SLT coding exercise #1

Locally Linear Embedding

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

Due on Monday, March 6th, 2017

Marc Fischer
13-943-568

Contents

The Model **3**

The Questions **4**
LLE and Neighborhood (a, b, d) 4
Effects of different K (d) 4
Mutual information as similarity measure 4
Cluster structure (c) 4
Linear manifold interpolation (e) 5

The Implementation **6**
Nearest Neighbors 6
Finding weights 6
Finding embedding 6

Your Page **7**
t-SNE and PCA 7
Appendix 7

The Model

Input

- data $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathbb{R}^D$
- target dimension d
- some distance metric, along with integer K or radius r, which determines the size of the neighborhood

Output

- $Y = \{\mathbf{y}_i\}$ such that $\mathbf{x}_i$ is a d-Dimensional representation of x_i

Algorithm and equations Given high dimensional data $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathbb{R}^D$, we want to represent it in a low dimensional space $\mathbb{R}^d$, $d \ll D$. The Local Linear Embedding algorithm accomplishes this in 3 steps.

1. For each $\mathbf{x}_i$ compute the neighborhood $N(\mathbf{X}_i)$. Either find the K nearest $\mathbf{x}_j$ or consider a fixed radius around $\mathbf{x}_i$. We will denote the neighborhood of $\mathbf{x}_i$ by $\mathcal{N}(\mathbf{x})_i$. And define it as

$$\mathcal{N}(\mathbf{x}_i) = \underset{\substack{j_1,\dots,j_k \\ \forall k \in [1,K] j_k \neq i \\ \forall k < l \in [1,K] j_k \neq j_l}}{\arg\min} \sum_{j_1,\dots,j_k} \|\mathbf{x}_i - \mathbf{x}_{j_k}\|. \tag{1}$$

   Other distance metrics can be used. I will investigate mutal information as such.

2. Find weights $\mathbf{W}$ such that the error in equation 2 is minimal. For a point $\mathbf{x}_i$ the weights w_{ij} describe how the vector $\mathbf{x}_i$ can be reconstructed from other vectors. For a certain j the weight w_{ij} indicates how much $\mathbf{x}_j$ contributes to this reconstruction. We also require that only for $j \in \mathcal{N}(\mathbf{x}_i)$ the weight w_{ij} is non-zero and that $\sum_j w_{ij} = 1$

$$\mathcal{E}(\mathbf{W}) = \sum_i \|\mathbf{x}_i - \sum_j w_{ij}\mathbf{x}_j\|^2 \tag{2}$$

   .

3. Compute the vectors $\mathbf{y}_i$ such that the following error term is minimized. The resulting $\mathbf{y}_i$ is a d-dimensional description of $\mathbf{x}_i$.

$$\mathcal{E}(\mathbf{y}_1, \dots, \mathbf{y}_N) = \sum_i \|\mathbf{y}_i - \sum_j w_{ij}\mathbf{y}_j\|^2 \tag{3}$$

The Questions

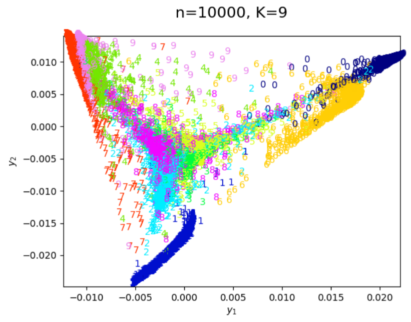
For the ease of writing I created my own section structure. In the parenthesis after the section name I indicate which of the original subproblems I address.

LLE and Neighborhood (a, b, d)

Figure 1 shows the clustering in 2D and 3D respectively. We can clearly see clusters for the different numbers. The best K (size of neighborhood) for 2D seems to be around 9 and for 3D around 8.

Effects of different K (d)

For higher K first there forms a cluster of 1s and a cluster of all other numbers. For even higher K they all merge into one cluster. The effect of the ones separating is likely due to the fact that they are the most different class to all others. The merging into one cluster for high K is probably due to the fact that the large neighborhoods often include numbers from different categories. Thus we need to describe most with few linear models. For lower K the numbers again lump into a single cluster. Probably because there weights carry too little information, to really form clusters. The effects of high and low K can be seen in Figure 9 on page 8.



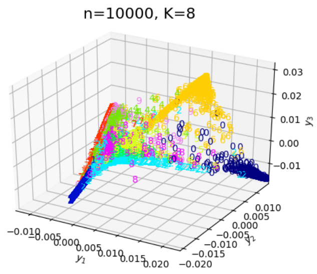


Figure 1: Visualizations of LLE in 2D and 3D for best K.

Mutual information as similarity measure

A quick search of literature and Google suggested that a common similarity metric for images is their mutual information. The probabilities for the mutual information are estimated by histograms. I also implemented LLE where there the K nearest neighbors with highest mutual information are used to form the neighborhood. Figure 2 visualizes the neighborhood defined by the mutual information.

Figure 4 shows the resulting LLEs. Again we see that for low K (around 4-5) we have somewhat well separated clusters. Results for higher K again merge into one big cluster. However around $K = 40$ they start to get better again (see Figure 10 on page 8). It should be noted that the computation of the mutual information is much more expensive than the the euclidean distances (which is why I show $n = 1000$ instead of $n = 10000$).

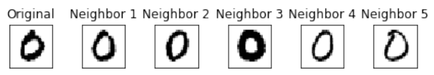
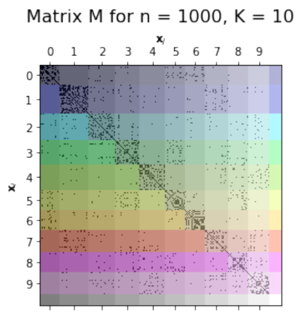


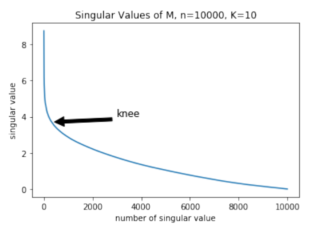
Figure 2: The $K = 5$ neighborhood of the image on the left, as measured by mutual information.

Cluster structure (c)

Figure 3a shows the structure of Matrix M, when the rows and columns are rearranged with respect to their class. We clearly see that there are diagonal blocks representing the clusters, but also a lot of off-diagonal noise. Those are just random interaction between the clusters, but we can also observe that they increase between similar looking numbers such as 0 and 6. For larger n these trends get harder to spot as clusters become smaller relative to the whole



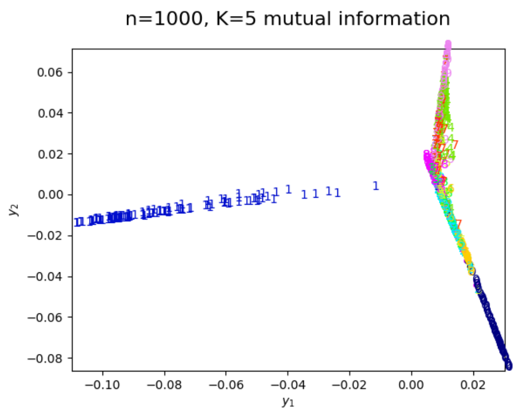
(a) Structure of matrix M.



(b) Singular values of the matrix M.

Figure 3: Investigation of matrix M.

matrix. This can be seen in Figure 11 on page 9. Figure 3b shows the singular values of the matrix M. We see a knee in the plot. Left of the knee eigenvalues are much larger than those to the right. An ideal dimension would be somewhere in that knee. This is similar to what one does in PCA.



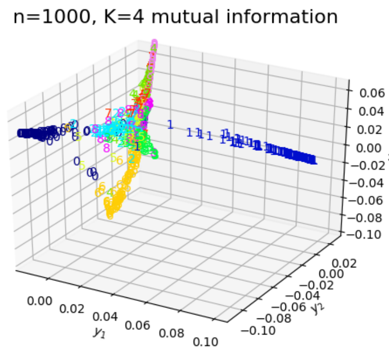


Figure 4: Visualizations of LLE in 2D and 3D using mutual information to measure similarity.

Linear manifold interpolation (e)

In order to map a point $\mathbf{y_i}$ from the embedding space back to the original space one:

1. finds the neighborhood $\mathcal{N}(\mathbf{y}_i)$
2. uses this neighborhoods to compute the weights $\mathbf{W}$ analogous to the weights computed in the high dimensional space when applying LLE
3. approximate the neighborhood in the high dimensional space $\mathcal{N}(\mathbf{x}_i) \approx \hat{\mathcal{N}}(\mathbf{x}_i) = \{\mathbf{x}_j \,|\, \mathbf{y}_j \in \mathcal{N}(\mathbf{y}_i)\}$ and use this to approximate $\mathbf{x}_i \approx \sum_{\mathbf{x}_j \in \hat{\mathcal{N}}(\mathbf{x}_i)} \mathbf{W}_{ij}\mathbf{x}_j$

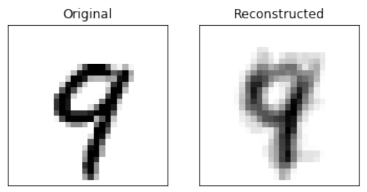


Figure 5: Reconstruction of a number.

