GE 461 Introduction to Data Science Spring 2023 Project-2: Dimensionality Reduction and Visualization

Question-1

Question-1.2

The plot shows the percentage of variance explained by each principal component in descending order of importance. Each point on the graph represents a principal component, and the x-axis shows the number of principal components used to calculate the variance. The y-axis shows the corresponding eigenvalue of each principal component.

In this plot, we can see that the first principal component explains the majority of the variance in the data, followed by the second, third, and so on. The slope of the curve starts high, indicating that the initial principal components capture a lot of the variance in the data. As we move along the x-axis, the slope starts to flatten out, indicating that the remaining principal components contribute less to the overall variance explained.

The plot also shows that around 100 principal components are enough to explain 90% of the variance in the data. Therefore, we can use the top 100 principal components to reduce the dimensionality of the dataset while retaining most of the variance information.

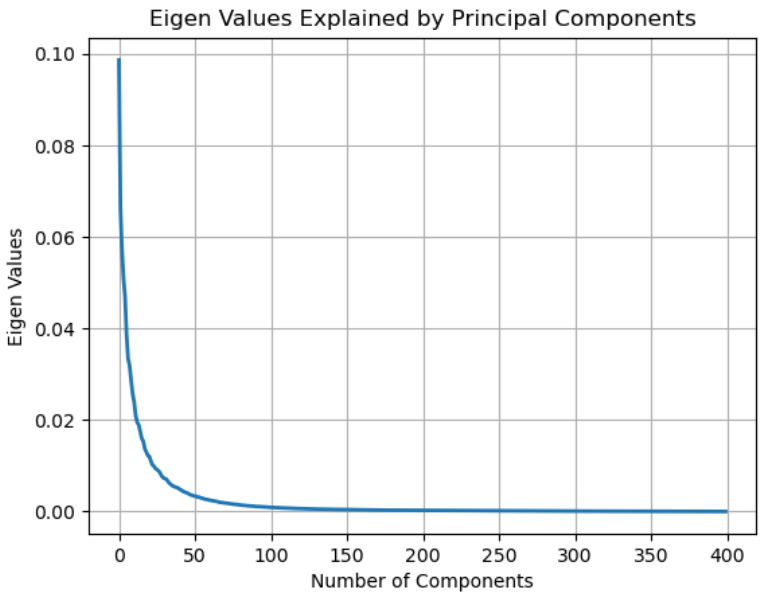

Figure 1

Question-1.3

I computed the sample mean of the training dataset using the np.mean function. Since the dataset has 400 components, I transformed the mean data into a 20 by 20 array. The resulting mean data is illustrated in Figure-2.

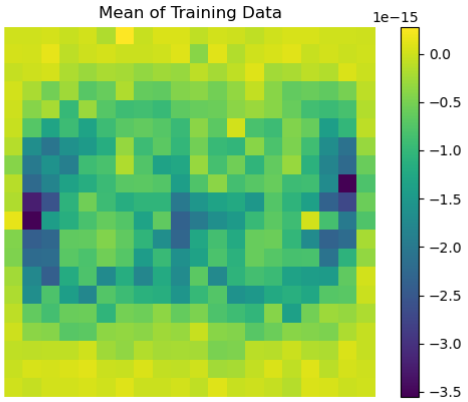

Figure 2

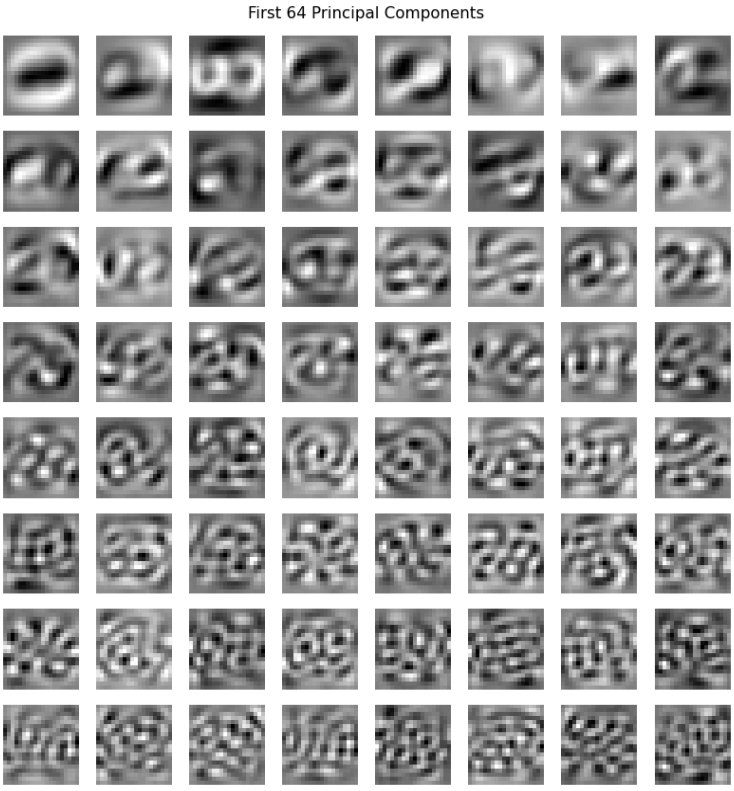

Figure 3

After examining the image depicted in Figure 2, I noted that the mean data has a greater abundance of circular points that are similar to digits 0, 3, 6, 8, and 9, in contrast to sharp points like digits 1, 4, and 7. This implies that digits contain more circular points than sharp points. Additionally, Figure-2 reveals that the digits have more lines positioned in the middle top and middle bottom regions, as shown by the yellow color, suggesting that these points occur more frequently in the data. As I predicted, the mean data has more circular points than sharp points, since only digits 1, 4, 5, and 7 feature sharp points, while digits 0, 2, 3, 5, 6, 8, and 9 contain circular points.

After examining the plot illustrated in Figure-1, I can deduce that roughly 64 principal components can be chosen. This conclusion is drawn from the fact that the plot starts to level off after this point, indicating a reduction in the variation between the principal components.

Question-1.5

The Figure 4 titled "Classification Error vs Number of Principal Components" shows how the classification error changes as the number of principal components used in the classification model increases. The plot shows two lines, one for the training error and one for the testing error. Both lines start high and decrease as more principal components are added to the model. However, the training error decreases much more quickly than the testing error. At around 40 principal components, the training error reaches a plateau and continues to decrease very slowly, while the testing error continues to decrease at a slower rate. This suggests that using more than 40 principal components does not significantly improve the performance of the model on the testing dataset, and may even lead to overfitting.

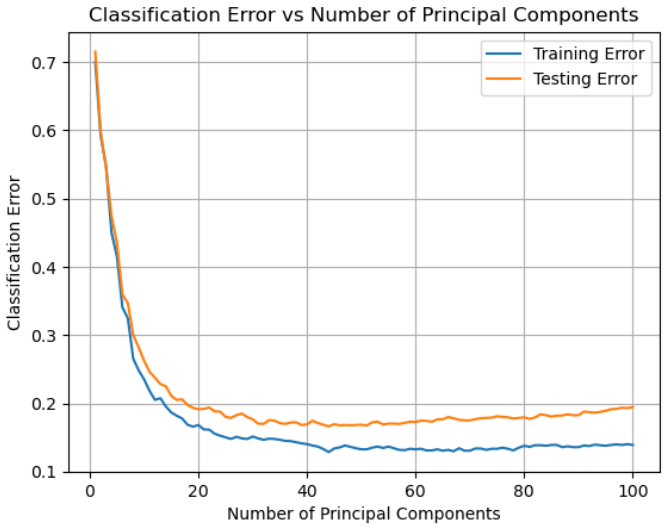

Figure 4

The Figure 5 titled "Training Result Statistic" shows the training error only as a function of the number of principal components. This plot confirms that the training error reaches a plateau at around 40 principal components, after which adding more principal components does not significantly improve the performance on the training dataset.

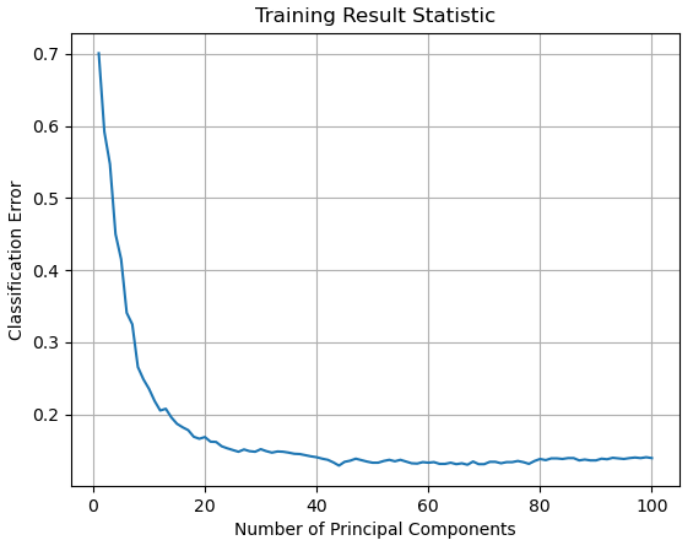

Figure 5

The Figure 6 titled "Testing Result Statistic" shows the testing error only as a function of the number of principal components. This plot shows that the testing error decreases continuously as more principal components are added to the model, but at a decreasing rate. This plot also suggests that using more than 40 principal components does not significantly improve the performance of the model on the testing dataset.

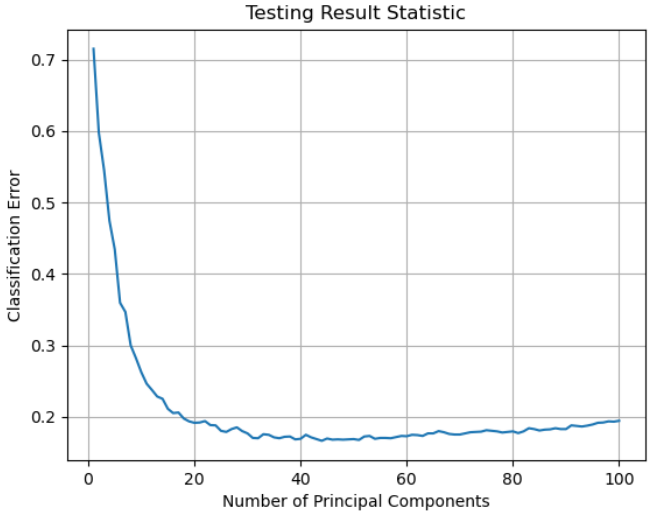

Figure 6

Question-2

Question-2.1

Figure 7 shows the embeddings of the MNIST dataset using the Isomap algorithm, where each image is represented as a point in a two-dimensional space. The colors of the points indicate the digit labels associated with each image. We can see that the plot shows some separation between the different digit classes, indicating that the Isomap algorithm was able to capture some of the underlying structure in the dataset. We can also observe some clusters of points, such as a tight cluster of points near the origin, which might correspond to a particular digit class. However, there is still some overlap between the different classes, suggesting that the dataset is not perfectly separable in the two-dimensional embedding space.

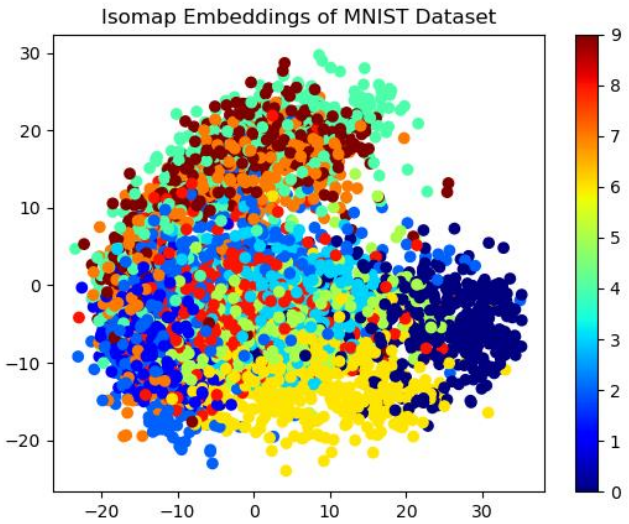

Figure 7

Question-2.2

The Figure 8 shows only the test error vs. dimensionality, and confirms that the test error plateaus for higher dimensions. This suggests that a lower-dimensional representation of the data obtained through Isomap can be used without significant loss of predictive accuracy.

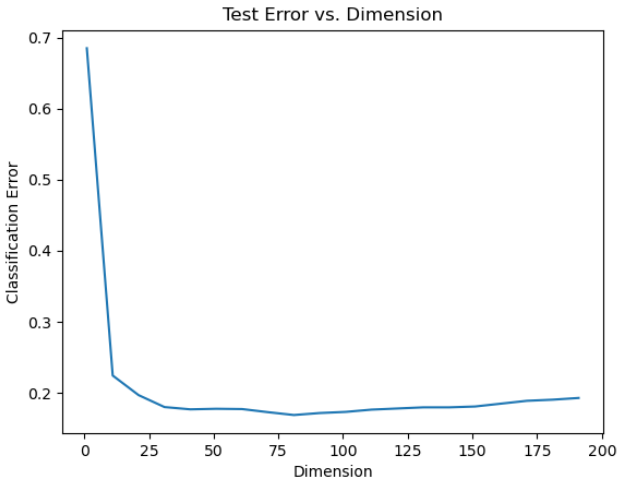

Figure 8

The Figure 9 shows only the training error vs. dimensionality, and confirms that the training error also plateaus for higher dimensions. This suggests that increasing the dimensionality beyond a certain point may not provide any additional benefit for model training.

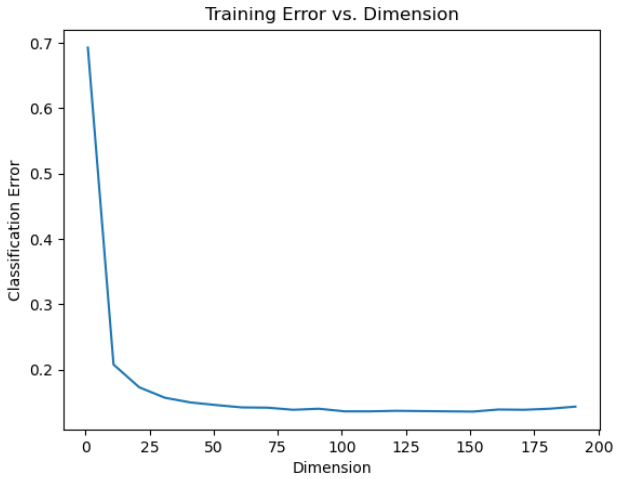

Figure 9

The Figure 10 shows the test error and training error vs. dimensionality for a Gaussian Naive Bayes classifier trained on the MNIST dataset using Isomap dimensionality reduction. As the dimensionality of the reduced data increases, the training error initially decreases, but then begins to level off and even increase slightly for higher dimensions. Meanwhile, the test error initially decreases at a faster rate than the training error, but then begins to level off and plateau as well. This suggests that increasing the dimensionality beyond a certain point does not provide any additional benefit and may even lead to overfitting.

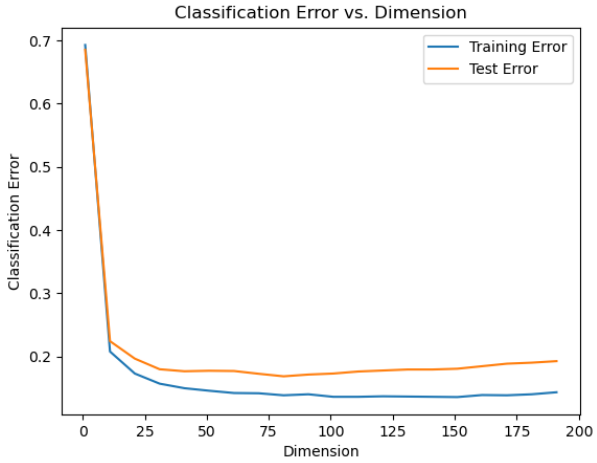

Figure 10

PCA and Isomap graphs are similar, it indicates that both methods are performing similarly in terms of dimensionality reduction. This could mean that the structure of the data in the original high-dimensional space is relatively linear, and therefore PCA, which preserves linear relationships, is able to capture most of the important structure in the data.

Question-3

n_components: This parameter determines the number of dimensions in which the data is embedded. In this case, the code sets n_components=2, meaning that the algorithm will produce a two-dimensional visualization.

perplexity: This parameter controls the balance between preserving local and global structure in the data. A perplexity value of 50.0 is used in some of the visualizations, while a value of 30.0 is used in others.

n_iter: This parameter controls the number of iterations of the optimization algorithm. The code uses values of 1500, 3000, and 4500 for n_iter in various visualizations.

random_state: This parameter sets the seed for the random number generator used by the algorithm. This allows for reproducibility of the results.

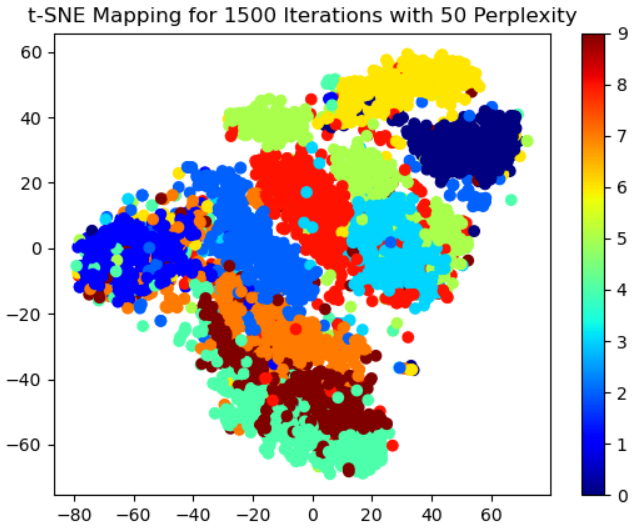

Figure 11

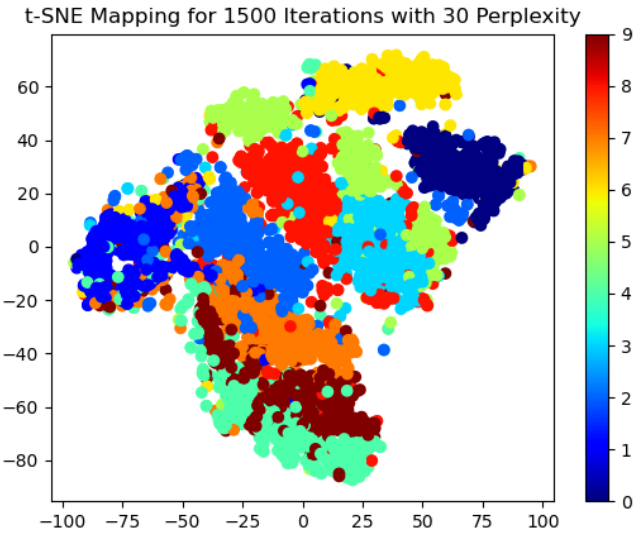

Figure 12

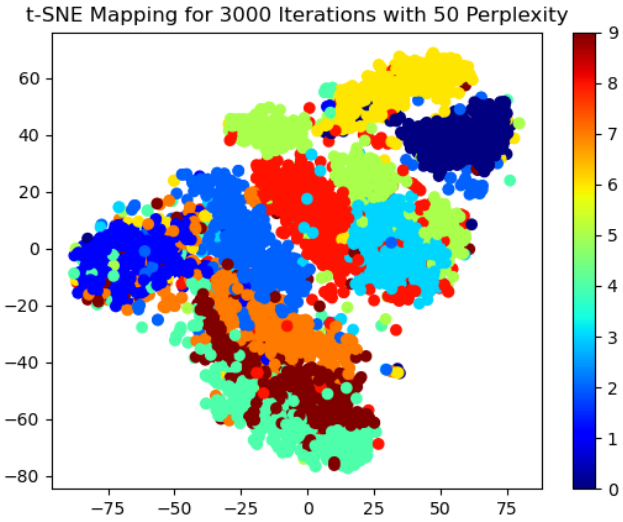

Figure 13

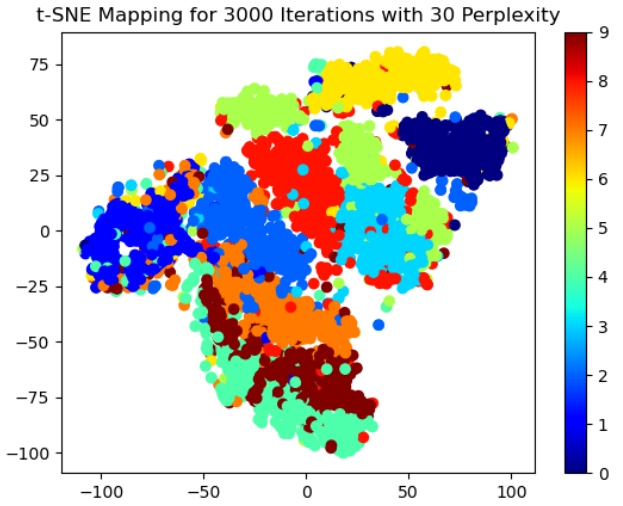

Figure 14

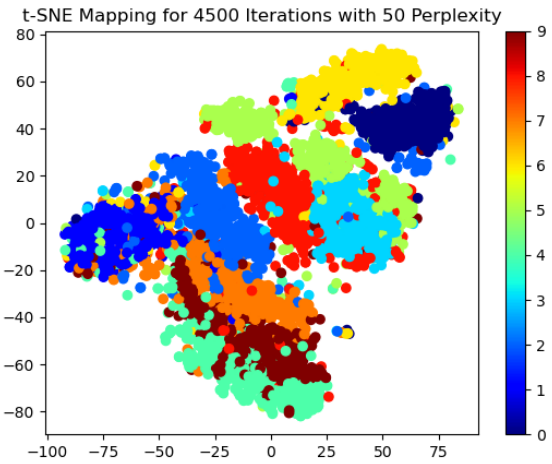

Figure 15

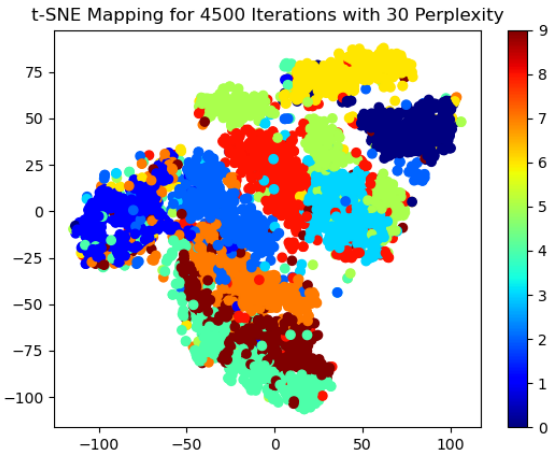

Figure 16

Overall, the resulting visualizations show that t-SNE is able to effectively cluster the MNIST digits into distinct groups, with similar digits appearing near each other in the visualization. In some cases, such as the visualization with perplexity=50.0 and n_iter = 4500, the clusters are relatively well-separated and easy to distinguish. In other cases, such as the visualization with perplexity = 30.0 and n_iter = 1500, the clusters are less well-defined and more jumbled together.

One thing to note is that t-SNE is a stochastic algorithm, meaning that the exact same set of parameters can produce slightly different results each time it is run. As a result, it is often useful to run the algorithm multiple times with different random seeds and visualize the resulting embeddings to get a more robust understanding of the underlying structure in the data.

References:

1) NumPy linalg PCA: https://numpy.org/doc/stable/reference/generated/numpy.linalg.pca.html

2) PCA explained simply by StatQuest with Josh Starmer: https://www.youtube.com/watch?v=_UVHneBUBW0

3) Implementation example in Python using scikit-learn: https://towardsdatascience.com/manifold-learning-how-isomap-works-d2a406e5d252

4) Isomap tutorial by the Scikit-learn library: https://scikit-learn.org/stable/modules/manifold.html#isomap

5) Visualizing Data using t-SNE: A Beginner's Guide: https://towardsdatascience.com/visualizing-data-using-t-sne-technique-21e161fa0596

6) An Introduction to t-SNE with Python Example: https://builtin.com/data-science/step-step-explanation-t-sne

7) Gaussian Naive Bayes Classification https://towardsdatascience.com/gaussian-naive-bayes-classification-8d0657d8bafb

8) Lecture slides and documents help me to understand the concept and idea of these algorithms.