Eigenfaces_PCA_SVM

January 29, 2022

[18]:
```
%matplotlib inline
```

1 The eigenfaces example: chaining PCA and Naive Bayes Classifier (SVMs)

The goal of this example is to show how an unsupervised method and a supervised one can be chained for better prediction.

Here we'll take a look at a simple facial recognition example. Ideally, we would use a dataset consisting of a subset of the `Labeled Faces in the Wild <http://vis-www.cs.umass.edu/lfw/>___` data that is available with :func:`sklearn.datasets.fetch_lfw_people`.

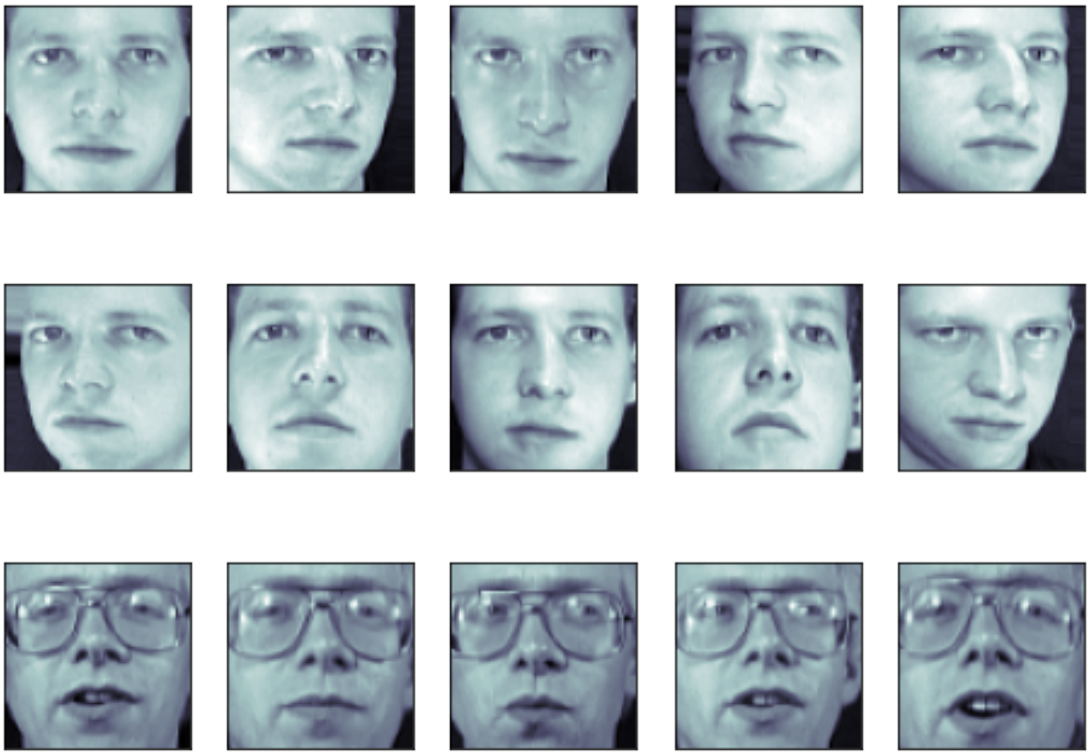
However, this is a relatively large download (~200MB) so we will do the tutorial on a simpler, less rich dataset.

[19]:
```
from sklearn import datasets
faces = datasets.fetch_olivetti_faces()
faces.data.shape
```

[19]:
```
(400, 4096)
```

Let's visualize these faces to see what we're working with

[20]:
```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(8, 6))
# plot several images
for i in range(15):
    ax = fig.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    ax.imshow(faces.images[i], cmap=plt.cm.bone)
```



Note is that these faces have already been localized and scaled to a common size.

This is an important preprocessing piece for facial recognition, and is a process that can require a large collection of training data.

This can be done in scikit-learn, but the challenge is gathering a sufficient amount of training data for the algorithm to work.

We'll perform a Support Vector classification of the images. We'll do a typical train-test split on the images:

[21]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(faces.data,
        faces.target, random_state=0)

print(X_train.shape, X_test.shape)
```

```
(300, 4096) (100, 4096)
```

1.1 Preprocessing: Principal Component Analysis

We can use PCA to reduce these features to a manageable size, while maintaining most of the information in the dataset.

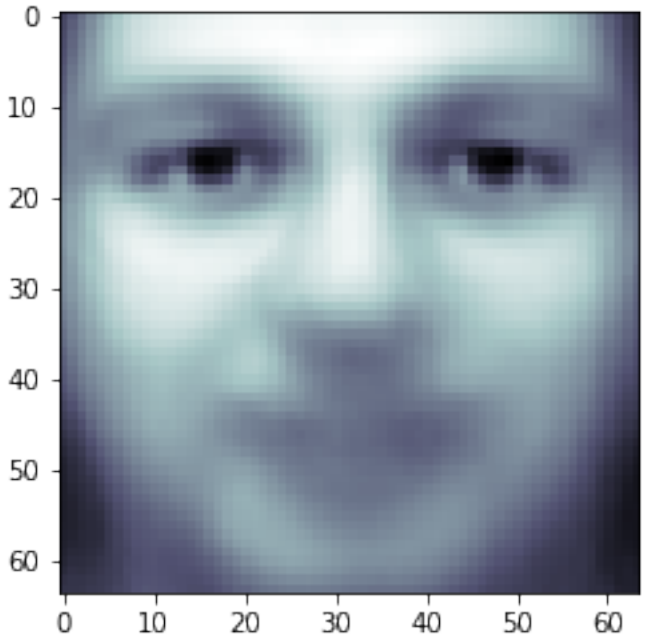
[22]:
```
from sklearn import decomposition
pca = decomposition.PCA(n_components=150, whiten=True)
pca.fit(X_train)
```

[22]:
```
PCA(n_components=150, whiten=True)
```

One interesting part of PCA is that it computes the "mean" face, which can be interesting to examine:

[23]:
```
plt.imshow(pca.mean_.reshape(faces.images[0].shape),
           cmap=plt.cm.bone)
```

[23]:
```
<matplotlib.image.AxesImage at 0x13fde6070>
```



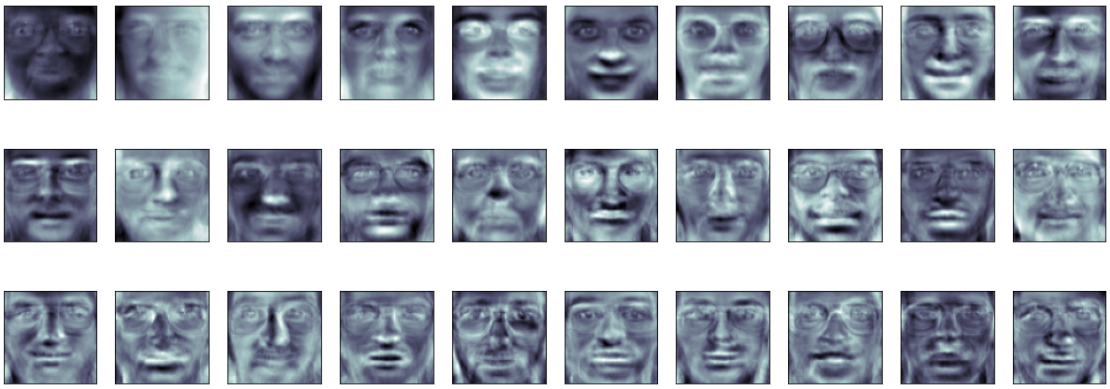
The principal components measure deviations about this mean along orthogonal axes.

[24]:
```
print(pca.components_.shape)
```

```
(150, 4096)
```

It is also interesting to visualize these principal components:

[25]:
```
fig = plt.figure(figsize=(16, 6))
for i in range(30):
    ax = fig.add_subplot(3, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(pca.components_[i].reshape(faces.images[0].shape),
              cmap=plt.cm.bone)
```



The components ("eigenfaces") are ordered by their importance from top-left to bottom-right. We see that the first few components seem to primarily take care of lighting conditions; the remaining components pull out certain identifying features: the nose, eyes, eyebrows, etc.

With this projection computed, we can now project our original training and test data onto the PCA basis:

[26]:
```
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape)
```

```
(300, 150)
```

[27]:
```
print(X_test_pca.shape)
```

```
(100, 150)
```

These projected components correspond to factors in a linear combination of component images such that the combination approaches the original face.

1.2 Doing the Learning: Naive Bayes Classifier (Support Vector Machines)

Now we'll perform support-vector-machine classification on this reduced dataset:

[28]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train_pca, y_train)
```

[28]:
```
GaussianNB()
```

from sklearn import svm clf = svm.SVC(C=5., gamma=0.001) clf.fit(X_train_pca, y_train)

Finally, we can evaluate how well this classification did. First, we might plot a few of the test-cases with the labels learned from the training set:

[29]:
```
import numpy as np
fig = plt.figure(figsize=(8, 6))
```

```
for i in range(15):
    ax = fig.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    ax.imshow(X_test[i].reshape(faces.images[0].shape),
              cmap=plt.cm.bone)
    y_pred = gnb.predict(X_test_pca[i, np.newaxis])[0]
    color = ('black' if y_pred == y_test[i] else 'red')
    ax.set_title(y_pred, fontsize='small', color=color)
```

The classifier is correct on an impressive number of images given the simplicity of its learning model! Using a linear classifier on 150 features derived from the pixel-level data, the algorithm correctly identifies a large number of the people in the images.

Again, we can quantify this effectiveness using one of several measures from :mod:`sklearn.metrics`. First we can do the classification report, which shows the precision, recall and other measures of the "goodness" of the classification:

[30]:
```
from sklearn import metrics
y_pred = gnb.predict(X_test_pca)
print(metrics.classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00         6
```

```
              1       1.00      0.75      0.86         4
              2       0.50      0.50      0.50         2
              3       0.12      1.00      0.22         1
              4       1.00      1.00      1.00         1
              5       1.00      0.80      0.89         5
              6       1.00      1.00      1.00         4
              7       1.00      0.67      0.80         3
              9       0.14      1.00      0.25         1
             10       1.00      1.00      1.00         4
             11       0.50      1.00      0.67         1
             12       0.67      1.00      0.80         2
             13       1.00      1.00      1.00         3
             14       1.00      1.00      1.00         5
             15       0.75      1.00      0.86         3
             16       0.00      0.00      0.00         0
             17       0.00      0.00      0.00         6
             19       0.75      0.75      0.75         4
             20       1.00      1.00      1.00         1
             21       1.00      1.00      1.00         1
             22       1.00      1.00      1.00         2
             23       1.00      1.00      1.00         1
             24       1.00      1.00      1.00         2
             25       1.00      0.50      0.67         2
             26       1.00      0.50      0.67         4
             27       1.00      1.00      1.00         1
             28       1.00      1.00      1.00         2
             29       1.00      1.00      1.00         3
             30       1.00      1.00      1.00         4
             31       1.00      1.00      1.00         3
             32       1.00      0.67      0.80         3
             33       1.00      1.00      1.00         2
             34       0.75      1.00      0.86         3
             35       0.50      1.00      0.67         1
             36       1.00      1.00      1.00         3
             37       1.00      1.00      1.00         3
             38       1.00      1.00      1.00         1
             39       1.00      1.00      1.00         3

    accuracy                           0.79       100
   macro avg       0.81      0.85      0.80       100
weighted avg       0.81      0.79      0.78       100


/Users/brijeshbhatt/MyProjects/env/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/Users/brijeshbhatt/MyProjects/env/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/brijeshbhatt/MyProjects/env/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/brijeshbhatt/MyProjects/env/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/brijeshbhatt/MyProjects/env/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/brijeshbhatt/MyProjects/env/lib/python3.8/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Another interesting metric is the *confusion matrix*, which indicates how often any two items are mixed-up. The confusion matrix of a perfect classifier would only have nonzero entries on the diagonal, with zeros on the off-diagonal:

[31]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[0 0 0 … 0 0 0]
 [0 3 0 … 0 0 0]
 [0 0 1 … 0 0 0]
 …
 [0 0 0 … 3 0 0]
 [0 0 0 … 0 1 0]
 [0 0 0 … 0 0 3]]
```

1.3 Pipelining

Above we used PCA as a pre-processing step before applying our support vector machine classifier. Plugging the output of one estimator directly into the input of a second estimator is a commonly used pattern; for this reason scikit-learn provides a `Pipeline` object which automates this process. The above problem can be re-expressed as a pipeline as follows:

[32]:
```
from sklearn.pipeline import Pipeline
clf = Pipeline([('pca', decomposition.PCA(n_components=150, whiten=True)),
```

```
                ('gnb', GaussianNB())])

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(metrics.confusion_matrix(y_pred, y_test))
plt.show()
```

```
[[1 0 0 … 0 0 0]
 [0 3 0 … 0 0 0]
 [0 0 1 … 0 0 0]
 …
 [0 0 0 … 3 0 0]
 [0 0 0 … 0 1 0]
 [0 0 0 … 0 0 3]]
```

1.4 A Note on Facial Recognition

Here we have used PCA "eigenfaces" as a pre-processing step for facial recognition. The reason we chose this is because PCA is a broadly-applicable technique, which can be useful for a wide array of data types. Research in the field of facial recognition in particular, however, has shown that other more specific feature extraction methods are can be much more effective.

Exercise: 1. Train the Naive Bayes model without PCA and compare the result with PCA + Naive Bayes. Write down your observations. 2. Run PCA on IRIS dataset. Visualise the output in 2 dimensions using to Principal components. Choose different pairs of principal components and note down your observations. 3. Run PCA + Naive Bayes classifier on IRIS dataset and calculate precision and recall of the system.