UNSUPERVISED LEARNING

UNIT-2

GAUSSIAN MIXTURE MODEL

- Gaussian Mixture Model is a probabilistic model that assumes all data points are generated from a mixture of several Gaussian distributions with unknown parameters. Unlike hard clustering methods like K-Means that assigns each data point to one cluster based on the closest centroid which often doesn't align with the complexity of real-world data. GMMs perform soft clustering meaning each data point belongs to multiple clusters with certain probabilities. Each Gaussian in the mixture is defined by:

- Mean (μ): The center of the distribution.
- Covariance (Σ): Describes the spread and orientation.
- Mixing coefficient (π): Represents the proportion of each Gaussian in the mixture.

HOW GAUSSIAN MIXTURE MODELS WORK?

- Now that we understand the key differences between K-Means and GMM let's go deeper into how GMM actually works. Here Each cluster is represented by a Gaussian distribution and the data points are assigned probabilities of belonging to different clusters based on their distance from each Gaussian.The Gaussian mixture model assigns a probability to each data point *Xn* of belonging to a cluster. The probability of data point coming from Gaussian cluster k is expressed as

$$P(z_n = k \mid x_n) = \frac{\pi_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma_k)}$$

where:

- z_n=k is a latent variable indicating which Gaussian the point belongs to.
- π_k is the mixing probability of the k-th Gaussian.
- $\mathcal{N}(x_n \mid \mu_k, \Sigma_k)$is the Gaussian distribution with mean μ_k and covariance Σk

Next we need to calculate the overall likelihood of observing a data point x_n under all Gaussians. This is achieved by summing over all possible clusters (Gaussians) for each point:

$$P(x_n) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$$

where:

- $P(x_n)$ is the overall likelihood of observing the data point x_n
- The sum accounts for all possible Gaussians k.

THE EXPECTATION-MAXIMIZATION (EM) ALGORITHM

- To fit a Gaussian Mixture Model to the data we use the Expectation-Maximization (EM) algorithm which is an iterative method that optimize the parameters of the Gaussian distributions like mean, covariance and mixing coefficients. It works in two main steps:

  - Expectation Step (E-step): In this step the algorithm calculates the probability that each data point belongs to each cluster based on the current parameter estimates (mean, covariance, mixing coefficients).

  - Maximization Step (M-step): After estimating the probabilities the algorithm updates the parameters (mean, covariance and mixing coefficients) to better fit the data.

- These two steps are repeated until the model converges meaning the parameters no longer change significantly between iterations. Here's a simple breakdown of the GMM process:

1. Initialization: Start with initial guesses for the means, covariances and mixing coefficients of each Gaussian distribution.

2. E-step: For each data point, calculate the probability of it belonging to each Gaussian distribution (cluster).

3. M-step: Update the parameters (means, covariances, mixing coefficients) using the probabilities calculated in the E-step.

4. Repeat: Continue alternating between the E-step and M-step until the log-likelihood of the data (a measure of how well the model fits the data) converges.

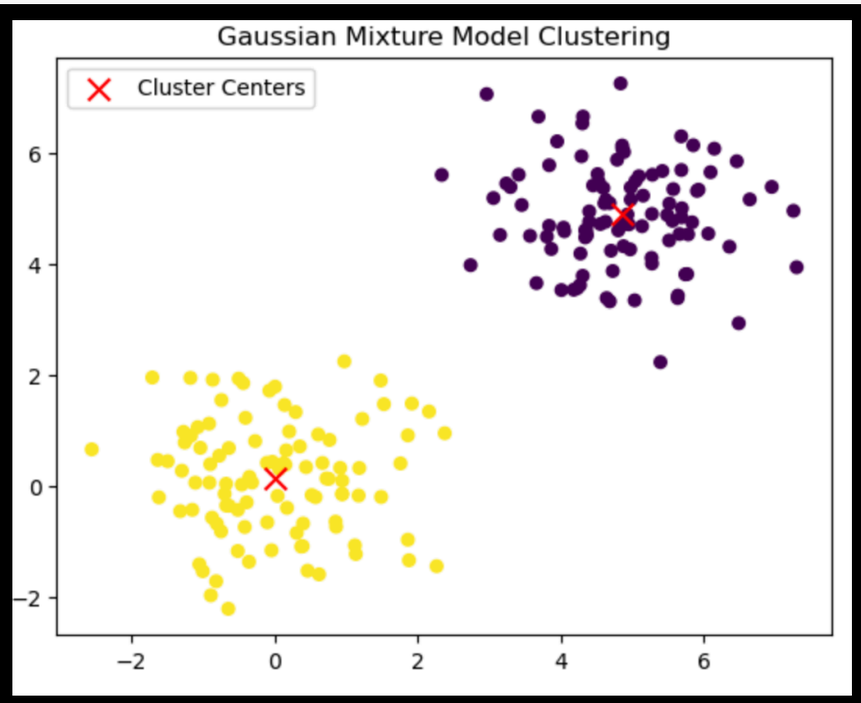
Formula:

$$L(\mu_k, \Sigma_k, \pi_k) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$$

The E-step computes the probabilities that each data point belongs to each Gaussian while the M-step updates the parameters μk, Σk and πk based on these probabilities.

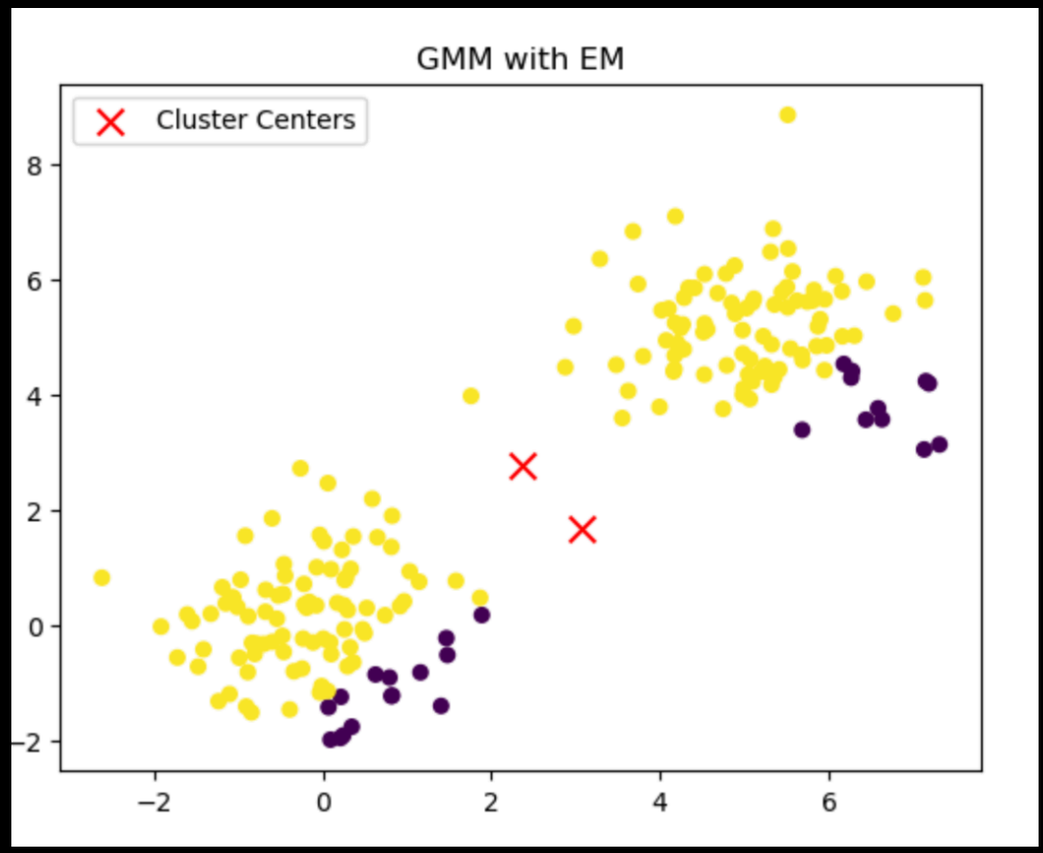
example using **scikit-learn**:

```python
from sklearn.mixture import GaussianMixture
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data (two Gaussian clusters)
np.random.seed(0)
X1 = np.random.normal(0, 1, (100, 2))
X2 = np.random.normal(5, 1, (100, 2))
X = np.vstack((X1, X2))

# Fit Gaussian Mixture Model
gmm = GaussianMixture(n_components=2, random_state=0)
gmm.fit(X)

# Predict cluster labels
labels = gmm.predict(X)

# Plot results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=30)
plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], c='red', marker='x', s=100, label="Cluster Centers")
plt.title("Gaussian Mixture Model Clustering")
plt.legend()
plt.show()
```



```python
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt

# Step 1: Generate synthetic data
np.random.seed(42)
X1 = np.random.normal(0, 1, (100, 2))
X2 = np.random.normal(5, 1, (100, 2))
X = np.vstack((X1, X2))

# Step 2: Initialize parameters
k = 2  # number of clusters
n, d = X.shape
weights = np.ones(k) / k
means = X[np.random.choice(n, k, replace=False)]
covariances = [np.cov(X.T) for _ in range(k)]

print("Initial Means:\n", means)
print("Initial Weights:", weights)
print("Initial Covariances:\n", covariances, "\n")

# Step 3: EM algorithm
for it in range(1, 6):  # Run only 5 iterations for clarity
    print(f"--- Iteration {it} ---")

    # E-step: responsibilities
    resp = np.zeros((n, k))
    for j in range(k):
        resp[:, j] = weights[j] * multivariate_normal.pdf(X, means[j], covariances[j])
    resp = resp / resp.sum(axis=1, keepdims=True)

    # Show sample responsibilities
    print("Responsibilities (first 5 points):\n", resp[:5])

    # M-step: update parameters
    Nk = resp.sum(axis=0)
    for j in range(k):
        means[j] = (resp[:, j][:, np.newaxis] * X).sum(axis=0) / Nk[j]
        covariances[j] = np.cov(X.T, aweights=resp[:, j], bias=True)
        weights[j] = Nk[j] / n

# Step 4: Final cluster assignments
labels = resp.argmax(axis=1)

# Step 5: Plot results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis", s=30)
plt.scatter(means[:, 0], means[:, 1], c="red", marker="x", s=100, label="Cluster Centers")
plt.title("GMM with EM")
plt.legend()
plt.show()
```

```
Initial Means:
 [[ 1.40279431 -1.40185106]
 [-0.19236096  0.30154734]]
Initial Weights: [0.5 0.5]
Initial Covariances:
 [array([[7.81138329, 6.57351555],
       [6.57351555, 7.23628434]]), array([[7.81138329, 6.57351555],
       [6.57351555, 7.23628434]])]

--- Iteration 1 ---
Responsibilities (first 5 points):
 [[0.28468656 0.71531344]
 [0.02577876 0.97422124]
 [0.11938051 0.88061949]
 [0.33383662 0.66616338]
 [0.02213673 0.97786327]]
Updated Means:
 [[3.24083418 1.73621077]
 [2.30247092 2.76151542]]
Updated Weights: [0.21726268 0.78273732]

--- Iteration 2 ---
Responsibilities (first 5 points):
 [[0.29028678 0.70971322]
 [0.0308779  0.9691221 ]
 [0.13874972 0.86125028]
 [0.32692707 0.67307293]
 [0.02970503 0.97029497]]
Updated Means:
 [[3.1950694  1.72257351]
 [2.31868896 2.76113496]]
Updated Weights: [0.21412307 0.78587693]

--- Iteration 3 ---
Responsibilities (first 5 points):
 [[0.29036004 0.70963996]
 [0.03594216 0.96405784]
 [0.14894257 0.85105743]
 [0.32167143 0.67832857]
 [0.0362311  0.9637689 ]]
Updated Means:
 [[3.14999182 1.70179387]
 [2.33437082 2.76237571]]
Updated Weights: [0.21084721 0.78915279]

--- Iteration 4 ---
Responsibilities (first 5 points):
 [[0.28844704 0.71155296]
 [0.0402948  0.9597052 ]
 [0.1554334  0.8445666 ]
 [0.31580251 0.68419749]
 [0.04184568 0.95815432]]
Updated Means:
 [[3.10852618 1.680231  ]
 [2.34849965 2.76378862]]
Updated Weights: [0.20768036 0.79231964]

--- Iteration 5 ---
Responsibilities (first 5 points):
 [[0.28584968 0.71415032]
 [0.04395343 0.95604657]
 [0.15994789 0.84005211]
 [0.31000498 0.68999502]
 [0.04667851 0.95332149]]
Updated Means:
 [[3.0712502  1.65968349]
 [2.36097942 2.76495875]]
Updated Weights: [0.20465831 0.79534169]
```



IMPLEMENTATION OF GMM IN PYTHON

- So far, we have discussed about the working of GMM. Let's take a simple example using the Iris dataset and fit a Gaussian Mixture Model with 3 clusters since we know there are 3 species of Iris.

- Dataset: We use only two features: sepal length and sepal width.
- Fitting the Model: We fit the data as a mixture of 3 Gaussians.
- Result: The model assigns each data point to a cluster based on the probabilities. After convergence GMM model will have updated the parameters to best fit the data.

ADVANTAGES OF GMM

- Flexible Cluster Shapes: Unlike K-Means which assumes spherical clusters it can model clusters with arbitrary shapes.

- Soft Assignment: GMM assign a probability for each data point to belong to each cluster while K-Means assigns each point to exactly one cluster.

- Handles Overlapping Data: GMM performs well when clusters overlap or have varying densities. Since it uses probability distributions, it can assign a point to multiple clusters with different probabilities

LIMITATIONS OF GMM

- Computational Complexity: GMM can take a lot of time and computer power especially when working with large datasets. because it uses a step-by-step method to keep improving its guesses about the data.
- Choosing the Number of Clusters: Just like other clustering methods GMM needs you to decide how many groups you want to split the data into before starting. But there are more methods like the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) can help in selecting the optimal number of clusters based on the data.

WHY GMM IS NEEDED ?

1. Cluster Shape Assumption

- **K-Means** assumes clusters are **spherical** and of **equal size** (only uses distance to nearest center).

- GMM allows clusters to be elliptical, stretched, rotated, or unequal in size, because it models each cluster with a Gaussian distribution (mean + covariance).

- Example: If one cluster is elongated (like a banana shape), K-Means fails, but GMM works.

2. Hard vs. Soft Clustering

- **K-Means** → assigns each point to exactly one cluster (**hard assignment**).
- **GMM** → gives a **probability** of belonging to each cluster (**soft assignment**).
    - Example: A point may be 70% cluster A, 30% cluster B — useful for uncertain or overlapping data.

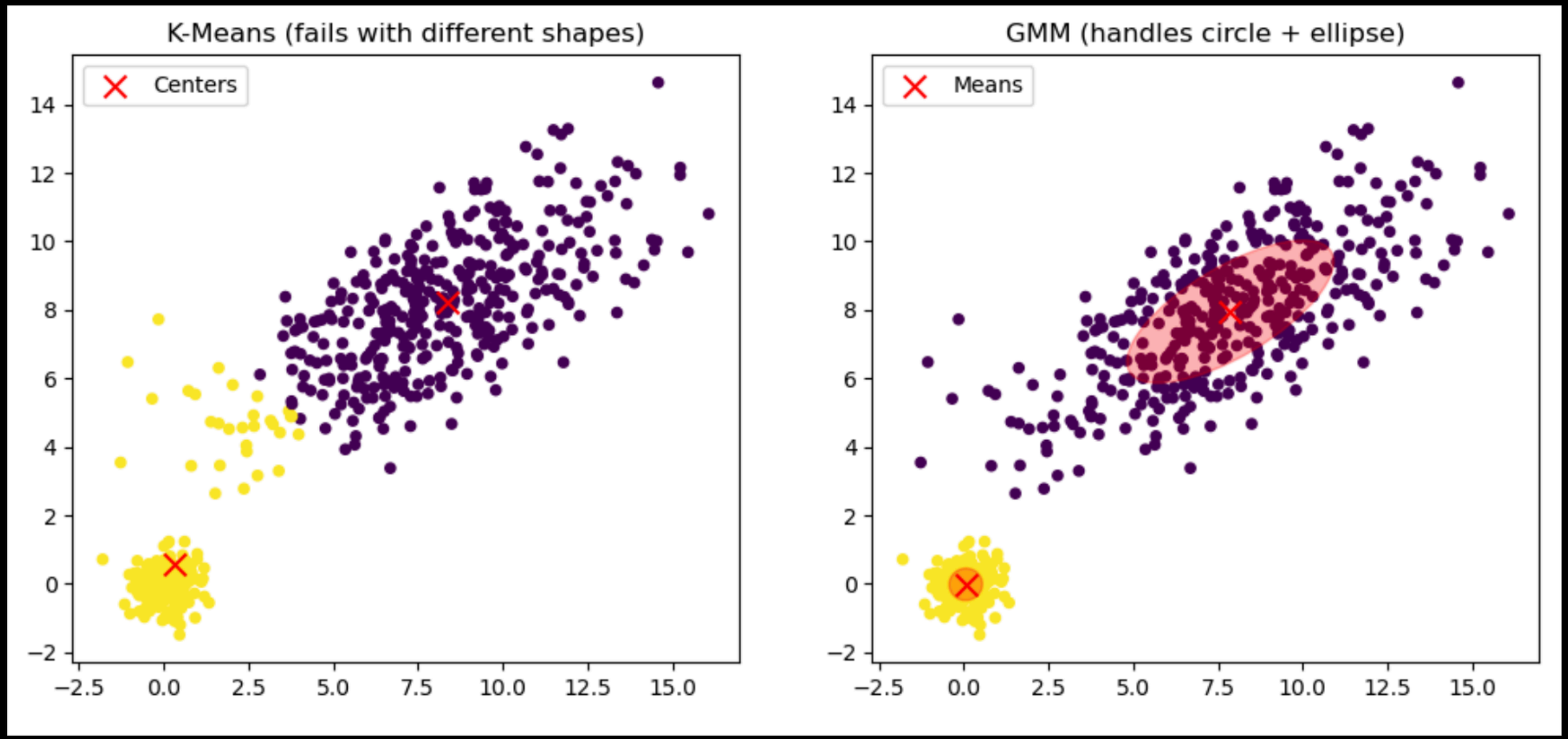
3. Flexibility with Covariance

- **K-Means** only uses cluster **centroids**.
- GMM uses both mean (centre) and covariance (shape + orientation) of clusters.
    - Makes GMM more powerful in real-world cases (finance, biology, speech recognition, etc.).

