K-graph Laplacians for supervised metric learning

Alexandre L. M. Levada

Computing Department, Federal University of São Carlos, São Carlos, SP, Brazil

Abstract

Metric learning is concerned with the generation of adaptive distance functions for each dataset prior to clustering and classification tasks. Most graph-based approaches employ extrinsic distances, such as the Euclidean distance, to weight the edges of the similarity graph built from the samples. In this paper, we propose the definition of *K-graphs*, using the notion of local curvature from differential geometry to provide an intrinsic cost function for the edges. By approximating the tangent spaces with the PCA subspace, it is possible to compute patch-based estimatives for the principal curvatures of an edge in the graph, in a way that we assign the minimum local curvature to within-class edges and the sum of the maximum local curvatures for between-class edges. The eigenvectors associated to the smallest non-zero aigenvalues of the *K-graph* Laplacian capture relevant information in terms of data clustering. Experiments with several real datasets show that the proposed method can obtain better clusters than some well known dimensionality reduction based metric learning algorithms such as Supervised PCA, LDA, t-SNE and UMAP.

Keywords: Metric learning, dimensionality reduction, differential geometry, curvature, Laplacian Eigenmaps

1. Introduction

Dimensionality reduction based metric learning is an area of machine learning whose objective is to uncover the underlying geometric structure of data by learning a non Euclidean distance function that is better suited to represent the similarity between samples, prior to clustering or classification tasks [1, 2, 3]. Among the computational methods for this purpose, manifold learning algorithms are the most relevant ones. These algorithms are capable of finding more compact and meaningful representations for the observed data by preserving the intrinsic local geometry of the data. Laplacian Eigenmaps is a manifold learning algorithm that performs non-linear dimensionality reduction based in the spectral decomposition of the Laplacian matrix of graphs [4]. Despite several positive aspects, one of the main limitations of Laplacian Eigenmaps is

Email address: alexandre.levada@ufscar.br (Alexandre L. M. Levada)

its sensitivity to noise and outliers. Typically, in datasets that do not lead to smooth manifolds, the performance can be heavily penalized [5].

The foundations of the Laplacian Eigenmaps method are based in the fact that by approximating a manifold by an undirected connected graph $G = (V, E)$, we can find a non-linear mapping from the set of vertices V to an Euclidean subspace R^d, such that locality is preserved. The main property of such transformation is that it defines a smooth mapping, in the sense that neighboring vertices in the graph will remain close together after the output space. Such map is given by the smallest eigenvectors of the graph Laplacian matrix [5]. The representation map obtained by Laplacian Eigenmaps can be interpreted as a discrete approximation to a continuous map that naturally arises from the intrinsic geometry of the manifold: the Laplace-Beltrami operator [6]. 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 [7]. In machine learning, the Laplace Eigenmaps method is closely related to spectral clustering, an unsupervised classification approach for data clustering [8].

Recently, several extensions and improvements have been proposed in order to mitigate some of the limitations in regular Laplacian Eigenmaps. The work of Malik et. al presents the GENILE algorithm, a generalized incremental Laplacian Eigenmaps, which is able to cope with online data, that is, it can deal with the out-of-sample problem by finding the low dimensional representation of samples that do not belong to the training set, as they appear [9]. In the paper of Chen et. al, a soft adaptive loss based Laplacian Eigenmaps (SALE) is proposed to adaptively emphasize the topological relationship between the samples and the clustering structure of data [10]. According to the authors, the L2-norm based loss function in regular Laplacian Eigenmaps is unable to preserve the topological relationship and the method does not represent the real intrinsic structure of data, breaking the manifold structure into multiple local areas in the embedding space. Geometric Laplacian Eigenmap Embedding (GLEE), a method with deep connections with the simplex geometry of the Laplacian matrix, has been proposed to deal to graph reconstruction and link prediction tasks [11]. The main feature of GLEE is that the geometric embedding s_i of the node i will always be orthogonal to each each geometric embedding s_j of non-neighboring nodes j. Learning good graph models from noisy data is crucial for improving the performance of Laplacian Eigenmaps, sice the KNN graph induced from the observations is a rough discrete approximation for the underlying manifold [12]. Given the above, it is clear there is room for improvement in the way that regular Laplacian Eigenmaps algorithm works.

On the other hand, most Laplacian Eigenmaps extensions perform unsupervised metric learning. Since manifold learning and metric learning are intrinsically related, there has been a great interest in studying geometry-aware supervised and semi-supervised dimensionality reduction and metric learning techniques for improving data clustering and classification [13, 14]. Recent studies have reported how dissimilarity measures affect maintaining manifold neighborhood structure and also how supervised manifold learning methods could contribute to the reduction of classification error [15]. The authors point out that it is advisable to use supervised manifold learning techniques as a preprocessing step in clustering and classification tasks, however it may not be reasonable to employ supervised manifold learning for visualization purposes only, since the low dimensional representation is not guaranteed to preserve the internal data structure. In this context, this paper presents *K-graph* Laplacians, a novel dimensionality reduction based supervised metric learning algorithm that uses differential geometry to build an intrinsic similarity measure in terms of the local curvature of an edge. The notion of curvature is directly related to how the tangent space changes along a parametric curve. In summary, the main contribution of this work is: by replacing the extrinsic pointwise Euclidean distance by a patch-based intrinsic one, we believe that the proposed *K-graph* Laplacians can be more robust to the presence of noise and outliers in the observations than regular Laplacian Eigenmaps. Moreover, with the incorporation of the class labels, it is possible to perform supervised metric learning. To validate the method, computational experiments were performed to show that *K-graph* Laplacian can generate better defined clusters in terms of Silhouette coefficient and higher average classification accuracies than several state-of-the-art dimensionality reduction algorithms, such as Supervised PCA [16], LDA [17], t-SNE [18] and UMAP [19, 20].

The remainder of the paper is organized as follows: Section 2 describes the mathematical background of the differential geometry of curves, with emphasis on how the notion of curvature plays an important role in characterizing the shape of parametric curves. Section 3 presents an overview of Laplacian Eigenmaps together with the proposed method, describing in details how to build the *K-graphs* using two approximations for the local curvature. Section 4 shows the experiments and results, and, finally, Section 5 presents our conclusions and final remarks.

2. Differential geometry of curves

The mathematical background for the proposed *K-graphs* is the differential geometry of curves, which formalizes how an arbitrary curve embedded in the space can be completely characterized by intrinsic properties. We provide an introduction on the differential geometry of curves based in some classic literature about the subject [21, 22, 23]. Let $\vec{\alpha}(t) = (x(t), y(t), z(t))$, with $\vec{\alpha} : [a, b] \to R^3$ be an arbitrary parametric curve embedded in the 3D space. In summary, our goal is to show that any curve can be characterized by variations in the Frenet trihedron, an adaptive orthogonal basis that adjusts itself to the curve as we move along it.

First, recall that the tangent vector $\vec{\alpha}'(t) = (x'(t), y'(t), z'(t))$, where $f'(t)$ denotes the first derivative (rate of change) of the function $f(t)$. Another important measure of a parametric curve is the arc length, which measures the total displacement from an initial point $t = A$ to a final point $t = B$. Figure 1 illustrates a diagram that shows how we can approximate the arc length between $t = r$ and $t = s$ by the norm of the difference vector $\vec{s}$.

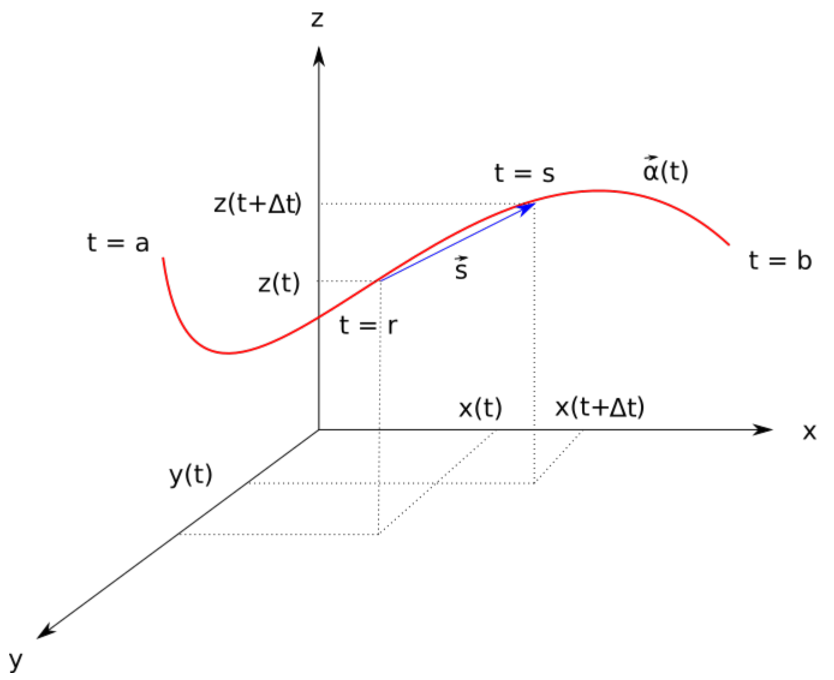

Figure 1: The computation of the arc length can be approximated by the norm of the difference vector.

