Data Mining

Clustering

https://data-mining.github.io/winter-2026/

CS 453/553 – Winter 2026
Yu Wang, Ph.D.
Assistant Professor
Computer Science
University of Oregon

Fun Facts about ML/AI/DM

Yann Lecun

Geoffrey Hinton

Yoshua Bengio

Godfathers of Deep Learning

Mila

Fun Facts about ML/AI/DM

Fundamental AI Research

FAIR

- **Llama (Large Language Model Meta AI):** FAIR released the Llama series (Llama 1, 2, and 3.1), which are among the most influential open-source, pre-trained large language models. These allow for democratization, customization, and cost-effective AI development compared to proprietary models.
- **Segment Anything Model (SAM):** A foundational computer vision model that can "segment" (identify and distinguish) any object in a photo, which has become a staple for visual AI tasks.
- PyTorch: FAIR open-sourced this deep learning framework, which is now one of the most widely used libraries for AI research and development globally.

PyTorch

Yann LeCun

Fun Facts about ML/AI/DM

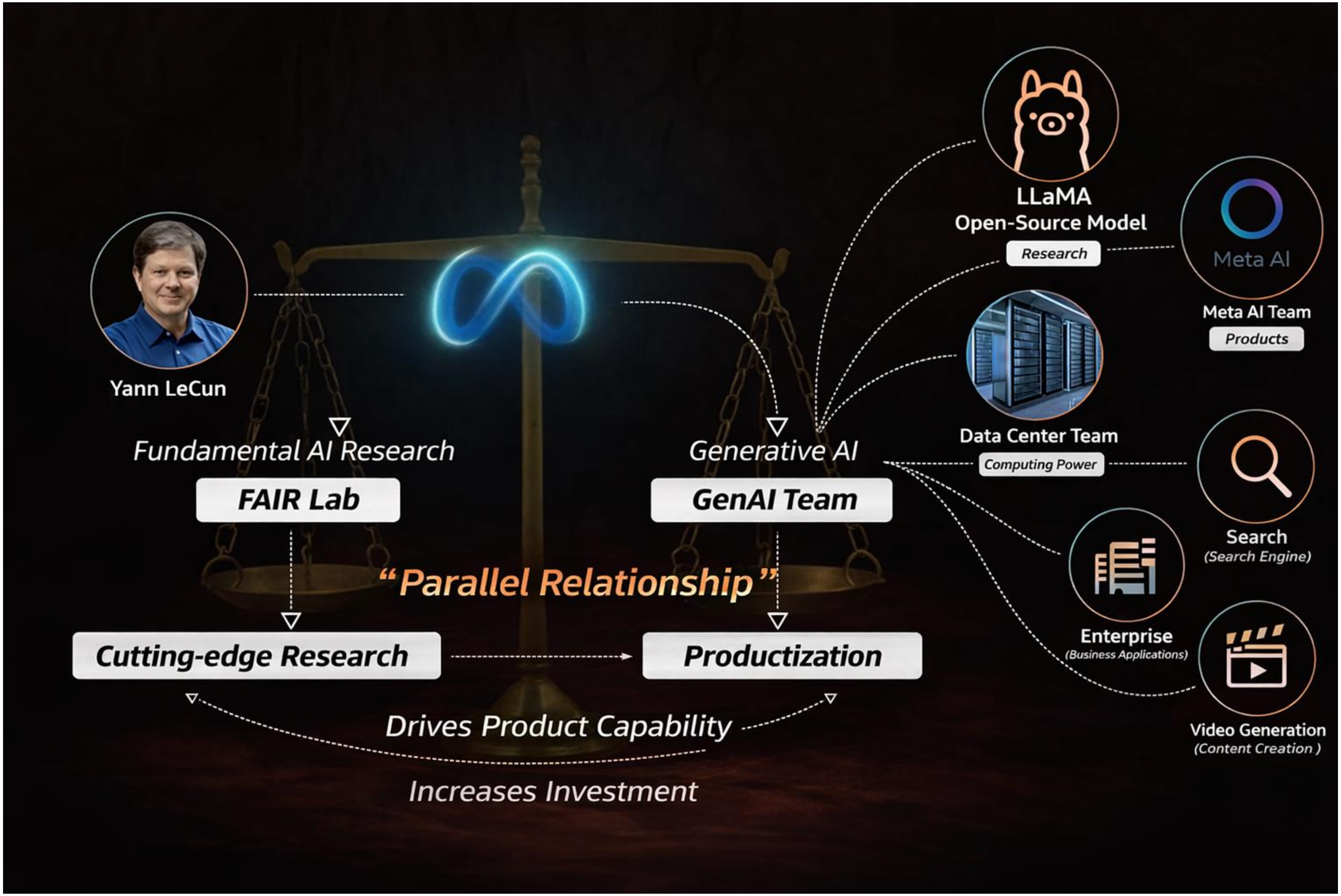
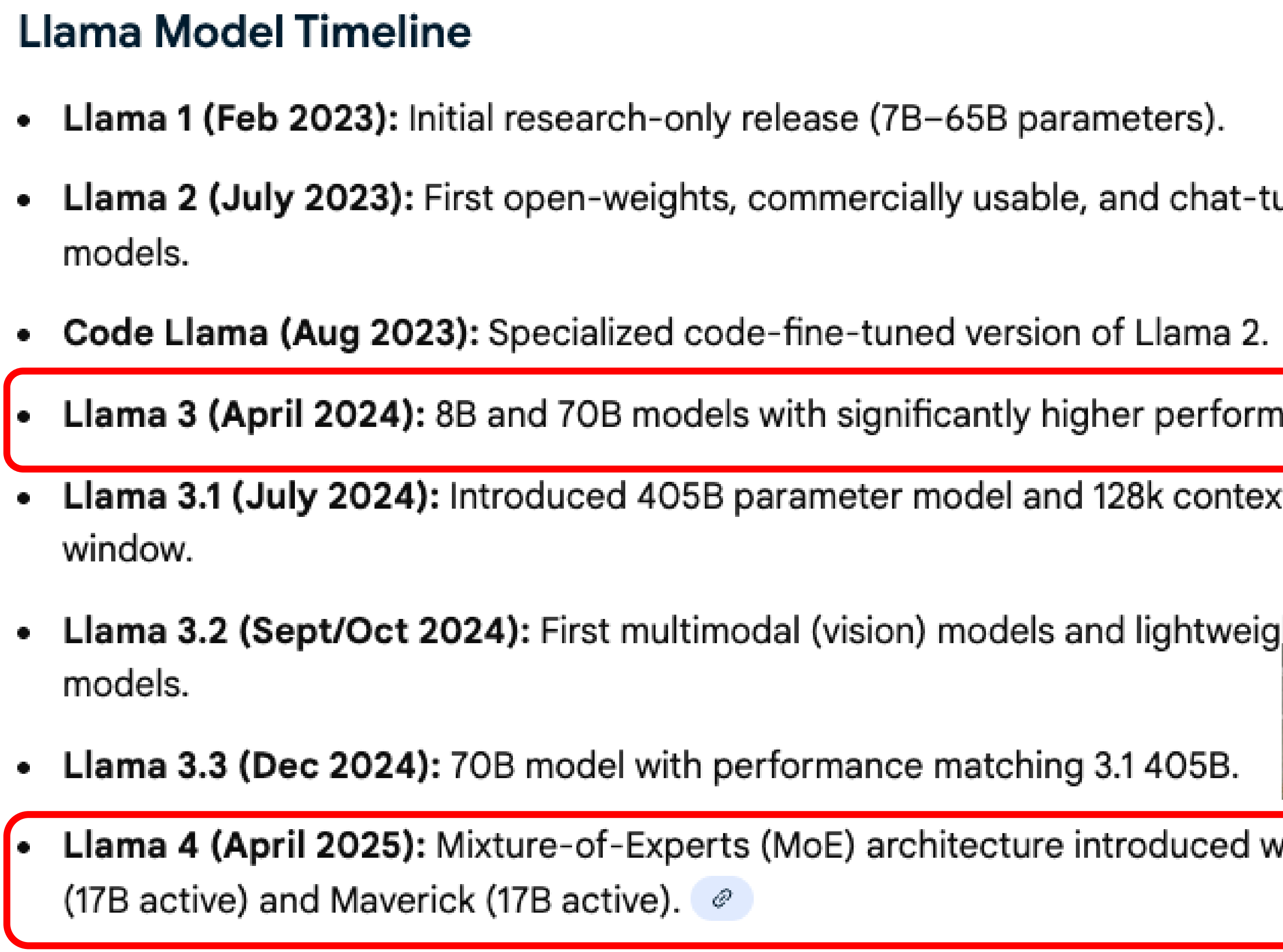


Figure modified by ChatGPT

Fun Facts about ML/AI/DM



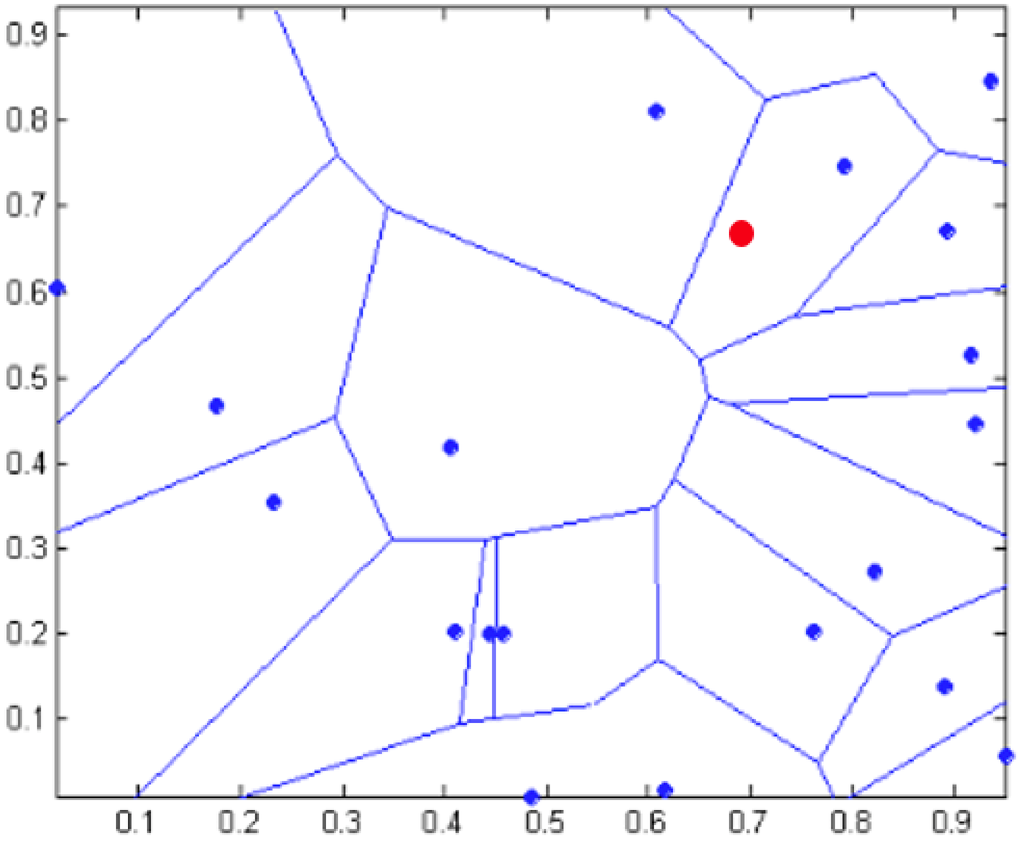
Llama Model Timeline

- **Llama 1 (Feb 2023):** Initial research-only release (7B–65B parameters).
- **Llama 2 (July 2023):** First open-weights, commercially usable, and chat-tuned models.
- **Code Llama (Aug 2023):** Specialized code-fine-tuned version of Llama 2.
- **Llama 3 (April 2024):** 8B and 70B models with significantly higher performance.
- **Llama 3.1 (July 2024):** Introduced 405B parameter model and 128k context window.
- **Llama 3.2 (Sept/Oct 2024):** First multimodal (vision) models and lightweight edge models.
- **Llama 3.3 (Dec 2024):** 70B model with performance matching 3.1 405B.
- **Llama 4 (April 2025):** Mixture-of-Experts (MoE) architecture introduced with Scout (17B active) and Maverick (17B active).

Question Time!

