FVE3011

Multivariable linear regression/ logistic regression classifier/ SoftMax classifier

담당교수: 최 학남 (xncui@inha.ac.kr)

인하대학교

Contents

- Multivariable linear regression
- logistic regression classifier
- SoftMax classifier

Multi-variable linear regression

- Hypothesis

$$H(x) = Wx + b$$

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$H(x_1, x_2, x_3, ..., x_n) = w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n + b$$

- Cost function(multi-variable)

$$cost(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right) - y^{(i)}\right)^2$$

Multivariable example: predicting score

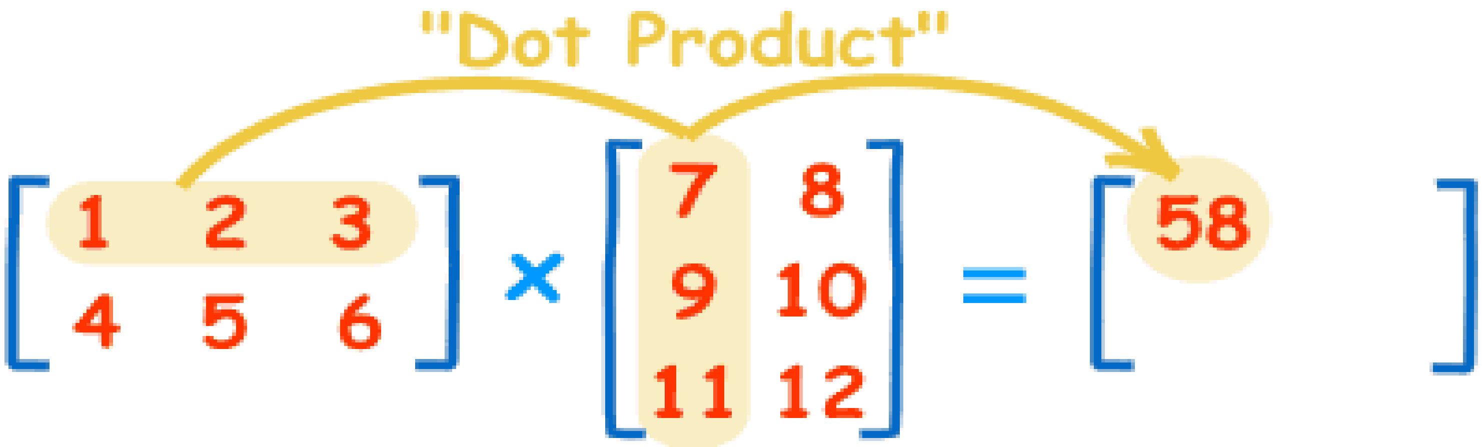
x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$H(x_1, x_2, x_3) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$cost(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right) - y^{(i)}\right)^2$$

Hypothesis using matrix

$$w_1 \; x_1 + w_2 \; x_2 \; + w_3 \; x_3 \; + \cdots + w_n \; x_n$$



$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = x_1 \; w_1 + x_2 \; w_2 \; + x_3 \; w_3$$

$$\boldsymbol{H(X) = XW}$$

https://www.mathsisfun.com/algebra/matrix-multiplying.html

Hypothesis using matrix

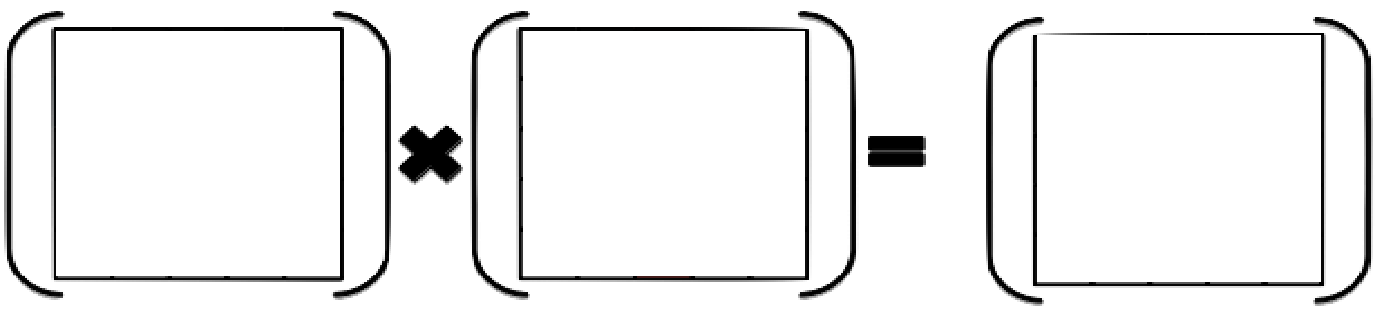
x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$\boldsymbol{H(X) = XW}$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

Hypothesis using matrix

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$



[5, 3] [?, ?] [5, 1]

$$H(X) = XW$$

Hypothesis using matrix

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \quad ? \quad = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

[n, 3] [?, ?] [n, 2]

$$H(X) = XW$$

WX vs XW

- Theory :

$$H(x) = Wx + b$$

- Implementation(TensorFlow)

$$H(X) = XW$$

Hypothesis without b

$$\begin{bmatrix} b & w1 & w2 & w3 \end{bmatrix} \times \begin{bmatrix} 1 \\ x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} b \times 1 + w1 \times x1 + w2 \times x2 + w3 \times x3 \end{bmatrix}$$

$$H(X) = WX$$

Multi-variable linear regression

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$H(x_1, x_2, x_3) = x_1w_1 + x_2w_2 + x_3w_3$$

```
# X and Y data
x_data = [[73., 80., 75.], [93., 88., 93.],
          [89., 91., 90.], [96., 98., 100.], [73., 66., 70.]]
y_data = [[152.], [185.], [180.], [196.], [142.]]

X = Variable(torch.Tensor(x_data))
Y = Variable(torch.Tensor(y_data))

# Our hypothesis XW+b
model = nn.Linear(3, 1, bias=True)
```

Multi-variable linear regression

```python
# Lab 4 Multi-variable linear regression
import torch
import torch.nn as nn
from torch.autograd import Variable

torch.manual_seed(777)   # for reproducibility

# X and Y data
x_data = [[73., 80., 75.], [93., 88., 93.],
          [89., 91., 90.], [96., 98., 100.], [73., 66., 70.]]
y_data = [[152.], [185.], [180.], [196.], [142.]]

X = Variable(torch.Tensor(x_data))
Y = Variable(torch.Tensor(y_data))

# Our hypothesis XW+b
model = nn.Linear(3, 1, bias=True)

# cost criterion
criterion = nn.MSELoss()

# Minimize
optimizer = torch.optim.SGD(model.parameters(), lr=1e-5)

# Train the model
for step in range(2001):
    optimizer.zero_grad()
    # Our hypothesis
    hypothesis = model(X)
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 10 == 0:
        print(step, "Cost: ", cost.data.numpy(), "\nPrediction:\n", hypothesis.data.numpy())
```

```
0 Cost:  19614.8
Prediction:
[ 21.69748688  39.10213089
31.82624626  35.14236832  3
2.55316544]
10 Cost:  14.0682
Prediction:
[ 145.56100464  187.949584
96  178.50236511  194.86721
802  146.08096313]

...

1990 Cost:  4.9197
Prediction:
[ 148.15084839  186.886322
02  179.6293335  195.81796
265  144.46044922]
2000 Cost:  4.89449
Prediction:
[ 148.15931702  186.880554
2  179.63194275  195.81971
741  144.45298767]
```

Hypothesis using matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1 w_1 + x_2 w_2 + x_3 w_3 \end{pmatrix} \qquad H(X) = XW$$

```
# X and Y data
x_data = [[73., 80., 75.], [93., 88., 93.],
          [89., 91., 90.], [96., 98., 100.], [73., 66., 70.]]
y_data = [[152.], [185.], [180.], [196.], [142.]]

X = Variable(torch.Tensor(x_data))
Y = Variable(torch.Tensor(y_data))

# Our hypothesis XW+b
model = nn.Linear(3, 1, bias=True)
```

