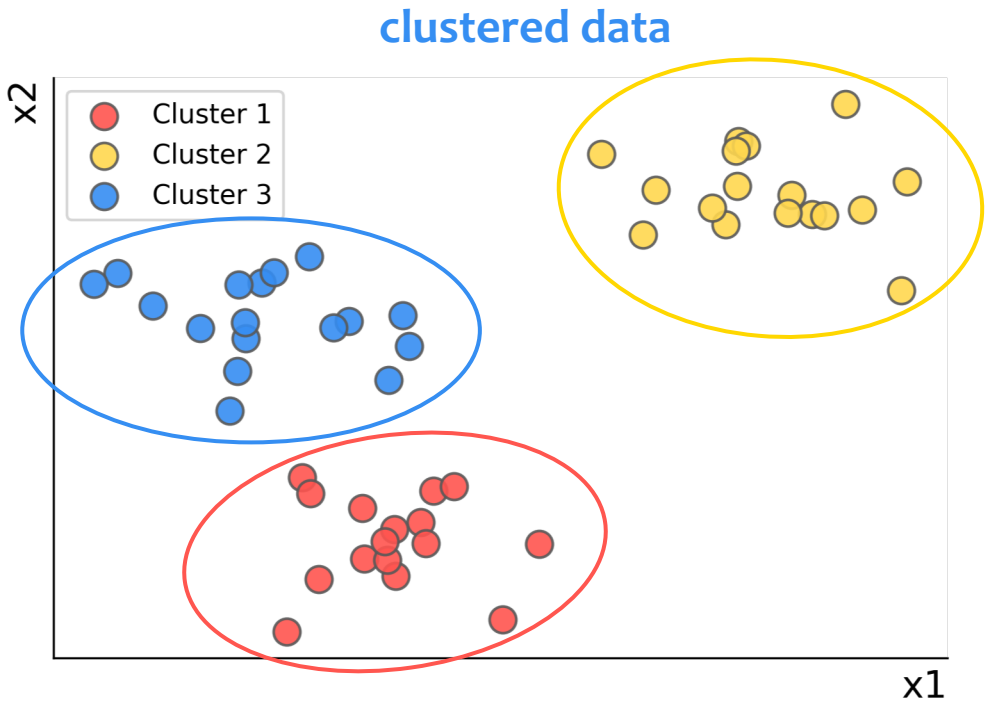


One could use some **clustering algorithm** **to group similar instances** together into *clusters/groups.*

One may then **analyze each cluster individually** in order to find some specific pattern/behaviour/characteristic between their instances.



This is often done during **exploratory data analysis**.



Clustering Approaches

Clustering Approaches

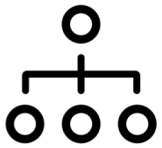
Partition methods

- Given a dataset with m samples, a partition method constructs **k partitions, k ≤ m**, where each partition represents a **cluster**, satisfying:
  - Each cluster must contain at least one sample;
  - Each sample must belong to exactly one cluster;

Clustering Approaches

Partition methods

- Given a dataset with m samples, a partition method constructs **k partitions, k ≤ m**, where each partition represents a **cluster**, satisfying:
  - Each cluster must contain at least one sample;
  - Each sample must belong to exactly one cluster;



Hierarchical methods

- Creates a **hierarchical decomposition** of the given dataset (agglomerative or divisive).
- Hierarchical methods suffer form the fact that once a step (merge or split) is done, it can never be undone.

Clustering Approaches

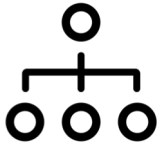
Partition methods

- Given a dataset with m samples, a partition method constructs **k partitions, k ≤ m**, where each partition represents a **cluster**, satisfying:
  - Each cluster must contain at least one sample;
  - Each sample must belong to exactly one cluster;

Density-based methods

- Clustering methods based on the notion of **density**;
- Their general idea is to continue growing the given cluster as long as the **density** (number of data points) in the neighborhood exceeds some threshold;
- They can be used to filter out noise (outliers) and discover clusters of arbitrary shape;

Hierarchical methods



- Creates a **hierarchical decomposition** of the given dataset (agglomerative or divisive).
- Hierarchical methods suffer form the fact that once a step (merge or split) is done, it can never be undone.

Clustering Approaches

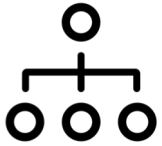
Partition methods

- Given a dataset with m samples, a partition method constructs **k partitions, k ≤ m**, where each partition represents a **cluster**, satisfying:
  - Each cluster must contain at least one sample;
  - Each sample must belong to exactly one cluster;

Density-based methods

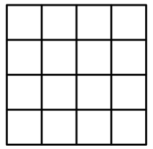
- Clustering methods based on the notion of **density**;
- Their general idea is to continue growing the given cluster as long as the **density** (number of data points) in the neighborhood exceeds some threshold;
- They can be used to filter out noise (outliers) and discover clusters of arbitrary shape;

Hierarchical methods



- Creates a **hierarchical decomposition** of the given dataset (agglomerative or divisive).
- Hierarchical methods suffer form the fact that once a step (merge or split) is done, it can never be undone.

Grid-based methods

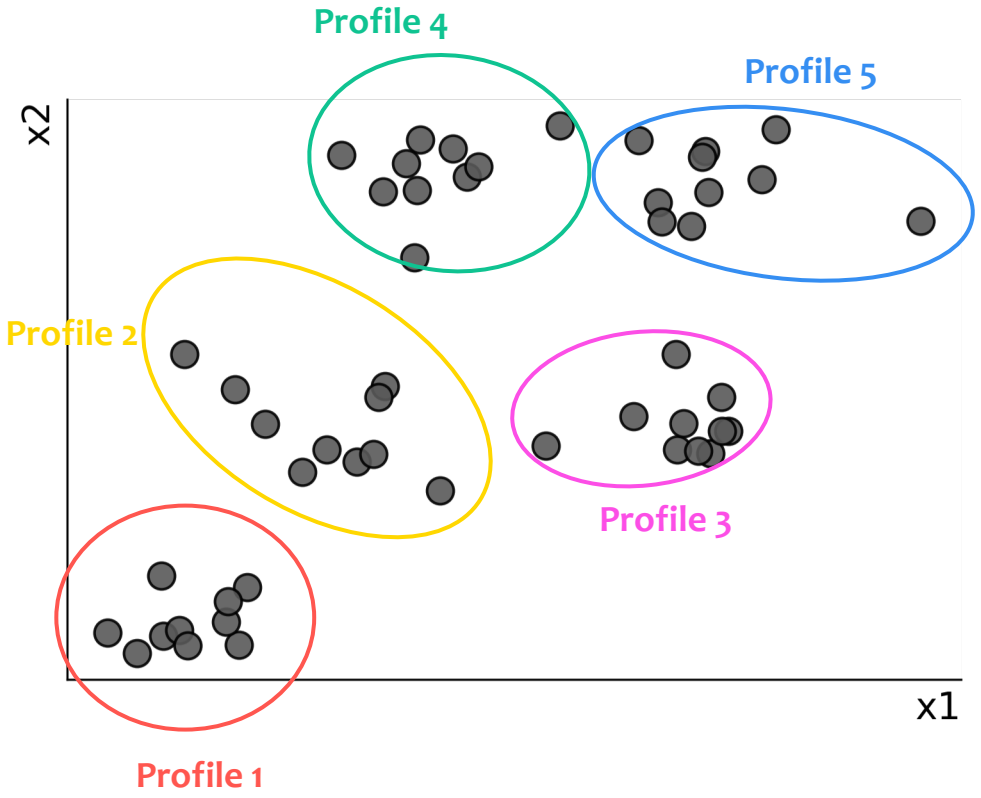


- First divide the space into regular grids, then perform **clustering** on the grids;

Clustering is used in several applications, including:

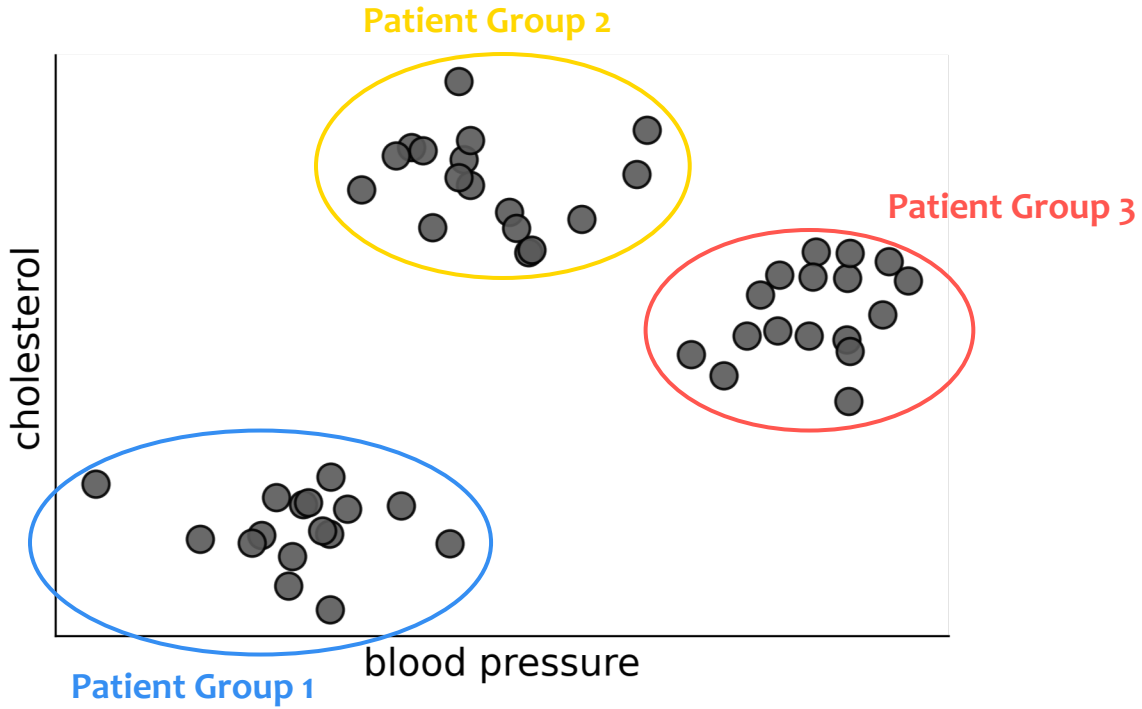
Clustering is used in several applications, including:

- Customer segmentation / recommender systems;



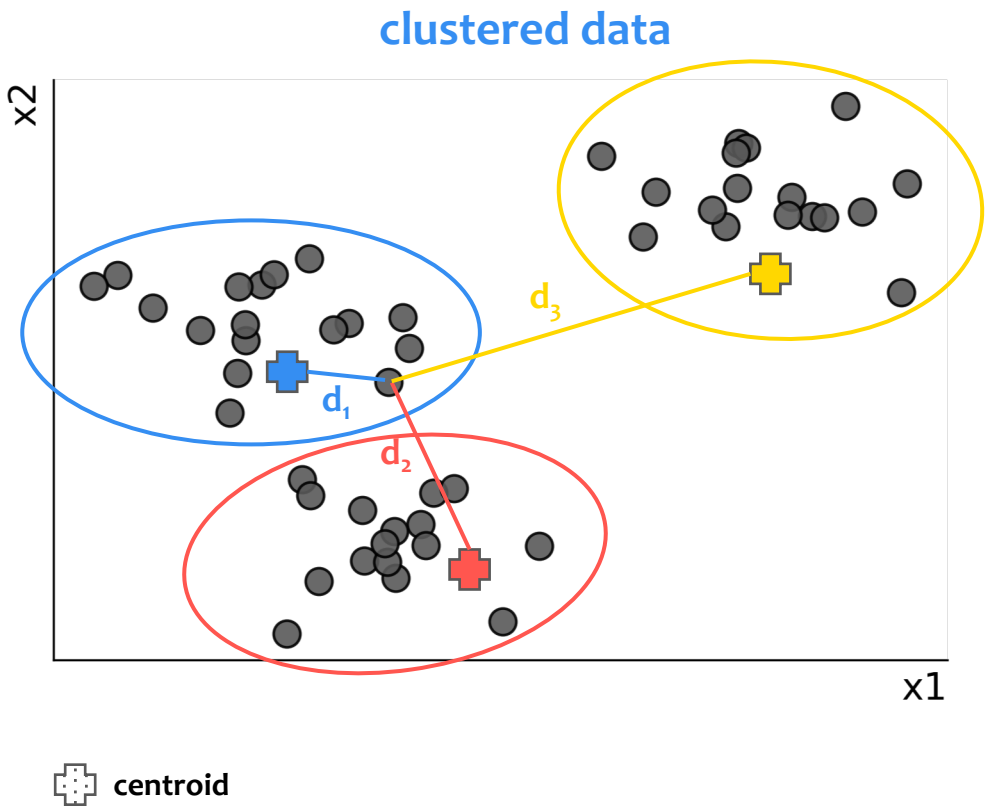
Clustering is used in several applications, including:

- Customer segmentation / recommender systems;
- Data Analysis;



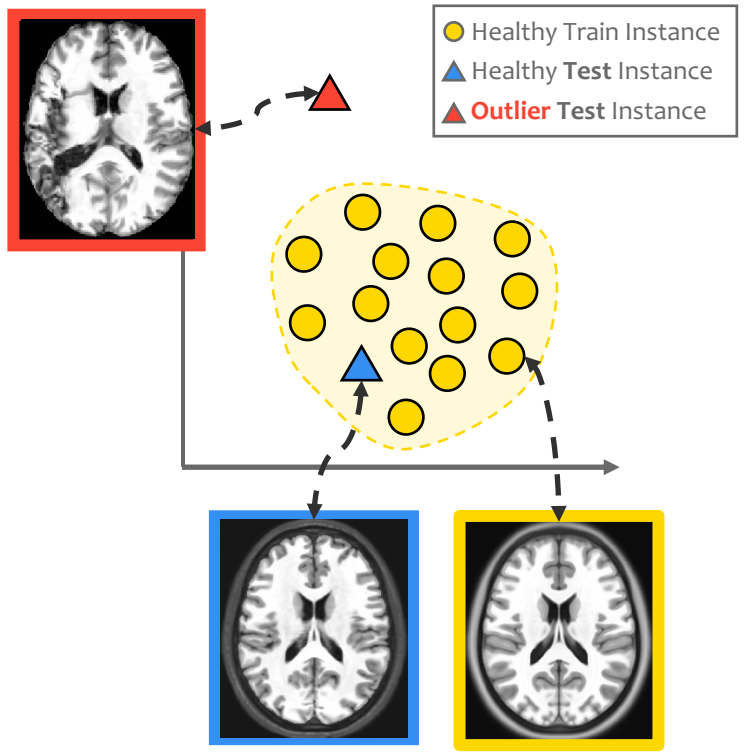
Clustering is used in several applications, including:

- Customer segmentation / recommender systems;
- Data Analysis;
- Dimensionality reduction technique;



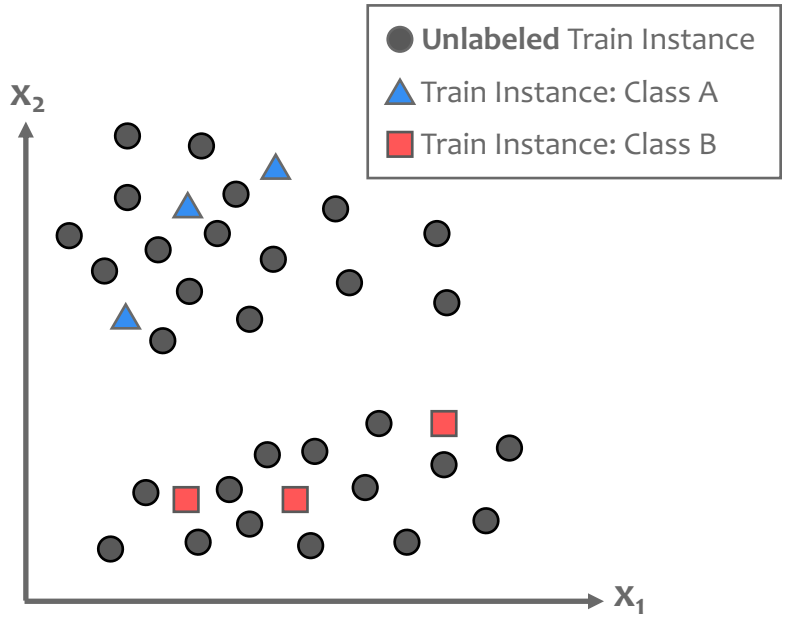
Clustering is used in several applications, including:

- Customer segmentation / recommender systems;
- Data Analysis;
- Dimensionality reduction technique;
- Anomaly detection;



Clustering is used in several applications, including:

- Customer segmentation / recommender systems;
- Data Analysis;
- Dimensionality reduction technique;
- Anomaly detection;
- Semi-supervised learning;



Clustering is used in several applications, including:

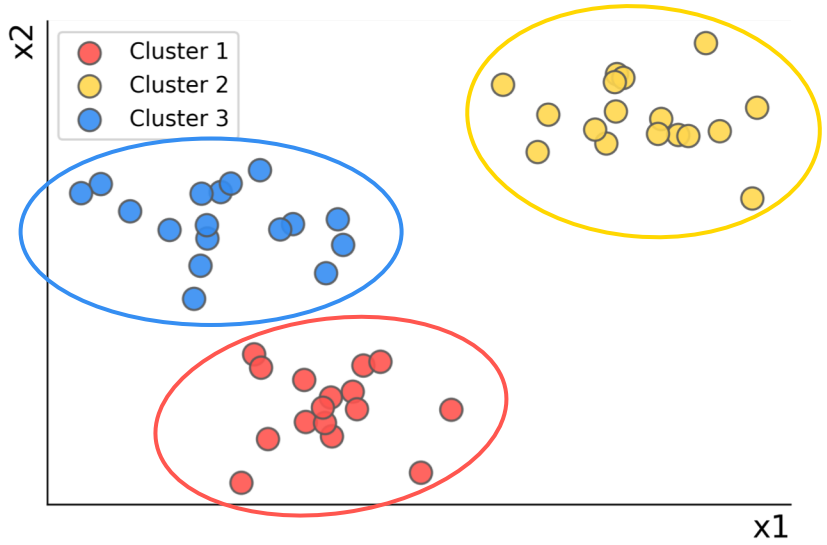
- Customer segmentation / recommender systems;
- Data Analysis;
- Dimensionality reduction technique;
- Anomaly detection;
- Semi-supervised learning;
- others...

Clustering is used in several applications, including:

- Customer segmentation / recommender systems;
- Data Analysis;
- Dimensionality reduction technique;
- Anomaly detection;
- Semi-supervised learning;
- others...

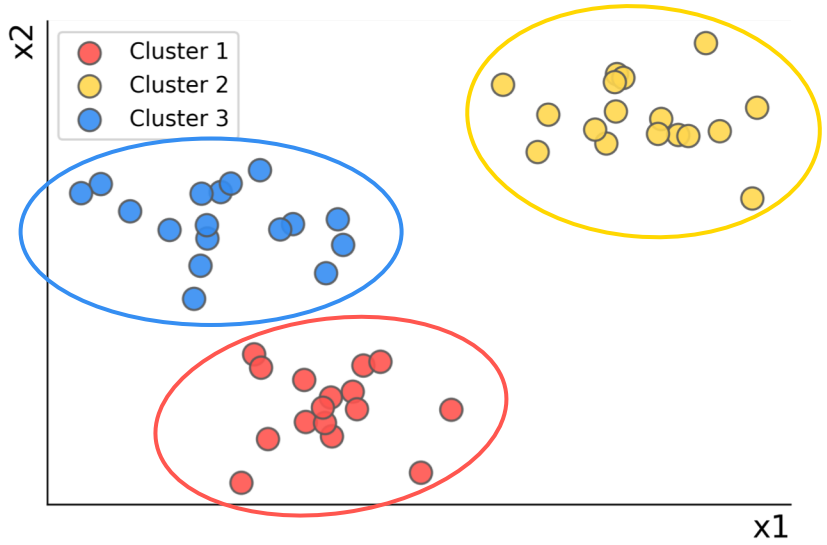
Let's see **KMeans:** the most classical clustering algorithm!

KMeans



KMeans

- Proposed by Stuart Lloyd at Bell Labs in 1957 but only published in 1982;
- Sometimes referred to as Lloyd–Forgy.

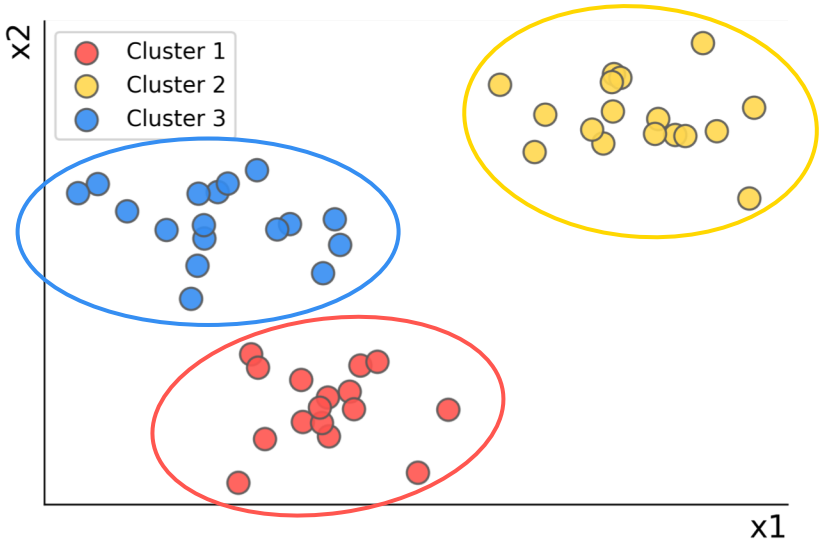


KMeans

- Proposed by Stuart Lloyd at Bell Labs in 1957 but only published in 1982;
- Sometimes referred to as Lloyd–Forgy.

Goal

- **To partition** the dataset into k **clusters**:
  - Each cluster must contain **at least one** sample;
  - Each sample must belong to **exactly one** cluster;
  - The **mass center (mean)** of a cluster corresponds to its **centroid**;



KMeans

Initialization:

1) Choose the **number k** of clusters;

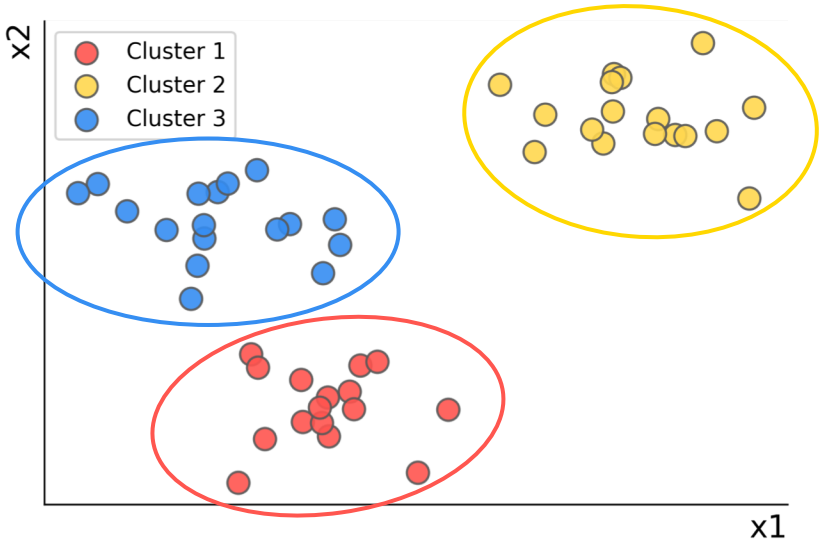
2) Choose the **initial k centroids** (cluster centers);

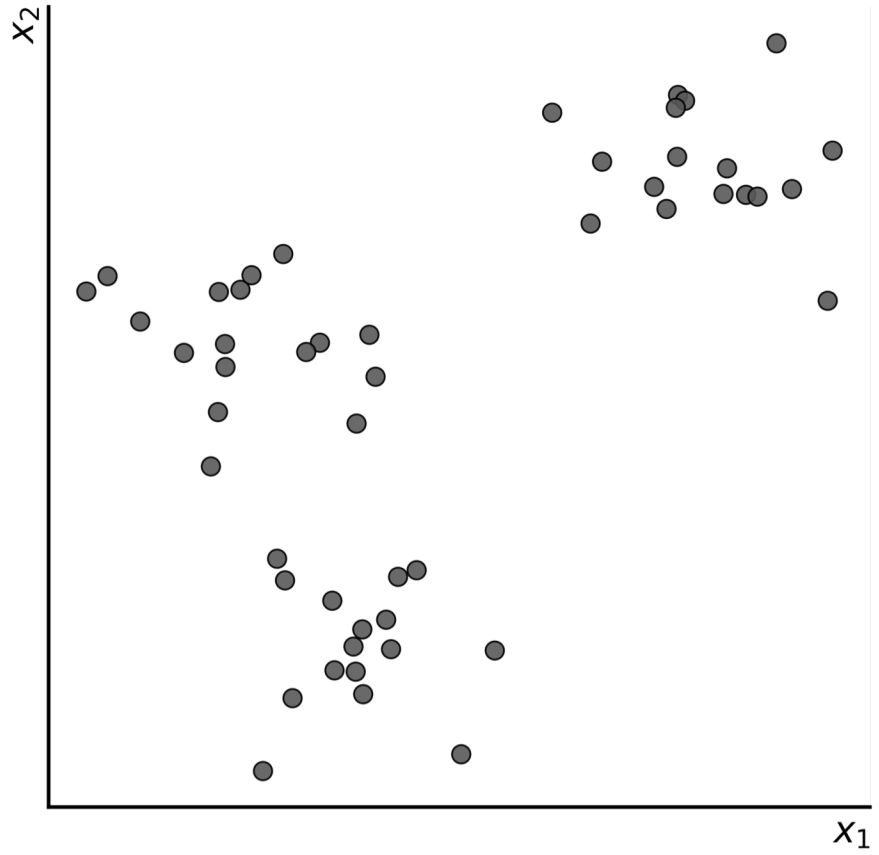
Learning:

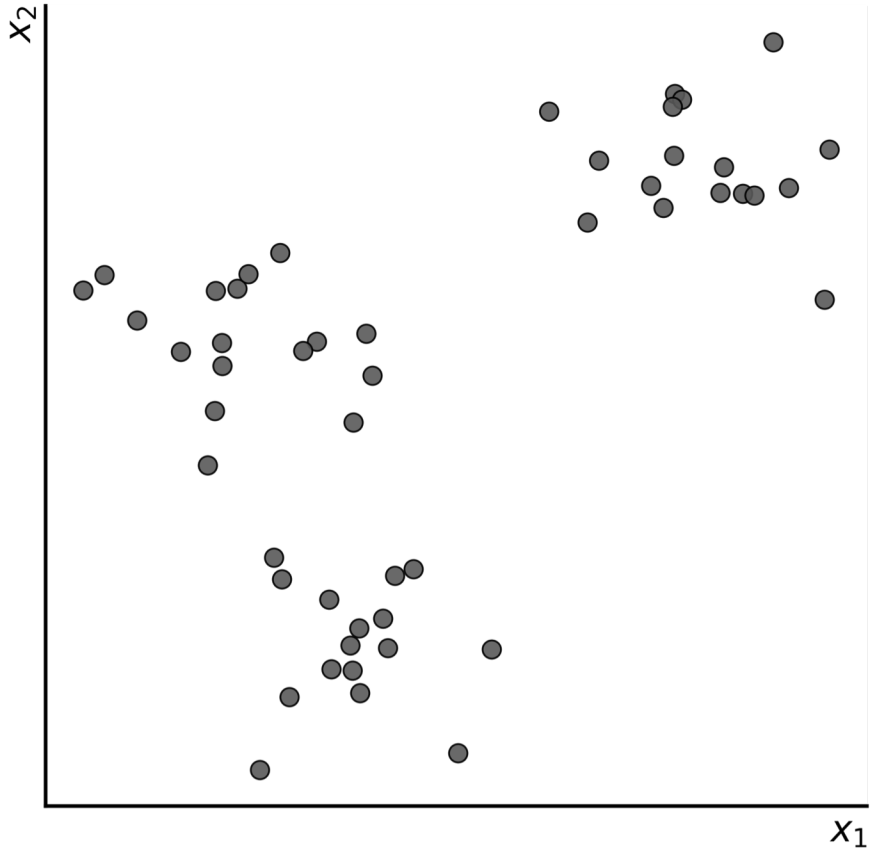
3) Repeat until **convergence**:

- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the centers of their clusters;

6) Assign each instance to **the final nearest** centroid;

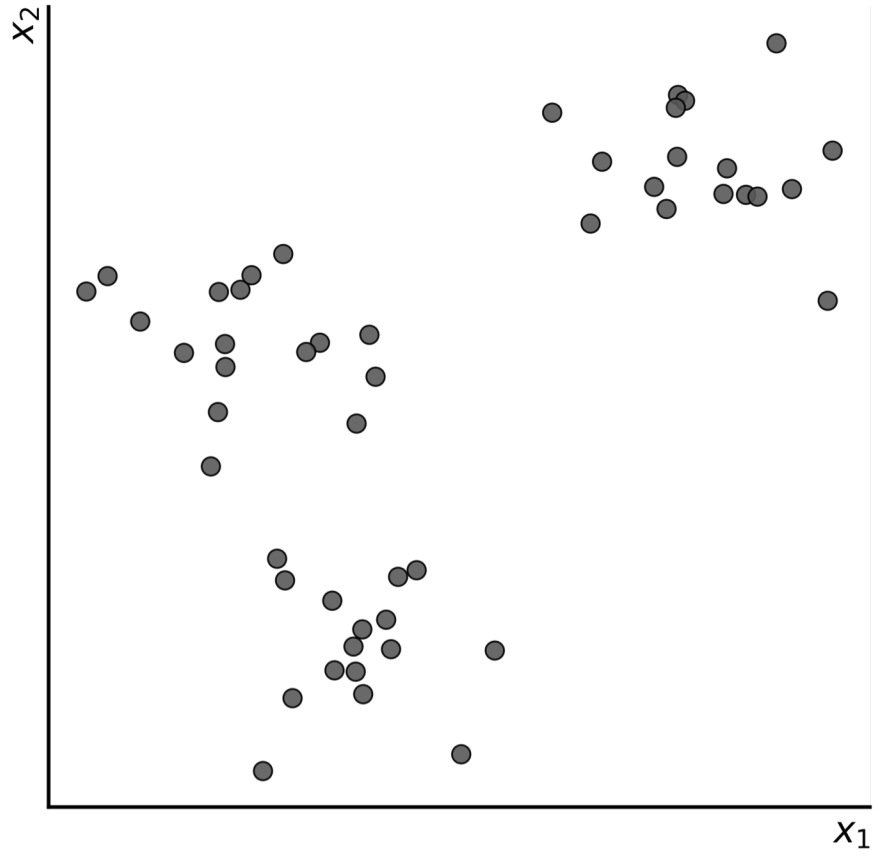



x_2
x_1



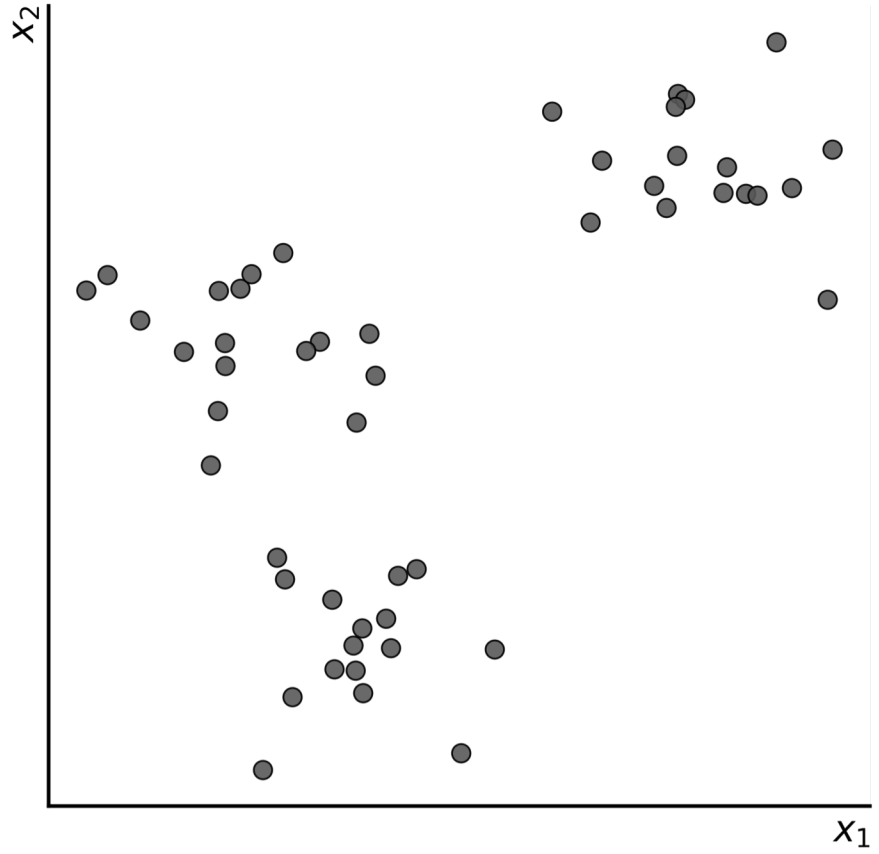
Consider that $\mathbf{x_1}$ and $\mathbf{x_2}$ have the **same scale**.

1) Choose the **number k** of clusters;



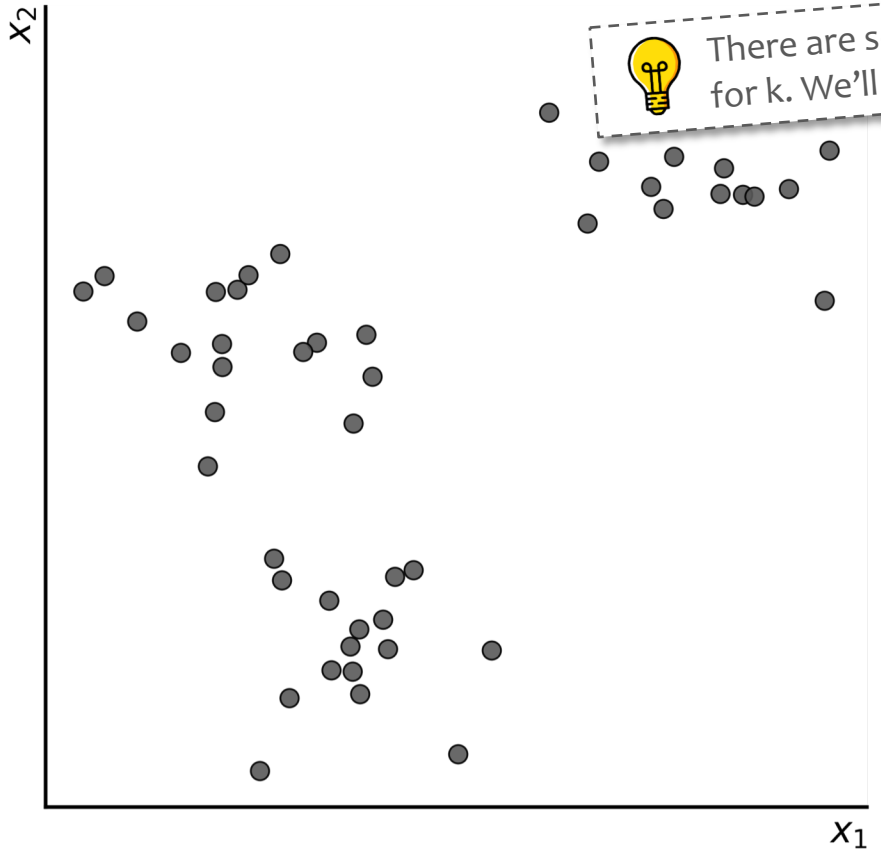
1) Choose the **number k** of clusters;

Let's consider: **k=3**



1) Choose the **number k** of clusters;

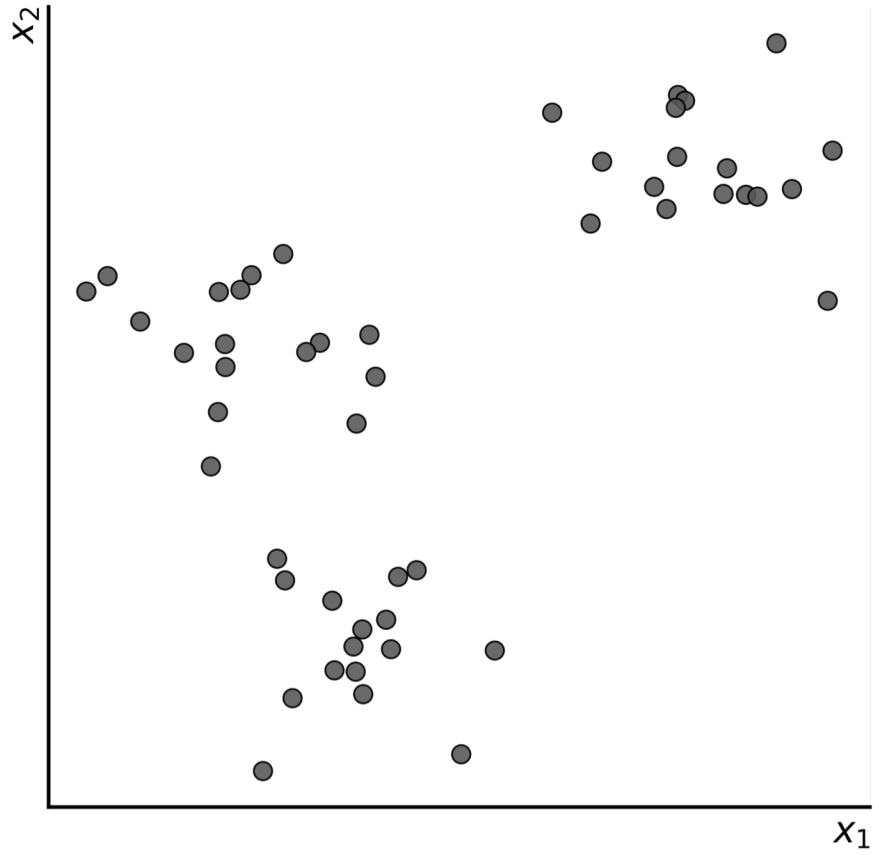
Let's consider: **k=3**



This **hyperparameter is crucial** to obtain high-quality models.

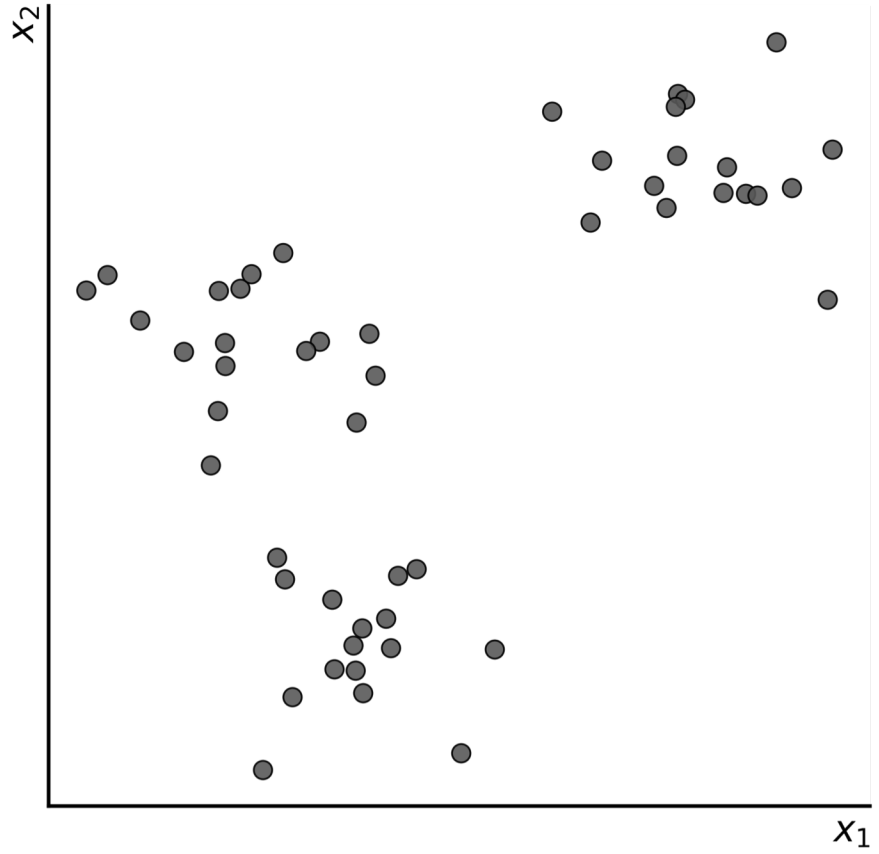
There are some ways to find a proper value for k. We'll see them soon!

2) Choose the **initial k centroids** (cluster centers)



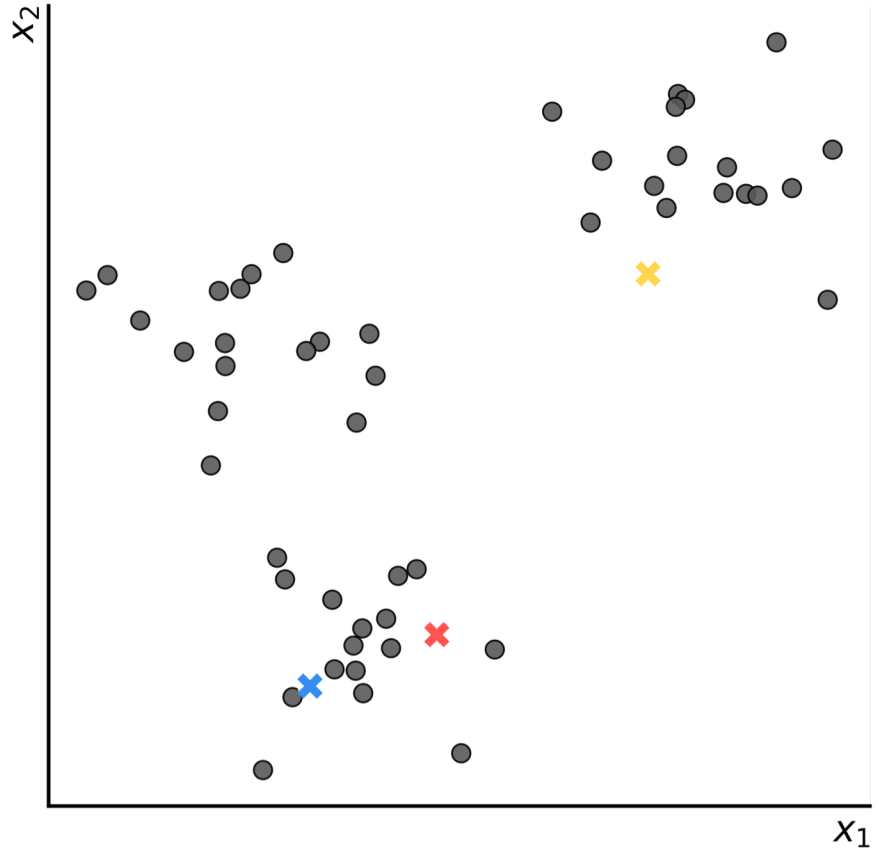
2) Choose the **initial k centroids** (cluster centers)

- random instances from the dataset (Forgy initialization); or
- random positions in the input feature space; or
- pre-defined k positions; or
- positions resulting from some optimization technique;

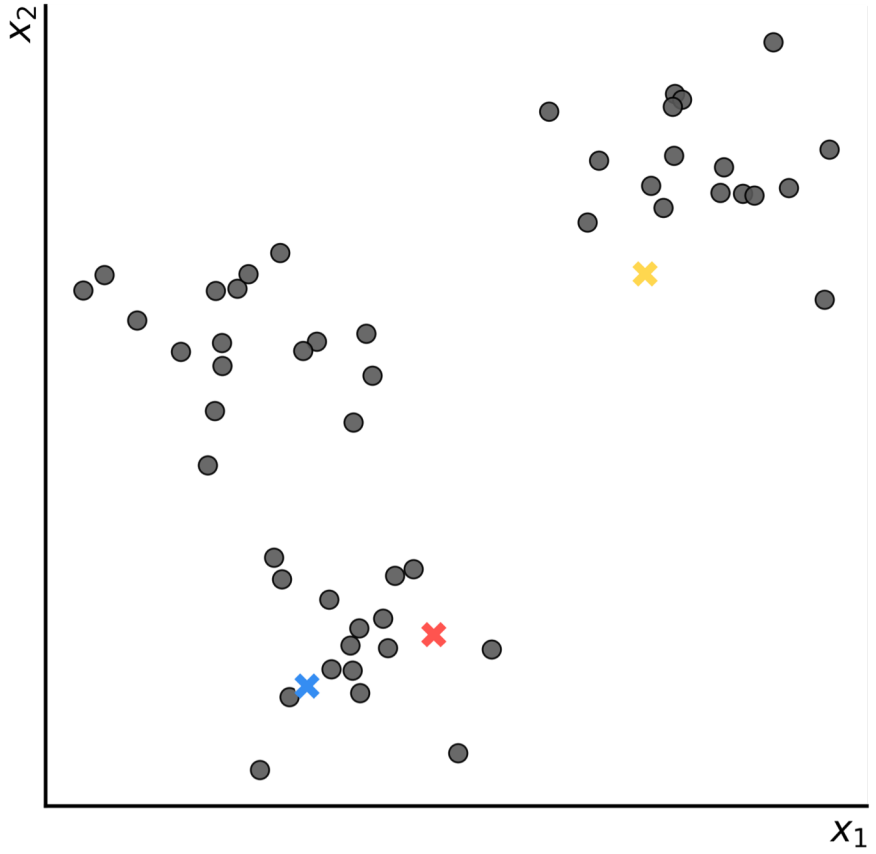


2) Choose the **initial k centroids** (cluster centers)

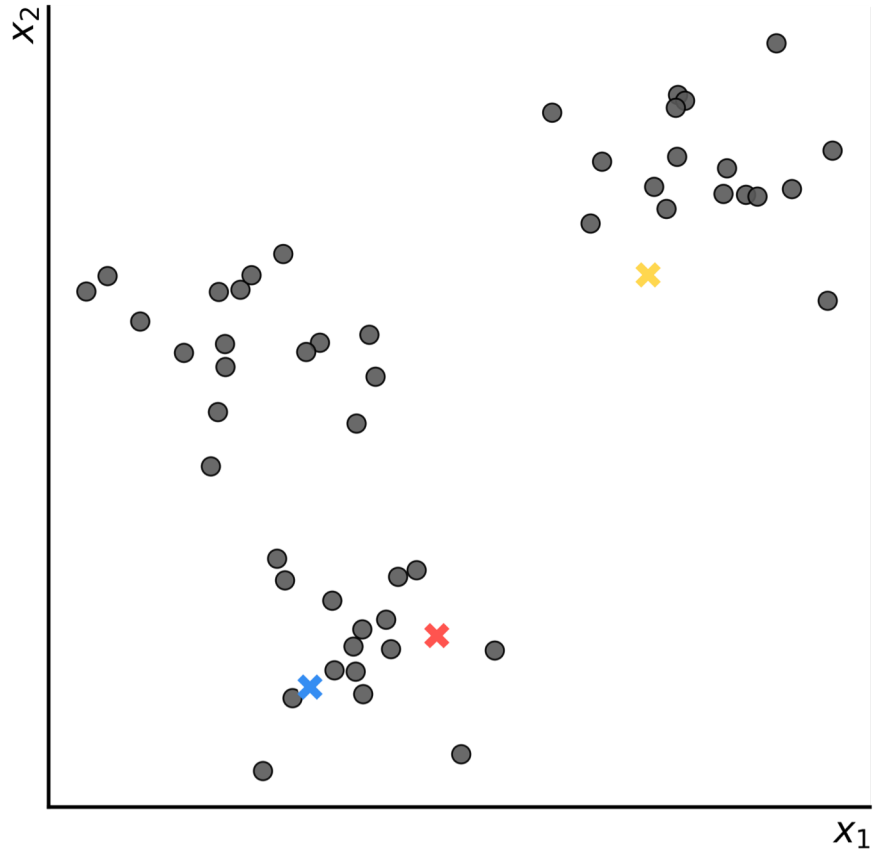
- random instances from the dataset (Forgy initialization); or
- random positions in the input feature space; or
- pre-defined k positions; or
- positions resulting from some optimization technique;



2) Choose the **initial k centroids** (cluster centers)

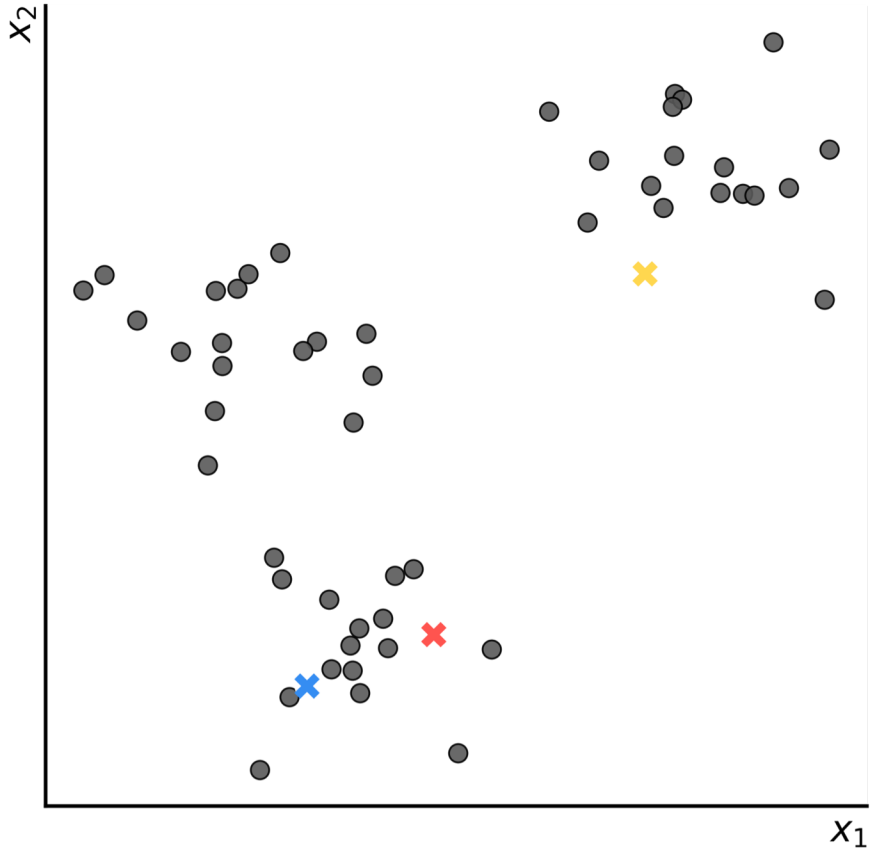


3) Repeat until **convergence**:



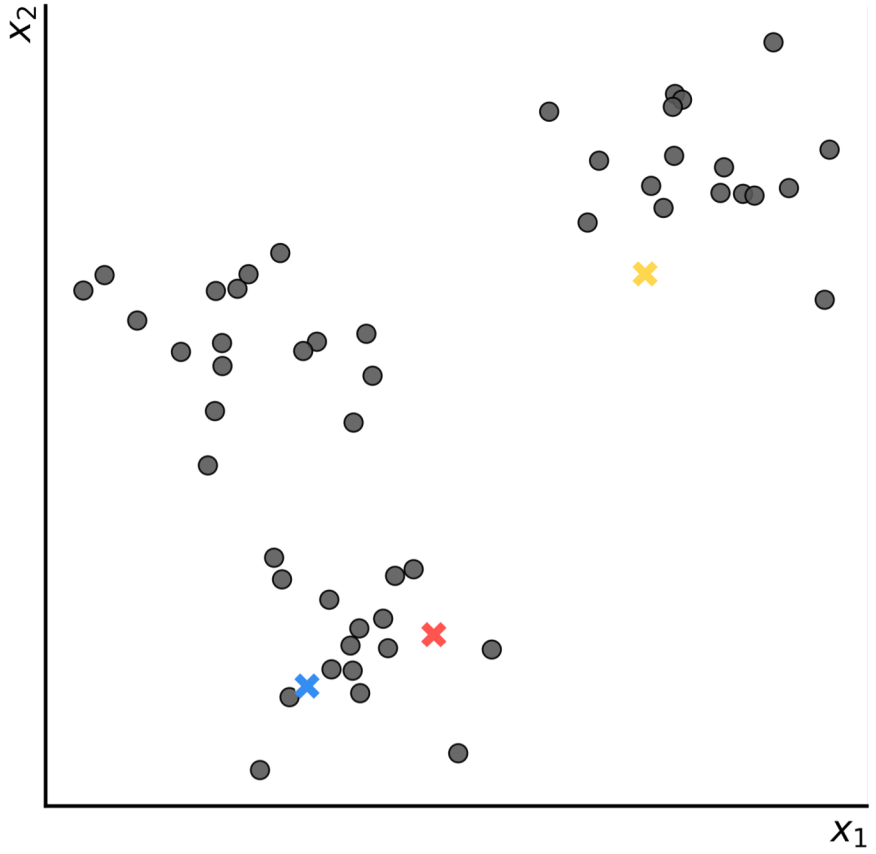
3) Repeat until **convergence**:

- No reassigment occurrs; and/or
- Centroids do not move significantly; and/or
- Maximum number of iterations reached;

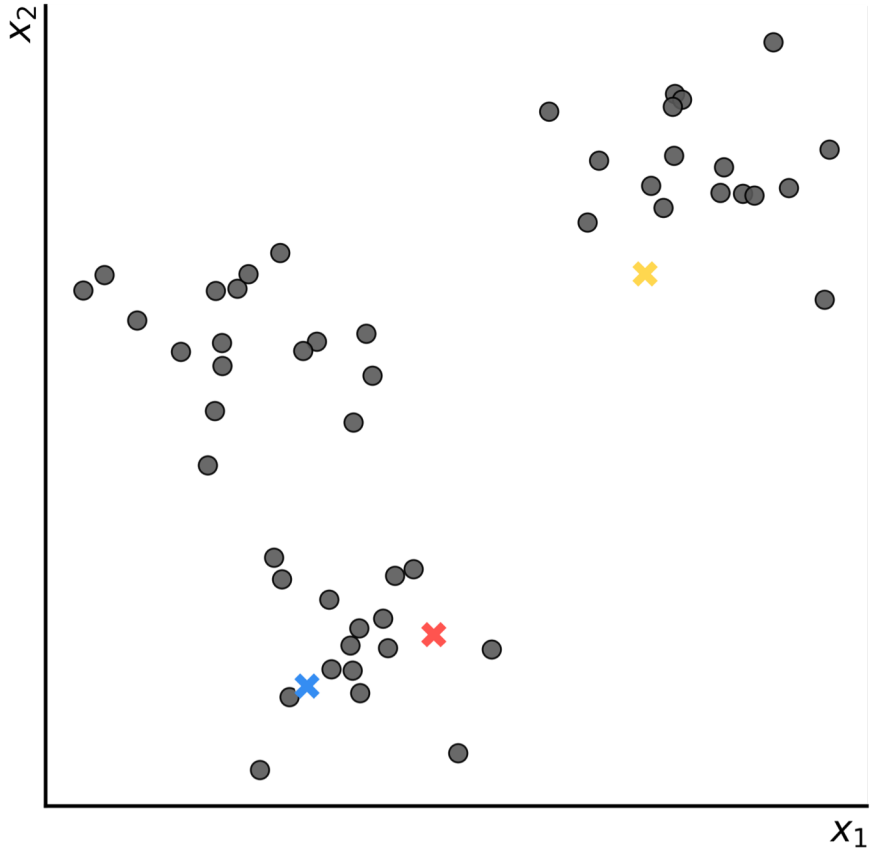


3) Repeat until **convergence**:

- No reassigment occurrs; and/or
- Centroids do not move significantly; and/or
- Maximum number of iterations reached;



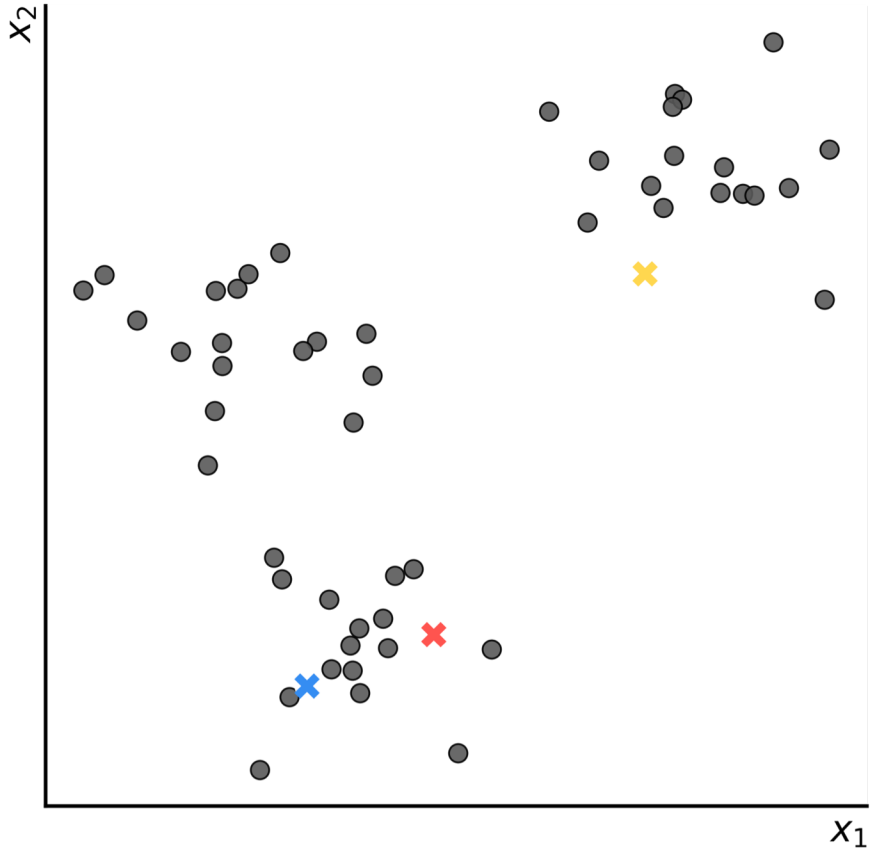
3) Repeat until **centroids do not move significantly**:



3) Repeat until **centroids do not move significantly**:

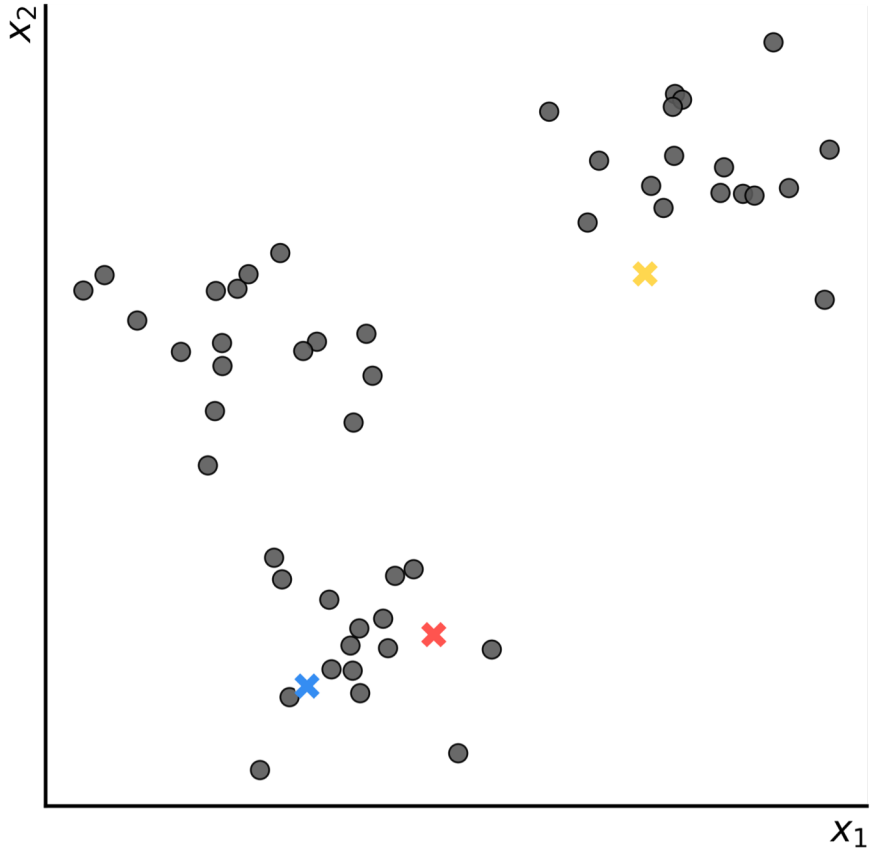
iteration 1

- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



4) Assign each instance to **the nearest** centroid:

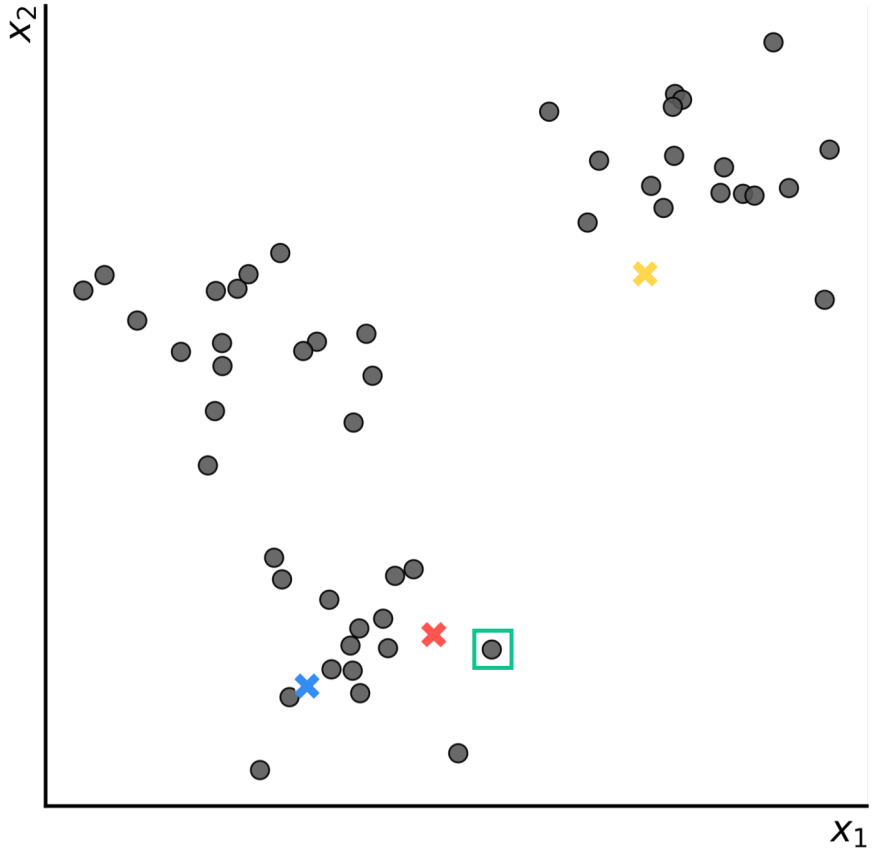
iteration 1



4) Assign each instance to **the nearest** centroid:

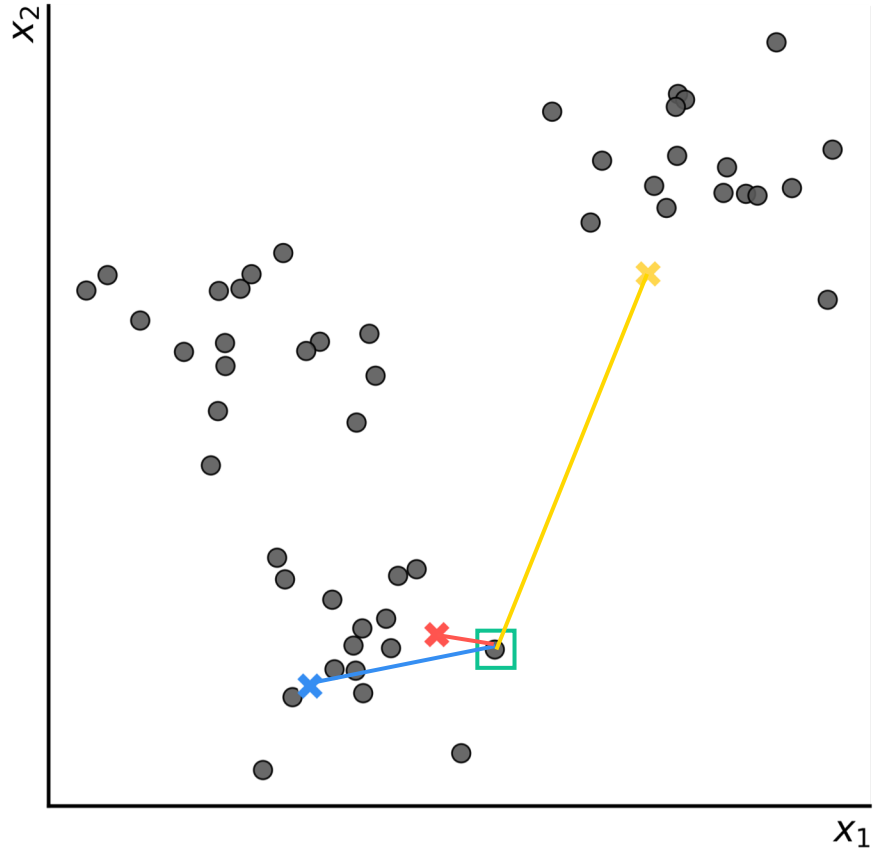
iteration 1

- For each sample $\boldsymbol{x}^{(i)}$:



4) Assign each instance to **the nearest** centroid:

- For each sample $\boldsymbol{x}^{(i)}$:
  - Compute the distance from $\boldsymbol{x}^{(i)}$ to each centroid $\boldsymbol{c}^{(j)}$;

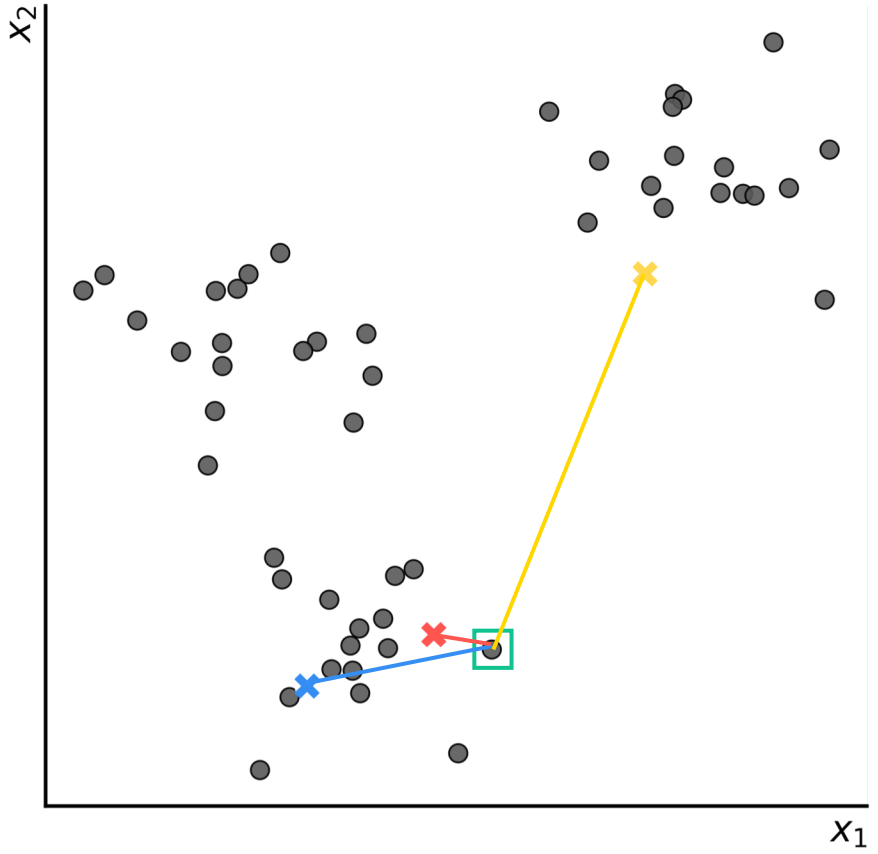


Euclidean distance is the common metric (default) used in KMeans.

iteration 1

4) Assign each instance to **the nearest** centroid:

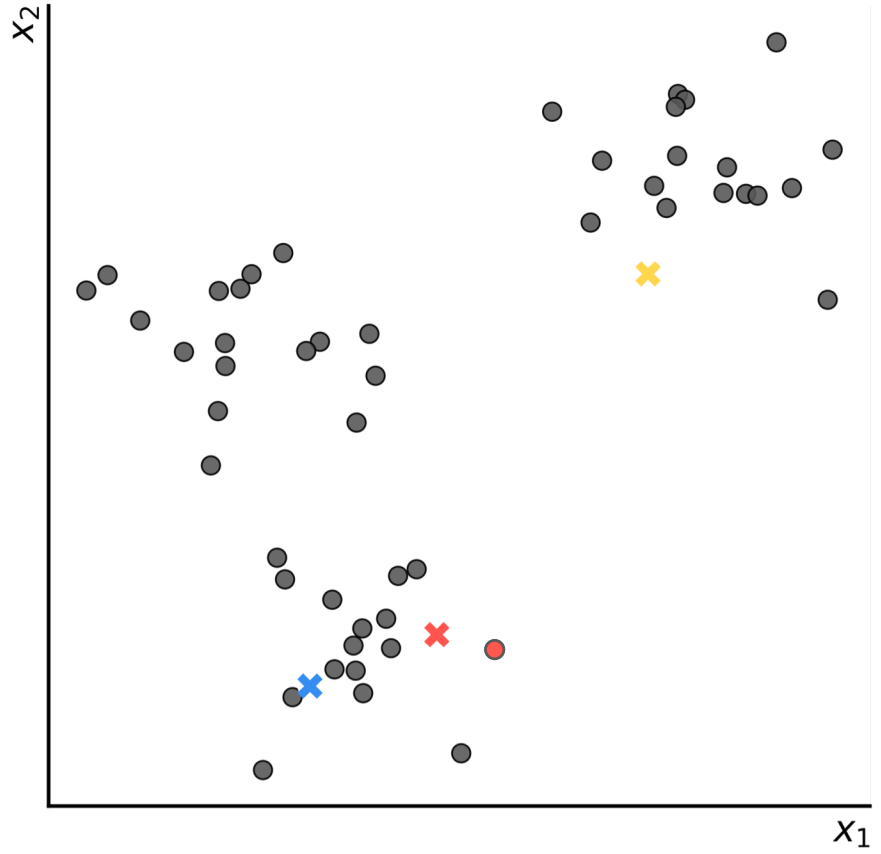
- For each sample $\boldsymbol{x}^{(i)}$:
  - Compute the distance from $\boldsymbol{x}^{(i)}$ to each centroid $\boldsymbol{c}^{(j)}$;
  - Assign the sample to the nearest cluster center with distance $\boldsymbol{d}^{(i)} = \min_{j} \boldsymbol{d}(\boldsymbol{x}^{(i)}, \boldsymbol{c}^{(j)})$



4) Assign each instance to **the nearest** centroid:

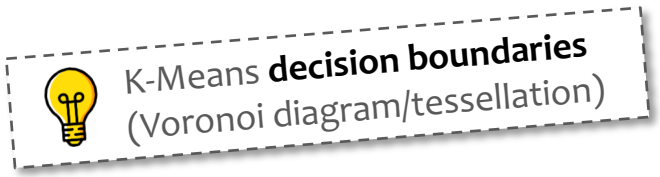
iteration 1

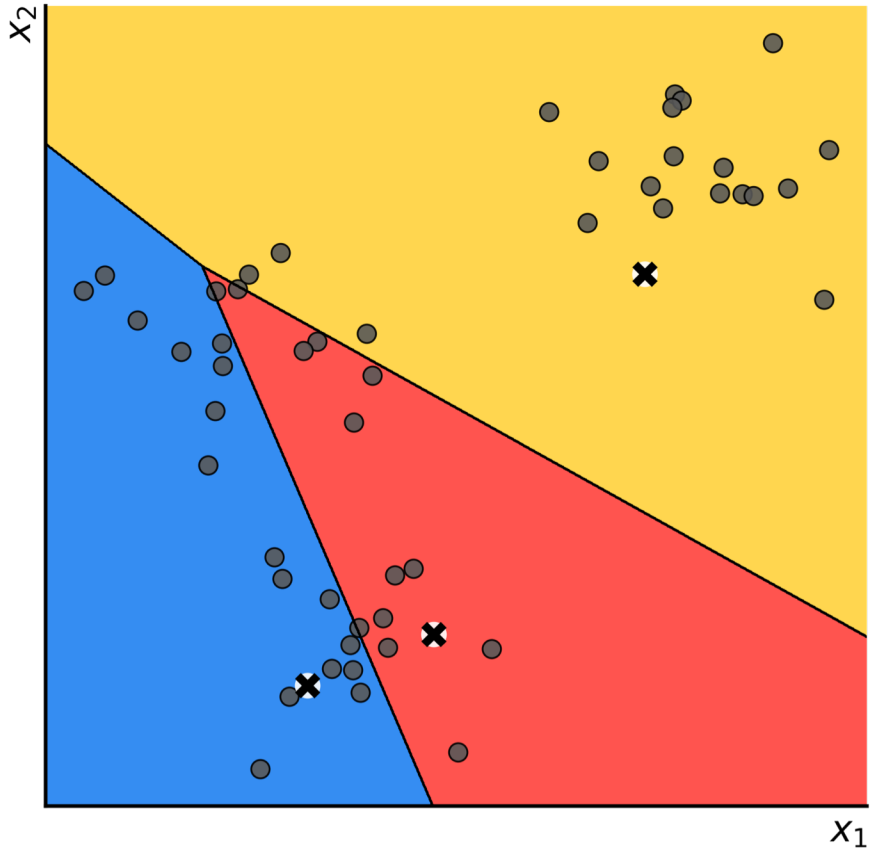
- For each sample $\boldsymbol{x}^{(i)}$:
  - Compute the distance from $\boldsymbol{x}^{(i)}$ to each centroid $\boldsymbol{c}^{(j)}$;
  - Assign the sample to the nearest cluster center with distance $\boldsymbol{d}^{(i)} = \min_{j} \boldsymbol{d}(\boldsymbol{x}^{(i)}, \boldsymbol{c}^{(j)})$



4) Assign each instance to **the nearest** centroid:

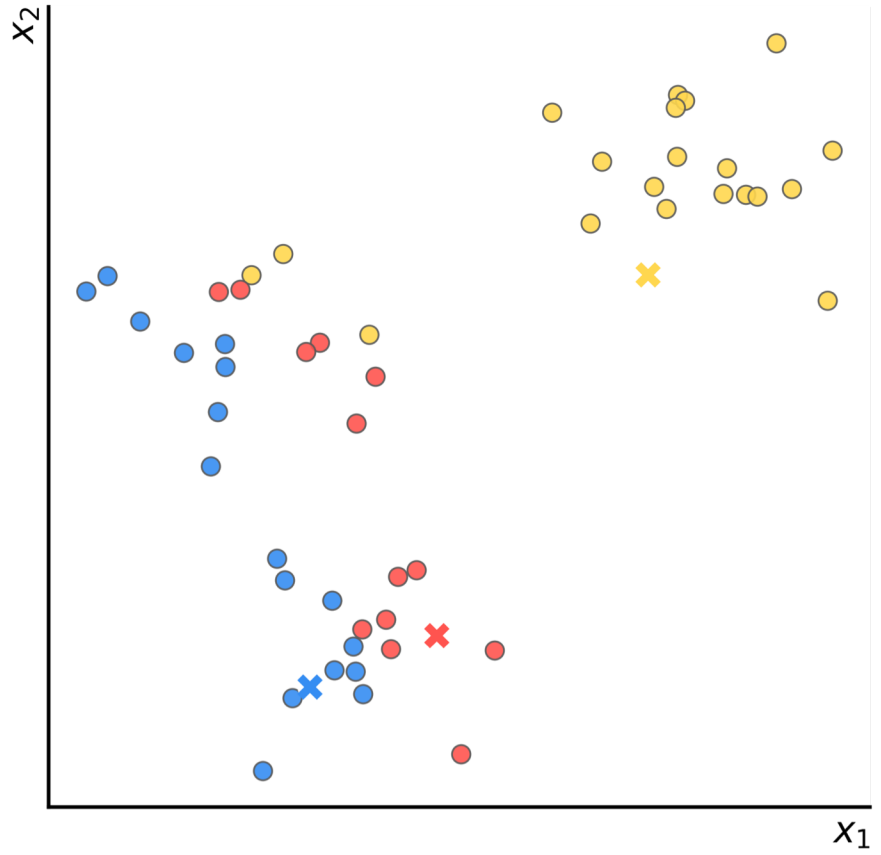
- For each sample $\boldsymbol{x}^{(i)}$:
  - Compute the distance from $\boldsymbol{x}^{(i)}$ to each centroid $\boldsymbol{c}^{(j)}$;
  - Assign the sample to the nearest cluster center with distance $\boldsymbol{d}^{(i)} = \min_{j} \boldsymbol{d}(\boldsymbol{x}^{(i)}, \boldsymbol{c}^{(j)})$





5) Move the centroids to the centers of their clusters

iteration 1

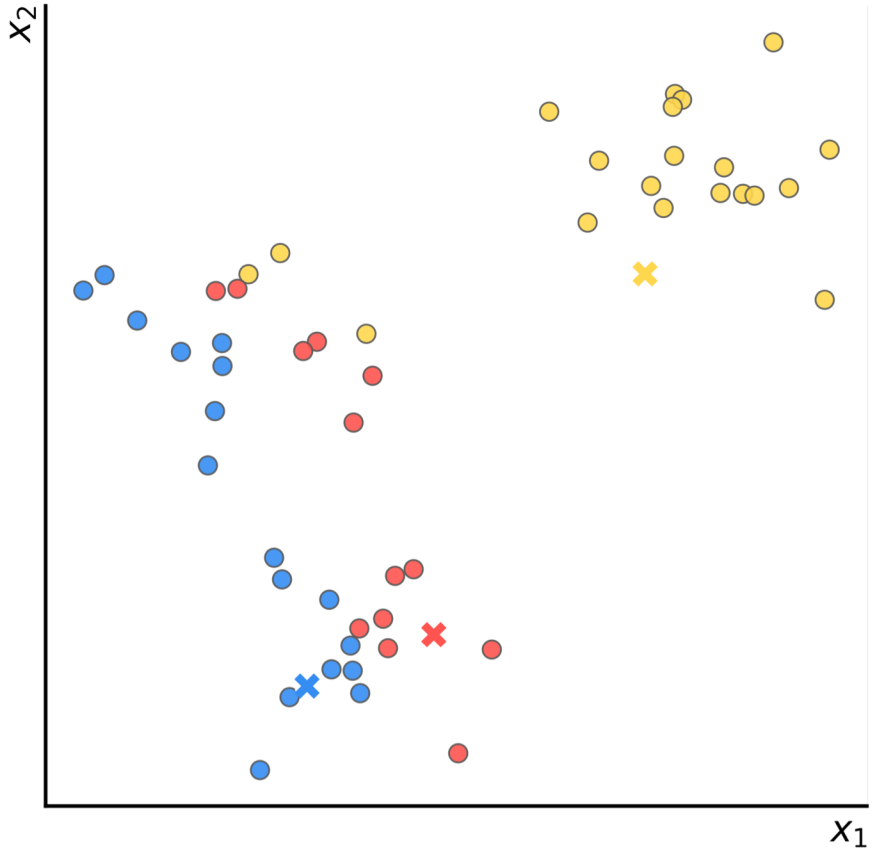


5) Move the centroids to the centers of their clusters

iteration 1

- For each cluster $c^{(j)} = [c_1^{(j)}, c_2^{(j)}, \cdots]$:
  - $c^{(j)} = \frac{1}{m^{(j)}} \sum_{i=1}^{m^{(j)}} x^{(j)}$

number of samples in **cluster j**

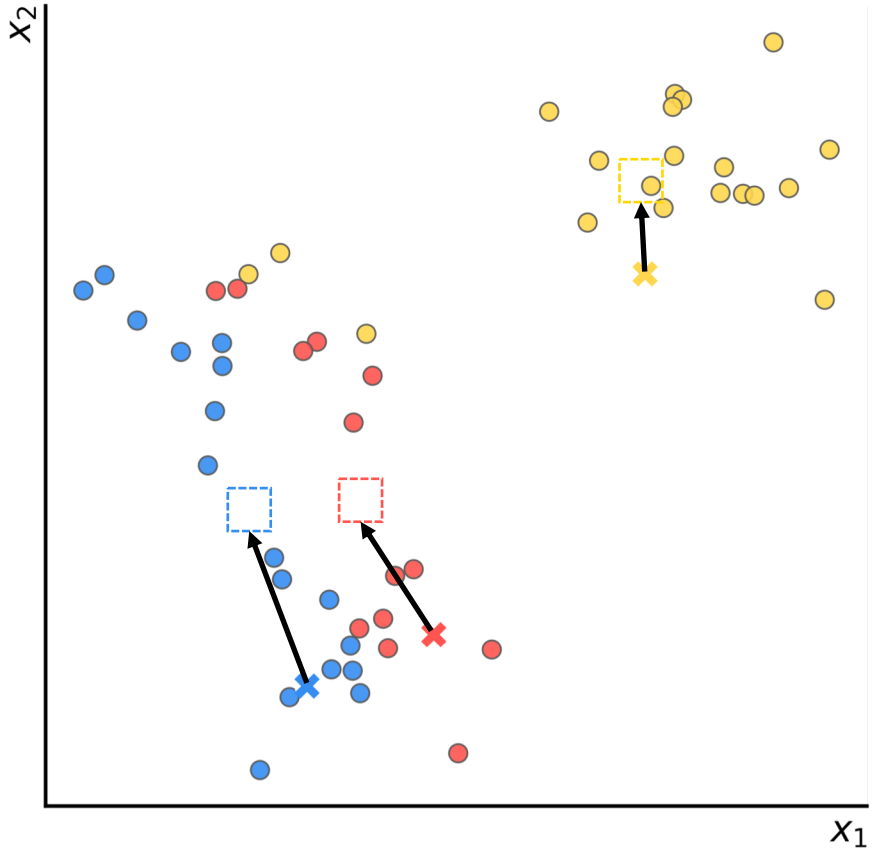


5) Move the centroids to the centers of their clusters

iteration 1

- For each cluster $c^{(j)} = [c_1^{(j)}, c_2^{(j)}, \cdots]$:
  - $c^{(j)} = \frac{1}{m^{(j)}} \sum_{i=1}^{m^{(j)}} x^{(j)}$

number of samples in **cluster j**

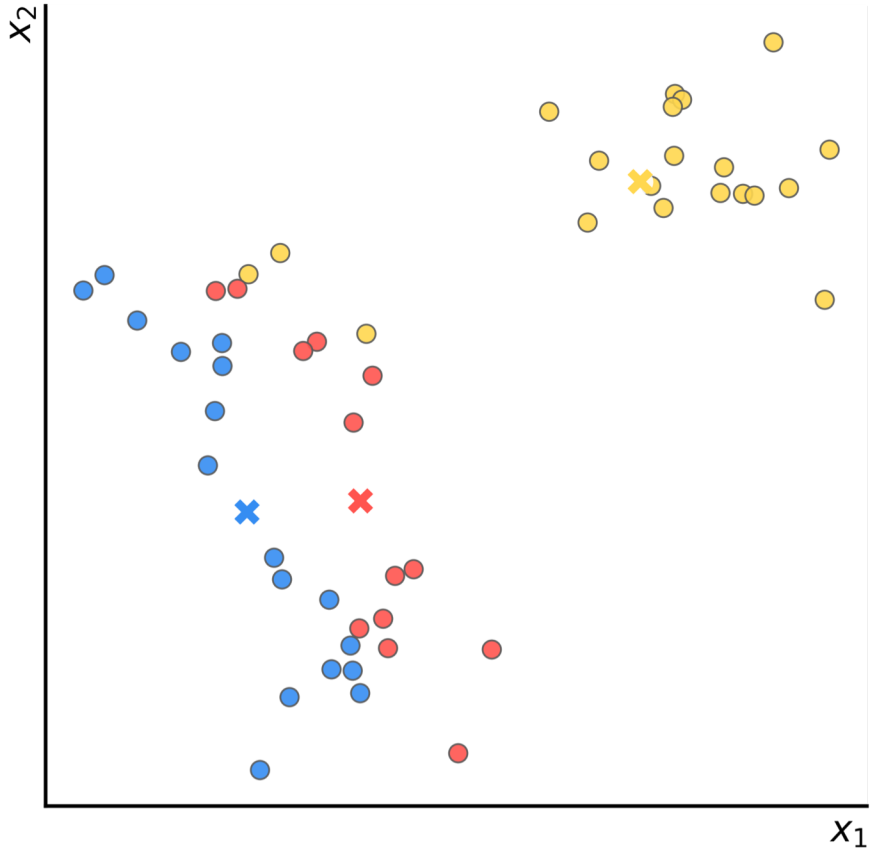


5) Move the centroids to the centers of their clusters

iteration 1

- For each cluster $c^{(j)} = [c_1^{(j)}, c_2^{(j)}, \cdots]$:
  - $c^{(j)} = \frac{1}{m^{(j)}} \sum_{i=1}^{m^{(j)}} x^{(j)}$

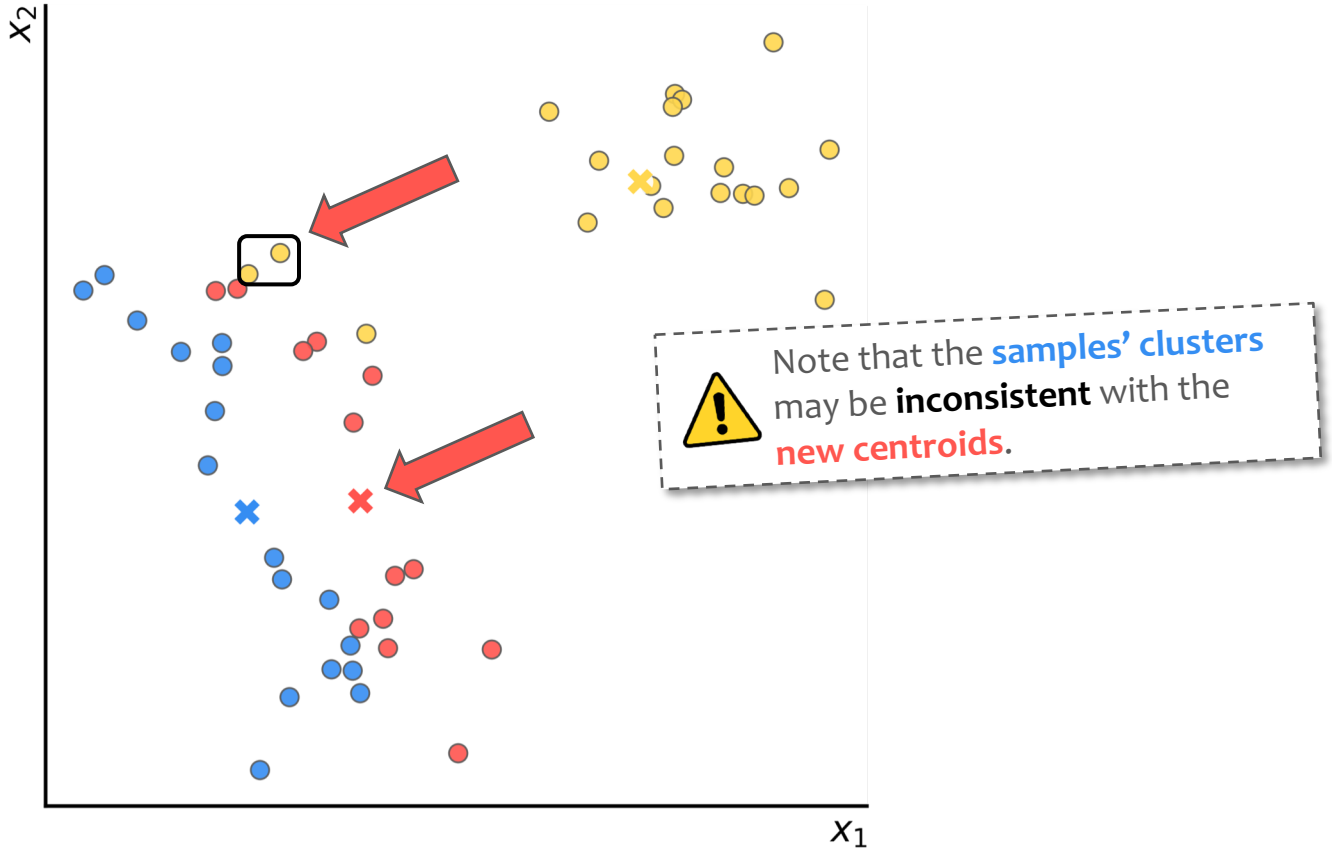
number of samples in **cluster j**



5) Move the centroids to the centers of their clusters

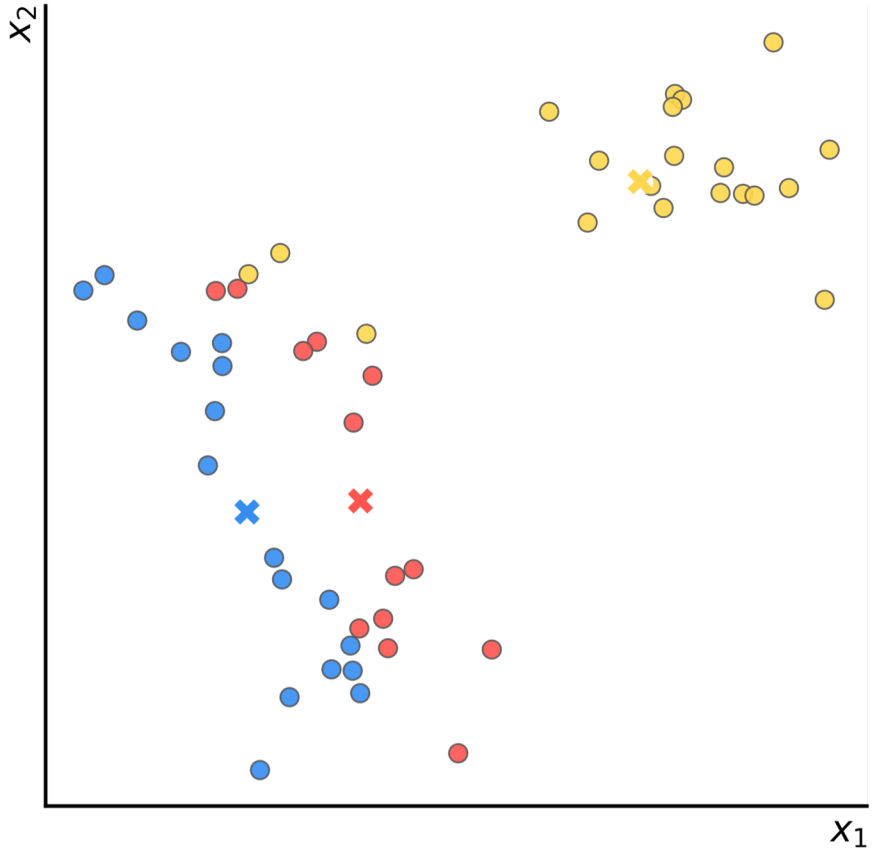
- For each cluster $c^{(j)} = [c_1^{(j)}, c_2^{(j)}, \cdots]$:
  - $c^{(j)} = \frac{1}{m^{(j)}} \sum_{i=1}^{m^{(j)}} x^{(j)}$

number of samples in **cluster j**



3) Repeat until **centroids do not move significantly**:

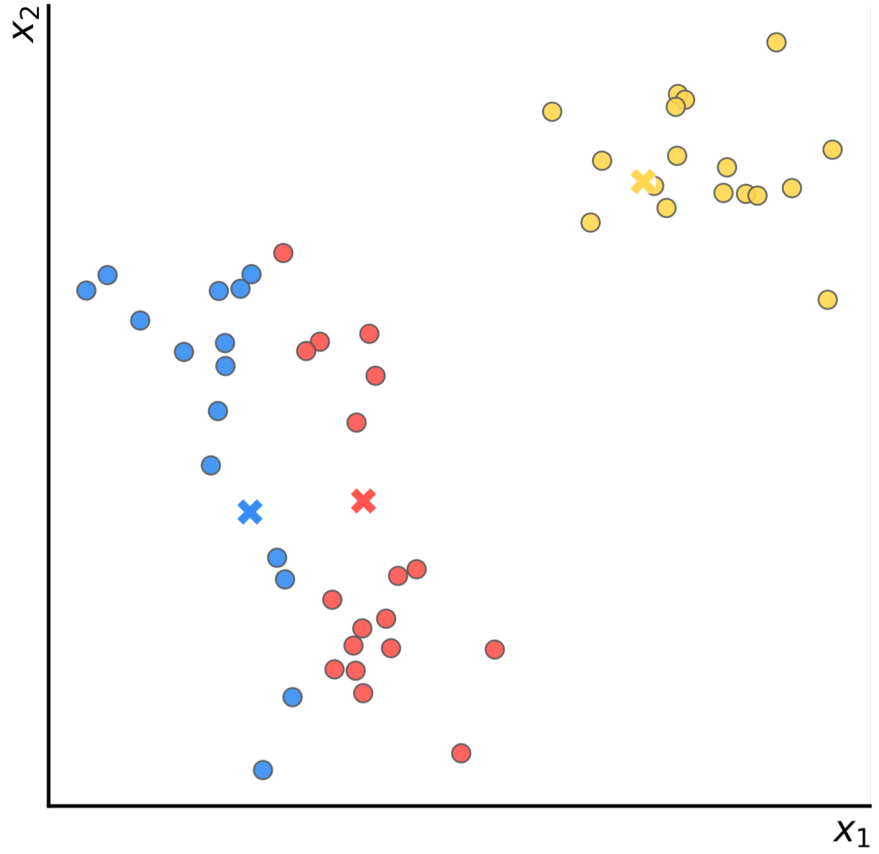
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 2

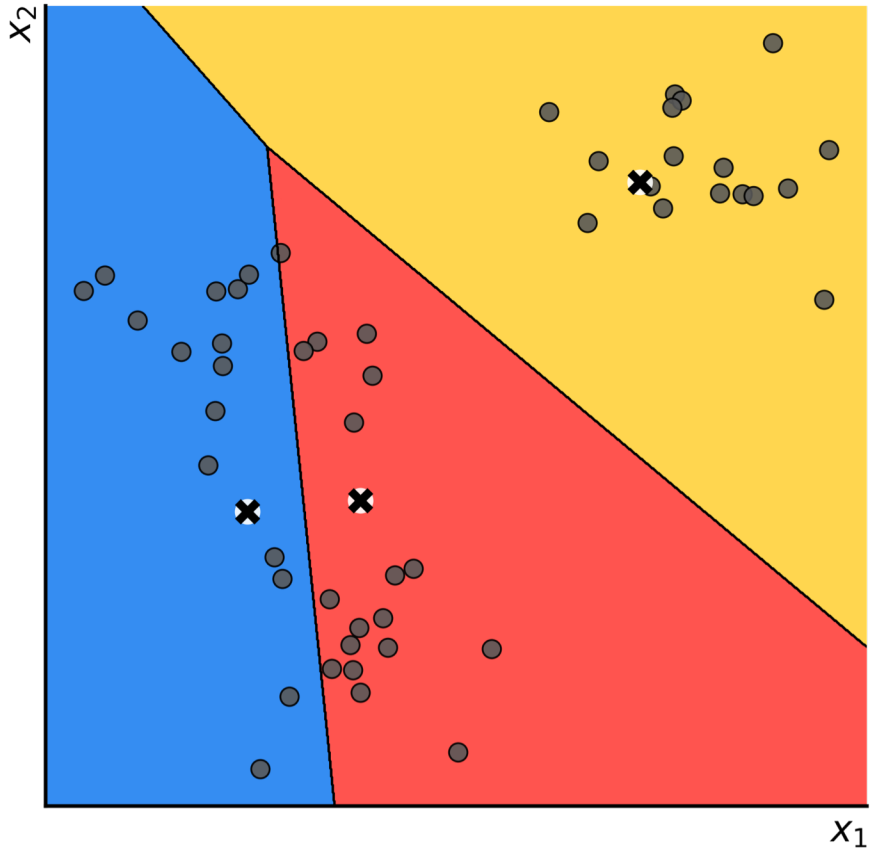
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 2

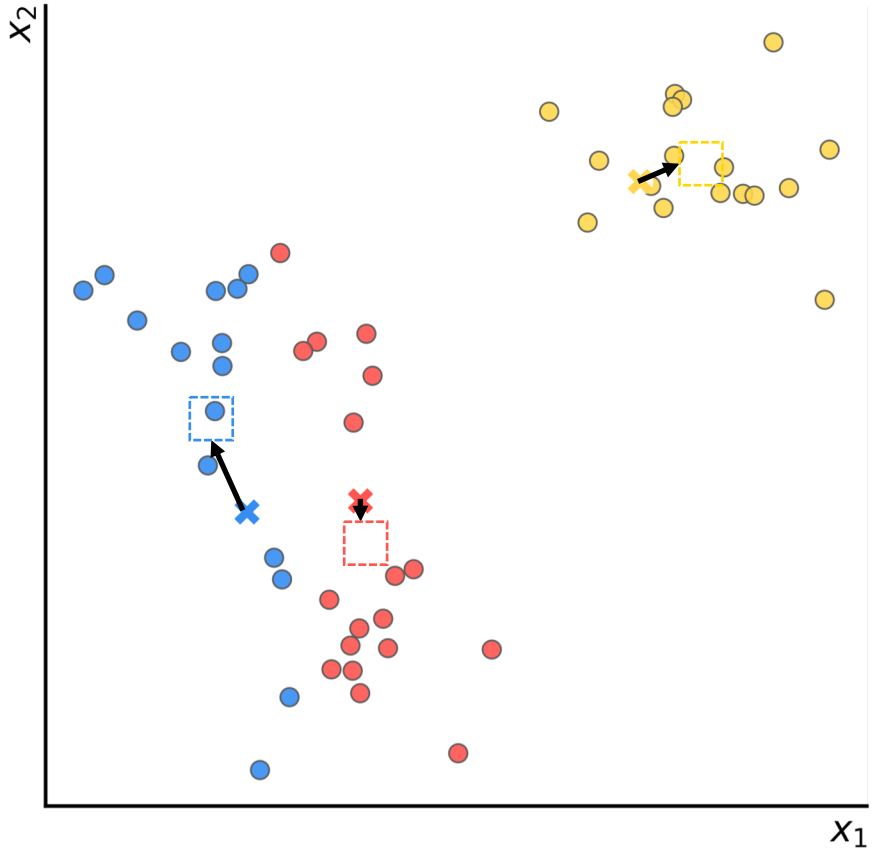
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 2

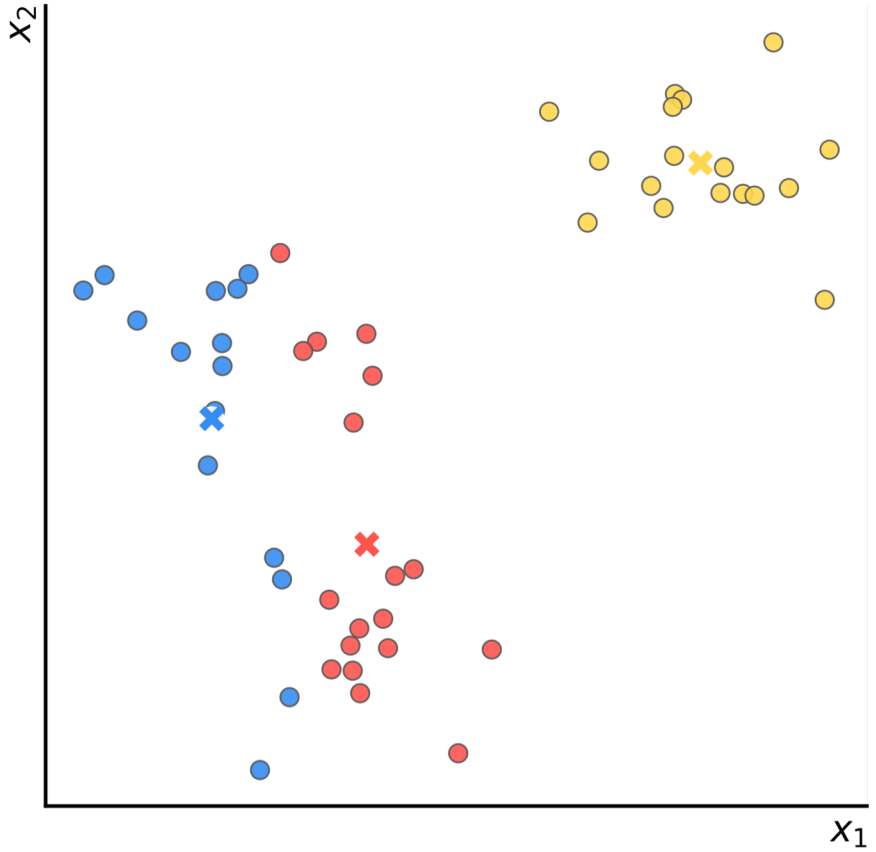
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 2

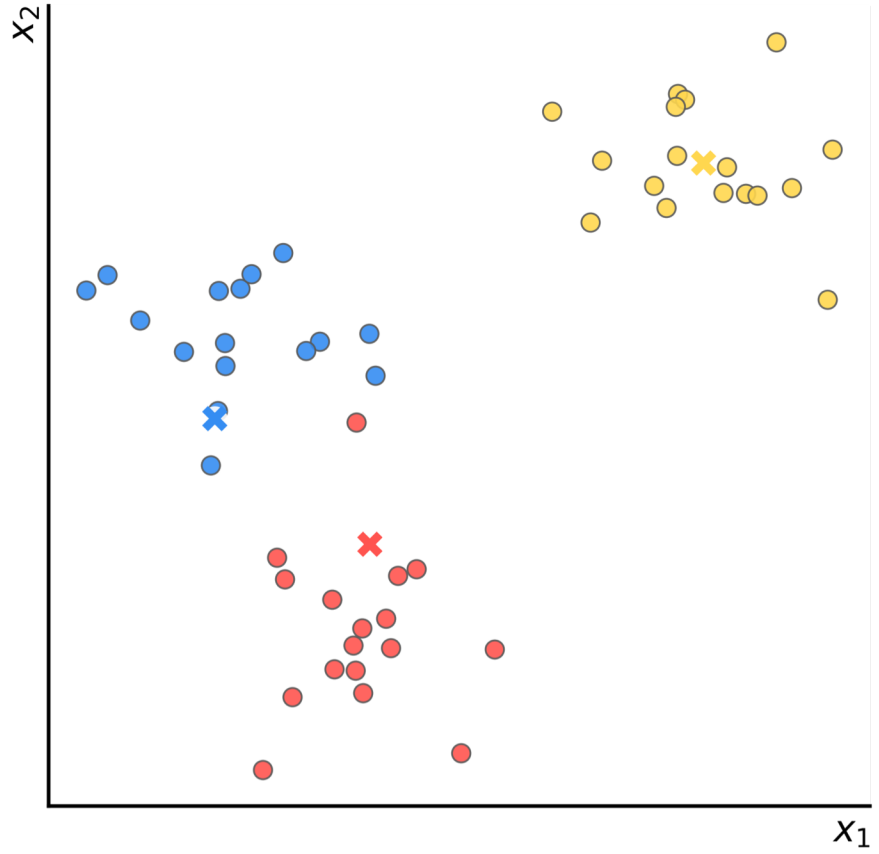
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



Note that the **samples' clusters** may be **inconsistent** with the **new centroids**.

3) Repeat until **centroids do not move significantly**:

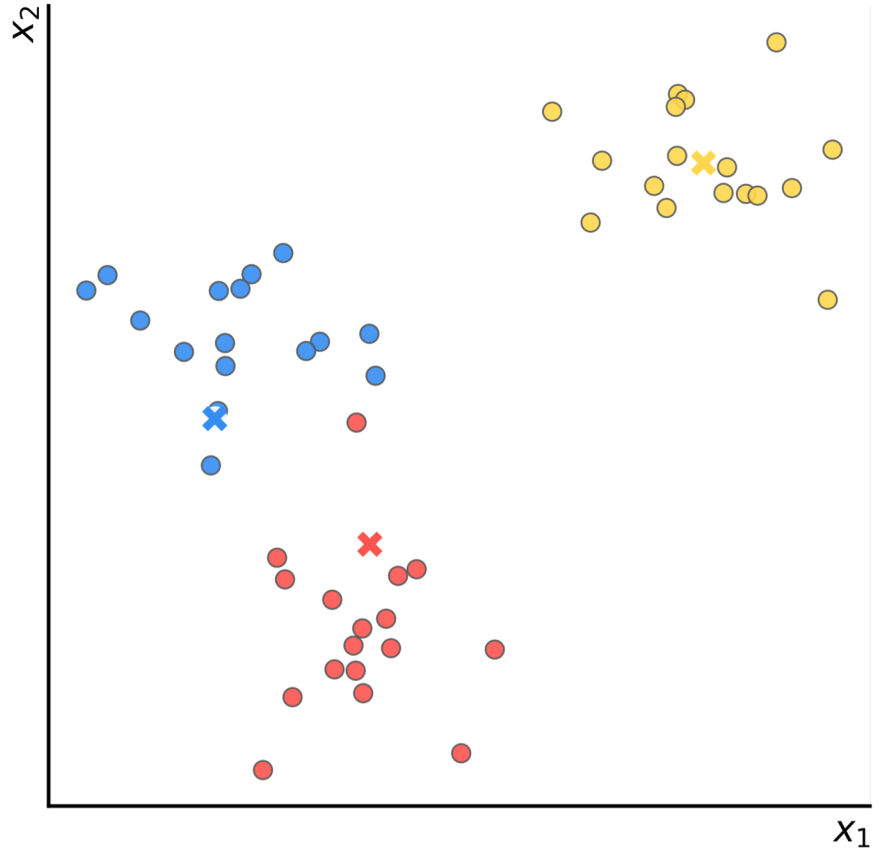
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 3

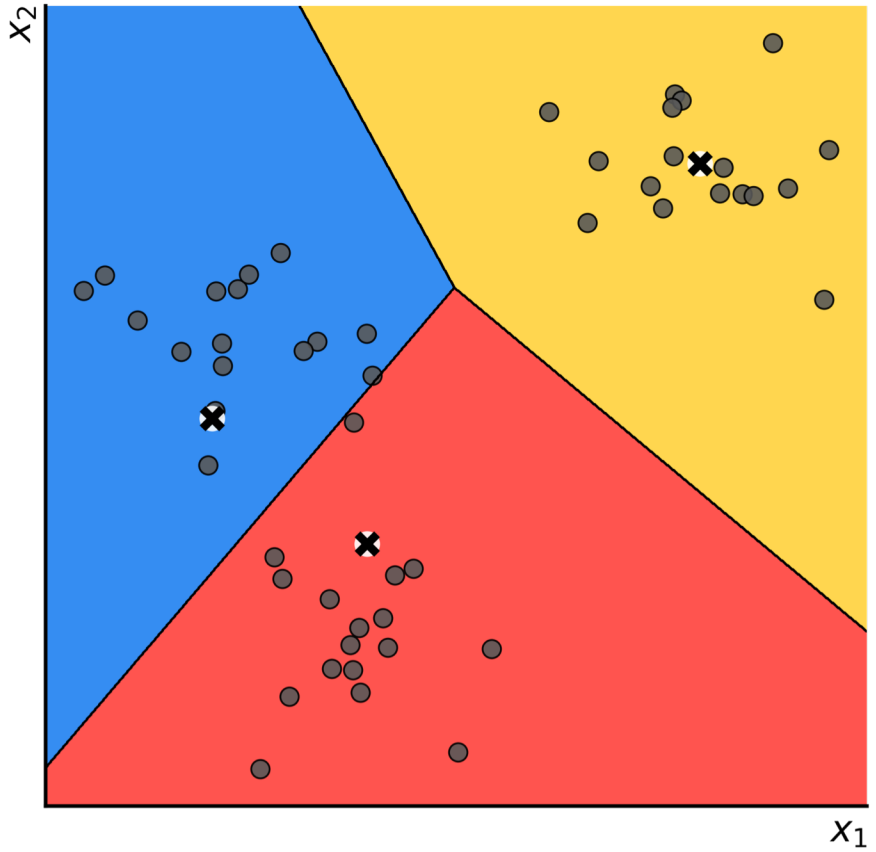
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 3