4. Likelihood-Based

- **K-Means** minimises distance (sum of squared errors).
- **GMM** maximises **likelihood** that the data came from a mixture of Gaussians.
    - This is statistically stronger and more interpretable.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from matplotlib.patches import Ellipse

# Helper function to draw ellipses for GMM
def draw_ellipse(position, covariance, ax, **kwargs):
    if covariance.shape == (2, 2):
        U, s, _ = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    ax.add_patch(Ellipse(position, width, height, angle, **kwargs))

# Generate different shaped clusters
np.random.seed(10)
circle = 0.5 * np.random.randn(200, 2) + [0, 0]       # small circular cluster
ellipse = np.dot(np.random.randn(400, 2), [[3, 1], [1, 2]]) + [8, 8]  # elongated ellipse
X = np.vstack([circle, ellipse])

# K-Means
kmeans = KMeans(n_clusters=2, random_state=0)
labels_kmeans = kmeans.fit_predict(X)

# GMM
gmm = GaussianMixture(n_components=2, covariance_type='full', random_state=0)
gmm.fit(X)
labels_gmm = gmm.predict(X)

# Plot
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# --- K-Means ---
axes[0].scatter(X[:, 0], X[:, 1], c=labels_kmeans, cmap="viridis", s=20)
axes[0].scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
                c="red", marker="x", s=100, label="Centers")
axes[0].set_title("K-Means (fails with different shapes)")
axes[0].legend()

# --- GMM ---
axes[1].scatter(X[:, 0], X[:, 1], c=labels_gmm, cmap="viridis", s=20)
axes[1].scatter(gmm.means_[:, 0], gmm.means_[:, 1],
                c="red", marker="x", s=100, label="Means")
for pos, covar in zip(gmm.means_, gmm.covariances_):
    draw_ellipse(pos, covar, axes[1], alpha=0.3, color="red")
axes[1].set_title("GMM (handles circle + ellipse)")
axes[1].legend()

