Implementation of t-SNE and its comparison with other Dimensionality Reduction Models

Team Kaaju
Aaradhya Gupta (2019114010)
Akhilesh Aravapalli (2019114016)
Chayan Kochar (2019114008)
Jayant Panwar (2019114013)

7 Dec, 2021

Github Repository: https://github.com/ChayanK2000/SMAI$_{project}$

Introduction

Dimensionality reduction refers to the process of reducing the number of input variables in a dataset. Dimensionality reduction is an important technique in the domain of machine learning. It helps in increasing the performance of all machine learning models by reducing the number of input variables in such a way that there is very little information. If data can be reduced to 2 dimensions, it can help in visualization of the data in a very clear and intuitive way. The goal of these algorithms is to reduce the dimensionality without losing any important information

In this project, we implement the t-SNE technique for dimensionality reduction, and demonstrate its effectiveness over other pre-existing dimensionality reduction techniques.

Dataset Preprocessing

Basic Overview

- Experimented with different datasets.
- Selected UCI Image dataset
- Basic preprocessing and preliminary dimensionality reduction

We decided to use UCI dataset over the MNIST dataset because of technological and temporal limitations. UCI images dataset is a relatively smaller dataset consisting of approximately 2000 images, of 8 * 8 dimensions. We loaded the dataset using the sklearn library. This gives us a list of vectors with 64 dimensions. First, we apply standard scaling on the dataset to improve performance. Since t-SNE takes a lot of time with higher dimensional datasets, we perform PCA component reduction on our dataset to reduce the number of dimensions to 30 before proceeding with other dimensionality reduction techniques.

Implementation of t-SNE

We have implemented a basic version of the t-SNE algorithm from scratch. We wrote it in a modular fashion keeping in mind the various calculations we needed to do for various parts in the algorithm. Thus we have made different functions which calculate: the neighbours for each point/sample. Some essential entities of our algorithm are as follows:

- calculate_neighbours
- $Q_{i|j}$
- $P_{i|j}$
- Kullback-Leibler Cost function

The paper associates P to the higher dimensional dataset and Q to the reduced version of the same dataset. The *calculate_neighbours* function returns the list of nearest neighbours acknowledging the values of perplexity and number of nearest neighbours, the functions Q_{ij} and P_{ij} were used to calculate the similarity between two points distributed on a local Gaussian curve. Kullback-Leibler algorithm was chosen as a cost function to implement gradient descent. After calculating the $P_{i|j}$ and $Q_{i|j}$, we then calculate the tables of P and Q which we use to store the values of probabilities of datapoints being a neighbour to one another. Finally, we perform the gradient descent on P and Q, which results in a list of 2-dimensional points that can be visualised.

Problem with initial t-SNE algorithm

While referring to the paper[2], we had implemented the basic version of t-SNE algorithm as stated above. Some insights and drawbacks of the code done till the mid evaluations observed are:

- The neighbours for any point to calculate the probability matrix, was calculated every time we need to access the neighbours. This contributed to a lot of complexity.
- A constant value, sigma(Gaussian variable) was used for all elements. Thus the given perplexity was not used in calculating the sigma for every element and so the whole concept of Shannon Entropy was unused. Due to this reason, there was some discrepancy in the plotting of the resultant 2-dimensional points
- All the mathematical equations and expressions were done from scratch, that is using nested for-loops, etc. Like we had made separate functions which calculate Q_{ij}, P_{ij} gradient descent, etc. Though this did help us to understand the equations in detail, it posed a greater problem in terms of time complexity.

Due to the reasons mentioned above, our code had exponential time complexity, and was not even able to handle more than 50 sample data points. We identified the major reasons for the huge time complexity :-

- The redundant and repetitive calling of function to get neighbours
- The usage of nested loops.

After identifying these major issues, we began optimization using linear algebra and utilizing the Numpy library.

Optimization of initial t-SNE algorithm

As stated above, there were some major reasons why the code was not performing well. Thus, the rectification of those were done and can be listed as follows:

- Firstly, a global dictionary of neighbours was created and repetitive calls of the neighbour function were discarded. We could now access neighbours of any elements in constant time after the overhead of the first time calculations. This greatly improved the complexity, though not up-to the par. This change alone, made our code efficient enough to run for 200 data points in under 12 seconds.

- Still, for the whole 1797 samples, it took way more time, around 25 minutes - showing how the code was exponentially complex, primarily due to the usage of nested for loops. This was overcome using numpy functions. Because overall a task executed in Numpy is around 5 to 100 times faster than lists and use of loops, which is a significant leap in terms of speed. Numpy is designed to be efficient with matrix operations. More specifically, most processing in Numpy is vectorized. Vectorization involves expressing mathematical operations, such as the multiplication we're using here, as occurring on entire arrays rather than their individual elements (as in our for-loop). With vectorization, the underlying code is parallelized such that the operation can be run on multiple array elements at once, rather than looping through them one at a time. Also using numpy gave a great amount of leverage. **So we discarded the 'neighbours' concept and used another NxN matrix D, with entry D_{ij} = negative squared euclidean distance between rows X_i and X_j.**

  For example the following code snippet calculates the squared distance in just two lines with the help of numpy. Also it is faster than the brute method using nested loops as discussed above.

```
def neg_squared_euc_dists(X):
    sum_X = np.sum(np.square(X), 1)
    D = np.add(np.add(-2 * np.dot(X, X.T), sum_X).T, sum_X)
    return -D
```

  Similarly, in the following function *calc_prob_matrix*, using numpy made it easier such that using this function we can calculate the probability matrix for the whole dataset as well as just for a point - when needed in predicting sigma:

```
def calc_prob_matrix(distances, sigmas, zero_index=None):
"""Convert a distances matrix to a matrix of probabilities."""

#makes the sigmas array from (num_points, ) to (num_points,1) for
#ease of calculations later
two_sig_sq = 2. * np.square(sigmas.reshape((-1, 1)))
X = distances/two_sig_sq
e_x = np.exp(X - np.max(X, axis=1).reshape([-1, 1]))

# We usually want diagonal probailities to be 0.
if zero_index is None:
    np.fill_diagonal(e_x, 0.)
else:
    e_x[:, zero_index] = 0.

# Add a tiny constant for stability of log we take later
e_x = e_x + 1e-8  # numerical stability
```

```
return e_x / e_x.sum(axis=1).reshape([-1, 1])
```

- The dynamic calculation of sigma for each element was implemented. It is unlikely that there is a single value of σ_i that is optimal for all datapoints in the dataset. This is because density of the data is likely to vary. In dense regions, a smaller value of σ_i is usually more appropriate than in sparser regions. Any particular value of σ_i induces a probability distribution, P_i , over all of the other datapoints. This distribution has an entropy which increases a σ_i increases. Hence we used **Binary search** to find the optimal value of σ_i such that produces P_i with the given Perplexity. This was done such that for each σ_i we get the probability matrix and thus the Shannon entropy:

$$H(P_i) = -\sum p_{j|i} \log_2 p_{j|i} \tag{1}$$

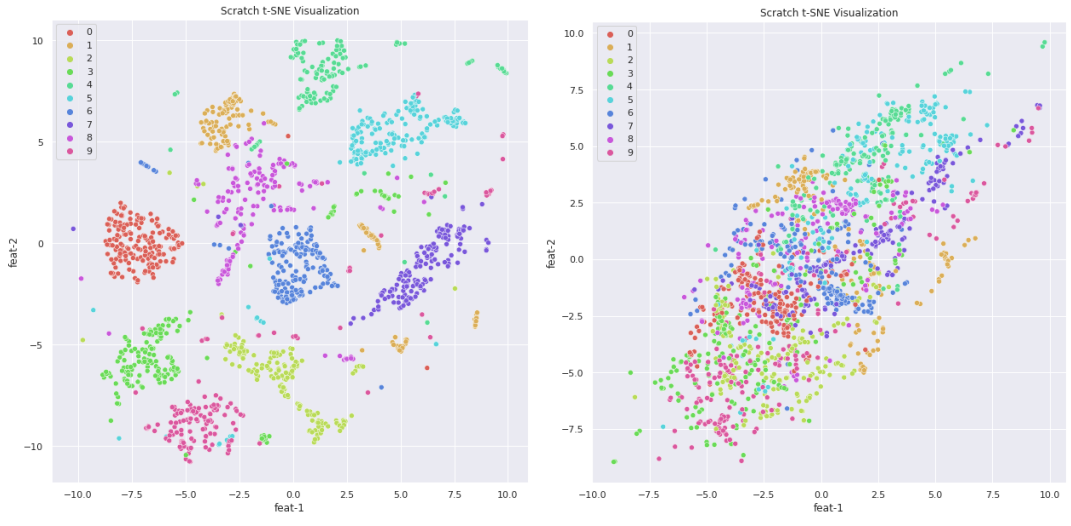
ultimately the Perplexity as :

$$Perp(P_i) = 2^{H(P_i)} \tag{2}$$

Hyperparameter Tuning

We spent some time with hyper-parameter tuning too, to get the best possible results for our t-SNE algorithm. We compared certain scenarios given the value of such parameters. The adjustable parameters include : Perplexity, Learning rate, Epochs and Momentum. To get comparable results, we seeded the randomness in numpy using np.random.seed(0) so that we compare on exactly similar environments.

Parameters for BASE:
PERPLEXITY = 50 MOMENTUM = 0.9
LEARNING_RATE = 10 EPOCHS = 500



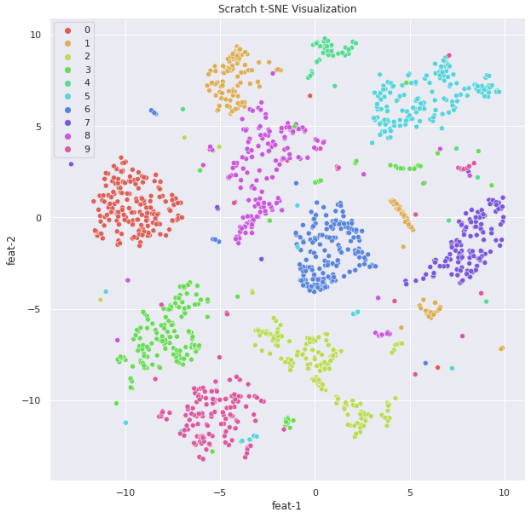


Figure 1: (a: BASE), (b: BASE with LR=1), (c: BASE with LR=50 EPOCH=200)

In the above 3 figures, we manipulated the value of Learning rate and no of EPOCHS, and such were the graph as observed.
The learning rate is a hyper-parameter that controls how much to change the model in each EPOCH. The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.
Here we observe that everything kept same, if we make the LEARNING RATE=1, the graph deteriorates. Whereas increasing LEARNING RATE to 50 poses not much harm, even though EPOCHS is decreased to 200. The main reason behind this is that the objective of the tSNE algorithm is thus to minimize the difference between the conditional probabilities in higher and lower dimensional space for the best representation of data points in lower-dimensional space. Small learning rate can make the difference so small that the cost function gets stuck in undesirable local minimum.

The following two figures compare the results with different values of PERPLEXITY:

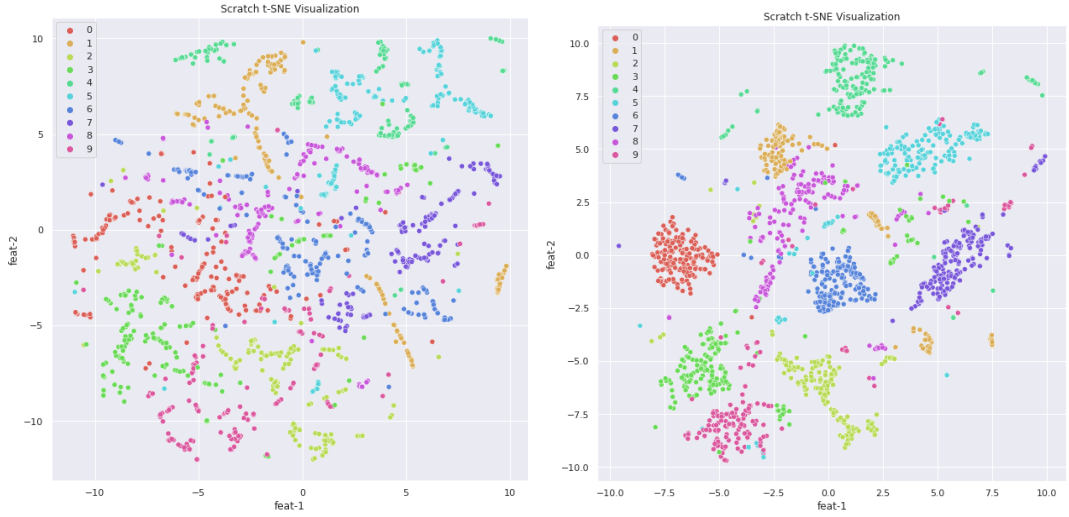


Figure 2: (a: BASE with PERP=10), (b: BASE with PERP=100)

