

(12) 행렬의 랭크

행렬의 랭크는 행이나 열로 구성된 벡터 공간의 차원입니다. 넘파이의 선형대수 메소드인 `matrix_rank`로 차원을 쉽게 구할 수 있습니다.

```
[1] import numpy as np
matrix = np.array([[1, 1, 1],
                  [1, 1, 10],
                  [1, 1, 15]])
np.linalg.matrix_rank(matrix)
```

2

(13) 행렬식 계산하기

넘파이의 선형대수 메소드인 `det`를 사용합니다.

```
[2] matrix = np.array([[1, 2, 3],
                      [2, 4, 6],
                      [3, 8, 9]])
np.linalg.det(matrix)
```

0.0

(14) 행렬의 대각원소 추출

행렬의 대각원소가 필요하면 넘파이의 `diagonal` 메소드로 쉽게 얻을 수 있습니다.

```
[3] matrix.diagonal()
array([1, 4, 9])
```

(15) 행렬의 대각합 계산

행렬의 대각합을 계산하려면 `trace` 메소드를 사용합니다.

```
[4] matrix.trace()
```

14

(16) 고윳값과 고유벡터

정방행렬의 고윳값과 고유벡터를 찾으려면 넘파이의 linalg.eig 메소드를 사용합니다.

```
[5] matrix = np.array([[1, -1, 3],
                      [1, 1, 6],
                      [3, 8, 9]])
eigenvalues, eigenvectors = np.linalg.eig(matrix)
```

```
[7] print(eigenvalues)
print(eigenvectors)
```

```
[13.55075847 0.74003145 -3.29078992]
[[-0.17622017 -0.96677403 -0.53373322]
 [-0.435951  0.2053623 -0.64324848]
 [-0.88254925  0.15223105  0.54896288]]
```

(17) 벡터의 내적 계산

두 벡터의 내적(dot product)을 구하려면 넘파이 도트 메소드를 사용합니다.

```
[8] vector_a = np.array([1,2,3])
vector_b = np.array([4,5,6])
np.dot(vector_a, vector_b)
```

32

(18) 행렬의 덧셈과 뺄셈

두 행렬을 더하고 빼기 위해서는 넘파이의 add와 subtract 메소드를 사용합니다.

```
[9] matrix_a = np.array([[1, 1, 1],
                         [1, 1, 1],
                         [1, 1, 2]])
matrix_b = np.array([[1, 3, 1],
                     [1, 3, 1],
                     [1, 3, 8]])
np.add(matrix_a, matrix_b)
```

```
array([[2, 4, 2],
       [2, 4, 2],
       [2, 4, 10]])
```

```
[10] np.subtract(matrix_a, matrix_b)
```

```
array([[0, -2, 0],
       [0, -2, 0],
       [0, -2, -6]])
```

행렬의 덧셈과 뺄셈은 +, - 연산자를 사용할 수도 있습니다.

```
[11] matrix_a + matrix_b
```

```
array([[2, 4, 2],
       [2, 4, 2],
       [2, 4, 10]])
```

(19) 행렬의 곱셈

두 행렬을 곱할려면 넘파이의 dot 메소드를 사용합니다.

```
[13] matrix_a = np.array([[1, 1],
                         [1, 2]])
matrix_b = np.array([[1, 3],
                     [1, 2]])
np.dot(matrix_a, matrix_b)

array([[2, 5],
       [3, 7]])
```

(20) 역행렬

정방행렬의 역행렬은 넘파이 선형대수 모듈의 `inv`메소드를 사용합니다.

```
[14] matrix = np.array([[1, 4],
                      [2, 5]])
np.linalg.inv(matrix)

array([[-1.66666667,  1.33333333],
       [ 0.66666667, -0.33333333]])
```

(21) 난수 생성

넘파이의 `random` 모듈을 사용합니다.

초기값을 지정하려면 `random` 모듈의 `seed` 메소드를 실행합니다.

```
[15] np.random.seed(0)
np.random.random(3)

array([0.5488135 , 0.71518937, 0.60276338])
```

위 그림은 0과 1사이에 있는 세 개의 난수를 실수로 생성합니다.

다음은 1과 10 사이에 있는 세 개의 정수 난수를 생성합니다.



```
np.random.randint(0, 11, 3)
```

```
array([3, 7, 9])
```

다음 그림은 평균이 0.0이고 표준편차가 1.0인 정규분포에서 세 개의 수를 뽑는 모습을 보여줍니다.



```
np.random.normal(0.0, 1.0, 3)
```

```
array([-1.42232584, 1.52006949, -0.29139398])
```

다음 그림은 정규분포가 아닌 로지스틱 분포에서 세 개의 수를 뽑는 모습입니다.



```
np.random.logistic(0.0, 1.0, 3)
```

```
array([-0.98118713, -0.08939902, 1.46416405])
```

다음은 1.0 보다 크거나 같고, 2.0보다 작은 세 개의 수를 뽑는 방법을 보여 줍니다.



```
np.random.uniform(1.0, 2.0, 3)
```

```
array([1.47997717, 1.3927848 , 1.83607876])
```

이 예제들을 따라하다보면 난수임에도 결과가 동일함을 관찰할 수 있습니다. 이렇게 되는 이유는 난수의 초기값을 고정시켰기 때문입니다. 동일한 초기값을 가진 난수의 생성은 항상 같은 결과를 나타냅니다.

2장. 데이터 적재

(1) 샘플 데이터셋 적재하기

사이킷런의 데이터셋을 적재하는 예를 들어보겠습니다. 사이킷런의 데이터셋은 아래 그림처럼 사이킷런에서 `datasets`를 임포트합니다. `load_digits` 메소드는 손글씨 이미지들을 불러들입니다. 인쇄를 할 경우 전체 자료 중 일부를 보여줍니다.

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

[[ 0.,  0.,  5., ... 0.,  0.,  0.]
 [ 0.,  0.,  0., ... 10.,  0.,  0.]
 [ 0.,  0.,  0., ... 16.,  9.,  0.]
 ...
 [ 0.,  0.,  1., ... 6.,  0.,  0.]
 [ 0.,  0.,  2., ... 12.,  0.,  0.]
 [ 0.,  0.,  10., ... 12.,  1.,  0.]]
```

데이터셋에서 **feature**는 독립변수를 의미합니다. 아래 그림에서 **features** 하나의 자료는 8X8 손글씨 이미지의 내용을 나타냅니다.

```
▶ features = digits.data
    features[0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
       0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

아래 그림처럼 숫자 이미지를 불러올 때 X(독립변수), y(종속변수)에 각각 저장하라는 것입니다. **n_class=5**로 제한하여 0~9의 수 중 5개만 가지고 온다는 의미입니다.

```
▶ X, y = datasets.load_digits(n_class=5, return_X_y=True)
    np.unique(y)
```

```
array([0, 1, 2, 3, 4])
```

이번에는 선형회귀를 위해 모의데이터셋을 구성해보겠습니다. 선형회귀는 선형방정식으로 자료를 예측할 수 있습니다. 독립변수가 많으면 혼동의 우려가 있으므로 독립변수 1개와 종속변수로 1차방정식을 가정합니다:

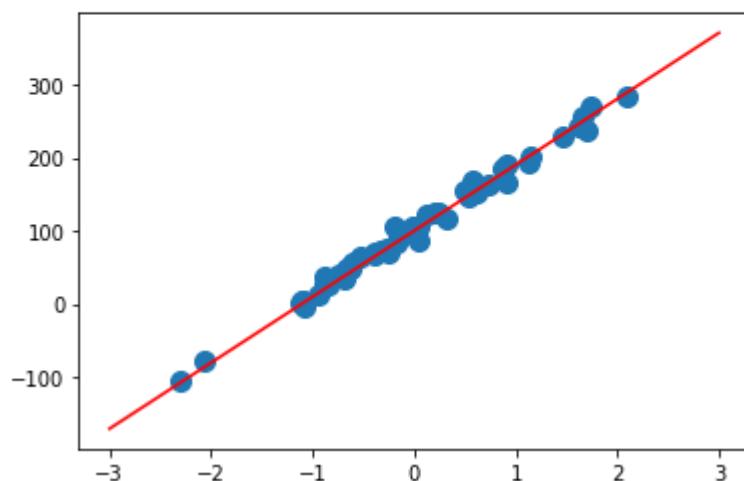
$$y = cX + b$$

`make_regression` 클래스로 선형방정식을 구성할 수 있습니다. 이 때 샘플의 갯수를 50개, 독립변수를 1개, 우리가 흔히 말하는 절편을 100, 노이즈를 10으로 둡니다. 노이즈가 0이면 완전한 직선이 됩니다.

```
[38] from sklearn.datasets import make_regression
X, y, c = make_regression(
    n_samples = 50, n_features = 1, noise = 10.0,
    bias = 100, coef = True, random_state = 1)
print('특성 행렬\n', X[:3])
print('타깃 벡터\n', y[:3])

import matplotlib.pyplot as plt
xw = np.linspace(-3, 3, 100)
yw = c * xw + 100
plt.plot(xw, yw, "r-")
plt.scatter(X, y, s=100)
plt.show()
```

특성 행렬
[[-0.24937038]
[0.19091548]
[-0.68372786]]
타깃 벡터
[68.98939337 124.09570635 38.94026069]



`linspace`의 매개변수 중 시작값 -3, 종료값 3, 간격을 100으로 높습니다. 또한 `plot` 매개 변수 중 세번째는 포맷 문자열로 선의 색상과 형태를 지정합니다. “r-”는 빨간색의 실선을 나타냅니다.

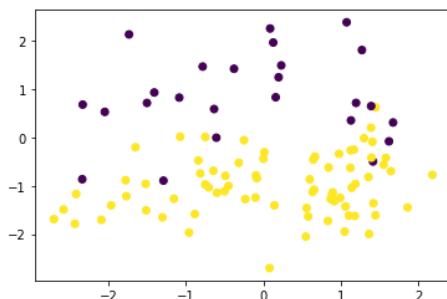
다음 모의데이터셋은 분류용 자료를 불러온 것입니다.

```
[43] from sklearn.datasets import make_classification
features, target = make_classification(n_samples = 100,
n_features = 3, n_informative = 3, n_redundant = 0,
n_classes = 2, weights = [.25, .75], random_state = 1)
print('특성 행렬\n', features[:3])
print('타깃 벡터\n', target[:3])
```

특성 행렬
[[1.06354768 -1.42632219 1.02163151]
[0.23156977 1.49535261 0.33251578]
[0.15972951 0.83533515 -0.40869554]]
타깃 벡터
[1 0 0]

`weights`는 분류시 한 쪽의 비중을 25%로 했다면, 다른쪽의 비중을 75%로 구성한다는 뜻입니다.

```
[44] plt.scatter(features[:,0], features[:,1], c=target)
plt.show()
```



군집(clustering) 알고리즘에 적용할 데이터셋이 필요할 경우 `make_blobs`를 사용합니다. 군집의 갯수는 `centers`로 지정할 수 있습니다.

```
[28] from sklearn.datasets import make_blobs
features, target = make_blobs(n_samples = 100,
                               n_features = 2,
                               centers = 3,
                               cluster_std = 0.5,
                               shuffle = True,
                               random_state = 1)
print('특성 행렬\n', features[:3])
print('타깃 벡터\n', target[:3])
```

특성 행렬
[[-1.22685609 3.25572052]
 [-9.57463218 -4.38310652]
 [-10.71976941 -4.20558148]]
타깃 벡터
[0 1 1]

다음 그림은 위 군집 모의데이터셋을 그림으로 나타낸 것입니다.

```
[29] import matplotlib.pyplot as plt
plt.scatter(features[:,0], features[:,1], c=target)
plt.show()
```