KNN – Time Complexity

1-nn decision boundary is a Voronoi Diagram



Assuming N Training points

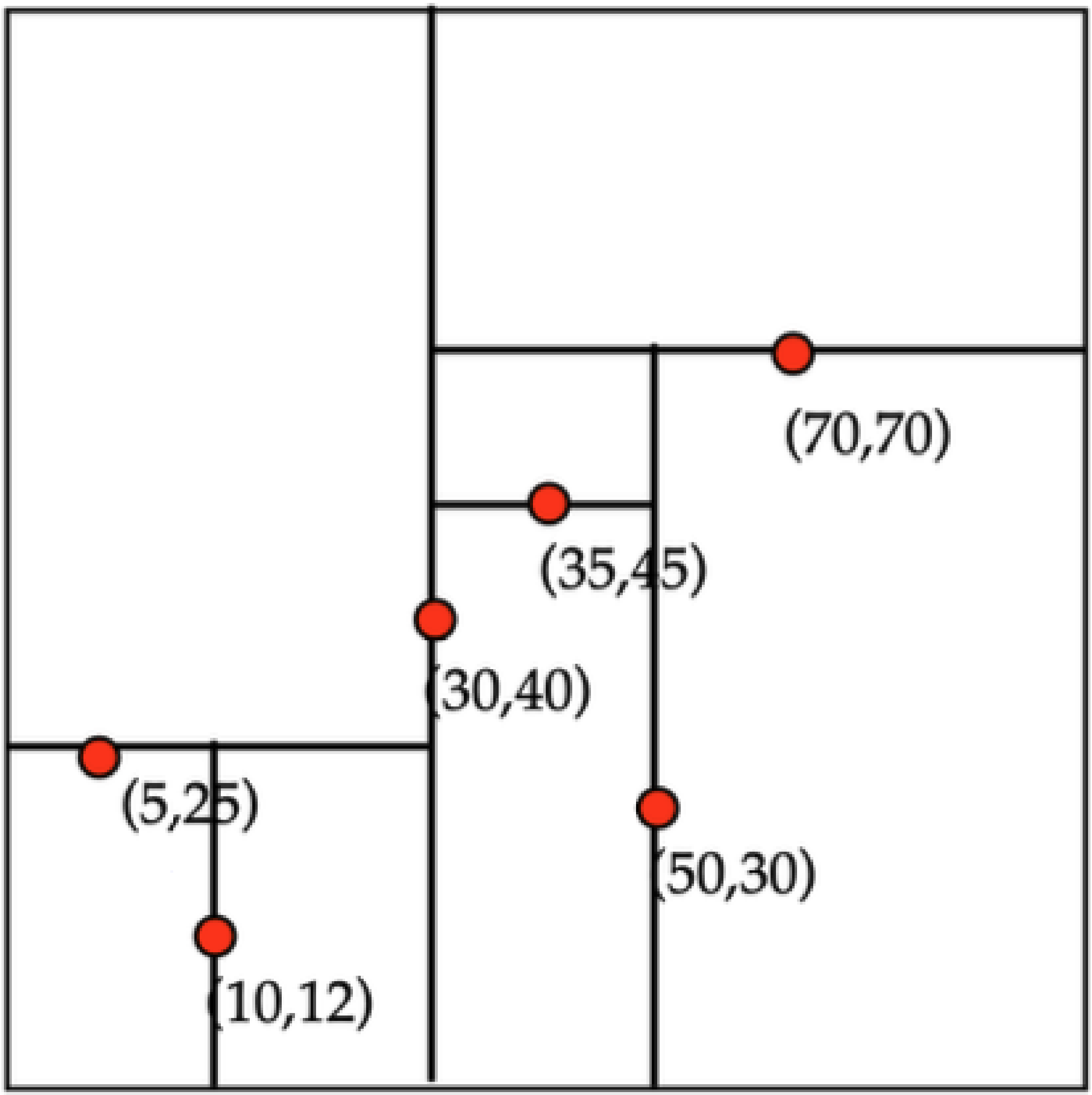
D Dimension

What is the time complexity of classifying that red point?

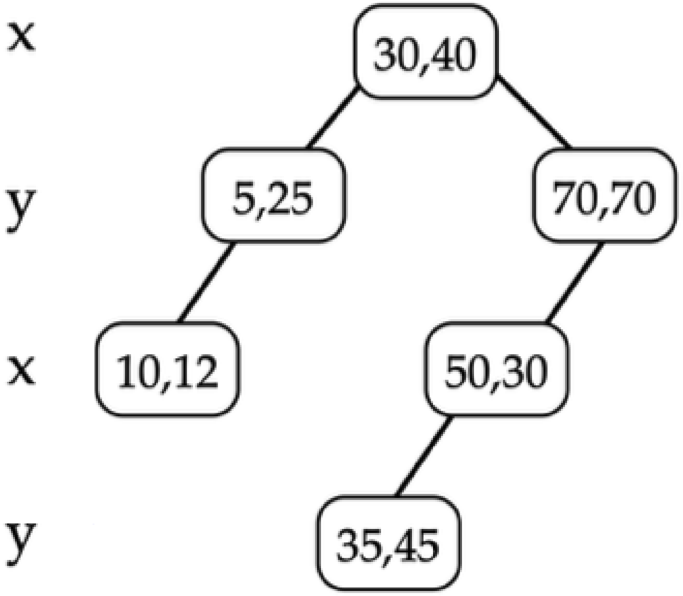
O(ND)

Everytime, we need to compare with every training points

KNN – KD Tree to Optimize Time Complexity



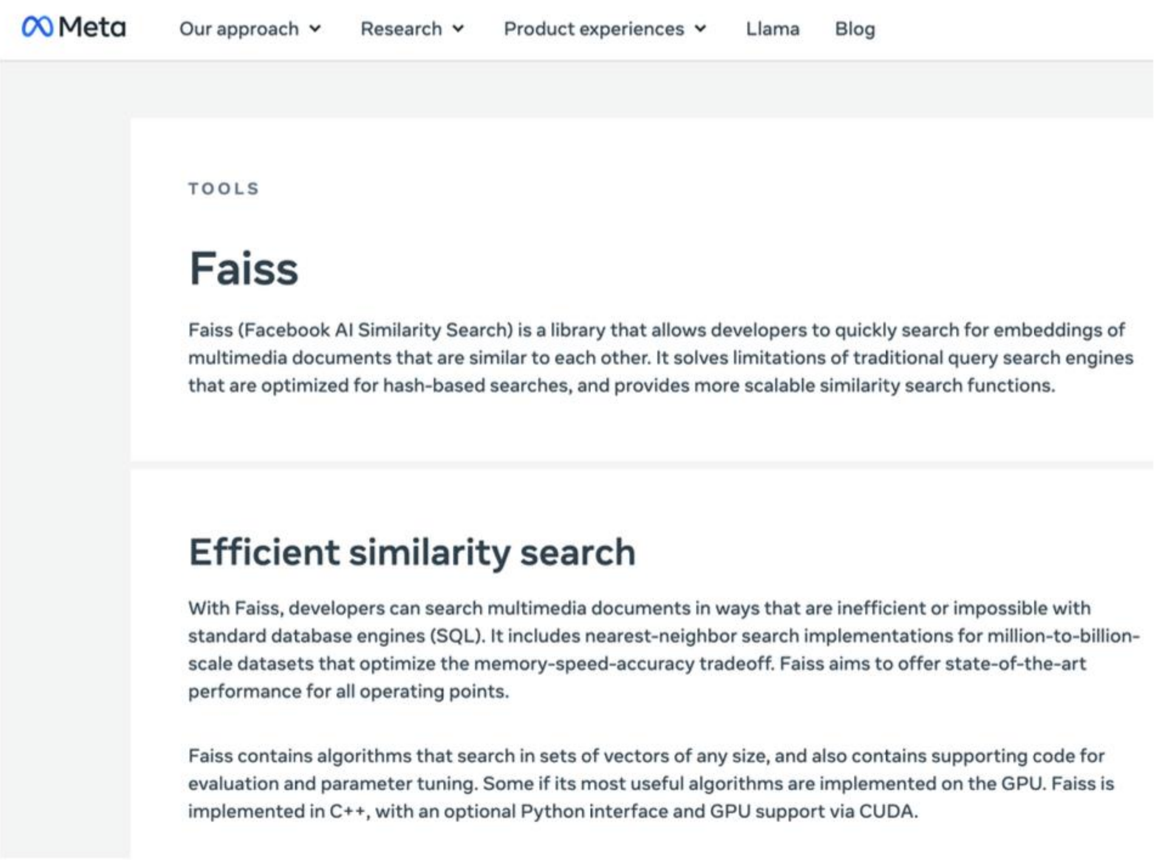
insert: (30,40), (5,25), (10,12), (70,70), (50,30), (35,45)



What is the time complexity of classifying that red point?

$$O(DlogN)$$

KNN – KD Tree and Faiss to Optimize Time Complexity



TOOLS

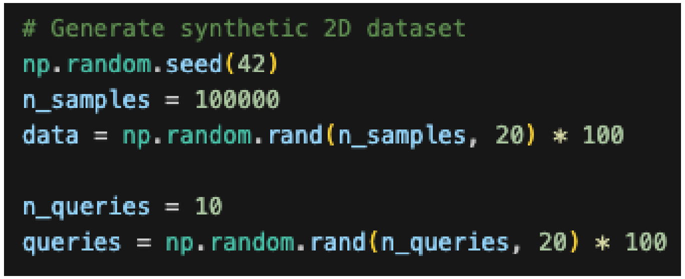
Faiss

Faiss (Facebook AI Similarity Search) is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves limitations of traditional query search engines that are optimized for hash-based searches, and provides more scalable similarity search functions.

Efficient similarity search

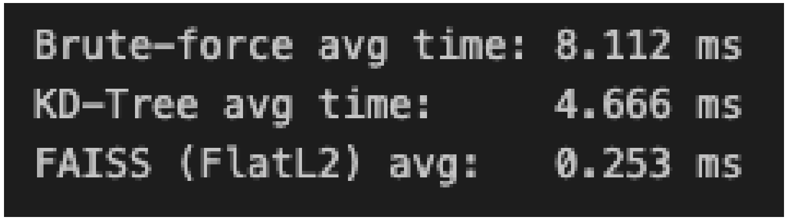
With Faiss, developers can search multimedia documents in ways that are inefficient or impossible with standard database engines (SQL). It includes nearest-neighbor search implementations for million-to-billion-scale datasets that optimize the memory-speed-accuracy tradeoff. Faiss aims to offer state-of-the-art performance for all operating points.

Faiss contains algorithms that search in sets of vectors of any size, and also contains supporting code for evaluation and parameter tuning. Some if its most useful algorithms are implemented on the GPU. Faiss is implemented in C++, with an optional Python interface and GPU support via CUDA.



