Hessian and Modified Locally Linear Embedding "A real world performance comparison"

Sven Borden
School of Engineering (STI)
École polytechnique fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: sven.borden@epfl.ch

Yves Martin
School of Engineering (STI)
École polytechnique fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: yves.martin@epfl.ch

***Abstract*—We have today different tools to tackle the problem of non linear dimensionality reduction. The idea is to find the meaningful low-dimensional structure hidden in high dimensional data. Two very similar algorithms, Modified Locally Linear Embedding and Hessian Locally Linear Embedding seems to perform well in theoretical cases, but how do they differentiate each other in real world datasets? We will apply a three parts method with ideal case, standard real case and a high dimension case to analyse and observe how they perform to reduce the dimensionality. Using each part of the method, we were able to demonstrate that in most case, Modified Locally Linear Embedding performs better than Hessian version of it. This results is underlying the fact that Modified Locally Linear Embedding is less sensitive to noise, dimentionality of the input data, and is easier to tune. On the other hand, Modified Locally Linear Embedding can perform better if specific requirements are satisfied, such as a low to medium input dimension.**

I. Introduction

Dimensionality reduction is concerned with the problem of mapping data points that lie on a low-dimensional manifold in a high-dimensional data space to a low dimensional embedding space. Traditional techniques have been extensively used for linear dimensionality reduction, we can name for example Principal Component Analysis (PCA) and Multidimensional Scaling (MDS). However, these methods are inadequate for embedding nonlinear manifolds. In the beginning of years 2000, some newly proposed methods, such as isometric feature mapping (Isomap) [1], locally linear embedding (LLE) [2] [3] and Laplacian eigenmap [4] have arousned a great deal of interest in non linear dimensionality reduction (NLDR) or in non linear manifold learning problems. Unlike previously proposed NLDR methods such as neural networks which require complex optimization techniques, these new NLDR methods enjoy the primary advantages of PCA and MDS as they still make use of simple linear algebra techniques that are easy to implement and are not prone to local minima. Despite the appealing properties of these new NLDR methods, they are not rebust against outliers in the data. More recently, two different version of locally linear embedding algorithm appears.

A. Locally Linear Embedding

Before going in specific case of modified or hessian Locally Linear Embedding, let's have a look on how the basic Locally linear embedding (LLE) works. It seeks a lower-dimensional projection of the data which would preserves distances within local neighborhoods. It can be thought of as a series of local PCA which are globally compared to find the best non linear embedding. It consists of 3 stages. The first one is the Nearest Neighbors Search, which is finding a set of the nearest neighbors of each point. It then computes a set of weights for each point that best describes the point as a linear combination of its neighbors. Finally, it uses an eigenvector-based optimization technique to find the low-dimensional embedding of points, such that each point is still described with the same linear combination of its neighbors. It's computational cost as shown on Equation 1 shows the three stages of computation. We consider N as the number of training data points, D as the input dimension, k as the number of nearest neighbors and d as the output dimension.

$$\mathcal{O}\left[Dlog(k)Nlog(N)\right] + \mathcal{O}\left[DNk^3\right] + \mathcal{O}\left[dN^2\right] \tag{1}$$

$$\mathcal{O}\left[Dlog(k)Nlog(N)\right] + \left(\mathcal{O}\left[DNk^3\right] + \mathcal{O}\left[N(k-D)k^2\right]\right) + \mathcal{O}\left[dN^2\right] \tag{2}$$

$$\mathcal{O}\left[Dlog(k)Nlog(N)\right] + \left(\mathcal{O}\left[DNk^3\right] + \mathcal{O}\left[Nd^6\right]\right) + \mathcal{O}\left[dN^2\right] \tag{3}$$

B. Modified Locally Linear Embedding

Modified Locally Linear Embedding (MLLE) correct one issue with LLE which is the regularization problem. When the number of neighbors is greater than the number of input dimensions, the matrix defining each local neighborhood is rank-deficient. To address this, LLE applies an arbitrary regularization parameter r, which is chosen relative to the trace of the local weight matrix. Though as $r \to 0$, the solution is not guarantee to converge to the desired embedding. To correct this MLLE use multiple weight vectors in each neighborhood. Only the second step of the LLE algorithm is changed, we see on its computational cost on Equation 2 that the central term has a higher complexity but if we look closely to the variable, we see that is most cases, the added cost of constructing the

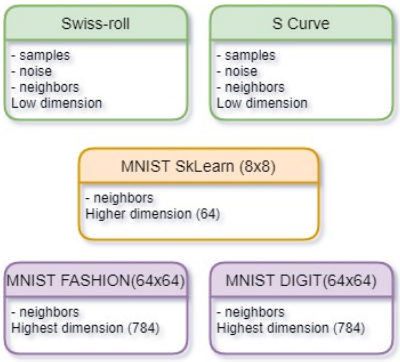

Fig. 1. This figure illustrate the three different step we used to compare the MLLE algorithm to the HLLE algorithm. In green, we gave the first step with the swiss-roll and the s curve. Both quasi ideal models with a low dimentionality from the beginning. In orange we have a higher dimentionality with the use of a real dataset, which is MNIST from SkLearn. Finally, we will use MNIST Fashion and MNIST digit datasets from OpenML to analyse the behavior of the algorithm with high dimension as input.

MLLE weight matrix is relatively small compared to the cost of stages 1 and 3 [5].

C. Hessian Locally Linear Embedding

Hessian Locally Linear Embedding (HLLE) is another method of solving the regularization problem of LLE. It revolves around a hessian based quadratic form at each neighborhood which is used to recover the locally linear structure. As for the MLLE, it only changes the way weights matrix is computed. On the Equation 3 we can see the second stage having an extra part from LLE, which reflects the QR decomposition of the local hessian estimator [6].

As those two algorithms seems very promising and it seems hard to distinguish witch of them is better given specific NLDR requirement, we will conduct a specific comparaison between the both of them.

II. Methods

To compare our algorithm, we go through different stages of comparaison. First, we will start to compare with Toy examples, such as the S Curve and the Swiss Roll. For each of them, we will see how well they performs in ideal situation but also in noisy environment. We will also be able to analyse how they perform with small number of sample or with a higher number of samples. Those Toy examples will be our testing case for low dimentionality reduction.
Secondly we will move the comparaison to real dataset with high dimensionality. We will first compare the two algorithm for a low dimensional MNIST dataset from SkLearn. Once done, we will move to a higher dimensional dataset, MNIST and MNIST-Fashion, both from OpenML. The Figure 1 resume the three different steps we will use to compare the two algorithms.

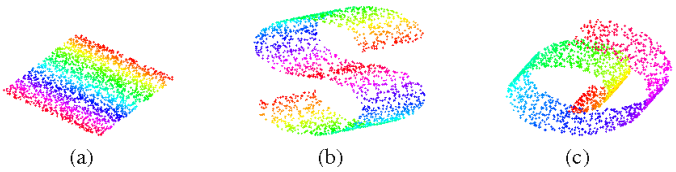

Fig. 2. Two examples of Toy Dataset that we used with the expected result. Both of them are a two dimensional rectangle that was shaped in a specific pattern in the 3D space. The one on the center is the Swiss Roll, it is the rectangle rolled over itself to produce this spiral shape. The one on the right is called S-curve, also coming from a rectangular shape, this one is folded to a S shape. It is expected from the algorithm to obtain the two dimensional shape of a rectangle. Both of them should produce the result on the left.

A. Parameters

We have different parameters we can change to analyse the differences between the two algorithms.

- **# of samples**: this parameter will allows us to compares if one algorithm needs a minimum amount of samples to performs and also if it can handle a large number of input in a reasonable time.
- **noise**: this parameter introduce noise in our ideal dataset. It correspond to the standard deviation of the points along the ideal shape.
- **# of neighbors**: this represents the number of neighbors used to compute each of the algorithm. We expect that one of them is more sensible to this parameter than the other.
- **# of dimensions**: this variable represent the complexity of the data input. We will start low with our toy dataset and increase it to a higher complexity to observe how each algorithm behaves.
- **# of components**: represents the number of components used to *project* our data in a lower dimension. This embedded space will always be equal to 2 as it is essential to show the results.

We will also have a look on the computation time of both algorithm.

B. Toy Dataset

To have a baseline of simple theoretical analysis and comparison between the two algorithm, we will use Toy dataset. Specifically the Swiss Roll and the S Curve as shown in Figure 2.

Those two datasets are ideal to compare how both of the algorithms performs with various parameters and with a tremendous control over the data.

1) Swiss-roll: The Swiss Roll, as shown on the middle of Figure 2 is constructed from a rectangle of random points in the 2D plane that we will *roll* to obtain the same rectangle but in a spiral shape in 3D space. We first compare both algorithms with an ideal Swiss-Roll, with no noise. We will select the best number of neighbors for different number of points.
Once done we will be able to do a second comparaison with a fixed number of points but with increasing noise in the

data. The noise will be represented by a standard deviation around the ideal shape of the Swiss Roll. We expect to see one algorithm to fail before the other.

