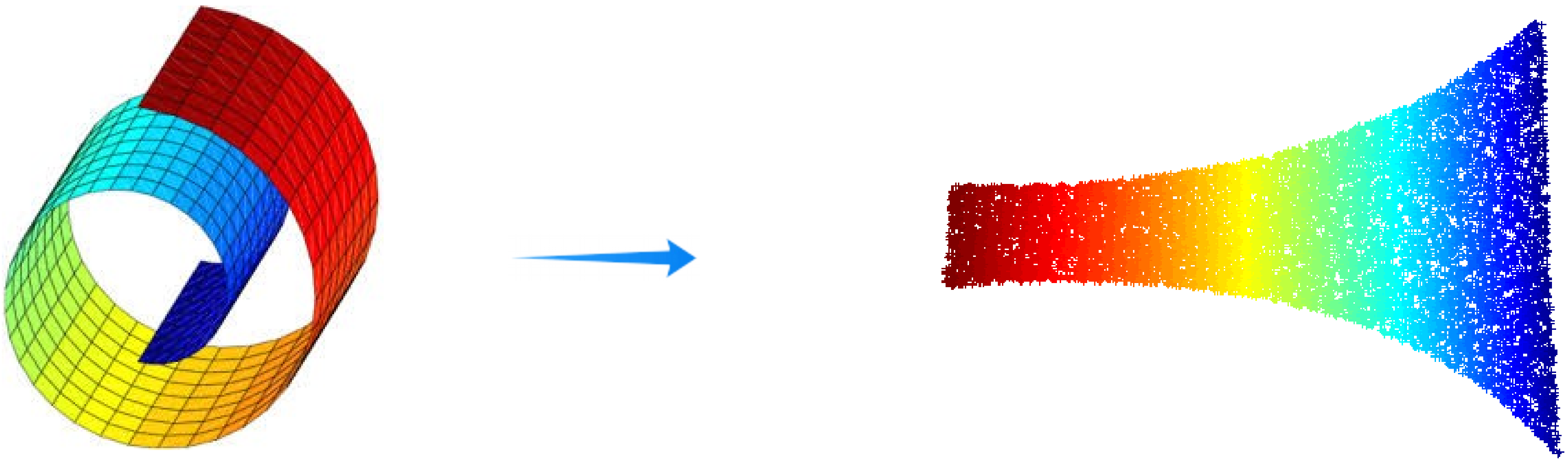
Locally Linear Embeddings for Pattern Recognition Applications



- Elaboration: Katsileros Petros
- Supervision: Nikolaos Pitsianis

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 analysis**
  - Future work
  - Conclusion

Why we need dimensionality reduction

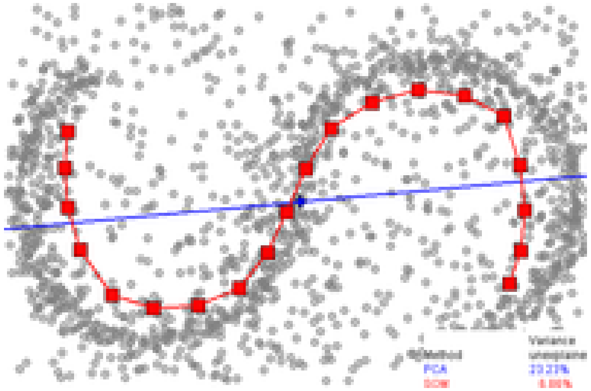
- Example: Image with size [480,640]
  - That is equal to a vector with size: 480x640 = 307200.
  - For a “small” dataset, N = 100K we have a matrix of size: 100.000 x 307200
- Very large Computational complexity
- Very large memory requirements
- A large amount of these pixels are just noise, affecting 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 480x640) meaningful key points

