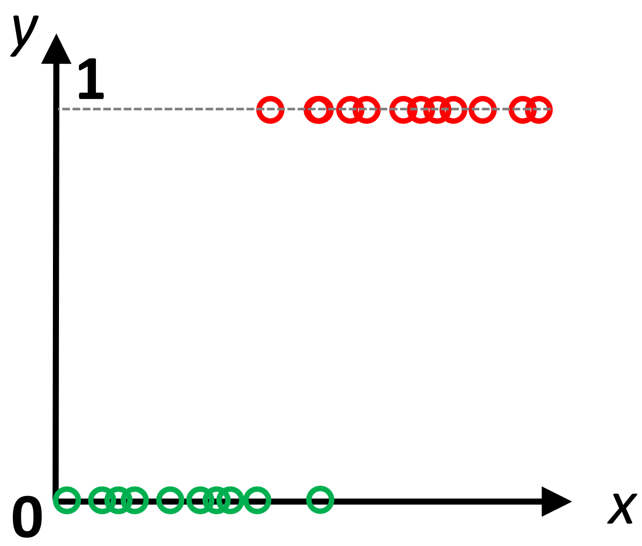
What We Practiced is the Simplest Case

1) One dimensional feature
2) Binary classification



Logistic Regression for Higher Dimension / Multi-Class

Hanwool Jeong

hwjeong@kw.ac.kr

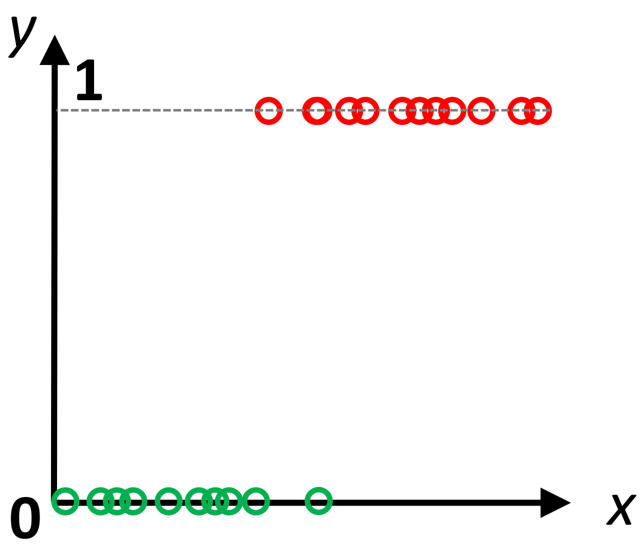
Revisit p(D/w)

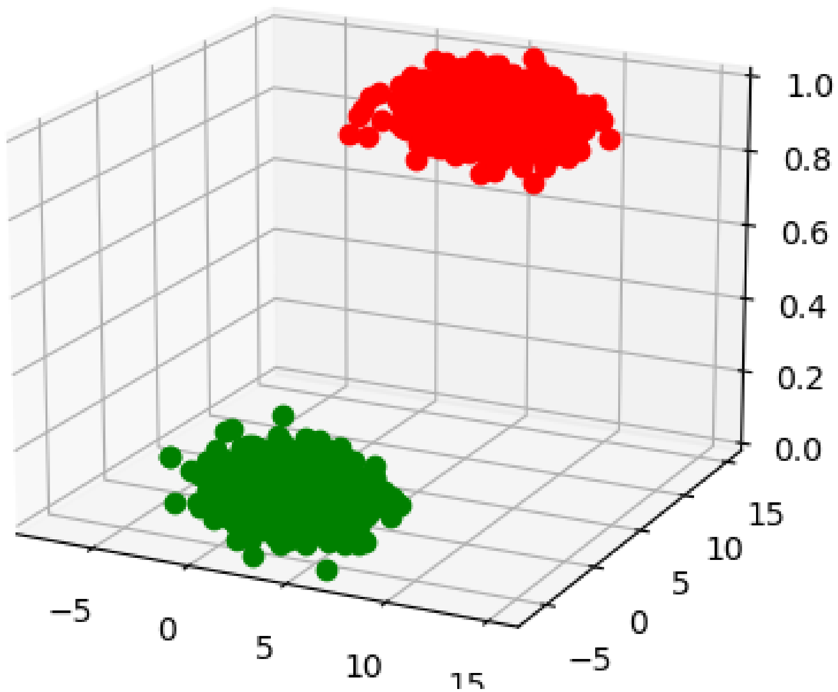
- $p(y|\mathbf{x}, \mathbf{w}) = Ber(y|sigm(\mathbf{w}^T\mathbf{x}))$

