### Introduction

While high-dimensional data can indeed carry more information, it also poses significant challenges for traditional machine learning models. High-dimensional datasets may lead to the "curse of dimensionality," where model accuracy decreases due to overfitting, as well as the Hughes phenomenon (Hughes, 1968), where too many features degrade performance. Additionally, the computational complexity grows exponentially with increasing dimensions, making training inefficient.

In this assignment, we applied various dimensionality reduction (DR) techniques, including Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), to three datasets representing different types of data: low-dimensional structured data (Wine dataset), medium-complexity image data (simplified MNIST), and high-dimensional complex colour image data (CIFAR-10). The goal is to investigate how these dimensionality reduction methods impact classification performance across different data complexities, and to understand the trade-offs between retaining dimensionality and improving classification accuracy.

### Methodology

## Datasets

3 datasets were used in this assignment for representing various data types.

* **Wine Dataset**: Contains 13 chemical data and 3 categories of wine, 178 samples.   
  Representing low-dimensional structured data.  
  Suitable for testing basic linear classification models.
* **Simplified MNIST**: Contains 64(8\*8) pixels gray level image data, 1797 samples.  
  Representing medium-complexity image data.  
  Used to test the dimensionality reduction effect of medium-dimensional data
* **CIFAR-10**: Contains 32\*32 pixels RGB image data, 60,000 samples.  
  Representing high-dimensional complex colour image data.  
  Explore dimensionality reduction and classification challenges for complex data.

## Preprocessing

Data loaded and normalized into the range [0,1] to ensure uniform feature scaling, which is important for the effective performance of most machine learning models. The datasets were split into training and test sets using an 80:20 ratio to ensure a robust evaluation of the classifiers' performance while maintaining sufficient training data.

## Dimensionality Reduction and Features Extraction

The Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) methods were used to reduce data dimensionality across the datasets. The CIFAR-10 dataset, due to its high dimensionality and complex data structure, posed a unique challenge, and therefore, a pre-trained ResNet18 model was used for feature extraction. ResNet18 was selected because of its balanced complexity and ability to extract robust features from image data without requiring extensive computational resources.

For each dataset, the appropriate PCA parameters were chosen based on the variance retention criteria. LDA parameters were automatically determined as the minimum of (C-1) and the input dimension.

## Classifiers

To evaluate the impact of dimensionality reduction on classification performance, three classifiers were selected based on their different strengths and applicability to various data types:

* **k-Nearest Neighbors (k-NN)**: A non-parametric classifier that makes predictions based on the closest training examples in the feature space. It is simple and effective for low-dimensional data, making it a good candidate for evaluating the effects of dimensionality reduction on structured and moderately complex datasets like Wine and simplified MNIST.
* **Logistic Regression**: A linear classifier that is efficient for linearly separable data. It is particularly suitable for datasets where the relationships between features can be effectively modeled as linear combinations. Logistic regression helps evaluate the linear separability of the data after dimensionality reduction, especially following LDA.
* **Mahalanobis Distance Classifier**: This classifier calculates the distance between points and class centers using the covariance matrix. It is sensitive to feature correlations and can handle complex feature spaces well, making it ideal for high-dimensional data and post-PCA/LDA transformation.

Each classifier was trained on the reduced-dimensional datasets, and their accuracy and performance were recorded and compared.

## Experimental Procedure

The experimental process follows a system pipeline where different configurations are generated based on a predefined parameter dictionary. The pipeline handles data preprocessing and experiment execution for various DR methods and classifiers. After the classifier predictions, performance metrics including accuracy and time are recorded. The above process is repeated for each dataset and the results are aggregated for further analysis.

A diagram of a computer

Description automatically generated

Figure 1Experiment Flow

### Experimental Results

In this section, we present the experimental results that illustrate the impact of different dimensionality reduction techniques on classification accuracy and time consumption across the three datasets: Wine, Simplified MNIST, and CIFAR-10. Three sets of figures and tables summarize the key findings.

## PCA Parameter Sensitivity

As shown in Figure 2 (see Appendix B for the detailed image) showing how varying PCA parameters (the number of retained components) impacts classification accuracy among datasets for the three classifiers. For each plot, the baseline (no dimensionality reduction) classification accuracy is marked for comparison.

A group of graphs with numbers

Description automatically generated

Figure 2: A brief overview of PCA sensitivity

## LDA Comparison with PCA+LDA

Figure 3 displays three subplots, each representing a dataset. These plots compare the classification accuracy of applying PCA with varying parameters, using LDA-only performance as the baseline.

A graph with red and green lines

Description automatically generated

Figure 3: PCA+LDA v.s. LDA

## CIFAR-10 Results: Comparison of Classifiers and Reduction Methods

Figure 4 (next page) provides a detailed comparison of different classifiers applied to the CIFAR-10 dataset, with and without dimensionality reduction. This figure consists of three subplots, each corresponding to one classifier, showing how PCA, LDA, and PCA+LDA perform relative to the baseline (no dimensionality reduction).

A graph with blue squares

Description automatically generated

Figure 4: Accuracies on Cifar-10

## Classification Time

The tables below present the classification speed (in FPS) before and after dimensionality reduction across all datasets and classifiers. The upper sub-row is FPS without DR applied. Lower sub-row is average FPS with DR applied. These results help illustrate the trade-offs between reduced dimensions and computational efficiency.

Table 1: Samples per Second before and after DR

|  |  |  |  |
| --- | --- | --- | --- |
|  | K-NN | Linear | Mahalanobis |
| Wine | 11494 | 140 | 12987 |
| 18867 | 136 | 19670 |
| MNIST | 1915 | 386 | 505 |
| 8379 | 1194 | 5136 |
| Cifar-10 | 2227 | 212 | T.O. |
| 7692 | 5555 | 733 |

### Discussion

This section analyzes the experimental results, focusing on the effects of different dimensionality reduction methods (PCA, LDA, PCA+LDA) and classifiers (k-NN, Logistic Regression, Mahalanobis Distance) on the three datasets. We will discuss key observations and provide explanations based on the behavior of the classifiers and the characteristics of the datasets.

## PCA

**k-NN and Mahalanobis Classifiers:** For both k-NN and Mahalanobis classifiers, we observe that as the number of dimensions retained by PCA increases, classification accuracy decreases, especially on the MNIST and CIFAR-10 datasets. This can be explained by PCA's focus on maximizing global variance, which can cause the loss of important local structures that carries space information of image. These classifiers are sensitive to the distance between points, and PCA may distort these distances in high-dimensional spaces.

**Logistic Regression:** Unlike the other classifiers, logistic regression shows an improvement in performance as the number of PCA-retained dimensions increases. This is likely because logistic regression benefits from the greater linear separability introduced by PCA in higher dimensions, especially on the more complex CIFAR-10 dataset.

## LDA

**Improved Linear Separability:** LDA significantly improves classification accuracy for logistic regression across all datasets, particularly the Wine dataset, where LDA is highly effective due to the linear separability of the classes. However, on CIFAR-10, LDA’s performance is somewhat limited due to the dataset's complexity and inherent non-linear relationships.

**k-NN and Mahalanobis Performance:** LDA performs better than PCA for k-NN and Mahalanobis classifiers on datasets like Wine and Simplified MNIST. This is because LDA optimizes class separability, which aligns well with the distance-based decision rules used by these classifiers. However, on CIFAR-10, LDA alone does not fully capture the complexity needed for effective classification.

## PCA+LDA

**Combining Methods:** The combination of PCA followed by LDA often outperforms PCA alone, especially for the Mahalanobis classifier on high-dimensional datasets like CIFAR-10. This suggests that reducing the dimensionality first through PCA and then applying LDA to maximize class separability provides a more stable and effective feature space for these classifiers. In addition, the reduction in dimensionality leads to faster matrix operations, which reduces the amount of computation exponentially.

### Conclusion

For low-dimensional structured data, there may be a linear relationship between features, and there may be some redundancy in the feature dimensions. Therefore, when PCA is used to reduce the dimension to a certain range, it is possible to retain enough information for the classifier to classify while reducing the noise impact caused by redundant features (Martinez & Kak, 2001), especially for K-NN and Mahalanobis distance classifiers that are sensitive to high-dimensional noise.

For high-dimensional image data with complex structures, PCA/LDA or hybrid methods may destroy the local fine structural features of the image due to their global view and assumption of linear relationship between features. For complex abstract features, using deep learning, such as CNN, to extract features can obtain more detailed and comprehensive results.

In addition, it is undeniable that by reducing the dimensionality of the data, the calculation dimension can be effectively reduced, and the calculation process of some classifiers can be effectively accelerated, especially the Mahalanobis distance classifier that requires the calculation of the inverse matrix.

# References:

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 55-63.

Martinez, A., & Kak, A. (2001). PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 228 - 233.

**Appendix A: Code**Also seen at: <https://github.com/AlvisWSY/EE6222_Assignment1/blob/main/code/Assignment.py>

import os

import time

import csv

import logging

import warnings

import numpy as np

from scipy.spatial.distance import cdist

from sklearn import datasets

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

from sklearn.exceptions import ConvergenceWarning

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from torch.utils.data import DataLoader

import torch

import torchvision

import torchvision.transforms as transforms

# Configure logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s: %(message)s')

# Ignore convergence warnings

warnings.filterwarnings("ignore", category=ConvergenceWarning)

path = '/media/user/volume2/students/s124md209\_01/WangShengyuan/6222/Assignment1/'

# Constants

RESULTS\_CSV\_PATH = os.path.join(path, 'result/results.csv')

DATA\_DIR = os.path.join(path, 'data')

# Cache for loaded datasets

\_cached\_data = {}

def main():

"""

Main function to run all experiments.

"""

initialize\_csv(RESULTS\_CSV\_PATH)

config = {

'mnist': {'n\_components\_list': [20, 30, 40, 50, 0.95]},

'wine': {'n\_components\_list': [2, 3, 5, 0.95]},

'cifar10': {'n\_components\_list': [50, 100, 200, 0.95]}

}

classifiers = ['knn', 'linear', 'mahalanobis']

# Generate all experiments

experiments = []

for dataset\_name, settings in config.items():

experiments.extend(generate\_experiments(dataset\_name, settings['n\_components\_list'], classifiers))

# Include CNN feature extraction and classification for CIFAR-10

if 'cifar10' in config:

cnn\_experiments = generate\_cnn\_experiments('cifar10', classifiers)

experiments.extend(cnn\_experiments)

# Run all experiments

for experiment in experiments:

logging.info(f"Starting experiment: {experiment}")

try:

pipeline(experiment)

except Exception as e:

logging.error(f"Error in experiment {experiment}: {e}")

logging.info(f"Finished experiment: {experiment}")

def generate\_experiments(dataset\_name, n\_components\_list, classifiers):

"""

Generate experiment configurations for a given dataset.

"""

experiments = []

# Define reduction methods and their corresponding n\_components

reduction\_methods = {

'none': [None],

'lda': [None],

'pca': n\_components\_list,

'pca+lda': n\_components\_list

}

for reduction\_method, n\_components\_values in reduction\_methods.items():

for n\_components in n\_components\_values:

for classifier\_name in classifiers:

experiments.append({

'dataset\_name': dataset\_name,

'reduction\_method': reduction\_method,

'classifier\_name': classifier\_name,

'n\_components': n\_components

})

return experiments

def generate\_cnn\_experiments(dataset\_name, classifiers):

"""

Generate experiments using CNN features for traditional classifiers.

"""

experiments = []

for classifier\_name in classifiers:

experiments.append({

'dataset\_name': dataset\_name,

'reduction\_method': 'cnn\_features',

'classifier\_name': classifier\_name,

'n\_components': None

})

return experiments

def initialize\_csv(csv\_file):

"""

Initialize the CSV file with headers.

"""

os.makedirs(os.path.dirname(csv\_file), exist\_ok=True)

with open(csv\_file, mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerow([

'Dataset',

'Reduction Method',

'Parameters',

'Explained Variance',

'Number of Components',

'Classifier',

'Accuracy',

'Reduction Time per Sample',

'Classification Time per Sample'

])

logging.info(f"Initialized CSV file at {csv\_file}")

def pipeline(params):

"""

Pipeline to run the experiment with the given parameters.

If CNN -> Extract features, else load and preprocess data.

Apply dimensionality reduction -> Classify Data -> Write Results to CSV.

"""

dataset\_name = params['dataset\_name']

reduction\_method = params['reduction\_method']

classifier\_name = params['classifier\_name']

n\_components = params['n\_components']

logging.info(f"Pipeline started for dataset: {dataset\_name}, reduction: {reduction\_method}, classifier: {classifier\_name}, n\_components: {n\_components}")

# Handle CNN feature extraction

if reduction\_method == 'cnn\_features':

# Load data and extract CNN features

X\_train, X\_test, y\_train, y\_test = extract\_cnn\_features(dataset\_name)

# Classify using traditional classifiers

accuracy, classification\_time = classify\_data(

classifier\_name, X\_train, y\_train, X\_test, y\_test, dataset\_name, reduction\_method

)

classification\_avg\_time = 'NA' if accuracy == 'NA' else f"{classification\_time / X\_test.shape[0]:.6f}s"

# Write results to CSV

write\_to\_csv(RESULTS\_CSV\_PATH, [

dataset\_name,

reduction\_method,

'N/A',

'N/A',

X\_train.shape[1],

classifier\_name,

accuracy,

'N/A',

classification\_avg\_time

])

logging.info(f"Dataset: {dataset\_name}, Reduction: {reduction\_method}, "

f"Classifier: {classifier\_name}, Accuracy: {accuracy}, "

f"Classification time per sample: {classification\_avg\_time}")

return

# Load and preprocess data

X\_train, X\_test, y\_train, y\_test = load\_and\_preprocess\_data(dataset\_name)

if reduction\_method == 'none':

# Classify without dimensionality reduction

accuracy, total\_time = classify\_data(

classifier\_name, X\_train, y\_train, X\_test, y\_test, dataset\_name, reduction\_method

)

avg\_time\_per\_sample = 'NA' if accuracy == 'NA' else f"{total\_time / X\_test.shape[0]:.6f}s"

write\_to\_csv(RESULTS\_CSV\_PATH, [

dataset\_name,

'none',

'N/A',

'N/A',

X\_train.shape[1],

classifier\_name,

accuracy,

'N/A',

avg\_time\_per\_sample

])

logging.info(f"Dataset: {dataset\_name}, Classifier: {classifier\_name}, "

f"Accuracy: {accuracy}, Time per sample: {avg\_time\_per\_sample}")

return

# Apply dimensionality reduction

X\_train\_reduced, X\_test\_reduced, explained\_variance, reduction\_avg\_time = apply\_dimensionality\_reduction(

reduction\_method, X\_train, X\_test, y\_train, n\_components

)

# After dimensionality reduction

scaler = StandardScaler()

X\_train\_reduced = scaler.fit\_transform(X\_train\_reduced)

X\_test\_reduced = scaler.transform(X\_test\_reduced)

# Classify reduced data

accuracy, classification\_time = classify\_data(

classifier\_name, X\_train\_reduced, y\_train, X\_test\_reduced, y\_test, dataset\_name, reduction\_method

)

classification\_avg\_time = 'NA' if accuracy == 'NA' else f"{classification\_time / X\_test\_reduced.shape[0]:.6f}s"

# Write results to CSV

write\_to\_csv(RESULTS\_CSV\_PATH, [

dataset\_name,

reduction\_method,

n\_components,

explained\_variance,

X\_train\_reduced.shape[1],

classifier\_name,

accuracy,

f"{reduction\_avg\_time:.6f}s",

classification\_avg\_time

])

logging.info(f"Dataset: {dataset\_name}, Reduction: {reduction\_method}, "

f"Classifier: {classifier\_name}, Accuracy: {accuracy}, "

f"Reduction time per sample: {reduction\_avg\_time:.6f}s, "

f"Classification time per sample: {classification\_avg\_time}")

def load\_and\_preprocess\_data(dataset\_name):

"""

Load and preprocess data for the given dataset.

"""

if dataset\_name in \_cached\_data:

return \_cached\_data[dataset\_name]

if dataset\_name == 'mnist':

X, y = load\_mnist()

elif dataset\_name == 'wine':

X, y = load\_wine()

elif dataset\_name == 'cifar10':

X, y = load\_cifar10()

else:

raise ValueError("Unsupported dataset. Choose from 'mnist', 'wine', 'cifar10'.")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

\_cached\_data[dataset\_name] = (X\_train, X\_test, y\_train, y\_test)

logging.info(f"Loaded and preprocessed dataset: {dataset\_name}")

return X\_train, X\_test, y\_train, y\_test

def load\_mnist():

"""

Load and preprocess the MNIST dataset.

"""

from sklearn.datasets import load\_digits

digits = load\_digits()

X = digits.data.astype('float32') / 16.0 # Normalize to [0, 1]

y = digits.target

return X, y

def load\_wine():

"""

Load and preprocess the Wine dataset.

"""

wine = datasets.load\_wine()

X = wine.data

y = wine.target

scaler = StandardScaler()

X = scaler.fit\_transform(X)

return X, y

def load\_cifar10():

"""

Load and preprocess the CIFAR-10 dataset for traditional classifiers.

"""

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

dataset = torchvision.datasets.CIFAR10(

root=DATA\_DIR, train=True, download=True, transform=transform

)

dataloader = DataLoader(dataset, batch\_size=len(dataset), shuffle=False)

X, y = next(iter(dataloader))

X = X.view(len(dataset), -1).numpy()

y = y.numpy()

X = X.astype('float32') / 2.0 + 0.5 # Normalize to [0, 1]

return X, y

def extract\_cnn\_features(dataset\_name):

"""

Extract features from CNN for CIFAR-10 dataset.

ResNet-18 model is used for feature extraction.

"""

if dataset\_name != 'cifar10':

raise ValueError("CNN feature extraction is only implemented for CIFAR-10.")

# Define transformations

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

# Load datasets

trainset = torchvision.datasets.CIFAR10(

root=DATA\_DIR, train=True, download=True, transform=transform

)

testset = torchvision.datasets.CIFAR10(

root=DATA\_DIR, train=False, download=True, transform=transform

)

# Data loaders

trainloader = DataLoader(trainset, batch\_size=64, shuffle=False)

testloader = DataLoader(testset, batch\_size=64, shuffle=False)

# Define CNN model (using a pretrained model)

model = torchvision.models.resnet18(pretrained=True)

model = model.to('cuda:0' if torch.cuda.is\_available() else 'cpu')

model.eval()

# Remove the final classification layer

features = torch.nn.Sequential(\*list(model.children())[:-1])

# Function to extract features

def get\_features(loader):

X\_features = []

y\_labels = []

with torch.no\_grad():

for inputs, labels in loader:

inputs = inputs.to('cuda' if torch.cuda.is\_available() else 'cpu')

outputs = features(inputs)

outputs = outputs.view(outputs.size(0), -1)

X\_features.append(outputs.cpu().numpy())

y\_labels.append(labels.numpy())

X\_features = np.concatenate(X\_features, axis=0)

y\_labels = np.concatenate(y\_labels, axis=0)

return X\_features, y\_labels

# Extract features

X\_train, y\_train = get\_features(trainloader)

X\_test, y\_test = get\_features(testloader)

return X\_train, X\_test, y\_train, y\_test

def apply\_dimensionality\_reduction(method, X\_train, X\_test, y\_train, n\_components):

"""

Apply the specified dimensionality reduction method to the data.

"""

start\_time = time.time()

explained\_variance = 'N/A'

if method == 'pca':

X\_train\_reduced, X\_test\_reduced, explained\_variance = apply\_pca(

X\_train, X\_test, n\_components

)

elif method == 'lda':

X\_train\_reduced, X\_test\_reduced = apply\_lda(X\_train, X\_test, y\_train)

elif method == 'pca+lda':

X\_train\_pca, X\_test\_pca, explained\_variance = apply\_pca(X\_train, X\_test, n\_components)

X\_train\_reduced, X\_test\_reduced = apply\_lda(X\_train\_pca, X\_test\_pca, y\_train)

else:

raise ValueError("Unsupported reduction method. Choose from 'pca', 'lda', 'pca+lda'.")

reduction\_time = time.time() - start\_time

reduction\_avg\_time = reduction\_time / (X\_train.shape[0] + X\_test.shape[0])

logging.info(f"Applied {method} reduction in {reduction\_time:.2f}s")

return X\_train\_reduced, X\_test\_reduced, explained\_variance, reduction\_avg\_time

def apply\_pca(X\_train, X\_test, n\_components):

"""

Apply PCA to the data.

"""

pca = PCA(n\_components=n\_components)

X\_train\_reduced = pca.fit\_transform(X\_train)

X\_test\_reduced = pca.transform(X\_test)

explained\_variance = np.sum(pca.explained\_variance\_ratio\_)

logging.info(f"PCA explained variance: {explained\_variance:.2f}")

return X\_train\_reduced, X\_test\_reduced, explained\_variance

def apply\_lda(X\_train, X\_test, y\_train):

"""

Apply LDA to the data.

"""

n\_classes = len(np.unique(y\_train))

lda\_n\_components = min(n\_classes - 1, X\_train.shape[1])

lda = LDA(n\_components=lda\_n\_components)

X\_train\_reduced = lda.fit\_transform(X\_train, y\_train)

X\_test\_reduced = lda.transform(X\_test)

logging.info(f"LDA reduced data to {lda\_n\_components} components")

return X\_train\_reduced, X\_test\_reduced

def classify\_data(classifier\_name, X\_train, y\_train, X\_test, y\_test, dataset\_name, reduction\_method):

"""

Train and evaluate the specified classifier.

"""

start\_time = time.time()

try:

if classifier\_name == 'knn':

logging.info(f"Training KNN classifier on dataset: {dataset\_name} with reduction: {reduction\_method}")

classifier = KNeighborsClassifier(n\_neighbors=3)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

elif classifier\_name == 'linear':

logging.info(f"Training LogisticRegression on dataset: {dataset\_name} with reduction: {reduction\_method}")

# Adjust parameters for high-dimensional data

if (dataset\_name == 'cifar10') and (reduction\_method == 'none' or X\_train.shape[1] > 500):

logging.info("Using 'saga' solver for high-dimensional data")

# Limit training data size to speed up

max\_samples = 5000

if X\_train.shape[0] > max\_samples:

logging.info(f"Reducing training samples to {max\_samples}")

X\_train = X\_train[:max\_samples]

y\_train = y\_train[:max\_samples]

classifier = LogisticRegression(

max\_iter=100,

solver='saga',

tol=1e-2,

multi\_class='multinomial',

n\_jobs=-1

)

else:

logging.info("Using default 'lbfgs' solver")

classifier = LogisticRegression(

max\_iter=200,

solver='lbfgs',

multi\_class='auto',

n\_jobs=-1

)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

elif classifier\_name == 'mahalanobis':

logging.info(f"Training Mahalanobis classifier on dataset: {dataset\_name} with reduction: {reduction\_method}")

# Limit sample size and dimensions for Mahalanobis classifier on CIFAR-10

if dataset\_name == 'cifar10':

if X\_train.shape[1] > 50:

logging.warning("Data dimension too high for Mahalanobis classifier on CIFAR-10. Skipping.")

return 'NA', 'NA'

else:

max\_samples = 5000 # Adjust based on your computational resources

if X\_train.shape[0] > max\_samples:

logging.warning(f"Reducing training samples to {max\_samples} for Mahalanobis classifier.")

X\_train = X\_train[:max\_samples]

y\_train = y\_train[:max\_samples]

y\_pred = mahalanobis\_classifier(X\_train, y\_train, X\_test)

else:

raise ValueError("Unsupported classifier. Choose from 'knn', 'linear', 'mahalanobis'.")

accuracy = accuracy\_score(y\_test, y\_pred)

total\_time = time.time() - start\_time

logging.info(f"Classifier {classifier\_name} achieved accuracy {accuracy:.4f} in {total\_time:.2f}s")

return accuracy, total\_time

except Exception as e:

logging.error(f"Error in classification with {classifier\_name}: {e}")

return 'NA', 'NA'

def mahalanobis\_classifier(X\_train, y\_train, X\_test):

"""

Classify using the Mahalanobis distance.

"""

cov\_matrix = np.cov(X\_train, rowvar=False)

inv\_cov\_matrix = np.linalg.pinv(cov\_matrix)

X\_train\_centered = X\_train - np.mean(X\_train, axis=0)

X\_test\_centered = X\_test - np.mean(X\_train, axis=0)

distances = cdist(

X\_test\_centered, X\_train\_centered, metric='mahalanobis', VI=inv\_cov\_matrix

)

y\_pred = y\_train[np.argmin(distances, axis=1)]

return y\_pred

def write\_to\_csv(csv\_file, row\_data):

"""

Write a row of data to the CSV file.

"""

with open(csv\_file, mode='a', newline='') as file:

writer = csv.writer(file)

writer.writerow(row\_data)

logging.debug(f"Wrote data to CSV: {row\_data}")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Appendix B: Detailed Image for Figure 2**

**A group of graphs with numbers

Description automatically generated**