Advances in Data Analysis and Classification (2021) 15:829–868
https://doi.org/10.1007/s11634-020-00434-3

REGULAR ARTICLE

PCA-KL: a parametric dimensionality reduction approach for unsupervised metric learning

Alexandre L. M. Levada[1]

Received: 10 December 2019 / Revised: 29 April 2020 / Accepted: 6 December 2020 /
Published online: 7 January 2021

Abstract
Dimensionality reduction algorithms are powerful mathematical tools for data analysis and visualization. In many pattern recognition applications, a feature extraction step is often required to mitigate the curse of the dimensionality, a collection of negative effects caused by an arbitrary increase in the number of features in classification tasks. Principal Component Analysis (PCA) is a classical statistical method that creates new features based on linear combinations of the original ones through the eigenvectors of the covariance matrix. In this paper, we propose PCA-KL, a parametric dimensionality reduction algorithm for unsupervised metric learning, based on the computation of the entropic covariance matrix, a surrogate for the covariance matrix of the data obtained in terms of the relative entropy between local Gaussian distributions instead of the usual Euclidean distance between the data points. Numerical experiments with several real datasets show that the proposed method is capable of producing better defined clusters and also higher classification accuracy in comparison to regular PCA and several manifold learning algorithms, making PCA-KL a promising alternative for unsupervised metric learning.

Keywords Dimensionality reduction · PCA · KL-divergence · Unsupervised Metric learning

Mathematics Subject Classification 62H30 · 94A16 · 94A17 · 68T10

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

✉ Alexandre L. M. Levada
alexandre.levada@ufscar.br

[1] Computing Department, Federal University of São Carlos, São Carlos, Brazil

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 (Trunk 1979; Lee and Verleysen 2007; Hughes 1968; Zimek et al. 2012; Marimont and Shapiro 1979; Chávez et al. 2001).

One of the major drawbacks in high-dimensional data analysis is the *curse of the dimensionality*. This term was first mentioned by Bellman (1961) 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 (Carreira-Perpinan 1997). In contrast with the usual 3D Euclidean space, the geometric properties of hyperspaces are highly non-intuitive, making the learning of supervised discriminative functions a painful task (Jimenes and Landgrebe 1998). 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, it is a quadratic function of the dimensionality (Fukunaga 1990). 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 (Scott 1992; Hwang et al. 1994). Thus, in order to extract relevant information from high-dimensional data, it is required a large n, which is not always possible under certain circumstances. 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 issue with high-dimensional data is that the usual Euclidean distance as a measure of dissimilarity tends to behave poorly. In this context, unsupervised metric learning methods try to overcome this limitation by finding suitable distance functions. A class of algorithms extremely relevant to this problem is manifold learning. Manifold learning is 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 learns a distance function that geometrically is better suited to represent a similarity measure between a pair of objects in the collection (Li and Tian 2018; Wang and Sun 2015; Yang and Jin 2006; Bellet et al. 2013; Suárez 2018).

The key idea of dimension reduction is to find the most compact low dimensional structure that is embedded in a higher dimensional space. Historically, Occam's razor has been used to justify dimension reduction (Domingos 1999). The basic concept in Occam's razor is to choose the simplest model from a set of equivalent models to explain a given phenomenon (Huo et al. 2008). There are many approaches to dimensionality reduction based on several assumptions and used in a variety of contexts. In this paper, we propose a parametric PCA algorithm based on a information-theoretic measure: the relative entropy. The main goal is to find a surrogate for the covariance matrix replacing the Euclidean distance in the feature space by the KL-divergence between Gaussian distributions estimated in each local neighborhood. One possible limitation of PCA is that this method maximizes the variance of the retained data, which often produces clusters with large scattering. This can be a negative side-effect to many classification problems. In summary, the main contribution of the proposed method is that unlike traditional dimensionality reduction methods, PCA-KL is a patch-based

method, which means it is less sensitive to the presence of noise and outliers in data, making the clusters more compact, in the sense that their intra-class scatter is reduced. As a consequence, in several different datasets, the features extracted by PCA-KL show more discriminant power in comparison to the features obtained by some manifold learning algorithms, making the proposed method a promising alternative for unsupervised metric learning.

The remainder of the paper is organized as follows: Sect. 2 describes the classic dimensionality reduction methods PCA, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE. In Sect. 3, we briefly discuss the KL-divergence and its computation in the Gaussian case. In Sect. 4, we describe the proposed PCA-KL method in details. Section 5 shows the experiments and results. Finally, in Sect. 6 we present the conclusions, final remarks and future directions for research in dimensionality reduction for unsupervised metric learning.

2 Dimensionality reduction for unsupervised metric learning

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 (Hotelling 1933), a classical multivariate statistical technique that expands a given random vector $\mathbf{x} \in R^m$ in the eigenvectors of its covariance matrix (Jolliffe 2002; Shlens 2005). 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 (Hastie et al. 2009). Since it depends 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 $\mathbf{x} \in R^m$, 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.

2.1.1 PCA by the maximization of the variance

Let $Z = [T^T, S^T]$ be an orthonormal basis for R^m in which $T^T = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_d]$ denotes the $d < m$ components that we wish to retain during the dimensionality reduction process and $S^T = [\mathbf{w}_{d+1}, \mathbf{w}_{d+2}, \ldots, \mathbf{w}_m]$ are the remaining 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 (Young and Calvert 1974).

The problem in question can be summarized as: given an input feature space, we want to find d directions $\mathbf{w}_j$, for $j = 1, 2, \ldots, d$ 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 $\mathbf{w}_j$? Without loss of generality, we assume that the sample $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ has zero mean, that is, the data points are centred around the origin.

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

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

where c_j are the coefficients of the expansion.

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

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

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

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

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 (Hyvarinen et al. 2001):

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

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

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

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

Hence, we have the following constrained optimization problem:

$$\arg\max_{\mathbf{w}_j} \sum_{j=1}^{d} \mathbf{w}_j^T \Sigma_x \mathbf{w}_j \quad \text{subject to} \quad \|\mathbf{w}_j\| = 1 \quad \text{for} \quad j = 1, 2, \ldots, d \tag{6}$$

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

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

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

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

which leads to the eigenvector equation:

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

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

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

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. Algorithm 1 describes dimensionality reduction by PCA.

Algorithm 1 Principal Component Analysis

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

The great advantage of PCA is that it is a very fast method. It has been verified that the time complexity of PCA is $O(max(n^2m, nm^2))$, where n is the number of samples and m is the number of input features (Nguyen and Holmes 2019).

2.2 Non-negative matrix factorization

Non-Negative Matrix Factorization (NMF) is applied to dimensionality reduction of multivariate data by considering as input a set of of multivariate m-dimensional data vectors, $V_{m \times n}$ where n is the number of examples in the data set. This matrix is then approximately factorized into an $m \times r$ matrix W and an $r \times n$ matrix H, that is, $X \approx WH$. Usually r is chosen to be smaller than n or m, so that W and H are smaller than the original matrix X. This results in a compressed version of the original data matrix (Lee and Seung 1999). The significance of this representation is that $\mathbf{v} \approx W\mathbf{h}$, that is each vector is approximated by a linear combination of the columns of W. In

this context, H plays the role of the extracted features, that is, each column $\mathbf{h}_j$ of H represents a vector in the transformed space.

To measure how close V is from WH, a cost function is usually adopted. Two common choices of distance measures are the Euclidean distance and the Kullback–Leibler divergence, given by:

$$D_E(A, B) = \|A - B\|^2 = \sum_{i,j}(A_{i,j} - B_{i,j})^2 \tag{11}$$

$$D_{KL}(A, B) = \sum_{i,j}\left[A_{i,j}\, log\left(\frac{A_{i,j}}{B_{i,j}}\right) - A_{i,j} + B_{i,j}\right] \tag{12}$$

Theorem 1 *The Euclidean distance $D_E(V, WH) = \|V - WH\|^2$ is non-increasing for a finite number of steps of the multiplicative rules:*

$$H_{i,j}^{k+1} = H_{i,j}^{k}\frac{((W^k)^T V)_{i,j}}{((W^k)^T W^k H^k)_{i,j}} \tag{13}$$

$$W_{i,j}^{k+1} = W_{i,j}^{k}\frac{(V(H^{k+1})^T)_{i,j}}{(W^k H^{k+1}(H^{k+1})^T)_{i,j}} \tag{14}$$

where k is iteration counter and W^0 and H^0 are initialized with positive random numbers, typically sampled from a $[0, 1]$ *uniform distribution.*

Theorem 2 *The KL-divergence $D_{KL}(V, WH)$ is non-increasing for a finite number of steps of the multiplicative rules:*

$$W_{i,j}^{k+1} = W_{i,j}^{k}\frac{\sum_r H_{jr}^k \frac{V_{ir}}{(W^k H^k)_{ir}}}{\sum_r H_{jr}^k} \tag{15}$$

$$H_{i,j}^{k+1} = H_{i,j}^{k}\frac{\sum_r W_{ri}^{k+1} \frac{V_{rj}}{(W^{k+1} H^k)_{rj}}}{\sum_r W_{ri}^{k+1}} \tag{16}$$

where k is iteration counter and W^0 and H^0 are initialized with positive random numbers, typically sampled from a $[0, 1]$ *uniform distribution.*

The complete proofs for Theorems 1 and 2 are described in details in Lee and Seung's seminal paper (Lee and Seung 1999). The computational complexity of NMF has been shown to be $O(nmd)$ times the number of iterations for convergence, where n is the sample size, m is the number of features and d is the output dimensionality (Lin 2007).

2.2.1 The clustering property

It has been shown that NMF has an intrinsic clustering property, that is, it automatically clusters the columns of the data matrix $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in R^m$ (Ding et al. 2005). More precisely, the approximation of V as WH is achieved by minimizing the error function:

$$\min_{W,H} ||V - WH||^2 \tag{17}$$

subject to $W \geq 0$ and $V \geq 0$. By imposing the orthogonality constraint $HH^T = I$, the NMF minimization problem is mathematically equivalent to the minimization of K-means clustering (Ding et al. 2005). When the error function to be minimized is the KL-divergence, NMF is identical to the PLSA (Probabilistic latent semantic analysis), another popular clustering method (Ding et al. 2008).

