Unsupervised Learning

What is Unsupervised Learning?

- **Definition**: Unsupervised learning is a type of machine learning that involves learning patterns from untagged data. The system tries to learn without explicit instructions.
- **Objective**: The main goal is to model the underlying structure or distribution in the data to learn more about the data itself.
- **How It Works**: It identifies patterns based on the inherent similarities and differences in the data points. Algorithms are left to find and present the underlying structure in the data.
- **Key Points**:
  - No labels or annotations are provided, meaning the model tries to find the relationships between data points on its own.
  - Focuses on discovering the intrinsic structure of data, such as grouping or clustering of data points.
  - Useful in exploratory data analysis, dimensionality reduction, and feature learning.

Comparison: Supervised vs. Unsupervised vs. Semi-Supervised Learning

- **Supervised Learning**:
  - Data is labeled, providing a clear learning signal (input and output pairs).
  - Used for classification and regression tasks.
  - Example: Email spam filter (labeled as spam or not spam).
- **Unsupervised Learning**:
  - Data is unlabeled, and the model learns the patterns without any guidance.
  - Used for clustering, association, and dimensionality reduction.
  - Example: Customer segmentation in marketing.
- **Semi-Supervised Learning**:
  - Combines a small amount of labeled data with a large amount of unlabeled data during training.
  - Used when labeling data is expensive or time-consuming.
  - Example: Image recognition where only some images are labeled.
- **Key Differences**:
  - **Supervision**: The presence or absence of labels during training.
  - **Applications**: Depends on whether the outcome needs classification, clustering, or a mix of both.
  - **Data Requirements**: Supervised learning requires more comprehensive labeling, whereas unsupervised learning can work with raw, unstructured data.

Applications and Examples of Unsupervised Learning

- **Market Segmentation**:
  - Groups customers based on purchasing behavior, demographics, and engagement to tailor marketing strategies.
- **Anomaly Detection**:
  - Identifies unusual patterns that do not conform to expected behavior. Used in fraud detection, network security, and fault detection.
- **Dimensionality Reduction**:
  - Techniques like PCA are used to reduce the number of variables under consideration, helping in data visualization and improving model performance by eliminating noise.
- **Recommendation Systems**:
  - Analyzes user behavior and patterns to recommend items. For example, suggesting movies on streaming platforms or products on e-commerce sites.

Introduction to Clustering in Unsupervised Learning

Clustering is a method of unsupervised learning that involves grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups.

- The objective is to discover the inherent groupings in the data, such as grouping customers by purchasing behavior or articles by topics.

- **Importance of Clustering**:
  - **Data Exploration**: Offers initial insights into the data distribution and structure, facilitating hypothesis generation and feature selection.
  - **Pattern Recognition**: Essential in identifying patterns or behaviors in datasets, aiding in anomaly detection, customer segmentation, and bioinformatics.
  - **Efficiency in Computation**: By grouping similar data points, clustering can reduce the complexity of data processing for further analysis or supervised learning.

- **Key Concepts**:
  - **Similarity Measures**: The foundation of clustering. Common measures include Euclidean distance, Manhattan distance, and Cosine similarity, determining how 'close' data points are to each other.
  - **Types of Clustering**:
    - **Exclusive (Hard) Clustering**: Each data point belongs to exactly one cluster.
    - **Overlapping (Soft) Clustering**: Data points can belong to multiple clusters with varying degrees of membership.
    - **Hierarchical Clustering**: Builds a multilevel hierarchy of clusters by creating a dendrogram.
    - **Density-Based Clustering**: Forms clusters based on areas of high density, separating outliers in sparse areas.
- **Challenges and Considerations**:
  - **Determining the Number of Clusters**: Often, the optimal number of clusters is not known a priori and must be determined using methods like the Elbow Method or the Silhouette Method.
  - **Scalability and Computational Cost**: Some algorithms, like hierarchical clustering, may not scale well to very large datasets.
  - **Sensitivity to Initial Conditions**: Algorithms like K-means can produce different outcomes depending on initial seed points, requiring multiple runs for robust results.
- **Real-World Application Highlight**:
  - **Customer Segmentation**: By clustering customers based on their purchase history, demographics, and behavior, businesses can tailor marketing strategies, personalize customer service, and improve customer retention.

K-means Clustering

K-means is one of the simplest and most commonly used clustering algorithms that partitions n observations into k clusters where each observation belongs to the cluster with the nearest mean (cluster centers or centroids), serving as a prototype of the cluster.

- **Algorithm Steps**:
  - **Initialization**: Select k initial centroids (either randomly or based on some heuristic).
  - **Assignment**: Assign each data point to the nearest centroid based on the chosen distance metric (usually Euclidean distance), forming k clusters.
  - **Update**: Recalculate the centroids as the mean of all points in the cluster.
  - **Repeat**: Continue the assignment and update steps until the centroids no longer change significantly, indicating convergence.

Example

- Assume we have the following 2D points:
-
- Points = (1, 2), (1.5, 1.8), (5, 8), (8, 8), (1, 0.6), (9, 11)
-
- Steps of K-means Clustering
-
- 1. Initialization:
- - Choose K = 2 initial centroids randomly.
- - Let's assume the initial centroids are C1 = (1, 2) and C2 = (5, 8).
-
- 2. Assignment:
- - Calculate the distance of each point from the centroids and assign it to the nearest centroid.
- Use Euclidean distance $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