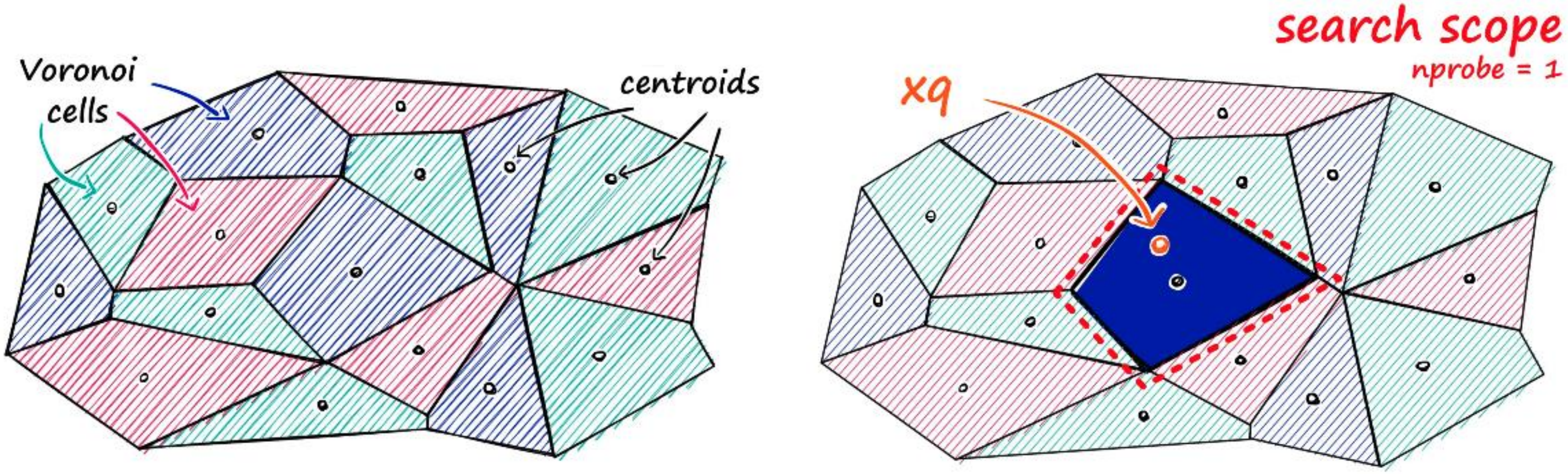
```
# Generate synthetic 2D dataset
np.random.seed(42)
n_samples = 100000
data = np.random.rand(n_samples, 20) * 100

n_queries = 10
queries = np.random.rand(n_queries, 20) * 100
```



```
Brute-force avg time: 8.112 ms
KD-Tree avg time:     4.666 ms
FAISS (FlatL2) avg:   0.253 ms
```

KNN – K-Means Clustering to Help



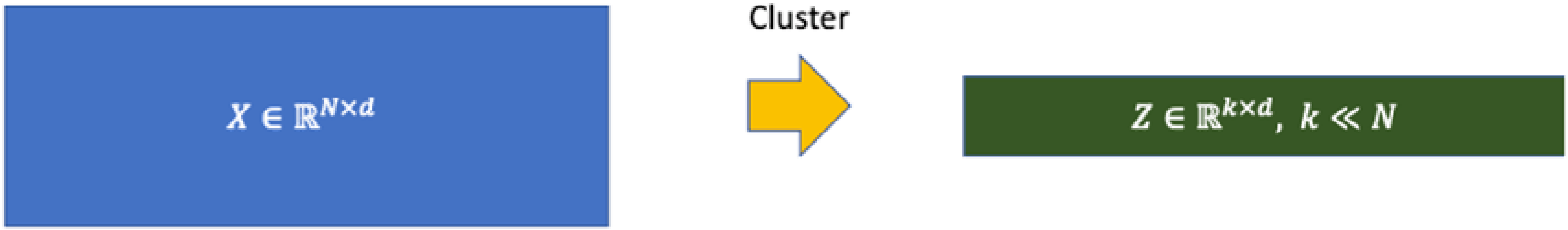
Given a query q, instead of comparing with every other points, we only compare with points in certain regions

But how to derive regions and their point belonging relations?

Question Time!

K-Means Clustering

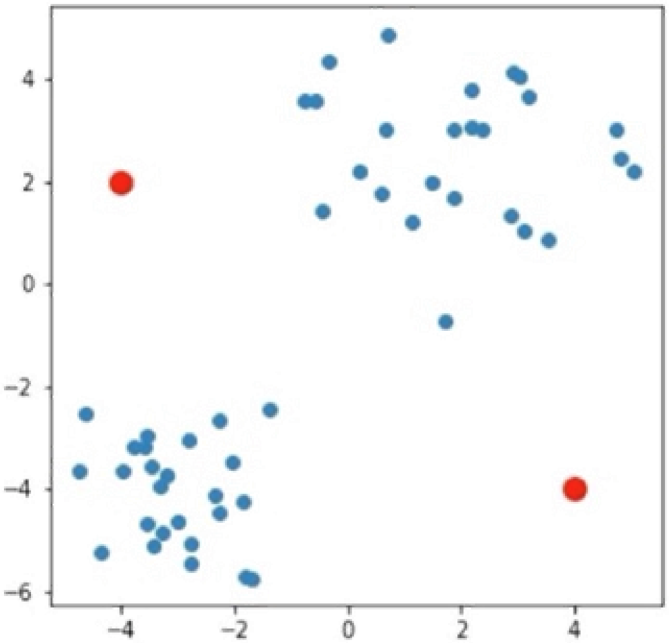
- Clustering:

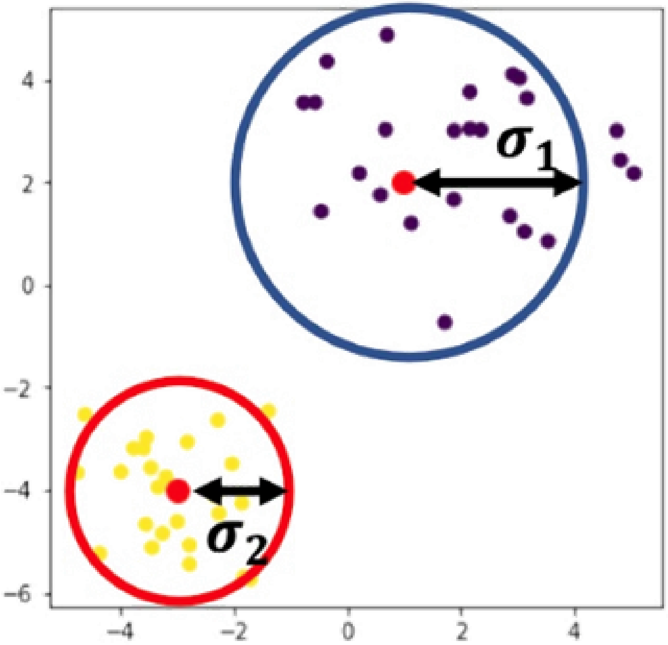


$X \in \mathbb{R}^{N \times d}$

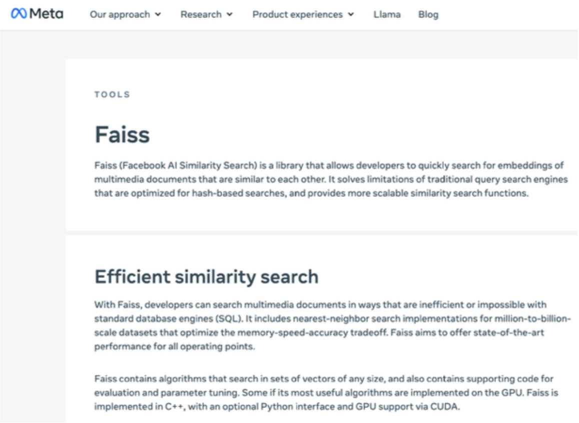
Cluster

$Z \in \mathbb{R}^{k \times d},\ k \ll N$





Applications



TOOLS

Faiss

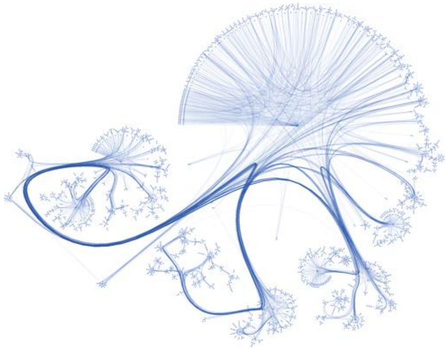
Faiss (Facebook AI Similarity Search) is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves limitations of traditional query search engines that are optimized for hash-based searches, and provides more scalable similarity search functions.

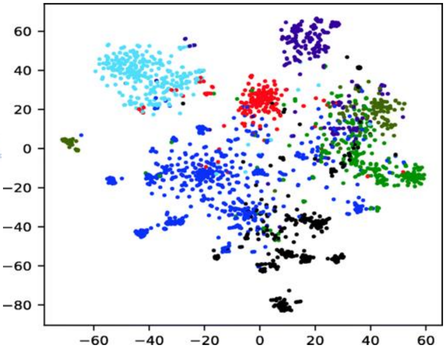
Efficient similarity search

With Faiss, developers can search multimedia documents in ways that are inefficient or impossible with standard database engines (SQL). It includes nearest-neighbor search implementations for million-to-billion-scale datasets that optimize the memory-speed-accuracy tradeoff. Faiss aims to offer state-of-the-art performance for all operating points.

Faiss contains algorithms that search in sets of vectors of any size, and also contains supporting code for evaluation and parameter tuning. Some if its most useful algorithms are implemented on the GPU. Faiss is implemented in C++, with an optional Python interface and GPU support via CUDA.

FAISS
Scalable Search With Facebook AI





Quickly Calculate KNN

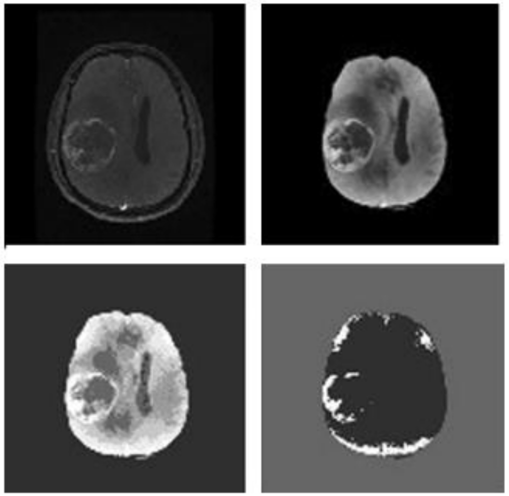
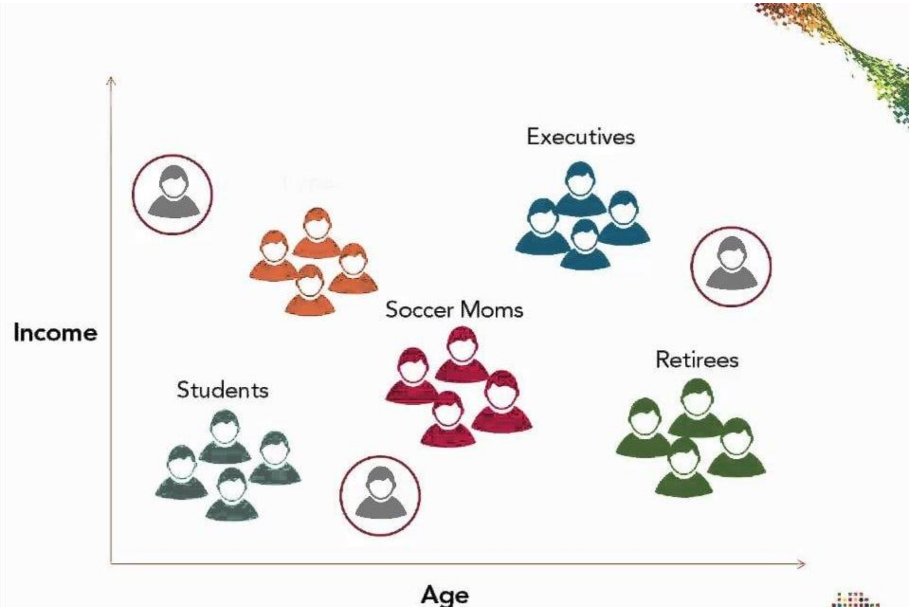


Fig. 3 Raw MR image 1 preprocessed image, clustered classes with k-Means and clustered classes with FCM.



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \dots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$

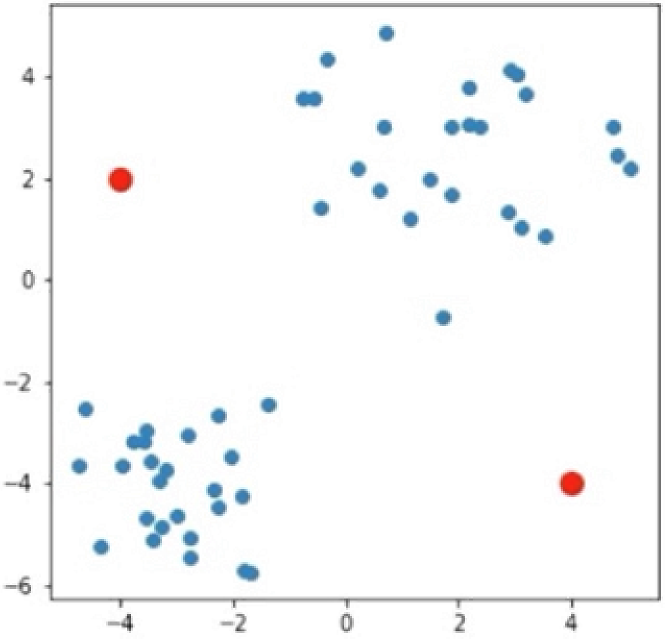
Procedure:

1. Initialize with K random centroids (i.e., cluster center),$\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - Assign a cluster to every sample $x^{(i)}$:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - Update centroids according to clusters:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \dots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$

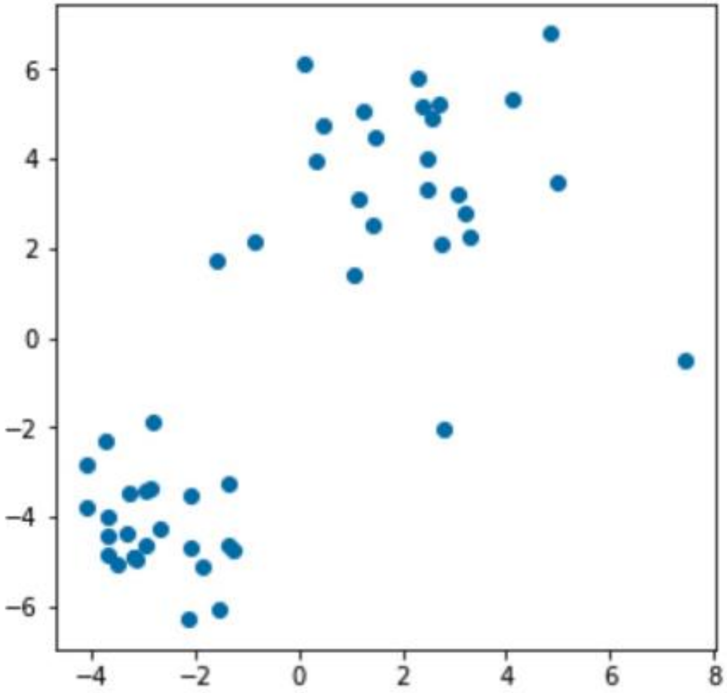
Procedure:

1. Initialize with K random centroids (i.e., cluster center),$\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\}x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

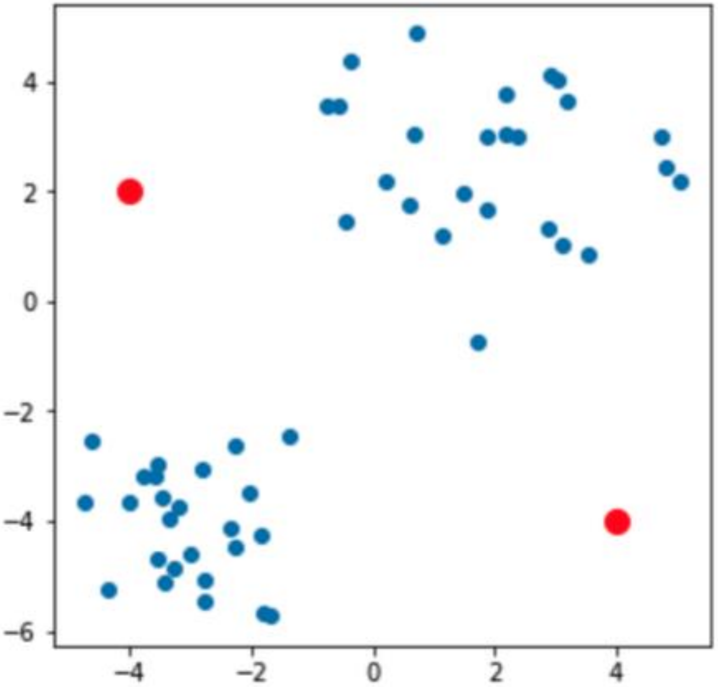
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

Procedure:

1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:
     $$c^{(i)} = argmin_k\ dist(x^{(i)}, \mu^{(k)})$$
   - For each k, set:
     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

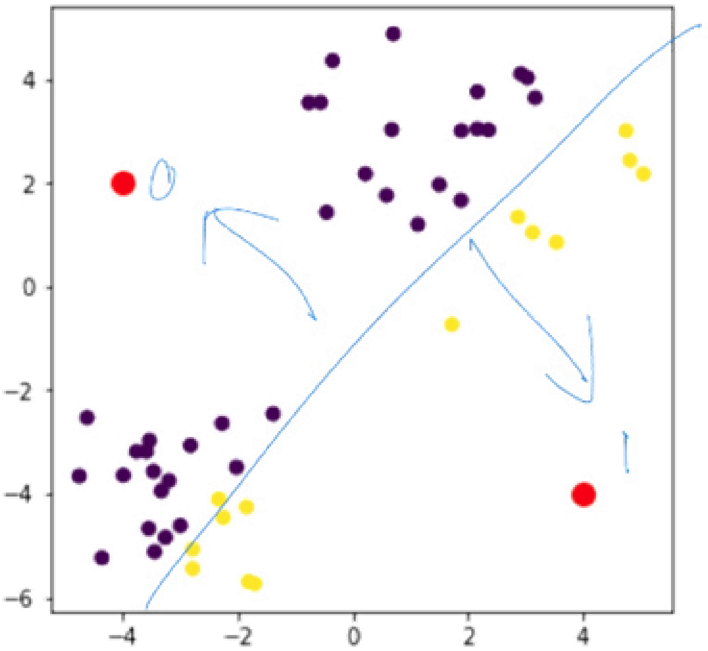
Procedure:

1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

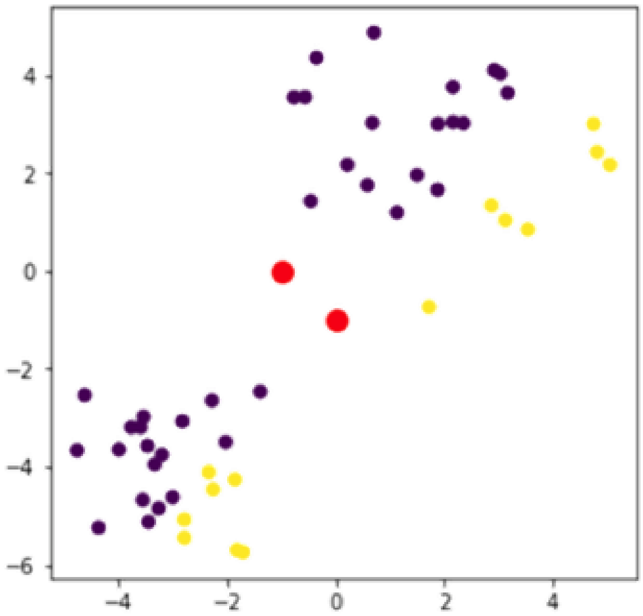
Procedure:

1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. **Repeat until convergence:**
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\}x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

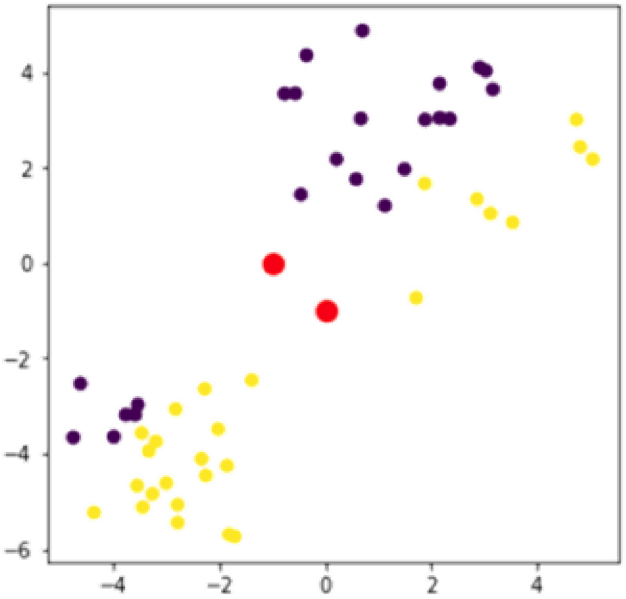
Procedure:

1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \, dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\}x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \dots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$

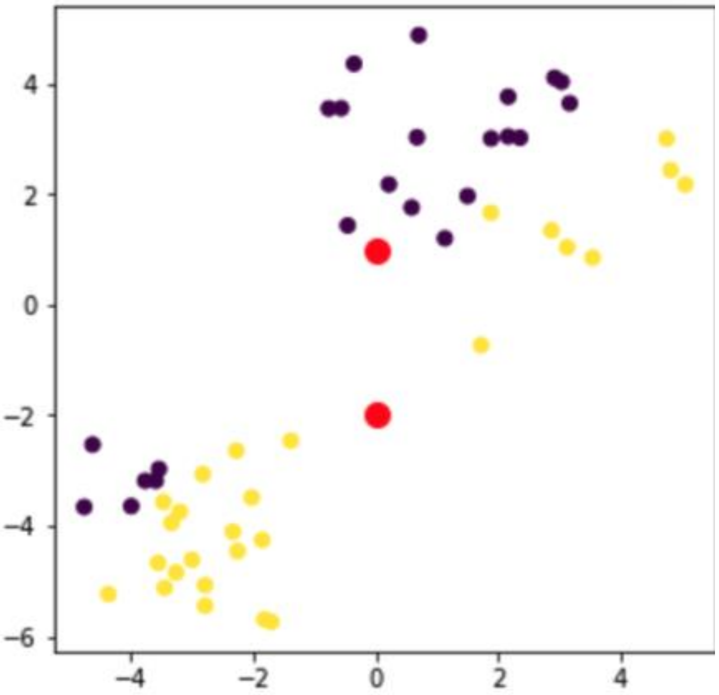
Procedure:

1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

  Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

  Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

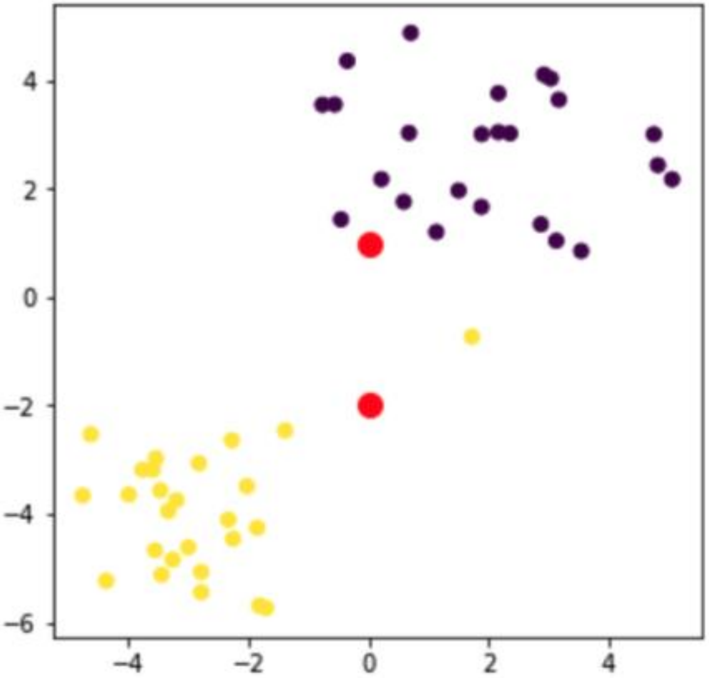
  Procedure:

  1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
  2. Repeat until convergence:
     - For every sample $x^{(i)}$, set:

       $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

     - For each k, set:

       $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

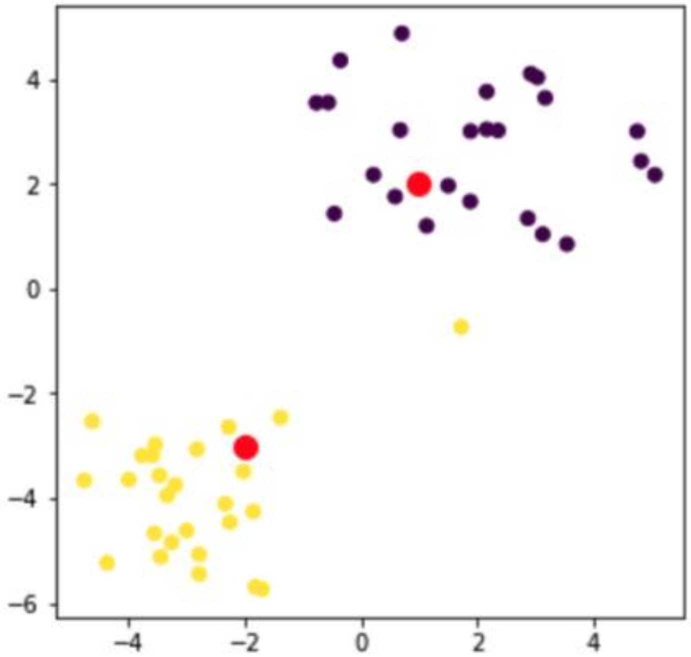
Procedure:

1. Initialize with K random centroids (i.e., cluster center), $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. **Repeat until convergence:**
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

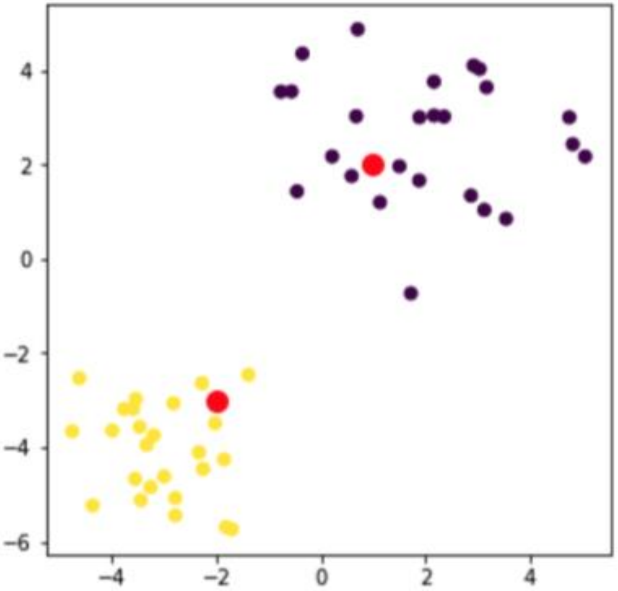
Procedure:

1. Initialize with K random centroids (i.e., cluster center),$\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

$$c^{(i)} = argmin_k\ dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

$$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\}x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$

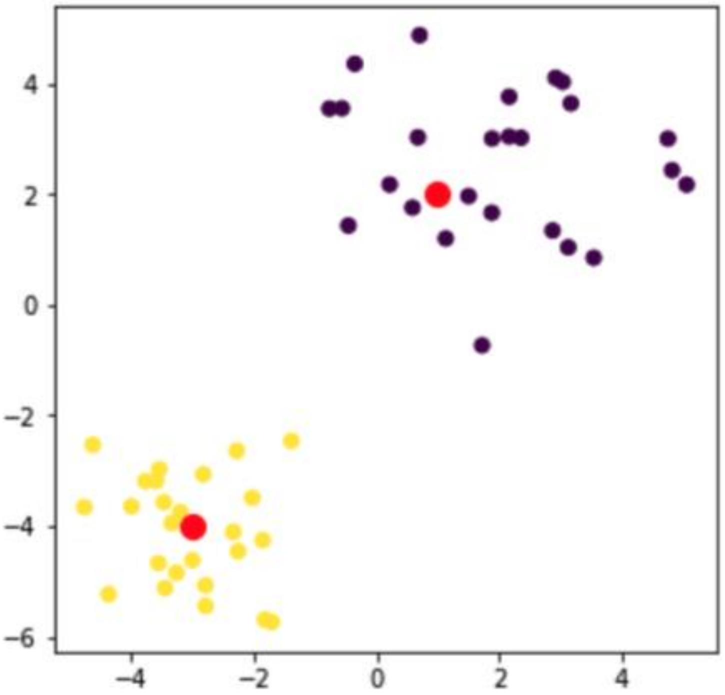
Procedure:

1. Initialize with K random centroids (i.e., cluster center),$\{\mu^{(1)}, \ldots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k \; dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\}x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



K-Means Clustering

- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

Input: Data $\{x^{(1)}, \dots, x^{(N)} \in \mathbb{R}^d\}$

Output: Centroids $\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$

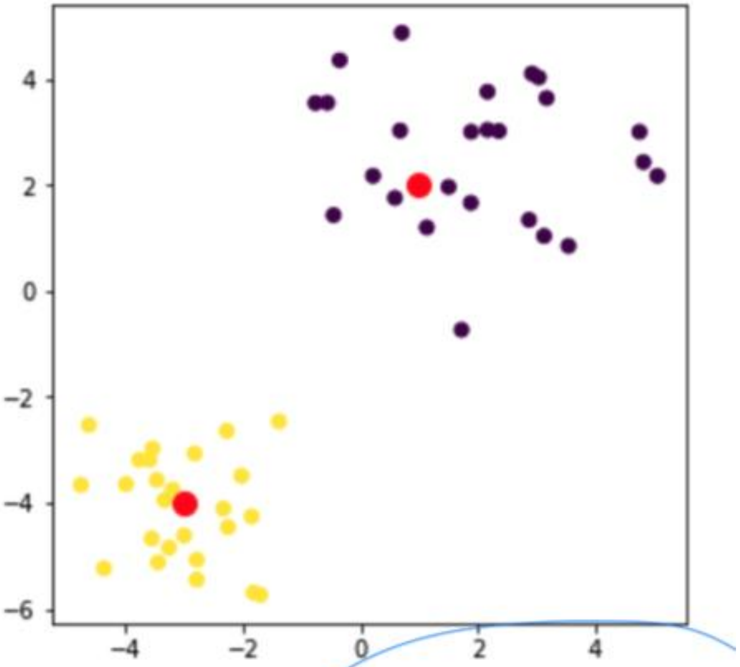
Procedure:

1. Initialize with K random centroids (i.e., cluster center),$\{\mu^{(1)}, \dots, \mu^{(K)} \in \mathbb{R}^d\}$.
2. Repeat until convergence:
   - For every sample $x^{(i)}$, set:

     $$c^{(i)} = argmin_k\ dist(x^{(i)}, \mu^{(k)})$$

   - For each k, set:

     $$\mu^{(k)} = \frac{\sum_{i=1}^{N} 1\{c^{(i)}=k\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)}=k\}}$$



Converged! Nothing will change.

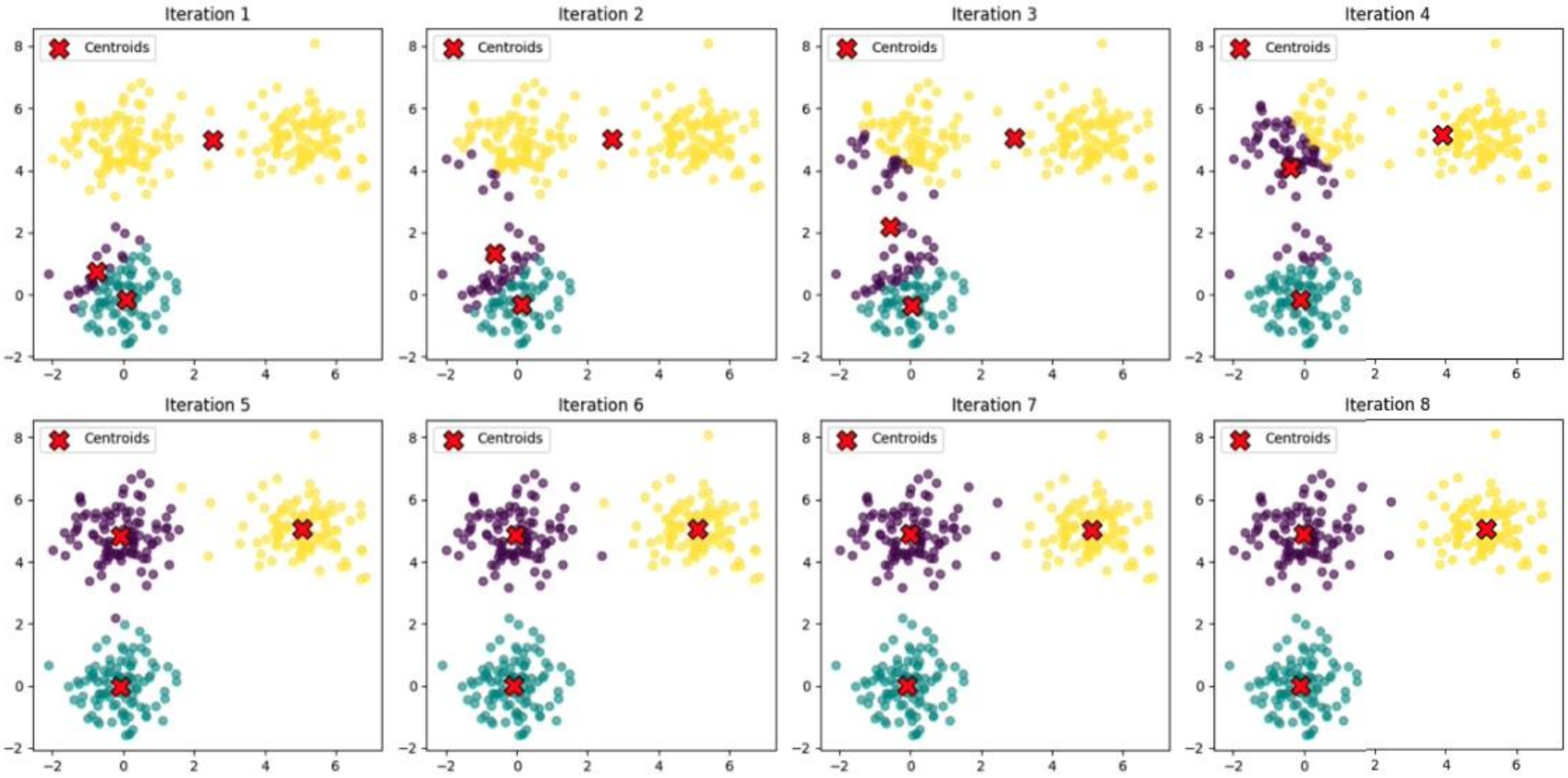
K-Means Clustering

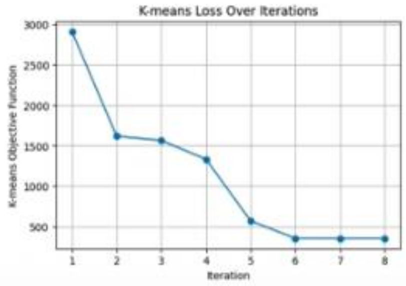
- A common objective function (used with Euclidean distance measure) is Sum of Squared Error (SSE)
  - For each point, the error is the distance to the nearest cluster center
  - To get SSE, we square these errors and sum them.

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

  - *x* is a data point in cluster C_i and m_i is the centroid (mean) for cluster C_i
  - SSE improves in each iteration of K-means until it reaches a local or global minima.

K-Means Clustering





K-Means Clustering

1. **Guaranteed to Converge in a finite number of iterations**

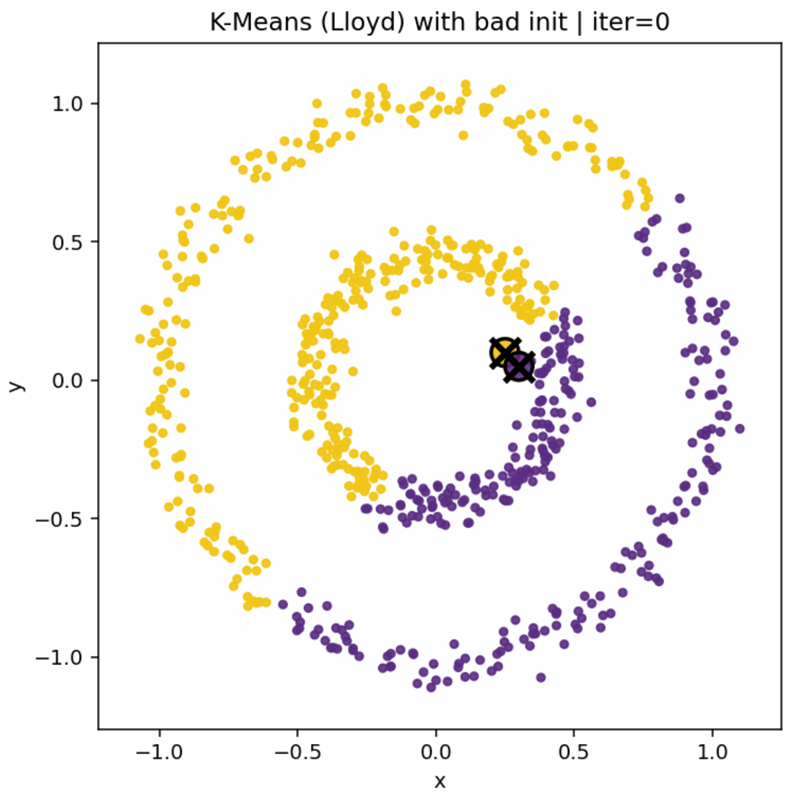
2. **The converging point really depends on the initial centroids**

