PCA Mini-Project

July 3, 2018

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 [48]: n_components = 500

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

         # TODO: Create an instance of PCA, initializing with n_components=n_components and whit
         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 500 eigenfaces from 966 faces
done in 0.779s
```

Projecting the input data on the eigenfaces orthonormal basis

```
In [49]: 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 0.054s
```

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 [51]: 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=0.0001, 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 [52]: 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       0.55      0.92      0.69        13
     Colin Powell       0.65      0.85      0.73        60
  Donald Rumsfeld       0.65      0.63      0.64        27
    George W Bush       0.87      0.76      0.81       146
Gerhard Schroeder       0.52      0.44      0.48        25
```

```
 Hugo Chavez       0.67      0.53      0.59        15
  Tony Blair       0.66      0.64      0.65        36

 avg / total       0.74      0.72      0.72       322
```

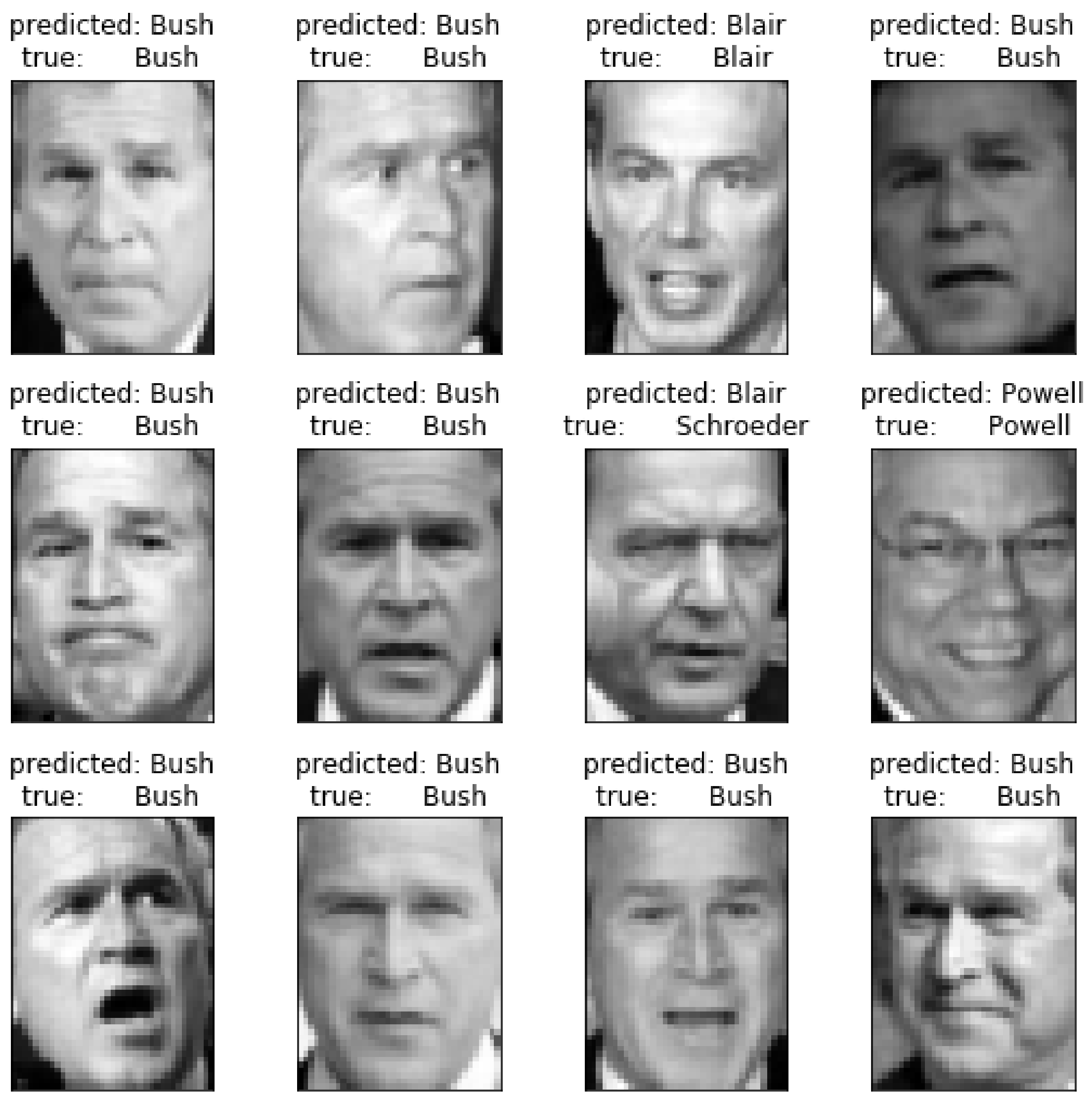
2. Confusion Matrix Another way to look at the performance of the classifier is by looking the confusion matrix. We can do that by simply invoking sklearn.metrics.confusion_matrix:

```
In [29]: print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))
```

