Nonlinear Dimensionality Reduction Applied to the Binary Classification of Images

Chae A. Clark
cclark18@math.umd.edu
University of Maryland, College Park

Advisor
Dr. Kasso A. Okoudjou
kasso@math.umd.edu
Department of Mathematics
University of Maryland, College Park

0. Abstract

In this project we are interested in the reduction of high-dimensional data points $\mathbf{x}$ from a space $\mathbb{R}^D$ to a lower dimensional space $\mathbb{R}^d$ (where $d << D$) in a way that preserves certain important characteristics. In particular, we are interested in reducing the size of high-dimensional points for their application in the binary classification of signals. We first examine a MatLab implementation of the Locally Linear Embeddings (LLE) algorithm, and it to a specific image database. We then use the output of the LLE and the original dataset to test and compare the performance ability of a MatLab implementation of the support vector machine.

1. Background and Motivation

In the subsections below we give an introduction to the problem of handling high-dimensional data, as well as define necessary notions used later in the proposal.

1.1. High-Dimensional Data

At a basic level, the field of dimension reduction is concerned with taking high-dimensional data points and representing them with fewer elements while preserving the important features of the data. In current data collection and processing applications, an enormous amount of data can now be stored easily and efficiently. This allows for tons of information to be collected on any specific task. The issue then becomes the usefulness of this high-dimensional data. Data points in higher dimensions will require more computing power to efficiently handle them, but it may not be true that all of the collected data is meaningful in any way. Redundant and unnecessary information may be present. This is the curse of Dimensionality; we want to collect more information to create more accurate model predictions, but we sometimes are not sure which information is relevant to the application at hand.

1.2. Dimension Reduction Algorithms

To combat the curse of dimensionality, various techniques and algorithms have

been developed to choose only the important features from a high-dimensional point. There are two fields in dimension reduction, linear techniques that use a linear mapping to reduce the dimension, and nonlinear techniques, which are the focus of this project, that make the assumption that the data available is embedded on a manifold (or surface in lower dimensional space). The data is then mapped onto a lower-dimensional manifold for more efficient processing.

Specifically for this project, we only consider the nonlinear reduction scheme, Local Linear Embeddings (LLE). In this scheme each high-dimensional data point $\mathbf{x}$ is represented by a linear combination of its nearest neighbors (in Euclidean distance). A lower-dimensional point $\mathbf{y}$ is then constructed to preserve local properties in an optimal way. This scheme is computationally simple and has other qualities that render it useful in a variety of signal processing applications. This is further discussed in the Approach section of the proposal.

1.3. Image Types and Processing

An image here, is defined to be three $m \times n$ matrices representing the red, green, and blue (RGB) color intensities of the image. A gray-scale image is defined to be a single $m \times n$ matrix of intensity values representing the image. Here we standardize the images to live in the range of $[0, 1]$ (as opposed to the usual $[0, 255]$). A binary image is defined to be a gray-scale image where every element of the matrix is in the set $\mathcal{B} = \{0, 1\}$.

With the rapid increase in camera (or detector) resolutions, digital images contain an astounding amount of pixels. And not surprisingly, in classification tasks, transforming the images into data points (usually stacking the images by row or column) and running algorithms on them can become computationally prohibitive. This is especially true when there are sets of these high resolution (or equivalently, high-dimensional) images. The solution to this computational nightmare lies in the fact that much of the image contains pixels irrelevant to the classification tasks.

1.4. Notation

$\mathbf{x}_i$ - data point in $\mathbb{R}^D$

$\mathbf{y}_i$ - data point in $\mathbb{R}^d$

$\mathcal{X}$ - a matrix of data points in $\mathbb{R}^D$ ($\mathcal{X} \in \mathbb{R}^{N \times D}$)

$\mathcal{Y}$ - a matrix of data points in $\mathbb{R}^d$ ($\mathcal{Y} \in \mathbb{R}^{N \times d}$)

$\mathbf{W}$ - a matrix of weight elements w_{ij} ($\mathbf{W} \in \mathbb{R}^{N \times N}$)

$\mathcal{S}_i$ - a set of data points

C - a matrix of correlation elements C_{ij} ($C \in \mathbb{R}^{K \times K}$)

$\mathcal{Q}$ - a matrix of eigenvectors

$\mathcal{I}_i$ - a matrix representing an image

z_i - the class designation of data point x_i

$\mathcal{H}$ - a hyperplane represented by its normal vector $\mathbf{v}$ and an offset term v_0

2. Approach

This section outlines the two methods to be examined. The dimension reduction algorithm Locally Linear Embeddings (LLE), and a binary classifier version of Support Vector Machines (SVM). Also discussed, are the implementation of these algorithms in the programming language MatLab, their extensions, and possible numerical difficulties and testing hardware.

2.1. Locally Linear Embeddings (LLE)

Locally Linear Embeddings is a nonlinear dimension reduction scheme that maps a high-dimensional data point $\mathbf{x} \in \mathbb{R}^D$ to a lower dimensional data point $\mathbf{y} \in \mathbb{R}^d$, where d is much smaller that D. It was developed by Dr. Sam Roweis and Dr. Lawrence K. Saul [1], and is described as a neighborhood preserving embedding algorithm. The high-dimensional points are assumed to lie on a well-behaved manifold where we further assume that any small patch of the manifold that we view is approximately linear. With this, we also assume that the data can be efficiently represented in a lower-dimensional space. This is equivalent to saying that there is irrelevant or redundant information in the data point. An LLE algorithm attempts to reduce the dimensionality of the data set $\mathcal{X} \in \mathbb{R}^{N \times D}$ to a dataset $\mathcal{Y} \in \mathbb{R}^{N \times d}$ in a way that preserves relative distances between "close" (in Euclidean distance) points.

The algorithm proceeds as follows.

Step 1: In the first step, we must find the $K-$nearest neighbors of each data point $\mathbf{x}_i \in X,\ i = 1, 2, \ldots, N$, as the nearest neighbors of x_i are what we want to preserve in our embedding. The K-nearest neighbors of $\mathbf{x}_i$ will be denoted $\{\mathbf{z}_{ij}\}_{j=1}^{K}$. Here, the subscript ij refers to the ith data point $\mathbf{x}_i$ and its jth neighbor $\mathbf{z}_{ij}$. It's important to note that we choose the K-nearest neighbors of $\mathbf{x}_i$, excluding $\mathbf{x}_i$ itself.