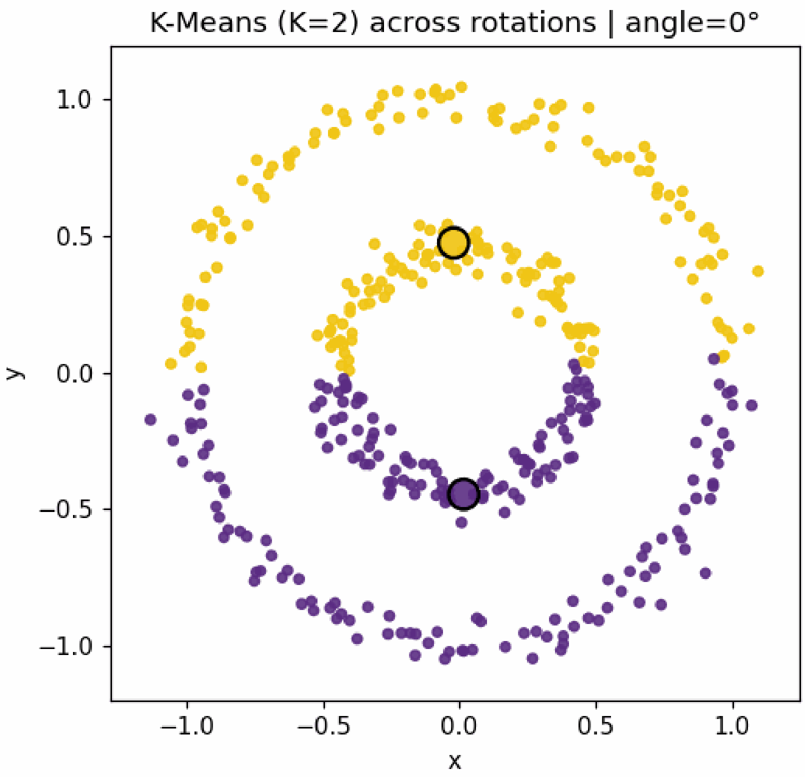
   Converge does not mean you get a good solution

   Converge does not mean you get the same outcome

3. **Running time**
   (1) Assign data points to closest centroid:
   (2) Change the cluster center:

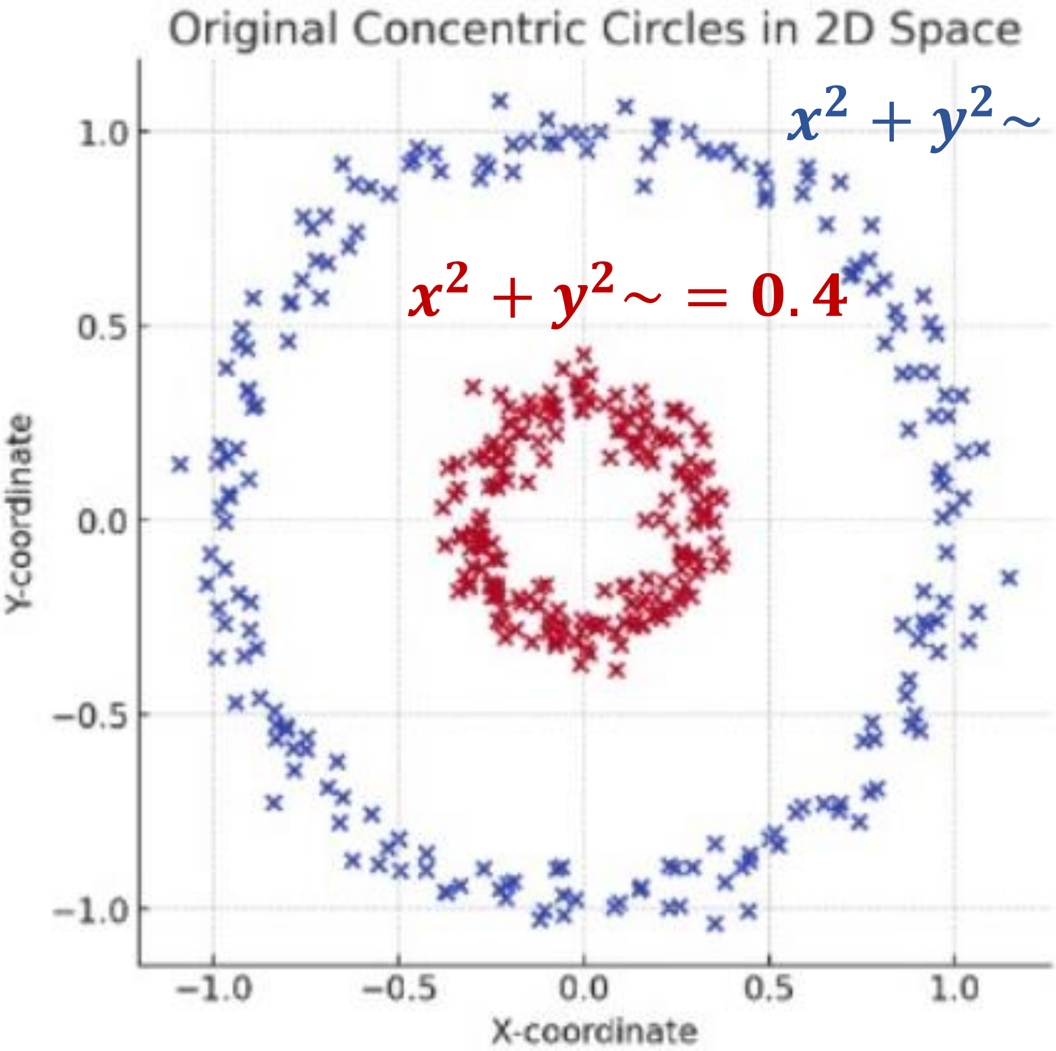
K-Means Clustering





Why only using two points in this plane cannot work?

K-Means Clustering



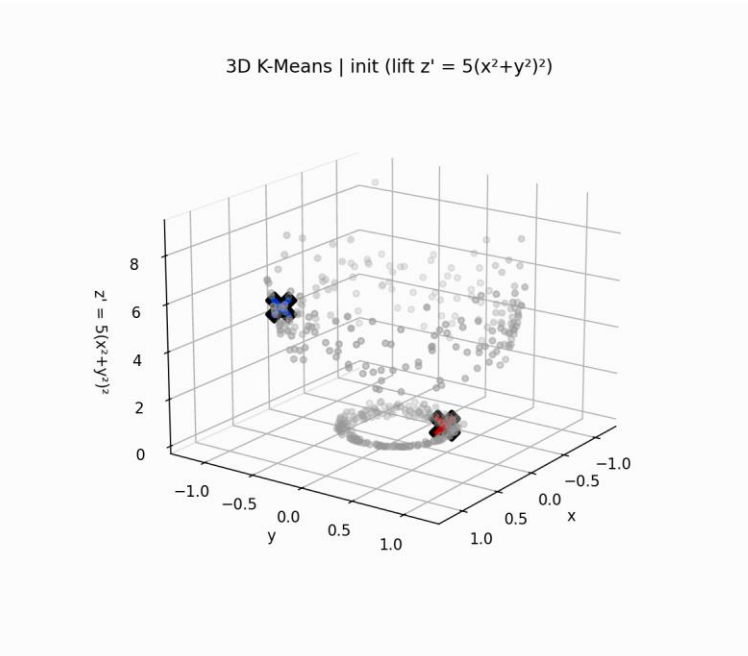
$x^2 + y^2 \sim = 1$

$x^2 + y^2 \sim = 0.4$

$$[x, y, z]$$

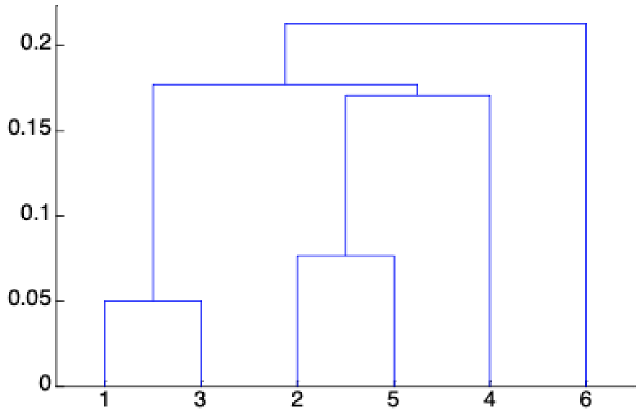
$$\downarrow$$

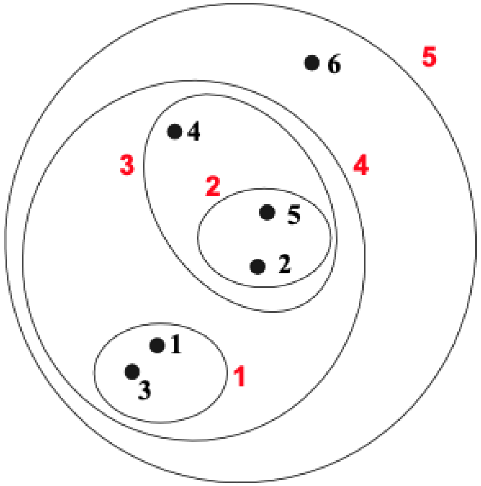
$$[x, y, z = 5(x^2 + y^2)]$$



Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  - A tree like diagram that records the sequences of merges or splits





K-Means Clustering

- Do not have to assume any particular number of clusters
  - Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level
- They may correspond to meaningful taxonomies
  - Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, …)

K-Means Clustering

- Two main types of hierarchical clustering
  - Agglomerative:
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - Divisive:
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains an individual point (or there are k clusters)

K-Means Clustering

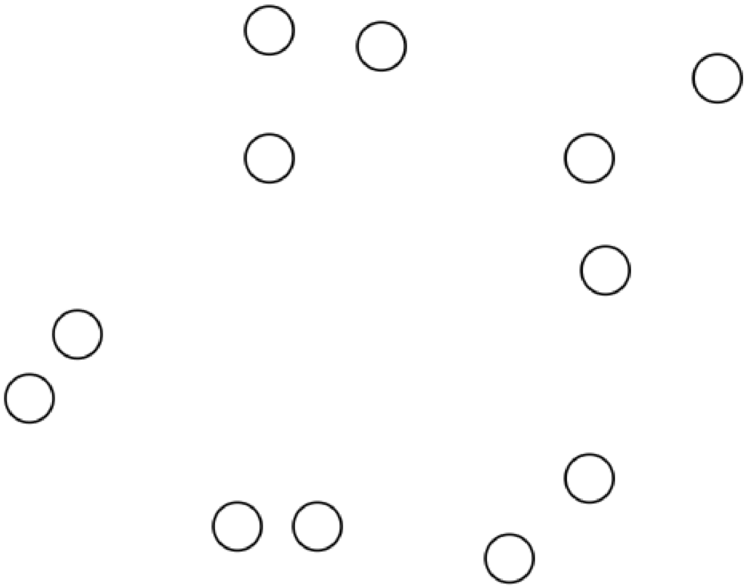
- **Key Idea: Successively merge closest clusters**
- **Basic algorithm**

1. **Compute the proximity matrix**
2. **Let each data point be a cluster**
3. **Repeat**
4. **Merge the two closest clusters**
5. **Update the proximity matrix**
6. **Until only a single cluster remains**

- **Key operation is the computation of the proximity of two clusters**
  - **Different approaches to defining the distance between clusters distinguish the different algorithms**

