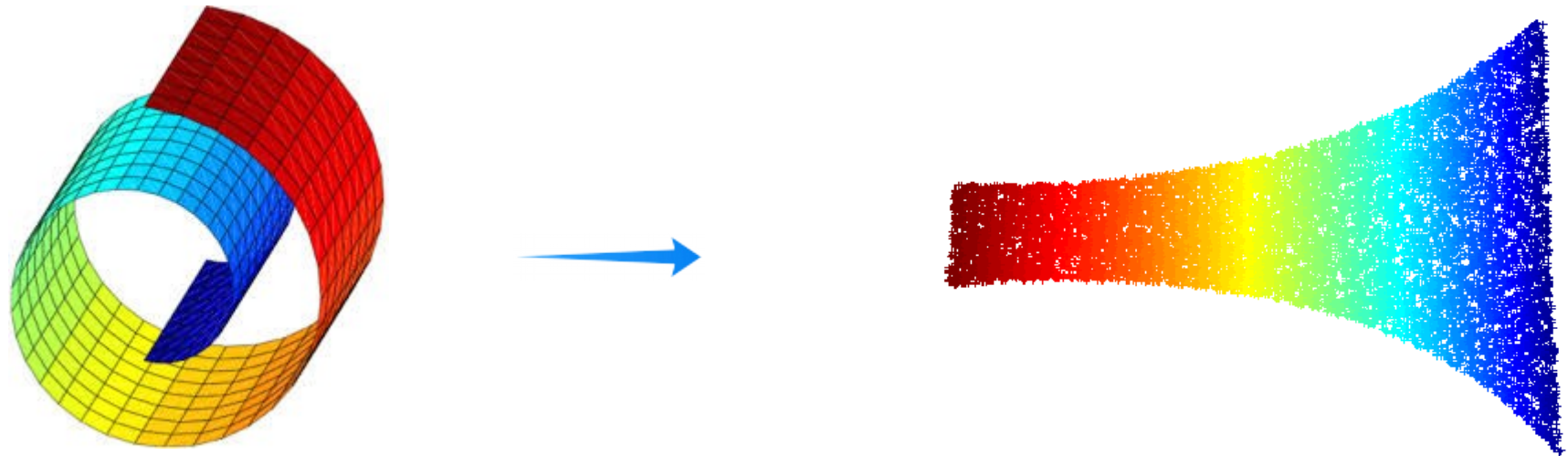


# NonLinear Dimensionality Reduction in Pattern Recognition Applications



- Elaboration: Katsileros Petros
- Supervision: Nikolaos Pitsianis, Nikos Sismanis



# Introduction

- Introduction about Dimensionality Reduction
- Presentation sections
  - Why we need information compression
  - Dimensionality Reduction Algorithms (PCA, SVD, ISOMAP, Laplassian Eigenmaps, LLE)
  - The Locally Linear Embeddings Algorithm (LLE)
  - **Surpass LLE limitations with two new variations of LLE algorithm**
  - Experiments using native LLE and the two new methods
    - Datasets: MNIST, SVHN, Arcene
  - **Results visualization and explanation**
  - Future work
  - Conclusion



# Why we need information compression

- Example: Image with size [480,640]
  - That is equal to a vector with size:  $480 \times 640 = 307200$ .
  - For a “small” dataset,  $N = 100K$  we have a matrix of size:  $100.000 \times 307200$
- Very large Computational complexity
- Very large memory requirements
- A large amount of these pixels are just Noise, affecting negative machine learning or image processing algorithms.
- Why dont we just simulate the Human brain ...
- How? Using keypoints into each image
  - Features like SIFT, HoG, PFH etc ...
  - Find out which pixels have meaningful information
- Dimensionality reduction techniques are able to extract  $d$  (ex.  $8 < d < 256$ , from  $480 \times 640$ ) meaningful key points



# Dimensionality Reduction Algorithms

- Linear:

- **Principal Component Analysis (PCA)**

- Minimize a mean square error equation (sum of eigenvalues)

- $\mathbf{x} = \sum_{i=0}^{N-1} y_i \mathbf{a}_i, \quad y(i) = \mathbf{a}_i^T \mathbf{x}$

