Evaluation and Comparison of Dimensionality Reduction Techniques

Authors: Milos Vukadinovic, Ricardo Parada, Sanjit Dandapanthula

Professor : Rishi Sonthalia

University of California Los Angeles

December 5, 2022

Abstract

Dimensionality reduction algorithms aim to project high dimensional data to low dimensional space and preserve the structure of the data. However, it is not clear if the performance of algorithms is dataset-dependent. That is why we tested different techniques on different datasets, including the non-euclidean ones, and inspected what happens to the performance as the embedding dimension increases. We evaluated the performance of diffusion maps, UMAP, and ISOMAP using six metrics: stress, strain, classification accuracy, instability, visual quality, and running time. The three main results of our project are the following: We concluded that UMAP has the best performance on all metrics and works well on non-euclidean datasets. We reproduce technique comparison results from McInnes et al. [MHM18] and MDS-as-dimension-increases results from Sonthalia et al. [SVBRG21]. Finally, we gained insights into the performance of UMAP and ISOMAP as the embedding dimension increases and observed that preprocessing the similarity matrix with a lower bound algorithm improves the accuracy on some datasets. we UMAP was a clear winner in all metrics and was the only one whose classification accuracy did not decrease on non-euclidean datasets as the dimension increased.

Contents

Introduction

The curse of dimensionality is a notable problem in fields such as numerical analysis, data science, and machine learning where data sets often lie in high dimensional space. For instance, often one would like to visualize data in order to identify underlying patterns. However, this is mostly impossible for data lying beyond three dimensions. In addition, working in higher dimensional spaces often comes at expensive computational costs such as storing an $n \times n$ matrix that contains the pixels of a large and highly detailed image. Multiple dimensionality reduction techniques have been developed to get around working with higher dimensional spaces and thus avoiding such issues. In this project, we discuss some of these dimensionality reduction methods and compare their efficacy against different data sets via quantitative and qualitative metrics.

We will be focusing on three main techniques: Diffusion Maps, Uniform Manifold Approximation and Projection (UMAP), and Isometric Maps (ISOMAP). The procedures of these algorithms are similar in structure and philosophy. In each algorithm, we create a graph out of the given data points. Then, we strategically assign a weight to each edge. Lastly, we use linear algebra techniques to find low-dimensional representations of each point that preserve certain characteristics of the overall data set that we wish to study in further detail. While the steps are similar for each algorithm, the implementations vary. For instance, the weights assigned to each edge of the graphs will be different depending on whether we use diffusion maps (which assign probabilistic weight to each edge) or UMAP (which assigns weights using a k-neighbor scheme). We will go into further detail on implementations in later sections.

To measure the efficacy of these functions, we made careful choices of datasets and metrics. For the datasets, we chose a mixture of toy data sets and real-world data sets. The toy data sets (the S-curve and the Swiss Roll) will serve as a frame of reference for how each of the methods should behave. We will then analyze the application of each algorithm on the real-world data sets and determine their effectiveness on each set. The metrics will serve to give us a numerical evaluation of how well each algorithm performs on the data set. For instance, we will examine the KNN accuracy of each algorithm in order to determine how well distances are preserved when we shift from a high dimensional space to a lower one. We will specify how these metrics give us such measures in later sections.

Methods

2.1 Datasets

2.1.1 S-curve

The first of the two toy data sets that we selected was the S-curve from the scikit-learn module in Python. The idea behind using the S-curve was to get a sense of how each of the methods behaved with a basic low-dimensional (in this case $\mathbb{R}^3$) example of of data. Moreover, the S-cure can be viewed as a two-dimensional manifold embedded in $\mathbb{R}^3$. Hence, for the chosen methods that seek to find underlying manifolds of high dimensional data sets and project to lower dimensions, the S-curve serves as a reference for visualizing how these techniques act on nonlinear data.

2.1.2 Swiss Roll

The second of the two toy data sets is the Swiss Roll data set, also imported from the scikit-learn module in Python. This is a classically used data set in literature to test nonlinear dimensionality reduction techniques. Like the S-curve, the data points of the Swiss roll lie on a two-dimensional manifold embedded in $\mathbb{R}^3$. As with the S-curve, the Swiss Roll was chosen to compare the accuracy of the visual quality for the dimensionality reduction techniques with those found in the literature. In addition, we added small noisy data points to the Swiss Roll in order to investigate if there were any dramatic changes in the embeddings from each method.

2.1.3 MNIST

The first real-world data set we used to evaluate the dimensional reduction methods is the MNIST data set. The MNIST data set is a collection of $70,000$ handwritten digits from 0 through 9, each stored in a 28×28 pixel image. In Python, each digit is stored as a 728-dimensional vector. Because of this high dimensional setting, we used a 20% sample of the data set so that the methods we used would have a faster runtime while still producing significant results. In particular, the method of diffusion maps implemented in the pydiffmap library does not appear to be optimized to handle large data sets efficiently. Hence, reducing the number of MNIST data points is enough to observe how diffusion maps perform on the MNIST data set.