2) S Curve: The S Curve, shown on the right of Figure 2 is also constructed from a rectangle of random points in the 2D plane. This rectangle is then shaped to a S form in the 3D space. As for the Swiss-roll, we will compare both algorithms with an ideal S Curve. The best number of neighbors will be estimated and in a second part, we will fix the number of points and increase the noise in the data to observe which algorithm handles the noise better. We also expect to see some differences with the results we will obtain between the Swiss-roll and the S Curve.

C. MNIST digits (8x8)

Once the Toy dataset explored and analysed, we will move to real dataset. As it often differs from the ideal cases. Also this will allows us to compare how each algorithm performs when we increase the number of dimension. We will use the dataset from SkLearn, which is a derivative from MNIST dataset. It consists of 1797 samples (around 180 per class) of digits between zeros and nine. Each digit is represented by an image of 8 pixels by 8 pixels. This represent 64 columns for each sample. We will run different parameter for each algorithm and used the embedded space of each algorithm to classify and get a score using a simple k-nearest neighbors algorithm (KNN).

D. MNIST fashion and digit (28x28)

As a third step mentioned in the Figure 1 takes MLLE and HLLE to the highest dimentionality of this report. We will use two different datasets, MNIST fashion and MNIST digit. Both from OpenML. MNIST fashion is a dataset of clothes and MNIST digit is a dataset of handwritten digits. Each of them has 10 classes but we will not always try to reduce the 10 classes. We will compare how effective the algorithms are to different number of classes. The particularity is that we have now pictures of 28 pixels by 28, which increase by a scale of 10 the number of dimension. We will also run a KNN algorithm after the reduction in the embedded space to get the performence of each algorithm.

III. Results

First thing noticeable before jumping to any result that the reader should pay attention is the fact that those algorithms are sensible to initialization. Sometime the initialization isn't good enough to get a good solution so it may useful to run each test different time to get the best solution.

A. Toy Dataset

We have seen on Figure 2 what the two algorithms are expected to output and we obtain exactly this result when we use 2000 points to represent each curves. Also both of the algorithm performed equally well here, we fixed the parameter of 20 neighbors to compute. If we increase the number of points without adding noise we observe that both of algorithm

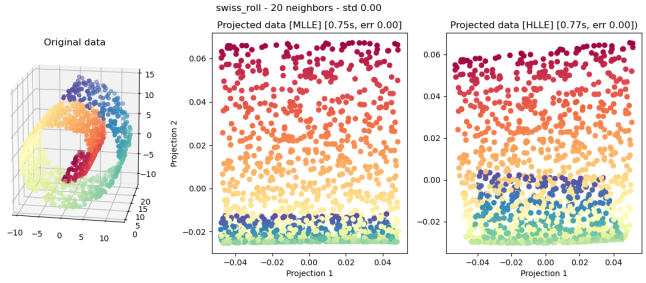

Fig. 3. Example of degradation of low dimensionality embedding when the number of points becomes too small. We have 1300 points along the shape of a Swiss Roll. We see that green and blue points are swapped, with a better performances from MLLE.

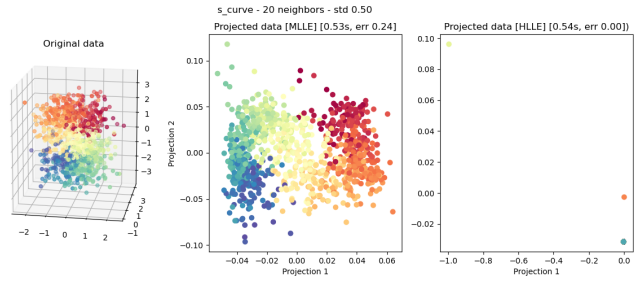

Fig. 4. Increasing noise around ideal S Curve shape. We have 1000 points with a standard deviation of 0.5. MLLE was able to create a low dimension space. HLLE on the right didn't found a space to represent the noisy data.

are behaving exactly the same. It becomes more interesting when we decrease the number of points from the Swiss Roll as shown in Figure 3. We used 1300 points instead of 2000 and we see that the low dimensional embedding has swapped for both algorithm the blue points and the green points. However we can see that the HLLE is performed poorer than MLLE in this case. Interestingly, if we apply a sample reduction to S Curve shape, even with extremely low numbers of points (100 points), both algorithm are able to performs it correctly, the Figure 12 in Appendix shows it.

If we vary the noise around the idea shape, we observe different reaction from MLLE and HLLE, one example is the S Curve, who seems giving more trouble, with a standard deviation of 0.5 from the ideal shape on Figure 4. We see that MLLE somehow found some low dimension space to project data as HLLE doesn't.

We can see on Figure 13 from the Appendix that the Swiss Roll is less sensible to noise as HLLE was still be able to find a acceptable plane, even if it performs poorer than MLLE.

We can now change the number of neighbors used to compute the reduction. If we increase this number as showed in Figure 5, we see that MLLE is performing perfectly and HLLE on the right is starting to mix blue points with orange points because too many neighbors were selected.

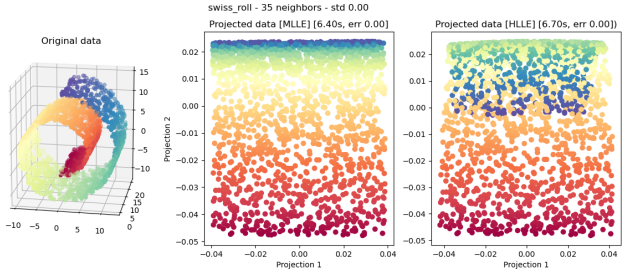

Fig. 5. Increasing the number of neighbors can change how well the reduction is performed. We have here 2000 points in a perfect Swiss Roll shape. We have for both algorithm 35 neighbors. We see the HLLE on the right doesn't correctly represent the rectangular shape and is mixing differents parts. MLLE on the other side behaves perfectly.

# classes	# neighbors (KNN)	MLLE Accuracy	HLLE Accuracy
10	3[MLLE], 7[HLLE]	82%	80%
7	4[MLLE], 5[HLLE]	96%	95%
5	3[MLLE], 4[HLLE]	95%	90%

TABLE I. Results of KNN classification after dimensionality reduction performed by MLLE and HLLE. We have varying classes from 10 classes of MNIST SkLearn dataset down to 5 classes. Classification performs the best for 7 classes in our case.

B. Medium dimensional dataset

We can now move to real dataset. For a medium dimensional dataset, we are using MNIST digits from SkLearn. It contains 1797 samples of handwritten digits of dimension 64. We have 10 classes going from 0 to 9. We are analysing how MLLE and HLLE perfoms to find an embedding structure for those digits. We can see on Figure 6 a dimensionality reduction. We perform different test with 5, 7 and 10 classes used and the even if it performs well for all the cases, we have a peak of performance for 7 classes, in fact we have more than 95% of accuracy with this model (Table I).The accuracy also drop when we are trying to reduce and classify the 10 digits. We can see in Figure 14 on the Appendix how well both algorithms performed with only 5 classes and Figure 15 shows results for 7 classes.

For all the results performed for the medium dataset, the hyper-parameter # of neighbors for MLLE and HLLE algorithm has be set to 25 as it was giving the best results. If we increase this number up to 30 or more, no reduction was correctly done and under 25, the dimensionality wasn't performing well too.

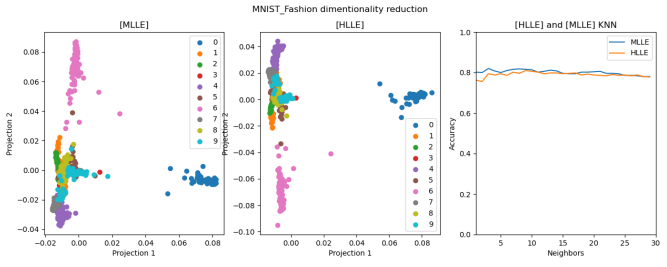

Fig. 6. Result of non linear dimensionality reduction of 10 digits from SkLearn dataset. The accuracy of the classification using KNN is 82% for MLLE and 80% for HLLE

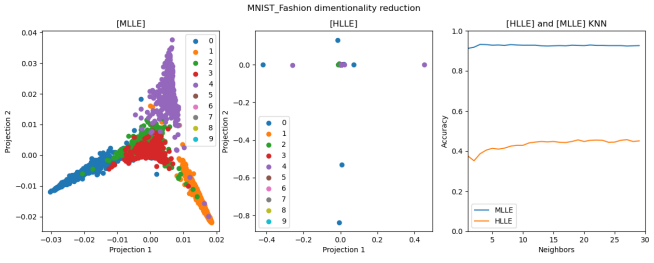

Fig. 7. Comparison between MLLE and HLLE with high dimensional data, MNIST of 784 dimensions with 25 neighbors to compute the dimensionality reduction. We used only 5 classes. HLLE wasn't able to find connection between points and fails to reduce data. Best score for MLLE is 92% and for HLLE score is less than 50%

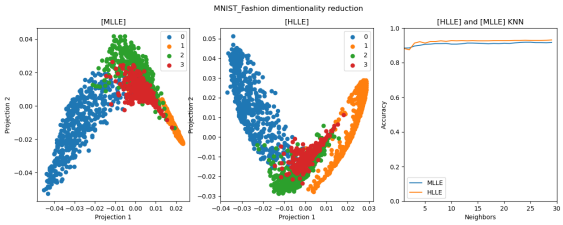

Fig. 8. Comparison between MLLE and HLLE with high dimensional data, MNIST of 784 dimensions with 150 neighbors to compute the dimensionality reduction. We used only 4 classes. Both of the algorithms are now able reduce data to a low dimension and allow KNN to performs well (91% for MLLE and 93% for HLLE). This is the first time HLLE is performing better than MLLE

C. Large dimensional dataset

With large dimensional dataset, we want to see how well MLLE and HLLE handle large dimensions. For this we used two datasets.

1) MNIST Digits: The first one is the same as the medium dimensional dataset analysed previously but instead of having 64 dimensions for each handwritten digit, we now have 784 dimensions. If we keep 25 neighbors to build our algorithm, we see on Figure 7 that as MLLE is capable to handle it at a good accuracy (93%), it is not the case for HLLE who wasn't able to perform the reduction.

If we increase the number of neighbors up to 150, we see on Figure 8 that this time MLLE and HLLE are capable of reducing dimentionality with in a good shape for KNN classification. This is only a classification between 4 classes of digits but we have for the first time HLLE (93%) which performs better than MLLE (91%). By increasing the number of neighbors we also increase the computation time needed to get results.

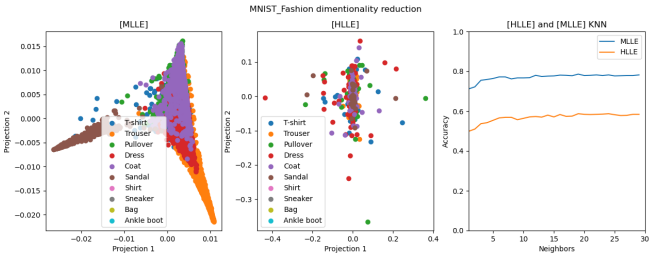

Fig. 9. Comparison between MLLE and HLLE with high dimensional data, MNIST Fashion of 784 dimensions with 25 neighbors to compute the dimensionality reduction. We used 6 classes. HLLE wasn't able to find connection between points and fails to reduce data. Best score for MLLE is 78% and for HLLE score is less than 60%

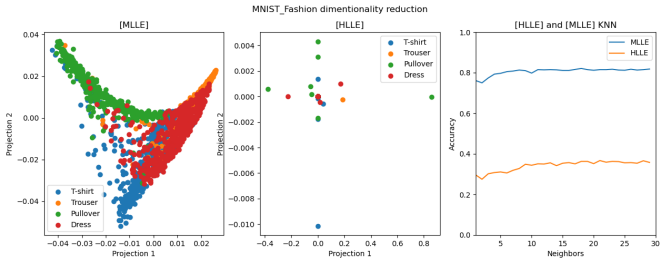

Fig. 10. Comparison between MLLE and HLLE with high dimensional data, MNIST Fashion of 784 dimensions with 150 neighbors to compute the dimensionality reduction. We used 4 classes. HLLE wasn't able to find connection between points and fails to reduce data. Best score for MLLE is 81% and for HLLE score is less than 40%

When we try to reduce the dimensionality and classify more digits, for examples having 9 classes of digits on Figure 16 from Appendix, we see that the performance of both the algorithms drop to around 50%.

2) MNIST Fashion: To compare our result with an other dataset, we used MNIST Fashion dataset from OpenML. This is also a high dimensional dataset with 784 features for each image. This time we also start with 25 neighbors for both the algorithms as we can see on Figure 9, once again, MLLE is able to performs correctly while HLLE fails to reduce data well enough.

We can also increase the number of neighbors to observe how HLLE will perform in this case. Figure 10 shows us that this time HLLE is clearly not able to reduce dimensionality correctly as visually it *exploded* and we see that KNN is not able to classify too.

For both MNIST Digits and MNIST Fashion, the results are slightly the same if we increase or decrease the number of training points. We used 5000 images for both tests.

D. Output dimension

As we can see in the Equation 2 and 3, the output dimension is a parameter that appears, mainly on HLLE algorithm. We ran a test with MNIST Fashion and 2000 samples with a number of neighbors set to 80 and an output dimension of 10.

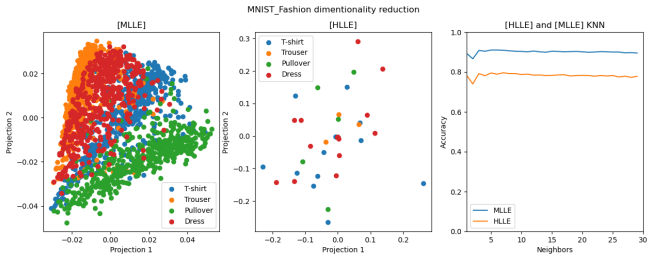

Fig. 11. This figure shows a test with an output dimension of 10. The figure on the left and the middle is a projection from those 10 components to a plotable 2 dimension. We can see specifically the KNN performing very well for MLLE (91%) and HLLE (79%)

The Figure 11 shows us that it doesn't change the computation time, but it improves the classification from KNN.

E. Computational time

The computational cost is very similar between the two algorithm. Mainly because we are using an output dimension of 2, and that the number of neighbors are very low compare to the number of points which make the second term of Equations 2 and 3 irrelevant.

IV. Discussion

We have now results for both, MLLE and HLLE in different situations, toy dataset but also real world datasets. We have sufficient information to discuss the usability of MLLE over HLLE.

A. MLLE

Let's first focus on MLLE algorithm. We have seen in the result section that it is performing very well in various situation. Let's break out each part.

1) Sensitivity to neighbors: From result section, we have seen that with 25 neighbors, MLLE was able to perform well on Toy Dataset, SkLearn MNIST dataset, but also in high dimensional dataset from OpenML. it seems very robust in every case, which makes very easy to tune the number of neighbors to obtain a result. Regardless of the number of neighbors it gives very good results for the KNN classifier. The computational cost from the Equation 2 seems to indicate a big sensitivity to the number of neighbors, but in reality the second term of the equation is must lower than the others so we cannot measure this in real case scenario.

2) Sensitivity to dimension: Passing from low, to high dimension was not easy and we have seen that MLLE wasn't very sensitive to the number of dimension it has to reduce. This is important for real case dataset as there is often lots of dimensions. However, we can see that MLLE is still struggling a bit with high dimension data, and that it works way better with a medium dimentionality such as SkLearn MNIST dataset. Also from the Equation 2 we can see that the input dimension is only affecting the computational cost in a linear manner, which is great to handle very high dimension.

3) Sensitivity to noise: We have played with the notion of noise with the Toy Dataset in the beginning of the result section. We have pushed the noise so far that the MLLE algorithm lost a bit of its performance but for most cases, it was very robust to the noise and outliers.

4) Sensitivity to # samples: We need a minimal number of samples in order to perform the reduction with MLLE, we used between 1000 and 10'000 data in our tests and the only sensitivity we could measure was the computation time. We see from Equation 2 that it is very dependant from the number of samples.

5) Sensitivity to output dimension: From the Equation 2 we can see that the output dimension is negligible. This has been tested once. We changed the output dimension to 10, and we see that the computation time is not changing much, because the output dimension will always be very small compare to the number of samples. So the sensitivity to output dimension is negligible. We also have seen on Figure 11 that is we increase the output dimension, the classification after the reduction obtain better score.

B. HLLE

We will now focus on HLLE algorithm. It seems poorer than MLLE from the result section we had before be we will still have a deeper look.

1) Sensitivity to neighbors: We have seen on result section that the number of neighbors is a sensible parameter. If not well chosen HLLE algorithm is not capable to reduce the dimentionality of our data. The number of necessary neighbors seems correlated to the number of dimension of the data. We observed that from medium dimension dataset to high dimension dataset, the required number of neighbors for HLLE went from 25 to 150, roughly it require one order of difference for a dataset roughly one order bigger (64 dimensions to 784 dimensions). This sensitivity makes HLLE not very robust to be applied to a variety of datasets.

2) Sensitivity to dimension: As discussed in the sensitivity of neighbors in HLLE, we have seen that when we increase significantly the number of dimension, it becomes harder to tune HLLE. The problem of having higher dimension is that the meaning of neighborhood fade out. When we have *exploding* behavior of this algorithm such as in Figure 7 or Figure 10, it appears the source can have two possibilities. The first one is that we have some duplicate in our dataset, which is not the case because we make sure all our data are unique. The second possibility is that the resolution of the matrix cannot be correctly done because some data seems uncorrelated from the point of view of HLLE. This seems coherent because the points we see are actually very small clusters of many points. The increasing dimensionality is then a very sensible parameter for HLLE.

3) Sensitivity to noise: As for MLLE, this algorithm seems robust to noise in the data, not as much as MLLE algorithm, but it is still performing well with the increasing noise.

4) Sensitivity to # samples: As for MLLE, we need a minimal number of samples in order to perform the reduction with HLLE, we used between 1000 and 10'000 data in our tests and the only sensitivity we could measure was the computation time. We see from Equation 3 that it is very dependant from the number of samples.

5) Sensitivity to output dimension: From the Equation 3 we can see that the output dimension is at the power 6. Which seems to make the output dimension a very sensitive parameter. As we run most of our test with an output dimension of 2, we couldn't measure that directly. This effect has been tested once. We changed the output dimension to 10, and we see that the computation time is not changing much, because the output dimension will always be very small compare to the number of samples. So the sensitivity to output dimension is negligible. We also have seen on Figure 11 that is we increase the output dimension, the classification after the reduction obtain better score.

C. Comparison

From what we discussed above, we can see that in most of the cases, MLLE seems to perform better than HLLE. Bit more importantly, it appears that MLLE is more robust and easier to tune than HLLE.

D. Running tests

If a reader want to experiment by itself or reproduce the data presented in this paper. The source code has been released to this link[1]. We provide in the appendix the explanation to run the code.

V. Conclusion

In this paper, we have proposed a methodology to analyse and compare two algorithms that performs non linear dimentionality reduction, Modified Locally Linear Embedding, and Hessian Locally Linear Embedding. We first compare the two with a toy dataset with ideal shapes and low dimensionality to observe how they behaves in ideal case, and be able to add controlled noise to the data in order to measure how robust MLLE and HLLE are to noise and sample augmentation.

We then moved to a higher input dimensionality with MNIST digits from SkLearn dataset, which consists of handswritten digits of 64 dimensions. This analysis allows us to understand how sensitive to different hyper-parameters MLLE and HLLE are.

Finally we tested both algorithms to a high dimension dataset, with 784 dimensions coming from two dataset, MNIST Fashion and MNIST Digits. We could then see the sensitivity of each algorithms with an increased dimension.

For each of the real world dataset, we used k-nearest-neighborhood classifier to evaluate the performances of each algorithm.

What came out is that in most of the cases, Modified Locally Linear Embedding performs better than Hessian Locally Linear Embedding. Most importantly, we can make

[1] https://github.com/dxs/MLLE-HLLE

the following statements. MLLE is more robust to noise in data than HLLE, it can handle more noise before losing the real link between points in the sample. MLLE is more robust to neighbors selection than HLLE, it is very easy to find values that give good results, but it appears that if tuned well, HLLE performs better than MLLE. Also generally, MLLE require less neighbors to identify the shape than HLLE. MLLE is also less sensitive to the number of input dimension, we have seen HLLE *exploding* and struggling to find links between points when using high dimensional dataset. Both of MLLE and HLLE algorithms are not sensitive to the number of samples used and the output dimension.

With all of those information, we can estimate that for a real world case, Modified Locally Linear Embedding with works better in most cases, but if the data are not in a very high dimension, HLLE can, with the trade of fine tuning, perform better.

VI. Appendix

A. Run the Code

Once cloned, the code consists of two main files, *"basic.py"* which is used to generate data on Swiss Roll and S Curve dataset, and *advanced.py* which takes care of generating data from various MNIST datasets.
basic.py file and *advanced.py* file can take different parameters:

- `--n_train` which is the number of training parameters
- `--n_test` which is the number of testing parameters
- `--neighbors` which is the number of neighbors used by MLLE and HLLE
- `--std_deviation_noise` which is the deviation noise from the ideal shape [ONLY BASIC.PY]
- `--data_structure` which can be $swiss_roll$ or s_curve for basic.py or $mnist_fashion$, $mnist_digit$, $mnist_digit_sk$ for advanced.py depending of the shape wanted
- `--filename` the name of the plot that will be saved on your disk
- `--solver` can be auto or dense, dense is more consistant
- `--n_class` the number of class (up to 10) applied when MNIST dataset selected

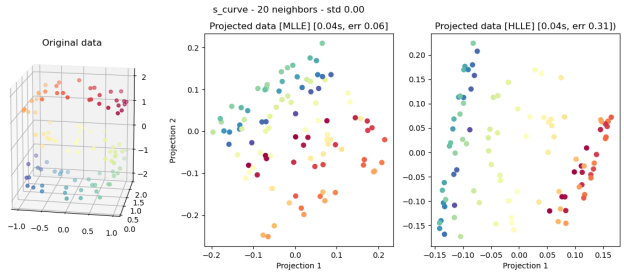

Fig. 12. Reducing the number of points with fixed neighbors (20) doesn't affect the S Curve as even with 100 points, the low dimensional shape can be correctly reconstructed

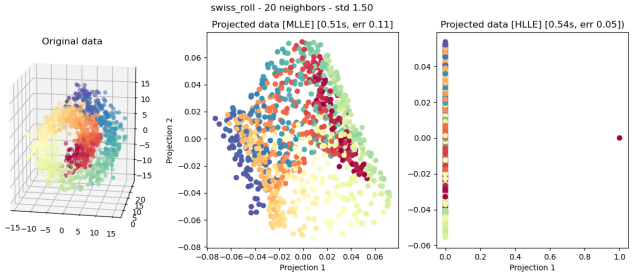

Fig. 13. Increasing noise around ideal Swiss Roll shape. We have 1000 points with a standard deviation of 1.5. MLLE was able to create a low dimension space. HLLE on the right didn't found an excellent space to represent the noisy data.

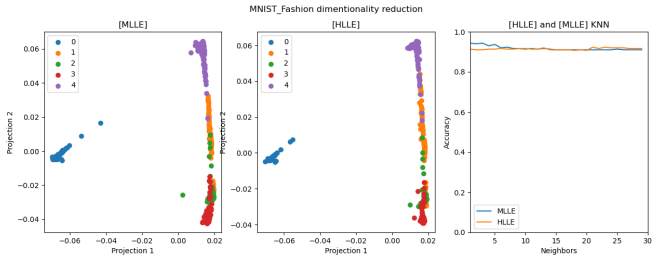

Fig. 14. Result of non linear dimensionality reduction of 5 digits from SkLearn dataset. The accuracy of the classification using KNN is 95% for MLLE and 90% for HLLE

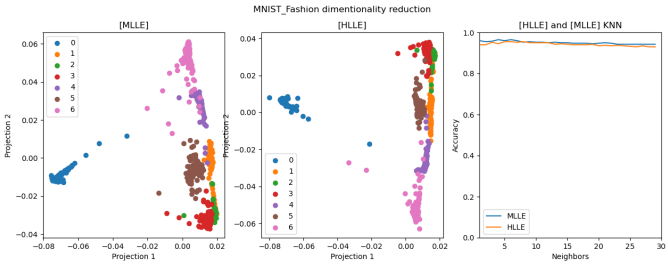

Fig. 15. Result of non linear dimensionality reduction of 7 digits from SkLearn dataset. The accuracy of the classification using KNN is 96% for MLLE and 95% for HLLE

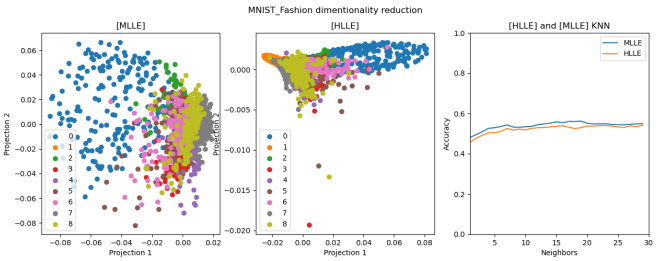

Fig. 16. Comparison between MLLE and HLLE with high dimensional data, MNIST of 784 dimensions with 150 neighbors to compute the dimensionality reduction. We used 9 classes. The reduction of the data is not very accurate as we have for both algorithm around 50% of accuracy from KNN

References

[1] V. de Silva J.B. Tenenbaum and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 2000.

[2] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 2000.

[3] S.T. Roweis and L.K. Saul. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 2003.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. in t.g. dietterich, s. becker, and z. ghahramani, editors, advances in neural information processing systems 14, pages 585–591. *MIT Press, Cambridge, MA, USA*, 2002.

[5] Zhenyue Zhang and Jing Wang. Mlle: Modified locally linear embedding using multiple weights. *Advances in neural information processing systems*, 2007.

[6] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 2003.