빅데이터분석 실습

군집알고리즘(병합군집, DBSCAN, 알고리즘 비교)

데이터 사이언스 전공
담당교수: 곽철완

강의 내용

- 병합 군집
- DBSCAN
- 군집 알고리즘 비교 평가
- 군집 알고리즘 요약

1. 병합 군집

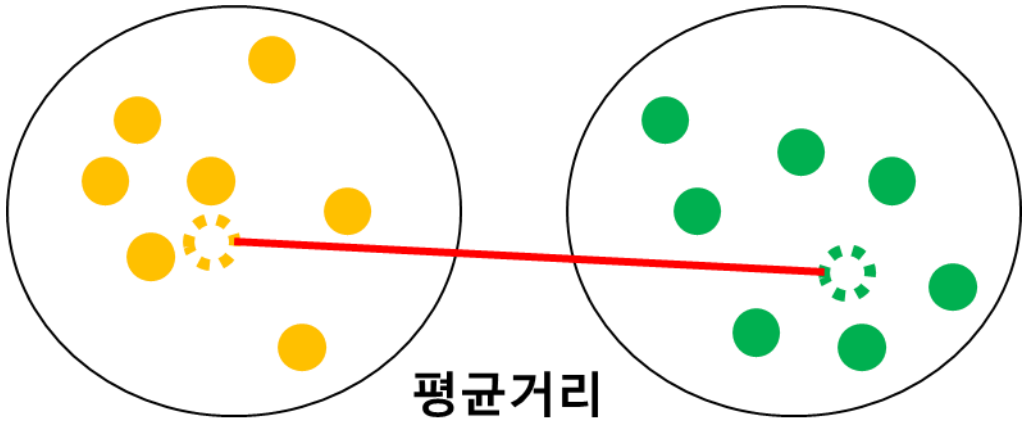
▪ 특징

- agglomerative clustering
- '계층적 군집 분석' 이라고도 한다
- 시작할 때, 각 포인트를 하나의 클러스터로 지정하고,
- 그 다음 어떤 종료 조건을 만족할 때까지 가장 비슷한 두 클러스터를 합쳐 나간다
  - scikit-learn에서 사용하는 종료 조건은 클러스터 개수
  - linkage 옵션에서 가장 비슷한 클러스터를 측정하는 방법 지정

▪ 클러스터 측정 옵션

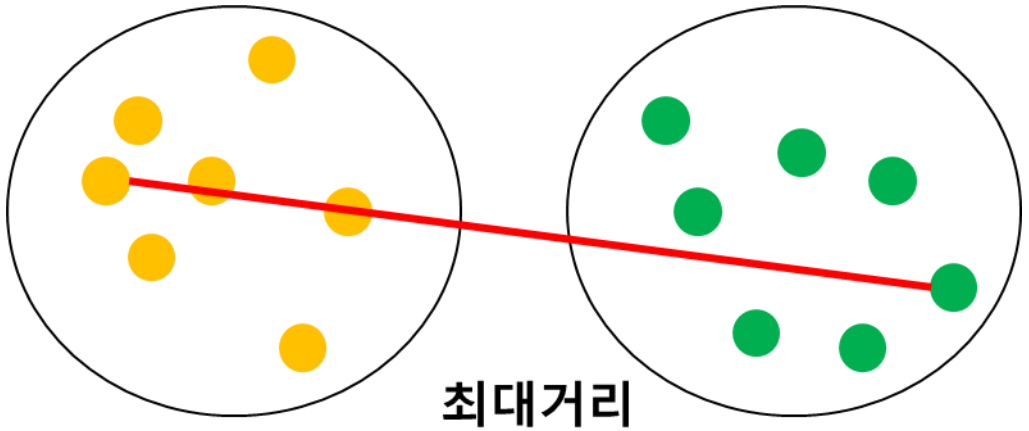
- 비슷한 두 클러스터를 합쳐 나가는 방법

- ward: 기본값으로 모든 클러스터 내의 분산을 가장 작게 증가시키는 두 클러스터를 합친다(비교적 크기가 비슷한 클러스터가 만들어진다)
- average: 클러스터 포인트 사이의 평균 거리가 가장 짧은 두 클러스터를 합친다
- complete: 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 합친다



평균거리
평균 측정법

average



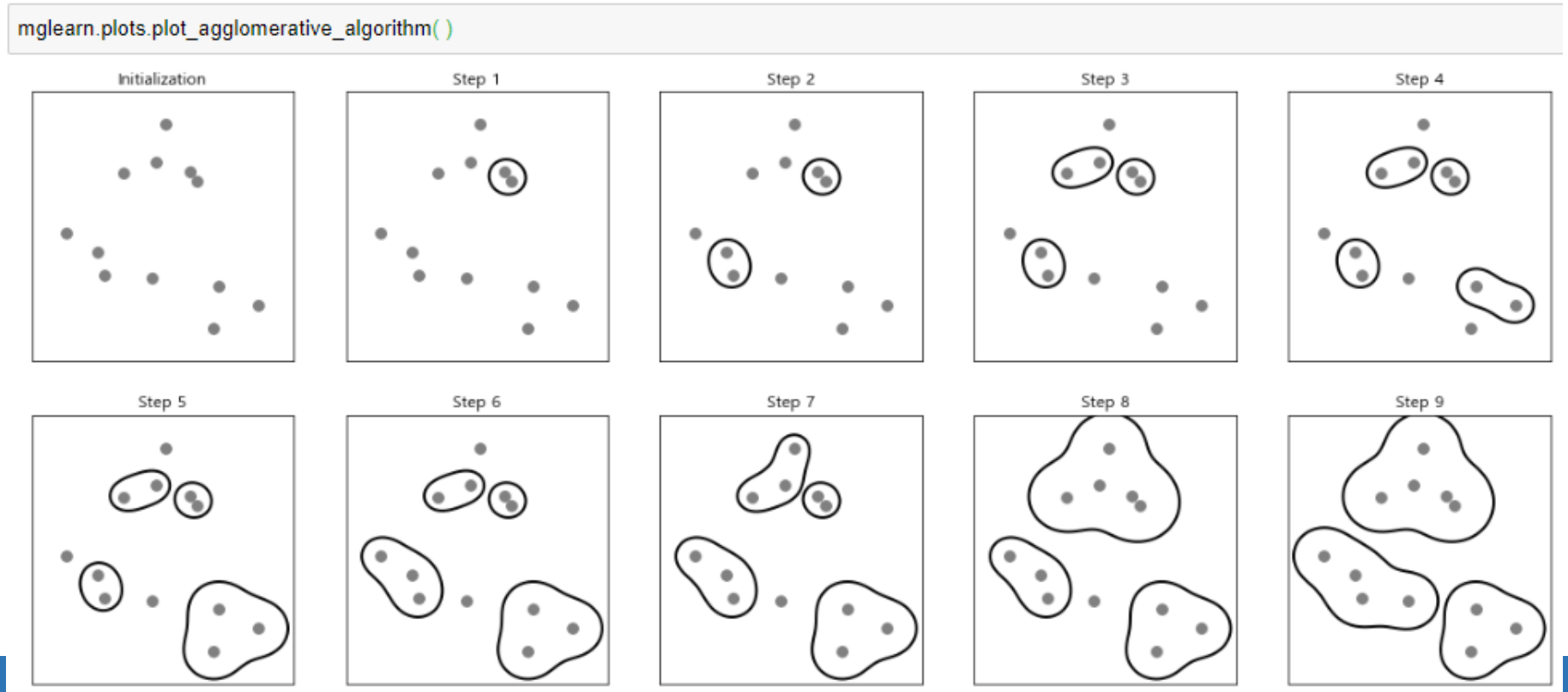
최대거리
완전 측정법

complete

■ 병합 군집 과정

◦ 2차원 데이터 세트에서 3개의 클러스터를 찾기 위한 병합 군집 과정

```
mglearn.plots.plot_agglomerative_algorithm( )
```



병합 군집 적용

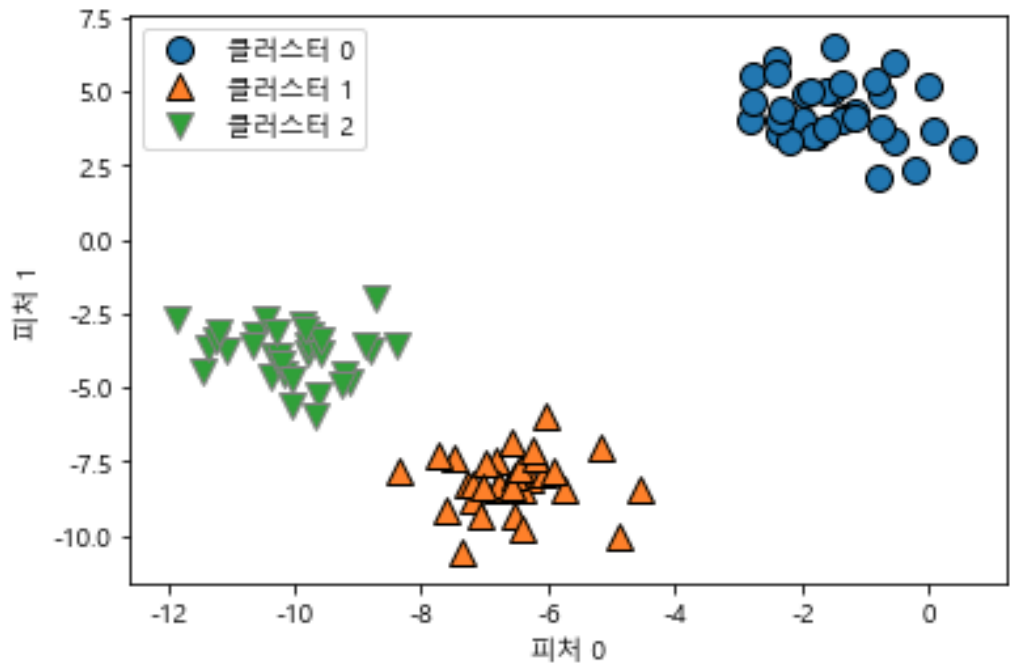
- 3개 클러스터가 있는 간단한 데이터 사용
- 병합 군집은 새로운 데이터에 대해 예측을 할 수 없다(그러므로 predict 메서드가 없다)
- 대신, 학습용 데이터 세트로 모델을 만들고, 클러스터 정보를 얻기 위해 fit_predict 메서드를 사용한다

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=1)
agg = AgglomerativeClustering(n_clusters=3)

assignment = agg.fit_predict(X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
plt.legend(["클러스터 0", "클러스터 1", "클러스터 2"],
loc="best")
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```

- 클러스터 결과는 완벽하게 나왔다
- 클러스터 개수를 지정을 했지만, 군집의 개수를 선택하는데 도움을 주기도 한다

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=1)
agg = AgglomerativeClustering(n_clusters=3)
assignment = agg.fit_predict(X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
plt.legend(["클러스터 0", "클러스터 1", "클러스터 2"], loc="best")
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```

```
Text(0, 0.5, '피처 1')
```



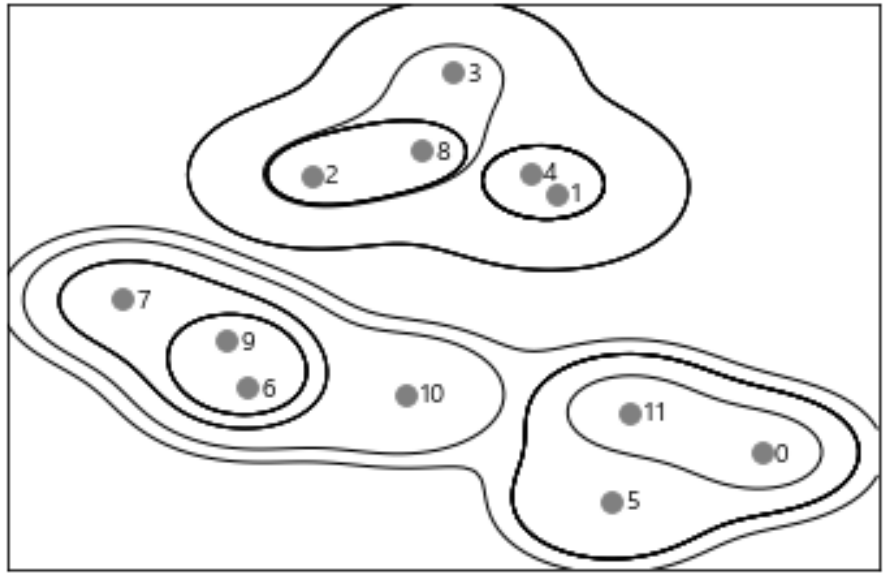
▪ 계층적 군집과 덴드로그램

- 병합 군집은 계층적 군집을 만든다

```
mglearn.plots.plot_agglomerative( )
```

- 오른쪽 그림에서 숫자는 군집이 만들어지는 순서이다
- 이는 2차원 데이터 세트에서 자세한 과정을 보여주지만 피처가 3 이상일 경우는 이 그림을 사용할 수 없다
- 덴드로그램은 다차원 데이터 세트를 처리할 수 있다

mglearn.plots.plot_agglomerative( )



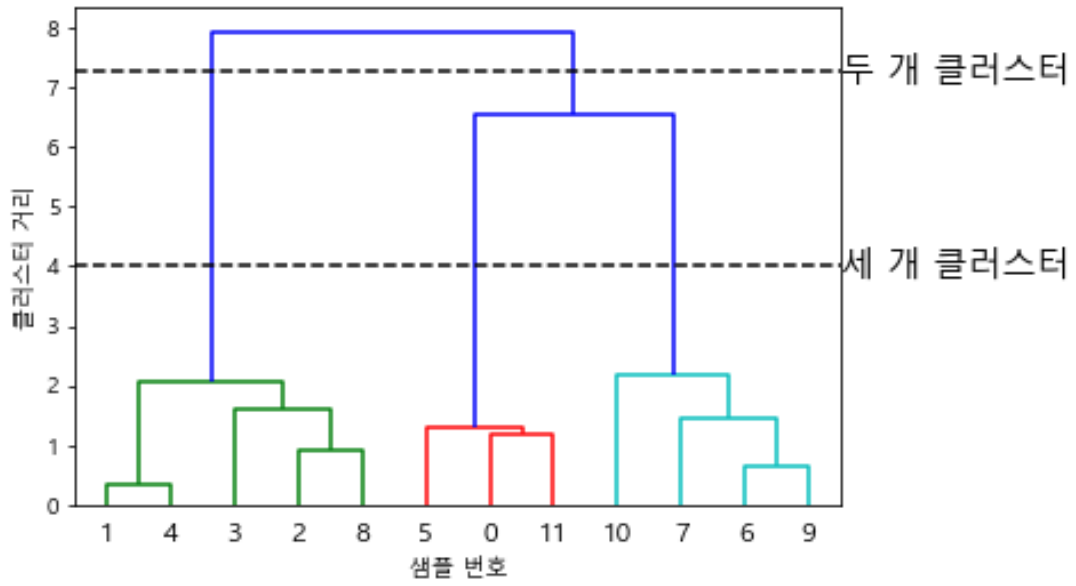
- 덴드로그램

```
from scipy.cluster.hierarchy import dendrogram, ward
X, y = make_blobs(random_state = 0, n_samples = 12)
linkage_array = ward(X)
dendrogram(linkage_array)
ax = plt.gca( )
bounds = ax.get_xbound( )
ax.plot(bounds, [7.25, 7.25], '—', c='k')
ax.plot(bounds, [4, 4], '—', c='k')
ax.text(bounds[1], 7.25, '두 개 클러스터', va = 'center', fontdict={'size': 15}
ax.text(bounds[1], 4, '세 개 클러스터', va = 'center', fontdict={'size': 15}
plt.xlabel("샘플 번호")
plt.ylabel("클러스터 거리")
```

- scypy에서 덴드로그램 함수와 ward 함수를 import 한다
- 샘플이 12개인 개체를 만든다
- 데이터 배열 X 에 ward 함수를 적용한다
- 클러스터 간의 거리 정보가 담긴 linkage_array를 이용해 덴드로그램을 그린다
- 2개와 3개의 클러스터를 구분하는 커트라인을 표시한다

```
from scipy.cluster.hierarchy import dendrogram, ward
X, y = make_blobs(random_state = 0, n_samples = 12)
linkage_array = ward(X)
dendrogram(linkage_array)
ax = plt.gca( )
bounds = ax.get_xbound( )
ax.plot(bounds, [7.25, 7.25], '--', c='k')
ax.plot(bounds, [4, 4], '--', c='k')
ax.text(bounds[1], 7.25, '두 개 클러스터', va = 'center', fontdict={'size': 15})
ax.text(bounds[1], 4, '세 개 클러스터', va = 'center', fontdict={'size': 15})
plt.xlabel("샘플 번호")
plt.ylabel("클러스터 거리")
```

```
Text(0, 0.5, '클러스터 거리')
```



2. DBSCAN

▪ 특징

- Density-based spatial clustering of application with noise
- 장점으로 클러스터 개수를 미리 지정할 필요가 없다
- 복잡한 형상도 찾을 수 있으며, 어떤 클래스에도 속하지 않는 포인트를 구분할 수 있다
- 속도가 병합 군집이나 k-Means 보다 느리지만 큰 데이터 세트에도 적용할 수 있다

▪ 군집 방법

- 피처 공간에서 데이터가 붐비는 지역의 포인트를 찾는다 → 피처 공간의 밀집 지역(dense region)이라 한다
- 데이터의 밀집 지역이 한 클러스터를 구성하며 비교적 비어있는 지역을 경계로 다른 클러스터와 구분한다
- 밀집 지역에 있는 포인트를 핵심 샘플(혹은 핵심 포인트)라 한다
- 매개변수 eps와 min_samples가 있다

- 절차
  - 무작위로 포인트 선택
  - 그 포인트에서 eps 거리 안의 모든 포인트를 찾는다
    - 만약, eps 거리 안에 있는 포인트 수가 min_samples 보다 적다면 그 포인트는 어떤 클래스에도 속하지 않는 잡음으로 레이블한다
  - eps 거리 안에 min_samples보다 많은 포인트가 있다면, 그 포인트는 '핵심 포인트'로 레이블하고 새로운 클러스터 레이블을 할당한다
  - 그 다음 그 포인트의(eps 거리 안의) 모든 이웃 포인트를 조사한다.
    - 만약 새 포인트가 어떤 클러스터에도 아직 할당되지 않았다면 바로 전에 만든 클러스터 레이블을 할당한다
    - 만약 핵심 포인트이면 그 포인트의 이웃을 차례로 방문한다

- 이런 식으로 계속 진행하여 클러스터는 eps 거리 안에 더 이상 '핵심 포인트' 없을 때 까지 진행된다
- 그런 후, 아직 방문하지 못한 포인트를 선택하여 같은 과정을 반복한다

- 포인트는 핵심 포인트, 경계 포인트(핵심 샘플에서 eps 거리 안에 있는 포인트), 잡음 포인트로 구성된다

■ 적용

◦ 병합 군집에 사용했던 데이터 세트 이용한다

```
from sklearn.cluster import DBSCAN
X, y = make_blobs(random_state=0, n_samples=12)

dbscan = DBSCAN( )
clusters = dbscan.fit_predict(X)
print("클러스터 레이블: \n", clusters)
```

◦ 모든 클러스터에 -1레이블 할당

◦ 이는 샘플이 작기 때문

```
from sklearn.cluster import DBSCAN
X, y = make_blobs(random_state=0, n_samples=12)

dbscan = DBSCAN( )
clusters = dbscan.fit_predict(X)
print("클러스터 레이블: \n", clusters)
```

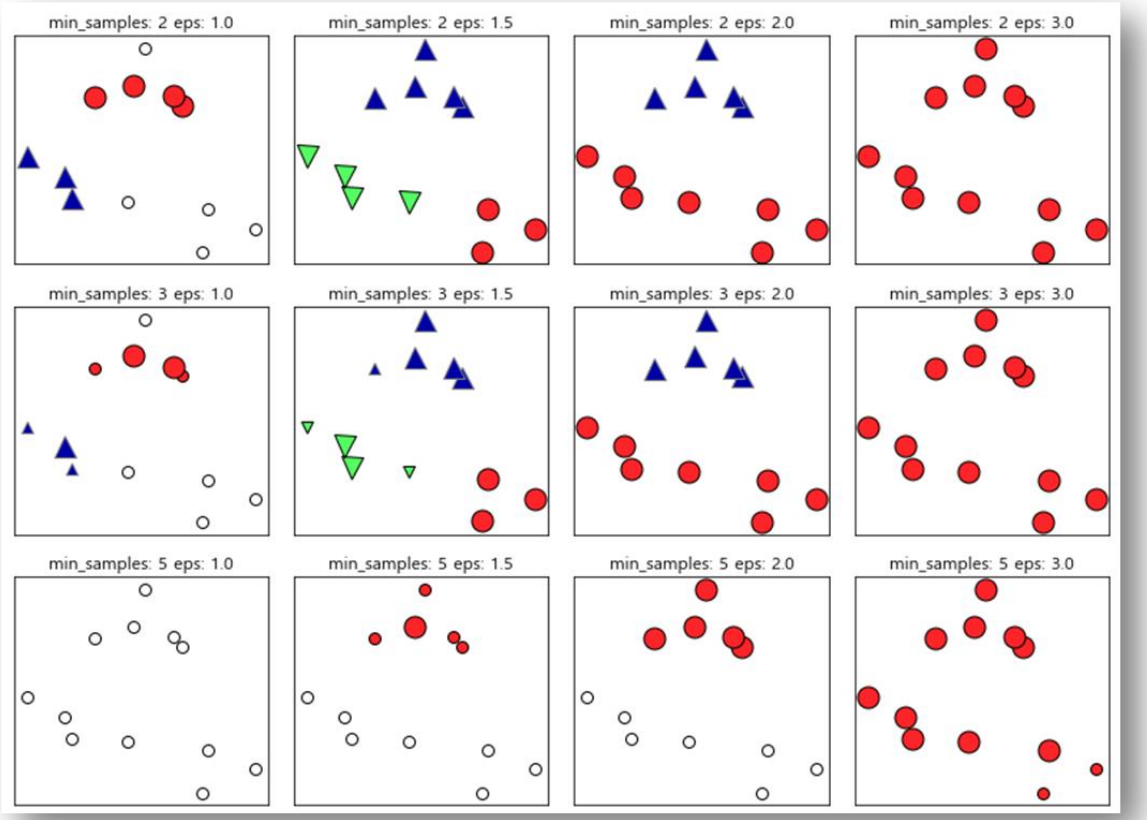
```
클러스터 레이블:
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
```

◦ 여러가지 min_samples와 eps에 대한 클러스터 할당 사례

```
mglearn.plots.plot_dbscan()
```

```
min_samples: 2 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 2 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 2 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 2 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 3 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 3 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 3 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 3 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 5 eps: 1.000000  cluster: [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
min_samples: 5 eps: 1.500000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 2.000000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
```

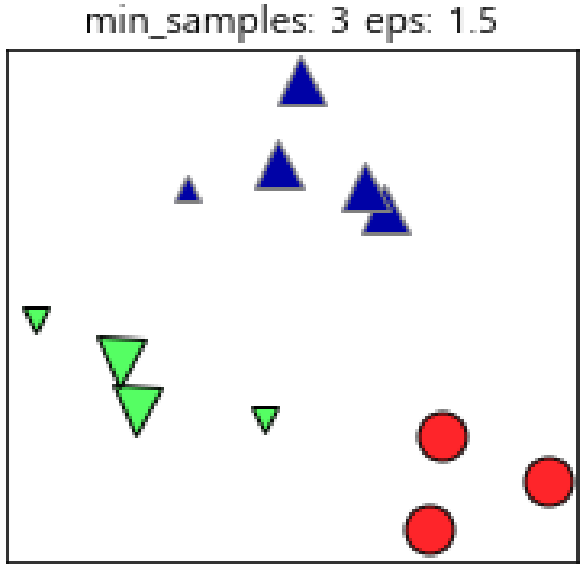
- 오른쪽 그림에서 클러스터에 속한 포인트는 색깔이 있다
- 핵심 포인트는 크게 표시
- 경계 포인트는 작게 표시
- esp가 증가되면 하나의 클러스트에 더 많은 포인트가 모인다
- min_samples가 커지면 핵심 포인트가 줄며, 잡음 포인트가 증가된다

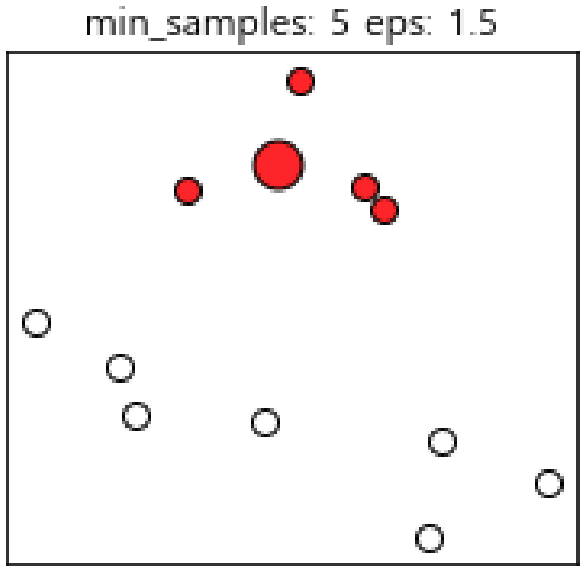


- eps 매개변수
  - 가까운 포인트의 범위를 결정하기 때문에 중요하다
  - eps를 매우 작게 하면 어떤 포인트도 핵심 포인트가 되지 못하고, 모든 포인트가 잡음 포인트가 될 수 있다
  - eps를 매우 크게 하면, 모든 포인트가 하나의 클러스터에 속하게 될 수 있다
- min_samples 매개변수
  - 덜 조밀한 지역의 포인트들이 잡음 포인트가 될지 아니면 하나의 클러스터가 될지 결정하는 역할을 담당한다
  - min_samples가 늘어나면 min_samples의 수보다 작은 클러스터들은 잡음 포인트가 된다 → 클러스터의 최소 크기 결정

- 사례: eps가 1.5일때, min_samples가 3인 경우와 5인 경우 비교

- eps의 값은 간접적으로 몇 개의 클러스터가 만들어질지 제어한다
- 적절한 eps 값을 찾으려면 StandardScaler나 MinMaxScaler로 모든 피처의 스케일을 비슷한 범위로 조정하는 것이 좋다





3. 군집 알고리즘의 비교와 평가

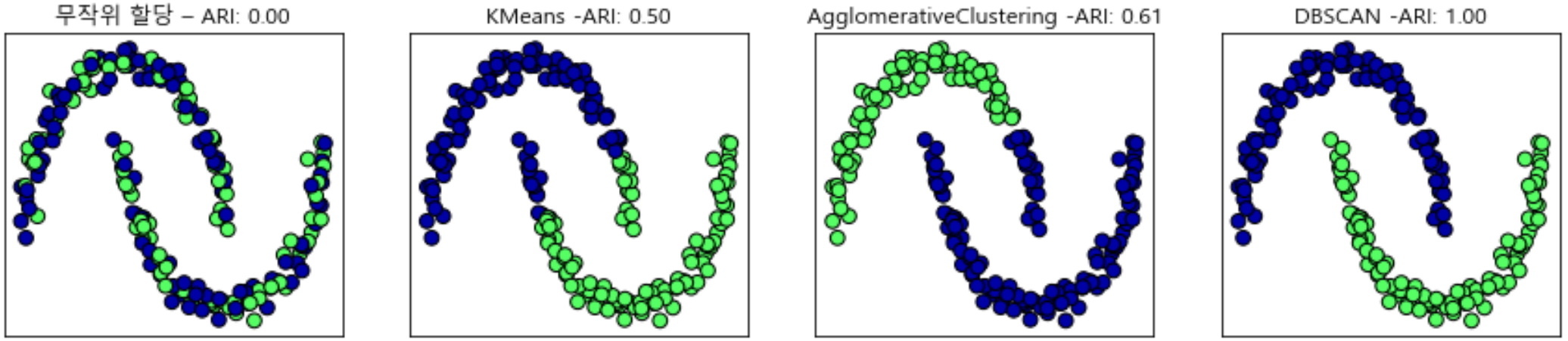
▪ 타깃값으로 군집 평가하기

- 평가지표
  - 군집 알고리즘의 결과를 실제 정답 클러스터와 비교하여 평가할 수 있다
  - ARI adjusted rand index
  - NMI normalized mutual information
    - 가장 널리 사용되는 지표이다
    - 분류가 최적일 때 1, 무작위로 분류될 때 0으로 나타나는 지표
- ARI를 이용하여 k-Means, 병합군집, DBSCAN 알고리즘 비교

```
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.datasets import make_moons
X, y = make_moons(n_samples = 200, noise = 0.05, random_state=0 )
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler( )
scaler.fit(X)
X_scaled = scaler.transform(X)

fig, axes = plt.subplots(1, 4, figsize=(15, 3), subplot_kw={'xticks': (), 'yticks': ()})

from sklearn.cluster import KMeans
algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2),
              DBSCAN( )]
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))
```

```
axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters,
                cmap=mglearn.cm3, s=60, edgecolors='black')
axes[0].set_title("무작위 할당 – ARI: {:.2f}".format(
                  adjusted_rand_score(y, random_clusters)))

for ax, algorithm in zip(axes[1: ], algorithms):
        clusters = algorithm.fit_predict(X_scaled)
        ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters,
                cmap=mglearn.cm3, s=60, edgecolors='black')
        ax.set_title("{ } – ARI: {:.2f}".format(algorithm.__class__.__name__,
                adjusted_rand_score(y, clusters)))
```

```
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.datasets import make_moons
X, y = make_moons(n_samples = 200, noise = 0.05, random_state=0 )

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler( )
scaler.fit(X)
X_scaled = scaler.transform(X)

fig, axes = plt.subplots(1, 4, figsize=(15, 3), subplot_kw={'xticks': (), 'yticks': ()})

from sklearn.cluster import KMeans

algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2), DBSCAN( )]
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))

axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters, cmap=mglearn.cm3, s=60, edgecolor='black')
axes[0].set_title("무작위 할당 – ARI: {:.2f}".format(adjusted_rand_score(y, random_clusters)))

for ax, algorithm in zip(axes[1: ], algorithms):
    clusters = algorithm.fit_predict(X_scaled)
    ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mglearn.cm3, s=60, edgecolor='black')
    ax.set_title("{} -ARI: {:.2f}".format(algorithm.__class__.__name__, adjusted_rand_score(y, clusters)))
```



- 클러스터를 무작위로 할당했을 때, ARI 점수가 0 이고, DBSCAN은 ARI점수가 1 이다 → 완벽하게 군집을 만들었으므로 1 이다
- 군집 모델 평가에서 adjusted_rand_score 혹은 normalized_mutual_infor_score 같은 군집용 도구를 사용해야 한다
  - accuracy_score 를 사용하면 오류가 발생한다

```
from sklearn.metrics import accuracy_score

clusters1 = [0, 0, 1, 1, 0]
clusters2 = [1, 1, 0, 0, 1]

print("정확도: {:.2f}".format(accuracy_score(clusters1, clusters2)))

print("ARI: {:.2f}".format(adjusted_rand_score(clusters1, clusters2)))
```

```
정확도: 0.00
ARI: 1.00
```

- 위의 결과를 보면 2개의 클러스터가 서로 다르다
- 정확도는 레이블로 판단하기 때문에 서로 달라서 0 이며
- ARI는 같은 포인트가 클러스터에 모여있으므로 1이다

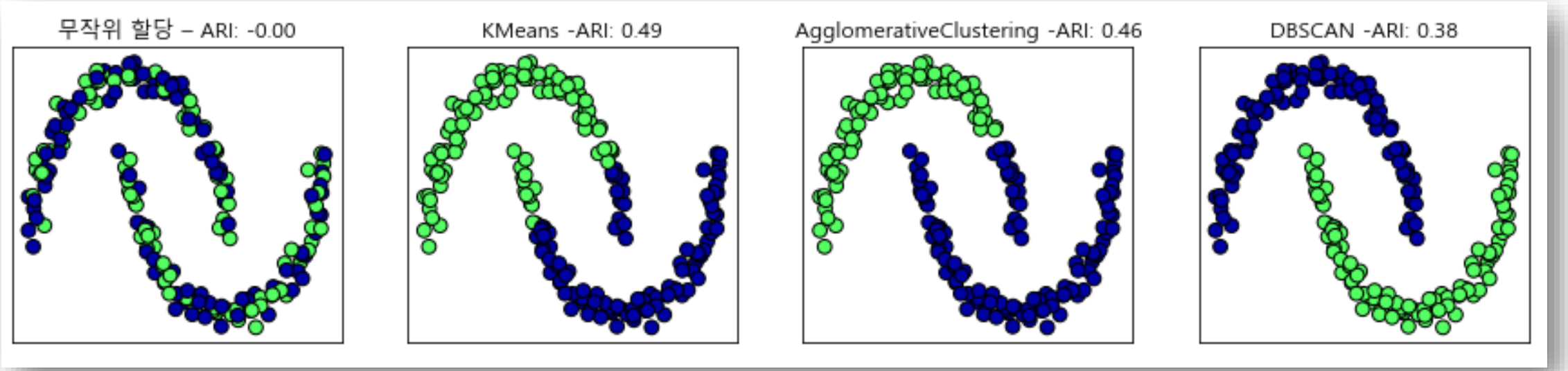
타깃값 없이 군집 평가하기

- 일반적으로 군집은 그 결과값을 비교할 타깃값이 없어, ARI 방법을 사용하기 어렵다
  - 그러므로 ARI나 NMI는 알고리즘 성능 평가보다, 알고리즘 개발에 사용된다
- 타깃값이 필요 없는 군집용 지표로 실루엣 계수 silhouette coefficient가 있다
  - 실제로 잘 동작하지 않는다
  - 클러스터 밀집 정도를 계산하는 것으로 최대 점수가 1이며 1에 가까울 수록 좋다
  - 모양이 복잡할 때는 잘 들어맞지 않는다

```
from sklearn.metrics.cluster import silhouette_score

X, y = make_moons(n_samples = 200, noise = 0.05, random_state=0 )

scaler = StandardScaler( )
scaler.fit(X)
X_scaled = scaler.transform(X)

fig, axes = plt.subplots(1, 4, figsize=(15, 3), subplot_kw={'xticks': (), 'yticks': ()})

algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2),
DBSCAN( )]

random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))
```

```
axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters,
                cmap=mglearn.cm3, s=60, edgecolors='black')
axes[0].set_title("무작위 할당 – ARI: {:.2f}".format(
                 silhouette_score(X_scaled, random_clusters)))


for ax, algorithm in zip(axes[1: ], algorithms):
        clusters = algorithm.fit_predict(X_scaled)
        ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters,
                cmap=mglearn.cm3, s=60, edgecolors='black')
        ax.set_title("{ } – ARI: {:.2f}".format(algorithm.__class__.__name__,
                 silhouette_score(X_scaled, clusters)))
```

```
from sklearn.metrics.cluster import silhouette_score

X, y = make_moons(n_samples = 200, noise = 0.05, random_state=0 )

scaler = StandardScaler( )
scaler.fit(X)
X_scaled = scaler.transform(X)

fig, axes = plt.subplots(1, 4, figsize=(15, 3), subplot_kw={'xticks': (), 'yticks': ()})

algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2), DBSCAN( )]

random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))

axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters, cmap=mglearn.cm3, s=60, edgecolor='black')
axes[0].set_title("무작위 할당 – ARI: {:.2f}".format(silhouette_score(X_scaled, random_clusters)))

for ax, algorithm in zip(axes[1: ], algorithms):
    clusters = algorithm.fit_predict(X_scaled)
    ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mglearn.cm3, s=60, edgecolor='black')
    ax.set_title("{} -ARI: {:.2f}".format(algorithm.__class__.__name__, silhouette_score(X_scaled, clusters)))
```



- 위의 결과를 보면, 그림으로는 DBSCAN 이 더 낫지만, 실루엣 점수는 KMeans가 더 높다
- 이러한 문제점 해결이 앞으로 요구된다

■ 얼굴 데이터 세트로 군집 알고리즘 비교

- LFW 데이터 세트를 이용하여 k-Means, DBSCAN, 병합 군집 알고리즘 적용 비교한다
- PCA로 생성한 100개의 주성분을 입력 데이터로 사용한다

```
from sklearn.decomposition import PCA
pca = PCA(n_components=100, whiten=TRUE, random_state=0)
X_pca = pca.fit_transform(X_people)
```

```
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

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])

    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]

X_people = X_people / 225.
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=100, whiten=True, random_state=0)

X_pca = pca.fit_transform(X_people)
```

- DBSCAN으로 얼굴 데이터 세트 분석하기

```
dbscan = DBSCAN( )
labels = dbscan.fit_predict(X_pca)
print("고유한 레이블: ", np.unique(labels))
```

```
dbscan = DBSCAN( )
labels = dbscan.fit_predict(X_pca)
print("고유한 레이블: ", np.unique(labels))
```

```
고유한 레이블:  [-1]
```

  - 레이블이 -1 뿐이므로 모든 데이터가 DBSCAN에 의해 잡음 포인트로 레이블 되었다 → eps, min_samples 값 변경 필요

- min_sample 값 변경

```
dbscan = DBSCAN(min_samples=3 )
labels = dbscan.fit_predict(X_pca)
print("고유한 레이블: ", np.unique(labels))
```

```
고유한 레이블:  [-1]
```

  - min_samples 값을 3으로 변경해도 모두 잡음 포인트이다

- eps 값을 크게 한다
  - 클러스터 포인트
  - 잡음 포인트를 얻었다

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

```
고유한 레이블:  [-1  0]
```

- 이 결과를 잡음 포인트와 클러스터에 속한 포인트가 몇 개인지 확인

```
print("클러스터별 포인트 수: ", np.bincount(labels + 1))
```

```
클러스터별 포인트 수:  [  32 2031]
```

◦ 잡음 포인트를 직접 확인

```
noise = X_people[labels == -1]
fig, axes = plt.subplots(3, 9, figsize=(12, 4), subplot_kw={‘xticks’: (), ‘yticks’: ()})
for image, ax in zip(noise, axes.ravel()):
    ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
```

```
noise = X_people[labels == -1]
fig, axes = plt.subplots(3, 9, figsize=(12, 4), subplot_kw={'xticks': (), 'yticks': ()})
for image, ax in zip(noise, axes.ravel()): ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
```

- 위의 이미지들이 잡음 레이블로 분류된 이유는, 손으로 입을 가리거나 잔을 들고 있거나 모자를 쓰거나 등 앞에서 본 이미지와 달라 잡음으로 처리되었다
- 특이한 것을 찾는 분석을 '이상치 검출 outlier detection'이라 한다

◦ eps 값에 따른 차이 비교

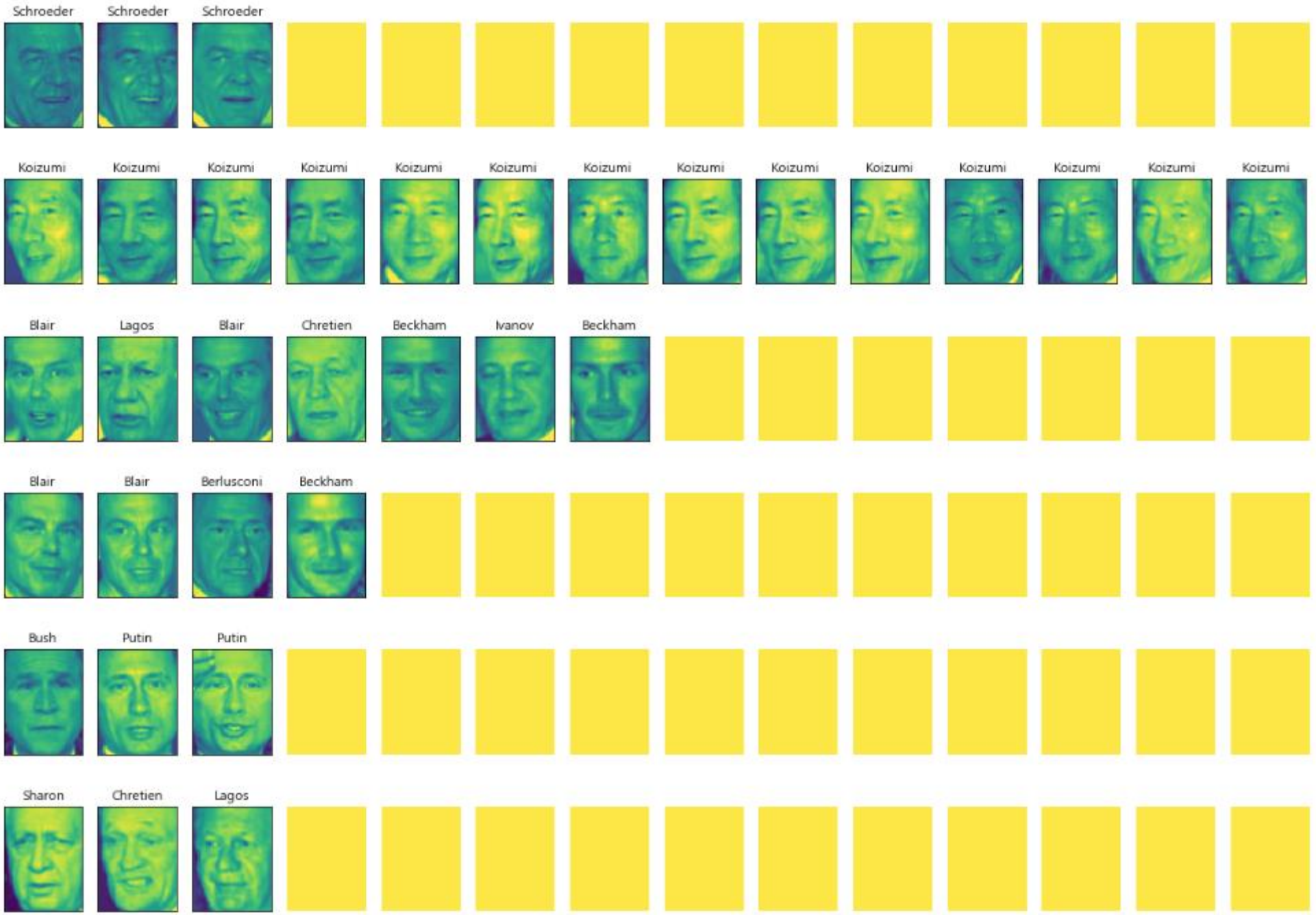
```
for eps in[1, 3, 5, 7, 9, 11, 13]:
      print(“\neps”, eps)
      dbscan = DBSCAN(eps=eps, min_samples=3)
      labels = dbscan.fit_predict(X_pca)
      print(“클러스터 수: “, len(np.unique(labels)))
      print(“클러스터 크기: “, np.bincount(labels + 1))
```

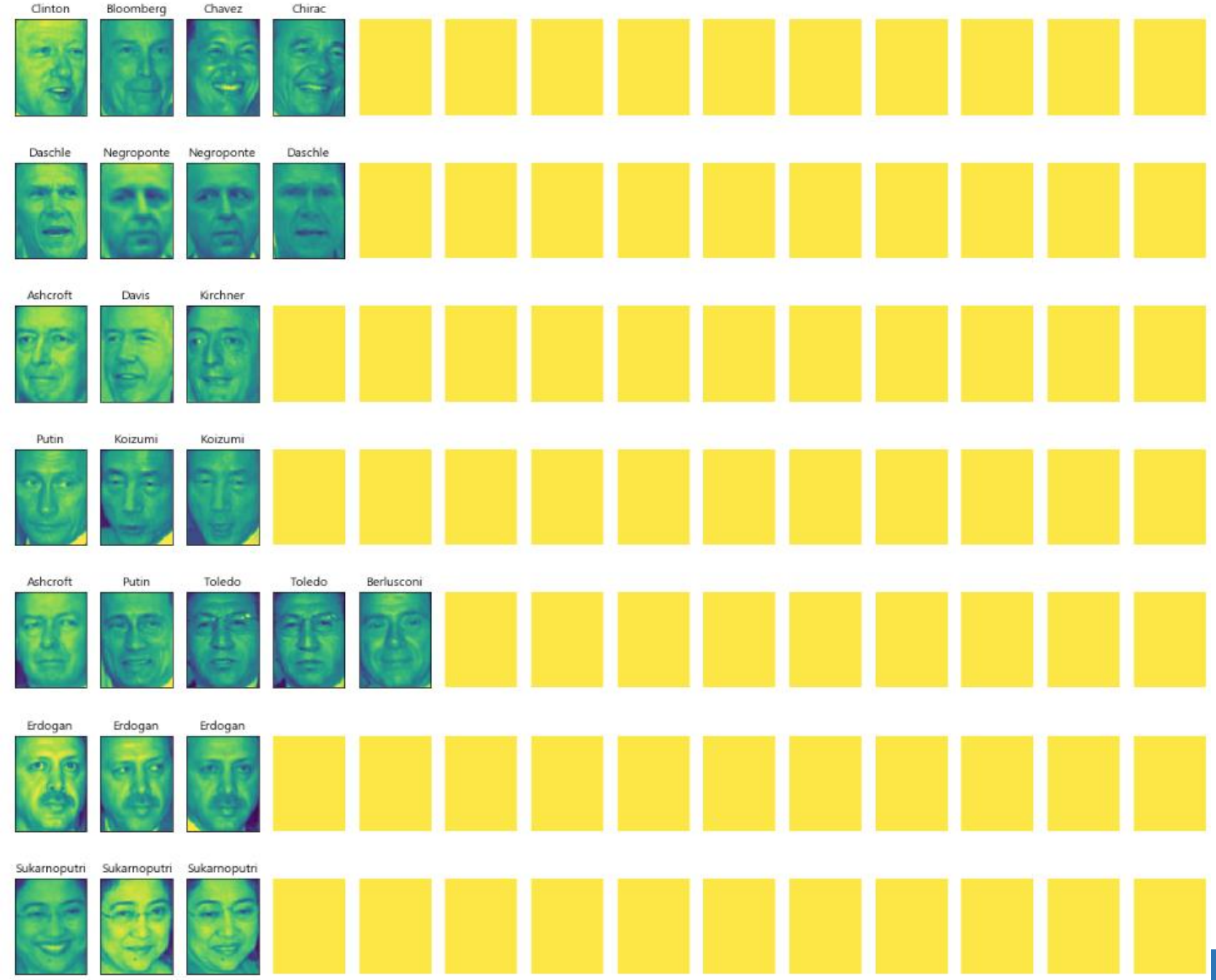
```
for eps in[1, 3, 5, 7, 9, 11, 13]:
    print("\neps", eps)
    dbscan = DBSCAN(eps=eps, min_samples=3)
    labels = dbscan.fit_predict(X_pca)
    print("클러스터 수: ", len(np.unique(labels)))
    print("클러스터 크기: ", np.bincount(labels + 1))
```

- eps가 작으면 모든 포인트가 잡음으로 레이블 된다
- eps=7에서 잡음 포인트가 많지만, 작은 클러스터도 많이 생겼다
- eps=11부터는 큰 클러스터 하나와 잡음 포인트가 생겼다
- 여기서 보면, 모두가 큰 클러스트 하나와 작은 클러스터들이다
- 모든 이미지는 동일하게 다른 이미지와 비슷하다

```
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]
```

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

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')
```


Schroeder Schroeder Schroeder
Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi Koizumi
Blair Lagos Blair Chretien Beckham Ivanov Beckham
Blair Blair Berlusconi Beckham
Bush Putin Putin
Sharon Chretien Lagos


Clinton
Bloomberg
Chavez
Chirac
Daschle
Negroponte
Negroponte
Daschle
Ashcroft
Davis
Kirchner
Putin
Koizumi
Koizumi
Ashcroft
Putin
Toledo
Toledo
Berlusconi
Erdogan
Erdogan
Erdogan
Sukarnoputri
Sukarnoputri
Sukarnoputri

- 13개 클러스터 중 몇 개 클러스터는 매우 뚜렷한 얼굴을 가진 사람들이며, 얼굴 표정과 각도가 거의 동일하다
- 13개 클러스터를 만들었지만 거의 얼굴이 비슷하다

- k-Means를 이용한 얼굴 데이터 세트 분석
  - DBSCAN은 하나의 큰 클러스터 외에는 만들 수 없었다
  - 이에 비해 병합 군집과 k-Means는 비슷한 크기의 클러스터는 만들 수 있지만 클러스터 개수를 지정해야 한다
  - 실제에서는 적용할 수 없지만, 분석을 위해 클러스터를 10개로 줄여서 적용한다

```
km = KMeans(n_clusters=10, random_state=0)
labels_km = km.fit_predict(X_pca)
print("k-Means의 클러스터 크기:", np.bincount(labels_km))
```

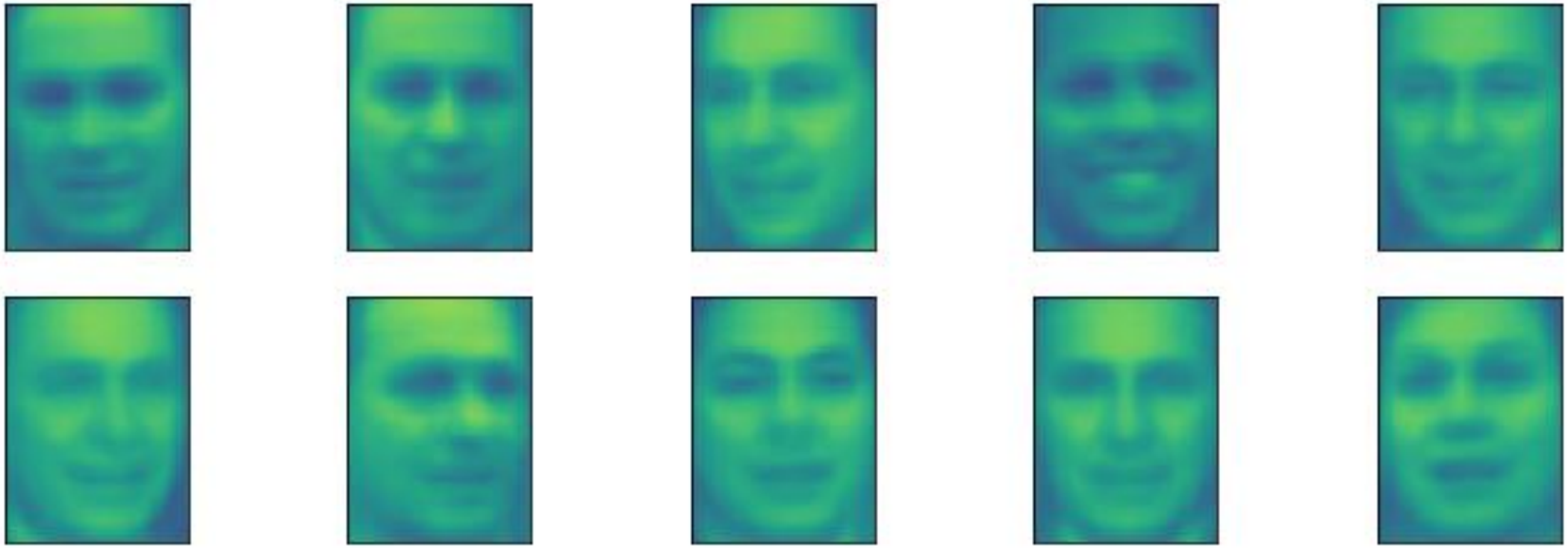
```
km = KMeans(n_clusters=10, random_state=0)
labels_km = km.fit_predict(X_pca)
print("k-Means의 클러스터 크기:", np.bincount(labels_km))
```

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

- k-Means는 크기가 91에서 358까지의 클러스터로 나누었다. 이 결과는 DBSCAN과 차이가 많다
- k-Means 결과를 시각화한다
  - PCA 성분으로 군집 알고리즘을 적용했기 때문에 k-Means의 클러스터 중심을 pca.inverse_transform을 사용해 원본 공간으로 되돌린 후 시각화한다

```
fig, axes = plt.subplots(2, 5, figsize=(12, 4), subplot_kw={‘xticks’: (), ‘yticks’: ()})
for center, ax in zip(km.cluster_centers_, axes.ravel()):
    ax.imshow(pca.inverse_transform(center).reshape(image_shape,
    vmin=0, vmax=1)
```

```
fig, axes = plt.subplots(2, 5, figsize=(12, 4), subplot_kw={'xticks': (), 'yticks': ()})
for center, ax in zip(km.cluster_centers_, axes.ravel()):
    ax.imshow(pca.inverse_transform(center).reshape(image_shape), vmin=0, vmax=1)
```



- 클러스터를 10개로 지정했을 때, k-Means가 찾은 클러스터 중심으로 매우 부드러운 이미지이다. 중심 이미지 91개에서 358개까지 얼굴 이미지의 평균이기때문이다
- 동시에 차원이 감소된 PCA 성분이 이미지를 더 부드럽게 만들었다

◦ 각 클러스터에 가장 대표되는 이미지(클러스터에 할당된 이미지 중 중심에서 가장 가까운 이미지) 5개와 가장 동떨어진 이미지(클러스터에 할당된 이미지 중 중심에서 가장 먼 이미지) 5개

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

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

- 오른쪽 그림의 왼쪽은 중심, 가운데는 중심에서 가까운 이미지, 오른쪽은 중심에서 먼 이미지이다
- 3번째 클러스터는 웃는 얼굴, 나머지는 얼굴 각도를 중시한 것 같다
- 중심과 먼 이미지는 많이 달라보이고 특별한 규칙이 없어 보인다
- 이는 k-Means이 DBSCAN과 달리 잡음 포인트 개념이 없이 모든 포인트를 구분하기 때문이다
- 클러스터가 증가하면 미세한 차이를 찾을 수 있으나 직접 조사는 어려워진다

- 병합 군집으로 얼굴 데이터 세트 분석하기

```
agglomerative = AgglomerativeClustering(n_clusters=10)
labels_agg = agglomerative.fit_predict(X_pca)
print("병합 군집의 클러스터 크기: ", np.bincount(labels_agg))
```

```
agglomerative = AgglomerativeClustering(n_clusters=10)
labels_agg = agglomerative.fit_predict(X_pca)
print("병합 군집의 클러스터 크기: ", np.bincount(labels_agg))
```

```
병합 군집의 클러스터 크기:  [169 660 144 329 217  85  18 261  31 149]
```

  - 병합 군집도 비교적 비슷한 크기인 18개에서 660개 크기의 클러스터를 만들었다
  - k-Means 보다 크기가 고르지 않지만, DBSCAN 보다 비슷한 크기다

◦ ARI 점수를 통해 병합 군집과 k-Means의 데이터 비교

```
print("ARI: {:.2f}".format(adjusted_rand_score(labels_agg, labels_km)))
```

```
ARI : 0.09
```

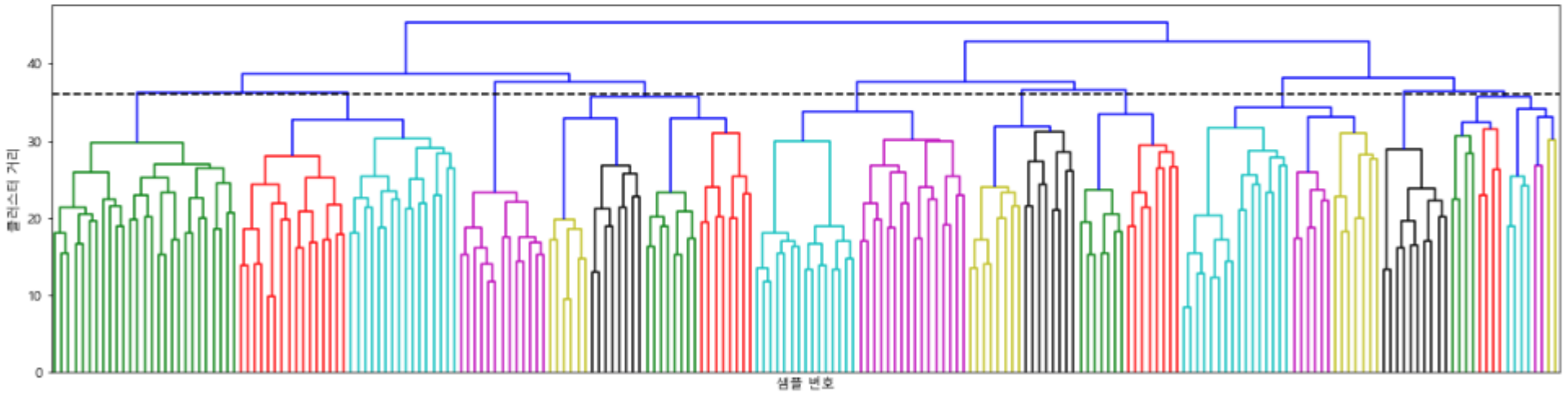
◦ ARI가 0.09 의미는 병합 군집과 k-Means의 공통 부분이 거의 없다는 뜻이다

◦ 덴드로그램으로 그리기

```
linkage_array = ward(X_pca)

plt.figure(figsize=(20, 5))
dendrogram(linkage_array, p=7, truncate_mode='level',
      no_labels=True)
plt.xlabel("샘플 번호")
plt.ylabel("클러스터 거리")
ax = plt.gca( )
bounds = ax.get_xbound( )
ax.plot(bounds, [36, 36], '—', c='k')
```

```
linkage_array = ward(X_pca)

plt.figure(figsize=(20, 5))
dendrogram(linkage_array, p=7, truncate_mode='level', no_labels=True)
plt.xlabel("샘플 번호")
plt.ylabel("클러스터 거리")
ax = plt.gca( )
bounds = ax.get_xbound( )
ax.plot(bounds, [36, 36], '--', c='k')
```

```
[<matplotlib.lines.Line2D at 0x142a678cc88>]
```



- 점선은 10개 클러스터를 자른 선이다
- 어떤 가지는 그룹을 잘 구분한 것으로 보이지만, 어떤 그룹은 알고리즘이 잘 적용되지 않은 것 같다

```
n_clusters = 10
for cluster in range(n_clusters):
    mask = labels_agg == cluster
    fig, axes = plt.subplots(1, 10, figsize=(15, 8), subplot_kw={'xticks': ( ), 'yticks': ()})
    axes[0].set_ylabel(np.sum(mask))
    for image, label, asdf, ax in zip(X_people[mask], y_people[mask], labels_agg[mask], axes):
        ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
        ax.set_title(people.target_names[label].split()[-1], fontdict={'fontsize': 9})
```


169
Kirchner Bush Bush Hewitt Beckham Bush Bush Moo-hyun Koizumi Silva
660
Menem Sharon Uribe Berlusconi Bush Chretien Menem Uribe Bush Bush
144
Robertson Chavez Giuliani Sharon Schwarzenegger Blair Uribe Schroeder Chavez Aniston
329
Bush Sampras Ryder Davenport Ridge Giuliani Blair Jolie Sharon Toledo


217
Clinton Capriati Williams Greenstock Jolie Putin Putin Capriati Lopez Bush
85
Bush Bush Williams Toledo Arroyo Capriati Aznar Chirac Davenport Arroyo
18
Negroponte Blix Berlusconi Negroponte Greenstock Putin Berlusconi Chirac Putin Giuliani
261
Chretien Powell Rumsfeld Schwarzenegger Ashcroft Schroeder Sampras Menem Blair Moo-hyun
31
Blair Capriati Williams Bush Ferrero Coria Chirac Ferrero Hewitt Hewitt
149
Ryder Erdogan Davis Powell Bush Agassi Powell Silva Powell Chavez

◦ 클러스터 수를 40개로 늘려서 조사

```
agglomerative = AgglomerativeClustering(n_clusters=40)
labels_agg = agglomerative.fit_predict(X_pca)
print("병합 군집의 클러스터 크기: ", np.bincount(labels_agg))

n_clusters = 40
for cluster in [13, 16, 23, 38, 39]:
    mask = labels_agg == cluster
    fig, axes = plt.subplots(1, 15, figsize=(15, 8), subplot_kw={'xticks': (), 'yticks': ()})
    cluster_size = np.sum(mask)
    axes[0].set_ylabel("#{}: {}".format(cluster, cluster_size))
    for image, label, asdf, ax in zip(X_people[mask], y_people[mask], labels_agg[mask], axes):
        ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
        ax.set_title(people.target_names[label].split()[-1], fontdict={'fontsize': 9})
    for i in range(cluster_size, 15):
        axes[i].set_visible(False)
```

```
병합 군집의 클러스터 크기:  [ 43 120 100 194  56  58 127  22   6  37  65  49  84  18 168  44  47  31
  78  30 166  20  57  14  11  29  23   5   8  84  67  30  57  16  22  12
  29   2  26   8]
```


#13: 18
Negroponte Blix Berlusconi Negroponte Greenstock Putin Berlusconi Chirac Putin Giuliani Chirac Woods Vajpayee Ivanov Agassi
#16: 47
Moo-hyun Koizumi Moo-hyun Koizumi Moo-hyun Hussein Moo-hyun Aniston Sukarnoputri Putin Hussein Koizumi Koizumi Koizumi Koizumi
#23: 14
Williams Williams Williams Williams Williams Williams Williams Williams Hewitt Williams Williams Williams Williams Williams
#38: 26
Bush Bush Bush Bush Bush Bush Bush Bush Jolie Bush Bush Bush Bush Annan Bush
#39: 8
Hussein Hussein Hussein Hussein Hussein Hussein Hussein Hussein

4. 군집 알고리즘 요약

- 군집 알고리즘을 적용하고 평가하는 것이 매우 정성적인 분석 과정이며 탐색적 데이터 분석에 크게 도움이 된다는 것을 알 수 있다
- k-Means, DBSCAN, 병합 군집 모두 세밀하게 조절할 수 있는 방법을 제공한다
  - k-Means와 병합 군집: 클러스터 수를 지정할 수 있다
  - DBSCAN은 eps 매개변수를 사용하여 클러스터 크기를 간접적으로 조절할 수 있다

- 각 알고리즘의 장점
  - k-Means: 각 데이터 포인트를 클러스터의 중심으로 대표할 수 있어 분해 방법이다
  - DBSCAN: 클러스터에 할당되지 않은 잡음 포인트를 인식할 수 있고 클러스터 개수를 자동으로 결정한다-복잡한 클러스터 인식
  - 병합 군집: 전체 데이터의 분할 계층도를 만들어주며 덴드로그램을 사용해 쉽게 확인 가능

요약

- 병합 군집
- DBSCAN
- 군집 알고리즘의 비교와 평가
- 군집 알고리즘 요약