Step 2: Using the neighbors $\{\mathbf{z}_{ij}\}_{j=1}^{K}$, we now must compute the reconstruction weights $w_{ij},\ i, j = 1, 2, \ldots, N$ that when applied to the neighbors $\mathbf{z}_{ij}$, best approximate $\mathbf{x}_i$. This search for the weight w_{ij} is equivalent to solving the following constrained optimization problem.

$$\mathbf{W} = \arg\min_{\mathbf{W}} : E(\mathbf{W}) = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \sum_{j=1}^{K} w_{ij} \mathbf{z}_{ij} \right\|^2 \tag{1}$$

Here $\mathbf{W} \in \mathbb{R}^{N \times N}$ is the matrix of optimal weight values w_{ij}. We take the convention that all norms $\| \cdot \|$ are the Euclidean distance unless specified to be otherwise.

Our first constraint is that

$$w_{ij} = 0 \tag{2}$$

for any indices j, where $\mathbf{x}_j$ is not a neighbor of $\mathbf{x}_i$, or more succinctly when $\mathbf{x}_j \notin \{\mathbf{z}_{ij'}\}_{j'=1}^{K}$. This constraint ensures that the algorithm doesn't attempt to preserve the relative distance of points not in the nearest neighbor set. It's also important to note that $w_{ii} = 0$.

The second constraint is that

$$\sum_{j=1}^{N} w_{ij} = 1, \text{ for } i = 1, 2, \ldots, N \tag{3}$$

This essentially says that the rows of the weight matrix $\mathbf{W}$ must sum to one. These constraints are required to ensure that the solution to the problem above is invariant to rotation, rescaling, and translations of the data. The benefit of this constraint is that the weights are no longer dependent on the current frame of reference, and instead, they represent their relationships between data in the set.

Step 3: After solving this optimization problem, we now have the weight matrix $\mathbf{W}$ that best represents the data points $\mathbf{x}_i$ as a linear combination of its nearest neighbors. The next step in the algorithm is to find the best representation of $\mathbf{x}_i$ as the lower-dimensional point $\mathbf{y}_i$. This is done using the weights determined in step 2 and solving the following constrained optimization problem.

$$\arg\min_{\mathcal{Y}} : e(\mathcal{Y}) = \sum_{i=1}^{N} \left\| \mathbf{y}_i - \sum_{j=1}^{N} w_{ij} \mathbf{y}_{ij} \right\|^2 \tag{4}$$

The first constraint here is that

$$\sum_{i=1}^{N} \mathbf{y}_i = 0 \tag{5}$$

This constraint is used to center the points around the origin (to remove translational invariance).

The other constraint required here is that

$$\frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i^T \mathbf{y}_i = I_d \tag{6}$$

Here, I_a refers to the identity matrix of size $a \times a$, and $\mathbf{y}_i^T \mathbf{y}_i$ is defined to be the outer product (as $\mathbf{y}_i$ is a row vector). This constraint removes the rotational invariance.

2.2. Implementation

Using details from the original paper by Roweis and Saul [1], the book by Theodoridis and Koutroumbas [2], and the book by O'Leary [10], implementations of solution methods for the steps above are presented here.

Nearest Neighbor Search

Here, we present three methods for finding the K-nearest neighbors in Euclidean distance.

1. First computing the pairwise distances between each point in the dataset $\mathcal{X}$, we then sort them and take the K smallest to be the nearest neighbors. An algorithm is provided below.

Algorithm 1 Nearest Neighbor Search by Full Enumeration

for each point $\mathbf{x}_i \in \mathcal{X}$ **do**
    compute the distance between $\mathbf{x}_i$ and each other point $\mathbf{x}_j \in \mathcal{X}$
    sort distances in ascending order (keeping track of the original indices)
    choose the K indices with the smallest distances
    return the indices of the K nearest neighbors
end for

2. Another method is accomplished through Binary Programming. We wish to find the K points that are closest to $\mathbf{x}_i$. We can model this problem as

$$\begin{aligned} \mathbf{u} = \arg\min_{\mathbf{u}} : & \sum_{j=1}^{N} \|\mathbf{x}_i - u_j \mathbf{x}_j\|^2 \\ \text{s.t.} : & \sum_{j=1}^{N} u_j = K \\ & \mathbf{u} \in \{0,1\}^N \end{aligned} \tag{7}$$

   Solving this problem for the non-zero components v_j (for each point $\mathbf{x}_i$ in our set) that correspond to the points $\mathbf{x}_j$, we get our K-nearest neighbors. This must be performed for each point in our data set $\mathcal{X}$. There are two important things to remember with this method. The first is that it is generally slow to find the solution (as the problem is NP-hard). In fact the only reason this method is presented is to be a starting point for the next solution method presented. There are very few occasions when this method would be suggested. The algorithm is presented below.

Algorithm 2 Nearest Neighbor Search by Binary Programming

for each point $\mathbf{x}_i \in \mathcal{X}$ **do**
    find $\mathbf{u}$ by solving the Binary Programming problem described in (7)
    return the indices of the K non-zero elements in $\mathbf{u}$
end for

3. The final method presented for nearest neighbor search, is an approximate solution to the Binary Programming problem above. We start with the problem above, and by using Lagrange multipliers results in the following system. The full derivation is available in Appendix A.

$$A\mathbf{u} = \lambda \mathbf{1}_N \tag{8}$$

where $A \in \mathbb{R}^{N \times N}$ is a diagonal matrix whose elements are $(\|\mathbf{x}_j\|^2 - \langle \mathbf{x}_i, \mathbf{x}_j \rangle)$, or in other words, $A = \text{diag}\{\|\mathbf{x}_j\|^2 - \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, for $j = 1, 2, \ldots, N\}$. λ is a Lagrange multiplier whose value is set so that $\sum_{j=1}^{N} u_j = K$. $\mathbf{1}_N$ is a vector of ones of length N. An algorithm for nearest neighbor computation is presented below. The important thing to note with this method is that it's solution is an approximate one.