Hypothesis using matrix

```python
# Lab 4 Multi-variable linear regression
import torch
import torch.nn as nn
from torch.autograd import Variable

torch.manual_seed(777)   # for reproducibility

# X and Y data
x_data = [[73., 80., 75.], [93., 88., 93.],
          [89., 91., 90.], [96., 98., 100.], [73., 66., 70.]]
y_data = [[152.], [185.], [180.], [196.], [142.]]

X = Variable(torch.Tensor(x_data))
Y = Variable(torch.Tensor(y_data))

# Our hypothesis XW+b
model = nn.Linear(3, 1, bias=True)

# cost criterion
criterion = nn.MSELoss()

# Minimize
optimizer = torch.optim.SGD(model.parameters(), lr=1e-5)

# Train the model
for step in range(2001):
    optimizer.zero_grad()
    # Our hypothesis
    hypothesis = model(X)
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 10 == 0:
        print(step, "Cost: ", cost.data.numpy(), "\nPrediction:\n", hypothesis.data.numpy())
```

```
0 Cost:  7105.46
Prediction:
[[ 80.82241058]
 [ 92.26364136]
 [ 93.70250702]
 [ 98.09217834]
 [ 72.51759338]]
10 Cost:  5.89726
Prediction:
[[ 155.35159302]
 [ 181.85691833]
 [ 181.97254944]
 [ 194.21760559]
 [ 140.85707092]]

...

1990 Cost:  3.18588
Prediction:
[[ 154.36352539]
 [ 182.94833374]
 [ 181.85189819]
 [ 194.35585022]
 [ 142.03240967]]
2000 Cost:  3.1781
Prediction:
[[ 154.35881042]
 [ 182.95147705]
 [ 181.85035706]
 [ 194.35533142]
 [ 142.036026  ]]
```

Load data from file

examdata.txt

```
# EXAM1,EXAM2,EXAM3,FINAL
73,80,75,152
93,88,93,185
89,91,90,180
96,98,100,196
73,66,70,142
53,46,55,101
```

```python
import numpy as np

xy = np.loadtxt('examdata.txt', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# Make sure the shape and data are OK
print(x_data.shape, x_data, len(x_data))
print(y_data.shape, y_data)
```

Slicing

```
nums = range(5)     # range is a built-in function that creates a list of integers
print nums          # Prints "[0, 1, 2, 3, 4]"
print nums[2:4]     # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print nums[2:]      # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print nums[:2]      # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print nums[:]       # Get a slice of the whole list; prints ["0, 1, 2, 3, 4]"
print nums[:-1]     # Slice indices can be negative; prints ["0, 1, 2, 3]"
nums[2:4] = [8, 9] # Assign a new sublist to a slice
print nums          # Prints "[0, 1, 8, 9, 4]"
```

Slicing

Indexing, Slicing, Iterating

- Arrays can be indexed, sliced, iterated much like lists and other sequence types in Python
- As with Python lists, slicing in NumPy can be accomplished with the colon (:) syntax
- Colon instances (:) can be replaced with dots (...)

```
a = np.array([1, 2, 3, 4, 5])
# array([1, 2, 3, 4, 5])

a[1:3]
# array([2, 3])

a[-1]
# 5

a[0:2] = 9

a
# array([9, 9, 3, 4, 5])
```

```
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
# array([[ 1,  2,  3,  4],
#        [ 5,  6,  7,  8],
#        [ 9, 10, 11, 12]])

b[:, 1]
# array([ 2,  6, 10])

b[-1]
# array([ 9, 10, 11, 12])

b[-1, :]
# array([ 9, 10, 11, 12])

b[-1, ...]
# array([ 9, 10, 11, 12])

b[0:2, :]
# array([[1, 2, 3, 4],
#        [5, 6, 7, 8]])
```

