Maximum Variance Unfolding on Disjoint Manifolds

PIC2 - Master in Computer Science and Engineering
Instituto Superior Técnico, Universidade de Lisboa

João André Roque Costa — 99088[*]
joaoarcosta@tecnico.ulisboa.pt

Advisor: Francisco Melo
Co-advisor: João Tomás Brazão Caldeira

Abstract Dimensionality reduction is a field in constant evolution. However, even though there are methods that excel at computing linear relation over datasets, the operation of linearising data points that relate non-linearly between them is a task that was not perfected yet. In this study we assess whether a powerful non-linear dimensionality reduction method, maximum variance unfolding, can be extended to datasets that lie on disjoint manifolds.

Keywords — Dimensionality Reduction, Non-linearity, Embedded Space, Low Dimension, Eigenvalue Decomposition

[*]I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa (https://nape.tecnico.ulisboa.pt/en/apoio-ao-estudante/documentos-importantes/regulamentos-da-universidade-de-lisboa/).

Contents

1 Introduction **3**

2 Literature Review **3**

2.1 Convex Optimisation . . . 3

2.2 Dimensionality Reduction . . . 4

2.2.1 Linear Methods . . . 4

2.2.2 Isomap . . . 7

2.2.3 Locally Linear Embedding (LLE) . . . 7

2.2.4 Laplacian Eigenmaps (LE) . . . 8

2.2.5 Hessian Locally Linear Embedding (H-LLE) . . . 9

2.2.6 Local Tangent Space Analysis (LTSA) . . . 9

2.2.7 t-distributed Stochastic Neighbour Embedding (t-SNE) . . . 10

2.2.8 Kernel PCA (K-PCA) . . . 11

2.2.9 Maximum Variance Unfolding (MVU) . . . 11

3 Related Work **12**

4 Proposed Approach **13**

4.1 Problem Specification . . . 13

4.2 Solution Specification . . . 13

4.3 Methodology . . . 14

4.3.1 Algorithms . . . 14

4.3.2 Datasets . . . 14

4.3.3 Metrics . . . 15

4.3.4 Experimental Setup . . . 16

4.4 Work Schedule . . . 16

Bibliography **17**

A Appendix **19**

1 Introduction

Dimensionality reduction is a vast research field whose premise is to reduce the dimensionality of a dataset in some way that approximates the number of variables used to its intrinsic dimensionality.

Acknowledging that there is the possibility for points to be related in non-linear ways, common approaches that deal with linear relations, like Principal Component Analysis (PCA) [1] and Multidimensional Scaling (MDS), do not perform well over them. To accurately perform non-linear dimensional reduction (NLDR) it is necessary to consider non-linear methods, which can be classified into neighbourhood graph (NG)-based and global (e.g., PCA, autoencoders [2, 3]) methods. The latter do not achieve competitive results when local properties are important to maintain, while methods that make use of NGs allow for control over geometric properties of the data when performing dimensionality reduction.

While NLDR methods based on neighbourhood graphs (NG-NLDR) are the most adequate for finding representations that preserve geometric properties of the data, they are not easily applicable in the common scenarios where data lie on multiple manifolds or are sampled sparsely from the underlying manifold. In these cases, since we usually connect each point to its k-nearest neighbours, the formed NG may consist of multiple connected components. For example, if we imagine two clusters of points that are very distant from each other, it might take a large value of k for them to become connected. However, as we will see, large values of k (i.e., considering large neighbourhoods around each point) reduces NG-NLDR methods' ability to focus on local geometric properties. On the other hand, disconnected NGs result in NG-NLDR procedures that are ill-defined or yield bad results.

To respond to this, there is a present need for methods to be robust to datasets of varying density. In [4–6] are discussed some implementations to fix this problem, from creating connections between disjoint components whenever they are not connected, or reducing a component's points into a single reference one, to then, after NLDR, rebuild the entire dataset from these reference points.

In this study we propose an extension to a valuable dimensionality reduction method, maximum variance unfolding (MVU), in order to address its inapplicability to the common setting of disconnected NGs. We reiterate that these emerge when the data lie on multiple manifolds or when certain regions of the manifold are more sparsely sampled, both of which occur naturally in real datasets.

First, we are describing the process of dimensionality reduction, presenting different methods dedicated to different purposes and also explaining their reasoning.

More specifically, how some linear methods, with some tweaking, can solve non-linear datasets; some non-linear methods can reach good results, but others that reach more precise solutions are limited in their usability on natural data. With the presence of class separation, many methods, mainly graph-based, can't perform well due to the sparsity over areas of lower point density.

After that, we analyse how there are some extensions to the methods that simplify the input data by a smaller dataset and still can perform as well. We also present some extensions that adapt to sparse areas, but on some related methods other them MVU.

2 Literature Review

2.1 Convex Optimisation

Convex Sets Given a set $S \in R^D$, it is considered convex if and only if for any pair of points, all points in the straight line between them are contained in the set. This is a simplification of the following condition:

$$(1-\alpha)x + \alpha y \in S, \tag{1}$$

for all $x, y \in S$ and $\alpha \in [0, 1]$.

Convex Functions By definition, a function f is convex if for any pair of values, x and y, the value of the function is never higher than the line segment between these two points. This can be written with the following mathematical expression:

$$f(\alpha x + (1-\alpha)y) \leq f(\alpha x) + f((1-\alpha)y), \tag{2}$$

for all $x,y \in \text{dom}(f)$ and $\alpha \in [0,1]$.

There are cases where a function, and thus an optimisation problem, might be convex but take no global minimum like the case of the exponential function. On the other hand, a constant function would take infinite global minima.

Optimisation Problem An optimisation problem consists of finding the optimal values of a set of variables x, possibly within a subset $\Omega \in D$ of their domain (which is represented by constraints), such that some function of interest f is either maximized or minimised, depending on the problem. When the purpose of the problem is to pursue a maximization of the optimised function, it shall be denominated utility function, while when it is meant to be minimised it is called cost function. We call "variables of interest" the variables to be changed in order to reach the optimal function value. My suggestion for presenting the general form of an optimisation problem is:

$$\min_{x \in D} \quad f(x) \tag{3}$$

$$\text{s.t.} \quad x \in \Omega, \text{ where } \Omega \subseteq D. \tag{4}$$

Each optimisation problem can be classified considering the nature of the function to optimise and the existence of the constraints. If a problem has any kind of constraint, it is declared as a Constrained Problem. If either the objective function is not convex, or the constraints define a non-convex set, then the whole problem is denoted as non-convex. This is, for a problem to be convex, all the functions to optimise and the constraints have to be convex.

According to the differences in optimisation problem and types of constraints, the optimisation problems can be classified as:

- **Linear Programming (LP)** problems take an affine function as an objective function and linear constraints. They are computationally the simplest to solve, usually solvable using the simplex algorithm or the interior point method.
- **Quadratic Programming (QP)** problems involve the optimisation of a quadratic function, while also being limited to linear constraints. Depending on the convexity of the problem, a simple gradient method may reach the global minimum if it is an unconstrained problem, while the Karush–Kuhn–Tucker conditions may be used to successfully solve constrained ones. On non-convex situations when either the objective function or the constraint's set are non-convex, a gradient method would need the help of a global search extension to then be able to find the global minima.
- **Second Order Cone Programming (SOCP)** problems consist of the optimisation problems constrained by constraints that form a second order cone, these are formulated as $\|\boldsymbol{A}x+b\|_2 \leq c^\top x+d$.
- **Semi-definite Programming (SDP)** problems represent the optimisation problems that are constrained by a linear matrix inequality of the form $x_1\boldsymbol{F}_1 + \ldots + x_n\boldsymbol{F}_n + \boldsymbol{G} \preceq 0$.