$p(y=1|x, w) = sigm(\mathbf{w}^T\mathbf{x})$

$p(y=0|x, w) = 1 - sigm(\mathbf{w}^T\mathbf{x})$

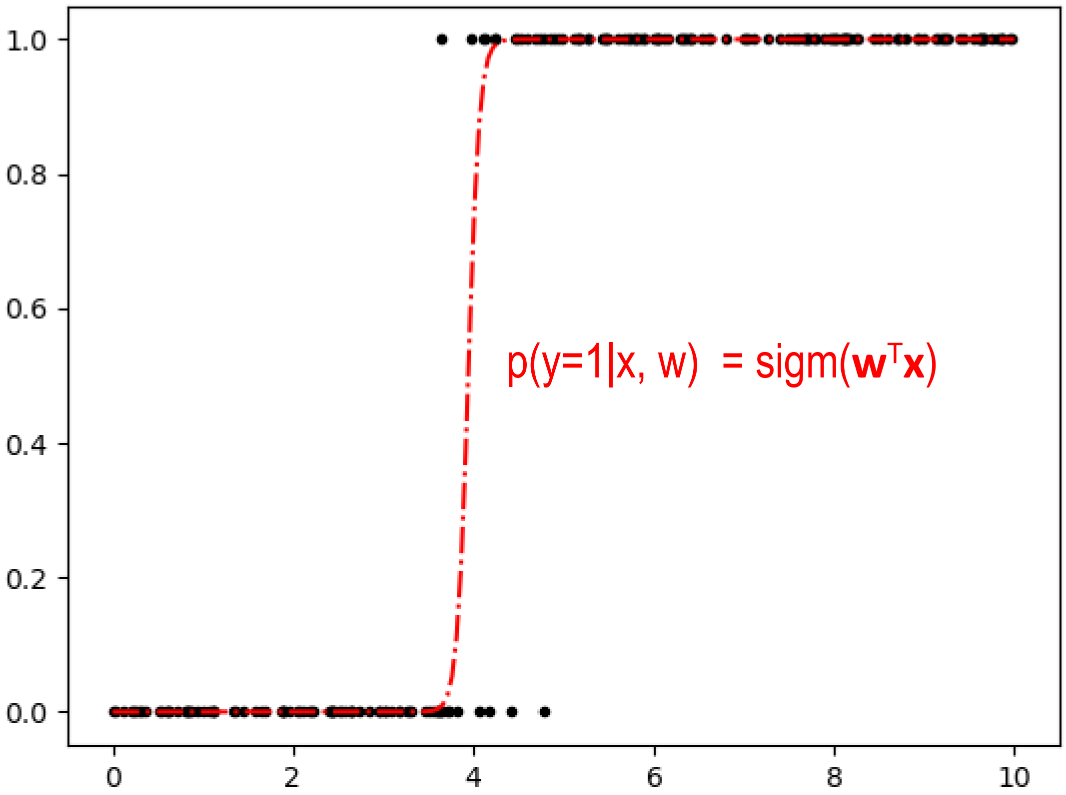
➔ We can expand it to higher dimension feature case





In 1D Feature,

- How about 2D?



Revisit; 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

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

$$= -\sum_{j} \log p(yj=1|\mathbf{x}_j, \mathbf{w}) - \sum_{k} \log p(yk=0|\mathbf{x}_k, \mathbf{w})$$

j : indices s.t. y = 1 in D
k : indices s.t. y = 0 in D

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

$\mu = \mathrm{sigm}(w^T x)$

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

- 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

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

Revisit Gradient of NLL

- You remember $\nabla NLL(\boldsymbol{w})$?

