Nonlinear Dimensionality Reduction Techniques Applied to the Problem of Classifying Images using Support Vector Machines

Amsc663/664 Final Presentation

Chae Clark
cclark18@math.umd.edu

Advisor: Kasso Okoudjou
kasso@math.umd.edu
Department of Mathematics

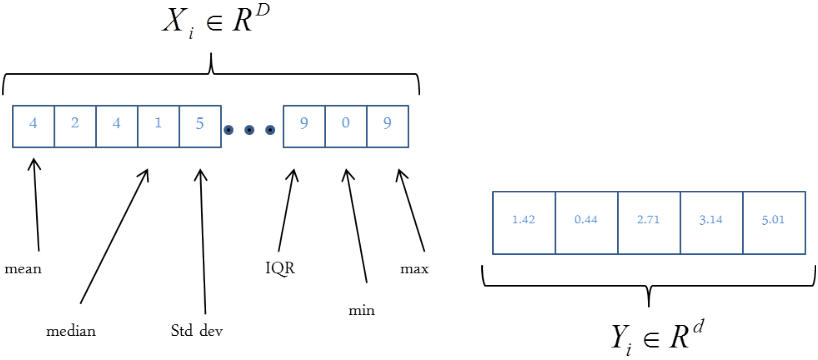
Project Goal

- Processing and analyzing with high dimensional data is difficult. Dimension reduction techniques allow us to reduce the size of data while keeping its key structure.
- This project focuses specifically on the preprocessing of datasets using Locally Linear Embedding, and testing a Support Vector Machine's ability to accurately classify this new data.

Outline

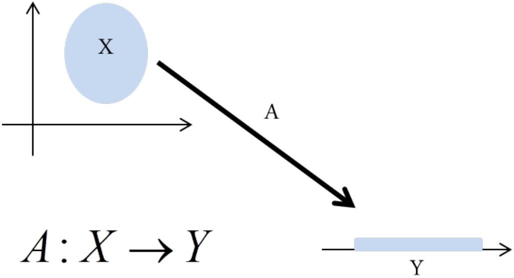
High Dimensional Data

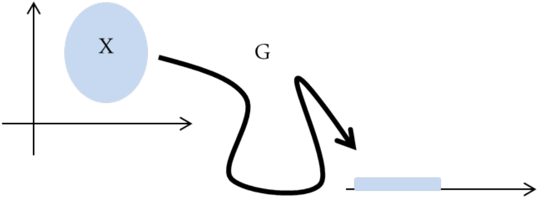
- In data processing schemes, large quantities of data can be collected, but it may not be true that this information is meaningful.
- In many cases, there's redundant/unnecessary information in a dataset.
- We want a smart way to reduce the redundant information, while keeping the structure of our dataset.


$X_i \in R^D$
4
2
4
1
5
9
0
9
mean
median
Std dev
IQR
min
max
1.42
0.44
2.71
3.14
5.01
$Y_i \in R^d$

Dimensionality Reduction Techniques

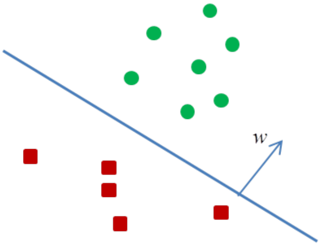
- With Linear Dimension Reduction [LDR] techniques, we are searching for a matrix A_{LDR}, such that when we apply it to our data, we get a *faithful* lower-dimensional representation.
- *Examples* $\{$*Principal Component Analysis, Discriminant Analysis, Independent Component Analysis, etc.*$\}$
- For the cases when our data does not lie on a linear manifold, we must resort to more powerful techniques to reduce our dimension. These are referred to as Nonlinear Dimension Reduction [NLDR] techniques.
- *Examples* $\{$*Locally linear Embedding, Laplacian Eigenmaps, Schrodinger Eigenmaps, etc.*$\}$


X
A
$A : X \to Y$
Y


X
G

Support Vector Machines [SVM]

