SLT coding exercise #1

Locally Linear Embedding

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

Due on Monday, March 6th, 2017

Robin Spiess
12-915-302

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

Reconstruction error

$$\mathcal{E}(W) = \sum_i |X_i - \sum_j W_{ij} X_j|^2$$

where X_i are the data points, W is the weight matrix we want to determine.
The optimal weights are

$$w_j = \frac{\sum_k C_{jk}^{-1}}{\sum_{lm} C_{lm}^{-1}}$$

with $C_{jk} = (x - \eta_j) \cdot (x - \eta_k)$ where η_i Neighbor i
The weights can be computed by solving the following linear system of equations

$$\sum_j C_{jk} w_k = 1$$

and then rescaling the weights so that they sum to one for each data point.

The embedding cost

$$\Phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2 = \sum_{ij} M_{ij} (Y_i \cdot Y_j)$$

Where Y_i is the low dimensional embedded vector of X_i and M is the matrix

$$M = (I - W)^T (I - W)$$

The optimal embedding is found by computing the bottom $d + 1$ eigenvectors of M. Discard the bottom eigenvector which is the unit vector. The remaining d eigenvectors are the d embedding coordinates found by LLE.

Algorithm:

1. Compute the Neighbors of each data point, X_i.
2. Compute the weights W_{ij} that best reconstruct each data point X_i from its neighbors, minimizing the cost by constrained linear fits.
3. Compute the vectors Y_i best reconstructed by the weights W_{ij} minimizing the quadratic form by its bottom nonzero eigenvectors.

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

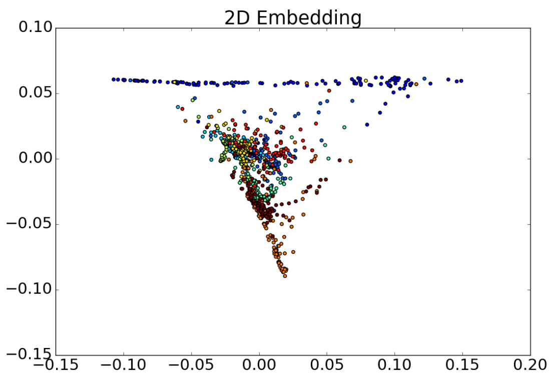
Investigate the influence of the choice of how many nearest neighbors you take into account. Additionally, try different metrics to find the nearest neighbors (we are dealing with images!).

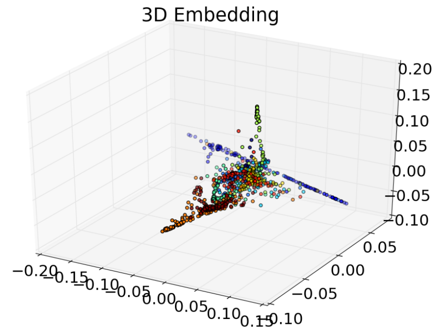
(e) Linear manifold interpolation

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?

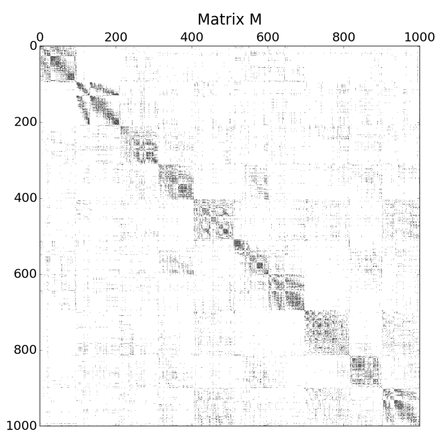
The Questions continued on next page. . .

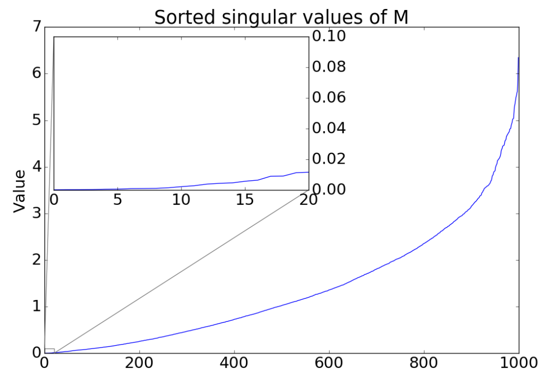
b) A description of the implementation is in the implementation section. The following two plots show the 2d and 3d embedding of 1000 samples with 10 neighbors. The colors correspond to the label of a data point. Clusters are visible, e.g. the blue and dark red points. In the 3d embedding the green data points form a very distinct cluster.





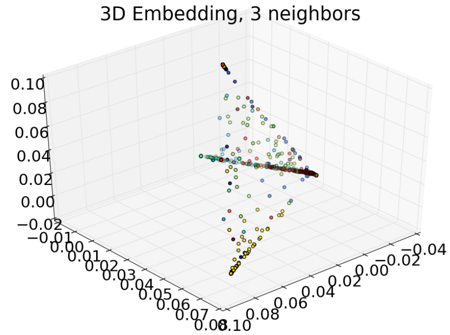
c) The matrix M contains block structures which appear when the samples are sorted by their label. The neighbors of a data point are very likely to have the same label. This is why these block structures appear.

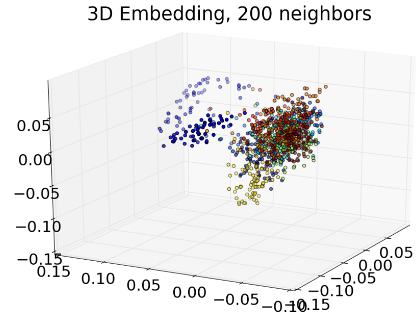




Since we want to minimize the cost we are looking for the eigenvectors with the lowest eigenvalues. The first 5 to 10 singular values are close to zero. This makes sense since we have 10 different labels. A good choice for the embedding dimension would therefore be 10 (one coordinate per label).

d) With more neighbors the manifold is smoother. The following two figures illustrate this:





I implemented another way to determine the neighbors of a datapoint. Since we are dealing with images I use the structural similarity[a] on the smoothed images (Gaussian filter with $\sigma = 1$). The resulting matrix M is slightly better diagonalized (less neighbors with different labels). See appendix for figures.

e) As a first attempt I determined the closest neighbors of a point in the embedded space. I assigned weights to these neighbors, so that the linear combination results in the point we want to map into high dimensional space (number of neighbors equals the dimension of the embedded space). Then I simply computed the weighted average of the high dimensional neighbors. But this method does only work if the point is very close to an already existing one. See gif at /plots/interpolation2d.gif.

[a] https://en.wikipedia.org/wiki/Structural_similarity

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

Branch name 12-915-302/1_locally_linear_embedding
The code is located at ./code/locally_linear_embedding.py.

The MNIST loading code is taken from
https://github.com/sorki/python-mnist/blob/master/mnist/loader.py.

The implementation of the LLE algorithm was done mostly from scratch with occasional comparison to the implementation of the sci-kit learn module
(https://github.com/scikit-learn/scikit-learn/blob/14031f6/sklearn/manifold/locally_linear.py#L508)
and the Matlab code provided here: https://www.cs.nyu.edu/ roweis/lle/code.html.

The implementation works with a sparse csr matrix and uses the arpack `eigsh` function to solve for the eigenvectors. I never encountered performance problems with this implementation because I only use the first 1000 samples of the training data. If I used more samples the visualizations become cluttered. Numerical instabilities are prevented by adding a small value to the main diagonal of the covariance matrix (as proposed in the paper, appendix A equation 6).
The hardest part of the implementation was at first understanding which steps were actually required for the algorithm and then to have the correct shapes, dimensions and transpositions for the matrices and vectors.

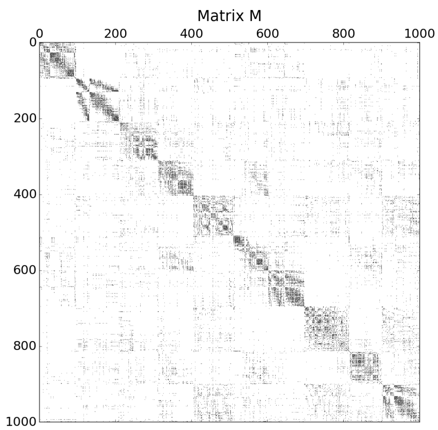
Your Page

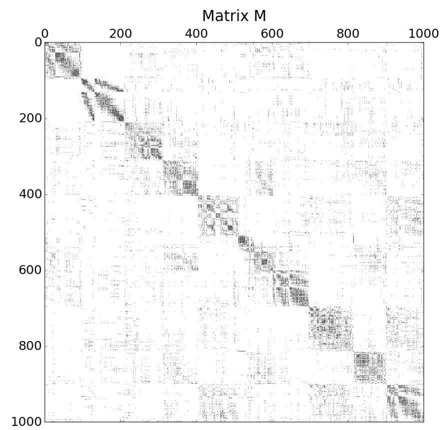
Your page gives you space to include ideas, observations and results which do not fall into the categories provided by us. You can also use it as an appendix to include things which did not have space in the other sections.
No page limit.

12-915-302/1_locally_linear_embedding

Comparison of different neighbor methods

The matrix on the left was constructed with neighbors determined by Euclidean distance. The matrix on the right by the structural similarity (SSIM) of the images. SSIM leads to a slightly better diagonalized matrix (check column 400 to 500).





The resulting embeddings are similar, except for a rotation. (Left Euclidean distance, right SSIM)