2.2 Dimensionality Reduction

Dimensionality reduction is the process of reconstructing a given dataset in a space described by fewer dimensions. This is done by finding a projection from the higher to lower dimensional space, approximating the dimensions of the dataset in analysis to its intrinsic dimensions, the theoretical number of variables necessary to fully represent the data in observation.

2.2.1 Linear Methods

Principal Component Analysis (PCA) The most commonly used is Principal Component Analysis [1] which, by comparing the linear relation between each pair of variables, creates a new set of variables in a linear subspace that retains as much variance as possible. Its simplified procedures are:

Let's consider a data set with n records represented by D variables ($\boldsymbol{X} \in \mathbb{R}^{n\times D}$), which we want to reduce to d variables ($\boldsymbol{Y} \in \mathbb{R}^{n\times d}$), where $d << D$.

For PCA to compare and find the best d variables composed of the original ones, it relates them by computing a covariance or correlation matrix which, importantly, centres each variable in its calculations.

$$\text{covariance}(\boldsymbol{X}_i, \boldsymbol{X}_j) = \frac{\sum_k (\boldsymbol{X}_{ik} - \overline{\boldsymbol{X}_i})(\boldsymbol{X}_{jk} - \overline{\boldsymbol{X}_j})}{n}, \qquad \text{correlation}(\boldsymbol{X}_i, \boldsymbol{X}_j) = \frac{\text{covariance}(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\sigma_{\boldsymbol{X}_i}\sigma_{\boldsymbol{X}_j}}. \tag{5}$$

Usually using the covariance, the similarity matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{\mathbb{D}\times\mathbb{D}}$ is constructed iteratively relating each pair of variables, but it can also be calculated matricially following:

$$\boldsymbol{X'}_k = \boldsymbol{X}_k - \overline{\boldsymbol{X}_k} \qquad , \forall k \in \{0, 1, \ldots, D\} \tag{6}$$

$$\underbrace{\boldsymbol{\Sigma}}_{D\times D} = \frac{\overbrace{\boldsymbol{X'}^\top}^{D\times n} \overbrace{\boldsymbol{X'}}^{n\times D}}{\boldsymbol{n}}, \tag{7}$$

where $\boldsymbol{X'}_k \in \mathbb{R}^n$ represents the $\boldsymbol{X}_k$ matrix centred by components.

Now that we have the *similarity matrix* $\boldsymbol{\Sigma}$, we look for a new orthogonal basis that maximises the variance of the dataset over the minimum components possible. To do this, since we are considering a symmetric and positive semi-definite matrix, performing an eigenvalue decomposition over the matrix $\boldsymbol{\Sigma}$ returns a set of orthogonal eigenvectors, and their eigenvalues, representing the orthogonal basis that maximises the dataset's variance.

These, when analysed individually, represent each of the new directions of the new eigenbasis and the scale of the variance of the dataset over that particular direction.

Note that this approach can be used over any type of similarity matrix, for example, any symmetric distance matrix between points can also be used, for example, the Gram matrix $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^\top$. However, they need to be double-centred following the transformation:

$$k_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_l d_{il}^2 - \frac{1}{n}\sum_l d_{lj}^2 + \frac{1}{n^2}\sum_{lm} d_{lm}^2\right), \tag{8}$$

where d_{ij} represent the euclidean distance between i and j, for example.

Since the purpose is to reduce the dimensionality of the data set, we do that by selecting the d new variables, i.e. eigenvectors, that correspond to the biggest eigenvalues, ending up with the new variables that display the most variance. The value of d is usually set as a way to retain a certain percentage of the variance in the original data, usually $95 - 99\%$ of the variance.

The operation to perform the eigenvalue decomposition is: The conditions that need to stand when performing the eigenvalue decomposition are:

$$\boldsymbol{\Sigma V} = \boldsymbol{V\lambda} \qquad \Longleftrightarrow \qquad \boldsymbol{\Sigma} = \boldsymbol{V\lambda V}^\top, \tag{9}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{d\times d}$ is the diagonal matrix of the d biggest eigenvalues of $\boldsymbol{\Sigma}$, and $\boldsymbol{V} \in \mathbb{R}^{D\times d}$ is the column matrix of eigenvectors corresponding to λ's eigenvalues. Importantly for these calculations, these two conditions are equivalent solely because $\boldsymbol{\Sigma}$ is symmetric also due to that, the eigenvector matrix $\boldsymbol{V}$ represent an orthogonal basis.

To project the data points from the higher dimensional space to the lower one perform:

$$\boldsymbol{Y} = \boldsymbol{XV}, \tag{10}$$

where $Y \in \mathbb{R}^{n\times d}$ represent the data points in the embedding space (i.e., the PCA projections) and $\boldsymbol{X} \in \mathbb{X}^{n\times D}$ is the source matrix.

Finally, we can reconstruct an approximation of the original data using the *projection matrix* $\boldsymbol{V}\boldsymbol{V}^\top$:

$$\hat{\boldsymbol{X}} = \underbrace{\boldsymbol{X}\boldsymbol{V}}_{\text{PCA projs.}} \boldsymbol{V}^\top. \tag{11}$$

We call the *embeddings*, the representation of the original data points in the lower dimensional space, but in this particular method, they are also called the PCA projections. If the original data were not already centred, the final step is to sum the mean of each variable to each point, returning the reconstructed data to the same relative position as the original. Note that, here, we operate over the reconstructed points in the data space, in $\boldsymbol{D}$ dimensions.

Multidimensional Scaling (MDS) Like PCA, Multidimensional Scaling (MDS) [7, 8] is a linear method that finds the best orientations for new variables from an eigenvalue decomposition, but this time the matrix to operate on can consider any form of distance between the original points and after the reduction.

It can also be represented as an optimisation problem, where the objective is to minimise the difference of the distances between the two spaces, i.e. for a given point on the original space, we want its distance to any other point to be equal to the *euclidean* distance between these points in the space after applying dimensionality reduction.

MDS is another linear method that given a desired number of variables, maintains the distance between each pair of points the best it can. Like PCA, it can be solved in closed form.

Analytically, the process is to build a centred inner-product matrix $\boldsymbol{B} \in \mathbb{R}^{n\times n}$, commonly considered a *similarity matrix* (can be viewed as how much two given points are related). It is built from the distance matrix $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ representing the distance between any two given points. $\boldsymbol{B}$ is then calculated:

$$\boldsymbol{B} = -\frac{1}{2}\boldsymbol{H}\boldsymbol{D} \times \boldsymbol{D}\boldsymbol{H}, \tag{12}$$

where $\boldsymbol{H}$ represents the centring matrix $\boldsymbol{H} = \mathbb{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top$, fundamentally, $\mathbf{1}_n\mathbf{1}_n^\top$ represent an $n \times n$ matrix of 1's. Note that $\boldsymbol{D} \times \boldsymbol{D}$ represent the pairwise product (Hadamard product).

