

Lecture 17:

# **Clustering and Unsupervised Learning Fundamentals**

**Ho-min Park**

[homin.park@ghent.ac.kr](mailto:homin.park@ghent.ac.kr)

[powersimmani@gmail.com](mailto:powersimmani@gmail.com)

# Lecture Contents

**Part 1:** Unsupervised Learning Overview

**Part 2:** Clustering Algorithms

**Part 3:** Dimensionality Reduction

**Part 4:** Basic Anomaly Detection

**Part 1/4:**

# **Unsupervised Learning Overview**

- 1.** What is Unsupervised Learning?
- 2.** Supervised vs Unsupervised vs Semi-supervised
- 3.** Applications of Unsupervised Learning
- 4.** Key Challenges and Evaluation Methods
- 5.** Data Preprocessing and Scaling

## Supervised Learning

- Labeled training data
- Explicit target variables
- Learning from examples
- Ground truth labels

## Unsupervised Learning

- Unlabeled data only
- No target variables
- Discover hidden patterns
- Self-directed insights

## Three Main Categories of Unsupervised Learning



### Clustering

Group similar data points together



### Dimensionality Reduction

Reduce feature space complexity



### Anomaly Detection

Identify unusual patterns



## Supervised Learning

### DATA TYPE

Labeled data ( $X, y$ )

### GOAL

Predict  $y$  from  $X$

- ✓ Requires labeled data
- ✓ High accuracy
- ✓ Expensive labeling
- ✓ Domain expertise needed

Labeling Cost



## Unsupervised Learning

### DATA TYPE

Only  $X$  (unlabeled)

### GOAL

Find structure in  $X$

- ✓ No labels needed
- ✓ Discover patterns
- ✓ Highly scalable
- ✓ Cost-effective

Labeling Cost



## Semi-supervised Learning

### DATA TYPE

Small labeled + Large unlabeled

### GOAL

Leverage both data types

- ✓ Best of both worlds
- ✓ Balanced approach
- ✓ Practical solution
- ✓ Often best results

Labeling Cost



### Biological Analysis

Gene expression patterns



### Customer Segmentation

Group customers by behavior patterns



### Image Compression

Reduce dimensionality efficiently



### Document Clustering

Organize articles and papers



### Anomaly Detection

Fraud detection, intrusion detection



### Recommendation Systems

Find similar items without ratings



### Market Basket Analysis

Discover product associations



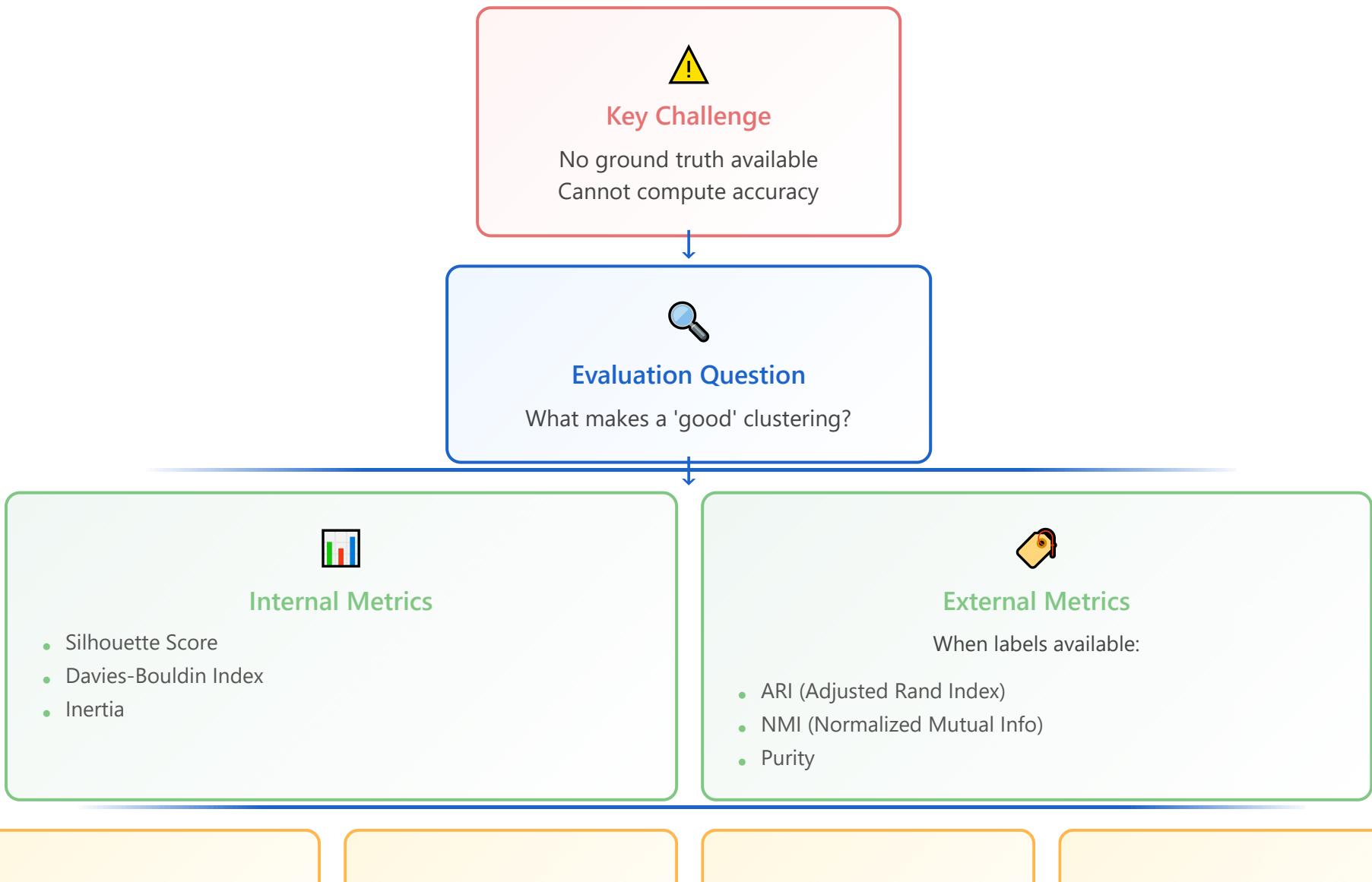
### Feature Engineering

Create better features for models



## Unsupervised Learning Applications

# Evaluation Framework for Unsupervised Learning





### Visual Inspection

Plot clusters in 2D/3D space



### Domain Validation

Do clusters make business sense?



### Stability Analysis

Consistent results across runs?



### Interpretability

Can we explain patterns?

# Data Preprocessing and Scaling

 Feature scaling is critical: Distance-based algorithms are sensitive to scale

## Scaling Methods



### StandardScaler

$$z = (x - \mu) / \sigma$$

- ✓ Zero mean
- ✓ Unit variance
- ✓ Assumes Gaussian



### MinMaxScaler

$$x' = (x - \min) / (\max - \min)$$

- ✓ Scale to [0, 1]
- ✓ Preserves zero
- ✓ Bounded range



### RobustScaler

$$x' = (x - \text{median}) / \text{IQR}$$

- ✓ Uses median
- ✓ Uses IQR
- ✓ Robust to outliers

## Preprocessing Pipeline

1



### Missing Values

Imputation before clustering

2



### Outlier Treatment

Remove or cap extremes

3



### Feature Selection

Remove irrelevant features

4



### Dimensionality

Handle curse of dimensions

## Calculation Examples

## StandardScaler Example

Data: [10, 20, 30, 40, 50]

$\mu = 30$   
 $\sigma = 14.14$   
For  $x = 10$ :  
 $z = (10 - 30) / 14.14$   
 $z = -1.41$

Result: [-1.41, -0.71, 0, 0.71, 1.41]

## MinMaxScaler Example

Data: [10, 20, 30, 40, 50]

min = 10, max = 50  
range = 50 - 10 = 40  
For  $x = 20$ :  
 $x' = (20 - 10) / 40$   
 $x' = 0.25$

Result: [0, 0.25, 0.5, 0.75, 1.0]

## RobustScaler Example

Data: [10, 20, 30, 40, 100]

median = 30  
Q1 = 20, Q3 = 40  
IQR = 40 - 20 = 20  
For  $x = 100$ :  
 $x' = (100 - 30) / 20 = 3.5$

Result: [-1.0, -0.5, 0, 0.5, 3.5]

**Part 2/4:**

# **Clustering Algorithms**

- 6.** Clustering Problem Definition
- 7.** K-Means Algorithm
- 8.** K-Means++ Initialization
- 9.** Hierarchical Clustering
- 10.** DBSCAN - Density-Based
- 11.** Mean Shift
- 12.** Gaussian Mixture Models
- 13.** Cluster Evaluation Metrics

