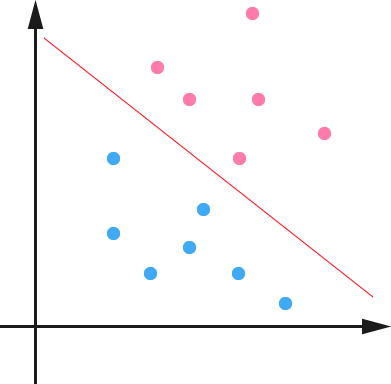
**Python支持向量机算法进行人脸识别**

**相关知识**

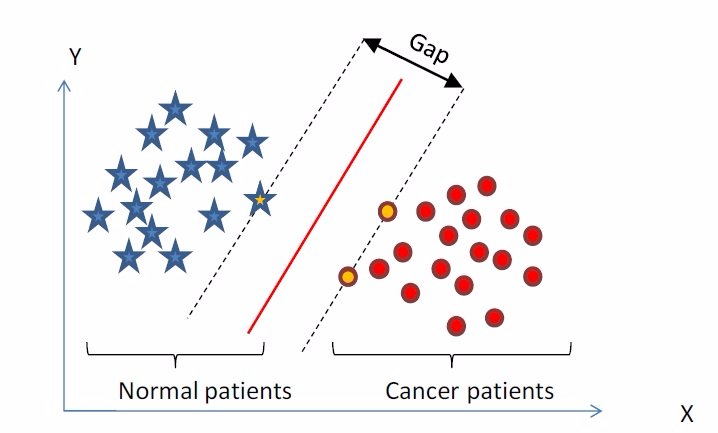
支持向量机(support vector machine)是一种分类算法，通过寻求结构化风险最小来提高学习机泛化能力，实现经验风险和置信范围的最小化，从而达到在统计样本量较少的情况下，亦能获得良好统计规律的目的。通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，即支持向量机的学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

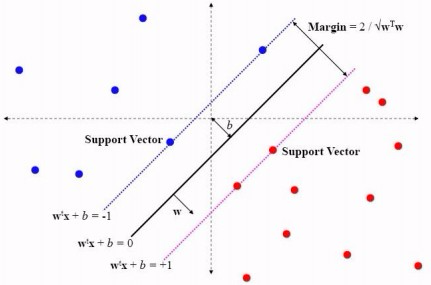
**具体原理：**

1. 在n维空间中找到一个分类超平面，将空间上的点分类。如下图是线性分类的例子。

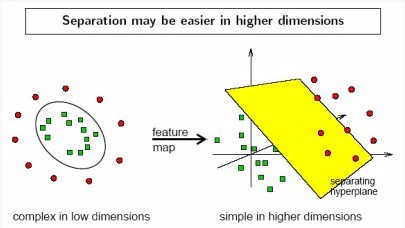
[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/01.png)

2. 一般而言，一个点距离超平面的远近可以表示为分类预测的确信或准确程度。SVM就是要最大化这个间隔值。而在虚线上的点便叫做支持向量Supprot Verctor。

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/02.png)

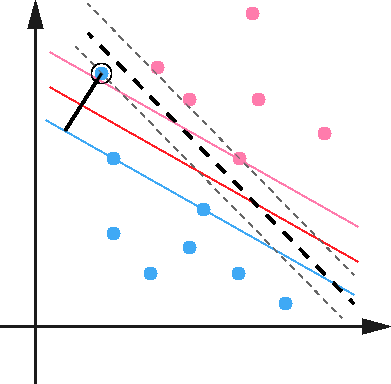
[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/03.png)

3. 实际中，我们会经常遇到线性不可分的样例，此时，我们的常用做法是把样例特征映射到高维空间中去(如下图)；

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/04.png)

3. 线性不可分映射到高维空间，可能会导致维度大小高到可怕的(19维乃至无穷维的例子)，导致计算复杂。核函数的价值在于它虽然也是讲特征进行从低维到高维的转换，但核函数绝就绝在它事先在低维上进行计算，而将实质上的分类效果表现在了高维上，也就如上文所说的避免了直接在高维空间中的复杂计算。

4.使用松弛变量处理数据噪音

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/05.png)

sklearn中SVM的结构，及各个参数说明如下

sklearn.svm.SVC ：

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,tol=0.001, cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape=None,random\_state=None)

参数说明：

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. C：C-SVC的惩罚参数C?默认值是1.0
2. C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。
3. kernel ：核函数，默认是rbf，可以是‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’
4. 0 – 线性：u'v
5. 1 – 多项式：(gamma\*u'\*v + coef0)^degree
6. 2 – RBF函数：exp(-gamma|u-v|^2)
7. 3 –sigmoid：tanh(gamma\*u'\*v + coef0)
8. degree ：多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。
9. gamma ： ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，则会选择1/n\_features
10. coef0 ：核函数的常数项。对于‘poly’和 ‘sigmoid’有用。
11. probability ：是否采用概率估计？.默认为False
12. shrinking ：是否采用shrinking heuristic方法，默认为**true**
13. tol ：停止训练的误差值大小，默认为1e-3
14. cache\_size ：核函数cache缓存大小，默认为200
15. class\_weight ：类别的权重，字典形式传递。设置第几类的参数C为weight\*C(C-SVC中的C)
16. verbose ：允许冗余输出？
17. max\_iter ：最大迭代次数。-1为无限制。
18. decision\_function\_shape ：‘ovo’, ‘ovr’ or None, **default**=None3
19. random\_state ：数据洗牌时的种子值，**int**值

主要调节的参数有：C、kernel、degree、gamma、coef0。

**系统环境**

Linux Ubuntu 16.04

Python3.6

**任务内容**

用SVM算法对fetch\_lfw\_people数据进行人脸识别，并将预测结果可视化。

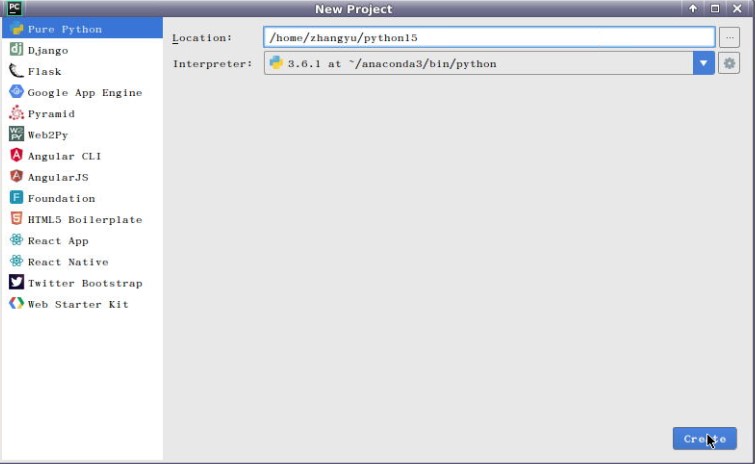
**任务步骤**

1.创建目录并下载实验所需的数据。

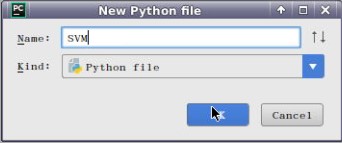
[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. mkdir -p /home/zhangyu/scikit\_learn\_data/lfw\_home
2. cd /home/zhangyu/scikit\_learn\_data/lfw\_home
3. wget http://192.168.1.100:60000/allfiles/ma\_learn/lfwfunneled.tgz
4. wget http://192.168.1.100:60000/allfiles/ma\_learn/pairsDevTest.txt
5. wget http://192.168.1.100:60000/allfiles/ma\_learn/pairsDevTrain.txt
6. wget http://192.168.1.100:60000/allfiles/ma\_learn/pairs.txt
7. tar xzvf lfwfunneled.tgz

2.新建Python project ，名为python15.

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/12.jpg)

在python15项目下，新建Python file，名为SVM

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/13.jpg)

3.用SVM算法对fetch\_lfw\_people数据进行人脸识别，并将预测结果可视化,完整代码如下：

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. from \_\_future\_\_ **import** print\_function
2. from time **import** time
3. **import** logging
4. **import** matplotlib.pyplot as plt
5. from sklearn.model\_selection **import** train\_test\_split
6. from sklearn.datasets **import** fetch\_lfw\_people
7. from sklearn.model\_selection **import** GridSearchCV
8. from sklearn.metrics **import** classification\_report
9. from sklearn.metrics **import** confusion\_matrix
10. from sklearn.decomposition **import** PCA
11. from sklearn.svm **import** SVC
12. # Display progress logs on stdout
13. logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')
14. ###############################################################################
15. # Download the data, if not already on disk and load it as numpy arrays
16. lfw\_people = fetch\_lfw\_people(min\_faces\_per\_person=70, resize=0.4)
17. # introspect the images arrays to find the shapes (for plotting)
18. n\_samples, h, w = lfw\_people.images.shape
19. # for machine learning we use the 2 data directly (as relative pixel
20. # positions info is ignored by this model)
21. X = lfw\_people.data
22. n\_features = X.shape[1]
23. # the label to predict is the id of the person
24. y = lfw\_people.target
25. target\_names = lfw\_people.target\_names
26. n\_classes = target\_names.shape[0]
27. print("Total dataset size:")
28. print("n\_samples: %d" % n\_samples)
29. print("n\_features: %d" % n\_features)
30. print("n\_classes: %d" % n\_classes)
31. ###############################################################################
32. # Split into a training set and a test set using a stratified k fold
33. # split into a training and testing set
34. X\_train, X\_test, y\_train, y\_test = train\_test\_split(
35. X, y, test\_size=0.25)
36. ###############################################################################
37. # Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
38. # dataset): unsupervised feature extraction / dimensionality reduction
39. n\_components = 150
40. print("Extracting the top %d eigenfaces from %d faces"
41. % (n\_components, X\_train.shape[0]))
42. t0 = time()
43. pca = PCA(svd\_solver='randomized',n\_components=n\_components, whiten=True).fit(X\_train)
44. print("done in %0.3fs" % (time() - t0))
45. eigenfaces = pca.components\_.reshape((n\_components, h, w))
46. print("Projecting the input data on the eigenfaces orthonormal basis")
47. t0 = time()
48. X\_train\_pca = pca.transform(X\_train)
49. X\_test\_pca = pca.transform(X\_test)
50. print("done in %0.3fs" % (time() - t0))
51. ###############################################################################
52. # Train a SVM classification model
53. print("Fitting the classifier to the training set")
54. t0 = time()
55. param\_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
56. 'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
57. clf = GridSearchCV(SVC(kernel='rbf', class\_weight='balanced'), param\_grid)
58. clf = clf.fit(X\_train\_pca, y\_train)
59. print("done in %0.3fs" % (time() - t0))
60. print("Best estimator found by grid search:")
61. print(clf.best\_estimator\_)
62. ###############################################################################
63. # Quantitative evaluation of the model quality on the test set
64. print("Predicting people's names on the test set")
65. t0 = time()
66. y\_pred = clf.predict(X\_test\_pca)
67. print("done in %0.3fs" % (time() - t0))
68. print(classification\_report(y\_test, y\_pred, target\_names=target\_names))
69. print(confusion\_matrix(y\_test, y\_pred, labels=range(n\_classes)))
70. ###############################################################################
71. # Qualitative evaluation of the predictions using matplotlib
72. def plot\_gallery(images, titles, h, w, n\_row=3, n\_col=4):
73. """Helper function to plot a gallery of portraits"""
74. plt.figure(figsize=(1.8 \* n\_col, 2.4 \* n\_row))
75. plt.subplots\_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
76. **for** i **in** range(n\_row \* n\_col):
77. plt.subplot(n\_row, n\_col, i + 1)
78. plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
79. plt.title(titles[i], size=12)
80. plt.xticks(())
81. plt.yticks(())
82. # plot the result of the prediction on a portion of the test set
83. def title(y\_pred, y\_test, target\_names, i):
84. pred\_name = target\_names[y\_pred[i]].rsplit(' ', 1)[-1]
85. true\_name = target\_names[y\_test[i]].rsplit(' ', 1)[-1]
86. **return** 'predicted: %s\ntrue:      %s' % (pred\_name, true\_name)
87. prediction\_titles = [title(y\_pred, y\_test, target\_names, i)
88. **for** i **in** range(y\_pred.shape[0])]
89. plot\_gallery(X\_test, prediction\_titles, h, w)
90. # plot the gallery of the most significative eigenfaces
91. eigenface\_titles = ["eigenface %d" % i **for** i **in** range(eigenfaces.shape[0])]
92. plot\_gallery(eigenfaces, eigenface\_titles, h, w)
93. plt.show()

4.对完整代码进行分部描述，用import导入实验所用到的包

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

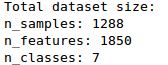
1. from \_\_future\_\_ **import** print\_function
2. from time **import** time
3. **import** logging
4. **import** matplotlib.pyplot as plt
5. from sklearn.model\_selection **import** train\_test\_split
6. from sklearn.datasets **import** fetch\_lfw\_people
7. from sklearn.model\_selection **import** GridSearchCV
8. from sklearn.metrics **import** classification\_report
9. from sklearn.metrics **import** confusion\_matrix
10. from sklearn.decomposition **import** PCA
11. from sklearn.svm **import** SVC

5.提取数据

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. lfw\_people = fetch\_lfw\_people(min\_faces\_per\_person=70, resize=0.4)
2. # introspect the images arrays to find the shapes (for plotting)
3. n\_samples, h, w = lfw\_people.images.shape
4. # for machine learning we use the 2 data directly (as relative pixel
5. # positions info is ignored by this model)
6. X = lfw\_people.data
7. n\_features = X.shape[1]
8. # the label to predict is the id of the person
9. y = lfw\_people.target
10. target\_names = lfw\_people.target\_names
11. n\_classes = target\_names.shape[0]
12. print("Total dataset size:")
13. print("n\_samples: %d" % n\_samples)
14. print("n\_features: %d" % n\_features)
15. print("n\_classes: %d" % n\_classes)

运行结果：

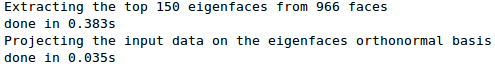
[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/06.png)

6.特征提取

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. X\_train, X\_test, y\_train, y\_test = train\_test\_split(
2. X, y, test\_size=0.25)
3. ###############################################################################
4. # Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
5. # dataset): unsupervised feature extraction / dimensionality reduction
6. n\_components = 150
7. print("Extracting the top %d eigenfaces from %d faces"
8. % (n\_components, X\_train.shape[0]))
9. t0 = time()
10. pca = PCA(svd\_solver='randomized',n\_components=n\_components, whiten=True).fit(X\_train)
11. print("done in %0.3fs" % (time() - t0))
12. eigenfaces = pca.components\_.reshape((n\_components, h, w))
13. print("Projecting the input data on the eigenfaces orthonormal basis")
14. t0 = time()
15. X\_train\_pca = pca.transform(X\_train)
16. X\_test\_pca = pca.transform(X\_test)
17. print("done in %0.3fs" % (time() - t0))

运行结果：

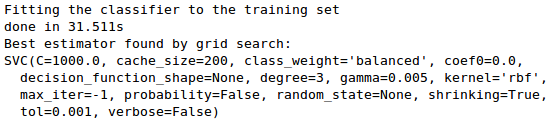
[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/07.png)

7.建立SVM分类模型

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. print("Fitting the classifier to the training set")
2. t0 = time()
3. param\_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
4. 'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
5. clf = GridSearchCV(SVC(kernel='rbf', class\_weight='balanced'), param\_grid)
6. clf = clf.fit(X\_train\_pca, y\_train)
7. print("done in %0.3fs" % (time() - t0))
8. print("Best estimator found by grid search:")
9. print(clf.best\_estimator\_)

运行结果：

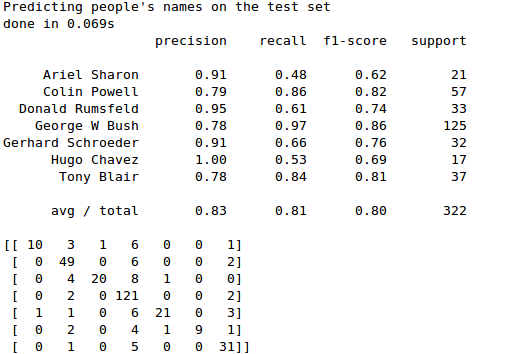
[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/08.png)

8.模型评估

[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. print("Predicting people's names on the test set")
2. t0 = time()
3. y\_pred = clf.predict(X\_test\_pca)
4. print("done in %0.3fs" % (time() - t0))
5. print(classification\_report(y\_test, y\_pred, target\_names=target\_names))
6. print(confusion\_matrix(y\_test, y\_pred, labels=range(n\_classes)))

运行结果：

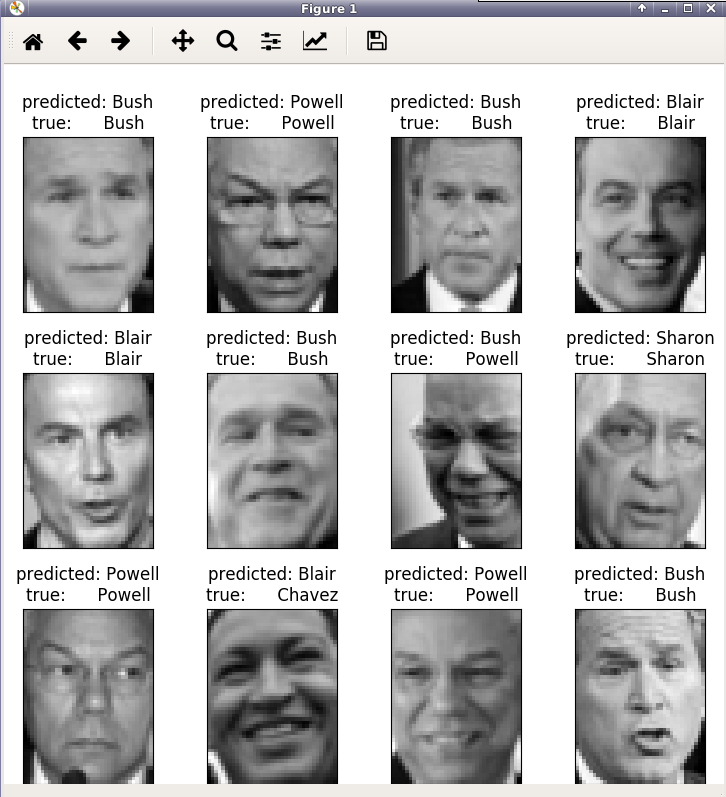
[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/09.png)

9.预测结果可视化

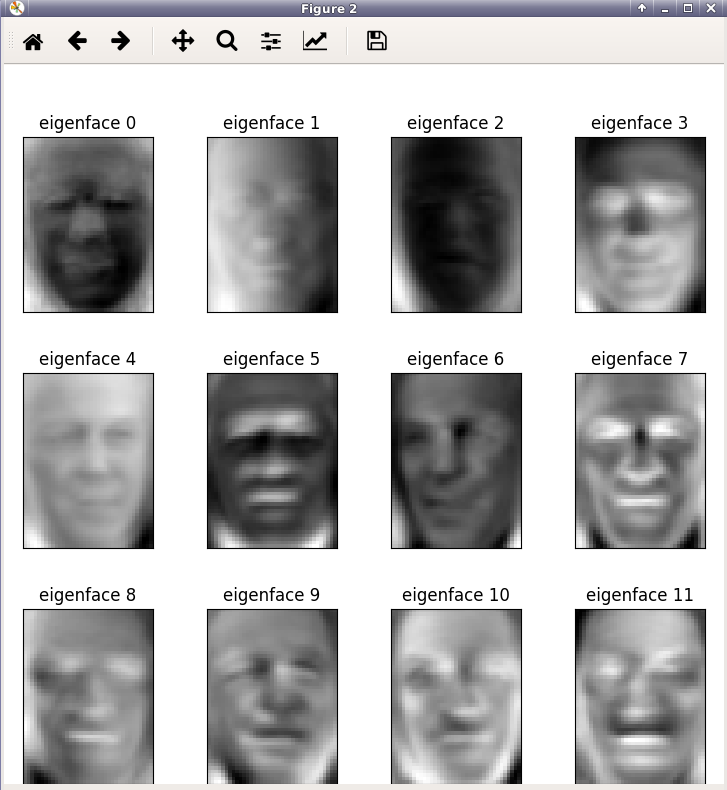
[view plain](https://www.ipieuvre.com/e/184/311/9539) [copy](https://www.ipieuvre.com/e/184/311/9539)

1. def plot\_gallery(images, titles, h, w, n\_row=3, n\_col=4):
2. """Helper function to plot a gallery of portraits"""
3. plt.figure(figsize=(1.8 \* n\_col, 2.4 \* n\_row))
4. plt.subplots\_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
5. **for** i **in** range(n\_row \* n\_col):
6. plt.subplot(n\_row, n\_col, i + 1)
7. plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
8. plt.title(titles[i], size=12)
9. plt.xticks(())
10. plt.yticks(())
11. # plot the result of the prediction on a portion of the test set
12. def title(y\_pred, y\_test, target\_names, i):
13. pred\_name = target\_names[y\_pred[i]].rsplit(' ', 1)[-1]
14. true\_name = target\_names[y\_test[i]].rsplit(' ', 1)[-1]
15. **return** 'predicted: %s\ntrue:      %s' % (pred\_name, true\_name)
16. prediction\_titles = [title(y\_pred, y\_test, target\_names, i)
17. **for** i **in** range(y\_pred.shape[0])]
18. plot\_gallery(X\_test, prediction\_titles, h, w)
19. # plot the gallery of the most significative eigenfaces
20. eigenface\_titles = ["eigenface %d" % i **for** i **in** range(eigenfaces.shape[0])]
21. plot\_gallery(eigenfaces, eigenface\_titles, h, w)
22. plt.show()

运行结果：

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/10.png)

eigenface：

[](https://www.ipieuvre.com/doc/exper/289a442c-91ad-11e9-beeb-00215ec892f4/img/11.png)