- $\hat{\mathbf{x}} = \sum_{i=0}^{N-1} y_i \mathbf{a}_i$

- $E[\|\mathbf{x} - \hat{\mathbf{x}}\|^2] = \sum_{i=m}^{N-1} \mathbf{a}_i^T \lambda_i \mathbf{a}_i = \sum_{i=m}^{N-1} \lambda_i$

- Produces statistical independent features (  $E[\mathbf{x}] = 0, E[\mathbf{y}] = 0$  )

- Multi Dimensional Scaling (MDS)

- **Singular Value Decomposition (SVD)**

- Solving the equation:  $X = U_r \Lambda^{(\frac{1}{2})} V_r^H$



# Dimensionality Reduction Algorithms

- Non-Linear:
  - Isometric Mapping (ISOMAP)
  - Laplassian-Eigenmaps
  - Locally Linear Embeddings



# Locally Linear Embeddings

- Step-1: Find K-Nearest Neighbors for each data (Adjacency matrix)

- Executed in CUDA

- Step-2: Find Weights matrix (W)

- Minimize cost function:  $\arg \min E_w = \sum_{i=1}^N \|X_i - \sum_{j=1}^N W(i, j) X_j\|^2$

- $E = \|\vec{x} - \sum_j w_j \vec{x}_j\|^2 = \|\sum_j w_j (\vec{x} - \vec{x}_j)\|^2 = \sum_{jk} w_j w_k C_{jk}$

- Gram matrix:  $C_{jk} = (\vec{x} - \vec{x}_j) \cdot (\vec{x} - \vec{x}_k)$

- Solve the system:  $\sum_j C_{jk} \cdot w_k = 1$



# Locally Linear Embeddings

- Step-3: Find the embedding coordinates, using the weights matrix  $W$

- Minimize cost function:  $\arg \min E_y = \sum_{i=1}^N \|Y_i - \sum_{j=1}^N W(i, j) Y_j\|^2$

- $E_y = \|(I - W)Y\|^2 = Y^T M Y$

- Find eigen-values of the square  $[N \times N]$  **sparse** matrix

- $M = (I - W)^T (I - W)$

- Step-4: Keep the final embedded coordinates

- Discard the  $\lambda_0$  eigenvalue (equal to zero)

- Keep the rest  $d$  eigen-values. The eigen-vectors are the final  $d$  embedded coordinates



# LLE Limitations

- Eigen-value decomposition step has complexity  $k \cdot O(N^2)$ 
  - For sparse matrix M, solving with Lanczos algorithm
- We must run LLE algorithm with both train and test datasets as input data
  - Dimensionality reduction on train and separate on test produces spaces with different base vectors
  - We cannot find any relation between the low dimensional test and train data
- **Solutions:**
  - Method-1 will solve 2<sup>nd</sup> limitation
    - Execute LLE on Train+Test data
  - Method-2 will solve 1<sup>st</sup> limitation
    - Runtime algorithm complexity