## Clustering Problem Definition

### Core Objective



Partition n observations into k groups



Maximize intra-cluster similarity  
Minimize inter-cluster similarity

**Distance Metrics:** Euclidean • Manhattan • Cosine • Mahalanobis

### Types of Clustering Approaches



### Hard Clustering

Each point belongs to exactly one cluster

*Binary membership*



### Soft Clustering

Points have probability distribution over clusters

*Fuzzy membership*



### Hierarchical

Nested clusters forming tree structure

*Dendrogram output*



### Density-Based

Clusters as high-density regions

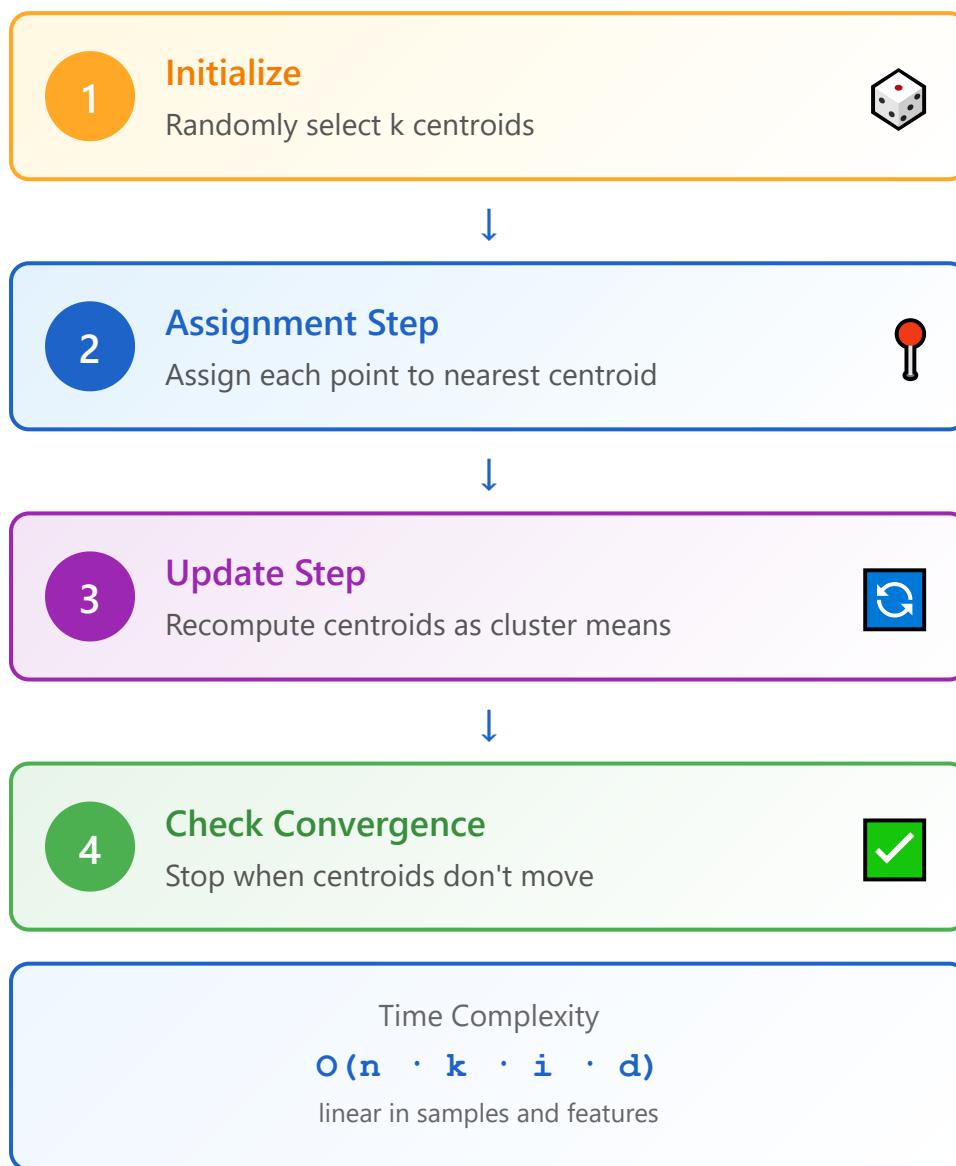
*Arbitrary shapes*

## Model-Based Approach



Data generated from mixture of probability distributions

# K-Means Algorithm



## Strengths

- ✓ Simple to understand
- ✓ Fast computation
- ✓ Scalable to large datasets

## Weaknesses

- ✗ Assumes spherical clusters
- ✗ Sensitive to initialization
- ✗ Requires pre-specified k

**Finding optimal k:** Use elbow method or silhouette analysis



## Detailed Computational Example (k=3)

0

### Initial Dataset

12  
34 Given 6 points in 2D space:

$$P_1 = (2, 3)$$

$$P_2 = (3, 3)$$

$$P_3 = (8, 7)$$

$$P_4 = (9, 6)$$

$$P_5 = (5, 2)$$

$$P_6 = (6, 3)$$



Goal: Partition these 6 points into k=3 clusters

1

### Initialize Centroids

🎲 Randomly selected initial centroids:

$$C_1 = (2, 3)$$

$$C_2 = (8, 7)$$

$$C_3 = (5, 2)$$



In practice, we randomly select k points from the dataset as initial centroids

2

## Iteration 1 - Assignment Step



Calculate Euclidean distances from each point to all centroids:

For  $P_1 = (2, 3)$ :

$$d(P_1, C_1) = \sqrt[(2-2)^2 + (3-3)^2] = \sqrt[0 + 0] = 0.00$$

$$d(P_1, C_2) = \sqrt[(2-8)^2 + (3-7)^2] = \sqrt[36 + 16] = 7.21$$

$$d(P_1, C_3) = \sqrt[(2-5)^2 + (3-2)^2] = \sqrt[9 + 1] = 3.16$$

→ Assign  $P_1$  to Cluster 1 (minimum distance: 0.00)

For  $P_2 = (3, 3)$ :

$$d(P_2, C_1) = \sqrt[(3-2)^2 + (3-3)^2] = \sqrt[1 + 0] = 1.00$$

$$d(P_2, C_2) = \sqrt[(3-8)^2 + (3-7)^2] = \sqrt[25 + 16] = 6.40$$

$$d(P_2, C_3) = \sqrt[(3-5)^2 + (3-2)^2] = \sqrt[4 + 1] = 2.24$$

→ Assign  $P_2$  to Cluster 1 (minimum distance: 1.00)

For  $P_3 = (8, 7)$ :

$$d(P_3, C_1) = \sqrt[(8-2)^2 + (7-3)^2] = \sqrt[36 + 16] = 7.21$$

$$d(P_3, C_2) = \sqrt[(8-8)^2 + (7-7)^2] = \sqrt[0 + 0] = 0.00$$

$$d(P_3, C_3) = \sqrt{(8-5)^2 + (7-2)^2} = \sqrt{9 + 25} = 5.83$$

→ Assign  $P_3$  to Cluster 2 (minimum distance: 0.00)

For  $P_4 = (9, 6)$ :

$$d(P_4, C_1) = \sqrt{(9-2)^2 + (6-3)^2} = \sqrt{49 + 9} = 7.62$$

$$d(P_4, C_2) = \sqrt{(9-8)^2 + (6-7)^2} = \sqrt{1 + 1} = 1.41$$

$$d(P_4, C_3) = \sqrt{(9-5)^2 + (6-2)^2} = \sqrt{16 + 16} = 5.66$$

→ Assign  $P_4$  to Cluster 2 (minimum distance: 1.41)

For  $P_5 = (5, 2)$ :

$$d(P_5, C_1) = \sqrt{(5-2)^2 + (2-3)^2} = \sqrt{9 + 1} = 3.16$$

$$d(P_5, C_2) = \sqrt{(5-8)^2 + (2-7)^2} = \sqrt{9 + 25} = 5.83$$

$$d(P_5, C_3) = \sqrt{(5-5)^2 + (2-2)^2} = \sqrt{0 + 0} = 0.00$$

→ Assign  $P_5$  to Cluster 3 (minimum distance: 0.00)

For  $P_6 = (6, 3)$ :

$$d(P_6, C_1) = \sqrt{(6-2)^2 + (3-3)^2} = \sqrt{16 + 0} = 4.00$$

$$d(P_6, C_2) = \sqrt{(6-8)^2 + (3-7)^2} = \sqrt{4 + 16} = 4.47$$

$$d(P_6, C_3) = \sqrt{(6-5)^2 + (3-2)^2} = \sqrt{1 + 1} = 1.41$$

→ Assign  $P_6$  to Cluster 3 (minimum distance: 1.41)

● Cluster 1

$$P_1 = (2, 3)$$

$$P_2 = (3, 3)$$

● Cluster 2

$$P_3 = (8, 7)$$

$$P_4 = (9, 6)$$

● Cluster 3

$$P_5 = (5, 2)$$

$$P_6 = (6, 3)$$

### 3 Iteration 1 - Update Centroids

↻ Calculate new centroids as the mean of assigned points:

New  $C_1$  :

Points in Cluster 1:  $P_1 (2, 3), P_2 (3, 3)$

$$C_1 \text{ new} = ((2+3)/2, (3+3)/2) = (2.5, 3.0)$$

$$C_1 : (2, 3) \rightarrow (2.5, 3.0) \checkmark \text{ Changed}$$

New  $C_2$  :

Points in Cluster 2:  $P_3 (8, 7), P_4 (9, 6)$

$$C_2 \text{ new} = ((8+9)/2, (7+6)/2) = (8.5, 6.5)$$

$$C_2 : (8, 7) \rightarrow (8.5, 6.5) \checkmark \text{ Changed}$$

New  $C_3$  :

Points in Cluster 3:  $P_5 (5, 2), P_6 (6, 3)$

$C_3 \text{ } \underline{\text{new}} = ((5+6)/2, (2+3)/2) = (5.5, 2.5)$

$C_3 : (5, 2) \rightarrow (5.5, 2.5) \checkmark \text{ Changed}$

💡 Updated centroids for Iteration 2:

$C_1 = (2.5, 3.0)$

$C_2 = (8.5, 6.5)$

$C_3 = (5.5, 2.5)$



Centroids have moved! Continue to Iteration 2...

2

## Iteration 2 - Assignment Step



Recalculate distances with new centroids:

For  $P_1 = (2, 3)$ :

$$d(P_1, C_1) = \sqrt[(2-2.5)^2 + (3-3)^2] = 0.50 \checkmark \text{ min}$$

$$d(P_1, C_2) = \sqrt[(2-8.5)^2 + (3-6.5)^2] = 7.27$$

$$d(P_1, C_3) = \sqrt[(2-5.5)^2 + (3-2.5)^2] = 3.54$$

→ Remains in Cluster 1

... (similar calculations for  $P_2, P_3, P_4, P_5, P_6$ ) ...

### Cluster 1

$$P_1 = (2, 3)$$

$$P_2 = (3, 3)$$

### Cluster 2

$$P_3 = (8, 7)$$

$$P_4 = (9, 6)$$

### Cluster 3

$$P_5 = (5, 2)$$

$$P_6 = (6, 3)$$



**Result:** No points changed clusters! (same as Iteration 1)

3

## Iteration 2 - Update Centroids

### Recalculate centroids:

$$C_1 \text{ new} = ((2+3)/2, (3+3)/2) = (2.5, 3.0)$$

$$C_2 \text{ new} = ((8+9)/2, (7+6)/2) = (8.5, 6.5)$$

$$C_3 \text{ new} = ((5+6)/2, (2+3)/2) = (5.5, 2.5)$$

### Centroids after Iteration 2:

$$C_1 = (2.5, 3.0) - \text{No change}$$

$$C_2 = (8.5, 6.5) - \text{No change}$$

$$C_3 = (5.5, 2.5) - \text{No change}$$

## 4

## Check Convergence

 Algorithm Converged!

