Data Mining: KNN and K-means clustering

Lecture Notes

Data Mining

https://ml-graph.github.io/winter-2025/

Yu Wang, Ph.D.

yuwang@uoregon.edu

Assistant Professor

Computer Science

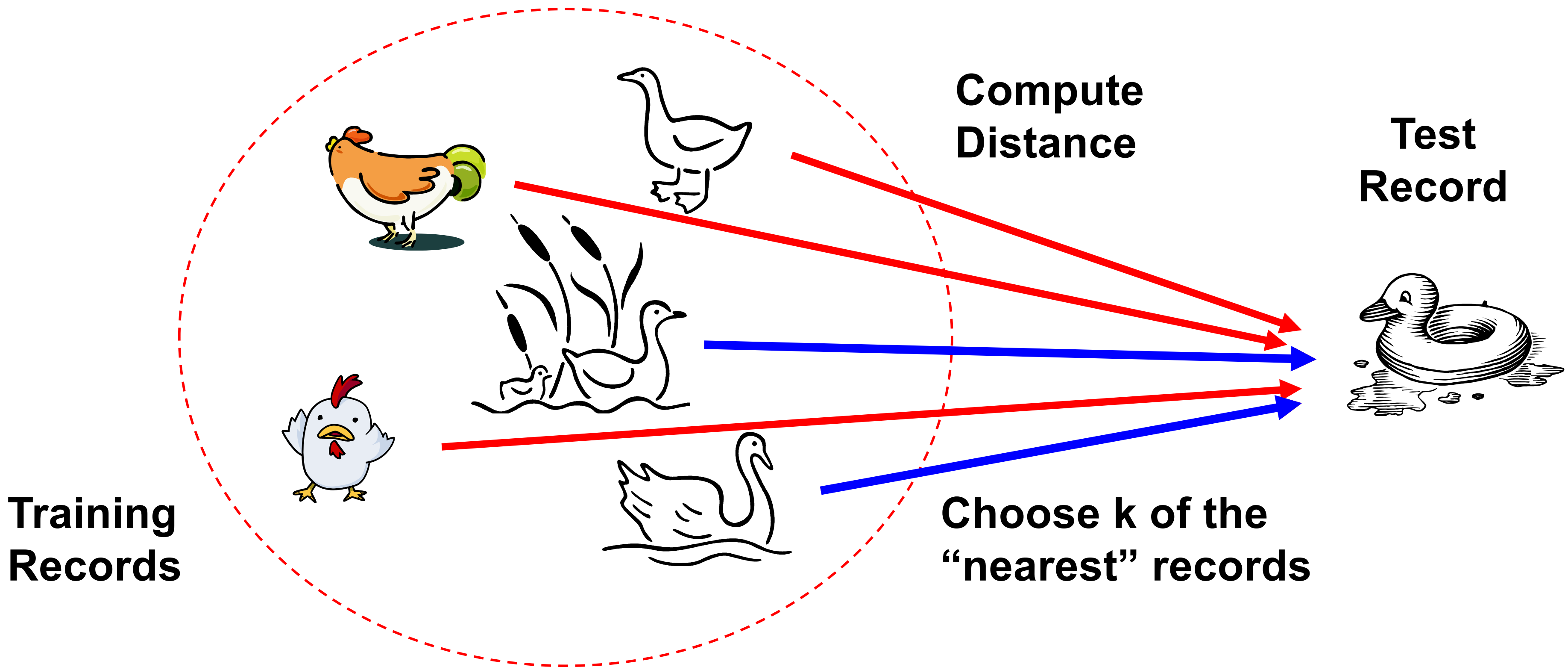
University of Oregon

CS 453/553 – Winter 2025

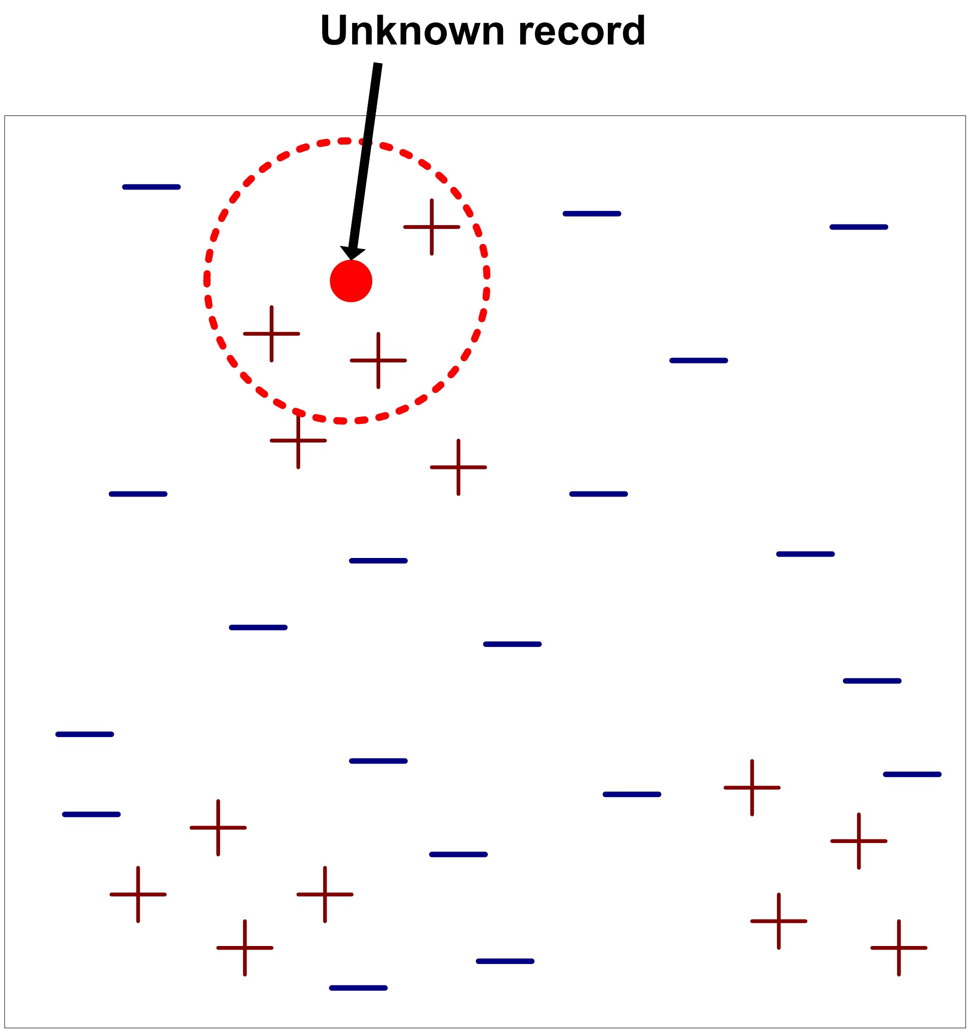
Course Lecture is very heavily based on
"Introduction to Data Mining"
by Tan, Steinbach, Karpatne, Kumar

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



Nearest Neighbor Classifiers



- Requires the following:
  - A set of labeled records
  - Proximity metric to compute distance/similarity between a pair of records
    - e.g., Euclidean distance
  - The value of *k*, the number of nearest neighbors to retrieve
  - A method for using class labels of K nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Nearest Neighbor Classifiers

- Take the majority vote of class labels among the k-nearest neighbors
- Weight the vote according to distance
  - weight factor, $w = 1/d^2$

Nearest Neighbor Classifiers

- **For documents, cosine is better than correlation or Euclidean**

1 1 1 1 1 1 1 1 1 1 1 0		0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1 1 1 1 1	vs	1 0 0 0 0 0 0 0 0 0 0 0

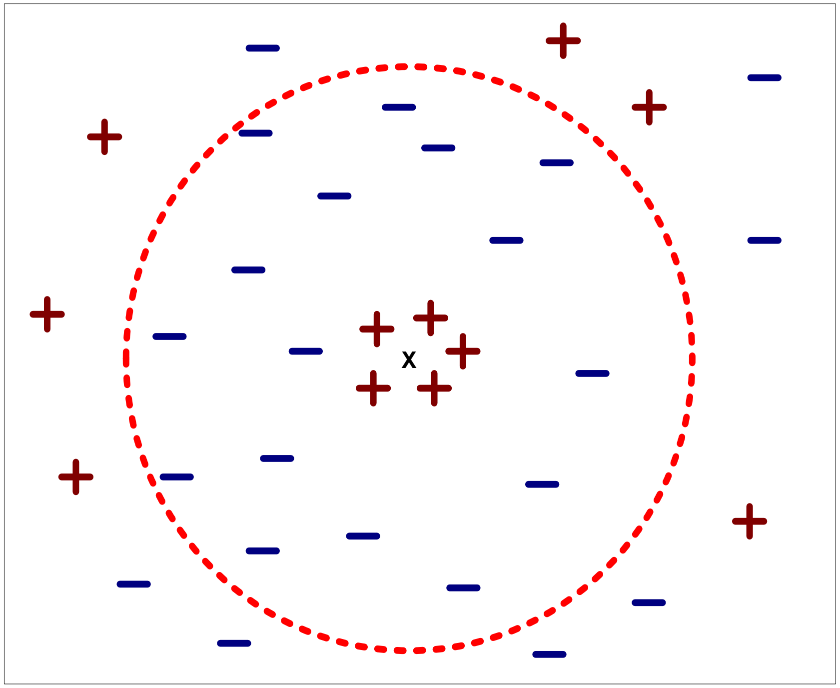
Euclidean distance = 1.4142 for both pairs, but the cosine similarity measure has different values for these pairs.

Nearest Neighbor Classifiers

- Data preprocessing is often required
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
    - Example:
      - height of a person may vary from 1.5m to 1.8m
      - weight of a person may vary from 90lb to 300lb
      - income of a person may vary from $10K to $1M
  - Time series are often standardized to have 0 means a standard deviation of 1

Nearest Neighbor Classifiers

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



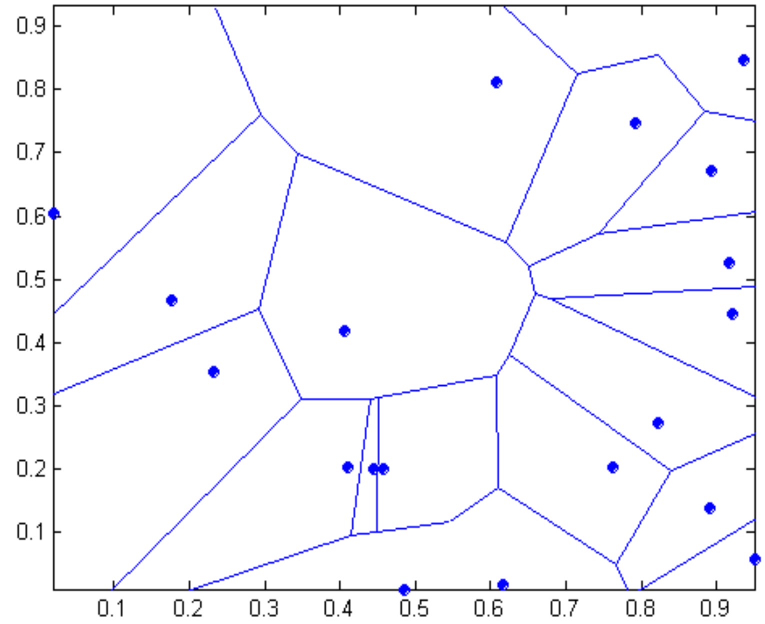
Nearest Neighbor Classifiers

- Nearest neighbor classifiers are local classifiers

Code Demo

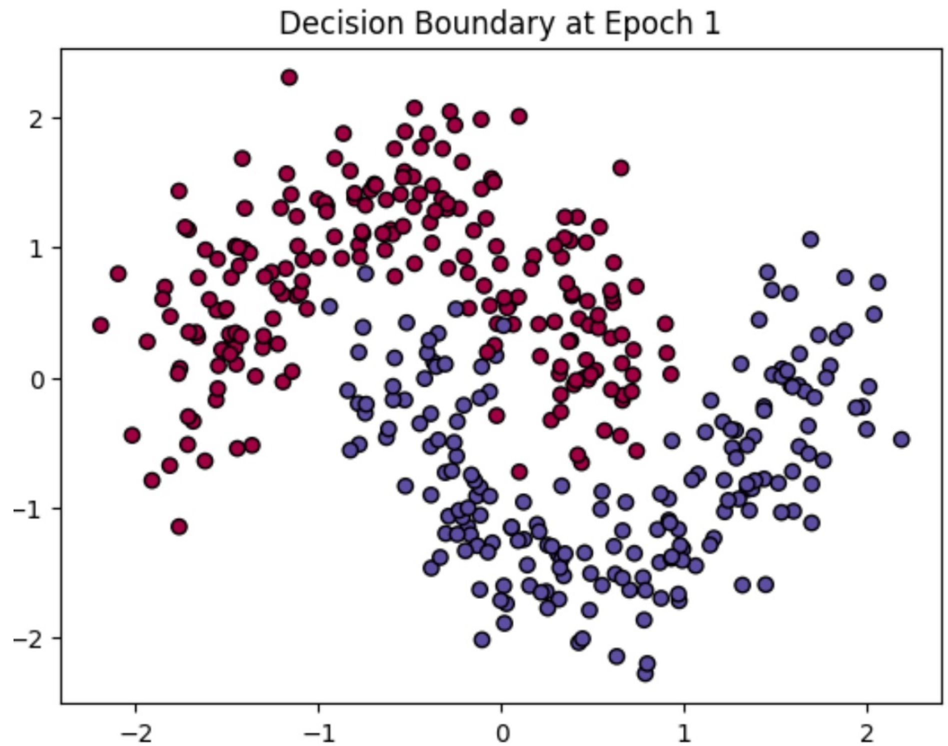
- They can produce decision boundaries of arbitrary shapes.

1-nn decision boundary is a Voronoi Diagram



Nearest Neighbor Classifiers

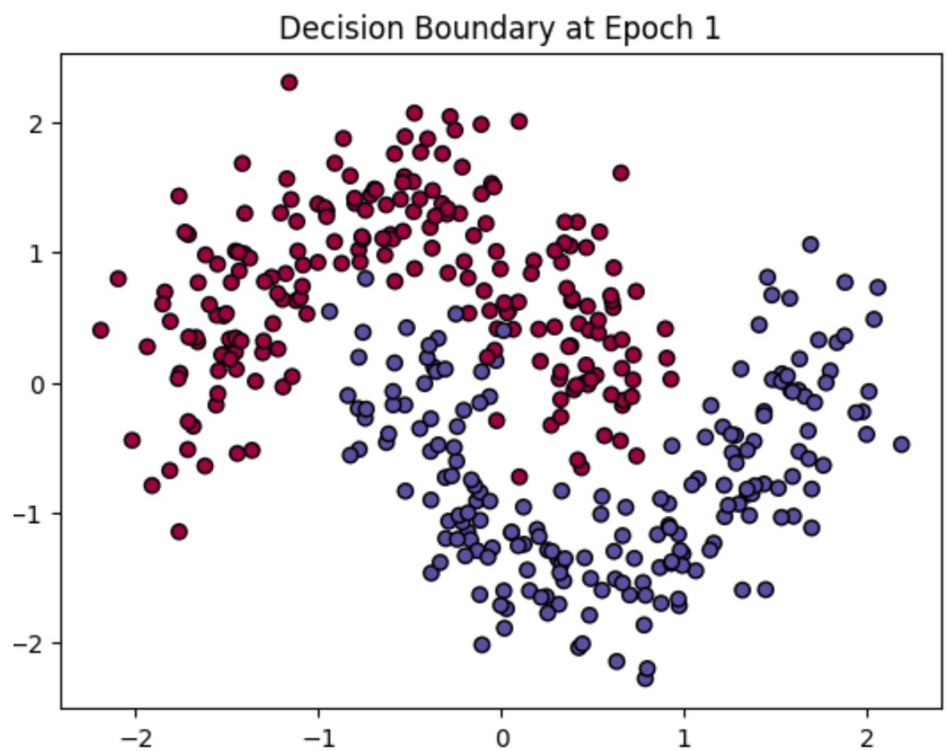
```python
# Generate synthetic 2D classification dataset
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

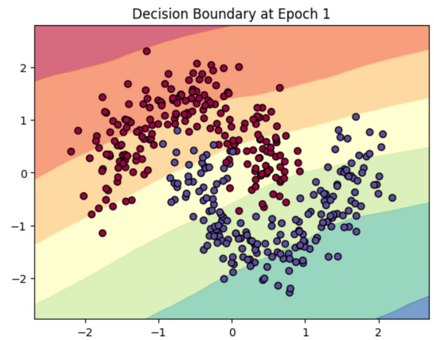


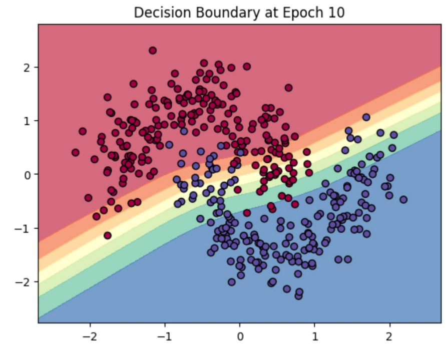

```python
# Define a simple MLP model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(2, 256)  # 2 input features, 10 hidden neurons
        self.fc2 = nn.Linear(256, 1)  # 1 output neuron (binary classification)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x
```

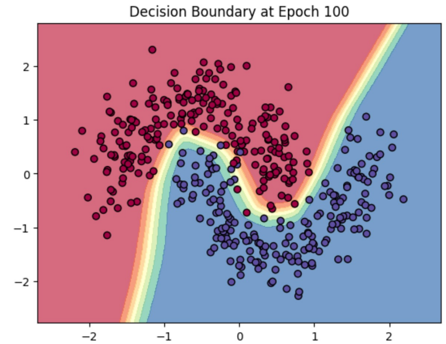
Nearest Neighbor Classifiers

```
# Generate synthetic 2D classification dataset
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



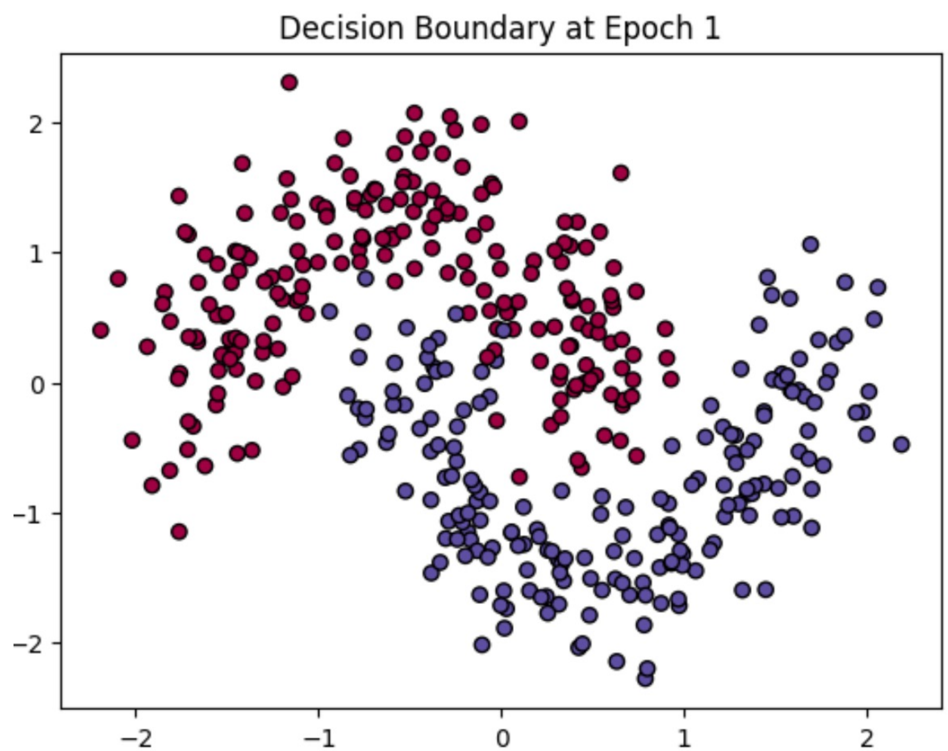


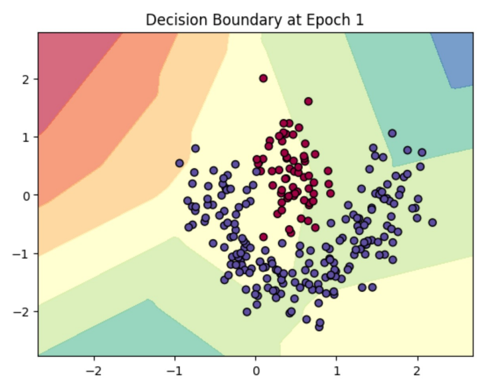


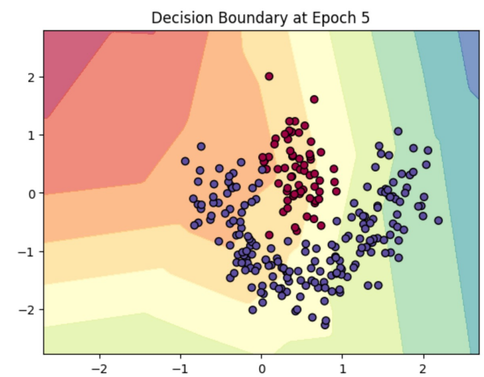


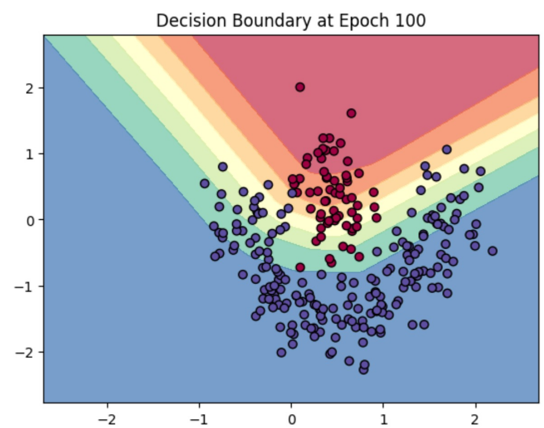
Nearest Neighbor Classifiers

```python
# Generate synthetic 2D classification dataset
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



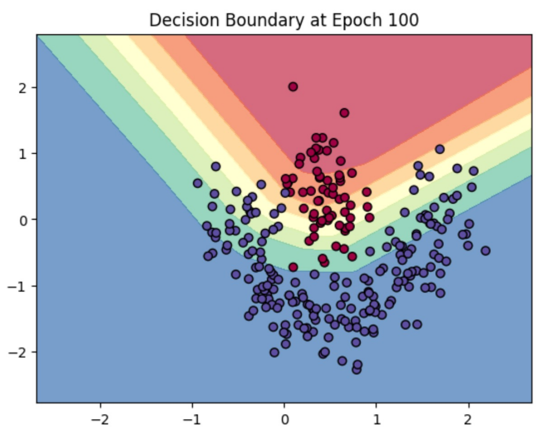


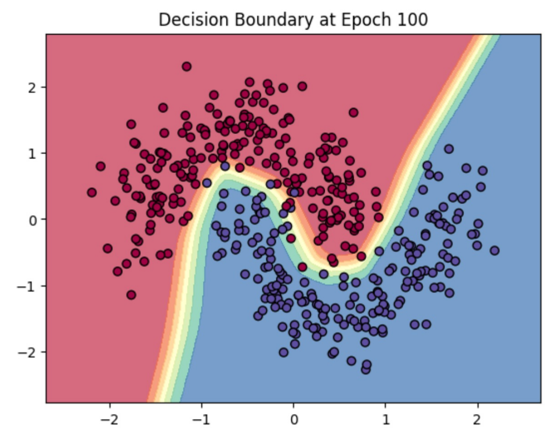




Nearest Neighbor Classifiers

```python
# Generate synthetic 2D classification dataset
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```






```python
# Define a simple MLP model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(2, 10)
        self.fc2 = nn.Linear(10, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x
```

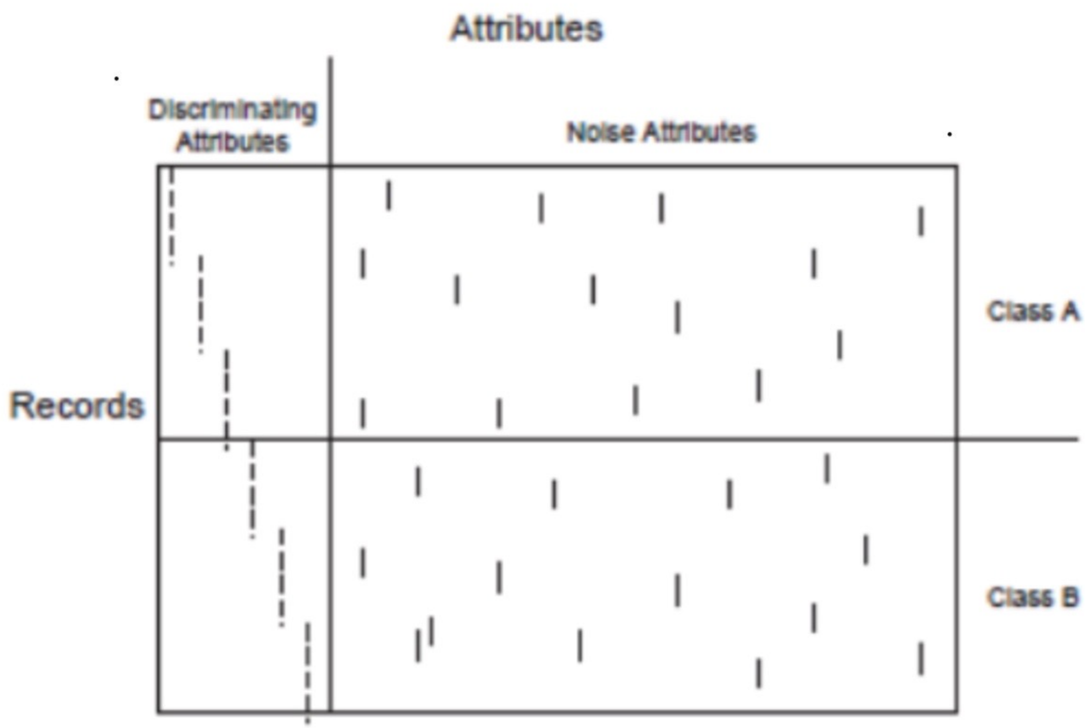
```python
# Define a simple MLP model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(2, 256)
        self.fc2 = nn.Linear(256, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x
```

Nearest Neighbor Classifiers

- **How to handle missing values in training and test sets?**
  - Proximity computations normally require the presence of all attributes
  - Some approaches use the subset of attributes present in two instances
    - This may not produce good results since it effectively uses different proximity measures for each pair of instances
    - Thus, proximities are not comparable

Nearest Neighbor Classifiers

– Irrelevant attributes add noise to the proximity measure
– Redundant attributes bias the proximity measure towards certain attributes



(a) Synthetic data set 1.

Improving KNN Efficiency

- Avoid having to compute distance to all objects in the training set
  - Multi-dimensional access methods (k-d trees)
  - Locality Sensitive Hashing (LSH)

Schedule from now on

Today – Finish KNN and K-means clustering

Wednesday – Naïve Bayesian

Presentation done, grade release for presentation

Next Monday – PCA/SVM + Review

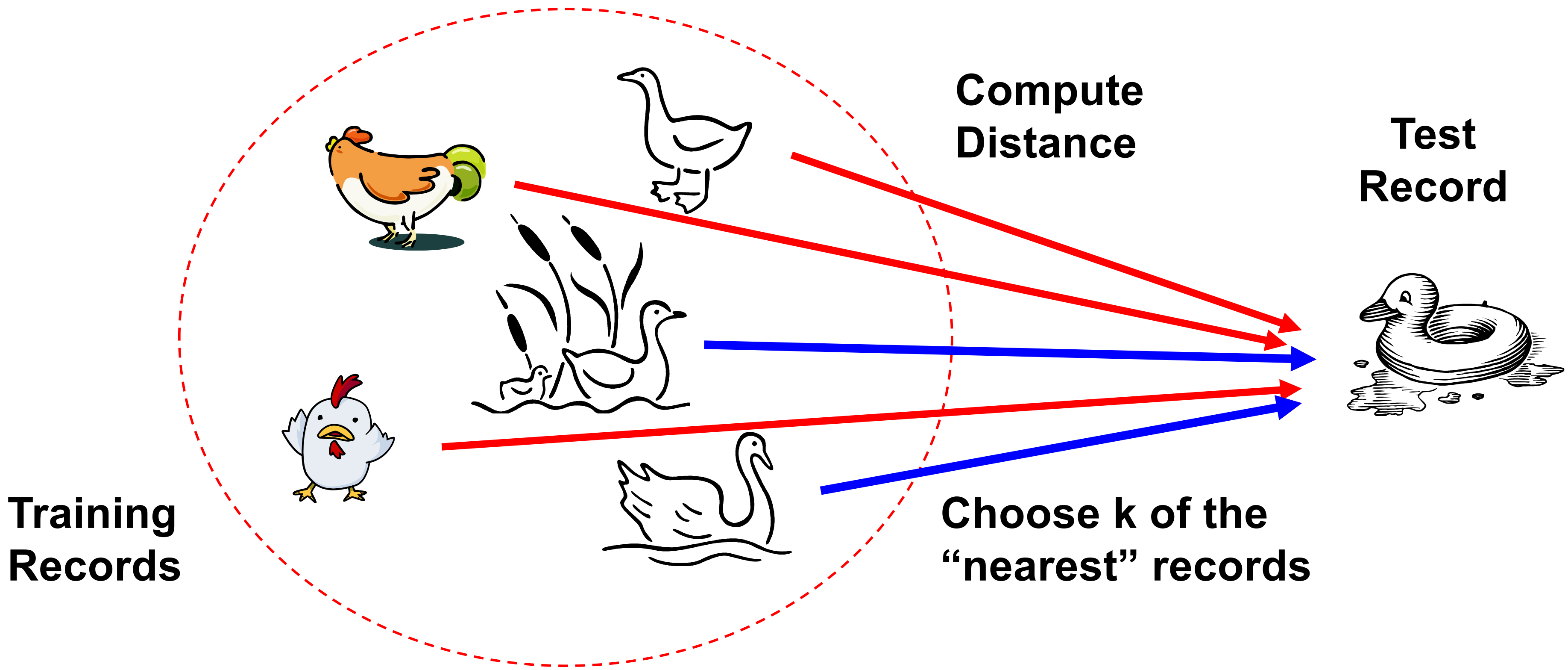
Next Wednesday – Quiz2 (No PCA/SVM)

3/10 Monday – PCA/SVM

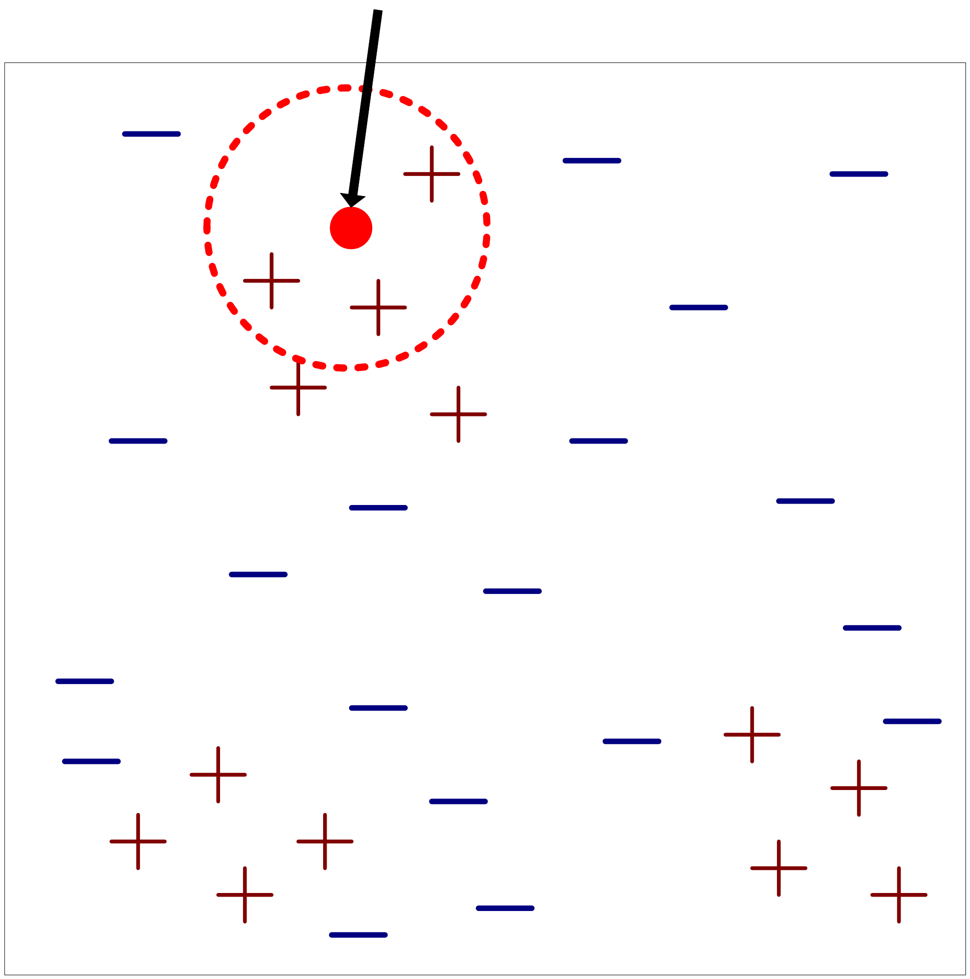
3/14 – Project Report

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck

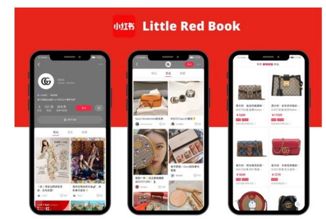


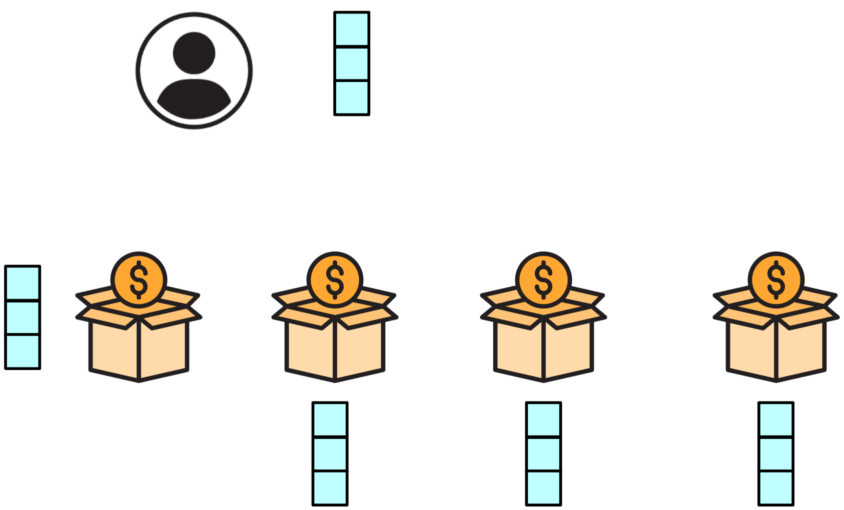
Nearest Neighbor Classifiers



- Requires the following:
  - A set of labeled records
  - Proximity metric to compute distance/similarity between a pair of records
    - e.g., Euclidean distance
  - The value of *k*, the number of nearest neighbors to retrieve
  - A method for using class labels of K nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Nearest Neighbor Classifiers



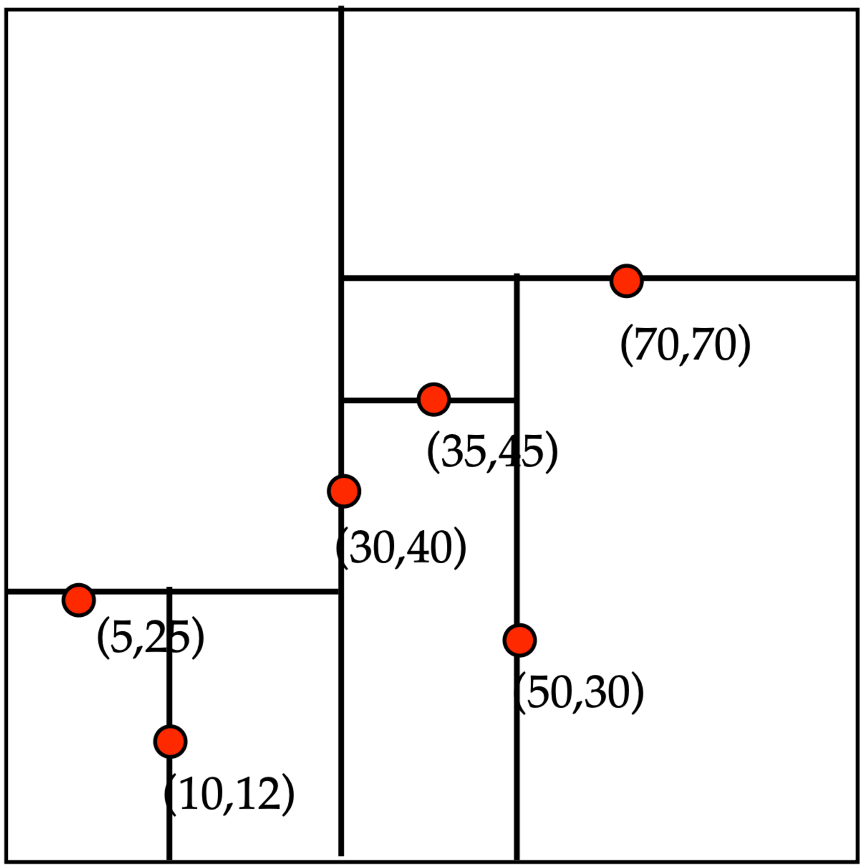


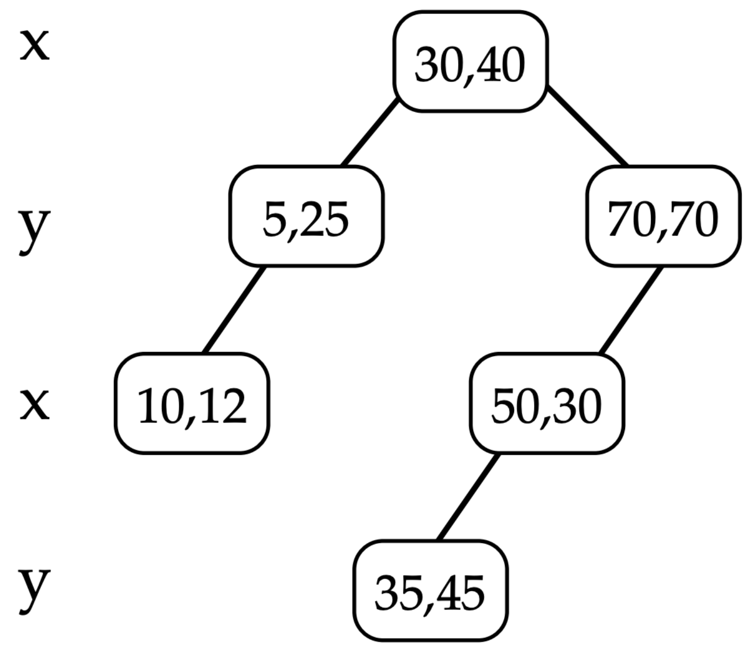
User as query to search nearest product

Nearest Neighbor Classifiers

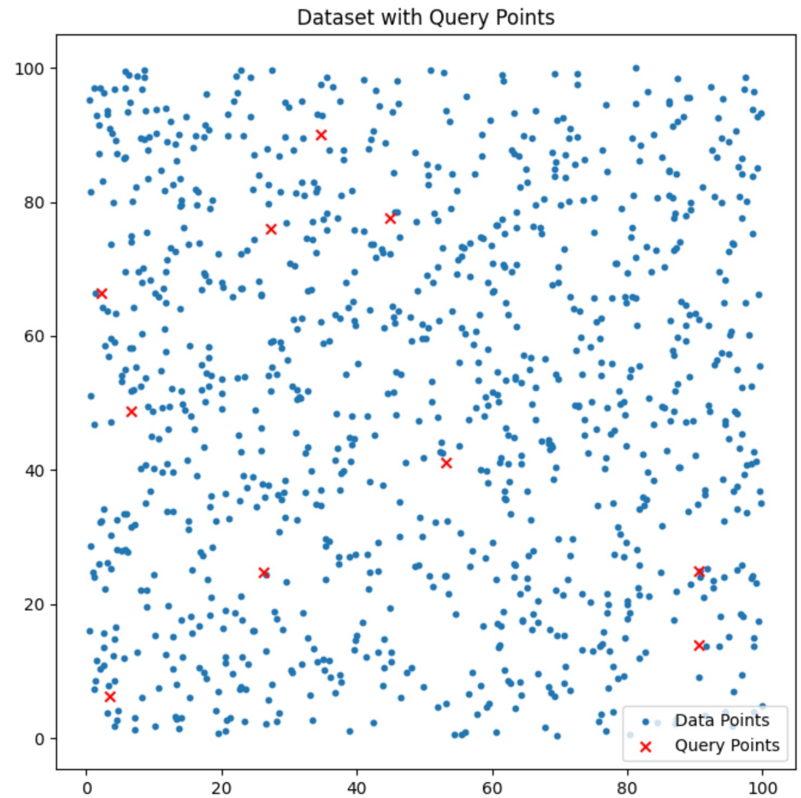
kd-tree example

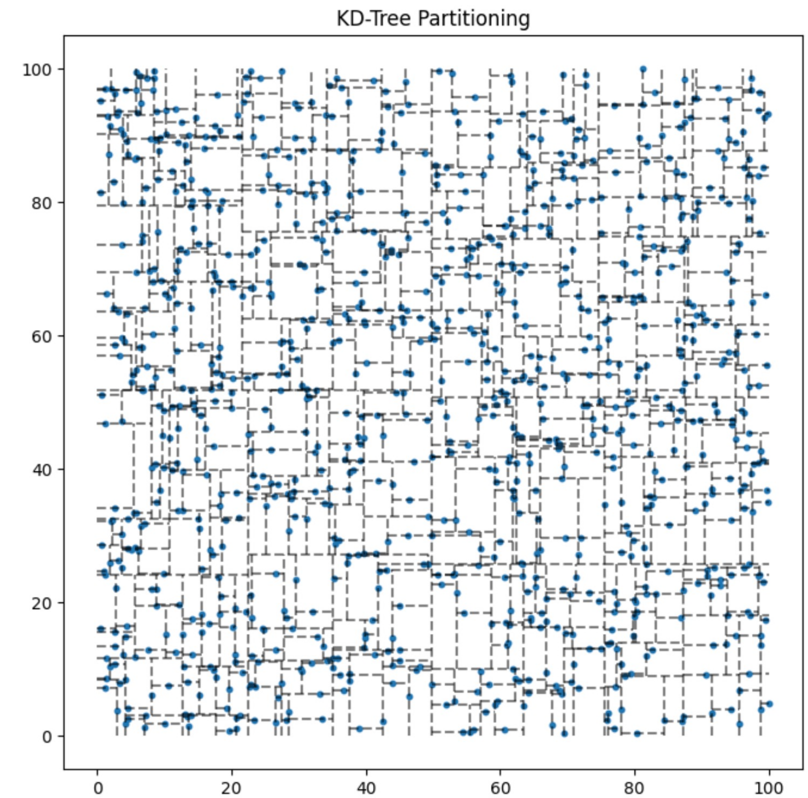
insert: (30,40), (5,25), (10,12), (70,70), (50,30), (35,45)





Nearest Neighbor Classifiers





Brute-force avg time: 0.131 ms
KD-Tree avg time: 0.059 ms

Nearest Neighbor Classifiers

Meta Our approach Research Product experiences Llama Blog

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.

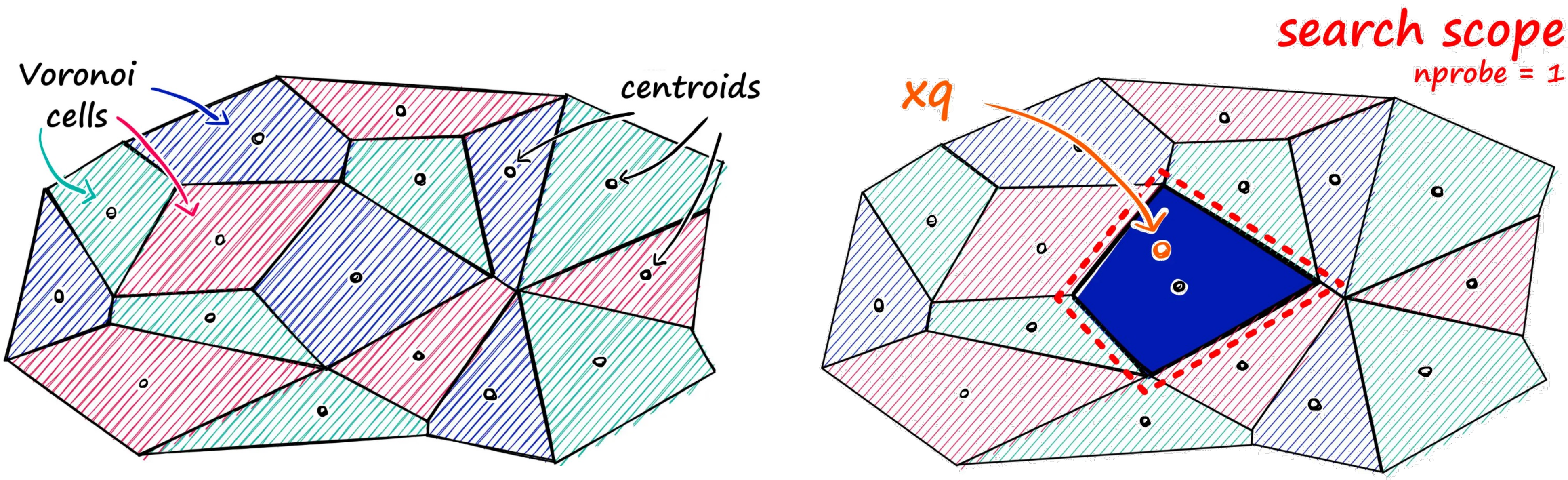
Faiss contains algorithms that search in sets of vectors of any size, and also contains supporting code for evaluation and parameter tuning. Some if its most useful algorithms are implemented on the GPU. Faiss is implemented in C++, with an optional Python interface and GPU support via CUDA.

FAISS

Scalable Search With Facebook AI

Nearest Neighbor Classifiers

K-means clustering



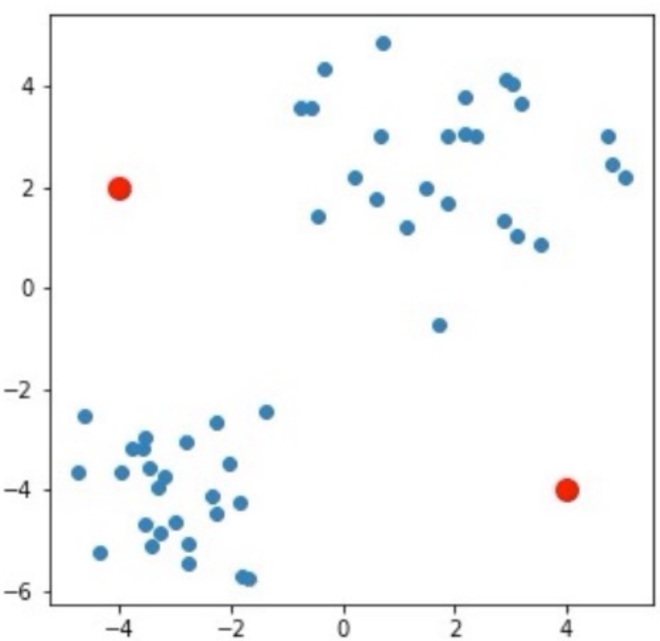
K-Means Clustering

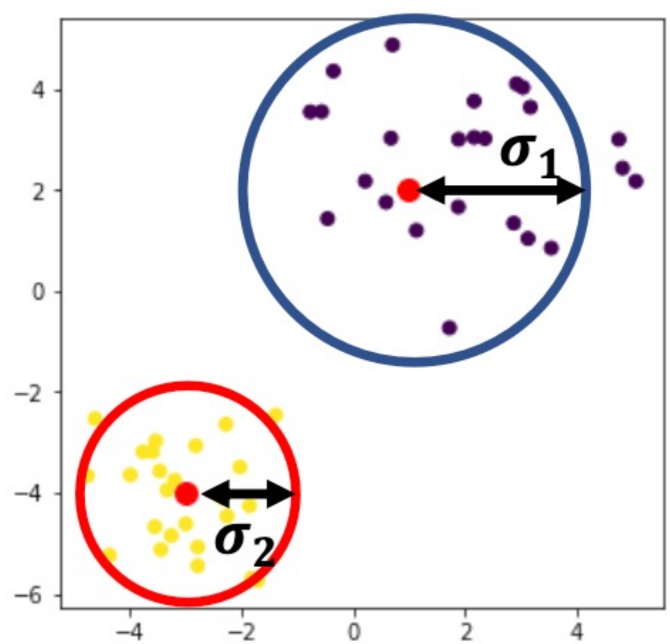
- Clustering:

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

Cluster

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





K-Means Clustering

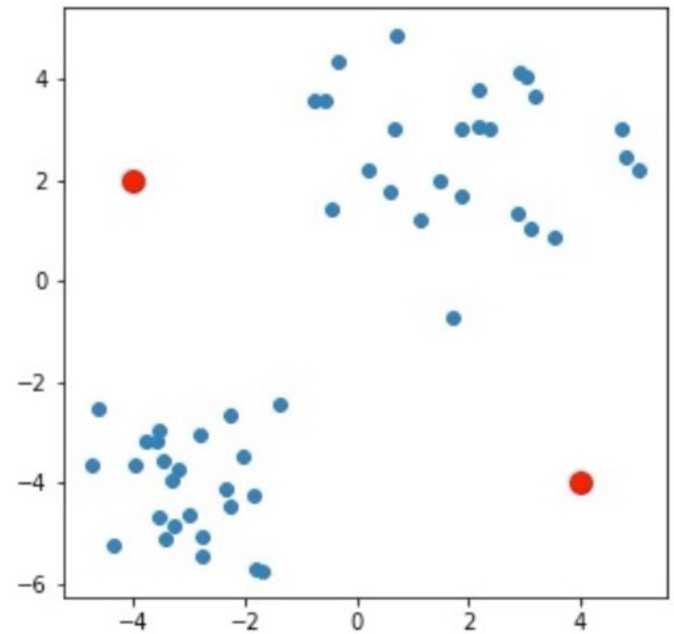
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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:
   - 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

