Logistic Regression

2021.07.21 AAI Lab. 세미나

Logistic Regression

Logistic Regression :

- 독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는데 사용되는 통계 기법

- 회귀를 사용하여 데이터가 어떤 범주에 속할 확률을 0에서 1사이의 값으로 예측하고 분류해주는 지도학습 알고리즘

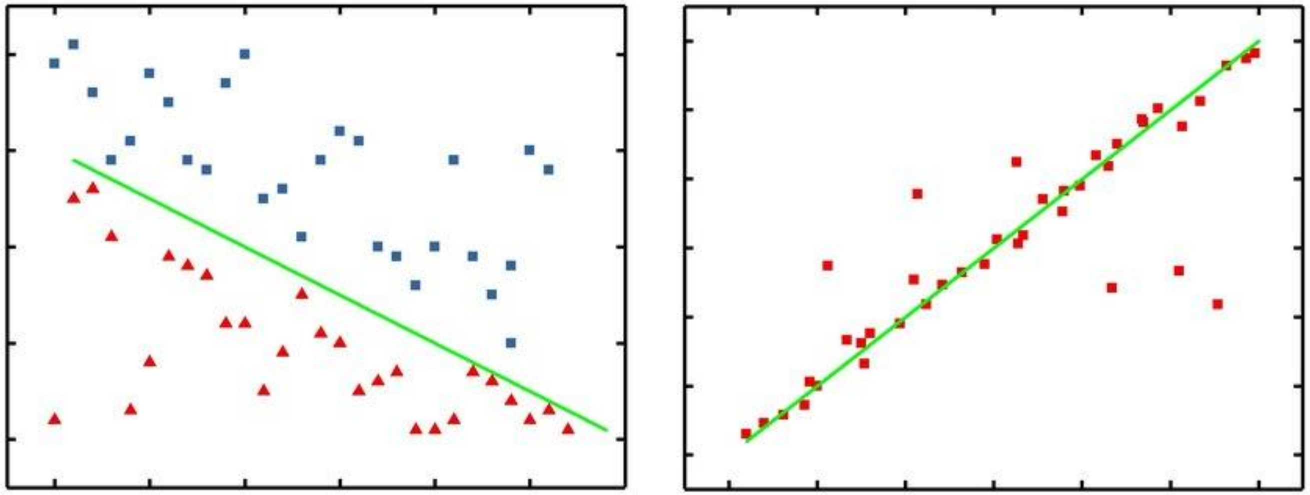
Logistic Regression

- Linear Regression과의 차이:

  Linear Regression은 연속된 값을 예측하는 반면,

  Logistic Regression은 discrete한 값을 예측한다.

  → classification



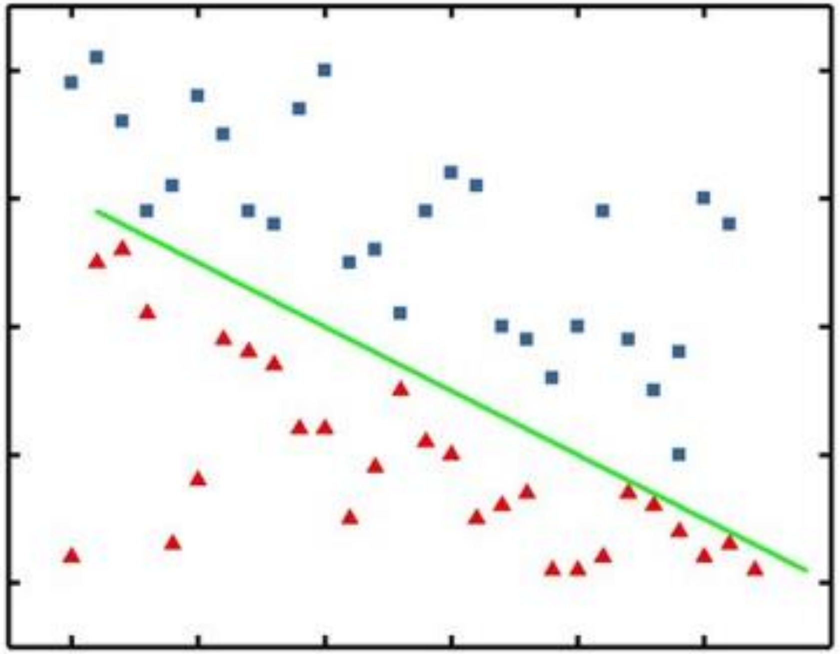
(a) Logistic Regression (b) Linear Regression

Logistic Regression - Classification

- Training Data의 특성과 상관관계 등을 파악한 후, 입력 데이터가 어떤 종류의 결과로 분류될지를 예측

- Ex) 스팸 문자, 암 판별

Logistic Regression - Classification



(a) Logistic Regression

1. Training Data 의 특성과 분포를 나타내는 최적의 직선을 찾는다.
2. 직선을 기준으로 분류 (위 = 1, 아래 = 0 등)
3. 미지의 데이터(Test Data)를 분류

Logistic Regression은 정확도가 높은 Classification 알고리즘

Sigmoid function

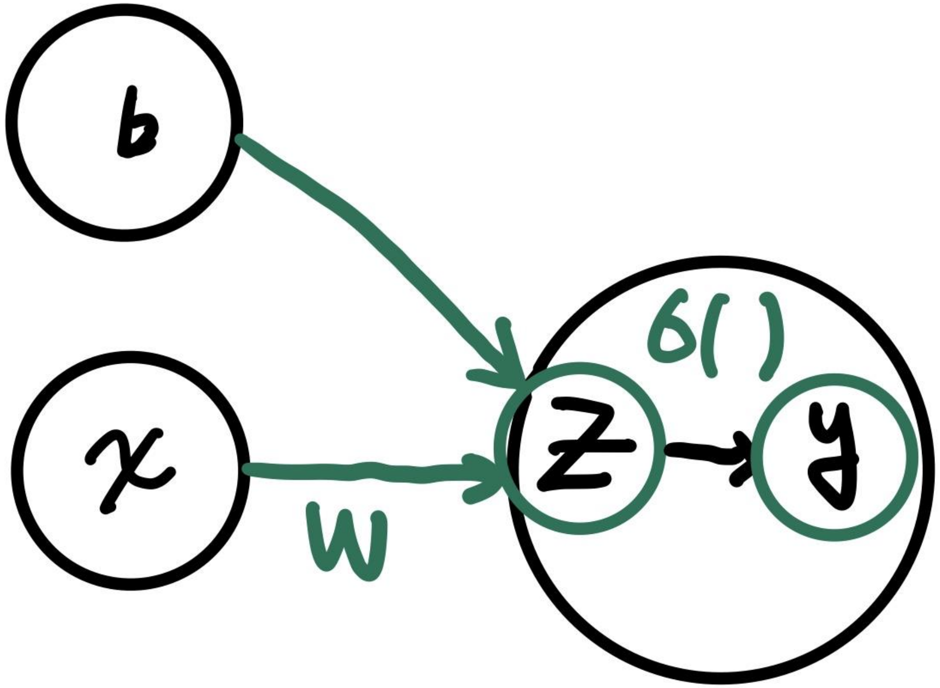
- 출력 값이 0과 1을 가져야 하는 즉, 분류 시스템에서 0~1사이의 값을 갖는 sigmoid함수를 이용함.

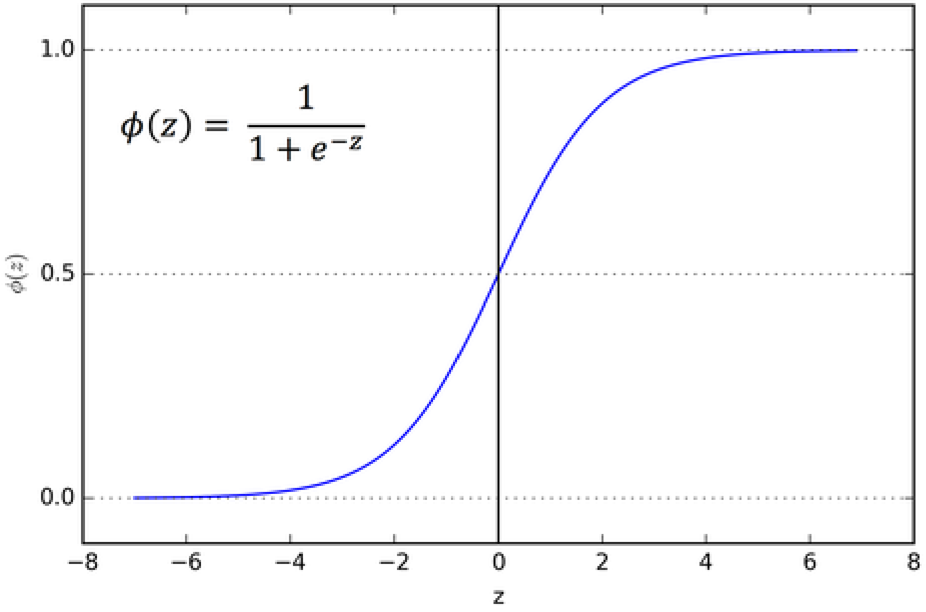
Cf) $\mathrm{y} > 0.5 \rightarrow \mathrm{y} = 1$, $y \leq 0.5 \rightarrow \mathrm{y} = 0$

Sigmoid(z)가 결과가 나타날 확률을 의미한다고 했는데 잘 이해가 안됨.

Sigmoid function

- $z = Wx + b$
- $y = sigmoid(z) = \sigma(z) = \frac{1}{1+e^{-z}}$





Sigmoid function, why?

확률 p가 주어졌을 때

$Odds(p) = \frac{p}{1-p},\ 0 \le p \le 1\ \rightarrow 0 \le Odds(p) \le \infty$

$\log(Odds(p)) = \log\left(\frac{p}{1-p}\right) \rightarrow -\infty \le \log(Odds(p)) \le \infty$

Sigmoid function, why?

$$\log Odds(p) = \log\left(\frac{p}{1-p}\right) = Wx + b \;\rightarrow \frac{p}{1-p} = e^{Wx+b}$$

$$\therefore p = \frac{1}{1+e^{-(Wx+b)}}$$

Cross-entropy loss function

- $E(W, b) = -\Sigma_{i=1}^{n}\{t_i \log y_i + (1 - t_i) \log(1 - y_i)\}$

- 유도

$p(C = 1|x) = y, \ \ p(C = 0|x) = 1 - y$

$p(C = t|x) = y^t(1 - y)^{1-t} \, , t \in \{1,2\}$

$L(W, b) = \Pi_{i=1}^{n} p(C = t_i|x_i) = \Pi_{\mathrm{i}=1}^{n} {y_i}^{t_i}(1 - y_i)^{1-t_i}$

$E(W, b) = -\log L(W, b)$

```
In [7]:
import numpy as np
x_data = np.array([2, 4, 6, 8, 10, 12, 14, 16, 18, 20]).reshape(10,1)
t_data = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1]).reshape(10,1)

W = np.random.rand(1, 1)
b = np.random.rand(1)
print(f'W = {W}, W.shape = {W.shape}, b = {b}, b.shape = {b.shape}')
```

```
W = [[0.98789444]], W.shape = (1, 1), b = [0.83843783], b.shape = (1,)
```

```
In [10]:
def sigmoid(x):
    return 1/(1+np.exp(-x))

def loss_func(x, t):
    delta = 1e-7
    z = np.dot(x, W)+b
    y = sigmoid(z)
    return -np.sum(t*np.log(y + delta)+(1-t)*np.log((1-y)+delta))
```

```python
def numerical_grad(f, x):
    h = 1e-4
    grad = np.zeros_like(x)
    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = tmp_val + h
        fxh1 = f(x)

        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2)/(2*h)
        x[idx] = tmp_val
    return grad

def error_val(x, t):
    delta = 1e-7
    z = np.dot(x,W) + b
    y = sigmoid(z)

    return -np.sum(t*np.log(y + delta)+(1-t)*np.log((1-y)+delta))

def predict(x):
    z = np.dot(x,W) + b
    y = sigmoid(z)

    if y > 0.5:
        result = 1
    else:
        result = 0
    return y, result
```

In [11]:

```python
lr = 0.01
epoch = 1000000
f = lambda x: loss_func(x_data, t_data)
print(f'초기값|| 에러:{error_val(x_data, t_data)}, W:{W}, b:{b} ')
for step in range(epoch):
    W -= lr*numerical_grad(f, W)
    b -= lr*numerical_grad(f, b)

    if step%1000 == 0:
        print(f'step :{step}, error value:{error_val}, W:{W}, b:{b}')
```

In [20]:

```python
real_val, logical_val = predict(12)
print(real_val, logical_val)
```

```
[[0.0092251]] 0
```