After that, an eigenvalue decomposition can be performed over the matrix $\boldsymbol{B}$ and we can select the eigenvectors corresponding to the d biggest eigenvalues. The procedure is the same as in PCA (Eq. 9).

MDS can be written as an optimisation problem:

$$\min_{\boldsymbol{Y}} \quad \sum_{ij} \left(d_{ij} - \|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2\right)^2, \tag{13}$$

where $\boldsymbol{Y} = \begin{bmatrix}\boldsymbol{y}_1 & \cdots & \boldsymbol{y}_n\end{bmatrix} \in \mathbb{R}^{n\times d}$, and d_{ij} represents a distance function between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ (i.e., points in the original space).

In *Classical* MDS, we attempt to preserve euclidean distances in the embedding space, i.e., we solve:

$$\min_{\boldsymbol{Y}} \quad \sum_{ij} \left(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 - \|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2\right)^2 \tag{14}$$

In the same circumstances, since we know that $\boldsymbol{B}$ will be symmetric and positive semidefinite, we can directly calculate it from the original dataset:

$$\boldsymbol{B} = (\boldsymbol{H}\boldsymbol{X})\,(\boldsymbol{H}\boldsymbol{X})^\top, \tag{15}$$

to further proceed with the closed-form approach.

Classical MDS gives the exact same solution as PCA. As we will see in the next section, we may choose other distance metrics to account for nonlinearities in the data manifold.

Observations Although linear methods can be very useful and accurate on many linear cases, because they assume linearity between variables, when put to use on data sampled from non-linear manifolds they can't capture this complex relation between variables, so their performance decreases in function of the curvature of the manifold.

2.2.2 Isomap

The Isomap [9] reduction method can be seen as an extension of MDS where the distance matrix to be used is not calculated by Euclidean distances but rather by geodesic distances.

Local linearity is an important and common assumption made by many non-linear methods where the curvature of the manifold in the small region around a given point can be considered linear, thus ignoring the curvature and allowing for the true distance between points to be approximated by the Euclidean distance. This property is crucial for geodesic distance calculations.

The geodesic distance is the theoretical distance between two points along a manifold. In practice, we assume local linearity. This assumption enables us to create neighbourhoods of data points around each one from the Euclidean distance between them, this is, by calculating the Euclidean distance from a point to all the others and selecting only the closest ones, we create a *neighbourhood graph*. Connecting neighbours in the neighbourhood graph and finding the shortest path between two given points is a good approximation of the geodesic distance. A usual approach for calculating it is using the Dijkstra or Floyd-Warshall algorithms.

In general, we cannot expect euclidean distances between points in the data space to be proportional to distances along the data manifold. This downside is even more noticeable on distant points. To better relate points along a manifold, we may use geodesic distances, since they basically calculate the same Euclidean distance if only the manifold was approximately laying flat, overcoming the non-linearity by traversing the manifold along neighbour points.

Here, the method builds a distance matrix $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ representing the geodesic distance from a given point to each of the others. Using geodesic distances is what makes this method non-linear. To calculate geodesic distances along a graph, we can use the Dijkstra or Floyd-Warshall algorithms. The optimisation problem is formulated as in Eq. 13, where the distance between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is given by the approximated geodesic distance d_{ij}.

Like MDS, we can solve the problem in closed form by squaring the distance matrices and centring them, like a PCA approach that uses a distance matrix instead. The approach is to calculate the similarity matrix $\boldsymbol{B}$ (as in Eq. 12) using (approximated) geodesic distances d_{ij} rather than euclidean distances in the data space.

Note that this geodesic distance function can also be viewed as a kernel for other methods, i.e., a transformation function/matrix relating the data points.

2.2.3 Locally Linear Embedding (LLE)

To mitigate MDS and Isomap's main flaw of giving too much focus to the distance between very far apart points instead of the closest and most important ones to optimise, the LLE [10] bases itself on the relation of each point with its neighbours.

Because its process of reducing the dimensionality is based on translation, rotation, and rescaling, by creating a matrix of influences $\boldsymbol{W}$ between neighbouring points, it is then able to use this relation matrix to embed the data. The problem is represented as:

$$\min_{\boldsymbol{Y}} \quad \sum_i \left\|\boldsymbol{y}_i - \sum_j w_{ij}\boldsymbol{y}_{ij}\right\|^2, \tag{16}$$

where $w_{ij} \in [0,1]$ represents how much a point j influences i's position. Note that we only calculate influences between points that are k-nearest neighbors. This is how the weight matrix is calculated:

$$\min_{\boldsymbol{W}} \quad \sum_i \left\|\boldsymbol{x}_i - \sum_j w_{ij}\boldsymbol{x}_{ij}\right\|^2 \tag{17}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{N}_i} w_{ij} = 1, \quad \forall i = 1,\dots,n, \tag{18}$$

where $\mathcal{N}_i$ is the set of neighbours of i. Consequently, the rest of the w_{ij} for points j that are not in the neighbourhood of i are left as 0, making $\boldsymbol{W}$ a sparse matrix.

With this said, the problem minimises the difference between the current position of a given point and the position that its neighbours would pull it to, originally. This creates a trivial solution where all points coincide at the origin so, to fix this, the constraint $\|y_i^{(k)}\|^2 = 1$ for all k dimensions is added, preventing points from ending up in the origin.

The optimal solution to the optimisation problem can be found following:

$$\sum_i \|\boldsymbol{y}_i - \sum_j \boldsymbol{w}_{ij}\boldsymbol{y}_{ij}\|^2 = (\boldsymbol{Y} - \boldsymbol{WY})^\top(\boldsymbol{Y} - \boldsymbol{WY}) \tag{19}$$

$$= \boldsymbol{Y}^\top(\boldsymbol{I} - \boldsymbol{W})^\top(\boldsymbol{I} - \boldsymbol{W})\boldsymbol{Y} \tag{20}$$

$$= (\boldsymbol{I} - \boldsymbol{W})^\top(\boldsymbol{I} - \boldsymbol{W}), \tag{21}$$

where $\boldsymbol{I} \in \mathbb{R}^{n\times n}$ is the identity matrix and $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ is the reconstruction matrix.

The resulting embedding matrix $\boldsymbol{Y} \in \mathbb{R}^{n\times n}$ can then be used as in Eq. 9 as a similarity matrix, where the selection of the d smallest eigenvectors ends up with the usual eigenbasis for the embedded space.

2.2.4 Laplacian Eigenmaps (LE)

Like LLE, Laplacian Eigenmaps (LE) [11] create a weight matrix $\boldsymbol{W}$ which quantifies the influence of each neighbour of a given point in its coordinates. This, while assuming that the specific point and its neighbours are in the same plane, that is, local linearity.

Instead of minimising the distance to the theoretical weighted position that each point's neighbours are pulling it to, LE minimises the weighted distance to each neighbour directly:

$$\min_{\boldsymbol{Y}} \quad \sum_{ij} \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 w_{ij}, \tag{22}$$

where $\boldsymbol{W}$ is a sparse matrix, that for each pair of neighbour points i and j, w_{ij} is computed using the Gaussian kernel function:

$$w_{ij} = e^{-\frac{\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2}{2\sigma^2}}. \tag{23}$$

This means that neighbour points in the high-dimensional space are put as close together as possible in the embedding space.

To further solve the optimisation problem, the following diagonal degree matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is built based on $m_{ii} = \sum_j \boldsymbol{w}_{ij}$, consisting of the sum of the weights at each point, note that they do not sum up to 1 as in Eq. 18.

The problem can then be decomposed into:

$$\sum_{ij} \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 \boldsymbol{w}_{ij} = \underbrace{\sum_i \|\boldsymbol{y}_i\|^2 m_{ii} + \sum_j \|\boldsymbol{y}_j\|^2 m_{jj}}_{2\boldsymbol{YMY}^\top} + 2\underbrace{\sum_{ij}(\boldsymbol{y}_i\boldsymbol{y}_j^\top)\boldsymbol{w}_{ij}}_{2\boldsymbol{YWY}^\top}, \tag{24}$$

optimising this problem finds the same solution as the following:

$$\min_{\boldsymbol{Y}} \quad \boldsymbol{Y}^\top \boldsymbol{LY} \tag{25}$$

$$\text{s.t.} \quad \boldsymbol{Y}^\top \boldsymbol{MY} = \boldsymbol{I_n}, \tag{26}$$

where the constraint is needed to prevent the same trivial solution as in LLE.

It can then be solved as the eigenvalue problem:

$$\boldsymbol{LV} = \lambda \boldsymbol{MV}, \tag{27}$$

where $\boldsymbol{L} \in \mathbb{R}^{D\times D}$ is the Laplacian matrix $\boldsymbol{L} = \boldsymbol{M} + \boldsymbol{W}$, selecting then the d smallest eigenvectors and proceeding like explained before.

2.2.5 Hessian Locally Linear Embedding (H-LLE)

Considerately related to LLE, but instead of calculating the weighted best coordinates, in the embedded space, for a point from its k-nearest neighbourhood, the Hessian LLE [12] approximately calculates the curviness of the manifold from the local Hessian and minimises it in order to flatten the dataset. This is done simply by performing an eigenvalue decomposition over the Hessian Matrix of local Hessian approximations and looking for the d smallest eigenvalues, the respective eigenvalues form the embedded $\boldsymbol{Y}$ matrix containing the low dimensional representation of the data.

To find the matrix of Hessian estimations at each point, H-LLE assumes local linearity across the neighbourhood of each point and computes its tangent space by applying PCA to its k-nearest neighbors (collected into a matrix), ending up with selecting the d most representative directions of the manifold around each data point.

After that, the estimate of the Hessian at each point is formulated from the matrix $\boldsymbol{Z}$ which is composed of the cross product between the principal components, with an added column of 1's, at each point, and then orthogonalised through the Gram-Schmidt process.

The local tangent Hessian estimation $\boldsymbol{H}_i$ is now built by transposing the selection of the last $\frac{d(d+1)}{2}$ columns of $\boldsymbol{Z}$, and the global $\mathcal{H}_l m$ Hessian estimator consists of:

$$\mathcal{H}_{lm} = \sum_i \sum_j \left((\boldsymbol{H}_i)_{jl} \times (\boldsymbol{H}_i)_{jm}\right). \tag{28}$$

After estimating the tangent spaces on each point, H-LLE proceeds by minimising the curvature of the data set, described by $\mathcal{H} \in \mathbb{R}^{n \times n}$, by solving the eigenvalue problem as in Eq. 9 and selecting the eigenvectors correspondent to the d smallest non-zero eigenvalues results in the $\boldsymbol{Y} \in \mathbb{R}^{n \times d}$ embedded data.

Importantly, the number k of nearest neighbours around each point to approximate the Hessian must be such that $k > d(1 + \frac{1+d}{2})$, where d is the number of components found by PCA. This means that for regions of the manifold that are nonlinear enough, the number of components necessary to represent the data in a linear subspace may result in a very large k, which may fail in capturing local geometry. This seems to happen in practice, as demonstrated by experiments on natural datasets, which tend to lie on nonlinear manifolds [12].

2.2.6 Local Tangent Space Analysis (LTSA)

Comparably to H-LLE, but linearising the data in a different manner, LTSA [13] describes its input data set as the tangent space of the manifold at each data point.

With a different perspective, the local tangent space at each point will be approximated to its tangent space in the lower dimensionality.

Fundamentally, the LTSA, from the local tangent space of the manifold at each data point, tries to find a mapping from that local tangent space, to the embedded position of the point in analysis. To put this in practice, the LTSA involves of the minimisation of differences between the possible embedded positions of a point, and the result after performing the mapping from the local tangent of the point to the lower dimensional space. This is formulated as the optimisation problem:

$$\min_{\boldsymbol{Y},\boldsymbol{L}} \quad \sum_i \|\boldsymbol{y}_i \boldsymbol{J}_i - \boldsymbol{L}_i \boldsymbol{\Theta}_i\|^2, \tag{29}$$

where $\boldsymbol{\Theta}_i$ represents the local tangent space calculated by the PCA over the manifold at the point $\boldsymbol{x}_i$, $\boldsymbol{L}$ is the map from the tangent of the manifold at the same point $\boldsymbol{x}_i$ to the objective point $\boldsymbol{y}_i$ and $\boldsymbol{J}_i$ is a double-centring matrix transformation equivalent to performing:

$$d_{ij} = -\frac{1}{2}\left(d_{ij} - \frac{1}{k}\sum_{l \in \mathcal{N}_i} d_{il} - \frac{1}{k}\sum_{l \in \mathcal{N}_i} d_{jl} + \frac{1}{k^2}\sum_{l,m \in \mathcal{N}_i} d_{lm}\right), \tag{30}$$

where k is the size of the neighbourhood of i and $\mathcal{N}_i$ represents the set of its points. This operation ensures that each row and column, of the resulting matrix, have a mean of 0, and also, the whole matrix is mean 0.

This means that LTSA is concurrently looking, for each data point, for the final position and the map from the local tangent space to its final position.

This optimisation method can also be solved as an eigenvalue problem over the alignment matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ which is iteratively computed from a matrix of 0's, where for the set $\mathcal{N}_i$ of nearest neighbours of $\boldsymbol{x}_i$ the matrix is updated following:

$$\boldsymbol{B}_{\mathcal{N}_i \mathcal{N}_i} = \boldsymbol{B}_{\mathcal{N}_{i-1} \mathcal{N}_{i-1}} + \boldsymbol{J}_k (\boldsymbol{I} - \boldsymbol{V}_i \boldsymbol{V}_i^\top) \boldsymbol{J}_k, \tag{31}$$

where $\boldsymbol{V_i} \boldsymbol{V_i}^\top$ represent the PCA projection calculated from the local neighbourhood of i, and subsequently, the local alignment of the manifold at the data point i is represented by $\boldsymbol{I} - \boldsymbol{V_i} \boldsymbol{V_i}^\top$.

The global alignment matrix $\boldsymbol{B}$ then represents the sum of all the local alignment matrices. Finishing the process, $\boldsymbol{B}$ is used to find the eigenbasis and subsequently the embedded data, by performing an eigenvalue decomposition over $0.5(\boldsymbol{B} + \boldsymbol{B}^\top)$ and selecting the eigenvectors corresponding to the d smallest non-zero eigenvalues.

2.2.7 t-distributed Stochastic Neighbour Embedding (t-SNE)

Originating from SNE [14], t-SNE [15] is also most focused on data visualisation, that is, it is best fitted for dimensionality reduction into 2 or 3 dimensions. Both need a hand-picked final number of dimensions and capture the differences between data points by computing the conditional probability of a neighbourhood, $p_{i|j}$ and a similar $q_{i|j}$ for the reduced space, which can be compared with the weights from LE and other weight-based methods, but with a different way of calculating:

$$p_{i|j} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_i\|_D^2 / 2\sigma^2)}{\sum_{k \neq l} \exp\left(-\|\boldsymbol{x}_k - \boldsymbol{x}_l\|_D^2 / 2\sigma^2\right)}, \qquad q_{i|j} = \frac{\exp(-\|\boldsymbol{y}_i - \boldsymbol{y}_i\|_d^2)}{\sum_{k \neq l} \exp\left(-\|\boldsymbol{y}_k - \boldsymbol{y}_l\|_d^2\right)} \tag{32}$$

Due to being asymmetric, these cause a problem on outlier points, where the distance to most points is very large, which causes the point to be too far away in the reduced space. t-SNE reduced its impact by using symmetric distances between each point i and j applying the mapping $p_{ij} = p_{ji} = \frac{p_{i|j} + p_{j|i}}{2n}$.

Also, to better organise and visualise the data, instead of following a Gaussian distribution of the neighbour points over a given point $\boldsymbol{x}_i$, the t-SNE changes the formulation of q_{ij} to follow a student t-distribution with one degree of freedom in order to spread the clusters further while not making the already separated ones too far:

$$q_{ij} = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)^{-1}}. \tag{33}$$

This creates a much more balanced gradient function, avoiding situations where SNE would be biased to attract points that were already in seemingly good positions.

The following action is to minimise the sum of the Kullback-Leiber Divergence over all the points:

$$C = KL(P\|Q) \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{34}$$

which, due to the symmetry of t-SNE, can be solved with a simple gradient function:

$$\frac{\partial C}{\partial \boldsymbol{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\boldsymbol{y}_i - \boldsymbol{y}_j). \tag{35}$$

More importantly, this new probability distribution on the lower-dimensional space also approaches the inverse square law, meaning that, at far distances, the scale of the map becomes virtually irrelevant, making far-apart clusters of map points behave as a single point.

This leads us to the most impactful change in t-SNE: considering clusters of points as single points, the original $O(n^2)$ formulation can be made into $O(n\log(n))$.

Besides, from the fact that the whole structure of this method is focused on 2D and 3D representations of the data, this latest optimisation also decays with the increase of dimensions, combined with the fact that it is needed to know into how many dimensions we want t-SNE to reduce the data to, in cases of higher and harder to calculate intrinsic dimensions, t-SNE does not get so competitive comparatively with other DR methods.

2.2.8 Kernel PCA (K-PCA)

The Kernel PCA [16] is a method generalization from the classic PCA. Instead of searching for the best eigenbasis directly on the matrix presenting the relation between each pairwise variable, the K-PCA performs an initial transformation on the source data, following a kernel function. This operation can be related to a Classical MDS where the objective function, on an Euclidean basis, is transformed following the kernel transformation that is virtually able to unbend any non-linearity present in a manifold. Because K-PCA no longer uses the covariance or correlation values between variables, which naturally centre the data, now, the K-PCA uses a double-centring transformation like Eq. 30.

After applying the kernel function and centring the data set, it ends up with an equivalent $\boldsymbol{X'} \in \mathbb{R}^{n\times D}$ matrix which can continue the PCA's solution as in Eq. 7: computing the covariance $\boldsymbol{\Sigma} = \boldsymbol{X'}\boldsymbol{X'}^{\top}$ and selecting the eigenvectors corresponding to the top d eigenvalues.

This method is not used more often because it is normally hard to find an adequate kernel function to linearise the dataset, since it is not usually found by any calculations, but rationalising, and some trial and error.

2.2.9 Maximum Variance Unfolding (MVU)

Maximum Variance Unfolding [17] has a consequence that ended up fixing K-PCA's main flaw as stated before. Its optimisation problem can be observed as the search for a good lineariser kernel process: in basic terms, the reasoning behind MVU is to stretch any existing manifold that there might exist in the data set. This is done by maximizing the distance between the points of the data set, while maintaining local isometry between neighbour points.

All this while keeping the data centred, to remove degenerate solutions, formulates:

$$\max_{\boldsymbol{Y}} \quad \sum_{ij} \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 \tag{36}$$

$$\text{s.t.} \quad \sum_{i} \boldsymbol{y}_i = 0 \tag{37}$$

$$\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall i, j \in \boldsymbol{G}, \tag{38}$$

where the first constraint is responsible for keeping the data set centred while the following is the one which maintains the local distances after the linearisation of the manifold. $\boldsymbol{G} \in \mathbb{R}^{n\times n}$ is the neighbourhood matrix which represents whether j is a k-nearest neighbour of i.

Since this operation virtually flattens the manifold, this means that the Euclidean distance between two points in the manifold, after being stretched, is as close to the theoretical geodesic distance as it can be. Since this takes no approximation in order to find the real distance along a manifold, Isomap can be seen as an approximation to MVU.

The difficulty of this problem comes from its non-convexity and the inexistence of an equivalent closed-form solution. With that, this optimisation problem can be converted[1] into an SDP, and thus

[1]The conversion calculations and constraint relaxation from the non-convex to the SDP problem can be seen in the Appendix.

convex problem [17]:

$$\max_{\boldsymbol{K}} \quad \operatorname{trace}(\boldsymbol{K}) \tag{39}$$

$$\text{s.t.} \quad \sum_{ij} k_{ij} = 0 \tag{40}$$

$$\boldsymbol{K} \succeq 0 \tag{41}$$

$$k_{ii} + k_{jj} - 2k_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \quad , \forall i, j \in \boldsymbol{G}, \tag{42}$$

where the $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ matrix and subsequently $k_{ij} \in \mathbb{R}$ represent the inner product between each the i-th and j-th points.

When the solution is reached, the value of $\boldsymbol{K}$ is the learned kernel, which this process can be viewed as. After having the final problem's matrix, the dataset is virtually linearised, and so a regular eigenvalue decomposition can be performed as in Eq. 9 have a representation in the embedded space with minimal loss of information.

As analysed in [18], MVU presents a computational complexity of $O((nk)^3)$, i.e. the computational expenses grow cubically with the n number of points and the k number of neighbours to consider as the neighbourhood of each point, presenting a much faster growth compared to other methods. This is most influenced by the size of the Gram objective matrix, in $\mathbb{R}^{n \times n}$, of inner products between each pair of data points, and on the size of the neighbourhood to consider, creating a constraint for each pair of data points. Along with that, and common to all numerical approaches, solving the final eigenvalue problem has the computational complexity of $O(n^3)$.

3 Related Work

Landmark MVU (L-MVU) [19] is an extension to MVU which facilitates its application on larger data sets. It approximates the manifold's whole structure from a subset of reference points, the *landmarks*, based on the original data. It was found that many of the constraints from MVU's formulation are redundant, so another improvement from this extension is the incremental insertion of those constraints, inserting more only when the solution takes invalid values.

This extension was also proposed for related methods such as Isomap and LLE. Their landmark versions L-Isomap [5, 20] and L-LLE [5, 21] result in speed improvements that allow its application to larger datasets at the cost of some precision.

However, when dealing with natural datasets, greater problems emerge. Typically, natural datasets lie on multiple or disjoint manifolds. Furthermore, data may not be well sampled from the underlying manifolds, resulting in sparse areas in the data space. Both of these result in neighbourhood graphs that are not connected. Many of the methods above either perform poorly or have undefined solutions when applied to disconnected neighbourhood graphs.

One solution to remedy this issue could be to embed a single component (e.g., the largest) and apply out-of-sample extensions to embed the others [22–24]. However, since out-of-extension methods rely on embedding each point as some function of other (relatively distant) points that are already embedded, it is likely that their distances in the data space are not very informative, yielding imprecise embeddings. Indeed, in practice we observe a degradation of the learned representations [25].

Another solution might be to simply increase the value of k until the resulting neighborhood graph is connected (with $k = n$ in the limit). However, using a large number of neighbors might result in the loss of local information, so that is not a good solution in practice [25].

As such, [4] proposes adding edges between connected components of a (disconnected) graph such that the intrinsic dimensionality of the edge space is equal to that of each of the components. This is done iteratively between pairs of components until the NG becomes connected. They apply their method to LLE, H-LLE, LE, and Isomap, generally improving their abilities to perform NLDR on data lying on disjoint manifolds. However, this method has a few disadvantages:

- It assumes that all manifolds/connected components have the same intrinsic dimensionality.

- It relies on an estimation/prior knowledge of the intrinsic dimensionality of the data.
- It creates as many connections as possible between components. This is disadvantageous for methods such as MVU, where the rigidity introduced by the new edges may prevent the unfolding of the manifold.

To conclude, most non-linear dimensionality reduction methods rely on building a so-called neighbourhood graph (NG) of the data, which is used to discover its underlying geometric structure. These neighbourhood graphs can be highly disconnected if data are sampled from disjoint manifolds. Linear techniques such as PCA work fine in these cases, but NG-based methods do not work in these situations either because relationships between points are undefined (e.g., Isomap, where geodesic distances can't be calculated globally) or because the problem becomes ill-defined (e.g., MVU, where the optimization becomes unbounded).

4 Proposed Approach

4.1 Problem Specification

As mentioned before, NG-based methods have big performance downgrades, or don't work at all, on datasets that present some separation between clusters or submanifolds; ending up not being capable of finding a solution in cases where a connection between submanifolds is non-existent.

Specifically for MVU, the problem originates from the fact that the constraints that take part in the optimisation problem are responsible for keeping the distances between points in its neighbourhood constant. In contrast, the overall optimisation function pulls every other point away. This results in an unbounded problem where MVU simply pulls the connected components further.

In particular, while existing solutions have increased the applicability of other methods, they do not yet allow for the application of MVU to real-world data. Due to the stronger isometry properties of MVU and the fact that it usually outperforms other non-linear dimensionality reduction methods on artificial data [25], we wish to extend its applicability to real-world data.

4.2 Solution Specification

In this subsection, we describe the procedure of our proposed solution:

Consider a dataset of disjoint manifolds, where $\boldsymbol{S}_i \in \mathbb{R}^{n_i \times D}$ is the subset of points belonging to the connected component of index i, where n_i is the number of points of the i'th disjoint component.

For each connected component $\boldsymbol{S}_i$ we preform MVU over it, in order to achieve an embedded representation $\boldsymbol{Y}_i \in \mathbb{R}^{n_i \times d}$. We now look for the subset $\boldsymbol{C}_i \in \mathbb{R}^{d+1 \times d}$ of $d+1$ points that are the furthest apart, which computationally would approximate to:

$$\max_{\boldsymbol{C}_i} \quad \|\boldsymbol{c}_m - \boldsymbol{c}_n\|^2, \forall \boldsymbol{c}_m, \boldsymbol{c}_n \in \boldsymbol{Y}_i. \tag{43}$$

Additionally, we look for the points that are closest to each of the other connected component, forming the subset $\boldsymbol{L} \in \mathbb{R}^{l-1 \times d}$, where l is the number of disjoint components:

$$\min_{\boldsymbol{L}_i} \quad \|\boldsymbol{L}_{im} - \boldsymbol{L}_{jn}\|^2, \forall \boldsymbol{L}_{im} \in \boldsymbol{Y}_i \land \boldsymbol{L}_{jn} \in \boldsymbol{Y}_j \tag{44}$$

After performing MVU over each individual connected component, it is possibler to achieve embedded spaces with different sizes, thus it is necessary components to have the same dimensionality, adding variables of value 0 until they equalise.

From each of the subsets $\boldsymbol{C}_i$ and $\boldsymbol{L}_i$ we now build a dataset of points from all the disjoint components, and preform MVU over it, now. It is important that the connection matrix of this formulation describes connections between all inner-component points, and the connections inter-component with the closest point between them:

$$\max_{\boldsymbol{Y}} \quad \sum_{ij} \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 \tag{45}$$

$$\text{s.t.} \quad \sum_{i} \boldsymbol{y}_i = 0 \tag{46}$$

$$\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 = \|\boldsymbol{C}_{im} - \boldsymbol{C}_{jn}\|^2, \forall C_{im} \in \boldsymbol{C}_i \wedge C_{jn} \in \boldsymbol{C}_j \tag{47}$$

$$\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 = \|\boldsymbol{L}_{im} - \boldsymbol{L}_{jn}\|^2. \tag{48}$$

Due to MVU's constraint of keeping connected point' distances untouched, it is the possible to position each manifold in its place based on the reference points used to preform the inner-component reduction.

4.3 Methodology

Along with the need to measure the success of each method modification, there will also be the need to compare those method modifications with the other presented methods. Given that they present different interpretations and approaches to lowering the dimension of a dataset, the comparison between them may not take the same approach as comparing different versions of the same method. With this said, below we present the various ways that we can compare the success of each method change:

4.3.1 Algorithms

We will test all the algorithms described above as they are the most widely used and are representative dimensionality reduction methods. Furthermore, we will augment them with the enhanced neighbourhood graph as in [4] to compare them and analyse their performance over datasets with disjoint manifolds.

Given that most methods in analysis use some type of neighbourhood graph, the k hyper-parameter can be considered a unifying variable which would offer comparable situations to the set of methods, making it easy to observe the impact that disjoint datasets present to each method. However, this parameter may not take a rather large range of values, and is highly dependent on the properties of each dataset. Thus, alike [25], we are going to test the different datasets with k ranging from 3 to 15. Relatively to the learning rate, in the case of t-SNE, we are going to explore values between 10 and 1000.

4.3.2 Datasets

We will consider both artificial, which present unique challenges to the different algorithms, and natural datasets. To construct each artificial dataset, we consider the variables p_i and q_i, uniformly distributed in the range $[0, 1]$. In artificial datasets, Gaussian noise with a small variance is added to all data points, as in [25].

Swiss Roll Addressing data that lies on a low-dimensional manifold that is isometric to Euclidean space, the Swiss Roll represents a manifold turning into itself.

It can be built following simple computational operations leaving the possibility of some fine-tuning relating the density of points along the manifold. Being artificially built, the number of points can vary as we prefer, as can the density of points along the manifold. However, the number of dimensions is expected to remain as the typical Swiss roll dataset, 3.

The Swiss Roll is thus formed of points $\boldsymbol{x}_i = [t_i \cos(t_i), t_i \sin(t_i), 30q_i]$ where $t_i = \frac{3\pi}{2}(1 + 2p_i)$.

Broken Swiss Roll presents an extension of the previous dataset, presenting the cases where the density of points in some locations reach null points, separating the continuous Swiss roll manifold into multiple disjoint components. This, is aimed to analyse data that lies on or near a disconnected manifold. The formulation of this dataset is also based on the previous, with the added clause of rejecting the points where $\frac{2}{5} < t_i < \frac{4}{5}$, resulting in two disjoint components.

Helix representing a closed coil in the span of a 3-dimensional space. Testing if each method handles data lying on a low-dimensional manifold that is not isometric to Euclidean space.

The dataset is built following $\boldsymbol{x}_i = [(2 + \cos(8p_i)) \cos(p_i), (2 + \cos(8p_i)) \sin(p_i), \sin(8p_i)]$.

Twinpeaks the dataset is formulated from $\boldsymbol{x}_i = [1 - 2p_i, \sin(\pi - 2\pi p_i)), \tanh(3 - 6q_i)]$. This dataset also aims to evaluate data lying on a low-dimensional manifold that is not isometric to Euclidean space.

High-Dimensional dataset consists of points randomly sampled from a 5-dimensional manifold embedded in a 10-dimensional space. This, aims to test data forming a manifold with a high intrinsic dimensionality by computing ten different combinations of the five uniformly distributed random variables, some of which are linear and some of which are nonlinear.

We now present the natural datasets to be used in our experiments:

Teapots a 400 76x101 pixel images dataset, in 3-byte colours, of the same teapot from a 360-degree range of rotation.

MNIST composed of up to 70000 20x20 pixel images of digits (0 through 9) written by hand, with a 256-bit grey scale.

UMIST is a dataset of 564 images of 20 individuals' faces from different angles and poses. Each image is 220 x 220 pixels with a 256-bit grey scale.

It is expected that disconnections in neighbourhood graphs will happen either in regions of the data space between different digits and individuals, or in areas where the data is not well sampled.

These tasks are relevant examples to evaluate if the methods are able to identify some relation between data with no extra information besides the data itself.

4.3.3 Metrics

We follow [25] and evaluate dimensionality reduction methods according to three metrics that assess the quality of the embedded data:

1-NN Classifier Error [26] Calculates the ratio of points which, its closest neighbour is maintained from the original space throughout the reduction method.

Trustworthiness [27] Considering that $r(i, j)$ represents the position of j in the ranking of the closest point to i in the original space, and $\boldsymbol{U}_i$ limits that search to j's that are neighbour of i in the lower-dimension space but not in the original space, the trustworthiness asserts whether the points in the k-neighbourhood of an embedded point are also neighbours to it in the original space:

$$T(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_i \sum_{j \in U_i^{(k)}} (r(i, j) - k). \tag{49}$$

Continuity [27] Analogously, continuity asserts whether the points in the k-neighbourhood of a point in the original space are still neighbours to it in the embedded space:

$$C(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_i \sum_{j \in V_i^{(k)}} (\hat{r}(i, j) - k), \tag{50}$$

where $\hat{r}(i, j)$ represents the ranking of points in the neighbourhood of i in the embedded space, and $\boldsymbol{V}_i$ the set of points in the neighbourhood of i in the original space but not in the embedded space.

4.3.4 Experimental Setup

All algorithms and experiments described in this section shall be implemented in Python 3.12.5 [28], resorting to numerical (e.g., NumPy [29]), scientific (e.g., SciPy [30]), and optimization (e.g., CvxPy [31, 32]) libraries whenever necessary.

Whenever possible, we will test our implementations of the relevant algorithms against existing publicly available ones, such as those in Scikit-Learn's [33] manifold learning package.

Finally, all experiments will run on GAIPS's Nexus and ADA1 servers.

4.4 Work Schedule

In this section we describe the work plan leading to the elaboration of a master's thesis.

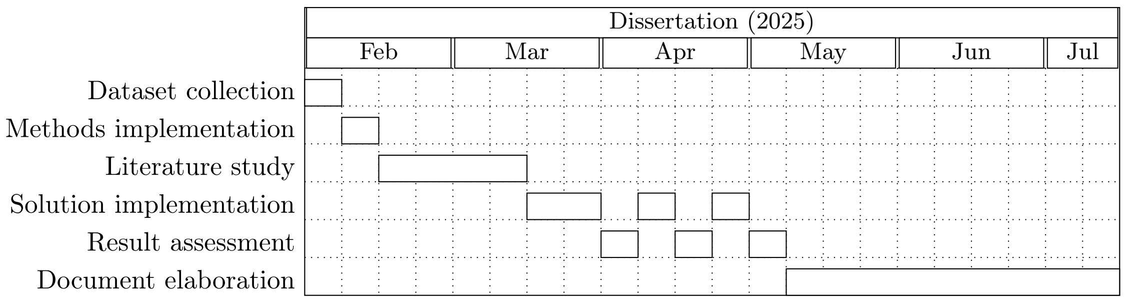

Figure 1: Gantt chart

After having acquired the datasets, we plan to setup an initial implementation for the base methods and test them against existing implementations (e.g., those in the manifold learning package of Sklearn [33]).

We will dedicate some time to studying approaches that can be taken and consider if there are any other alternative implementations to test. From that study, we plan to put our original approach into practice and any alternative solution we may find, alternating between their implementation and the analysis of the results achieved, hopefully leading to iterative improvement.

After testing all the approaches considered, we will write a master's thesis about the study done.

Bibliography

[1] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[3] D. E. Rumelhart, J. L. McClelland, P. R. Group *et al.*, *Parallel distributed processing, volume 1: Explorations in the microstructure of cognition: Foundations.* The MIT press, 1986.

[4] J. Fan, T. W. Chow, M. Zhao, and J. K. Ho, "Nonlinear dimensionality reduction for data with disconnected neighborhood graph," *Neural Processing Letters*, vol. 47, pp. 697–716, 2018.

[5] M. Vladymyrov and M. Á. Carreira-Perpinán, "Locally linear landmarks for large-scale manifold learning," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13.* Springer, 2013, pp. 256–271.

[6] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Unified framework for spectral dimensionality reduction, maximum variance unfolding, and kernel learning by semidefinite programming: Tutorial and survey," *arXiv preprint arXiv:2106.15379*, 2021.

[7] W. S. Torgerson, "Multidimensional scaling: I. theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[8] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[9] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[10] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[11] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[12] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *Proceedings of the National Academy of Sciences*, vol. 100, no. 10, pp. 5591–5596, 2003.

[13] Z. Zhang and H. Zha, "Principal manifolds and nonlinear dimensionality reduction via tangent space alignment," *SIAM journal on scientific computing*, vol. 26, no. 1, pp. 313–338, 2004.

[14] G. E. Hinton and S. Roweis, "Stochastic neighbor embedding," *Advances in neural information processing systems*, vol. 15, 2002.

[15] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[16] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[17] K. Q. Weinberger and L. K. Saul, "Unsupervised learning of image manifolds by semidefinite programming," *International journal of computer vision*, vol. 70, pp. 77–90, 2006.

[18] B. Borchers and J. G. Young, "Implementation of a primal–dual method for sdp on a shared memory parallel architecture," *Computational Optimization and Applications*, vol. 37, pp. 355–369, 2007.

[19] K. Weinberger, B. Packer, and L. Saul, "Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization," in *International Workshop on Artificial Intelligence and Statistics*. PMLR, 2005, pp. 381–388.

[20] V. Silva and J. Tenenbaum, "Global versus local methods in nonlinear dimensionality reduction," *Advances in neural information processing systems*, vol. 15, 2002.

[21] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Locally linear embedding and its variants: Tutorial and survey," *arXiv preprint arXiv:2011.10925*, 2020.

[22] T.-J. Chin and D. Suter, "Out-of-sample extrapolation of learned manifolds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 9, pp. 1547–1556, 2008.

[23] A. Gisbrecht, A. Schulz, and B. Hammer, "Parametric nonlinear dimensionality reduction using kernel t-sne," *Neurocomputing*, vol. 147, pp. 71–82, 2015.

[24] K. Bunte, M. Biehl, and B. Hammer, "A general framework for dimensionality-reducing data visualization mapping," *Neural Computation*, vol. 24, no. 3, pp. 771–804, 2012.

[25] L. Van Der Maaten, E. O. Postma, H. J. Van Den Herik *et al.*, "Dimensionality reduction: A comparative review," *Journal of machine learning research*, vol. 10, no. 66-71, p. 13, 2009.

[26] G. Sanguinetti, "Dimensionality reduction of clustered data sets," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 3, pp. 535–540, 2008.

[27] J. Venna and S. Kaski, "Local multidimensional scaling," *Neural Networks*, vol. 19, no. 6-7, pp. 889–899, 2006.

[28] G. van Rossum, Python, a general-purpose programming language.

[29] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[30] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[31] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[32] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

A Appendix

From the original formulation of MVU:

$$\max_{\hat{\boldsymbol{X}}} \quad \sum_{ij} \|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2 \tag{51}$$

$$\text{s.t.} \quad \|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall_{i,j} \in \boldsymbol{G} \tag{52}$$

$$\sum_i \hat{\boldsymbol{x}}_i = \vec{\mathbf{0}} \tag{53}$$

By centring data points beforehand, instead of considering the repelling force between all the points, since it is centred, all the forces would sum up to pushing each point away from the origin. The following simplification can be done:

$$\max_{\hat{\boldsymbol{X}}} \quad \sum_{i} \|\hat{\boldsymbol{x}}_i\|^2 \tag{54}$$

$$\text{s.t.} \quad \|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall_{i,j} \in \boldsymbol{G} \tag{55}$$

$$\sum_i \hat{\boldsymbol{x}}_i = \vec{\mathbf{0}} \tag{56}$$

The decomposition of the squares reaches the following:

$$\max_{\hat{\boldsymbol{X}}} \quad \sum_{i} \hat{\boldsymbol{x}}_i^\top \hat{\boldsymbol{x}}_i \tag{57}$$

$$\text{s.t.} \quad \hat{\boldsymbol{x}}_i^\top \hat{\boldsymbol{x}}_i + 2\hat{\boldsymbol{x}}_i^\top \hat{\boldsymbol{x}}_j + \hat{\boldsymbol{x}}_j^\top \hat{\boldsymbol{x}}_j = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall_{i,j} \in \boldsymbol{G} \tag{58}$$

$$\sum_i \hat{\boldsymbol{x}}_i^\top \underbrace{\sum_j \hat{\boldsymbol{x}}_j}_{\vec{\mathbf{0}}} = 0 \tag{59}$$

Then, generalizing to matricial calculations and considering $\boldsymbol{e} \in \mathbb{R}^{n \times n}$ to be the space neighbourhood matrix where $\boldsymbol{e}_i$ takes 1 at a given position if the corresponding data point is in the neighbourhood of the point at index i of the dataset:

$$\max_{\hat{\boldsymbol{X}}} \quad \text{trace}\left(\hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}}\right) \tag{60}$$

$$\text{s.t.} \quad \boldsymbol{e}_i^\top \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}} \boldsymbol{e}_i + 2\boldsymbol{e}_i^\top \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}} \boldsymbol{e}_j + \boldsymbol{e}_j^\top \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}} \boldsymbol{e}_j = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall_{i,j} \in \boldsymbol{G} \tag{61}$$

$$\mathbf{1}_n^\top \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}} \mathbf{1}_n = 0 \tag{62}$$

Now that the whole problem is written in order of $\hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}}$, by introducing the variable $\boldsymbol{K} \in \mathbb{R}^{n \times n}$:

$$\max_{\boldsymbol{K}, \hat{\boldsymbol{X}}} \quad \text{trace}\left(K\right) \tag{63}$$

$$\text{s.t.} \quad \boldsymbol{e}_i^\top \boldsymbol{K} \boldsymbol{e}_i + 2\boldsymbol{e}_i^\top \boldsymbol{K} \boldsymbol{e}_j + \boldsymbol{e}_j^\top \boldsymbol{K} \boldsymbol{e}_j = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall_{i,j} \in \boldsymbol{G} \tag{64}$$

$$\mathbf{1}_n^\top \boldsymbol{K} \mathbf{1}_n = 0 \tag{65}$$

$$\boldsymbol{K} = \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}} \tag{66}$$

Making sure that $\boldsymbol{K}$ is symmetric, positive semidefinite, and that its rank is greater than d, we isolate the non-convexity to the rank clause:

$$\begin{aligned}
\max_{\boldsymbol{K},\hat{\boldsymbol{X}}} \quad & \operatorname{trace}(K) & (67)\\
\text{s.t.} \quad & \boldsymbol{e}_i^\top \boldsymbol{K}\boldsymbol{e}_i + 2\boldsymbol{e}_i^\top \boldsymbol{K}\boldsymbol{e}_j + \boldsymbol{e}_j^\top \boldsymbol{K}\boldsymbol{e}_j = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2, \forall_{i,j} \in \boldsymbol{G} & (68)\\
& \mathbf{1}_n^\top \boldsymbol{K}\mathbf{1}_n = 0 & (69)\\
& \boldsymbol{K} \succeq 0 & (70)\\
& \operatorname{rank}(\boldsymbol{K}) \geq d & (71)
\end{aligned}$$

so relaxing the problem by removing the rank clause, we conclude with the convex SDP problem.:

$$\begin{aligned}
\max_{\boldsymbol{K}} \quad & \operatorname{trace}(\boldsymbol{K}) & (72)\\
\text{s.t.} \quad & \sum_{ij} k_{ij} = 0 & (73)\\
& \boldsymbol{K} \succeq 0 & (74)\\
& k_{ii} + k_{jj} - 2k_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \quad , \forall_{i,j} \in \boldsymbol{G} & (75)
\end{aligned}$$