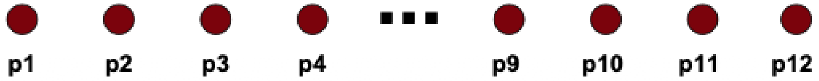
K-Means Clustering

- Start with clusters of individual points and a proximity matrix



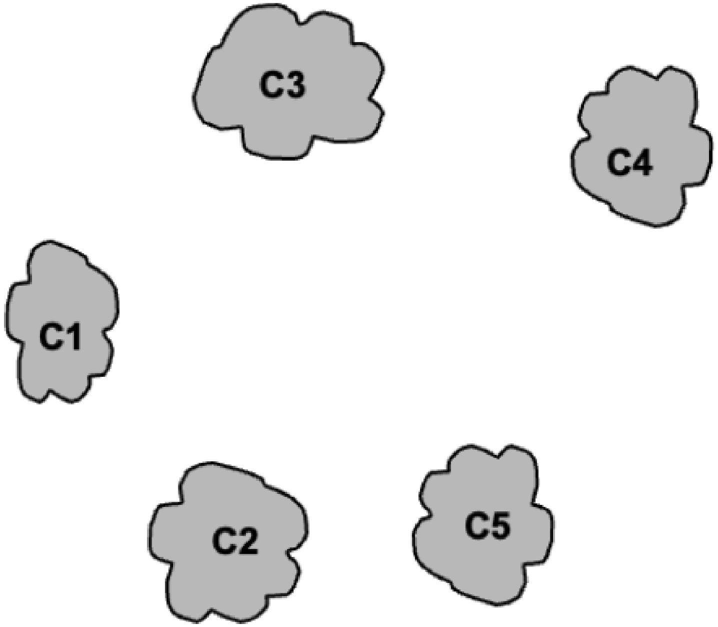
	p1	p2	p3	p4	p5	.
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix



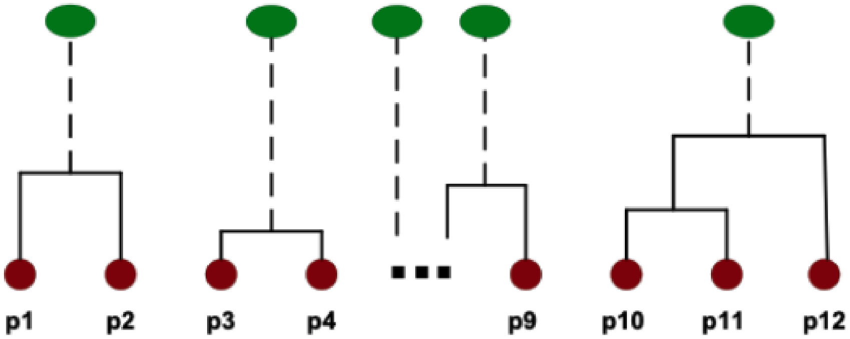
K-Means Clustering

- After some merging steps, we have some clusters



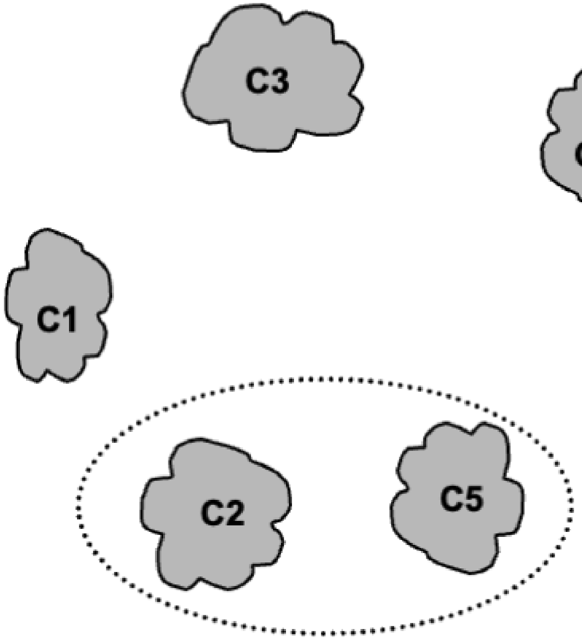
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

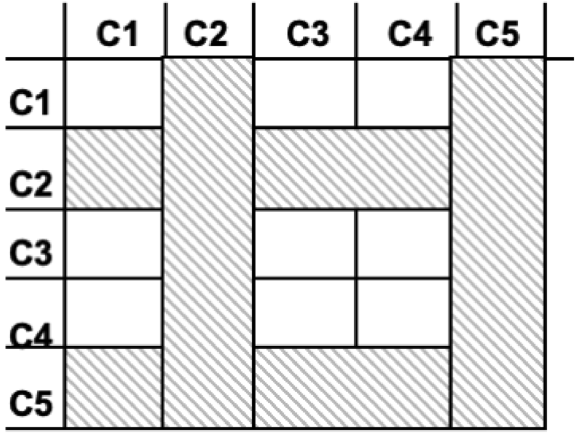
Proximity Matrix



K-Means Clustering

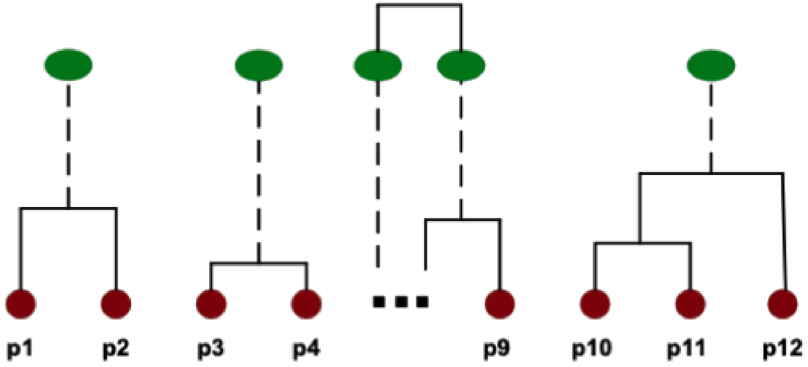
- We want to merge the two closest clusters (C2 and C5) and update the proximity matrix.





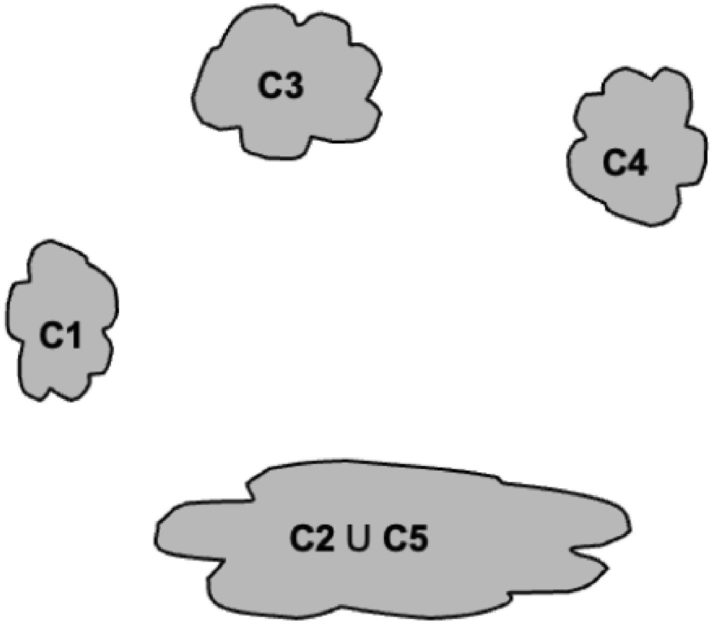
	C1	C2	C3	C4	C5
C1		▨			▨
C2	▨	▨	▨	▨	▨
C3		▨			▨
C4		▨			▨
C5	▨	▨	▨	▨	▨

Proximity Matrix



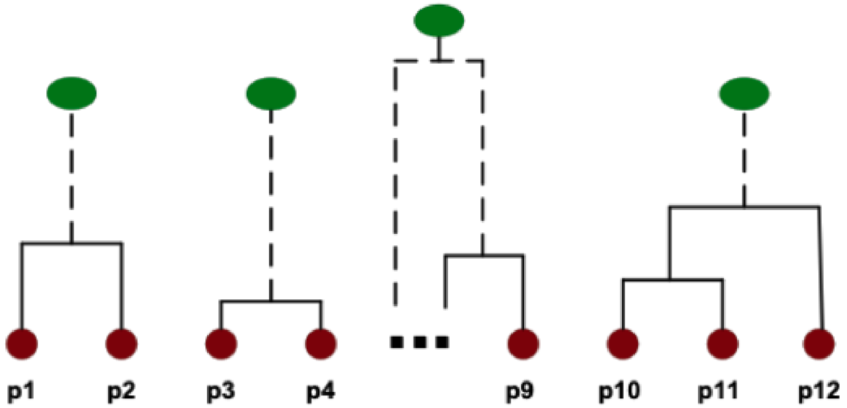
K-Means Clustering

- The question is “How do we update the proximity matrix?”

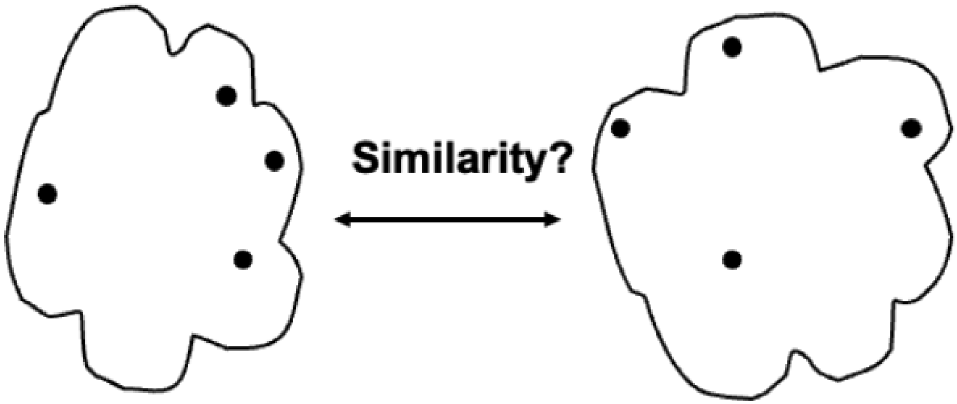


	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		

Proximity Matrix



K-Means Clustering

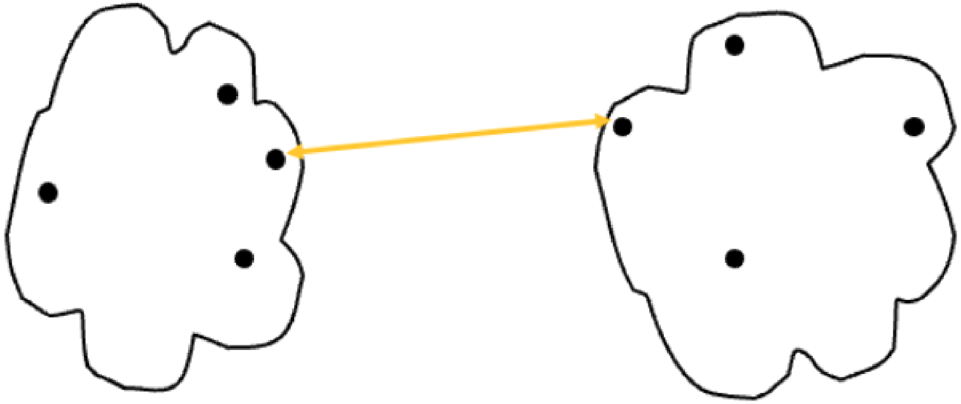


- MIN
- MAX
- Group Average
- Distance Between Centroids

	p1	p2	p3	p4	p5	.
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

K-Means Clustering



- MIN
- MAX
- Group Average
- Distance Between Centroids

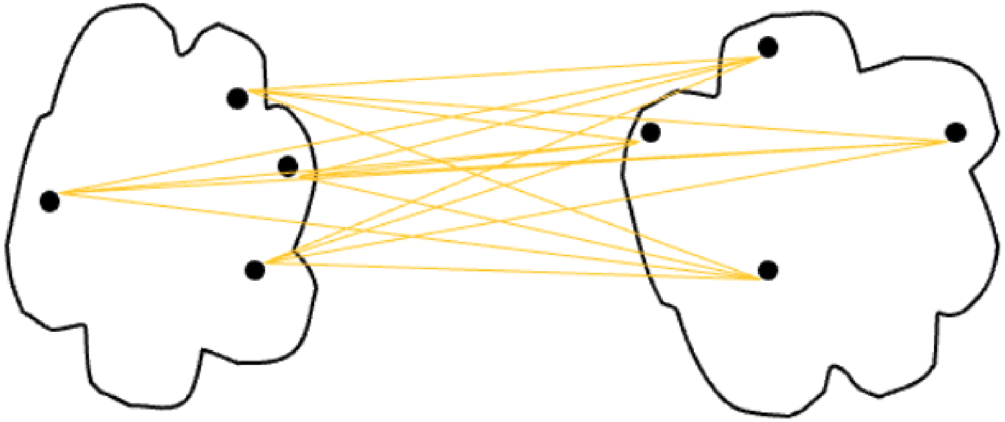
	p1	p2	p3	p4	p5	. .
p1						
p2						
p3						
p4						
p5						
.						

