AI BASED FACIAL RECOGNITION SYSTEM

SUBMITTED BY: ADITHYA SANTHILAL

DATE:

INTRODUCTION

The first study on automatic facial recognition systems was semi-automatic. When discussing about facial recognition approaches it is accurate to categorize it into 2D and 3D.We can classify the face recognition researches carried out with 2D approach in three categories; analytical (feature-based, local), global (appearance) and hybrid methods. While analytical approaches want to recognize by comparing the properties of the facial components, global approaches try to achieve a recognition with data derived from all the face. Hybrid approaches, together with local and global approaches, try to obtain data that expresses the face more accurately. Face recognition performed in this kernel can assessed under global face recognition approaches.

Global-based methods are applied to face recognition by researchers because they perform facial recognition without feature extraction which is troublesome in feature based methods. Globally based methods have been used in face recognition since the 1990s, since they significantly improve facial recognition efficiency. Kirby and Sirovich (1990) first developed a method known as Eigenface, which is used in facial representation and recognition based on Principal Component Analysis . With this method, Turk and Pentland transformed the entire face image into vectors and computed eigenfaces with a set of samples. PCA was able to obtain data representing the face at the optimum level with the data obtained from the image. The different facial and illumination levels of the same person were evaluated as the weakness point of PCA.

The face recognition performed here is totally based on Turk and Pentland work.

Dataset Overview

- Face images taken between April 1992 and April 1994.
- There are ten different image of each of 40 distinct people
- There are 400 face images in the dataset
- Face images were taken at different times, varying lighting, facial express and facial detail
- All face images have black background
- The images are gray level
- Size of each image is 64x64
- Image pixel values were scaled to [0, 1] interval
- Names of 40 people were encoded to an integer from 0 to 39

Library Reference

- **NumPy**: A library for numerical computing in Python.
- **Pandas**: A library for data manipulation and analysis.
- **Matplotlib**: A plotting library for creating visualizations in Python
- **Scikit-learn**: A machine learning library that provides tools for data mining and data analysis.
- **OS**: A module in Python providing a way to interface with the operating system
- **Warnings**: A module in Python used for issuing warning messages.
- **Seaborn**: A data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- **mglearn**: A helper library for the book "Introduction to Machine Learning with Python" by Andreas C. Müller and Sarah Guido

METHODOLOGY

In this study, face recognition is performed using the face images in the Olivetti data set. The steps for face recognition are as follows:

- Principal components of face images were obtained by PCA.
- Adequate number of principal components determined
- According to three different classification models, accuracy score obtained.
- According to three different classification models, cross-validation accuracy score are obtained.
- Parameter optimization of the best model will be made.

```
import numpy as np # numerical computing with Python

import pandas as pd # for data manipulation and analysis


#Visualization

import matplotlib.pyplot as plt


#Machine Learning

from sklearn.model_selection import train_test_split    # For splitting data into train and test sets

from sklearn.decomposition import PCA     # For Principal Component Analysis

from sklearn.svm import SVC      # For Support Vector Classifier

from sklearn.naive_bayes import GaussianNB        # For Gaussian Naive Bayes classifier

from sklearn.neighbors import KNeighborsClassifier   # For K-Nearest Neighbors classifier

from sklearn.tree import DecisionTreeClassifier    # For Decision Tree classifier

from sklearn.linear_model import LogisticRegression    # For Logistic Regression classifier

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis    # For Linear Discriminant Analysis

from sklearn import metrics      # For evaluating model performance metrics


#System

import os

print(os.listdir("/Users/adithyasanthilal/Downloads/olivetti"))


# Importing warnings module to handle warnings,Filtering out and ignoring warnings and printing
```

```
import warnings
warnings.filterwarnings('ignore')
print("Warnings ignored!!")

#loads the data and target labels from two separate numpy files located at the specified paths.
data=np.load("/Users/adithyasanthilal/Downloads/olivetti/olivetti_faces.npy")
target=np.load("/Users/adithyasanthilal/Downloads/olivetti/olivetti_faces_target.npy")

print("There are {} images in the dataset".format(len(data)))
print("There are {} unique targets in the dataset".format(len(np.unique(target))))
print("Size of each image is {}x{}".format(data.shape[1],data.shape[2]))
# Printing an example of scaled pixel values of the first image
print("Pixel values were scaled to [0,1] interval. e.g:{}".format(data[0][0,:4]))

#prints the unique target numbers or classes present in the target variable.
print("unique target number:",np.unique(target))

def show_40_distinct_people(images, unique_ids):
    """
    Function to display images of 40 distinct people in a dataset.

    Parameters:
        images (numpy.ndarray): Array containing the images to be displayed.
        unique_ids (list): List of unique IDs of the 40 distinct people.
```

```
    """
    #Creating 4X10 subplots in  18x9 figure size
    fig, axarr=plt.subplots(nrows=4, ncols=10, figsize=(18, 9))
    #For easy iteration flattened 4X10 subplots matrix to 40 array
    axarr=axarr.flatten()

    #iterating over user ids
    for unique_id in unique_ids:
        # Calculate the index of the first image associated with the current unique ID
        image_index=unique_id*10
        # Display the image associated with the calculated index on the corresponding subplot
        axarr[unique_id].imshow(images[image_index], cmap='gray')
        # Remove tick marks on both x-axis and y-axis  for  improving the visual clarity of the subplot
        axarr[unique_id].set_xticks([])
        axarr[unique_id].set_yticks([])
         # Sets the title of the subplot to indicate the ID of the person
        axarr[unique_id].set_title("face id:{}".format(unique_id))
# Sets a title for the entire figure
    plt.suptitle("There are 40 distinct people in the dataset")

show_40_distinct_people(data, np.unique(target))

def show_10_faces_of_n_subject(images, subject_ids):
```

```
    """
    This function essentially creates a subplot grid and iterates through the provided subject IDs.
    For each subject, it retrieves and displays 10 corresponding face images from the dataset in a grid layout,
    hiding the axes and setting informative titles.
    """
    cols=10# each subject has 10 distinct face images
    rows=(len(subject_ids)*10)/cols #calculates the number of rows needed in the subplot grid
    rows=int(rows)

    fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(18,9))
    #axarr=axarr.flatten()

    for i, subject_id in enumerate(subject_ids):
        #This loop iterates over each subject ID in the subject_ids list, along with its index i
        for j in range(cols):
            image_index=subject_id*10 + j
            #This line displays the image at the calculated index
            axarr[i,j].imshow(images[image_index], cmap="gray")
            axarr[i,j].set_xticks([])
            axarr[i,j].set_yticks([])
            axarr[i,j].set_title("face id:{}".format(subject_id))

#You can playaround subject_ids to see other people faces
show_10_faces_of_n_subject(images=data, subject_ids=[0,5, 21, 24, 36])
```

#This line reshapes the data array into a 2-dimensional array X where each row represents an image and each column represents a pixel

X=data.reshape((data.shape[0],data.shape[1]*data.shape[2]))

print("X shape:",X.shape)

Splitting the dataset into training and testing sets

X: Feature dataset containing pixel values of the images

test_size=0.3: Specifies that 30% of the data will be reserved for testing, and the remaining 70% will be used for training

stratify=target: Ensures that the distribution of classes in the training and testing sets remains similar to the original dataset

random_state=0: Sets the random seed for reproducibility, ensuring consistent results across multiple runs

X_train, X_test, y_train, y_test=train_test_split(X, target, test_size=0.3, stratify=target, random_state=0)

print("X_train shape:",X_train.shape)

print("y_train shape:{}".format(y_train.shape))

Creating an empty DataFrame to store the target variable information

y_frame=pd.DataFrame()

Adding the 'subject ids' column to the DataFrame and assigning it the values from the training target variable

y_frame['subject ids']=y_train

Grouping the DataFrame by the 'subject ids' column and calculating the size (number of occurrences) of each group

Plotting a bar chart to visualize the distribution of samples for each class (subject ids)

```
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15,8),title="Number of Samples for Each Classes")

import mglearn

mglearn.plots.plot_pca_illustration()


#Each sample in X is here represented by two principal components instead of the original higher-dimensional features

from sklearn.decomposition import PCA

# Creating a PCA object with 2 principal components

pca=PCA(n_components=2)

# Fitting the PCA model to the data X

pca.fit(X)

# Transforming the data X into the new principal component space

X_pca=pca.transform(X)


number_of_people=10

# Calculating the index range based on the number of people (each with 10 images)

index_range=number_of_people*10

fig=plt.figure(figsize=(10,8))

ax=fig.add_subplot(1,1,1)

# Creating a scatter plot of the PCA-transformed data

scatter=ax.scatter(X_pca[:index_range,0],   # x-coordinate (first principal component)

        X_pca[:index_range,1],        # y-coordinate (second principal component)

        c=target[:index_range],       # color based on target (face ID)

        s=10,                         # size of each point
```

```
        cmap=plt.get_cmap('jet', number_of_people)  # colormap for colors
       )

ax.set_xlabel("First Principle Component")
ax.set_ylabel("Second Principle Component")
ax.set_title("PCA projection of {} people".format(number_of_people))

fig.colorbar(scatter)

""" This plot helps in understanding how much information is captured by each principal component and
can guide decisions on how many components to retain for dimensionality reduction or feature extraction."""
pca=PCA()
pca.fit(X)

plt.figure(1, figsize=(12,8))

# Plot the explained variance of each principal component
plt.plot(pca.explained_variance_, linewidth=2)

plt.xlabel('Components')
plt.ylabel('Explained Variaces')
plt.show()

pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train)
```

```
# Create a new figure and axis for plotting

fig,ax=plt.subplots(1,1,figsize=(8,8))

# Display the average face image (mean of all faces) reshaped to original dimensions (64x64) using PCA

ax.imshow(pca.mean_.reshape((64,64)), cmap="gray")

ax.set_xticks([])

ax.set_yticks([])

ax.set_title('Average Face')


# Get the number of eigenfaces (principal components)

number_of_eigenfaces=len(pca.components_)

# Reshape the components to the original image dimensions (data.shape[1] x data.shape[2])

eigen_faces=pca.components_.reshape((number_of_eigenfaces, data.shape[1], data.shape[2]))


# Define the number of columns for subplots (eigenfaces per row)

cols=10

# Calculate the number of rows needed based on the number of eigenfaces and cols

rows=int(number_of_eigenfaces/cols)

# Create a figure with subplots arranged in rows and columns

fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(15,15))

# Flatten the subplot array for easy iteration

axarr=axarr.flatten()

# Iterate over each eigenface and display the eigenface in the corresponding subplot

for i in range(number_of_eigenfaces):
```

```
    axarr[i].imshow(eigen_faces[i],cmap="gray")
    axarr[i].set_xticks([])
    axarr[i].set_yticks([])
    axarr[i].set_title("eigen id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10*"=", 10*"="))

# Transform the training dataset (X_train) using the PCA model
X_train_pca=pca.transform(X_train)
# Transform the test dataset (X_test) using the same PCA model
X_test_pca=pca.transform(X_test)

import seaborn as sns
plt.figure(1, figsize=(12,8))
sns.heatmap(metrics.confusion_matrix(y_test, y_pred))

print(metrics.classification_report(y_test, y_pred))

models=[]
# Appends each model along with its name to the list 'models'
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(("LR",LogisticRegression()))
models.append(("NB",GaussianNB()))
models.append(("KNN",KNeighborsClassifier(n_neighbors=5)))
models.append(("DT",DecisionTreeClassifier()))
models.append(("SVM",SVC()))
```

```
# Iterate through each model in the 'models' list
for name, model in models:

    clf=model    # Instantiate the model

    clf.fit(X_train_pca, y_train)   # Fit the model on the training data

    y_pred=clf.predict(X_test_pca)   # Make predictions on the test data
    print(10*"=","{} Result".format(name).upper(),10*"=")
    print("Accuracy score:{:0.2f}".format(metrics.accuracy_score(y_test,
y_pred)))
    print()

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
pca=PCA(n_components=n_components, whiten=True)  # Instantiate PCA with
specified parameters
pca.fit(X)    # Fit PCA to the data and transform X
X_pca=pca.transform(X)
# Iterate through each model in the 'models' list
for name, model in models:
    kfold=KFold(n_splits=5, shuffle=True, random_state=0)   # Create a KFold
cross-validation object

    cv_scores=cross_val_score(model, X_pca, target, cv=kfold)   # Perform cross-
validation using the specified model and PCA-transformed data
    print("{} mean cross validations score:{:.2f}".format(name,
cv_scores.mean()))
```

```
lr=LinearDiscriminantAnalysis()     # Instantiate Linear Discriminant Analysis (LDA) classifier

lr.fit(X_train_pca, y_train)        # Fit LDA model to the PCA-transformed training data

y_pred=lr.predict(X_test_pca)       # Predict labels for the PCA-transformed test data

print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))


cm=metrics.confusion_matrix(y_test, y_pred)


plt.subplots(1, figsize=(12,12))

sns.heatmap(cm)


print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))


"""LeaveOneOut() creates an iterator that yields train/test indices for each sample.

It splits the dataset into n consecutive folds, where n is the number of samples.

For each iteration, one sample is used as the test set, and the rest are used as the training set."""

from sklearn.model_selection import LeaveOneOut

loo_cv=LeaveOneOut()

clf=LogisticRegression()   #initializes a logistic regression classifier.

""" computes the cross-validated scores for the specified classifier (clf).

It uses the provided dataset (X_pca and target) and cross-validation iterator (loo_cv) to split the data."""

cv_scores=cross_val_score(clf,
```

```
                X_pca,
                target,
                cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy
score:{:.2f}".format(clf.__class__.__name__,
                                                cv_scores.mean()))

from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LinearDiscriminantAnalysis()
cv_scores=cross_val_score(clf,
                X_pca,
                target,
                cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy
score:{:.2f}".format(clf.__class__.__name__,
                                                cv_scores.mean()))

#We can do GridSearchCV to improve model generalization performance. To
that we will tune the hyperparameters of Logistic Regression classifier.
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import LeaveOneOut

lr=LogisticRegression(C=1.0, penalty="l2")  #Initializing Logistic Regression
#Here c controls the regularization strength,A smaller value of C indicates
stronger regularization In this case, C=1.0 means moderate regularization.
```

```
lr.fit(X_train_pca, y_train)

print("lr score:{:.2f}".format(lr.score(X_test_pca, y_test)))
```

from sklearn.preprocessing import label_binarize #label_binarize is a function which converts categorical labels into a binary form suitable for multi-class classification.

from sklearn.multiclass import OneVsRestClassifier #OneVsRestClassifier is a strategy for multi-class classification where one binary classifier is trained for each class.

```
Target=label_binarize(target, classes=range(40))  #Binarizing the Target Variable

print(Target.shape)  #prints the shape of the binarized target matrix

print(Target[0])  #prints the binary representation of the first image's target label
```

n_classes=Target.shape[1] #n_classes is a variable which stores the total number of people in the dataset, which is equivalent to the number of columns in the binarized Target matrix.

