Eray Gündoğdu 22101962

# GE 461 Spring 2024/25 Project 2

## Preprocessing the Dataset

The dataset and corresponding labels are proived as a .txt file. These two files are loaded by using loadtxt and pixel values (dataset) is decoded as float32 whereas labels are decoded as int32. The design matrix (X) is the 10000x784 matrix which is the content of the fashion_mnist_data.txt file whereas true labels (y) are the content of the fashion_mnist_labels.txt file. After loading the dataset the shapes of both X and y are verified. It is desired to seperate the dataset in half to training and test processes. After assigning 22101962 to the random seed, the train and test indices are generated and each class is seperated from each other which results as 1000x1 column vectors. The contents (indices) of each 1000x1 column vector is shuffled to achieve randomness and then splitted into half to obtain train and test indices. After concatenating these train and test indices and use them to index original X and y the following matrix/vectors are obtained. X_train as the training data (5000x784), y_train (5000x1) as the training labels, X_test test data (5000x784), y_test true labels (5000x1) for performance measure.

## Used Libraries and Implementation Tools

Throughout this assignment sklearn library is used to utilize the desired models other than numpy and matplotlib. The PCA model is extracted from sklearn.decomposition/PCA [2]. LDA is extracted form sklearn.discriminant_analysis/LinearDiscriminantAnalysis [3]. As the last dimensionality reduction model t-SNE is extracted from sklearn.manifold/TSNE [4].

In quadratic gaussian classifier the purpose is to fit multivariate gaussian models to classes with MLE of covariance and mean. To meet these requirements sklearn.discriminant_analysis/QuadraticDiscriminantAnalysis is used. The link [1] explicitly demonstrates the likelihood function in the assigment and further documentation states that classification is achieved by fitting gaussian density. Also the model provides estimates of full covarience matrix (not merely diagonal estimate) hence becomes more accurate for cases wich include non-zero correlation.

## Question 1

1) It is desired to extract the mean of the whole data from each sample hence with np.mean operation mean of axis 0 is obtained and then substracted form train and test values.

2)

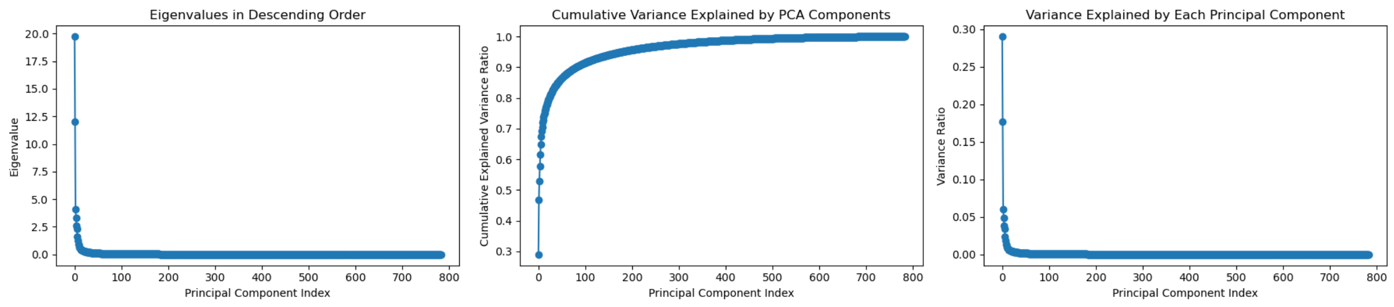


Figure 1 PCA plots for eigenvalue and variance observations

In PCA main goal is to maximize the variance that explains the data in a lower dimensional space. In Figure 1 the methadology can be justified by the Cumulative Variance Explained by PCA Components figure which indicates that the sum of proportions of explained variance is equal to 1.0 (%100) just as expected since the dataset lives in 784 dimensionlaity space and each principle components builds up to explain the variance in data %100 percent. It is also known if the eigenvalue turns out to be larger the better proportion of variance is explained. Eigenvalues in Descending Order indicates the value of the eigenvalue and Variance Explianed by Each Principle Component demonstrates the corresponding ratio. As it can be observed the values of eigenvalues drop drastically therefore choosing the first five components will explain approximately %60 (0.60) of the variance.

3) Before making the observations it is benefitial to mention that taking the mean is a low pass filtering operation therefore it is expected to recieve an image that each pixel roughly depicts the corresponding pixel for all samples. For the eigenvectors, it is expected to have a relation between the original data since they form the basis of the projection space and projection space is able to explain %50 of the variance. As a result, the eigenvector images expected to have a patterns from the original dataset and since the larger egienvalue explains more variance its corresponding eigenvector is expected to have more menaingful patterns. In Figure 2 the mean as an image can be observed. As it is expected due to low pass filtering, each pixel rougly explains the original pixels for each sample. In Figure 3 previos expectations can be verified since the figures demonstrate some kind of patterns.

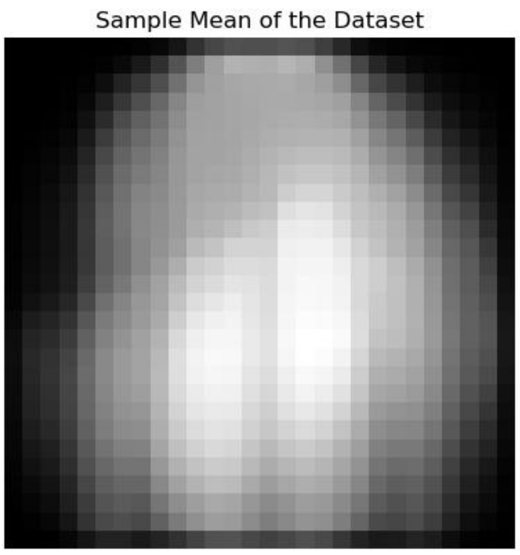


Figure 2 Sample mean of the dataset

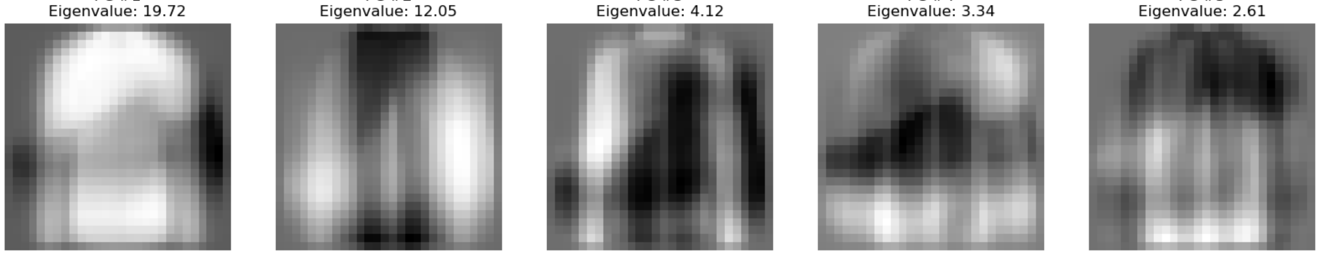
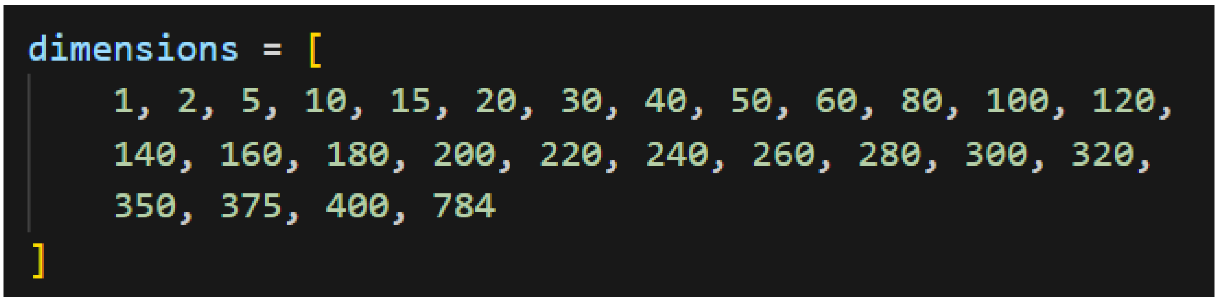


Figure 3 Reshaped eigenvectors

4) Now 25 different dimesions are determined to obsereve the influence of dimensionality reduction on the performance of classification. The dimension list can be observd as following. Note that an additioanl 784 dimensional space added to deepen the discussion which will be eleborated on the following part.



```
dimensions = [
    1, 2, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100, 120,
    140, 160, 180, 200, 220, 240, 260, 280, 300, 320,
    350, 375, 400, 784
]
```

Figure 4 Illustration of dimensions

5) In the implmentation, the multivariate gaussian is fitted to each class for different dimensions. The reson 784 is appended to dimesion list to discuss the overfitting situtation. In the light of previous discussions the PCA provides some kind of basis which are reffered as patterns. Therefore in lower dimensions (after the underfitting threshold) the model is more generalizable since the data used in training of the classifier have basis vectors that represent the patterns however it expected that as the dimensionality increases the PCA becomes more characteristic which means it carries the characteristics of the training set hence the classifier memorizes the training set which results in overfitting. In Figure 5 it can be clearly seen that around 50 dimensions the model performence stops improving which after that point test error starts to increase due to overfitting and up to 50 dimensions the model underfits with high classification error. Hence representing the data in around 50 dimensional space is the optimal method to train the quadratic gaussian classifier.

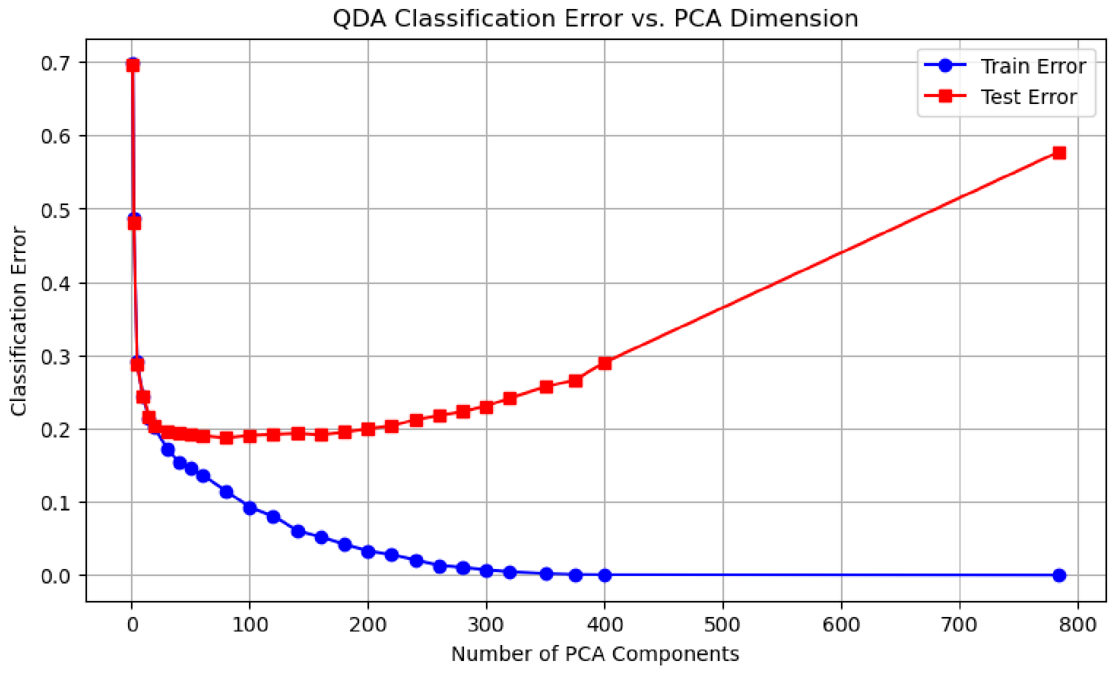


Figure 5 Test classification error and train classification error with PCA applied

**Question 2**

1)Initially LDA model is fit to the datset and to observe whether LDA fit to the dataset .means_ extension results are displayed as images and as a result the LDA can be considered as fitted as Figure 6 is observed.

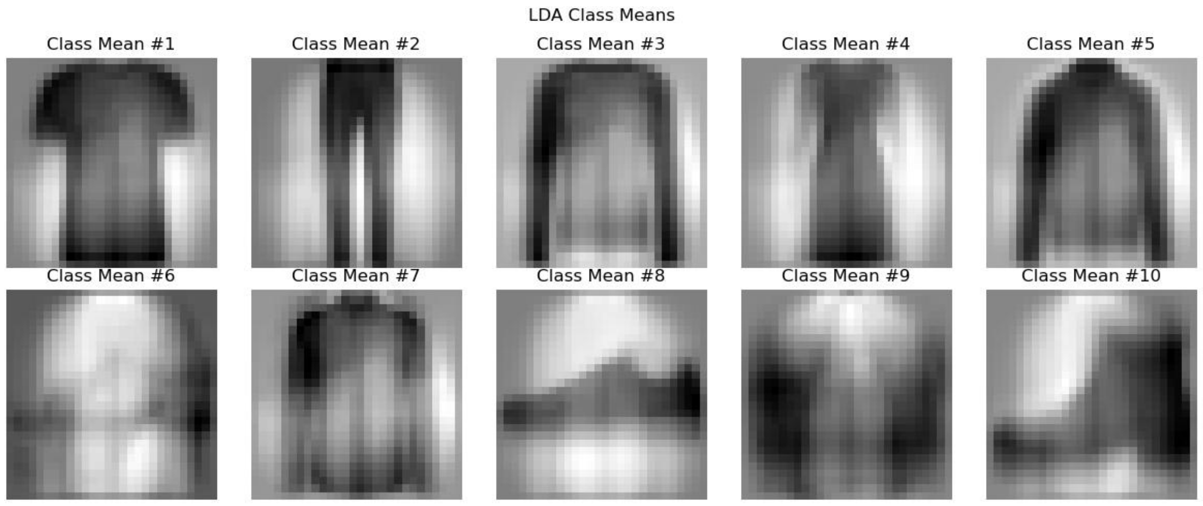


Figure 6 Class demonstrations after LDA fit

LDA finds linear combinations of features that maximize class separability and upto 9 basis vectors are observable. The basis vectors are displayed as 28x28 grayscale images. It is benefitial to mention the difference with PCA case. In PCA the data is reduced to lower dimensions by taking all of the classes (all of the data) into consideration and variance of the data is prioritized whereas LDA forms the basis functions such that they best explain the seperation of classes hence as basis vectors of LDA different images than PCA components. After observing Figure 7, the basis vectors as images may seem abstract and uniform however they are originally 784x1 vectors with different feature values hence as vectors they have significant responsibility in class seperation. The subtle variations in grayscale intensity across the components reveal how LDA captures discriminative features.

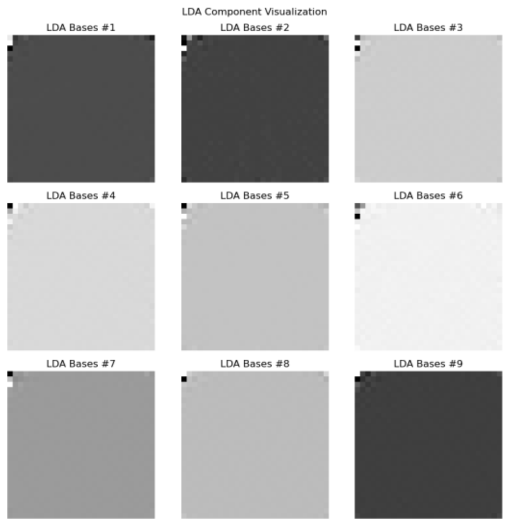
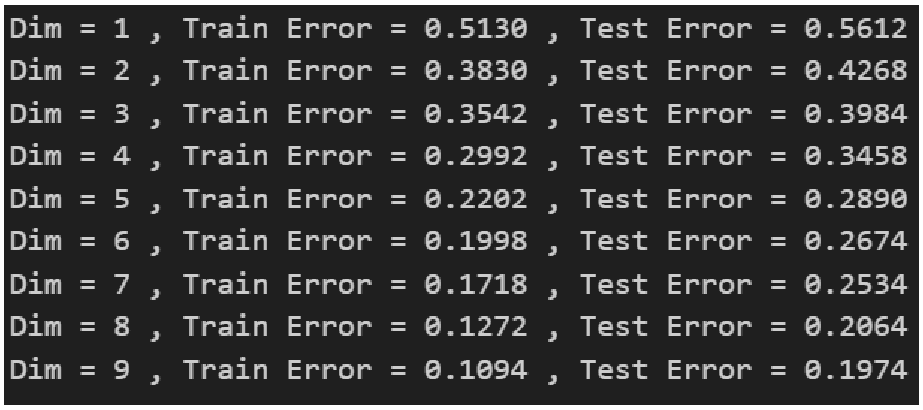


Figure 7 Basis vectors of LDA with 9 dimensions

2)The same quadratic gaussian classifer is used in the previos part is again used to observe the influence of dimensionality. The classifier is trained and tested on 1 through 9 dimensions seperately and the corresponding training and test error will be calculated. In other words, LDA with different dimesnisons is applied to the dataset and the relevant dataset is used for training and testing the classifer.

3)As the number of components increase LDA is able to define the class seperations more precisely hence as the dimensionality of LDA increases a better seperation with the classifier is expected. This statement can be clearly observed experimentally in Figure 9, both test and the train error decreases as dimensinoality increses.

If the results are compared with PCA, in lower dimensions LDA is successful compared to PCA also there is no significant performance superiority of PCA around 50 dimensions when compared to 9 dimensional LDA (best cases). Another distinction is that, LDA does not overfit to the training data since it is optimized for class seperation whereas PCA is optimized for reconstruction which results in overffitting after a threshold. The reson that LDA performs valid even in 9 dimensions can also be explained by the fact that it is optimized for class seperation.



```
Dim = 1 , Train Error = 0.5130 , Test Error = 0.5612
Dim = 2 , Train Error = 0.3830 , Test Error = 0.4268
Dim = 3 , Train Error = 0.3542 , Test Error = 0.3984
Dim = 4 , Train Error = 0.2992 , Test Error = 0.3458
Dim = 5 , Train Error = 0.2202 , Test Error = 0.2890
Dim = 6 , Train Error = 0.1998 , Test Error = 0.2674
Dim = 7 , Train Error = 0.1718 , Test Error = 0.2534
Dim = 8 , Train Error = 0.1272 , Test Error = 0.2064
Dim = 9 , Train Error = 0.1094 , Test Error = 0.1974
```

Figure 8 Numeric results for train and test error

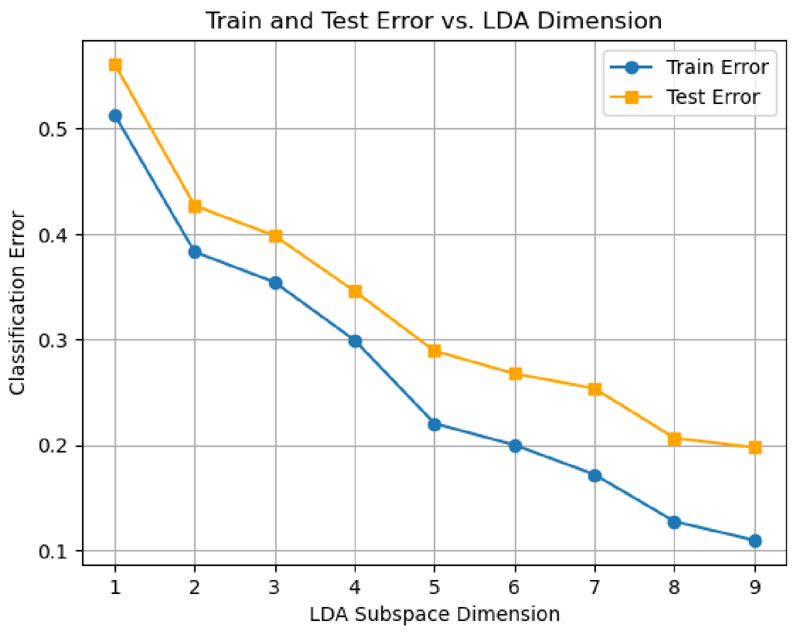


Figure 9 Test classification error and train classification error with LDA applied

## Question 3

In this question as the t-SNE model, TSNE model form sklearn.manifold is used. The model permits the user to change the variables which are defined as n_components(dimensions), perplexity, learning_rate, n_iterations, init. The n_components is fixed to 2 due to the fact that in this question it is desired to represent the dataset in 2 dimensional space. Also applying 2 dimensional t-SNE consumes significant amount of time due to model is permitted to be used only on CPU hence, the initialization parameter is fixed to 'pca' since it fastens the process.

In this case  t-SNE implementation has three variables. Perplexity controls the balance between local and global aspects of the data structure. It can be seen as the effective number of neighbors for each point. Learning _rate determines how much the point positions are updated during optimization, too high or too low can lead to poor results. n_iter sets the number of optimization iterations and affects the convergence quality of the embedding.

In the following figures 4 different values are implemented for each variable and in the relative case other variables are determined to be constant to observe the hyperparameters influence better. It is benefitial to mention that t-SNE is applied to the whole datatset (without train and test split).

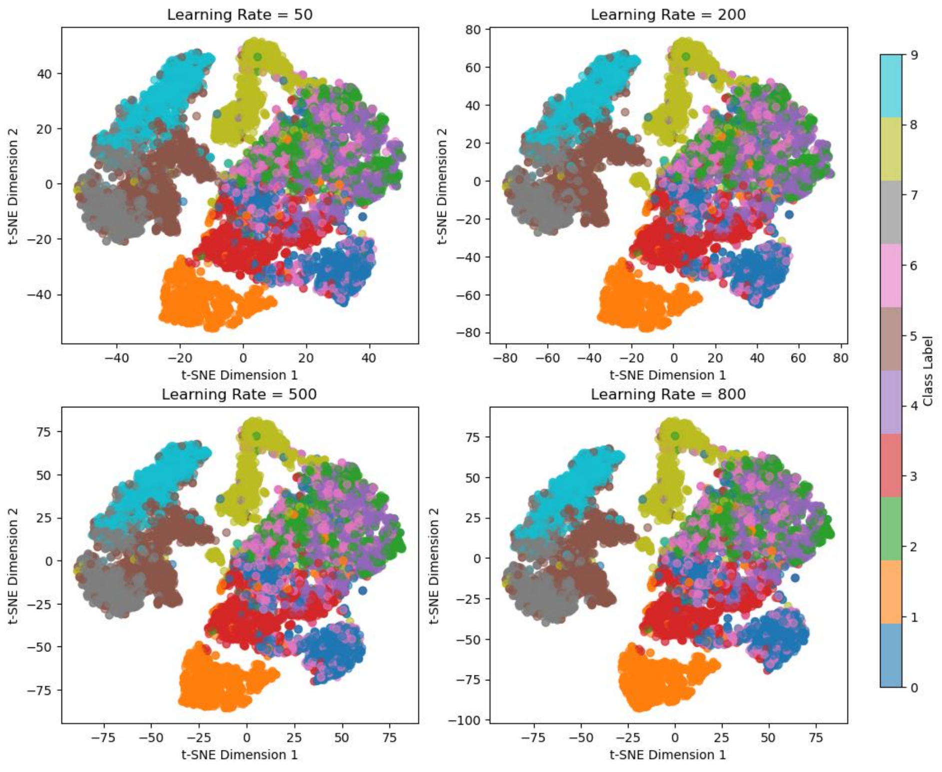


Figure 10 2D visualization of the dataset by using t-SNE with different learning rates

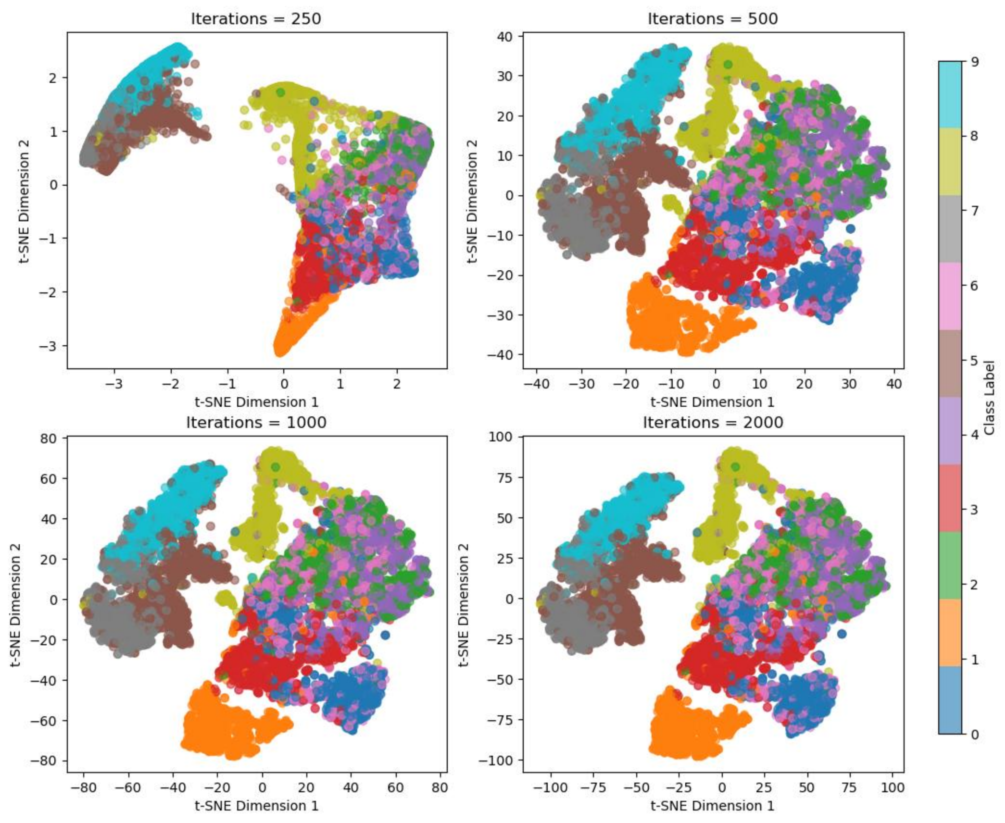


Figure 11 2D visualization of the dataset by using t-SNE with different number of iterations

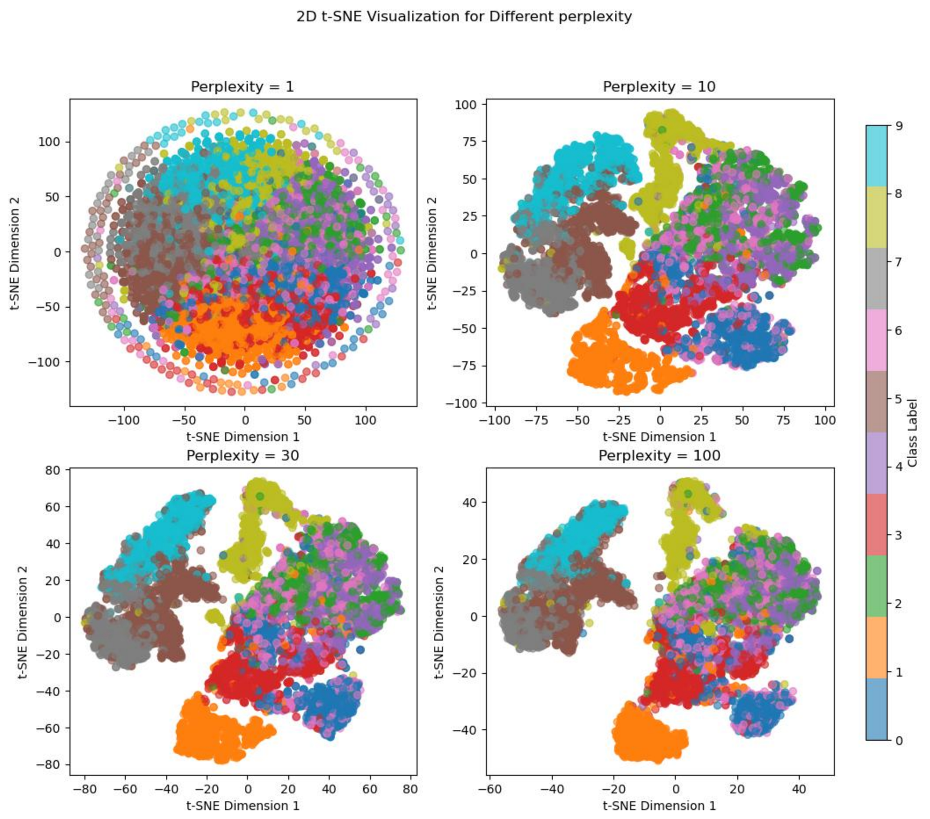


Figure 12 2D visualization of the dataset by using t-SNE with different values of perplexity

As the conclusion of the observations there were minimal changes in the representation of the dataset for different valus of  learning rates are considered. The outstanding change in representation of dataset can be observed when number of iterations and perplexity are changed. According to the observations after 1000 iterations the t-SNE algorithm converges. The iteration is an important factor in determining the computational efficiency hence stopping and 1000 iterations or even with performance tradeoff 500 iterations can be recommended. Moreover higher perplexity resulted in better representation of classes however high perplexity also comes with the cost tradeoff. Hence as a conclusion, a t-SNE model with perplexity assigned as 30 (since there is no significant visual difference and there is computational efficency), learning rate assigned as 200, and stopped at 500 iterations with PCA initialization provides a computationally efficient model and sufficient 2D representation of the dataset. Additionally, 2D representations of the dataset using PCA, LDA and, t-SNE can be observed in the following figure.

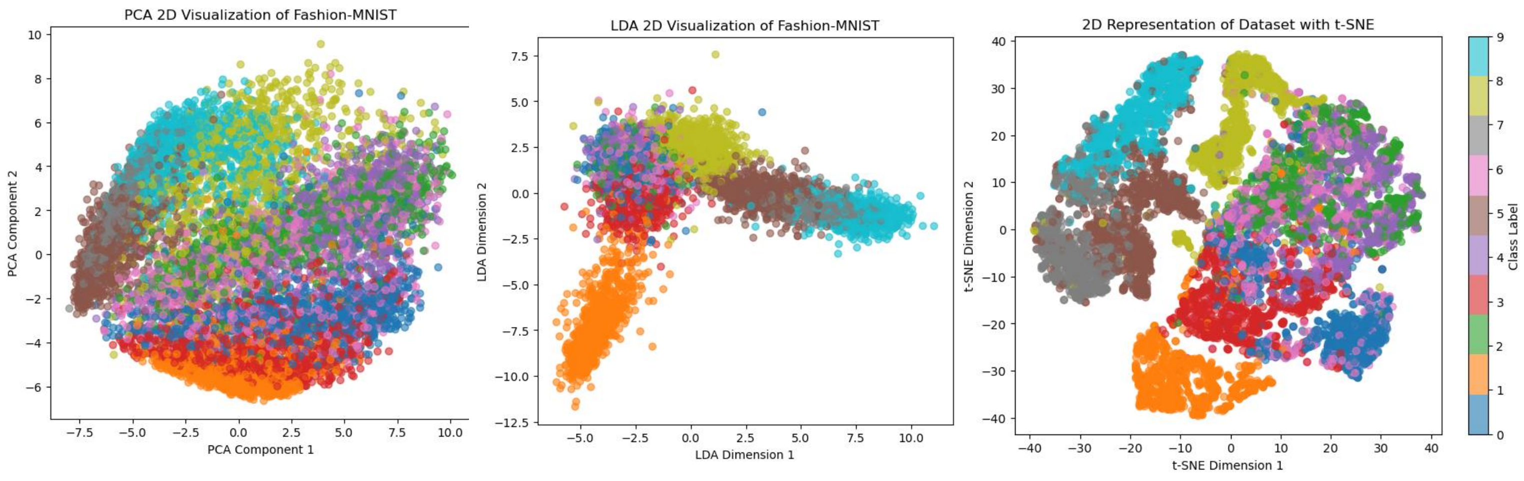


Figure 13 2D representations of the Fashion-MNIST dataset using PCA, LDA, t-SNE

## References

[1] "1.2. Linear and Quadratic Discriminant Analysis," *scikit-learn*, 2025. https://scikit-learn.org/stable/modules/lda_qda.html#mathematical-formulation-of-the-lda-and-qda-classifiers (accessed Mar. 29, 2025).

