HW1_Q3

November 11, 2022

1 HW1 - Q3: Face Recognition with Eigenfaces (40 points)

Keywords: Principal Component Analysis (PCA), Eigenvalues and Eigenvectors

About the dataset:
Labeled Faces in the Wild dataset consists of face photographs designed for studying the problem of unconstrained face recognition. The original dataset contains more than 13,000 images of faces collected from the web.

Agenda: * In this programming challenge, you will be performing face recognition on the *Labeled Faces in the Wild* dataset using PyTorch. * First, you will do Principal Component Analysis (PCA) on the image dataset. PCA is used for dimentionality reduction which is a type of unsupervised learning. * You will be applying PCA on the dataset to extract the principal components (Top k *eigenvalues*). * As you will see eventually, the reconstruction of faces from these *eigenvalues* will give us the *eigen-faces* which are the most representative features of most of the images in the dataset. * Finally, you will train a simple PyTorch Neural Network model on the modified image dataset. * This trained model will be used prediction and evaluation on a test set.

Note: * Run all the cells in order. * **Do not edit** the cells marked with !!DO NOT EDIT!! * Only **add your code** to cells marked with !!!! YOUR CODE HERE !!!! * Do not change variable names, and use the names which are suggested.

---

[1]:
```
# !!DO NOT EDIT!!
# loading the dataset directly from the scikit-learn library (can take about␣
↪3-5 mins)
import numpy as np
from sklearn.datasets import fetch_lfw_people
dataset = fetch_lfw_people(min_faces_per_person=80)

# each 2D image is of size 62 x 47 pixels, represented by a 2D array.
# the value of each pixel is a real value from 0 to 255.
count, height, width = dataset.images.shape
print('The dataset type is:',type(dataset.images))
print('The number of images in the dataset:',count)
print('The height of each image:',height)
print('The width of each image:',width)
```

```
# sklearn also gives us a flattened version of the images which is a vector of␣
↪size 62 x 47 = 2914.
# we can directly use that for our exercise
print('The shape of data is:',dataset.data.shape)
```

```
The dataset type is: <class 'numpy.ndarray'>
The number of images in the dataset: 1140
The height of each image: 62
The width of each image: 47
The shape of data is: (1140, 2914)
```

For optimum performance, we have only considered people who have more than 80 images. This restriction notably reduces the size of the dataset.
Now let us look at the labels of the people present in the dataset

[2]:
```
# !!DO NOT EDIT!!
# create target label - target name pairs
targets = [(x,y) for x,y in zip(range(len(np.unique(dataset.target))), dataset.
↪target_names)]
print('The target labels and names are:\n', targets)
```

```
The target labels and names are:
 [(0, 'Colin Powell'), (1, 'Donald Rumsfeld'), (2, 'George W Bush'), (3,
'Gerhard Schroeder'), (4, 'Tony Blair')]
```

1.0.1 (a) Preprocessing: Using the `train_test_split` API from sklearn, split the data into train and test dataset in the ratio 3:1. Use `random_state=42`.

1.0.2 For better performance, it is recommended to normalize the features which can have different ranges with huge values. As all our features here are in the range [0,255], it is not explicitly needed here. However, it is a good exercise. Use the `StandardScaler` class from sklearn and use that to normalize X_train and X_test. Validate and show your result by printing the first 5 columns of 5 images of X_train (This result can vary from pc to pc). (5 points)

[3]:
```
# !!DO NOT EDIT!!
X = dataset.data
y = dataset.target
```

[4]:
```
#######
# !!!! YOUR CODE HERE !!!!

# output variable names -  X_train, X_test, y_train, y_test
#######

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=.33, random_state=42)

# Rescale
sc = StandardScaler()
sc.fit(X_train)

X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# First 5 instances and 5 features
X_train[:5, :5]
```

[4]:
```
array([[-0.452481  ,  1.6899482 ,  2.5489297 ,  2.2936993 ,  2.2403533 ],
       [ 1.6113589 ,  1.3253247 ,  1.2654899 ,  1.2568955 ,  1.172775  ],
       [-0.452481  , -0.49779323, -0.5414581 , -0.55885416, -0.53747517],
       [-0.452481  ,  0.3833804 ,  0.7194652 ,  0.26306805, -0.17630434],
       [-0.452481  , -0.49779323, -0.5414581 , -0.55885416, -0.5799659 ]],
      dtype=float32)
```