**Condition met:** Centroids did not move between iterations

- $C_1: (2.5, 3.0) \rightarrow (2.5, 3.0)$  - Movement: 0.00
- $C_2: (8.5, 6.5) \rightarrow (8.5, 6.5)$  - Movement: 0.00
- $C_3: (5.5, 2.5) \rightarrow (5.5, 2.5)$  - Movement: 0.00

 Final Clustering Result: Cluster 1

$$P_1 = (2, 3)$$

$$P_2 = (3, 3)$$

$$\text{Centroid: } (2.5, 3.0)$$

 Cluster 2

$$P_3 = (8, 7)$$

$$P_4 = (9, 6)$$

$$\text{Centroid: } (8.5, 6.5)$$

 Cluster 3

$$P_5 = (5, 2)$$

$$P_6 = (6, 3)$$

$$\text{Centroid: } (5.5, 2.5)$$



**Summary:** The algorithm converged in 2 iterations. Total distance calculations:  $6 \text{ points} \times 3 \text{ centroids} \times 2 \text{ iterations} = 36$  distance computations

## K-Means++ Initialization

Improved initialization strategy for K-Means

### Algorithm Steps

1 **First centroid:** Choose randomly from data points

2 **Subsequent centroids:** Probability proportional to squared distance

3 **Result:** Spreads initial centroids far apart in data space

Selection Probability

$$P(x) \propto D(x)^2$$

$D(x)$  = distance to nearest centroid



### Faster Convergence

Significantly reduces iterations to convergence



### Better Quality

Large gains in clustering quality



### Provable Guarantee

$O(\log k)$  approximation to optimal solution



### Widely Adopted

Default in scikit-learn and other libraries



### Trade-off Analysis

Small overhead in initialization → Large gains in quality and speed

# Hierarchical Clustering

Builds hierarchy of clusters (dendrogram)



## Agglomerative

Bottom-Up

Start with n clusters (each point)  
Merge iteratively



## Divisive

Top-Down

Start with 1 cluster (all points)  
Split iteratively

### Linkage Criteria

#### Single

Minimum distance

#### Complete

Maximum distance

#### Average

Mean distance

#### Ward

Minimizes variance



#### No k Required

Cut dendrogram at desired level



#### $O(n^3)$ Complexity

Not scalable to large datasets



#### Exploratory Analysis

Understanding data structure



## Agglomerative Clustering Algorithm

1

**Initialize**

Start with  $n$  clusters (each data point as individual cluster)

2

**Compute Distance Matrix**

Calculate distances between all pairs of clusters to create distance matrix

3

**Find Closest Pair**

Find the two closest clusters from the distance matrix

4

**Merge Clusters**

Merge the two closest clusters into one and record the merge distance

5

**Update Distance Matrix**

Recalculate distances between the newly formed cluster and other clusters

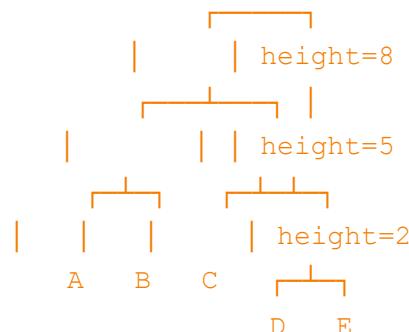
6

**Repeat**

Repeat steps 3-5 until all points form a single cluster



## Dendrogram Structure



### Y-axis (Height)

Represents the distance at which clusters merge. Higher values indicate merging of more dissimilar clusters



### Horizontal Lines

Represent the merging of two clusters. The height of the line indicates the distance at merge



### Cutting Dendrogram

Visualizes the height (distance) at which merges occur

Cutting horizontally at a desired height determines the number of clusters at that level

### Optimal Cut Point

Cutting where there are large height differences reveals natural cluster boundaries



### Phylogenetics

Essential for constructing phylogenetic trees representing evolutionary relationships among species



### Data Exploration

Visually understand natural groupings and hierarchical structure of data through dendrograms



### Intuitive Interpretation

Easy to understand cluster similarity by viewing merge order and distances at a glance



### Flexible Clustering

More flexible than k-means as you can choose the desired number of clusters after analysis

# DBSCAN - Density-Based Clustering

Density-Based Spatial Clustering of Applications with Noise

## Parameters

### $\epsilon$ (epsilon)

Neighborhood radius

### minPts

Minimum density threshold



Discovers clusters of arbitrary shape

## Advantages

- ✓ Finds outliers automatically
- ✓ No need to specify k
- ✓ Handles arbitrary shapes
- ✓ Works with non-spherical clusters

## Core Points

$\geq$  minPts neighbors within  $\epsilon$

## Challenges

- ⚠ Sensitive to parameters ( $\epsilon$ , minPts)
- ⚠ Struggles with varying densities
- ⚠ Difficulty in high dimensions



## Border Points

In neighborhood of core, but not core



## Noise Points

Neither core nor border (outliers)

# DBSCAN Step-by-Step Calculation Example

Detailed Process Using 2D Coordinate Data

Radius (Epsilon)

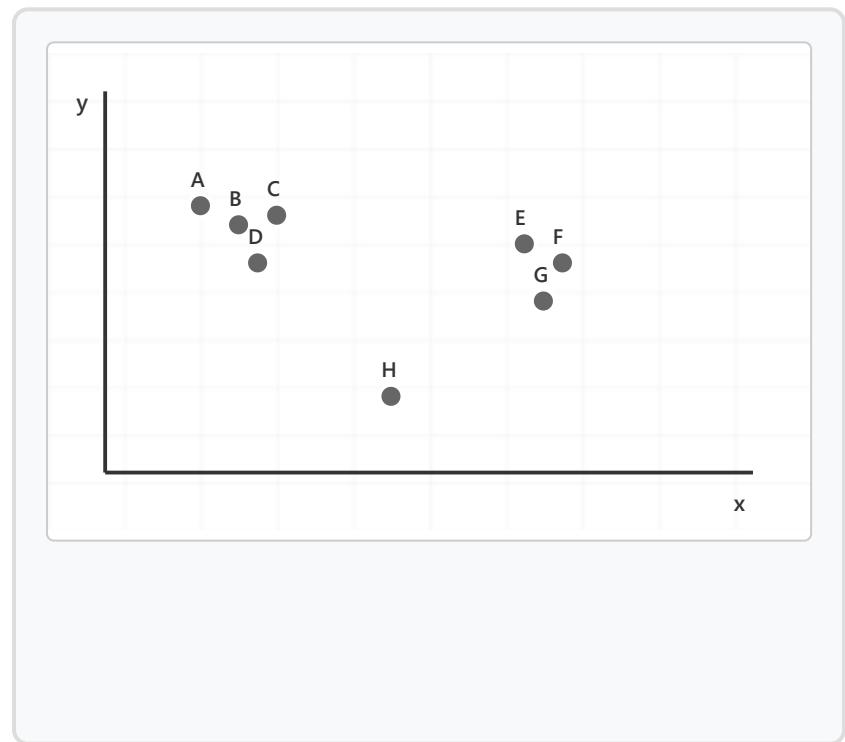
$\epsilon = 1.5$

Minimum Points

**minPts = 3**

1

## Dataset Definition



Point	x coord	y coord
A	2.0	3.0
B	2.5	2.5
C	3.0	2.7
D	2.8	1.8
E	6.5	2.5
F	7.0	2.0
G	6.8	1.5
H	5.0	0.5

**Total 8 points**

Data distributed on 2D plane

2

## Distance Calculation - Euclidean Distance

### Finding neighbors of Point A (2.0, 3.0)

$$d(A, B) = \sqrt{(2.5-2.0)^2 + (2.5-3.0)^2} = \sqrt{0.25 + 0.25} = 0.71 < 1.5 \checkmark$$

$$d(A, C) = \sqrt{(3.0-2.0)^2 + (2.7-3.0)^2} = \sqrt{1.0 + 0.09} = 1.04 < 1.5 \checkmark$$

$$d(A, D) = \sqrt{(2.8-2.0)^2 + (1.8-3.0)^2} = \sqrt{0.64 + 1.44} = 1.44 < 1.5 \checkmark$$

$$d(A, E) = \sqrt{(6.5-2.0)^2 + (2.5-3.0)^2} = \sqrt{20.25 + 0.25} = 4.53 > 1.5 \times$$

Neighbors of A: {B, C, D} → 3 points ≥ minPts(3) → Core Point

3

## Finding Neighbors for All Points

Point	Neighbors (within $\epsilon=1.5$ )	# Neighbors	Type
A	{B, C, D}	3	Core
B	{A, C, D}	3	Core
C	{A, B, D}	3	Core
D	{A, B, C}	3	Core
E	{F, G}	2	Non-Core
F	{E, G}	2	Non-Core
G	{E, F}	2	Non-Core
H	{}	0	Noise

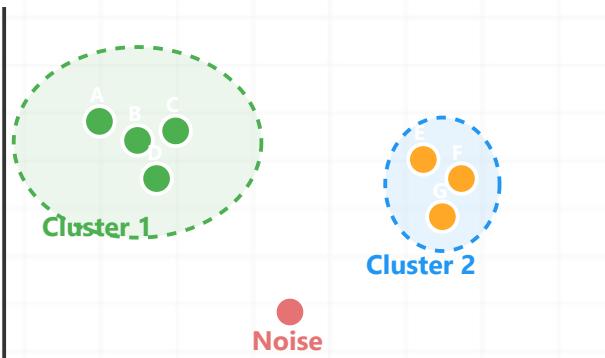
**Core Points:** A, B, C, D

**Non-Core Points:** E, F, G (neighbors < minPts)

**Isolated Point:** H (no neighbors)

4

## Cluster Formation - Connectivity Check



#### Cluster 1 Formation:

- A is Core Point (starting point)
- Add neighbors B, C, D of A
- B, C, D are all Core Points
- Connected together to form one cluster

#### Cluster 2 Check:

- E, F, G are Non-Core Points
- Neighbors of each other but  $< \text{minPts}$
- E, F, G are connected but
- No Core Point, needs special handling

H is not a neighbor of any Core Point → Noise

5

## Final Classification Results

Point	Final Classification	Cluster	Description
A	Core Point	Cluster 1	Meets density condition, cluster center
B	Core Point	Cluster 1	Meets density condition, cluster center
C	Core Point	Cluster 1	Meets density condition, cluster center
D	Core Point	Cluster 1	Meets density condition, cluster center
E	Border Point	Cluster 2	Insufficient neighbors, connected to other Borders
F	Border Point	Cluster 2	Insufficient neighbors, connected to other Borders
G	Border Point	Cluster 2	Insufficient neighbors, connected to other Borders

Point

Final Classification

Cluster

Description

## Clustering Results Summary

Cluster 1

**4 points**

A, B, C, D  
(All Core)

Cluster 2

**3 points**

E, F, G  
(All Border)

Noise

**1 point**

H  
(Outlier)

## Mean Shift

Non-parametric clustering using kernel density estimation

## Algorithm Process

## Key Features

### Detailed Calculation Example (2D Data)

- ✓ Automatically determines number of clusters
- ✓ Finds clusters of arbitrary shape
- ✓ Robust to outliers
- ✓ No assumption on cluster shape

#### Input Data Points

$x_1$   
(1, 2)

$x_2$   
(2, 3)

$x_3$   
(2, 2)

$x_4$   
(8, 7)

$x_5$   
(9, 8)

#### Goal

Starting from point  $x_1 = (1, 2)$ , we'll calculate its mean shift and update its position iteratively until convergence. We'll use Gaussian kernel with bandwidth  $h = 3$ .