105 By increasing the number of points in the interval $[r, s]$, the approximation becomes more accurate:

$$s(t) \approx \sum_{t=r}^{s} \sqrt{(x(t+\Delta t) - x(t))^2 + (y(t+\Delta t) - y(t))^2 + (z(t+\Delta t) - z(t))^2} \tag{1}$$

Multiplying and dividing by Δt, it is possible to express $s(t)$ as:

$$s(t) \approx \sum_{t=r}^{s} \sqrt{\left(\frac{x(t+\Delta t) - x(t)}{\Delta t}\right)^2 + \left(\frac{y(t+\Delta t) - y(t)}{\Delta t}\right)^2 + \left(\frac{z(t+\Delta t) - z(t)}{\Delta t}\right)^2} \Delta t \tag{2}$$

Recall that the definition of the derivative of a function is the rate of change, that is:

$$f'(t) = \lim_{\Delta t \to 0} \frac{f(t+\Delta t) - f(t)}{\Delta t} \tag{3}$$

which, in the limiting case ($\Delta t \to 0$) leads to the exact length:

$$s(t) = \int_a^b \sqrt{x'(t)^2 + y'(t)^2 + z'(t)^2} dt = \int_a^b \sqrt{\vec{\alpha}'(t)^T \vec{\alpha}'(t)} dt = \int_a^b \|\vec{\alpha}'(t)\| dt \tag{4}$$

In some cases, an arc length parametrization is a more natural choice. The idea with this change of variables is to have a 1-1 mapping between time (t) and space (s), so that instead of $t \in [a, b]$ we have $s \in [0, L]$, where $L = \int_a^b \|\vec{\alpha}'(t)\| dt$. This is equivalent to moving along the curve with a constant velocity equal to one, that is:

$$\vec{\alpha}'(s) = \frac{\vec{\alpha}'(t)}{\|\vec{\alpha}'(t)\|} = \vec{T}(t) \tag{5}$$

With the arc length parametrization we have both tangent and normal vector fields normalized. In the following, we will see that the normal vector, which is the second derivative of $\vec{\alpha}(t)$, is orthogonal to the tangent vector. As $\|\vec{\alpha}'(s)\|^2 = 1$, we have:

$$x'(s)^2 + y'(s)^2 + z'(s)^2 = 1 \tag{6}$$

Differentiation with respect to s leads to:

$$x'(s)x''(s) + y'(s)y''(s) + z'(s)z''(s) = 0 \tag{7}$$

which can be expressed by:

$$\vec{\alpha}'(s)^T \vec{\alpha}''(s) = 0 \tag{8}$$

As $\vec{\alpha}'(s) = \vec{T}(t)$, then $\vec{\alpha}''(s) = \vec{T}'(t)$, which means that the normal field is in fact the variation of the tangent field. The normal vector can be normalized as:

$$\vec{N}(t) = \frac{\vec{T}'(t)}{\|\vec{T}'(t)\|} = \frac{\vec{\alpha}''(s)}{\|\vec{\alpha}''(s)\|} \tag{9}$$

At this point, we are ready to introduce the notion of curvature of a curve. The curvature of $\vec{\alpha}(s)$ at a point $s \in [0, L]$ is the rate of change of the unit tangent vector at this point, given by:

$$K(s) = \|\vec{\alpha}''(s)\| = \|\vec{T}'(t)\| \tag{10}$$

To compute the curvature at a given point, we need to express it in terms of known quantities. Note that:

$$\vec{\alpha}'(s) = \frac{\vec{\alpha}'(t)}{\|\vec{\alpha}'(t)\|} \tag{11}$$

which leads to:

$$\vec{\alpha}'(t) = \|\vec{\alpha}'(t)\|\vec{\alpha}'(s) \tag{12}$$

The arc length of a curve is:

$$s = \int_a^b \|\vec{\alpha}'(t)\|dt \tag{13}$$

And by the fundamental theorem of calculus:

$$\frac{ds}{dt} = \|\vec{\alpha}'(t)\| \tag{14}$$

Replacing equation (14) into (12):

$$\vec{\alpha}'(t) = \frac{ds}{dt}\vec{\alpha}'(s) \tag{15}$$

By the product rule, we can differentiate both sides with respect to t, leading to:

$$\vec{\alpha}''(t) = \frac{d^2s}{dt^2}\vec{\alpha}'(s) + \frac{ds}{dt}\frac{d}{ds}\vec{\alpha}'(s)\frac{ds}{dt} = \frac{d^2s}{dt^2}\vec{\alpha}'(s) + \left(\frac{ds}{dt}\right)^2 \vec{\alpha}''(s) \tag{16}$$

Computing the cross product with respect to $\vec{\alpha}'(t)$, we have:

$$\vec{\alpha}'(t) \times \vec{\alpha}''(t) = \vec{\alpha}'(t) \times \left[\frac{d^2s}{dt^2}\vec{\alpha}'(s) + \left(\frac{ds}{dt}\right)^2 \vec{\alpha}''(s)\right] \tag{17}$$

Note that $\vec{\alpha}'(s)$ and $\vec{\alpha}'(t)$ point to the same direction so their cross product is equal to zero:

$$\vec{\alpha}'(t) \times \vec{\alpha}''(t) = \left(\frac{ds}{dt}\right)^2 [\vec{\alpha}'(t) \times \vec{\alpha}''(s)] \tag{18}$$

But from equation (14), we have:

$$\vec{\alpha}'(t) \times \vec{\alpha}''(t) = \|\vec{\alpha}'(t)\|^2 [\vec{\alpha}'(t) \times \vec{\alpha}''(s)] \tag{19}$$

By computing the norm of the cross products, we can write:

$$\|\vec{\alpha}'(t) \times \vec{\alpha}''(t)\| = \|\vec{\alpha}'(t)\|^2\|\vec{\alpha}'(t)\|\|\vec{\alpha}''(s)\|sin(\theta) \tag{20}$$

where θ denotes the angle between $\vec{\alpha}'(t)$ and $\vec{\alpha}''(s)$. However, as the tangent vectors and the normal vectors are orthogonal, the final expression for the curvature $K(t)$ is given by:

$$K(s) = \|\vec{\alpha}''(s)\| = \frac{\|\vec{\alpha}'(t) \times \vec{\alpha}''(t)\|}{\|\vec{\alpha}'(t)\|^3} \tag{21}$$

In the following, we describe the important role of curvature $K(s)$ in the study of variations in the tangent field, which here, in our graph-based model, is defined by the tangent space at a given point in a manifold M, represented by a vertex in the graph.

We now define the binormal unit vector as $\vec{B}(t) = \vec{T}(t) \times \vec{N}(t)$. It should be clear at this point that the three vectors $\{\vec{T}(t), \vec{N}(t), \vec{B}(t)\}$, known as the Frenet Trihedron, define an orthonormal basis at each point $t \in [a, b]$ of the parametric curve. In summary, the Frenet Trihedron is an adaptive referential that adjusts itself to a curve $\vec{\alpha}(t)$ as we move along it. Our next step consists in expressing the rate of change of the Frenet Trihedron in terms of itself.

To understand how the tangent vector changes from one point to a neighboring one, note that:

$$\vec{T}'(t) = \|\vec{T}'(t)\|\vec{N}(t) = K(t)\vec{N}(t) \tag{22}$$

which means that the variation in the tangent field is expressed only in terms of the normal field, through the curvature $K(t)$.

Moving forward to changes in the binormal vector, an expansion of $\vec{B}'(t)$ in the components of the Frenet trihedron can be expressed as a linear combination:

$$\vec{B}'(t) = c_1\vec{T}(t) + c_2\vec{N}(t) + c_3\vec{B}(t) \tag{23}$$

As $\vec{B}(t)^T\vec{B}(t) = 1$, by simple differentiation with respect to t, we have:

$$\vec{B}'(t)^T\vec{B}(t) + \vec{B}(t)^T\vec{B}'(t) = 0 \tag{24}$$

which leads to:

$$2\vec{B}'(t)^T\vec{B}(t) = 0 \tag{25}$$

meaning that $\vec{B}'(t)$ is orthogonal to $\vec{B}(t)$ and $c_3 = 0$. We also know that $\vec{B}(t)^T\vec{T}(t) = 0$, so by differentiating with respect to t, we have:

$$\vec{B}'(t)^T\vec{T}(t) + \vec{B}(t)^T\vec{T}'(t) = 0 \tag{26}$$

However $\vec{T}'(t) = K(t)\vec{N}(t)$ and we can write:

$$\vec{B}'(t)^T\vec{T}(t) + K(t)\vec{B}(t)^T\vec{N}(t) = 0 \tag{27}$$

Finally, as $\vec{B}(t)^T\vec{N}(t) = 0$, we reach:

$$\vec{B}'(t)^T\vec{T}(t) = 0 \tag{28}$$

which means that $c_1 = 0$. Therefore, as $\vec{B}'(t) \perp \vec{T}(t)$ and $\vec{B}'(t) \perp \vec{N}(t)$, the only possible situation is to have $\vec{B}'(t)$ parallel to the normal vector $\vec{N}(t)$:

$$\vec{B}'(t) = c_2\vec{N}(t) \tag{29}$$

By a simple convention, $c_2 = -\tau(t)$ is the negative of the torsion of the curve. In summary, this quantity measures how fast the curve escapes the plane defined by the vectors $\vec{T}(t)$ and $\vec{N}(t)$.

Similarly, it is possible to express $\vec{N}'(t)$ in terms of the components or the Frenet Trihedron:

$$\vec{N}'(t) = c_1\vec{T}(t) + c_2\vec{N}(t) + c_3\vec{B}(t) \tag{30}$$

Note that $\|\vec{N}(t)\| = 1$, which leads to $\vec{N}'(t)^T\vec{N}(t) = 0$, by simple differentiation. Hence, $c_2 = 0$. Note also that $\vec{N}(t)^T\vec{T}(t) = 0$ and by differentiating in t leads to:

$$\vec{N}'(t)^T\vec{T}(t) + \vec{N}(t)^T\vec{T}'(t) = 0 \tag{31}$$

$$\vec{N}'(t)^T\vec{T}(t) + K(t)\vec{N}(t)^T\vec{N}(t) = 0 \tag{32}$$

$$\vec{N}'(t)^T\vec{T}(t) = -K(t) \tag{33}$$

The previous equation states that the projection of the vector $\vec{N}'(t)$ into the tangent vector is the negative of the curvature, that is, $c_1 = -K(t)$. Moreover, as $\vec{N}(t)^T\vec{B}(t) = 0$, simple differentiation leads to:

$$\vec{N}'(t)^T\vec{B}(t) + \vec{N}(t)^T\vec{B}'(t) = 0 \tag{34}$$

$$\vec{N}'(t)^T\vec{B}(t) - \tau(t)\vec{N}(t)^T\vec{N}(t) = 0 \tag{35}$$

$$\vec{N}'(t)^T\vec{B}(t) = \tau(t) \tag{36}$$

showing that the projection of the vector $\vec{N}'(t)$ into the binormal vector is the torsion, that is, $c_3 = \tau(t)$. Therefore, we have the Frenet-Serret equations:

$$\vec{T}'(t) = K(t)\vec{N}(t) \tag{37}$$

$$\vec{N}'(t) = -K(t)\vec{T}(t) + \tau(t)\vec{B}(t) \tag{38}$$

$$\vec{B}'(t) = -\tau(t)\vec{N}(t) \tag{39}$$

The importance of these equations is that they state the following:

- **Variation** in the **tangent field** depends **only** of the **curvature** $K(t)$.
- Variation in the normal field depends of the negative of the curvature, $-K(t)$, and the torsion, $\tau(t)$.
- Variation in the binormal field depends only of the negative of the torsion, $-\tau(t)$.

Hence, in the study and analysis of parametric curves embedded in an ambient space, knowledge of the curvature $K(t)$ and the torsion $\tau(t)$ implies in having a complete intrinsic characterization of $\vec{\alpha}(t)$.

3. K-graph Laplacians

A parametric curve defined on a manifold can be approximated by shortest paths in the discrete graph. When we move along one of these curves, the Frenet Trihedron encodes relevant information about its shape. In this paper, we propose to extend this concept by using the orthonormal basis defined by the tangent space at a given point of the manifold to extract geometric information. Since all the variation in the tangent field is expressed in terms of the curvature $K(t)$, our proposal consists in measuring the variation of the tangent spaces along the edges of the KNN graph, which is a discrete approximation for the underlying manifold, in order to estimate the local curvature.

3.1. K-graphs

The K-graphs are built by weighting the edges of the KNN graph with the local curvatures, in a way that edges with higher curvature are more distorted than edges with lower curvature. The most important aspect in building the K-graphs is to properly estimate the local curvature of the edges.

Let $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$, with $\vec{x}_i \in R^m$, be the input data matrix in which each column denotes a sample. The first step is the creation of a discrete approximation for the manifold, given by the KNN graph from the samples in X. To identify the K nearest neighbors of a given sample, we use the regular Euclidean distance. Let η_i be the neighborhood system of $\vec{x}_i$ and a patch P_i be defined as the set $\{\vec{x}_i \cup \eta_i\}$. Note that the cardinality of P_i is $K + 1$, for $i = 1, 2, ..., n$. In matrix notation, a patch P_i is:

$$P_i = [\vec{x}_i, \vec{x}_{i1}, \vec{x}_{i2}, ..., \vec{x}_{ik}] = \begin{bmatrix} x_i(1) & x_{i1}(1) & \ldots & x_{ik}(1) \\ x_i(2) & x_{i1}(2) & \ldots & x_{ik}(2) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ldots & \vdots \\ x_i(m) & x_{i1}(m) & \ldots & x_{ik}(m) \end{bmatrix}_{m\times(k+1)} \tag{40}$$

One way to approximate the tangent space at $\vec{x}_i$, is to compute the m principal directions that span the PCA subspace. It is known that these vectors define an orthogonal basis. The algorithm for building the tangent space at $\vec{x}_i$ can be summarized by:

1. First, compute the covariance matrix of the samples in P_i, denoted by Σ_i;
2. Then, perform an eigendecomposition of the covariance matrix Σ_i, as:

$$\Sigma_i = U_i Q_i U_i^T \tag{41}$$

where $U_i = [\vec{u}_{i1}, \vec{u}_{i2}, \ldots, \vec{u}_{im}]$ is a $m \times m$ matrix in which each column represents an eigenvector and Q_i is a $m \times m$ diagonal matrix with the eigenvalues of Σ_i

The columns of U_i spans the tangent space at $\vec{x}_i$. The goal of building the K-graph is to quantify the variation of the tangent spaces when we move along the edges of the KNN graph. In order to compute the principal curvatures of an edge of the graph, we employ two approaches: A) Naive approximation; and B) Differential geometry approximation.

3.1.1. Naive approximation for the local curvature

Recall that, by definition, curvature is the rate of change of the tangent vector, that is:

$$K(t) = \|\vec{T}'(t)\| \tag{42}$$

Then, a direct and naive approximation can be obtained with a simple finite difference scheme. We begin by defining $P = v_a v_1 v_2 \dots v_b$ as the shortest path between the samples $\vec{x}_a$ and $\vec{x}_b$ in the KNN graph. Let P_i be the patch for the i-th sample in the path P. The ideia consists in finding the covariance matrices Σ_i and Σ_j for the edge (v_i, v_j) of the KNN graph using the patches P_i and P_j. Using PCA, we can compute the tangent spaces at $\vec{x}_i$ and $\vec{x}_j$. Let $U_i = [\vec{u}_{i1}, \vec{u}_{i2}, \dots, \vec{u}_{im}]$ and $U_j = [\vec{u}_{j1}, \vec{u}_{j2}, \dots, \vec{u}_{jm}]$ be the orthogonal basis (principal directions) that define the tangent spaces. Recall that we have m tangent vectors, so we will associate one curvature to each principal direction. Our naive approximation for the l-th principal curvature along the edge (v_i, v_j) is the difference between the l-th principal directions at $\vec{x}_i$ and $\vec{x}_j$:

$$K_{ij}^{(l)} = \|\vec{u}_{il} - \vec{u}_{jl}\| \qquad \text{for } l = 1, 2, ..., m \tag{43}$$

where $\vec{K}_{ij}$ is a vector of principal curvatures.

