**Face Recognition Using Random Forrest Classifier**

**Introduction**

This report provides a detailed explanation of a Python script designed to load, preprocess, and classify data using a Random Forest Classifier. The script also evaluates the model's performance and saves it for future use. The following sections describe each step in the script.

**Face Recognition Process**

**How It Works**

**Data Representation**: Each image is represented by a vector of pixel intensity values.

**Labeling**: Each image is labeled with a class/person number, identifying the person in the image.

**Training**: The Random Forest Classifier learns patterns in the pixel intensity values that correspond to different persons during the training phase.

**Prediction**: For new images, the trained model predicts the person based on the learned patterns.

**Evaluation**: The model's performance is evaluated to ensure it accurately recognizes faces.

**Advantages of Random Forest for Face Recognition**

**Ensemble Learning**: Combines multiple decision trees to improve classification accuracy and robustness.

**Overfitting Reduction**: By averaging multiple decision trees, the model reduces the risk of overfitting.

**Feature Importance**: Identifies important features (pixel intensities) that contribute most to the classification.

**Step-by-Step Explanation**

**1. Importing Necessary Libraries**

The script starts by importing essential libraries for data manipulation, machine learning, and model evaluation.

import numpy as np  
import pandas as pd  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, classification\_report  
import joblib

**NumPy**: Used for numerical operations and handling arrays.

**Pandas**: Utilized for data manipulation and analysis.

**Scikit-learn**: Provides tools for machine learning, including the Random Forest algorithm and evaluation metrics.

**Joblib**: Used for saving the trained model to a file.

**2. Loading Data**

The load\_data function reads data from a file, processes it, and returns it as a NumPy array along with the number of features.

def load\_data(file\_path):  
 with open(file\_path, 'r') as file:  
 lines = file.readlines()  
  
 L = int(lines[0].strip())  
 N = int(lines[1].strip())  
 data = []  
  
 for line in lines[2:]:  
 values = list(map(float, line.strip().split()))  
 data.append(values)  
  
 return np.array(data), N

**File Reading**: Opens the file and reads all lines.

**Extracting Parameters**: The first two lines provide metadata (L and N).

**Data Processing**: Subsequent lines contain data values that are processed into a list of float numbers.

**Returning Data**: The function returns the data as a NumPy array and the number of features (N).

**3. Preprocessing Data**

The preprocess\_data function separates features and labels from the dataset.

def preprocess\_data(data, N):  
 X = data[:, :N] # Pixel intensities  
 attributes = data[:, N:] # Attributes  
 y = attributes[:, 2] # Class/person number  
 return X, y, attributes

**Feature Extraction**: X contains pixel intensities (first N columns).

**Attribute Extraction**: attributes contains additional attributes.

**Label Extraction**: y contains the class or person number (third column of attributes).

**4. Splitting Data by Person**

The split\_data\_by\_person function divides the dataset into training and testing sets, ensuring an even split for each class.

def split\_data\_by\_person(X, y, attributes):  
 unique\_classes = np.unique(y)  
 X\_train, X\_test = [], []  
 y\_train, y\_test = [], []  
  
 for cls in unique\_classes:  
 indices = np.where(y == cls)[0]  
 np.random.shuffle(indices)  
 split\_point = len(indices) // 2  
  
 train\_indices = indices[:split\_point]  
 test\_indices = indices[split\_point:]  
  
 X\_train.extend(X[train\_indices])  
 y\_train.extend(y[train\_indices])  
  
 X\_test.extend(X[test\_indices])  
 y\_test.extend(y[test\_indices])  
  
 return np.array(X\_train), np.array(X\_test), np.array(y\_train), np.array(y\_test)

**Class Identification**: Identifies unique classes in the dataset.

**Index Shuffling and Splitting**: Shuffles indices and splits data evenly into training and testing sets for each class.

**Data Aggregation**: Aggregates the split data into training and testing sets.

**5. Loading and Combining Datasets**

The script loads three datasets and combines them into a single dataset for preprocessing.

# Load datasets  
x\_data, N = load\_data('x24x24.txt')  
y\_data, \_ = load\_data('y24x24.txt')  
z\_data, \_ = load\_data('z24x24.txt')  
  
# Combine datasets  
data = np.vstack((x\_data, y\_data, z\_data))

**Loading Datasets**: Calls load\_data for each dataset file.

**Combining Data**: Vertically stacks the data arrays into a single array.

**6. Preprocessing the Combined Data**

The combined data is preprocessed to extract features, labels, and attributes.

X, y, attributes = preprocess\_data(data, N)

**7. Splitting the Data**

The data is split into training and testing sets based on the class/person.

X\_train, X\_test, y\_train, y\_test = split\_data\_by\_person(X, y, attributes)

**8. Training the Random Forest Classifier**

The script trains a Random Forest Classifier using the training data.

rf\_classifier = RandomForestClassifier(random\_state=42)  
rf\_classifier.fit(X\_train, y\_train)

**Model Initialization**: Initializes the Random Forest Classifier with a random state for reproducibility.

**Model Training**: Fits the classifier to the training data.

**9. Evaluating the Model**

The trained model is evaluated on the test data, and the results are printed.

rf\_predictions = rf\_classifier.predict(X\_test)  
  
print("Random Forest Classifier Accuracy:", accuracy\_score(y\_test, rf\_predictions))  
print("\nRandom Forest Classifier Report:\n", classification\_report(y\_test, rf\_predictions))

**Model Prediction**: Uses the model to predict labels for the test data.

**Accuracy Calculation**: Calculates and prints the accuracy of the model.

**Detailed Report**: Prints a detailed classification report including precision, recall, and F1-score for each class.

**Accuracy achieved**: around 65%

**10. Saving the Model**

The trained model is saved to a file for future use.

joblib.dump(rf\_classifier, 'rf\_classifier.pkl')

**Model Saving**: Saves the trained Random Forest model to a file named rf\_classifier.pkl.

**Conclusion**

This script effectively demonstrates the process of training and evaluating a machine learning model using a Random Forest Classifier. It covers data loading, preprocessing, splitting, training, evaluation, and model saving. The use of a Random Forest Classifier results in a robust and accurate model suitable for complex classification tasks.