A Review on Dimensionality Reduction for Unsupervised Metric Learning: From Linear Spaces to Manifolds

Alexandre Levada

A Review on Dimensionality Reduction for Unsupervised Metric Learning: From Linear Spaces to Manifolds

Alexandre L. M. Levada

Computing Department, Federal University of São Carlos, São Carlos, SP, Brazil
alexandre.levada@ufscar.br

Dimensionality reduction techniques are crucial for modern data analysis applications due to the large complexity of the datasets found in several domains of science. In order to find relevant patterns in this vast amount of data, a feature extraction step is often required. Linear approaches were the first class of methods applied to reduce dimensionality of data. However, they assume that the extracted features lie in an Euclidean space, which is not a reasonable assuption in many real problems. In 2000, with advances on kernel methods and computational resources, the research field manifold learning started to emerge with the pioneering ISOMAP algorithm, showing a much more realistic model for dimensionality reduction for metric learning. This paper presents a review of unsupervised metric learning techniques ranging from Principal Component Analysis (PCA) to modern manifold learning algorithms, such as t-SNE, Local Tangent Space Aligment, Diffusion Maps, among others, with a special attention to the mathematical background, but at the same time trying to elucidate the intuition behind each method. This year we celebrate the 20th anniversary of this remarkable field which had profound impact in the study of unsupervised metric learning methods, in the sense that these methods can learn a distance function that geometrically is better suited to represent a similarity measure between a pair of objects in a given dataset. Basically, our goal with this review is to provide a complete guide to researchers interested in initiating at the field.

Keywords: Dimensionality Reduction; Unsupervised Metric Learning; Manifold Learning

1. Introduction

The presence of multivariate data in pattern recognition and machine learning applications has been increasing drastically over the years. Modern datasets are often composed by a large number of examples, each of which having an even larger number of features. If the effects of a large sample size are positive to learning processes in general, the effects of an arbitrary increase in the number of features are known to be highly negative, especially in pattern classification tasks [29,76,89].

During the project of a pattern recognition system one of the first questions arised by most machine learning practioners is: Why should I use a dimensionality reduction algorithm if this method will probably increase the computational burden and slow the system down? In order to answer this question, in the following we provide some arguments that show how learning from high dimensional data can be a painful task.

The main goal of pattern recognition is to develop mathematical models to allow automatic discovery of regularities in data through computational algorithms. Two obvious reasons for dimensionality reduction are: to alleviate the computational burden and to perform data visualization. Tipically, a dataset with n samples and m features is represented by

a data matrix $X_{n\times m}$. Clearly, for every $m' < m$, both memory usage and computational cost are reduced, making the algorithms faster and more efficient. Moreover, data visualization can bring important insights to any data analyst interested in solving large scale problems.

However, one of the major drawbacks in high-dimensional data analysis is the *curse of the dimensionality*. This term was first mentioned by Bellman in 1961 [9] and refers to the fact that to estimate a function of several variables to a given degree of accuracy, the sample size needs to grow with the number of variables. A related fact is that hyperspaces are inherently sparse, causing the empty space phenomenon [15]. It has been shown that for linear classifiers, such as the Nearest Mean Classifier, the number of training samples needed in supervised classification problems is a linear function of the dimensionality and for quadratic classifiers, such as the Bayesian classifier under Gaussian hypothesis with different covariance matrices for each class, it is a quadratic function of the dimensionality [32]. In case of non parametric classifiers, the situation is even worse, since experimental results have shown that as dimensionality increases, the number of samples must grow exponentially [67,42]. Thus, in order to extract relevant information from high-dimensional data, it is required a large n, which is not always possible under certain circunstances. Therefore, a natural way to mitigate this problem and indirectly reduce the value of n is to significantly reduce the data dimensionality, that is, m.

Another negative effect caused by the curse of the dimensionality is that, in the presence of linear correlations in the data, a very likely situation in high dimensions, the optimal mean squared error when estimating the data density will be very large even if the sample size n arbitrarily increases [67].

In Bayesian decision theory, where we deal with an unlimited number of samples, since the probability distributions are assumed to be completely known a priori, the error in classification monotonically decreases as the number of features increases [32], that is, the probability of error approaches zero as dimensionality grows. On the other hand, in the real world scenario where the model parameters are estimated from data, the probability of error stays above a limiting bound as the dimensionality increases [78]. Moreover, another interesting effect of the curse of the dimensionality is the Hughes phenomena. It has been shown that for a finite number of samples n, there is an optimal dimensionality m^* beyond which the mean classification accuracy actually decreases [40]. Hence, finding the optimal number of features m^* through dimensionality reduction techniques is a crucial step for minimizing undesired errors in a pattern recognition system.

Another issue found in high-dimensional spaces is related to the weak discrimination power of a metric. As dimensionality grows, the contrast provided by usual metrics decreases, i.e., the distribution of norms in a given distribution of points tends to concentrate. This is known as the concentration phenomenon [50]. In other words, the norm of random vectors grows proportionally to $\sqrt{m}$, as naturally expected, but the variance remains more or less constant for a sufficiently large m. Therefore, high-dimensional random i.i.d. vectors seem to be distributed close to the surface of a hypersphere. In contrast with the usual 3D Euclidean space, the geometric properties of hyperspaces are highly non-intuitive. It is known that as dimensionality increases, the volume of a hyperelipsoid is concentrated in

the outer shell, that is, close to the boundaries [45]. In statistical terms, this is equivalent to have a Gaussian distribution in which data concentrates in the tails, far from the distribution mean. This is a difficult problem in multivariate density estimation, as regions of relatively very low density can contain a considerable part of the distribution, whereas regions of apparently high density can be completely devoid of observations in a sample of moderate size [70]. For a 1-D standard normal distribution $N(0,1)$, it is known that about 70% of the mass is concentrated at points contained in a sphere of radius one standard deviation (i.e. the $[-1,1]$ interval), while for a 10-D $N(0,I)$, that same hypersphere contains only 0.02% of the mass and one has to take a radius of more than 3 standard deviations to achieve approximately 70% of the mass [15]. In opposition to our intuition, in high-dimensional distributions the tails play a much more important role than in the 1-D case. Furthermore, another problematic issue related to hyperspaces is that as dimensionality increases, the diagonal vectors tend to become orthogonal to the basis vectors. The projection of a cluster in one of these "wrong directions" can possibly ruin the information within the data [45].

The key idea of dimension reduction is to find the most compact low dimensional structure that is embedded in a higher dimensional space [49]. Historically, Occam's razor has been used to justify dimension reduction. The key idea of Occam's razor is to choose the simplest model from a set of equivalent models to explain a given phenomenon [41]. There are many approaches to dimensionality reduction based on several assumptions and used in a variety of contexts. In this review, we will focus on manifold learning methods. A manifold is a topological space that is locally Euclidean (i.e., around every point, there is a neighborhood that is topologically the same as the open unit ball in R^n). A good example of a manifold is our planet. Locally, at each point on the Earth's surface, we have a 3-D coordinate system: two for location and the last one for the altitude. Globally, it is a 2-D sphere embedded in a 3-D space [41]. The intuition behind a manifold is that the observed data points lie along a low-dimensional structure embedded in a high-dimensional space, where the low-dimensional space reflects some unknown underlying parameters (i.e., local coordinates) and high-dimensional space is the feature space. Attempting to uncover this manifold structure in a dataset is referred to as manifold learning [16,96]. Manifold learning is also deeply connected to unsupervised metric learning in the sense that besides learning a more compact and meaningful representation for the observed dataset, these methods also learn a distance function that geometrically is better suited to represent a similarity measure between a pair of objects in the collection [53,84,93,8,73]. In other words, by learning the manifold structure, in general, we earn a more powerful metric for free. Furthermore, as a side note, the study of manifold learning techniques for dimensionality reduction has begun in early 2000's, with the pioneering Isometric Feature Mapping (ISOMAP) and LLE (Locally Linear Embedding) algorithms [75,61], so this year we celebrate the 20th anniversary of this remarkable research field.

The remainder of the paper is organized as follows: we begin by looking at linear approaches with the mathematical derivation of Principal Component Analysis (PCA) in Section 2, considered to be the pioneering method for dimensionality reduction. In Section 3, we introduce Kernel PCA, a generalization of PCA to produce nonlinear features that

was the predecessor of most manifold learning techniques. In Section 4, we introduce manifold learning by describing the basic problem and providing the mathematical background to ellucidate many different algorithms. Chapter 5 presents the Isometric Feature Mapping (ISOMAP) algorithm in details, a pioneering algorithm in manifold learning, as well as Multidimensional Scaling. Chapter 6 discusses the Locally Linear Embedding (LLE) algorithm. Laplacian Eigenmaps, a method based on spectral graph theory, is described in Chapter 7. Chapter 8 introduces the Locality Preserving Projections (LPP) algorithm, a linear approximation to Laplacian Eigenmaps and shows its kernel counterpart. Chapter 9 presents the Hessian Eigenmaps algorithm, which is an extension of Laplacian Eigenmaps that uses the Hessian matrix instead of the Laplacian matrix. Chapter 10 describes the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm, an iterative method based on the minimization of the KL-divergence. Semidefinte Embedding, or Maximum Variance Unfolding (MVU), is presented in Chapter 11. Chapter 12 describes the Local Tangent Space Aligment (LTSA) algorithm, which is based on the fact that in an unfolded manifold all tangent spaces are aligned. Chapter 13 describes Diffusion Maps, another variation of Laplacian Eigenmaps that uses a Markov Chain to compute a non-Euclidean metric known as diffusion distance. Finally, Section 14 presents our conclusions, final remarks and future directions for reseach in nonlinear dimensionality reduction.

2. Linear Dimensionality Reduction

Linear dimensionality reduction methods have been developed throughout many areas of science such as statistics, optimization, machine learning, and other applied fields for over a century, and these methods have become powerful mathematical tools for analyzing high dimensional and noisy data [25]. Part of the success of the linear methods comes from the fact that they have simple geometric interpretations and typically attractive computational properties [25]. Basically, in linear methods we seek a projection matrix T that maps the samples in the original feature space (R^m) to a linear subspace in R^d, with $d < m$. There are several ways to define linear dimensionality reduction methods, but here we adopt the optimization framework introduced by Cunningham and Ghahramani [25].

Definition 1. (Linear Dimensionality Reduction). Given n m-dimensional data points $X = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n] \in R^{m \times n}$ and a choice of dimensionality $d < m$, optimize some objective function $f_X(.)$ to produce a linear transformation $T \in R^{d \times m}$, and call $Y = TX \in R^{d \times n}$ the lower dimensional transformed data.

2.1. *Principal Component Analysis*

Principal Component Analysis, or simply PCA, is a computational method that implements the Karhunen-Loève transform, also known as Hotteling transform, a classical multivariate statistical technique that expands a given random vector $\vec{x} \in R^m$ in the eigenvectors of its covariance matrix [46]. PCA is the most widely known method for data compression and feature extraction. PCA does not make assumptions on probability density functions, since all information needed by the method can be estimated directly from data [36]. Since it de-

pends solely on the covariance matrix, PCA is a second order statistical method. PCA is optimal in two different ways: 1) by maximizing the variance of the new compact representation Y; 2) by minimizing the mean square error between the original data X and the new compact representation Y. From a statistical point of view, the goal of PCA is to reduce the redundancy between the random variables that compose the random vector $\vec{x} \in R^m$, which is measured by the correlations between them. In this sense, PCA first decorrelates the features and then reduce the dimensionality by finding new features that are linear combinations of the original ones.

2.2. *PCA by the Maximization of the Variance*

Let $Z = [T^T, S^T]$ be an orthonormal basis for R^m in which $T^T = [\vec{w}_1, \vec{w}_2, ..., \vec{w}_d]$ denotes the $d < m$ components that we wish to retain during the dimensionality reduction process and $S^T = [\vec{w}_{d+1}, \vec{w}_{d+2}, ..., \vec{w}_m]$ are the remaning components that should be discarded. In other words, T defines the linear PCA subspace and S defines the linear subspace eliminated by the reduction process [97].

The problem in question can be summarized as: given an input feature space, we want to find d directions $\vec{w}_j$, for $j = 1, 2, ..., d$ that, when projecting the data, the variance is maximized. In other words, we want the directions that maximizes data scattering. The question is: how to obtain the directions $\vec{w}_j$? Without loss of generality, we assume that the sample $X = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n]$ has zero mean, that is, the data points are centered around the origin.

Note that we can write $\vec{x} \in R^m$ as an expansion in the orthonormal basis Z as:

$$\vec{x} = \sum_{j=1}^{m} (\vec{x}^T \vec{w}_j)\vec{w}_j = \sum_{j=1}^{m} c_j \vec{w}_j \tag{1}$$

where c_j are the coefficients of the expansion.

Thus, the new vector $\vec{y} \in R^d$ can be obtained by the transformation $\vec{y} = T\vec{x}$, that is:

$$\vec{y}^T = \vec{x}^T T^T = \sum_{j=1}^{m} c_j \vec{w}_j^T [\vec{w}_1, \vec{w}_2, ..., \vec{w}_d] \tag{2}$$

As we have an orthonormal basis, $\vec{w}_i^T \vec{w}_j = 1$ for $i = j$ and $\vec{w}_i^T \vec{w}_j = 0$ for $i \neq j$, leading to:

$$\vec{y}^T = [c_1, c_2, ..., c_d] \tag{3}$$

In this way, a linear transformation T is sought that maximizes the variance retained in the data, that is, we want to maximize the following functional [44]:

$$J_1^{PCA}(T) = E[\|\vec{y}\|^2] = E[\vec{y}^T \vec{y}] = \sum_{j=1}^{d} E[c_j^2] \tag{4}$$

Since c_j is the projection of $\vec{x}$ in $\vec{w}_j$, that is, $c_j = \vec{x}^T \vec{w}_j$, we have:

$$J_1^{PCA}(T) = \sum_{j=1}^{d} E\left[\vec{w}_j^T \vec{x}\vec{x}^T \vec{w}_j\right] = \sum_{j=1}^{d} \vec{w}_j^T E\left[\vec{x}\vec{x}^T\right] \vec{w}_j = \sum_{j=1}^{d} \vec{w}_j^T \Sigma_x \vec{w}_j \tag{5}$$

where Σ_x denotes the covariance matrix of the data points X.

Hence, we have the following constrained optimization problem:

$$\arg\max_{\vec{w}_j} \sum_{j=1}^{d} \vec{w}_j^T \Sigma_x \vec{w}_j \qquad \text{subject to} \quad \left\|\vec{w}_j\right\| = 1 \quad \text{for} \quad j = 1, 2, ..., d \tag{6}$$

which is solved by Lagrange mutipliers. The Lagrangian function is given by:

$$J_1^{PCA}(T, \lambda_1, \lambda_2, ..., \lambda_d) = \sum_{j=1}^{d} \vec{w}_j^T \Sigma_x \vec{w}_j - \sum_{j=1}^{d} \lambda_j \left(\vec{w}_j^T \vec{w}_j - 1\right) \tag{7}$$

Differentiating with respect to $\vec{w}_j$ and setting the result to zero gives us the necessary condition for the optimum:

$$\frac{\partial}{\partial \vec{w}_j} J_1^{PCA}(T, \lambda_1, \lambda_2, ..., \lambda_d) = \Sigma_x \vec{w}_j - \lambda_j \vec{w}_j = 0 \tag{8}$$

which leads to the eigenvector equation:

$$\Sigma_x \vec{w}_j = \lambda_j \vec{w}_j \tag{9}$$

Going back to the optimization problem, we can rewrite it as:

$$\arg\max_{\vec{w}_j} \sum_{j=1}^{d} \vec{w}_j^T \Sigma_x \vec{w}_j = \arg\max_{\vec{w}_j} \sum_{j=1}^{d} \vec{w}_j^T \lambda_j \vec{w}_j = \arg\max_{\vec{w}_j} \sum_{j=1}^{d} \lambda_j \tag{10}$$

which means that we should select to compose the basis of the linear PCA subspace the k eigenvectors associated to the k largest eigenvalues of the data covariance matrix.

2.3. *PCA by the Minimization of the MSE*

Another optimal property of the PCA subspace is that it minimizes the mean square error between X and Y. In other words, the PCA approximation is the best representation in terms of data compression. Let the mean square error between a random vector $\vec{x} \in R^m$ and a random vector $\vec{y} \in R^d$ be:

$$J_2^{PCA}(T) = E\left[\|\vec{x} - \vec{y}\|^2\right] = E\left[\left\|\vec{x} - \sum_{j=1}^{d} (\vec{w}_j^T \vec{x})\vec{w}_j\right\|^2\right] \tag{11}$$

Expanding the norm, we have the following expression for the mean square error:

$$J_2^{PCA}(T) = E\left[\left(\vec{x} - \sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right)^T\left(\vec{x} - \sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right)\right] \tag{12}$$

Applying the distributive property leads to:

$$\begin{aligned} J_2^{PCA}(T) = E\Bigg[\vec{x}^T\vec{x} - \vec{x}^T\left(\sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right) - \left(\sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right)^T\vec{x} \\ + \left(\sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right)\left(\sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right)^T\Bigg] \end{aligned} \tag{13}$$

Using the linearity of the expected value operator and rearranging the terms:

$$\begin{aligned} J_2^{PCA}(T) = E\left[\|\vec{x}\|^2\right] - E\left[\sum_{j=1}^{d}(\vec{w}_j^T\vec{x})(\vec{w}_j^T\vec{x})\right] - E\left[\sum_{j=1}^{d}\vec{w}_j^T(\vec{x}^T\vec{w}_j)\vec{x}\right] \\ + E\left[\left(\sum_{j=1}^{d}\vec{w}_j^T(\vec{x}^T\vec{w}_j)\right)\left(\sum_{j=1}^{d}(\vec{w}_j^T\vec{x})\vec{w}_j\right)\right] \end{aligned} \tag{14}$$

Simplifying the dot products, we have:

$$\begin{aligned} J_2^{PCA}(T) = E\left[\|\vec{x}\|^2\right] - \sum_{j=1}^{d}E\left[(\vec{w}_j^T\vec{x})^2\right] - \sum_{j=1}^{d}E\left[(\vec{w}_j^T\vec{x})^2\right] \\ + \sum_{j=1}^{d}\sum_{k=1}^{d}E\left[(\vec{w}_j^T\vec{x})(\vec{x}^T\vec{w}_k)\vec{w}_j^T\vec{w}_k\right] \end{aligned} \tag{15}$$

As $\vec{w}_j$ for $j = 1, 2, ..., d$ defines a set of orthonormal vectors:

$$\begin{aligned} J_2^{PCA}(T) &= E\left[\|\vec{x}\|^2\right] - \sum_{j=1}^{d}E\left[(\vec{w}_j^T\vec{x})^2\right] - \sum_{j=1}^{d}E\left[(\vec{w}_j^T\vec{x})^2\right] + \sum_{j=1}^{d}E\left[(\vec{w}_j^T\vec{x})^2\right] \\ &= E\left[\|\vec{x}\|^2\right] - \sum_{j=1}^{d}E\left[(\vec{w}_j^T\vec{x})^2\right] \\ &= E\left[\|\vec{x}\|^2\right] - \sum_{j=1}^{d}\vec{w}_j^T\Sigma_x\vec{w}_j \end{aligned} \tag{16}$$

Note that the first term is a constant (it does not depend on $\vec{w}_j$), so the optimization problem is given by:

$$\arg\min_{\vec{w}_j} \; -\sum_{j=1}^{d} \vec{w}_j^T \Sigma_x \vec{w}_j \qquad \text{subject to} \quad \left\|\vec{w}_j\right\| = 1 \quad \text{for} \quad j = 1,2,...,d \tag{17}$$

which is equivalent to the maximization of the variance.

As mentioned before, an interesting property of PCA is that it decorrelates the data. This can be easily seen by using the eigendecomposition of Σ_x in $Q\Lambda Q^T$, where Q is the matrix of the m eigenvectors of Σ_x and $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_m)$ is a diagonal matrix of the eigenvalues of Σ_x. We know that prior to the dimensionality reduction $\vec{y} = Z\vec{x}$, where $Z = [T^T, S^T]$ and the covariance matrix of the transformed vector is given by $\Sigma_y = Z^T \Sigma_x Z$. But in PCA, Z is composed by the eigenvectors of the covariance matrix, so $Z = Q$, leading to $\Sigma_y = Z^T Q\Lambda Q^T Z = Q^T Q\Lambda Q^T Q = \Lambda$, where the orthonormality of the eigenvectors implies in $Q^T Q = I$.

2.4. *Geometric Interpretation of PCA*

In the PCA transformation, first the data points are centralized, by subtracting the mean from each sample in X. Then, the random vector $\vec{x} \in R^m$ is projected into the eigenvalues of the covariance matrix Σ_x, in a way that the redundancy introduced by the correlation between the components of $\vec{x}$ is eliminated. Geometrically, such condition is achieved through a rotation of the coordinate axes. In this process, the variances of the projections of $\vec{x}$ in the new axes are maximized and discarding the directions that have less variance makes the mean square error minimum.

Under the multivariate Gaussian hypothesis, the rotated coordinate space correspond to the space composed by the principal axes of the hiperellipsoid that define the distribution. Figure 1, adapted from [32], shows an example for the 2D Gaussian case. The principal components are the projections of the data into the two axes, ϕ_1 and ϕ_2. The variances, denoted here by λ_1 and λ_2, are distinct in most problems and a large number of them is so small that the corresponding components can be discarded. In the following, we present an algorithm for dimensionality reduction by PCA.

Algorithm 1 Principal Component Analysis

1: **function** PCA(X)
2: Compute the sample mean and the sample covariance matrix by:
$\mu_x = \frac{1}{n}\sum_{i=1}^{n} \vec{x}_i$
$\Sigma_x = \frac{1}{n-1}\sum_{i=1}^{n} (\vec{x}_i - \mu_x)(\vec{x}_i - \mu_x)^T$
3: Compute the eigenvalues and eigenvectors of Σ_x
4: Define the transformation matrix $T = [\vec{w}_1, \vec{w}_2, ..., \vec{w}_d]$ with the d eigenvetors associated to the d largest eigenvalues.
5: Project the data X into the PCA subspace:
$\vec{y}_i = T\vec{x}_i \quad \text{for} \quad i = 1,2,...,n$
6: **return** Y
7: **end function**

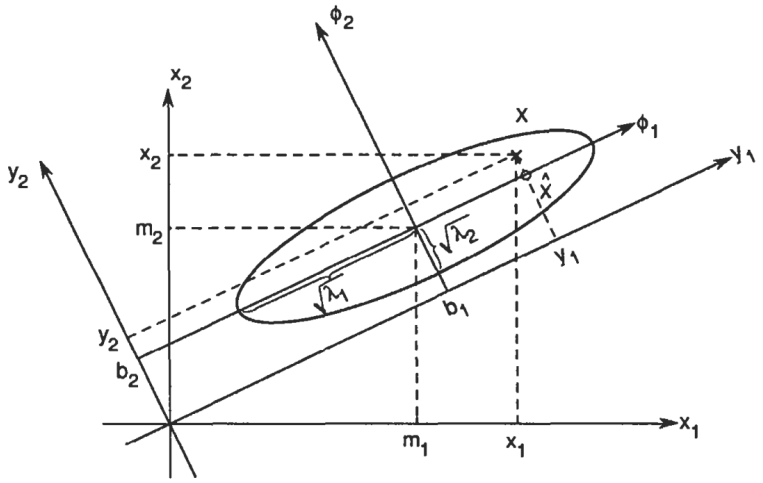

Figure 1. An illustration of the Karhunen-Loeve transform for a 2D Gaussian model.

2.5. *Beyond PCA*

There are many extensions to PCA, among which we can mention PPCA or Probabilistic PCA [77,60], where the definition of a likelihood measure enables a comparison with other probabilistic techniques, while facilitating statistical testing and allowing the application of Bayesian models, and robust PCA, where the main goal is to introduce outlier insensitivity via different vector norms [34,18]. PCA suffers from the fact that each principal component is a linear combination of all the original variables, thus it is often difficult to interpret the results. To overcome this issue, PCA has been made sparse (Sparse PCA) in several contexts to allow features with sparse loadings, that is, each principal component depends only on few variables [100,47].

In the last decades, several PCA variations were proposed to mitigate the limitations of the original method. Kernel PCA is a nonlinear generalization that corresponds to PCA performed in a reproducing kernel Hilbert space associated with a positive definite kernel [66]. While PCA is optimal in minimizing the mean squared error, it is still sensitive to outliers in the data. It is a common practice to remove outliers before PCA. However, outliers can be difficult to identify. Robust PCA is a PCA generalization developed to deal with the presence of outliers and random noise in data [94]. Another issue with PCA is that every feature is a linear combination of all input variables. In multilinear subspace learning, dimensionality reduction is performed on a data tensor. Multilinear PCA extracts features directly from these tensor representations by performing PCA in each mode of the tensor iteratively [51].

Independent Component Analysis (ICA) is a class of methods that uses higher-order statistics to go beyond the uncorrelatedness of PCA [44]. The maximization of the non-Gaussianity is motivated by the Central Limit Theorem, which states that the distribution

of the sum (average or linear combination) of n independent random variables approaches Gaussian as n grows. In this context, non-Gaussianity means independence. This is the base for the FastICA algorithm [43].

A very popular approach to estimating independent components is the model of maximum likelihood, since it is a fundamental technique in the statistical theory used to solve several problems. Often, numerical optimization algorithms such as gradient descent are applied to find the solution. This is the idea behind the Infomax algorithm [7,52]. A natural approach to ICA is the minimization of mutual information, which is an information-theoretic measure of dependence between random variables, which is always nonnegative, and zero, if and only if, the variables are statistically independent, that is, it can be used as an objective function in the ICA estimation [1,59]. Additionally, mutual informtion can be interpreted as a distance, that is, as the Kullback-Leibler divergence between the joint distribution of $\vec{x}$ and the product of the marginals distributions of x_i, for $i = 1, 2, .., n$.

3. Kernel PCA

Principal Component Analysis only allows linear dimensionality reduction. However, if the data has more complicated structures which are nonlinear functions of the original features, standard PCA will fail in capturing meaningful information. Fortunately, kernel PCA allows us to generalize standard PCA to nonlinear dimensionality reduction [66].

The Vapnik-Chervonenkis theory shows that under certain circumstances mappings which take us into a higher dimensional space than the dimension of the input space often provide us with greater classification power [82]. We consider a nonlinear mapping $\phi(\vec{x})$ from the original m-dimensional input space to a M-dimensional feature space, where $M > m$. However, high-dimensional mappings can seriously increase the computational cost. Fortunately, we can make use of the kernel trick: given any algorithm that can be expressed solely in terms of dot products, this trick allows us to construct different nonlinear versions of it [76]. With the kernel trick we can compute the inner product in a higher dimensional space, without the need of projecting the data itself. That the key idea behind kernel PCA. It allows us to extract up to n (number of samples) nonlinear principal components without expensive computations [65].

The first assumption is that the mean of the data after the mapping to the high-dimensional space is zero, that is:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(\vec{x}_i) = 0 \tag{18}$$

Thus, the $M \times M$ sample covariance matrix of the projected data is given by:

$$C = \frac{1}{n}\sum_{i=1}^{n}\phi(\vec{x}_i)\phi(\vec{x}_i)^T \tag{19}$$

and the eigenvectors of C are:

$$C\vec{v}_k = \lambda_k \vec{v}_k \quad \text{for} \quad k = 1,2,...,M \tag{20}$$

The following result show that we can write the eigenvalues of the covariance matrix in terms of $\phi(\vec{x}_i)$.

Theorem 1. *The eigenvectors of C can be expressed as a linear combination of the features, that is:*

$$\vec{v}_k = \sum_{i=1}^{n} \alpha_{ki} \phi(\vec{x}_i) \tag{21}$$

Note that from equations (19) and (20), we have:

$$C\vec{v}_k = \frac{1}{n} \sum_{i=1}^{n} \phi(\vec{x}_i)\phi(\vec{x}_i)^T \vec{v}_k = \lambda \vec{v}_k \tag{22}$$

which implies in:

$$\vec{v}_k = \frac{1}{n\lambda_k} \sum_{i=1}^{n} (\phi(\vec{x}_i)^T \vec{v}_k)\phi(\vec{x}_i) = \sum_{i=1}^{n} \alpha_{ki} \phi(\vec{x}_i) \tag{23}$$

where $\alpha_{ki} = \frac{1}{n\lambda_k} \phi(\vec{x}_i)^T \vec{v}_k$. So, finding the eigenvectors is equivalent to finding the coefficients α_{ki}. By substituting back equation (23) into equation (22), we have:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\vec{x}_i)\phi(\vec{x}_i)^T \left(\sum_{j=1}^{n} \alpha_{kj} \phi(\vec{x}_j) \right) = \lambda_k \sum_{j=1}^{n} \alpha_{kj} \phi(\vec{x}_j) \tag{24}$$

Rewritting equation (24), we can express it as:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\vec{x}_i) \left(\sum_{j=1}^{n} \alpha_{kj} \phi(\vec{x}_i)^T \phi(\vec{x}_j) \right) = \lambda_k \sum_{j=1}^{n} \alpha_{kj} \phi(\vec{x}_j) \tag{25}$$

And using the kernel trick, that is, $K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T \phi(\vec{x}_j)$, we have:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\vec{x}_i) \left(\sum_{j=1}^{n} \alpha_{kj} K(\vec{x}_i, \vec{x}_j) \right) = \lambda_k \sum_{j=1}^{n} \alpha_{kj} \phi(\vec{x}_j) \tag{26}$$

Multiplying both sides by $\phi(\vec{x}_l)^T$ leads to:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\vec{x}_l)^T \phi(\vec{x}_i) \left(\sum_{j=1}^{n} \alpha_{kj} K(\vec{x}_i, \vec{x}_j) \right) = \lambda_k \sum_{j=1}^{n} \alpha_{kj} \phi(\vec{x}_l)^T \phi(\vec{x}_j) \tag{27}$$

Using the kernel trick once again, we have:

$$\frac{1}{n}\sum_{i=1}^{n} K(\vec{x}_l,\vec{x}_i)\left(\sum_{j=1}^{n}\alpha_{kj}K(\vec{x}_i,\vec{x}_j)\right) = \lambda_k \sum_{j=1}^{n}\alpha_{kj}K(\vec{x}_l,\vec{x}_j) \tag{28}$$

Using the matrix vector notation we can express the equation as [66]:

$$K^2\vec{\alpha}_k = (\lambda_k n)K\vec{\alpha}_k \tag{29}$$

where $K_{i,j} = K(\vec{x}_i,\vec{x}_j)$ and $\vec{\alpha}_k$ is the n-dimensional column vector of α_{ki}, that is, $\vec{\alpha}_k = [\alpha_{k1},\alpha_{k2},...,\alpha_{kn}]^T$. Simplifying equation (29), we finally reach:

$$K\vec{\alpha}_k = (\lambda_k n)\vec{\alpha}_k \tag{30}$$

showing that the $\vec{\alpha}_k$ are the eigenvectors of the kernel matrix. We have a normalization condition for the $\vec{\alpha}_k$ eigenvectors. First, we know that $\vec{v}_k^T\vec{v}_k = 1$, which implies:

$$\sum_{r=1}^{n}\sum_{s=1}^{n}\alpha_{kr}\alpha_{ks}\phi(\vec{x}_r)^T\phi(\vec{x}_s) = 1 \quad \rightarrow \quad \vec{\alpha}_k^T K\vec{\alpha}_k = 1 \tag{31}$$

But multiplying equation (30) by $\vec{\alpha}_k^T$ leads to:

$$\vec{\alpha}_k^T K\vec{\alpha}_k = (\lambda_k n)\vec{\alpha}_k^T\vec{\alpha}_k \quad \rightarrow \quad (\lambda_k n)\vec{\alpha}_k^T\vec{\alpha}_k = 1 \quad \rightarrow \quad \vec{\alpha}_k^T\vec{\alpha}_k = \frac{1}{n\lambda_k} \tag{32}$$

For a new point $\vec{x}$, its projection onto the k-th principal component is given by:

$$y_k(\vec{x}) = \phi(\vec{x})^T\vec{v}_k = \sum_{i=1}^{n}\alpha_{ki}\phi(\vec{x})^T\phi(\vec{x}_i) = \sum_{i=1}^{n}\alpha_{ki}K(\vec{x},\vec{x}_i) \tag{33}$$

The advantage of employing the kernel trick is that we do not have to compute $\phi(\vec{x}_i)$ explicitly for $i = 1,2,...,n$. We can directly construct the kernel matrix from the training data. Two widely used nonlinear kernels are the polynomial kernel:

$$K(\vec{x},\vec{y}) = (\vec{x}^T\vec{y} + c)^d \tag{34}$$

where $c >= 0$ is a constant, and the Gaussian kernel:

$$K(\vec{x},\vec{y}) = exp\left(-\frac{\|\vec{x}-\vec{y}\|^2}{2\sigma^2}\right) \tag{35}$$

with parameter σ^2. In case the projected data does not have zero mean, we need to centralize the data making:

$$\tilde{\phi}(\vec{x}_i) = \phi(\vec{x}_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(\vec{x}_k) \tag{36}$$

Hence, the corresponding kernel matrix is given by:

$$\begin{aligned}\tilde{K}(\vec{x}_i,\vec{x}_j) = \tilde{\phi}(\vec{x}_i)^T\tilde{\phi}(\vec{x}_i) &= \left(\phi(\vec{x}_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(\vec{x}_k)\right)^T\left(\phi(\vec{x}_j) - \frac{1}{n}\sum_{k=1}^{n}\phi(\vec{x}_k)\right)\\ &= \phi(\vec{x}_i)^T\phi(\vec{x}_j) - \frac{1}{n}\sum_{k=1}^{n}\phi(\vec{x}_i)^T\phi(\vec{x}_k) - \frac{1}{n}\sum_{k=1}^{n}\phi(\vec{x}_k)^T\phi(\vec{x}_j)\\ &\quad + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n}\phi(\vec{x}_k)^T\phi(\vec{x}_l)\\ &= K(\vec{x}_i,\vec{x}_j) - \frac{1}{n}\sum_{k=1}^{n}K(\vec{x}_i,\vec{x}_k) - \frac{1}{n}\sum_{k=1}^{n}K(\vec{x}_k,\vec{x}_j)\\ &\quad + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n}K(\vec{x}_k,\vec{x}_l)\end{aligned} \tag{37}$$

In matrix form, we have to replace the kernel matrix K by the Gram matrix $\tilde{K}$:

$$\tilde{K} = K - 1_n K - K 1_n + 1_n K 1_n \tag{38}$$

where 1_n is the $n \times n$ matrix with all elements equal to $\frac{1}{n}$. In the following, we present an algorithm for dimensionality reduction through kernel PCA.

Algorithm 2 Kernel Principal Component Analysis

1: **function** KPCA(X)
2: Construct the kernel matrix K from the training dataset X: $K_{i,j} = K(\vec{x}_i,\vec{x}_j)$
3: Compute the Gram matrix $\tilde{K}$ using equation (38)
4: Use equation (30) to solve for the vectors $\vec{\alpha}_k$ (using $\tilde{K}$ instead of K)
5: Compute the kernel principal components $y_k(\vec{x})$ using equation (33) for $k = 1,2,...,d$
6: **return** Y
7: **end function**

4. Manifold Learning

Nonlinear dimensionality reduction involves finding low-dimensional representations in high-dimensional space. This problem arises naturally when analyzing data like hyperspectral imagens, human faces, speech waveforms and handwritten characters. Previous algorithms like Principal Component Analysis, Independent Component Analysis and Non-Negative Matrix Factorization often fail to capture the hidden non-linear representation of the data. It has been found that manifolds are used to explain how our visual perception and learning work by recognizing the variability of perceptual stimuli and other types of high-dimensional data [57,54,68]. In order to present manifold learning algorithms for non-linear dimensionality reduction, we first need to introduce the problem, which requires

some preliminary definitions. The question is: from a mathematical point of view, what is a manifold? There are three definitions that help to answer this question [16].

Definition 2. A homeomorfism is a continuos function whose inverse is also a continuos function.

Two spaces with a homeomorphism between them are called homeomorphic, and from a topological viewpoint they are the same.

Definition 3. A d-dimensional manifold M is a set that is locally homeomorphic with R^d. For each $x \in M$, there is an open neighborhood around x, N_x, and a homeomorphism $f : N_x \rightarrow R^d$. These neighborhoods are referred to as coordinate patches, and the map is referred to a coordinate chart. The image of the coordinate charts is referred to as the parameter space.

Manifolds are conceptual objects widely studied by mathematicians and play a central role in many parts of geometry and modern mathematical physics. Briefly speaking, a manifold is a topological space that locally resembles Euclidean space near each point. For example, every surface is a 2D manifold.

In the dimensionality reduction approach, the interest is particularly in the case where M is a subset of R^m (the input space), where $m >> d$, with d being the intrinsic dimensionality, that is, the manifold will lie in a high-dimensional space (R^m), but will be homeomorphic with a low-dimensional space (R^d). Moreover, manifold learning algorithms require some smootness constraints [16].

Definition 4. A smooth or differentiable manifold is a manifold such that each coordinate chart is differentiable with a differentiable inverse (i.e., each coordinate chart is a diffeomorphism).

An embedding of a manifold M into R^d is then a smooth homeomorphism from M to a subset of R^d. The algorithms presented in this section find embeddings of a discrete dataset $X = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n]$, where $\vec{x}_i \in R^m$, to a set of points in another subspace of R^d. The classic example is the unfold of the swiss roll. Figure 2 shows the classic dataset swiss roll, and how, despite the points have originally three coordinates, they can be represented as a 2D plane, by the colormap. One of the most important features in manifold learning algorithms is that points close to each other in the manifold must remain close in the low-dimensional representation. Clearly, linear methods such as PCA will fail in unfolding the data, that is in finding a meaningful low-dimensional representation.

In other words, the goal is to chart a manifold. Charting is the task of assigning a low-dimensional coordinate system to data points in a high-dimensional sample space. The assumptions are that the data lies on or near a low dimensional manifold embedded in the sample space, and that there exists a one to one smooth nonlinear transform between the manifold and a low-dimensional vector space [13]. More formally, the manifold learning problem can be described as follows [16].

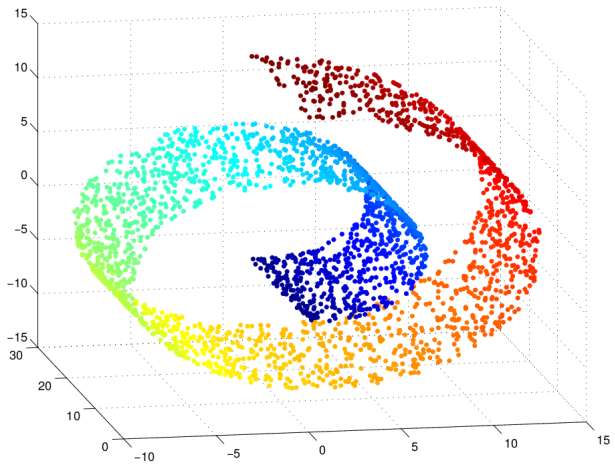

Figure 2. In the swiss roll dataset each sample is originally 3D, but with manifold learning algorithms it is possible to express each point using only two coordinates by unfolding the data.

Definition 5. (Manifold Learning Problem) Given points $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n \in R^m$ that lie on or near a d-dimensional manifold M that can be described by a single coordinate chart $f : M \to R^d$, with $d < m$, find $\vec{y}_1, \vec{y}_2, ..., \vec{y}_n \in R^d$ where $\vec{y}_i = f(\vec{x}_i)$ for $i = 1, 2, .., n$.

In the upcoming sections we will present a review of several algorithms for solving the manifold learning problem, focusing on how each one of them achieves the solution being sought.

5. Isometric Feature Mapping (ISOMAP)

ISOMAP was one of the pioneering algorithms in manifold learning for dimensionality reduction. The authors propose an approach that combines the major algorithmic features of PCA and Multidimensional Scaling [24,12] (MSD) - computational efficiency, global optimality, and asymptotic convergence guarantees - with the flexibility to learn a broad class of nonlinear manifolds [75]. The basic idea of the ISOMAP algorithm is first to build a graph by joining the k-nearest neighbors (KNN) in the input space, then compute the shortest paths between each pair of vertices in the graph and, knowing the approximate geodesic distances between the points, find a mapping to the an Euclidian subspace of R^d that preserves those distances.

The hypothesis of the ISOMAP algorithm is that the shortest paths in the KNN graph are good appproximations for the true geodesic distances in the manifold. It has been shown that for both ε-neighborhood rule and KNN-rule based graphs, under certain regularity conditions, the following result is valid [10].

Theorem 2. *(Asymptotic Convergence Theorem) Given* $\lambda_1, \lambda_2, \mu > 0$ *then for a sufficiently*

large number of samples $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n \in R^m$ *the following inequality:*

$$(1-\lambda_1)d_M(\vec{x}_i,\vec{x}_j) \leq d_G(\vec{x}_i,\vec{x}_j) \leq (1+\lambda_2)d_M(\vec{x}_i,\vec{x}_j) \tag{39}$$

is satisfied with probability $(1-\mu)$*, where* $d_G(\vec{x}_i,\vec{x}_j)$ *is the shortest path approximation in the graph and* $d_M(\vec{x}_i,\vec{x}_j)$ *is the geodesic distance in the manifold.*

The ISOMAP algorithm can be divided in three main steps:

(1) From the input data $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n \in R^m$ build an undirected proximity graph using the KNN rule or the ε-neighborhood rule [83];
(2) Compute the pairwise distance matrix D using n executions of the Dijkstra's algorithm or one execution of the Floyd-Warshall algorithm [23];
(3) Estimate the new coordinates of the points in an Euclidean subspace of R^d by preserving the distances through the Multidimensional Scaling (MDS) method.

It is clear from the algorithm that the embedding is built by the MDS method, which plays a central role in ISOMAP. For this reason, in the following, we provide a detailed mathematical description of MDS, showing how it recover the points $\vec{y}_1, \vec{y}_2, ..., \vec{y}_n \in R^d$ from the distance matrix D obtained in step two.

5.1. *Multidimensional Scaling*

Basically, the main goal of MDS is, given an $n \times n$ matrix of pairwise distances, recover the coordinates of the n points $\vec{x}_r \in R^d$ for $r = 1, 2, ..., n$ in an Euclidean subspace, where d, the target dimensionality, is a parameter of the algorithm [24,12].

We begin by noting that the pairwise distance matrix is given by $D = \{d_{rs}^2\}$, for $r, s = 1, 2, ..., n$ where the distance between two arbitrary points $\vec{x}_r$ and $\vec{x}_s$ is:

$$d_{rs}^2 = \|\vec{x}_r - \vec{x}_s\|^2 = (\vec{x}_r - \vec{x}_s)^T(\vec{x}_r - \vec{x}_s) \tag{40}$$

Let B denote the inner products matrix, that is $B = \{b_{rs}\}$, where $b_{rs} = \vec{x}_r^T \vec{x}_s$. To find the embedding, MDS needs the matrix B, not D. Knowing this, we can raise two relevant questions:

(1) How to find the matrix B from the distance matrix D?
(2) How to recover the coordinate of the points from the matrix B?

In the following, we provide answers for both questions.

5.2. *Finding the matrix* B

In answering the first question, we need to assume a hypothesis that the data has zero mean, that is:

$$\sum_{r=1}^{n} \vec{x}_r = 0 \tag{41}$$

otherwise there would be infinitely many different solutions, since the application of any arbitrary translation in the set of points, would preserve the pairwise distances.

From equation (40), applying the distributive law we have:

$$d_{rs}^2 = \vec{x}_r^T \vec{x}_r + \vec{x}_s^T \vec{x}_s - 2\vec{x}_r^T \vec{x}_s \tag{42}$$

From the matrix D, we can calculate the mean of an arbitrary column s by:

$$\begin{aligned}\frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 &= \frac{1}{n}\sum_{r=1}^{n} \vec{x}_r^T \vec{x}_r + \frac{1}{n}\sum_{r=1}^{n} \vec{x}_s^T \vec{x}_s - 2\frac{1}{n}\sum_{r=1}^{n} \vec{x}_r^T \vec{x}_s \\ &= \frac{1}{n}\sum_{r=1}^{n} \vec{x}_r^T \vec{x}_r + \frac{n}{n}\vec{x}_s^T \vec{x}_s - 2\vec{x}_s \frac{1}{n}\sum_{r=1}^{n} \vec{x}_r^T \\ &= \frac{1}{n}\sum_{r=1}^{n} \vec{x}_r^T \vec{x}_r + \vec{x}_s^T \vec{x}_s \end{aligned} \tag{43}$$

Similarly, we can compute the mean of an arbitrary row r as:

$$\begin{aligned}\frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 &= \frac{1}{n}\sum_{s=1}^{n} \vec{x}_r^T \vec{x}_r + \frac{1}{n}\sum_{s=1}^{n} \vec{x}_s^T \vec{x}_s - 2\frac{1}{n}\sum_{s=1}^{n} \vec{x}_r^T \vec{x}_s \\ &= \frac{n}{n}\vec{x}_r^T \vec{x}_r + \frac{1}{n}\sum_{s=1}^{n} \vec{x}_s^T \vec{x}_s - 2\vec{x}_r^T \frac{1}{n}\sum_{s=1}^{n} \vec{x}_s \\ &= \vec{x}_r^T \vec{x}_r + \frac{1}{n}\sum_{s=1}^{n} \vec{x}_s^T \vec{x}_s \end{aligned} \tag{44}$$

Finally, we can compute the mean of all elements of D as:

$$\begin{aligned}\frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2 &= \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} \vec{x}_r^T \vec{x}_r + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} \vec{x}_s^T \vec{x}_s - 2\frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} \vec{x}_r^T \vec{x}_s \\ &= \frac{1}{n}\sum_{r=1}^{n} \vec{x}_r^T \vec{x}_r + \frac{1}{n}\sum_{s=1}^{n} \vec{x}_s^T \vec{x}_s = \frac{2}{n}\sum_{r=1}^{n} \vec{x}_r^T \vec{x}_r \end{aligned} \tag{45}$$

Note that from equation (42), it is possible to define b_{rs} as:

$$b_{rs} = \vec{x}_r^T \vec{x}_s = -\frac{1}{2}(d_{rs}^2 - \vec{x}_r^T \vec{x}_r - \vec{x}_s^T \vec{x}_s) \tag{46}$$

But from equation (44) we can isolate the term $-\vec{x}_r^T \vec{x}_r$ as:

$$-\vec{x}_r^T \vec{x}_r = -\frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 + \frac{1}{n}\sum_{s=1}^{n} \vec{x}_s^T \vec{x}_s \tag{47}$$

And from equation (43) we can isolate the term $-\vec{x}_s^T \vec{x}_s$ as:

$$-\vec{x}_s^T\vec{x}_s = -\frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 + \frac{1}{n}\sum_{r=1}^{n}\vec{x}_r^T\vec{x}_r \tag{48}$$

Now making equation (47) minus equation (48) leads to:

$$-\vec{x}_r^T\vec{x}_r - \vec{x}_s^T\vec{x}_s = -\frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 - \frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 + \frac{2}{n}\sum_{r=1}^{n}\vec{x}_r^T\vec{x}_r \tag{49}$$

From equation (45) we know that:

$$\frac{2}{n}\sum_{r=1}^{n}\vec{x}_r^T\vec{x}_r = \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2 \tag{50}$$

Finally, we can express an arbitrary b_{rs} as a function of the elements of the pairwise distaces matrix D as:

$$b_{rs} = -\frac{1}{2}\left(d_{rs}^2 - \frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 - \frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2\right) \tag{51}$$

Making $a_{rs} = -\frac{1}{2}d_{rs}$ we can write:

$$a_{r.} = \frac{1}{n}\sum_{s=1}^{n} a_{rs} \qquad a_{.s} = \frac{1}{n}\sum_{r=1}^{n} a_{rs} \qquad a_{..} = \frac{1}{n}\sum_{r=1}^{n}\sum_{s=1}^{n} a_{rs} \tag{52}$$

Thus, we can express b_{rs} as:

$$b_{rs} = a_{rs} - a_{r.} - a_{.s} + a_{..} \tag{53}$$

Defining the matrix $A = \{a_{rs}\}$, for $r,s = 1,2,...,n$ as $A = -\frac{1}{2}D$ and the matrix H as:

$$H = I - \frac{1}{n}\vec{1}\vec{1}^T \tag{54}$$

where $\vec{1}^T = [1,1,...,1]_n$ so we have:

$$\vec{1}\vec{1}^T = U = \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & 1 & 1 & ... & 1 \\ . & . & . & ... & . \\ 1 & 1 & 1 & ... & 1 \end{bmatrix}_{n\times n} \tag{55}$$

it is possible to compute all values of the matrix B simultaneously by $B = HAH$. To see this, note that:

$$B = HAH = \left(I - \frac{1}{n}U\right)A\left(I - \frac{1}{n}U\right) = A - A\frac{U}{n} - \frac{U}{n}A + \frac{1}{n^2}UAU \tag{56}$$

which is the matrix form of equation (53).

5.3. *Recovering the coordinates of the points*

At this level, the problem is: how to find the embedding, that is, the coordinates of the points in R^d? First, it is easy to see that inner products matrix B can be written as:

$$B_{n\times n} = X^T_{n\times m} X_{m\times n} \tag{57}$$

where n and m denote the number of samples and the input dimensionality, repectively and $X_{n\times m} = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n]$ is our data matrix. In summary, it has been shown that the matrix B has three important properties [24]:

- It is symmetric
- The rank of B is m
- It is positive semidefinite.