3. Update Centroids:
   - Compute the new centroid of each cluster as the mean of all points assigned to that cluster.

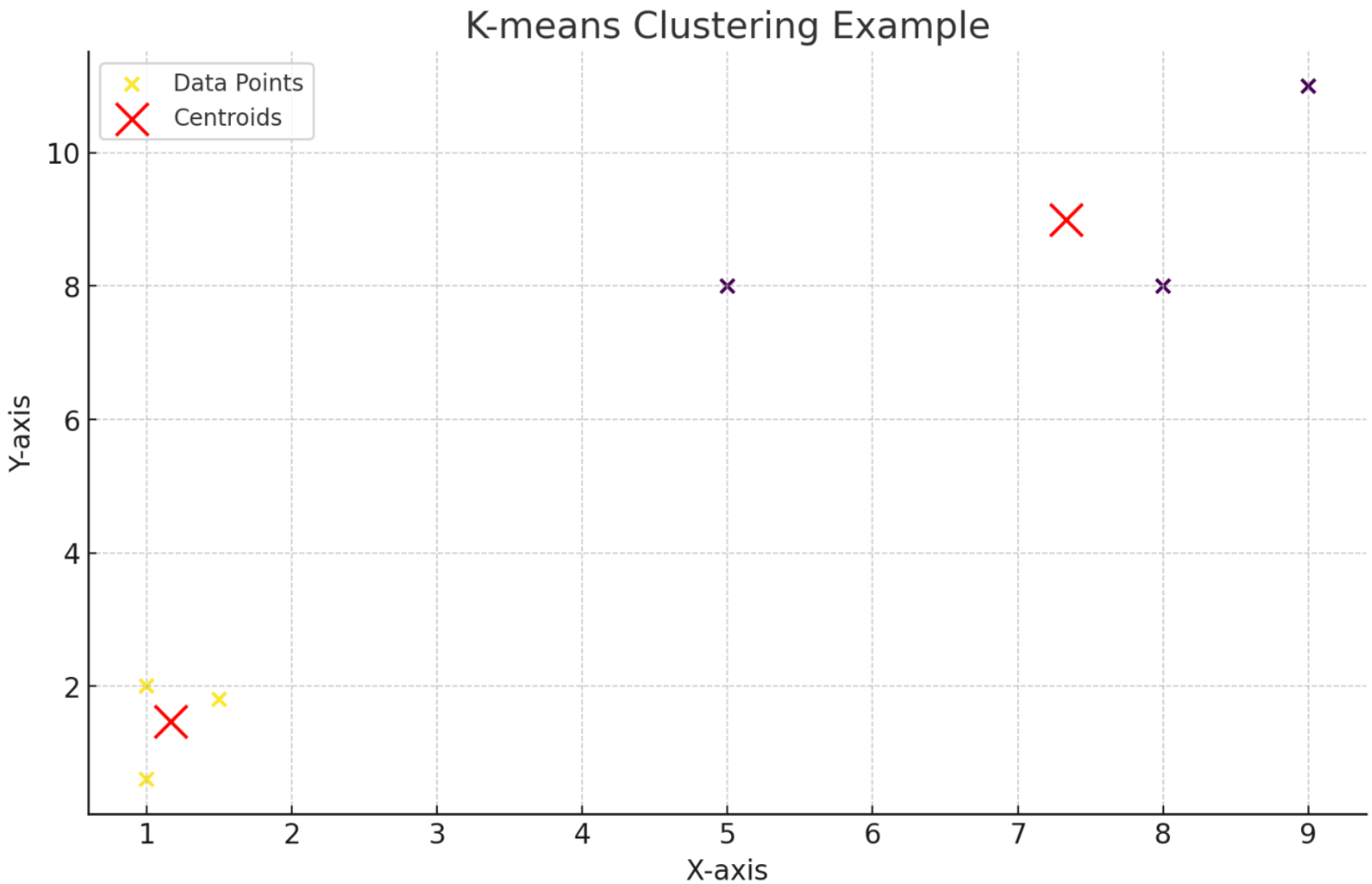
4. Repeat:
   - Repeat the assignment and update steps until the centroids do not change significantly.

1. Initialization:
   - C1 = (1, 2)
   - C2 = (5, 8)

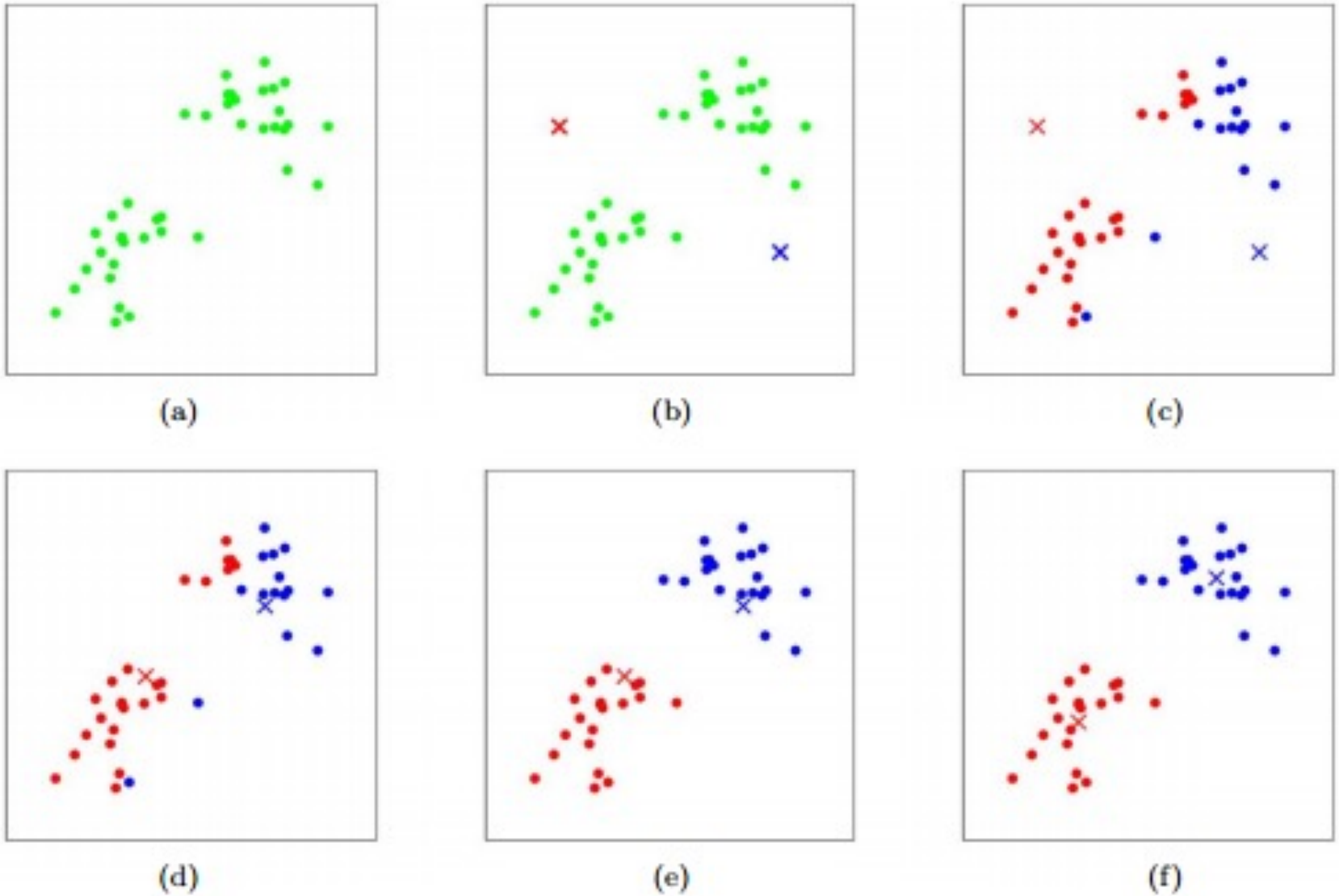
- Assignment:
- - For point (1, 2) :
- - Distance to C1 = $\sqrt{(1-1)^2 + (2-2)^2} = 0$
- - Distance to C2 = $\sqrt{(5-1)^2 + (8-2)^2} = 7.2$
- - Assign to C1.
-
- - For point (1.5, 1.8):
- - Distance to C1 = $\sqrt{(1-1.5)^2 + (2-1.8)^2} = 0.54$
- - Distance to C2 = $\sqrt{(5-1.5)^2 + (8-1.8)^2} = 7.12$
- - Assign to C1 .
-
- - For point (5, 8):
- - Distance to C1 = $\sqrt{(1-5)^2 + (2-8)^2} = 7.2$
- - Distance to C2 = $\sqrt{(5-5)^2 + (8-8)^2} = 0$
- - Assign to C2 .
-
- - Repeat similar calculations for remaining points.