2.3 Kernel PCA

Principal Component Analysis only allows linear dimensionality reduction. However, if the data has more complicated structures which are non-linear functions of the original features, standard PCA will fail in capturing meaningful information. Fortunately, kernel PCA allows us to generalize standard PCA to non-linear dimensionality reduction (Schölkopf et al. 1999). The first assumption is that the mean of the data after the mapping to the high-dimensional space is zero, that is:

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

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

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

and the eigenvectors of C are:

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

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

Theorem 3 *The eigenvectors of C can be expressed as a linear combination of the features, that is:*

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

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

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

which implies in:

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

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

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

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

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

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

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

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

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

Using the kernel trick once again, we have:

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

Using the matrix vector notation we can express the equation as (Schölkopf et al. 1999):

$$K^2\alpha_k = (\lambda_k n)K\alpha_k \tag{29}$$

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

$$K\alpha_k = (\lambda_k n)\alpha_k \tag{30}$$

showing that the α_k are the eigenvectors of the kernel matrix. For a new point $\mathbf{x}$, its projection onto the k-th principal component is given by:

$$y_k(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{v}_k = \sum_{i=1}^{n} \alpha_{ki} \phi(\mathbf{x})^T \phi(\mathbf{x}_i) = \sum_{i=1}^{n} \alpha_{ki} K(\mathbf{x}, \mathbf{x}_i) \tag{31}$$

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

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^d \tag{32}$$

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

$$K(\mathbf{x}, \mathbf{y}) = exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) \tag{33}$$

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

$$\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \frac{1}{n} \sum_{k=1}^{n} \phi(\mathbf{x}_k) \tag{34}$$

Hence, the corresponding kernel matrix is given by:

$$\begin{aligned}
\tilde{K}(\mathbf{x}_i, \mathbf{x}_j) = \tilde{\phi}(\mathbf{x}_i)^T \tilde{\phi}(\mathbf{x}_i) &= \left(\phi(\mathbf{x}_i) - \frac{1}{n} \sum_{k=1}^{n} \phi(\mathbf{x}_k)\right)^T \left(\phi(\mathbf{x}_j) - \frac{1}{n} \sum_{k=1}^{n} \phi(\mathbf{x}_k)\right) \\
&= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) - \frac{1}{n} \sum_{k=1}^{n} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_k) - \frac{1}{n} \sum_{k=1}^{n} \phi(\mathbf{x}_k)^T \phi(\mathbf{x}_j) \\
&\quad + \frac{1}{n^2} \sum_{k=1}^{n} \sum_{l=1}^{n} \phi(\mathbf{x}_k)^T \phi(\mathbf{x}_l) \\
&= K(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{n} \sum_{k=1}^{n} K(\mathbf{x}_i, \mathbf{x}_k) - \frac{1}{n} \sum_{k=1}^{n} K(\mathbf{x}_k, \mathbf{x}_j) \\
&\quad + \frac{1}{n^2} \sum_{k=1}^{n} \sum_{l=1}^{n} K(\mathbf{x}_k, \mathbf{x}_l)
\end{aligned} \tag{35}$$

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

$$\tilde{K} = K - 1_n K - K 1_n + 1_n K 1_n \tag{36}$$

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: $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$
3: Compute the Gram matrix $\tilde{K}$ using equation (36)
4: Use equation (30) to solve for the vectors α_k (using $\tilde{K}$ instead of K)
5: Compute the kernel principal components $y_k(\mathbf{x})$ using equation (31) for $k = 1, 2, \ldots, d$
6: **return** Y
7: **end function**

It has been shown that the computational complexity of the original Kernel PCA is $O(n^3)$, where n denotes the number of samples. However, faster algorithms can perform KPCA in $O(dn^2)$, such as Fast Iterative Kernel PCA (Günter et al. 2007).

2.4 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 (Cox and Cox 2001; Borg and Groenen 2005) (MSD)—computational efficiency, global optimality, and asymptotic convergence guarantees - with the flexibility to learn a broad class of non-linear manifolds (Tenenbaum et al. 2000). 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 Euclidean 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 approximations for the true geodesic distances in the manifold.

The ISOMAP algorithm can be divided in three main steps:

1. From the input data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in R^m$ build an undirected proximity graph using the KNN rule or the ϵ-neighborhood rule (von Luxburg 2007);
2. Compute the pairwise distance matrix D using n executions of the Dijkstra's algorithm or one execution of the Floyd–Warshall algorithm (Cormen et al. 2009);
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.

2.4.1 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 $\mathbf{x}_r \in R^d$ for $r = 1, 2, \ldots, n$ in an Euclidean subspace, where d, the target dimensionality, is a parameter of the algorithm (Cox and Cox 2001; Borg and Groenen 2005).

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

$$d_{rs}^2 = \|\mathbf{x}_r - \mathbf{x}_s\|^2 = (\mathbf{x}_r - \mathbf{x}_s)^T(\mathbf{x}_r - \mathbf{x}_s) \tag{37}$$

Let B denote the inner products matrix, that is $B = \{b_{rs}\}$, where $b_{rs} = \mathbf{x}_r^T \mathbf{x}_s$. To find the embedding, MDS needs the matrix B, not D. First, we need to assume a hypothesis that the data has zero mean, 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 Eq. (37), applying the distributive law we have:

$$d_{rs}^2 = \mathbf{x}_r^T \mathbf{x}_r + \mathbf{x}_s^T \mathbf{x}_s - 2\mathbf{x}_r^T \mathbf{x}_s \tag{38}$$

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

$$\frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 = \frac{1}{n}\sum_{r=1}^{n} \mathbf{x}_r^T \mathbf{x}_r + \mathbf{x}_s^T \mathbf{x}_s \tag{39}$$

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

$$\frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 = \mathbf{x}_r^T \mathbf{x}_r + \frac{1}{n}\sum_{s=1}^{n} \mathbf{x}_s^T \mathbf{x}_s \tag{40}$$

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

$$\frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2 = \frac{2}{n}\sum_{r=1}^{n} \mathbf{x}_r^T \mathbf{x}_r \tag{41}$$

Note that from Eq. (38), it is possible to define b_{rs} as:

$$b_{rs} = \mathbf{x}_r^T \mathbf{x}_s = -\frac{1}{2}(d_{rs}^2 - \mathbf{x}_r^T \mathbf{x}_r - \mathbf{x}_s^T \mathbf{x}_s) \tag{42}$$

Combining Eqs. (39), (40) and (41) we have:

$$b_{rs} = -\frac{1}{2}\left(d_{rs}^2 - \frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 - \frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2\right) \tag{43}$$

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

$$a_{r\cdot} = \frac{1}{n}\sum_{s=1}^{n} a_{rs} \qquad a_{\cdot s} = \frac{1}{n}\sum_{r=1}^{n} a_{rs} \qquad a_{\cdot\cdot} = \frac{1}{n}\sum_{r=1}^{n}\sum_{s=1}^{n} a_{rs} \tag{44}$$

leading to:

$$b_{rs} = a_{rs} - a_{r\cdot} - a_{\cdot s} + a_{\cdot\cdot} \tag{45}$$

which in matrix notation becomes $B = HAH$, where:

$$H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T \tag{46}$$

is the centring matrix. To find the embedding, that is, the coordinates of the points in R^d, we have to perform an eigendecomposition of the matrix B, that is:

$$B = V \Lambda V^T \tag{47}$$

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

$$\tilde{V} = \begin{bmatrix} | & | & \ldots & \ldots & | \\ | & | & \ldots & \ldots & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \ldots & \ldots & \mathbf{v}_d \\ | & | & \ldots & \ldots & | \\ | & | & \ldots & \ldots & | \end{bmatrix}_{n\times d} \tag{48}$$

$$\tilde{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_d) \tag{49}$$

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

It has been shown that the overall complexity of the ISOMAP algorithm is given by $O(n^2(m + log\ n))$ (Nguyen and Holmes 2019), where n and m denote the number of samples and number of features, respectively.

2.5 Locally linear embedding

The ISOMAP algorithm is a global method in the sense that to find the coordinates of a given input vector $\mathbf{x}_i \in R^m$ in the manifold, it uses information from all the samples through the matrix B. On the other hand, Locally Linear Embedding (LLE), as the name emphasizes, is a local method, that is, the new coordinates of any $\mathbf{x}_i \in R^m$ 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 $\mathbf{x}_i$ and its neighbors define a linear patch, that is, they all belong to an Euclidean subspace (Roweis and Saul 2000).

Basically, the LLE algorithm require as inputs an $n \times m$ data matrix X, with rows $\mathbf{x}_i$, 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 $\mathbf{y}_i$. The LLE algorithm can be divided in three main steps (Roweis and Saul 2000; Saul and Roweis 2003):

1. From each $\mathbf{x}_i \in R^m$ find its k nearest neighbors;
2. Find the weight matrix W which minimizes the reconstruction error for each data point $\mathbf{x}_i \in R^m$;
3. Find the coordinates Y which minimize the reconstruction error using the optimum weights;

2.5.1 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 $\mathbf{x}_i$ as:

$$E(\mathbf{w}) = \left\| \sum_j w_j (\mathbf{x}_i - \mathbf{x}_j) \right\|^2 = \sum_j \sum_k w_j w_k (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_k)^T \tag{50}$$

Defining the matrix C as:

$$C_{jk} = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_k) \tag{51}$$

we have the following expression for the local reconstruction error:

$$E(\mathbf{w}) = \sum_j \sum_k w_j C_{jk} w_k = \mathbf{w}^T C \mathbf{w} \tag{52}$$

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 $\mathbf{x}_i$ separately, which drastically simplifies the computations. Thus, we have n independent constrained optimization problems given by:

$$\arg\min_{\mathbf{w}_i} \ \mathbf{w}_i^T C_i \mathbf{w}_i \qquad \text{subject to} \quad \mathbf{1}^T \mathbf{w}_i = 1 \quad \text{for} \quad i = 1, 2, \ldots, n \tag{53}$$

Using Lagrange multipliers, we write the Lagrangian function as:

$$L(\mathbf{w}_i, \lambda) = \mathbf{w}_i^T C_i \mathbf{w}_i - \lambda(\mathbf{1}^T \mathbf{w}_i - 1) \tag{54}$$

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

$$\frac{\partial}{\partial \mathbf{w}_i} L(\mathbf{w}_i, \lambda) = 2C_i \mathbf{w}_i - \lambda \mathbf{1} = 0 \tag{55}$$

which leads to

$$C_i \mathbf{w}_i = \frac{\lambda}{2}\mathbf{1} \tag{56}$$

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

$$C_i \mathbf{w}_i = \mathbf{1} \tag{57}$$

and then normalize the solution to guarantee that $\sum_j w_i(j) = 1$.

2.5.2 Finding the coordinates

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:

$$\Phi(Y) = \sum_{i=1}^{n} \left\| \mathbf{y}_i - \sum_j w_{ij}\mathbf{y}_j \right\|^2 \tag{58}$$

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 transformed data is the identity matrix, that is, there is not correlation between the components of $\mathbf{y} \in R^d$;

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

$$\Phi(Y) = Tr(Y^T(I - W)^T(I - W)Y) \tag{59}$$

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

$$M = (I - W)^T(I - W) \tag{60}$$

we get the following optimization problem:

$$\arg\min_Y \; Tr(Y^T M Y) \qquad \text{subject to} \quad \frac{1}{n}Y^T Y = I \tag{61}$$

Thus, the Lagrangian function is given by:

$$L(Y, \lambda) = Tr(Y^T M Y) - \lambda\left(\frac{1}{n}Y^T Y - I\right) \tag{62}$$

Differentiating the function and setting the result to zero gives:

$$2MY - 2\frac{\lambda}{n}Y = 0 \tag{63}$$

$$MY = \beta Y \tag{64}$$

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 $\mathbf{1}$. This bottom eigenvector corresponds to the mean of Y and should be discarded to enforce the constraint that $\sum_{i=1}^{n} \mathbf{y}_i = 0$ (de Ridder and Duin 2002). Algorithm 4 shows a summary of the LLE method.

