딥러닝 구현을 위한 텐서플로우 개발

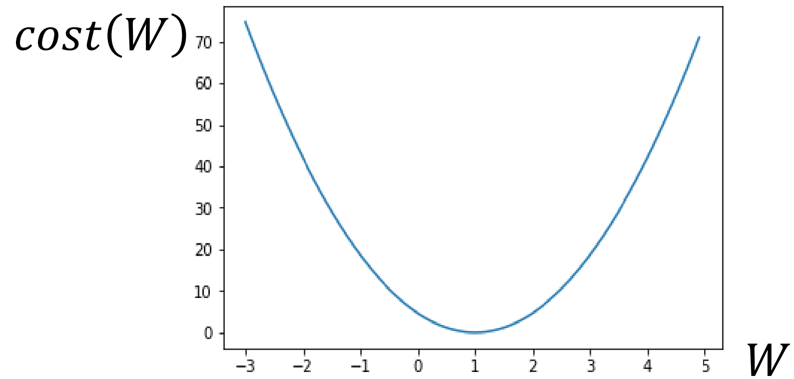
김성균

딥러닝 구현을 위한 텐서플로우 개발

3강
Logistic Regression

01. 선형회귀(Linear Regression)

- 추론함수 : $H(x) = Wx$
- 비용함수 : $cost(W) = \frac{1}{m}\sum_{i=1}^{m}(H(x_i) - y_i)^2$



- 경사도하강법(Gradient descent)

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx_i - y_i)x_i$$

- 다변량예측

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180

$$H(x_1, x_2, x_3) = W_1x_1 + W_2x_2 + W_3x_3$$

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} W_1 \\ W_2 \\ W_3 \end{pmatrix} = (x_1W_1 + x_2W_2 + x_3W_3)$$

02.로지스틱회귀(Logistic Regression)

- 사건의 발생 가능성을 예측하는데 사용되는 통계 기법
- D.R.Cox가 1958년에 제안한 확률 모델
- 의료, 통신, 데이터마이닝과 같은 다양한 분야에서 분류 및 예측을 위한 모델로서 폭넓게 사용
  - 이메일 스팸검출(스팸인가 아닌가?)
  - 신용카드 비정상거래검출(비정상 거래인가?)
  - 게임에서 어뷰징 사용자 검출(어뷰징 사용자인가?)

변수	변수명	척도	변수설명	하위범주
독립 변수	연령대	범주형	연령	1=20대, 2=30대, 3=40대, 4=50대, 5=60대 이상
	교육수준	범주형	교육정도	1=중졸이하, 2=고졸, 3=전문대, 4=대졸, 5=대학원 이상
	연봉	연속형	연봉	
종속 변수	분배정의	이분형	분배의 공정성에 대한 의견	0=불공정, 1=공정(이항) 1=불공정, 2=중립, 3=공정(다항)

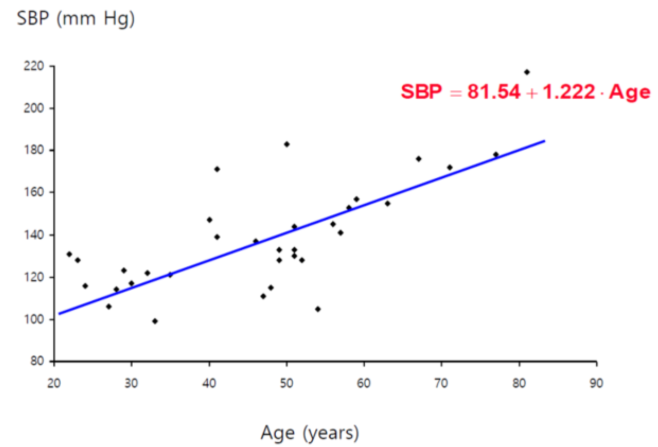
02.로지스틱회귀(Logistic Regression)

- 선형 회귀 모형과 유사하나 종속변수가 이분형인 모형에 적합
- 예) 33명의 성인 여성에 대한 나이와 혈압 데이터

Age	SBP
22	131
23	128
24	116
27	106
28	114
29	123
30	117
32	122
33	99
35	121
40	147

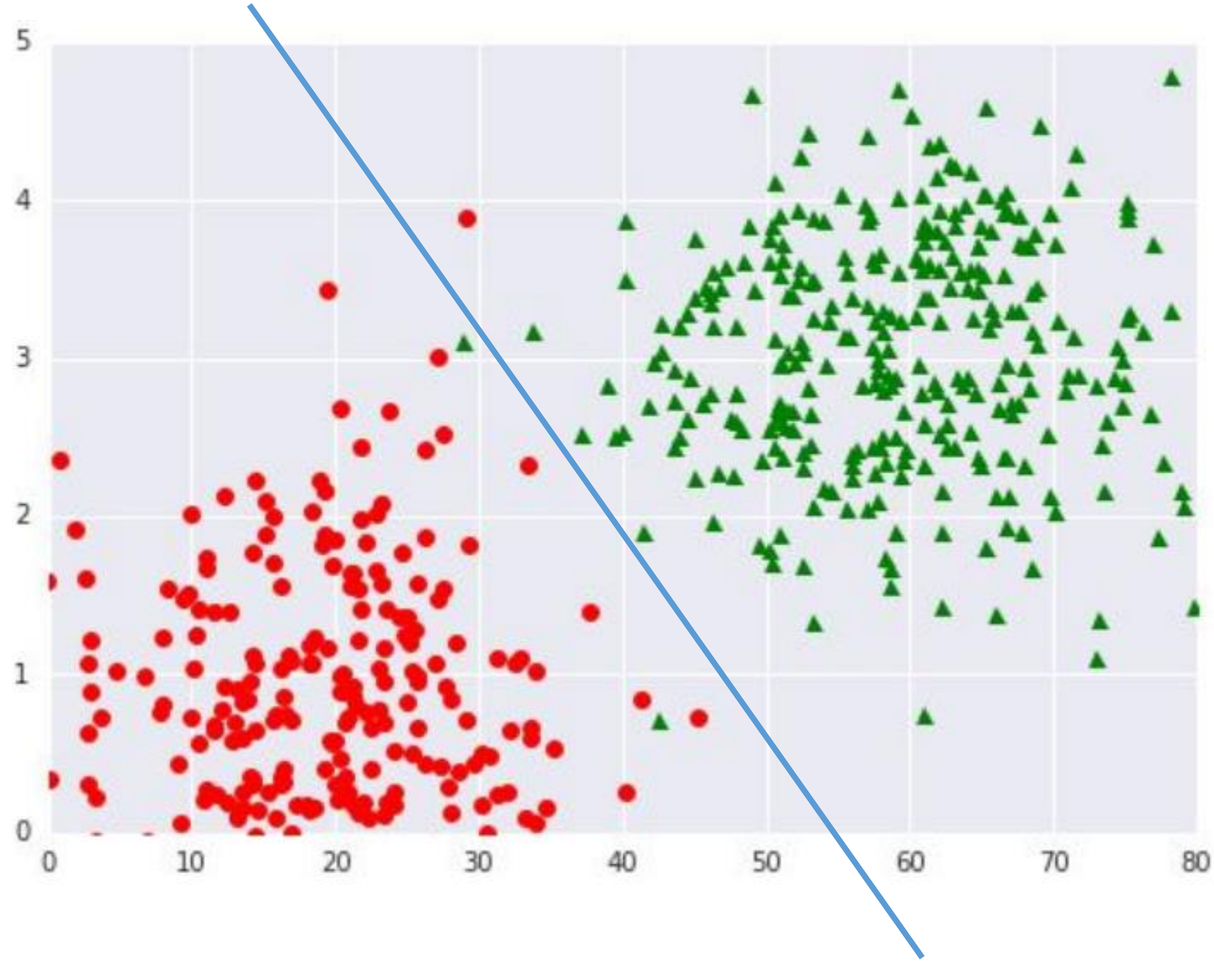
Age	SBP
41	139
41	171
46	137
47	111
48	115
49	133
49	128
50	183
51	130
51	133
51	144

Age	SBP
52	128
54	105
56	145
57	141
58	153
59	157
63	155
67	176
71	172
77	178
81	217



02.로지스틱회귀(Logistic Regression)

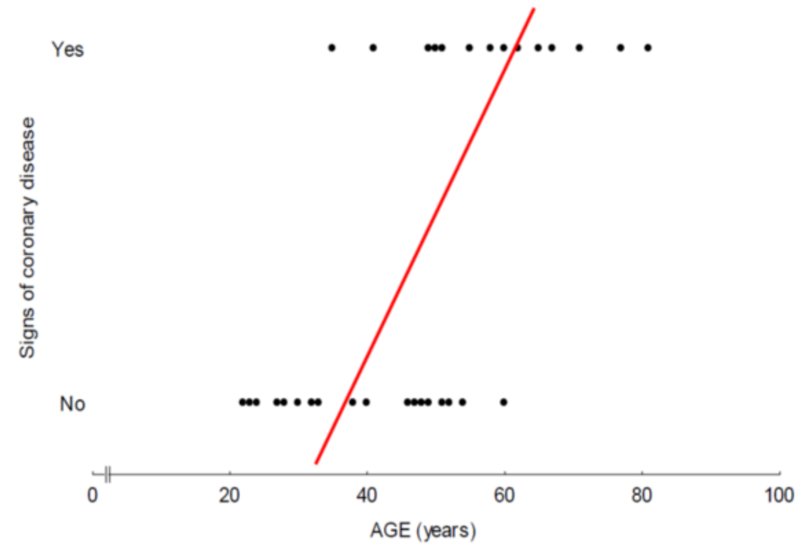
- 붉은색과 녹색을 분류할 수 있는 이상적인 직선 그래프를 찾는것



02.로지스틱회귀(Logistic Regression)

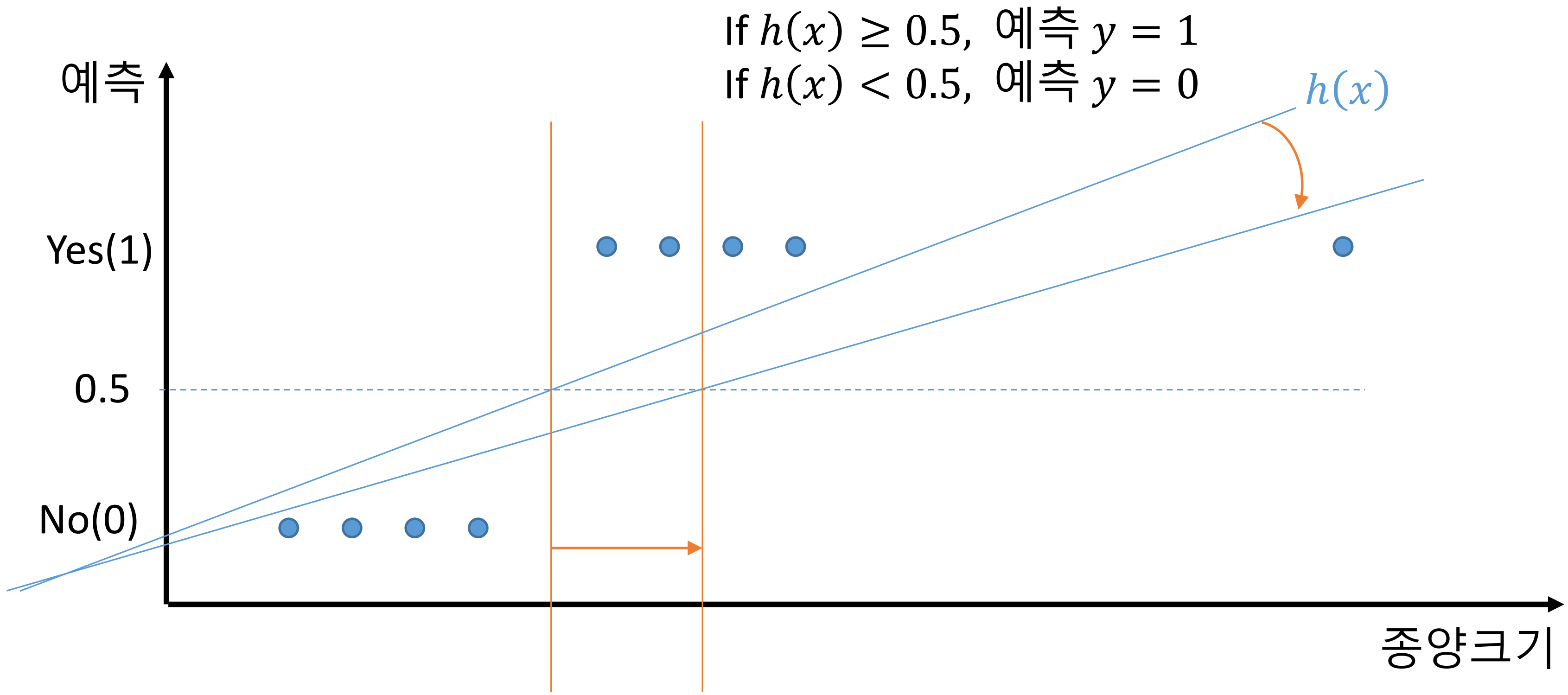
- 나이와 암 발생여부(1이면 발병, 0이면 정상)

Age	CD	Age	CD	Age	CD
22	0	40	0	54	0
23	0	41	1	55	1
24	0	46	0	58	1
27	0	47	0	60	1
28	0	48	0	60	0
30	0	49	1	62	1
30	0	49	0	65	1
32	0	50	1	67	1
33	0	51	0	71	1
35	1	51	1	77	1
38	0	52	0	81	1



- 문제가 발생하는 근본적인 이유는 종속변수 Y의 성질 때문
    - 발병(1)과 정상(0) 사이에 중간 범주가 없음
    - 정상을 1, 발병을 0으로 바꾸어도 문제가 없음
- Y가 범주형(categorical) 변수 => 다중선형회귀 모델을 그대로 적용할 수 없음

02.로지스틱회귀(Logistic Regression)



