Impact of Dimensionality Reduction on Classification Accuracy: A Comparative Study Using PCA and LDA with Difference Classifiers on the MNIST Dataset

TAN CHOO HUI
Matric No. : G2408251B
School of Electrical and Electronic Engineering
Email: CHOOHUI001@e.ntu.edu.sg

***Abstract*—In this project, I have tried different methods to reduce the dimension of the dataset and benchmarked performance with different classification models, and the dataset used in this project is the MNIST digit dataset. Datasets with large feature spaces will normally lead to issues such as slow computation and increased risk of overfitting while training the model. To resolve these problems, Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are used to reduce the number of features before feeding into the classifiers. There are three types of classifiers that are used in this project, which are the Mahalanobis distance, logistic regression, and K-Nearest Neighbors (k-NN). The various combinations of dimensional reduction methods with different numbers of dimensions are tested and benchmarked for their performances across accuracy. Our experiment found that when PCA using only 31 components with the k-NN classifier performed the highest accuracy, which is 97.55%. These findings highlight the importance of effective preprocessing in image-based classification tasks and demonstrate the potential of k-NN combined with PCA for improving digit recognition accuracy.**

***Index Terms*—dimensionality reduction, classification, mahalanobis distance, K-Nearest Neighbors, Logistic Regression, Principal Component Analysis, Linear Discriminant Analysis**

I. Introduction

Using high-dimensional data usually adds to the computation cost and overfitting, while the performance of the classification models drops. Dimensionality reduction techniques help transform the data into a lower-dimensional space to remove noise from the data while keeping important information [1]. This project examined two important methods, PCA and LDA. PCA is an unsupervised method that recognizes the directions of maximum variance to achieve feature compression [2]. Linear Discriminant Analysis (LDA) is a supervised method that recognizes the directions that best separate the classes. LDA is generally also evaluated to see at which fewer number of features maximum classification accuracy is received [3]. MNIST dataset is used in this project which has total of 70000 grayscale images of handwritten digits which are of $28x28$ size. This is a standard dataset used for testing classification models and dimensionality reduction techniques in the literature. Three classifiers have been used which include the Mahalanobis distance, logistic regression and K-nearest neighbor and all of them feed on a different type of dataset which is got after preprocessing. PCA and LDA are being evaluated to see how the dimensionality is impacting the accuracy and also benchmarking across each other. This project looks into the influence of various dimensionality reductions on classification accuracy and assesses the best combinations.

II. Background and Literature Review

A. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is used for transforms data from a high-dimensional space to a low-dimensional space. PCA is also commonly used for dimensionality reduction, data compression and feature extraction. The utilization of PCA is motivated by two fundamental objectives which are Facilitation of Dimensionality Reduction and Mitigation of Overfitting. By identifies the principal components that along which the variance of the data is maximized, and it is obtained by the eigenvalue decomposition of the covariance matrix or the singular value decomposition (SVD) of the data matrix. The principal components are having relation to each other, ensuring that the reduced dimensions are uncorrelated.

B. Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a supervised learning method used for dimensionality reduction and classification, introduced by R.A. Fisher in 1936 [10]. It works by finding linear combinations of features that maximize class separability, based on the ratio of between-class to within-class variance. Assuming normally distributed classes with equal covariances, LDA provides optimal linear decision boundaries and is widely used in fields like pattern recognition, bioinformatics, and finance. Over time, it has been extended through methods like regularized LDA, kernel LDA, and sparse LDA to handle high-dimensional and non-linear data, maintaining its relevance due to simplicity, interpretability, and efficiency.

C. Classification Technique

Mahalanobis distance : Mahalanobis distance was proposed by Mahalanobis [4] and is known for the ability to measure distance in the presence of correlation and different scale (variance) in features. It is popularly used in classification problems where the variance/covariance of the features impacts the class boundary significantly. Research like De Maesschalck et al. shows that the Mahalanobis distance is effective for multivariate data analysis, and classification which otherwise have elliptical distributions [5].

Logistic regression : Logistic regression remains a staple in classification due to its probabilistic framework and interpretability. As highlighted in Hosmer and Lemeshow's work [6], logistic regression effectively models the relationship between input features and class probabilities. Its application spans across domains like medical diagnostics and text classification. Ng and Jordan [7] further explored its performance in high-dimensional spaces, showing its advantages over generative models when data is abundant.

K-Nearest Neighbors (k-NN) : Cover and Hart devised a non-parametric k-NN based classification method [8], which relies on distance. It is useful in many fields due to its simplicity, including image recognition and pattern recognition. Researchers Beyer and others show that k-NN works better with the right distance metric [9]. This effect becomes stronger when the dimensionality of data is increased. In such cases, use of traditional metric like Euclidean distance may not perform as well.

III. Methodology

In Figure 1, the whole process flow of this project is shown. The project begins with the loading of the MNIST dataset, and the MNIST dataset will then be preprocessed using normalization. After that, the dimensionality reduction techniques such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and combined methods such as PCA followed by LDA (PCA→LDA) and LDA followed by PCA (LDA→PCA) are applied to the dataset and will be evaluated by accuracy. Next, the processed dataset will be fed into models such as Mahalanobis distance classifier, Logistic Regression, and k-Nearest Neighbors (k-NN). The accuracy of each model will be evaluated, and the best performance classifier with its dimensionality reduction method will be further evaluated by confusion matrix and other metrics such as F1-score, accuracy, recall, and precision. Further details will be explained in the following section.

A. Dataset Description

In this project, Modified National Institute of Standards and Technology (MNIST) data is used as the benchmarking dataset for the experiment conducted. This dataset consists of $70,000$ images of handwritten digits from 0 to 9 and it is in grayscale image format. All the images in the MNIST dataset have a

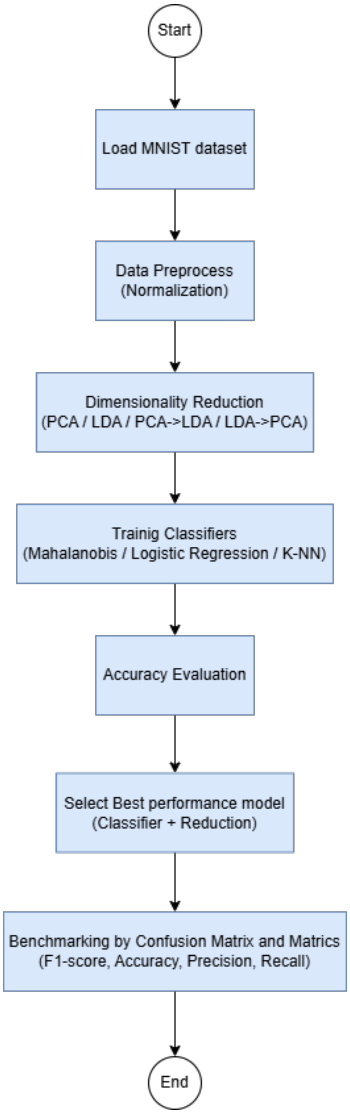

Fig. 1: Flowchart of this project

standard of $25x25$ pixels in size. The dataset is then split into $60,000$ images for the training set and $10,000$ images into the test set. MNIST is a famous dataset nowadays for testing the performance of models due to its simplicity and accessibility. Figure 2 is showing the examples from the MNIST dataset.

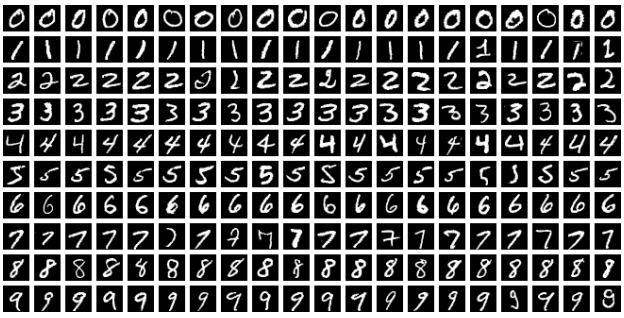

Fig. 2: Examples from the MNIST dataset

B. Data Pre-processing

The MNIST dataset was loaded through the PyTorch library directly and after loaded the dataset, it being convert from images to tensors format. The dataset's global mean and standard deviation is computed across all the training samples. The raw image data had scaled from the original range [0,255] to [0,1] by dividing by 255 and then reshaped into a 2-dimensional array with the shape of (N, 784), where N is the total number of training images. After that, the mean and standard deviation is computed formular 1 and formular 2 respectively.

$$\mu = \frac{1}{N \times D} \sum_{i=1}^{N} \sum_{j=1}^{D} x_{ij} \tag{1}$$

$$\sigma = \sqrt{\frac{1}{N \times D} \sum_{i=1}^{N} \sum_{j=1}^{D} (x_{ij} - \mu)^2} \tag{2}$$

where x_{ij} denotes the pixel value at position j in sample i, and $D = 784$ represents the flattened dimensionality of each image. The computed values ($\mu \approx 0.1307$, $\sigma \approx 0.3081$) were subsequently used to normalize both the training and test datasets by the transformation formular 3.

$$x' = \frac{x - \mu}{\sigma} \tag{3}$$

The normalization is purpose to make the pixel values to be centered around zero with unit variance, this aim to stabilizing and accelerating the training of the classification models. At the end of the data pre-processing, it results a final shape of (60000, 784) for training set and final shape of (10000, 784) in the test set.

C. Experiments Method

After the MNIST dataset pre-processing is done, will then apply the dimensionality reduction techniques to it to simplify the feature space and analyze their effect on the classification performance. PCA with multiple configuration will be tested by varying the number of components from 1 to 784 in steps of 10, along with the full 784-dimensional space for baseline comparison. For LDA will be tested from range 1 to 9 which corresponding to the maximum number of classes minus one in MNIST. The combination of PCA and LDA will also being implemented for further explore for both ways, PCA followed by LDA and LDA followed by PCA. In the combination of PCA and LDA, the component counts were carefully adjusted to remain within feasible limits.

After this, Mahalanobis classifier, Logistice Regression and k-NN (k=5) will then be trained by all the possible outcome of dataset from the dimensionality reduction before and evaluated by accuracy in the test set.

After the evaluation, the best classifier with it configuration that has the highest accuracy will then be further evaluate by the confusion matrix and others performance matric such as F1-score, accuracy, precision and recall as shown in the formular below:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

This evaluation will provide more insight to enable a deeper comparison across classifiers and dimensionality reduction techniques.

IV. Results and Discussion

A. Result of Dimensionality Reduction on difference components

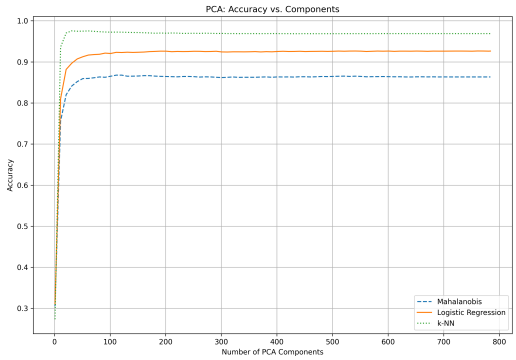

Fig. 3: Graph of accuracy versus components for PCA

First of all, we analyze the effectiveness of all the dimensionality reduction method towards the classifier performance in accuracy. The first dimensionality reduction to be start with is PCA. The figure 3 shows the correlation between PCA components and classification accuracies for three particular models. All models experience a rapid increase in accuracy as more components are added. The increases in accuracies are more significant with the first 100 components. In general, the k-NN model produces the highest accuracy metrics but plateaus at around 97.5% with approximately 30 components. The accuracy of the Logistic Regression model applies a consistent metric increase, measuring around 92.6%, which utilizes about 760 PCA components. The Mahalanobis model demonstrates the lowest accuracy metrics of approximately 86.7%, with about 110 components. The above results imply that PCA will be an effective methodology in reducing dimensionality while maintaining an acceptable accuracy level. This reduction in dimensionality will benefit the proposed k-NN model having fewer components.

Next, come to the analysis about the LDA technique based on the figure 4. The graph shows that all classifiers show a

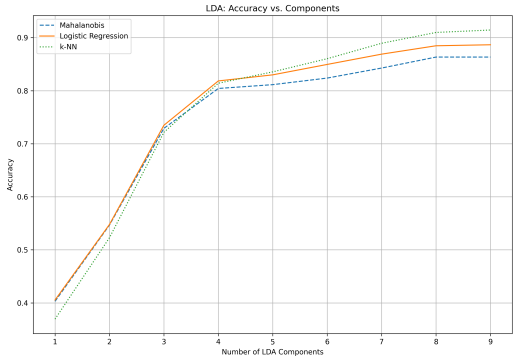

Fig. 4: Graph of accuracy versus components for LDA

consistent increase in accuracy as LDA components increase from 1 to 9. In the number of components range between 1 to 4, the Logistic Regression and Mahalanobis perform better than k-NN, but after the number of components increases from 5 to 9, the k-NN overperforms across those methods and achieves the peak accuracy about 91% at 9 components. This trend highlights that LDA effectively enhances class separability with increasing component.

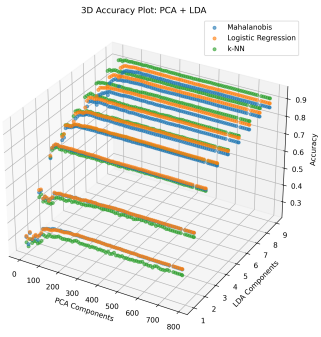

Fig. 5: 3D Graph of accuracy versus components for PCA→LDA

For the PCA→LDA method, the figure 5 shows the effect of different combinations of PCA and LDA components on classifier accuracy. From the view of number of LDA components, the accuracy increases consistently as the number of LDA components increases. For the PCA components increase, the model accuracy has a rapid increase before around 100 components, but after that, the model performance shows almost the same, no significant increment. The k-NN gains the highest accuracy overall in this approach. The graph indicates that applying both PCA and LDA can improve performance, but the gain becomes marginal beyond a certain PCA component count. This trend shows that PCA→LDA helps in reducing dimensionality while retaining classification performance.

Last but not least, the figure 6 shows the effect of different

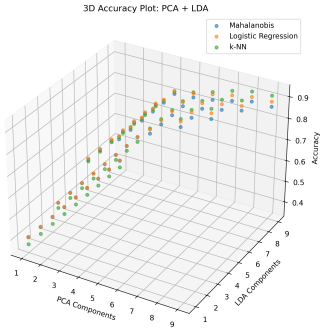

Fig. 6: 3D Graph of accuracy versus components for LDA→PCA

combinations of LDA and PCA components on model accuracy for the LDA→PCA method. As the number of LDA components increases from 1 to 9, the accuracy improves steadily. Similarly, as PCA components increase, the model accuracy also increases, but the gain is more significant at lower component counts, especially from 1 to 5. After that, the improvement becomes minor. Among the models, k-NN achieves the highest accuracy overall, followed by Logistic Regression and Mahalanobis. This trend shows that applying LDA before PCA can effectively reduce dimensionality while maintaining good classification performance.

TABLE I: Best Accuracy and Component Configuration for Each Classifier and Dimensionality Reduction Method

Method	**Mahalanobis**	**Logistic Regression**	**k-NN (k=5)**
PCA	86.77% (111)	92.65% (761)	**97.55% (31)**
LDA	86.34% (8)	88.66% (9)	91.44% (9)
PCA → LDA	86.81% (191, 8)	89.12% (171, 9)	92.01% (161, 9)
LDA → PCA	86.34% (8, 8)	88.66% (9, 9)	91.44% (9, 9)

The Table I is showing the comparison of the best accuracy of different dimensionality reduction and classifiers. From the statistic we get an insight that k-NN consistently achieve as the highest performance with all the dimensionality reduction technique applied. k-NN also achieve the highest accuracy across all the experiment method which is 97.55% while using PCA with 31 components. This means that k-NN is benefits significantly from PCA's ability to preserve local structure with relatively few components. In Logistic Regression, it achieved a high accuracy 92.65% when with PCA at 761 components. This show that it requires more comprehensive feature retention for optimal performance. For Mahalanobis classifier, it present a slightly lower performance across all method which is it only achieve the highest accuracy only 86.81% when using the PCA first and then apply LDA technique with combination of components (191, 8).

when we take an overall look on the result, PCA yielded the best overall accuracy in all the classifiers especially for

the k-NN. The combined methods such ad PCA→LDA and LDA→PCA provided moderate overall performance over LDA alone but did not outperform compared to PCA alone. This indicates that, for the MNIST dataset, PCA is highly effective as a standalone dimensionality reduction technique.

B. Final Result of Classifier Performance

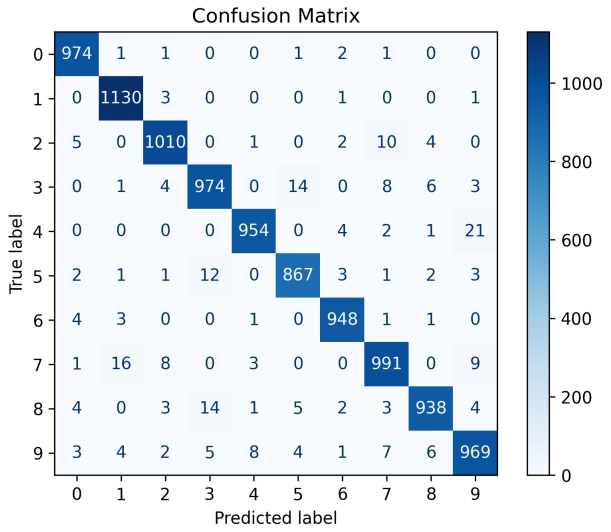

Fig. 7: Confusion Matrix of final model

The figure 7 show the final result from the k-NN classifier with number of k is 5 and using the PCA with 31 components for dimensionality reduction. From the confusion matrix, we can notice that there's high number of correct classifications which the confusion matrix is concentrated along the diagonal. The models is usually confuses about the behavior of number that look similar such as 3 and 5, and 4 and 9. We can notice this with that the confusion matrix show that digit 3 is often to be classified into 5 and 8 while 4 always been classified as 9 when misclassified happened.

TABLE II: Classification Performance Metrics

Metric	Value
F1-score	97.55%
Accuracy	97.55%
Recall	97.55%
Precision	97.55%

From the evaluation metrics in table II, the classification model show it ability to perform well in MNIST with very high percentage across all the performance metric which achieve as high as 97.55%. This result show that the model maintain a good balance between the precision and recall, which is a reliable classifier for the given dataset.

The figure 8 and figure 9 shows some samples that predited correctly and incorrect respectively.we can notice that most of the wrong predictions are caused by improper or ambiguous handwriting, where the digit shape closely resembles another, leading to misclassification.

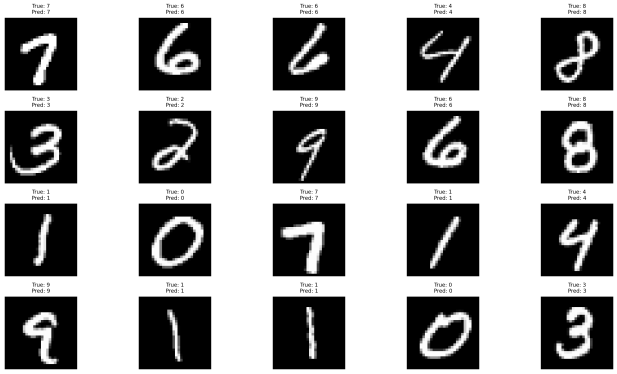

Fig. 8: Samples of correctly predicted

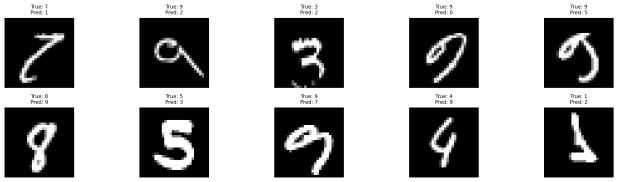

Fig. 9: Samples of incorrectly predicted

V. Conclusion

In this project, we had successfully investigate the effect of dimensionality reduction using PCA, LDA and the cross-combination of both of the technique and evaluate the performance on MNIST dataset. The results of this research show that the PCA alone with 31 components applied to the k-NN classifiers achieved the highest accuracy which is 97.55% in the final evaluation. This results highlighted that the strength of PCA in preserving key features with minimal dimensions. Beside that, the others method such as LDA and cross combination between PCA and LDA also aids as a help to improve the classification accuracy but still not as good as the PCA. The confusion matrix and evaluation metrics further confirm the model's reliability. Most of the errors from the final model caused by ambiguous handwriting. Overall, the project demonstrates that effective dimensionality reduction, especially PCA, significantly boosts classification performance and efficiency.

References

[1] X. Jiang, "Linear Subspace Learning-Based Dimensionality Reduction," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 16–26, Mar. 2011.

[2] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002.

[3] X. Jiang, B. Mandal, and A. Kot, "Asymmetric Principal Component and Discriminant Analyses for Pattern Classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 931–937, May 2009.

[4] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences of India*, vol. 12, pp. 49–55, 1936.

[5] R. De Maesschalck, D. Jouan-Rimbaud, and D. L. Massart, "The Mahalanobis distance," *Chemometrics and Intelligent Laboratory Systems*, vol. 50, no. 1, pp. 1–18, 2000.

[6] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd ed. Hoboken, NJ, USA: Wiley, 2013.

[7] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Advances in Neural Information Processing Systems*, vol. 14, pp. 841–848, 2002.
[8] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.
[9] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'nearest neighbor' meaningful?" in *Proc. 7th Int. Conf. Database Theory (ICDT)*, 1999, pp. 217–235.
[10] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

APPENDIX

The source code are all written in the jupyter notebook and will be included in this appendix and also can be look forward the repository at link : https://github.com/ChooHui0429/EE6222-Assigment-1

A. Part 1: Evaluation testing for dimensionality reduction

1) Library Import:

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt

import torch
from torchvision import datasets, transforms

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

2) MNIST Dataset Loading:

```
# Load MNIST without normalization
raw_transform = transforms.ToTensor()

train_dataset_raw = datasets.MNIST(root="./data",
    train=True, transform=raw_transform, download=
    True)
test_dataset_raw = datasets.MNIST(root="./data",
    train=False, transform=raw_transform, download=
    True)

# Calculation of mean and standard deviation for
    normalization purpose
X_train_raw = train_dataset_raw.data.numpy().astype(
    'float32') / 255.0
y_train = train_dataset_raw.targets.numpy()
X_test_raw = test_dataset_raw.data.numpy().astype('
    float32') / 255.0
y_test = test_dataset_raw.targets.numpy()

X_train = X_train_raw.reshape(len(X_train_raw), -1)
X_test = X_test_raw.reshape(len(X_test_raw), -1)

mean = X_train.mean()
std = X_train.std()
print(f"Computed␣Mean:␣{mean:.4f},␣Std:␣{std:.4f}")

# Normalized MNIST
normalized_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((mean,), (std,))
])

train_dataset = datasets.MNIST(root="./data", train=
    True, transform=normalized_transform, download=
    False)
test_dataset = datasets.MNIST(root="./data", train=
    False, transform=normalized_transform, download=
    False)

print("Training␣data␣shape:", X_train.shape, "Labels
    ␣shape:", y_train.shape)
print("Test␣data␣shape:", X_test.shape, "Labels␣
    shape:", y_test.shape)
```

3) Function of Computation of PCA and LDA:

```
def apply_pca(X_train, X_test, n_components):
    pca = PCA(n_components=n_components)
    return pca.fit_transform(X_train), pca.transform
        (X_test)

def apply_lda(X_train, y_train, X_test, n_components
    ):
    lda = LDA(n_components=n_components)
    return lda.fit_transform(X_train, y_train), lda.
        transform(X_test)
```

4) Mahalanobis Classifier Function:

```
class MahalanobisClassifier:
    def fit(self, X, y):
        X = np.atleast_2d(X)
        self.classes_ = np.unique(y)
        self.class_means_ = {c: X[y == c].mean(axis
            =0) for c in self.classes_}
        cov_matrix = np.cov(X, rowvar=False, ddof=0)
        if cov_matrix.ndim == 0:
            self.inv_cov_ = 1 / (cov_matrix + 1e-6)
        else:
            self.inv_cov_ = np.linalg.pinv(
                cov_matrix + 1e-6 * np.eye(
                cov_matrix.shape[0]))
        return self

    def predict(self, X):
        X = np.atleast_2d(X)
        y_pred = []
        for x in X:
            distances = [np.dot(np.dot((x - self.
                class_means_[c]), self.inv_cov_), (x
                 - self.class_means_[c]).T) for c in
                 self.classes_]
            y_pred.append(self.classes_[np.argmin(
                distances)])
        return np.array(y_pred)

    def score(self, X, y_true):
        return np.mean(self.predict(X) == y_true)
```

5) Experimental Training:

```
# Define of PCA and LDA possible value and results
    storage
pca_components_list = list(range(1, 785, 10)) +
    [784]
lda_components_list = list(range(1, 10))

results_pca = {"Components": [], "Mahalanobis": [],
    "Logistic␣Regression": [], "k-NN": []}
results_lda = {"Components": [], "Mahalanobis": [],
    "Logistic␣Regression": [], "k-NN": []}
results_pca_lda = {"PCA␣Components": [], "LDA␣
    Components": [], "Mahalanobis": [], "Logistic␣
    Regression": [], "k-NN": []}
results_lda_pca = {"PCA␣Components": [], "LDA␣
    Components": [], "Mahalanobis": [], "Logistic␣
    Regression": [], "k-NN": []}
```

```
# Test PCA
for n_comp in pca_components_list:
    print("Testing␣PCA␣n_comp␣=␣" + str(n_comp))
    pca = PCA(n_components=n_comp)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    mah_clf = MahalanobisClassifier().fit(
        X_train_pca, y_train)
    log_clf = LogisticRegression(max_iter=500,
        solver='lbfgs', multi_class='multinomial').
        fit(X_train_pca, y_train)
    knn_clf = KNeighborsClassifier(n_neighbors=5).
        fit(X_train_pca, y_train)

    results_pca["Components"].append(n_comp)
    results_pca["Mahalanobis"].append(mah_clf.score(
        X_test_pca, y_test))
    results_pca["Logistic␣Regression"].append(
        log_clf.score(X_test_pca, y_test))
    results_pca["k-NN"].append(knn_clf.score(
        X_test_pca, y_test))

# Test LDA
for n_comp in lda_components_list:
    print("Testing␣LDA␣n_comp␣=␣" + str(n_comp))
    lda = LDA(n_components=n_comp)
    X_train_lda = lda.fit_transform(X_train, y_train
        )
    X_test_lda = lda.transform(X_test)

    mah_clf = MahalanobisClassifier().fit(
        X_train_lda, y_train)
    log_clf = LogisticRegression(max_iter=500,
        solver='lbfgs', multi_class='multinomial').
        fit(X_train_lda, y_train)
    knn_clf = KNeighborsClassifier(n_neighbors=5).
        fit(X_train_lda, y_train)

    results_lda["Components"].append(n_comp)
    results_lda["Mahalanobis"].append(mah_clf.score(
        X_test_lda, y_test))
    results_lda["Logistic␣Regression"].append(
        log_clf.score(X_test_lda, y_test))
    results_lda["k-NN"].append(knn_clf.score(
        X_test_lda, y_test))

# Test PCA -> LDA
for pca_comp in pca_components_list:
    pca = PCA(n_components=pca_comp)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    max_lda_components = min(pca_comp, 9)

    for lda_comp in range(1, max_lda_components + 1)
        :
        print("Testing␣PCA␣->␣LDA␣n_comp␣=␣" + str(
            pca_comp) + ",␣" + str(lda_comp))
        try:
            lda = LDA(n_components=lda_comp)
            X_train_pca_lda = lda.fit_transform(
                X_train_pca, y_train)
            X_test_pca_lda = lda.transform(
                X_test_pca)

            mah_clf = MahalanobisClassifier().fit(
                X_train_pca_lda, y_train)
            log_clf = LogisticRegression(max_iter
                =500, solver='lbfgs').fit(
                X_train_pca_lda, y_train)
            knn_clf = KNeighborsClassifier(
                n_neighbors=5).fit(X_train_pca_lda,
                y_train)

            results_pca_lda["PCA␣Components"].append
                (pca_comp)
            results_pca_lda["LDA␣Components"].append
                (lda_comp)
            results_pca_lda["Mahalanobis"].append(
                mah_clf.score(X_test_pca_lda, y_test
                ))
            results_pca_lda["Logistic␣Regression"].
                append(log_clf.score(X_test_pca_lda,
                 y_test))
            results_pca_lda["k-NN"].append(knn_clf.
                score(X_test_pca_lda, y_test))

        # Skip this iteration if LDA fails
        except np.linalg.LinAlgError:
            print(f"      ␣SVD␣did␣not␣converge␣for␣
                PCA={pca_comp},␣LDA={lda_comp}␣   ␣
                Skipping!")
            continue

# Test LDA -> PCA
for lda_comp in lda_components_list:
    try:
        lda = LDA(n_components=lda_comp)
        X_train_lda = lda.fit_transform(X_train,
            y_train)
        X_test_lda = lda.transform(X_test)

        for pca_comp in range(1, lda_comp + 1):
            print("Testing␣LDA␣->␣PCA␣n_comp␣=␣" +
                str(lda_comp) + ",␣" + str(pca_comp)
                )

            pca = PCA(n_components=pca_comp)
            X_train_lda_pca = pca.fit_transform(
                X_train_lda)
            X_test_lda_pca = pca.transform(
                X_test_lda)

            mah_clf = MahalanobisClassifier().fit(
                X_train_lda_pca, y_train)
            log_clf = LogisticRegression(max_iter
                =500, solver='lbfgs').fit(
                X_train_lda_pca, y_train)
            knn_clf = KNeighborsClassifier(
                n_neighbors=5).fit(X_train_lda_pca,
                y_train)

            # Store results
            results_lda_pca["LDA␣Components"].append
                (lda_comp)
            results_lda_pca["PCA␣Components"].append
                (pca_comp)
            results_lda_pca["Mahalanobis"].append(
                mah_clf.score(X_test_lda_pca, y_test
                ))
            results_lda_pca["Logistic␣Regression"].
                append(log_clf.score(X_test_lda_pca,
                 y_test))
            results_lda_pca["k-NN"].append(knn_clf.
                score(X_test_lda_pca, y_test))

    except np.linalg.LinAlgError:
        print(f"      ␣SVD␣did␣not␣converge␣for␣LDA
            ={lda_comp}␣   ␣Skipping!")
        continue
```

6) Get the Best Component Number for Best Model in terms of Accuracy for All 4 Method:

```
# Function to get the best accuracy and
    corresponding components
```

```
def get_best_accuracy(results_dict, method_name):
    best_idx_mahal = np.argmax(results_dict["
        Mahalanobis"])
    best_idx_logistic = np.argmax(results_dict["
        Logistic Regression"])
    best_idx_knn = np.argmax(results_dict["k-NN"])

    best_mahalanobis = results_dict["Mahalanobis"][
        best_idx_mahal]
    best_logistic = results_dict["Logistic
        Regression"][best_idx_logistic]
    best_knn = results_dict["k-NN"][best_idx_knn]

    best_comp_mahal = results_dict["Components"][
        best_idx_mahal] if "Components" in
        results_dict else (results_dict["PCA
        Components"][best_idx_mahal], results_dict["
        LDA Components"][best_idx_mahal])
    best_comp_logistic = results_dict["Components"][
        best_idx_logistic] if "Components" in
        results_dict else (results_dict["PCA
        Components"][best_idx_logistic],
        results_dict["LDA Components"][
        best_idx_logistic])
    best_comp_knn = results_dict["Components"][
        best_idx_knn] if "Components" in
        results_dict else (results_dict["PCA
        Components"][best_idx_knn], results_dict["
        LDA Components"][best_idx_knn])

    print(f"\  n    Best Accuracy for {method_name
        }:")
    print(f"   - Mahalanobis: {best_mahalanobis:.4f}
        at components {best_comp_mahal}")
    print(f"   - Logistic Regression: {best_logistic
        :.4f} at components {best_comp_logistic}")
    print(f"   - k-NN: {best_knn:.4f} at components
        {best_comp_knn}")

# Display best results at the end of execution
get_best_accuracy(results_pca, "PCA")
get_best_accuracy(results_lda, "LDA")
get_best_accuracy(results_pca_lda, "PCA -> LDA")
get_best_accuracy(results_lda_pca, "LDA -> PCA")
```

7) Graphical Insight: **PCA**

```
plt.figure(figsize=(12, 8))

plt.plot(results_pca["Components"], results_pca["
    Mahalanobis"], label="Mahalanobis", linestyle='
    dashed')
plt.plot(results_pca["Components"], results_pca["
    Logistic Regression"], label="Logistic
    Regression")
plt.plot(results_pca["Components"], results_pca["k-
    NN"], label="k-NN", linestyle='dotted')
plt.xlabel("Number of PCA Components")
plt.ylabel("Accuracy")
plt.title("PCA: Accuracy vs. Components")
plt.legend()
plt.grid(True)

plt.savefig("./Images/pca_accuracy_vs_components.png
    ", dpi=300)
plt.show()
```

LDA

```
plt.figure(figsize=(12, 8))

plt.plot(results_lda["Components"], results_lda["
    Mahalanobis"], label="Mahalanobis", linestyle='
    dashed')
plt.plot(results_lda["Components"], results_lda["
    Logistic Regression"], label="Logistic
    Regression")
plt.plot(results_lda["Components"], results_lda["k-
    NN"], label="k-NN", linestyle='dotted')
plt.xlabel("Number of LDA Components")
plt.ylabel("Accuracy")
plt.title("LDA: Accuracy vs. Components")
plt.legend()
plt.grid(True)

plt.savefig("./Images/lda_accuracy_vs_components.png
    ", dpi=300)
plt.show()
```

PCA→LDA

```
# Create 3D Plot
pca_lda = plt.figure(figsize=(12, 8))
ax = pca_lda.add_subplot(111, projection='3d')

# Plot each classifier's accuracy in 3D
ax.scatter(results_pca_lda["PCA Components"],
    results_pca_lda["LDA Components"],
    results_pca_lda["Mahalanobis"], label="
    Mahalanobis", alpha=0.6)
ax.scatter(results_pca_lda["PCA Components"],
    results_pca_lda["LDA Components"],
    results_pca_lda["Logistic Regression"], label="
    Logistic Regression", alpha=0.6)
ax.scatter(results_pca_lda["PCA Components"],
    results_pca_lda["LDA Components"],
    results_pca_lda["k-NN"], label="k-NN", alpha
    =0.6)

# Labels and title
ax.set_xlabel("PCA Components")
ax.set_ylabel("LDA Components")
ax.set_zlabel("Accuracy")
ax.set_title("3D Accuracy Plot: PCA + LDA")
ax.legend()

pca_lda.savefig("./Images/pca_lda_3d_accuracy_plot.
    png", dpi=300)
```

LDA→PCA

```
# Create 3D Plot
lda_pca = plt.figure(figsize=(12, 8))
ax = lda_pca.add_subplot(111, projection='3d')

# Plot each classifier's accuracy in 3D
ax.scatter(results_lda_pca["PCA Components"],
    results_lda_pca["LDA Components"],
    results_lda_pca["Mahalanobis"], label="
    Mahalanobis", alpha=0.6)
ax.scatter(results_lda_pca["PCA Components"],
    results_lda_pca["LDA Components"],
    results_lda_pca["Logistic Regression"], label="
    Logistic Regression", alpha=0.6)
ax.scatter(results_lda_pca["PCA Components"],
    results_lda_pca["LDA Components"],
    results_lda_pca["k-NN"], label="k-NN", alpha
    =0.6)

# Labels and title
ax.set_xlabel("PCA Components")
ax.set_ylabel("LDA Components")
ax.set_zlabel("Accuracy")
ax.set_title("3D Accuracy Plot: PCA + LDA")
ax.legend()

lda_pca.savefig("./Images/lda_pca_3d_accuracy_plot.
    png", dpi=300)
```

B. Part 2: Evaluation for Final Model

1) Library Import:

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt

import torch
from torchvision import datasets, transforms

from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix,
    ConfusionMatrixDisplay, f1_score, accuracy_score
    , recall_score, precision_score
import matplotlib.pyplot as plt
```

2) MNIST Dataset Loading:

```
# Load MNIST without normalization
raw_transform = transforms.ToTensor()

train_dataset_raw = datasets.MNIST(root="./data",
    train=True, transform=raw_transform, download=
    True)
test_dataset_raw = datasets.MNIST(root="./data",
    train=False, transform=raw_transform, download=
    True)

# Calculation of mean and standard deviation for
    normalization purpose
X_train_raw = train_dataset_raw.data.numpy().astype(
    'float32') / 255.0
y_train = train_dataset_raw.targets.numpy()
X_test_raw = test_dataset_raw.data.numpy().astype('
    float32') / 255.0
y_test = test_dataset_raw.targets.numpy()

X_train = X_train_raw.reshape(len(X_train_raw), -1)
X_test = X_test_raw.reshape(len(X_test_raw), -1)

mean = X_train.mean()
std = X_train.std()
print(f"Computed␣Mean:␣{mean:.4f},␣Std:␣{std:.4f}")

# Normalized MNIST
normalized_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((mean,), (std,))
])

train_dataset = datasets.MNIST(root="./data", train=
    True, transform=normalized_transform, download=
    False)
test_dataset = datasets.MNIST(root="./data", train=
    False, transform=normalized_transform, download=
    False)

print("Training␣data␣shape:", X_train.shape, "Labels
    ␣shape:", y_train.shape)
print("Test␣data␣shape:", X_test.shape, "Labels␣
    shape:", y_test.shape)
```

3) Function of Computation of PCA:

```
def apply_pca(X_train, X_test, n_components):
    pca = PCA(n_components=n_components)
    return pca.fit_transform(X_train), pca.transform
        (X_test)
```

4) Training:

```
pca = PCA(n_components=31)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_clf = KNeighborsClassifier(n_neighbors=5).fit(
    X_train_pca, y_train)
```

5) Compute of Confusion Matrix and Performance matric:

```
# Predict using the trained classifier
y_pred = knn_clf.predict(X_test_pca)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion␣Matrix")
plt.savefig("./Images/confusion_matrix.png", dpi
    =300, bbox_inches='tight')  # Save as PNG
plt.show()

# Metrics Calculation
f1 = f1_score(y_test, y_pred, average='weighted')
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average='
    weighted')
precision = precision_score(y_test, y_pred, average=
    'weighted')

# Output the results
print(f"F1-score:␣{f1:.4f}")
print(f"Accuracy:␣{accuracy:.4f}")
print(f"Recall:␣{recall:.4f}")
print(f"Precision:␣{precision:.4f}")
```

6) Example of Final Result: **Correct Prediction Result**

```
correct_indices = np.where(y_pred == y_test)[0]

num_samples = min(20, len(correct_indices))
selected_indices = np.random.choice(correct_indices,
     num_samples, replace=False)

rows, cols = 4, 5
plt.figure(figsize=(15, 8))

for i, idx in enumerate(selected_indices):
    plt.subplot(rows, cols, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='
        gray')
    plt.title(f"True:␣{y_test[idx]}\nPred:␣{y_pred[
        idx]}", fontsize=8)
    plt.axis('off')

plt.tight_layout()
plt.savefig("./Images/sample_predictions_true.png",
    dpi=300, bbox_inches='tight')
plt.show()
```

Wrong Prediction Result

```
wrong_indices = np.where(y_pred != y_test)[0]

num_samples = min(10, len(wrong_indices))
selected_indices = np.random.choice(wrong_indices,
    num_samples, replace=False)

rows, cols = 4, 5
plt.figure(figsize=(15, 8))

for i, idx in enumerate(selected_indices):
```

```
    plt.subplot(rows, cols, i + 1)
    plt.imshow(X_test[idx].reshape(28, 28), cmap='
        gray')
    plt.title(f"True: {y_test[idx]}\nPred: {y_pred[
        idx]}", fontsize=8)
    plt.axis('off')

plt.tight_layout()
plt.savefig("./Images/sample_predictions_false.png",
     dpi=300, bbox_inches='tight')
plt.show()
```