Algorithm 4 Locally Linear Embedding

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

$$C_i(j,k) = (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_k)^T \tag{65}$$

    Solve the linear system $C_i \mathbf{w}_i = \mathbf{1}$ to estimate the weights $\mathbf{w}_i \in R^K$.
    Normalize the weights in $\mathbf{w}_i$ so that $\sum_j \mathbf{w}_i(j) = 1$.
  end for
  Construct the $n \times n$ matrix W, whose lines are the estimated $\mathbf{w}_i$.
  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:

$$Y = \begin{bmatrix} | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \dots & \mathbf{v}_d \\ | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \end{bmatrix}_{n \times d} \tag{66}$$

  return Y
end function

It has been shown that the overall complexity of the LLE algorithm is given by $O((m\ log\ k)(n\ log\ n)) + O(mnk^3) + O(dn^2)$ (Roweis and Saul 2000), where n, m, k and d denote the number of samples, number of features, number of neighbors in the KNN graph and the output dimensionality, respectively.

2.6 Laplacian Eigenmaps

Basically, the Laplacian Eigenmaps algorithm require as inputs an $n \times m$ data matrix X, with each row $\mathbf{x}_i$ 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, with rows $\mathbf{y}_i$. The algorithm can be divided in three main steps (Belkin and Niyogi 2003):

1. Construct the neighborhood graph $G = (V, E)$ by linking nodes v_i and v_j if $\mathbf{x}_i$ and $\mathbf{x}_j$ are close. The two variants are:
   - ϵ-neighborhood: connect v_i and v_j by an edge if $\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 \leq \epsilon$.
   - 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 and v_j are connected, make

$$W_{ij} = exp\left\{-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{t}\right\} \tag{67}$$

     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.

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

$$J(Y) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\left\|\mathbf{y}^{(i)} - \mathbf{y}^{(j)}\right\|^2 \tag{68}$$

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

$$J(Y) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}(\mathbf{y}^{(i)} - \mathbf{y}^{(j)})(\mathbf{y}^{(i)} - \mathbf{y}^{(j)})^T \tag{69}$$

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

$$\begin{aligned} J(Y) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[W_{ij}\mathbf{y}^{(i)}\mathbf{y}^{(i)^T} - W_{ij}\mathbf{y}^{(i)}\mathbf{y}^{(j)^T} - W_{ij}\mathbf{y}^{(j)}\mathbf{y}^{(i)^T} + W_{ij}\mathbf{y}^{(j)}\mathbf{y}^{(j)^T}\right] \\ &= \frac{1}{2}\left[\sum_{i=1}^{n} d_i\mathbf{y}^{(i)}\mathbf{y}^{(i)^T} - 2\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\mathbf{y}^{(i)}\mathbf{y}^{(j)^T} + \sum_{j=1}^{n} d_j\mathbf{y}^{(j)}\mathbf{y}^{(j)^T}\right] \\ &= \frac{1}{2}\left[2\sum_{i=1}^{n} d_i\mathbf{y}^{(i)}\mathbf{y}^{(i)^T} - 2\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\mathbf{y}^{(i)}\mathbf{y}^{(j)^T}\right] \\ &= \sum_{i=1}^{n} d_i\mathbf{y}^{(i)}\mathbf{y}^{(i)^T} - \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\mathbf{y}^{(i)}\mathbf{y}^{(j)^T} \end{aligned} \tag{70}$$

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:

$$J(Y) = Tr(DYY^T) - Tr(WYY^T) \tag{71}$$

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

$$\begin{aligned} J(Y) &= Tr(Y^T DY) - Tr(Y^T WY) = Tr(Y^T(DY - WY)) \\ &= Tr(Y^T(D - W)Y) = Tr(Y^T LY) \end{aligned} \tag{72}$$

Thus, we have the following constrained optimization problem:

$$\arg\min_{Y} \; Tr(Y^T LY) \qquad \text{subject to} \quad Y^T DY = I \tag{73}$$

whose Lagrangian function is given by:

$$L(Y, \lambda) = Tr(Y^T LY) - \lambda(Y^T DY - I) \tag{74}$$

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

$$\frac{\partial}{\partial Y}L(Y, \lambda) = 2LY - 2\lambda DY = 0 \tag{75}$$

leading to the following eigenvector problem:

$$LY = \lambda DY \tag{76}$$

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 $D^{-1}L$. Algorithm 5 shows a summary of the Laplacian Eigenmaps method.

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.

$$W_{ij} = exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right\} \quad \text{if} \quad v_j \in N(v_i) \tag{77}$$

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

$$d_i = \sum_{j=1}^{n} W_{ij} \tag{78}$$

5: Compute the Laplacian matrix $L = D - W$
6: Select the bottom d eigenvectors with non-zero eigenvalues of $D^{-1}L$:

$$Y = \begin{bmatrix} | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \dots & \mathbf{v}_d \\ | & | & \dots & \dots & | \\ | & | & \dots & \dots & | \end{bmatrix}_{n\times d} \tag{79}$$

7: **return** Y
8: **end function**

It has been shown that the overall complexity of the Laplacian Eigenmaps algorithm is given by $O((m\ log\ k)(n\ log\ n)) + O(mnk^3) + O(dn^2)$ (Belkin and Niyogi 2003), where n, m, k and d denote the number of samples, number of features, number of neighbors in the KNN graph and the output dimensionality, respectively.

2.7 t-Distributed stochastic neighbor embedding

The algorithm t-SNE is based on its predecessor Stochastic Neighbor Embedding, or simply SNE (Hinton and Roweis 2003). Basically, SNE converts Euclidean distances between samples $\mathbf{x}_i \in R^m$ for $i = 1, 2, \dots, n$ into conditional probabilities. The similarity between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is the conditional probability $p_{j|i}$ that $\mathbf{x}_i$ would choose $\mathbf{x}_j$ as a neighbor if neighbor selection is done in proportion to a probability density under a Gaussian centered at $\mathbf{x}_i$ (van der Maaten and Hinton 2008):

$$p_{j|i} = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2\right)}{\sum_{k\neq i} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2\right)} \tag{80}$$

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

measure is given by:

$$q_{j|i} = \frac{\exp\left(-\left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\mathbf{y}_i - \mathbf{y}_k\right\|^2\right)} \tag{81}$$

Note that, $p_{i|i} = q_{i|i} = 0$. 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 samples using gradient descent. Thus, we have to minimize (Hinton and Roweis 2003):

$$C = \sum_{i=1}^{n} KL(P_i||Q_i) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{82}$$

where P_i represents the conditional probability distribution over all samples given $\mathbf{x}_i$ and Q_i represents the conditional probability distribution over all other map points given map point $\mathbf{y}_i$. The SNE cost function is proposed so that we retain the local structure of the data in the map.

2.7.1 Defining the variance of the Gaussians

It is not suitable to assume that there is a single value of σ_i^2 that is optimal for samples. In dense regions, a smaller value is more appropriate than in sparse regions. The perplexity measure is then defined as (van der Maaten and Hinton 2008):

$$\text{Perp}(P_i) = 2^{H(P_i)} \tag{83}$$

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

$$H(P_i) = -\sum_{j=1}^{n} p_{j|i} log_2 p_{j|i} \tag{84}$$

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 (van der Maaten and Hinton 2008). The minimization of the objective function (KL divergence) is performed by a gradient descent approach with momentum to speed up convergence:

$$\mathscr{Y}^{(t)} = \mathscr{Y}^{(t-1)} - \eta \frac{\partial C}{\partial \mathscr{Y}} + \alpha(t)\left(\mathscr{Y}^{(t-1)} - \mathscr{Y}^{(t-2)}\right) \tag{85}$$

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

2.7.2 Computing the gradient in t-SNE

At each stage of the optimization process, we have a set of points $\mathbf{y}_1.\mathbf{y}_2, \ldots, \mathbf{y}_n$ and the computation of the gradient is required. In the SNE algorithms, the whole process is summarized by Melville (2015):

- Create a matrix where the element d_{ij} represents the Euclidean distance between $\mathbf{y}_i$ and $\mathbf{y}_j$;
- Transform the distances to create f_{ij} (squaring the distances);
- Apply a weighting function to define a weight w_{ij}, in a way that the larger the weight, the smaller the distance;
- Convert the weights into probabilities $q_{ij} = q_{j|i}$, by normalizing over their sum;

With the probabilities at hand, we can compute the gradient. Basically, the cost function employed by t-SNE has two different aspects:

1. it uses a symmetrized version of the SNE cost function (Cook et al. 2007).
2. it uses a Student t distribution to compute the similarity between two points in the low-dimensional space.

In the symmetric version, we minimize a single KL divergence:

$$C = KL(P||Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}\log\frac{p_{ij}}{q_{ij}} \tag{86}$$

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

Moreover, the pairwise similarities in the high dimensional space are given by:

$$p_{ij} = \frac{\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2/2\sigma^2\right)}{\sum_{k\neq l}\exp\left(-\left\|\mathbf{x}_k - \mathbf{x}_l\right\|^2/2\sigma^2\right)} \tag{87}$$

where the normalization constant involves all pairs of points.

By using the Student's t distribution with one degree of freedom, the probabilities q_{ij} are defined by:

$$q_{ij} = \frac{\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}}{\sum_{k\neq l}\left(1 + \left\|\mathbf{y}_k - \mathbf{y}_l\right\|^2\right)^{-1}} = \frac{w_{ij}^{-1}}{\sum_{k\neq l} w_{kl}^{-1}} = \frac{w_{ij}^{-1}}{Z} \tag{88}$$

and the objective function is:

$$\begin{aligned} C &= \sum_{k=1}^{n}\sum_{l=1}^{n}\left(p_{kl}\log\, p_{kl} - p_{kl}\log\, q_{kl}\right) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n}\left(p_{kl}\log\, p_{kl} - p_{kl}\log\, w_{kl}^{-1} + p_{kl}\log\, Z\right) \end{aligned} \tag{89}$$

Differentiating with respect to $\mathbf{y}_i$ leads to:

$$\frac{\partial C}{\partial \mathbf{y}_i} = -\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \mathbf{y}_i}\log w_{kl}^{-1} + \sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \mathbf{y}_i}\log Z \tag{90}$$

For the first term of the gradient, the derivative is non zero if $\forall j, k = i$ or $l = i$. Also, as $p_{ij} = p_{ji}$ and $w_{ij} = w_{ji}$:

$$\begin{aligned} -\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \mathbf{y}_i}\log w_{kl}^{-1} &= -\sum_{j=1}^{n}\left(p_{ij}\frac{\partial}{\partial \mathbf{y}_i}\log w_{ij}^{-1} + p_{ji}\frac{\partial}{\partial \mathbf{y}_i}\log w_{ji}^{-1}\right) \\ &= -2\sum_{j=1}^{n} p_{ij}\frac{\partial}{\partial \mathbf{y}_i}\log w_{ij}^{-1} \end{aligned} \tag{91}$$

The derivative of the inverse of the weight is:

$$\frac{\partial w_{ij}^{-1}}{\partial \mathbf{y}_i} = -2w_{ij}^{-2}\left(\mathbf{y}_i - \mathbf{y}_j\right) \tag{92}$$

so the first term of the gradient becomes:

$$4\sum_{j=1}^{n} p_{ij} w_{ij}^{-1}\left(\mathbf{y}_i - \mathbf{y}_j\right) \tag{93}$$