- 3. Update Centroids:
- - Calculate the new centroids based on the mean of points in each cluster.
- - Assume points assigned to C1 are (1, 2), (1.5, 1.8), (1, 0.6):
-
- New $C1 = \left(\frac{1 + 1.5 + 1}{3}, \frac{2 + 1.8 + 0.6}{3} \right) = (1.17, 1.47)$

- - Assume points assigned to C2 are (5, 8), (8, 8), (9, 11):
-
- New $C2 = \left(\frac{5 + 8 + 9}{3}, \frac{8 + 8 + 11}{3} \right) = (7.33, 9)$
-
- 4. Repeat:
- - Reassign points based on new centroids and update centroids again if necessary.
- - The process continues until the centroids stabilize.
-


K-means Clustering Example
Data Points
Centroids
Y-axis
X-axis
10
8
6
4
2
1
2
3
4
5
6
7
8
9

- **Key Features**:
  - **Simplicity and Efficiency**: Easy to understand and implement, making it suitable for a wide range of applications.
  - **Scalability**: Works well with large datasets, especially with optimizations like the Elkan's algorithm.
  - **Versatility**: Applicable to any type of data where a distance measure can be defined.
- **Challenges**:
  - **Choosing k**: Determining the right number of clusters is critical and often achieved using methods like the Elbow Method, the Silhouette Coefficient, or the Gap Statistic.
  - **Sensitivity to Initial Centroids**: The final outcome can significantly depend on the initial selection of centroids. Techniques like k-means++ improve centroid initialization.
  - **Convexity Bias**: Assumes clusters are convex and isotropic, which might not be the case, leading to poor performance on complex shapes.
- **Variants and Extensions**:
  - **K-means++**: Improves the initialization phase, leading to better convergence.
  - **Mini-Batch K-means**: Uses small, random batches of data for each iteration, improving time efficiency for large datasets.
  - **Fuzzy K-means**: Allows data points to belong to multiple clusters with varying degrees of membership, providing a solution to the exclusive clustering limitation.
- **Application Example**:
  - **Image Compression**: K-means can reduce the number of colors in an image by clustering similar colors, significantly reducing the image's size while preserving visual similarity.


(a)
(b)
(c)
(d)
(e)
(f)

See Lesson 10.pynb

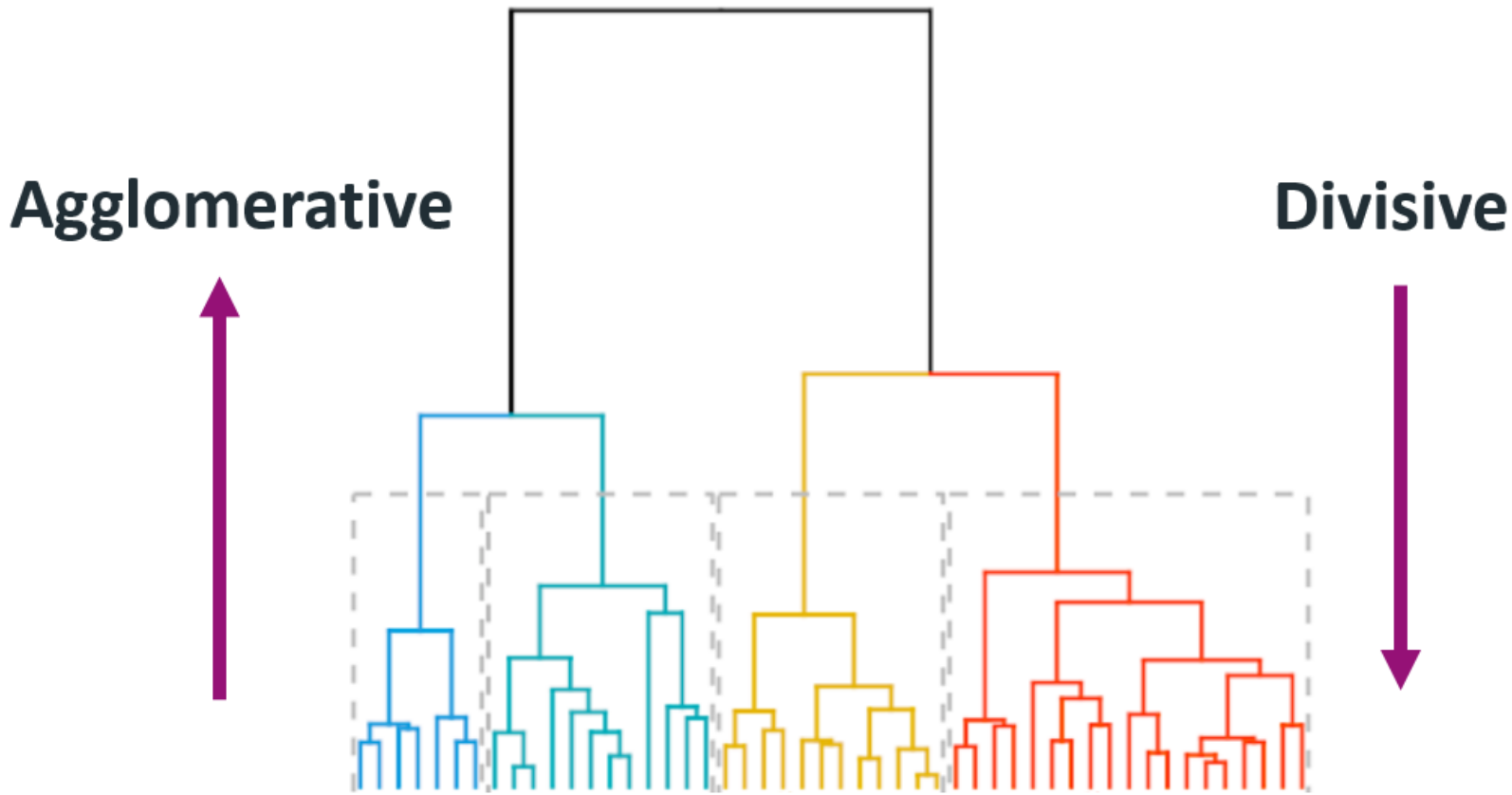
- Code Cell
- `10.1 K Means`

Hierarchical Clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Unlike K-means, it does not require pre-specifying the number of clusters to be generated. The process is visualized as a dendrogram - a tree-like diagram that records the sequences of merges or splits.

- **Types of Hierarchical Clustering**:
  - **Agglomerative (Bottom-Up)**: Starts with each data point as its own cluster and merges them into larger clusters.
  - **Divisive (Top-Down)**: Starts with all data points in a single cluster and recursively splits the most heterogeneous cluster until only singleton clusters remain.
- **Key Steps (Agglomerative)**:
  - **Initialization**: Treat each data point as a single cluster.
  - **Find the Closest Pair**: Identify the two clusters closest to each other based on a chosen distance metric (e.g., Euclidean, Manhattan, Cosine).
  - **Merge**: Combine the closest pairs into a single cluster.
  - **Update Distance Matrix**: Recalculate the distance between the new cluster and the original clusters.
  - **Repeat**: Continue until all data points are clustered into a single hierarchical tree.
- **Distance Metrics**:
  - The choice of distance metrics (between clusters) significantly impacts the clusters' shape. Common methods include:
    - **Single Linkage**: Distance between the closest members of clusters.
    - **Complete Linkage**: Distance between the farthest members of clusters.
    - **Average Linkage**: Average distance between all members of two clusters.
    - **Ward’s Method**: Minimize the variance within each cluster.

