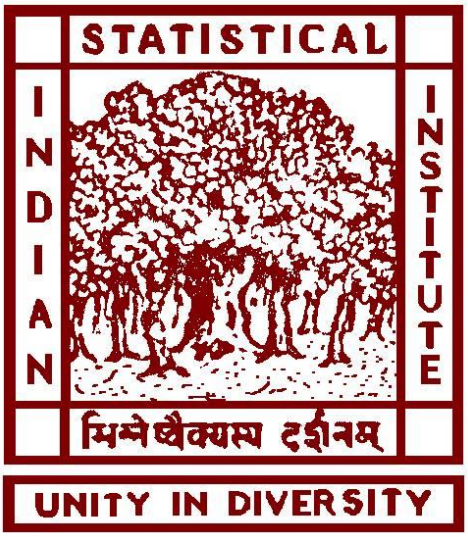
INDIAN STATISTICAL INSTITUTE



ASSIGNMENT

Select a total of three dimensionality reduction techniques (projection based) related to feature extraction, write down its mathematical/statistical principles, write codes to execute for a big dataset (construct yourselves) and describe their comparative results.

Submitted By:

BELAL AHMED SIDDIQUI

SURAJ KUMAR

PD SAI SRINIVAS

PIYUSH DHURWEY

VIJAY SURESHCHANDRA YADAV

HEMANTA KUMAR MANDAL

Introduction to Dimensionality Reduction Techniques

Dimensionality reduction is a crucial step in data preprocessing, especially when dealing with high-dimensional datasets. It involves reducing the number of features (dimensions) in a dataset while retaining as much information as possible. This process helps in mitigating the "curse of dimensionality," improving model performance, and facilitating data visualization. Among various dimensionality reduction techniques, **t-SNE (t-Distributed Stochastic Neighbor Embedding), Isomap (Isometric Mapping), and UMAP (Uniform Manifold Approximation and Projection)** are popular projection-based methods. These techniques are particularly effective in uncovering the underlying structure of complex, high-dimensional data.

1. t-SNE (t-Distributed Stochastic Neighbor Embedding)

Mathematical/Statistical Principles: t-SNE is a nonlinear dimensionality reduction technique that excels in visualizing high-dimensional data. It works by minimizing the divergence between two distributions: one representing pairwise similarities of the points in the high-dimensional space and the other representing the corresponding similarities in the low-dimensional space. The similarities are computed using conditional probabilities that measure the likelihood of one point being a neighbor of another, based on a Gaussian distribution in the high-dimensional space and a Student's t-distribution in the low-dimensional space.

Key Steps:

1. Compute pairwise similarities between data points in the high-dimensional space.
2. Convert these similarities into conditional probabilities using a Gaussian distribution.
3. Map these probabilities into a low-dimensional space using a Student’s t-distribution.
4. Minimize the Kullback-Leibler divergence between the two distributions using gradient descent.

2. Isomap (Isometric Mapping)

Mathematical/Statistical Principles: Isomap is a manifold learning technique that seeks to preserve the global geometric structure of the data. It extends classical Multidimensional Scaling (MDS) by incorporating geodesic distances rather than Euclidean distances. Isomap constructs a neighborhood graph, estimates the shortest paths between all pairs of points (geodesic distances), and then applies MDS to these distances to obtain a lower-dimensional embedding. This approach is particularly effective for data that lies on a non-linear manifold.

Key Steps:

1. Construct a neighborhood graph using k-nearest neighbors or a fixed-radius approach.
2. Compute the shortest paths between all pairs of points using Dijkstra’s or Floyd-Warshall algorithm.
3. Apply classical MDS to the geodesic distance matrix to obtain the low-dimensional embedding.

3. UMAP (Uniform Manifold Approximation and Projection)

Mathematical/Statistical Principles: UMAP is a powerful and flexible dimensionality reduction technique based on manifold learning and topological data analysis. It constructs a high-dimensional graph representation of the data by balancing local and global structure preservation. UMAP then optimizes a low-dimensional graph to be as structurally similar to the high-dimensional graph as possible. The method is based on rigorous mathematical foundations, including concepts from Riemannian geometry and algebraic topology.

Key Steps:

1. Construct a weighted k-nearest neighbor graph in the high-dimensional space.
2. Optimize a fuzzy topological representation of this graph.
3. Map the high-dimensional graph to a low-dimensional space by minimizing the cross-entropy between the high-dimensional and low-dimensional representations.

t-SNE

(t-distributed Stochastic Neighbor Embedding)

Introduction:

t-SNE (t-Distributed Stochastic Neighbour Embedding) is a powerful machine learning algorithm designed for dimensionality reduction, particularly suited for visualizing high-dimensional data in lower dimensions, typically two or three. Unlike traditional methods like Principal Component Analysis (PCA), t-SNE focuses on preserving the local structure of the data by converting similarities between data points in the high-dimensional space into probabilities and then mapping them to a lower-dimensional space. This method is particularly effective in revealing complex patterns, clusters, and relationships within the data that might not be easily discernible in higher dimensions. Its ability to maintain local relationships makes t-SNE a popular tool in exploratory data analysis, especially in fields like bioinformatics, natural language processing, and computer vision, where data often reside in complex, high-dimensional spaces.

Principles:

Stochastic measures for deciding Neighbourhood:

For points in higher dimension

Suppose we have high dimensional Datapoints $x_1, x_2, x_3, \cdots x_n$ and our goal is to project them onto a 2D or 3D space as Datapoints $y_1, y_2, y_3, \cdots y_n$ .The neighbourhood of a particular data point i in the higher dimensional space is decided by the probability measure $p_{j|i}$ which considers both the facts that the closer the point j is to the point i, the more is the chance of considering it as a neighbour and when the points other than j are spread even distant from i then j becomes the more obvious neighbour. Such $p_{j|i}$ would look as written below

$$p_{j|i} = \frac{\exp\left[-\left\|X_i - X_j\right\|^2 / \sigma_i\right]}{\sum_{k \neq i} \exp\left[-\|X_i - X_k\|^2 / \sigma_i\right]}$$

The choice of the standard deviation σ_i depends on the number of neighbours you would like to fit in terms of their distance from the point i in a normal probability distribution within a preselected multiple of this standard deviation

We can see that the probability is assigned proportional to the density of the normal distribution of the distance. Now, the probability that the points i and j are grouped in the higher dimensional space is given by p_{ij} which is proportional to the average of the

probabilities for each of the points i, j being seen as a neighbour by the point of consideration,i.e $p_{i|j}$ and $p_{j|i}$

$$p_{ij} = \frac{1}{2N} p_{i|j} + \frac{1}{2N} p_{j|i}$$

For points in lower dimension

The same neighbourhood measure can work for the points projected to the lower(2 or 3) dimensions but with a small change. The probability that point j is selected as neighbour of point i in the lower dimensional space is proportional to t-distribution with symmetric probability density curve similar to normal density curve but with heavier tails.These heavier tails can accommodate points in the lower dimensional space which went through the curse of dimensionality during projection. When we want to preserve the similarity between points in very high dimensional space while projecting them onto a lower dimension, the projections becomes sparse to compensate the lack of room as we lose on the dimensions which had accommodated them in the higher dimensional space. So the t distribution with degree of freedom '1' ,also known as Cauchy distribution can be used to consider the neighbourhood points which got spread out after projection, by giving a greater probability to them compared to normal distribution. So the probability that the point j is selected as neighbour to i in the lower dimensional space is given as

$$q_{j|i} = \frac{\left[1 + \left\|Y_i - Y_j\right\|^2\right]^{-1}}{\sum_{k \neq l} [1 + \|Y_k - Y_l\|^2]^{-1}}$$

and the probability that the point i and j are considered to be of one cluster is given as

$$q_{ij} = \frac{1}{2N} q_{i|j} + \frac{1}{2N} q_{j|i}$$

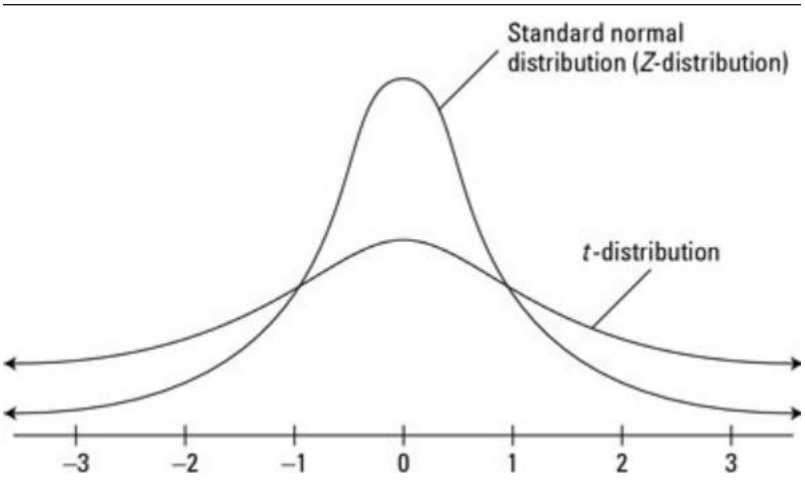


Figure 1:Comparision of Normal and Cauchy distribution

Objective function:

As similarity among the points expected to be preserved during projection, the probabilities of the points i and j belonging to the same cluster in both higher and lower dimensional spaces must be same.This needs to be achieved by minimizing the difference between p_{ij} and q_{ij}.KL divergence shown below, is a measure representing this difference, so minimizing KL divergence using Gradient Descent with respect to the points Y_i and Y_j would give us final projection of higher dimensional points onto a lower dimensional space.

OBJECTIVE FUNCTION

$$min(KL(P|Q))$$

where the function *KL* represents KL Divergence for given i and j, which is given by

KL DIVERGENCE

$$KL(P|Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

Isomap: Isometric Mapping

1 Introduction

Isomap, or Isometric Mapping, is a nonlinear dimensionality reduction technique that aims to preserve the intrinsic geometric structure of high-dimensional data by maintaining the geodesic distances between points. Unlike linear techniques such as PCA, Isomap is designed to uncover the underlying manifold structure in the data, which is particularly useful for datasets that are distributed on a nonlinear manifold. Isomap builds on classical Multidimensional Scaling (MDS) by incorporating geodesic distances instead of simple Euclidean distances, thus providing a more accurate low-dimensional representation of the data.

2 Mathematical Foundations

Isomap is grounded in several key mathematical principles that allow it to effectively map high-dimensional data to a lower-dimensional space while preserving its essential structure.

2.1 Geodesic Distance

The geodesic distance between two points on a manifold is the length of the shortest path along the surface of the manifold that connects them. This concept is crucial for Isomap because it allows the algorithm to account for the true underlying geometry of the data, which may be distorted in high-dimensional space. Unlike the Euclidean distance, which may underestimate the true distance between points on a curved surface, the geodesic distance captures the correct separation by following the manifold's curvature.

2.2 Graph Construction

To approximate the manifold's structure, Isomap represents the data as a weighted graph $G = (V, E)$, where each vertex $v_i \in V$ corresponds to a data point x_i. The edges $e_{ij} \in E$ are created between each data point and its k-nearest neighbors based on the Euclidean distance. The weight of an edge w_{ij} is the Euclidean distance between the connected points, which serves as an initial approximation of the local geodesic distance.

$$w_{ij} = \begin{cases} d(x_i, x_j) & \text{if } x_j \text{ is a neighbor of } x_i, \\ \infty & \text{otherwise.} \end{cases}$$

This step effectively captures the local neighborhood structure of the manifold.

2.3 Shortest Path Algorithms

Once the graph is constructed, Isomap uses a shortest path algorithm, such as Floyd-Warshall or Dijkstra's algorithm, to compute the shortest paths between all

pairs of points. These paths represent the geodesic distances on the manifold. The idea is that by piecing together these short paths, we can approximate the global structure of the manifold.

$$d_G(x_i, x_j) = \min \left(\sum_{e_{ab} \in P_{ij}} w_{ab} \right),$$

where P_{ij} represents all possible paths between points x_i and x_j, and w_{ab} is the weight of edge e_{ab}. The computed $d_G(x_i, x_j)$ gives a more accurate measure of the true distance between points on the manifold.

