Logistic Regression Classifier ¶

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

$$cost(W) = -\frac{1}{m} \sum ylog(H(x)) + (1 - y)(log(1 - H(x))$$

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

EX. 당뇨병 예측 문제

-예측값이 1이면 당뇨병 O

-예측값이 0이면 당뇨병 X

In [2]:

```
import pandas as pd

df = pd.read_csv("./data/data-03-diabetes.csv", header=None)
df.head(10)
```

Out[2]:

	0	1	2	3	4	5	6	7	8
0	-0.294118	0.487437	0.180328	-0.292929	0.000000	0.001490	-0.531170	-0.033333	0
1	-0.882353	-0.145729	0.081967	-0.414141	0.000000	-0.207153	-0.766866	-0.666667	1
2	-0.058824	0.839196	0.049180	0.000000	0.000000	-0.305514	-0.492741	-0.633333	0
3	-0.882353	-0.105528	0.081967	-0.535354	-0.777778	-0.162444	-0.923997	0.000000	1
4	0.000000	0.376884	-0.344262	-0.292929	-0.602837	0.284650	0.887276	-0.600000	0
5	-0.411765	0.165829	0.213115	0.000000	0.000000	-0.236960	-0.894962	-0.700000	1
6	-0.647059	-0.216080	-0.180328	-0.353535	-0.791962	-0.076006	-0.854825	-0.833333	0
7	0.176471	0.155779	0.000000	0.000000	0.000000	0.052161	-0.952178	-0.733333	1
8	-0.764706	0.979899	0.147541	-0.090909	0.283688	-0.090909	-0.931682	0.066667	0
9	-0.058824	0.256281	0.573770	0.000000	0.000000	0.000000	-0.868488	0.100000	0

In [3]:

```
# Lab 5 Logistic Regression Classifier
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility

xy = np.loadtxt('./data/data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1] #확률
y_data = xy[:, [-1]] #당뇨병 유무

# print(x_data.shape, y_data.shape)
print(" x_data.shape : {x_shape} ₩n y_data.shape : {y_shape}".format(
        x_shape = x_data.shape,
        y_shape = y_data.shape
    ))

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([8, 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)))
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=0.01).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))
```

```
 x_data.shape : (759, 8)
 y_data.shape : (759, 1)
```

In [6]:

```
print(len(xy)) ,print(len(x_data)), print(len(y_data))
```

```
759
759
759
```

Out[6]:

```
(None, None, None)
```

In [5]:

```
xy[0]
```

Out[5]:

```
array([-0.294118  ,  0.487437  ,  0.180328  , -0.292929  ,  0.        ,
        0.00149028, -0.53117   , -0.0333333 ,  0.        ], dtype=float32)
```

In [8]:

```
from tqdm import tqdm_notebook

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in tqdm_notebook(range(10001)): #10000번 학습
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0 or step < 10 :
            print("Step : {} Wt Cost : {}".format(step, cost_val))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy],
                       feed_dict={X: x_data, Y: y_data})
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Step : 0        Cost : 0.8279397487640381
Step : 1        Cost : 0.8272486925125122
Step : 2        Cost : 0.8265633583068848
Step : 3        Cost : 0.8258837461471558
Step : 4        Cost : 0.8252099752426147
Step : 5        Cost : 0.8245416879653931
Step : 6        Cost : 0.823879063129425
Step : 7        Cost : 0.8232219815254211
Step : 8        Cost : 0.822570264339447
Step : 9        Cost : 0.8219239115715027
Step : 200      Cost : 0.755180835723877
Step : 400      Cost : 0.7263553738594055
Step : 600      Cost : 0.7051790356636047
Step : 800      Cost : 0.6866306066513062
Step : 1000     Cost : 0.6698529720306396
Step : 1200     Cost : 0.6546030044555664
Step : 1400     Cost : 0.64073646068573
Step : 1600     Cost : 0.6281296014785767
Step : 1800     Cost : 0.6166679859161377
Step : 2000     Cost : 0.6062455773353577
Step : 2200     Cost : 0.5967641472816467
Step : 2400     Cost : 0.5881334543228149
Step : 2600     Cost : 0.5802706480026245
Step : 2800     Cost : 0.5731005072593689
Step : 3000     Cost : 0.5665547847747803
Step : 3200     Cost : 0.5605713725090027
Step : 3400     Cost : 0.5550946593284607
Step : 3600     Cost : 0.5500746965408325
Step : 3800     Cost : 0.5454661846160889
Step : 4000     Cost : 0.5412290692329407
Step : 4200     Cost : 0.5373273491859436
Step : 4400     Cost : 0.5337287187576294
Step : 4600     Cost : 0.5304044485092163
Step : 4800     Cost : 0.5273289680480957
Step : 5000     Cost : 0.5244792103767395
```

```
Step : 5200      Cost : 0.521834671497345
Step : 5400      Cost : 0.5193769931793213
Step : 5600      Cost : 0.5170896649360657
Step : 5800      Cost : 0.5149580240249634
Step : 6000      Cost : 0.5129687190055847
Step : 6200      Cost : 0.5111098289489746
Step : 6400      Cost : 0.5093705654144287
Step : 6600      Cost : 0.5077412724494934
Step : 6800      Cost : 0.5062130689620972
Step : 7000      Cost : 0.5047780871391296
Step : 7200      Cost : 0.503429114818573
Step : 7400      Cost : 0.5021596550941467
Step : 7600      Cost : 0.5009635090827942
Step : 7800      Cost : 0.49983569979667664
Step : 8000      Cost : 0.4987708628177643
Step : 8200      Cost : 0.4977647662162781
Step : 8400      Cost : 0.4968130588531494
Step : 8600      Cost : 0.49591222405433655
Step : 8800      Cost : 0.495058536529541
Step : 9000      Cost : 0.4942488372325897
Step : 9200      Cost : 0.493480384349823
Step : 9400      Cost : 0.4927503168582916
Step : 9600      Cost : 0.4920562505722046
Step : 9800      Cost : 0.4913957715034485
Step : 10000     Cost : 0.4907667934894562
```

In [12]:

```
print("# Accuracy: {a} ₩n₩n # Hypothesis: ₩n{h} ₩n₩n# Correct (Y): ₩n{c}".format(
    # h = h, c = c, a = a ## 10000개는 너무 많으니까 일정 수만 보여주자
    h = h[:20], c = c[:20], a=a #어처피 정확도는 하나
))
```

```
# Accuracy: 0.7628458738327026

 # Hypothesis:
[[0.4434849 ]
 [0.9153646 ]
 [0.22591159]
 [0.93583125]
 [0.3376363 ]
 [0.70926887]
 [0.94409144]
 [0.6341791 ]
 [0.25953043]
 [0.4643435 ]
 [0.6474513 ]
 [0.20137012]
 [0.25898227]
 [0.35072374]
 [0.74845016]
 [0.48230034]
 [0.7001772 ]
 [0.9126371 ]
 [0.81194925]
 [0.56007695]]

# Correct (Y):
[[0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]]
```