SLT coding exercise #1

Locally Linear Embedding

https://gitlab.vis.ethz.ch/vwegmayr/slt-coding-exercises

Due on Monday, March 6th, 2017

Asha Anoosheh
16-948-358

Contents

The Model

The model section is intended to allow you to recapitulate the essential ingredients used in Locally Linear Embedding. Write down the *necessary* equations to specify Locally Linear Embedding and and shortly explain the variables that are involved. This section should only introduce the equations, their solution should be outlined in the implementation section.
Hard limit: One page

I feel like there aren't enough confusing equations for me to need this section this time :) The Implementation section is concise enough, in my opinion.

The Questions

(a) Data was downloaded using SkLearn's built-in dataset manager to avoid file format importing issues.

(b) **Figures 1 & 2** show the results of my LLE implementation. They are run on the same random 10% of MNIST data (7000 datum), using 4 nearest-neighbors and Euclidean distance as the metric.

As the points are colored by their digit label, it is very possible to see distinct clusters for most classes even in 2 or 3 dimensions, though it is clearer in the latter.

(c) After post-processing the visualization of the M matrix to clearly see something as **Figure 3**, I noticed the diagonal has a consistent "deterioration" from both corners toward the middle, where values go toward zero. The matrix, as expected, is also extremely sparse as most values are zero or close to zero (visualized as red).

And plotting the reverse-sorted singular values in **Figure 4** (equal to the eigenvalues of a PSD matrix such as M) initially shows only an elbow drop in the values on the larger end. But looking closer at the small end in **Figure 5**, we can find a flat "elbow" of sorts for our purposes as well, since we want the smaller ones instead. The optimal dimension could be the one with eigenvalues before a certain rise in slope - similar to the corner method in many applications.

(d) Holding everything else constant and changing the number of nearest-neighbors for a sample of 4000 datum, one notices using less than four nearest neighbors results in a degenerate M matrix - with negative eigenvalues and thus not PSD. The reconstruction error, equivalent to the sum of the eigenvalues for positive eigenvalues, is positive and lowest for $k = 4$ neighbors, and this is plotted in **Figure 6**.

Additionally, I tried another distance function (cosine angle) for NN that makes sense for images (and high-dim data in general). Although its reconstruction error was usually on the same order of magnitude as euclidean - regardless of data points used, it was consistently 6x larger. Surprisingly, Euclidean distance did better, at least according to this error definition.

(e) The process simply involves finding nearest neighbors of a chosen point in the embedded space and weighing them by inverse distance (normalized), but adding together the weighted 784-dim images corresponding to the embedded neighbors instead.

I have excluded some images for lack of space due to the 2-page requirement, but I have included two images of points as I moved from the space within digit 7 towards 3/8 (**Figures 7 & 8**).

Images resulting from "outside" the simplex region (not pictured) are still fine because it uses a fixed number of neighbors anyway that usually end up being of the same class. The idea should hold in higher dim, but the distance function used to calculate neighbors deteriorates in higher dim of course.

Performing this in the original space (not pictured) just looks like overlaid digit pixels with less structure (though not completely noisy). The original dimension lacks the geometry that maintains the reconstruction properties that the embedded dimension does, where distances between points imply manifold distance rather than whatever some high-dim distance function would.

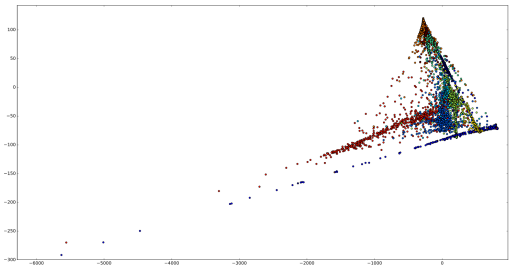

Figure 1: 2D Embedding

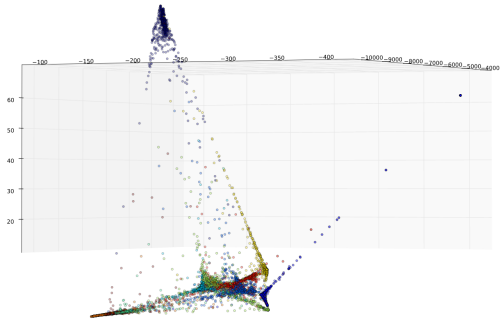

Figure 2: 3D Embedding

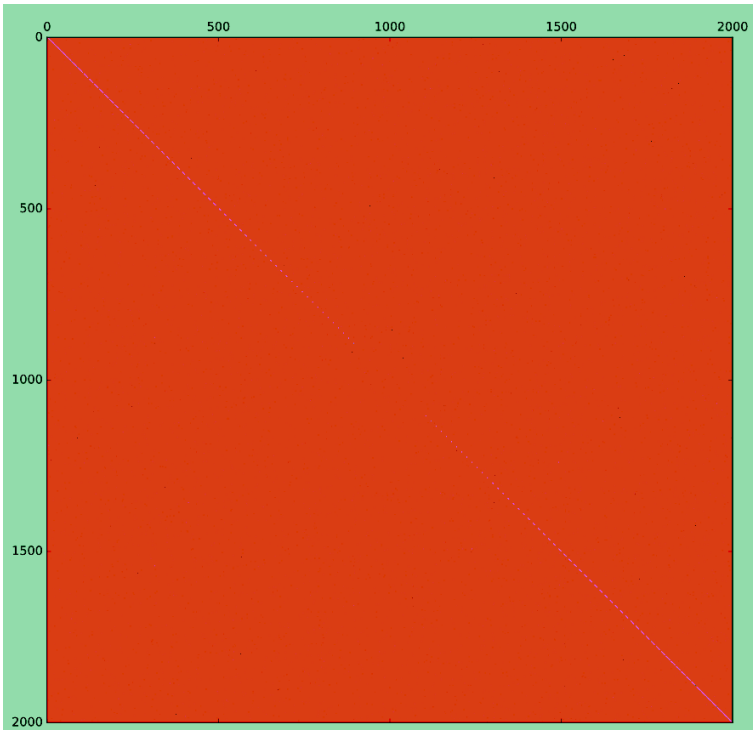

Figure 3: M matrix colorized

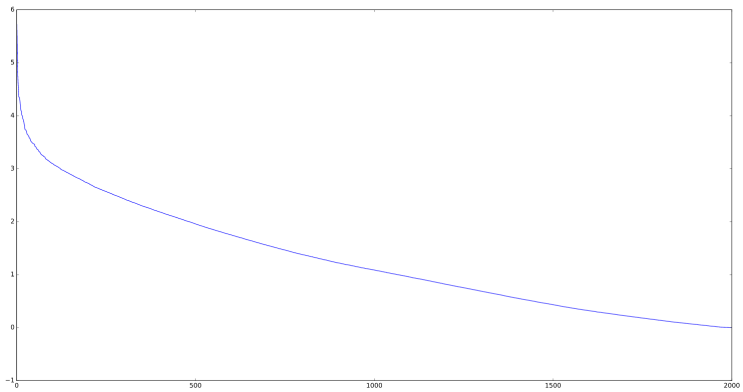

Figure 4: eigenvalues of M

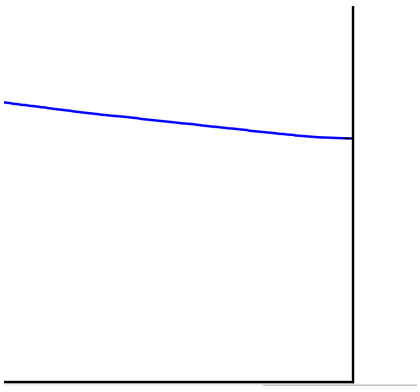

Figure 5: zoomed-in on smallest

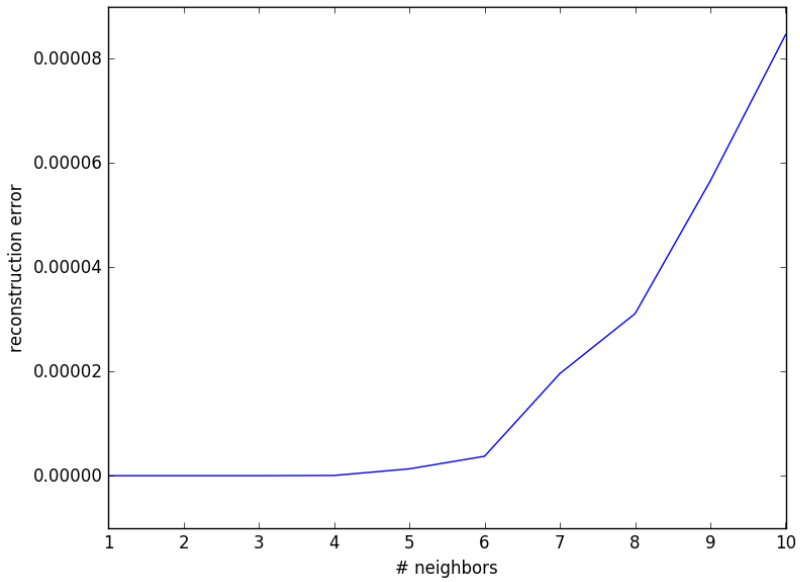

Figure 6

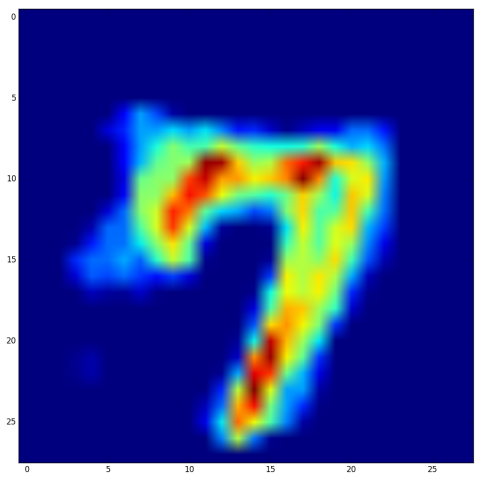

Figure 7: Interpolated 7

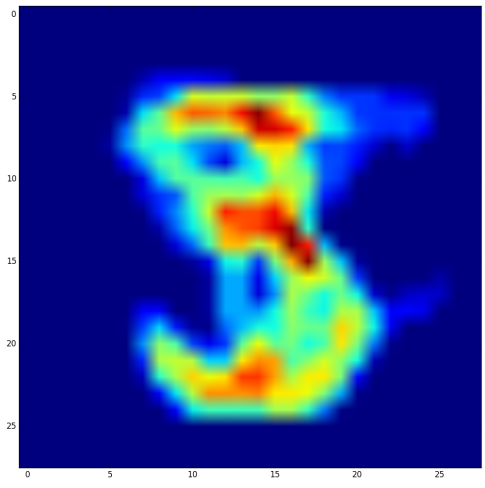

Figure 8: Interpolated between 3/8

The Implementation

My LLE implementation first creates the W matrix mentioned in the original paper, which represents a graph of barycentric weights from each data point to another.

This involves finding the K nearest neighbors of all points to another, which is done using a built-in function with multi-core functionality and the default Minkowski (Euclidean in this case) distance which performs the best, somehow.

Then the optimal reconstruction weights are found by solving a least squares problem on the covariance matrix of the difference between each point and its neighbors, essentially leading to finding weights w such that $Cw = 1$. A w vector is found per data point, using a built-in least-squares solver, and stored in an $N \times k$ matrix W.

After deriving a lot of other stuff, we essentially need to create a matrix $M = (I - W)^T(I - W)$ and extract its eigenvectors corresponding to the smallest eigenvalues, which can be done using SciPy's built-in eigenvalue solver. Strangely, the sparse solver does not seem to work whereas the dense one does, and I cannot seem to figure out why.

The resulting smallest eigenvectors can be used as the reduced-dimension data, although technically the smallest eigenvalue is apparently always zero, so that one should be discarded of course.

Problems Encountered along the Way:

The experiments run on the MNIST dataset were done on a (deterministically) random subset of the images instead of all original 70,000, due to significant computation time when performing nearest-neighbors. Usually under 10000 images were used.

Initally I made my own nearest-neighbors function, but I switched to SkLearn's version after seeing it had multi-processing support.

Numerical issues were encountered when using fewer than 4 neighbors, as M would be created as a non-PSD matrix whose eigenvalues would be negative.

The returned eigenvectors are apparently known to be negated and require scaling by the square root of the eigenvalues of M to be considered truly-transformed data, but of course this is not a necessity.

M is known to be sparse and handling it as a dense matrix is memory-intensive compared to using a sparse matrix from SciPy's CSR implementation.

Your Page

YOUR GIT BRANCH

References:

1. https://www.cs.nyu.edu/~roweis/lle/papers/lleintro.pdf
2. http://axon.cs.byu.edu/Dan/678/miscellaneous/Manifold.example.pdf