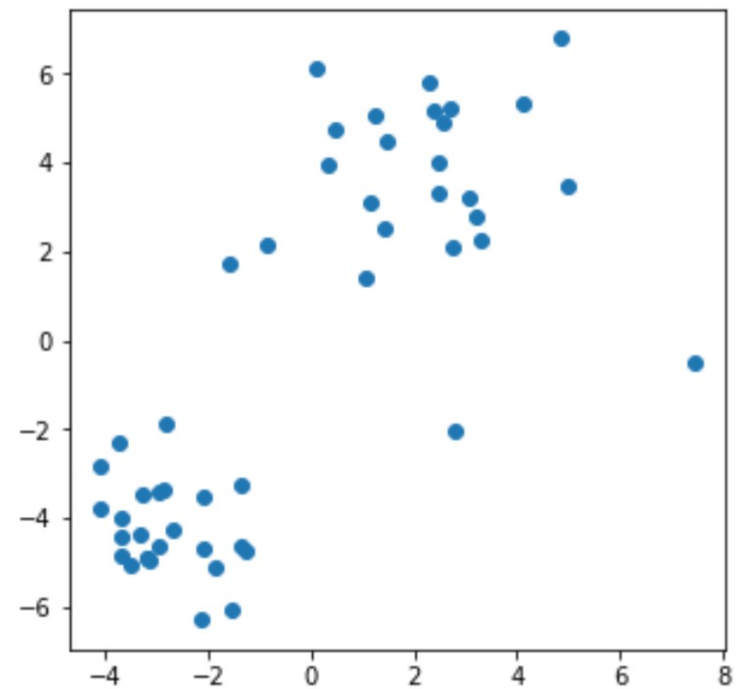
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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

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

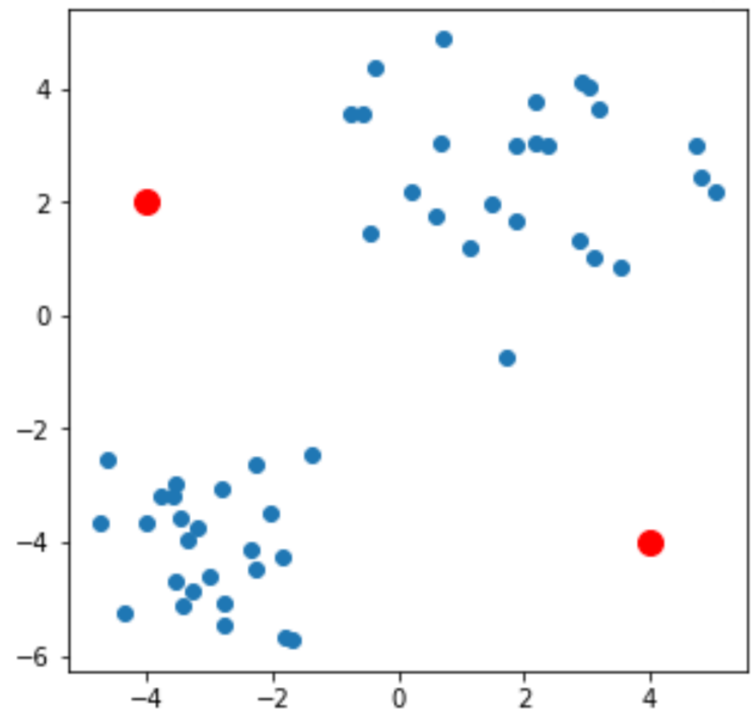
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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

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

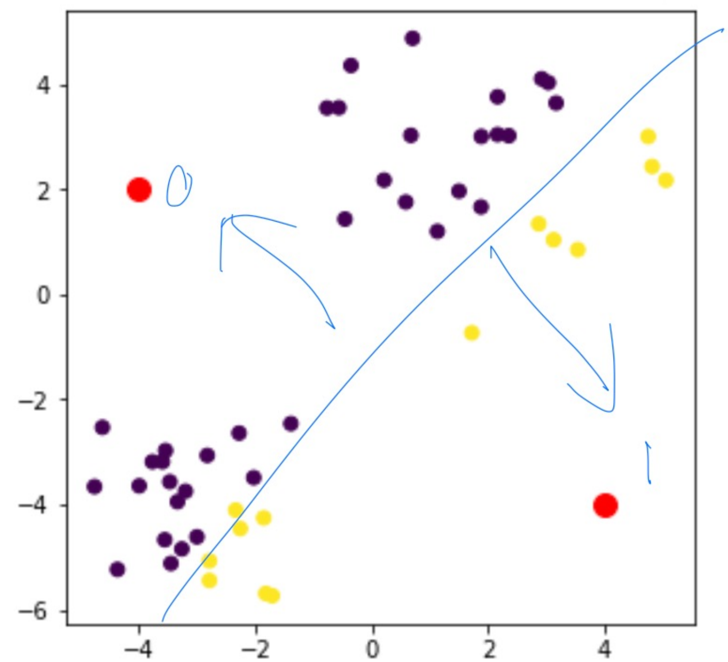
  Procedure:

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

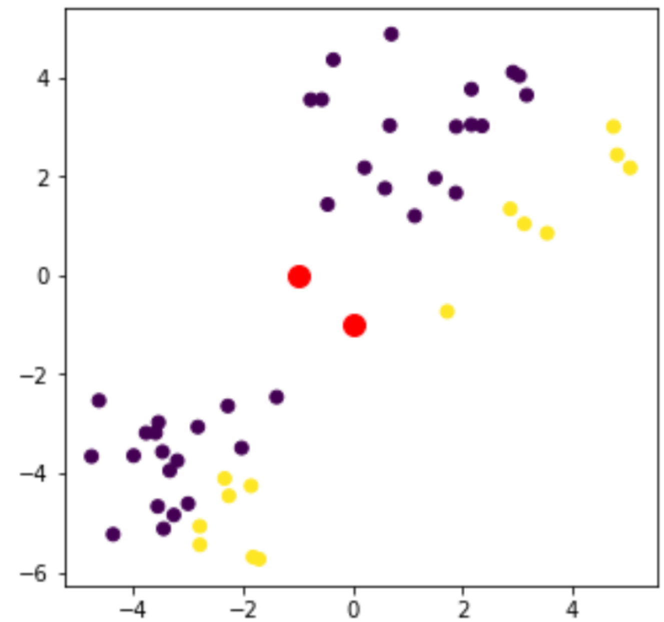
Procedure:

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

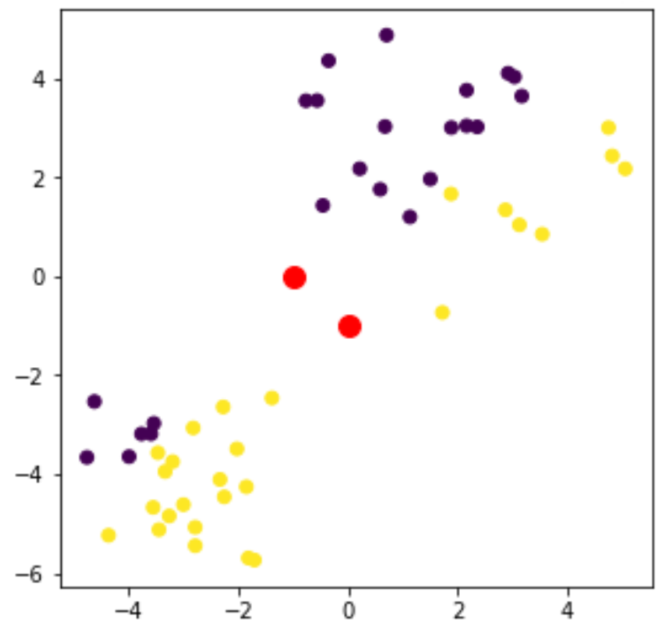
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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

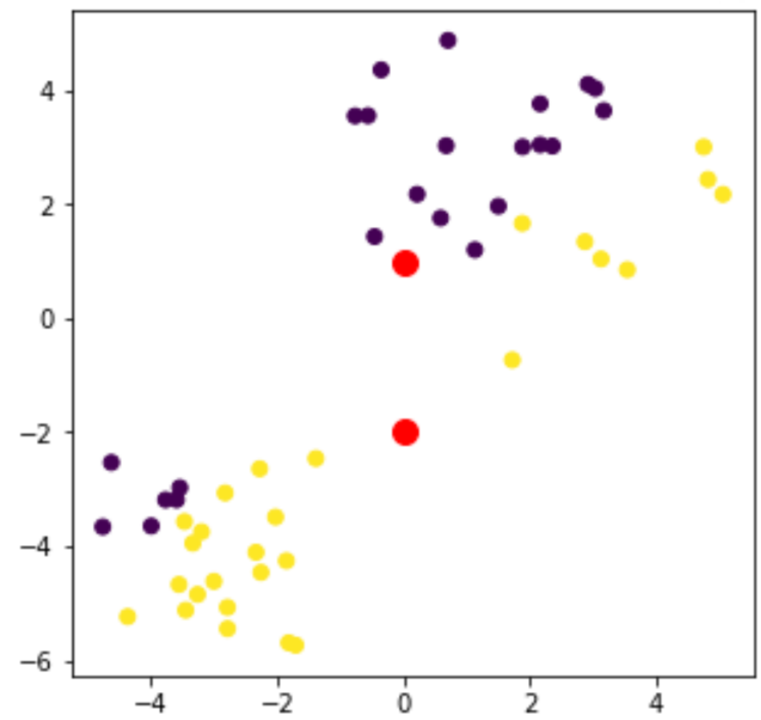
Procedure:

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

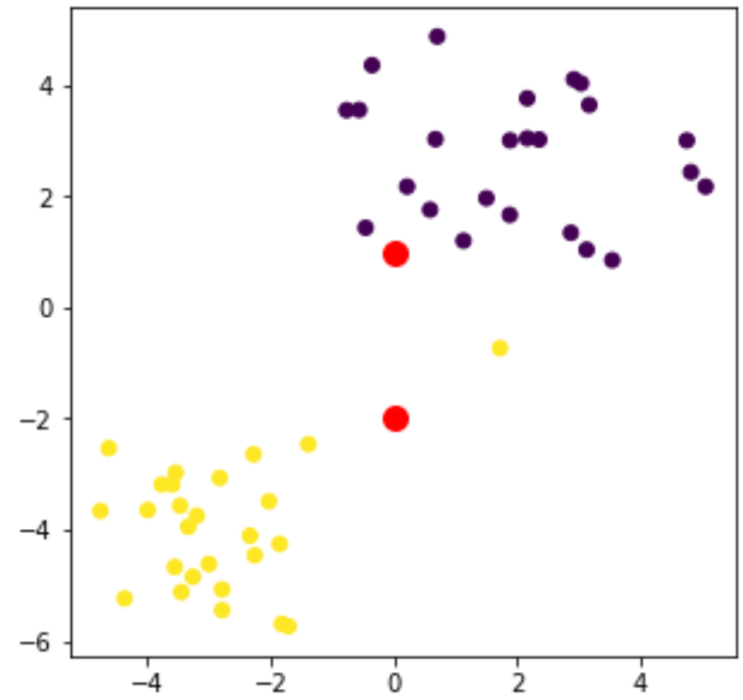
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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

