COMP 551 Mini Project 4: Visualizing-High-Dimensional-Data

Dian Basit
260771254
dian.basit@mail.mcgill.ca

Hadi Zia
260775855
syed.h.rizvi@mail.mcgill.ca

Sagar Nandeshwar
260920948
sagar.nandeshwar@mail.mcgill.ca

Abstract

This paper examines different visualization algorithms and strategies that compress data and/or limit the number of attributes to be examined and shown in order to overcome the visualization issues provided by huge and high-dimensional data. The dataset that have been used in this project are; MNIST, Fashion-MNIST, Olivetti Faces and COIL-20. For better computational efficiency we inititally reduced the dimensionality of a data set, we employed principal component analysis (PCA). We show how t-SNE performs on our data sets and compare it to other visualization techniques such as SNE, Isomap, Locally Linear Embedding (LLE) and Uniform Manifold Approximation and Projection (UMAP). Hence we used these reduction techniques to present high-dimensional data into two-dimension. We presented a thorough analysis of our findings and drew some conclusions.

1 Introduction

In this project we implemented t-SNE and SNE from scratch using numpy and evaluated the performance of them with other visualization techniques, Isomap, LLE and UMAP. The goal of dimensionality reduction is to keep as much of the high-dimensional data's significant structure as feasible in the low-dimensional map. An already present work that we used to do our project on, is written by Laurens van der Maaten of Tilburg University and Geoffrey Hinton of the University of Toronto, and explores data visualization using t-SNE and other techniques mentioned above[1]. Various solutions to this problem have been presented, each preserving a different form of structure. PCA, for example, is a linear dimensionality reduction technique that focuses on keeping low-dimensional representations of distinct datapoints far away. It is frequently more necessary to retain the low-dimensional representations of extremely similar datapoints close together for high-dimensional data that lies on or near a low-dimensional, non-linear manifold, which is typically not attainable with a linear mapping. Lets move forward and discuss the various dimension reduction algorithms we have used.

1.1 Principal Component Analysis (PCA)

The goal of principle component analysis (PCA) is to reduce the dimensionality of a data set with many connected variables while maintaining as much variance as possible. By projecting the original data in the direction where the variance between data points is largest, PCA turns a set of possibly correlated variables into a set of uncorrelated linear combinations of those variables called principal components. The outcome is a two-dimensional scatter plot, which makes data easier to understand. When utilising PCA, there are a few issues to consider. For starters, the technique is sensitive to the data set's varying scale. As a result, mean centering and using the correlation matrix, which is normalised, is recommended. If the data is not normalised, the characteristics on the biggest scale will dominate the new main components. In addition, PCA isn't ideal for classification. Finally, a cumulative explained variance threshold must be specified or modified.

1.2 Isometric Mapping (Isomap)

Isomap is a non-linear dimensionality reduction method based on spectral theory that aims to keep geodesic distances in the lower dimension intact. Isomap begins by putting together a neighbourhood network. The approximate geodesic distance between all pairs of points is then calculated using graph distance. The low dimensional embedding of the dataset is subsequently discovered using eigenvalue decomposition of the geodesic distance matrix. The Euclidean

Submitted to ML Reproducibility Challenge 2020. Do not distribute.

metric for distance remains true in non-linear manifolds if and only if the neighbourhood structure can be approximated as linear. Euclidean distances can be deceiving if the vicinity has holes. In contrast, if we follow the manifold to estimate the distance between two points, we will get a better indication of how far or near two points are.

1.3 Stochastic Neighbor Embedding (SNE)

The high-dimensional Euclidean distances between datapoints are converted into conditional probabilities that express similarities in SNE. The conditional probability, p jli, that xi would choose xj as its neighbour if neighbours were chosen in proportion to their probability density under a Gaussian centred at xi is the similarity of datapoint xj to datapoint xi. For close datapoints, p jli is rather high, whereas for far apart datapoints, p jli is nearly infinitesimal (for tolerable values of the Gaussian variance, I The conditional probability pjli is calculated mathematically as follows:

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2/2\sigma_i^2\right)};$$

Figure 1

Our 2D representation of X is matrix Y, which is a Nx2 matrix. We may construct distribution q based on Y in the same way that we constructed p. This is defined as:

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i}\exp\left(-\|y_i - y_k\|^2\right)}.$$

Figure 2

Our ultimate goal is to select points in Y that produce a conditional probability distribution q that is comparable to p. This is accomplished by lowering a cost: the difference in KL between the two distributions. We wish to keep this cost to a minimum. We're just interested in the gradient with regard to our 2D representation Y because we're going to apply gradient descent. Perplexity is a parameter that we set in SNE (and t-SNE) (usually between 5 and 50). The i's are then set so that the perplexity of each row of P is equal to our desired perplexity – the parameter we selected. The greater the i the closer the probability distribution is to having all probabilities equal to 1/N. So, if we desire more perplexity, we'll increase the size of our 's, which will lead the conditional probability distributions to flatten. This effectively raises each point's number of neighbours. To verify that the perplexity of each row of P, Perp(Pi) = target perplexity, we just execute a binary search over each i until Perp(Pi) = our target perplexity. Perplexity Perp(Pi) is a monotonically growing function of i hence this is possible. We use a matrix of negative euclidean distances and a target perplexity to find all i's. We execute a binary search over all possible values of i's for each row of the distances matrix until we locate the one that results in the target perplexity. We then return a numpy vector with the found optimal i's. By reducing the gradient of the cost C with respect to Y until convergence, we might obtain a good 2D representation Y. However, because SNE's gradient is more difficult to implement, we'll instead employ Symmetric SNE. We minimise a KL divergence over the joint probability distributions with entries pij and qij, rather than conditional probabilities pilj and qilj, in Symmetric SNE. So now we have all of the information we need to calculate Symmetric SNE.

1.4 t-Distributed Stochastic Neighbor Embedding (t-SNE)

SNE produces reasonably good visualizations, but it is plagued by a difficult-to-optimize cost function and an issue we call the "crowding problem." The t-SNE project tries to solve these issues. The cost function used by t-SNE differs from that used by SNE in two ways: it uses a symmetrized version of the SNE cost function with simpler gradients, and it computes the similarity between two points in the low-dimensional space using a Student-t distribution rather than a Gaussian distribution. To solve both the crowding and optimization concerns of SNE, t-SNE uses a heavy-tailed distribution in low-dimensional space.

It's easy to transition from Symmetric SNE to t-SNE. Only the way we define the joint probability distribution matrix Q, which includes entries qij, differs, and gives us the following:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}.$$

Figure 3

This is obtained by assuming that the qij follow a one-degree-of-freedom Student t-distribution. Essentially, this indicates that the technique is nearly invariant to the low-dimensional mapping's general scale. As a result, the optimisation works the same for very far apart locations as it does for ones that are closer together.

1.5 Locally Linear Embedding (LLE)

The LLE algorithm begins by identifying a set of each point's closest neighbours. It aids in representing the point as a linear combination of its neighbours after locating the nearest neighbours by determining the weights set for each point. It also employs an eigenvector optimization technique in the final step of determining the low dimensional embedding for points. Finally, each point may be described as a linear combination of its neighbours thanks to the final step. There is no internal model in LLE.

The LLE employs the barycentric coordinates of Xi to compute the neighbour Xj of any point Xi. The algorithm reconstructs the original point using the linear combination. The linear combination to point Xi is given by the weight matrix Wij. The algorithm's main purpose is to create data of dimensions D of size d, where $D \gg d$. The idea behind the creation of neighbourhood preserving maps is that the same weight matrix used to rebuild original points in data with dimensions D may also be used to reconstruct the same original point in data with dimensions d. In addition, the cost function minimization is considered, and each point Xi in the D dimensional data is mapped onto a point Yi in the D dimensional data.

1.6 Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction approach comparable to t-SNE that can be used for visualisation as well as non-linear dimension reduction. The data is uniformly distributed on a Riemannian manifold, the Riemannian metric is locally constant (or can be estimated as such), and the manifold is locally connected, according to the algorithm. It is possible to model the manifold with a fuzzy topological structure based on these assumptions. The embedding is discovered by looking for the closest possible equivalent fuzzy topological structure in a low-dimensional projection of the data.

2 Datasets

2.1 MNIST

The Modified National Institute of Standards and Technology dataset is referred to as the MNIST dataset. It's a collection of 60,000 small squared grayscale photos of handwritten single numbers ranging from 0 to 9[2]. Total dimensions are 784.

2.2 Fashion MNIST

The Fashion MNIST Dataset is an MNIST-like dataset of 70,000 28x28 labeled fashion images. It is divided into two subsets: one for training(60,000) and the other one for testing(10,000). Each row in the dataset is a separate image. The first column is the class label whereas the remaining columns are pixel numbers (784 total). Each value is the darkness of the pixel (1 to 255)[3]. Total dimensions are 784.