In the above 2 figures, we see the effect the value of perplexity has on the scatter-plot generated by our algorithm. Higher perplexity values results in more distinctly separated clusters, while a lower perplexity results in a lot of intermixing of these clusters.
The parameter, "perplexity," signifies how to balance attention between local and global aspects of your data. The parameter is, in a sense, a guess about the number of close neighbors each point has. The perplexity value has a complex effect on the resulting pictures. To help develop an intuition, we can say that optimal number for perplexity can be considered as $N^{0.5}$, that is why we chose PERPLEXITY = 50 in BASE.

Other dimensionality reduction techniques

In order to compare our t-SNE algorithm developed from scratch, we consider various dimensionality reduction techniques. Some of them are linear while other happen to be non-linear. The major foundation of the comparison between the techniques is visualizing our dataset on a 2-Dimensional plot. This will help us to evaluate and determine which dimensionality reduction technique is able to retain the maximum information when reduced from 64 dimensions to 2 dimensions.

Principal Component Analysis

Principal Component Analysis, popularly known as PCA, is a linear dimensionality reduction technique. It reduces dimensions by projecting each data point only onto the first few principal components to form a lower-dimensional data while preserving as much variation of the original data as possible.

Multidimensional Scaling

Multidimensional Scaling, also known as MDS, is a linear dimensionality reduction technique in which the level of similarity of individual cases of a dataset is used to facilitate dimensionality reduction. It has various extensions. It can be converted into non linear dimensionality reduction technique by placing more importance on the small distances like in Isomap or Sammon Mapping.

Isomap

Isomap is a non-linear dimensionality reduction technique. To reduce the dimensions of the data, this technique estimates the intrinsic geometry of a data manifold based on a rough estimate of every data point's neighbors on the manifold. It is one of the extensions of Multidimensional Scaling (MDS).

Sammon Mapping

Sammon mapping[4] is a non linear dimensionality reduction technique which is an extension of the MDS technique discussed earlier. It lowers the dimensions of the given data by focusing on preserving the structure of inter-point distances in high-dimensional space in the lower dimension projection.

Locally Linear Embedding

Locally Linear Embedding (LLE) is a non linear dimensionality reduction technique. It reduces the dimensions of the data by trying to retain the geometric features of the original non linear feature structure. It has other extensions like Hessian LLE and Modified LLE but our focus is on the original LLE in our analysis.

Spectral Embedding

Spectral Embedding is a non linear dimensionality reduction technique. It tries to reduce the dimensions of the data by forming an affinity matrix and applying spectral decomposition to the corresponding graph Laplacian.

Results and Analysis

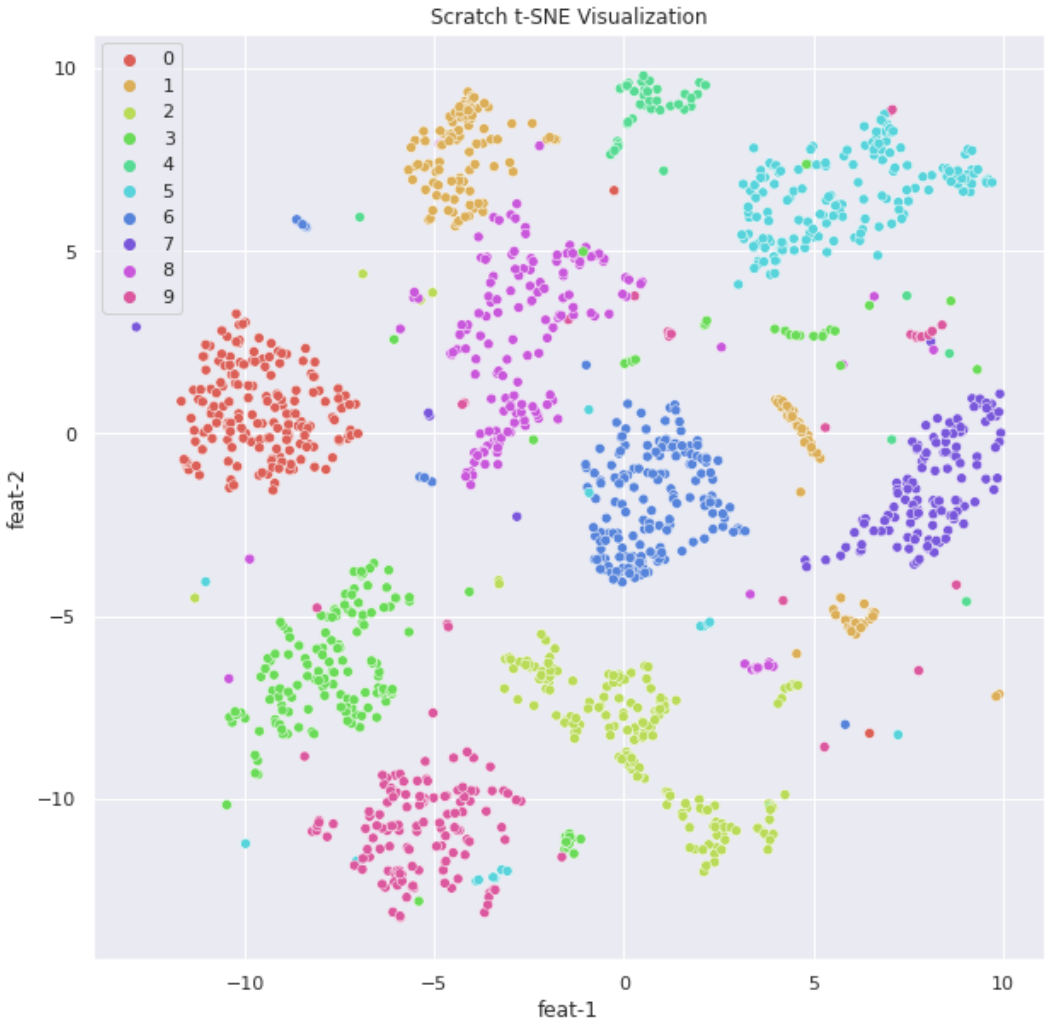


Figure 3: Scratch t-SNE Visualization

As visible from the plot in Figure-3, our t-SNE algorithm written from scratch does a decent job at visualizing the datasets. A detailed look at the plot reveals that most of the data points are clustered properly with the data points belonging to the same class. This serves as evidence of the proficiency of the algorithm and how well it is able retain the original information of higher dimension data in lower dimension as well.

