May 2016

Inverse Methods for Manifold Learning

Kathleen Rose Kay
Worcester Polytechnic Institute

Inverse Methods for Manifold Learning

A Major Qualifying Project

submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree in Bachelor of Science by

Kathleen Kay

5 May 2016

Approved:

Professor Randy Paffenroth

Abstract

In this MQP project, our focus is on inverse methods for non-linear manifold learning. Basically, methods for taking high-dimensional data and non-linearly projecting it to low-dimensions are well known. However, methods for going the other direction (from low-dimensional data to high-dimensional data) are less well studied.

Such inverse methods are important as they are closely connected with modern ideas in Deep Learning. Deep Learning is currently used in many applications (such as voice and image recognition) and the results of this MQP support new work in non-linear deep auto-encoders.

Here, we examine the use of linear and nonlinear methods of dimension reduction and study the particulars of the inverse mapping (i.e. type of radial basis function kernel, and other parameters dealing with the kernel matrix distances). Furthermore, we study which combinations of parameters are more effective in the case where there are restrictions on the dimension of our projections. We explore a variety of different projection algorithms including Principal Component Analysis, Isomap, and Local Tangent Space Alignment. Finally, the effectiveness of our approaches will be demonstrated on the image recognition problem of classifying handwritten digits.

Contents

1 Introduction **1**

2 Manifold Learning **3**
2.1 Linear Methods . . . 3
2.1.1 Principal Component Analysis . . . 3
2.1.2 Random Sparse Matrix . . . 3
2.2 Nonlinear Methods . . . 3
2.2.1 Isomap . . . 3
2.2.2 Local Tangent Space Alignment . . . 4
2.2.3 Spectral Embedding . . . 5

3 Inverting Nonlinear Dimensionality Reduction with Scale-Free Radial Basis Function Interpolation **7**
3.1 How It Works . . . 7
3.1.1 What They are Doing . . . 7
3.1.2 Restrictions and Requirements . . . 7
3.1.3 Pertinent Equations . . . 8
3.1.4 Radial Basis Function . . . 9
3.1.5 Matrices . . . 9
3.1.6 Recovery . . . 10

4 Numerical Examples **12**
4.0.7 Equations . . . 12
4.1 About the Dataset Being Used . . . 12
4.1.1 Training and Testing Images . . . 13
4.1.2 Using Different RBFs . . . 13
4.2 Linear Methods . . . 14
4.2.1 PCA . . . 14
4.2.2 Random Sparse Matrix . . . 15
4.3 Nonlinear Methods . . . 19
4.3.1 Isomap . . . 19
4.3.2 Local Tangent Space Alignment . . . 21
4.3.3 Spectral Embedding . . . 24
4.4 Exploring the Results . . . 26
4.4.1 Various Dimensions for Projecting and Their Effects . . . 26

5 Conclusions **29**

Appendices **30**

A Python Code **30**
A.1 The Methods . . . 30

B Extensions, Notes, and Ideas **31**

1 Introduction

In this report, we will attempt to accomplish several tasks. First, we will go into detail about particular methods, the algorithm used, some numerical examples and results, and some applications and connections. With the understanding that not everyone is well versed in the topics at hand, there will be multiple levels of explanations. There will be a particular analogy running throughout, geared for those who are not as knowledgeable. It will show up in the more in depth explanations as well.

Is there a method to this madness? Yes. Projecting images from high dimensional space to low dimensional space is used with data compression, ease of visualization, and providing better features for machine learning.

In Figure 1, we see two plots – one with hands and other with faces. Within these images are a multitude of points. These points are low dimensional representations of either hands or faces. The circled points correspond with the adjacent image. Much is known about projecting high dimensional data, such as images, to lower dimensional representations. The methods featured and to be explained later are Principal Component Analysis[1], Random Sparse Matrix[1], Isomap[1], Local Tangent Space Alignment[1], and Spectral Embedding[1].

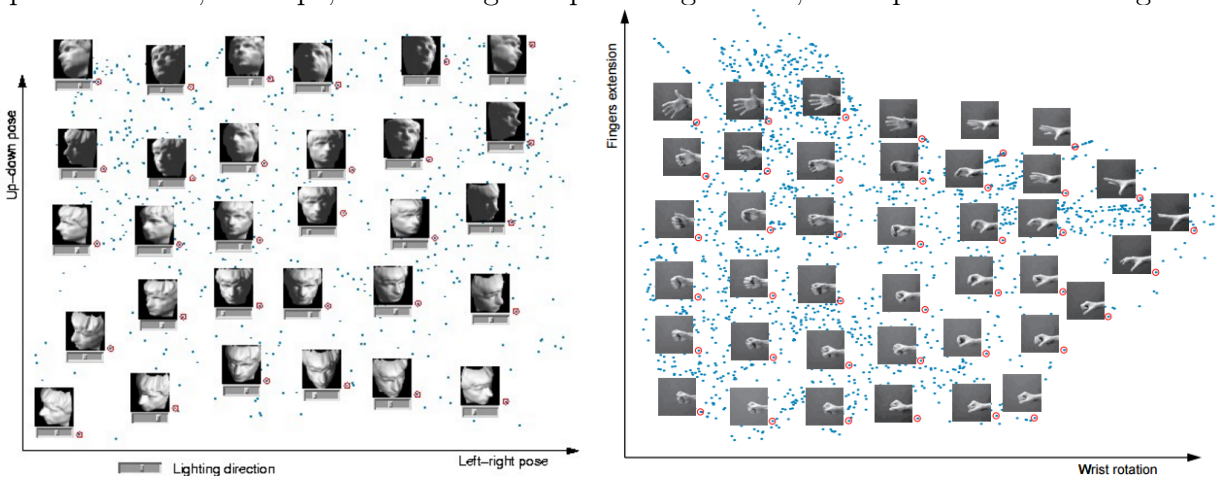


Figure 1: On the left we have images of hands and to the right, images of faces.[2] The images are located in high dimensional space. The points are low dimensional representations of the images.

In Figure 2, the first Einstein[2] is taken and projected to a lower dimension represented by the middle image – we go from great detail to less detail. This middle picture, an essence of Einstein, is used to recover the last image. The essence of Einstein is in a smaller dimension than our original and recovered images. Essences of images will be used to signify points in the projection space.

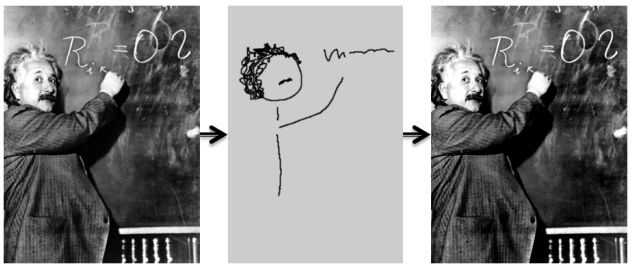


Figure 2: This is for our analogy. The first and last images differ slightly in color contrast. This represents the error in recovery.

[1] See Sections 2.1.1, 2.1.2, 2.2.1, 2.2.2, and 2.2.3, respectively, for how they work.

[2] Image from [7]

Our New Contribution: We implemented various projections to lower dimensions and we looked at the interactions between these and the kernel function[3] of use for going from low dimensions back to high dimensions. We have an in-depth study of how varying the projection and the kernel affects the recoverability of images. More-so, we studied how these interactions affect the predictability of unknown or not yet existing images based purely off of points in the projection space. For example, taking what we think is an essence of Einstein and predicting a never-before-seen Einstein image. Our novel contribution has been to study this interaction, for our reference[4] deals with only image recovery. We predict an essence's higher-dimensional image. This does not mean we are not reconstructing images – we focus on known image reconstruction for cross validation.[5]

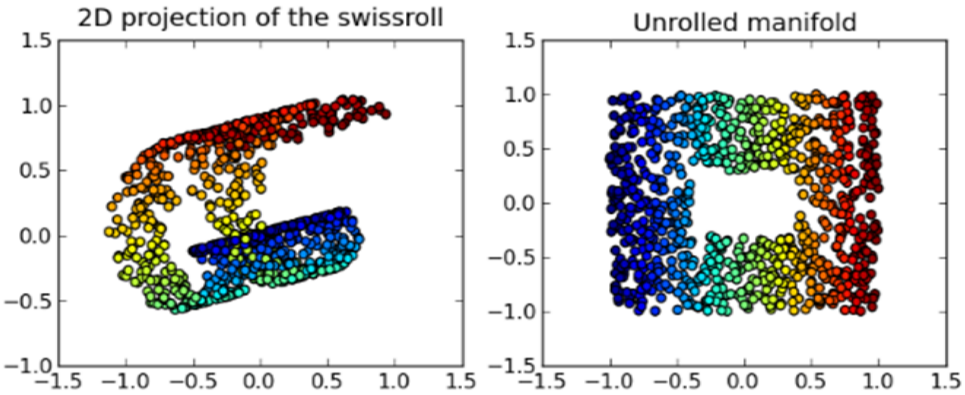


Figure 3: On the left we see a manifold in high dimensions and on the right is the same manifold flattened out.

More specifically, we are looking at data on manifolds. These points are projected into a lower dimension, say 2 dimensions, and then we raise them up back to their starting dimension. Figure 3 shows two plots. The left has points in high dimensional space who live on a manifold. The second image shows the same manifold except it has been unrolled. Figure 4 shows a selection of handwritten digits projected into 2-dimensional space. We will be featuring the same dataset in with our results. But now mind you, no method is perfect due to irreducible error but we explore the method detailed in the paper[4] by Nathan Monnig, Bengt Fornberg, and Francois Meyer.

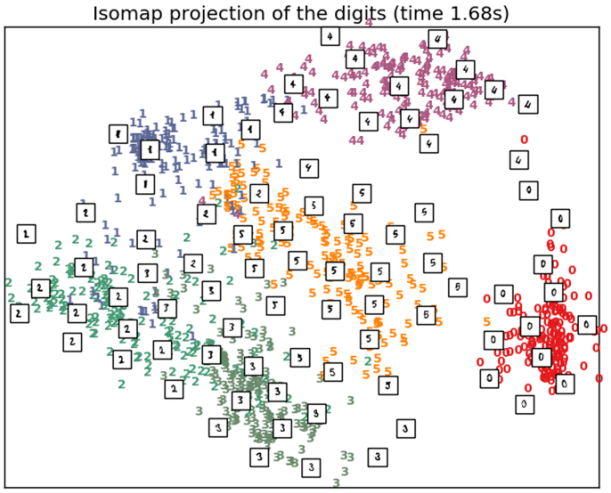


Figure 4: This is a plot of a two dimensional Isomap projection from sklearn. This uses the digits dataset of handwritten digits. We will see this dataset used in 4. Each of the numbers are number and color coded.[3]

[3]Deals with distances but will be defined in Section 3.1.4.