[2] scikit-learn, "sklearn.decomposition.PCA — scikit-learn 0.20.3 documentation," *Scikit-learn.org*, 2009. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[3] scikit-learn, "sklearn.discriminant_analysis.LinearDiscriminantAnalysis — scikit-learn 0.24.1 documentation," *scikit-learn.org*. https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

[4] scikit-learn developers, "sklearn.manifold.TSNE — scikit-learn 0.21.3 documentation," *Scikit-learn.org*, 2014. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

## APPENDIX

### Python Codes

```
import numpy as np

X = np.loadtxt('fashion_mnist_data.txt', dtype=np.float32)
y = np.loadtxt('fashion_mnist_labels.txt', dtype=np.int32)
print(X.shape)
print(y.shape)
seed = 22101962
np.random.seed(seed)
train_data = []
test_indices = []
num_classes = 10


for c in range(num_classes):
    index_class = np.where(y == c)[0]


    np.random.shuffle(index_class)

    c_train = index_class[:500]
    c_test = index_class[500:1000]

    train_data.append(c_train)
    test_indices.append(c_test)

train_data = np.concatenate(train_data)
test_indices = np.concatenate(test_indices)

X_train = X[train_data]
y_train = y[train_data]
X_test = X[test_indices]
y_test = y[test_indices]


mean_vec = np.mean(X_train, axis=0)
mean_vec_test = np.mean(X_test, axis = 0)
mean_1 = np.mean(X, axis =0)
X_train_centered = X_train - mean_vec
X_test_centered = X_test - mean_vec

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA()
pca.fit(X_train_centered)
```

```python
eigenvalues = pca.explained_variance_
variance_ratio = pca.explained_variance_ratio_
cumulative_variance = variance_ratio.cumsum()

fig, axs = plt.subplots(1, 3, figsize=(18, 4))

axs[0].plot(eigenvalues, marker='o')
axs[0].set_title("Eigenvalues in Descending Order")
axs[0].set_xlabel("Principal Component Index")
axs[0].set_ylabel("Eigenvalue")

axs[1].plot(cumulative_variance, marker='o')
axs[1].set_title("Cumulative Variance Explained by PCA Components")
axs[1].set_xlabel("Principal Component Index")
axs[1].set_ylabel("Cumulative Explained Variance Ratio")

axs[2].plot(variance_ratio, marker='o')
axs[2].set_title("Variance Explained by Each Principal Component")
axs[2].set_xlabel("Principal Component Index")
axs[2].set_ylabel("Variance Ratio")

plt.tight_layout()
plt.show()

eigen_vector = np.array(eigenvalues)
#print(eigen_vector.shape)
mean_img = mean_1.reshape(28, 28)
plt.figure()
plt.imshow(mean_img, cmap='gray')
plt.title("Sample Mean of the Dataset")
plt.axis('off')
plt.show()

num_bases = 5
components = pca.components_[:num_bases]
eigenvalues = pca.explained_variance_[:num_bases]

fig, axs = plt.subplots(1, num_bases, figsize=(15, 3))

for i in range(num_bases):
    pc_img = components[i].reshape(28, 28)
    axs[i].imshow(pc_img, cmap='gray')
    axs[i].set_title(f"PC #{i+1}\nEigenvalue: {eigenvalues[i]:.2f}")
    axs[i].axis('off')

plt.tight_layout()
plt.show()

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
import matplotlib.pyplot as plt
import numpy as np

dimensions = [
    1, 2, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100, 120,
    140, 160, 180, 200, 220, 240, 260, 280, 300, 320,
    350, 375, 400, 784
]

train_errors = []
test_errors = []

for dim in dimensions:
    pca_dim = PCA(n_components=dim)
    pca_dim.fit(X_train_centered)

    projected_train = pca_dim.transform(X_train_centered)
    projected_test = pca_dim.transform(X_test_centered)

    gauss = QuadraticDiscriminantAnalysis(store_covariance=True)
    gauss.fit(projected_train, y_train)

    y_train_hat = gauss.predict(projected_train)
    train_err = 1.0 - np.mean(y_train_hat == y_train)

    y_test_hat = gauss.predict(projected_test)
    test_err = 1.0 - np.mean(y_test_hat == y_test)

    train_errors.append(train_err)
    test_errors.append(test_err)

    print(f"Dim = {dim:3d} | Train Err = {train_err:.4f} | Test Err =
{test_err:.4f}")

print(train_errors, test_errors)
plt.figure(figsize=(9, 5))
plt.plot(dimensions, train_errors, marker='o', color='blue', label='Train
Error')
plt.plot(dimensions, test_errors, marker='s', color='red', label='Test Error')
plt.xlabel("Number of PCA Components")
plt.ylabel("Classification Error")
plt.title("Classification Error vs. PCA Dimension without regularization")
plt.legend()
plt.grid(True)
plt.show()
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
QuadraticDiscriminantAnalysis
import matplotlib.pyplot as plt
```

```
train_errors = []
test_errors = []
dims_to_try = range(1, 10)

for dim in dims_to_try:
    lda = LinearDiscriminantAnalysis(n_components=dim)
    lda.fit(X_train_centered, y_train)

    X_train_lda = lda.transform(X_train)
    X_test_lda = lda.transform(X_test)

    qda = QuadraticDiscriminantAnalysis(store_covariance=True)
    qda.fit(X_train_lda, y_train)

    y_train_pred = qda.predict(X_train_lda)
    y_test_pred = qda.predict(X_test_lda)

    train_err = 1.0 - np.mean(y_train_pred == y_train)
    test_err = 1.0 - np.mean(y_test_pred == y_test)

    train_errors.append(train_err)
    test_errors.append(test_err)

    print(f"Dim = {dim} , Train Error = {train_err:.4f} , Test Error =
{test_err:.4f}")

plt.figure()
plt.plot(list(dims_to_try), train_errors, marker='o', label='Train Error')
plt.plot(list(dims_to_try), test_errors, marker='s', color='orange',
label='Test Error')
plt.xlabel("LDA Subspace Dimension")
plt.ylabel("Classification Error")
plt.title("Train and Test Error vs. LDA Dimension")
plt.legend()
plt.grid(True)
plt.show()

lda_full = LinearDiscriminantAnalysis(n_components=9)
lda_full.fit(X_train, y_train)

print("lda_full.scalings_.shape:", lda_full.scalings_.shape)

num_bases_lda = min(9, lda_full.scalings_.shape[1])


complete_lda = LinearDiscriminantAnalysis(n_components=9)
complete_lda.fit(X_train_centered, y_train)

bases = complete_lda.scalings_
```

```
fig, axs = plt.subplots(3, 3, figsize=(9, 9))
for i in range(min(9, bases.shape[0])):
    mean_img = bases[:,i].reshape(28, 28).T
    axs[i // 3, i % 3].imshow(mean_img, cmap='gray_r')
    axs[i // 3, i % 3].set_title(f'LDA Bases #{i+1}')
    axs[i // 3, i % 3].axis('off')

plt.suptitle("LDA Component Visualization")
plt.tight_layout()
plt.show()

complete_lda = LinearDiscriminantAnalysis(n_components=9)
complete_lda.fit(X_train_centered, y_train)

means = complete_lda.means_

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

for i in range(10):
    row, col = divmod(i, 5)
    mean_img = means[i].reshape(28, 28).T
    axs[row, col].imshow(mean_img, cmap='gray_r')
    axs[row, col].set_title(f'Class Mean #{i+1}')
    axs[row, col].axis('off')

plt.suptitle("LDA Class Means")
plt.tight_layout()
plt.show()
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200, n_iter=500,
            init='pca', random_state=42)
X_tsne = tsne.fit_transform(X)

plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='tab10', alpha=0.6)
plt.title("2D Representation of Dataset with t-SNE")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.colorbar(label='Class Label')
plt.savefig("tsne_plot_nit500.png")

plt.show()


pca_2d = PCA(n_components=2)

X_pca_2d = pca_2d.fit_transform(X)
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X_pca_2d[:, 0], X_pca_2d[:, 1], c=y, cmap='tab10', alpha=0.6)
plt.title("2D Representation of Dataset with PCA")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label='Class Label')
plt.savefig("pca_plot.png")
plt.show()

lda_2d = LinearDiscriminantAnalysis(n_components=2)

X_lda_2d = lda_2d.fit_transform(X, y)

plt.figure(figsize=(8, 6))
plt.scatter(X_lda_2d[:, 0], X_lda_2d[:, 1], c=y, cmap='tab10', alpha=0.6)
plt.title("2D Representation of Dataset with LDA")
plt.xlabel("LDA Dimension 1")
plt.ylabel("LDA Dimension 2")
plt.colorbar(label='Class Label')
plt.savefig("lda_plot.png")
plt.show()

learning_rates = [50, 200, 500, 800]

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

for ax, lr in zip(axes.flatten(), learning_rates):
    tsne = TSNE(n_components=2, perplexity=30, learning_rate=lr, n_iter=1000,
                init='pca', random_state=22101962)
    X_tsne = tsne.fit_transform(X)

    sc = ax.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='tab10', alpha=0.6)
    ax.set_title(f"Learning Rate = {lr}")
    ax.set_xlabel("t-SNE Dimension 1")
    ax.set_ylabel("t-SNE Dimension 2")

plt.subplots_adjust(right=0.85)

cbar_ax = fig.add_axes([0.88, 0.15, 0.02, 0.7])

cbar = fig.colorbar(sc, cax=cbar_ax)
cbar.set_label("Class Label")

plt.suptitle("2D t-SNE Visualization for Different Learning Rates")
plt.show()

iterations = [250, 500, 1000,2000]

fig, axes = plt.subplots(2, 2, figsize=(12, 10))
```

```
for ax, it in zip(axes.flatten(), iterations):
    tsne = TSNE(n_components=2, perplexity=30, learning_rate=200, n_iter=it,
                init='pca', random_state=22101962)
    X_tsne = tsne.fit_transform(X)

    sc = ax.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='tab10', alpha=0.6)
    ax.set_title(f"Iterations = {it}")
    ax.set_xlabel("t-SNE Dimension 1")
    ax.set_ylabel("t-SNE Dimension 2")

plt.subplots_adjust(right=0.85)

cbar_ax = fig.add_axes([0.88, 0.15, 0.02, 0.7])

cbar = fig.colorbar(sc, cax=cbar_ax)
cbar.set_label("Class Label")

plt.suptitle("2D t-SNE Visualization for Different Learning Rates")
plt.show()

perplexity = [1, 10, 30,100]

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

for ax, per in zip(axes.flatten(), perplexity):
    tsne = TSNE(n_components=2, perplexity=per, learning_rate=200,
n_iter=1000,
                init='pca', random_state=22101962)
    X_tsne = tsne.fit_transform(X)

    sc = ax.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='tab10', alpha=0.6)
    ax.set_title(f"Perplexity = {per}")
    ax.set_xlabel("t-SNE Dimension 1")
    ax.set_ylabel("t-SNE Dimension 2")

plt.subplots_adjust(right=0.85)

cbar_ax = fig.add_axes([0.88, 0.15, 0.02, 0.7])

cbar = fig.colorbar(sc, cax=cbar_ax)
cbar.set_label("Class Label")

plt.suptitle("2D t-SNE Visualization for Different perplexity")
plt.show()
```