GE-461: Introduction to Data Science

Dimensionality Reduction and Visualization

Muhammad Abdullah S. Mulkana – 21801075

21.04.2022

Table of Contents

Table of Figures

Introduction

In this assignment, we aim to explore several dimensionality reduction methods in order to understand how they perform on high-dimensional data. For this, we are using the MNIST data provided, which contains 5000 images of dimensions 400 (20x20) for the digits from 0 to 9. In order to evaluate the performance, we use a Quadratic Gaussian classifier. We experiment with Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) in order to determine their effect on classification error. We use more complex techniques such as Sammon's mapping, and t-Distributed Stochastic Neighbor Embedding to visualize our high dimensional data in two dimension.

Quadratic Gaussian Classifier

To evaluate and compare the performances of our preprocessing techniques, we use a quadratic Gaussian classifier. The classifier models training samples of each class to a multivariate gaussian:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

Where d is the number of samples in the class, Σ is the covariance matrix and μ is the mean vector. We calculate the mean vector and covariance matrix as follows:

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x_i}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i}-\hat{\boldsymbol{\mu}})(\mathbf{x_i}-\hat{\boldsymbol{\mu}})^T$$

For each of the 10 classes, we have the probability densities $p(x|\mu, \Sigma)$. Then, for each test sample, we calculate the probabilities and the one with the maximum is our predicted class.

A common classifier that works like this is the Naïve Bayes classifier which assumes that the features are independent and therefore the covariance matrix is simply a diagonal matrix where the diagonal elements are the variances of the features. We do not assume that for our case and thus the covariance matrix that we use is not diagonal. To implement a quadratic Gaussian classifier, we use 'QuadraticDiscriminantAnalysis' from 'sklearn.discriminant_analysis' [1]. The module documentation states what is explained above and conforms with the classifier model that we are told to use [2].

Question 1: PCA

In this part, we analyze the performance of Principal Component Analysis (PCA) on the data. First, we split the data into train and test sets so that we have 50% of samples from each class. In order to achieve this we use the 'train_test_split' function from 'sklearn.model_selection' [3]. We use the 'stratify' parameter to ensure that our split has 50% of samples from each class. We keep this split the same over all models so that the results can be accurately compared.

Part 1

In this part, we need to perform PCA on the training data. To do this, we use the PCA class from the sklearn library [4]. An important preprocessing step before applying PCA is to centralize and standardize the data. We do this we use the StandardScaler function from sklearn [5]. **Error! Not a valid bookmark self-reference.** (left) shows the 400 eigenvalues plotted in descending order. It shows how much of the variance each of the principal components explain.

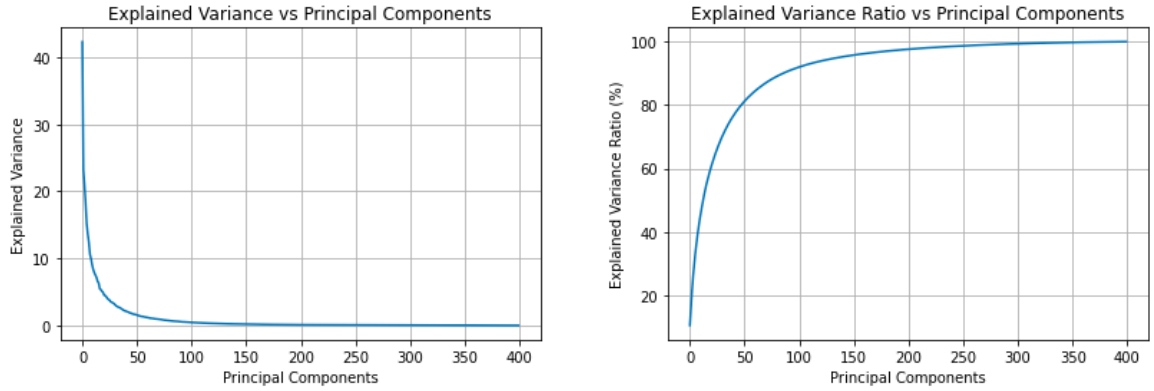

Figure 1: Plots of explained variance (left) and cumulative explained variance ratio (right) vs principal components.

In order to understand better, we may plot the percentage of variance explained by each PC. **Error! Not a valid bookmark self-reference.** (right) shows the cumulative sum of the explained variance ratio. The first 81 PCs approximately explain 90 % of the variance. Choosing this as the number of dimensions to project our data would be a good choice since we will still retain most (approximately 90%) of the variance and reduce the number of features, thus reducing model complexities and decreasing training time.

Part 2

Figure 2 shows the mean of all the training images. Looking at the figure, we can observe that it resembles some of the digits, especially 8 or 3. This is because in most of the samples, pixels in these locations are used.

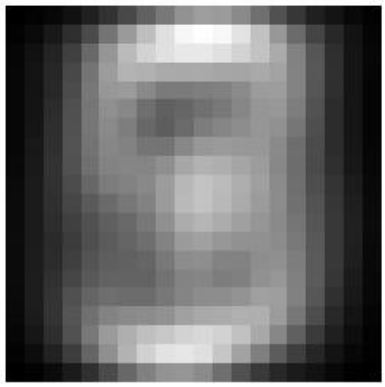

Figure 2: Mean of the training images.

As mentioned in part 1, choosing the first 81 dimensions would be a good choice. Figure 3 shows the first 81 eigenvectors represented as 20x20 images.

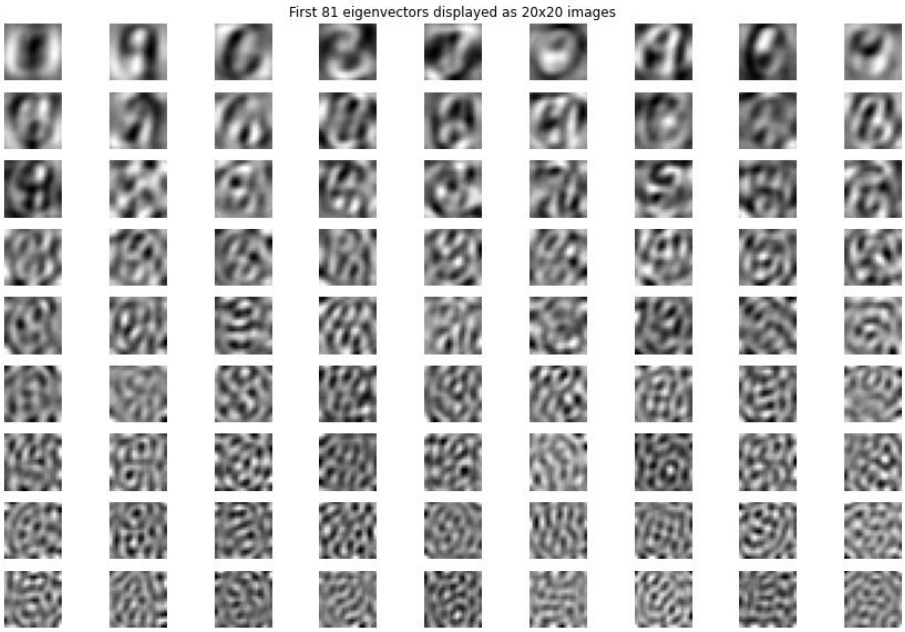

Figure 3: First 81 eigenvectors displayed as images.

We see that the first few eigenvectors seem to represent some of the digits. The first eigenvector, corresponding to the highest eigenvalue resembles circular digits, especially zero. The fourth eigenvector resembles a 2 or an 8. The 7th eigenvector resembles a 9. As we proceed, we get more noisy eigenvectors. This shows that those dimensions contain mostly noise.

Part 3 & 4

In these parts, we fit a quadratic Gaussian classifier to the training data for PCA dimensions between 1 and 200. **Error! Reference source not found.** shows the error for both the training

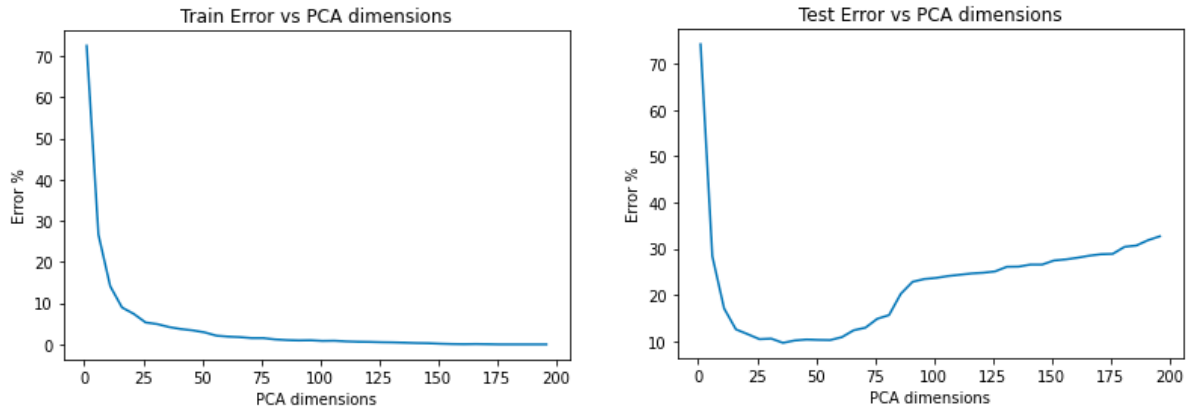

Figure 4: Error on the train and test set vs principal components chosen.

data and the testing data.

From the figure above we see that the training error decreases as the number of dimensions are increased. This is as expected since we have more information and our models can thus differentiate better between classes. From the testing error, we see that it decreases for a while but then starts to increase. From Figure 3, we see that eigenvector corresponding to higher PC's do not resemble our data but rather capture noise. We can say that our model is overfitting, i.e., fitting on the noise as well which causes it to perform worse on test data as dimensions are

increased. This is a problem known as ‘Curse of Dimensionality’ or Hughes phenomenon which states:

“…with a fixed number of training samples, the predictive power of any classifier first increases as the number of dimensions increase, but after a certain value of number of dimensions, the performance deteriorates [6].”

Question 2: LDA

In this part, we apply Linear Discriminant Analysis (LDA) to the training data. Then we train and test a quadratic Gaussian classifier for all possible number of bases (1 to 9)

Part 1

In this part, we apply LDA using the ‘LinearDiscriminantAnalysis()’ class from ‘sklearn.discriminant_analysis’ [7]. The parameter is the number of components that we consider which we keep as 9 (num_classes-1) in order to get the bases.

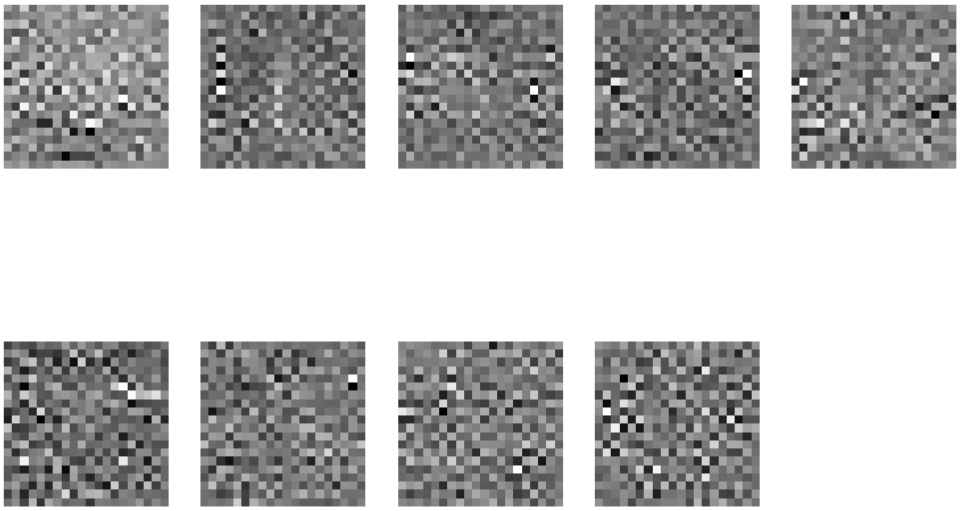

Figure 5: All 9 bases displayed as images.

Figure 5 shows the 9 bases for LDA. As we can see, unlike the PCA, the bases do not resemble any of the digits. In PCA, our aim is to retain the maximum variances which help us visualize the data better but the LDA bases transform the data in a way that enables best classification [8].

Part 2 & 3

These parts are equivalent to parts 3 and 4 in question 1. Here we vary the number of components from 1 to 9 in LDA to project the training and testing data. Then we apply the quadratic Gaussian classifier as before. **Error! Reference source not found.** show the training and testing errors respectively.

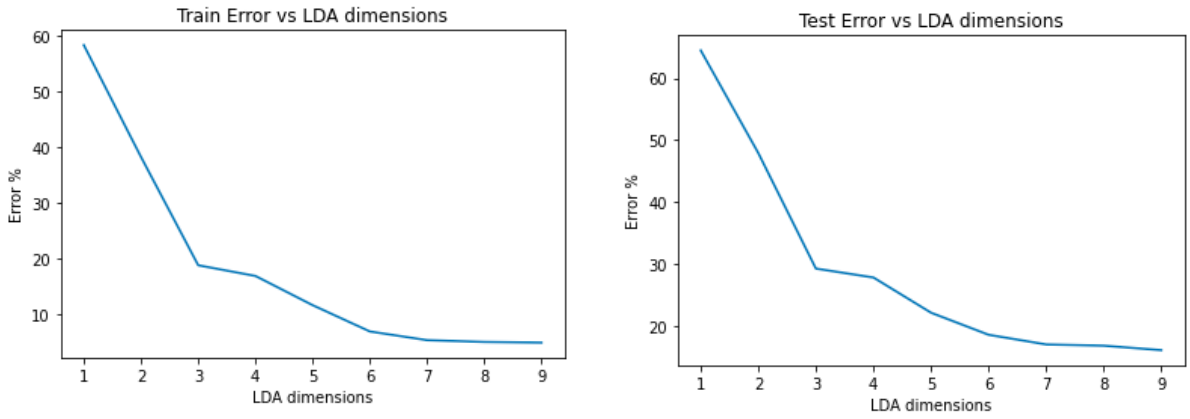

Figure 6: Train and test errors vs LDA dimensions used.

From **Error! Reference source not found.** we see that the classification errors for both training and testing set decreases as the number of LDA dimensions increases. This means that the more dimensions that we project to, the better the data can be classified and there is no overfitting to noise. If we were to select a subspace with a certain number of dimensions to project the data to, we would choose 6 or 7 as they give very little error and increasing this gives very little improvement in performance.

Question 3: Sammon's Mapping & t-SNE

Part 1: Sammon's Mapping

In this part, we apply Sammons Mapping to represent our 4000 dimensional data to 2 dimensional so that we can plot it and then observe if the classes can be separated easily or not. Sammon's mapping is different in the sense that it is non-linear and retains the underlying structure while reducing the dimensions so that we can visualize the data [9]. Figure 7 shows the result of using Sammon's Mapping. As we can see there are not many distinct clusters and that the scatter plot is circular in nature. The data points for '1' are the only ones distinctively clustered while the rest seem to overlap.

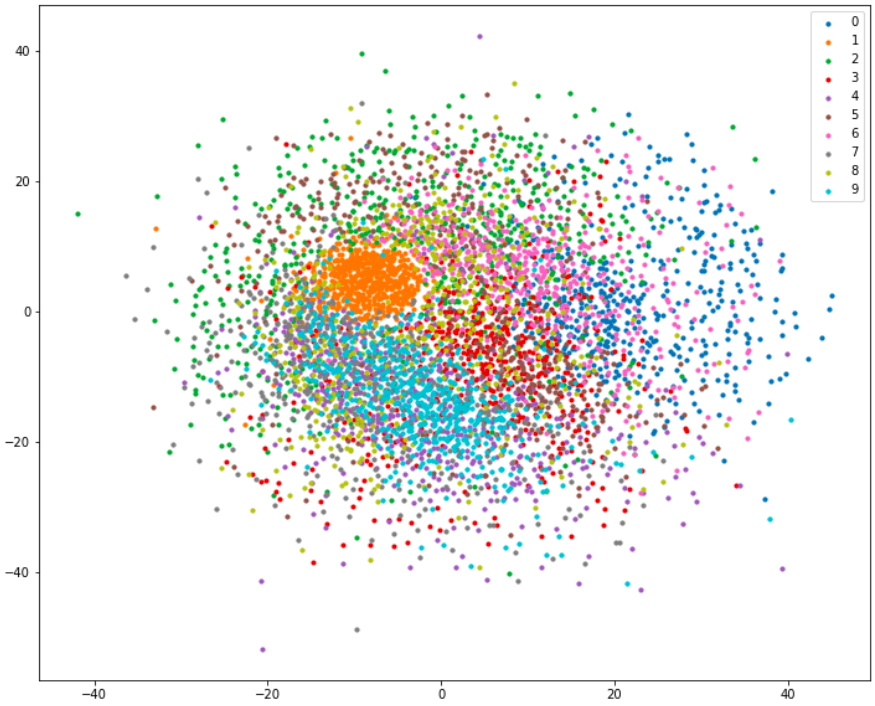

Figure 7: Data projected to two dimensions using Sammon's Mapping.

For implementing Sammon's mapping, we use the sammon function from [10]. Table 1 lists the parameters that we use.

Table 1: Parameters for sammon function. From [10].

Parameter	Description	Value
x	Data to be transformed.	data_norm
n	Number of dimensions to transform to.	2
inputdist	Flag to determine if input is distances or not.	'raw' (Default)
maxhalves	Maximum number of step mappings.	50 (Default)
maxiter	Maximum number of iterations.	500 (Default)
tolfun	Relative tolerance on objective function.	1e-9 (Default)
init	Default initialization.	'default' (Default)

Part 2: t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is another method to represent and visualize high dimensional data in 2-dimensions. t-SNE uses the students t-distribution to find the probabilities of a data point being the neighbor of another data point [11]. For data points closer, this probability is higher and vice versa [11]. This probability is calculated for both high dimensional data and low dimensional data and the loss function that we try to minimize is the difference between these probabilities [11]. To implement t-SNE, we use the 'TSNE' function from 'sklearn.manifold' [12].

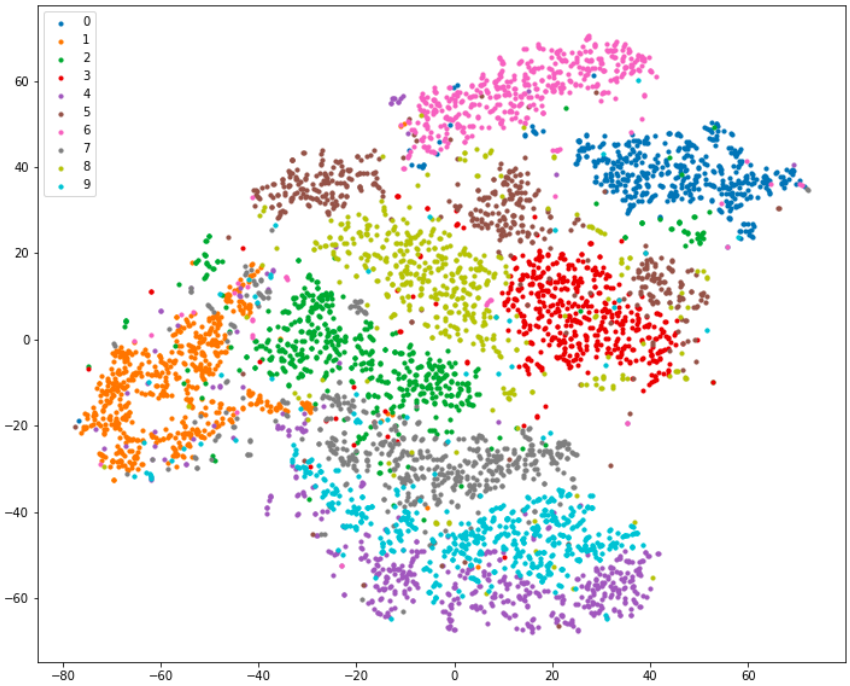

Figure 8: Data projected to 2-D using t-SNE

Figure 8 shows the result of t-SNE. As we can see, this performs much better than the Sammon's mapping where the clusters do not overlap much. We can see that the cluster for '0' and '6' are relatively distinct. However, '2', '7', '9', and '4' are very close to each other and sometimes overlap. This is because these digits are similar to each other. Furthermore, we see an overlap between '3', and '8' showing that in the high dimensional space, these data points for these two classes are closer, again because the digits themselves are close in shape.

For both Sammon's Mapping and t-SNE, we normalize and scale the data before applying the algorithms [13] [14]. Table 2 show the parameters used for t-SNE.

Table 2: Parameters for the TSNE function. Descriptions From [12].

Parameter	Description	Value
n_components	Number of dimensions of embedded space.	2
perplexity	Parameter related to number of nearest neighbors.	30 (Default)
Early_exaggeration	Controls how tight natural clusters in the original space are in the embedded space and the spacing between them.	12.0 (Default)
Learning_rate	Learning rate of the algorithm.	'auto'
n_iter	Maximum number of iterations for the optimization.	1000 (Default)
n_iter_without_progress	Maximum number of iterations without progress before we abort optimization.	300 (Default)
min_grad_norm	Lower limit for gradient norm.	1e-7 (Default)
metric	metric used for calculating the distance between the data points.	'euclidean' (Default)
init	initiation of embedding.	'random' (Default)
verbose	Printing results	0 (Default)
Random_state	Determines the random number generator.	0

method	gradient calculation algorithm.	'barnes_hut' (Default)
angle	angular size of a distant node as measured from a point.	0.5 (Default)
n_jobs	number of parallel jobs to run for neighbors search.	None (Default)
Square_distances	Whether TSNE should square the distance values.	'legacy' (Default)

Most of these are default, only ones set by myself are n_components (2), random_state (0), learning_rate ('auto'), and init('random').

Some other libraries used:

- Scipy.io (function 'loadmat') [15] – reading the .mat file
- Numpy [16]
- Pandas [17]
- Matplotlib.pyplot [18]
- Sklearn.metrics (function 'accuracy_score') [19] – to calculate classification accuracy

References

[1] "'QuadraticDiscriminantAnalysis'," sklearn_quadratic_analysis, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html.

[2] "Linear and Quadratic Discriminant Analysis," sklearn, [Online]. Available: https://scikit-learn.org/stable/modules/lda_qda.html#lda-qda.

[3] "sklearn.model_selection.train_test_split -- scikit-learn 0.19.2 documentation," sklearn.model_selection, [Online]. Available: https://scikit-learn.org/0.19/modules/generated/sklearn.model_selection.train_test_split.html.

[4] "PCA," sklearn.decomposition, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[5] "StandardScalar," sklearn.preprocessing, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[6] S. Karanam, "Curse of Dimensionality - A "Curse to Machine Learning"," towards datascience, 11 August 2021. [Online]. Available: https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb.

[7] "LinearDiscriminantAnalysis," sklearn.discriminant_analysis, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html.

[8] S. Aksoy, "Feature Reduction and Selection," Spring '22. [Online]. Available: http://www.cs.bilkent.edu.tr/~ge461/current/lib/exe/fetch.php?media=ge461_dimensionality.pdf.

[9] "Sammons mapping: A non-linear mapping for data visualization," Data Farmers, [Online]. Available: https://data-farmers.github.io/2019-06-10-sammon-mapping/.

[10] tompollard, "Sammon mapping in Python," [Online]. Available: https://github.com/tompollard/sammon.

[11] saurabh.jaju2, "Comprehensive Guide on t-SNE algorithm with implementation in R & Python," Analytics Vidhya, 22 January 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/.

[12] "sklearn.manifold.TSNE -- scikit-learn 1.0.2 documentation," sklearn.manifold, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html.

[13] jon_simon, "Should data be centered+scaled before applying t-SNE?," stackexchange, 9 June 2016. [Online]. Available: https://stats.stackexchange.com/questions/164917/should-data-be-centeredscaled-before-applying-t-sne.

[14] Archie, "Is normalization required in Sammon mapping," stackexchange, 9 May 2017. [Online]. Available: https://stats.stackexchange.com/questions/212796/is-normalization-required-in-sammon-mapping.

[15] "scipy.io.loadmat - SciPy v1.8.0 Manual," scipy.io, [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html.

[16] "Numpy documentation," NumPy, [Online]. Available: https://numpy.org/doc/stable/.

[17] "pandas documentation," pandas, [Online]. Available: https://pandas.pydata.org/docs/.

[18] "Matplotlib.pyplot - Matplotlib 3.5.0 documentation," Matplotlib, [Online]. Available: https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.html.

[19] "accuracy_score," sklearn.metrics, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.