plt.show()
```


K-Means (fails with different shapes)
Centers
GMM (handles circle + ellipse)
Means
14
12
10
8
6
4
2
0
−2
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0

VARIATIONAL BAYESIAN INFERENCE FOR GAUSSIAN MIXTURE

- A Gaussian Mixture Model assumes the data to be segregated into clusters in such a way that each data point in a given cluster follows a particular Multi-variate Gaussian distribution and the Multi-Variate Gaussian distributions of each cluster is independent of one another.

- To cluster data in such a model, the posterior probability of a data-point belonging to a given cluster given the observed data needs to be calculated. An approximate method for this purpose is the Baye's method.

- But for large datasets, the calculation of marginal probabilities is very tedious. As there is only a need to find the most probable cluster for a given point, approximation methods can be used as they reduce the mechanical work.

- One of the best approximate methods is to use the Variational Bayesian Inference method. The method uses the concepts of KL Divergence and Mean-Field Approximation.

NEED ?

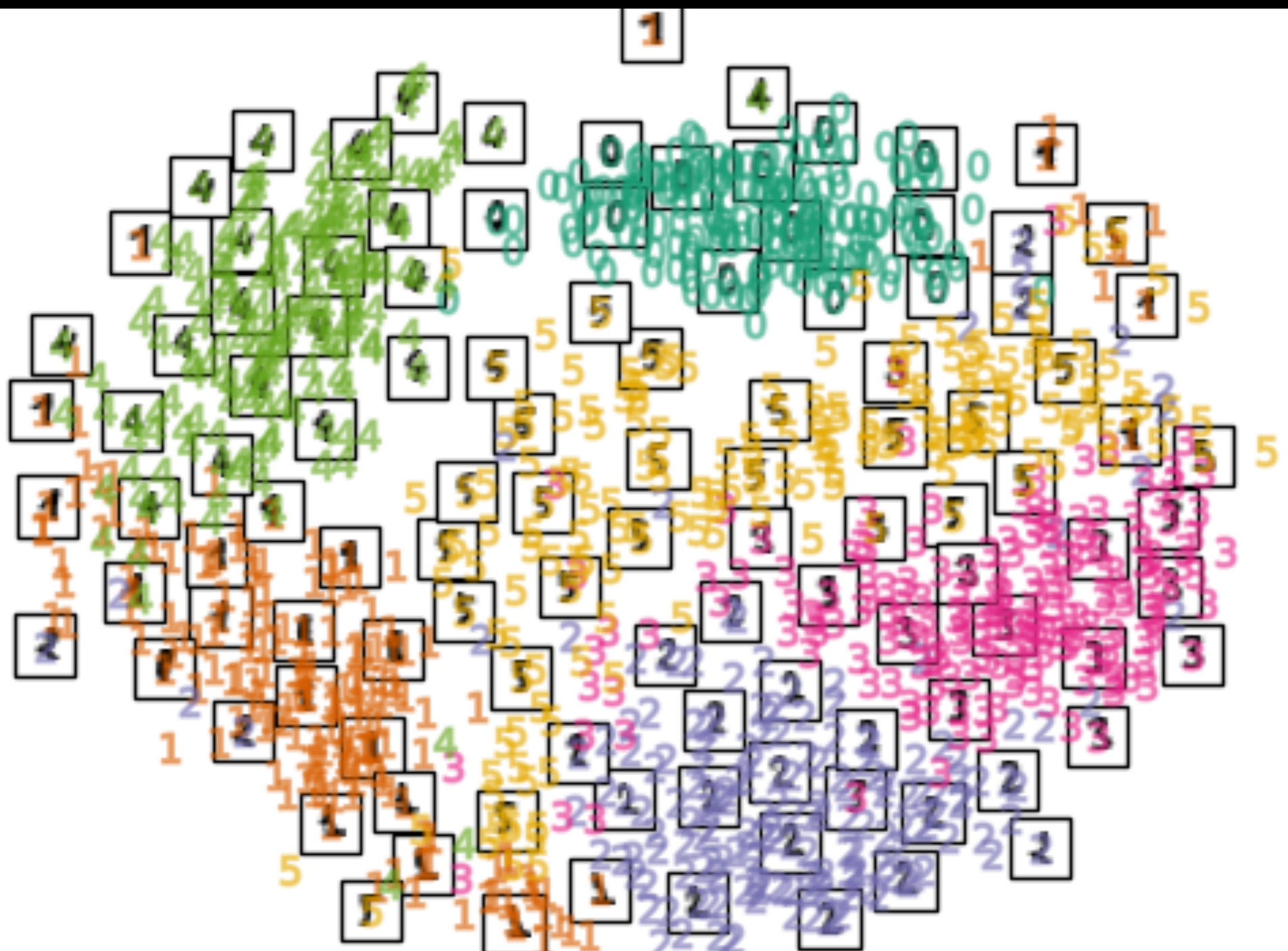
- The problem with standard GMMs
  - A classical Gaussian Mixture Model (GMM) is usually trained with Expectation–Maximization (EM).
  - In EM:
    - We estimate mixture weights, means, and covariances as *point estimates* (maximum likelihood or MAP).
    - Problems:
      1. **Number of components (K) must be fixed in advance** – EM doesn't tell you how many clusters are really needed.
      2. **Overfitting** – EM can fit spurious components to noise.
      3. **Singularities** – likelihood can blow up if a covariance matrix collapses.
      4. **No uncertainty** – EM gives only one set of parameters, not a distribution.

- What Variational Bayesian Inference does
  - Instead of point estimates, VBI gives a **posterior distribution** over:
    - mixture weights π
    - means μk
    - covariances Σk
  - The variational method approximates the intractable posterior by a factorized distribution and optimizes it.

- Advantages:
  - **Automatic complexity control**: You can start with a large number of components, but the variational posterior prunes unused ones (weights go to ~0). This is like "automatic model selection."
  - **Prevents overfitting**: Priors (Dirichlet, Normal-Inverse-Wishart) act as regularizers.
  - **Uncertainty quantification**: You don't just get "the mean is 2.1," but also "with variance 0.3" → more robust decisions.
  - **Stable training**: Less likely to collapse into singular solutions compared to EM.

UNIT-II

MANIFOLD LEARNING



"Manifold learning is an approach to non-linear dimensionality reduction. Algorithms for this task are based on the idea that the dimensionality of many data sets is only artificially high."

INTRODUCTION

- High-dimensional datasets can be very difficult to visualize. While data in two or three dimensions can be plotted to show the inherent structure of the data, equivalent high-dimensional plots are much less intuitive. To aid visualization of the structure of a dataset, the dimension must be reduced in some way.

- The simplest way to accomplish this dimensionality reduction is by taking a random projection of the data. Though this allows some degree of visualization of the data structure, the randomness of the choice leaves much to be desired. In a random projection, it is likely that the more interesting structure within the data will be lost.

- VTo address this concern, a number of supervised and unsupervised linear dimensionality reduction frameworks have been designed, such as Principal Component Analysis (PCA), Independent Component Analysis, Linear Discriminant Analysis, and others. These algorithms define specific rubrics to choose an "interesting" linear projection of the data.

- These methods can be powerful, but often miss important non-linear structure in the data.

- Manifold Learning can be thought of as an attempt to generalize linear frameworks like PCA to be sensitive to non-linear structure in data. Though supervised variants exist, the typical manifold learning problem is unsupervised: it learns the high-dimensional structure of the data from the data itself, without the use of predetermined classifications.

ISOMAP

- One of the earliest approaches to manifold learning is the Isomap algorithm, short for Isometric Mapping. Isomap can be viewed as an extension of Multi-dimensional Scaling (MDS) or Kernel PCA. Isomap seeks a lower-dimensional embedding which maintains geodesic distances between all points. Isomap can be performed with the object Isomap.

The Isomap algorithm comprises three stages:

1. **Nearest neighbor search.** Isomap uses `BallTree` for efficient neighbor search. The cost is approximately $O[D \log(k) N \log(N)]$, for k nearest neighbors of N points in D dimensions.
2. **Shortest-path graph search.** The most efficient known algorithms for this are *Dijkstra's Algorithm*, which is approximately $O[N^2(k + \log(N))]$, or the *Floyd-Warshall algorithm*, which is $O[N^3]$. The algorithm can be selected by the user with the `path_method` keyword of `Isomap`. If unspecified, the code attempts to choose the best algorithm for the input data.
3. **Partial eigenvalue decomposition.** The embedding is encoded in the eigenvectors corresponding to the d largest eigenvalues of the $N \times N$ isomap kernel. For a dense solver, the cost is approximately $O[dN^2]$. This cost can often be improved using the `ARPACK` solver. The eigensolver can be specified by the user with the `eigen_solver` keyword of `Isomap`. If unspecified, the code attempts to choose the best algorithm for the input data.

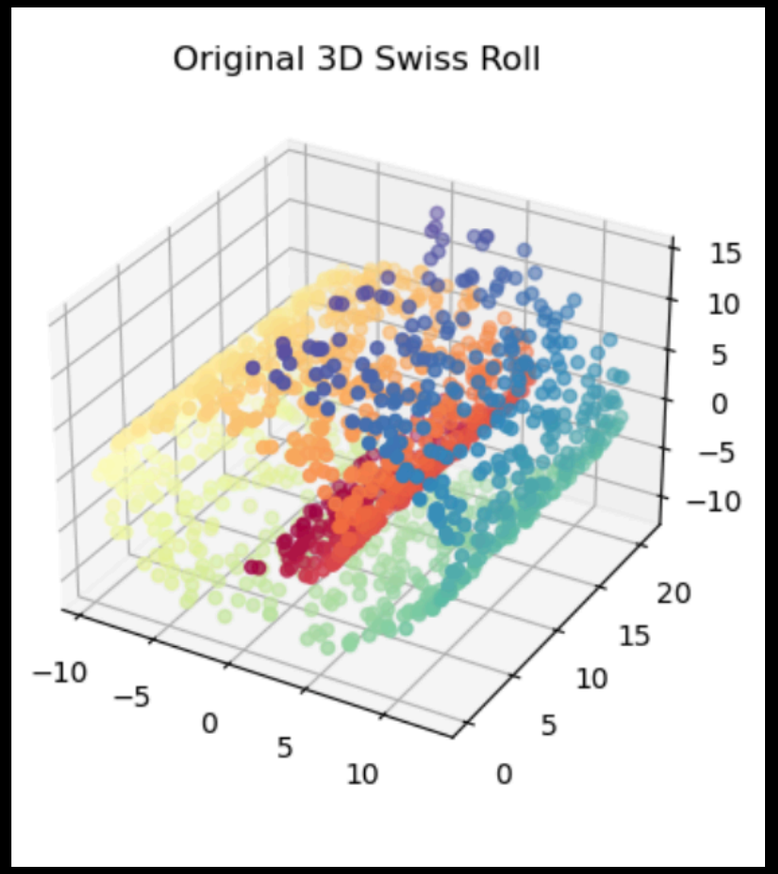
The overall complexity of Isomap is
$O[D \log(k) N \log(N)] + O[N^2(k + \log(N))] + O[dN^2]$.

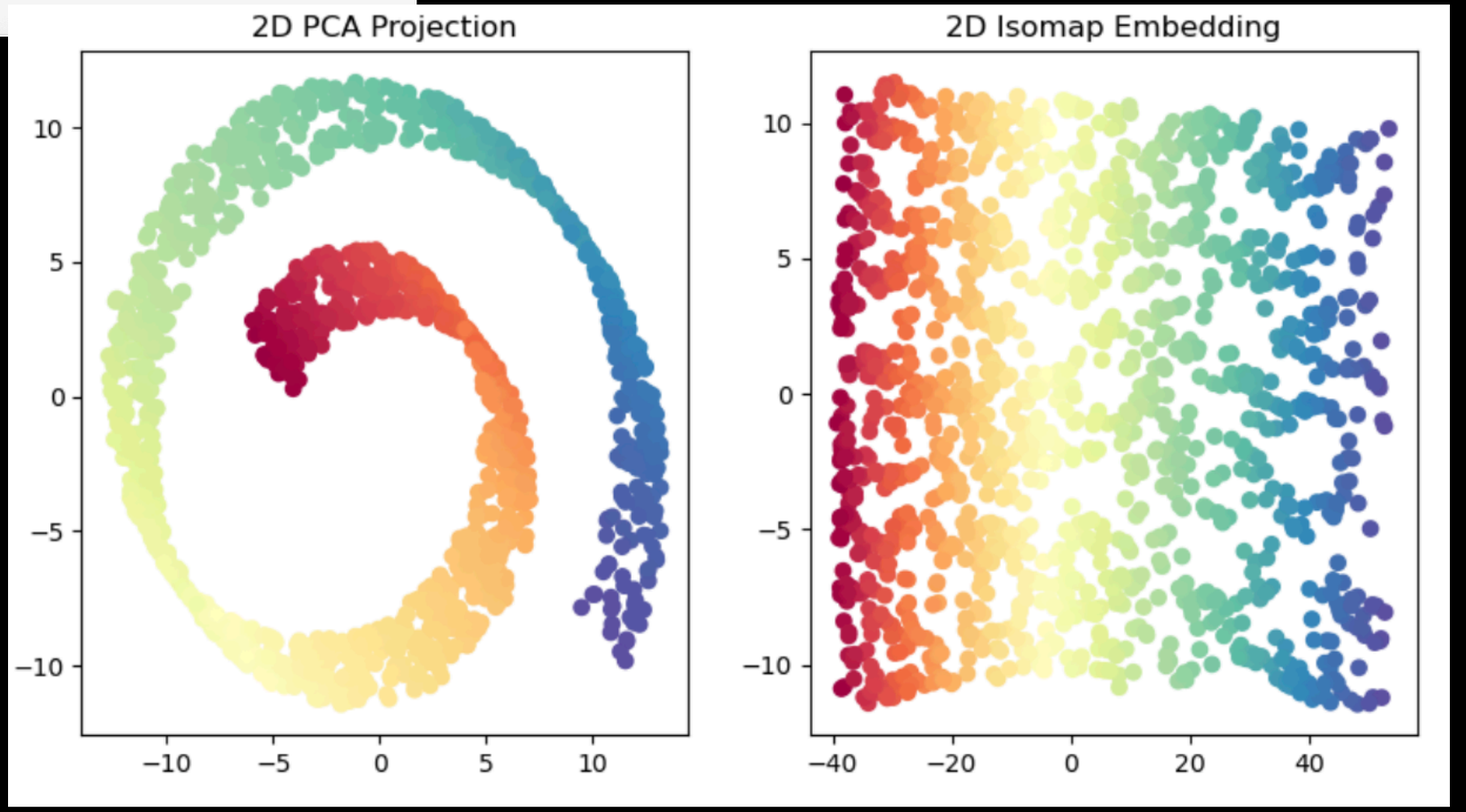
- N : number of training data points
- D : input dimension
- k : number of nearest neighbors
- d : output dimension

APPLICATIONS :

- **Visualization** of high-dimensional data (e.g., images, text embeddings).
- Preprocessing step before clustering/classification.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap

# Step 1: Generate Swiss Roll dataset
n_samples = 1500
X, color = make_swiss_roll(n_samples, noise=0.05)

# Step 2: PCA (3D -> 2D)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Step 3: Isomap (3D -> 2D)
isomap = Isomap(n_neighbors=10, n_components=2)
X_isomap = isomap.fit_transform(X)

# Step 4: Plot results
fig = plt.figure(figsize=(15,5))

# Original 3D Swiss Roll
ax = fig.add_subplot(131, projection='3d')
ax.scatter(X[:,0], X[:,1], X[:,2], c=color, cmap=plt.cm.Spectral)
ax.set_title("Original 3D Swiss Roll")

# PCA Projection
ax2 = fig.add_subplot(132)
ax2.scatter(X_pca[:,0], X_pca[:,1], c=color, cmap=plt.cm.Spectral)
ax2.set_title("2D PCA Projection")

# Isomap Embedding
ax3 = fig.add_subplot(133)
ax3.scatter(X_isomap[:,0], X_isomap[:,1], c=color, cmap=plt.cm.Spectral)
ax3.set_title("2D Isomap Embedding")

plt.show()
```





NEED ?

- We need **Isomap** because **linear techniques (like PCA) can't capture nonlinear relationships**. Isomap helps in **unfolding complex manifolds** and preserving true distances, making data analysis and visualization more accurate.

HOW DOES ISOMAP WORK?