# Method-1: LLE with Test-Projection

---

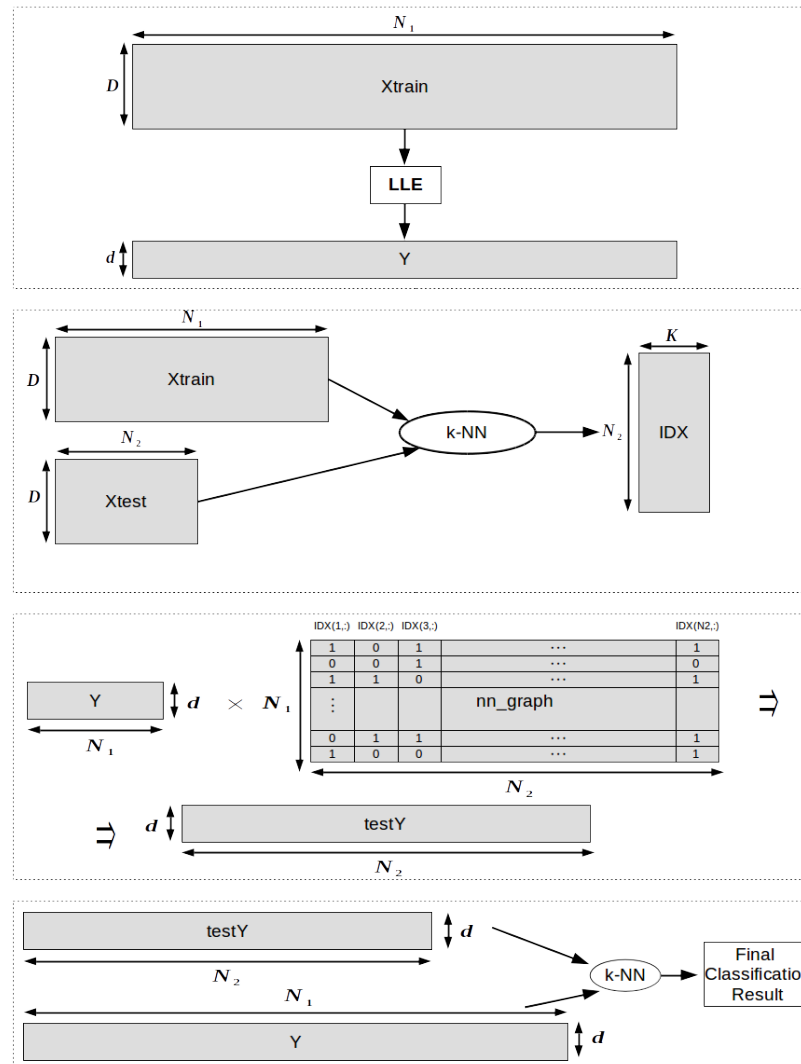
**Algorithm 1** Projection Method

---

- 1: Let  $X_{train}$  be  $[D \times N_1]$  Train\_dataset matrix and  $X_{test}$  be  $[D \times N_2]$  Test\_dataset matrix  
     $\triangleright N_1, N_2$  declare the number of data and  $D$  the number of dimensions
  - 2:
  - 3: Let matrix  $Y$  be  $[d \times N_1]$  Train data, after dimensionality reduction  $\triangleright d < D$
  - 4:
  - 5: Let matrix  $nn\_graph$  with size  $[N_1 \times N_2]$  and all elements equal to zero
  - 6:
  - 7: **for**  $i = 1$  to  $N_2$  **do**
  - 8:     Find K-Nearest Neighbors from  $X_{train}$
  - 9: **end for**
  - 10:
  - 11: Keep the results to matrix  $IDX$  with size  $[N_2 \times K]$   $\triangleright K$  is the number of nearest neighbors
  - 12: **for**  $i = 1$  to  $N_2$  **do**
  - 13:     Set  $IDX(i, 1:K)$  cells of  $nn\_graph$  matrix equal to ones
  - 14:     Make the matrix multiplication  $Y \times nn\_graph(1 : N_1, i)$  and store the result to
  - 15:  $testY(1 : d, i)$   $\triangleright testY(:, i)$  is the result of dimensionality reduced  $X_{test}_i$
  - 16: **end for**
  - 17:
  - 18: Final matrix  $testY$  has size  $[d \times N_2]$  and represents the projection of  $X_{test}$   $D$ -dimensional data into the  $d$ -dimensional embedding subspace.
  - 19:
  - 20: Now execute K-NN Classification between  $testY$  and  $Y$  datasets, to the  $d$ -dimensional space
- 



# Method-1: LLE with Test-Projection

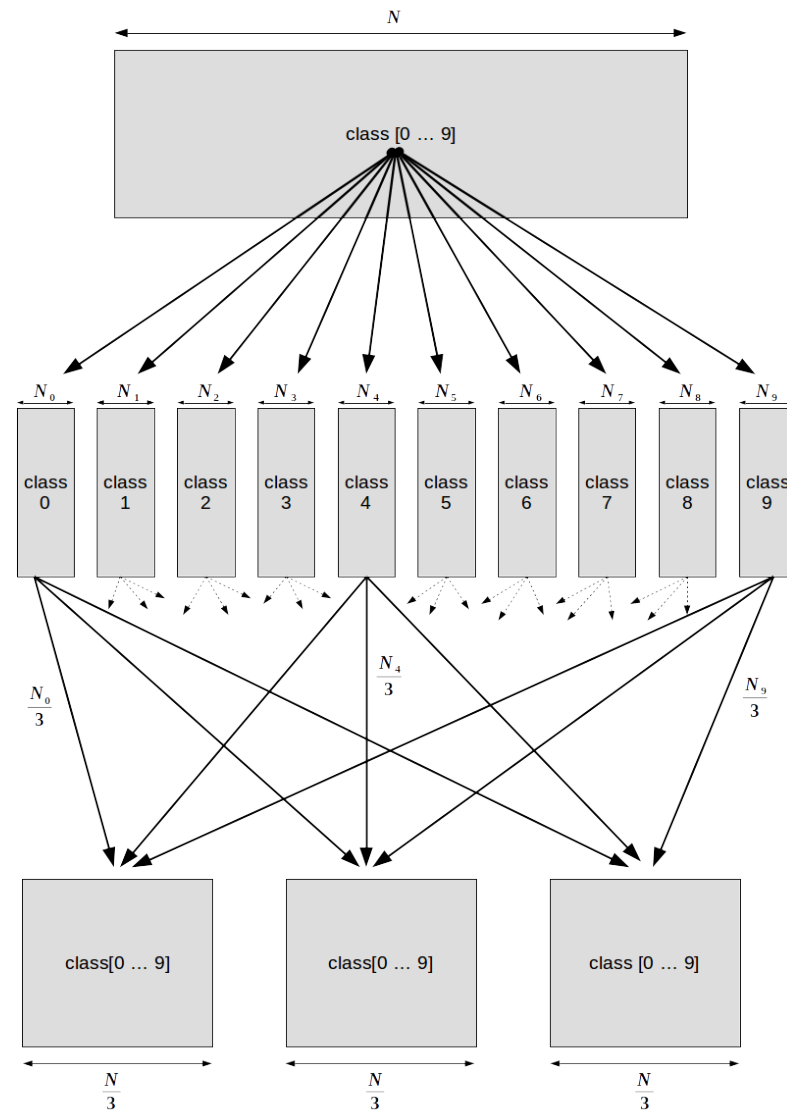


# Method-2: LLE with Subsampling and Majority Voting

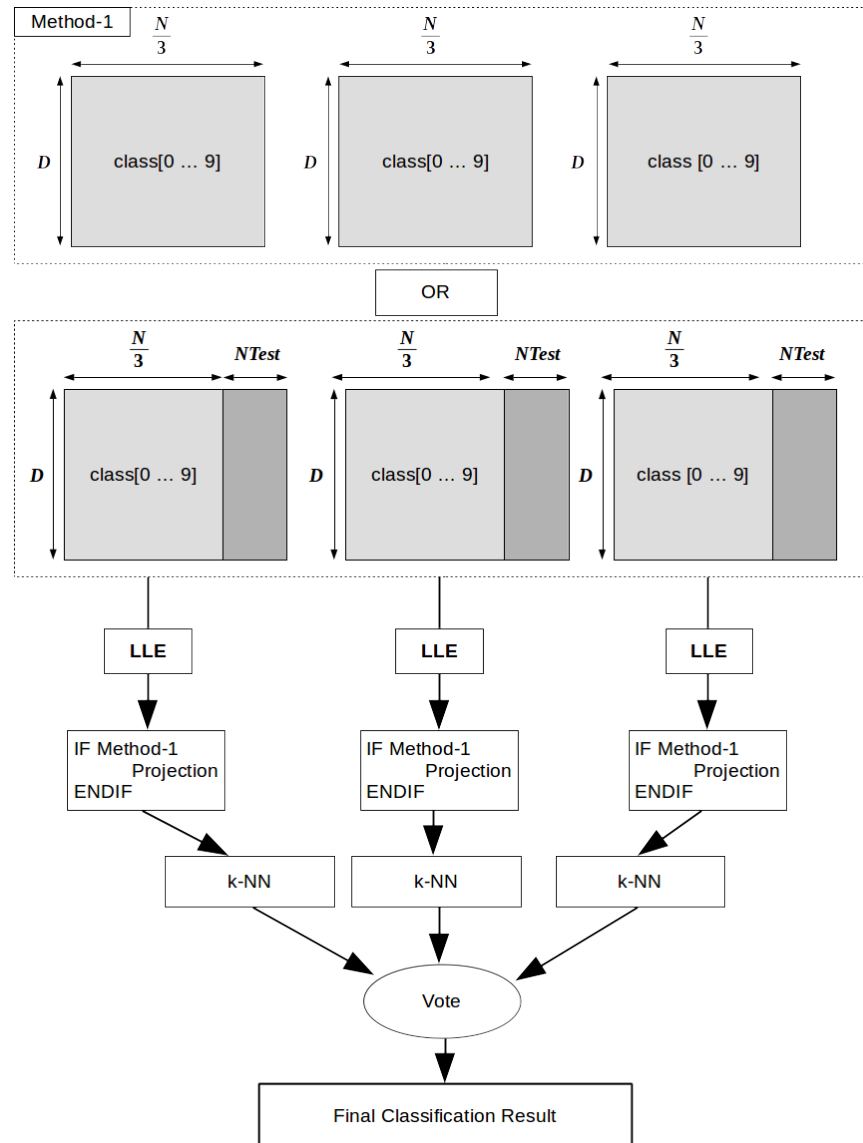
- LLE algorithm needs just a well sampled set of data. Not huge train datasets
- Split Train dataset into sub-sets
  - Sub-sets MUST contain equal information for every class
- Execute dimensionality reduction using LLE on every sub-set
  - Method-1 can be used, in order to avoid including Test data into each sub-dataset
- Execute classification process (k-NN) for every Test data
- The final classification result is the majority voting class from every sub-dataset



# Method-2: LLE with Subsampling and Majority Voting



# Method-2: LLE with Subsampling and Majority Voting

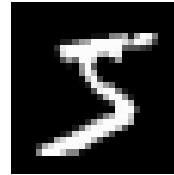


# Experiments

- Datasets: MNIST, SVHN, ARCENE

- MNIST: Gray scale images of handwritten digits.

- 60K Train data, 10K Test data,  $D=[28 \times 28]$



- SVHN: Google street view house numbers  $[32 \times 32]$  RGB images.

- 73257 Train data, 26032 Test data,  $D=[32 \times 32]$  (531131 additional, somewhat less difficult Train data)

- Arcene: Cancer dataset with a large number of predefined features for each patient.

- 200 Train data, 700 Test data,  $D = 10K$ .

- Classification algorithm: k-NearestNeighbors

- Classification metric: Mean average % error

- Accuracy from every class / number of classes



# MNIST Experiments

- 1<sup>st</sup>-Exp: Invest how LLE parameters (k, d) affect the classification process. Also find out if sub-sampling (Method-2) can lead to acceptable results.
- $K = [6, 7, 8, 9, 10, 12, 16, 20, 24, 32, 64]$ ,  $d = [10, 16, 20, 24, 32, 40, 52, 64, 96, 128, 256]$ , subSet\_size=[60000,30000,20000] of Train dataset
  - Classification error using k-NN (k=2): **K=12, d=128, subSet\_size=60K**, qual to **3.06%**
  - Classification error using k-NN (k=2) without dimensionality reduction: **3.5%**
  - Classification error using k-NN (k=2): **K=8, d=10, subSet\_size=60K** equal to **3.31%**
  - From the above results, we can say that LLE algorithm can be used as a feature extraction process with a great data compression ability.



# MNIST Experiments

- Method-2 results:
  - K=16, d=256, batch\_size=20K. Best classification error equal to **3.27%**.
  - K=10, d=128, batch\_size=10K. Best classification error equal to **3.31%**.
  - **Huge reduction both in time and space.** (Step-3 of LLE has  $O(N^2)$  complexity)
- Method-1: Use of LLE dimensionality reduction in “Real time” applications
  - K=8, d=256, batch\_size=60K. Best classification error equal to 3.85%.





# SVHN Experiments

- Extract HoG features due to noise and light distortions.
  - Split image into sub sections and calculate the gradient of pixel intensity.
- **1<sup>st</sup>-Exp: Investigate LLE behaviour to this dataset. Also find out the best HoG kernel size.**
  - $K = [8, 10, 12]$ ,  $d = [16, 20, 32, 64, 96, 128, 164, 196, 256]$ , 30K of SVHN Train data-set
  - HoG Kernel size:  $[2 \times 2]$ ,  $[4 \times 4]$ ,  $[8 \times 8]$  produce features of length: 8100, 1764, 324 .
  - **Best parameters for SVHN dataset:  $K=12$ ,  $d=32$ , kernel= $[4 \times 4]$ .**
- **2<sup>nd</sup>-Exp: Classification result with vs without LLE dimensionality reduction.**
  - $K=12$ ,  **$d=32$** , kernel= $[4 \times 4]$ . Full SVHN dataset (73257 Train data, 26032 Test data)
  - Classification error with LLE dimensionality reduction is equal to: **16.67%**.
  - Classification without dimensionality reduction ( **$D=1024$** ) is equal to: **17.00%**.
  - The above classification results produced from k-NN Classification algorithm with  $k=8$ .