In the differentiation of the second term, note that Z does not depend on k or l and the sum of the probabilities is equal to one:

$$\begin{aligned} \sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl}\frac{\partial}{\partial \mathbf{y}_i}\log Z &= \frac{\partial}{\partial \mathbf{y}_i}\log Z\sum_{k=1}^{n}\sum_{l=1}^{n} p_{kl} = \frac{\partial}{\partial \mathbf{y}_i}\log Z \\ &= \frac{1}{Z}\frac{\partial}{\partial \mathbf{y}_i}Z = \frac{1}{Z}\sum_{k=1}^{n}\sum_{l=1}^{n}\frac{\partial}{\partial \mathbf{y}_i} w_{kl}^{-1} \end{aligned} \tag{94}$$

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

$$\frac{1}{Z}\sum_{k=1}^{n}\sum_{l=1}^{n}\frac{\partial}{\partial \mathbf{y}_i} w_{kl}^{-1} = \frac{1}{Z}\sum_{j=1}^{n}\left(\frac{\partial}{\partial \mathbf{y}_i} w_{ij}^{-1} + \frac{\partial}{\partial \mathbf{y}_i} w_{ji}^{-1}\right) \tag{95}$$

Due to symmetry, $w_{ij} = w_{ji}$, leading to:

$$\frac{1}{Z}\sum_{j=1}^{n}\left(\frac{\partial}{\partial \mathbf{y}_i} w_{ij}^{-1} + \frac{\partial}{\partial \mathbf{y}_i} w_{ji}^{-1}\right) = 2\sum_{j=1}^{n}\frac{1}{Z}\frac{\partial}{\partial \mathbf{y}_i} w_{ij}^{-1} \tag{96}$$

From Eq. (92), we can write:

$$2\sum_{j=1}^{n}\frac{1}{Z}\frac{\partial}{\partial \mathbf{y}_i}w_{ij}^{-1} = -4\sum_{j=1}^{n}\frac{w_{ij}^{-1}}{Z}w_{ij}^{-1}\left(\mathbf{y}_i-\mathbf{y}_j\right) = -4\sum_{j=1}^{n}q_{ij}w_{ij}^{-1}\left(\mathbf{y}_i-\mathbf{y}_j\right) \quad (97)$$

Finally, by combining Eqs. (93) and (97), we have an expression for the gradient in the t-SNE iteration:

$$\begin{aligned}\frac{\partial C}{\partial \mathbf{y}_i} &= 4\sum_{j=1}^{n}\left(p_{ij}-q_{ij}\right)w_{ij}^{-1}\left(\mathbf{y}_i-\mathbf{y}_j\right)\\ &= 4\sum_{j=1}^{n}\left(p_{ij}-q_{ij}\right)\left(1+\left\|\mathbf{y}_i-\mathbf{y}_j\right\|^2\right)^{-1}\left(\mathbf{y}_i-\mathbf{y}_j\right)\end{aligned} \quad (98)$$

Algorithm 6 shows the main steps of t-SNE. The parameters of the algorithm are the input data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, the number of iterations T, the perplexity *Perp* (a measure of the effective number of neighbors), the learning rate η and the momentum $\alpha(t)$, used in the gradient descent optimization method.

Algorithm 6 t-distributed Stochastic Neighbor Embedding

1: **function** T- SNE(X, $Perp$, T, η, $\alpha(t)$)
2: Compute pairwise probabilities $p_{j|i}$ and $p_{i|j}$ using equation (80).
3: Set the value of p_{ij} as

$$p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n} \quad (99)$$

4: Sample initial solution $\mathscr{Y}^{(0)} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n\}$ from $\mathcal{N}(0, 10^{-4}I)$.
5: **for** t = 1 to T **do**
6: Compute low dimensional affinities q_{ij} using equation (88).
7: Compute the gradient using equation (98).
8: Update the coordinates with gradient descent with momentum:

$$\mathscr{Y}^{(t)} = \mathscr{Y}^{(t-1)} - \eta\frac{\partial C}{\partial \mathscr{Y}} + \alpha(t)\left(\mathscr{Y}^{(t-1)} - \mathscr{Y}^{(t-2)}\right) \quad (100)$$

9: **end for**
10: **return** $\mathscr{Y}^{(T)}$
11: **end function**

It has been shown that the complexity of the t-SNE algorithm is $O(n^2m + n^2h)$ (Nguyen and Holmes 2019), where n, m and h denote the number of samples, the number of features and the number of iterations, respectively. The t-SNE algorithm is considered as the state-of-the-art in dimensionality reduction for data visualization. Despite its remarkable performance, t-SNE also has some limitations: (1) SNE and t-SNE approaches do not preserve long-range interactions between data points and

generate visualizations in which the arrangement of non-neighboring groups of observations is not informative (Nguyen and Holmes 2019); (2) the high computational cost, especially due to the numerical optimization of its objective function makes it prohibitive for larger datasets (it does not have a closed-form solution); (3) since the algorithm is stochastic, several initializations using different seeds can produce distinct embeddings.

3 Kullback–Leibler divergence

In pattern recognition and machine learning, the problem of quantifying the similarity between different objects or clusters is a challenging task, especially in cases where the standard Euclidean distance is not a reasonable choice. Many works on feature selection adopt statistical divergences to choose the set of features that maximize some measure of separation between classes. Part of their success comes from the fact that most dissimilarity measures are related to distance metrics. In this context, entropy and related divergences provide a fruitful mathematical background for metric learning.

We begin by introducing the entropy of a random variable x as the expected value of the self-information:

$$H(p) = -\int p(x)[log\ p(x)]dx = -E\left[log\ p(x)\right] \tag{101}$$

where $p(x)$ is the probability density function (pdf) of x. Assuming x is normally distributed as $N(\mu, \sigma^2)$, its entropy is given by:

$$H(p) = \frac{1}{2} log\ (2\pi\sigma^2) + \frac{1}{2\sigma^2} E[(x-\mu)^2] = \frac{1}{2}\left(1 + log\ (2\pi\sigma^2)\right) \tag{102}$$

In a similar way, we can define the cross-entropy between two probability density functions as:

$$H(p, q) = -\int p(x)[log\ q(x)]dx \tag{103}$$

The Kullback–Leibler divergence, or simply relative entropy, is the difference between the cross-entropy of $p(x)$ and $q(x)$ and the entropy of $p(x)$, that is:

$$\begin{aligned} D_{KL}(p, q) = H(p, q) - H(p) &= -\int p(x)[log\ q(x)]dx + \int p(x)[log\ p(x)]dx \\ &= \int p(x) log\left(\frac{p(x)}{q(x)}\right) dx \\ &= E_p\left[log\left(\frac{p(x)}{q(x)}\right)\right] \end{aligned} \tag{104}$$

It should be mentioned that the relative entropy is always non-negative, that is, $D_{KL}(p, q) \geq 0$, being equal to zero if, and only if, $p(x) = q(x)$. Let $p(x)$ and $q(x)$

be univariate Gaussian densities, $N(\mu_1, \sigma_1^2)$ and $N(\mu_2, \sigma_2^2)$. Then, the KL-divergence between then is given by:

$$\begin{aligned} D_{KL}(p,q) &= E_p\left[-log\ \sigma_1 - \frac{1}{2\sigma_1^2}(x-\mu_1)^2 + log\ \sigma_2 - \frac{1}{2\sigma_2^2}(x-\mu_2)^2\right] \\ &= log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{1}{2\sigma_2^2}E_p[(x-\mu_2)^2] - \frac{1}{2\sigma_1^2}E_p[(x-\mu_1)^2] \end{aligned} \tag{105}$$

It is straightforward to note that:

$$E_p[(x-\mu_1)^2] = \sigma_1^2 \tag{106}$$

$$E_p[(x-\mu_2)^2] = E[x^2] - 2E[x]\mu_2 + \mu_2^2 \tag{107}$$

$$E[x^2] = Var[x] + E^2[x] = \sigma_1^2 + \mu_1^2 \tag{108}$$

which finally leads to:

$$\begin{aligned} D_{KL}(p,q) &= log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{1}{2\sigma_2^2}(\sigma_1^2 + \mu_1^2 - 2\mu_1\mu_2 + \mu_2^2) - \frac{1}{2} \\ &= log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \end{aligned} \tag{109}$$

Note that $D_{KL}(p,q) \neq D_{KL}(q,p)$, that is, the relative entropy is not symmetric. The symmetrized KL-divergence between $p(x)$ and $q(x)$ is:

$$\begin{aligned} D_{KL}^{sym}(p,q) &= \frac{1}{2}[D_{KL}(p,q) + D_{KL}(q,p)] \\ &= \frac{1}{4}\left[\frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{\sigma_2^2} + \frac{\sigma_2^2 + (\mu_1-\mu_2)^2}{\sigma_1^2} - 2\right] \\ &= \frac{1}{4\sigma_1^2\sigma_2^2}\left[\left(\sigma_1^2 - \sigma_2^2\right)^2 + (\mu_1-\mu_2)^2\left(\sigma_1^2 + \sigma_2^2\right)\right] \end{aligned} \tag{110}$$

4 PCA-KL: a parametric dimensionality reduction algorithm

One issue found in high-dimensional spaces is related to the weak discrimination power of the Euclidean metric. As dimensionality grows, the contrast provided by the Euclidean distance decreases, i.e., the distribution of norms in a given distribution of points tends to concentrate. This is known as the concentration phenomenon (Lee and Verleysen 2007). In other words, the L2-norm of random vectors grows proportionally to the number of features, $\sqrt{m}$, 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. Im summary, when data is high dimensional, PCA can behave in unexpected ways: upward bias in sample eigenvalues and inconsistency of sample eigenvectors are among the most notable phenomena that appear (Johnstone and Paul 2018). These findings have been exploited to develop new estimation and hypothesis testing methods for the population covariance matrix (Vaswani et al. 2018).

In the last decades, several PCA variations were proposed to mitigate the limitations of the original method. Kernel PCA is a non-linear generalization that corresponds to PCA performed in a reproducing kernel Hilbert space associated with a positive definite kernel (Schölkopf et al. 1999). 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 (Yang and Wang 1999). Another issue with PCA is that every feature is a linear combination of all input variables. Sparse PCA mitigates this issue by finding linear features that contain just a few input variables. It is an extension of PCA for dimensionality reduction that adds sparsity constraints on the input variables (Kunzhe and Huaitie 2018). 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 (Lee and Park 2012).

4.1 Proposed method

As a way to mitigate some limitations of PCA, we propose PCA-KL, a parametric patch-based algorithm for unsupervised metric learning based on the computation of the entropic covariance matrix, a surrogate for the covariance matrix of the data, using the KL-divergence between Gaussian distributions instead of the usual Euclidean distance between the data points.

We denote by $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, with $\mathbf{x}_i \in R^m$ the input data matrix. We can build a KNN graph $G = (V, E)$, with $|V| = n$, by connecting each sample $\mathbf{x}_i$ with its k nearest neighbors. Since the neighborhood can be well approximated by a linear patch, we use the Euclidean distance as similarity measure in this step. Let a patch P_i be the set $\{\mathbf{x}_i\} \cup \{\mathbf{x}_j \in N(i)\}$, where $N(i)$ is the neighborhood set of $\mathbf{x}_i$. Then, we can define the patch matrix P_i as:

$$P_i = [\mathbf{x}_i, \mathbf{x}_{i1}, \mathbf{x}_{i2}, \ldots, \mathbf{x}_{ik}] \tag{111}$$

to denote the $m \times (k+1)$ data matrix that compose the i-th patch. In this study, we assume a parametric model $p(\mathbf{x}; \theta)$ for each feature of the patch matrix, that is, for each line of P_i, where $\theta \in R^L$ is a vector of L parameters. For instance, if we consider a Gaussian model, then $L = 2$, where $\theta_1 = \mu$ is the mean and $\theta_2 = \sigma^2$ is the variance.