- **Advantages**:
  - **No Need to Specify Number of Clusters**: The dendrogram provides a rich representation that can be cut at different heights to yield different clustering solutions.
  - **Flexibility in Distance Metrics**: Allows the use of various distance metrics, tailored to the specific needs of the dataset.
- **Challenges**:
  - **Computational Complexity**: Especially for agglomerative clustering, the computational cost is high for large datasets, making it less scalable than K-means.
  - **Sensitivity to Noise and Outliers**: Similar to other clustering methods, hierarchical clustering is sensitive to noise and outliers, which can lead to misleading hierarchies.
- **Applications**:
  - **Gene Sequence Analysis**: Used in bioinformatics to find groups of genes with similar expression patterns.
  - **Customer Segmentation**: Groups customers based on purchasing behavior for targeted marketing.
  - **Document Clustering**: Organizes a large number of text documents into a hierarchy for easier navigation.


Agglomerative
Divisive

Example

- Initial Data Points
-
- Points =(1, 2), (1.5, 1.8), (5, 8), (8, 8), (1, 0.6), (9, 11)
-
- Initial Distance Matrix

Points	(1, 2)	(1.5, 1.8)	(5, 8)	(8, 8)	(1, 0.6)	(9, 11)
(1, 2)	0	0.54	7.21	10.63	1.40	12.73
(1.5, 1.8)	0.54	0	7.12	10.52	1.30	12.63
(5, 8)	7.21	7.12	0	3.00	8.22	5.39
(8, 8)	10.63	10.52	3.00	0	10.68	3.60
(1, 0.6)	1.40	1.30	8.22	10.68	0	13.04
(9, 11)	12.73	12.63	5.39	3.60	13.04	0

- Step-by-Step Merging
-
- 1. Merge (1, 2) and (1.5, 1.8):
- - New Centroid: (1.25, 1.9)
- - Update Distance Matrix:
- - Calculate distances from the new cluster (1.25, 1.9) to all other points.
-
- Updated Distance Matrix:

Points	(1.25, 1.9)	(5, 8)	(8, 8)	(1, 0.6)	(9, 11)
(1.25, 1.9)	0	7.04	10.47	1.32	12.50
(5, 8)	7.04	0	3.00	8.22	5.39
(8, 8)	10.47	3.00	0	10.68	3.60
(1, 0.6)	1.32	8.22	10.68	0	13.04
(9, 11)	12.50	5.39	3.60	13.04	0

- 2. Merge (1.25, 1.9) and (1, 0.6):
- - New Centroid: (1.17, 1.47)
- - Update Distance Matrix:
- - Calculate distances from the new cluster (1.17, 1.47) to all other points.
-
- Updated Distance Matrix:

Points	(1.17, 1.47)	(5, 8)	(8, 8)	(9, 11)
(1.17, 1.47)	0	7.23	10.62	12.64
(5, 8)	7.23	0	3.00	5.39
(8, 8)	10.62	3.00	0	3.60
(9, 11)	12.64	5.39	3.60	0

-
- 1. Merge (5, 8) and (8, 8):
- - New Centroid: (6.5, 8)
- - Update Distance Matrix:
-
- Updated Distance Matrix:

Points	(1.17, 1.47)	(6.5, 8)	(9, 11)
(1.17, 1.47)	0	8.23	12.64
(6.5, 8)	8.23	0	3.20
(9, 11)	12.64	3.20	0

- 2. Next Merge: (6.5, 8) and (9, 11):
- - New Centroid: (7.75, 9.5)
- - Update Distance Matrix:
-
- Updated Distance Matrix:

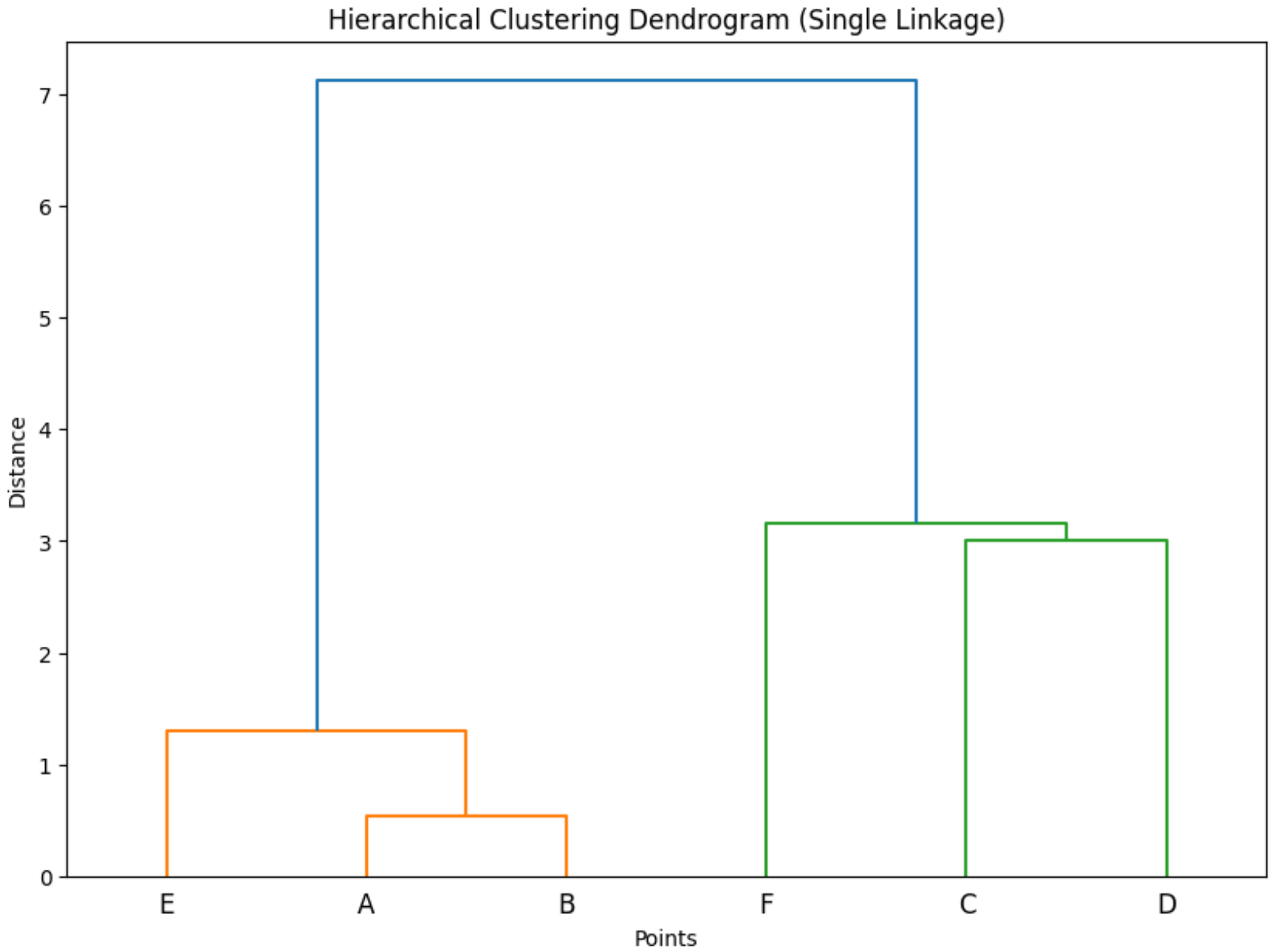
Points	(1.17, 1.47)	(7.75, 9.5)
(1.17, 1.47)	0	11.41
(7.75, 9.5)	11.41	0

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Sample data
X = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])