1.0.3 (b) Dimentionality reduction : In this section, use the PCA API from sklearn to extract the top 100 principal components of the image matrix and fit it on the training dataset. We can then visualize some of the top few components as an image (eigenfaces). (5 points)

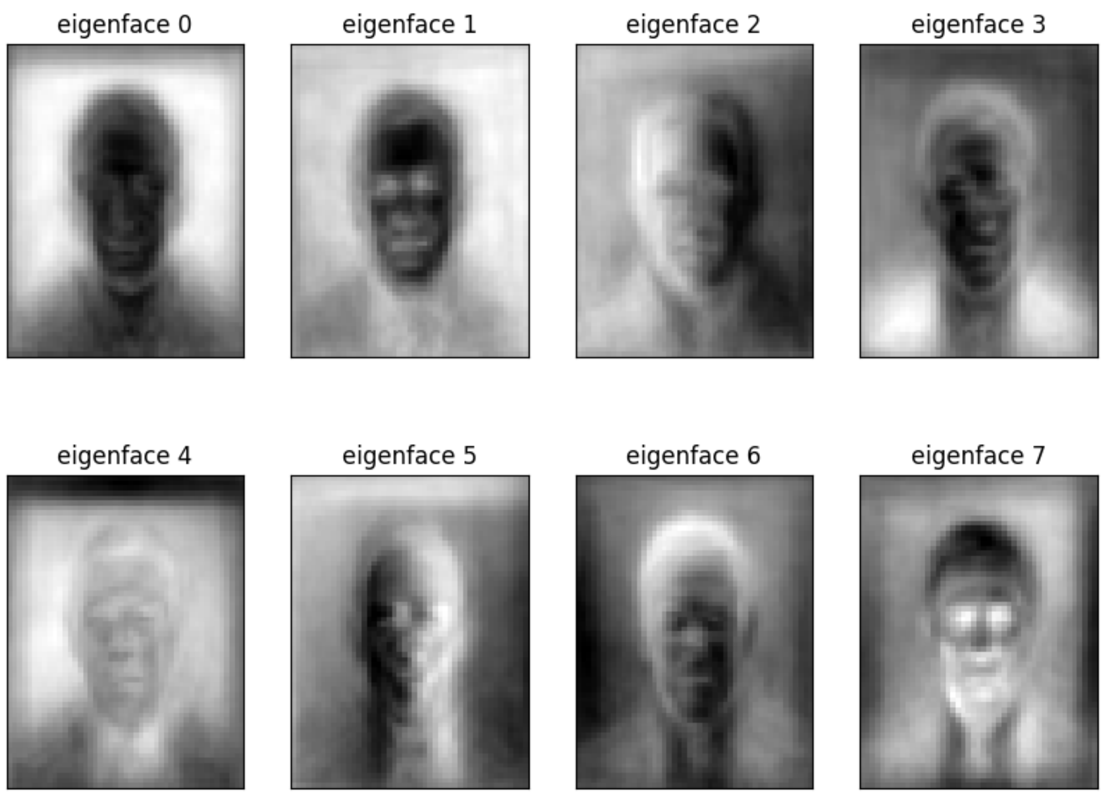
[5]:
```python
from sklearn.decomposition import PCA

n_components=100
pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True)
X_train_pca = pca.fit_transform(X_train)
```

1.0.4 Now we will plot the most representative eigenfaces:

[6]:
```python
# !!DO NOT EDIT!!
# Helper function to plot
import matplotlib.pyplot as plt
def plot_gallery(images, titles, height, width, n_row=2, n_col=4):
    plt.figure(figsize=(2* n_col, 3 * n_row))
    plt.subplots_adjust(bottom=0, left=0.01, right=0.99, top=0.90, hspace=0.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((height, width)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
```

```
        plt.xticks(())
        plt.yticks(())
```

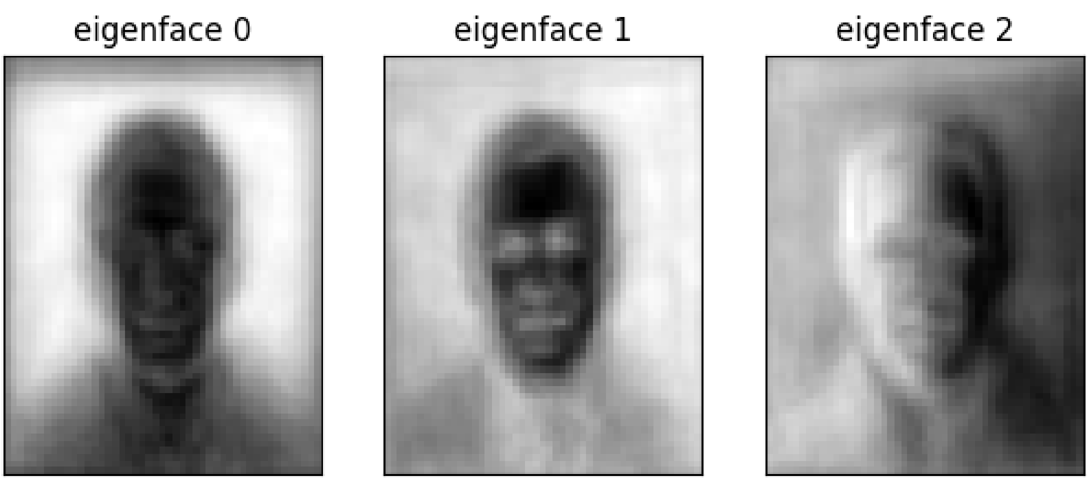
[7]:
```
# !!DO NOT EDIT!!
# get the 100 eigen faces and reshape them to original image size which is 62 x␣
↪47 pixels
eigenfaces = pca.components_.reshape((n_components, height, width))

# plot the top 8 eigenfaces
eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, height, width)

plt.show()
```



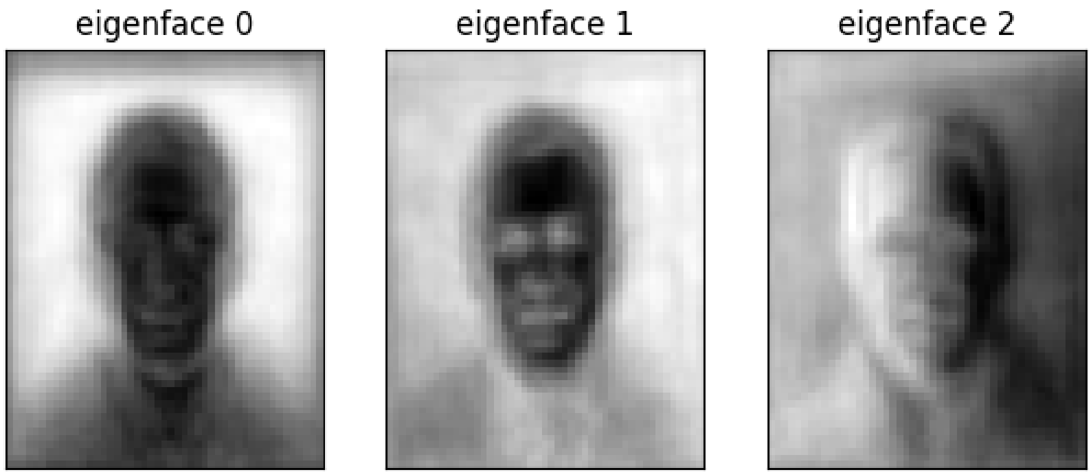
1.0.5 (c) Face reconstruction: In this section, we will reconstruct an image from its point projected on the principal component basis. Project the first three faces on the eigenvector basis using PCA models trained with varying number of principal components. Using the projected points, reconstruct the faces, and visualize the images. Your final output should be a 3×5 image matrix, where the rows are the data points, and the columns correspond to original image and reconstructed image for n_components= $[10, 100, 150, 500]$. (15 points)

[8]:
```
n_components = [5, 100, 150, 500]
eigenface_titles = ["eigenface %d" % i for i in range(3)]

for n in n_components:

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

    # Get eigenfaces
    eigenfaces = pca.components_.reshape((n, height, width))

    # Plot
    print(f'n_components={n}')
    plot_gallery(eigenfaces[:3], eigenface_titles,
                 height, width, n_row=1, n_col=3)
    plt.show()
```

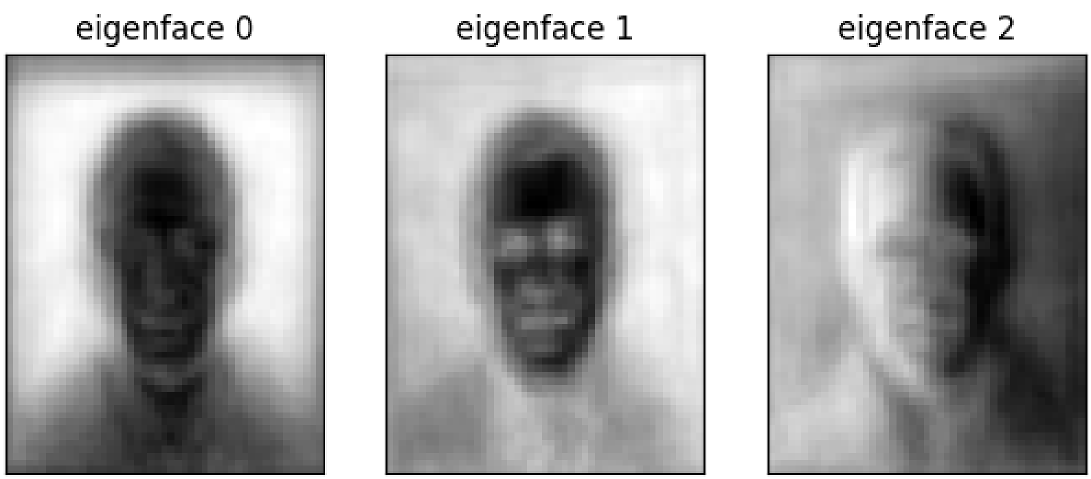
```
n_components=5
```



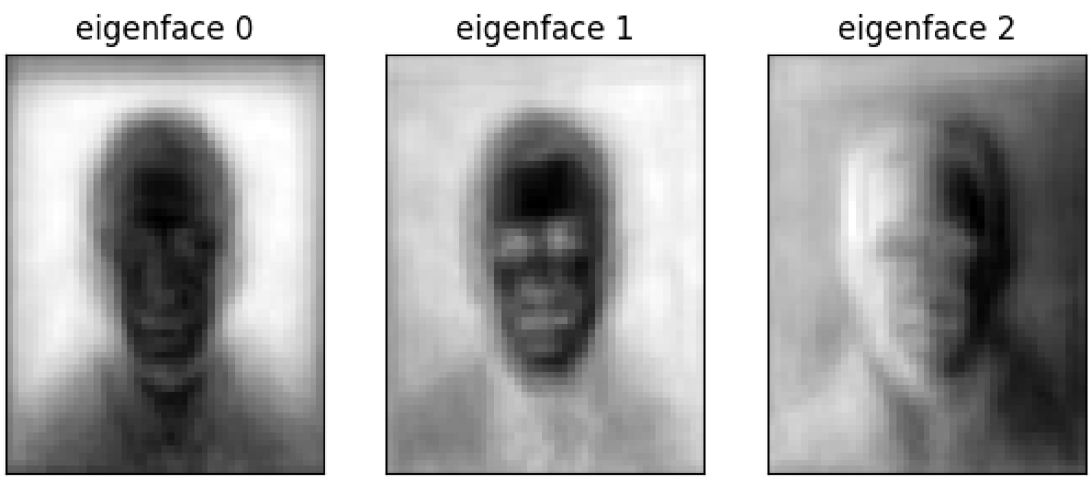

```
n_components=100
```



n_components=150



n_components=500



1.0.6 (d) Prediction: In this section, we will train a neural network classifier in PyTorch on the transformed dataset. This classifier will help us with the face recognition task. Complete each of the steps below.

1.0.7 For PyTorch reference see documentation. (15 points)

[9]:
```
# !!DO NOT EDIT!!
# define imports here
import torch
import torch.nn as nn
```

1.0.8 Before we start training, we need to transform the training and test dataset to reduced forms (100 dimensions) using the pca function defined in (b).

1.0.9 we will also need to move the train and test dataset to torch tensors in order to work with pytorch.

[10]:
```
# 1. project X_train and X_test on orthonormal basis using the PCA API
↪initialized in part (b).
n_components = 100

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

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

# 2. now convert X_train_pca, X_test_pca, y_train and y_test to torch.tensor.
↪For y_train and y_test, set dtype=torch.long
```

```
# output variable names -  X_train_pca_torch, X_test_pca_torch, y_train_torch,␣
↪y_test_torch
X_train_pca_torch = torch.from_numpy(X_train_pca)
X_test_pca_torch = torch.from_numpy(X_test_pca)

y_train_torch = torch.tensor(y_train, dtype=torch.long)
y_test_torch = torch.tensor(y_test, dtype=torch.long)
```

[11]:
```
# 3. We will implement a simple multilayer perceptron (MLP) in pytorch with one␣
↪hidden layer.
# Using this neural network model, we will train on the transformed dataset.
class MLP(torch.nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        # Initalize various layers of MLP as instructed below
        self.seq = torch.nn.Sequential(
            # DO: initialze two linear layers: 100 -> 1024  and 1024-> 5
            nn.Linear(100, 1024),
            nn.Linear(1024, 5),

            # DO: initialize relu activation function
            torch.nn.ReLU(),

            # DO: initialize LogSoftmax
            torch.nn.LogSoftmax()
        )

    def forward(self, x):
        # DO: define the feedforward algorithm of the model and return the␣
↪final output
        return self.seq(x)
```

[12]:
```
# 4. create an instance of the MLP class here
model = MLP()

# 5. define loss (use negative log likelihood loss: torch.nn.NLLLoss)
criterion = torch.nn.NLLLoss()

# 6. define optimizer (use torch.optim.SGD (Stochastic Gradient Descent)).
# Set learning rate to 1e-1 and also set model parameters
optimizer = torch.optim.SGD(model.parameters(), lr=1e-1)

# !!DO NOT EDIT!!
# 7. train the classifier on the PCA-transformed training data for 500 epochs
# This part is already implemented.
# Go through each step carefully and understand what it does.
for epoch in range(501):
```

```
    # reset gradients
    optimizer.zero_grad()

    # predict
    output=model(X_train_pca_torch)

    # calculate loss
    loss=criterion(output, y_train_torch)

    # backpropagate loss
    loss.backward()

    # performs a single gradient update step
    optimizer.step()

    if epoch%50==0:
        print('Epoch: {}, Loss: {:.3f}'.format(epoch, loss.item()))
```

```
/home/rphammonds/projects/dsc210/.env/lib/python3.10/site-
packages/torch/nn/modules/container.py:204: UserWarning: Implicit dimension
choice for log_softmax has been deprecated. Change the call to include dim=X as
an argument.
  input = module(input)
```

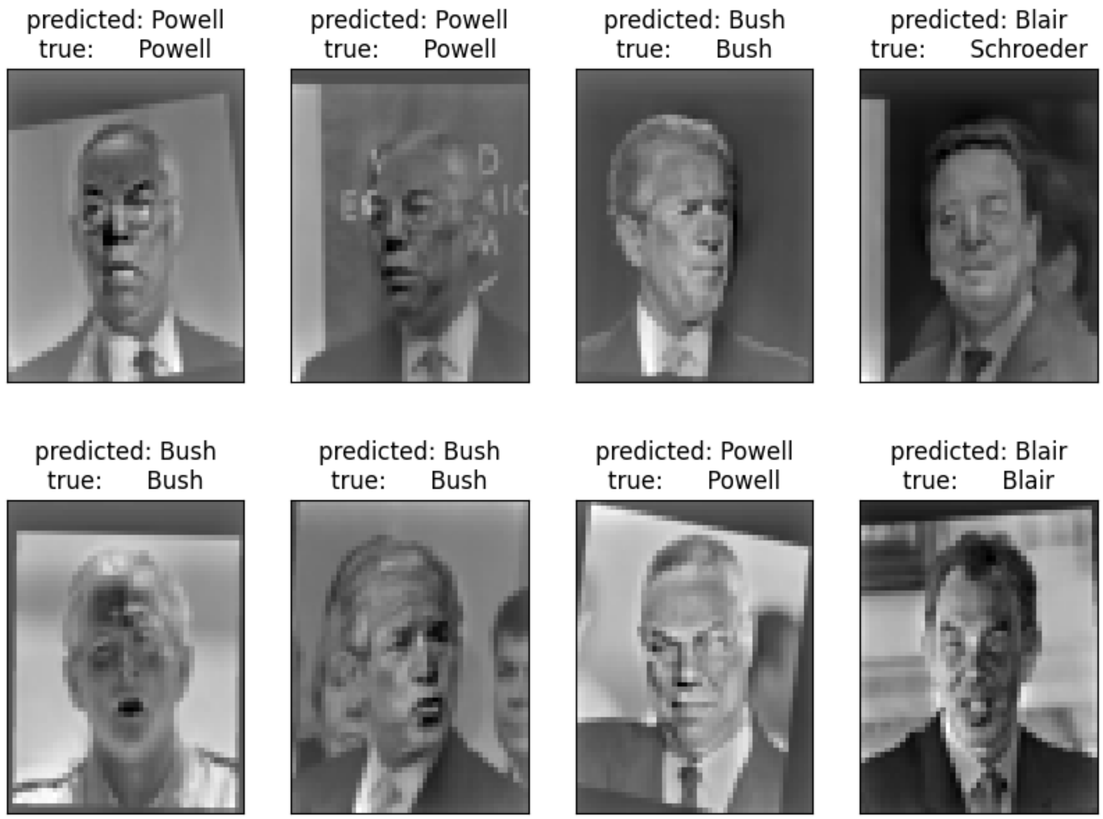
```
Epoch: 0, Loss: 1.613
Epoch: 50, Loss: 0.273
Epoch: 100, Loss: 0.172
Epoch: 150, Loss: 0.131
Epoch: 200, Loss: 0.107
Epoch: 250, Loss: 0.091
Epoch: 300, Loss: 0.079
Epoch: 350, Loss: 0.071
Epoch: 400, Loss: 0.064
Epoch: 450, Loss: 0.059
Epoch: 500, Loss: 0.055
```

[13]:
```
# !!DO NOT EDIT!!
# predict on test data
predictions = model(X_test_pca_torch) # gives softmax logits
y_pred = torch.argmax(predictions, dim=1).numpy() # get the labels from
 ↪prdictions: nx5 -> nx1
```

[14]:
```
# !!DO NOT EDIT!!
# here, we will print the multi-label classification report: precision, recall,
 ↪f1-score etc.
from sklearn.metrics import classification_report
target_names=[y for x,y in targets]
```

```python
print(classification_report(y_test, y_pred, target_names=target_names))

# let us validate some of the predictions by plotting images
# display some of the results
def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(" ", 1)[-1]
    true_name = target_names[y_test[i]].rsplit(" ", 1)[-1]
    return "predicted: %s\ntrue:      %s" % (pred_name, true_name)

prediction_titles = [
    title(y_pred, y_test, target_names, i) for i in range(y_pred.shape[0])
]

plot_gallery(X_test, prediction_titles, height, width)
```

```
                   precision    recall  f1-score   support

     Colin Powell       0.91      0.86      0.88        83
  Donald Rumsfeld       0.74      0.76      0.75        42
    George W Bush       0.90      0.92      0.91       178
Gerhard Schroeder       0.89      0.69      0.77        35
       Tony Blair       0.70      0.85      0.77        39

         accuracy                           0.86       377
        macro avg       0.83      0.81      0.82       377
     weighted avg       0.86      0.86      0.86       377
```


predicted: Powell
true: Powell
predicted: Powell
true: Powell
predicted: Bush
true: Bush
predicted: Blair
true: Schroeder
predicted: Bush
true: Bush
predicted: Bush
true: Bush
predicted: Powell
true: Powell
predicted: Blair
true: Blair