2.3 Olivetti Faces

There are ten different images of 40 distinct people in this dataset. There are 400 face images in total. Face images were taken at different times, varying lighting, facial expressions and facial details. All images have black backgrounds

and are gray scaled. Size of each image is 64x64 and Image pixel values were scaled to [0, 1]. Names of the 40 people were encoded to an integer from 0 to 39[4]. Total dimensions are 4096.

Methods	parameter
SNE	perplixity=40
ISOMAP	k=12
LLE	k=12
UMAP	k=12

3 Results

As far as importing data from the datasets is concerned, the MNIST and Fashion MNIST dataset were both divided into training and test sets of 60,000 and 10,000 respectively and they both had 10 classes, where we only considered 2% of training data in this example. Olivetti faces dataset had 400 samples with 40 classes representing random individual and the COIL-20 dataset had 1440 data-pieces for train and test sets with 20 classes representing objects. Firstly we extracted 2% of MNIST and Fashion MNIST dataset for training purposes. This is needed due to limited computational power, we then normalized the data by constraining to [0, 1]. We decided with not using the COIL-20 dataset for testing purposes as it has too many dimensions and we were unable to use PCA to reduce it down to 30 dimensions. Further, SVD decomposition was extremely expensive so we used fashion MNIST dataset instead. We utilized PCA to reduce the high dimension of the datasets to 30 dimensions as mentioned in the referenced paper. This eases computation and allows for easier testing.

Our purpose for testing was to show overall converging process of the models and how the datapoints visually separate. The main difference in SNE and t-SNE is in the cost functions they use. SNE uses softmax related gaussian distribusion whereas t-SNE uses t-distribution where closer points are bought even closer and further points are moved away. The following images show the dynamic visualization of the SNE and t-SNE models and show how they both perform data separation on the MNIST Dataset for the 0th, 500th and 1000th iterations.

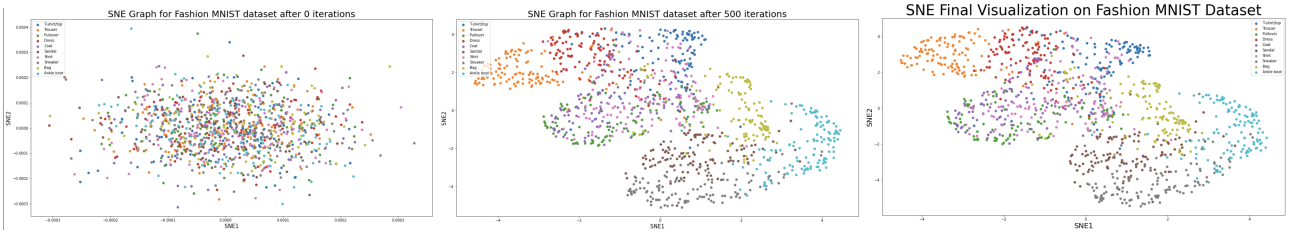


Figure 4: SNE on Fashion MNIST

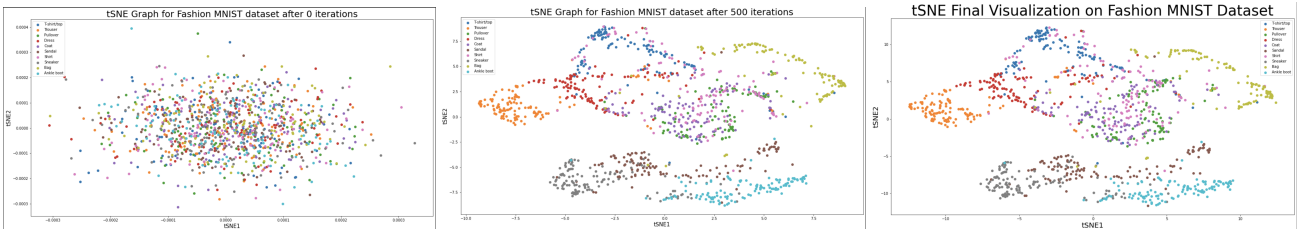


Figure 5: t-SNE on Fashion MNIST

Comparison graphs of the MNIST dataset for t-SNE, SNE, isomap, LLE and UMAP are shown in the appendix in the figure 6 titled '**MNIST**'.

Comparison graphs of the Fashion MNIST dataset for t-SNE, SNE, isomap, LLE and UMAP are shown in the appendix in the figure 7 titled '**Fashion MNIST**'.

Comparison graphs of the Olivetti Faces dataset for t-SNE, SNE, isomap, LLE and UMAP are shown in the appendix in the figure 8 titled '**Olivetti Faces**'.

4 Discussion and Conclusion

The comparison graphs are mentioned in the coding section in detail. We see that UMAP performed better cluster separation compared to other. Although this reduction technique wasn't mentioned in the research paper, but we wanted to add this to show why UMAP is now widely used as a visualization technique. After UMAP, tSNE had a superior performance over other techniques like SNE, LLE and Isomap. Since tSNE is the primary focus of this report, we impelemented tSNE and SNE from scratch. We will be comparing the mathematical advantage of tSNE over SNE and why the former performed better than the latter. In addition, we'll also discuss the supriority of the modern reduction technique i.e. UMAP over tSNE and why it was able to significantly separate the clusters.

When comparing tSNE and SNE reduction technique, tSNE has a higher mathematically advantage over SNE because, former uses a t-distribution and a slight addition to KL Cost function which reduces a "crowded-problem" in the SNE convergence. This special "t-distribution" has an added advantage that points closer to the datapoint in focus i.e. x_i, similar closer points becomes "attracted" to the target datapoint while other dissimilar classes datapoints, are repelled more strongly. In addition, since the distribution has a longer tail-end, further points become more further apart.

UMAP is relatively new technique introduced in the industry and as we can see, it was able to outperform all the other techniques, in terms of visualizing classified datapoints, including tSNE. The reason for this superiority is because UMAP has a mathematically superior cost function as it uses "Binary Cross-Entropy", compared to tSNE which uses "KL Divergence" cost function. KL divergence performs well, if datapoints are closer together in high-dimension. Because if they are closer together in high-dimension, then to minimize the cost-function in lower-dimension, they have to be closer-together. However, if in the higher-dimension, datapoints are further apart, then in low-dimension, whether datapoints are further or closer together, then minimization of cost function becomes irrelevant, hence it only preserves local structure. Whereas, UMAP is able to preserve global structure due to its cross-entropy cost function as points further in high-dimension, and vice versa in order for gradient descent to converge.

5 Statement of Contribution

Dian - Primary responsibility for coding. Implemented SNE and tSNE from scratch and then compared tSNE against other reduction techniques over different datasets to get better visual understanding. Final touchings on the report and explained mathematical reasoning behind the results.

Hadi - Worked on UMAP implementation and testing. Analyzing test results, summarize findings and writing the report.

Sagar Nandeshwar - Implemented Isomap, LLE and Sammon Mapping. Worked on the report and wrote findings and analysis.

References

[1] L. Van der Maaten G. Hinton, 'Visualizing data using t-SNE', Journal of machine learning research, . 9, . 11, 2008.

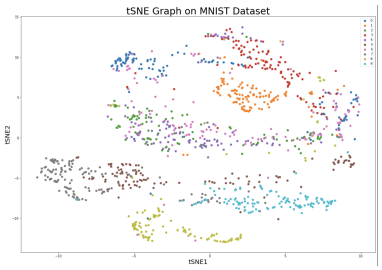
[2] J. Brownlee, "How to Develop a CNN for MNIST Handwritten Digit Classification", Machine Learning Mastery, 2022. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/: :text=The%20MNIST%20dataset%20is%20an,digits%20between%200%20and%209.

[3] "Fashion MNIST", Kaggle.com, 2022. [Online]. Available: https://www.kaggle.com/datasets/zalando-research/fashionmnist. [Accessed: 01- Apr- 2022].

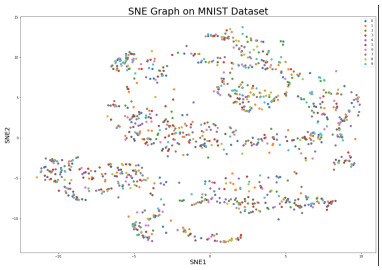
[4] "Face Recognition on Olivetti Dataset", Kaggle.com, 2022. [Online]. Available: https://www.kaggle.com/code/serkanpeldek/face-recognition-on-olivetti-dataset/notebook. [Accessed: 27- Apr- 2022].

[5] Citeseerx.ist.psu.edu, 2022. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.641.1322rep=rep1type=pdf [Accessed: 27- Apr- 2022].

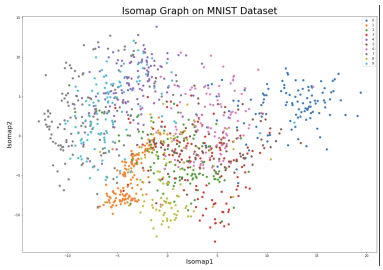
6 Appendix



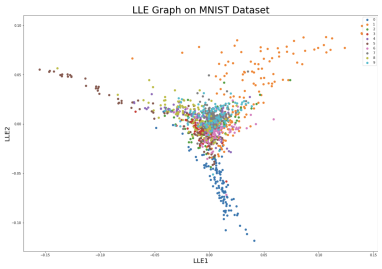
(a) t-SNE



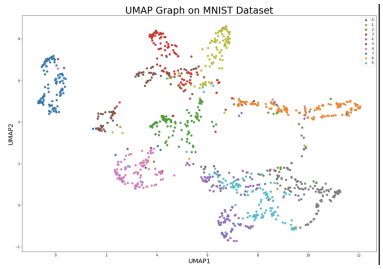
(b) SNE



(c) Isomap

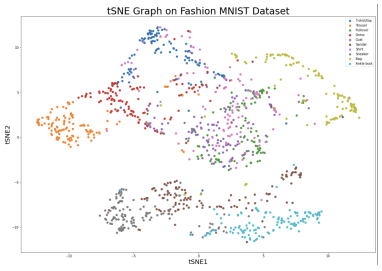


(d) LLE

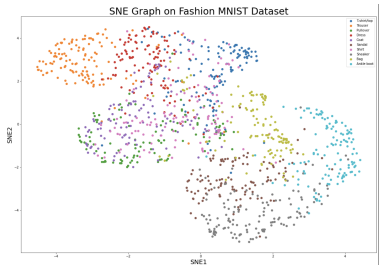


(e) UMAP

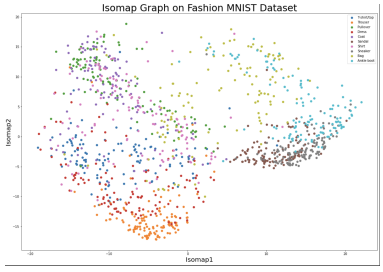
Figure 6: **MNIST**



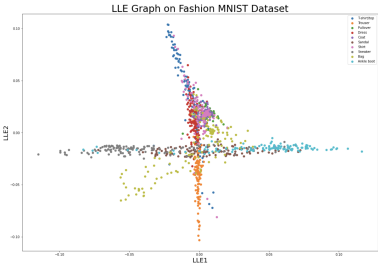
(a) t-SNE



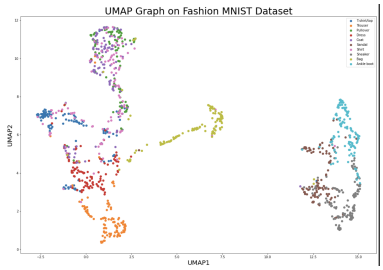
(b) SNE



(c) Isomap

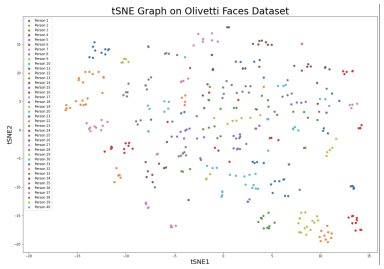


(d) LLE

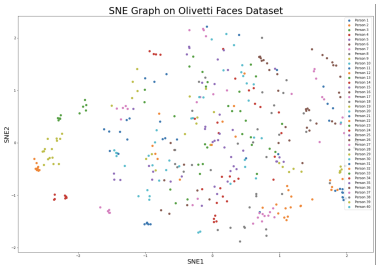


(e) UMAP

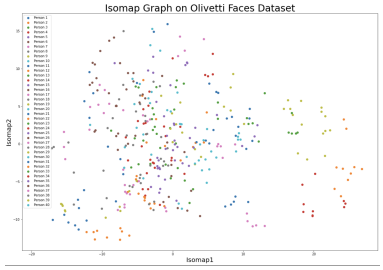
Figure 7: **Fashion MNIST**



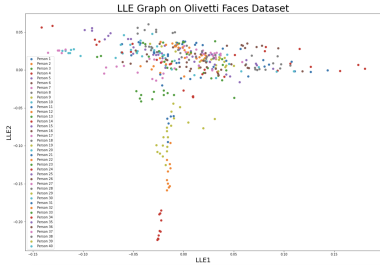
(a) t-SNE



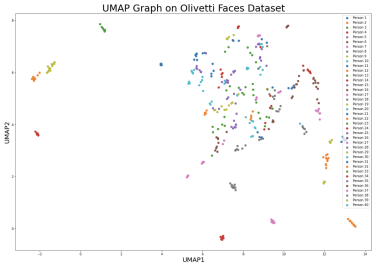
(b) SNE



(c) Isomap



(d) LLE



(e) UMAP

Figure 8: **Olivetti Faces**