# Perform hierarchical clustering using single linkage
Z = linkage(X, method='single')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z, labels=['A', 'B', 'C', 'D', 'E', 'F'])
plt.title('Hierarchical Clustering Dendrogram (Single Linkage)')
plt.xlabel('Points')
plt.ylabel('Distance')
plt.show()
```


Hierarchical Clustering Dendrogram (Single Linkage)
Distance
0
1
2
3
4
5
6
7
E
A
B
F
C
D
Points

See Lesson 10.pynb

- Code Cell

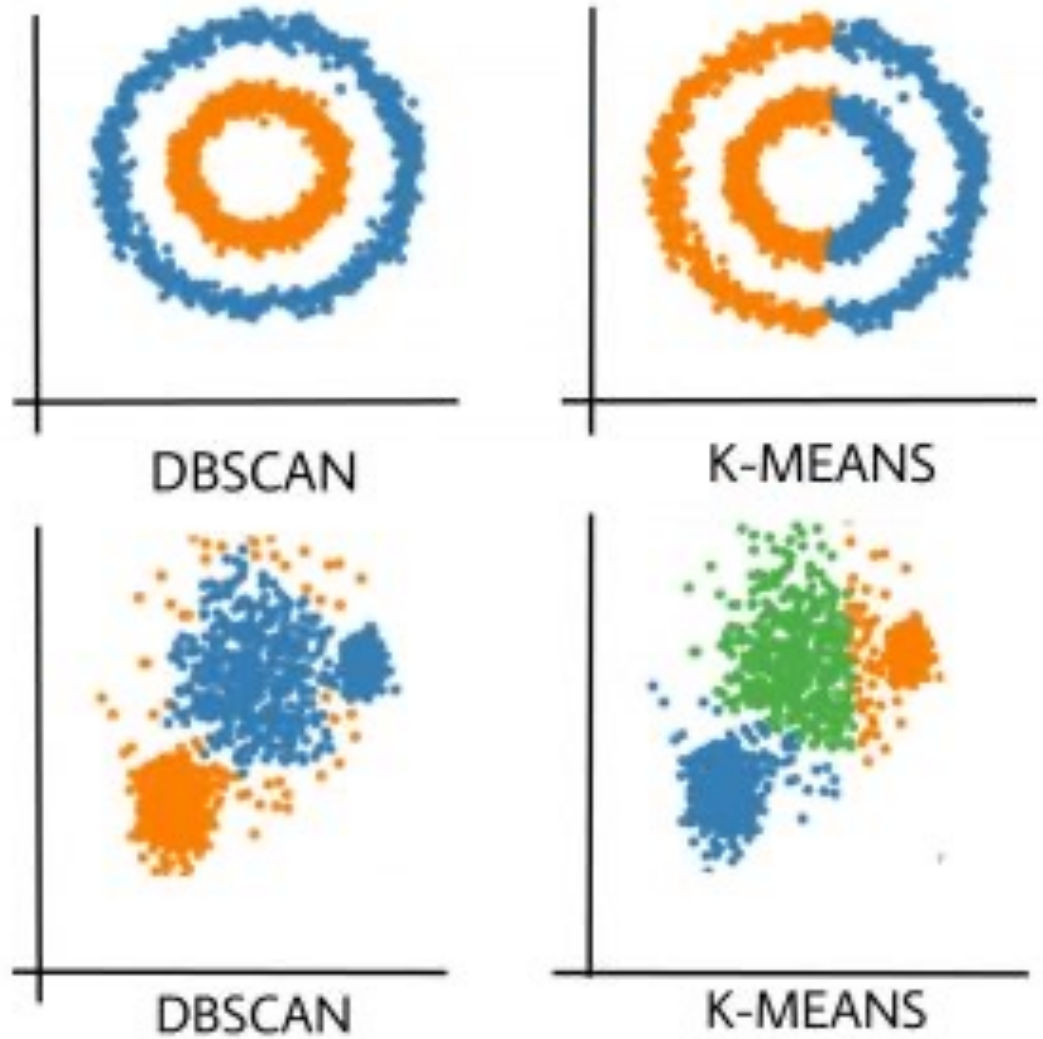
- `10.2 Hierarchical Clustering`

DBSCAN

- DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise, a popular clustering algorithm known for its ability to identify clusters of varying shapes and sizes, and its robustness to outliers. Unlike centroid-based algorithms like K-means, DBSCAN is based on density estimation.

- **Core Principles**:
  - **Density Estimation**: Clusters are defined as high-density areas separated by areas of low density. This allows for the identification of clusters with irregular shapes.
  - **Core Points**: A point is a core point if at least a minimum number of points (MinPts) are within a given distance (ε, epsilon) of it, indicating a dense region.
  - **Border Points**: Points that are not core points but are within ε distance of a core point.
  - **Noise Points**: Points that are neither core nor border points. These are considered outliers.
- **Algorithm Steps**:
  - **Classification of Points**: Each point is classified as a core, border, or noise point based on ε and MinPts.
  - **Cluster Formation**: Starting from a core point, the cluster is expanded by recursively adding all directly reachable points (core or border) to the cluster.
  - **Process Each Point**: Iterate through each point in the dataset, skipping already classified points, until all points are assigned to a cluster or marked as noise.

- **Advantages**:
  - **No Assumption on Cluster Shapes**: Effectively handles clusters of arbitrary shapes and sizes.
  - **Robustness to Outliers**: Naturally identifies and ignores noise or outlier data points.
  - **Minimal Input Parameters**: Requires only two parameters (ε and MinPts), making it relatively easy to configure.
- **Challenges**:
  - **Parameter Sensitivity**: Choosing appropriate values for ε and MinPts can be non-trivial and heavily influences the clustering outcome.
  - **Variable Density Clusters**: Struggles with clusters of varying densities, where a single set of parameters might not fit all clusters well.
  - **Scalability**: The computational complexity can be high for large datasets with many dimensions (curse of dimensionality).
- **Applications**:
  - **Geospatial Data Analysis**: Ideal for spatial data clustering such as identifying regions of similar weather conditions.
  - **Anomaly Detection**: Effective in distinguishing outliers from core groups in various domains, including fraud detection and network security.
  - **Ecology and Astronomy**: Used to identify regions of density in spatial data, such as animal habitats or celestial bodies.


DBSCAN
K-MEANS
DBSCAN
K-MEANS

Example

Let's consider the following 2D points:

Points = (1, 2), (2, 2), (2, 3), (8, 7), (8, 8), (25, 80)

Steps of DBSCAN Clustering

1. Parameter Initialization:
   - Epsilon (ϵ): The maximum distance between two points to be considered neighbors.
   - MinPts: The minimum number of points required to form a dense region (core point).

2. Classify Points:
   - Core Point: A point with at least MinPts neighbors within ϵ.
   - Border Point: A point that is not a core point but is within ϵ distance of a core point.
   - Noise Point: A point that is neither a core nor a border point.

3. Cluster Formation:
   - Expand Cluster: Start with a core point and recursively add all its reachable points (ϵ) to form a cluster.

- Parameters
- - ϵ = 2
- - MinPts = 2
-
- 1. Initial Data Points:

- Points = (1, 2), (2, 2), (2, 3), (8, 7), (8, 8), (25, 80)
-
- 2. Calculate Distances:

Points	(1, 2)	(2, 2)	(2, 3)	(8, 7)	(8, 8)	(25, 80)
(1, 2)	0	1	1.41	8.60	9.21	78.90
(2, 2)	1	0	1	7.81	8.49	78.10
(2, 3)	1.41	1	0	7.07	7.81	77.36
(8, 7)	8.60	7.81	7.07	0	1	73.41
(8, 8)	9.21	8.49	7.81	1	0	72.80
(25, 80)	78.90	78.10	77.36	73.41	72.80	0

3. Classify Points:

- Point (1, 2):
  - Neighbors: (2, 2), (2, 3)
  - MinPts: 3
  - Type: Core point

- Point (2, 2):
  - Neighbors: (1, 2), (2, 3)
  - MinPts: 3
  - Type: Core point

- Point (2, 3):
  - Neighbors: (1, 2), (2, 2)
  - MinPts: 3
  - Type: Core point

- Point (8, 7):
  - Neighbors: (8, 8)
  - MinPts: 2
  - Type: Core point

- Point (8, 8):
  - Neighbors: (8, 7)
  - MinPts: 2
  - Type: Core point

- Point (25, 80):
  - Neighbors: None within epsilon
  - MinPts: 0
  - Type: Noise point

-
- 4. Form Clusters:
-
- - Cluster 1: (1, 2), (2, 2), (2, 3)
- - All points within epsilon of each other and form a dense region.
-
- - Cluster 2: (8, 7), (8, 8)
- - Both points are within epsilon of each other and form a dense region.
-
- - Noise: (25, 80)
- - This point is not within epsilon of any core point and does not meet MinPts.

Example 2

-
- Parameters
-
- - epsilon = 2
- - MinPts = 3
-
- Step-by-Step Process
-
- 1. Initial Data Points:
-

Points	(1, 2)	(2, 2)	(2, 3)	(8, 7)	(8, 8)	(25, 80)	(7, 8)	(7.5, 8.5)
(1, 2)	0	1	1.41	8.60	9.21	78.90	7.07	7.91
(2, 2)	1	0	1	7.81	8.49	78.10	6.40	7.21
(2, 3)	1.41	1	0	7.07	7.81	77.36	5.66	6.52
(8, 7)	8.60	7.81	7.07	0	1	73.41	1	1.12
(8, 8)	9.21	8.49	7.81	1	0	72.80	1.41	0.71
(25, 80)	78.90	78.10	77.36	73.41	72.80	0	72.76	73.14
(7, 8)	7.07	6.40	5.66	1	1.41	72.76	0	0.71
(7.5, 8.5)	7.91	7.21	6.52	1.12	0.71	73.14	0.71	0

- Points = (1, 2), (2, 2), (2, 3), (8, 7), (8, 8), (25, 80), (7, 8), (7.5, 8.5)
-
- 2. Calculate Distances:

-
- 3. Classify Points:
-
- - Point (1, 2):
- - Neighbors: (2, 2), (2, 3)
- - MinPts: 2 (less than 3)
- - Type: Border point
-
- - Point (2, 2):
- - Neighbors: (1, 2), (2, 3)
- - MinPts: 2 (less than 3)
- - Type: Border point
-
- - Point (2, 3):
- - Neighbors: (1, 2), (2, 2)
- - MinPts: 2 (less than 3)
- - Type: Border point
-
- -

Point (8, 7):
- Neighbors: (8, 8), (7, 8), (7.5, 8.5)
- MinPts: 4
- Type: Core point

- Point (8, 8):
- Neighbors: (8, 7), (7, 8), (7.5, 8.5)
- MinPts: 4
- Type: Core point

- Point (25, 80):
- Neighbors: None within \(\epsilon\)
- MinPts: 0
- Type: Noise point

- Point (7, 8):
- Neighbors: (8, 7), (8, 8), (7.5, 8.5)
- MinPts: 4
- Type: Core point

- Point (7.5, 8.5):
- Neighbors: (8, 7), (8, 8), (7, 8)
- MinPts: 4

- Type: Core point

-
- 4. Form Clusters:
-
- - Cluster 1: (8, 7), (8, 8), (7, 8), (7.5, 8.5)
- - All points within epsilon of each other and form a dense region.
- - Border points: (1, 2), (2, 2), (2, 3)
-
- - Noise: (25, 80)
- - This point is not within epsilon of any core point and does not meet MinPts.
-

See Lesson 10.pynb

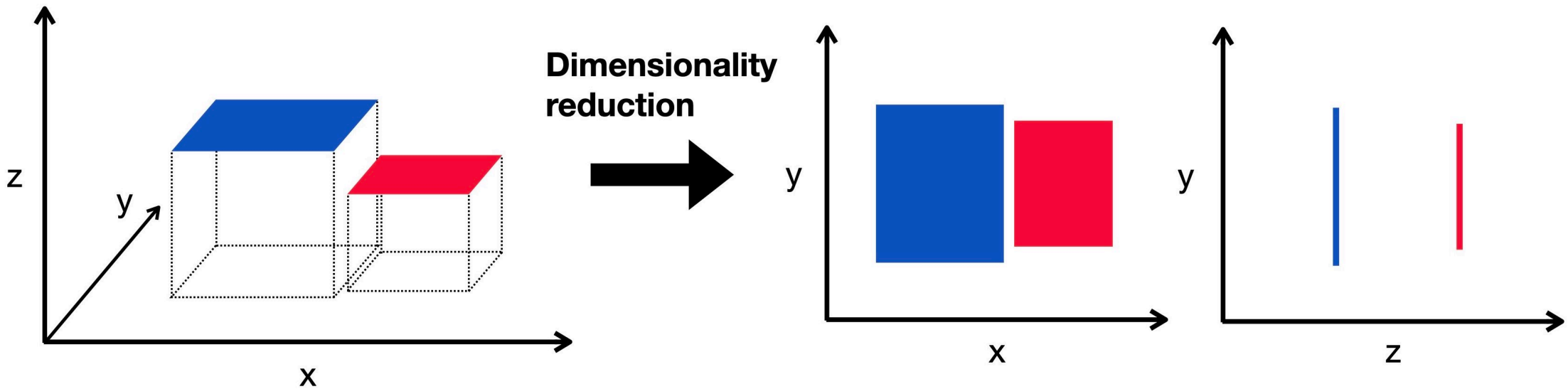
- Code Cell
- `10.3 DBSCAN`

Dimensionality Reduction

- Dimensionality reduction is a fundamental preprocessing step in machine learning that involves reducing the number of input variables in a dataset. It helps in simplifying models, accelerating algorithms, combating the "curse of dimensionality," and enhancing data visualization.

- **Why Dimensionality Reduction?**:
  - **Curse of Dimensionality**: As the number of features or dimensions grows, the amount of data needed to generalize accurately grows exponentially.
  - **Noise Reduction**: Eliminates redundant and irrelevant features, improving model performance.
  - **Improved Visualization**: Reduces dimensions to 2D or 3D for visual analysis, revealing patterns and relationships not visible in high-dimensional space.
  - **Computational Efficiency**: Reduces the computational cost of processing, storing, and analyzing high-dimensional datasets.
- **Key Techniques**:
  - **Feature Selection**: Selecting a subset of the most relevant features to use in model construction. Techniques include filter methods, wrapper methods, and embedded methods.
  - **Feature Extraction**: Transforming the data into a lower-dimensional space where the transformed features are a combination of the original ones. Techniques include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-Distributed Stochastic Neighbor Embedding (t-SNE).

- **Principal Component Analysis (PCA)**:
  - A statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
  - The first principal component has the largest possible variance, and each succeeding component has the highest variance possible under the constraint that it is orthogonal to the preceding components.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)**:
  - A non-linear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions.
  - Particularly effective at revealing patterns in the data by grouping similar data points together and separating dissimilar ones.
- **Challenges**:
  - **Choosing the Right Technique**: The effectiveness of dimensionality reduction depends heavily on the technique chosen and its appropriateness for the dataset and task at hand.
  - **Loss of Information**: While reducing dimensions, there's a trade-off between simplicity and retaining the variance (information) of the original dataset.
  - **Interpretability**: Some techniques, especially non-linear ones like t-SNE, can make the resulting dimensions hard to interpret in terms of the original features.
- **Applications**:
  - **Data Visualization**: Simplifying complex datasets to 2D or 3D for graphical representation.
  - **Noise Reduction**: Cleaning data by removing irrelevant features.
  - **Feature Engineering**: Creating more informative and non-redundant datasets for predictive models.


z
y
x
Dimensionality reduction