However, the plot also contains some data points that are clustered with the incorrect class. The reason for some of these points corresponding to wrong class is the presence of distorted images present in the dataset. A distorted image can make a '2' seem similar to a '5'. Nonetheless t-SNE does a good job of revealing the natural classes present in the dataset.

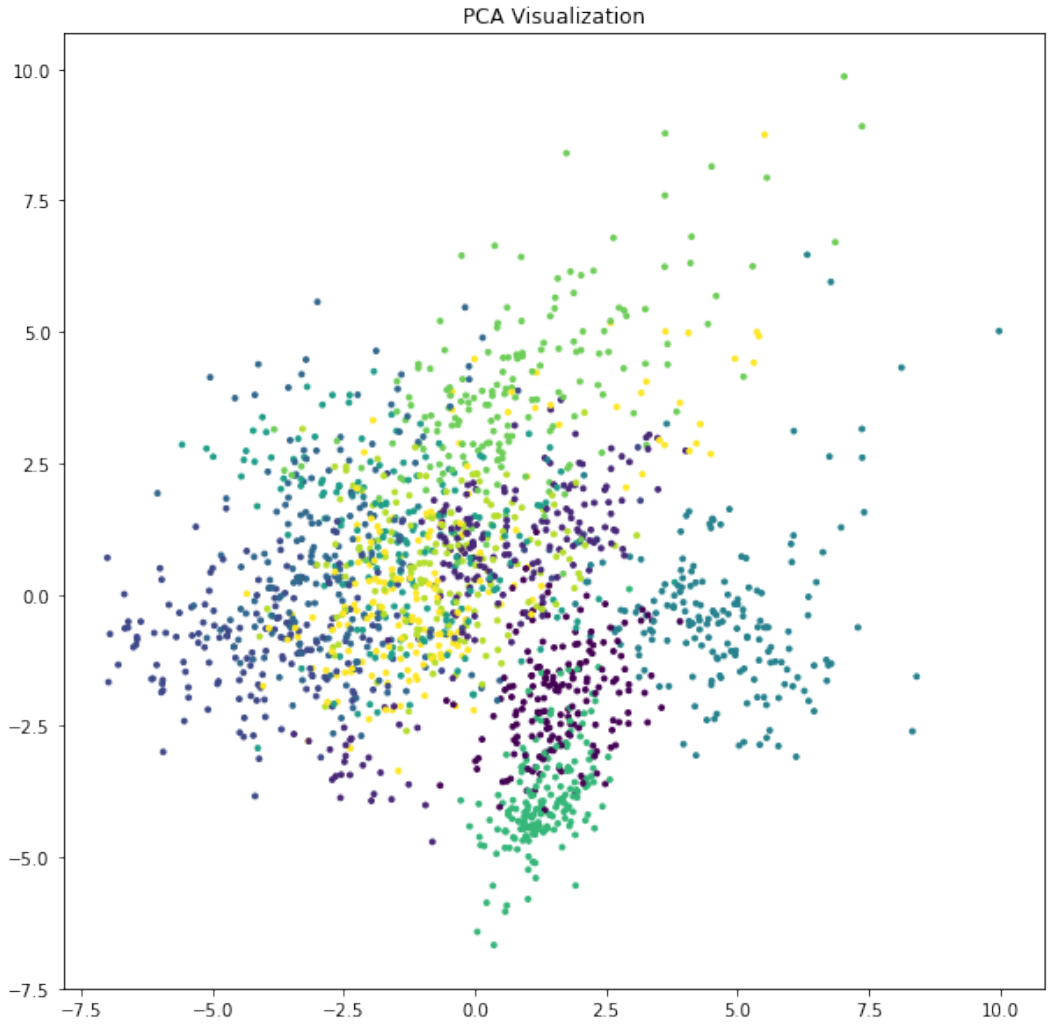


Figure 4: PCA Visualization

The plot in figure 4 clearly depicts that PCA does not do as well a job at visualizing the dataset as our scratch t-SNE did. All the clusters seem inter-mixed with one another. No clear distinction can be made amongst them.

PCA tries to preserve the Global structure of the data while not giving any importance to the local structure. So, when it is converting the data from higher dimensions to lower dimensions, it attempts to map all the clusters as a whole. This results in losing the local structures, which our scratch t-SNE algorithm is able to preserve. Another important drawback is that PCA is highly affected by outliers in the dataset whereas t-SNE is not affected so much.