- Calculate Pairwise Distances: First we find the Euclidean distances between all pairs of data points.
- Find Nearest Neighbors: For each point find the closest other points based on distance.
- Create a Neighborhood Graph: Connect each point to its nearest neighbors to form a graph.
- Calculate Geodesic Distances: Use algorithms like Floyd-Warshall to measure the shortest paths between points by following the graph connections.
- Perform Dimensional Reduction: Move points into a simpler space while keeping their distances as accurate as possible.

IMPLEMENTATION OF ISOMAP

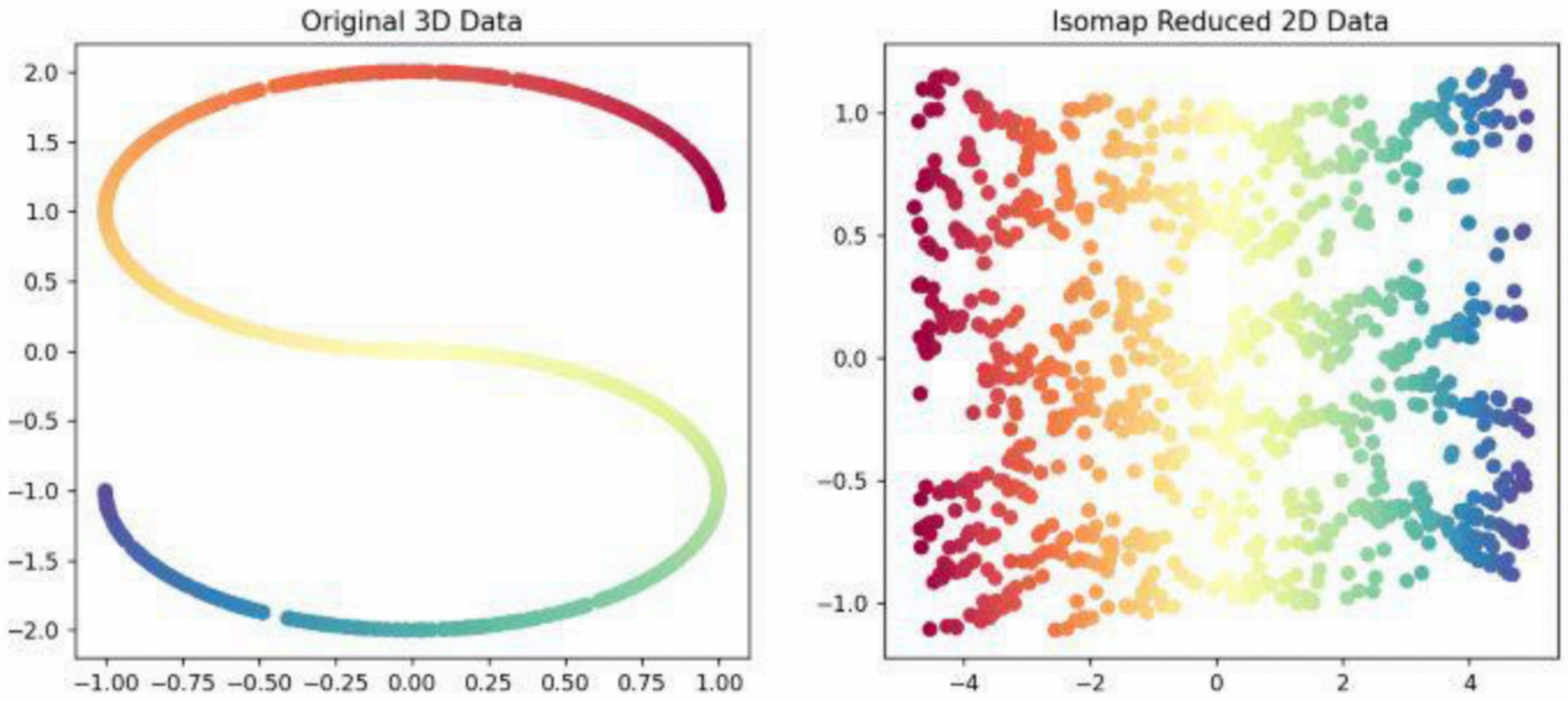
APPLYING ISOMAP TO S-CURVE DATA

This part generates a 3D S-curve dataset and applies Isomap to reduce it to 2D for visualization. It highlights how Isomap preserves the non-linear structure by flattening the curve while keeping the relationships between points intact.

- make_s_curve() creates a 3D curved dataset shaped like an "S".
- Isomap() reduces the data to 2D while keeping its true structure.

```python
from sklearn.datasets import make_s_curve
from sklearn.manifold import Isomap
import matplotlib.pyplot as plt

X, color = make_s_curve(n_samples=1000, random_state=42)

isomap = Isomap(n_neighbors=10, n_components=2)
X_isomap = isomap.fit_transform(X)

fig, ax = plt.subplots(1, 2, figsize=(12, 5))

ax[0].scatter(X[:, 0], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax[0].set_title('Original 3D Data')

ax[1].scatter(X_isomap[:, 0], X_isomap[:, 1], c=color,
cmap=plt.cm.Spectral)
ax[1].set_title('Isomap Reduced 2D Data')

plt.show()
```

Output:



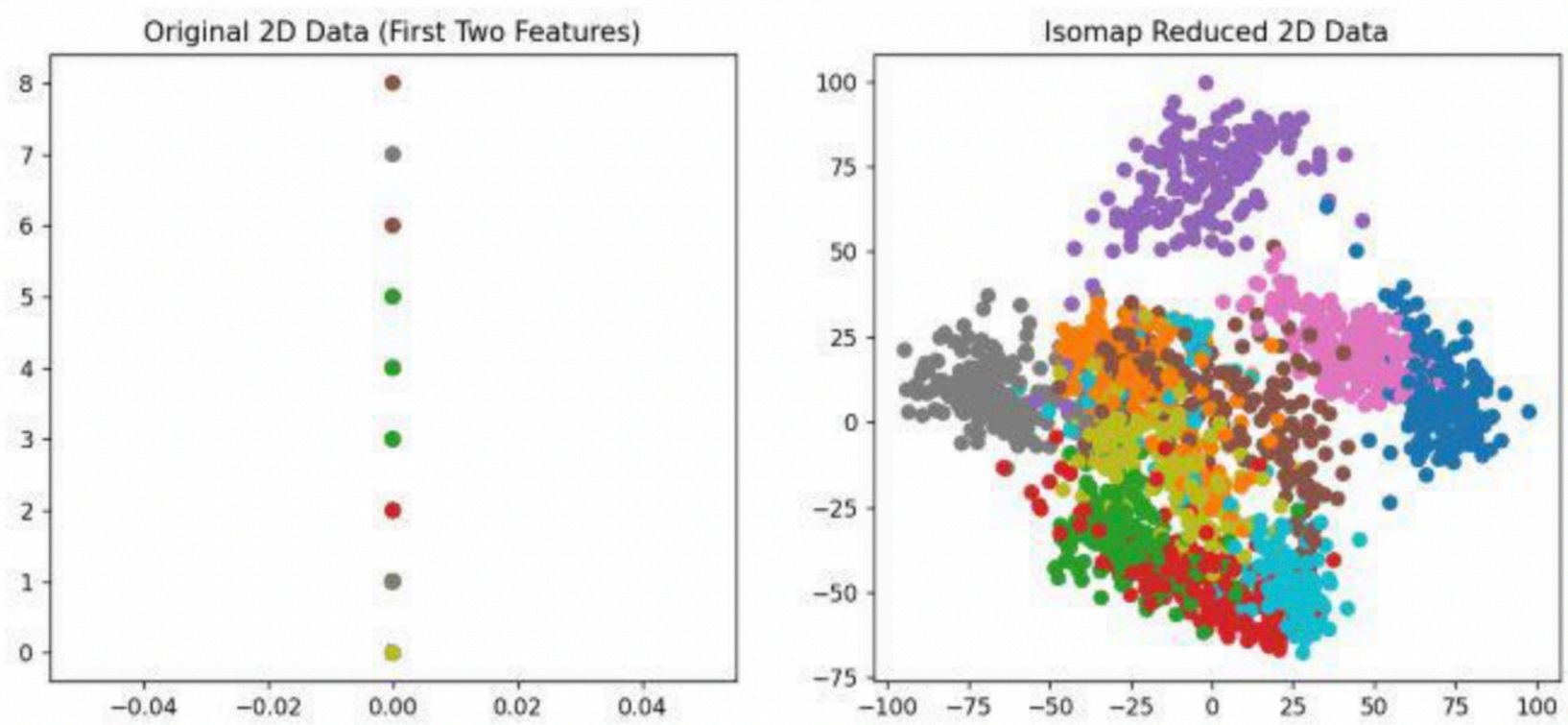
Output of the above code

Scatter plot shows how Isomap clusters S shaped dataset together while preserving the dataset's inherent structure.

APPLYING ISOMAP TO DIGITS DATASET

```python
from sklearn.datasets import load_digits
from sklearn.manifold import Isomap
import matplotlib.pyplot as plt

digits = load_digits()

isomap = Isomap(n_neighbors=30, n_components=2)
digits_isomap = isomap.fit_transform(digits.data)

fig, ax = plt.subplots(1, 2, figsize=(12, 5))

ax[0].scatter(digits.data[:, 0], digits.data[:, 1],
c=digits.target, cmap=plt.cm.tab10)
ax[0].set_title('Original 2D Data (First Two Features)')

ax[1].scatter(digits_isomap[:, 0], digits_isomap[:, 1],
c=digits.target, cmap=plt.cm.tab10)
ax[1].set_title('Isomap Reduced 2D Data')

plt.show()
```

Output:



Output of the above code

The scatter plot shows how Isomap clusters similar digits together in the 2D space, preserving the dataset's inherent structure.

