SLT coding exercise #1

Locally Linear Embedding

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

Due on Monday, March 6th, 2017

Maximilian Grüner
13-061-007

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

The following equations are needed to describe the problem

- Error function for reconstruction of data points depending on the weight vector
  $\xi(W) = \sum_i |\overrightarrow{X_i} - \sum_j W_{ij} \overrightarrow{X_j}|$
  W is the weight vector for the data points which are described with $\overrightarrow{X}$. W_{ij} is 0 for all $\overrightarrow{X_j}$ which are not one of the k-th neighbours of $\overrightarrow{X_i}$. $\sum_j W_{ij} = 1$ as well.

- Error function for the reconstruction of the low dimensional representation of the data points. This depends on the choice of the low dimensional representation of $\overrightarrow{X}$
  $\phi(Y) = \sum_i |\overrightarrow{Y_i} - \sum_j W_{ij} \overrightarrow{Y_j}|$
  W is the weight vector for the data points which are described with $\overrightarrow{Y}$. W_{ij} is 0 for all $\overrightarrow{Y_j}$ which are not one of the k-th neighbours of $\overrightarrow{Y_i}$

The Questions

This is the core section of your report, which contains the tasks for this exercise and your respective solutions. Make sure you present your results in an illustrative way by making use of graphics, plots, tables, etc. so that a reader can understand the results with a single glance. Check that your graphics have enough resolution or are vector graphics. Consider the use of GIFs when appropriate.
Hard limit: Two pages

(a) Get the data

For this exercise we will work with the MNIST data set. In order to learn more about it and download it, go to http://yann.lecun.com/exdb/mnist/.

(b) Locally linear embedding

Implement the LLE algorithm and apply it to the MNIST data set. Provide descriptive visualizations for 2D & 3D embedding spaces. Is it possible to see clusters?

(c) Cluster structure

Investigate the cluster structure of the data. Can you observe block structures in the M matrix (use matrix plots)? Also plot the singular values of M. Do you notice something? Can you think of ways to determine the optimal embedding dimension?

(d) Nearest Neighbors

Investigate the influence of the choice of how many nearest neighbors you take into account. Additionally, try different metrics to find the nearest neighbors (we are dealing with images!).

(e) Linear manifold interpolation

Assume you pick some point in the embedding space. How can you map it back to the original (high dimensional) space? Investigate how well this works for points within and outside the manifold (does it depend on the dimensionality of the embedding space?) Try things like linearly interpolating between two embedding vectors and plot the sequence of images along that line. What happens if you do that in the original space?

The Questions continued on next page...

If not described differently the minoskwi distance is used.

1. Locally linear embedding

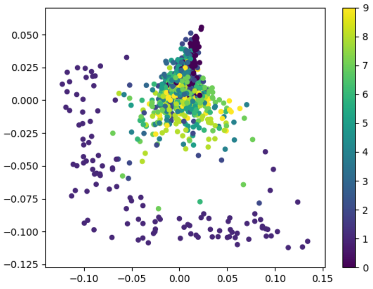

Figure 1: 2D Plot of embedding of 1000 numbers with 30 neighbors

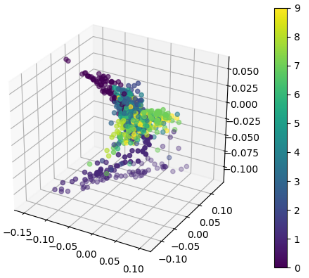

Figure 2: 3D Plot of embedding of 1000 numbers with 30 neighbors

We can clearly see clusters formed by similar numbers.

2. Cluster structure

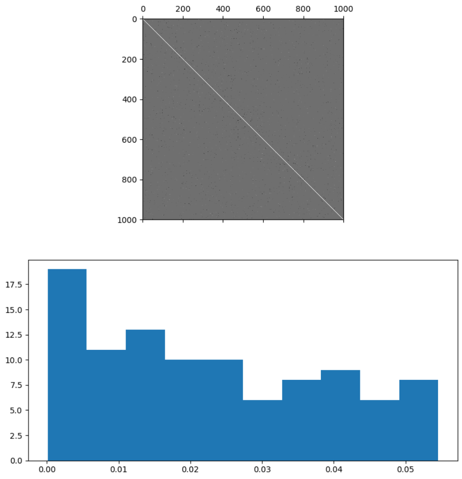

Figure 3: Matrix M and plot of the best 100 eigenvalues

It can be seen here that the most prominent values are on the diagonal of the M Matrix. So we almost have a diagonal matrix. The eigenvalues themselves occur less often if their magnitude is higher. This makes sense due to the fact that there shouldn't be that many "prominent" directions in the M matrix. We can asses the reconstruction error by summing up the considered eigenvalues and taking their inverse. So if the sum would largely grow by including another eigenvalue we can assume that the embedding is good enough.

3. Nearest Neighbors

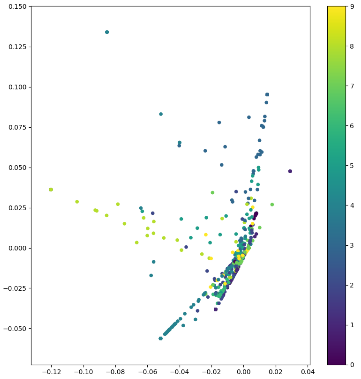

Figure 4: 2D Plot of embedding of 1000 numbers with 3 neighbors

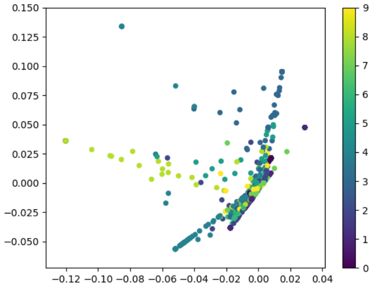

Figure 5: 2D Plot of embedding of 1000 numbers with 3 neighbors and euclidan distance

It can be seen here that the choice of neighbors influences the low dimensional embedding greatly. This is due to the fact that we determine here how many neighbours do define the "local" structure. The distances measure on the other hand is not that important. It becomes crucial when we're working in a very high dimensional space and the euclidian distance becomes useless due to the curse of dimensionality.

4. Linear manifold interpolation

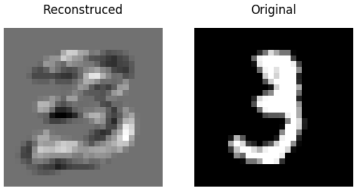

Figure 6: Reconstruction from the Manifold with 20D embedding space and 3 neighbors

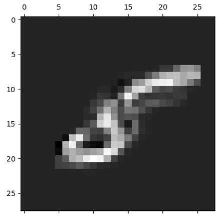

Figure 7: Reconstruction of a random number with 20D embedding space and 3 neighbors

We have to first search for the k nearest points in the embedding space. Then we can calculate the best weights to reconstruct the point. These weights are then again used in the original space with the original representation of the nearest points to reconstruct the point in the original space. Outside the manifold we tend to see a number structure but of course we don't obtain real good images of numbers. When we choose a higher dimensional embedding space the structure of the numbers becomes clearer. If we interpolate along that line one image fades into the other. This is the same behaviour seen in the real space.

The Implementation

In the implementation section you give a concise insight to the practical aspects of this coding exercise. It mainly mentions the optimization methods used to solve the model equations. Did you encounter numerical or efficiency problems? If yes, how did you solve them? Provide the link to your git branch of this coding exercise.
Hard limit: One page

The url is as follows: 13-061-007/Locally Linear Embedding
Optimization methods are not really needed. I use the methods as provided by sklearn, scipy and numpy. The most heavily optimized functions are probably the knn algorithm and the eigenvalue computation method. I did not encounter numerical problems. As described earlier one might run problems if one uses an inappropriate metric for high dimensional vectors when searching for the nearest neighbors. I might have encountered performance problems if I hadn't used the linear equation representation to solve for the weights.