Université Paris Descartes
Faculté d'informatique

Rapport de projet

Module : Réduction de dimension

Master 2 : MLDS

Implémentation d'une méthode de réduction de dimensions combinée et comparaisons sur des benchmarks

- Réalisé par :

  Nazim Messous

  Wissam Benhaddad

04-01-2020

1 Introduction

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

2 The importance of Dimensionality Reduction

The Curse of Dimensionality. is when we add as many features as possible to get more and more information at first, but then the model will decrease in term of performance. Because of the high number of features ,the dimensionality increase and the space grow and becomes more and more sparse. In consequences, it is often easier for the model to find a perfect-match solution which most likely leads to over-fitting.

3 State of the art

Dimensionality reduction can be subsided into two families:

- Linear methods : The most common and well known dimensionality reduction methods are the ones that apply linear transformations, the method performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized.
- Non linear method : Non-linear transformation methods or manifold learning methods are used when the data does not lie on a linear subspace. It is based on the manifold hypothesis which says that in a high dimensional structure, most relevant information is concentrated in small number of low dimensional manifolds.

We can say that both methods consider that sometimes the dataset dimensionality is superficially high, and that we can reduce it without removing too much important information.

4 Linear approach : PCA

Principal component analysis (PCA) is a technique used for identification of a smaller number of uncorrelated variables known as principal components from a larger set of data, this transformation is defined in such a way that the first principal component has the largest possible variance. We can also say that PCA is an approach when we drop the "least important features" . It is the most common linear approach for dimensionality reduction.

- Organize the data set : create a matrix with n samples, each of them is described by p variables.
- Calculate the empirical mean

$$u_j = \frac{1}{n} \sum_{i=1}^{n} X_{ij} \tag{1}$$

- Calculate the deviations from the mean :

$$B = X - hu^T \tag{2}$$

where h is an $n \times 1$ column vector of 1's

- Calculate the co-variance matrix

$$C = \frac{1}{n-1} B * B \tag{3}$$

- Find the eigenvectors and eigenvalues of C.

$$V^{-1} C V = D \tag{4}$$

V is the eigenvectors matrix

- Reorder the eigenvectors and eigenvalues in terms of of decreasing eigenvalue.
- Select a subset of the eigenvectors as basis vectors Select the the L biggest eigenvalues where $L << p$ and their corresponding eigenvectors.

Here one exemple you can find the rest in the attached notebook PCA_on_fcps.ipynb (the original on the left and the reconstructed one in the right):

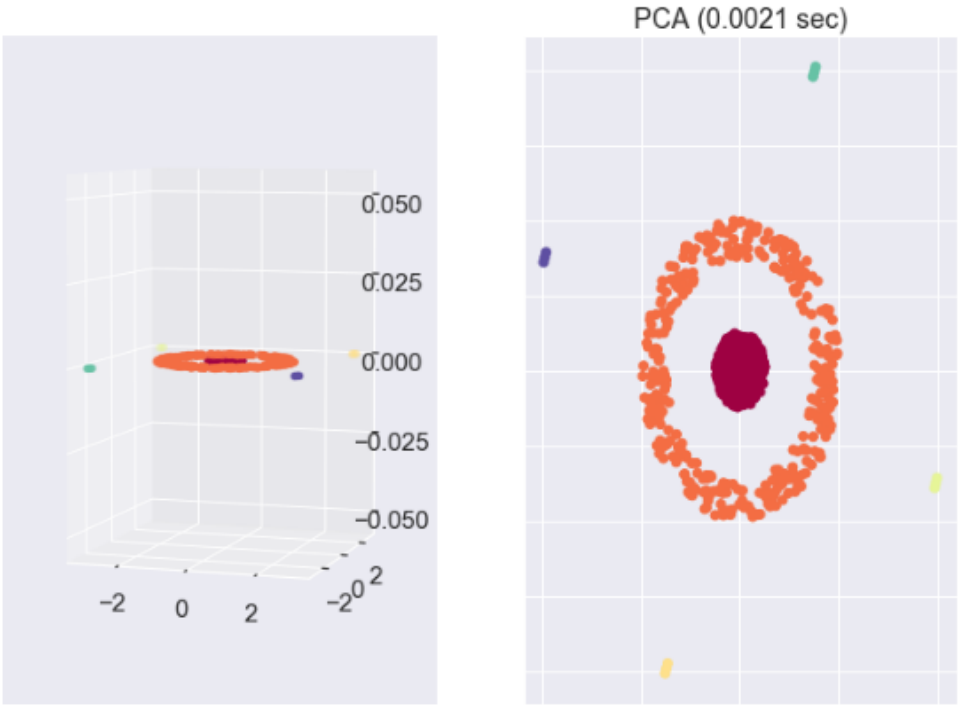

Figure 1: exemple with PCA on the dataset target

5 Non linear approach

5.1 Multidimensional Scaling (MDS)

The goal of MDS is that from dissimilarities between pairs of objects, we represent these dissimilarities as distance in a lower dimension (such that the distances correspond as closely as possible to the dissimilarities)

5.1.1 metric MDS

The classical MDS version used for quantitative data has showed how starting with a matrix of distances between points in a Euclidean space, coordinates for the points can be found such that distances are preserved. Loss function is called *strain* :

$$Strain_D = (\frac{\sum_{ij}(b_{ij} - (x_i, x_j))^2}{\sum_{ij} b_{ij}^2})^{\frac{1}{2}} \tag{5}$$

The steps are :

- create the squared proximity matrix

$$D^{(2)} = [d_{ij}^2] \tag{6}$$

- Apply double centering:

$$B = -\frac{1}{2} J D^{(2)} J \tag{7}$$

- Determine the m largest eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$ and the corresponding eigenvectors $e_1, e_2, ...e_m$
- We find the new coordinates with the formula below where E_m is the matrix of m eigenvectors Λ_m is the diagonal matrix of m eigenvalues of B .

$$X = E_m \Lambda_m^{\frac{1}{2}} \tag{8}$$

5.1.2 Non metric MDS

Non-metric MDS is used for ordinal data. It tries to keep the order of dissimialrity metrics intact, but we'll work with the classical metric version of MDS sot we will not dwell on it any longer. Here one exemple you can find the rest in the attached notebook MDS_on_fcps.ipynb (the original on the left and the reconstructed one in the right):

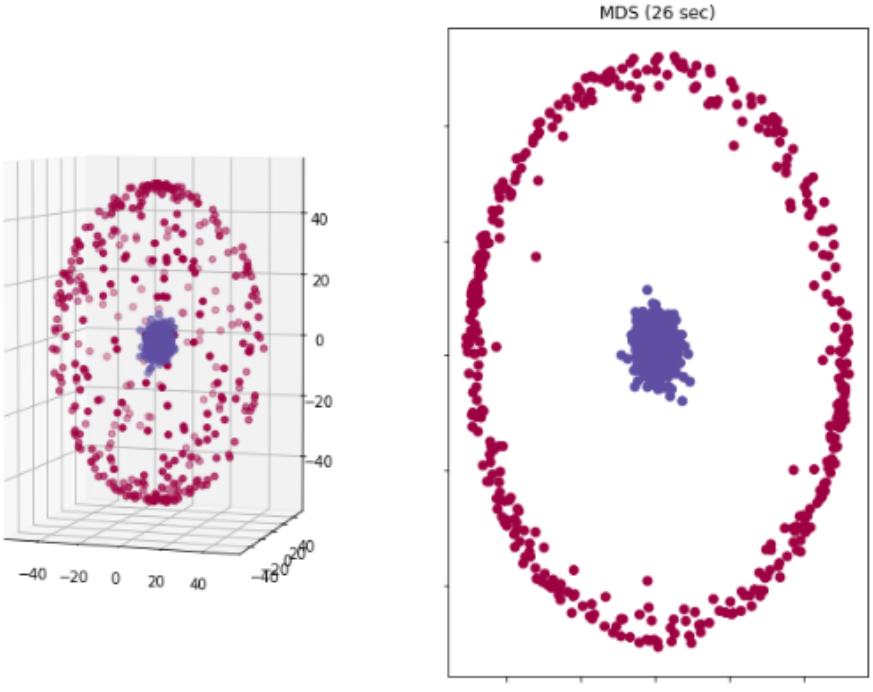

Figure 2: exemple with MDS on the dataset Atom

5.2 Locally Linear Embedding (LLE)

5.2.1 Definition

Locally Linear Embedding is a method to find low-dimensional global coordinate when the data lies on a manifold structure it is based on the fact that locally the manifold looks linear so the LLE algorithm seeks to find a linear dimensionality reduction at each point.

5.2.2 Steps

- Finding the neighborhoods: The first step of LLE is to create a neighborhood graph, using a distance metric like Euclidean, Mahalanobis, hamming etc. The neighborhood graph can be created using many strategies :

  1. e-neighbourhood
  2. K-nearest neighbors
  3. Locality-sensitive hashing

- Finding the weights: This method is named as locally linear embedding, because of the hypothesis that the manifold is exactly linear around the point x_i (and we know that a manifold is locally linear, by taking a sufficiently small region around each point), we can reconstruct x_i as a weighted sum of its neighbors. Reconstruction is achieved by linear coefficients of weights :

$$\vec{x_i} = \sum(wij\vec{x_j}) \tag{9}$$

With the constraint that :

$$\sum_j(w_{ij}) = 1 \tag{10}$$

- Finding the coordinates The neighborhood graph and the local Weight matrix capture the topology of the manifold. the purpose is to find coordinates Y which minimize the loss function :

$$\Phi(Y) = \sum_i \|(\vec{y_i} - \sum_{i \neq j}(\vec{y_i}wij))\| \tag{11}$$

  This equation constrains the output to be centered at origin making it translation invariant:

$$\sum_i(X_i) = 0 \tag{12}$$

  And this equation is required to make the output rotation and scaling invariant :

$$XX^T = I \tag{13}$$

Here one exemple you can find the rest in the attached notebook LLE_on_FCPS.ipynb (the original on the left and the reconstructed one in the right):

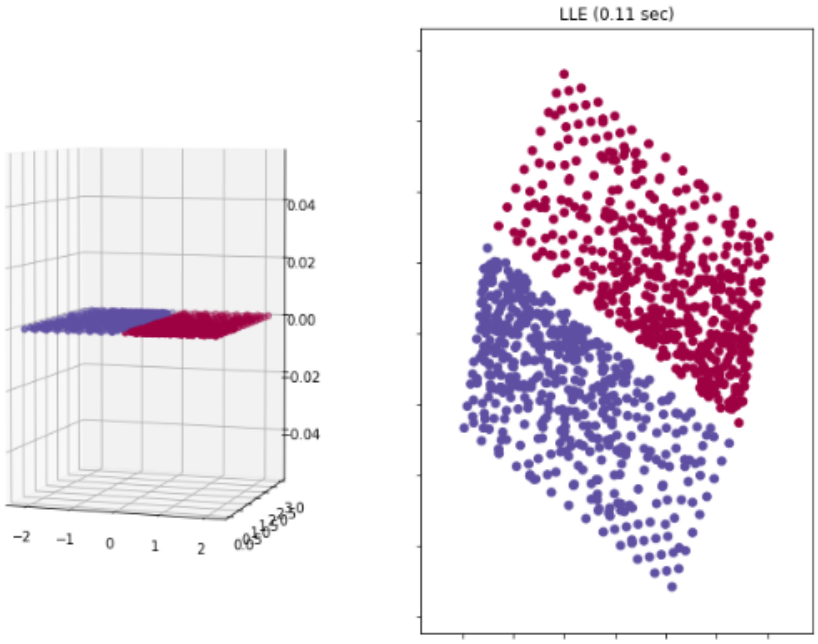

Figure 3: exemple with LLE on the dataset WingNut

5.3 Eigenmaps

Eigenmaps is a non-linear embedding. The main idea is to preserve local information only, in other words construct graph from data points and projecting them into a low dimension space using "eigenvectors of the graph".

- Graph Construction Similarity Graphs: Model local neighborhood relations between data points We construct the similarity graph $G(V, E, W)$, such as:
  - V represent the vertices of the graph (data points)
  - E represent the edges, we add a edge between two point if the distance between the two points is less than a threshold epsilon (ϵ–neighborhood graph strategy) or by choosing the k nearest neighbors.
  - W are the weights of the edges E.g. 1 if connected, 0 otherwise.
- Embed using Graph Laplacian:
  -W is the Weight matrix
  -D is Degree matrix = diag(d1,....,dn)

$$d_i = \sum_j w_{i,j} \tag{14}$$

  -L is the laplacian graph L=D-W
  After getting the Laplacian matrix we solve generalized eigenvalue problem

$$Lf = \lambda Df \tag{15}$$

  Finally we order the eigenvalues, and we embed data points in d-dim space, project data points into eigenvectors associated with $\lambda_1, \lambda_3, ..., \lambda_{d+1}$, with $d <<< D$ (we ignore the first eigenvector)
  note: Unlike PCA, eigenvectors directly gives the embedding of data points.

Here one exemple you can find the rest in the attached notebook eigenmaps_on_FCPS.ipynb (the original on the left and the reconstructed one in the right):

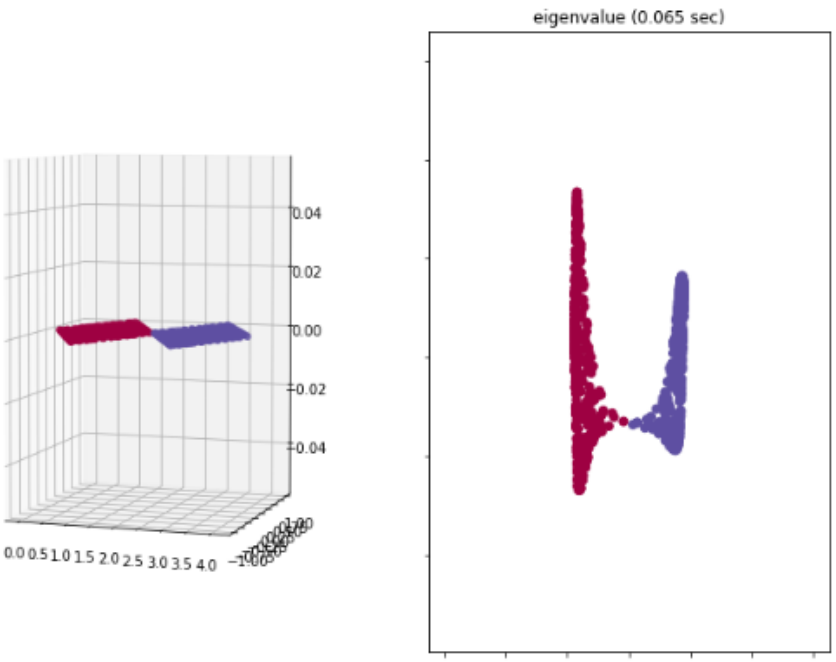

Figure 4: exemple with eigenmaps on the dataset TWoDiamonds

5.4 isometric mapping (ISOMAP)

Isomap is a non-linear dimensionality reduction method based on the spectral theory which tries to preserve the geodesic distances in the lower dimension.

5.4.1 Why geodesic distances are better than the Euclidean distances in nonlinear manifolds?

If we look at the Euclidean mapping, we can clearly see that that point like a and c ave been mapped poorly, only the points which can be approximated to lie on a linear manifold(c and d) give satisfactory results, in the other hand the the mapping with geodesic distances approximate nicely close points as neighbors and far away points as distant.The geodesic distances use graph distance algorithm like Djikstra to approximate the distance between two points, while Euclidean distances should not be used for approximating the distance between two points in non-linear manifolds.

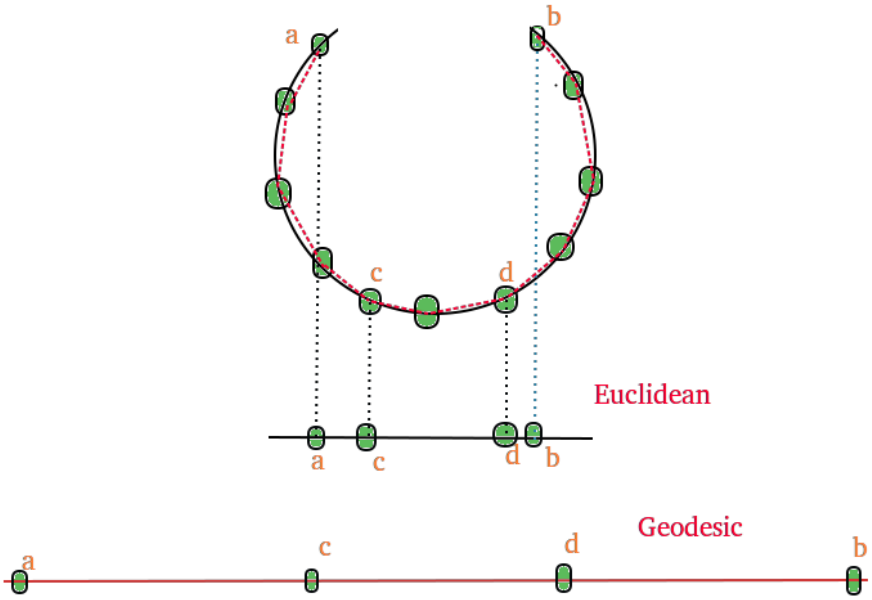

Figure 5: The difference between Euclidean and Geodesic distance in terms of proportions conservation

5.4.2 Steps

- Determine the neighbors of each point using K nearest neighbors method or the ϵ-neighborhood method .
- Construct a neighborhood graph.
- Compute shortest path between two nodes:(Dijkstra's algorithm,Floyd–Warshall algorithm..)
- Compute lower-dimensional embedding (using the MDS approach)

Here one exemple you can find the rest in the attached notebook ISOMAP_on_FCPS.ipynb (the original on the left and the reconstructed one in the right):

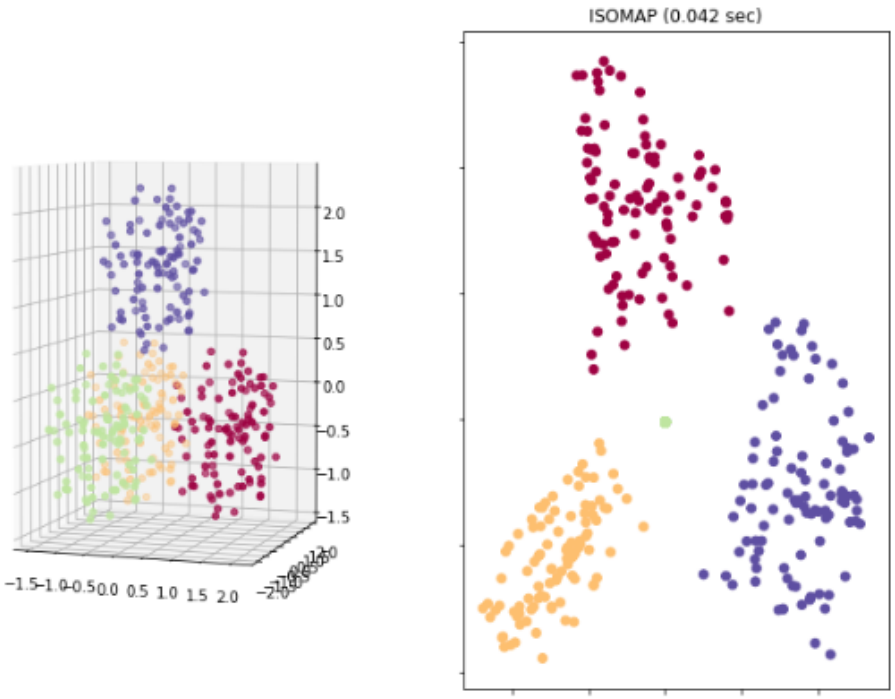

Figure 6: exemple with ISOMAP on the dataset Tetra

5.5 Deep Antoencoder:

An Autoencoder is a neural network that takes input data with a very high dimensionality and learns to copy its input to its output. The Antoencoder take place in two step :

- The first step is the encoding step :
  The encoder corresponds to a heap of layers, it can be compose of fully connected layers or convolutional layer that will take the input and compress it until get a smaller representation of the input, this is what called the bottleneck
- From the bottleneck, the second part of the algorithm corresponds an attempt to reconstruct the initial data using fully connected or convolutional layers.