#### Step 1: Estimate Density using Gaussian Kernel

##### Gaussian Kernel Formula

$$K(x) = \frac{1}{(2\pi h^2)} \cdot \exp(-||x||^2 / (2h^2))$$

$$\text{For } h = 3: K(x) = \frac{1}{56.55} \cdot \exp(-||x||^2 / 18)$$

Calculate distances from  $x_1 = (1, 2)$  to all points

Distance to  $x_1$ :

$$d_1 = \| (1, 2) - (1, 2) \| = 0$$

Distance to  $x_2$ :

$$d_2 = \| (1, 2) - (2, 3) \| = \sqrt{((1-2)^2 + (2-3)^2)} = \sqrt{2} \approx 1.414$$

Distance to  $x_3$ :

$$d_3 = \| (1, 2) - (2, 2) \| = \sqrt{((1-2)^2 + (2-2)^2)} = 1$$

Distance to  $x_4$ :

$$d_4 = \| (1, 2) - (8, 7) \| = \sqrt{((1-8)^2 + (2-7)^2)} = \sqrt{74} \approx 8.602$$

Distance to  $x_5$ :

$$d_5 = \| (1, 2) - (9, 8) \| = \sqrt{((1-9)^2 + (2-8)^2)} = \sqrt{100} = 10$$

## Step 2: Calculate Kernel Weights

Apply Gaussian kernel to each distance

$K(d_1)$ :

$$K(0) = \exp(-0^2/18) = 1.000$$

$K(d_2)$ :

$$K(1.414) = \exp(-2/18) \approx 0.895$$

$K(d_3)$ :

$$K(1) = \exp(-1/18) \approx 0.946$$

$K(d_4)$ :

$$K(8.602) = \exp(-74/18) \approx 0.015$$

$K(d_5)$ :

$$K(10) = \exp(-100/18) \approx 0.004$$

### Observation

Notice that points  $x_1, x_2, x_3$  (which are close) have high weights ( $\approx 0.9-1.0$ ), while distant points  $x_4, x_5$  have very low weights ( $< 0.02$ ). This ensures local density estimation.

## Step 3: Compute Mean Shift Vector

### Mean Shift Formula

$$m(x) = [\sum_i x_i \cdot K(\|x_i - x\|)] / [\sum_i K(\|x_i - x\|)] - x$$

### Calculate weighted sum of points

Numerator (weighted sum):

$$\begin{aligned} \sum_i x_i \cdot K(d_i) &= (1, 2) \cdot 1.000 + (2, 3) \cdot 0.895 + (2, 2) \cdot 0.946 + (8, 7) \cdot 0.015 + (9, 8) \cdot 0.004 \\ &= (1.000, 2.000) + (1.790, 2.685) + (1.892, 1.892) + (0.120, 0.105) + (0.036, 0.032) \\ &= (4.838, 6.714) \end{aligned}$$

Denominator (sum of weights):

$$\sum_i K(d_i) = 1.000 + 0.895 + 0.946 + 0.015 + 0.004 = 2.860$$

Weighted mean:

$$\bar{x} = (4.838, 6.714) / 2.860 = (1.691, 2.347)$$

Mean shift vector:

$$m(x_1) = (1.691, 2.347) - (1, 2) = (0.691, 0.347)$$

12  
34

## Step 4: Update Position

### Iteration 1 Update

New position:

$$x_1^{(1)} = x_1^{(0)} + m(x_1^{(0)}) = (1, 2) + (0.691, 0.347) = (1.691, 2.347)$$

### Iteration 2

Starting from  $x_1^{(1)} = (1.691, 2.347)$ :

- Recalculate distances to all points
- Compute new kernel weights
- Calculate new mean shift vector
- Update position:  $x_1^{(2)} \approx (1.85, 2.50)$

### Iteration 3 and beyond

Continue iterations until  $\|m(x)\| < \epsilon$  (convergence threshold)

Typically converges to a local mode around  $(1.9, 2.5)$  after 5-10 iterations

12  
34

## Step 5: Cluster Assignment

### Final Result

After running mean shift for all points:

- Points  $x_1, x_2, x_3$  converge to mode  $\approx (1.9, 2.5)$  → **Cluster 1**
- Points  $x_4, x_5$  converge to mode  $\approx (8.5, 7.5)$  → **Cluster 2**

Points that converge to the same mode are assigned to the same cluster.

### Convergence Criterion

The algorithm stops when the mean shift vector magnitude falls below a threshold:

$$\|m(x)\| < \epsilon, \text{ where } \epsilon \text{ is typically set to 0.01 or 0.001}$$

# Gaussian Mixture Models (GMM)

Model-based soft clustering approach

## Model Components

Data generated from mixture of k Gaussian distributions



Mean



Covariance



Weight

## K-means vs GMM

### K-means

Hard assignment  
Spherical clusters

### GMM

Soft assignment  
Elliptical clusters

## EM Algorithm



### Expectation Step

Assign probabilities to clusters



### Maximization Step

Update parameters ( $\mu, \Sigma, \pi$ )



### Iterate

Repeat until convergence



## Advantages

- ✓ Probability distribution over clusters
- ✓ Models elliptical clusters
- ✓ Soft assignment flexibility
- ✓ Captures uncertainty



## Model Selection

BIC • AIC

Choose optimal k



Assumes Gaussian distribution

# Cluster Evaluation Metrics

## Internal Metrics

### Silhouette Score

Range:  $[-1, 1]$

Measures separation and cohesion

Higher is better

### Davies-Bouldin Index

Range:  $[0, \infty)$

Ratio of within to between cluster distances

Lower is better

### Calinski-Harabasz Index

Range:  $[0, \infty)$

Ratio of between to within variance

Higher is better

### Inertia (WCSS)

Range:  $[0, \infty)$

Within-cluster sum of squares

## External Metrics

When ground truth labels are available

### Adjusted Rand Index (ARI)

Range:  $[-1, 1]$

Similarity between two clusterings

Closer to 1 is better

### Normalized Mutual Info (NMI)

Range:  $[0, 1]$

Mutual information between clusterings

Closer to 1 is better

### Fowlkes-Mallows Index

Range:  $[0, 1]$

Geometric mean of precision and recall

Closer to 1 is better

Minimize for K-means



### Visual Methods

Silhouette plots, cluster distributions



### Domain Validation

Do clusters make business sense?

★ Domain validation is the most important!



### Try Interactive Clustering Visualization

Test K-means, DBSCAN, Hierarchical, and MeanShift clustering algorithms

[Open Clustering Visualizer →](#)

**Part 3/4:**

# **Dimensionality Reduction**

- 14.** Curse of Dimensionality
- 15.** PCA Principles and Implementation
- 16.** Kernel PCA
- 17.** t-SNE Algorithm
- 18.** UMAP
- 19.** Autoencoder Dimensionality Reduction
- 20.** VAE Basics

## ⚠ Curse of Dimensionality

High dimensions create counterintuitive geometric properties



### Distance Concentration

All points become equidistant in high dimensions



### Volume Concentration

Data occupies tiny fraction of space



### Exponential Growth

Required samples grow exponentially with features



### Nearest Neighbors

Become meaningless



### Visualization

Impossible beyond 3D



### Computational Cost

Memory & time increase



### Solution: Dimensionality Reduction

Preserve structure in lower dimensions while maintaining essential information and relationships

# PCA: Principal Component Analysis

Linear dimensionality reduction technique

## Core Concepts

 **Goal:** Find directions of maximum variance

 **Principal Components:** Orthogonal eigenvectors of covariance matrix

 **Eigenvalues:** Variance explained by each component

## Component Selection

Use scree plot to choose k components

Example: Select k explaining 95% variance

## Limitations

 Linear transformation only

 Sensitive to feature scaling

 May lose interpretability

## Implementation Steps

1 Center data (subtract mean)



2 Compute covariance matrix



3 Eigen decomposition

## Applications

 Data compression

 Noise reduction

 Visualization

4

Project onto top k components



## PCA Step-by-Step Numerical Example

2D to 1D Dimension Reduction

<sup>12</sup>  
34

### Sample Dataset

Sample	$x_1$	$x_2$
1	2	1
2	3	5
3	4	3
4	5	6
5	6	7
6	7	8

#### Step 1: Calculate Mean Vector ( $\mu$ )

$$\begin{aligned}\mu_1 &= (2+3+4+5+6+7)/6 = 4.5 \\ \mu_2 &= (1+5+3+6+7+8)/6 = 5.0\end{aligned}$$

**Mean Vector:**  $\mu = [4.5, 5.0]$

#### Step 2: Center the Data (Subtract Mean)

```
X_centered = X - μ [-2.5, -4.0] [-1.5, 0.0] [-0.5, -2.0] [ 0.5, 1.0] [ 1.5, 2.0] [ 2.5, 3.0]
```

### Step 3: Compute Covariance Matrix

$$Cov(X) = (1/(n-1)) \times X_{centered}^T \times X_{centered}$$

Covariance Matrix: [ 3.5 3.75 ] [ 3.75 10.0 ]



**Interpretation:** The diagonal elements (3.5, 10.0) represent variances of  $X_1$  and  $X_2$ . Off-diagonal elements (3.75) show positive correlation between variables.



