Ihsan Dogramaci Bilkent University

GE 461: Introduction to Data Science

Dimensionality Reduction and Visualization

Name and Surname: Ali Aral Takak

Student ID: 22001758

Department: EEE

Introduction

This assignment focuses on exploring dimensionality reduction and visualization techniques for classification tasks. In this project, we use a subset of the Fashion-MNIST dataset, which contains 10.000 grayscale images of clothing items represented as 784-dimensional vectors. The primary objective is to investigate how reducing the data's dimensionality through methods PCA, LDA, and t-SNE affects the performance of a Gaussian classifier. By comparing the classification errors on different subspaces, the project aims to provide an insight into the pros and cons of these techniques.

Results of Question I

In the first question, we have started by preprocessing the data. In this step, the overall mean of the entire dataset is computed and displayed as an image, and then the data is centered by subtracting this mean from each sample. Next, Principal Component Analysis (PCA) is applied to the centered training data to extract up to 400 principal components. A scree plot is generated to show the distribution of eigenvalues, and the top 10 eigenvectors are visualized as images to illustrate the main directions of variance. Finally, the code defines functions to train a Gaussian classifier by calculating class-specific means and covariance matrices, and to predict labels using a vectorized approach based on log-likelihood computations. The classifier is then evaluated over various PCA subspace dimensions, with classification errors computed and plotted for both training and test sets to analyze the effect of dimensionality on performance. The figure given below displays the mean image of the dataset before centering:

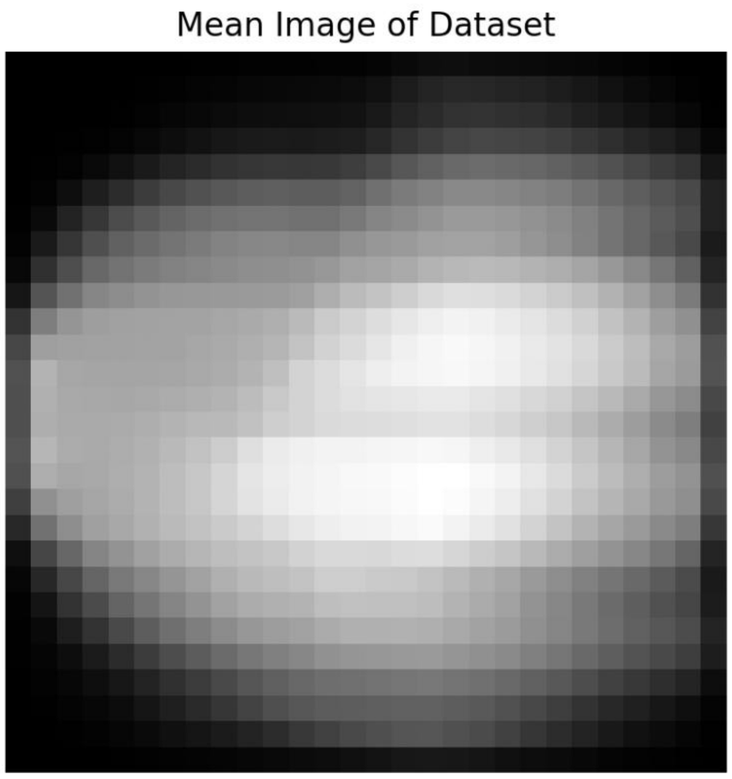


Figure 1: *Mean image of the dataset before centering.*

The figure provided below displays the Scree plot for the principal components and their respective Eigenvalues:

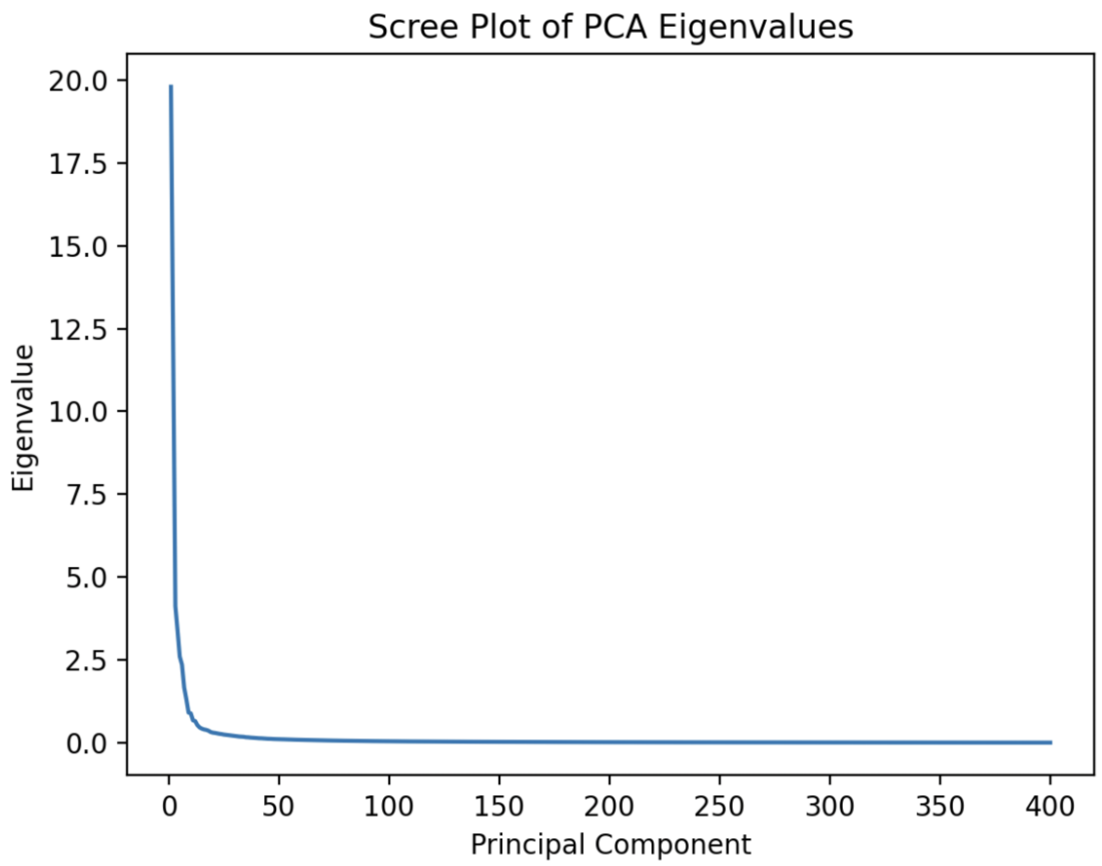


Figure 2: *Scree Plot of the Principal Component Analysis Eigenvalues.*

The figure provided below displays the top ten PCA Eigenvectors:

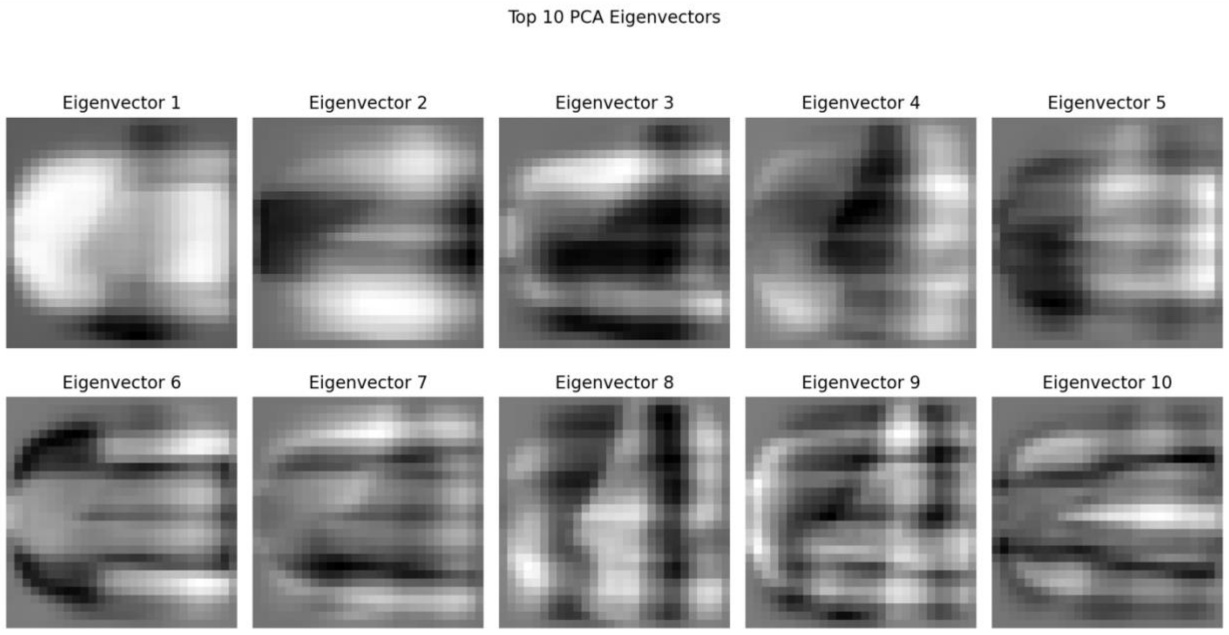
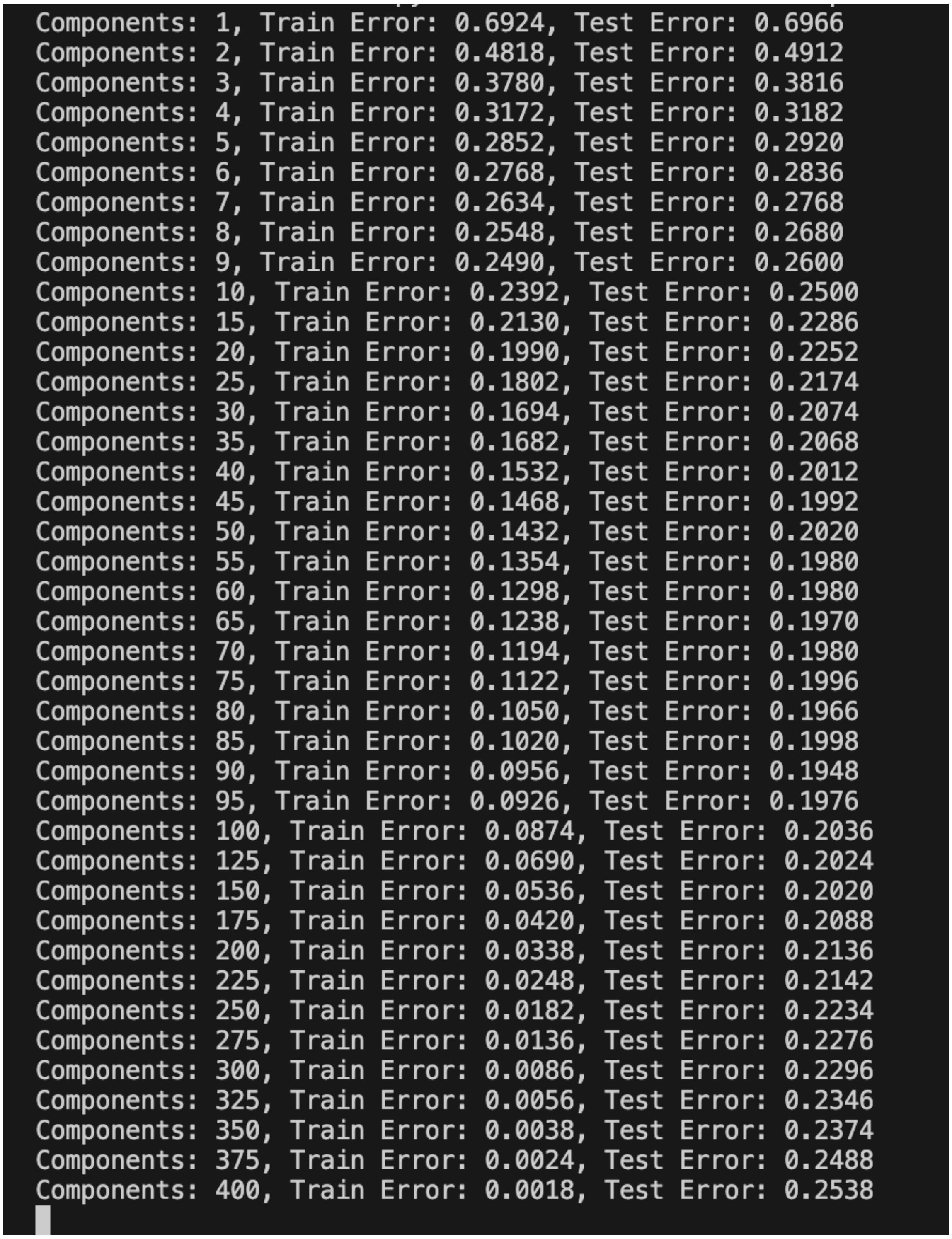


Figure 3: *Top ten Principal Component Analysis Eigenvectors in image format.*

Inspecting the top ten eigenvectors as visualized shows that they resemble the items in the FMNIST dataset, like t-shirts, shoes, trousers etc., but in a mixed way. However, we can infer that the data is represented in a composed manner in these eigenvectors.

Further, we continue to train the Gaussian classifier using multiple number of components. The figure given below displays training proces with various number of components, and their respective train error and test error:



```
Components: 1, Train Error: 0.6924, Test Error: 0.6966
Components: 2, Train Error: 0.4818, Test Error: 0.4912
Components: 3, Train Error: 0.3780, Test Error: 0.3816
Components: 4, Train Error: 0.3172, Test Error: 0.3182
Components: 5, Train Error: 0.2852, Test Error: 0.2920
Components: 6, Train Error: 0.2768, Test Error: 0.2836
Components: 7, Train Error: 0.2634, Test Error: 0.2768
Components: 8, Train Error: 0.2548, Test Error: 0.2680
Components: 9, Train Error: 0.2490, Test Error: 0.2600
Components: 10, Train Error: 0.2392, Test Error: 0.2500
Components: 15, Train Error: 0.2130, Test Error: 0.2286
Components: 20, Train Error: 0.1990, Test Error: 0.2252
Components: 25, Train Error: 0.1802, Test Error: 0.2174
Components: 30, Train Error: 0.1694, Test Error: 0.2074
Components: 35, Train Error: 0.1682, Test Error: 0.2068
Components: 40, Train Error: 0.1532, Test Error: 0.2012
Components: 45, Train Error: 0.1468, Test Error: 0.1992
Components: 50, Train Error: 0.1432, Test Error: 0.2020
Components: 55, Train Error: 0.1354, Test Error: 0.1980
Components: 60, Train Error: 0.1298, Test Error: 0.1980
Components: 65, Train Error: 0.1238, Test Error: 0.1970
Components: 70, Train Error: 0.1194, Test Error: 0.1980
Components: 75, Train Error: 0.1122, Test Error: 0.1996
Components: 80, Train Error: 0.1050, Test Error: 0.1966
Components: 85, Train Error: 0.1020, Test Error: 0.1998
Components: 90, Train Error: 0.0956, Test Error: 0.1948
Components: 95, Train Error: 0.0926, Test Error: 0.1976
Components: 100, Train Error: 0.0874, Test Error: 0.2036
Components: 125, Train Error: 0.0690, Test Error: 0.2024
Components: 150, Train Error: 0.0536, Test Error: 0.2020
Components: 175, Train Error: 0.0420, Test Error: 0.2088
Components: 200, Train Error: 0.0338, Test Error: 0.2136
Components: 225, Train Error: 0.0248, Test Error: 0.2142
Components: 250, Train Error: 0.0182, Test Error: 0.2234
Components: 275, Train Error: 0.0136, Test Error: 0.2276
Components: 300, Train Error: 0.0086, Test Error: 0.2296
Components: 325, Train Error: 0.0056, Test Error: 0.2346
Components: 350, Train Error: 0.0038, Test Error: 0.2374
Components: 375, Train Error: 0.0024, Test Error: 0.2488
Components: 400, Train Error: 0.0018, Test Error: 0.2538
```

Figure 4: *Train and test error on various numbers of components.*

The figure given below displays the plot for training error versus number of PCA components:

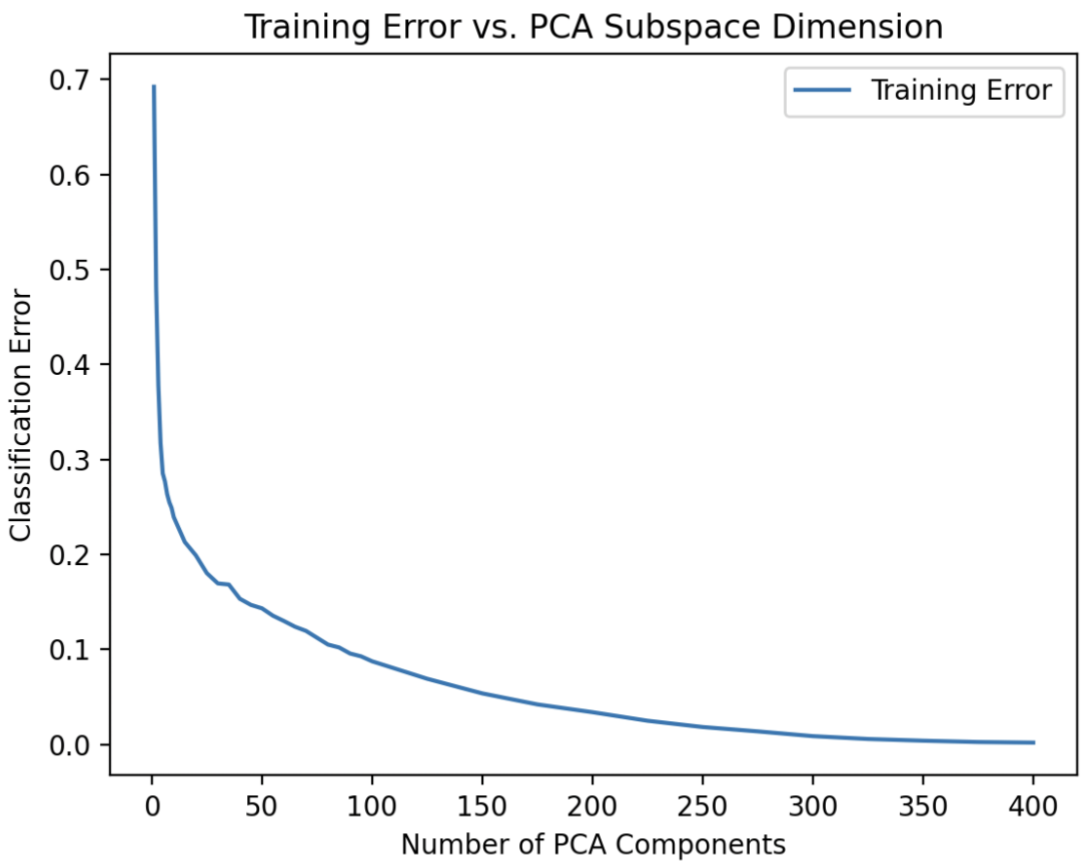


Figure 5: *Training error versus PCA subspace dimension.*

The figure given below displays the plot for test error versus number of PCA components:

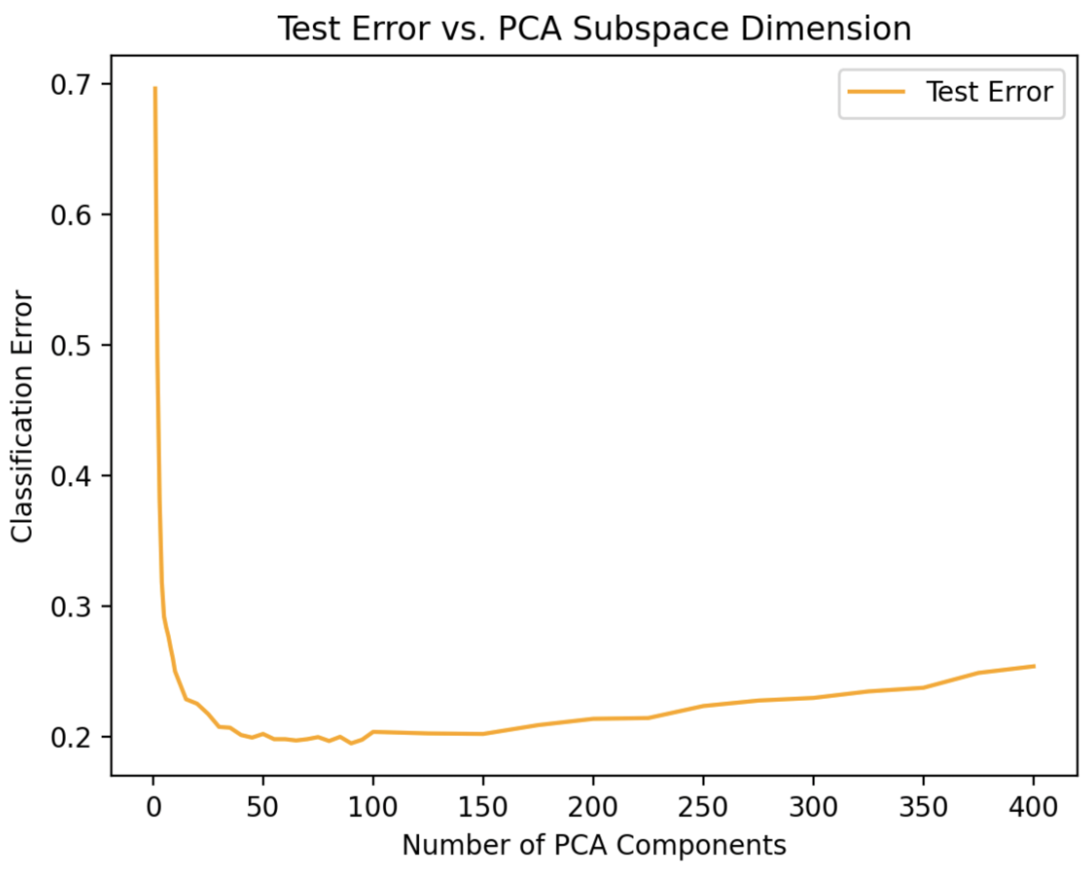


Figure 6: *Test error versus PCA subspace dimension.*

These results yield that as the number of PCA components increases, the training error steadily decreases, almost approaching zero at high dimensions. However, while initially the test error also decreases, reaching its lowest value around 90 components, beyond this point, the test error begins to increase gradually. We can infer that retaining more components allows the model to fit the training data more precisely, it also leads to overfitting, which decreases the model's performance on unseen data. Hence, we can state that the optimal subspace appears to be around 90 to 100 components, as tested.

Results of Question II

In the second question, we use Fisher Linear Discriminant Analysis (LDA) to reduce the 784-dimensional Fashion-MNIST data to lower-dimensional subspaces for classification. First, the data is loaded, split into training and test sets, and then centered by subtracting the overall mean of the entire dataset. Next, LDA is applied on the centered training data to extract up to 9 discriminant directions. These LDA bases are then visualized as images, so we can see the directions that best separate the different classes. After visualizing the LDA bases, we evaluate a Gaussian classifier. For each subspace dimension (from 1 to 9), the training and test data are projected onto the first d LDA components. The Gaussian classifier is trained on the projected training data, and predictions are made on both training and test sets. The classification error is computed for each dimension, and the results are plotted to show how the error changes with the number of LDA components. The figure given below displays the bases obtained in LDA:

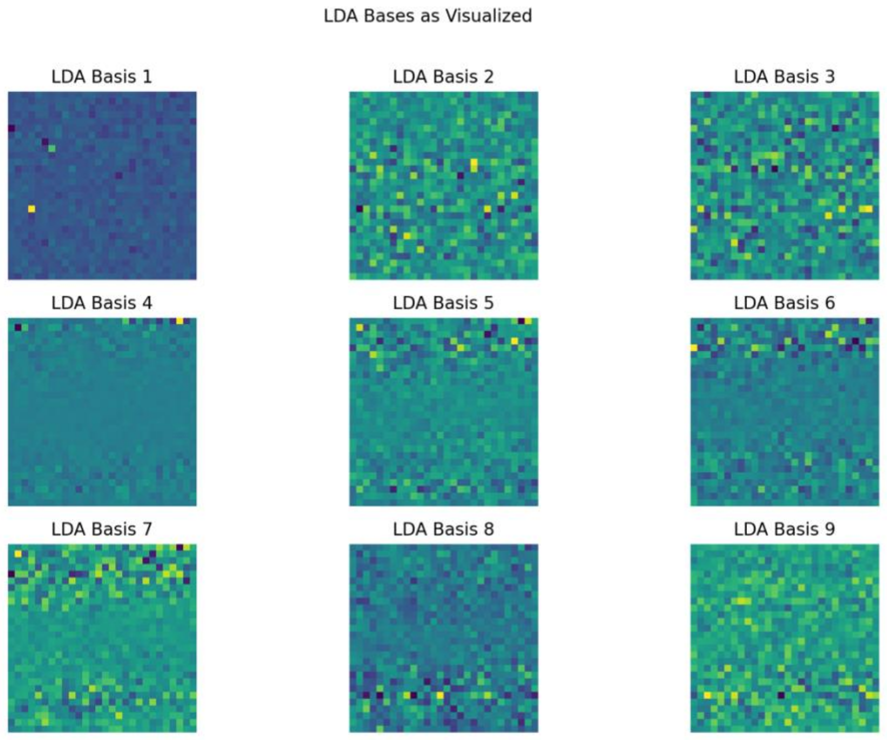
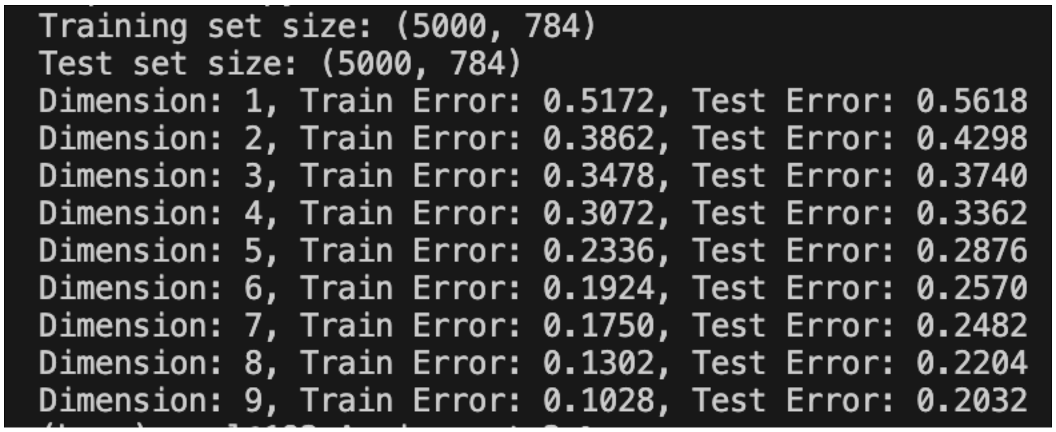


Figure 7: *LDA Bases as visualized.*

When we compare the LDA bases with PCA components, it seems that PCA components provide a visually more understandable representation of the elements of the set. This could be due to an error, but LDA components seem to be less-explaining in terms of visualization than PCA components.

The figure given below displays the test and training accuracy for every individual dimension:



```
Training set size: (5000, 784)
Test set size: (5000, 784)
Dimension: 1, Train Error: 0.5172, Test Error: 0.5618
Dimension: 2, Train Error: 0.3862, Test Error: 0.4298
Dimension: 3, Train Error: 0.3478, Test Error: 0.3740
Dimension: 4, Train Error: 0.3072, Test Error: 0.3362
Dimension: 5, Train Error: 0.2336, Test Error: 0.2876
Dimension: 6, Train Error: 0.1924, Test Error: 0.2570
Dimension: 7, Train Error: 0.1750, Test Error: 0.2482
Dimension: 8, Train Error: 0.1302, Test Error: 0.2204
Dimension: 9, Train Error: 0.1028, Test Error: 0.2032
```

Figure 8: *Training and test error per dimension.*

The figure given below displays the plot for training accuracy per component:

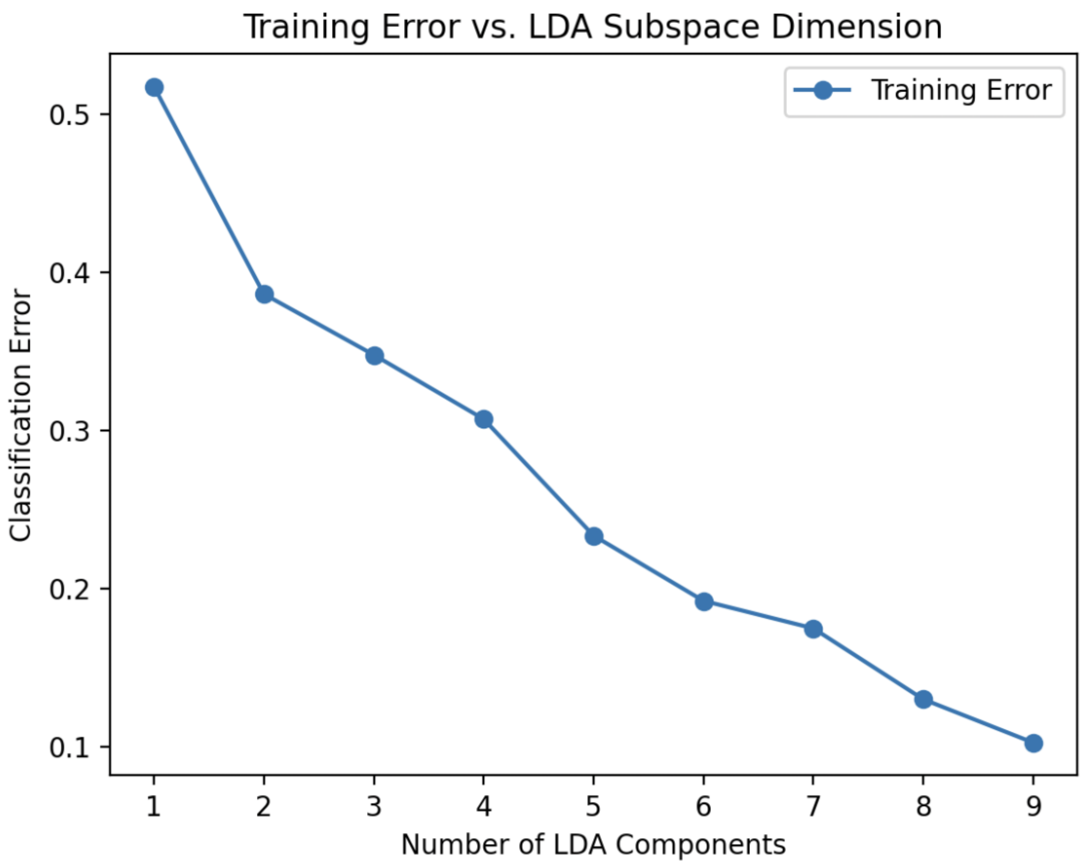


Figure 9: *Training error per component.*

The figure given below displays the plot for test accuracy per component:

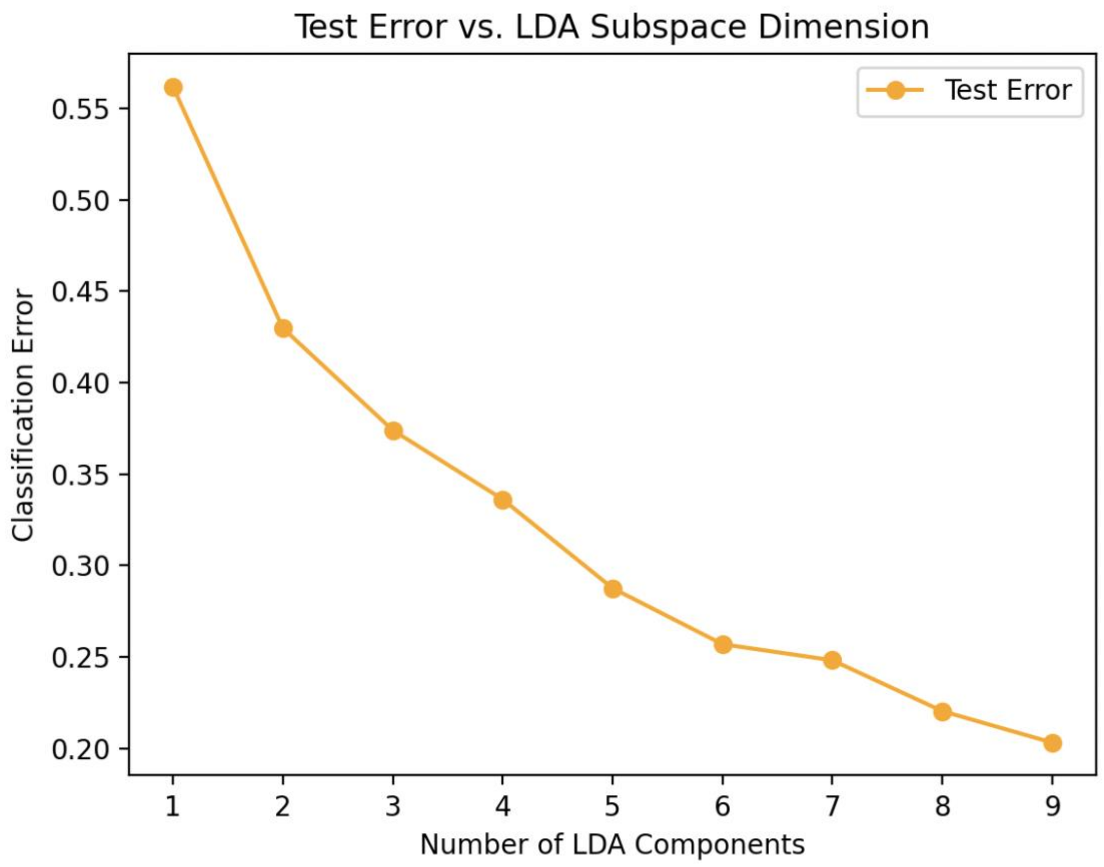


Figure 10: *Test error per component.*

When we compare the results of PCA with LDA, LDA is computationally more efficient in terms of runtime, in both training and test set; however, in both sets, PCA seems to yield a better accuracy in terms of classification.

Results of Question III

In the third question, we apply t-SNE to reduce the high-dimensional Fashion-MNIST data to two dimensions for visualization. In this step, we begin by centering the data by computing the overall mean of the entire dataset and subtracting it from every sample. Next, we define a range of parameter values for perplexity (5, 10, 15, 20, 25, 30) and for the number of iterations (250, 500, 750, 1000). For each combination of these parameters, t-SNE is applied to the centered data to compute a 2D mapping, and an individual scatter plot is generated. The figure given below displays the results for perplexity set as 5 and number of iterations set as 250:

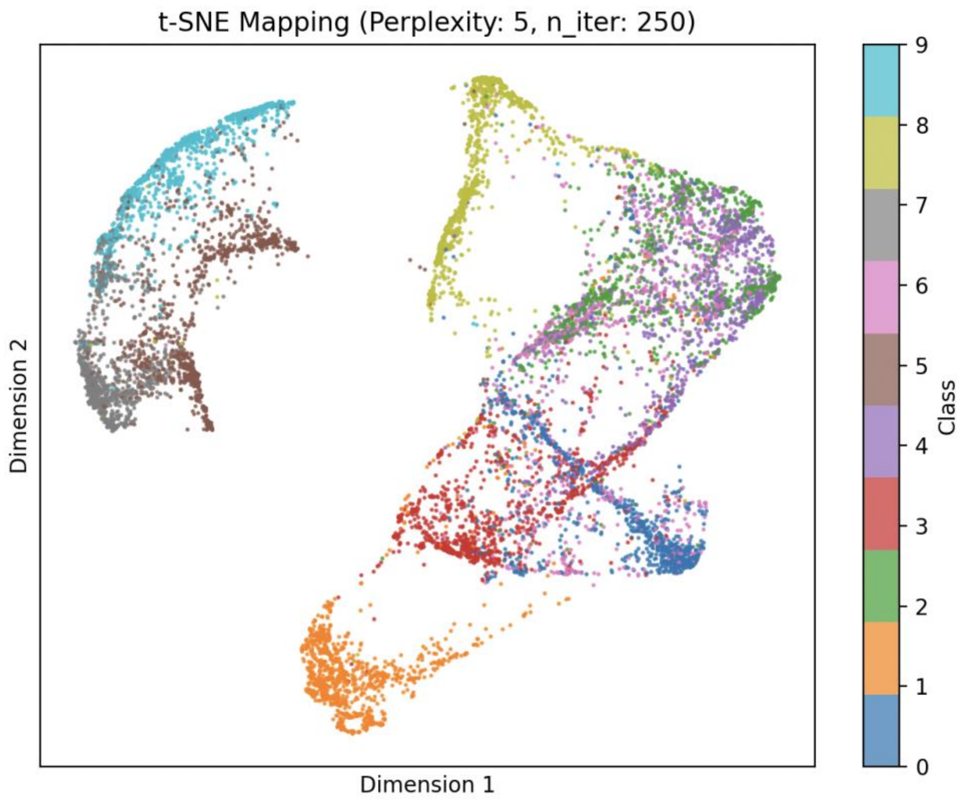


Figure 11: *Mapping for perplexity = 5 and number of iterations = 250.*

The figure given below displays the results for perplexity set as 10 and number of iterations set as 250:

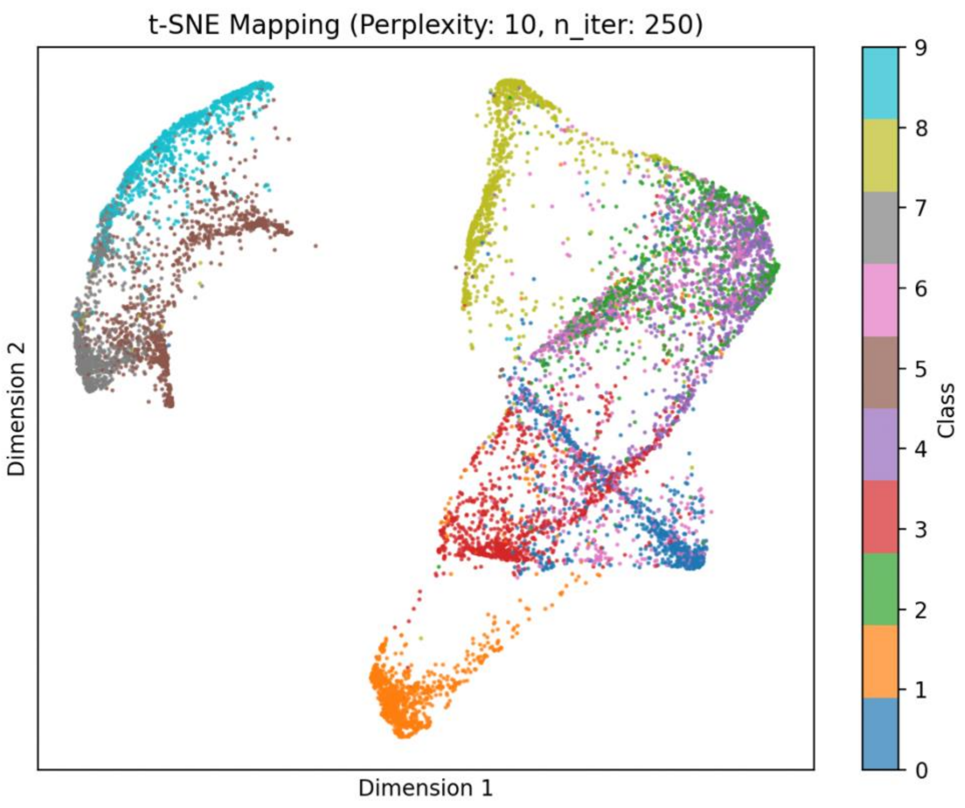


Figure 12: *Mapping for perplexity = 10 and number of iterations = 250.*

The figure given below displays the results for perplexity set as 15 and number of iterations set as 250:

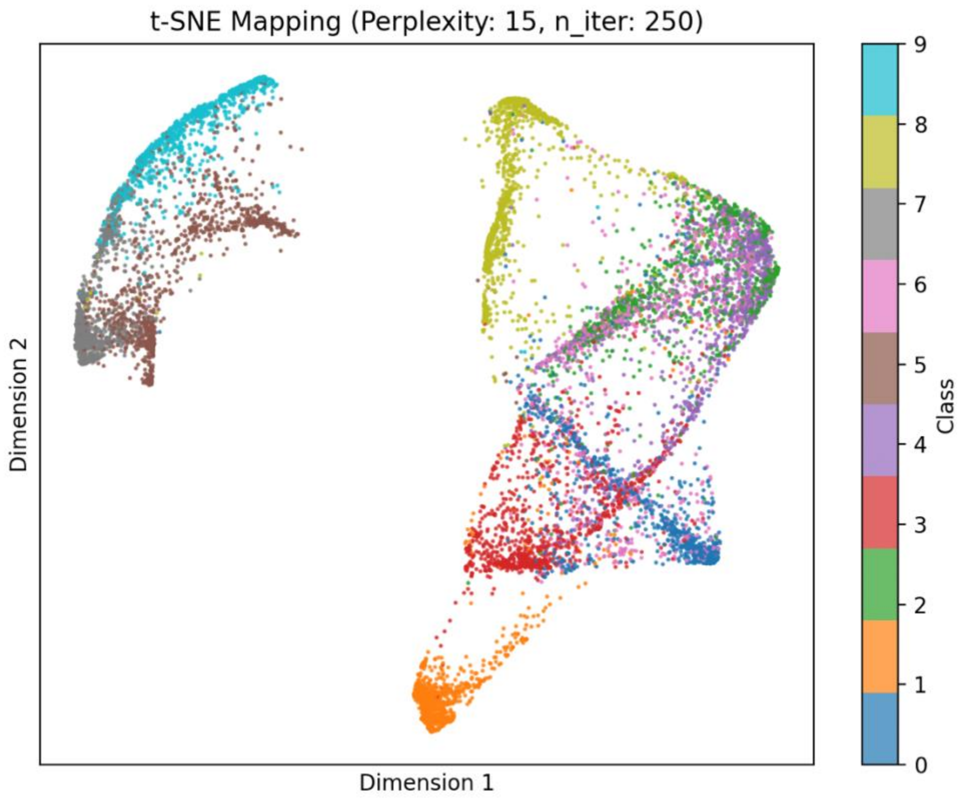


Figure 13: *Mapping for perplexity = 15 and number of iterations = 250.*

The figure given below displays the results for perplexity set as 20 and number of iterations set as 250:

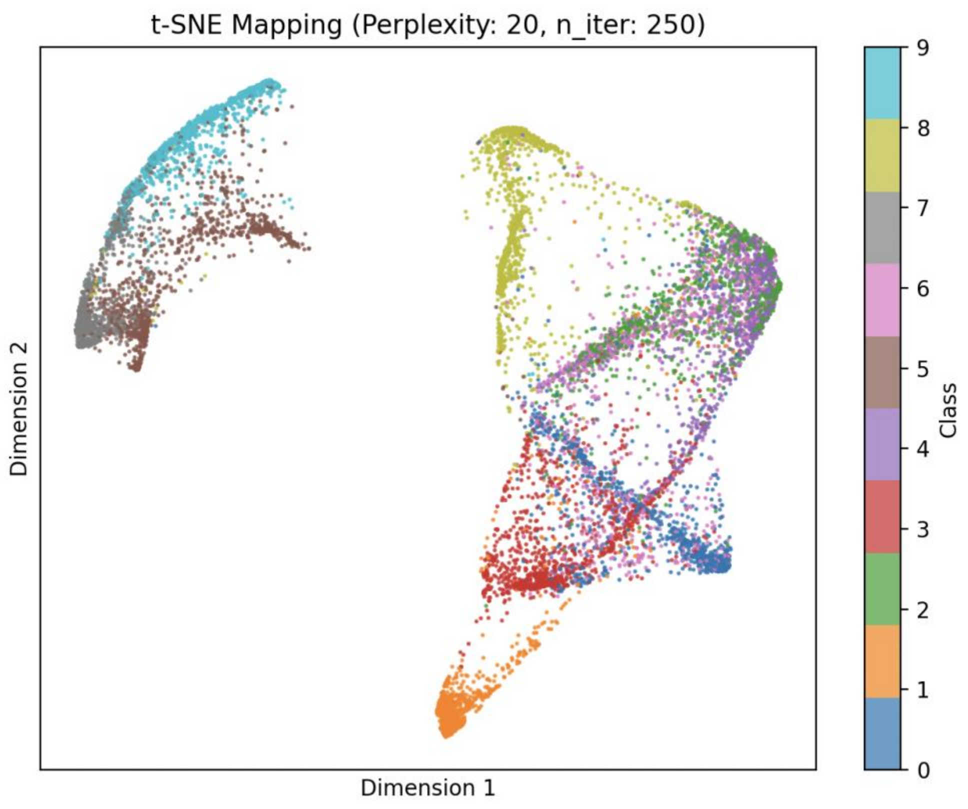


Figure 14: *Mapping for perplexity = 20 and number of iterations = 250.*

The figure given below displays the results for perplexity set as 25 and number of iterations set as 250:

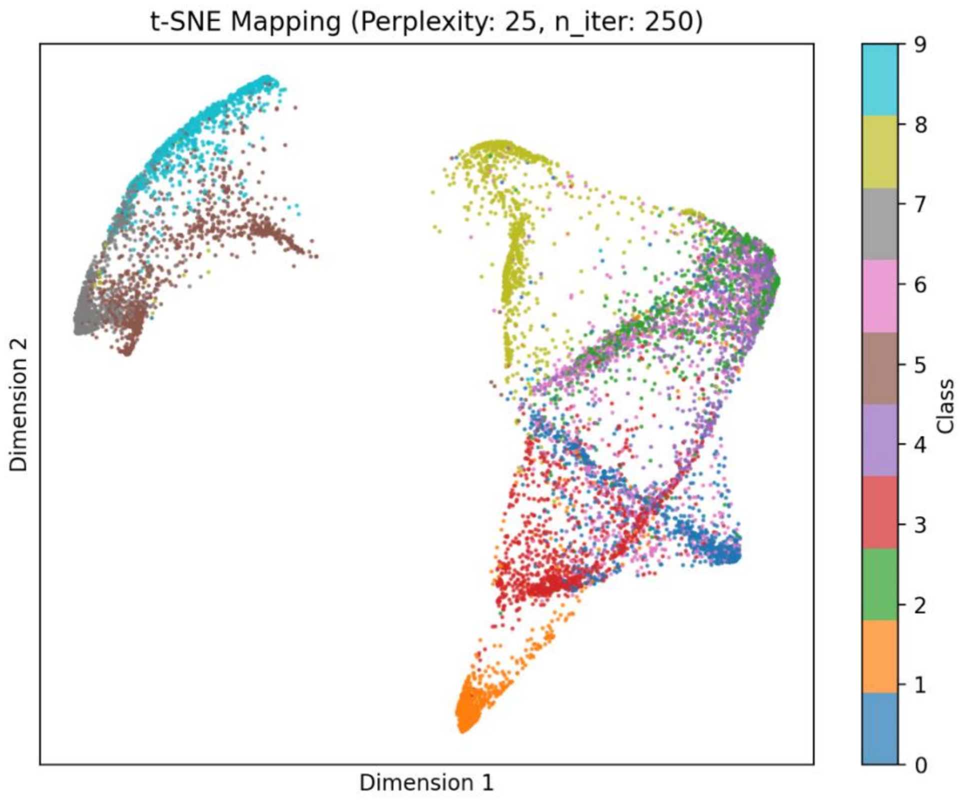


Figure 15: *Mapping for perplexity = 25 and number of iterations = 250.*

The figure given below displays the results for perplexity set as 30 and number of iterations set as 250:

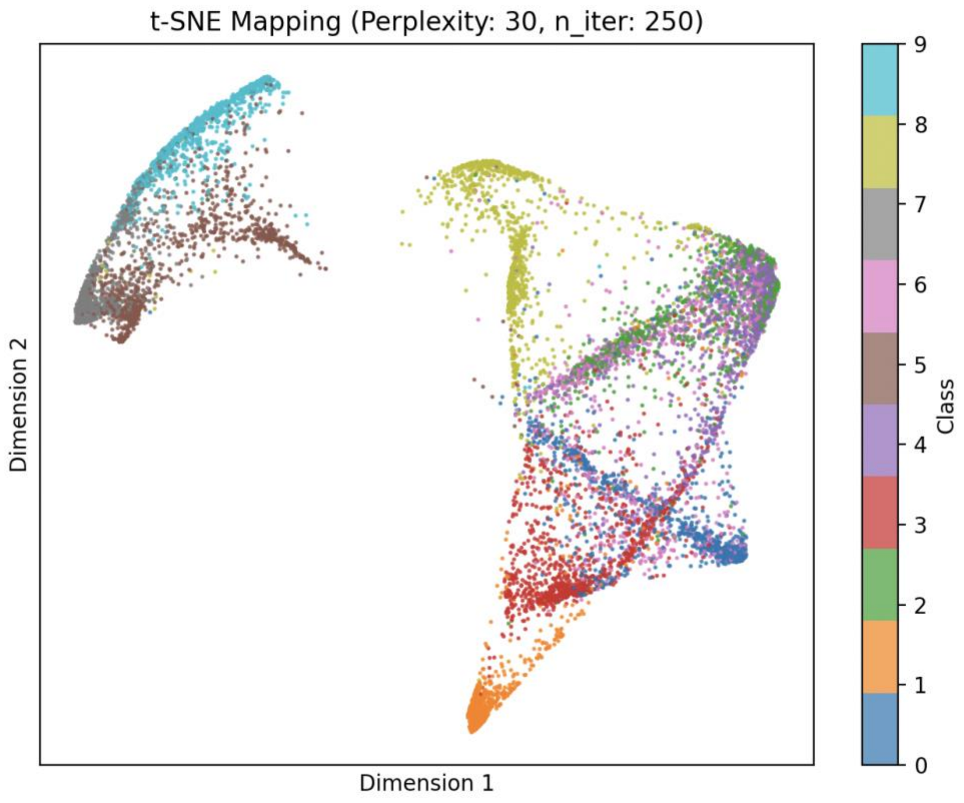


Figure 16: *Mapping for perplexity = 30 and number of iterations = 250.*

The figure given below displays the results for perplexity set as 5 and number of iterations set as 500:

