Bilkent University

GE461 Introduction to Data Science

Project 2 Report

Kerem Şahin

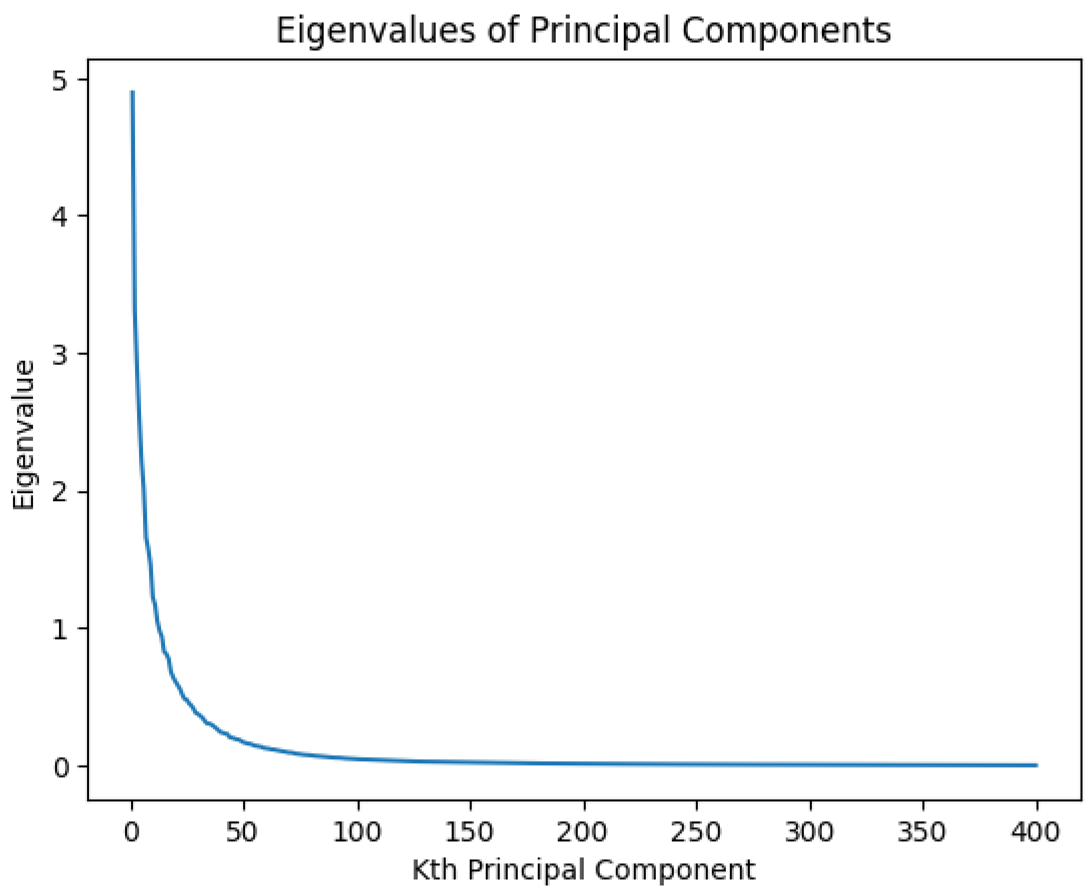
21901724

In order to not interfere with the discussions and interpretations of the result, I have added the libraries I have used and their implementation details at the end of the report.