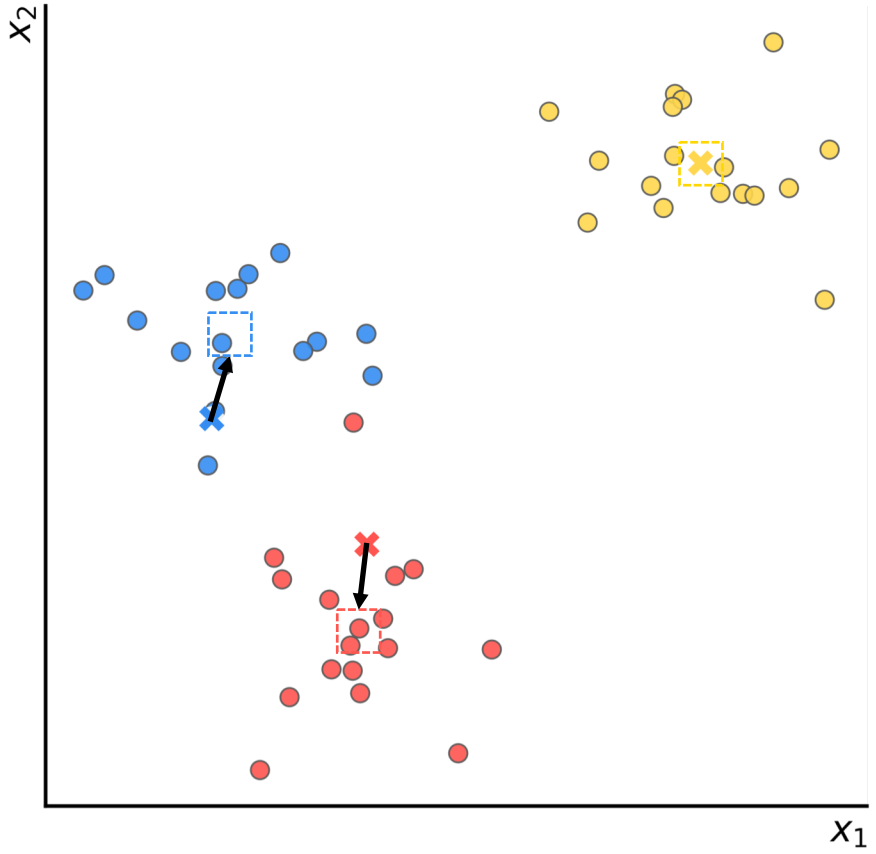
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 3

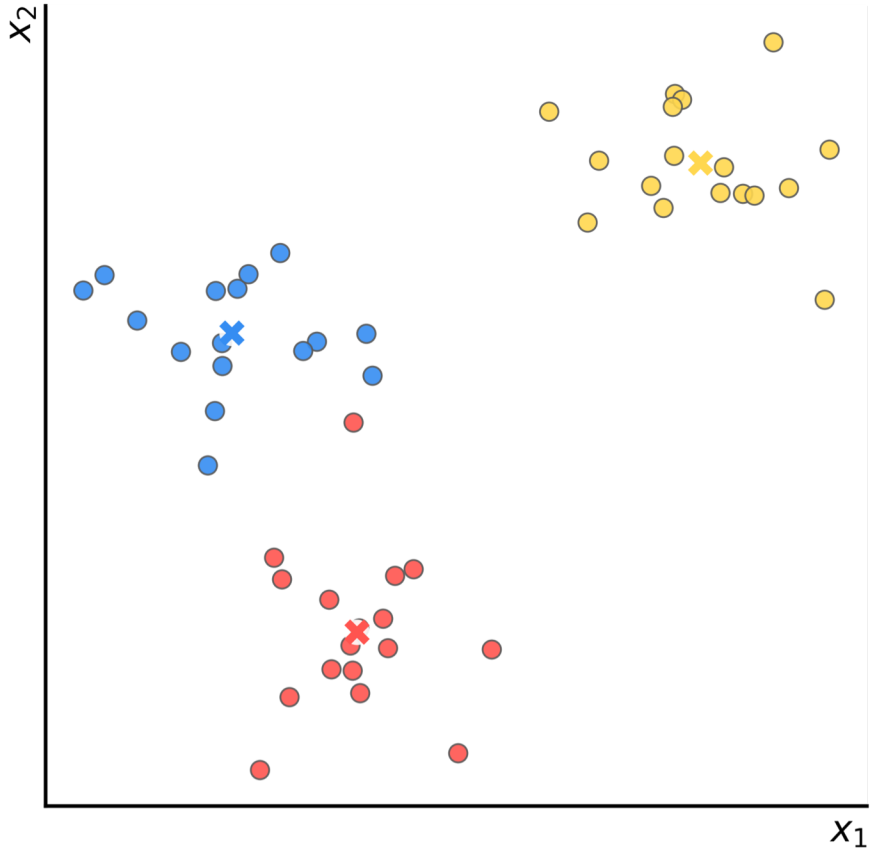
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 3

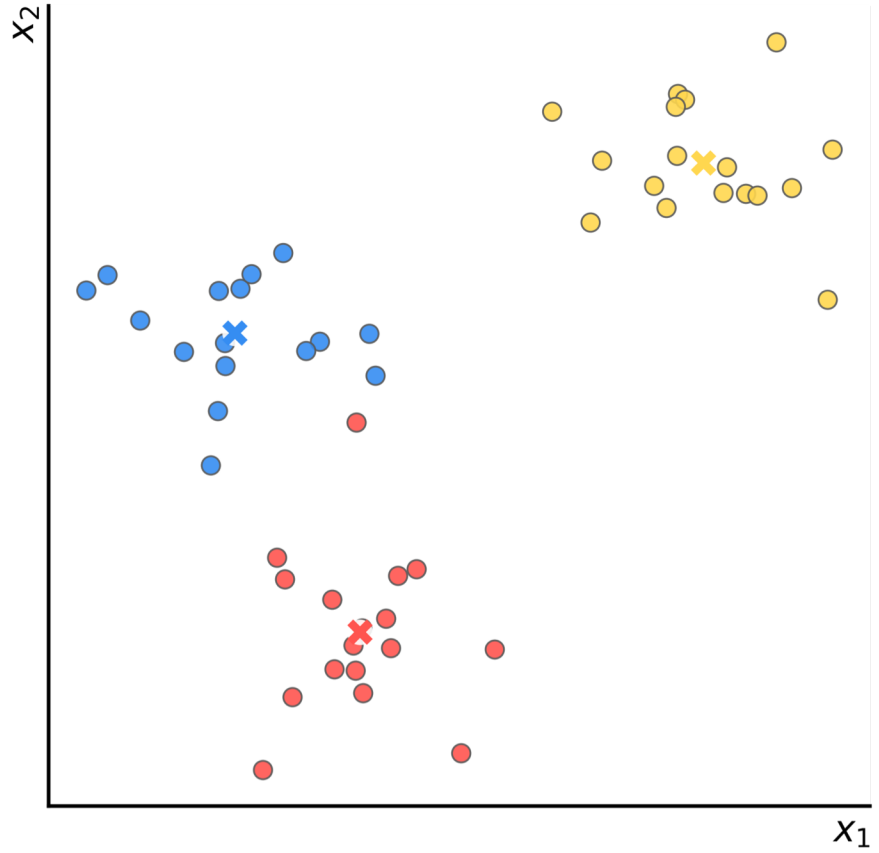
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



Note that the **samples' clusters** may be **inconsistent** with the **new centroids**.

3) Repeat until **centroids do not move significantly**:

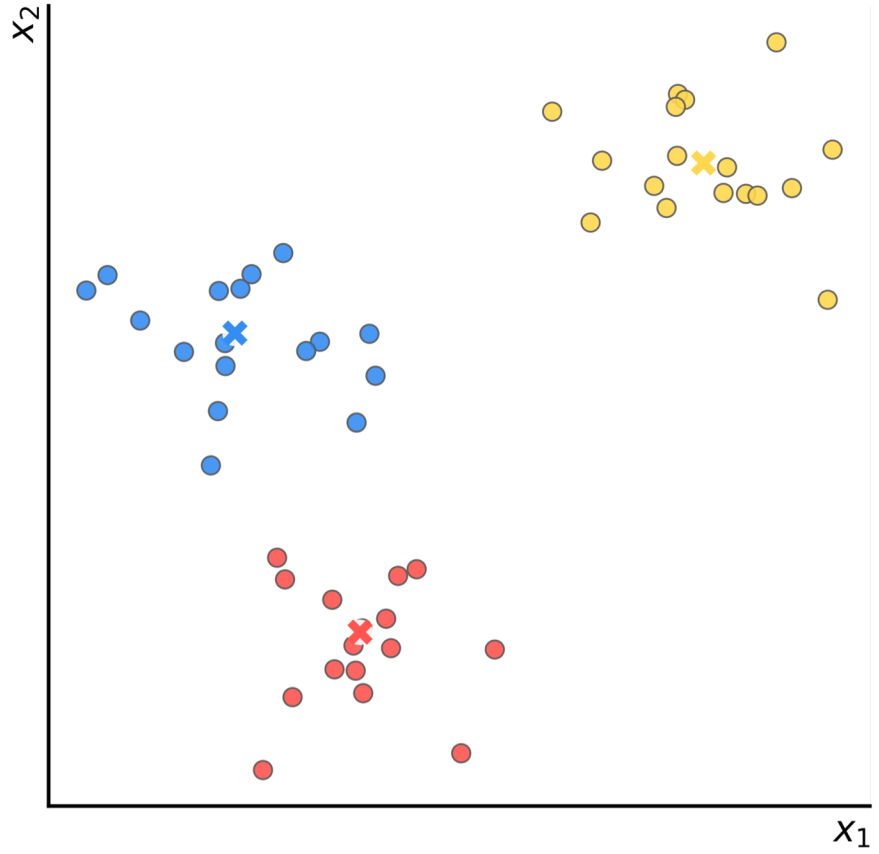
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 4

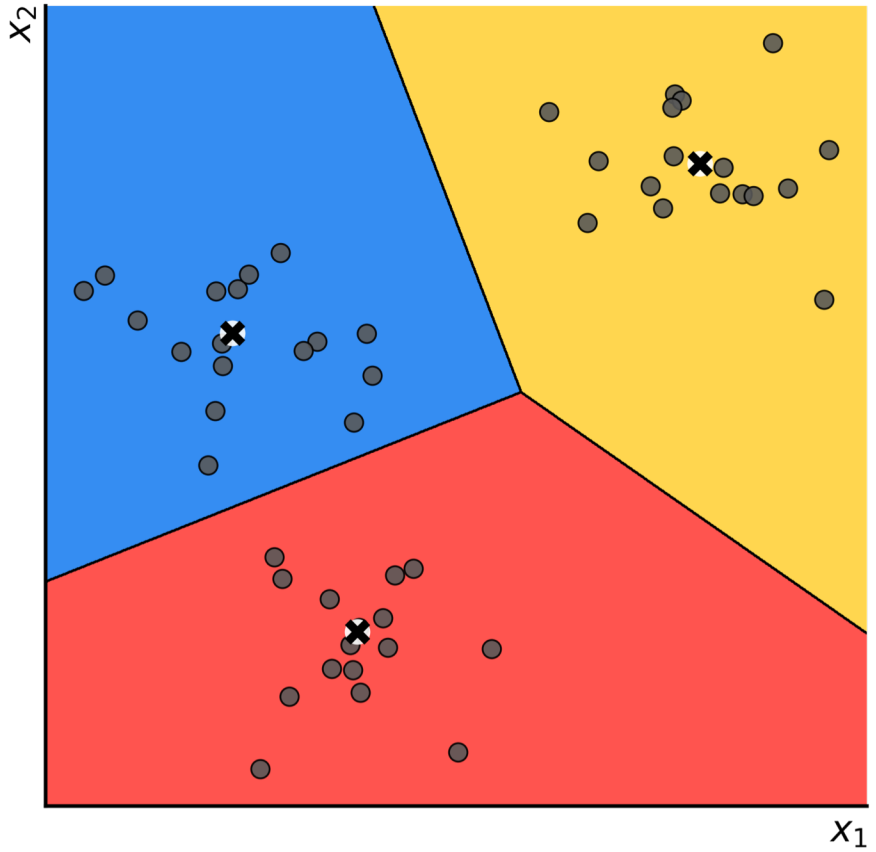
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 4

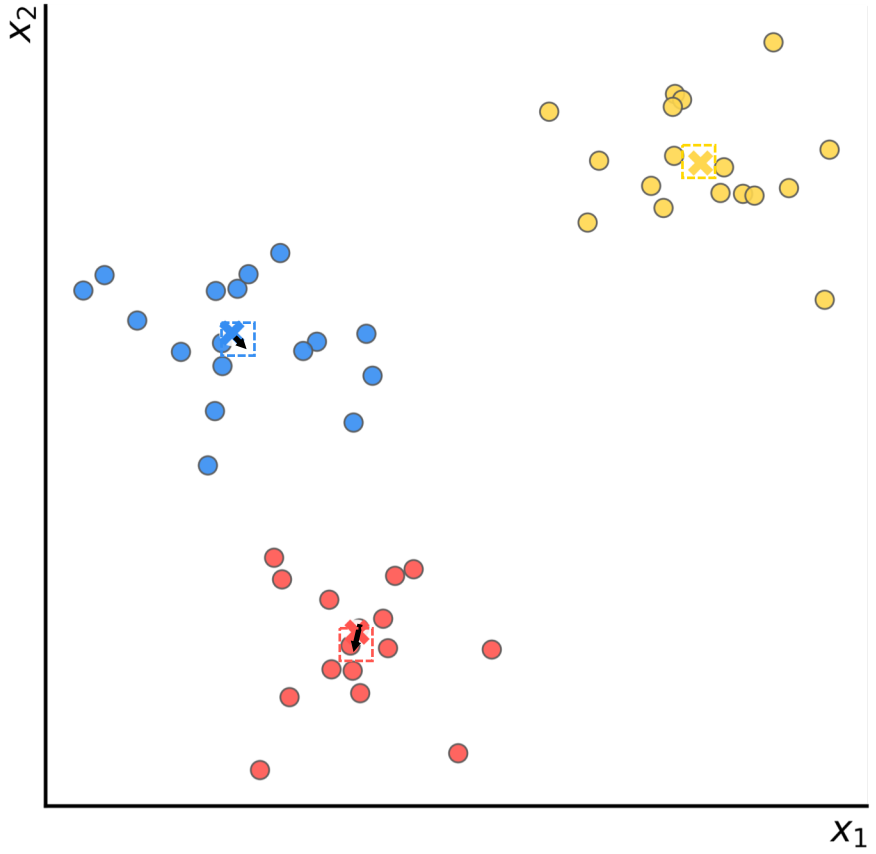
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 4

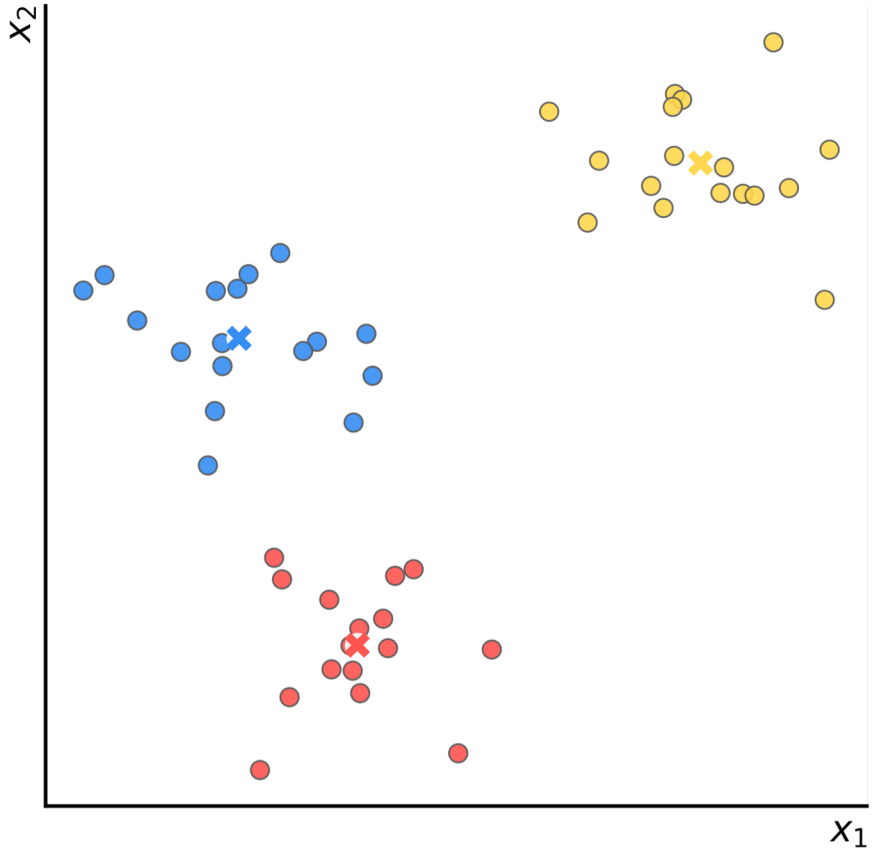
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 4

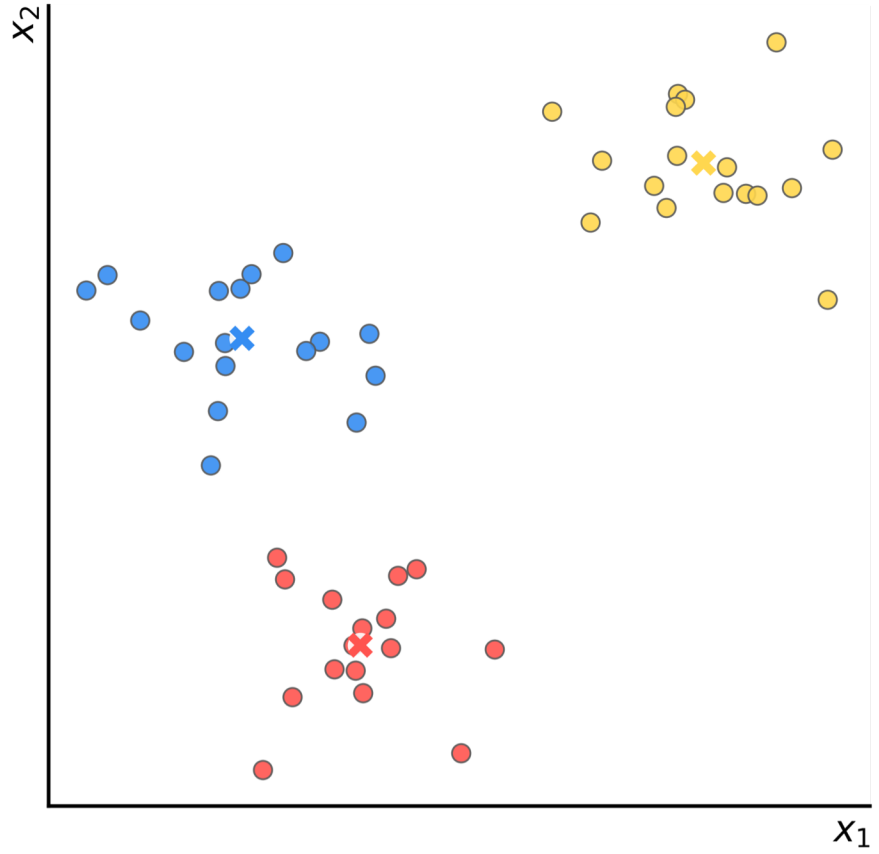
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



Note that the **samples' clusters** may be **inconsistent** with the **new centroids**.

3) Repeat until **centroids do not move significantly**:

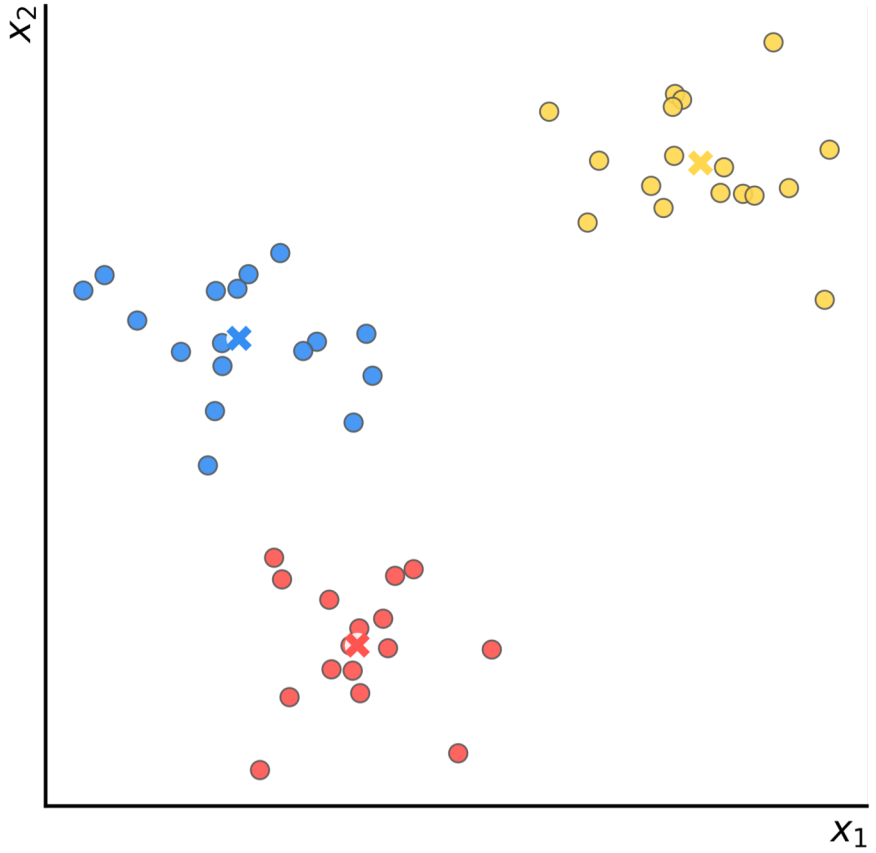
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 5

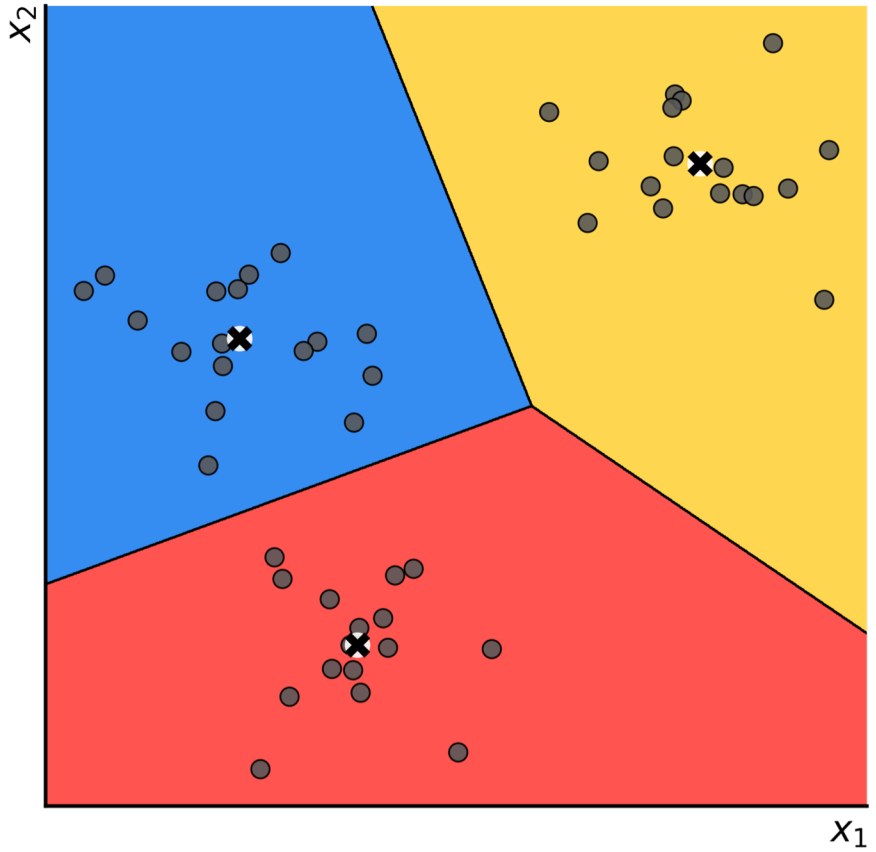
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 5

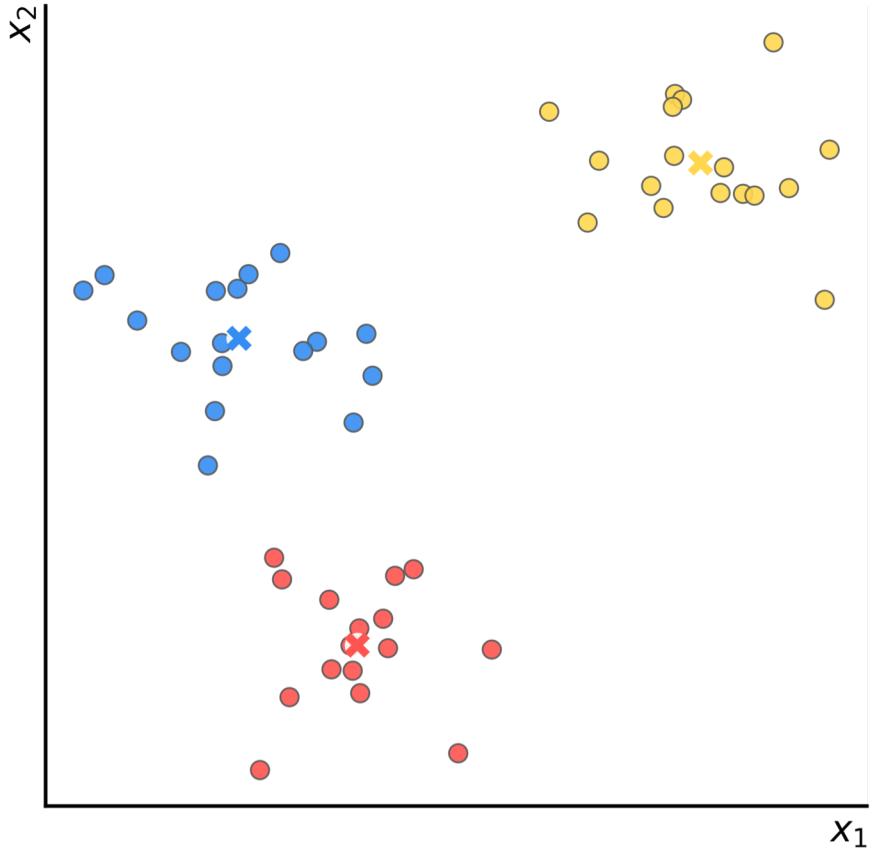
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

iteration 5

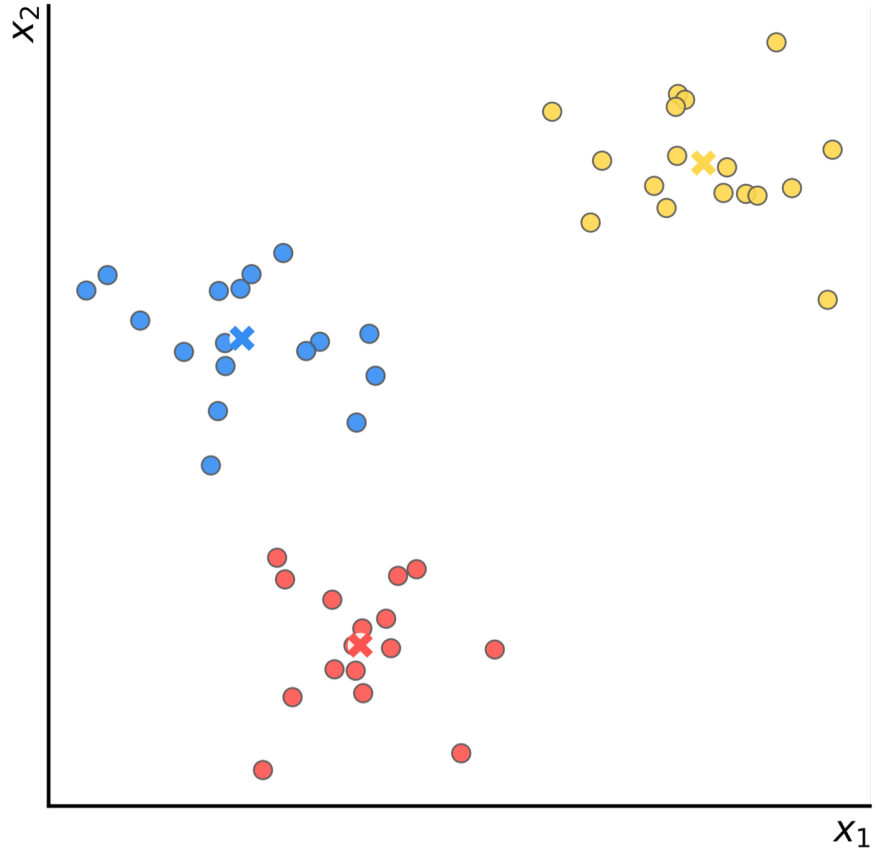
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



no centroid reposition

3) Repeat until **centroids do not move significantly**:

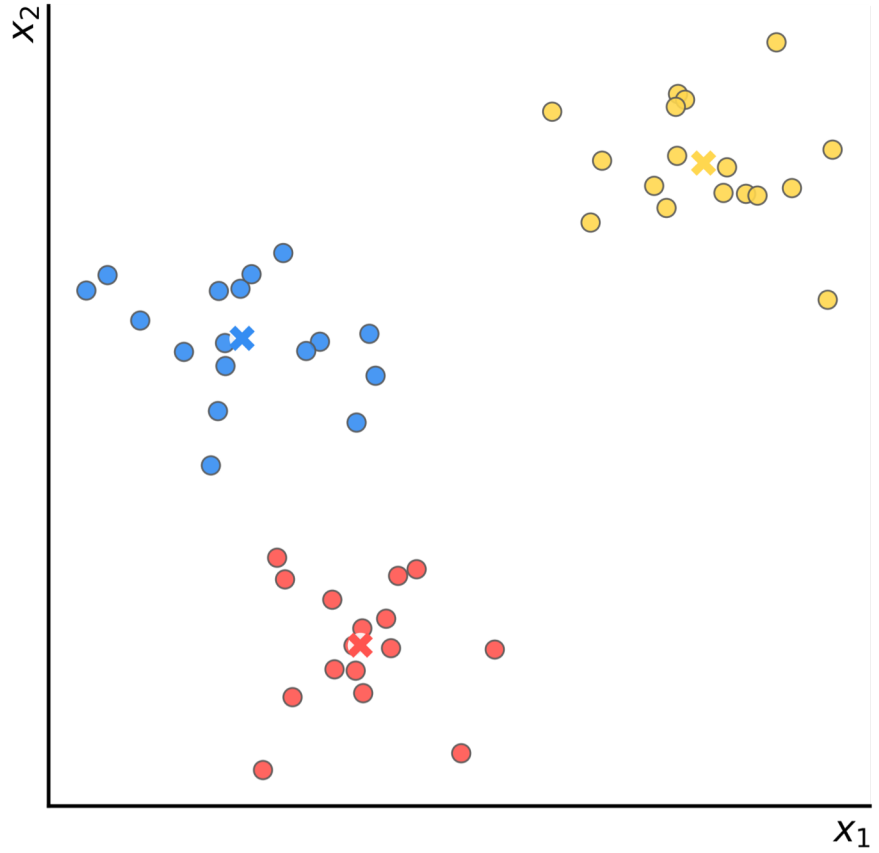
- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



3) Repeat until **centroids do not move significantly**:

- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters

Stop!

