

# Introduction to Machine Learning with Python

## 3. Unsupervised Learning

Honedae Machine Learning Study Epoch #2

# Contacts

Haesun Park

Email : [haesunrpark@gmail.com](mailto:haesunrpark@gmail.com)

Meetup: <https://www.meetup.com/Hongdae-Machine-Learning-Study/>

Facebook : <https://facebook.com/haesunrpark>

Blog : <https://tensorflow.blog>

# Book

파이썬 라이브러리를 활용한 머신러닝, 박해선.

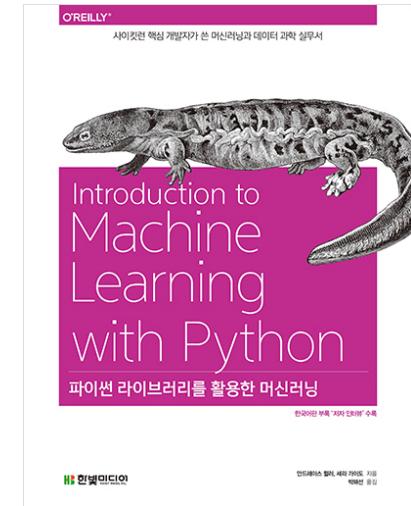
(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

Github:

[https://github.com/rickiepark/introduction\\_to\\_ml\\_with\\_python/](https://github.com/rickiepark/introduction_to_ml_with_python/)



# 머신러닝 구분

지도 학습(Supervised) – KNN, Linear models, SVM, Random Forest, Boosting, Neural Networks

비지도 학습(Unsupervised) – 변환(Scaler, PCA, NMF, tSNE), 군집(K-means, 병합군집, DBSCAN)

준지도 학습(Semisupervised) – Label propagation, Label spreading  
(`sklearn.semi_supervised`)

강화 학습(Reinforcement Learning)

# 비지도 학습

# 종류

비지도 변환 unsupervised transformation

사람이나 머신러닝 알고리즘을 위해 새로운 데이터 표현을 만듭니다

차원 축소: PCA, NMF, t-SNE, 오토인코더 autoencoder

토픽 모델링: LDA (e.g. 소셜 미디어 토론 그룹핑, 텍스트 문서의 주제 추출)

군집 clustering

비슷한 데이터 샘플을 모으는 것 (e.g. 사진 어플리케이션: 비슷한 얼굴 그룹핑)

k-평균, 병합, DBSCAN

데이터 스케일링 scaling

이상치 탐지 anomaly detection, GAN generative adversarial networks

# 비지도 학습의 어려움

레이블이 없기 때문에 올바르게 학습되었는지 알고리즘을 평가하기가 어렵습니다

사진 애플리케이션에서 옆모습, 앞모습으로 군집되었다 해도 직접 눈으로 확인하기 전에는 결과를 평가하기가 어렵습니다.

비지도 학습은 데이터를 잘 이해하기 위해서(탐색적 분석) 사용되거나,

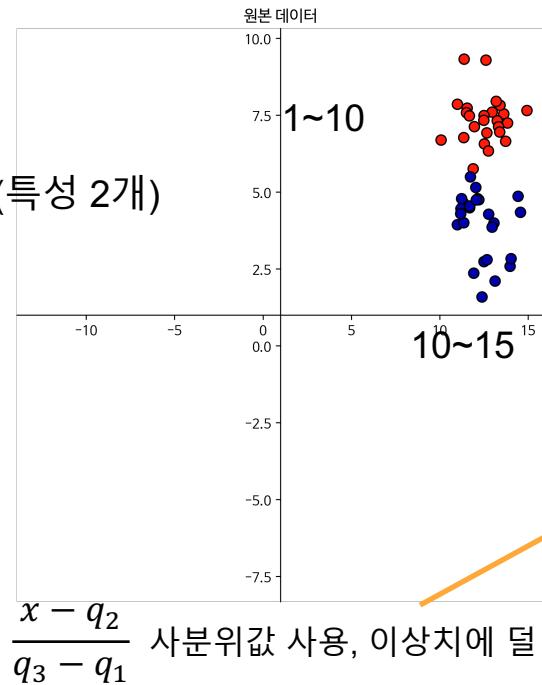
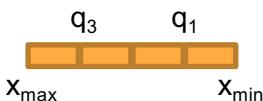
지도학습 알고리즘의 정확도 향상과 메모리/시간 절약을 위해 새로운 데이터 표현을 만드는 전처리 단계로 활용

# 데이터 스케일 조정

# 네 가지 스케일링

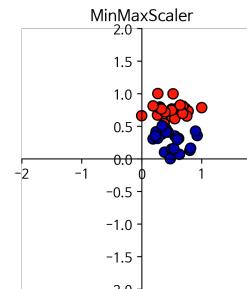
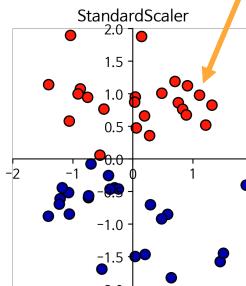
SVM, 신경망등은 스케일에 민감합니다.(신경망에서는 배치정규화로 이를 완화시킵니다)

인위적으로 만든  
이진 분류 데이터(특성 2개)



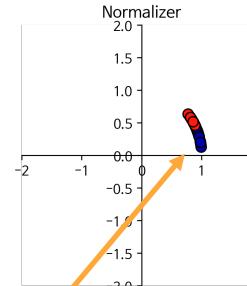
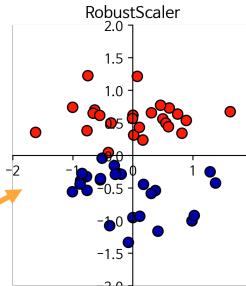
$\frac{x - q_2}{q_3 - q_1}$  사분위값 사용, 이상치에 덜 민감

$\frac{x - \bar{x}}{\sigma}$  표준점수, z-점수: 평균 0, 분산 1



$\frac{x - x_{min}}{x_{max} - x_{min}}$

모든 특성이  
0과 1사이에 위치



Normalizer(norm='l2')  
다른 방법과 달리  
행(포인트) 별로 적용  
(유clidean 거리를 1로 만들)

$$\frac{X}{\sqrt{x_1^2 + x_2^2}}$$

# cancer + MinMaxScaler

```
In [4]: from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    random_state=1)

print(X_train.shape)
print(X_test.shape)
```

(426, 30)  
(143, 30)

NumPy 배열 (569, 30)

```
In [5]: from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

```
In [6]: scaler.fit(X_train)

Out[6]: MinMaxScaler(copy=True, feature_range=(0, 1))
```

y\_train은 사용하지 않음  
fit(): 특성마다 최대, 최솟값을 계산합니다  
transform(): 데이터를 변환합니다

# MinMaxScaler.transform

학습한 변환 내용을 훈련 데이터에 적용

fit → transform

In [7]:

```
# 데이터 변환
X_train_scaled = scaler.transform(X_train)
# 스케일이 조정된 후 데이터셋의 속성을 출력합니다
print("변환된 후 크기: {}".format(X_train_scaled.shape))
print("스케일 조정 전 특성별 최소값:\n{}".format(X_train.min(axis=0)))
print("스케일 조정 전 특성별 최대값:\n{}".format(X_train.max(axis=0)))
print("스케일 조정 후 특성별 최소값:\n{}".format(X_train_scaled.min(axis=0)))
print("스케일 조정 후 특성별 최대값:\n{}".format(X_train_scaled.max(axis=0)))
```

numpy 배열의  
min, max 메서드

변환된 후 크기: (426, 30)

스케일 조정 전 특성별 최소값:

```
[ 6.981    9.71   43.79   143.5     0.053    0.019     0.      0.
  0.106    0.05   0.115    0.36     0.757    6.802     0.002    0.002
  0.        0.      0.01    0.001    7.93     12.02    50.41    185.2
  0.071    0.027   0.       0.      0.157    0.055]
```

스케일 조정 전 특성별 최대값:

```
[ 28.11    39.28   188.5   2501.     0.163    0.287    0.427
  0.201    0.304   0.096    2.873    4.885    21.98    542.2
  0.031    0.135   0.396    0.053    0.061    0.03     36.04
  49.54    251.2   4254.    0.223    0.938    1.17     0.291
  0.577    0.1491]
```

스케일 조정 후 특성별 최소값:

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

스케일 조정 후 특성별 최대값:

```
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

MinMaxScaler()으로  
최소와 최댓값이  
0, 1로 바뀜

# transform(X\_test)

$$\frac{x_{test} - x_{train\_min}}{x_{train\_max} - x_{train\_min}}$$

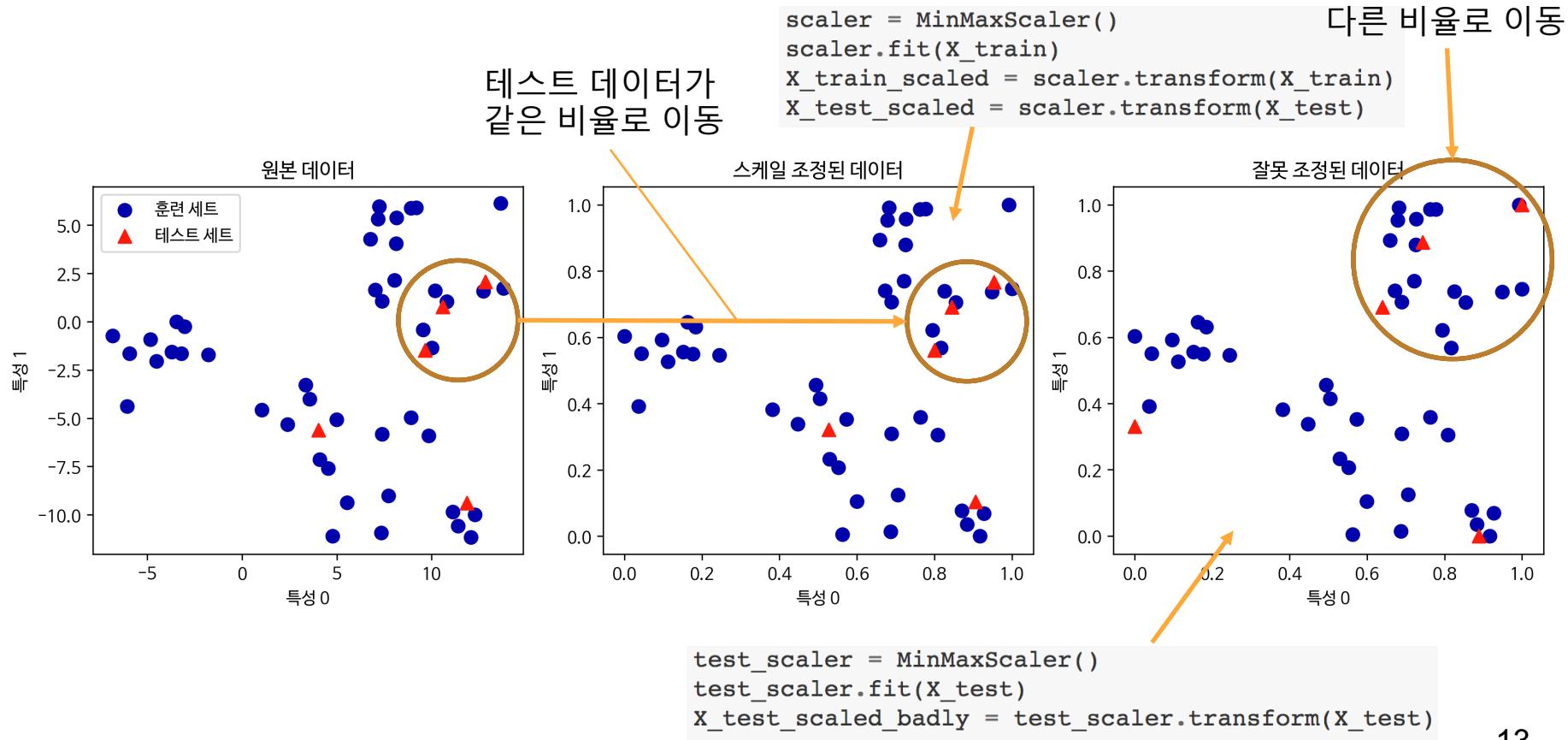
훈련 세트로 학습시킨 scaler를 사용해 X\_test를 변환  
(SVM의 성능을 측정하기 위해)

```
In [8]: # 테스트 데이터 변환
X_test_scaled = scaler.transform(X_test)
# 스케일이 조정된 후 테스트 데이터의 속성을 출력합니다
print("스케일 조정 후 특성별 최소값:\n{}".format(X_test_scaled.min(axis=0)))
print("스케일 조정 후 특성별 최대값:\n{}".format(X_test_scaled.max(axis=0)))
```

|                   |  |
|-------------------|--|
| 스케일 조정 후 특성별 최소값: | [ 0.034 0.023 0.031 0.011 0.141 0.044 0. 0. 0.154 -0.006<br>-0.001 0.006 0.004 0.001 0.039 0.011 0. 0. -0.032 0.007<br>0.027 0.058 0.02 0.009 0.109 0.026 0. 0. -0. -0.002 ]               |
| 스케일 조정 후 특성별 최대값: | [ 0.958 0.815 0.956 0.894 0.811 1.22 0.88 0.933 0.932 1.037<br>0.427 0.498 0.441 0.284 0.487 0.739 0.767 0.629 1.337 0.391<br>0.896 0.793 0.849 0.745 0.915 1.132 1.07 0.924 1.205 1.631 ] |

테스트 세트의 최소, 최댓값이 0~1 사이를 벗어남

# train과 test의 스케일 조정



# <단축 메서드>

```
In [10]: from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
# 메소드 체이닝(chaining)을 사용하여 fit과 transform을 연달아 호출합니다  
X_scaled = scaler.fit(X_train).transform(X_train)  
# 위와 동일하지만 더 효율적입니다  
X_scaled_d = scaler.fit_transform(X_train)
```

transform 메서드를 가진 변환기는 모두 fit\_transform 메서드를 가지고 있습니다.

TransformerMixin 클래스를 상속하는 대부분의 경우 fit\_transform은 단순히 scaler.fit().transform()처럼 연이어 호출합니다.

테스트 데이터를 변환하려면 반드시 transform()을 사용합니다.

일부의 경우 fit\_transform 메서드가 효율적인 경우가 있습니다(PCA)

# scaler + SVC

```
In [11]: from sklearn.svm import SVC  
  
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,  
                                                    random_state=0)  
  
svm = SVC(C=100)  
svm.fit(X_train, y_train)  
print("테스트 세트 정확도: {:.2f}".format(svm.score(X_test, y_test)))
```

테스트 세트 정확도: 0.63

```
In [12]: # 0~1 사이로 스케일 조정  
scaler = MinMaxScaler()  
scaler.fit(X_train)  
X_train_scaled = scaler.transform(X_train)  
X_test_scaled = scaler.transform(X_test)  
  
# 조정된 데이터로 SVM 학습  
svm.fit(X_train_scaled, y_train)
```

2장에서 수동으로 했던 것을  
MinMaxScaler 클래스로 대신함

```
# 스케일 조정된 테스트 세트의 정확도  
print("스케일 조정된 테스트 세트의 정확도: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

스케일 조정된 테스트 세트의 정확도: 0.97

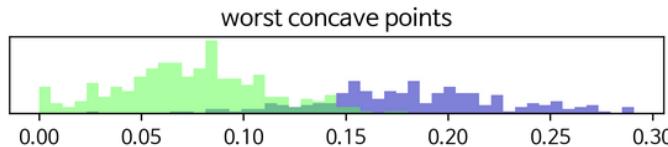
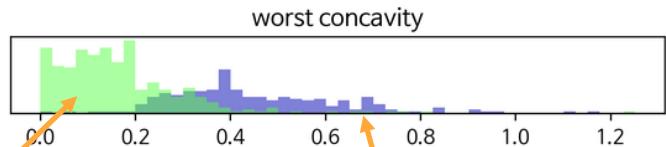
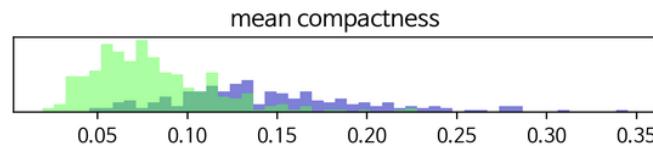
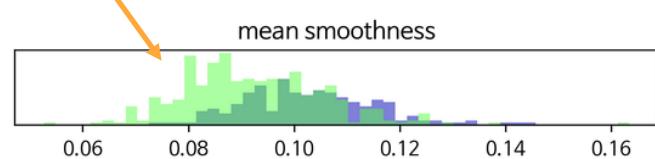
# PCA

# cancer 히스토그램

cancer 데이터의 산점도 행렬은  $\binom{30}{2} = 435$ 개가 그려집니다.

대신 2개씩 짹지어 히스토그램을 그릴 수 있습니다.

아주 유용한 특성



양성

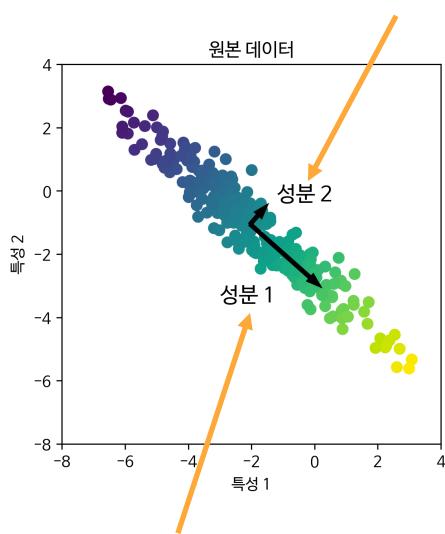
악성

유용성이 떨어짐

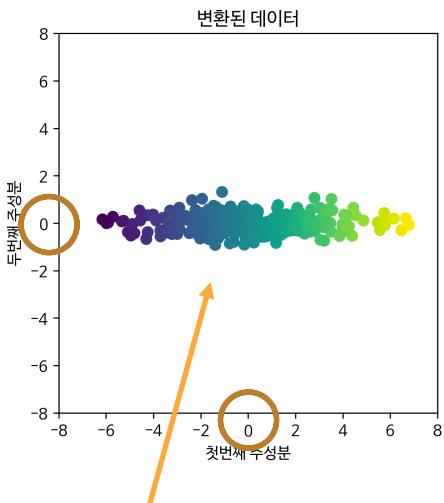
# 주성분 분석 Principal component analysis

상관관계가 없는 특성으로 데이터셋을 변환(회전)하고 일부 특성을 선택하는 기술

첫번째와 직각이면서  
가장 분산이 큰 두번째 방향

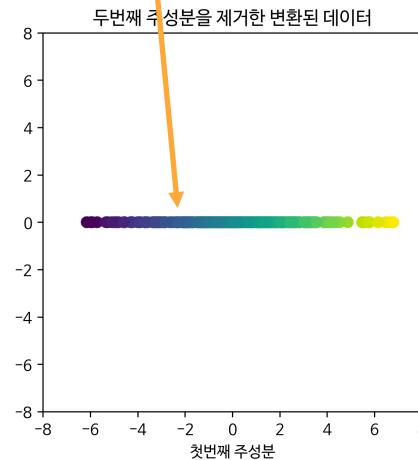


분산이 가장 큰 방향  
(화살표 방향은 의미 없음)

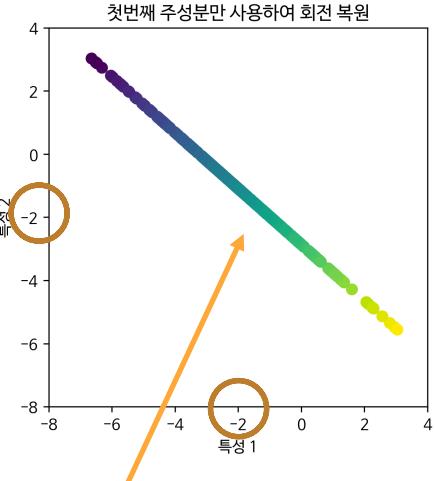


성분1, 2로 데이터를 변환  
(특성의 수 == 주성분의 수)

두번째 성분 제거(차원 축소)



다시 원래 특성 차원으로 변환  
(노이즈제거, 유용한 특성 시각화)



# PCA와 특이값분해

행렬  $X$ 의 공분산

$$\text{Cov}(X_i, X_j) = \frac{1}{n-1} [(X_i - \bar{X}_i)(X_j - \bar{X}_j)]$$

$$\begin{bmatrix} V_1 & C_{1,2} & C_{1,3} \\ C_{1,2} & V_2 & C_{2,3} \\ C_{1,3} & C_{2,3} & V_3 \end{bmatrix}$$

평균을 뺀 행렬로 표현하면(평균이 0이 되도록 변환)

$$\text{Cov}(X, X) = \frac{1}{n-1} X^T X$$

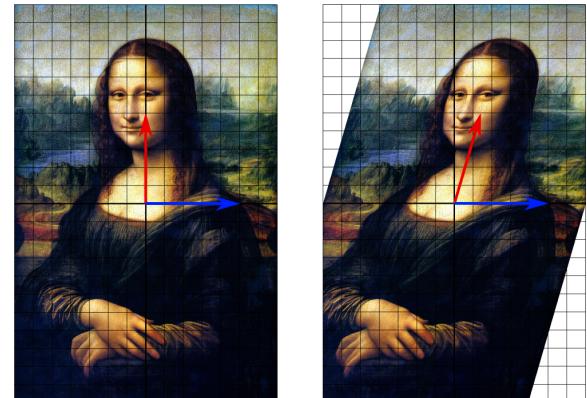
특이값 분해(SVD)  $X = USV^T$ 로 부터

$$\text{Cov}(X, X) = \frac{1}{n-1} X^T X = \frac{1}{n-1} (VSU^T)USV^T = V \frac{S^2}{n-1} V^T$$

$\text{Cov}(X, X)$  행렬의 고유벡터( $V$ )  $\rightarrow$  분산의 주방향

$$X_{pca} = XV = USV^T V = US$$

행렬의 고유벡터  $\rightarrow$   
크기만 바뀌고 방향은 바뀌지 않음  
 $y = Ax = \lambda x$   
 $A = V d(\lambda) V^{-1}$



# PCA 변환 차원

$$X_{pca} = XV = USV^T V = US$$

$X = [m \times n]$  행렬, m: 샘플수, n: 특성수, p: 주성분수

$U = [m \times m]$  행렬,  $S = [m \times n]$  행렬,  $V = [n \times n]$  행렬

$X_{new}V = [m \times n] \cdot [n \times p] = [m \times p]$  :  $X_{new}$ 가 주어지면 V의 몇 개 열만 사용, fit() → transform()

$US = [m \times m] \cdot [m \times p] = [m \times p]$  : 학습한 데이터를 변환할 때는 S의 몇 개 열만 사용, fit\_transform()

# PCA + cancer

올바른 주성분을 찾기 위해  
특성의 스케일을  
분산이 1이 되도록 변경

```
scaler = StandardScaler()  
scaler.fit(cancer.data)  
X_scaled = scaler.transform(cancer.data)
```

In [17]: `from sklearn.decomposition import PCA`

# 데이터의 처음 두 개 주성분만 유지시킵니다

pca = PCA(n\_components=2)

# 유방암 데이터로 PCA 모델을 만듭니다

pca.fit(X\_scaled)

# 처음 두 개의 주성분을 사용해 데이터를 변환합니다

X\_pca = pca.transform(X\_scaled)

print("원본 데이터 형태: {}".format(str(X\_scaled.shape)))

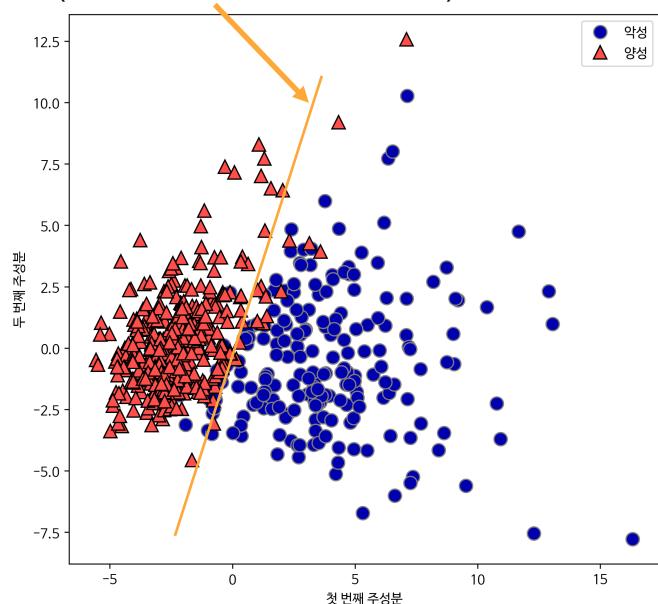
print("축소된 데이터 형태: {}".format(str(X\_pca.shape)))

원본 데이터 형태: (569, 30)

축소된 데이터 형태: (569, 2)

PCA 변환: X를 2개의 주성분으로 변환(회전, 차원축소)

클래스 정보 없이 데이터를 구분했습니다  
(선형 분류기도 타당해 보임)



# pca.components\_

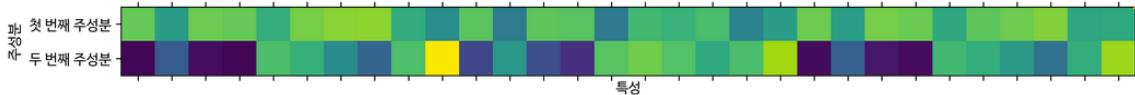
components\_ 속성에 주성분 방향 principal axis  $V^T([nxn])$ 가 부분 저장되어 있습니다.

components\_ 크기 (n\_components, n\_features)

```
In [17]: from sklearn.decomposition import PCA  
# 데이터의 처음 두 개 주성분을 유지시킵니다  
pca = PCA(n_components=2)  
# 유방암 데이터로 PCA 모델을 만듭니다  
pca.fit(X_scaled)
```

```
In [19]: print("PCA 주성분 형태: {}".format(pca.components_.shape))  
PCA 주성분 형태: (2, 30)
```

```
In [20]: print("PCA 주성분: {}".format(pca.components_))  
PCA 주성분: [[ 0.219  0.104  0.228  0.221  0.143  0.239  0.258  0.261  0.138  0.064  
   0.206  0.017  0.211  0.203  0.015  0.17   0.154  0.183  0.042  0.103  
   0.228  0.104  0.237  0.225  0.128  0.21   0.229  0.251  0.123  0.132]  
  [-0.234 -0.06  -0.215 -0.231  0.186  0.152  0.06  -0.035  0.19   0.367  
  -0.106  0.09  -0.089 -0.152  0.204  0.233  0.197  0.13   0.184  0.28  
  -0.22  -0.045 -0.2   -0.219  0.172  0.144  0.098 -0.008  0.142  0.275]]
```



주성분 방향을 직관적으로 설명하기 어렵습니다.

30개의 특성

# LFW dataset

원본 데이터 보다 분석하기 좋은 표현을 찾는 특성 추출로 사용 가능합니다.

메사추세츠 애머스트 주립대의 비전랩에서 만든 LFW Labeled Faces in the Wild 데이터셋  
(인터넷에서 모은 2000년대 초반 유명 인사의 얼굴 이미지)