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

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

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

Importance of Visualization

- **To understand the concept, you should visualize it!**
- Strategy
  1) Generate 2D binary classification dataset (using np.random module)
  2) Visualize the 2D dataset **for your understanding**
  3) Finding logistic regression model that fits the data
  4) Visualize the logistic regression model for your understanding

Generating 2D Dataset

- Generating random 1D data following Gaussian distribution

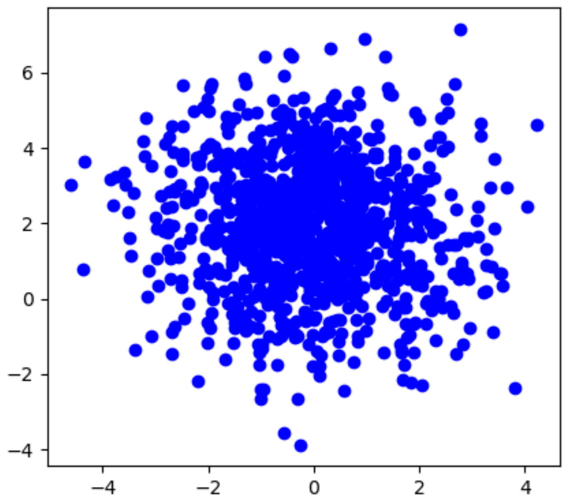
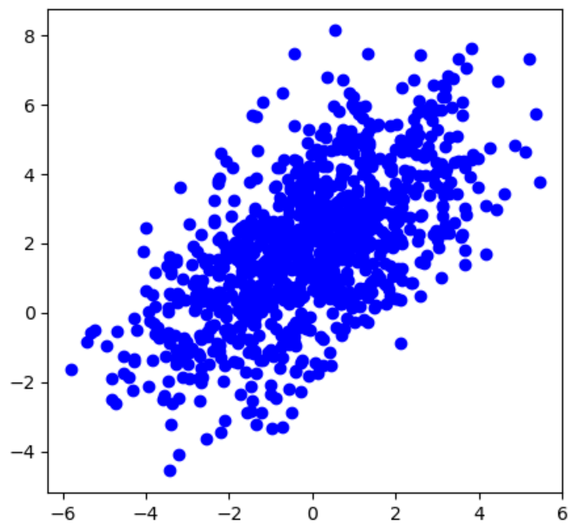
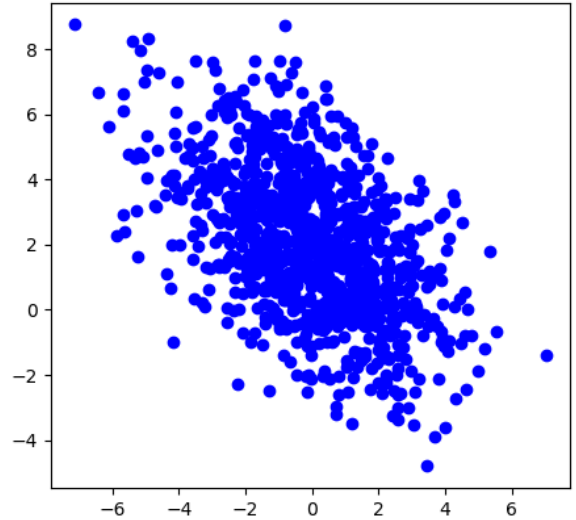
```
Ndata=1000
A = np.random.randn(Ndata)
f1 = plt.figure()
ax1= plt.axes()
ax1.hist(A, bins=10)
```

- Expand it to 2D Data

```
f2 = plt.figure()
ax2= plt.axes()
d1 = np.random.multivariate_normal(mean=[0,2], cov=[[2,-5],[-5,3]], size=Ndata)
d2 = np.random.multivariate_normal(mean=[8,6], cov=[[5,-3],[-3,8]], size=Ndata)
plt.scatter(d1[:,0], d1[:,1], c="b")
plt.scatter(d2[:,0], d2[:,1], c="r")
```

Covariance?

- 2D multivariate Gaussian data

Independent	Positive Correlated	Negative Correlated
		

Covariance Matrix for 2D Data

- Covariance matrix Cov(A)

$$\mathrm{Cov}(A) = \begin{bmatrix} \dfrac{\sum (x_i - \bar{X})(x_i - \bar{X})}{N} & \dfrac{\sum (x_i - \bar{X})(y_i - \bar{Y})}{N} \\ \dfrac{\sum (x_i - \bar{X})(y_i - \bar{Y})}{N} & \dfrac{\sum (y_i - \bar{Y})(y_i - \bar{Y})}{N} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{Cov}(X, X) & \mathrm{Cov}(Y, X) \\ \mathrm{Cov}(X, Y) & \mathrm{Cov}(Y, Y) \end{bmatrix}$$

- Correlation between X and Y

$$\mathrm{Corr}(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sigma_x \sigma_y}$$