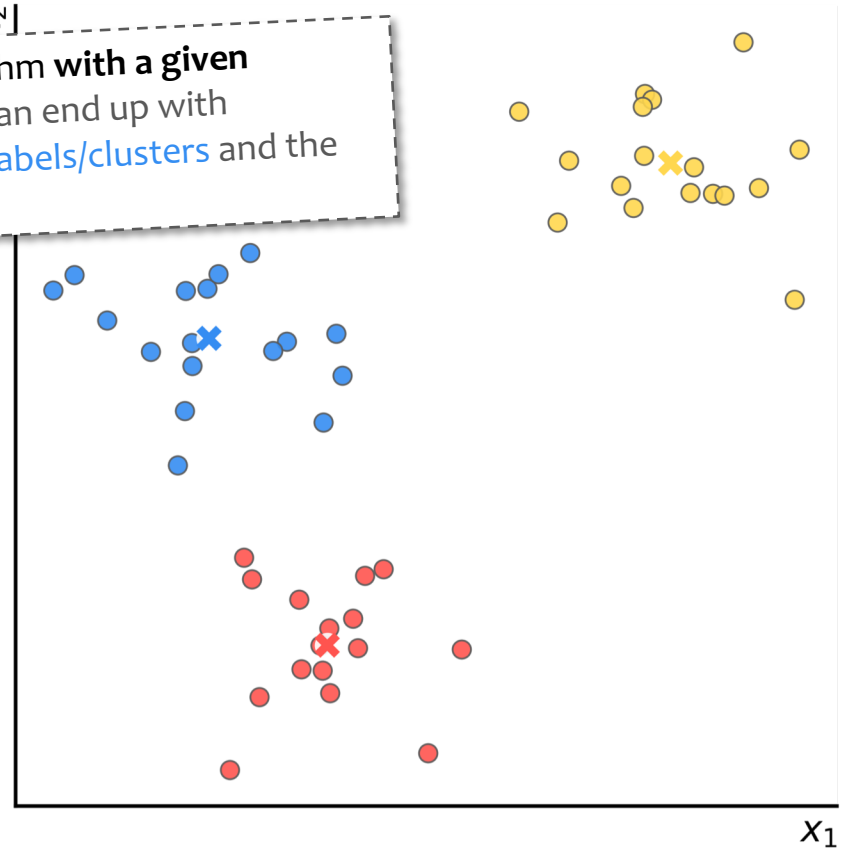


3) Repeat until **centroids do not move significantly**:

Stop!

- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters

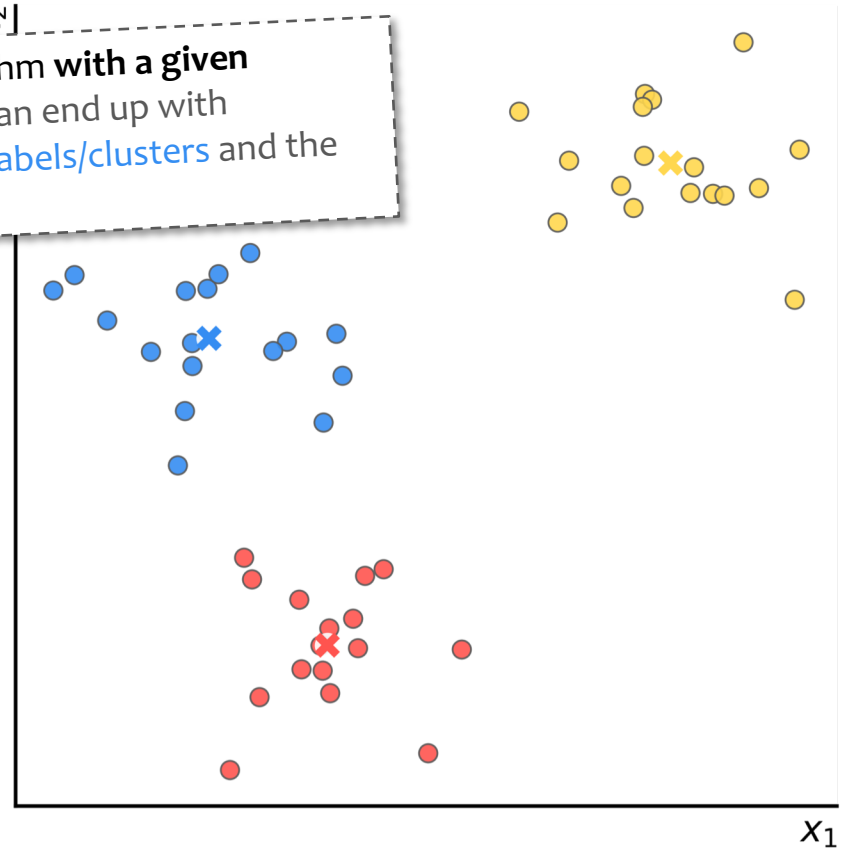
If you consider allow to stop the algorithm **with a given tolerance for centroid reposition**, we can end up with **inconsistencies** between the samples' labels/clusters and the final **centroids**!



3) Repeat until **centroids do not move significantly**:

Stop!

- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



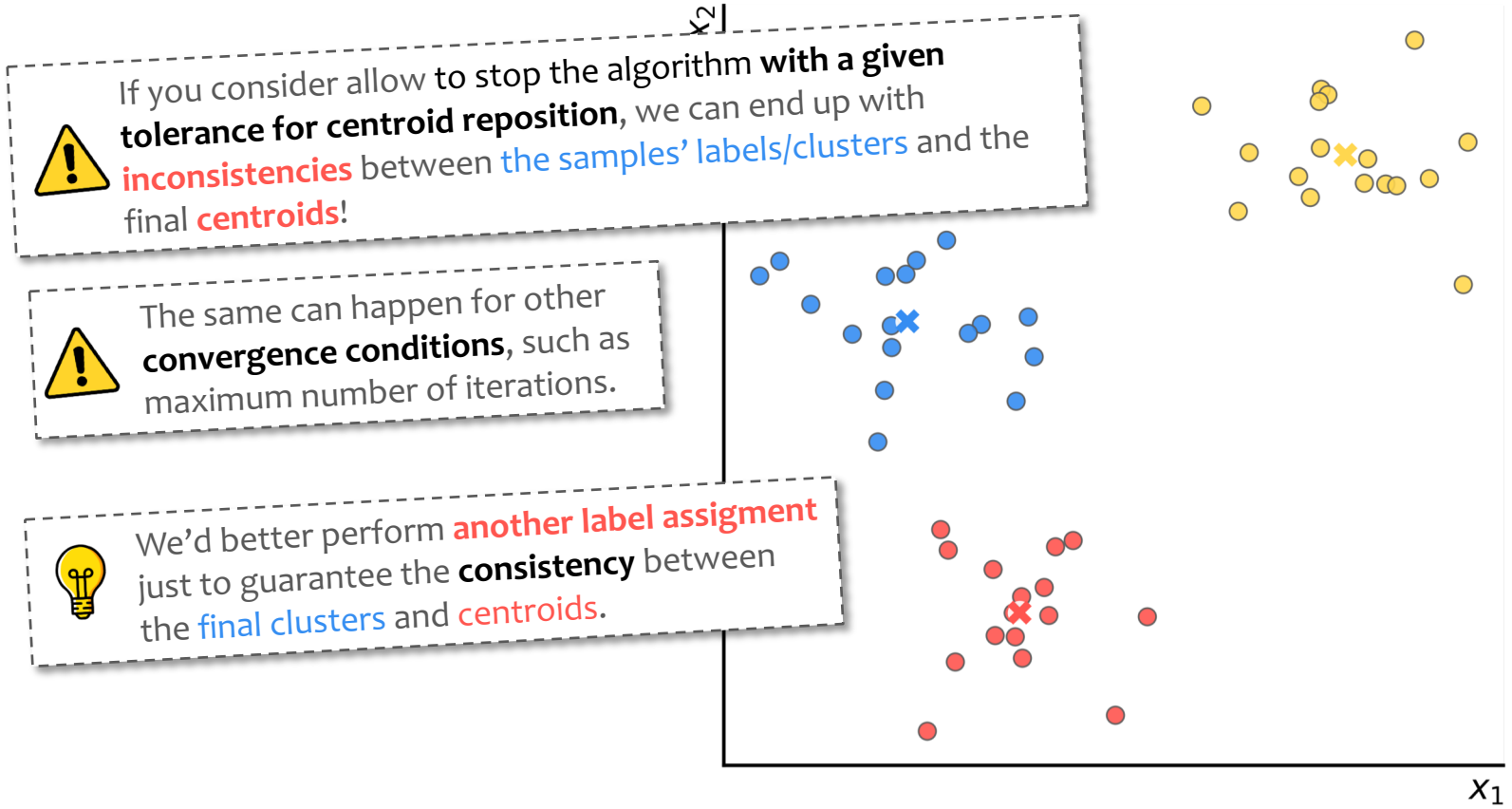
If you consider allow to stop the algorithm **with a given tolerance for centroid reposition**, we can end up with **inconsistencies** between the samples' labels/clusters and the final **centroids**!

The same can happen for other **convergence conditions**, such as maximum number of iterations.

3) Repeat until **centroids do not move significantly**:

Stop!

- **4)** Assign each instance to **the nearest** centroid;
- **5)** Move the centroids to the (new) centers of their clusters



If you consider allow to stop the algorithm **with a given tolerance for centroid reposition**, we can end up with **inconsistencies** between the samples' labels/clusters and the final **centroids**!

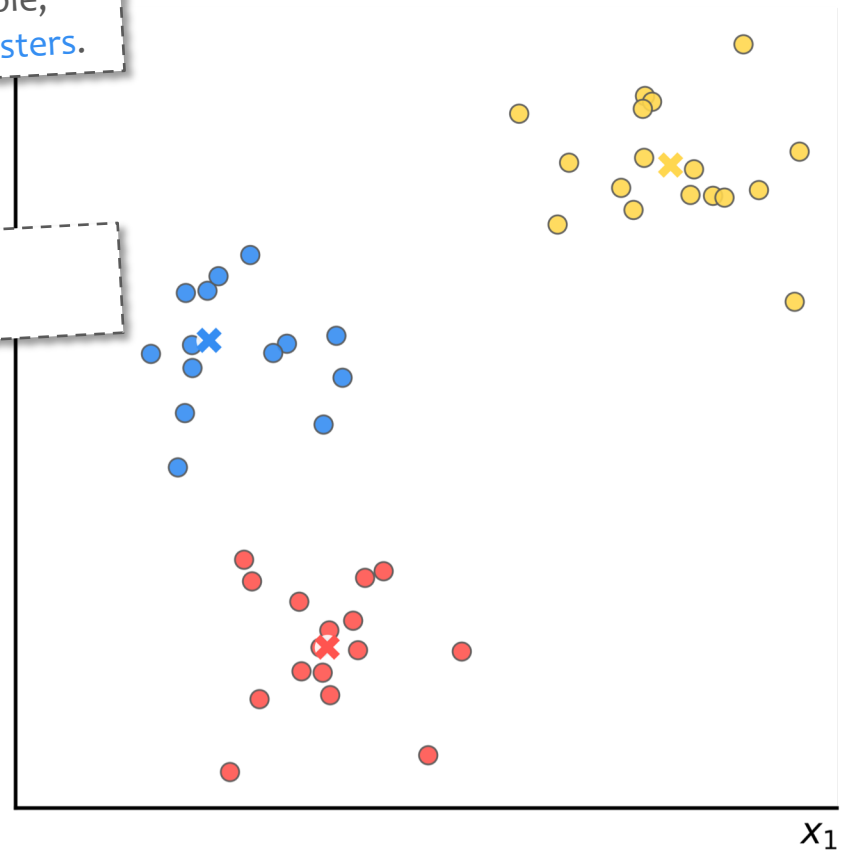
The same can happen for other **convergence conditions**, such as maximum number of iterations.

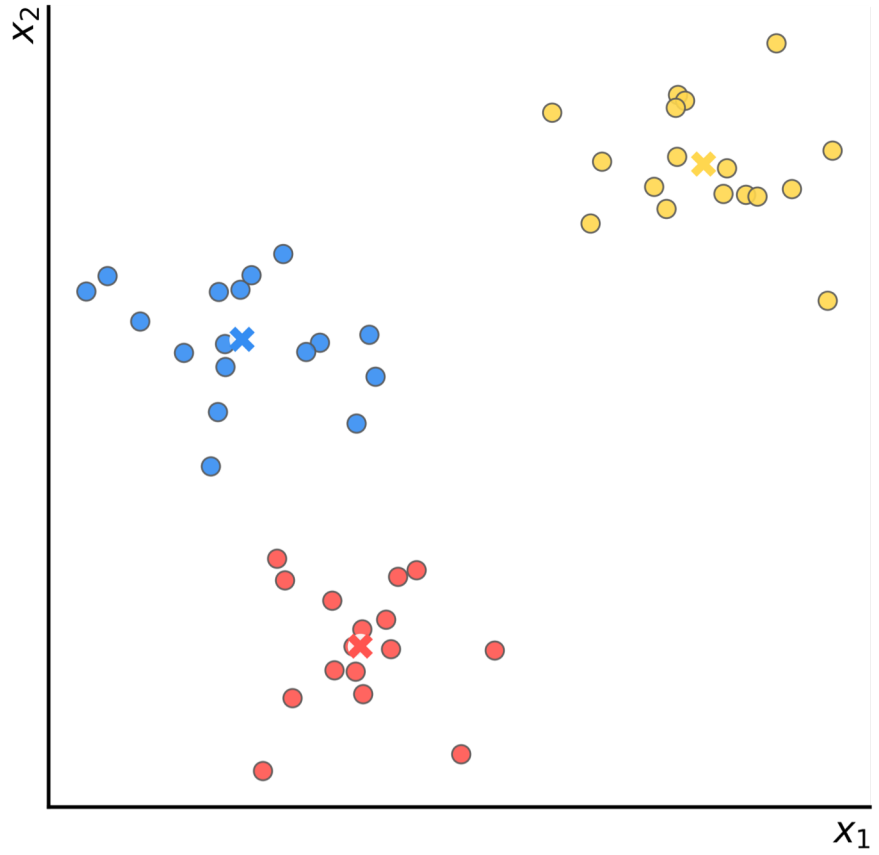
We'd better perform **another label assigment** just to guarantee the **consistency** between the final clusters and centroids.

6) Assign each instance to **the final nearest** centroid;

Since the final centroids are exactly the same from its previous iteration, for this example, this label assigment obtains the same clusters.

KMeans implementation in Scikit-Learn performs **this final label assignment.**

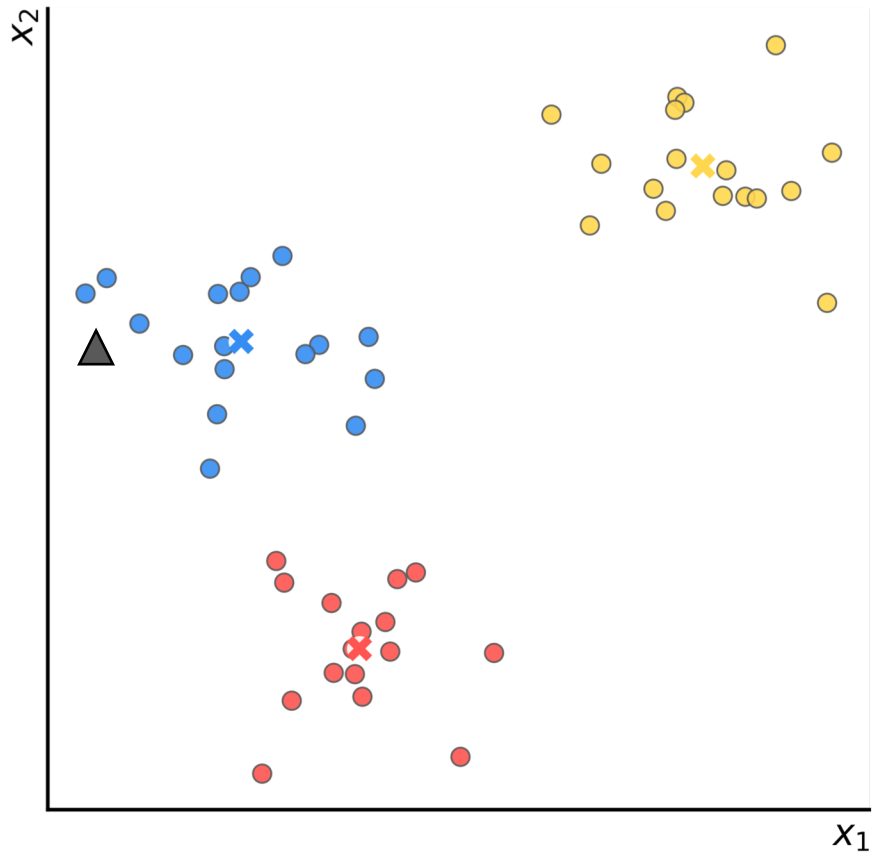



x_2
x_1

KMeans

Usage

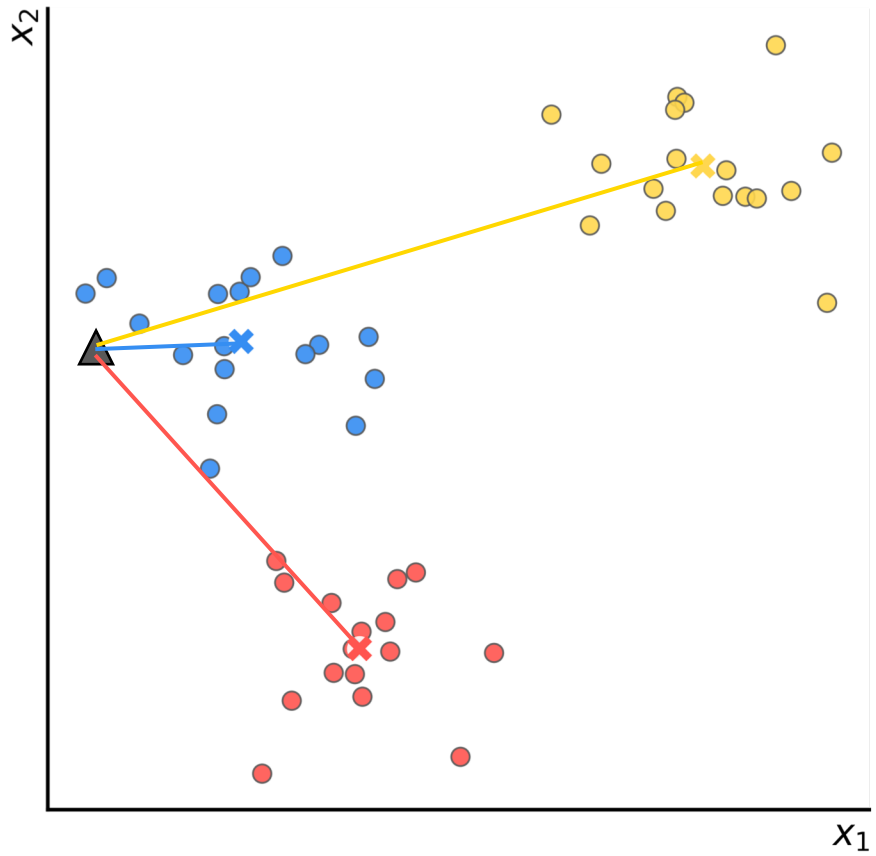
- For each new **testing sample**:
  - Assign each instance to **the nearest** centroid;



KMeans

Usage

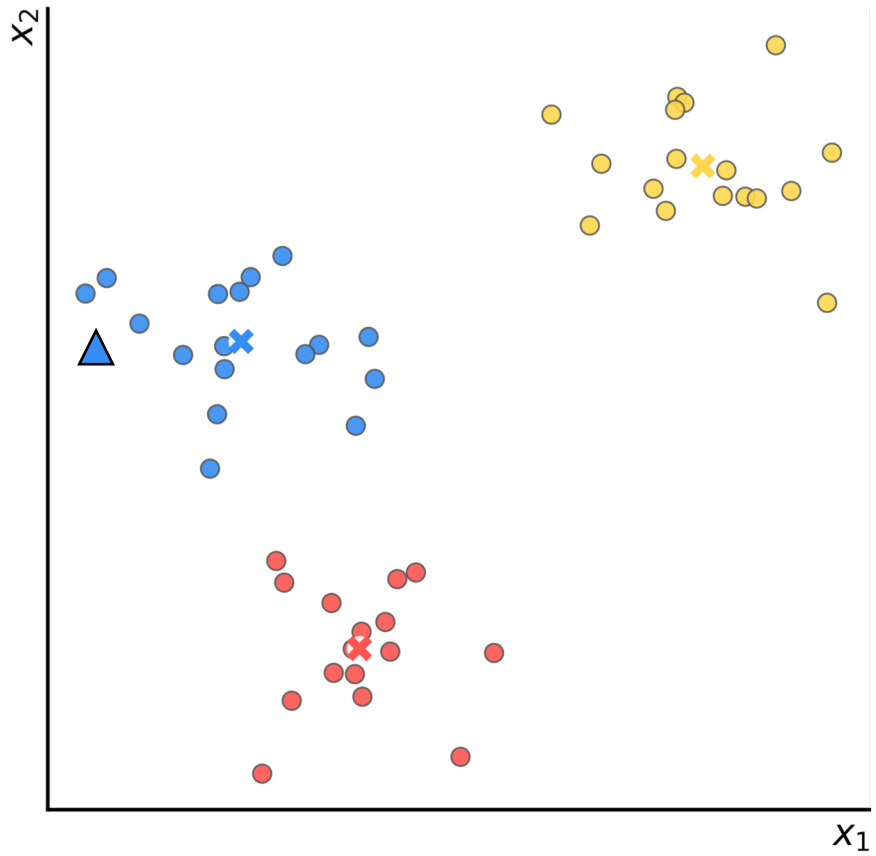
- For each new **testing sample**:
  - Assign each instance to **the nearest** centroid;



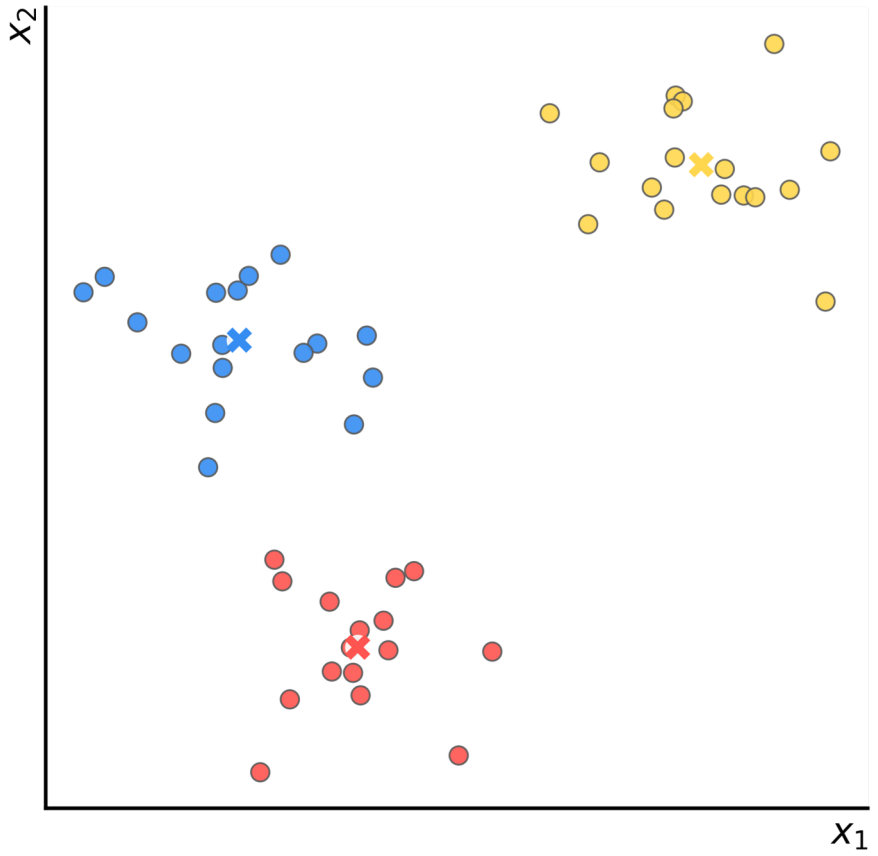
KMeans

Usage

- For each new **testing sample**:
  - Assign each instance to **the nearest** centroid;

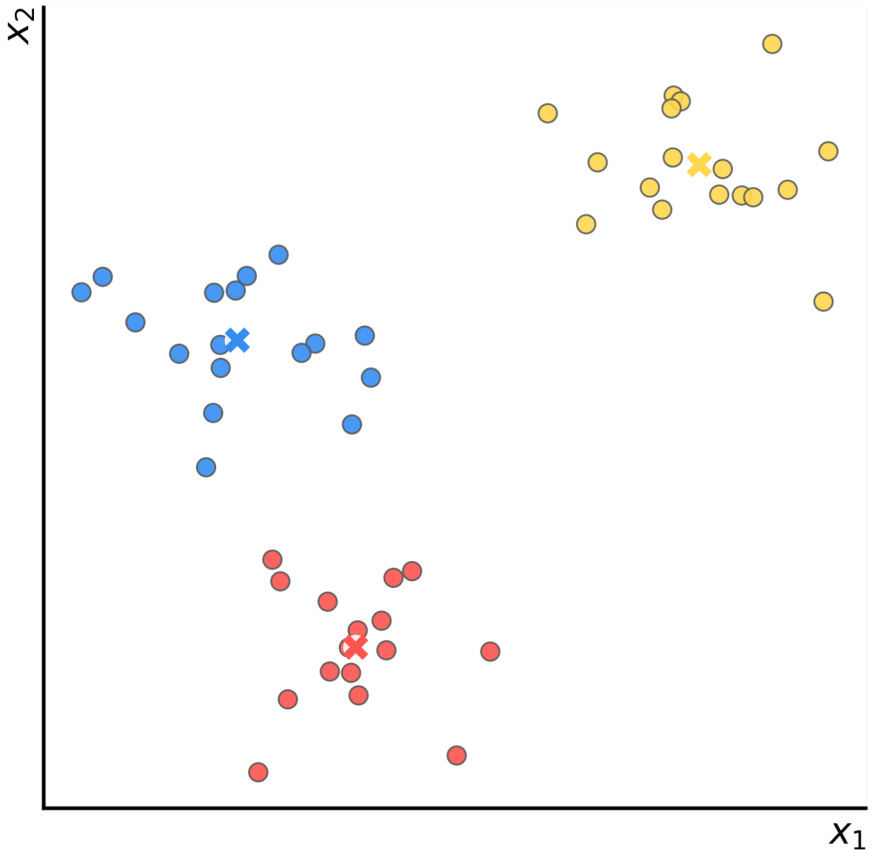


The algorithm is **guaranteed to converge**, but **it may not converge** to the **right solution** (*i.e.*, it may converge to **a local optimum**).