Algorithm 3 Approximate Nearest Neighbor Search

1: **for** each point $\mathbf{x}_i \in \mathcal{X}$ **do**
2: build the matrix $A = \text{diag}\{\|\mathbf{x}_j\|^2 - \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, for $j = 1, 2, \ldots, N\}$
3: solve the system $A\mathbf{u}' = \mathbf{1}_N$ for $\mathbf{u}'$
4: set $\lambda = \frac{K}{\sum_{j=1}^{N} u'_j}$
5: set $\mathbf{u}$ to be $\lambda \cdot \mathbf{u}'$
6: sort $\mathbf{u}$ in descending order (keeping track of the original indices)
7: return the top K indices that correspond to the nearest neighbors
8: **end for**

Minimization of $E(\mathbf{W})$

Here we present two methods for the solution of this minimization problem. The first, from Roweis and Saul [1], and the other from a report by Shalizi [11].

1. For a data point $\mathbf{x}_i$ with neighbors $\mathcal{S}_i = \{\mathbf{z}_{ij}\}_{j=1}^{K}$, compute the neighborhood correlation matrix $C \in \mathbb{R}^{K \times K}$, formed by taking the pairwise inner-products of the elements $\mathbf{z}_{ij}, \mathbf{z}_{ik} \in \mathcal{S}_i$, or $C_{jk} = \langle \mathbf{z}_{ij}, \mathbf{z}_{ik} \rangle$. Here, we view our data points $\mathbf{x}_i$ as sequences of measurements, such that their scaled correlation is $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. If the data points are normalized, then the elements of the correlation matrix C_{jk} are in the interval $[-1, 1]$. For each correlation matrix, find the inverse C^{-1}.

   Compute the Lagrange multiplier $\lambda = \frac{\alpha}{\beta}$ that enforces the sum-to-one constraint. Here,

$$\alpha = 1 - \sum_{j=1}^{K} \sum_{k=1}^{K} C_{jk}^{-1} (\mathbf{x}_i \cdot \mathbf{z}_{ik}) \tag{9}$$

   and

$$\beta = \sum_{j=1}^{K} \sum_{k=1}^{K} C_{jk}^{-1} \tag{10}$$

   It's worth noting that λ changes with the point $\mathbf{x}_i$.

Finally, compute the reconstruction weights for data point $\mathbf{x}_i$ using the formula

$$w_{ij} = \sum_{k=1}^{K} C_{jk}^{-1}[(\mathbf{x}_i \cdot \mathbf{z}_{ij}) + \lambda] \tag{11}$$

The above reconstruction states that for each point $\mathbf{x}_i$ and its neighbor $\mathbf{z}_{ij}$, the weight w_{ij} is computed as stated. In the cases where the correlation matrix are nearly singular, a small multiple of the identity matrix can be added to overcome major problems.

Algorithm 4 Weight Construction I

for each point $\mathbf{x}_i \in \mathcal{X}$ given its neighbors $\{\mathbf{z}_{ij}\}$ **do**
    compute the correlation matrix $C \in \mathbb{R}^{K \times K}$
    compute the inverse matrix $C^{-1} \in \mathbb{R}^{K \times K}$
    compute the Lagrange multiplier $\lambda = \frac{\alpha}{\beta}$
    return the vector of weights $w_{ij} = \sum_{k=1}^{K} C_{jk}^{-1}[(\mathbf{x}_i \cdot \mathbf{z}_{ij}) + \lambda]$
end for

2. The second method constructs the weights through solving the system

$$G\mathbf{w}_i = \frac{\lambda}{2}\mathbf{1}_N \tag{12}$$

Here, G is formed by first defining $\widetilde{\mathbf{z}}_{ij} := \mathbf{z}_{ij} - \mathbf{x}_i$ for $j = 1, 2, \ldots, N$. This centers our points around $\mathbf{x}_i$. We then define G to be the Gram matrix whose elements are inner products of $\widetilde{\mathbf{z}}_{ij}$ for all j. λ is the Lagrange multiplier whose value is set so that the sum-to-one constraint is satisfied ($\sum_{j=1}^{N} w_{ij} = 1$). For the full derivation, refer to Appendix B.

Algorithm 5 Weight Construction II

for each point $\mathbf{x}_i \in \mathcal{X}$ given its neighbors $\{\mathbf{z}_{ij}\}$ **do**
    center the points, $\widetilde{\mathbf{z}}_{ij} = \mathbf{z}_{ij} - \mathbf{x}_i$
    compute the Gram matrix $G = [G_{jk}]$ whose elements are $\langle \widetilde{\mathbf{z}}_{ij}, \widetilde{\mathbf{z}}_{ik} \rangle$
    solve the system $G\mathbf{w}_i' = \frac{1}{2}\mathbf{1}_N$ for $\mathbf{w}_i'$
    compute the Lagrange multiplier $\lambda = \frac{1}{\sum_{j=1}^{K} w_j'}$
    set $\mathbf{w}_i = \lambda \cdot \mathbf{w}_i'$
    return the row of weights $\mathbf{w}_i$
end for

Minimization of $e(\mathcal{Y})$

1. For the minimization of $e(\mathcal{Y})$, and the computation of $\mathcal{Y}$, it has been that by performing an eigen-decomposition of $(\mathbf{W} - I_N)^T(\mathbf{W} - I_N)$. We denote the matrix whose columns are eigenvectors $\mathcal{Q} = [\mathbf{q}_1\, \mathbf{q}_2\, \ldots\, \mathbf{q}_D]$. Our lower-dimensional dataset (whose rows are data points) is then set to be $Y =$

$[\mathbf{q}_1\, \mathbf{q}_2\, \ldots\, \mathbf{q}_d]$, where the eigenvectors correspond to the smallest eigenvalues (this assumes the eigenvalue/eigenvector pair corresponding to an eigenvalue of 0 is removed). An algorithm is presented below.

Algorithm 6 Embedding Construction

1: construct the matrix $(\mathbf{W} - I_N)^T(\mathbf{W} - I_N)$
2: compute the eigenvalues and eigenvectors of $(\mathbf{W} - I_N)^T(\mathbf{W} - I_N)$
3: remove the eigenvalue/eigenvector pair corresponding to the eigenvector of 0
4: collect the eigenvectors corresponding to the lowers d eigenvalues
5: return the eigenvectors $[\mathbf{q}_1\, \mathbf{q}_2\, \ldots\, \mathbf{q}_d]$ as the dataset $\mathcal{Y}$

2.3. Computational Considerations

The computational complexity of minimizing $E(\mathbf{W})$ scales as $O(NK^3)$, due to the solving of linear systems in the solution method, but we expect to have many more data points than nearest neighbors so our weight matrix should be fairly sparse. Using this fact to our advantage, we can implements sparse methods to handle the linear systems more efficiently, thereby reducing computation time. The efficient use of the sparsity in $\mathbf{W}$ can also result in a complexity sub-quadratic in N for the eigen-decomposition problem. The algorithm used in MatLab eig.m, which allows for the efficient computation of eigenvectors when structure is present in the matrices. The complexity of the full enumeration search is quadratic in N.