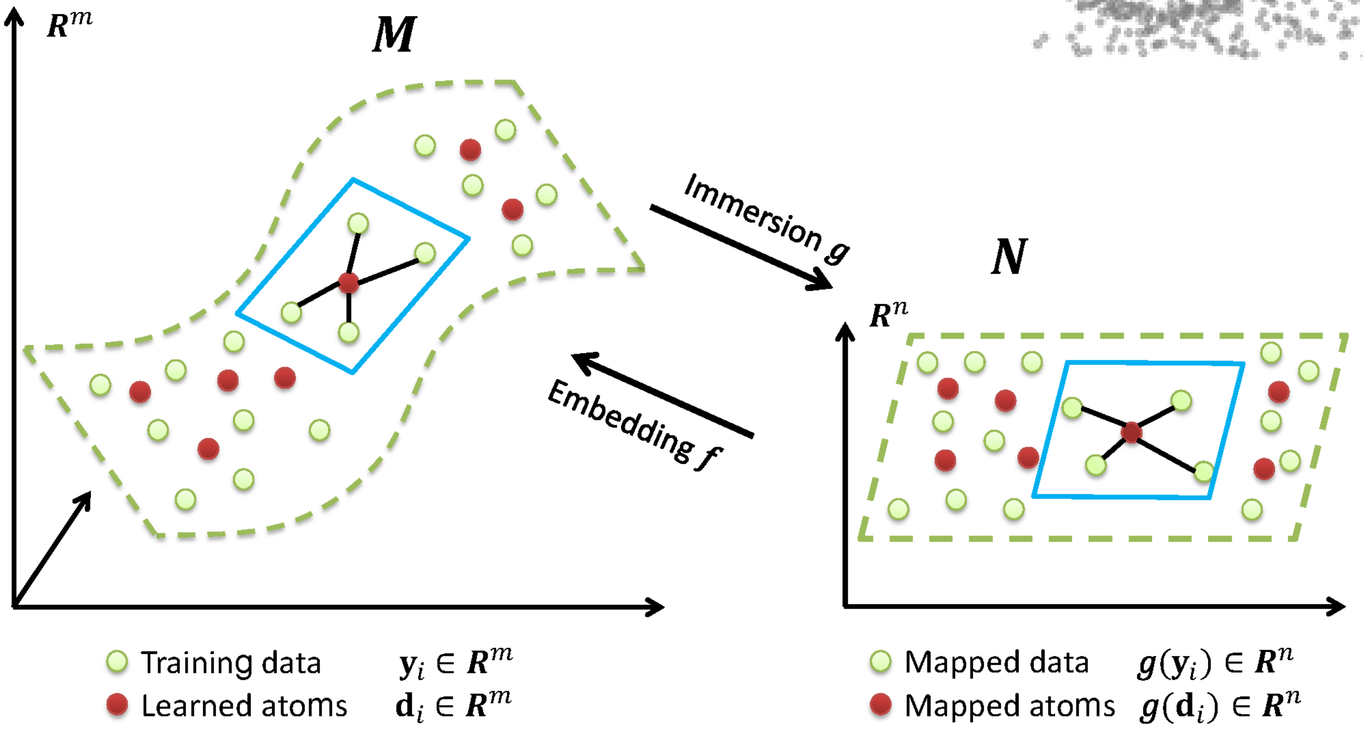
Dimensionality Reduction Algorithms

- Linear:
  - **Principal Component Analysis (PCA)**
    - Minimize a mean square error equation (sum of eigenvalues)
      - $\boldsymbol{x} = \sum_{i=0}^{N-1} y_i \boldsymbol{a_i}$, $y(i) = \boldsymbol{a_i^T} x$
      - $\hat{\boldsymbol{x}} = \sum_{i=0}^{m-1} y_i \boldsymbol{a_i}$
      - $E[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] = \sum_{i=m}^{N-1} \boldsymbol{a_i^T} \lambda_i \boldsymbol{a_i} = \sum_{i=m}^{N-1} \lambda_i$
    - Produces statistical independent features ($E[\boldsymbol{x}]=0, E[\boldsymbol{y}]=0$)
  - Multi Dimensional Scaling (MDS)
  - **Singular Value Decomposition (SVD)**
    - Solving the equation: $X = U_r \Lambda^{\left(\frac{1}{2}\right)} V_r^H$