3.1.2. Differential geometry approximation for the local curvature

We know that the curvature at a point t can be computed exactly by:

$$K(t) = \frac{\|\vec{\alpha}'(t) \times \vec{\alpha}''(t)\|}{\|\vec{\alpha}'(t)\|^3} \tag{44}$$

where $\vec{\alpha}'(t)$ and $\vec{\alpha}''(t)$ are the tangent and normal vectors. Denoting by $U_i = [\vec{u}_{i1}, \vec{u}_{i2}, \dots, \vec{u}_{im}]$ and $U_j = [\vec{u}_{j1}, \vec{u}_{j2}, \dots, \vec{u}_{jm}]$ the orthogonal basis that define the tangent spaces at $\vec{x}_i$ and $\vec{x}_j$, we use finite difference to approximate the normal vectors (second derivatives) as:

$$N = [\vec{n}_1, \vec{n}_2, ..., \vec{n}_m] = [\vec{u}_{i1} - \vec{u}_{j1}, \vec{u}_{i2} - \vec{u}_{j2}, \dots, \vec{u}_{im} - \vec{u}_{jm}] \tag{45}$$

Hence, it is possible to compute the numerator of (44) for the l-th tangent vector as:

$$\|\vec{u}_{il} \times \vec{n}_l\| = \|\vec{u}_{il}\| \|\vec{n}_l\| sin(\theta_l) \qquad \text{for } l = 1, 2, ..., m \tag{46}$$

where the angle θ_l is given by:

$$\theta_l = arccos\left(\frac{\vec{u}_{il}^T \vec{n}_l}{\|\vec{u}_{il}\| \|\vec{n}_l\|}\right) \qquad \text{for } l = 1, 2, ..., m \tag{47}$$

The denominator of (44) can be computed as:

$$\|\vec{u}_{il}\|^3 = \left(u_{il}(1)^2 + u_{il}(2)^2 + \cdots + u_{il}(m)^2\right)^{3/2} \tag{48}$$

Therefore, the vector of principal curvatures becomes:

$$\hat{K}_{ij}^{(l)} = \frac{\|\vec{u}_{il}\| \|\vec{n}_l\| sin(\theta_l)}{\|\vec{u}_{il}\|^3} \qquad \text{for } l = 1, 2, ..., m \tag{49}$$

3.2. K-graphs for supervised metric learning

In order to perform dimensionality reduction based metric learning, we replace the extrinsic Euclidean distance used to weight the edges of the KNN graph by the following rule: if there is an edge between $\vec{x}_i$ and $\vec{x}_j$ and both belong to the same class, the the weight of the corresponding edge is the minimum between the smallest principal curvature in K_{ij}, estimated by equation (43), and the smallest principal curvature in $\hat{K}_{ij}$, estimated by equation (49). However, if $\vec{x}_i$ and $\vec{x}_j$ belong to different classes, the weight of the corresponding edge is the summation of the maximum principal curvature in K_{ij} and the maximum principal curvature in $\hat{K}_{ij}$. In summary, the supervised K-graph have edge weights defined by:

$$w_{ij} = \begin{cases} min\{min(K_{ij}), min(\hat{K}_{ij})\} & \text{if } l_i = l_j \\ max(K_{ij}) + max(\hat{K}_{ij}) & \text{if } l_i \neq l_j \end{cases} \tag{50}$$

where l_i denotes the label of the sample $\vec{x}_i$.

3.3. Laplacian Eigenmaps

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

$$d_i = \sum_{j=1}^{n} w_{ij} \tag{51}$$

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 [24, 25, 26, 27, 28].

Definition 3. *The unnormalized graph Laplacian matrix is defined by:*

$$L = D - W \tag{52}$$

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 [8]. More details about the Laplacian spectrum and advanced properties can be found in Mohar's paper [29].

Theorem 1. *The Laplacian matrix L satisfies the following properties:*

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

$$\vec{f}^T L \vec{f} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (f_i - f_j)^2 \tag{53}$$

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 $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_m$.*

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

$$\begin{aligned}
\vec{f}^T L \vec{f} &= \vec{f}^T D \vec{f} - \vec{f}^T W \vec{f} = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i=1}^{n} \sum_{j=1}^{n} f_i w_{ij} f_j \\
&= \frac{1}{2} \left(2 \sum_{i=1}^{n} d_i f_i^2 - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} f_i w_{ij} f_j \right) \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} d_i f_i^2 - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} f_i f_j + \sum_{i=1}^{n} d_j f_j^2 \right) \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} f_i^2 - 2 \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} f_i f_j + \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} f_j^2 \right) \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (f_i - f_j)^2
\end{aligned} \tag{54}$$

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:

$$L\vec{1} = (D - W)\vec{1} = D\vec{1} - W\vec{1} = \sum_{i=1}^{n} d_i - \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} = \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} d_i = 0 \quad (55)$$

showing that the constant eigenvector $\vec{1}$ has zero eigenvalue. Finally, the fourth statement is a direct consequence of statements (2) and (3).

3.4. 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:

$$J(\vec{y}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(y_i - y_j)^2 = \vec{y}^T L \vec{y} \quad (56)$$

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:

$$\arg\min_{\vec{y}} \; y^T L y \qquad \text{subject to} \quad y^T D y = 1 \quad (57)$$

where the constraint $y^T D y = 1$ removes an arbitrary scaling factor in the embedding [6]. 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:

$$L(\vec{y}, \lambda) = y^T L y - \lambda(y^T D y - 1) \quad (58)$$

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

$$\frac{\partial}{\partial \vec{y}} L(\vec{y}, \lambda) = 2L\vec{y} - 2\lambda D\vec{y} = 0 \tag{59}$$

which leads to:

$$L\vec{y} = \lambda D\vec{y} \tag{60}$$

$$(D^{-1}L)\vec{y} = \lambda\vec{y} \tag{61}$$

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 [30, 24].

3.4.1. 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:

$$J(Y) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left\| \vec{y}^{(i)} - \vec{y}^{(j)} \right\|^2 \tag{62}$$

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:

$$J(Y) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} (\vec{y}^{(i)} - \vec{y}^{(j)})(\vec{y}^{(i)} - \vec{y}^{(j)})^T \tag{63}$$

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

$$\begin{aligned}
J(Y) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[W_{ij}\vec{y}^{(i)}\vec{y}^{(i)^T} - W_{ij}\vec{y}^{(i)}\vec{y}^{(j)^T} - W_{ij}\vec{y}^{(j)}\vec{y}^{(i)^T} + W_{ij}\vec{y}^{(j)}\vec{y}^{(j)^T}\right] \\
&= \frac{1}{2}\left[\sum_{i=1}^{n} d_i\vec{y}^{(i)}\vec{y}^{(i)^T} - 2\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\vec{y}^{(i)}\vec{y}^{(j)^T} + \sum_{j=1}^{n} d_j\vec{y}^{(j)}\vec{y}^{(j)^T}\right] \\
&= \frac{1}{2}\left[2\sum_{i=1}^{n} d_i\vec{y}^{(i)}\vec{y}^{(i)^T} - 2\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\vec{y}^{(i)}\vec{y}^{(j)^T}\right] \\
&= \sum_{i=1}^{n} d_i\vec{y}^{(i)}\vec{y}^{(i)^T} - \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\vec{y}^{(i)}\vec{y}^{(j)^T} \qquad (64)
\end{aligned}$$

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:

$$J(Y) = Tr(DYY^T) - Tr(WYY^T) \qquad (65)$$

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

$$\begin{aligned}
J(Y) &= Tr(Y^TDY) - Tr(Y^TWY) = Tr(Y^T(DY - WY)) \\
&= Tr(Y^T(D-W)Y) = Tr(Y^TLY) \qquad (66)
\end{aligned}$$

Therefore, we have the following constrained optimization problem:

$$\arg\min_{Y}\ Tr(Y^TLY) \qquad \text{subject to} \quad Y^TDY = I \qquad (67)$$

whose Lagrangian function is given by:

$$L(Y,\lambda) = Tr(Y^TLY) - \lambda(Y^TDY - I) \qquad (68)$$

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

$$\frac{\partial}{\partial Y}L(Y,\lambda) = 2LY - 2\lambda DY = 0 \qquad (69)$$

leading to the following eigenvector problem:

$$LY = \lambda DY \qquad (70)$$

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, some extensions of Laplacian Eigenmaps have been made [5]. Algorithm 1 shows a summary of the Laplacian Eigenmaps method.