Basically, the proposed patch-based method maps each patch P_i to a m-dimensional vector composed by L-dimensional tuples, where each tuple j contains the maximum likelihood estimators of the model parameters. As we have m distinct features, we will

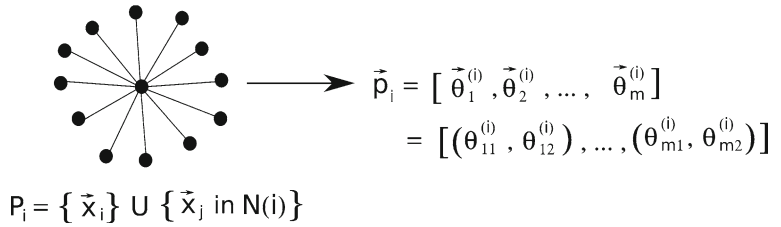

Fig. 1 Mapping from a patch P_i on a graph to a parametric feature vector $\mathbf{p}_i$.

have exactly m tuples. Figure 1 illustrates the mapping from a patch P_i to a parametric feature vector $\mathbf{p}_i$.

The parametric feature vector $\mathbf{p}_i$ for the patch P_i can be expressed as:

$$\mathbf{p}_i = \left[\theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_m^{(i)}\right] \tag{112}$$

where each component is a tuple of L parameters:

$$\theta_j^{(i)} = \left(\theta_{j1}^{(i)}, \theta_{j2}^{(i)}, \ldots, \theta_{jL}^{(i)}\right) \tag{113}$$

The set of all $\mathbf{p}_i$, for $i = 1, 2, \ldots, n$ defines our entropic feature space. We associate to this feature space a centroid given by the sample average of all $\mathbf{p}_i$'s:

$$\tilde{\mathbf{p}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{p}_i \tag{114}$$

We propose to define the entropic difference between two vectors $\mathbf{p}_i$ and $\mathbf{p}_j$ in the parametric feature space as the symmetrized KL-divergence between each one of the tuples, that is:

$$\mathbf{p}_i - \mathbf{p}_j = \left[d_{KL}(\theta_1^{(i)}, \theta_1^{(j)}), \ldots, d_{KL}(\theta_m^{(i)}, \theta_m^{(j)})\right] = \mathbf{d}_{KL}\left(\mathbf{p}_i, \mathbf{p}_j\right) \tag{115}$$

For univariate Gaussian models, a closed-form expression for the symmetrized KL-divergence is given by Eq. (110). Given the above, we can define the entropic covariance matrix C as:

$$C = E\left[\mathbf{d}_{KL}(\mathbf{p}_i, \tilde{\mathbf{p}})\, \mathbf{d}_{KL}(\mathbf{p}_i, \tilde{\mathbf{p}})^T\right] \tag{116}$$

Note that both the regular covariance matrix (used in PCA) and the kernel matrix (used in KPCA) are defined in terms of each sample individually, whereas the entropic covariance matrix is defined in terms of patches, making it less sensitive to outliers. Besides, the proposed entropic PCA has the advantage of producing a projection matrix, exactly like regular PCA. It means that new instances that do not belong to the training set, can be mapped to their new representation in a very easy way.

This is an advantage of PCA-KL in comparison to manifold learning algorithms, such as ISOMAP (Tenenbaum et al. 2000), LLE (Roweis and Saul 2000), Laplacian Eigenmaps (Belkin and Niyogi 2003) and t-SNE (van der Maaten and Hinton 2008).

In the following, we present the PCA-KL algorithm for unsupervised metric learning under Gaussian hypothesis. The input for PCA-KL is a data matrix $X_{m\times n}$, in which each column $\mathbf{x}_j \in R^m$ is a sample. Basically, the main steps of PCA-KL can be summarized as:

1. From the input data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in R^m$ build an undirected proximity graph using the KNN rule;
2. For each patch, that is, $\mathbf{x}_i \cup \eta_i$, where η_i denotes the local neighborhood around $\mathbf{x}_i$, compute the mean and variance of each feature:

$$\mu_k = \frac{1}{|\eta_i|} \sum_{j\in\eta_i} x_j(k) \tag{117}$$

$$\sigma_k^2 = \frac{1}{|\eta_i| - 1} \sum_{j\in\eta_i} (x_j(k) - \mu_k)^2 \tag{118}$$

for $k = 1, 2, \ldots, m$. For simplicity we are assuming a Gaussian model, but other distributions could be adopted at this stage. At the end, this step generates for each patch, the following parametric vector:

$$\mathbf{p}_i = \left[(\mu_1, \sigma_1^2), (\mu_2, \sigma_2^2), \ldots (\mu_m, \sigma_m^2)\right] \tag{119}$$

3. Compute the mean parametric vector for all patches, $\tilde{\mathbf{p}}$, which represents the average distribution, given all the dataset:

$$\tilde{\mathbf{p}} = \left[(\tilde{\mu}_1, \tilde{\sigma}_1^2), (\tilde{\mu}_2, \tilde{\sigma}_2^2), \ldots, (\tilde{\mu}_m, \tilde{\sigma}_m^2)\right] \tag{120}$$

4. Compute the entropic covariance matrix C, a surrogate for the usual covariance matrix based on the KL-divergence between each parametric vector $\mathbf{p}_i$ and the average distribution $\tilde{\mathbf{p}}$ as:

$$C = \frac{1}{n-1} \sum_{i=1}^{n} \mathbf{d}_{KL}(\mathbf{p}_i, \tilde{\mathbf{p}})\, \mathbf{d}_{KL}(\mathbf{p}_i, \tilde{\mathbf{p}})^T \tag{121}$$

where $\mathbf{d}_{KL}(\mathbf{p}_i, \tilde{\mathbf{p}})$ is a column vector of KL-divergences:

$$\mathbf{d}_{KL}(\mathbf{p}_i, \tilde{\mathbf{p}}) = \left[D_{KL}^{sym}\left(\theta_1, \tilde{\theta}_1\right), \ldots, D_{KL}^{sym}\left(\theta_m, \tilde{\theta}_m\right)\right]^T \tag{122}$$

with $\theta_i = (\mu_i, \sigma_i^2)$ and $\tilde{\theta}_i = (\tilde{\mu}_i, \tilde{\sigma}_i^2)$.

5. Select the $d < m$ eigenvectors associated to the d largest eigenvectors of the matrix C to compose the projection matrix W_{PCAKL}.

6. Project the data into the subspace spanned by W_{PCAKL}.

The entropic covariance matrix C is real, symmetric and positive semidefinite, so all its eigenvalues are non-negative, which is a desirable property of the proposed method. Moreover, the method is fully unsupervised since it does not make any assumptions regarding the class labels.

5 Experiments and results

In order to test and evaluate the proposed method for unsupervised dimensionality reduction in classification tasks, we compared its performance against the usual PCA, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE in several public datasets available at www.openml.org. It is worth mentioning that the selected datasets have significant variations in the number of samples and features, as well as different number of classes.

In the first set of experiments, we used an internal index to assess the clusters quality obtained after the unsupervised metric learning provided by the dimensionality reduction methods. We chose the Silhouette coefficient, which is a method of interpretation and validation of consistency within clusters of data (Rousseeuw 1987). Let C_i denote the i-th cluster, then for each data point $i \in C_i$ let $a(i)$ be the mean distance between i and all other data points in the same cluster C_i:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i, j) \tag{123}$$

where $d(i, j)$ is the distance between data points i and j in the cluster C_i. In other words, we can interpret $a(i)$ as a measure of how well the data point i is assigned to its cluster (the smaller the value, the better). Then, we define the mean dissimilarity of a data point i to a cluster C as the mean of the distances from i to all points in C. For each data point i, let $b(i)$ be the smallest mean distance of i to all points in any other cluster which i is not a member:

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j) \tag{124}$$

The cluster with the smallest mean dissimilarity is the neighboring cluster of i because it is the next best fit cluster for point i. Let:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \text{ if } |C_i| > 1 \tag{125}$$

be the silhouette value of the data point i and

$$s(i) = 0, \text{ if } |C_i| = 1 \tag{126}$$

Combining both definitions we have:

$$s(i) = \begin{cases} 1 - \dfrac{a(i)}{b(i)} & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ \dfrac{b(i)}{a(i)} - 1 & \text{if } a(i) > b(i) \end{cases} \tag{127}$$

Note that $-1 \leq s(i) \leq 1$. An $s(i)$ close to one means that the data is appropriately clustered. If $s(i)$ tends to negative one, then i should be clustered in its neighboring cluster. An $s(i)$ near zero means that the data point i is on the border of two natural clusters. The mean $s(i)$ over all points of a cluster is a measure of how tightly grouped all the points in the cluster are. Therefore, the mean $s(i)$ over all data of the entire dataset, known as the Silhouette coefficient is a measure of how appropriately the data have been clustered.

A few datasets have a prevalence of negative features, so NMF is not an option. Table 1 summarizes the results. The results suggest that PCA-KL is more efficient in building a meaningful representation in terms of the consistency within clusters of data than PCA. Note that in 22 of 44 datasets, PCA-KL obtained the highest Silhouette coefficient, that is, in 50% of the cases the proposed method produced better defined clusters than the other linear and non-linear methods. Another important aspect to be highlighted is that PCA-KL, in average, outperformed not only PCA and NMF (linear methods), but also all manifold learning algorithms, which indicates that the proposed method can be a promising alternative to dimensionality reduction for unsupervised metric learning. Moreover, according to a Wilcoxon signed-rank test, PCA-KL produced significantly better clusters (in terms of Silhouette coefficient) than PCA (p value $= 1.12 \times 10^{-8}$), NMF (p value $= 4.60 \times 10^{-5}$), Kernel PCA (p value $= 1.81 \times 10^{-6}$), ISOMAP (p value $= 5.54 \times 10^{-7}$), LLE (p value $= 9.04 \times 10^{-8}$), Laplacian Eigenmaps (p value $= 1.42 \times 10^{-4}$) and t-SNE (p value $= 9.02 \times 10^{-5}$) for a significance level $\alpha = 1\%$.

To illustrate how the proposed method is capable of producing better defined clusters, we present some scatter plots for the two dimensional case, comparing PCA and PCA-KL. Figures 2 and 3 show the clusters for the iris and cardiotocography datasets. Note that the clusters produced by PCA-KL have a lower intra-class scattering, that is, they tend to be more concentrated around the mean. Moreover, it is possible to notice that the variance in the principal components is smaller on PCA-KL.

In the second set of experiments, we compared the performance of PCA-KL against PCA, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE in supervised classification. For this purpose, eight different parametric and non-parametric classifiers were selected: K-Nearest Neighbors (k = 7), Support Vector Machine (linear), Naive Bayes and Quadratic Discriminant Analysis under Gaussian hypothesis, Decision Tree, Multilayer Perceptron, Gaussian Process Classifier and Random Forest Classifier. In all experiments, we selected 60% of the samples for training and 40% of the samples for testing. Tables 2, 3 and 4 show the classification accuracies for several datasets after dimensionality reduction. The best result in a line is boldfaced and the second best result is underlined.

Table 1 Silhouette coefficients for clusters produced by PCA, PCA-KL, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE for several datasets from OpenML.org (2D case)

	PCA	PCA-KL	NMF	KPCA	ISO	LLE	LAP	t-SNE
Iris	0.401	**0.620**	0.588	0.469	0.452	0.365	0.541	0.498
Wine	0.526	0.535	0.161	0.610	0.547	0.242	**0.750**	0.553
Blood	0.086	**0.179**	0.109	0.026	0.082	0.000	0.004	0.042
kc1	0.371	0.450	**0.550**	0.210	0.187	0.236	−0.441	−0.025
mfeat-fourier	−0.085	0.000	0.000	−0.072	−0.066	−0.050	0.002	**0.234**
Cardio	−0.082	0.498	−0.191	0.035	0.316	0.087	**0.682**	0.602
Texture	−0.092	0.072	X	−0.081	0.116	0.074	0.248	**0.454**
Satimage	0.219	0.233	X	0.247	0.229	0.151	0.204	**0.252**
First-order	−0.167	**−0.076**	X	−0.105	−0.108	−0.104	−0.491	−0.079
Wall-robot	−0.054	**−0.021**	−0.201	−0.102	−0.068	−0.116	−0.172	−0.095
car(3)	−0.110	**0.042**	−0.035	−0.082	−0.103	−0.062	−0.060	0.017
Australian	0.279	**0.362**	0.135	0.276	0.291	0.130	0.345	0.268
Haberman	0.060	0.096	**0.100**	−0.024	0.061	−0.004	−0.032	0.001
Statlog	0.264	0.272	0.044	0.246	**0.295**	0.071	0.259	0.239
Sonar	0.026	**0.096**	0.014	0.018	0.038	0.016	0.023	0.016
tae	−0.059	−0.027	−0.035	**−0.004**	−0.072	−0.023	−0.045	−0.029
Speech	−0.019	**0.052**	X	−0.014	−0.202	−0.755	−0.121	−0.162
Transplant	0.485	**0.551**	0.459	0.436	0.485	0.410	0.438	0.494
geyser1(2)	0.328	0.328	0.023	**0.346**	0.339	0.205	0.187	0.120
hayes-roth	−0.023	**0.084**	0.030	0.038	−0.010	−0.013	−0.013	−0.008
SPECTF	−0.018	−0.015	−0.029	**0.093**	−0.036	−0.029	0.019	0.046
newton_hema	0.087	0.105	0.038	**0.113**	0.082	0.077	0.090	0.107

Table 1 continued

	PCA	PCA-KL	NMF	KPCA	ISO	LLE	LAP	t-SNE
Servo	0.121	**0.201**	0.075	0.105	0.114	0.104	0.085	0.129
wildcat(2)	0.151	0.172	**0.202**	0.081	0.125	0.149	0.027	0.114
veteran(2)	0.001	**0.083**	0.023	−0.007	0.002	0.016	0.002	0.012
Datatrieve	0.239	**0.251**	0.199	0.010	0.096	0.066	0.080	0.009
machine_cpu	0.498	**0.540**	0.524	0.399	0.492	0.496	0.410	0.427
grub_damage(2)	0.042	0.097	0.002	0.050	0.066	0.094	**0.131**	0.072
mu284(2)	0.301	0.322	0.031	0.338	0.287	0.284	**0.389**	0.314
census6(2)	0.003	**0.017**	0.011	−0.004	0.007	0.009	−0.006	0.000
disclosure_z(2)	−0.001	**0.007**	0.000	0.006	−0.001	0.000	−0.001	0.001
fem._bladder(2)	0.122	0.230	**0.370**	0.008	0.170	0.143	0.030	0.058
triazines(2)	0.009	0.020	0.027	**0.064**	0.015	0.054	0.023	0.044
page_blocks(2)	0.419	0.531	0.477	0.218	0.527	**0.581**	0.436	0.226
male_lung(2)	0.563	0.791	**0.877**	−0.182	0.673	0.697	−0.057	0.165
stock(2)	0.161	0.229	0.156	0.194	0.215	0.127	**0.308**	0.214
glass(2)	0.018	**0.081**	−0.017	0.004	0.011	0.029	−0.009	0.017
mw1	0.349	0.392	**0.401**	0.122	0.286	0.175	0.180	0.051
car(2)	0.039	**0.178**	0.092	0.157	0.046	0.163	0.079	0.079
optdigits(2)	0.013	**0.366**	0.057	0.146	−0.069	−0.221	−0.351	0.075
retinol(2)	−0.008	**0.154**	0.003	0.004	0.010	−0.008	0.012	0.010
ar1	0.265	**0.444**	0.313	0.029	0.216	−0.004	−0.002	−0.180
diggle(2)	0.406	**0.476**	0.296	0.409	0.450	0.328	0.304	0.106
rmftsa(2)	0.228	**0.299**	0.254	0.242	0.238	0.185	0.230	0.221
Average	0.145	**0.234**	0.153	0.115	0.155	0.099	0.107	0.130
SD	0.197	0.209	0.227	**0.176**	0.206	0.225	0.254	0.187

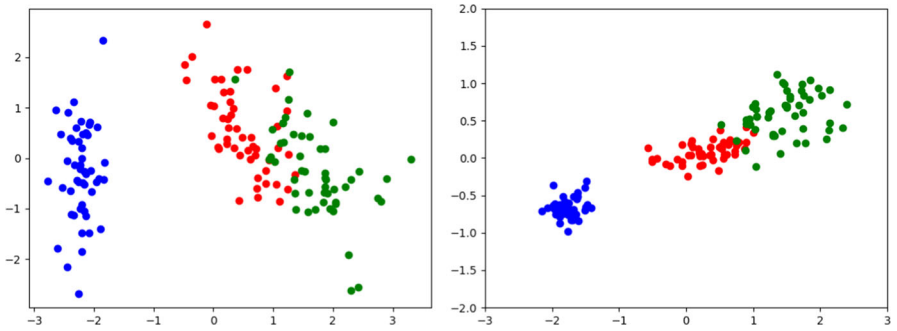

Fig. 2 Scatterplots of iris dataset for the 2D case: PCA versus PCA-KL

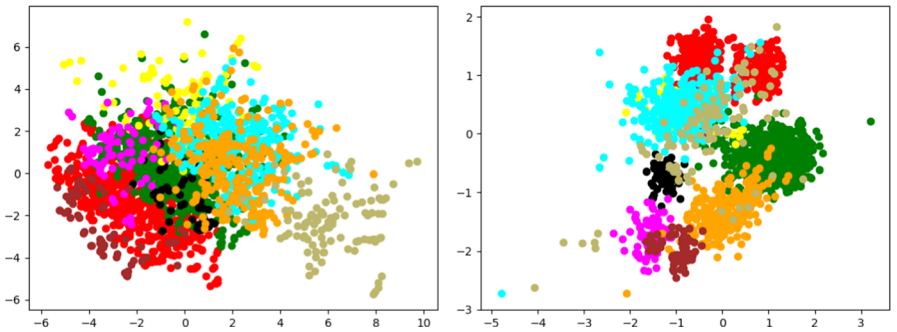

Fig. 3 Scatterplots of cardiotocography dataset for the 2D case: PCA versus PCA-KL

At first glance, it is difficult to evaluate the results, since there is no method that is uniformly superior to all the others. However, looking at the average accuracy, the results are more conclusive. Table 5 shows the average and standard deviation of all accuracies for each feature extraction algorithm. The results indicate that for these datasets, in average, PCA-KL outperformed all linear and non-linear dimensionality reduction methods. We also performed a hypothesis test to check whether the differences are statistically significant. According to a Wilcoxon signed-rank test, PCA-KL produces significantly higher classification accuracies than PCA (p value = 2.40×10^{-16}), NMF (p value = 1.61×10^{-12}), Kernel PCA (p value = 6.32×10^{-14}), ISOMAP (p value = 4.10×10^{-11}), LLE (p value = 9.31×10^{-14}) and Laplacian Eigenmaps (p value = 2.73×10^{-13}) for $\alpha = 0.01$, that is, a significance level of 1%. For the same value of α, we cannot conclude that there are significant differences between PCA-KL and t-SNE (p value = 0.371).

The obtained results emphasize that the proposed PCA-KL method is competitive with the existing dimensionality reduction algorithms, since, overall, it is capable of producing features that are more discriminant than those generated by PCA, NMF and some manifold learning algorithms. In other words, PCA-KL is a viable option for dimensionality reduction and unsupervised metric learning in classification problems. One important aspect to be highlighted in the proposed method is related to the choice of the number of neighbors (K) in the KNN graph. In some datasets, PCA-KL can

Table 2 Supervised classification accuracy obtained by different classifiers after PCA, PCA-KL, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE dimensionality reduction for datasets iris, mfeat-fourier, cardiotocography, car(3) and haberman from OpenML.org (2D case)

	PCA	PCA-KL	NMF	KPCA	ISOMAP	LLE	LAP	t-SNE
Iris dataset								
KNN	0.966	0.983	**1.000**	0.900	0.900	0.950	0.866	0.950
SVM	0.950	**0.983**	0.980	0.833	0.916	0.666	0.300	0.933
NB	0.916	**0.983**	**0.983**	0.850	0.850	0.966	0.833	0.850
DT	0.900	0.966	0.960	0.833	0.900	**0.983**	0.816	0.950
QDA	0.950	**0.983**	0.966	0.833	0.916	0.966	0.883	0.850
MPL	0.950	**0.983**	0.950	0.833	0.916	0.916	0.300	0.900
GPC	0.933	**0.983**	**0.983**	0.833	0.916	0.816	0.300	0.950
RFC	0.916	**0.983**	0.980	0.900	0.933	**0.983**	0.866	0.930
mfeat-fourier dataset								
KNN	0.420	0.531	0.512	0.436	0.515	0.581	0.533	**0.780**
SVM	0.440	0.568	0.262	0.367	0.513	0.190	0.172	**0.710**
NB	0.428	0.527	0.550	0.446	0.515	0.495	0.465	**0.662**
DT	0.365	0.456	0.482	0.403	0.433	0.550	0.515	**0.767**
QDA	0.432	0.551	0.538	0.452	0.531	0.572	0.555	**0.716**
MPL	0.442	0.567	0.523	0.416	0.533	0.387	0.087	**0.755**
GPC	0.435	0.578	0.347	0.386	0.528	0.190	0.175	**0.792**
RFC	0.377	0.502	0.510	0.436	0.478	0.572	0.548	**0.785**
Cardiotocography dataset								
KNN	0.619	**0.995**	0.283	0.668	0.929	0.606	0.925	**0.995**
SVM	0.652	0.640	0.309	0.599	0.909	0.265	0.265	**0.967**
NB	0.613	0.854	0.292	0.666	0.933	0.325	0.914	**0.996**
DT	0.552	**0.998**	0.233	0.634	0.920	0.325	0.916	0.990
QDA	0.632	0.982	0.298	0.662	0.972	0.643	0.985	**0.996**
MPL	0.640	0.946	0.313	0.609	0.914	0.325	0.265	**0.996**
GPC	0.650	0.607	0.316	0.573	0.911	0.265	0.265	**0.996**
RFC	0.599	**0.998**	0.240	0.662	0.928	0.325	0.903	0.995
car(3) dataset								
KNN	0.705	0.862	0.726	0.794	0.807	0.894	0.739	**0.907**
SVM	0.693	0.780	0.721	0.693	0.817	0.693	0.693	**0.855**
NB	0.682	0.767	0.708	0.709	0.690	0.731	0.693	**0.793**
DT	0.680	0.874	0.774	0.791	0.789	0.891	0.702	**0.893**
QDA	0.676	0.776	0.702	0.697	0.693	0.731	0.703	**0.793**
MPL	0.682	0.768	0.713	0.705	0.640	0.738	0.693	**0.822**
GPC	0.693	0.789	0.719	0.709	0.800	0.693	0.693	**0.921**
RFC	0.686	0.877	0.777	0.794	0.807	0.900	0.728	**0.901**

Table 2 continued

	PCA	PCA-KL	NMF	KPCA	ISOMAP	LLE	LAP	t-SNE
Haberman dataset								
KNN	0.747	**0.756**	0.772	0.699	0.723	0.731	0.699	0.731
SVM	0.731	**0.773**	0.756	0.731	0.731	0.731	0.731	0.731
NB	**0.764**	0.756	0.747	0.731	0.739	0.739	0.731	0.731
DT	0.658	**0.715**	0.674	0.634	0.609	0.650	0.674	0.593
QDA	0.723	**0.772**	0.756	0.731	0.723	0.739	0.731	0.731
MPL	**0.731**	**0.731**	**0.731**	**0.731**	**0.731**	**0.731**	**0.731**	**0.731**
GPC	0.723	0.764	**0.772**	0.731	0.715	0.731	0.731	0.682
RFC	0.691	0.707	0.691	0.715	**0.723**	0.674	0.674	0.715

Table 3 Supervised classification accuracy obtained by different classifiers after PCA, PCA-KL, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE dimensionality reduction for datasets sonar, tae, hayes-roth, newton_hema and servo from OpenML.org (2D case)

	PCA	PCA-KL	NMF	KPCA	ISOMAP	LLE	LAP	t-SNE
Sonar dataset								
KNN	0.642	**0.678**	0.595	0.607	0.619	0.619	0.607	0.642
SVM	0.678	0.654	0.595	0.654	0.630	0.535	0.535	**0.773**
NB	0.607	**0.714**	0.523	0.571	0.702	0.630	0.619	0.535
DT	0.642	0.666	0.547	0.607	0.595	0.559	**0.750**	0.690
QDA	0.630	**0.738**	0.571	0.619	0.678	0.607	0.607	0.476
MPL	0.619	**0.738**	0.535	0.535	0.714	0.535	0.535	0.690
GPC	0.666	0.666	0.607	0.630	0.619	0.535	0.535	**0.809**
RFC	0.607	0.666	0.607	0.571	0.571	0.583	0.654	**0.773**
tae dataset								
KNN	0.393	0.557	0.311	0.524	0.495	0.459	0.491	**0.639**
SVM	0.409	**0.557**	0.344	0.508	0.540	0.295	0.393	**0.557**
NB	0.557	0.557	0.344	0.508	0.557	0.442	**0.590**	0.491
DT	0.540	0.557	0.573	**0.622**	0.524	0.524	0.540	**0.622**
QDA	0.491	0.557	0.327	0.508	0.557	0.442	**0.590**	0.557
MPL	0.426	**0.573**	0.327	0.327	0.409	0.295	0.372	0.344
GPC	0.459	0.557	0.311	0.524	0.508	0.344	0.393	**0.590**
RFC	0.557	0.524	**0.590**	0.557	0.557	0.540	0.491	0.573
hayes-roth dataset								
KNN	0.509	0.811	**0.849**	0.603	0.603	0.547	0.528	0.433
SVM	0.566	**0.811**	0.566	0.566	0.584	0.566	0.566	0.773
NB	0.566	**0.679**	**0.679**	0.584	0.566	0.566	0.566	0.622
DT	0.641	0.773	0.698	**0.792**	0.660	0.773	0.716	**0.792**
QDA	0.566	0.603	**0.716**	0.584	0.566	0.566	0.566	0.622

Table 3 continued

	PCA	PCA-KL	NMF	KPCA	ISOMAP	LLE	LAP	t-SNE
MPL	0.566	**0.716**	0.679	0.566	0.566	0.566	0.566	0.566
GPC	0.547	**0.811**	0.641	0.566	0.622	0.566	0.566	0.754
RFC	0.773	**0.792**	0.773	0.773	0.716	**0.792**	0.754	**0.792**
newton_hema dataset								
KNN	0.678	**0.732**	0.589	**0.732**	0.696	0.714	0.660	**0.732**
SVM	0.750	0.714	0.660	0.660	0.750	0.464	0.464	**0.803**
NB	0.696	**0.714**	0.482	0.660	0.642	0.571	0.625	0.589
DT	0.660	**0.767**	0.553	0.750	0.696	0.714	0.732	0.714
QDA	0.660	0.660	0.517	**0.696**	0.642	0.571	0.607	0.607
MPL	0.607	**0.678**	0.607	0.660	0.607	0.464	0.464	0.607
GPC	0.732	0.767	0.625	0.642	0.785	0.517	0.464	**0.803**
RFC	0.732	**0.767**	0.625	0.750	0.767	0.714	0.732	**0.767**
Servo dataset								
KNN	0.776	0.940	0.701	0.746	0.791	0.850	0.746	**0.955**
SVM	0.805	**0.940**	0.731	0.731	0.805	0.731	0.731	**0.940**
NB	**0.940**	**0.940**	0.746	0.716	0.820	**0.940**	0.776	0.850
DT	0.880	**0.925**	0.671	0.731	0.791	0.865	0.716	**0.925**
QDA	0.925	**0.940**	0.761	0.716	0.820	0.895	0.791	0.880
MPL	0.791	**0.940**	0.776	0.731	.791	0.731	0.731	0.731
GPC	0.791	0.925	0.761	0.731	0.820	0.731	0.731	**0.955**
RFC	0.880	**0.940**	0.641	0.776	0.880	0.895	0.716	**0.940**

produce significantly different results by changing this parameter, showing to be quite sensitive. One should keep in mind the tradeoff between locality preservation and precision in the parameter estimation in the definition of the best value of K. In our experiments, we observed that in all datasets the values of K that were chosen belonged to the interval $[20, n/5]$, where n is the number of samples. In all experiments, the fine-tuning of this parameter was performed by a line search in which the lower bound was set to 20, the increment was set to 10 and the upper bound was $n/2$.

Finally, a comparison of computational times for each dimensionality reduction method was performed to investigate how efficient the different algorithms are in practice. Our hardware platform consists in a 64-bit Core i7-4510 2.0GHz with 8Gb of RAM. The software platform is comprised by Linux Ubuntu with Anaconda Python distribution. Table 6 shows the elapsed times in seconds for each dimensionality reduction algorithm considering several public datasets. Note that, as expected, PCA is the fastest method by a large margin. The results also show that t-SNE is, in average, about 175 times slower than PCA-KL and ISOMAP is about 4 times slower than the proposed method.

Table 4 Supervised classification accuracy obtained by different classifiers after PCA, PCA-KL, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE dimensionality reduction for datasets disclosure_z, plasma_retinol, diggle_table_a2 and rmftsa_ladata from OpenML.org (2D case)

	PCA	PCA-KL	NMF	KPCA	ISOMAP	LLE	LAP	t-SNE
disclosure_z dataset								
KNN	0.464	0.486	**0.520**	**0.520**	0.467	0.456	0.490	0.501
SVM	0.490	0.543	0.543	**0.600**	0.490	0.532	0.532	0.464
NB	0.528	0.554	0.535	**0.577**	0.532	0.498	0.532	0.539
DT	0.505	0.494	0.483	0.498	**0.550**	0.475	**0.550**	0.490
QDA	0.524	0.554	0.535	**0.581**	0.528	0.494	0.547	0.520
MPL	0.501	**0.562**	0.509	0.532	0.486	0.532	0.532	0.501
GPC	0.483	0.528	0.535	**0.584**	0.475	0.532	0.532	0.456
RFC	0.483	0.487	0.456	0.486	**0.509**	0.501	0.494	0.490
plasma_retinol dataset								
KNN	0.500	0.555	0.523	0.531	0.555	0.523	0.579	**0.619**
SVM	0.531	0.611	0.468	0.563	**0.619**	0.563	0.563	0.555
NB	0.555	0.619	0.571	0.587	0.579	0.539	0.619	**0.626**
DT	0.436	**0.611**	0.492	0.531	0.539	0.492	0.539	0.484
QDA	0.523	0.611	0.579	0.611	**0.626**	0.523	**0.626**	0.563
MPL	0.555	**0.563**	**0.563**	**0.563**	**0.563**	**0.563**	**0.563**	**0.563**
GPC	0.531	0.611	0.476	0.563	**0.634**	0.563	0.563	0.547
RFC	0.476	**0.611**	0.436	0.587	0.507	0.484	0.595	0.563
diggle_table_a2 dataset								
KNN	0.911	0.951	**0.975**	0.967	0.919	0.879	0.927	**0.975**
SVM	0.887	**0.967**	**0.967**	0.919	0.919	0.524	0.516	0.911
NB	0.870	0.862	0.774	**0.927**	0.887	0.846	0.879	0.717
DT	0.951	**0.992**	0.951	0.983	0.959	0.903	0.959	0.967
QDA	0.854	**0.951**	0.879	0.927	0.862	0.927	0.887	0.774
MPL	0.854	0.870	0.967	0.879	0.862	0.516	0.516	**0.991**
GPC	0.903	**0.984**	**0.984**	0.887	0.903	0.887	0.516	0.975
RFC	0.943	**0.984**	**0.984**	0.951	0.951	0.935	0.975	0.975
rmftsa_ladata dataset								
KNN	0.784	0.833	0.745	0.794	0.779	0.745	0.789	**0.848**
SVM	0.774	**0.838**	0.784	0.774	.784	0.578	0.568	0.735
NB	0.769	**0.833**	0.789	0.803	0.789	0.784	0.794	0.789
DT	0.725	0.725	0.671	0.715	0.759	0.676	.759	**0.808**
QDA	0.750	**0.838**	0.754	0.774	0.779	0.754	0.774	0.789
MPL	0.769	**0.823**	0.784	0.759	0.789	0.754	0.568	0.799
GPC	0.784	**0.838**	0.779	0.759	0.784	0.647	0.568	**0.838**
RFC	0.769	0.799	0.696	0.779	0.779	0.759	0.799	**0.823**

Table 5 Average accuracies obtained by different classifiers after PCA, PCA-KL, NMF, Kernel PCA, ISOMAP, LLE, Laplacian Eigenmaps and t-SNE dimensionality reduction for the OpenML.org datasets in Tables 2, 3 and 4 (2D case)

	PCA	PCA-KL	NMF	KPCA	ISO	LLE	LAP	t-SNE
Average	0.664	**0.750**	0.634	0.667	0.704	0.629	0.624	0.749
Std. Dev.	0.159	0.159	0.198	**0.140**	0.151	0.187	0.179	0.164

