Supervised Locally Linear Embedding Algorithm for Pattern Recognition

Olga Kouropteva*, Oleg Okun, and Matti Pietikäinen

Machine Vision Group
Infotech Oulu and Department of Electrical and Information Engineering
P.O.Box 4500, FIN-90014 University of Oulu, Finland
{kouropte,oleg,mkp}@ee.oulu.fi

Abstract. The dimensionality of the input data often far exceeds their intrinsic dimensionality. As a result, it may be difficult to recognize multidimensional data, especially if the number of samples in a dataset is not large. In addition, the more dimensions the data have, the longer the recognition time is. This leads to the necessity of performing dimensionality reduction before pattern recognition. Locally linear embedding (LLE) [1, 2] is one of the methods intended for this task. In this paper, we investigate its extension, called supervised locally linear embedding (SLLE), using class labels of data points in their mapping into a low-dimensional space. An efficient eigendecomposition scheme for SLLE is derived. Two variants of SLLE are analyzed coupled with a k nearest neighbor classifier and tested on real-world images. Preliminary results demonstrate that both variants yield identical best accuracy, despite of being conceptually different.

1 Introduction

In pattern recognition, raw data acquired by a camera or scanner are often fed as they are or after simple pre-processing to a recognition module. Though being straightforward, this approach suffers from several major drawbacks: a large data dimensionality makes recognition difficult and time-consuming and in combination with a small data set, the effect known as the curse of dimensionality unavoidably lowers the accuracy rate.

To eliminate unfavorable consequences of using multidimensional data for recognition, a kind of dimensionality reduction is wanted. One popular way to do this is to perform a transformation of the original data, lowering their dimension. PCA is undoubtly the most frequently used technique for this purpose. Despite of its simplicity and good results obtained in solving many tasks, PCA is essentially a linear technique, which can make it an inappropriate choice when the data possess intrinsic nonlinearity.

To overcome this problem, a new technique, called locally linear embedding (LLE) has been recently proposed [1, 2], which is able to do nonlinear dimensionality reduction in an unsupervised way. Among its advantages over many other

* Olga Kouropteva is grateful to the Infotech Oulu Graduate School.

similar methods are 1) good preservation of local geometry of high-dimensional data in a low-dimensional space, 2) only two parameters to be set, 3) a single global coordinate system in the low-dimensional space, and 4) a non-iterative solution scaling well to large data sets due to a sparse eigenvector problem and thus avoiding local minima inherent to many iterative techniques.

For pattern recognition, however, a class membership of each sample in the training set is usually known in advance and this paved the way for the concept of a supervised LLE, where labels are employed [3, 4]. Thus, the dimensionality reduction became both nonlinear and supervised.

In this paper, we apply two variants of the supervised LLE (SLLE), proposed independently in [3] and [4], in combination with a k nearest neighbor classifier to handwritten digit recognition. The purpose is to see what one can reach while carrying out the recognition in a low-dimensional space generated by SLLE. In addition, we derive an efficient eigendecomposition scheme for SLLE.

2 LLE

As an input, LLE takes a set of N D-dimensional vectors (each of which may represent one image, for example) assembled in a matrix $\boldsymbol{X}$ of size DxN. Its output is another set of N d-dimensional vectors ($d \ll D$) assembled in a matrix $\boldsymbol{Y}$ of size dxN. As a result, the kth column vector of $\boldsymbol{Y}$ corresponds to the kth column vector of $\boldsymbol{X}$. Further we will treat vectors as points either in $\mathbb{R}^D$ or in $\mathbb{R}^d$, depending on the context.

The original, unsupervised LLE consists of three steps:

1. Find K nearest neighbors of each point $\boldsymbol{X}_i$ in $\mathbb{R}^D$, $i = 1, \ldots, N$. The Euclidean distance is used as a similarity measure. Proximity information is collected in a matrix $\boldsymbol{A}$ (of size KxN). The jth column of $\boldsymbol{A}$ stores indices of K points closest to $\boldsymbol{X}_j$ (A_{1j} and A_{Kj} correspond to the highest and lowest proximity, respectively).
2. Assigning weights to pairs of neighboring points. Each weight W_{ij} characterizes a degree of closeness of $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$. The following optimization task must be solved [1]:

$$\varepsilon(\boldsymbol{W}) = \sum_{i=1}^{N} \parallel \boldsymbol{X}_i - \sum_{j=1}^{K} W_{ij \in \boldsymbol{A}_i} \boldsymbol{X}_{j \in \boldsymbol{A}_i} \parallel^2 \ , \tag{1}$$

   subject to constraints $\sum_{j=1}^{K} \boldsymbol{W}_{j \in \boldsymbol{A}_i} = 1$ and $W_{ij} = 0$, if $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ are not neighbors.
3. Computing the low-dimensional embedding. Since the goal of LLE is to preserve a local linear structure of a high-dimensional space as accurately as possible in a low-dimensional space, weights W_{ij} are kept fixed and the following cost function is minimized [1]:

$$\delta(\boldsymbol{Y}) = \sum_{i=1}^{N} \parallel \boldsymbol{Y}_i - \sum_{j=1}^{K} W_{ij \in \boldsymbol{A}_i} \boldsymbol{Y}_{j \in \boldsymbol{A}_i} \parallel^2 \tag{2}$$

under constraints $\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{Y}_i \boldsymbol{Y}_i^T = \boldsymbol{I}$ (normalized unit covariance) and $\sum_{i=1}^{N} \boldsymbol{Y}_i = 0$ (translation-invariant embedding), which provide a unique solution. To find the matrix $\boldsymbol{Y}$ under these constraints, a new matrix is constructed based on the matrix $\boldsymbol{W}$: $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{W})^{\boldsymbol{T}}(\boldsymbol{I} - \boldsymbol{W})$. LLE then computes the bottom $d+1$ eigenvectors of $\boldsymbol{M}$, associated with the $d+1$ smallest eigenvalues. The first eigenvector (composed of 1's) whose eigenvalue is close to zero is excluded. The remaining d eigenvectors yield the final embedding $\boldsymbol{Y}$.

3 Supervised LLE

Being unsupervised, the original LLE does not make use of class membership of each point to be projected. To complement the original LLE, a *supervised* LLE was proposed. Its name implies that membership information influences on which points are included in the neighborhood of each point. That is, the supervised LLE employes prior information about a task to perform dimensionality reduction.

So far, two approaches to the supervised LLE have been proposed. The first approach (abbreviated as SLLE1) forms the neighborhood of $\boldsymbol{X}_i$ *only* from points belonging to the same class as that of $\boldsymbol{X}_i$ [4]. The second approach (abbreviated as SLLE2) expands the interpoint distance if the points belong to different classes; otherwise, the distance remains unchanged [3]. Either approach modifies Step 1 of the original LLE, while leaving other two steps unchanged.

3.1 SLLE1

Suppose that the first N_1 columns of $\boldsymbol{X}$ are occupied by the data of the first class, the next N_2 columns are composed of the data of the second class, etc., i.e. data of a certain class are compactly stored in $\boldsymbol{X}$. This does not affect the algorithm logic, but simplifies explanation. As a result, we can assume that $\boldsymbol{X}$ is composed of submatrices $\boldsymbol{\Xi}_i$ of size $D\text{x}N_i$, $i = 1, \ldots, L$, where L is the number of different classes.

The nearest neighbors for each $\boldsymbol{X}_j \in \boldsymbol{\Xi}_i$ are then sought in $\boldsymbol{\Xi}_i$ only. When applied to all $\boldsymbol{X}_j$'s $\in \boldsymbol{\Xi}_1$, this procedure leads to a construction of the matrix $\boldsymbol{A}_1$. By repeating the same for $\boldsymbol{\Xi}_2, \ldots, \boldsymbol{\Xi}_L$, matrices $\boldsymbol{A}_2, \ldots, \boldsymbol{A}_L$ are generated. Because each $\boldsymbol{A}_i$ contains indices of points and it was constructed independently of other matrices, it is obvious that several matrices can have identical elements, however, referring to different points.

To distinguish points belonging to different classes, we add a shift to values of all elements of the matrices starting from $\boldsymbol{A}_2$. The shift value for $\boldsymbol{A}_i$ is computed as $\sum_{j=1}^{i-1} N_j$. Such a procedure guarantees that no two matrices $\boldsymbol{A}_i$ and $\boldsymbol{A}_j$ will make reference to the same point.

Having set elements of all matrices, we then concatenate $\boldsymbol{A}_i$'s into a single matrix $\boldsymbol{A}$ whose size is $K\text{x}N$, where now $N = \sum_{i=1}^{L} N_i$. We also concatenate $\boldsymbol{\Xi}_i$'s into a single matrix $\boldsymbol{X}$ whose size is $D\text{x}N$.

3.2 SLLE2

Another alternative consists of increasing the interpoint distance in $\mathbb{R}^D$ if points belong to different classes. In doing this, different classes can become more spatially separated. It implies that they will remain spatially separated in $\mathbb{R}^d$ after their mapping, i.e. this will facilitate classification in $\mathbb{R}^d$.

Let $\boldsymbol{\Omega}$ be the initial matrix of interpoint distances in $\mathbb{R}^D$. Then a modification rule is

$$\boldsymbol{\Omega} = \begin{cases} \boldsymbol{\Omega}, & \text{if points belong to the same class} \\ \boldsymbol{\Omega} + \alpha \max(\boldsymbol{\Omega}), & \text{if points belong to different classes.} \end{cases}$$

That is, the magnitude of a distance expansion is parameterized by α which belongs to [0,1]. The smaller (larger) α, the less (more) class labels affect a choice of the nearest neighbors for each point.

After adjusting distances, K nearest neighbors are sought as in case of the original LLE.

4 Efficient Eigendecomposition for SLLE1

Because $\boldsymbol{M}$ is sparse, eigenvector computation is quite efficient, though for large N's it anyway remains the most time-consuming step. However, it is still possible to exploit the mathematical structure of $\boldsymbol{M}$ (at least in case of SLLE1) to efficiently compute eigendecompositions when N is large.

For SLLE1, $\boldsymbol{M}$ *always* has a block diagonal form, where one block corresponds to one class. That is, in general, if there are L classes, there are L blocks. This is because the data of each class are compactly stored in $\boldsymbol{X}$ so that nonzero entries in $\boldsymbol{M}$ are localized within a range of indices allocated for each class.

Let us consider $\boldsymbol{M}$ (of size NxN) as in (3) when the number of classes is equal to 3 ($N = l + m + n$):

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{A}^{l\times l} & \mathbf{0}^{l\times m} & \mathbf{0}^{l\times n} \\ \mathbf{0}^{m\times l} & \boldsymbol{B}^{m\times m} & \mathbf{0}^{m\times n} \\ \mathbf{0}^{n\times l} & \mathbf{0}^{n\times m} & \boldsymbol{C}^{n\times n} \end{pmatrix} \ , \tag{3}$$

where $\boldsymbol{A}^{l\times l}$, $\boldsymbol{B}^{m\times m}$, $\boldsymbol{C}^{n\times n}$ are $l \times l$, $m \times m$, $n \times n$ blocks, respectively.

We chose three classes only for simplicity of explanation and everything discussed below is equally valid for an arbitrary number of classes.

Let us assume that the eigenproblems for all blocks are written as follows:

$$\boldsymbol{Ax} = \lambda \boldsymbol{x},\ \boldsymbol{By} = \mu \boldsymbol{y},\ \boldsymbol{Cz} = \nu \boldsymbol{z} \ , \tag{4}$$

where pairs $(\boldsymbol{x}\,\lambda)$, $(\boldsymbol{y}\,\mu)$ and $(\boldsymbol{z}\,\nu)$ stand for the eigenvectors and eigenvalues for $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, respectively.

Now let us show that the following proposition is true.

Proposition 1. *Given $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ as solutions of the respective eigenproblems in (4),*

1. *the vectors* $\tilde{\boldsymbol{x}} = (\boldsymbol{x}^{l\times 1}\mathbf{0}^{(m+n)\times 1})$, $\tilde{\boldsymbol{y}} = (\mathbf{0}^{l\times 1}\boldsymbol{y}^{m\times 1}\mathbf{0}^{n\times 1})$, $\tilde{\boldsymbol{z}} = (\mathbf{0}^{(l+m)\times 1}\boldsymbol{z}^{n\times 1})$ *are the eigenvectors of* $\boldsymbol{M}$,
2. *the eigenvalues of* $\boldsymbol{M}$ *include those of* $\boldsymbol{A}$, $\boldsymbol{B}$ *and* $\boldsymbol{C}$.

Proof. Multiply first $\boldsymbol{M}$ by $\tilde{\boldsymbol{x}}$:

$$\boldsymbol{M}\tilde{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{A}^{l\times l} & \mathbf{0}^{l\times m} & \mathbf{0}^{l\times n} \\ \mathbf{0}^{m\times l} & \boldsymbol{B}^{m\times m} & \mathbf{0}^{m\times n} \\ \mathbf{0}^{n\times l} & \mathbf{0}^{n\times m} & \boldsymbol{C}^{n\times n} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{l\times 1} \\ \mathbf{0}^{m\times 1} \\ \mathbf{0}^{n\times 1} \end{pmatrix} = \begin{pmatrix} (\boldsymbol{A}\boldsymbol{x})^{l\times 1} \\ \mathbf{0}^{m\times 1} \\ \mathbf{0}^{n\times 1} \end{pmatrix} = \begin{pmatrix} \lambda\boldsymbol{x}^{l\times 1} \\ \mathbf{0}^{m\times 1} \\ \mathbf{0}^{n\times 1} \end{pmatrix} = \lambda\tilde{\boldsymbol{x}} \ . \tag{5}$$

Since $\boldsymbol{M}\tilde{\boldsymbol{x}} = \lambda\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{x}}$ is the eigenvector of $\boldsymbol{M}$ and λ is the eigenvalue of $\boldsymbol{M}$.

Next, multiply $\boldsymbol{M}$ by $\tilde{\boldsymbol{z}}$:

$$\boldsymbol{M}\tilde{\boldsymbol{z}} = \begin{pmatrix} \boldsymbol{A}^{l\times l} & \mathbf{0}^{l\times m} & \mathbf{0}^{l\times n} \\ \mathbf{0}^{m\times l} & \boldsymbol{B}^{m\times m} & \mathbf{0}^{m\times n} \\ \mathbf{0}^{n\times l} & \mathbf{0}^{n\times m} & \boldsymbol{C}^{n\times n} \end{pmatrix} \begin{pmatrix} \mathbf{0}^{l\times 1} \\ \mathbf{0}^{m\times 1} \\ \boldsymbol{z}^{n\times 1} \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{l\times 1} \\ \mathbf{0}^{m\times 1} \\ (\boldsymbol{C}\boldsymbol{z})^{n\times 1} \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{l\times 1} \\ \mathbf{0}^{m\times 1} \\ \nu\boldsymbol{z}^{n\times 1} \end{pmatrix} = \nu\tilde{\boldsymbol{z}} \ . \tag{6}$$

Since $\boldsymbol{M}\tilde{\boldsymbol{z}} = \nu\tilde{\boldsymbol{z}}$, $\tilde{\boldsymbol{z}}$ is the eigenvector of $\boldsymbol{M}$ and ν is the eigenvalue of $\boldsymbol{M}$.

Finally, multiply $\boldsymbol{M}$ by $\tilde{\boldsymbol{y}}$:

$$\boldsymbol{M}\tilde{\boldsymbol{y}} = \begin{pmatrix} \boldsymbol{A}^{l\times l} & \mathbf{0}^{l\times m} & \mathbf{0}^{l\times n} \\ \mathbf{0}^{m\times l} & \boldsymbol{B}^{m\times m} & \mathbf{0}^{m\times n} \\ \mathbf{0}^{n\times l} & \mathbf{0}^{n\times m} & \boldsymbol{C}^{n\times n} \end{pmatrix} \begin{pmatrix} \mathbf{0}^{l\times 1} \\ \boldsymbol{y}^{m\times 1} \\ \mathbf{0}^{n\times 1} \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{l\times 1} \\ (\boldsymbol{B}\boldsymbol{y})^{m\times 1} \\ \mathbf{0}^{n\times 1} \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{l\times 1} \\ \mu\boldsymbol{y}^{m\times 1} \\ \mathbf{0}^{n\times 1} \end{pmatrix} = \mu\tilde{\boldsymbol{y}} \ . \tag{7}$$

Since $\boldsymbol{M}\tilde{\boldsymbol{y}} = \mu\tilde{\boldsymbol{y}}$, $\tilde{\boldsymbol{y}}$ is the eigenvector of $\boldsymbol{M}$ and μ is the eigenvalue of $\boldsymbol{M}$. □

It means that $\boldsymbol{M}$ and block matrices composing it have common eigenvalues and the eigenvectors of $\boldsymbol{M}$ can be easily derived from those of the individual blocks by inserting zeroes in appropriate positions.

Remember that we need to compute the bottom $d+1$ eigenvectors of $\boldsymbol{M}$ corresponding to the $d+1$ smallest eigenvalues in order to find the embedding. According to [4], the value for d should be *less by one than the number of classes*. This condition leads to the minimal (zero) cost in (2), which we are interested in. However, how to select appropriate eigenvectors if we would like to work with blocks instead of the whole $\boldsymbol{M}$?

It turned out that the smallest $d+1$ eigenvalues of $\boldsymbol{M}$ are clustered near zero. When very small eigenvalues are sought as in our case, this may lead to ill-conditioned eigenvalues and eigenvectors, i.e. those sensitive to small perturbations in $\boldsymbol{M}$. However, even if each individual eigenvector may be sensitive to such perturbations, the eigenspace spanned by *all* the eigenvectors associated with the clustered eigenvalues is not! This fact is confirmed by a large difference in magnitude for the smallest $(d+1)$th and $(d+2)$th (and next) eigenvalues of $\boldsymbol{M}$.

SLLE1, applied to $\boldsymbol{M}$, yields the bottom $d+1$ piecewise constant eigenvectors similar to pulse functions[1]. Each eigenvector has one "pulse" corresponding to

[1] Similar shapes of the eigenvectors were also observed while performing the spectral clustering [5] when the data were optimally partitioned into clusters according to a certain criterion. The respective eigenvalues were clustered as well [6].

the data of a certain class and different "pulses" do not intersect each other since the eigenvectors are mutually orthogonal. It implies that different classes are *linearly* separated in $\mathbb{R}^d$.

Now let us turn attention to the eigenproblems of the individual blocks of $\boldsymbol{M}$. We figured out that each block has one distinct eigenvalue which is very small compared to others. It came as no surprise that the eigenvector associated with that eigenvalue belonged to the subspace found for $\boldsymbol{M}$. As a result, we only have to compute *one* eigenvector (corresponding to the smallest eigenvalue) per block in $\boldsymbol{M}$ since the number of classes is equal to that of blocks.

By summarizing the abovementioned, an efficient decomposition for $\boldsymbol{M}$ in case of SLLE1 can be found in block-by-block fashion. The eigenvectors of the blocks are to be padded by zeroes. A precise location for padding is determined by a block index as shown in Proposion 1. In particular, if L is the number of blocks, then the eigenvector taken from the jth $(1 < j < L)$ block should be padded by $\sum_{i=1}^{j-1} N_i$ zeroes from the front and $\sum_{i=j+1}^{L} N_i$ zeroes from the end, where N_i is the size of the ith block.

5 Experiments

To compare LLE, SLLE1 and SLLE2, we took all samples of "1", "3", "7", "8", and "9" from the MNIST database of handwritten digits [7]. Because of large memory requirements for LLE and SLLE2, only these five classes were used and the MNIST training/test sets were interchanged.

When picking classes, we aimed at selecting digits of similar shapes, such as 1 and 7 or 3, 8 and 9 in order to make the task of digit recognition more challenging. The training set comprised 5,000 samples whereas the test set consisted of 30,000 samples. Entries to the matrix $\boldsymbol{X}$ were raw grayscale pixel values without preprocessing.

We ran each algorithm in order to map the training set into $\mathbb{R}^d$ with K=10 and d=4 (because of 5 classes; see Sect. 4). Given such parameter values, *all* samples belonging to the same class were projected to one point in $\mathbb{R}^d$ in case of SLLE1 and SLLE2 ($\alpha = 0.3, \ldots, 1$). For LLE and SLLE2 ($\alpha = 0.1, \ldots, 0.2$), projections formed "clouds".

The samples from the test set were then mapped into $\mathbb{R}^d$ by using the non-parametric generalization [2] with K set to 10. Knowing the coordinates of the test samples in $\mathbb{R}^d$, a k (k=15) nearest neighbor classification was carried out in $\mathbb{R}^d$ to classify them (since d is small, such a large value for k almost did not affect the search time, while better classification was achieved). Results of classification are presented in Tables 1-4, where the last columns show the accuracy rates attained in classifying a certain class of digits.

One can see that with given parameters, the original LLE lost to both supervised algorithms when classifying data in the embedded space. The average accuracy rate for SLLE2 varied depending on α and attained the highest value at α=0.3, which afterwards did not change as α grew. This highest rate is indentical to that achieved with the *non-parametric* SLLE1. Classification results for

Table 1. Confusion matrix (LLE). Average accuracy rate - 93.83%

	1	3	7	8	9	
1	6,654	27	34	22	5	98.7%
3	19	5,577	37	460	38	91.0%
7	66	11	5,870	22	296	93.7%
8	94	236	36	5,396	89	92.2%
9	17	73	198	95	5,566	93.6%

Table 2. Confusion matrix (identical for SLLE1 and SLLE2 with $\alpha = 0.3, \ldots, 1$). Average accuracy rate - 95.86%

	1	3	7	8	9	
1	6,698	12	21	2	9	99.3%
3	149	5,847	49	61	25	95.4%
7	101	2	6,067	1	94	96.8%
8	265	95	16	5,410	65	92.5%
9	110	51	104	15	5,669	95.3%

Table 3. Confusion matrix (SLLE2 with $\alpha = 0.1$). Average accuracy rate - 95.48%

	1	3	7	8	9	
1	6,630	18	69	2	23	98.3%
3	16	5,669	46	315	85	92.5%
7	41	13	5,865	7	339	93.6%
8	63	25	63	5,594	106	95.6%
9	6	46	60	44	5,793	97.4%

Table 4. Confusion matrix (SLLE2 with $\alpha = 0.2$). Average accuracy rate - 94.66%

	1	3	7	8	9	
1	6,683	12	35	2	10	99.1%
3	27	5,675	378	35	16	92.6%
7	58	2	6,087	0	118	97.2%
8	111	35	452	5,218	35	89.2%
9	19	26	235	7	5,662	95.2%

Table 5. Confusion matrix (k nearest neighbors in $\mathbb{R}^D$). Average accuracy rate - 94.02%

	1	3	7	8	9	
1	6,701	15	16	1	9	99.4%
3	99	5,783	89	82	78	94.3%
7	175	4	5,950	3	133	95.0%
8	278	203	45	5,121	204	87.5%
9	63	79	202	19	5,669	93.9%

all methods depend on k. The Euclidean metric, lacking the robustness to noise and outliers, mainly attributed to the large k we used. As to K, only SLLE1 was truly insensitive to its choice while LLE and SLLE2 were completely or partly dependent on K.

Finally, in order to see to what extent the curse of dimensionality was reduced when doing the classification in $\mathbb{R}^d$, the k nearest neighbor classification was also done in $\mathbb{R}^D$ with $k = 15$ (Table 5). It turned out that the average accuracy rate when classifying in the LLE-reduced space was slightly worse than that in the original space. It means that LLE should not be employed for classification (this reasoning was also given in [3]). In contrast, the average accuracy achieved in the SLLE-reduced space was superior to that in $\mathbb{R}^D$, while the time spent was approximately the same in both cases.

6 Conclusion

In this paper, we compared two algorithms (SLLE1 and SLLE2) for supervised dimensionality reduction augmented with the capability to generalize to new data. Since they employ class labels when performing a mapping $\mathbb{R}^D \rightarrow \mathbb{R}^d$, they were coupled with a k nearest neighbor classifier in order to test their performance on handwritten digit images. Experiments demonstrated that SLLE and k nearest neighbors outperforms k nearest neighbors operating alone in the original space.

The efficient eigendecomposition scheme was also proposed in case of SLLE1. Such a scheme dramatically reduces memory requirements, since only one block at a time will be processed. It means that a large $\boldsymbol{M}$ is not a restriction for SLLE1. In addition, for large N's such a strategy can lower the time spent on the eigendecomposition.

References

1. Roweis, S., Saul, L.: Nonlinear dimensionality reduction by locally linear embedding. Science **290** (2000) 2323–2326
2. Saul, L., Roweis, S.: Think globally, fit locally: unsupervised learning of nonlinear manifolds. Technical Report MS CIS-02-18, University of Pennsylvania (2002)
3. de Ridder, D., Duin, R.: Locally linear embedding for classification. Technical Report PH-2002-01, Delft University of Technology (2002)
4. Kouropteva, O., Okun, O., Pietikäinen, M.: Classification of handwritten digits using supervised locally linear embedding algorithm and support vector machine. In: Proc. of the 11th European Symp. on Artificial Neural Networks, Bruges, Belgium. (2003)
5. Meilă, M., Shi, J.: Learning segmentation by random walks. In Leen, T., Dietterich, T., Tresp, V., eds.: Advances in Neural Information Processing Systems. Volume 13. MIT Press, Cambridge, MA (2001) 873–879
6. Shawe-Taylor, J., Cristianini, N.: One the concentration of spectral properties. In Dietterich, T., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems. Volume 14. MIT Press, Cambridge, MA (2002) 511–517
7. (http://yann.lecun.com/exdb/mnist/index.html)