2.4 Classical Multidimensional Scaling (MDS)

After computing the geodesic distances, Isomap applies classical Multidimensional Scaling (MDS) to these distances to find a low-dimensional embedding of the data. The MDS technique seeks to preserve the pairwise distances between points in the low-dimensional space, effectively capturing the global structure of the manifold. The key step in MDS involves solving an eigenvalue problem for the doubly centered distance matrix:

$$\mathbf{B} = \mathbf{H}\mathbf{D}_G\mathbf{H},$$

where $\mathbf{B}$ is the doubly centered distance matrix, and $\mathbf{H}$ is the centering matrix defined as:

$$\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top,$$

with $\mathbf{I}$ being the identity matrix and $\mathbf{1}$ being a vector of ones. The matrix $\mathbf{B}$ is then diagonalized to extract the top d eigenvectors, which correspond to the low-

dimensional coordinates of the data points.

3 Objective Function

The goal of Isomap is to minimize the stress function, which quantifies the difference between the geodesic distances in the original high-dimensional space and the Euclidean distances in the low-dimensional embedding. The stress function is defined as:

$$\text{Stress} = \sum_{i<j} \left(d_G(x_i, x_j) - \|y_i - y_j\|\right)^2,$$

where y_i and y_j are the low-dimensional representations of the high-dimensional data points x_i and x_j, respectively, and $\|\cdot\|$ denotes the Euclidean distance in the low-dimensional space. By minimizing this stress function, Isomap ensures that the low-dimensional embedding preserves the manifold's original geometry as closely as possible.

4 Applications of Isomap

Isomap has been successfully applied in various fields due to its ability to reveal the intrinsic low-dimensional structure of high-dimensional data:

- **Manifold Learning:** Isomap is extensively used in manifold learning to uncover and visualize the hidden structure of complex datasets, such as those found in robotics, biology, and astronomy.

- **Image Processing:** In image processing, Isomap is used to reduce the dimensionality of image data, making it easier to analyze and process while preserving the relationships between different images.
- **Genomics:** In genomics, Isomap helps in reducing the dimensionality of gene expression data, enabling the discovery of patterns and relationships that are not easily detectable in the original high-dimensional space.
- **Speech Analysis:** Isomap has been applied to speech data to reduce its dimensionality, which improves the efficiency and accuracy of speech recognition systems.

5 Summary

Isomap is a powerful tool for nonlinear dimensionality reduction, particularly when the objective is to preserve the global structure of the data as reflected by geodesic distances. By combining graph-based distance computation with classical MDS, Isomap effectively uncovers the low-dimensional manifold structure of high-dimensional datasets, making it a valuable technique for manifold learning and various data analysis tasks.

UMAP(Uniform Manifold Approximation and Projection)

Uniform Manifold Approximation and Projection (UMAP) is a powerful dimension reduction technique that has gained significant traction in the fields of machine learning and data visualization. Developed by Leland McInnes, John Healy, and James Melville, UMAP is built on solid mathematical foundations, including Riemannian geometry and algebraic topology. This article delves into the technical aspects of UMAP, its underlying principles, implementation, and practical applications.

UMAP, was introduced by Leland McInnes, John Healy, and James Melville in 2018, it gained immediate popularity for its ability to address the shortcomings of traditional methods. UMAP belongs to the family of manifold learning techniques and stands out for its efficiency, scalability, and capability to capture both local and global structures in the data.

Mathematical Foundations for UMAP

UMAP is grounded in several key mathematical concepts:

- **Riemannian Manifold**: UMAP assumes that the data is uniformly distributed on a Riemannian manifold. This means that the data points lie on a smooth, curved surface that can be locally approximated by Euclidean space.
- **Riemannian Metric**: The Riemannian metric is locally constant or can be approximated as such. This metric defines the distance between points on the manifold.
- **Topological Data Analysis**: UMAP leverages topological data analysis to capture the structure of the data. It constructs a fuzzy topological representation of the data, which is then optimized to find a low-dimensional embedding.

1. Graph Construction

a. Nearest Neighbor Search

For each data point x_i, UMAP identifies its k-nearest neighbors. This is typically done using a distance metric such as Euclidean distance. Let's denote the set of k-nearest neighbors of x_i as $\{x_{i1}, x_{i2}, \ldots, x_{ik}\}$.

b. Similarity Computation

The similarity between points x_i and x_j is computed using a Gaussian kernel:

$$sim_{ij} = exp(-\frac{d(xi, xj)^2}{\sigma_i^2})$$

where $d(x_i, x_j)$ is the Euclidean distance between x_i and x_j , and σ_i is a local scaling parameter. The value of σ_i is chosen such that the probability p_{ij} of selecting x_i as a neighbor of x_j can be controlled, typically by ensuring that only k neighbors are considered.

The scaling parameter σ_i is chosen so that the distance $d(x_i,x_j)$ falls within a range that makes sim_{ij} meaningful for the local neighborhood structure.

2. Probabilistic Modeling

a. High-Dimensional Space

For a given data point x_i, the probability of x_j being a neighbor is given by:

$$p_{ij} = \frac{sim_{ij}}{\sum_{k \neq i} sim_{ij}}$$

This is the probability of observing x_j given x_i in the high-dimensional space. It ensures that the probability mass is distributed according to the similarities between x_i and its neighbors.

b. Low-Dimensional Space

In the low-dimensional space, we want to model the data points y_i and y_j such that the distances between them reflect the similarities from the high-dimensional space. The probability q_{ij} of y_j being a neighbor of y_i is given by:

$$q_{ij} = \frac{\exp\left(-dist(y_i, y_j)2\right.}{\sum_{k \neq i} \exp\left(-dist(y_i, y_k)2\right)}$$

where dist(yi,yj)\ is typically the Euclidean distance between yi and yj in the low-dimensional space.