The higher the number of dimensions in the latent space, the more the reconstructions will be clear and robust, which nevertheless requires more information in the bottleneck
The cost function of an autoencoder is simply to reconstruct the set of initial data and calculate the reconstruction loss with respect to the input. In his most basic form, the optimal solution of an autoencoder is strongly linked to the PCA
To calculate the values of the hidden layers, we simply multiply the weights between the input and the layer and the input, we have the following formula:

$$z = f(Wx) \tag{16}$$

Likewise, to get the output, we multiply the weights between the hidden layers and the output by the values of the hidden layers, we thus obtain:

$$y = g(Vz) \tag{17}$$

The functions f and g are free, as long as they are non-linear and can be derived for back-propagation.
We therefore obtain the following function:

$$y = g(Vf(Wx)) = VWx \tag{18}$$

Allowing us to obtain the objective function:

$$J = \sum_{n=1}^{N} |x(n) - VWx(n)|^2 \tag{19}$$

This corresponds to the objective function of the PCA.
However, the autoencoder is much more flexible than PCA: each layer has an activation function (Relu, tanh, sigmoide, ..) allowing to introduce the notion of non-linearity, contravention of the PCA which can only represent linear transformations. With constraints of dimensionality and parsimony autoencoders can therefore learn from data projections more interesting than PCA or other basic techniques. In addition, they can be stacked, which makes them even more powerful. Here one exemple you can find the rest in the attached notebook DeepAE_on_FCPS.ipynb (the original on the left and the reconstructed one in the right):

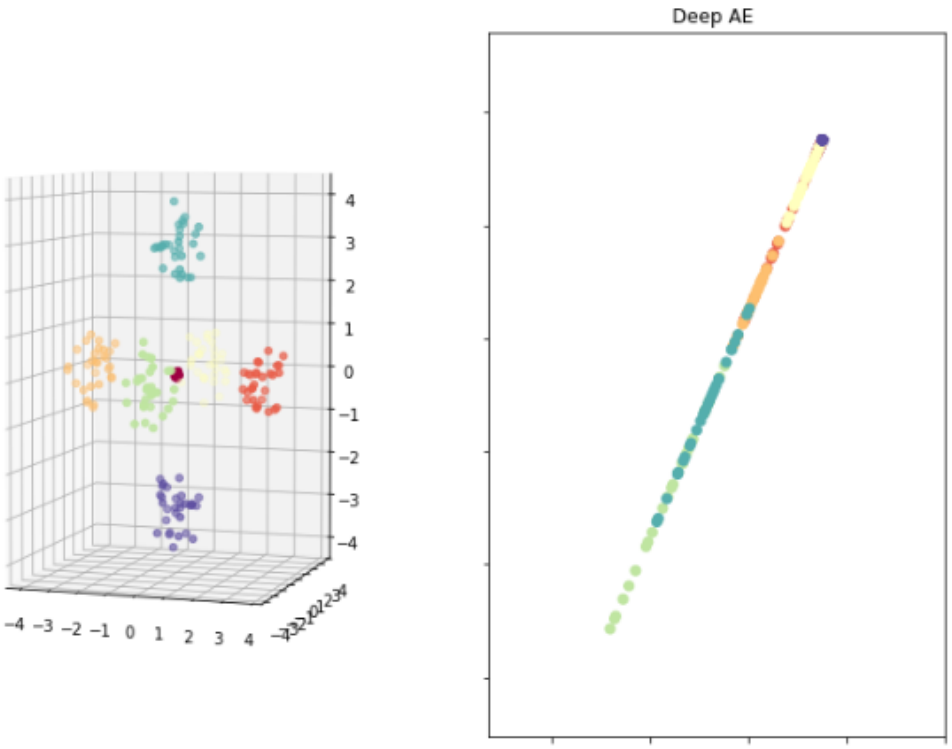

Figure 7: exemple with Deep AE on the dataset Hepta

6 Visualization methods

In this project we will use 2 different method T-SNE for the encoding matrix, and UMAP for the embedding matrix.

6.1 TSNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data. In simpler terms, t-SNE gives you a feel or intuition of how the data is arranged in a high-dimensional space.
The t-SNE algorithm calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space. It then tries to optimize these two similarity measures using a cost function. Let's break that down into 3 basic steps.

- Step 1, measure similarities between points in the high dimensional space.

- Step 2 is similar to step 1, but instead of using a Gaussian distribution you use a Student t-distribution with one degree of freedom, which is also known as the Cauchy distribution
- The last step is that we want these sets of probabilities from the low-dimensional space (Q_{ij}) to reflect those of the high dimensional space (P_{ij}) as best as possible

It is however necessary to keep and not to forget that t-SNE is not really a dimension reduction but rather visualization algorithm , t-SNE does not perform a point by point mapping from the original vector space to the space of reduced dimension. Therefore, it is useless to apply methods on the points generated by t-SNE hoping to keep information from the initial space as for the other methods. t-SNE is simply trying to place the points in a reduced space so that the points having a high dissimilarity are distant. The only relationship that t-SNE guarantees unlike other algorithms, is that a point p will be the same in the two vector spaces.

6.2 UMAP

UMAP, at its core, works very similarly to t-SNE, both use graph layout algorithms to arrange data in low-dimensional space. In the simplest sense, UMAP constructs a high dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible.
MAP essentially consist of two steps :

- Building a graph in high dimensions and computing the bandwidth of the exponential probability,but unlike t-sne in this step we dropped the log-part in the definition of the number of nearest neighbors, and we omit normalization of the high-dimensional probability which make UMAP faster.
- Optimization of the low-dimensional representation via Gradient Descent. UMAP actually becomes faster on the second step as well. This improvement is due to the use of Stochastic Gradient Descent (SGD) instead of regular Gradient Descent, we skipped normalization not only for high-dimensional but also for low-dimensional probabilities and high-dimensional probabilities and finally we introduce the local connectivity ρ parameter which glues together (to some extent) the sparse regions.

7 The new Algorithm combining DEEP AE + LLE

Recently, a certain number of works have studied the clustering strategies which combine classical clustering algorithms and reduction methods dimension or deep learning. These approaches often follow a sequential method, where a representation in a dimension space reduced is learned before obtaining groups using a clustering technique.
To reduce the drawback of traditional dimension reduction methods, we propose a new regularized objective function. The proposed method can be seen as a procedure seeking simultaneously a new representation of the data containing the maximum of information (using a Deep AE), and a similarity graph best characterizing the proximity between the points (using LLE). this method consists in optimizing the following criterion:

$$min_{\theta 1,\theta 2,s}||X - g_{\theta 2}(f_{\theta 1}(X))||^2 + \lambda||f_{\theta 1}(X) - f_{\theta 1}(X)||^2 \tag{20}$$

$\theta 1, \theta 2$ are the parameters of the EA encoder and decoder blocks, respectively. S is the weight matrix, calculated with LLE. This objective function is broken down into two terms, the first corresponds to the function objective of an Autoencoder and the second teme corresponds to the objective function of the LLE method. Updating S is done by optimizing the second term, the optimal weights are those that minimize the cost function:

$$\epsilon(S) = \sum_i ||f_{\theta 1}(X_i) - \sum_j s_{ij} f_{\theta 1}(X_j)||^2 \tag{21}$$

under the constraint: the sum of each line of the weighting matrix S is equal to 1:

$$\sum_j s_{ij} = 1 \tag{22}$$

We build S using the same strategy as LLE. We assume that each individual with its k nearest neighbors is on a locally linear variety and it can be reconstructed by a linear combination of its k nearest neighbors. So, S is a matrix with sij indicates the contribution from point j in the reconstruction of i; therefore it is equal to 0 if i and j are not neighbors.

8 Experimental results

The Deep DR algorithm is evaluated on three datasets: MNIST, USPS and Fashion-mnist Theer two datasetd are often used by the machine learning community to compare and validate new algorithms, if it does not work on MNIST and usps, there is little chance so that this is a relevant algorithm, in addition the sparsity of USPS being 0 percent and being an unbalanced dataset, and the mnist being very sparse (80 percent) and a balanced daatset it is interesting to compare the new algorithm on these two data sets, finally mnist and USPS are considered two easy this is why we add the fashion mnist

- USPS : the dataset has 9298 images with 256 features, and 10 different classes of Handwritten Digits
- Mnist:The data set contains 60,000 and 10,000 images respectively size 28x28 of handwritten figures ranging from 0 to 9. The images are grayscale.
- Fashion Mnist:The data set contains 60,000 and 10,000 images respectively size 28x28 of 10 different clothes categories from Zalendo.

8.1 Parameters

8.1.1 Choose the parameters l and the number of neighbors n

The performance of the lle+ deep ae algorithm being directly correlated to influence of its cost function and the number of neighbors selected for the calculation of the matrix S in the LLE step, we first tried a multitude of possible combinations for these two values, and then we evaluated the quality of the models via a K-means clustering.
L determines how much the second term is more or less important than the first one.

8.1.2 Choosing the perplexity value for T-SNE

Perplexity is a measure for information that is defined as 2 to the power of the Shannon entropy. The perplexity of a fair dice with k sides is equal to k. In t-SNE, the perplexity may be viewed as a knob that sets the number of effective nearest neighbors. It is comparable with the number of nearest neighbors k that is employed in many manifold learners.
The performance of t-SNE is fairly robust under different settings of the perplexity. The most appropriate value depends on the density of your data. Loosely speaking, one could say that a larger / denser dataset requires a larger perplexity. Typical values for the perplexity range between 5 and 50.

8.1.3 Why choosing UMAP over TSNE for the embedding matrix

For the embedding matrix we get bad visualization with T-SNE so we choosed instead UMAP, and it worled far much better probably for these reasons

- T-SNE does not scale well for rapidly increasing sample sizes
- T-SNE does not preserve global data structure
- T-SNE can practically only embed into 2 or 3 dimensions
- T-SNE performs a non-parametric mapping
- T-SNE can not work with high-dimensional data directly
- T-SNE consumes too much memory

8.1.4 Choose the number of neighbors for UMAP

Low values of number of neighbors will force UMAP to concentrate on very local structure (potentially to the detriment of the big picture), while large values will push UMAP to look at larger neighborhoods of each point when estimating the manifold structure of the data, losing fine detail structure for the sake of getting the broader of the data

8.1.5 Choose the metrics

- ARI: measures the degree of agreement between an estimated partition and a reference partition
- NMI :quantifies how informative the estimated partition is on the real score,

A high NMI / ARI ratio is preferable and then symbolizes adequate partitioning.

8.2 Mnist dataset

8.2.1 Clustering results

For any combination of parameters l and number of neighbors we get two representation, the encoding matrix and embedding matrix, then we try a clustering via K-means on both of them. We can see from the 2 pictures below (first one is the results of clustering on the encoding matrix and second one is results of clustering on the embedding matrix) that The encoding matrix was a better representation than the embedding matrix.
The nmi and ari were good for the encoding matrix and acceptable for the embedding matrix. We can add that the range of values for ARI and NMI varies just a little for the encoding matrix and varies a lot for the embedding matrix.

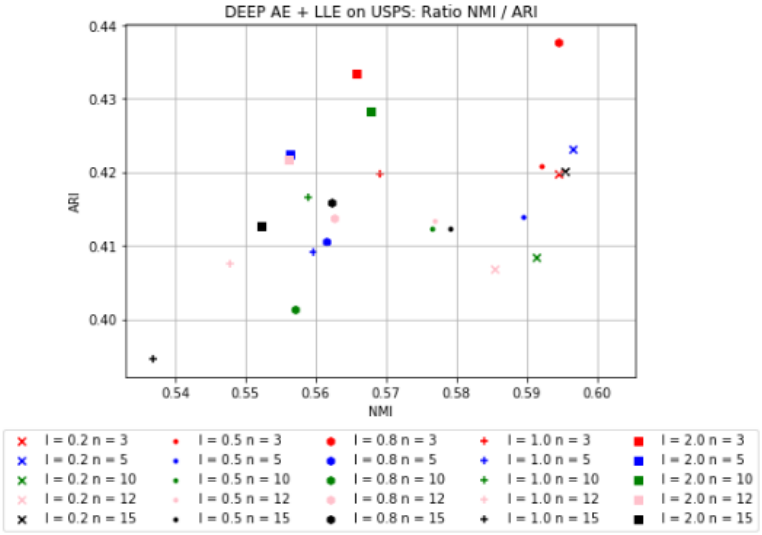

Figure 8: DAE + LEE (encoding matrix representation) on mnist ratio NMI/ARI

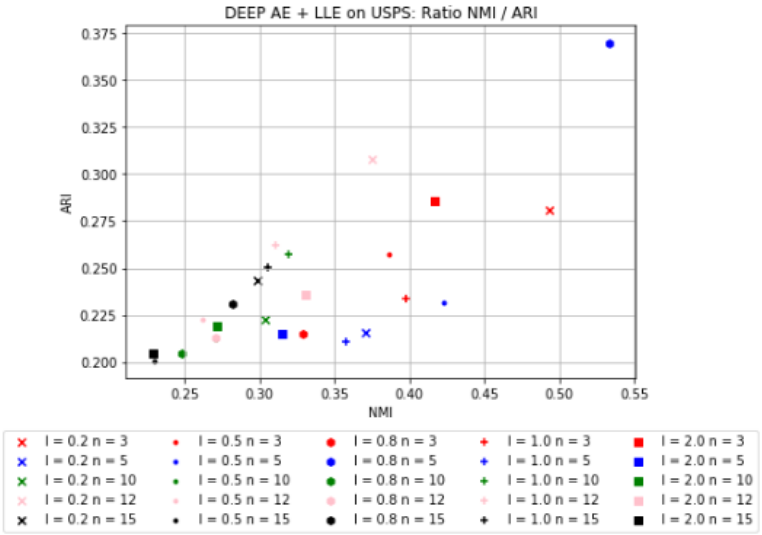

Figure 9: DAE + LEE (Embedding matrix representation) on mnist ratio NMI/ARI

8.2.2 Visualization

Good visualization for clusters given by the encoding matrix, we can distinguish the clusters from one another, still We have clusters which are mixed up with one another as it happens the cluster 4 with 9 and cluster 2 with 8.
Here the 2 best visualization with T-SNE:

8.2.3 Comparing with others methods

We can see that if the antoencoder and PCA didn't gave better NMI ARI and a better visualization than the the new algorithm for both representations (encoding and embedding), the LLE algorithm outperform the clustering on the embedding matrix with any combination of parameters,and it did a little better than the clustering on the the encoding matrix always with any combination of parameters .

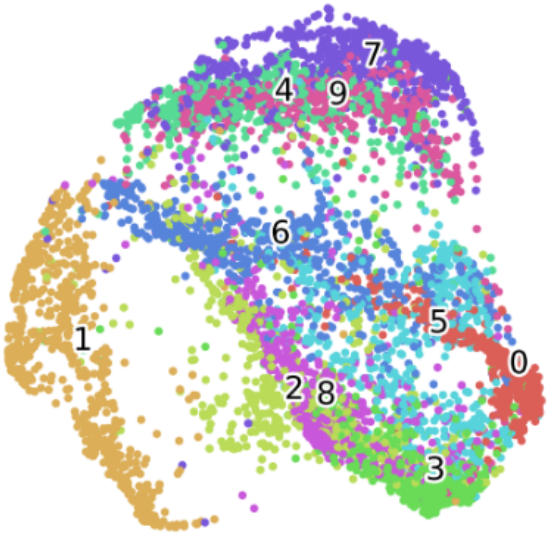

Figure 10: visualization with T-sne of the results obtained after application of a clustering on the encoding matrix with perplexity=10

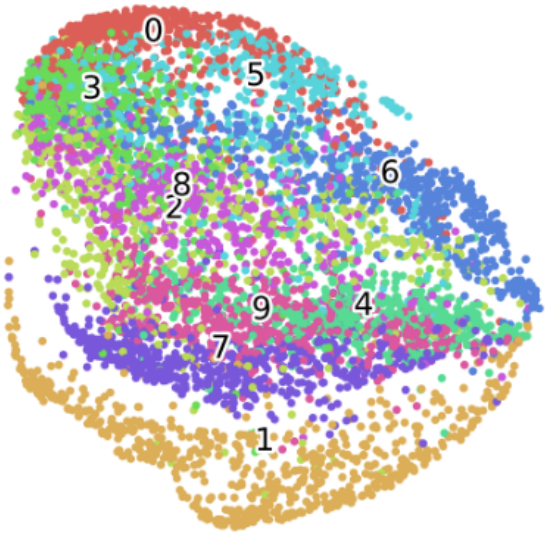

Figure 11: visualization with T-sne of the results obtained after application of a clustering on the encoding matrix with perplexity=30

Algorithm	NMI	ARi
DEEP AE + LLE (embedding representation)	0.55	0.375
DEEP AE + LLE (encoding representation)	0.59	0.44
DEEP AE	0.38	0.24
PCA	0.36	0.22
LLE	0.60	0.47

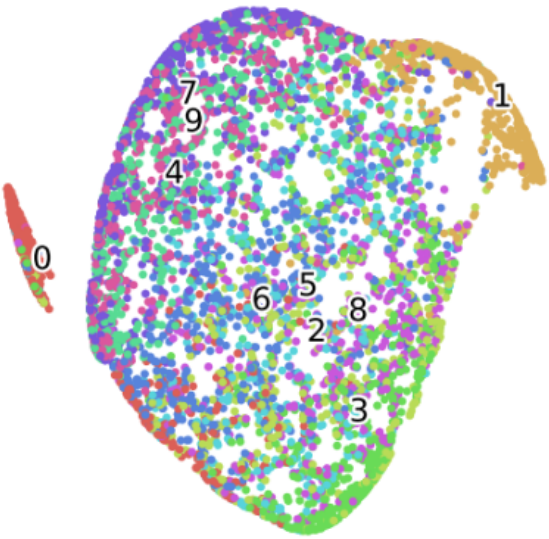

Figure 12: visualization with T-sne of the results obtained after application of a clustering on PCA representation

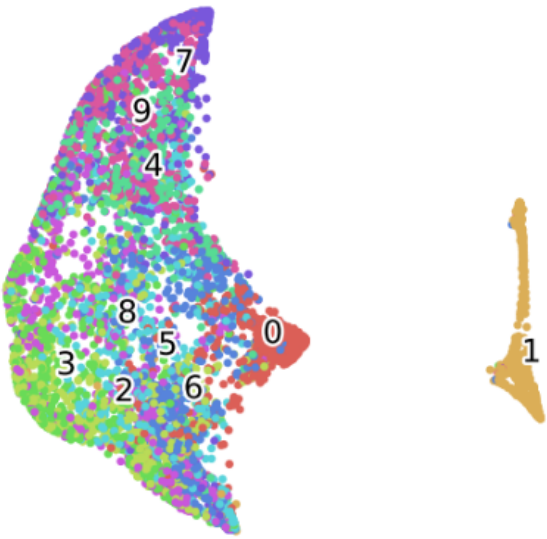

Figure 13: visualization with T-sne of the results obtained after application of a clustering on DEEP AE representation

