```
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, recall_score, f1_score, classification_report
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from base64 import b64decode, b64encode
import cv2
import numpy as np
import matplotlib.pyplot as plt
import html
import pandas as pd
import seaborn as sns
from google.colab import files, drive
import os
import pickle as pk

drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
def tirar_foto(quality=0.8, texto_botao="Capturar"):
  js = Javascript('''
    async function takePhoto(qual, texto) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = texto;
      div.appendChild(capture);

      // Abre a câmera
      const stream = await navigator.mediaDevices.getUserMedia({video: true});
      // Mostra a saída da câmera
      const video = document.createElement('video');
      video.style.display = 'block';
      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', qual);
    }
    ''')
  display(js)
  return eval_js('takePhoto({}, "{}")'.format(quality, texto_botao))


def crop_square(img, size, interpolation=cv2.INTER_AREA):
    h, w = img.shape[:2]
    min_size = np.amin([h,w])

    # Centralize and crop
    crop_img = img[int(h/2-min_size/2):int(h/2+min_size/2), int(w/2-min_size/2):int(w/2+min_size/2)]
    resized = cv2.resize(crop_img, (size, size), interpolation=interpolation)

    return resized
```

```
def crop_rate(img, x,y,largura,altura, largura_lfw = 94, altura_lfw = 125, interpolation=cv2.INTER_AREA): #cv2.INTER_AREA
      razao_aspecto = altura_lfw/largura_lfw
      centro_x = x + largura/2
      centro_y = y + altura/2
      area = largura*altura
      largura_adj = np.sqrt(area/razao_aspecto)
      altura_adj = razao_aspecto*largura_adj
      x_min = int(np.floor(centro_x-largura_adj/2))
      x_max = int(np.ceil(centro_x+largura_adj/2))
      y_min = int(np.floor(centro_y-altura_adj/2 + 0.5))
      y_max = int(np.ceil(centro_y+altura_adj/2 + 0.5))
      if y_min <0:
         y_max -= y_min
         y_min = 0
      if x_min <0:
         x_max -= x_min
         x_min = 0
      # Centralize and crop
      crop_img = img[y_min:y_max, x_min:x_max]
      img_lfw = cv2.resize(crop_img, (largura_lfw, altura_lfw), interpolation=interpolation)

      print(img_lfw.shape)
      return img_lfw
```

```
haar_face_cascade = cv2.CascadeClassifier(cv2.samples.findFile(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'))
```

```
file_path = '/content/gdrive/MyDrive/AprendizadoMaquinasI/2024/ThisIsMe/Dataset/'

with open(file_path+'turma_data.pkl', 'rb') as arquivo:
    # Carregar o objeto do arquivo pickle
    turma_data = pk.load(arquivo)

with open(file_path+'turma_target.pkl', 'rb') as arquivo:
    # Carregar o objeto do arquivo pickle
    turma_target = pk.load(arquivo)

with open(file_path+'turma_target_names.pkl', 'rb') as arquivo:
    # Carregar o objeto do arquivo pickle
    turma_target_names = pk.load(arquivo)

altura = 125
largura = 94

turma_images = turma_data.reshape(len(turma_data), altura, largura)
```

