GE461 Introduction to Data Science

Project 2 Report

İrem Ecem Yelkanat
21702624
Section 1

Dimensionality Reduction and Visualization

In this project, I have focused on one of the classical problems in pattern recognition which is digit recognition. Using the MNIST database of handwritten digits, I have applied several algorithms to classify and recognize handwritten digits.

First, I split the data which consists of 5000 samples and 400 features into half as test set and train set. To randomly select 2500 samples for train data and 2500 samples for test data, I used train_test_split function from Scikit-learn library, and I assign 42 to random state of splitting to produce same output at every run of the code. Using the train set and test set obtained by this procedure, following questions were implemented.

Question 1

In this question, I lowered the feature dimension of the data which is 400 by using principal component analysis (PCA). I used PCA implementation from Scikit-learn library. Then, using the low dimensional data compared to original data, I trained a Gaussian classifier to measure the effect of dimensionality to performance of the classification.

1.1. In this part, I used PCA on training data which consists of 2500 samples, and I obtained new bases to project the training data onto it. For 400 features, I had 400 eigenvectors as a new set of bases. For each eigenvector, I found the corresponding eigenvalues and sorted them in descending order. Principal components and their corresponding eigenvectors can be seen in Figure 1. And principal components and their corresponding proportion of variance explained can be seen in Figure 2.

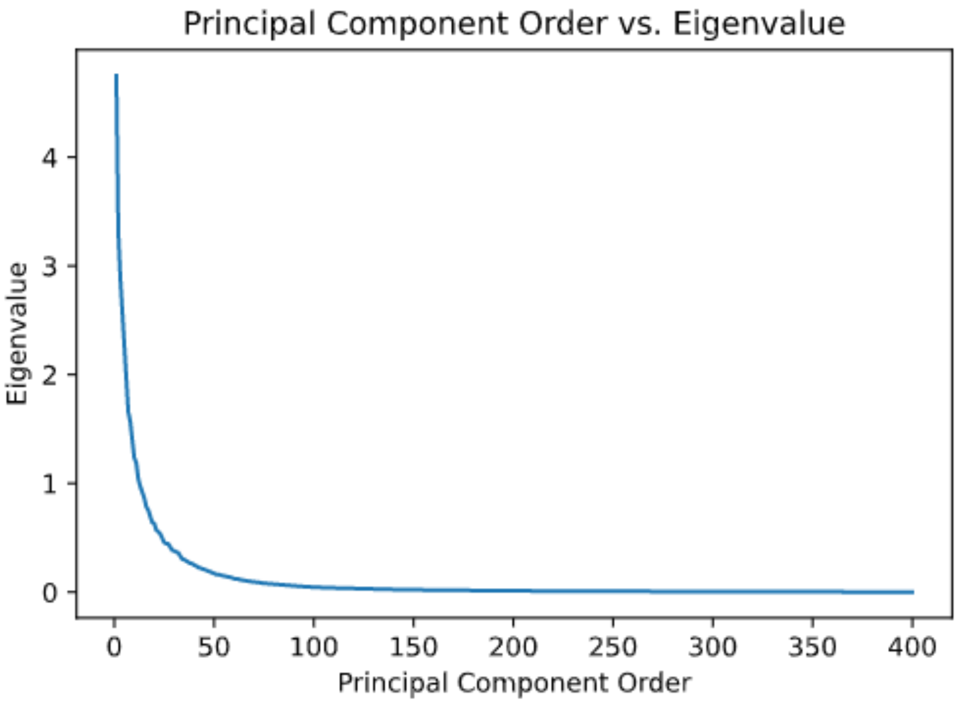

Figure 1: Principal Component vs. Eigenvalue

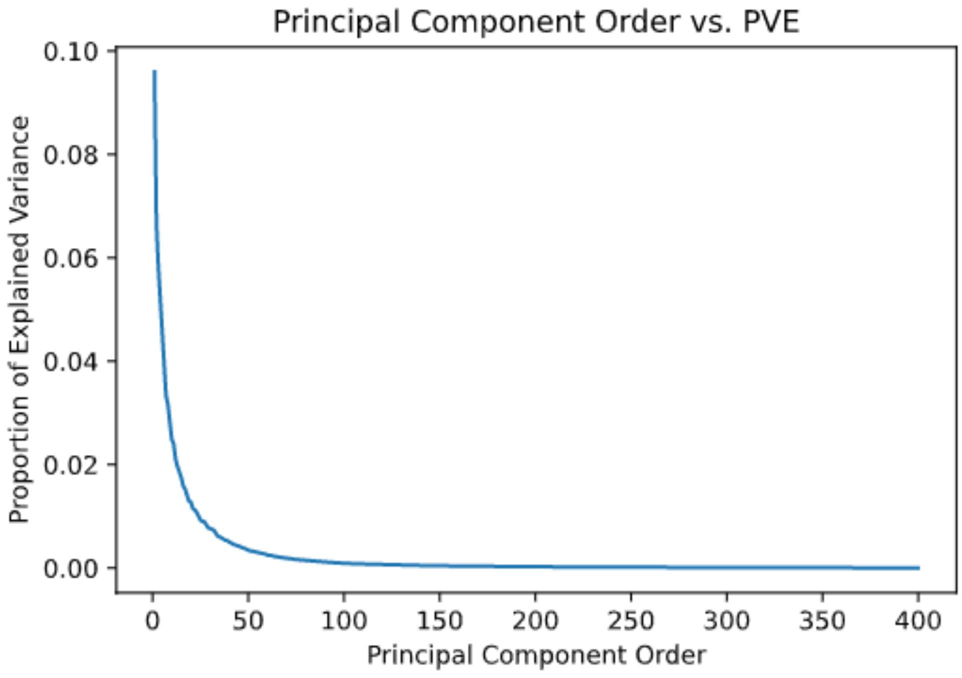

Figure 2: Principal Component vs. PVE

From these figures, it can be seen that there is a steep drop between first principal component and fiftieth principal component. It suggests from this fact that we should keep at least first fifty principal components of the data. Between fiftieth principal component and hundredth principal component, eigenvalue becomes closer to 0 in *Figure 1*, similarly, proportion of variance explained becomes closer to 0 in *Figure 2*. It suggests from this fact that after a certain number of principal components, principal components have eigenvalues close to 0 and thus adding new principal components does not increase the proportion of variance explained. In both of the figures *Figure 1* and *Figure 2*, approximately after 75 principal components, the shape of the curves becomes a straight line. Henceforth, I would choose 75 components by just looking at plots in *Figure 1* and *Figure 2*.

1.2. In this part of the question, I first calculated the mean of training data, as finding the mean of every feature with all samples. Taking the mean of every sample in training data, the result can be seen in *Figure 3*.

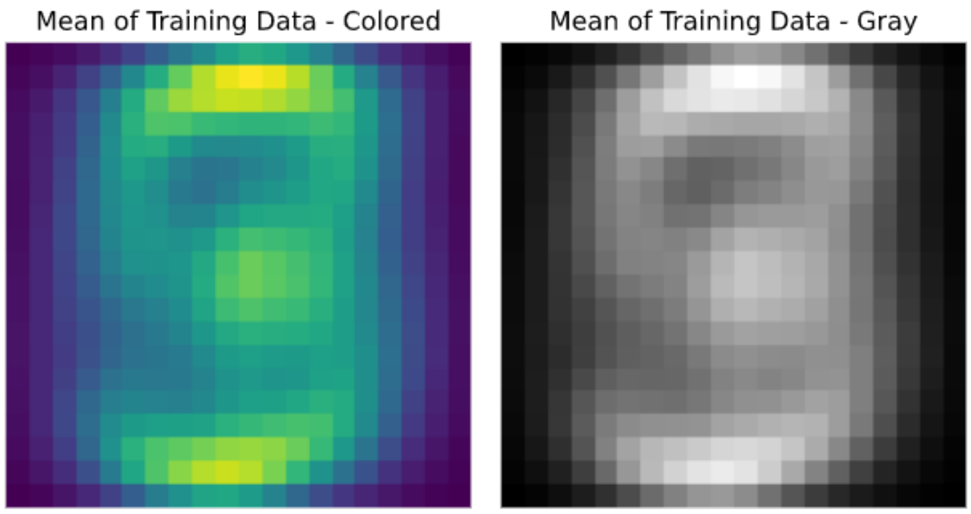

Figure 3: Mean of Training Data

As is can be seen in *Figure 3*, there are nearly perfect white pixels in the upper part and the lower part of the grayscale image of mean of training data. Considering all of the digits, 1 to 9, one of the common characteristics of handwritten digits is that they have rounded curves or straight lines in their upper and lower parts. In addition, in the middle of the grayscale image of mean of training data, there is also light gray pixels. Again, almost all of the handwritten digits have components in the their middle parts. Considering all of these facts, lights parts in *Figure 3* corresponds to more common patterns in handwritten digits and dark parts corresponds to less common patterns in handwritten digits as I expected. In addition, the images in *Figure 3* seems like a digit similar to 8 and 9. The digits 2, 3, 5, 6, 8, 9 and 0 have rounded patterns in common they correspond to 70% of the digits. Considering this fact, the mean of the training data have patterns representing the mixture of mentioned digits as I expected.

Additionally, I displayed eigenvectors corresponds to eigenvalues discussed in previous part as with selected 75 eigenvectors. I reshaped the components to 20x20 matrix and displayed them. The result can be seen in *Figure 4*.

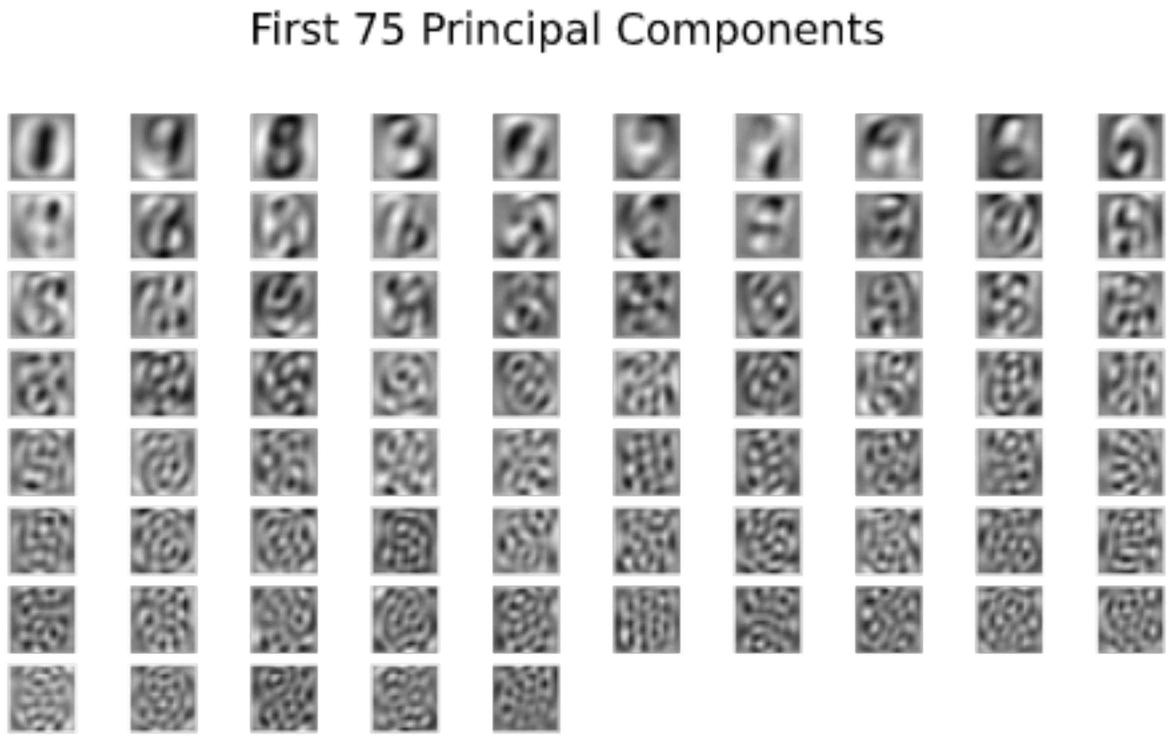

Figure 4: First 75 Eigenvectors - Principal Components

Principal components are ordered from first to last first from left to right and from top to bottom in *Figure 4*. These principal components show key characteristics of handwritten digits. First 5 principal components can be recognized as digits, for example first one is similar to 0, second one is similar to 9 etc. However, the other principal components are hard to recognize as a handwritten digits. As it can be seen, as order of the principal components increases, the details of the images become complicated as I expected since principal components are ordered with respect to their proportion of explained variance and proportion of explained variance decreases as the order of principal component increases as in *Figure 2*. Also, considering the mean of the training data and first four principal components, the resulting images support each other in shape and patterns. In addition, first ten eigenvectors are similar to the digits 0, 3, 6, 8 and 9. Since principal components are ordered with respect to their proportion of explained variance, and all of the mentioned digits have rounded patterns in common and they are similar, the patterns belonging to them corresponds to high proportion of variance in the data. Since the mentioned digits corresponds to half of the digits, as I expected, the patterns that they have in common are at the top of principal components that have most of the variance in the training data. In addition, other digits can also be seen in top eigenvectors, as black color corresponds to one side of the eigenvector and white color corresponds to other side of the eigenvector. For example, one can see digit 1 along with 0 in the first eigenvector, and 7 along with 0 in the fifth eigenvector.