.

.

Proximity Matrix

K-Means Clustering

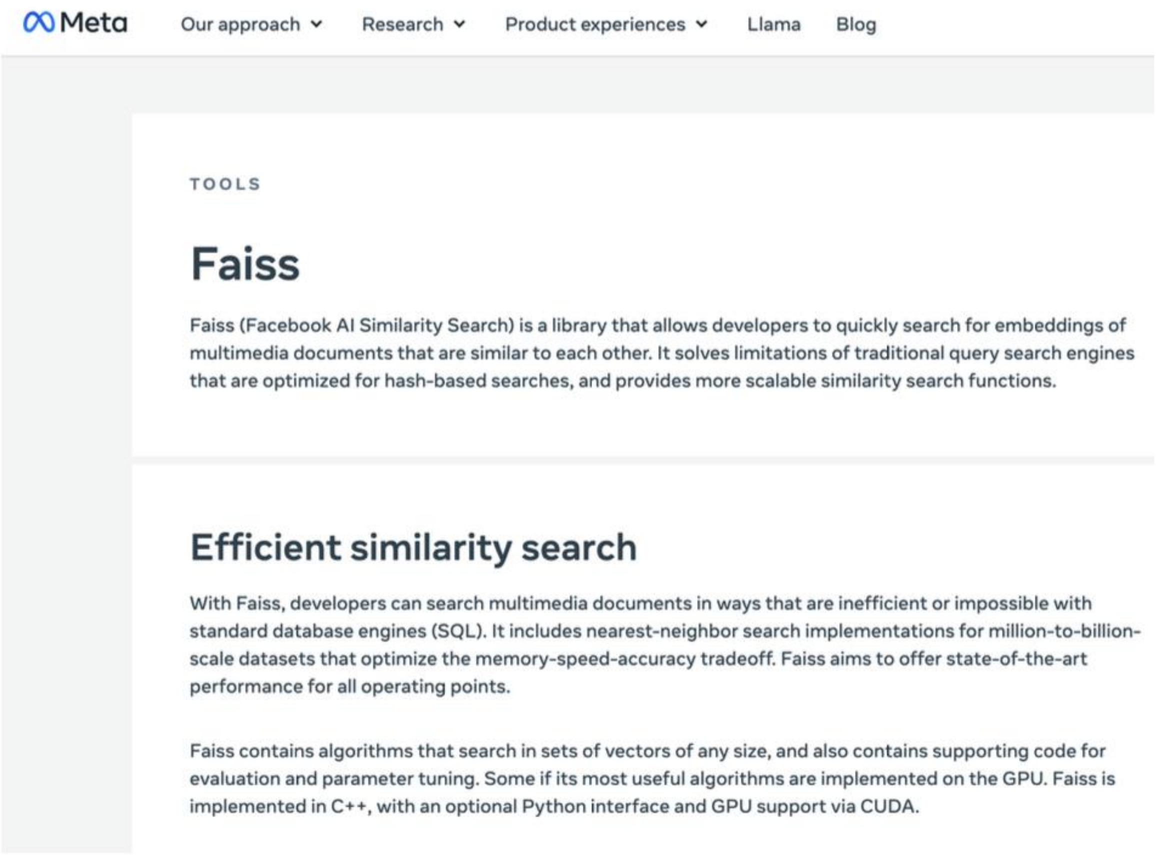


- MIN
- MAX
- Group Average
- Distance Between Centroids

	p1	p2	p3	p4	p5	.
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

FAISS



TOOLS

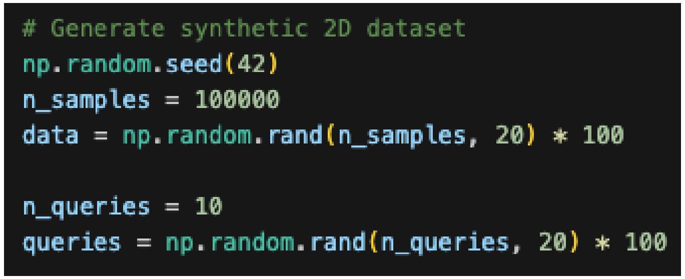
Faiss

Faiss (Facebook AI Similarity Search) is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves limitations of traditional query search engines that are optimized for hash-based searches, and provides more scalable similarity search functions.

Efficient similarity search

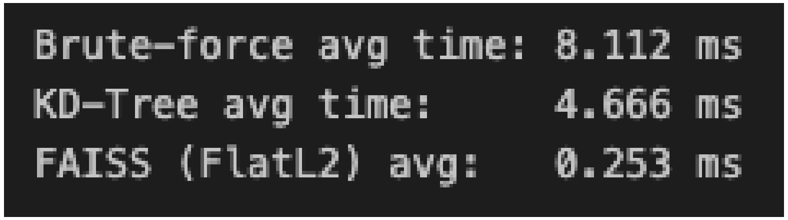
With Faiss, developers can search multimedia documents in ways that are inefficient or impossible with standard database engines (SQL). It includes nearest-neighbor search implementations for million-to-billion-scale datasets that optimize the memory-speed-accuracy tradeoff. Faiss aims to offer state-of-the-art performance for all operating points.

Faiss contains algorithms that search in sets of vectors of any size, and also contains supporting code for evaluation and parameter tuning. Some if its most useful algorithms are implemented on the GPU. Faiss is implemented in C++, with an optional Python interface and GPU support via CUDA.



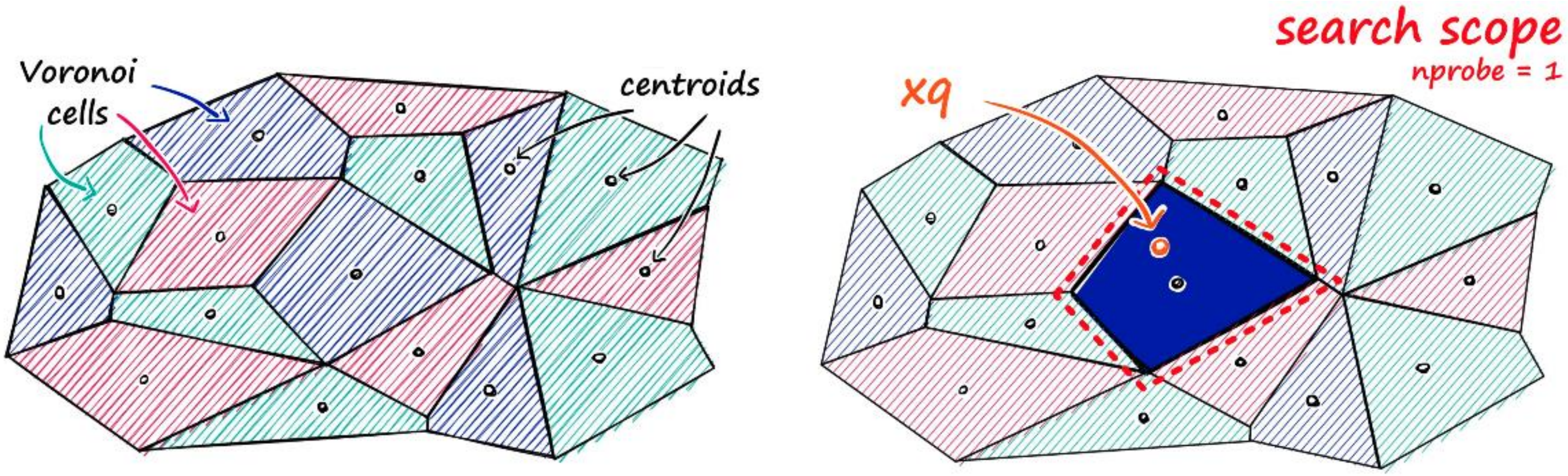
```
# Generate synthetic 2D dataset
np.random.seed(42)
n_samples = 100000
data = np.random.rand(n_samples, 20) * 100

n_queries = 10
queries = np.random.rand(n_queries, 20) * 100
```



```
Brute-force avg time: 8.112 ms
KD-Tree avg time:     4.666 ms
FAISS (FlatL2) avg:   0.253 ms
```

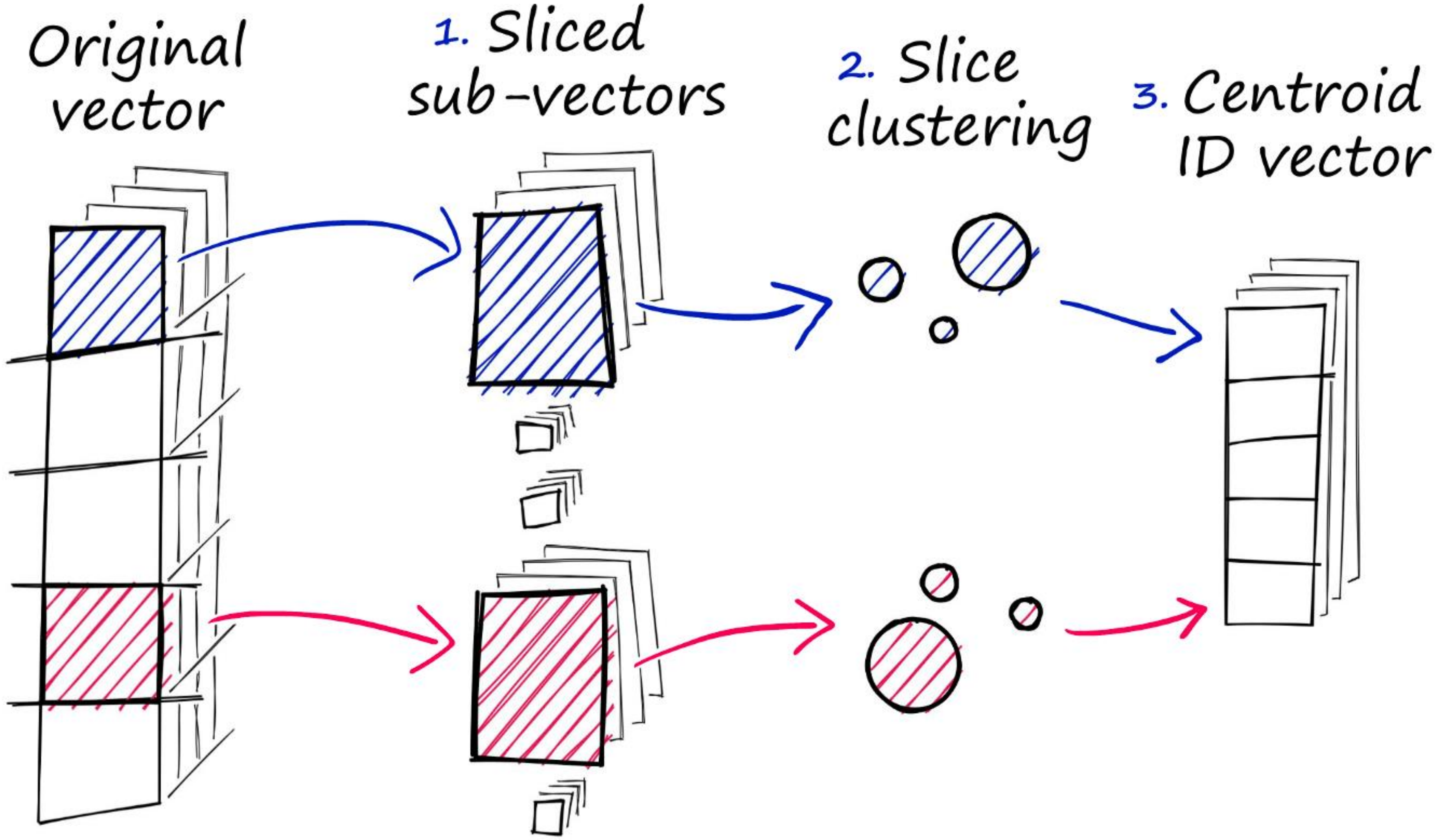
FAISS



Given a query q, instead of comparing with every other points, we only compare with points in certain regions

What if N is very small but d is very large?

FAISS


Original vector
1. Sliced sub-vectors
2. Slice clustering
3. Centroid ID vector

Question Time!