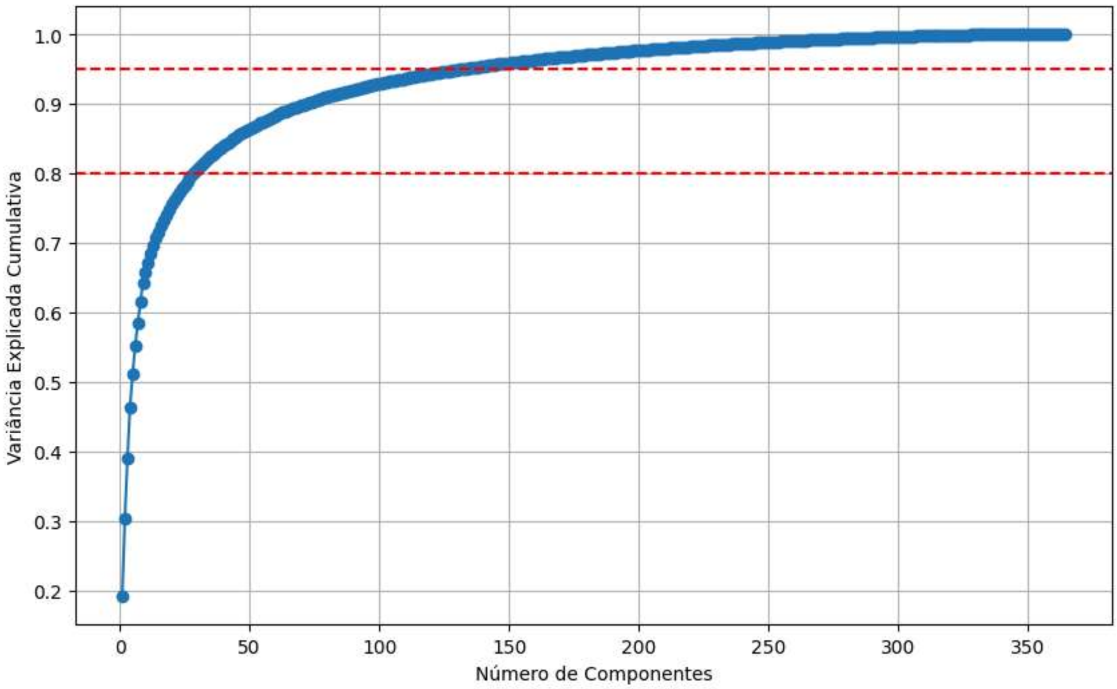
```
len(turma_target_names)
```

```
46
```

```
len(turma_data)
```

```
455
```

```
# Dividir os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(turma_data, turma_target, test_size=0.2, random_state=42)
n_components = len(X_train)

#achando o numero de componentes
nComponentes = PCA(n_components=n_components,svd_solver='randomized', whiten=True)
nComponentes.fit(X_train)
cumsum=np.cumsum(nComponentes.explained_variance_ratio_)

cumulative_variance_explained = np.cumsum(nComponentes.explained_variance_ratio_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, n_components + 1), cumulative_variance_explained, marker='o')
plt.axhline(y=0.95, color='r', linestyle='--', label='95% Variância Explicada')
plt.axhline(y=0.8, color='r', linestyle='--', label='80% Variância Explicada')
plt.xlabel('Número de Componentes')
plt.ylabel('Variância Explicada Cumulativa')
plt.grid(True)
plt.show()

qtd_pcs_80 = len(cumulative_variance_explained[cumulative_variance_explained <= 0.8])
qtd_pcs_95 = len(cumulative_variance_explained[cumulative_variance_explained <= 0.95])

print(f'Quantidade de PCs que explicam 80% da variância: {qtd_pcs_80}')
print(f'Quantidade de PCs que explicam 95% da variância: {qtd_pcs_95}')


# Aplicar PCA
pca = PCA(n_components=qtd_pcs_95,svd_solver='randomized', whiten=True).fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```



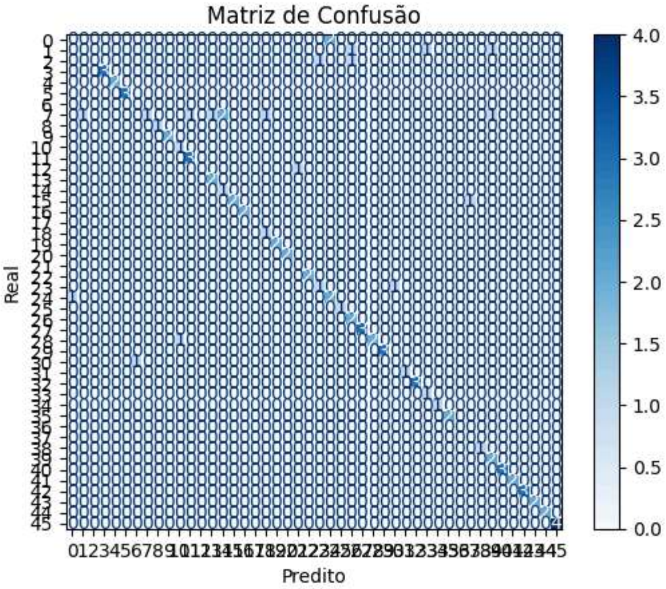
```
Quantidade de PCs que explicam 80% da variância: 28
Quantidade de PCs que explicam 95% da variância: 131
```

```
# Treinar o modelo SVM
param_grid = {
    'C': [1e3, 5e3, 1e4, 5e4, 1e5],
    'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
}

#validacao cruzada
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)

print("Melhor estimador encontrado pelo grid search:")
print(clf.best_estimator_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated class in y has only
  warnings.warn(
Melhor estimador encontrado pelo grid search:
SVC(C=1000.0, class_weight='balanced', gamma=0.001)
```

```
# Fazer previsões nos dados de teste
y_pred = clf.predict(X_test_pca)

# Avaliar o modelo
#print(classification_report(y_test, y_pred, target_names=turma_target_names))
#print(confusion_matrix(y_test, y_pred, labels=range(len(turma_target_names))))
cm=confusion_matrix(y_test, y_pred, labels=range(len(turma_target_names)))
# Mostrar a matriz de confusão
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(len(turma_target_names)))
plt.figure(figsize=(12, 12))
disp.plot(cmap=plt.cm.Blues, values_format='.4g')
plt.title('Matriz de Confusão')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.show()
```

```
<Figure size 1200x1200 with 0 Axes>
```

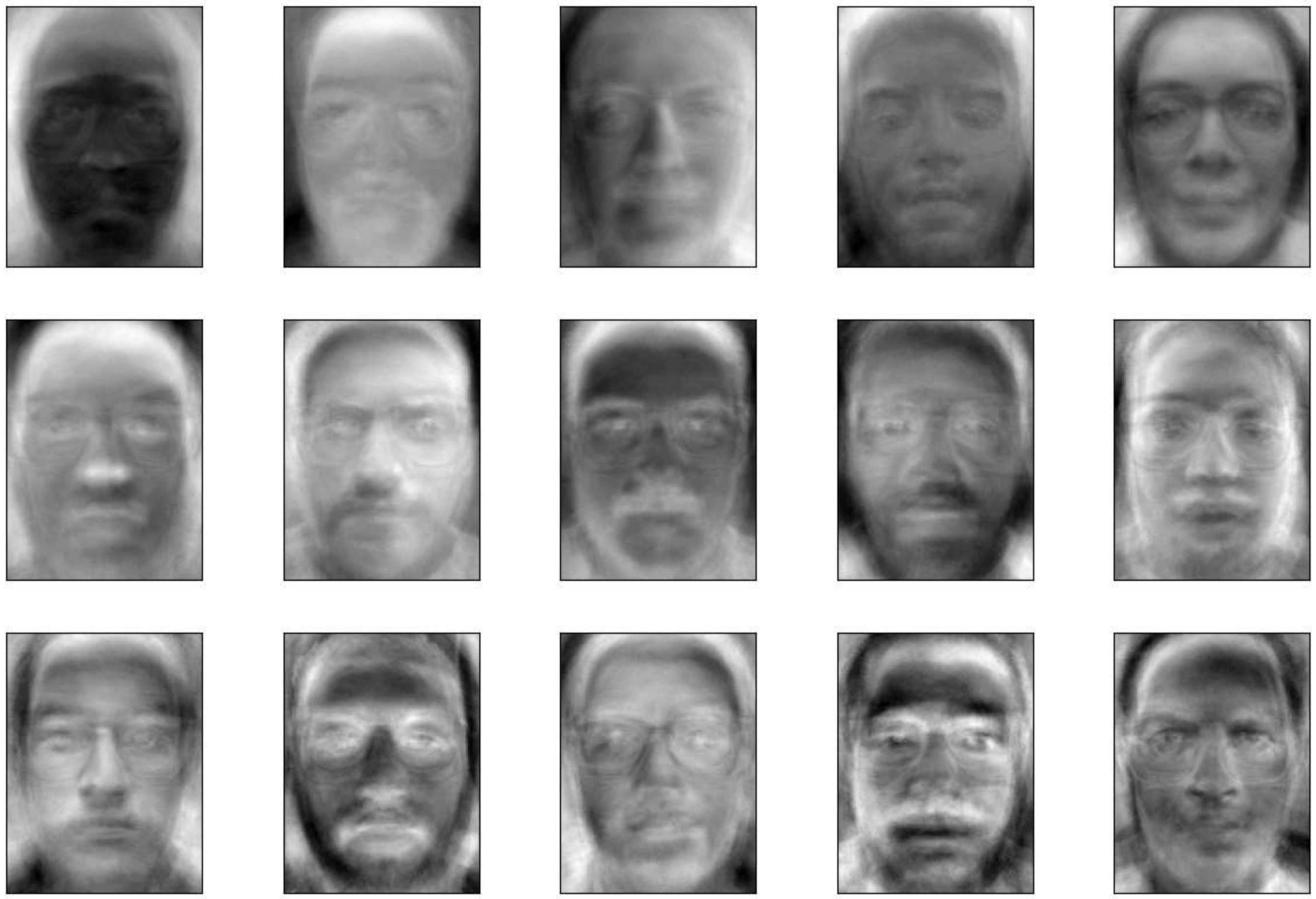


```
# Calcular as métricas
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

# Exibir as métricas
print(f'Acurácia: {accuracy:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')
```

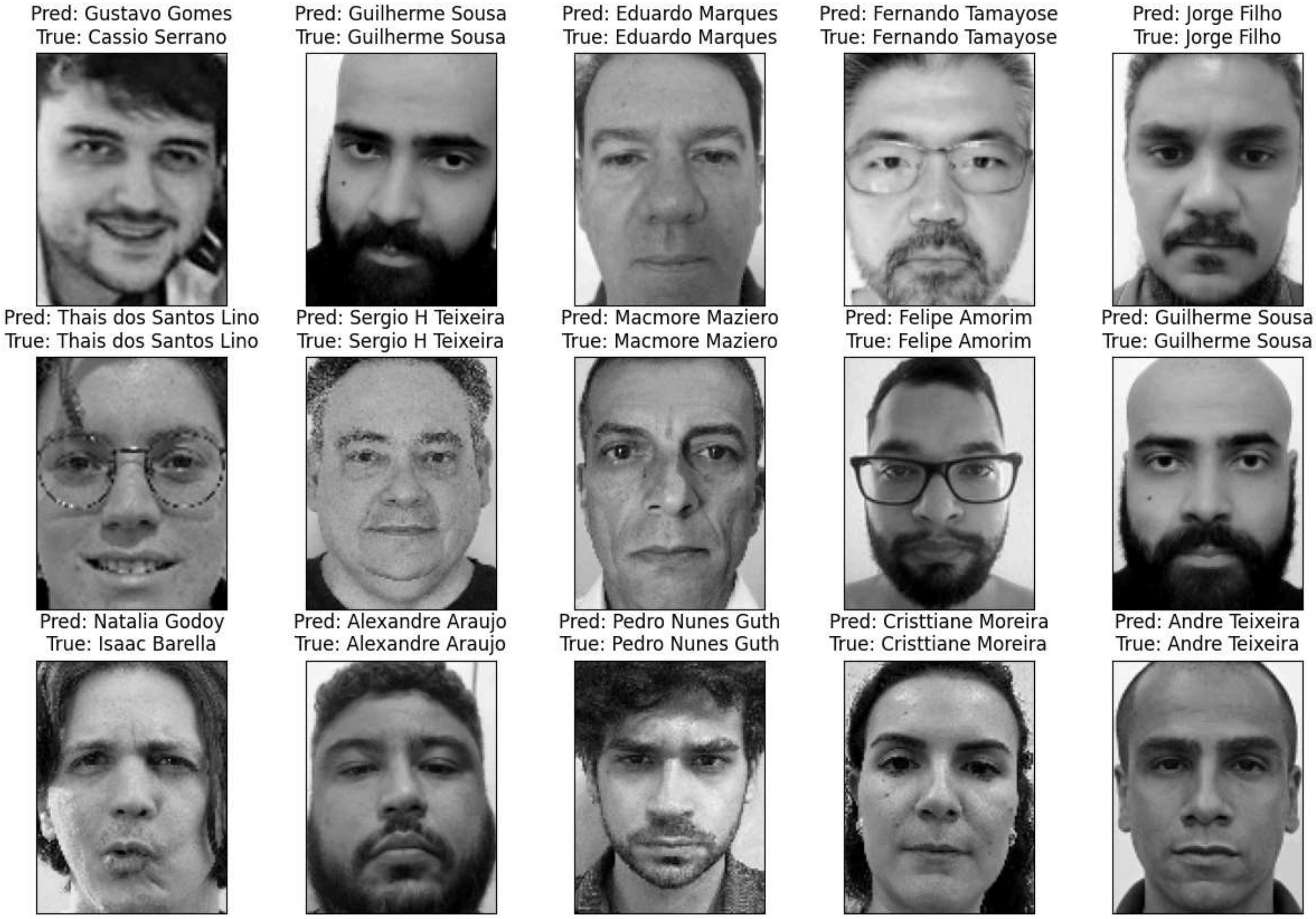
```
Acurácia: 0.78
Recall: 0.76
F1 Score: 0.71
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Plotar as primeiras eigenfaces
eigenfaces = pca.components_.reshape((qtd_pcs_95, altura, largura))

fig, axes = plt.subplots(3, 5, figsize=(15, 10))
for i, ax in enumerate(axes.flat):
    ax.imshow(eigenfaces[i], cmap='gray')
    ax.set_xticks([])
    ax.set_yticks([])
plt.suptitle("Primeiras 15 Eigenfaces")
plt.show()
```

Primeiras 15 Eigenfaces



```
# Plotar algumas imagens de teste com as previsões
fig, axes = plt.subplots(3, 5, figsize=(15, 10))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape(altura, largura), cmap='gray')
    ax.set_title(f"Pred: {turma_target_names[y_pred[i]]}\nTrue: {turma_target_names[y_test[i]]}")
    ax.set_xticks([])
    ax.set_yticks([])
plt.show()
```


Pred: Gustavo Gomes
True: Cassio Serrano
Pred: Guilherme Sousa
True: Guilherme Sousa
Pred: Eduardo Marques
True: Eduardo Marques
Pred: Fernando Tamayose
True: Fernando Tamayose
Pred: Jorge Filho
True: Jorge Filho
Pred: Thais dos Santos Lino
True: Thais dos Santos Lino
Pred: Sergio H Teixeira
True: Sergio H Teixeira
Pred: Macmore Maziero
True: Macmore Maziero
Pred: Felipe Amorim
True: Felipe Amorim
Pred: Guilherme Sousa
True: Guilherme Sousa
Pred: Natalia Godoy
True: Isaac Barella
Pred: Alexandre Araujo
True: Alexandre Araujo
Pred: Pedro Nunes Guth
True: Pedro Nunes Guth
Pred: Cristtiane Moreira
True: Cristtiane Moreira
Pred: Andre Teixeira
True: Andre Teixeira

```
try:
  imagem_urlb64 = tirar_foto()
  imbytes = b64decode(imagem_urlb64.split(',')[1])
  im = cv2.imdecode(np.frombuffer(imbytes, dtype=np.uint8), flags=1)
  # plt.imshow(im, cmap='gray'),plt.title('Imagem capturada')
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))

gray = cv2.cvtColor(im,cv2.COLOR_RGB2GRAY)
# Reconhecimento de faces

rostos = haar_face_cascade.detectMultiScale(im)

# print(f'{rostos.shape[0]} rosto(s) detectado(s)')
n_rostos = rostos.shape[0]

# print(f'\n A matriz rostos:\n {rostos}')
# Colocando o retângulo ao redor da face reconhecida
for (x, y, largura, altura) in rostos:
  im_r = cv2.rectangle(im,(x,y),(x+largura,y+altura),(255,0,0),2)
#Plotando a imagem
```