| Matplotlib

1. 파이썬의 다양한 그래프 그리기를 구현하는 라이브러리이다.
2. 꺽은선 그래프와 산포도 등에 상세하게 표시를 설정할 수 있다.
3. 학습 결과를 그래프로 표시할 때 사용한다.

난수 기반 배열 생성

1. NumPy는 난수 발생 및 배열 생성을 생성하는 numpy.random 모듈을 제공한다.
    1. np.random.normal
    2. np.random.rand
    3. np.random.randn
    4. np.random.randint
    5. np.random.random

np.random.normal

np.random.normal

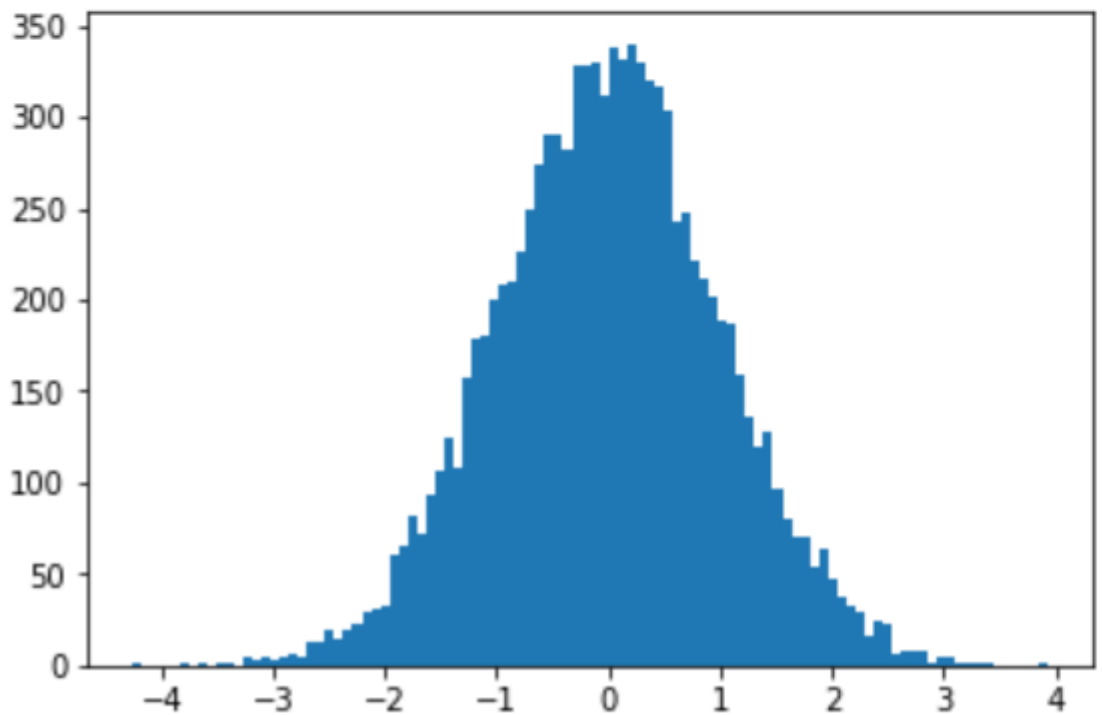
- normal(loc=0.0, scale=1.0, size=None)
- 정규 분포 확률 밀도에서 표본 추출
- loc: 정규 분포의 평균
- scale: 표준편차

```
mean = 0
std = 1
a = np.random.normal(mean, std, (2, 3))
print(a)
```

```
[[ 0.38660907 -0.10453156  0.78794226]
 [ 0.53626446 -0.24135191  0.79789835]]
```

- np.random.normal이 생성한 난수는 정규 분포의 형상을 갖습니다.
- 다음 예제는 정규 분포로 10000개 표본을 뽑은 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 100개 구간으로 구분할 때, 정규 분포 형태를 보이고 있습니다.

```
data = np.random.normal(0, 1, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=100)
plt.show()
```



np.random.rand

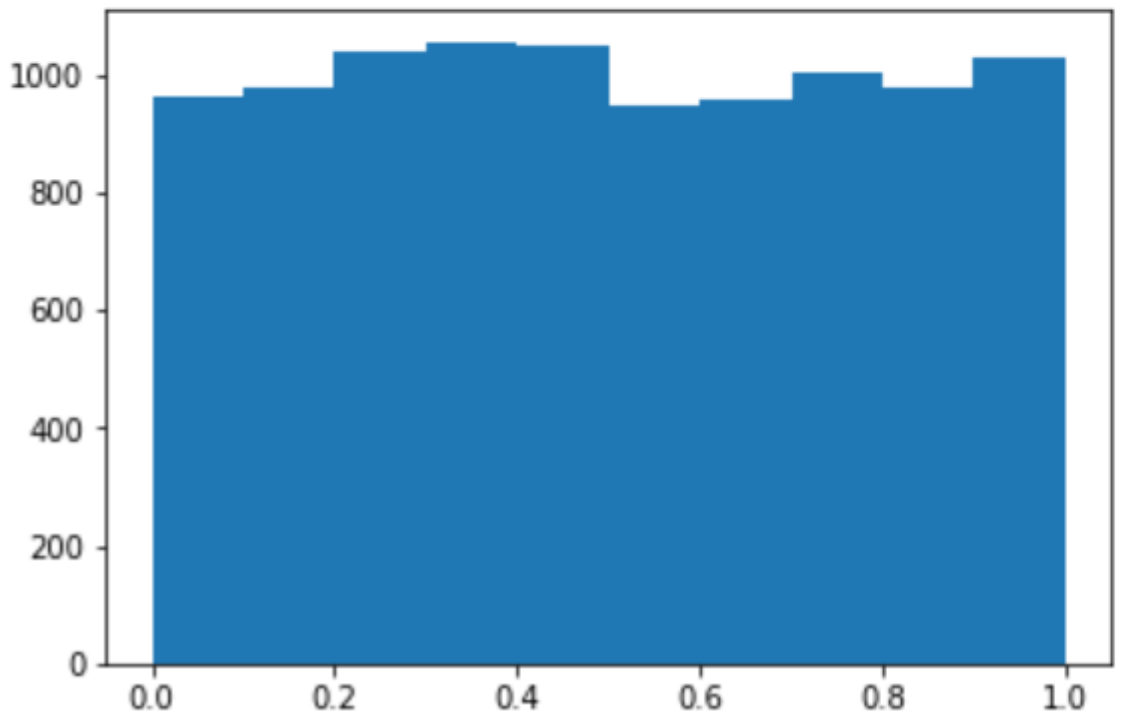
- numpy.random.rand(d0, d1, ..., dn)
- Shape이 (d0, d1, ..., dn) 인 배열 생성 후 난수로 초기화
- 균등 분포(Uniform Distribution) 형상으로 표본 추출
- Gaussina normal

```
a = np.random.rand(3,2)
print(a)
```

```
[[0.61463242 0.96086011]
 [0.6484841  0.04940953]
 [0.57119312 0.10391525]]
```

- np.random.rand는 균등한 비율로 표본 추출
- 다음 예제는 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 균등한 분포 형태를 보이고 있습니다.

```
data = np.random.rand(10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



np.random.randn

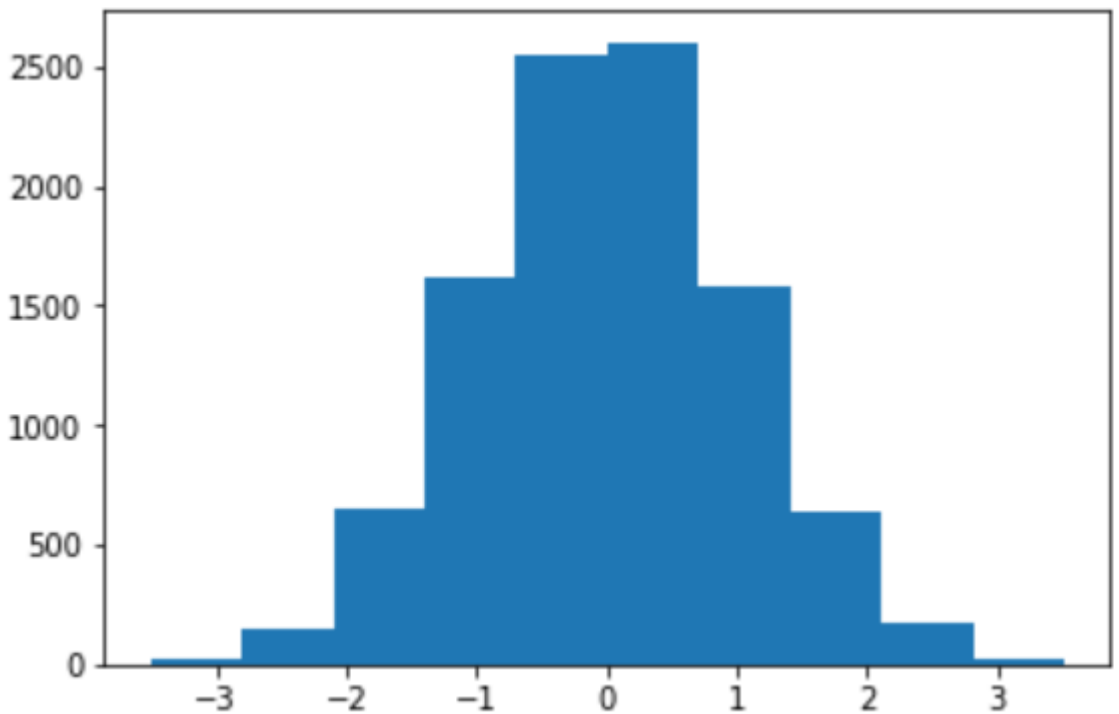
- numpy.random.randn(d0, d1, ..., dn)
- (d0, d1, ..., dn) shape 배열 생성 후 난수로 초기화
- 난수: 표준 정규 분포(standard normal distribution)에서 표본 추출

```
a = np.random.randn(2, 4)
print(a)
```

```
[[-1.87382728  1.46673698 -0.02607291  0.14752896]
 [ 0.08198552  0.74275047 -1.06008112  1.15105555]]
```

- np.random.randn은 정규 분포로 표본 추출
- 다음 예제는 정규 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 정규 분포 형태를 보이고 있습니다.

```
data = np.random.randn(10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



np.random.randint

- numpy.random.randint(low, high=None, size=None, dtype='l')
- 지정된 shape으로 배열을 만들고 low 부터 high 미만의 범위에서 정수 표본 추출

```
a = np.random.randint(5, 10, size=(2, 4))
pprint(a)
```

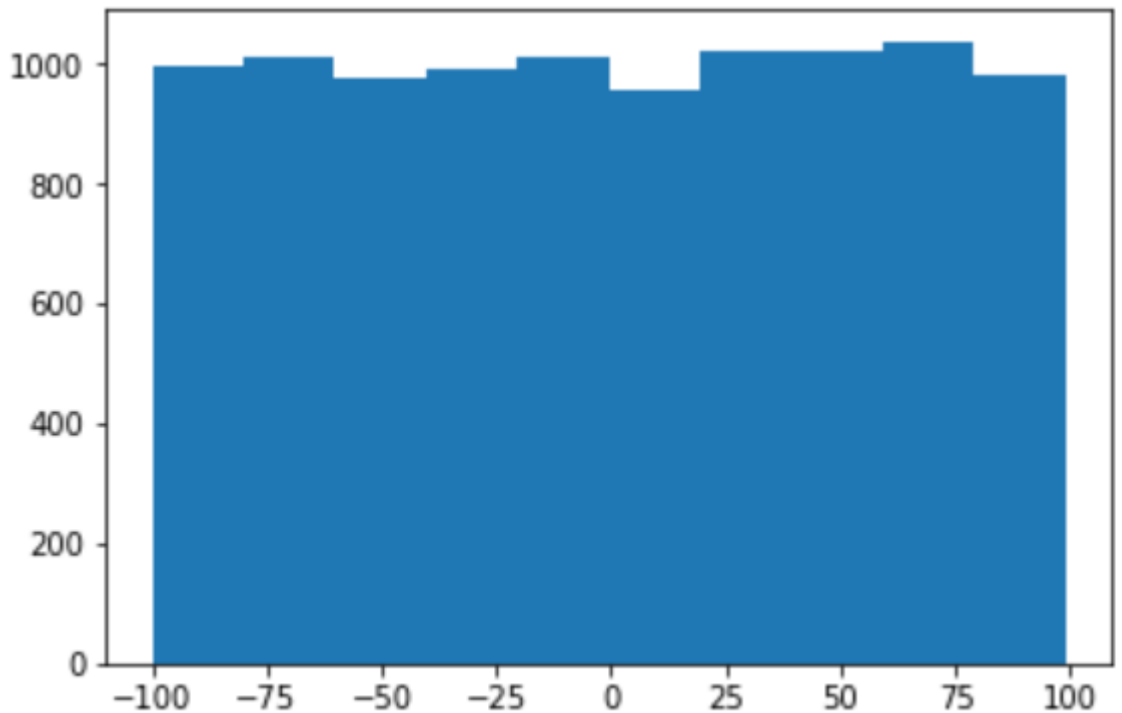
```
[[6 6 9 5]
 [6 8 7 6]]
```

```
a = np.random.randint(1, size=10)
print(a)
```

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

- 100에서 100의 범위에서 정수를 균등하게 표본 추출합니다.
- 다음 예제에서 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 균등한 분포 형태를 보이고 있습니다.

```
data = np.random.randint(-100, 100, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



np.ramdom.random

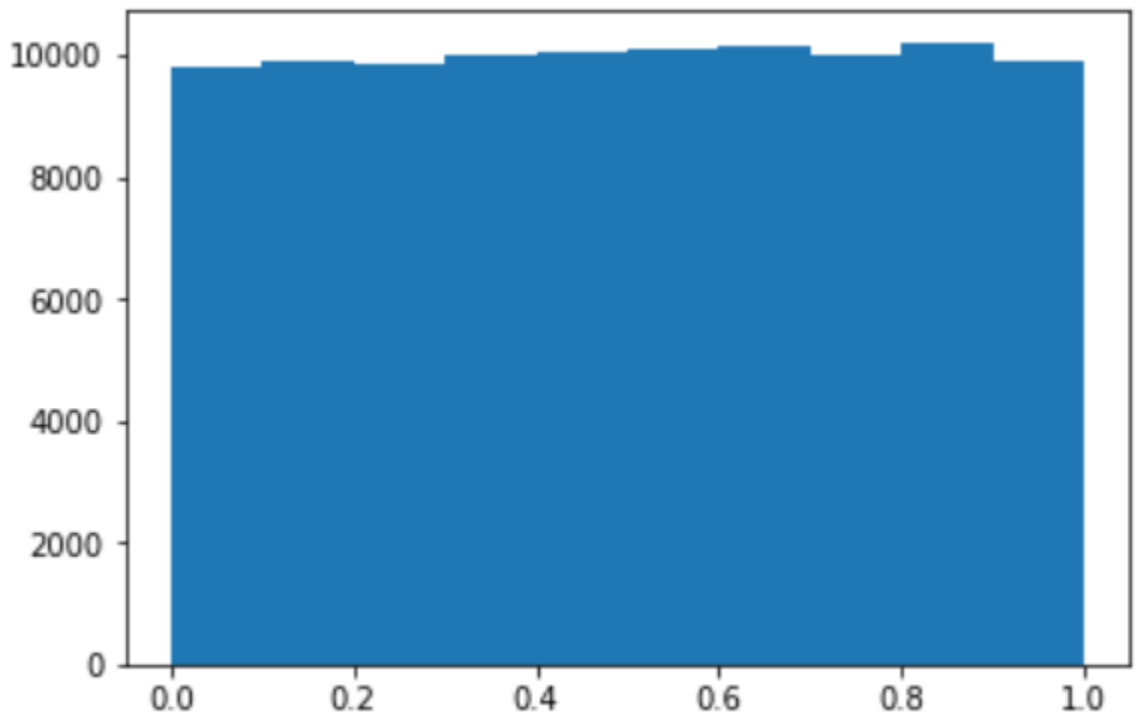
- np.random.random(size=None)
- 난수: [0., 1.)의 균등 분포(Uniform Distribution)에서 표본 추출

```
a = np.random.random((2, 4))
print(a)
```

```
[[0.00308614 0.16123964 0.5885869   0.01248252]
 [0.62078491 0.71088922 0.30589675 0.37935089]]
```

- np.random.random은 균등 분포로 표본을 추출합니다.
- 다음 예제는 정규 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 정규 분포 형태를 보이고 있습니다.

```
data = np.random.random(100000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



회귀문제

회귀 문제는 수치를 예측하는 지도 학습 문제이다.

학습할 때 입력 데이터와 출력 데이터셋에서 대응하는 규칙을 배우고 미지의 입력 데이터에서도 적절한 출력을 하도록 한다.

입력과 출력 관계(함수)를 추정하므로 근사 문제라고도 한다.

회귀 문제를 푸는 것은 주어진 데이터에 대해 관계를 나타내는 식을 가정하고, 해당 데이터에 가장 알맞은 식의 계수를 정해 나간다는 것이다.

| 회귀 문제의 분류

1. 회귀 문제는 구해야 하는 식의 형식이나 변수 개수로 분류한다.
    1. 선형 회귀
        1. 직선적인 관계를 추정하는 회귀이다.
        2. 핵심을 구해야 하는 대상이 선형이라는 것이며, 식 자체가 1차식인 것을 의미하지 않는다.
    2. 비선형 회귀
        1. 선형 이외의 모든 회귀
    3. 단순 회귀
        1. 입출력 관계가 변수 하나로 구성된 식(예:y=ax+b)을 상정해 푸는 회귀문제이다. 변수 개수가 문제이므로 다음 차수가 올라도 변수가 1개이면 단순 회귀
    4. 다중 회귀
        1. 변수를 2개 이상 쓰는 회귀

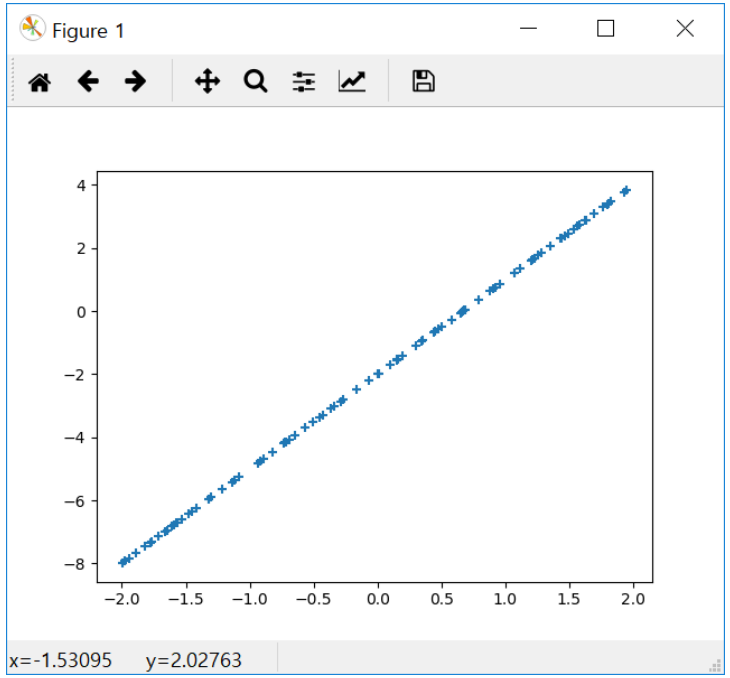
| 최소 제곱법과 평가 방법

1. 대상을 평면상의 점으로 하고, 그 관계를 가장 잘 나타내는 직선이 무엇인지 생각한다.
2. 모든 점을 지나는 직선을 발견하면 좋겠지만 실제로는 오차가 들어가므로, 진짜로는 직선 관계가 있더라도 선에서 벗어난 점이 발생한다.
3. 실제 값과 예측 값의 차이가 가장 작아지는 직선이 좋아 보이지만, 그 차이는 플러스와 마이너스 값이 모두 있기 때문에 단순히 모두 더하면 의도하는 값이 되지 않는다.
4. 최소제곱법의 아이디어??
    1. 실제 값과 예측 값 차이에 대한 제곱의 합을 최소화로 하는 것

scikit에는 최소제곱합을 구현한 sklearn.linear_model.LinearRegression이 있다.

| 선형 단순 회귀

```
import matplotlib.pyplot as plt
import numpy as np
### y = 3x -2 데이터 생성
x = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x = x * 4 - 2               # 값의 범위를 -2~2로 변경
y = 3 * x - 2  # y = 3x - 2
### 그래프 표시
plt.scatter(x, y, marker='+')
plt.show()
```



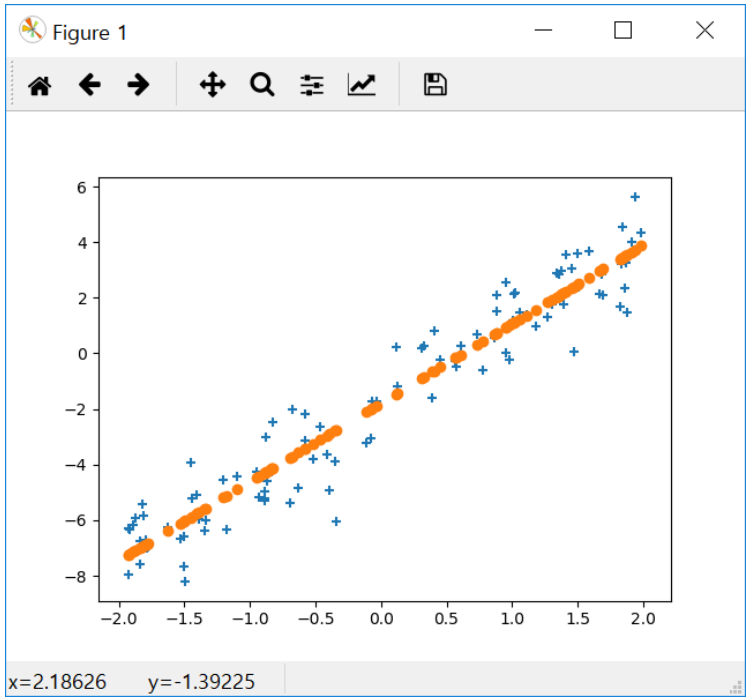
| 선형 단순 회귀

```
import matplotlib.pyplot as plt
import numpy as np
### y = 3x -2 데이터 생성
x = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x = x * 4 - 2               # 값의 범위를 -2~2로 변경
y = 3 * x - 2  # y = 3x - 2

### 학습
from sklearn import linear_model
model = linear_model.LinearRegression() #최소제곱법 계산
model.fit(x, y)
#모델 함수 fit을 호출하는 것으로 학습이 끝났다.

### 계수, 절편을 표시
print('계수', model.coef_)
print('절편', model.intercept_)

### 그래프 표시
plt.scatter(x, y, marker='+')
plt.show()
```

```
계수 [[3.]]
절편 [-2.]
```

```
import matplotlib.pyplot as plt
import numpy as np

### 분산 y = 3x -2 데이터를 생성
x = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x = x * 4 - 2               # 값의 범위를 -2~2로 변경
y = 3 * x - 2  # y = 3x - 2

y += np.random.randn(100, 1)  # 표준 정규 분포(평균 0, 표준 편차 1)의 난수를 추가함

### 학습
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(x, y)

### 계수, 절편, 결정 계수를 표시
print('계수', model.coef_)
print('절편', model.intercept_)

### 그래프 표시
plt.scatter(x, y, marker ='+')
plt.scatter(x, model.predict(x), marker='o')
plt.show()
```

```
계수 [[2.83860346]]
절편 [-1.77590462]
```



| 회귀에서 평가 : 결정 계수

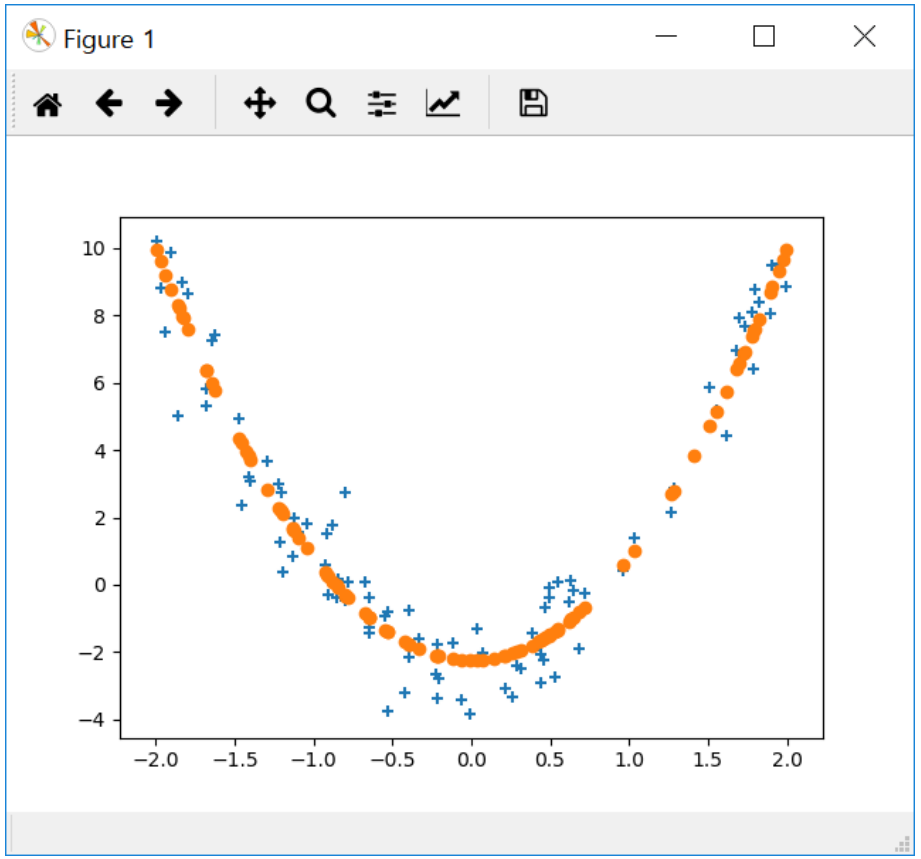
1. 회귀는 결과의 타당성을 객관적으로 평가하는 지표로 R^2 결정계수가 알려져 있다.
    1. R^2 = 1-(관측 값과 예측 값 차의 2 제곱 합/관측 값과 측정값 전체 평균의 2 제곱 합)
2. 관측된 값은 오차를 포함하므로 정답과 다르다. 관측 값과 예측 값이 참값에 가까우면 분자가 0에 가까워지면서 R^2 결정 계수는 1에 가까워 진다.
3. 값과 관측 값 차이가 크다면 분자는 0에서 멀어진다. 따라서 분수 부분도 0에서 멀어져 결과적으로 결정 계수는 1에서 멀어진 값이 된다.
4. 즉, 결정 계수의 값이 1에 가까울수록 그 예측 모델은 좋은 모델이라고 할 수 있다.
5. sklearn.linear_model의 각 클래스에는 score 함수가 있다. 이 함수는 그 모델의 결정 계수를 돌려준다.

```
r2 = model.score(x, y)
print('결정계수', r2)
```

```
결정계수 0.9114829569852603
```

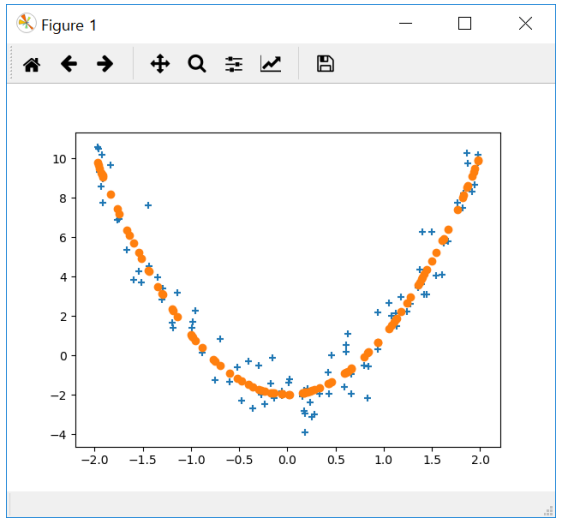
| $y=ax^2+b$를 구한다.

1. 2차 방정식 y=ax2+b를 상정한 데이터를 구해본다.
2. 처음부터 오차를 갖게 한다.



```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x = x * 4 - 2               # 값의 범위를 -2~2로 변경
y = 3 * x**2 - 2  # y=3x²-2
y += np.random.randn(100, 1)  # 표준 정규 분포(평균 0, 표준 편차 1)의 난수를 추가함
### 학습
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(x**2, y)  # x를 제곱해 전달
### 계수, 절편, 결정 계수를 표시
print('계수', model.coef_)
print('절편', model.intercept_)
print('결정계수', model.score(x**2, y))
### 그래프 표시
plt.scatter(x, y, marker ='+')
plt.scatter(x, model.predict(x**2), marker='o')  # predict에도 x를 제곱해 전달
plt.show()
```

```
계수 [[3.02241352]]
절편 [-1.96892866]
결정계수 0.9412926946427712
```

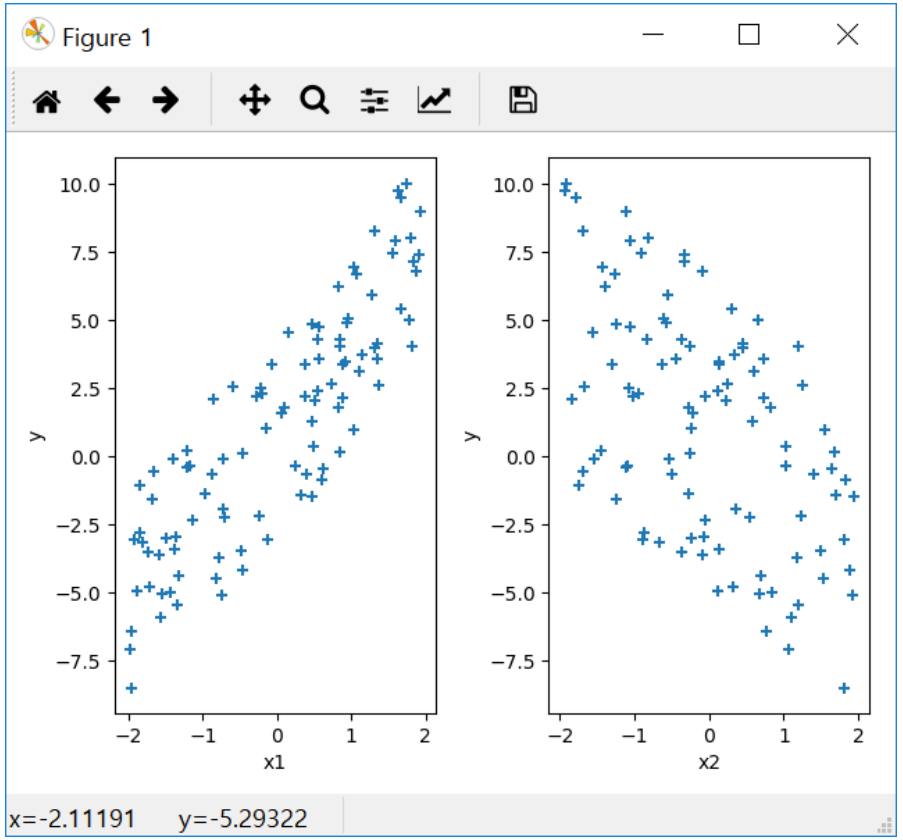


오차를 주기 전 식 $y=3x^2-2$에 가깝고 R^2결정계수도 0.94로 상당히 정밀도 높은 예측이다.

| 다중 회귀 시험

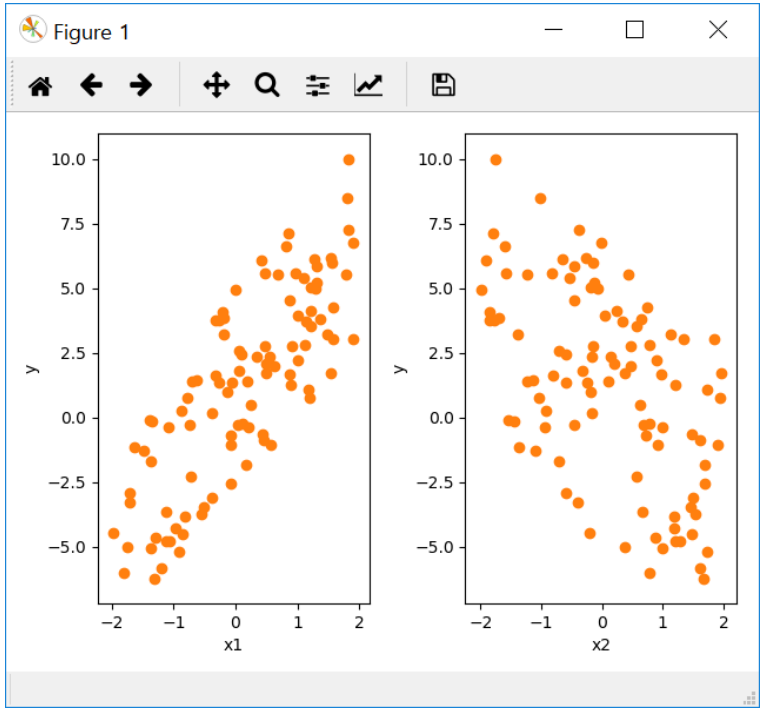
1. 다중 회귀는 변수를 여러 개 사용하는 식을 상정해 푸는 문제이다.
2. 예) $y=ax_1+bx_2+c$ 관계가 되는 데이터를 만들어 진짜로 회귀 문제를 풀 수 있는지 시험해 보자.
3. $y=3x_1-2x_2+1$

```
import matplotlib.pyplot as plt
import numpy as np
x1 = np.random.rand( 100, 1 )      # 0~1까지 난수를 100개 만든다
x1 = x1 * 4 - 2                    # 값의 범위를 -2~2로 변경

x2 = np.random.rand( 100, 1 )      # x2에 대해서도 마찬가지
x2 = x2 * 4 - 2

y = 3 * x1 - 2 * x2 + 1

plt.subplot( 1, 2, 1 )
plt.scatter( x1, y, marker='+' )
plt.xlabel( 'x1' )
plt.ylabel( 'y' )

plt.subplot( 1, 2, 2 )
plt.scatter( x2, y, marker='+' )
plt.xlabel( 'x2' )
plt.ylabel( 'y' )

plt.tight_layout()
plt.show()
```

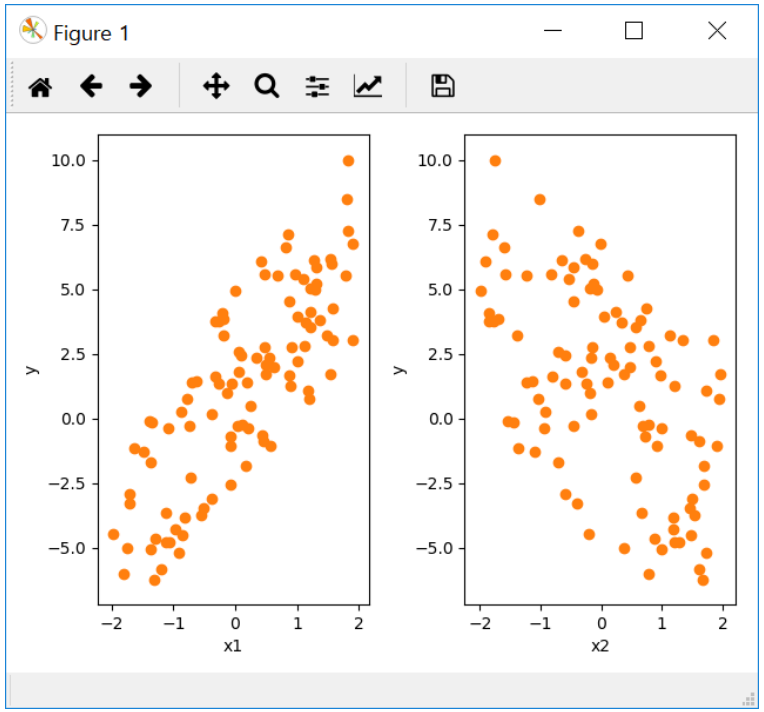


| x1,x2와 y관계에 대해 최소 제곱법

```
import matplotlib.pyplot as plt
import numpy as np

### y = 3x_1 - 2x_2 + 1 데이터 생성

x1 = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x1 = x1 * 4 - 2              # 값의 범위를 -2~2로 변경

x2 = np.random.rand(100, 1)  # x2에 대해서도 같게
x2 = x2 * 4 - 2
y = 3 * x1 - 2 * x2 + 1

### 학습
from sklearn import linear_model
#numpy.c_ :마지막 축을 기준으로 하나로 결합시키는 것
x1_x2 = np.c_[x1, x2]  # [[x1_1, x2_1], [x1_2, x2_2], ..., [x1_100, x2_100]]
                       # 형태로 변환
model = linear_model.LinearRegression()
model.fit(x1_x2, y)
```



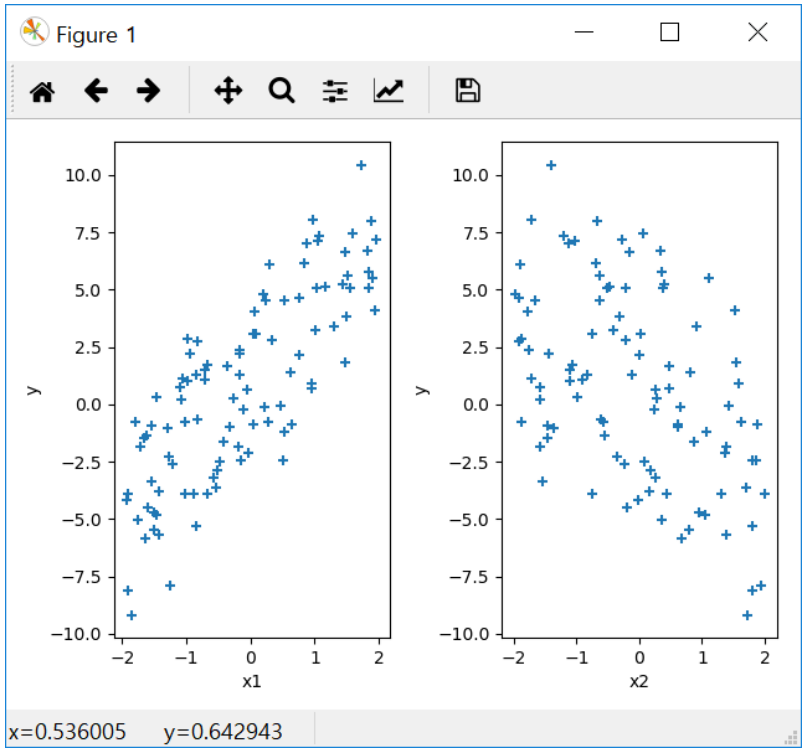
```
### 계수, 절편, 결정 계수를 표시
print('계수', model.coef_)
print('절편', model.intercept_)
print('결정계수', model.score(x1_x2, y))

### 그래프 표시
y_ = model.predict(x1_x2)  # 구한 회귀식으로 예측

plt.subplot(1, 2, 1)
plt.scatter(x1, y, marker='+')
plt.scatter(x1, y_, marker='o')
plt.xlabel('x1')
plt.ylabel('y')

plt.subplot(1, 2, 2)
plt.scatter(x2, y, marker='+')
plt.scatter(x2, y_, marker='o')
plt.xlabel('x2')
plt.ylabel('y')

plt.tight_layout()
plt.show()
```




```
계수 [[ 3. -2.]]
절편 [1.]
결정계수 1.0
```

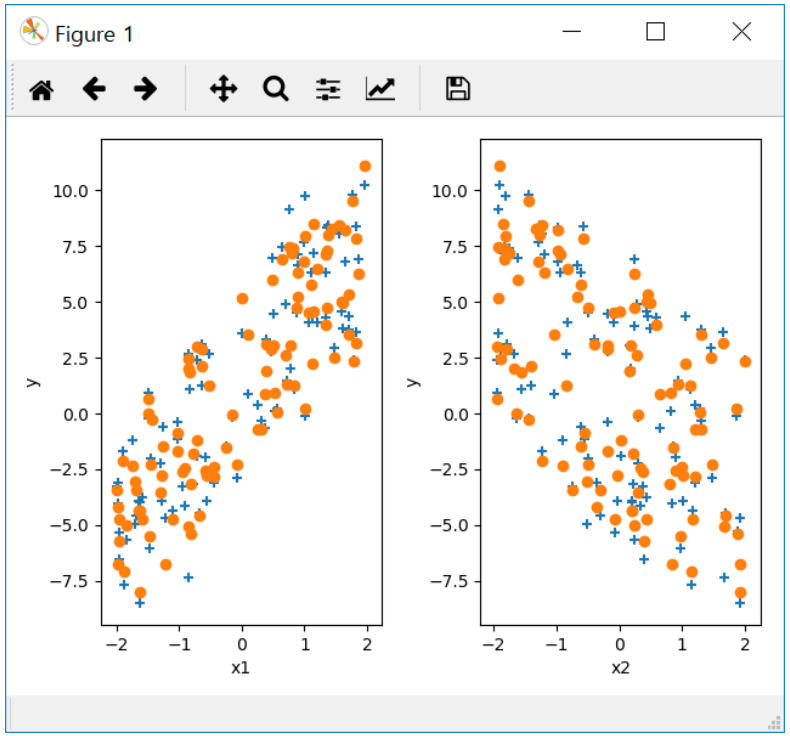
| 데이터에 오차를 주면?

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.random.rand( 100, 1 )   # 0~1까지 난수를 100개 만들
x1 = x1 * 4 - 2                 # 값의 범위를 -2~2로 변경

x2 = np.random.rand( 100, 1 )   # x2에 대해서도 마찬가지
x2 = x2 * 4 - 2

y = 3 * x1 - 2 * x2 + 1

y += np.random.randn( 100, 1 )    # 표준 정규 분포(평균0, 표준 편차1)
                                  # 난수를 더한다
```



```
plt.subplot( 1, 2, 1 )
plt.scatter( x1, y, marker='+' )
plt.xlabel( 'x1' )
plt.ylabel( 'y' )

plt.subplot( 1, 2, 2 )
plt.scatter( x2, y, marker='+' )
plt.xlabel( 'x2' )
plt.ylabel( 'y' )

plt.tight_layout()
plt.show()
```

| x1,x2와 y관계에 대해 최소 제곱법




```
import matplotlib.pyplot as plt
import numpy as np
### y = 3x_1 - 2x_2 + 1 데이터 생성
x1 = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x1 = x1 * 4 - 2              # 값의 범위를 -2~2로 변경
x2 = np.random.rand(100, 1)  # x2에 대해서도 같게
x2 = x2 * 4 - 2
y = 3 * x1 - 2 * x2 + 1
y += np.random.randn(100, 1)  # 표준 정규 분포(평균 0, 표준 편차 1)의 난수를 추가함

### 학습
from sklearn import linear_model
x1_x2 = np.c_[x1, x2]  # [[x1_1, x2_1], [x1_2, x2_2], ..., [x1_100, x2_100]]
                       # 형태로 변환
model = linear_model.LinearRegression()
model.fit(x1_x2, y)
```

```
### 계수, 절편, 결정 계수를 표시
print('계수', model.coef_)
print('절편', model.intercept_)
print('결정계수', model.score(x1_x2, y))

### 그래프 표시
y_ = model.predict(x1_x2)  # 구한 회귀식으로 예측
plt.subplot(1, 2, 1)
plt.scatter(x1, y, marker='+')
plt.scatter(x1, y_, marker='o')
plt.xlabel('x1')
plt.ylabel('y')
plt.subplot(1, 2, 2)
plt.scatter(x2, y, marker='+')
plt.scatter(x2, y_, marker='o')
plt.xlabel('x2')
plt.ylabel('y')
plt.tight_layout()
plt.show()
```

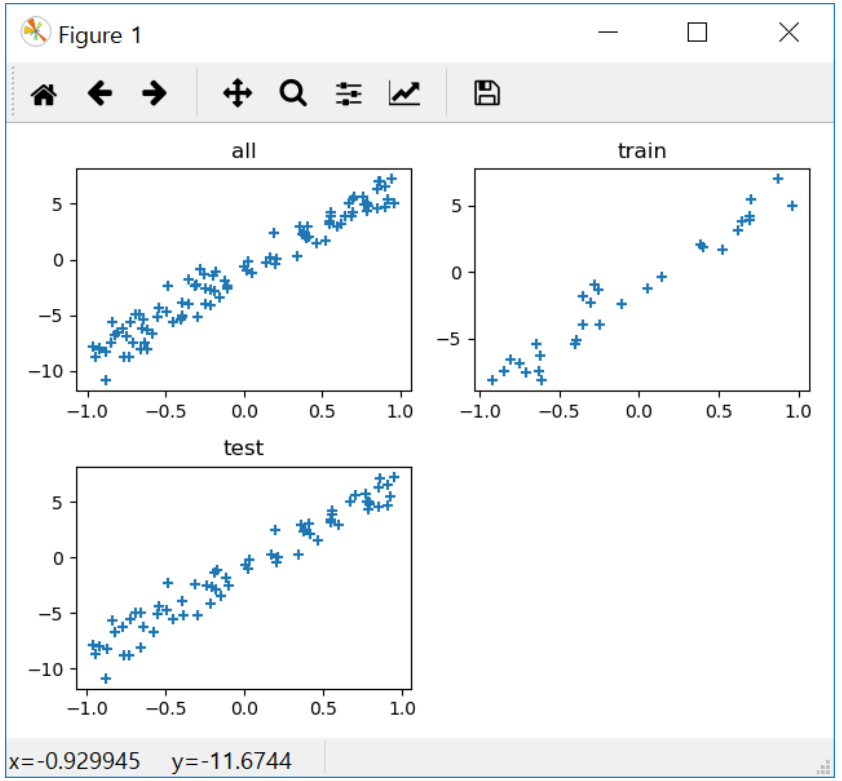
```
C:\JIN\Anaconda3\envs\tf\python.
계수 [[ 2.95431675 -1.89489344]]
절편 [0.8834315]
결정계수 0.9504760542053178
```

| 머신 러닝에서 피해야 하는 부분 : 과적합

1. 과적합
    1. 주어진 학습 데이터에 너무 적응해서 미지의 데이터에 적합하지 않은 상태
    2. 즉, 일반화 성능이 좋아지지 않는 것이다.
    3. 머신 러닝에서는 이 과적합을 어떻게 억제하고 일반화 성능을 확보할지가 큰 이슈이다.

    4. 오차가 있는 데이터로 학습 데이터 30개, 검증용 테스트 데이터 70개, 총 100개를 준비해 구현한다.
        1. $y = 4x^3-3x^2+2x-1$

```
x = np.random.rand(100, 1)       # 0~1까지 난수를 100개 만든다
x = x * 2 - 1                    # 값의 범위를 -2~2로 변경
y = 4 * x * 3 - 3 * x * 2 + 2 * x - 1
y += np.random.randn( 100, 1 )  # 표준 정규 분포(평균0, 표준 편차1)        #난수를 더한다
# 학습 데이터 30개
x_train = x[:30]
y_train = y[:30]
# 테스트 데이터 70개
x_test = x[30:]
y_test = y[30:]
plt.subplot( 2, 2, 1 )
plt.scatter( x, y, marker='+' )
plt.title( 'all' )
plt.subplot( 2, 2, 2 )
plt.scatter( x_train, y_train, marker='+' )
plt.title( 'train' )
plt.subplot( 2, 2, 3 )
plt.scatter( x_test, y_test, marker='+' )
plt.title( 'test' )
plt.tight_layout()
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
### y = 4x^3 - 3x^2 + 2x - 1 데이터 생성
x = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x = x * 2 - 1               # 값의 범위를 -2~2로 변경

y = 4 * x**3 - 3 * x**2 + 2 * x - 1

y += np.random.randn(100, 1)  # 표준 정규 분포(평균 0, 표준 편차 1)의 난수를 추가


# 학습 데이터 30개
x_train = x[:30]
y_train = y[:30]

# 테스트 데이터 30개
x_test = x[30:]
y_test = y[30:]
```

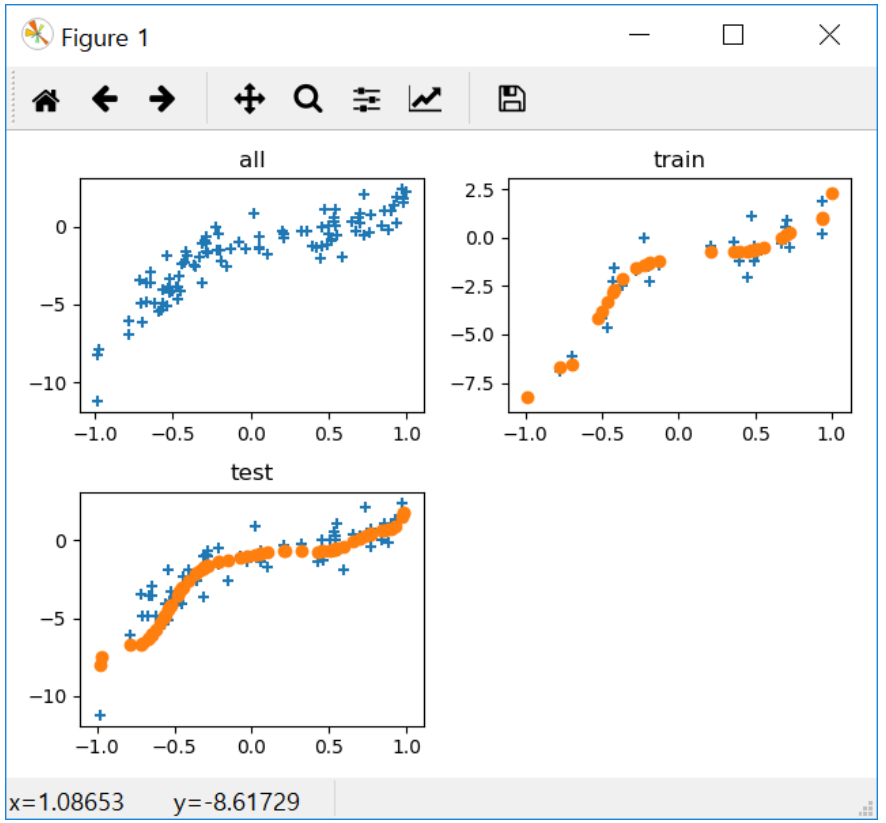
```
### 최소제곱법으로 9차식으로 회귀를 취해 본다

from sklearn import linear_model

# 학습용 입력 데이터
X_TRAIN = np.c_[x_train**9, x_train**8, x_train**7, x_train**6, x_train**5,
                x_train**4, x_train**3, x_train**2, x_train]

model = linear_model.LinearRegression()
model.fit(X_TRAIN, y_train)


### 계수, 절편, 학습 데이터에 의한 결정계수를 표시

print('계수 ( 학습 데이터 ) ', model.coef_)
print('절편 ( 학습 데이터 ) ', model.intercept_)

print('결정계수 ( 학습 데이터 ) ', model.score(X_TRAIN, y_train))
```

테스트 데이터에 의한 결정계수를 표시

```
X_TEST = np.c_[x_test**9, x_test**8, x_test**7, x_test**6, x_test**5,
               x_test**4, x_test**3, x_test**2, x_test]

print('결정계수 ( 테스트 데이터 ) ', model.score(X_TEST, y_test))
```

그래프 표시

```
plt.subplot(2, 2, 1)
plt.scatter(x, y, marker='+')
plt.title('all')
plt.subplot(2, 2, 2)
plt.scatter(x_train, y_train, marker='+')
plt.scatter(x_train, model.predict(X_TRAIN), marker='o')
plt.title('train')
plt.subplot(2, 2, 3)
plt.scatter(x_test, y_test, marker='+')
plt.scatter(x_test, model.predict(X_TEST), marker='o')
plt.title('test')
plt.tight_layout()
plt.show()
```

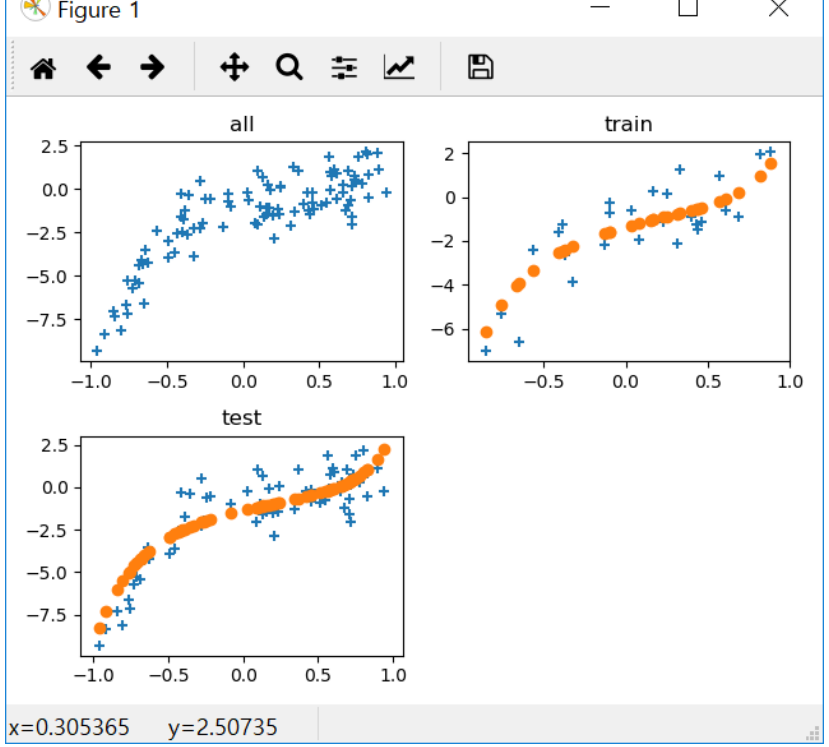



```
C:\JIN\Anaconda3\envs\tf\python.exe C:/Users/jin/PycharmProjects/alg/4.3-overfitting.py
계수 (학습 데이터)  [[  91.56352094  -45.68156187 -168.41277229   82.67175583   88.5618968
   -40.19724555   -7.54663911    0.62948693    1.78069398]]
절편 (학습 데이터)  [-0.98453441]
결정계수 (학습 데이터)  0.9145276117333918
결정계수 (테스트 데이터)  0.7850355599093786
```

| 머신 러닝에서 피해야 하는 부분 : 과적합

1. 그래프를 보면 알 수 있듯이 제대로 예측하지 못했다.
2. R^2 결정 계수도 학습 데이터와 비교해 수치가 나쁘다. 바로 과적합 상태이다.
3. 일반적으로 과적합이 발생하는 것은 학습 데이터에 비해 모델이 너무 복잡하기 때문이다.
    1. 원래 4차식인 부분을 9차식으로 회귀해서 나온 결과이다.

4. 과적합 대응 : 정규화 회귀 분석
    1. 최소제곱법은 예측 값과 실제 값 오차의 제곱합을 최소화 하는 방법
    2. 이 최소화 하는 대상을 오차만이 아니라 모델 복잡도까지 더한 것이 정규화 회귀분석
    3. 이 방법보다는 리지 회귀 기법이 더 많이 알려져 있다.
        1. 리지모델은 sklearn.linear_model.Ridge에 구현되어 있다.




```
import matplotlib.pyplot as plt
import numpy as np

###  y = 4x^3 - 3x^2 + 2x - 1 데이터 생성

x = np.random.rand(100, 1)  # 0~1까지 난수를 100개 만든다
x = x * 2 - 1               # 값의 범위를 -2~2로 변경

y = 4 * x**3 - 3 * x**2 + 2 * x - 1

y += np.random.randn(100, 1)  # 표준정규분포 ( 평균 0,표준편차 1 ) 의 난수를 더한다
```

```
# 학습 데이터 30개
x_train = x[:30]
y_train = y[:30]
# 테스트 데이터 30개
x_test = x[30:]
y_test = y[30:]

### Ridge 로 9차식으로서 회귀를 취해 본다
from sklearn import linear_model
# 학습용의 입력 데이터
X_TRAIN = np.c_[x_train**9, x_train**8, x_train**7, x_train**6, x_train**5,
                x_train**4, x_train**3, x_train**2, x_train]

model = linear_model.Ridge()
model.fit(X_TRAIN, y_train)
```

```
### 계수、절편、학습 데이터에 의한 결정계수를 표시

print('계수 ( 학습 데이터 ) ', model.coef_)
print('절편 ( 학습 데이터 ) ', model.intercept_)

print('결정계수 ( 학습 데이터 ) ', model.score(X_TRAIN, y_train))


### 테스트 데이터에 의한 결정계수를 표시

X_TEST = np.c_[x_test**9, x_test**8, x_test**7, x_test**6, x_test**5,
               x_test**4, x_test**3, x_test**2, x_test]

print('결정계수 ( 테스트 데이터 ) ', model.score(X_TEST, y_test))
```

그래프 표시

```
plt.subplot(2, 2, 1)
plt.scatter(x, y, marker='+')
plt.title('all')

plt.subplot(2, 2, 2)
plt.scatter(x_train, y_train, marker='+')
plt.scatter(x_train, model.predict(X_TRAIN), marker='o')
plt.title('train')

plt.subplot(2, 2, 3)
plt.scatter(x_test, y_test, marker='+')
plt.scatter(x_test, model.predict(X_TEST), marker='o')
plt.title('test')

plt.tight_layout()
plt.show()
```

```
C:\JIN\Anaconda3\envs\tf\python.exe C:/Users/jin/PycharmProjects/alg/4.4-ridge.py
계수 (학습 데이터)  [[ 0.57034    -0.009902    0.8137433  -0.13522762  1.16702897 -0.44443425
   1.63182965 -1.13943559  2.08881929]]
절편 (학습 데이터)  [-1.36466513]
결정계수 (학습 데이터)  0.760026595732703
결정계수 (테스트 데이터)  0.7946771219875042
```