#uses train_test_split to split the data (X) and the binarized target (Target) into training and testing sets for a multi-class classification problem

```
X_train_multiclass, X_test_multiclass, y_train_multiclass, y_test_multiclass=train_test_split(X,
                                                    Target,
                                                    test_size=0.3,
                                                    stratify=Target,
                                                random_state=0)
```

Initialize PCA with specified number of components and whiten the data

```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train_multiclass)   # Fit PCA on the training data to learn the principal components

# Transform the training data into the reduced feature space defined by the principal components
X_train_multiclass_pca=pca.transform(X_train_multiclass)
# Transform the testing data into the same reduced feature space using the learned PCA transformation
X_test_multiclass_pca=pca.transform(X_test_multiclass)

# Create a One-vs-Rest classifier using Logistic Regression as the base estimator
oneRestClassifier=OneVsRestClassifier(lr)

# Fit the One-vs-Rest classifier on the PCA-transformed training data and corresponding target labels
oneRestClassifier.fit(X_train_multiclass_pca, y_train_multiclass)
# Calculate the decision function scores for the PCA-transformed testing data
y_score=oneRestClassifier.decision_function(X_test_multiclass_pca)

# Initialize dictionaries to store precision, recall, and average precision scores for each class
precision = dict()
recall = dict()
average_precision = dict()

# Calculate precision, recall, and average precision for each class using precision_recall_curve
```

```
for i in range(n_classes):
    precision[i], recall[i], _ = metrics.precision_recall_curve(y_test_multiclass[:,
i],
                                                        y_score[:, i])
    average_precision[i] = metrics.average_precision_score(y_test_multiclass[:,
i], y_score[:, i])


# Calculate micro-average precision, recall, and average precision
precision["micro"], recall["micro"], _ =
metrics.precision_recall_curve(y_test_multiclass.ravel(),
    y_score.ravel())
average_precision["micro"] =
metrics.average_precision_score(y_test_multiclass, y_score,
                                                    average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))


# Create step_kwargs dictionary based on the availability of 'step' argument in
plt.fill_between
step_kwargs = {'step': 'post'} if 'step' in
plt.fill_between.__code__.co_varnames else {}


# Create a new figure for plotting precision-recall curve
plt.figure(1, figsize=(12, 8))


# Plot the precision-recall curve using plt.step
plt.step(recall['micro'], precision['micro'], color='b', alpha=0.2, where='post')
# Fill the area under the precision-recall curve using plt.fill_between
```

```
plt.fill_between(recall["micro"], precision["micro"], alpha=0.2, color='b',
**step_kwargs)

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('Average precision score, micro-averaged over all classes:
AP={0:0.2f}'.format(average_precision["micro"]))

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Determine the number of components for LDA based on input data

n_components = min(X_train.shape[1], len(np.unique(y_train)) - 1)

# Create an instance of LinearDiscriminantAnalysis with specified number of
components

lda = LinearDiscriminantAnalysis(n_components=n_components)

# Fit LDA to the training data and transform it

X_train_lda = lda.fit(X_train, y_train).transform(X_train)

# Transform the test data using the trained LDA model

X_test_lda=lda.transform(X_test)

# Create a LogisticRegression instance with specified regularization parameters

lr=LogisticRegression(C=1.0, penalty="l2")

# Fit the logistic regression model using the LDA-transformed training data

lr.fit(X_train_lda,y_train)

# Predict the target labels for the LDA-transformed test data
```

```
y_pred=lr.predict(X_test_lda)


print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))

print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))

"""Application of machine learning on data sets has a standard workflow. Sklearn offers the Pipeline object to automate this workflow.

Pipeline allows standard work flows for performing machine learning operations such as scaling, feature extraction and modeling.

The Pipeline guarantees the same operation in the entire data set, ensuring that the training and test data are consistent"""

from sklearn.pipeline import Pipeline


work_flows_std = list()

# Defines the components of your pipeline

work_flows_std.append(('lda', LinearDiscriminantAnalysis(n_components=n_components)))

work_flows_std.append(('logReg', LogisticRegression(C=1.0, penalty="l2")))

# Create a Pipeline object

model_std = Pipeline(work_flows_std)

# Fit the pipeline on the training data

model_std.fit(X_train, y_train)

y_pred=model_std.predict(X_test)


print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))

print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python 3 (ipykernel)

[1]:
```
import numpy as np # numerical computing with Python
import pandas as pd # for data manipulation and analysis

#Visualization
import matplotlib.pyplot as plt

#Machine Learning
from sklearn.model_selection import train_test_split    # For splitting data into train and test sets
from sklearn.decomposition import PCA     # For Principal Component Analysis
from sklearn.svm import SVC      # For Support Vector Classifier
from sklearn.naive_bayes import GaussianNB        # For Gaussian Naive Bayes classifier
from sklearn.neighbors import KNeighborsClassifier   # For K-Nearest Neighbors classifier
from sklearn.tree import DecisionTreeClassifier    # For Decision Tree classifier
from sklearn.linear_model import LogisticRegression    # For Logistic Regression classifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis    # For Linear Discriminant Analysis
from sklearn import metrics      # For evaluating model performance metrics

#System
import os
print(os.listdir("/Users/adithyasanthilal/Downloads/olivetti"))
```

```
['olivetti_faces_target.npy', 'olivetti_faces.npy']
```

[2]:
```
# Importing warnings module to handle warnings,Filtering out and ignoring warnings and printing
import warnings
warnings.filterwarnings('ignore')
print("Warnings ignored!!")
```

```
Warnings ignored!!
```

[3]:
```
#loads the data and target labels from two separate numpy files located at the specified paths.
data=np.load("/Users/adithyasanthilal/Downloads/olivetti/olivetti_faces.npy")
target=np.load("/Users/adithyasanthilal/Downloads/olivetti/olivetti_faces_target.npy")
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python 3 (ipykernel)

[4]:
```
print("There are {} images in the dataset".format(len(data)))
print("There are {} unique targets in the dataset".format(len(np.unique(target))))
print("Size of each image is {}x{}".format(data.shape[1],data.shape[2]))
# Printing an example of scaled pixel values of the first image
print("Pixel values were scaled to [0,1] interval. e.g:{}".format(data[0][0,:4]))
```

```
There are 400 images in the dataset
There are 40 unique targets in the dataset
Size of each image is 64x64
Pixel values were scaled to [0,1] interval. e.g:[0.30991736 0.3677686  0.41735536 0.44214877]
```

[5]:
```
#prints the unique target numbers or classes present in the target variable.
print("unique target number:",np.unique(target))
```

```
unique target number: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]
```

[6]:
```
def show_40_distinct_people(images, unique_ids):
    """
    Function to display images of 40 distinct people in a dataset.

    Parameters:
        images (numpy.ndarray): Array containing the images to be displayed.
        unique_ids (list): List of unique IDs of the 40 distinct people.

    """
    #Creating 4X10 subplots in  18x9 figure size
    fig, axarr=plt.subplots(nrows=4, ncols=10, figsize=(18, 9))
    #For easy iteration ,we flattened 4X10 subplots matrix to 40 array
    axarr=axarr.flatten()

    #iterating over user ids
    for unique_id in unique_ids:
        # Calculate the index of the first image associated with the current unique ID
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python 3 (ipykernel)

[6]:
```
def show_40_distinct_people(images, unique_ids):
    """
    Function to display images of 40 distinct people in a dataset.

    Parameters:
        images (numpy.ndarray): Array containing the images to be displayed.
        unique_ids (list): List of unique IDs of the 40 distinct people.

    """
    #Creating 4X10 subplots in  18x9 figure size
    fig, axarr=plt.subplots(nrows=4, ncols=10, figsize=(18, 9))
    #For easy iteration ,we flattened 4X10 subplots matrix to 40 array
    axarr=axarr.flatten()

    #iterating over user ids
    for unique_id in unique_ids:
        # Calculate the index of the first image associated with the current unique ID
        image_index=unique_id*10
        # Display the image associated with the calculated index on the corresponding subplot
        axarr[unique_id].imshow(images[image_index], cmap='gray')
        # Remove tick marks on both x-axis and y-axis for improving the visual clarity of the subplot
        axarr[unique_id].set_xticks([])
        axarr[unique_id].set_yticks([])
         # Sets the title of the subplot to indicate the ID of the person
        axarr[unique_id].set_title("face id:{}".format(unique_id))
         # Sets a title for the entire figure
    plt.suptitle("There are 40 distinct people in the dataset")
```

[7]:
```
show_40_distinct_people(data, np.unique(target))
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

There are 40 distinct people in the dataset

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[8]:
```
def show_10_faces_of_n_subject(images, subject_ids):
    """
    This function essentially creates a subplot grid and iterates through the provided subject IDs.
    For each subject, it retrieves and displays 10 corresponding face images from the dataset in a grid layout,
    hiding the axes and setting informative titles.
    """
    cols=10# each subject has 10 distinct face images
    rows=(len(subject_ids)*10)/cols #calculates the number of rows needed in the subplot grid
    rows=int(rows)

    fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(18,9))
    #axarr=axarr.flatten()

    for i, subject_id in enumerate(subject_ids):
        #This loop iterates over each subject ID in the subject_ids list, along with its index i
        for j in range(cols):
            image_index=subject_id*10 + j
            #This line displays the image at the calculated index
            axarr[i,j].imshow(images[image_index], cmap="gray")
            axarr[i,j].set_xticks([])
            axarr[i,j].set_yticks([])
            axarr[i,j].set_title("face id:{}".format(subject_id))
```

[9]:
```
#You can playaround subject_ids to see other people faces
show_10_faces_of_n_subject(images=data, subject_ids=[0,5, 21, 24, 36])
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[10]:
```
#Machine learning models can work on vectors. Since the image data is in the matrix form, it must be converted to a vector.
#This line reshapes the data array into a 2-dimensional array X where each row represents an image and each column represents a pixel
X=data.reshape((data.shape[0],data.shape[1]*data.shape[2]))
print("X shape:",X.shape)
```

```
X shape: (400, 4096)
```

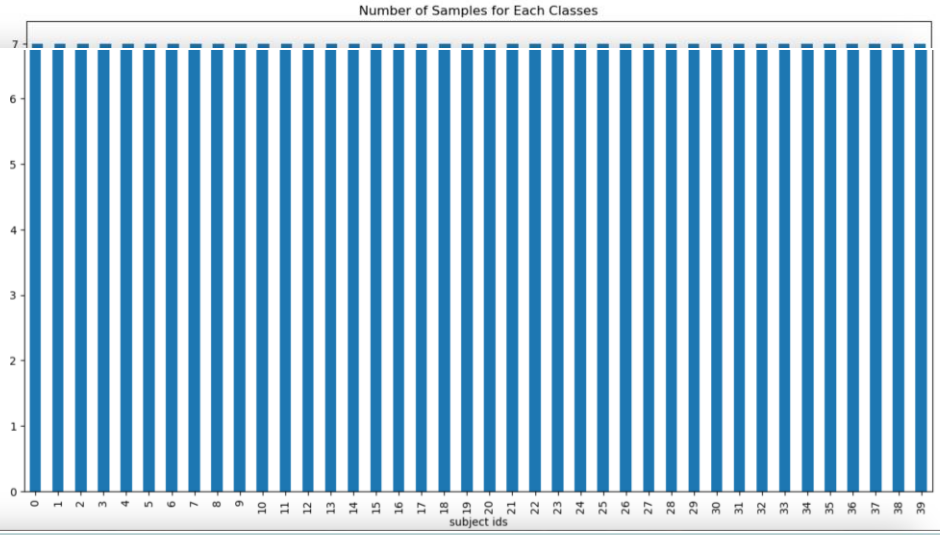
[11]:
```
# Splitting the dataset into training and testing sets
# X: Feature dataset containing pixel values of the images
# test_size=0.3: Specifies that 30% of the data will be reserved for testing, and the remaining 70% will be used for training
# stratify=target: Ensures that the distribution of classes in the training and testing sets remains similar to the original dataset
# random_state=0: Sets the random seed for reproducibility, ensuring consistent results across multiple runs

X_train, X_test, y_train, y_test=train_test_split(X, target, test_size=0.3, stratify=target, random_state=0)
print("X_train shape:",X_train.shape)
print("y_train shape:{}".format(y_train.shape))
```

```
X_train shape: (280, 4096)
y_train shape:(280,)
```

[12]:
```
# Creating an empty DataFrame to store the target variable information
y_frame=pd.DataFrame()
# Adding the 'subject ids' column to the DataFrame and assigning it the values from the training target variable
y_frame['subject ids']=y_train
# Grouping the DataFrame by the 'subject ids' column and calculating the size (number of occurrences) of each group
# Plotting a bar chart to visualize the distribution of samples for each class (subject ids)
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15,8),title="Number of Samples for Each Classes")
```

```
<Axes: title={'center': 'Number of Samples for Each Classes'}, xlabel='subject ids'>
```



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

[13]:
```
import mglearn
```

[14]:
```
pip install mglearn
```

```
Requirement already satisfied: mglearn in /opt/anaconda3/lib/python3.11/site-packages (0.2.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (1.26.4)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (3.8.0)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (1.4.1.post1)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (2.1.4)
Requirement already satisfied: pillow in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (10.2.0)
Requirement already satisfied: cycler in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (0.11.0)
Requirement already satisfied: imageio in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (2.33.1)
Requirement already satisfied: joblib in /opt/anaconda3/lib/python3.11/site-packages (from mglearn) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /opt/anaconda3/lib/python3.11/site-packages (from matplotlib->mglearn) (1.2.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda3/lib/python3.11/site-packages (from matplotlib->mglearn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda3/lib/python3.11/site-packages (from matplotlib->mglearn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.11/site-packages (from matplotlib->mglearn) (23.1)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/anaconda3/lib/python3.11/site-packages (from matplotlib->mglearn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3/lib/python3.11/site-packages (from matplotlib->mglearn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.11/site-packages (from pandas->mglearn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/anaconda3/lib/python3.11/site-packages (from pandas->mglearn) (2023.3)
Requirement already satisfied: scipy>=1.6.0 in /opt/anaconda3/lib/python3.11/site-packages (from scikit-learn->mglearn) (1.13.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.11/site-packages (from scikit-learn->mglearn) (3.4.0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.11/site-packages (from python-dateutil>=2.7->matplotlib->mglearn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[15]:
```
import mglearn
```

