Mustafa Cankan Balcı

22101761

7 April 2025

# GE461 Introduction to Data Science

# Project: Dimensionality Reduction and Visualization

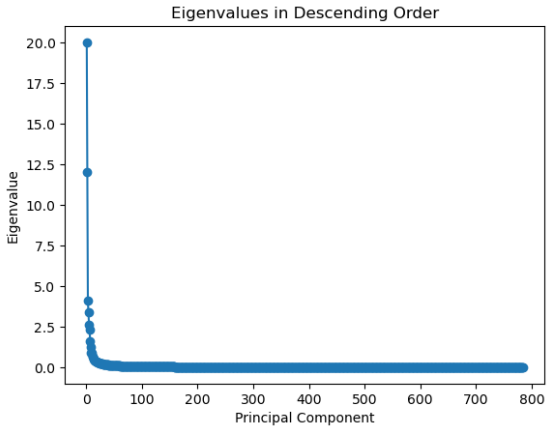
## Introduction

The project evaluates various dimensionality reduction methods—such as PCA and LDA—applied to a classification task using the Fashion-MNIST dataset. Each sample in this dataset is represented by a 784-dimensional vector corresponding to one of 10 categories: 0 represents t-shirts/tops, 1 represents trousers, 2 represents pullovers, 3 represents dresses, 4 represents coats, 5 represents sandals, 6 represents shirts, 7 represents sneakers, 8 represents bags, and 9 represents ankle boots. A subset of 10,000 items is used, containing 1,000 samples per category. This subset is then divided into two groups by randomly selecting half of the samples from each class for training and allocating the remaining half for testing.

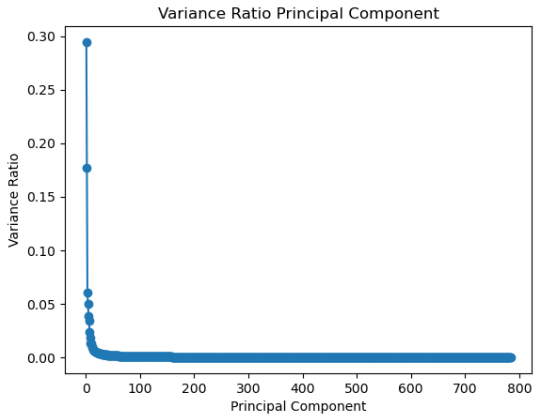
## Question 1

Principal Component Analysis (PCA) is employed to project the high-dimensional 784-dimensional data onto lower-dimensional subspaces [1]. This reduction is essential for visualizing and understanding how the dimensionality of the data impacts the performance of a Gaussian classifier. Essentially, PCA seeks a transformation that captures the maximum variance of the data while minimizing the reconstruction error in a least-squares sense. This is achieved by identifying the principal components—orthogonal directions that correspond to the eigenvectors of the covariance matrix of the data, ordered by their associated eigenvalues.

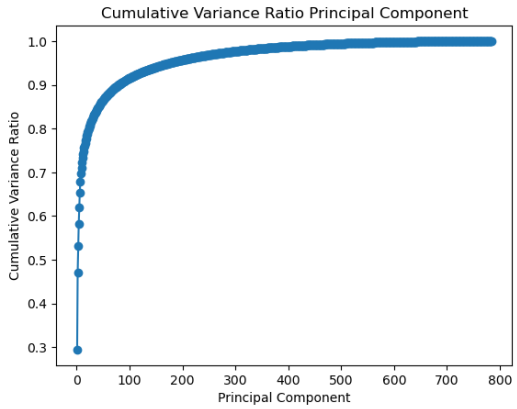
First, the global mean is subtracted from both the training and testing datasets to center the data. Then, PCA is applied to the centered data to reduce its dimensionality. Graph 1 illustrates the PCA results, helping to determine a sustainable number of dimensions. According to the plots, after the first 10 principal components, the eigenvalues become sparse. The Cumulative Variance Ratio confirms that the total explained variance sums to 1.0 (or 100%), as expected given the original 784 dimensions. Additionally, the Eigenvalues in Descending Order plot shows the magnitude of each eigenvalue, while the Variance Explained by Each Principal Component plot indicates the corresponding ratio for each component.




**Graph 1:** Eigenvalues in Descending Order




**Graph 2:** Variance Ratio Principal Component




**Graph 3:** Cumulative Variance Ratio Principal Component

The sample mean for the whole training dataset is figure 1. Also, the eigenvalues found from PCA is shown in figure 2. According to my expectations, the principal components will get the edges and features of the dataset. Figure 2 is a combination of all classifications.

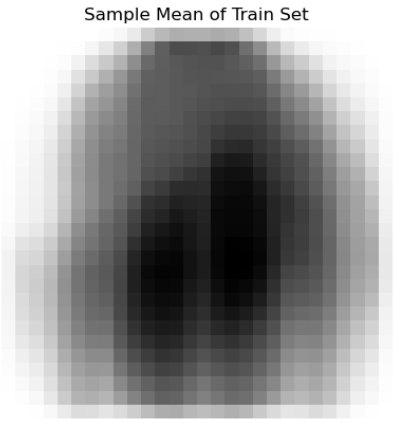



**Fig. 1:** Sample Mean of Train Set

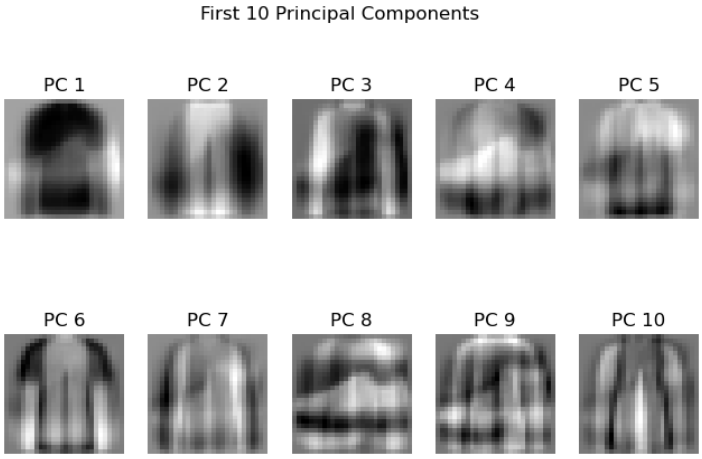
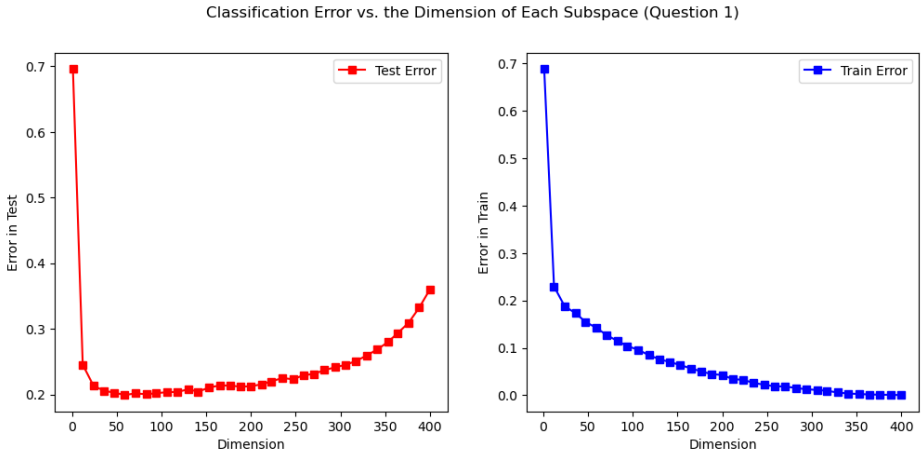



**Fig. 2:** First 10 Principal Components

Furthermore, 35 different subspaces with dimensions between 1 and 400 is selected. According to dimension, the data was projected by using the transformation matrix estimated from the training data onto these subspaces. Finally, Gaussian classifier is trained using the data. QuadraticDiscriminantAnalysis is used for Gaussian classifier at the project [2].

Graph 4 displays the classification error as a function of the number of components used for each subspace. The training error consistently decreases as more dimensions are added, indicating that the model is progressively learning additional features. However, after reaching approximately 150 dimensions, the test error begins to rise, which suggests that the model is starting to overfit. This outcome aligns with expectations, as the curse of dimensionality typically degrades model performance in higher-dimensional space.




**Graph 4:** Classification Error vs. Each Subspace Dimension

## Question 2

In this section, Fisher's Linear Discriminant Analysis (LDA) is applied as a dimensionality reduction technique to project the original 784-dimensional data onto a lower-dimensional subspace [3]. LDA's objective is to find a transformation that maximizes the separation between classes while minimizing the variability within each class, thereby providing an optimal representation of the data for classification purposes.

Figure 3 depicts the LDA result, which consists of nine components. These components are the result of a supervised algorithm that utilizes class labels to determine which directions in the data best split the various classes. Unlike PCA, which emphasizes variation and frequently exposes structural patterns, LDA components might look uniform or abstract. Each component, however, plays an important role in optimizing class separability, allowing the model to discriminate between distinct categories (for example, digits 0-9). The tiny changes in grayscale intensity across components show how LDA encodes discriminative information.

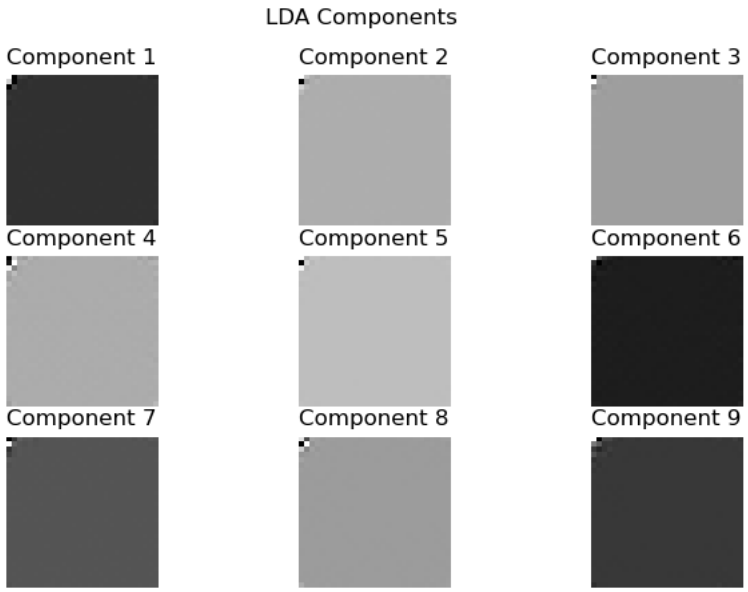
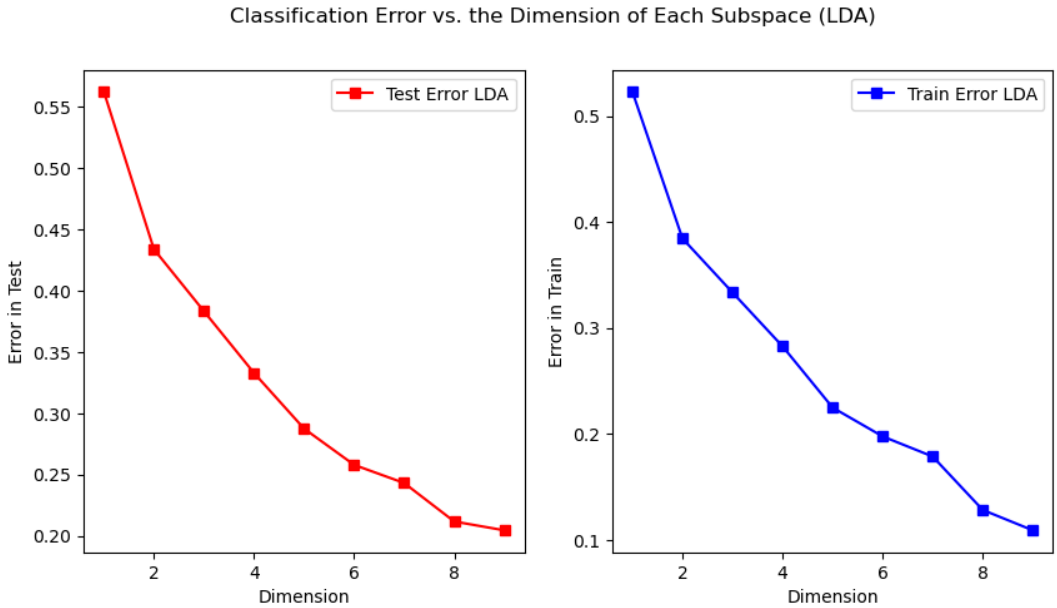



**Fig. 3:** LDA Components

Both the training and testing datasets are projected onto subspaces with dimensions ranging from 1 to 9, which is the maximum achievable using LDA for a dataset containing 10 classes. Afterwards, a Gaussian classifier is trained on the transformed data using Quadratic Discriminant Analysis.

The graphs illustrate the classification error plotted against the number of components used for each subspace. Initially, with only one dimension, the error rate is high. As more dimensions are added, both training and test errors gradually decrease. When the maximum LDA dimension is reached, the model is considered optimal. In contrast to PCA, LDA achieves this optimal point with fewer dimensions and at a faster rate.



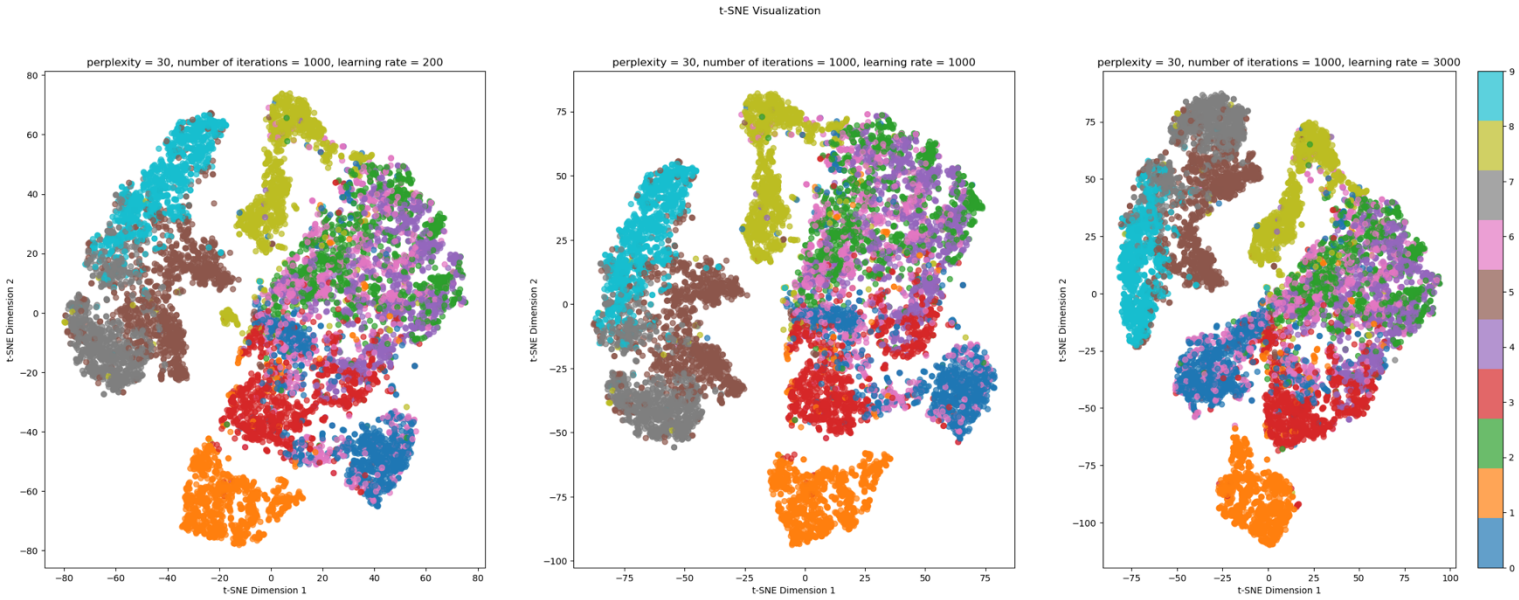

**Graph 5:** Classification Error vs. Each Subspace Dimension (LDA)

## Question 3

t-SNE is a nonlinear dimensionality reduction technique that preserves the local structure of high-dimensional data by mapping it to a lower-dimensional space (typically 2D or 3D) [4]. It converts pairwise distances into conditional probabilities that capture similarities, then finds a mapping that minimizes the difference between these probabilities in both spaces. This approach often highlights clusters corresponding to different classes, even if the overall global structure is less clear.

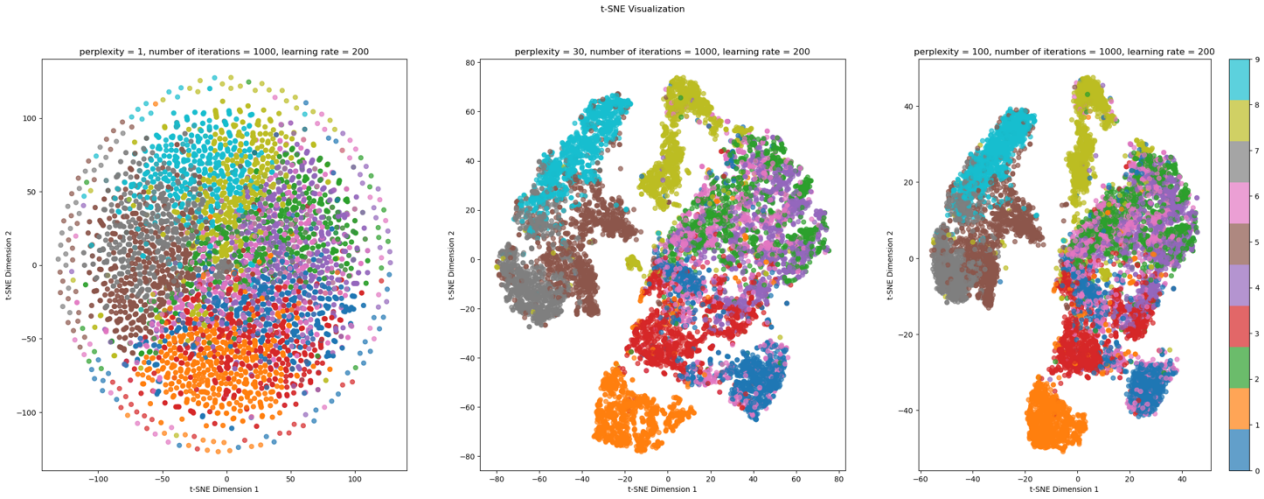
In the experimental setup, several key parameters significantly influence the output. The objective is to visualize the data in two dimensions, so we set n_components to 2. Perplexity serves as a smooth measure of the effective number of neighbors, while the learning rate governs the speed of convergence. Additionally, the number of iterations is chosen to be sufficiently high to ensure the mapping converges. PCA is also used for initialization, which helps provide a more stable embedding and leads to more interpretable visualizations. As a result, 3 paratemeters are investigated in question 3: perplexity, number of iterations and learning rate.

Graph 6 below illustrate the effects of varying learning rates on training. At rates of 300 and 1000, performance remains consistent with no significant differences observed. However, increasing the rate to 3000 causes a noticeable shift in the diagram, indicating a substantial change in training dynamics. This change may suggest instability or overshooting optimal parameters, emphasizing the need to choose a proper learning rate for reliable model performance.



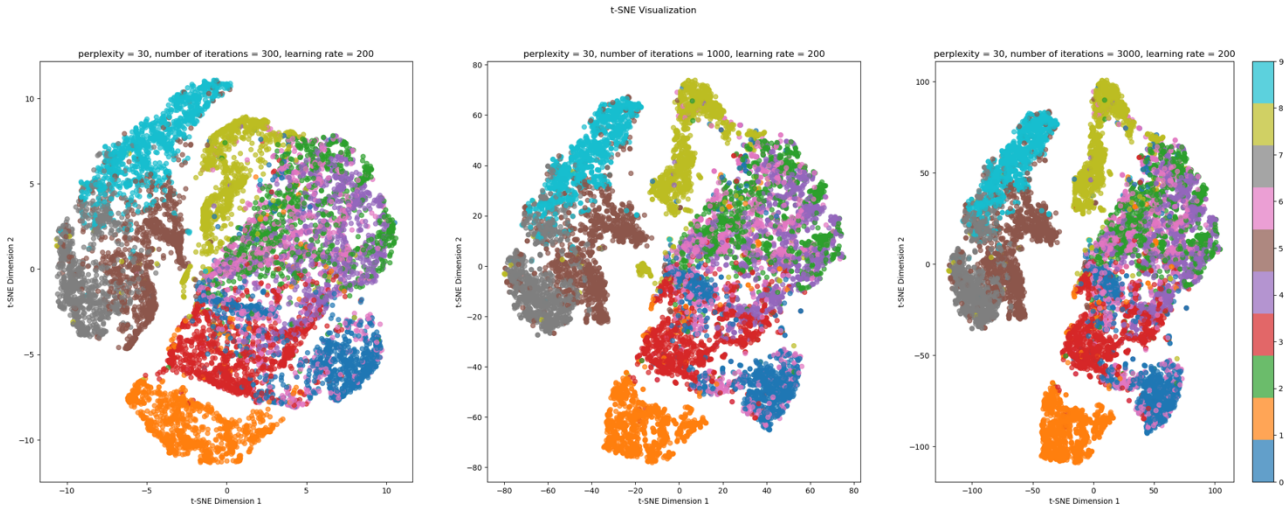

**Graph 6:** Plots of different learning rate

Graph 7 shows how t-SNE outcomes depend on perplexity. The left panel depicts the two concentric circles dataset, which shows that higher perplexity recovers a circular topology; however, circle sizes and distances deviate from the original. In contrast, the right panel, which displays the S-curve dataset, shows that even higher perplexity values do not preserve the original S-curve structure. Instead, the shapes differ significantly from the original curve. These findings highlight that t-SNE visualizations can vary in cluster size, distance, and shape, and that parameter selection is critical for accurately and clearly reflecting the underlying data geometry.




**Graph 7:** Plots of different learning rate

Graph 8 demonstrates the impact of varying the number of iterations on t-SNE visualizations. As the number of iterations increases from 300 to 3000, the clusters in the dataset appear to become more compact and closer to each other. This excessive closeness may lead to overfitting in machine learning models, as it can distort the representation of the data's actual structure and exaggerate the similarity between different clusters, reducing model generalizability.




**Graph 8:** Plots of number of iterations

## References

[1] "PCA," scikit-learn: Machine Learning in Python, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (accessed Apr. 7, 2025).

[2] "QuadraticDiscriminantAnalysis," scikit-learn: Machine Learning in Python, https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html (accessed Apr. 7, 2025).

[3] "LinearDiscriminantAnalysis," scikit-learn: Machine Learning in Python, https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html (accessed Apr. 7, 2025).

[4] "TSNE," scikit-learn: Machine Learning in Python, https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html (accessed Apr. 7, 2025).

## Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.manifold import TSNE
from tqdm import tqdm
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from tqdm import tqdm
from sklearn.naive_bayes import GaussianNB

np.random.seed(22101761)
labels_data = np.loadtxt('fashion_mnist/fashion_mnist_labels.txt',dtype=int)
print(labels_data[0])
fashion_images = np.loadtxt('fashion_mnist/fashion_mnist_data.txt')

fashion_images_reshaped = np.reshape(fashion_images, (10000,28,28))

plt.imshow(fashion_images[0].reshape((28,28)).T,cmap='gray')
plt.title('Image from dataset')
plt.axis("off")
n_samples = fashion_images.shape[0]
d = fashion_images.shape[1]
train_idx = []
test_idx = []
classes = np.unique(labels_data)
for cls in classes:
    indices = np.where(labels_data == cls)[0]
    np.random.shuffle(indices)
    split = len(indices) // 2
    train_idx.extend(indices[:split])
    test_idx.extend(indices[split:])
np.random.shuffle(train_idx)
np.random.shuffle(test_idx)

train_set_x = fashion_images[train_idx]
train_set_y = labels_data[train_idx]


test_set_x = fashion_images[test_idx]
test_set_y = labels_data[test_idx]

general_mean = np.mean(fashion_images,axis=0)

centered_train_set_x = train_set_x - general_mean
centered_test_set_x = test_set_x - general_mean
```

```
hold = centered_train_set_x.shape[1]
pca = PCA(n_components=hold)
pca.fit(centered_train_set_x)
eigenvalues_pca = pca.explained_variance_


plt.figure()
plt.plot(range(1,785), eigenvalues_pca,marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.title('Eigenvalues in Descending Order')

pca_variance  = pca.explained_variance_ratio_


plt.figure()
plt.plot(range(1,785), pca_variance,marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Variance Ratio')
plt.title('Variance Ratio Principal Component')
plt.show()

plt.figure()
plt.plot(range(1,785), np.cumsum(pca.explained_variance_ratio_),marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Cumulative Variance Ratio')
plt.title('Cumulative Variance Ratio Principal Component')
plt.show()

global_mean_train = np.mean(train_set_x,axis=0)

plt.figure()
plt.imshow(global_mean_train.reshape((28,28)).T,cmap='gray_r')
plt.title('Sample Mean of Train Set')
plt.axis("off")
plt.show()
eigenvectors = pca.components_[:10]
fig, axes = plt.subplots(2, 5, figsize=(8, 5))
for i, ax in enumerate(axes.flatten()):
        ax.imshow(eigenvectors[i].reshape(28, 28).T, cmap='gray_r')
        ax.set_title(f'PC {i+1}')
        ax.axis('off')
plt.suptitle('First 10 Principal Components')
plt.show()
dims = np.linspace(1, 400, num=35, dtype=int)

models = []
```

```
error_test = []
error_train = []

for dim in tqdm(dims):
    pca = PCA(n_components=dim)
    X_train_pca = pca.fit_transform(centered_train_set_x)
    X_test_pca = pca.transform(centered_test_set_x)
    model = QuadraticDiscriminantAnalysis().fit(X_train_pca, train_set_y)

    score_test =  1 - model.score(X_test_pca,test_set_y)
    error_test.append(score_test)

    score_train =  1 - model.score(X_train_pca,train_set_y)
    error_train.append(score_train)

    models.append(model)
    print(f"PCA Dim: {dim}, Test Error: {score_test:.4f} Train Error:
{score_train:.4f}")

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))


# Plot for Test Error with red color
axes[0].plot(dims, error_test, marker='s', color='red', label='Test Error')
axes[0].set_xlabel('Dimension')
axes[0].set_ylabel('Error in Test')
axes[0].legend()

# Plot for Train Error with blue color
axes[1].plot(dims, error_train, marker='s', color='blue', label='Train Error')
axes[1].set_xlabel('Dimension')
axes[1].set_ylabel('Error in Train')
axes[1].legend()

plt.suptitle('Classification Error vs. the Dimension of Each Subspace (Question 1)')
plt.show()

clf = LinearDiscriminantAnalysis(n_components=8)
clf.fit_transform(train_set_x, train_set_y)

      # Plot each LDA basis as an image.


# Plot each LDA basis as an image.

lda_components = clf.scalings_[:,:9]

fig, axes = plt.subplots(3, 3, figsize=(8, 5))
```

```
for i, ax in enumerate(axes.flatten()):
        ax.imshow(lda_components[:, i].reshape(28, 28).T, cmap='gray_r')
        ax.set_title(f'Component {i+1}')
        ax.axis('off')

plt.suptitle('LDA Components')
dims_list = np.arange(1,10)

model_lda = []

error_test_lda = []
error_train_lda = []
for dim in dims_list:
    clf = LinearDiscriminantAnalysis(n_components=dim)
    X_train_lda = clf.fit_transform(train_set_x, train_set_y)
    X_test_lda = clf.transform(test_set_x)
    model = QuadraticDiscriminantAnalysis().fit(X_train_lda, train_set_y)

    score_test =  1 - model.score(X_test_lda,test_set_y)
    error_test_lda.append(score_test)

    score_train =  1 - model.score(X_train_lda,train_set_y)
    error_train_lda.append(score_train)

    model_lda.append(model)


    print(f"LDA Dim: {dim}, Test Error: {score_test:.4f}, Train Error:
{score_train:.4f}")

fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Plot for Test Error with red color
axes[0].plot(dims_list, error_test_lda, marker='s', color='red', label='Test Error
LDA')
axes[0].set_xlabel('Dimension')
axes[0].set_ylabel('Error in Test')
axes[0].legend()

# Plot for Train Error with blue color
axes[1].plot(dims_list, error_train_lda, marker='s', color='blue', label='Train Error
LDA')
axes[1].set_xlabel('Dimension')
axes[1].set_ylabel('Error in Train')
axes[1].legend()

plt.suptitle('Classification Error vs. the Dimension of Each Subspace (LDA)')
plt.show()
```

```
def question_3(perplexity,n_iter,learning_rate,axs):

    tsne = TSNE(n_components=2, init='pca', random_state=22101761,
perplexity=perplexity, n_iter=n_iter,learning_rate=learning_rate)
    X_tsne = tsne.fit_transform(fashion_images)


    scatter = axs.scatter(X_tsne[:, 0], X_tsne[:, 1], c=labels_data, cmap='tab10',
alpha=0.7)
    axs.set_xlabel('t-SNE Dimension 1')
    axs.set_ylabel('t-SNE Dimension 2')
    axs.set_title(f' perplexity = {perplexity}, number of iterations = {n_iter},
learning rate = {learning_rate} ')
    return scatter

lr = [200,1000,3000]
fig, axes = plt.subplots(1, 3, figsize=(30, 10))

perplexity = 30
n_iter = 1000

for i in range(len(lr)):
    scatter = question_3(perplexity,n_iter,lr[i],axes[i])

plt.suptitle('t-SNE Visualization')
plt.colorbar(scatter, ticks=np.arange(10))

perplexity = [1,30,100]
fig, axes = plt.subplots(1, 3, figsize=(30, 10))

lr = 200
n_iter = 1000

for i in range(len(perplexity)):
    scatter = question_3(perplexity[i],n_iter,lr,axes[i])

plt.suptitle('t-SNE Visualization')
plt.colorbar(scatter, ticks=np.arange(10))
n_iter = [300,1000,3000]
fig, axes = plt.subplots(1, 3, figsize=(30, 10))

lr = 200
perplexity = 30

for i in range(len(n_iter)):
    scatter = question_3(perplexity,n_iter[i],lr,axes[i])

plt.suptitle('t-SNE Visualization')
```

```
plt.colorbar(scatter, ticks=np.arange(10))
```