When performing the computing tasks on the dataset $\mathcal{X}$, efficient memory management is imperative. Understanding the programming languages memory allocation, storage, and retrieval system will allow for efficient and steam-lined code. This will reduce the setup time between steps in the LLE algorithm.

2.4. Algorithm Extensions

There are a number of possible extensions to this algorithm. Time permitting, these extensions can increase the effectiveness and efficiency of our LLE implementation.

The number of nearest neighbors K taken is an important parameter to the weight construction, and in the algorithm above, there is no consideration to the choice of this variable. In the paper by Kouropteva, Okun, and Pietikainen, a method is presented for the optimal selection of the number of nearest neighbors [3].

In another paper by Kouropteva, Okun, and Pietikainen, a method is presented for the incremental implementation of the LLE algorithm. This allows for the solving of smaller optimization problems (as opposed to the original algorithm, or batch LLE) [4].

And one of the things that cause the LLE algorithm to fail is when the least-squares problem, solved to find the weights, is ill-conditioned. To combat this, Zhang and Wang propose a method of finding weight vectors that are linearly independent and nearly optimal [9].

2.5. Support Vector Machines

The specific application of this proposed project is in image classification. Given a set of images $\{\mathcal{I}_i\}$, we want to be able to correctly distinguish between their various properties. Here we take the simple case of binary classification, where we assume that any image considered is either in a class $\mathcal{A}$ or class $\mathcal{B}$. One of the main goals in pattern recognition is to be able to detect the differences between various objects. There are a variety of ways to do this, but one of the most popular utilizes Support Vector Machines (SVM). Various features of the image under study are arranged into a vector (data point $\mathbf{x}_i$) that is a representation of its properties. Each data point is then assigned one of the classes above. For notational convenience, we will designate z_i as the class designation of a data point $\mathbf{x}_i$, where $z_i \in \{-1, 1\}$.

With support vector machines, a number of these data points are collected and called a training set, as their classes are known. These are used to find a hyperplane $\mathcal{H} = \langle \mathbf{v}, v_0 \rangle$ that separates the two classes. Here, $\mathbf{v}$ is the normal vector to the hyperplane. v_0 is the offset of the hyperplane from the origin. In general there are an infinite number of hyperplanes that have this property, but the hyperplane desired is the one that maximizes the distance between elements in a class and the hyperplane itself (this is called the maximal margin hyperplane). The objective for this problem becomes finding the hyperplane whose normal vector $\mathbf{v}$ accurately classifies each training vector and has maximal margin. Our problem can be solved through constrained optimization, formulated below.

$$\arg\min : \frac{1}{2}\|\mathbf{v}\|^2 \tag{13}$$

This problem is subject to the constraint that **(a)** all of the data points $\mathbf{x}_i$ are labeled correctly, or

$$z_i(\mathbf{v}^T\mathbf{x}_i + v_0) \geq 1 \tag{14}$$

This problem will only have a solution if there exists a hyperplane that separates the data. To account for the case where our data is not separable in this way, we can relax the constraints, or form the Lagrangian problem, presented below.

$$\arg\min L(\mathbf{v}, v_0, \mu) = \frac{1}{2}\|\mathbf{v}\|^2 - \sum_{i=1}^{N} \mu_i[z_i(\mathbf{v}^T\mathbf{x}_i + v_0) - 1] \tag{15}$$

With this, we can test classification accuracy, when the dataset output for LLE is applied.

2.6. Software

The algorithms stated above are to be implemented in the programming language MatLab. This decision is primarily due to the languages' flexibility in syntax, its ubiquitous use by the scientific community, and the wide availability of support and toolboxes. In particular, the optimization, linear algebra, and sparse matrix support, make it an ideal choice for scientific computing tasks.

2.7. Hardware

In these early planning stages, the planned machine to run and develop these algorithms is a personal computer. We plan to scale up to the computers available in the Norbert Weiner Center computer lab if more computing power is necessary. But as of now, no special hardware seems to be required.

3. Validation

There are two main methods we plan to use to validate our implementation of LLE.

3.1. Standard Topological Manifolds

To test the correctness of the LLE implementation, we will apply the algorithm to data points sampled from well-known and well-studied 3-D surfaces that have a well-known representation in 2-D space. The four functions we plan to use are presented below.

Swiss Roll:

$$F : (x, y) \to (x\cos(x), y, x\sin(x)) \tag{16}$$

Gaussian Distribution:

$$f(x, y) = \frac{1}{\sqrt{2\pi}} e^{-\left(\frac{x^2+y^2}{2}\right)} \tag{17}$$

Twin Peaks Function:

$$g(x, y) = x^4 + 2x^2 + 4y^2 + 8x \tag{18}$$

Logistic Function:

$$h(x, y) = \frac{1}{1 + e^{-x}} \tag{19}$$

For these functions, the 2 dimensional manifold is known, which allows us to check our LLE output against the known 2-D manifolds of these surfaces.

3.2. MatLab Dimension Reduction Toolbox

Available through Delft University of Technology, there is a dimensionality reduction toolbox for MatLab with various reduction algorithms (including an LLE implementation). Using this toolbox, and some data from the database we plan to use, the results can be compared to ensure that we have implemented LLE correctly.

3.3. Validation of Nearest Neighbors

In this section, the validation results for the three nearest neighbor algorithms vs MatLab's nearest neighbor algorithm (knnsearch.m) are presented. knnsearch.m is

partially based on work by Friedman, Bently, and Finkel [12] utilizing k-d search for nearest neighbor searching (essentially partitioning the space and grouping close points). They were able to prove that the expected complexity of performing can be logarithmic (in the average case). In larger dimensions though, it is not suitable as it essentially performs a full enumeration. It is worth mentioning that for data points larger than 10, knnsearch.m uses an exhaustive search instead of a k-d tree.