Algorithm 1 Laplacian Eigenmaps

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

$$W_{ij} = exp\left\{-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{t}\right\} \quad \text{if} \quad v_j \in N(v_i) \tag{71}$$

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

$$d_i = \sum_{j=1}^{n} W_{ij} \tag{72}$$

```
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
```

3.4.2. Graph cuts and Laplacian Eigenmaps

It has been shown that there exists a deep relation between the problem of finding the minimum cut in a weighted graph and Spectral Clustering, which is the application of K-means after Laplacian Eigenmaps. In the following, we present a brief discussion about this intrinsic connection based in the seminal paper of Luxburg [8]. First, recall that the normalized cut RCut is:

$$RCut(A_1, A_2, ..., A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{w(A_i, \overline{A_i})}{|A_i|} \tag{73}$$

where $|A_i|$ denotes the number of elements in the partition A_i and:

$$w(A_i, \overline{A_i}) = \sum_{i \in A_i; j \in \overline{A_i}} w_{ij} \tag{74}$$

is the summation of the weights of the edges with one vertex in A_i and another vertex in $\overline{A_i}$. Typically, the problem of finding the cut that minimizes

RCut is NP-Hard. For a binary problem, that is, $k = 2$, we have to minimize $RCut(A, \overline{A})$, where:

$$RCut(A, \overline{A}) = \frac{1}{2}\left[\frac{w(A, \overline{A})}{A} + \frac{w(\overline{A}, A)}{\overline{A}}\right] \tag{75}$$

It is possible to associate the value of RCut with the Laplacian matrix of the graph. Let $\vec{f} \in R^n$ be defined by:

$$f_i = \begin{cases} \sqrt{\frac{|\overline{A}|}{|A|}} & v_i \in A \\ -\sqrt{\frac{|A|}{|\overline{A}|}} & v_i \in \overline{A} \end{cases} \tag{76}$$

We know that:

$$\vec{f}^T L \vec{f} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(f_i - f_j)^2 \tag{77}$$

It has been shown that the previous equation can also be computed by [8]:

$$\vec{f}^T L \vec{f} = |V| RCut(A, \overline{A}) \tag{78}$$

Therefore, minimization of RCut is mathematically equivalent to:

$$\arg\min \ \vec{f}^T L \vec{f} \quad \text{s.t.} \quad \vec{f}^T \vec{1} = 0 \quad \text{and} \quad \|f\| = \sqrt{n} \tag{79}$$

Note that the problem is NP-Hard, since, as the dimensionality of the solution vector $\vec{f}$ is n, and each component f_i can assume one of two possible values, we have a total of 2^n candidate solutions. Exhaustive search in unfeasible for large values of n. By relaxing the problem, that is, by allowing that $f_i \in R$, the solution to the relaxed problem is known to be $\vec{f} = \vec{v}_1$, where $\vec{v}_1$ is the eigenvector associated to the smallest non-zero eigenvalue of the Laplacian matrix (recall that the smallest eigenvalue is zero). After that, we quantize the components of the vector $\vec{f}$, making $f_i = 0$ if $f_i < 0$ and $f_i = 1$ if $f_i \geq 0$, which can be performed by a clustering algorithm such as K-means. In fact, it has been shown that the regular Laplacian leads to an approximation to the minimization of the RCut, while the normalized Laplacian leads to an approximation for NCut, defined by:

$$NCut(A_1, A_2, ..., A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{w(A_i, \overline{A_i})}{vol(A_i)} \tag{80}$$

where $vol(A_i)$ is the summation of the degrees of the nodes in A_i. Basically, by relaxing the problem we have different sub-optimal solutions depending on the type of the Laplacian matrix (normalized versus unnormalized). In the following, we present the algorithm for K-graph Laplacian Eigenmaps.

Algorithm 2 K-graph Laplacian Eigenmaps

```
1: function K-GraphLaplacian($X, K, d, t, l$)
2:     From the input data $X_{m \times n}$ build a KNN graph, linking each sample to its K nearest neighbors.
3:     For each neighborhood of the KNN graph, compute the local covariance matrix and find its $m$ eigenvectors.
4:     For each edge $(i, j)$ of the KNN graph, compute the two approximations for the principal curvatures, resulting in the vectors $K_{ij}$ and $\hat{K}_{ij}$
5:     Assign novel weights to the edges of the KNN graph in order to define the K-graph as:
           $$DK_{ij} = \begin{cases} min\{min(K_{ij}), min(\hat{K}_{ij})\} & \text{if } l_i = l_j \\ max(K_{ij}) + max(\hat{K}_{ij}) & \text{if } l_i \neq l_j \end{cases} \quad (81)$$
6:     Choose the weights to define the adjacency matrix $W$ as:
           $$W_{ij} = exp\left\{ -\frac{DK_{ij}^2}{t} \right\} \quad (82)$$
7:     Compute the diagonal matrix $D$ with the degrees $d_i$ for $i = 1, 2, ..., n$.
           $$d_i = \sum_{j=1}^{n} W_{ij} \quad (83)$$
8:     Compute the Laplacian matrix $L = D - W$
9:     Select the bottom $d$ eigenvectors with non-zero eigenvalues of $D^{-1}L$ and define the matrix $Y$, where each column is an eigenvector.
10:    return $Y$
11: end function
```

In the next section, we present some computational experiments to compare the performance of the proposed *K-graph* Laplacian Eigenmaps against other dimensionality reduction algorithms in terms of metric learning. The idea is to analyze the quality of the clusters and the discriminant power of the extracted features in the 2D case for supervised classification.

4. Experiments and results

In order to test and evaluate the performance of the proposed *K-graph* Laplacian for supervised metric learning, we perform two computational experiments: 1) analysis of the quality of the clusters obtained after dimensionality reduction to 2-D spaces with the Silhouette Coefficient; and 2) analysis of the maximum classification accuracies obtained by eight supervised classifiers (KNN, SVM, Decision Trees, Naive Bayes, Quadratic Discriminant Analysis, Multilayer Perceptron, Random Forest and Gaussian Process classifiers) after dimensionality reduction to 2-D spaces. It is expected that the better the metric learning process, the higher the values of the performance evaluation measures. We compared our method against PCA, Supervised PCA, ISOMAP, t-SNE, UMAP and LDA. The Silhouette Coefficients obtained in the first set of experiments are shown in Table 1. All 50 datasets used in the experiments, as well as detailed information regarding the number of instances, features and classes for each one of them, are freely available at `openML.org`. Note that the proposed *K-graph* Laplacian Eigenmaps method outperformed the other algorithms in all datasets. To illustrate the obtained clusters after the dimensionality reduction process, Figure 4 shows a comparison of the 2D point clouds for the mfeat-fourier dataset, composed by two thousand 76 dimensional vectors divided in 10 classes. It is possible to see that the cluster structure provided by the proposed *K-graph* Laplacian Eigenmaps is more suitable in terms of data discrimination.

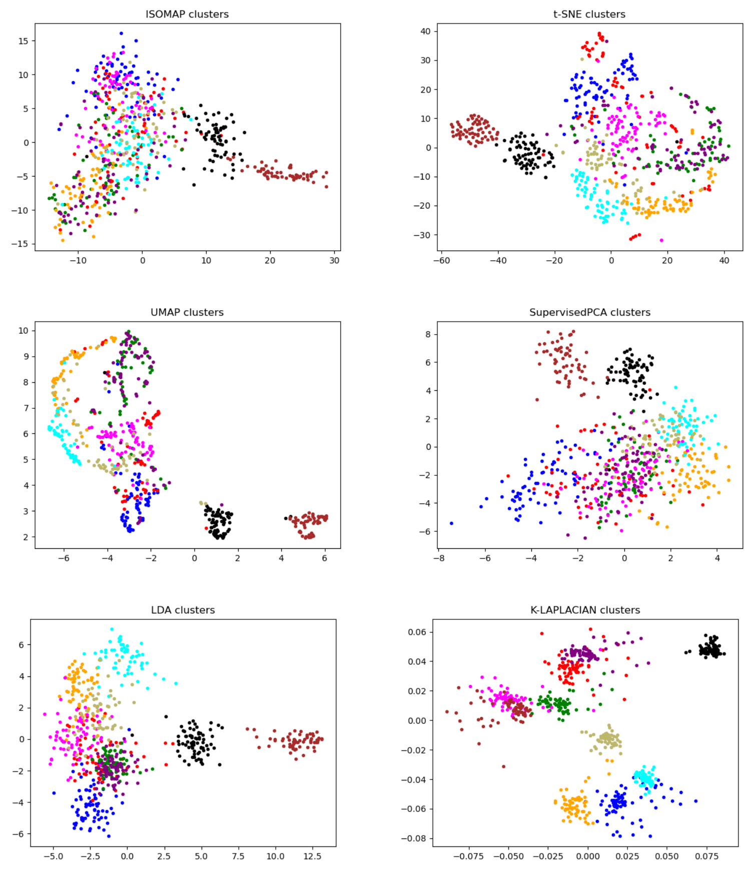

Figure 2: Clusters after dimensionality reduction of the mfeat-fourier dataset. From left to right, top to bottom: ISOMAP, t-SNE, UMAP, Supervised PCA, LDA and *K-graph* Laplacian.

One interesting observation concerns the definition of the parameters K, the number of neighbors in the KNN graph, and t, the variance of the Gaussian kernel in *K-graph* Laplacian Eigenmaps. Overall, an empirical analysis based in these 50 datasets suggests that large values of K leads to better clusters (with high inter-class distances and low intra-class distances). Our strategy was to define the number of neighbors as $K = 0.75n$ (where n is the number of samples). In some datasets, a smaller value of K, such as $K = 0.5n$, can lead to better results, but in average, selecting around 75% of the nodes is a good empirical choice. With respect to the parameter t, in most cases, a value of $t = 1$ is a good choice, but variations in this parameter does not have a drastic effect in the performance.

