In [1]:

```
from sklearn.svm import SVC
from sklearn.datasets import fetch_lfw_people
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

1. SVM

In [2]:

```
# ##############################################################################
# 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

# 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]

# the label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.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)
```

```
Total dataset size:
n_samples: 1288
n_features: 1850
n_classes: 7
```

In [3]:

```
# split into a training and testing set
#X_train = X[0:int(n_samples*0.8), :]
#y_train = y[0:int(n_samples*0.8)]
#X_test  = X[int(n_samples*0.8):, :]
#y_test  = y[int(n_samples*0.8):]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
2)
```

In [4]:

```
svm = SVC(kernel='rbf',C=1e3, gamma=0.001, class_weight='balanced')
```

In [5]:

```
svm.fit(X_train, y_train)
```

Out[5]:

```
SVC(C=1000.0, break_ties=False, cache_size=200, class_weight='balanced',
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.001,
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [6]:

```
y_pred = svm.predict(X_test)
```

In [7]:

```
svm.score(X_test, y_test)
```

Out[7]:

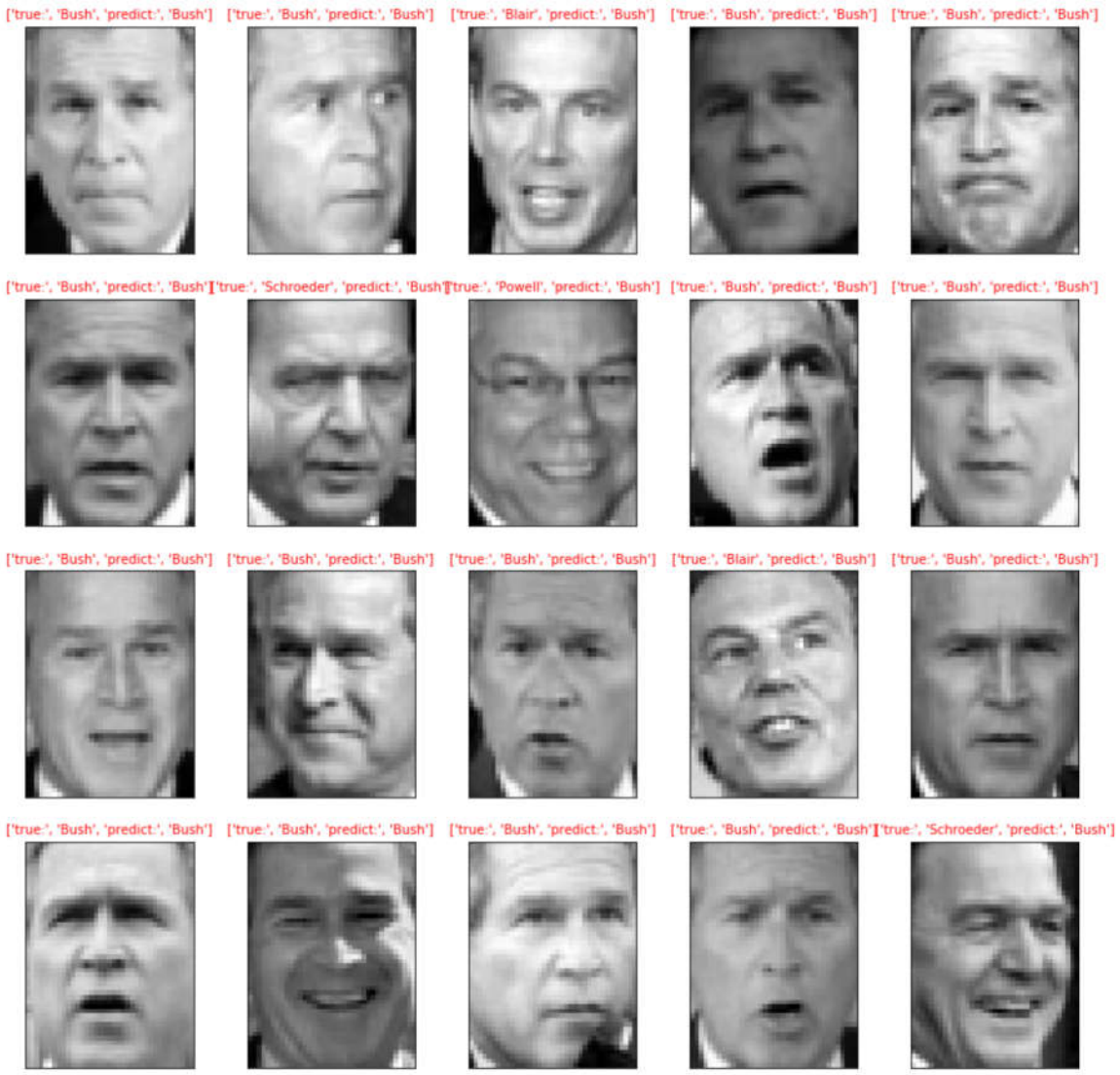
```
0.46124031007751937
```

In [8]:

```
pred_name = target_names[y_pred[0]].rsplit(' ', 1)[-1]
pred_name = target_names[y_test[0]].rsplit(' ', 1)[-1]
```

In [9]:

```
plt.figure(figsize=(15,15))
for i in range(20):
    plt.subplot(4, 5, i + 1)
    plt.imshow(X_test[i].reshape((h, w)), cmap=plt.cm.gray)
    plt.title(["true:", target_names[y_test[i]].rsplit(' ', 1)[-1], "predict:", target_
names[y_pred[i]].rsplit(' ', 1)[-1] ], size=10, c='r')
    plt.xticks(())
    plt.yticks(())
```



2. fix C, change gamma to see the score

In [10]:

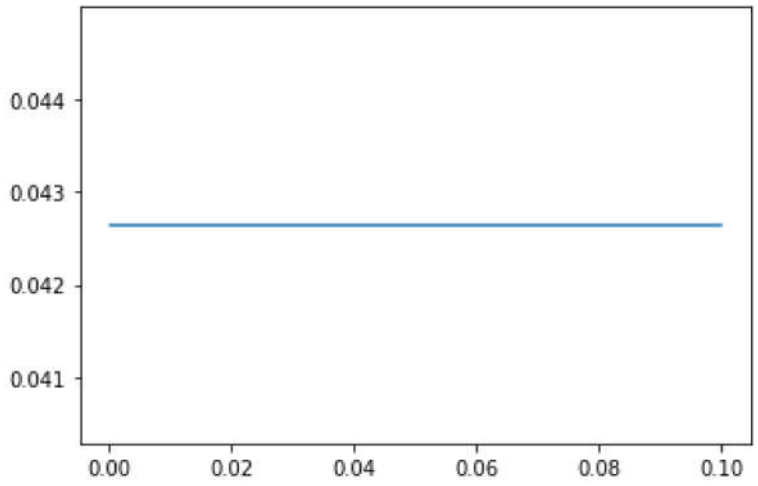
```
gammas = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1]
c = 0.01
scores = []
for gam in gammas:
    svm = SVC(kernel='rbf',C=c, gamma=gam, class_weight='balanced')
    svm.fit(X_train, y_train)
    score = svm.score(X_test, y_test)
    scores.append(score)
```

In [11]:

```
plt.plot(gammas, scores)
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x1dcdf73dbb0>]
```



3. fix gamma, change C to see the score

In [12]:

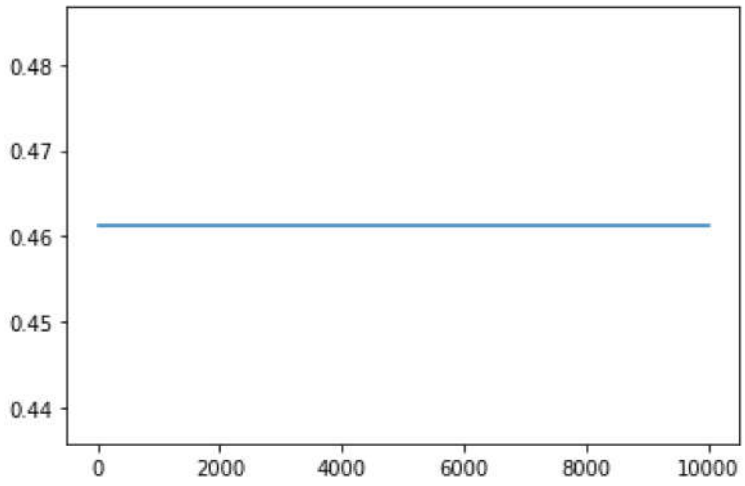
```
C = [1, 1e1, 1e2, 1e3, 1e4]
gam = 0.01
scores = []
for c in C:
    svm = SVC(kernel='rbf',C=c, gamma=gam, class_weight='balanced')
    svm.fit(X_train, y_train)
    score = svm.score(X_test, y_test)
    scores.append(score)
```

In [13]:

```
plt.plot(C, scores)
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x1dcdf120190>]
```



4. Compare KMeans and SVM

SVM and k-means are totally different approaches. SVM is supervised classification, whereas k-means is unsupervised clustering approach. Selon the problem (whether it is classification or clustering), we can choose different method. To be more precise, K-means is utilized when we don't know the labels of our training samples, whereas SVMs are used when we know the class that each training sample belongs to.