이항 분류모델은 0또는 1의 값을 갖는다

$H(x) = Wx + b$는
매우크거나 작은값을 갖을 수 있음

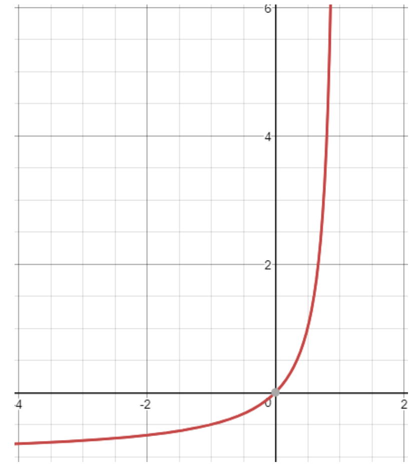
선형 회귀 분석 모델(Linear regression) 은
이항 분류에 적절하지 않다

https://www.coursera.org/learn/machine-learning/lecture/wlPeP/classification

02.로지스틱회귀(Logistic Regression)

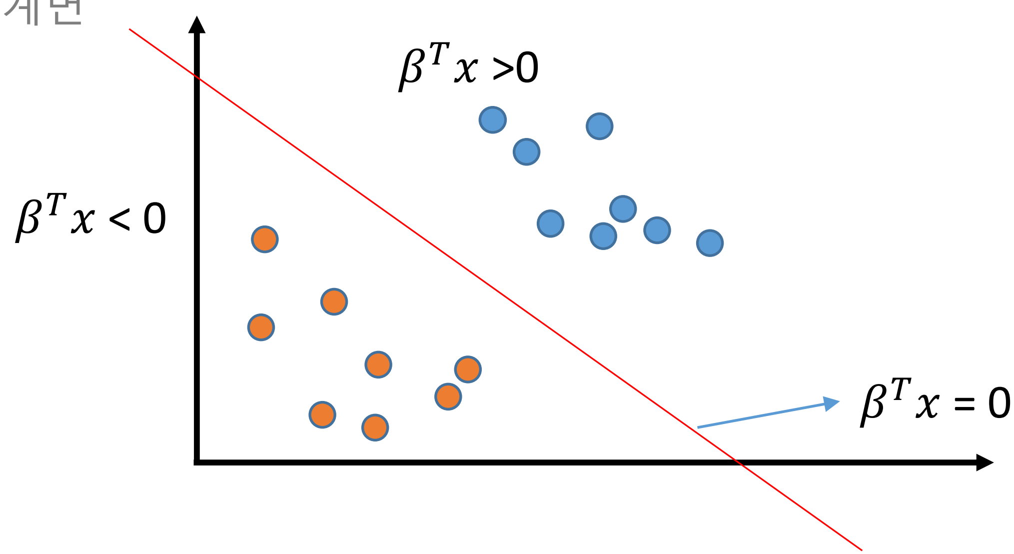
- 승산(Odds) – 임의의 사건 A가 발생하지 않을 확률 대비 일어날 확률의 비율

$$odds = \frac{P(A)}{P(A^c)} = \frac{P(A)}{1 - P(A)}$$



- P(A)가 1에 가까워질 수록 승산은 매우 커짐

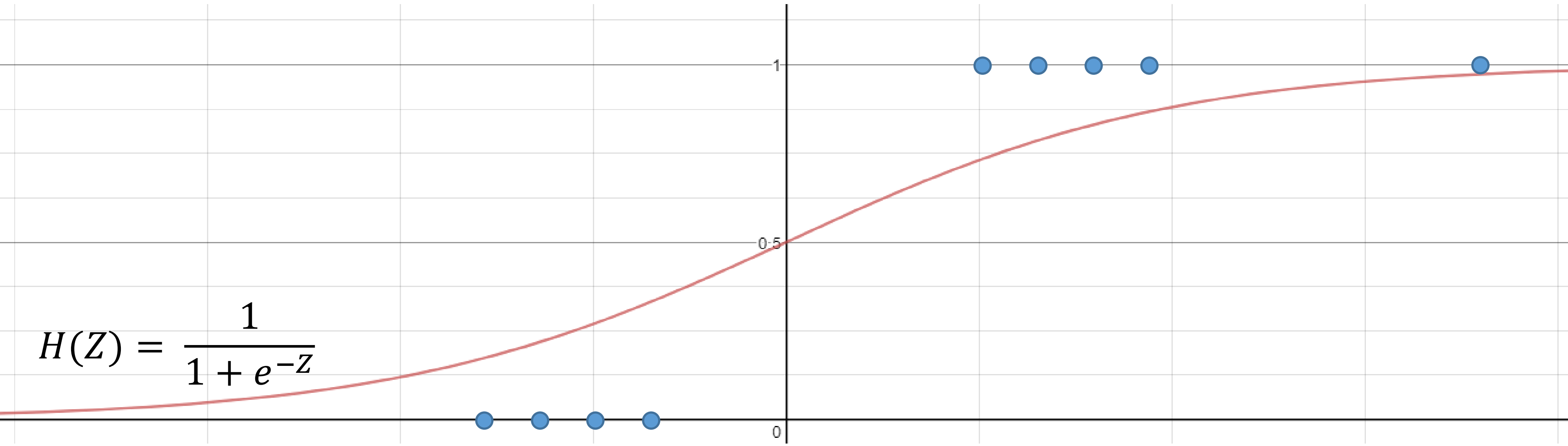
- 하이퍼플레인($\beta^T x$) :
  - 로지스틱 모델의 결정경계면



