Bilkent University
Department of Computer Engineering

GE 461 Introduction to Data Science

Spring 2024

Project 2

Dimensionality Reduction and Visualization

Görkem Kadir Solun

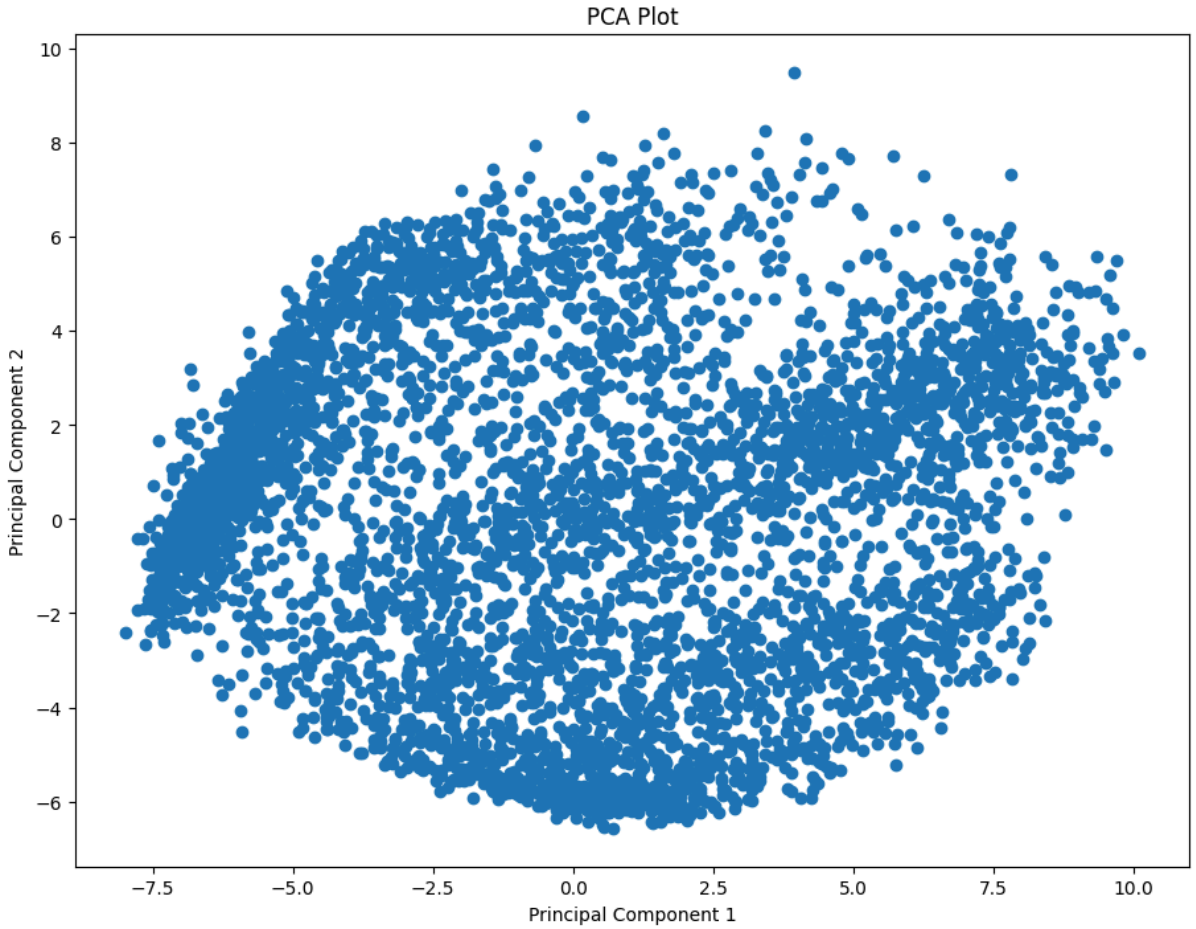
22003214

I completed the project using Python[1] and its libraries, Scikit-learn[2], Numpy[3], Matplotlib[4].

Question 1 PCA

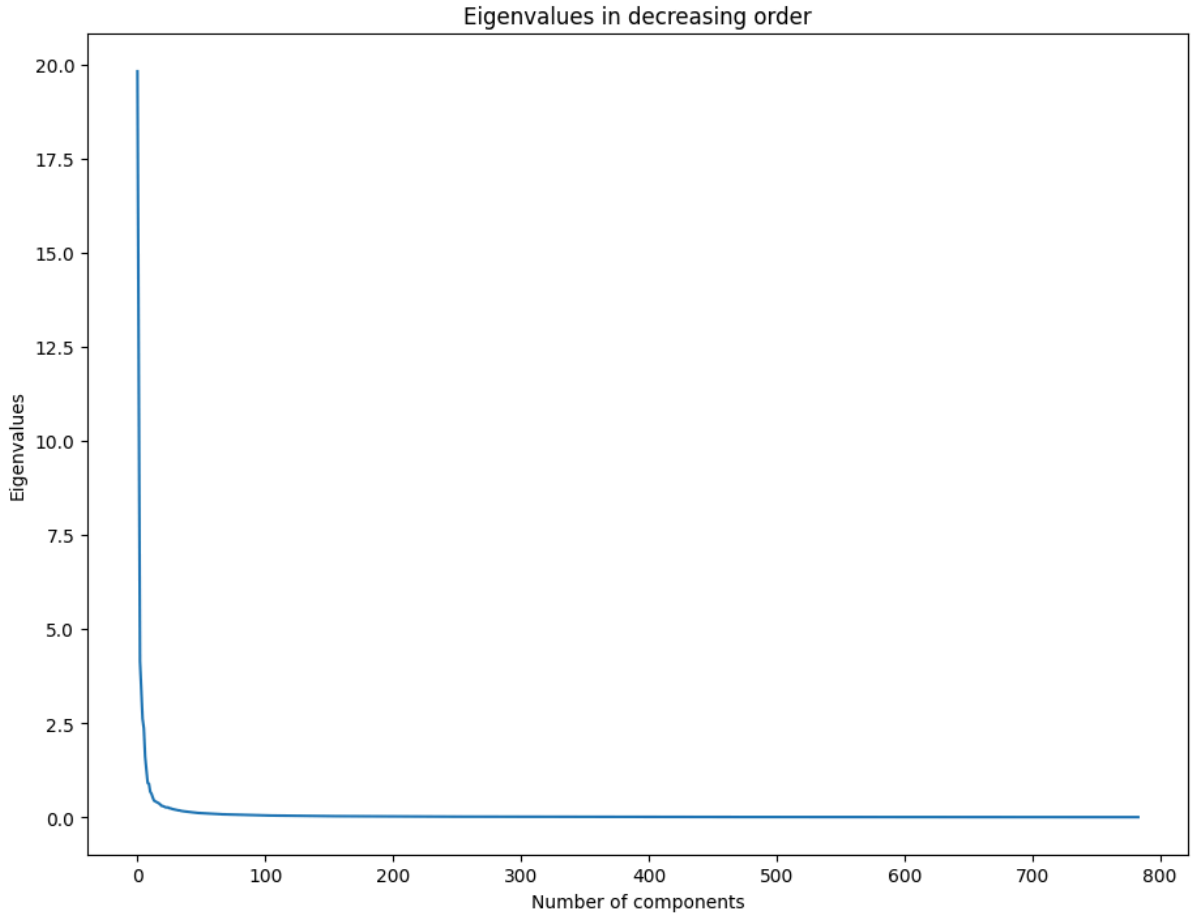
I have followed the guidelines provided in the question.

I have split the data and subtracted the mean; while doing this, I have maintained stable label distribution in the splits.
Then, I implemented PCA manually and used the Scikit-learn[2] library to understand it better. I found out that eigenvalues are the same. I plotted a PCA plot of two prominent dimensions (the first two PCs).

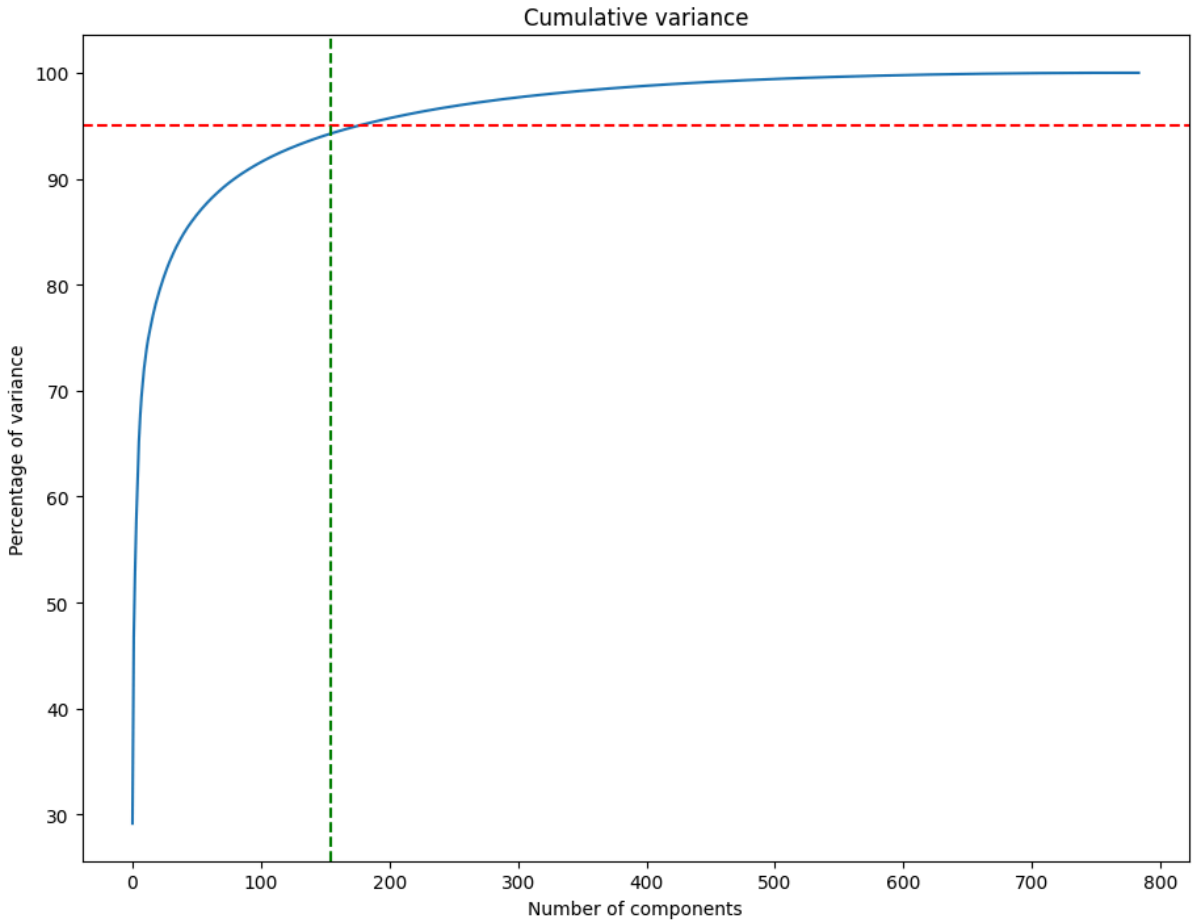


Then, I plotted the eigenvalues in descending order to see which covers most of the variance. I plotted our eigenvectors (principal components) according to their eigenvalues. Since we have 784 features, the maximum number of principal components we can obtain is 784, which are all plotted with their eigenvalues. The obtained eigenvalues will be proportional to the variance our principal components explain. When we look at the figure,

we can realize that the first principal components have drastically higher eigenvalues than the first couple of principal components. The decrease is non-linear, and the decrease is much quicker up until approximately the 150th principal component. After this point, the eigenvalues are slowly converging to zero. Therefore, by looking at the plot, the first 150 eigenvectors would be a decent choice for PCA dimension reduction. On top of that, I plotted a cumulative variance graph to understand the amount of variance explained by the eigenvectors.



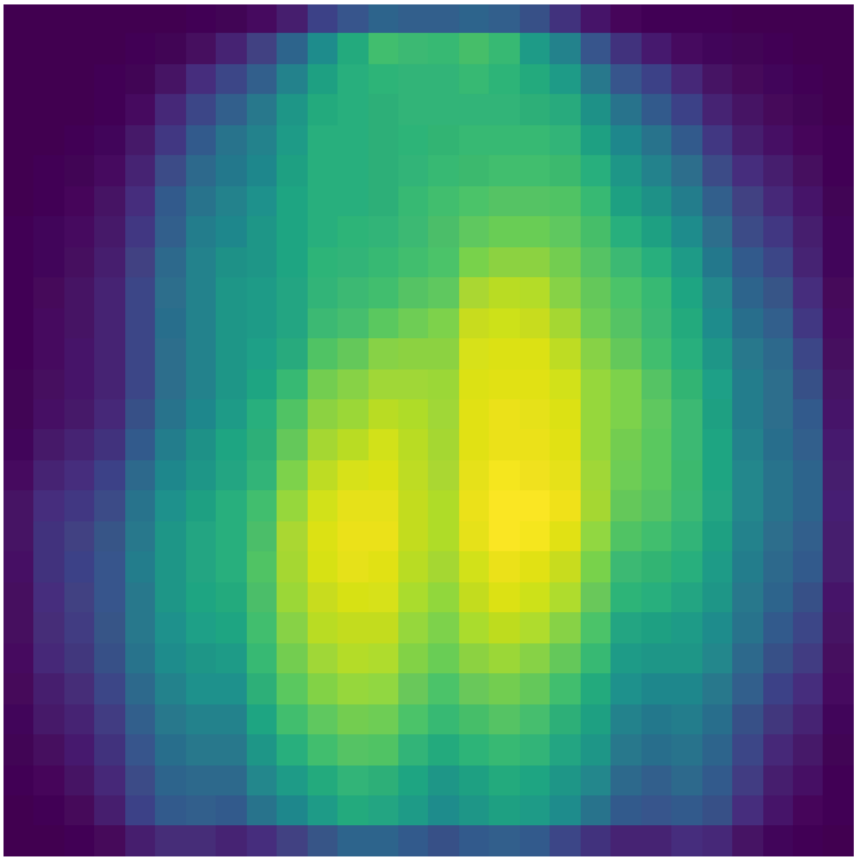
To gain insight into how the proportion of variance changes relative to the number of principal components, I subsequently plotted the cumulative variance. When examining the cumulative variances explained by the first 154 eigenvectors, we observe that they collectively account for approximately 95% of the total variance. Consequently, the estimation from the previous plot proved valuable, as a variance of 95% signifies a threshold that preserves the majority of the information embedded within our data. Additionally, it facilitates a substantial reduction in dimensions for our subsequent analysis.