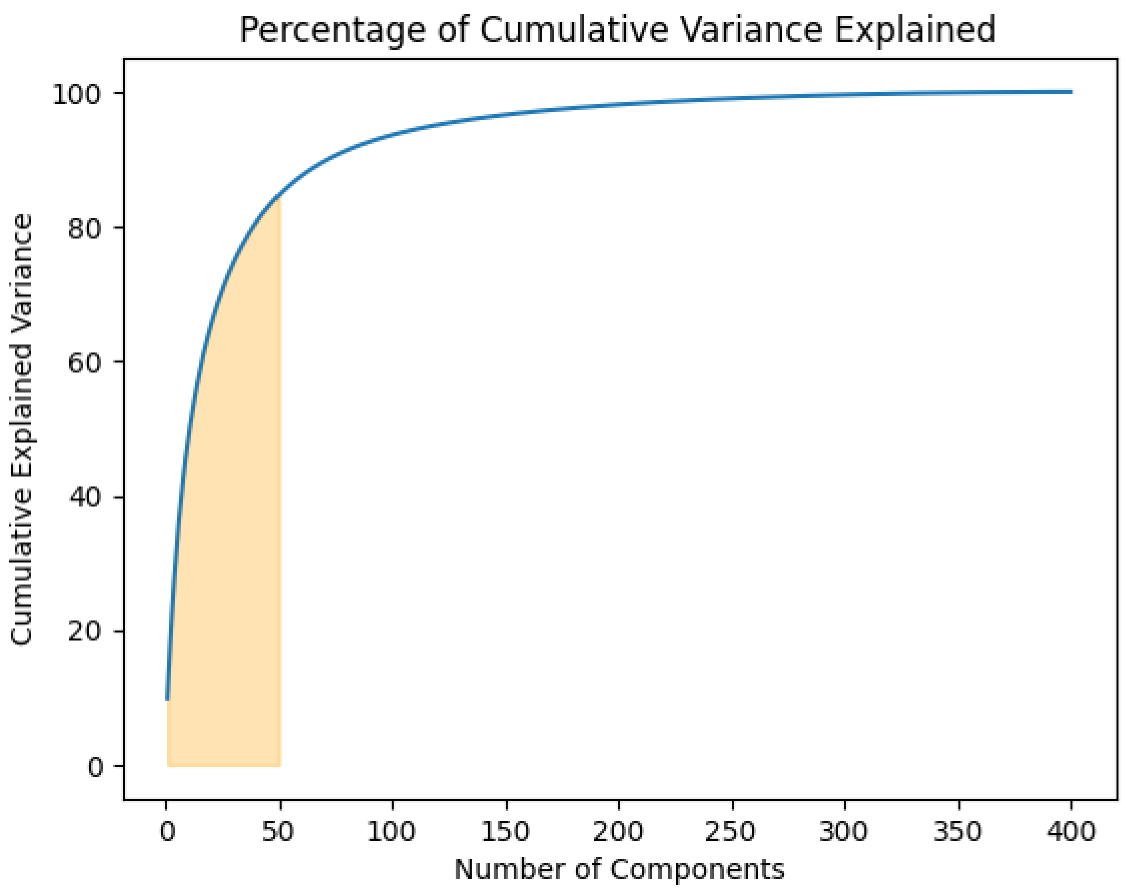
Question 1)

In question 1, I have followed the guidelines provided in the question. Simply, I have used a PCA library and I have used my centered train data for fitting it to the PCA. After that, I have used a Quadratic Gaussian Classifier implementation called as “Quadratic Discriminant Analysis”. The implementation fits a Gaussian for all the different classes and it makes the MLE estimations in accordance with the provided formulas in the Project PDF.

1.2)

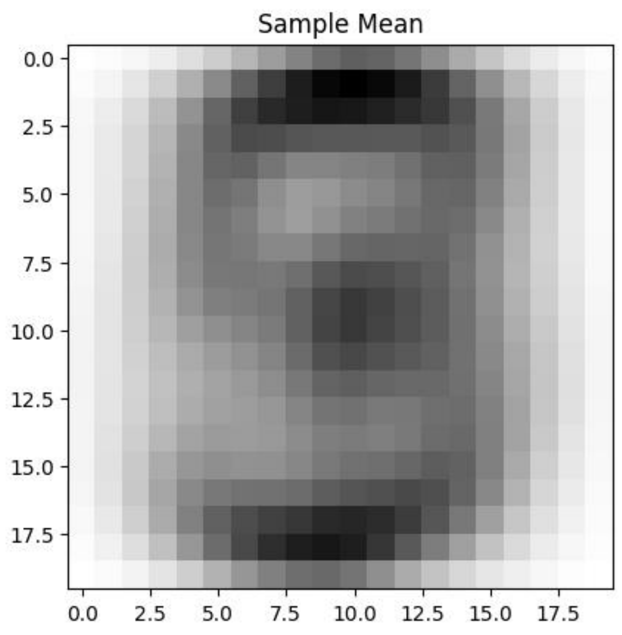


I obtained the figure above when we plot our eigenvectors (principal components) according to their eigenvalues. Since we have 400 features, the maximum number of principal components we can obtain is 400, which are all plotted with their eigenvalues. The eigenvalues obtained will be proportional to the amount of variance that our principal components will explain. When we look at the figure, we can realize that the first principal components that the first couple principal components have drastically higher eigenvalues. The decrease is non-linear and the decrease is much quicker up until approximately the 50th principal component. After this point, the eigenvalues are slowly converging to zero. Therefore, by just looking at the plot, I would guess that the first 50 eigenvectors would be a decent choice for PCA dimension reduction. On top of that, we can also plot a cumulative PVE (Cumulative Proportion of Variance Explained) graph in order to understand the amount of variance explained by the first 50 eigenvectors.

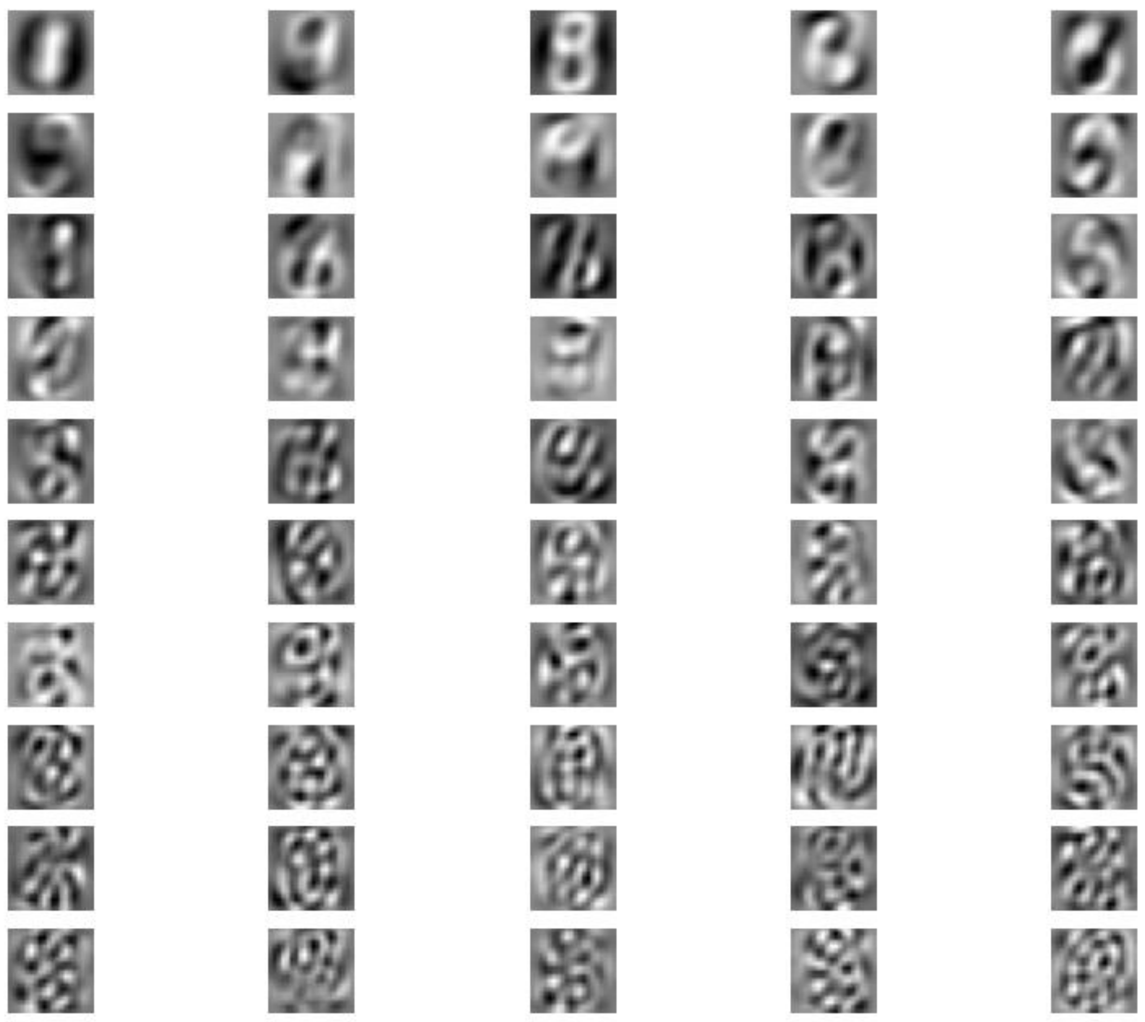


When we look at the cumulative variances explained by 50 eigenvectors, we can see that it corresponds to approximately 85% of the variance. Therefore, our guess from the previous plot proved itself to be useful because a variance of 85% can be interpreted as a value that will be able to retain the majority of the information gathered from our data, and it also uses only 1/8 of all of the eigenvectors, which may be considered as a sufficient reduction of dimensions for our analysis.

1.3)



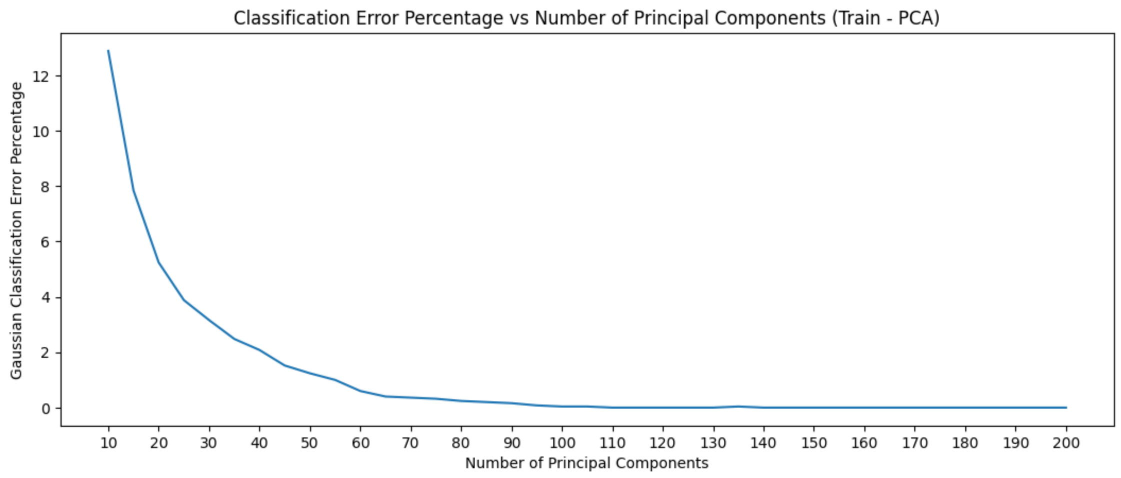
First 50 Principal Components

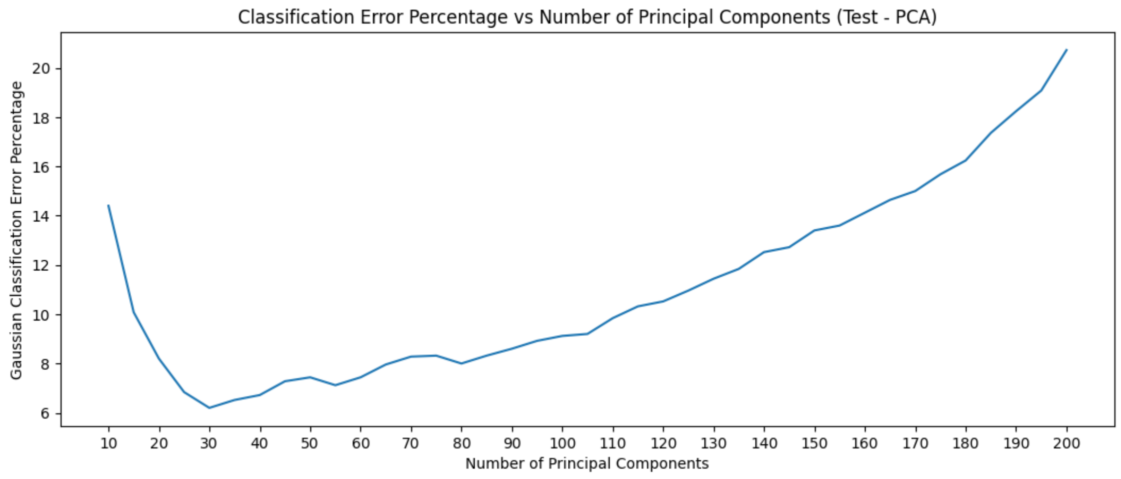


I used a “gray_r” coloring in order to visualize the data. Before I look at the sample figure, logically, I would guess that since the mean will be a mean of the pixels, I would expect to see an image that is a blurry image that is somewhat resembling certain aspects of all the digits. Also, I would expect certain aspects that are dominant across various digits to be more visible. For example, the digits 0, 8, 9, and 6 contain circular shapes within themselves. Therefore, I would expect to see this circular shape be contained in the mean more visibly. When we look at the sample mean, we can see that this expectation is satisfied. We can see traces of certain numbers more densely, including 3, 9, and 8.

As for Principal Components, from our knowledge, we know that the first couple of principal components retains the most variance. This means that the first components are expected to capture more general structures of the digits. As we go down to lower-variance valued principal components, we would expect them to capture more details from the images, instead of capturing general structures. In the image, the eigenvalues of the eigenvectors decrease from left to right, and from top to bottom. When we look at the first eigenvector image, we can realize a 0-shaped structure. As I have told before, since this circular-curved shape is contained in various images, the first principal component captures the 0-shaped structure. As we look at the 2^{nd}, 3^{rd} … eigenvectors, we are capturing images that are looking like digits such as 9, 8, 3, 7… The second image being close to the digit 9 makes sense because digits such as 6 and 3 have similar structures, which means that a shape that resembles 9 also contains high variance. We have also made a similar argument when discussing the mean of the images and said that the dominant structures of the data are going to be more visible. This suggests that it is not a coincidence for the mean image and for the first couple of principal components to resemble each other. Lastly, as we go down to the bottom of the figure, the images are not very interpretable and they are most likely capturing minor details across the obtained samples such as the curvatures, minor differences in the shapes due to different handwritings, densities of the pixels, and thickness. The bottom ones may capture the curves of different digits (e.g. the curve at the top of the digit 6 may differ for different handwritings, which may be captured by the lower valued eigenvectors).

1.5)





I have used eigenvectors in the range from 10 to 200 in increments of 5 (10, 15, … 200), and have obtained the figures above which show the percentage of the error.

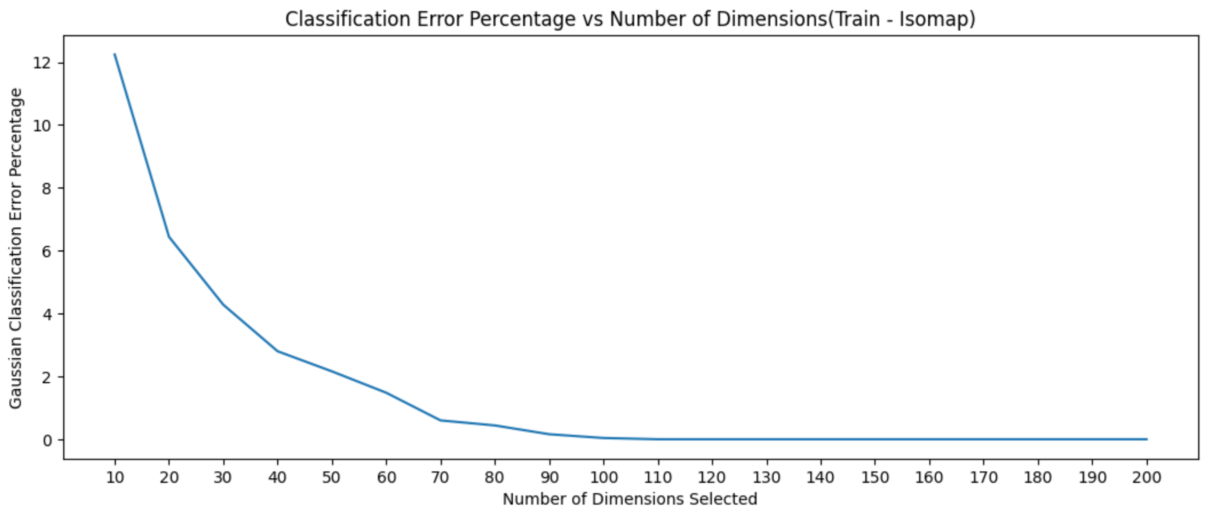
When we look at the plot for the training dataset, we can see a constant decrease in the classification error as we increase the number of principal components. The reason for this is the fact that we have used our train dataset for obtaining the principal components and we have also used the transformed version of the train dataset in order to train the Gaussian. When we increase the number of dimensions of the Principal Components, they will better represent the train dataset. When the number of principal components gets as much as 200, it will retain nearly all of the train data information, and therefore the Gaussian is able to predict it with nearly a 0% error as the number of dimensions increases. The model is basically predicting the data that it has already learned.

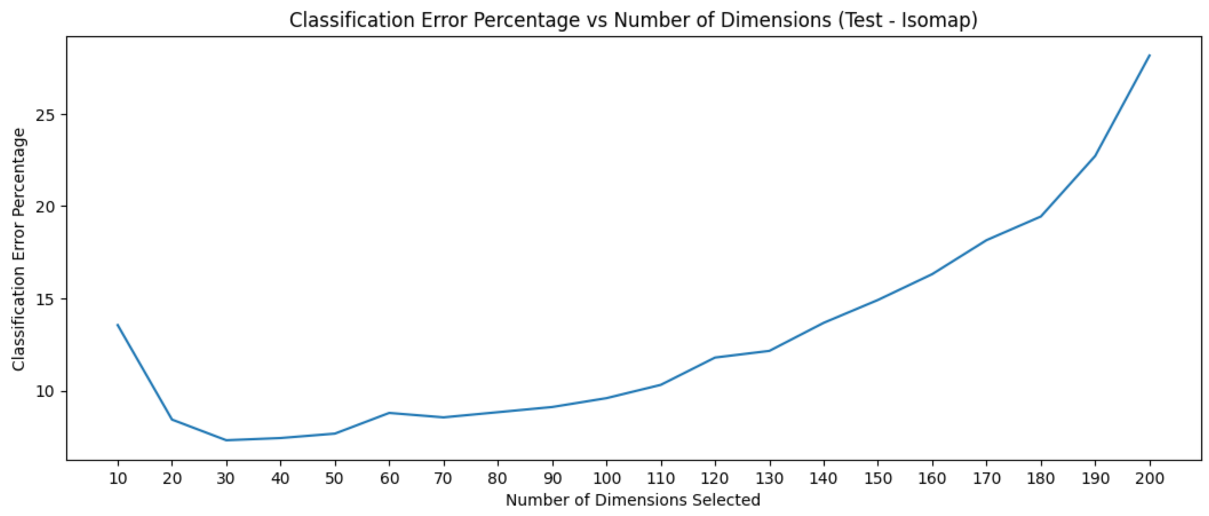
However, for the test dataset, we can see that this is not the case. Different than the train dataset, the test dataset is unseen from our model, and therefore its error percentages will not be the same as the train dataset. If we look at the graph for the test dataset, we can see that the lowest error percentage is approximately 5% with a dimension of 30. Also, unlike the train dataset graph, the error percentages do not constantly decrease, but they actually start to increase again after 30 dimensions. This means that after a certain point, we start overfitting the data, and consequently we are fitting the noise of the train dataset into our model instead of trying to capture the underlying generalizable patterns. In this case, the model will not be able to capture the general underlying pattern of the data. This causes the error percentages to increase after a certain dimension. As for the high error percentage in low dimensions such as 10, we can argue that there is not enough information to create generalizable patterns of the data. This time, opposite to the overfitting argument, the reason for low error percentages is the lack of data.

Question 2)

2.3)

I have used the Isomap implementation from sklearn.manifold. I have only changed the n_component parameter in the range from 10 to 200 with increments of 10. The parameter denotes the number of dimensions. Other than that, parameters such as n_neigbours have been modified. The default number of neighbors is 5. However, since we have enough samples in our data (approx. 500 sample for all digits) I have decided to increase the n_neighbours to 50. When I did this, the error percentages have gone down. However, after 50, even if the errors were decreasing slowly, the computation time was increasing too much. Therefore, I decided to stick with 50. But I should note that with a higher n_neighbour, I may have gotten an even less error score. Also, while fitting the Isomap, I used all the data as specified. After that, I changed the number of dimensions and gave it to the Gaussian Classifier to obtain the predictions.





The results gathered by the Isomap are parallel to the pattern we have observed with PCA. For the training, we can see that as the number of dimensions selected decreases, our error % constantly decreases and comes close to zero. This is again the same reason with the PCA. Since we are using the training_data for training our Quadratic Gaussian Classifier, as we increase the dimensions of the Isomap, our dimensions will retain more and more information which can represent the training data. Although we have given the whole data for our isomap, since the training data is also contained in it, the results are parallel to what we have observed in PCA. Similarly, the pattern for the test prediction is the same. The error % first goes down up until a certain point, and then it increases again when we come to a point of overfitting. The error percentages are the lowest at 30 dimensions, with an error percentage of ~7%. In conclusion, the interpretations of train and test dataset classification error percentages are very similar to PCA.

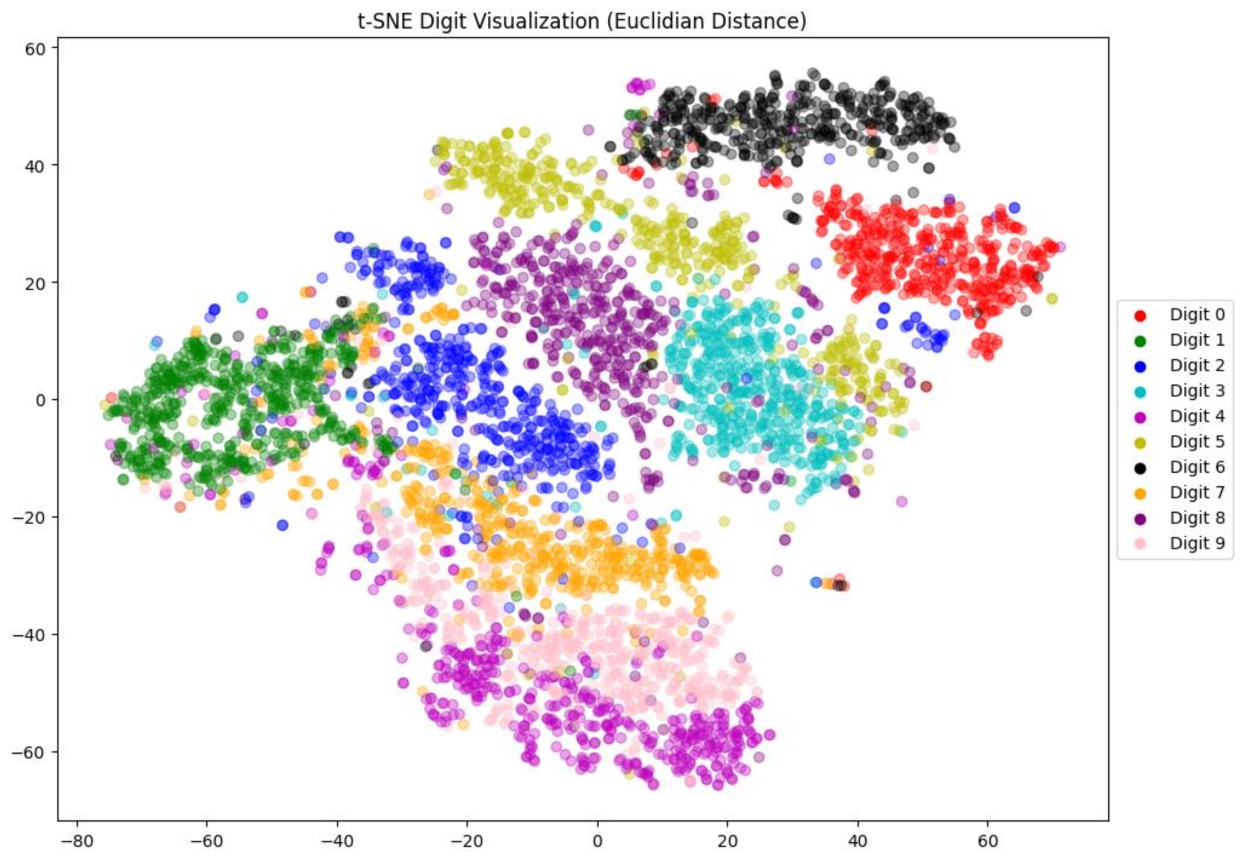
When we compare PCA and Isomap, we can see that both have the potential to represent the data with a fairly low classification error in similar dimensions. PCA achieves an error percentage of 6% with 30 components, whereas Isomap achieves a 7% error with 30 dimensions. This means that when using our data for Gaussian classification, both can provide a fairly good representation of the dataset. However, there is an important thing to note. First of all, due to the computation time required, I choose not to increase the n_neighbours further. However, if we were to increase it, the general trend was downwards and we may have gotten slightly smaller error percentages. This may be caused by the core differences between PCA and Isomap. PCA is a linear method for dimensionality reduction. However, Isomap is a non-linear method for dimensionality reduction. Therefore, the pattern that both of these methods capture will differ. Since Isomap is able to capture different non-linear relations, it may be better at capturing digit information that is more complex, which is not capturable by linear methods.

According to our results, we have obtained the following insights. In general, both dimension reduction methods are applicable to our dataset, and both can retain important information about digits in smaller dimensions. Both methods have achieved their lowest error percentage at 30 dimensions, which may suggest that the general trends that are embedded inside our dataset is represented fairly well in 30 dimensions. Both have achieved an approximate error score between 6-7%. Lastly, there are certain tradeoffs between the models. The computation power required by Isomap is higher than the PCA, and PCA achieved a relatively low error percentage and the computation power required is minimal. On top of that, if we were to tune certain hyperparameters on a faster computer with a bigger dataset, Isomap may have provided even better error percentages as it can capture more complex non-linear patterns. Due to its fast computation time and relatively well accuracy score, I would argue that PCA might be a better choice for our small and simple dataset. However, as I noted before, this may change according to the reasons I have pointed out previously.

Question 3)

3.3)

I have used the t-SNE implementation from sklearn.manifold in order to display my data. As for the parameters, I have tried out different values for the perplexity, learning_rate, n_iter, and metric parameters, and stuck with the default values for other parameters. By default, the method uses "Barnes-Hut" approximation in order to find the pairwise distance between data points, which reduces the computation time without sacrificing much of the accuracy. Without using the approximation, the running time takes too much time and therefore I stuck with the Barnes-Hut approximation. Perplexity is correlated with the nearest neighbor count. In other words, a lower perplexity is more likely to capture local features, and a higher perplexity is more likely to capture global features. The usual recommendation is between 5 to 50. After several trials, I saw that the value of **40** gave the most logical result. Increasing the perplexity higher than 40 and less than it did not make the graph any better and they gave slightly worse clusters, but the differences are still minor. For the learning_rate, I have decided to use the "auto" selection of the parameter, which gives an optimal learning rate according to the formula " max(N / early_exaggeration / 4, 50)". Since our sample size is 5000 and the default early_exxageration value is 12, the learning rate is approximately founded as **110**. I have tried different rates and found that this result is providing optimal results. The n_iters is the max number of iterations for the optimization. The default value is 1000. When I lowered the value, the clusters were not formed as well as 1000. When I increased it, the execution took longer and the clusters did not necessarily get any better. An interesting difference came from the "metric" parameter. This parameter lets us choose whether to use Euclidian distance or different distances such as "Cosine Distance" in order to form the clusters. Since cosine distance is used for assessing similarity in different areas (Cosine Similarity), I had an intuition to try it out as well. Now let's first look at the graph obtained with Euclidian Distance:

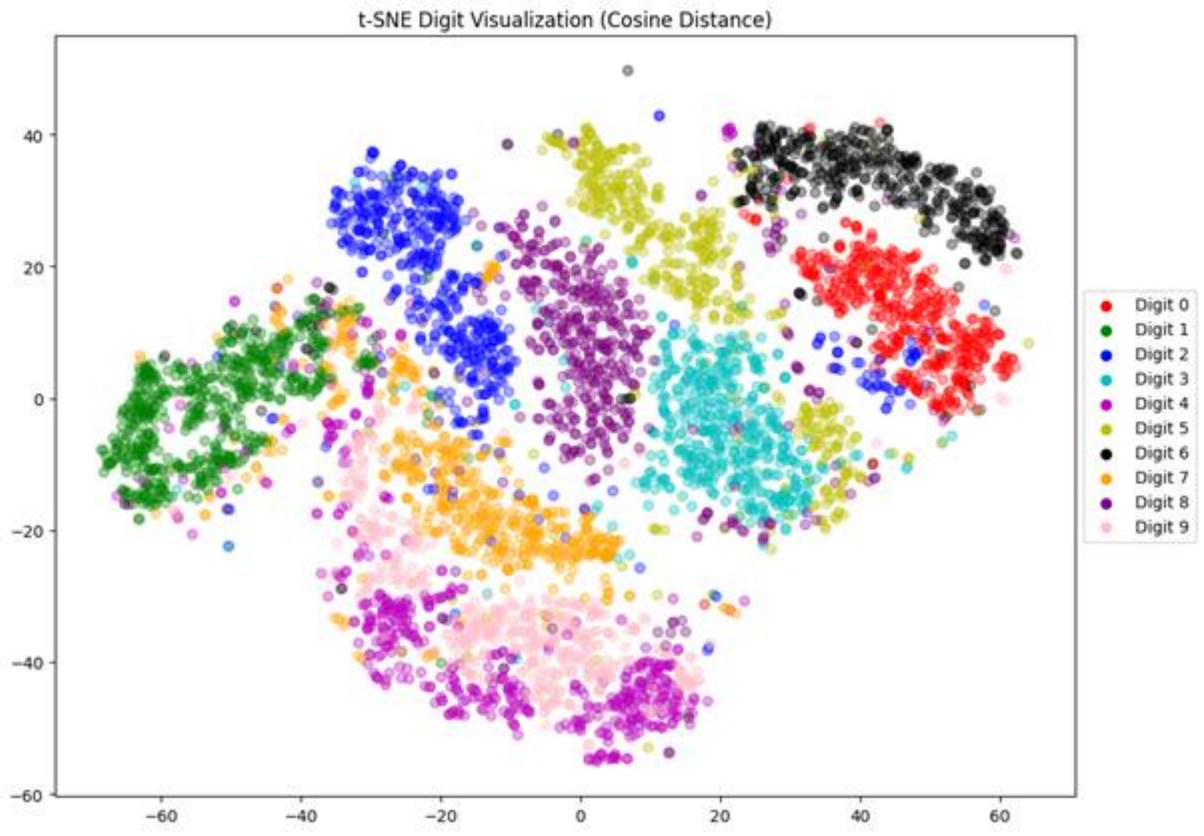


We can see that t-SNE was able to form somewhat distinct clusters for all of the digits. When we look at the image, we can see that certain digits such as digit 0 have a well-separated cluster and it does not show up too much in the boundaries of other clusters. This may be caused by 0 having a fairly easy shape to draw, as compared to other digits and therefore it is not easily mixed with other digits. It only has a certain boundary crossing with the digit 6, which may be caused by the fact that 6 may look like a 0 if the circular shape at the bottom is drawn too big proportional to the downward curve at the top.

Secondly, we can realize certain digits have their boundaries very close to each other and their data points are somewhat mixed. As an example, we can look at the digits 1 and 7. Although they have their unique clusters, there are certain points on the boundary of the green clusters (digit 1) where we also see a lot of orange clusters. This may be caused by the fact that the shapes of 1 and 7 are somewhat similar and their shape may be confused according to different handwritings.

One last observation I have realized is that there are 2 different clusters for the digit 5. Although there are small disconnections in the clusters for some of the other digits as well, 5 has a separated cluster that has an obvious separation with each other. Since t-SNE checks for local similarities, this may indicate that there are two distinct different ways of writing 5. However, this conclusion does not have to be true, and other metrics should be checked before making such a conclusion. This result may be caused by different several factors including an insufficient variety of samples.

Now, we can look at the Cosine Distance graph:



As we can see, most of the shapes found in the previous graph are still valid. However, one can realize that with Cosine Distance, the boundaries of different clusters are not as near to each other as in the case of Euclidian Distance, and therefore the clusters are better separated from other clusters around them. Arguably, this may be a result caused by the power of cosine distance in detecting similarity. However, for example for the digit 4, there is a slight separation when compared to the previous single cluster for the graph with Euiclidian Distance. However, I would still argue that cosine distance produced a good and appropriate clustering visualization for our dataset.

Libraries, Tools, and How they were used

numpy: The library has been used for its array structure while making operations on the data.

sklearn.model_selection: The library has been used for its “train_test_split”, which is used to create randomized, evenly distributed train/test set throughout our analysis.

sklearn.discriminant_analysis: The library has been used for its “QuadraticDiscriminantAnalysis”, which is basically the Quadratic Gaussian Classifier. The underlying structure that the library uses is consistent with the background information provided.

sklearn.decomposition: The library has been used for its “PCA” implementation in question 1. The methods I have used were transform, and fit. I have given the results of the PCA implementation to the Gaussian Classifier, and made my predictions as it was specified in the project explanation.

sklearn.manifold: The library has been used for its “Isomap” and “t-SNE”. I have specified the methods and the parameters I have used in the appropriate sections of the report.

matplotlib.pyplot: Lastly, I have used pyplot in order to plot all my graphs.

Reference

1) NumPy. (n.d.).

   https://numpy.org/

2) sklearn.model_selection.train_test_split. scikit. (n.d.).

   https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

3) sklearn.discriminant_analysis.Quadraticdiscriminantanalysis. scikit. (n.d.-a).

   https://scikitlearn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html

4) sklearn.decomposition. scikit. (n.d.-a).

   https://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition

5) sklearn.manifold. scikit. (n.d.-a).

   https://scikit-learn.org/stable/modules/classes.html#module-sklearn.manifold

6) Matplotlib.pyplot. matplotlib.pyplot - Matplotlib 3.5.3 documentation. (n.d.). https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html