[16]:
```
# Importing the mglearn library and calling its plot_pca_illustration function
# This function generates a visualization illustrating the concept of Principal Component Analysis (PCA)
# It helps in understanding how PCA transforms data points from a high-dimensional space to a lower-dimensional space
# and how it captures the main directions of variance in the data
# This visualization aids in comprehending the process of dimensionality reduction and information preservation with PCA
```

jupyter face_recognition Last Checkpoint: 5 days ago

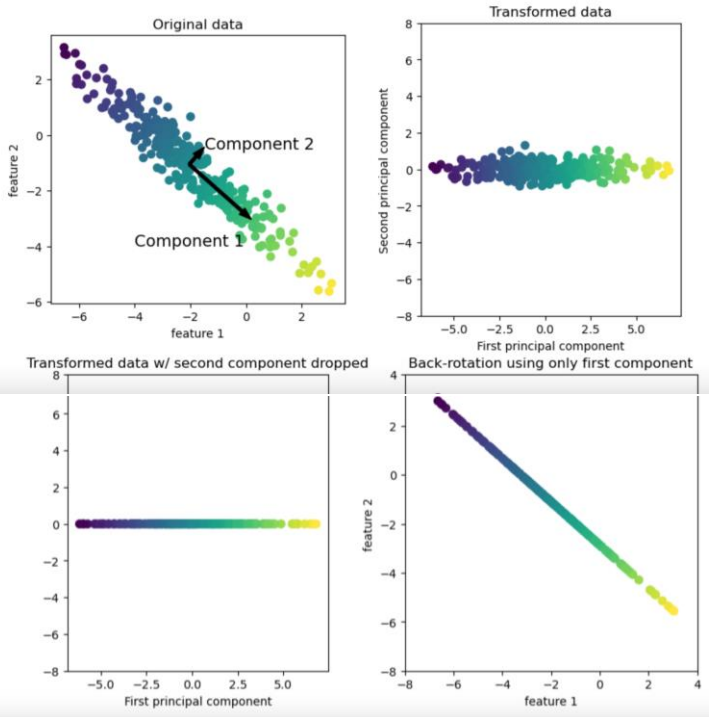
File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

[16]:
```
# Importing the mglearn library and calling its plot_pca_illustration function
# This function generates a visualization illustrating the concept of Principal Component Analysis (PCA)
# It helps in understanding how PCA transforms data points from a high-dimensional space to a lower-dimensional space
# and how it captures the main directions of variance in the data
# This visualization aids in comprehending the process of dimensionality reduction and information preservation with PCA
mglearn.plots.plot_pca_illustration()
```



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

First principal component feature 1

[17]:
```
#Each sample in X is here represented by two principal components instead of the original higher-dimensional features
from sklearn.decomposition import PCA
# Creating a PCA object with 2 principal components
pca=PCA(n_components=2)
# Fitting the PCA model to the data X
pca.fit(X)
# Transforming the data X into the new principal component space
X_pca=pca.transform(X)
```

[18]:
```
number_of_people=10
# Calculating the index range based on the number of people (each with 10 images)
index_range=number_of_people*10
fig=plt.figure(figsize=(10,8))
ax=fig.add_subplot(1,1,1)
# Creating a scatter plot of the PCA-transformed data
scatter=ax.scatter(X_pca[:index_range,0],   # x-coordinate (first principal component)
           X_pca[:index_range,1],           # y-coordinate (second principal component)
           c=target[:index_range],          # color based on target (face ID)
           s=10,                            # size of each point
          cmap=plt.get_cmap('jet', number_of_people)  # colormap for colors
         )

ax.set_xlabel("First Principle Component")
ax.set_ylabel("Second Principle Component")
ax.set_title("PCA projection of {} people".format(number_of_people))

fig.colorbar(scatter)
```

[18]: <matplotlib.colorbar.Colorbar at 0x132a38890>

PCA projection of 10 people

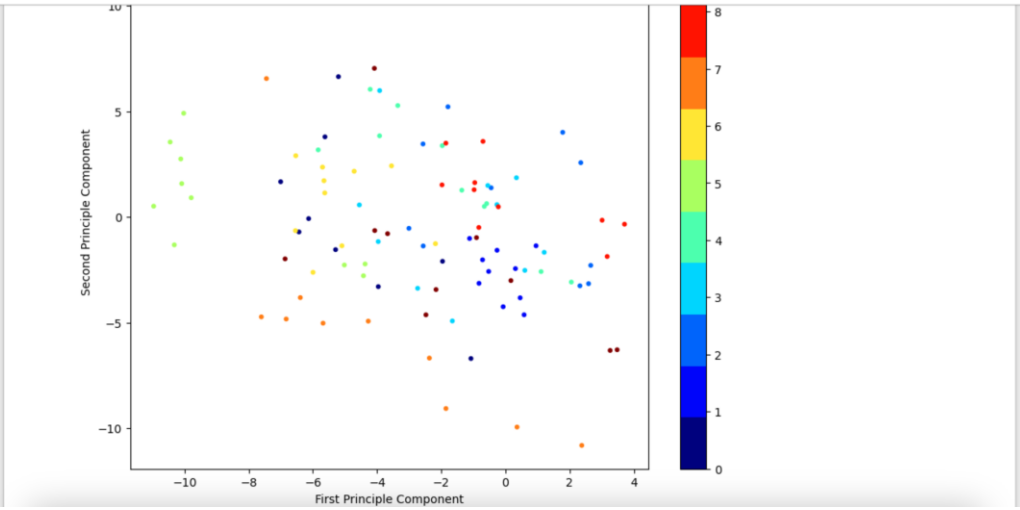
jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

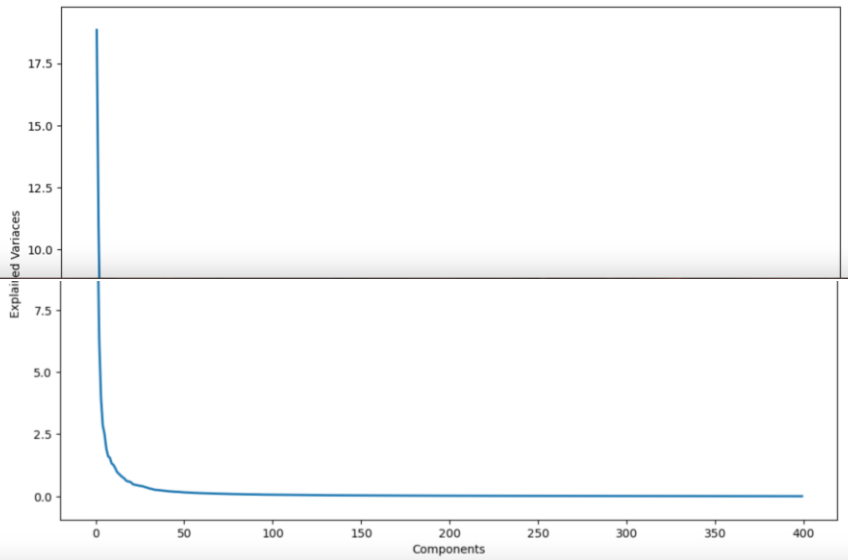
JupyterLab Python 3 (ipykernel)

[19]:
```
""" This plot helps in understanding how much information is captured by each principal component and
can guide decisions on how many components to retain for dimensionality reduction or feature extraction."""
pca=PCA()  # Create a PCA object
pca.fit(X)   # Fit the PCA model to your data matrix X

plt.figure(1, figsize=(12,8))

# Plot the explained variance of each principal component
plt.plot(pca.explained_variance_, linewidth=2)

plt.xlabel('Components')
plt.ylabel('Explained Variaces')
plt.show()
```



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

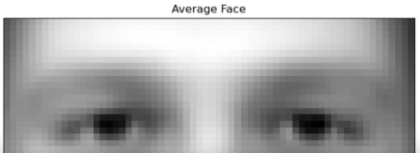
[20]:
```
n_components=90
```

[21]:
```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train)
```

[21]:
```
PCA(n_components=90, whiten=True)
```

[22]:
```
# Create a new figure and axis for plotting
fig,ax=plt.subplots(1,1,figsize=(8,8))
# Display the average face image (mean of all faces) reshaped to original dimensions (64x64) using PCA
ax.imshow(pca.mean_.reshape((64,64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
```

[22]:
```
Text(0.5, 1.0, 'Average Face')
```



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

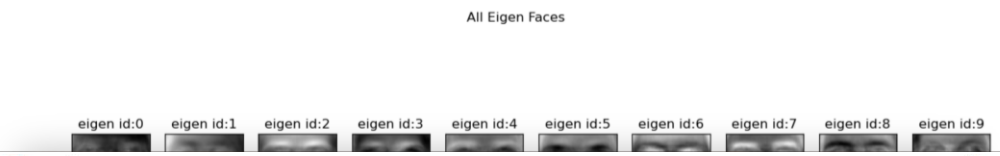
Markdown

JupyterLab Python 3 (ipykernel)

[23]:
```
# Get the number of eigenfaces (principal components)
number_of_eigenfaces=len(pca.components_)
# Reshape the components to the original image dimensions (data.shape[1] x data.shape[2])
eigen_faces=pca.components_.reshape((number_of_eigenfaces, data.shape[1], data.shape[2]))

# Define the number of columns for subplots (eigenfaces per row)
cols=10
# Calculate the number of rows needed based on the number of eigenfaces and cols
rows=int(number_of_eigenfaces/cols)
# Create a figure with subplots arranged in rows and columns
fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(15,15))
# Flatten the subplot array for easy iteration
axarr=axarr.flatten()
# Iterate over each eigenface and display the eigenface in the corresponding subplot
for i in range(number_of_eigenfaces):
    axarr[i].imshow(eigen_faces[i],cmap="gray")
    axarr[i].set_xticks([])
    axarr[i].set_yticks([])
    axarr[i].set_title("eigen id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10*"=", 10*"="))
```

[23]: Text(0.5, 0.98, 'All Eigen Faces')



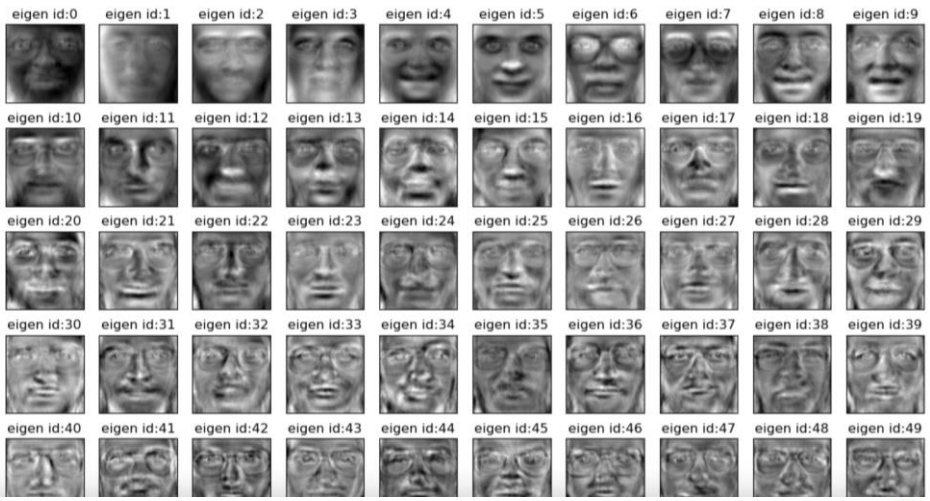
jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)



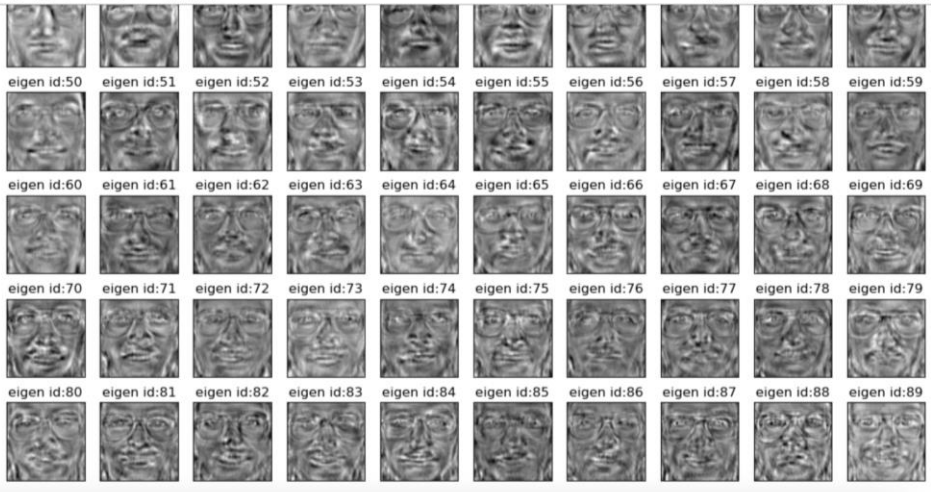
jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Markdown

JupyterLab Python 3 (ipykernel)

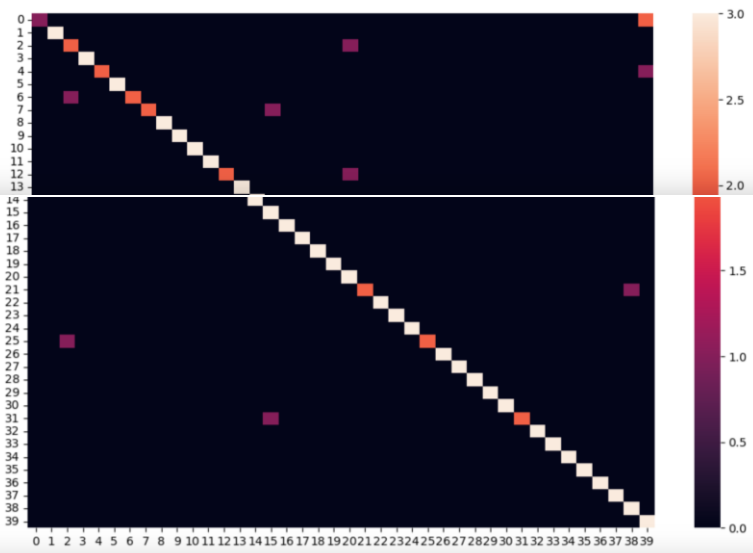
[24]:
```
# Transform the training dataset (X_train) using the PCA model
X_train_pca=pca.transform(X_train)
# Transform the test dataset (X_test) using the same PCA model
X_test_pca=pca.transform(X_test)
```

[25]:
```
# Create a Support Vector Classifier (SVC) object
clf = SVC()
# Train the SVC model using the transformed training dataset (X_train_pca) and corresponding labels (y_train)
clf.fit(X_train_pca, y_train)
# Use the trained model to predict labels for the transformed test dataset (X_test_pca)
y_pred = clf.predict(X_test_pca)
print("accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
```

```
accuracy score:0.92
```

[26]:
```
import seaborn as sns
plt.figure(1, figsize=(12,8))
sns.heatmap(metrics.confusion_matrix(y_test, y_pred))
```

[26]: <Axes: >



jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[27]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       1.00      1.00      1.00         3
           2       0.50      0.67      0.57         3
           3       1.00      1.00      1.00         3
           4       1.00      0.67      0.80         3
           5       1.00      1.00      1.00         3
           6       1.00      0.67      0.80         3
           7       1.00      0.67      0.80         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       1.00      1.00      1.00         3
          15       0.60      1.00      0.75         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       0.60      1.00      0.75         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         3
          24       1.00      1.00      1.00         3
          25       1.00      0.67      0.80         3
          26       1.00      1.00      1.00         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       1.00      1.00      1.00         3
          31       1.00      0.67      0.80         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       0.60      1.00      0.75         3

    accuracy                           0.93       120
   macro avg       0.96      0.93      0.93       120
weighted avg       0.95      0.93      0.93       120
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[33]:
```
"""LeaveOneOut() creates an iterator that yields train/test indices for each sample.
It splits the dataset into n consecutive folds, where n is the number of samples.
For each iteration, one sample is used as the test set, and the rest are used as the training set."""
from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LogisticRegression()   #initializes a logistic regression classifier.
""" computes the cross-validated scores for the specified classifier (clf).
It uses the provided dataset (X_pca and target) and cross-validation iterator (loo_cv) to split the data."""
cv_scores=cross_val_score(clf,
                          X_pca,
                          target,
                          cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy score:{:.2f}".format(clf.__class__.__name__,
                                                                            cv_scores.mean()))
```

```
LogisticRegression Leave One Out cross-validation mean accuracy score:0.95
```

[34]:
```
from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LinearDiscriminantAnalysis()
cv_scores=cross_val_score(clf,
                          X_pca,
                          target,
                          cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy score:{:.2f}".format(clf.__class__.__name__,
                                                                            cv_scores.mean()))
```

```
LinearDiscriminantAnalysis Leave One Out cross-validation mean accuracy score:0.98
```

[35]:
```
#We can do GridSearchCV to improve model generalization performance. To that we will tune the hyperparameters of Logistic Regression classifie
from sklearn.model_selection import GridSearchCV
```

[36]:
```
from sklearn.model_selection import LeaveOneOut
#You can use parameter:{'C': 1.0, 'penalty': 'l2'}
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[36]:
```
from sklearn.model_selection import LeaveOneOut
#You can use parameter:{'C': 1.0, 'penalty': 'l2'}
#grid search cross validation score:0.93
"""
params={'penalty':['l1', 'l2'],
                'C':np.logspace(0, 4, 10)
                }
clf=LogisticRegression()
#kfold=KFold(n_splits=3, shuffle=True, random_state=0)
loo_cv=LeaveOneOut()
gridSearchCV=GridSearchCV(clf, params, cv=loo_cv)
gridSearchCV.fit(X_train_pca, y_train)
print("Grid search fitted..")
print(gridSearchCV.best_params_)
print(gridSearchCV.best_score_)
print("grid search cross validation score:{:.2f}".format(gridSearchCV.score(X_test_pca, y_test)))
"""
```

[36]:
```
'\nparams={\'penalty\':[\'l1\', \'l2\'],\n                \'C\':np.logspace(0, 4, 10)\n                }\nclf=LogisticRegression()\n#kfold=KF
old(n_splits=3, shuffle=True, random_state=0)\nloo_cv=LeaveOneOut()\ngridSearchCV=GridSearchCV(clf, params, cv=loo_cv)\ngridSearchCV.fit(X_tr
ain_pca, y_train)\nprint("Grid search fitted..")\nprint(gridSearchCV.best_params_)\nprint(gridSearchCV.best_score_)\nprint("grid search cross
validation score:{:.2f}".format(gridSearchCV.score(X_test_pca, y_test)))\n'
```

[37]:
```
lr=LogisticRegression(C=1.0, penalty="l2")  #Initializing Logistic Regression
#Here c controls the regularization strength,A smaller value of C indicates stronger regularization In this case, C=1.0 means moderate regular
lr.fit(X_train_pca, y_train)
print("lr score:{:.2f}".format(lr.score(X_test_pca, y_test)))
```

```
lr score:0.93
```

[38]:
```
from sklearn.preprocessing import label_binarize   #label_binarize is a function which converts categorical labels into a binary form suitable
from sklearn.multiclass import OneVsRestClassifier  #OneVsRestClassifier is a strategy for multi-class classification where one binary classif

Target=label_binarize(target, classes=range(40))  #Binarizing the Target Variable
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[38]:
```
from sklearn.preprocessing import label_binarize   #label_binarize is a function which converts categorical labels into a binary form suitable
from sklearn.multiclass import OneVsRestClassifier  #OneVsRestClassifier is a strategy for multi-class classification where one binary classif

Target=label_binarize(target, classes=range(40))  #Binarizing the Target Variable
print(Target.shape)  #prints the shape of the binarized target matrix
print(Target[0])   #prints the binary representation of the first image's target label

n_classes=Target.shape[1]   #n_classes is a variable which stores the total number of people in the dataset, which is equivalent to the number
```

```
(400, 40)
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```

[39]:
```
#uses train_test_split to split the data (X) and the binarized target (Target) into training and testing sets for a multi-class classification
X_train_multiclass, X_test_multiclass, y_train_multiclass, y_test_multiclass=train_test_split(X,
                                                                                              Target,
                                                                                             test_size=0.3,
                                                                                             stratify=Target,
                                                                                     random_state=0)
```

[40]:
```
# Initialize PCA with specified number of components and whiten the data
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train_multiclass)   # Fit PCA on the training data to learn the principal components

# Transform the training data into the reduced feature space defined by the principal components
X_train_multiclass_pca=pca.transform(X_train_multiclass)
# Transform the testing data into the same reduced feature space using the learned PCA transformation
X_test_multiclass_pca=pca.transform(X_test_multiclass)
```

[41]:
```
# Create a One-vs-Rest classifier using Logistic Regression as the base estimator
oneRestClassifier=OneVsRestClassifier(lr)

# Fit the One-vs-Rest classifier on the PCA-transformed training data and corresponding target labels
```

jupyter face_recognition Last Checkpoint: 5 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[40]:
```
# Initialize PCA with specified number of components and whiten the data
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train_multiclass)   # Fit PCA on the training data to learn the principal components

# Transform the training data into the reduced feature space defined by the principal components
X_train_multiclass_pca=pca.transform(X_train_multiclass)
# Transform the testing data into the same reduced feature space using the learned PCA transformation
X_test_multiclass_pca=pca.transform(X_test_multiclass)
```

```
# Create a One-vs-Rest classifier using Logistic Regression as the base estimator
oneRestClassifier=OneVsRestClassifier(lr)

# Fit the One-vs-Rest classifier on the PCA-transformed training data and corresponding target labels
oneRestClassifier.fit(X_train_multiclass_pca, y_train_multiclass)
# Calculate the decision function scores for the PCA-transformed testing data
y_score=oneRestClassifier.decision_function(X_test_multiclass_pca)
```

[42]:
```
# Initialize dictionaries to store precision, recall, and average precision scores for each class
precision = dict()
recall = dict()
average_precision = dict()

# Calculate precision, recall, and average precision for each class using precision_recall_curve
for i in range(n_classes):333
    precision[i], recall[i], _ = metrics.precision_recall_curve(y_test_multiclass[:, i],
                                                                y_score[:, i])
    average_precision[i] = metrics.average_precision_score(y_test_multiclass[:, i], y_score[:, i])

# Calculate micro-average precision, recall, and average precision
precision["micro"], recall["micro"], _ = metrics.precision_recall_curve(y_test_multiclass.ravel(),
    y_score.ravel())
average_precision["micro"] = metrics.average_precision_score(y_test_multiclass, y_score,
```

jupyter face_recognition Last Checkpoint: 1 minute ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
    average_precision[i] = metrics.average_precision_score(y_test_multiclass[:, i], y_score[:, i])

# Calculate micro-average precision, recall, and average precision
precision["micro"], recall["micro"], _ = metrics.precision_recall_curve(y_test_multiclass.ravel(),
    y_score.ravel())
average_precision["micro"] = metrics.average_precision_score(y_test_multiclass, y_score,
                                                             average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

```
Average precision score, micro-averaged over all classes: 0.97
```

[48]:

```
# Create step_kwargs dictionary based on the availability of 'step' argument in plt.fill_between
step_kwargs = {'step': 'post'}

# Create a new figure for plotting precision-recall curve
plt.figure(1, figsize=(12, 8))
# Plot the precision-recall curve using plt.step
plt.step(recall['micro'], precision['micro'], color='b', alpha=0.2, where='post')
# Fill the area under the precision-recall curve using plt.fill_between
plt.fill_between(recall["micro"], precision["micro"], alpha=0.2, color='b', **step_kwargs)

# Customize plot labels and title
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Average precision score, micro-averaged over all classes: AP={0:0.2f}'.format(average_precision["micro"]))

# Show the plot
plt.show()
```

Average precision score, micro-averaged over all classes: AP=0.97

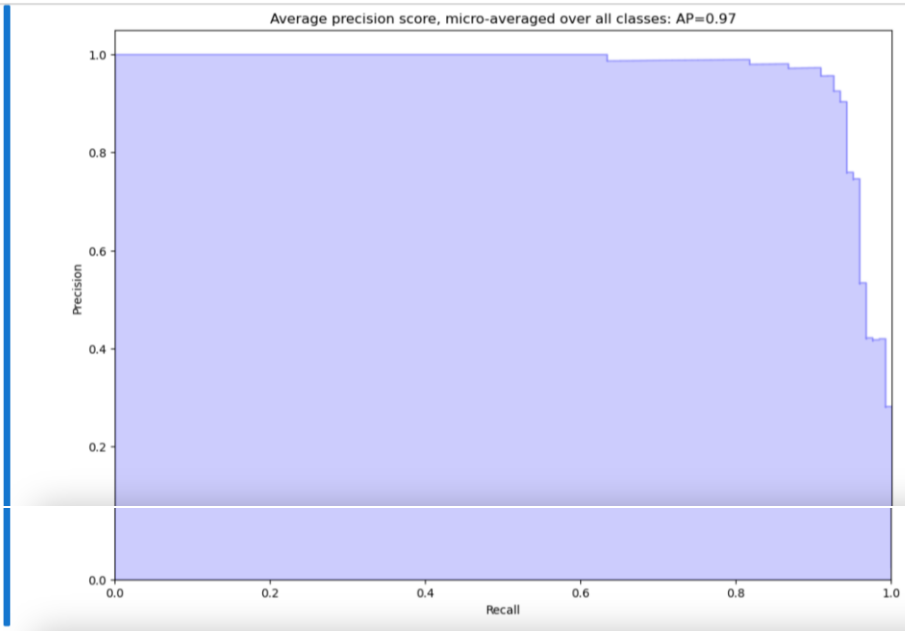
jupyter face_recognition Last Checkpoint: 2 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)



jupyter face_recognition Last Checkpoint: 5 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[49]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

[50]:
```
lda = LinearDiscriminantAnalysis(n_components=n_components)
X_train_lda = lda.fit(X_train, y_train).transform(X_train)
X_test_lda=lda.transform(X_test)
```

[51]:
```
# Determine the number of components for LDA based on input data

n_components = min(X_train.shape[1], len(np.unique(y_train)) - 1)
# Create an instance of LinearDiscriminantAnalysis with specified number of components

lda = LinearDiscriminantAnalysis(n_components=n_components)

# Fit LDA to the training data and transform it

X_train_lda = lda.fit(X_train, y_train).transform(X_train)

# Transform the test data using the trained LDA model

X_test_lda=lda.transform(X_test)
```

[52]:
```
# Create a LogisticRegression instance with specified regularization parameters

lr=LogisticRegression(C=1.0, penalty="l2")

# Fit the logistic regression model using the LDA-transformed training data

lr.fit(X_train_lda,y_train)

# Predict the target labels for the LDA-transformed test data
```

jupyter face_recognition Last Checkpoint: 5 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[52]:
```
# Create a LogisticRegression instance with specified regularization parameters

lr=LogisticRegression(C=1.0, penalty="l2")

# Fit the logistic regression model using the LDA-transformed training data

lr.fit(X_train_lda,y_train)

# Predict the target labels for the LDA-transformed test data

y_pred=lr.predict(X_test_lda)
```

[53]:
```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))

print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

```
Accuracy score:0.95
Classification Results:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       1.00      1.00      1.00         3
           2       0.60      1.00      0.75         3
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       0.75      1.00      0.86         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       1.00      1.00      1.00         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         3
          24       1.00      1.00      1.00         3
          25       1.00      0.67      0.80         3
          26       1.00      1.00      1.00         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       0.60      1.00      0.75         3
          31       1.00      1.00      1.00         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       1.00      1.00      1.00         3

    accuracy                           0.95       120
   macro avg       0.97      0.95      0.95       120
weighted avg       0.97      0.95      0.95       120
```

jupyter face_recognition Last Checkpoint: 7 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[54]:
```
"""Application of machine learning on data sets has a standard workflow. Sklearn offers the Pipeline object to automate this workflow.
Pipeline allows standard work flows for performing machine learning operations such as scaling, feature extraction and modeling.
The Pipeline guarantees the same operation in the entire data set, ensuring that the training and test data are consistent"""
from sklearn.pipeline import Pipeline
```

[55]:
```
work_flows_std = list()
# Defines the components of your pipeline
work_flows_std.append(('lda', LinearDiscriminantAnalysis(n_components=n_components)))
work_flows_std.append(('logReg', LogisticRegression(C=1.0, penalty="l2")))
# Create a Pipeline object
model_std = Pipeline(work_flows_std)
# Fit the pipeline on the training data
model_std.fit(X_train, y_train)
y_pred=model_std.predict(X_test)
```

[56]:
```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

```
Accuracy score:0.95
Classification Results:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       1.00      1.00      1.00         3
           2       0.60      1.00      0.75         3
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       0.75      1.00      0.86         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       1.00      1.00      1.00         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         3
          24       1.00      1.00      1.00         3
          25       1.00      0.67      0.80         3
          26       1.00      1.00      1.00         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       0.60      1.00      0.75         3
          31       1.00      1.00      1.00         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       1.00      1.00      1.00         3

    accuracy                           0.95       120
   macro avg       0.97      0.95      0.95       120
weighted avg       0.97      0.95      0.95       120
```