Covarianced normalized by Standard Deviation

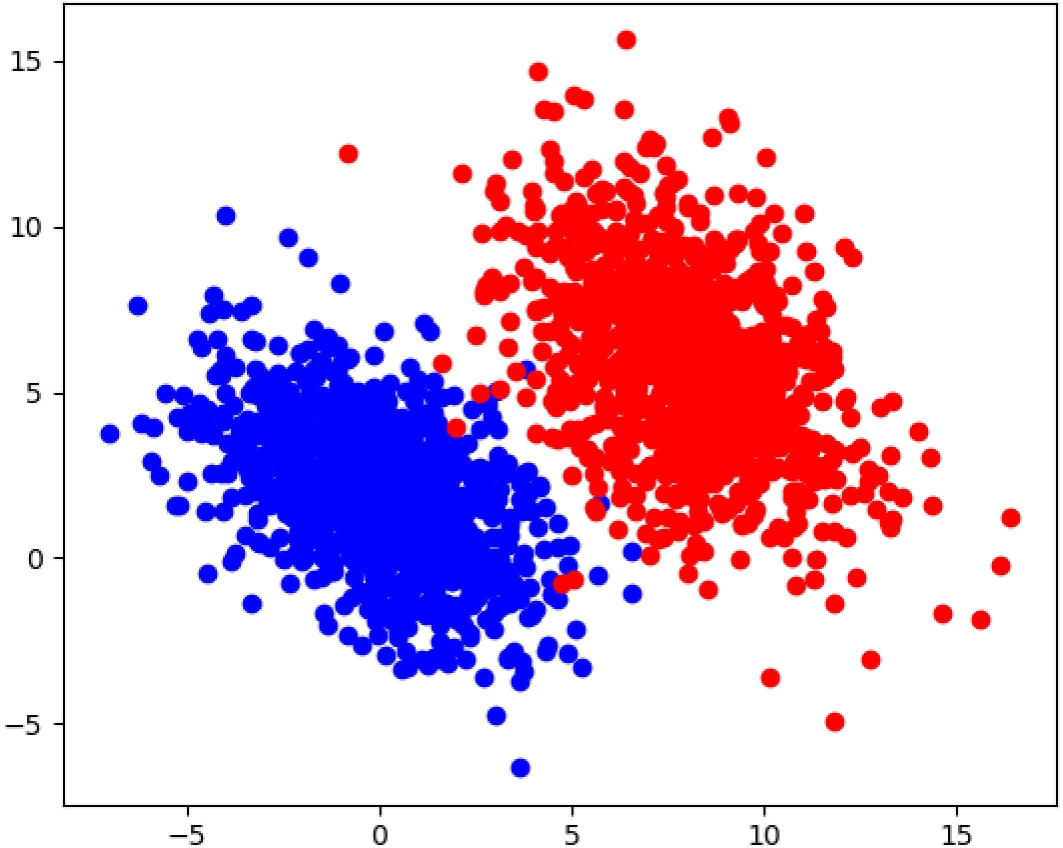
Correlation between X and Y

Standard deviation of X

Standard deviation of Y

Revisit 2D Data Generating

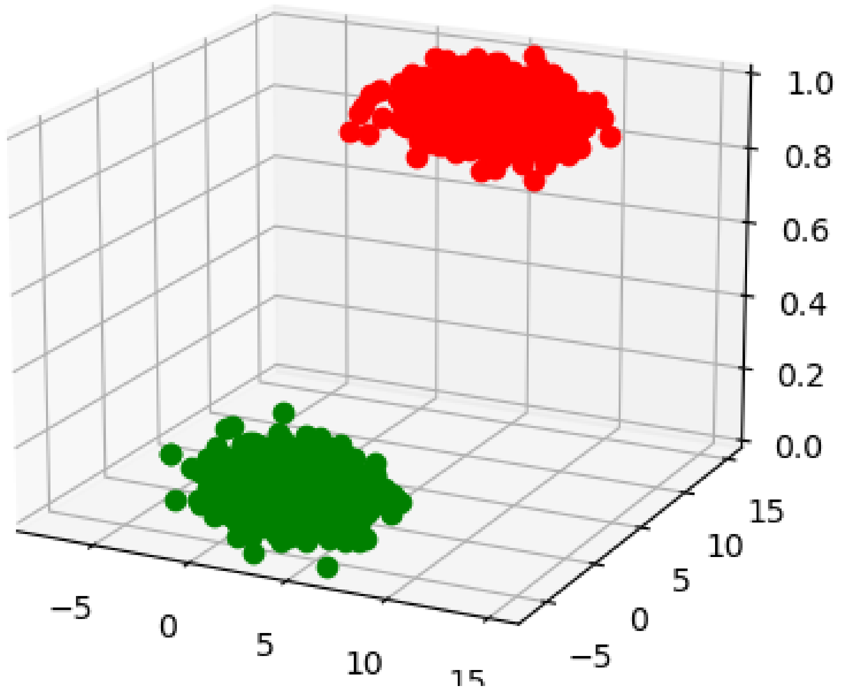
```
f2 = plt.figure()
ax2= plt.axes()
d1 = np.random.multivariate_normal(mean=[0,2], cov=[[2,-5],[-5,3]], size=Ndata)
d2 = np.random.multivariate_normal(mean=[8,6], cov=[[5,-3],[-3,8]], size=Ndata)
plt.scatter(d1[:,0], d1[:,1], c="b")
plt.scatter(d2[:,0], d2[:,1], c="r")
```



3D Plot

- You can add projection='3d' as plt.axes argument as

```
f3 = plt.figure()
ax3= plt.axes(projection ='3d')
ax3.plot(d1[:,0], d1[:,1], 0, 'go')
ax3.plot(d2[:,0], d2[:,1], 1, 'ro')
```



Defining X and y

```
N = Ndata
X1 = np.c_[np.ones([N,1]), d1]
X2 = np.c_[np.ones([N,1]), d2]
X = np.r_[X1, X2]
y1 = np.zeros([N,1])
y2 = np.ones([N,1])
y = np.r_[y1, y2]
```

Training for
Reusing 1D Logistic Regression Code

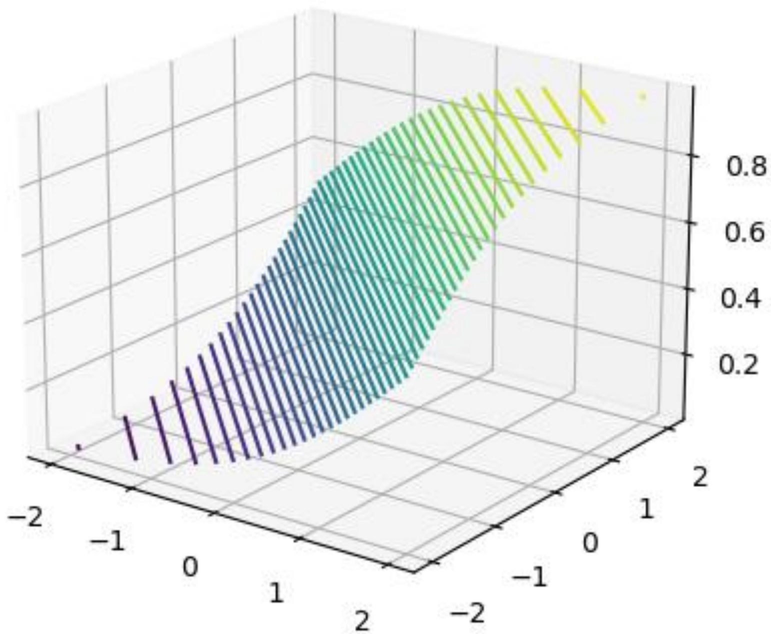
```python
eta = 0.1
n_iterations = 100
wGD = np.zeros([2,1])
wGDbuffer = np.zeros([2,n_iterations+1])

for iteration in range(n_iterations):
    mu = sigmoid(wGD.T.dot(x_bias)).T
    gradients = X.T.dot(mu-y)
    wGD = wGD - eta*gradients
    wGDbuffer[:,iteration+1] = [wGD[0], wGD[1]]
```

```python
eta = 0.1
n_iterations = 100
wGD = np.zeros([3,1]) # 2 --> 3
wGDbuffer = np.zeros([3,n_iterations+1])

for iteration in range(n_iterations):
    mu = sigmoid(wGD.T.dot(X.T)).T
    gradients = X.T.dot(mu-y)
    wGD = wGD - eta*gradients
    wGDbuffer[:,iteration+1] = [wGD[0], wGD[1], wGD[2]] #3 dim
```

Contour for Function 3D Plot

- Find the following codes:

```
x1sig = np.linspace(-2, 2, 100)
x2sig = np.linspace(-2, 2, 100)
[x1Sig, x2Sig] = np.meshgrid(x1sig, x2sig)
ySig = sigmoid(x1Sig+x2Sig)

f5 = plt.figure()
ax5= plt.axes(projection ='3d')
ax5.contour3D(x1Sig, x2Sig, ySig, 50)
```



Plotting Trained Model

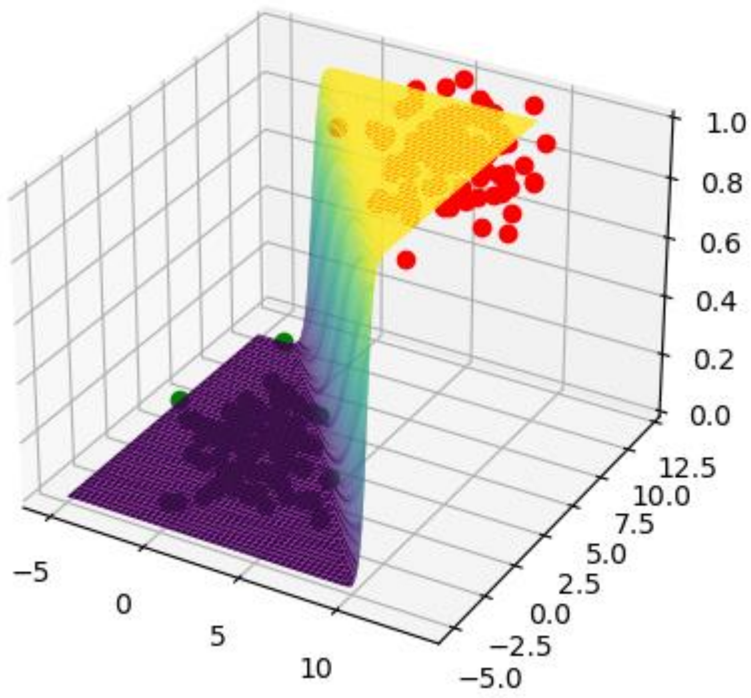
- Nothing but p(y|**x**, **w**) = Ber(y|sigm($\mathbf{w}^T\mathbf{x}$))

```
x1sig = np.linspace(-5, 10, 100)
x2sig = np.linspace(-5, 10, 100)
[x1Sig, x2Sig] = np.meshgrid(x1sig, x2sig)
ySig = sigmoid( wGD[1]*x1Sig+ wGD[2]*x2Sig + wGD[0])
ax3.contour3D(x1Sig, x2Sig, ySig, 50)
```

- Or you can use other methods for 3D plot:

```
#ax3.contour3D(x1Sig, x2Sig, ySig, 50)
ax3.plot_surface(x1Sig, x2Sig, ySig, cmap="viridis")
```

Resultant Plot

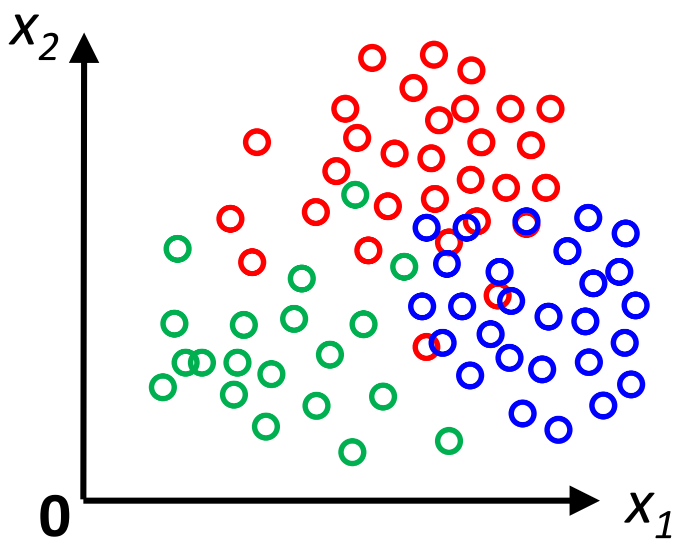


Using Trained Model?

```
x1sig = np.linspace(-5, 10, 100)
x2sig = np.linspace(-5, 10, 100)
[x1Sig, x2Sig] = np.meshgrid(x1sig, x2sig)
ySig = sigmoid( wGD[1]*x1Sig+ wGD[2]*x2Sig + wGD[0])
ax3.contour3D(x1Sig, x2Sig, ySig, 50)
```

How Can We Implement Multi-Class?

- Remind that, so far, what we focus is the probability that the certain sample is classified as "1".
- That is, we can use the strategy **one vs. rest**



One vs. Rest