Appendix B: Python Demonstration on Using PCA and SVM in Face Recognition

```
print __doc__

from time import time
import logging
import pylab as pl

from sklearn.cross_validation import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import RandomizedPCA
from sklearn.svm import SVC

# Display progress logs on stdout
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')


###############################################################################
# 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

# fot 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


###############################################################################
# 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)
```

```
###############################################################################
# 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])
t0 = time()
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
print "done in %0.3fs" % (time() - t0)

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

print "Projecting the input data on the eigenfaces orthonormal basis"
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print "done in %0.3fs" % (time() - t0)


###############################################################################
# Train a SVM classification model

print "Fitting the classifier to the training set"
t0 = time()
param_grid = {
 'C': [1e3, 5e3, 1e4, 5e4, 1e5],
 'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
}
clf = GridSearchCV(SVC(kernel='rbf', class_weight='auto'), param_grid)
clf = clf.fit(X_train_pca, y_train)
print "done in %0.3fs" % (time() - t0)
print "Best estimator found by grid search:"
print clf.best_estimator_


###############################################################################
# Quantitative evaluation of the model quality on the test set

print "Predicting the people names on the testing set"
t0 = time()
y_pred = clf.predict(X_test_pca)
print "done in %0.3fs" % (time() - t0)

print classification_report(y_test, y_pred, target_names=target_names)
print confusion_matrix(y_test, y_pred, labels=range(n_classes))


###############################################################################
# Qualitative evaluation of the predictions using matplotlib
```

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


# plot the result of the prediction on a portion of the test set

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, 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)

pl.show()
```

```
Built-in functions, exceptions, and other objects.

Noteworthy: None is the `nil' object; Ellipsis represents `...' in slices.
Total dataset size:

n_samples: 1288
n_features: 1850
n_classes: 7
Extracting the top 150 eigenfaces from 966 faces
done in 0.489s

Projecting the input data on the eigenfaces orthonormal basis
done in 0.054s

Fitting the classifier to the training set
done in 17.447s

Best estimator found by grid search:
SVC(C=1000.0, cache_size=200, class_weight=auto, coef0=0.0, degree=3,
  gamma=0.005, kernel=rbf, probability=False, shrinking=True, tol=0.001,
  verbose=False)
Predicting the people names on the testing set
```

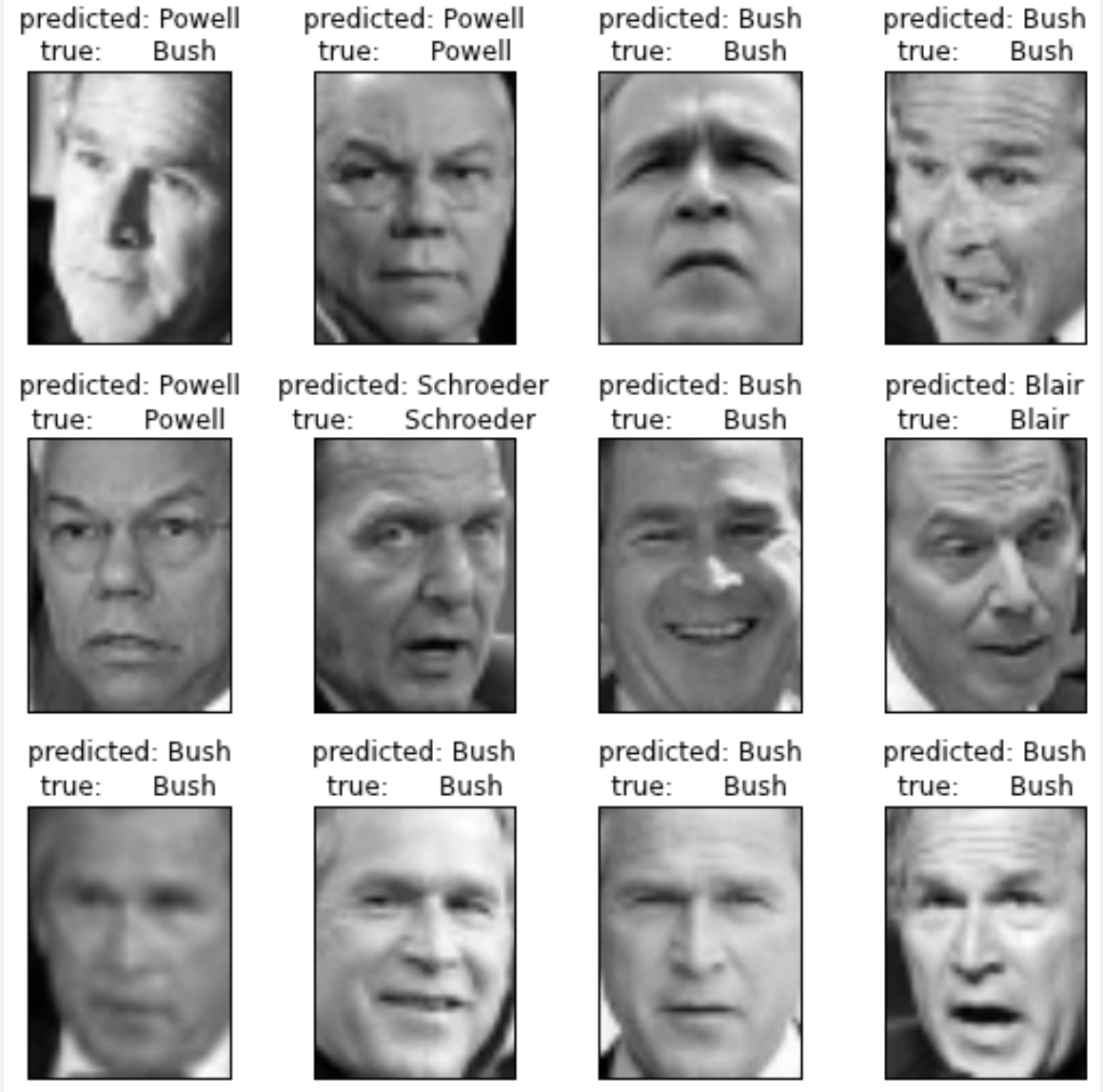
```
done in 0.054s
```

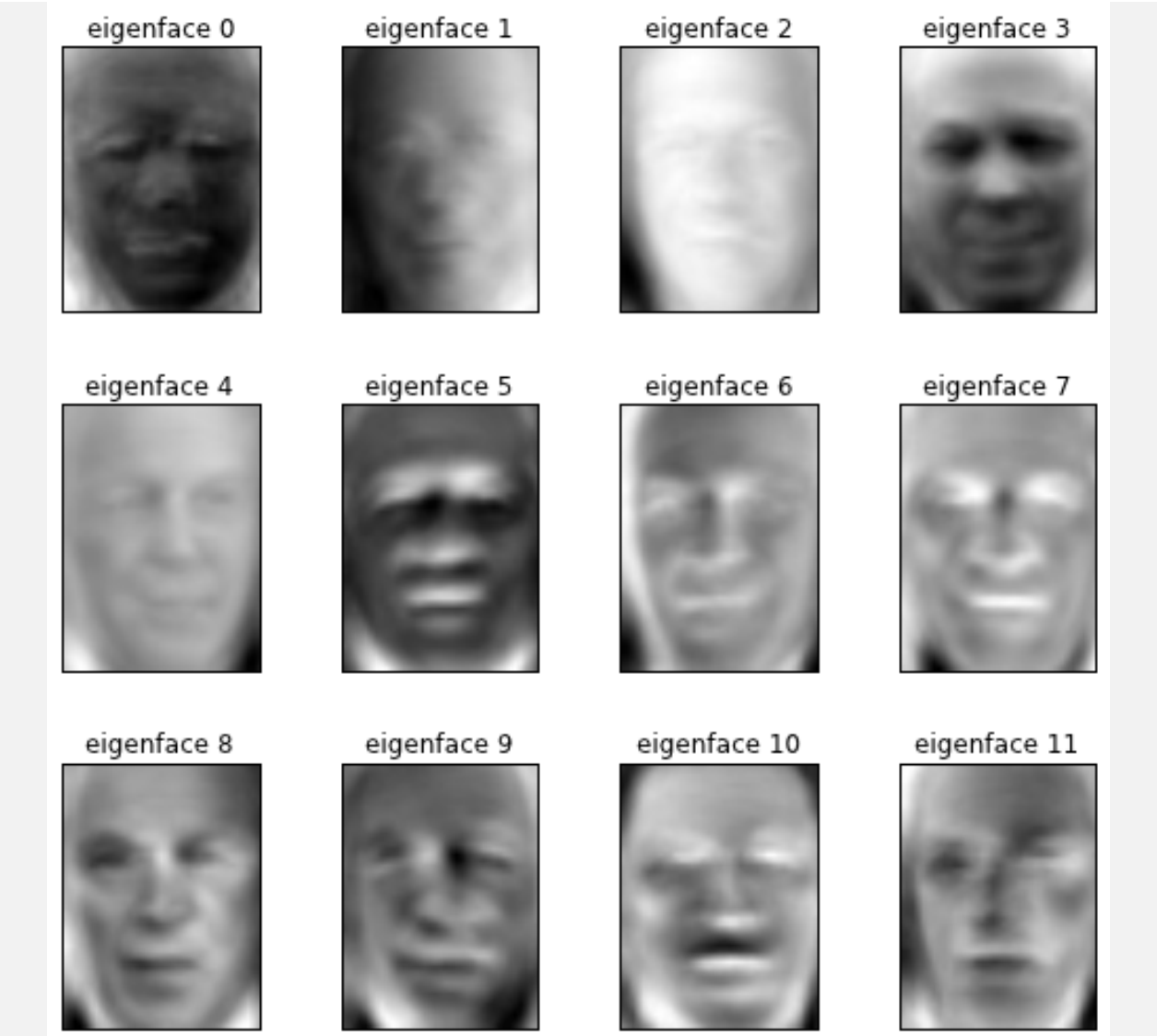
```
precision    recall  f1-score   support

     Ariel Sharon       1.00      0.61      0.76        18
     Colin Powell       0.82      0.92      0.87        64
  Donald Rumsfeld       0.95      0.70      0.81        27
    George W Bush       0.80      0.95      0.87       124
Gerhard Schroeder       0.96      0.75      0.84        36
      Hugo Chavez       1.00      0.65      0.79        23
       Tony Blair       0.90      0.87      0.88        30

      avg / total       0.87      0.85      0.85       322

[[ 11   4   1   2   0   0   0]
 [  0  59   0   5   0   0   0]
 [  0   1  19   7   0   0   0]
 [  0   4   0 118   1   0   1]
 [  0   0   0   7  27   0   2]
 [  0   4   0   4   0  15   0]
 [  0   0   0   4   0   0  26]]
```


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


eigenface 0
eigenface 1
eigenface 2
eigenface 3
eigenface 4
eigenface 5
eigenface 6
eigenface 7
eigenface 8
eigenface 9
eigenface 10
eigenface 11