얼굴 데이터 셋으로 군집 알고리즘 비교

학습 내용

- 얼굴 데이터 셋을 이용한 다양한 비지도 학습 분석을 수행해 봅니다.

목차

01. 라이브러리 불러오기 및 데이터 준비
02. PCA, DBSCAN 활용한 데이터 분석
03. 모델을 활용한 이상치 검출
04. K-Mean로 얼굴 데이터 분석

01. 라이브러리 불러오기 및 데이터 준비

목차로 이동하기

In [1]:

```
# !pip install mglearn
```

In [2]:

```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.decomposition import NMF
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
```

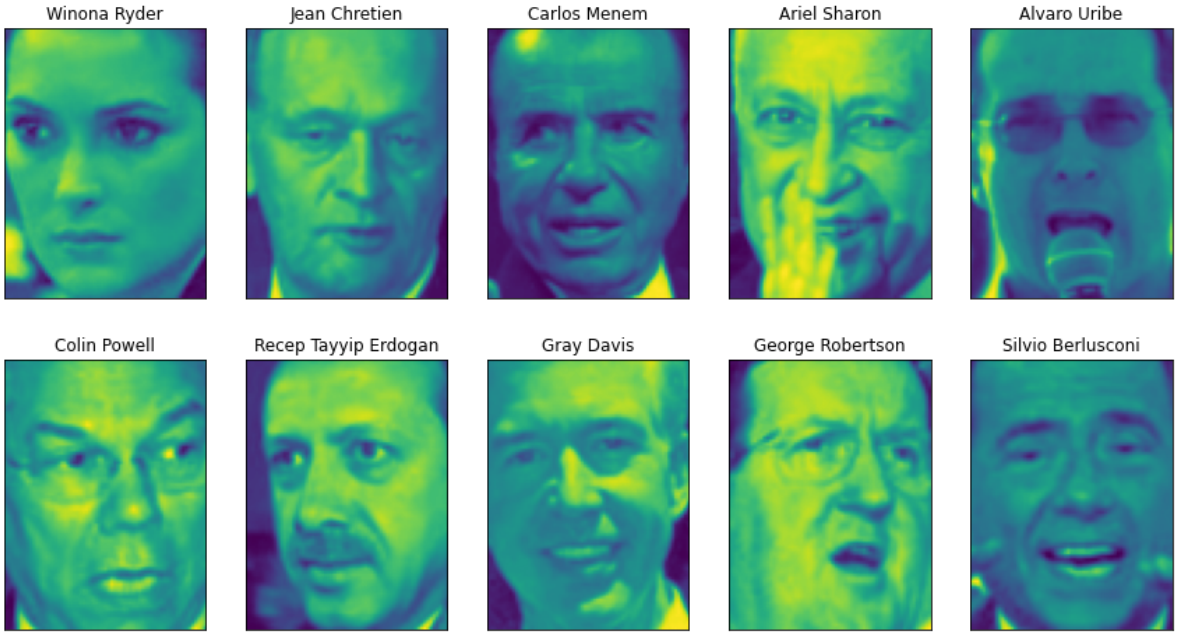
In [3]:

```
# 데이터 가져오기
from sklearn.datasets import fetch_lfw_people
people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
image_shape = people.images[0].shape
print(people.images.shape, image_shape)
```

```
(3023, 87, 65) (87, 65)
```

In [4]:

```
# 이미지 확인
fig, axes = plt.subplots(2, 5, figsize=(15, 8),
                         subplot_kw={'xticks': (), 'yticks': ()})
for target, image, ax in zip(people.target, people.images, axes.ravel()):
    ax.imshow(image)
    ax.set_title(people.target_names[target])
```



In [5]:

```
# 하나의 이미지에 편중되지 않도록, 최대 50장까지만 가져오기
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
    mask[np.where(people.target == target)[0][:50]] = 1

X_people = people.data[mask]
y_people = people.target[mask]

# 0~255 사이의 흑백 이미지의 픽셀 값을 0~1 사이로 스케일 조정합니다.
# (옮긴이) MinMaxScaler를 적용하는 것과 거의 동일합니다.
X_people = X_people / 255.

# 최대 50장을 뽑는 것을 통해 3023장의 이미지가 2063장의 이미지로 변경.
print(X_people.shape, y_people.shape)
```

```
(2063, 5655) (2063,)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Deprec
ationWarning: `np.bool` is a deprecated alias for the builtin `bool`.
To silence this warning, use `bool` by itself. Doing this will not mod
ify any behavior and is safe. If you specifically wanted the numpy sca
lar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://nump
y.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.or
g/devdocs/release/1.20.0-notes.html#deprecations)
```

02. PCA, DBSCAN 활용한 데이터 분석

목차로 이동하기

PCA 주성분 분석

In [6]:

```
# LFW 데이터에서 고유얼굴을 찾은 다음 데이터를 변환합니다
# 5655 이미지 특징 중에 100개의 주성분을 추출
from sklearn.decomposition import PCA
pca = PCA(n_components=100, whiten=True, random_state=0)
pca.fit_transform(X_people)
X_pca = pca.transform(X_people)
print(X_pca.shape, X_pca[0:2])
```

```
(2063, 100) [[-0.5776009   1.1019667  -0.2527034   0.3961173  -0.26037
958  0.42363334
   0.6594225   2.6802251   0.8431763   2.123537   -0.8691332  -0.76645
565
  -1.4438914   1.1299523   1.0088823   0.23699626 -0.8747193  -0.85362
977
   0.4929373  -1.0919951   0.31964263 -0.9997152  -0.9129653  -0.51845
94
  -1.878265    0.18399611 -0.57764924  3.2747946  -1.0267903  -0.18476
966
  -0.01347931 -1.9123693  -0.14336799  0.84771055  1.0098797  -0.93712
25
  -0.58914846 -0.0296761  -2.0612726  -0.78305495  0.51441     0.69881
  -0.09484298 -0.11707406  1.0904773   2.4722183   0.28585142  1.21562
72
   0.32028303 -0.21233189  0.8192284   0.02998489  1.7521871  -0.01803
724
  -0.37436813 -0.05500055 -0.05015875 -0.3640683   1.4550769  -0.21518
269
   0.7045642  -2.359124   -0.05915413  1.1087399  -1.6805235   1.50782
8
  -1.1168935  -1.3016984   0.07638115 -1.4920644   1.6042349   2.15480
4
   0.5785956  -0.17820854  1.5202429   0.6555185  -1.3855449  -0.70783
54
   0.8620969   1.9189268  -1.3289963   1.0716947  -0.28919673  0.44240
066
   1.7564868  -0.9406258  -0.5737608  -0.815196   -1.3026611  -0.93181
95
  -1.4483687  -0.09871251  0.61983395 -0.63637686  1.3104647   0.26345
503
  -1.7959632   0.17702867  0.47081783 -0.5796015 ]
 [-0.14939314  1.2696043  -0.03931826 -1.8890839   0.7706745   1.18752
28
  -0.95361626 -0.84368     1.806659   -0.32330284  1.3630071   0.37037
2
  -0.28121585 -0.87192124 -0.12844986 -0.9498648  -0.32421157 -0.65231
91
   0.9918745   0.42398694 -1.357532    0.92427224 -0.02034467 -0.81774
01
  -0.72337437  0.22131155  0.0954654   0.30270526 -0.03058637  0.25245
377
   0.13703908 -1.3022842   0.05439928  0.36119235 -0.6233561  -1.68822
  -1.3495908   0.14511314 -0.53721184  0.15309976  0.5637055  -1.66490
73
  -1.1472461  -0.464753   -0.13295485  0.20098528 -0.3405233   0.84197
89
   1.2610011   1.3328848  -0.08671156 -0.8339527  -0.9912669  -1.08075
79
   0.7191906  -1.79932     0.5317492   0.6210324  -1.2947179   0.11837
532
```

```
  -0.1057082  -0.9525517  -0.2293261  -1.2758894   2.5233665   1.70721
69
   0.27160695 -0.14020415  0.68996054 -0.29820892 -0.5722398  -1.29681
29
  -0.05541072 -0.18259186 -0.07929219  0.2691147   0.77599     0.46130
103
  -0.7516428  -1.0681628  -0.9023423   0.52873844  0.9465816  -1.28848
39
  -0.30755424  0.19042103  0.47268778  0.07442597  0.80200046  1.03902
11
  -0.2935476   0.534312    0.6342999   1.9349052  -0.9924627   0.03021
515
   1.1026143   0.7542718   0.7045167  -0.5661256 ]]
```

DBSCAN

In [7]:

```
# 추출된 100개의 특징을 DBSCAN을 적용하여 데이터 분류
# 기본 매개변수로 DBSCAN을 적용합니다
dbscan = DBSCAN()
labels = dbscan.fit_predict(X_pca)
print("고유한 레이블: {}".format(np.unique(labels)))
```

```
고유한 레이블: [-1]
```

- 레이블이 -1이므로 모든 데이터가 DBSCAN에 의해 잡음 포인트로 레이블됨.
  - 해결 : eps값을 크게 하거나, min_samples값을 낮추기

In [8]:

```
dbscan = DBSCAN(min_samples=3)
labels1 = dbscan.fit_predict(X_pca)
print("고유한 레이블: {}".format(np.unique(labels1)))
```

```
고유한 레이블: [-1]
```

- eps 값을 키워보기
  - eps 값을 15로 키운 후, 그룹이 2개 생겼다.

In [10]:

```
dbscan = DBSCAN(min_samples=3, eps=15)
labels2 = dbscan.fit_predict(X_pca)
print("고유한 레이블: {}".format(np.unique(labels2)))
```

```
고유한 레이블: [-1  0]
```

- eps와 min_samples의 값에 의해 그룹이 더 많이 생길 수 있다.

In [11]:

```
dbscan = DBSCAN(min_samples=2, eps=15)
labels3 = dbscan.fit_predict(X_pca)
print("고유한 레이블: {}".format(np.unique(labels3)))
```

```
고유한 레이블: [-1  0  1]
```

잡음 포인트와 클러스터 포인트 확인하기

In [12]:

```
# bincount는 음수를 받을 수 없어서 labels3에 1을 더했습니다.
# 고유한 레이블: [-1  0  1]
# 반환값의 첫 번째 원소는 잡음 포인트의 수입니다.
print("클러스터별 포인트 수: {}".format(np.bincount(labels1 + 1)))
print("클러스터별 포인트 수: {}".format(np.bincount(labels2 + 1)))
print("클러스터별 포인트 수: {}".format(np.bincount(labels3 + 1)))
```

```
클러스터별 포인트 수: [2063]
클러스터별 포인트 수: [  32 2031]
클러스터별 포인트 수: [  30 2031    2]
```

03. 모델을 활용한 이상치 검출

목차로 이동하기

DBSCAN의 알고리즘이 잡음이라 말한 이미지는 확인해 보기

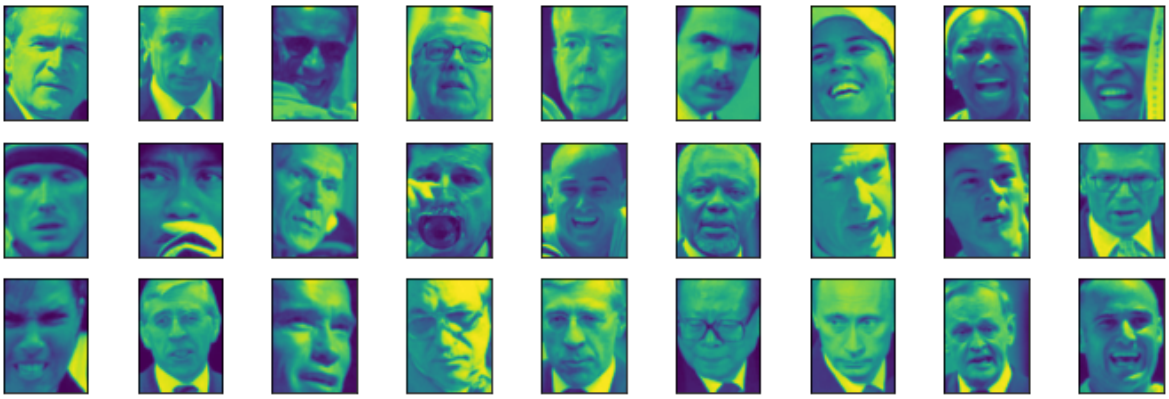
- label1의 -1의 값은 32개, label3의 -1의 값은 30개나 있음.

In [13]:

```
noise = X_people[labels2==-1]  # min_samples=3, eps=15

fig, axes = plt.subplots(3, 9,
                         subplot_kw={'xticks': (), 'yticks': ()},
                         figsize=(12, 4))

for image, ax in zip(noise, axes.ravel()):
    ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
```

In [14]:

```
noise = X_people[labels3==-1]  # min_samples=2, eps=15

fig, axes = plt.subplots(3, 9,
                         subplot_kw={'xticks': (), 'yticks': ()},
                         figsize=(12, 4))

for image, ax in zip(noise, axes.ravel()):
    ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
```



- 이미지 얼굴 앞에 안경, 손 등의 다른 물건이 있음.
- 모자와 얼굴 각도가 조금 다름.

이와 같이 특이한 것을 찾아내는 것을 이상치 검출(outlier detection)이라 한다.

- 실제 애플리케이션이라면 이미지 여백을 잘라 일정한 데이터 셋을 만드는 것이 좋음.
- 큰 클러스터 하나보다 더 많은 클러스터를 찾으려면 eps를 0.5(기본값) ~ 15 사이 정도로 줄여야 함.

In [15]:

```
dbscan = DBSCAN(min_samples=2, eps=10)
labels4 = dbscan.fit_predict(X_pca)
print("고유한 레이블: {}".format(np.unique(labels4)))
```

```
고유한 레이블: [-1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 1
8 19 20 21 22
 23]
```

eps에 따른 클러스터 수의 크기와 크기

In [16]:

```
for eps in [1,3,5,7,9,11,13]:
    print("\n eps = ", eps)
    dbscan = DBSCAN(eps =  eps, min_samples=3)
    labels = dbscan.fit_predict(X_pca)
    print("클러스터 수 : ", len(np.unique(labels)))
    print("클러스터 크기 : ", np.bincount(labels + 1))
```

```
 eps =  1
클러스터 수 :  1
클러스터 크기 :  [2063]

 eps =  3
클러스터 수 :  1
클러스터 크기 :  [2063]

 eps =  5
클러스터 수 :  1
클러스터 크기 :  [2063]

 eps =  7
클러스터 수 :  14
클러스터 크기 :  [2004    3   14    7    4    3    3    4    4    3    3
5    3    3]

 eps =  9
클러스터 수 :  4
클러스터 크기 :  [1307  750    3    3]

 eps =  11
클러스터 수 :  2
클러스터 크기 :  [ 413 1650]

 eps =  13
클러스터 수 :  2
클러스터 크기 :  [ 120 1943]
```

- 큰 클러스터가 하나 있고, 그 이외에는 작은 클러스터
  - 확연히 구별되는 얼굴 이미지가 두 세 개가 아닌 모든 이미지는 거의 동일하게 나머지 이미지들과 비슷(또는 비슷하지 않음)것을 의미

작은 클러스터가 많이 만들어진 eps=7, min_samples=3의 결과 확인

In [17]:

```
X_pca.shape
```

Out[17]:

```
(2063, 100)
```

In [18]:

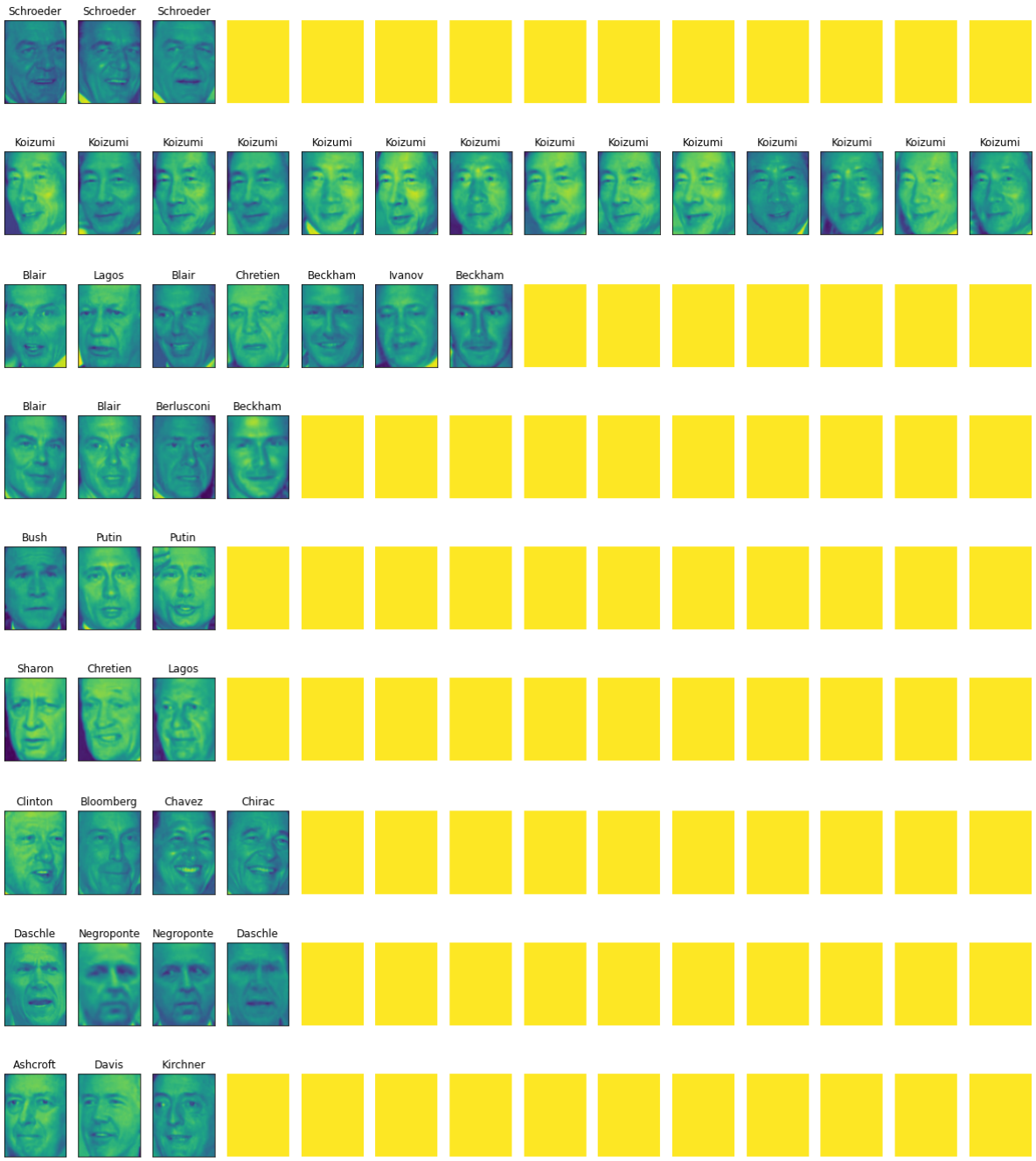
```
dbscan = DBSCAN(eps =  7, min_samples=3)
labels = dbscan.fit_predict(X_pca)
np.unique(labels)
```

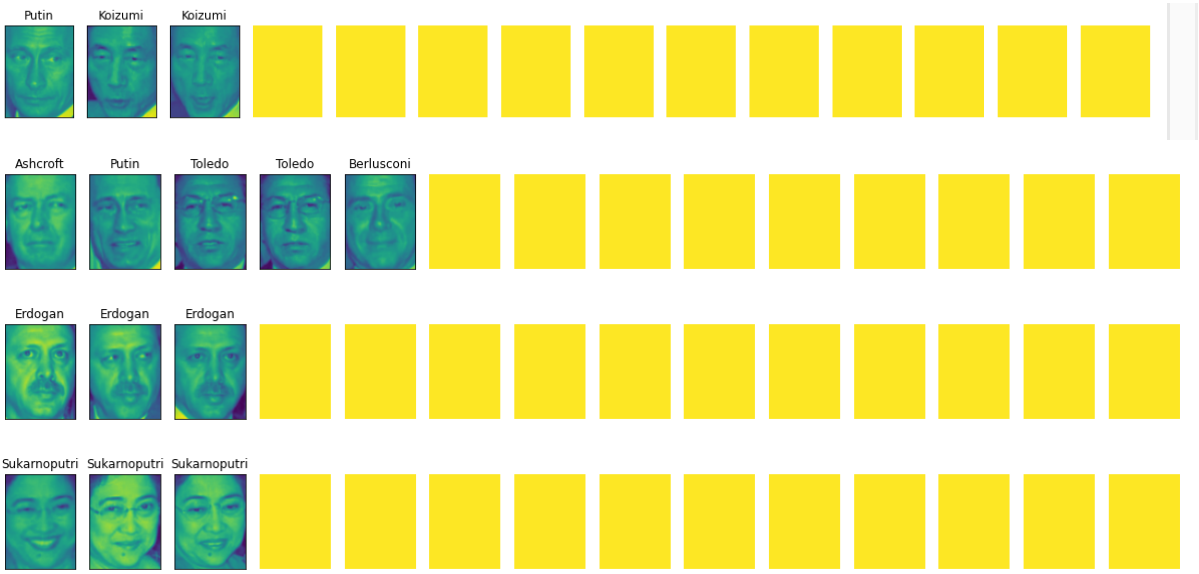
Out[18]:

```
array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [19]:

```
# labels : 만들어진 그룹
# eps =  7, min_samples=3
for cluster in range(max(labels) + 1):
    mask = labels == cluster
    n_images =  np.sum(mask)
    fig, axes = plt.subplots(1, 14, figsize=(14*1.5, 4),
                             subplot_kw={'xticks': (), 'yticks': ()})
    i = 0

    for image, label, ax in zip(X_people[mask], y_people[mask], axes):
        ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
        ax.set_title(people.target_names[label].split()[-1])
        i += 1

    for j in range(len(axes) - i):
        axes[j+i].imshow(np.array([[1]*65]*87), vmin=0, vmax=1)
        axes[j+i].axis('off')
```





04. k-평균으로 얼굴 데이터 분석

목차로 이동하기

- DBSCAN에서는 하나의 큰 클러스터 외에는 만들 수 없음.
- 병합군집과 K-means은 사용자가 지정하여 비슷한 크기의 클러스터를 만들 수 있음.

In [20]:

```
from sklearn.cluster import KMeans
```

In [21]:

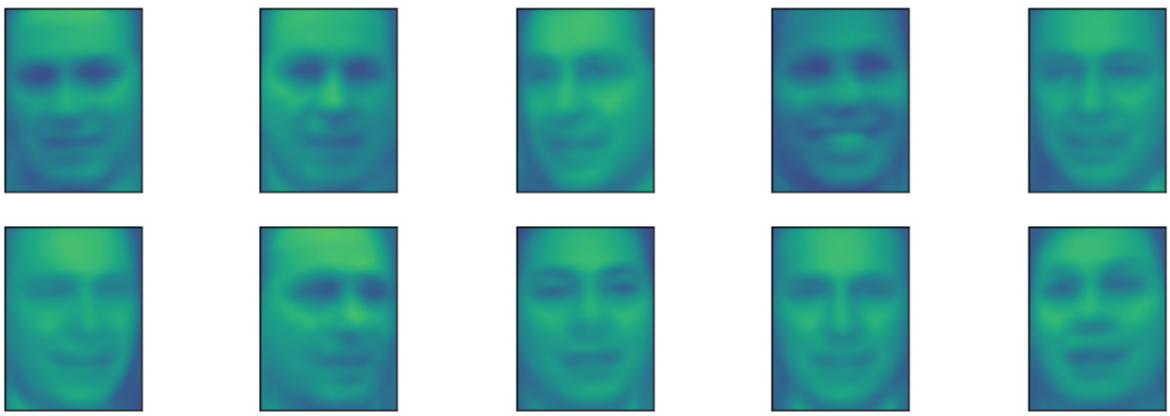
```
# k-평균으로 클러스터를 추출합니다
km = KMeans(n_clusters=10, random_state=0)
labels_km = km.fit_predict(X_pca)
print("k-평균의 클러스터 크기: {}".format(np.bincount(labels_km)))
```

```
k-평균의 클러스터 크기: [155 175 238  75 358 257  91 219 323 172]
```

In [22]:

```
fig, axes = plt.subplots(2, 5,
                         subplot_kw={'xticks': (), 'yticks': ()},
                         figsize=(12, 4))

print(km.cluster_centers_.shape) # 10개 그룹의 중심점.
print(image_shape)

for center, ax in zip(km.cluster_centers_, axes.ravel()):
    print(center.shape, image_shape)
    ax.imshow(pca.inverse_transform(center).reshape(image_shape),
              vmin=0, vmax=1)
```

```
(10, 100)
(87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
(100,) (87, 65)
```



- k-평균이 찾은 클러스터 중심은 매우 부드러운 얼굴 이미지.
- 89개에서 326개까지 얼굴 이미지의 평균이다.
- 차원이 감소된 PCA 성분이 이미지를 더 부드럽게 만든다

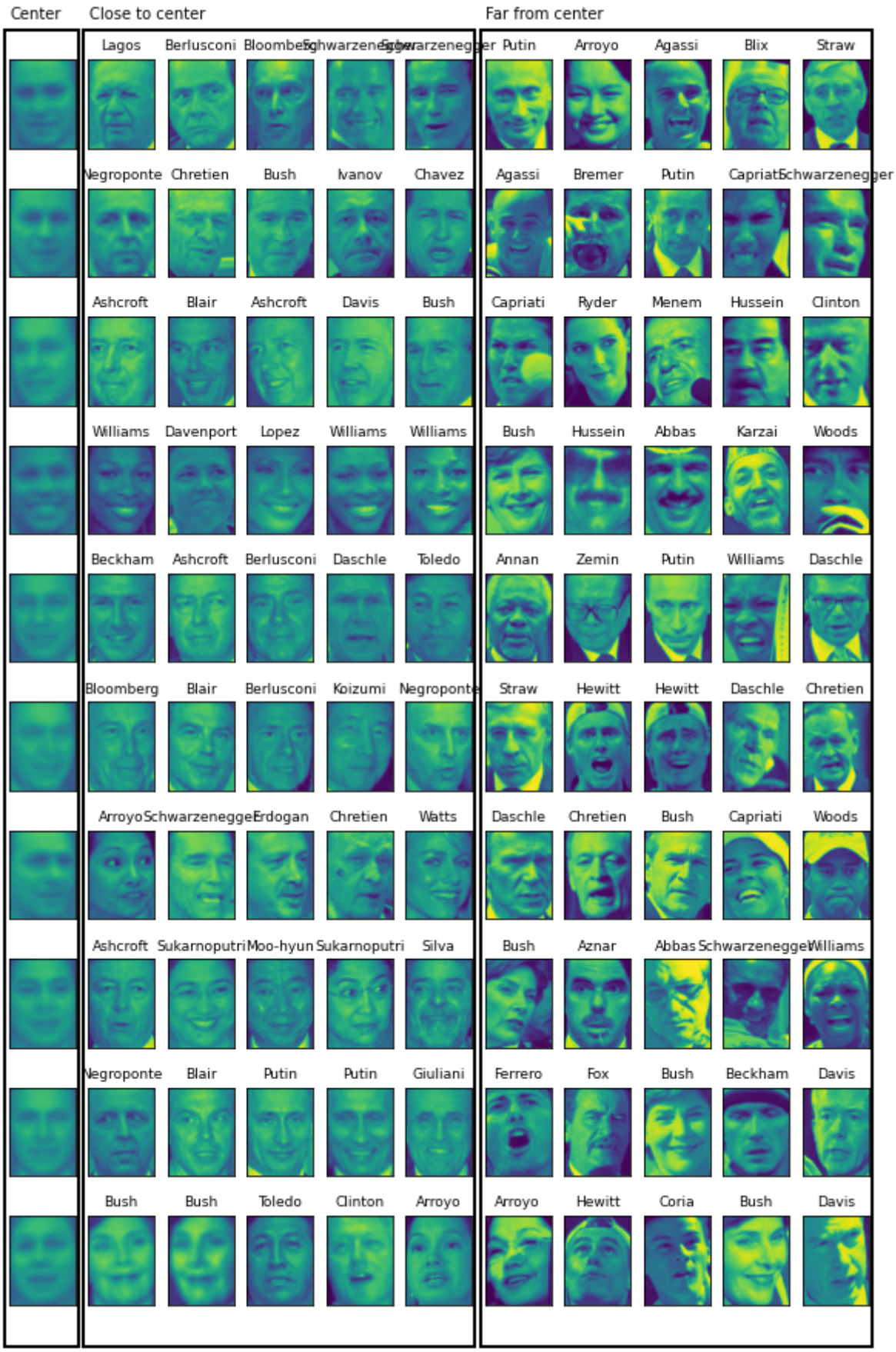
각 클러스터에서 가장 대표 이미지(가장 가까운), 가장 동떨어진 이미지(중심에서 가장 먼 이미지) 다섯개

In [23]:

```
import mglearn
```

In [24]:

```
mglearn.plots.plot_kmeans_faces(km, pca, X_pca, X_people,
                                y_people, people.target_names)
```



- 두번째 클러스터가 웃는 얼굴.
- 나머지 클러스터는 얼굴 각도 중시