2.1.4 COIL20/COIL100

Similar to MNIST, the COIL-20 and COIL-100 data sets are a collection of 128×128 pixel images. The COIL-20 data set contains gray-scale images, while the COIL-100 data set contains colored images. Both of these data sets have the option of being imported in processed and unprocessed versions. For our purposes, we used the processed version where the background of the images is greatly reduced. This made it easier to analyze the visual quality of the embedding techniques and helped increase the accuracy in other metrics as we will see in the results.

2.2 Non-Euclidean Datasets

We want to make our distance matrices non-Euclidean so that we can witness the decrease in classification accuracy as dimension. There are three methods that we implemented to do this [SVBRG21]:

1. Perturb the Euclidean distance matrix with another matrix having zeroes on the diagonal and Gaussian noise on the off-diagonals.
2. Compute the k-nearest neighbor graph and use the distance on this graph (the distance used in ISOMAP).
3. Imagine each vector has missing entries and compute the distance between the entries two vectors have in common.

2.3 Algorithms

2.3.1 Diffusion Maps

Diffusion maps [JdlP08] are a nonlinear dimensionality reduction technique that reveal underlying geometric structures in a data set via a time-dependent diffusion process that relies on the intuition behind random walks. Given a high dimensional data set $\{x_i\}_{i=1}^N$ where $x_i \in \mathbb{R}^n$, we create a graph out of the points in the data set. The idea is to assign each edge connecting two points in the graph a probability $p(x_i, x_j)$ based on the likelihood that we can traverse from one point x_i to another point x_j. The closer two points are to one another, the higher the probability that we can travel between them. These probabilities $p(x_i, x_j)$ are also called the *connectivity* between points x_i and x_j.

To obtain a local measure of similarity within a neighborhood, we define the *diffusion kernel* $k(x, y)$. In the diffusion map algorithm implemented in Python, we take the Gaussian kernel

$$k(x, y) = \exp\left(-\frac{|x-y|^2}{\epsilon}\right)$$

where a carefully chosen ϵ gives us the size of a neighborhood around a given point where the distances in the desired metric (e.g. Euclidean distance) can be accurately measured. Then for every point in the graph, we apply the diffusion kernel $k(x_i, x_j)$ and create a kernel matrix K. By normalizing the rows of the kernel matrix via the relation

$$p(x, y) = \frac{1}{d_X} k(x, y)$$

where d_X is the row sums of K, we now have a *diffusion matrix*. By writing the connectivites in matrix form P where P_{ij} is the probability of going from point x_i to x_j and normalizing using d_X, we now have a transition matrix. By diagonalizing P and sort the eigenvalues in descending order, then by taking the $d < n$ dominant eigenvectors, we have reduced the dimension in which we are working with by tracing the lower-dimensional geometric structure of the data.

2.3.2 UMAP

UMAP [MHM18] is an algorithm for dimensionality reduction theoretically derived by approximating the manifold that data lies on and constructing fuzzy simplicial sets.UMAP is efficient, scalable and it doesn't have a restriction on the number of dimensions of the input data.

UMAP we can also be explained in terms of weighted graphs. Namely, the two steps of the algorithm are:

1. Construct a weighted k-neighbour graph
2. Find a low dimensional representation which preserves properties you care about

Given the input data $X = \{x_1, \dots, x_N\}$ and a metric $d : X \times X \to \mathbb{R}_{\geq 0}$ we compute k nearest neighbours for each X_i using neighbor descent algorithm. Then for each x_i we define ρ_i and σ_i. ρ_i is a distance to the nearest neighbour.

$$\rho_i = min\{d(x_i, x_{i_j}) \mid 1 \leq j \leq k, d(x_i, x_{i_j}) > 0\}$$

and set σ_i to be the value such that

$$\sum_{j=1}^{k} \exp\left(\frac{-max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) = \log_2(k)$$

This normalization helps us deal with the curse of dimensionality. Then we can define a graph $\bar{G} = (V, E, w)$ The vertices V of $\bar{G}$ are simply the set X. We can then form the set of directed edges $E = \{(x_i, x_{i_j}) \mid 1 \leq j \leq k, 1 \leq i \leq N\}$ and define the weight functions w by setting

$$w((x_i, x_{i_j})) = \exp\left(\frac{-max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right)$$

Next, to obtain an undirected graph, take A - the weighted adjacency matrix of $\bar{G}$ then set

$$B = A + A^T - A \circ A^T$$

B is an undirected adjacency matrix where B_{ij} represents the probability that an edge between x_i and x_j exists.

This summarizes the firs step. Next, UMAP uses a force directed graph layout algorithm. The goal is to find coordinates y_i in low dimensional space. We first initialize y_i using spectral layout. Then we use a set of attractive forces along edges and a set of repulsive forces among vertices. Convergence to a local minima is guaranteed.