Figure 6 features two lines. The blue lies within the manifold, the red without. The reconstruction of the blue line can be found in Figure 7a. The left most and right most image are the original data points. The second left and second right respectively are their reconstructions. The ones in the middle are the reconstructions of the interpolation. The reconstruction of the red line is in Figure 7b. Although it does not lie within the manifold, we see number like shapes. This is because the shape is determined by its closed neighbors.

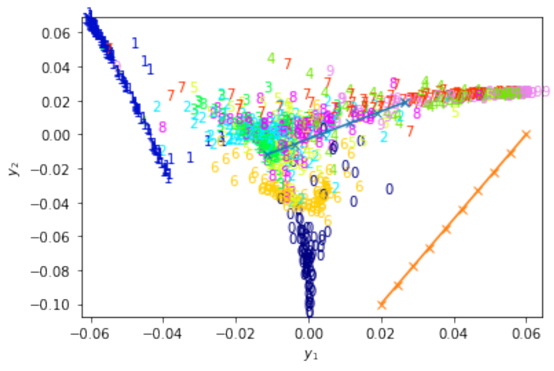
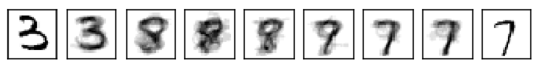
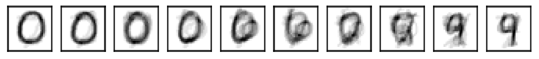


Figure 6: 2D Embedding showing 2 lines. The blue within the manifold, the red without.



(a) Reconstruction of the blue line in in Figure 6.



(b) Reconstruction of the red line in in Figure 6.

Figure 7: Reconstructions of lines within and without the manifold.

The Questions [Linear manifold interpolation (e)] continued on next page...

The Implementation

git branch: `https://gitlab.vis.ethz.ch/vwegmayr/slt-coding-exercises/tree/13-943-568/1_locally_linear_embedding`
implementation file: `https://gitlab.vis.ethz.ch/vwegmayr/slt-coding-exercises/blob/13-943-568/1_locally_linear_embedding/lle.ipynb`

Nearest Neighbors

In order to find the nearest neighbors given by Equation 1 I used a fast implemenation of a KD-Tree[1], which is a good data structure for such distance queries.

```
kdt = scipy.spatial.cKDTree(data)
_,q = kdt.query(data, K+1)
```

`q[i]` then holds the indices of the K nearest neighbors as well as i itself.
When I use mutual information as a similarity criterion the calculation of the nearest neighbors, based on nested loops, becomes much slower as the KD-Tree can't be used.

Finding weights

To find the weights, we want to minimize the error given by Equation 2. To do so I use the inverse local covariance matrices, as derived in exercise sheet 1 and suggested in the paper: $w_{ij} = \frac{\sum_k C_{jk}^{(i)-1}}{\sum_{lk} C_{lk}^{(i)-1}}$.
I suspected that the inversion might be numerically problematic. It turns out that for the normal MNIST dataset this is not a problem. I had a bug in my implementation which sometimes made my $C_{jk}^{(i)}$ singular. As suggested in the original paper I added a small diagonal component to the $C_{jk}^{(i)}$ matrices. This ensures that they are not singular. After resolving the bug this was not necessary anymore, but is still in the code (see line 5 below) as it has no noticeable impact on the solution.

```
1   W = np.zeros( (N,N) )
2   for i in xrange(N):
3       neighborhood = q[i][1:]
4       # calculate C = C^i_{jk}
5       C = C + (1.0/K) * np.eye(K)
6       C_inv = np.linalg.inv(C)
7       noms = np.sum(C_inv, axis=1) #row wise sum
8       denom = np.sum(C_inv)
9       w_ij = noms/denom
10      W[i, neighborhood] = w_ij
```

Finding embedding

To find the embedding vectors, we need to minimize Equation 3. As suggested in the paper I use the eigenvectors of $M = (\mathbb{I} - \mathbf{W})^T(\mathbb{I} - \mathbf{W})$. I started to compute them with `numpy.linalg.eig`, which worked but became very slow for larger matrices (larger numbers of data points). I switched to `scipy.linalg.eigh`, which allows to pass the range of the desired eigenvectors (i.e. the lowest $d+1$). Only computing the needed eigenvectors provided a sizable speed-up. However this computation is still the bottleneck. As mentioned in the paper a further improvement would be possible by by adapting the eigensolver so that matrix M never needs to be constructed. However I decided against this, as scipy already uses high performance BLAS underneath. Creating custom code with similar performance would have required a huge time investment.

```
e, v = scipy.linalg.eigh(M, eigvals=(0, d+1))
Y = v[:,1:(d+1)]
```

The current implementation runs suitably fast for $n = 10000$, taking around 500s.

[1]https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.spatial.cKDTree.html

Your Page

t-SNE and PCA

For another project in another lecture I currently use t-SNE, which also provides a very good embedding of MNIST (and high dimensional manifolds in general). Figure 8 shows t-SNE on this dataset. The actual t-SNE algorithm can be applied to data of any dimension. However, usually first a PCA to some medium number of dimensions (such as 30) is applied. I did the same for LLE. This gives a combination of two parameters to search over: The number of neighbors K and the number of PCA dimensions. Below a good choice is shown. In contrast to the normal LLE we see that the cluster of ones is much tighter for example. Likely because PCA removed a lot of noise within the cluster. This is also true (to an extend) for the other clusters.
I firmly believe that there are still better choices for PCA-dimensions, number of neighbors and maybe the neighborhood measure.

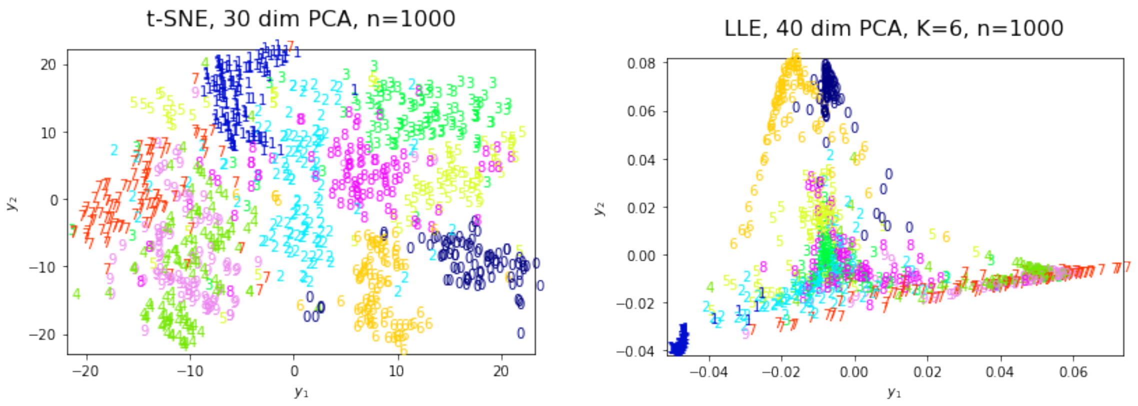


Figure 8: Visualizations of LLE and t-SNE in 2D.

Appendix

I use this section to show more plots and results that did not find into the 2 pages of the question section.

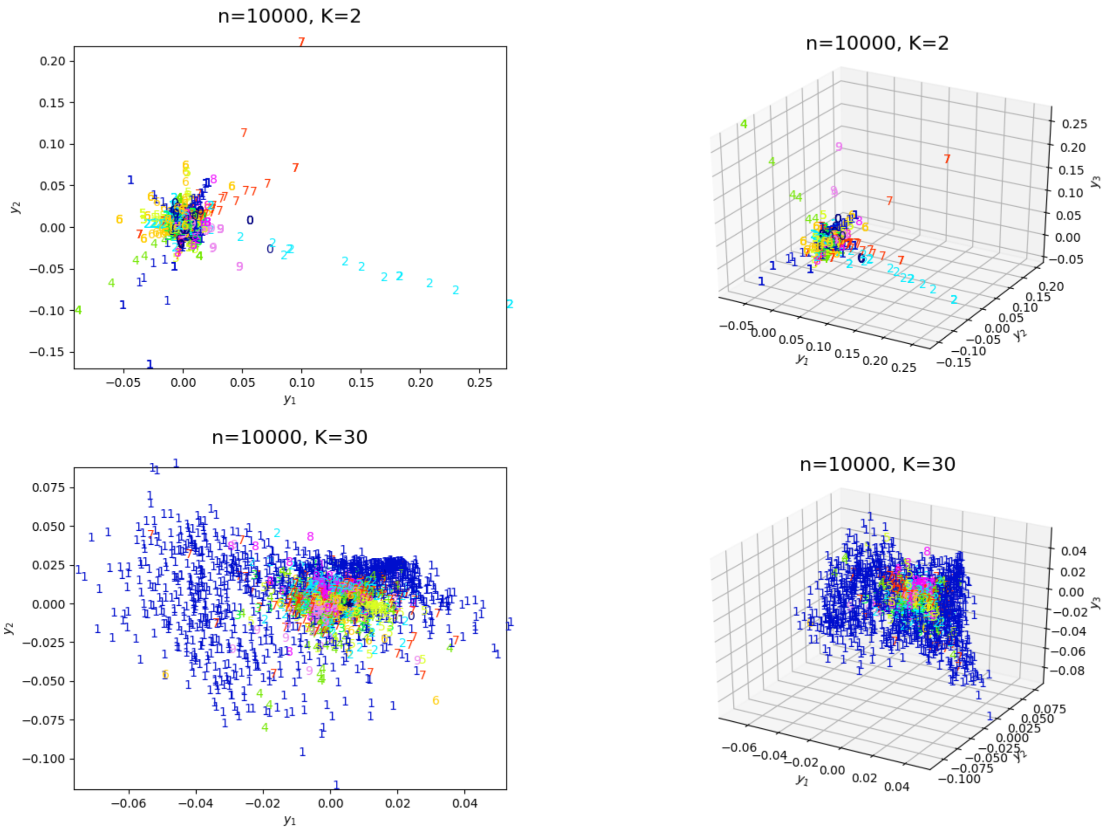


Figure 9: Visualizations of LLE in 2D and 3D for bad K.

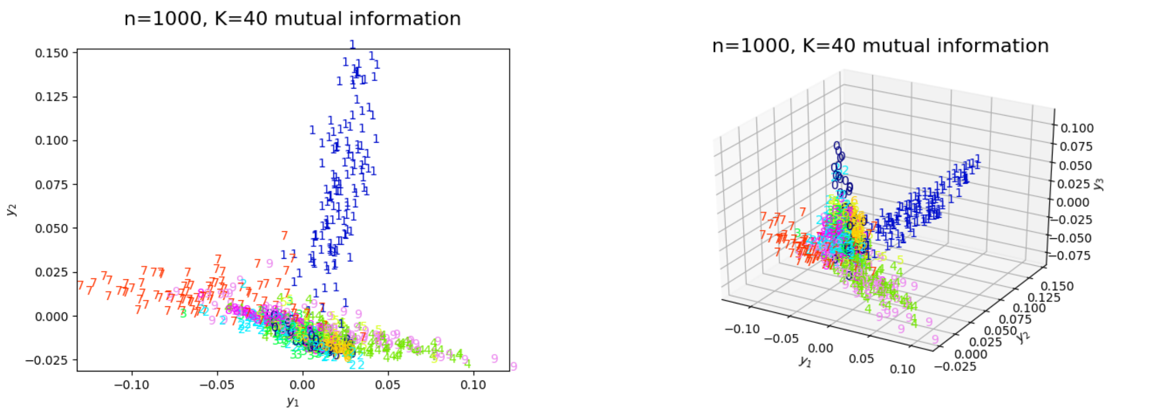


Figure 10: Visualizations of LLE in 2D and 3D using mutual information to measure similarity for $K = 40$.

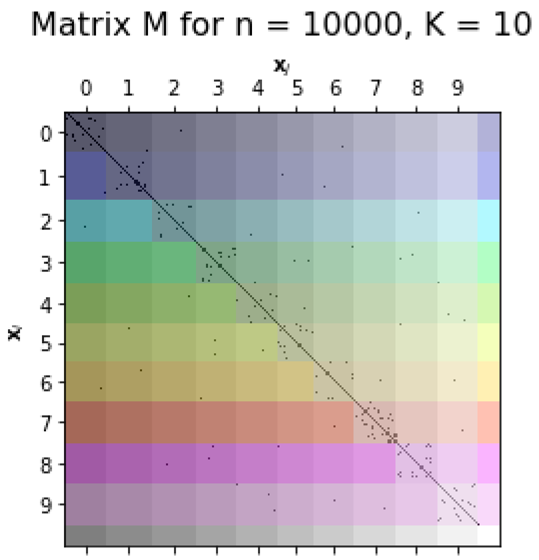


Figure 11: Structure of matrix M.

13-943-568/1_locally_linear_embedding