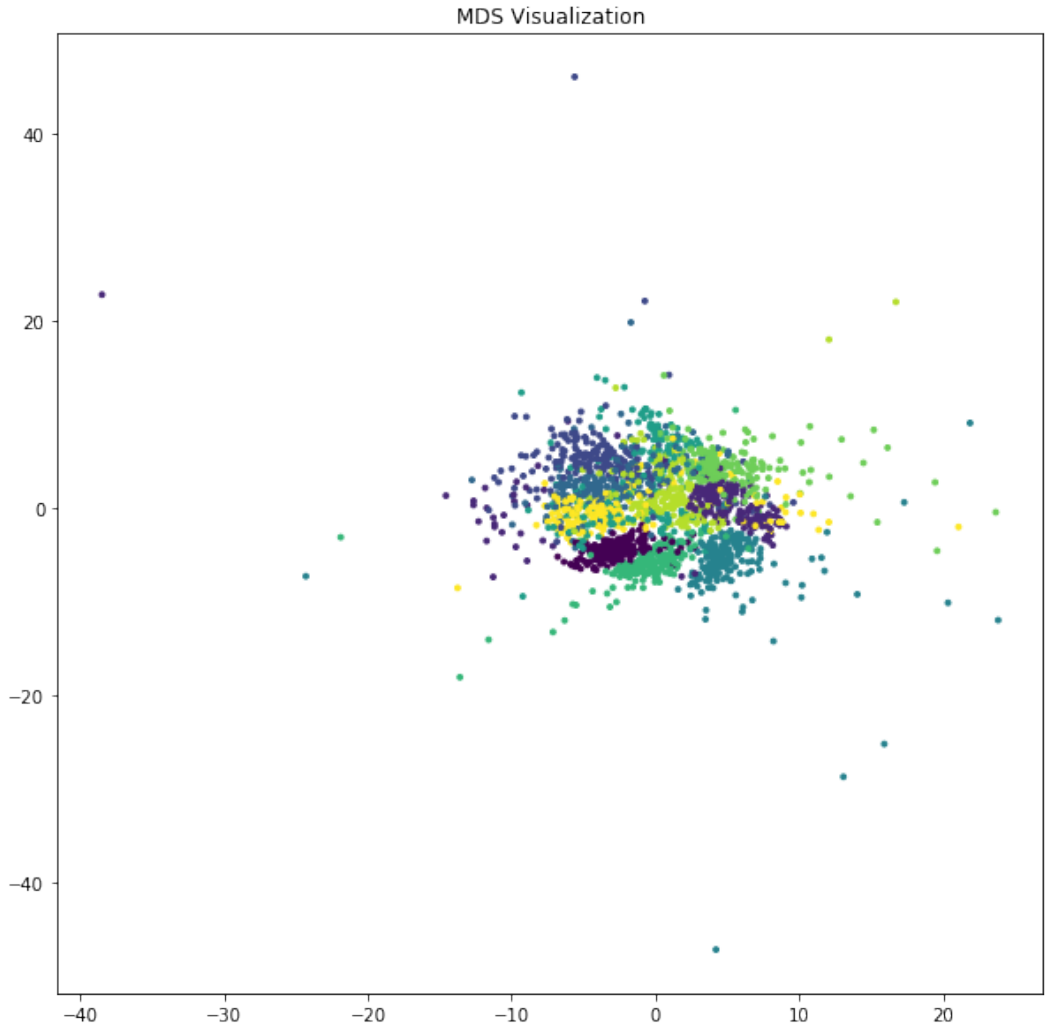


Figure 5: MDS Visualization

It is fair to say that the MDS plot does a fair job of visualization after seeing Figure 5. The clusters are inter-mixed but the distance between the data points is far less than what we saw in the PCA plot.

The reason for this not so clear visualization could be the usage of only one distance or dissimilarity matrix. Real-life datasets should be analyzed with multiple estimations of distance instead of focusing on only one. Moreover, the interval scale condition may not always be met in the data.

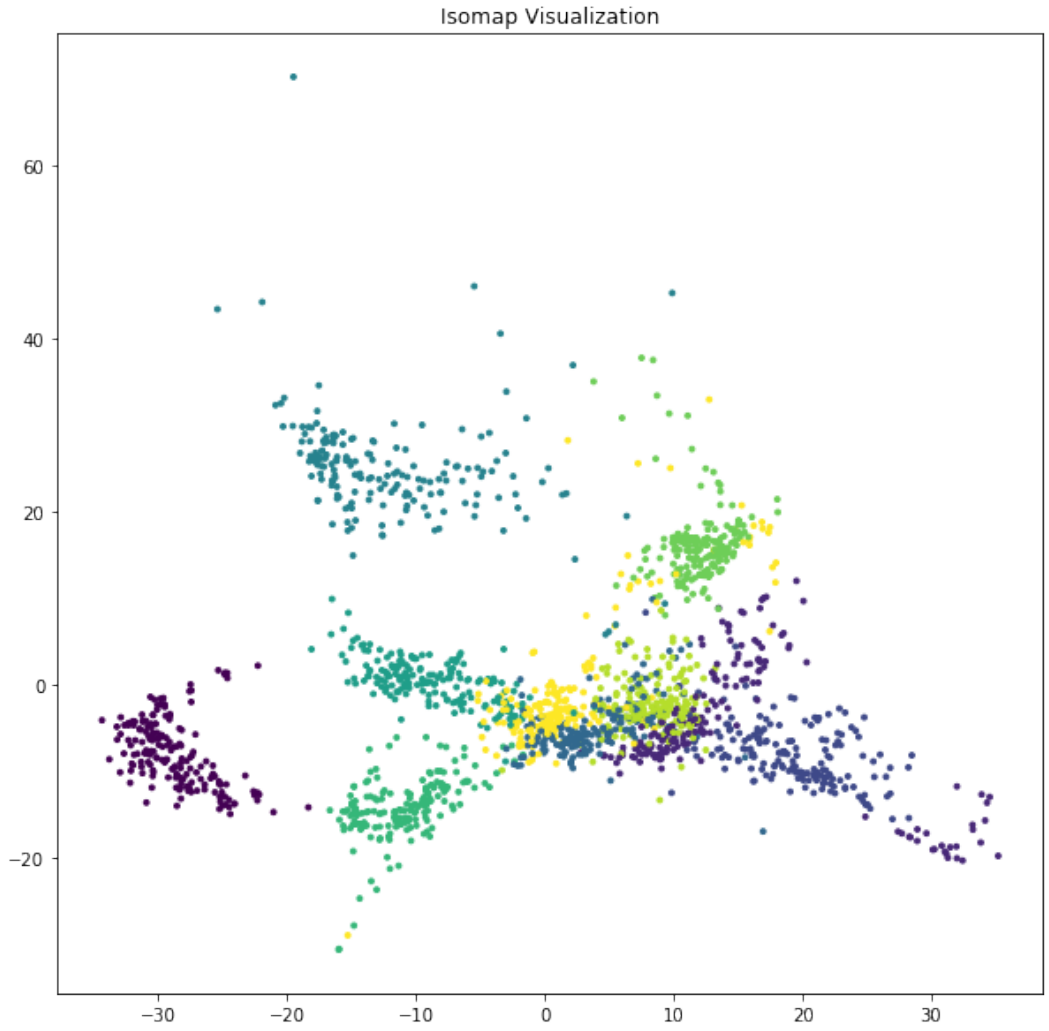


Figure 6: Isomap Visualization

We can observe that Isomap results in a much more scattered graph than t-SNE. The clusters formed from Isomap algorithm are also mixed up with one another not giving much importance to the local distances in clusters.