y
x
y
z

Example: Dimensionality Reduction with PCA

- Step 1: Data Preparation
-
- Consider a dataset with three features (dimensions):
- X_1, X_2, X_3
-
- Here's a sample dataset with 5 data points:

X_1	X_2	X_3
2.5	2.4	1.5
0.5	0.7	0.2
2.2	2.9	2.1
1.9	2.2	2.0
3.1	3.0	2.8

- Step 2: Standardize the Data
-
- Compute the mean and standard deviation of each feature, then standardize the data.

$$\text{Mean}(X_1) = 2.04, \quad \text{Std}(X_1) = 0.87$$
$$\text{Mean}(X_2) = 2.24, \quad \text{Std}(X_2) = 0.83$$
$$\text{Mean}(X_3) = 1.72, \quad \text{Std}(X_3) = 0.96(1)$$

- Standardized data (Z):

Z_1	Z_2	Z_3
0.53	0.19	−0.23
−1.77	−1.86	−1.57
0.18	0.79	0.40
−0.17	−0.04	0.29
1.23	0.92	1.12

- Step 3: Covariance Matrix
-
- Calculate the covariance matrix of the standardized data. $\Sigma = \frac{1}{n-1} Z^T Z$
- Covariance matrix:
-

$$\Sigma = \begin{pmatrix} 1.00 & 0.95 & 0.87 \\ 0.95 & 1.00 & 0.91 \\ 0.87 & 0.91 & 1.00 \end{pmatrix}$$

- Step 4: Eigenvalues and Eigenvector
- - Eigenvalues (λ) indicate the amount of variance captured by each principal component.
- - Eigenvectors (v) represent the direction of these principal components.
-
- 3. Calculation:
- - Solve the characteristic equation $|\Sigma - \lambda I| = 0$ to find eigenvalues (λ)
- - For each eigenvalue, solve. $(\Sigma - \lambda I)\mathbf{v} = 0$ to find the corresponding eigenvector v.
-
- 4. Interpretation:
- - Eigenvalues tell us how much variance each principal component explains.
- - Eigenvectors provide the directions of the new feature space.
-

- - An identity matrix I is a square matrix with ones on the diagonal and zeros elsewhere.
- - For an m x m matrix, the identity matrix I is:

$$I = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

- For a 3x3 covariance matrix Sigma:

$$\Sigma = \begin{pmatrix} 1.00 & 0.95 & 0.87 \\ 0.95 & 1.00 & 0.91 \\ 0.87 & 0.91 & 1.00 \end{pmatrix}$$

- The identity matrix I would be:

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- - Given the covariance matrix: $\Sigma = \begin{pmatrix} 1.00 & 0.95 & 0.87 \\ 0.95 & 1.00 & 0.91 \\ 0.87 & 0.91 & 1.00 \end{pmatrix}$
- We find the eigenvalues and eigenvectors:
- - Eigenvalues: $\lambda_1 = 2.69, \lambda_2 = 0.29, \lambda_3 = 0.02$
- - Eigenvectors:

$$\mathbf{v}_1 = \begin{pmatrix} 0.58 \\ 0.57 \\ 0.58 \end{pmatrix}, \quad \mathbf{v}_2 = \begin{pmatrix} -0.71 \\ 0.01 \\ 0.71 \end{pmatrix}, \quad \mathbf{v}_3 = \begin{pmatrix} 0.39 \\ -0.82 \\ 0.42 \end{pmatrix}$$

- Step 5: Project Data onto New Dimensions
-
- Objective: Transform the original data into a new subspace defined by the top k principal components.
-
- 1. Select Principal Components:
- - Choose the top k eigenvectors corresponding to the largest eigenvalues.
- - These eigenvectors form the projection matrix W.
-
- 2. Form Projection Matrix W:
- - W is composed of the selected eigenvectors as columns.
- - For k = 2, W will have 2 columns.

- - For our eigenvalues $\lambda_1 = 2.69$ and $\lambda_2 = 0.29$, we select v_1 and v_2.

- Projection matrix W:

$$W = \begin{pmatrix} 0.58 & -0.71 \\ 0.57 & 0.01 \\ 0.58 & 0.71 \end{pmatrix}$$

- Project Data:
- - Multiply the standardized data matrix Z by the projection matrix W:
-
- Y = Z . W
-
- - Y is the transformed dataset in the new K-dimensional subspace.

- If Z is our standardized data matrix, the projected data Y is:

$$Y = \begin{matrix} 0.67 & -0.95 \\ -2.44 & 0.28 \\ 0.74 & -0.56 \\ 0.09 & -0.52 \\ 1.03 & 1.74 \end{matrix}$$

- This transformation effectively reduces the dimensionality of the data while preserving the most significant variance.

PCA and t-SNE

- **Principal Component Analysis (PCA)**:
  - **Essence**: PCA is a linear dimensionality reduction technique that transforms a set of possibly correlated features into a smaller set of linearly uncorrelated variables known as principal components.
  - **Mathematical Foundation**: It utilizes eigenvalue decomposition of the data covariance matrix or singular value decomposition (SVD) of the data matrix to find the principal components.
  - **Key Properties**:
    - **Variance Capturing**: The first principal component captures the most variance in the data, with each subsequent component capturing progressively less.
    - **Orthogonality**: Each principal component is orthogonal (i.e., independent) to the others, ensuring that the reduced dimensions are uncorrelated.
    - **Application Scenarios**: Ideal for pre-processing before applying linear models, visualizing genetic distances, or simplifying datasets with many variables.

- **t-Distributed Stochastic Neighbor Embedding (t-SNE)**:
  - **Essence**: t-SNE is a non-linear technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets.
  - **Mechanism**: It converts affinities of data points to probabilities. The similarity of datapoints is represented by joint probabilities, and the goal is to minimize the divergence between these probabilities in the high-dimensional and low-dimensional space.
  - **Key Features**:
    - **Non-linearity**: Capable of capturing complex polynomial relationships by mapping high-dimensional data to a lower-dimensional space (usually 2D or 3D).
    - **Clusters Visualization**: Excellently reveals the clustering of data points and the relationships between clusters in a visually intuitive manner.
    - **Sensitivity to Parameters**: The choice of perplexity (a key t-SNE parameter) greatly influences the resulting visualization, requiring experimentation for optimal settings.
- **Comparative Insights**:
  - **Linearity vs. Non-Linearity**: PCA is linear, making it effective for datasets where linear relationships dominate. t-SNE is non-linear and can uncover complex patterns that PCA might miss.
  - **Scalability**: PCA scales better to large datasets and can be computed more efficiently than t-SNE, which can become computationally intensive, especially on very large datasets.
  - **Interpretability**: PCA components have a direct correlation with the original features, making them more interpretable. t-SNE dimensions do not correspond to any linear combination of features, prioritizing data structure and relationships over interpretability.
  - **Use Cases**: PCA is often used for pre-processing and noise reduction before applying other learning algorithms, while t-SNE is predominantly used for exploratory data analysis and visualizing high-dimensional data.

See Lesson 10.pynb

- Code Cell

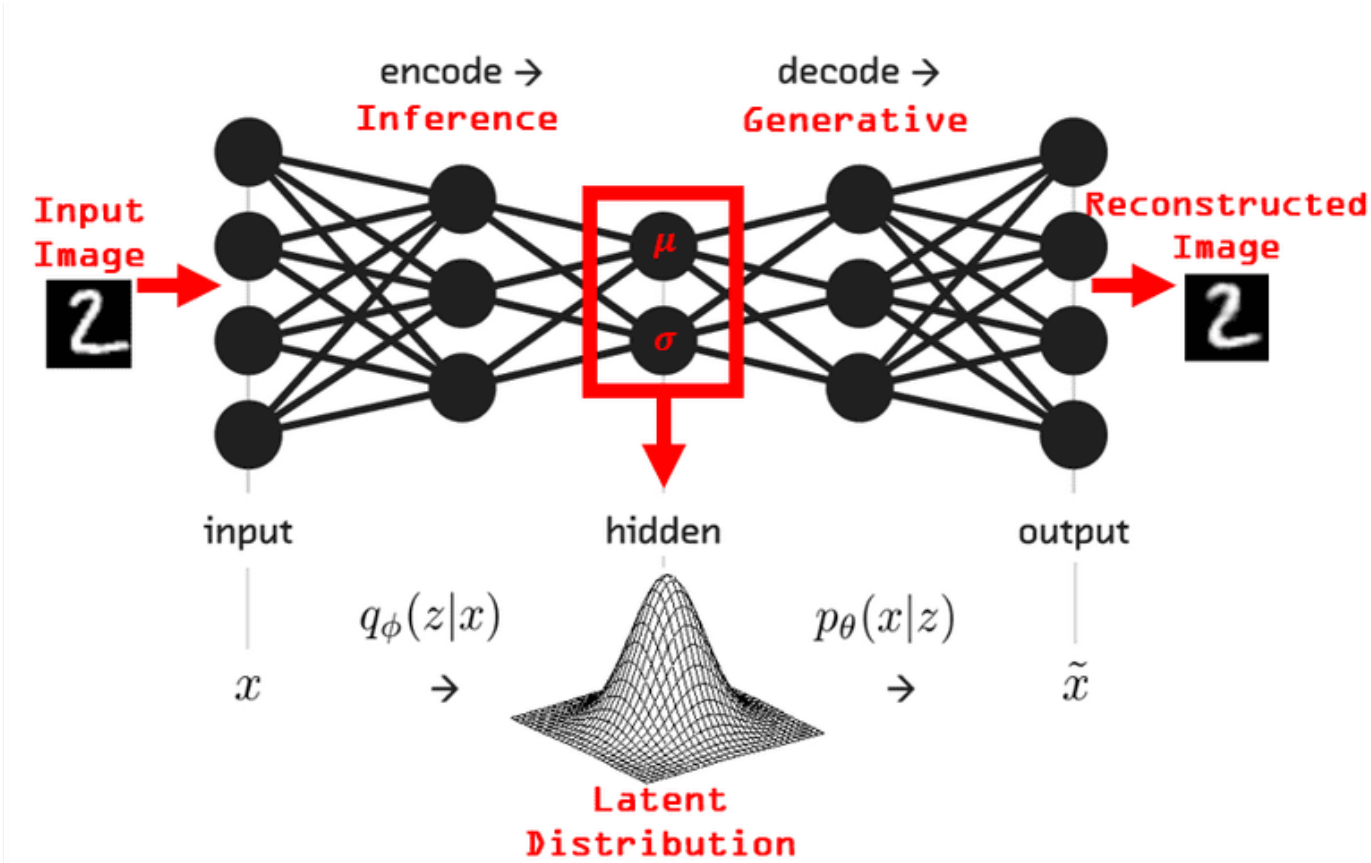
- `10.4 Dimensionality Reduction`

Autoencoders

- Autoencoders are a type of neural network used for unsupervised learning. They aim to learn a compressed, distributed representation (encoding) for a dataset, typically for the purpose of dimensionality reduction or feature learning.

- **Architecture**:
  - **Encoder**: The first part of the network compresses the input into a latent-space representation. It learns to preserve as much of the significant information as possible while reducing dimensionality.
  - **Decoder**: The second part reconstructs the input data from the latent space representation. The goal is to minimize the difference between the input and its reconstruction, typically measured by a loss function like mean squared error.
  - **Latent Space**: A compressed knowledge representation of the original input data. It is the output of the encoder and the input to the decoder.
- **Working Mechanism**:
  - By training the network to reproduce its input, autoencoders learn to identify patterns and important features in the data. The process forces the network to prioritize which aspects of the input should be retained or discarded, resulting in a powerful tool for feature extraction and data compression.

- **Types of Autoencoders**:
  - **Sparse Autoencoders**: Introduce sparsity in the encoded representations to improve feature selection.
  - **Denoising Autoencoders**: Train the network to ignore "noise" in the input data, enhancing its ability to capture the underlying structure of the data.
  - **Variational Autoencoders (VAEs)**: Use probabilistic layers to model the latent space, enabling them to generate new data points similar to the input data.
- **Applications**:
  - **Dimensionality Reduction**: Similar to PCA but more powerful due to its non-linear nature.
  - **Feature Learning**: Can automatically learn features for other tasks like classification.
  - **Data Denoising**: Effective in cleaning noisy data by learning to ignore the noise component.
  - **Data Generation**: VAEs, in particular, can generate new data that's similar to the training data, useful in domains like drug discovery and content creation.
- **Advantages**:
  - **Flexibility**: Can model complex, non-linear relationships in the data.
  - **Efficiency**: Reduces data dimensionality while retaining meaningful structure, aiding in faster computation and less storage space.
  - **Generative Capabilities**: Some autoencoders can generate new data points, offering capabilities beyond traditional dimensionality reduction techniques.
- **Challenges**:
  - **Architecture Design**: Choosing the right architecture and hyperparameters can be challenging and requires experimentation.
  - **Overfitting**: Without proper regularization, autoencoders can memorize the input data rather than learning a generalized representation.
  - **Interpretability**: The latent space representation is not as interpretable as linear methods like PCA.


encode →
Inference
decode →
Generative
Input
Image
Reconstructed
Image
μ
σ
input
hidden
output
$q_\phi(z|x)$
$p_\theta(x|z)$
x
$\tilde{x}$
Latent
Distribution

See Lesson 10.pynb

- Code Cell
- `10.5 Autoencoders`