Dimensionality Reduction Algorithms

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




$\boldsymbol{R}^m$
$\boldsymbol{M}$
Immersion g
Embedding f
$\boldsymbol{N}$
$\boldsymbol{R}^n$
○ Training data $\mathbf{y}_i \in \boldsymbol{R}^m$
● Learned atoms $\mathbf{d}_i \in \boldsymbol{R}^m$
○ Mapped data $g(\mathbf{y}_i) \in \boldsymbol{R}^n$
● Mapped atoms $g(\mathbf{d}_i) \in \boldsymbol{R}^n$

Locally Linear Embeddings

- Step-1: Find K-Nearest Neighbors for each data (Adjacency matrix)
  - Executed in CUDA
- Step-2: Linear prediction for every point using it’s neighbors. Find Weight matrix (W).
  - Minimize cost function: $arg\,min\,E_w = \sum_{i=1}^{N} \| X_i - \sum_{j=1}^{N} W(i,j) X_j \|^2$
  - Weights W must follow these two restrictions:
    - W(i,j) = 0, if j and i are not neighbors
    - $\sum_{i=1}^{N} W(i,j) = 1$
    - Following these restrictions we have Translation, Rotation and Scaling independence.

Locally Linear Embeddings

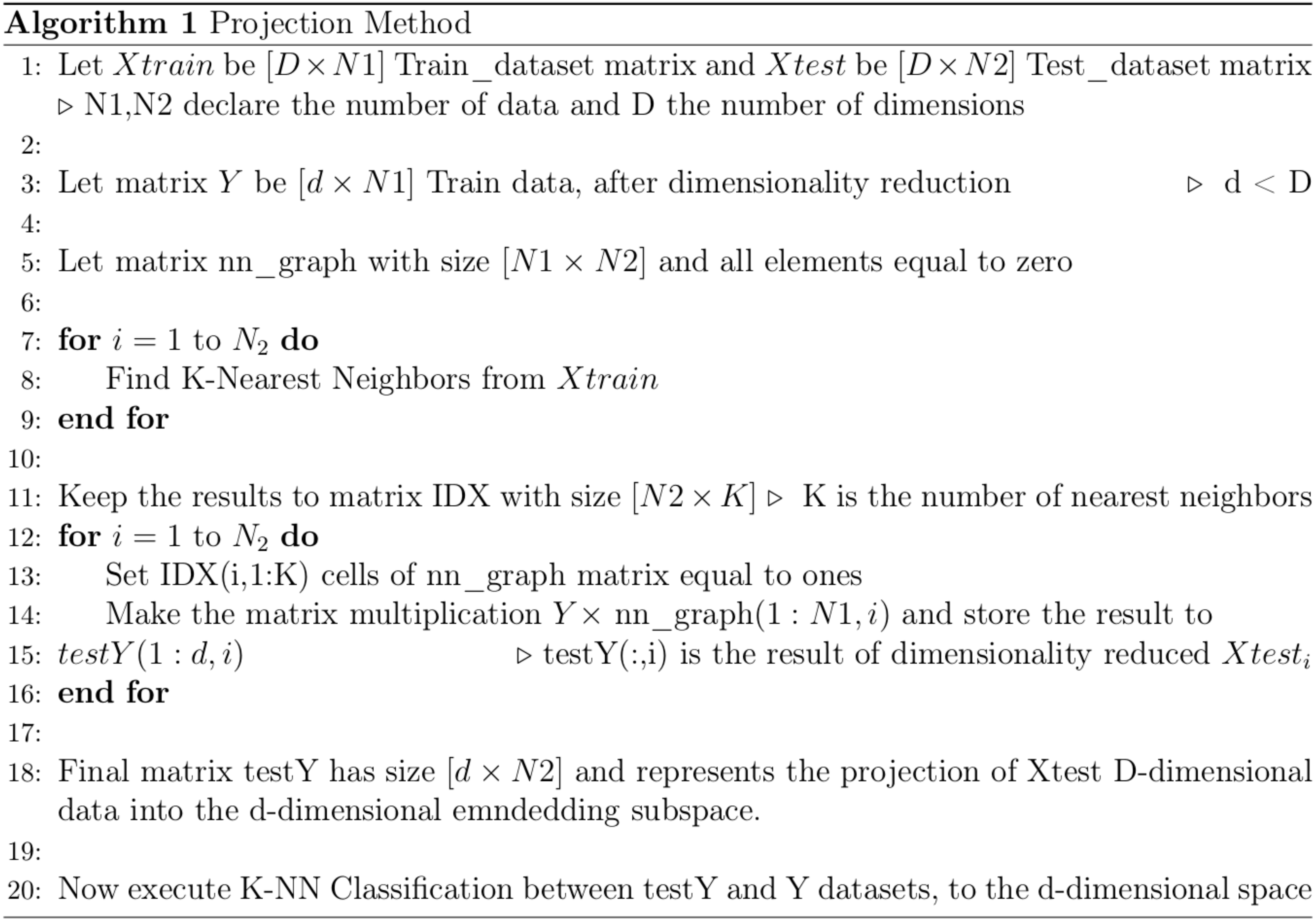
- Step-3: Find the embedding coordinates, using the weights matrix W
  - Minimize cost function: $arg\,min\, E_y = \sum_{i=1}^{N} \| Y_i - \sum_{j=1}^{N} W(i,j) Y_j \|^2$
    - $E_y = |(I-W)Y|^2 = Y_T M Y$
  - Find eigen-values of the square [NxN] **sparse** matrix
    - $M = (I-W)^T (I-W)$
- Step-4: Keep the final embedded coordinates
  - Discard the λ_0 eigenvalue (equal to zero)
  - Keep the rest d eigen-values. Their 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 realation between the low dimensional test and train data