Presence of Neighbors

In this test, given a random data matrix, the indices of the K nearest neighbors are chosen using the four methods (Full Enumeration, Binary Programming, Heuristics, MatLab's Implementation), and then checking the results of our algorithms vs. MatLab's implementation for the correspondence of nearest neighbor indices. By the presence of nearest neighbors, we mean that in our vector of indices, if an index is present in the our implementation and MatLab's (regardless of order), then they are considered to produce the same result. The results are presented below.

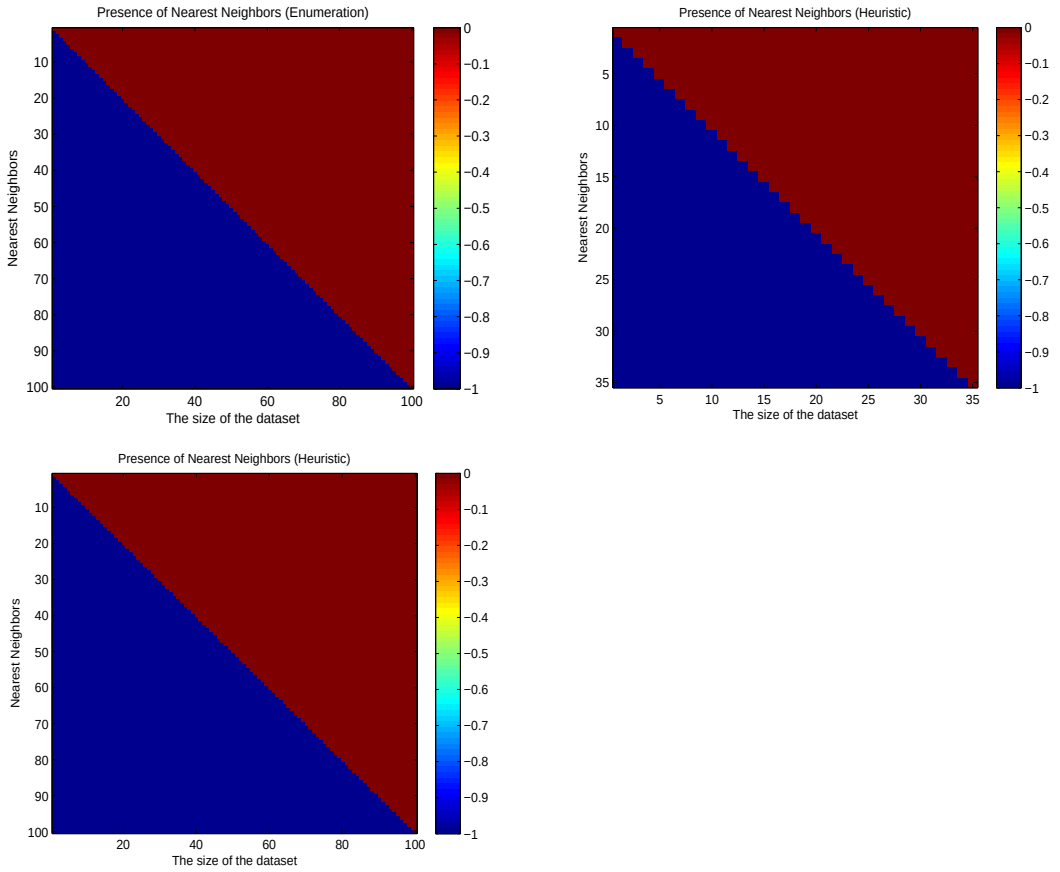

Figure 1: (Top Left) The error percentage using Full Enumeration. (Top Right) The error percentage using Binary Programming. (Bottom Left) The error percentage using Heuristics.

In the figures above, it is clear that all of our implementations reach near parity with the MatLab results.

Preservation of Neighbors

In this test, the same method is followed as in the "Presence of Neighbors" test above, with the exception that now the order of the neighbors is considered. This means that even if all of the same neighbor indices are present in the results, if they are not in the same order, they are not considered the same. The graphical results are below.

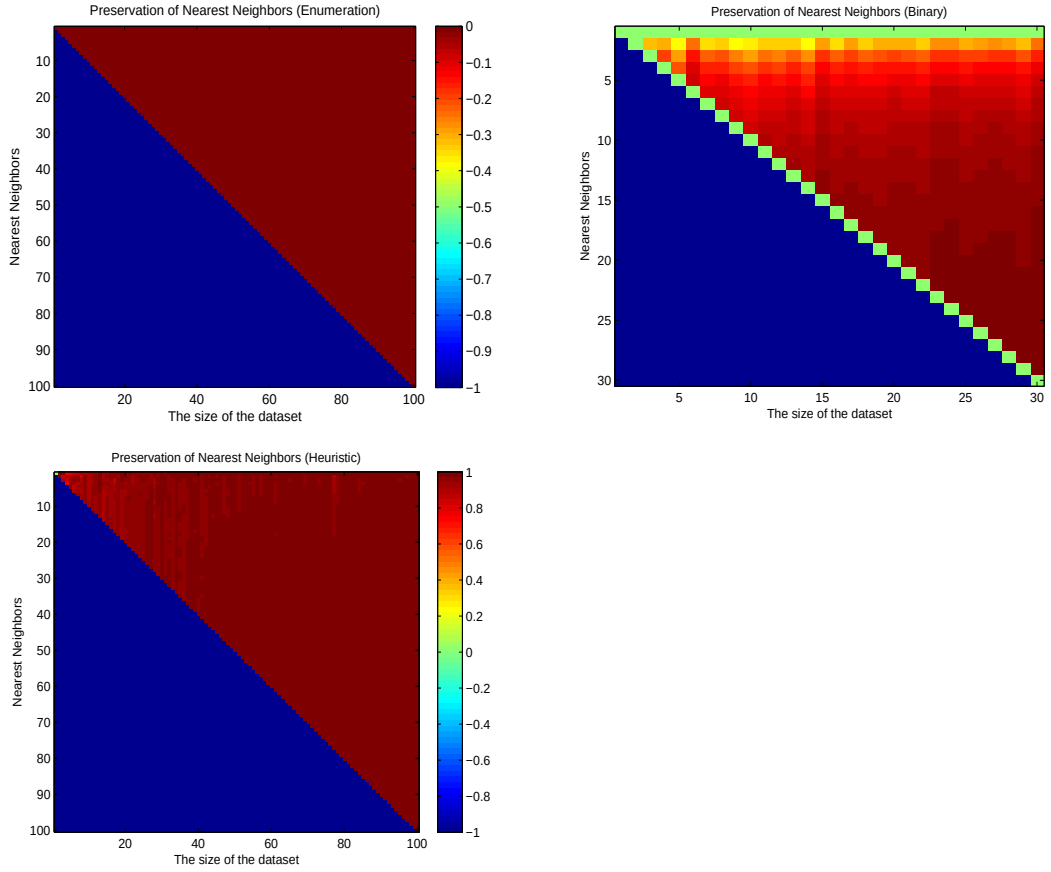

Figure 2: (Top Left) The error percentage using Full Enumeration. (Top Right) The error percentage using Binary Programming. (Bottom Left) The error percentage using Heuristics.

In the figures above, it is clear that while the Full Enumeration method produces analogous results, the Binary Programming and Heuristic algorithms can be vastly different.

3.4. Weight Construction Validation

There are two variations of the Weight Construction algorithm to validate. Rather than checking output of the methods, we will check that the sum-to-one constraints have been satisfied. Given a random dataset, and using the Full Enumeration

nearest neighbor search method, the weights are constructed and checked that the weights for each data point sum to 1. Graphical results are presented below. As is clear from the graph, both methods produce results that meet the constraints.

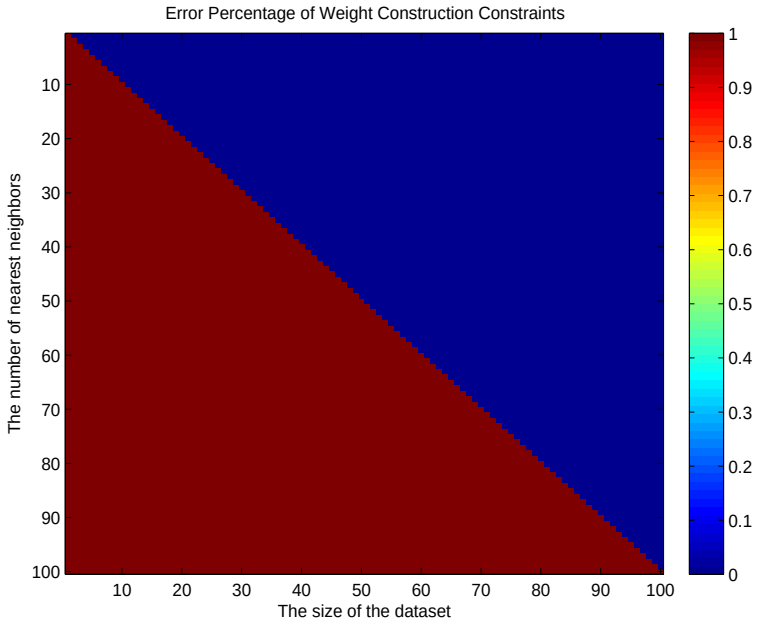

Figure 3: The error percentage of sum-to-one constraints.

3.5. Author's Locally Linear Embedding Code

Available from the LLE author's website (http://www.cs.nyu.edu/ roweis/lle/) there is an implemented and validated code set for performing Locally Linear Embedding. Using this code, and some of the test function mentioned in Standard Topological Manifolds, the results can be compared to ensure an accurate implementation.

3.6. Validation of LLE Algorithm

With the constituent parts validated, we can now validate the entire LLE algorithm against the Author's implementation. The first set of validations are the correspondence of results for the Topological Surfaces, the outputs of the two algorithms are available further down the paper.

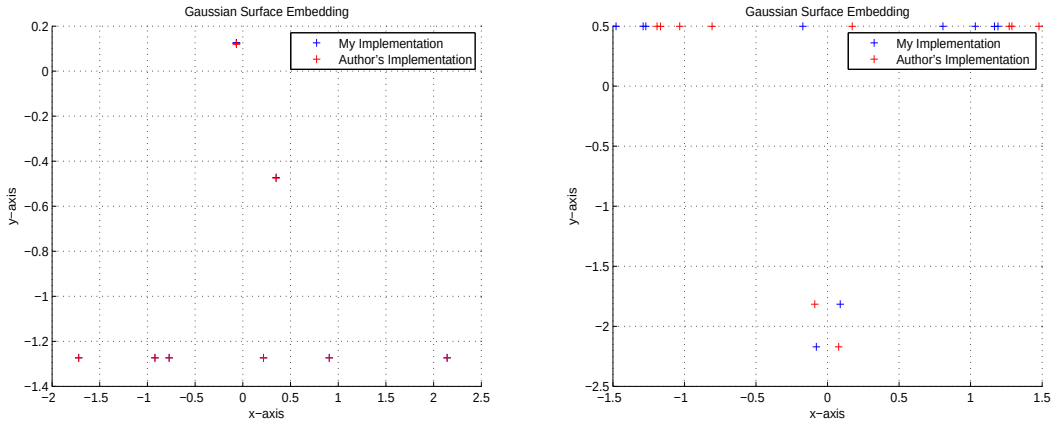

Figure 4: (Left) Gaussian Embedding with a norm difference of absolute value $7.3131e-10$. (Right) Gaussian Embedding with a norm difference of absolute value $5.1161e-12$

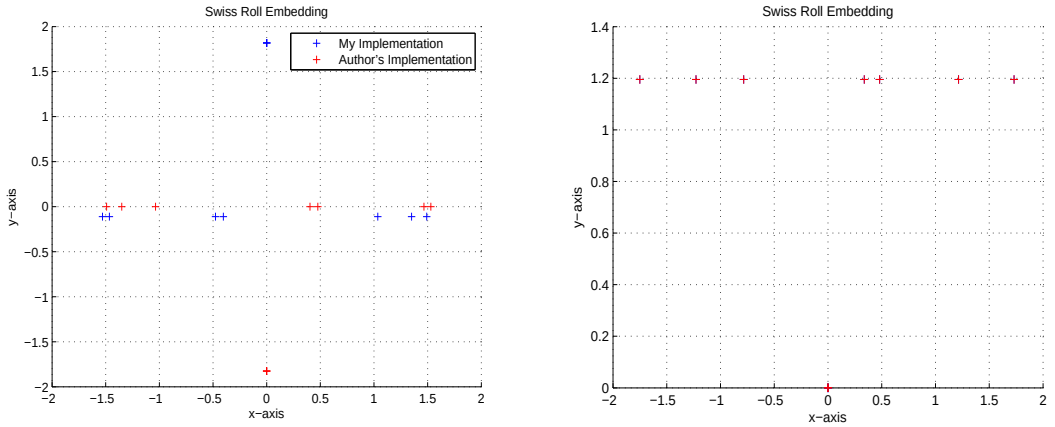

Figure 5: (Left) Swiss Roll Embedding with a norm difference of absolute value $5.2766e-13$. (Right) Swiss Roll with a norm difference of absolute value $5.9560e-13$

