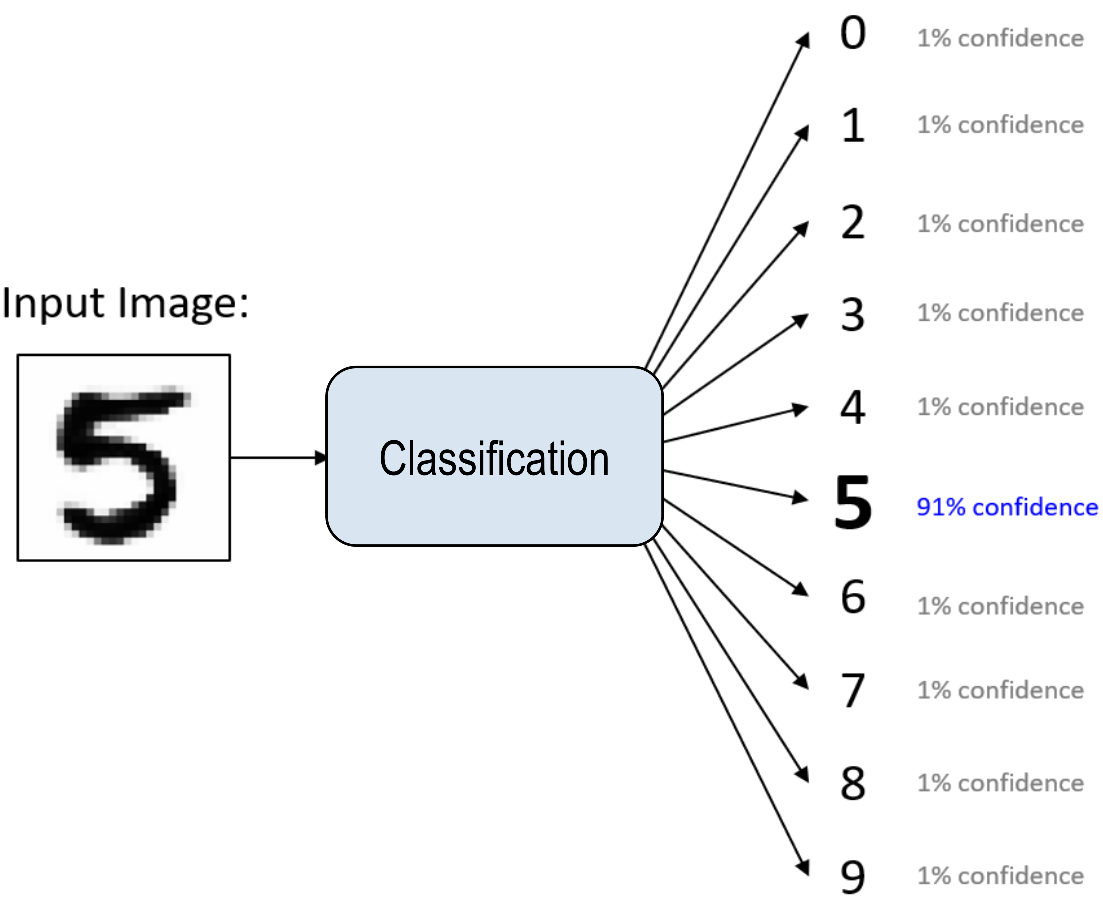

Disgust
Surprise
Sad
Angry
Fear
Happy
Contempt
Neutral

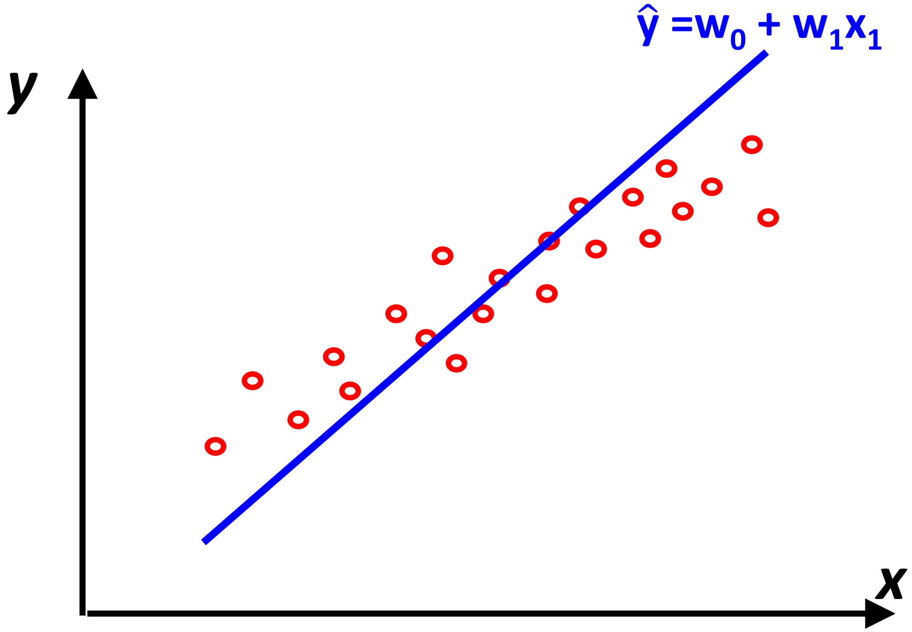

Input Image:
Classification
0 1% confidence
1 1% confidence
2 1% confidence
3 1% confidence
4 1% confidence
5 91% confidence
6 1% confidence
7 1% confidence
8 1% confidence
9 1% confidence

Logistic Regression

Hanwool Jeong

hwjeong@kw.ac.kr

Review on Linear Regression



Prediction ➔ $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2)$

MLE Training ➔ $\arg\max_{\mathbf{w}} p(D|\mathbf{w}) = \arg\max_{\mathbf{w}} \log p(D|\mathbf{w})$

$= \arg\max_{\mathbf{w}} \Sigma \log \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2)$

You remember Gradient Descent?

Maximum Likelihood (MLE) vs. Cost Minimization

- What is the relationship between MLE vs. cost minimization?
  ➔ MLE = cost minimization w/ certain cost func.
- Revisit that MLE is $\arg\max_{\boldsymbol{w}} p(D|\mathbf{w}) = \arg\max_{\boldsymbol{w}} \log p(D|\mathbf{w})$
- When we define the negative log-likelihood (NLL) as

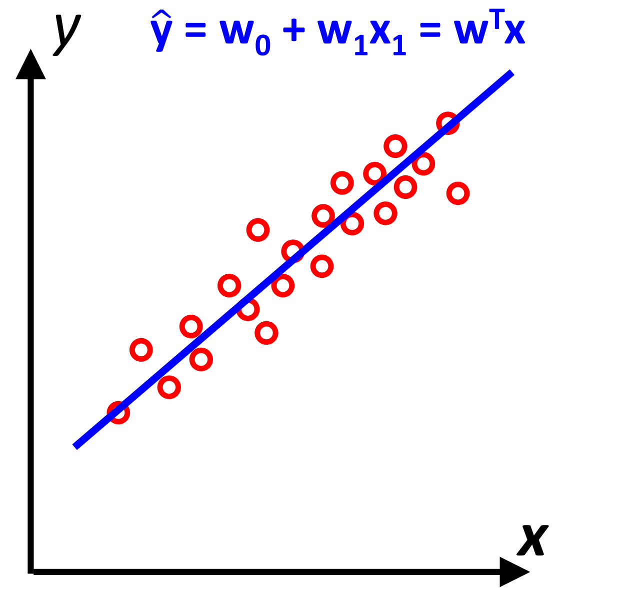
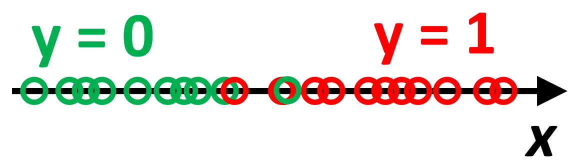
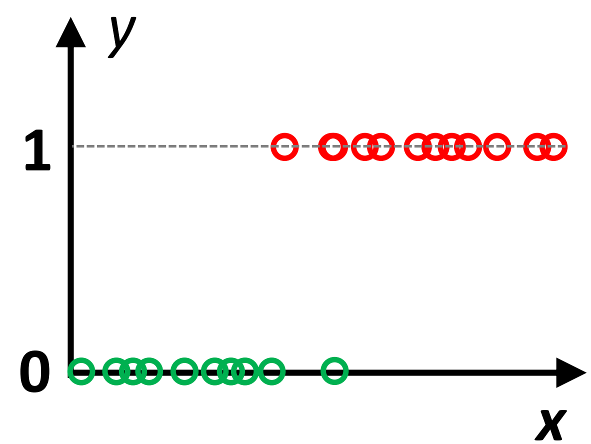
$$\text{NLL} = -\log p(D|\mathbf{w}) = -\sum_{i=1}^{N} \log p(yi|\boldsymbol{x}_i, \mathbf{w})$$

  ➔ That is, minimizing NLL = MLE.
- In many cases, NLL is used as the cost function.
  ➔ MLE is the cost minimizing with cost function = NLL
- Then, we can derive w* by repeatedly performing :

$$w_{next} = w_{present} - \eta \nabla NLL(\boldsymbol{w})$$ **: Gradient Descent**

Classification vs. Regression

- What we learned previously is linear "regression."

Regression	Classification
	 
$p(y\|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y\|\mathbf{w}^T\mathbf{x}, \sigma^2)$	$p(y\|\mathbf{x}, \mathbf{w}) = ?$

- We have to derive "**decision criterion**" represented by **w.**

Higher Dimensional or Multi-Class

- There will be more complex classifications
- We will first cover "binary" classification with 1D ***x***! That is,

$$y \in \{0, 1\}$$

What We Need First is Probability Model

Usually, an algorithm is named by this!

- That is, the form of PMF such as

$$p(D) = p(y_1=1/x_1)\times p(y_2=1/x_2)\times p(y_3=0/x_3)\times p(y_4=1/x_4)\times p(y_5=0/x_5)\times \ldots$$

$$p(y_i = 0/x_i) = w_1 x_i \qquad p(y_i = 1/x_i) = 1 - w_1 x_i$$

$$p(y_i = 1/x_i) = \cos^2(w_0 + w_1 x_i) \qquad p(y_i = 0/x_i) = \sin^2(w_0 + w_1 x_i)$$

$$p(y_i = 0/x_i) \begin{cases} 0.7 & x_i > w_0 \\ 0.3 & x_i < w_0 \end{cases} \qquad p(y_i = 1/x_i) \begin{cases} 0.3 & x_i > w_0 \\ 0.7 & x_i < w_0 \end{cases}$$

➔ By parameterizing **w**, p(D) or p(y/x) becomes p(D/**w**) or p(y/x,**w**)

- Then, we can decide w based on MLE, or equivalently, minimizing NLL.

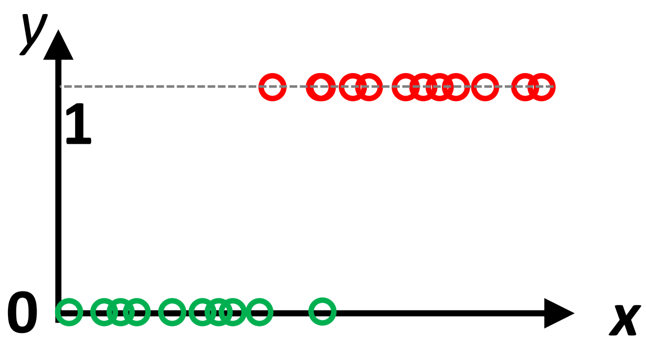
Training Strategy First!

- With proper p(D|**w**), the following is defined as the cost:

$$\text{NLL} = -\log p(D|\mathbf{w})$$
$$= -\sum_{i=1}^{N} \log p(yi=\text{class}|\mathbf{x}_i, \mathbf{w}) = -\sum_{i=1}^{N} \log p(yi=\text{class}|\mathbf{x}_i, \mathbf{w})$$

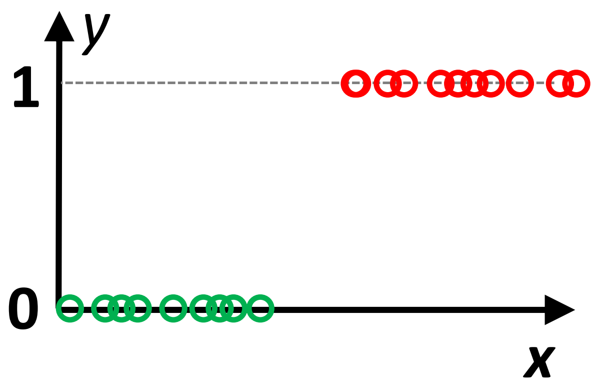
- For binary classification, only $p(y_i=1|\boldsymbol{x}_i,\boldsymbol{w})$ is needed
- Then, we can apply Gradient Descent to find optimal w*

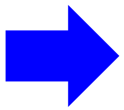
$$w_{\text{next}} = w_{\text{present}} - \eta \nabla NLL(\boldsymbol{w})$$

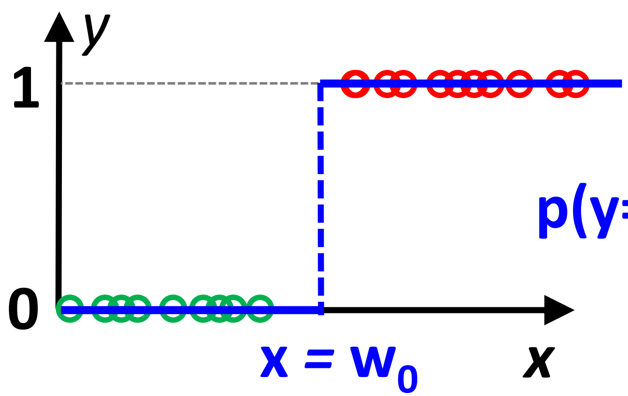


Example of p(D|w)

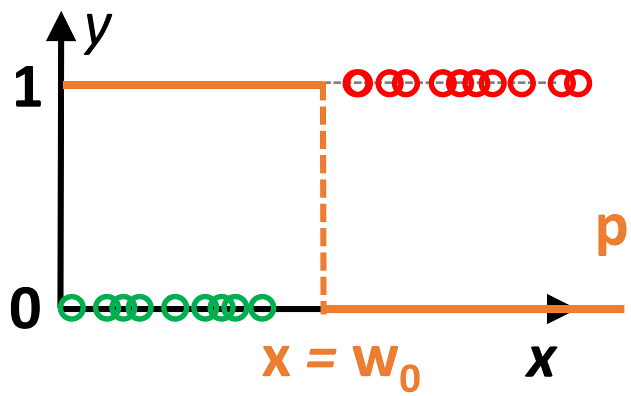
- How about p(D|w) as follows?







$$p(y=1|w) = \begin{cases} 1, & x > w_0 \\ 0, & x < w_0 \end{cases}$$

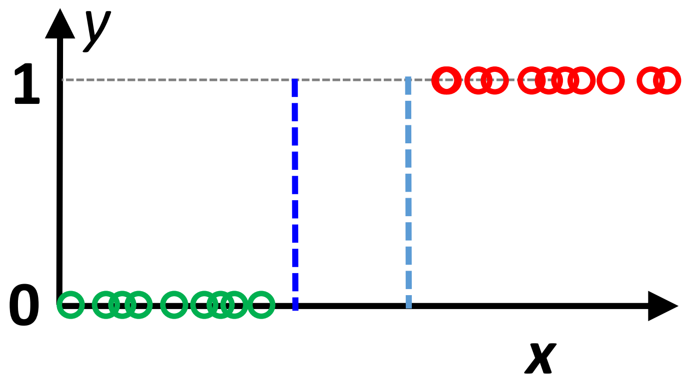


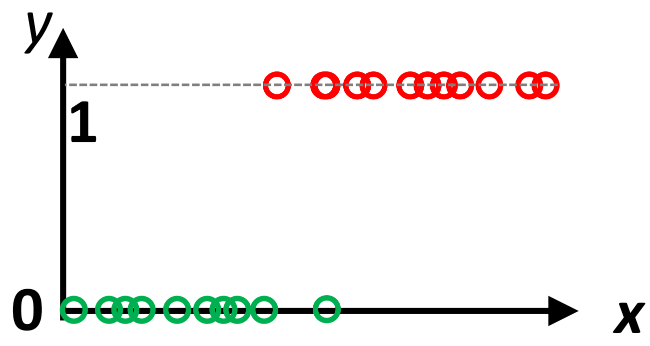
$$p(y=0|w) = \begin{cases} 0, & x > w_0 \\ 1 & x < w_0 \end{cases}$$

- Can you determine NLL?

Limitation of Abrupt & Extreme p(D|w)

- Can you decide the optimal **w** thru Gradient Descent?





- You cannot distinguish among different **w**'s or the optimal **w** does not exist at all.

We Need Soft Decision Criterion!

- To find the best **w** during the training process, we need soft boundary that can answer the following question:

To what degree does $\mathbf{x}_i$ belong to $y_i=1$ for the given **w?**

- in terms of $p(y_i=1/x_i,\mathbf{w})$. Not,

Does $\mathbf{x}_i$ belong to y=1 for the given **w?**

- Then, for the prediction phase, we will be able to answer like, for example,

For given ***x***, it will be classified into y=1 with prob = 0.7

- Not just,

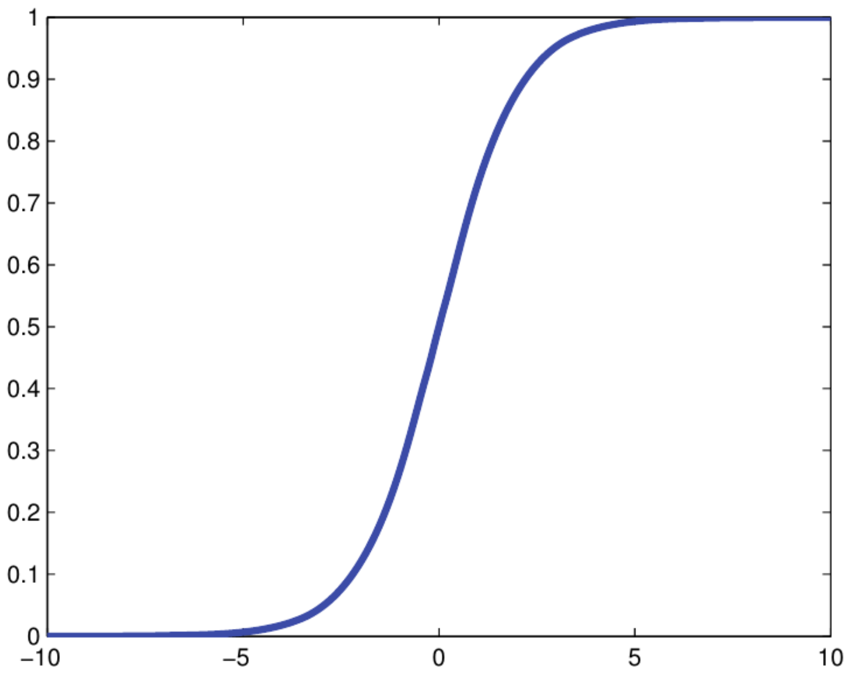
For given ***x***, it will be classified into y=1

Background; Sigmoid Function

- Sigmoid function sigm(x) is defined as

$$\text{sigm}(x) \triangleq \frac{1}{1+exp(-x)} = \frac{e^x}{1+e^x}$$

- The term sigmoid means S-shaped.



✓ We can use this for denoting p(y=1/**x**, **w**)

✓ ***w*** can change the exact shape of pmf by sigm($\mathbf{w}^T\mathbf{x}$)

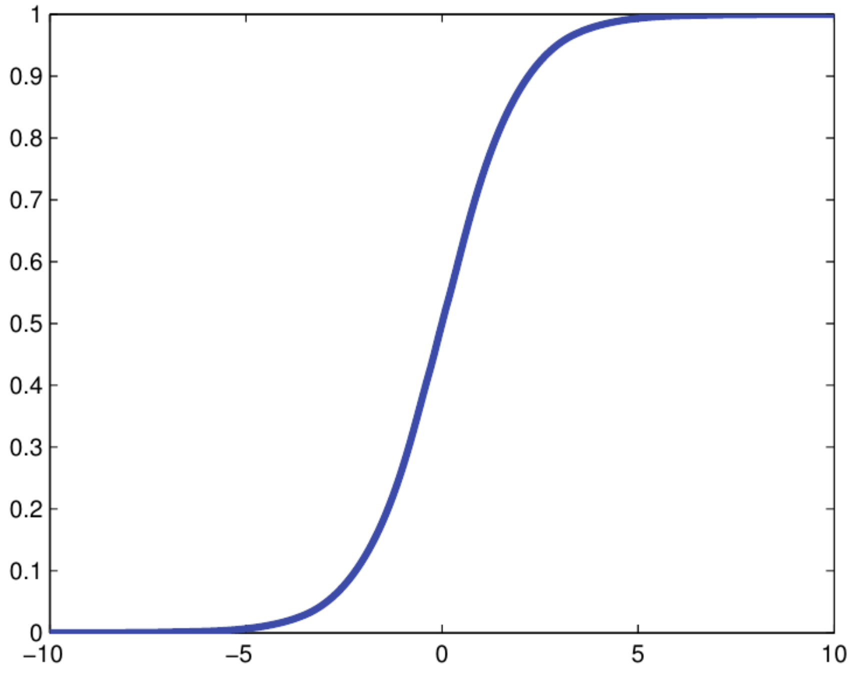
Linear Transformation in General Function

