Face Recognition Pipeline with PCA and SVM

This document provides a detailed description of a face recognition system implemented using Principal Component Analysis (PCA) for dimensionality reduction and a Support Vector Machine (SVM) for classification. The system processes a dataset of face images, trains a model, evaluates its performance, and visualizes the results. For our model, we have 40 different subjects/people. For each subject, we have 10 different images.

Contents

1 Importing Libraries

The script starts by importing the necessary libraries:

```
import os
import numpy as np
import imageio
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import time
import math
```

2 Helper Functions

2.1 find_nearest_square(n)

This function checks if a given number n is a perfect square. If not, it finds the nearest larger perfect square and returns its square root.

- **Input**: A positive integer n.
- **Output**: The integer square root of the nearest larger perfect square if n is not a perfect square, otherwise the square root of n.

```
def find_nearest_square(n):
    # Calculate the integer square root
    sqrt_n = int(math.sqrt(n))

    # Check if n is a perfect square
    if sqrt_n * sqrt_n == n:
        return int(math.sqrt(n))  # n is a perfect square
    else:
        # Find the next perfect square
        next_sqrt = sqrt_n + 1
        next_square = next_sqrt * next_sqrt
        return int(math.sqrt(next_square))
```

3 Main Function

3.1 process_faces(image_dir, num_subjects)

This is the core function that processes the face images, trains the model, and evaluates its performance.

- **Inputs**:
  - `image_dir`: The directory containing subject folders with face images.
  - `num_subjects`: The number of subjects to process.

3.1.1 Loading Images

The function begins by initializing empty lists to hold the images and labels. It then loops through the specified number of subject directories, reads the images, and stores them in the lists.

```
# Initialize lists to hold the images and labels
images = []
labels = []

# Loop through the first `num_subjects` subject directories
subject_dirs = sorted(os.listdir(image_dir))[:num_subjects + 1]
for subject_dir in subject_dirs:
    subject_path = os.path.join(image_dir, subject_dir)
    if os.path.isdir(subject_path):
        label = int(subject_dir[1:]) - 1  # Assuming directories are named as 'sX'
        # Loop through each image in the subject directory
        for image_name in os.listdir(subject_path):
            if image_name.endswith('.pgm'):
                image_path = os.path.join(subject_path, image_name)
                image = imageio.imread(image_path)
                images.append(image)
                labels.append(label)
```

3.1.2 Preprocessing

The lists of images and labels are converted to NumPy arrays. The images are reshaped to a 2D array suitable for the model.

```
# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Reshape images for the model
n_samples, h, w = images.shape
X = images.reshape(n_samples, h * w)
y = labels

print(f"Loaded␣{n_samples}␣images␣with␣shape␣{h}x{w}")
```

3.1.3 Splitting Data

The dataset is split into training and testing sets, with stratification to ensure balanced class distribution.

```
# Split the dataset into training and testing sets with stratification
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.25, random_state=42
```

3.1.4 Model Definition and Training

A PCA and SVM pipeline is defined. The number of PCA components is determined dynamically based on the training data. A grid search is performed to find the best hyperparameters for the SVM.

```
# Define the PCA and SVM pipeline
# Calculate the number of PCA components dynamically
n_components = min(len(Xtrain), Xtrain.shape[1], 20)  # we can adjust 20 to any max c
pca = PCA(n_components=n_components, whiten=True, svd_solver='randomized', random_sta
svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)

# Define parameter grid for GridSearchCV
param_grid = {'svc__C': [1, 5, 10, 50],
              'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model, param_grid)

# Measure execution time
start_time = time.time()
grid.fit(Xtrain, ytrain)
end_time = time.time()

print(f"GridSearchCV took {end_time - start_time:.2f} seconds")
print(grid.best_params_)
```

3.1.5 Model Evaluation and Visualization

The best model is used to make predictions on the test set. The results are visualized by plotting some of the predictions and displaying a confusion matrix.

```
# Predict and visualize
model = grid.best_estimator_
yfit = model.predict(Xtest)

# Plot some predictions
n_plots = min(num_subjects*10, len(Xtest))  # Plot len(Xtest) number of images, times
PictureD = find_nearest_square(len(Xtest)) # find dimesnions for pictures to look goo
#print("xtest number of pictures: ", len(Xtest))
fig, ax = plt.subplots(PictureD, PictureD, figsize=(12, 8))
for i, axi in enumerate(ax.flat):
    if i < n_plots:
        axi.imshow(Xtest[i].reshape(h, w), cmap='bone')
        axi.set(xticks=[], yticks=[])
        axi.set_ylabel(f"Predicted: {yfit[i]}\nTrue: {ytest[i]}", color='black' if yf
    else:
        axi.axis('off')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=1)
plt.show()
```

```
# Classification report
print(classification_report(ytest, yfit))

# Confusion matrix
mat = confusion_matrix(ytest, yfit)
plt.figure(figsize=(10, 8))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=np.arange(num_subjects),  # Adjust labels to number of subjec
            yticklabels=np.arange(num_subjects))
plt.xlabel('True Label')
plt.ylabel('Predicted Label')
plt.title('Confusion Matrix')
plt.show()
```

4 Execution

The process_faces function is called with the directory containing the face images and the number of subjects to process.

```
# call the main function
image_dir = r"C:\Users\Zakaria\Desktop\att_faces-#2"
num_subjects = 7  # Change this value to any number of subjects you want to process
process_faces(image_dir, num_subjects)
```