Nonlinear Dimensionality Reduction for Hyperspectral Image Classification

Midyear Report

Tim Doster (tdoster@umd.edu)
Advisors:
Dr. John Benedetto (jjb@math.umd.edu)
Dr. Wojciech Czaja (wojtek@math.umd.edu)

December 17, 2010

Abstract

Today with sensors becoming more complex and cost no longer a deterrent to storing large amounts of data; analysts need methods to reduce the volume of stored data and reveal its important facets. Dimensionality reduction, particularly non-linear dimensionality reduction, is a solution to this problem. In this paper, we will look at two nonlinear dimensionality reduction algorithms, Local Linear Embedding and ISOMAP. These algorithms both have been shown to work well with artificial and real world data sets, but are computationally expensive to execute. We solve this problem for both algorithms by applying landmarks or out of sample extensions which perform computationally expensive calculations on a small subset of the data and then map projection onto non-landmark data . Finally, we will apply these algorithms first to artificial data sets for validation and then to hyperspectral images for the application of classification.

1 Background

Dimensionality reduction is a field of mathematics that deals with the complexities of very large data sets and attempts to reduce the dimensionality of the data while preserving the important characteristics of the data. These algorithms are becoming more important today because the complexity of sensors have increased as well as the ability to store massive amounts of data. For example, hyperspectral sensors, which we will discuss below, record roughly a hundred times the amount of information as a typical optical sensor. With this high number of dimensions being recorded it becomes no longer feasible for analysts to examine the data without the assistance of computer algorithms to reduce the number of dimensions but still keep the intrinsic structure of the data intact.

There are two main branches of dimensionality reduction: linear and non-linear. In this project we will focus on non-linear algorithms as they have been shown to perform at least as well as linear algorithms but in many cases much better. We have chosen to study two of the leading non-linear algorithms, of about fifteen [6], in the field of dimensionality reduction: Local Linear Embedding and ISOMAP. The details of these algorithms will be presented in following sections.

A hyperspectral image (HSI), in general, has hundreds of spectral bands in contrast to a normal digital image which has three spectral bands (blue, red, and green) and thus offers a more complete part of the light spectrum for viewing and analysis [7]. This high dimensionality makes HSI good candidates for the methods of dimensionality reduction. A regular digital image can be viewed as a collection of three-dimensional spectral vectors, each representing the information for one pixel. Similarly a hyperspectral image can be viewed as a collection of D-dimensional spectral vectors, each representing the information for one pixel. Hyperspectral images typically include spectral bands representing the ultraviolet (200-400 nanometers), visible (400-700 nanometers), near infrared (700-1000 nanometers), and short-wave infrared (1000-4000 nanometers). In Figure 1, a representation of the light spectrum is shown with the approximate coverage of a hyperspectral image.

Thus, HSI are favored over regular images for some applications such as forestry [14] and crop analysis [5], mineral exploration [10], and surveillance [8]. The spectrum of vegetation, for example, is quite different from that of man-made objects (around 1100-1600 nanometers) even if painted to camouflage in with local vegetation. In this case, a simple photograph would not be able to pick out the man made objects as well as a hyperspectral image [8]. A hyperspectral image can produce a traditional

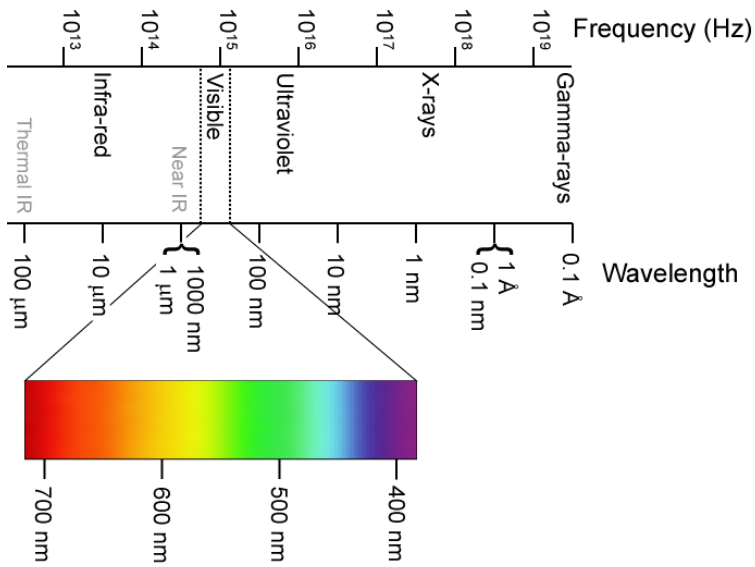


Figure 1: Electromagnetic Spectrum showing the ultra violet, visible, near-infrared, and shortwave infrared

red-blue-green image by resampling the image using the human visual response or any three spectral bands desired.

HSI are collected with special detectors that can be placed on high structures, flown in planes, or contained in satellites. For example, if a plane is used to collect the data, it will record the amount solar radiation reflected back from the ground at specific wavelengths line by line (like a push broom). Later the readings can be assembled, with necessary smoothing done to remove effects from the uneven travel of the plane, into a complete hyperspectral image [12]. The sensor onboard the plane works by collecting the emitted solar radiation that is reflected off the ground or object on the ground. As the solar radiation enters the atmosphere, it is altered by the presence of water molecules and other particulate matter in the atmosphere as shown in Figure 2. The same effect happens once the solar radiation is reflected off the ground or object. The data that are recorded by the sensor are known as the radiance spectrum. The reflectance spectrum for a particular band is the ratio of the reflected radiation at that band to the incident radiation at that band, and can be recovered from the collected radiation spectrum by using atmospheric correction equations. In this paper we will use the Quick Atmospheric Correction (QUAC) algorithm to correct any raw

images[2].

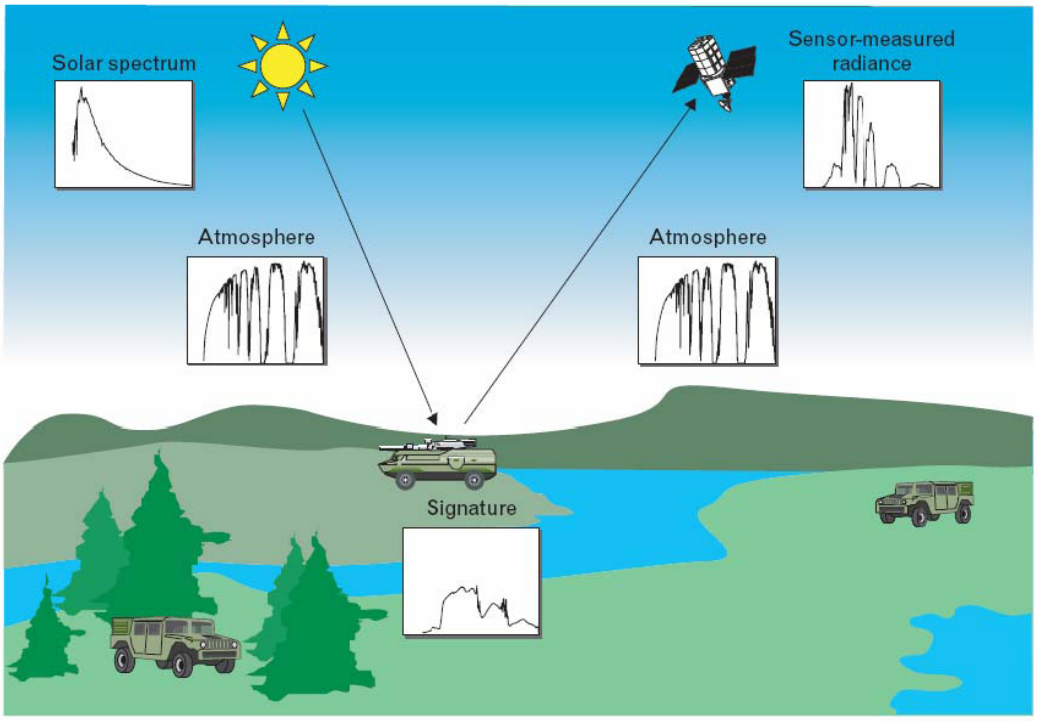


Figure 2: The path of solar radiation from the sun to the hyperspectral sensor (in this case on a satellite) [7]

One of the areas of research into HSI is image classification. The major goal of image classification is to classify the pixels of an image into some number of classes with the use of training data. This allows for example the creation of vegetation maps near wetlands. Bachmann [1] proposes using non-linear dimensionality reduction algorithms to first process the data into a lower dimension before using classification algorithms on the data set. This allows for the similarities and dissimilarities of the data members to become more evident as well as reducing the computation time (though this is greatly offset by the dimensionality reduction algorithm complexities).

2 Dimension Reduction Algorithms

We will apply Local Linear Embedding and ISOMAP to our hyperspectral images because due to the similarity among spectral bands that are next to each other, hyperspectral data can be assumed to lie on a lower dimensional manifold.

2.1 Local Linear Embedding

Local Linear Embedding (LLE) [11, 9] developed by Saul and Roweis is a nonlinear manifold-based approach to dimensionality reduction. LLE seeks to preserve the local properties for each data point when the data is projected to a lower dimension.

The LLE algorithm contains three major steps (note that steps 1 and 2 are often done in unison for efficiency) and proceeds as follows:

Step 0: Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of vectors (in our case the spectrum of each pixel) with $X_i \in \mathbb{R}^D$.

Step 1: Create a directed graph, G_k, for the data set X, where node X_i is connected to node X_j if it is one of the k-nearest-neighbors (KNN) of X_i. Any metric can be used for the KNN calculation but Euclidean (which we will use) and spectral angle (the angle between two pixels when viewed as vectors in R^D) are the most common. For X_i calculate the vector $E = [E_1, E_2, \ldots, E_n]$ where $E_j = ||X_i - X_j||_2$ and let E' be a vector corresponding to the indices of E sorted such that the smallest distances appear first. Let $U_i = [E'_2, E'_3, \ldots, E'_{k+1}]$, we will call U_i the KNN for X_i. Let $U = \{U_i\}_{i=1}^N$.

A directed graph is a collection of objects called nodes with an associated set of ordered pairs of nodes called edges. Each edge has a weight attached to it defining the distance between the nodes it connects. A directed graph differs from a graph in that edges are only traversable in one direction. To find the

Step 2: Calculate the reconstruction weights, W, by minimizing the cost function:

$$E(W) = \sum_{i=1}^{N} |X_i - \sum_{j \neq i} W_{i,j} X_j|^2.$$

To find $W(i, j)$, the cost function is minimized subject to $W(i, l) = 0$ if $X_l \notin U_i$ and

$\sum_{j=1}^{N} W(i,j) = 1$. By forcing the weights to sum to 1 we are removing the effects of translations of points. The use of the cost function ensures that points are not dependent upon rotations and rescaling. Now the set of weights will represent the underlying geometric properties of the data set.

In practice we find the reconstruction weights, W_i, for each X_i, by finding $C = A^T A$, where $A = [X_{U_{i,1}}, X_{U_{i,2}}, \ldots, X_{U_{i,k}}] - [X_i, X_i, \ldots, X_i]$ or the matrix of X_i centered neighbors of X_i. We then solve $CW_i = 1$ and let $W_i = W_i / \sum W_i$. Now we let the sparse matrix $W = [W_1, W_2, \ldots, W_n]$.

Step 3: Now by use of a similar cost function we will map each X_i to a lower dimensional Y_i. The cost function we are minimizing is:

$$\Phi(Y) = \sum_{i=1}^{N} |Y_i - \sum_{j \neq i} W_{i,j} Y_j|^2,$$

and we minimize it by fixing $W(i,j)$ and optimizing Y_j. Saul and Roweis were able to show that minimizing the cost function is equivalent to finding the $d+1$ smallest eigenvalues, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{L+1}$, and their corresponding eigenvectors, $V_1, V_2, \ldots, V_{d+1}$ of the matrix $(I - W)^T (I - W)$. We reject the smallest eigenvector as it is the unit vector with eigenvalue 0. Now we use the remaining eigenvectors to map X from D dimensions to d dimensions: $X_i \mapsto (V_2(i), V_3(i), \ldots, V_{d+1}(i))$.

2.2 ISOMAP

ISOMAP [15] developed by Tenenbaum, Silva, and Langford, is a nonlinear manifold based approach to dimensionality reduction like LLE. ISOMAP tries to maintain the geodesic distances between points in the data set when the data is projected down. By focusing on geodesic distance and not the distance in the higher dimensional space ISOMAP is less prone to short circuiting. The algorithm is as follows:

Step 0 and 1: Apply Step 0 and 1 from LLE algorithm.

Step 2: Let G be a graph constructed from the information in G_K. The edge (i,j), the distance from X_i to X_j, will be defined as the pairwise Euclidean distance if $X_j \in U_i$ and if $X_j \notin U_i$ then the distance will be ∞. Now find the shortest geodesic distance matrix S such that $S_{i,j}$ is the minimum geodesic distance between X_i and

X_j. To find the pairwise shortest path distance we will use either Floyd-Warshall [4] or Dijkstra's algorithm [4].

Step 3: Find the optimal embedding by minimizing the cost function:

$$\Phi(Y) = \sum_{i,j}^{N} |S_{i,j}^2 - ||Y_i - Y_j||^2.$$

Tenenebaum showed that minimizing the cost function is equivalent to finding the d principle eigenvalues, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_d$, and their corresponding eigenvectors, $V_1, V_2, \ldots, V_d$ of $\frac{1}{2}H^T S^2 H$, where H=$I - \frac{1}{n}1^T 1$ is defined as the centering matrix where I is the identity matrix and 1 is a vector of ones. Now we map X from D dimensions to d dimensions: $X_i \mapsto (\sqrt{\lambda_1}V_1(i), \sqrt{\lambda_2}V_2(i), \ldots, \sqrt{\lambda_d}V_d(i))$, were the eigenvalues are sorted descending.

2.2.1 Floyd-Warshall Algorithm

Floyd-Warshall is a $O(n^3)$ algorithm for finding the shortest pairwise distance in a graph with n nodes. It requires only that the graph contain no negative weights or distances between nodes (or pixels in our case) which will not occur due to our problem definition. The algorithm is as follows:

Step 1: Create a matrix M from a graph G such that $M(i,j)$ corresponds to the weight between node i and node j. If node i and j are not connected in G then $M(i,j) = \infty$; $M(i,i) = 0$.

Step 2: For node $a = 1, 2, 3, \ldots, n$ let $M(i,j) = \min\{M(i,j), M(i,a) + M(a,j)\}$. This step checks to see if it is shorter to travel from node i to node j through an intermediary node a.

2.2.2 Dijkstra's Algorithm

Dijkstra's Algorithm is a $O(n^2)$ for finding the shortest distance from a node i to all other nodes in a graph with n nodes. We will extend the algorithm by iterating it over all nodes in our graph thus increasing the complexity to $O(n^3)$. The algorithm is as follows

Step 1: Create a matrix M from a graph G such that $M(i,j)$ corresponds to the weight between node i and node j. If node i and j are not connected in G then $M(i,j) = \infty$; $M(i,i) = 0$.

Step 2: For node $a = 1, 2, 3, \ldots, n$ complete steps 3-6

Step 3: Create an empty set S which will hold nodes that have been visited. Set the current node $z = a$. Let $S = S \cup \{z\}$. Create the vector B where $B(i) = \infty$ for $i \neq a$ and $B(a) = 0$.

Step 4: For $i \notin S$ let $B(i) = \min\{B(i), B(z) + M(z,i)\}$.

Step 5: Let $z = i$, where $i = \min B(i)$ such that $i \notin S$. Let $S = S \cup \{z\}$. If $S = \{1, 2, \ldots, n\}$ then go to Step 6 otherwise go to Step 4.

Step 6: Let M(i,:) = B.

It has been shown [3] that Dijkstra's algorithm can be speed up to $O(n^2 \log(n) + cn)$, where c is the number of starting edges in the graph using Fibonacci Heaps.

2.2.3 Comparison

Dijkstra's algorithm will take less operations than the Floyd-Warshall algorithm but requires more memory and initialization time. For small data sets we will use Floyd-Warshall

2.3 Landmarks

Landmarks work by doing the computationally complex work on a small subset of the data points of size δ, which are either chosen at random or intelligently, and then applying their mapping to the other non-landmark points in such a way as to minimize the error between the normal embedding and the landmark embedding. By using landmarks we reduce the complexity of finding KNN to $O(\delta \log \delta)$ and solving the eigensystem in $O(\delta^2)$. Applying the mappings to the other data points will have much less complexity than finding the KNN and solving the eigensystem for all the points. The tradeoff is of course accuracy in the final embedding.

3 Implementation

To complete the objectives of this project we have implemented LLE and ISOMAP in C++. Both algorithms were modeled after the founding papers and the implementation in the Matlab Dimensionality Toolbox. We have chosen to implement these algorithms in C++ to improve their speed, memory management, and make them more open. The implementation of LLE and ISOMAP were given the same inputs for simplicity; the number of nearest neighbors to use, the dimension to project down to, and the location of the data file.

We originally planned to use the ARPACK sparse eigensolver with LLE as it is a sparse eigenproblem but for other than a few data points the C++ implementation and the Matlab implementation differed by an unacceptable margin. To solve this we chose to implement LLE with a general symmetric eigensolver from LAPACK and concentrate on switching to the ARPACK solver during the code optimization phase in January.

3.1 Libraries

We have made use of the Boost Ublas, ARPACK(eventually), and LAPACK. Boost UBLAS contains containers for matrices and vectors and has access to the Basic Linear Algebra levels I,II,III Fortran code base. This allows use to efficiently do vector-vector products, matrix-vector products, matrix-matrix products, calculate norms, and to solve a system of linear equations (Ax=b). The ARPACK function DSAUPD, using the Implicitly Restarted Arnoldi method implemented in Fortran, will allows us to find the smallest eigenvalues by magnitude and their associated eigenvectors for sparse symmetric matrices. We also made use LAPACK function DGEEV and DYSEV which allowed us to find the eigenvalues and eigenvectors of a general real matrix and a general real symmetric matrix respectively.

As we choose to program in C++ we made use of wrapper functions, dgeev.h, dsaupd.h, and dsyev.h, for ARPACK and LAPACK created by Scot Shaw [13]. These functions simply create a clean C++ type method and convert any data types needed for calling the ARPACK and LAPACK libraries.

We faced many difficulties installing the Intel Math kernel Library so we choose to utilize the open source code bases discussed above. Installation of the codebases

was handled through Macports which optimized them automatically for our current architecture.

3.2 File IO

The data file format we have chosen in comma separated values (csv) with a header line. The header line contains: [number of rows], [number of columns], [dimensionality of data], where all numbers are integers greater than 0. We choose to use a header line because it makes it quicker and simpler to read the data file into our program. Knowing the number of rows and columns also gives us the opportunity to correctly reconstruct an image that is passed in. Other than the first line of the data file each line corresponds to the spectrum for an individual pixel.

When we write out output, the lower dimensional mapping, we create the file filename.[lle or isomap].out. We first write a header line: [number of rows], [number of columns], [dimensionality of data], where all numbers are integers greater than 0. After the first line we write the embedding for each pixel using csv.

3.3 Compiling

The code can be compiled, assuming the libraries are installed correctly, using g++ as such:

```
>>g++ -Wno-write-strings lle.cpp /path/to/libarpack.1.dylib
-I /path/to/boost_1_44_0/ -o lle
```

```
>>g++ -Wno-write-strings isomap.cpp /path/to/liblapack.dylib
-I /path/to/boost_1_44_0/ -o isomap
```

3.4 Execution

The executables can be run as such

```
>>./lle [number of nearest neighbors] [intrinsic dimensionality]
[./path/to/datafile]
```

```
>>./isomap [number of nearest neighbors] [intrinsic dimensionality]
[./path/to/datafile]
```

4 Validating Implementation

To validate out C++ implementation we compared results from C++ to the Matlab Dimension Reduction Toolbox for the Swiss Roll, Broken Swiss Roll, Twin Peaks, and Helix, which are defined in three dimensions but are known to lie on a two dimensional manifold. Due to the topological nature of these structures, they are perfect for dimensionality reduction testing since we can be sure that the data actually lies in the plane.

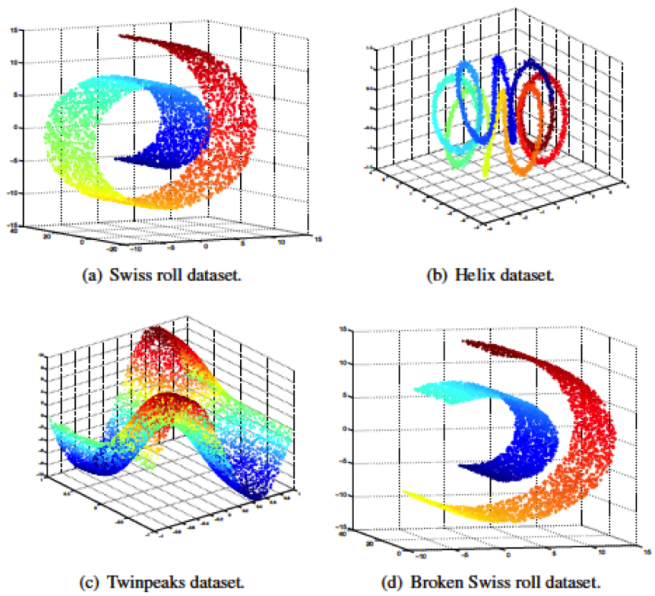


Figure 3: Clockwise from top right: Helix, Broken Swiss Roll, Twin Peaks, Swiss Roll[6].

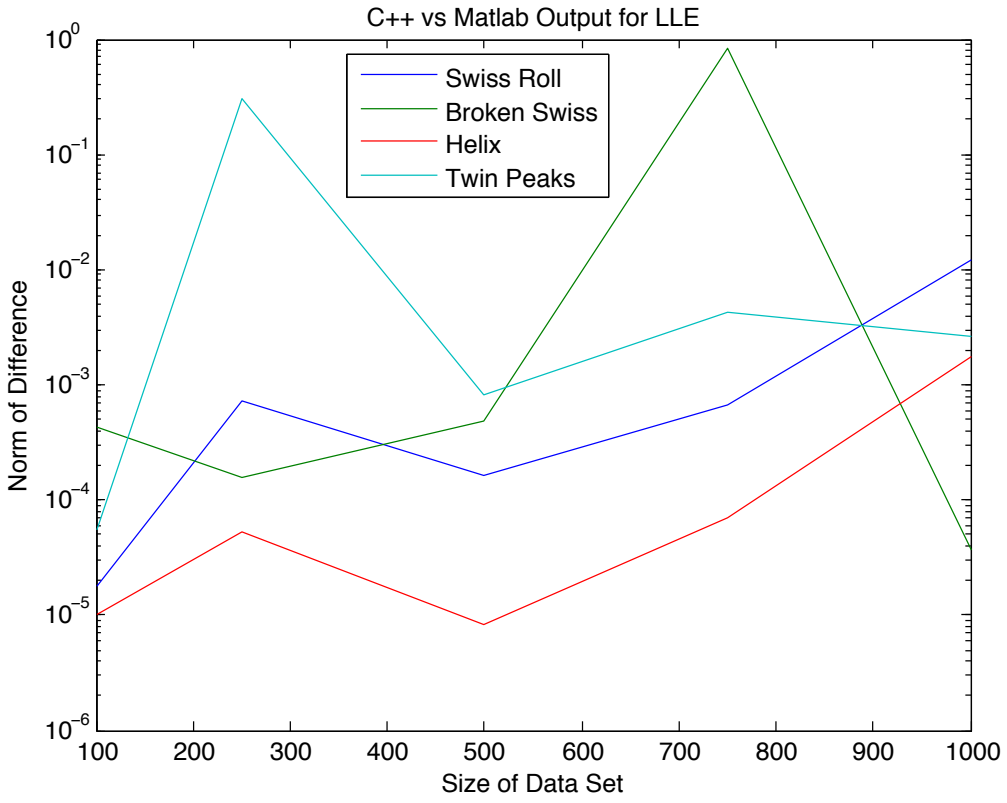
We created data sets using the Matlab Dimensionality Reduction Toolbox function generate_data for the Swiss Roll, Broken Swiss Roll, Helix, and Twin Peaks for 100,250,500,750, and 1000 data points. This data was written to a csv file with a proper header line for use with the C++ code (the Matlab algorithms are also able to read these data files so we be assured that they both have the same data).

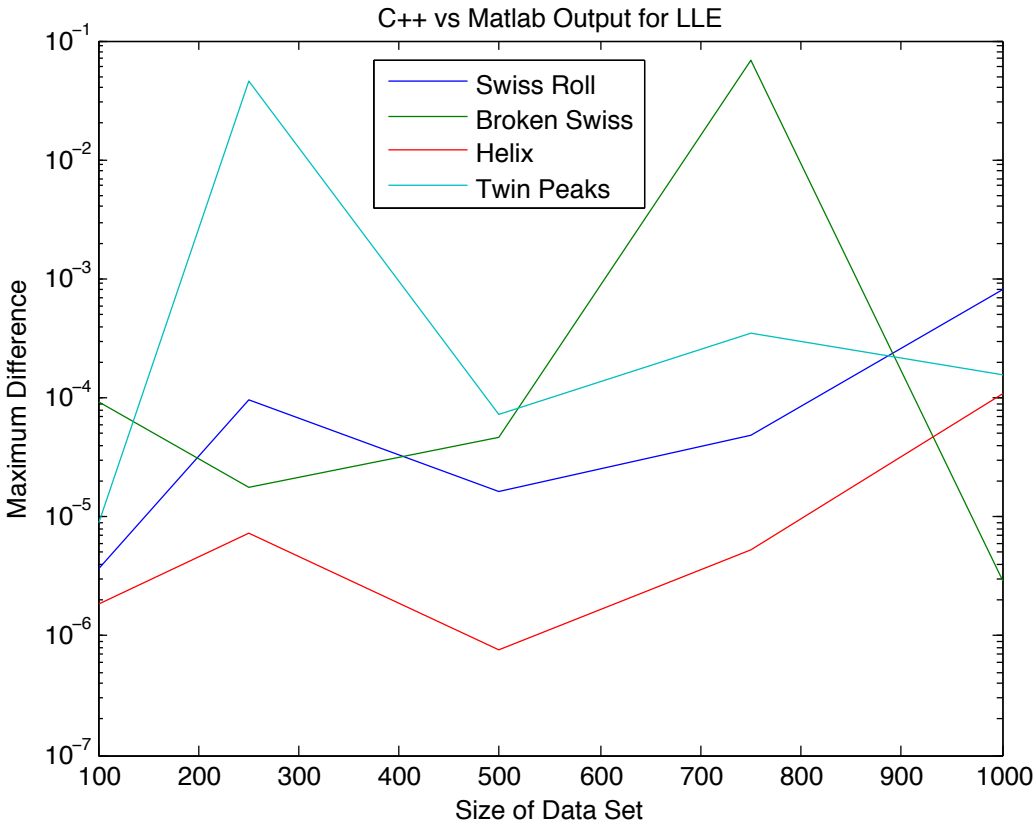
To compare the Matlab embedding, $V = [v_1, v_2]$, with the C++ embedding, $W = [w_1, w_2]$, we used the L2 norm difference, $||w_1 - v_1|| + ||w_2 - v_2||$, and max element difference, $\max\{\max\{|w_1 - v_1|\}, \max\{|w_2 - v_2|\}\}$. Since different eigensolver implementations normalize eigenvectors in slightly different ways we normalize the C++ embedding to the Matlab embedding by finding the ratio of the first elements in each embedding dimension.

Due to the some additional code in the Matlab Dimensionality Reduction Toolbox which removes outlier points (something we are not interested in doing) and the fact that some of the data sets need more nearest neighbors to construct a good embedding we had to vary the number of nearest neighbors supplied to the code slightly as seen in the tables in the subsequent subsections.

4.1 LLE

For LLE we had some difficultly implementing the sparse ARPACK solver as planned and had to instead use a general symmetric eigensolver through LAPACK. Using this different eigensolver produced some additional error between the implementations, as the eigensolvers are using different algorithms. Despite these set backs we were able to show good correspondence between the two implementations, especially for the Swiss Roll and Helix data sets, the sets which are easier to embed. When we consider that our data files had 1E-3 accuracy, the two embeddings corresponded well below this for most of the test data sets. Considering these results we can say that lle.cpp is working properly. We will rerun these validation tests when we are able to implement LLE with a sparse solver and hopefully this will show even better correspondence.

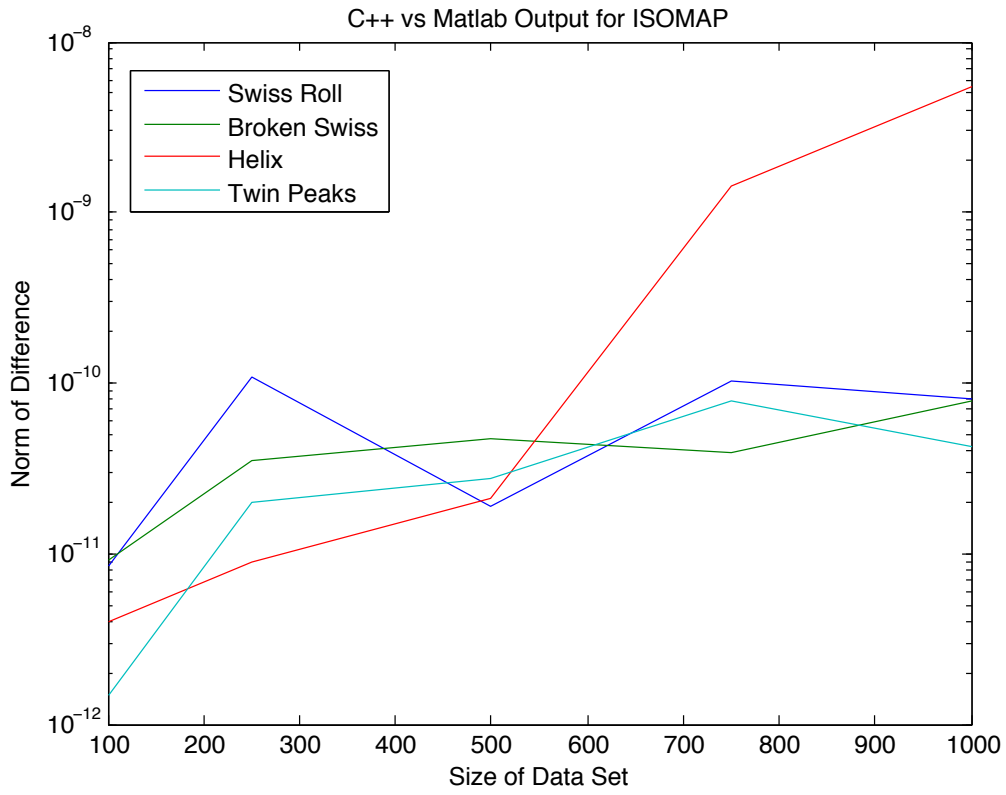

C++ vs Matlab Output for LLE
Swiss Roll
Broken Swiss
Helix
Twin Peaks
Norm of Difference
Size of Data Set
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
100
200
300
400
500
600
700
800
900
1000

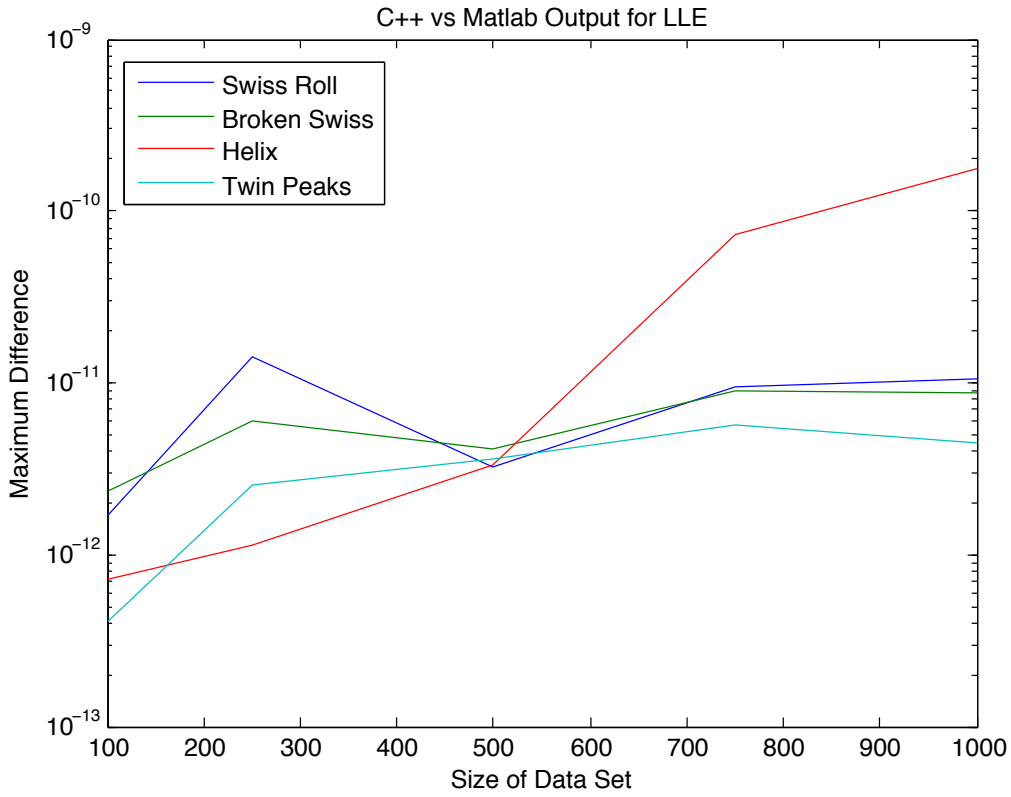

C++ vs Matlab Output for LLE
Swiss Roll
Broken Swiss
Helix
Twin Peaks
Maximum Difference
Size of Data Set
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
100
200
300
400
500
600
700
800
900
1000

ISOMAP				
Data Set	# points	# neighbors	Normed Difference	Max Difference
Swiss Roll	100	12	1.76E-5	3.67E-6
	250	12	7.31E-4	9.50E-5
	500	12	1.62E-4	1.66E-5
	750	12	6.79E-4	4.72E-5
	1000	12	1.21E-2	8.13E-4
Broken Swiss Roll	100	12	4.35E-4	9.39E-5
	250	12	1.58E-4	1.74E-5
	500	14	4.77E-4	4.58E-5
	750	18	8.46E-1	6.74E-2
	1000	29	3.74E-5	2.87E-6
Helix	100	12	1.02E-5	1.82E-6
	250	12	5.29E-5	7.35E-6
	500	12	8.32E-6	7.57E-7
	750	12	7.05E-5	5.22E-6
	1000	12	1.79E-3	1.09E-4
Twin Peaks	100	12	5.47E-5	8.09E-6
	250	12	3.01E-1	4.54E-2
	500	20	8.03E-4	7.36E-5
	750	30	4.23E-3	3.42E-4
	1000	40	2.59E-3	1.57E-4

4.2 ISOMAP

For ISOMAP we were able to show near perfect correspondence between the C++ and established Matlab implementation using the normed difference and max difference metric. The difference between the two embeddings were very low (and close to machine precision, E-16), as seen in the following table and graphs, so we can claim that the isomap.cpp program is working correctly.


C++ vs Matlab Output for ISOMAP
Swiss Roll
Broken Swiss
Helix
Twin Peaks
Norm of Difference
Size of Data Set
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
100
200
300
400
500
600
700
800
900
1000


C++ vs Matlab Output for LLE
Swiss Roll
Broken Swiss
Helix
Twin Peaks
Maximum Difference
Size of Data Set
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
100
200
300
400
500
600
700
800
900
1000

ISOMAP				
Data Set	# points	# neighbors	Normed Difference	Max Difference
Swiss Roll	100	12	8.44E-12	1.71E-12
	250	12	1.09E-10	1.43E-11
	500	12	1.87E-11	3.20E-12
	750	12	1.02E-10	9.43E-12
	1000	12	8.17E-11	1.05E-12
Broken Swiss Roll	100	12	9.27E-12	2.32E-12
	250	12	3.51E-11	6.00E-12
	500	14	4.66E-11	4.15E-12
	750	18	3.89E-11	8.92E-12
	1000	29	7.87E-11	8.77E-12
Helix	100	12	3.98E-12	7.28E-13
	250	12	8.91E-12	1.14E-12
	500	12	2.10E-11	3.35E-12
	750	12	1.41E-9	7.22E-11
	1000	12	5.48E-9	1.76E-10
Twin Peaks	100	12	1.48E-12	4.07E-13
	250	12	2.02E-11	2.54E-12
	500	20	2.74E-11	3.65E-12
	750	30	7.91E-11	5.66E-12
	1000	40	4.19E-11	4.51E-12

4.3 Further Validation

As we were able to match the established output in the Dimension Reduction Toolbox for these sample data sets, at this time, there is no need to perform additional validation steps. When we move on to landmark points next semester (and time permitting make use of some approximate nearest neighbor codes) we will most likely have to use of the addition validation criterion because of the random nature of choosing landmarks points (and approximate neighbors). The addition validation options that we have are to make use of the trustworthiness, $T(k)$ and continuity, $C(k)$[6]: measures that measure how good an embedding is. These measures are defined as:

$$T(k) = 1 - \frac{2}{nk(2n-3k-1)} \sum_{i=1}^{n} \sum_{j \in U_i^{(k)}} (r(i,j) - k)$$

$$C(k) = 1 - \frac{2}{nk(2n-3k-1)} \sum_{i=1}^{n} \sum_{j \in V_i^{(k)}} (\hat{r}(i,j) - k)$$

where $r(i,j)$ is rank of data point $j \in Y_n$ according to pairwise distances between Y_n data points, $U_i^{(k)}$ is the set of points that are among the KNN but not in X_n. Similarly, $\hat{r}(i,j)$ and $V_i^{(k)}$ are defined as the dual of $r(i,j)$ and $U_i^{(k)}$.

We can compare the results of our implementation to the established results found in [6]. The trustworthiness measure ranks the mapping produced on how well it avoids putting points close together in the low dimensional space that are not close in the high dimensional space. The continuity measure, in much the same way as the trustworthiness measure, ranks the mapping on how well it preserves points that are close in the high dimensional space by checking if they are close in the low dimensional space.

5 Improvements to Efficiency

To maximize the usefulness of this codebase we have created we will need to test it on large data sets and this requires making the code more efficient to handle larger amounts of data. We plan to, before moving on to landmark points, to make improvements in the LLE and ISOMAP classes.

In particular we will be looking to improve memory management, make use of sparsity when possible, and switch from using Floyd-Warshall algorithm to Dijkstra's algorithm (when data sets are large).

6 Code Delivered

• lle.cpp - Performs Local Linear Embedding algorithm given an intrinsic dimensionality, a number of nearest neighbors, and a data file
• isomap.cpp - Performs ISOMAP algorithm given an intrinsic dimensionality, a number of nearest neighbors, and a data file
• write.m - Creates the data sets used

• read.m - Reads data sets and embeddings into Matlab
• process_results.m - Automates validation testing and makes graphs
• s####.in, b####.in, t####.in, h####.in - Swiss Roll, Broken Swiss Roll, Twin Peaks, and Helix data sets

7 Future Timeline

•January - Write code to link C++ algorithms with IDL/ENVI and optimize code.
•February and March - Implement landmarks and validate algorithms again with landmarks.
•April - Use algorithms with and without landmarks on hyperspectral classification images.
•May - Prepare final presentation.

7.1 Possible Extensions

Time permitting and after completion of the tasks defined in this proposal document we would like to look at perhaps parallelizing these algorithms by tiling the images among several processors using OpenMP. We may also consider using other nonlinear dimensionality reduction techniques or implementing algorithms to figure out the optimal number of nearest neighbors to select.

8 Milestones

• February 1 - IDL/ENVI can call C++ code and C++ code can return to IDL/ENVI.
• April 1 - Landmark code has passed validation tests.
• May 1 - Results from HSI classification has been obtained.

References

[1] Bachmann, Ainsworth, and Fusina, *Exploting Manifold Geometry in Hyperspectral Imagery*, IEEE Transactions of Geoscience and Remote Sensing **43** (2005), 441–454.

[2] L. Bernstein, S. Adler-Golden, R. Sundberg, R. Levine, T. Perkins, A Berk, A. Ratkowski, G. Felde, and M. Hoke, *A New Method for Atmospheric Correction and Aerosol Optical Property Retrieval for VIS-SWIR Multi- and Hyperspectral Imaging Sensors: QUAC (QUick Atmospheric Correction)*, 2006.

[3] M. Fredman and R. Tarjan, *Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms*, Journal of the ACM **34** (1987).

[4] G. Gutin J. Bang-Jensen, *Digraphs: Theory, algorithms and applications*, Springer, 2010.

[5] C. Lelong, P. Pinet, and H. Poilve, *Hyperspectral Imaging and Stress Mapping in Agriculture: A Case Study on Wheat in Beauce (France)*, Remote Sensing of Environment **66** (1998), no. 2, 179 – 191.

[6] Maatan, Postma, and Herik, *Dimensionality Reduction: A Comparative Review*, (2008).

[7] D. Manolakis, D. Marden, and G. Shaw, *Hyperspectral Image Processing for Automatic Target Detection Applications*, Lincoln Laboratory Journal **14** (2003), 79–116.

[8] R. Mayer, R. Priest, C. Steliman, G. Hazel, and A. Schaum, *Detection of Camouaged Targets in Cluttered Backgrounds Using Fusion of Near Simultaneous Spectral and Polarimetric Imaging*, Tech. report, Naval Research Lab, 2000.

[9] S. Roweis and L. Saul, *An Introduction of Local Linear Embedding*, Unpublished manuscript, 2001.

[10] F. Sabins, *Remote Sensing for Mineral Exploration*, Ore Geology Reviews **14** (1999), no. 3-4, 157 – 183.

[11] L. Saul and S. Roweis, *Reduction by Locally Linear Embedding*, Science **209** (2000), 2323–2327.

[12] J. Schott, *Remote Sensing: The Image Chain Approach*, Oxford University Press, New York, 1997.

[13] S. Shaw, *LAPACK/ARPACK*, `http://www-heller.harvard.edu/people/shaw/programs/lapack.html`.

[14] M. Smith, S. Ollinger, M. Martin, J. Aber, and R. Hallettand C. Goodale, *Direct Estimation of Aboveground Forest Productivity Through Hyperspectral Remote Sensing of Canopy Nitrogren*, Ecological Applications **12** (2002), no. 5, 1286–1302.

[15] Tenenbaum, Silva, and Langford, *A Global Geometric Framework for Nonlinear Dimensionality Reduction*, Science **209** (2000), 2319–2323.