[4]See Section 3 or [6]

[5]Reconstructing an image would be like putting a puzzle back together while having the picture in front of you. Predicting images would be similar to putting the puzzle back together but the pieces might not be accurate or there are multiple puzzles' pieces mixed together, and you do not know what it could be.

2 Manifold Learning

2.1 Linear Methods

2.1.1 Principal Component Analysis

Our first linear method is Principal Component Analysis, or PCA. We use principal components, or linear combinations of our variables. A lot about the variance can be explained with these. We write this linear combination as[6]

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p \tag{1}$$

We rewrite this as

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{p1}x_{ip} \tag{2}$$

with the following constraints

$$\max_{\phi_{11},...,\phi_{p1}} \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^2 \tag{3}$$

$$\text{subject to} \sum_{j=1}^{p}\phi_{j1}^2 = 1 \tag{4}$$

We also need this, (5), for (3) to work.

$$\frac{1}{n}\sum_{i=1}^{n}x_{ij} = 0 \tag{5}$$

2.1.2 Random Sparse Matrix

We use the Random Sparse Matrix (RSM) from sklearn's random projection module.[1] This module can use a Gaussian random matrix and a sparse random matrix.

The matrix is created as follows:

$$\begin{cases} -\sqrt{\frac{s}{n_{components}}} & \\ 0 & \text{with probability} \\ +\sqrt{\frac{s}{n_{components}}} & \end{cases} \begin{matrix} \frac{1}{2s} \\ \text{1-1/s} \\ \frac{1}{2s} \end{matrix} \tag{6}$$

where $s = 1/\text{density}$ and $n_{components}$ is the projected subspace size.

2.2 Nonlinear Methods

2.2.1 Isomap

Isometric Feature Mapping, or Isomap, is a low-dimensional embedding method. It works by using each data point's neighbors to estimate the geometry of the data manifold they lie

[6] See [4, p. 375-376]

on.

The following is the Isomap algorithm as described in [5, p. 107]).

1. Build a graph with either the K-rule or the ϵ-rule.
2. Weight the graph by labeling each edge with its Euclidean length.
3. Compute all pairwise graph distances with Dijkstra's algorithm, square them, and store them in matrix $\mathbf{D}$.
4. Convert the matrix of distances $\mathbf{D}$ into a Gram matrix $\mathbf{S}$ by double centering.
5. Once the Gram matrix is known, compute its spectral decomposition $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$.
6. A P-dimensional representation of $\mathbf{Y}$ is obtained by computing the product $\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{1/2}\mathbf{U}^T$.

2.2.2 Local Tangent Space Alignment

Local Tangent Space Alignment (LTSA) is best described as follows:

> Based on a set of unorganized data points sampled with noise from a parameterized manifold, the local geometry of the manifold is learned by constructing an approximation for the tangent space at each data point, and those tangent spaces are then aligned to give the global coordinates of the data points with respect to the underlying manifold.[9]

The algorithm can be implemented as described by [8]:

1. [Extracting local information.] For each $i = 1, \ldots, N$,
   (a) Determine k nearest neighbors x_{i_j} of x_i, $j = 1, \ldots, k$.
   (b) Compute the d largest eigenvectors $g_1, \ldots, g_d$ of the correlation matrix $(X_i - \vec{x}_i e^T)^T(X_i - \vec{x}_i e^T)^T$, and set $G_i = [e/\sqrt{k}, g_1, \ldots, g_d]$.
2. [Constructing the alignment matrix.] Form the the alignment matrix ϕ by locally summation if a direct eigen-solver will be used. Otherwise implement a routine that computes matrix-vector multiplication Bu for an arbitrary vector u.
3. [Computing global coordinates.] Compute the $d+1$ smallest eigenvectors of ϕ and pick up the eigenvector matrix $[u_2, \ldots, u_{d+1}]$ corresponding to the 2nd to $d+1$st smallest eigenvalues, and set $T = [u_2, \ldots, u_{d+1}]^T$.

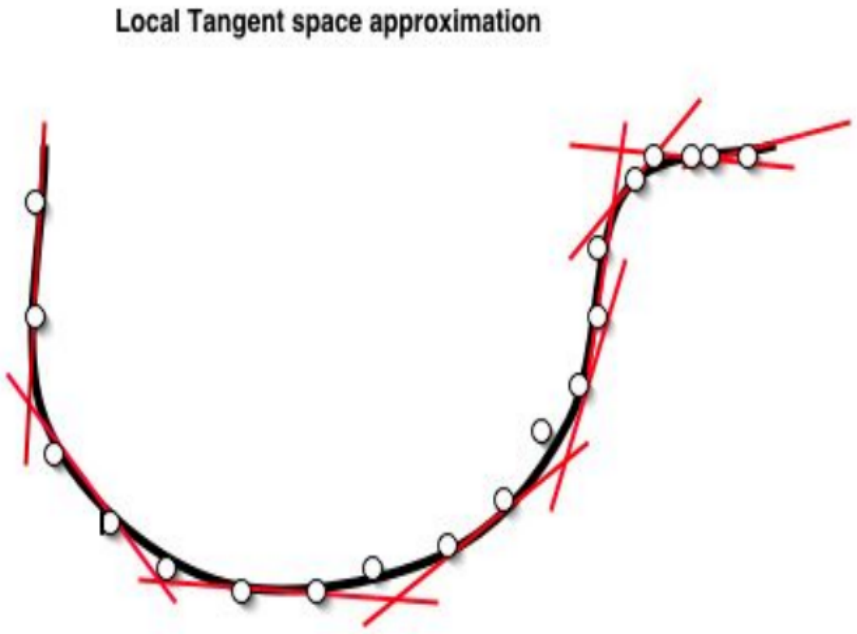


Figure 5: An illustration of local tangent space approximation from [8].

2.2.3 Spectral Embedding

Spectral Embedding is also known as Laplacian Eigenmaps.

We use the following equations[5, p. 160]:

$$w_{i,j} = \exp\left(-\frac{\|\mathbf{y}(i) - \mathbf{y}(j)\|_2^2}{2T^2}\right) \tag{7}$$

$$\mathbf{L} = \mathbf{W} - \mathbf{D} \tag{8}$$

where diagonal matrix $\mathbf{D}$ has entries $d_{i,i} = \sum_{j=1}^{N} w_{i,j}$.

The following procedure is from [5, p. 162]

1. If data consist of pairwise distances, then skip step 2 and go directly to step 3.
2. If data consist of vectors, then compute all pairwise distances.
3. Determine either K-ary neighborhoods or ϵ-ball neighborhoods.
4. Build the corresponding graph and its adjacency matrix $\mathbf{A}$.
5. Apply the heat kernel (or another one) to adjacent data points, and build matrix $\mathbf{W}$ as in Equation (7).
6. Sum all columns of $\mathbf{W}$ in order to build the diagonal matrix $\mathbf{D}$, which consists of the rowwise sums of $\mathbf{W}$.
7. Compute $\mathbf{L}$, the Laplacian of matrix $\mathbf{W}$: $\mathbf{L} = \mathbf{W} - \mathbf{D}$.

8. Normalize the Laplacian matrix: $\mathbf{L}' = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$.

9. Compute the EVD of the normalized Laplacian: $\mathbf{L}' = \mathbf{U}\mathbf{\Gamma}\mathbf{U}^T$.

10. A low-dimensional embedding is finally obtained by multiplying eigenvectors by $\mathbf{D}^{1/2}$, transposing them, and keeping those associated with the P smallest eigenvalues, except the last one, which is zero.

3 Inverting Nonlinear Dimensionality Reduction with Scale-Free Radial Basis Function Interpolation

Note: The $\mathbf{x}^{(i)}$'s will be often referred to as images, i.e. known Einstein images. The $\mathbf{y}^{(i)}$'s then correspond to essences.

What is the purpose of the paper?[7]

The authors of the paper [6] suggest an algorithm for recovering known images using a radial basis function kernel. It uses a set of n $\mathbf{x}^{(i)}$'s of dimension D and their corresponding $\mathbf{y}^{(i)}$'s of dimension d ($D > d$) and uses them to set up the system for reconstructing images from points (vectors, our essences) in d-dimensional space.

Another way of thinking of this is with a puzzle. Say you have a 1000-piece puzzle of Albert Einstein. Take some of the pieces and put them in a blender, the nonlinear transform, and you have shreds of Einstein pieces. This would be our $\mathbf{y}^{(i)}$'s. We see an essence of Einstein but it's not all there. We use these shreds to recover the original image. This is where the algorithm does its magic.

3.1 How It Works

3.1.1 What They are Doing

The method depicted uses the radial basis function (RBF) kernel to create a matrix of distances,[8] K. The distances are between the projected $\mathbf{y}^{(i)}$'s. Using K and our $\{\mathbf{x}^{(i)}\}_{i=1}^{n}$, we find the A matrix[9] which solves the system. We can use this A matrix to solve for new images given any $\mathbf{y}$, regardless of random or chosen entries[10]. We find the distances between these $\mathbf{y}$'s and our original $\mathbf{y}^{(i)}$'s using the same method. This becomes our new K matrix. By multiplying K and A, we get our new images. One thing to note is that the first column of the X matrix is $x_1^{(i)}$, these are the first components of the n images. The second column is the second, and so on and so forth until you have D columns. This corresponds to the dimension of $\mathbf{x}$.

The new set of $\mathbf{y}$'s can have any number of $\mathbf{y}$'s as long as the dimension (of $\mathbf{y}$) makes sense.

3.1.2 Restrictions and Requirements

To initially set this up, the $\mathbf{x}^{(i)}$'s need to be of a higher dimension than the $\mathbf{y}^{(i)}$'s. Not only this, but for our initial $\mathbf{x}$ and $\mathbf{y}$ sets, we do not want to have any $\mathbf{y}^{(i)}$'s the same. This will cause the matrix K to be singular and we will be unable to solve for A. ($\mathbf{y}^{(i)} \neq \mathbf{y}^{(j)}$, $\forall i \neq j$)

[7] The paper is the title of this section, *Inverting Nonlinear Dimensionality Reduction with Scale-Free Radial Basis Function Interpolation*, see [6]

[8] This is being used loosely. See Section 3.1.4 for an explanation

[9] The A in the A matrix is read as "Alpha" for *alpha* is used in our equations. See Section 3.1.3 or Equation (12), among others.

