Dimensionality Reduction for Data Visualization Using Nature-Inspired Algorithms

1. Introduction

This project aims to simplify the visualization of high-dimensional data through a comparative exploration of traditional and nature-inspired dimensionality reduction techniques. The system supports repeated executions to assess algorithm stability and performance variability.

2. Dataset

• Dataset Used: Load Digits Dataset from Scikit-learn

• Features: 64 numerical attributes representing pixel values of 8x8 images of handwritten digits (0–9).

• Classes: 10 distinct classes representing digits 0 to 9.

• Preprocessing: Standardization using StandardScaler was applied to normalize the feature set prior to dimensionality reduction

3. Algorithms Implemented

ALGORITHM	DESCRIPTION
PCA	Linear projection preserving global variance
T-SNE	Non-linear embedding preserving local structure
UMAP	Non-linear method optimizing topology and manifold learning
ISOMAP	Graph-based approach preserving geodesic distances

SELF-ORGANIZING MAP (SOM)	Neural network simulating topological self-organization
AUTOENCODER	Deep learning model compressing and reconstructing data
AUTOENCODER + T-SNE	Combines autoencoder-based compression with t-SNE visualization

4. Algorithm Descriptions

4.1 PCA (Principal Component Analysis)

• **How It Works:**

1. **Standardize the data** (mean = 0, variance = 1).
2. **Compute the covariance matrix** to understand how features vary together.
3. **Calculate eigenvectors** (shows a direction in the feature space) **and eigenvalues** (tells how important that direction) of the covariance matrix.
4. **Sort the eigenvectors** by their corresponding eigenvalues in descending order (most important first).
5. **Select the top *k* eigenvectors** to form the new feature space.
6. **Project the data** onto this new subspace.

• **Strengths:**

✓ Fast and efficient

✓ Effective on linear datasets

✓ Preserves global structure

• **Limitations:**

✗ Ineffective on non-linear datasets

✗ Sensitive to feature scaling

✗ May lose local data structure

• **Best For:**

Linear datasets, preprocessing for other machine learning tasks

• **Real-World Application:**

✓ Gene Expression Analysis – PCA helps reduce thousands of gene expression variables to visualize differences between cancerous and normal cells.

4.2 t-SNE (t-distributed Stochastic Neighbor Embedding)

• **How It Works:**

1. **Compute Pairwise Similarities:**
   In high-dimensional space, t-SNE measures the similarity between points using a Gaussian distribution.
2. **Map to Lower Dimensions:**
   It places points in 2D/3D space, it uses a Student's t-distribution to model the pairwise similarity.
3. **Minimize the Difference:**
   It minimizes the Kullback-Leibler divergence (KL divergence) between the high-dimensional and low-dimensional similarity distributions by gradient descent.

• **Strengths:**

✓ Excellent at revealing clusters

✓ Powerful for visualizing complex datasets

• **Limitations:**

✗ Computationally intensive ($O(n^2)$)

✗ Results vary between runs

✗ Global structure is not preserved

• **Real-World Application:**

✓ Word Embedding Visualization – Used to visualize semantic relationships in pre-trained word vectors (e.g., Word2Vec or GloVe).

4.3 UMAP (Uniform Manifold Approximation and Projection)

• **How It Works:**

1. **Builds a high-dimensional graph:**
   Models your data's local neighborhood using a distance metric (like Euclidean).
2. **Projects into low-dimensional space:**
   Tries to preserve the structure of the graph in a lower-dimensional space (e.g., 2D).
3. **Optimizes a cross-entropy loss between high- and low-dim graphs:**
   Unlike t-SNE (which minimizes KL-divergence), UMAP uses cross-entropy to balance local and global fidelity.

• **Strengths:**

✓ Faster than t-SNE

✓ Preserves both local and global structure

✓ More stable than t-SNE

• **Limitations:**

✗ Sensitive to hyperparameters

✗ May underperform with extremely high-dimensional data

• **Best For:**

Large-scale data visualization

• **Real-World Application:**

✓ Single-cell RNA sequencing (scRNA-seq) – UMAP is widely used to visualize cell clusters in biomedical research.

4.4 Isomap (Isometric Mapping)

• **How It Works:**

1. **Builds a neighborhood graph**:
   - Connect each point to its k-nearest neighbors.
   - Edges represent distances between connected points.

2. **Compute geodesic distances**:
   Use shortest path between two points along a curved surface between all pairs along the graph.
3. **Apply classical MDS**:
   Find a low-dimensional embedding that preserves these pairwise distances.

• **Strengths:**

✓ Effective for non-linear manifolds

✓ Outperforms PCA on curved datasets

• **Limitations:**

✗ Sensitive to noise

✗ Requires full connectivity for meaningful paths

• **Best For:**

Manifold learning in datasets like 3D objects and sensor networks

• **Real-World Application:**

✓ 3D Pose Estimation – Isomap is used to reduce high-dimensional motion capture data for visualization and clustering.

4.5 SOM (Self-Organizing Map)

• **How It Works:**

1. Initialize a 2D grid of nodes (neurons), each with a random weight vector the same size as input vectors.
2. **For each input:**
   - Find the Best Matching Unit (BMU) — the node whose weight vector is closest to the input.
   - Update BMU and its neighbors to move closer to the input vector.
   - Over time, neighboring neurons become specialized to similar inputs.
3. **This results in a 2D map where:**
   After many rounds, each neuron becomes specialized — One neuron might represent "red flowers".

• **Strengths:**

✓ Produces intuitive visualizations

✓ Maintains topological relationships

• **Limitations:**

✗ Rigid 2D grid structure

✗ Requires careful tuning of parameters

• **Best For:**

Clustering, interpretable 2D mappings

• **Real-World Application:**

✓ Customer Segmentation – Used in marketing to segment customers based on purchase behavior and demographics.

4.6 Autoencoder

• **How It Works:**

1. **Encoder:**
   - Learns to compress the input.
   - Example: 784-dim image → 32-dim code.
2. **Bottleneck (Latent Space):**
   - The compressed low-dimensional representation.
   - This is the useful part for visualization or clustering.
3. **Decoder:**
   - Learns to reconstruct the input from the compressed code.

• **Strengths:**

✓ Learns complex non-linear structures

✓ Flexible architecture (e.g., denoising, variational)

• **Limitations:**

✗ Requires large training data

✗ Interpretability is low compared to linear models

• **Best For:**

High-dimensional data (e.g., images, audio), anomaly detection

• **Real-World Application:**

✓ Network Intrusion Detection – Autoencoders detect anomalies in network traffic by learning a compact representation of normal patterns.

4.7 Autoencoder + t-SNE

• **How It Works:**

1. **Train Autoencoder:**
   Input → Encoder → Bottleneck → Decoder → Output
   Goal: output ≈ input
2. **Extract Bottleneck Features:**
   Use only the encoder to compress your data:
   compressed_data = encoder.predict(X)
3. **Apply t-SNE:**
   Use t-SNE(n_components=2).fit_transform(compressed_data) to reduce the compressed vectors to 2D.
4. **Plot the Result :**
   You'll get a 2D scatter plot showing natural clusters and patterns.

• **Strengths:**

✓ More efficient than pure t-SNE

✓ Preserves structure while denoising input

• **Limitations:**

✗ Inherits t-SNE limitations

✗ Requires two-step model training

• **Best For:**

Extremely high-dimensional datasets (e.g., genomics)

• **Real-World Application:**

✓ Genomics Data Visualization – Applied in bioinformatics to reduce and visualize thousands of gene features in genetic disease studies.

5. GUI Functionality

- Dropdown to select dimensionality reduction algorithm
- Dynamic parameter input fields based on selected algorithm
- "Run" button to execute algorithm
- "Run 30 Repetitions" button for reproducibility testing
- Embedded matplotlib canvas to display 2D scatter plots

7. Parameter Customization

METHOD	PARAMETERS	ROLES
PCA	n_components	Specifies how many principal components you want to keep when reducing the dimensionality
T-SNE	n_components perplexity	Effective neighbors : Low perplexity → focuses more on local structure (small clusters) High perplexity → preserves larger-scale structure
UMAP	n_components n_neighbors min_dist	Controls local & global structure preservation. Controls how tightly UMAP clusters. Low min_dist (e.g., 0.1) → Tighter clusters. High min_dist (e.g., 0.8) → Looser, more spread-out clusters.
ISOMAP	n_components n_neighbors	
SOM	grid_size	Determines the number of nodes (neurons) available

		to represent the input space. Larger grids → finer representation but slower training. Smaller grids → faster but may under-represent complex data.
	sigma	Controls the degree of smoothing during training. High sigma → broad influence (more generalization). Low sigma → narrow influence (more detail).
	learning_rate	Determines the update strength for neurons during training. Higher learning rate → faster adaptation, but risk of instability. Lower learning rate → slower but more stable training.
AUTOENCODER	encoding_dim	Smaller encoding_dim → More compression (may lose information) Larger encoding_dim → Retains more information but less compression
	epochs	Monitor loss on validation data to avoid overtraining.

	batch_size	Number of samples processed before model weights are updated. Smaller batch size → Slower but more generalizable Larger batch size → Faster training but risk of local minima.
AUTOENCODER + T-SNE	All above + perplexity	

8. Self-Organizing Map (SOM) Configuration

- **Library Used:** MiniSom
- **Grid Topology:** grid_size × grid_size
- **Training Iterations:** 100
- **Distance Metric:** Euclidean
- **Initialization:** Random weights

Here is how you can incorporate the **Results** section into your documentation in a clear, professional, and consistent format:

---

9. Results

To evaluate and compare the effectiveness of each dimensionality reduction technique, multiple quantitative metrics were computed. These include:

- **ARI (Adjusted Rand Index):** Measures clustering similarity to ground truth labels.
- **NMI (Normalized Mutual Information):** Quantifies shared information between cluster assignments and true classes.
- **Silhouette Score:** Assesses how well-separated and compact the clusters are.
- **Trustworthiness:** Indicates the preservation of local structure from high- to low-dimensional space.
- **Accuracy (KNN):** Classification accuracy using a 5-Nearest Neighbors classifier on the 2D-transformed data.

- **Time (s):** Execution time in seconds (single run).

Algorithm	ARI	NMI	Silhouette	Trustworthiness	Accuracy (KNN)	Time (s)
PCA	0.3249	0.4642	0.3770	0.8180	0.6767	3.066
t-SNE	0.7705	0.8333	0.5750	0.9928	0.9783	4.362
UMAP	**0.8712**	**0.8985**	**0.6961**	0.9804	**0.9811**	2.892
Isomap	0.5576	0.7010	0.4739	0.8575	0.8692	3.095
SOM	0.1357	0.2535	0.4284	0.9495	0.8531	2.643
Autoencoder	0.1074	0.2359	0.3492	0.7191	0.5565	2.948
Autoencoder + t-SNE	0.6890	0.7716	0.5212	**0.9868**	0.9577	3.344

Key Observations:

- **UMAP** consistently outperformed other techniques across most metrics including ARI, NMI, Silhouette Score, and KNN accuracy, making it the most effective method for this dataset.
- **t-SNE** and **Autoencoder + t-SNE** also delivered high-quality visualizations with strong clustering behavior and trustworthiness.
- **SOM** performed surprisingly well in preserving topology but had limited clustering capability.
- **PCA** and **Isomap** were moderate performers, with PCA being faster but less expressive.
- **Autoencoder** alone underperformed in this context, indicating that raw latent space may not always align with class boundaries without additional techniques like t-SNE.

Let me know if you want a visual comparison (e.g., a bar chart or radar plot) or if you want to include visual outputs (scatter plots) for each method.

10. Repetition and Reproducibility

• The “Run 30 Repetitions” feature runs the selected algorithm 30 times with different random seeds.

• **Seed Logging:** All seeds used are saved to seeds_used.txt for future reference and reproducibility.