```
In [22]: from sklearn.datasets import fetch_lfw_people  
people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
```



총 62명의 얼굴  
3,023개 → 2,063개 이미지  
(사람마다 최대 50개씩)

각 이미지는 87x65 사이즈  
→ 5,655 개의 특성

# k-NN + Ifw

얼굴인식: DB에 있는 얼굴 중 하나를 찾는 작업(사람마다 사진수가 적음)

얼굴 인식을 위해 (지도 학습) 1-최근접 이웃을 적용합니다.

87x65 픽셀 값의 거리를 계산하므로 위치에 매우 민감합니다.

1547개

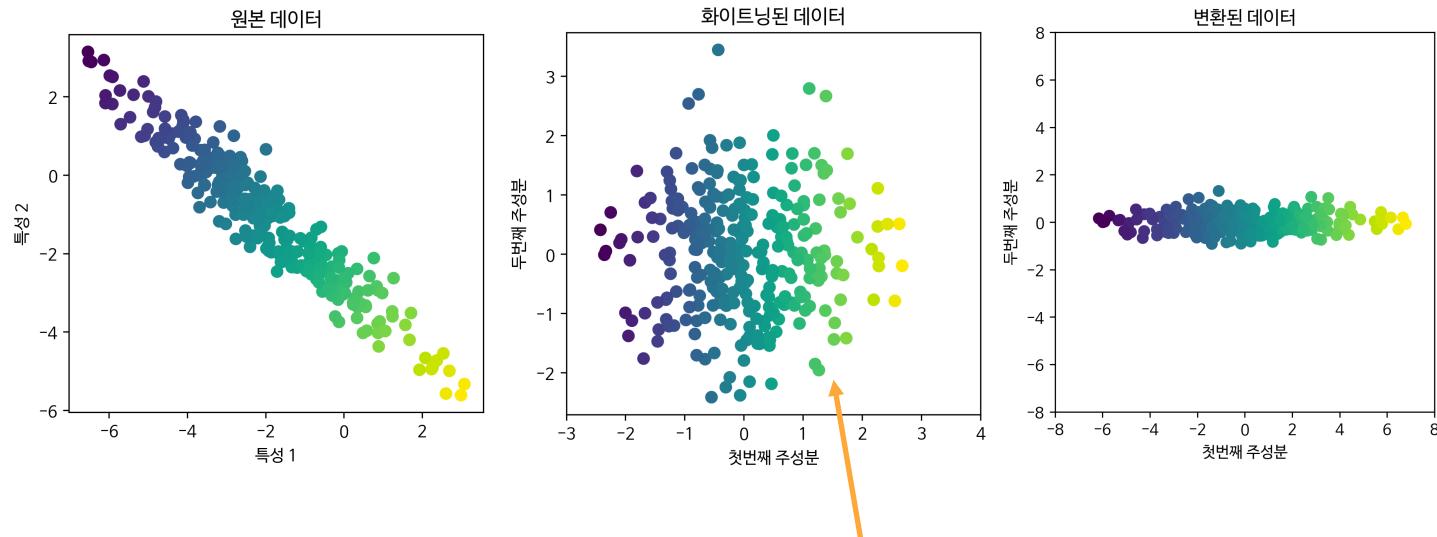
```
In [27]: from sklearn.neighbors import KNeighborsClassifier
# 데이터를 훈련 세트와 테스트 세트로 나눕니다
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
# 이웃 개수를 한 개로 하여 KNeighborsClassifier 모델을 만듭니다
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("1-최근접 이웃의 테스트 세트 점수: {:.2f}".format(knn.score(X_test, y_test)))
```

1-최근접 이웃의 테스트 세트 점수: 0.23

무작위로 분류할 경우  $1/62=0.016$

# 화이트닝 whitening

백색소음에서 이름이 유래됨, 특성 간의 (대각행렬을 제외한) 공분산이 모두 0이 되고(PCA) 특성의 분산이 1로 되는(StandardScaler) 변환



# k-NN + whitening

```
In [29]: pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
In [30]: knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print("테스트 세트 정확도: {:.2f}".format(knn.score(X_test_pca, y_test)))
```

테스트 세트 정확도: 0.31

주성분의 갯수

```
In [31]: print("pca.components_.shape: {}".format(pca.components_.shape))
```

pca.components\_.shape: (100, 5655)

픽셀수==특성의 수==주성분의 벡터 방향

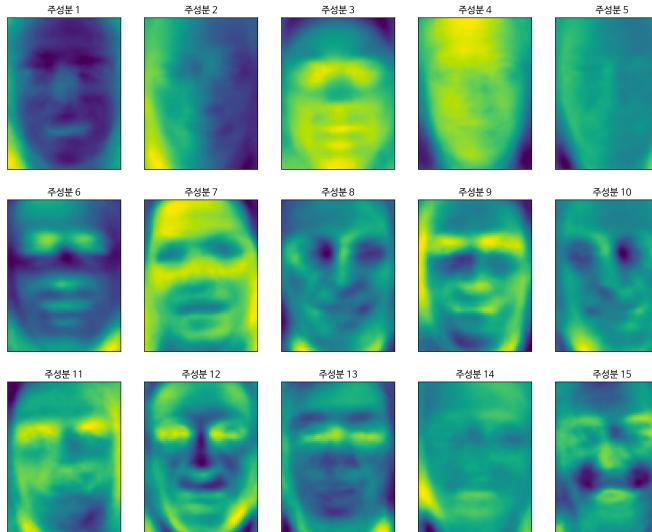
# Ifw 주성분

배경, 명양, 조명의 차이를 구분하고 있습니다. 하지만 사람이 이미지를 판단할 때는 정성적인 속성(나이, 표정, 인상 등)을 사용합니다.

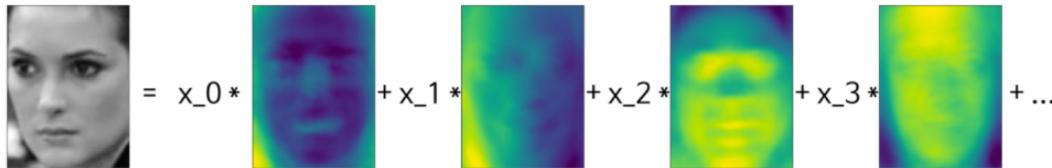
신경망: 입력 이미지에서 유용한 특성을 찾음(representative learning, end-to-end learning)

$$V^T = \text{components}_  
= [n\_component, features]$$

$$5655 = 87 \times 65$$



# PCA as weighted sum

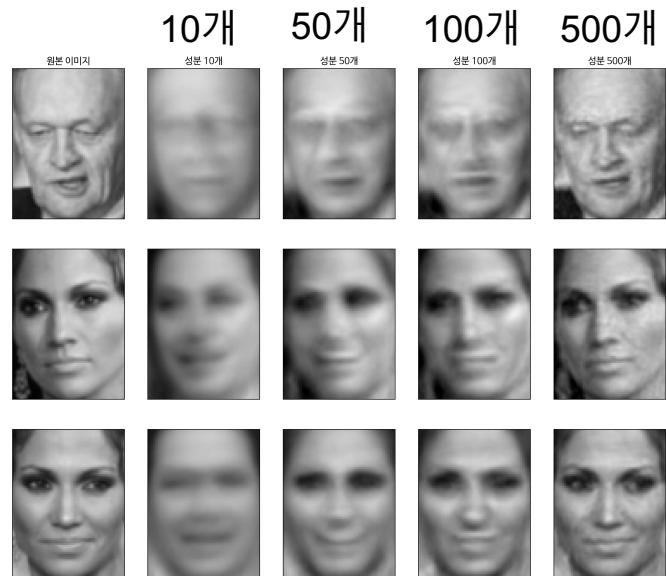

$$\text{Portrait} = x_0 * \text{Component 1} + x_1 * \text{Component 2} + x_2 * \text{Component 3} + x_3 * \text{Component 4} + \dots$$

주성분 방향으로 변환한 데이터(X\_trained\_pca)에  
주성분 방향을 곱하여 원본을 복원할 수 있습니다.

$$[1 \times 5655] = [1 \times 100] \cdot [100 \times 5655]$$

원본 샘플을 주성분의 가중치 합으로  
표현할 수 있습니다.

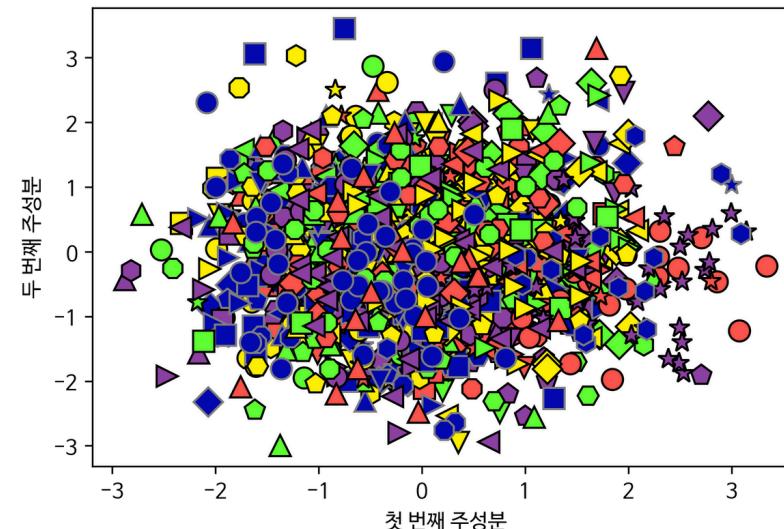
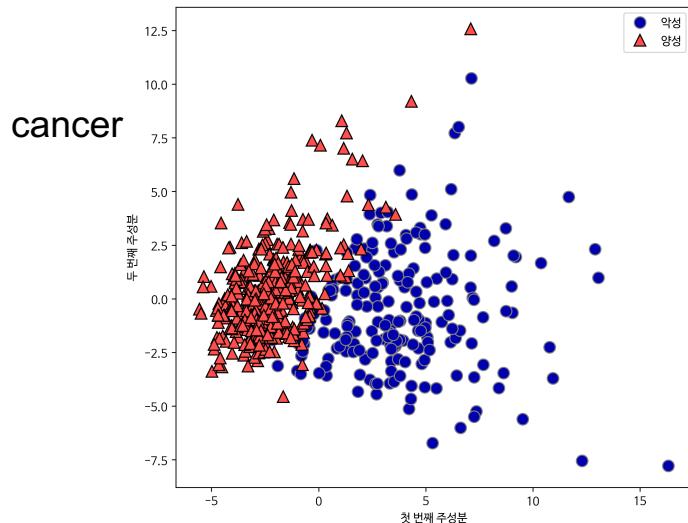
5655개의 주성분을 사용하면 완전히 복원됩니다.



# Ifw 산점도

cancer 때와 마찬가지로 2개의 주성분으로 산점도를 그려서 비교합니다.

10개의 주성분으로도 원본이 잘 표현되지 못했으므로 이런 산점도 결과를 예상할 수 있습니다.



# 비음수 행렬 분해(NMF)

# NMF

PCA와 비슷하고 차원 축소에 사용할 수 있습니다.

원본 = 뽑아낸 성분의 가중치의 합

음수가 아닌 성분과 계수를 만드므로 양수로 이루어진 데이터에 적용 가능합니다.

여러 악기나 목소리가 섞인 오디오 트랙처럼 독립된 소스가 덧붙여진 데이터를 분해하는데 유용합니다.

성분과 계수가 음수를 가진 PCA 보다 NMF를 이해하기가 쉽습니다.

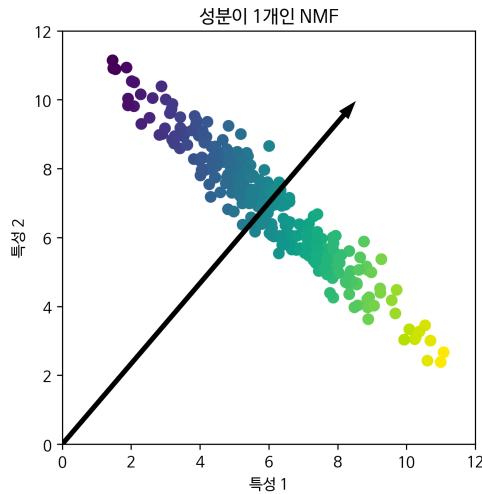
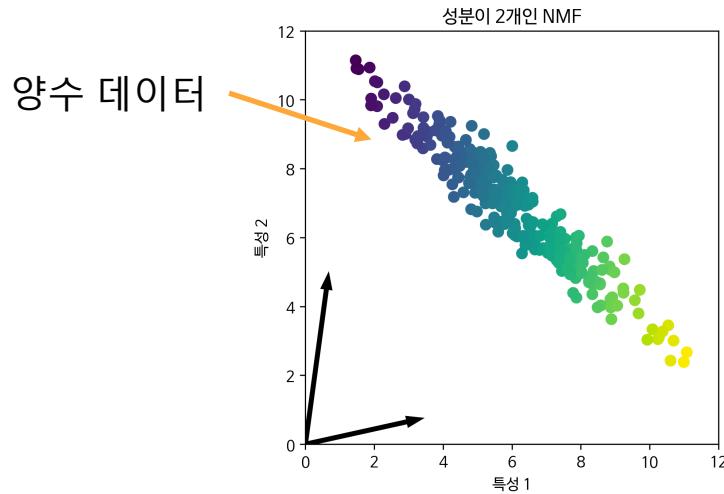
$X = WH$  를 만족하는  $W$ 와  $H$ 를 구함

$W$ : 변환된 데이터,  $H$ : 성분(*components\_*에 저장)

손실함수(L2 노름의 제곱):  $\frac{1}{2} \sum (X_{ij} - WH_{ij})^2$

좌표하강법

# NMF + 인공 데이터셋



성분이 특성 개수 만큼 많다면 데이터의 끝 부분을 가리키고 하나일 경우에는 평균 방향을 가리킵니다.

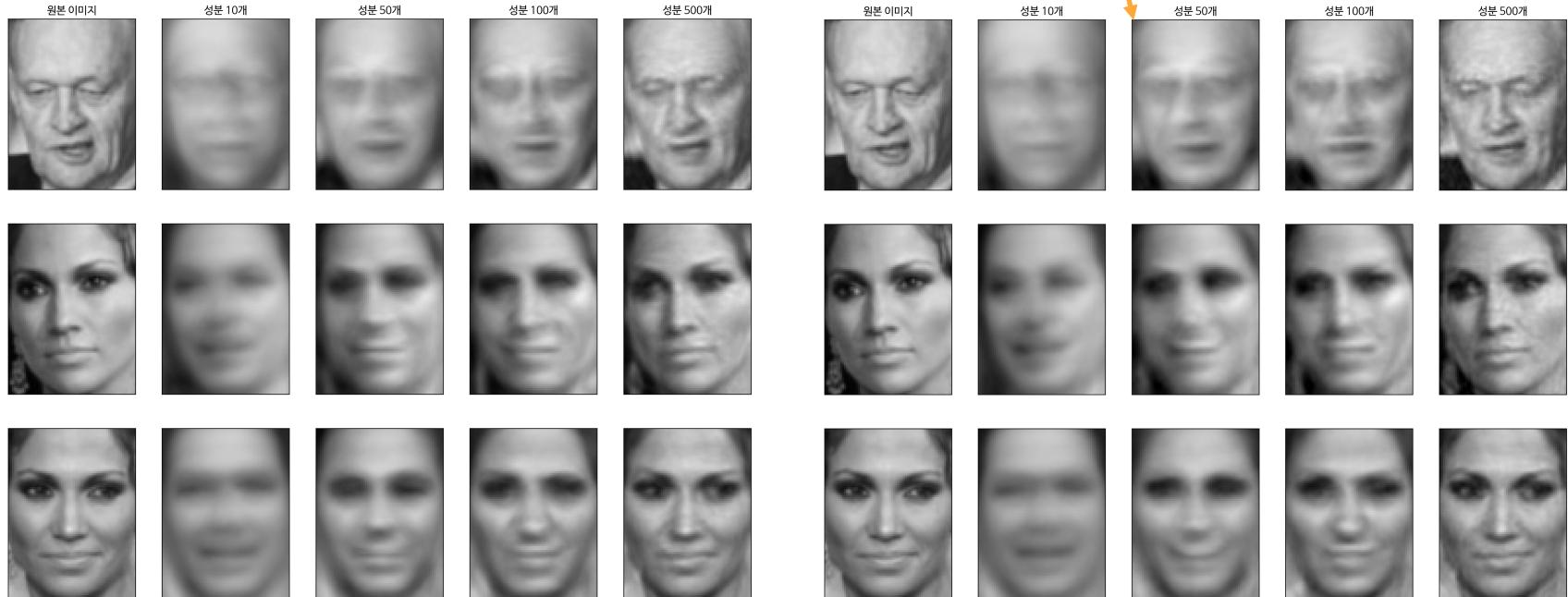
성분의 개수를 변경하면 전체 성분이 모두 바뀌고 성분에 순서가 없습니다.

무작위 초기화를 사용하므로 난수 초기 값에 따라 결과가 달라집니다.

데이터 평균을 구해서 성분 개수로 나눈 제곱근에 정규분포 난수를 곱해서  $W$ ,  $H$  를 초기화합니다.

# NMF + Ifw

PCA가 재구성에 유리한 성분(분산)을 찾습니다.



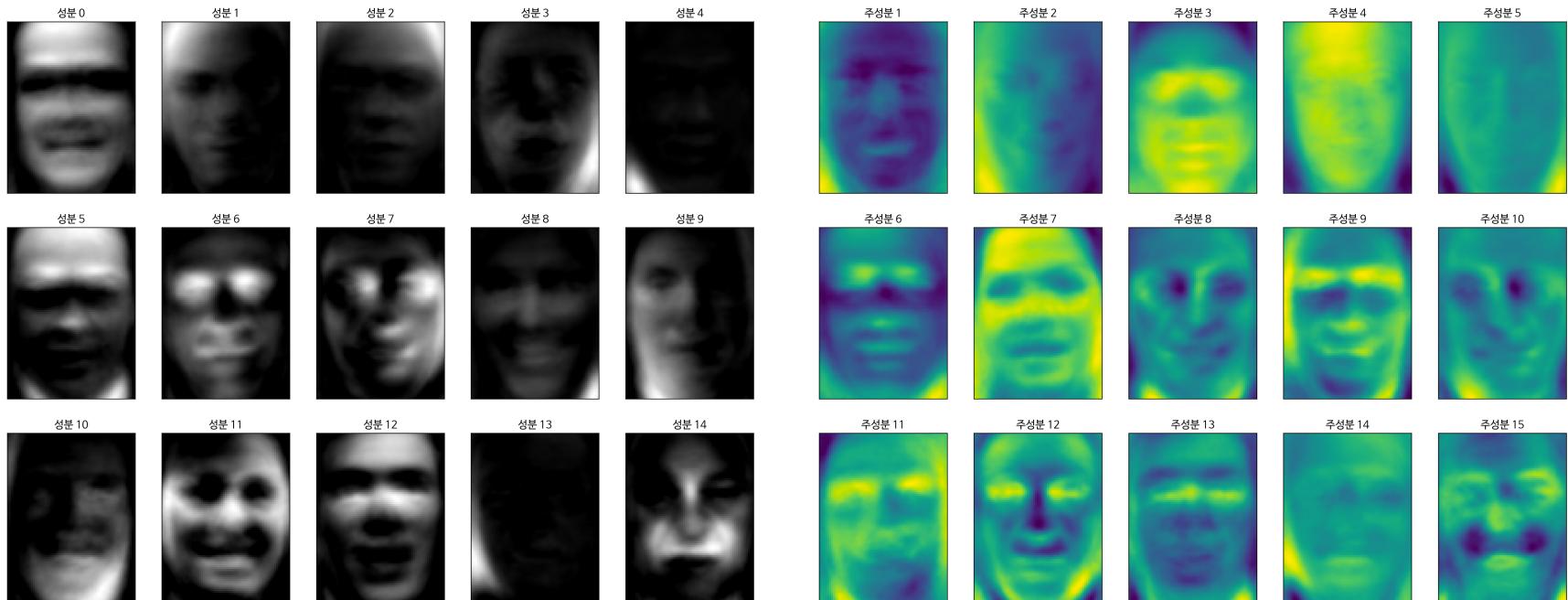
NMF

<

PCA

NMF는 재구성 보다는 주로 데이터에 있는 패턴을 분할하는데 주로 사용합니다.

# NMF - Ifw의 성분



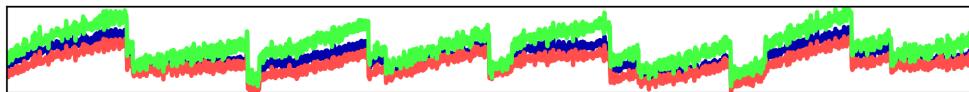
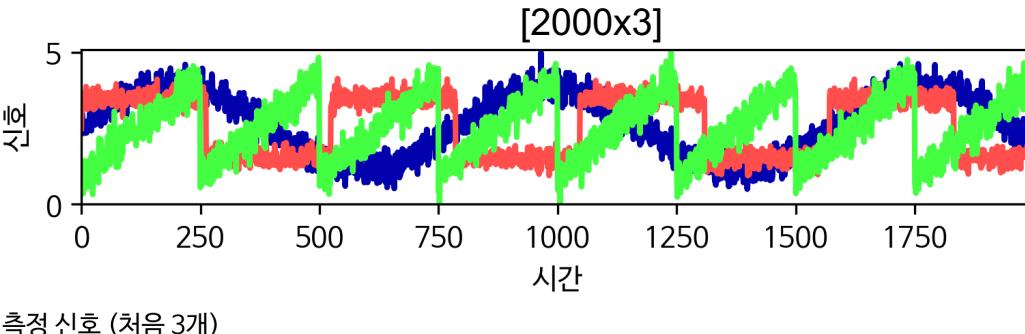
NMF

PCA

성분이 모두 양수이므로 얼굴 이미지와 가까운 형태를 띕니다.

# NMF + signal data

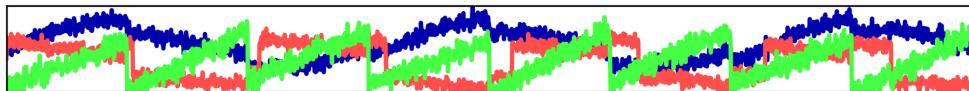
난수 행렬을 곱해서  
[2000x100]로 만듭니다



In [42]:

```
nmf = NMF(n_components=3, random_state=42)
S_ = nmf.fit_transform(X)
```

NMF로 복원한 신호

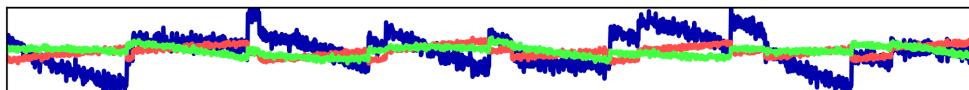


다시 [2000x3]으로 만듭니다.

In [43]:

```
pca = PCA(n_components=3)
H = pca.fit_transform(X)
```

PCA로 복원한 신호



PCA는 첫번째 성분이 대부분의 분산을 가져갑니다.

# t-SNE

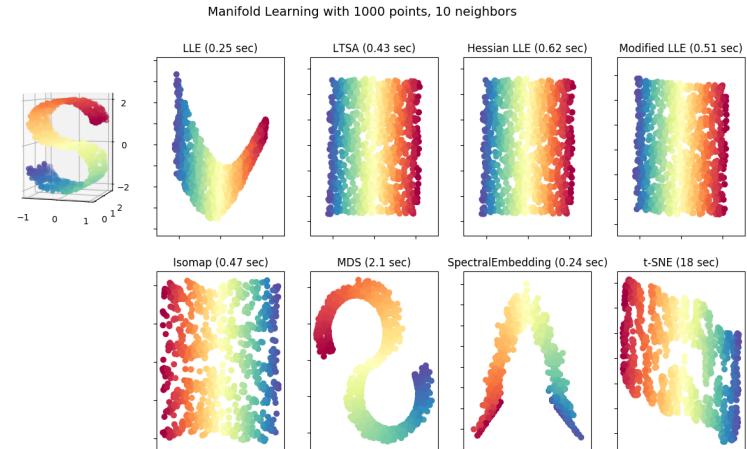
# t-SNE t-Distributed Stochastic Neighbor Embedding

PCA는 태생상 데이터를 회전하고 방향을 제거하기 때문에 시각화에 성능이 좋지 않습니다.

매니폴드 학습 manifold learning은 비선형 차원 축소의 기법으로 시각화에 뛰어난 성능을 발휘합니다(3개 이상 특성을 뽑지 않음)

탐색적 데이터 분석에 사용(transform()) 메서드가 없고 새로운 데이터에 적용하지 못합니다.  
fit\_transform()만 있습니다.)

2차원 평면에 데이터를 퍼뜨린 후 원본 특성에서의 거리를 잘 보존하는 2차원 표현을 학습합니다.



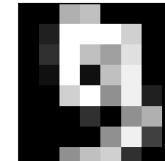
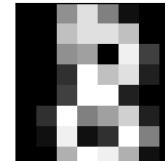
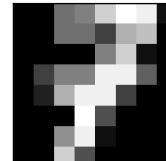
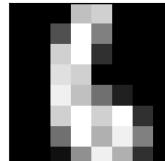
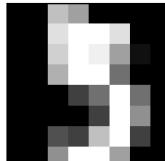
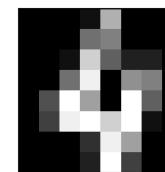
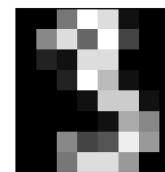
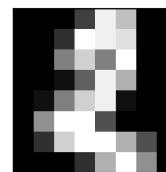
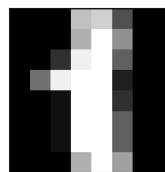
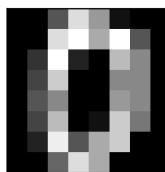
# load\_digits

8x8 흑백 이미지의 손글씨 숫자 데이터셋

(MNIST 데이터셋 아님. UC 얼바인 머신러닝 저장소:

<http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>)

```
In [45]: from sklearn.datasets import load_digits  
digits = load_digits()
```

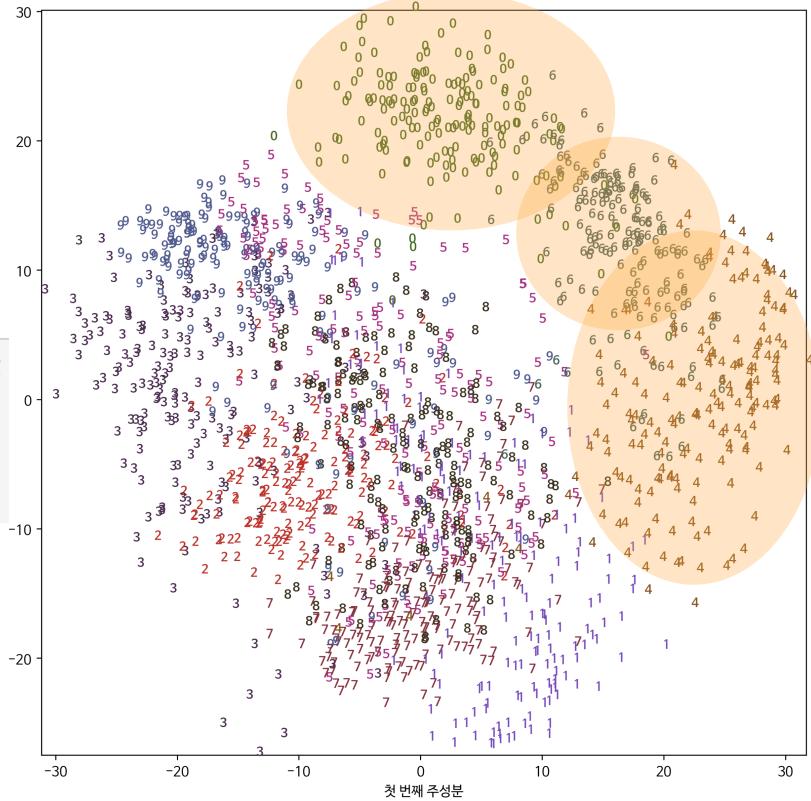


# PCA + load\_digits

PCA 주성분 2개를 구해 산점도를 그립니다.

대부분 많이 겹쳐 있습니다.

```
In [46]: # PCA 모델을 생성합니다  
pca = PCA(n_components=2)  
pca.fit(digits.data)  
# 처음 두 개의 주성분으로 숫자 데이터를 변환합니다  
digits_pca = pca.transform(digits.data)
```



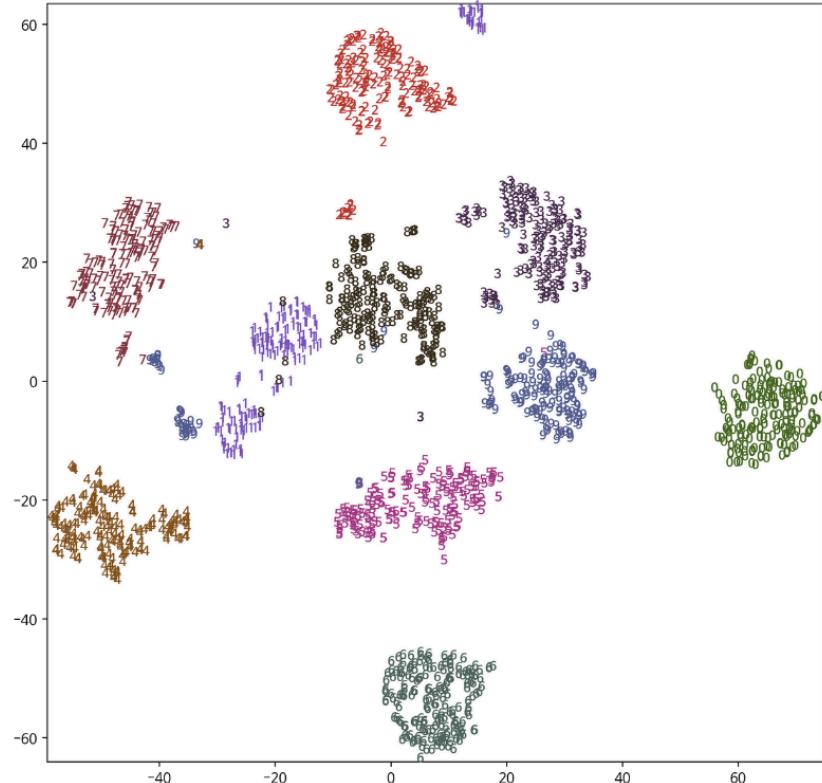
# TSNE + load\_digits

뛰어난 클래스 시각화 능력을 발휘합니다.

perplexity(5~50, default 30)가 크면 이웃을 많이 포함시킵니다(큰 데이터셋)

early\_exaggeration(default 4)에서 초기 과장 단계의 정도를 지정합니다. 클수록 초기간격이 넓어지지만 기본값에서도 대부분 잘 작동합니다.

```
In [47]: from sklearn.manifold import TSNE
tsne = TSNE(random_state=42)
# TSNE에는 transform 메소드가 있으므로 대신 fit_transform 사용
digits_tsne = tsne.fit_transform(digits.data)
```



$k$ -평균 군집

# 군집

데이터셋을 클러스터<sup>cluster</sup>라는 그룹으로 나누는 작업입니다.

한 클러스터 안의 데이터는 매우 비슷하고 다른 클러스터와는 구분되도록 만듭니다.

분류 알고리즘처럼 테스트 데이터에 대해서 어느 클러스터에 속할지 예측을 만들 수 있습니다.

- k-평균 군집
- 병합 군집
- DBSCAN

## k-평균means 군집

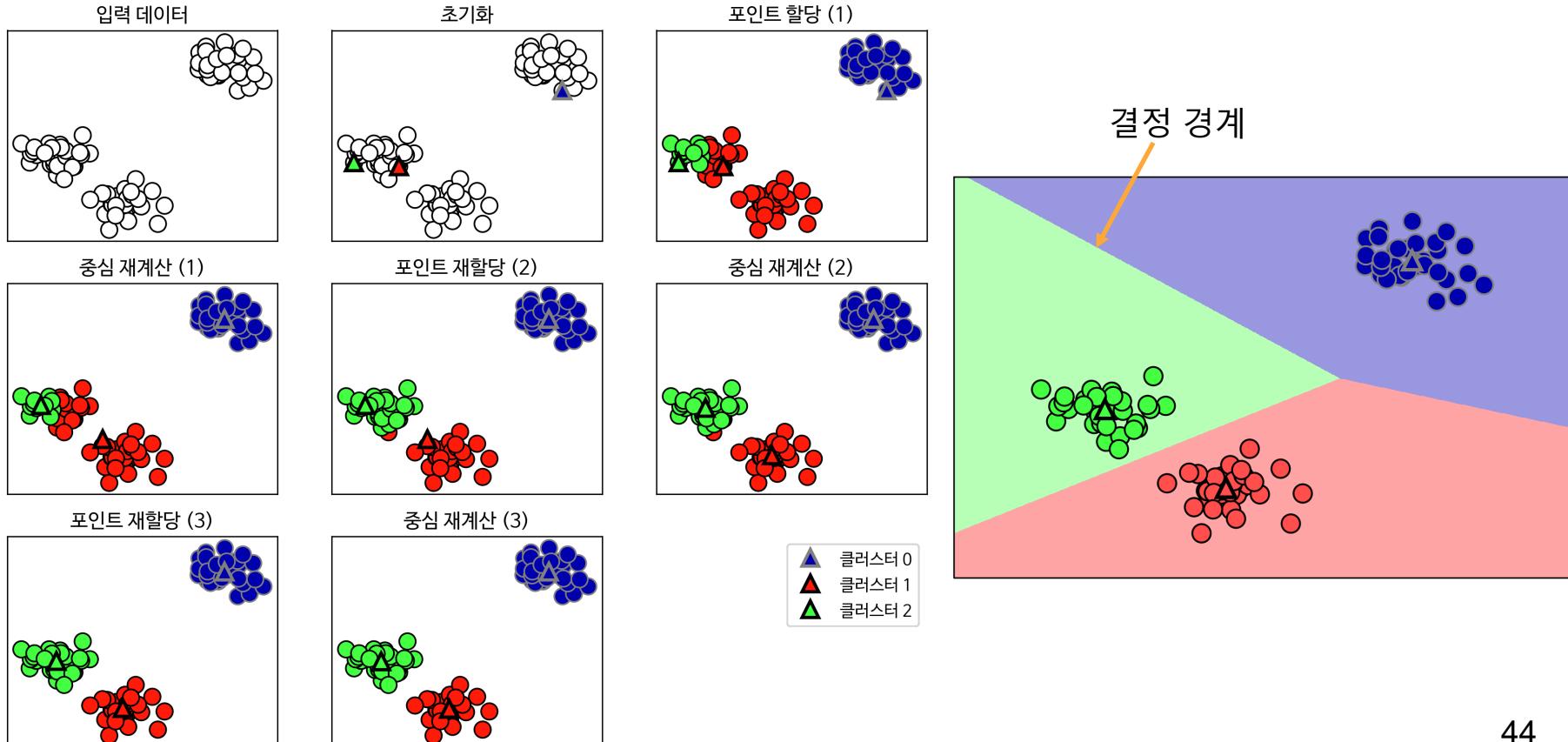
가장 간단하고 널리 사용되는 군집 알고리즘입니다.

임의의 클러스터 중심에 데이터 포인트를 할당합니다.

그 다음 클러스터안의 데이터 포인트를 평균을 내어 다시 클러스터 중심을 계산하고 이전 과정을 반복합니다.

클러스터에 할당되는 데이터 포인트의 변화가 없으면 알고리즘 종료됩니다.

# k-평균 example



# KMeans

```
In [51]: from sklearn.datasets import make_blobs  
from sklearn.cluster import KMeans
```

# 인위적으로 2차원 데이터를 생성합니다

```
X, y = make_blobs(random_state=1)
```

# 군집 모델을 만듭니다

```
kmeans = KMeans(n_clusters=3)
```

```
kmeans.fit(X)
```

기본값 8

비지도 학습이므로 타깃(y)을 넣지 않습니다.

레이블에 어떤 의미가 없으며  
순서가 무작위임(분류와 다른점)  
직접 데이터를 확인해야 레이블의  
의미를 알 수 있습니다.

```
In [52]: print(kmeans.labels_)
```

n\_clusters=3 이므로  
0~2까지 할당

```
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 2 0 2 0 1 2 0 0 1 1 0 1 1 0 1 2 0 2  
2 2 0 0 2 1 2 2 0 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 0 1 0 1 2 2 2 0 1  
1 2 0 0 1 2 1 2 2 0 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1 0 1 1]
```

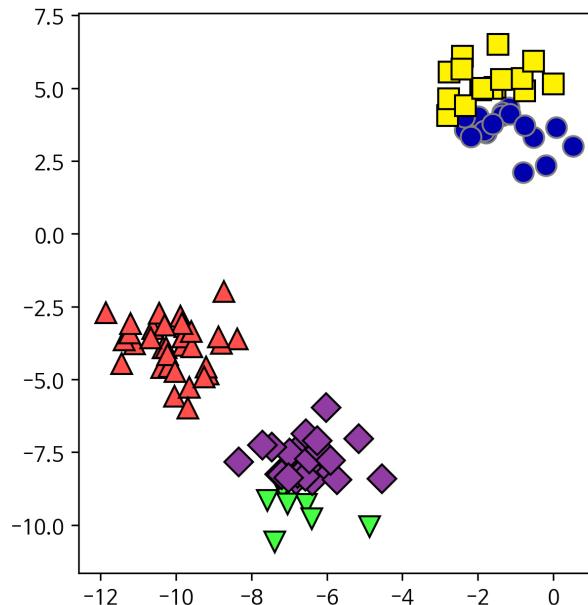
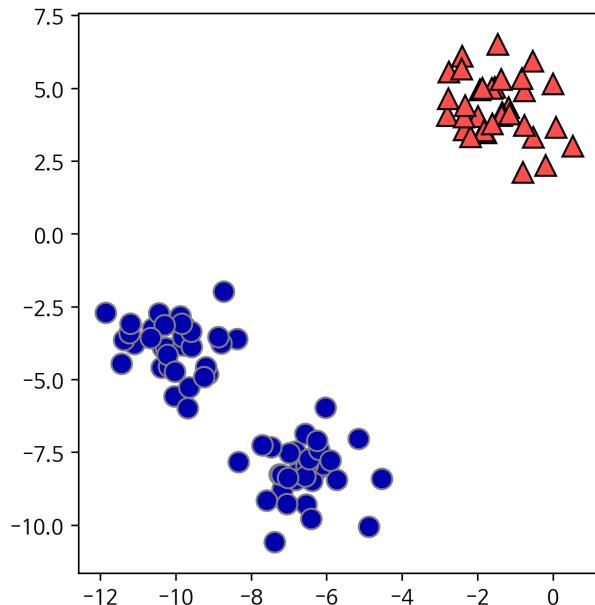
```
In [53]: print(kmeans.predict(X))
```

```
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 2 0 2 0 1 2 0 0 1 1 0 1 1 0 1 2 0 2  
2 2 0 0 2 1 2 2 0 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 2 0 1 0 1 2 2 2 0 1  
1 2 0 0 1 2 1 2 2 0 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1 0 1 1]
```

# n\_clusters=2 or 5

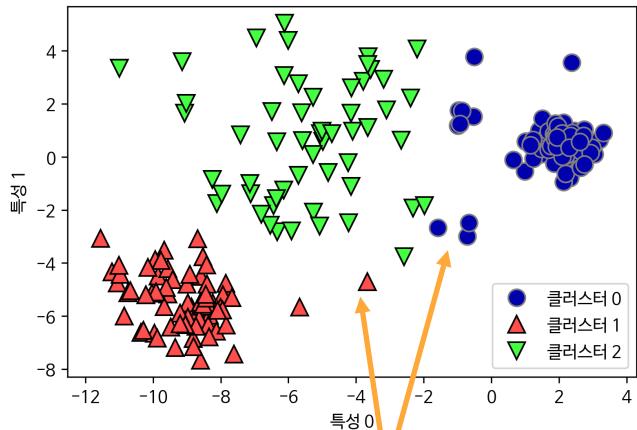
`n_clusters` 매개변수에 따라 군집이 크게 좌우됩니다.

따라서 클러스터 개수(하이퍼파라미터)를 잘 지정해야만 합니다.

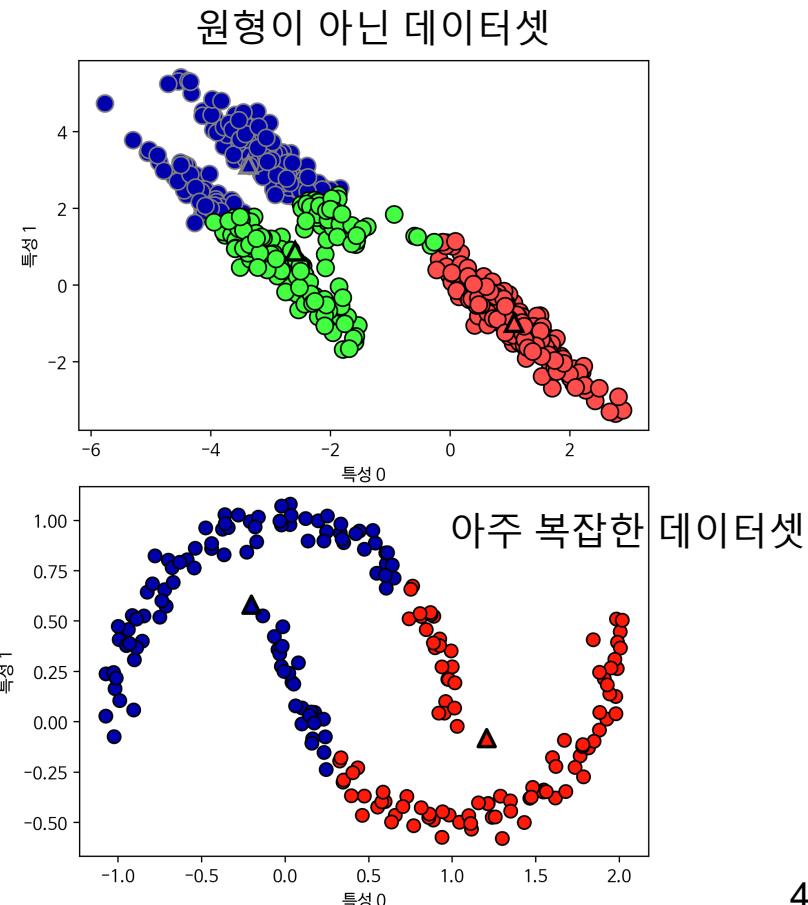


# k-평균의 단점

모든 클러스터가 원형이고  
반경이 동일하다고 간주하므로  
복잡한 형태를 구분하지 못합니다.

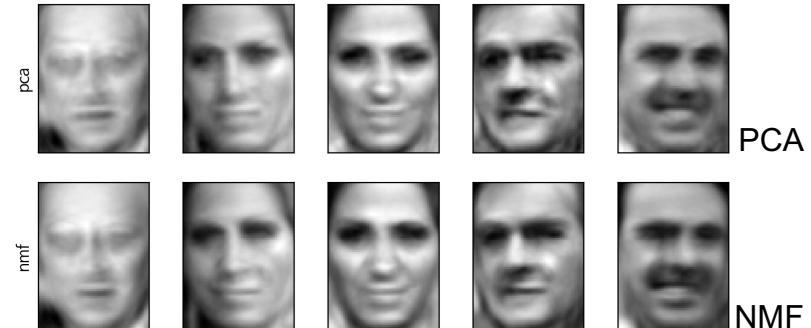
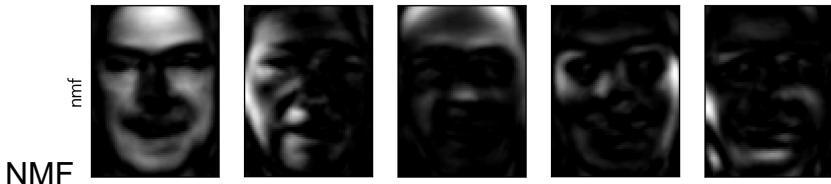
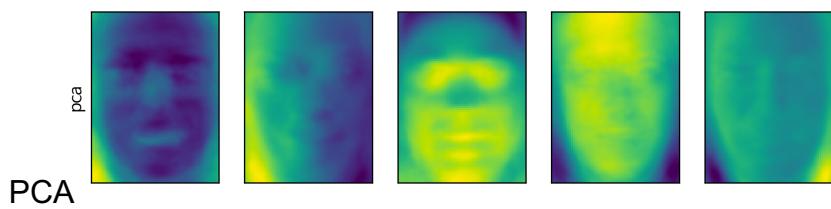


중심에서 멀리 떨어진  
포인트까지 포함합니다



# 벡터 양자화

PCA, NMF가 데이터 포인트를 성분의 합으로 표현할 수 있는 것처럼 k-평균은 데이터 포인트를 하나의 성분으로 나타내는 것으로 볼 수 있음(벡터 양자화)

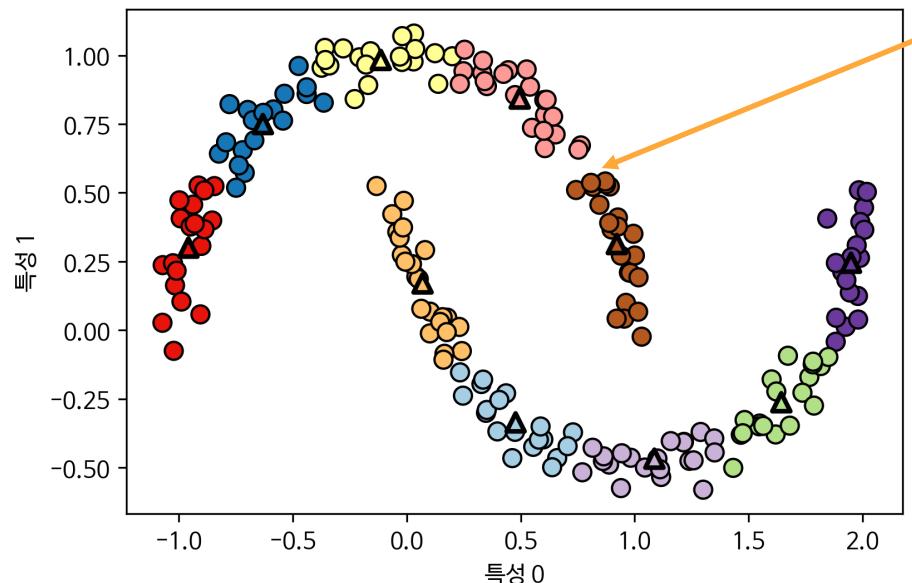


# 차원 확대

2차원 데이터에는 PCA와 NMF가 할 수 있는 것이 없지만 Kmeans는 특성 보다 더 많은 클러스터를 할당할 수 있습니다.

```
In [61]: X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
```

```
kmeans = KMeans(n_clusters=10, random_state=0)  
kmeans.fit(X)
```



두개의 특성에서  
10개의 특성으로 늘어남  
(해당 클러스터를 제외하고는 0)  
e.g. [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

# KMeans.transform()

각 데이터 포인트에서 모든 클러스터까지의 거리를 반환합니다.

10개의 새로운 특성으로 활용 가능합니다.

```
In [62]: distance_features = kmeans.transform(X)
print("클러스터 거리 데이터의 형태: {}".format(distance_features.shape))
print("클러스터 거리:\n{}".format(distance_features))
```

클러스터 거리 데이터의 형태: (200, 10)

클러스터 거리:

```
[[ 0.922  1.466  1.14 ...,  1.166  1.039  0.233]
 [ 1.142  2.517  0.12 ...,  0.707  2.204  0.983]
 [ 0.788  0.774  1.749 ...,  1.971  0.716  0.944]
 ...,
 [ 0.446  1.106  1.49 ...,  1.791  1.032  0.812]
 [ 1.39   0.798  1.981 ...,  1.978  0.239  1.058]
 [ 1.149  2.454  0.045 ...,  0.572  2.113  0.882]]
```

# 장단점

## 장점

이해와 구현이 쉽고 비교적 빨라서 인기가 높습니다.

대규모 데이터셋에 적용 가능합니다(MiniBatchKMeans)

## 단점

batch\_size=100 기본값

무작위로 초기화하므로 난수에 따라 결과가 달라집니다.

n\_init(default 10) 매개변수만큼 반복하여 클러스터 분산이 작은 결과를 선택합니다

클러스터 모양을 원형으로 가정하고 있어 제한적입니다.

실제로는 알 수 없는 클러스터 개수를 지정해야 합니다.

# 병합 군집

# 병합 군집 agglomerative clustering

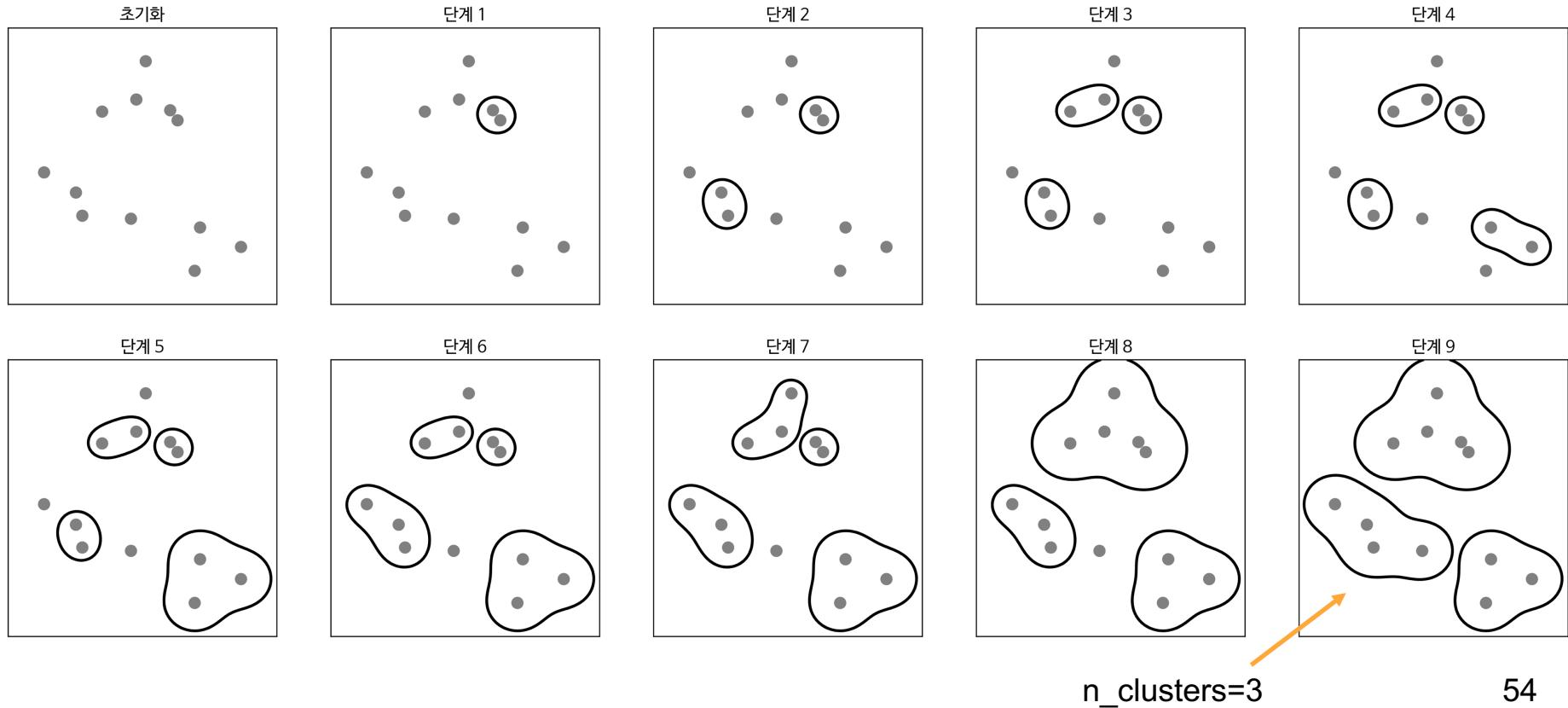
시작할 때 데이터 포인트를 하나의 클러스터로 지정해 지정된 클러스터 개수에 도달할 때까지 두 클러스터를 합쳐 나갑니다.

비슷한 클러스터를 선택하는 방법을 linkage 매개변수에 지정합니다.

- ward: 기본값, 클러스터 내의 분산을 가장 작게 만드는 두 클러스터를 병합(클러스터의 크기가 비슷해짐)
- average: 클러스터 포인트 사이의 평균 거리가 가장 작은 두 클러스터를 병합
- complete: 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 병합

클러스터 크기가 매우 다를 때 average나 complete가 나을 수 있습니다.

# 병합 군집 example



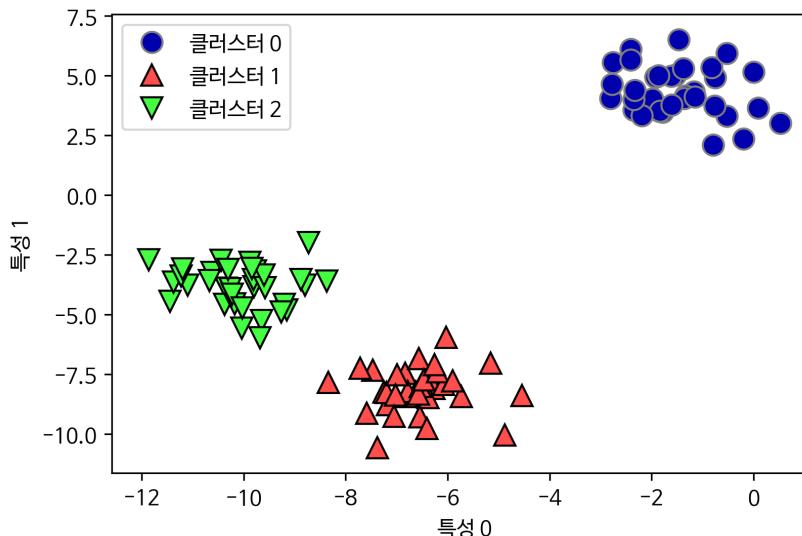
# AgglomerativeClustering

Kmeans와 다르게 새로운 데이터에 적용할 수 없습니다(fit\_predict 만 있음)

```
In [64]: from sklearn.cluster import AgglomerativeClustering  
X, y = make_blobs(random_state=1)
```

```
agg = AgglomerativeClustering(n_clusters=3)  
assignment = agg.fit_predict(X)
```

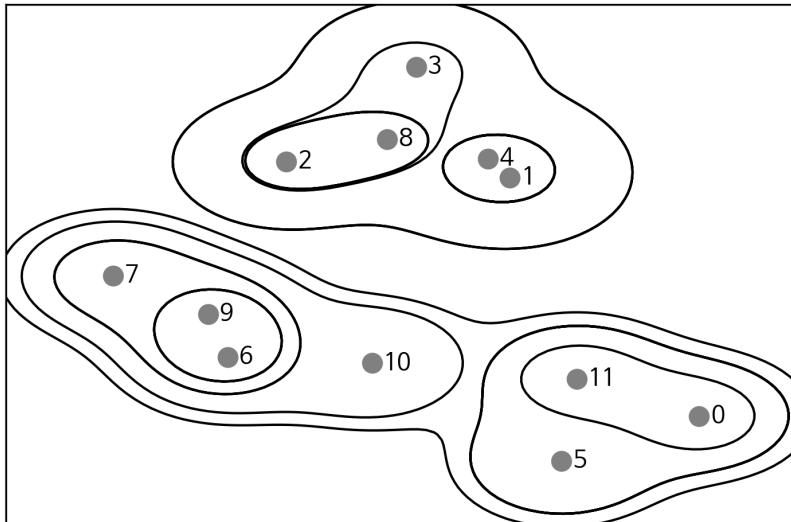
agg.labels\_ 을 반환



# 계층적 군집 hierarchical clustering

병합 군집을 사용하려면 Kmeans와 같이 클러스터의 개수를 지정해야 합니다.

하지만 병합 군집은 **계층적 군집**이므로 클러스터가 병합되는 과정을 살피면 클러스터 개수를 선택하는 데 도움을 얻을 수 있습니다(2차원일 경우에)



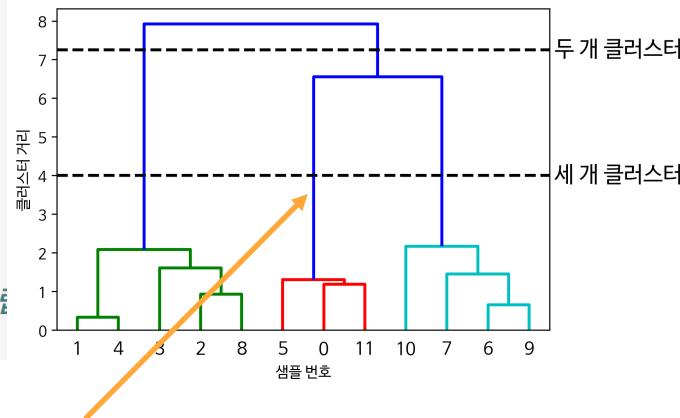
# 덴드로그램 dendrogram

2차원 이상의 계층적 군집을 표현할 수 있습니다.

scipy의 연결 linkage 함수와 덴드로그램 함수를 사용합니다.

```
In [66]: # SciPy에서 ward 구집 함수와 덴드로그램 함수를 임포트합니다
from scipy.cluster.hierarchy import dendrogram, ward

X, y = make_blobs(random_state=0, n_samples=12)
# 데이터 배열 X에 ward 함수를 적용합니다
# SciPy의 ward 함수는 병합 군집을 수행할 때 생성된
# 거리 정보가 담긴 배열을 리턴합니다
linkage_array = ward(X)
# 클러스터 간의 거리 정보가 담긴 linkage_array를 사용해 덴드로그램을 그림
dendrogram(linkage_array)
```



# DBSCAN

# DBSCAN 특징

클러스터 개수를 미리 지정할 필요 없습니다.

복잡한 형상에 적용 가능하고, 어느 클러스터에도 속하지 않는 노이즈 포인트를 구분합니다.

k-평균이나 병합 군집보다는 다소 느리지만 큰 데이터셋에 적용 가능합니다.

데이터가 많은 밀집 지역을 찾아 다른 지역과 구분하는 클러스터를 구성합니다.

밀집 지역의 포인트를 핵심 샘플이라고 부릅니다.

한 데이터 포인트에서  $\text{eps}$  안의 거리에  $\text{min\_samples}$  개수만큼 데이터가 있으면 이 포인트를 핵심 샘플로 분류함. → 클러스터가 됨

# DBSCAN 알고리즘

처음 무작위로 포인트를 선택하고  $\text{eps}$  거리안의 포인트를 찾습니다.

$\text{eps}$  거리안의 포인트 <  $\text{min\_samples}$  이면 잡음 포인트(-1)로 분류합니다.

$\text{eps}$  거리안의 포인트 >  $\text{min\_samples}$  이면 핵심 포인트로 분류, 새 클러스터 할당

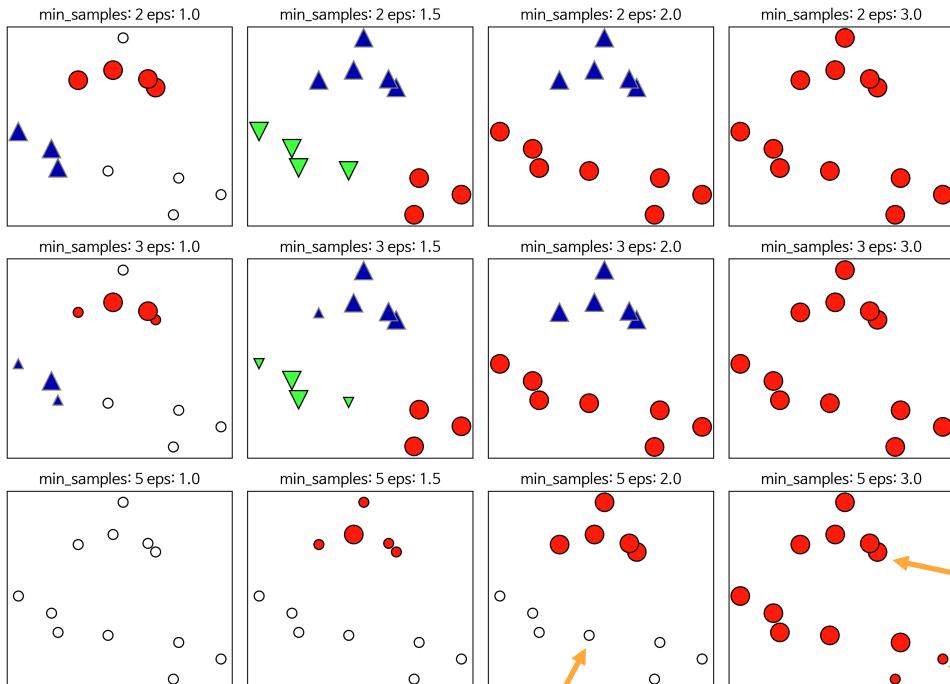
핵심 포인트에서  $\text{eps}$  안의 포인트를 확인하여 클러스터 할당이 없으면 핵심 샘플의 클러스터를 할당합니다(경계 포인트)

$\text{eps}$  안의 포인트가 이미 핵심 샘플이면 그 포인트의 이웃을 또 차례로 방문하여 클러스터가 자라납니다. 그리고 아직 방문하지 못한 포인트를 다시 선택해 동일한 과정을 반복합니다.

여러번 실행해도 같은 핵심 포인트, 잡음 포인트를 찾지만 경계 포인트는 바뀔 수 있습니다. 보통 경계 포인트는 많지 않아 크게 영향을 미치지 않습니다.

# DBSCAN + 인공 데이터셋

클러스터 증가, eps 증가



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

```
dbscan = DBSCAN()  
clusters = dbscan.fit_predict(X)
```

클러스터 레이블:

`[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]`

`min_samples=5  
eps=0.5`

잡음 포인트 감소,  
`min_samples` 감소

핵심 포인트

경계 포인트

잡음 포인트

# eps, min\_samples

eps가 가까운 포인트의 범위를 정하기 때문에 더 중요합니다.

eps가 아주 작으면 핵심 포인트가 생성되지 않고 모두 잡음으로 분류됩니다. eps를 아주 크게하면 모든 포인트가 하나의 클러스터가 됩니다.

eps로 클러스터의 개수를 간접적으로 조정할 수 있습니다.

min\_samples는 덜 조밀한 지역이 잡음 포인트가 될지를 결정합니다.

min\_samples 보다 작은 개수가 모여 있는 지역은 잡음 포인트가 됩니다.(클러스터 최소 크기)

StandardScaler나 MinMaxScaler로 특성의 스케일을 조정할 필요 있습니다.

잡음 포인트의 레이블이 -1이므로 군집의 결과를 다른 배열의 인덱스로 사용할 때는 주의해야 합니다.

# DBSCAN + Two Moons

```
In [69]: X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
```

# 평균이 0, 분산이 1이 되도록 데이터의 스케일을 조정합니다

```
scaler = StandardScaler()
```

```
scaler.fit(X)
```

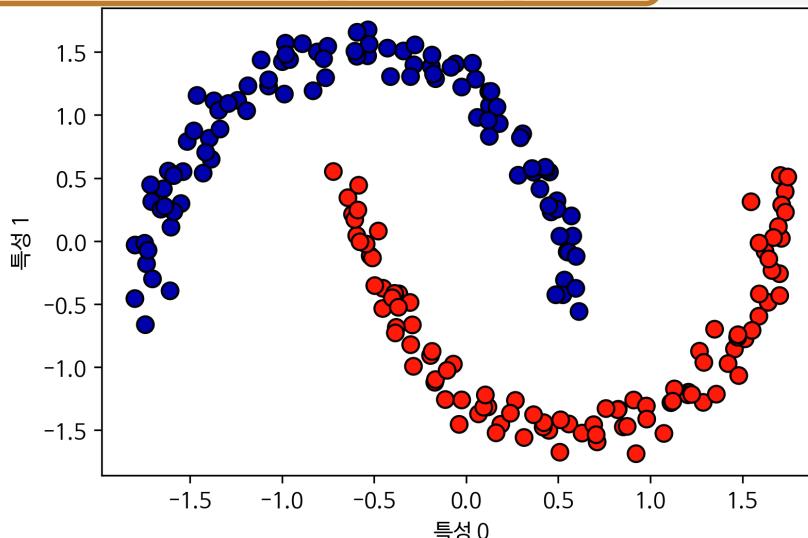
```
X_scaled = scaler.transform(X)
```

평균 0, 분산 1

```
dbSCAN = DBSCAN()
```

```
clusters = dbSCAN.fit_predict(X_scaled)
```

다른 데이터에 적용하지 못함



군집 알고리즘 비교, 평가

# ARI, NMI

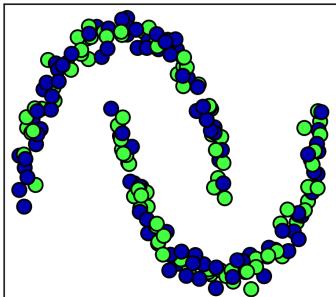
실제 정답 클래스(타깃)과 비교해야 합니다.

0(무작위)~1(최적) 사이의 값, ARI의 최저값은 -0.5 혹은 -1 임

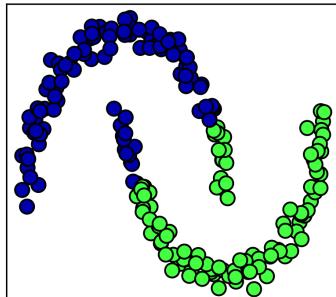
```
In [70]: from sklearn.metrics.cluster import adjusted_rand_score  
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)  
  
# 평균이 0, 분산이 1이 되도록 데이터의 스케일을 조정합니다  
scaler = StandardScaler()  
scaler.fit(X)  
X_scaled = scaler.transform(X)  
  
clusters = algorithm.fit_predict(X_scaled)  
adjusted rand score(y, clusters)))
```

NMI: normalized\_mutual\_info\_score()

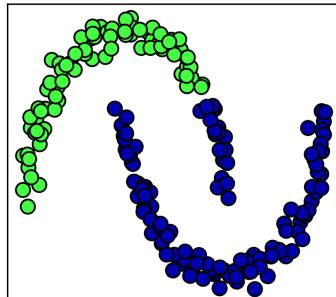
무작위 할당 - ARI: 0.00



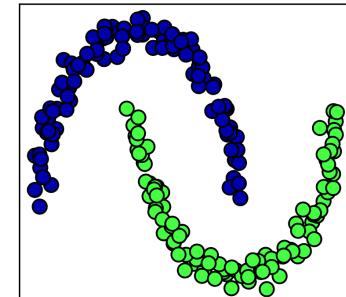
KMeans - ARI: 0.50



AgglomerativeClustering - ARI: 0.61



DBSCAN - ARI: 1.00



# accuracy?

군집의 클래스 레이블은 의미가 없으므로 정확도를 사용하면 클래스 레이블이 같은지를 확인하기 때문에 잘못된 평가가 됩니다.

```
In [71]: from sklearn.metrics import accuracy_score  
  
# 포인트가 클러스터로 나뉜 두 가지 경우  
clusters1 = [0, 0, 1, 1, 0]  
clusters2 = [1, 1, 0, 0, 1]  
# 모든 레이블이 달라졌으므로 정확도는 0입니다  
print("정확도: {:.2f}".format(accuracy_score(clusters1, clusters2)))  
# 같은 포인트가 클러스터에 모였으므로 ARI는 1입니다  
print("ARI: {:.2f}".format(adjusted_rand_score(clusters1, clusters2)))
```

정확도: 0.00

ARI: 1.00

# 실루엣 silhouette 계수

ARI, NMI는 타깃값이 있어야 가능하므로 애플리케이션 보다 알고리즘을 개발할 때 도움이 됩니다.

타깃값 필요없이 클러스터의 밀집 정도를 평가합니다.

-1: 잘못된 군집, 0: 중첩된 군집, 1: 가장 좋은 군집

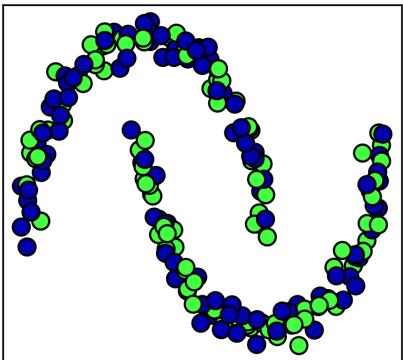
모양이 복잡할 때는 밀집 정도를 평가하는 것이 잘 맞지 않습니다.

원형 클러스터의 실루엣 점수가 높게 나옵니다.

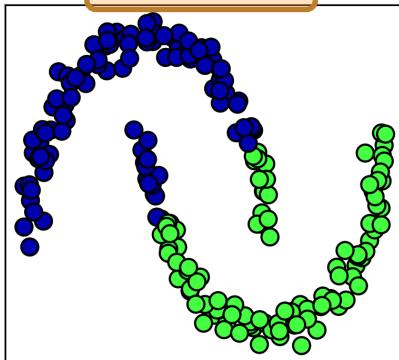
# silhouette\_score

```
In [72]: from sklearn.metrics.cluster import silhouette_score  
silhouette_score(X_scaled, clusters)
```

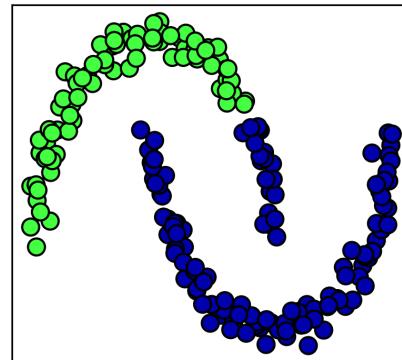
무작위 할당: -0.00



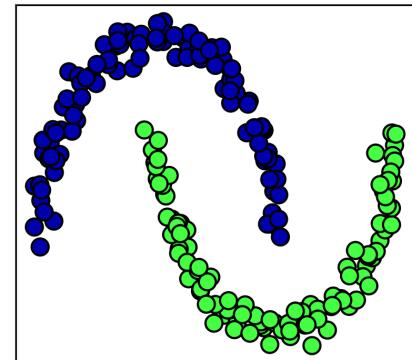
KMeans : 0.49



AgglomerativeClustering : 0.46



DBSCAN : 0.38



# 군집 평가의 어려움

실루엣 점수가 높다하더라도 찾아낸 군집이 흥미로운 것인지는 여전히 알 수 없습니다.

사진 애플리케이션이 두 개의 클러스터를 만들었다면

앞모습 vs 옆모습

밤사진 vs 낮사진

아이폰 vs 안드로이드

어떻게 만들었을지 알 수 없습니다.

클러스터가 기대한 대로인지는 직접 확인해야만 알 수 있습니다.

군집 + LFW

# DBSCAN

2063x100

In [73]:

```
# LFW 데이터에서 고유얼굴을 찾은 다음 데이터를 변환합니다
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)
```

In [74]:

```
# 기본 매개변수로 DBSCAN을 적용합니다
dbscan = DBSCAN()
labels = dbscan.fit_predict(X_pca)
print("고유한 레이블: {}".format(np.unique(labels)))
```

min\_samples=5

고유한 레이블: [-1] 잡음 포인트

In [75]:

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

고유한 레이블: [-1]

클러스터 범위 증가

In [76]:

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

고유한 레이블: [-1 0]

# DBSCAN 잡음 포인트

```
In [76]: dbSCAN = DBSCAN(min_samples=3, eps=15)  
labels = dbSCAN.fit_predict(X_pca)  
print("고유한 레이블: {}".format(np.unique(labels)))
```

고유한 레이블: [-1 0]



이상치 검출 용도로 활용할 수 있습니다.

# LFW의 클러스터

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 ]

대다수의 얼굴이미지는 비슷하거나 비슷하지 않음

# k-평균 + LFW

In [81]:

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

원본 차원으로 복원:  
100x5655

```
pca.inverse_transform(km.cluster_centers_)
```

2063x100

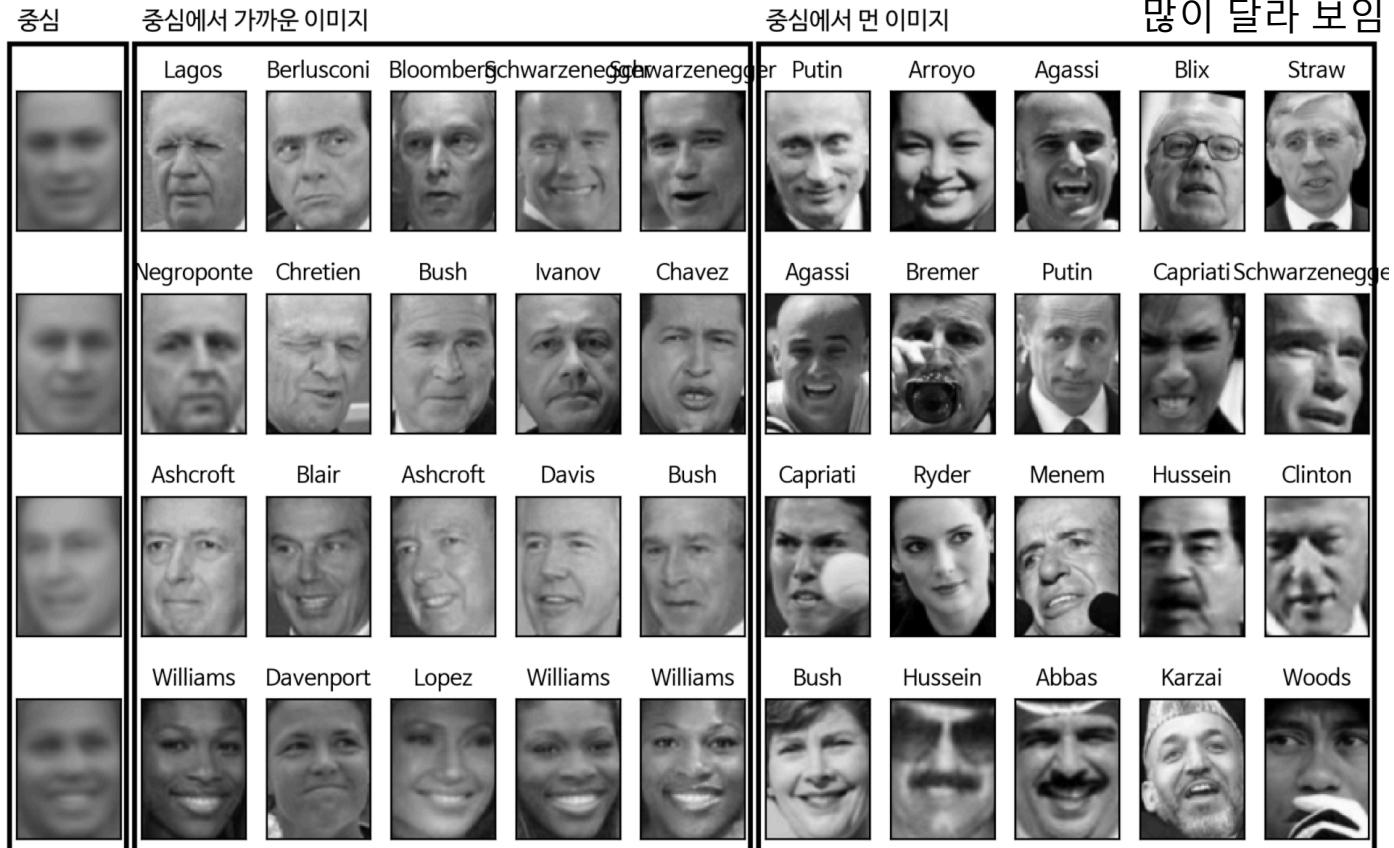
클러스터 중심:  
10x100



클러스터에 있는 얼굴의 평균이므로 매우 부드럽습니다

# k-평균의 중심에서 가깝고 먼 이미지

중심에서 먼 이미지들은  
많이 달라 보임



# 병합 군집 + LFW

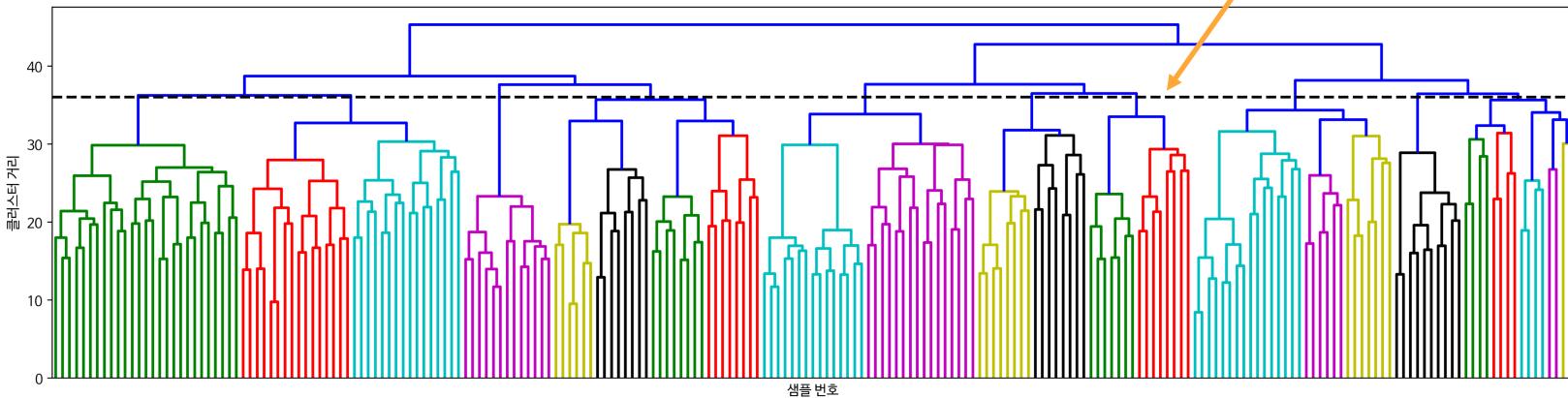
```
In [84]: # 병합 군집으로 클러스터를 추출합니다  
agglomerative = AgglomerativeClustering(n_clusters=10)  
labels_agg = agglomerative.fit_predict(X_pca)  
print("병합 군집의 클러스터 크기: {}".format(  
    np.bincount(labels_agg)))
```

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

```
In [85]: print("ARI: {:.2f}".format(adjusted_rand_score(labels_agg, labels_km)))
```

ARI: 0.09

자를 위치가 명확하지 않습니다



# 장단점

군집 알고리즘은 정성적 분석 과정이나 탐색적 분석 단계에 유용합니다.

k-평균, 병합군집: 클러스터 개수 지정

k-평균: 클러스터의 중심을 데이터 포인트의 분해 방법으로 볼 수 있음

DBSCAN: eps 매개변수로 간접적으로 클러스터 크기를 조정, 잡음 포인트 인식, 클러스터 개수 자동 인식, 복잡한 형상 파악 가능

병합 군집: 계층적 군집으로 덴드로그램을 사용해 표현할 수 있음