ADVANTAGES OF ISOMAP

- Captures Non-Linear Relationships: Unlike PCA Isomap can find complex, non-linear patterns in data.
- Preserves Global Structure: It retains the overall geometry of the data and provide a more accurate representation of the data relationships.
- Global Optimal Solution: It guarantees that the optimal solution is found for the neighborhood graph and ensure accurate dimensionality reduction.

DISADVANATGES OF ISOMAP

- Computational Cost: Isomap can be slow for large datasets especially when calculating geodesic distances.
- Sensitive to Parameters: Incorrect parameter choices can led to poor results so it requires careful tuning.
- Complex Manifolds: It may struggle with data that contains topological complexity such as holes in the manifold.

APPLICATIONS OF ISOMAP

- Visualisation: It makes it easier to see complex data like face images by turning it into 2D or 3D form so we can understand it better with plots or graphs.
- Data Exploration: It helps to find groups or patterns in the data that might be hidden when the data has too many features or dimensions.
- Anomaly Detection: Outliers or anomalies in the data can be identified by understanding how they deviate from the manifold.
- Pre-processing for Machine Learning: It can be used as a pre-processing step before applying other machine learning techniques improve model performance

LOCALLY LINEAR EMBEDDING

- Locally linear embedding (LLE) seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. It can be thought of as a series of local Principal Component Analyses which are globally compared to find the best non-linear embedding.

- Locally linear embedding can be performed with function locally_linear_embedding or its object-oriented counterpart LocallyLinearEmbedding.

The standard LLE algorithm comprises three stages:

1. **Nearest Neighbors Search**. See discussion under Isomap above.
2. **Weight Matrix Construction**. $O[DNk^3]$. The construction of the LLE weight matrix involves the solution of a $k \times k$ linear equation for each of the N local neighborhoods.
3. **Partial Eigenvalue Decomposition**. See discussion under Isomap above.

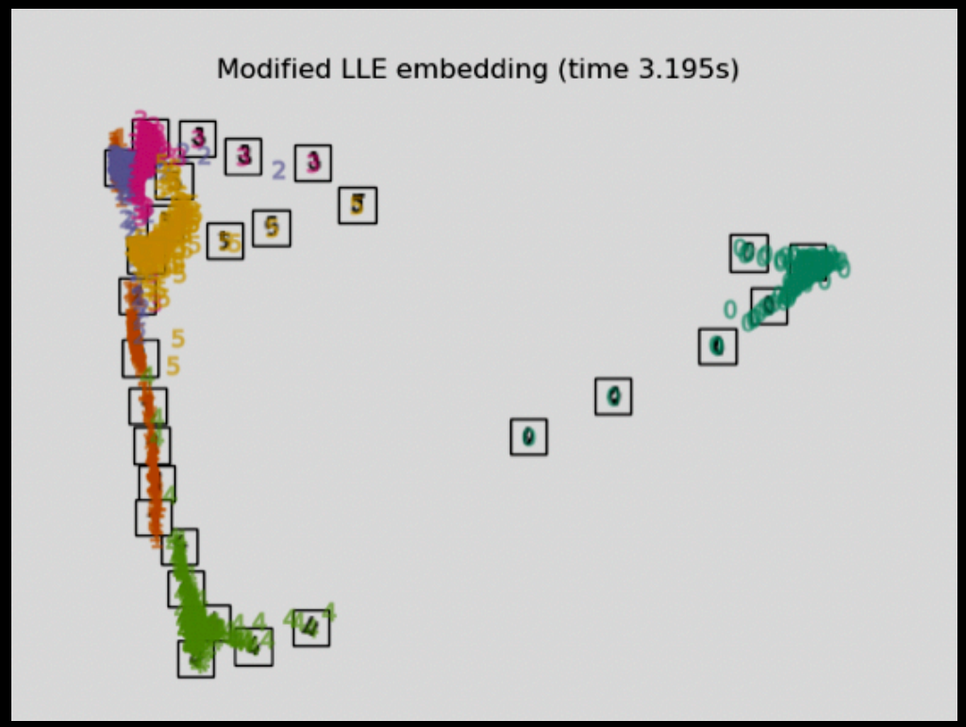
The overall complexity of standard LLE is $O[D \log(k) N \log(N)] + O[DNk^3] + O[dN^2]$.

- N : number of training data points
- D : input dimension
- k : number of nearest neighbors
- d : output dimension

MODIFIED LOCALLY LINEAR EMBEDDING

- One well-known issue with LLE is the regularization problem. When the number of neighbors is greater than the number of input dimensions, the matrix defining each local neighborhood is rank-deficient. To address this, standard LLE applies an arbitrary regularization parameter, which is chosen relative to the trace of the local weight matrix. Though it can be shown formally that as, the solution converges to the desired embedding, there is no guarantee that the optimal solution will be found for.

- This problem manifests itself in embeddings which distort the underlying geometry of the manifold.



COMPLEXITY

The MLLE algorithm comprises three stages:

1. **Nearest Neighbors Search**. Same as standard LLE
2. **Weight Matrix Construction**. Approximately $O[DNk^3] + O[N(k-D)k^2]$. The first term is exactly equivalent to that of standard LLE. The second term has to do with constructing the weight matrix from multiple weights. In practice, the added cost of constructing the MLLE weight matrix is relatively small compared to the cost of stages 1 and 3.
3. **Partial Eigenvalue Decomposition**. Same as standard LLE

The overall complexity of MLLE is

$$O[D\log(k)N\log(N)] + O[DNk^3] + O[N(k-D)k^2] + O[dN^2].$$

- N : number of training data points
- D : input dimension
- k : number of nearest neighbors
- d : output dimension

SPECTRAL EMBEDDING

- Spectral Embedding is an approach to calculating a non-linear embedding. Scikit-learn implements Laplacian Eigenmaps, which finds a low dimensional representation of the data using a spectral decomposition of the graph Laplacian. The graph generated can be considered as a discrete approximation of the low dimensional manifold in the high dimensional space.

- Minimization of a cost function based on the graph ensures that points close to each other on the manifold are mapped close to each other in the low dimensional space, preserving local distances. Spectral embedding can be performed with the function spectral_embedding or its object-oriented counterpart SpectralEmbedding.

COMPLEXITY

The Spectral Embedding (Laplacian Eigenmaps) algorithm comprises three stages:

1. **Weighted Graph Construction**. Transform the raw input data into graph representation using affinity (adjacency) matrix representation.
2. **Graph Laplacian Construction**. unnormalized Graph Laplacian is constructed as $L = D - A$ for and normalized one as $L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$.
3. **Partial Eigenvalue Decomposition**. Eigenvalue decomposition is done on graph Laplacian.

The overall complexity of spectral embedding is $O[D\log(k)N\log(N)] + O[DNk^3] + O[dN^2]$.

- N : number of training data points
- D : input dimension
- k : number of nearest neighbors
- d : output dimension

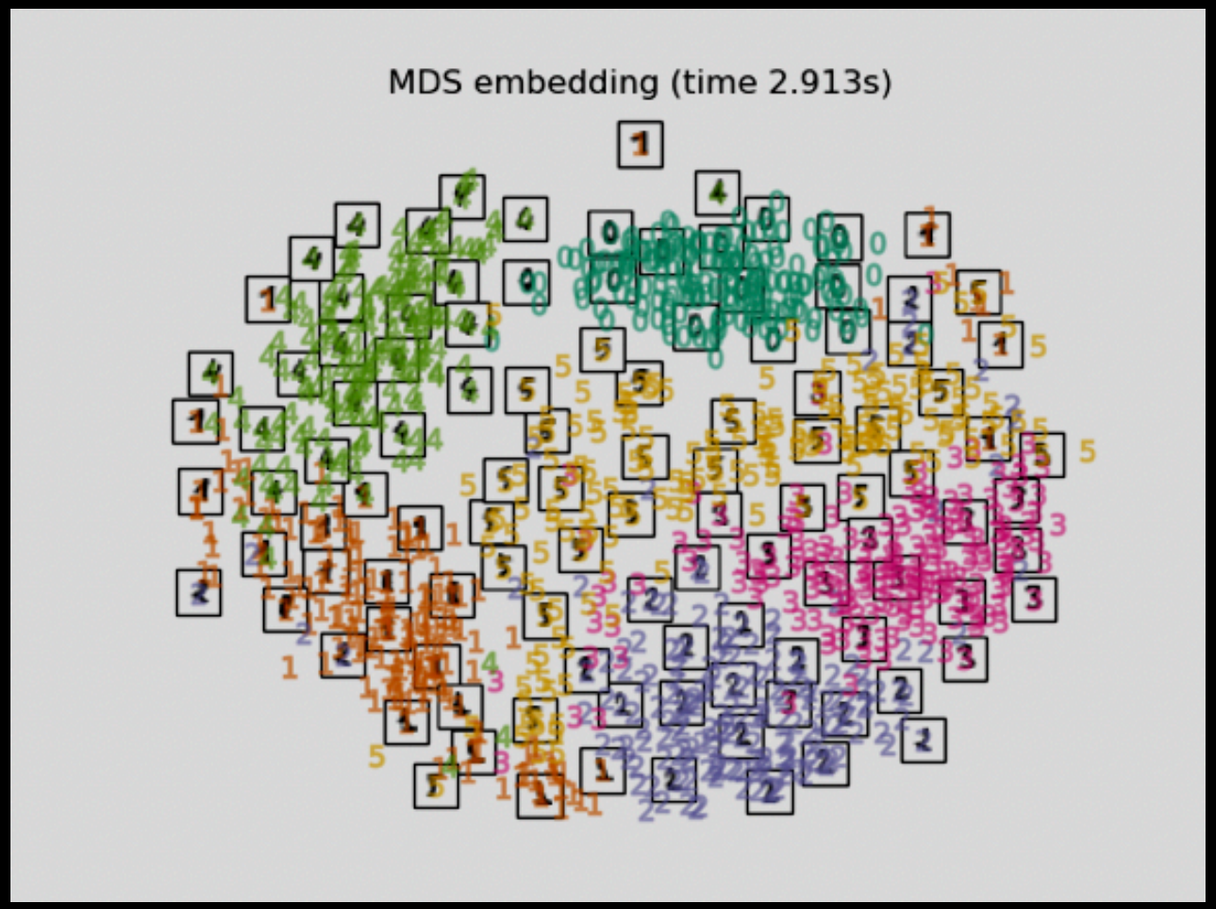
MULTI-DIMENSIONAL SCALING (MDS)