[10] Random entries relate more to essences of Einsteins who would be predicted to never-before-seen images. The chosen entries would be more for cross validation or for essences for which we know the recovered image.

When we are bringing the essences to images, we require the essences to be in the same dimensional space as we were projecting into to start with.[11]

We do not have the same restriction on the $\mathbf{y}^{(i)}$'s as in the initial set up when we are recovering images. Having some $\mathbf{y}^{(i)}$'s and $\mathbf{y}^{(j)}$'s equal will not cause any issues. Multiple essences will just be recovered as the same image[12].

3.1.3 Pertinent Equations

Note: Equations from [6].
With $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\} \subset \mathbb{R}^D$ on a smooth manifold $\mathcal{M} \subset \mathbb{R}^D$, we have a mapping

$$\mathbf{\Phi} : \mathcal{M} \subset \mathbb{R}^D \longrightarrow \mathbb{R}^d \tag{9}$$

$$\mathbf{x}^{(i)} \longmapsto \mathbf{y}^{(i)} = \mathbf{\Phi}(\mathbf{x}^{(i)}),\ i = 1, ..., n \tag{10}$$

The $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$ are the original high dimensional images.
$\mathbf{\Phi}^\dagger$ is the inverse, where

$$\mathbf{\Phi}^\dagger : \mathbf{\Phi}(\mathcal{M}) \longrightarrow \mathbb{R}^D, \text{ with } \mathbf{\Phi}^\dagger(\mathbf{y}^{(i)}) = \mathbf{x}^{(i)} \tag{11}$$

The following is what we use in practice:

$$\text{for all } \mathbf{y} \in \mathbf{\Phi}(\mathcal{M}),\ \phi^\dagger(\mathbf{y}) = \sum_{j=1}^{n} \alpha_i^{(j)} k(\mathbf{y}, \mathbf{y}^{(j)}) \tag{12}$$

where k is the radial basis function kernel with $k(\mathbf{z}, \mathbf{w}) = g(\|\mathbf{z} - \mathbf{w}\|)$,[13] and where the α's are the weights.

In matrix form, we first have

$$\begin{bmatrix} k(\mathbf{y}^{(1)}, \mathbf{y}^{(1)}) & \dots & k(\mathbf{y}^{(1)}, \mathbf{y}^{(n)}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{y}^{(n)}, \mathbf{y}^{(1)}) & \dots & k(\mathbf{y}^{(n)}, \mathbf{y}^{(n)}) \end{bmatrix} \begin{bmatrix} \alpha_1^{(1)} \\ \vdots \\ \alpha_1^{(n)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ \vdots \\ x_1^{(n)} \end{bmatrix} \tag{13}$$

This is for the first component in each image. For all of the components we use

$$\begin{bmatrix} k(\mathbf{y}^{(1)}, \mathbf{y}^{(1)}) & \dots & k(\mathbf{y}^{(1)}, \mathbf{y}^{(n)}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{y}^{(n)}, \mathbf{y}^{(1)}) & \dots & k(\mathbf{y}^{(n)}, \mathbf{y}^{(n)}) \end{bmatrix} \begin{bmatrix} \alpha_1^{(1)} & & \alpha_D^{(1)} \\ \vdots & \dots & \vdots \\ \alpha_1^{(n)} & & \alpha_D^{(n)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & & x_D^{(1)} \\ \vdots & \dots & \vdots \\ x_1^{(n)} & & x_D^{(n)} \end{bmatrix} \tag{14}$$

The first matrix, denoted K, contains the kernel distances between low dimensional images and it holds the main input of the algorithm. The second matrix,A contains our parameters the we use to fit the high dimensional images in X.[14]

We write the following for predicting images from essences.

$$\mathbf{\Phi}(\mathbf{y})^T = k(\mathbf{y}, \cdot)^T A = k(\mathbf{y}, \cdot)^T K^{-1} X = \hat{\mathbf{x}} \tag{15}$$

$$\text{where } k(\mathbf{y}, \cdot) = \begin{bmatrix} k(\mathbf{y}, \mathbf{y}^{(1)}) & \dots & k(\mathbf{y}, \mathbf{y}^{(n)}) \end{bmatrix}^T \tag{16}$$

[11] We can only recover to a certain dimension – the one we started with, where the $\mathbf{x}^{(i)}$'s reside.

[12] Because we are in a lower dimension, there is the possibility of having some of the images sharing essences. This is an interesting situation to note and be aware of.

[13] See Section 3.1.4.

[14] See Section 3.1.5 for more on these matrices.

3.1.4 Radial Basis Function

There are six types of radial basis function kernels that we will be focusing on. They are Gaussian, multiquadratic, inverse quadratic, inverse multiquadratic, and polyharmonic splines, and thin plate spline. Despite suggesting the use of the polyharmonic splines in the paper, they show images of Gaussian, cubic, and Sheppard[15]. The code implemented has options to use any of the six RBFs mentioned. The chosen RBF does have an effect on the images recovered.

The six radial basis function kernels are as follows:

$$k(\mathbf{z}, \mathbf{w}) = g(\|\mathbf{z} - \mathbf{w}\|)$$

$$\text{Gaussian: } g(r) = e^{(-(\epsilon r)^2)} \tag{17}$$

$$\text{Multiquadratic: } g(r) = \sqrt{1 + (\epsilon r)^2} \tag{18}$$

$$\text{Inverse quadratic: } g(r) = \frac{1}{1 + (\epsilon r)^2} \tag{19}$$

$$\text{Inverse multiquadratic: } g(r) = \frac{1}{\sqrt{1 + (\epsilon r)^2}} \tag{20}$$

$$\text{Polyharmonic splines: } g(r) = r^k,\ k = 1, 3, 5, ... \tag{21}$$

$$g(r) = r^k \ln(r),\ k = 2, 4, 6, ... \tag{22}$$

$$\text{Thin plate spline: } g(r) = r^2 \ln(r) \tag{23}$$

where ϵ is a restraint. We have code for thin plate splines but we have to make sure that the value for when $r = 0$ is zero. We have to add this condition because we cannot take the natural logarithm of 0.

The radial basis function kernel is a function which uses the distances between points. For simplicity, we shall refer to the outputs of the RBF kernels as distances.

3.1.5 Matrices

To set up the method, we use the RBF kernel to make a matrix of distances. This matrix and our known images are used to make the A matrix required to recover unknown images from the projected space. The A matrix acts as weights for each of the components.

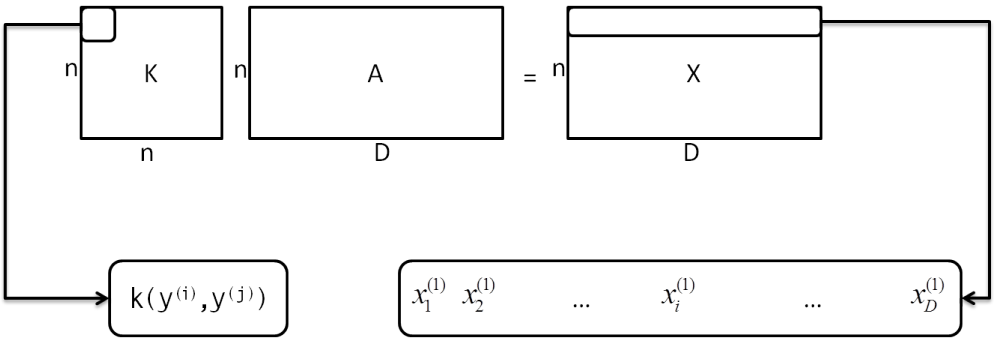


Figure 6: Dimensions of the three matrices and components of K and X.

[15]This is an RBF type but we do not use it in our paper. Please see [6] for information on it.

In terms of our Einstein picture, K contains essences of Einstein and X is our Einsteins. K is an n by n matrix whose entries are filled with distances obtained from the kernel function. The X matrix is n by D and each row is an image. The A matrix, which is n by D, is what we use to recover new images from the projected space. It does not depend linearly on any of the low dimensional projections whereas K does depend nonlinearly. But the $\mathbf{y}$'s never directly appear.

3.1.6 Recovery

Through the use of our A matrix, we can recover known images from their low dimensional representation and we can predict images from the same low dimensional space.

The i, j entries in the new K matrix (K_{new}) are the respective distances between points using the kernel function, same thing as before but with a slight difference. Here, the $\mathbf{y}^{(j)}$'s are from our known set, so essences of Einsteins that we know to be Einsteins. The $\mathbf{y}^{(i)}$'s are what we believe to be essences of Einsteins but we do not actually know if they truly are Einsteins. These are what we are predicting/raising up. (See Figure 7.)

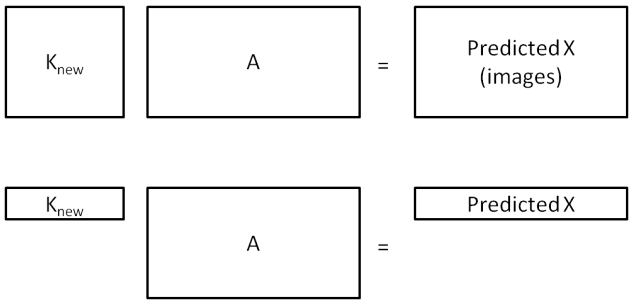


Figure 7: Possible recovery/prediction set ups.

Each A matrix that is created is used on similar image data. We would not hope to predict a Newton image if we were only looking at Einstein images. We would need to train on what we hope to predict or recover.

After calculating the new kernel matrix, all one needs to do is multiply it with the solved A matrix to reconstruct the images from the low dimensional projection. There are no (major) restrictions on the number of images we can recover[16]. Our new K matrix does not need to be square. It can have 10, 20, or even just one row. The amount of rows in K_{new} matches with the number of essences we will be predicting to images. We should note that predicting full pictures of humans is quite difficult so here we focus on images of handwritten digits.

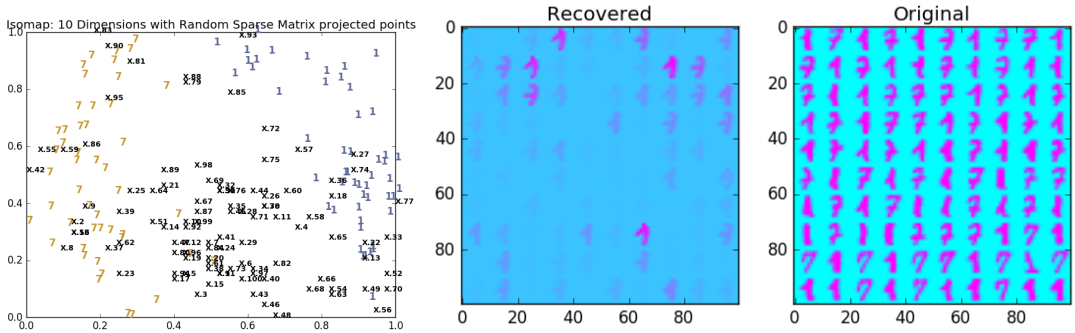


Figure 8: We use the A matrix from an Isomap projection to predict the images associated with a Random Sparse Matrix projection.

[16]Yes, you can predict as many as you want from the A matrix, but having a larger initial set would yield better results.

There are many linear and nonlinear methods that can be used to project the images down into essence. Using an A from one method to predict images from another does not guarantee the same results. In Figure 8, we show that using essences from a RSM projection with the A matrix from Isomap training does not recover the original numbers. By why is this? The points are defined differently.

4 Numerical Examples

In this section, we will be looking at the performance of the algorithm when used with different projection types. We will be using the types from Section 2. In Section 4.1.2, we will introduce how varying the RBF kernel function affects the recoverability and predictability.[17] As we will find, and in agreement with the paper, the Gaussian RBF kernel works fine but it is not the best one.

4.0.7 Equations

Defining error as

$$\text{Error} = \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \tag{24}$$

where $\mathbf{x}$ is the true value and $\hat{\mathbf{x}}$ is the predicted value.

The error is between the true image and the recovered image, so the error pixel by pixel. The error in (24) is deceiving, high error does not necessarily mean our recovery was poor. We might still recover the same type of image but it is not exactly as it was before. We shall see later that a digit image can be recovered to the show the same digit but the image may not look the same. Basically, we recognize or identify the original and recovered images as displaying the same number.

4.1 About the Dataset Being Used

We explore the use of the digits dataset from sklearn. This is a set of handwritten digits spanning 0 through 9. Each digit image is in $\mathbb{R}^{8\times 8}$ but we use the digits as a column vector, so they are in $\mathbb{R}^{64}$. Each of the digits has a number corresponding to what it is. As a side note, when the images of the digits are shown, they have a one pixel border, which is why recovered image groups have intervals of 10.

Here is the code to obtain the digits dataset:

```
from sklearn import datasets

digits = datasets.load_digits(n_class=6)
X = digits.data
y = digits.target
n_samples, n_features = X.shape
n_neighbors = 30
```

With the code we have currently implemented, we can choose any of the RBF kernels discussed as well as a few other details. The main function call is as follows:

```
algFull(x,y_old,y_new,L,
     x_new=[],ipc=1,ipr=1,nipr=1,nipc=1,rho=3,eps=1,nt=2,maptype = '',plots = 1)
```

[17] This will be seen more in each methods' section.

`x`	Our training set
`y_old`	The corresponding projections to our training set
`y_new`	The projections (essences) to be predicted
`L`	The type of RBF kernel that will be used
`x_new`	The images that will be recovered, if known; used for showing recovery error
`ipc, ipr`	The number of images per row or column for the predicted set
`nipr, nipc`	The number of images per row or column for the training set
`rho`	Used for polyharmonic splines; default set to cubic
`eps`	ϵ from the RBF kernel equations
`nt`	The type of norm taken to measure distances in the kernel; by default L_2
`maptype`	Choice of plot coloring; default spectral, options for cool and binary
`plots`	Choice of displaying the plots created; by default they will show

Our main focus for what the code returns is the predicted or recovered images.

Now that we have been introduced to the dataset, we will show what an image will look like at different stages. A one in the original space and in $\mathbb{R}^{5\times5}$ is seen in Figure 9. A Random Sparse Matrix was used to project the image into a lower space. The essence of a one is truly that, it does not resemble a one.

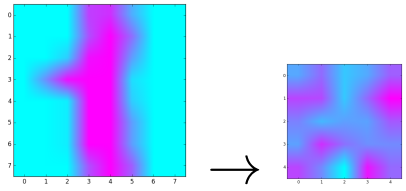
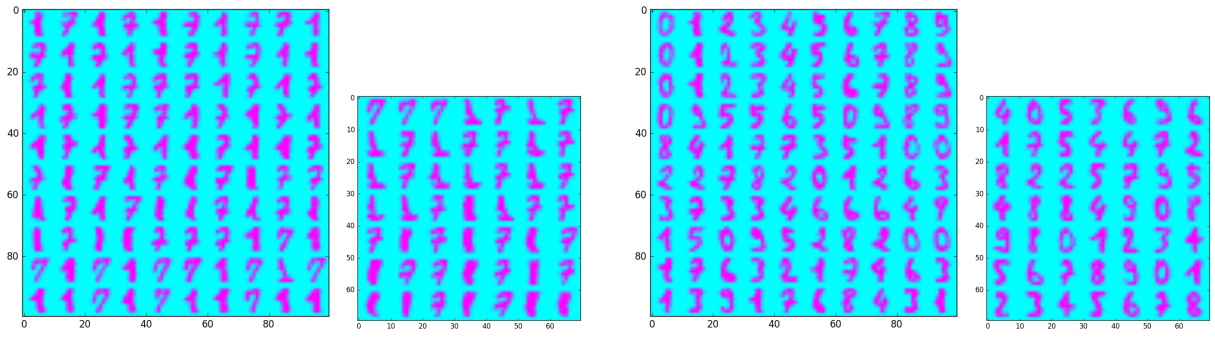


Figure 9: An 8 by 8 image of a one to a 5 by 5 essence of a one.

4.1.1 Training and Testing Images



(a) Ones and sevens

(b) Zero through nine

Figure 10: Training sets and testing sets

4.1.2 Using Different RBFs

An interesting finding is that the radial basis function kernel being used does make a difference in our accuracy. We shall see the affects on these in our cross validation test to figure out which is the best to use on prediction new images. After cross validation on the type of kernel, tests were done to see which dimension would be best to do predictions on. These are seen in Section 4.4.1.

4.2 Linear Methods

4.2.1 PCA

Cross Validation

Here, the A matrix is specialized for ones and sevens. This is comparable to working with essences and pictures of Einsteins and Newtons.

In Figure 11, we see what different kernels do to the recoverability of the handwritten digits. The accompanied errors are in Table 1. The lowest errors are from when multiquadratic and inverse multiquadratic are used. This is the case for the average, minimum, and maximum errors. Excluding the Gaussian kernel, the average, minimum, and maximum errors are of a similar magnitude.

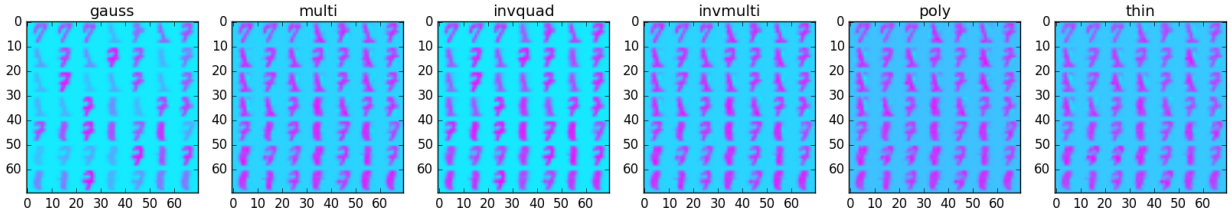


Figure 11: We show how varying the radial basis function affects the recovered images when they are originally projected from $\mathbb{R}^{64}$ into 10 dimensions using PCA. We do this to see which one is optimal and to be used in prediction. The errors are shown in Table 1.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.4541	0.1428	0.8109
multi	0.2414	0.1184	0.4427
invquad	0.2924	0.1388	0.4820
invmulti	0.2414	0.1184	0.4427
poly	0.2645	0.1212	0.5502
thin	0.2777	0.1422	0.5834

Table 1: Here are the errors between the original images and the recovered images when we used PCA to project down to 10 dimensions on the digits one and seven.

The next set we do cross validation on includes handwritten digits from zero through nine. As will be a reoccurring theme, the thin plate spline has the lowest error on the testing set.

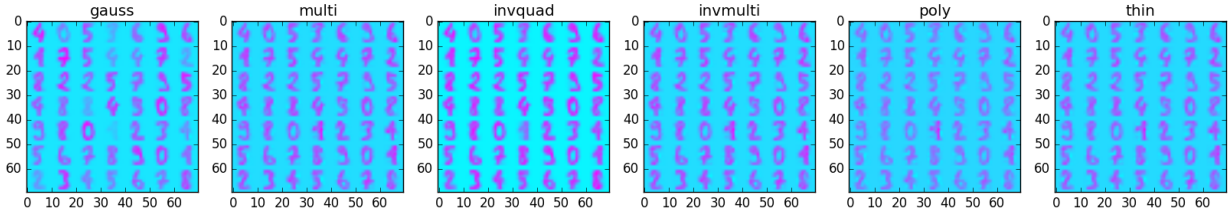


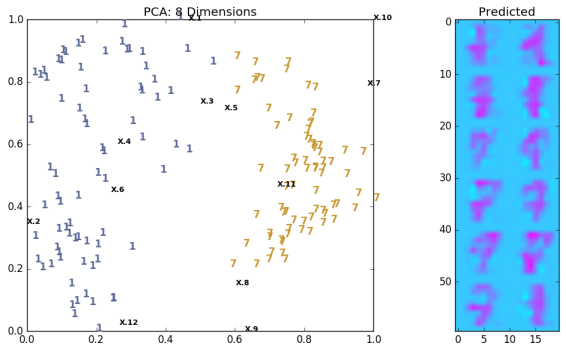
Figure 12: Above are the recovered images for when a PCA projection down to 10 dimensions was used on images of ones and sevens and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.3988	0.1336	0.8667
multi	0.2624	0.1330	0.4153
invquad	0.2794	0.1249	0.4859
invmulti	0.2624	0.1330	0.4153
poly	0.2834	0.1477	0.6138
thin	0.2530	0.1223	0.3950

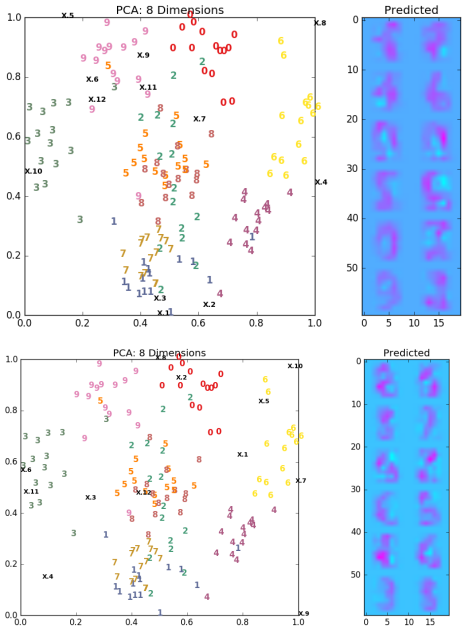
Table 2: Here are the errors between the original images and the recovered images when we used PCA to project down to 10 dimensions on the digits zero through nine.

Predicting

In Figure 13, we are using PCA to project the given ones and sevens down to a smaller dimension. After finding the corresponding A matrix, we are taking a randomly selected array of numbers that are within a certain constraint (between the maximum and minimum values of each row) to try and recover.



(a) PCA on images of ones and sevens.



(b) PCA projection using the numbers zero through nine into eight-dimensional space. The recovered images are featured.

Figure 13: Predicting after cross validation on the kernel.

4.2.2 Random Sparse Matrix

Cross Validation

For our kernel cross validation for the random sparse matrix, the minimum error was different from the lowest maximum and average errors. On the ones and sevens, multiquadratic and inverse multiquadratic did the best for the latter while inverse quadratic did the best on the former. Looking at the recovered images, all three recover the images correctly, as in we can identify what they are just by looking. The error is meaningful if exact pixel by pixel recovery is wanted. We don't always need that; we just need to know enough to make decisions or to confirm or deny a particular person painted that painting.

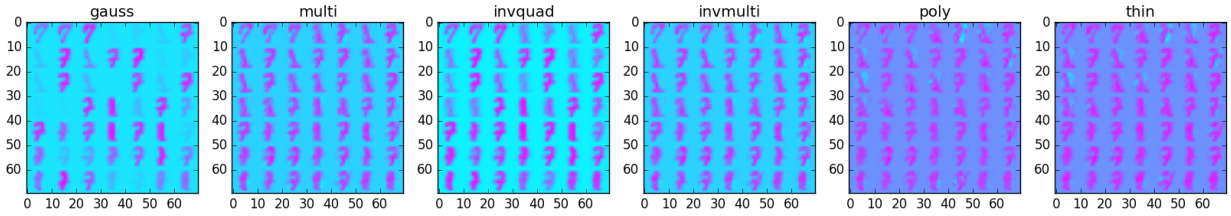


Figure 14: Above are the recovered images for when a RSM projection down to 10 dimensions was used on images of ones and sevens and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.5613	0.1499	0.9851
multi	0.3483	0.1537	0.5378
invquad	0.3890	0.1488	0.6608
invmulti	0.3483	0.1537	0.5378
poly	0.4492	0.1646	1.0220
thin	0.3924	0.1558	0.7924

Table 3: Here are the errors between the original images and the recovered images when we used RSM to project down to 10 dimensions on the digits one and seven.

On zero through nine, we see that the minimal errors range between 15 and 17 percent. The lowest here is greater than the highest minimum error of PCA. Another surprising find is that even though a greater variety of numbers are being used to create the K and A matrices, the errors for using ten digits is almost exclusively better. But this is understandable, ones and sevens can look alike but a six and a two do not. There are less similarities between each digit giving rise to more adaptability.

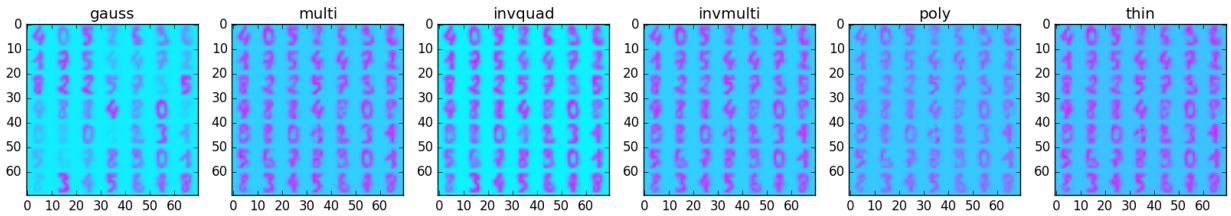


Figure 15: RSM in 10 dimensions

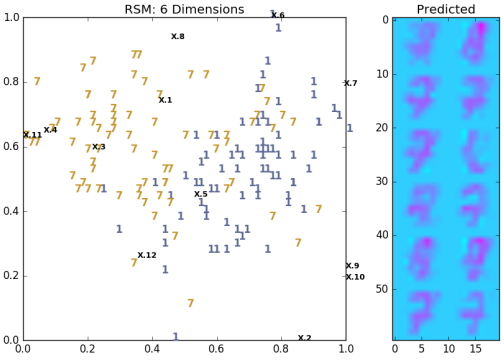
Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.5242	0.1541	0.9574
multi	0.3534	0.1621	0.6213
invquad	0.3623	0.1511	0.5894
invmulti	0.3534	0.1621	0.6213
poly	0.3874	0.1746	0.9019
thin	0.3529	0.1650	0.5731

Table 4: Here are the errors between the original images and the recovered images when we used RSM to project down to 10 dimensions on the digits zero through nine.

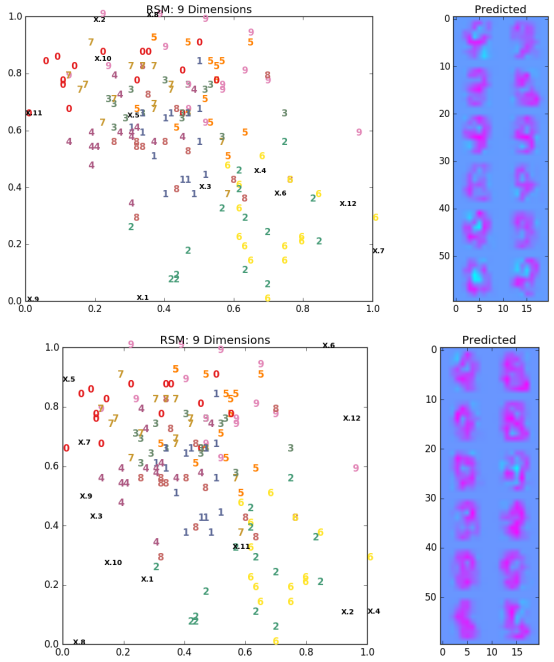
Predicting

For predicting our new low dimensional representations of hopeful digits, we see various degrees of recognizable digits. The first image looks like a three, an "s", or even a goose. The second image on the ones and seven looks like an eight. We are not using any eights in this set. The point X.2 is located away from the cluster of ones and sevens. It's not the only one – X.9 and X.10 are closer but still far away. X.10 looks like a seven but X.9 looks like a six.

X.2 of the first zero through nine does in fact look like what it is nearest in the two dimensional plot, a nine. X.7 looks like a six, to which it is close. X.8 is in the same boat with it looking like a nine. In the second of the zero through nine, X.4 is a very pleasing 8.



(a) RSM on images of ones and sevens.



(b) RSM projection using the numbers zero through nine into nine-dimensional space. The recovered images are featured.

Figure 16: Predicting after cross validation on the kernel.

Now that we are done with the linear methods, we can move onto the nonlinear methods.

4.3 Nonlinear Methods

4.3.1 Isomap

Cross Validation

Like with the linear methods, we see that the Gaussian kernel has poor error. Switching to any of the other kernels cuts the average error in half. This error is still pixel by pixel. We can agree that from simply looking at the Gaussian kernel's image recoveries that it does not recover images very well. Visually, one might opt for the inverse quadratic kernel but for our predicting purposes, we shall choose to use multiquadractic.

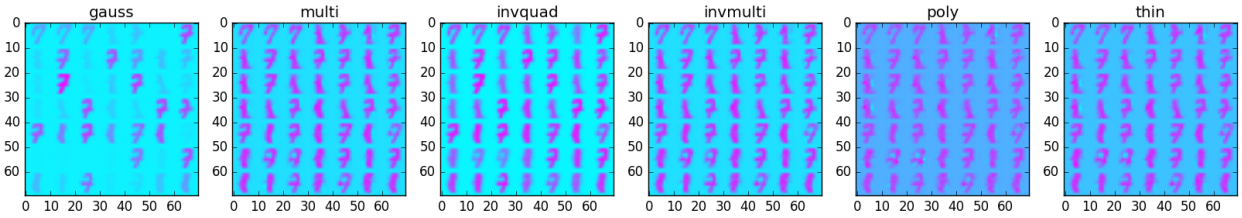


Figure 17: Above are the recovered images for when an Isomap projection down to 10 dimensions was used on images of ones and sevens and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.7056	0.2597	0.9991
multi	0.3288	0.1726	0.6432
invquad	0.3884	0.1789	0.6903
invmulti	0.3288	0.1726	0.6432
poly	0.3759	0.1907	0.7919
thin	0.3310	0.1825	0.6744

Table 5: Here are the errors between the original images and the recovered images when we used Isomap to project down to 10 dimensions on the digits one and seven.

Looking at zero through nine and their errors, the thin plate spline and multiquadratic/inverse multiquadratic have the lowest error. Despite not having the lowest average error, think plate was chosen for its maximum and minimum errors were the least.