The algorithm is **guaranteed to converge**, but **it may not converge** to the **right solution** (*i.e.*, it may converge to **a local optimum**).

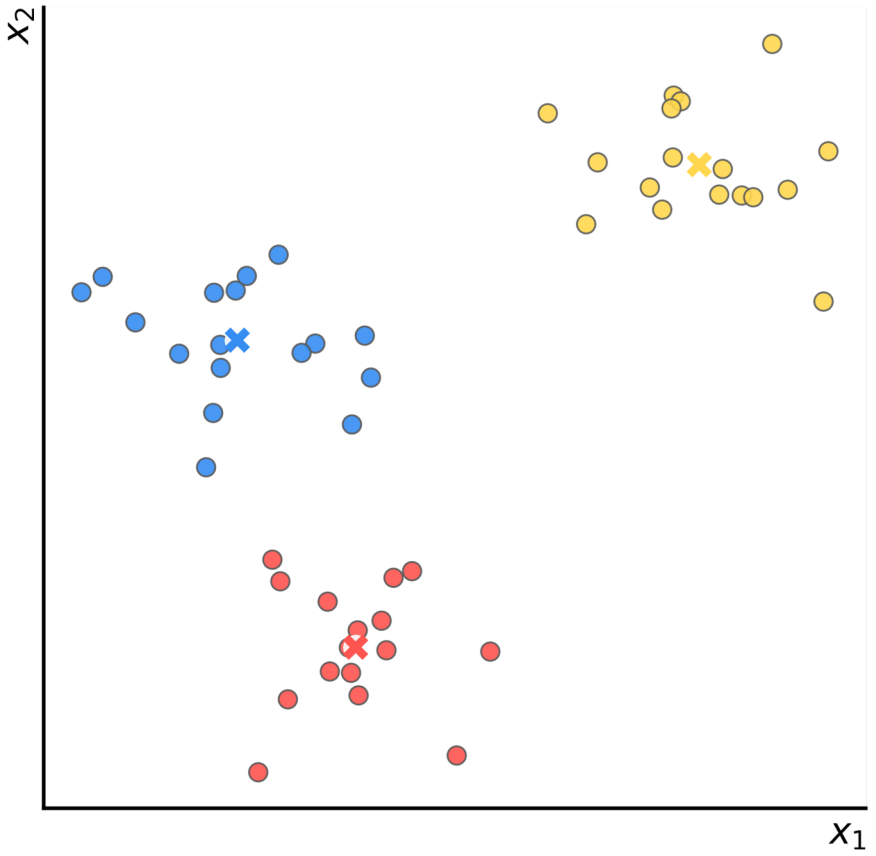
This depends on the **centroid initialization**.



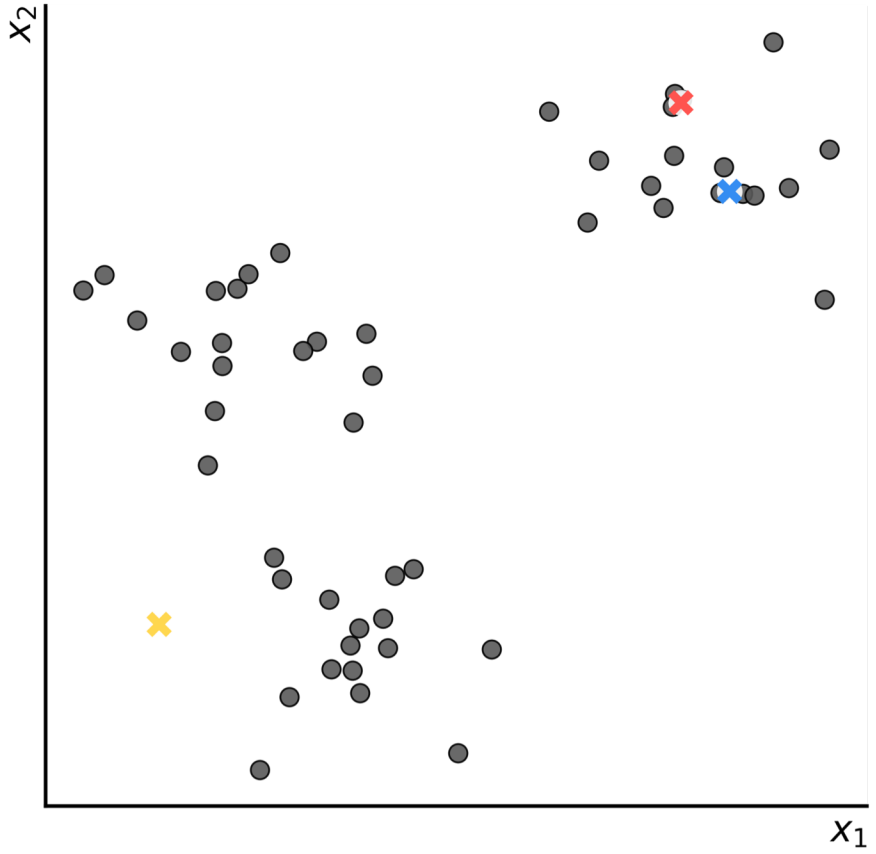
The algorithm is **guaranteed to converge**, but **it may not converge** to the **right solution** (*i.e.*, it may converge to **a local optimum**).

This depends on the **centroid initialization**.

Poor initial centroids result to **poor clusters**.



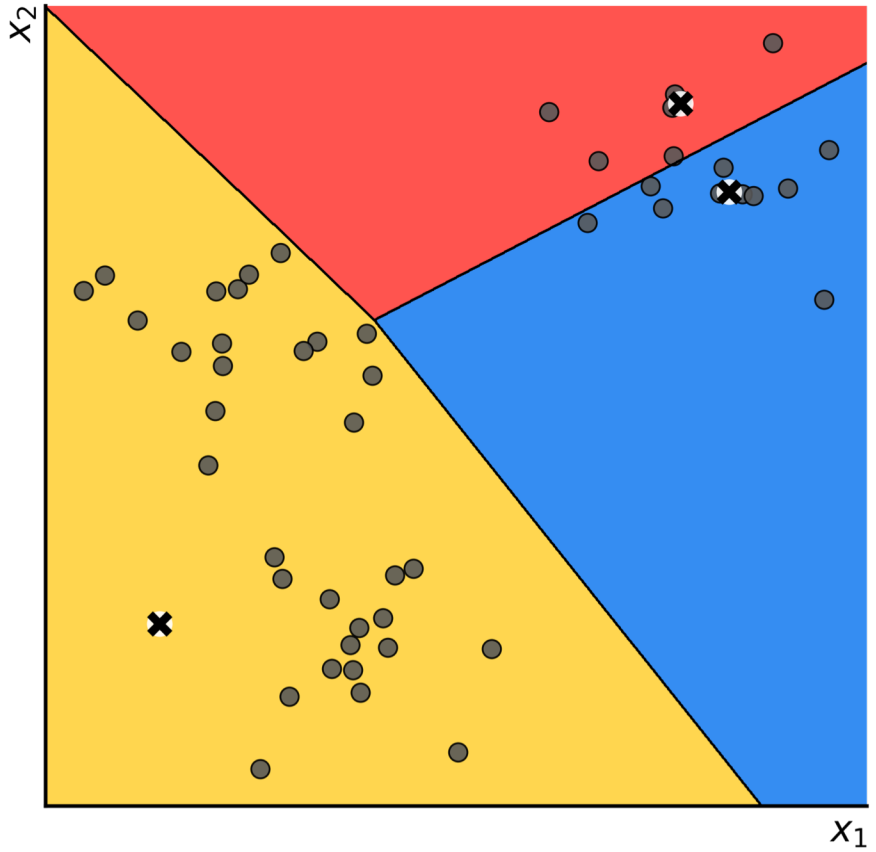
Example with poor initial centroids



initial centroids

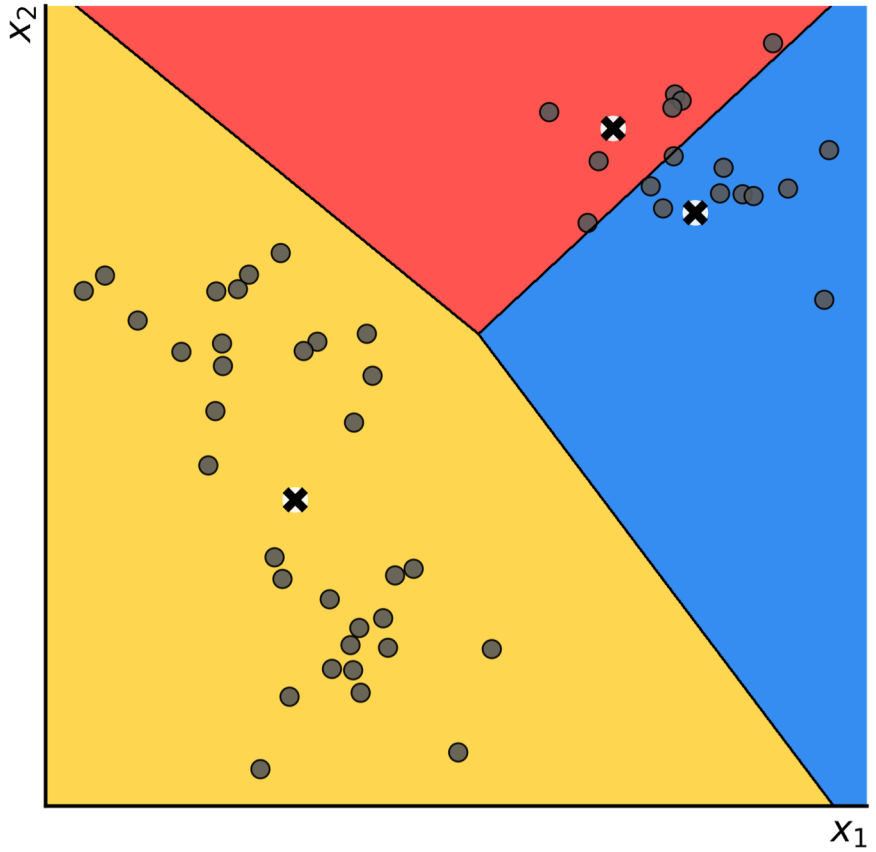
Example with poor initial centroids

Iteration 1



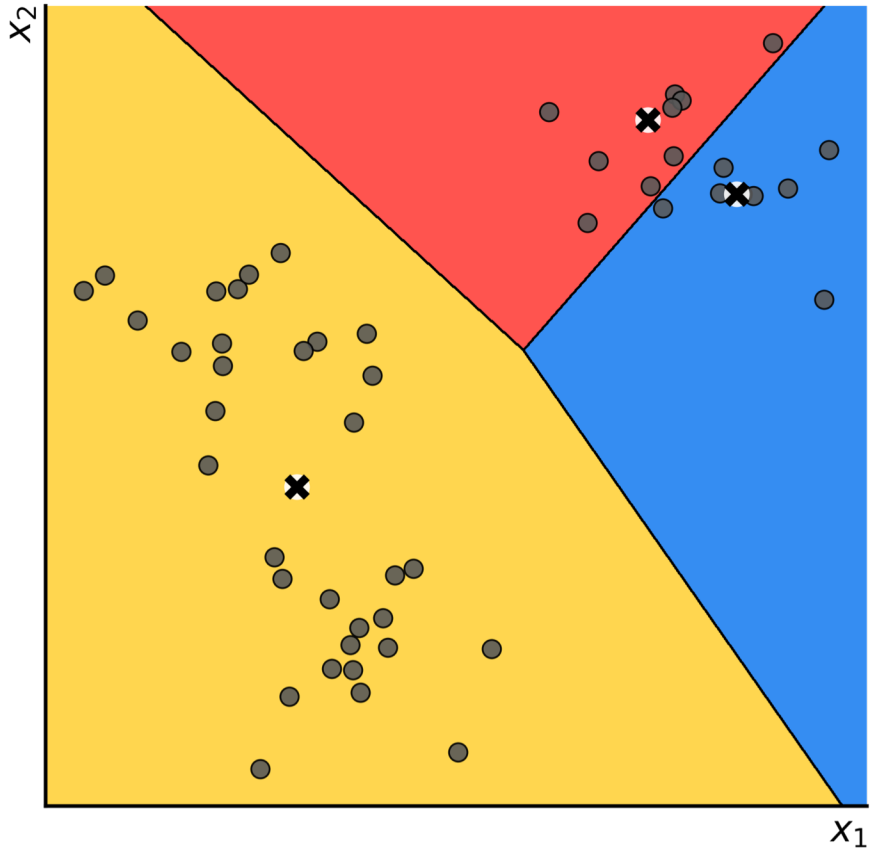
Example with poor initial centroids

Iteration 2



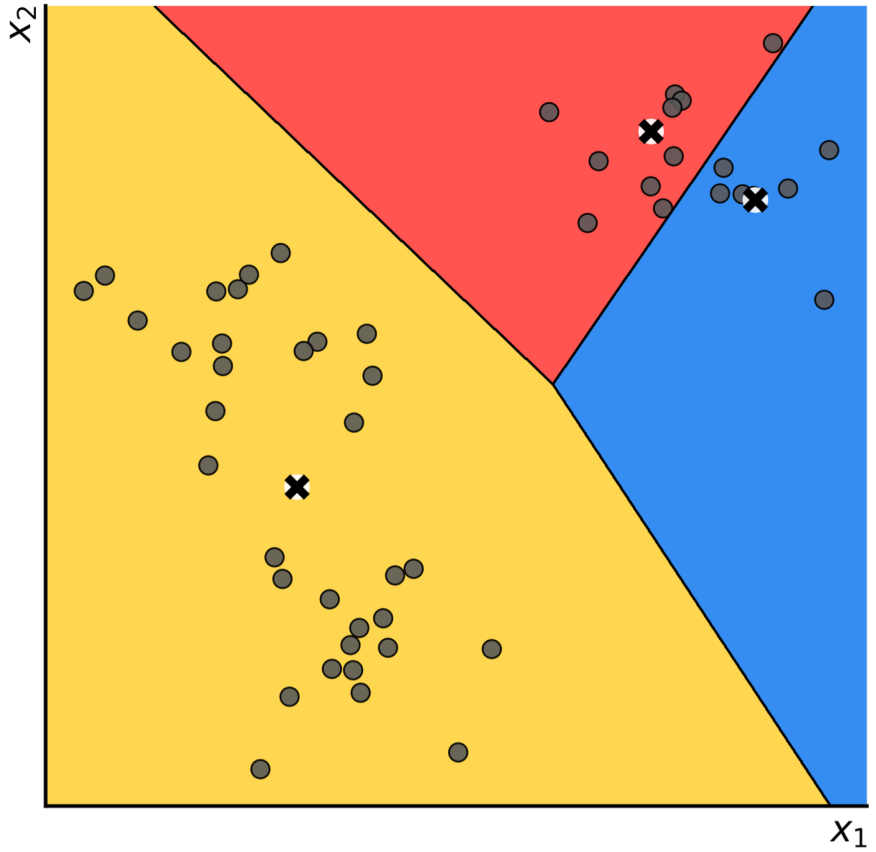
Example with poor initial centroids

Iteration 3



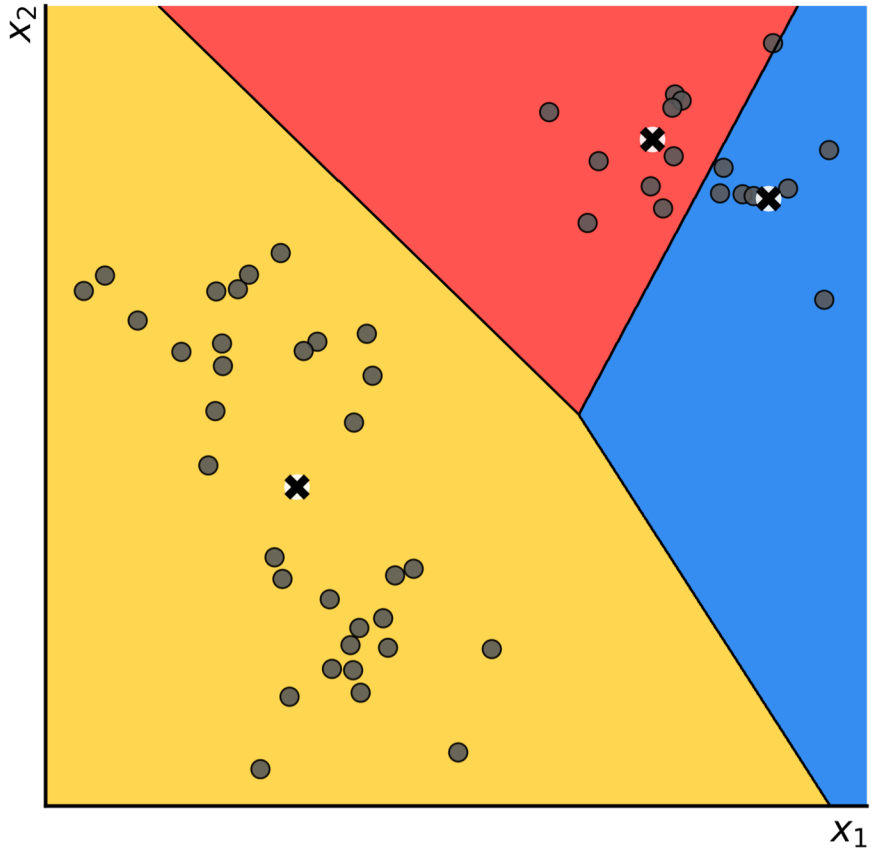
Example with poor initial centroids

Iteration 4

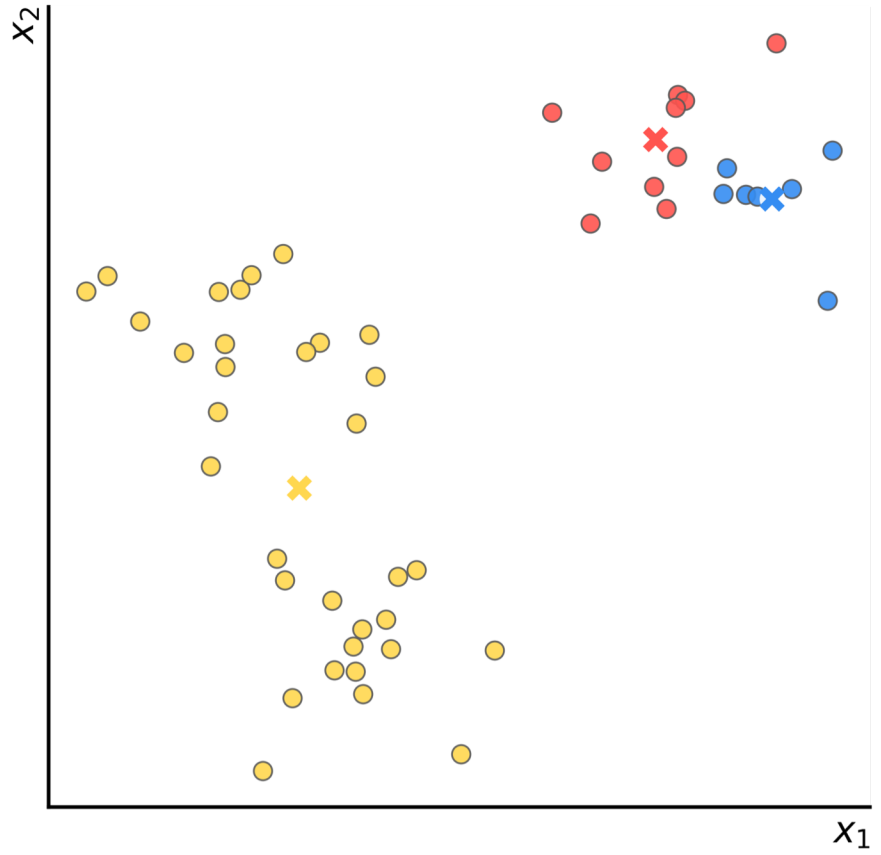


Example with poor initial centroids

Final
Decision Boundary

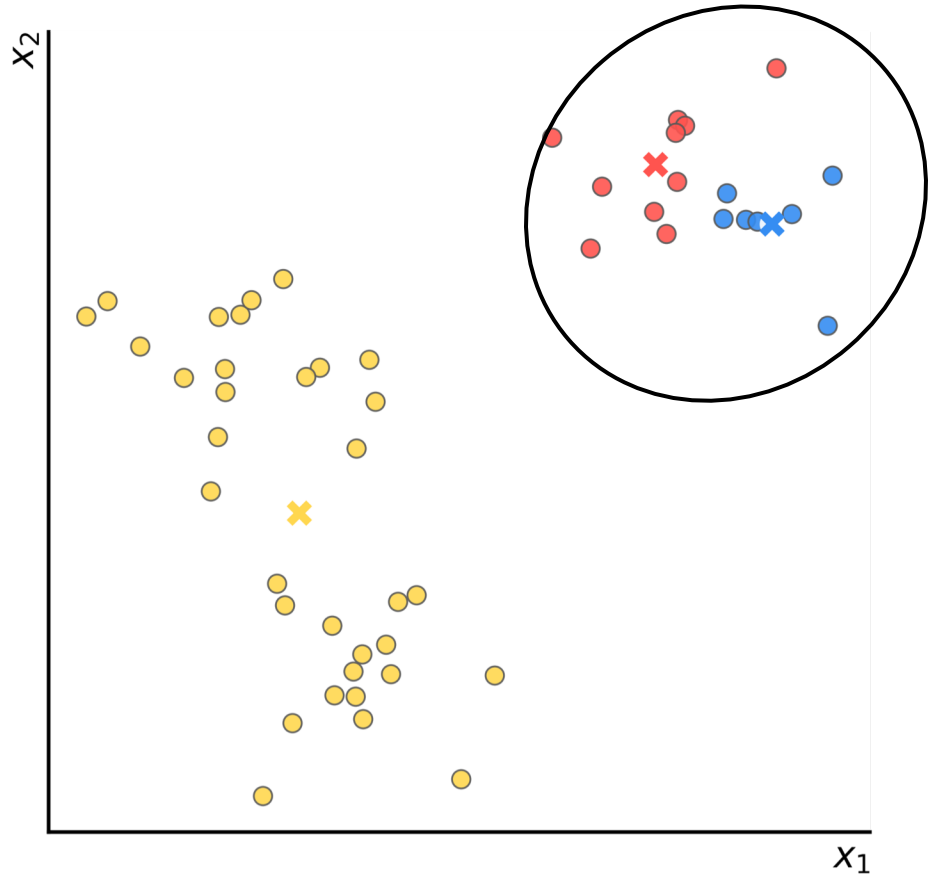


Example with poor initial centroids



Final
Clusters and Centroids

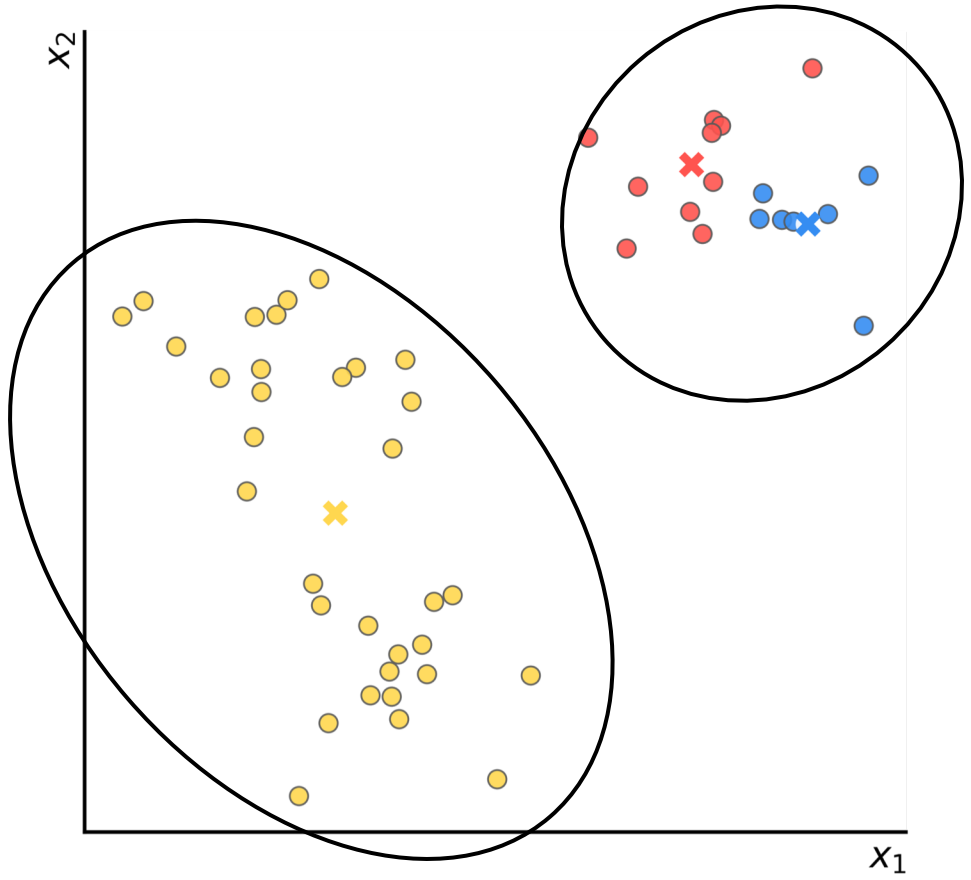
Example with poor initial centroids



A **cluster** is overpartitioned into two clusters

Example with poor initial centroids

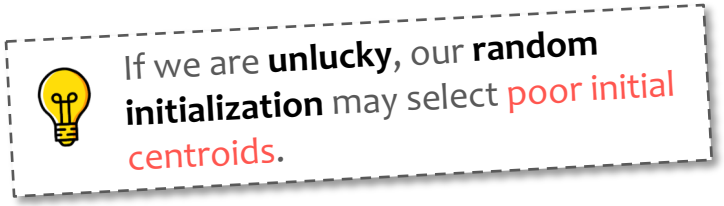
Two **clusters** is underpartitioned into a single cluster



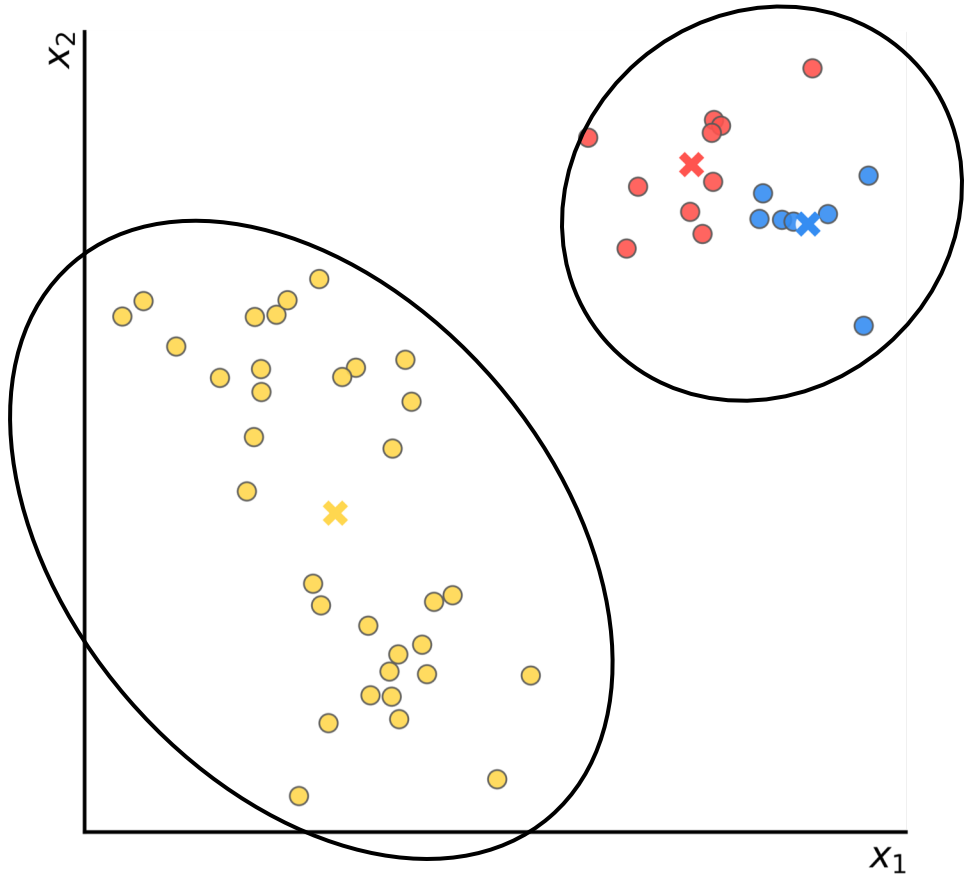
A **cluster** is overpartitioned into two clusters

Example with poor initial centroids

Two **clusters** is underpartitioned into a single cluster



If we are **unlucky**, our **random initialization** may select poor initial centroids.

