ML-C3

22.07.14

Unsupervised Learning(비지도학습)

알고있는 출력값이나 정보 없이 학습 알고리즘을 가르쳐야 하는 모든종류의 머신러닝을 가르킴.

->Unsupervised Transformation(비지도 변환) / Clustering(군집화)

비지도 변환 : 데이터를 새롭게 표현하여 사람이나 머신러닝 알고리즘이 원래 데이터보다 쉽게 해석할 수 있도록 만드는 알고리즘이다. 데이터를 특성의 수를 줄이면서 꼭 필요한 특징을 포함한 데이터로 표현하는 방법인 차원축소(dimesionality reduction)이다. Ex) 시각화를 위해 데이터셋을 2차원으로 변경하는 경우.

군집알고리즘 : 데이터를 비슷한 것 끼리 그룹으로 묶는 것.

Data_Preprocessing and Adjusting Scale

스케일 기법 : StandardScaler / MinmaxScaler / RobustScaler / Normalizer

StandardScaler는 각 특성의 평균을 0, 분산을 1로 변경하여, 모든특성이 같은 크기를 가지게 한다. -> 특성의 최솟값과 최댓값크기를 제한하지 않는다. (X-평균X)/ 표준편차

MinMaxScaler는 모든 특성이 정확하게 0과1사이에 위치하도록 데이터를 변경한다.

2차원데이터셋일 경우 모든데이터가 x축의 0과1, y축의 0과1사이 사각영역에 담긴다. (X-Xmin)/(Xmax –Xmin)

RobustScaler는 특성들이 같은 스케일을 갖게 된다는 통계적 측면에서 Standard와 비슷하지만, 평균과 분산 대신 중간값과 사분위 값을 사용한다. -> 전체데이터와 아주 동떨어진 데이터 포인트에 영향을 받지 않는다. -> Outlier의 영향을 받지 않는다.

(X-Q2)/(Q3-Q1) -> q2는 중간값 q1 1사분위값 , q3 3사분위값

Normalizer 다른 스케일 기법. 특성 벡터의 유클리디안 길이가 1이 되도록 데이터 포인트를 조정. 지름이 1인 원에 데이터포인트를 투영한다. 각 데이터포인트가 다른 비율로 스케일이 조정된다는 뜻입니다. 정규화는 특성 벡터의 길이는 상관없고 데이터의 방향만이 중요할 때 사용. -> l1 l2 max

Data_Preprocessing 예제(Cancer_dataset)->Minmax

```
from sklearn.preprocessing import MinMaxScaler
Xc_train, Xc_test, yc_train , yc_test =
train_test_split(cancer.data,cancer.target, random_state = 0)
Xm, ym = make_moons(n_samples =100,noise=0.25,random_state=3)
Xm_train, Xm_test , ym_train, ym_test =
train_test_split(Xm,ym,stratify=ym,random_state =0)
scaler = MinMaxScaler()
scaler.fit(Xc_train)

Xc_train_scaled = scaler.transform(Xc_train)
print(Xc_train_scaled.shape)
print(Xc_train.min(axis=0))
print(Xc_train_scaled.min(axis=0))
```

MinMaxScaler는 모든 특성이 정확하게 0과1사이에 위치하도록 데이터를 변경한다.
2차원데이터셋일 경우 모든데이터가 x축의 0과1, y축의 0과1사이 사각영역에 담긴다. ->cancer.fearture_names.shape : (30,)

(426, 30)

train [6.981e+00 9.710e+00 4.379e+01 1.435e+02 5.263e-02 1.938e-02 0.000e+00 0.000e+00 1.060e-01 4.996e-02 1.115e-01 3.628e-01 7.570e-01 7.228e+00 1.713e-03 2.252e-03 0.000e+00 0.000e+00 7.882e-03 8.948e-04 7.930e+00 1.202e+01 5.041e+01 1.852e+02 7.117e-02 2.729e-02 0.000e+00 0.000e+00 1.565e-01 5.504e-02]

transformed_train [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. 0. 0.]

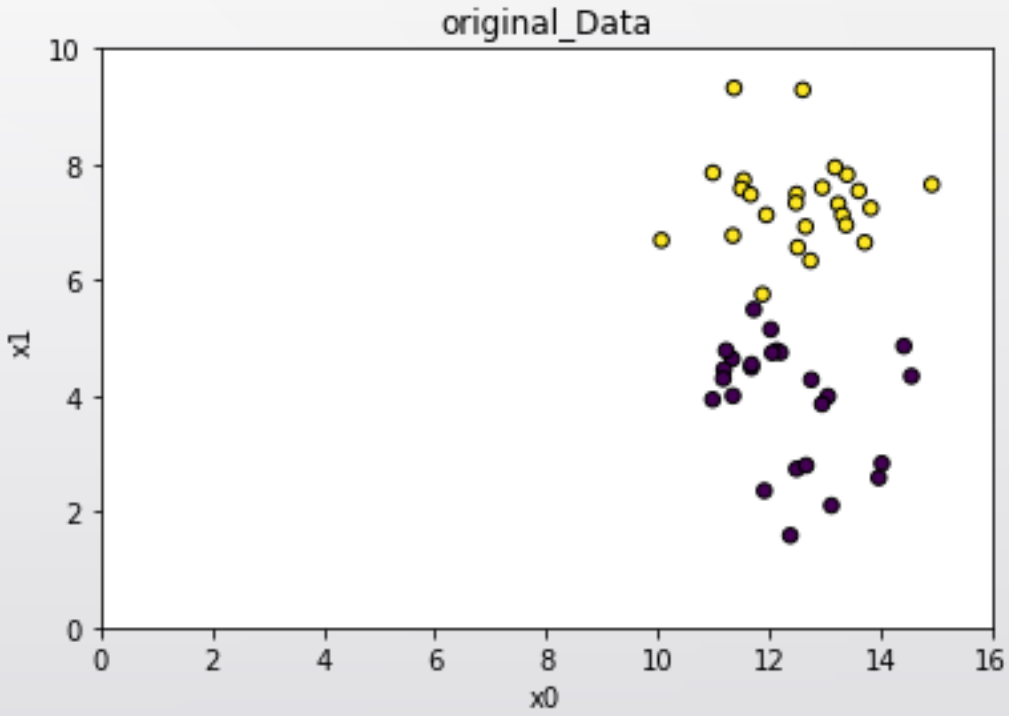
test [2.321e+01 3.928e+01 1.535e+02 1.670e+03 1.634e-01 3.454e-01 4.264e-01 1.823e-01 2.906e-01 9.502e-02 1.370e+00 3.647e+00 1.107e+01 1.765e+02 3.113e-02 1.354e-01 1.438e-01 4.090e-02 7.895e-02 2.193e-02 3.101e+01 4.487e+01 2.068e+02 2.944e+03 1.902e-01 9.327e-01 1.170e+00 2.910e-01 5.440e-01 1.446e-01]

transformed_test [0.76809125 1.22697095 0.75813696 0.64750795 1.20310633 1.11643038 0.99906279 0.90606362 0.93232323 0.94903117 0.45573058 0.72623944 0.48593507 0.31641282 1.36082713 1.2784499 0.36313131 0.77476795 1.32643996 0.72672498 0.82106012 0.87553305 0.77887345 0.67803775 0.78603975 0.87843331 0.93450479 1.0024113 0.76384782 0.58743277]

Data_Preprocessing QuantileTransformer

1000개 분위를 사용하여, 데이터를 균등하게 분포시킨다.
Outlier에 민감하지 않으며 전체 데이터를 0과 1사이에서
압축한다.

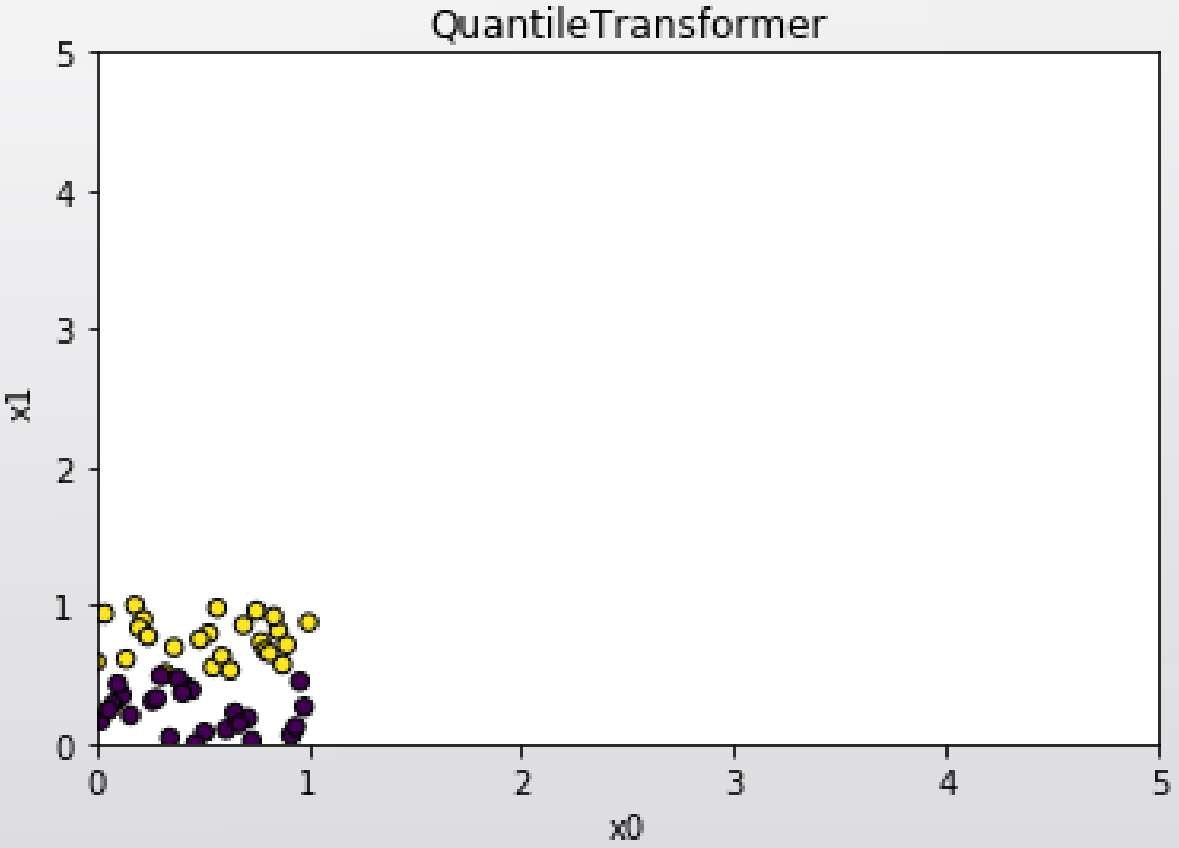
```
#원본데이터 산점도
X,y = make_blobs(n_samples=50, centers =2, random_state
=4,cluster_std=1)
X+=3

plt.scatter(X[:,0],X[:,1],c=y, s=30, edgecolors='black')
plt.xlim(0,16)
plt.ylim(0,10)
plt.xlabel('x0')
plt.ylabel('x1')
plt.title("original_Data")
```



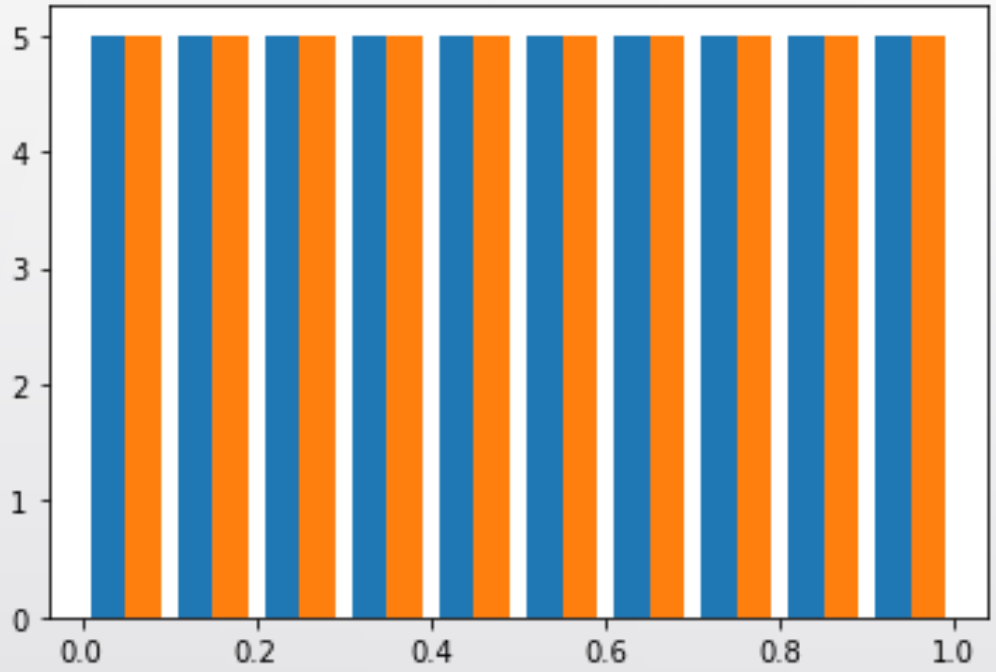
Data_Preprocessing QuantileTransformer

1000개 분위를 사용하여, 데이터를 균등하게 분포시킨다.
Outlier에 민감하지 않으며 전체 데이터를 0과 1사이에서
압축한다.

```
#QuantileTransformer 적용 데이터 산점도
X,y = make_blobs(n_samples=50, centers =2, random_state
=4,cluster_std=1)
X+=3

scaler = QuantileTransformer()
X_trans = scaler.fit_transform(X)

plt.scatter(X_trans[:,0],X_trans[:,1],c=y,s=30,edgecolors='black')
plt.xlim(0,5)
plt.xlabel('x0')
plt.ylim(0,5)
plt.ylabel('x1')
plt.title(type(scaler).__name__)
```



Data_Preprocessing QuantileTransformer



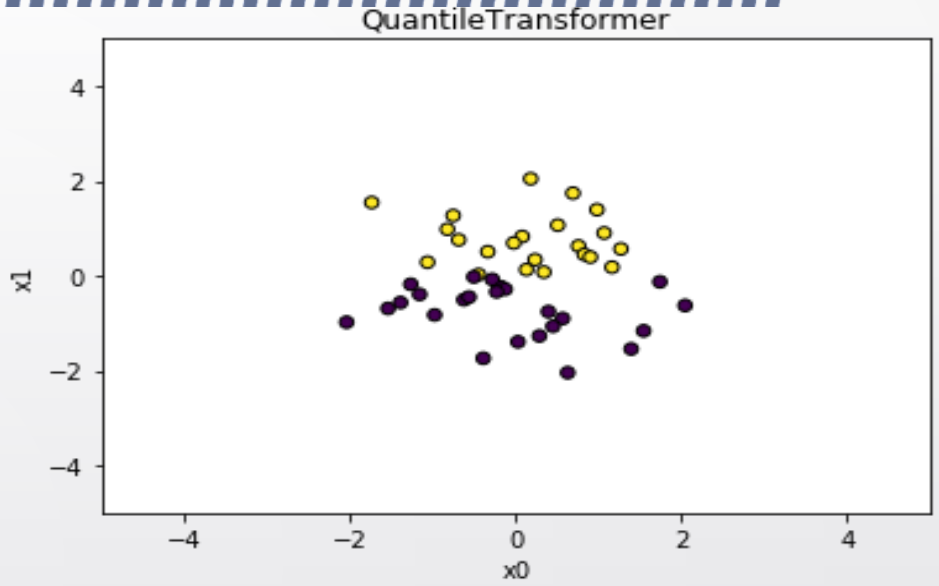

```
x = np.array([[1],[2],[3],[4],[5]])
print(np.percentile(x[:,0],[0,25,50,75,100]))
```

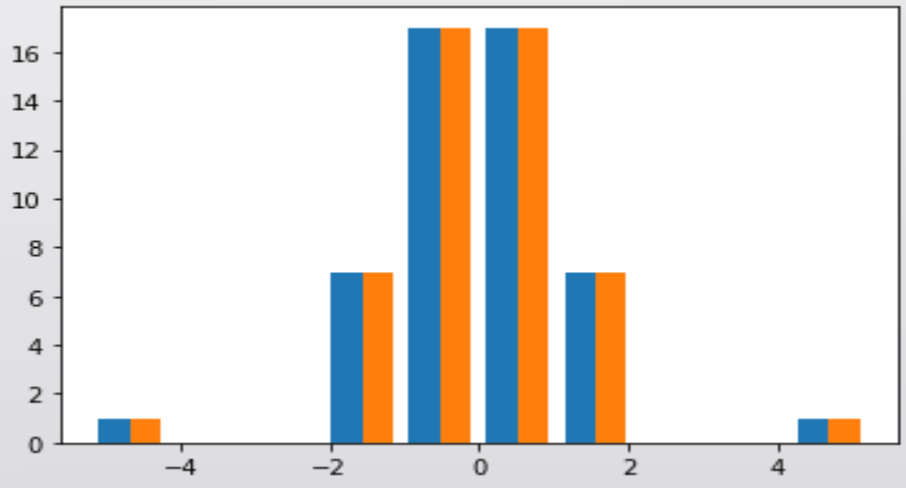
```
[1. 2. 3. 4. 5.]
```

기본값은 균등분포이다.

Data_Preprocessing QuantileTransformer

```
X,y = make_blobs(n_samples=50, centers =2, random_state
=4,cluster_std=1)
X+=3

scaler = QuantileTransformer(output_distribution='normal')
X_trans = scaler.fit_transform(X)

plt.scatter(X_trans[:,0],X_trans[:,1],c=y,s=30,edgecolors='black')
plt.xlim(-5,5)
plt.xlabel('x0')
plt.ylim(-5,5)
plt.ylabel('x1')
plt.title(type(scaler).__name__)
```

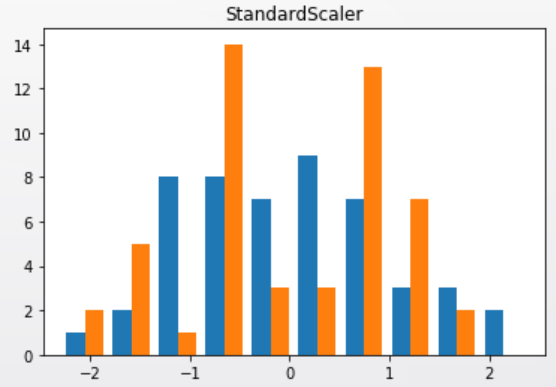


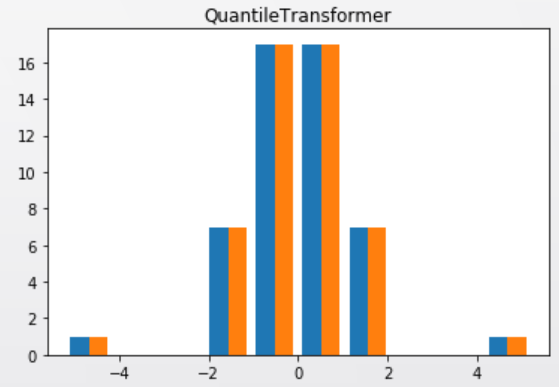


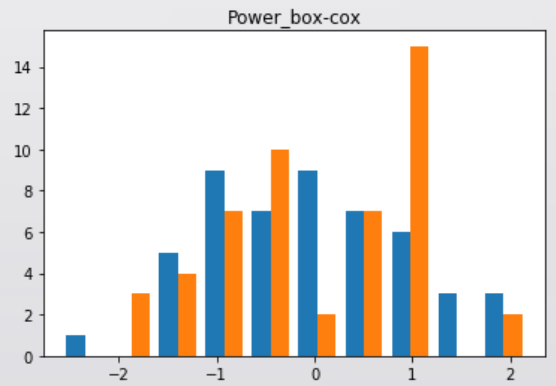
Output_distribution = ‘normal’로 하면 균등분포를 정규분포형태로 바꾸어준다.

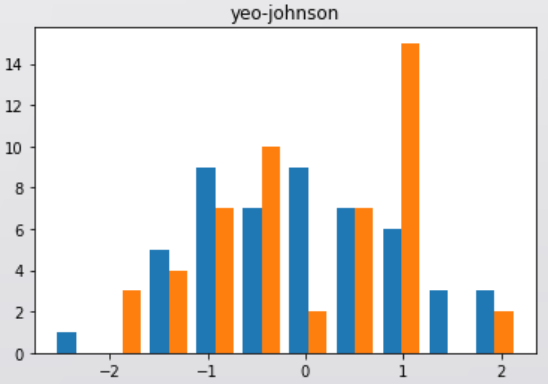
Data_Preprocessing Transformation Comparsion

```
from sklearn.preprocessing import MinMaxScaler, QuantileTransformer, StandardScaler,
PowerTransformer
X,y = make_blobs(n_samples=50, centers =2, random_state =4,cluster_std=1)
X+=3

scaler = QuantileTransformer(output_distribution='normal')
X_trans = scaler.fit_transform(X)

plt.hist(X_trans)
plt.title('QuantileTransformer')
plt.show() # figure를 초기화한다.
X_trans = StandardScaler().fit_transform(X)
plt.hist(X_trans)
plt.title('StandardScaler')
plt.show()
X_trans = PowerTransformer(method='box-cox').fit_transform(X)
plt.hist(X_trans)
plt.title('Power_box-cox')
plt.show()
X_trans = PowerTransformer(method='yeo-johnson').fit_transform(X)
plt.hist(X_trans)
plt.title('yeo-johnson')
plt.show()
```







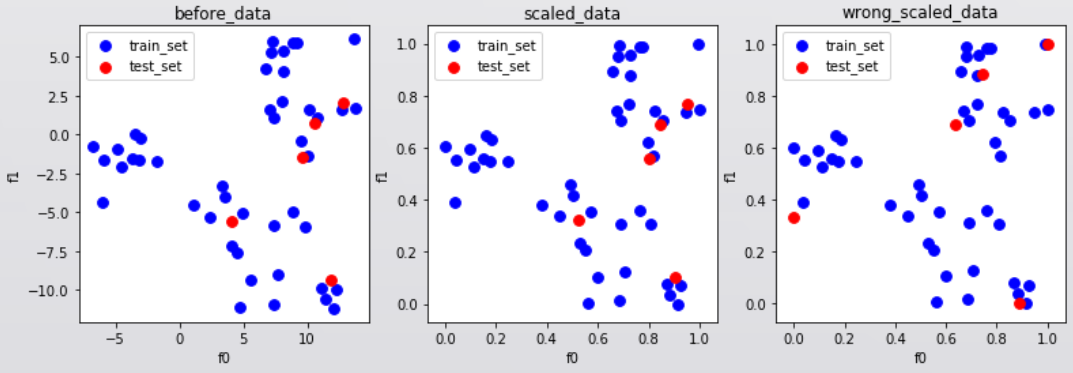


Data_Preprocessing Transformation Comparsion

```
from sklearn.datasets import make_blobs
X,_ = make_blobs(n_samples=50,centers=5, random_state=4, cluster_std=2)

X_train, X_test = train_test_split(X,random_state=5, test_size=.1) # train :9 / test : 1

fig, axes = plt.subplots(1,3,figsize=(13,4))
axes[0].scatter(X_train[:,0],X_train[:,1],c='blue',label="train_set",s=60)
axes[0].scatter(X_test[:,0],X_test[:,1],c='red',label="test_set",s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("before_data")
axes[0].set_xlabel("f0")
axes[0].set_ylabel("f1")

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Fit_transform()을 통해 fit과 transfor을 동시에 할 수 있지만,
Test_data를 fit_transfor을 하면 훈련세트의 학습 내용이 사라지기때문에,
Transform메소드를 통해 변환 시켜준다.

```
#scaled_scatter

axes[1].scatter(X_train_scaled[:,0],X_train_scaled[:,1],c='blue',label="train_set",s=60)
axes[1].scatter(X_test_scaled[:,0],X_test_scaled[:,1],c='red',label="test_set",s=60)
axes[1].legend(loc='upper left')
axes[1].set_title("scaled_data")
axes[1].set_xlabel("f0")
axes[1].set_ylabel("f1")

test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_wtest_scaled = test_scaler.transform(X_test)

axes[2].scatter(X_train_scaled[:,0],X_train_scaled[:,1],c='blue',label="train_set",s=60)
axes[2].scatter(X_wtest_scaled[:,0],X_wtest_scaled[:,1],c='red',label="test_set",s=60)
axes[2].legend(loc='upper left')
axes[2].set_title("wrong_scaled_data")
axes[2].set_xlabel("f0")
axes[2].set_ylabel("f1")
```



Scaling은 train_data를 통해만 진행한다. Test_data와 train_data의 scaling을 따로 진행하면 안된다.

Data_Preprocessing Transformation (Cancer_dataset)

```
cancer= load_breast_cancer()
Xc_train, Xc_test, yc_train , yc_test =
train_test_split(cancer.data,cancer.target, random_state =0)
svm = SVC(C=100) #r규제 약화 -> 과적합
svm.fit(Xc_train, yc_train)
svm.score(Xc_test,yc_test)
```

```
0.629370629370629 4
```

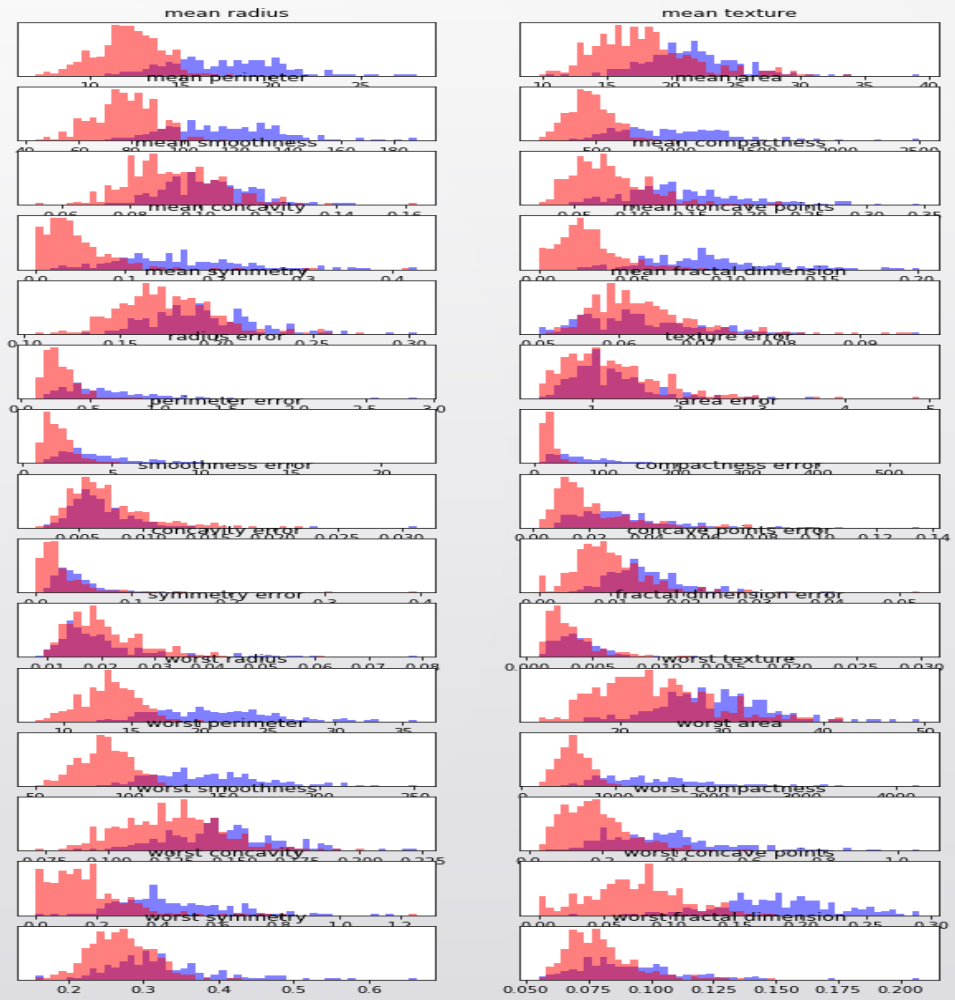
```
scaler = MinMaxScaler()
scaler.fit(Xc_train)
Xc_train_scaled=scaler.transform(Xc_train)
Xc_test_scaled=scaler.transform(Xc_test)

svm = SVC(C=100) #r규제 약화 -> 과적합

svm.fit(Xc_train_scaled, yc_train)
svm.score(Xc_test_scaled,yc_test)
```

```
0.965034965034965
```

Dimension_descending -> cancer_dataset_histogram

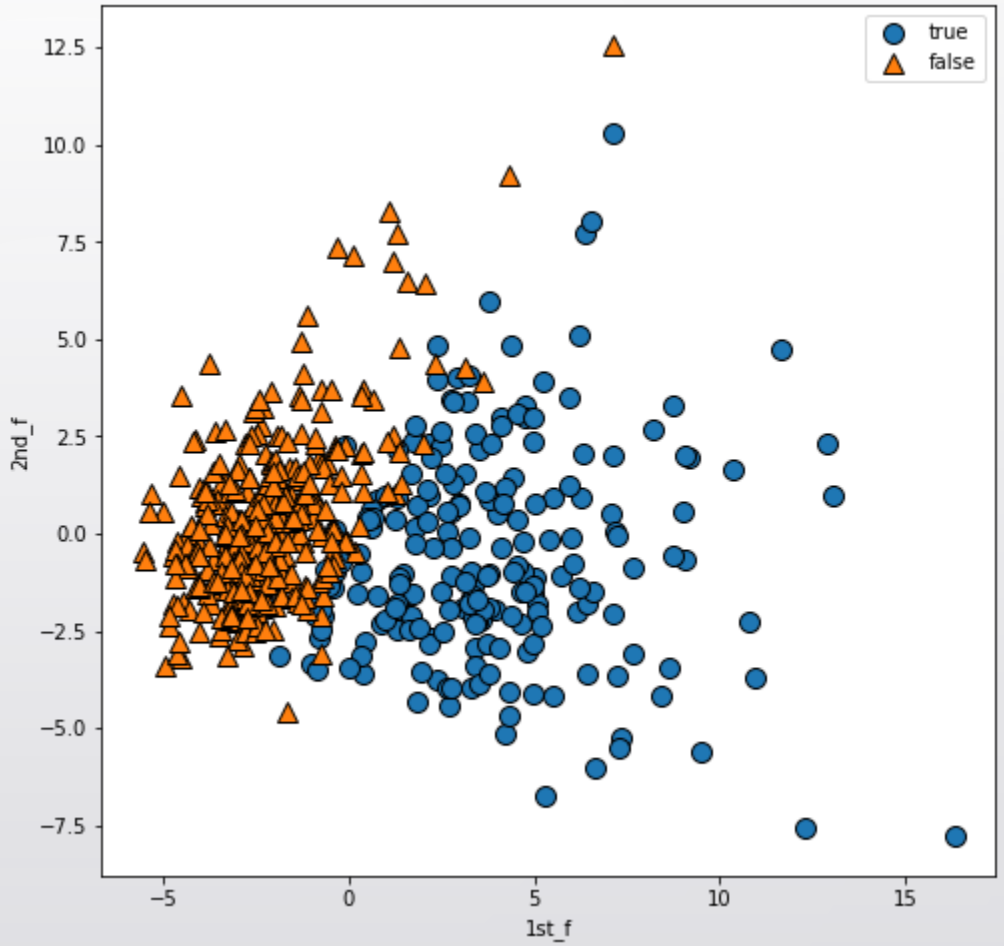
PCA기법 : 주성분 분석법 .

```
fig, axes = plt.subplots(15,2,figsize=(10,20))
malignant = cancer.data[cancer.target ==0] # 음성
benign = cancer.data[cancer.target==1] # 양성

ax = axes.ravel() #flat

for i in range(30):
    _, bins = np.histogram(cancer.data[:,i],bins=50)
#막대기가 그래프당 50개

ax[i].hist(malignant[:,i],bins=bins,color='blue',alpha=.5)
    ax[i].hist(benign[:,i],bins=bins,color='red',alpha=.5)
    ax[i].set_title(cancer.feature_names[i])
    ax[i].set_yticks(())
```



Dimension_descending -> cancer_dataset_pca

```
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaler.fit(cancer.data)
Xc_scaled = scaler.transform(cancer.data)

pca = PCA(n_components=2)
# 주성분 두개만 남긴다.

pca.fit(Xc_scaled)

X_pca = pca.transform(Xc_scaled)

plt.figure(figsize=(8,8))
mglearn.discrete_scatter(X_pca[:,0],X_pca[:,1],cancer.target)
plt.legend(["true" ,"false"],loc="best")
#plt.gca().set_aspect("equal")
plt.xlabel("1st_f")
plt.ylabel("2nd_f")
```

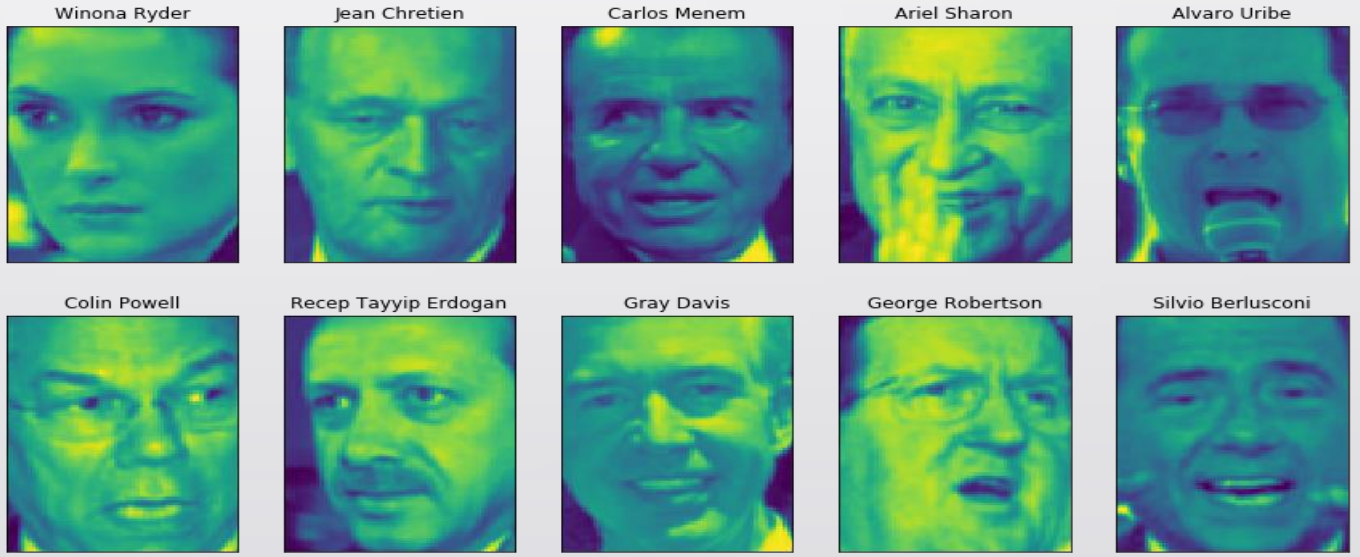



```
(569, 2)
```

Dimension_descending ->eigenface_feature_extraction

with LFW_Dataset

```
from sklearn.datasets import fetch_lfw_people
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

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




```
(3023, 87, 65) class : 62
```

Dimension_descending ->eigenface_feature_extraction

with LFW_Dataset

```
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

counts = np.bincount(people.target)
for i , (count, name) in enumerate(zip(counts, people.target_names)):
    print("{0:25} {1:3}".format(name,count), end=' ')
    if (i%2)==0 :
        print()
```

```
Alejandro Toledo          39
Alvaro Uribe              35 Amelie Mauresmo          21
Andre Agassi              36 Angelina Jolie           20
Ariel Sharon              77 Arnold Schwarzenegger    42
Atal Bihari Vajpayee      24 Bill Clinton             29
Carlos Menem              21 Colin Powell            236
David Beckham             31 Donald Rumsfeld         121
George Robertson          22 George W Bush           530
Gerhard Schroeder        109 Gloria Macapagal Arroyo  44
Gray Davis                26 Guillermo Coria          30
Hamid Karzai              22 Hans Blix                39
Hugo Chavez               71 Igor Ivanov              20
Jack Straw                28 Jacques Chirac           52
Jean Chretien             55 Jennifer Aniston         21
Jennifer Capriati         42 Jennifer Lopez           21
Jeremy Greenstock         24 Jiang Zemin              20
John Ashcroft             53 John Negroponte          31
Jose Maria Aznar          23 Juan Carlos Ferrero      28
Junichiro Koizumi         60 Kofi Annan               32
Laura Bush                41 Lindsay Davenport        22
Lleyton Hewitt            41 Luiz Inacio Lula da Silva 48
Mahmoud Abbas             29 Megawati Sukarnoputri    33
Michael Bloomberg         20 Naomi Watts              22
Nestor Kirchner           37 Paul Bremer              20
Pete Sampras              22 Recep Tayyip Erdogan     30
Ricardo Lagos             27 Roh Moo-hyun             32
Rudolph Giuliani          26 Saddam Hussein           23
Serena Williams           52 Silvio Berlusconi        33
Tiger Woods               23 Tom Daschle              25
Tom Ridge                 33 Tony Blair              144
Vicente Fox               32 Vladimir Putin           49
Winona Ryder              24
```

Data의 편향이 존재한다.

Dimension_descending ->eigenface_feature_extraction

with LFW_Dataset

```
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

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 / 255.

X_train, X_test, y_train , y_test = train_test_split(X_people,y_people,stratify=y_people,
random_state =0)

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)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print(knn.score(X_test_pca,y_test))
```

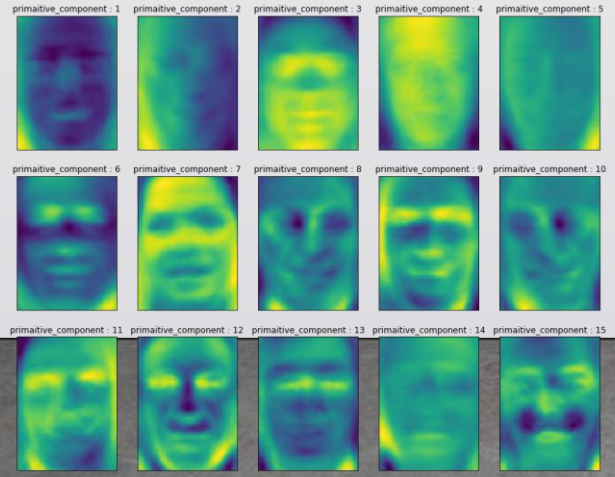
```
0.312015503875969
```

Dimension_descending ->eigenface_feature_extraction

with LFW_Dataset

```
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

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 / 255.

X_train, X_test, y_train , y_test =
train_test_split(X_people,y_people,stratify=y_people, random_state =0)
```

```
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)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)

print(pca.components_.shape)

fig, axes = plt.subplots(3,5,figsize=(15,12),subplot_kw={'xticks': (), 'yticks' : ()})

for i , (component , ax) in enumerate(zip(pca.components_,axes.ravel())):
    ax.imshow(component.reshape(image_shape),cmap = 'viridis')
    ax.set_title("primaitive_component : {}".format(i+1))
```

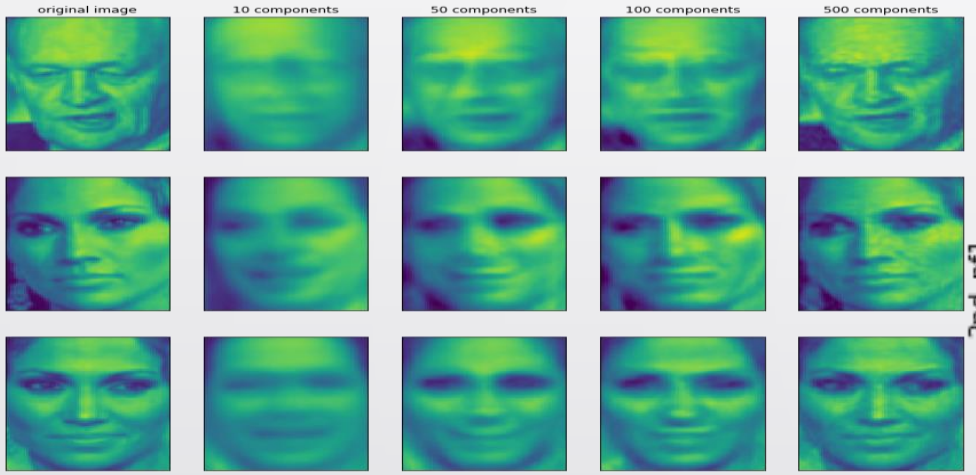


왼쪽그림과 같은 방식은 사람이 얼굴을 인식하는 방식과는
거리가 있으므로, 주성분으로 원래공간으로 돌린다.

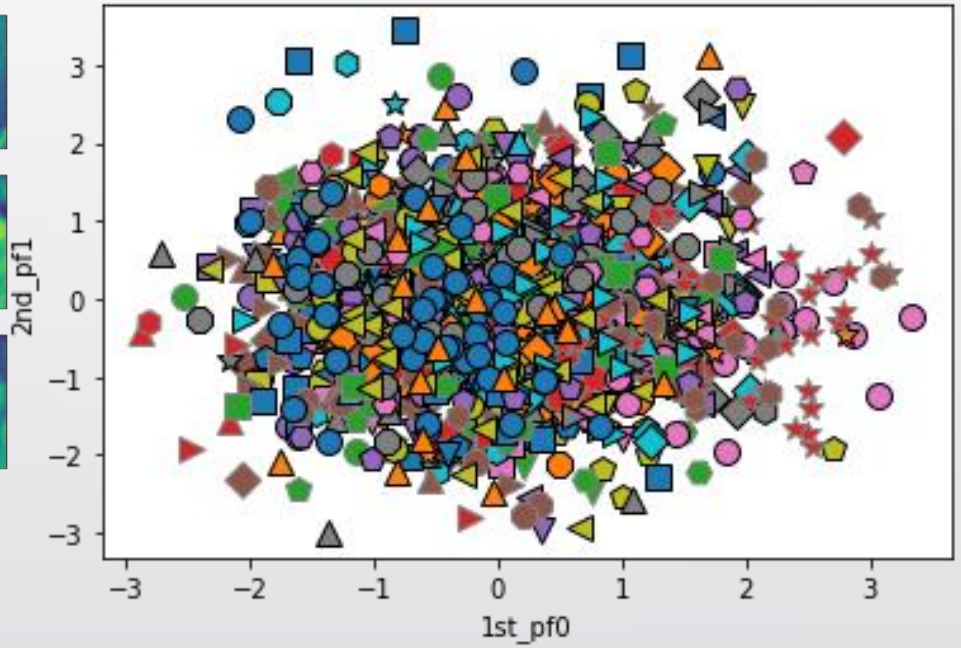
Dimension_descending ->eigenface_feature_extraction

with LFW_Dataset

mglearn.plots.plot_pca_faces(X_train,X_test,image_shape)




```
mglearn.discrete_scatter(X_train_pca[:,0],X_train_pca[:,1],y_train)
plt.xlabel("1st_pf0")
plt.ylabel("2nd_pf1")
```



NMF(Non-negative Matrix Factorization) ->비음수행렬분해

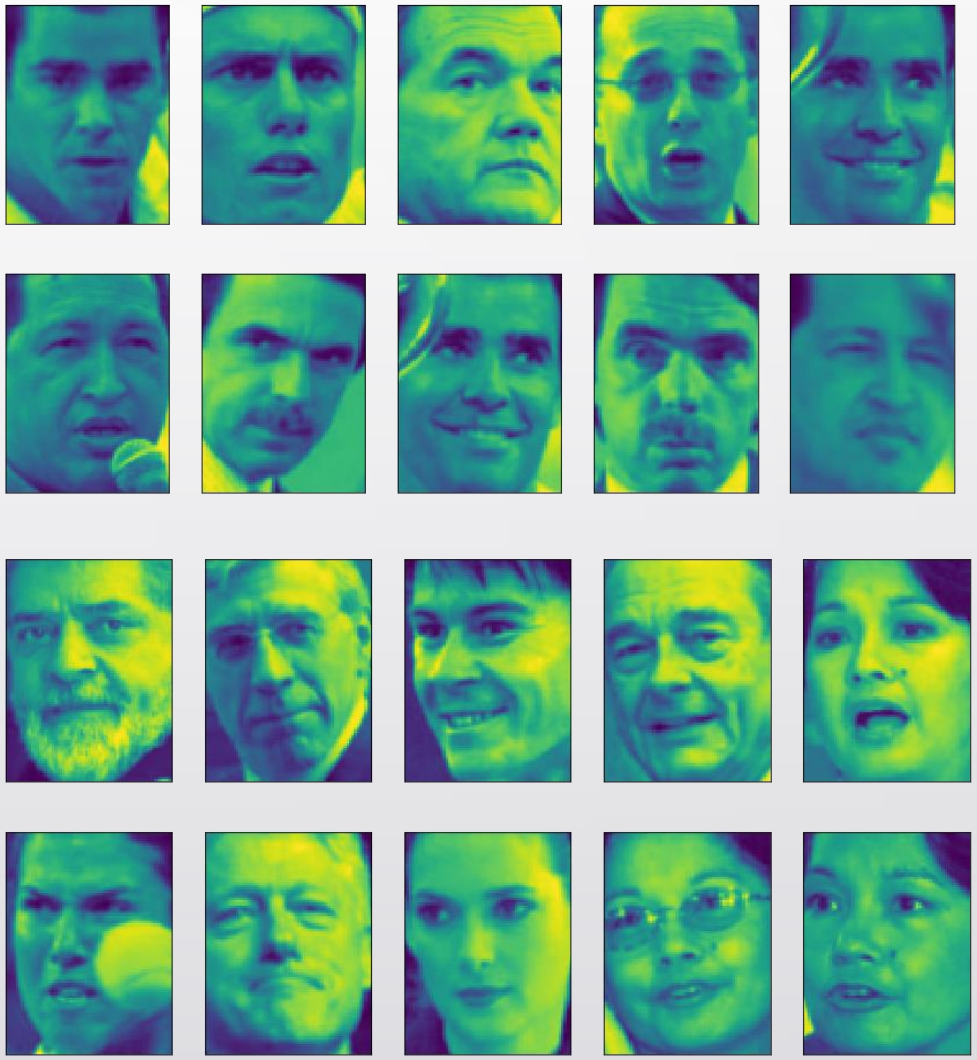
음수 가중치가 없다.
데이터 재구성이나 인코딩하는데는 PCA가 유용하지만 유용한 패턴을 찾는데는 NMF를 사용한다.

```
nmf = NMF(n_components=15, random_state=0)
nmf.fit(X_train)
X_train_nmf = nmf.transform(X_train)
X_test_nmf=nmf.transform(X_test)

fig, axes = plt.subplots(3,5,figsize=(15,12),subplot_kw = {'xticks' : () , 'yticks' : ()})

for i , (comp, ax) in enumerate (zip(nmf.components_, axes.ravel())):
    ax.imshow(comp.reshape(image_shape))
    ax.set_title("component {}".format(i))
```

3번째 주성분과 7번째 주성분의 원본이미지를 한번 보자.

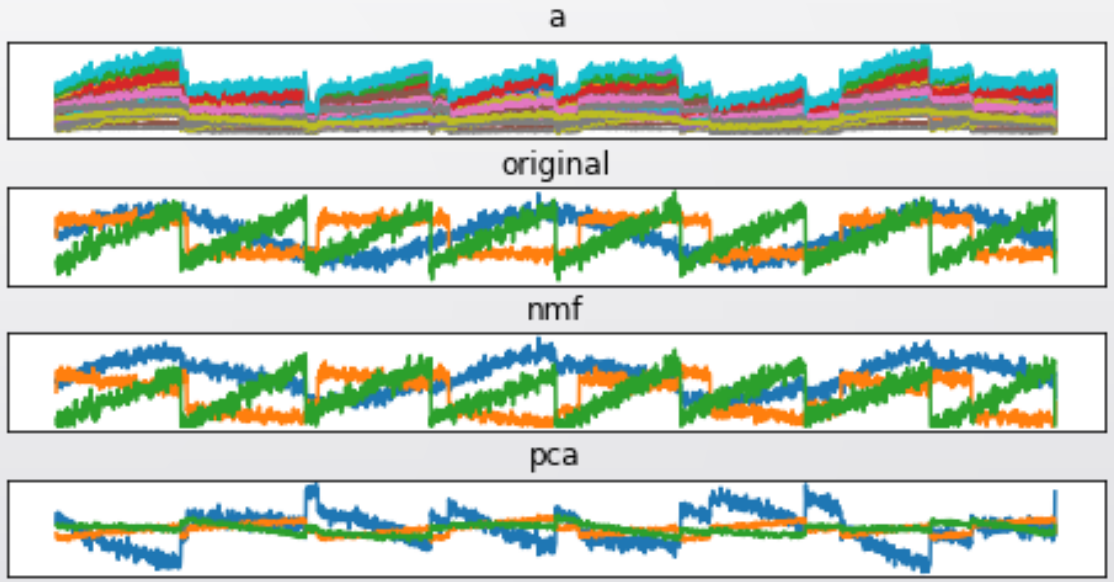
NMF(Non-negative Matrix Factorization) ->비음수행렬분해

```
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)

nmf = NMF(n_components=15, random_state=0)
nmf.fit(X_train)
X_train_nmf = nmf.transform(X_train)
X_test_nmf=nmf.transform(X_test)

compn = 3 # 4번째 주성분
inds = np.argsort(X_train_nmf[:, compn])[::-1]
fig, axes = plt.subplots(2,5,figsize=(15,8),subplot_kw = {'xticks' : (), 'yticks' : ()})
for i , (ind, ax) in enumerate(zip(inds, axes.ravel())):
    ax.imshow(X_train[ind].reshape(image_shape))

compn = 7 #8번째 주성분
inds = np.argsort(X_train_nmf[:, compn])[::-1]
fig, axes = plt.subplots(2,5,figsize=(15,8),subplot_kw = {'xticks' : (), 'yticks' : ()})
for i , (ind, ax) in enumerate(zip(inds, axes.ravel())):
    ax.imshow(X_train[ind].reshape(image_shape))
```



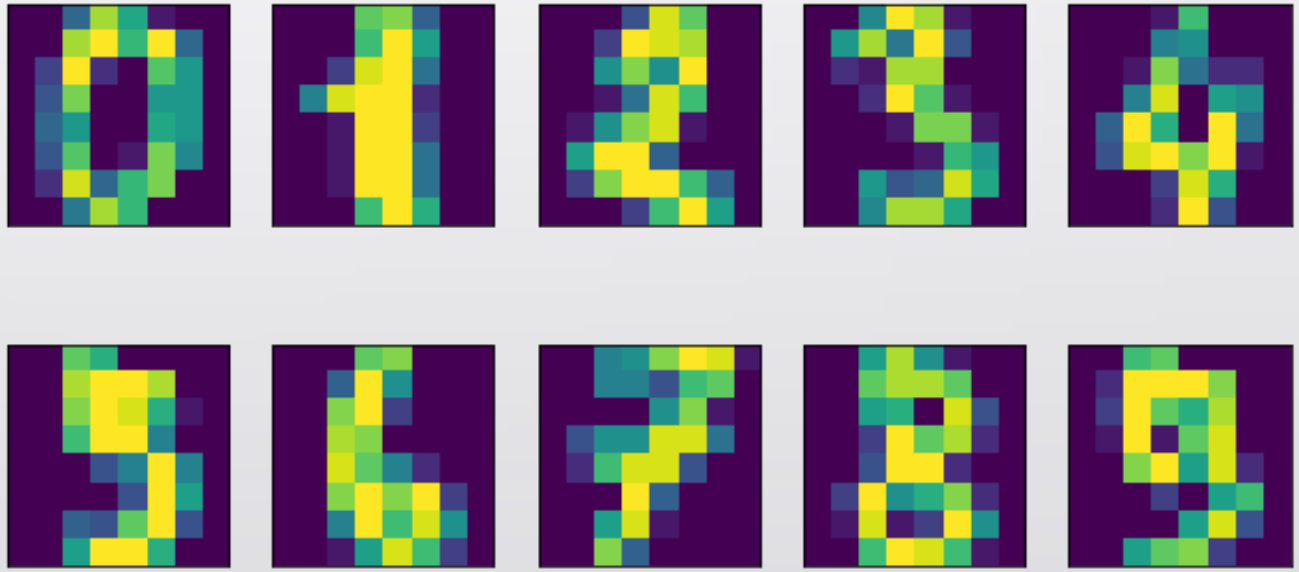
NMF(Non-negative Matrix Factorization) ->비음수행렬분해

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

X_people = X_people / 255.

X_train, X_test, y_train , y_test = train_test_split(X_people,y_people,stratify=y_people, random_state =0)

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)

nmf = NMF(n_components=15, random_state=0)
nmf.fit(X_train)
X_train_nmf = nmf.transform(X_train)
X_test_nmf=nmf.transform(X_test)

S = mglearn.datasets.make_signals()

A = np.random.RandomState(0).uniform(size=(100,3))
X = np.dot(S, A.T)

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

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

models = [X, S , S_, H]
name = ["a", "original", "nmf", "pca"]

fig, axes = plt.subplots(4,figsize=(8,4),gridspec_kw={'hspace' : .5},subplot_kw={'xticks' : (), 'yticks' : ()})

for model , name, ax in zip(models, name, axes):
    ax.set_title(name)
    ax.plot(model,'-')
```



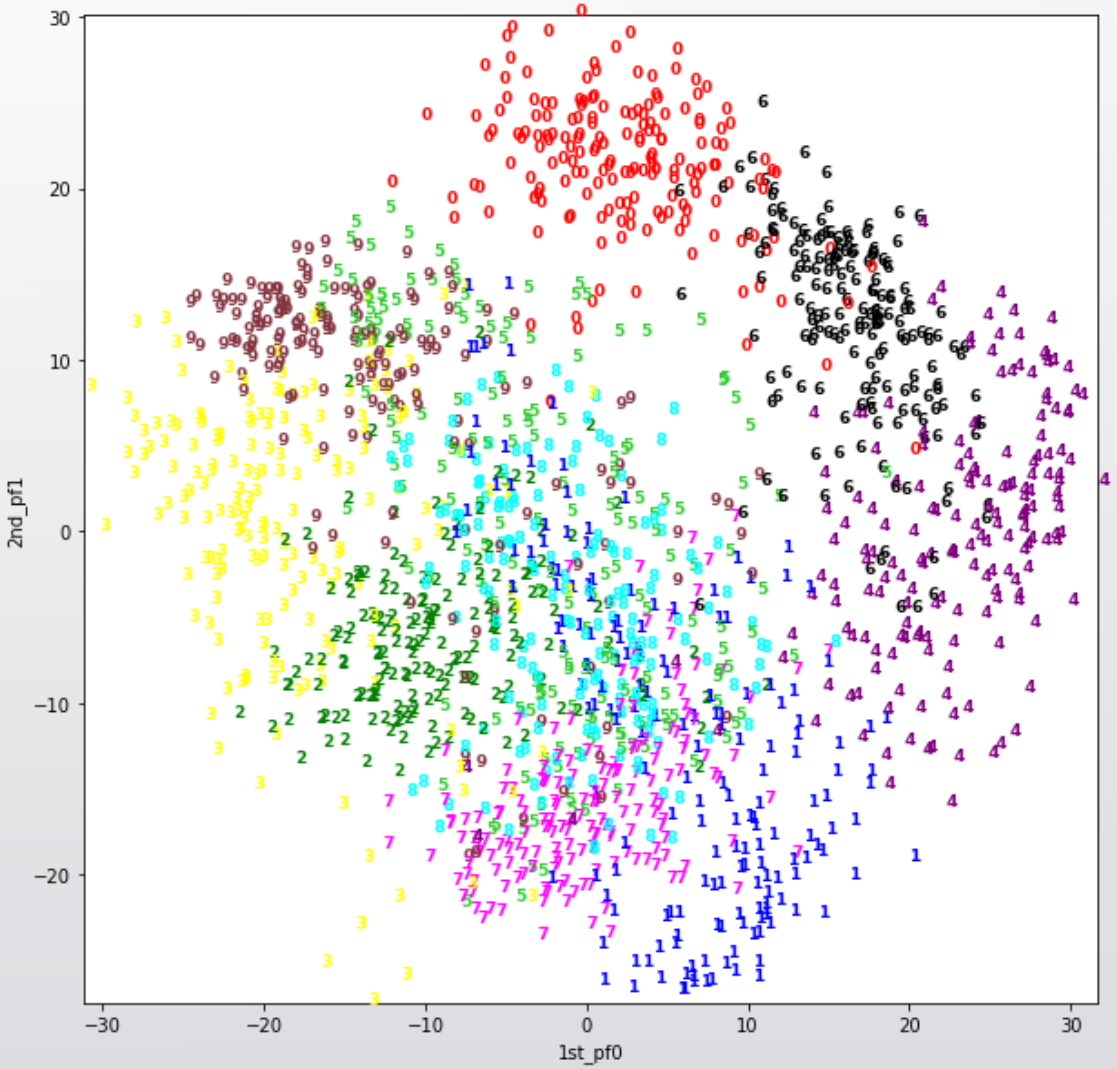
T-sne를 이용한 매니폴드 학습

```
from sklearn.datasets import load_digits
digits = load_digits()
fig ,axes = plt.subplots(2,5, figsize =(10,5),subplot_kw={'xticks':(),'yticks':()})
for ax, img in zip(axes.ravel(),digits.images):
    ax.imshow(img)
```



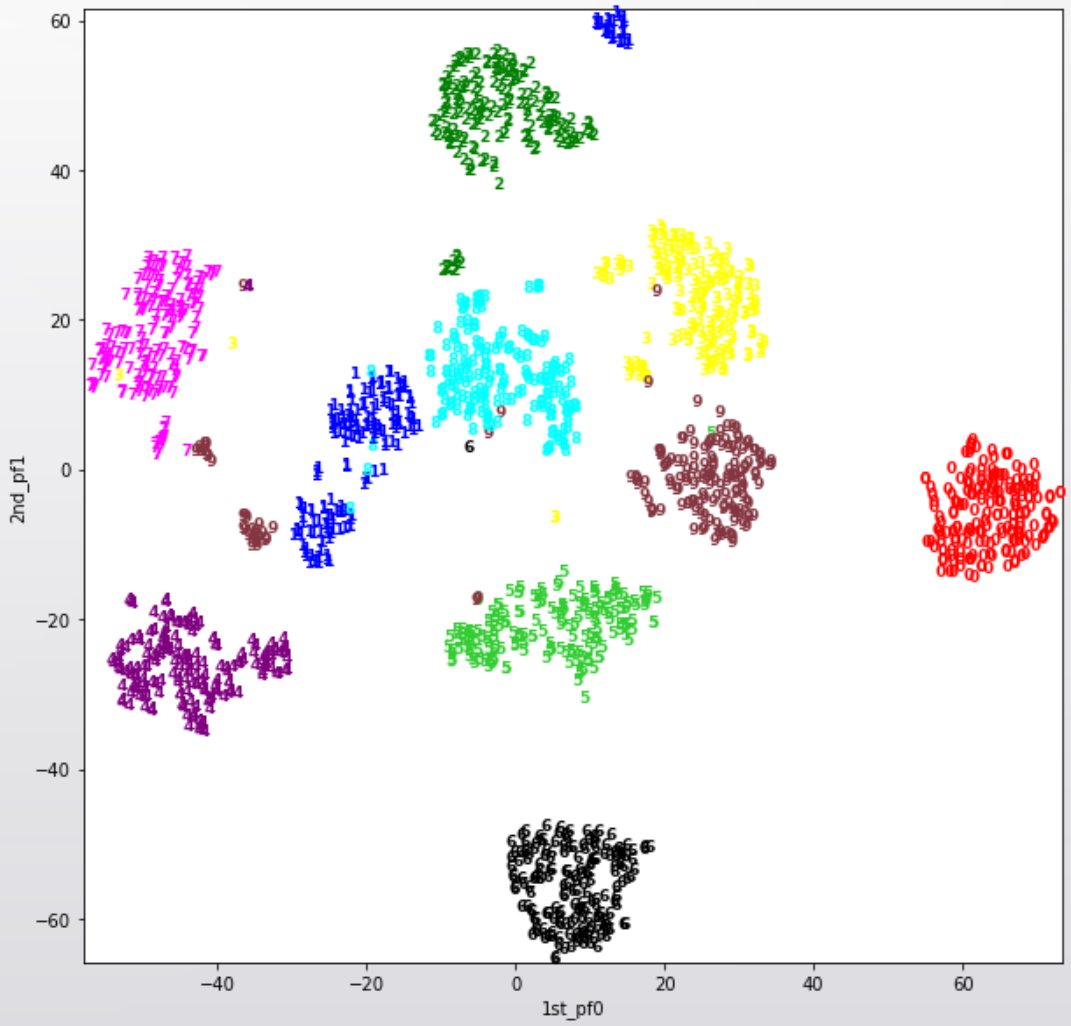
T-sne를 이용한 매니폴드 학습 (pca를 통한 시각화)

```
digits = load_digits()
pca = PCA(n_components=2)
pca.fit(digits.data)

digits_pca = pca.transform(digits.data)

plt.figure(figsize=(10,10))
plt.xlim(digits_pca[:,0].min(),digits_pca[:,0].max())
plt.ylim(digits_pca[:,1].min(),digits_pca[:,1].max())
colors =
["red","blue","green","yellow","purple","limegreen","black","magenta","cyan","#853
541"]
for i in range(len(digits.data)):

plt.text(digits_pca[i,0],digits_pca[i,1],str(digits.target[i]),color=colors[digits.target
[i]],fontdict={'weight': 'bold','size':9})
plt.xlabel("1st_pf0")
plt.ylabel("2nd_pf1")
```



T-sne를 이용한 매니폴드 학습 (t-sne를 통한 시각화)

```
digits = load_digits()
tsne =TSNE(random_state=42)
digit_tsne = tsne.fit_transform(digits.data)


plt.figure(figsize=(10,10))
plt.xlim(digit_tsne[:,0].min(),digit_tsne[:,0].max()+1)
plt.ylim(digit_tsne[:,1].min(),digit_tsne[:,1].max()+1)
colors =
["red","blue","green","yellow","purple","limegreen","black","magenta","cyan","#853
541"]
for i in range(len(digits.data)):

plt.text(digit_tsne[i,0],digit_tsne[i,1],str(digits.target[i]),color=colors[digits.target[i]
],fontdict={'weight': 'bold','size':9})
plt.xlabel("1st_pf0")
plt.ylabel("2nd_pf1")
```



비교적 잘 분포한다. 두가지 주성분을 통해서.
단점 : 새로운 데이터(테스트 데이터)에 적용할 수 없다.
오로지 시각화를 의한 것이다.

Clutsering(군집)

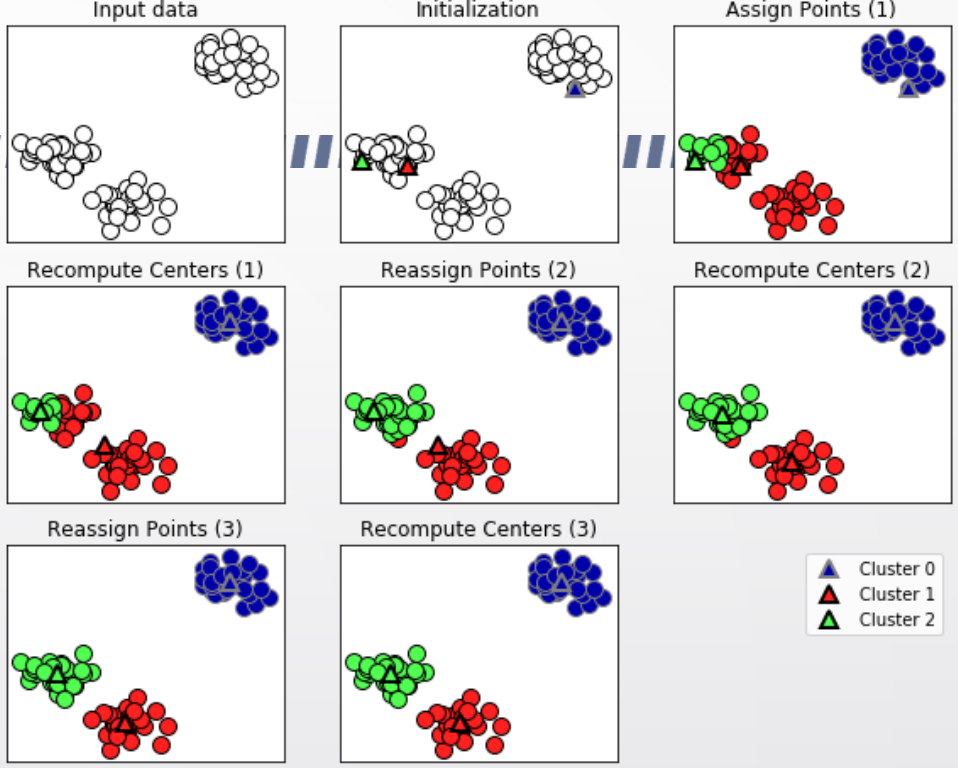
분류와 비슷하지만, 레이블은 없으며, 데이터포인트끼리 비슷하고 다른 클러스터의 데이터포인트와는 구분되도록 하는 것이 목표이다.

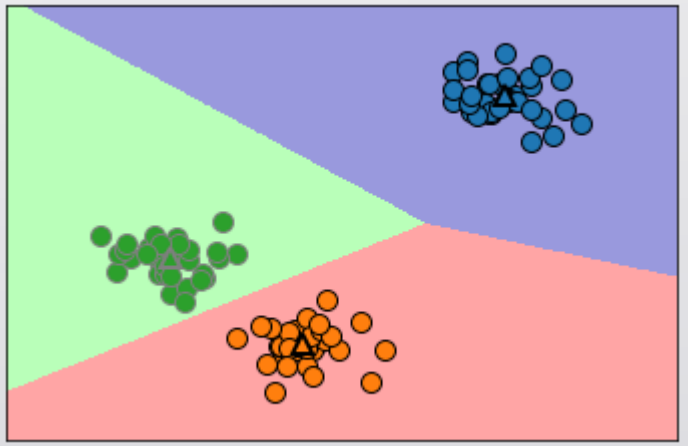
1. K-means clustering

어떤영역을 대표하는 클러스터 중심을 찾는다.
먼저 데이터포인트를 가장 가까운 클러스터 중심에 할당하고

다음 클러스터에 할당된 데이터 포인트의 평균으로 클러스터의 중심을 재정의한다.

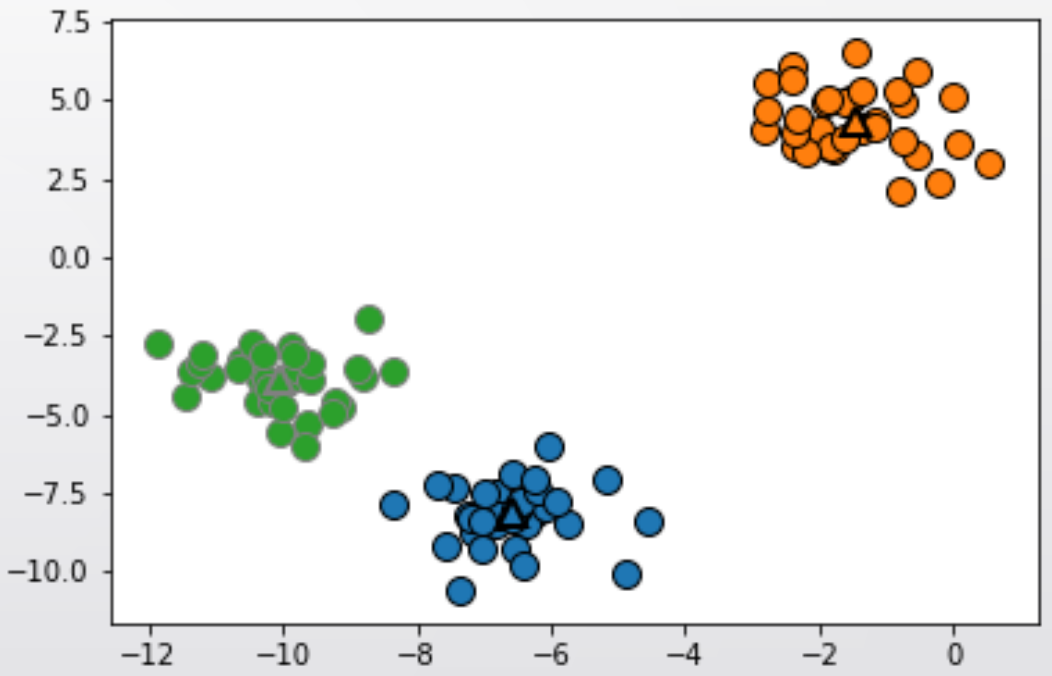
클러스터에 할당되는 데이터 포인트에 변화가 없을 때 알고리즘이 종료된다.





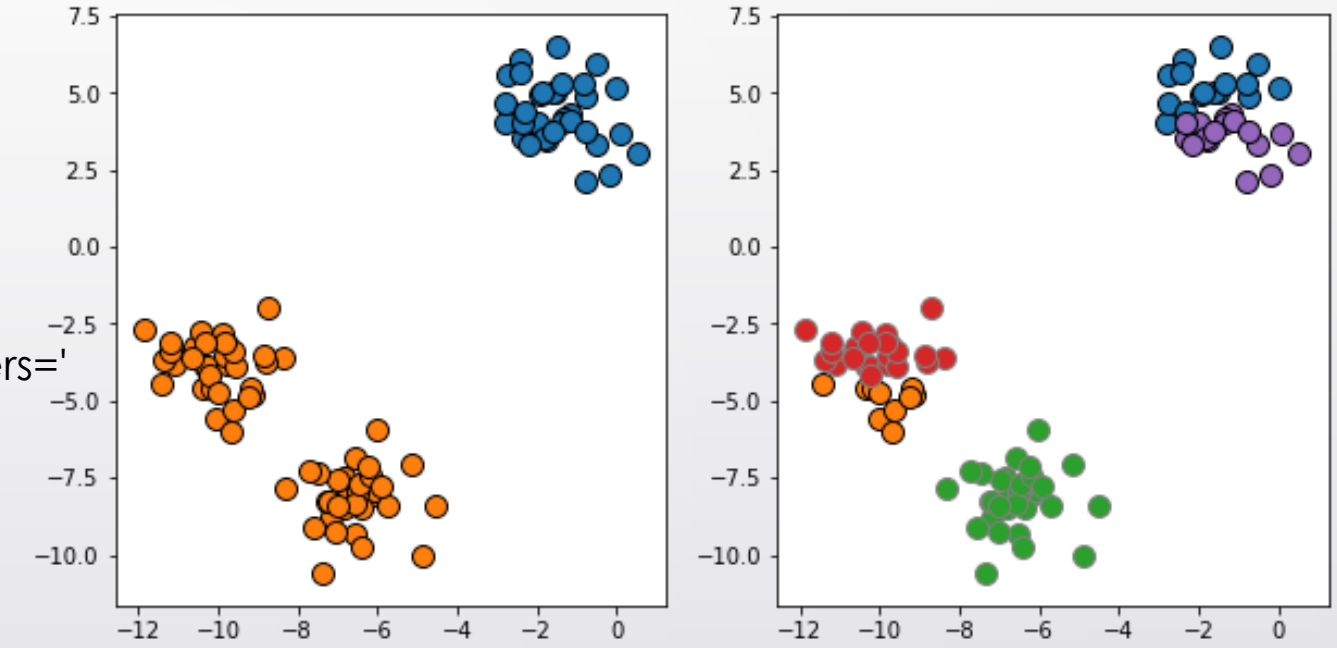
Clutsering(군집)

```
from sklearn.cluster import Kmeans

X, y = make_blobs(random_state=1)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

mglearn.discrete_scatter(X[:,0],X[:,1],kmeans.labels_,
markers='o')
mglearn.discrete_scatter(kmeans.cluster_centers_[:,0
],kmeans.cluster_centers_[:,1],[0,1,2],markers='^',mark
eredgewidth=2)
```



Clutsering(군집)

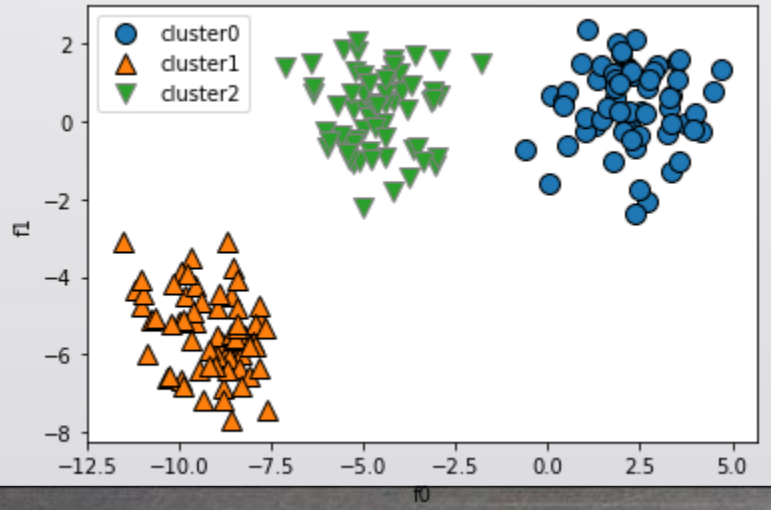
```
from sklearn.cluster import Kmeans

X, y = make_blobs(random_state=1)
fig, axes = plt.subplots(1,2,figsize=(10,5))

kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
assignments=kmeans.labels_
mglearn.discrete_scatter(X[:,0],X[:,1],assignments,markers='
o',ax=axes[0])

kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
assignments=kmeans.labels_
mglearn.discrete_scatter(X[:,0],X[:,1],assignments,markers='
o',ax=axes[1])
```

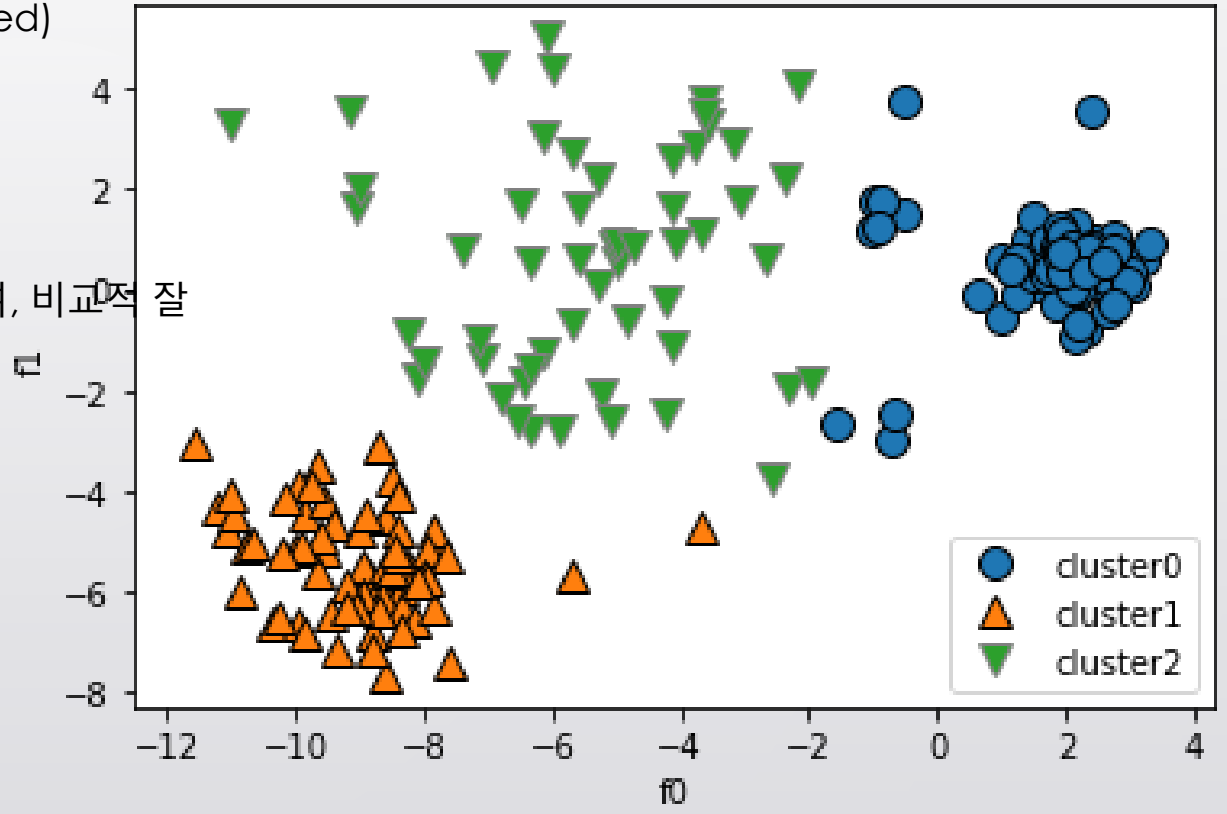


Clutsering(군집) : clustering_std_error 예시

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
X_varied, y_varied = make_blobs(n_samples=200,cluster_std=[1.0,2.5,0.5],random_state=170)
#cluster_std는 각 군집당 표준편차. 군집을 3개로 제한 할 것이므로, 3개 집단에 대한 표준편차를 설정해주었다.
y_pred = KMeans(n_clusters=3,random_state=0).fit_predict(X_varied)

mglearn.discrete_scatter(X_varied[:,0],X_varied[:,1],y_pred)
plt.legend(["cluster0", "cluster1", "cluster2"],loc='best')
plt.xlabel("f0")
plt.ylabel("f1")
plt.show()
```

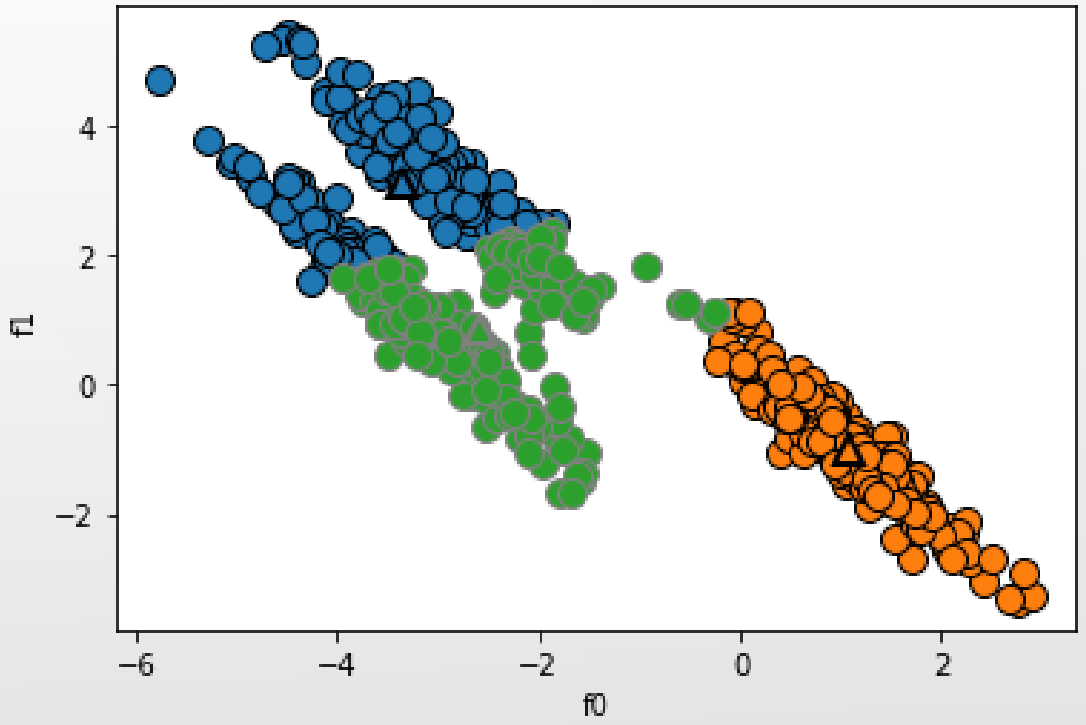
기본설정인 표준편차를 1.0으로 설정해주면 밀도가 일정하여, 비교적 잘 군집화한다는 것을 알 수 있다.





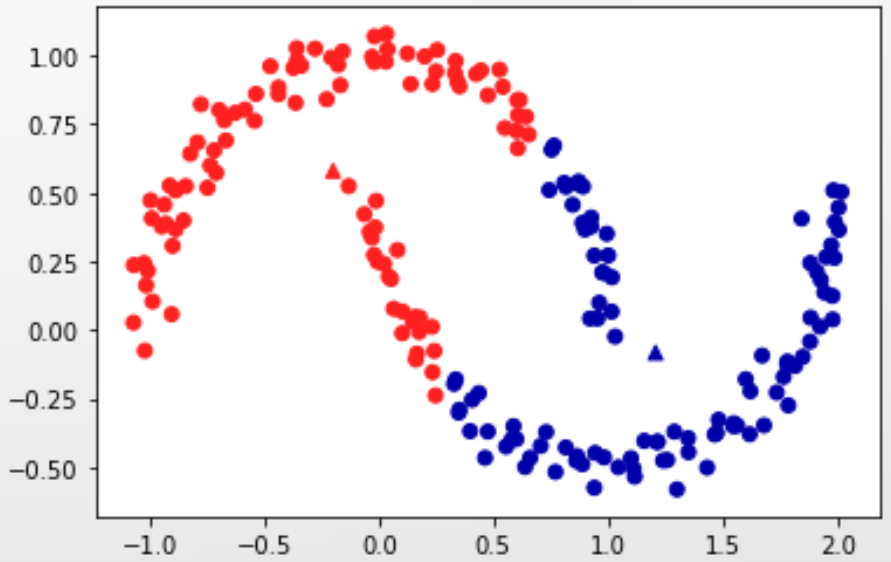
Clutsering(군집) : 정규분포 형태 데이터 클러스터링 예시

```
X, y= make_blobs(n_samples=600,random_state=170)
rng = np.random.RandomState(74)
trans=rng.normal(size=(2,2))
X=np.dot(X,trans)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_pred = kmeans.predict(X)

mglearn.discrete_scatter(X[:,0],X[:,1],kmeans.labels_,
markers='o')
mglearn.discrete_scatter(kmeans.cluster_centers_[:,0
],kmeans.cluster_centers_[:,1],[0,1,2],markers='^',mark
eredgewidth=2)
plt.xlabel("f0")
plt.ylabel("f1")

plt.show()
```



클러스터링에서 모든방향이 중요하다고 생각하기 때문에, 원형이 아닌 데이터셋에서 클러스터링이 제대로 되지 않는다.

Clutsering(군집) :two_moons dataset




```
X, y= make_moons(n_samples=200,noise=0.05,random_state=0)
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
y_pred=kmeans.predict(X)

plt.scatter(X[:,0],X[:,1],c=y_pred,cmap=mglearn.cm2)
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],c=[mglearn.cm2(0),mglearn.cm2(1)],marker='^')
```

Clutsering(군집) : LFW_dataset

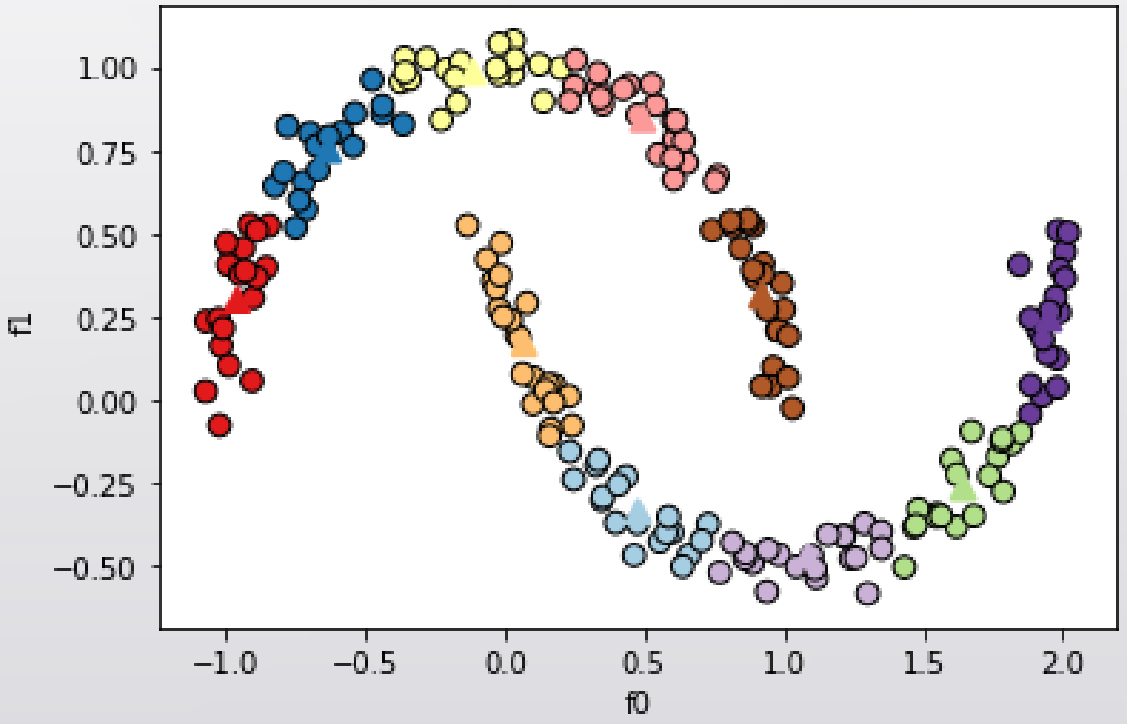
```
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

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 / 255.

X_train, X_test, y_train , y_test = train_test_split(X_people,y_people,stratify=y_people,
random_state =42)
```

```
nmf = NMF(n_components=100,random_state=0)
nmf.fit(X_train)
pca = PCA(n_components=100,random_state=0)
pca.fit(X_train)

km = KMeans(n_clusters=100,random_state=0)
km.fit(X_train)

X_re_pca = pca.inverse_transform(pca.transform(X_test))
X_re_km = km.cluster_centers_(km.predict(X_test))
X_re_nmf = np.dot(nmf.transform(X_test),nmf.components_)

fig, axes = plt.subplots(3,5,figsize=(8,8),subplot_kw={'xticks' : (), 'yticks' : ()})

for ax , c_kmeans, c_pca, c_nmf in zip(axes.T,
km.cluster_centers_,pca.components_,nmf.components_c):
    ax[0].imshow(c_kmeans.reshape(image_shape))
    ax[1].imshow(c_pca.reshape(image_shape))
    ax[2].imshow(c_nmf.reshape(image_shape))

fig , axes = plt.subplots(4,5,figsize=(8,8),subplot_kw={'xticks' : (), 'yticks' : ()})

fig.suptitle("reconstruct")

for ax , orig, re_kmeans, re_pca, re_nmf in zip(axes.T,X_test,T,X_re_km, X_re_pca,X_re_nmf):
    ax[0].imshow(orig.reshape(image_shape))
    ax[1].imshow(re_kmeans.reshape(image_shape))
    ax[2].imshow(re_pca.reshape(image_shape))
    ax[3].imshow(re_nmf.reshape(image_shape))
```

Clutsering(군집) :two_moons dataset

```
X,y = make_moons(n_samples=200, noise=0.05, random_state=0)
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(X)
y_pred=kmeans.predict(X)

plt.scatter(X[:,0],X[:,1],c=y_pred,s=60, cmap='Paired',edgecolors='black')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=60,marker
='^',c=range(kmeans.n_clusters),linewidth=2,cmap='Paired')
plt.xlabel("f0")
plt.ylabel("f1")

print(kmeans.cluster_centers_)
print(y_pred)
```

```
[[ 0.47784629 -0.33587051] [-0.63082981 0.74997507]
[ 1.64363048 -0.26255916] [ 0.49568033 0.84280265]
[-0.95763266 0.30017383] [ 0.06971286 0.16994249]
[ 1.08659435 -0.46800151] [ 1.9491367 0.24515351]
[-0.11342346 0.98338261] [ 0.92190054 0.3132417 ]]
 [9 2 5 4 2 7 9 6 9 6 1 0 2 6 1 9 3 0 3 1 7 6 8 6 8 5 2 7
 5 8 9 8 6 5 3 7 0 9 4 5 0 1 3 5 2 8 9 1 5 6 1 0 7 4 6 3
3 6 3 8 0 4 2 9 6 4 8 2 8 4 0 4 0 5 6 4 5 9 3 0 7 8 0 7 5
 8 9 8 0 7 3 9 7 1 7 2 2 0 4 5 6 7 8 9 4 5 4 1 2 3 1 8 8
4 9 2 3 7 0 9 9 1 5 8 5 1 9 5 6 7 9 1 4 0 6 2 6 4 7 9 5 5
3 8 1 9 5 6 3 5 0 2 9 3 0 8 6 0 3 3 5 6 3 2 0 2 3 0 2 6 3 4
4 1 5 6 7 1 1 3 2 4 7 2 7 3 8 6 4 1 4 3 9 9 5 1 7 5 8 2]
```



Clutsering(군집) :two_moons dataset

```
distance_feature=kmeans.transform(X)
print(distance_feature.shape)
print(distance_feature)
```

200개의 샘플에 대해 10개의 군집을 설정하였으므로, 10개의 군집 중심까지의 거리를 transform메소드는 반환한다.

```
(200, 10)
[[0.9220768 1.46553151 1.13956805 ... 1.16559918 1.03852189 0.23340263]
[1.14159679 2.51721597 0.1199124 ... 0.70700803 2.20414144 0.98271691]
[0.78786246 0.77354687 1.74914157 ... 1.97061341 0.71561277 0.94399739] ...
[0.44639122 1.10631579 1.48991975 ... 1.79125448 1.03195812 0.81205971]
[1.38951924 0.79790385 1.98056306 ... 1.97788956 0.23892095 1.05774337]
[1.14920754 2.4536383 0.04506731 ... 0.57163262 2.11331394 0.88166689]]
```

Agglomerative Clutsering(병합군집)

병합군집은 각 포인트는 하나의 클러스터이며, 설정한 클러스트가 될때까지, 클러스터를 병합시킨다.

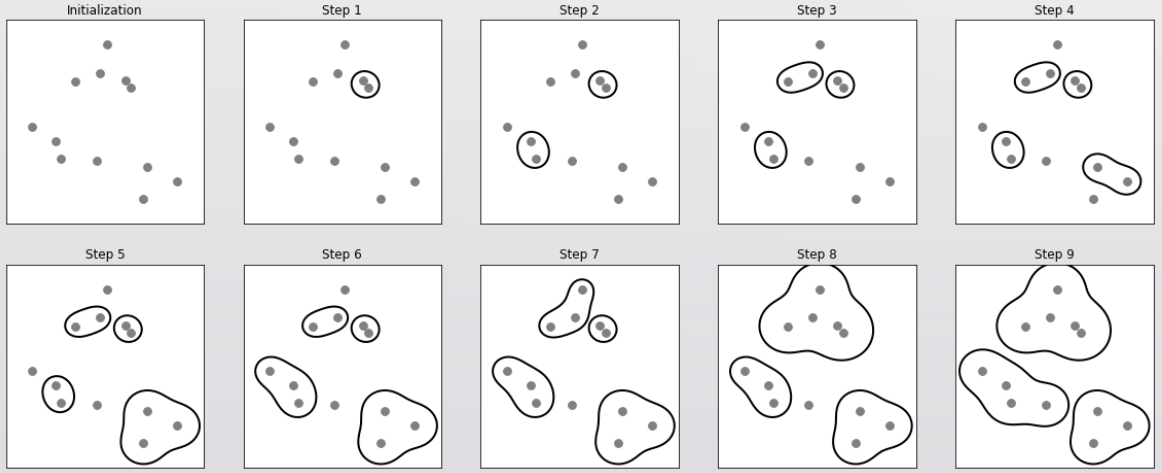
Linkage는 연결조건이며

ward / average / complete가 있다.

Ward는 모든 클러스터내의 분산을 가장 적게 증가시키는 두클러스터를 병합

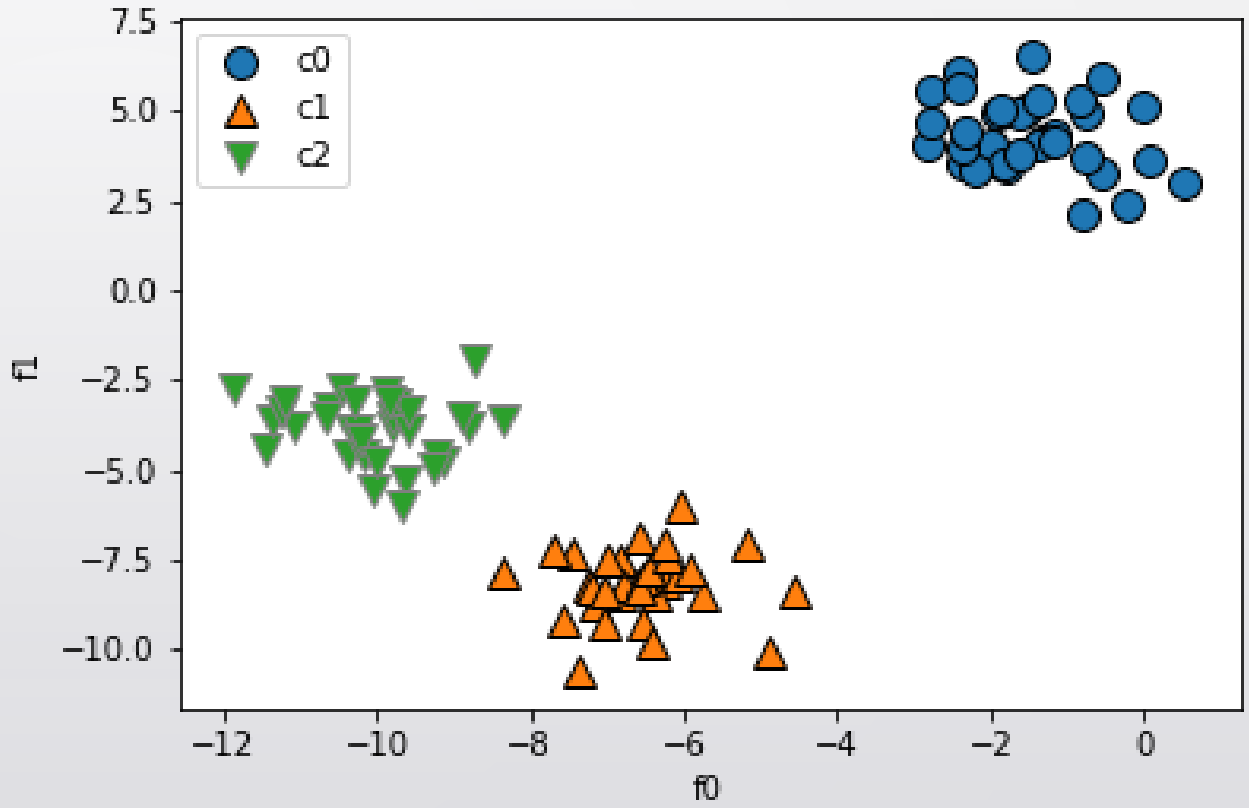
Average는 클러스터 포인트 사이의 평균거리가 가장 짧은 두 클러스터를 병합

Complete는 클러스터 포인트 사이의 최대 거리가 가장 짧은 두클러스터를 병합한다.



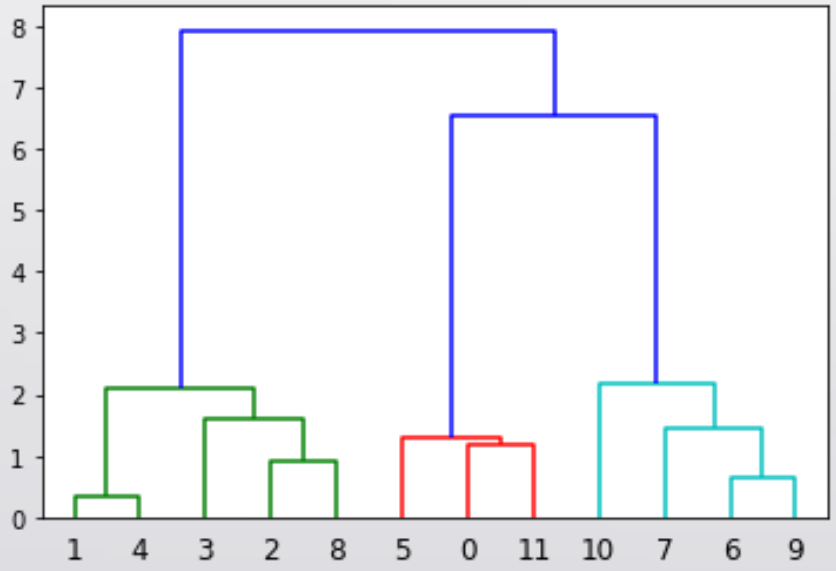
Agglomerative Clutsering(병합군집)

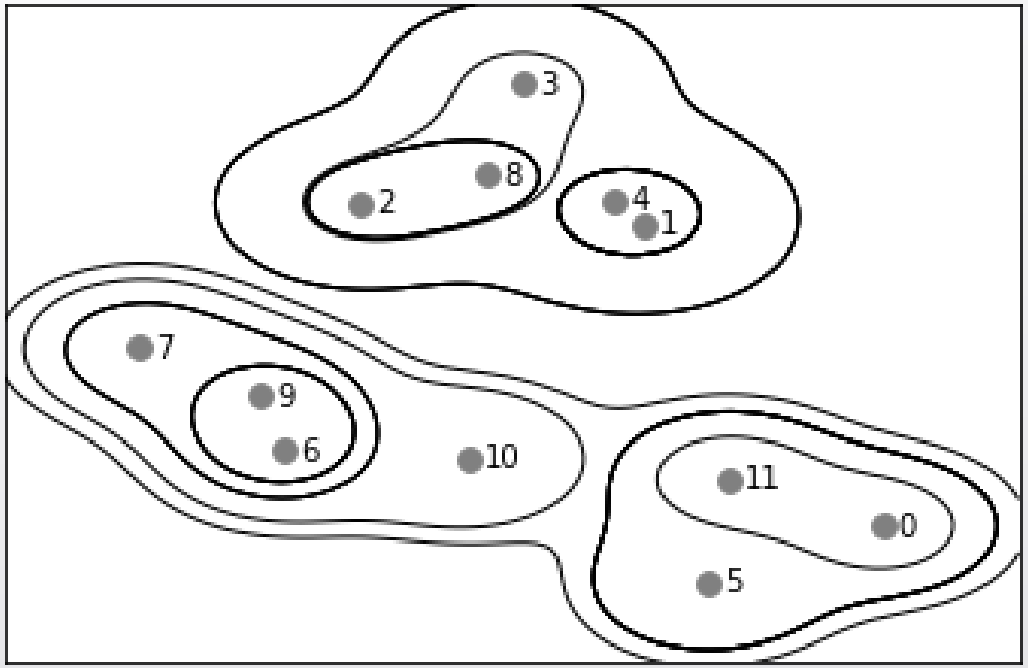
```
from sklearn.cluster import AgglomerativeClustering

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

mglearn.discrete_scatter(X[:,0],X[:,1],assign)
plt.legend(["c0","c1","c2"],loc="best")
plt.xlabel("f0")
plt.ylabel("f1")
```



Agglomerative Clutsering(병합군집) : 계층적군집

```
from scipy.cluster.hierarchy import dendrogram, ward
X,y = make_blobs(random_state=0,n_samples=12)

linkage_arr = ward(X)
dendrogram(linkage_arr)

ax=plt.gca()
ax
```





군집 : DBSCAN

군집개수 정할 필요 없음.
-> min_samples / eps를 정해야함.

한데이터 포인트에서 eps거리안에 데이터가 min_samples만큼 있으면 이 데이터 포인트를 핵심 샘플로 분류한다.

Eps보다 가까운 데이터포인트들은 동일한 클러스터로 합쳐진다.

1. 무작위로 포인트를 선택하고, eps거리안 모든 포인트를 찾고, min_samples이하면 그 포인트는 잡음이며, 많으면 핵심 샘플로 레이블하고 새로운 클러스터 레이블을 할당한다.
2. 그 포인트의 모든 이웃을 살피고, 어떤 클러스터에도 할당되지 않았다면, 전에 만든 클러스터 레이블을 할당한다.

```
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]
```

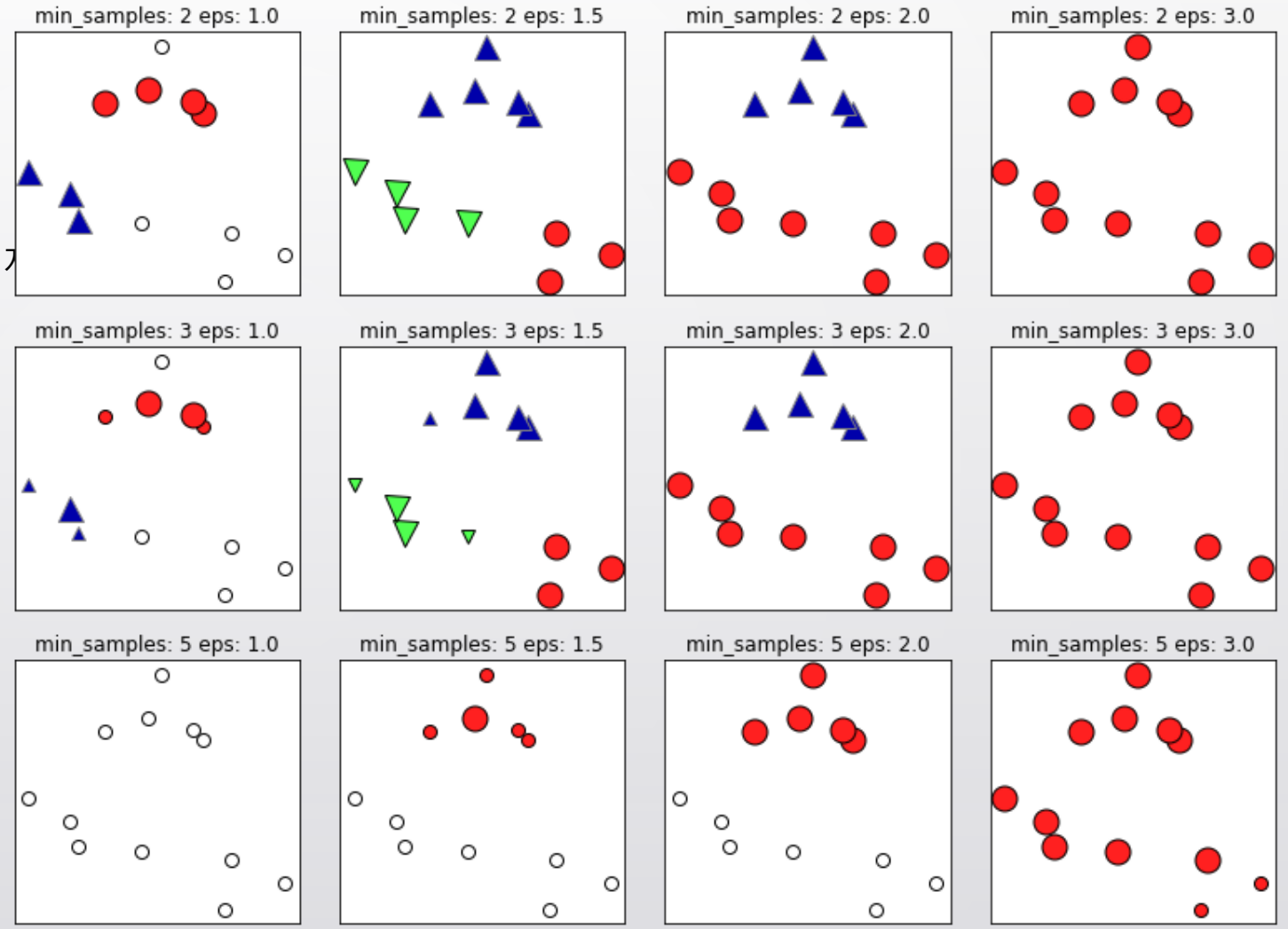
군집 : DBSCAN

군집개수 정할 필요 없음.
-> min_samples / eps를 정해야함.

Eps가 커지면 단하나의 클러스터가 될수 있으며
너무 작아지면, 모든 데이터포인트가 잡음 처리될 수 있다.

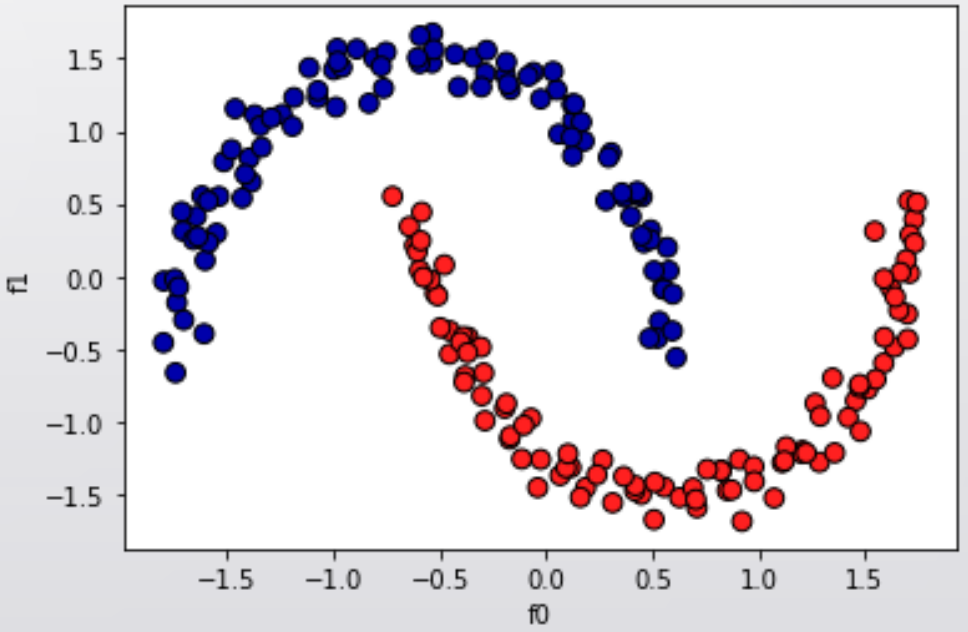
Min_samples는 조밀한 지역에 있는 포인트들이 잡음이 될것인지
클러스터가 될것인지 결정.

늘리면, 잡음 포인트가 증가하고,
Min_sampels는 클러스터의 최소크기를 결정한다.



군집 : DBSCAN

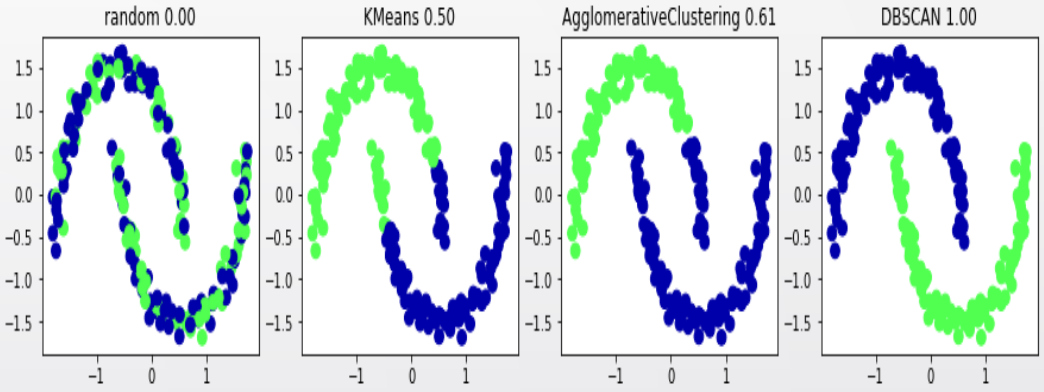
```
from sklearn.preprocessing import MinMaxScaler, QuantileTransformer, StandardScaler,
PowerTransformer
X,y = make_moons(random_state=0,noise=0.05,n_samples=200)
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
dbscan = DBSCAN() # eps 기본값 0.5
clusters = dbscan.fit_predict(X_scaled)

plt.scatter(X_scaled[:,0],X_scaled[:,1],c=clusters,cmap=mglearn.cm2,s=60,edgecolors='black')
plt.xlabel("f0")
plt.ylabel("f1")
```



군집 알고리즘의 비교와 평가. (NMI, ARI)

군집 알고리즘의 결과를 실제 정답 클러스터와 비교하여 평가할 수 있는 지표들이 있다. 1(최적일때) , 0(무작위로 분류될때) 사이의 값을 제공하는 ARI(Adjusted Rand Index)와 NMI(Normalized Mutual Information)이 가장 널리 사용하는 지표이다.

```
X,y = make_moons(random_state=0,noise=0.05,n_samples=200)
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

fig, axes = plt.subplots(1,4,figsize=(15,3))
algo = [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.c
m3,s=60)
axes[0].set_title("random {:.2f}".format(adjusted_rand_score(y,random_clusters)))

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



accuarcy_score는 지도학습용 메트릭이다!

군집알고리즘에서는 ARI나 NMI같은 지표를 사용해야한다.

하지만 ARI나 NMI 역시, y같은 레이블을 요구로하는 메소드 이므로,

실제 애플리케이션에서는 아무 쓸모가 없다.

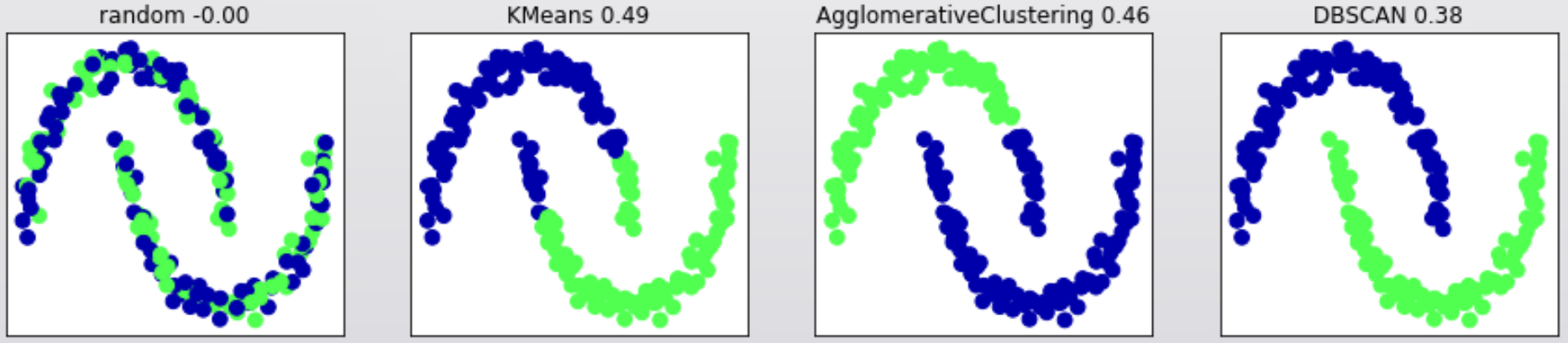
레이블이 없을때 사용하는 실루엣 계수는 잘동작하지 않는다.

군집 알고리즘의 비교와 평가. (실루엣 계수)

실루엣계수는 클러스터의 밀집 정도를 계산하는 것으로 높을 수록 좋으며 최대 점수는 1점.

밀집된 클러스터가 좋지만 모양이 복잡할때는 밀집도를 활용한 평가가 맞지 않다.
->이렇듯 클러스터링 할때는 직접 확인하는 것이 중요하다.

```
axes[0].scatter(X_scaled[:,0],X_scaled[:,1],c=random_clusters,cmap=mglearn.cm3,s=60)
axes[0].set_title("random {:.2f}".format(silhouette_score(X_scaled,random_clusters)))

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

#silhouette_score는 전처리된 X_scaled와 cluster를 파라미터로 전달한다.
```



군집 알고리즘의 비교와 평가. (얼굴 데이터셋)

```
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

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 / 255.

pca = PCA(n_components=100,whiten=True,random_state=0)
X_pca=pca.fit_transform(X_people)

dbscan = DBSCAN(min_samples=3, eps=15)
labels=dbscan.fit_predict(X_pca)
noise=X_people[labels==-1]

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))
```

이렇게 잡음에 대한 데이터를 살펴보는 것을 Outlier Detection이라고 한다.

군집 알고리즘의 비교와 평가. (얼굴 데이터셋)

```
people = fetch_lfw_people(min_faces_per_person=20,resize=.7)
image_shape=people.images[0].shape

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 / 255.


pca = PCA(n_components=100,whiten=True,random_state=0)
X_pca=pca.fit_transform(X_people)

dbscan = DBSCAN(min_samples=3, eps=15)
labels=dbscan.fit_predict(X_pca)
noise=X_people[labels==-1]

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

```
1
cluster : 1 cluster_size : [2063]
3
cluster : 1 cluster_size : [2063]
5
cluster : 1 cluster_size : [2063]
7
cluster : 14 cluster_size : [2004 3 14 7 4 3 3 4 4 3 3 5 3 3]
9
cluster : 4 cluster_size : [1307 750 3 3] 11
cluster : 2 cluster_size : [ 413 1650]
13
cluster : 2 cluster_size : [ 120 1943]
```

군집 알고리즘의 비교와 평가. (얼굴 데이터셋) -> kmeans

```
km = KMeans(n_clusters=10,random_state=0)
labels_km = km.fit_predict(X_pca)
print(np.bincount(labels_km))
```

```
[155 175 238 75 358 257 91 219 323 172]
```