- **Solutions**:
  - Method-1 will solve 2nd limitation
    - Execute LLE on Train+Test data
  - Method-2 will solve 1st limitation
    - Runtime algorithm complexity

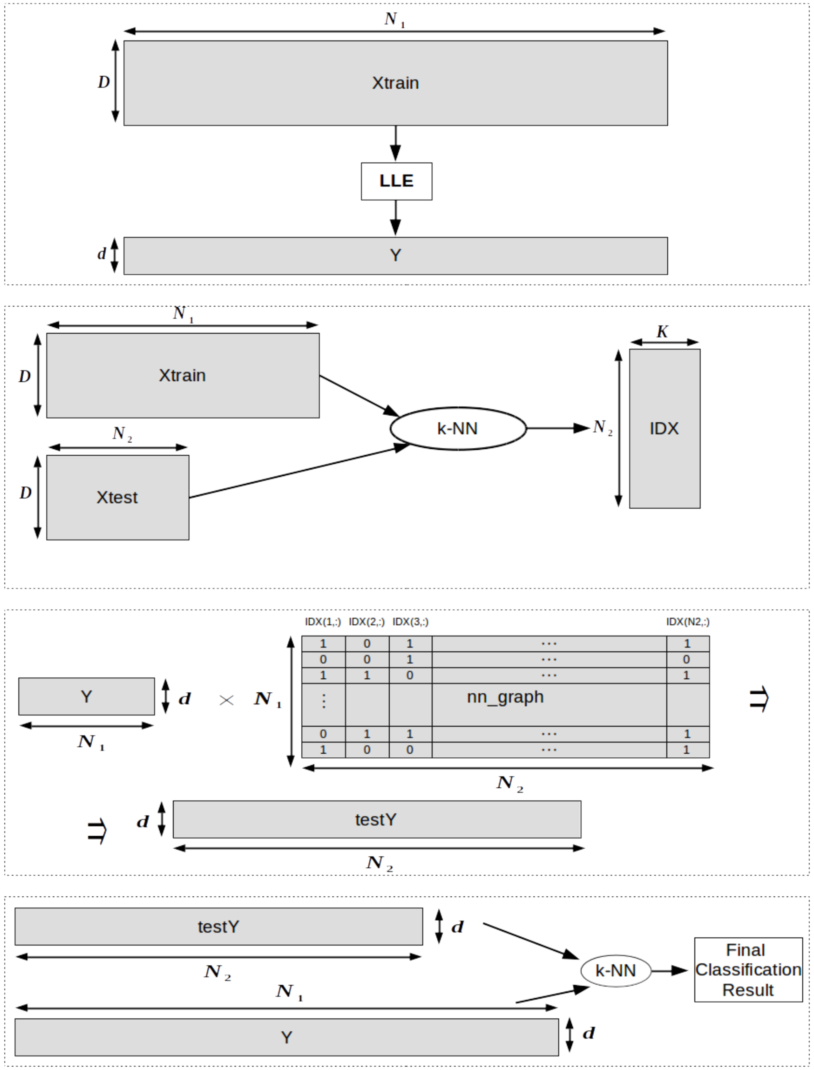
Method-1: LLE with Test-Projection



Algorithm 1 Projection Method

```
Let $Xtrain$ be $[D \times N1]$ Train_dataset matrix and $Xtest$ be $[D \times N2]$ Test_dataset matrix
    ▷ N1,N2 declare the number of data and D the number of dimensions

Let matrix $Y$ be $[d \times N1]$ Train data, after dimensionality reduction    ▷ d < D

Let matrix nn_graph with size $[N1 \times N2]$ and all elements equal to zero

for $i = 1$ to $N_2$ do
    Find K-Nearest Neighbors from $Xtrain$
end for

Keep the results to matrix IDX with size $[N2 \times K]$ ▷ K is the number of nearest neighbors
for $i = 1$ to $N_2$ do
    Set IDX(i,1:K) cells of nn_graph matrix equal to ones
    Make the matrix multiplication $Y \times$ nn_graph$(1 : N1, i)$ and store the result to
$testY(1 : d, i)$    ▷ testY(:,i) is the result of dimensionality reduced $Xtest_i$
end for

Final matrix testY has size $[d \times N2]$ and represents the projection of Xtest D-dimensional
    data into the d-dimensional emndedding subspace.

Now execute K-NN Classification between testY and Y datasets, to the d-dimensional space
```