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

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

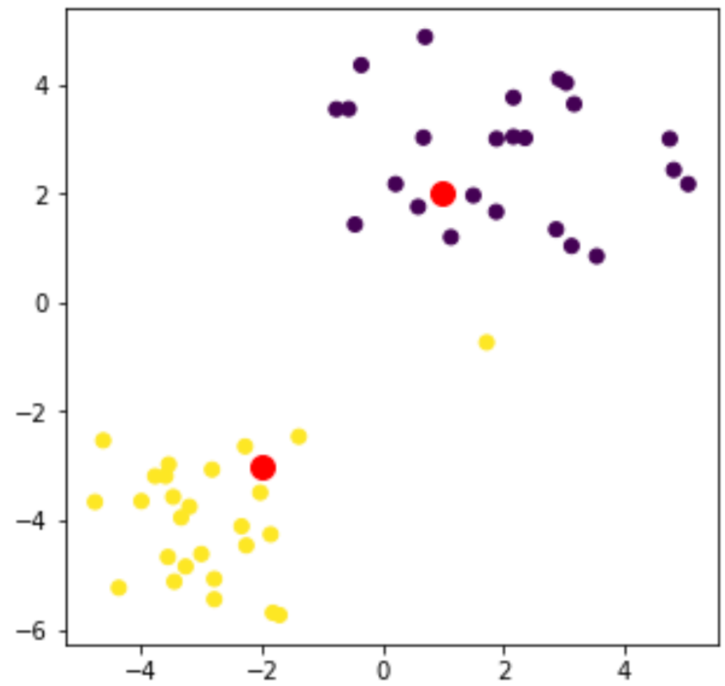
Procedure:

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

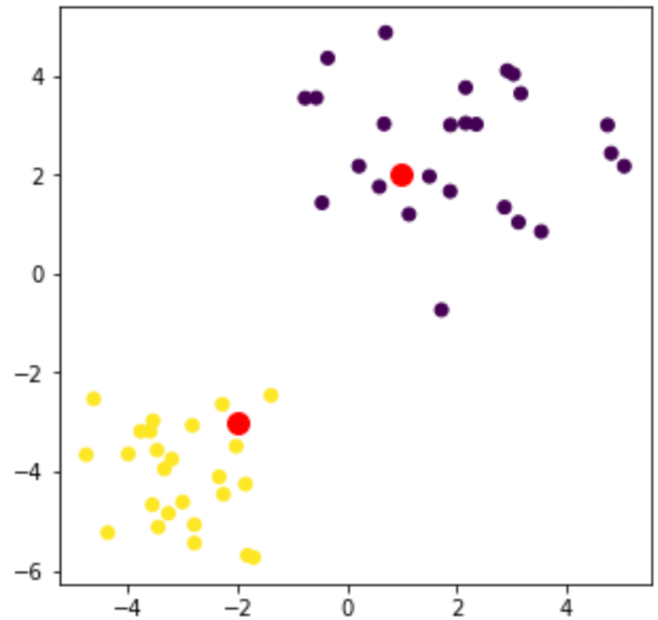
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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

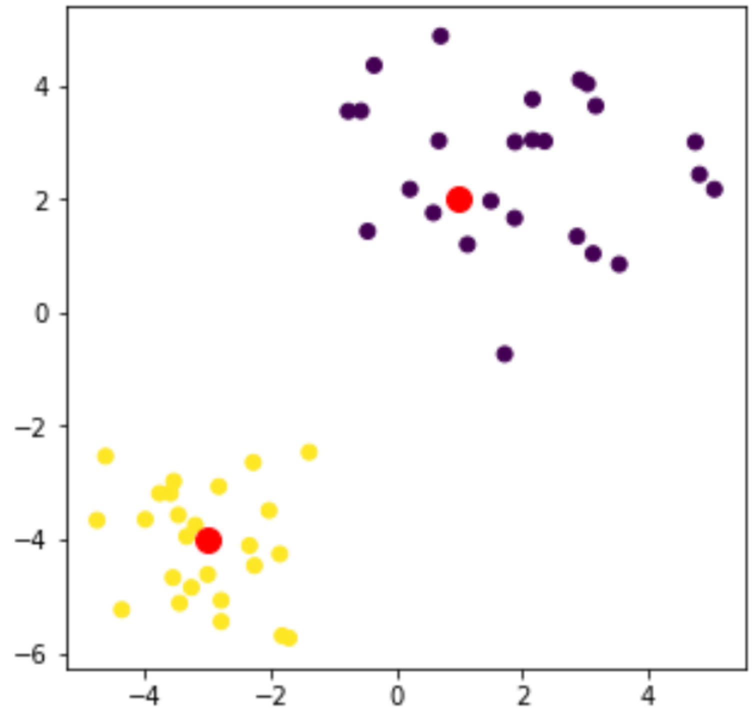
  Procedure:

  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

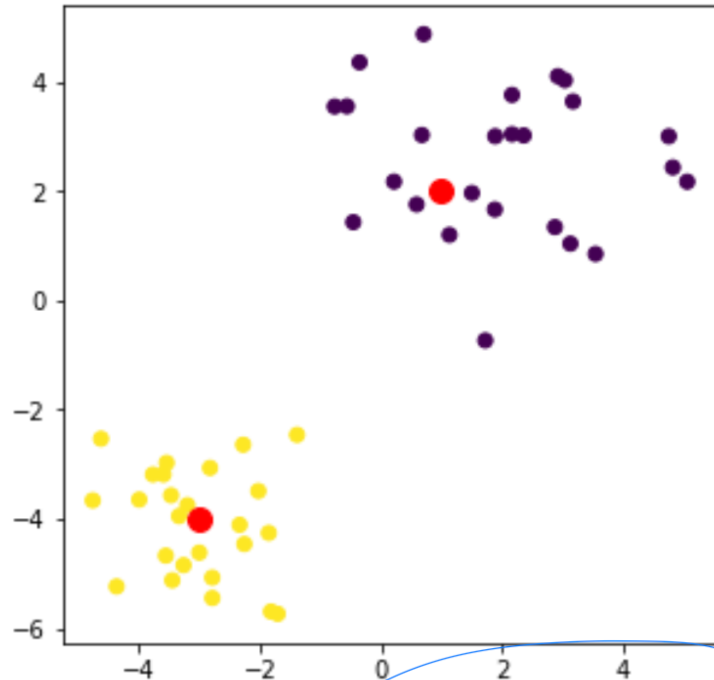
- K-Means: An iterative algorithm for clustering
- K-Means Algorithm:

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



Converged! Nothing will change.

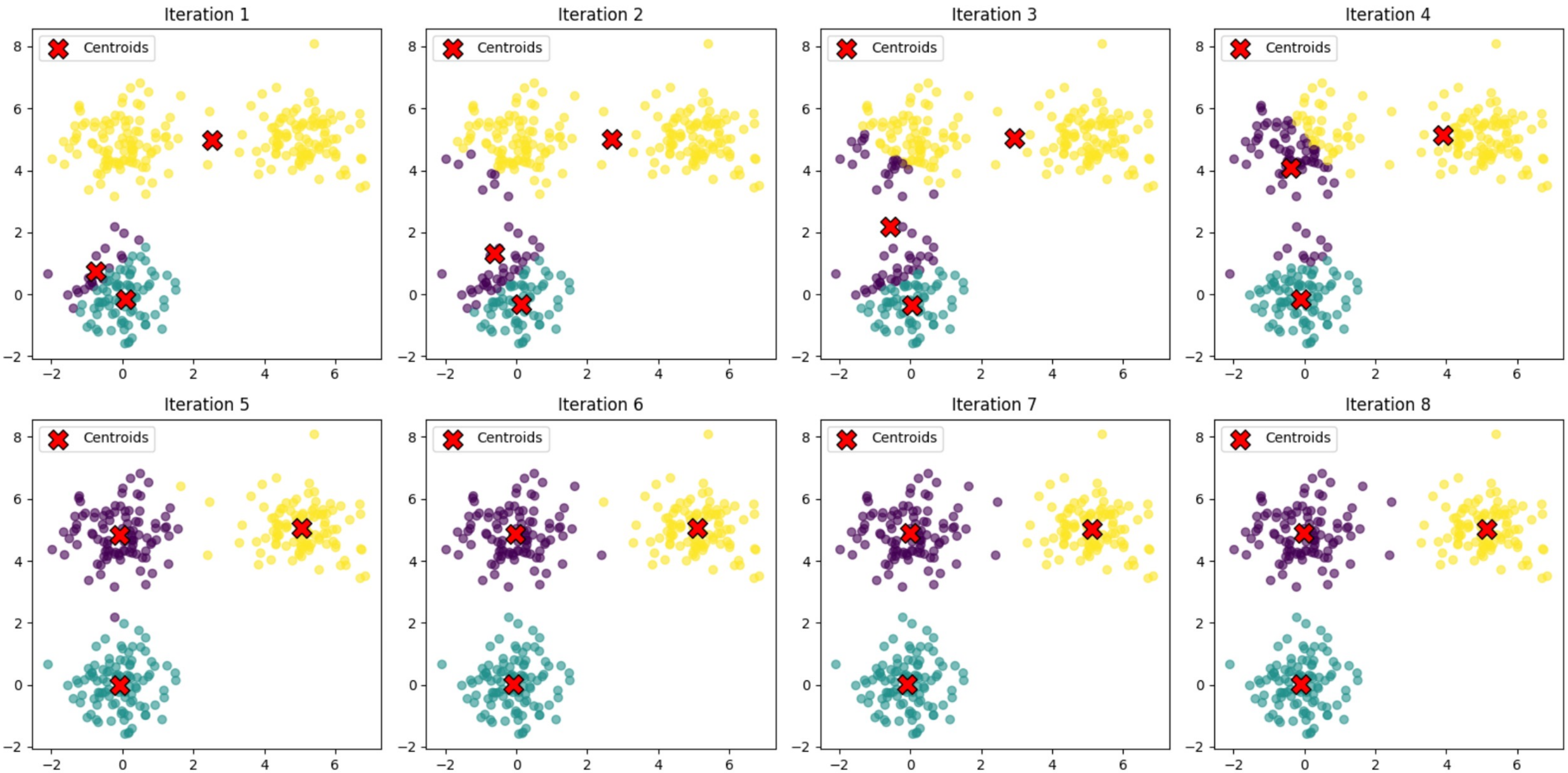
K-Means Clustering

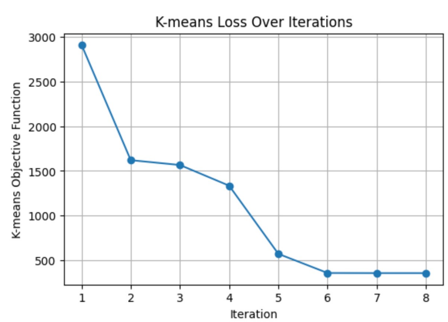
- A common objective function (used with Euclidean distance measure) is Sum of Squared Error (SSE)
  - For each point, the error is the distance to the nearest cluster center
  - To get SSE, we square these errors and sum them.

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

  - 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

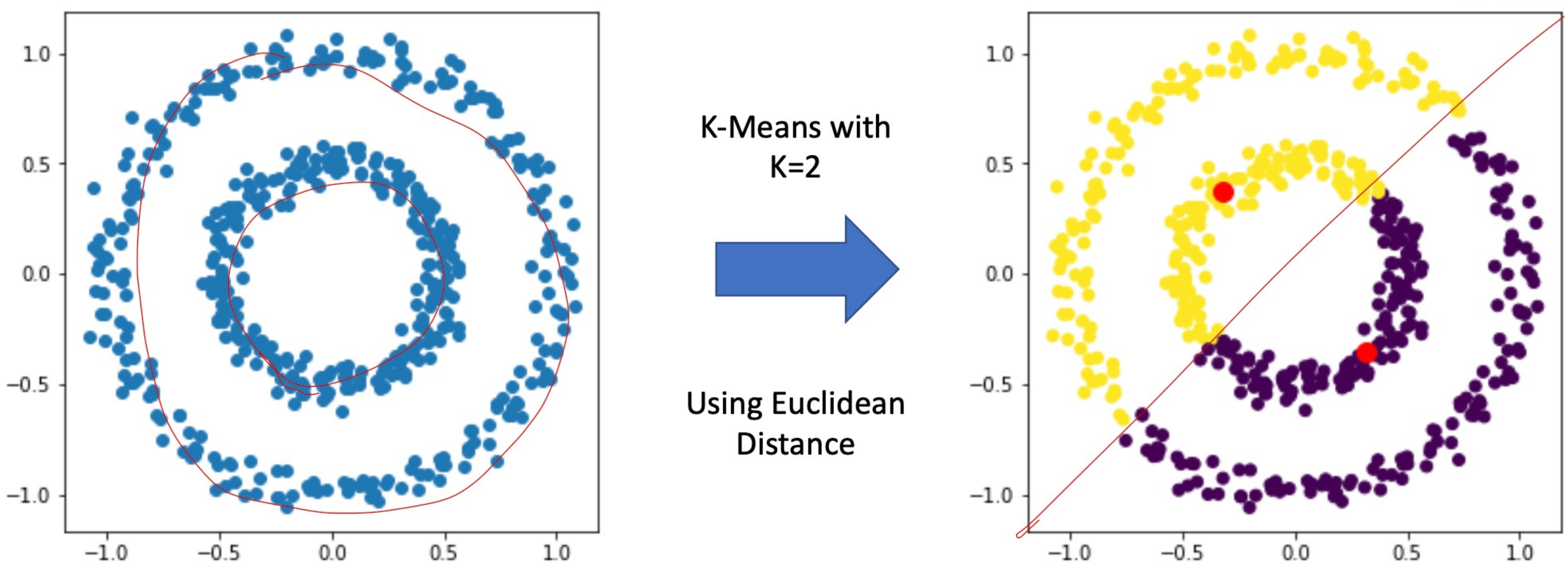
3. Running time

   (1) Assign data points to closest centroid: O(KND)

   (2) Change the cluster center: O(ND)

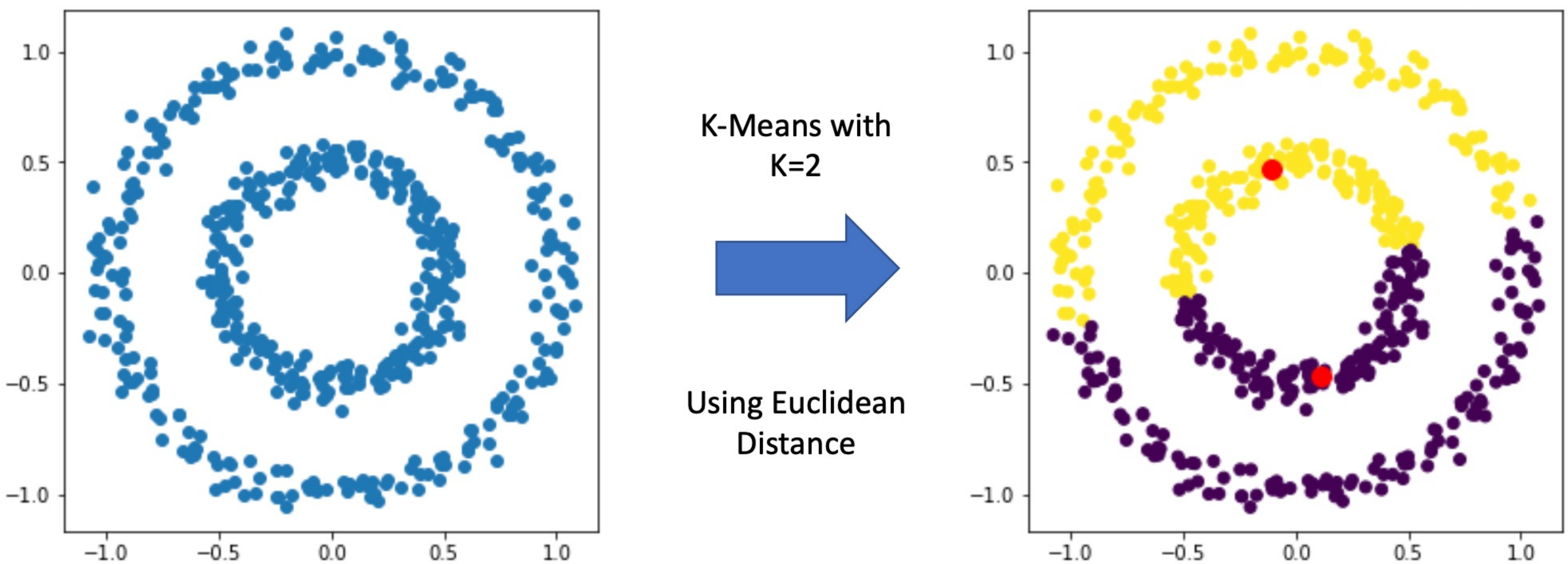
K-Means Clustering

Choice of Distance Matters



K-Means Clustering

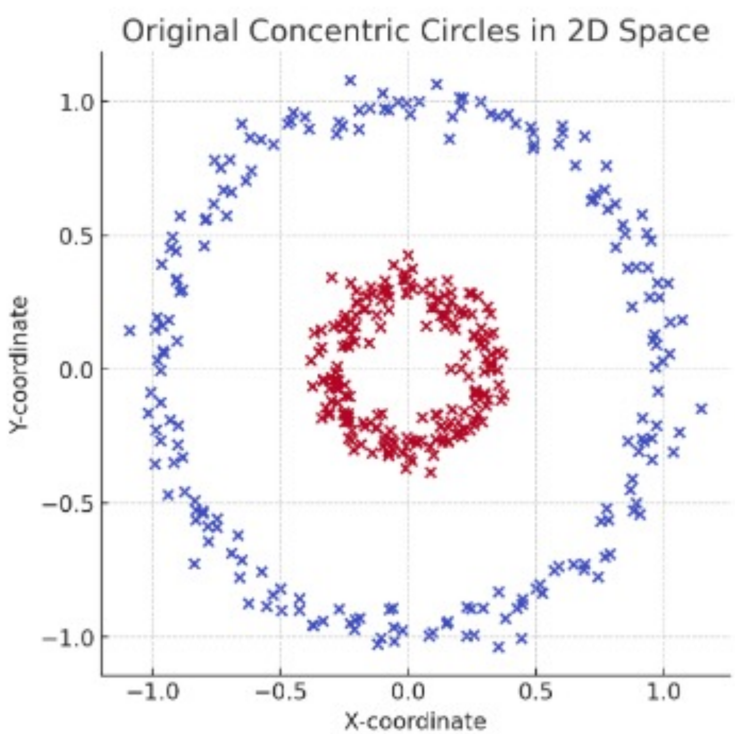
Choice of Distance Matters

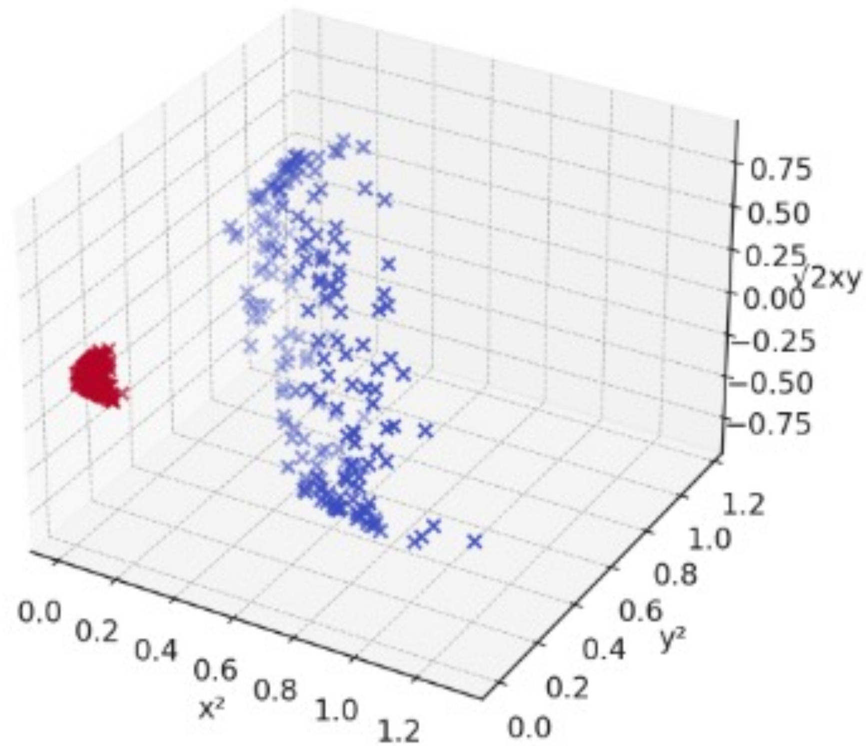


K-Means Clustering

Kernel + Clustering

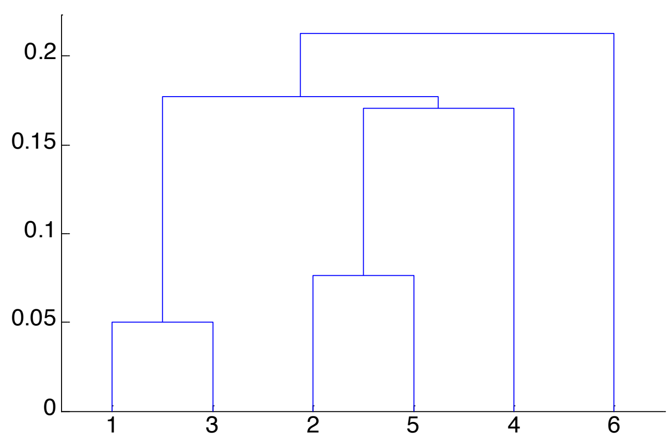
$$\Phi(x, y) = \begin{bmatrix} x^2 \\ y^2 \\ \sqrt{2}xy \end{bmatrix}$$

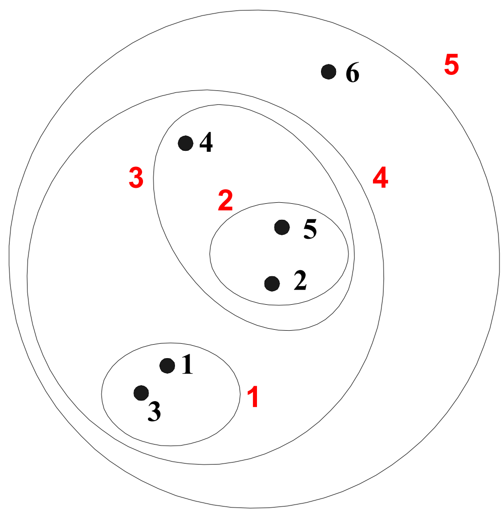




Hierarchical Clustering

- 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





Hierarchical 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, …)**

Hierarchical 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)**

Hierarchical Clustering

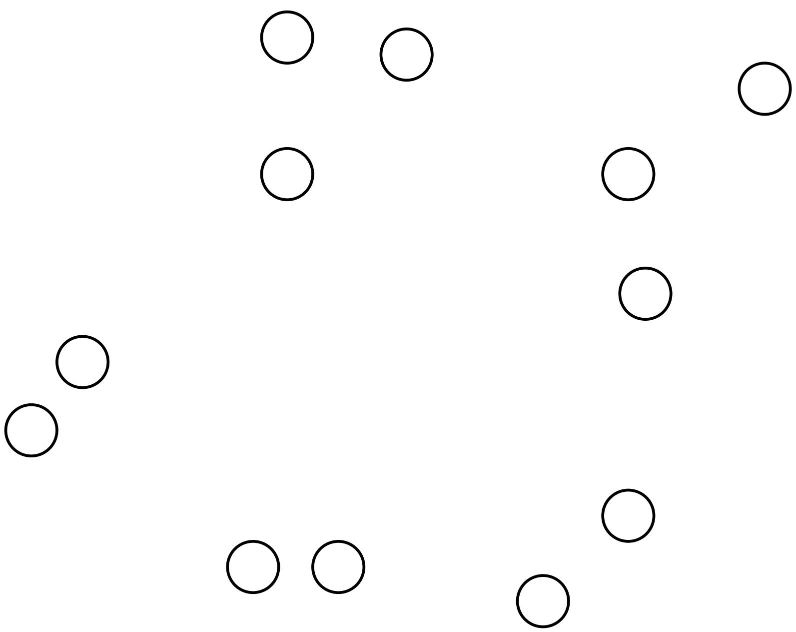
- **Key Idea: Successively merge closest clusters**

- **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

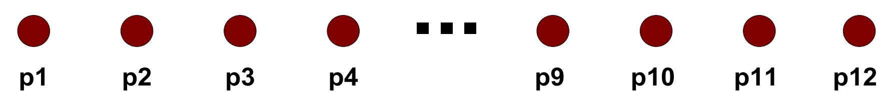
Hierarchical Clustering

- Start with clusters of individual points and a proximity matrix



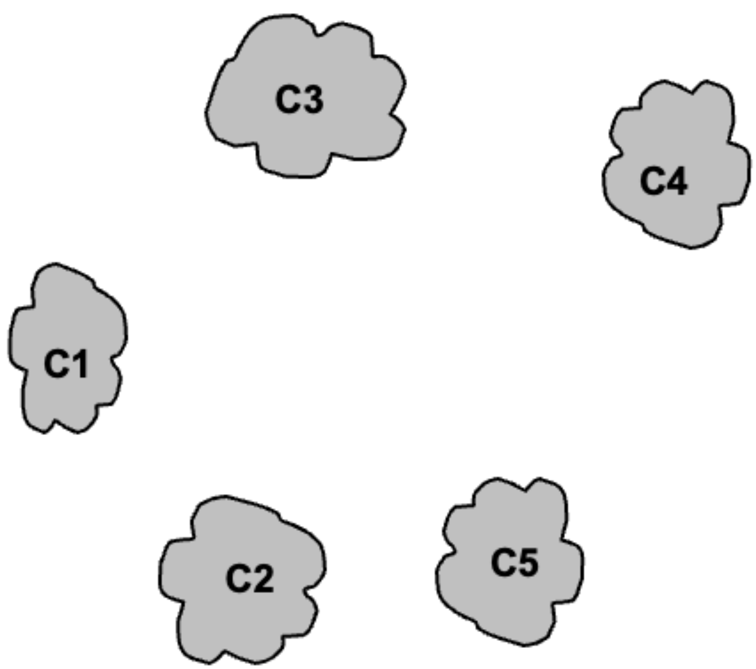
	p1	p2	p3	p4	p5	.
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix



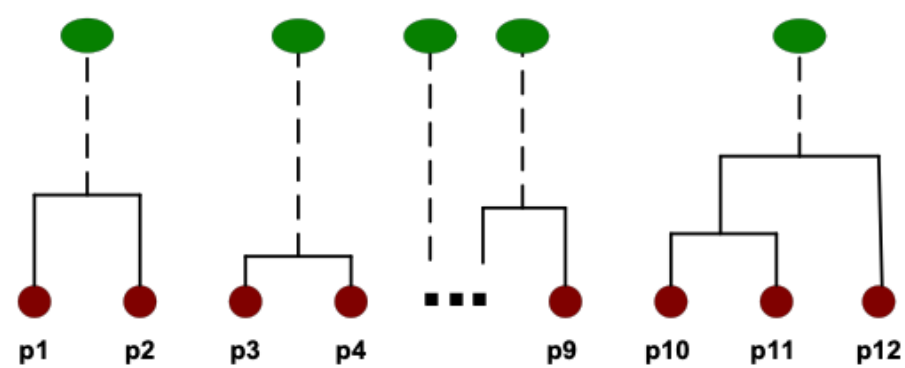
Hierarchical Clustering

- After some merging steps, we have some clusters



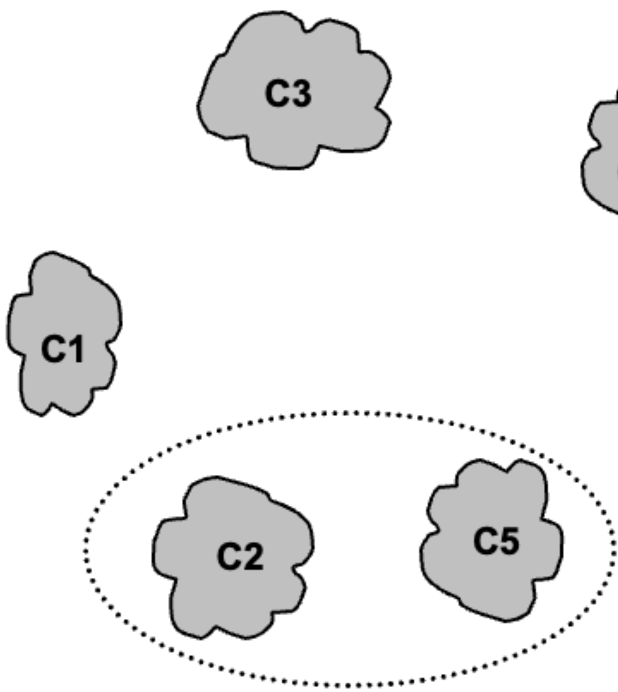
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

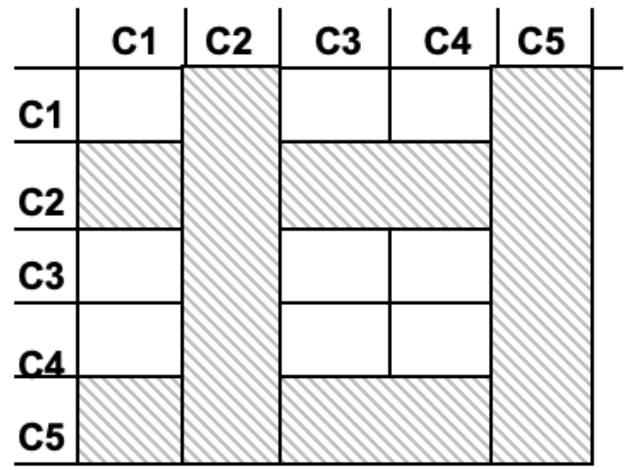
Proximity Matrix



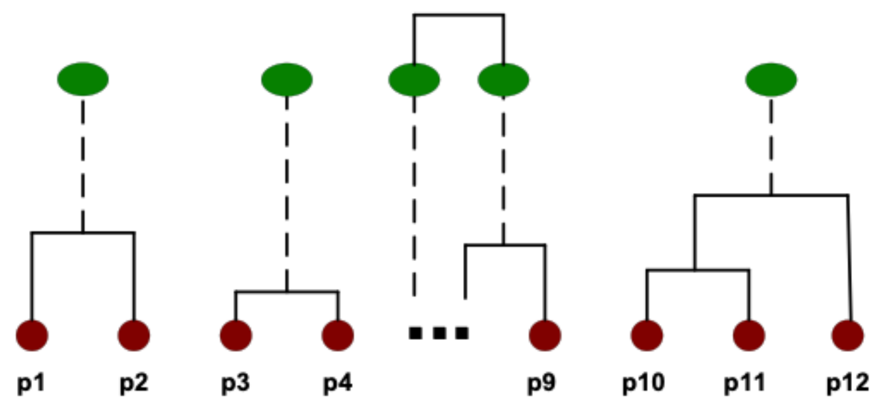
Hierarchical Clustering

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



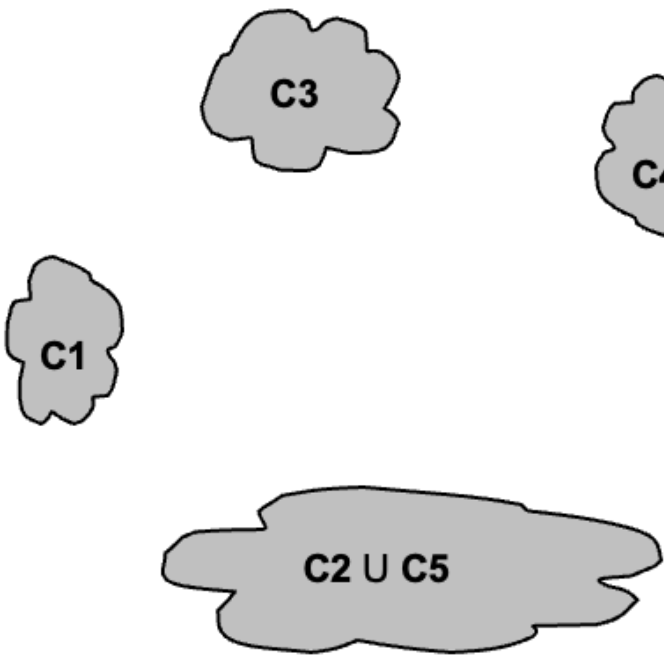


Proximity Matrix



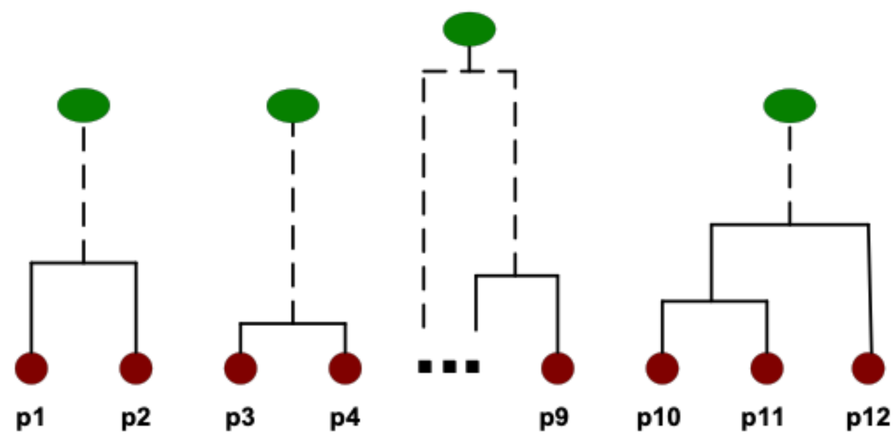
Hierarchical Clustering

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

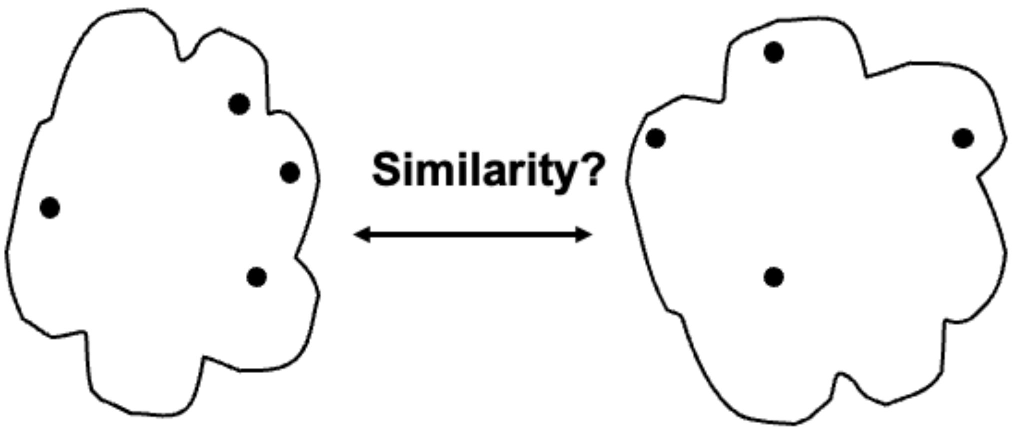


	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		

Proximity Matrix



Hierarchical Clustering

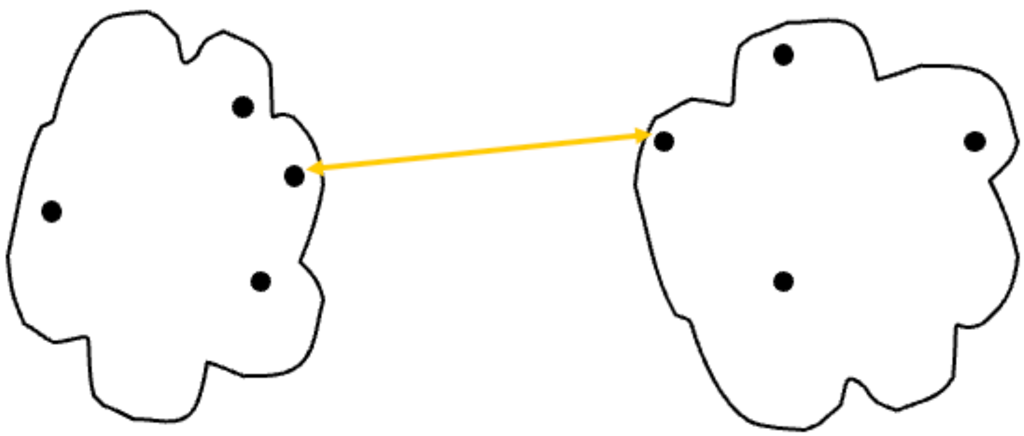


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids

Hierarchical Clustering

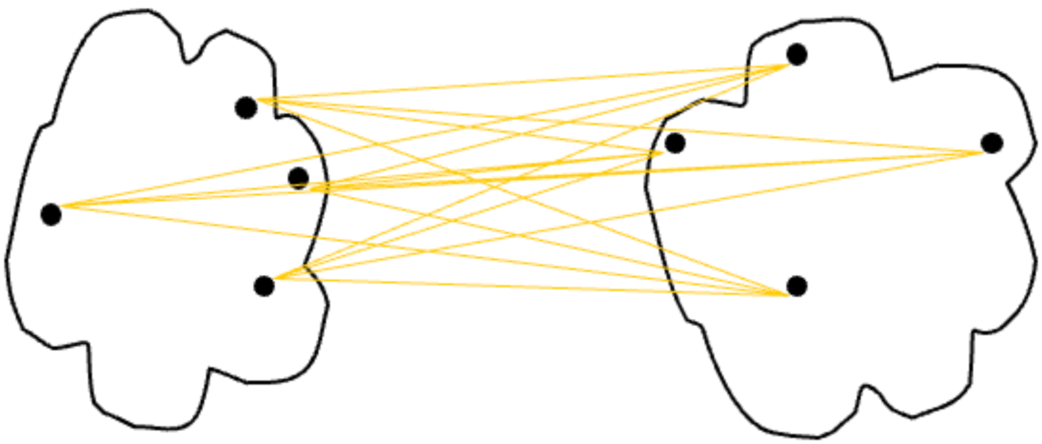


- MIN
- MAX
- Group Average
- Distance Between Centroids

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

Proximity Matrix

Hierarchical Clustering



- MIN
- MAX
- Group Average
- Distance Between Centroids

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

Proximity Matrix