# SVHN Experiments

- 3<sup>rd</sup>-Exp Method-1: Find out if Method-1 can lead to acceptable results.
  - K=12, d=32, kernel=[4x4]. 42K of SVHN Train dataset
  - Classification error with LLE dimensionality reduction on Train data and projection for Test data is equal to **18.34%**
  - Classification error with LLE dimensionality reduction on Train+Test data is equal to 16.67%
  - Classification error without dimensionality reduction is equal to **18.07%**
- 4<sup>th</sup>-Exp Method-2: Investigate how the parameter #number\_of\_spaces affects the classification result
  - K=12, d=32, kernel=[4x4]. 30K of SVHN Train dataset. Method-2 parameters: **3 subspaces**
  - Classification results for each of 3 subspaces: 20.19%, 19.05%, 19.97%
  - Classification after subspace voting: **18.30%**



# SVHN Experiments

- 5<sup>th</sup>-Exp Method-2:
  - $K=12$ ,  $d=32$ ,  $\text{kernel}=[4 \times 4]$ . 30K of SVHN Train dataset. Method-2 parameters: **5 subspaces**
  - Classification results for each of 5 subspaces: 20.28%, 21.11%, 20.93%, 20.92%, 21.11%
  - Classification error after subspace voting: **18.37%**
  - Classification error for 3 subspaces (4<sup>th</sup>-Exp) is equal to **18.30%**
- 6<sup>th</sup>-Exp Method-2:
  - LLE parameters:  $K=12$ ,  $d=32$ ,  $\text{kernel}=[4 \times 4]$ . 30K of SVHN Train dataset. Method-2 parameters: **10 subspaces**
  - Classification error after subspace voting: **18.58%**
- 7<sup>th</sup>-Exp Method-2:
  - LLE parameters:  $K=12$ ,  $d=32$ ,  $\text{kernel}=[4 \times 4]$ . 30K of SVHN Train dataset. Method-2 parameters: **20 subspaces**
  - Classification error after subspace voting: **19.13%**



# ARCENE Experiments

- 1<sup>st</sup>-Exp: Investigate LLE dimensionality reduction as feature selection/extraction algorithm.
  - $K = [10, 12, 16, 20, 24, 32, 64]$ ,  $d = [10, 16, 20, 24, 32, 40, 52, 64, 96, 128]$
  - Lack of test labels. 150 Train patients and 50 Test patients with 10K features for each one.
  - For every  $K$  and  $d$  ( $\leq 96$ ) combinations, classification error after LLE dimensionality reduction is by far better than the  $D=10K$  dimensional space
  - Best classification error after LLE dimensionality reduction is for parameters  $(K=10, d=10)$ ,  $(K=10, d=52)$ ,  $(K=12, d=128)$ , equal to **10%**
  - Classification error without dimensionality reduction ( $D=10K$ ) is equal to **24%**.
  - **Selecting 10 of 10K features, we gain a boost of 14% classification accuracy.**
  - **We can produce accurate classification results with 90% successful probability .**



# Future Work

- Parallelization: Even though k-NN algorithm both in LLE and in Classification process is executed in CUDA, further parallelization can be achieved from sub-set splitting into threads.
- Extend to large datasets and invest the behaviour of subspace majority voting
- Find the cutting edge for the size of Train dataset subsets and Test data.
- Maybe a very accurate dimensionality reduction algorithm for Medical image retrieval.



# Conclusion

- LLE produces very accurate results after huge dimensionality reduction.
- LLE algorithm can be used both as feature extraction and feature selection algorithm.
- LLE has the ability to remove noise-data, producing very accurate classification results.
- Huge space and time savings at Final Classification Step ( $d \ll D$ ).
- Method-1 can lead to real time Classification algorithms. Using LLE Classification can be executed for low dimensional Train and Test spaces.
- Method-2 achieves dramatically reduction both in space and time, with minimal information loss.
- Method-1 and Method-2 make the execution of LLE algorithm available for normal PCs.



Thank You

