Exercise 1

您将如何定义集群？你能列举出一些聚类算法吗？

答案：

在机器学习中，聚类是将相似实例分组在一起的无监督任务。相似性的概念取决于手头的任务：例如，在某些情况下，两个附近的实例将被视为相似，而在其他情况下，只要它们属于同一个密集组，相似的实例可能相距甚远。

流行的聚类算法包括 K-Means、DBSCAN、agglomerative clustering、BIRCH、Mean-Shift、亲和力传播、spectral。

Exercise 2

聚类算法的一些主要应用是什么？

答案：

聚类算法的主要应用包括数据分析、客户细分、推荐系统、搜索引擎、图像分割、半监督学习、降维、异常检测和新颖性检测。

Exercise 3

描述在使用 k-means 时选择正确数量的集群的两种技术。

答案：

拐点规则（Elbow Rule） 是使用 K-Means 时选择聚类数量的一种简单技术：只需绘制惯性（从每个实例到最近的质心的均方距离）作为聚类数量的函数，并找到惯性停止快速下降的曲线（“弯头”）上的点。这通常接近最佳簇数。

另一种方法是将轮廓分数绘制为聚类数量的函数。往往会有一个峰值，最佳簇数一般就在附近。轮廓得分是所有实例的平均轮廓系数。该系数从位于集群内部且远离其他集群的实例的 +1 到与另一个集群非常接近的实例的 –1 不等。您还可以绘制轮廓图并进行更彻底的分析。

Exercise 4

什么是标签传播？你为什么要实现它，以及如何实现它？

答案：

标记数据集既费钱又费时。因此，通常有大量未标记的实例，但很少有标记的实例。标签传播是一种技术，包括将部分（或全部）标签从已标记实例复制到类似的未标记实例。这可以极大地扩展标记实例的数量，从而使监督算法达到更好的性能（这是半监督学习的一种形式）。一种方法是对所有实例使用聚类算法，例如 K-Means，然后为每个集群找到最常见的标签或最具代表性实例的标签（即最接近质心的标签）并将其传播到同一集群中未标记的实例。

Exercise 5

你能说出两种可以扩展到大型数据集的聚类算法吗？还有两个寻找高密度区域的？

答案：

K-Means 和 BIRCH 可以很好地扩展到大型数据集。DBSCAN 和 Mean-Shift 寻找高密度区域。

Exercise 6

你能想到一个对主动学习会很有用的用例吗？你将如何实现它？

答案：

当你有大量未标记的实例但标记成本很高时，主动学习很有用。在这种情况下（这很常见），与其随机选择要标记的实例，不如执行主动学习，在这种情况下，人类专家与学习算法进行交互，在算法请求时为特定实例提供标签。一种常见的方法是不确定性抽样（参见第 9 章的主动学习部分）。

Exercise 7

异常检测和新颖性检测有什么区别？

答案：

许多人互换使用异常检测和新颖性检测这两个术语，但它们并不完全相同。

- 在异常检测中，算法在可能包含异常值的数据集上进行训练，目标通常是识别这些异常值（在训练集中），以及新实例中的异常值。
- 在新颖性检测中，算法在假定为"干净"的数据集上进行训练，目标是严格检测新实例中的新颖性。

一些算法最适合异常检测（Isolation Forest），而其他算法更适合新颖性检测（one-class SVM）。

Exercise 8

什么是 Gaussian mixture？你可以用它来做什么任务呢？

答案：

高斯混合模型 (GMM) 是一种概率模型，它假设实例是从几个参数未知的高斯分布的混合中生成的。换句话说，假设数据被分组到有限数量的簇中，每个簇都具有椭圆形状（但簇可能具有不同的椭圆形状、大小、方向和密度），并且我们不知道每个实例属于哪个簇。该模型可用于密度估计、聚类和异常检测。

Exercise 9

当使用高斯混合模型时，你能说出两种找到正确簇数量的技术吗？

答案：

使用高斯混合模型时找到正确簇数的一种方法是绘制贝叶斯信息准则 (BIC) 或 Akaike 信息准则 (AIC) 作为簇数的函数，然后选择最小化簇数 BIC 或 AIC。 另一种技术是使用贝叶斯高斯混合模型，它会自动选择聚类的数

量。

Exercise 10

经典的 Olivetti 人脸数据集包含 400 张 64 × 64 像素的人脸灰度图像。每个图像都被展平为大小为 4,096 的一维矢量。拍摄了 40 个不同的人（每个人 10 次），通常的任务是训练一个模型来预测每张照片中代表的是哪个人。使用 sklearn.datasets.fetch_olivetti_faces() 函数加载数据集，然后将其拆分为训练集、验证集和测试集（注意数据集已经在 0 和 1 之间缩放）。由于数据集非常小，您可能希望使用分层抽样来确保每组中每个人的图像数量相同。接下来，使用 k-means 对图像进行聚类，并确保您有足够数量的聚类（使用本章讨论的一种技术）。可视化聚类：您是否在每个聚类中看到相似的面孔？

答案：

In [2]:
```python
from sklearn.datasets import fetch_olivetti_faces

olivetti = fetch_olivetti_faces()
```
```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to C:\Use
rs\tu'tu\scikit_learn_data
```

In [3]:
```python
print(olivetti.DESCR)
```
```
.. _olivetti_faces_dataset:

The Olivetti faces dataset
--------------------------

`This dataset contains a set of face images`_ taken between April 1992 and
April 1994 at AT&T Laboratories Cambridge. The
:func:`sklearn.datasets.fetch_olivetti_faces` function is the data
fetching / caching function that downloads the data
archive from AT&T.

.. _This dataset contains a set of face images: https://cam-orl.co.uk/facedatabase.html

As described on the original website:

    There are ten different images of each of 40 distinct subjects. For some
    subjects, the images were taken at different times, varying the lighting,
    facial expressions (open / closed eyes, smiling / not smiling) and facial
    details (glasses / no glasses). All the images were taken against a dark
    homogeneous background with the subjects in an upright, frontal position
    (with tolerance for some side movement).

**Data Set Characteristics:**

    =================   =====================
    Classes                                40
    Samples total                         400
    Dimensionality                       4096
    Features            real, between 0 and 1
    =================   =====================

The image is quantized to 256 grey levels and stored as unsigned 8-bit
integers; the loader will convert these to floating point values on the
interval [0, 1], which are easier to work with for many algorithms.

The "target" for this database is an integer from 0 to 39 indicating the
identity of the person pictured; however, with only 10 examples per class, this
relatively small dataset is more interesting from an unsupervised or
semi-supervised perspective.
```

```
The original dataset consisted of 92 x 112, while the version available here
consists of 64x64 images.

When using these images, please give credit to AT&T Laboratories Cambridge.
```

In [4]:
```python
olivetti.target
```

Out[4]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,
        5,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
       17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18,
       18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20,
       20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22,
       22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23,
       23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25,
       25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27,
       27, 27, 27, 27, 27, 27, 27, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28,
       28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30,
       30, 30, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32,
       32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
       34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35,
       35, 35, 35, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 37, 37, 37,
       37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 39])
```

然后将其拆分为训练集、验证集和测试集（注意数据集已经在 0 和 1 之间缩放）。由于数据集非常小，您可能希望使用分层抽样来确保每组中每个人的图像数量相同。

In [5]:
```python
from sklearn.model_selection import StratifiedShuffleSplit

strat_split = StratifiedShuffleSplit(n_splits=1, test_size=40, random_state=42)

train_valid_idx, test_idx = next(strat_split.split(olivetti.data,
                                                   olivetti.target))

X_train_valid = olivetti.data[train_valid_idx]
y_train_valid = olivetti.target[train_valid_idx]
X_test = olivetti.data[test_idx]
y_test = olivetti.target[test_idx]

strat_split = StratifiedShuffleSplit(n_splits=1, test_size=80, random_state=43)

train_idx, valid_idx = next(strat_split.split(X_train_valid, y_train_valid))

X_train = X_train_valid[train_idx]
y_train = y_train_valid[train_idx]
X_valid = X_train_valid[valid_idx]
y_valid = y_train_valid[valid_idx]
```

In [12]:
```python
print(X_train.shape, y_train.shape)
print(X_valid.shape, y_valid.shape)
print(X_test.shape, y_test.shape)
```

```
(280, 4096) (280,)
(80, 4096) (80,)
(40, 4096) (40,)
```

为了加快速度，我们将使用 PCA 降低数据的维度：

In [13]:
```python
from sklearn.decomposition import PCA

pca = PCA(0.99)

X_train_pca = pca.fit_transform(X_train)
X_valid_pca = pca.transform(X_valid)
X_test_pca = pca.transform(X_test)

pca.n_components_
```

Out[13]:
```
199
```

接下来，使用 K-Means 对图像进行聚类，并确保您有足够数量的聚类（使用本章讨论的一种技术）。

In [14]:
```python
from sklearn.cluster import KMeans

k_range = range(5, 150, 5)

kmeans_per_k = []

for k in k_range:
    print(f"k={k}")
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_pca)
    kmeans_per_k.append(kmeans)
```

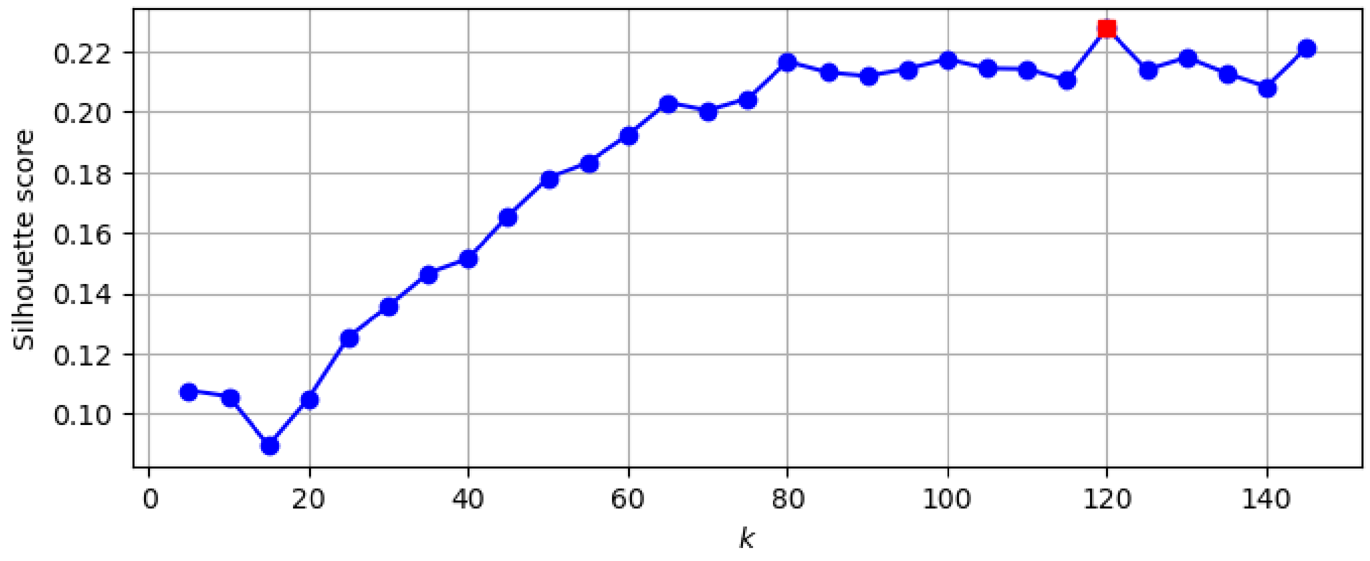
```
k=5
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=10
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=15
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=20
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=25
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=30
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
```

```
DS=2.
  warnings.warn(
k=35
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=40
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=45
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=50
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=55
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=60
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=65
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=70
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=75
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=80
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=85
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=90
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=95
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=100
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=105
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=110
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=115
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=120
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=125
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=130
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=135
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
```

```
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=140
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
k=145
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
```

In [19]:

```python
from sklearn.metrics import silhouette_score
import numpy as np
import matplotlib.pyplot as plt

silhouette_scores = [silhouette_score(X_train_pca, model.labels_)
                     for model in kmeans_per_k]

best_index = np.argmax(silhouette_scores)
best_k = k_range[best_index]
best_score = silhouette_scores[best_index]

plt.figure(figsize=(8, 3))
plt.plot(k_range, silhouette_scores, "bo-")
plt.xlabel("$k$")
plt.ylabel("Silhouette score")
plt.plot(best_k, best_score, "rs")
plt.grid()
plt.show()
```



In [20]:

```python
best_k
```

Out[20]:

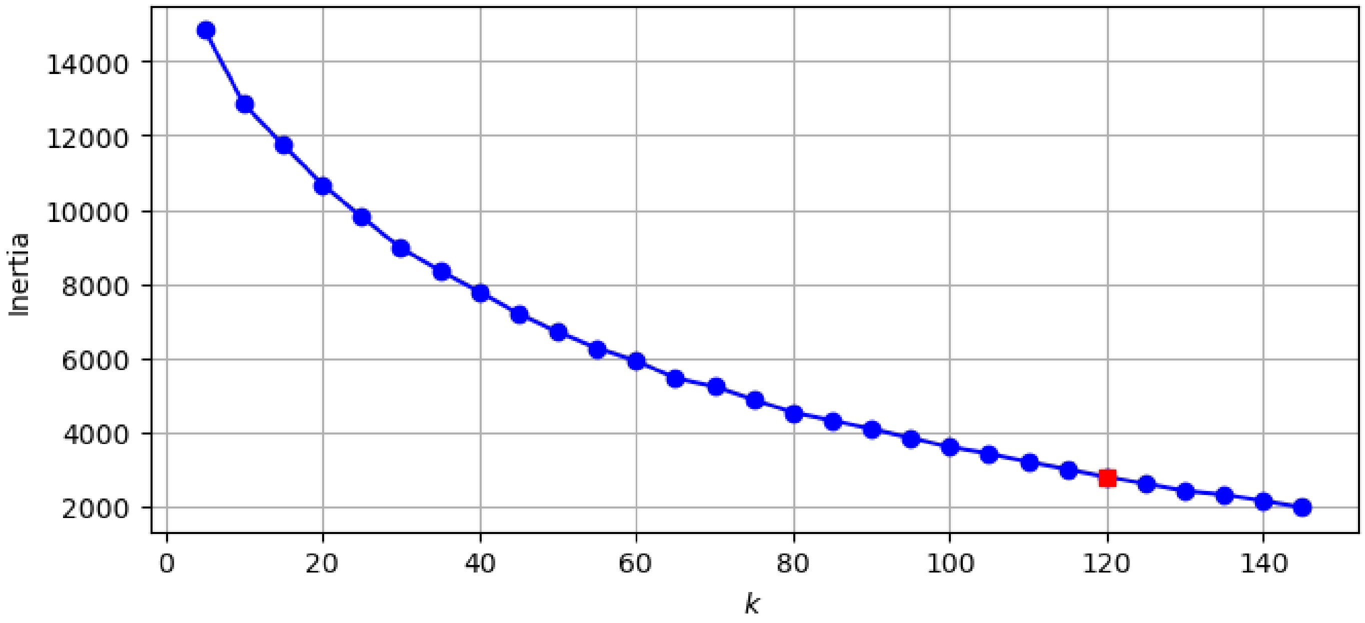
```
120
```

看起来最佳聚类数相当高，为 120。您可能期望它是 40，因为图片上有 40 个不同的人。然而，同一个人在不同的照片上可能看起来完全不同（例如，戴眼镜或不戴眼镜，或者只是向左或向右移动）。

In [21]:

```python
inertias = [model.inertia_ for model in kmeans_per_k]
```

```
best_inertia = inertias[best_index]

plt.figure(figsize=(8, 3.5))
plt.plot(k_range, inertias, "bo-")
plt.xlabel("$k$")
plt.ylabel("Inertia")
plt.plot(best_k, best_inertia, "rs")
plt.grid()
plt.show()
```



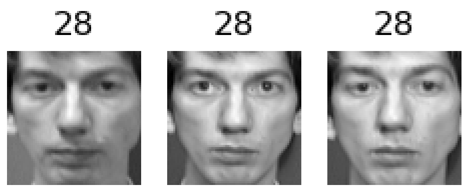
这个惯性图上的最佳簇数并不清楚，因为没有明显的弯头，所以让我们坚持 k=120 。

In [22]:
```
best_model = kmeans_per_k[best_index]
```

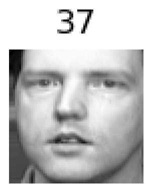
可视化聚类：您是否在每个聚类中看到相似的面孔？

In [23]:
```
def plot_faces(faces, labels, n_cols=5):
    faces = faces.reshape(-1, 64, 64)
    n_rows = (len(faces) - 1) // n_cols + 1
    plt.figure(figsize=(n_cols, n_rows * 1.1))
    for index, (face, label) in enumerate(zip(faces, labels)):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(face, cmap="gray")
        plt.axis("off")
        plt.title(label)
    plt.show()

for cluster_id in np.unique(best_model.labels_):
    print("Cluster", cluster_id)
    in_cluster = best_model.labels_==cluster_id
    faces = X_train[in_cluster]
    labels = y_train[in_cluster]
    plot_faces(faces, labels)
```

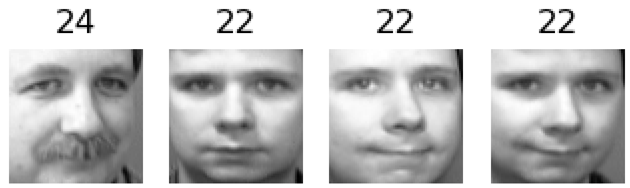
```
Cluster 0
```



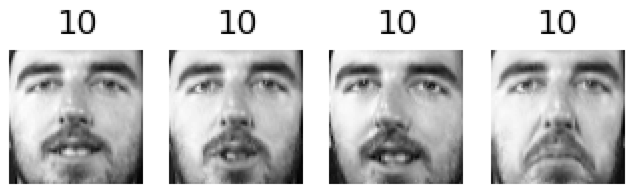
```
Cluster 1
```



```
Cluster 2
```

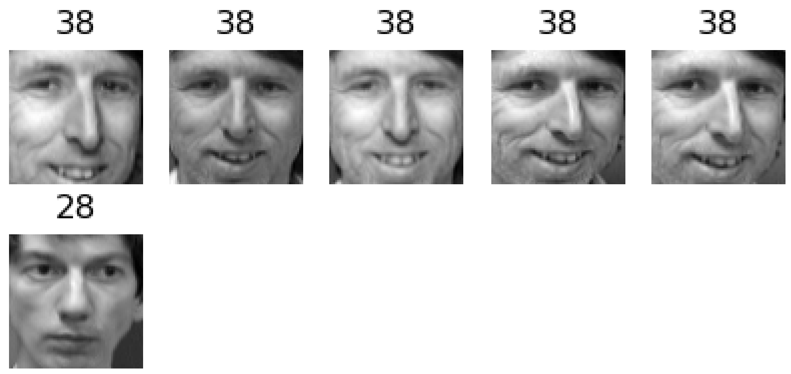


```
Cluster 3
```

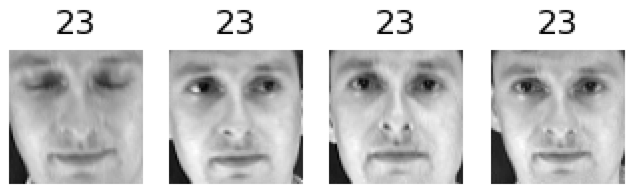


```
Cluster 4
```

```
Cluster 5
```

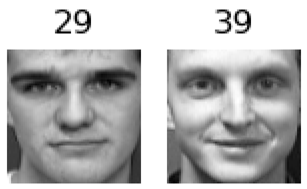


```
Cluster 6
```



```
Cluster 7
```

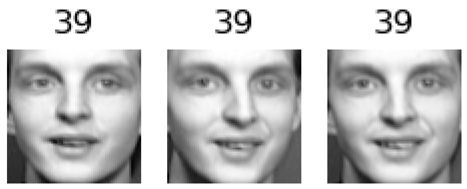
```
Cluster 8
```



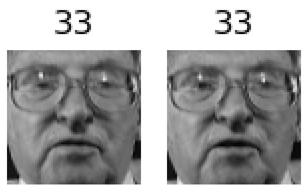
Cluster 9

Cluster 10

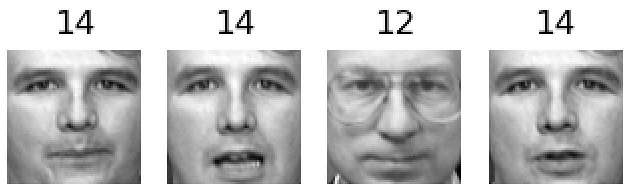
Cluster 11



Cluster 12



Cluster 13

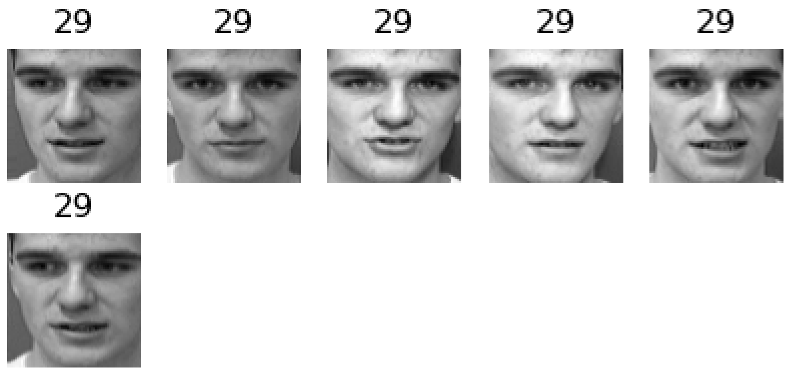


Cluster 14

Cluster 15

Cluster 16

Cluster 17

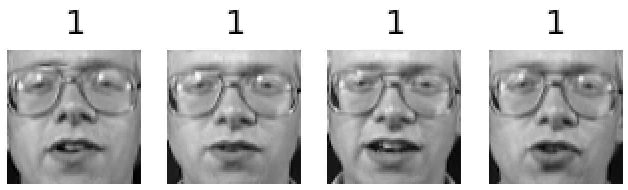


Cluster 18

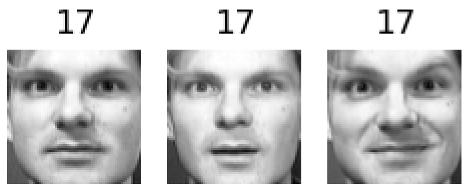
Cluster 19

Cluster 20

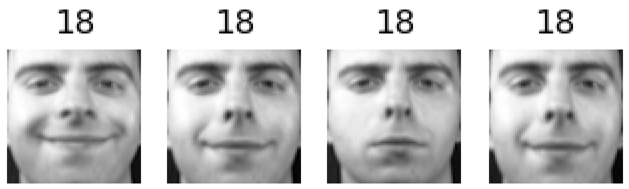
Cluster 21



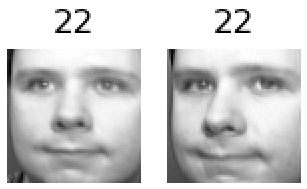
Cluster 22



Cluster 23



Cluster 24

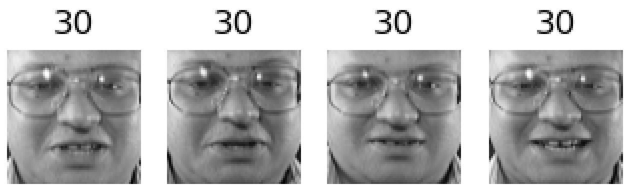


Cluster 25

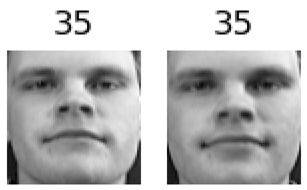
Cluster 26

Cluster 27

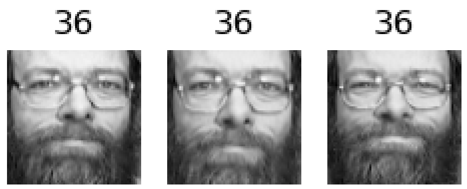
Cluster 28



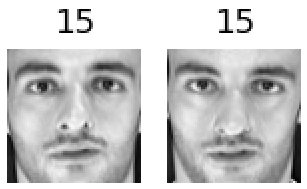
Cluster 29



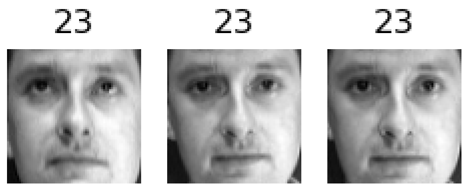
Cluster 30



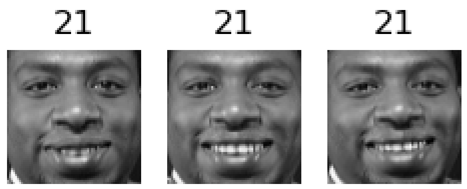
Cluster 31



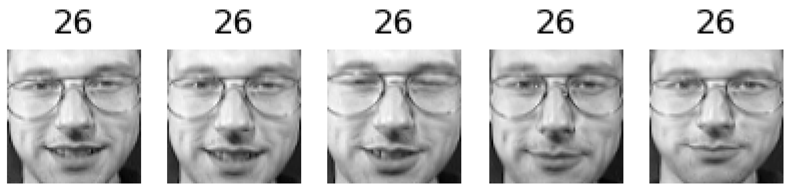
```
Cluster 32
```



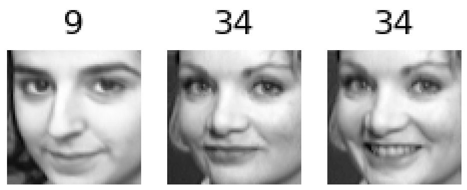
```
Cluster 33
```



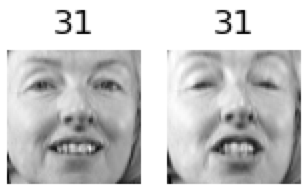
```
Cluster 34
```



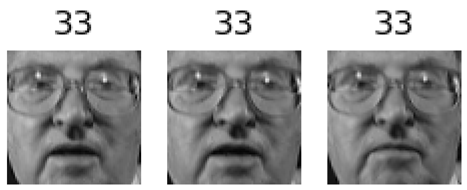
```
Cluster 35
```



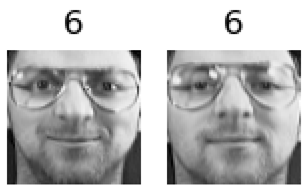
```
Cluster 36
```



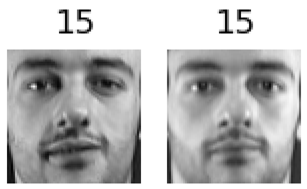
```
Cluster 37
```



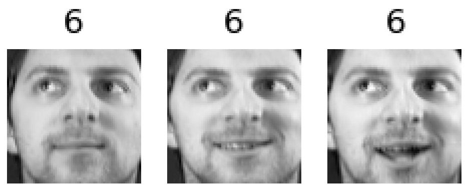
```
Cluster 38
```



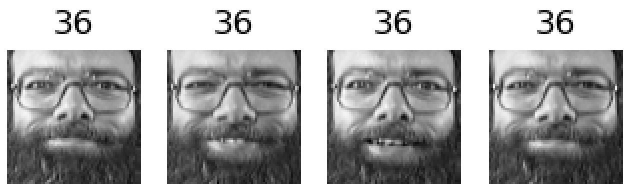
```
Cluster 39
```



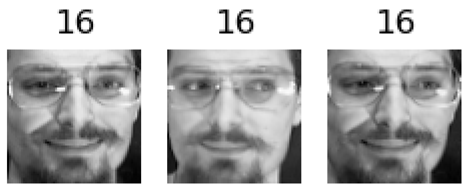
Cluster 40



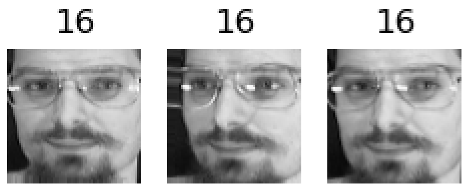
Cluster 41



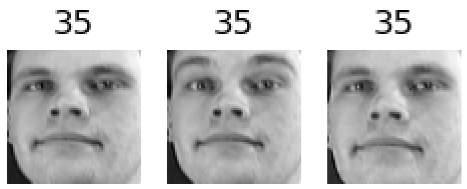
Cluster 42



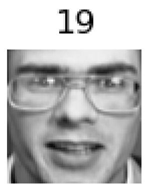
Cluster 43



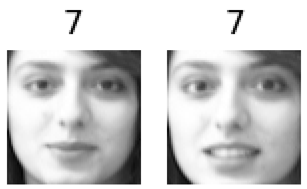
Cluster 44



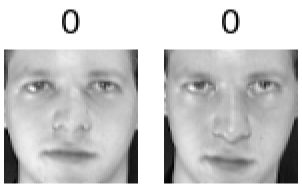
Cluster 45



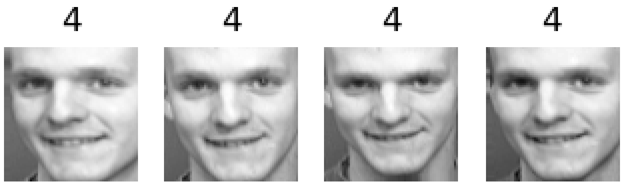
Cluster 46



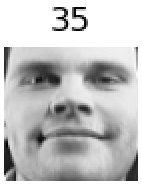
Cluster 47



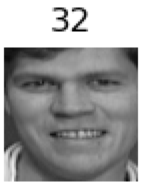
```
Cluster 48
```



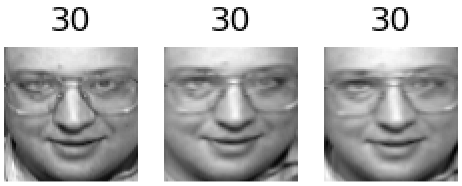
```
Cluster 49
```



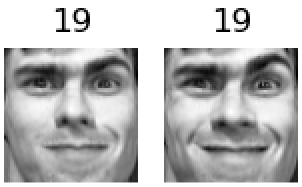
```
Cluster 50
```



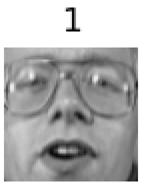
```
Cluster 51
```



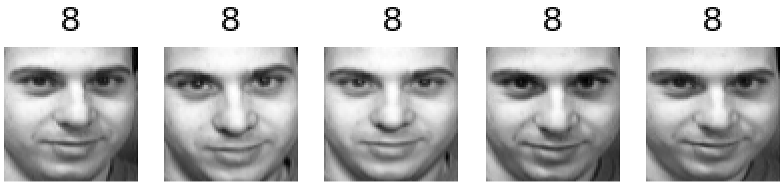
```
Cluster 52
```



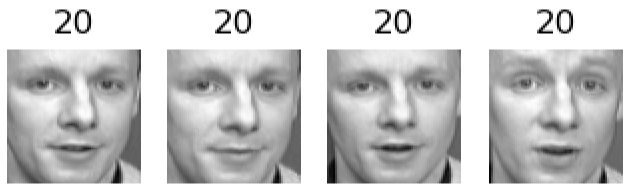
```
Cluster 53
```



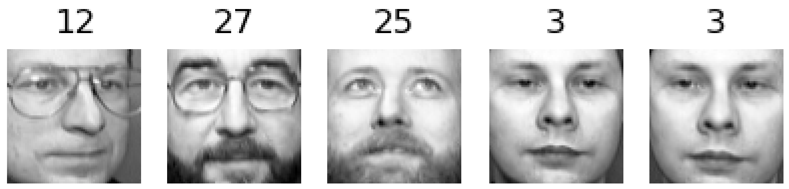
```
Cluster 54
```



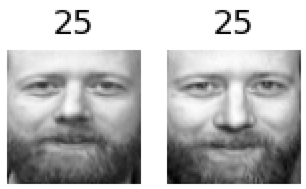
```
Cluster 55
```



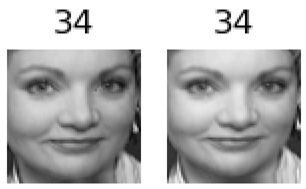
Cluster 56



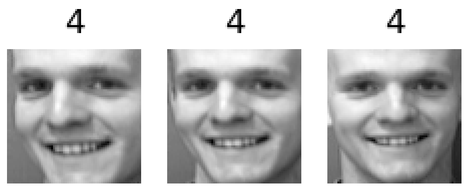
Cluster 57



Cluster 58



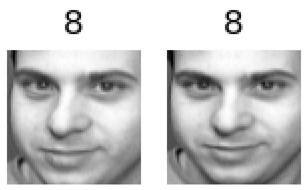
Cluster 59



Cluster 60

Cluster 61

Cluster 62



Cluster 63

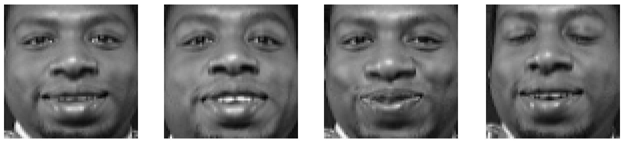
27

Cluster 64

15

Cluster 65

21 21 21 21



Cluster 66

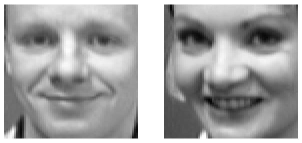
6

Cluster 67

3 3 3

Cluster 68

20 34



Cluster 69

34 34

Cluster 70

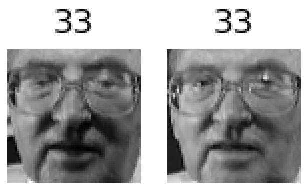
37 37 37 37

Cluster 71

Cluster 72

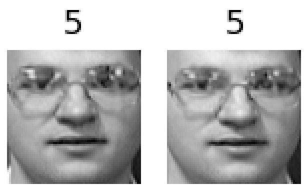
Cluster 73

Cluster 74

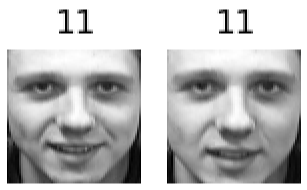


Cluster 75

Cluster 76



Cluster 77



Cluster 78

Cluster 79

3

Cluster 80

7

Cluster 81

9

Cluster 82

13 13

Cluster 83

10 10

Cluster 84

1 1

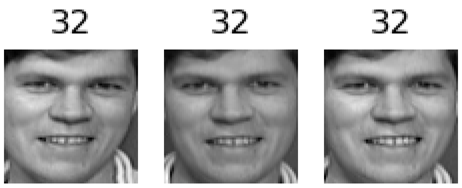
Cluster 85

18 18

Cluster 86

0

Cluster 87



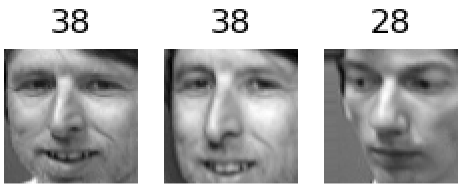
```
Cluster 88
```

```
Cluster 89
```

```
Cluster 90
```

```
Cluster 91
```

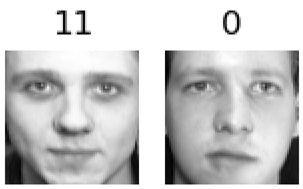
```
Cluster 92
```



```
Cluster 93
```

```
Cluster 94
```

```
Cluster 95
```



Cluster 96

Cluster 97

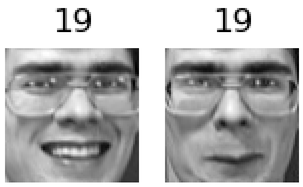
Cluster 98

Cluster 99

Cluster 100

Cluster 101

Cluster 102



Cluster 103

```
Cluster 104
```

```
Cluster 105
```

```
Cluster 106
```

```
Cluster 107
```

```
Cluster 108
```

```
Cluster 109
```

```
Cluster 110
```

```
Cluster 111
```

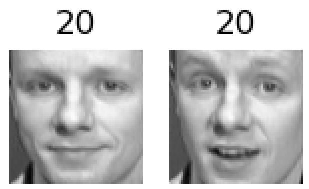
Cluster 112

Cluster 113

Cluster 114

Cluster 115

Cluster 116



Cluster 117

Cluster 118

Cluster 119

大约三分之二的聚类是有用的：也就是说，它们至少包含两张图片，都是同一个人。然而，其余集群要么有一个或多个入侵者，要么只有一张图片。

以这种方式对图像进行聚类可能过于不精确，无法在训练模型时直接发挥作用（正如我们将在下面看到的），但在对新数据集中的图像进行标记时却非常有用：它通常会使标记速度更快。

Exercise 11

继续使用 Olivetti 人脸数据集，训练一个分类器来预测每个图片中代表了哪个人，并在验证集中对其进行评估。接下来，使用 k-means 作为降维工具，并在降维集上训练一个分类器。搜索允许分类器获得最佳性能的集群的数量：您可以达到什么性能？如果您将简化集中的特性附加到原始特性中（同样，搜索集群的最佳数量），会怎么样？

答案：

继续使用 Olivetti 人脸数据集，训练一个分类器来预测每张图片中代表的是哪个人，并在验证集上对其进行评估。

In [25]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=150, random_state=42)

clf.fit(X_train_pca, y_train)

clf.score(X_valid_pca, y_valid)
```

Out[25]:
```
0.9
```

接下来，使用 K-Means 作为降维工具，并在降维集上训练分类器。

In [26]:
```
X_train_reduced = best_model.transform(X_train_pca)
X_valid_reduced = best_model.transform(X_valid_pca)
X_test_reduced = best_model.transform(X_test_pca)

clf = RandomForestClassifier(n_estimators=150, random_state=42)
clf.fit(X_train_reduced, y_train)

clf.score(X_valid_reduced, y_valid)
```

Out[26]:
```
0.7
```

哎呀！那一点都不好！让我们看看调整集群的数量是否有帮助。

搜索允许分类器获得最佳性能的簇数：你可以达到什么性能？

我们可以像之前在本笔记本中所做的那样使用 **GridSearchCV**，但由于我们已经有了验证集，所以我们不需要 K 折交叉验证，而且我们只探索单个超参数，因此手动运行更简单的循环：

```
from sklearn.pipeline import make_pipeline
```

In [27]:
```
for n_clusters in k_range:
    pipeline = make_pipeline(
        KMeans(n_clusters=n_clusters, random_state=42),
        RandomForestClassifier(n_estimators=150, random_state=42)
    )
    pipeline.fit(X_train_pca, y_train)
    print(n_clusters, pipeline.score(X_valid_pca, y_valid))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
5 0.3875
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
10 0.575
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
15 0.6
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
20 0.6625
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
25 0.6625
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
30 0.6625
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
35 0.675
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
40 0.75
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
45 0.7375
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
50 0.725
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
55 0.7125
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
60 0.7125
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
65 0.7375
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
70 0.7375
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
75 0.7375
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
80 0.7875
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
85 0.7625
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
90 0.75
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
95 0.7125
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
```

```
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
100 0.775
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
105 0.75
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
110 0.725
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
115 0.7125
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
120 0.7
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
125 0.75
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
130 0.725
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
135 0.75
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
140 0.7625
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
145 0.6875
```

哦，好吧，即使通过调整集群的数量，我们也永远不会超过 80% 的准确率。看起来到簇质心的距离不像原始图像那样信息丰富。

如果您将缩减集中的特征附加到原始特征（再次搜索最佳数量的聚类）怎么办？

In [28]:
```
X_train_extended = np.c_[X_train_pca, X_train_reduced]
X_valid_extended = np.c_[X_valid_pca, X_valid_reduced]
X_test_extended = np.c_[X_test_pca, X_test_reduced]
```

In [29]:
```
clf = RandomForestClassifier(n_estimators=150, random_state=42)
clf.fit(X_train_extended, y_train)
clf.score(X_valid_extended, y_valid)
```

Out[29]:
```
0.8125
```

这比没有集群功能要好一些，但仍然很糟。在这种情况下，集群对于直接训练分类器没有用（但在标记新训练实例时它们仍然可以提供帮助）。

Exercise 12

在 Olivetti 人脸数据集上训练高斯混合模型。为了加快算法速度，您可能应该降低数据集的维度（例如，使用 PCA，保留 99% 的方差）。使用模型生成一些新面孔（使用 sample() 方法），并将它们可视化（如果您使用 PCA，则需要使用它的 inverse_transform() 方法）。尝试修改一些图像（例如，旋转、翻转、变暗）并查看模型是否可以检测到异常（即，比较正常图像和异常图像的 score_samples() 方法的输出）。

答案：

在 Olivetti 人脸数据集上训练高斯混合模型。要加快算法速度，您可能应该降低数据集的维度（例如，使用 PCA，保留 99% 的方差）。

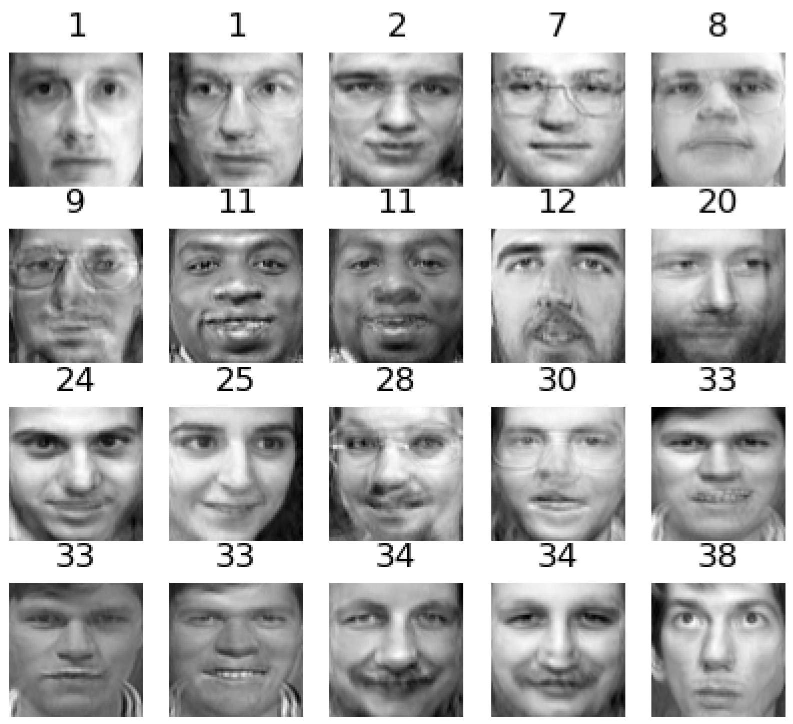
In [30]:
```
from sklearn.mixture import GaussianMixture

gm = GaussianMixture(n_components=40, random_state=42)

y_pred = gm.fit_predict(X_train_pca)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=2.
  warnings.warn(
```

使用模型生成一些新面孔（使用 sample() 方法），并将它们可视化（如果您使用 PCA，则需要使用它的 inverse_transform() 方法）。

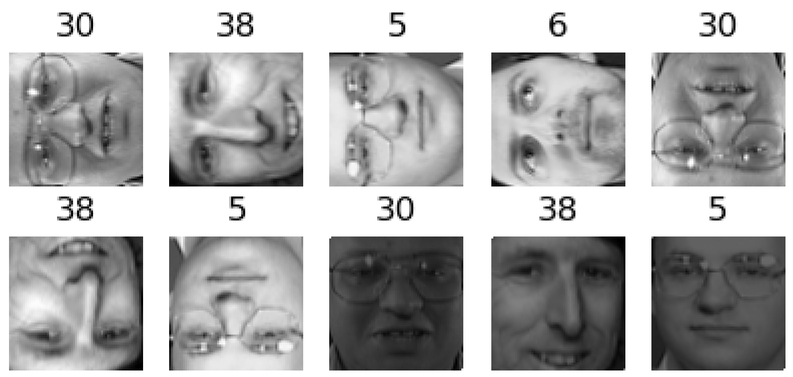
In [31]:
```
n_gen_faces = 20
gen_faces_reduced, y_gen_faces = gm.sample(n_samples=n_gen_faces)
gen_faces = pca.inverse_transform(gen_faces_reduced)
```

In [32]:
```
plot_faces(gen_faces, y_gen_faces)
```



尝试修改一些图像（例如，旋转、翻转、变暗）并查看模型是否可以检测到异常（即，比较正常图像和异常图像的 score_samples() 方法的输出）。

In [34]:
```
n_rotated = 4
rotated = np.transpose(X_train[:n_rotated].reshape(-1, 64, 64), axes=[0, 2, 1])
rotated = rotated.reshape(-1, 64*64)
y_rotated = y_train[:n_rotated]

n_flipped = 3
flipped = X_train[:n_flipped].reshape(-1, 64, 64)[:, ::-1]
flipped = flipped.reshape(-1, 64*64)
y_flipped = y_train[:n_flipped]

n_darkened = 3
darkened = X_train[:n_darkened].copy()
darkened[:, 1:-1] *= 0.3
y_darkened = y_train[:n_darkened]

X_bad_faces = np.r_[rotated, flipped, darkened]
y_bad = np.concatenate([y_rotated, y_flipped, y_darkened])

plot_faces(X_bad_faces, y_bad)
```



In [35]:
```
X_bad_faces_pca = pca.transform(X_bad_faces)
```

In [36]:
```
gm.score_samples(X_bad_faces_pca)
```

```
Out[36]: array([-2.43643136e+07, -1.89784915e+07, -3.78112338e+07, -4.98187720e+07,
                -3.20479016e+07, -1.37531240e+07, -2.92374135e+07, -1.05488872e+08,
                -1.19575129e+08, -6.74255524e+07])
```

高斯混合模型认为坏脸都不太可能出现。将此与一些训练实例的分数进行比较：

```
In [37]: gm.score_samples(X_train_pca[:10])
```

```
Out[37]: array([1163.02020883, 1134.03638083, 1156.3213269 , 1170.67602707,
                1141.45404746, 1154.35205119, 1091.32894431, 1111.41149386,
                1096.43048917, 1132.98982662])
```

Exercise 13

一些降维技术也可以用于异常检测。例如，以 Olivetti 人脸数据集为例，用 PCA 来减少它，并保留了 99% 的方差。然后计算每幅图像的重建误差。接下来，取您在前面的练习中构建的一些修改过的图像，看看它们的重建错误：注意它有多大。如果你绘制一个重建的图像，你就会看到为什么：它试图重建一个正常的人脸。

答案：

我们之前已经使用 PCA 减少了数据集：

```
In [38]: X_train_pca.round(2)
```

```
Out[38]: array([[  3.78,  -1.85,  -5.14, ...,  -0.14,  -0.21,   0.06],
                [ 10.15,  -1.53,  -0.77, ...,   0.12,  -0.14,  -0.02],
                [-10.02,   2.88,  -0.92, ...,   0.07,  -0.  ,   0.12],
                ...,
                [  2.48,   2.96,   1.3 , ...,  -0.02,   0.03,  -0.15],
                [ -3.22,   5.35,   1.39, ...,   0.06,  -0.23,   0.16],
                [ -0.92,  -3.65,   2.26, ...,   0.14,  -0.07,   0.06]],
              dtype=float32)
```

```
In [39]: def reconstruction_errors(pca, X):
             X_pca = pca.transform(X)
             X_reconstructed = pca.inverse_transform(X_pca)
             mse = np.square(X_reconstructed - X).mean(axis=-1)
             return mse
```

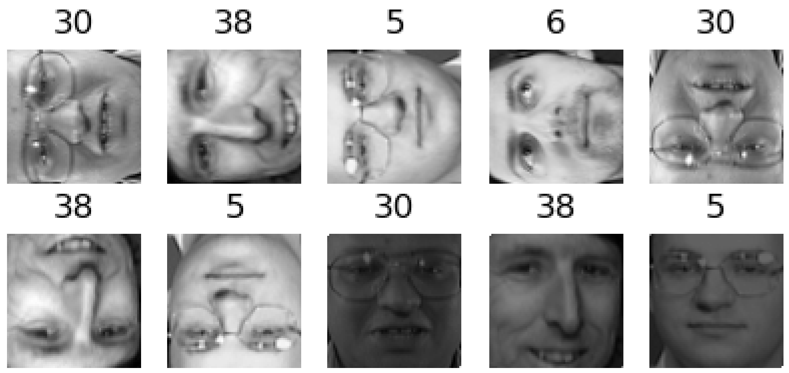
```
In [40]: reconstruction_errors(pca, X_train).mean()
```

```
Out[40]: 0.0001920535
```

```
In [41]: reconstruction_errors(pca, X_bad_faces).mean()
```

```
Out[41]: 0.004707354
```

```
In [42]: plot_faces(X_bad_faces, y_bad)
```



In [43]:
```
X_bad_faces_reconstructed = pca.inverse_transform(X_bad_faces_pca)
plot_faces(X_bad_faces_reconstructed, y_bad)
```