## Eigendecomposition & Component Selection



### Computing Eigenvectors and Eigenvalues

### Step 4: Solve Characteristic Equation

$$\det(Cov(X) - \lambda I) = 0$$

$$| 3.5-\lambda \ 3.75 | \ | 3.75 \ 10.0-\lambda | = 0 \quad (3.5-\lambda)(10.0-\lambda) - (3.75)^2 = 0 \quad \lambda^2 - 13.5\lambda + 20.94 = 0$$

#### Eigenvalues:

$\lambda_1 = 11.93$  (88.4% variance explained)

$\lambda_2 = 1.57$  (11.6% variance explained)

### Step 5: Calculate Eigenvectors

For  $\lambda_1 = 11.93$ : Eigenvector  $v_1 = [0.478, 0.878]$  (normalized) For  $\lambda_2 = 1.57$ : Eigenvector  $v_2 = [-0.878, 0.478]$  (normalized)



**Principal Component 1 (PC1):** Direction of maximum variance, pointing towards [0.478, 0.878]

### Step 6: Variance Explained

Component	Eigenvalue	% Variance	Cumulative %
PC1	11.93	88.4%	88.4%
PC2	1.57	11.6%	100.0%

### Step 7: Project Data onto PC1

$$Z = X_{centered} \times v_1$$

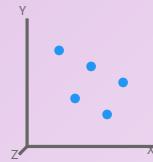
Transformed Data (1D):  $z_1 = [-2.5, -4.0] \cdot [0.478, 0.878] = -4.71$   $z_2 = [-1.5, 0.0] \cdot [0.478, 0.878] = -0.72$   $z_3 = [-0.5, -2.0] \cdot [0.478, 0.878] = -1.99$   $z_4 = [0.5, 1.0] \cdot [0.478, 0.878] = 1.12$   $z_5 = [1.5, 2.0] \cdot [0.478, 0.878] = 2.47$   $z_6 = [2.5, 3.0] \cdot [0.478, 0.878] = 3.83$

# 📊 PCA Visualization: 3D to 2D Example

Understanding dimensionality reduction visually

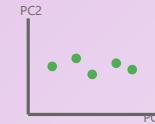
## 🟡 3D Data Visualization

Original 3D Data Space



X<sub>1</sub>   X<sub>2</sub>   X<sub>3</sub>

Reduced 2D Space (PC1 & PC2)



PC1   PC2



**Dimension Reduction:** From 3D (X<sub>1</sub>, X<sub>2</sub>, X<sub>3</sub>) → 2D (PC1, PC2) while preserving 95%+ of variance



Visual Interpretation Guide

 **PC1 Direction:** Points along the direction of maximum spread (variance) in the data. This is where the data varies the most.

 **PC2 Direction:** Orthogonal to PC1, captures the second largest variance. Always perpendicular to all previous components.

 **Data Projection:** Each data point is projected onto the new PC axes. Distance from origin indicates the component score.

 **Information Loss:** The reduction from 3D to 2D means discarding PC3. The eigenvalue of PC3 tells us how much information is lost.

## Python Implementation Example

Using NumPy and scikit-learn

### Method 1: NumPy (From Scratch)

```
# Import libraries
import numpy as np

# Sample data
X = np.array([[2, 1], [3, 5], [4, 3],
              [5, 6], [6, 7], [7, 8]])
```

```
# Step 1: Center the data
X_mean = np.mean(X, axis=0)
X_centered = X - X_mean

# Step 2: Compute covariance matrix
cov_matrix = np.cov(X_centered.T)

# Step 3: Eigendecomposition
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 4: Sort by eigenvalues
idx = eigenvalues.argsort()[:-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Step 5: Project data onto PC1
PC1 = eigenvectors[:, 0]
X_pca = X_centered @ PC1

print("Eigenvalues:", eigenvalues)
print("PC1:", PC1)
print("Transformed data:", X_pca)
```

## ⚡ Method 2: scikit-learn (Production Ready)

```
# Import PCA from scikit-learn
from sklearn.decomposition import PCA
import numpy as np

# Sample data
```

```

X = np.array([[2, 1], [3, 5], [4, 3],
             [5, 6], [6, 7], [7, 8]])

# Create PCA object (reduce to 1 component)
pca = PCA(n_components=1)

# Fit and transform
X_pca = pca.fit_transform(X)

# Access results
print("Explained variance ratio:",
      pca.explained_variance_ratio_)
print("Principal components:",
      pca.components_)
print("Transformed data:", X_pca)

# For 3D to 2D reduction
pca_3d = PCA(n_components=2)
X_3d_to_2d = pca_3d.fit_transform(X_3d_data)

```

### Before PCA

- ✓ Original dimensions: 3D or more
- ✓ All features present
- ✓ Full information retained
- ✗ Computational cost high
- ✗ Difficult to visualize
- ✗ Possible multicollinearity

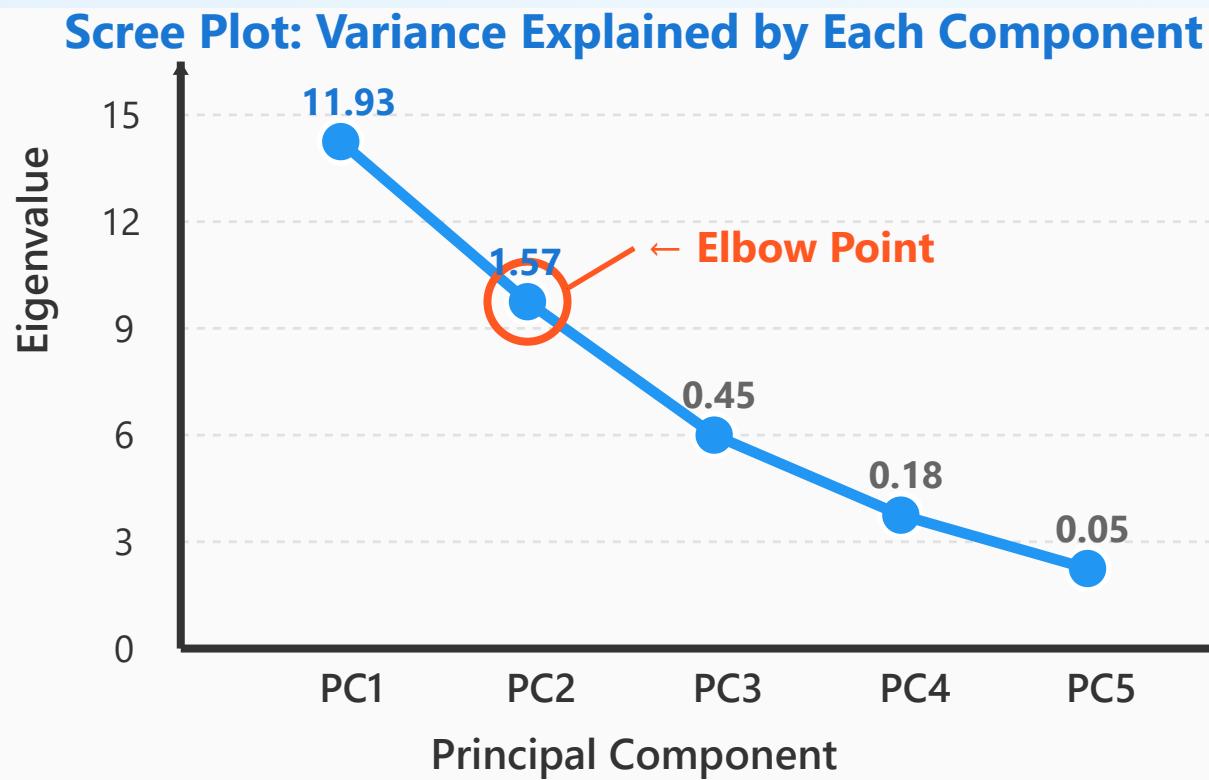
### After PCA

- ✓ Reduced dimensions: 2D or 1D
- ✓ Uncorrelated components
- ✓ 95%+ variance retained
- ✓ Faster computation
- ✓ Easy visualization
- ✗ Interpretability reduced

## 📈 Component Selection & Scree Plot

How many components should we keep?

### 📊 Scree Plot Analysis



**Elbow Method:** Choose the number of components at the "elbow" point where the eigenvalue curve flattens out. In this example, keeping 2 components is optimal.

Decision Criteria for Component Selection

Method	Rule	Example
<b>Variance Threshold</b>	Keep components explaining 95% variance	If $PC1+PC2 = 96\%$ , keep 2 components
<b>Kaiser Rule</b>	Keep eigenvalues $> 1$	If $\lambda_1=11.93, \lambda_2=1.57$ , keep both
<b>Elbow Method</b>	Visual inspection of scree plot	Choose point where curve bends
<b>Fixed Dimension</b>	Reduce to 2D or 3D for visualization	Always keep exactly 2 or 3 PCs

## 🎓 Practical Guidelines

### ✓ When to Use PCA:

- High-dimensional data ( $p > 50$  features)
- Features are highly correlated
- Need for data visualization
- Computational efficiency required
- Noise reduction in signal processing

### ✗ When NOT to Use PCA:

- Need to maintain feature interpretability
- Features have different scales (standardize first!)
- Non-linear relationships in data (use Kernel PCA)
- Small sample size ( $n < p$ )
- Sparse data (many zeros)



## Real-World PCA Applications

From theory to practice



### Case Study 1: Image Compression

**Problem:** A  $100 \times 100$  grayscale image has 10,000 pixels (dimensions)

#### PCA Solution

**Step 1:** Treat each image as a 10,000-dimensional vector

**Step 2:** Apply PCA to find principal components

**Step 3:** Keep top 100 components (99% variance)

**Step 4:** Reconstruct image using only 100 components

**Result:** 100 $\times$  compression ratio with minimal quality loss!

Storage: 10,000  $\rightarrow$  100 coefficients per image



### Case Study 2: Stock Market Analysis

**Problem:** Analyzing 500 stocks (S&P 500) with daily returns

### PCA Insights

```
# Stock returns data: 252 days × 500 stocks
from sklearn.decomposition import PCA
import pandas as pd

# Apply PCA
pca = PCA(n_components=10)
pc_scores = pca.fit_transform(stock_returns)

# Analyze variance explained
variance_explained = pca.explained_variance_ratio_
cumulative_variance = variance_explained.cumsum()

print("PC1 explains:", variance_explained[0])
# Output: PC1 explains: 0.42 (42% of market movement!)
```

**Interpretation:** PC1 often represents "the market" - overall market movement. PC2-PC3 capture sector-specific effects.



### Case Study 3: Gene Expression Analysis

**Problem:** 20,000 genes measured across 100 patients

## Bioinformatics Application

Original	After PCA	Benefit
20,000 dimensions	50 dimensions	400× reduction
Impossible to visualize	2D/3D plot	Pattern discovery
High noise	Filtered signal	Better classification



PCA revealed hidden patient subgroups that weren't visible in original data!

### Reference

#### Interactive 3D Visualization Tutorial:

[LearnPCA - Visualizing PCA in 3D](#)

# Kernel PCA

Non-linear extension of PCA using kernel trick



## Linear PCA

- Linear transformations only
- Works in original space
- Fast computation
- Limited to linear patterns



## Kernel PCA

- Captures non-linear patterns
- Projects to high-dim space
- No explicit computation
- More expressive power



## Common Kernel Functions



### RBF (Gaussian)

Most popular choice



### Polynomial

For polynomial features



### Sigmoid

Neural network-like



## Key Advantages

- ✓ Handles non-linear relationships
- ✓ No explicit feature mapping
- ✓ Works with complex data structures



## Trade-off

More powerful but computationally expensive.  
Requires careful hyperparameter tuning.

## The Kernel Trick

### How It Works

#### Without Kernel Trick

Explicitly map data to high-dimensional space → Computationally expensive for very high dimensions

#### With Kernel Trick

Compute inner products in high-dimensional space without explicit mapping → Much more efficient!