- Given some dataset $X \in \mathbb{R}^{D \times N}$ with different classes of datapoints $\vec{x}_i$ (images are an example), it's reasonable to assume that datapoints of the same class are close together.
- For convenience, let's assume that our dataset X contains datapoints of two classes {-1,1}. SVM's attempt to find a hyperplane that separates (geometrically) the dataset by class.
- A hyperplane can be represented as a vector normal to the plane $\vec{w}$ and an offset from the origin w_0. So once a datapoint $\vec{x}^*$ is given, all that is required to determine its class, is to see which side of the hyperplane it resides (which is accomplished by determining the sign of $\langle \vec{w}, \vec{x}^* \rangle + w_0$).



$$\arg\min : \frac{1}{2}\|w\|^2$$

Constraints

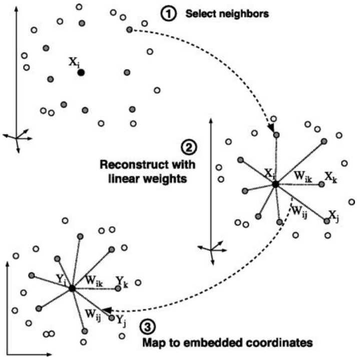
$$y_i(w^T x_i + w_0) \geq 1$$

Outline

Locally Linear Embedding

- The specific DR technique explored in this project is the Locally Linear Embedding [LLE] algorithm developed by Lawrence Saul and Sam Roweis.
- LLE is designed to map datapoints in high dimensional space to a lower dimensional space while preserving local neighborhoods.
- This is accomplished by viewing the data as having a linear local structure.

Locally Linear Embedding (cont.)



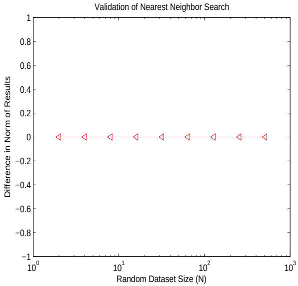
The algorithm has three steps:

1. Determine the neighborhood of each point in our dataset $X \in \mathbb{R}^{D\times N}$. These are the neighborhoods we wish to preserve.
2. For each datapoint $\vec{x}_i \in X$, determine the *weight* each datapoint in the neighborhood affects $\vec{x}_i$.
3. Given the weights, find the lower-dimensional dataset $Y \in \mathbb{R}^{d\times N}$ that preserves them.

Nearest Neighbor Search

- The first step in the process, is to determine the neighborhoods of each of the datapoints $\vec{x}_i \in X$.
- To accomplish this, we employ a K nearest neighbor search.
- For each datapoint $\vec{x}_i$, we calculate its distance to every other point in X. The K datapoints with the smallest distance to $\vec{x}_i$ are used to define the local neighborhood.

Validation



1. To validate our implementation we look at the difference (in norm) of found nearest neighbors and actual nearest neighbors.
2. Finding the K nearest neighbors of a datapoint requires $O[N^2]$ distance computations.

Reconstruction Weights

- The second step in the algorithms, is to determine weights associated with each point in our neighborhood.
- To determine these weights, we must solve the following optimization problem:

$$W = \arg\min \left\{ \sum_{i=1}^{N} \left\| \vec{x}_i - \sum_{j=1}^{K} W_{ij} \vec{\eta}_{ij} \right\|_2^2 \right\}$$

where $\vec{\eta}_{ij}$ is the jth neighbor of datapoint $\vec{x}_i$.

- This problem also has the constraints

$$\sum_{j=1}^{K} W_{ij} = 1 \text{ for all } i = 1, 2, \ldots, N$$

Reconstruction Weights (cont.)

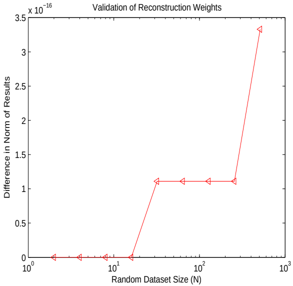
- Following from the constraints is the fact that

$W_{ij} = 0$ for all datapoints that are not neighbors.

- This problem is convex, and as such, if we find a local minimum, we have found a global minimum. Furthermore, this problem has a closed-form solution.

$$\vec{W}_i = \frac{G_i^{-1}\vec{1}}{\sum G_i^{-1}\vec{1}} \text{ for all } i = 1, 2, \ldots, N$$

Validation



1. To validate our implementation we look at the difference (in norm) of weights found and the true optimal weights.
2. The solution to this problem can become ill-conditioned for $K > D$. To handle these cases, a small regularization term is added to the objective function.

Embedding Construction

- The final step in the LLE algorithm requires us to find the lower-dimensional representation $Y \in \mathbb{R}^{d\times N}$ such that our weights between neighbors are preserved. This can be written as an optimization problem.

$$Y = \arg\min \left\{ \sum_{i=1}^{N} \left\| \vec{y}_i - \sum_{j=1}^{K} W_{ij}\vec{\rho}_{ij} \right\|_2^2 \right\}$$

where $\vec{\rho}_{ij}$ is the jth neighbor of the dimension-reduced datapoint $\vec{y}_i$.

- This problem has the constraints

$$\sum_{j=1}^{N} \vec{y}_i = 0 \text{ (center the points around the origin)}$$

and

$$Y \cdot Y^T = I \text{ (essentially fill the same role as } \sum W_{ij} = 1)$$

Embedding Construction (cont.)

- It has been shown that solving this problem is equivalent to performing an eigen-decomposition of the matrix

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

- To be exact, we find $\vec{\lambda}$ and V where these are the eigenvalues and eigenvectors of M respectively. From here, the smallest d non-zero eigenvalues $\vec{\lambda}_d$ and their eigenvectors V_d are chosen.
- The matrix $V_d \in \mathbb{R}^{N \times d}$ is set to be the lower dimesnional embedding of our dataset.

Computing Eigen-values/vectors

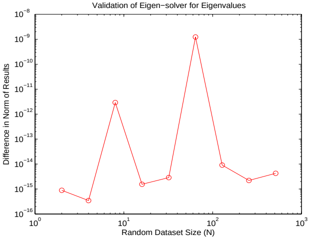
- Looking at our matrix M, we notice some interesting properties.
  1. M is real-valued
  2. M is symmetric
  3. M is positive semi-definite
- Given this set of properties, there are a number of methods to find our needed eigen-values/vectors.

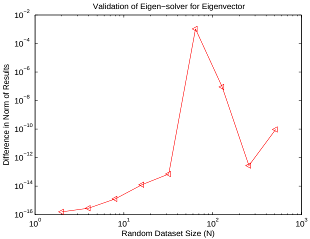
Computing Eigen-values/vectors (cont.)

- To perform the eigen-decomposition, we employ an iterative QR method, detailed as follows

1. Start with a real, symmetric, positive semi-definite matrix A
2. Set $[D^{(0)}, V^{(0)}]$ to be A and I respectively
3. For $k = 1, 2, \ldots, m$
4. factor matrix $D^{(k-1)}$ into Q and R, where Q is unitary and R is lower triangular, and $D^{(k-1)} = QR$
5. $V^{(k)} = V^{(k-1)} \cdot Q$
6. $D^{(k)} = R \cdot Q$
7. endFor
8. Matrix $D^{(m)}$ is diagonal and contains the eigenvalues of A
9. Matrix $V^{(m)}$'s columns are the eigenvectors of A

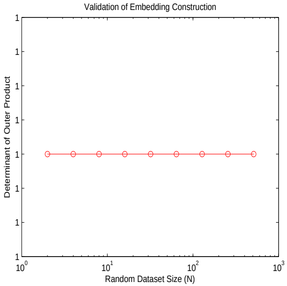
Eigen-solver Validation





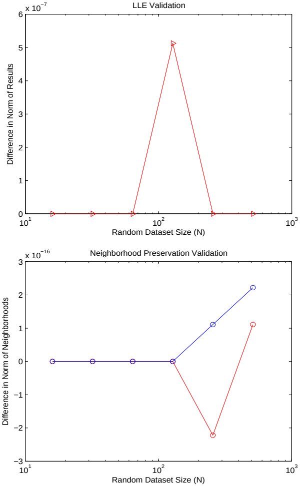
- The top figure shows the difference (in norm) of the found eigenvalues and the actual eigenvalues.
- The bottom figure shows the difference (in norm) of the found eigenvectors and the actual eigenvectors.