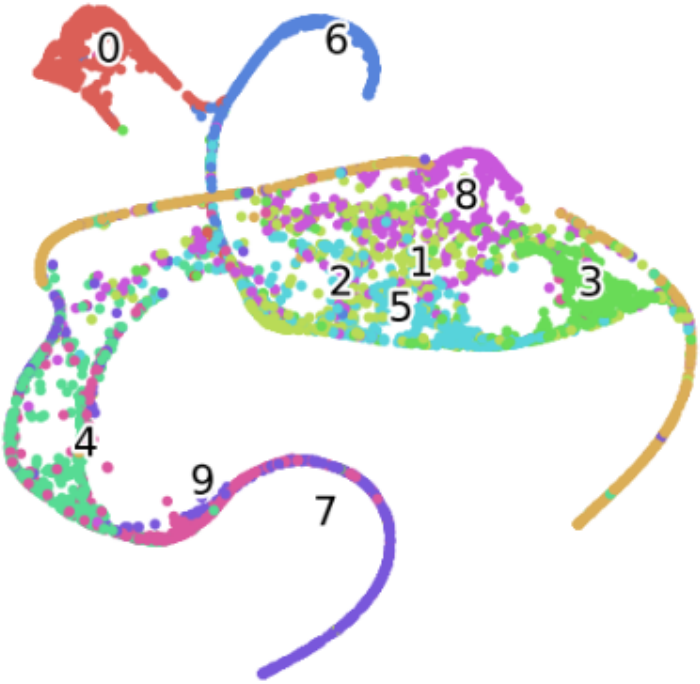

Figure 14: visualization with T-sne of the results obtained after application of a clustering on LLE representation

8.3 USPS dataset

8.3.1 Clustering results

One more time the clustering on the encoding matrix representation outperform in term of NMI and ARI the clustering on the embedding matrix, and we note that for the encoding matrix the range of values for ARI and NMI was quite small from 0.79 to 0.82 for NMI and from 0.71 to 0.79 for the ARI (the results were stable), in the same time the range of nmi and ari values was very large, for exemple if we took the exemple were l=0.8 and n=12 the nmi and ari were equal to 0.2 and for best case nmi was 0.75 and ari more than 0.6, this induce that the embedding representation is unstable and is less trustworthy than the encoding representation in addition to having worse NMI and ARI.

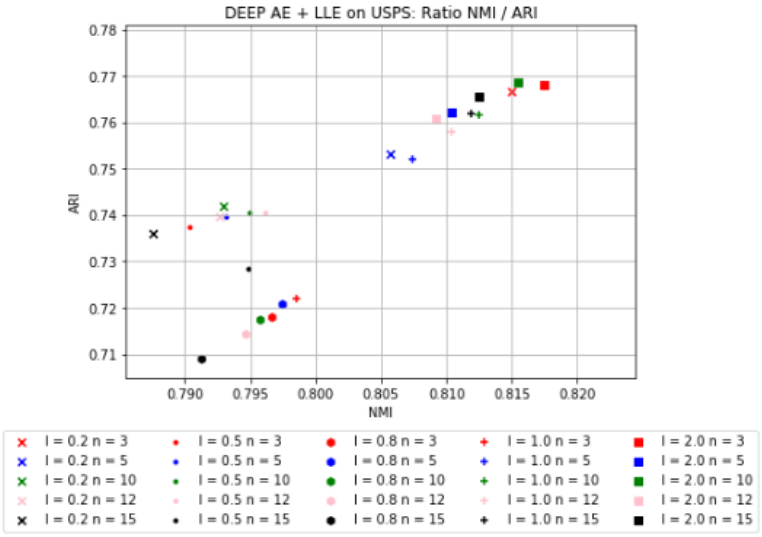

Figure 15: DAE + LEE (encoding matrix representation) on USPS ratio NMI/ARI

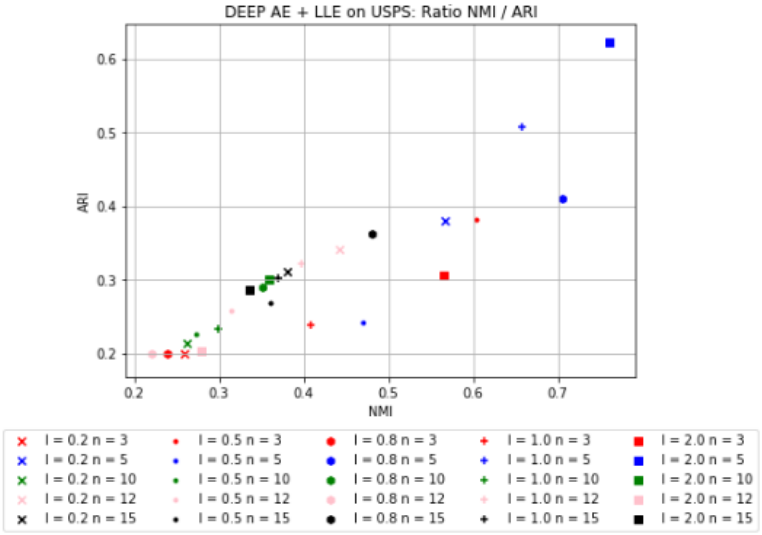

Figure 16: DAE + LEE (Embedding matrix representation) on USPS ratio NMI/ARI

8.3.2 visualization

We get very good representation in the both representation (encoding and embedding), the clusters are very well separated.
Note: we tried UMAP for the embedding matrix after T-SNE failed to gave good representation, we worked with T-SNE for the encoding matrix because T-SNE and UMAP gave almost the same thing (we use UMAP only if T-SNE fail to separate clusters).