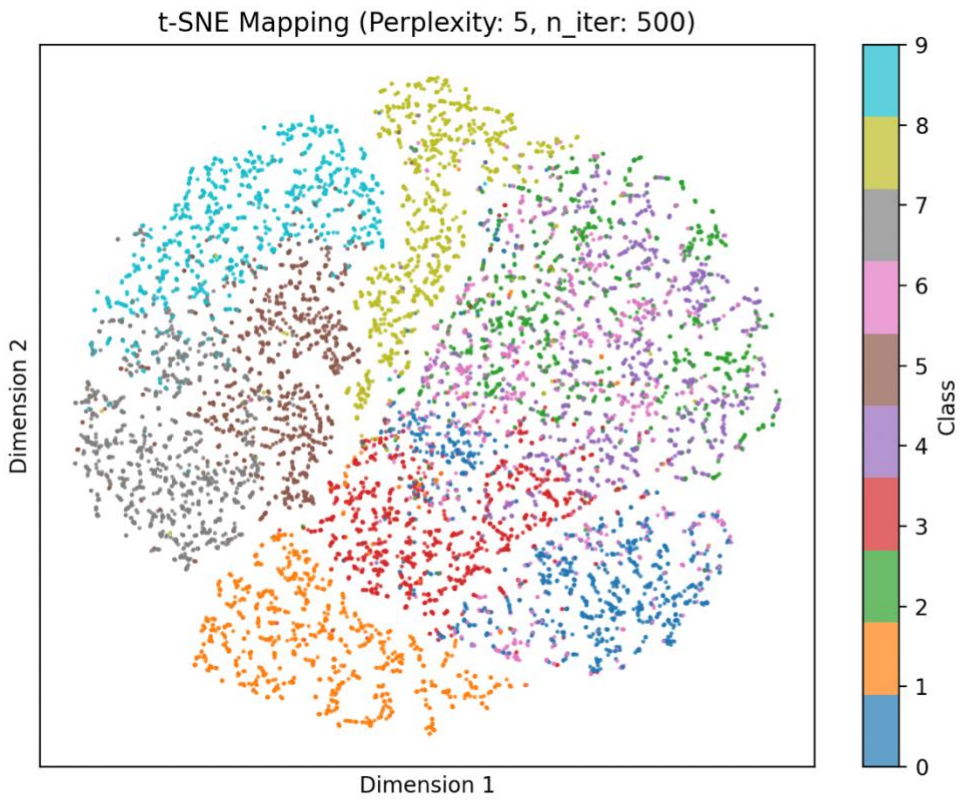


Figure 17: *Mapping for perplexity = 5 and number of iterations = 500.*

The figure given below displays the results for perplexity set as 10 and number of iterations set as 500:

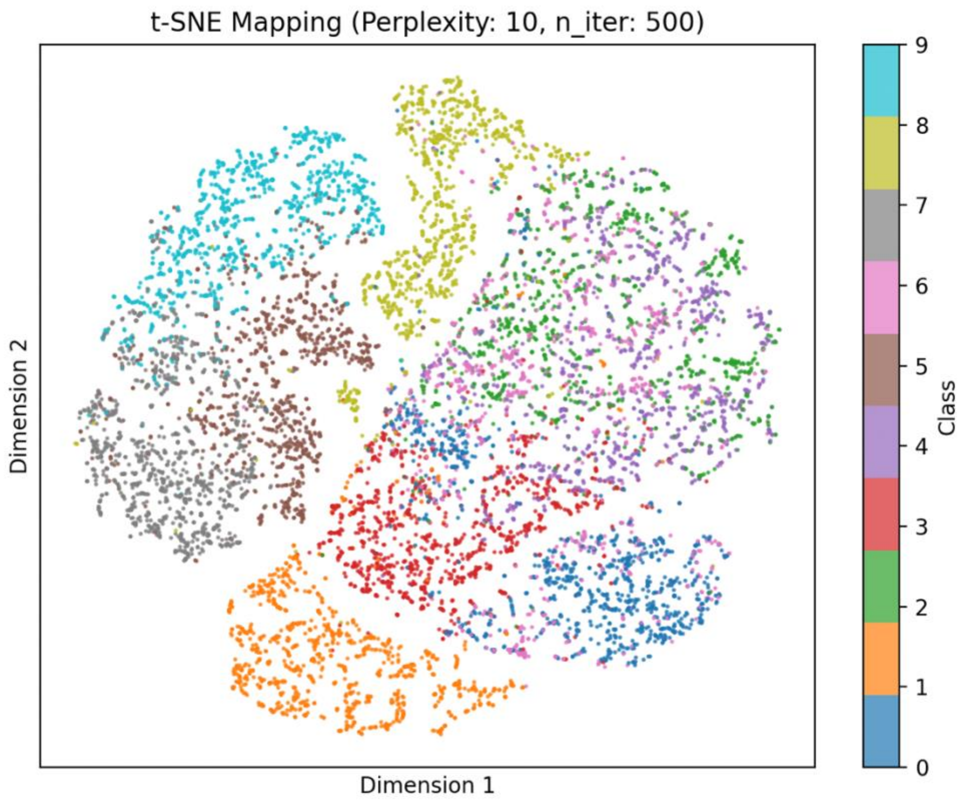


Figure 18: *Mapping for perplexity = 10 and number of iterations = 500.*

The figure given below displays the results for perplexity set as 15 and number of iterations set as 500:

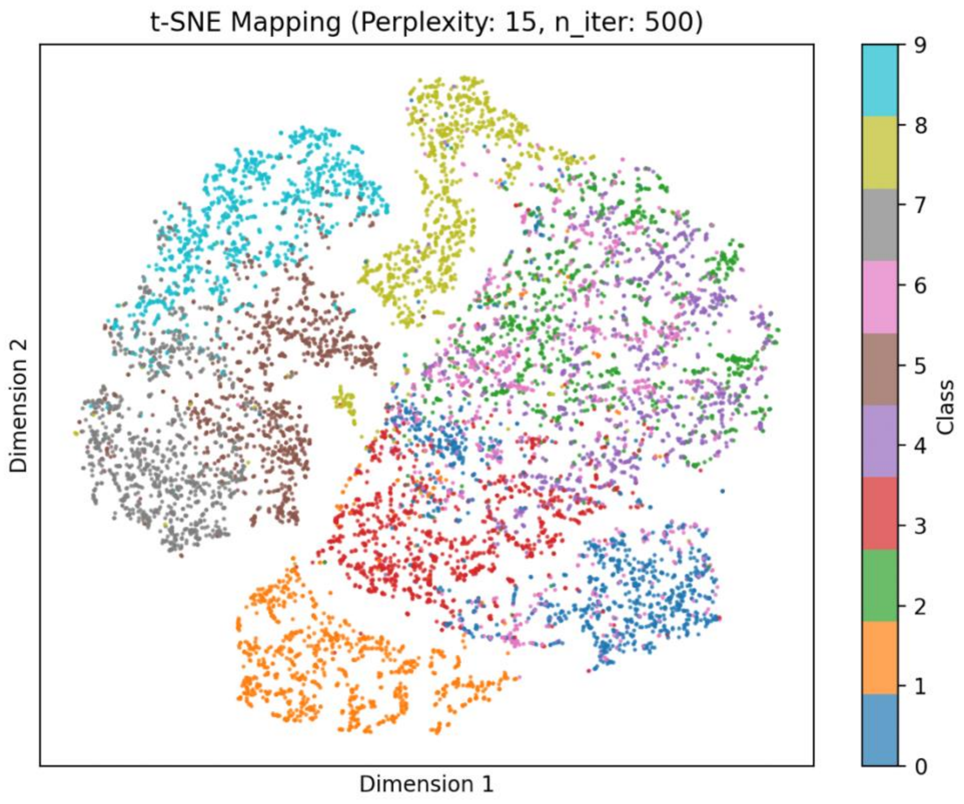


Figure 19: *Mapping for perplexity = 15 and number of iterations = 500.*

The figure given below displays the results for perplexity set as 20 and number of iterations set as 500:

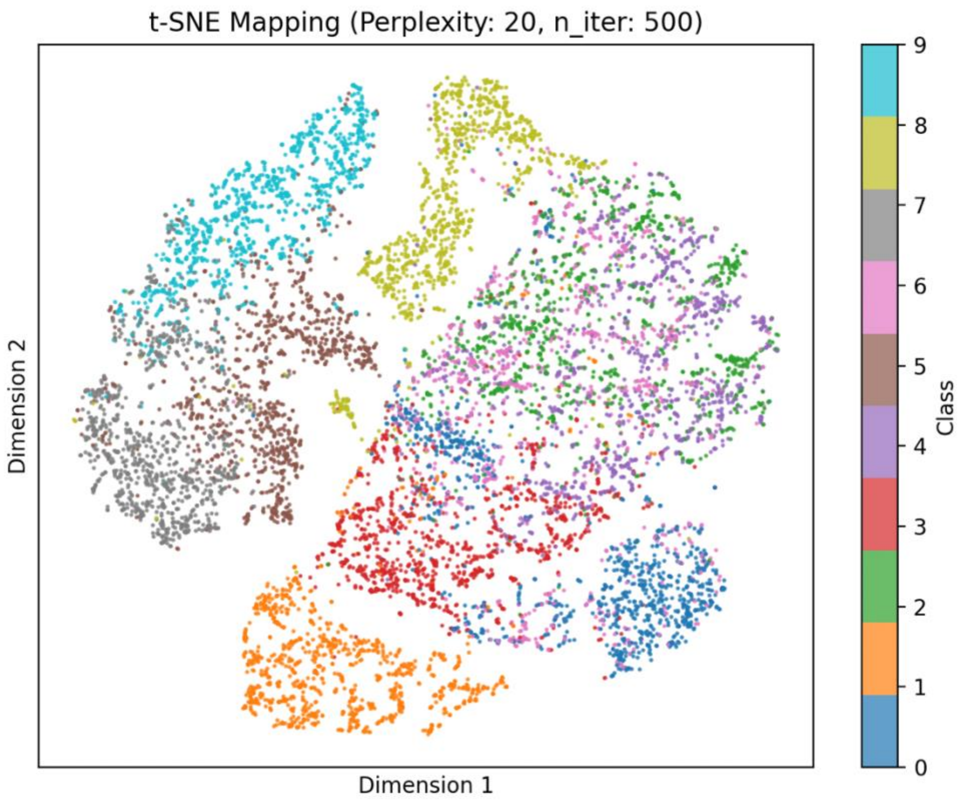


Figure 20: *Mapping for perplexity = 20 and number of iterations = 500.*

The figure given below displays the results for perplexity set as 25 and number of iterations set as 500:

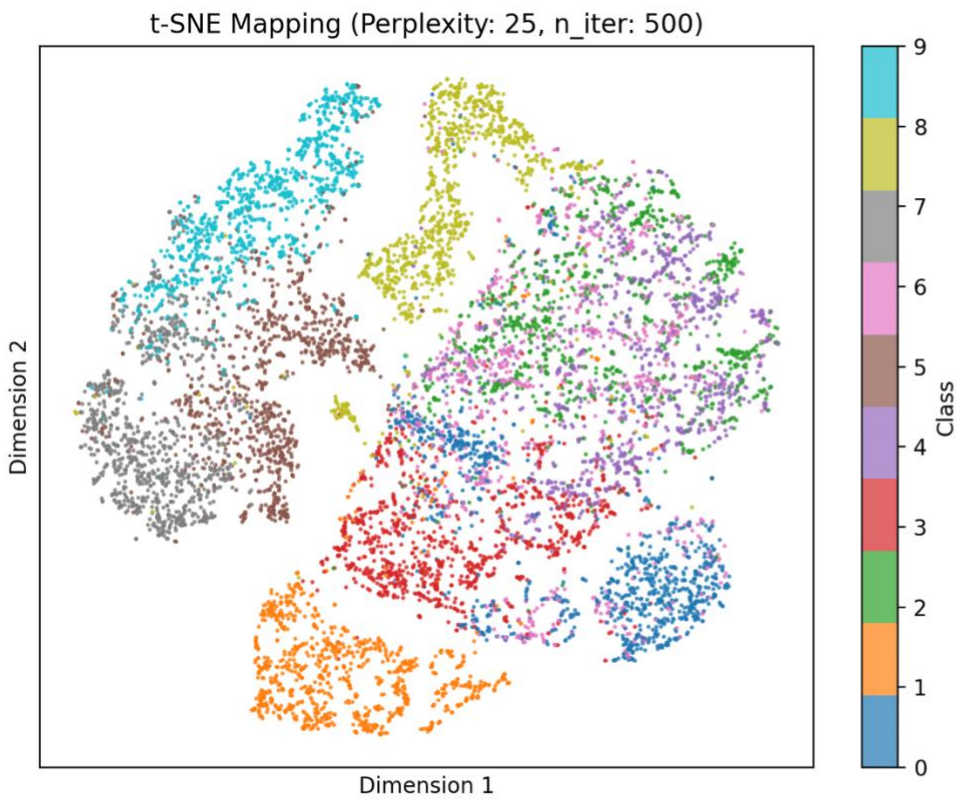


Figure 21: *Mapping for perplexity = 25 and number of iterations = 500.*

The figure given below displays the results for perplexity set as 30 and number of iterations set as 500:

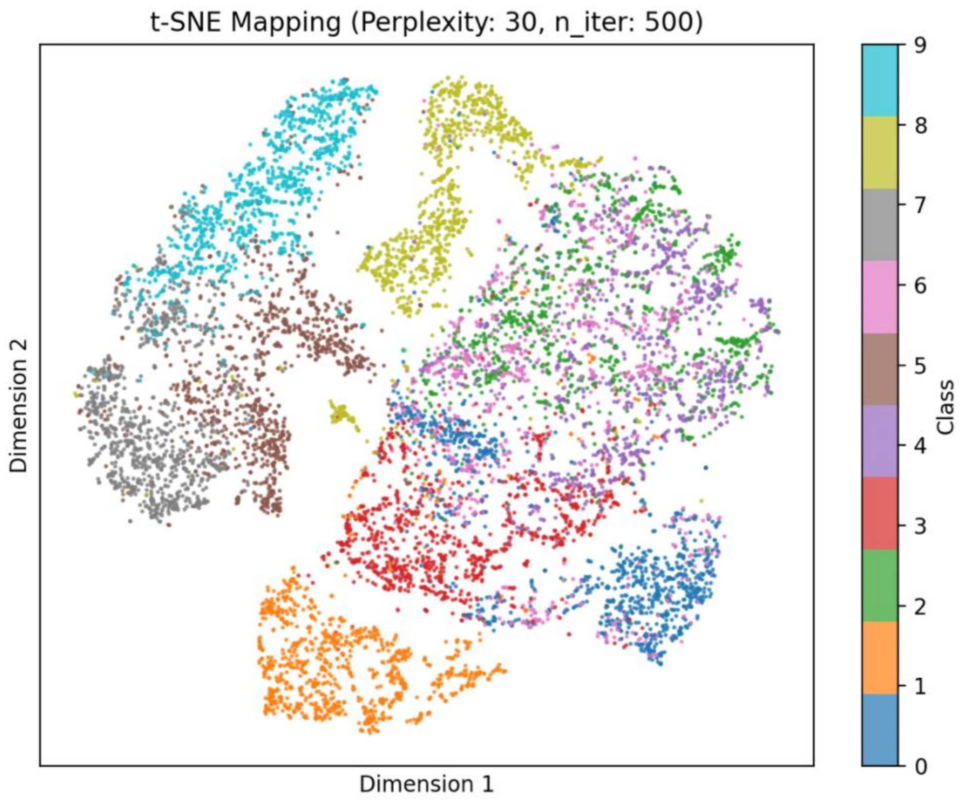


Figure 22: *Mapping for perplexity = 30 and number of iterations = 500.*

The figure given below displays the results for perplexity set as 5 and number of iterations set as 750:

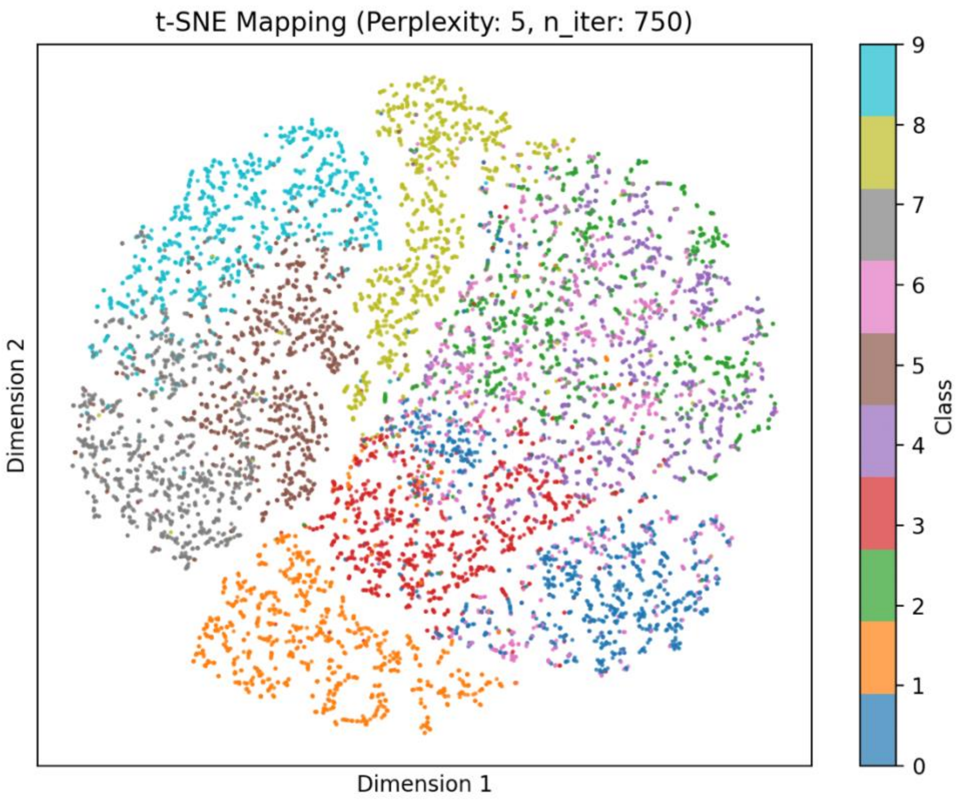


Figure 23: *Mapping for perplexity = 5 and number of iterations = 750.*

The figure given below displays the results for perplexity set as 10 and number of iterations set as 750:

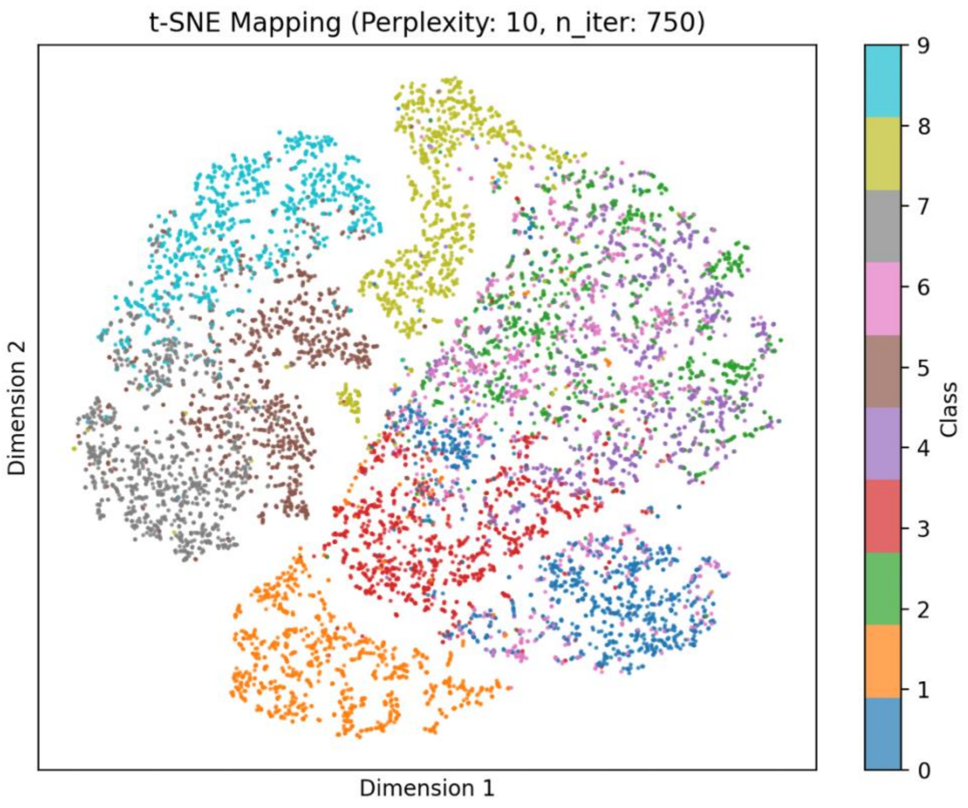


Figure 24: *Mapping for perplexity = 10 and number of iterations = 750.*

The figure given below displays the results for perplexity set as 15 and number of iterations set as 750:

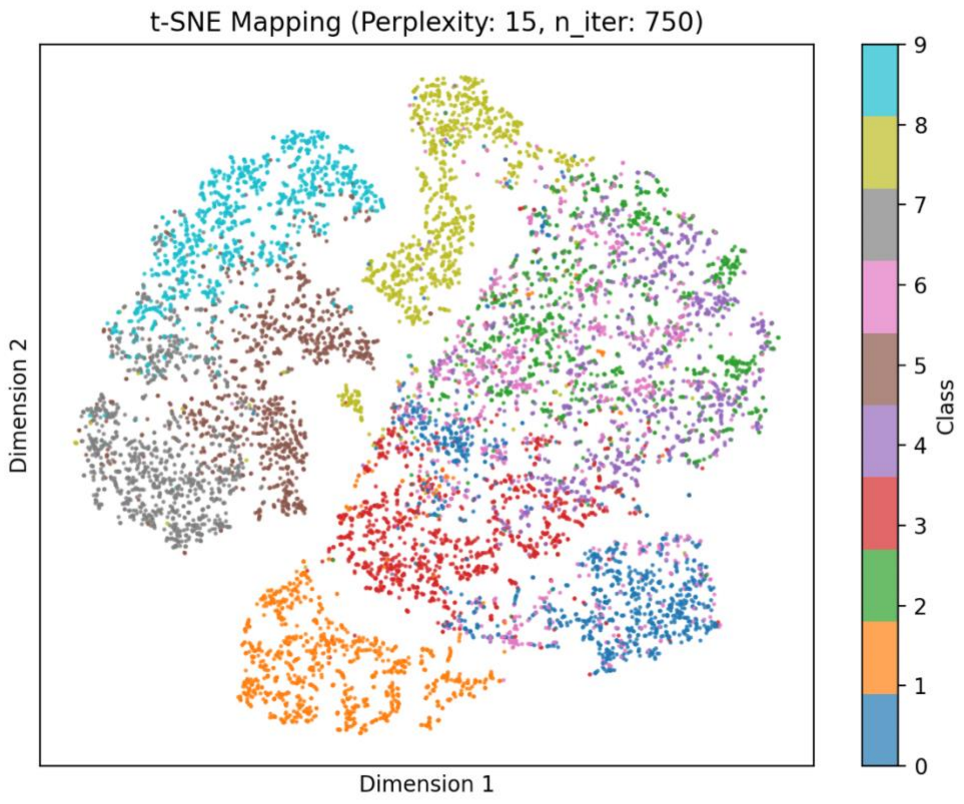


Figure 25: *Mapping for perplexity = 15 and number of iterations = 750.*

The figure given below displays the results for perplexity set as 20 and number of iterations set as 750:

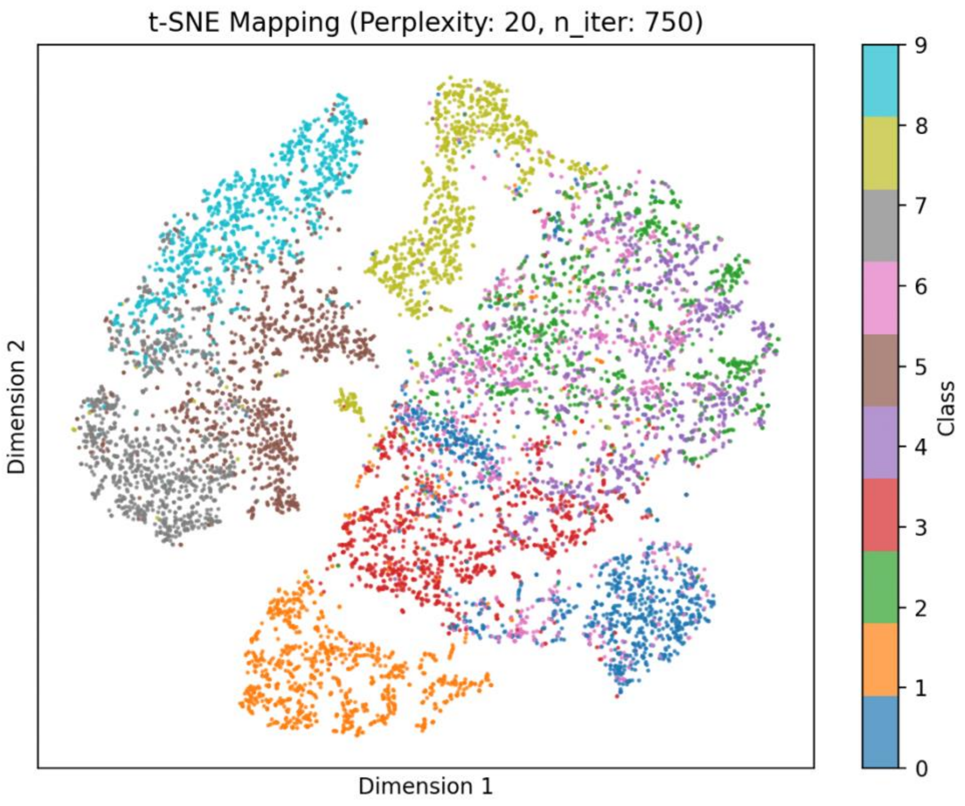


Figure 26: *Mapping for perplexity = 20 and number of iterations = 750.*

The figure given below displays the results for perplexity set as 25 and number of iterations set as 750:

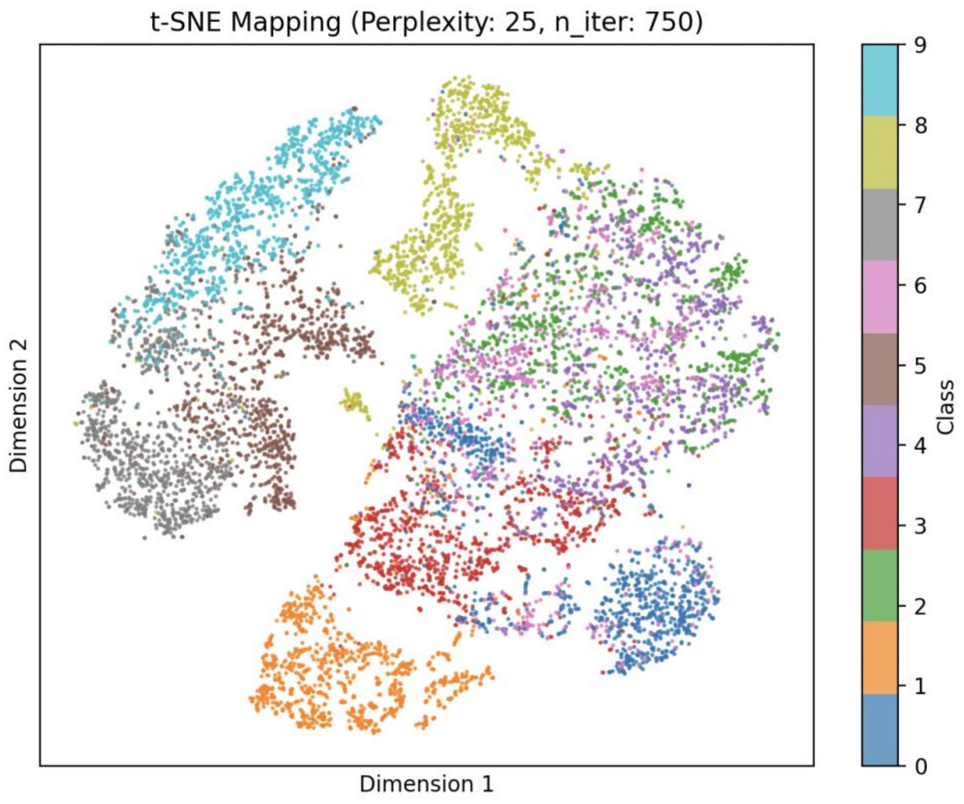


Figure 27: *Mapping for perplexity = 25 and number of iterations = 750.*

The figure given below displays the results for perplexity set as 30 and number of iterations set as 750:

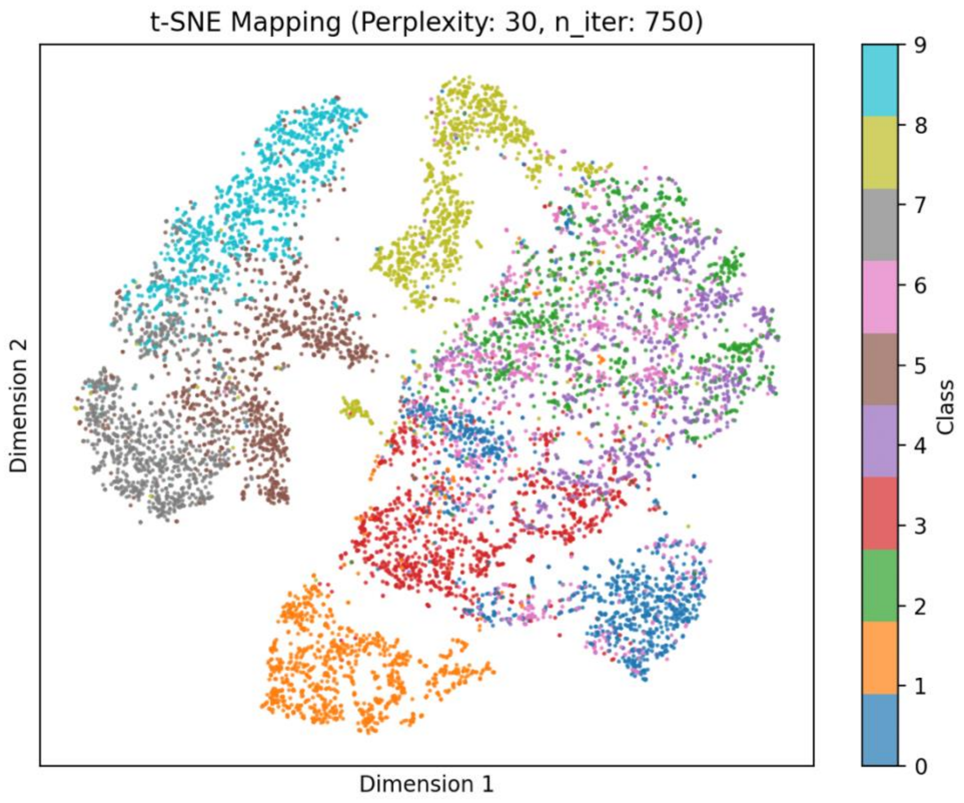


Figure 28: *Mapping for perplexity = 30 and number of iterations = 750.*

The figure given below displays the results for perplexity set as 5 and number of iterations set as 1000:

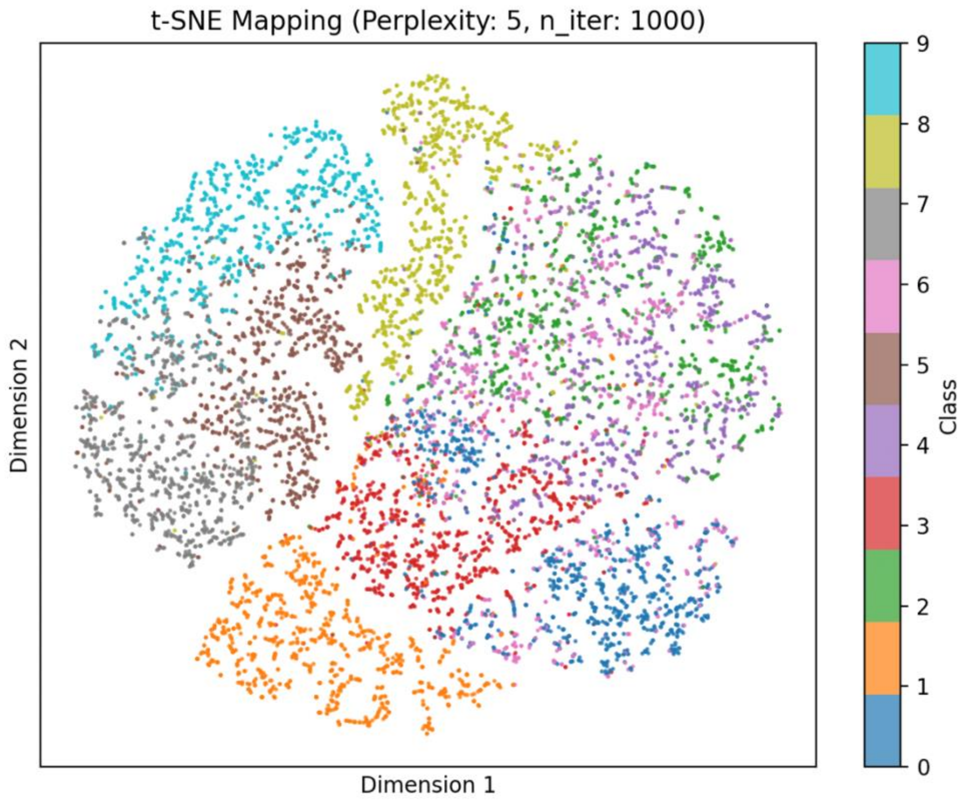


Figure 29: *Mapping for perplexity = 5 and number of iterations = 1000.*

The figure given below displays the results for perplexity set as 10 and number of iterations set as 1000:

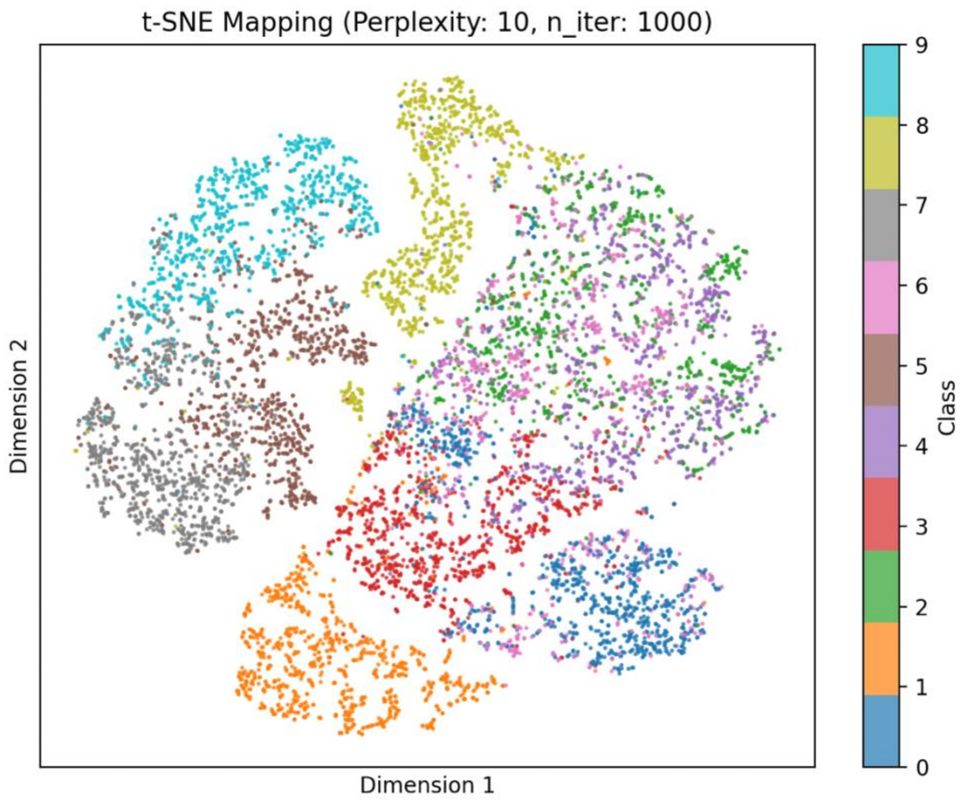


Figure 30: *Mapping for perplexity = 10 and number of iterations = 1000.*

The figure given below displays the results for perplexity set as 15 and number of iterations set as 1000:

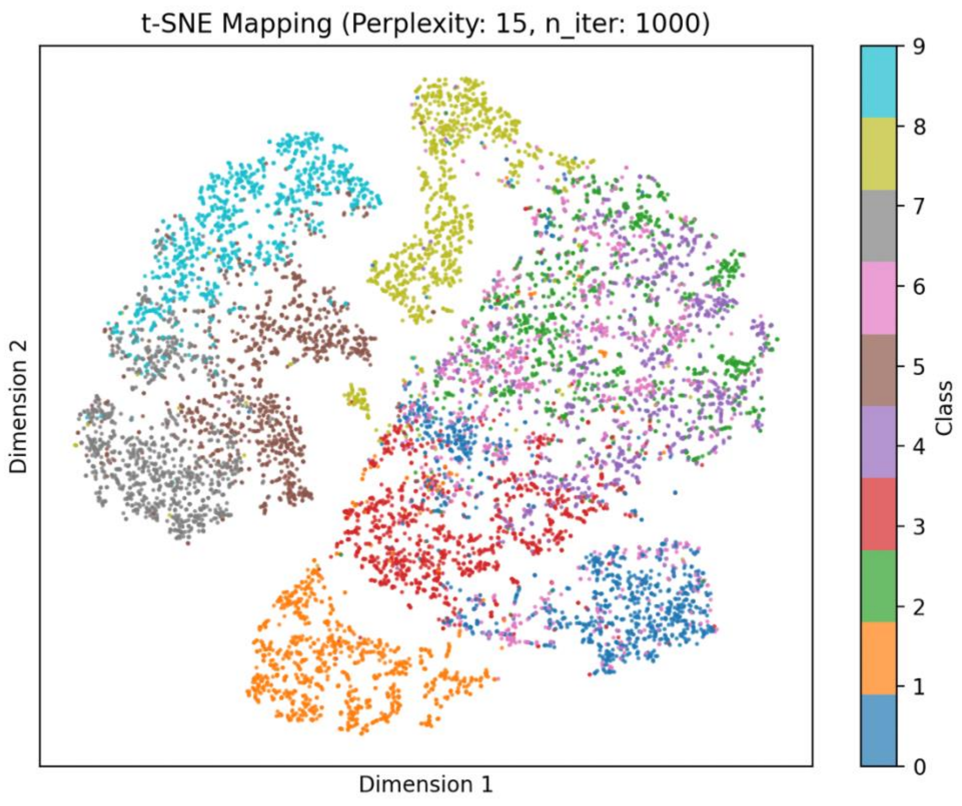


Figure 31: *Mapping for perplexity = 15 and number of iterations = 1000.*

The figure given below displays the results for perplexity set as 20 and number of iterations set as 1000:

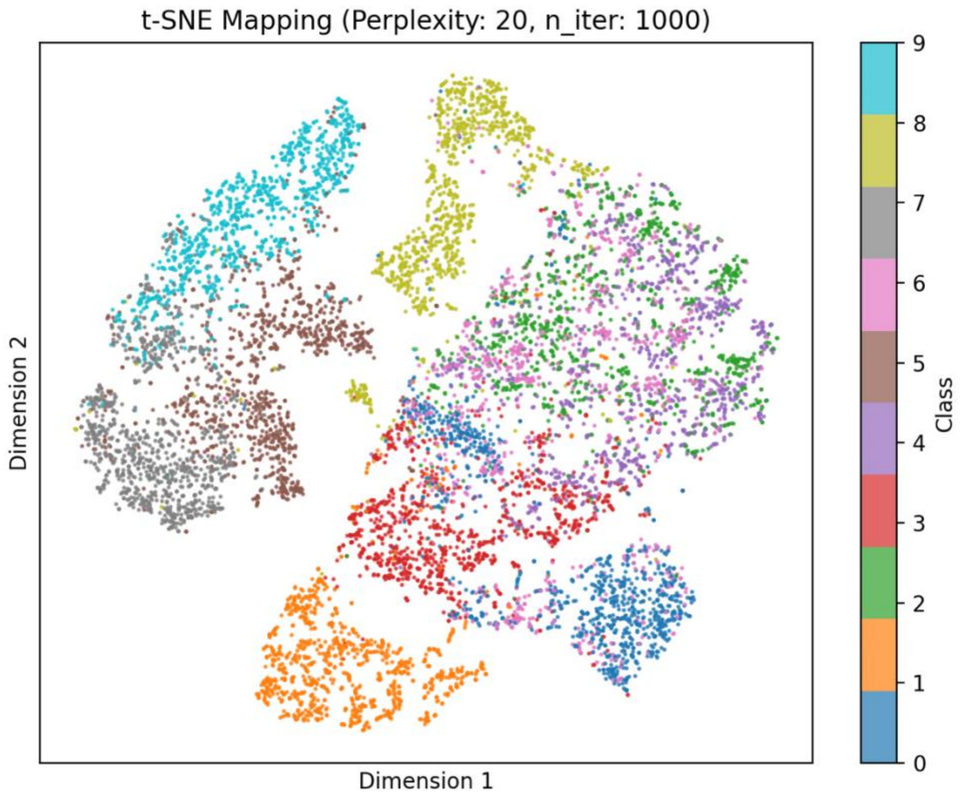


Figure 32: *Mapping for perplexity = 20 and number of iterations = 1000.*

The figure given below displays the results for perplexity set as 25 and number of iterations set as 1000:

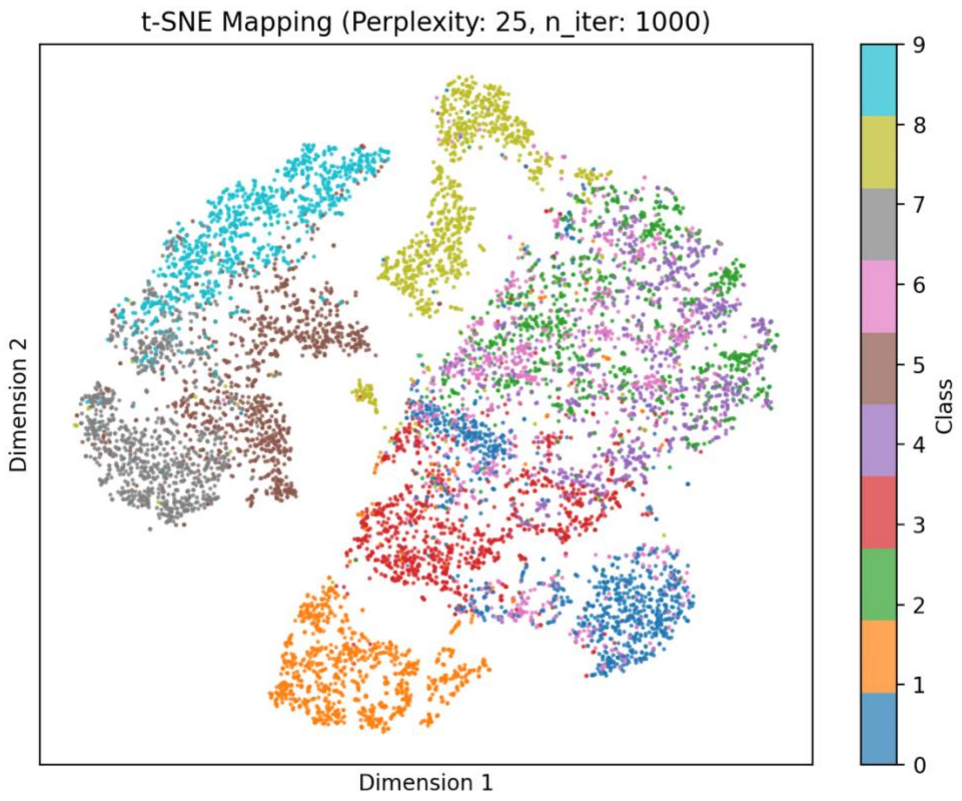


Figure 33: *Mapping for perplexity = 25 and number of iterations = 1000.*

The figure given below displays the results for perplexity set as 30 and number of iterations set as 1000:

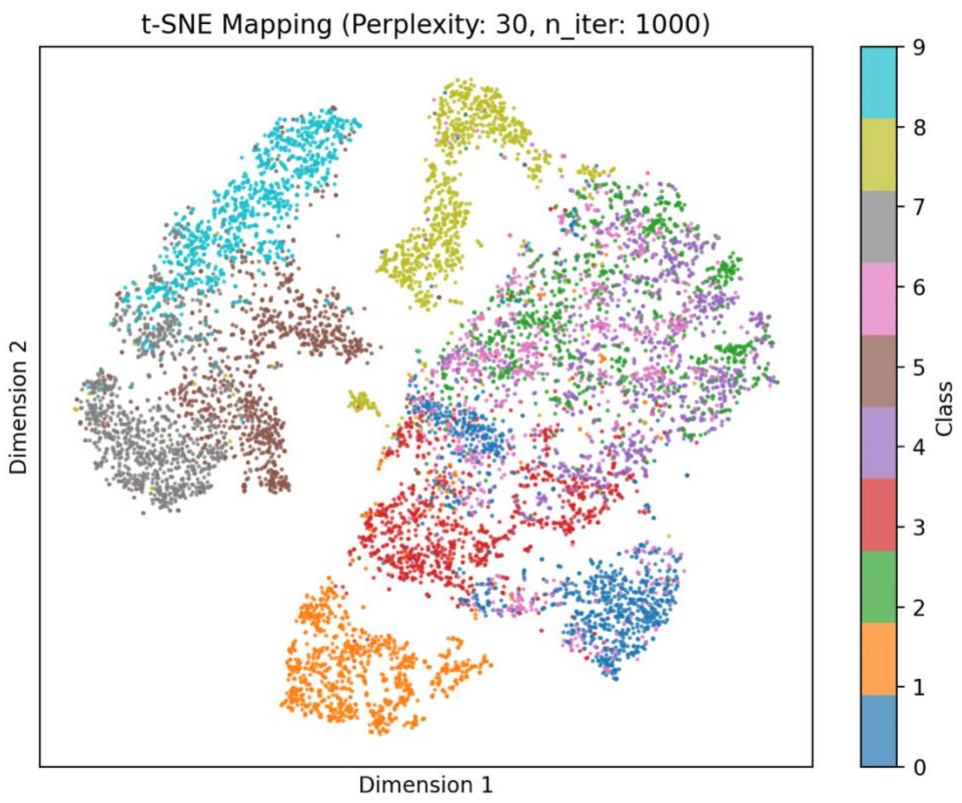


Figure 34: *Mapping for perplexity = 30 and number of iterations = 1000.*

As we gradually increase the perplexity and the number of iterations, the clusters become more distinct and clearer. This shows that t-SNE is better capturing the true structure of the data. A higher perplexity makes the algorithm consider more neighbors, which gives a broader view of the data and helps separate different groups more clearly. Also, running more iterations allows the algorithm more time to fine-tune the mapping, reducing the difference between the original high-dimensional relationships and the low-dimensional display. In the end, naturally different classes form clear, separate clusters, making the visualization easier to understand.

Conclusion

In conclusion, this assignment has shown how different dimensionality reduction techniques can help us understand and work with high-dimensional data. We used PCA to reduce the data and observed that the principal components capture overall variance and reveal mixed features of the images. With LDA, we focused on class separation, which produced bases that emphasize small but important differences between classes. Finally, using t-SNE, we visualized the data in two dimensions and explored how various parameter settings affect the state of clusters.