- Do you know the shape difference between f(x) vs. f(x+k)?
- Do you know the shape difference between f(x) vs. f(ax)?
- Do you know the shape difference between f(x) vs. f(ax+b)?

Shape of Sigmoid Function

- Can you apply linear transformation?

$$\text{sigm}(x) \triangleq \frac{1}{1+exp(-x)} = \frac{e^x}{1+e^x} \Rightarrow \text{sigm}(w_1 x + w_0) = \frac{\exp(w_1 x + w_0)}{1+\exp(w_1 x + w_0)}$$



Expressing PMF w/ Sigmoid

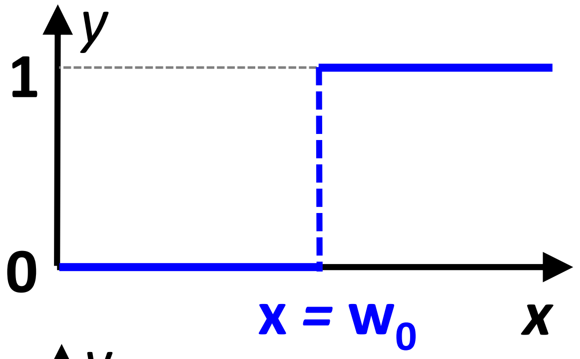
$$p(y=1|w) = \begin{cases} 1, & x > w_0 \\ 0, & x < w_0 \end{cases}$$

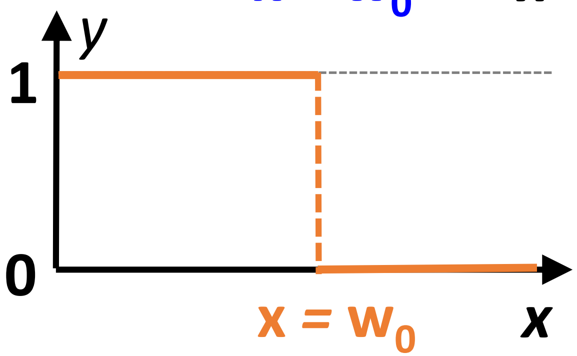
$$p(y=0|w) = \begin{cases} 0, & x > w_0 \\ 1 & x < w_0 \end{cases}$$

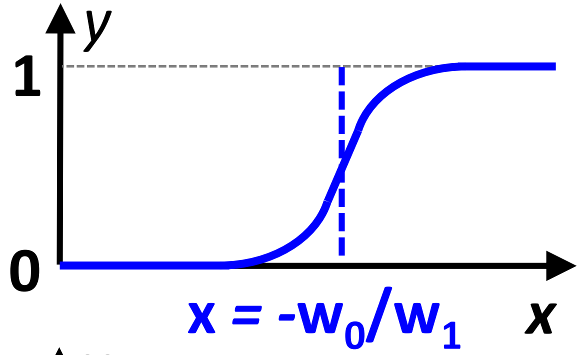
$$p(y=1|x, w) = sigm(w_1x+w_0)$$

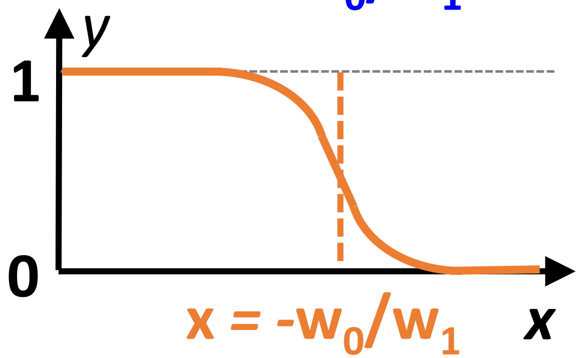
$$p(y=0|x, w) = 1 - sigm(w_1x+w_0)$$

$$sigm(x) = \frac{e^x}{1+e^x}$$









Background; Bernoulli Distribution

- Let Y ∈ {0, 1} be a binary discrete random variable, with the probability that p(y=1) is θ.
- We say that Y follows a Bernoulli distribution Y ~ Ber(θ) and the probability mass function is defined as

$$\mathrm{Ber}(y|\theta) = \begin{cases} \theta & ,y = 1 \\ 1 - \theta & ,y = 0 \end{cases}$$

- Or we can write as

Remember that θ is p(y=1), which is mean of y, μ(y)

$$\mathrm{Ber}(y|\theta) = \theta^{\mathbb{I}(y=1)}(1-\theta)^{\mathbb{I}(y=0)}$$

where

$$\mathbb{I}(x=k) = \begin{cases} 1 & \text{for } x = k \\ 0 & otherwise \end{cases}$$

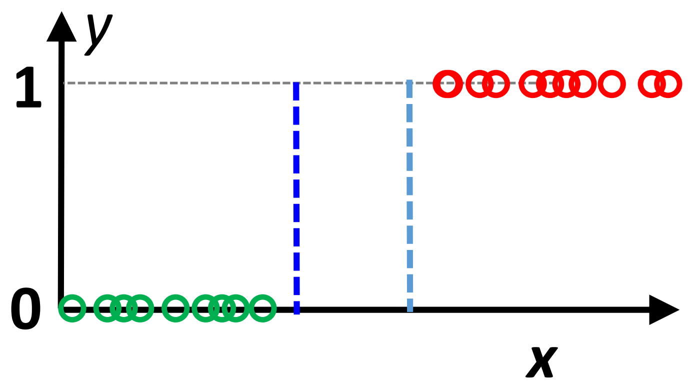
Determination of $p(D|w) = p(y|\vec{x}, \vec{w})$ for Classification

- Putting these two previous concepts together, p(y|**x**, **w**) can be determined as

$$p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}(y|\mathrm{sigm}(\mathbf{w}^T\mathbf{x}))$$

$$p(y=1|x, w) = \mathrm{sigm}(w_1x+w_0)$$
$$p(y=0|x, w) = 1 - \mathrm{sigm}(w_1x+w_0)$$

- How will be the shape of the sigm changed according to **w**?
- Sigmoid function is also known as logistic or logit function.
- This is called **logistic regression** due to its similarity to linear regression (although it is a form of classification, not regression!)

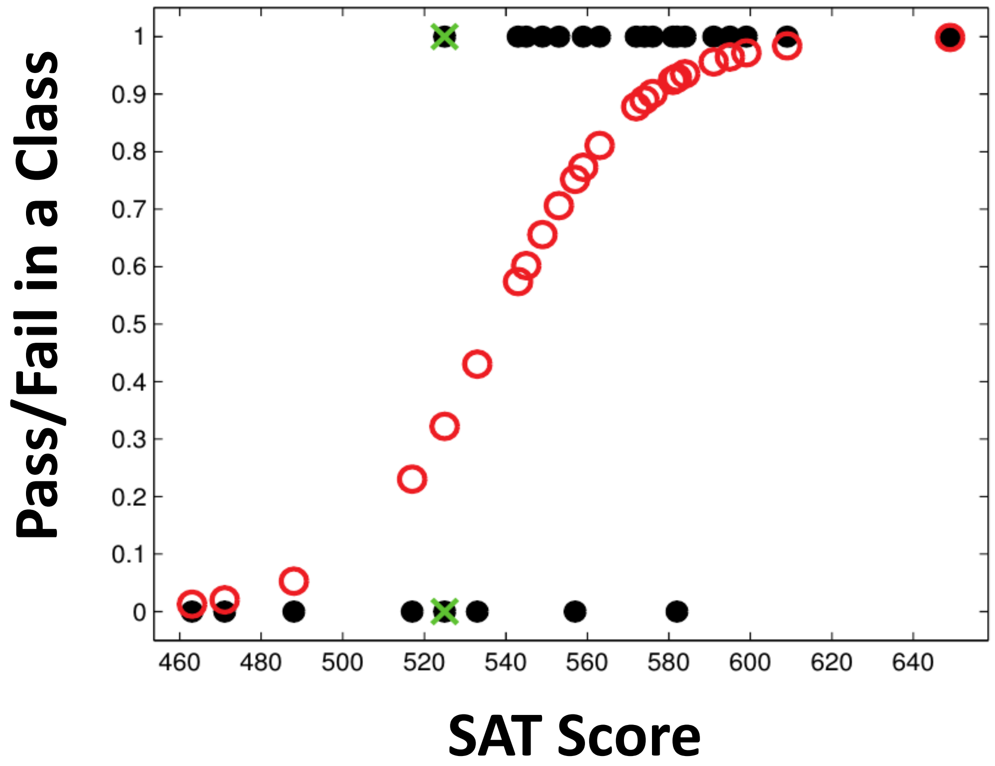


Example

- Black dots are training data. Red circles plot $p(y=1/\mathbf{x}_i,\mathbf{w^*})$.
- For example, We can induce a decision rule as

$$\hat{y}(x) = 1 \iff p(y=1|\mathbf{x}) > 0.5$$

- Decision boundary?
- It is not linearly separable



MLE through Minimizing NLL

- With defining the mean of Bernoulli RV of y or $p(y_i=1)$ as μ_i, NLL can be determined as

$$NLL(w) = -\sum_{i=1}^{N} \log p(yi=class \mid \mathbf{x}_i, \mathbf{w})$$

$$= -\sum_{j} \log p(y_j=1 \mid \mathbf{x}_j, \mathbf{w}) - \sum_{k} \log p(y_k=0 \mid \mathbf{x}_k, \mathbf{w})$$

j : indices s.t. y = 1 in D

k : indices s.t. y = 0 in D

$$= -\sum_{i=1}^{N} \log [p(yi=1 \mid \mathbf{x}_i, \mathbf{w})^{\mathbb{I}(yi=1)} p(yi=0 \mid \mathbf{x}_i, \mathbf{w})^{\mathbb{I}(yi=0)}]$$

$\mu = sigm(w^T x)$

$$= -\sum_{i=1}^{N} \log [\boldsymbol{\mu}_i^{\mathbb{I}(yi=1)} (1-\boldsymbol{\mu}_i)^{\mathbb{I}(yi=0)}] = -\sum_{i=1}^{N} y_i \log \boldsymbol{\mu}_i + (1-y_i)\log(1-\boldsymbol{\mu}_i)$$

- Unlike linear regression, we can no longer write down the MLE in closed form. Instead, we need gradient descent algorithm to compute it by repeatedly performing

$$w_{next} = w_{present} - \eta \nabla NLL(w)$$

Derivative of Sigmoid

- sigm(x) is

$$\text{sigm}(x) = \frac{1}{1+exp(-x)}$$

- Then,

$$\frac{\partial \text{sigm}(x)}{\partial x} = \frac{-exp(-x)}{\{1+exp(-x)\}^2} = \text{sigm}(x)\cdot\{1-\text{sigm}(x)\}$$

- Then how about the derivative of sigm(kx)?

- How about gradient of sigm($\mathbf{w}^T\mathbf{x}$) about **w**, $\nabla_w \text{sigm}(\mathbf{w}^T\mathbf{x})$?

$$\frac{\partial \text{sigm}(\mathbf{w}^T\mathbf{x})}{\partial \boldsymbol{w}} = \text{sigm}(\mathbf{w}^T\mathbf{x})\cdot\{1-\text{sigm}(\mathbf{w}^T\mathbf{x})\}\ \mathbf{x}$$

Gradient of NLL

- Can you derive $\nabla NLL(\boldsymbol{w})$?

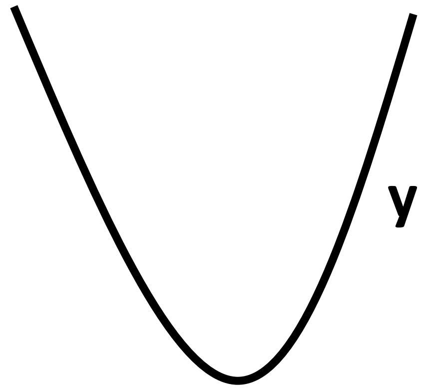
$$\nabla\{-\sum_{i=1}^{N} y_i \log \boldsymbol{\mu}_i + (1-y_i)\log(1-\boldsymbol{\mu}_i)\}$$

$$= \nabla\{-\sum_{i=1}^{N} y_i \log sigm(\boldsymbol{w}^T \boldsymbol{x}_i) + (1-y_i)\log(1 - sigm(\boldsymbol{w}^T \boldsymbol{x}_i))\}$$

$$= \mathbf{X}^T(\boldsymbol{\mu}-\mathbf{y})$$

- Derivation: Let's focus on the inside Σ:

Convexity



- Given $y = f(x)$, $f''(x) > 0$ ➔ Convex
- How about multiple variable case? Like,

$$y = f(x_1, x_2) = x_1^2 + 2x_2^2$$

- How can we guarantee that the $f(x_1,x_2)$ is convex?
  ➔ Using Hessian Matrix

Convexity of NLL in Logistic Regression

- Gradient and Hessian of NLL:

$$\begin{aligned}
\mathbf{g} &= \frac{d}{d\mathbf{w}} f(\mathbf{w}) = \sum_i (\mu_i - y_i)\mathbf{x}_i = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}) \\
\mathbf{H} &= \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^T = \sum_i (\nabla_{\mathbf{w}} \mu_i)\mathbf{x}_i^T = \sum_i \mu_i(1-\mu_i)\mathbf{x}_i\mathbf{x}_i^T \\
&= \mathbf{X}^T\mathbf{S}\mathbf{X}
\end{aligned}$$

- The components of Hessian are always larger than 0, (positive definite), which means that the convexity of NLL is guaranteed.

Now, We Are Ready For

- Training!

$$w_{\text{next}} = w_{\text{present}} - \eta \nabla NLL(w)$$

- Gradient of NLL is $= \mathbf{X}^T(\boldsymbol{\mu}-\mathbf{y})$

Summary

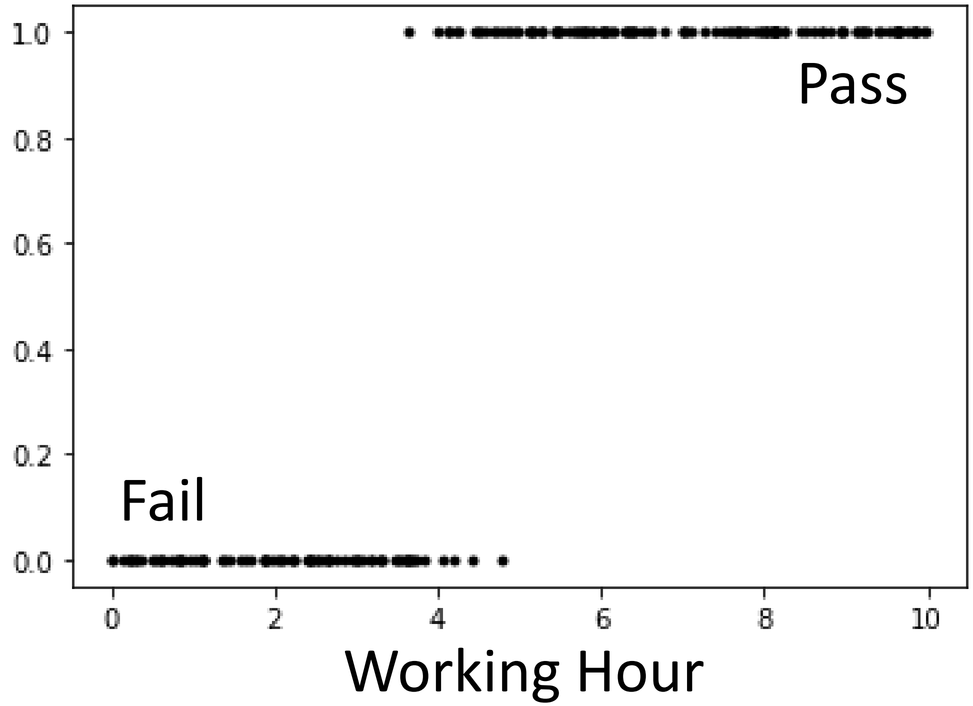
- MLE vs. cost function
- NLL as the general cost function
- Probability model for logistic regression
- Given probability model, NLL derivation
- With derived NLL, you can find w*!

Gradient Descent for NLL in Logistic Regression

- Python coding for logistic regression
- Do simple practice for the following data:
- https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/testData_LogisticRegression.txt

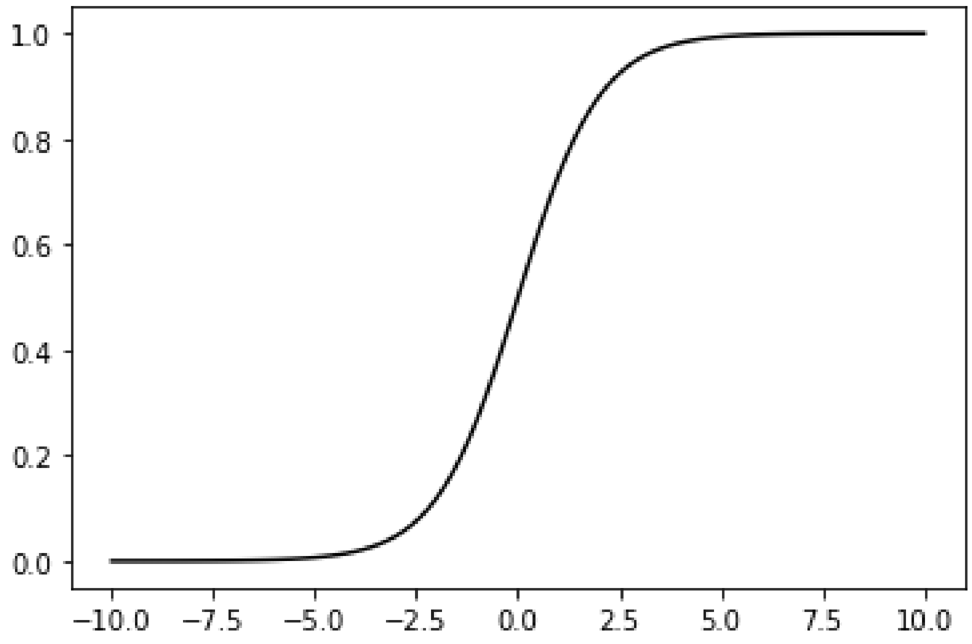
You Can Plot the Data

```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt

dfLoad= pd.read_csv('File path in the previous slide', sep="\s+")
xxRaw = np.array(dfLoad.values[:,0])
yyRaw = np.array(dfLoad.values[:,1])
plt.plot(xxRaw, yyRaw, "k.")
```



Defining Sigmoid & Plot

```
def sigmoid(x):
    return 1.0/(1+np.exp(-x))

xxTest = np.linspace(-10, 10, num=101)
plt.plot(xxTest, sigmoid(xxTest), "k-")
```



Implementing MLE w/ Gradient Descent

- Recall that the gradient of NLL is

$$\mathbf{X}^T(\boldsymbol{\mu}-\mathbf{y})$$

- We will declare design matrix:

```
N = len(xxRaw)
x_bias = np.c_[np.ones([N,1]), xxRaw].T #Padding ones for x0
y = yyRaw.reshape(N,1)
X = x_bias.T
```

- Note that μ is derived by sigm($\mathbf{w}^T\mathbf{x}$):

```
eta = 0.1   #learning rate
n_iterations = 1000
wGD = np.zeros([2,1])    #initialized to 0
wGDbuffer = np.zeros([2,n_iterations+1])

for iteration in range(n_iterations):
    mu = sigmoid(wGD.T.dot(x_bias)).T
    gradients= X.T.dot(mu-y)
    #gradients = - xHeight_bias.dot(wGD)
    #gradients = - (2/N)*(xx_bias.T.dot(yy-xx_bias.dot(wGD)))
    #gradients = - (2/N)*(xHeight_bias.T.dot(yWeight-xHeight_bias.dot(wGD)))
    wGD = wGD - eta*gradients
    wGDbuffer[:,iteration+1] = [wGD[0], wGD[1]]
```

Result Check

```
xxTest = np.linspace(0, 10, num=N).reshape(N,1)
xxTest_bias = np.c_[np.ones([N,1]), xxTest]
aaa = sigmoid(wGD.T.dot(xxTest_bias.T))
#plt.plot(aaa)
plt.plot(xxTest, sigmoid(wGD.T.dot(xxTest_bias.T)).T, "r-.")
```