In other words, this means that the matrix B has m non-negative eigenvalues and $n-m$ zero eigenvalues. Thus, by the eigendecomposition of B, we have:

$$B = V\Lambda V^T \tag{58}$$

where $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_n)$ is the diagonal matrix with the eigenvalues of B and V is the matrix whose columns are the eigenvectors of B:

$$V = \begin{bmatrix} | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \\ \vec{v}_1 & \vec{v}_2 & \dots & \dots & \vec{v}_n \\ | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \end{bmatrix}_{n\times n} \tag{59}$$

Without loss of generality, we can assume that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$. Due to the $n-m$ null eigenvalues, the matrix B can be expressed by:

$$B = \tilde{V}\tilde{\Lambda}\tilde{V}^T \tag{60}$$

where $\tilde{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_m)$ is the diagonal matrix of the non-zero eigenvalues of B and $\tilde{V}$ is the $n \times m$ matrix whose columns are the m eigenvectors associated to the m non-zero eigenvalues:

$$\tilde{V} = \begin{bmatrix} | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \\ \vec{v}_1 & \vec{v}_2 & \dots & \dots & \vec{v}_m \\ | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \end{bmatrix}_{n\times m} \tag{61}$$

We now have the following identity regarding the B matrix:

$$B = X^T X = \tilde{V}\tilde{\Lambda}\tilde{V}^T = \tilde{V}\tilde{\Lambda}^{1/2}\tilde{\Lambda}^{1/2}\tilde{V}^T \tag{62}$$

which finally means that:

$$X = \tilde{\Lambda}^{1/2}\tilde{V}^T \tag{63}$$

where $\tilde{\Lambda}^{1/2} = diag(\sqrt{\lambda_1}, \sqrt{\lambda_2}, ..., \sqrt{\lambda_m})$. In practice, we choose the intrinsic dimensionality, d, a parameter of the algorithm, to be smaller than the dimensionality of the input data, m, so each column of X will represent a sample in the manifold. Usually, we select to compose the matrix $\tilde{V}$ the $d < m$ eigenvectors associated to the top d eigenvalues. Therefore, the algorithm returns a $d \times n$ data matrix which is a more compact representation of the input data. Algorithm 3 summarizes the whole process in a sequence of logical and objective steps.

Algorithm 3 Isometric Feature Mapping

1: **function** ISOMAP(X)
2: From the input data $X_{m\times n}$ build a KNN graph.
3: Compute the pairwise distances matrix $D_{n\times n}$.
4: Compute $A = -\frac{1}{2}D$.
5: Compute $H = I - \frac{1}{n}U$, where U is a $n \times n$ matrix of 1's.
6: Compute $B = HAH$.
7: Find the eigenvalues and eigenvectors of the matrix B.
8: Select the top $d < m$ eigenvalues and eigenvalues of B and define:

$$\tilde{V} = \begin{bmatrix} | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \\ \vec{v}_1 & \vec{v}_2 & \dots & \dots & \vec{v}_d \\ | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \end{bmatrix}_{n\times d} \tag{64}$$

$$\tilde{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_d) \tag{65}$$

9: Compute $\tilde{X} = \tilde{\Lambda}^{1/2}\tilde{V}^T$
10: **return** $\tilde{X}$
11: **end function**

The ISOMAP algorithm is not perfect and has its problems. The connectivity of each data point in the KNN graph is defined as its Euclidean k nearest neighbors in the high-dimensional space. It has been shown that ISOMAP is vulnerable to "short-circuit errors" if k is too large with respect to the manifold structure or if noise in the data moves the points slightly off the manifold [3]. Even a single short-circuit error can alter many entries

in the geodesic distance matrix D, which can lead to a totally different low-dimensional embedding. Conversely, if k is too small, the neighborhood graph may become too sparse to approximate geodesic paths accurately. But improvements have been made to this algorithm to make it work better for sparse and noisy data sets [64]. Extensions of the ISOMAP algorithm include kernel ISOMAP, whose main idea is to make the $B = HAH$ matrix a Mercer kernel matrix (in some cases this computation leads to a non positive semidefinite matrix) [17], L-ISOMAP or Landmark ISOMAP, a faster variant of the algorithm [27] and C-ISOMAP, which involves amplifying the high density regions and attenuating the low density regions of data points in the manifold [27].

5.4. *The relation between ISOMAP and Kernel PCA*

Recall that in Kernel PCA, after computing the kernel matrix K, we have to remove the mean from the data by:

$$\tilde{K} = K - 1_n K - K 1_n + 1_n K 1_n \tag{66}$$

where 1_n is a $n \times n$ matrix with all elements equal to $1/n$. Let $\vec{e}$ be defined as:

$$\vec{e} = \frac{1}{\sqrt{n}} [1, 1, ..., 1]^T \tag{67}$$

It is straightforward to see that $1_n = \vec{e}\vec{e}^T$. Then, we have:

$$\begin{aligned} \tilde{K} &= K - \vec{e}\vec{e}^T K - K\vec{e}\vec{e}^T + \vec{e}\vec{e}^T K \vec{e}\vec{e}^T \\ &= \left[\left(I - \vec{e}\vec{e}^T\right) K\right] - \left[\left(I - \vec{e}\vec{e}^T\right) K\right] \vec{e}\vec{e}^T \\ &= \left(I - \vec{e}\vec{e}^T\right) K \left(I - \vec{e}\vec{e}^T\right) \end{aligned} \tag{68}$$

In ISOMAP, the first step consists in obtaining the dot product matrix B from the geodesic distance matrix D, which is equivalent to:

$$B = -\frac{1}{2} HDH \tag{69}$$

where $H = \left(I - \vec{e}\vec{e}^T\right)$, leading to [35]:

$$K_{iso} = -\frac{1}{2} \left(I - \vec{e}\vec{e}^T\right) D \left(I - \vec{e}\vec{e}^T\right) \tag{70}$$

Therefore, Kernel PCA becomes ISOMAP when the kernel matrix K is minus one-half of the geodesic distance matrix.

6. Locally Linear Embedding

The ISOMAP algorithm is a global method in the sense that to find the coordinates of a given input vector $\vec{x}_i \in R^m$ in the manifold, it uses information from all the samples through the matrix B. On the ohter hand, Locally Linear Embedding (LLE), as the name emphasizes, is a local method, that is, the new coordinates of any $\vec{x}_i \in R^m$ depends only on the neighborhood of that point. The main hypothesis behind LLE is that for a sufficiently high density of samples, it is expected that a vector $\vec{x}_i$ and its neighbors define a linear patch, that is, they all belong to an Euclidean subspace [61]. In this way, it is possible to characterize the local geometry by linear coefficients:

$$\hat{\vec{x}}_i \approx \sum_j w_{ij}\vec{x}_j \qquad \text{for} \qquad \vec{x}_j \in N(\vec{x}_i) \tag{71}$$

that is, we can reconstruct a vector as a linear combination of its neighbors.

Basically, the LLE algorithm require as inputs an $n \times m$ data matrix X, with rows $\vec{x}_i$, a desired number of dimensions $d < m$ and an integer $k > d+1$ for finding local neighborhoods. The output is a $n \times d$ matrix Y, wtih rows $\vec{y}_i$. The LLE algorithm can be divided in three main steps [61,62]:

(1) From each $\vec{x}_i \in R^m$ find its k nearest neighbors;
(2) Find the weight matrix W which minimizes the reconstruction error for each data point $\vec{x}_i \in R^m$;

$$E(W) = \sum_{i=1}^{n} \left\| \vec{x}_i - \sum_j w_{ij}\vec{x}_j \right\|^2 \tag{72}$$

where $w_{ij} = 0$ unless $\vec{x}_j$ is one of $\vec{x}_i$'s k-nearest neighbors and for each i, $\sum_j w_{ij} = 1$.
(3) Find the coordinates Y which minimize the reconstruction error using the optimum weights;

$$\Phi(Y) = \sum_{i=1}^{n} \left\| \vec{y}_i - \sum_j w_{ij}\vec{y}_j \right\|^2 \tag{73}$$

subject to the constraints that $\sum_i Y_{ij} = 0$ for each j, and that $Y^T Y = I$.

In the following, we describe how to obtain the solution to each step of LLE.

6.1. *Finding Local Linear Neighborhoods*

In the basic version of the LLE algorithm, the number of neighbors is fixed for every sample and the metric adopted to rank the nearest ones is the simple Euclidean distance. However, different criteria can be considered in choosing the nearest neighbors, such as selecting all the samples that are inside a ball of fixed radius. The number of neighbors may also change from neighborhood to neighborhood. Some alternative rules are: 1) choose all samples within a certain radius R_i, up to a maximum number N_i; 2) choose a certain number of neighbors N_i, but none outside a maximum radius R_i [62].

A relevant aspect of LLE is that the algorithm is capable of recovering embeddings whose intrinsic dimensionality d is smaller that the number of neighbors, k. Moreover, the assumption of a linear patch enforces us to have an upper bound on k. For instance, in highly curved datasets, it is not reasonable to have a large k, or we could violate this assumption. In the uncommon situation where $k > m$, it has been shown that each sample can be perfectly reconstructed from its neighbors and another problem arises: the reconstruction weights are not unique anymore. To overcome this limitation, some regularization is required in order to break the degeneracy [62].

Finally, another concern in the LLE algorithm is related to the connectivity of the KNN graph. If the graph has multiple connected components, then LLE should be applied separately on each one of them, or the neighborhood selection process should be modified to assure global connectivity [62].

6.2. *Least-Squares Estimation of the Weights*

The second step of LLE is to reconstruct each data point from its nearest neighbors. The optimal reconstruction weights can be computed in closed form. Without loss of generality, we can express the local reconstruction error at point $\vec{x}_i$ as:

$$E(\vec{w}) = \left\| \sum_j w_j (\vec{x}_i - \vec{x}_j) \right\|^2 = \sum_j \sum_k w_j w_k (\vec{x}_i - \vec{x}_j)^T (\vec{x}_i - \vec{x}_k) \tag{74}$$

Defining the matrix C as:

$$C_{jk} = (\vec{x}_i - \vec{x}_j)^T (\vec{x}_i - \vec{x}_k) \tag{75}$$

we have the following expression for the local reconstruction error:

$$E(\vec{w}) = \sum_j \sum_k w_j C_{jk} w_k = \vec{w}^T C \vec{w} \tag{76}$$

About the constraint $\sum_j w_j = 1$, it can be understood in two different ways: geometrically and probabilistically. From a geometric point of view, it provides invariance under translation, that is, by adding any constant vector $\vec{c}$ to $\vec{x}_i$ and all of its neighbors, the reconstruction error is not changed. Let $\tilde{\vec{x}}_i = \vec{x}_i + \vec{c}$ and $\tilde{\vec{x}}_j = \vec{x}_j + \vec{c}$. Then, the new local reconstruction error is given by:

$$\begin{aligned} \tilde{E}(\vec{w}) &= \left\| \tilde{\vec{x}}_i - \sum_j w_j \tilde{\vec{x}}_j \right\|^2 = \left\| \vec{x}_i + \vec{c} - \sum_j w_j (\vec{x}_j + \vec{c}) \right\|^2 \\ &= \left\| \vec{x}_i + \vec{c} - \sum_j w_j \vec{x}_j - \sum_j w_j \vec{c} \right\|^2 = \left\| \vec{x}_i + \vec{c} - \sum_j w_j \vec{x}_j - \vec{c} \right\|^2 = E(\vec{w}) \end{aligned} \tag{77}$$

In terms of probability, enforcing the weights to sum to one makes W a stochastic transition matrix [62], directly related to Markov Chains and diffusion maps.

We will show that in the minimization of the squared error, the solution is found through an eigenvalue problem. Actually, the estimation of the matrix W reduces to n eigenvalue problems: as there are no constraints across the rows of W, we can find the optimal weights for each sample $\vec{x}_i$ separately, which drastically simplifies the computations.

Thus, we have n independent constrained optimization problems given by:

$$\arg\min_{\vec{w}_i} \ \vec{w}_i^T C_i \vec{w}_i \qquad \text{subject to} \quad \vec{1}^T \vec{w}_i = 1 \quad \text{for} \quad i = 1,2,...,n \tag{78}$$

Using Lagrange multipliers, we write the Lagrangian function as:

$$L(\vec{w}_i, \lambda) = \vec{w}_i^T C_i \vec{w}_i - \lambda(\vec{1}^T \vec{w}_i - 1) \tag{79}$$

Taking the derivatives with relation to $\vec{w}_i$:

$$\frac{\partial}{\partial \vec{w}_i} L(\vec{w}_i, \lambda) = 2C_i \vec{w}_i - \lambda \vec{1} = 0 \tag{80}$$

which leads to

$$C_i \vec{w}_i = \frac{\lambda}{2} \vec{1} \tag{81}$$

If the matrix C_i is invertible, we have a closed form solution:

$$\vec{w}_i = \frac{\lambda}{2} C_i^{-1} \vec{1} \tag{82}$$

where λ can be adjusted to ensure $\sum_j w_i(j) = 1$. Actually, there is a closed form expression for $w_i(j)$ [63]:

$$w_i(j) = \frac{\sum_k C_i^{-1}(j,k)}{\sum_k \sum_l C_i^{-1}(k,l)} \tag{83}$$

In order to speed up the algorithm, instead of computing the inverse of the matrix C, it is usual to solve the linear system:

$$C_i \vec{w}_i = \vec{1} \tag{84}$$

and then normalize the solution to guarantee that $\sum_j w_i(j) = 1$ by dividing each coefficient of the vector $\vec{w}_i$ by the sum of all the coefficients:

$$w_i(j) = \frac{w_i(j)}{\sum_j w_i(j)} \qquad \text{for} \qquad j = 1,2,...,m \tag{85}$$

If k, the number of neighbors, is greater than m, the number of features, then (in general) the space spanned by k distinct vectors is the whole space. It means that $\vec{x}_i$ can be written exactly as a linear combination of its k-nearest neighbors. In fact, if $k > m$, there are generally infinitely many solutions to $\vec{x}_i = \sum_j w_j \vec{x}_j$, because there are more unknowns (k) than equations (m). In this case, the optimization problem is ill-posed, and regularization is required. A common regularization technique is Tikonov regularization, which instead of directly minimizing:

$$\left\| \vec{x}_i - \sum_j w_j \vec{x}_j \right\|^2 \tag{86}$$

adds a penalization term to the least squares problem:

$$\left\| \vec{x}_i - \sum_j w_j \vec{x}_j \right\|^2 + \alpha \sum_j w_j^2 \tag{87}$$

where α controls the degree of regularization. In other words, select the weights which minimize a combination of reconstruction error and the sum of the squared weights. As $\alpha \to 0$, we have the least-squares problem. In the opposite limit, $\alpha \to \infty$, the squared-error term becomes negligible, and we want to minimize the Euclidean norm of the weight vector $\vec{w}$. Tipically, α is set to be a small but non-zero value. In this case, the n independent constrained optimization problems are:

$$\arg\min_{\vec{w}_i} \ \vec{w}_i^T C_i \vec{w}_i + \alpha \vec{w}_i^T \vec{w}_i \qquad \text{subject to} \quad \vec{1}^T \vec{w}_i = 1 \quad \text{for} \quad i = 1,2,...,n \tag{88}$$

The Lagrangian function is defined by:

$$L(\vec{w}_i, \lambda) = \vec{w}_i^T C_i \vec{w}_i + \alpha \vec{w}_i^T \vec{w}_i - \lambda (\vec{1}^T \vec{w}_i - 1) \tag{89}$$

Taking the derivative with respect to $\vec{w}_i$ and setting the result to zero:

$$2C_i \vec{w}_i + 2\alpha \vec{w}_i = \lambda \vec{1} \tag{90}$$

$$(C_i + \alpha I)\vec{w}_i = \frac{\lambda}{2} \vec{1} \tag{91}$$

$$\vec{w}_i = \frac{\lambda}{2} (C_i + \alpha I)^{-1} \vec{1} \tag{92}$$

where λ is chosen to properly normalize $\vec{w}_i$. In other words, to regularize the problem, we should add a small perturbation in the main diagonal of the matrix C_i.

6.3. *Finding the coordinates*

If the local neighborhoods are small enough compared to the curvature of the manifold, the optimal reconstruction weights in the embedding space and the weights reconstruction on the manifold are approximatelly the same (the two sets of weights are exactly equal for linear subspaces, and for general manifolds they can be brought arbitrarily close to each other by shrinking the neighborhood sufficiently) [69].

The key idea behind the third step of the LLE algorithm is to use the optimal reconstruction weights estimated by least-squares as the proper weights on the manifold and solve for the local manifold coordinates. Thus, fixing the weight matrix W, the goal is to solve another quadratic minimization problem to minimize:

$$\Phi(Y) = \sum_{i=1}^{n} \left\| \vec{y}_i - \sum_j w_{ij} \vec{y}_j \right\|^2 \tag{93}$$

In other words, we have to answer the question: what are the coordinates $\vec{y}_i \in R^d$ (approximately on the manifold), that these weights (W) reconstruct them?

In order to avoid degeneracy, we have to impose two constraints:

(1) The mean of the data in the transformed space is zero, otherwise we would have an infinite number of solutions;
(2) The covariance matrix of the trasnformed data is the identity matrix, that is, there is not correlation between the components of $\vec{y} \in R^d$;

However, unlikely the estimation of the weights W, finding the coordinates does not simplify into n independent problems, because each row of Y appears in Φ multiple times, once as the central vector y_i and again as one of the neighbors of other vectors.

First, we will rewrite equation (93) in a more meaningful way using matrices. Note that:

$$\Phi(Y) = \sum_{i=1}^{n} \left[\left(\vec{y}_i - \sum_j w_{ij} \vec{y}_j \right)^T \left(\vec{y}_i - \sum_j w_{ij} \vec{y}_j \right) \right] \tag{94}$$

Applying the distriutive law, we have:

$$\Phi(Y) = \sum_{i=1}^{n} \left[\vec{y}_i^T \vec{y}_i - \vec{y}_i^T \left(\sum_j w_{ij} \vec{y}_j \right) - \left(\sum_j w_{ij} \vec{y}_j \right)^T \vec{y}_i + \left(\sum_j w_{ij} \vec{y}_j \right)^T \left(\sum_j w_{ij} \vec{y}_j \right) \right] \tag{95}$$

Expanding the summation leads to:

$$\Phi(Y) = \sum_{i=1}^{n} \vec{y}_i^T \vec{y}_i - \sum_{i=1}^{n} \sum_{j} \vec{y}_i^T w_{ij} \vec{y}_j - \sum_{i=1}^{n} \sum_{j} \vec{y}_j^T w_{ji} \vec{y}_i + \sum_{i=1}^{n} \sum_{j} \sum_{k} \vec{y}_j^T w_{ji} w_{ik} \vec{y}_k \tag{96}$$

Denoting by Y the $d \times n$ matrix in which each column $\vec{y}_i$ for $i = 1, 2, ..., n$ stores the coordinates of of the i-th point in the manifold and knowing that $\vec{w}_i(j) = 0$ unless $\vec{y}_j$ is one of the neighbors of $\vec{y}_i$, we can write $\Phi(Y)$ as:

$$\begin{aligned}
\Phi(Y) &= Tr(Y^T Y) - Tr(Y^T W Y) - Tr(Y^T W^T Y) + Tr(Y^T W^T W Y) \\
&= Tr(Y^T Y) - Tr(Y^T (WY)) - Tr((WY)^T Y) + Tr((WY)^T (WY)) \\
&= Tr(Y^T (Y - WY) - (WY)^T (Y - WY)) \\
&= Tr((Y - WY)^T (Y - WY)) \\
&= Tr(((I - W)Y)^T ((I - W)Y)) \\
&= Tr(Y^T (I - W)^T (I - W) Y)
\end{aligned} \tag{97}$$

Defining the $n \times n$ matrix M as:

$$M = (I - W)^T (I - W) \tag{98}$$

we get the following optimization problem:

$$\arg\min_{Y} \; Tr(Y^T M Y) \qquad \text{subject to} \quad \frac{1}{n} Y^T Y = I \tag{99}$$

Thus, the Lagrangian function is given by:

$$L(Y, \lambda) = Tr(Y^T M Y) - \lambda \left(\frac{1}{n} Y^T Y - I \right) \tag{100}$$

Differentiating the function and setting the result to zero gives:

$$2MY - 2\frac{\lambda}{n} Y = 0 \tag{101}$$

$$MY = \beta Y \tag{102}$$

where $\beta = \frac{\lambda}{n}$, showing that the Y must be composed by the eigenvectors of the matrix M. Since we have a minimization problem, we want to select to compose Y the d eigenvectors associated to the d smallest eigenvalues. Note that M being a $n \times n$ matrix, it has n eigenvalues and n orthogonal eigenvectors. Although the eigenvalues are real and non-negative, the smallest of them is always zero, with the constant eigenvector $\vec{1}$. This bottom eigenvector corresponds to the mean of Y and should be discarded to enforce the constraint that $\sum_{i=1}^{n} \vec{y}_i = 0$ [26]. Note that each row of W must sum one, and then:

$$W\vec{1} = \vec{1} \tag{103}$$

$$\vec{1} - W\vec{1} = 0 \tag{104}$$

$$(I-W)\vec{1} = 0 \tag{105}$$

$$(I-W)^T(I-W)\vec{1} = 0 \tag{106}$$

$$M\vec{1} = 0 \tag{107}$$

Therefore, to get $\vec{y}_i \in R^d$, where $d < m$, we must select the $d+1$ smallest eigenvectors and discard the constant eigenvector with zero eigenvalue. In other words, we must select the d eigenvectors associated to the bottom non-zero eigenvalues.

The LLE algorithm has three important parameters: the intrinsic dimensionality d, the number of nearest neighbors, k, and in some cases the regularization parameter α. Dimensionality reduction with LLE is quite sensitive to variations in these parameters. If d is set too high, the mapping will enhance noise; if it is set too low, distinct parts of the data set might be mapped on top of each other. If k is set too small, the mapping will not reflect any global properties; if it is too high, the mapping will lose its nonlinear character and behave like traditional PCA, as the entire data set is seen as local neighbourhood. Finally, if α is set incorrectly, the eigenanalysis may not converge [26]. Algorithm 4 shows a summary of the LLE method. The input is a $m \times n$ data matrix X whose columns are the samples and the output is a $n \times d$ matrix Y whose rows represent the coordinates of the points in the learned manifold.

Algorithm 4 Locally Linear Embedding

function LLE(X, K, d)
  From the input data $X_{m\times n}$ build a KNN graph.
  for $\vec{x}_i \in X^T$ **do**
    Compute the $K \times K$ matrix C_i as:

$$C_i(j,k) = (\vec{x}_i - \vec{x}_j)^T(\vec{x}_i - \vec{x}_k) \tag{108}$$

    Solve the linear system $C_i\vec{w}_i = \vec{1}$ to estimate the weights $\vec{w}_i \in R^K$.
    Normalize the weights in $\vec{w}_i$ so that $\sum_j \vec{w}_i(j) = 1$.
  end for
  Construct the $n \times n$ matrix W, whose lines are the estimated $\vec{w}_i$.
  Compute $M = (I-W)^T(I-W)$.
  Find the eigenvalues and eigenvectors of the matrix M.
  Select the bottom d non-zero eigenvectors of M and define the matrix Y, where each column is an eigenvector.
  return Y
end function

7. Laplacian Eigenmaps

The basic idea behind the Laplacian Eigenmaps method is that if we approximate a manifold by a connected and undirected basic graph, then it is possible to find a map from the vertices of the graph to an Euclidean subspace R^d, such that locality is preserved, or in other words, the map is smooth in the sense that neighboring points in the graph will remain close together after the mapping is performed. Such map is given by the eigenvectors of the graph Laplacian matrix [11]. The representation map generated by the algorithm may be viewed as a discrete approximation to a continuous map that naturally arises from the geometry of the manifold: the Laplace-Beltrami operator [5]. It has been shown the convergence of the eigenvectors of the graph Laplacian associated to a point cloud dataset to eigenfunctions of the Laplace-Beltrami operator when the data is sampled from a uniform probability distribution on an embedded manifold [6]. In machine learning, the Laplace Eigenmaps method is closely related to spectral clustering, an unsupervised classification approach for data clustering [83].

7.1. *Overview*

Basically, the Laplacian Eigenmaps algorithm require as inputs an $n \times m$ data matrix X, with each row $\vec{x}_i$ defining a data point, a desired number of dimensions $d < m$ and an integer k for finding local neighborhoods. The output is a $n \times d$ matrix Y, wtih rows $\vec{y}_i$. The algorithm can be divided in three main steps [5]:

(1) Construct the neighborhood graph $G = (V,E)$ by linking nodes v_i and v_j if $\vec{x}_i$ and $\vec{x}_j$ are close. The two variants are:
   - ε-neighborhood: connect v_i and v_j by an edge if $\left\|\vec{x}_i - \vec{x}_j\right\|^2 \leq \varepsilon$.
   - k-nearest neighbors: connect v_i and v_j by an edge if v_i is among the k-nearest neighbors of v_j or v_j is among the k-nearest neighbors of v_i.

(2) Choose the weights to define the adjacency matrix W. There are also two variations:
   - Heat kernel (with parameter $t \in R$): if nodes v_i abd v_j are connected, make
   $$W_{ij} = exp\left\{-\frac{\left\|\vec{x}_i - \vec{x}_j\right\|^2}{t}\right\} \tag{109}$$
   otherwise make $W_{ij} = 0$. The justification for this choice is given by the heat equation.
   - Binary weights: make $W_{ij} = 1$ if nodes v_i and v_j are connected by an edge and $W_{ij} = 0$ if v_i and v_j are not connected by an edge. There is no need to choose t.

(3) Embedding: find the coordinates Y by choosing the d eigenvectors associated to the d smallest non-zero eigenvalues of the graph Laplacian L.

In the following, we present the reasons why the Laplacian Eigenmaps algorithm is able to provide optimal embeddings in terms of locality and neighborhood preservation.

7.2. *Graph Laplacian and its Properties*

Let $G = (V,E)$ be an undirected graph with vertex set $V = \{v_1, v_2, ..., v_n\}$. We consider that the graph is weighted, that is, each edge between v_i and v_j has a non-negative weight $w_{ij} \geq 0$. Tipically, the weights w_{ij} represent a similarity measure or a pairwise distance between vectors $\vec{x}_i \in R^m$ and $\vec{x}_j \in R^m$.

Definition 6. The weighted adjacency matrix of an undirected graph $G = (V,E)$ with $|V| = n$ is the symmetric matrix $W = \{w_{ij}\}$ for $i, j = 1, 2, ..., n$. If $w_{ij} = 0$ the vertices v_i and v_j are not connected by an edge.

Definition 7. The degree of a vertex $v_i \in V$ is defined by the sum of the elements of the i-th row of W:

$$d_i = \sum_{j=1}^{n} w_{ij} \tag{110}$$

The degree matrix D is defined as the diagonal matrix with degrees $d_1, d_2, ..., d_n$.

Laplacian matrices and their properties have been deeply investigated in spectral graph theory, a very mature research field whose objective is the study of graphs in regards to the characteristic polynomial, eigenvalues, and eigenvectors of all kinds of matrices associated with a graph [19,14,72,58,80].

Definition 8. The unnormalized graph Laplacian matrix is defined by:

$$L = D - W \tag{111}$$

where D is the degree matrix and W is the adjacency matrix.

In the following, we present some basic but very important mathematical properties of the graph Laplacian [83]. More details about the Laplacian spectrum and advanced properties can be found in Mohar's paper [56].

Theorem 3. *The Laplacian matrix L satisfies the following properties:*

(1) For every column vector $\vec{f} \in R^n$ we have:

$$\vec{f}^T L \vec{f} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (f_i - f_j)^2 \tag{112}$$

(2) L is symmetric and positive semi-definite.
(3) The smallest eigenvalue of L is zero and the corresponding eigenvector is the constant $\vec{1}$ vector
(4) L has n non-negative, real eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_m$.

To prove the first statement, note that by the definition of L and D we have:

$$\begin{aligned}
\vec{f}^T L\vec{f} &= \vec{f}^T D\vec{f} - \vec{f}^T W\vec{f} \\
&= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} f_i w_{ij} f_j \\
&= \frac{1}{2}\left(2\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} f_i w_{ij} f_j\right) \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} f_i f_j + \sum_{i=1}^{n} d_j f_j^2\right) \\
&= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} f_i^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} f_i f_j + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} f_j^2\right) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(f_i - f_j)^2
\end{aligned} \tag{113}$$

The second statement is divided in two parts: the first, about the symmetry, follows directly from the symmetry of the matrices D and W; the second part, regarding the positive semidefiniteness, it is clear that $(f_i - f_j)^2 \geq 0, \forall f_i, f_j \in R$, and because $w_{ij} \geq 0$ for $i, j = 1, 2, ..., n$, $\vec{f}^T L\vec{f} \geq 0$.

To prove the third statement, note that:

$$L\vec{1} = (D - W)\vec{1} = D\vec{1} - W\vec{1} = \sum_{i=1}^{n} d_i - \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} = \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} d_i = 0 \tag{114}$$

showing that the constant eigenvector $\vec{1}$ has zero eigenvalue.

Finally, the fourth statement is a direct consequence of statements (2) and (3).

7.3. *Laplacian Embedding on the Line*

In this section, we will see that the embedding provided by Laplacian Eigenmaps is optimal in terms of preserving local information, that is, neighboring points in the graph are close and distant points in the graph are far apart after the embedding. Suppose we have a connected weighted graph $G = (V, E)$ whose nodes are the data points in $X = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n]$. The problem in question can be formulated as: how to map the nodes of G to a line so that connected points stay as close together as possible? The goal of Laplacian Eigenmaps is to answer this motivating question.

Let $\vec{y} = [y_1, y_2, ..., y_n]^T \in R^n$ be a map of the vertices $v_1, v_2, ..., v_n$ to the real line. A good objective function should heavily penalize neighboring points that are mapped far apart. A suitable choice for a given adjacency matrix W is the following function:

$$J(\vec{y}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(y_i - y_j)^2 = \vec{y}^T L\vec{y} \tag{115}$$

where L is the Laplacian matrix. Note that $J(\vec{y})$ is a measure of how scattered the points are distributed in the real line, so minimizing it is an attempt to guarantee that if $\vec{x}_i$ and $\vec{x}_j$ are close in the input space, then the coordinates y_i and y_j are also close in the line. Thus, we can formulate the constrained optimization problem:

$$\arg\min_{\vec{y}} \; y^T L y \qquad \text{subject to} \quad y^T D y = 1 \tag{116}$$

where the constraint $y^T D y = 1$ removes an arbitrary scaling factor in the embedding [5]. In other words, we are interested in the direction of the vector $\vec{y}$. If there were no constraint, one could further minimize the objective function by simply dividing the components of $\vec{y}$ by a constant. Once again, writing the Lagrangian function, we have:

$$L(\vec{y}, \lambda) = y^T L y - \lambda (y^T D y - 1) \tag{117}$$

Differentiating with respect to $\vec{y}$ and setting the result to zero gives:

$$\frac{\partial}{\partial \vec{y}} L(\vec{y}, \lambda) = 2L\vec{y} - 2\lambda D\vec{y} = 0 \tag{118}$$

which leads to:

$$L\vec{y} = \lambda D\vec{y} \tag{119}$$

$$(D^{-1}L)\vec{y} = \lambda \vec{y} \tag{120}$$

showing that we have a generalized eigenvector problem. Since it is a minimization problem, we have to choose the eigenvector of $D^{-1}L$ associated to the smallest eigenvalue. As the constant eigenvector $\vec{1}$ has zero eigenvalue, we must discard it. It makes sense that in order to minimize the scatter of the points all of them are mapped to the same coordinate. However, this trivial solution has no practical use. Therefore, $\vec{y}$ should be the eigenvector associated to the smallest non-zero eigenvalue, also known as Fiedler vector [31,19].

In fact, there are some variations based on the calculation of the graph Laplacian. The matrix $D^{-1}L$ is one of the normalized Laplacians. Another way to compute the normalized Laplacian is by $D^{-1/2}LD^{-1/2}$. It is also possible to use the unnormalized Laplacian L directly, but often the normalized versions are preferred due to interesting mathematical properties. It has been shown that in spectral clustering, normalized Laplacians have important connections with graph-cut minimization problems [83].

7.4. *Laplacian Embedding on* R^d

Consider the generalized problem of embedding the graph $G = (V, E)$ into a d-dimensional Euclidean space. Now each node $v_i \in V$ has to be mapped to a point in R^d, that is, we need to estimate d coordinates for each node. We denote the final embedding by a $n \times d$ matrix

$Y = [\vec{y}_1, \vec{y}_2, ..., \vec{y}_d]$, where the i-th row, $\vec{y}^{(i)}$, provides the coordinates of v_i in the manifold. The objective function is generalized to:

$$J(Y) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left\| \vec{y}^{(i)} - \vec{y}^{(j)} \right\|^2 \tag{121}$$

where $\vec{y}^{(i)} = [\vec{y}_1(i), \vec{y}_2(i), ..., \vec{y}_d(i)]$ is the d-dimensional representation of v_i. Note that, considering Y as a $n \times d$ matrix in which each row represents a $\vec{y}^{(i)}$, for $i = 1, 2, ..., n$ we rewrite the objective function as:

$$J(Y) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} (\vec{y}^{(i)} - \vec{y}^{(j)})(\vec{y}^{(i)} - \vec{y}^{(j)})^T \tag{122}$$

Expanding the expression for $J(Y)$, we can simplify it to:

$$\begin{aligned} J(Y) &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left[W_{ij} \vec{y}^{(i)} \vec{y}^{(i)^T} - W_{ij} \vec{y}^{(i)} \vec{y}^{(j)^T} - W_{ij} \vec{y}^{(j)} \vec{y}^{(i)^T} + W_{ij} \vec{y}^{(j)} \vec{y}^{(j)^T} \right] \\ &= \frac{1}{2} \left[\sum_{i=1}^{n} d_i \vec{y}^{(i)} \vec{y}^{(i)^T} - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \vec{y}^{(i)} \vec{y}^{(j)^T} + \sum_{j=1}^{n} d_j \vec{y}^{(j)} \vec{y}^{(j)^T} \right] \\ &= \frac{1}{2} \left[2 \sum_{i=1}^{n} d_i \vec{y}^{(i)} \vec{y}^{(i)^T} - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \vec{y}^{(i)} \vec{y}^{(j)^T} \right] \\ &= \sum_{i=1}^{n} d_i \vec{y}^{(i)} \vec{y}^{(i)^T} - \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \vec{y}^{(i)} \vec{y}^{(j)^T} \end{aligned} \tag{123}$$

Considering $Y_{n \times d}$ the matrix of the coordinates for the n points, $D_{n \times n}$ the diagonal matrix of the degrees d_i and $W_{n \times n}$ the adjacency matrix, we can rewrite the equation using a matrix-vector notation as:

$$J(Y) = Tr(DYY^T) - Tr(WYY^T) \tag{124}$$

As the trace is an operator that is invariant under cyclic permutations, we have:

$$\begin{aligned} J(Y) &= Tr(Y^T DY) - Tr(Y^T WY) = Tr(Y^T (DY - WY)) \\ &= Tr(Y^T (D - W)Y) = Tr(Y^T LY) \end{aligned} \tag{125}$$

Thus, we have the following constrained optimization problem:

$$\arg\min_{Y} \; Tr(Y^T LY) \qquad \text{subject to} \quad Y^T DY = I \tag{126}$$

whose Lagrangian function is given by:

$$L(Y,\lambda) = Tr(Y^T LY) - \lambda(Y^T DY - I) \tag{127}$$

Taking the derivative and setting the result to zero leads to:

$$\frac{\partial}{\partial Y} L(Y,\lambda) = 2LY - 2\lambda DY = 0 \tag{128}$$

leading to the following eigenvector problem:

$$LY = \lambda DY \tag{129}$$

This result shows that we should select to compose the columns of the matrix Y the d eigenvectors associated to the d smallest non-zero eigenvalues of the normalized Laplacian $D^{-1}L$. Some variants of the algorithm include the eigendecomposition of different versions of the graph Laplacian. The most common choices are another form of normalized Laplacian, given by $L_{sym} = D^{-1/2}LD^{-1/2}$, and the pure unnormalized Laplace $L = D - W$. When applying Laplacian Eigenmaps to some real-world data, several limitations have been exposed such as uneven data sampling, out-of-sample problem, small sample size, discriminant fea ture extraction and selection, etc. In order to overcome these problems, a large number of extensions to Laplacian Eigenmaps have been made [11].

Algorithm 5 shows a summary of the Laplacian Eigenmaps method. The input is a $m \times n$ data matrix X whose columns are the samples and the output is a $n \times d$ matrix Y whose rows represent the coordinates of the points in the learned manifold.

Algorithm 5 Laplacian Eigenmaps

1: **function** LAPLACEEIGEN(X, K, d)
2: From the input data $X_{m\times n}$ build a KNN graph.
3: Choose the weights to define the adjacency matrix W.

$$W_{ij} = exp\left\{-\frac{\left\|\vec{x}_i - \vec{x}_j\right\|^2}{t}\right\} \qquad \text{if} \qquad v_j \in N(v_i) \tag{130}$$

4: Compute the diagonal matrix D with the degrees d_i for $i = 1, 2, ..., n$.

$$d_i = \sum_{j=1}^{n} W_{ij} \tag{131}$$

5: Compute the Laplacian matrix $L = D - W$
6: Select the bottom d eigenvectors with non-zero eigenvalues of $D^{-1}L$ and define the matrix Y, where each column is an eigenvector.
7: **return** Y
8: **end function**

7.5. *The Laplace-Beltrami Operator*

The motivation for the Laplacian Eigenmaps algorithm is the role of the Laplace-Beltrami operator on manifolds. This section explain the reasons why the eigenfunctions of this operator have desirable properties for finding an embedding to the real line, based on the seminal paper of Belkin and Niyogi [5]. Consider a smooth d-dimensional manifold M embedded in R^m. Suppose we have a map $f : M \to R$ that assigns a real value $f(\vec{x})$ to each point $\vec{x}$ in the manifold. The gradient $\nabla f(\vec{x})$ is a vector field that assigns a vector for each point of M, such that for a small displacement $\delta\vec{x}$ we have [4]:

$$|f(\vec{x}+\delta\vec{x}) - f(\vec{x})| \approx |\langle \nabla f(\vec{x}), \delta\vec{x}\rangle| \tag{132}$$

The intuition behind this approximation is that if the displacement vector $\delta\vec{x}$ is orthogonal to the gradient and the dot product is zero we are at a level curve of the function, so $|f(\vec{x}+\delta\vec{x}) - f(\vec{x})| = 0$. Thus, the variation in f is zero. Applying the Cauchy-Schwarz inequality, we have:

$$|\langle \nabla f(\vec{x}), \delta\vec{x}\rangle| \leq \|\nabla f(\vec{x})\| \, \|\delta\vec{x}\| \tag{133}$$

leading to:

$$|f(\vec{x}+\delta\vec{x}) - f(\vec{x})| \leq \|\nabla f(\vec{x})\| \, \|\delta\vec{x}\| \tag{134}$$

This result shows that if the norm of the gradient vector is small, then points in the vicinity of an arbitrary $\vec{x} \in M$ will be mapped close to $f(\vec{x})$. In other words, to assure that the map f is smooth, that is, that the mapping is, in average, the optimal one in terms of preserving localities we seek to solve a minimization problem:

$$\arg\min_f \int_M \|\nabla f(\vec{x})\|^2 \qquad \text{with} \qquad \|f\| = 1 \tag{135}$$

However, we can rewrite the minimization problem in terms of the Laplacian operator. First, note that the Laplacian is a differential operator that can be computed by the divergent of the gradient:

$$L(f) = \text{div}(\nabla f) = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} \tag{136}$$

where the div operator transforms a vector field into a scalar field.

Second, it has been shown that the divergent and gradient operators are adjoint, that is, for a function $f : M \to R$ and a vector field X [4]:

$$\int_M \langle X, \nabla f\rangle = \int_M \text{div}(X) f \tag{137}$$

Thus, the objective function in the minimization problem can be expressed by:

$$\int_M \|\nabla f(\vec{x})\|^2 = \int_M \langle \nabla f, \nabla f \rangle = \int_M \langle \text{div}(\nabla f), f \rangle = \int_M L(f) f \tag{138}$$

where $L(f)$ is the Laplacian operator. The solution for this minimization problem is known to be the eigenfunctions of the Laplacian operator. This result establishes the connection between the eigenvectors of the Laplacian matrix and the construction of smooth and locality preserving mappings for manifold learning.

7.6. *The Weight Matrix and the Heat Flow*

An important question that remains open in the Laplacian Eigenmaps method is: why the weights between neighboring nodes in the graph are computed using a Gaussian kernel? According to the authors of the algorithm, there is a connection between the Laplace-Beltrami operator and the heat flow [5].

Let $f : M \rightarrow R$ be the initial heat distribution and $u(\vec{x},t)$ be the heat distribution at time t. Then, it is clear that $u(\vec{x},0) = f(\vec{x})$. The heat equation is a parabolic partial differential equation given by:

$$\frac{\partial u}{\partial t} + Lu = 0 \tag{139}$$

where L is the Laplacian operator. It can be shown that the solution of this differential equation is the function:

$$u(\vec{x},t) = \int_M H_t(\vec{x},\vec{y}) f(\vec{y}) \tag{140}$$

where $H_t(\vec{x},\vec{y})$ is the heat kernel. Combining equations (139) and (140), we can write:

$$Lf(\vec{x}) = Lu(\vec{x},0) = -\frac{\partial u}{\partial t} = -\left(\frac{\partial}{\partial t} \left[\int_M H_t(\vec{x},\vec{y}) f(\vec{y}) \right] \right)_{t=0} \tag{141}$$

It has been shown that H_t is approximately Gaussian:

$$H_t(\vec{x},\vec{y}) = (4\pi t)^{-d/2} \exp\left\{ -\frac{\|\vec{x}-\vec{y}\|^2}{4t} \right\} (\phi(\vec{x},\vec{y}) + O(t)) \tag{142}$$

where $\phi(\vec{x},\vec{y})$ is a smooth function with $\phi(\vec{x},\vec{x}) = 1$. When $\vec{x}$ and $\vec{y}$ are close and t is small, equation (142) is simplified to:

$$H_t(\vec{x},\vec{y}) \approx (4\pi t)^{-d/2} \exp\left\{ -\frac{\|\vec{x}-\vec{y}\|^2}{4t} \right\} \tag{143}$$

From equation (141), applying the definition of derivative:

$$Lf(\vec{x}) = -\lim_{\Delta t \to 0} \left[\frac{\int_M H_{t+\Delta t}(\vec{x},\vec{y}) f(\vec{y}) - \int_M H_t(\vec{x},\vec{y}) f(\vec{y})}{\Delta t} \right] \tag{144}$$

For small values of t, we have $t \approx \Delta t$:

$$\begin{aligned} Lf(\vec{x}) &= -\lim_{t \to 0} \left[\frac{\int_M H_{t+\Delta t}(\vec{x},\vec{y}) f(\vec{x}) - \int_M H_t(\vec{x},\vec{y}) f(\vec{y})}{t} \right] \\ &\approx \frac{1}{t} \left[\lim_{t \to 0} \int_M H_t(\vec{x},\vec{y}) f(\vec{y}) - \int_M H_{t+\Delta t}(\vec{x},\vec{y}) f(\vec{y}) \right] \end{aligned} \tag{145}$$

As $t \to 0$, the heat kernel $H_t(\vec{x},\vec{y})$ tends to a Dirac's delta function, and $\vec{x}$ tends to $\vec{y}$, that is:

$$\lim_{t \to 0} \int_M H_t(\vec{x},\vec{y}) f(\vec{y}) = f(\vec{x}) \tag{146}$$

which leads to:

$$\begin{aligned} Lf(\vec{x}) &\approx \frac{1}{t} \left[f(\vec{x}) - \int_M H_{\Delta t}(\vec{x},\vec{y}) f(\vec{y}) \right] \\ &\approx \frac{1}{t} \left[f(\vec{x}) - (4\pi\Delta t)^{(-d/2)} \int_M \exp\left\{ -\frac{\|\vec{x}-\vec{y}\|^2}{4\Delta t} \right\} f(\vec{y}) \right] \end{aligned} \tag{147}$$

If $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ is a random sample of points in the manifold M, then the previous expression can be discretized to:

$$Lf(\vec{x}_i) \approx \frac{1}{t} \left[f(\vec{x}_i) - \frac{1}{k} (4\pi t)^{-(d/2)} \sum_{0 \le \left\|\vec{x}_i - \vec{x}_j\right\| \le \varepsilon} \exp\left\{ -\frac{\left\|\vec{x}_i - \vec{x}_j\right\|^2}{4t} \right\} f(\vec{x}_j) \right] \tag{148}$$

where the coefficient $1/t$ is global and does not affect the eigenvectors of the discrete Laplacian. As the intrinsic dimensionality d is often unknown, we define:

$$\alpha = \frac{1}{k} (4\pi t)^{-(d/2)} \tag{149}$$

Knowing that the result of the Laplacian operator applied to the constant function is zero, it follows that:

$$L\vec{1} = \left[1 - \alpha \sum_{\vec{x}_j \in \eta_i} \exp\left\{ -\frac{\left\|\vec{x}_i - \vec{x}_j\right\|^2}{4t} \right\} \right] = 0 \tag{150}$$

where $\vec{1} = [1,1,...,1]$ and η_i denotes the neighborhood of $\vec{x}_i$. From this, we have:

$$\alpha = \left(\sum_{\vec{x}_j \in \eta_i} \exp\left\{ -\frac{\left\|\vec{x}_i - \vec{x}_j\right\|^2}{4t} \right\} \right)^{-1} \tag{151}$$

This result motivates the choice of the weights between neighboring nodes according to a Gaussian kernel in the Laplacian Eigenmaps method.

8. Locality Preserving Projections

One problem with nonlinear dimensionality reduction techniques like ISOMAP, LLE and Laplacian eigenmaps is that these methods are defined only on the training data points and it is unclear how to evaluate the map for new test points. The main motivation for the Locality Preserving Projection (LPP) algorithm is to produce a method that can be simply applied to any new test data point in order to locate it in the reduced representation space [37]. The basic idea of LPP is to provide a linear approximation of the nonlinear Laplacian Eigenmaps method.

As in the Laplacian Eigenmaps method, we seek a smooth map that preserves locality, that is, proximity in the graph must imply in proximity in the line. We have shown in previous sections that if we minimize the following criterion, then the map $\vec{y} = [y_1, y_2, ..., y_n]$ is optimal in that sense:

$$\vec{y}^T L \vec{y} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (y_i - y_j)^2 \tag{152}$$

where L is the Laplacian matrix of the KNN graph induced by the $m \times n$ data matrix $X = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n]$.

In LPP, it is assumed that the relationship between $\vec{x}_i \in R^m$ and $y_i \in R$ is linear, that is, $y_i = \vec{a}^T \vec{x}_i$, where $\vec{a} \in R^m$ is a column vector. Hence, the objective function can be expressed as:

$$\begin{aligned}
\vec{y}^T L \vec{y} &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (\vec{a}^T \vec{x}_i - \vec{a}^T \vec{x}_j)^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \left[\vec{a}^T \vec{x}_i \vec{x}_i^T \vec{a} - 2\vec{a}^T \vec{x}_i \vec{x}_j^T \vec{a} + \vec{a}^T \vec{x}_j \vec{x}_j^T \vec{a} \right] \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} 2w_{ij} \vec{a}^T \vec{x}_i \vec{x}_i^T \vec{a} - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} 2w_{ij} \vec{a}^T \vec{x}_i \vec{x}_j^T \vec{a} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \vec{a}^T \vec{x}_i \vec{x}_i^T \vec{a} - \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \vec{a}^T \vec{x}_i \vec{x}_j^T \vec{a}
\end{aligned} \tag{153}$$

Since $d_i = \sum_{j=1}^{n} w_{ij}$, we have:

$$\vec{y}^T L\vec{y} = \sum_{i=1}^{n} \vec{a}^T \vec{x}_i d_i \vec{x}_i^T \vec{a} - \sum_{i=1}^{n}\sum_{j=1}^{n} \vec{a}^T \vec{x}_i w_{ij} \vec{x}_j^T \vec{a} \tag{154}$$

Note that we can rewrite the equation using a matrix-vector notation as:

$$\vec{y}^T L\vec{y} = \vec{a}^T XDX^T \vec{a} - \vec{a}^T XWX^T \vec{a} \tag{155}$$

where X is the $m \times n$ data matrix, D is the $n \times n$ diagonal matrix of the degrees and W is the $n \times n$ weight matrix. Knowing that $L = D - W$, we finally reach:

$$\vec{y}^T L\vec{y} = \vec{a}^T X(D-W)X^T \vec{a} = \vec{a}^T XLX^T \vec{a} \tag{156}$$

Thus, we have to solve the following constrained minimization problem:

$$\arg\min_{\vec{a}} \ \vec{a}^T XLX^T \vec{a} \qquad \text{subject to} \qquad \vec{a}^T XDX^T \vec{a} = 1 \tag{157}$$

where the constraint is a general form to express that the norm of the vector $\vec{a}$ is a constant. The Lagrangian function is given by:

$$L(\vec{a}, \lambda) = \vec{a}^T XLX^T \vec{a} - \lambda(\vec{a}^T XDX^T \vec{a} - 1) \tag{158}$$

Taking the derivative with respect to $\vec{a}$ and setting the result to zero leads to:

$$\frac{\partial}{\partial \vec{a}} L(\vec{a}, \lambda) = XLX^T \vec{a} - \lambda XDX^T \vec{a} = 0 \tag{159}$$

Therefore, we have a generalized eigenvector problem:

$$XLX^T \vec{a} = \lambda XDX^T \vec{a} \tag{160}$$

$$(XDX^T)^{-1} XLX^T \vec{a} = \lambda \vec{a} \tag{161}$$

showing that in order to minimize the objective function, we shoud select the vector a as the smallest eigenvector of the matrix $(XDX^T)^{-1}XLX^T$. The multivariate version of the problem considers a $m \times d$ matrix A where each column $\vec{a}_j$ represents a direction in which data will be projected:

$$(XDX^T)^{-1}(XLX^T)A = \lambda A \tag{162}$$

In this case, we sould select to compose the cloumns of A the d eigenvalues associated to the d smallest eigenvalues of $(XDX^T)^{-1}XLX^T$. Algorithm 6 shows a summary of the LPP method for dimensionality reduction. Note that the transformation matrix A has m rows and d columns and the output matrix Y has d rows and n columns, meaning that each

column $\vec{y}_j$ for $j = 1,2,...,n$ stores the coordinates of the point after the dimensionality reduction.

Algorithm 6 Locality Preserving Projections

1: **function** LPP(X, K, d)
2: From the input data $X_{m\times n}$ build a KNN graph.
3: Choose the weights to define the adjacency matrix W.

$$W_{ij} = exp\left\{-\frac{\left\|\vec{x}_i - \vec{x}_j\right\|^2}{t}\right\} \qquad \text{if} \qquad v_j \in N(v_i) \tag{163}$$

$$W_{ij} = 0 \qquad \text{if} \qquad v_j \notin N(v_i) \tag{164}$$

4: Compute the diagonal matrix D with the degrees d_i for $i = 1,2,...,n$.

$$d_i = \sum_{j=1}^{n} W_{ij} \tag{165}$$

5: Compute the Laplacian matrix $L = D - W$
6: Select the bottom d eigenvectors of the matrix $(XDX^T)^{-1}XLX^T$:

$$A = \begin{bmatrix} | & | & ... & ... & | \\ | & | & ... & ... & | \\ \vec{v}_1 & \vec{v}_2 & ... & ... & \vec{v}_d \\ | & | & ... & ... & | \\ | & | & ... & ... & | \end{bmatrix}_{m\times d} \tag{166}$$

7: Project the data using the matrix A:

$$Y = A^T X \tag{167}$$

8: **return** Y
9: **end function**

8.1. *Kernel LPP*

As LPP is a linear approximation to the Laplacian Eigenmaps algorithm, we can make it nonlinear through kernel methods. Consider a nonlinear mapping $\phi : R^m \to R^M$, with $M > m$ and let $\phi(X)$ denote the data matrix in the Hilbert space R^M, that is, $\phi(X) = [\vec{x}_1, \vec{x}_2, ..., \vec{x}_n]$. Then, the eigenvector problem in the Hilbert space can be expressed by [37]:

$$\phi(X)L\phi(X)^T\vec{v} = \lambda\phi(X)D\phi(X)^T\vec{v} \tag{168}$$

which leads to the following generalized eigenvector problem:

$$(\phi(X)D\phi(X)^T)^{-1}\phi(X)L\phi(X)^T\vec{v} = \lambda\vec{v} \tag{169}$$

In order to generalize LPP to the nonlinear case, the problem has to be formulated in terms of inner products, since by the kernel trick we have:

$$K(\vec{x}_i,\vec{x}_j) = \phi(\vec{x}_i)^T\phi(\vec{x}_j) \tag{170}$$

As we have seen before in section 3, the eigenvectors in equation (169) can be expressed as a linear combination of $\phi(\vec{x}_1), \phi(\vec{x}_2), ..., \phi(\vec{x}_n)$, that is:

$$\vec{v} = \sum_{i=1}^{n} \alpha_i\phi(\vec{x}_i) = \phi(X)\vec{\alpha} \tag{171}$$

where $\vec{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_n]^T \in R^n$. Thus, equation (168) can be expressed as:

$$\phi(X)L\phi(X)^T\phi(X)\vec{\alpha} = \lambda\phi(X)D\phi(X)^T\phi(X)\vec{\alpha} \tag{172}$$

Left multiplication by $\phi(X)^T$ leads to:

$$\phi(X)^T\phi(X)L\phi(X)^T\phi(X)\vec{\alpha} = \lambda\phi(X)^T\phi(X)D\phi(X)^T\phi(X)\vec{\alpha} \tag{173}$$

Using the kernel trick, we can write:

$$KLK\vec{\alpha} = \lambda KDK\vec{\alpha} \tag{174}$$

and we finally reach:

$$(KDK)^{-1}(KLK)\vec{\alpha} = \lambda\vec{\alpha} \tag{175}$$

showing that we should choose as $\vec{\alpha}_1, \vec{\alpha}_2, ...\vec{\alpha}_d$ the d bottom eigenvectors of $(KDK)^{-1}(KLK)$. For a new vector $\vec{x}$ in the test set, the projections onto the eigenvectors $\vec{v}_k$, for $k = 1, 2, ..., d$ are given by:

$$\vec{v}_k^T\phi(\vec{x}) = \sum_{i=1}^{n} \alpha_k(i)\phi(\vec{x}_i)^T\phi(\vec{x}) = \sum_{i=1}^{n} \alpha_k(i)\phi(\vec{x})^T\phi(\vec{x}_i) = \sum_{i=1}^{n} \alpha_k(i)K(\vec{x},\vec{x}_i) \tag{176}$$

where $\alpha_k(i)$ is the i-th element of the vector $\vec{\alpha}_k$.

9. Hessian Eigenmaps

Roughly speaking, Hessian Eigenmaps can be considered as a variation of Locally Linear Embedding and Laplacian Eigenmaps. From a theoretical perspective, the main modification concerns the substitution of the Laplacian matrix by a quadratic form based on the Hessian matrix, which is the matrix of second-order derivatives. The main goal of Hessian Eigenmaps is to recover the local coordinates of the manifold in which the data is lying on, which is locally isometric to an open and connected subset of R^d. Unlike previous manifold learning methods (ISOMAP, LLE and Laplacian Eigenmaps), Hessian Eigenmaps does not require that the subset be convex, which is a significant generalization in terms of isometry classes that can be solved [28].

To better understand the reason why the Hessian matrix should be used to replace the Laplacian matrix, we provide an argument raised by Wang [86]. It has been shown that a linear function has everywhere vanishing Laplacian, but the opposite is not true, that is a function with everywhere vanishing Laplacian may not be linear. However, it is known that a function is linear if and only if it has everywhere vanishing Hessian. Thus, by replacing the Laplacian by the Hessian, it is possible to find an embedding that is, in practical terms, linear in every set of local tangent coordinates. Besides, since Hessian Eigenmaps dimensionality reduction process is based on a locally linear embedding, the original manifold is not required to be convex.

The theoretical background of Hessian Eigenmaps follows from mathematical properties of the quadratic form:

$$\mathscr{H}(f) = \int_M \left\| H_f(m) \right\|^2 dm \tag{177}$$

defined on functions $f : M \to R$. The quantity $\mathscr{H}(f)$ measures the average, over the data manifold M, of the Frobenius norm of the Hessian matrix of f. Suppose we have a parameter space $\Theta \subset R^d$ and a smooth mapping $\psi : \Theta \to R^n$, where the embedding space R^n obeys $d < n$. We define the data manifold, that is, the manifold on which our data $\vec{x}_i$ for $i = 1, 2, ..., n$ must lie, as $M = \psi(\Theta)$. The following result shows that the eigenfunctions of $\mathscr{H}(f)$ provide an orthogonal basis for R^d, so we can express the coordinates of the data points in this space [28].

Theorem 4. *Suppose $M = \psi(\Theta)$ where Θ is an open, connected subset of R^d, and ϕ is a locally isometric embedding of Θ into R^n. Then, $\mathscr{H}(f)$ has a (d+1) dimensional null space consisting of the constant function and a d dimensional space of functions spanned by the original isometric coordinates.*

One question that arises is: how to define the Hessian matrix at each point $\vec{x}_i$? The answer is that orthogonal coordinates on the tangent spaces of M are used to approximate $H_f(m)$ [28]. The basic idea consists in applying PCA to find a d dimensional tangent subspace at m, denoted by $T_m(M)$. In summary, the Hessian Eigenmaps method is composed by five main steps [95]:

(1) Choose the neighboring set for all points $\vec{x}_i$, for $i = 1, 2, ..., n$ (KNN graph)

(2) For each point $\vec{x}_i$, obtain the local tangent subspace coordinates.
(3) Compute least-squares Hessian estimates based on projections on the tangent subspace.
(4) Find a discrete approximation to $\mathscr{H}(f)$.
(5) Find embedding: choose the d eigenvectors associated to the d smallest non-zero eigenvalues of the approximation $\mathscr{H}(f)$.

9.1. *Approximating the functional* $\mathscr{H}(f)$

Suppose that $M \subset R^n$ is a smooth manifold and the tangent space $T_m(M)$ is well defined at each point $m \in M$. It is possible to associate to each tangent space an orthonormal coordinate system using the inner product from R^n. We can think of $T_m(M)$ as an affine subspace of R^n that is tangent to M at point m, with the origin $0 \in T_m(M)$ identified with m. There is a neighborhood N_m of m such that each point $m' \in N_m$ has a unique closest point $v' \in T_m(M)$. This point in $T_m(M)$ has coordinates given by our choice of orthonormal coordinates for $T_m(M)$. Let these local coordinates for a neighborhood N_m be represented by a d dimensional subspace spanned by $\vec{x}_1^{(tan,m)}, \vec{x}_2^{(tan,m)}, ..., \vec{x}_d^{(tan,m)}$ [28]. We can think of these vectors as being the eigenvectors of the local covariance matrix of the points in N_m.

Basically, the method uses the local coordinates to define the Hessian of a function $f : M \to R$ that is second order continuously differentiable near m. Suppose that $m' \in N_m$ has local coordinates $\vec{x} = \vec{x}'_{T(M)}$ in the tangent space. Then, the rule $g(\vec{x}) = f(m')$ defines a function $g : U \to R$, where U is the neighborhood of zero in R^d. The Hessian of f at m in tangent coordinates as the ordinary Hessian of g [28]:

$$\left(H_f^{(tan)}(m)\right)_{i,j} = \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j} g(\vec{x})\bigg|_{x=0} \tag{178}$$

In summary, at each point m, we use the tangent coordinates and differentiate f in that coordinate system. This is known as the tangent Hessian approximation. Given the above, the quadratic form we want to minimize is given by:

$$\mathscr{H}(f) = \int_M \left\| H_f^{(tan)}(m) \right\|^2 dm \tag{179}$$

In other words, $\mathscr{H}(f)$ measures the average curviness of the function f over the manifold.

9.2. *Building the Hessian estimator*

To build the Hessian estimator let us consider the case in which $d = 2$, that is, the intrinsic dimensionality of the manifold M is two. Define a matrix X^i consisting of the following columns:

$$X^i = \left[1, U_1, U_2, U_1^2, U_2^2, (U_1 \times U_2)\right] \tag{180}$$

where U_i is the i-th principal component of the local covariance matrix regarding the elements of the neighborhood N_i. For the general case $d > 2$, create a matrix with $1 + d + d(d+1)/2$ columns, where the first $d+1$ columns consists of a vector of ones followed by the d principal components of the local covariance matrix, and the remaining $d(d+1)/2$ columns are the squares of the principal components and the various cross products between them. For instance, if $d = 3$ we have:

$$X^i = \left[1, U_1, U_2, U_3, U_1^2, U_2^2, U_3^2, (U_1 \times U_2), (U_1 \times U_3), (U_2 \times U_3)\right] \tag{181}$$

Proceed performing the usual Gram-Schmidt orthonormalization process on the matrix X^i to obtain a matrix $\tilde{X}^i$ with orthonormal columns. The Hessian estimator H^i is defined by extracting the last $d(d+1)/2$ columns and transposing it, that is:

$$\left(H^i\right)_{r,l} = \left(\tilde{X}^i\right)_{l,1+d+r} \tag{182}$$

After estimating the Hessian at all points $\vec{x}_i$, for $i = 1, 2, ..., n$, we need to compute the quadratic form $\mathscr{H}$. Define a matrix $\mathscr{H}_{i,j}$ having in coordinate pair i, j the quantity:

$$\mathscr{H}_{i,j} = \sum_l \sum_r \left(H^l\right)_{r,i} \left(H^l\right)_{r,j} \tag{183}$$

where H^l is the $d(d+1)/2 \times k$ matrix associated with estimating the Hessian over neighborhood N^l, whose rows r correspond to specific entries in the Hessian Matrix and columns i correspond to specific points in the neighborhood.

9.3. *Finding the embedding coordinates*

In this step, we perform an eigendecomposition of the matrix $\mathscr{H}$ estimated previously and identify the (d+1) dimensional subspace corresponding to the (d+1) smallest eigenvalues. There is a zero eigenvalue associated to the constant $\vec{1}$ eigenvector that should be discarded. The next d eigenvalues will correspond to eigenvectors spanning a d dimensional space V_d in which the embedding coordinates are to be found.

Let V be the $n \times d$ matrix of nonconstant eigenvectors, where each column $\vec{v}_j$ for $j = 1, 2, ..., d$ denotes one distinct eigenvector, and let $V_{l,r}$ denote the l-th entry of the r-th eigenvector of $\mathscr{H}$. Defining the matrix R as:

$$(R)_{r,s} = \sum_{j=1}^{n} V_{j,r} V_{j,s} \tag{184}$$

Finally, the desired $n \times d$ matrix of embedding coordinates, where each line $\vec{w}_i$, for $i = 1, 2, ..., n$ represents a point in R^d, is given by:

$$W = VR^{-1/2} \tag{185}$$

Despite its strong asymptotic guarantees, the Hessian Eigenmaps method has some known issues. First, it assumes that data points lie precisely on some manifold M, an unrealistic hypothesis that for many real datasets. We can at most assure that the data is close in a probabilistic sense to a manifold M. Also, it requires local, empirical estimates of differential operators. Some problems can happen to these estimates when the manifold is sparsely or irregularly sampled.

10. t-distributed Stochastic Neighbor Embedding

To understand t-SNE we need to present its predecessor and motivational method: Stochastic Neighbor Embedding, or simply SNE [39]. SNE starts by converting Euclidean distances between data points $\vec{x}_i \in R^m$ for $i = 1, 2, ..., n$ into conditional probabilities as a way to represent similarity. The similarity between two points $\vec{x}_i$ and $\vec{x}_j$ is the conditional probability $p_{j|i}$ that $\vec{x}_i$ would choose $\vec{x}_j$ as a neighbor if neighbor selection is done in proportion to a probability density under a Gaussian centered at $\vec{x}_i$ [79]:

$$p_{j|i} = \frac{\exp\left(-\left\|\vec{x}_i - \vec{x}_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\vec{x}_i - \vec{x}_k\right\|^2 / 2\sigma_i^2\right)} \tag{186}$$

where σ_i^2 is the variance of the Gaussian centered on $\vec{x}_i$. Note that if $\vec{x}_i$ and $\vec{x}_j$ are close enough then $p_{j|i}$ has a significantly high value, but if they are widely separated $p_{j|i}$ tends to zero. It is possible to compute a similar conditional probability for the low dimensional representation vectors $\vec{y}_i$ and $\vec{y}_j$ in R^d, denoted by $q_{j|i}$. Setting the variance of the Gaussian to $1/\sqrt{2}$, the similarity measure is given by:

$$q_{j|i} = \frac{\exp\left(-\left\|\vec{y}_i - \vec{y}_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\vec{y}_i - \vec{y}_k\right\|^2\right)} \tag{187}$$

Note that, as we are modeling pairwise similarities, $p_{i|i} = q_{i|i} = 0$. The idea is that the low dimensional representation points $\vec{y}_i$ and $\vec{y}_j$ correctly model the similarities between the high dimensional vectors $\vec{x}_i$ and $\vec{x}_j$, then the conditional probabilities $p_{j|i}$ and $q_{j|i}$ are equal. The goal of SNE is to find a low dimensional representation that minimizes the distance between the two probabilities, making them as close as possible. A statistical measure of how close two probability distributions are is the Kullback-Leibler divergence, also known as relative entropy. SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method. The objetive function to be minimized is [39]:

$$C = \sum_{i=1}^{n} KL(P_i||Q_i) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{188}$$

where P_i epresents the conditional probability distribution over all other data points given data point $\vec{x}_i$ and Q_i represents the conditional probability distribution over all other map points given map point $\vec{y}_i$. In other words, the SNE cost function focuses on retaining the local structure of the data in the map (for reasonable values of the variance of the Gaussian in the high-dimensional space, σ_i^2).

10.1. *Defining the variance of the Gaussians*

In order to work properly, SNE has to define the variance σ_i^2 of the Gaussian that is centered over each high dimensional point $\vec{x}_i \in R^m$. It is not reasonable to assume that there is a single value of σ_i^2 that is optimal for all points. In dense regions, a smaller value of σ_i^2 is more appropriate than in sparse regions. A given σ_i^2 induces a probability distribution P_i, which has an entropy that is an increasing function of the variance. The perplexity measure is then defined as [79]:

$$\text{Perp}(P_i) = 2^{H(P_i)} \tag{189}$$

where $H(P_i)$ is the Shannon entropy in bits:

$$H(P_i) = -\sum_{j=1}^{n} p_{j|i} log_2 p_{j|i} \tag{190}$$

SNE searches for a value of σ_i^2 that produces a P_i with a fixed perplexity, defined by the user. The perplexity can be interpreted as a smooth measure of the effective number of neighbors, and typical values are between 5 and 50 [79]. The minimization of the objective function (KL divergence) is performed by a gradient descent approach with momentum to speed up convergence, that is, after the initialization, the coordinates of the low dimensional points are iteratively updated by:

$$\mathscr{Y}^{(t)} = \mathscr{Y}^{(t-1)} - \eta \frac{\partial C}{\partial \mathscr{Y}} + \alpha(t)\left(\mathscr{Y}^{(t-1)} - \mathscr{Y}^{(t-2)}\right) \tag{191}$$

where $\mathscr{Y}^{(t)}$ denotes the solution at iteration t, η denotes the learning rate and $\alpha(t)$ represents the momentum at iteration t.

10.2. *Computing the gradient in SNE*

It is clear that from equation (191) the key ingredient to the iterative algorithm is the computation of the gradient. At each stage of the optimization, we start with a set of points $\vec{y}_1.\vec{y}_2, ..., \vec{y}_n$ and we need to estimate the gradient. The process can be summarized as [55]:

- Create a distance matrix, where the element d_{ij} represents the Euclidean distance between $\vec{y}_i$ and $\vec{y}_j$;
- Transform the distances to create f_{ij} (usually, squaring the distances);

- Apply a weighting function to define a weight w_{ij}, such that the larger the weight, the smaller the distance (similarity measure);
- Convert the weights into probabilities $q_{ij} = q_{j|i}$, by normalizing over the sum of the weights;

Having the probabilities, we are ready to compute the gradient. In the original SNE algorithm, the weights are converted into probabilities by:

$$q_{j|i} = q_{ij} = \frac{w_{ij}}{\sum_k w_{ik}} = \frac{w_{ij}}{Z_i} \tag{192}$$

where $w_{ij} = \exp\left(-\left\|\vec{y}_i - \vec{y}_j\right\|^2\right)$. Note that $w_{ij} = w_{ji}$. Renaming the variables i, j to k, l, the objective function becomes [30]:

$$\begin{aligned} C &= \sum_{k=1}^{n}\sum_{l=1}^{n}(p_{kl}\log p_{kl} - p_{kl}\log q_{kl}) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n}(p_{kl}\log p_{kl} - p_{kl}\log w_{kl} + p_{kl}\log Z_k) \end{aligned} \tag{193}$$

Deriving the objective function with relation to $\vec{y}_i$ we have:

$$\frac{\partial C}{\partial \vec{y}_i} = -\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log w_{kl} + \sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log Z_k \tag{194}$$

Note that in the first term, the derivative is non zero when $k = i$ or $l = i$, leading to:

$$-\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log w_{kl} = -\sum_{j=1}^{n}\left(p_{ij}\frac{\partial}{\partial \vec{y}_i}\log w_{ij} + p_{ji}\frac{\partial}{\partial \vec{y}_i}\log w_{ji}\right) \tag{195}$$

Computing the derivative of $\log w_{ij}$ with respect to $\vec{y}_i$ gives us:

$$\frac{\partial}{\partial \vec{y}_i}\log w_{ij} = -2\frac{1}{w_{ij}} w_{ij}(\vec{y}_i - \vec{y}_j) \tag{196}$$

and the derivative of $\log w_{ji}$ with respect to $\vec{y}_i$ is:

$$\frac{\partial}{\partial \vec{y}_i}\log w_{ji} = 2\frac{1}{w_{ij}} w_{ij}(\vec{y}_j - \vec{y}_i) \tag{197}$$

Hence, the first term of the gradient becomes:

$$\begin{aligned} &-\sum_{j=1}^{n}\left(-2p_{ij}\frac{1}{w_{ij}} w_{ij}(\vec{y}_i - \vec{y}_j) + 2p_{ji}\frac{1}{w_{ji}} w_{ji}(\vec{y}_j - \vec{y}_i)\right) = \\ &-\sum_{j=1}^{n}(-2p_{ij}(\vec{y}_i - \vec{y}_j) - 2p_{ji}(\vec{y}_i - \vec{y}_j)) = 2\sum_{j=1}^{n}(p_{ij} + p_{ji})(\vec{y}_i - \vec{y}_j) \end{aligned} \tag{198}$$

In the second term, note that Z_k does not depend on l, so it can be taken out the internal summation:

$$\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log Z_k = \sum_{k=1}^{n}\frac{\partial}{\partial \vec{y}_i}\log Z_k \sum_{l=1}^{n} p_{kl} = \sum_{k=1}^{n}\frac{\partial}{\partial \vec{y}_i}\log Z_k \tag{199}$$

since the sum of the elements of P_k (a row of the matrix P) is one. Changing the variable k to j and differentiating the logarithm leads to:

$$\sum_{j=1}^{n}\frac{1}{Z_j}\sum_{k=1}^{n}\frac{\partial w_{jk}}{\partial \vec{y}_i} \tag{200}$$

Note that the partial derivative is non zero only when $k = i$ or $j = i$, so we can write:

$$\sum_{j=1}^{n}\left[\frac{1}{Z_j}\frac{\partial w_{ji}}{\partial \vec{y}_i} + \frac{1}{Z_i}\frac{\partial w_{ij}}{\partial \vec{y}_i}\right] \tag{201}$$

Computing the derivatives leads to:

$$\begin{aligned}\sum_{j=1}^{n}\left(\frac{1}{Z_j}2w_{ji}(\vec{y}_j-\vec{y}_i)\right) - \sum_{j=1}^{n}\left(\frac{1}{Z_i}2w_{ij}(\vec{y}_i-\vec{y}_j)\right) = \\ -2\sum_{j=1}^{n}\left(\frac{w_{ji}}{Z_j}(\vec{y}_i-\vec{y}_j)\right) - 2\sum_{j=1}^{n}\left(\frac{w_{ij}}{Z_i}(\vec{y}_i-\vec{y}_j)\right) = \\ -2\sum_{j=1}^{n}(q_{ji}+q_{ij})(\vec{y}_i-\vec{y}_j)\end{aligned} \tag{202}$$

Finally, combining equations (198) and (202) we have:

$$\begin{aligned}\frac{\partial C}{\partial \vec{y}_i} &= 2\sum_{j=1}^{n}(p_{ij}+p_{ji})(\vec{y}_i-\vec{y}_j) - 2\sum_{j=1}^{n}(q_{ji}+q_{ij})(\vec{y}_i-\vec{y}_j) \\ &= 2\sum_{j=1}^{n}(p_{ij}-q_{ij}+p_{ji}-q_{ji})(\vec{y}_i-\vec{y}_j)\end{aligned} \tag{203}$$

According to the authors, the gradient can be interpreted as the resultant force created by a set of springs between the map point $\vec{y}_i$ and all the points $\vec{y}_j$. The force exerted by the spring between $\vec{y}_i$ and $\vec{y}_j$ is proportional to its length, and also proportional to its stiffness, which is the mismatch $(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})$ between the pairwise similarities of the data points and the map points [79].

10.3. *The crowding problem in SNE*

Let's begin by considering that we have a manifold M with ten intrinsic dimensions embedded in a space S of dimensionality $d >> 10$. The question is: is it possible to suitably map

the pairwise distances on the space S to pairwise distances between points in the manifold M? The answer for this question is no, and in the following we provide some arguments originally described by van der Maaten and Hinton [79]. First, note that in S it is possible to have a huge number of points that are mutually equidistant. Clearly, there is no feasible way to organize these points on the ten dimensional manifold M, because there is too many of them. This is exactly what defines the crowding problem. Basically, the crowding problem can be summarized as: the area in a low dimensional map that is available to place distant points will not be large enough in comparison with the area available to place nearby points. In practical terms, to accurately model the small distances in the map, most distant points from $\vec{x}_i$ will have to be scattered too far away in the map. Therefore, the authors argue that one way to alleviate this problem is to consider, in the low dimensional map, distributions with heavier tails than a Gaussian model. That's why the Student's t distribution is chosen as a replacement to the traditional Gaussian distribution used in the SNE algorithm [39].

10.4. *Computing the gradient in t-SNE*

The cost function employed by t-SNE differs from the one used by SNE in two ways:

(1) it uses a symmetrized version of the SNE cost function with simple gradient computation [22].
(2) it uses a Student t distribution rather than a Gaussian to compute the similarity between two points in the low-dimensional space, a heavy-tailed distribution that alleviates the crowding problem as well as optimization problems.

In the symmetric version, we minimize a single KL divergence between a joint probability distribution P in the high-dimensional space and a joint probability distribution Q in the low dimensional space:

$$C = KL(P||Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}\log\frac{p_{ij}}{q_{ij}} \tag{204}$$

where $p_{ii} = q_{ii} = 0$, $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$, $\forall i, j$.

In the symmetric version employed in t-SNE, the pairwise similarities in the high dimensional space are given by:

$$p_{ij} = \frac{\exp\left(-\left\|\vec{x}_i - \vec{x}_j\right\|^2 / 2\sigma^2\right)}{\sum_{k\neq l}\exp\left(-\left\|\vec{x}_k - \vec{x}_l\right\|^2 / 2\sigma^2\right)} \tag{205}$$

where the normalization summation in the denominator involves all pairs of points, not only the the points linked to $\vec{x}_i$, as in the original SNE.

Using the Student's t distribution with one degree of freedom the joint probabilities q_{ij} are defined as:

$$q_{ij} = \frac{\left(1 + \left\|\vec{y}_i - \vec{y}_j\right\|^2\right)^{-1}}{\sum_{k \neq l}\left(1 + \left\|\vec{y}_k - \vec{y}_l\right\|^2\right)^{-1}} = \frac{w_{ij}^{-1}}{\sum_{k \neq l} w_{kl}^{-1}} = \frac{w_{ij}^{-1}}{Z} \tag{206}$$

so that the objective function is:

$$\begin{aligned} C &= \sum_{k=1}^{n}\sum_{l=1}^{n}\left(p_{kl}\log\, p_{kl} - p_{kl}\log\, q_{kl}\right) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n}\left(p_{kl}\log\, p_{kl} - p_{kl}\log\, w_{kl}^{-1} + p_{kl}\log\, Z\right) \end{aligned} \tag{207}$$

To find the gradient, we differentiate with respect to $\vec{y}_i$, leading to:

$$\frac{\partial C}{\partial \vec{y}_i} = -\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log\, w_{kl}^{-1} + \sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log\, Z \tag{208}$$

For the first term of the gradient, note that the derivative is non zero when $\forall\, j$, $k = i$ or $l = i$. Besides, note that $p_{ij} = p_{ji}$ and $w_{ij} = w_{ji}$, which leads to:

$$\begin{aligned} -\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log\, w_{kl}^{-1} &= -\sum_{j=1}^{n}\left(p_{ij}\frac{\partial}{\partial \vec{y}_i}\log\, w_{ij}^{-1} + p_{ji}\frac{\partial}{\partial \vec{y}_i}\log\, w_{ji}^{-1}\right) \\ &= -2\sum_{j=1}^{n} p_{ij}\frac{\partial}{\partial \vec{y}_i}\log\, w_{ij}^{-1} \end{aligned} \tag{209}$$

Note that the derivative of the inverse of the weight is:

$$\frac{\partial w_{ij}^{-1}}{\partial \vec{y}_i} = -2w_{ij}^{-2}\left(\vec{y}_i - \vec{y}_j\right) \tag{210}$$

so the first term of the gradient becomes:

$$4\sum_{j=1}^{n} p_{ij} w_{ij}^{-1}\left(\vec{y}_i - \vec{y}_j\right) \tag{211}$$

To derive the second term of the gradient, we use the fact that Z does not depend on k or l and that the summation of the probabilities is equal to one:

$$\begin{aligned}\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \vec{y}_i}\log Z &= \frac{\partial}{\partial \vec{y}_i}\log Z \sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl} = \frac{\partial}{\partial \vec{y}_i}\log Z \\ &= \frac{1}{Z}\frac{\partial}{\partial \vec{y}_i} Z = \frac{1}{Z}\sum_{k=1}^{n}\sum_{l=1}^{n}\frac{\partial}{\partial \vec{y}_i} w_{kl}^{-1}\end{aligned} \tag{212}$$

Once again, the derivative is non zero when $\forall j$, $k = i$ or $l = i$, that is:

$$\frac{1}{Z}\sum_{k=1}^{n}\sum_{l=1}^{n}\frac{\partial}{\partial \vec{y}_i} w_{kl}^{-1} = \frac{1}{Z}\sum_{j=1}^{n}\left(\frac{\partial}{\partial \vec{y}_i} w_{ij}^{-1} + \frac{\partial}{\partial \vec{y}_i} w_{ji}^{-1}\right) \tag{213}$$

As $w_{ij} = w_{ji}$ we have:

$$\frac{1}{Z}\sum_{j=1}^{n}\left(\frac{\partial}{\partial \vec{y}_i} w_{ij}^{-1} + \frac{\partial}{\partial \vec{y}_i} w_{ji}^{-1}\right) = 2\sum_{j=1}^{n}\frac{1}{Z}\frac{\partial}{\partial \vec{y}_i} w_{ij}^{-1} \tag{214}$$

From equation (210), we can write:

$$2\sum_{j=1}^{n}\frac{1}{Z}\frac{\partial}{\partial \vec{y}_i} w_{ij}^{-1} = -4\sum_{j=1}^{n}\frac{w_{ij}^{-1}}{Z} w_{ij}^{-1}\left(\vec{y}_i - \vec{y}_j\right) = -4\sum_{j=1}^{n} q_{ij} w_{ij}^{-1}\left(\vec{y}_i - \vec{y}_j\right) \tag{215}$$

Finally, combining equations (211) and (215), we have the gradient for the t-SNE iteration:

$$\begin{aligned}\frac{\partial C}{\partial \vec{y}_i} &= 4\sum_{j=1}^{n}\left(p_{ij} - q_{ij}\right) w_{ij}^{-1}\left(\vec{y}_i - \vec{y}_j\right) \\ &= 4\sum_{j=1}^{n}\left(p_{ij} - q_{ij}\right)\left(1 + \left\|\vec{y}_i - \vec{y}_j\right\|^2\right)^{-1}\left(\vec{y}_i - \vec{y}_j\right)\end{aligned} \tag{216}$$

Algorithm 7 shows a summary of the t-SNE method for dimensionality reduction. The parameters of the algorithm are the input data set $X = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$, the number of iterations T, the perplexity *Perp*, which can be interpreted as a smooth measure of the effective number of neighbors (typical values are between 5 and 50), the learning rate η and the momentum $\alpha(t)$, used in the gradient descent optimization method.

Algorithm 7 t-distributed Stochastic Neighboring Embedding

function T-SNE($X, Perp, T, \eta, \alpha(t)$)
  Compute pairwise probabilities $p_{j|i}$ and $p_{i|j}$ using equation (186).
  Set the value of p_{ij} as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{217}$$

  Sample initial solution $\mathscr{Y}^{(0)} = \{\vec{y}_1, \vec{y}_2, ..., \vec{y}_n\}$ from $\mathscr{N}(0, 10^{-4}I)$.
  for t = 1 to T **do**
    Compute low dimensional affinities q_{ij} using equation (206).
    Compute the gradient using equation (216).
    Update the coordinates with gradient descent with momentum:

$$\mathscr{Y}^{(t)} = \mathscr{Y}^{(t-1)} - \eta \frac{\partial C}{\partial \mathscr{Y}} + \alpha(t)\left(\mathscr{Y}^{(t-1)} - \mathscr{Y}^{(t-2)}\right) \tag{218}$$

  end for
  return $\mathscr{Y}^{(T)}$
end function

11. Semidefinite Embedding

Semidefinite Embedding (SDE), or Maximum Variance Unfolding (MVU) [88], is a manifold learning algorithm based on semidefinite programming [81]. The main idea behind SDE is that if we learn a kernel matrix optimizing the criterion of variance maximization in the feature space while preserving the distances and angles between neighboring samples, then we can unfold the manifold in which data is lying on [90,91]. Moreover, it has been shown that SDE has several nice properties [92]: 1) the optimization problem is convex and the objective function preserves the local geometry; 2) the kernel function learned by SDE is always a semipositive definite matrix; 3) it is possible to estimate the intrinsic dimensionality through the eigenvalues of the kernel matrix; 4) there is no need to compute approximations to geodesic distances between samples.

Basically, the main concept behind SDE is the notion of isometry. An isometry is a smooth invertible mapping that looks locally like a rotation plus translation, that is, it is a map that preserves distances. In the 2D scenario, we can think of isometry any kind of transformation we can perform on a sheet of paper without making holes, tears or self-intersections [90].

In the following, we briefly describe the notion of isometry between two datasets $X = \{\vec{x}_i\}_{i=1}^n$ and $Y = \{\vec{y}_i\}_{i=1}^n$. Let the $n \times n$ matrix η indicate an adjacency relation on the elements of X and Y, such that $\vec{x}_j$ is a neighbor of $\vec{x}_i$ if and only if $\eta_{ij} = 1$. The datasets X and Y are locally isometric under η if, for every point $\vec{x}_i$, there exists a rotation, reflection and/or translation that maps $\vec{x}_i$ and its neighbors onto $\vec{y}_i$ and its neighbors [90]. Note that the local mapping between neighborhoods exists if and only if the distances and angles between points and neighbors are preserved. For local isometry, we must have:

$$(\vec{y}_i - \vec{y}_j)(\vec{y}_i - \vec{y}_k) = (\vec{x}_i - \vec{x}_j)(\vec{x}_i - \vec{x}_k) \tag{219}$$

because the triangle formed by a point and its neighbors is determined up to rotation, reflection and translation by specifying two sides and the angle between them. Moreover, X and Y are locally isometric under η if whenever $\vec{x}_i$ and $\vec{x}_j$ are neighbors or common neighbors of another point, we have:

$$\left\|\vec{y}_i - \vec{y}_j\right\|^2 = \left\|\vec{x}_i - \vec{x}_j\right\|^2 \tag{220}$$

$$\vec{y}_i^T\vec{y}_i - \vec{y}_i^T\vec{y}_j - \vec{y}_j^T\vec{y}_i + \vec{y}_j^T\vec{y}_j = \vec{x}_i^T\vec{x}_i - \vec{x}_i^T\vec{x}_j - \vec{x}_j^T\vec{x}_i + \vec{x}_j^T\vec{x}_j \tag{221}$$

Let $G_{ij} = \vec{x}_i^T\vec{x}_j$ and $K_{ij} = \vec{y}_i^T\vec{y}_j$ denote the Gram matrices of the inputs and outputs. Then, we have:

$$K_{ii} + K_{jj} - K_{ij} - K_{ji} = G_{ii} + G_{jj} - G_{ij} - G_{ji} \tag{222}$$

At this point, a question is arised: can we find a Gram matrix K that satisfies the constraint in equation (222) for which the vectors $\vec{y}_i$ lie in a subspace of dimensionality $d < m$? Let's begin by building a KNN graph $G = (V, E, w)$, that is, we connect each sample to its k nearest neighbors. As mentioned before, by imposing the local isometry constraints under this neighborhood system, we must assure that the weights of the edges in G, that is $w(e)$ for $e \in E$ are preserved and at the same time the angles between these edges are kept untouched. Suppose we add new edges $e' \notin E$ to G in order to connect the neighbors of each node, generating a new graph G'. Note that if we preserve the distances of all edges in G', we guarantee the local isometry constraints because if there is no change in the weights of the new edges, then the angles of the edges in the original KNN graph are preserved.

Another constraint is to have the outputs centered on the origin:

$$\sum_{i=1}^{n} \vec{y}_i = 0 \tag{223}$$

which can be expressed in terms of the Gram matrix K as:

$$0 = \left\|\sum_{i=1}^{n} \vec{y}_i\right\|^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} \vec{y}_i^T\vec{y}_j = \sum_{i=1}^{n}\sum_{j=1}^{n} K_{ij} \tag{224}$$

Thus, SDE defines a manifold learning algorithm as an optimization problem over a Gram matrix K rather than vectors $\vec{y}_i$. Not all matrices, however, can be interpreted as Gram matrices: only symmetric matrices with nonnegative eigenvalues can be interpreted in this way. Thus, we must further constrain the optimization to the cone of semidefinite matrices, that is, $K \succcurlyeq 0$ [81].

However, what could be a suitable function of the Gram matrix K to optimize in order to unfold a manifold? Note that any "fold" between two points on a manifold decreases the

Euclidean distance between the points. This suggests an optimization rule: maximize the sum of pairwise squared distances between outputs, that is [90]:

$$J(Y) = \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\|\vec{y}_i - \vec{y}_j\right\|^2 \tag{225}$$

By maximizing the above equation, SDE pulls the outputs as far away as possible, subject to the constraints defined earlier. Note that an advantage of this particular objective function is that it is upper bounded, which means we cannot pull the outputs infinitely far apart. Let τ be the maximal distance between any two neighbors:

$$\tau = \max_{ij}\left[\eta_{ij}\left\|\vec{x}_i - \vec{x}_j\right\|\right] \tag{226}$$

For a conected graph, the size of the longest path is at most $n\tau$. Besides, the size of a path is an upper bound for the Euclidean distance between their extrema. Thus, for all $\vec{y}_i$ and $\vec{y}_j$, we have $\left\|\vec{y}_i - \vec{y}_j\right\| \leq n\tau$, which leads to an upper bound for the objective function:

$$J(Y) \leq \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(n\tau)^2 = \frac{1}{2n}n^2n^2\tau^2 = \frac{n^3\tau^2}{2} \tag{227}$$

It is possible to express the objective function in terms of the Gram matrix K of the outputs $\vec{y}_i$. Note that:

$$\begin{aligned} J(Y) &= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\|\vec{y}_i - \vec{y}_j\right\|^2 = \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\vec{y}_i - \vec{y}_j\right)^T\left(\vec{y}_i - \vec{y}_j\right) \\ &= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\vec{y}_i^T\vec{y}_i - \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\vec{y}_i^T\vec{y}_j - \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\vec{y}_j^T\vec{y}_i + \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}\vec{y}_j^T\vec{y}_j \\ &= \frac{1}{2}\sum_{i=1}^{n}K_{ii} + \frac{1}{2}\sum_{j=1}^{n}K_{jj} = \sum_{i=1}^{n}K_{ii} = Tr(K) \end{aligned} \tag{228}$$

We can interpret the objective function for the outputs in several ways: as a sum over pairwise distances, as a measure of variance or as the trace of their Gram matrix. According to the authors, the maximization of variance reminds of PCA, but whereas in PCA we compute the best linear projection, in SDE we compute the locally isometric embedding. In other words, the ob- jective function remains the same, but we change the allowed form of the dimensionality reduction [90].

In summary, the SDE problem can be summarized as: maximize the variance of the outputs $\{\vec{y}_i\}_{i=1}^n$ subject to the constraints that they are centered and locally isometric to the inputs $\{\vec{x}_i\}_{i=1}^n$:

$$\begin{aligned}
&\max\ Tr(K) \text{ subject to} \\
&K \succcurlyeq 0 \\
&\sum_{i=1}^{n}\sum_{j=1}^{n} K_{ij} = 0 \\
&K_{ii} + K_{jj} - K_{ij} - K_{ji} = G_{ii} + G_{jj} - G_{ij} - G_{ji} \\
&\forall i, j \text{ such that } \eta_{ij} = 1 \text{ or } [\eta^T \eta]_{ij} = 1
\end{aligned} \tag{229}$$

This problem is an instance of semidefinite programming (SDP): the domain is the cone of semidefinite matrices intersected with hyperplanes (represented by equality constraints), and the objective function is linear in the matrix elements. It is also convex, eliminating the possibility of local maxima [90]. Several Matlab and Python libraries for solving semidefinite programming problems can be found in the literature, among which we can cite CVXOPT [2], a free software package for convex optimization, and PyLMI-SDP [71], a free Python package for symbolic linear matrix inequalities (LMI) and semi-definite programming (SDP) tools.

From the Gram matrix K learned by the optimization problem, we recover the coordinates of the outputs $\vec{y}_i$ through its eigendecomposition. Let $\vec{v}_j$ denote the j-th eigenvector of K, with eigenvalue λ_j. The Gram matrix K can be expressed as:

$$K = \sum_{j=1}^{n} \lambda_j \vec{v}_j \vec{v}_j^T \tag{230}$$

Denoting by Λ the diagonal matrix of the eigenvalues, an n-dimensional embedding that is locally isometric to the inputs $\vec{x}_i$ is obtained by:

$$\vec{y}_i = \Lambda^{1/2} \vec{v}_i \tag{231}$$

The eigenvalues of K are all non-negative. Without loss of generality, we assume that the eigenvalues are sorted from largest to smallest. The quality of the approximation depends on how many eigenvalues and eigenvectors we select to keep. A large gap in the eigenvalue spectrum indicates the input lie on or near a d dimensional manifold [90].

12. Local Tangent Space Aligment

Local tangent space alignment (LTSA) is a method for manifold learning, originally proposed by Zhang and Zha [99], which can efficiently learn a nonlinear embedding into low-dimensional coordinates from high-dimensional data. Basically, the intuition behind LTSA is: when a manifold is correctly unfolded, all of the tangent hyperplanes to the manifold will become aligned. Figure 3, originally from [87], illustrates how the tangent spaces in a curved manifold are not aligned.

The geometric intuitions of LTSA are related to LLE, but with some specific differences: first, recall that in LLE each sample and the set of its neighbors define a locally

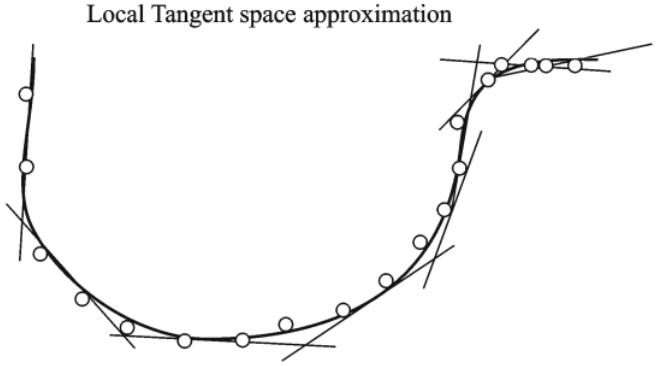

Figure 3. In a curved manifold the tangent spaces, used to to represent the local geometry, are not aligned.

linear patch. Using least squares, the optimal reconstruction weights are computed and finally, the best low dimensional coordinates (embedding) are obtained preserving the optimal weights. The main difference is that in LTSA the locally linear patch is built by approximating the tangent space at a given sample through the application of PCA on the neighborhood system. As each neighbor has its coordinate representation in the tangent space, we use them as a low dimensional representation of the patch. Then, all the local representations (patches) are aligned to a global one. Basically, the main steps of LTSA can be summarized as [87]:

(1) Build a KNN graph from the data points
(2) Compute the tangent space at every point by selecting the first d principal components
(3) Solve an optimization problem to find an embedding that aligns the tangent spaces

In the following, we present a brief introduction to some mathematical concepts employed in the derivation of the LTSA algorithm. Our description is based on the definitions of Wang in his book on high-dimensional data and nonlinear dimensionality reduction [87].

12.1. *Tangent coordinates*

Let $M \subset R^D$ be a d-dimensional manifold and $f : R^d \rightarrow M$ be a parametrization of M such that for each $\vec{x} \in M$, $\vec{x} = f(\vec{y})$, $\vec{y} \in R^d$. Let the inverse of f, denoted by $h = f^{-1}$ be the coordinate mapping on M so that $\vec{y} = h(\vec{x})$, $\vec{x} \in M$, provides the intrinsic coordinates of $\vec{x}$. Suppose we are given a dataset $X = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$ that lies on a unknown manifold M. Then, if we are able to find the mapping $\vec{y}_i = h(\vec{x}_i)$, $1 \leq i \leq n$, the coordinate set $Y = \{\vec{y}_1, \vec{y}_2, ..., \vec{y}_n\}$ represents an embedding of M in R^d, that is, the Euclidean geometry assumptions are valid.

For each point $p \in M$, we can learn the local geometry of its neighborhood O_p from the local tangent space T_pM. Hence, the orthogonal projection of O_p into T_pM, denoted by Q_p, can be approximated by:

$$Q_p(\vec{x}-\vec{p}) \approx \vec{x}-\vec{p} \tag{232}$$

for $\vec{x} \in O_p$. We will define an orthonormal basis on the tangent space T_pM as $U = \{\vec{u}_1, \vec{u}_2, ..., \vec{u}_d\}$. Thus, we can express the orthogonal projection as a linear combination of the basis vectors:

$$Q_p(\vec{x}-\vec{p}) = \sum_{i=1}^{d} \alpha_i \vec{u}_i \tag{233}$$

for $\vec{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_d] \in R^d$. Therefore, we have:

$$\vec{x}-\vec{p} \approx \sum_{i=1}^{d} \alpha_i \vec{u}_i \tag{234}$$

which provides a local representation for $\vec{x}$.

However, each point $\vec{x} \in M$ also has a set of manifold coordinates, known as the global geometry $\vec{x} = f(\vec{y})$, $\vec{y} \in R^d$. We will call $\vec{q} = h(\vec{p})$ and $f'(\vec{q})$ the derivative of f at $\vec{q}$. Thus, we can use a first order Taylor's expansion, omitting second and higher order terms, to write the following approximation:

$$\vec{x}-\vec{p} \approx f'(\vec{q})(\vec{y}-\vec{q}) \tag{235}$$

where $f'(\vec{q})$ is an invertible linear transformation. Note that equations (234) and (235) show that the manifold coordinate $\vec{y}$ of the vector $\vec{x}$ can be obtained by aligning the local coordinate $\vec{\alpha}$ of $\vec{x}$. In the following, we describe how to obtain the local coordinates.

12.2. *Obtaining the local coordinate*

Suppose we have as input the dataset $X = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$. Similarly to the previous algorithms, the first step consists in building a KNN graph from X. We will denote $O(i)$ the neighborhood of the sample $\vec{x}_i$ and $N(i) = \{i_1, i_2, ..., i_k\}$ the index set of $O(i)$, that is, $N(i)$ stores the indices of all neighbors of $\vec{x}_i$. It is straightforward to see that the set $N(i)$ is formed by the non-zero entries of the i-th row of the adjacency matrix of the graph, denoted here by A. Since each vertex in the graph has k neighbors, the number of elements in $N(i)$ is also k. We will define the geometric center of $O(i)$ as the average of the neighboring samples:

$$\bar{\vec{x}} = \frac{1}{k} \sum_{j \in N(i)} \vec{x}_j \tag{236}$$

For our purposes, let's assume that $\bar{\vec{x}} \in M$ and let T_i be the tangent space of M at the geometric center of the neighborhood $\bar{\vec{x}}$. Then we can define the tangent hyperplane as $H_i = \bar{\vec{x}} + T_i$. Since all points in a neighborhood are close together, it is intuitive to see that

they must be very similar to their projections on the tangent hyperplane H_i. Define the orthogonal projection $F : O(i) \rightarrow H_i$ such that:

$$F(\vec{x}_j) = \bar{\vec{x}} + \vec{p}_j \tag{237}$$

for $j \in N(i)$ and $\vec{p}_j \in T_i$. It is important to note that the vectors $\vec{p}_j$ belong to the tangent space. Then, by averaging the vectors $\vec{p}_j$, we have:

$$\frac{1}{k}\sum_{j \in N(i)} \vec{p}_j = \frac{1}{k}\sum_{j \in N(i)} F(\vec{x}_j) - \bar{\vec{x}} \approx 0 \tag{238}$$

that is, the vector set $\{\vec{p}_j\}$ for $j \in N(i)$ is centered, which means that their mean is zero. Expressing this fact in matrix form, leads to:

$$P_i = P_i H \tag{239}$$

where H is the $k \times k$ centering matrix and $P_i = [\vec{p}_{i1}, \vec{p}_{i2}, ...\vec{p}_{ik}]$ is a $D \times k$ matrix in which each column represents a different neighbor. If we assume that $k > d$, then the vector set $\{\vec{p}_j\}$ for $j \in N(i)$ spans the entire tangent space T_i. So, by a singular value decomposition of the matrix P_i, it is possible to build a complete orthonormal basis for the tangent space T_i as:

$$P_i = U \Sigma V^T \tag{240}$$

where the column vectors of the $D \times d$ matrix $U = [\vec{u}_1, \vec{u}_2, ..., \vec{u}_d]$ are the components of such basis, Σ is a $d \times d$ matrix and V^T is a $d \times k$ matrix. Note that by equation (240) we can write the local coordinate matrix of P_i as:

$$\Theta_i = \Sigma V^T = [\alpha_{s,j}] \quad \text{for} \quad s = 1, 2, ..., d \quad \text{and} \quad j = 1, 2, ..., k \tag{241}$$

where each $\vec{p}_j$ can be expressed as a linear combination of the columns of U, that is:

$$\vec{p}_j = \sum_{s=1}^{d} \alpha_{s,j} \vec{u}_s \tag{242}$$

for $j \in N(i)$. In order to properly compute the local coordinates, we have to replace the $\vec{p}_j$'s by the approximations $\vec{x}_j - \bar{\vec{x}}$. First, we define $X_i = [\vec{x}_{i1}, \vec{x}_{i2}, ..., \vec{x}_{ik}]$ and compute the SVD of the centered data matrix $X_i H$ as:

$$X_i H = U_D \Sigma_D V_D^T \tag{243}$$

where $U_D = [\vec{u}_1, ..., \vec{u}_D]$ is an orthonormal matrix, Σ_D is a $D \times D$ diagonal matrix of all singular values of $X_i H$ and $V_D = [\vec{v}_1, ..., \vec{v}_D]$, where each column $\vec{v}_j \in R^k$.

We will now keep only the $d < D$ eigenvectors associated to the largest eigenvalues of the matrix X_iH. Let $U_d = [\vec{u}_1, ..., \vec{u}_d]$, $\Sigma_d = diag(\sigma_1, ..., \sigma_d)$ and $V_d = [\vec{v}_1, ..., \vec{v}_d]$ denote our new reduced matrices. It is worth mentioning that this step is equivalent to applying PCA on the neighborhood $O(i)$ to build the tangent space. Then, we rewrite equation (243) using only the principal components, that is, discarding information from the smallest eigenvalues:

$$X_iH = U_d\Sigma_d V_d^T = U_d\Theta_i \tag{244}$$

Moving forward, in the next step we need to establish a relation between the local coordinate representation of X_i and its global manifold representation. To do so, we will use equation (235) with the set X_i with the following replacements: $\vec{p} = \overline{\vec{x}}$ and $\vec{q} = \overline{\vec{y}}$. Then, we can write:

$$\vec{x}_j - \overline{\vec{x}} = f'(\overline{\vec{y}})(\vec{y}_j - \overline{\vec{y}}) \tag{245}$$

where $\overline{\vec{y}} = h(\overline{\vec{x}})$. Rewriting the equation in terms of the centering matrices leads to:

$$X_iH = f'(\overline{\vec{y}})Y_iH \tag{246}$$

Recall that $f'(\overline{\vec{y}})$ is an invertible mapping. So, let $f'(\overline{\vec{y}})^{-1} = h'(\overline{\vec{x}})$. Then, we can combine equations (244) ad (246) to write:

$$Y_iH = h'(\overline{\vec{x}})X_iH = L\Sigma_d V_d^T \tag{247}$$

where $L = h'(\overline{\vec{x}})U_d$, which finally leads to:

$$L = Y_iHV_d\Sigma_d^{-1} \tag{248}$$

It is worth mentioning that the product between Y_iH and $(I - V_dV_d^T)$ is zero. From equation (248), we have :

$$Y_iH(I - V_dV_d^T) = L\Sigma_d V_d^T - L\Sigma_d V_d^T V_d V_d^T \tag{249}$$

As the columns of the matrix V_d are unit eigenvectors, $V_d^T V_d = I$ and then:

$$Y_iH(I - V_dV_d^T) = L\Sigma_d V_d^T - L\Sigma_d V_d^T = 0 \tag{250}$$

This relationship between the matrices Y_i and V_d is crucial for the next stage of the LTSA algorithm, which is finding the global alignment.

12.3. *Global aligment*

First of all, in order to find the global alignment for the local tangent approximations, we have to define some constraints on the matrix Y. The first assumption is that Y has zero mean and its covariance matrix is the identity, which mathematically translates to $YY^T = I$. Similarly to the previous algorithms, we need these constraints to assure the solution exists and it is unique. In the following, the local relation depicted in equation (250) must be transformed into a global one. Let V_d be the matrix of the singular value decomposition in equation (244). We now define the matrix W_i as:

$$W_i = H(I - V_d V_d^T) \tag{251}$$

We have shown previously by equation (250), that:

$$Y_i W_i = 0 \tag{252}$$

At this point, we will extend the $k \times k$ matrix W_i to a larger $n \times n$ matrix W^i, given by:

$$W^i(j,k) = \begin{cases} W_i(r,s) & \text{if } j = i_r, k = i_s \quad \text{for } i_r, i_s \in N(i) \\ 0 & \text{otherwise} \end{cases} \tag{253}$$

In other words, we are just spreading the d^2 elements of the matrix W_i into the $n \times n$ matrix W^i, but at the same time preserving the adjacency relation defined by the matrix A. Similarly, it is possible to extend the $d \times k$ matrix Y_i to a larger $d \times n$ matrix Y^i such that Y_i is the submatrix of Y^i with the column indices of $N(i)$ and all other columns of Y^i are zero. Once again, by equation (250) we can write:

$$Y^i W^i = 0 \tag{254}$$

Note that since the non-zero columns of W^i have indices in the heighborhood set $N(i)$, the previous equation can be generalized to:

$$YW^i = 0 \tag{255}$$

By definition, the centering matrix H can be expressed as:

$$H = I - \frac{1}{n}\vec{1}\vec{1}^T \tag{256}$$

in a way that left multiplication by a vector of ones, $\vec{1}^T$, leads to:

$$\vec{1}^T H = \vec{1}^T I - \vec{1}^T \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix} = \vec{1}^T - \vec{1}^T = 0 \tag{257}$$

which implies:

$$\vec{1}^T W^i = \vec{1}^T H(I - V_d V_d^T) = 0 \tag{258}$$

Note that the data matrix Y satisfies equation (250) for all $i = 1, 2, ..., n$. So, we define the LTSA kernel as the summation of all matrices W^i, for $i = 1, 2, ..., n$, that is:

$$K = \sum_{i=1}^{n} W^i \tag{259}$$

But invoking equation (255), it is easy to see that:

$$YK = \sum_{i=1}^{n} YW^i = 0 \tag{260}$$

Note also that left multiplication of the kernel matrix by a constant vector of $1/n$'s leads to zero:

$$\frac{1}{n}\vec{1}^T K = \frac{1}{n}\sum_{i=1}^{n} \vec{1}^T W^i = 0 \tag{261}$$

Since the matrix K is symmetric, we can write:

$$K\left[\frac{1}{n}\vec{1} \mid Y^T\right] = 0 \tag{262}$$

where the matrix:

$$\hat{Y} = \left[\frac{1}{n}\vec{1} \mid Y^T\right] \tag{263}$$

is build by the concatenation of the constant vector $(1/n)\vec{1}$ and the $n \times d$ matrix Y^T. Observing equation (263) we note that the columns of the matrix $\hat{Y}$ form an orthonormal basis for the null space of the kernel K. The theoretical implication of this statement is that the intrinsic coordinates can be thought to be eigenvectors of K with zero eigenvalue. Moreover, the LTSA kernel K is a positive semi-definite matrix. It is also clear that W^i is positive semi-definite if and only if W_i is positive semi-definite. Since from equation (239), $P_i = P_i H$, we can write:

$$U\Sigma V^T = U\Sigma V^T H \tag{264}$$

which directly implies that $V^T = V^T H$.

Note also that:

$$W_i = H(I - VV^T) = H - HVV^T = H - HVV^T H \tag{265}$$

As the centering matrix H is idempotent, that is $H = H^2 = HH$, we can write:

$$W_i = HIH - HVV^T H = H(I - VV^T)H \tag{266}$$

To see that W_i is a positive semi-definite matrix, note that VV^T is a projection matrix. Therefore, we know that the eigenvalues of VV^T are constrained in $[0,1]$, which implies that the eigenvalues of $I - VV^T$ are non negative. Since K is the sum of positive semi-definite matrices, it must be positive semi-definite too. Considering the existence of possible approximation errors due to the presence of noise in the observed data, the dimensionality reduction problem can be stated as an optimization one:

$$Y = \arg\min_{Y} \; Tr[YKY^T] \qquad \text{subject to} \qquad YY^T = I \tag{267}$$

Building the Lagrangian function leads to:

$$L(Y, \lambda) = Tr[YKY^T] - \lambda(YY^T - I) \tag{268}$$

Differentiating it and setting the result to zero gives:

$$KY = \lambda Y \tag{269}$$

showing that we select the d eigenvectors associated to the d smallest non-zero eigenvalues of K to compose the matrix Y. Note that, similarly to LLE, since the zero eigenvalue is associated to the constant eigenvector, we should skip it.

Algorithm 8 shows a summary of the LTSA method for dimensionality reduction. The parameters of the algorithm are the input data set $X = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$, the number of neighbors K, and the number of features d.

Algorithm 8 Local Tangent Space Aligment

1: **function** LTSA(X,k,d)
2: From the input data X, build a KNN graph.
3: Let $X_i = [\vec{x}_{i1},...,\vec{x}_{ik}]$ be the matrix whose columns are the k neighbors of $\vec{x}_i$.
4: Compute $\vec{\bar{x}}$, the mean of the neighbors and centralize the data producing $\hat{X}_i = [\vec{x}_{i1} - \vec{\bar{x}},...,\vec{x}_{ik} - \vec{\bar{x}}]$.
5: Apply PCA in $\hat{X}_i$ to find the local coordinates (tangent space). Let $V_i = [\vec{v}_1,\vec{v}_2,...,\vec{v}_d]$ be the d eigenvectors associated to the d largest eigenvalues of the covariance matrix of $\hat{X}_i$. Set:

$$G_i = \left[\frac{1}{k}\vec{1} \mid V_i\right] \tag{270}$$

and

$$W_i = I - G_i G_i^T \tag{271}$$

6: Build the LTSA Kernel matrix K: initialize K with the zero matrix of dimensions $n \times n$. Let $N(i)$ be the index set of the neighborhood of $\vec{x}_i$ and $K(N(i),N(i))$ be the submatrix of K containing the rows and columns of the indices $N(i)$. Then, update K by setting:

$$K(N(i),N(i)) = K(N(i),N(i)) + W_i \tag{272}$$

7: Perform an eigendecomposition of the kernel matrix K and let $Y = [\vec{u}_1,...,\vec{u}_d]^T$ be the eigenvectors associated to the smallest non-zero eigenvalues. Then, each column of Y has the coordinates of a given point in the manifold.
8: **return** Y
9: **end function**

A linear version of LTSA was proposed later to allow direct mapping of test samples efficiently in a LPP fashion. The interested reader is referred to the work of Zhang and colleagues for more details [98].

13. Diffusion Maps

Diffusion maps is a nonlinear dimensionality reduction algorithm originally proposed by Coifman and Lafon that learns a new metric called diffusion distance as a measure of similarity between data points [20,21]. It has been shown in Laplacian Eigenmaps that the Laplace-Beltrami operator is directly related to the heat diffusion model. This intrinsic relationship indicates that a diffusion kernel in the manifold itself can be used to perform dimensionality reduction [85]. Consider a set $X = \{\vec{x}_1,\vec{x}_2,...,\vec{x}_n\}$ of observations where $\vec{x}_i \in R^m$, for $i = 1,2,...,n$. Basically, the diffusion maps framework consists of the following steps [74]:

(1) construction of a kernel matrix K based on pairwise similarity measures between sam-

ples (this is equivalent to build a complete weighted graph where the vertices are the samples);
(2) definition of a Markov process on the graph G via the construction of a transition probability matrix P (to model the diffusion process);
(3) nonlinear mapping of the samples into a new embedded space based on a parametrization of the graph, which forms an intrinsic representation of the dataset;

In order to characterize diffusion processess on graphs, we first need to introduce Markov Chains, the mathematical model governing random walks [33].

13.1. *Markov Chains*

Let $G = (V,E)$ be a connected graph with $|V| = n$ and $|E| = m$. Informally, a random walk on a graph is a stochastic process in which, given an initial vertex v_0, we select a random neighbor v_1 and move forward, repeting the process for every subsequent visited vertex, until we reach a maximum number of steps. A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event [33].

A discrete time Markov Chain defined on a finite discrete state space can be represented by a transition probability matrix.

Definition 9. (Transition probability matrix) Let $G = (V,E)$ be a connected graph with weighted adjacency matrix W. The matrix defined by $P = D^{-1}W$ is the transition probability matrix (or Markov matrix) of the graph, where $D = diag(d(v_1),...,d(v_n))$ is the diagonal matrix of the degrees $d(v_i) = \sum_{j=1}^{n} w_{ij}$, for $i = 1,2,...,n$.

A discrete time Markov Chain defined on a finite discrete state space can be represented by a transition probability matrix P (Markov matrix). Due to the Markovian property, the joint probability for a sequence $v_0, v_1, v_2, ..., v_n$ is simplified. Note that:

$$\begin{aligned} P(v_o, v_1, ..., v_n) &= \prod_{i=0}^{n} P(v_i \mid v_0, v_1, ..., v_{i-1}) \\ &= P(v_0)P(v_1 \mid v_0)P(v_2 \mid v_0, v_1)...P(v_n \mid v_0, v_1, ..., v_{n-1}) \end{aligned} \tag{273}$$

If the process is Markovian, that is, memoryless, we have:

$$P(v_n \mid v_0, v_1, ..., v_{n-1}) = P(v_n \mid v_{n-1}) \tag{274}$$

and all conditional probability pair can be arranged into the Markov matrix.

Let $\vec{w}_0 \in R^n$ be the initial probability distribution of a random walk on the graph. For example, let's suppose that we want to start our walk from the vertices v_0 or v_1, with equal probabilities. Then, our $\vec{w}_0$ vector is given by:

$$\vec{w}_0 = \left[\frac{1}{2}, \frac{1}{2}, 0, 0, ..., 0\right] \tag{275}$$

It has been shown that our probability distribution vector can be updated by a diffusion process given by the Chapman-Kolmogorov equations, which in the discrete case are given by a matrix-vector notation:

$$\vec{w}_k = \vec{w}_{k-1} P \tag{276}$$

for $k > 0$, where P is the Markov matrix. In the following, we present four important definitions concerning properties of a Markov Chain.

Definition 10. (Homogeneous Markov Chain) A Markov chain is homogenous if the Markov matrix does not change over time, that is, it is time invariant.

Definition 11. (Irredutible Markov Chain) A Markov chain is irredutible if it is possible to visit any state i from any other state j.

Definition 12. (Aperiodic Markov Chain) A Markov chain is aperiodic if there exists k for which P^k has only non-zero elements.

Definition 13. (Ergodic Markov Chain) A Markov chain is ergodic if it is irreducible and aperiodic.

It can be shown that random walks on undirected graphs are modeled by ergoodic Markov Chains, since we have the following equivalences:

- P is irreducible if and only if G is connected.
- P is aperiodic if and only if G is not bipartite.

The following theorem states that in ergodic Markov Chains, there is a unique steady-state which does not depend on the inicial probability distribution $\vec{w}_0$.

Theorem 5. *Let P be the transition probability matrix of a ergodic homogeneous Markov Chain. Then, the stationary distribution $\vec{w}^*$ exists and it is unique, regardless of the inicial $\vec{w}_0$:*

$$lim_{k\to\infty} P^k = W \tag{277}$$

where

$$W = \begin{bmatrix} \vec{w}^* \\ ... \\ \vec{w}^* \end{bmatrix} \tag{278}$$

If an ergodic Markov Chain is reversible, than it satisfies the balance condition:

$$w_i p_{ij} = w_j p_{ji} \tag{279}$$

for all $i, j \in V$, where V is the vertex set of the graph. It has been shown that if we manage to find a distribution $\vec{w}$ which satisfies the balance condition, then $\vec{w}$ is the steady-state of the Markov Chain, since:

$$w_i = w_i \sum_{j=1}^{n} p_{ij} = \sum_{j=1}^{n} w_i p_{ij} = \sum_{j=1}^{n} w_j p_{ji} \tag{280}$$

that is:

$$\vec{w} = \vec{w}P \tag{281}$$

meaning that the distribution does not change in time. In case of an ergodic Markov Chain, it is possible to compute the stationary distribution analitycally. Note that from the balance condition, and knowing that $p_{ij} = 1/d(v_i)$, we have:

$$\frac{w_i}{d(v_i)} = \frac{w_j}{d(v_j)} = K \tag{282}$$

where K is an arbitrary constant. Hence, we can write:

$$w_i = Kd(v_i) \tag{283}$$

Summing for all vertices of the graph leads to:

$$\sum_{i=1}^{n} w_i = K \sum_{i=1}^{n} d(v_i) \tag{284}$$

Since the left hand size is equal to one, we solve for K:

$$K = \frac{1}{\sum_{i=1}^{n} d(v_i)} \tag{285}$$

which finally leads to the solution being sought:

$$w_i^* = \frac{d(v_i)}{\sum_{i=1}^{n} d(v_i)} \tag{286}$$

for $i = 1, 2, ..., n$.

There is a direct relation between the Markov matrix P and the normalized Laplacian matrix $\overline{L} = D^{-1}L$ of the graph. It is straightforward to see that:

$$\overline{L} = D^{-1}L = D^{-1}(D - W) = I - D^{-1}W = I - P \tag{287}$$

Note that both $\overline{L}$ and P share the same eigenvalues and eigenvectors, however the smallest eigenvectors of $\overline{L}$ are the largest eigenvalues of P:

$$\begin{gathered} \overline{L}\vec{v} = \lambda\vec{v} \quad \rightarrow \quad (D^{-1}W)\vec{v} = \lambda\vec{v} \quad \rightarrow \quad (I-P)\vec{v} = \lambda\vec{v} \quad \rightarrow \\ \rightarrow \quad \vec{v} - P\vec{v} = \lambda\vec{v} \quad \rightarrow \quad P\vec{v} = (1-\lambda)\vec{v} \end{gathered} \tag{288}$$

since $0 \leq \lambda \leq 1$. It means that while the constant eigenvector $\vec{1}$ is associated to the zero eigenvalue in the normalized Laplacian $\overline{L}$, it is associated to the unit eigenvalue in the Markov matrix P.

13.2. *The kernel matrix*

Unlike most manifold learning algorithms, Diffusion Maps doesn't require a KNN graph. Usually, the first step consists in building a complete graph $G = (V, E)$ from the data points. Let $k_\sigma(\vec{x}_i, \vec{x}_j)$ be a kernel function representing a notion of pairwise similarity between samples, with a scale parameter σ. For example, we have the popular Gaussian kernel:

$$k_\sigma(\vec{x}_i, \vec{x}_j) = exp\left(-\frac{||\vec{x}_i - \vec{x}_j||^2}{\sigma}\right) \tag{289}$$

A kernel function mus t satisfy the following properties:

- symmetry: $k_\sigma(\vec{x}_i, \vec{x}_j) = k_\sigma(\vec{x}_j, \vec{x}_i)$
- nonnegativity: $k_\sigma(\vec{x}_i, \vec{x}_j) \geq 0$
- locality: given a scale parameter $\sigma > 0$, $k_\sigma(\vec{x}_i, \vec{x}_j) \to 1$ for $||\vec{x}_i - \vec{x}_j|| \ll \sigma$ and $k_\sigma(\vec{x}_i, \vec{x}_j) \to 0$ for $||\vec{x}_i - \vec{x}_j|| \gg \sigma$

According to several studies, setting the scale parameter properly is crucial for the success of the manifold learning task [38]. If the scale parameter is too small, it may result in a poorly connected graph. On the other hand, a too large scale parameter makes the kernel insensitive to variations in the distances. In practice, it is usual to define the scale parameter in terms of the standard deviation of the data. Based on the kernel with a locality property, a weighted graph G is created in a way that a neighborhood of radius σ around each sample $\vec{x}_i$ is defined. In many applications, the Euclidean distance in the kernel can be replaced by any other metric.

Note that the kernel matrix K can be seen as an adjacency matrix for the graph G, which is symmetric. Let $P = D^{-1}K$ be the Markov matrix, where D is the diagonal matrix of the degrees $d(\vec{x}_i)$, defined as the sum of the elements of the i-th row of K. In this context, the Markov matrix P is also known as the diffusion matrix. Propagating the Markov Chain k steps forward corresponds to raising the diffusion matrix to the power of k, that is, to computing P^k. We denote the probability to go from $\vec{x}_i$ to $\vec{x}_j$ in k steps as $p_k(\vec{x}_i, \vec{x}_j)$.

13.3. *Diffusion distance*

By applying an eigendecomposition in the diffusion matrix P^k we can decompose it on a set of left and right eigenvectors $\{\Phi, \Psi\}$, where $\Phi = [\vec{\phi}_1, ..., \vec{\phi}_n]$ and $\Psi = [\vec{\psi}_1, ..., \vec{\psi}_n]$, and a set of eigenvalues $\lambda_0 \geq \lambda_1 \geq ... \geq \lambda_{n-1}$. Hence, the diffusion mapping of the samples into an Euclidean R^d space is defined by:

$$\vec{DM}_k(\vec{x}_i) = \left[\lambda_1^k \psi_1(i), ..., \lambda_d^k \psi_d(i)\right] \tag{290}$$

where d is the new space dimensionality (ranging from 1 to $m-1$).

By left and right eigenvectors of P, we mean vectors satisfying:

$$\vec{\phi}_j^T P = \lambda_j \vec{\phi}_j^T \tag{291}$$

$$P\vec{\psi}_j = \lambda_j \vec{\psi}_j \tag{292}$$

for $j = 1, 2, ..., n$. It can be verified that the largest eigenvalue of P is one, that is, $\lambda_0 = 1$, and since the rows of P sum to one:

$$P\vec{1} = \vec{1} \tag{293}$$

showing that $\vec{\psi}_0 = \vec{1}$. Note that $\vec{\phi}_0$ is the stationary distribution of the Markov Chain, since:

$$\vec{\phi}_0^T = \vec{\phi}_0^T P \tag{294}$$

Besides, it can also be verified that $\vec{\phi}_k^T \vec{\psi}_l = \delta_{kl}$, where $\delta_{kl} = 1$ if $k = l$ and $\delta_{kl} = 0$ if $k \neq l$. Note that there is two ways to compute $\vec{\phi}_k^T P \vec{\psi}_l$ as:

$$(\vec{\phi}_k^T P)\vec{\psi}_l = \lambda_k \vec{\phi}_k^T \vec{\psi}_l \tag{295}$$

$$\vec{\phi}_k^T (P\vec{\psi}_l) = \lambda_l \vec{\phi}_k^T \vec{\psi}_l \tag{296}$$

which means that

$$\lambda_k \vec{\phi}_k^T \vec{\psi}_l - \lambda_l \vec{\phi}_k^T \vec{\psi}_l = (\lambda_k - \lambda_l)\vec{\phi}_k^T \vec{\psi}_l = 0 \tag{297}$$

and for $k \neq l$ we have $\lambda_k - \lambda_l \neq 0$ since the eigenvalues are assumed distinct. Hence, $\vec{\phi}_k^T \vec{\psi}_l = 0$, that is, the vectors are orthogonal. Moreover, it has been shown that the set of left eigenvectors of P and the set right eigenvectors of P are related by [48]:

$$\psi_l(\vec{x}_i) = \frac{\phi_l(\vec{x}_i)}{\phi_0(\vec{x}_i)} \tag{298}$$

for $i = 1, 2, ..., n$.

We can normalize the left eigenvectors of P with respect with $1/\phi_0$:

$$||\vec{\phi}_l||^2_{1/\phi_0} = \sum_{i=1}^{n} \frac{1}{\phi_0(\vec{x}_i)} \phi_l(\vec{x}_i)^2 = \sum_{i=1}^{n} \frac{\phi_l(\vec{x}_i)}{\phi_0(\vec{x}_i)} \phi_l(\vec{x}_i) = \sum_{i=1}^{n} \psi_l(\vec{x}_i)\phi_l(\vec{x}_i) = 1 \tag{299}$$

Similarly, we can normalize the right eigenvectors with respect to ϕ_0:

$$||\vec{\psi}_l||^2_{\phi_0} = \sum_{i=1}^{n} \psi_l(\vec{x}_i)^2 \phi_0(\vec{x}_i) = \sum_{i=1}^{n} \frac{\phi_l(\vec{x}_i)^2}{\phi_0(\vec{x}_i)^2} \phi_0(\vec{x}_i) = \sum_{i=1}^{n} \frac{1}{\phi_0(\vec{x}_i)} \phi_l(\vec{x}_i)^2 = 1 \tag{300}$$

The Markov process enables the integration of information from the entire set into the affinity metric $p_k(\vec{x}_i,\vec{x}_j)$ between two individual samples, so we can define a new similarity measure between samples, known as the diffusion distance [48]:

$$D_k^2(\vec{x}_i,\vec{x}_j) = ||p_k(\vec{x}_i,.) - p_k(\vec{x}_j,.)||_{1/\phi_0}^2 = \sum_{r=1}^{n} \frac{(p_k(\vec{x}_i,\vec{x}_r) - p_k(\vec{x}_j,\vec{x}_r))^2}{\phi_0(\vec{x}_r)} \tag{301}$$

where the weights $1/\phi_0$ penalize discrepancies on domains of low density more than those of high density. As $p_k(\vec{x}_i,\vec{x}_j)$ is the kernel of the k-th iterate P^k, we will have the following biorthogonal spectral decomposition:

$$p_k(\vec{x}_i,\vec{x}_j) = \sum_{s=1}^{n} \lambda_s^k \psi_s(\vec{x}_i)\phi_s(\vec{x}_j) \tag{302}$$

The first d terms of the summation provide the best rank-d approximation of P^k, according to the following weighted metric for matrices:

$$||P||^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} \phi_0(\vec{x}_i)p(\vec{x}_i,\vec{x}_j)^2 \frac{1}{\phi_0(\vec{x}_j)} \tag{303}$$

which can be interpreted as a weighted PCA of the matrix P.

Now, plugging equation (302) into equation (301), we have:

$$\begin{aligned} D_k^2(\vec{x}_i,\vec{x}_j) &= \sum_{r=1}^{n} \left\{ \frac{1}{\phi_0(\vec{x}_r)} \left[\sum_{s=0}^{n-1} \lambda_s^k \psi_s(\vec{x}_i)\phi_s(\vec{x}_r) - \sum_{s=0}^{n-1} \lambda_s^k \psi_s(\vec{x}_j)\phi_s(\vec{x}_r) \right]^2 \right\} \\ &= \sum_{r=1}^{n} \left\{ \frac{\left[\sum_{s=0}^{n-1} \lambda_s^k \left(\psi_s(\vec{x}_i) - \psi_s(\vec{x}_j)\right)\phi_s(\vec{x}_r)\right]^2}{\phi_0(\vec{x}_r)} \right\} \\ &= \sum_{r=1}^{n} \left\{ \frac{\sum_{s=0}^{n-1}\sum_{t=0}^{n-1} \lambda_s^k\lambda_t^k \left(\psi_s(\vec{x}_i) - \psi_s(\vec{x}_j)\right)\left(\psi_t(\vec{x}_i) - \psi_t(\vec{x}_j)\right)\phi_s(\vec{x}_r)\phi_t(\vec{x}_r)}{\phi_0(\vec{x}_r)} \right\} \\ &= \sum_{s=0}^{n-1}\sum_{t=0}^{n-1} \lambda_s^k\lambda_t^k \left(\psi_s(\vec{x}_i) - \psi_s(\vec{x}_j)\right)\left(\psi_t(\vec{x}_i) - \psi_t(\vec{x}_j)\right) \sum_{r=1}^{n} \frac{1}{\phi_0(\vec{x}_r)}\phi_s(\vec{x}_r)\phi_t(\vec{x}_r) \end{aligned} \tag{304}$$

However, note that:

$$\sum_{r=1}^{n} \frac{1}{\phi_0(\vec{x}_r)}\phi_s(\vec{x}_r)\phi_t(\vec{x}_r) = \frac{1}{\sqrt{\phi_0(\vec{x}_r)}}\vec{\phi}_s^T \frac{1}{\sqrt{\phi_0(\vec{x}_r)}}\vec{\phi}_t = \begin{cases} 1 & \text{if } s = t \\ 0 & \text{if } s \neq t \end{cases} \tag{305}$$

Therefore, we can simplify the equation for $D_k^2(\vec{x}_i,\vec{x}_j)$ to:

$$\begin{aligned} D_k^2(\vec{x}_i,\vec{x}_j) &= \sum_{s=0}^{n-1} (\lambda_s^k)^2 \left(\psi_s(\vec{x}_i) - \psi_s(\vec{x}_j)\right)^2 \\ &= ||\vec{DM}_k(\vec{x}_i) - \vec{DM}_k(\vec{x}_j)||^2 \end{aligned} \tag{306}$$

which is the Euclidean distance between the diffusion maps. Usually, the diffusion distance can be well approximated by using the first d non-trivial eigenvalues. The mapping from the graph to the R^d provides a parametrization of the dataset in a way that the Euclidian distance in R^d is an approximation to the diffusion distance. That is one of the most important features of Diffusion Maps: unlike other manifold learning algorithms, the starting point is an explicitly defined distance metric [48]. Algorithm 9 describes the main steps in Diffusion Maps.

Algorithm 9 Diffusion Maps

1: **function** DIFFUSIONMAPS(X, k_n, k_t, d)
2: From X, build a KNN or ε-neighborhood graph $G = (V,E)$ where $|V| = n$.
3: Create the weight matrix W by:

$$w_{ij} = \begin{cases} exp\left(-\frac{||\vec{x}_i - \vec{x}_j||^2}{\sigma}\right) & \text{for } j \in N(i) \cup \{i\} \\ 0, \quad \text{otherwise} \end{cases} \tag{307}$$

where the σ parameter is estimated by:

$$\sigma = \frac{1}{N^*} \sum_{i=1}^{n} \sum_{j \in N(i)} ||\vec{x}_i - \vec{x}_j||^2 \tag{308}$$

with

$$N^* = \sum_{i=1}^{n} |N(i)| \tag{309}$$

4: Compute the stochastic Markov matrix $P = D^{-1}W$.
5: Apply the diffusion process, that is, compute:

$$P^{k_t} = P \times P \times ... \times P \qquad (k_t \text{ times}) \tag{310}$$

6: Compute the eigenvalues and right eigenvectors of the diffusion matrix P^{k_t}
7: Map to the d dimensionl diffusion space, using the d largest non-trivial eigenvalues and eigenvectors, that is, define $\vec{y}_i$ as:

$$\vec{y}_i = \left[\lambda_1^k \psi_1(i), ..., \lambda_d^k \psi_d(i)\right] \tag{311}$$

where λ_j denotes the j-th eigenvalue and $\psi_j(i)$ denotes the i-th coefficient of the j-th eigenvector
8: **return** $Y = [\vec{y}_1, \vec{y}_2, ..., \vec{y}_n]$
9: **end function**

Diffusion maps has four main parameters: σ for the kernel, k_n for the neighborhood of the induced graph, k_t for the diffusion process and d for the dimensionality reduction. It should be noted that as k increases and the diffusion process is happening, the dacay rate of the singular values are also increased [19]. As a result, we may set d to be smaller,

capturing the underlying structure of the samples in less dimensions. Note also that Diffusion Maps is reduced to Laplacian Eigenmaps when the eigenvalues are discarded from the mapping. It means that Laplacian Eigenmaps carry information on the underlying model and the temporal dynamics without the information revealed by the Markov process [74]. A diffusion process reveal the global geometric structure of a dataset, since paths along the true geometric structure have high probability.

Note that the neighborhood definition for Diffusion Maps can also be performed by thresholding the weights appropriately. If the weight threshohld method is chosen, then the first two steps of the algorithm are merged into a single one: let α be a weight threshold and $g_{ij} = K(\vec{x}_i, \vec{x}_j)$ for $i, j = 1, 2, ..., n$. The weight matrix $W = \{w_{ij}\}$ is [85]:

$$w_{ij} = \begin{cases} g_{ij}, & \text{if } g_{ij} \leq \alpha \\ 0, & \text{if } g_{ij} > \alpha \end{cases} \tag{312}$$

14. Conclusions and Final Remarks

Dimensionality reduction methods are powerful mathematical tools for data analysis and visualization, since the presence of multivariate data is ubiquitous in pattern recognition and machine learning applications. In this paper, we presented a review of the most important linear and non-linear dimensionality reduction methods for unsupervised metric learning, from PCA to manifold learning algorithms. As each method has its advantages and limitations, several challenges and open problems remain unsolved in this field. Even though manifold learning has become an increasingly popular area of study and has shown to be relevant in several pattern recognition applications, much is still unknown. For instance, evaluating these algorithms is a difficult problem. The usual methodology consists in running the algorithm in some artificial data set and visually analyse whether the results are intuitively suitable, which can be totally subjective. Objective measures are needed to allow quantitative analysis. To perform a robust experimental evaluation, one requires knowledge of the underlying manifolds behind the data (ground-truth) and the determination of whether the manifold learning algorithm is preserving this information somehow, which is not always possible in the current framework of dimensionality reduction. In a more theoretical perspective, some relevant open questions arised by the community include: how to choose the optimal number of neighbors in the KNN graph? Are there better ways to build a graph from the data points in X, instead of KNN and ε-neighborhood? How can we efficiently make these algorithms supervised? How can we estimate the intrinsic dimensionality of the data in a meaningful way? Is the assumption of a manifold structure reasonable? If so, how to measure if these algorithms are successful at uncovering this structure? It is not clear how to properly answer these questions, but the pattern recognition community seems to agree that we still need stronger theoretical guarantees and also more results related to learning in the finite-sample case.

Recently, much has been said about how deep learning is considered by many practioners and researchers as the state-of-the-art in learning features from high dimensional datasets, such as images databases. Deep learning is a class of machine learning algorithms

that uses multiple layers to progressively extract higher level features from the raw input. Despite evident differences in computational cost, since deep learning algorithms often require robust hardware in order to train neural networks with millions of parameters to be adjusted, one key aspect that shows the disparity between deep learning and manifold learning is that, while the former usually require a huge amount of data to achieve good results, the later is suitable for reduced size sample problems or in cases when the number of features is equal or greater than the number of samples. Hence, it does not seem reasonable to assume that, eventually, deep learning algorithms will replace manifold learning methods. There is room for both approaches in feature extraction and dimensionality reduction, depending on constraints in the number of available training samples. Moreover, most deep learning models work with the supervised learning paradigm, since they are generalizations of multilayer perceptrons, meaning that we need information about the class labels, which is not always possible.

Finally, we would like to point out some ideas on how to extend the dimensionality reduction algorithms for the purpose of metric learning. One line of research that seems promissing is the study of how information-theoretic measures, such as KL-divergence, Bhattacharyya and Hellinger distances, can be applied to improve unsupervised metric learning, by creating parametric versions of the usual methods. The similarity between two points in the KNN graph can be measured by stochastic divergences between the patches defined by the point itself and the set of its neighbors. Different statistical models can be considered instead of the usual Gaussian distribution. For instance, Gaussian-Markov random field models are able to capture the spatial dependence between the data points through the definition of an additional coupling parameter, known as inverse temperature. Other models include Gaussian Mixture Models (GMM's) and Generalized Gaussians. Another possibility is to use the Fisher information matrix, which defines the metric tensor of the underlying parametric space, to approximate the geodesic distance beween the distributions of different neighboring patches along the KNN graph.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Bibliography

1. S. Amari, A. Cichocki and H. Yang, A new learning algorithm for blind source separation, *Advances in Neural Information Processing Systems* **8** (1996) 757–763.
2. M. Andersen, J. Dahl and L. Vandenberghe, Cvxopt: a free python software package for convex optimization (2019), http://cvxopt.org/.
3. M. Balasubramanian and E. L. Schwartz, The isomap algorithm and topological stability, *Science* **295** (2002) 7–8.
4. M. Belkin and P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in T. G. Dietterich, S. Becker and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems 14* (MIT Press, 2002) pp. 585–591.
5. M. Belkin and P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, *Neural Computation* **15** (June 2003) 1373–1396.

6. M. Belkin and P. Niyogi, Convergence of laplacian eigenmaps, in B. Schölkopf, J. C. Platt and T. Hoffman (eds.), *Advances in Neural Information Processing Systems 19* (MIT Press, 2007) pp. 129–136.
7. A. J. Bell and T. J. Sejnowski, An information maximization approach to blind separation and blind deconvolution, *Neural Computation* **7** (1995) 1129–1159.
8. A. Bellet, A. Habrard and M. Sebban, A survey on metric learning for feature vectors and structured data, *CoRR* **abs/1306.6709** (2013).
9. R. Bellman, *Adaptive Control Processes: A Guided Tour* (Princeton University Press, 1961).
10. M. Bernstein, V. de Silva, J. Langford and J. Tenenbaum, Graph approximations to geodesics on embedded manifolds (2000).
11. L. Bo, L. Yan-Rui and Z. Xiao-Long, A survey on laplacian eigenmaps based manifold learning methods, *Neurocomputing* **335** (2018) 336–351.
12. I. Borg and P. Groenen, *Modern Multidimensional Scaling: theory and applications*, 2 edn. (Springer-Verlag, 2005).
13. M. Brand, *Charting a manifold* in S. Becker, S. Thrun and K. Obermayer (eds.), *Advances in Neural Information Processing Systems*. (MIT Press, 2003), pp. 961–968.
14. A. E. Brouwer and W. H. Haemers (eds.), *Spectra of Graphs* (Springer, 2011).
15. M. A. Carreira-Perpinan, A Review of Dimension Reduction Techniques, tech. rep., University of Sheffield (jan 1997).
16. L. Cayton, Algorithms for Manifold Learning, tech. rep., University of California San Diego (UCSD) (2005).
17. H. Choi and S. Choi, Robust kernel isomap, *Pattern Recognition* **40**(3) (2007) 853–862.
18. V. Choulakian, L1-norm projection pursuit principal component analysis, *Computational Statistics and Data Analysis* **50**(6) (2006) 1441–1451.
19. F. R. K. Chung, *Spectral Graph Theory* (American Mathematical Society, 1997).
20. R. Coifman and S. Lafon, Diffusion maps, *Applied and Computational Harmonic Analysis* **21** (2006) 5–30.
21. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner and S. W. Zucker, Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps, *Proceedings of the National Academy of Sciences (PNAS)* **102**(21) (2005) 7426–7431.
22. J. A. Cook, I. Sutskever, A. Mnih and G. Hinton, Visualizing similarity data with a mixture of maps, in *Proceedings of the 11 th International Conference on Artificial Intelligence and Statistics*, Vol. 2 (2007) pp. 67–74.
23. T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 3 edn. (MIT Press, 2009).
24. T. F. Cox and M. A. A. Cox, *Multidimensional Scaling*, Monographs on Statistics and Applied Probability, Vol. 88 (Chapman & Hall, 2001).
25. J. P. Cunningham and Z. Ghahramani, Linear dimensionality reduction: Survey, insights, and generalizations, *Journal of Machine Learning Research* **16** (2015) 2859–2900.
26. D. de Ridder and R. P. Duin, Locally linear embedding for classification, tech. rep., Delft University of Technology (2002).
27. V. de Silva and J. B. Tenenbaum, Global versus local methods in nonlinear dimensionality reduction, in *Proceedings of the 15th International Conference on Neural Information Processing Systems (NIPS'02)* (2002) pp. 721–728.
28. D. L. Donoho and C. Grimes, Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data, *Proceedings of the National Academy of Sciences* **100**(10) (2003) 5591–5596.
29. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification*, 2 edn. (Wiley-Interscience, 2000).
30. F. Errica, Step-by-step derivation of sne and t-sne gradients (2018),

http://pages.di.unipi.it/errica/curious/derivations-sne-tsne.
31. M. Fiedler, Laplacian of graphs and algebraic connectivity, *Banach Center Publications* **25**(1) (1989) 57–70.
32. K. Fukunaga, *Introduction to Statistical Pattern Recognition* (Academic Press, 1990).
33. P. A. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation* (John Wiley & Sons, 2017).
34. J. S. Galpin and D. M. Hawkins, Methods of l1 estimation of a covariance matrix, *Computational Statistics and Data Analysis* **5** (1987) 305–319.
35. J. Ham, D. D. Lee, S. Mika and B. Schölkopf, A kernel view of the dimensionality reduction of manifolds, in *Proceedings of the Twenty-first International Conference on Machine Learning*, *ICML '04* (ACM, New York, NY, USA, 2004) pp. 47–54.
36. T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2 edn. (Springer, 2009).
37. X. He and P. Niyogi, Locality preserving projections, in S. Thrun, L. K. Saul and B. Schölkopf (eds.), *Advances in Neural Information Processing Systems 16* (MIT Press, 2004) pp. 153–160.
38. M. Hein and Y. Audibert, Intrinsic dimensionality estimation of submanifold in r^d, in *In Proceedings of the International Conference on Machine Learning (ICML)* (2005) pp. 289–296.
39. G. E. Hinton and S. T. Roweis, Stochastic neighbor embedding, in S. Becker, S. Thrun and K. Obermayer (eds.), *Advances in Neural Information Processing Systems 15* (MIT Press, 2003) pp. 857–864.
40. G. F. Hughes, On the mean accuracy of statistical pattern recognizers, *IEEE Trans. on Information Theory* **14** (1968) 55–63.
41. X. Huo, X. Ni and A. K. Smith, *A Survey of Manifold-Based Learning Methods* in T. W. Liao and E. Triantaphyllou (eds.), *Recent Advances in Data Mining of Enterprise Data: Algorithms and Applications*, Series on Computers and Operations Research, Vol. 6. (World Scientific, 2008), pp. 691–745.
42. J. Hwang, S. Lay and A. Lippman, Nonparametric multivariate density estimation: A comparative study, *IEEE Trans. on Signal Processing* **42**(10) (1994) 2795–2810.
43. A. Hyvaninen, Fast and robust fixed-point algorithms for independent component analysis, *IEEE Transactions on Neural Networks* **10**(3) (1999) 626–634.
44. A. Hyvarinen, J. Karhunen and E. Oja, *Independent Component Analysis* (John Wiley & Sons, 2001).
45. L. O. Jimenes and D. Landgrebe, Supervised classification in high dimensional space: Geometrical, statistical and asymptotical properties of multivariate data, *IEEE Trans. on Systems, Man and Cybernetics* **28**(1) (1998) 39–54.
46. I. T. Jolliffe, *Principal Component Analysis*, 2 edn. (Springer, 2002).
47. M. Journee, Y. Nesterov, P. Richtarik and R. Sepulchre, Generalized power method for sparse principal component analysis, *Journal of Machine Learning Research* **11** (2010) 517–553.
48. S. Lafon and A. B. Lee, Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization, *IEEE Trans. On Pattern Analysis and Machine Intelligence* **29**(9) (2006) 1393–1403.
49. J. A. Lee and M. Verleysen, Unsupervised dimensionality reduction: Overview and recent advances, in *The 2010 International Joint Conference on Neural Networks (IJCNN)* (July 2010) pp. 1–8.
50. J. A. Lee and M. Verleysen, *Nonlinear Dimensionality Reduction* (Springer, 2007).
51. K. Lee and H. Park, Probabilistic learning of similarity measures for tensor pca, *Pattern Recognition Letters* **33**(10) (2012) 1364–1372.
52. T. W. Lee, M. Girolami and T. J. Sejnowski, Independent component analysis using an extended infomax algorithm for mixed subgaussian and supergaussian sources, *Neural Computation* **11**(2) (1999) 626–634.

53. D. Li and Y. Tian, Survey and experimental study on metric learning methods, *Neural Networks* **105** (2018) 447–462.
54. J. McClurkin, B. L.M. Optican and T. Gawne, Concurrent processing and complexity of temporally encoded neuronal messages in visual perception, *Science* **253** (1991) 675–677.
55. J. Melville, sneer: Stochastic neighbor embedding experiments in r (2015), http://jlmelville.github.io/sneer/gradients.html.
56. B. Mohar, The laplacian spectrum of graphs, in *Graph Theory, Combinatorics, and Applications* (Wiley, 1991) pp. 871–898.
57. H. Murase and S. Nayar, Visual learning and recognition of 3d objects from appearance, *International Journal Computer Vision* **14** (1995) 5–24.
58. B. Nica, *A Brief Introduction to Spectral Graph Theory* (American Mathematical Society, 2018).
59. D.-T. Pham, Fast algorithms for mutual information based independent component analysis, *IEEE Transactions on Signal Processing* **52**(10) (2004) 2690–2700.
60. S. T. Roweis, Em algorithms for pca and sensible pca, in *Advances in Neural Information Processing Systems* (1997)
61. S. Roweis and L. Saul, Nonlinear dimensionality reduction by locally linear embedding, *Science* **290** (2000) 2323–2326.
62. L. Saul and S. Roweis, Think globally, fit locally: Unsupervised learning of low dimensional manifolds, *Journal of Machine Learning Research* **4** (2003) 119–155.
63. L. K. Saul and S. T. Roweis, An introduction to locally linear embedding, tech. rep., New York University (2000).
64. A. Saxena, A. Gupta and A. Mukerjee, Non-linear dimensionality reduction by locally linear isomaps, *Lecture Notes in Computer Science* **3316** (2004) 1038–1043.
65. B. Schölkopf, A. Smola and K. R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, *Neural Computation* **10**(5) (1998) 1299–1319.
66. B. Schölkopf, A. Smola and K. R. Müller, Kernel principal component analysis, in *Advances in Kernel Methods – Support Vector Learning* (MIT Press, 1999) pp. 327–352.
67. D. W. Scott, *Multivariate Density Estimation* (John Wiley & Sons, 1992).
68. H. S. Seung and D. D. Lee, The manifold ways of perception, *Science* **290** (2000) 2268–2269.
69. C. Shalizi, Nonlinear dimensionality reduction i: Local linear embedding (2009), http://www.stat.cmu.edu/ cshalizi/350/lectures/14/lecture-14.pdf.
70. B. W. Silverman, *Density Estimation for Statistics and Data Analysis* (Chapman & Hall, 1986).
71. C. D. Sousa, Pylmi-sdp: Symbolic linear matrix inequalities (lmi) and semi-definite programming (sdp) tools for python (2014), https://github.com/cdsousa/PyLMI-SDP.
72. D. A. Spielman, Spectral graph theory and its applications, in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)* (Oct 2007) pp. 29–38.
73. J.-L. Suárez, S. García and F. Herrera, A tutorial on distance metric learning: Mathematical foundations, algorithms and software, *CoRR* **abs/1812.05944** (2018).
74. R. Talmon, I. Cohen, S. Gannot and R. R. Coifman, Diffusion maps for signal processing: A deeper look at manifold-learning techniques based on kernels and graphs, *IEEE Signal Processing Magazine* **30**(4) (2013) 75–86.
75. J. B. Tenenbaum, V. de Silva and J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, *Science* **290** (2000) 2319–2323.
76. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4 edn. (Academic Press, 2008).
77. M. E. Tipping and C. M. Bishop, Probabilistic principal component analysis, *Journal of the Royal Statistical Society, Series B* **61**(3) (1999) 611–622.
78. G. V. Trunk, A problem of dimensionality: A simple example, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **1**(3) (1979) 306–307.
79. L. van der Maaten and G. Hinton, Visualizing high-dimensional data using t-sne, *Journal of*

Machine Learning Research **9** (2008) 2579–2605.
80. P. van Mieghem, *Graph Spectra for Complex Networks* (Cambridge University Press, 2010).
81. L. Vandenberghe and S. P. Boyd, Semidefinite programming, *SIAM Review* **38**(1) (1996) 49–95.
82. V. N. Vapnik, *The Nature of Statistical Learning Theory* (Springer-Verlag, 1993).
83. U. von Luxburg, A tutorial on spectral clustering, *Statistics and Computing* **17** (2007) 395–416.
84. F. Wang and J. Sun, Survey on distance metric learning and dimensionality reduction in data mining, *Data Min. Knowl. Discov.* **29** (March 2015) 534–564.
85. J. Wang, Diffusion maps, in *Geometric Structure of High-Dimensional Data and Dimensionality Reduction* (Springer, 2012) pp. 267–298.
86. J. Wang, Hessian locally linear embedding, in *Geometric Structure of High-Dimensional Data and Dimensionality Reduction* (Springer, 2012) pp. 249–265.
87. J. Wang, Local tangent space aligment, in *Geometric Structure of High-Dimensional Data and Dimensionality Reduction* (Springer, 2012) pp. 221–234.
88. J. Wang, Maximum variance unfolding, in *Geometric Structure of High-Dimensional Data and Dimensionality Reduction* (Springer, 2012) pp. 181–202.
89. A. R. Webb and K. D. Copsey, *Statistical Pattern Recognition*, 3 edn. (Wiley, 2011).
90. K. Q. Weinberger and L. K. Saul, Unsupervised learning of image manifolds by semidefinite programming, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 04)*, Vol. 2 (2004) pp. 988–995.
91. K. Q. Weinberger, B. D. Packer and L. K. Saul, Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization, in *in Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics* (2005) pp. 381–388.
92. K. Q. Weinberger, F. Sha and L. K. Saul, Learning a kernel matrix for nonlinear dimensionality reduction, in *In Proceedings of the Twenty First International Conference on Machine Learning (ICML-04* (ACM Press, 2004) pp. 839–846.
93. L. Yang and R. Jin, Distance metric learning: A comprehensive survey, *Michigan State Universiy* **2** (2006).
94. T.-N. Yang and S.-D. Wang, Robust algorithms for principal component analysis, *Pattern Recognition Letters* **20**(9) (1999) 927–933.
95. Q. Ye and W. Zhi, Discrete hessian eigenmaps method for dimensionality reduction, *Journal of Computational and Applied Mathematics* **278** (2015) 197–212.
96. H. Yin and W. Huang, Adaptive nonlinear manifolds and their applications to pattern recognition, *Information Sciences* **180**(14) (2010) 2649–2662.
97. T. Y. Young and T. W. Calvert, *Classification, Estimation and Pattern Recognition* (Elsevier, 1974).
98. T. Zhang, J. Yang, D. Zhao and X. Ge, Linear local tangent space alignment and application to face recognition, *Neurocomputing* **70** (2007) 1547–1553.
99. Z. Y. Zhang and H. Y. Zha, Principal manifolds and nonlinear dimensionality reduction via tangent space aligment, *SIAM Journal on Scientific Computing* **26**(1) (2004) 313–338.
100. H. Zou, T. Hastie and R. Tibshirani, Sparse principal component analysis, *Journal of Computational and Graphical Statistics* **15**(2) (2006) 265–286.