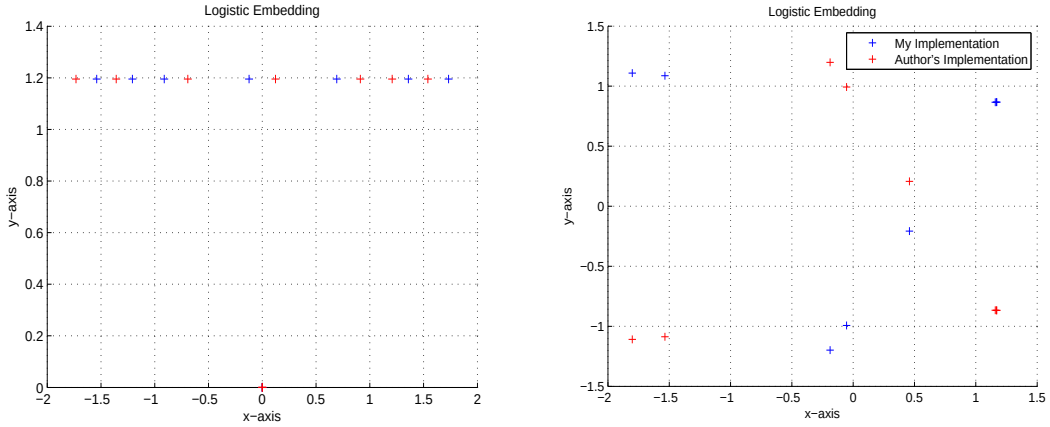

Figure 6: (Left) Logistic Embedding with a norm difference of absolute value $1.4568e-12$. (Right) Logistic Embedding with a norm difference of absolute value $1.3797e-9$

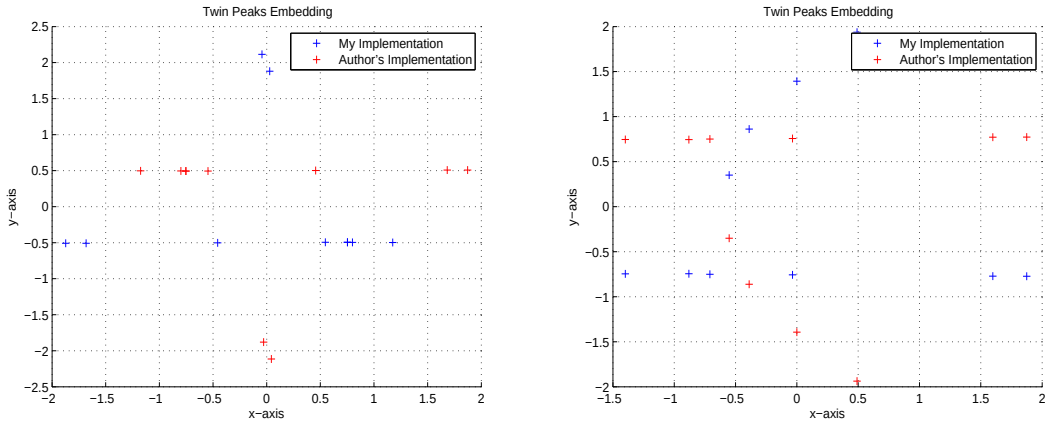

Figure 7: (Left) Twin Peaks Embedding with a norm difference of absolute value $1.8453e-9$. (Right) Twin Peaks Embedding with a norm difference of absolute value $4.6439e-13$

The next test is to use a randomly generated dataset and running the two implementations for varying sizes and nearest neighbors used. As the embedding dimension is generally higher than three, we can't graph the output points, so the results below will compare the nearest neighbors presence and preservation of the two implementations. The norm of the difference is also presented.

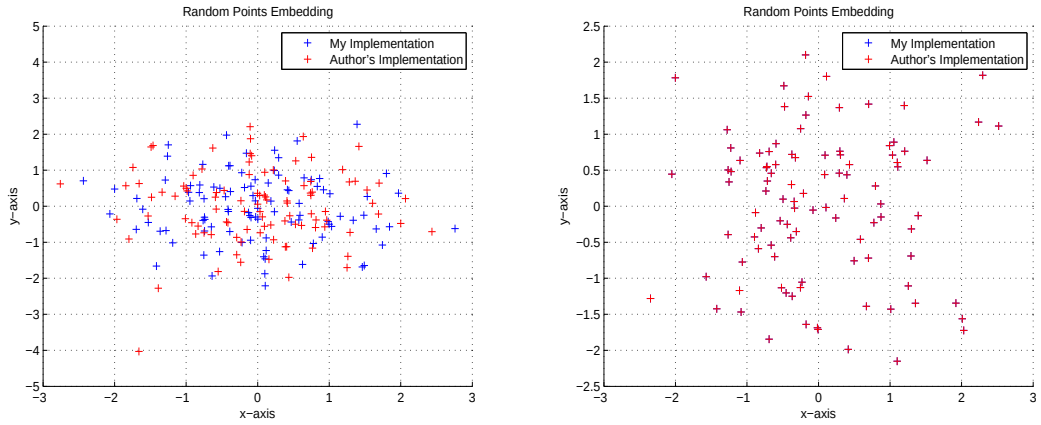

Figure 8: (Left) Random Data Embedding with a norm difference of absolute value $1.1165e-11$. (Right) Random Data with a norm difference of absolute value $2.2495e-12$

3.7. Validation of SVM

Time-permitting, we plan to implement certain components of the support vector machine library (see section 4.). To ensure that any code that we implement is working properly, we must validate it. The library (LIBSVM) comes packed with an array of testing/validation data, where the classification result is already known. Using this, we can be assured of our implementation.

4. Testing and Data

To test our implementation of LLE and SVM for image classification, there are a number of public databases available, but specifically, our images will be faces supplied by Yale University. Titled "The Yale Face Database B", the database contains 5760 single light source images of 10 different subjects, each under 576 viewing conditions. Included with the database are 65 background illuminations used when photographing the subjects [8].

Using this as our dataset, we can test our implementation of the LLE algorithm, and then use these results in conjunction with the support vector machine. This database is available at http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html.

The SVM is initially to be used from the library LIBSVM, which contains an implementation of the support vector machine. Form this we can test the classification accuracy of data after LLE has been applied. Time-permitting, this may be substituted for an implementation we create [7].

The library is available here http://www.csie.ntu.edu.tw/ cjlin/libsvm/faq.html#f203

5. Project Schedule and Milestones

September 2012 - November 2012
Plan and implement the LLE algorithm in MatLab, efficiently handling storage and memory management issues.

Perform unit tests to correct any bugs present in code.
Validate code on standard topological structures (Swiss Roll, etc.).
Compare results of algorithm output to the results of the LLE method present in the Dimension Reduction Toolbox.
Test the LLE algorithm on a dataset from a publicly available database.

November 2012 - December 2012
Make any necessary preprocessing changes to the image database used.
Prepare the mid-year (end of semester) report and presentation.
Deliver mid-year report.

January 2013
Implement a pre-developed SVM package for MatLab.
Test binary classification accuracy of SVM on dimension-reduced dataset.
Assess effectiveness.

February 2013 - April 2013
Implement SVM in MatLab (time permitting).
Implement LLE extensions.
Compare results of original LLE implementation to extended versions.
Parallelize original LLE algorithm (time permitting).

April 2013 - May 2013
Prepare final presentation and report.
Make any last minute adjustments to code that are required.
Package deliverables.
Ensure the safe delivery of source code and other project materials.

6. Deliverables

The deliverables for this project are the MatLab code that implements the LLE algorithm and any code used for testing (i.e. scripts for the surface creation, MatLab toolbox files, and other pre-packaged code). The code will be optimized for performance and effective memory management, as well as being fully documented. Reports at various stages throughout the course will detail the approach, implementation, validation, testing, and extensions of the algorithm. With this information, a researcher will be able to reproduce any results present in our reports.

7. 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 Pietikäinen, 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, 38:1764–1767, 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 1593–1600.

[10] Dianne P. O'Leary, Scientific Computing with Case Studies, SIAM Press, Philadelphia, 2009.

[11] Cosma Shalizi, Nonlinear Dimensionality Reduction I: Local Linear Embedding, Data Mining, 2009.

[12] Friedman, Jerome H. and Bentley, Jon Louis and Finkel, Raphael Ari, An Algorithm for Finding Best Matches in Logarithmic Expected Time, ACM Trans. Math. Softw., Sept. 1977, 3, 3, 1977.

Appendix A (Heuristic Nearest Neighbor Search Derivation)

Start with the Binary Programming problem:

$$\begin{aligned} \mathbf{u} = \arg\min_{\mathbf{u}} : & \sum_{j=1}^{N} \|\mathbf{x}_i - u_j \mathbf{x}_j\|^2 \\ \text{s.t.} : & \sum_{j=1}^{N} u_j = K \\ & \mathbf{u} \in \{0,1\}^N \end{aligned} \tag{20}$$

Expanding this, we have

$$\begin{aligned} \min_{\mathbf{u}} : & \sum_{j=1}^{N} \|\mathbf{x}_i - u_j \mathbf{x}_j\|^2 \\ & \sum_{j=1}^{N} \|\mathbf{x}_i\|^2 - 2\langle \mathbf{x}_i, u_j \mathbf{x}_j \rangle + u_i^2 \|\mathbf{x}_j\|^2 \end{aligned}$$

As our variables u_j are binary, $u_j^2 = u_j$. We can then simplify our objective function

$$N\|\mathbf{x}_i\|^2 + \sum_{j=1}^{N} u_i^2 (\|\mathbf{x}_j\|^2 - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle)$$

We will now form the Lagrangian using the constraint.

$$\begin{aligned} L(\mathbf{u}) =& N\|\mathbf{x}_i\|^2 + \sum_{j=1}^{N} u_i^2 (\|\mathbf{x}_j\|^2 - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle) + \lambda (K - \sum_{j=1}^{N} u_j) \\ \nabla_{\mathbf{u}} L =& 2 \cdot \text{diag}\{\|\mathbf{x}_j\|^2 - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle\}\mathbf{u} - \lambda \cdot \mathbf{1}_N = \mathbf{0} \end{aligned} \tag{21}$$

This gives the system

$$2 \cdot \text{diag}\{\|\mathbf{x}_j\|^2 - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle\} = \lambda \cdot \mathbf{1}_N$$

Appendix B (Weights Construction Derivation) The find the weights for the data point $\mathbf{x}_i$, we minimize

$$\min_{\mathbf{W}} : E(\mathbf{W}) = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \sum_{j=1}^{K} w_{ij} \mathbf{z}_{ij} \right\|^2$$

Our weights sum to one, implying that

$$\sum_{j=1}^{K} w_{ij} \mathbf{x}_i = \mathbf{x}_i$$

We can now write our objective function as

$$\min_{\mathbf{W}} : E(\mathbf{W}) = \left\| \sum_{j=1}^{K} w_{ij}(\mathbf{x}_i - \mathbf{z}_{ij}) \right\|^2$$

which we can further re-write as

$$\min_{\mathbf{W}} : E(\mathbf{W}) = \left\| \sum_{j=1}^{K} w_{ij}(\mathbf{z}_{ij} - \mathbf{x}_i) \right\|^2$$

Let $\tilde{\mathbf{z}}_i = [\mathbf{z}_{i1} - \mathbf{x}_i \, , \, \mathbf{z}_{i2} - \mathbf{x}_i \, , \ldots]$. Expanding the norm, we have

$$E(\mathbf{W}) = \mathbf{w}_i^T \tilde{\mathbf{z}}_i^T \tilde{\mathbf{z}}_i^T \mathbf{w}_i = \mathbf{w}_i^T G \mathbf{w}_i$$

Forming the Lagrangian with our sum-to-one constraint, we have

$$L(\mathbf{w}_i, \lambda) = \mathbf{w}_i^T G \mathbf{w}_i - \lambda(\mathbf{1}^T \mathbf{w}_i - 1)$$

$$\nabla_{\mathbf{w}_i} L = 2G\mathbf{w}_i - \lambda \mathbf{1} = \mathbf{0}$$

This is now the system presented in the weights construction II algorithm.