1.3. In this part of the question, I used different subspaces of PCA to project the training data onto it and trained a Gaussian classifier with the projected data in each subspace that I created. For Gaussian classifier, I used Scikit-learn library's Gaussian classifier implementation. First, I determine the different subspaces that I will perform experiments with, and I decided to use 100 different subspaces with dimensions 2, 4, 6, …, 200 to measure the effect of dimensionality to classification more clearly. For each dimension, first, I get the principal components of the training data corresponding to dimension specified, then, I projected the training data and test data onto principal components specified. I trained a Gaussian classifier with the projected train data and using the Gaussian classifier that I trained, I predicted the labels of digits for both the projected train data and projected test data. I recorded the misclassification error for projected train data and projected test data for each of the dimensions I specified.

1.4. In this part of the question, I used the data that I recorded for classification error, and I shown the results on two separate plots, one for training data and another one for test data. The results can be seen in *Figure 5* and *Figure 6*.

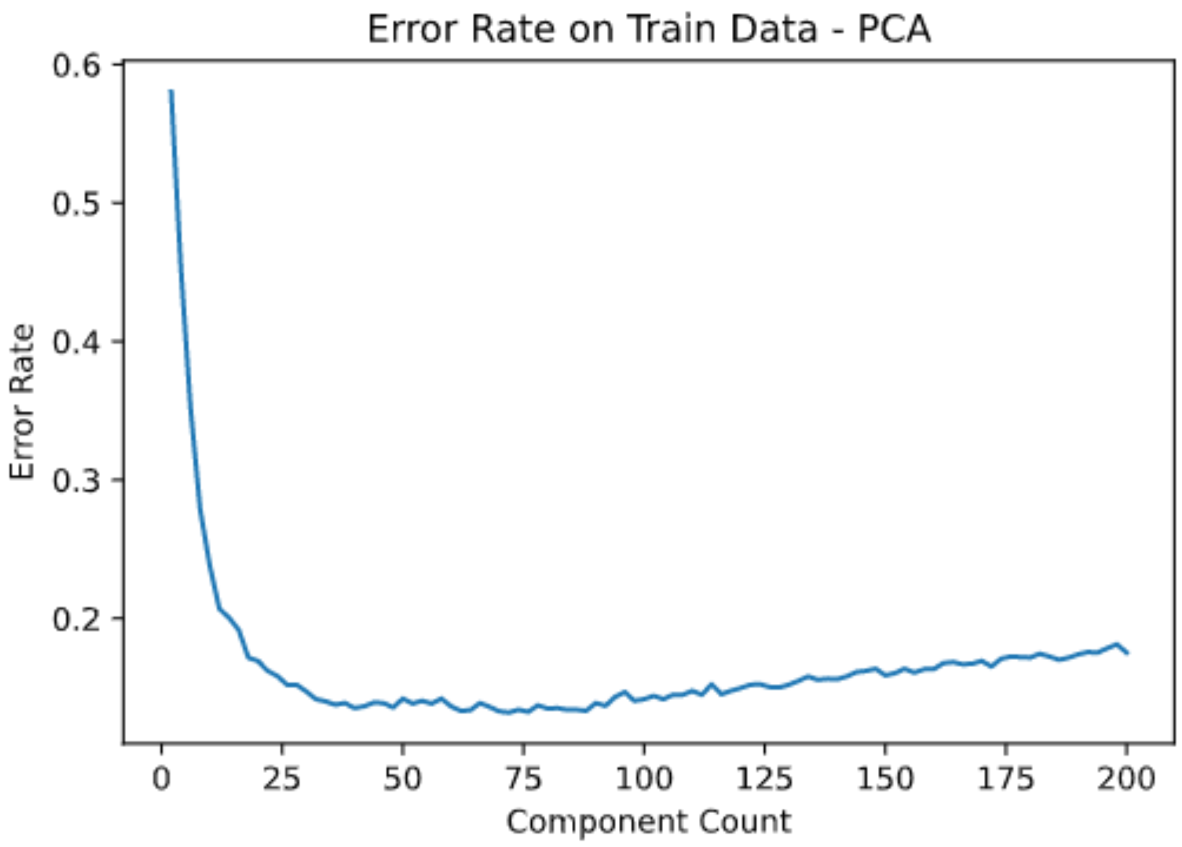

Figure 5: Classification Error vs. the Number of Components Used for Training Data

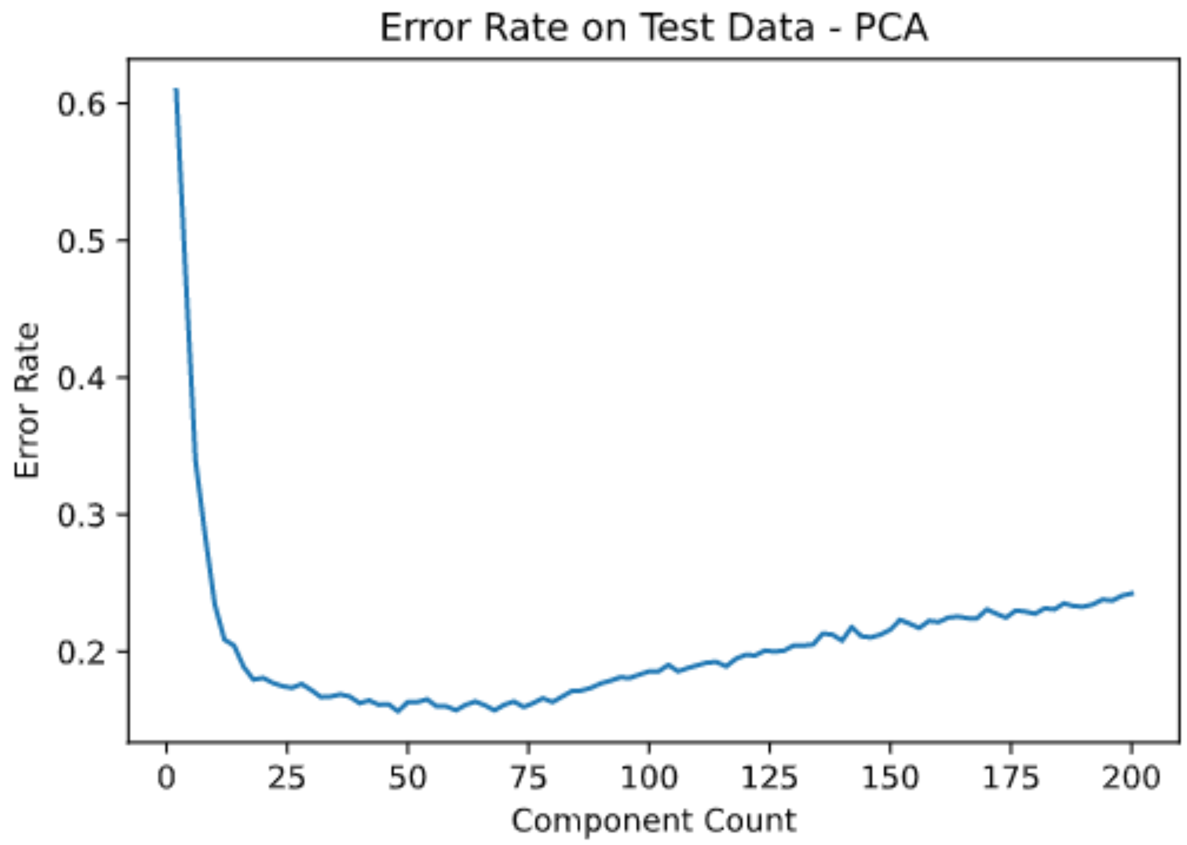

Figure 6: Classification Error vs. the Number of Components Used for Test Data

As it can be seen *Figure 5*, there is a sharp drop in the error rate for training data between the component counts 2 and 20. After using 15 principal components, the error rate fluctuates around 10% and after 75 principal component count, the error rate mostly increases. In other words, after a certain number of features, the performance of the model decreases rather than increasing as expected. As it can be seen in *Figure 6*, similar to *Figure 5*, the error rate for test data decreases sharply between the component counts 2 and 20. The error rate fluctuates around 10% between component counts between 15 and 75. After 75 component count, the error rate mostly increases while fluctuating.

Until the count of principal components used is 100, the models performed similarly in both test data and training data. When component count exceeds 100, the classification errors differ significantly regarding test data and training data. It can be considered normal as model knows the patterns in the training set well and the model will perform poorly on test data compared to training data as expected. When number of principal components used is 200, the misclassification rate for training set is approximately 15% whereas the misclassification rate for test data is approximately 25%. In addition, one of the main concerns of the PCA method is that finding the dimensions that will reduce the reconstruction errors and not the classification errors. Also, as the curse of dimensionality, adding more features to the model does not improve the performance of the model as more features means more complex model and it introduces the curse of dimensionality as expected.

For the test data and training data that we have for this assignment, also considering the splitting procedure, the optimal principal component count for a Gaussian classifier is found to be 48 by looking at *Figure 6*, which approximately corresponds to 12% of original feature count.

Question 2

In this question, I lowered the feature dimension of the data which is 400 by using Fisher linear discriminant analysis (LDA). I used LDA implementation from Scikit-learn library, Then, using the low dimensional data compared to original data, I trained a Gaussian classifier to measure the effect of dimensionality to performance of the classification.

2.1. In this part, I used LDA on training data which consists of 2500 samples, and I obtained new bases to project data onto it. The training data have 10 digits in total, I had a new set of bases which consists of 9 new hence we would have at most 9 bases. Then, I displayed these bases as images. The resulting images can be seen in *Figure 7*.

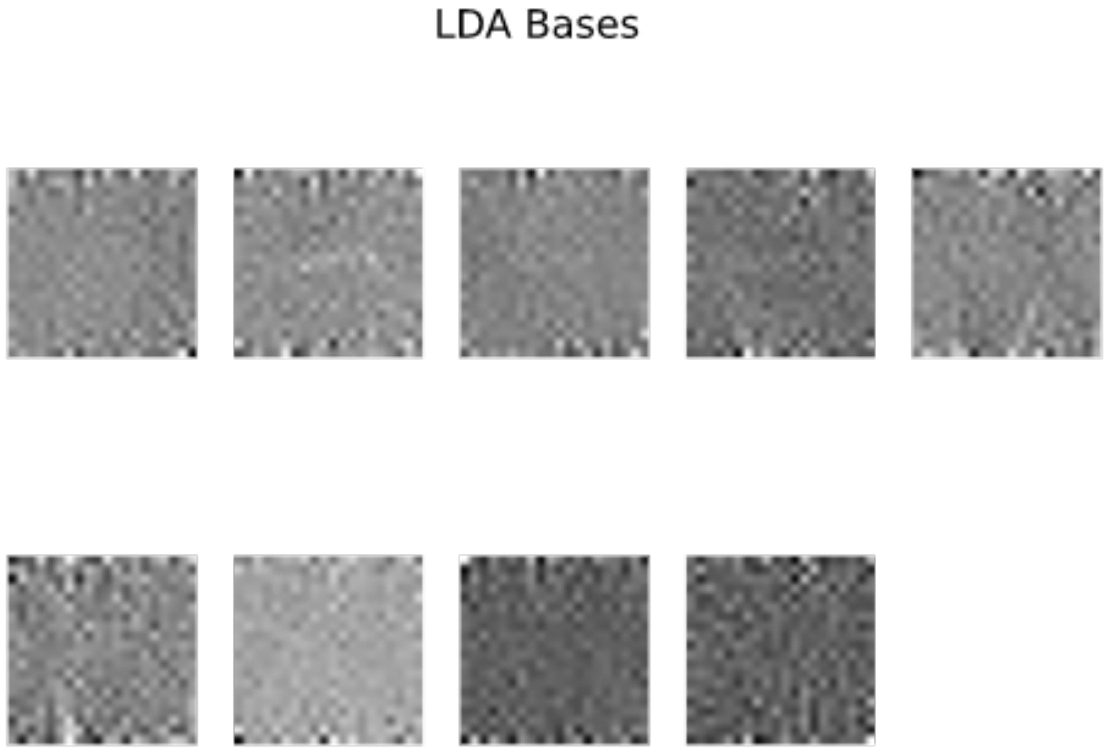

Figure 7: New Set of Bases Obtained by LDA

Unlike PCA, which seeks for the direction that minimizes the reconstruction error, LDA seeks for the directions that minimizes the separation or classification error. When we examined the eigenvectors in *Figure 4*, since top eigenvectors corresponds to higher eigenvalues and thus higher proportion of explained variance. Since LDA bases do not reflect the vectors that carry most of the information but they reflect the vectors that best separates the data, LDA bases in *Figure 7* cannot be identified as digits unlike PCA bases in *Figure 4*. In short, we have 9 new bases obtained by LDA, and these new bases are the vectors that best separates the data with respect to labels. Therefore, LDA bases are not similar to any of the patterns that digits have as expected, and they show patterns that cannot be identified and recognized unlike PCA.

2.2. In this part of the question, I used different subspaces of LDA to project the training data onto it and trained a Gaussian classifier with the projected data in each subspace that I created. For Gaussian classifier, I used Scikit-learn library's Gaussian classifier implementation. I used dimensions between 1 and 9 to measure the effect of dimensionality to classification. For each dimension, first, I get the vectors of the training data corresponding to dimension specified, then, I projected the training data and test data onto vectors specified. I trained a Gaussian classifier with the projected train data and using the Gaussian classifier that I trained, I predicted the labels of digits for both the projected train data and projected test data. I recorded the misclassification error for projected train data and projected test data for each of the dimensions I specified.

2.3. In this part of the question, I used the data that I recorded for classification error, and I shown the results on two separate plots, one for training data and another one for test data. The results can be seen in *Figure 8* and *Figure 9*.

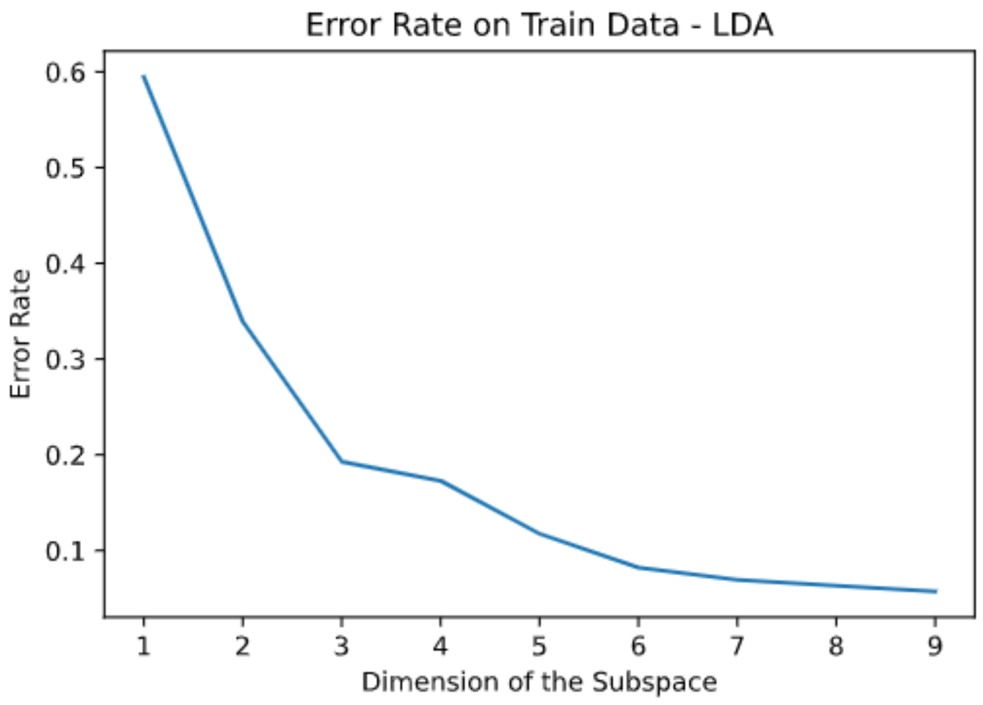

Figure 8: Classification Error vs. Dimension of Subspace - Training Data

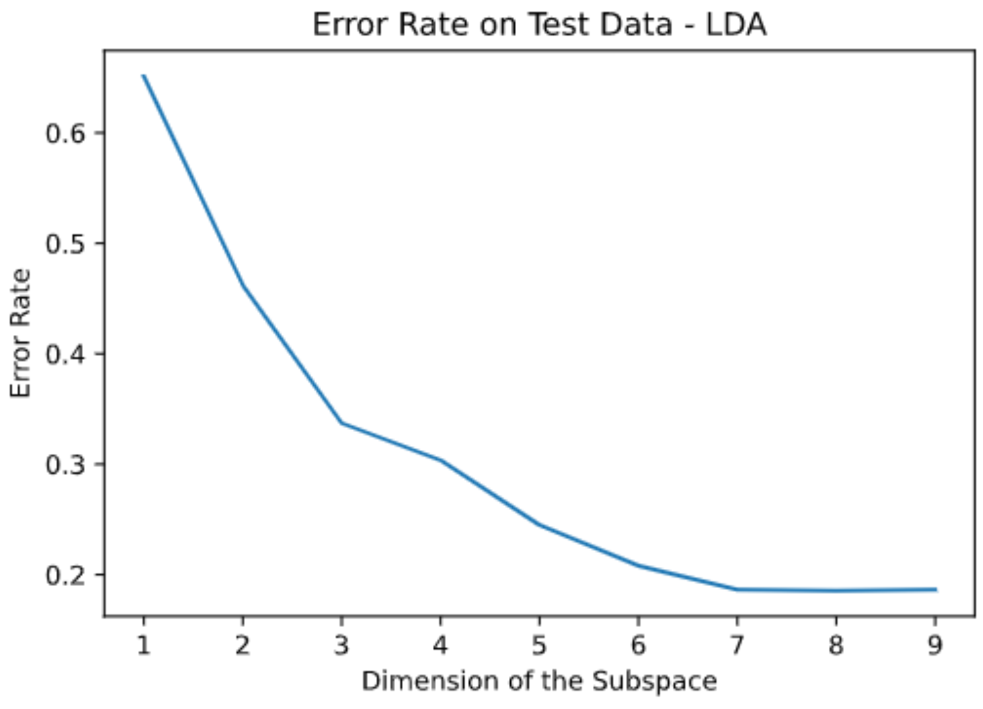

Figure 9: Classification Error vs. Dimension of Subspace - Test Data

As it can be seen in *Figure 8*, the error rate of classification decreases continuously as the dimension of the subspace increases. With one dimensional subspace, the error rate of classification is approximately 60%, and with nine dimensional subspace, the error rate of classification is approximately 5%. Although there is not a considerable difference on error rates between the subspaces that have dimensions between 6 and 9, it can be safely said that more LDA components leads to more accurate predictions on the given training data. Comparing the performance of Gaussian classifiers after PCA and LDA are applied, the best performance on training data obtained by using PCA is approximately the same with the best performance obtained by using LDA as it can be seen in *Figure 5* and *Figure 8*.

In addition, as it can be seen in *Figure 9*, the error rate of classification decreases across the subspaces that have the dimensions between 1 and 7, and the error rate of classification remains to be same across the subspaces that have the dimensions between 7 and 9. With one dimensional subspace, the error rate of classification is approximately 65%, and with nine dimensional subspace, the error rate of classification is approximately 20%. Although there is not a considerable difference on error rates between the subspaces that have dimensions between 7 and 9, it can be safely said that more LDA components leads to more accurate predictions on the given test data. Comparing the performance of Gaussian classifiers after PCA and LDA are applied, the best performance on test data obtained by using PCA is better than the best performance obtained by using LDA as it can be seen in *Figure 6* and *Figure 9*. However, considering that the best performance obtained by using PCA was using approximately 48 principal components, which means the data was 48 dimensional. With using a subspace that is 9 dimensional, we can achieve worst but similar result compared to PCA with 48 principal components, which corresponds to a lower dimensional model, a less complex model and more efficient model in terms of memory space and time space during calculations. In addition, LDA model show more consistent result that PCA model since, as mentioned before, its main goal is finding the directions that best separates the data rather than reconstructing it as the main goal of PCA.

Question 3

In this question, I lowered the feature dimension of the data which is 400 by using two different approaches: Sammon's mapping and t-SNE, which are techniques that aims to visualize data in lower dimensions. With these two approaches, the dimension of the data reduced to 2 from 400. I used Sammon's mapping implementation from GitHub repository owned by Tom Pollard and t-SNE implementation from Scikit-learn library. Then, using the new bases that I obtain from these techniques, I projected the data and displayed the results in scatter plots respectively.

Sammon's mapping and t-SNE are dimensionality reduction techniques that are based on optimization. In other words, while reducing the dimension of the data, the algorithms try to minimize the cost of mapping, which is the distances between two points in original space and mapped space. They cannot ensure perfectly distanced mapped data, however, they lead to more clear visualization of the data.

• **Sammon's mapping:** It uses a cost function of euclidian distance that puts more importance on the distance between two points of the data in original space. Since it an iterative algorithm, Sammon's mapping is slow. As a result of my experiments, performing 1000 iteration takes approximately 1 hour. I tried different iteration counts as 250, 500 and 1000. Considering the running time, I only included 500 iterations in the code, the others are commented, so you can remove comments and execute them. However, the results for all iteration counts are included in the report and notebook file. Other than iteration counts, the other parameters are the same. The parameter inputdist was set to raw as the data given to the function contains raw data rather than pairwise distances. The parameter maxhalves was set to 20 as maximum number of step halvings. The parameter tolfun was set to 1e-9 as relative tolerance on objective function, which means when relative difference is less then 1e-9, the algorithm will stop. And init parameter was set to default besides the values of pca, cmdscale and random. The results can be seen in figures ranging from *Figure 10* through *Figure 22*.

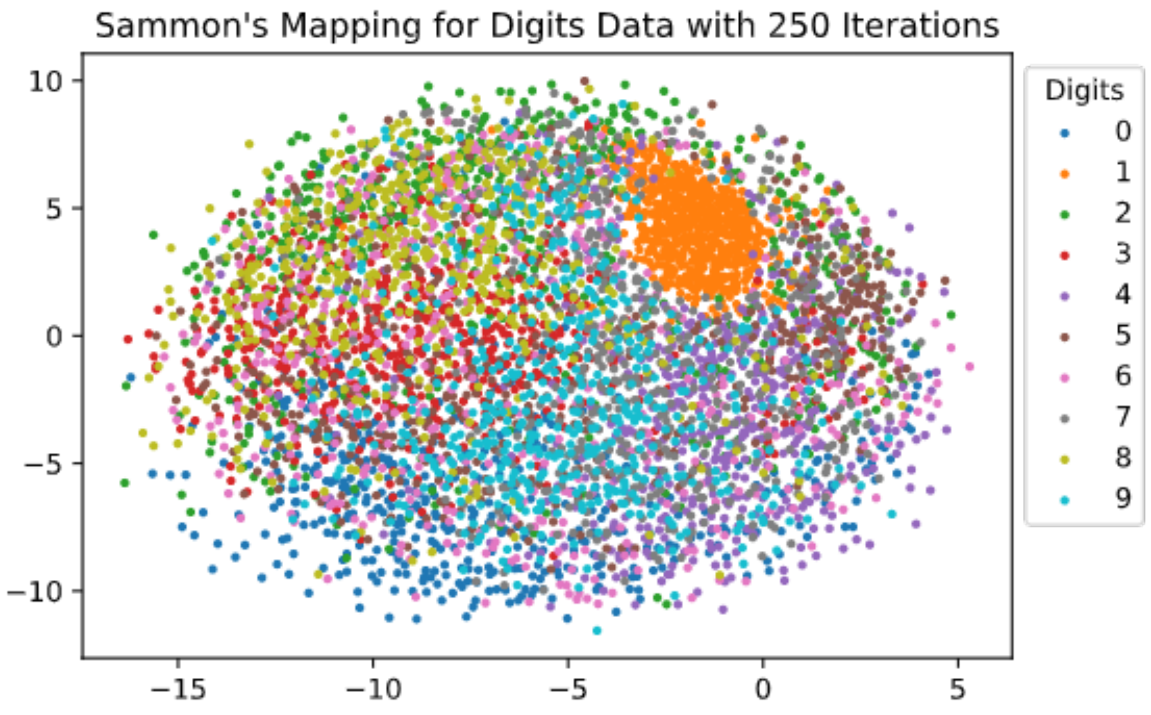

Figure 10: Sammon's Mapping for Digits Data with 250 Iterations

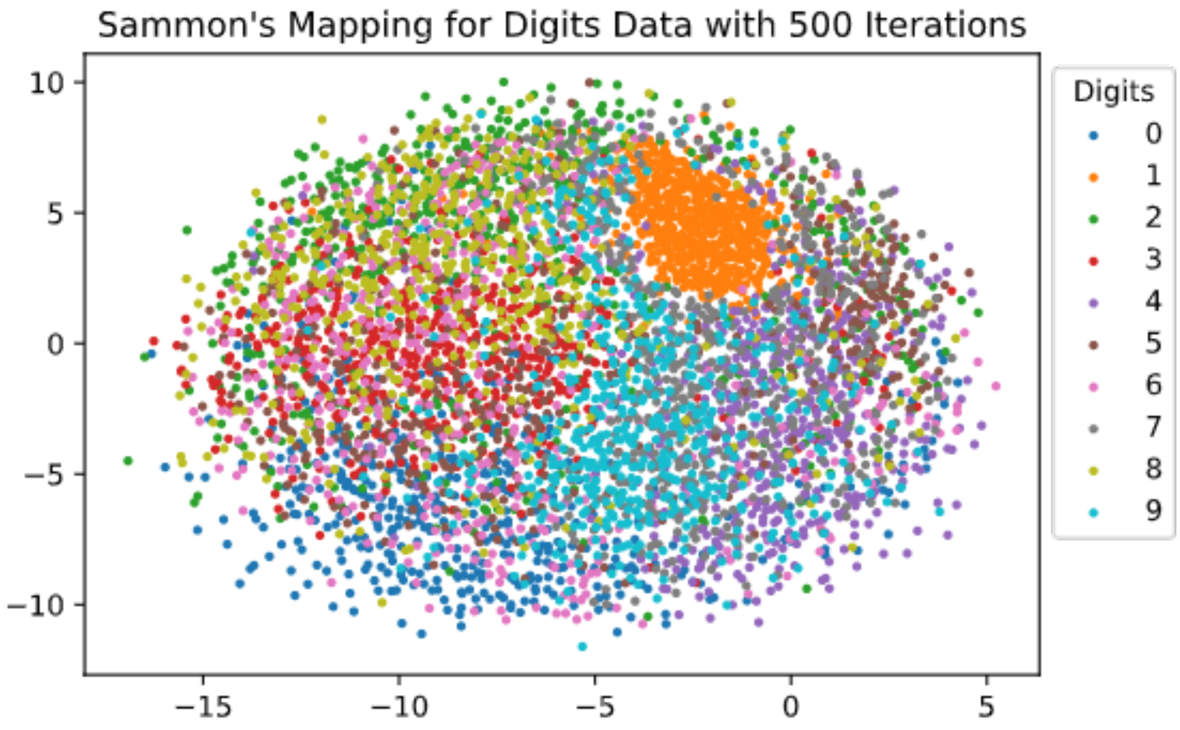

Figure 11: Sammon's Mapping for Digits Data with 500 Iterations

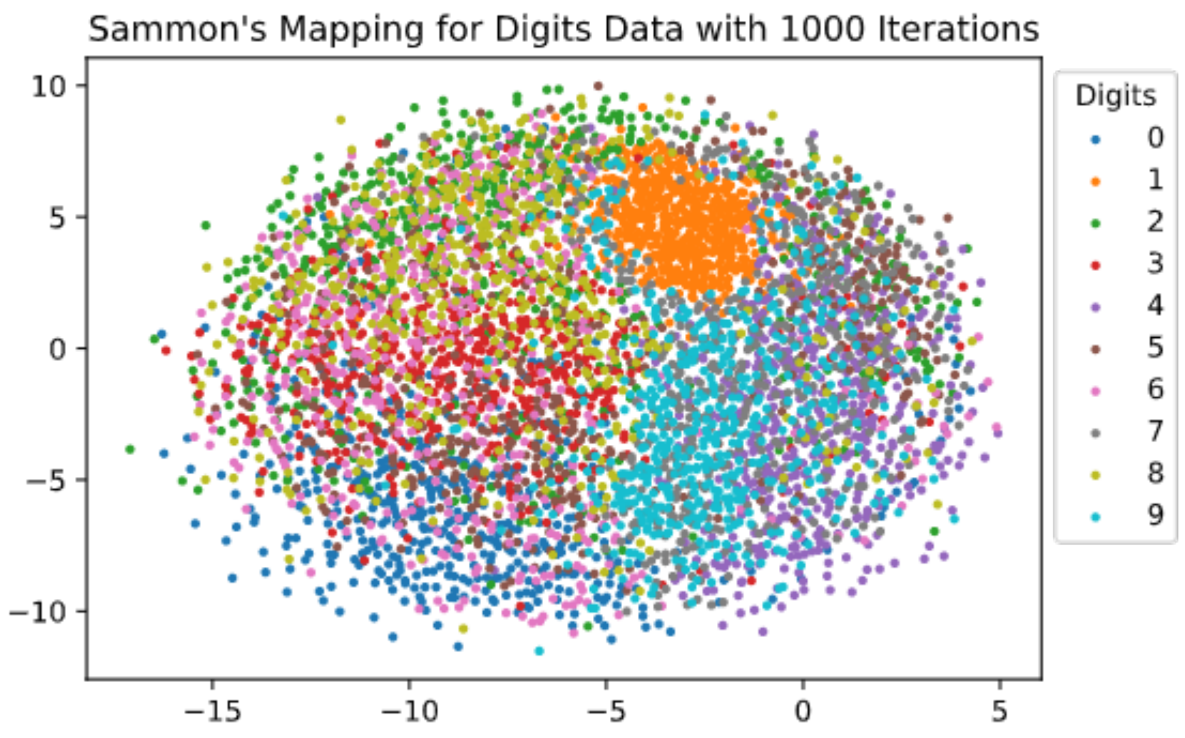

Figure 12: Sammon's Mapping for Digits Data with 1000 Iterations

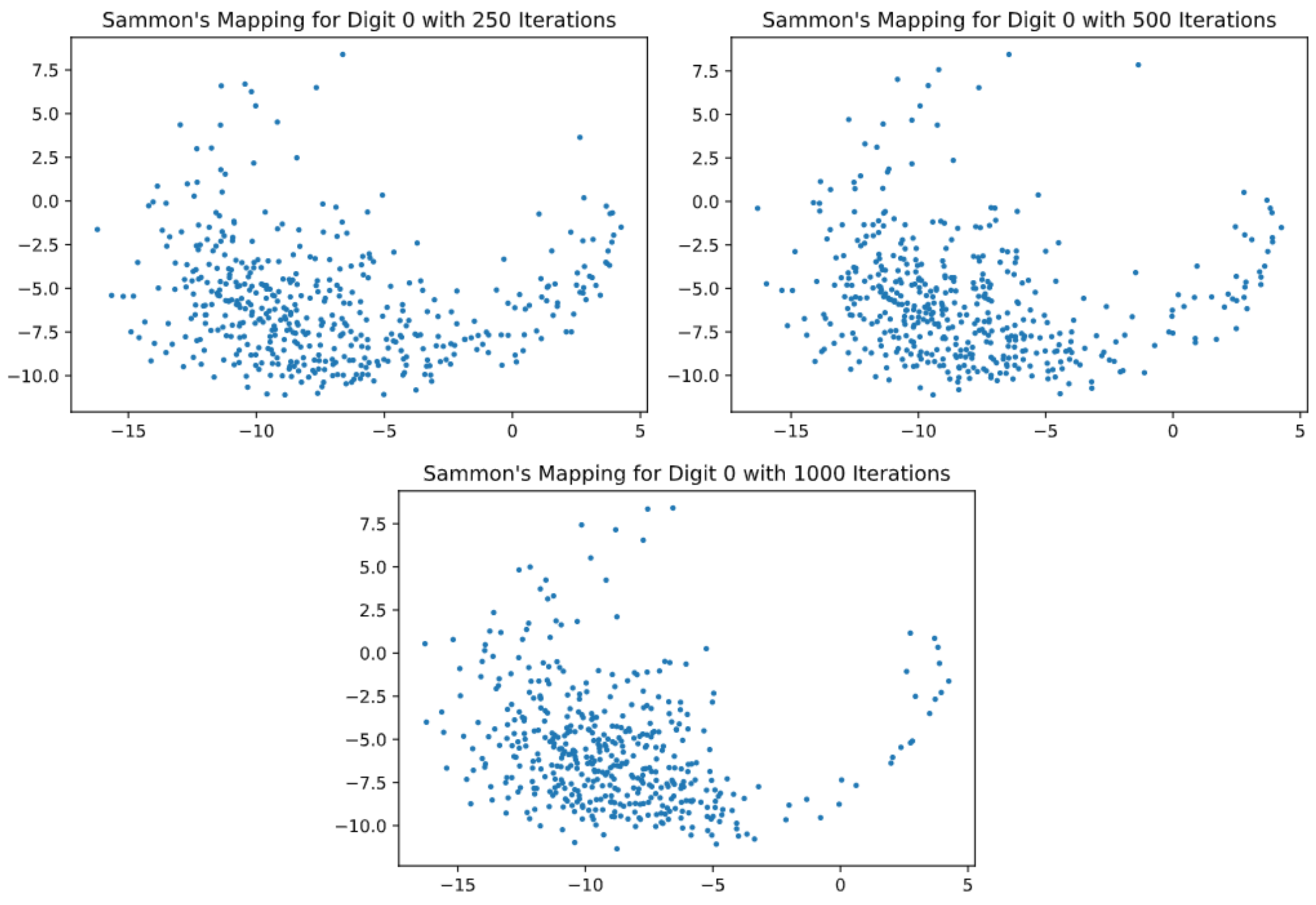

Figure 13: Sammon's Mapping for Digit 0

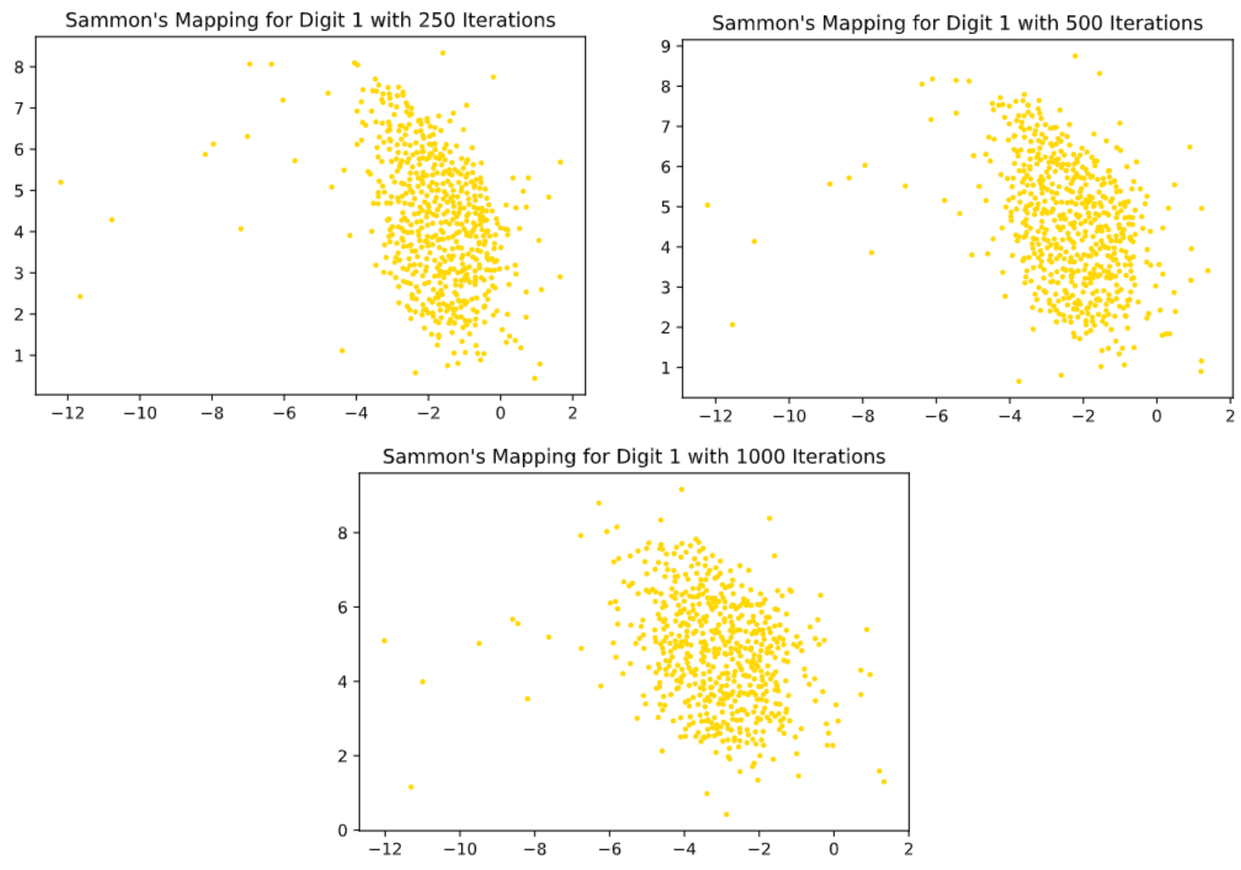

Figure 14: Sammon's Mappings for Digit 1

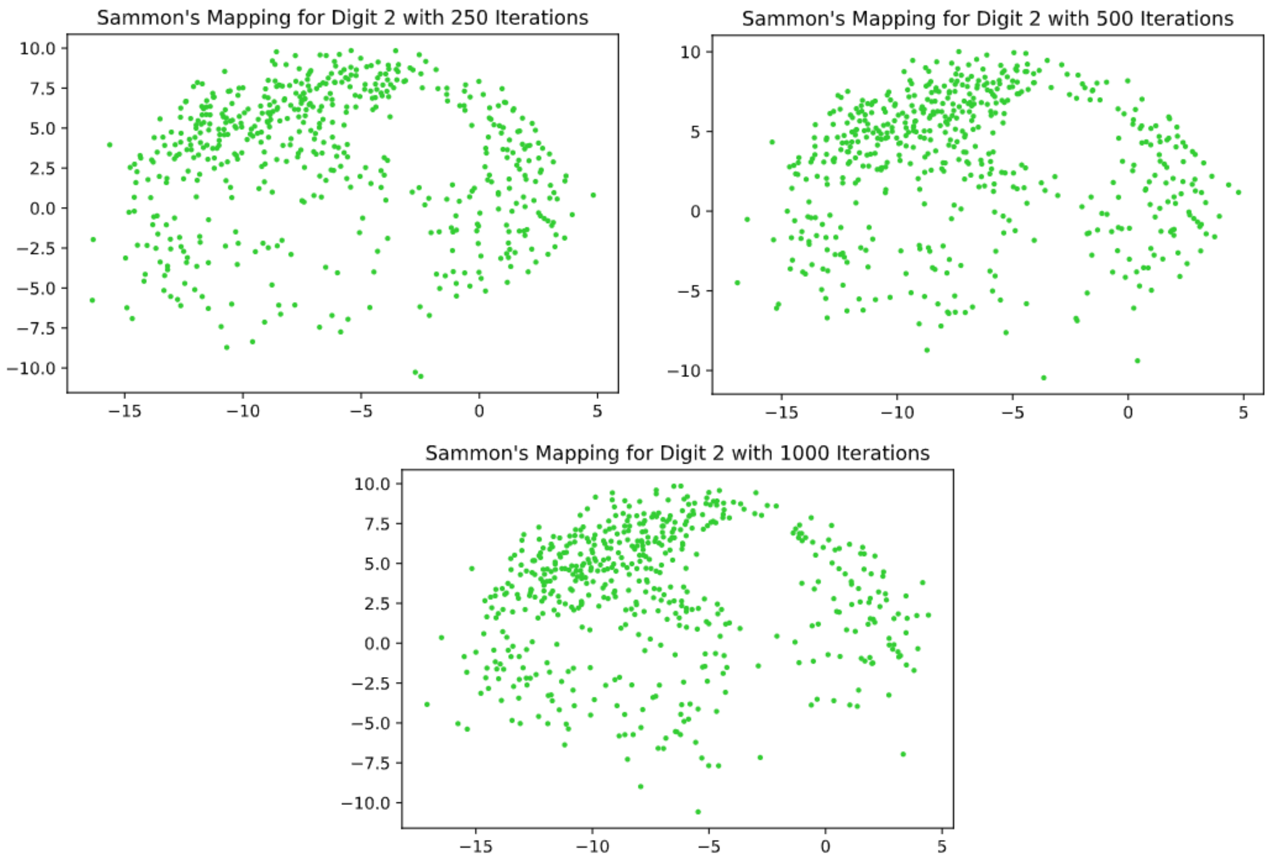

Figure 15: Sammon's Mappings for Digit 2

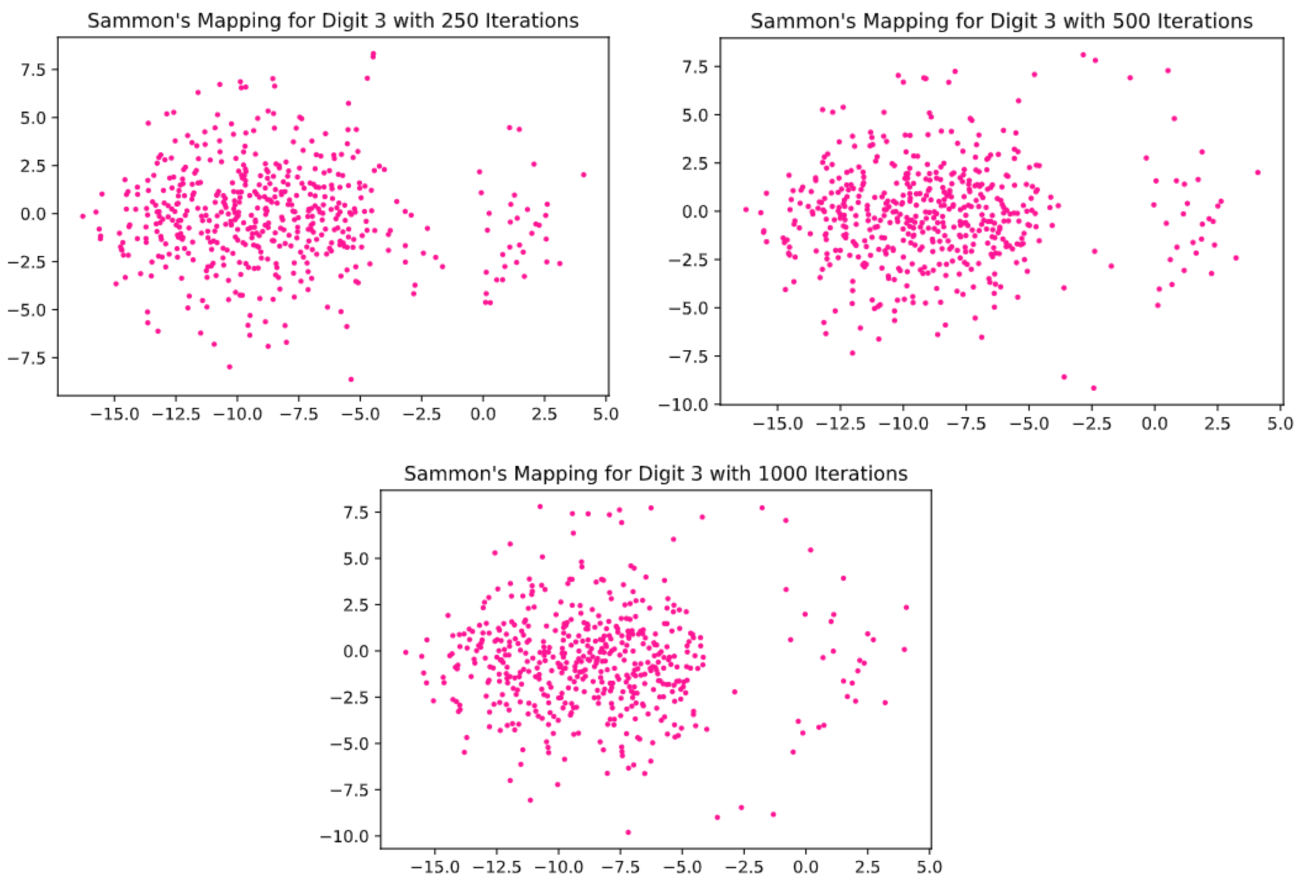

Figure 16: Sammon's Mappings for Digit 3

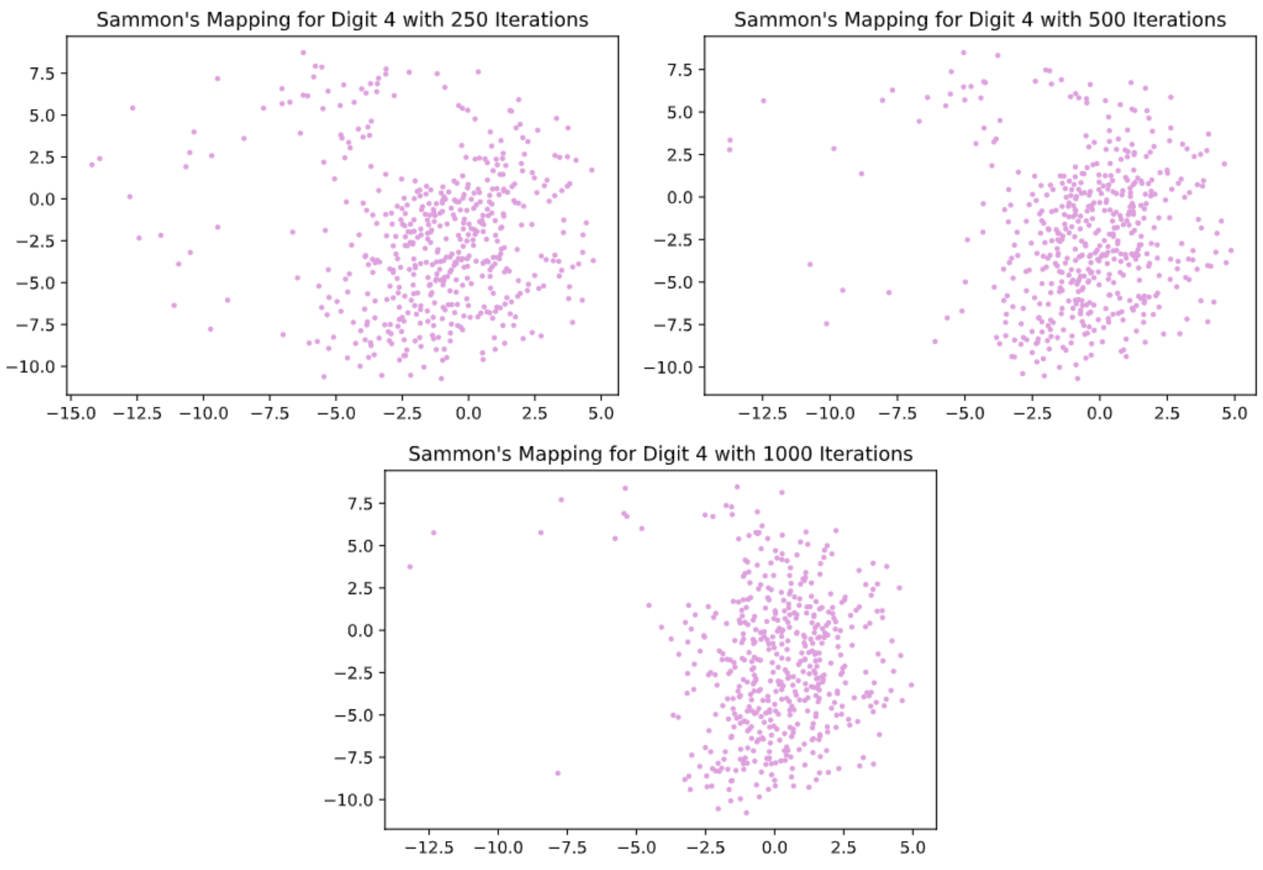

Figure 17: Sammon's Mappings for Digit 4

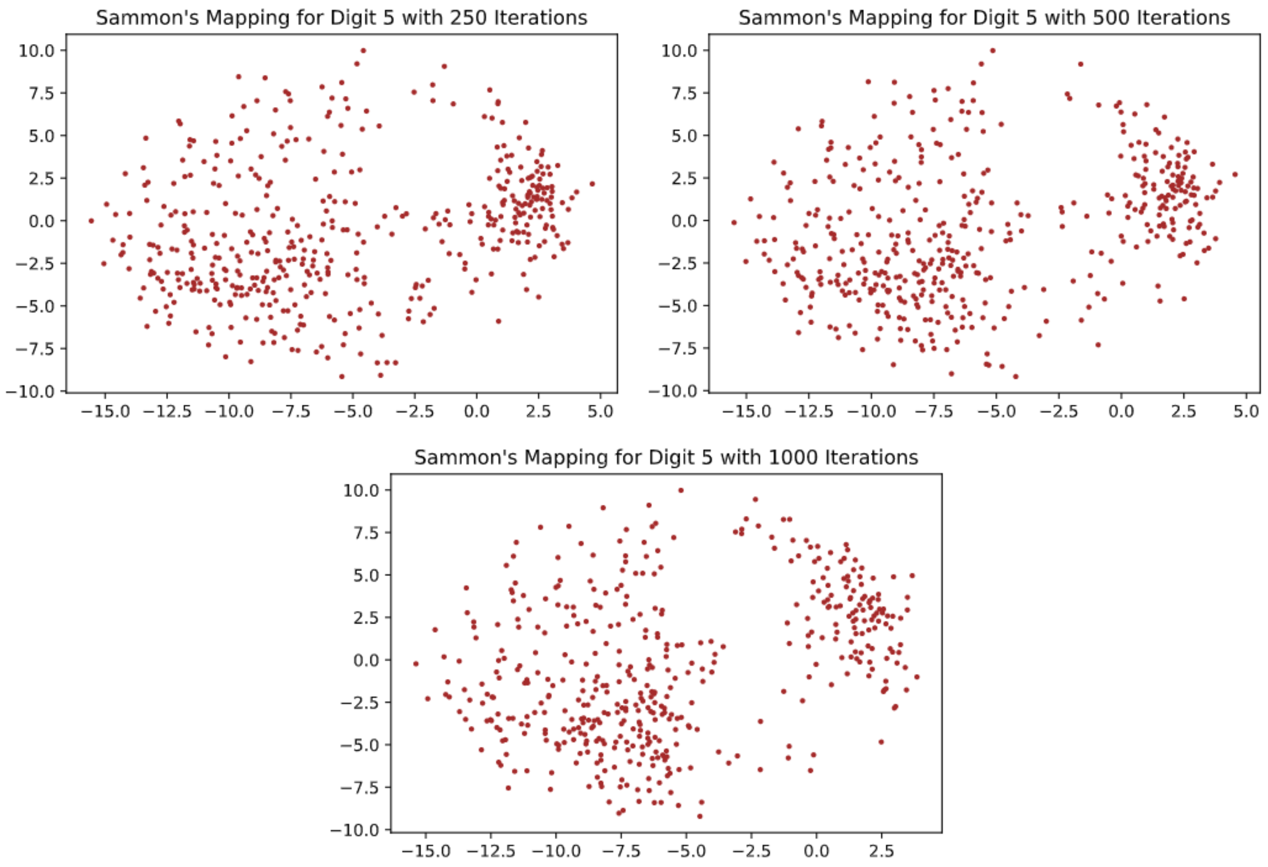

Figure 18: Sammon's Mappings for Digit 5

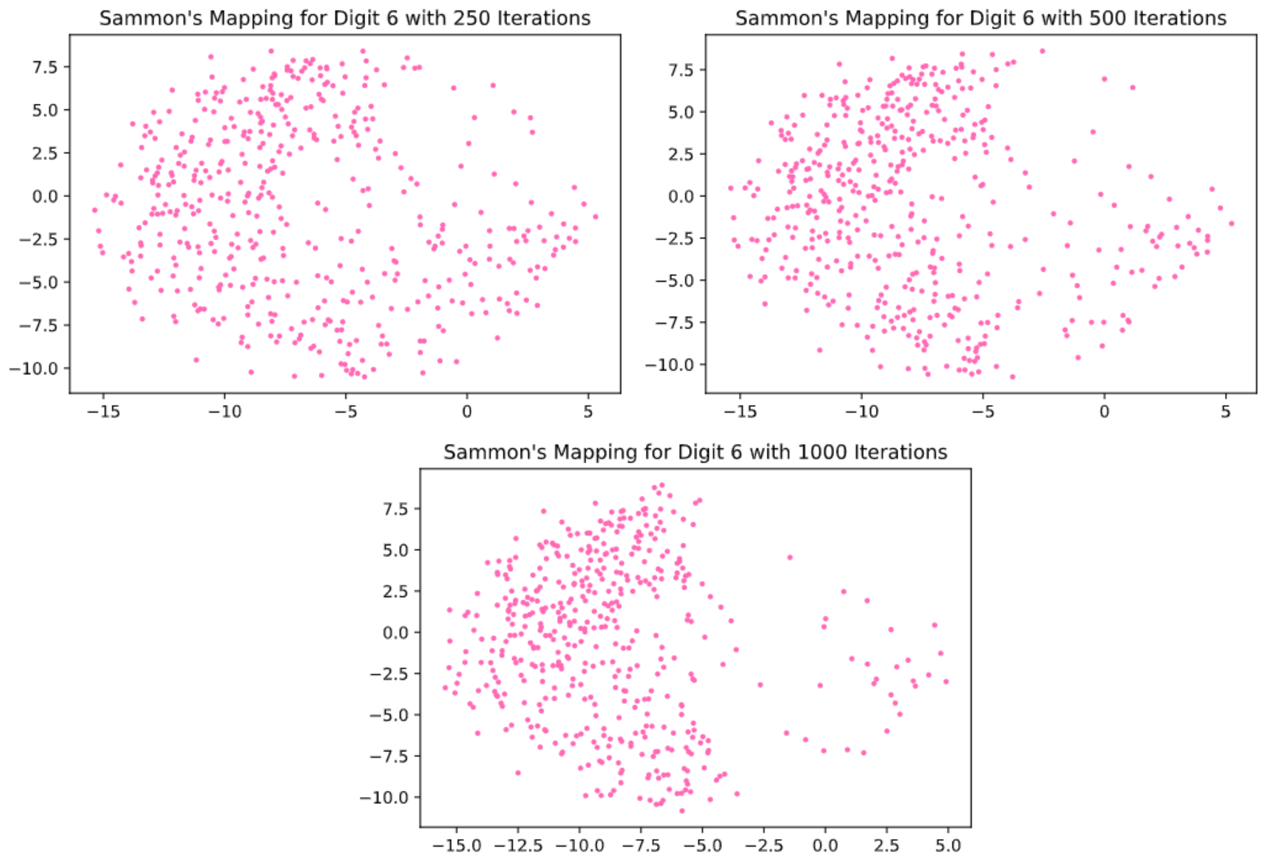

Figure 19: Sammon's Mappings for Digit 6

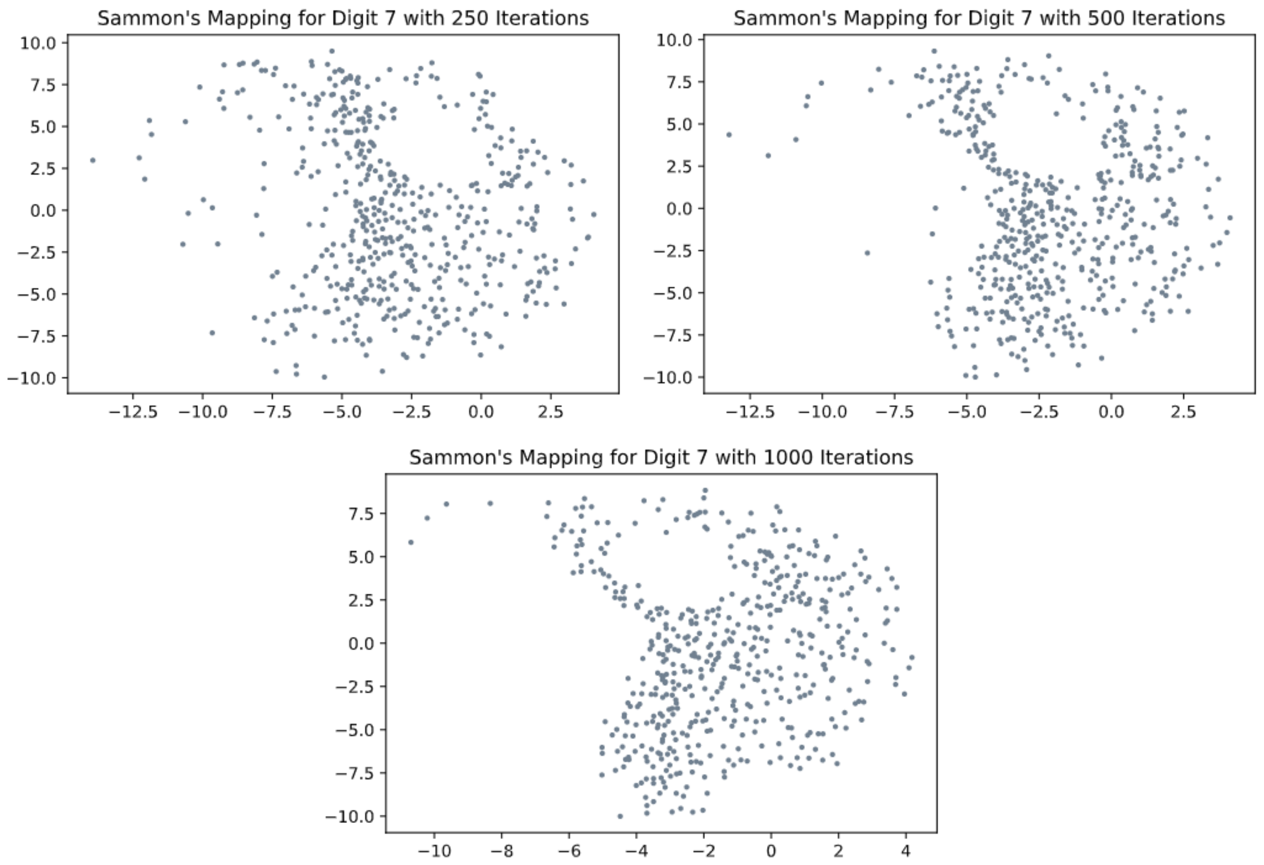

Figure 20: Sammon's Mappings for Digit 7

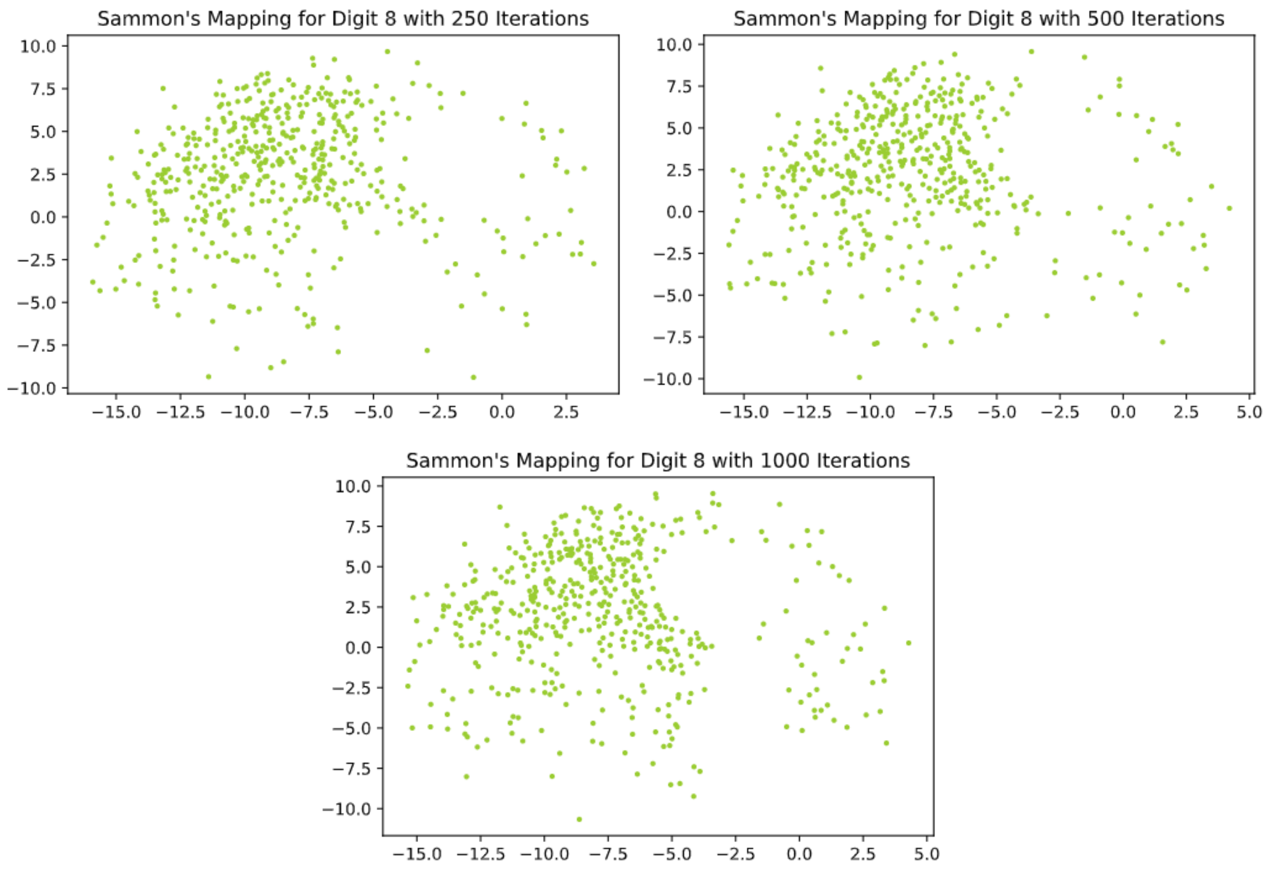

Figure 21: Sammon's Mappings for Digit 8

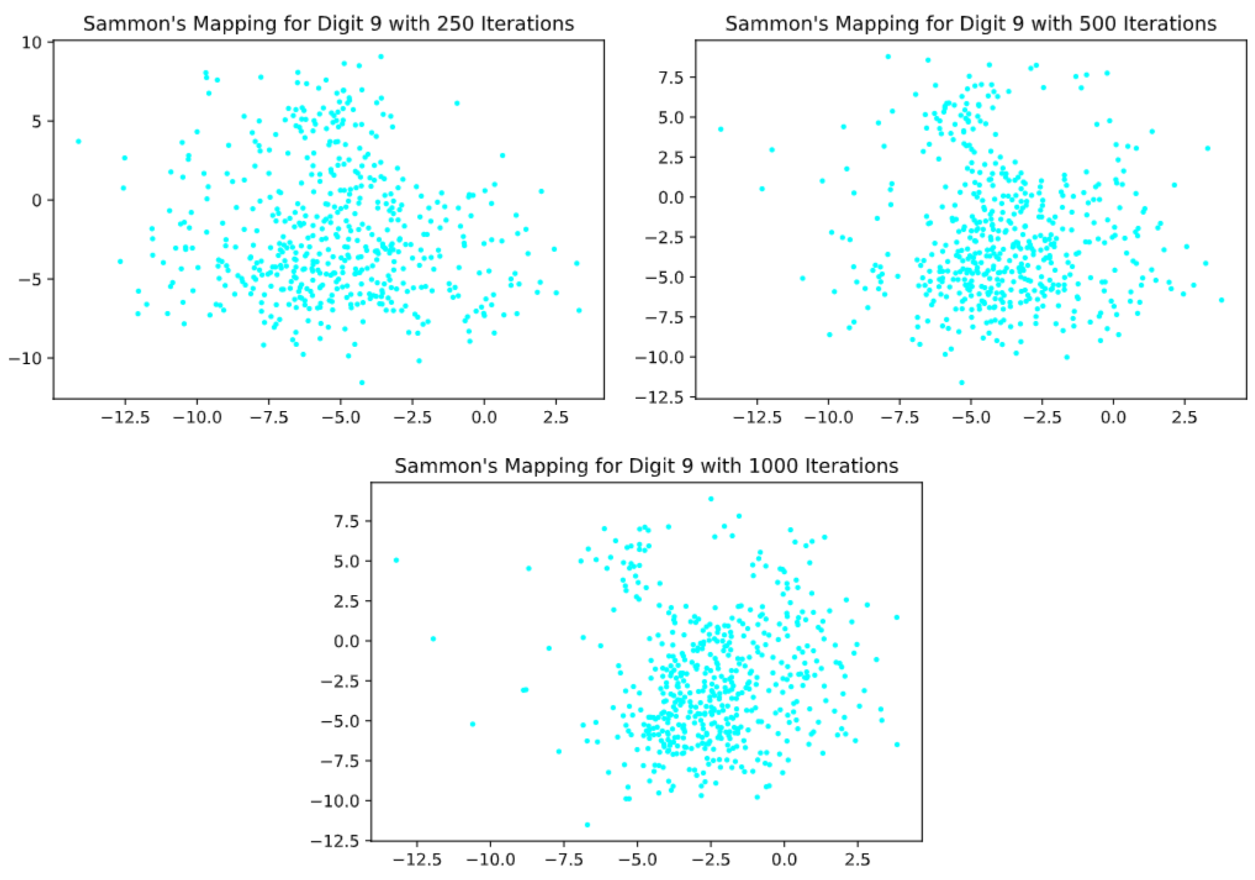

Figure 22: Sammon's Mappings for Digit 9

From the figures *Figure 10*, *Figure 11* and *Figure 12*, it can be said that the data is reduced to two dimensions and corresponding points are labeled with respect to their corresponding digits. The resulting mappings for different iterations are similar to each other i.e, the locations of colors corresponding to digits are nearly identical. In addition, in all of the figures *Figure 10*, *Figure 11* and *Figure 12*, although digits are accumulated in specific locations in the plots, the points in the graph mostly overlap and it is hard to distinguish the digits except for some of them such as 1. Considering this fact, I decided to plot the digits separately for each of the iteration counts to see the distinction between the digits as they can be seen in the figures *Figure 13* through *Figure 22*. As it can be seen from these figures, increased iteration count lead specific digits that have the same labels to be closer to each other. If the iteration is increased beyond 1000, we might see better results than we got with 1000 iterations. However, as mentioned before, these experiments took approximately 2 hours in total with a computer that has 16GB of RAM and 2.6GHz processor and thus Sammon's mapping requires considerable amount amount of computational power. It successfully reduced the dimension of the data from 400 to 2, although the distinction between the digits are not clear as I expected. However, it simplified the visualization of the data considerably.

• **t-SNE:** It is a popular technique in the era of deep learning and deep neural networks. It uses a cost function to minimize like Sammon's mapping, but it is more complicated than the cost function of Sammon's mapping as it involves exponentials and logarithms and it also deepens on conditional probabilities. In addition, optimization approach t-SNE is about keeping the topology of the data same. It tries to make each data point with equally number of neighbors and then tries to embed them. The performance of the algorithm when it comes to finding clusters in the data is considerably good, however, there is a drawback regarding getting stuck at local minima. The data that we have for this assignment is not large, it only have 5000 samples. If we had more samples, we might obtain better results. To have the best visualization of the data on a two dimensional data, I performed many experiments on the data with t-SNE that mainly involves changing the perplexity, and iteration count. The model have the following parameters as common: early exaggeration of 12, learning rate of 200, iteration count without progress of 300, minimum grad norm of 1e-7, metric of euclidean, embedding of random, method of Barnes-Hut, angle of 0.5, number of parallel jobs of None and square distances of legacy. In my opinion, the best result that I obtain can be seen in *Figure 23*, which has the perplexity of 5, iteration count of 5000 and random state of None. Unfortunately, since I did not set any value for the random state, I wasn't able to reproduce the same result with the same parameters.

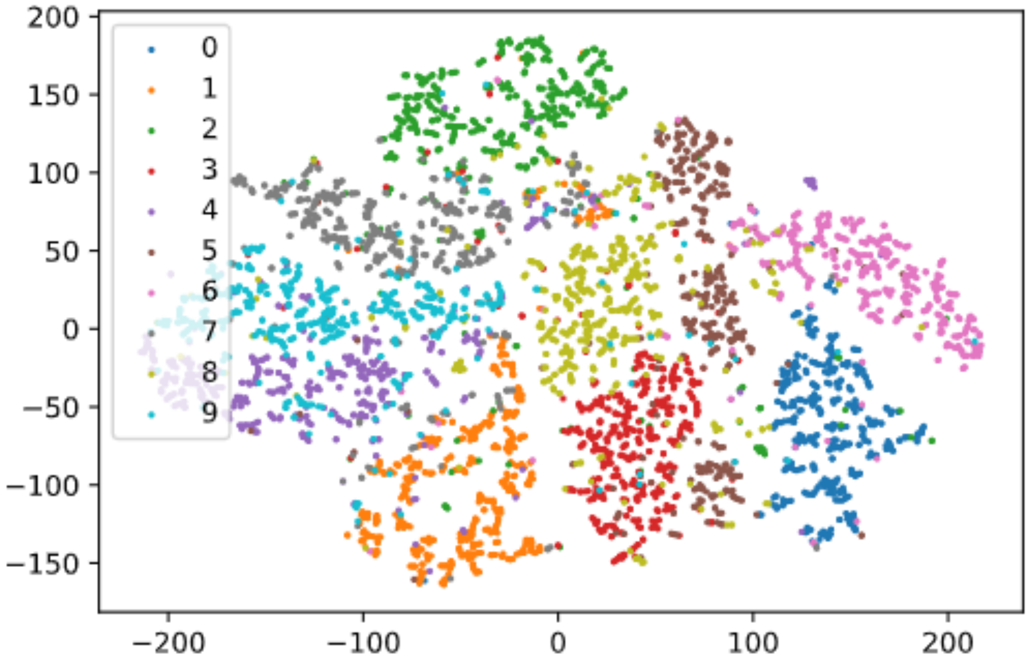

Figure 23: t-SNE with 5 Perplexity and 5000 Iterations

In addition, I included different setups in my code, and their results can be seen in figures *Figure 24*, *Figure 25*, *Figure 26* and *Figure 27*.

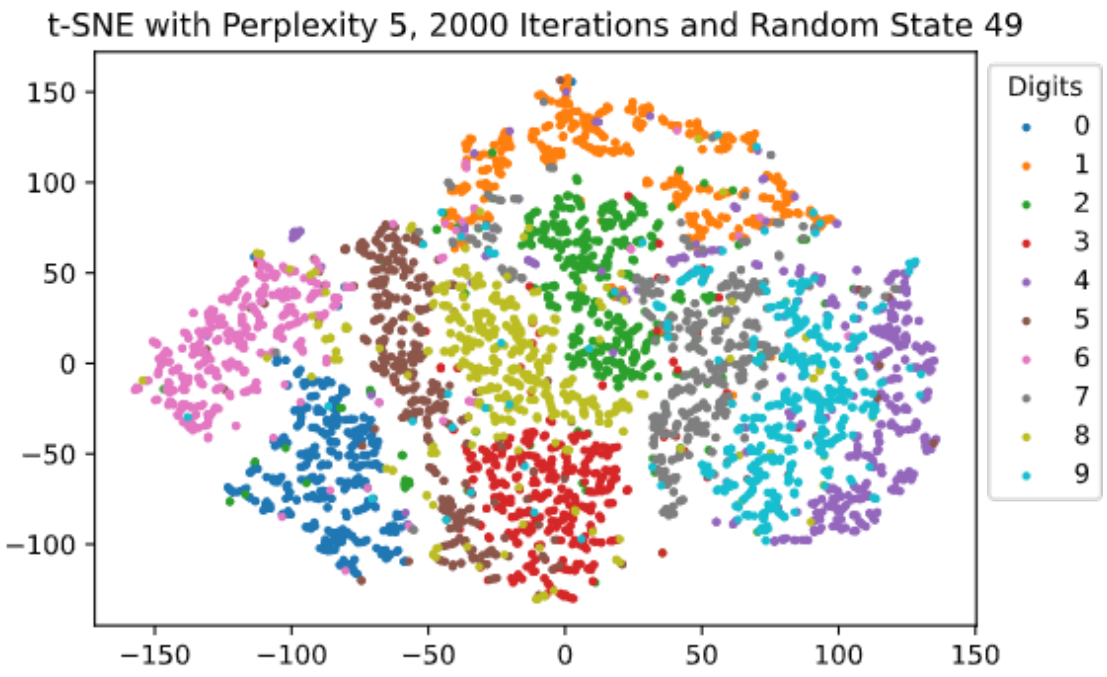

Figure 24: t-SNE Mapping with 5 Perplexity, 2000 Iterations and Random State 49

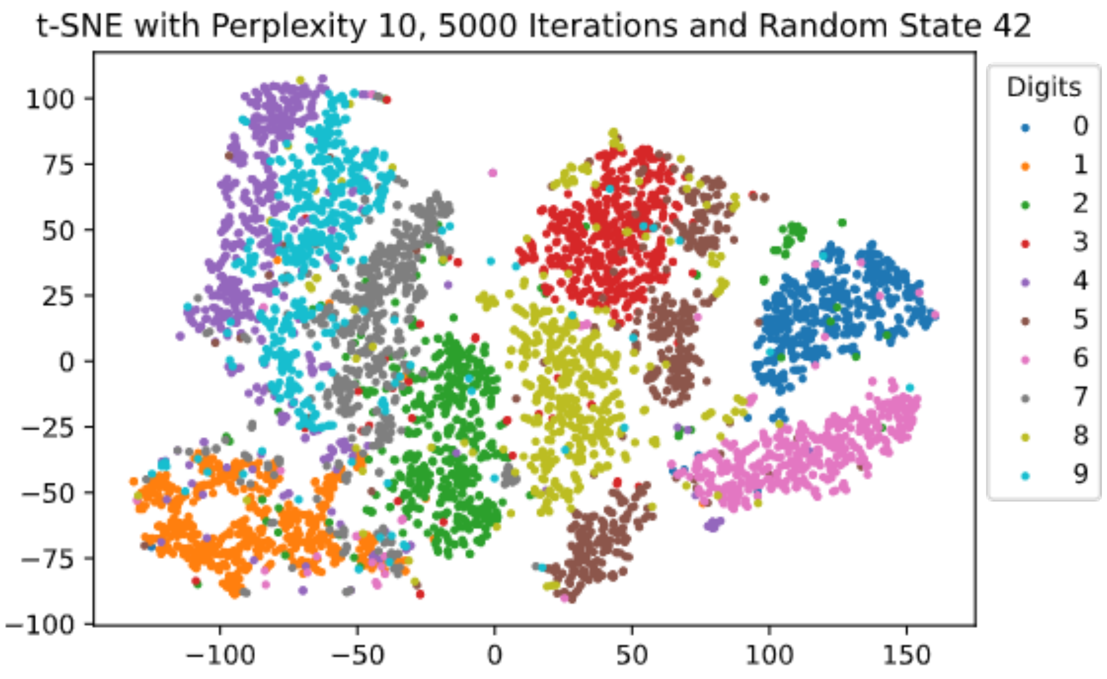

Figure 25: t-SNE Mapping with 10 Perplexity, 5000 Iterations and Random State 42

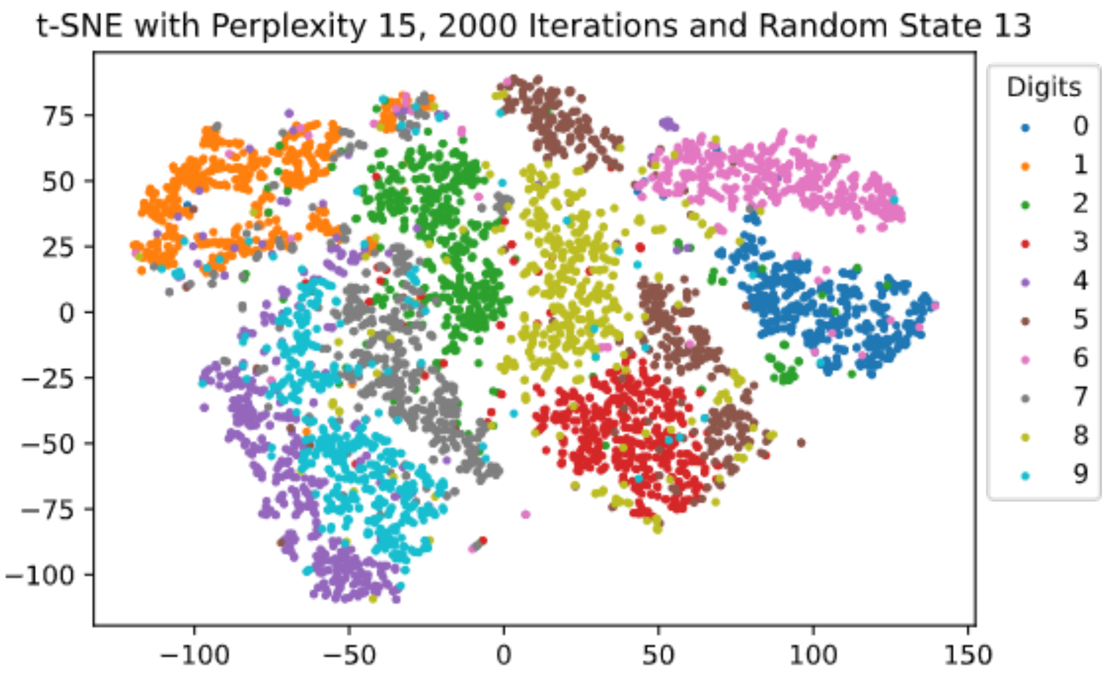

Figure 26: t-SNE Mapping with 10 Perplexity, 2000 Iterations and Random State 13

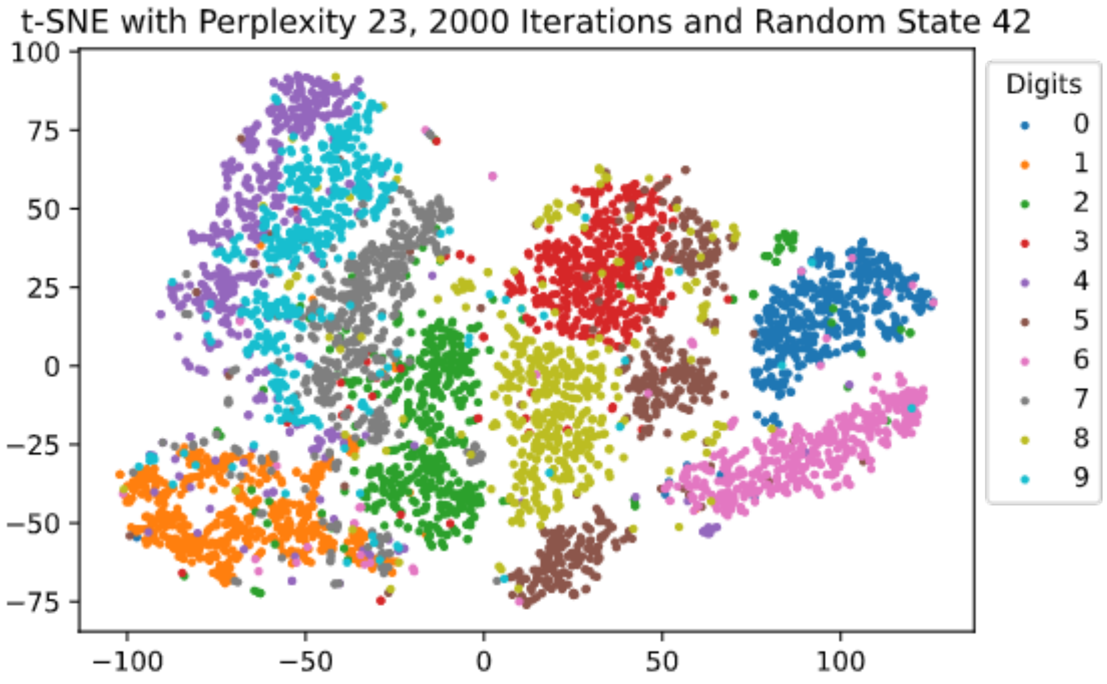

Figure 27: t-SNE Mapping with 10 Perplexity, 2000 Iterations and Random State 13

From the figures *Figure 23* through *Figure 27*, it can be said that the data is reduced to two dimensions and corresponding points are labeled with respect to their corresponding digits. The resulting mappings for different setups are similar to each other when they have the same random state, with small differences as it can be seen in *Figure 25* and *Figure 27*. Likewise, the resulting mappings for different setup differs from each other when the have different random state. This means that, the resulting mappings are mostly the result of the random state given to the model as it determines the neighboring process in the algorithm.

Considering the t-SNE plots, it can be said that the digits are separated and thus they can be distinguished regarding their corresponding labels. The algorithm can find the clusters in the data clearly. Comparing t-SNE and Sammon's mapping, t-SNE requires less computational power and hence it is faster, and it finds the cluster of the digits more clearly although there are outliers. I tried to visualize the points better and achieve a better separation between the clusters by also changing the other parameters like learning rate, early exaggeration and most importantly random state, but the results did not change considerably. As mentioned before, we have a small dataset which consists of 5000 samples. Increasing the dataset would surely increase the clustering performance of t-SNE and we would see more clear distinctions between clusters of the digits.

Tools and Other Implementations Used

- loadmat() function from scipy.io module to load the dataset
- train_test_split() function from sklearn.model_selection module to split the dataset into training set and test set.
- matplotlib.pyplot module to plot the necessary data
- PCA model from sklearn.decomposition to apply PCA
- LinearDiscriminantAnalysis model from sklearn.discriminant_analysis to apply LDA
- GaussianNB model from sklearn.naive_bayes to apply Gaussian Naive Bayes
- Metrics module from sklearn to measure the accuracy of the NB model on test data
- Sammon's Mapping implementation from GitHub repository owned by Tom Pollard to apply Sammon's Mapping
- TSNE implementation from sklearn.manifold to apply TSNE

References

- Virtanen, P., Gommers, R., Oliphant, M., Reddy, T., Cournapeau, E., Peterson, P., Weckesser, J., Walt, M., Wilson, J., Millman, N., Nelson, A., Jones, R., Larson, E., Carey, ., Feng, Y., Moore, J., Laxalde, D., Perktold, R., Henriksen, I., Quintero, C., Archibald, A., Pedregosa, P., & SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, 261–272. https://scipy.org/

- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://scikit-learn.org/stable/

- Hunter, J. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://matplotlib.org/

- Pollard, T. (2014). Sammon mapping in Python. https://github.com/tompollard/sammon.