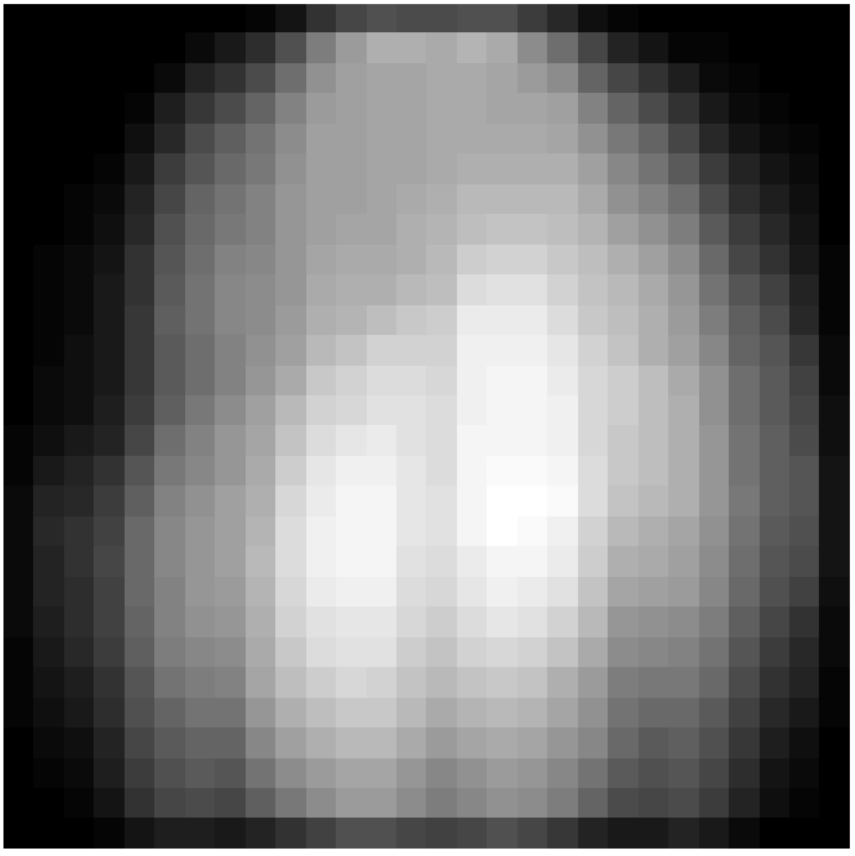
Then, I displayed the mean and PCs as images to visualize.

Since the mean will be a mean of the pixels, I expect to see an image that is a blurry image that resembles certain aspects of all the samples, mainly the clothes. When we look at the sample mean, we can see that this expectation is satisfied. We can see traces of clothes and shoes, with a shoe-like brightness in the middle.

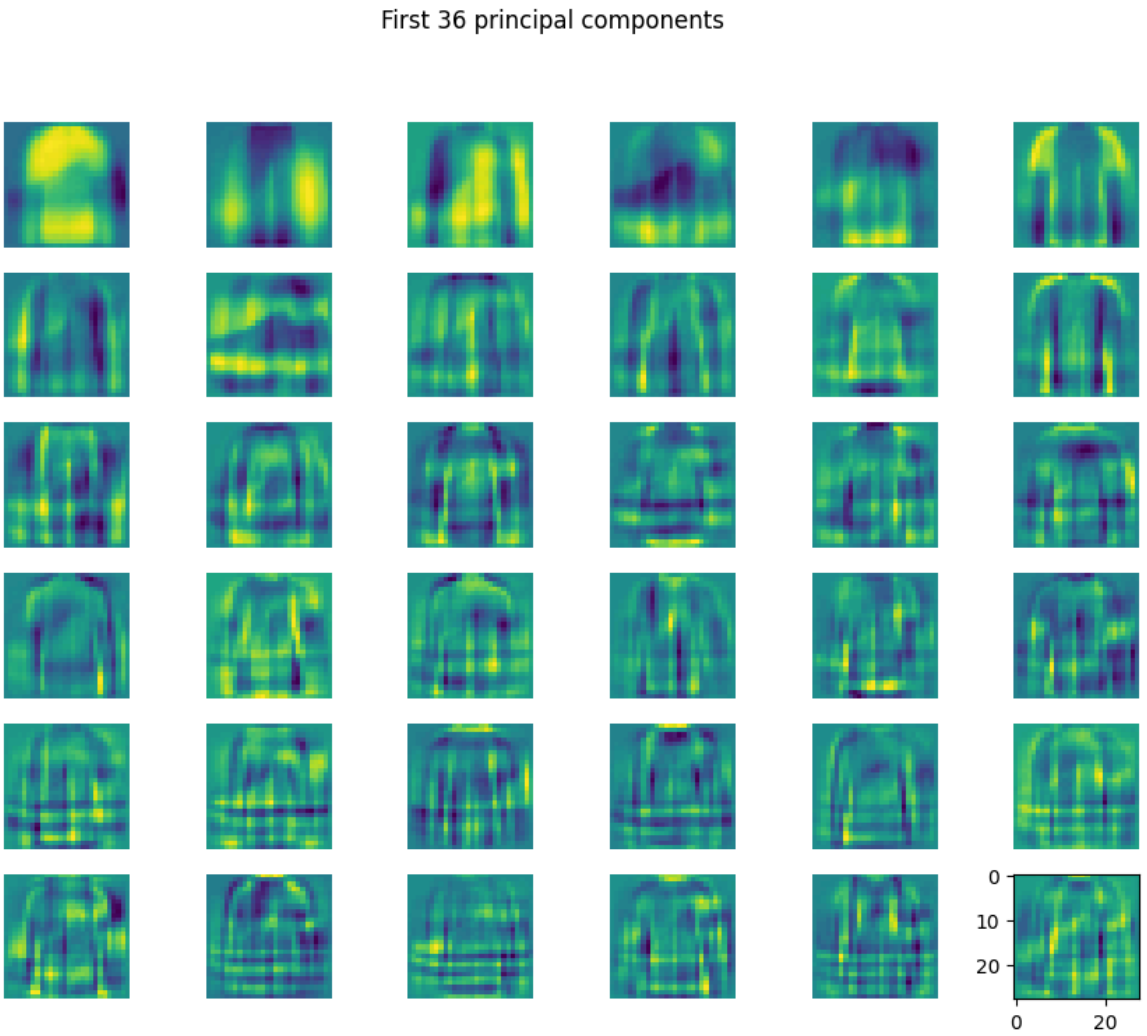
Mean image

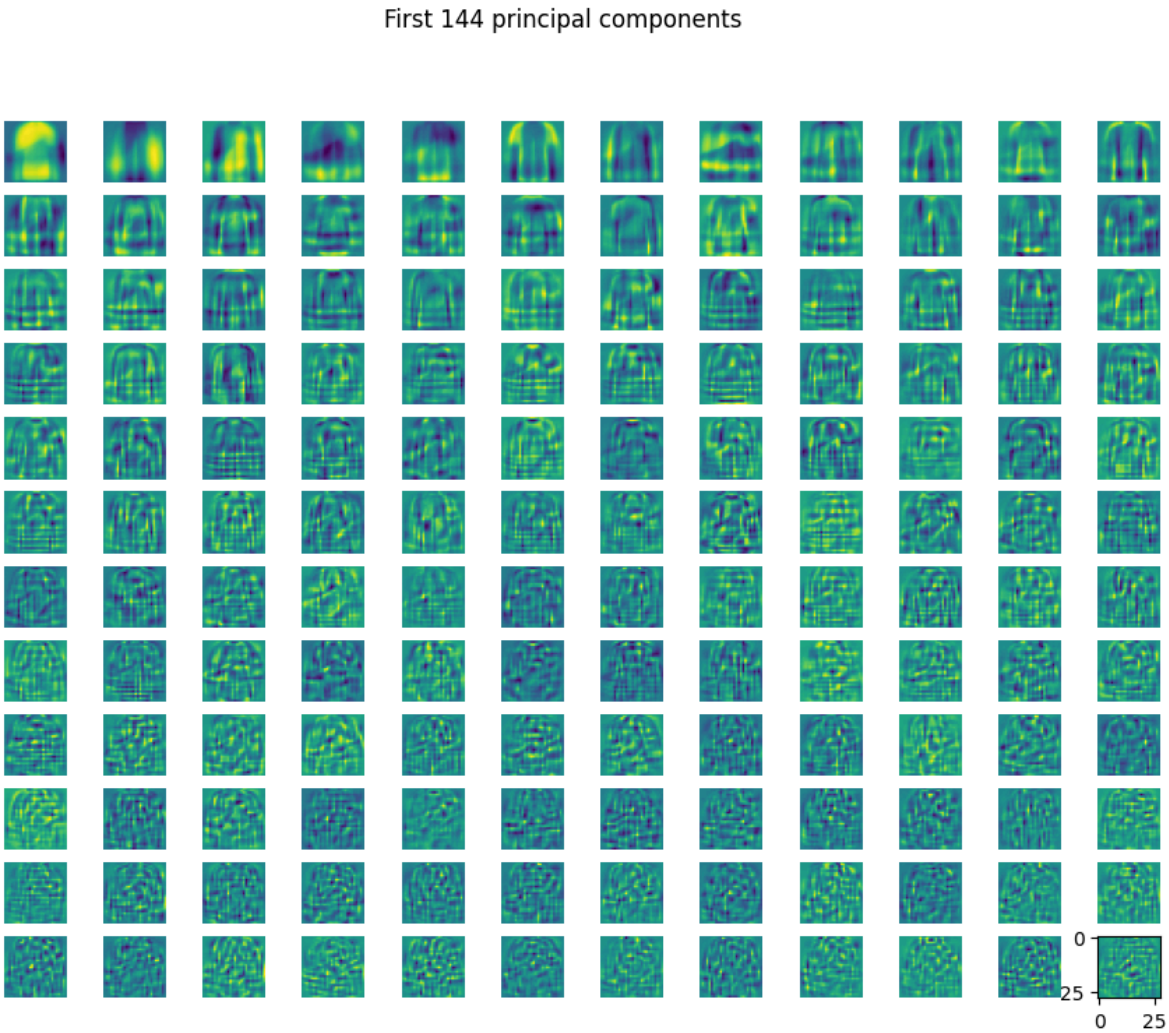


Mean image (grayscale)



As for principal components, we know that the first couple retains the most variance, meaning that the first components are expected to capture more general structures of the samples. 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 see a shirt and a shallow shoe. We capture images like trousers and pullovers as we look at the second and third eigenvectors. The second image has bright arms and a dark trouser-like sense, which means this eigenvector has a shape resembling trousers and shirts and contains high variance. We have also made a similar argument when discussing the mean image. We said that the data's dominant structures will be more visible, meaning it is not a coincidence for the mean image and for the first couple of principal components to resemble each other. The images are not very interpretable as we go down to the other eigenvectors. They are most likely capturing minor details across the obtained samples.

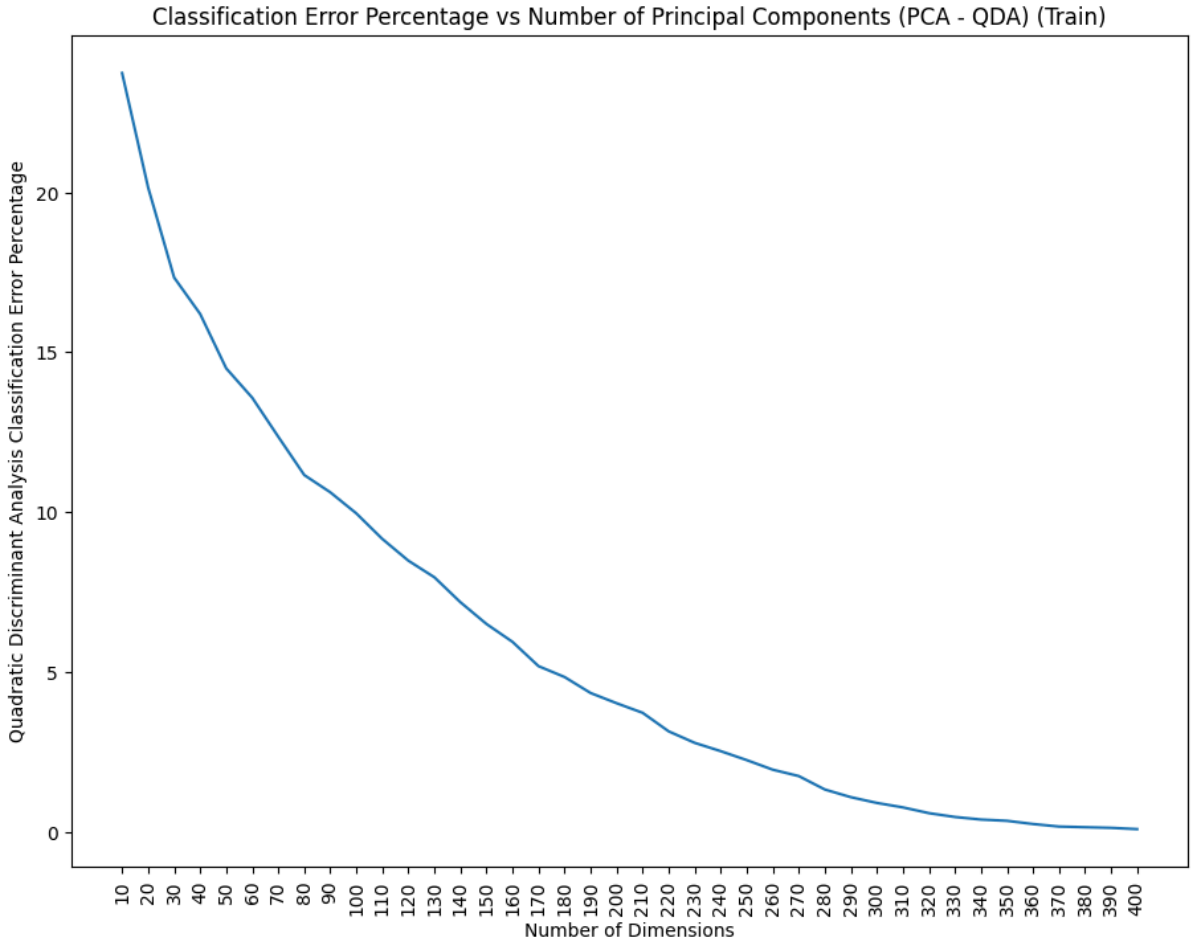


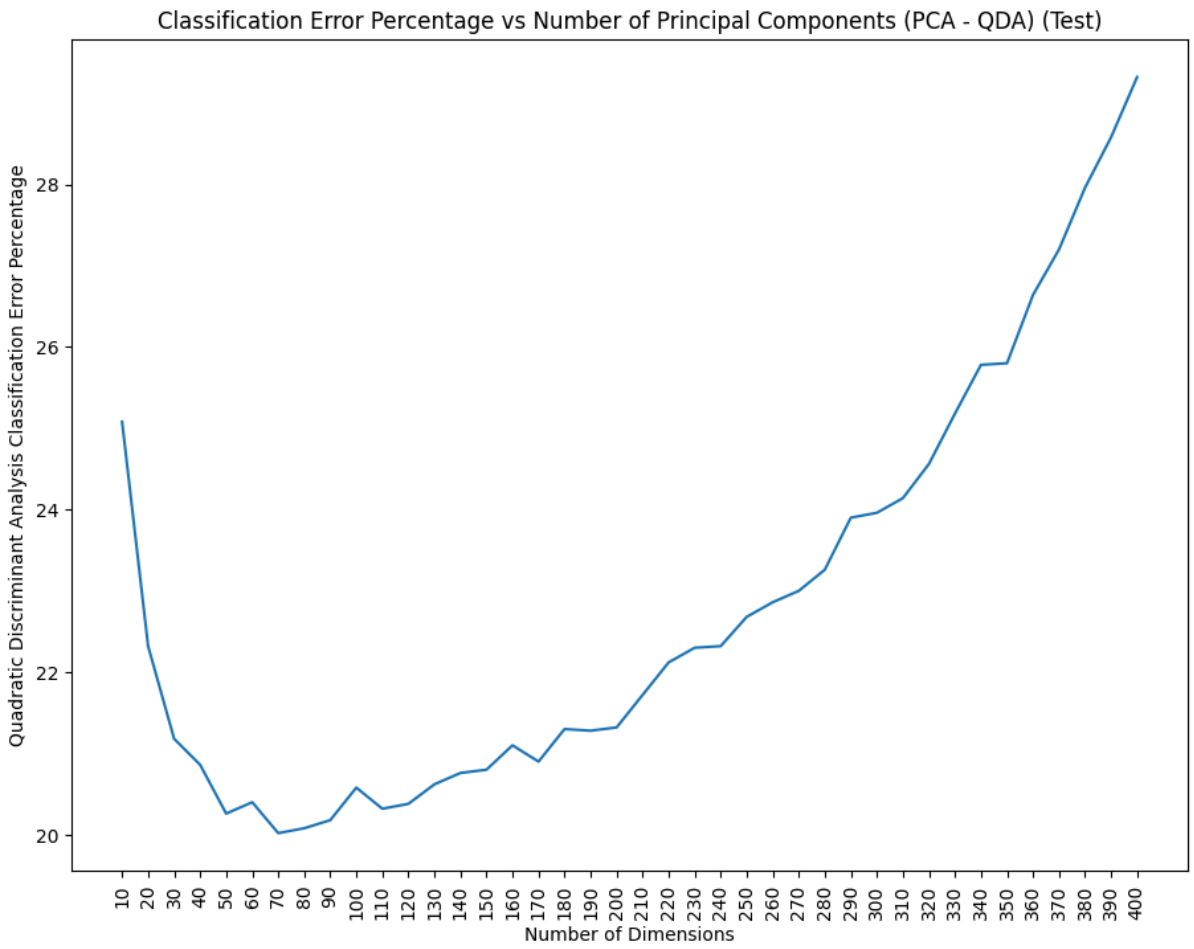


After that, I chose different subspaces (40) and trained two Gaussian classifiers (QDA, GNB) on them. I chose these Gaussian classifiers because QDA allows different covariance matrices for each class (explained in the project document), and GNB assumes features are independent. These allowed me to see differences between dependencies in the data. I used eigenvectors from 10 to 400 in steps of 10. I have obtained the figures below, which show the error percentages.

When we look at the QDA training dataset plot, we can see a constant decrease in the classification error as we increase the number of principal components. This is because we have used our train dataset to obtain the principal components, and we have also used the transformed version of the train dataset to train the QDA. Increasing the dimensions will better represent the training dataset as QDA considers dependencies. When the number of principal components is 400, it will retain nearly all the train data information. Therefore, the QDA can predict it with nearly a 0% error as the number of dimensions increases. The model predicts the data it has already learned, and an overfitting issue may exist.

However, we can see that this is different for the test dataset. Unlike the training dataset, the test dataset is unseen from our model; therefore, its error percentages will not be the same as the training dataset. If we look at the graph for the test dataset, we can see that the lowest error percentage is approximately 20% with a dimension of 70. Also, unlike the train dataset graph, the error percentages do not constantly decrease but increase after, meaning that after a certain point, we start overfitting the data. Consequently, we are fitting the noise of the training 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; thus, the error percentages increase after a particular dimension. As for the high error percentage in low dimensions, such as 10, there is insufficient information to create generalizable data patterns.

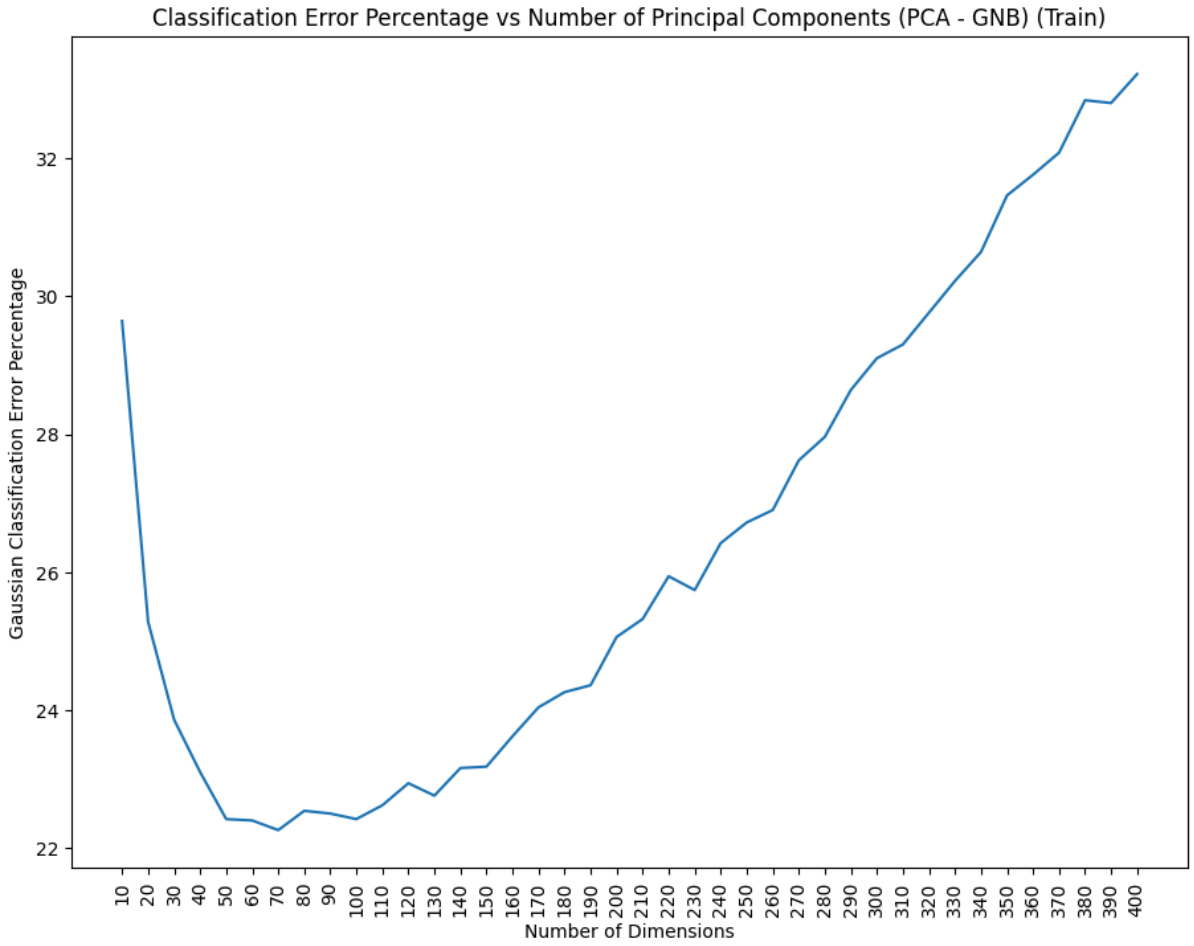


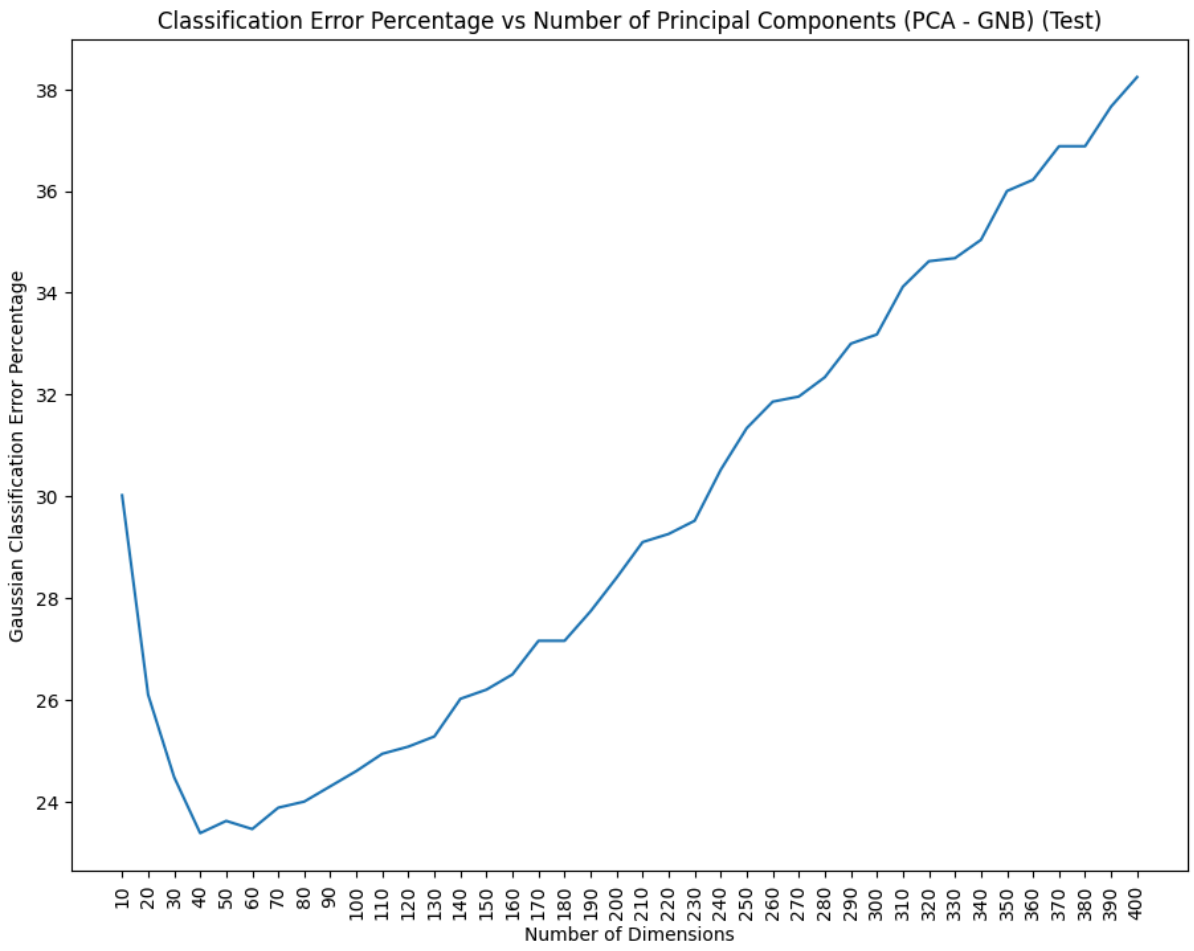


Unlike the QDA training dataset plot, the GNB training and test dataset plot shows a similar structure to the QDA test dataset plot. This is due to GNB's assumption that every feature is independent. As a result, increasing the number of dimensions initially decreases the error but eventually leads to an increase.

When the number of principal components is 400, in contrast to the QDA training dataset, we are fitting the noise of the training dataset. Therefore, the error of the GNB training datasets is high. The model predicts the data it has already learned with independence of features, and an overfitting issue does not exist. We can see that this is the same for the test dataset. However, the test dataset is unseen from our model; therefore, its error percentages will not be the same as the training dataset. The test datasets are somewhat higher. If we look at the graph for the test dataset, we can see that the lowest error percentage is approximately 23% with a dimension of 40. In both dataset graphs, the error percentages do not constantly decrease but increase after specific dimensions, meaning after a certain point. Consequently, we are fitting the noise of the training 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; thus, the error percentages increase after a particular dimension.





Question 2 Random Projections

I have skipped the eigenvalues part (2) and visualization part (3) in the previous question.

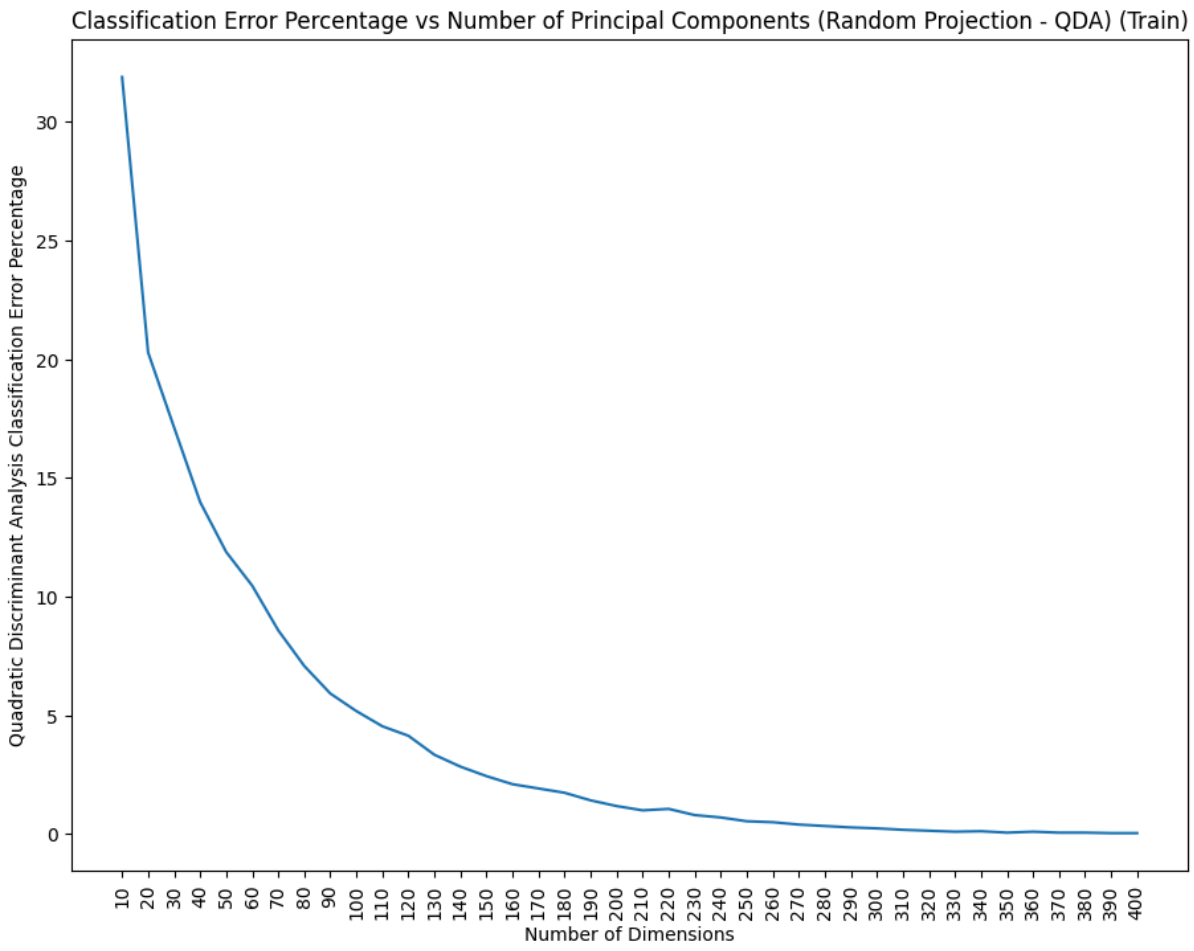
I generated random projection matrices for the subspaces of dimensions 1 to 400. I used dimensions from 10 to 400 in steps of 10. Random projections offer a faster way to reduce dimensionality compared to PCA. Instead of computing computationally expensive eigenvectors, random projections are used, as the name suggests, as a randomly generated matrix to project data into lower dimensions. Surprisingly, this method often preserves essential information.

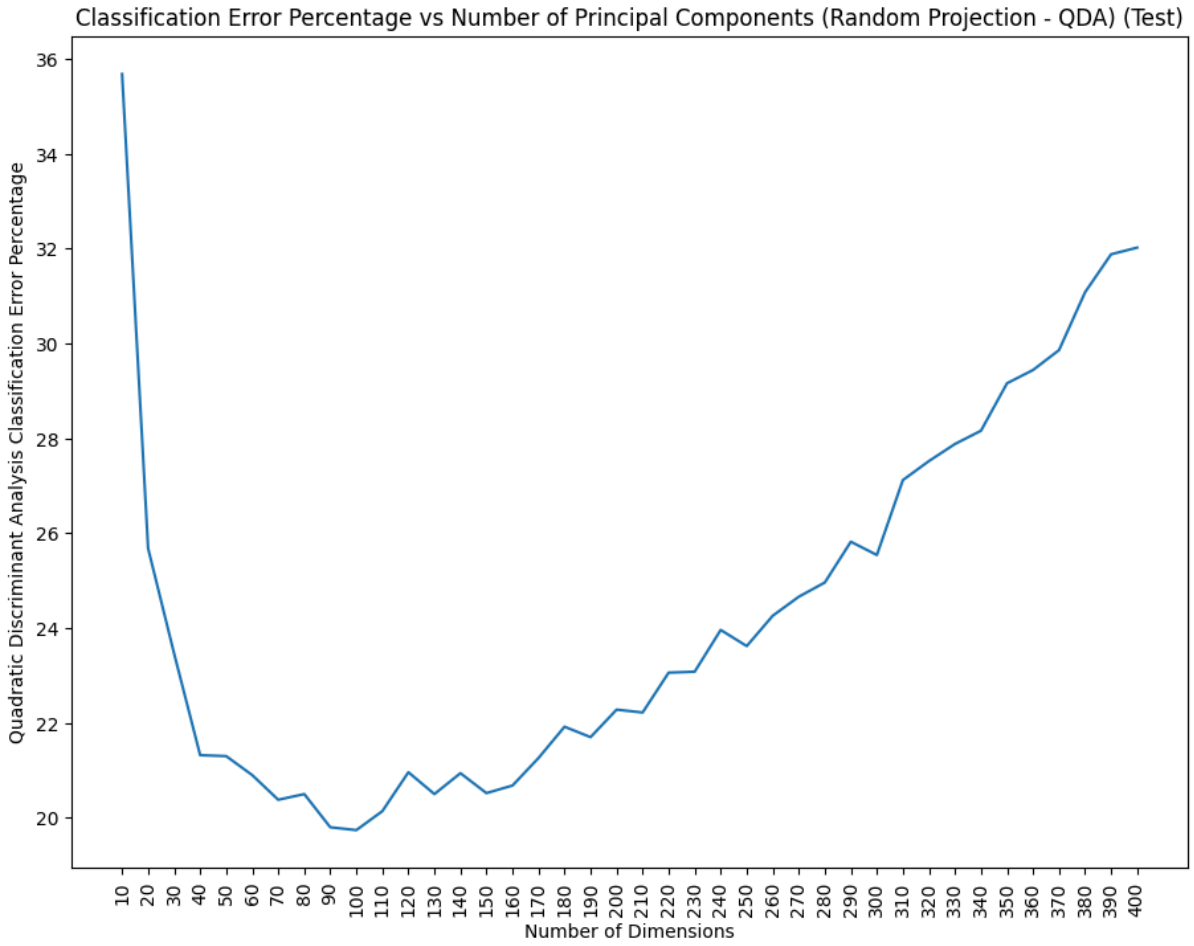
Again, I chose different subspaces (40) and trained two Gaussian classifiers (QDA, GNB) on them. I chose these Gaussian classifiers because QDA allows different covariance matrices for each class (explained in the project document), and GNB assumes features are independent. These allowed me to see differences between dependencies in the data. I have obtained the figures below, which show the error percentages.

When we look at the QDA training dataset plot, we can see a constant decrease in the classification error as we increase the number of principal components. This is because we have used our train dataset to obtain the principal components, and we have also used the transformed version of the train dataset to train the QDA. Increasing the dimensions will better represent the training dataset as QDA considers dependencies. When the number of principal components is 400, it will retain nearly all the train data information. Therefore, the QDA can predict it with nearly a 0% error as the number of dimensions increases. The model predicts the data it has already learned, and an overfitting issue may exist.

However, we can see that this is different for the test dataset. Unlike the training dataset, the test dataset is unseen from our model; therefore, its error percentages will not be the same as the training dataset. If we look at the graph for the test dataset, we can see that the lowest error percentage is approximately 20% with a dimension of 100, noting that this may change as it is random. Also, unlike the train dataset graph, the error percentages do not constantly decrease but increase after, meaning that after a certain point, we start overfitting the data. Consequently, we are fitting the noise of the training 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; thus, the error percentages increase after a particular dimension. As for the high error percentage in low dimensions, such as 10, there is insufficient information to create generalizable data patterns.

We observe a distinct structure when comparing random projection to PCA in the QDA training dataset. In higher dimensions, the error percentage is lower than in lower dimensions. This could be attributed to the randomness, but it can also be explained by the fact that random projection might not select dimensions with higher variances.

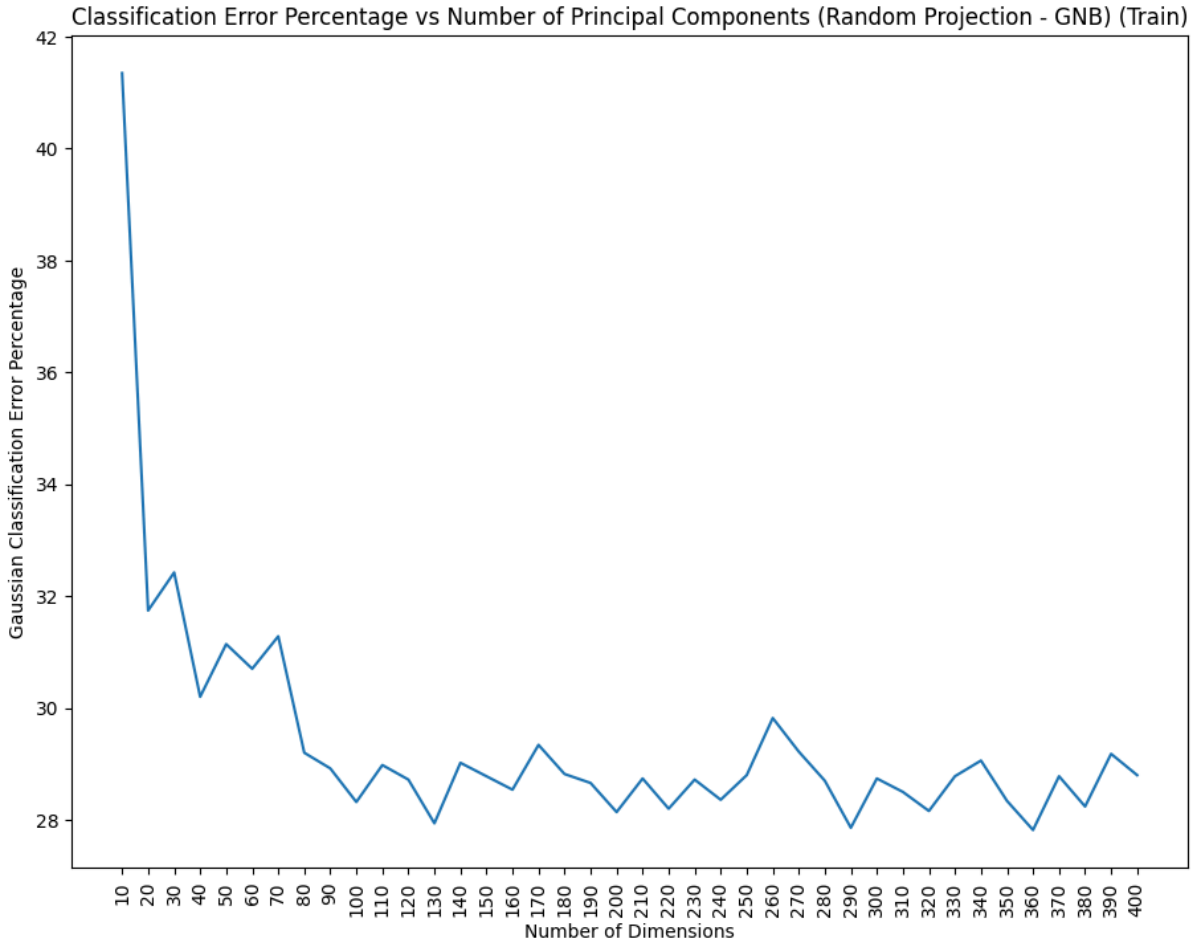

Classification Error Percentage vs Number of Principal Components (Random Projection - QDA) (Train)
Quadratic Discriminant Analysis Classification Error Percentage
0
5
10
15
20
25
30
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400
Number of Dimensions

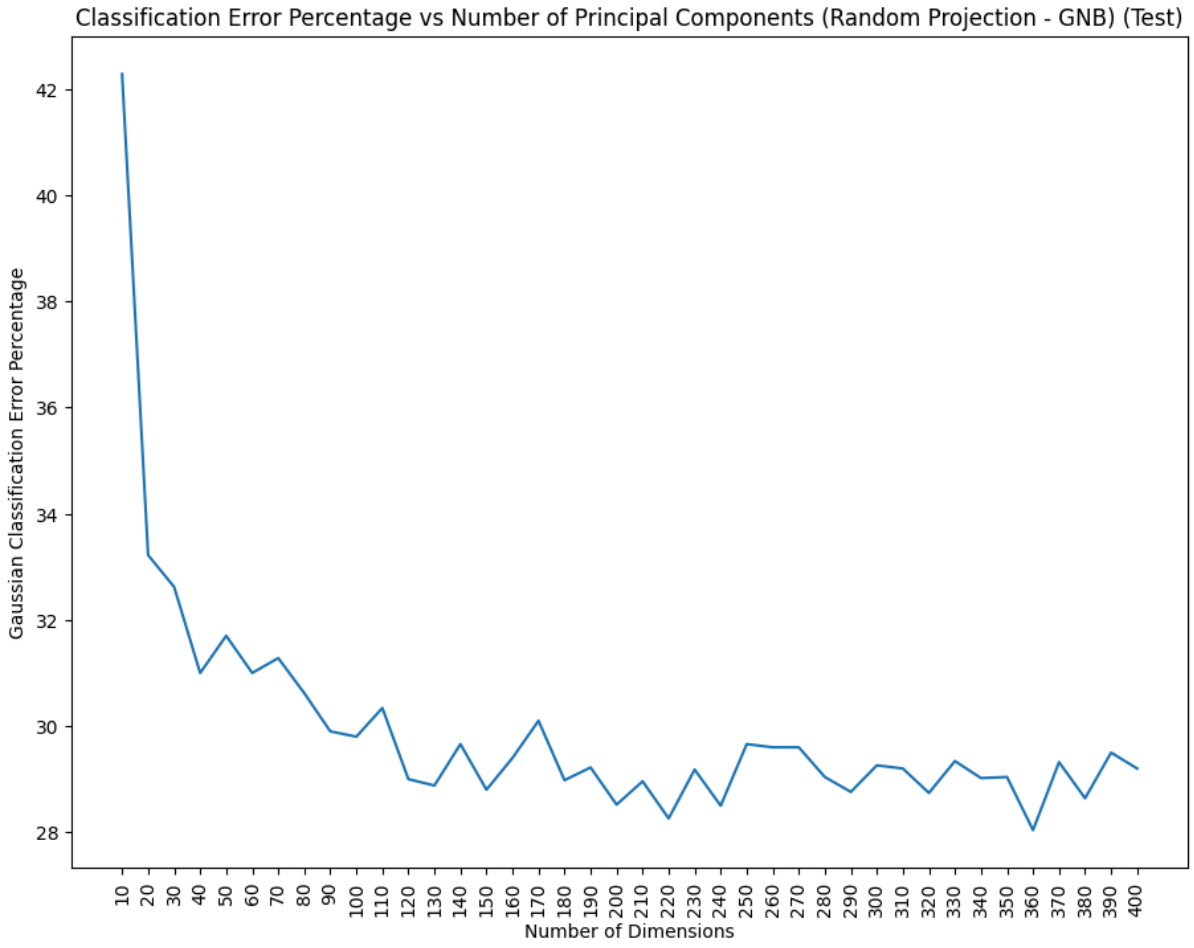


When comparing the GNB training and test dataset plots with the PCA and QDA test and training datasets of random projection, we observe a distinct structure with higher percentages than all other graphs. This difference arises from GNB's feature independence assumption and the randomness that random projection introduces. Consequently, increasing the number of dimensions first reduces the error to a certain extent, followed by fluctuations.

When the number of principal components is near 400, in contrast to the previous datasets (excluding QDA training) dataset, we are fitting the noise of the training dataset. However, the randomness introduced decreases its error by up to 4%, noting that this may not be true for all cases as it is random. Moreover, randomized projection does not induce overfitting as the dimensionality increases. Instead, the graph exhibits fluctuations of around 28%. Therefore, the error of the GNB training datasets is high and fluctuates. The model predicts the data it has already learned with independence of features, and an overfitting issue does not exist. We can see that this is the same for the test dataset. However, the test dataset is unseen from our model; therefore, its error percentages will not be the same as the training dataset. Again, in contrast to previous GNB dataset results, the test datasets are not higher

but very similar. Looking at both graphs, we can see that the lowest error percentage is approximately 28% with various dimensions. In both dataset graphs, the graph's structure is similar, meaning the local minimums and maximums tend to occur at the same dimensions.

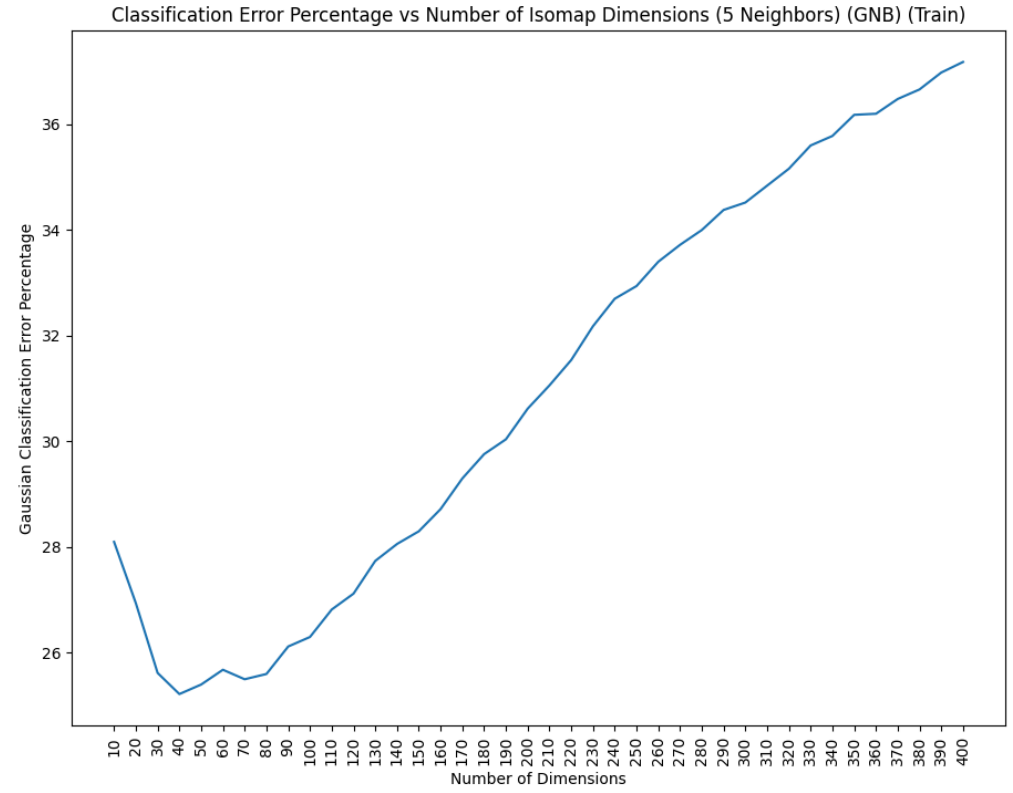


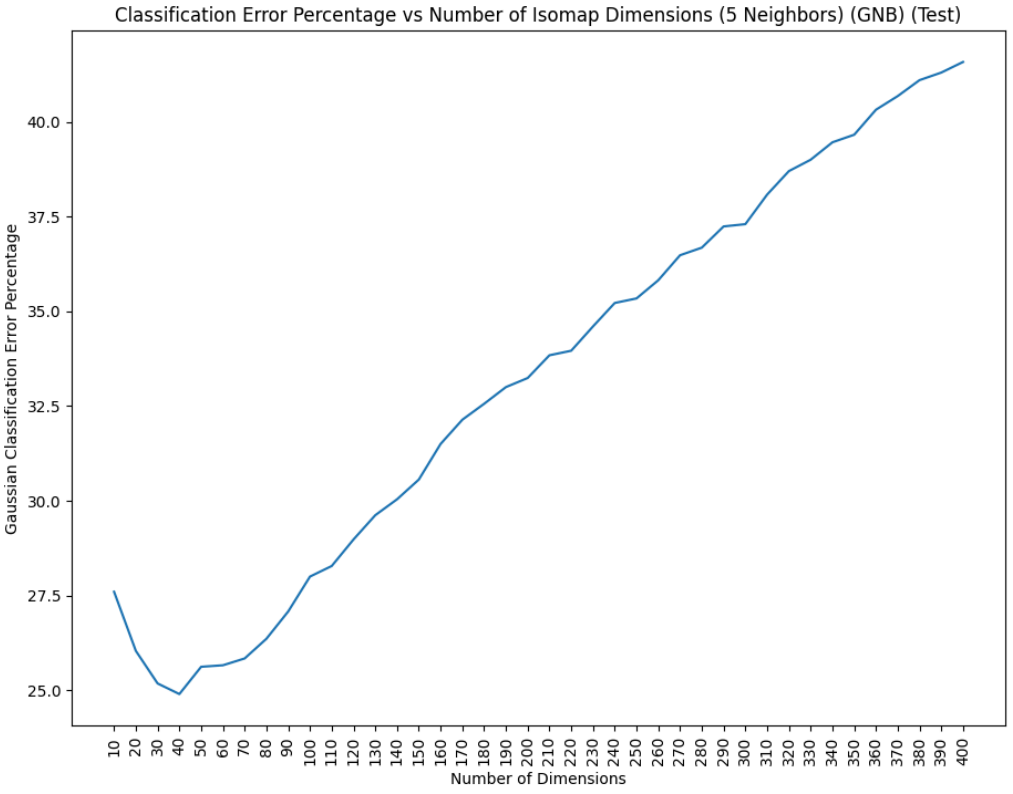


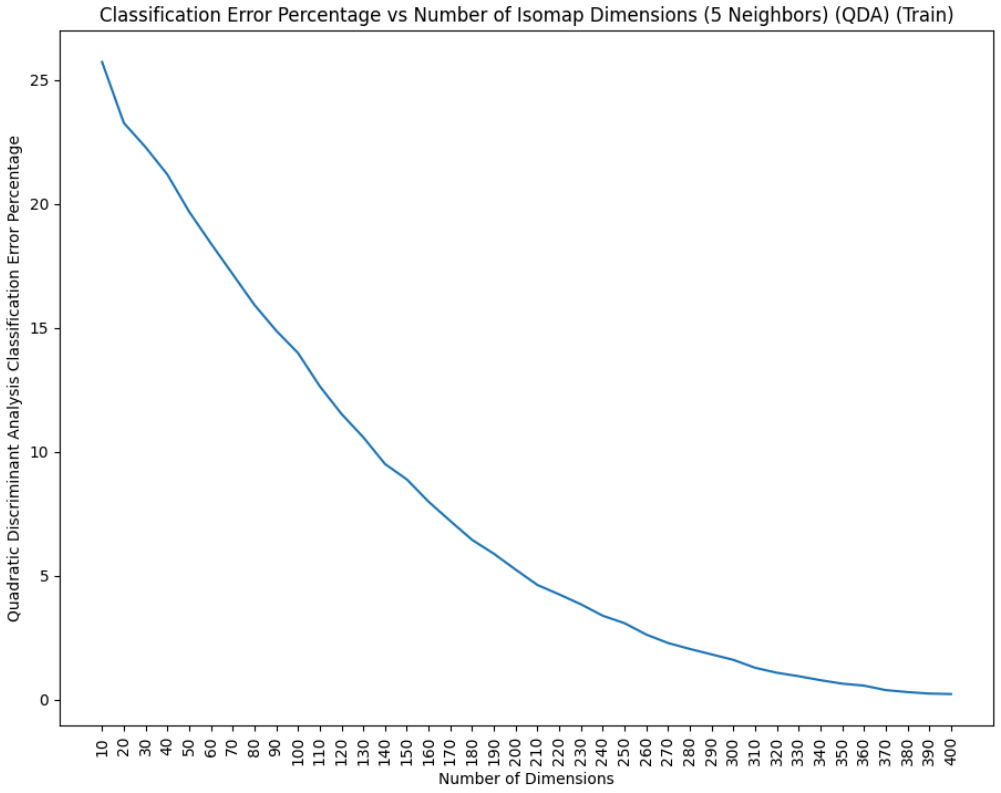
Question 3 Isomap

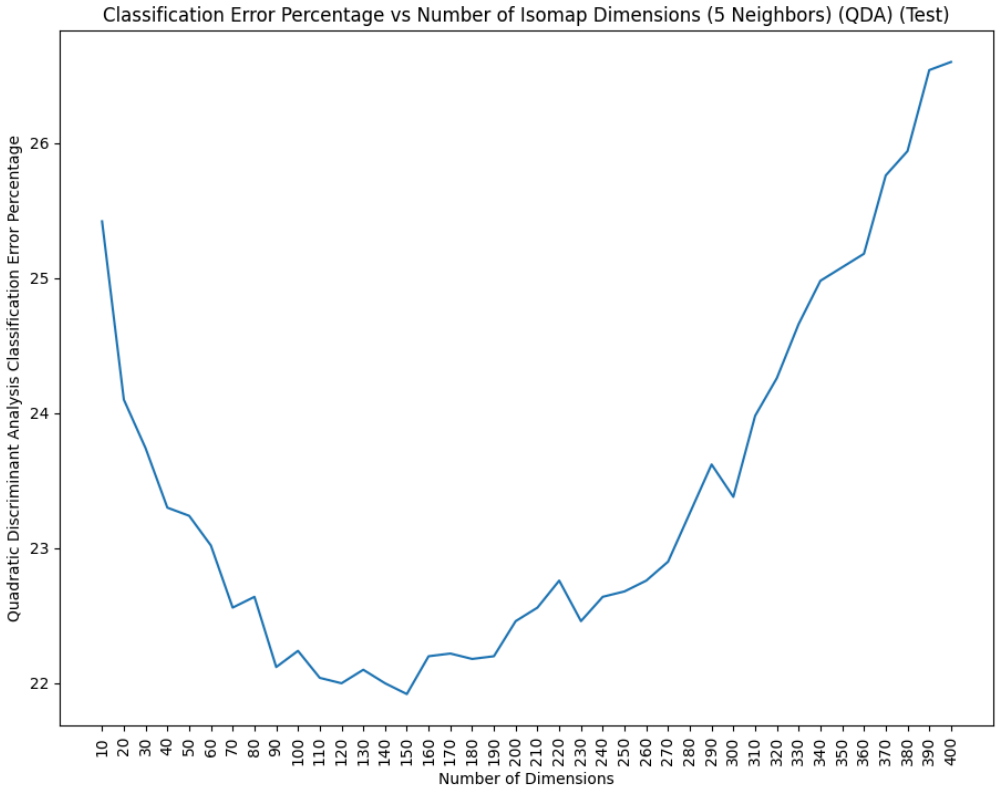
I used the Isomap implementation from the Scikit-learn[2] library. I have only changed the n_component parameter (dimensions) and used dimensions from 10 to 400 in steps of 10. The other parameter I have changed is n_neighbors. I used 5 (default) and 20. With 20 as n_neighbors, error percentages are somewhat lowered, but there is not much difference even if computation time has increased. I may have gotten a lower error score, but the computation time is very high. I trained two Gaussian classifiers (GNB and QDA) to get these results as in the previous steps.

Graphs with n_neighbor is 5:

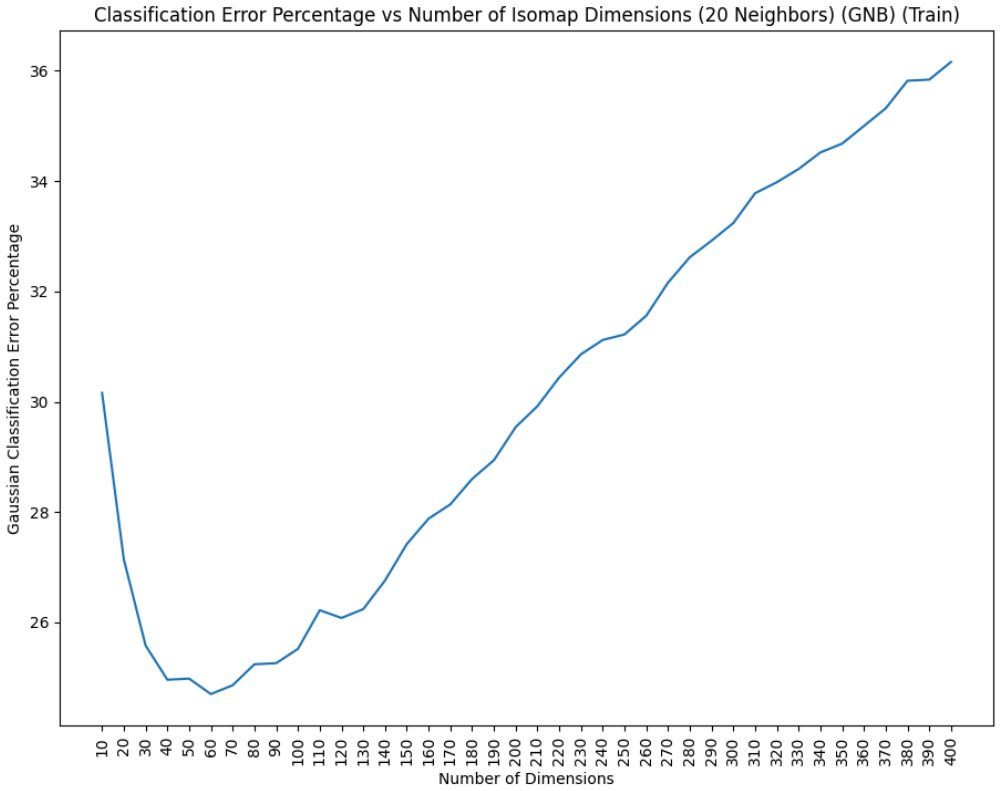

Classification Error Percentage vs Number of Isomap Dimensions (5 Neighbors) (GNB) (Train)
Gaussian Classification Error Percentage
Number of Dimensions

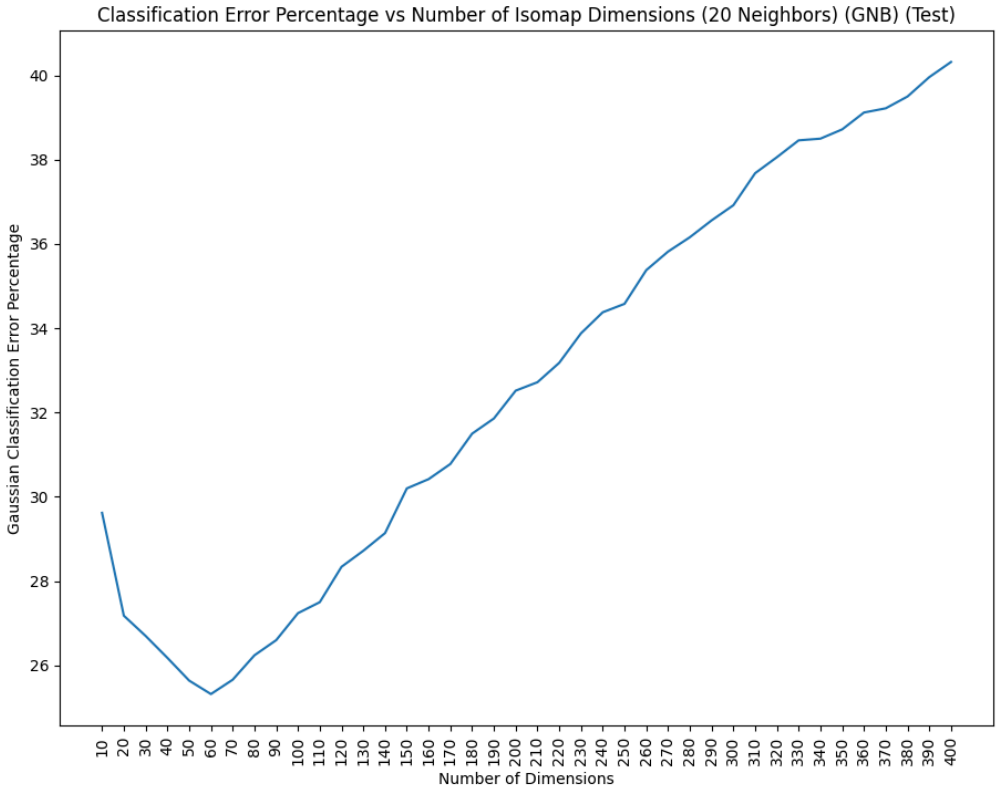

Classification Error Percentage vs Number of Isomap Dimensions (5 Neighbors) (GNB) (Test)
Gaussian Classification Error Percentage
Number of Dimensions

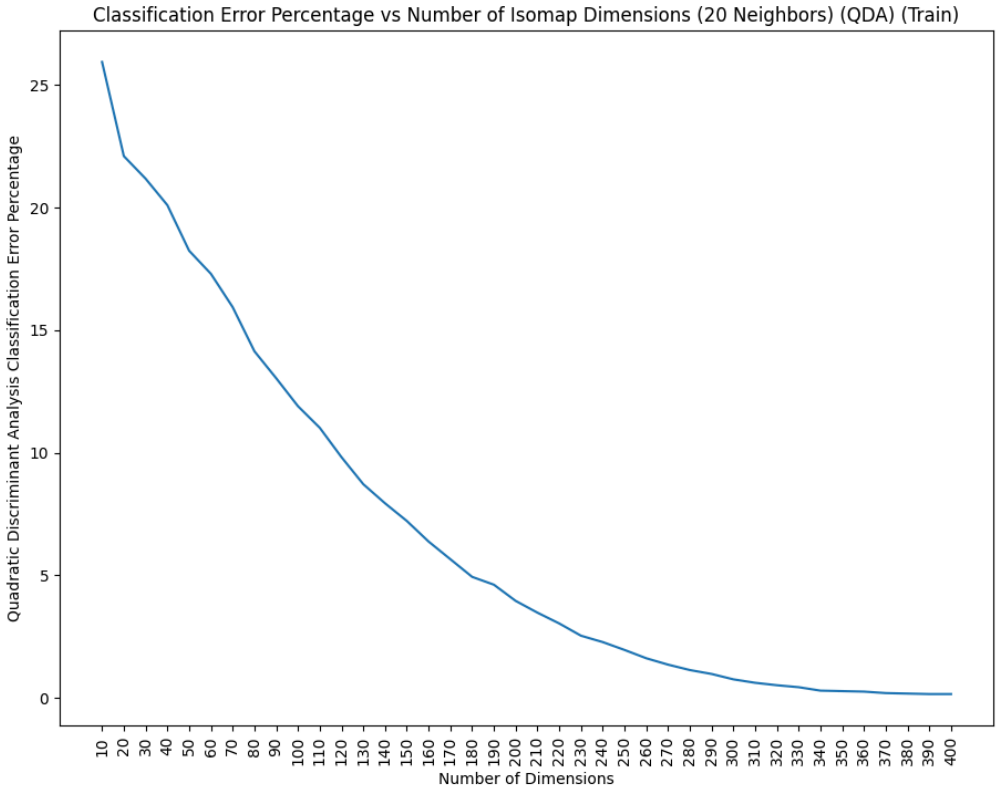

Classification Error Percentage vs Number of Isomap Dimensions (5 Neighbors) (QDA) (Train)
Quadratic Discriminant Analysis Classification Error Percentage
0
5
10
15
20
25
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400
Number of Dimensions

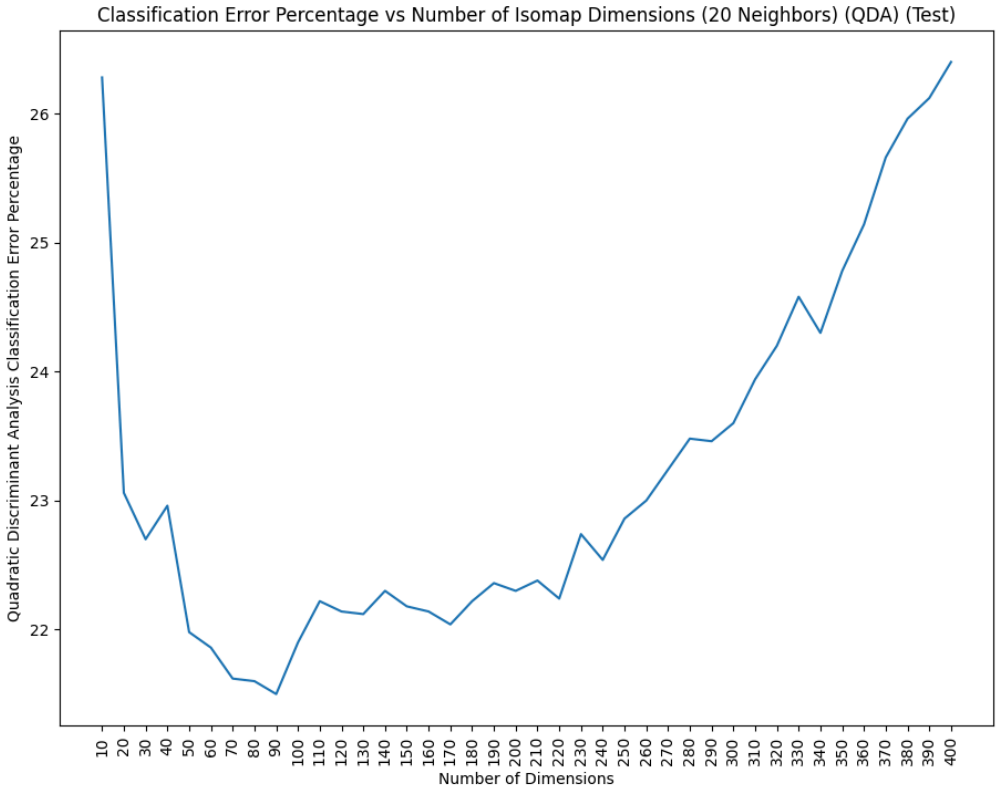

Classification Error Percentage vs Number of Isomap Dimensions (5 Neighbors) (QDA) (Test)
Quadratic Discriminant Analysis Classification Error Percentage
22
23
24
25
26
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400
Number of Dimensions

Graphs with n_neighbor is 20:




Classification Error Percentage vs Number of Isomap Dimensions (20 Neighbors) (GNB) (Test)
Gaussian Classification Error Percentage
Number of Dimensions


Classification Error Percentage vs Number of Isomap Dimensions (20 Neighbors) (QDA) (Train)
Quadratic Discriminant Analysis Classification Error Percentage
Number of Dimensions



The Isomap's results parallel the pattern we observed with PCA, and the reasoning is the same for the results. For the QDA training dataset, we can see that our error constantly decreases as the number of dimensions selected increases. As we increase the dimensions of the Isomap, our QDA dimensions will retain more information that can represent the training data evaluating the variance between them. Similarly, the pattern for the QDA test dataset graph is the same. The error first goes down until a certain point, then it increases again when we come to the point of overfitting. The error percentages are the lowest at 150 dimensions, with an error percentage of ~22%. We can draw the same conclusions from the GNB training and test dataset as in the PCA results. The GNB test dataset errors are higher than the training dataset, and as GNB considers features independent, both graphs have the same structure decrease increase structure.

In conclusion, the interpretations are very similar to PCA. When we compare PCA and Isomap, both have the potential to represent the data with a reasonably low classification error in similar dimensions. PCA achieves a 24-20% error percentage with ~50 components, whereas Isomap achieves a 26-22% error with 40 and 150 dimensions, meaning both can provide a reasonably good representation of the dataset. However, due to the computational

time complexity, I would not use Isomap or increase its n_neighbours parameter. The core differences between PCA and Isomap may cause this. While PCA is a linear method for dimensionality reduction, Isomap is a non-linear method. Consequently, the similarities that each of these approaches identifies will vary.

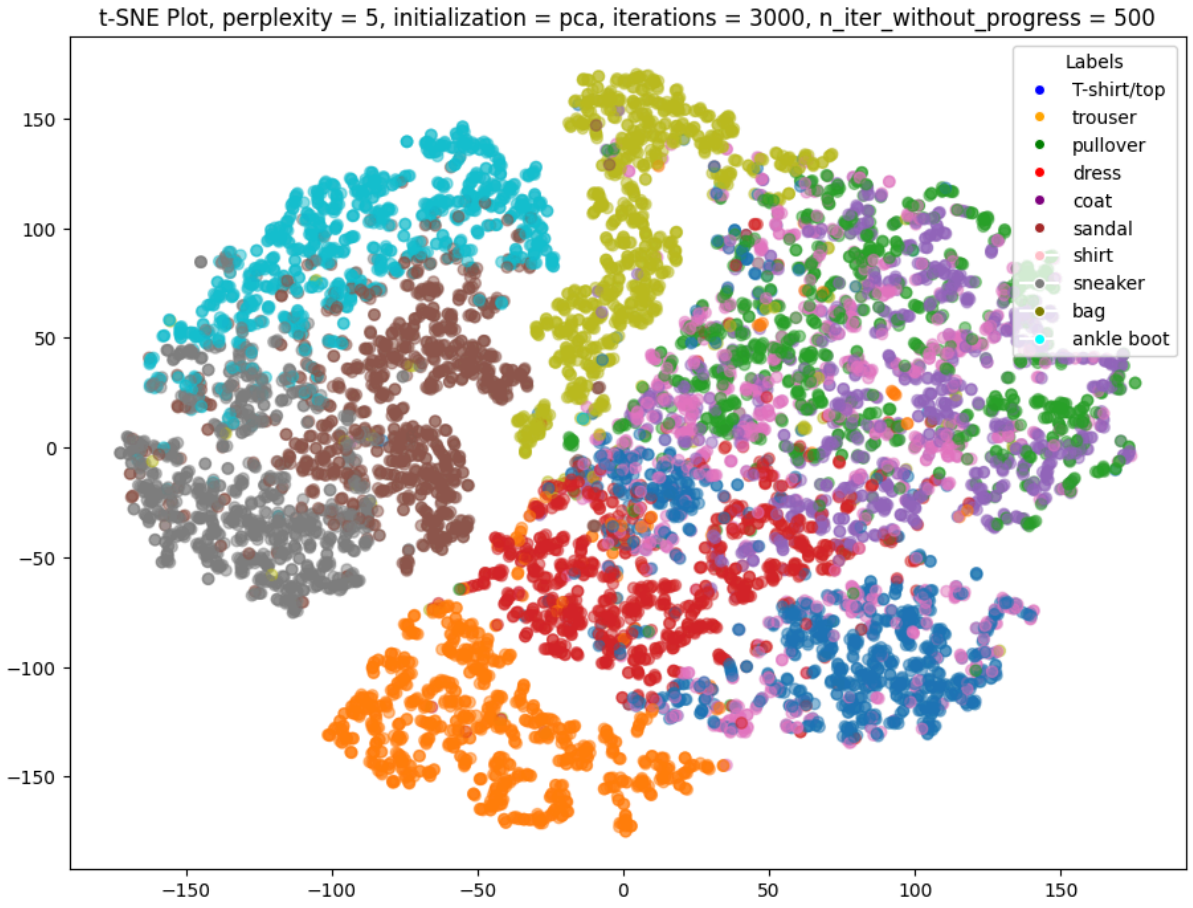
According to our results, we have obtained the following insights. In general, both dimension reduction methods apply to our dataset and can retain important information about samples in smaller dimensions. Our data analysis indicates that both methods achieved their lowest error percentages at approximately 50 dimensions. This suggests that the general patterns in our dataset are adequately represented in that dimensionality. In comparing the models, there are tradeoffs to consider. Isomap requires more computation power than PCA, but PCA achieved a lower error percentage with minimal computational requirements. If our dataset were larger and we could access a faster computer, Isomap could provide even better error percentages, as it captures more complex nonlinear patterns. PCA may be the better choice for our small, simple dataset due to its fast computation time and relatively high accuracy score. However, this choice could change depending on factors mentioned earlier, such as the size and complexity of the dataset and the availability of computational resources.

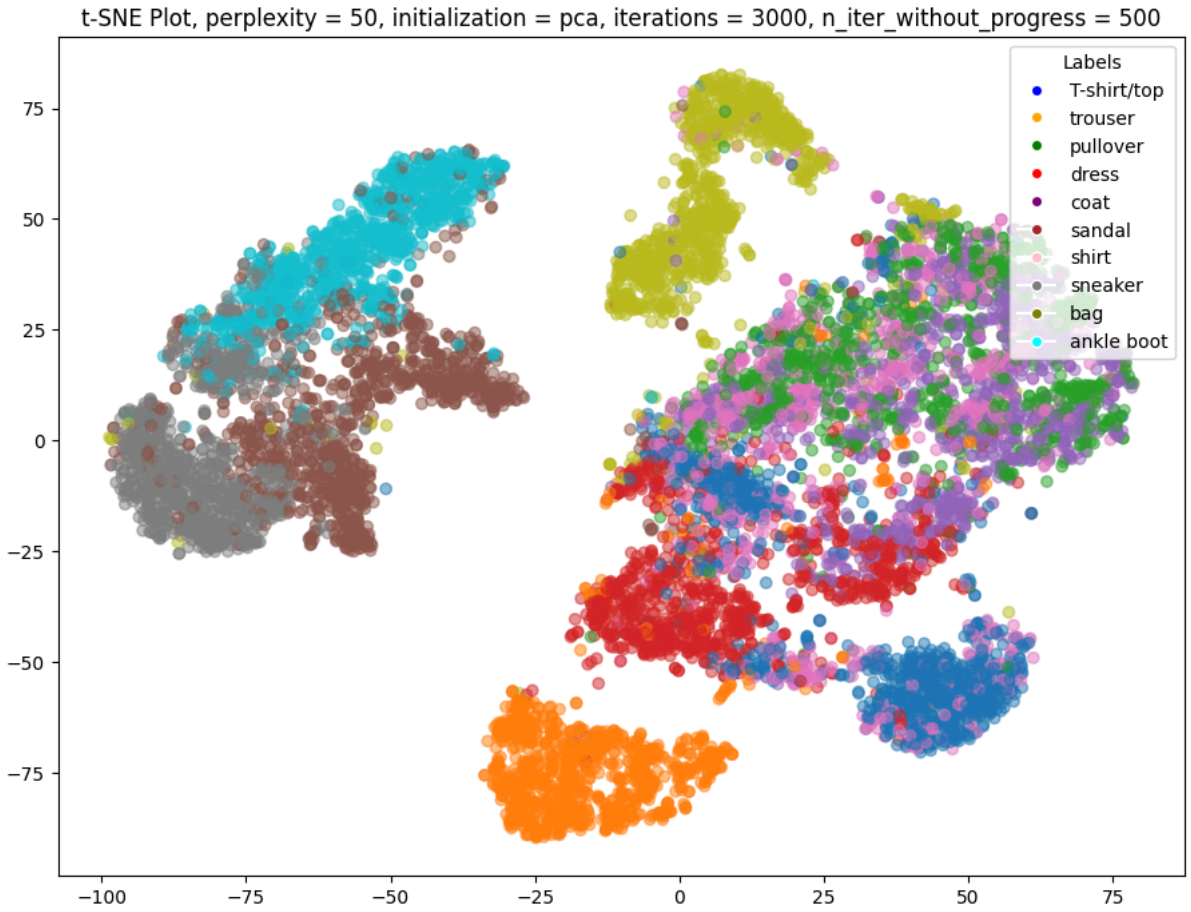
Question 4 t-SNE

I used the t-SNE implementation from the scikit-learn library [1] to map the dataset to two dimensions. To explore the effects of key parameters on t-SNE results, I conducted experiments varying perplexity (10, 30, 50), metric ('cosine,' 'euclidean'), iterations (1000, 3000), early exaggeration (300, 500), and initialization ('random,' 'pca'). I chose these parameters due to their known influence on t-SNE embeddings, balancing this influence with computational limitations. However, other parameters exist, such as learning_rate, angle, method.

To optimize computation time, the method applies the “Barnes-Hut” approximation by default. This approximation efficiently estimates the pairwise distances between data points, resulting in significant time savings without compromising the accuracy. Utilizing this approximation was necessary to reduce the excessive running time encountered without it. Thus, I opted to employ the Barnes-Hut approximation for its effectiveness. I did not include any plots for the method.

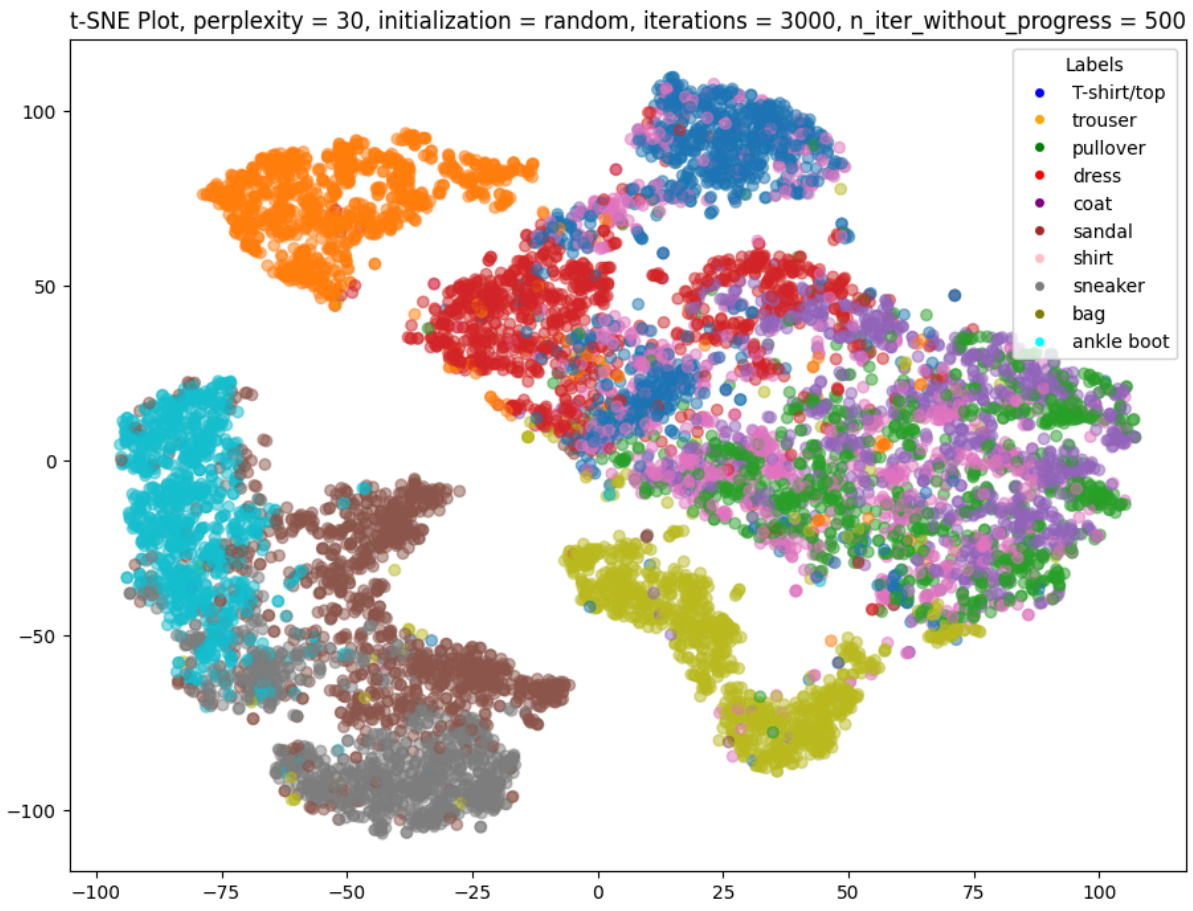
Perplexity is closely linked to the number of neighbors in a t-SNE visualization. A lower perplexity value tends to emphasize local features, while a higher perplexity value captures more global features. The generally recommended perplexity range is between 5 and 50. After experimenting with different perplexity values, I found that the default value of 30 yielded the most interpretable and logical results. Increasing the perplexity to 50 minimizes clusters and makes them closer to themselves. Decreasing the perplexity spreads the clusters.

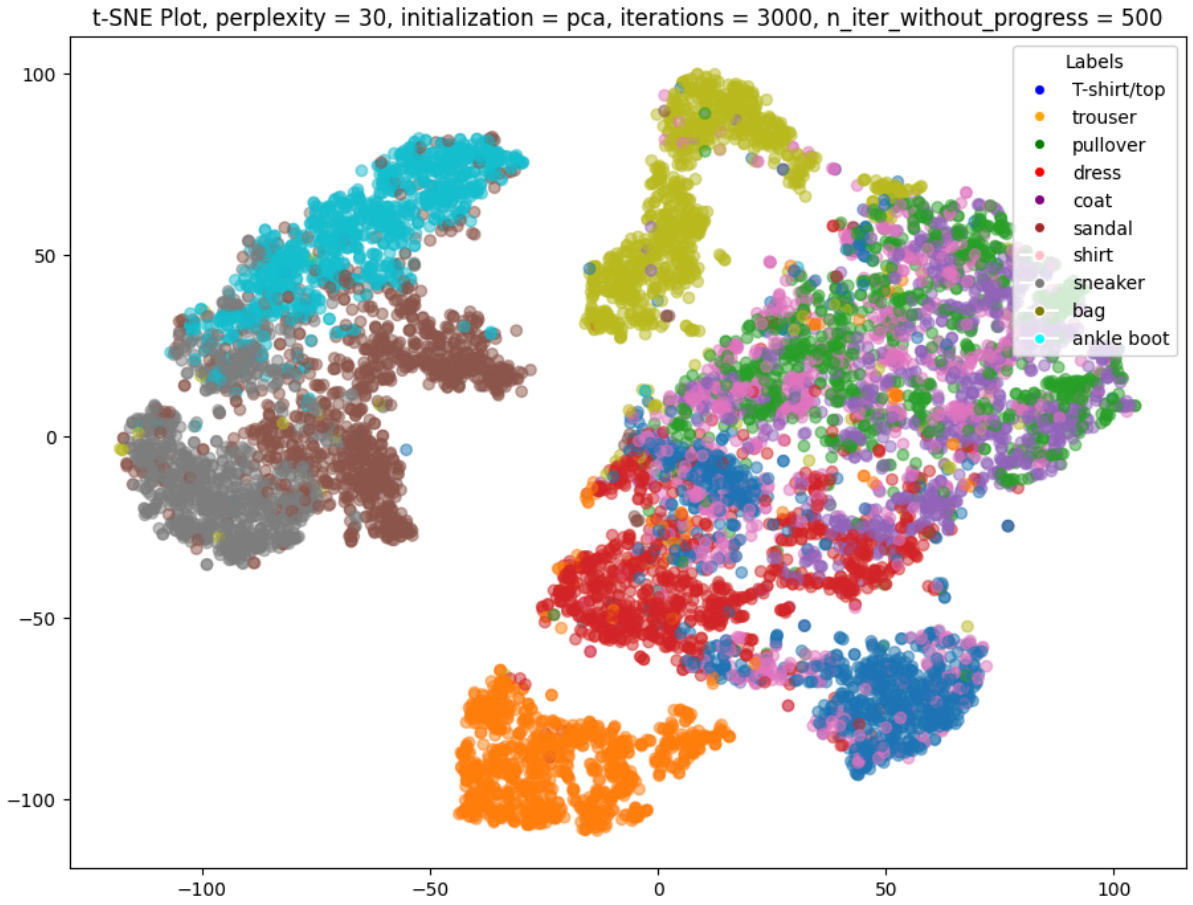




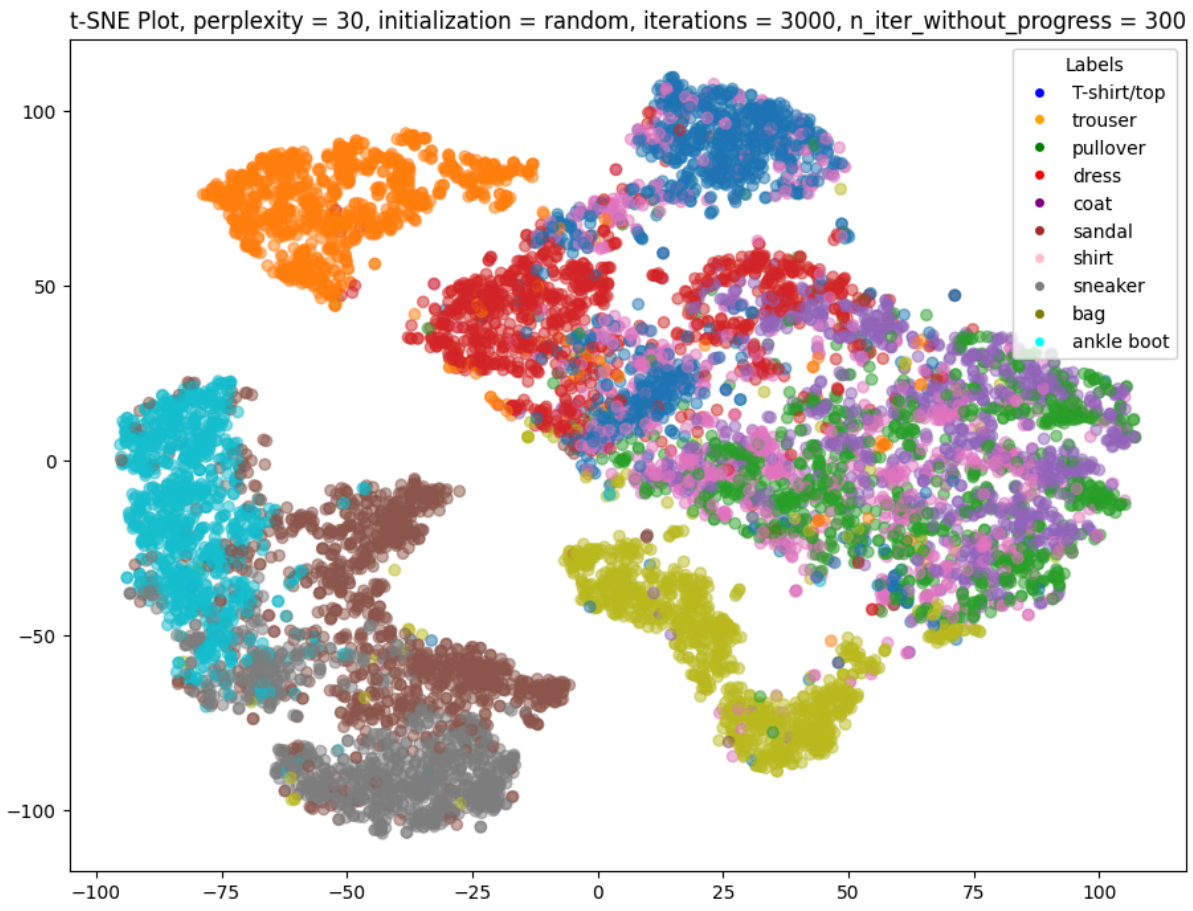
Regarding the learning rate, I have opted for the "auto" parameter selection, which provides an optimal learning rate. I experimented with various rates but discovered that the differences were relatively insignificant. I did not include any plots for the learning rate.

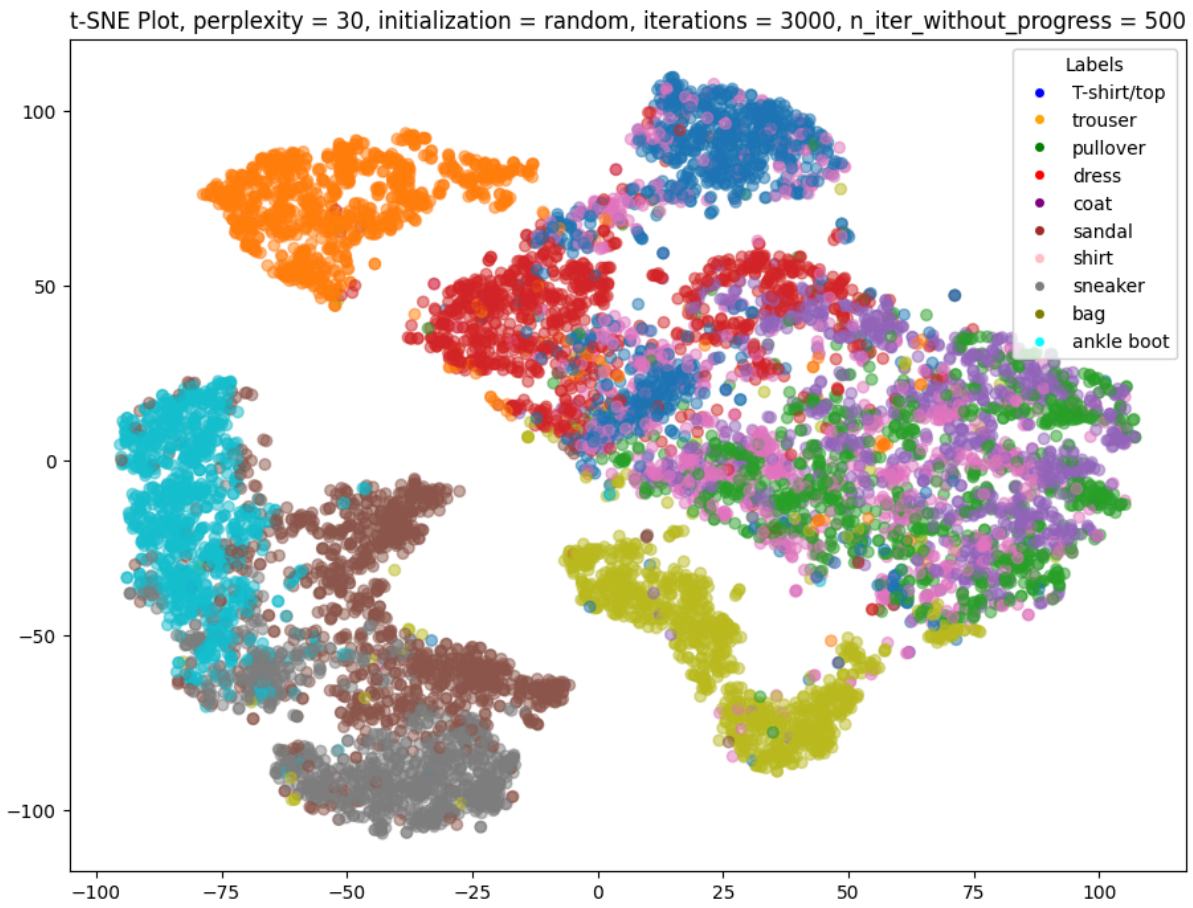
The init parameter governs the generation of the initial low-dimensional data embedding, highly influencing the final t-SNE outcomes. The two primary options are random and Principal Component Analysis (PCA). Random initialization places points randomly in the low-dimensional space, while PCA initialization uses Principal Component Analysis to initialize the embedding. Starting with PCA initialization frequently allows for faster convergence and better overall outcomes than random initialization. We can see the major difference immediately; the clusters' places drastically changed.


t-SNE Plot, perplexity = 30, initialization = random, iterations = 3000, n_iter_without_progress = 500
Labels
T-shirt/top
trouser
pullover
dress
coat
sandal
shirt
sneaker
bag
ankle boot
100
50
0
−50
−100
−100
−75
−50
−25
0
25
50
75
100

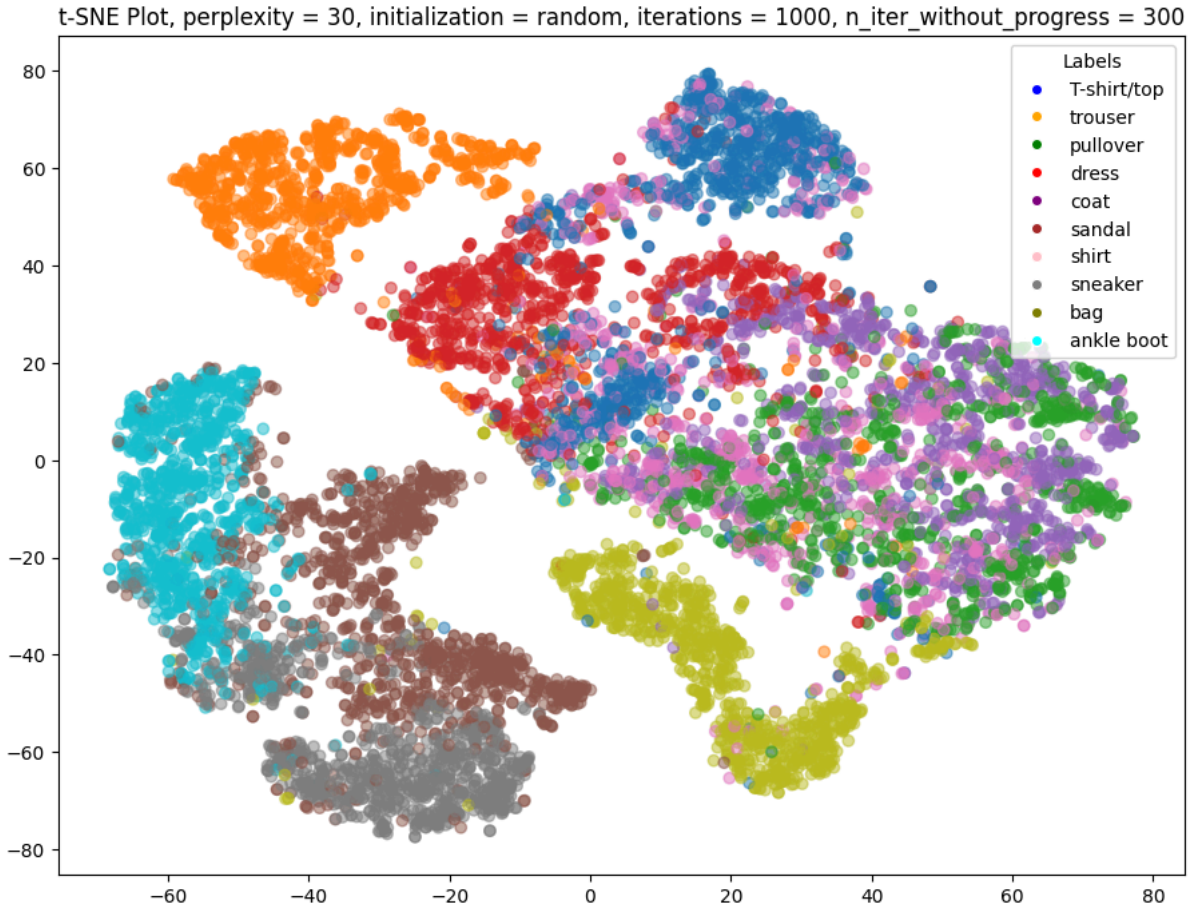


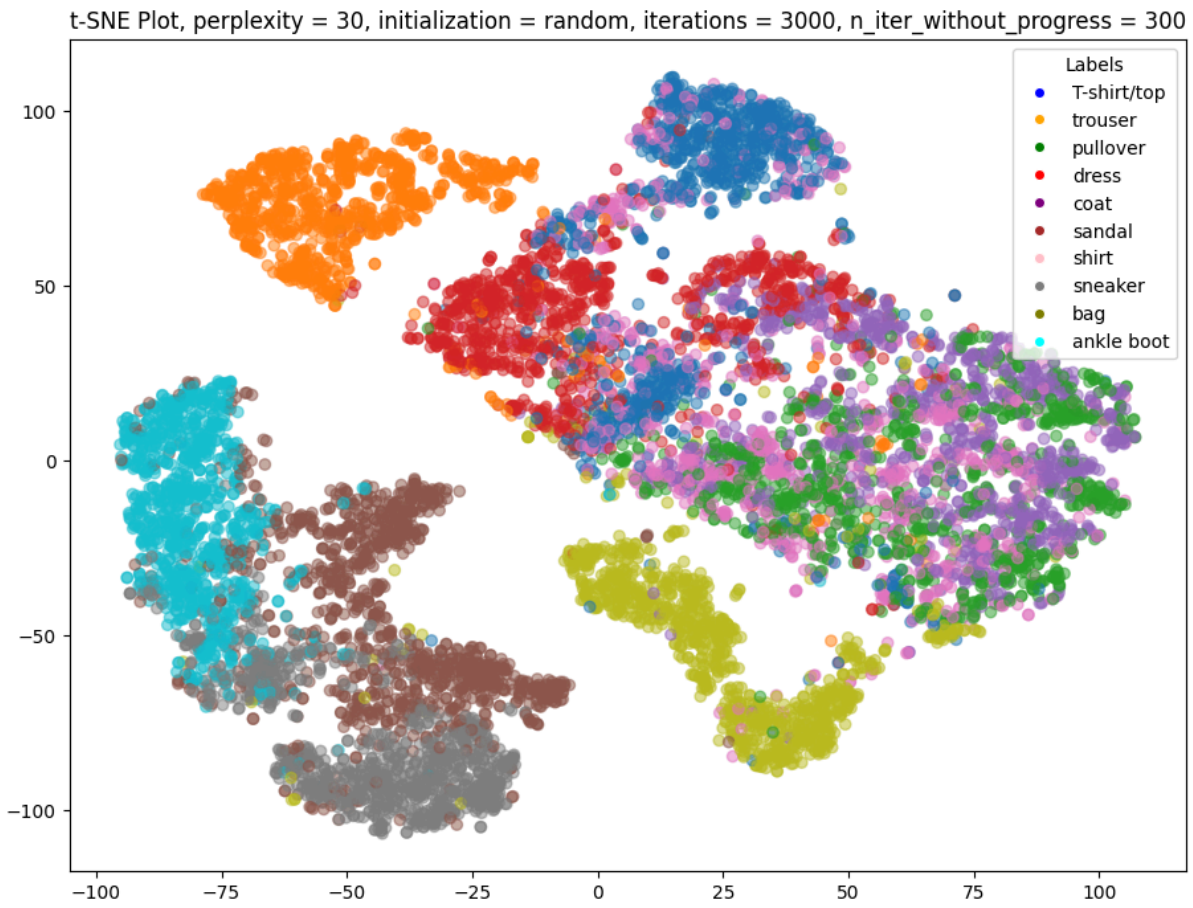
The maximum number of iterations without progress before we abort the optimization is the iterations without progress. I have tried 300 (default) and 500 to test with the ending condition. However, there were no significant changes.


t-SNE Plot, perplexity = 30, initialization = random, iterations = 3000, n_iter_without_progress = 300
Labels
T-shirt/top
trouser
pullover
dress
coat
sandal
shirt
sneaker
bag
ankle boot
100
50
0
−50
−100
−100
−75
−50
−25
0
25
50
75
100



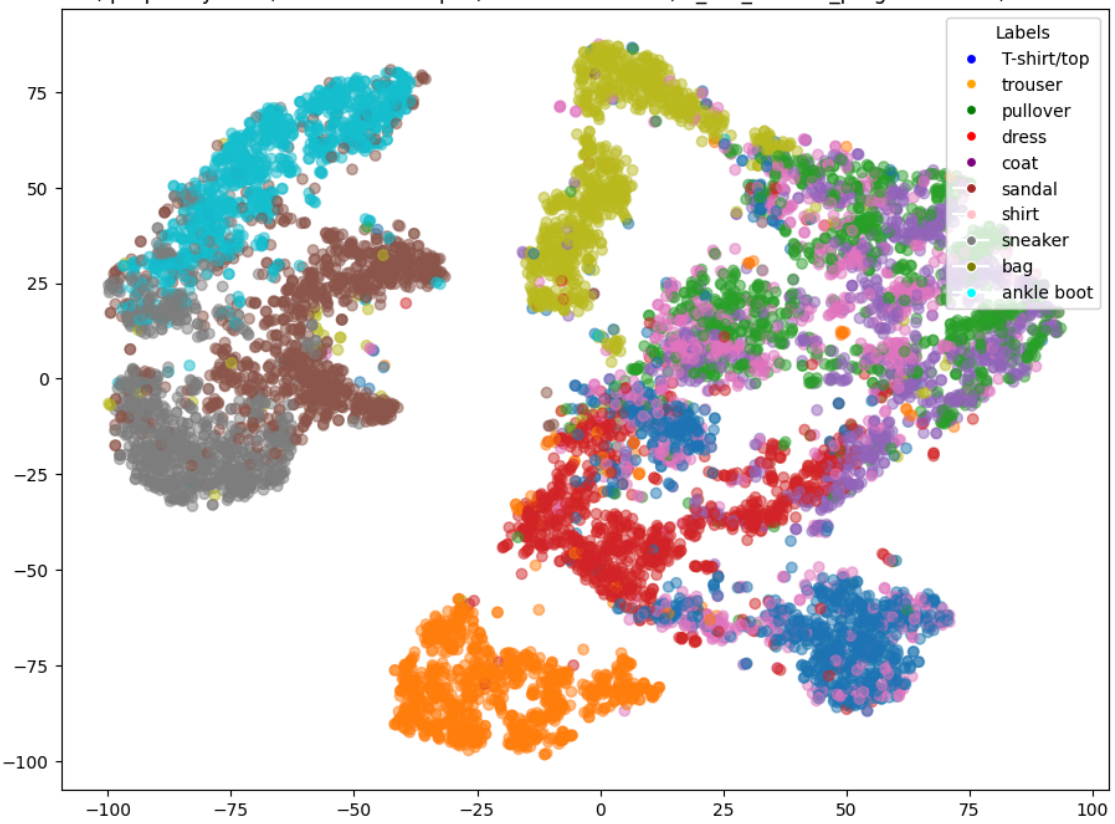
The n_iters is the maximum number of iterations for the optimization. The default value is 1000. When I increased it to 3000, the execution took longer, and the cluster distances increased.


t-SNE Plot, perplexity = 30, initialization = random, iterations = 1000, n_iter_without_progress = 300
Labels
T-shirt/top
trouser
pullover
dress
coat
sandal
shirt
sneaker
bag
ankle boot
80
60
40
20
0
−20
−40
−60
−80
−60
−40
−20
0
20
40
60
80

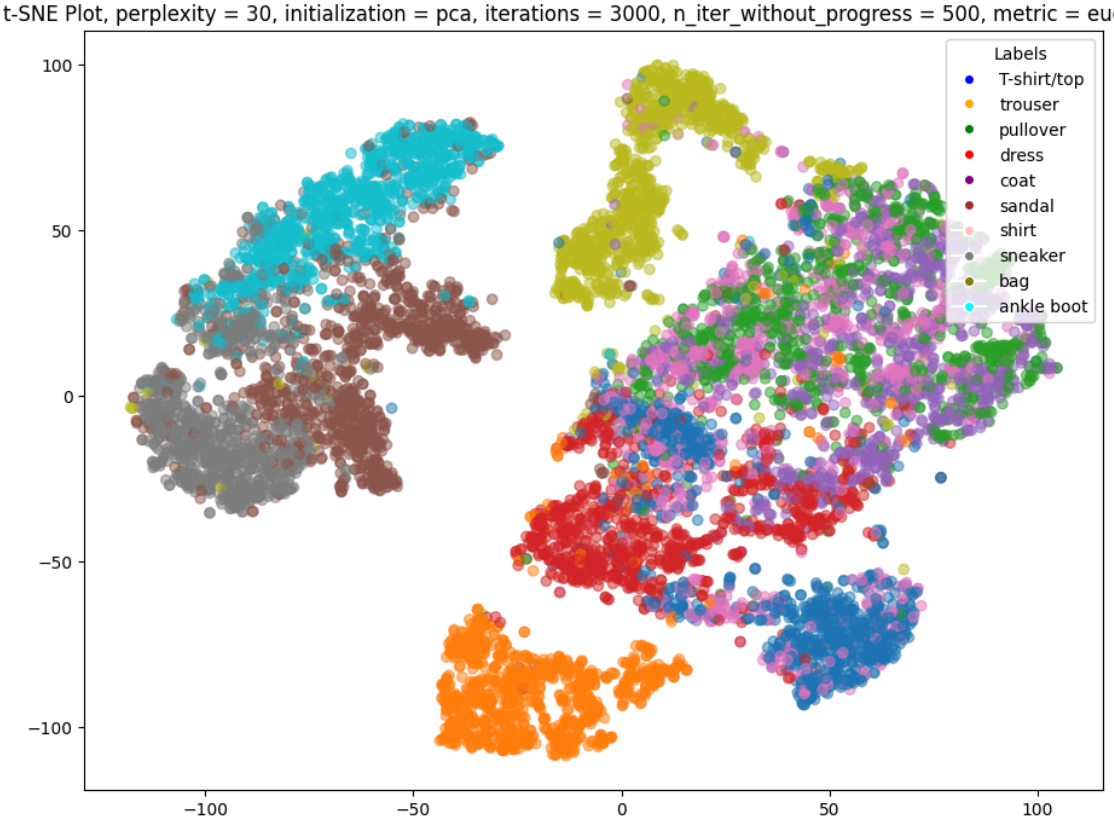


The metric parameter allows us to utilize various distance calculation methods, including Euclidean (default) and Cosine, to form the clusters. I have tried them. One can realize that with Cosine Distance, the boundaries of different clusters are not as near to each other as in the case of Euclidian Distance. Therefore, the clusters are better separated from other clusters around them.

t-SNE Plot, perplexity = 30, initialization = pca, iterations = 3000, n_iter_without_progress = 500, metric = cosine



t-SNE Plot, perplexity = 30, initialization = pca, iterations = 3000, n_iter_without_progress = 500, metric = euclidean



In the t-SNE visualization, distinct clusters were formed for all samples. Upon closer examination of the image, some samples, like trousers and bags, exhibited well-separated clusters that did not overlap significantly with other clusters. This distinction may be attributed to the unique shape of the trousers, which sets them apart from other samples and minimizes mixing. Moreover, certain samples are located very close, and their boundary points tend to overlap. For instance, sneakers, ankle boots, and sandals exhibit this phenomenon. While they have distinct clusters, some points lie on the borderline. This might be because these footwear items share similar shapes, leading to potential confusion in their classification. Notably, there are two distinct clusters for the dress. While other digits exhibit minor disconnections in their clusters, the dress stands out with a separated cluster and an apparent separation. Considering that t-SNE focuses on local similarities, this may suggest the existence of two distinct dress types. However, evaluating other metrics before drawing such a conclusion is crucial, as this observation alone may not be sufficient to establish the presence of two different dress types.

Tools Used

Python imports are like the following:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.random_projection import GaussianRandomProjection
from sklearn.manifold import Isomap
from sklearn.manifold import TSNE
from sklearn import metrics
```

References

[1] Python Software Foundation. Python v3.x. https://www.python.org/. [Accessed: Apr. 7, 2024].

[1] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," scikit-learn.org, 2024. [Online]. Available: https://scikit-learn.org. [Accessed: Apr. 7, 2024].

[2] N. Oliphant et al., "NumPy," NumPy.org, 2024. [Online]. Available: https://numpy.org/. [Accessed: Apr. 7, 2024].

[3] J. Hunter et al., "Matplotlib," Matplotlib.org, 2024. [Online]. Available: https://matplotlib.org/. [Accessed: Apr. 7, 2024].