Attractive force is defined by:

$$\frac{-2ab||y_i - y_j||_2^{2(b-1)}}{1 + ||y_i - y_j||_2^2} w((x_i, x_j))(y_i - y_j)$$

where a and b are hyperparameters

Repulsive forces are given by

$$\frac{2b}{(\epsilon + ||y_i - y_j||_2^2)(1 + a||y_i - y_j||_2^{2b})}(1 - w((x_i, x_j)))(y_i - y_j)$$

where ϵ is a small number to prevent division by zero

2.3.3 ISOMAP

ISOMAP [TSL00] is an algorithm for multi-dimensional scaling with respect to an approximation of "geodesic" distance. Essentially, we want to estimate the geometry of the manifold containing our data using information from each point's neighbors. We can also use this information to compute a lower-dimensional embedding. The algorithm works as follows:

1. Find the k nearest neighbors for any given point.
2. Create a graph G where two points x_i and x_j are connected if they are k-nearest neighbors and the edge length is $\|x_i - x_j\|_2$.

3. Define $d(x_i, x_j)$ for all i, j to be the shortest path between x_i and x_j in G (e.g., this can be computed using Dijkstra's algorithm).
4. Use MDS or another embedding algorithm to embed $\{x_i\}$ into $\mathbb{R}^k$ using the metric d.

Using this algorithm attempts to approximate the "geodesic distance" between any two points and this is reflected in our embedding. As a result, the embedding is representative of some underlying topological properties of the data we're given.

2.3.4 Lower Bound Algorihm

Lower Bound Algorithm [SVBRG21] is not a dimensionality reduction technique but rather a preprocessing step that improves the accuracy of MDS as dimension increases. Namely, because cMDS is minimizing strain objective, the error $err = ||D_{cmds} - D||_F$ will start to decrease at some value of r (embedding dimension) 2.3.4. That is because, the excess in the trace appears when D is not euclidean. To solve this problem Lower Bound Algorithm projects D onto $\mathcal{K}(r)$ which is the space of symmetric, trace 0 matrices A, such that $\hat{A}$ is negative semi-definite of rank at most r. Note that $\hat{A}$ is defined from Hayden and Wells EDM characterization ($QAQ = \begin{bmatrix} \hat{A} & f(A) \\ f(A)^T & \epsilon(A) \end{bmatrix}$) The algorithm steps are given in [SVBRG21] as **Algorithm 2**. We implemented it in python using numpy.

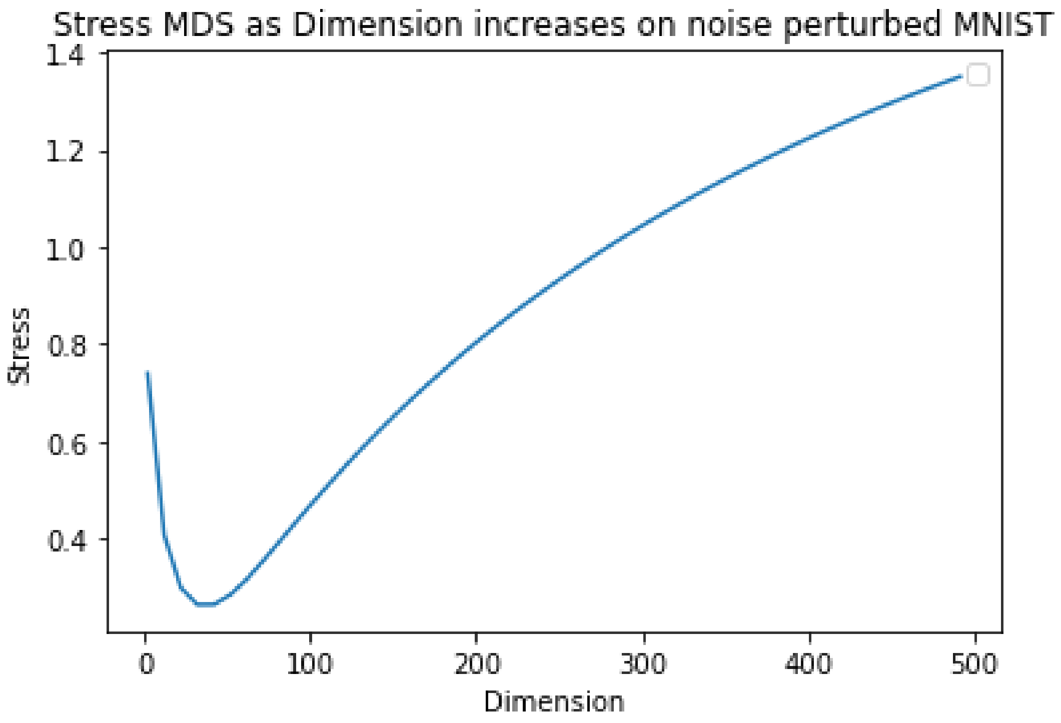


Figure 2.1: Stress of cMDS on MNIST as dimension increases

2.4 Metrics

2.4.1 KNN Accuracy

KNN accuracy is testing how well the embedding has preserved the important local structure of the dataset. It is computed as follows. We split the embeddings dimensionality reduction algorithm produced into train and test subsets, we train a KNN classifier on the train subset and compute classification accuracy given by $acc = \frac{TP+TN}{TP+FN+TN+FP}$ on the test subset. KNN accuracy is the mean classification accuracy over 10-folds of cross-validation.