8.3.3 Comparing with others methods

We can see here that all the others methods failed to outperform the new algorithm even the LLE failed this time to outperform the clustering on the embedding matrix.
Here the 2 best visualization with T-SNE on the encoding matrix and the best visualization with UMAP on the embedding matrix:

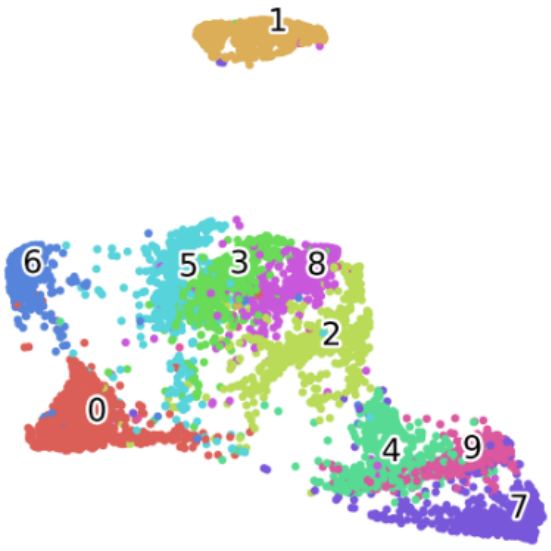

Figure 17: visualization with T-sne of the results obtained after application of a clustering on the encoding matrix with perplexity=10

Figure 18: visualization with T-sne of the results obtained after application of a clustering on the encoding matrix with perplexity=40

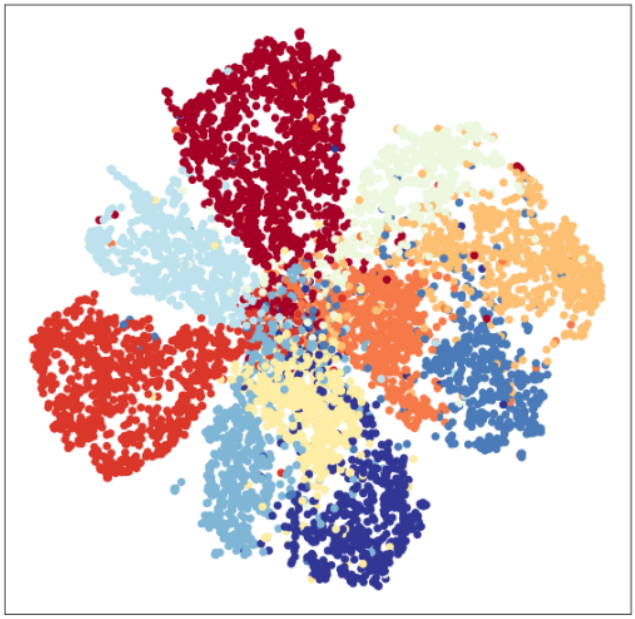

Figure 19: visualization with UMAP of the results obtained after application of a clustering on the embeding matrix with numbe rof neighbors=10

8.3.4 Comparing with others methods

Algorithm	NMI	ARi
DEEP AE + LLE (embedding representation)	0.75	0.62
DEEP AE + LLE (encoding representation)	0.82	0.77
DEEP AE	0.44	0.27
PCA	0.44	0.35
LLE	0.48	0.33

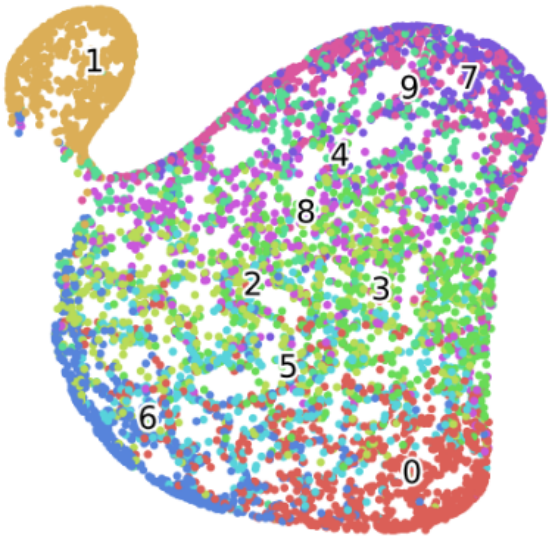

Figure 20: visualization with T-sne of the results obtained after application of a clustering on PCA representation

Figure 21: visualization with T-sne of the results obtained after application of a clustering on DEEP AE representation

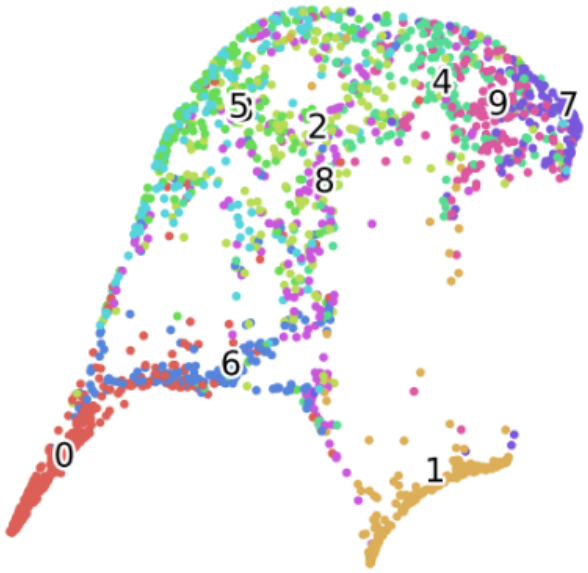

Figure 22: visualization with T-sne of the results obtained after application of a clustering on LLE representation

9 Conclusion

This report introduces Deep AE + LLE , an algorithm combining dimension reduction and deep learning to learn a more accurate representation of data. After having made a brief state of the art of dimension reduction, a benchmarking of Deep AE + LLE on two sets of data has been realized in order to confront it to certain methods of the state of the art. We can conclude from our experimentation that this new algorithm surpasses the other methods with both representations encoding matrix and embedding matrix, except like we could see for LLE algorithm this last one surpasses the proposes algorithm for the MNIST dataset, but fails to surpass it for USPS dataset.

We can also conclude that the encoding matrix representation is far better in terms of NMI, ARI and even representation quality (the embedding matrix didn't give acceptable visualization plots for the MNIST dataset). In addition, the embedding matrix costs a lot of cpu and memory resources and like it has been shown in the tables above the range of ARI and NMI for the embedding matrix varies a lot for different combinations of parameters, making the embedding matrix representation unstable for clustering.

10 Future Work

A large number of hyper parameters, tests and experiences have been abandoned or left in the future for lack of time. Future work mainly concerns on further analysis of the optimization of the Deep AE + LLE algorithm, we have also some suggestions to try different methods or simply perform tests on other data sets out of curiosity. Some of these ideas ca be improved or added to this project :

- Using of multi core T-sne : we only used T-sne with one core
- We can try other clustering algorithm instead of K-means like mixture models
- We can try using convolutional Auto Encoder
- on other datasets

- choose the Auto Encoder architecture