Method-1: LLE with Test-Projection

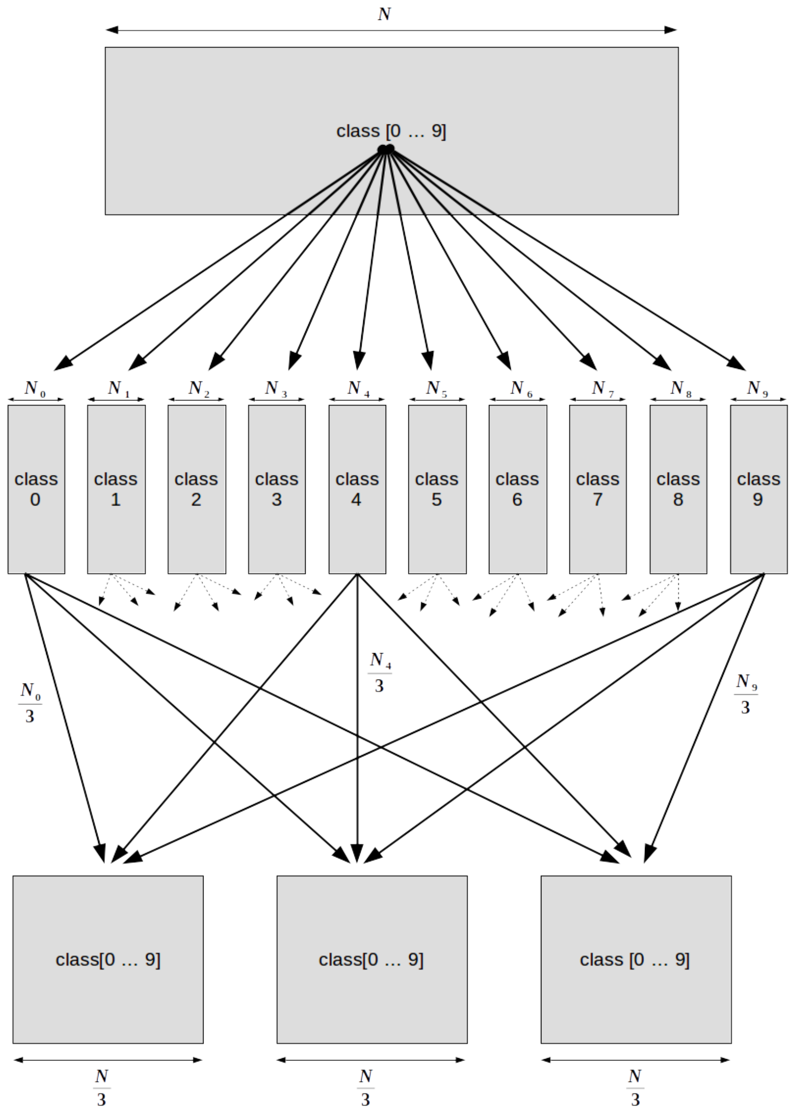


IDX(1,:)	IDX(2,:)	IDX(3,:)		IDX(N2,:)
1	0	1	⋯	1
0	0	1	⋯	0