3. Objective Function

UMAP's objective is to make the low-dimensional probability distribution qij as similar as possible to the high-dimensional probability distribution pij. To achieve this, UMAP minimizes a loss function based on the divergence between pij and qij. The typical objective function used is:

$$L = \sum_i \sum_{j \neq i} \log\left(\frac{p_{ij}}{q_{ij}}\right) p_{ij}$$

This can also be expressed in terms of Kullback-Leibler divergence:

$L=KL(P\|Q)= \sum_i \sum_{j \neq i} \log\left(\frac{p_{ij}}{q_{ij}}\right) p_{ij}$

where KL(P‖Q) measures how one probability distribution P diverges from a second, expected probability distribution Q.

4. Optimization

UMAP optimizes the objective function using stochastic gradient descent (SGD). The steps involved are:

a. Initialization

- Initialize the low-dimensional coordinates {yi} randomly.

b. Gradient Calculation

- Compute gradients of the objective function with respect to the low-dimensional coordinates. The gradient for the loss function L is:
- $\frac{\delta L}{\delta y_i} = -\sum_{j \neq i}(p_{ij} - q_{ij}) \frac{\delta}{\delta y_i} dist(y_i, y_j)2$

Where $\frac{\delta}{\delta y_i} dist(y_i, y_j)2$ represents how changes in yi affect the dist(yi,yj).

c. Update

- Update the coordinates yi using the gradients and a learning rate η:
- $y_i \leftarrow y_i - \eta \frac{\delta L}{\delta y_i}$

5. Embedding

Through the optimization process, the coordinates {yi} are adjusted to minimize L. The final embedding is a low-dimensional representation where the pairwise distances approximate the similarities in the high-dimensional space, effectively capturing the manifold structure.

Applications of UMAP

UMAP has been applied in various fields, including genomics, where it has helped reveal cryptic population structures and phenotype correlations

Classical Machine Learning: UMAP is employed in various machine learning tasks, such as clustering, classification, and anomaly detection. Its ability to preserve the intrinsic structure of the data makes it a valuable preprocessing step for improving the performance of machine learning models.

Image Analysis: UMAP has proven effective in image analysis, aiding researchers and practitioners in visualizing and exploring high-dimensional image datasets. This is particularly valuable in fields such as computer vision and medical imaging.

Social Network Analysis: UMAP is applied to understand and visualize relationships in social networks. By reducing the dimensionality of the data while preserving relevant structures, it becomes easier to identify communities and patterns in large-scale social graphs.

In summary, UMAP stands out as a crucial tool for exploratory data analysis and preprocessing within machine learning pipelines, thanks to its efficient handling of large datasets and its adept preservation of both local and global data structures.

Comparative Analysis of Computational Complexity of t-SNE, Isomap, and UMAP

1. t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Computational Complexity:**
  - The exact t-SNE algorithm has a complexity of $O(N^2)$, where N is the number of data points. This complexity arises from the calculation of pairwise similarities between data points.
  - The Barnes-Hut t-SNE approximation reduces the complexity to $O(N \log N)$ by using a quadtree to approximate the joint probability distribution, making it more feasible for large datasets.
- **Memory Complexity:**
  - The memory complexity is $O(N^2)$, primarily due to storing the pairwise similarity matrix.

2. Isomap (Isometric Mapping)

- **Computational Complexity:**
  - The complexity of Isomap is dominated by three main steps:
    1. **Constructing the k-nearest neighbor graph:** This step typically has a complexity of $O(N^2 \log N)$.
    2. **Computing the shortest paths using the Floyd-Warshall algorithm:** The complexity is $O(N^3)$, but with Dijkstra's algorithm and a Fibonacci heap, it can be reduced to $O(N^2 \log N)$.
    3. **Performing Multidimensional Scaling (MDS):** The complexity is $O(N^3)$, as it involves eigenvalue decomposition of the geodesic distance matrix.
- **Memory Complexity:**
  - The memory complexity is $O(N^2)$, due to the storage of the geodesic distance matrix.

3. UMAP (Uniform Manifold Approximation and Projection)

- **Computational Complexity:**
  - UMAP's complexity is approximately $O(N \log N)$ for the construction of the k-nearest neighbor graph using approximate nearest neighbors, followed by an optimization process with complexity around $O(N \log N)$.

  - The actual embedding step is typically linear, making UMAP much faster than t-SNE for large datasets.

- **Memory Complexity:**
  - The memory complexity of UMAP is generally $O(N)$, making it more memory-efficient compared to t-SNE and Isomap.

Summary

- **t-SNE:** $O(N^2)$ complexity, but $O(N \log N)$ with approximations (Barnes-Hut t-SNE).
- **Isomap:** Complexity ranges from $O(N^2 \log N)$ to $O(N^3)$, depending on the algorithm used for shortest path computation and MDS.
- **UMAP:** $O(N \log N)$ complexity, making it the most scalable among the three for large datasets.

Comparative Analysis of t-SNE,Isomap,and UMAP using Trustworthiness metric

1 Dataset Generation

The dataset was generated using the `make_classification` function from the `sklearn.datasets` module. The parameters used for generating the dataset are as follows:

- **Number of Samples (n_samples)**: 10,000
- **Number of Features (n_features)**: 10
- **Number of Informative Features (n_informative)**: 8
- **Number of Redundant Features (n_redundant)**: 2
- **Number of Classes (n_classes)**: 2
- **Number of Clusters per Class (n_clusters_per_class)**: 2
- **Class Separation (class_sep)**: 1.5
- **Random State (random_state)**: 42

The `make_classification` function generates a synthetic dataset that is particularly useful for supervised learning problems, especially for classification tasks. The dataset contains:

- **Informative Features**: Features that are directly related to the class labels.
- **Redundant Features**: Features that are linear combinations of the informative features, adding complexity to the dataset.

Given the specified parameters, the dataset is expected to have two well-separated classes, making it a suitable candidate for applying machine learning algorithms.

2 Application of Trustworthiness

After generating the dataset, the trustworthiness metric was applied to evaluate the quality of a lower-dimensional embedding of the dataset. This metric assesses how well the local structure of the high-dimensional data is preserved in the lower-dimensional space.