- Multidimensional scaling (MDS) seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space.

- In general, MDS is a technique used for analyzing dissimilarity data. It attempts to model dissimilarities as distances in a Euclidean space. The data can be ratings of dissimilarity between objects, interaction frequencies of molecules, or trade indices between countries.

- There exist two types of MDS algorithm: metric and non-metric. In scikit-learn, the class MDS implements both. In metric MDS, the distances in the embedding space are set as close as possible to the dissimilarity data. In the non-metric version, the algorithm will try to preserve the order of the distances, and hence seek for a monotonic relationship between the distances in the embedded space and the input dissimilarities.

Let δ_{ij} be the dissimilarity matrix between the n input points (possibly arising as some pairwise distances $d_{ij}(X)$ between the coordinates X of the input points). Disparities $\hat{d}_{ij} = f(\delta_{ij})$ are some transformation of the dissimilarities. The MDS objective, called the raw stress, is then defined by $\sum_{i<j}(\hat{d}_{ij} - d_{ij}(Z))^2$, where $d_{ij}(Z)$ are the pairwise distances between the coordinates Z of the embedded points.



TYPES OF MULTIDIMENSIONAL SCALING

1. Classical Multidimensional Scaling : Classical Multidimensional Scaling is a technique that takes an input matrix representing dissimilarities between pairs of items and produces a coordinate matrix that minimizes the strain.

Mathematically, strain is defined as:

$$\text{Strain}_D(x_1, x_2, \ldots, x_n) = \left(\frac{\sum_{i,j} (b_{ij} - x_i^T x_j)^2}{\sum_{i,j} b_{ij}^2} \right)^{1/2}$$

Where

- x_i denotes vectors in an N-dimensional space
- $?_i^T ?_?$ denotes the scalar product between x_i and x_j
- b_{ij} are the elements of the matrix B

The steps of a Classical MDS algorithm include setting up the squared proximity matrix $D^{(2)}$, applying double centering to compute matrix B, determining the m largest eigenvalues and corresponding eigenvectors of B, and obtaining the coordinates matrix X.

2. Metric Multidimensional Scaling : Metric Multidimensional Scaling generalizes the optimization procedure to various loss functions and input matrices with known distances and weights. It minimizes a cost function called "stress," often minimized using a procedure called stress majorization.Stress is defined as a residual sum of squares:

$$\text{Stress}_D(x_1, x_2, \ldots, x_n) = \sqrt{\sum_{i \neq j = 1,\ldots,n} (d_{ij} - \|x_i - x_j\|)^2}$$

3. Non-metric Multidimensional Scaling : Non-metric Multidimensional Scaling finds a non-parametric monotonic relationship between dissimilarities and Euclidean distances between items, along with the location of each item in the low-dimensional space. It defines a "stress" function to optimize, considering a monotonically increasing function f.

$$S(x_1, \ldots, x_n; f) = \sqrt{\frac{\sum_{i<j}(f(d_{ij}) - \hat{d}_{ij})^2}{\sum_{i<j} \hat{d}_{ij}^2}}$$

where

- d_{ij} are the observed dissimilarities between pairs of items i and j.
- $\widehat{d_{ij}}$ are the distances between items i and j in the lower-dimensional space.
- $f(d_{ij})$ is a monotonic transformation of the observed dissimilarities d_{ij} to best approximate the distances $\widehat{d_{ij}}$ in the reduced space.
- The summation $\Sigma_{i<j}$ is taken over all pairs of items.

CHOOSING BETWEEN TYPES

1. Classical MDS is chosen when the distance data are Euclidean and accurate preservation of these distances is crucial.

2. Metric MDS is suitable when distances are non-Euclidean or when the scale of measurement levels varies.

3. Non-metric MDS is beneficial for qualitative data or when only the order of distances (not the actual distances) matters.

COMPARISON WITH OTHER DIMENSIONALITY REDUCTION TECHNIQUES

Dimensionality Reduction Technique	Objective	Visualization	Applicability	Interpretation
Multidimensional Scaling (MDS)	Preserves original pairwise distances or dissimilarities	Provides intuitive visualizations of similarities/dissimilarities	Suitable for data with known dissimilarities or similarities, applicable across various domains	Emphasizes the preservation of relationships, facilitating qualitative interpretation
Principal Component Analysis (PCA)	Maximizes variance along orthogonal axes	Efficient for capturing global structure but may not preserve pairwise distances	Suitable for linear data transformations, often used for feature extraction	Focuses on capturing variance, useful for dimensionality reduction in high-dimensional data

t-Distributed Stochastic Neighbor Embedding (t-SNE)	Emphasizes local similarities by mapping high-dimensional data to a low-dimensional space	Creates dense clusters for similar data points, but distances are not preserved	Effective for visualizing high-dimensional data with complex structures	Primarily used for visualization, less emphasis on preserving global relationships
Isomap	Preserves geodesic distances to uncover underlying manifold structure	Captures non-linear relationships, useful for data with intrinsic dimensionality	Effective for data with non-linear structures, such as images or sensor networks	Focuses on uncovering intrinsic structure, helpful for understanding non-linear relationships

ADVANTAGES OF MULTIDIMENSIONAL SCALING

- Reduces the dimensionality of the original relationships between objects while preserving the original information, hence, helping to understand the objects better without the loss of crucial information.

- The adaptable nature of the scheme makes it suitable for various disciplines and data types, thus, allowing it to fit into any research category.

- It assists in discovering the hidden structures inside the data, thus, revealing the underlying patterns and relationships which may not be easily noticed.

- It helps to the hypothesis testing and the clustering analysis, thus the data-driven decision-making which is the basis of the scales.

LIMITATIONS OF MULTIDIMENSIONAL SCALING

- Sensitivity to outliers: The MDS results can be distorted by outliers, which in turn can affect the image or the interpretation of the connections.

- Computational complexity: MDS can be quite a process that demands a lot of computational resources and time, especially when it comes to large datasets.

- Subjectivity in interpretation: The process of interpreting MDS outcomes may be a matter of subjective decision of the meaning of the spatial arrangements which can result in the possible bias.

- Difficulty in determining the optimal number of dimensions: The right number of dimensions for the reduced space to be identified can be a difficult task and may necessitate of the experimentation.

APPLICATIONS OF MULTIDIMENSIONAL SCALING

1. Psychology and Cognitive Science:

- MDS is the standard approach in psychology to study the human perception, cognition and the process of decision making.

- It, on the other hand, helps the psychologists to realize the mechanism of the perception of the similarities or the differences between the stimuli, for example, the words, the images, or the sounds.

2. Market Research and Marketing:

- Market research applies MDS to the tasks of brand positioning, product positioning, and market segmentation.

- The marketers employ the MDS to visualize and interpret the consumer perceptions of the brands, products or services, which is hence they to make the decisions strategically and for the marketing campaigns.

3. Geography and Cartography:

- MDS is employed in geography and cartography to see and learn the spatial relationships between places, areas, or geographical features.

- It permits the cartographers to make maps that are true to the actual nature of the geographical entities and their close proximity to each other.

4. Biology and Bioinformatics:

- In biology, MDS is mostly applied for phylogenetic analysis, protein structure prediction and comparative genomics.

- Bioinformaticians employ MDS to represent and comprehend the similar or different genetic sequences, protein structures or evolutionary relationships among the different species.

5. Social Sciences and Sociology:

- MDS is utilized in sociology and the social sciences for the analysis of the social networks, intergroup relationships, and cultural differences.

- The sociologists employ the MDS to the survey data, the questionnaire responses or the relational data to understand the social structures and dynamics.

T-DISTRIBUTED STOCHASTIC NEIGHBOUR EMBEDDING (T-SNE)

- t-SNE (TSNE) converts affinities of data points to probabilities. The affinities in the original space are represented by Gaussian joint probabilities and the affinities in the embedded space are represented by Student's t-distributions. This allows t-SNE to be particularly sensitive to local structure and has a few other advantages over existing techniques:
    - Revealing the structure at many scales on a single map
    - Revealing data that lie in multiple, different, manifolds or clusters
    - Reducing the tendency to crowd points together at the center

- T-distributed Stochastic Neighbor Embedding (t-SNE) is a non linear dimensionality reduction technique used for visualizing high-dimensional data in a lower-dimensional space mainly in 2D or 3D. Unlike linear methods such as Principal Component Analysis (PCA), t-SNE focus on preserving the local structure and pattern of the data.

- t-SNE works by looking at the similarity between data points in the high-dimensional space. The similarity is computed as a conditional probability. It calculates how likely it is that one data point would be near another.

IMPLEMENTATION OF T-SNE ON MNIST DATASET

- Now let's use the sklearn implementation of the t-SNE algorithm on the MNIST dataset which contains 10 classes that are for the 10 different digits in the mathematics.

```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
```

Now let's load the MNIST dataset into pandas dataframe.

```
mnist = fetch_openml('mnist_784', version=1)

d = mnist.data
l = mnist.target

df = pd.DataFrame(d)
df['label'] = l

print(df.head(4))
```

Output:

```
   pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  pixel9  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0

   pixel10  ...  pixel776  pixel777  pixel778  pixel779  pixel780  pixel781  \
0        0  ...         0         0         0         0         0         0
1        0  ...         0         0         0         0         0         0
2        0  ...         0         0         0         0         0         0
3        0  ...         0         0         0         0         0         0

   pixel782  pixel783  pixel784  label
0         0         0         0      5
1         0         0         0      0
2         0         0         0      4
3         0         0         0      1

[4 rows x 785 columns]
```

Before applying the t-SNE algorithm on the dataset we must standardize the data. As we know that the t-SNE algorithm is a complex algorithm which utilizes some complex non-linear methods.

```
from sklearn.preprocessing import StandardScaler

standardized_data = StandardScaler().fit_transform(df)
print(standardized_data.shape)
```

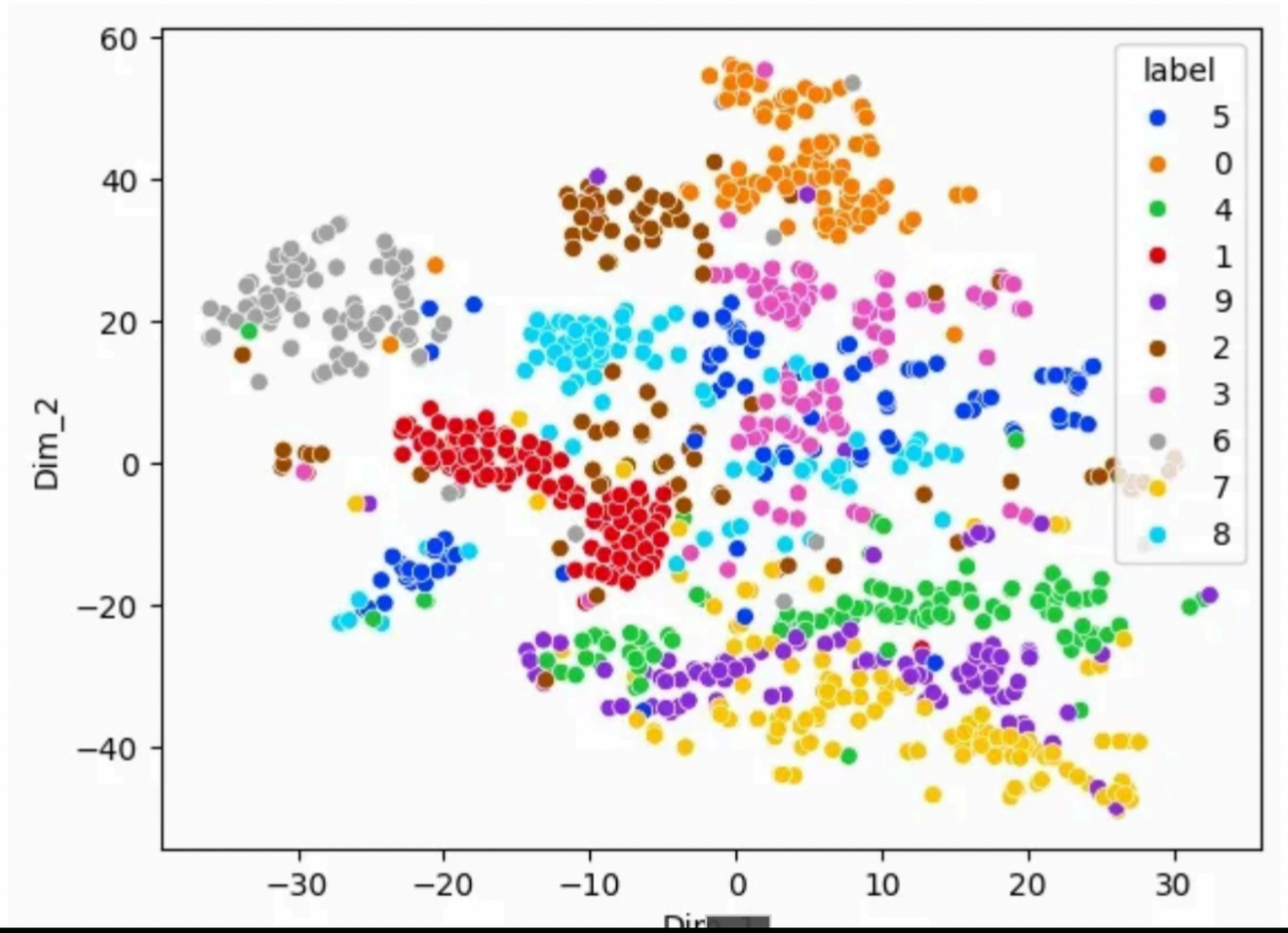
Output:

```
(70000, 785)
```

Now let's reduce the 784 columns data to 2 dimensions so that we can create a scatter plot to visualize the same.

```
data_1000 = standardized_data[0:1000, :]
labels_1000 = l[0:1000]

model = TSNE(n_components = 2, random_state = 0)

tsne_data = model.fit_transform(data_1000)

tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data = tsne_data,
     columns =("Dim_1", "Dim_2", "label"))

sn.scatterplot(data=tsne_df, x='Dim_1', y='Dim_2',
               hue='label', palette="bright")
plt.show()
```

Output:



DISADVANTAGES

- Computationally Intensive: t-SNE is slower and more computationally expensive compared to linear methods especially on large datasets.
- Non-deterministic Output: The output can vary with each run due to its randomness unless a fixed random_state is used.
- Not Scalable for Large Datasets: It struggles with very large datasets (e.g., millions of points) unless optimized or approximated versions are used.
- Not Good for Downstream Tasks: t-SNE is mainly for visualization and is not suitable for dimensionality reduction when feeding data into other ML algorithms.
- No Global Structure Preservation: It may distort global distances and structures in the data focusing more on preserving local neighborhoods.

ADVANTAGES OF T-SNE

- Great for Visualization: t-SNE is particularly used to convert complex high-dimensional data into 2D or 3D for visualization making patterns and clusters easy to observe.

- Preserve Local Structure: Unlike linear techniques like PCA t-SNE focus on maintaining the local relationships between data points meaning similar data points remain close in the lower-dimensional space.

- Non-Linear Capability: It captures non-linear dependencies in the data which makes it suitable for complex datasets where linear methods fail.

- Cluster Separation: Helps in clearly visualizing clusters and class separability in datasets like MNIST making it easier for interpretation and exploration.

OPTIMISING T-SNE

- The main purpose of t-SNE is visualization of high-dimensional data. Hence, it works best when the data will be embedded on two or three dimensions.

- Optimizing the KL divergence can be a little bit tricky sometimes. There are five parameters that control the optimization of t-SNE and therefore possibly the quality of the resulting embedding:

  - perplexity
  - early exaggeration factor
  - learning rate
  - maximum number of iterations
  - angle (not used in the exact method)

The perplexity is defined as $k = 2^{(S)}$ where S is the Shannon entropy of the conditional probability distribution. The perplexity of a k-sided die is k, so that k is effectively the number of nearest neighbors t-SNE considers when generating the conditional probabilities. Larger perplexities lead to more nearest neighbors and less sensitive to small structure. Conversely a lower perplexity considers a smaller number of neighbors, and thus ignores more global information in favour of the local neighborhood. As dataset sizes get larger more points will be required to get a reasonable sample of the local neighborhood, and hence larger perplexities may be required. Similarly noisier datasets will require larger perplexity values to encompass enough local neighbors to see beyond the background noise.

The maximum number of iterations is usually high enough and does not need any tuning. The optimization consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration the joint probabilities in the original space will be artificially increased by multiplication with a given factor. Larger factors result in larger gaps between natural clusters in the data. If the factor is too high, the KL divergence could increase during this phase. Usually it does not have to be tuned. A critical parameter is the learning rate. If it is too low gradient descent will get stuck in a bad local minimum. If it is too high the KL divergence will increase during optimization. A heuristic suggested in Belkina et al. (2019) is to set the learning rate to the sample size divided by the early exaggeration factor. We implement this heuristic as `learning_rate='auto'` argument. More tips can be found in Laurens van der Maaten's FAQ (see references). The last parameter, angle, is a tradeoff between performance and accuracy. Larger angles imply that we can approximate larger regions by a single point, leading to better speed but less accurate results.