1	1	0	⋯	1
⋮			nn_graph	
0	1	1	⋯	1
1	0	0	⋯	1

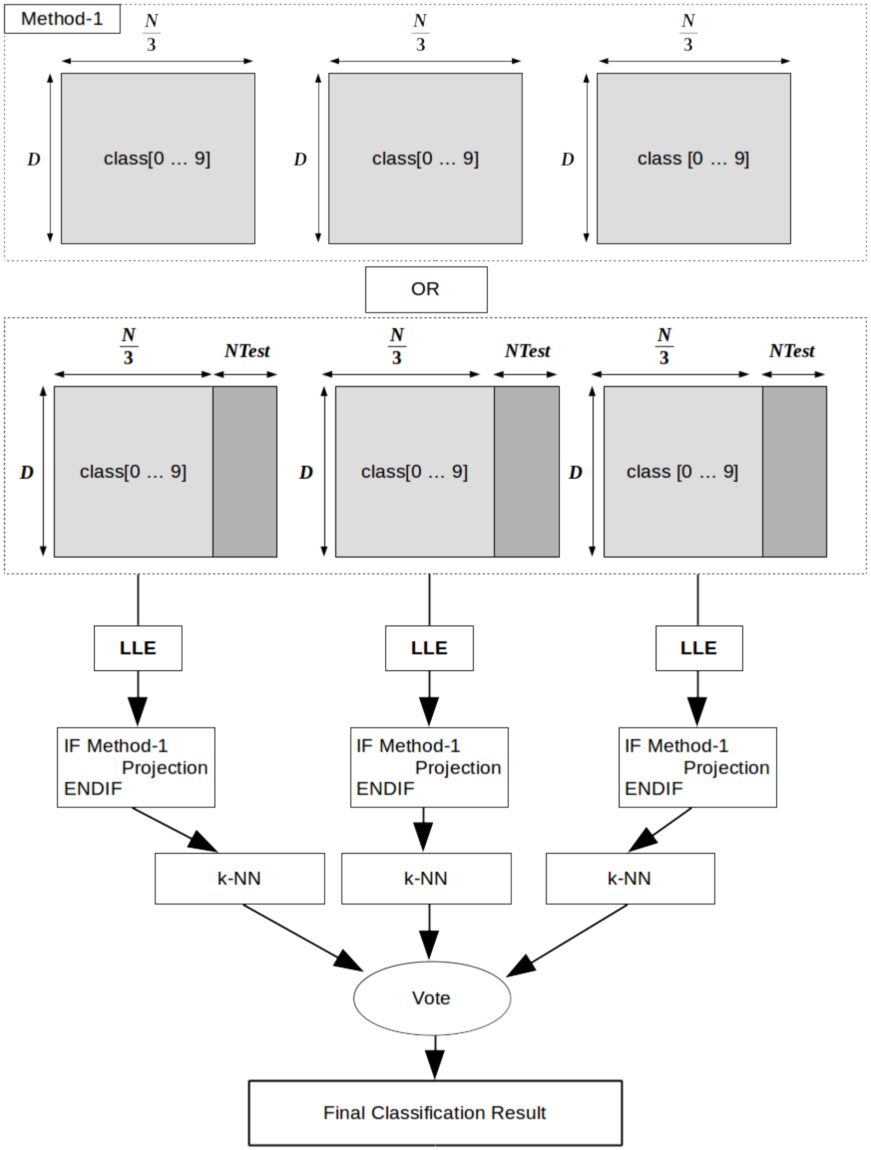
Method-2: LLE with Subsampling and Majority Voting

- 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

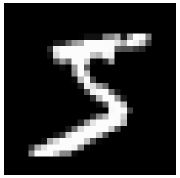


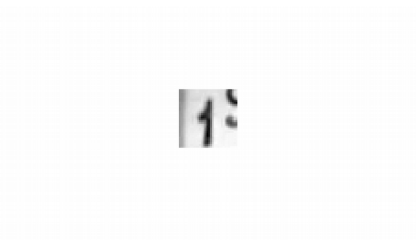
Method-2: LLE with Subsampling and Majority Voting

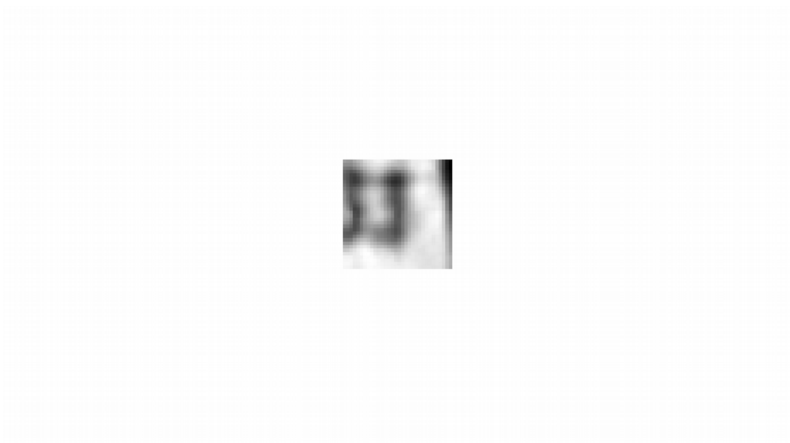


Experiments

- Datasets: MNIST, SVHN, ARCENE
  - MNIST: Gray scale images of handwritten digits.
    - 60K Train data, 10K Test data, D=[28x28]
  - SVHN: Google street view house numbers [32x32] RGB images.
    - 73257 Train data, 26032 Test data, D=[32x32] (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
  - Acuuracy from every class / number of classes







MNIST Experiments

- 1st-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) without dimensionality reduction (D=784): **3.5%**
    - ➢ 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): **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.

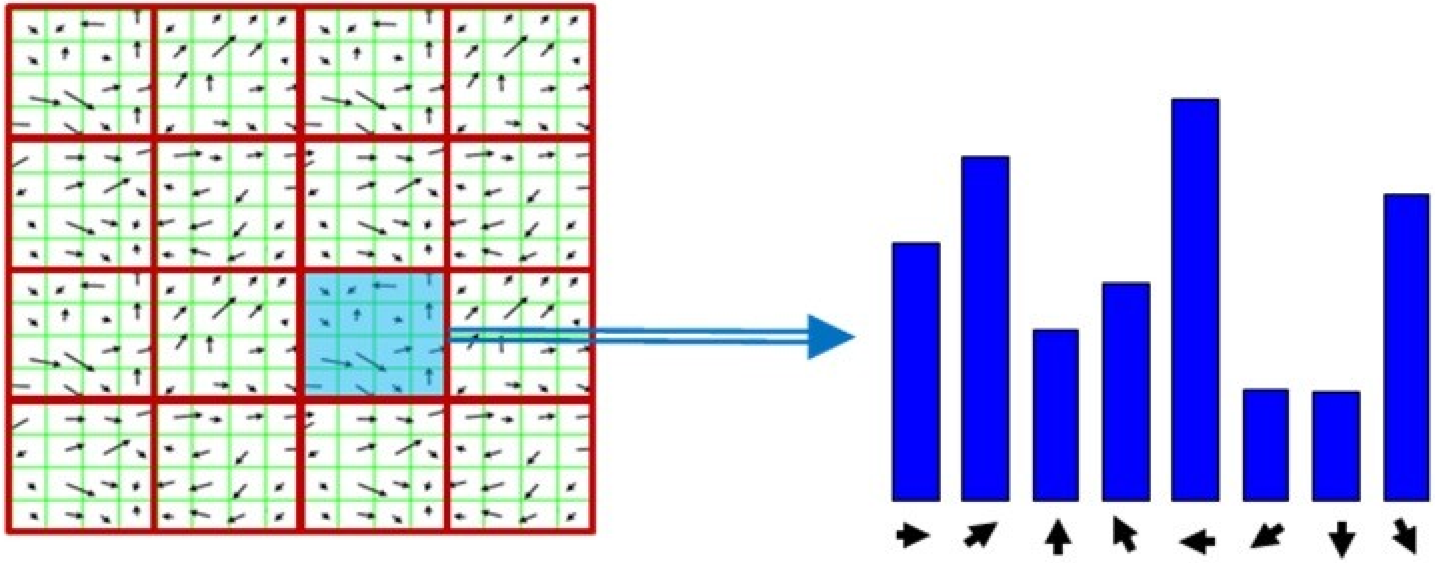
MNIST Experiments

- 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.



SVHN Experiments

- **1st-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: [2x2], [4x4], [8x8] produce features of length: 8100, 1764, 324 .
  - **Best parameters for SVHN dataset: K=12, d=32, kernel=[4x4].**
- **2nd-Exp: Classification result with vs without LLE dimensionality reduction.**
  - K=12, **d=32**, kernel=[4x4]. 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

- 3rd-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%**
- 4th-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

- 5th-Exp Method-2:
  - K=12, d=32, kernel=[4x4]. 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 (4th-Exp) is equal to **18.30%**
- 6th-Exp Method-2:
  - LLE parameters: K=12, d=32, kernel=[4x4]. 30K of SVHN Train dataset. Method-2 parameters: **10 subspaces**
  - Classification error after subspace voting: **18.58%**
- 7th-Exp Method-2:
  - LLE parameters: K=12, d=32, kernel=[4x4]. 30K of SVHN Train dataset. Method-2 parameters: **20 subspaces**
  - Classification error after subspace voting: **19.13%**

ARCENE Experiments

- 1st-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 (<=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 retreival.

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 << 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.

Big Thanks to Nikos Sismanis

Thank You