Better performance of t-SNE(Fig. 3) compared to Isomap technique(Fig. 6) can be understood by the problem of "short-circuiting" that the Isomap technique faces in some cases. The t-SNE technique understands nearest neighbour distances uniquely by integrating all the paths through the neighbours, whereas, the Isomap technique uses the geodesic distance(shortest path distance) for one neighbour to calculate the pairwise similarities. This difference adds up when the number of neighbours is too large, where the Isomap algorithm drastically fails to store local information of clusters when points are too sparse (k is large).

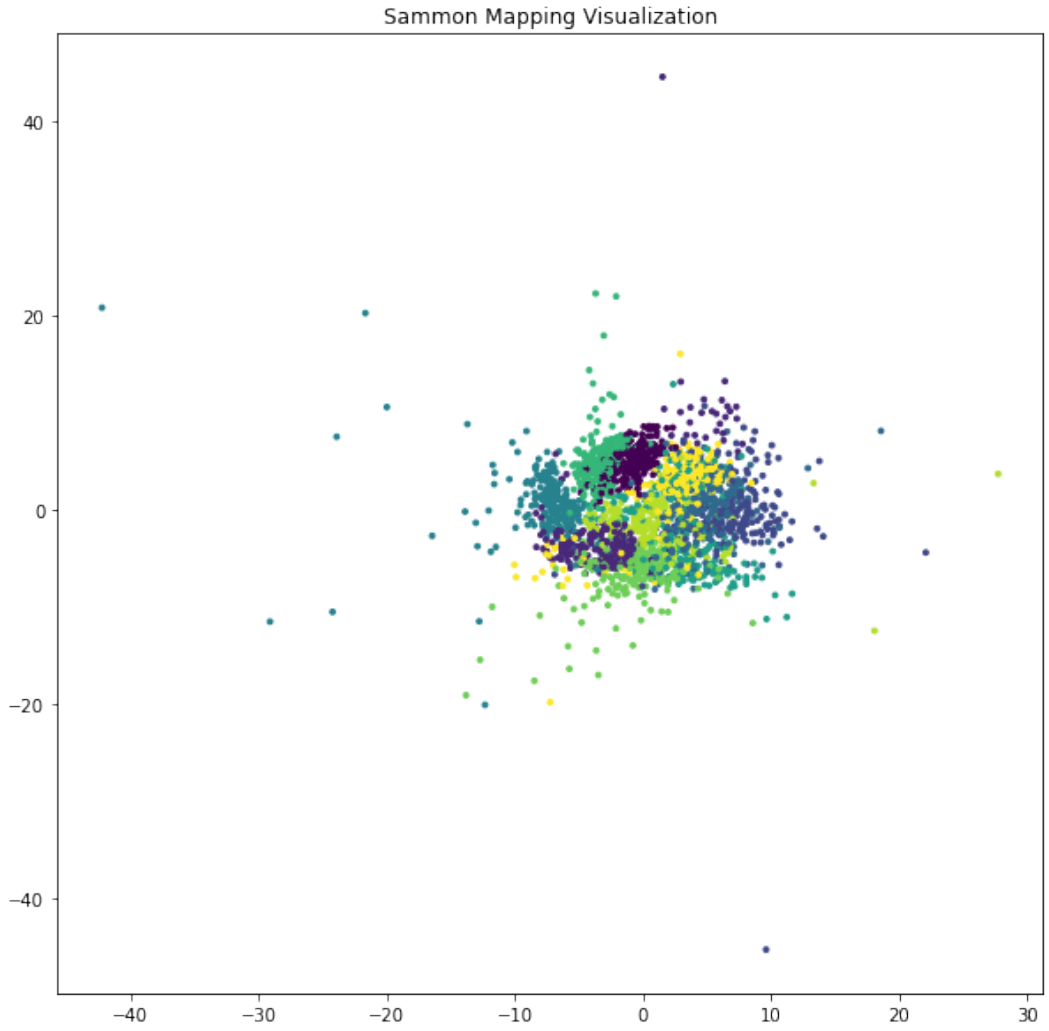


Figure 7: Sammon Mapping Visualization

If we take a better look at all the plots formed by different techniques, sammon mapping(Fig. 7) seems to stand is working very well placing itself just after t-SNE. The clusters formed by sammon mapping, are clearly visible and clearly separated, however, they still need to be separated more clearly like Fig. 3(t-SNE) for better visualisation.

It is first developed to address the problem of Classical scaling where the model tries to linearly reduce the dimensions giving more importance to the widespread data. Sammon mapping uses weighted cost function so large or small distances are treated with the proper precision and scale. The advantage that t-SNE has over Sammon Mapping is the higher cost associated with the wrong clustering of nearby points and lower cost associated with wrong clustering of widespread points. This allows the algorithm to separate the clusters more clearly than Sammon Mapping.

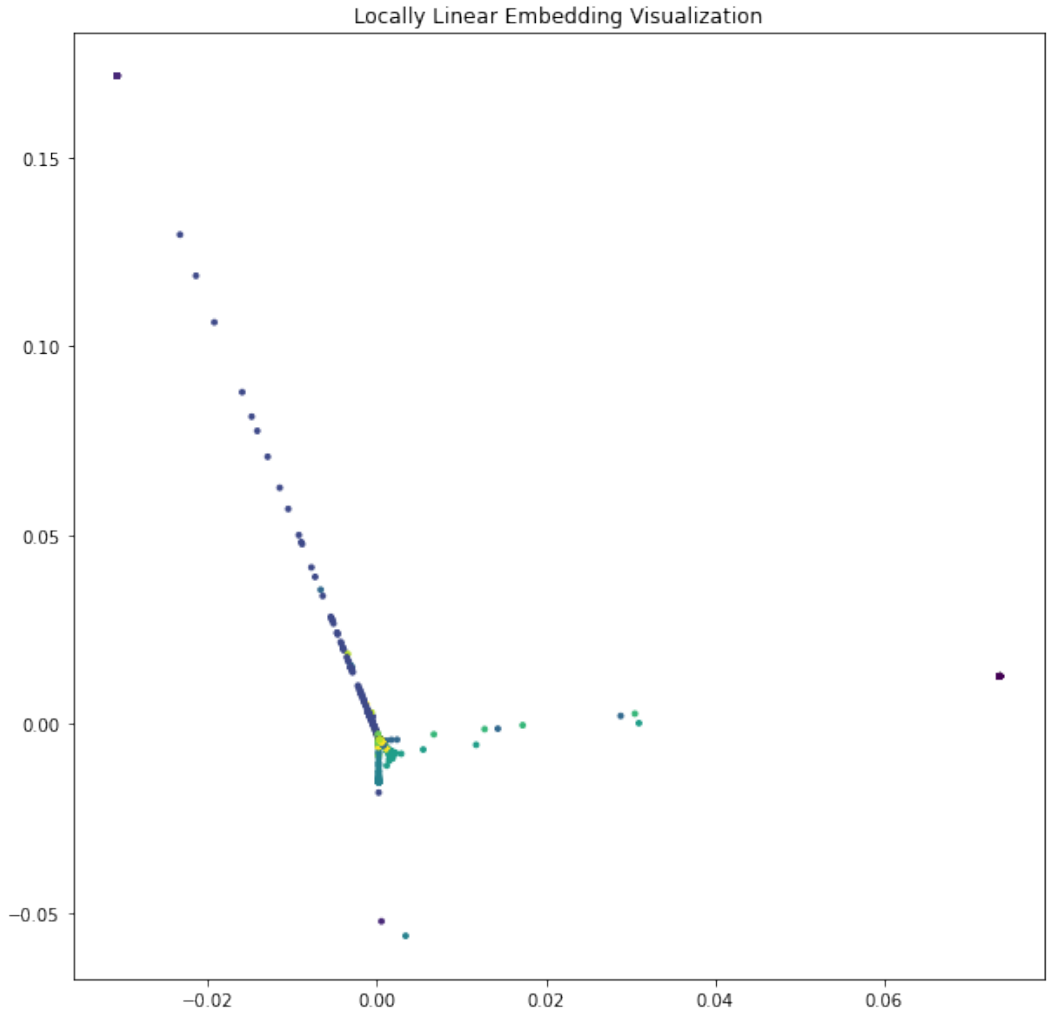


Figure 8: Locally Linear Embedding Visualization

The plot(Fig. 8) clearly shows that LLE clearly did not cluster the dataset correctly. LLE did not cluster the dataset into all the possible clusters , also it could not represent the visualised clusters correctly.

LLE works to preserve the geometric structure of higher dimensional data, As we cannot understand how the data is positioned in the higher dimensions, LLE gives us a understanding to how the data might exist. The strong performance of t-SNE compared to LLE is mainly due to a basic weakness of LLE: the only thing that prevents all datapoints from collapsing onto a single point is a constraint on the co-variance of the low-dimensional representation.

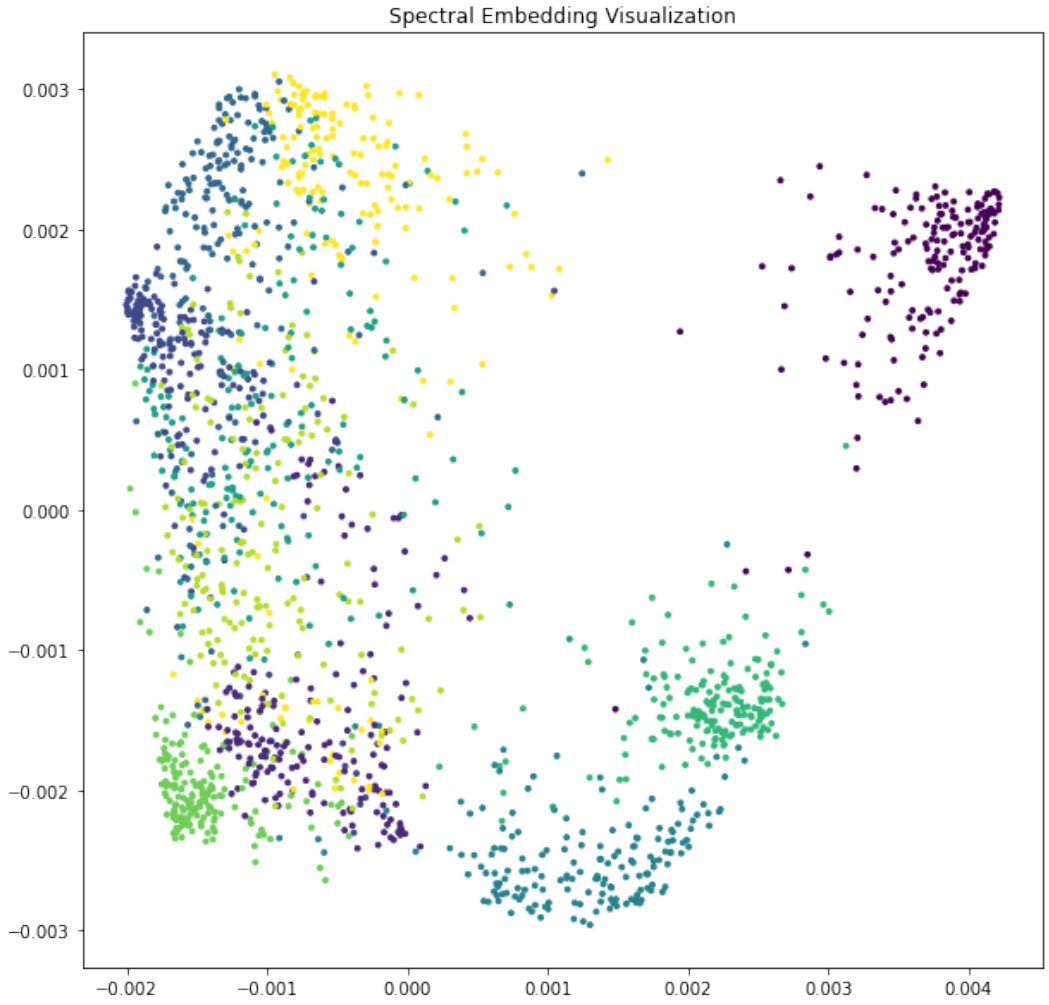


Figure 9: Spectral Embedding Visualization

As visible from the plot in Figure 9, the Spectral Embedding technique does not do a good job at visualizing the dataset but is certainly better off than some other techniques we have seen so far like LLE and Isomap. The clusters are distinguishable to a certain extent in the plot. For example, the purple, green, and blue clusters on the right side. However, on the left side we can see data points from the same cluster mixing with other clusters, making it harder to differentiate between the data points.

The major reason can be that at times the cuts produced by the spectral embedding method can be highly unbalanced. This decreases the usefulness of the algorithm for visualizing the dataset.

Conclusion

From our analysis, it is fair to conclude that our scratch t-SNE algorithm out performed all the other major linear and non linear visualization techniques. The reasons can be narrowed to a simple point that t-SNE has lot of complexity involved and it tries to preserve the local structure as well as the global structure of the data points in the lower dimensions. The major advantage it provides to visualization is that the data points that were close in higher dimensions will be close to each other in lower dimensions as well.

Future Work

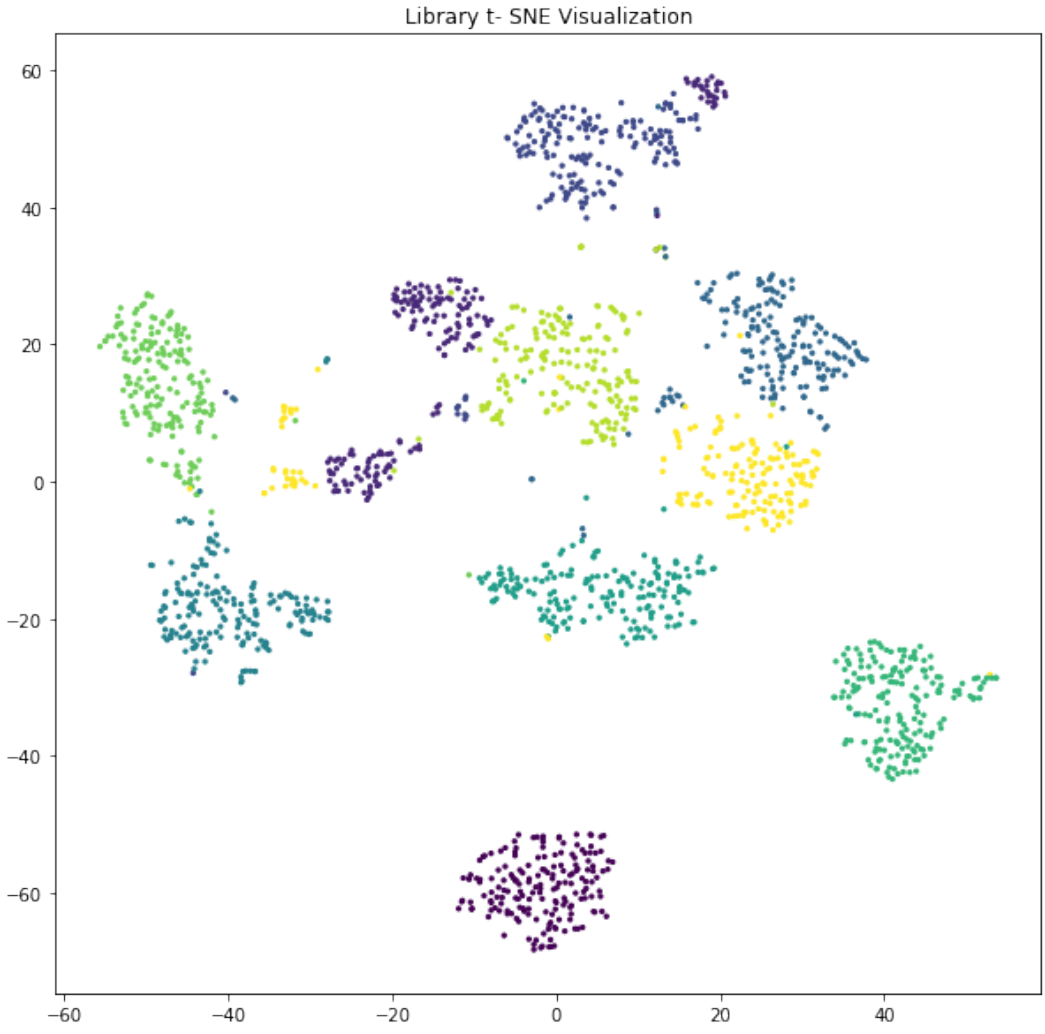


Figure 10: Library t-SNE implementation

The t-SNE model we implemented, although performs very well, needs some more optimisations with respect to time complexity and other considerations. Our model currently takes considerable amount of time for visualising higher dimensional and larger datasets. We can clearly see that the

library implementation of t-SNE (using sklearn) Fig. 10 performed much better than our algorithm. This has to do with the better techniques for optimisation and many more hyper parameters that the library version uses. Some of these important hyper parameters include maximum number of iterations without progress, minimum gradient norm, square distances, and angle.

Optimizing our scratch t-SNE algorithm with the specified hyper parameters will help us to generate better visualizations and produce a result somewhat similar to the library implementation of t-SNE technique.

Work Distribution

- Data selection & preprocessing - Aaradhya, Chayan
- Implementation of initial t-SNE - Jayant, Akhilesh
- Optimisation of initial t-SNE - Chayan, Aaradhya
- Hyper parameter Tuning - Chayan, Aaradhya
- Library implementation of other dimensionality reduction techniques - Jayant, Akhilesh
- Comparative Study of the techniques with t-SNE - Akhilesh, Jayant
- Report - everyone

References

[1] *Comparison of manifold learning methods*. URL: https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html#sphx-glr-auto-examples-manifold-plot-compare-methods-py.

[2] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[3] Tom Pollard. *Sammon*. https://github.com/tompollard/sammon. 2019.

[4] J. Sammon. "A Nonlinear Mapping for Data Structure Analysis". In: *IEEE Transactions on Computers* 18.05 (1969), pp. 401–409. ISSN: 1557-9956. DOI: 10.1109/T-C.1969.222678.

*****