Table 6 Computational time in seconds for each dimensionality reduction algorithm for several public OpenML.org datasets

	PCA	PCA-KL	NMF	KPCA	ISO	LLE	LAP	t-SNE
Blood	0.0005	0.11	0.58	1.27	0.16	0.08	0.07	23.36
kc1	0.02	0.88	0.05	0.25	2.12	0.34	1.33	163.1
mfeat-fourier	0.003	5.18	0.27	0.29	3.58	1.17	0.84	128.68
Cardio	0.001	1.85	x	0.38	3.21	0.41	0.31	173.1
Texture	0.003	6.23	x	1.58	24.20	3.31	2.74	1174.1
Satimage	0.003	5.95	x	2.76	60.98	10.14	5.33	1619.5
Theorem	0.003	10.64	0.23	4.38	50.17	6.32	10.16	1405.53
Wall-robot	0.001	2.79	0.05	1.75	25.44	4.65	3.1	1086.29
car(3)	0.0007	0.21	0.01	0.23	2.21	0.99	0.48	113.68
Australian(4)	0.0007	0.22	0.04	0.02	0.37	0.15	0.06	22.48
Haberman	0.0004	0.02	0.002	0.005	0.06	0.04	0.02	6.05
Heart-statlog	0.0005	0.07	0.009	0.008	0.05	0.04	0.01	5.7
Sonar	0.001	0.38	0.01	0.003	0.03	0.03	0.01	1.92
census6	0.0005	0.04	0.005	0.01	0.11	0.06	0.03	7.9
disclosure_z	0.0005	0.03	0.01	0.02	0.29	0.11	0.03	17.91
page-blocks	0.0007	1.03	0.01	1.94	21.28	1.55	1.6	1115.56
male_lung	0.0005	0.04	0.002	0.2	0.21	0.08	0.03	13.39
Stock	0.0006	0.16	0.01	0.1	0.44	0.11	0.15	41.76
mw1	0.0008	0.36	0.004	0.007	0.1	0.05	0.03	8.15
Optdigits	0.004	11.07	0.08	1.9	30.67	7.62	5.37	1214.2
rmftsa_ladata	0.0004	0.14	0.01	0.02	0.19	0.11	0.05	17.6
Average	**0.002**	2.25	0.08	0.81	10.75	1.78	1.51	398.1
SD	**0.004**	3.49	0.15	1.18	17.98	2.93	2.60	576.08

6 Conclusions

Dimensionality reduction is a fundamental step in the analysis of multivariate data in many pattern recognition and machine learning applications. From simple visualization to feature extraction, these methods play an important role in classification problems. Besides, it has been shown that learning from high dimensional data can be a challenging task.

Among all dimensionality reduction methods, PCA is still the de-facto standard algorithm considered as the first choice by machine learning researchers. PCA is based on finding the orthogonal directions that maximize the variance of the data. This is optimal from a data compression point of view, since it has been shown to be equivalent to the minimization of the mean square error between the original and the reduced representation. For this reason, after the PCA transformation, data is organized in clusters with large scattering, which is undesirable for classification problems. To overcome this limitation, in this paper we presented PCA-KL, a parametric method based on a information-theoretic measure to find a surrogate for the covariance matrix replacing the standard Euclidean distance in the feature space by the relative entropy between Gaussian distributions estimated in each local neighborhood (patch). Results with real datasets showed that besides improving the quality of the produced clusters, which is a desirable feature in unsupervised classification, PCA-KL can also improve the classification accuracy for several supervised classifiers from the pattern recognition literature, indicating that it is a promising alternative to dimensionality reduction and unsupervised metric learning.

Briefly speaking, the main advantages of PCA-KL are: (1) the evaluation of new instances is very straightforward, since once the projection matrix is build, the mapping is direct, unlike most manifold learning algorithms (ISOMAP, LLE, Laplacian Eigenmaps and t-SNE); (2) PCA-KL is a patch-based method so it can be less sensitive to the presence of noise and outliers in data; (3) the method can be easily extended to different statistical models and divergences. However, PCA-KL has some limitations, the major one being the sensitivity to the patch size K. Experiments have shown that variations on this parameter can lead to significantly different classification results. We still do not have a complete solution regarding the estimation of this parameter.

Future works may include the use of other information-theoretic measures such as the Bhattacharyya and Hellinger distances. Furthermore, we also intend to employ distances based on the Fisher information metric. A supervised version of PCA-KL, considering only neighbors that belong to the same class of the central data point is another possible improvement. Extensions to non-linear dimensionality reduction by the incorporation of different kernels is another possible extension to the proposed method. Different statistical models can be considered instead of the Gaussian distribution. For instance, Gaussian–Markov random field models can be used to capture the spatial dependence between the data points through the definition of an additional coupling parameter, known as inverse temperature. Another relevant problem to be tackled in the future is the adaptive definition of the appropriate patch size. Local analysis of the Hessian matrix can provide insights about how the adjustment of this parameter. Points exhibiting high curvature suggest that a smaller neighborhoods are preferred whereas points with lower curvature indicate that larger neighborhoods can be considered. A limitation with this approach would be an increase in the computational cost. Finally, we intend to study how information-theoretic measures can be applied in manifold learning algorithms as a way to improve unsupervised metric learning.

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

References

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

Bellet A, Habrard A, Sebban M (2013) A survey on metric learning for feature vectors and structured data. arXiv:1306.6709

Bellman R (1961) Adaptive control processes: a guided tour. Princeton University Press, Princeton

Borg I, Groenen P (2005) Modern multidimensional scaling: theory and applications, 2nd edn. Springer, Berlin

Carreira-Perpinan MA (1997) A Review of Dimension Reduction Techniques. Technical report, University of Sheffield

Chávez E, Navarro G, Baeza-Yates R, Marroquín JL (2001) Searching in metric spaces. ACM Comput Surv 33(3):273–321

Cook JA, Sutskever I, Mnih A, Hinton G (2007) Visualizing similarity data with a mixture of maps. In: Proceedings of the 11 th international conference on artificial intelligence and statistics, vol 2, pp 67–74

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Cambridge

Cox TF, Cox MAA (2001) Multidimensional scaling. In: Monographs on statistics and applied probability, vol 88. Chapman & Hall

de Ridder D, Duin RP (2002) Locally linear embedding for classification. Technical report, Delft University of Technology

Ding C, He X, Simon H (2005) On the equivalence of nonnegative matrix factorization and spectral clustering. In: Proceedings of SIAM international conference on data mining, pp 606–610

Ding C, Li T, Peng W (2008) On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. Comput Stat Data Anal 52:3913–3927

Domingos P (1999) The role of occam's razor in knowledge discovery. Data Min Knowl Discov 3(4):409–425

Errica F (2018) Step-by-step derivation of sne and t-sne gradients. http://pages.di.unipi.it/errica/curious/derivations-sne-tsne

Fukunaga K (1990) Introduction to statistical pattern recognition. Academic Press, London

Günter S, Schraudolph NN, Vishwanathan S (2007) Fast iterative kernel principal component analysis. J Mach Learn Res 8:1893–1918

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning. Springer, Berlin

Hinton GE, Roweis ST (2003) Stochastic neighbor embedding. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems, vol 15. MIT Press, Cambridge, pp 857–864

Hotelling H (1933) Analysis of a complex of statistical variables into principal components. J Educ Psychol 24:498–520

Hughes GF (1968) On the mean accuracy of statistical pattern recognizers. IEEE Trans Inf Theory 14(1):55–63

Huo X, Ni X, Smith AK (2008) A survey of manifold-based learning methods. In: Series on computers and operations research, vol 6. World Scientific, pp 691–745

Hwang J, Lay S, Lippman A (1994) Nonparametric multivariate density estimation: a comparative study. IEEE Trans Signal Process 42(10):2795–2810

Hyvarinen A, Karhunen J, Oja E (2001) Independent component analysis. Wiley, Hoboken

Jimenes LO, Landgrebe D (1998) Supervised classification in high dimensional space: geometrical, statistical and asymptotical properties of multivariate data. IEEE Trans Syst Man Cybern 28(1):39–54

Johnstone IM, Paul D (2018) Pca in high dimensions: an orientation. Proc IEEE 106(8):1277–1292

Jolliffe IT (2002) Principal component analysis, 2nd edn. Springer, Berlin

Kunzhe W, Huaitie X (2018) Sparse kernel feature extraction via support vector learning. Pattern Recognit Lett 101(1):67–73

Lee DD, Seung HS (1999) Learning the parts of objects by non-negative matrix factorization. Nature 401:788–791

Lee JA, Verleysen M (2007) Nonlinear dimensionality reduction. Springer, Berlin

Lee K, Park H (2012) Probabilistic learning of similarity measures for tensor PCA. Pattern Recognit Lett 33(10):1364–1372

Li D, Tian Y (2018) Survey and experimental study on metric learning methods. Neural Netw 105:447–462

Lin C (2007) On the convergence of multiplicative update algorithms for nonnegative matrix factorization. IEEE Trans Neural Netw 18(6):1589–1596
Marimont R, Shapiro M (1979) Nearest neighbour searches and the curse of dimensionality. IMA J Appl Math 24(1):59–70
Melville J (2015) SNEER: stochastic neighbor embedding experiments in R. http://jlmelville.github.io/sneer/gradients.html
Nguyen LH, Holmes S (2019) Ten quick tips for effective dimensionality reduction. PLoS Comput Biol 15(6):e1006907
Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65
Roweis S, Saul L (2000) Nonlinear dimensionality reduction by locally linear embedding. Science 290:2323–2326
Saul L, Roweis S (2003) Think globally, fit locally: unsupervised learning of low dimensional manifolds. J Mach Learn Res 4:119–155
Schölkopf B, Smola A, Müller KR (1999) Kernel principal component analysis. In: Advances in kernel methods—support vector learning. MIT Press, pp 327–352
Scott DW (1992) Multivariate density estimation. Wiley, Hoboken
Shlens J (2005) A tutorial on principal component analysis. In: Systems neurobiology laboratory. Salk Institute for Biological Studies
Suárez JL, García S, Herrera F (2018) A tutorial on distance metric learning: mathematical foundations, algorithms and software. arXiv:1812.05944
Tenenbaum JB, de Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290:2319–2323
Trunk GV (1979) A problem of dimensionality: a simple example. IEEE Trans Pattern Anal Mach Intell 1(3):306–307
Vaswani N, Chi Y, Bouwmans T (2018) Rethinking pca for modern data sets: Theory, algorithms, and applications. Proc IEEE 106(8):1274–1276
van der Maaten L, Hinton G (2008) Visualizing high-dimensional data using t-sne. J Mach Learn Res 9:2579–2605
von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17:395–416
Wang F, Sun J (2015) Survey on distance metric learning and dimensionality reduction in data mining. Data Min Knowl Discov 29(2):534–564
Yang L, Jin R (2006) Distance metric learning: a comprehensive survey. Mich State Univ 2:4
Yang TN, Wang SD (1999) Robust algorithms for principal component analysis. Pattern Recognit Lett 20(9):927–933
Young TY, Calvert TW (1974) Classification, Estimation and Pattern Recognition. Elsevier, Amsterdam
Zimek A, Schubert E, Kriegel HP (2012) A survey on unsupervised outlier detection in high-dimensional numerical data. Stat Anal Data Min 5(5):363–387