#### Key Insight

Kernel function  $K(x, y)$  computes the dot product in feature space without ever computing the coordinates in that space

#### Mathematical Beauty

Instead of computing  $\varphi(x) \cdot \varphi(y)$  explicitly, we directly compute  $K(x, y) = \varphi(x) \cdot \varphi(y)$

## Kernel PCA Process

### Step-by-Step Algorithm

1

2

### Choose Kernel

Select appropriate kernel function (RBF, polynomial, etc.) and set hyperparameters

### Compute Kernel Matrix

Calculate pairwise similarities between all data points using kernel function

3

### Center the Matrix

Center the kernel matrix in feature space to ensure zero mean

4

### Find Eigenvectors

Compute eigenvalues and eigenvectors of the centered kernel matrix

5

### Sort Components

Sort eigenvectors by eigenvalues in descending order

6

### Project Data

Project original data onto top k principal components for dimensionality reduction



## Visual Example



### Donut-Shaped Data Separation

#### Problem: Non-Linearly Separable Data

Imagine red points forming a circle in the center, surrounded by blue points in a donut shape. In 2D, no straight line can separate these classes.

#### Solution: Transform to Higher Dimension

Apply transformation:  $f(x, y) = (x, y, 2x^2 + 2y^2)$  to map 2D  $\rightarrow$  3D

$$f((x, y)) = (x, y, 2x^2 + 2y^2)$$

### 💡 Result: Linear Separation

In 3D space, the classes become linearly separable. Kernel PCA can now find principal components that effectively separate red and blue points!

## 🌐 Real-World Applications

### 🚀 Where Kernel PCA Shines



#### Facial Recognition

Captures complex non-linear facial features for identity verification and security systems



#### NLP & Text Analysis

Reduces dimensionality of text data for sentiment analysis, document clustering, and classification tasks



#### Genomics

Analyzes gene expression data and DNA sequences where non-linear biological relationships exist



#### Financial Modeling

Captures complex patterns in stock prices and market data for prediction and risk analysis



#### Image Processing



#### Speech Recognition

Object detection and recognition by extracting non-linear visual features from image data

Processes audio signals to identify non-linear patterns in speech for better transcription accuracy

## Kernel Selection Guide

### Detailed Kernel Comparison

#### RBF (Radial Basis Function) Kernel

**Best for:** Most general-purpose applications, unknown data patterns

$$K(u, v) = \exp(-\gamma \|u - v\|^2)$$

**Hyperparameter  $\gamma$ :** Controls influence radius. Larger  $\gamma$  = more local influence

#### Polynomial Kernel

**Best for:** Data with polynomial relationships, specific degree interactions

$$K(u, v) = (\gamma u \cdot v + c)^d$$

**Hyperparameters:** degree  $d$ , coefficient  $\gamma$ , constant  $c$

#### Sigmoid Kernel

**Best for:** Neural network-like transformations

$$K(u, v) = \tanh(\gamma u \cdot v + c)$$

**Note:** Similar to neural network activation functions

### Selection Strategy

Use cross-validation to test different kernels and hyperparameters. Start with RBF as default, then experiment with polynomial if domain knowledge suggests polynomial relationships.

# t-SNE Algorithm

t-Distributed Stochastic Neighbor Embedding

Non-linear dimensionality reduction for visualization

## Two-Step Process

1 Compute pairwise similarities in high-dimensional space



2 Optimize low-dimensional embedding to match similarities

## Key Features

 **Preserves local structure:** Similar points stay close

 **Student t-distribution:** Avoids crowding problem

 **Stochastic:** Different runs produce different results

## Key Parameter

### Perplexity

Range: 5 – 50

Balances local vs global structure

## Primary Use

 2D/3D visualization

 NOT for general dim reduction

## Limitations

 Computationally expensive:  $O(n^2)$

 Best with <10k samples

 Non-deterministic results

 Sensitive to hyperparameters



## Step-by-Step Example: MNIST Digits

3D → 2D Visualization Process

## 1

# Input: High-Dimensional Data

## Data Preparation

MNIST handwritten digit data (simplified example)

**Original dimensions:** 784D ( $28 \times 28$  pixels) → simplified to 3D

**Number of samples:** 12 (4 samples each for digits 0, 1, 2)

$$\left. \begin{array}{l} X = [x_1, x_2, \dots, x_{12}] \\ x_i \in \mathbb{R}^3 \end{array} \right\}$$

Sample	Label	$x_1$	$x_2$	$x_3$
1	0	2.1	3.4	1.2
2	0	2.3	3.1	1.5
3	0	1.9	3.6	1.0
4	0	2.4	3.3	1.3
...	...	...	...	...

## 3D Space Visualization

High-Dimensional Space (3D)



Digit 0



Digit 1



Digit 2



### Similarity Calculation (Gaussian Distribution)

Compute conditional probabilities between each pair of points

$$\begin{aligned} p(j|i) &= \exp(-\|x_i - x_j\|^2 / 2\sigma_i^2) / \sum_k \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2) \\ p_{ij} &= (p(j|i) + p(i|j)) / 2n \end{aligned}$$

**Perplexity = 5** (determines number of neighbors)

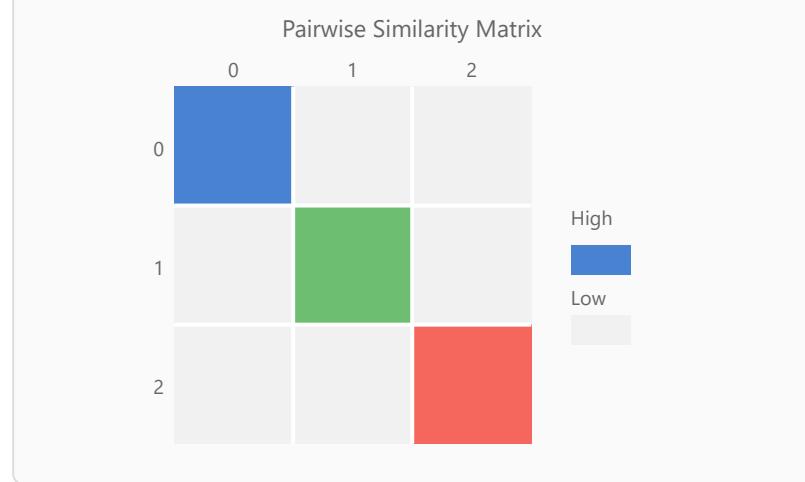
**Intermediate result:** 12×12 similarity matrix P

generated

- Same class:  $p_{ij} \approx 0.08 \sim 0.12$
- Different class:  $p_{ij} \approx 0.001 \sim 0.005$

	S1	S2	S5	S9
S1(0)	-	0.095	0.003	0.001
S2(0)	0.095	-	0.002	0.001
S5(1)	0.003	0.002	-	0.089
S9(2)	0.001	0.001	0.089	-

### Similarity Matrix Heatmap



## 3

### Initialize Low-Dimensional Embedding

#### Random Initialization in 2D Space

Randomly place points in the target dimension (2D)

$$\begin{aligned} \mathbf{Y}^{(0)} &= [\mathbf{y}_1^{(0)}, \mathbf{y}_2^{(0)}, \dots, \mathbf{y}_{n-2}^{(0)}] \\ \mathbf{y}_i^{(0)} &\sim \mathcal{N}(0, 0.0001 \cdot \mathbf{I}) \end{aligned}$$

##### Initial state:

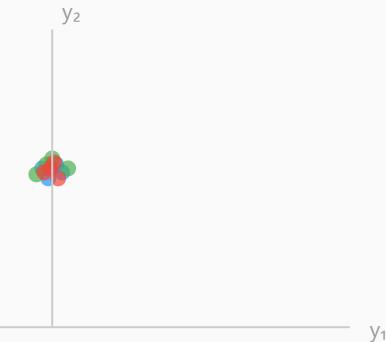
- All points clustered near the origin
- No class structure
- No meaningful patterns yet

**Iteration 0:** Completely random placement

Cost = Very high ( $\approx 8.5$ )

#### Initial Random Embedding

Iteration 0: Random Initialization



## 4

### Iterative Optimization (Gradient Descent)

## Matching Similarities in Low-Dimensional Space

Compute low-dimensional similarities using Student t-distribution

$$q_{ij} = (1 + ||y_i - y_j||^2)^{-1} / \sum_{kl} (1 + ||y_k - y_l||^2)^{-1}$$

$$\text{Cost} = KL(P || Q) = \sum_{ij} p_{ij} \log(p_{ij}/q_{ij})$$

### Gradient:

$$\partial C / \partial y_i = 4 \sum_j (p_{ij} - q_{ij}) (y_i - y_j) (1 + ||y_i - y_j||^2)^{-1}$$

### Progress:

Iteration	Cost (KL)	Status
0	8.523	Random
100	3.214	Clusters forming
500	0.892	Well separated
1000	0.435	Converged

**Learning rate:** Early exaggeration ( $\eta=200$ ) → Fine-tuning ( $\eta=50$ )

## Optimization Progress

Iterations: 0 → 100 → 500 → 1000

Iter 0 Iter 100 Iter 500 Iter 1000



Iter 100



Iter 500



Iter 1000



Cost (KL Divergence)

High

Low

Iterations



### Final Visualization Result

Low-dimensional embedding after convergence

#### Result Analysis:

- ✓ Same classes clustered together
- ✓ Different classes clearly separated
- ✓ Local structure preserved
- ✓ Visually interpretable

#### Performance Metrics:

- Final Cost: 0.435
- Convergence time: ~15s
- Total Iterations: 1000

#### Key Characteristics:

- Inter-cluster distances are meaningless
- Cluster sizes are also meaningless
- Only local structure is reliable
- Different results on re-runs possible

### Final t-SNE Visualization

Converged 2D Embedding



Digit 0



Digit 1



Digit 2

✓ Clusters Well Separated

t-SNE



### Key Takeaways

#### What t-SNE Preserves

- Close neighbor relationships

#### What t-SNE Does NOT Preserve

- Absolute distances

- Local cluster structure
- Density within same class
- Relative distances between clusters
- Cluster sizes/densities

### ⚡ Practical Tips:

- Adjust Perplexity: Small values (5-10) emphasize local structure, large values (30-50) consider global structure
- Run multiple times to verify stable patterns
- For >10,000 samples, first reduce to 50D with PCA, then apply t-SNE
- Use only for visualization purposes, not suitable for classification/regression preprocessing

# UMAP

Uniform Manifold Approximation and Projection

★ State-of-the-art for visualization and dimensionality reduction



## t-SNE

- Slower computation
- Local structure focus
- Limited scalability
- Mainly for visualization