2.4.2 NN Accuracy

Neural Net accuracy is computed the same way as KNN accuracy, but instead of using a KNN classifier we use a Neural Network. We build the same NN architecture as [SVBRG21], namely with three linear layers, each one having 100 neurons, each using ReLU activation function. Output has 10 values (corressponding to 10 mnist digits), and log softmax is applied on it.

2.4.3 Instability

Under sub-sampling of data embeddings that dimensionality reduction methods produce will be different. Additionally, some dimensionality reduction techniques will produce different embeddings from run to run because of their stochastic component. This is a potential concern in some use cases and that is why it is important to evaluate the robustness of the embedding technique.

To do that we adapted the instability metric [MHM18]. Essentially, instability computes the procrustes distance between two embeddings: embedding when we first subsample and then apply reduction and the embedding when we first apply reduction and then subsample.

More formally, let Y be the original dataset with n points, $F(\cdot)$ be the reduction algorithm, $D_p(\cdot,\cdot)$ procrustes distance and $k = 0.2n$. Then

$$Instability = D_p(F(Y[1:k]), F(Y)[1:k])$$

2.4.4 Kruskal's Stress

The Kruskal stress objective is defined by $\text{Stress}_D(\tilde{D}) = \|D - \tilde{D}\|_F^2$, where D is our original distance matrix and $\tilde{D}$ is the matrix with our embedding distances. The intuition for this metric comes from attracting and repulsing energies from physics [CB13]. Namely, we have for some constant c:

$$\text{Stress}_D(\tilde{D}) = \|D - \tilde{D}\|_F^2 = \sum_{i,j}(d_{ij} - \tilde{d}_{ij})^2 = \sum_{i,j}(\tilde{d}_{ij}^2 - 2d_{ij}\tilde{d}_{ij}) + c.$$

Therefore, minimizing stress is equivalent to minimizing $\sum_{i,j}(\tilde{d}_{ij}^2 - 2d_{ij}\tilde{d}_{ij})$. Then, the $\tilde{d}_{ij}^2$ terms are the "attracting" force since it provides an incentive to minimize distances between the points in the embedding. In addition, this term increases with the square of distance, which is reminiscent of the inverse square law for magnetic force with respect to distance between two magnetic poles. On the other hand, the $-2d_{ij}\tilde{d}_{ij}$ terms constitute a "repulsive" force since they are minimized by $\tilde{d}_{ij} = +\infty$. The idea is that the attractive and repulsive forces balance each other out at the minimizer $d_{ij} = D_{ij}$.

2.4.5 Strain

The strain objective is defined by $\text{Strain}_D(X) = \|D - XX^T\|_F^2$, where X is the matrix containing our embedded points as rows and D is our original distance matrix. We showed in class that classical MDS is the embedding technique which minimizes the strain.

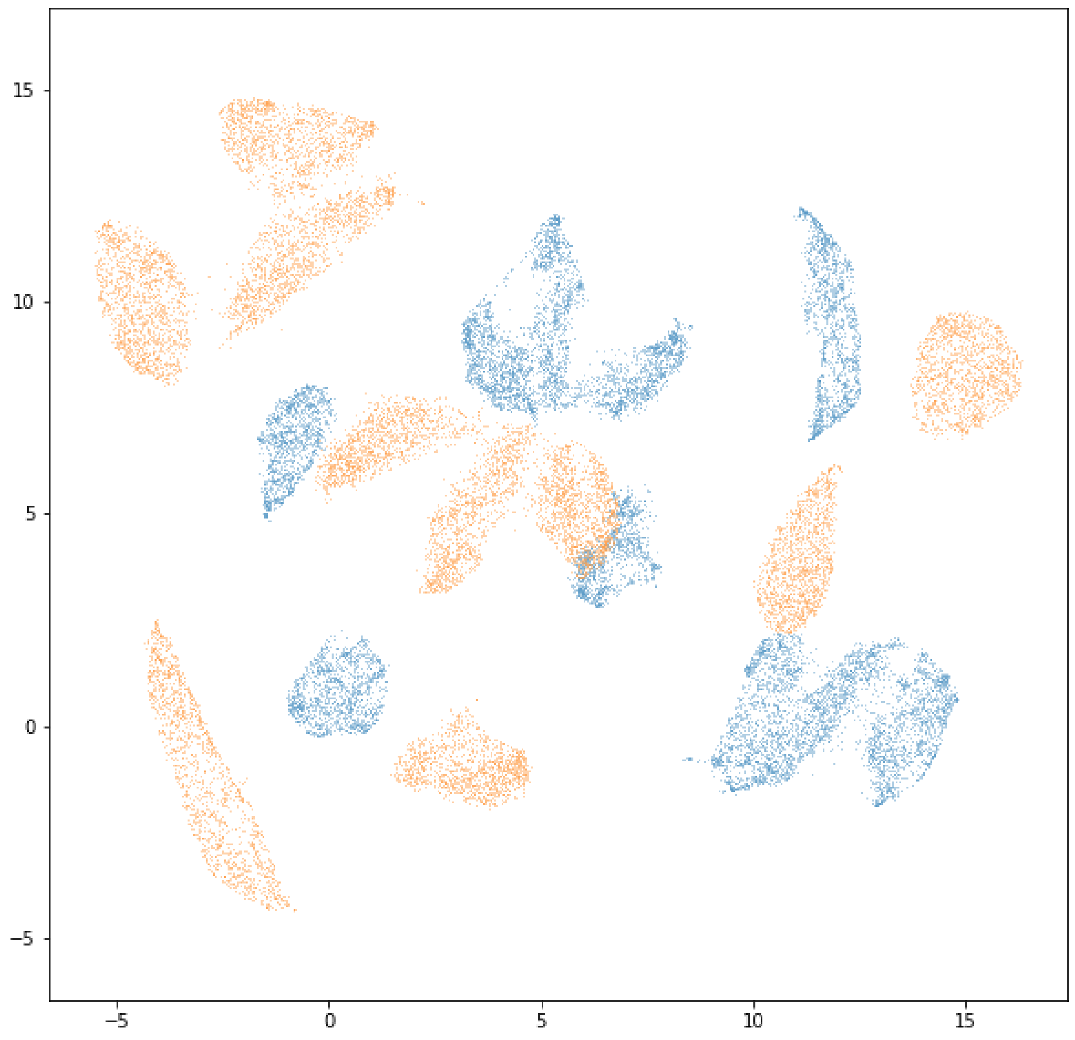


Figure 2.2: Blue points represent dataset that was first subsamples and then reduced, Orange points are the dataset that was first reduces and then subsamples.

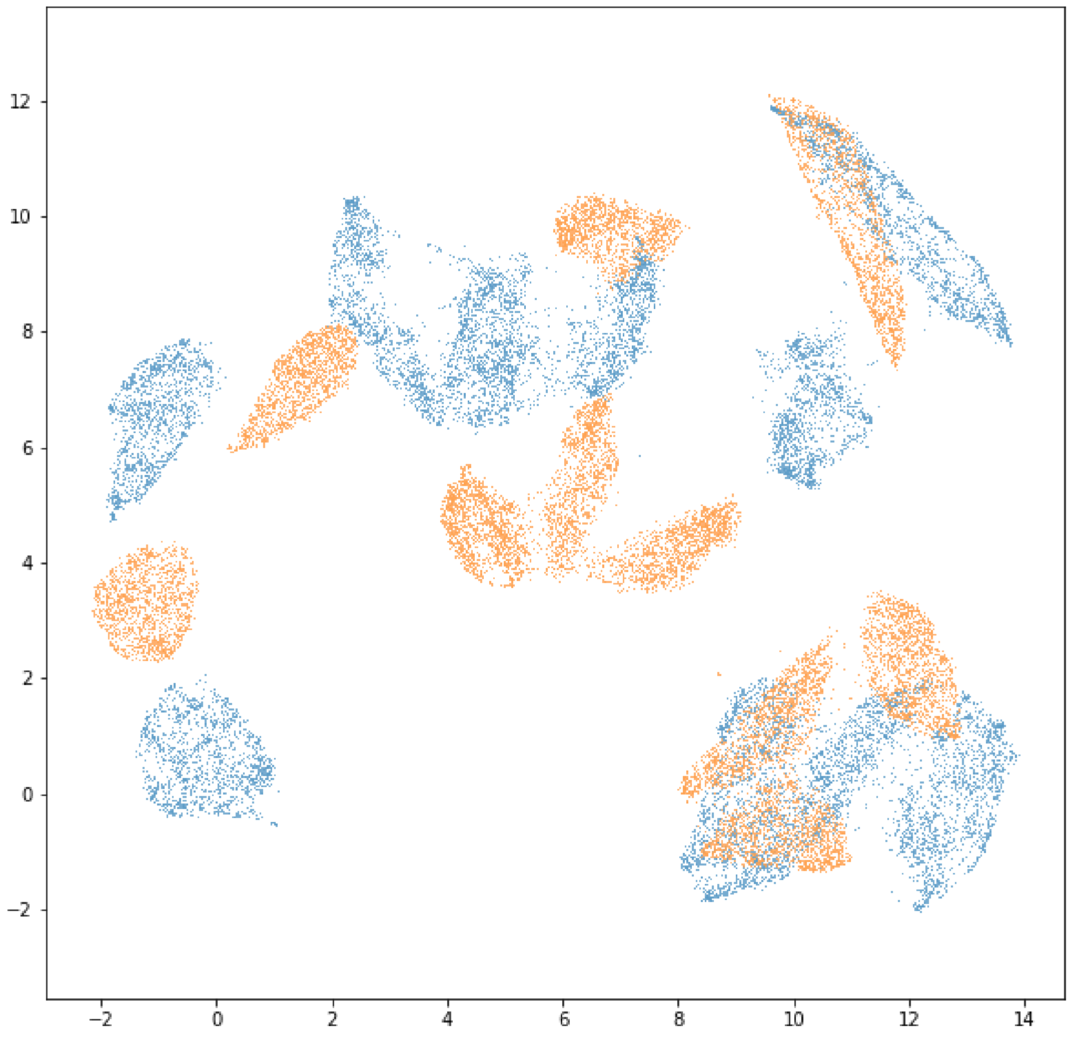


Figure 2.3: Procrustes aligned version of the previous plot

Results

3.1 Performance Metrics

Before introducing results for a general n-dimensional space, we first consider the performance of each of the techniques in the case $n = 2$. As stated in Section 2.4, we will be looking into the KNN accuracy, instability, stress, and strain objectives. Note that we will omit the runtime results in Table 3.7 since these are self-explanatory.

We begin by looking at the visual quality of each dimensonality reduction method as shown in Table 3.1. As we discussed when introducing the data sets, we expect that each method works perfectly (or nearly perfect) on the toy data sets. Indeed, when we consider the embeddings of the S-curve and the Swiss Roll, we see that each of the methods provides a proper visual of the data points in $\mathbb{R}^2$. While the shapes differ, we see that we can distinguish the transition from the points on one end of the S-curve (respectively, the Swiss Roll) to the other end.

We begin to see a difference in the visual quality of each method when we consider the real world data sets. For the MNIST data set, we see that UMAP has the best visual quality of the three methods. There is not much we can gather from the diffusion map embedding, as all the points are clustered close to the origin. On the other hand, ISOMAP appears to do a slightly better job at clustering some of the points but the overlap causes difficulty in distinguishing the data points. Similarly, we see that the UMAP method works well with COIL-20. Diffusion maps work better on COIL-20 than on MNIST, but we observe that there is still some overlap in the data points and hence does not work as well as UMAP. A similar behavior occurs with ISOMAP. We note that the visual results of COIL-100 are similar to that of COIL-20 and hence we chose to omit the these results in Table 3.1.

Next, we turn our attention to the KNN accuracy and instability of each model. We observe from Table 3.3 that UMAP outperforms diffusion maps and ISOMAP. In Table 3.4, we see that the UMAP method has the smallest values and hence the most stable of the three algorithms. However, we note that for the diffusion map and ISOMAP techniques, the values have almost no deviations (i.e. ± 0). Unlike UMAP, diffusion maps and ISOMAP have no stochastic processes involved in the algorithms. Instead, both of these methods use eigenvectors and eigenvalues to determine the embedding coordinates. Hence, the eigenvectors are essentially the same up to scaling and thus the instability values remain same.

Lastly, we observe the values of the strain and stress objectives on each technique. From Tables 3.5 and 3.6, we see once again that UMAP performs the best on the 20% of the MNIST dataset. In both cases, we can confirm that these results are accurate from the visual quality. We see that the MNIST embedding distance is much further from their actual distances in a much higher dimension.

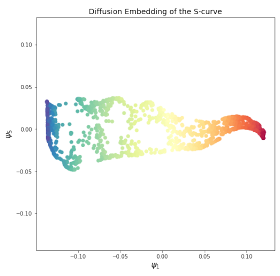
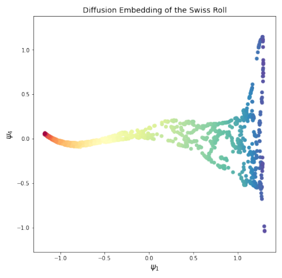
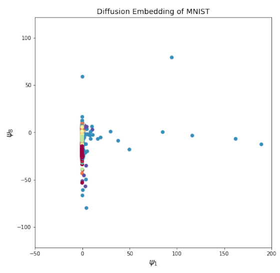
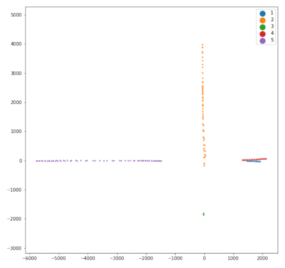
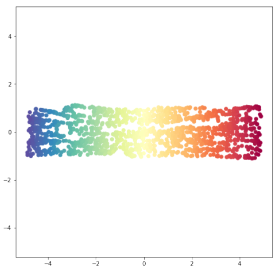
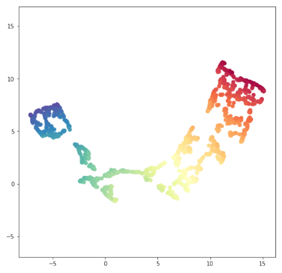
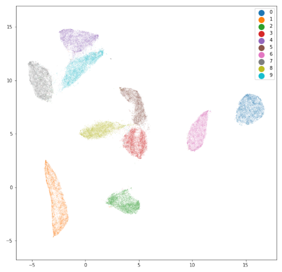
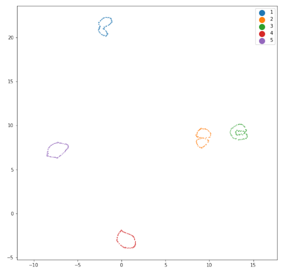
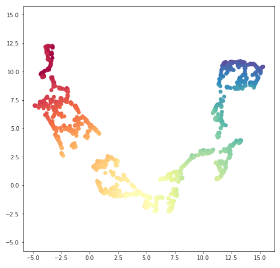
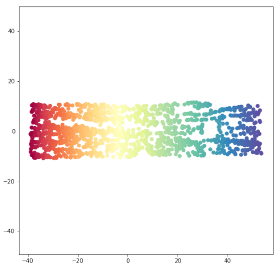
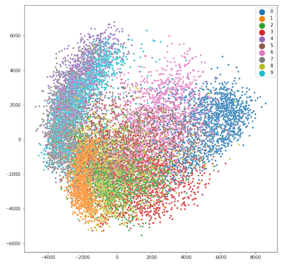
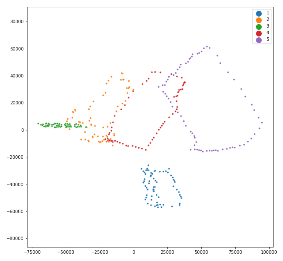
Nr.	S Curve	Swiss roll	MNIST	Coil20
Dif				
Umap				
Iso				

Table 3.1: Visual Quality of Different Embeddings

	20% MNIST	**Coil20**	**20% Coil100**
Number of samples	14000	360	1440
Shape of a sample	28×28	448×416	$128 \times 128 \times 3$

Table 3.2: Datasets

3.2 Stress and Strain as the dimension increases

Using the "noise" paradigm to make our distances non-Euclidean, we computed the stress and strain as the dimension of embedding increases. We get the characteristic U-shape for the stress curve as dimension increases for MDS:

3.3 Accuracy as the dimension increases

We computed KNN accuracy for MDS, UMAP, and ISOMAP on noise, missing and isom modified datasets 3.9. We can see that for MDS, as the embedding dimension increases, KNN

$\circ$	**20% MNIST**	**Coil20**	**20 % Coil100**
DifMaps	0.804 ± 0.007	0.7569 ± 0.092	0.7181 ± 0.097
UMAP	0.937 ± 0.004	1.0 ± 0	0.78 ± 0.020
ISOMAP	0.524 ± 0.007	0.679 ± 0.051	0.439 ± 0.016

Table 3.3: KNN accuracy

	MNIST	**Coil20**	**Coil100**
DifMaps	1.54 ± 0	0.538	0.731 ± 0
UMAP	0.19 ± 0.0	0.343 ± 0.100	0.211 ± 0.032
ISOMAP	0.325	0.762	0.402 ± 0

Table 3.4: Instability

Model	Strain
Diffusion Map	461554048.699
UMAP	6551600.872
ISOMAP	89658596476467.720

Table 3.5: Strain of various models we tested on 20% of the MNIST dataset

Model	Kruskal's stress
Diffusion Map	6914623.798
UMAP	6860058.433
ISOMAP	8631254.644

Table 3.6: Kruskal's stress of various models we tested on 20% of the MNIST dataset

	20% MNIST	**Coil20**	**20% Coil100**
DifMaps	10	9	8
UMAP	4	3	3
ISOMAP	5	10	8

Table 3.7: Runtimes (in μs)

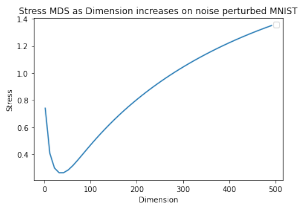
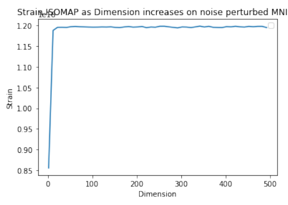
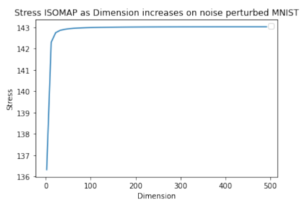
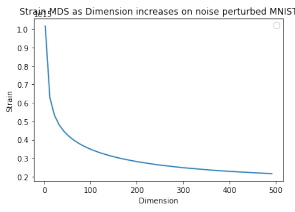
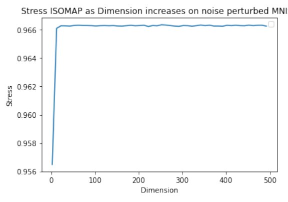
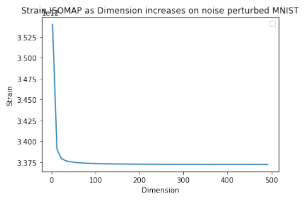
∘	MDS	UMAP	Isomap
Stress			
Strain			

Table 3.8: Kruskal's squared stress and strain as dimension increases on 20% of MNIST

accuracy first increases but then starts decreasing. This is counter-intuitive because as we increase the dimension we do not lose any information from data. The math behind this phenomenon is previously explained in the Lower Bound Algorithm section. However, when we pre-process the similarity matrix with Lower Bound Algorithm, KNN accuracy does not decrease. Next, for UMAP KNN accuracy increases in the range $[0, 20]$ and then is almost constant as we increase the dimension. LB algorithm applied on similarity matrix before feeding it into UMAP doesn't improve accuracy for missing and isom modified dataset, but for the noise modified dataset we see a significant improvement in accuracy. Finally, for the Isomap algorithm, KNN accuracy doesn't decrease on noise modified dataset, but it decreases for missing and isom modified datasets. When we use ISOMAP in conjunction with LB, we see an increase in accuracy only on noise modified dataset.

Because of the possibility that the KNN classifier suffers from the curse of dimensionality itself, we also computed neural network classification accuracy 3.10. The results are similar to the KNN accuracy: LB improves cMDS and UMAP on noise modified MNIST. The only difference that we observed is that accuracy varies more as the dimension increases.

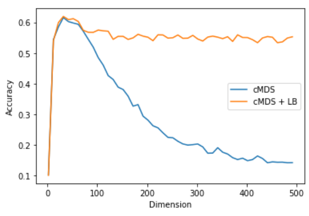
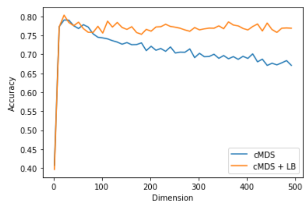
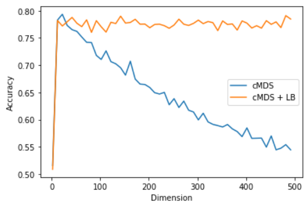
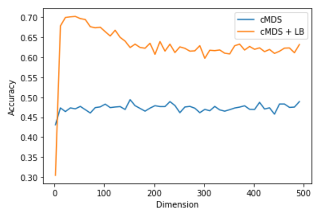
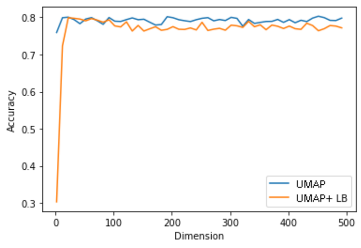
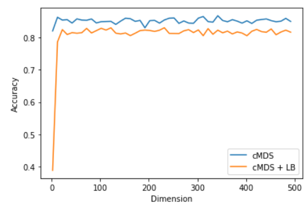
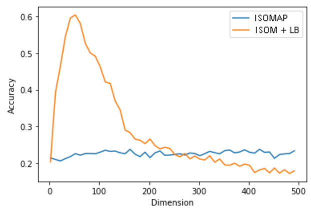
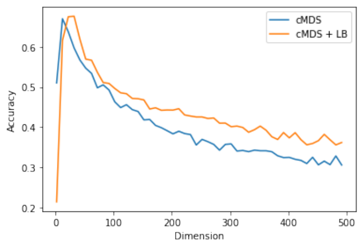
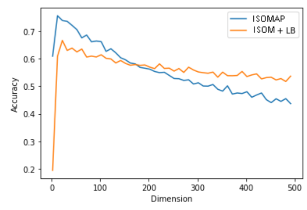
◦	Noise	Missing	Isom
MDS			
UMAP			
Isomap			

Table 3.9: KNN accuracy as dimension increases on MNIST

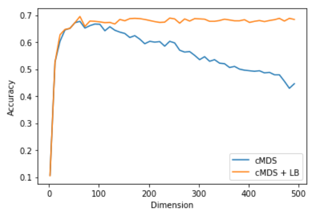
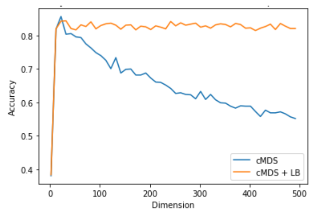
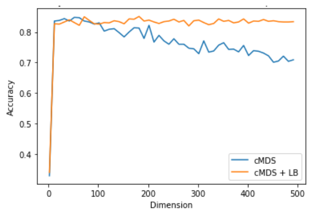
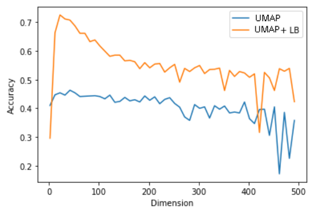
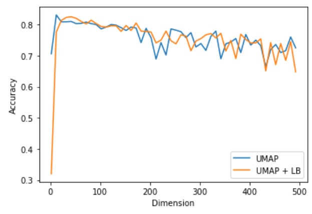
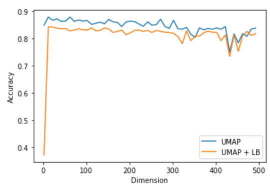
◦	Noise	Missing	Isom
MDS			
UMAP			

Table 3.10: NN classification accuracy as dimension increases on MNIST

Bibliography

[CB13] Lisha Chen and Andreas Buja. Stress functions for nonlinear dimension reduction, proximity analysis, and graph drawing. *Journal of Machine Learning Research*, 14:1145, 2013.

[JdlP08] W. Hereman S.J. van der Walt J. de la Port, B.M. Herbst. An introduction to diffusion maps, 2008.

[MHM18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018.

[SVBRG21] Rishi Sonthalia, Greg Van Buskirk, Benjamin Raichel, and Anna Gilbert. How can classical multidimensional scaling go wrong? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12304–12315. Curran Associates, Inc., 2021.

[TSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, December 2000.