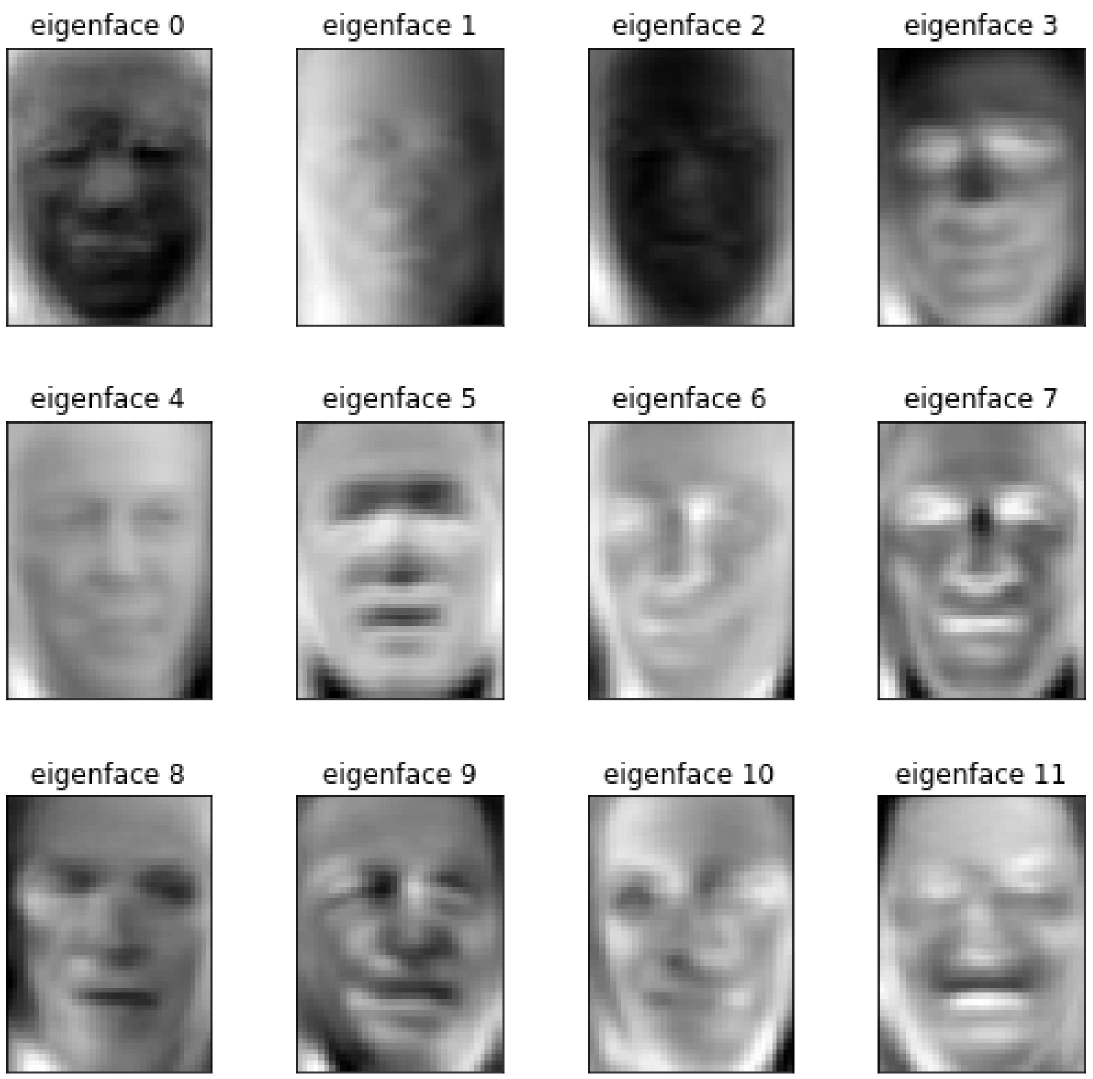
```
[[  9   2   0   2   0   0   0]
 [  1  54   2   2   0   1   0]
 [  2   2  18   4   0   0   1]
 [  2   7   0 132   2   1   2]
 [  0   1   0   4  19   0   1]
 [  0   4   0   2   1   7   1]
 [  1   3   0   4   1   0  27]]
```

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)

         pl.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 [11]: print(pca.explained_variance_ratio_)
```

```
[ 0.19346534  0.1511682   0.07083675  0.05951796  0.05157499  0.02887153
  0.02514483  0.02176464  0.02019381  0.01902125  0.01682212  0.015806
  0.01223362  0.01087938  0.01064453  0.00979653  0.00892399  0.00854845
  0.00835712  0.00722636  0.00696569  0.00653856  0.00639558  0.00561316
  0.00531107  0.00520152  0.00507466  0.00484209  0.00443588  0.0041783
  0.00393704  0.00381729  0.00356061  0.00351201  0.00334556  0.00329931
  0.00314626  0.00296217  0.00290136  0.00284723  0.00280004  0.00267555
  0.00259901  0.00258401  0.00240919  0.00238994  0.00235403  0.00222587
  0.00217507  0.00216566  0.00209064  0.00205426  0.00200421  0.00197395
  0.0019383   0.00188764  0.00180173  0.00178897  0.00174819  0.00173054
  0.00165647  0.00162948  0.00157409  0.00153429  0.00149965  0.00147262
  0.00143931  0.00141882  0.00139698  0.00138147  0.00134005  0.00133169
  0.00128812  0.00125595  0.00124247  0.00121864  0.00120948  0.00118305
  0.00115092  0.00113676  0.00112622  0.00111621  0.00109399  0.00107175
  0.00105669  0.0010436   0.00102396  0.00101696  0.00099776  0.00096374
  0.00094236  0.00091984  0.00091324  0.00089209  0.00087176  0.00086249
  0.00084313  0.00083927  0.00082853  0.00080313  0.00078747  0.00078228
  0.00075691  0.00075292  0.00074768  0.0007347   0.00073158  0.00071689
  0.00070577  0.00069715  0.00066866  0.00066503  0.00065623  0.00063944
  0.00063688  0.00062721  0.00061795  0.00061033  0.00060175  0.00059333
  0.00058297  0.00057581  0.00056649  0.00056236  0.00055077  0.00054499
  0.00053279  0.00052337  0.00051409  0.00051363  0.00051042  0.00049526
  0.00048876  0.00047817  0.00047104  0.00046536  0.00046507  0.00045444
  0.00044802  0.00044516  0.00043847  0.00043143  0.00042947  0.00042014
  0.00041234  0.0004096   0.0004038   0.00040279  0.00039076  0.00038591]
```

```
In [12]: """These are the ratios showing how much variance can be attributed to each of the resp

Out[12]: 'These are the ratios showing how much variance can be attributed to each of the respec
```

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 [ ]: # 5 compenents - F1 Score: .35
        # 150 compenents - F1 Score: .83
        # 300 compenents - F1 Score: .82
```

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?

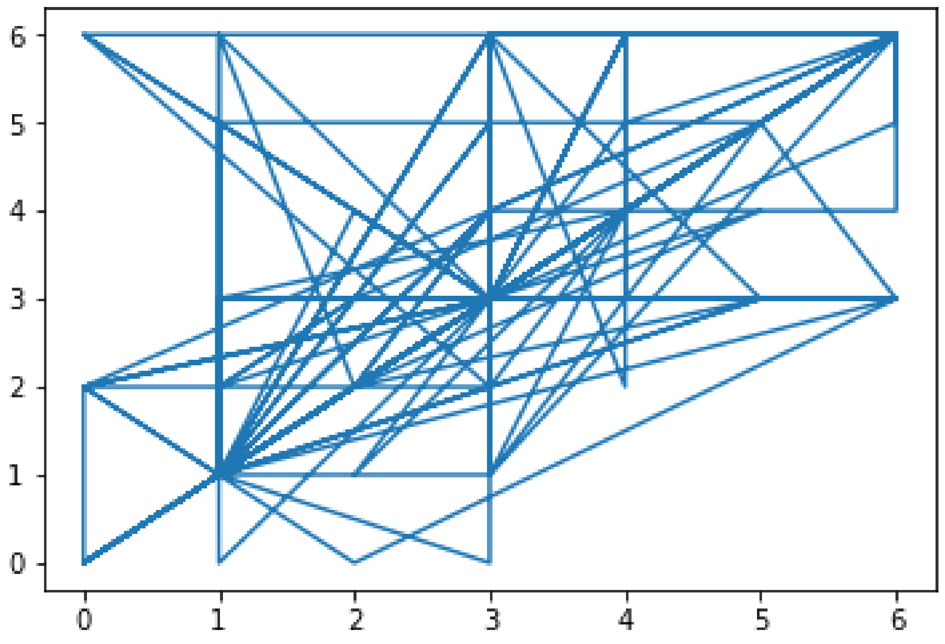
```
In [ ]: # 10 F-Score .12
        # 25
        # 50 F-Score .69
        #250 F-Score .64
```

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 [40]: pl.plot(y_pred,y_test)
```

```
Out[40]: [<matplotlib.lines.Line2D at 0x7fec4e53c7b8>]
```



```
In [41]: print(y_pred)
```

```
[3 3 6 3 3 3 4 1 3 3 3 3 3 4 3 3 6 1 3 4 1 0 3 0 0 1 0 3 3 3 2 3 3 3 2 3 3
 1 3 1 3 1 3 1 1 1 4 3 3 3 3 3 0 3 6 2 1 3 5 3 1 1 1 4 3 4 6 4 3 3 6 6 3 2
 3 2 1 6 6 4 3 0 4 3 3 3 3 3 3 3 3 6 3 2 1 3 1 1 6 6 3 3 3 1 3 1 3 3 3 1 3
 1 6 4 3 1 3 4 1 3 1 3 3 0 3 4 4 3 1 3 6 6 6 3 4 4 3 3 1 1 2 2 5 1 3 5 1 3
 3 1 1 1 1 3 3 3 6 0 1 3 6 5 5 1 3 1 5 1 3 3 1 1 6 1 5 6 3 2 2 3 3 3 2 1 2
 3 3 3 3 2 3 2 3 2 6 3 3 6 3 3 5 2 1 2 3 3 6 2 1 2 6 5 3 3 3 3 3 0 0 1 3 3
 1 1 6 3 3 3 1 3 3 3 1 0 3 1 6 3 3 3 3 5 2 3 3 0 3 3 4 4 4 3 3 0 3 4 3 1 6
 0 3 3 3 1 3 4 1 1 3 6 1 1 3 3 4 3 6 3 3 3 1 1 3 3 1 1 3 3 3 4 3 3 5 3 3 0
 4 2 3 4 3 0 6 2 1 3 1 5 1 3 3 3 1 3 3 3 1 1 3 2 5 2]
```

```
In [42]: from sklearn.metrics import accuracy_score

In [45]: print("Number of compenents is 50:",accuracy_score(y_pred,y_test))

Number of compenents is 50: 0.813664596273


In [53]: print("Number of compenents is 500:",accuracy_score(y_pred,y_test))

Number of compenents is 500: 0.723602484472


In [54]: #The evidence of overfitting seems muted by use of PCA but still present as the accurac
```