03.추론함수

- 비용 함수 (cost function)의 값이 최소가 되는 지점을 찾는 과정에서 추론함수가 미분되어야 하지만 $odds = \frac{P(A)}{1-P(A)}$ 는 미분불가능
- 따라서, 미분이 가능한 시그모이드(Sigmoid)함수를 사용

큰 데이타 (x=100)가 추가되어도 값이 1로 수렴되기 때문에,
앞의 선형 회귀 분석의 경우처럼 암의 양성/음성인 경우를 결정하는 예측결과가 변화 하지 않음



$$H(Z) = \frac{1}{1 + e^{-Z}}$$

$H(x) = sigmoid(Wx + b)$ $\mathrm{Z} = \mathrm{W} \cdot x + b$

$\mathrm{Z} < 0, y = 0$ 를 만족시키는
$\mathrm{Z} > 0, y = 1$ W, b를 구하는 문제

03.추론함수

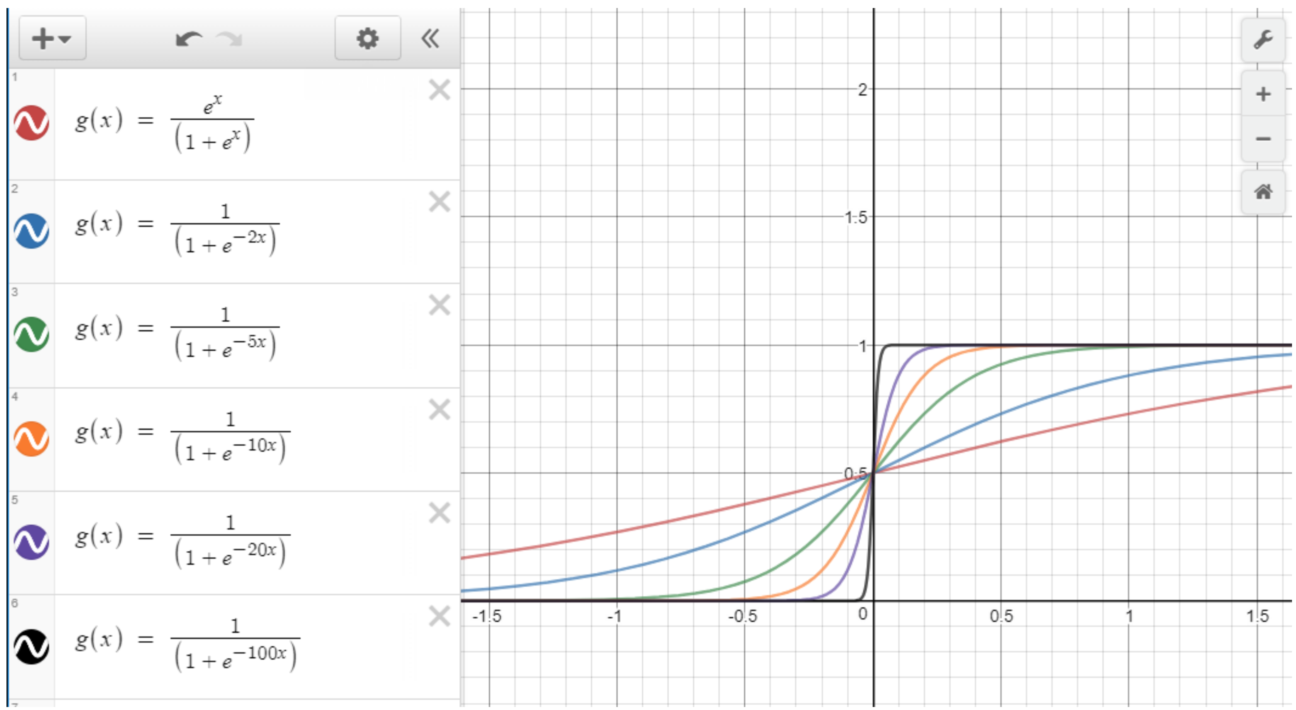
- 비용 함수 (cost function)의 값이 최소가 되는 지점을 찾는 과정에서 추론함수가 미분되어야 하지만 $odds = \frac{P(A)}{1-P(A)}$ 는 미분불가능
- 따라서, 미분이 가능한 시그모이드(Sigmoid)함수를 사용

$$\mathrm{H}(X) = \frac{e^{W \cdot X}}{1+e^{W \cdot X}}$$

$$= \frac{e^{W \cdot X}}{1+e^{W \cdot X}} \cdot \frac{\frac{1}{e^{W \cdot X}}}{\frac{1}{e^{W \cdot X}}}$$