## UMAP

- Faster computation
- Local + global structure
- Better scalability
- Visualization + reduction



## Mathematical Foundation

Riemannian geometry & topological data analysis



## Scalability

Handles larger datasets than t-SNE



## Structure Preservation

Both local and global structure



## Supervised Mode

Supports dimensionality reduction with labels



## Key Parameters

n\_neighbors

min\_dist



## UMAP Computation Principles

**1**

### High-dimensional Graph Construction

Find k-nearest neighbors (k-NN) to create a weighted graph. Calculate distances from each data point to its neighbors to determine probabilistic connection strengths.

$$w(i,j) = \exp(-(d(i,j) - \rho_i) / \sigma_i)$$

**2**

### Fuzzy Simplicial Complex

Transform the weighted graph into a fuzzy topological structure. This is a mathematical representation that preserves the topological properties of the data.

$$P(i \leftrightarrow j) = w(i,j) + w(j,i) - w(i,j) \times w(j,i)$$

**3**

### Low-dimensional Optimization

Optimize the low-dimensional embedding using Stochastic Gradient Descent (SGD). Minimize cross-entropy to reduce the difference between high-dimensional and low-dimensional structures.

$$\text{Loss} = \sum P(i,j) \log(P(i,j)/Q(i,j)) + (1-P(i,j)) \log((1-P(i,j))/(1-Q(i,j)))$$



## High-dimensional Space

- ▶ Local metric estimation
- ▶ k-NN graph construction
- ▶ Topological structure representation



## Low-dimensional Space

- ▶ Initial embedding generation
- ▶ Iterative optimization
- ▶ Structure preservation verification

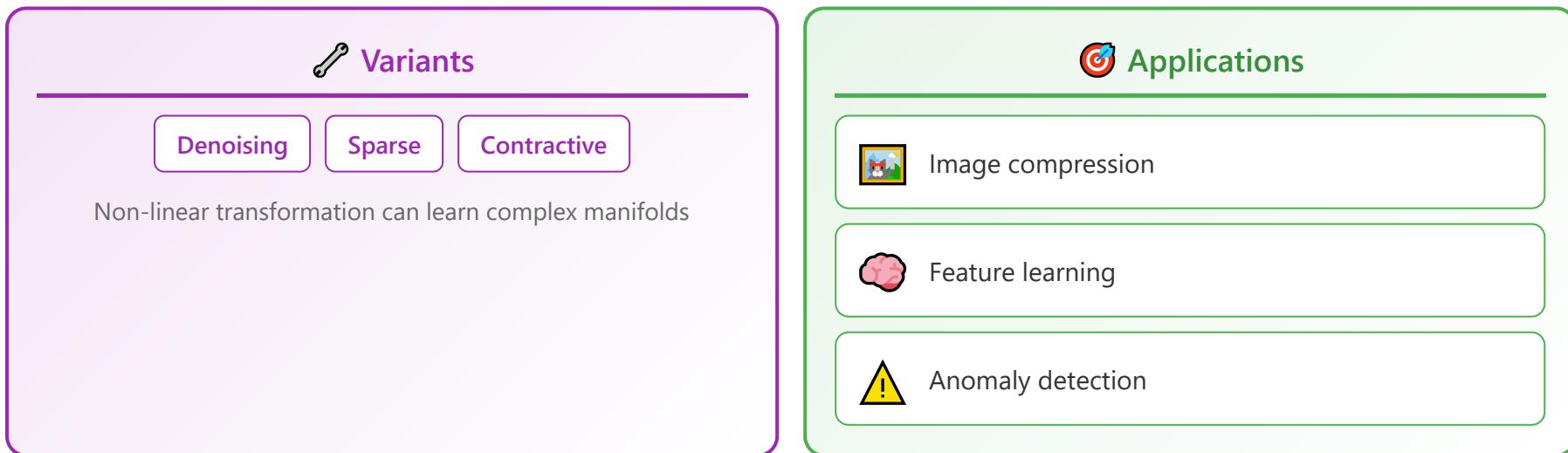
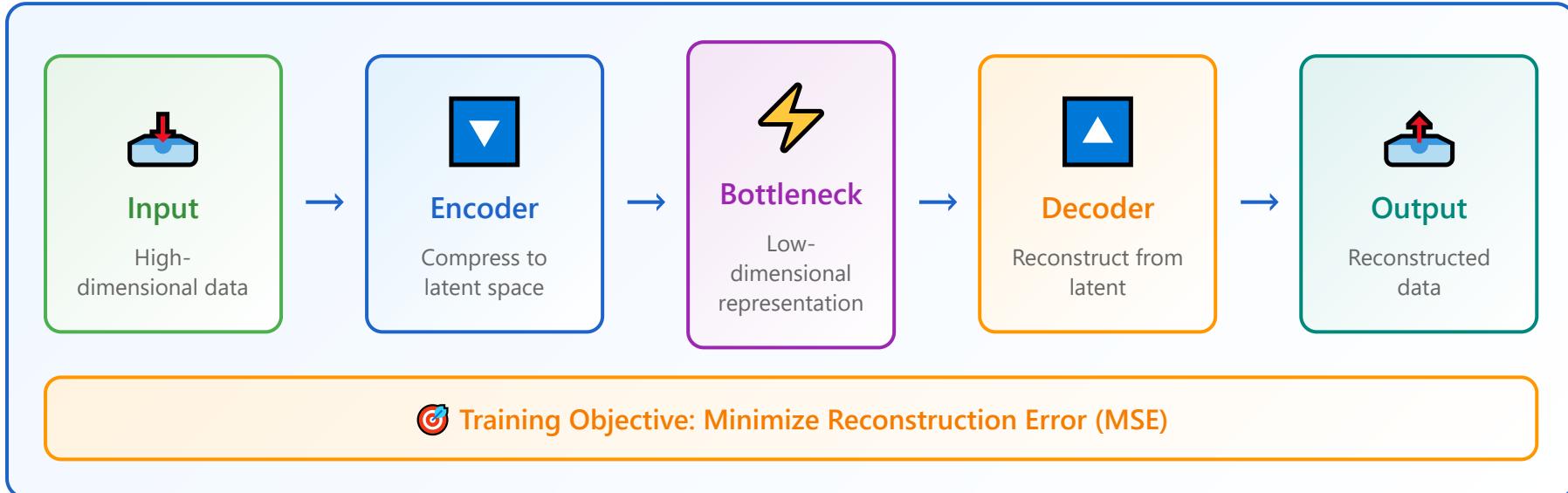


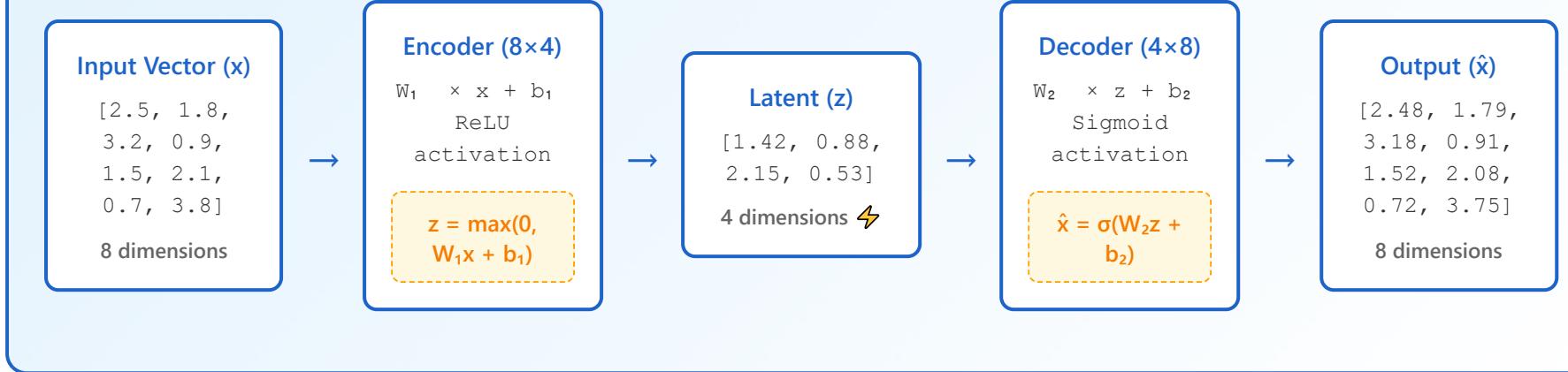
## Interactive UMAP Explorer

[Try UMAP Explorer →](#)

# Autoencoder for Dimensionality Reduction

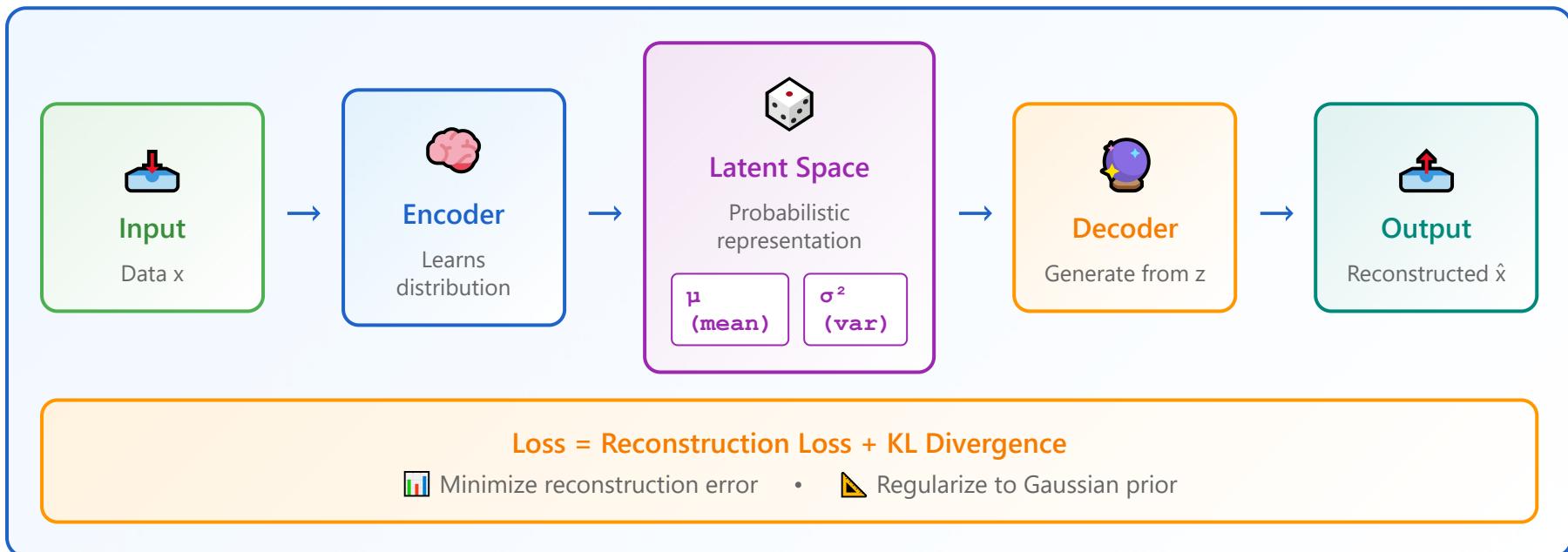
Neural network approach with encoder-bottleneck-decoder architecture





# VAE: Variational Autoencoder

Probabilistic generative model with latent distribution



## Key Concepts

**Probabilistic:** Distribution over latent space

**Reparameterization:** Enables backpropagation

**Generation:** Sample latent → new data

## Applications

Image generation

Data augmentation



Semi-supervised learning

**Part 4/4:**

# **Basic Anomaly Detection**

- 21.** Anomaly Detection Overview
- 22.** Statistical Methods
- 23.** Isolation Forest
- 24.** Hands-on: Implementation with scikit-learn
- 25.** Summary and Next Lecture Preview

## Anomaly Detection Overview

Identify rare items/events that differ from normal patterns



## Popular Anomaly Detection Algorithms

Key methods for identifying outliers and anomalies

### Statistical Methods

- **Z-Score**

Measures standard deviations from mean. Points beyond  $3\sigma$  are anomalies.