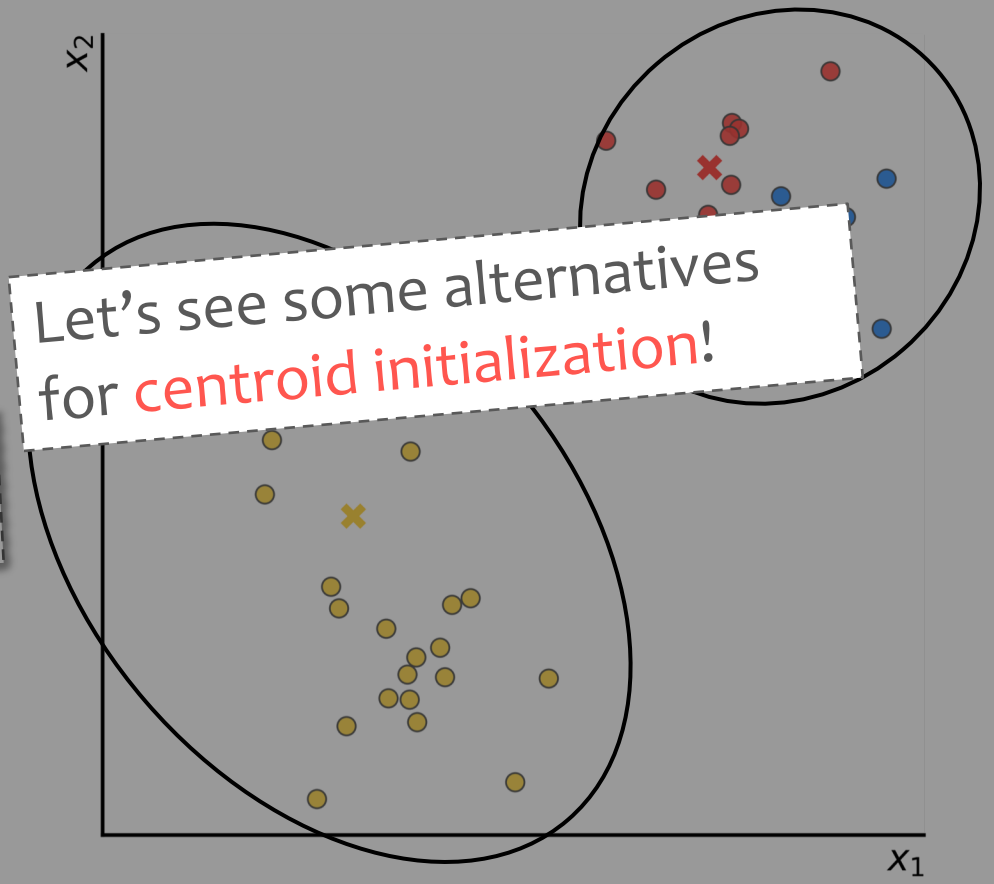


A **cluster** is overpartitioned into two clusters

Example with poor initial centroids

A **cluster** is overpartitioned into two clusters

Two **clusters** is underpartitioned into a single cluster



Let's see some alternatives for centroid initialization!

If we are **unlucky**, our **random initialization** may select poor initial centroids.

KMeans++

Initialization:

1) Take one centroid $\boldsymbol{c^{(1)}}$, chosen uniformly at random from the dataset $\mathbf{X}$;

2) Take a new centroid $\boldsymbol{c^{(l)}}$, choosing an instance $\boldsymbol{x^{(i)}} \in \mathbf{X}$ with probability $\frac{D(\boldsymbol{x^{(i)}})^2}{\sum_{j=1}^{m} D(\boldsymbol{x^{(j)}})^2}$;

- where $D(\boldsymbol{x^{(i)}})$is the distance between the instance $\boldsymbol{x^{(i)}}$ and **the nearest centroid** that **was already chosen**;
- This probability distribution ensures that instances farther away from already chosen centroids are much more likely be selected as centroids.

3) Repeat the previous step until all k centroids have been chosen;

KMeans++

Initialization:

1) Take one centroid $\boldsymbol{c^{(1)}}$, chosen uniformly at random from the dataset $\mathbf{X}$;

2) Take a new centroid $\boldsymbol{c^{(l)}}$, choosing an instance $\boldsymbol{x^{(i)}} \in \mathbf{X}$ with probability $\frac{D(\boldsymbol{x^{(i)}})^2}{\sum_{j=1}^{m} D(\boldsymbol{x^{(j)}})^2}$;

- where $D(\boldsymbol{x^{(i)}})$is the distance between the instance $\boldsymbol{x^{(i)}}$ and **the nearest centroid** that **was already chosen**;
- This probability distribution ensures that instances farther away from already chosen centroids are much more likely be selected as centroids.

3) Repeat the previous step until all k centroids have been chosen;

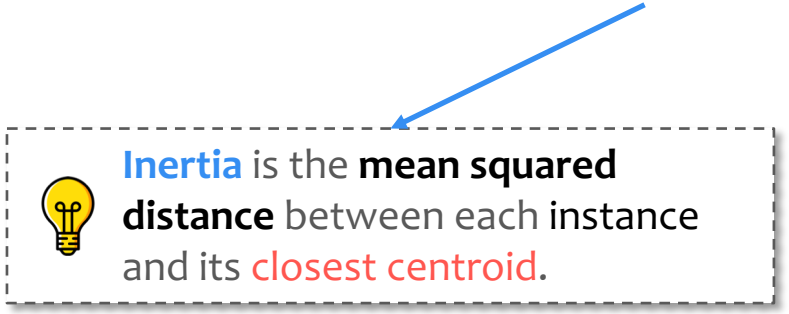
This is the **default initialization** for `KMeans` in Scikit-learn, but we can also force to use the **the original method** (*i.e.*, picking k instances randomly as initial centroids).

Optimization Strategy

Another solution to get proper initial centroids and, consequently, good **resulting clusters** is **to run KMeans** multiple times with different random initializations and **keep the best model**, which is that with **the lowest inertia**.

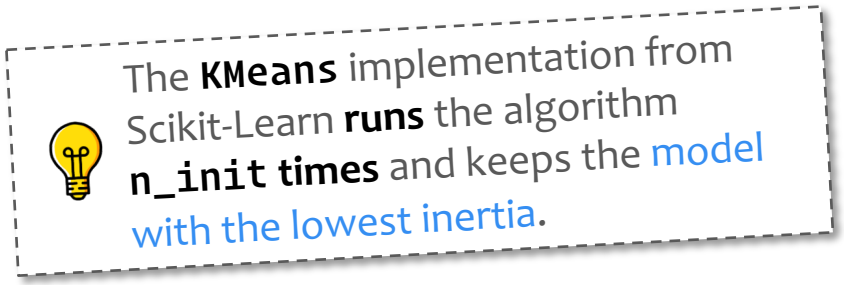
Optimization Strategy

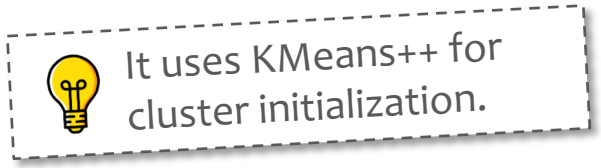
Another solution to get proper initial centroids and, consequently, good **resulting clusters** is **to run KMeans** multiple times with different random initializations and **keep the best model**, which is that with **the lowest inertia**.

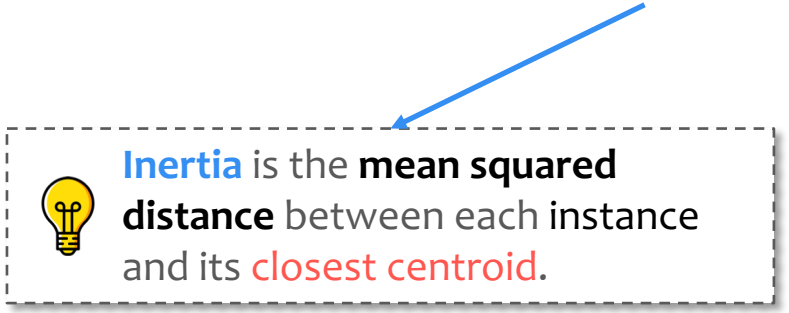


Optimization Strategy

Another solution to get proper initial centroids and, consequently, good **resulting clusters** is **to run KMeans** multiple times with different random initializations and **keep the best model**, which is that with **the lowest inertia**.







Codes

Finding the optimal number of clusters

Finding the optimal number of clusters

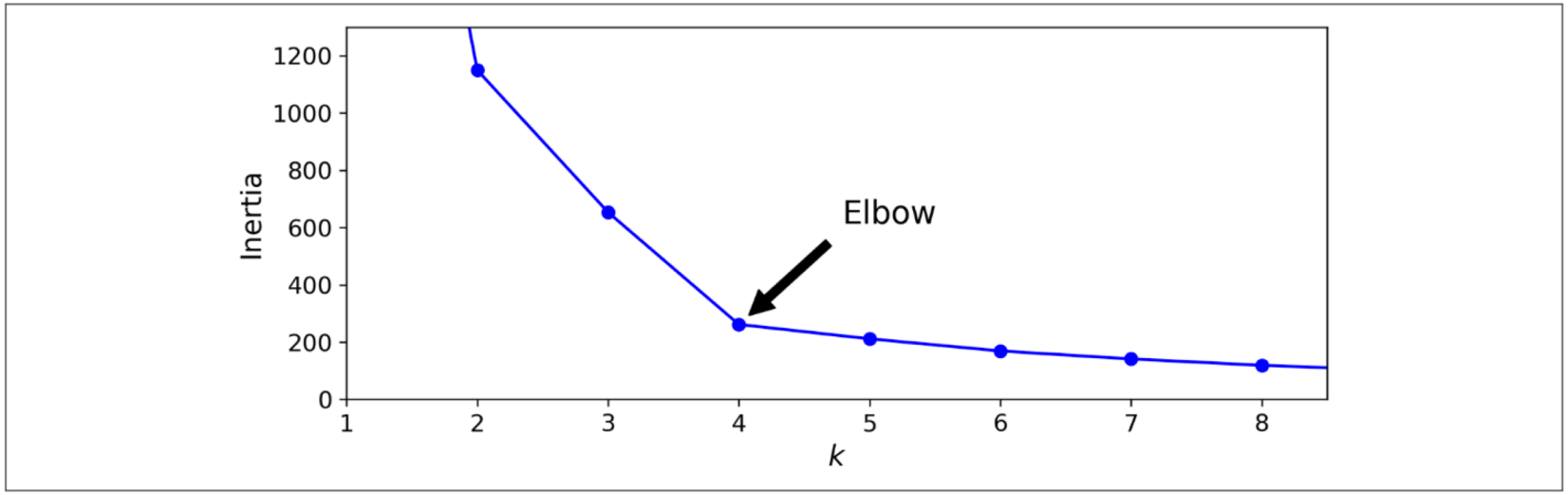


Figure 9-8. When plotting the inertia as a function of the number of clusters k, the curve often contains an inflexion point called the "elbow"

https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb

Finding the optimal number of clusters

This technique for choosing the best value for the number of clusters is rather coarse.

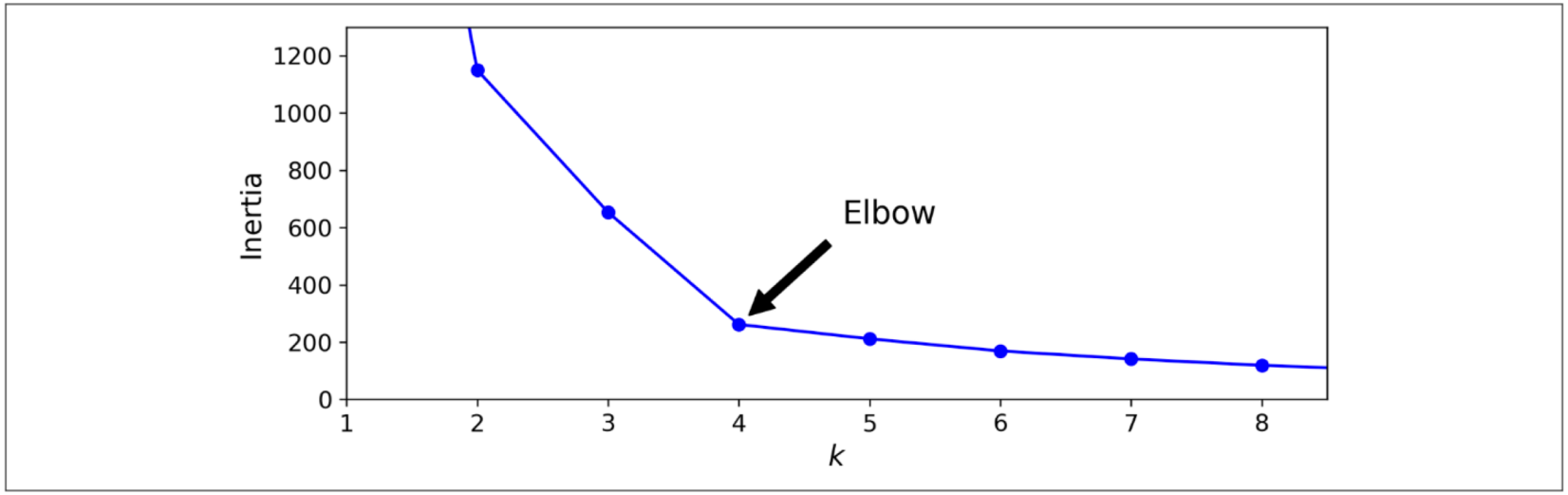


Figure 9-8. When plotting the inertia as a function of the number of clusters k, the curve often contains an inflexion point called the "elbow"

https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb

Finding the optimal number of clusters

This technique for choosing the best value for the number of clusters is rather coarse.

An alternative is to use the **silhouette coefficient** instead!

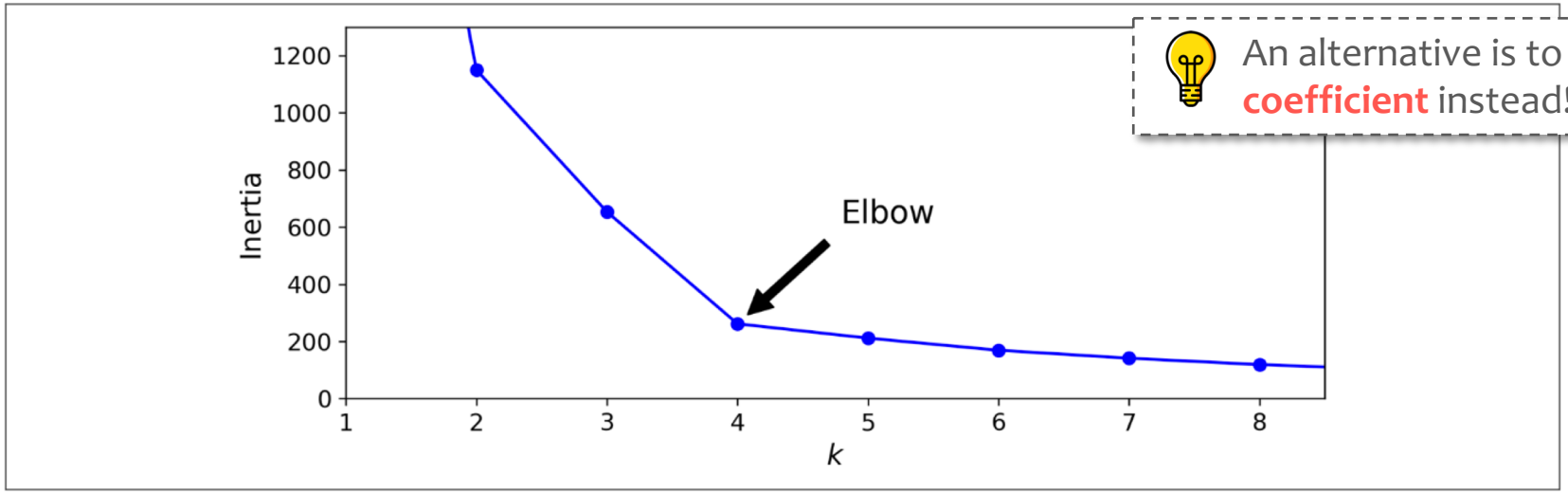
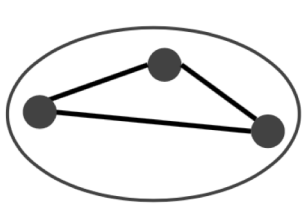


Figure 9-8. When plotting the inertia as a function of the number of clusters k, the curve often contains an inflexion point called the "elbow"

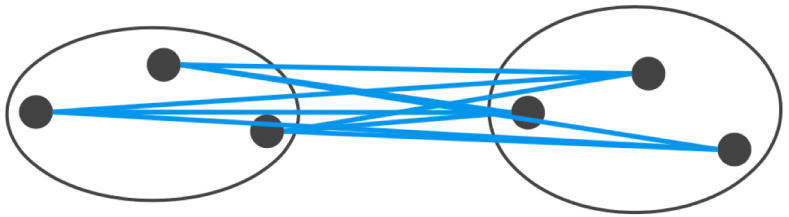
https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb

Silhouette Coefficient

The silhouette value is a measure of how similar a sample is to its own cluster (**cohesion**) compared to other clusters (**separation**)



Cohesion



Separation

Silhouette Coefficient

- The silhouette value is a measure of how similar a sample is to its own cluster (**cohesion**) compared to other clusters (**separation**)
- The silhouette ranges from −1 to +1
  - High value = the clustering configuration is appropriate
  - Low value = the clustering configuration may have too many or too few clusters

Silhouette Coefficient

- The Silhouette Coefficient is defined for each sample and is composed of two scores:
  - **a:** The mean distance between **a sample** and all other points in **the same cluster**
  - **b:** The mean distance between **a sample** and all other points in **the next nearest cluster**
- The Silhouette Coefficient s for a single sample is given as:

$$s = \frac{b - a}{\max(a, b)}$$

- The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering (a $\ll$ b).
- Scores around zero indicate overlapping clusters

Silhouette Coefficient

```
>>> from sklearn.metrics import silhouette_score
>>> silhouette_score(X, kmeans.labels_)
0.655517642572828
```

Let's compare the silhouette scores for different numbers of clusters (see Figure 9-9).

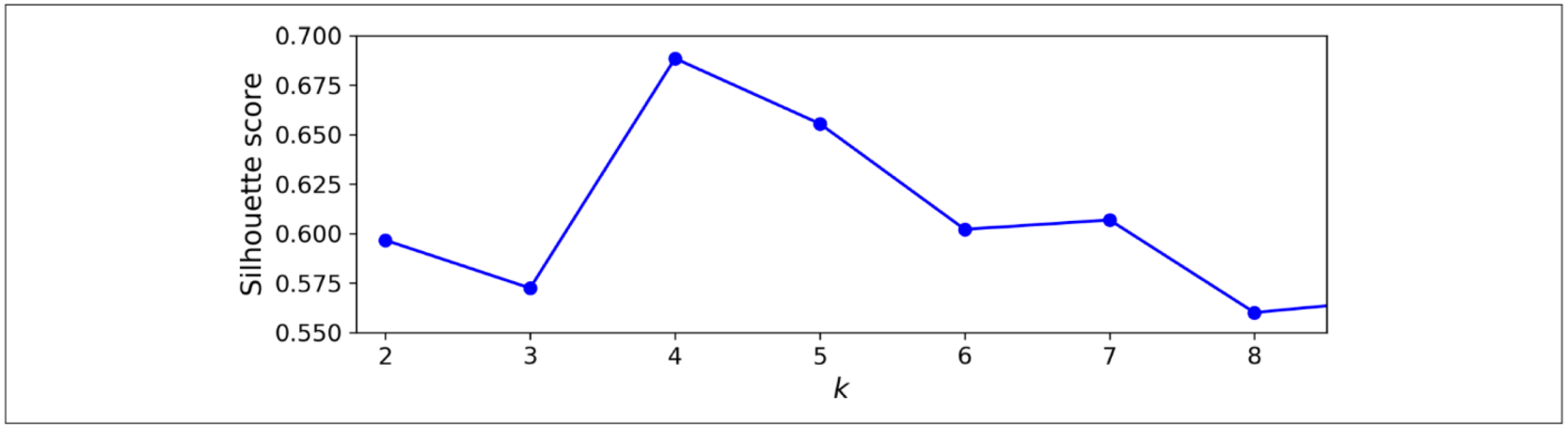


Figure 9-9. Selecting the number of clusters k using the silhouette score

Silhouette Coefficient

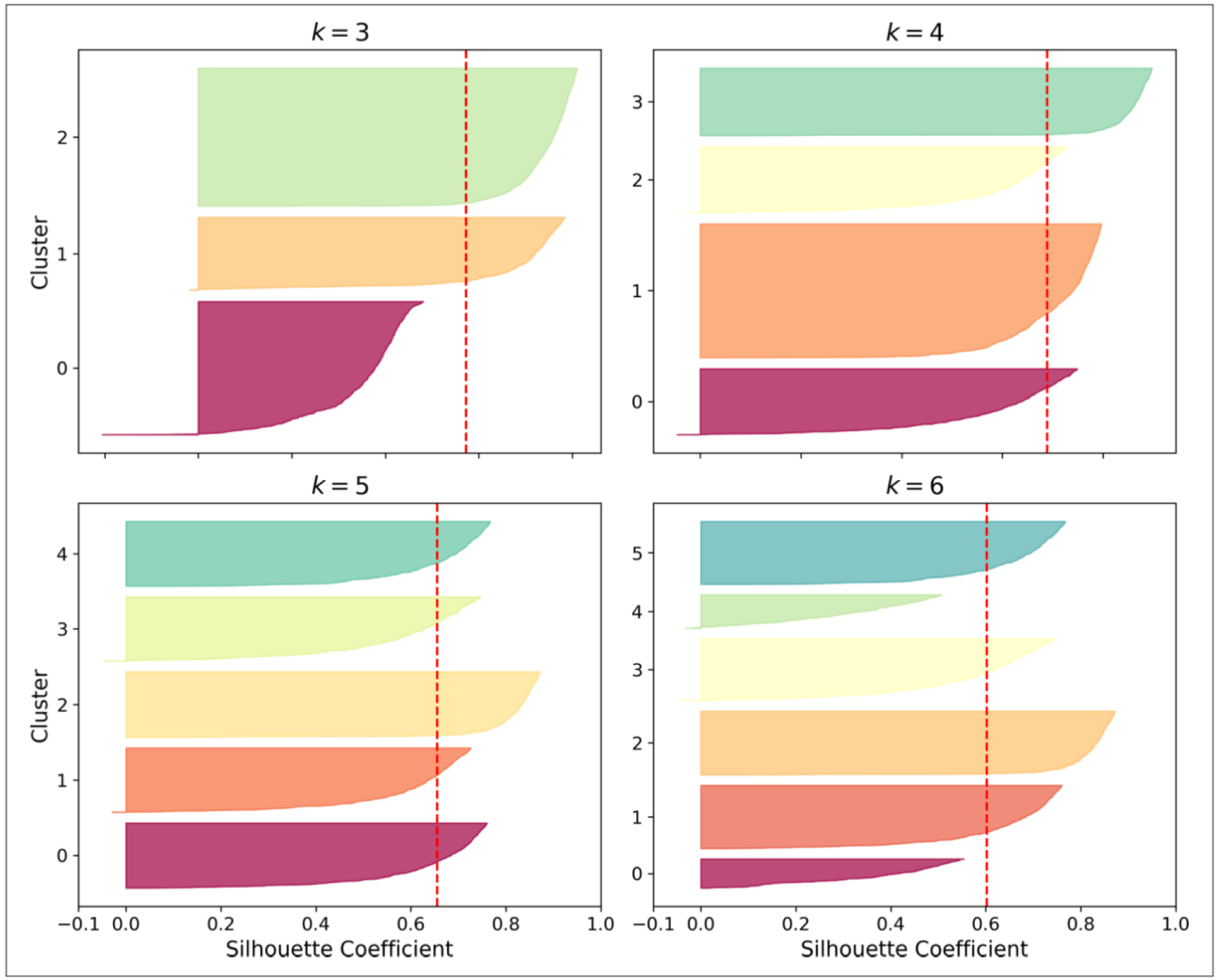


Figure 9-10. Analyzing the silhouette diagrams for various values of k

Aprendizado de Máquina e Reconhecimento de Padrões

Kmeans

Prof. Dr. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas