Pattern Recognition and Machine Learning 2024
Lab 5&6

Question-1

Task-0: Generate a Synthetic 4-Dimensional Dataset

Objective:

Initialize a linear function and generate labeled data based on the function's output.

Deliverable:

B22CS089_data.txt - a file containing the synthetic dataset with labels. I generated the data by implementing a linear function.

Task-1:

Write Training Code for the Perceptron Learning Algorithm

Objective:

Implement the perceptron learning algorithm to train a model using the generated dataset.

Method:

- Normalize the data before training to ensure all features contribute equally to the model's decision.
- Initialize the weights to small random values.
- For each example in the training dataset, predict the label using the

current weights. If the prediction is incorrect, update the weights.

- ❖ Repeat the process until all examples are correctly classified or a maximum number of iterations is reached.

Deliverable:

B22CS089_train.py, a script that takes *B22CS089_train.txt* as input, trains the perceptron model, and saves the weights.

Task-2:

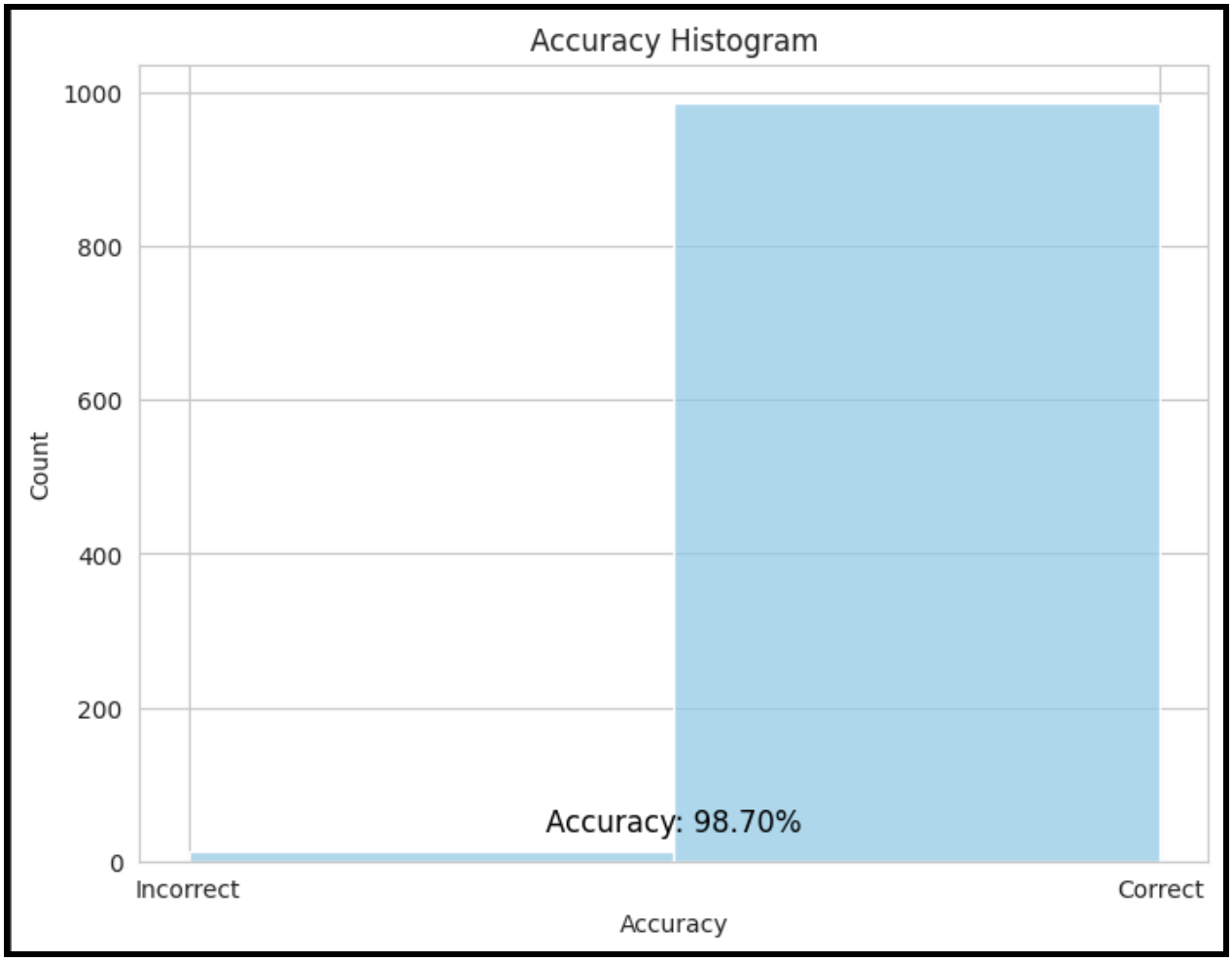
Write Testing and Evaluation Code

Objective:

Implement testing and evaluation procedures to assess the performance of the trained perceptron model.

