SLT coding exercise #1

# Locally Linear Embedding

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

Due on Monday, March 6th, 2017

Dario Fuoli
07-909-773

# 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 idea is to reduce the dimension of the data points, while keeping their relations they have according to some metric (KNN distance) in the higher space, also in the lower space. The following equations need to be satisfied/minimized:

$$E(W) = \sum_i \|x_i - \sum_j w_{ij} x_j\|^2$$

$$s.t. \sum_j w_{ij} = 1$$

Minimize the Error $E(W)$ for $w_{ij}$ under the constraint for the weights $w_{ij}$. The vectors $x_j$ are the KNN's of data point $x_i$. Weights that not belong to KNN's are set to 0.

$$E(Y) = \sum_i \|y_i - \sum_j w_{ij} y_j\|^2$$

With the fixed weights the new coordinates for the lower dimensional data points need to be found. Minimize the error $E(Y)$ for the data points $y_i$ and their KNN's.

# 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!).

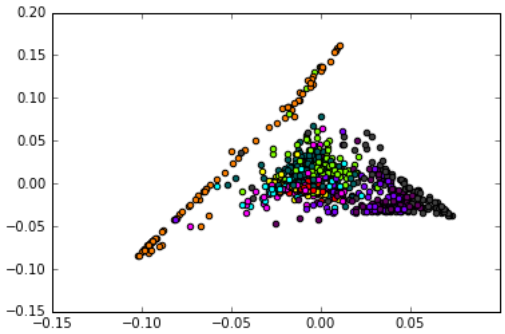
## (e) Linear manifold interpolation

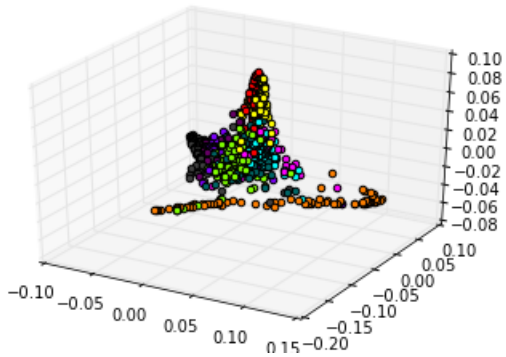
Assume you pick some point in the embedding space. How can you map it back to the original (high dimensional) space? Investigate how well this works for points within and outside the manifold (does it depend on the dimensionality of the embedding space?) Try things like linearly interpolating between 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... 

**(a)** The mnist images are downloaded and put into numpy array format. Then I use pickle to load them directly in numpy format.

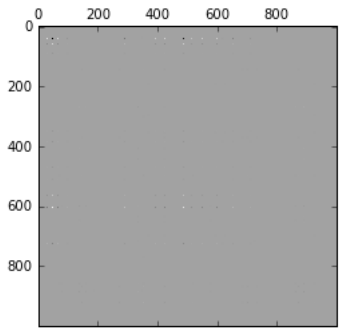
**(b)** 2d/3d: Subset of 500 images (performance). KNN's=10.

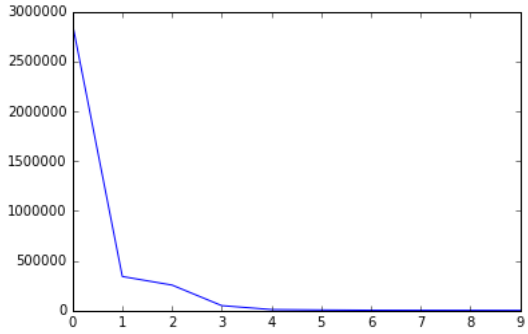



It is possible to see clusters. However, the clusters are not seperated too well. I suspect that there might be a better similarity metric for the KNN's that are better suited for images.

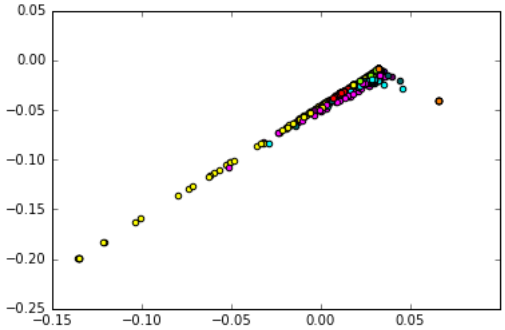
**(c)** There are some "block structures" in the matrix M: around the points (0,0), (0,600), (0,400-600), (600,500). There are 3 singular values that are considerably larger than all others. Because we want to find the d (d: dimensions of lower space) smallest eigenvectors to represent the points in the lower space this is a good thing; most information can be represented in 3 dimensions. For that reason set d=#(largest singular values that are considerably larger than all the others)

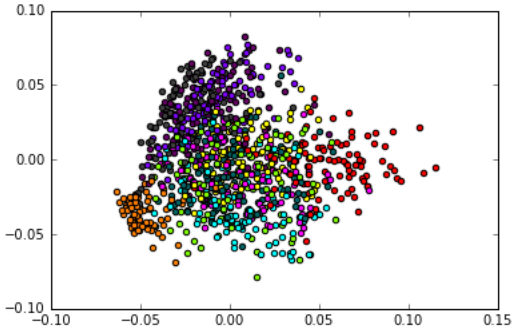



**(d)**

The left plot shows 3 KNN's, the right shows 999 KNN's. 1000 data points are used. I also plotted several k nearest neighbours inbetween and could observe the following: As we increase the number of neighbours the "cloud" grows and seperation of data points is better. For more neighbours it is easier to represent the data point $x_i$ as a linear combination of its neighbours and the error $E(W)$ can be further minimized. The data points are related to more neighbours and can therefore preserve the local structure more precisely.

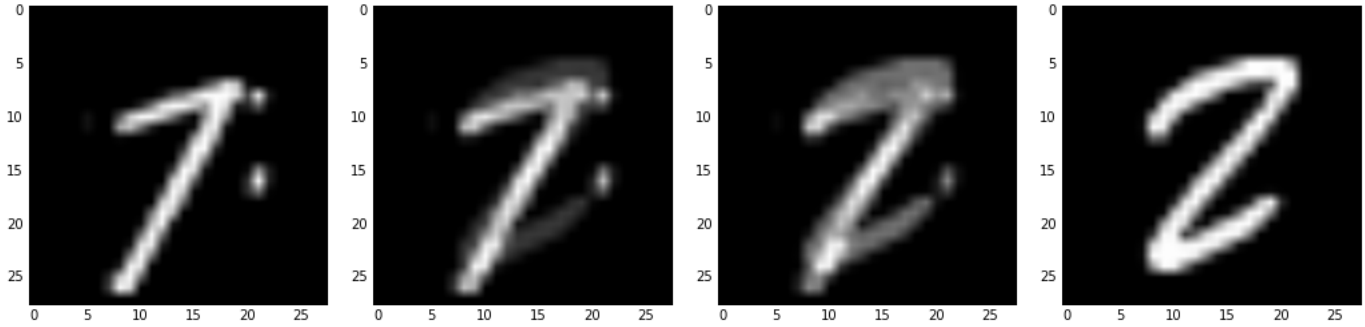



I tried different metrics, but couldn't see prominent changes in the clusters.

The Questions continued on next page...

**(e)** To map back a point from the embedding space to the original space, we need to find the KNN's of that point in the embedding space. Then use the corresponding weights from these KNN's and calculate the original $x_i$ as linear combination of the weights and the corresponding original $x_j$'s. The higher the embedding space the better this reconstruction should be, because we discard fewer information. However when the manifold's dimension is equal to the embedding's dimension, higher embedding dimensions don't contribute much more precision to the recondstruction (the manifold can be "rolled out" almost perfectly and thus keep relations between neighbours in the original space also in the embedding space). If we interpolate between two data points in the embedding space, the reconstructed images fade into eachother, if the embedding space has enough dimensions. Otherwise there might be other numbers which should be far away "popping up" in the interpolation. Here is an interpolation in the original space from 1 to 2:



## 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

GIT BRANCH: 07-909-773/1_locally_linear_embedding
I mainly used sklearn, pyplot and numpy/scipy. There were efficiency problems when I tried to use too many images, so I restricted myself to around 1000 images (or less) to be able to test and evaluate my functions. These were enough images to see all the important effects.
I had difficulties to make my implementation perform as good as the sklearn implementation. I suspect there are problems with the eigenvalue decomposition of the sparse matrix M.