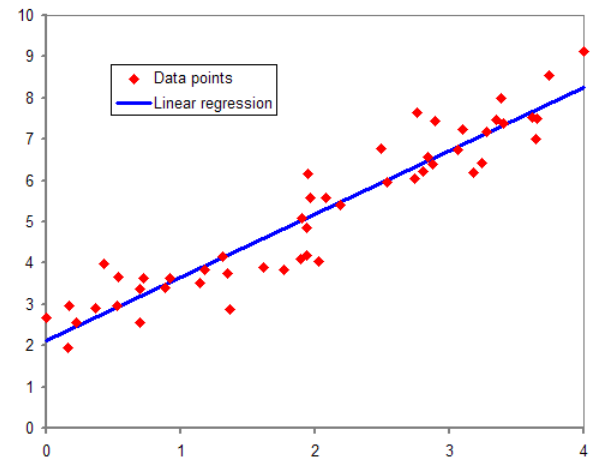
Multi-variable linear regression

```python
# Lab 4 Multi-variable linear regression
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np

torch.manual_seed(777)   # for reproducibility

xy = np.loadtxt('data-01-test-score.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# Make sure the shape and data are OK
print(x_data.shape, x_data, len(x_data))
print(y_data.shape, y_data)

x_data = Variable(torch.from_numpy(x_data))
y_data = Variable(torch.from_numpy(y_data))

# Our hypothesis XW+b
model = nn.Linear(3, 1, bias=True)

# cost criterion
criterion = nn.MSELoss()

# Minimize
optimizer = torch.optim.SGD(model.parameters(), lr=1e-5)
```

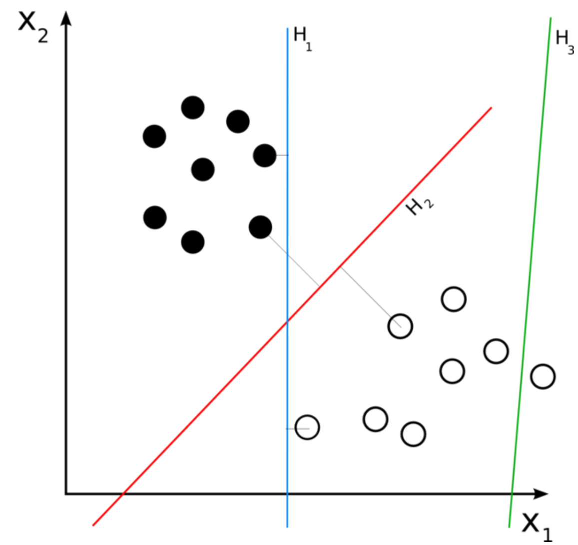
```python
# Train the model
for step in range(2001):
    optimizer.zero_grad()
    # Our hypothesis
    hypothesis = model(x_data)
    cost = criterion(hypothesis, y_data)
    cost.backward()
    optimizer.step()

    if step % 10 == 0:
        print(step, "Cost: ", cost.data.numpy(), "\nPrediction:\n", hypothesis.data.numpy())

# Ask my score
print("Your score will be ", model(Variable(torch.Tensor([[100, 70, 101]]))).data.numpy())
print("Other scores will be ", model(Variable(torch.Tensor([[60, 70, 110], [90, 100, 80]]))).data.numpy())
```

