Reyhaneh Derafshi 610396098 HW3

Face recognition using PCA and classification using SVM and neural network. I used the tutorials linked in the project description

In [1]:
```
from sklearn.datasets import fetch_lfw_people
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from time import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

This function displays some of original images for better visual sense

In [2]:
```
def show_orignal_images(pixels):
    fig, axes = plt.subplots(6, 10, figsize=(11, 7),
                             subplot_kw={'xticks': [], 'yticks': []})
    for i, ax in enumerate(axes.flat):
        ax.imshow(np.array(pixels)[i].reshape(50,37), cmap='gray')
    plt.show()
```

function to visualize features(sigenfaces). explained later.

In [10]:
```
def show_eigenfaces(pca):
    fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                             subplot_kw={'xticks': [], 'yticks': []})
    for i, ax in enumerate(axes.flat):
        ax.imshow(pca.components_[i].reshape(50, 37), cmap='gray')
        ax.set_title("PC " + str(i ))
    plt.show()
```

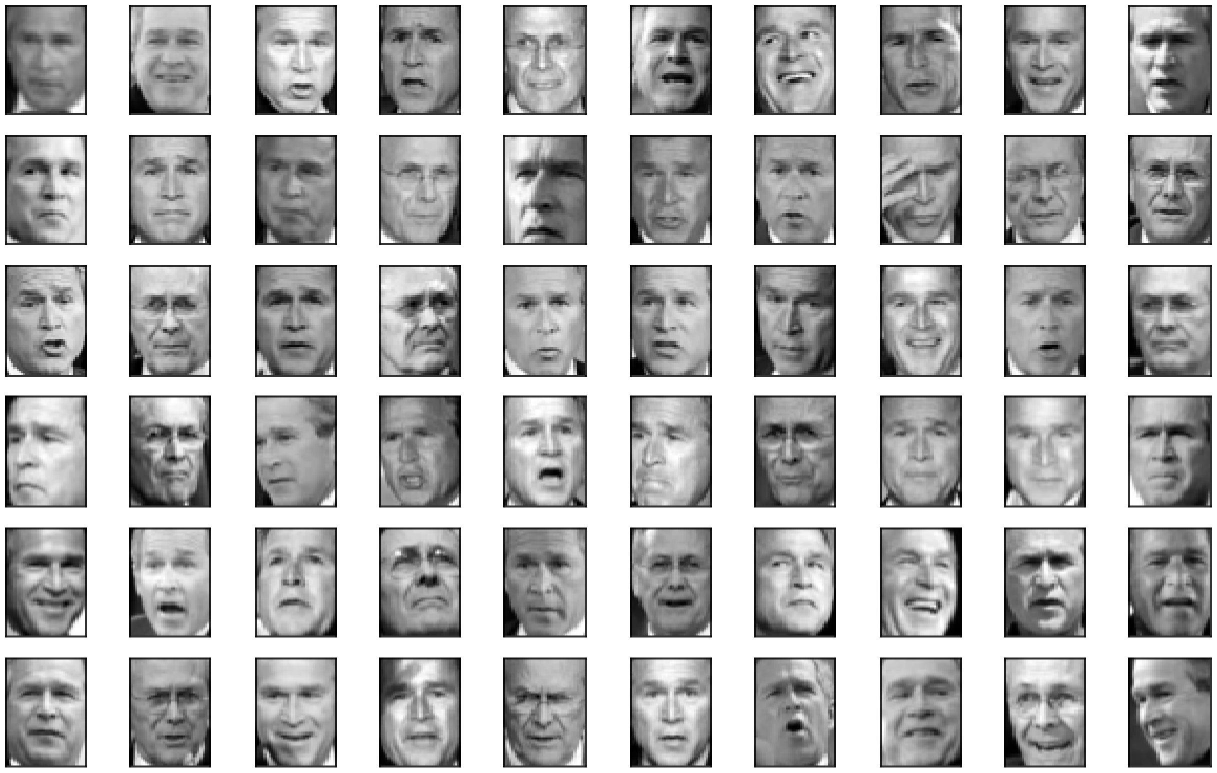
downloading data and seperating images and classes as explained in project description. as you can see, first class has 121 images and second class has 530 images (which shows that our data is unbalanced).

In [32]:
```
lfw_people = fetch_lfw_people(min_faces_per_person= 120, resize=0.4)
x = lfw_people.data
y = lfw_people.target
print("number of images of class 0 :",list(y).count(0))
print("number of images of class 1 :",list(y).count(1))
```

```
number of images of class 0 : 121
number of images of class 1 : 530
```

some of the images has been shown using show_oringinal_images function. As you can see pictures of two persons has displayed randomly.

In [4]:
```
show_orignal_images(x)
```

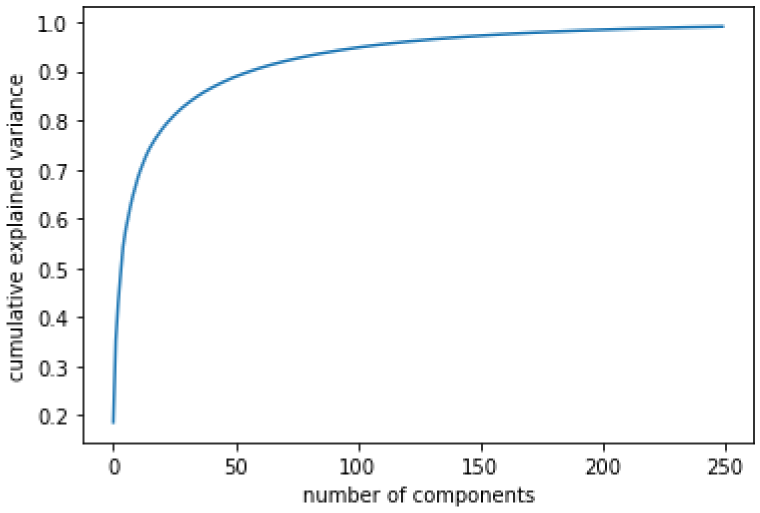


seperating data into train set and test set using train_test_split function.

```
In [5]:
x_train, x_test, y_train, y_test = train_test_split(x, y)
```

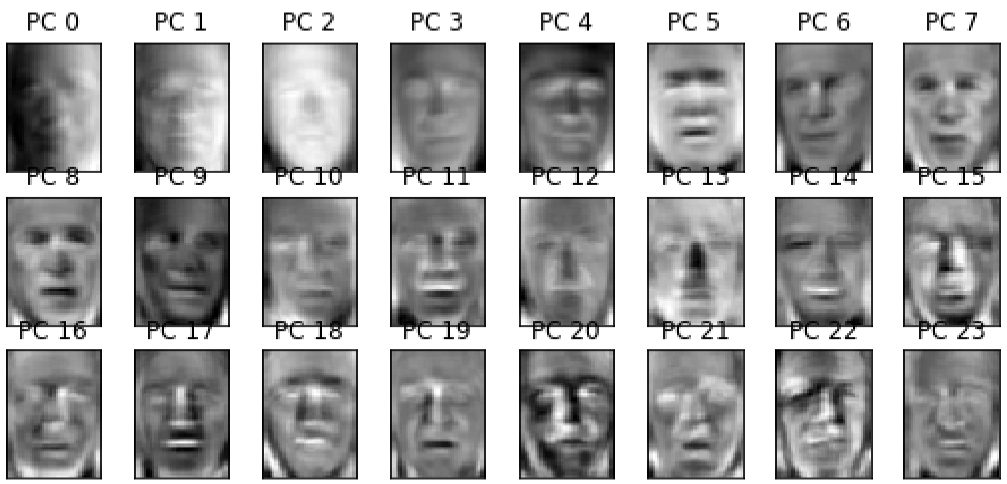
performing PCA and train it with our traning set. I chose 250 componants, then drew a plot to underestand as we increase the number of components how much variation are captured and know how many componants would be enough. As you can see in the plot, more than 170 componants don't make much of a difference. so I chose 170 components.

```
In [6]:
pca = PCA(n_components=170).fit(x_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



visualizing eigenfaces. they give us a sense of along which directions we have the maximum of variation in data. on the white points there is maximum variation and on the black points there is minimum variation.

```
In [11]: show_eigenfaces(pca)
```



projecting train data to pca. then make our svm classifier using SVC. C is Regularization parameter,and defines how muuch errors it should allow, 100 worked the best for me. gamma is Kernel coefficient.if gamma='scale'(default) is passed then it uses 1 / (n_features * X.var()) as value of gamma.

```
In [33]: x_train_pca = pca.transform(x_train)
         clf = SVC(C=100,gamma='scale')
```

1. I calculated the time used for training our model and classifying our data for comparison.
2. then fit classifier with pca form of trained data.
3. then project pca to test data and classify it with predict function.
4. I printed the time it used and classification report.

we had 0.93 accuracy and used 0.114 s

```
In [34]: t0 = time()
         clf = clf.fit(x_train_pca, y_train)
         x_test_pca = pca.transform(x_test)
         y_pred = clf.predict(x_test_pca)
         print("done in %0.3fs" % (time() - t0))
         print(classification_report(y_test, y_pred))
```

```
done in 0.114s
              precision    recall  f1-score   support

           0       0.86      0.67      0.75        27
           1       0.94      0.98      0.96       136

    accuracy                           0.93       163
   macro avg       0.90      0.82      0.85       163
```

```
weighted avg       0.92      0.93      0.92       163
```

I created a neural netwok classifier using sklearn.neural_networks_NLPClassifier.

parameters according to scikit-learn.org:

1. solver{‘lbfgs’, ‘sgd’, ‘adam’}, default=’adam’ The solver for weight optimization.

‘lbfgs’ is an optimizer in the family of quasi-Newton methods. ‘sgd’ refers to stochastic gradient descent. ‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, ‘lbfgs’ can converge faster and perform better.

1. alphafloat, default=0.0001 L2 penalty (regularization term) parameter.

2. hidden_layer_sizestuple, length = n_layers - 2, default=(100,) The ith element represents the number of neurons in the ith hidden layer.

3. random_stateint, RandomState instance, default=None Determines random number generation for weights and bias initialization, train-test split if early stopping is used, and batch sampling when solver=’sgd’ or ‘adam’. Pass an int for reproducible results across multiple function calls.

In [35]:
```
clf = MLPClassifier(solver='lbfgs',alpha=0.001,
                    hidden_layer_sizes=(1200, 200),random_state=1)
```

1. I calculated the time usage for training the model and testing
2. then fit classifier with pca form of trained data.
3. then project pca to test data and classify it with predict function.
4. I printed the time it used and classification report.

we had 0.90 accuracy 24.89s used

In [36]:
```
t0 = time()
clf.fit(x_train_pca, y_train)
y_pred = clf.predict(x_test_pca)
print("done in %0.3fs" % (time() - t0))
print(classification_report(y_test, y_pred))
```

```
done in 24.894s
              precision    recall  f1-score   support

           0       0.70      0.70      0.70        27
           1       0.94      0.94      0.94       136

    accuracy                           0.90       163
   macro avg       0.82      0.82      0.82       163
weighted avg       0.90      0.90      0.90       163
```

conclusion:

neural network classifier used much more time and gave us less accuracy campared to support vector machine.