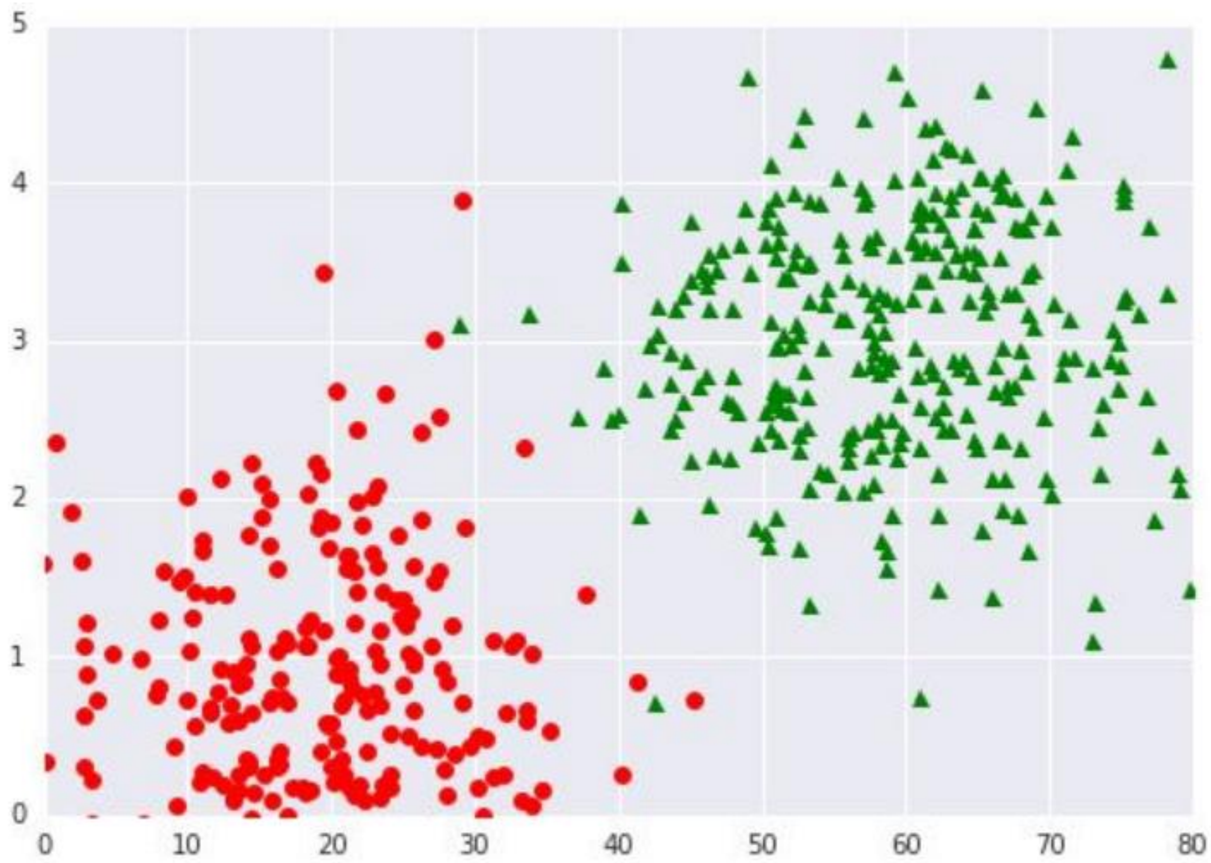
$$= \frac{1}{\frac{1}{e^{W \cdot X}}+1}$$

$$= \frac{1}{1+e^{-W \cdot X}}$$



https://www.desmos.com/calculator

03.추론함수



$$y = sigmoid\,(W1 * x1 + W2 * x2 + b)$$

02.비용함수(cost function)

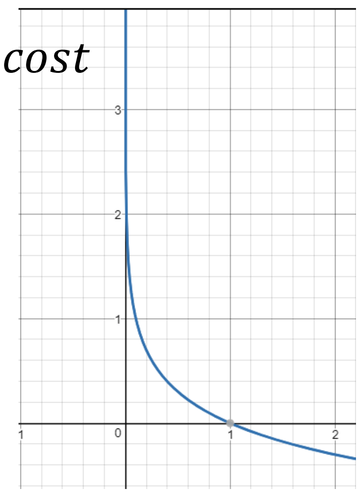
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m} C(H(x_i), y_i)$$

$C(H(x_i), y_i)$:x_i, y_i에서의 비용

$$C(H(x_i), y_i) = \begin{cases} -\log\left(H(x_i)\right) & : y = 1 \\ -\log\left(1 - H(x_i)\right) & : y = 0 \end{cases}$$

$-\log(H(x))$

$cost$



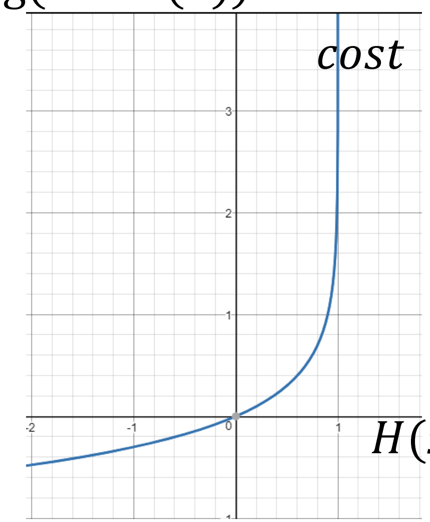
$H(x) = 0, cost = \infty$

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

$H(x)$

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

$cost$



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

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

$H(x)$

02.비용함수(cost function)

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m} C(H(x_i), y_i)$$

$C(H(x_i), y_i)$:x_i, y_i에서의 비용

$$C(H(x_i), y_i) = \begin{cases} -\log\left(H(x_i)\right) & : y = 1 \\ -\log\left(1 - H(x_i)\right) & : y = 0 \end{cases}$$

$$C(H(x_i), y_i) = -y_i log\left(H(x_i)\right) - (1 - y_i)\log(1 - H(x_i))$$

02. 경사도 하강법

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m} C(H(x_i), y_i)$$

$C(H(x_i), y_i)$:x_i, y_i에서의 비용

$$C(H(x_i), y_i) = \begin{cases} -\log\big(H(x_i)\big) & : y = 1 \\ -\log\big(1 - H(x_i)\big) & : y = 0 \end{cases}$$

$$C(H(x_i), y_i) = -y_i log\big(H(x_i)\big) - (1 - y_i)\log(1 - H(x_i))$$

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

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

train = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)
```

03.예제1

```
x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
y_data = [[0], [0], [0], [1], [1], [1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
```

03.예제1

```python
x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
y_data = [[0], [0], [0], [1], [1], [1]]

# placeholders for a tensor that will be always fed.
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')

# Hypothesis using sigmoid : tf.div(1., 1. + tf.exp(tf.matmul(X,W) +b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

train = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)
```

03.예제1

```
x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
y_data = [[0], [0], [0], [1], [1], [1]]

# placeholders for a tensor that will be always fed.
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')

# Hypothesis using sigmoid : tf.div(1., 1. + tf.exp(tf.matmul(X,W) +b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

train = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)

# Accuracy computation
# True if hypothesis > 0.5 else false
predicted = tf.cast(hypothesis > 0.5, dtype = tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

03.예제1

```
# Launch graph
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(10001):
    cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
    if i % 200 == 0:
        print(i, cost_val)

# Accruacy report
h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
print("₩nHypothesis: ", h, "₩nCrrect (Y): ", c , "₩nAccuracy: ", a)
```

04.전체코드

```python
LEARNING_RATE = 0.01

x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
y_data = [[0], [0], [0], [1], [1], [1]]

# placeholders for a tensor that will be always fed.
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')

# Hypothesis using sigmoid : tf.div(1., 1. + tf.exp(tf.matmul(X,W) +b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

train = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)

# Accuracy computation
# True if hypothesis > 0.5 else false
predicted = tf.cast(hypothesis > 0.5, dtype = tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(10001):
    cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
    if i % 200 == 0:
        print(i, cost_val)

# Accruacy report
h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
print("\nHypothesis: ", h, "\nCrrect (Y): ", c , "\nAccuracy: ", a)
```

```
0 1.62297
200 0.993332
400 0.757851
600 0.654146
800 0.600046
1000 0.565053
1200 0.53845
1400 0.516079
1600 0.496154
1800 0.477843
2000 0.46073
2200 0.444595
2400 0.429311
2600 0.414801
2800 0.401011
3000 0.387897
3200 0.375425
Hypothesis:  [[ 0.03888373]
 [ 0.1686022 ]
 [ 0.34172902]
 [ 0.76501077]
 [ 0.92879385]
 [ 0.97662121]]
Crrect (Y):  [[ 0.]
 [ 0.]
 [ 0.]
 [ 1.]
 [ 1.]
 [ 1.]]
Accuracy:  1.0
```

05.예제2

- 당뇨병 분류하기

-0.29412	0.487437	0.180328	-0.292929	0	0.00149028	-0.53117	-0.033333	0
-0.88235	-0.145729	0.0819672	-0.414141	0	-0.207153	-0.766866	-0.666667	1
-0.05882	0.839196	0.0491803	0	0	-0.305514	-0.492741	-0.633333	0
-0.88235	-0.105528	0.0819672	-0.535354	-0.777778	-0.162444	-0.923997	0	1
0	0.376884	-0.344262	-0.292929	-0.602837	0.28465	0.887276	-0.6	0
-0.41177	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.64706	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.76471	0.979899	0.147541	-0.090909	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.05882	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.52941	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1
-0.88235	0.899497	-0.016393	-0.535354	1	-0.102832	-0.726729	0.266667	0
-0.17647	0.00502513	0	0	0	-0.105812	-0.653288	-0.633333	0

```
xy = np.loadtxt("data-03-diabetes.csv", delimiter=",", dtype=np.float32)

x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

```python
import tensorflow as tf
import numpy as np

LEARNING_RATE = 0.01
X_DATA_COLS = 8

xy = np.loadtxt("data-03-diabetes.csv", delimiter=",", dtype=np.float32)

x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, X_DATA_COLS])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([X_DATA_COLS,1]), name = 'weight')
b = tf.Variable(tf.random_normal([1]), name = 'bias')

# Hypothesis using sigmoid : tf.div(1., 1. + tf.exp(tf.matmul(X,W) +b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

train = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)

# Accuracy computation
# True if hypothesis > 0.5 else false
predicted = tf.cast(hypothesis > 0.5, dtype = tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(10001):
    cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
    if i % 200 == 0:
        print(i, cost_val)

# Accruacy report
h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
print("₩nHypothesis: ", h, "₩nCrrect (Y): ", c , "₩nAccuracy: ", a)
```

```
0 1.14486
200 0.669222
400 0.586093
600 0.56637
800 0.557131
1000 0.550242
1200 0.544264
1400 0.538886
1600 0.534007
1800 0.529569
2000 0.525526
2200 0.521839
2400 0.518472

[ 0.89978296]
[ 0.76492989]
[ 0.68877327]
[ 0.83228236]
[ 0.73317266]
[ 0.89137626]]

 [ 1.]
 [ 1.]
 [ 1.]]
Accuracy:  0.772069
```

05.예제3

```
filename_queue = tf.train.string_input_producer(['data-03-diabetes.csv'], shuffle=False, name='filename_queue')

reader = tf.TextLineReader()
key, value = reader.read(filename_queue)

# Default values, in case empty columns. Also specifies the type of the decode result.
record_defaults = [[0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults = record_defaults)

train_x_batch, train_y_batch = tf.train.batch([xy[0: -1], xy[-1:]], batch_size = CONST_BATCH_SIZE)

# Default values, in case empty columns. Also specifies the type of the decode result.
record_defaults = [[0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults = record_defaults)

train_x_batch, train_y_batch = tf.train.batch([xy[0: -1], xy[-1:]], batch_size = CONST_BATCH_SIZE)
```

```python
import tensorflow as tf

CONST_BATCH_SIZE = 10

LEARNING_RATE = 0.01
X_DATA_COLS = 8

filename_queue = tf.train.string_input_producer(['data-03-diabetes.csv'], shuffle=False, name='filename_queue')

reader = tf.TextLineReader()
key, value = reader.read(filename_queue)

# Default values, in case empty columns. Also specifies the type of the decode result.
record_defaults = [[0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults = record_defaults)

train_x_batch, train_y_batch = tf.train.batch([xy[0: -1], xy[-1:]], batch_size = CONST_BATCH_SIZE)

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, X_DATA_COLS])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([X_DATA_COLS,1]), name = 'weight')
b = tf.Variable(tf.random_normal([1]), name = 'bias')

# Hypothesis using sigmoid : tf.div(1., 1. + tf.exp(tf.matmul(X,W) +b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

train = tf.train.GradientDescentOptimizer(learning_rate = LEARNING_RATE).minimize(cost)

# Accuracy computation
# True if hypothesis > 0.5 else false
predicted = tf.cast(hypothesis > 0.5, dtype = tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
sess = tf.Session()
sess.run(tf.global_variables_initializer())

coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess = sess, coord = coord)

a_sum = 0;
a_cnt = 0;
for i in range(10001):
    x_batch, y_batch = sess.run([train_x_batch, train_y_batch])
    cost_val, _ = sess.run([cost, train], feed_dict={X: x_batch, Y:y_batch})
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_batch, Y: y_batch})
    print("₩nHypothesis: ", h, "₩nCrrect (Y): ", c, "₩nAccuracy: ", a)
    a_sum += a
    a_cnt +=1
    if i % 200 == 0:
        print(i, cost_val)


coord.request_stop()
coord.join(threads)

# Accruacy report
print("₩nAccuracy: ", a_sum/a_cnt)
```

06. 연습

- https://www.kaggle.com