To test if the Silhouette Coefficients obtained by the proposed method are statistically superior than those obtained by the other methods, we performed a Kruskal-Wallis test [31], a non-parametric version of one-way ANOVA. For a significant level $\alpha = 0.01$, we conclude that there are strong evidences against the null hypothesis that all methods are identical ($p = 5.74 \times 10^{-24}$). In order to analyze which groups are significantly different, we performed a Nemenyi post-hoc test [32, 33]. According to the test, for a significance level $\alpha = 0.01$, there are strong evidences that *K-graph* Laplacian Eigenmaps produced significantly higher SC's than PCA ($p < 10^{-18}$), ISOMAP ($p < 10^{-18}$), t-SNE ($p < 10^{-18}$), UMAP ($p < 10^{-18}$), Supervised PCA ($p = 3.24 \times 10^{-12}$) and LDA ($p = 0.0003$).

In the second round of experiments, for each dataset, after dimensionality reduction based metric learning is performed, we used 50% of the samples to train eight different supervised classifiers: KNN ($K = 7$), SVM (linear), NB, DT, QDA (Gaussian hypothesis), MPL, RFC and GPC. Then, each one of them was used to classify the 50% remaining samples from the test set and the maximum accuracy among them was selected to evaluate the behavior of supervised classification. All the results can be found in Table 2. Note that in all datasets, the proposed *K-graph* Laplacian Eigenmaps obtained the higher classification accuracy. Once again, to visualization purposes, Figure 4 shows a comparison of the 2D clusters for the vehicle dataset, composed by 846 samples with 18 features divided in 4 classes.

To test if the maximum classification accuracies obtained by the proposed method are statistically superior than those obtained by the other methods, we performed another Kruskal-Wallis test [31]. For a significant level $\alpha = 0.01$, we conclude that there are strong evidences against the null hypothesis that all methods are identical ($p = 9.51 \times 10^{-19}$). In order to analyze which groups are significantly different, we performed a Nemenyi post-hoc test [32, 33]. According to the test, for a significance level $\alpha = 0.01$, there are strong evidences that *K-graph* Laplacian Eigenmaps produced significantly higher classification accuracies than PCA ($p < 10^{-18}$), ISOMAP ($p < 10^{-18}$), t-SNE ($p = 4.21 \times 10^{-11}$), UMAP ($p = 1.11 \times 10^{-15}$), Supervised PCA ($p = 3.07 \times 10^{-13}$) and LDA ($p = 2.33 \times 10^{-6}$).

In comparison to LDA, a positive aspect of the proposed method is that it the number of features to be extracted does not depend on the number of classes, which makes it a good choice for binary classification problems. On the other

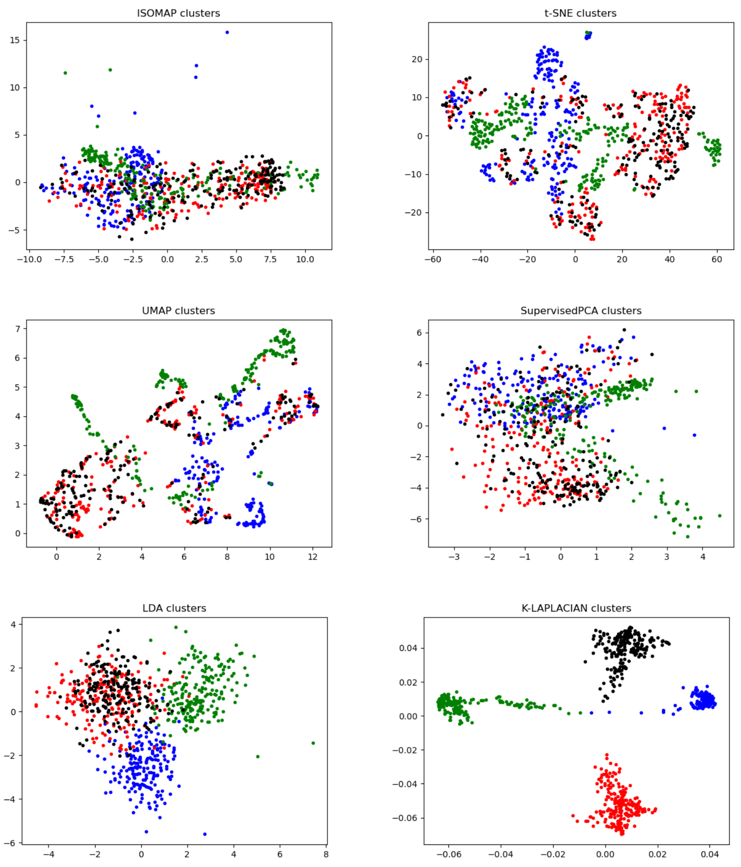

Figure 3: Clusters after dimensionality reduction of the vehicle dataset. From left to right, top to bottom: ISOMAP, t-SNE, UMAP, Supervised PCA, LDA and *K-graph* Laplacian.

hand, in binary classification problems, LDA can only extract a single feature, which is not always enough for data discrimination. However, the proposed method has some drawbacks. One limitation of the proposed method is the high computational burden due to the estimation of the tangent spaces used in the computation of the curvature of the edges in the building of the *K-graph*. Note that this process is equivalent to n applications of PCA, where n is the number of samples. Moreover, we have m principal curvatures, where m is the number of features. Therefore, for large datasets, the proposed *K-graph* Laplacian Eigenmaps can be time consuming. Our Python implementation can take hours to produce a result, depending on the sample size. However, for smaller datasets, the results are obtained in few seconds. Another limitation is the out-of-sample problem, common to most manifold learning algorithms. In summary, there is no efficient way to find the low dimensional representation of samples that do not belong to the training set, which, in practice, is quite frustrating. For the interested reader, the source code can be found at: `https://github.com/alexandrelevada/k-graph-laplacian`.

5. Conclusions

Supervised dimensionality reduction based metric learning is concerned with the construction of low dimensional data representation optimized for clustering and classification tasks. In this paper, a differential geometry based approach was proposed to incorporate intrinsic curvature information into the KNN graph, creating the *K-graphs*. In summary, the idea is to deform the edges of the graph according to the local geometry of the manifold: the higher the curvature, the more an edge is twisted. Our claim is that the proposed *K-graph* Laplacian Eigenmaps is an alternative to the existing methods in the literature, since the computational experiments supported two main points: 1) in terms of Silhouette Coefficients, the clusters produced by *K-graph* Laplacian Eigenmaps can be superior to those obtained by other manifold learning algorithms; and 2) the features extracted by *K-graph* Laplacian Eigenmaps can be more discriminative in supervised classification than features obtained by other dimensionality reduction based metric learning algorithms.

In general, the performance of *K-graph* Laplacian is promising due to the main contribution of the proposed method: we employ an intrinsic patch-based distance function to measure the similarity between the samples, which is less sensitive than the pointwise extrinsic Euclidean distance to the presence of noise and outliers in the observed data. However, the method is not perfect. The main limitations of *K-graph* Laplacian can be summarized by: 1) depending on the number of samples (n) and the number of features (m), the proposed method cab be computationally expensive and quite slow; 2) it is not possible to find a map of out-of-sample instances in an fast and straightforward way. On the other hand, a positive feature is that, unlike autoencoders and other deep-learning based approaches, *K-graph* Laplacian works very well in small sample size problems.

Future works may include the application of other techniques for the estimation of tangent spaces: Robust PCA and Sparse PCA are some of the options. Other methods for the estimation of the curvature from computational geometry can also be employed in our framework. Moreover, it is possible to extend several graph-based manifold learning algorithms, such as ISOMAP, Locally Linear Embedding and Diffusion Maps through the incorporation of differential geometry concepts in the extraction of intrinsic distance functions. To avoid the out-of-sample problem, we intend to extend the Locality Preserving Projections (LPP) algorithm, which is an attempt to linearize Laplacian Eigenmaps in a way that we can build a projection matrix, similar to what is done in PCA, allowing the direct mapping of novel instances. Our idea consists in studying ways to extend LPP through the definition of the *K-graph*, making it possible to overcome this limitation. Finally, in order to accelerate the method, we intend to implement the algorithm in Julia language, which is notably faster than Python.

References

[1] F. Wang, J. Sun, Survey on distance metric learning and dimensionality reduction in data mining, Data Min. Knowl. Discov. 29 (2) (2015) 534–564.

[2] D. Li, Y. Tian, Survey and experimental study on metric learning methods, Neural Networks 105 (2018) 447–462.

[3] W. Wu, D. Tao, H. Li, Z. Yang, J. Cheng, Deep features for person re-identification on metric learning, Pattern Recognition 110 (2021) 107424.

[4] W. N. A. Jr., T. D. Morley, Eigenvalues of the laplacian of a graph, Linear and Multilinear Algebra 18 (2) (1985) 141–145.

