carol-final-1-1-1-1-2-4

November 5, 2024

[1]:
```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
import numpy as np
import os,cv2

def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
```

[2]:
```
dir_name= "dataset/faces/"
y=[];X=[];target_names=[]
person_id=0;h=w=300
n_samples=0
class_names=[]
for person_name in os.listdir(dir_name):
    # print(person_name)
    dir_path = dir_name+person_name+"/"
    class_names.append(person_name)
    for image_name in os.listdir(dir_path):
        # formulate the image path
        image_path = dir_path+image_name
        # Read the input image
        img = cv2.imread(image_path)
        # Convert into grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
        # resize image to 300*300 dimension
        resized_image= cv2.resize(gray,(h,w))
        # convert matrix to vector
        v = resized_image.flatten()
        X.append(v)
        # increase the number of samples
        n_samples =n_samples+1
        # Addinng th categorical label
        y.append(person_id)
        # adding the person name
        target_names.append(person_name)
    # Increase the person id by 1
    person_id=person_id+1
# ##############################################################################
# transform list to numpy array
y=np.array(y)
X=np.array(X)
target_names =np.array(target_names)
n_features = X.shape[1]
print(y.shape,X.shape,target_names.shape)
print("Number of sampels:",n_samples)
# Download the data, if not already on disk and load it as numpy arrays

# lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# # introspect the images arrays to find the shapes (for plotting)
# n_samples, h, w = lfw_people.images.shape
# print(n_samples, h, w)
# # for machine learning we use the 2 data directly (as relative pixel
# # positions info is ignored by this model)
# X = lfw_people.data
# n_features = X.shape[1]

# print(X.shape)
# # the label to predict is the id of the person
# y = lfw_people.target
# print(y)
# if 0 in y:
#     print("yes")
# target_names = lfw_people.target_names
# print(target_names)
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)
```

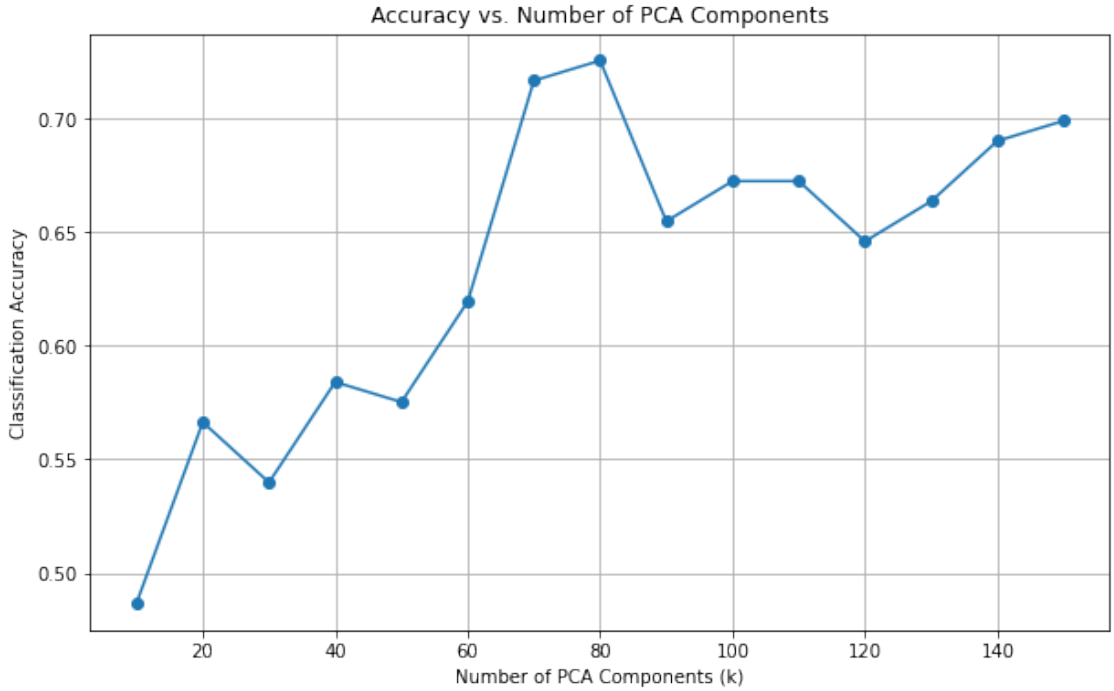
```
(450,) (450, 90000) (450,)
Number of sampels: 450
Total dataset size:
n_samples: 450
n_features: 90000
n_classes: 450
```

[6]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Range of k values (number of PCA components) to test
k_values = range(10, 160, 10)
accuracies = []

for k in k_values:
    # Apply PCA
    pca = PCA(n_components=k, svd_solver='randomized', whiten=True).fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    # Apply LDA
    lda = LinearDiscriminantAnalysis()
    lda.fit(X_train_pca, y_train)
    X_train_lda = lda.transform(X_train_pca)
    X_test_lda = lda.transform(X_test_pca)

    # Train MLP Classifier
    clf = MLPClassifier(random_state=1, hidden_layer_sizes=(10, 10),␣
 ↪max_iter=1000, verbose=False)
    clf.fit(X_train_lda, y_train)

    # Calculate accuracy
    accuracy = clf.score(X_test_lda, y_test)
    accuracies.append(accuracy)
    print(f"k={k}, Accuracy={accuracy:.2f}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.xlabel('Number of PCA Components (k)')
plt.ylabel('Classification Accuracy')
plt.title('Accuracy vs. Number of PCA Components')
plt.grid(True)
plt.show()
```

```
c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=10, Accuracy=0.49

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=20, Accuracy=0.57

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=30, Accuracy=0.54

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=40, Accuracy=0.58

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=50, Accuracy=0.58

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=60, Accuracy=0.62

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=70, Accuracy=0.72
```

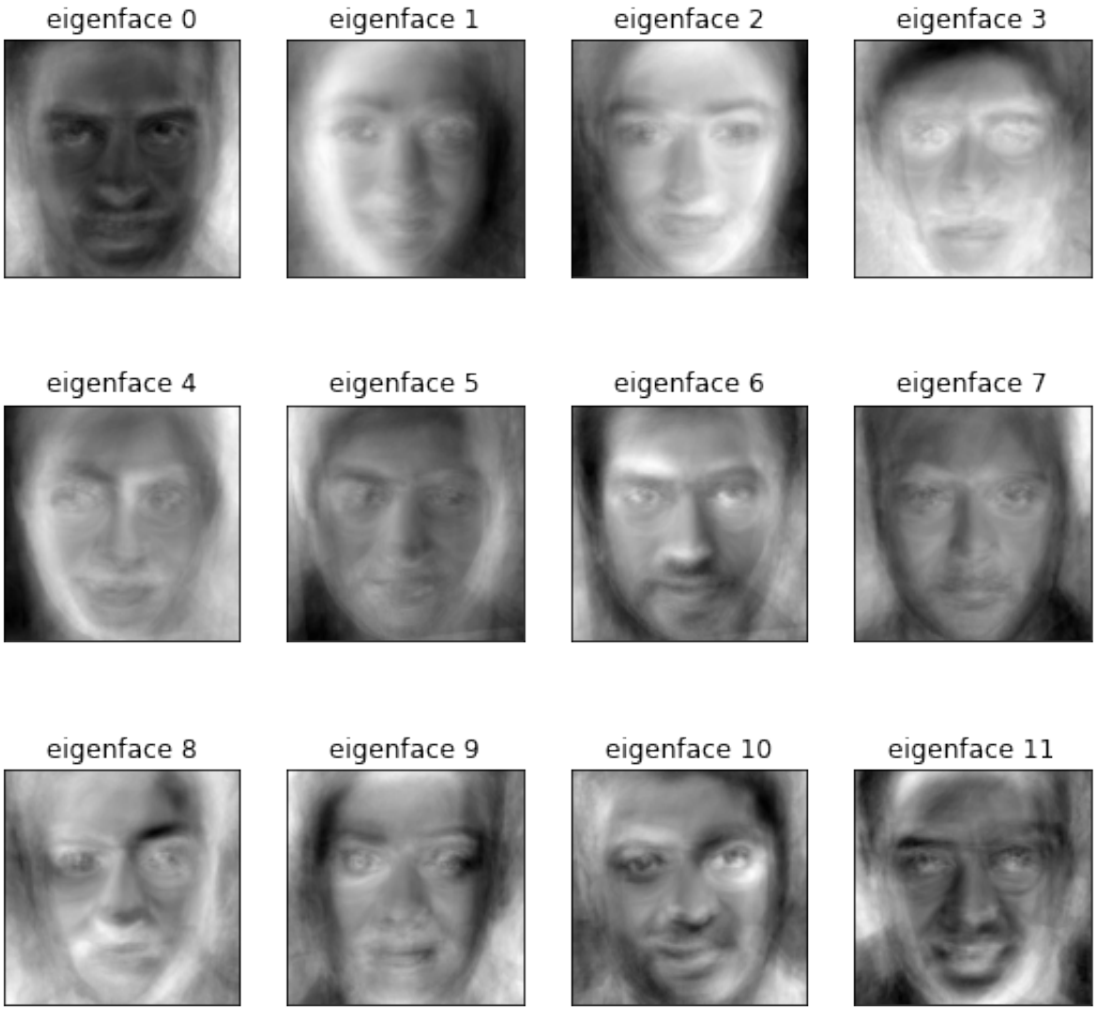
```
c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=80, Accuracy=0.73

c:\users\keith\appdata\local\programs\python\python37\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:585:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

k=90, Accuracy=0.65
k=100, Accuracy=0.67
k=110, Accuracy=0.67
k=120, Accuracy=0.65
k=130, Accuracy=0.66
k=140, Accuracy=0.69
k=150, Accuracy=0.70
```



[3]:
```
# ##############################################################################
# Split into a training set and a test set using a stratified k fold

# split into a training and testing set
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)


# ##############################################################################
# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
n_components = 150

print("Extracting the top %d eigenfaces from %d faces"% (n_components, X_train.
 ↪shape[0]))

# Applying PCA
pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True).
 ↪fit(X_train)

# Generating eigenfaces
eigenfaces = pca.components_.reshape((n_components, h, w))

# plot the gallery of the most significative eigenfaces

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)

plt.show()

print("Projecting the input data on the eigenfaces orthonormal basis")
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape,X_test_pca.shape)

# %%Compute Fisherfaces
lda = LinearDiscriminantAnalysis()
#Compute LDA of reduced data
lda.fit(X_train_pca, y_train)

X_train_lda = lda.transform(X_train_pca)
X_test_lda = lda.transform(X_test_pca)
print("Project done...")
```

```
Extracting the top 150 eigenfaces from 337 faces
```




```
Projecting the input data on the eigenfaces orthonormal basis
(337, 150) (113, 150)
Project done…
```

[4]:
```
# Training with Multi layer perceptron
clf = MLPClassifier(random_state=1, hidden_layer_sizes=(10, 10),max_iter=1000,␣
 ↪verbose=True).fit(X_train_lda, y_train)
print("Model Weights:")
model_info = [coef.shape for coef in clf.coefs_]
print(model_info)
```

```
Iteration 1, loss = 3.35711960
Iteration 2, loss = 3.29662669
Iteration 3, loss = 3.24161857
Iteration 4, loss = 3.18559509
Iteration 5, loss = 3.13004984
```

```
Iteration 6, loss = 3.07880129
Iteration 7, loss = 3.02700064
Iteration 8, loss = 2.97783803
Iteration 9, loss = 2.93103517
Iteration 10, loss = 2.88534285
Iteration 11, loss = 2.84047991
Iteration 12, loss = 2.79845635
Iteration 13, loss = 2.75538703
Iteration 14, loss = 2.71552589
Iteration 15, loss = 2.67369424
Iteration 16, loss = 2.63475561
Iteration 17, loss = 2.59665139
Iteration 18, loss = 2.55705992
Iteration 19, loss = 2.52311556
Iteration 20, loss = 2.48676237
Iteration 21, loss = 2.45408788
Iteration 22, loss = 2.42004978
Iteration 23, loss = 2.38809878
Iteration 24, loss = 2.35623838
Iteration 25, loss = 2.32520438
Iteration 26, loss = 2.29568197
Iteration 27, loss = 2.26433717
Iteration 28, loss = 2.23473870
Iteration 29, loss = 2.20489186
Iteration 30, loss = 2.17579716
Iteration 31, loss = 2.14704027
Iteration 32, loss = 2.11910315
Iteration 33, loss = 2.09076050
Iteration 34, loss = 2.06160470
Iteration 35, loss = 2.03412940
Iteration 36, loss = 2.00788599
Iteration 37, loss = 1.98101701
Iteration 38, loss = 1.95392702
Iteration 39, loss = 1.92863532
Iteration 40, loss = 1.90307538
Iteration 41, loss = 1.87785057
Iteration 42, loss = 1.85378446
Iteration 43, loss = 1.82966026
Iteration 44, loss = 1.80595560
Iteration 45, loss = 1.78284216
Iteration 46, loss = 1.75949075
Iteration 47, loss = 1.73717560
Iteration 48, loss = 1.71468006
Iteration 49, loss = 1.69324229
Iteration 50, loss = 1.67197632
Iteration 51, loss = 1.65090963
Iteration 52, loss = 1.63070291
Iteration 53, loss = 1.61080122
```

```
Iteration 54, loss = 1.59056395
Iteration 55, loss = 1.57181692
Iteration 56, loss = 1.55329316
Iteration 57, loss = 1.53462094
Iteration 58, loss = 1.51666148
Iteration 59, loss = 1.49866615
Iteration 60, loss = 1.48098624
Iteration 61, loss = 1.46370767
Iteration 62, loss = 1.44678981
Iteration 63, loss = 1.42977181
Iteration 64, loss = 1.41268604
Iteration 65, loss = 1.39609921
Iteration 66, loss = 1.37943277
Iteration 67, loss = 1.36265282
Iteration 68, loss = 1.34598203
Iteration 69, loss = 1.32945276
Iteration 70, loss = 1.31336626
Iteration 71, loss = 1.29654830
Iteration 72, loss = 1.27986368
Iteration 73, loss = 1.26384547
Iteration 74, loss = 1.24744463
Iteration 75, loss = 1.23086063
Iteration 76, loss = 1.21457823
Iteration 77, loss = 1.19855055
Iteration 78, loss = 1.18201424
Iteration 79, loss = 1.16632506
Iteration 80, loss = 1.14953471
Iteration 81, loss = 1.13329650
Iteration 82, loss = 1.11727579
Iteration 83, loss = 1.10146597
Iteration 84, loss = 1.08513044
Iteration 85, loss = 1.06948595
Iteration 86, loss = 1.05345529
Iteration 87, loss = 1.03768902
Iteration 88, loss = 1.02228746
Iteration 89, loss = 1.00730394
Iteration 90, loss = 0.99240636
Iteration 91, loss = 0.97723702
Iteration 92, loss = 0.96299640
Iteration 93, loss = 0.94800496
Iteration 94, loss = 0.93407213
Iteration 95, loss = 0.91975440
Iteration 96, loss = 0.90575606
Iteration 97, loss = 0.89203852
Iteration 98, loss = 0.87846156
Iteration 99, loss = 0.86530973
Iteration 100, loss = 0.85185011
Iteration 101, loss = 0.83925474
```

```
Iteration 102, loss = 0.82583801
Iteration 103, loss = 0.81331150
Iteration 104, loss = 0.80081894
Iteration 105, loss = 0.78870338
Iteration 106, loss = 0.77670123
Iteration 107, loss = 0.76480892
Iteration 108, loss = 0.75339706
Iteration 109, loss = 0.74204776
Iteration 110, loss = 0.73077648
Iteration 111, loss = 0.71993900
Iteration 112, loss = 0.70926402
Iteration 113, loss = 0.69853771
Iteration 114, loss = 0.68823539
Iteration 115, loss = 0.67803751
Iteration 116, loss = 0.66791968
Iteration 117, loss = 0.65803724
Iteration 118, loss = 0.64846654
Iteration 119, loss = 0.63880194
Iteration 120, loss = 0.62920508
Iteration 121, loss = 0.61980843
Iteration 122, loss = 0.61043612
Iteration 123, loss = 0.60114885
Iteration 124, loss = 0.59194390
Iteration 125, loss = 0.58303558
Iteration 126, loss = 0.57410594
Iteration 127, loss = 0.56504806
Iteration 128, loss = 0.55638319
Iteration 129, loss = 0.54749148
Iteration 130, loss = 0.53904637
Iteration 131, loss = 0.53083654
Iteration 132, loss = 0.52216742
Iteration 133, loss = 0.51388768
Iteration 134, loss = 0.50571294
Iteration 135, loss = 0.49764036
Iteration 136, loss = 0.48951939
Iteration 137, loss = 0.48161767
Iteration 138, loss = 0.47370671
Iteration 139, loss = 0.46606867
Iteration 140, loss = 0.45821928
Iteration 141, loss = 0.45068076
Iteration 142, loss = 0.44310182
Iteration 143, loss = 0.43550391
Iteration 144, loss = 0.42833070
Iteration 145, loss = 0.42128653
Iteration 146, loss = 0.41399620
Iteration 147, loss = 0.40681362
Iteration 148, loss = 0.39998497
Iteration 149, loss = 0.39305111
```

```
Iteration 150, loss = 0.38625193
Iteration 151, loss = 0.37977100
Iteration 152, loss = 0.37294651
Iteration 153, loss = 0.36646495
Iteration 154, loss = 0.35989550
Iteration 155, loss = 0.35383323
Iteration 156, loss = 0.34755819
Iteration 157, loss = 0.34125593
Iteration 158, loss = 0.33537676
Iteration 159, loss = 0.32945066
Iteration 160, loss = 0.32372696
Iteration 161, loss = 0.31808620
Iteration 162, loss = 0.31264123
Iteration 163, loss = 0.30725319
Iteration 164, loss = 0.30198433
Iteration 165, loss = 0.29682414
Iteration 166, loss = 0.29174629
Iteration 167, loss = 0.28704665
Iteration 168, loss = 0.28208040
Iteration 169, loss = 0.27770029
Iteration 170, loss = 0.27319263
Iteration 171, loss = 0.26875315
Iteration 172, loss = 0.26462875
Iteration 173, loss = 0.26035991
Iteration 174, loss = 0.25655172
Iteration 175, loss = 0.25269606
Iteration 176, loss = 0.24873296
Iteration 177, loss = 0.24513765
Iteration 178, loss = 0.24154463
Iteration 179, loss = 0.23813250
Iteration 180, loss = 0.23481382
Iteration 181, loss = 0.23143435
Iteration 182, loss = 0.22832329
Iteration 183, loss = 0.22524373
Iteration 184, loss = 0.22218573
Iteration 185, loss = 0.21934989
Iteration 186, loss = 0.21647822
Iteration 187, loss = 0.21373613
Iteration 188, loss = 0.21099380
Iteration 189, loss = 0.20842030
Iteration 190, loss = 0.20589638
Iteration 191, loss = 0.20332855
Iteration 192, loss = 0.20086987
Iteration 193, loss = 0.19862857
Iteration 194, loss = 0.19618570
Iteration 195, loss = 0.19393715
Iteration 196, loss = 0.19173923
Iteration 197, loss = 0.18968773
```

```
Iteration 198, loss = 0.18762063
Iteration 199, loss = 0.18552303
Iteration 200, loss = 0.18355990
Iteration 201, loss = 0.18159426
Iteration 202, loss = 0.17970946
Iteration 203, loss = 0.17781223
Iteration 204, loss = 0.17598682
Iteration 205, loss = 0.17420075
Iteration 206, loss = 0.17245606
Iteration 207, loss = 0.17066409
Iteration 208, loss = 0.16897221
Iteration 209, loss = 0.16723507
Iteration 210, loss = 0.16555947
Iteration 211, loss = 0.16382760
Iteration 212, loss = 0.16226434
Iteration 213, loss = 0.16062451
Iteration 214, loss = 0.15906724
Iteration 215, loss = 0.15747068
Iteration 216, loss = 0.15590521
Iteration 217, loss = 0.15446132
Iteration 218, loss = 0.15293192
Iteration 219, loss = 0.15149450
Iteration 220, loss = 0.15010518
Iteration 221, loss = 0.14870224
Iteration 222, loss = 0.14734514
Iteration 223, loss = 0.14599645
Iteration 224, loss = 0.14462928
Iteration 225, loss = 0.14333306
Iteration 226, loss = 0.14206442
Iteration 227, loss = 0.14080719
Iteration 228, loss = 0.13961790
Iteration 229, loss = 0.13836196
Iteration 230, loss = 0.13721565
Iteration 231, loss = 0.13603387
Iteration 232, loss = 0.13488353
Iteration 233, loss = 0.13381096
Iteration 234, loss = 0.13269703
Iteration 235, loss = 0.13162939
Iteration 236, loss = 0.13054410
Iteration 237, loss = 0.12950564
Iteration 238, loss = 0.12849959
Iteration 239, loss = 0.12743870
Iteration 240, loss = 0.12646845
Iteration 241, loss = 0.12553565
Iteration 242, loss = 0.12454057
Iteration 243, loss = 0.12363174
Iteration 244, loss = 0.12274146
Iteration 245, loss = 0.12180782
```

```
Iteration 246, loss = 0.12093360
Iteration 247, loss = 0.12008065
Iteration 248, loss = 0.11922785
Iteration 249, loss = 0.11837087
Iteration 250, loss = 0.11756810
Iteration 251, loss = 0.11678512
Iteration 252, loss = 0.11593972
Iteration 253, loss = 0.11519316
Iteration 254, loss = 0.11439990
Iteration 255, loss = 0.11363383
Iteration 256, loss = 0.11284638
Iteration 257, loss = 0.11214276
Iteration 258, loss = 0.11141730
Iteration 259, loss = 0.11068712
Iteration 260, loss = 0.10998056
Iteration 261, loss = 0.10923780
Iteration 262, loss = 0.10855139
Iteration 263, loss = 0.10785192
Iteration 264, loss = 0.10718500
Iteration 265, loss = 0.10652265
Iteration 266, loss = 0.10583563
Iteration 267, loss = 0.10520494
Iteration 268, loss = 0.10452994
Iteration 269, loss = 0.10386048
Iteration 270, loss = 0.10328644
Iteration 271, loss = 0.10261367
Iteration 272, loss = 0.10199802
Iteration 273, loss = 0.10139217
Iteration 274, loss = 0.10078436
Iteration 275, loss = 0.10022614
Iteration 276, loss = 0.09960856
Iteration 277, loss = 0.09904243
Iteration 278, loss = 0.09844323
Iteration 279, loss = 0.09785858
Iteration 280, loss = 0.09731441
Iteration 281, loss = 0.09675286
Iteration 282, loss = 0.09623342
Iteration 283, loss = 0.09565556
Iteration 284, loss = 0.09512675
Iteration 285, loss = 0.09459651
Iteration 286, loss = 0.09407259
Iteration 287, loss = 0.09358032
Iteration 288, loss = 0.09300901
Iteration 289, loss = 0.09251379
Iteration 290, loss = 0.09201203
Iteration 291, loss = 0.09149935
Iteration 292, loss = 0.09105558
Iteration 293, loss = 0.09049568
```

```
Iteration 294, loss = 0.09001590
Iteration 295, loss = 0.08956702
Iteration 296, loss = 0.08907558
Iteration 297, loss = 0.08858805
Iteration 298, loss = 0.08813962
Iteration 299, loss = 0.08766808
Iteration 300, loss = 0.08723279
Iteration 301, loss = 0.08676806
Iteration 302, loss = 0.08636101
Iteration 303, loss = 0.08586906
Iteration 304, loss = 0.08544676
Iteration 305, loss = 0.08501575
Iteration 306, loss = 0.08460116
Iteration 307, loss = 0.08417784
Iteration 308, loss = 0.08374357
Iteration 309, loss = 0.08332131
Iteration 310, loss = 0.08290385
Iteration 311, loss = 0.08252030
Iteration 312, loss = 0.08210664
Iteration 313, loss = 0.08172490
Iteration 314, loss = 0.08129894
Iteration 315, loss = 0.08093699
Iteration 316, loss = 0.08053476
Iteration 317, loss = 0.08016538
Iteration 318, loss = 0.07975217
Iteration 319, loss = 0.07937513
Iteration 320, loss = 0.07899104
Iteration 321, loss = 0.07863203
Iteration 322, loss = 0.07825063
Iteration 323, loss = 0.07787525
Iteration 324, loss = 0.07751700
Iteration 325, loss = 0.07715181
Iteration 326, loss = 0.07678223
Iteration 327, loss = 0.07642534
Iteration 328, loss = 0.07607324
Iteration 329, loss = 0.07572739
Iteration 330, loss = 0.07538264
Iteration 331, loss = 0.07502720
Iteration 332, loss = 0.07468835
Iteration 333, loss = 0.07431551
Iteration 334, loss = 0.07400791
Iteration 335, loss = 0.07366826
Iteration 336, loss = 0.07333461
Iteration 337, loss = 0.07299158
Iteration 338, loss = 0.07263178
Iteration 339, loss = 0.07231027
Iteration 340, loss = 0.07200412
Iteration 341, loss = 0.07165805
```

```
Iteration 342, loss = 0.07134643
Iteration 343, loss = 0.07100378
Iteration 344, loss = 0.07069072
Iteration 345, loss = 0.07037657
Iteration 346, loss = 0.07007591
Iteration 347, loss = 0.06974854
Iteration 348, loss = 0.06945294
Iteration 349, loss = 0.06912379
Iteration 350, loss = 0.06882558
Iteration 351, loss = 0.06852674
Iteration 352, loss = 0.06825161
Iteration 353, loss = 0.06794975
Iteration 354, loss = 0.06762918
Iteration 355, loss = 0.06736358
Iteration 356, loss = 0.06706341
Iteration 357, loss = 0.06678012
Iteration 358, loss = 0.06651134
Iteration 359, loss = 0.06623471
Iteration 360, loss = 0.06597798
Iteration 361, loss = 0.06565712
Iteration 362, loss = 0.06539575
Iteration 363, loss = 0.06513261
Iteration 364, loss = 0.06487515
Iteration 365, loss = 0.06460177
Iteration 366, loss = 0.06435296
Iteration 367, loss = 0.06407543
Iteration 368, loss = 0.06380517
Iteration 369, loss = 0.06356307
Iteration 370, loss = 0.06330774
Iteration 371, loss = 0.06305666
Iteration 372, loss = 0.06280971
Iteration 373, loss = 0.06256495
Iteration 374, loss = 0.06234167
Iteration 375, loss = 0.06209858
Iteration 376, loss = 0.06184077
Iteration 377, loss = 0.06159222
Iteration 378, loss = 0.06135787
Iteration 379, loss = 0.06114159
Iteration 380, loss = 0.06090400
Iteration 381, loss = 0.06066993
Iteration 382, loss = 0.06044452
Iteration 383, loss = 0.06019077
Iteration 384, loss = 0.05997387
Iteration 385, loss = 0.05975141
Iteration 386, loss = 0.05951201
Iteration 387, loss = 0.05930741
Iteration 388, loss = 0.05907470
Iteration 389, loss = 0.05887129
```

```
Iteration 390, loss = 0.05865031
Iteration 391, loss = 0.05842317
Iteration 392, loss = 0.05822068
Iteration 393, loss = 0.05801005
Iteration 394, loss = 0.05781239
Iteration 395, loss = 0.05759715
Iteration 396, loss = 0.05740676
Iteration 397, loss = 0.05719892
Iteration 398, loss = 0.05696657
Iteration 399, loss = 0.05676173
Iteration 400, loss = 0.05657123
Iteration 401, loss = 0.05636739
Iteration 402, loss = 0.05620323
Iteration 403, loss = 0.05597338
Iteration 404, loss = 0.05576023
Iteration 405, loss = 0.05556520
Iteration 406, loss = 0.05537101
Iteration 407, loss = 0.05520428
Iteration 408, loss = 0.05498679
Iteration 409, loss = 0.05479584
Iteration 410, loss = 0.05460425
Iteration 411, loss = 0.05442417
Iteration 412, loss = 0.05422994
Iteration 413, loss = 0.05404271
Iteration 414, loss = 0.05385586
Iteration 415, loss = 0.05369124
Iteration 416, loss = 0.05349711
Iteration 417, loss = 0.05330425
Iteration 418, loss = 0.05315295
Iteration 419, loss = 0.05295015
Iteration 420, loss = 0.05276845
Iteration 421, loss = 0.05258903
Iteration 422, loss = 0.05243559
Iteration 423, loss = 0.05224749
Iteration 424, loss = 0.05207472
Iteration 425, loss = 0.05188679
Iteration 426, loss = 0.05171794
Iteration 427, loss = 0.05154009
Iteration 428, loss = 0.05138327
Iteration 429, loss = 0.05119241
Iteration 430, loss = 0.05105287
Iteration 431, loss = 0.05086280
Iteration 432, loss = 0.05070339
Iteration 433, loss = 0.05054563
Iteration 434, loss = 0.05038529
Iteration 435, loss = 0.05020875
Iteration 436, loss = 0.05007000
Iteration 437, loss = 0.04987730
```

```
Iteration 438, loss = 0.04973792
Iteration 439, loss = 0.04956301
Iteration 440, loss = 0.04940108
Iteration 441, loss = 0.04925079
Iteration 442, loss = 0.04908832
Iteration 443, loss = 0.04896102
Iteration 444, loss = 0.04878096
Iteration 445, loss = 0.04862926
Iteration 446, loss = 0.04846653
Iteration 447, loss = 0.04832866
Iteration 448, loss = 0.04817263
Iteration 449, loss = 0.04801510
Iteration 450, loss = 0.04785837
Iteration 451, loss = 0.04770621
Iteration 452, loss = 0.04757603
Iteration 453, loss = 0.04741279
Iteration 454, loss = 0.04727440
Iteration 455, loss = 0.04712444
Iteration 456, loss = 0.04698423
Iteration 457, loss = 0.04681494
Iteration 458, loss = 0.04669795
Iteration 459, loss = 0.04654718
Iteration 460, loss = 0.04640458
Iteration 461, loss = 0.04626563
Iteration 462, loss = 0.04611859
Iteration 463, loss = 0.04597482
Iteration 464, loss = 0.04586577
Iteration 465, loss = 0.04569781
Iteration 466, loss = 0.04556020
Iteration 467, loss = 0.04541763
Iteration 468, loss = 0.04529005
Iteration 469, loss = 0.04514294
Iteration 470, loss = 0.04500577
Iteration 471, loss = 0.04488923
Iteration 472, loss = 0.04473984
Iteration 473, loss = 0.04460437
Iteration 474, loss = 0.04448390
Iteration 475, loss = 0.04433843
Iteration 476, loss = 0.04421614
Iteration 477, loss = 0.04408981
Iteration 478, loss = 0.04395941
Iteration 479, loss = 0.04380831
Iteration 480, loss = 0.04366284
Iteration 481, loss = 0.04354230
Iteration 482, loss = 0.04341152
Iteration 483, loss = 0.04328882
Iteration 484, loss = 0.04315958
Iteration 485, loss = 0.04303572
```

```
Iteration 486, loss = 0.04291089
Iteration 487, loss = 0.04278234
Iteration 488, loss = 0.04265840
Iteration 489, loss = 0.04251250
Iteration 490, loss = 0.04239482
Iteration 491, loss = 0.04227840
Iteration 492, loss = 0.04215680
Iteration 493, loss = 0.04202820
Iteration 494, loss = 0.04189432
Iteration 495, loss = 0.04178131
Iteration 496, loss = 0.04166722
Iteration 497, loss = 0.04153010
Iteration 498, loss = 0.04142776
Iteration 499, loss = 0.04129522
Iteration 500, loss = 0.04119002
Iteration 501, loss = 0.04109723
Iteration 502, loss = 0.04094083
Iteration 503, loss = 0.04082230
Iteration 504, loss = 0.04071473
Iteration 505, loss = 0.04058453
Iteration 506, loss = 0.04046280
Iteration 507, loss = 0.04033485
Iteration 508, loss = 0.04021030
Iteration 509, loss = 0.04008784
Iteration 510, loss = 0.03997318
Iteration 511, loss = 0.03983969
Iteration 512, loss = 0.03972643
Iteration 513, loss = 0.03959480
Iteration 514, loss = 0.03946294
Iteration 515, loss = 0.03935261
Iteration 516, loss = 0.03923478
Iteration 517, loss = 0.03910088
Iteration 518, loss = 0.03898466
Iteration 519, loss = 0.03886757
Iteration 520, loss = 0.03872499
Iteration 521, loss = 0.03861655
Iteration 522, loss = 0.03848972
Iteration 523, loss = 0.03838842
Iteration 524, loss = 0.03825330
Iteration 525, loss = 0.03812975
Iteration 526, loss = 0.03801798
Iteration 527, loss = 0.03789927
Iteration 528, loss = 0.03778728
Iteration 529, loss = 0.03766907
Iteration 530, loss = 0.03756112
Iteration 531, loss = 0.03743940
Iteration 532, loss = 0.03731656
Iteration 533, loss = 0.03720433
```

```
Iteration 534, loss = 0.03708856
Iteration 535, loss = 0.03697924
Iteration 536, loss = 0.03686946
Iteration 537, loss = 0.03674748
Iteration 538, loss = 0.03664305
Iteration 539, loss = 0.03653292
Iteration 540, loss = 0.03643478
Iteration 541, loss = 0.03630215
Iteration 542, loss = 0.03620980
Iteration 543, loss = 0.03610797
Iteration 544, loss = 0.03598586
Iteration 545, loss = 0.03588351
Iteration 546, loss = 0.03577165
Iteration 547, loss = 0.03565663
Iteration 548, loss = 0.03555378
Iteration 549, loss = 0.03547587
Iteration 550, loss = 0.03534724
Iteration 551, loss = 0.03524678
Iteration 552, loss = 0.03513869
Iteration 553, loss = 0.03502144
Iteration 554, loss = 0.03491770
Iteration 555, loss = 0.03481144
Iteration 556, loss = 0.03470670
Iteration 557, loss = 0.03459249
Iteration 558, loss = 0.03449818
Iteration 559, loss = 0.03439267
Iteration 560, loss = 0.03426852
Iteration 561, loss = 0.03417519
Iteration 562, loss = 0.03404789
Iteration 563, loss = 0.03395514
Iteration 564, loss = 0.03384119
Iteration 565, loss = 0.03372390
Iteration 566, loss = 0.03361416
Iteration 567, loss = 0.03354016
Iteration 568, loss = 0.03341099
Iteration 569, loss = 0.03331265
Iteration 570, loss = 0.03319893
Iteration 571, loss = 0.03309310
Iteration 572, loss = 0.03298976
Iteration 573, loss = 0.03289067
Iteration 574, loss = 0.03279440
Iteration 575, loss = 0.03268254
Iteration 576, loss = 0.03257627
Iteration 577, loss = 0.03248351
Iteration 578, loss = 0.03238825
Iteration 579, loss = 0.03230071
Iteration 580, loss = 0.03219733
Iteration 581, loss = 0.03207594
```

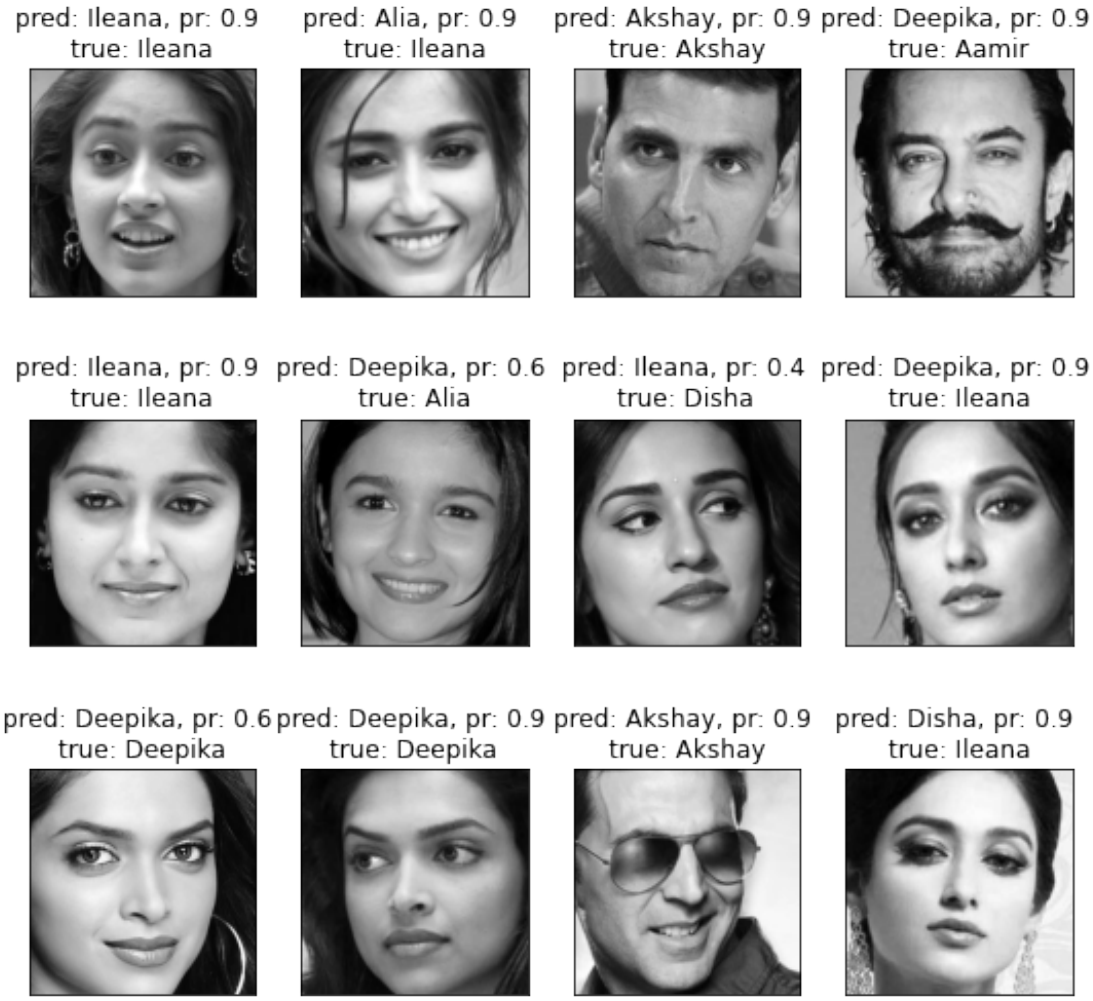
```
Iteration 582, loss = 0.03199706
Iteration 583, loss = 0.03189046
Iteration 584, loss = 0.03179096
Iteration 585, loss = 0.03169974
Iteration 586, loss = 0.03159731
Iteration 587, loss = 0.03150296
Iteration 588, loss = 0.03142145
Iteration 589, loss = 0.03131650
Iteration 590, loss = 0.03121533
Iteration 591, loss = 0.03113591
Iteration 592, loss = 0.03102897
Iteration 593, loss = 0.03092627
Iteration 594, loss = 0.03084159
Iteration 595, loss = 0.03075502
Iteration 596, loss = 0.03065294
Iteration 597, loss = 0.03055259
Iteration 598, loss = 0.03046124
Iteration 599, loss = 0.03037375
Iteration 600, loss = 0.03027414
Iteration 601, loss = 0.03018658
Iteration 602, loss = 0.03010968
Iteration 603, loss = 0.03000765
Iteration 604, loss = 0.02992904
Iteration 605, loss = 0.02982816
Iteration 606, loss = 0.02973232
Iteration 607, loss = 0.02964754
Iteration 608, loss = 0.02956100
Iteration 609, loss = 0.02945763
Iteration 610, loss = 0.02937746
Iteration 611, loss = 0.02928602
Iteration 612, loss = 0.02919458
Iteration 613, loss = 0.02909995
Iteration 614, loss = 0.02901474
Iteration 615, loss = 0.02894712
Iteration 616, loss = 0.02884490
Iteration 617, loss = 0.02876444
Iteration 618, loss = 0.02866297
Iteration 619, loss = 0.02857918
Iteration 620, loss = 0.02849035
Iteration 621, loss = 0.02840401
Iteration 622, loss = 0.02831625
Iteration 623, loss = 0.02824757
Iteration 624, loss = 0.02814970
Iteration 625, loss = 0.02806205
Iteration 626, loss = 0.02798024
Iteration 627, loss = 0.02792202
Iteration 628, loss = 0.02784087
Iteration 629, loss = 0.02774168
```

```
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
Model Weights:
[(8, 10), (10, 10), (10, 9)]
```

[5]:
```python
y_pred=[];y_prob=[]
for test_face in X_test_lda:
    prob = clf.predict_proba([test_face])[0]
    # print(prob,np.max(prob))
    class_id = np.where(prob == np.max(prob))[0][0]
    # print(class_index)
    # Find the label of the mathed face
    y_pred.append(class_id)
    y_prob.append(np.max(prob))

# Transform the data
y_pred = np.array(y_pred)


prediction_titles=[]
true_positive = 0
for i in range(y_pred.shape[0]):
    # print(y_test[i],y_pred[i])
    # true_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    # pred_name = target_names[y_pred[i]].rsplit(' ', 1)[-1]
    true_name = class_names[y_test[i]]
    pred_name = class_names[y_pred[i]]
    result = 'pred: %s, pr: %s \ntrue: %s' % (pred_name, str(y_prob[i])[0:3],␣
 ↪true_name)
    # result = 'prediction: %s \ntrue:      %s' % (pred_name, true_name)
    prediction_titles.append(result)
    if true_name==pred_name:
        true_positive =true_positive+1

print("Accuracy:",true_positive*100/y_pred.shape[0])

# # Plot results
plot_gallery(X_test, prediction_titles, h, w)
plt.show()

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.xlabel('Number of PCA Components (k)')
plt.ylabel('Classification Accuracy')
plt.title('Accuracy vs. Number of PCA Components')
plt.grid(True)
```

```
plt.show()
```

```
Accuracy: 70.79646017699115
```



[3]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
import numpy as np
import os
from PIL import Image

def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
```

```
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())

# Load dataset
dir_name = "C:/Users/CAROL MARIA DIAS/dataset/faces/"
y = []
X = []
target_names = []
person_id = 0
h = w = 300
n_samples = 0
class_names = []

for person_name in os.listdir(dir_name):
    dir_path = os.path.join(dir_name, person_name)
    class_names.append(person_name)
    for image_name in os.listdir(dir_path):
        image_path = os.path.join(dir_path, image_name)

        # Read the input image using Pillow
        img = Image.open(image_path)

        # Convert into grayscale
        gray = img.convert('L')

        # Resize image to 300x300 dimension
        resized_image = gray.resize((h, w))

        # Convert image to numpy array and flatten it
        v = np.array(resized_image).flatten()
        X.append(v)

        # Increase the number of samples
        n_samples += 1

        # Adding the categorical label
        y.append(person_id)

    # Increase the person id by 1
    person_id += 1

# Transform list to numpy array
```

```
y = np.array(y)
X = np.array(X)
target_names = np.array(target_names)
n_features = X.shape[1]

print(y.shape, X.shape, target_names.shape)
print("Number of samples:", n_samples)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Fit PCA on training data
n_components = 150  # Choose the number of components you want
pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True).
 ↪fit(X_train)

# Transform training data with PCA
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# Fit LDA on PCA-transformed training data
lda = LinearDiscriminantAnalysis()
lda.fit(X_train_pca, y_train)

# Transform training and testing data with LDA
X_train_lda = lda.transform(X_train_pca)
X_test_lda = lda.transform(X_test_pca)

# Train MLP Classifier
clf = MLPClassifier(random_state=1, hidden_layer_sizes=(10, 10), max_iter=1000,␣
 ↪verbose=False)
clf.fit(X_train_lda, y_train)

# Directory containing imposter images
imposter_folder = 'C:/Users/CAROL MARIA DIAS/dataset/Imposter/'
imposter_images = []
for filename in os.listdir(imposter_folder):
    if filename.endswith('.jpg') or filename.endswith('.png'):  # Adjust␣
 ↪extensions as needed
        img_path = os.path.join(imposter_folder, filename)
        img = Image.open(img_path)
        img = img.resize((300, 300))  # Resize to match the training data␣
 ↪dimensions
        img = img.convert('L')  # Ensure it's grayscale, like the training␣
 ↪images
```

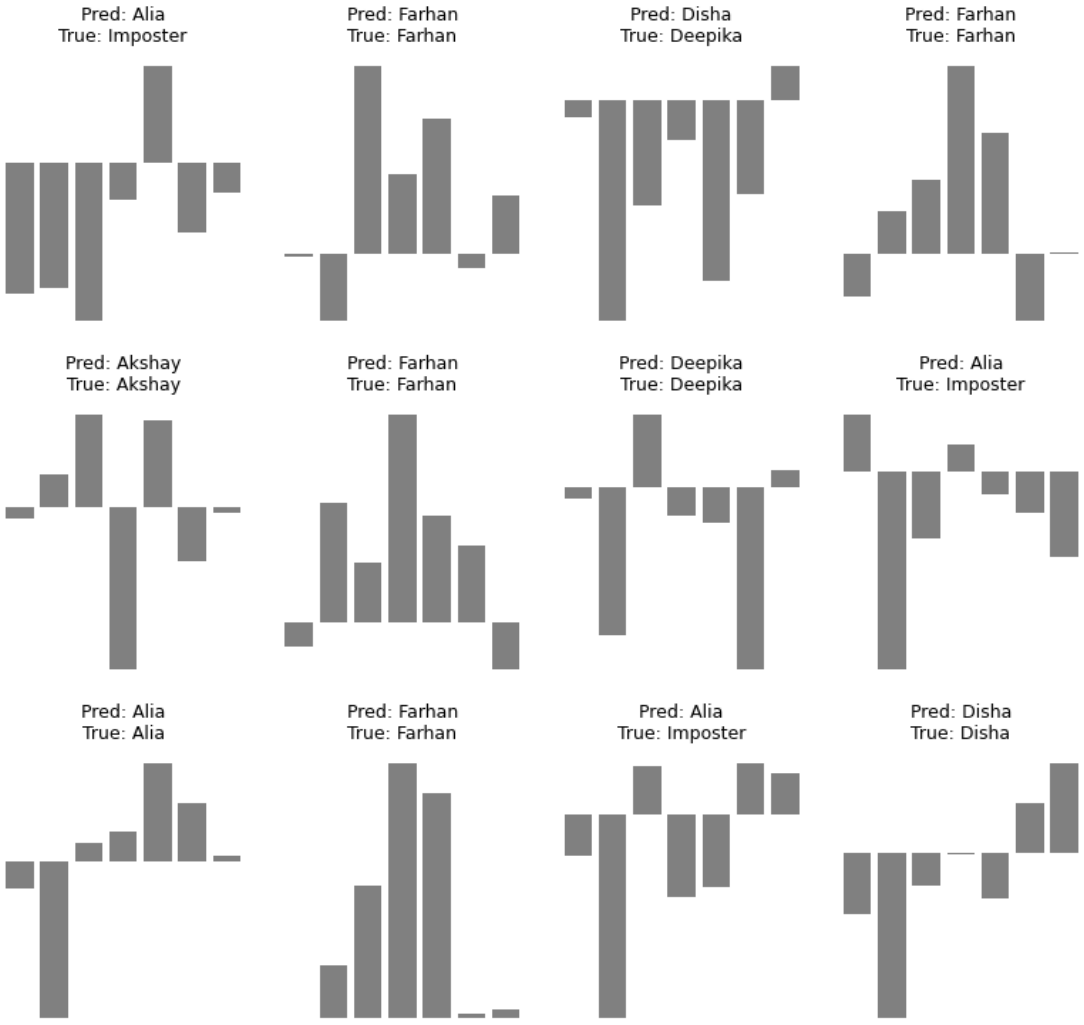
```
        img_array = np.array(img).flatten()  # Flatten the image
        imposter_images.append(img_array)

# Convert the imposter images to a numpy array
X_imposters = np.array(imposter_images)
y_imposters = np.full(len(X_imposters), -1)

# Apply PCA and LDA transformations to match training feature dimensions
X_imposters_pca = pca.transform(X_imposters)
X_imposters_lda = lda.transform(X_imposters_pca)

print("Number of imposter images:", len(X_imposters))
print("Shape of X_imposters_lda:", X_imposters_lda.shape)

# Combine original test data with imposter data
X_test_combined = np.vstack((X_test_lda, X_imposters_lda))
y_test_combined = np.concatenate((y_test, y_imposters))

# Assuming you've already combined the data:
# X_test_combined = np.vstack((X_test_lda, X_imposters_lda))
# y_test_combined = np.concatenate((y_test, y_imposters))

# Step 1: Prediction
y_pred = clf.predict(X_test_combined)
y_prob = clf.predict_proba(X_test_combined)

# Step 2: Evaluation
from sklearn.metrics import classification_report, confusion_matrix

# Calculate overall accuracy
accuracy = np.mean(y_pred == y_test_combined)
print(f"Overall Accuracy: {accuracy:.2f}")

# Generate classification report
print(classification_report(y_test_combined, y_pred))

# Generate confusion matrix
cm = confusion_matrix(y_test_combined, y_pred)
print("Confusion Matrix:")
print(cm)

# Step 3: Analysis
# Calculate False Accept Rate (FAR) and False Reject Rate (FRR)
imposter_indices = y_test_combined == -1
genuine_indices = y_test_combined != -1

far = np.mean(y_pred[imposter_indices] != -1)
```

```
frr = np.mean(y_pred[genuine_indices] == -1)

print(f"False Accept Rate: {far:.2f}")
print(f"False Reject Rate: {frr:.2f}")

# Visualize some results after LDA transformation
n_row = 3
n_col = 4
plt.figure(figsize=(2 * n_col, 2.5 * n_row))
for i in range(n_row * n_col):
    plt.subplot(n_row, n_col, i + 1)
    idx = np.random.randint(len(X_test_combined))

    # Plot the reduced features as a bar plot
    plt.bar(range(len(X_test_combined[idx])), X_test_combined[idx],␣
 ↪color='gray')
    pred_name = class_names[y_pred[idx]] if y_pred[idx] != -1 else "Imposter"
    true_name = class_names[y_test_combined[idx]] if y_test_combined[idx] != -1␣
 ↪else "Imposter"
    plt.title(f"Pred: {pred_name}\nTrue: {true_name}", size=9)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

```
(400,) (400, 90000) (0,)
Number of samples: 400
Number of imposter images: 49
Shape of X_imposters_lda: (49, 7)
Overall Accuracy: 0.49
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00        49
           0       0.80      0.53      0.64        15
           1       0.93      0.81      0.87        16
           2       0.52      0.85      0.65        13
           3       0.21      1.00      0.34         8
           4       0.73      0.85      0.79        13
           5       0.47      0.67      0.55        12
           6       0.21      0.50      0.30         8
           7       0.71      0.67      0.69        15

    accuracy                           0.49       149
   macro avg       0.51      0.65      0.54       149
weighted avg       0.42      0.49      0.43       149

Confusion Matrix:
```

```
[[ 0  0  0  3 28  1  5 12  0]
 [ 0  8  1  4  0  1  0  0  1]
 [ 0  2 13  0  0  0  0  0  1]
 [ 0  0  0 11  0  0  1  1  0]
 [ 0  0  0  0  8  0  0  0  0]
 [ 0  0  0  0  2 11  0  0  0]
 [ 0  0  0  0  0  0  8  2  2]
 [ 0  0  0  0  1  0  3  4  0]
 [ 0  0  0  3  0  2  0  0 10]]
False Accept Rate: 1.00
False Reject Rate: 0.00
```

```
C:\Users\CAROL MARIA DIAS\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\CAROL MARIA DIAS\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\CAROL MARIA DIAS\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```


Pred: Alia
True: Imposter
Pred: Farhan
True: Farhan
Pred: Disha
True: Deepika
Pred: Farhan
True: Farhan
Pred: Akshay
True: Akshay
Pred: Farhan
True: Farhan
Pred: Deepika
True: Deepika
Pred: Alia
True: Imposter
Pred: Alia
True: Alia
Pred: Farhan
True: Farhan
Pred: Alia
True: Imposter
Pred: Disha
True: Disha

[ ]:

[ ]:

[ ]:

[ ]: