군집 알고리즘의 비교와 평가

학습 내용

- 01 라이브러리 불러오기 및 데이터 준비
- 02 값의 스케일 조정
- 03 그래프로 값을 확인해 보기
- 04 ARI(adjusted rand index)를 확인해 보기
- 05 실루엣 점수를 이용한 평가

01 라이브러리 불러오기 및 데이터 준비

In [1]:
```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
import mglearn
import numpy as np
from sklearn.preprocessing import StandardScaler    # 표준화
from sklearn.metrics.cluster import adjusted_rand_score

### 한글
import matplotlib
from matplotlib import font_manager, rc
font_loc = "C:/Windows/Fonts/malgunbd.ttf"
font_name = font_manager.FontProperties(fname=font_loc).get_name()
matplotlib.rc('font', family=font_name)
```

In [2]:
```
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
print(X[1:15], y[1:15])
print(np.mean(X))
print(np.std(X))
```

```
[[ 1.61859642 -0.37982927]
 [-0.02126953  0.27372826]
 [-1.02181041 -0.07543984]
 [ 1.76654633 -0.17069874]
 [ 1.8820287  -0.04238449]
 [ 0.97481551  0.20999374]
 [ 0.88798782 -0.48936735]
 [ 0.89865156  0.36637762]
 [ 1.11638974 -0.53460385]
 [-0.36380036  0.82790185]
 [ 0.24702417 -0.23856676]
 [ 1.81658658 -0.13088387]
 [ 1.2163905  -0.40685761]
 [-0.8236696   0.64235178]] [1 1 0 1 1 0 1 0 1 0 1 1 1 0]
0.37434879176820846
0.7188000865361358
```

02 평균이 0, 분산이 1이 되도록 데이터의 스케일을 조정

In [3]:
```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
X_scaled[1:15]

print(X_scaled.shape)                # 조정된 스케일
print(np.std(X), np.std(X_scaled))   # 표준편차
print(np.var(X), np.var(X_scaled))   # 분산
```

```
print(np.mean(X[:,0]), "{:.5f}".format( np.mean(X_scaled[:,0]) ) )
print(np.mean(X[:,1]), "{:.5f}".format( np.mean(X_scaled[:,1]) ) )
```

```
(200, 2)
0.7188000865361358 1.0
0.5166735644043563 1.0
0.4964808628141405 0.00000
0.25221672072227647 -0.00000
```

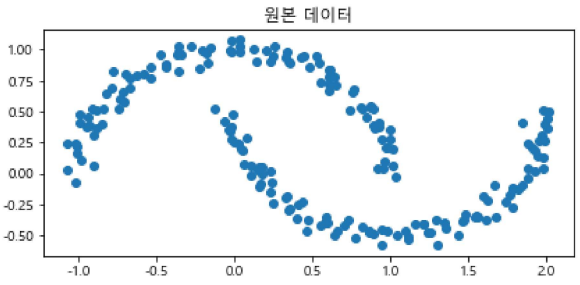
03 데이터 시각화

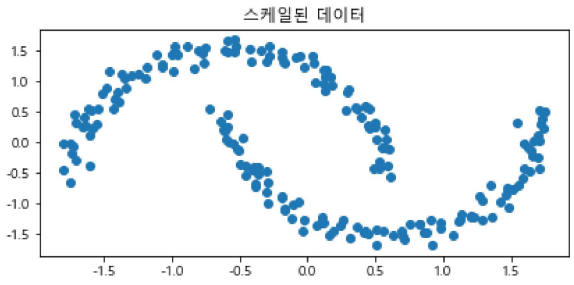
- 원본 데이터와 스케일된 데이터 값을 확인해 보기

In [5]:
```
matplotlib.rcParams['axes.unicode_minus'] = False
```

In [6]:
```
fig, axes = plt.subplots(1, 2, figsize=(15,3) )

# 무작위로 할당한 클러스터를 그린다.
axes[0].scatter(X[:,0], X[:,1])
axes[0].set_title("원본 데이터")

# 무작위로 할당한 클러스터를 그린다.
axes[1].scatter(X_scaled[:,0], X_scaled[:,1])
axes[1].set_title("스케일된 데이터")
```

Out[6]: Text(0.5, 1.0, '스케일된 데이터')





04 ARI(adjusted rand index)를 확인해 보기

- ARI의 경우 타깃값을 아는 경우에 쓰이는 방법이다.
- 군집알고리즘의 결과를 실제 정답 클러스터와 비교하여 평가할 수 있는 지표가 있다. ARI와 NMI가 그 지표이다.
- 데이터 클러스터링에서 Rand 인덱스 값은 두 데이터 클러스터링 간의 유사성의 측정값.

In [8]:
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
```

데이터 준비

In [10]:
```
# 사용할 알고리즘 모델의 리스트를 만들기
algorithms = [KMeans(n_clusters=2),
              AgglomerativeClustering(n_clusters=2),
              DBSCAN()]

# 비교를 위해 무작위로 클러스터를 할당
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))

random_clusters
```

Out[10]: array([0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,

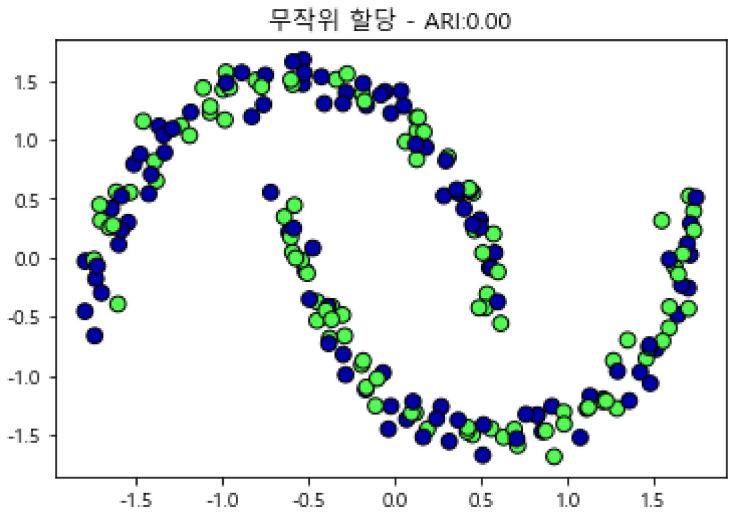
```
       1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0])
```

무작위 할당한 클러스터 확인

In [11]:
```
print(mglearn.cm3)
plt.scatter(X_scaled[:,0],
                X_scaled[:,1],
                c=random_clusters,
                cmap=mglearn.cm3,
                s=60,                    # 점 크기
                edgecolors='black')

plt.title("무작위 할당 - ARI:{:.2f}".format(adjusted_rand_score(y, random_clusters)))
```

```
<matplotlib.colors.ListedColormap object at 0x0000023A9D816250>
```

Out[11]: Text(0.5, 1.0, '무작위 할당 - ARI:0.00')

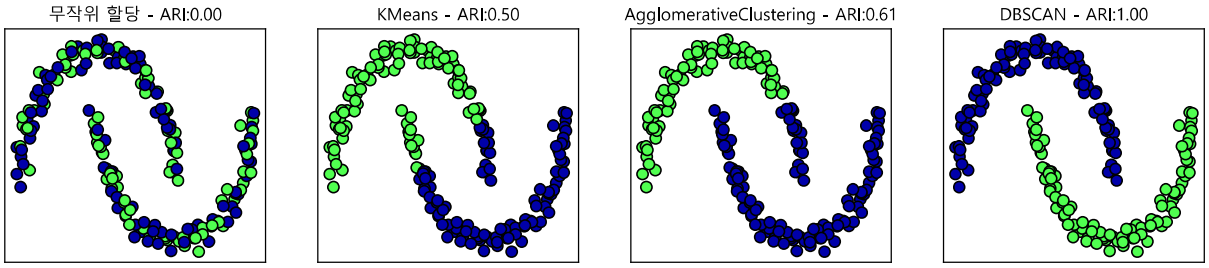


군집 알고리즘 비교 (ARI 지표를 활용한)

- 첫번째 그래프 : 무작위 할당
- 두번째 그래프 : K-Means
- 세번째 그래프 : AgglomerativeClustering : 병합 군집
- 네번째 그래프 : DBSCAN

In [18]:
```
# from sklearn.metrics.cluster import adjusted_rand_score
fig, axes = plt.subplots(1, 4, figsize=(15,3),
                         subplot_kw = {'xticks':(), 'yticks':()})

# 무작위로 할당한 클러스터를 그린다.
axes[0].scatter(X_scaled[:,0],
                X_scaled[:,1],
                c=random_clusters,
                cmap=mglearn.cm3,
                s=60,                    # 점 크기
                edgecolors='black')

axes[0].set_title("무작위 할당 - ARI:{:.2f}".format(adjusted_rand_score(y, random_clus
```

```
for ax, algorithm in zip(axes[1:], algorithms):
    # 클러스터 할당과 클러스터 중심을 그린다.
    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)))
```



확인

클러스터를 무작위 할당했을 때는 ARI 점수는 0이다.
DBSCAN은 완벽하게 군집을 만들어냈으므로) 점수가 1이다.

실수 주의***
군집 모델 평가할 때, 흔히하는 실수는 **ARI(adjusted_rand_score)**나 **NMI(normalized_mutual_info_score)** 같은 군집용 측정도구를 사용하지 않고, **accuracy_score**를 사용하는 것이다.

정확도를 사용하면 할당된 클러스터의 레이블 이름이 실제 레이블과 맞는지 확인한다.
그러나 클러스터 레이블은 그 자체로 의미가 있는 것이 아니며 포인트들이 같은 클러스터에 속해
있는가만이 중요하다.

In [12]:
```
### 실제 예
from sklearn.metrics import accuracy_score

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

# 모든 레이블이 달라졌으므로 정확도는 0이다.
print("정확도 : {:.2f}".format(accuracy_score(clusters1, clusters2)))

# 같은 포인트가 한 클러스터에 모였으므로 ARI는 1이다.
print("ARI: {:.2f}".format(adjusted_rand_score(clusters1, clusters2)))
```

```
정확도 : 0.00
ARI: 1.00
```

문제 발생!!!

- 군집 알고리즘을 적용할 때는 보통 그 결과와 비교할 타깃값이 없다.
- 만약 정확한 클러스터를 알고 있다면 이 정보를 이용하여 분류기와 같은 지도학습 모델을 만들 것이다.

ARI, NMI같은 지표는 애플리케이션의 성능 평가가 아니라 알고리즘을 개발할 때 도움이 된다.

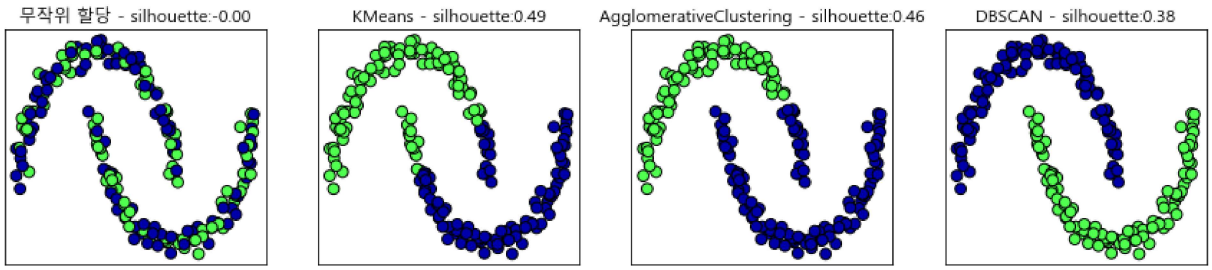
05 실루엣 점수를 이용한 평가

타깃값이 필요없는 군집용 지표.

- 한 클러스터 안의 데이터들이 다른 클러스터와 비교해서 얼마나 비슷한가를 나타내는 지표
  - 클러스터 안의 거리가 짧으면 좋고(cohesion).
  - 다른 클러스터와 비교해서 먼 것이 좋다.(separation)
  - 실루엣 점수는 -1~1까지의 값을 갖는다.
  - 값이 높을 수록 좋다.
- 다만, 밀집된 군집에는 평가가 잘 들어맞지 않음.

In [13]:
```
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)

# 사용할 알고리즘 모델의 리스트를 만들기
algorithms = [KMeans(n_clusters=2),
              AgglomerativeClustering(n_clusters=2),
              DBSCAN()]

# 비교를 위해 무작위로 클러스터를 할당
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))
```

In [14]:
```
# from sklearn.metrics.cluster import adjusted_rand_score
fig, axes = plt.subplots(1, 4, figsize=(15,3),
                         subplot_kw = {'xticks':(), 'yticks':()})

# 무작위로 할당한 클러스터를 그린다.
axes[0].scatter(X_scaled[:,0],
                X_scaled[:,1],
                c=random_clusters,
                cmap=mglearn.cm3,
                s=60,                    # 점 크기
                edgecolors='black')

# 실루엣 스코어
axes[0].set_title("무작위 할당 - silhouette:{:.2f}".format(silhouette_score(X_scaled,

for ax, algorithm in zip(axes[1:], algorithms):
    # 클러스터 할당과 클러스터 중심을 그린다.
    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("{} - silhouette:{:.2f}".format(algorithm.__class__.__name__,
                                                 silhouette_score(X_scaled, clusters)))
```



그래프 알아보기

(가) DBSCAN의 결과가 k-평균 실루엣 점수보다 높다.
클러스터 평가에 더 적합한 전략은 견고서 기반(robustness-based)의 지표이다.

클러스터 평가에 더 적합한 전략은 견고성 기반(robustness-based)의 지표

- 이 책의 쓰여진 시점에는 scikit-learn에는 기능이 구현되어 있지 않음