2.1 Mathematical Aspects of Trustworthiness

The trustworthiness metric $T(k)$ is defined for a given number of neighbors k. It quantifies how many of the nearest neighbors in the low-dimensional space are also nearest neighbors in the high-dimensional space. The mathematical formulation is as follows:

$$T(k) = 1 - \frac{2}{n \cdot k \cdot (2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in U_k(i)} (r(i,j) - k)$$

where:

- n is the number of data points.
- k is the number of nearest neighbors considered.
- $r(i,j)$ is the rank of point j among the nearest neighbors of point i in the high-dimensional space.
- $U_k(i)$ is the set of points that are in the top k nearest neighbors of point i in the low-dimensional space but not in the top k in the high-dimensional space.

2.2 Trustworthiness Results

The trustworthiness metric was computed for three popular manifold learning techniques: t-SNE, UMAP, and Isomap. The results are as follows:

- **t-SNE Trustworthiness**: 0.9986795376301041
- **UMAP Trustworthiness**: 0.9839488130504404
- **Isomap Trustworthiness**: 0.930199245396317

These results indicate that t-SNE has the highest trustworthiness, meaning it preserves the local structure of the high-dimensional data most effectively, followed by UMAP and Isomap.

3 Conclusion

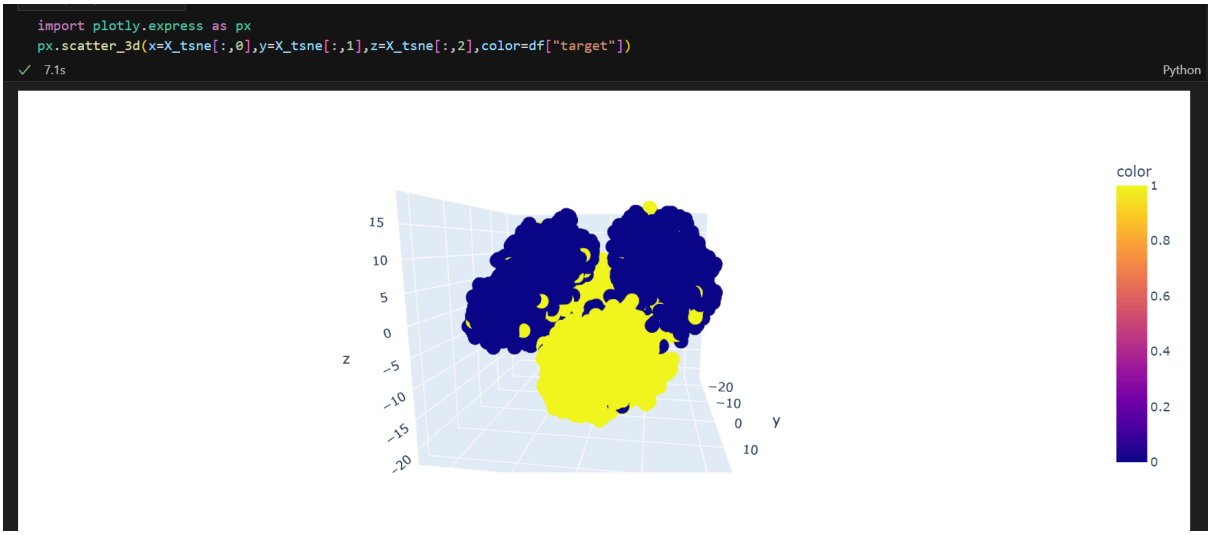
The generated dataset with 10 informative and redundant features provides a solid foundation for classification tasks. By applying the trustworthiness metric, we can evaluate the effectiveness of various dimensionality reduction techniques, ensuring that the lower-dimensional representation of the data maintains the critical relationships found in the original high-dimensional space. This process is essential for understanding the trade-offs involved in reducing data complexity while retaining its intrinsic properties.

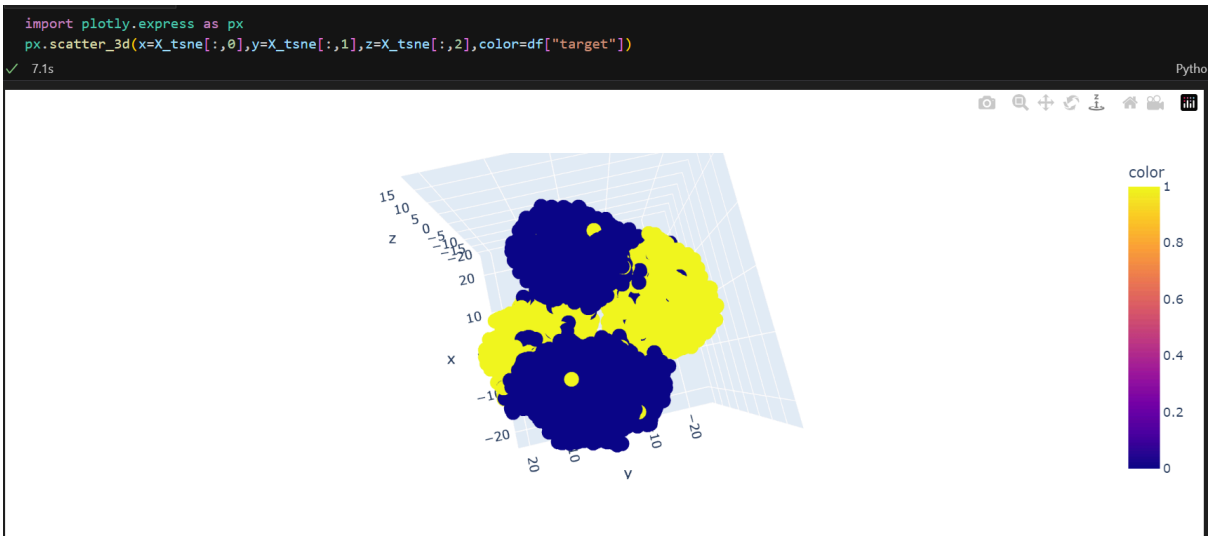
4 Python Code Availability

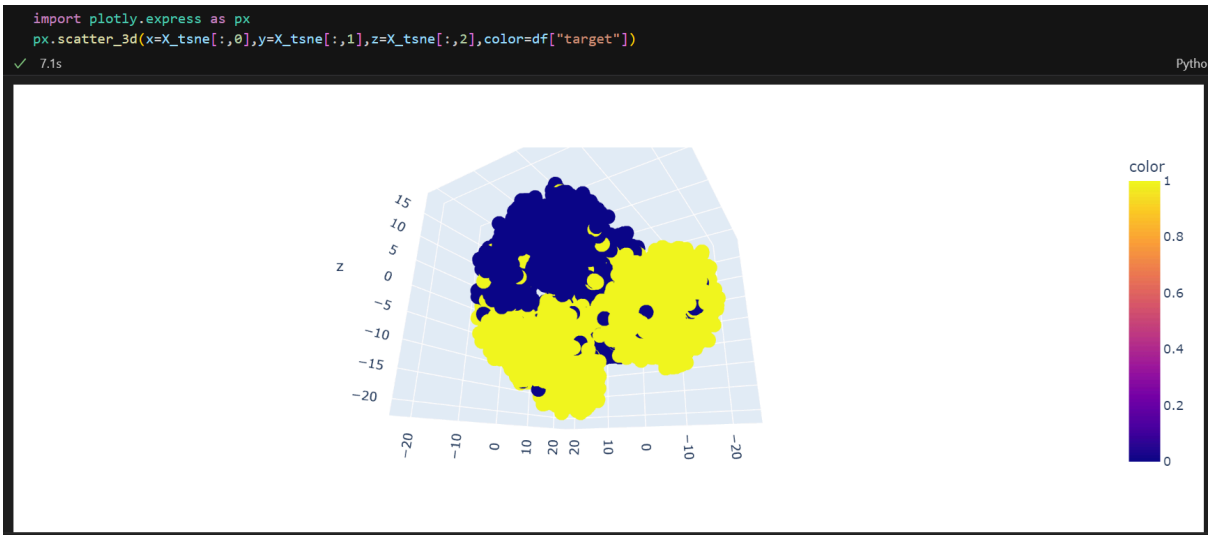
The Python code used for generating the dataset, applying the dimensionality reduction techniques, and calculating the trustworthiness metrics is available in the last portion of this report.

Comparing t-SNE,Isomap and UMAP via visualisations.

1. t-SNE





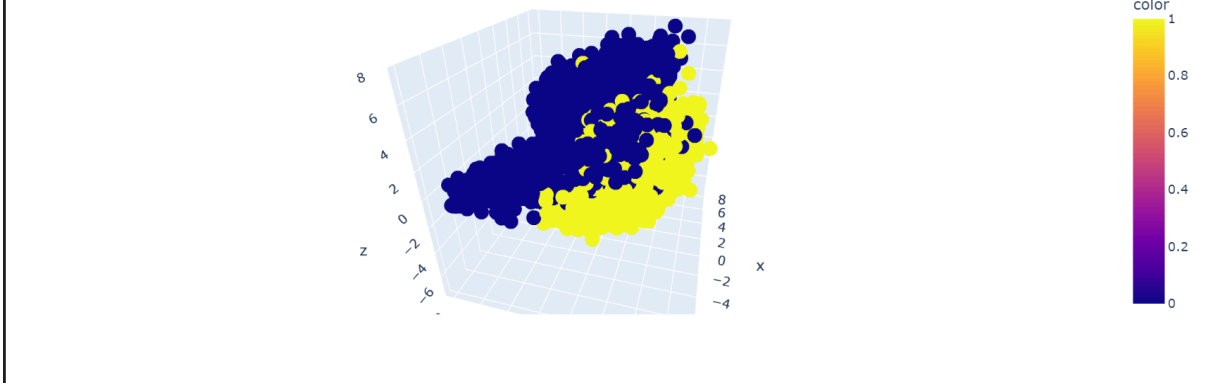


Comparing t-SNE,Isomap and UMAP via visualisations.

2 .Isomap

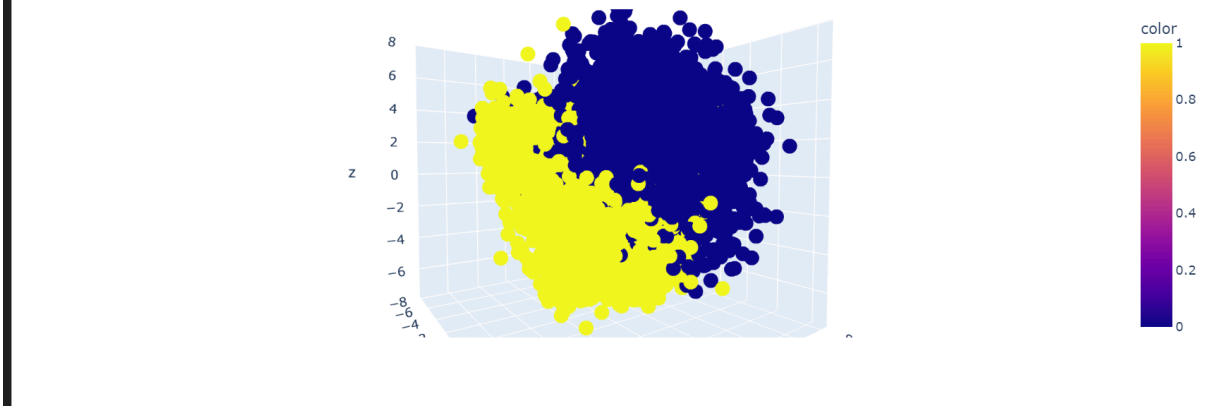
```
import plotly.express as px
px.scatter_3d(x=X_isomap[:,0],y=X_isomap[:,1],z=X_isomap[:,2],color=df["target"])
```

✓ 0.5s



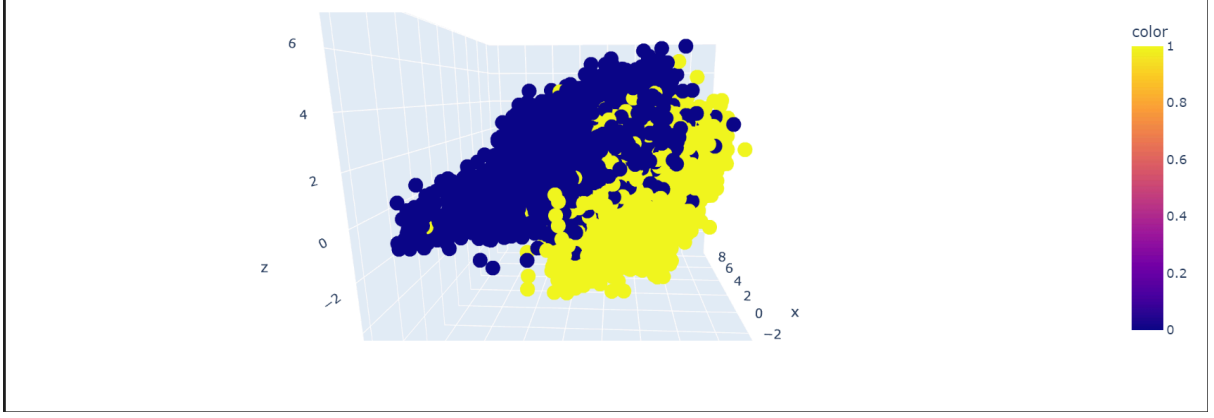

```
import plotly.express as px
px.scatter_3d(x=X_isomap[:,0],y=X_isomap[:,1],z=X_isomap[:,2],color=df["target"])
```

✓ 0.5s




```
import plotly.express as px
px.scatter_3d(x=X_isomap[:,0],y=X_isomap[:,1],z=X_isomap[:,2],color=df["target"])
```

✓ 0.5s

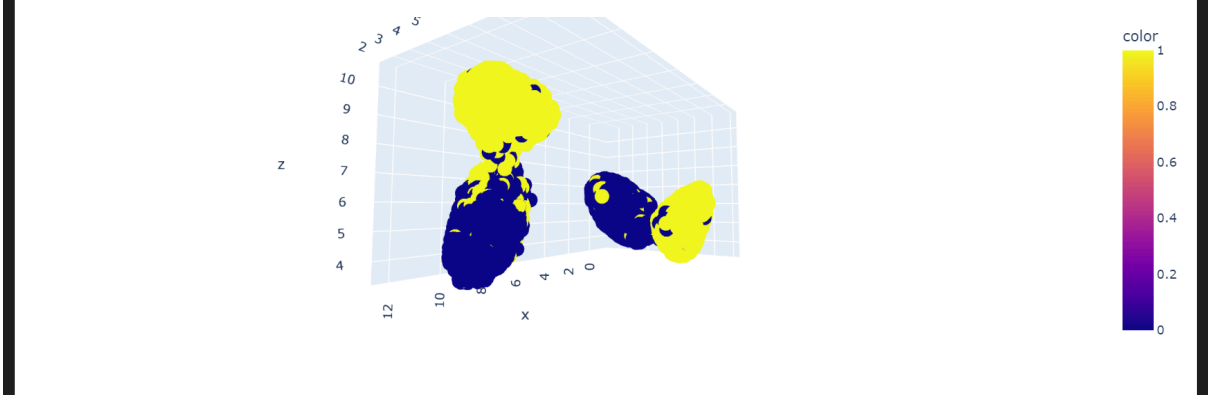


Comparing t-SNE,Isomap and UMAP via visualisations.

3. UMAP

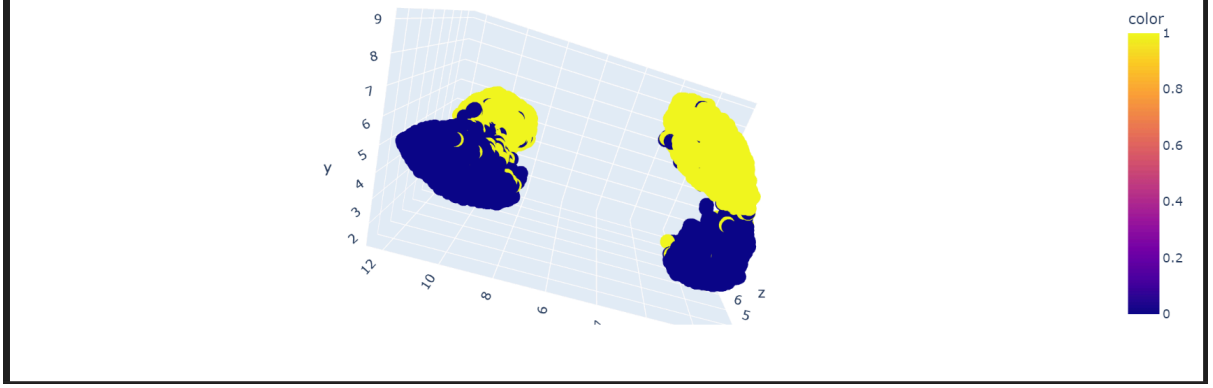
```
import plotly.express as px
px.scatter_3d(x=X_umap[:,0],y=X_umap[:,1],z=X_umap[:,2],color=df["target"])
```

✓ 0.6s Python



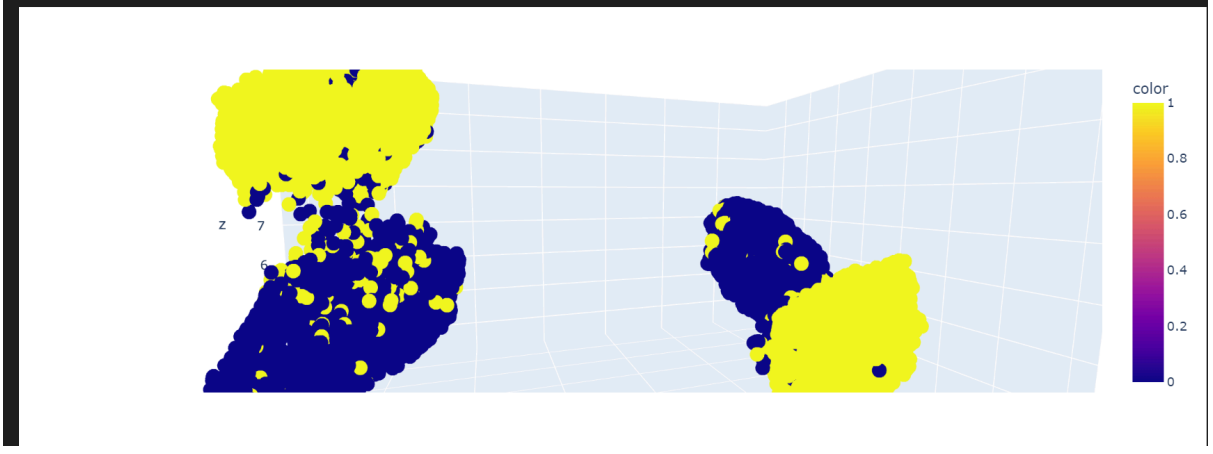

```
import plotly.express as px
px.scatter_3d(x=X_umap[:,0],y=X_umap[:,1],z=X_umap[:,2],color=df["target"])
```

✓ 0.6s Pytho




```
import plotly.express as px
px.scatter_3d(x=X_umap[:,0],y=X_umap[:,1],z=X_umap[:,2],color=df["target"])
```

✓ 0.6s Pytho



In [ ]:
```
!pip install numpy
!pip install matplotlib
!pip install scikit-learn
!pip install umap-learn
!pip install pandas
```

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.manifold import TSNE,Isomap
from umap import UMAP
import pandas as pd
from sklearn.metrics import pairwise_distances
```

```
c:\Users\lucius seneca\AppData\Local\Programs\Python\Python310\lib\site-p
ackages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/u
ser_install.html (https://ipywidgets.readthedocs.io/en/stable/user_instal
l.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
from sklearn.preprocessing import StandardScaler
```

In [3]:
```


n_samples = 10000
n_features = 10
n_informative = 8
n_redundant = 2
n_classes = 2

X, y = make_classification(
    n_samples=n_samples,
    n_features=n_features,
    n_classes=n_classes,
    n_informative=n_informative,
    n_redundant=n_redundant,
    n_clusters_per_class=2,
    class_sep=1.5,
    random_state=42
)
```

In [4]:
```
feature_names = [f'feature_{i+1}' for i in range(n_features)]


df_features = pd.DataFrame(X, columns=feature_names)


df = df_features.copy()
df['target'] = y
df
```

Out[4]:

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8
0	-2.482531	0.525135	-0.598951	-2.410959	-0.990930	-3.112120	-2.840507	-0.572464
1	-2.842556	-0.564530	0.096068	-0.815907	3.120619	-0.256869	-5.424947	0.809551
2	3.080255	1.789750	1.771519	2.924127	-1.811381	-1.350302	2.976814	-0.158628
3	-0.647513	-3.790428	2.014160	-0.585143	1.626927	-5.255379	0.320723	3.078251
4	-0.731565	0.873713	1.726333	2.005599	-1.218096	0.728442	3.282940	-2.123311
...	...	...	...	...	...	...	...	...
9995	-1.999463	-2.826460	-0.703037	2.727000	-2.420999	-0.465184	0.605673	-0.721480
9996	3.959000	1.900724	1.585159	2.672286	-4.300224	-3.274783	1.397944	-3.899414
9997	-3.021406	-2.273686	4.344588	-1.683138	3.043479	-3.205070	-2.935192	4.993852
9998	0.969112	3.597363	3.037797	0.119398	0.094230	-1.240834	0.838637	-2.383345
9999	2.283326	4.032966	3.894216	3.038547	-0.165755	0.285279	3.264131	-0.419280

10000 rows × 11 columns

In [5]:
```
df["target"].value_counts()
```

Out[5]:
```
target
0    5001
1    4999
Name: count, dtype: int64
```

In [6]:
```
# Standardizing the feature
scaler=StandardScaler()

X_scaled=scaler.fit_transform(X)
```

Applying t-SNE

In [7]:
```
tsne=TSNE(n_components=3,random_state=42,n_jobs=-1)
X_tsne=tsne.fit_transform(X_scaled)
```

Applying UMAP

In [8]:
```
umap=UMAP(n_components=3,random_state=42,n_jobs=-1)
X_umap=umap.fit_transform(X_scaled)
```

```
c:\Users\lucius seneca\AppData\Local\Programs\Python\Python310\lib\site-p
ackages\umap\umap_.py:1945: UserWarning: n_jobs value 1 overridden to 1 b
y setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_sta
te. Use no seed for parallelism.")
```

Applying Isomap

In [9]:
```
isomap=Isomap(n_components=3,n_jobs=-1)
X_isomap=isomap.fit_transform(X_scaled)
```

Comparing t-SNE,Isomap,and UMAP using Sklearn trustworthiness

In [10]:
```
from sklearn.manifold import trustworthiness
print("t-SNE truthworthiness",trustworthiness(X_scaled,X_tsne))
print("UMAP truthworthiness",trustworthiness(X_scaled,X_umap))
print("Isomap truthworthiness",trustworthiness(X_scaled,X_isomap))
```

```
t-SNE truthworthiness 0.9986795376301041
UMAP truthworthiness 0.9839488130504404
Isomap truthworthiness 0.930199245396317
```

In [11]:
```
X_tsne[:,0]
```

Out[11]:
```
array([ -8.667209, -18.230059,  16.932846, ..., -17.157583,  13.028165,
        16.258146], dtype=float32)
```

In [12]:
```
import plotly.express as px
px.scatter_3d(x=X_tsne[:,0],y=X_tsne[:,1],z=X_tsne[:,2],color=df["target"])
```

In [13]:
```
import plotly.express as px
px.scatter_3d(x=X_umap[:,0],y=X_umap[:,1],z=X_umap[:,2],color=df["target"])
```

In [14]:
```
import plotly.express as px
px.scatter_3d(x=X_isomap[:,0],y=X_isomap[:,1],z=X_isomap[:,2],color=df["tar
```

In [ ]:
```
```