[5] B. Li, Y.-R. Li, X.-L. Zhang, A survey on laplacian eigenmaps based manifold learning methods, Neurocomputing 335 (2019) 336–351.

[6] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Computation 15 (6) (2003) 1373–1396.

[7] M. Belkin, P. Niyogi, Convergence of laplacian eigenmaps, in: B. Schölkopf, J. C. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems 19, MIT Press, 2007, pp. 129–136.

[8] U. von Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (2007) 395–416.

[9] Z. K. Malik, A. Hussain, J. Wu, An online generalized eigenvalue version of laplacian eigenmaps for visual big data, Neurocomputing 173 (2016) 127–136.

[10] B. Chen, Y. Gao, S. Wu, J. Pan, J. Liu, Y. Fan, Soft adaptive loss based laplacian eigenmaps, Applied Intelligence.

[11] L. Torres, K. S. Chan, T. Eliassi-Rad, Glee: Geometric laplacian eigenmap embedding, Journal of Complex Networks 8 (2).

[12] Z. Kang, H. Pan, S. C. H. Hoi, Z. Xu, Robust graph learning from noisy data, IEEE Transactions on Cybernetics 50 (5) (2020) 1833–1843.

[13] M. T. Harandi, M. Salzmann, R. I. Hartley, Dimensionality reduction on SPD manifolds: The emergence of geometry-aware methods, IEEE Trans. Pattern Anal. Mach. Intell. 40 (1) (2018) 48–62.

[14] U. K. Dutta, M. Harandi, C. C. Sekhar, Semi-supervised metric learning: A deep ressurection, in: 35th AAAI Conference on Artificial Intelligence (AAAI-21), 2021, pp. 7279–7287.

[15] L. Hajderanj, D. Chen, I. Weheliye, The impact of supervised manifold learning on structure preserving and classification error: A theoretical study, IEEE Access 9 (2021) 43909–43922.

[16] E. Barshan, A. Ghodsi, Z. Azimifar, M. Zolghadri Jahromi, Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds, Pattern Recognition 44 (7) (2011) 1357–1371.

[17] A. Martinez, A. Kak, Pca versus lda, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (2) (2001) 228–233.

[18] L. van der Maaten, G. Hinton, Visualizing high-dimensional data using t-sne, Journal of Machine Learning Research 9 (2008) 2579–2605.

[19] L. McInnes, J. Healy, N. Saul, L. Großberger, Umap: Uniform manifold approximation and projection, Journal of Open Source Software 3 (29) (2018) 861.

[20] L. McInnes, J. Healy, J. Melville, Umap: Uniform manifold approximation and projection for dimension reduction (2020). `arXiv:1802.03426`.

[21] B. O'Neill, Elementary Differential Geometry, 2nd Edition, Elsevier, 2006.

[22] T. Shifrin, Differential Geometry: A First Course in Curves and Surfaces, University of Georgia, 2016.

[23] M. P. do Carmo, Differential Geometry of Curves and Surfaces, 2nd Edition, Dover Publications Inc., 2017.

[24] F. R. K. Chung, Spectral Graph Theory, American Mathematical Society, 1997.

[25] A. E. Brouwer, W. H. Haemers (Eds.), Spectra of Graphs, Springer, 2011.

[26] D. A. Spielman, Spectral graph theory and its applications, in: 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07), 2007, pp. 29–38.

[27] B. Nica, A Brief Introduction to Spectral Graph Theory, American Mathematical Society, 2018.

[28] P. van Mieghem, Graph Spectra for Complex Networks, Cambridge University Press, 2010.

[29] B. Mohar, The laplacian spectrum of graphs, in: Graph Theory, Combinatorics, and Applications, Wiley, 1991, pp. 871–898.

[30] M. Fiedler, Laplacian of graphs and algebraic connectivity, Banach Center Publications 25 (1) (1989) 57–70.

[31] W. H. Kruskal, W. A. Wallis, Use of ranks in one-criterion variance analysis, Journal of the American Statistical Association 47 (260) (1952) 583–621.

[32] P. Nemenyi, Distribution-free multiple comparisons, Ph.D. thesis, Princeton University (1963).

[33] M. Hollander, D. A. Wolfe, Nonparametric statistical methods, 2nd Edition, Wiley-Interscience, 1999.

Table 1: Silhouette coefficients for clusters produced by PCA, ISOMAP, t-SNE, UMAP, LDA, Supervised PCA and *K-graph* Laplacian for several `openML.org` datasets (2-D case).

Datasets	**PCA**	**ISO**	**TSNE**	**UMAP**	**LDA**	**SPCA**	**KLAP**	**param.**
iris	0.410	0.452	0.494	0.517	0.645	0.413	**0.653**	K=0.75. t=0.1
wine	0.526	0.547	0.556	0.575	0.663	0.568	**0.706**	K=0.75. t=0.1
fl2000	-0.438	-0.483	-0.292	-0.447	-0.192	-0.454	**0.381**	K=0.75. t=1
flags	-0.209	-0.235	-0.150	-0.232	-0.069	-0.153	**0.425**	K=0.75. t=1
vehicle	-0.062	-0.068	-0.051	-0.024	0.258	-0.040	**0.790**	K=0.75. t=1
mfeat-fourier	-0.032	0.010	0.167	0.209	0.251	0.107	**0.422**	K=0.9. t=1
digits	0.054	0.167	0.385	0.550	0.224	0.111	**0.722**	K=0.75. t=0.1
prnn_fglass	-0.085	-0.107	-0.006	-0.066	0.038	-0.007	**0.517**	K=0.75. t=1
micro-mass	-0.078	-0.020	0.190	0.300	0.307	0.003	**0.552**	K=0.95. t=1
diggle_table	0.084	0.125	0.198	0.179	0.318	0.292	**0.478**	K=0.9. t=0.0001
tae	-0.059	-0.072	-0.012	-0.020	-0.004	-0.009	**0.780**	K=0.75. t=1
smartphone	0.062	0.160	0.409	0.456	0.564	0.205	**0.671**	K=0.75. t=1
hayes-roth	0.037	-0.027	-0.024	-0.038	0.067	0.051	**0.689**	K=0.75. t=1
glass	-0.085	-0.107	-0.031	-0.006	0.038	-0.007	**0.494**	K=0.75. t=1
breast-tissue	-0.079	-0.063	-0.030	-0.039	-0.044	-0.043	**0.421**	K=0.75. t=1
seismic-bumps	0.443	0.470	0.531	0.548	0.593	0.452	**0.699**	K=0.75. t=1
user-knowledge	-0.036	-0.042	0.039	-0.037	0.460	0.119	**0.696**	K=0.75. t=1
vertebra_column	0.059	0.059	0.191	0.204	0.205	0.176	**0.716**	K=0.75. t=1
engine1	-0.133	-0.149	-0.155	-0.101	0.025	-0.016	**0.488**	K=0.75. t=1
PopularKids	-0.042	-0.050	-0.045	-0.065	-0.029	-0.042	**0.847**	K=0.75. t=1
heart-h	0.056	0.076	0.025	0.054	0.159	0.120	**0.419**	K=0.75. t=1
anneal	-0.123	-0.120	-0.160	-0.156	0.298	0.251	**0.761**	K=0.9. t=1
ecoli	0.233	0.271	0.316	0.317	0.253	0.223	**0.543**	K=0.75. t=1
thyroid-new	0.588	0.521	0.259	0.316	0.591	0.599	**0.618**	K=0.75. t=1
mammography	0.363	0.279	0.069	-0.215	0.784	0.583	**0.801**	K=0.75. t=1
spambase	0.268	0.251	0.242	0.209	0.505	0.361	**0.513**	K=0.75. t=1
bank-marketing	0.114	0.074	0.007	0.004	0.384	0.256	**0.693**	K=0.75. t=1
usp05	-0.426	-0.427	-0.41	-0.432	-0.016	-0.391	**0.307**	K=0.75. t=1
autoUniv-au6-750	-0.08	-0.072	-0.069	-0.067	-0.103	-0.094	**0.303**	K=0.75. t=1
cars1	0.045	0.075	0.087	0.009	0.128	0.056	**0.736**	K=0.75. t=1
calendarDOW	-0.085	-0.094	-0.106	-0.066	0.079	-0.080	**0.614**	K=0.75. t=1
solar-flare	0.058	-0.041	-0.021	-0.149	0.174	0.097	**0.572**	K=0.75. t=1
touch2	-0.053	-0.04	0.011	-0.004	0.143	-0.026	**0.24**	K=0.75. t=1
heart-long-beach	-0.109	-0.141	-0.093	-0.128	-0.063	-0.088	**0.654**	K=0.75. t=1
heart-switzerland	-0.128	-0.119	-0.165	-0.165	-0.151	-0.158	**0.617**	K=0.75. t=1
teachingAssistant	-0.058	-0.066	-0.052	-0.035	0.026	0.019	**0.793**	K=0.75. t=1
balance-scale	0.103	0.11	0.06	0.029	0.157	0.159	**0.63**	K=0.75. t=1
tic-tac-toe	0.009	0.014	0.006	0.049	0.045	0.033	**0.538**	K=0.75. t=1
mux6	0.015	0.021	0.033	0.004	0.12	0.076	**0.507**	K=0.75. t=1
car-evaluation	-0.136	-0.118	-0.06	-0.107	0.176	0.113	**0.605**	K=0.75. t=1
wine-quality-white	-0.125	-0.126	-0.125	-0.219	-0.099	-0.112	**0.602**	K=0.75. t=1
thyroid-allbp	-0.204	-0.219	-0.159	-0.103	-0.032	-0.069	**0.568**	K=0.75. t=1
waveform-5000	0.231	0.233	0.174	0.189	0.252	0.233	**0.755**	K=0.75. t=1
wine-quality-red	-0.084	-0.081	-0.113	-0.133	-0.089	-0.06	**0.544**	K=0.75. t=1
cmc	-0.047	-0.022	-0.015	-0.027	-0.026	-0.041	**0.688**	K=0.5. t=1
yeast	-0.09	-0.053	-0.042	-0.025	-0.013	-0.054	**0.497**	K=0.75. t=1
analcatdata_dmft	-0.069	-0.083	-0.054	-0.054	-0.076	-0.077	**0.367**	K=0.75. t=1
steel-plates-fault	-0.058	-0.139	-0.114	-0.228	0.021	-0.067	**0.549**	K=0.75. t=1
sleepdata	-0.047	-0.043	-0.059	-0.056	-0.045	-0.047	**0.171**	K=0.75. t=1
diabetes	0.117	0.115	0.086	0.077	0.256	0.168	**0.505**	K=0.75. t=1
Average	0.012	0.012	0.038	0.027	0.163	0.074	**0.577**	
Median	-0.047	-0.042	-0.014	-0.026	0.124	0.026	**0.587**	
Minimum	-0.438	-0.483	-0.41	-0.447	-0.192	-0.454	**0.171**	
Maximum	0.588	0.547	0.556	0.575	0.784	0.599	**0.847**	
Std. Dev.	0.202	0.207	0.200	0.232	0.237	0.215	**0.154**	