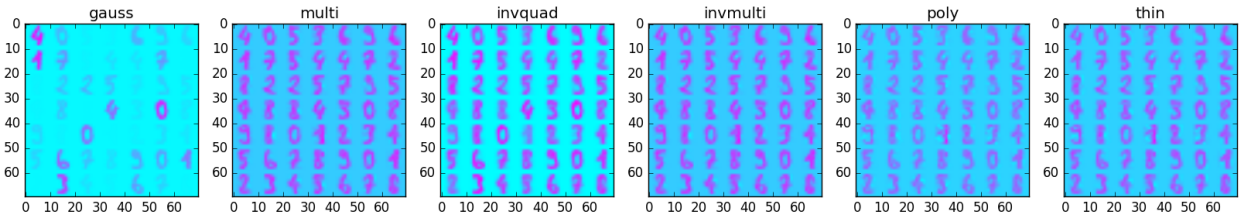


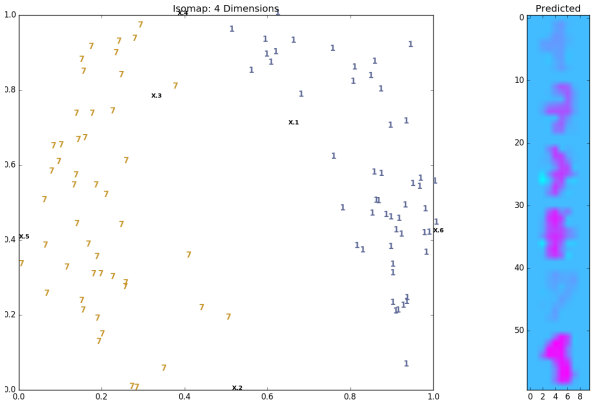
Figure 18: Above are the recovered images for when an Isomap projection down to 10 dimensions was used on images of zeros through nines and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.7944	0.2262	0.9971
multi	0.2848	0.1416	0.4982
invquad	0.3466	0.1441	0.5921
invmulti	0.2848	0.1416	0.4982
poly	0.3245	0.1457	0.5720
thin	0.2946	0.1379	0.4978

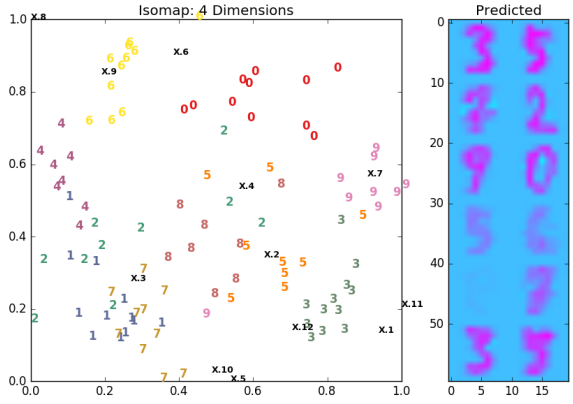
Table 6: Here are the errors between the original images and the recovered images when we used Isomap to project down to 10 dimensions on the digits zero through nine.

Predicting

In Figure 19a, we are using Isomap to project the given ones and sevens down to a smaller dimension. Here, we see point X.1 is predicted as a 1, point X.2 and X.3 as 7's, point X.5 as a faint 7, and point X.6 as a thick 1. Point X.4 most prominently resembles a 1 but we do see some characteristics of a seven with a bar. What is most interesting to note about this point is that it is between the ones and sevens in the plot on the left. This scatter plot plots the first two components of the training $\mathbf{y}^{(i)}$'s in 2-dimensional space with respective number labels. We implemented a Gaussian kernel and a four dimensional Isomap projection with training on ones and sevens.



(a) Isomap on images of ones and sevens.

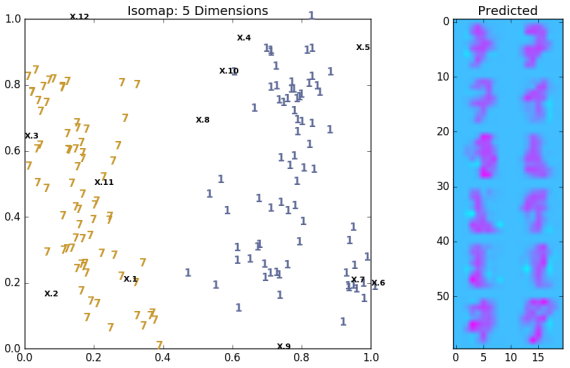


(b) Isomap projection using the numbers zero through nine into four-dimensional space. The recovered images are featured.

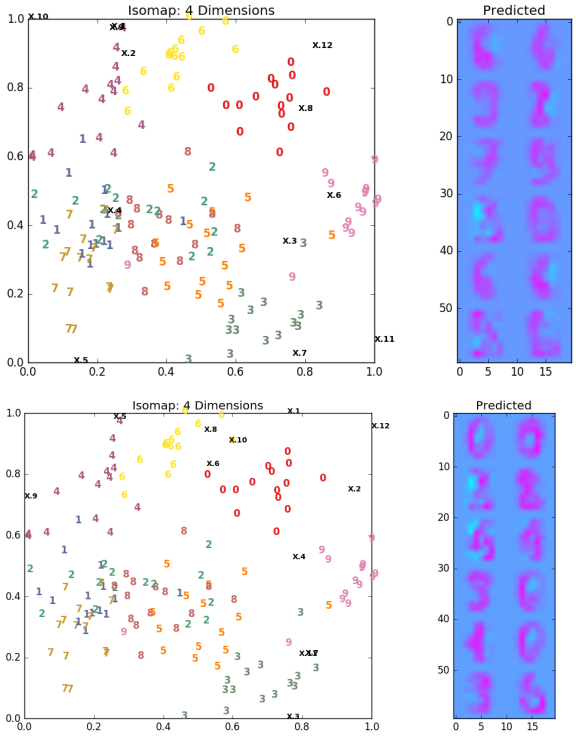
Figure 19: Predicting before cross validation on the kernel.

Before running any cross validation tests, we ran the algorithm on a four dimensional Isomap projection and found some nice results. The images in 19b, especially the easily recognizable as numbers, are in their number clusters. Even down to four dimensions, digits can be predicted to look like digits. It is like taking one-sixteenth of a blended puzzle and telling us what it is from just that.[18]

[18]It's not exactly the same but it gets the point across of the amazing feat.



(a) Isomap on images of ones and sevens.



(b) Isomap projection using the numbers zero through nine into four-dimensional space. The recovered images are featured.

Figure 20: Predicting after cross validation on the kernel.

Now we can look at what happens when we know what methods will work well. We use an Isomap down to five dimensions on the ones and sevens. Wonderfully, we recognize at least two-thirds of the images as either ones or sevens. Not included in this fraction are those that are blurry but still could be considered a particular number.

Isomap is really nice to look and plot, the numbers are clustered together. This observation is powerful. It means that, at this dimension, half of what we project down to can tell us what digit the point most likely would be. More impressively, a combination between a tree and the recovery algorithm could give one both the image and the number. This is just the case here.

4.3.2 Local Tangent Space Alignment

Cross Validation

LTSA does not perform very well visually. The only method whose maximum error is under 1 is inverse quadratic. Even its maximum error less than the polyharmonic kernel's average error.

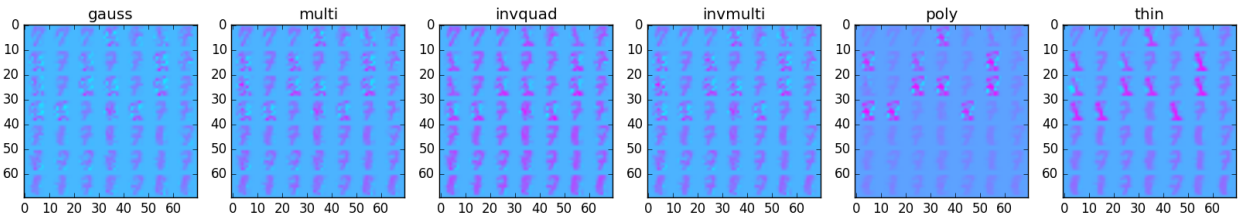


Figure 21: Above are the recovered images for when a LTSA projection down to 10 dimensions was used on images of ones and sevens and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.4948	0.1755	1.2447
multi	0.4598	0.1732	1.0701
invquad	0.3566	0.1637	0.8362
invmulti	0.4598	0.1732	1.0701
poly	0.9374	0.1152	4.4440
thin	0.4774	0.1314	1.5874

Table 7: Here are the errors between the original images and the recovered images when we used LTSA to project down to 10 dimensions on the digits one and seven.

Surprisingly, the errors when the kernels were varied on the digits zero through nine performed exceedingly better than just on the ones and sevens. Both the images and the errors are more accurate. The thin plate spline overall performs the best out of all six. We do have to be aware that the kernel selected through this process is preferable for 10 dimensions. Later, we use this one kernel for testing each of the dimensions to find the one with the smallest pixel by pixel error. This was done before the major of the predictions so the resulting dimension choice is purposeful.

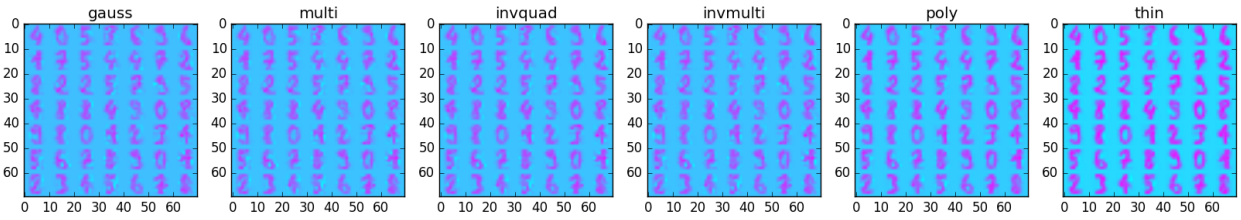


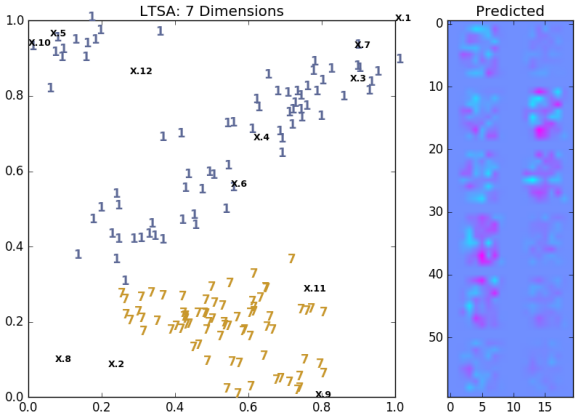
Figure 22: Above are the recovered images for when an LTSA projection down to 10 dimensions was used on images of zeros through nines and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.3612	0.1809	0.5990
multi	0.3588	0.1704	0.6287
invquad	0.3350	0.1667	0.5928
invmulti	0.3588	0.1704	0.6287
poly	0.2894	0.1563	0.4689
thin	0.2704	0.1324	0.3854

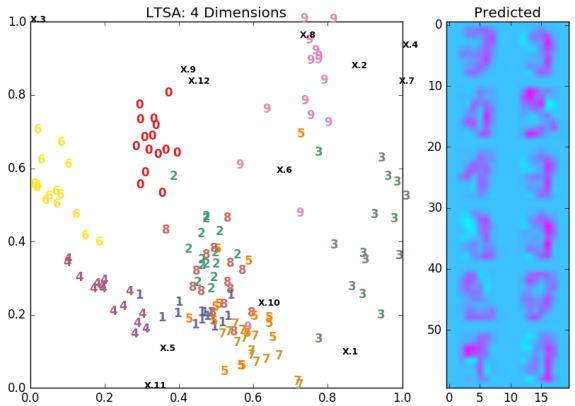
Table 8: Here are the errors between the original images and the recovered images when we used LTSA to project down to 10 dimensions on the digits zero through nine.

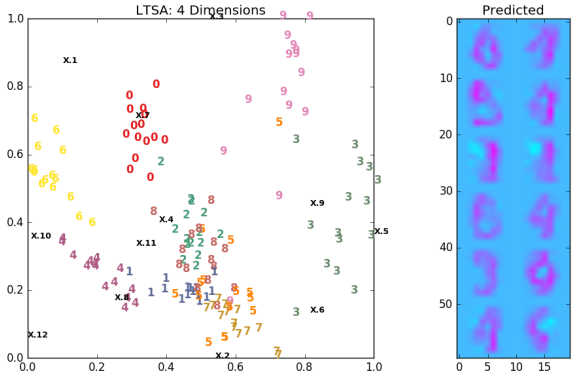
Predicting

Local Tangent Space Alignment poorly predicts any recognizable digits when dealing with just ones and sevens. As our previous finding, we see better digit images when dealing with zero through nine. The dimensions are different too. LTSA performed better at a lower dimension and with more digit types than did it ones and sevens counterpart.



(a) LTSA on images of ones and sevens.





(b) LTSA projection using the numbers zero through nine into four-dimensional space. The recovered images are featured.

Figure 23: Predicting after cross validation on the kernel.

A couple of points of interest in the bottom of Figure 23b are X.7, X.8, and X.9. These three images are located near digits of the same type. X.7 is in the cluster of zeros; X.8 is in the cluster of fours; and X.9 is near the cluster of threes.

4.3.3 Spectral Embedding

Cross Validation

Figure 24 shows our cross validation results on the digits one and seven. Preliminary observations suggest that the inverse quadratic kernel will be the best to use for recognizable digit recovery. The errors in Table 9 confirm the belief. The inverse quadratic kernel had the lowest of all three errors, strikingly so for the average error.

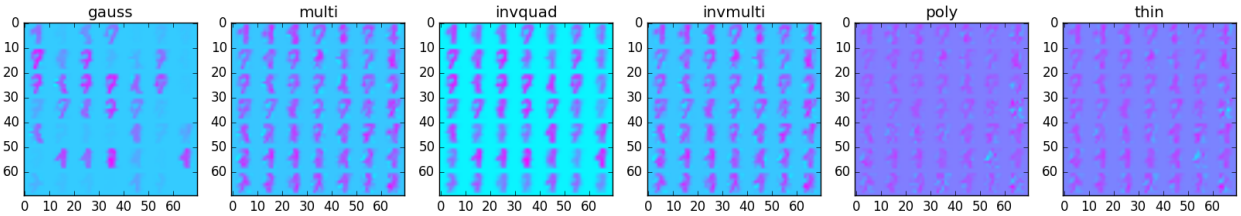


Figure 24: Above are the recovered images for when a Spectral Embedding projection down to 10 dimensions was used on images of ones and sevens and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.9184	0.7072	0.9998
multi	0.8089	0.4494	1.1071
invquad	0.6697	0.4006	0.8584
invmulti	0.8089	0.4494	1.1071
poly	1.2359	0.5363	2.5190
thin	1.0501	0.4370	1.7108

Table 9: Here are the errors between the original images and the recovered images when we used Spectral Embedding to project down to 10 dimensions on the digits one and seven.

Similarly when using the digits zero through nine, the inverse quadratic kernel has the best average and maximum errors. The minimum error is best with multiquadratic and inverse multiquadratic but the corresponding maximum errors are relatively high. This means, for both ones and sevens and zero through nine, the same kernel, inverse quadratic, is the kernel of choice.

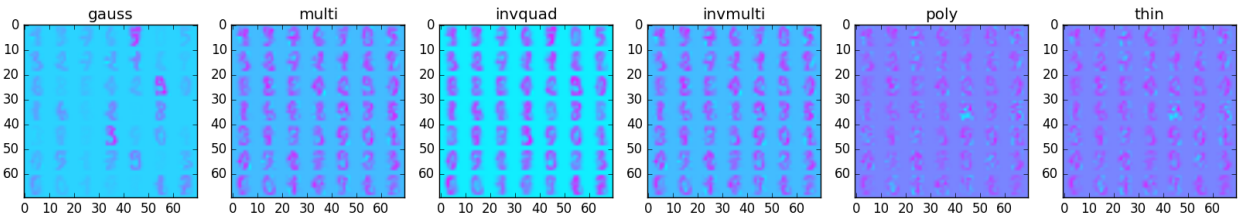


Figure 25: Above are the recovered images for when a Spectral Embedding projection down to 10 dimensions was used on images of zeros through nines and the kernel type was varied.

Kernel Type	Average Error	Minimum Error	Maximum Error
gauss	0.9190	0.7016	0.9988
multi	0.7913	0.4358	1.2477
invquad	0.6446	0.4777	0.8299
invmulti	0.7913	0.4358	1.2477
poly	1.1981	0.5549	2.6546
thin	1.0658	0.4683	2.0662

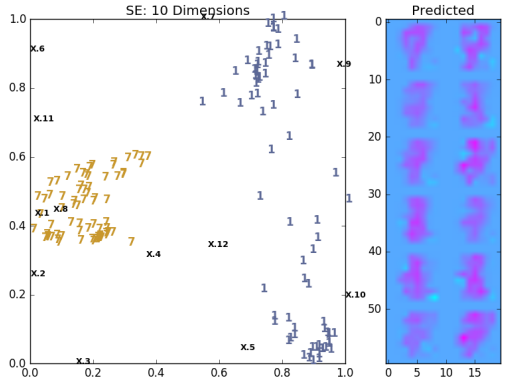
Table 10: Here are the errors between the original images and the recovered images when we used Spectral Embedding to project down to 10 dimensions on the digits zero through nine.

Predicting

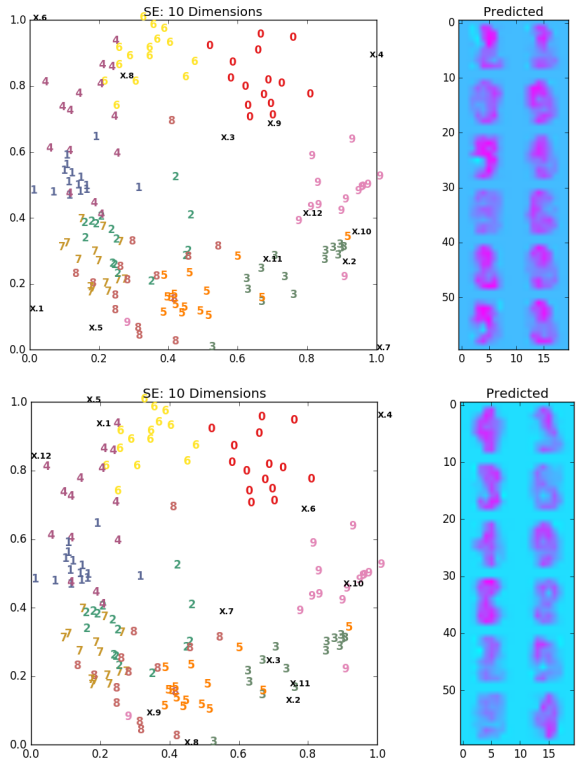
In Figure 26a, X.1, X.2, and X.8 are either in or near the cluster of sevens with respect to their first two components in the projection space. X.9 and X.10 are vaguely recognizable as ones and do happen to be near the ones.

Figure 26b has more recognizable digit images than did the images for ones and sevens. For the top, the images of note are X.2, located near the threes and looking like a three; X.3, resembling somewhat of a zero and near the zeros; and X.10 and X.12 who are near clusters of nines and threes and have characters of both, but more so of threes.

The bottom has notable images X.1 and X.12, looking like fours and near one cluster of fours; X.2 and X.3, located near the threes and recognized as threes; and X.8 which looks like an eight and happens to be located near one.



(a) Spectral Embedding on images of ones and sevens.



(b) Spectral Embedding projection using the numbers zero through nine into ten-dimensional space. The recovered images are featured.

Figure 26: Predicting after cross validation on the kernel.

4.4 Exploring the Results

Here we see how modifying parts of the code yields different results. Additionally, the limitations and observations found from doing this.

4.4.1 Various Dimensions for Projecting and Their Effects

Isomap

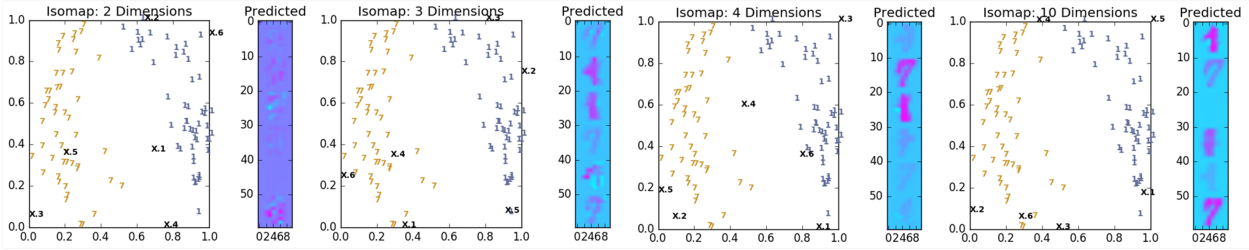


Figure 27: Isomap projections onto 4 different dimensions.

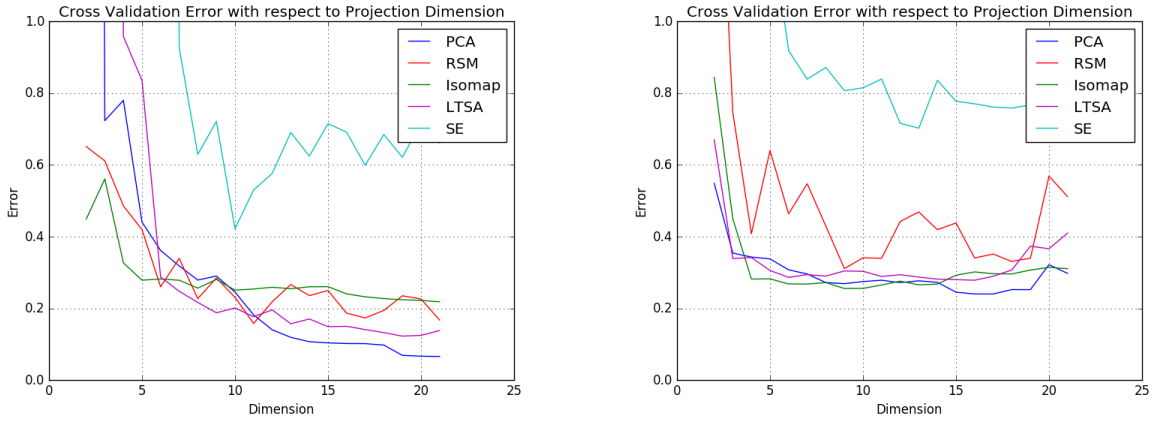
Displayed in Figure 27 are Isomap projections of four different dimensions, 2, 3, 4, and 10. The ones and sevens are color-coded. When we project down to 2 dimensions from the original 64, we see that there is not enough information retained to predict. As we go up in dimensions, even just one, our predicted images more distinctly resemble digits. You may notice we do not see a number in the third image of 10 dimensions. It happens to be that X.3 is relatively far from the clusters of ones and sevens. Mind you, the X.1 through X.6 points have no associated images to start off with. They are like the potential Einstein essences being used to predict never-before-seen Einsteins.

General

Between Figures 28a and 28b and Figures 29a and 29b, we see how the dimension affects the recovery of the image. In Figure 28, we see a general trend of better recovery as the dimension increases. The radial basis functions used here are the ones determined the best at the 10 dimensional space level. This actually does not mean that that kernel is best for the other dimensions, but it is fair to use the same. Again, the plots are misleading, in a sense. Typically, the kernel performed best on average and either the minimum or maximum error. One of the other kernels might have given lower error but that is not the purpose of these graphs. We want to see where the knee of the graph is so to determine which dimension we want to project into.

Personally, I would not recommend using Spectral Embedding or Local Tangent Space Alignment with the algorithm. For one, Spectral Embedding's error is quite high. LTSA might have started working better than Isomap as the dimension increased but more times than not, we want to make the dimension as small as possible while still being able to recover as much as possible. Isomap is consistent once you hit four or five dimensions. Using a higher projection space dimension will not make much of a difference.

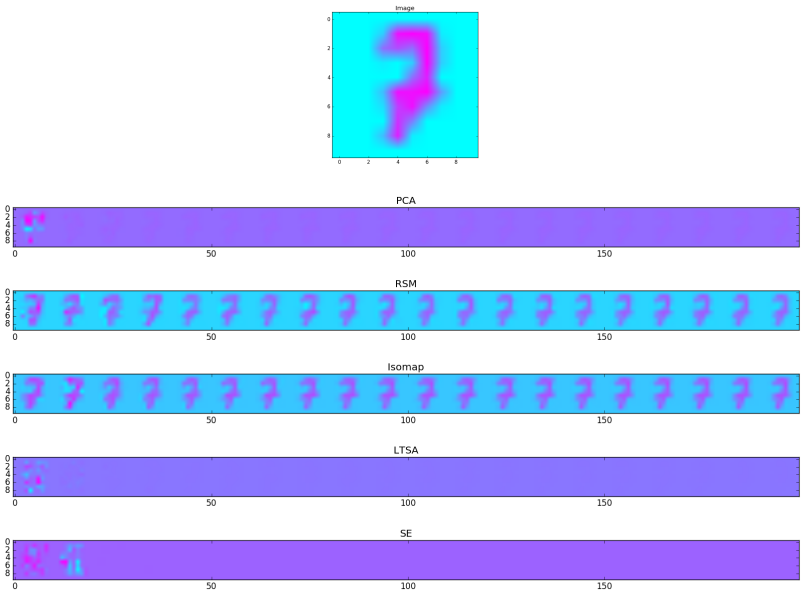
As expected, PCA performed better than the other methods with this algorithm as its projection dimension increased.



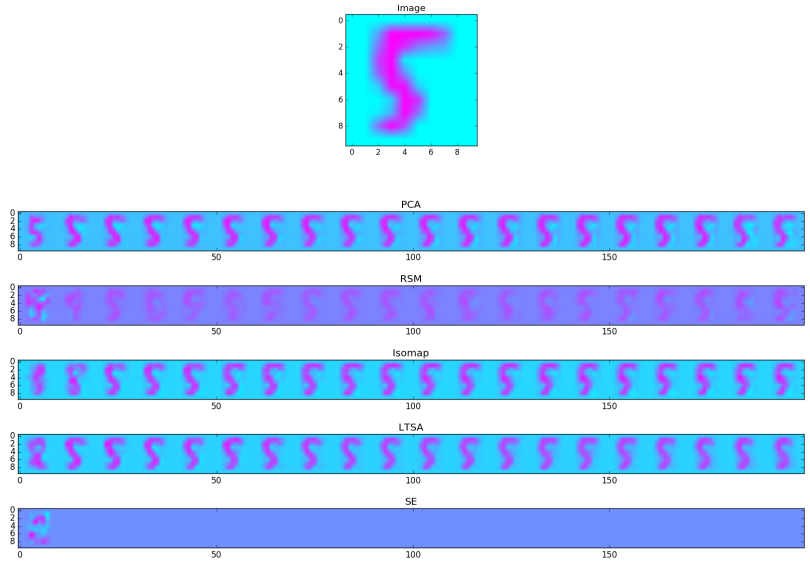
(a) On Ones and Sevens

(b) On Zero through Nine

Figure 28: Errors from the methods using the determined kernel from 2 dimensions up to 21.



(a) On Ones and Sevens



(b) On Zero through Nine

Figure 29: Images from the methods using the determined kernel from 2 dimensions up to 21.

5 Conclusions

Over the different linear and nonlinear methods, PCA and Isomap are the preferred methods to use with the recovery algorithm. If low level dimensions are needed, using an Isomap projection on the data works the best and is most consistent of the methods once the dimension is increased to four or five.

Even down at two dimensions, the algorithm was able to successfully recover the image, pixel by pixel, with around 50 percent error. This is an amazing feat since we started with 64 dimensions. What helps is that Isomap has more clearly defined clusters of digits on the two-dimensional plots than most of the others. We tended to find sixes and threes to be the easiest to recover and predict. Odds were that if the point corresponding to the first two components was located near or in a cluster of sixes, it would be recovered as a six.

We can say that the algorithm proposed in [6] works well and with increased image variety, thin plate splines tend to be the better choice of radial basis function kernels. A Gaussian kernel does not fare all that well in comparison.

Appendices

A Python Code

A.1 The Methods

PCA

```
import numpy as np
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
pca.fit(np.transpose(x))
y_pca = np.transpose(pca.transform(np.transpose(x)))
```

Where `x` is a matrix of column vectors. `n_components` can be changed to equal a different value, it does not have to be two. `y_pca` is a matrix with the same number of columns as `x` but with a different number of rows, two in the given case.

Random Sparse Matrix

```
import numpy as np
from sklearn import random_projection

rsm = random_projection.SparseRandomProjection(n_components = 2)
rsm.fit(np.transpose(x))
y_rsm = np.transpose(rsm.transform(np.transpose(x)))
```

Where `x` is a matrix of column vectors. `n_components` can be changed to equal a different value, it does not have to be two. `y_rsm` is a matrix with the same number of columns as `x` but with a different number of rows, two in the given case.

Isomap

```
import numpy as np
from sklearn import manifold

iso = manifold.Isomap(n_neighbors,n_components = 2)
iso.fit(np.transpose(x))
y_iso = np.transpose(iso.transform(np.transpose(x)))
```

Where `x` is a matrix of column vectors. `n_components` can be changed to equal a different value, it does not have to be two. `n_neighbors` is set to 30. `y_iso` is a matrix with the same number of columns as `x` but with a different number of rows, two in the given case.

Local Tangent Space Alignment

```
import numpy as np
from sklearn import manifold

ltsa = manifold.LocallyLinearEmbedding(n_neighbors,n_components = 2,method = 'ltsa')
ltsa.fit(np.transpose(x))
y_ltsa = np.transpose(ltsa.transform(np.transpose(x)))
```

Where `x` is a matrix of column vectors. `n_components` can be changed to equal a different value, it does not have to be two. `n_neighbors` is set to 30. `y_ltsa` is a matrix with the same number of columns as `x` but with a different number of rows, two in the given case.

Spectral Embedding

```
import numpy as np
from sklearn import manifold

se = manifold.SpectralEmbedding(n_components = 2,n_neighbors=30)
se.fit(np.fit_transpose(x))
y_se = np.transpose(se.fit_transform(np.transpose(x)))
```

Where `x` is a matrix of column vectors. `n_components` can be changed to equal a different value, it does not have to be two. `n_neighbors` is set to 30. `y_se` is a matrix with the same number of columns as `x` but with a different number of rows, two in the given case.

B Extensions, Notes, and Ideas

In this work, we put the images into column vectors before projecting down. Another way to explore is to take each individual image and project them down into lower dimension then to reshape it into a column vector. After that, running the algorithm to do the recovery or prediction.

Another thing to look at is to compare the different kernels and each of the projection dimensions. Here, we looked at the best kernel on 10 dimensions and used that to see which dimension has the best accuracy with wanting that value to be small.

References

[1] 4.5 random project – scikit-learn 0.17.1 documentation. http://scikit-learn.org/stable/modules/random_projection.html#random-projection.

[2] Data sets for nonlinear dimensionality reduction. Hands and Faces images, http://isomap.stanford.edu/datasets.html.

[3] author/editor. title. http://scikit-learn.org/stable/_images/plot_lle_digits_005.png.

[4] Trevor Hastie Gareth James, Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer.

[5] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer.

[6] Nathan D Monnig, Bengt Fornberg, and Francois G Meyer. Inverting nonlinear dimensionality reduction with scale-free radial basis function interpolation. *Applied and Computational Harmonic Analysis*, 37(1):162–170, 2014.

[7] Ker Than. Why great minds can't grasp consciousness, August 2005. Albert Einstein image, http://www.livescience.com/366-great-minds-grasp-consciousness.html.

[8] Yuan Yao. Lecture 11: Geometric data analysis: Local tangent space alignment (ltsa), May 2011. http://www.math.pku.edu.cn/teachers/yaoy/Spring2011/lecture11_2.pdf.

[9] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. 26. http://epubs.siam.org/doi/abs/10.1137/S1064827502419154.