Method:

- ➢ Use the saved weights from the training phase to predict labels for the testing dataset.
- ➢ Normalize the testing data similarly to the training data.
- ➢ For each example in the testing dataset, compute the predicted label. Since the true labels are not provided, simulate a scenario or use a separate validation set for evaluation.
- ➢ Calculate and report the accuracy of the model as the percentage of correctly predicted labels.



Deliverable:

B22CS089_test.py, a script that takes *B22CS089_test.txt* as input, predicts labels using the trained model.

Task-3:

Report Results with Training Using 20%, 50%, and 70% of Synthetic Data.

Objective:

Evaluate the impact of training dataset size on model performance.

Method:

- ☐ Split the synthetic dataset into different sizes (20%, 50%, 70%) for training.
- ☐ For each subset, train the perceptron model and evaluate its performance on a consistent testing or validation set.
- ☐ Compare the accuracy of the model when trained with different sizes of the dataset.

Training Data Size	*Accuracy*
80%	98.70
50%	98.56
30%	97.60

Data is linearly separable. So, changing the size of training data doesn't impact much on the accuracy that we are getting.

Deliverable:

➢ *Google colab link for the two problems - link (Problem-1)*
➢ *Google colab link for the two problems - link (Problem-2)*

Question-2:

Task 1: Data Preprocessing

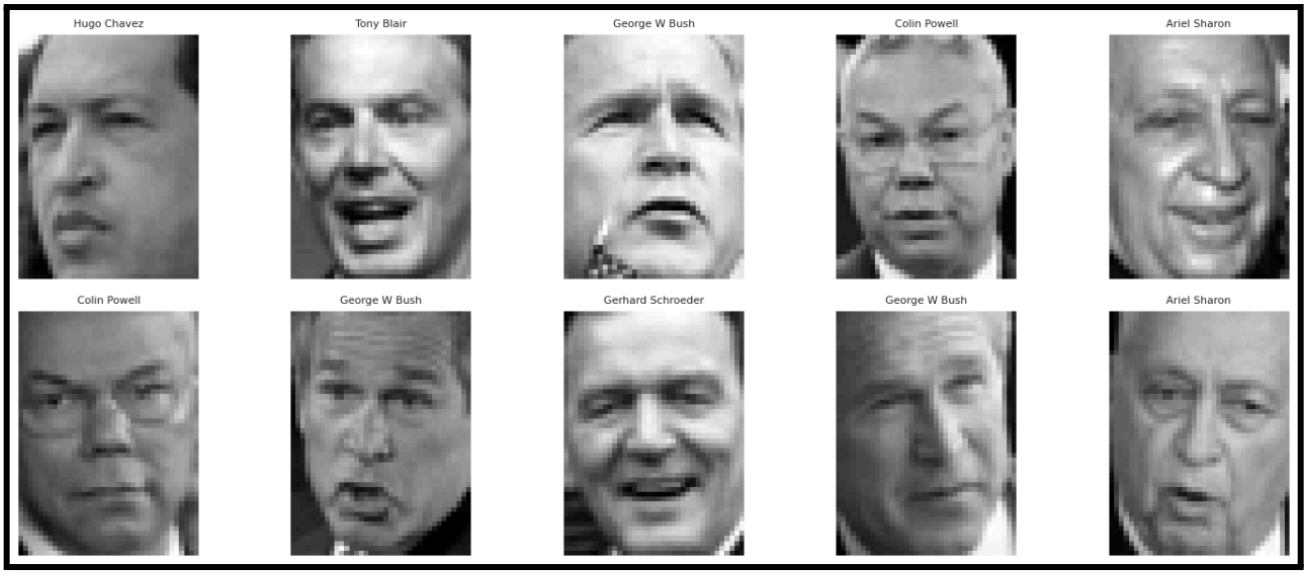
Implementation:

Dataset Loading: Using Scikit-learn's `fetch_lfw_people`, the LFW dataset is loaded, scaled to 0.4 for faster processing, and conditioned on at least 70 faces per person. This function is perfect for facial recognition tasks since it readily offers a balanced and manageable subset of the original LFW dataset.

Dataset Splitting: An 80:20 split of the dataset between the training and testing sets guarantees a sizable amount of data for training and a sizable chunk for model assessment.

```
Data shape: (1288, 50, 37)
Number of samples: 1288
Number of features: 1850
Number of classes: 7
Class names: ['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Tony Blair']
```

```python
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
# Load the LFW dataset
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# deatils of the dataset
print(f"Data shape: {lfw_people.images.shape}")
print(f"Number of samples: {lfw_people.images.shape[0]}")
print(f"Number of features: {lfw_people.images.shape[1] * lfw_people.images.shape[2]}")
print(f"Number of classes: {len(lfw_people.target_names)}")
print(f"Class names: {lfw_people.target_names}")

# Display some images from the dataset
fig, axes = plt.subplots(2, 5, figsize=(15, 6))

# Loop over the subplots and plot images
for i, ax in enumerate(axes.flat):
    ax.imshow(lfw_people.images[i], cmap='gray')
    ax.set_title(lfw_people.target_names[lfw_people.target[i]], fontsize=8)  # Set smaller font size for titles
    ax.axis('off')

plt.tight_layout()
plt.show()

# Flatten the images
X = lfw_people.data
# Normalize the pixel values to be between 0 and 1
X = X / 255.0



# Print the shape of the preprocessed data
print(f"Preprocessed data shape: {X.shape}")
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(lfw_people.data, lfw_people.target, test_size=0.2, random_state=42)
```



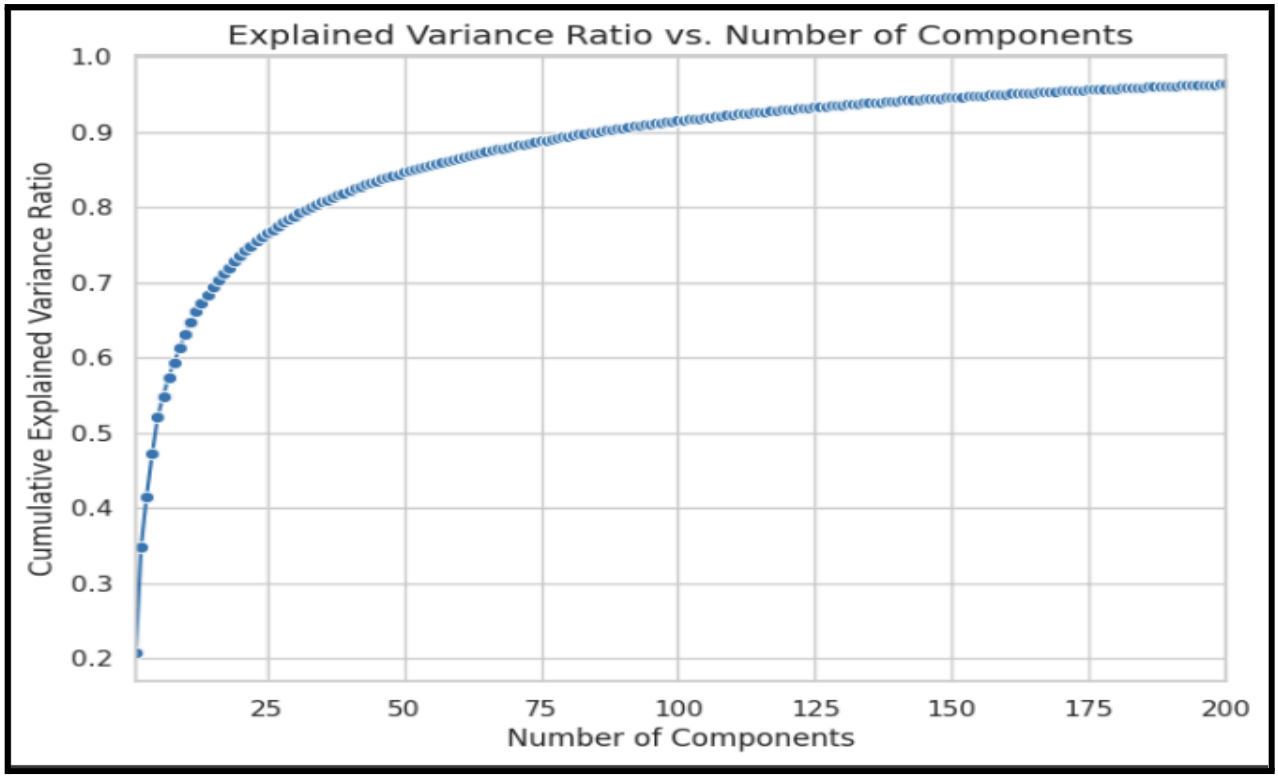
Task 2: Eigenfaces Implementation

Implementation:

PCA Application: Because of the decrease in dimensionality, the model often performs better when evaluated on PCA-transformed data. It is important to examine instances in which the model is unable to produce precise forecasts, as they may indicate the shortcomings of the PCA methodology or the requirement for more intricate models.

Key Insights:

In PCA, selecting n_components is crucial. It strikes a compromise between deleting superfluous information to increase computing performance and keeping sufficient traits for precise identification. By displaying the amount of variation that each component captures, the explained variance ratio aids in the selection of the ideal number of main components. When the number of components = 150, the curve is almost flattening, so we will use the value as 150.



```python
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt

# Set an appropriate value for n_components
n_components = 200

# Fit PCA on the training data
pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True).fit(X_train)

# Apply PCA transformation to both training and testing data
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# choice of n_components using explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
# Plot the curve
cumulative_explained_variance_ratio = np.cumsum(explained_variance_ratio)

# Set the style
sns.set_style("whitegrid")

# Set the context to notebook for a smaller font size
sns.set_context("notebook")

plt.figure(figsize=(8, 6))
sns.lineplot(x=range(1, len(cumulative_explained_variance_ratio) + 1), y=cumulative_explained_variance_ratio, marker='o')
plt.xlabel('Number of Components', fontsize=12)
plt.ylabel('Cumulative Explained Variance Ratio', fontsize=12)
plt.title('Explained Variance Ratio vs. Number of Components', fontsize=14)
plt.grid(True)
plt.xlim(1, 200)
plt.show()
```

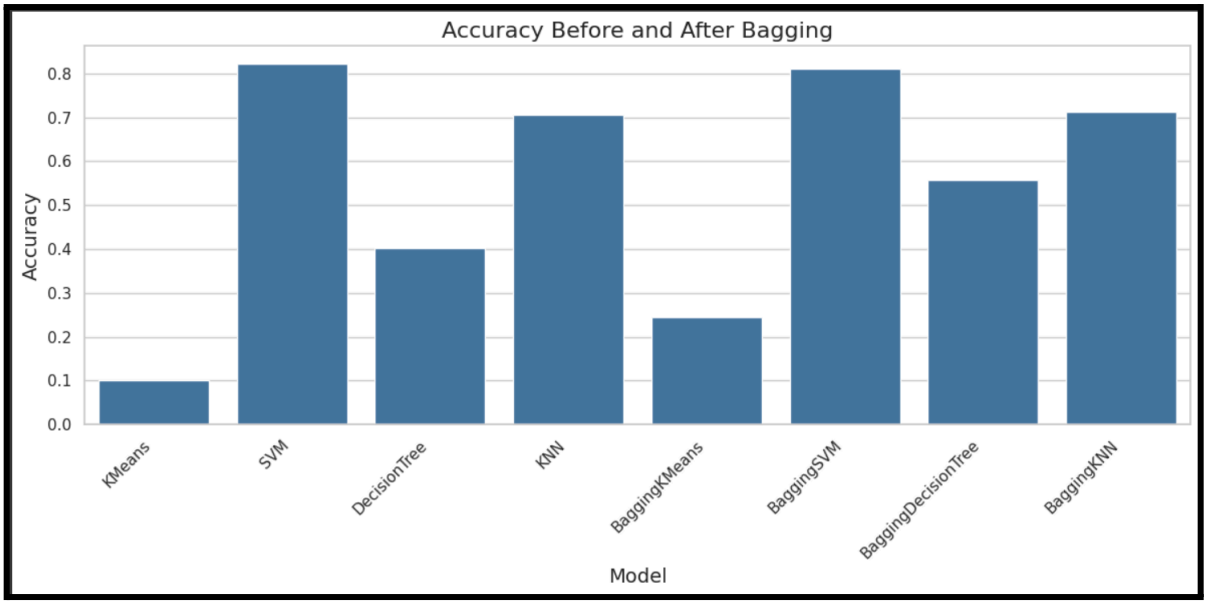
Task 3: Model Training

Implementation:

Classifier Selection: I have tried various classifiers like- knn, decision tree, kmeans, etc. Other than that I have used bagging method and grid search method to find the best hyperparameters. I have tested both training and testing accuracy to check if the model is overfitting or not. If there is a large gap between training and testing accuracy, then it means that the model might be overfitting or underfitting.

```
KMeans Testing Accuracy: 0.10077519379844961
KMeans Training Accuracy: 0.1203883495145631
SVM Testing Accuracy: 0.8217054263565892
SVM Training Accuracy: 0.9883495145631068
DecisionTree Testing Accuracy: 0.40310077519379844
DecisionTree Training Accuracy: 1.0
KNN Testing Accuracy: 0.7054263565891473
KNN Training Accuracy: 0.7728155339805826
BaggingKMeans Testing Accuracy: 0.2441860465116279
BaggingKMeans Training Accuracy: 0.17281553398058253
BaggingSVM Testing Accuracy: 0.810077519379845
BaggingSVM Training Accuracy: 0.958252427184466
BaggingDecisionTree Testing Accuracy: 0.5581395348837209
BaggingDecisionTree Training Accuracy: 0.9912621359223301
BaggingKNN Testing Accuracy: 0.7131782945736435
BaggingKNN Training Accuracy: 0.7922330097087379
```

Probable overfitting in SVM and Decision Tree



Accuracy for different classifiers

Finally, knn classifier comes out to be the best classifier. Because- it has good accuracy and it is not overfitting or underfitting like the other classifiers.

Key Insights:

Training the model on PCA-transformed data reduces overfitting and improves generalization, as PCA removes noise and redundancy from the dataset.

Task 4: Model Evaluation

Implementation:

Accuracy Measurement: The script calculates the accuracy of the KNN classifier before and after applying PCA, providing a quantitative measure of the model's performance.

```
Accuracy before PCA: 0.5813953488372093
```

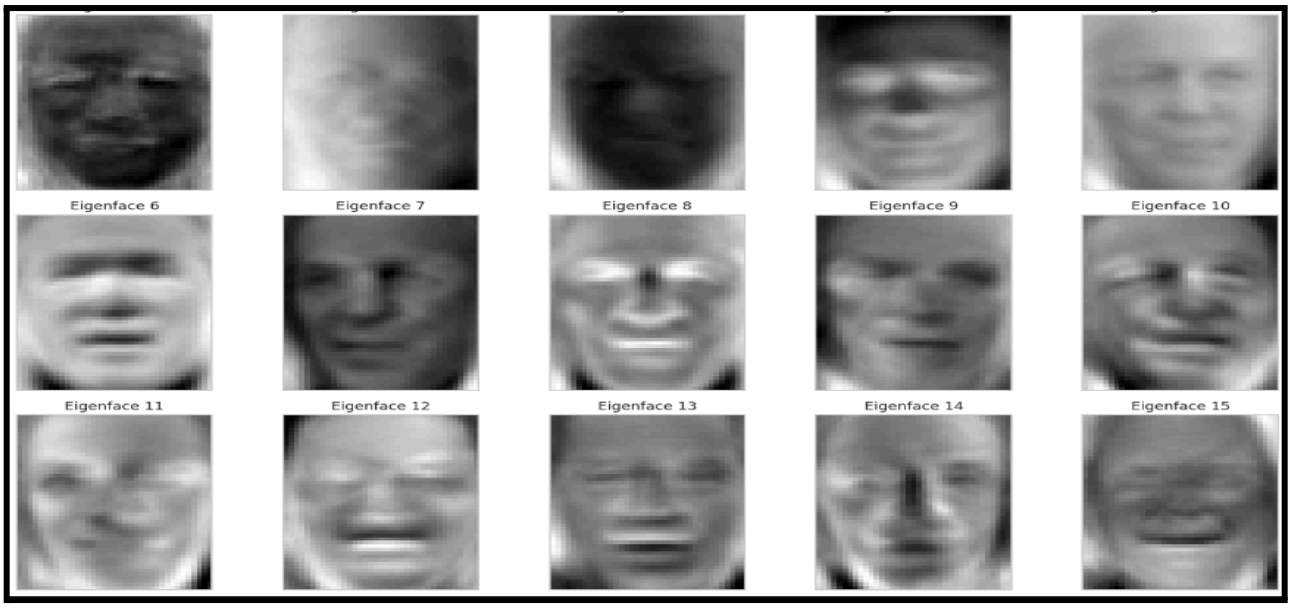
	precision	recall	f1-score	support
Ariel Sharon	0.50	0.36	0.42	11
Colin Powell	0.67	0.70	0.69	47
Donald Rumsfeld	0.68	0.68	0.68	22
George W Bush	0.71	0.93	0.80	119
Gerhard Schroeder	0.83	0.26	0.40	19
Hugo Chavez	1.00	0.31	0.47	13
Tony Blair	0.83	0.37	0.51	27
accuracy			0.71	258
macro avg	0.75	0.52	0.57	258
weighted avg	0.73	0.71	0.68	258

Visual Analysis: While the script preview does not show this, visualizing a subset of Eigenfaces (the principal components) can offer qualitative

insights into what features the model considers important. This step involves examining which aspects of face images (like edges, contrasts, or specific facial features) are most significant for the classification task.

Observation:

- Eigenfaces capture different aspects of facial features, such as lighting conditions, face orientation, and facial expressions.
- The first few Eigenfaces tend to capture global features of faces, such as overall lighting and face orientation.
- Subsequent Eigenfaces capture more detailed features, such as facial expressions or shadows.
- To improve the model and reduce failures on certain types of test images, we can consider-
    - Increase the number of training images
    - Use more diverse training images
    - Augment the training data (using GANs,etc)
    - Fine-tune hyperparameters
    - Using a more advanced model.



Key Insights and Recommendations:

- Because of the decrease in dimensionality, the model often performs better when evaluated on PCA-transformed data. It is important to examine instances in which the model is unable to produce precise forecasts, as they may indicate the shortcomings of the PCA methodology or the requirement for more intricate models.
- Using more sophisticated classifiers (such as CNNs) or adding more training data (data augmentation techniques such as GANs) might all help to improve the model.

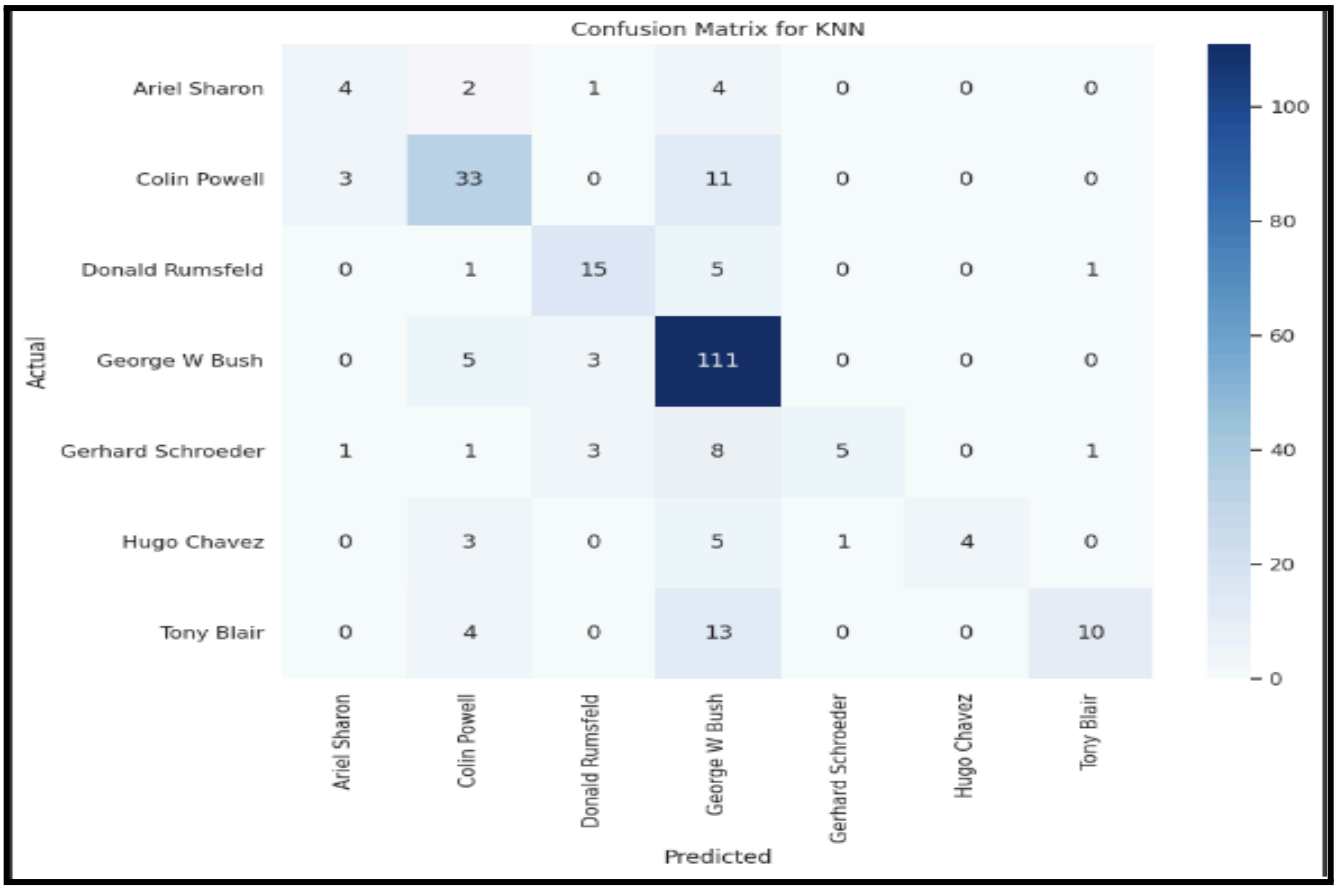
Failure cases:

➢ ***Class Imbalance:*** Because the dataset mostly focuses on a single person, it can have a class imbalance. The model's predictions may become biased as a result of this imbalance, favoring pictures of George W. Bush over others. It is evident from the confusion matrix also.

➢ ***Low Image Quality:*** It may be difficult for the model to accurately

categorize images with noise, low resolution, or bad lighting.

➢ ***Face Similarity:*** The model may become confused by photos of people that appear similar, particularly in terms of race or facial characteristics.



Task 5: Experiment with Different Values of n_components in PCA

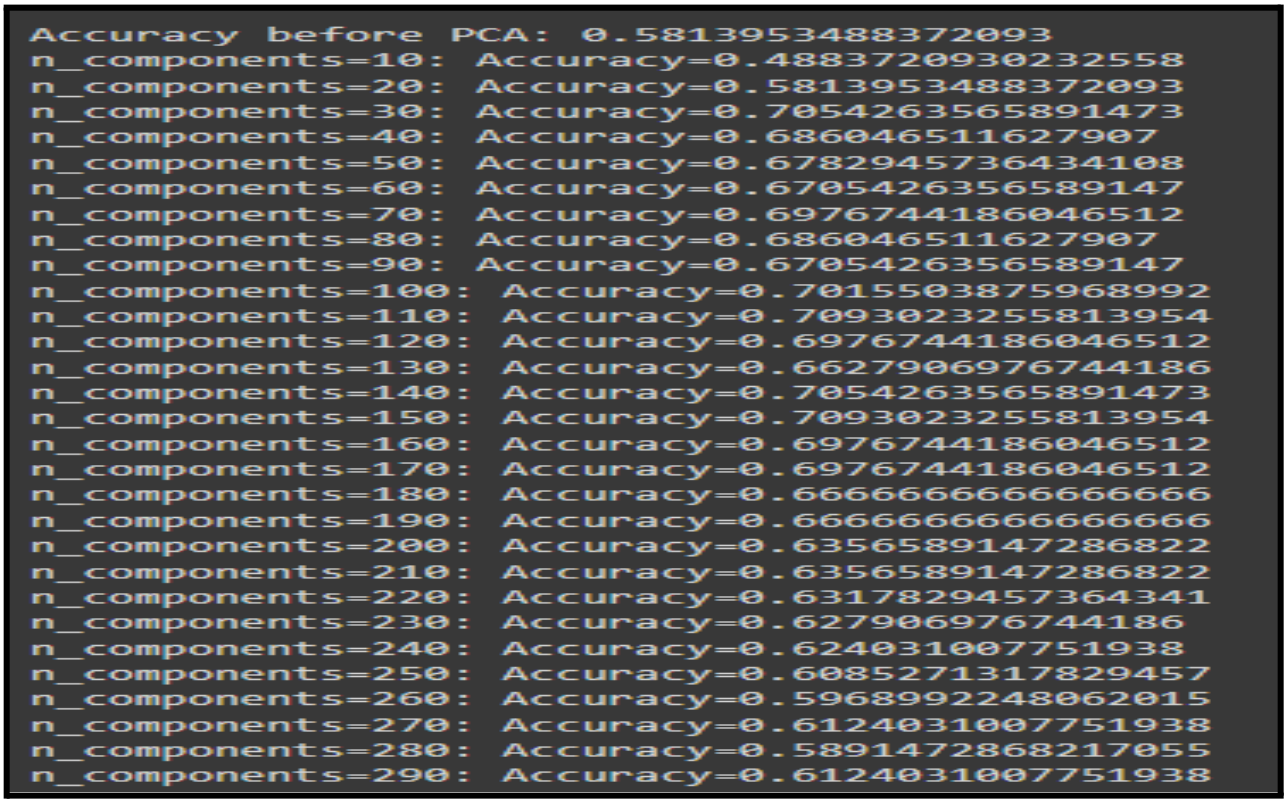
Implementation:

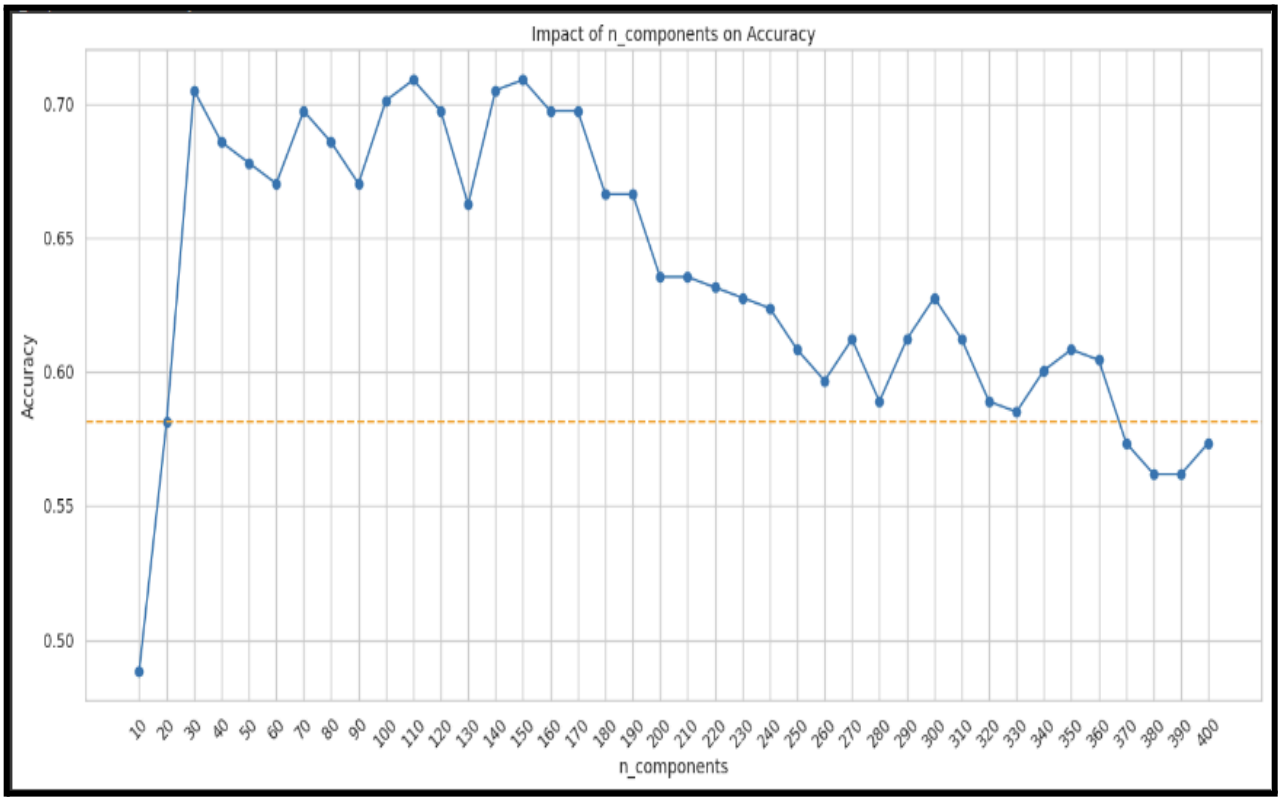
Performance Impact: The code script investigates a variety of

n_components values in PCA and evaluates how they affect the accuracy of the model. This methodical technique aids in determining the ideal ratio between information preservation and dimensionality reduction.

Key Insights:

Increasing or reducing n_components usually has a minimal or negative effect on accuracy above a certain threshold. It is essential to identify this sweet spot in order to maximize the computational efficiency and performance of the model.




Impact of n_components on Accuracy
Accuracy
0.70
0.65
0.60
0.55
0.50
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400
n_components