Embedding Validation



1. To validate our implementation we look at the difference in norm of thr embedding found and the true optimal embedding.

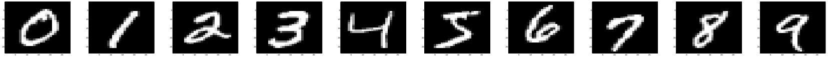
Full LLE Validation



- The top figure shows the difference (in norm) of the found lower-dimensional embedding and the true optimal lower-dimensional embedding.
- The bottom figure shows a comparison of the *quality* of the preserved neighborhood.

Outline

Test Configuration



- A dataset of handwritten images (of the digits 1-9) was used in all testing. The full dataset contains 60,000 (28 x 28) training images, and 10,000 (28 x 28) testing images.
- To process the images, we stack the columns into a single 784-length vector. The dataset is provided by the National Institute of Standards and Technology [NIST] and is available here:

http://yann.lecun.com/exdb/mnist/

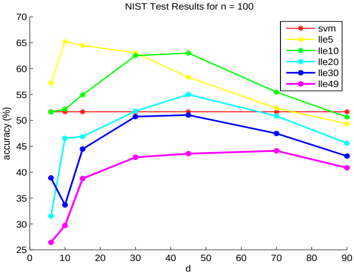
Test Configuration (cont.)

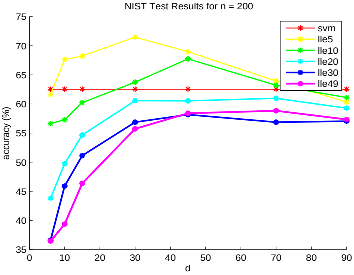
- We use an open source library [LibSVM] for classification available at:

  http://www.csie.ntu.edu.tw/˜cjlin/libsvm/

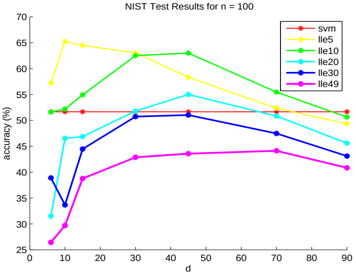
- For the tests, we combine the training and testing sets into a single dataset $X = [\text{train}, \text{test}] \in \mathbb{R}^{D\times(N+M)}$. LLE is applied to X, then an SVM is trained with the embedded training set. We then use the embedded test set to view performance.

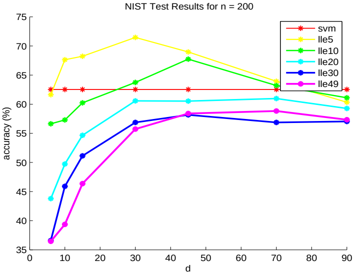
Results for $N = \{100, 200\}$, $M = 3000$



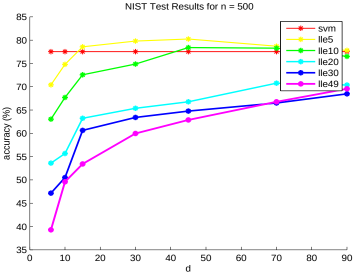


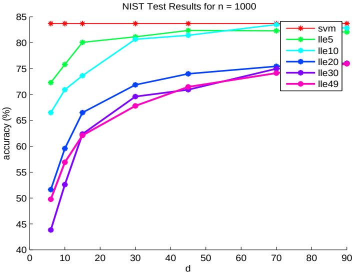
Results for $N = \{300, 400\}$, $M = 3000$



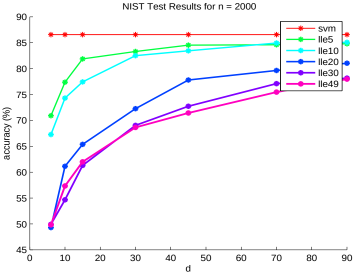


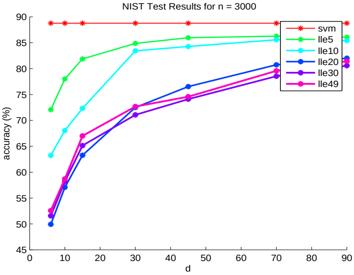
Results for $N = \{500, 1000\}$, $M = 3000$



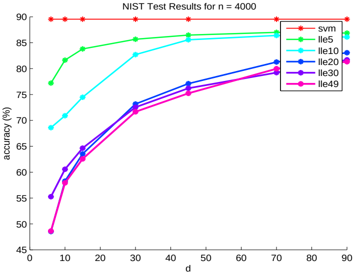


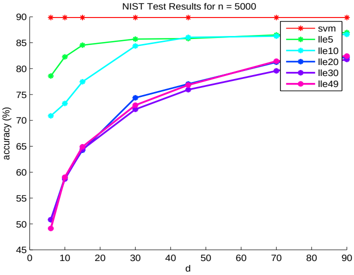
Results for $N = \{2000, 3000\}$ $M = 3000$





Results for $N = \{4000, 5000\}$, $M = 3000$





Conclusion

- The accuracy of SMV classification, in some cases, can be increased by using dimensionality reduction techniques as preprocessors (especially for smaller datasets).
- Even though (at higher dataset sizes) we lose some accuracy in classification, for *intelligently* chosen target dimension and nearest neighbors, we can mitigate this loss.

Deliverables

- Code to compute the nearest neighbors
- Code to compute the reconstruction weights
- Code for the computation of eigen-values/vectors
- Code for the LLE algorithm
- Testing script for all above code
- Files required from LibSVM
- Files required from dimensionality reduction toolbox

References

1. Sam Roweis and Lawrence Saul, Nonlinear Dimensionality Reduction by Locally Linear Embeddings, Science v.290 no.5500, Dec.22, 2000. pp.2323–2326.
2. Sergios Theodoridis and Konstantinos Koutroumbas, Pattern Recognition, Fourth Edition, Academic Press 2008.
3. Olga Kouropteva and Oleg Okun and Matti Pietikinen, Selection of the Optimal Parameter Value for the Locally Linear Embedding Algorithm, 1 st International Conference on Fuzzy Systems, 2002, 359–363.
4. O. Kouropteva and M. Pietikainen. Incremental locally linear embedding. Pattern Recognition, 38:17641767, 2005.
5. Boschetti and Fabio, Dimensionality Reduction and Visualization of Geoscientific Images via Locally Linear Embedding, Comput. Geosci., July, 2005, 31,6, 689–697.
6. Hong Chang and Dit-yan Yeung, Robust Locally Linear Embedding, 2005.
7. Chang, Chih-Chung and Lin, Chih-Jen, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology, 2, 3, 2011, 27:1–27:27.
8. Georghiades, A.S. and Belhumeur, P.N. and Kriegman, D.J., From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose, IEEE Trans. Pattern Anal. Mach. Intelligence, 2001, 23, 6, 643-660.
9. Zhang Z, Wang J (2007) MLLE: Modified locally linear embedding using multiple weights. Advances in Neural Information Processing Systems (NIPS) 19, eds Scho lkopf B, Platt J, Hofmann T (MIT Press, Cambridge, MA), pp 15931600.
10. O. Kouropteva, O. Okun, and M. Pietikanen. "Selection of the optimal parameter value for the locally linear embedding algorithm." (2002)