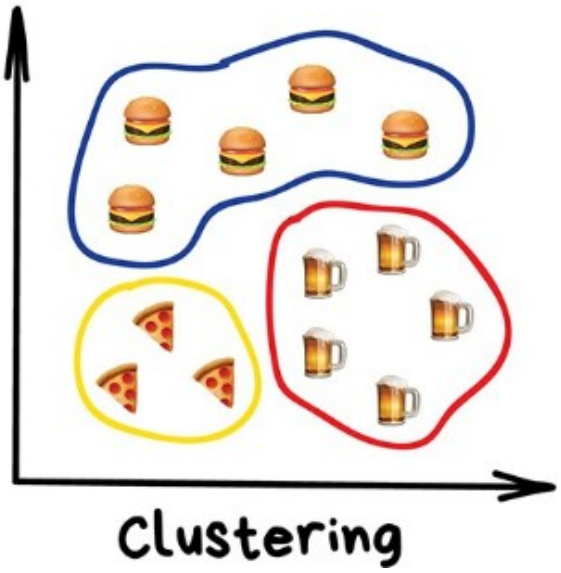
Machine Learning & Scikit-Learn

Day5. 군집

군집(clustering)

- 군집화(Clustering)
  - 비지도 학습
  - 데이터의 성질로 부터 최적으로 분할하고 레이블을 구함
  - 오늘날의 활용 분야
    - ✓ 시장 분할
    - ✓ 지도에서 가까운 지점을 병합
    - ✓ 영상 압축
    - ✓ 자료에 새로 레이블 부여
    - ✓ 이상행동 감지
  - 방법론
    - ✓ K-means clustering, DBSCAN, etc



#1

K-평균 군집화

군집 - (1) k-평균 군집화

- K-평균 군집화(k-means clustering)
  - 레이블이 없는 다차원 데이터 세트 내에 사전 정의된 군집의 개수를 찾아내는 방법
  - 최적의 군집화
    - ✓ '군집 중앙 ' 은 해당 군집에 속하는 모든 점의 산술 평균이다.
    - ✓ 각 점은 다른 군집의 중앙보다 자신이 속한 군집의 중앙에 더 가깝다.

군집 - (1) k-평균 군집화

- 군집화를 보여주기 위한 데이터

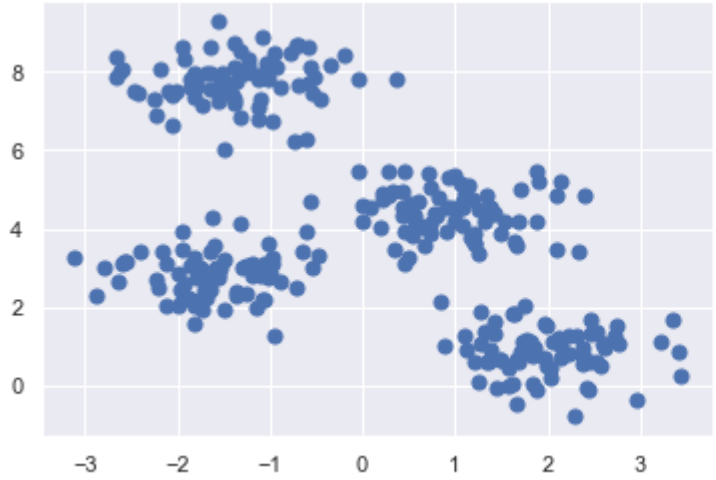
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

군집 - (1) k-평균 군집화

- 군집화를 보여주기 위한 데이터

```
# 4개 영역의 2차원 자료의 생성
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std = 0.60, random_state = 0)
plt.scatter(X[:, 0], X[:, 1], s=50)
```

```
<matplotlib.collections.PathCollection at 0x2427d3ddac8>
```



https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

군집 - (1) k-평균 군집화

- K-means 군집화 모델 인스턴스화 및 학습

```
# k-means clustering
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 4)
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

군집 - (1) k-평균 군집화

- K-means 군집화

```
y_kmeans = kmeans.predict(X)
```

```
print(y_kmeans)
```

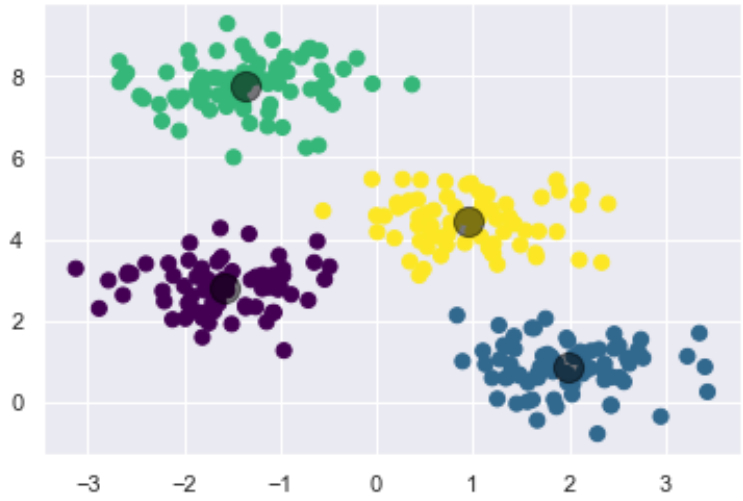
```
[1 2 3 2 1 1 0 3 2 2 0 2 3 2 1 3 3 1 0 0 1 1 3 0 0 3 1 3 0 3 2 2 3 2 2 2 2
 2 0 1 3 0 3 3 0 0 2 0 2 1 0 1 2 1 1 0 2 0 2 1 2 3 2 0 0 0 2 1 2 0 3 0 2 0
 0 2 0 3 1 2 1 3 1 1 2 3 1 3 2 2 3 1 2 0 0 3 1 1 3 0 2 1 2 1 3 1 1 3 2 3 0
 0 1 2 1 3 2 1 1 3 0 1 0 1 1 1 1 0 1 0 2 0 0 1 2 0 0 2 3 2 2 0 3 0 3 0 2 3
 2 2 2 3 2 3 1 0 2 0 1 3 2 3 3 1 3 0 0 3 1 3 3 2 1 3 0 2 1 1 3 0 1 3 0 0 3
 3 3 3 1 2 3 0 3 3 0 0 0 3 0 2 3 0 1 0 3 2 0 2 3 2 3 0 3 3 2 0 0 1 1 3 2 1
 1 0 1 0 3 2 2 3 3 2 3 1 0 3 1 0 2 0 1 3 1 2 2 2 2 0 0 2 3 0 1 3 0 0 0 1 1
 2 3 3 0 1 2 0 3 2 3 1 1 0 0 3 1 1 1 3 2 2 1 1 3 1 1 1 2 0 2 3 1 1 2 2 2 1
 1 3 2 0]
```

군집 - (1) k-평균 군집화

- K-means 군집화- 결과 시각화

```
# 그룹별로 색깔을 달리해 표현하기, 군집 중앙 표시
plt.scatter(X[:,0], X[:,1], c=y_kmeans, s=50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:,1], c='black', s=200, alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x2427ddb3c88>
```



군집 - (1) k-평균 군집화

- K-means 군집화 알고리즘
  - 기대값-최대화(E-M) 알고리즘

① 일부 군집 중심을 추측한다.(난수 초기값)

② 수렴될 때까지 다음을 반복한다.

✓ E-단계(기댓값 단계): 점을 가장 가까운 군집 중심에 할당한다.

✓ M-단계(최대화 단계): 군집 중심을 평균값으로 설정한다.

(군집에 속한 데이터의 산술 평균)

군집 - (1) k-평균 군집화

- 기대값-최대화(E-M) 알고리즘 관련 주의사항
  - ✓ 최초의 군집 중심을 난수 초기값으로 정하기 때문에 최적화된 결과를 얻지 못하는 경우도 있다.
  - ✓ 군집의 개수가 사전에 정해져야 한다.
  - ✓ K-평균 군집은 선형 군집 경계로 한정된다.
  - ✓ K-평균 균집은 표본 수가 많아지면 느려진다
    - 알고리즘을 반복할 때마다 데이터세트의 모든 점에 접근해야 하므로

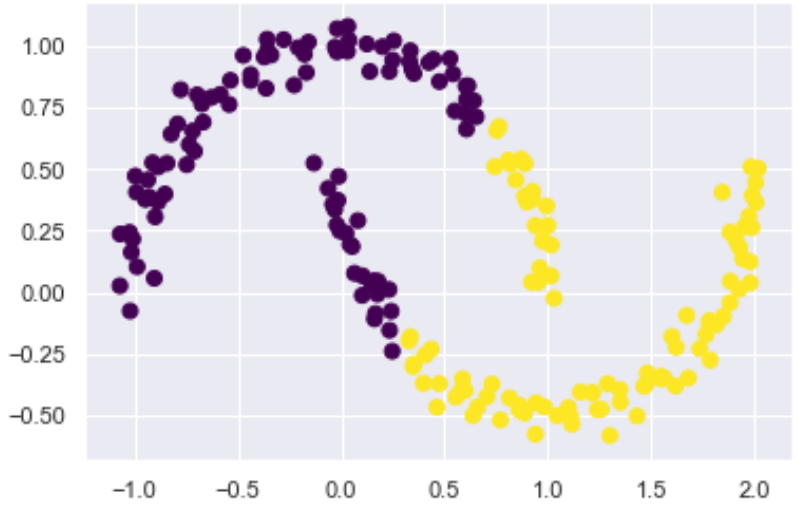
군집 - (1) k-평균 군집화

✓ K-평균 군집은 선형 군집 경계로 한정된다.

```
#비선형 경계를 가지는 자료의 경우
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=0.05, random_state = 0)
```

```
labels = KMeans(2, random_state =0).fit_predict(X)
plt.scatter(X[:,0], X[:,1], c=labels, s=50, cmap='viridis')
```

```
<matplotlib.collections.PathCollection at 0x2427de1da20>
```



Scikit-learn을 사용한 k-평균 군집화

- 필기체 숫자 인식에의 적용

(1) 데이터의 준비

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

```
(1797, 64)
```

Scikit-learn을 사용한 k-평균 군집화

- 필기체 숫자 인식에의 적용

(2) 학습 및 예측

```
# k-means clustering
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
```

```
kmeans.cluster_centers_.shape
```

```
(10, 64)
```

```
1  # 64차원의 군집 10개
```

Scikit-learn을 사용한 k-평균 군집화

- 필기체 숫자 인식에의 적용

(3) 클러스터 중심 확인

```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```

#2

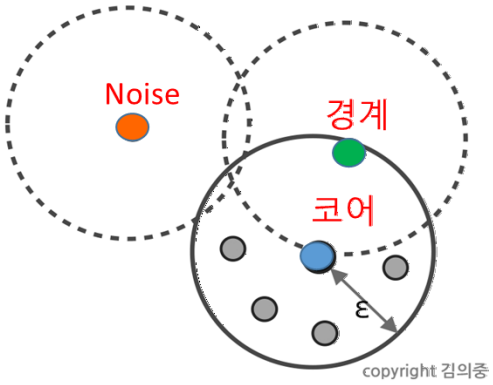
DBSCAN 분류기

군집-(2) DBSCAN 군집화

- DBSCAN
  - Density Based Spatial Clustering of Application with Noise
- 특징
  - 노이즈에 강한 군집 모델
  - 밀도있게 연결되어 있는 데이터 집합을 동일한 클러스터로 결정함
  - 일정한 밀도를 가지는 데이터 무리가 체인처럼 연결되어 있으면 거리의 개념과 관계없이 같은 클러스터로 판단함

군집-(2) DBSCAN 군집화

- DBSCAN 용어



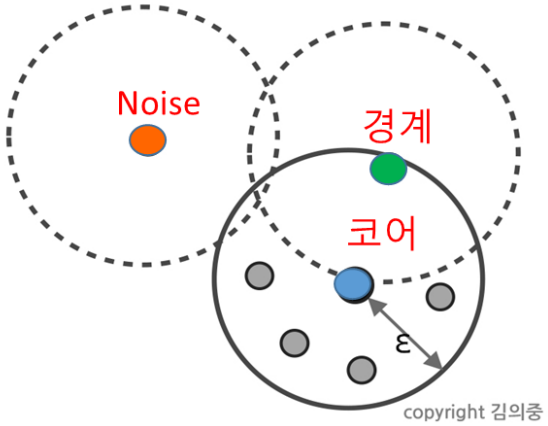
X : 학습 데이터 전체 집합
ε : 밀도측정 반지름
$MinPts$: 반지름 ε 이내에 있는 최소 데이터 개수
N(x) : 데이터 x의 반지름 ε 내에 있는 이웃 데이터(neighbor)
{x} : 데이터 x의 반지름 ε 내에 있는 이웃 데이터

- x is x_{core} if $N(x) \geq MinPts \; \forall_{x \in X}$
- x is x_{border} if $x \in \{x_{core}\}$ 이고 $N(x) < MinPts \; \forall_{x \in X}$
- x is x_{noise} if $x \notin \{x_{core}\}$ 이고 $N(x) < MinPts \; \forall_{x \in X}$

군집-(2) DBSCAN 군집화

- DBSCAN 알고리즘

1. 밀도 반지름 ε 반경 내 최소 데이터 개수(MinPts) 정의, C
2. 모든 데이터 x $\in$X에 대하여 다음을 수행
   1) x에 처음 방문하면 방문했다고 표시
   2) 만약 N(x) < MinPts 이면
      ① x는 Noise, C=C+1
      ② 2단계로 돌아가 다른 데이터로 다시 시작 (코어가 없으면
   3) 만약 N(x) > MinPts 이면
      ① x는 코어
      ② x가 아직 소속 클러스터가 없으면 C 할당
      ③ x의 밀도 반지름에 속해있는 모든 점들에 대해 2단계 반복



✓ 군집화의 과정은 코어→코어→코어→ …→경계 방향

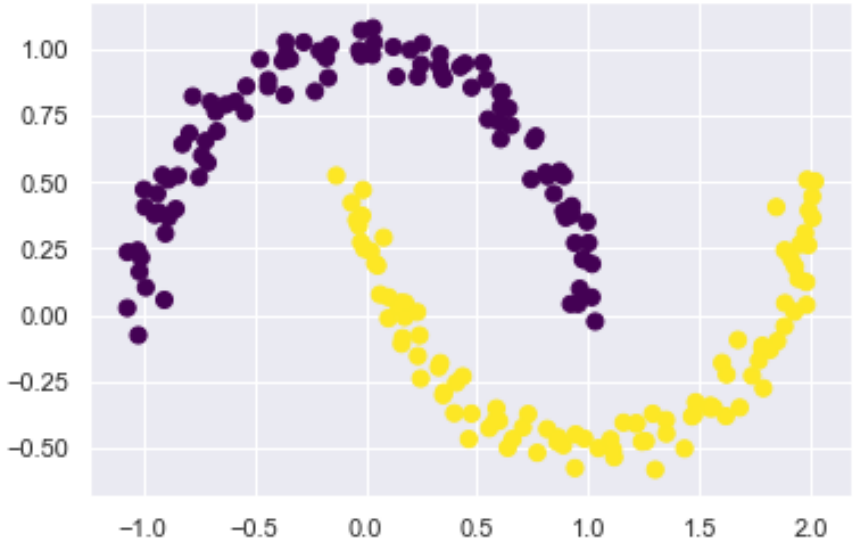
군집-(2) DBSCAN 군집화

- DBSCAN

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

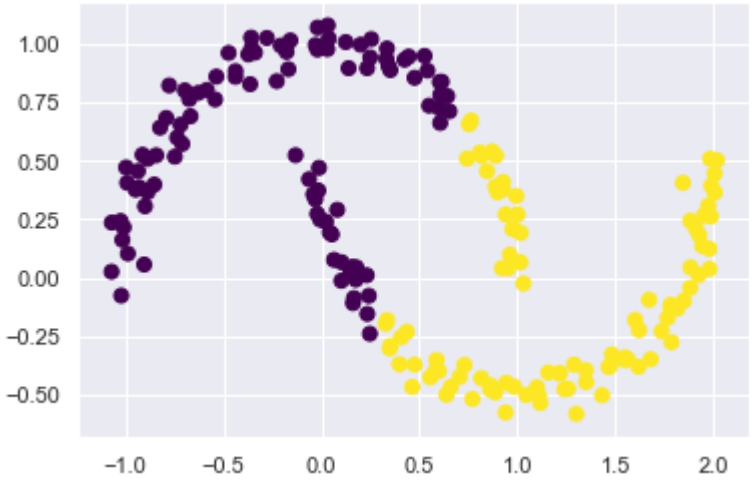
```
from sklearn.cluster import DBSCAN
D_labels = DBSCAN(eps=0.3, min_samples= 15).fit_predict(X)
plt.scatter(X[:,0], X[:,1], c=D_labels, s=50, cmap='viridis')
```

```
<matplotlib.collections.PathCollection at 0x214310661d0>
```

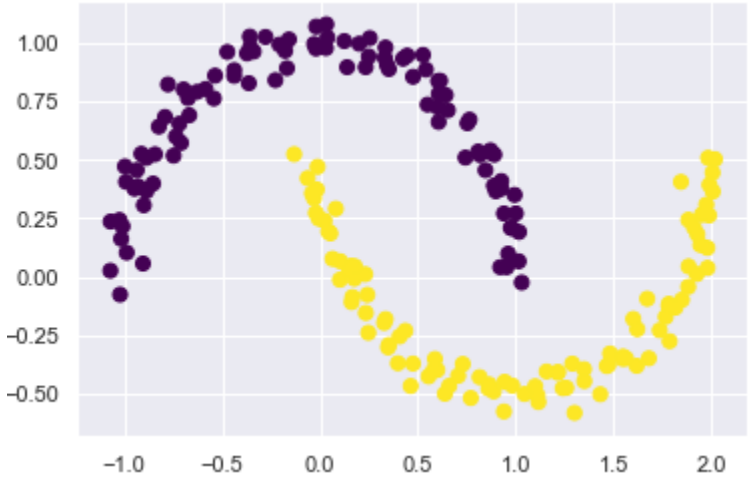


군집-(2) DBSCAN 군집화

- 장점
  - 도넛 모양이나 반달 모양의 데이터 세트에 대한 군집 가능



K-Means

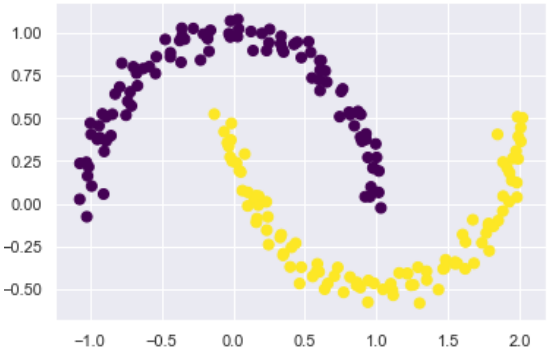


DBSCAN

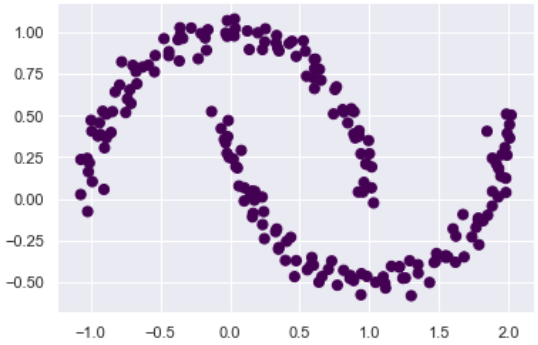
군집-(2) DBSCAN 군집화

- 단점
  - 밀도 반지름 및 최소 이웃 수가 문제의 특성에 따라 민감하게 작용함

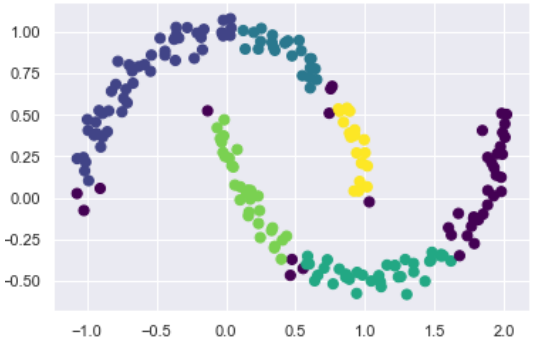
Eps=0.3
Min_sample=15



Eps=0.5
Min_sample=5



Eps=0.3
Min_sample=20



#3

주성분 분석 (PCA)

비지도 학습: PCA

- PCA(Principal Component Analysis)
  - 주성분 정보를 벡터와 길이로 분석
  - 활용 분야
    ① 주성분 분석:
      - 데이터의 주축(principal axes)의 목록을 구하고, 그 축을 사용해 데이터 세트를 설명
      - 특징 추출

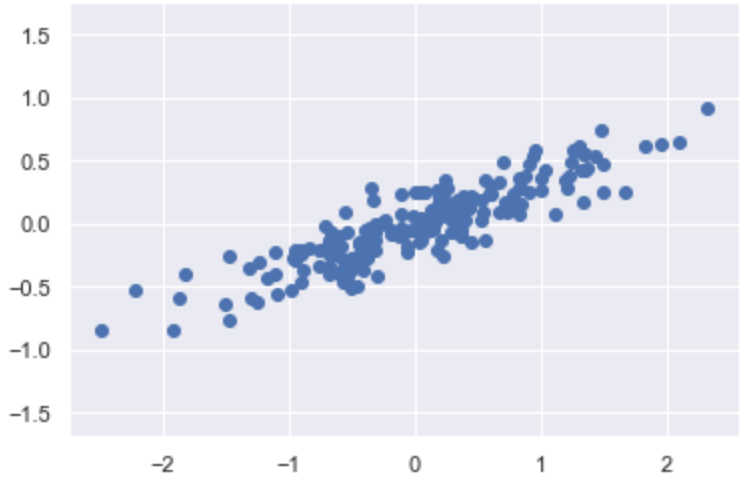
    ② 차원 축소:
      - 데이터의 분산 정보를 가장 많이 포함하는 주축으로 차원 축소

비지도 학습: PCA

(1) 주성분 분석

```
#PCA 를 위한 자료 준비
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2,2), rng.randn(2, 200)).T
plt.scatter(X[:,0], X[:, 1])
plt.axis('equal')
```

```
(-2.7391278364515688,
 2.5801310701596343,
 -0.9477947579593763,
 1.0195904306706842)
```



비지도 학습: PCA

(1) 주성분 분석

① 성분(component):벡터의 방향

② 설명 분산(explained variance): 해당 벡터의 제곱 길이

```
from sklearn.decomposition import PCA
mypca = PCA(n_components = 2)
mypca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

matplotlib.pyplot.gca(**kwargs)

Get the current Axes instance on the current figure r

```
print(mypca.components_ )
```

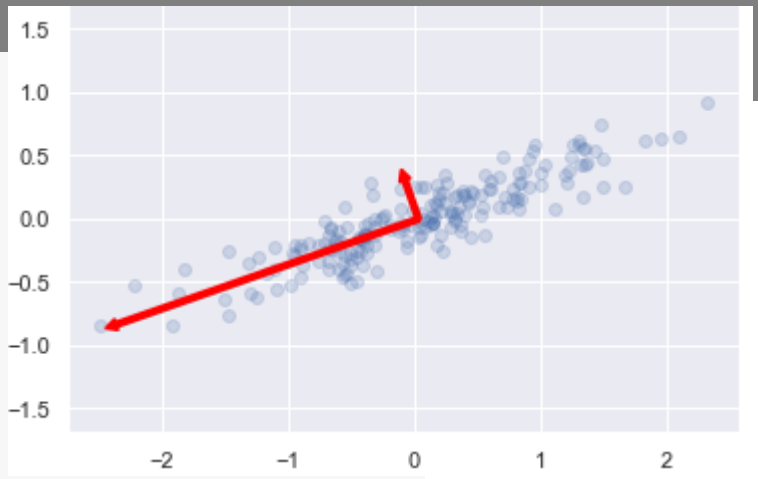
```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

```
print( mypca.explained_variance_ )
```

```
[0.7625315 0.0184779]
```

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops = dict(color='red',
                      arrowstyle='simple',
                      linewidth=2,
                      shrinkA=0, shrinkB=0 )
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# data plotting
plt.scatter(X[:, 0], X[:,1], alpha=0.2)
for length, vector in zip(mypca.explained_variance_, mypca.components_):
    v = vector * 3* np.sqrt(length)
    draw_vector(mypca.mean_, mypca.mean_ + v)
plt.axis('equal')
```




```
print(mypca.components_ )
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

```
print( mypca.explained_variance_ )
```

```
[0.7625315 0.0184779]
```

비지도 학습: PCA

(2) 차원 축소에의 응용

- 가장 작은 주성분 중 하나를 삭제해 최대 데이터 분산을 보존하는 더 작은 차원으로 데이터를 사영함.

```
dimpca = PCA(n_components=1)
dimpca.fit(X)
X_pca = dimpca.transform(X)
print('original shape: ', X.shape)
print('transformed shape: ', X_pca.shape)
```

```
original shape:  (200, 2)
transformed shape:  (200, 1)
```

비지도 학습: PCA

(2) 차원 축소

- 자료 출력

```
# 역변환
X_new = dimpca.inverse_transform(X_pca)
plt.scatter(X[:,0], X[:,1], alpha = 0.2)
plt.scatter(X_new[:,0], X_new[:,1], alpha=0.8)
plt.axis('equal')
```

0.0: 완전투명

```
(-2.77152878069022, 2.661757596590677, -0.9964674432667127, 1.0219081775900811)
```

비지도 학습: PCA

(2) 특징 추출: 얼굴 특징 추출

```
# 고유 얼굴 성분 찾기
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

비지도 학습: PCA

(2) 특징 추출: 얼굴 특징 추출

```
from sklearn.decomposition import PCA
face_pca = PCA(150)
face_pca.fit(faces.data)
```

```
PCA(copy=True, iterated_power='auto', n_components=150, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

비지도 학습: PCA

(2) 특징 추출: 얼굴 특징 추출

```
fig, axes = plt.subplots(3,8, figsize=(9,4),
                         subplot_kw={'xticks':[], 'yticks':[]},
                        gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(face_pca.components_[i].reshape(62,47), cmap='bone')
```

bone

Random 함수 설명

numpy.random.rand

```
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

`numpy.random.rand(d0, d1, ..., dn)`

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)` .

Parameters:	**d0, d1, ..., dn** : int, optional The dimensions of the returned array, should all be positive. If no argument is given a single Python float is returned.
Returns:	**out** : ndarray, shape `(d0, d1, ..., dn)` Random values.

numpy.random.randn(*d0, d1, ..., dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, randn generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

matplotlib.pyplot.gca(***kwargs*)

Get the current `Axes` instance on the current figure r

enumerate 함수 설명

script.py IPython Shell

```
1   grocery = ['bread', 'milk', 'butter']
2
3   for item in enumerate(grocery):
4     print(item)
5
6   print('\n')
7   for count, item in enumerate(grocery):
8     print(count, item)
9
10  print('\n')
11  # changing default start value
12  for count, item in enumerate(grocery, 100):
13    print(count, item)
```

script.py IPython Shell

```
(0, 'bread')
(1, 'milk')
(2, 'butter')


0 bread
1 milk
2 butter


100 bread
101 milk
102 butter
```

https://www.programiz.com/python-programming/methods/built-in/enumerate

#3

성능 평가

평가

- 예측 결과 평가 종류
  - TP(True Positive): 실제 양성인데, 검사 결과 양성
  - TN(True Negative): 실제는 음성인데, 검사 결과 음성
  - FP(False Positive): 실제는 음성인데 검사결과는 양성(거짓 양성)
  - FN(False Negative): 실제는 양성인데, 검사결과는 음성(거짓 음성)

	실제 양성	실제 음성
검사 양성	TP	FP
검사 음성	FN	TN

평가

	실제 양성	실제 음성
검사 양성	TP	FP
검사 음성	FN	TN

- 성능
  - 정확도(accuracy) = $\frac{TP+TN}{TP+TN+FP+FN}$
  - 정밀도(precision) = $\frac{TP}{TP+FP}$
  - 재현율(recall) = $\frac{TP}{TP+FN}$
  - 민감도(Sensitivity) = $\frac{TP}{TP+FP}$
  - 특이도(Specificity) = $\frac{TN}{TN+FP}$

scikit-image

https://scikit-image.org/docs/dev/api/skimage.data.html

Download Gallery Documentation Community Guidelines Source

Search documentation ...

Docs for 0.16.dev0
All versions

Module: data ¶

Standard test images.

For more images, see

- http://sipi.usc.edu/database/database.php

skimage.data.load (f[, as_gray])	Load an image file located in the data directory.
skimage.data.astronaut ()	Color image of the astronaut Eileen Collins.
skimage.data.binary_blobs ([length, ...])	Generate synthetic binary image with several rounded blob-like
skimage.data.brick ()	Brick wall.
skimage.data.camera ()	Gray-level "camera" image.
skimage.data.checkerboard ()	Checkerboard image.
skimage.data.chelsea ()	Chelsea the cat.
skimage.data.clock ()	Motion blurred clock.
skimage.data.coffee ()	Coffee cup.
skimage.data.coins ()	Greek coins from Pompeii.