

# PCA Mini-Project-Copy1

March 5, 2019

## 1 PCA Mini-Project

### 1.0.1 Faces recognition example using eigenfaces and SVMs

Our discussion of PCA spent a lot of time on theoretical issues, so in this mini-project we'll ask you to play around with some sklearn code. The eigenfaces code is interesting and rich enough to serve as the testbed for this entire mini-project.

Note: The dataset used in this example is a preprocessed excerpt of the "[Labeled Faces in the Wild](#)", aka [LFW\\_Download](#) (233MB). [Original source](#).

```
In [1]: from time import time
import logging
import pylab as pl
import numpy as np

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
from sklearn.decomposition import RandomizedPCA
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

### 1.1 Loading the dataset

```
In [2]: # Download the data, if not already on disk and load it as numpy arrays
lfw_people = fetch_lfw_people('data', 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
np.random.seed(42)

# for machine learning we use the data directly (as relative pixel
# position 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

```

### 1.1.1 Split into a training and testing set

```
In [3]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=4)
```

## 1.2 Compute PCA

We can now compute a [PCA](#) (eigenfaces) on the face dataset (treated as unlabeled dataset): unsupervised feature extraction / dimensionality reduction.

```
In [4]: n_components = 150
```

```

print("Extracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0])
t0 = time()

# TODO: Create an instance of PCA, initializing with n_components=n_components and whiten=True
pca = PCA(n_components=n_components, whiten=True, svd_solver='randomized')

#TODO: pass the training dataset (X_train) to pca's 'fit()' method
pca = pca.fit(X_train)

print("done in %0.3fs" % (time() - t0))

```

```

Extracting the top 150 eigenfaces from 966 faces
done in 0.374s

```

Projecting the input data on the eigenfaces orthonormal basis

```
In [5]: eigenfaces = pca.components_.reshape((n_components, h, w))
```

```

t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))

```

done in 0.038s

### 1.3 Train a SVM classification model

Let's fit a [SVM classifier](#) to the training set. We'll use [GridSearchCV](#) to find a good set of parameters for the classifier.

```

In [6]: param_grid = {
        'C': [1e3, 5e3, 1e4, 5e4, 1e5],
        'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
        }

# for sklearn version 0.16 or prior, the class_weight parameter value is 'auto'
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)

print("Best estimator found by grid search:")
print(clf.best_estimator_)

```

Best estimator found by grid search:

```

SVC(C=1000.0, 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)

```

### 1.4 Evaluation of the model quality on the test set

**1. Classification Report** Now that we have the classifier trained, let's run it on the test dataset and qualitatively evaluate its results. Sklearn's [classification\\_report](#) shows some of the main classification metrics for each class.

```

In [7]: y_pred = clf.predict(X_test_pca)

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

```

	precision	recall	f1-score	support
Ariel Sharon	0.56	0.69	0.62	13
Colin Powell	0.74	0.87	0.80	60
Donald Rumsfeld	0.76	0.81	0.79	27
George W Bush	0.93	0.87	0.90	146
Gerhard Schroeder	0.76	0.76	0.76	25

Hugo Chavez	0.73	0.53	0.62	15
Tony Blair	0.88	0.83	0.86	36
avg / total	0.84	0.83	0.83	322

**2. Confusion Matrix** Another way to look at the performance of the classifier is by looking the [confusion matrix](#). We can do that by simply invoking [sklearn.metrics.confusion\\_matrix](#):

```
In [8]: print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))
```

```
[[ 9  0  3  1  0  0  0]
 [ 2 52  1  4  0  1  0]
 [ 4  0 22  1  0  0  0]
 [ 1 11  2 127  3  1  1]
 [ 0  2  0  1 19  1  2]
 [ 0  3  0  1  2  8  1]
 [ 0  2  1  2  1  0 30]]
```

### 3. Plotting The Most Significant Eigenfaces

```
In [9]: 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(())

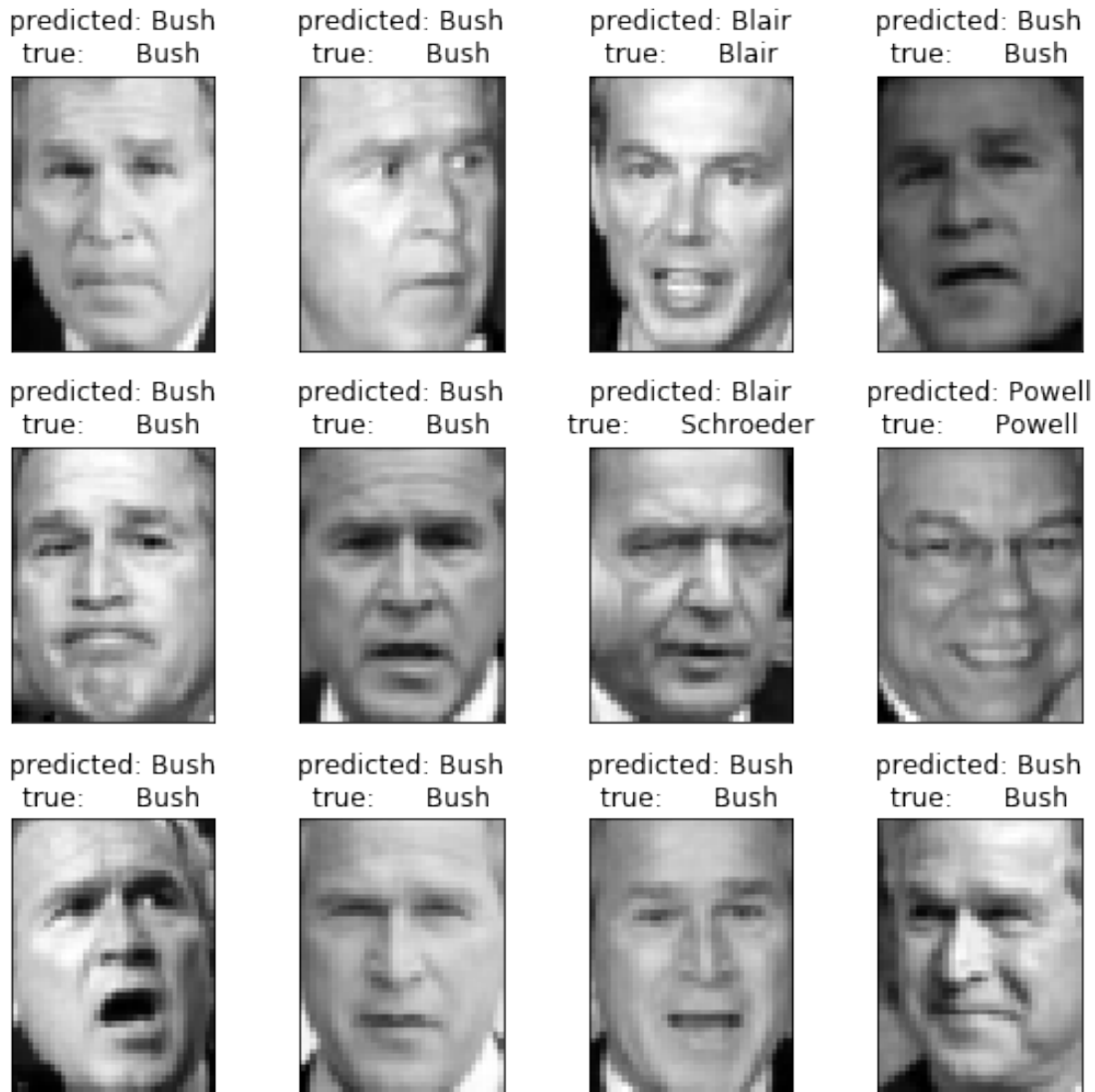
        # plot the result of the prediction on a portion of the test set

        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])]

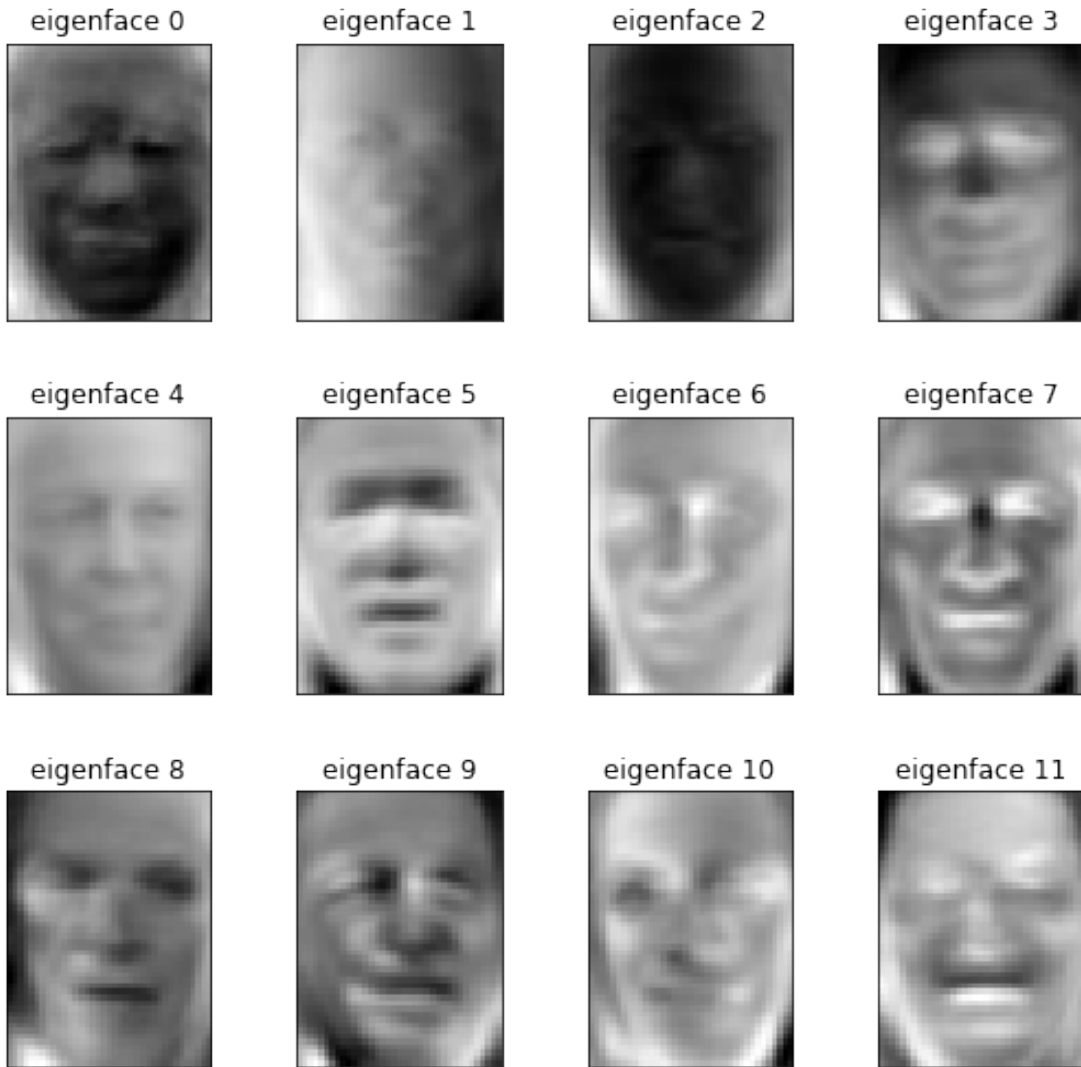
        plot_gallery(X_test, prediction_titles, h, w)

        pl.show()
```



```
In [10]: eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
         plot_gallery(eigenfaces, eigenface_titles, h, w)

         plt.show()
```



## 1.5 Quiz: Explained Variance Of Each PC

We mentioned that PCA will order the principal components, with the first PC giving the direction of maximal variance, second PC has second-largest variance, and so on. How much of the variance is explained by the first principal component? The second?

In [ ]:

## 1.6 Quiz: How Many PCs To Use?

Now you'll experiment with keeping different numbers of principal components. In a multiclass classification problem like this one (more than 2 labels to apply), accuracy is a less-intuitive metric than in the 2-class case. Instead, a popular metric is the F1 score.

We'll learn about the F1 score properly in the lesson on evaluation metrics, but you'll figure out for yourself whether a good classifier is characterized by a high or low F1 score. You'll do this by varying the number of principal components and watching how the F1 score changes in response.

As you add more principal components as features for training your classifier, do you expect it to get better or worse performance?

In [ ]:

### 1.7 Quiz: F1 Score Vs. No. Of PCs Used

Change `n_components` to the following values: [10, 15, 25, 50, 100, 250]. For each number of principal components, note the F1 score for Ariel Sharon. (For 10 PCs, the plotting functions in the code will break, but you should be able to see the F1 scores.) If you see a higher F1 score, does it mean the classifier is doing better, or worse?

In [ ]:

### 1.8 Quiz: Dimensionality Reduction And Overfitting

Do you see any evidence of overfitting when using a large number of PCs? Does the dimensionality reduction of PCA seem to be helping your performance here?

In [ ]: