Deep Learning with Python

Chapter 2 딥러닝 기초 02

Wedatalab 김정수

■ 목차

■ 학습목표

1. Binary Classification의 개념과 분류 문제를 이해한다.

2. Logistic function의 기능과 Logistic Regression의 과정을 이해한다.

3. Tensorflow를 이용하여 Logistic Regression을 실습한다.

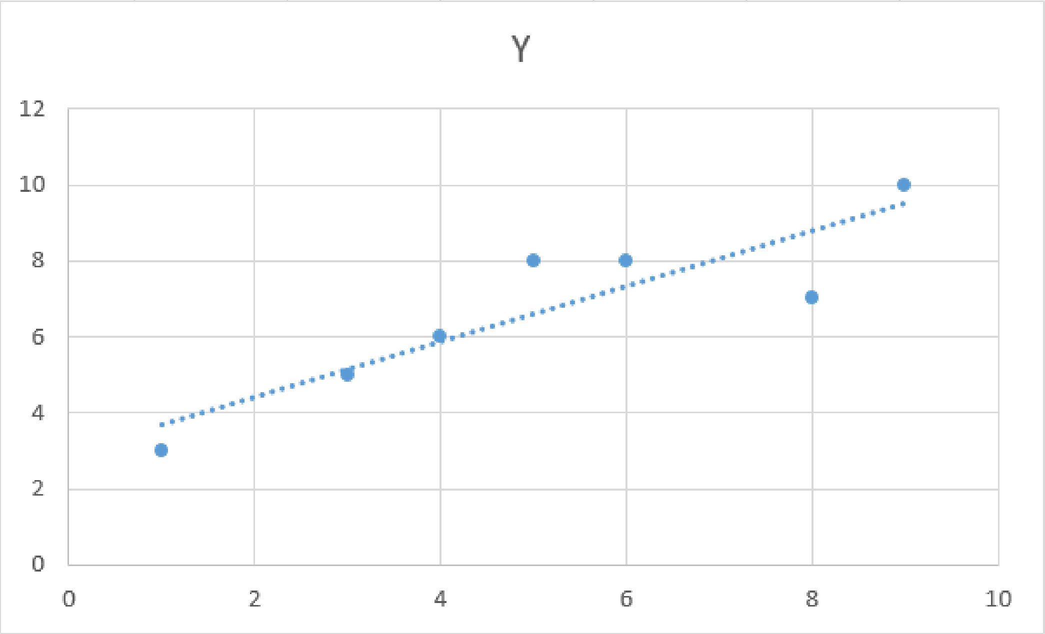
1. Logistic Regression

1. Logistic Regression

Regression이란?

- 새로운 데이터가 들어왔을 때 데이터에 해당하는 값을 예측(Prediction) 하는 문제로 Linear Regression을 사용할 수 있었다.

X	Y
1	3
3	5
4	6
5	8
6	8
8	7
9	10

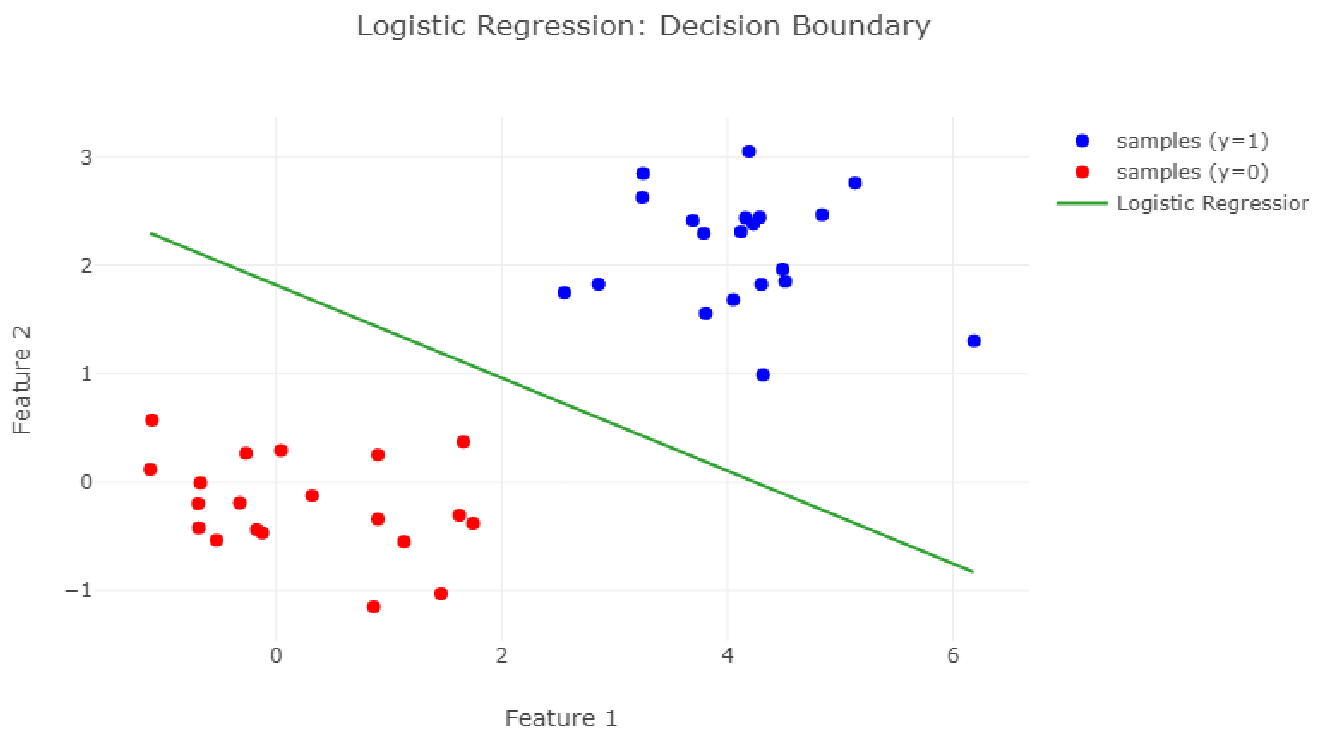


예) X: 시간(s), Y : 평균 속도(m/s)

1. Logistic Regression

Classification이란?

- 새로운 데이터가 들어왔을 때 기존 데이터의 그룹 중 어떤 그룹에 속하는지를 분류(Classification)하는 문제이다.



1. Logistic Regression

Classification 문제 예시 (1/4)

- 이메일 검출 : 스팸 메일 / 정상 메일

1. Logistic Regression

Classification 문제 예시 (2/4)

- 상품 판매 : 구매 / 비구매

1. Logistic Regression

Classification 문제 예시 (3/4)

- 게임(경기) : 승 / 패

1. Logistic Regression

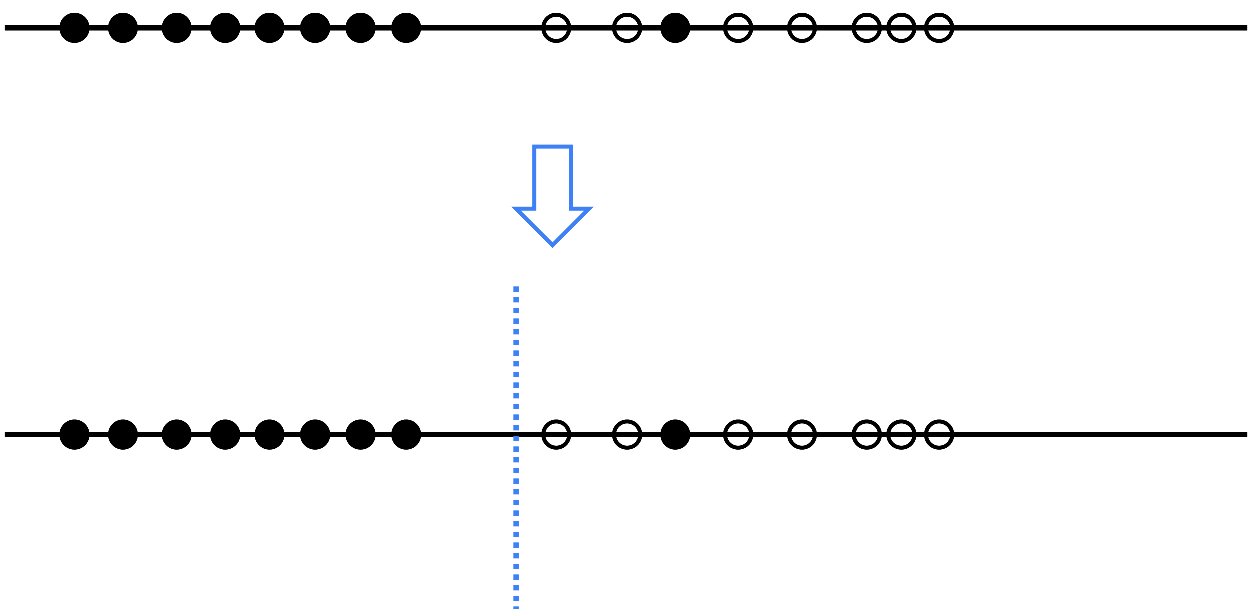
Classification 문제 예시 (4/4)

- 사기 탐지 기술(Fraud Detection System) : 진짜 / 사기

1. Logistic Regression

선형 판별 분석법 (1/3)

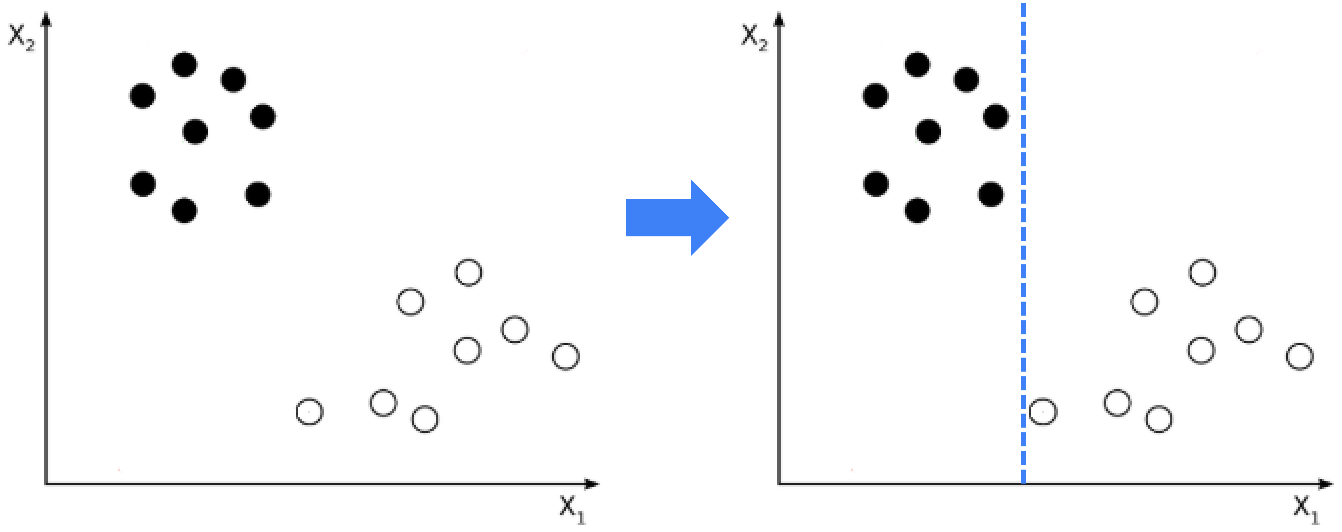
- 선형 분류 문제는 데이터를 분류하는 선형 결정 경계 (Decision boundary)를 찾는 것이다.
- 1차원의 경우 점(0차원)이 선형 결정 경계이다.



1. Logistic Regression

선형 판별 분석법 (2/3)

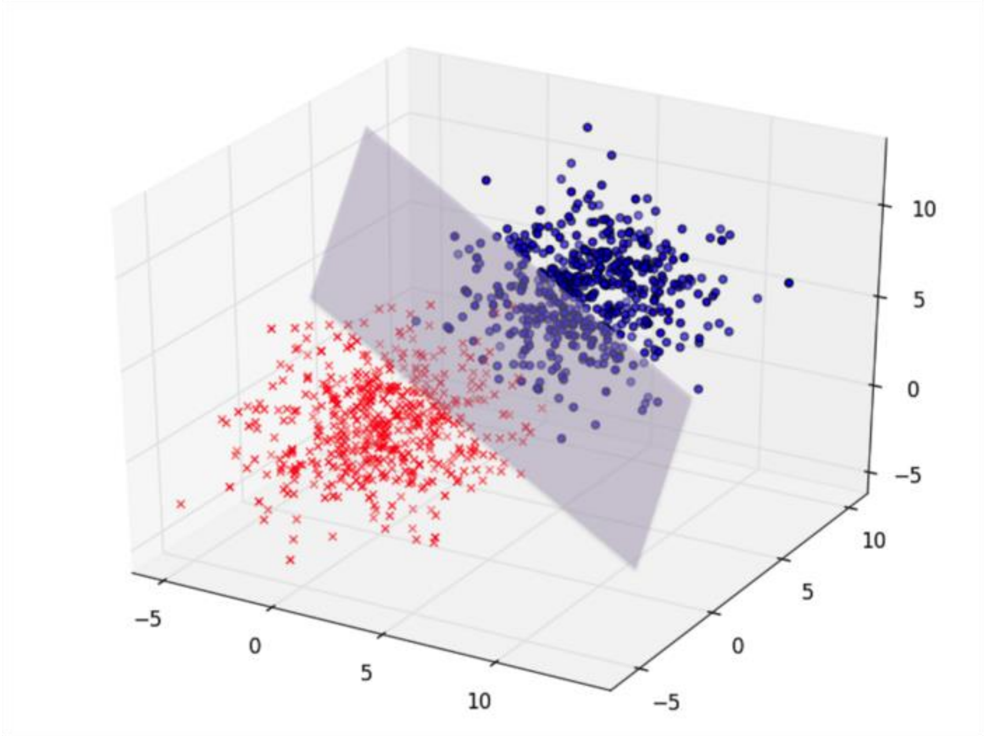
- 2차원의 경우 직선(1차원)이 선형 결정 경계이다.



1. Logistic Regression

선형 판별 분석법 (3/3)

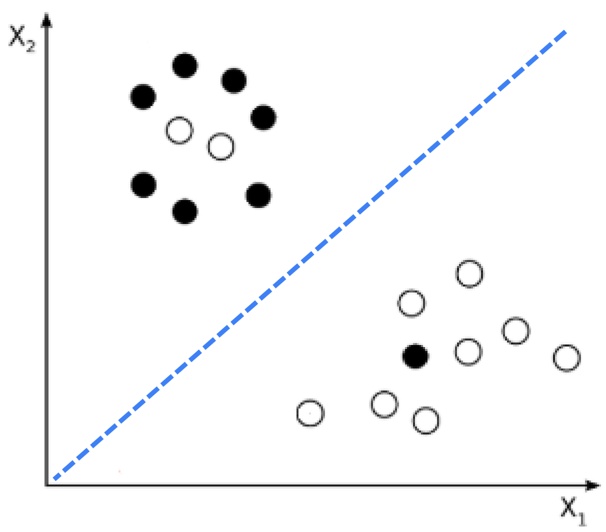
- 3차원의 경우 평면(2차원)이 선형 결정 경계이다.
- N차원의 경우 (N-1)차원의 초평면(hyperplane)이다.

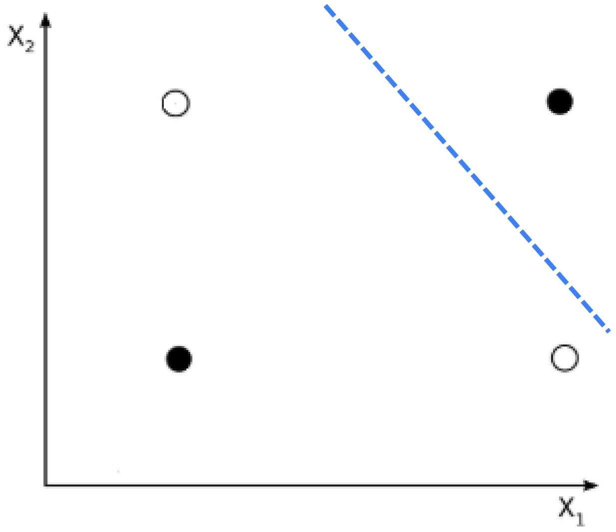


1. Logistic Regression

선형 판별 분석법의 문제 (1/2)

- 데이터의 양상에 따라 데이터가 선형으로 분류되지 않을 수도 있다.
- 이 경우 아무리 직선(또는 직선의 기울기)을 바꾸더라도 오분류되는 경우가 존재한다.

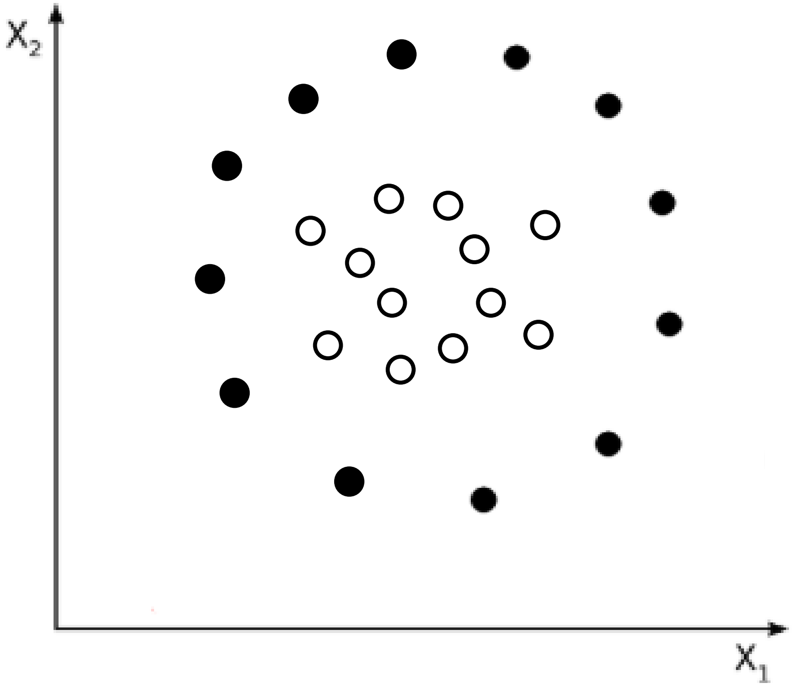




1. Logistic Regression

선형 판별 분석법의 문제 (2/2)

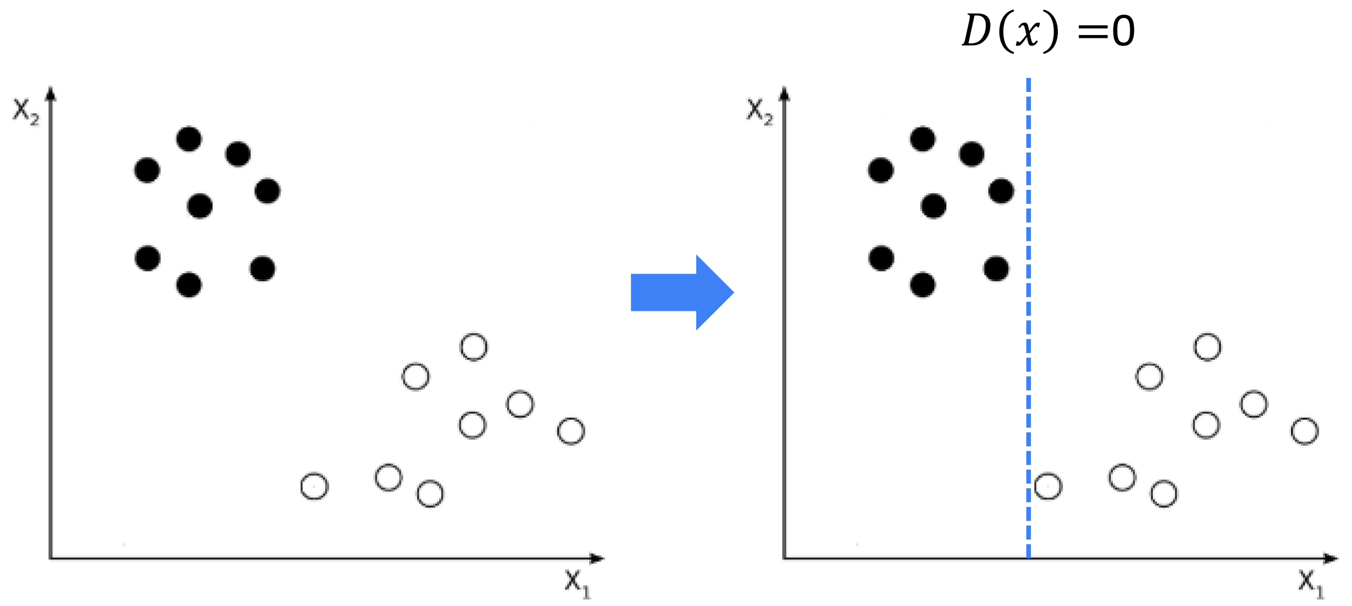
- 비선형 결정 경계를 쓰는 것이 바람직할 수도 있다.



1. Logistic Regression

선형 결정 경계 (Decision boundary)

- 2차원에서 선형으로 분리 가능한 경우(linearly separable)를 가정하면 결정 경계는 $D(x) = W_1x_1 + W_2 x_2 + b = 0$



1. Logistic Regression

Logistic Regression이란? (1/2)

- D.R.Cox가 1958년 제안한 확률모델이다.

- Logistic Regression은 특징(feature)들의 선형 결합으로 확률 (0~1사이의 값)을 예측하는 기법이다.

- Linear Regression과 연결함수를 합친 형태이다.

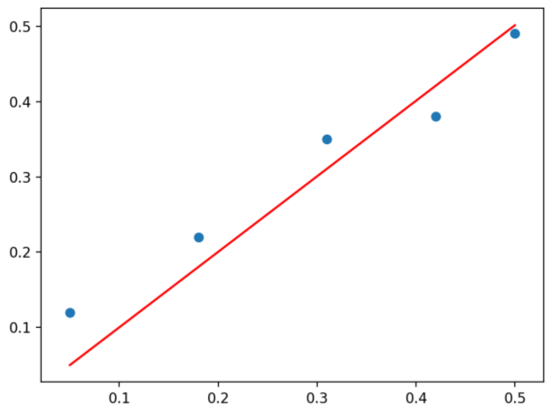
- 예측한 확률 값으로 y의 범주(0 또는 1)를 예측하므로 분류 (Classification) 문제에 활용할 수 있다.

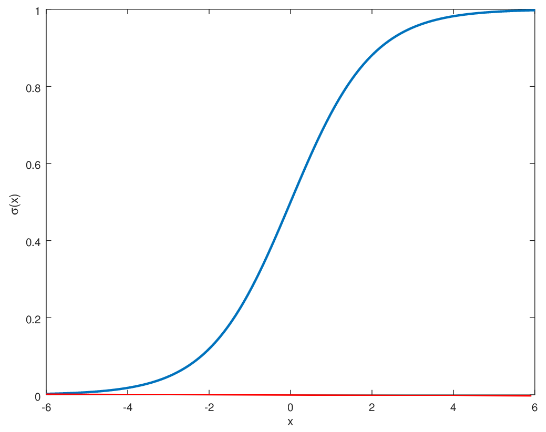
- 분류하려는 범주가 성공/실패, 예/아니오, 남/여 등 2가지 수준으로 나눠진 경우에 사용한다.

1. Logistic Regression

Logistic Regression이란? (2/2)

- Linear Regression으로 결정 경계를 구한다
- 연결 함수를 사용하여 Linear Regression의 결과를 0~1사이의 확률 값으로 변환

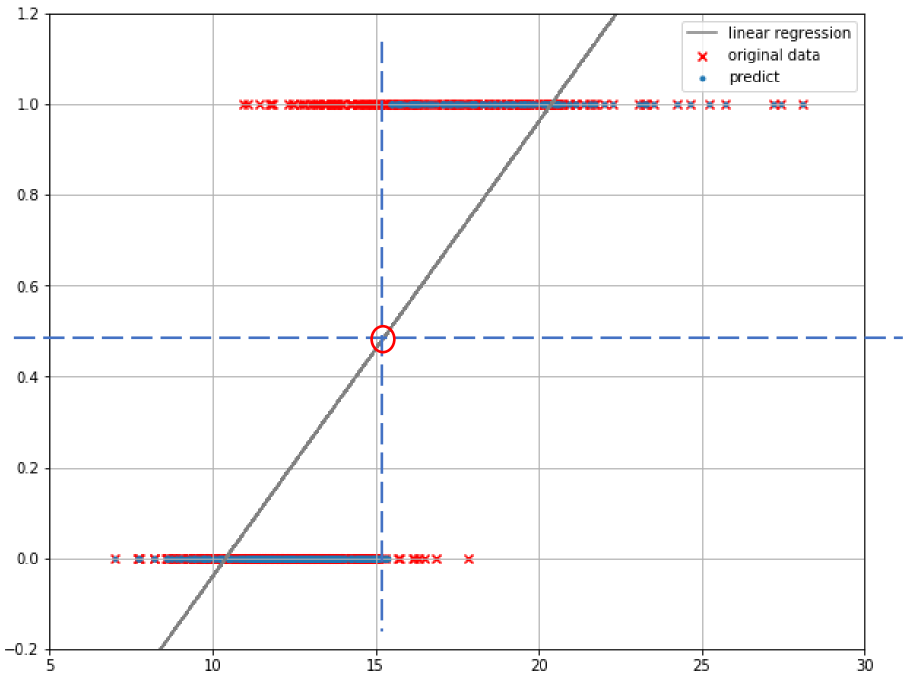




1. Logistic Regression

Linear Regression만 사용하는 경우 (1/2)

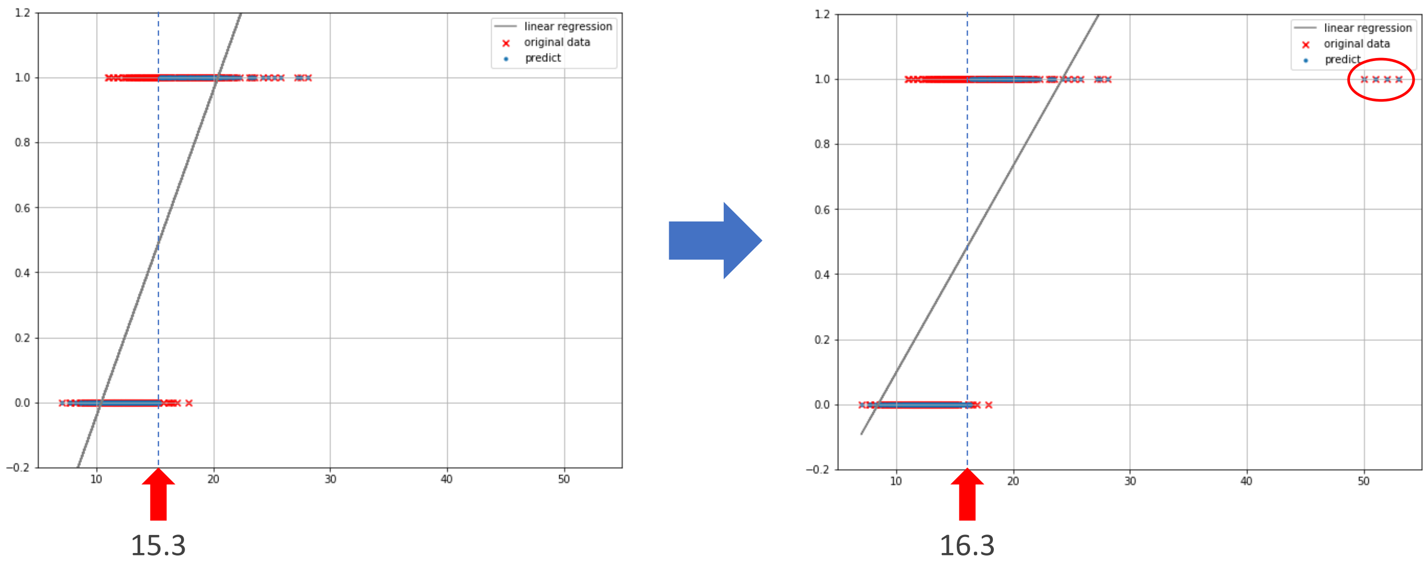
- Linear Regression으로 구한 예측값을 확률로 사용한다.
- 적당한 분계점(Cutoff)을 설정하여 0 또는 1로 분류한다.
  ex) 예측값이 0.5보다 크면 1, 작으면 0으로 분류



1. Logistic Regression

Linear Regression만 사용하는 경우 (2/2)

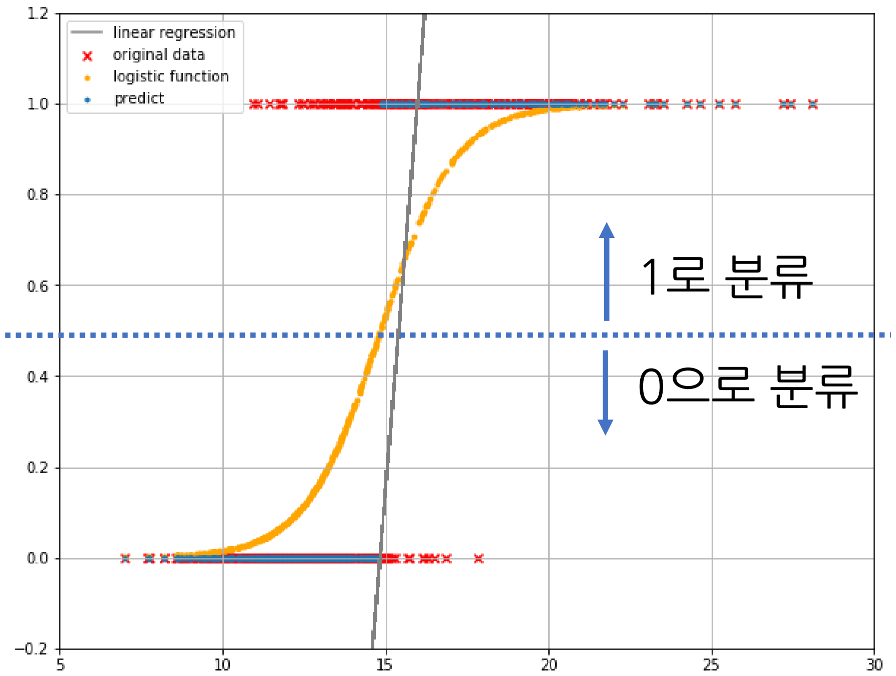
- [0, 1]을 넘어가는 값이 존재한다.
- 데이터 분포가 변하면 결정 경계와 분계점이 바뀌는 문제가 있다.



1. Logistic Regression

연결 함수로 Logistic function 사용 (1/2)

- Logistic function을 연결 함수로 사용하여 Linear regression의 문제점을 해결할 수 있다.

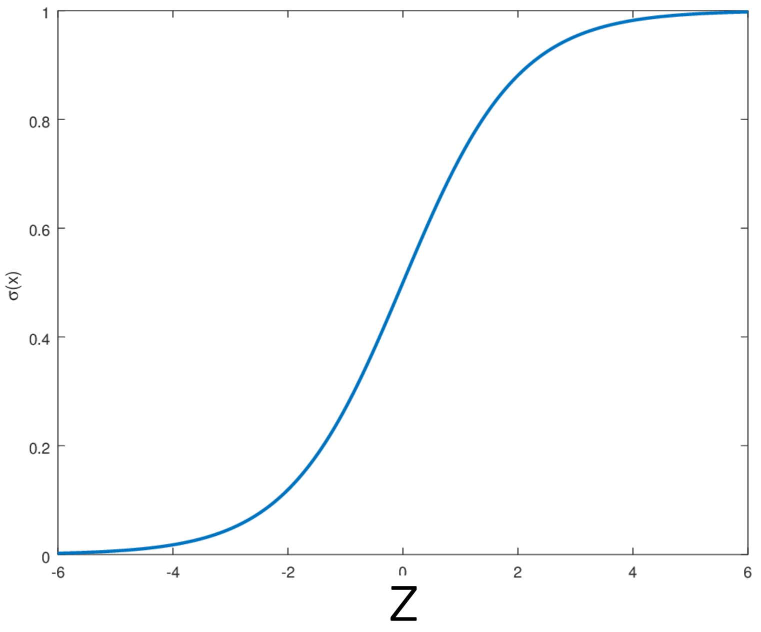


1. Logistic Regression

연결 함수로 Logistic function 사용 (2/2)

- Logistic function을 sigmoid function이라 하며 수식으로는 $g(z) = \frac{1}{1+e^{-z}} = \frac{e^{z}}{1+e^{z}}$ 로 나타낸다.
- Logistic function의 입력값의 범위는 [- ∞, ∞], 출력값의 범위는 [0,1]이다.

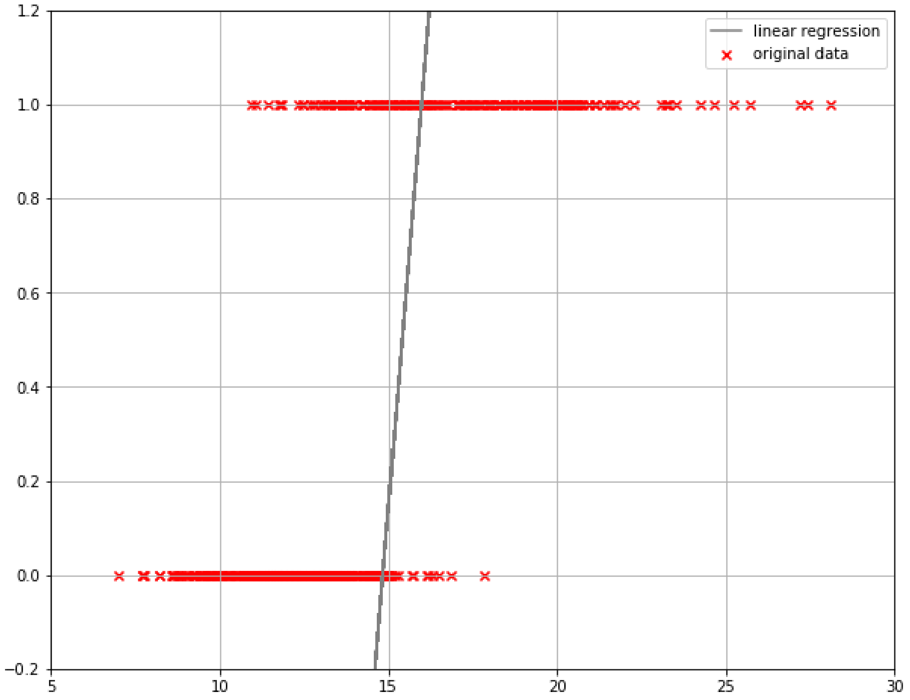
g(z)



1. Logistic Regression

1차원 데이터에 대한 Logistic Regression 과정 (1/4)

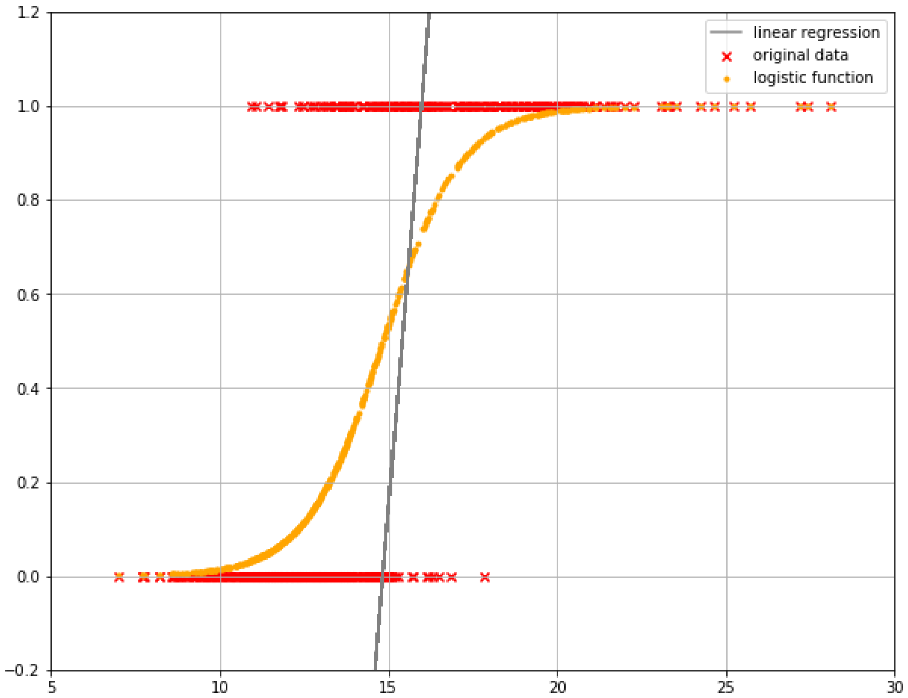
1. 데이터를 분류하는 선형 결정 경계 $z = D(x) = Wx + b$를 찾는다.



1. Logistic Regression

1차원 데이터에 대한 Logistic Regression 과정 (2/4)

1. 데이터를 분류하는 선형 결정 경계 $z = D(x) = Wx + b$를 찾는다.
2. 각각의 점에 확률값 $p = g(z) = \frac{e^z}{1+e^z} = \frac{e^{Wx+b}}{1+e^{Wx+b}}$를 배정한다.

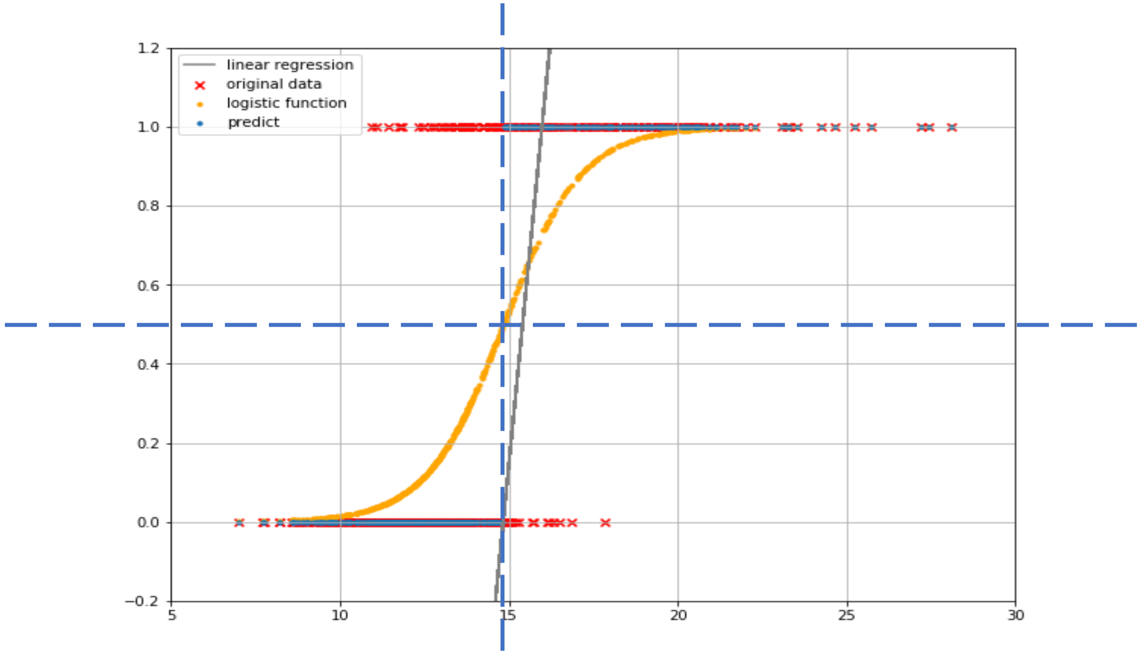


1. Logistic Regression

1차원 데이터에 대한 Logistic Regression 과정 (4/4)

1. 데이터를 분류하는 선형 경계 $z = D(x) = Wx + b$를 찾는다.
2. 각각의 점에 확률값 $p = g(z) = \frac{e^z}{1+e^z} = \frac{e^{Wx+b}}{1+e^{Wx+b}}$를 배정한다.
3. 분계점에 따라 분류한다.

   예) 확률이 0.5보다 크면 1, 0.5보다 작으면 0으로 분류



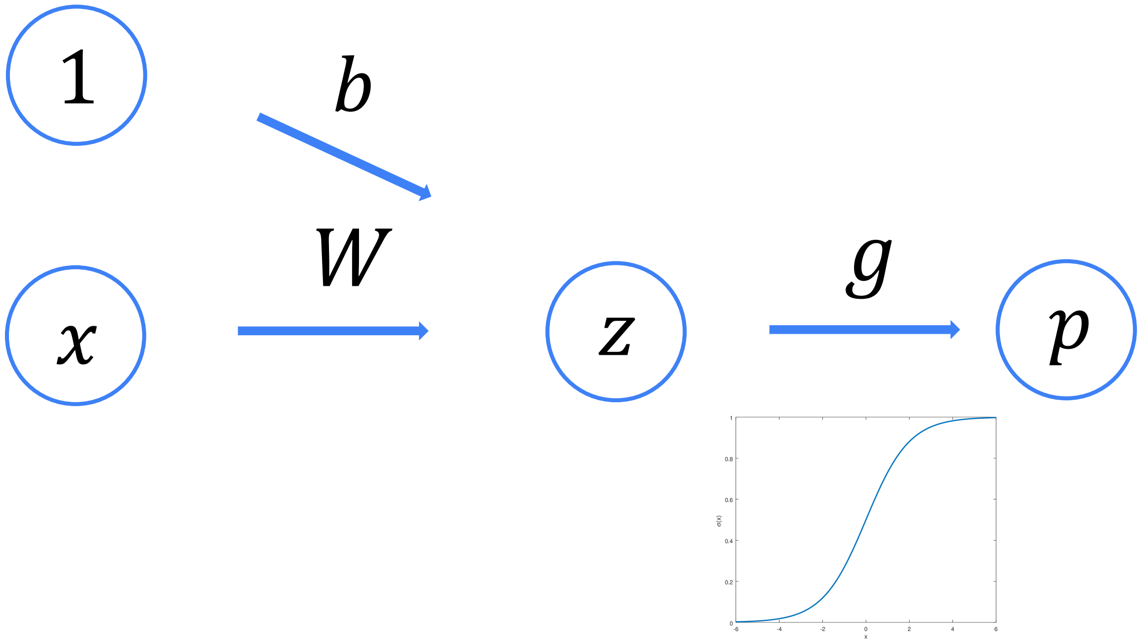
1. Logistic Regression

1차원 데이터에 대한 Logistic Regression 과정 (3/4)

- Logistic Regression의 과정을 퍼셉트론으로 표현하면 다음과 같다.

$$z = D(x) = Wx + b$$

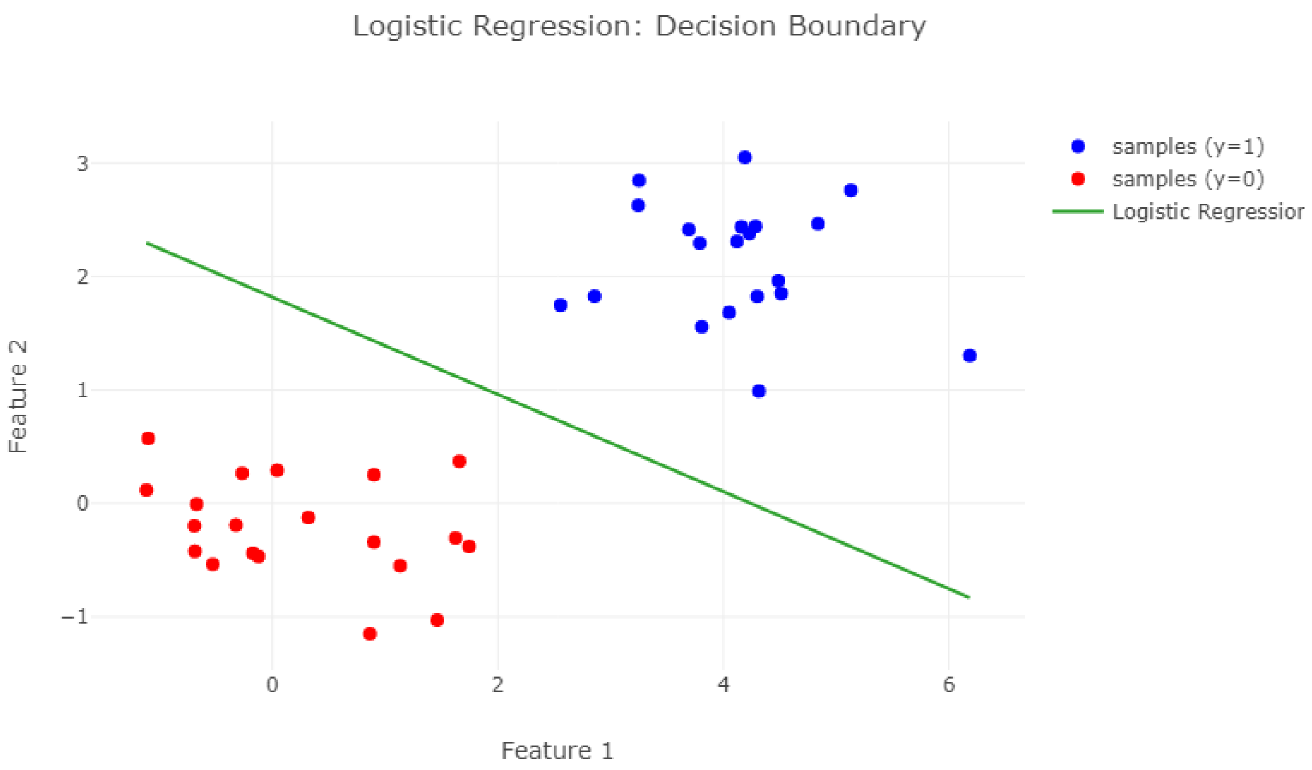
$$p = g(z) = \frac{e^z}{1+e^z} = \frac{e^{Wx+b}}{1+e^{Wx+b}}$$



1. Logistic Regression

2차원 데이터에 대한 Logistic Regression

- 1차원 데이터와 같은 방법으로 분류할 수 있다.



1. Logistic Regression

Logistic Regression 설계 요소

1. 모델(출력 계산 방법)
 : 주어진 데이터를 분류하는 선형 결정 경계로 퍼셉트론 모형을 사용하여 가중치(Weight)와 편향(bias)을 학습한다.
 - 가중치(Weight)는 뉴런(X)이 결과(Y)에 주는 영향력(중요도)을 조절하는 매개 변수이다.
 - 편향(bias)은 뉴런이 얼마나 쉽게 활성화하는지를 조정한다.
 - Sigmoid function을 통해 선형 결정 경계를 확률로 나타낸다.

2. 비용 함수(Cost Function)
 : Cross-entropy function를 통해 비용 함수를 설정한다.

3. 최적화(Optimization)
 : Cost(오차)를 최소화하기 위해 경사하강법(Gradient Descent) 알고리즘을 사용한다.

1. Logistic Regression

모델(출력 계산 방법) 정의하기

- 주어진 데이터를 분류하는 결정 경계를 구하기 위해
  가중치(Weight)와 편향(bias)을 사용하여 선형 결정 경계를 정의한다.

- Sigmoid function을 사용하여 선형 결정 경계를 확률로 변환한다.

- $z = D(x) = Wx + b$

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

$$x \xrightarrow{D} z \xrightarrow{g} p$$

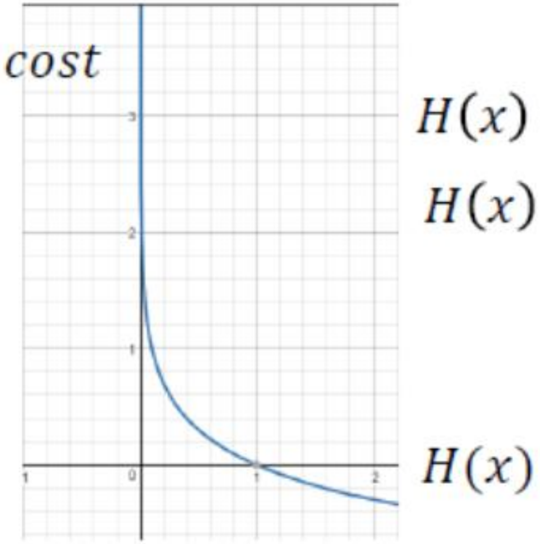
- z는 선형 결정 경계, p는 Sigmoid function으로 구한 확률값을 의미

1. Logistic Regression

비용 함수(Cost Function) 정의 (1/4)

- 0~1사이의 확률값인 예측값(p)과 실제값(y)을 비교하려면 y값에도 범위 제한이 필요하다.
- 따라서 Logistic Regression은 비용 함수를 y = 1인 경우와 y = 0인 경우 두 가지로 나눠서 함수를 정의한다.
- 먼저 y = 1일 때는 아래와 같다.

$$-\log(H(x))$$



$$H(x) = 0, cost = \infty$$

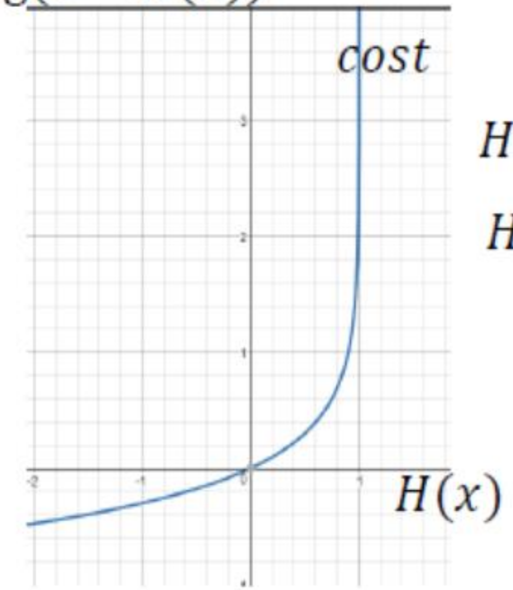
$$H(x) = 1, cost = 0$$

1. Logistic Regression

비용 함수(Cost Function) 정의 (2/4)

- y = 0일 때는 아래와 같다.
- 예측값(p)의 범위 0과 1사이에서 실제값(y)과 예측값(p)이 같을 때 비용이 최소가 되게 하고, 다를 때는 비용이 최대가 되도록 설계한 것이다.

$$-\log(1-H(x))$$



$$H(x) = 1, cost = \infty$$

$$H(x) = 0, cost = 0$$

1. Logistic Regression

비용 함수(Cost Function) 정의 (3/4)

- 비용 함수를 y = 1, y = 0 일 때로 나눠서 정의하면 다음과 같다.

$$\mathrm{c}(\mathrm{H}(\mathrm{x}), \mathrm{y}) = \begin{cases} -\log\big(H(x)\big) & : \; y = 1 \\ -\log\big(1 - H(x)\big) & : y = 0 \end{cases}$$

- 두 부분을 합치면 아래와 같이 표현할 수 있다.

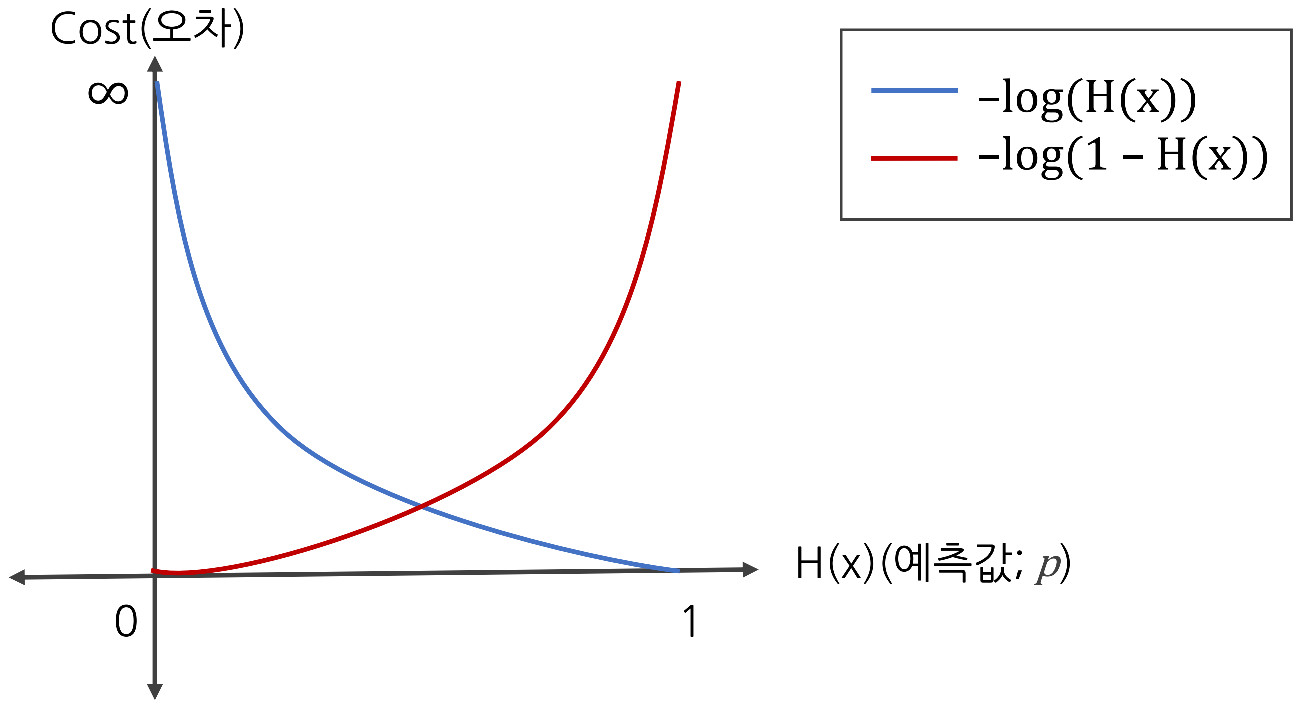
$$\mathrm{c}(\mathrm{H}(\mathrm{x}), \mathrm{y}) = -\,\mathrm{y}\log(\mathrm{H}(\mathrm{x})) - (1 - \mathrm{y})\log(1 - \mathrm{H}(\mathrm{x}))$$

- 이러한 형태의 함수를 Cross-entropy function이라 한다.

1. Logistic Regression

비용 함수(Cost Function) 정의 (4/4)

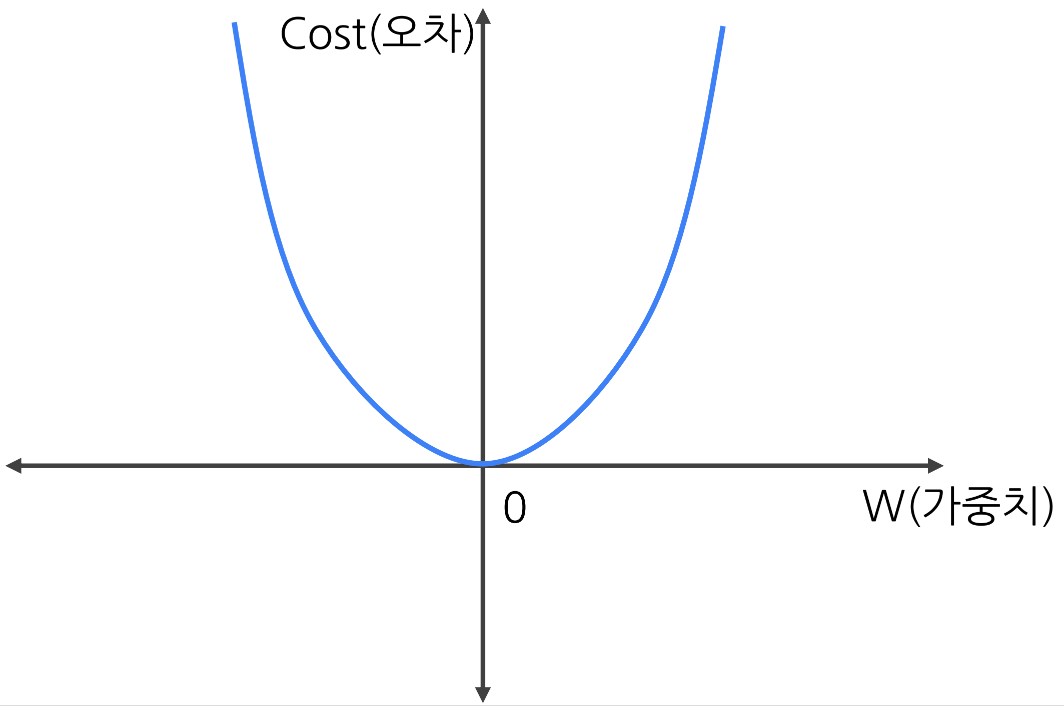
- Cross-entropy function의 그래프는 아래와 같다.
- 두 개의 로그 함수가 합쳐져서 아래로 볼록한 모양을 형성한다.



1. Logistic Regression

최적화(Optimization) 정의 (1/3)

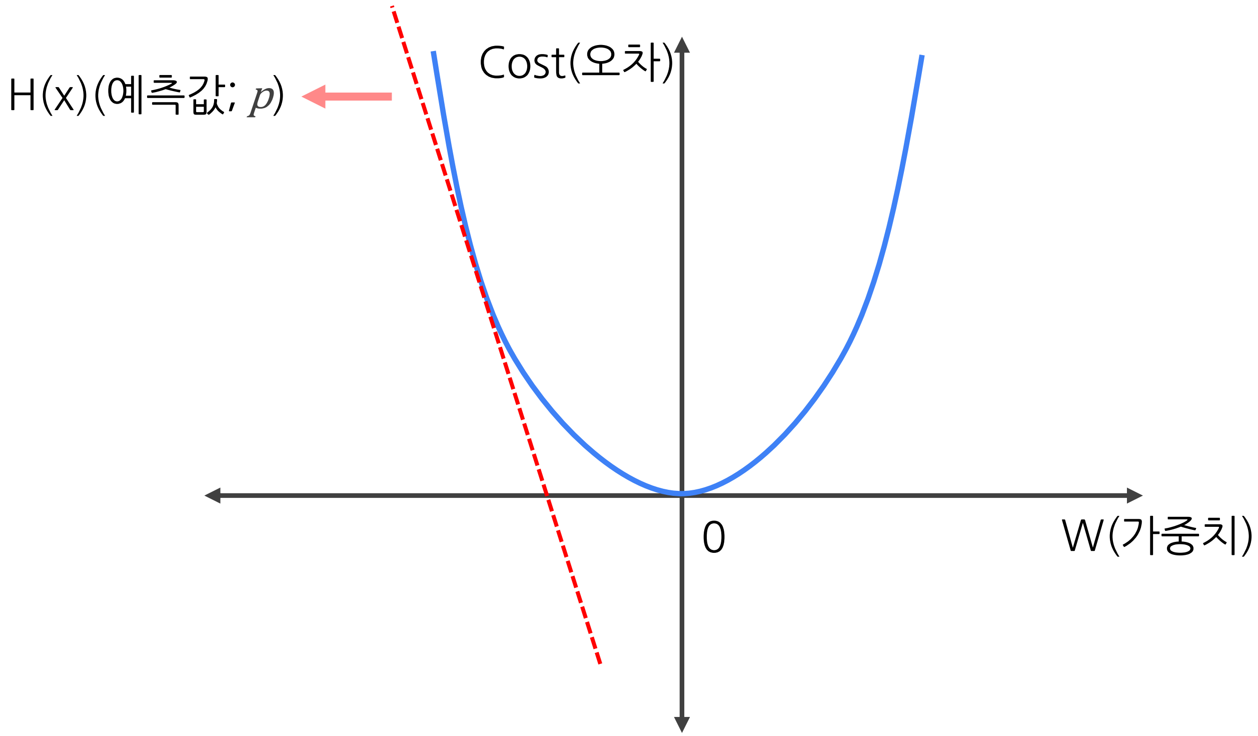
- W(가중치)에 따른 Cost(오차) 그래프를 나타내면 아래와 같다.
- 비용 함수로 Cross-entropy function을 사용했기 때문에 아래와 같은 그래프 형태가 가능하다.



1. Logistic Regression

최적화(Optimization) 설계하기 (2/3)

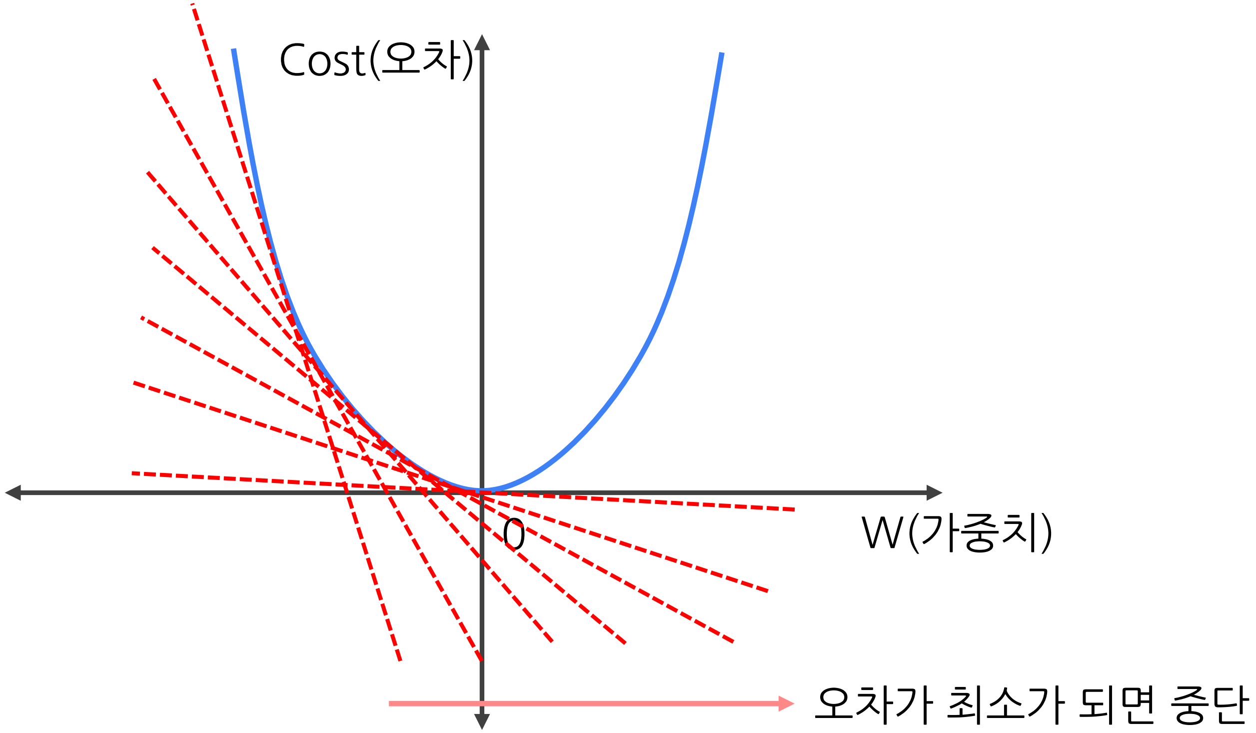
- Cost(오차)를 최소화하는 방법으로 경사하강법(Gradient Descent) 알고리즘을 사용한다.



1. Logistic Regression

최적화(Optimization) 설계하기 (3/3)

- 경사하강법(Gradient Descent) 알고리즘의 원리는 W(가중치)에 대한 미분값을 통해 W값을 업데이트하여 오차의 최저점을 찾는 것이다.
- $W := W - \alpha \frac{\partial}{\partial w} cost(W)$



1. Logistic Regression

Tensorflow를 이용한 Logistic Regression 설계하기

1. 그래프를 만든다. (Build graph)
2. session을 열고 sess.run()으로 그래프를 실행시킨다. (Run)
3. 반환되는 결과 값을 가지고 반복해서 변수를 업데이트하여 학습시킨다. (Update)

1. Logistic Regression

전체 코드 (1/3)

```
import tensorflow as tf
import numpy as np

learning_rate = 0.01
training_cnt = 10000
display_step = 1000

train_X = np.array([[2, 3], [4, 3], [4, 2], [2, 5], [6, 3], [5,
4] ])
train_Y = np.array([[0], [0], [0], [1], [1], [1]])

X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

pred = tf.sigmoid(tf.matmul(X ,W) + b)
```

1. Logistic Regression

전체 코드 (2/3)

```
cost = -tf.reduce_mean(
            Y * tf.log(pred) + (1 - Y) * tf.log(1 - pred))
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)

predicted = tf.cast(pred > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y),
               dtype=tf.float32))

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_cnt):
    r_cost, r_W, r_b, r_pred, r_c, r_a, _ = sess.run(
        [cost, W, b, pred, predicted, accuracy, op_train],
        feed_dict = {X: train_X, Y: train_Y})
```

1. Logistic Regression

전체 코드 (3/3)

```
        if (epoch+1) % display_step == 0:
        print("\nRun_count : [%04d], Train_cost =[%.4f], W
=[%.4f %.4f], b =[%.4f], \
               \npred =[%.4f %.4f %.4f %.4f %.4f %.4f], \
               \npred_Y = [%d %d %d %d %d %d ], \
               \ntrue_Y=  [%d %d %d %d %d %d ], \
               \naccuracy = [%.2f%%] "
              % (epoch+1, r_cost, r_W[0], r_W[1], r_b,
                 r_pred[0], r_pred[1], r_pred[2], r_pred[3],
r_pred[4], r_pred[5],
                 r_c[0], r_c[1], r_c[2], r_c[3], r_c[4], r_c[5],
                 train_Y[0], train_Y[1], train_Y[2], train_Y[3],
train_Y[4], train_Y[5],
                 r_a*100))

print("\nOptimization Finished!")
```

1. Logistic Regression

모델 구축(Build graph) (1/5)

파라메터 값 설정

- 머신러닝을 위한 기초 파라메터
- learning_rate : weight 값이 너무 적으면 Train 되지 않을 수 있고 weight값이 너무 크면 overshooting이 발생할 수 있다.
- training_cnt : data set에 대한 training 반복 횟수

```
# 파라메터값 설정
learning_rate = 0.01
training_cnt = 10000
display_step = 100  # 원하는 출력 빈도 조정
```

1. Logistic Regression

모델 구축(Build graph) (2/5)

트레이닝 데이터 변수 선언

- 학습 할 x data(input 2개), y data(output 1개) 설정
- numpy array를 사용

```
train_X = np.array([[2, 3], [4, 3], [4, 2], [2, 5],
[6, 3], [5, 4] ])
train_Y = np.array([[0], [0], [0], [1], [1], [1]])
```

tf graph input

- X : 들어오는 row는 정해진게 없고, column 은 2개 즉 입력변수 2개
- Y : 들어오는 row는 정해진게 없고, column 은 1개 즉 output 1개
- matrix 사용 하기 때문에 입력 변수를 담는 placeholder 1개로 된다.

```
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
```

1. Logistic Regression

모델 구축(Build graph) (3/5)

tf.random_normal

- bias, weight의 초기값을 난수로 생성

```
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
```

sigmoid 함수 사용

- 기존 pred 계산에 sigmoid 함수를 적용해 0~1사이의 값으로 변환
- sigmoid function => $H(X) = \frac{1}{1+e^{-Z}}$
- z = XW + b

```
pred = tf.sigmoid(tf.matmul(X ,W) + b)
```

1. Logistic Regression

모델 구축(Build graph) (4/5)

cost/loss function 구현

- 0~1사이의 값을 근사화 하기 위해서 log함수를 사용

$$C(H(x),y) = \frac{1}{m}\sum -(ylog(H(x)) - (1 - y)\log(1 - H(x))$$

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(pred) + (1 - Y) *
tf.log(1 - pred) )
```

학습 방법 ➔ cost를 최소화

- GradientDescent 함수 사용 (경사하강법)

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)
```

1. Logistic Regression

모델 구축(Build graph) (5/5)

학습된 예측값을 0과 1로 변환

- 0~1사이로 학습된 예측값을 0과 1로 나누어 분류할 수 있도록 만듬

```
predicted = tf.cast(pred > 0.5, dtype=tf.float32)
```

정확도

- accuracy를 계산하여 분류가 정확한지 확인
- 예측값과 실제 데이터의 일치 여부 계산
- 아래 코드는 평균을 이용한 정확도 계산

```
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

1. Logistic Regression

모델 실행(run/update) (1/5)

- r_pred는 sigmoid 함수를 통해 0~1사이의 값으로 나온다
- r_c는 pred에서 나온 값을 0과 1로 변환 시킨 값이다
- r_a는 예측한 Y값이 실제 Y값과 얼마나 일치하는가

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_cnt):
    r_cost, r_W, r_b, r_pred, r_c, r_a, _ = sess.run([cost, W, b, 
pred, predicted, accuracy, op_train],
                                                     feed_dict = 
{X: train_X, Y: train_Y})
```

1. Logistic Regression

모델 실행(run/update) (2/5)

- true_Y는 실제 Training Output데이터이다.

```
        if (epoch+1) % display_step == 0:
        print("\nRun_count : [%04d], Train_cost =[%.4f], W
=[%.4f %.4f], b =[%.4f], \
               \npred =[%.4f %.4f %.4f %.4f %.4f %.4f], \
               \npred_Y = [%d %d %d %d %d %d ], \
               \ntrue_Y=  [%d %d %d %d %d %d ], \
               \naccuracy = [%.2f%%] "
              % (epoch+1, r_cost, r_W[0], r_W[1], r_b,
                 r_pred[0], r_pred[1], r_pred[2], r_pred[3],
r_pred[4], r_pred[5],
                 r_c[0], r_c[1], r_c[2], r_c[3], r_c[4], r_c[5],
                 train_Y[0], train_Y[1], train_Y[2], train_Y[3],
train_Y[4], train_Y[5],
                 r_a*100))

print("\nOptimization Finished!")
```

1. Logistic Regression

모델 실행(run/update) (3/5)

```
Run_count : [1000], Train_cost =[0.7163], W =[-0.1394 0.1032], b =[0.5376],
pred =[0.6385 0.5720 0.5466 0.6846 0.5028 0.5630],
pred_Y = [1 1 1 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [50.00%]

Run_count : [2000], Train_cost =[0.6474], W =[-0.0613 0.2498], b =[-0.2754],
pred =[0.5870 0.5569 0.4948 0.7007 0.5265 0.6028],
pred_Y = [1 1 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [66.67%]

Run_count : [3000], Train_cost =[0.5887], W =[0.0107 0.3858], b =[-1.0260],
pred =[0.5382 0.5435 0.4474 0.7159 0.5487 0.6389],
pred_Y = [1 1 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [66.67%]

Run_count : [4000], Train_cost =[0.5386], W =[0.0768 0.5124], b =[-1.7191],
pred =[0.4930 0.5313 0.4045 0.7304 0.5693 0.6714],
pred_Y = [0 1 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [83.33%]
```

1. Logistic Regression

모델 실행(run/update) (4/5)

```
Run_count : [5000], Train_cost =[0.4957], W =[0.1375 0.6303], b =[-2.3601],
pred =[0.4516 0.5202 0.3660 0.7439 0.5880 0.7002],
pred_Y = [0 1 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [83.33%]

Run_count : [6000], Train_cost =[0.4589], W =[0.1934 0.7400], b =[-2.9539],
pred =[0.4142 0.5100 0.3318 0.7564 0.6051 0.7258],
pred_Y = [0 1 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [83.33%]

Run_count : [7000], Train_cost =[0.4271], W =[0.2451 0.8424], b =[-3.5057],
pred =[0.3804 0.5005 0.3015 0.7679 0.6206 0.7483],
pred_Y = [0 1 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [83.33%]

Run_count : [8000], Train_cost =[0.3995], W =[0.2930 0.9382], b =[-4.0197],
pred =[0.3500 0.4917 0.2746 0.7786 0.6348 0.7682],
pred_Y = [0 0 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [100.00%]
```

1. Logistic Regression

모델 실행(run/update) (5/5)

```
Run_count : [9000], Train_cost =[0.3754], W =[0.3375 1.0279], b =[-4.5000],
pred =[0.3228 0.4835 0.2509 0.7883 0.6477 0.7857],
pred_Y = [0 0 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [100.00%]

Run_count : [10000], Train_cost =[0.3543], W =[0.3791 1.1122], b =[-4.9501],
pred =[0.2983 0.4757 0.2298 0.7972 0.6594 0.8012],
pred_Y = [0 0 0 1 1 1 ],
true_Y=  [0 0 0 1 1 1 ],
accuracy = [100.00%]

Optimization Finished!
```

감사합니다.

참고자료(Reference)

분류 문제 예시1

http://dongascience.donga.com/news.php?idx=14439
https://www.thealternativeboard.ie/제-

분류 문제 예시2

content/uploads/2017/03/382213-1303883183551-o1.jpg

분류 문제 예시3

https://www.goalprofits.com/제-content/uploads/2016/03/win-draw-win-bet-explained-image-740x430.jpg https://gigaom.com/제-content/uploads/sites/1/2013/07/shutterstock_131848391.jpg

로지스틱 회귀

https://florianhartl.com/logistic-regression-geometric-intuition.html

선형 판별 분석법

https://www.projectrhea.org/rhea/images/thumb/2/22/Hyperplane.png/700px-Hyperplane.png

시그모이드 함수

https://hvidberrrg.github.io/deep_learning/activation_functions/assets/sigmoid_function.png

로지스틱 비용 함수

https://blog.naver.com/ssse88/221190641711