Table 2: Maximum classification accuracies for 8 different supervised classifiers after dimensionality reduction based metric learning with PCA, ISOMAP, t-SNE, UMAP, LDA, Supervised PCA and *K-graph* Laplacian for several `openML.org` datasets (2-D case).

Datasets	**PCA**	**ISO**	**TSNE**	**UMAP**	**LDA**	**SPCA**	**KLAP**	**param.**
iris	0.960	0.933	0.973	1.000	0.986	0.960	**1.000**	K=0.75. t=0.1
wine	0.966	0.988	0.988	0.943	1.000	0.988	**1.000**	K=0.75. t=0.1
fl2000	0.676	0.735	0.705	0.676	0.882	0.676	**0.912**	K=0.75. t=1
flags	0.453	0.422	0.464	0.453	0.577	0.546	**0.866**	K=0.75. t=1
vehicle	0.501	0.510	0.699	0.650	0.747	0.539	**1.000**	K=0.75. t=1
mfeat-fourier	0.438	0.447	0.771	0.732	0.708	0.588	**0.892**	K=0.9. t=1
digits	0.531	0.705	0.963	0.959	0.695	0.648	**0.973**	K=0.75. t=0.1
prnn_fglass	0.644	0.654	0.701	0.728	0.644	0.635	**0.888**	K=0.75. t=1
micro-mass	0.672	0.561	0.861	0.861	0.861	0.650	**0.950**	K=0.95. t=1
diggle_table	0.774	0.780	0.974	0.948	0.838	0.883	**0.993**	K=0.9. t=0.0001
tae	0.486	0.592	0.565	0.552	0.631	0.631	**1.000**	K=0.75. t=1
smartphone	0.700	0.766	0.911	0.911	0.933	0.800	**0.988**	K=0.75. t=1
hayes-roth	0.650	0.625	0.650	0.587	0.725	0.712	**1.000**	K=0.75. t=1
glass	0.626	0.588	0.626	0.644	0.588	0.626	**0.906**	K=0.75. t=1
breast-tissue	0.415	0.415	0.415	0.452	0.471	0.471	**0.905**	K=0.75. t=1
seismic-bumps	0.923	0.933	0.942	0.923	0.971	0.923	**1.000**	K=0.75. t=1
user-knowledge	0.569	0.663	0.757	0.737	0.925	0.831	**0.995**	K=0.75. t=1
vertebra_column	0.709	0.735	0.800	0.800	0.838	0.780	**1.000**	K=0.75. t=1
engine1	0.817	0.885	0.921	0.906	0.880	0.864	**1.000**	K=0.75. t=1
PopularKids	0.531	0.493	0.510	0.543	0.556	0.548	**1.000**	K=0.75. t=1
heart-h	0.680	0.700	0.693	0.687	0.700	0.687	**0.871**	K=0.75. t=1
anneal	0.882	0.864	0.942	0.917	0.962	0.917	**0.982**	K=0.9. t=1
ecoli	0.817	0.817	0.890	0.914	0.811	0.835	**0.975**	K=0.75. t=1
thyroid-new	0.972	0.990	0.962	0.981	0.981	0.981	**0.990**	K=0.75. t=1
mammography	0.979	0.976	0.978	0.976	0.981	0.978	**1.000**	K=0.75. t=1
spambase	0.867	0.861	0.865	0.869	0.906	0.893	**1.000**	K=0.75. t=1
bank-marketing	0.892	0.892	0.895	0.892	0.899	0.893	**1.000**	K=0.75. t=1
usp05	0.627	0.617	0.715	0.676	0.715	0.607	**0.892**	K=0.75. t=1
autoUniv-au6-750	0.221	0.2	0.178	0.202	0.256	0.245	**0.618**	K=0.75. t=1
cars1	0.704	0.729	0.734	0.714	0.719	0.698	**1.000**	K=0.75. t=1
calendarDOW	0.585	0.605	0.600	0.505	0.685	0.625	**0.980**	K=0.75. t=1
solar-flare	0.626	0.576	0.601	0.582	0.740	0.670	**0.936**	K=0.75. t=1
touch2	0.458	0.481	0.721	0.676	0.714	0.481	**0.767**	K=0.75. t=1
heart-long-beach	0.23	0.240	0.300	0.330	0.390	0.380	**0.980**	K=0.75. t=1
heart-switzerland	0.387	0.322	0.435	0.403	0.467	0.483	**0.871**	K=0.75. t=1
teachingAssistant	0.473	0.565	0.526	0.618	0.592	0.592	**1.000**	K=0.75. t=1
balance-scale	0.853	0.769	0.811	0.779	0.897	0.897	**0.981**	K=0.75. t=1
tic-tac-toe	0.616	0.616	0.716	0.750	0.645	0.729	**1.000**	K=0.75. t=1
mux6	0.687	0.687	0.781	0.687	0.718	0.750	**1.000**	K=0.75. t=1
car-evaluation	0.699	0.687	0.68	0.671	0.918	0.905	**0.97**	K=0.75. t=1
wine-quality-white	0.536	0.55	0.601	0.556	0.577	0.554	**0.974**	K=0.75. t=1
thyroid-allbp	0.632	0.646	0.677	0.677	0.713	0.705	**0.981**	K=0.75. t=1
waveform-5000	0.87	0.835	0.772	0.797	0.884	0.873	**1.000**	K=0.75. t=1
wine-quality-red	0.532	0.558	0.608	0.552	0.602	0.585	**0.97**	K=0.75. t=1
cmc	0.468	0.442	0.483	0.454	0.533	0.497	**1.000**	K=0.5. t=1
yeast	0.482	0.505	0.591	0.588	0.532	0.547	**0.863**	K=0.75. t=1
analcatdata_dmft	0.218	0.205	0.225	0.21	0.215	0.21	**0.909**	K=0.75. t=1
steel-plates-fault	0.562	0.607	0.73	0.718	0.63	0.608	**0.958**	K=0.75. t=1
sleepdata	0.507	0.494	0.5	0.507	0.498	0.502	**0.875**	K=0.75. t=1
diabetes	0.736	0.726	0.734	0.726	0.804	0.76	**0.992**	K=0.75. t=1
Average	0.637	0.644	0.703	0.692	0.723	0.688	**0.952**	
Median	0.630	0.636	0.716	0.687	0.717	0.673	**0.981**	
Minimum	0.218	0.2	0.178	0.202	0.215	0.21	**0.618**	
Maximum	0.979	0.990	0.988	**1.000**	**1.000**	0.988	**1.000**	
Std. Dev.	0.194	0.196	0.198	0.197	0.189	0.185	**0.072**	