- **IQR (Interquartile Range)**

Uses quartiles Q1 and Q3. Outliers beyond  $Q1 - 1.5 \times IQR$  or  $Q3 + 1.5 \times IQR$ .

 Imbalanced data distribution

 Defining Normal behavior

Models normal data, identifies points in low-probability regions.

 Lack of labeled anomalies

 Evolving patterns over time

### Distance-based Methods

- **K-Nearest Neighbors (KNN)**

Detects anomalies based on distance to k-nearest neighbors.

- **Mahalanobis Distance**

Accounts for correlation between features in multivariate data.

### Density-based Methods

- **LOF (Local Outlier Factor)**

Measures local density deviation. Fast training, effective for local anomalies.

- **DBSCAN**

Clustering algorithm that identifies points not belonging to any cluster.

- **K-Means Clustering**

Points far from cluster centers are considered anomalies.

### Ensemble Methods

- **Isolation Forest**

Isolates anomalies using decision trees. Works well on most datasets.

- **Random Forest**

Can be adapted for anomaly detection in classification tasks.



# Advanced Methods & Learning Approaches

Modern techniques and learning paradigms for anomaly detection



## Machine Learning Methods

- **One-Class SVM**

Creates hypersphere around normal data, separating anomalies.

- **Autoencoders**

Neural networks that reconstruct data. High reconstruction error indicates anomalies.

- **LSTM Networks**

Captures temporal patterns in time-series data for sequence anomaly detection.



## Dimensionality Reduction

- **PCA (Principal Component Analysis)**

Reduces dimensions, highlights influential features and outliers.

- **t-SNE**

Non-linear technique for visualizing high-dimensional data and clusters.

- **UMAP**

Fast dimensionality reduction preserving global structure.



## Learning Paradigms

### Supervised

Requires labeled data with normal and anomalous examples

*SVM, Neural Networks, Random Forest*

### Unsupervised

No labels needed. Most common approach in practice

*Isolation Forest, LOF, K-Means, Autoencoders*

### Semi-supervised

Trained on normal data only, detects deviations

*One-Class SVM, Novelty Detection*



## Key Insight

Algorithm choice depends on data characteristics, computational resources, and whether labels are available. Isolation Forest and LOF are popular starting points due to their effectiveness and ease of use.



# Implementation & Popular Tools

Python libraries and practical considerations

## Python Libraries

### Scikit-learn

Isolation Forest, One-Class SVM, LOF, DBSCAN

### PyOD

Python Outlier Detection - 40+ algorithms specialized for anomaly detection

### TensorFlow/PyTorch

Deep learning models: Autoencoders, LSTM, Variational Autoencoders

### PyCaret

Low-code ML library with built-in anomaly detection module

## Implementation Tips

### Data Preprocessing

Normalize/standardize features for distance-based methods

### Feature Engineering

Create relevant features that capture domain knowledge

### Threshold Tuning

Adjust contamination rate based on expected anomaly percentage

### Model Comparison

Test multiple algorithms to find best fit for your data

### Interpretability

Use explainable methods when stakeholder understanding is critical



## Performance Considerations



### Speed

LOF: Fast training  
Isolation Forest: Moderate



### Accuracy

Depends on data characteristics & tuning



### Scalability

Consider distributed algorithms for big data



### Adaptability

Retrain models for evolving patterns



## Best Practice

Start with simple statistical methods (Z-score, IQR) for baseline performance, then progress to more sophisticated algorithms (Isolation Forest, One-Class SVM) if needed. Always validate results with domain experts.

# Statistical Methods for Anomaly Detection

Assume data follows a known distribution (usually Gaussian)



## Z-Score Method

$$|z| > \text{threshold} \quad (\text{e.g., } \pm 3\sigma)$$

Points beyond threshold are anomalies



## Box Plot Method

$$\begin{aligned} Q1 &- 1.5 \times \text{IQR} \\ Q3 &+ 1.5 \times \text{IQR} \end{aligned}$$

Points beyond  $1.5 \times \text{IQR}$  from quartiles



## Mahalanobis Distance

$$D^2 = (x - \mu)^T \Sigma^{-1} (x - \mu)$$

For multi-feature data



## Strengths

- ✓ Simple and interpretable
- ✓ Fast computation
- ✓ Well-understood theory
- ✓ Easy to implement



## Limitations

- ✗ Assumes specific distribution
- ✗ Sensitive to outliers in training
- ✗ May not work for complex patterns
- ✗ Requires distribution knowledge



## Works Well For

Low-dimensional data with known distribution



## Preprocessing Tip

Remove known anomalies before fitting distribution



## Z-Score Example

Student Test Score Analysis



## Box Plot Example

Employee Salary Data Analysis



## Mahalanobis Example

Customer Purchase Pattern (Amount,

Scores: 85, 90, 88, 92, 87, 89,  
150 Mean( $\mu$ ): 94.4 Std Dev( $\sigma$ ):  
22.9 Z-score(150) = 2.43

150 detected as outlier ( $|z| > 2$ )

Salary(\$10k): 300, 320, 310, 330,  
340, 325, 800 Q1 = 310, Q3 = 340  
IQR = 30 Upper Fence = 340 +  
 $1.5 \times 30 = 385$

\$8M detected as outlier

### Frequency)

Normal: (\$500k, 10 times)  
Suspicious: (\$2M, 2 times)  
Mahalanobis Distance considering  
correlation = 5.8

Abnormal pattern detected ( $D^2 > 4$ )



# Isolation Forest

Tree-based anomaly detection algorithm



**Key Principle: Anomalies are easier to isolate  
(require fewer splits)**



## Key Parameters

### n\_estimators

Number of trees in the ensemble

### contamination

Expected ratio of anomalies in dataset



## Algorithm Process

1 Build ensemble of random trees

2 Use random feature splits

3 Compute average path length

4 Calculate anomaly score



## Advantages

- ✓ No distribution assumptions
- ✓ Handles high dimensions
- ✓ Efficient and scalable
- ✓ Works on large datasets



## Anomaly Scoring



Anomalies



Normal Points

Time Complexity

$O(n \log n)$

Shorter paths  
Easy to isolate

Longer paths  
Hard to isolate

Scales to large datasets

## Isolation Process: Path Length Comparison

Why anomalies are easier to isolate

### Normal Data Point



Dense data around the point

Split 1

Split 2

Split 3

Split 4

Split 5

Split 6

Path Length

6

### Anomaly Point



Few data points nearby

Split 1

Split 2

Path Length

2

### Key Insight

Anomalies are isolated with **fewer node splits** → Shorter path length → **Higher anomaly score**

# Anomaly Score Formula

Normalizing path length to 0~1 range

## Formula

$$s(x, n) = 2^{-E(h(x))/c(n)}$$

$$c(n) = 2H(n-1) - 2(n-1)/n$$

$$H(i) = \ln(i) + 0.5772 \text{ (Euler's constant)}$$

**x**

Specific data point

**n**

Total number of data points

**$h(x)$**

Path length of data point x

**$E[h(x)]$**

Average of  $h(x)$  across all trees

**$c(n)$**

Average path length of all data (normalization constant)

## Score Interpretation

 **Anomaly**

$$E[h(x)] \rightarrow 0$$

 **Normal Data**

$$E[h(x)] \rightarrow n-1$$

$$s(x, n) \rightarrow 2^0$$

**≈ 1.0**

$$s(x, n) \rightarrow 2^{-(n-1)}$$

**≈ 0.0**

 **Note:** Score near 0.5 indicates ambiguous cases ( $E[h(x)] \approx c(n)$ )

# 💻 Hands-on: Implementation with scikit-learn

Practical implementation guide for unsupervised learning algorithms

## 📦 Key Imports

- from sklearn.cluster import KMeans
- from sklearn.cluster import DBSCAN
- from sklearn.decomposition import PCA
- from sklearn.ensemble import IsolationForest

## 📊 Evaluation Metrics

- silhouette\_score
- davies\_bouldin\_score
- calinski\_harabasz\_score

## ⌚ ML Pipeline

1 StandardScaler



2 Dimensionality Reduction



## 📈 Visualization Tools

matplotlib

seaborn

For plotting results and analysis

## ⭐ Best Practices

- ✓ Cross-validation for robustness
- ✓ Careful parameter tuning
- ✓ Domain knowledge validation
- ✓ Iterative experimentation





## Summary and Next Lecture Preview

### Today's Lecture: Unsupervised Learning

Discovering patterns without labels



#### Clustering

K-Means, DBSCAN, Hierarchical, GMM - each has trade-offs



#### Dimensionality Reduction

PCA, t-SNE, UMAP - visualization and compression



#### Anomaly Detection

Statistical methods, Isolation Forest - identify outliers



#### Key Considerations

Scaling, evaluation metrics, interpretability



### Next Lecture

#### Advanced Topics in Deep Learning



Deep Autoencoders



Generative Adversarial Networks (GANs)



Self-Supervised Learning



#### Practice Assignment

Apply these algorithms to real-world datasets

# Thank you

Ho-min Park

[homin.park@ghent.ac.kr](mailto:homin.park@ghent.ac.kr)

[powersimmani@gmail.com](mailto:powersimmani@gmail.com)