Logistic classification

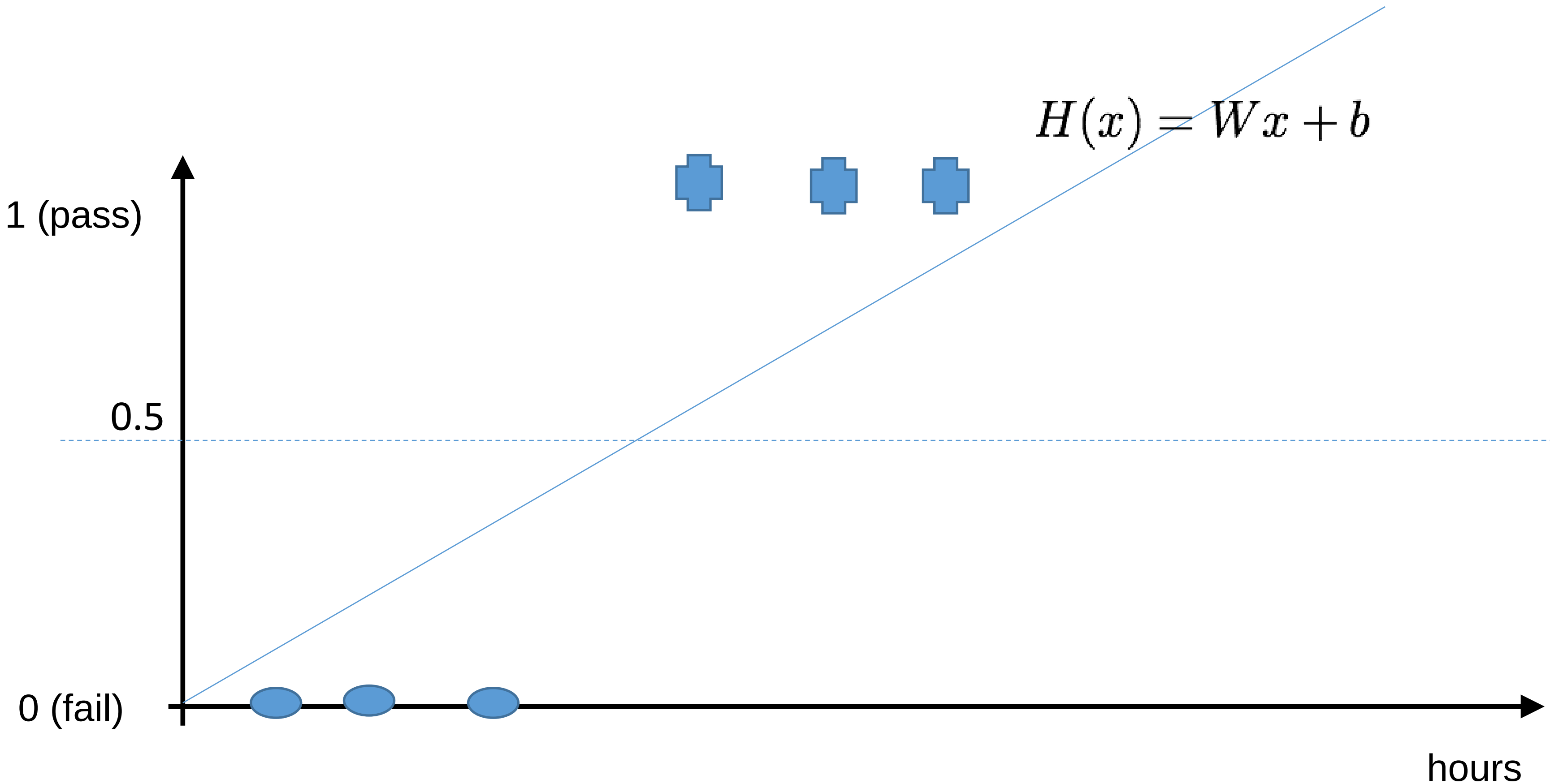
- Regression



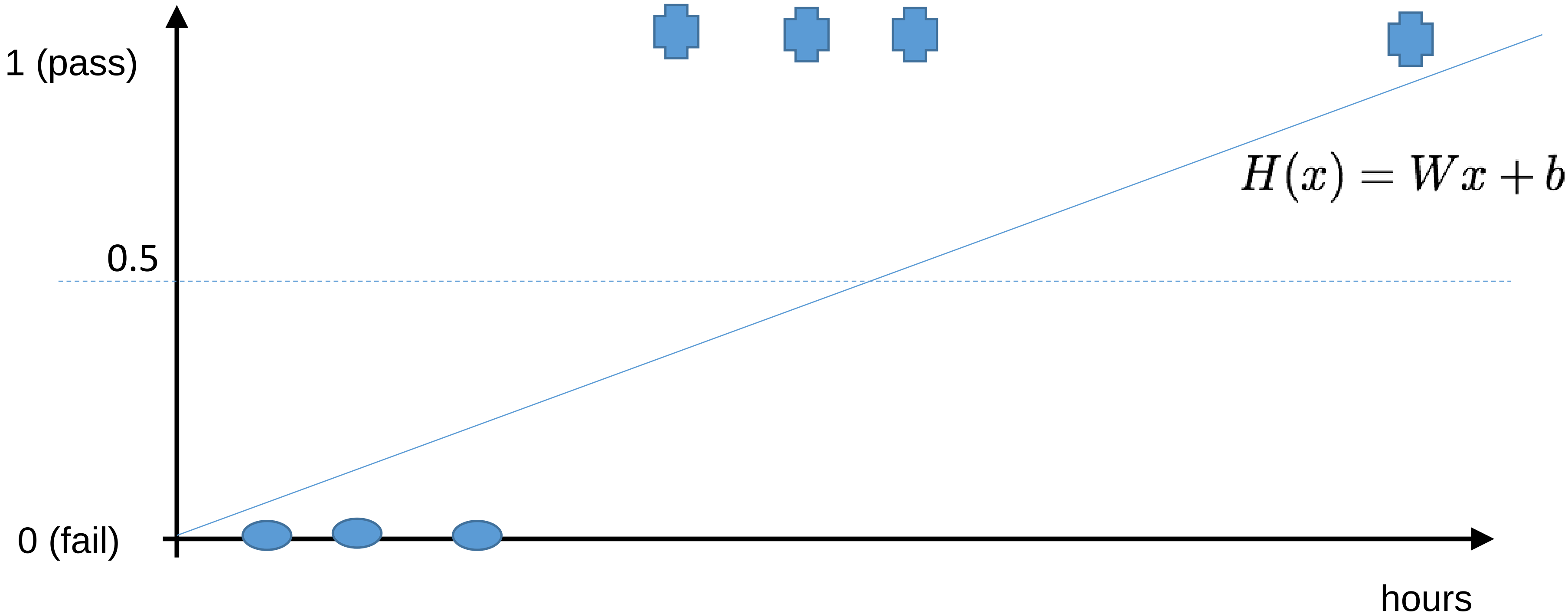
- Classification
  - Cat and dog
  - Cat(1) and dog(0)
  - Pass(1) and Fail(0)



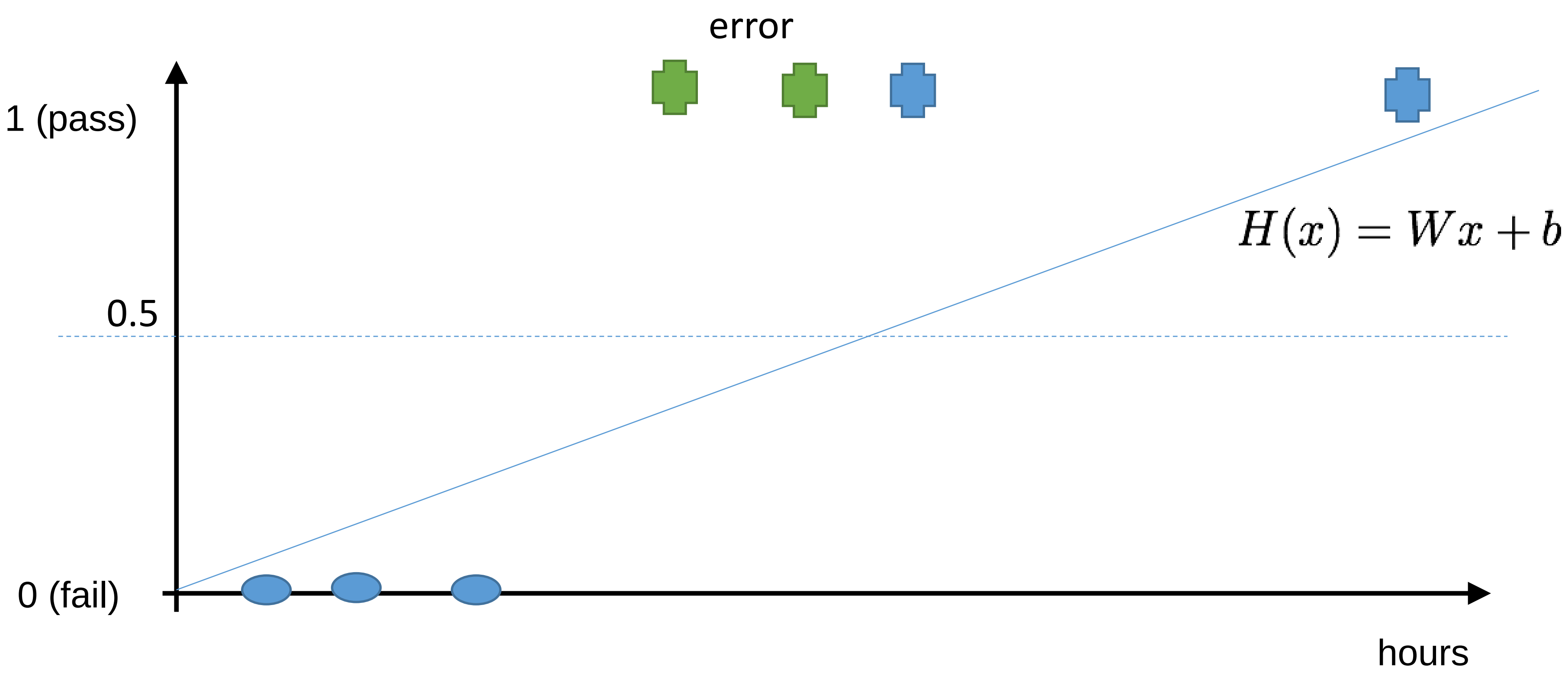
Logistic classification



Logistic classification



Logistic classification



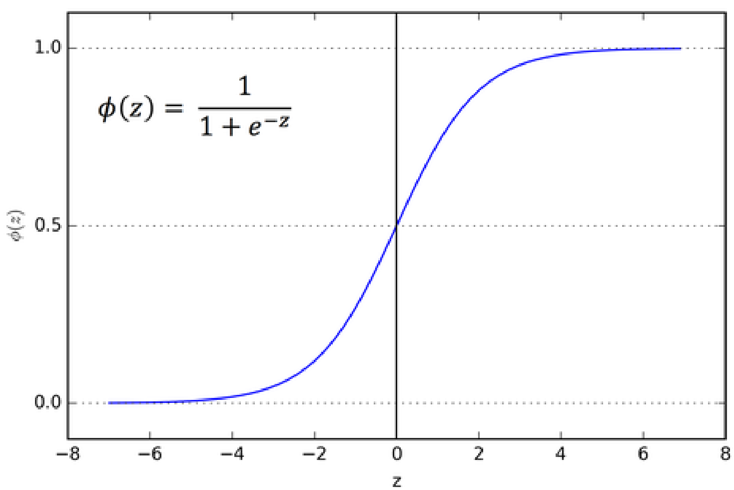
Logistic classification

- We know Y is 0 or 1
- Hypothesis can give values large than 1 or less than 0

$$H(x) = Wx + b$$

- Try to get the H(x) value between 0 and 1 → sigmoid

sigmoid function



$$Z = H(x) = Wx + b$$

Logistic classification

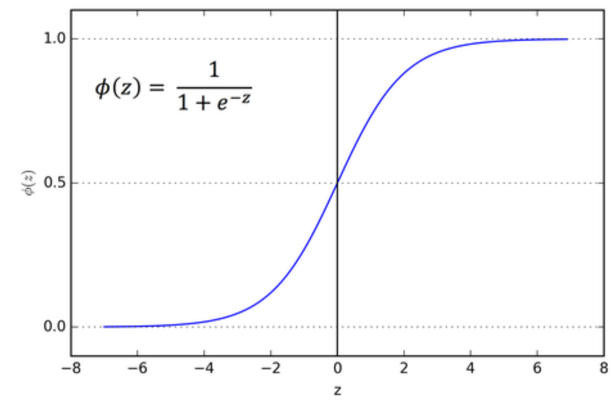
Logistic hypothesis(sigmoid function)

H:

$$H(x) = Wx + b$$

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

sigmoid function



C:

$$Cost(W) = \frac{1}{m}\sum C(H(x), y)$$

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

Remove if

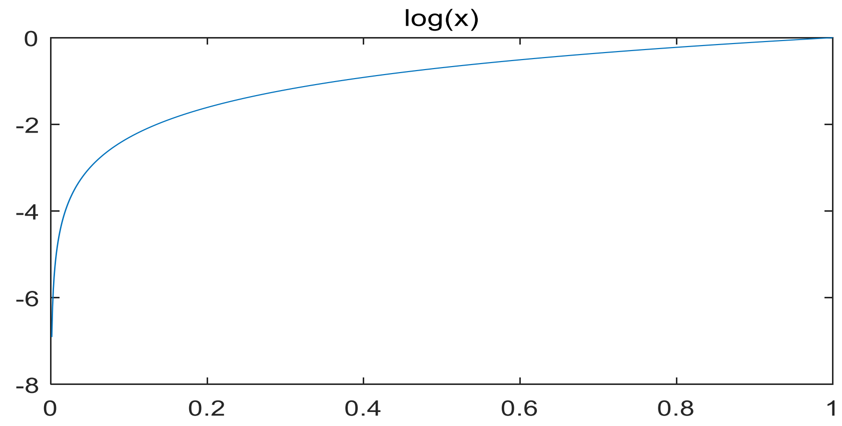
$$cost(W) = -\frac{1}{m}\sum y log(H(x)) + (1 - y)(log(1 - H(x))$$

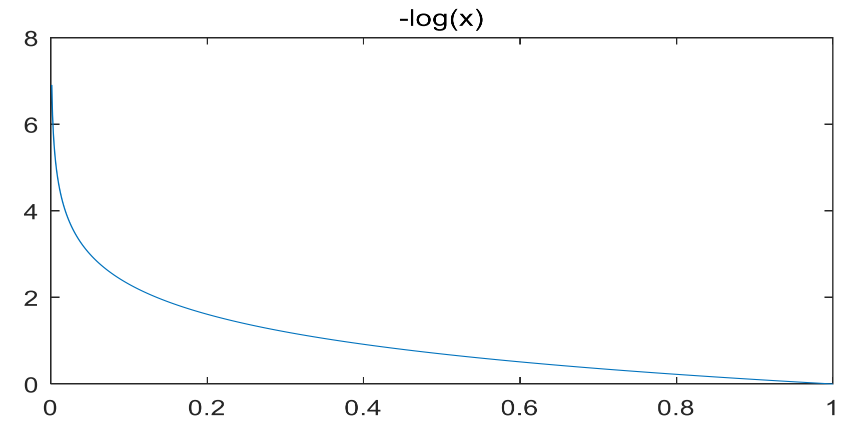
G:

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

Logistic classification

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





Logistic classification

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

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

Logistic classification

```
x_data = np.array([[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]], dtype=np.float32)
y_data = np.array([[0], [0], [0], [1], [1], [1]], dtype=np.float32)

X = Variable(torch.from_numpy(x_data))
Y = Variable(torch.from_numpy(y_data))

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
linear = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear, sigmoid)

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

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

Logistic classification

```python
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)
    # cost/loss function
    cost = -(Y * torch.log(hypothesis) + (1 - Y)
             * torch.log(1 - hypothesis)).mean()
    cost.backward()
    optimizer.step()

    if step % 200 == 0:
        print(step, cost.data.numpy())

# Accuracy computation
predicted = (model(X).data > 0.5).float()
accuracy = (predicted == Y.data).float().mean()
print("\nHypothesis: ", hypothesis.data.numpy(), "\nCorrect (Y): ", predicted.numpy(), "\nAccuracy: ", accuracy)
```

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

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

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

HW : diabetes

1. 주어진 당뇨병 데이터를 이용하여 당뇨병 여부를 판단할 수 있는 모델을 디자인 하고 학습 하시오.
   - Training set: testing set = 50:50
   - Training set: testing set = 70:30
   - Training set: testing set = 80:20

당뇨 여부

-0.294118	0.487437	0.180328	-0.292929	0	0.00149028	-0.53117	-0.0333333	0
-0.882353	-0.145729	0.0819672	-0.414141	0	-0.207153	-0.766866	-0.666667	1
-0.0588235	0.839196	0.0491803	0	0	-0.305514	-0.492741	-0.633333	0
-0.882353	-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

검사지표

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