Classification

Industrial AI Lab.

Prof. Seungchul Lee

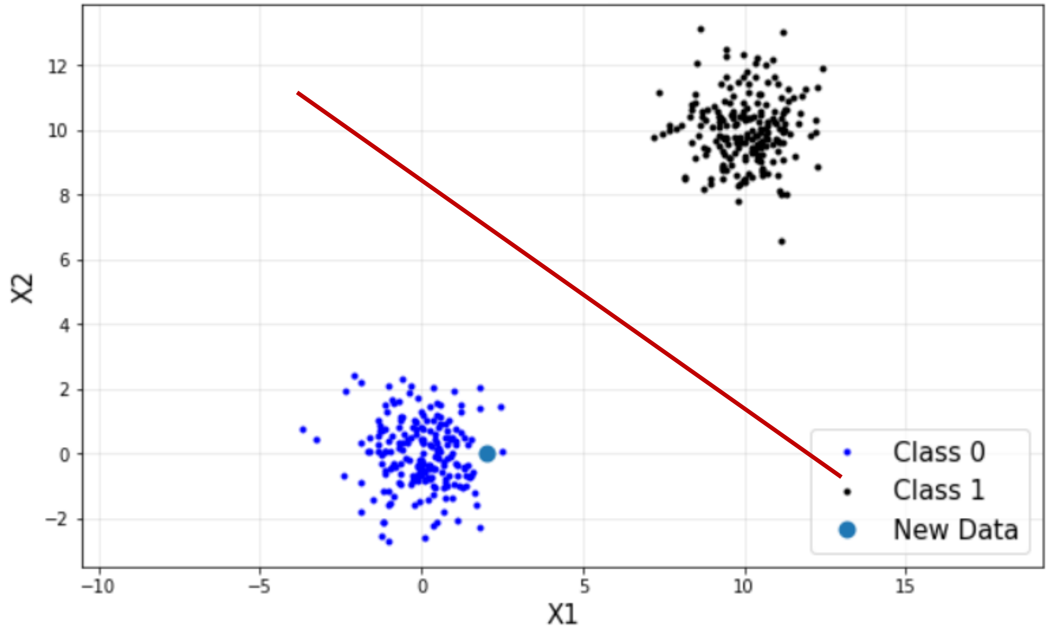
Yunseob Hwang, Juwon Na

Classification

- We will learn
  - Perceptron
  - Logistic regression

- To find

  a classification boundary

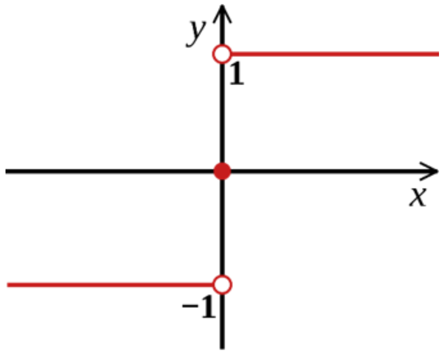


Perceptron

Perceptron

- For input $x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$ 'attributes of a customer'
- Weights $\omega = \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_d \end{bmatrix}$

$$\text{Approve credit if } \sum_{i=1}^{d} \omega_i x_i > \text{threshold},$$
$$\text{Deny credit if } \sum_{i=1}^{d} \omega_i x_i < \text{threshold}.$$



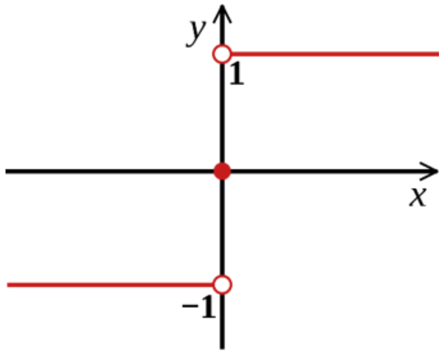
$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) - \text{threshold}\right) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) + \omega_0\right)$$

Perceptron

$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) - \text{threshold}\right) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) + \omega_0\right)$$

- Introduce an artificial coordinate $x_0 = 1$:

$$h(x) = \text{sign}\left(\sum_{i=0}^{d} \omega_i x_i\right)$$

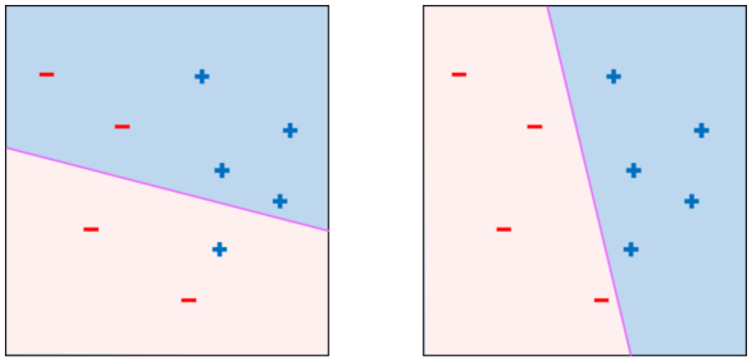


- In a vector form, the perceptron implements

$$h(x) = \text{sign}\left(\omega^T x\right)$$

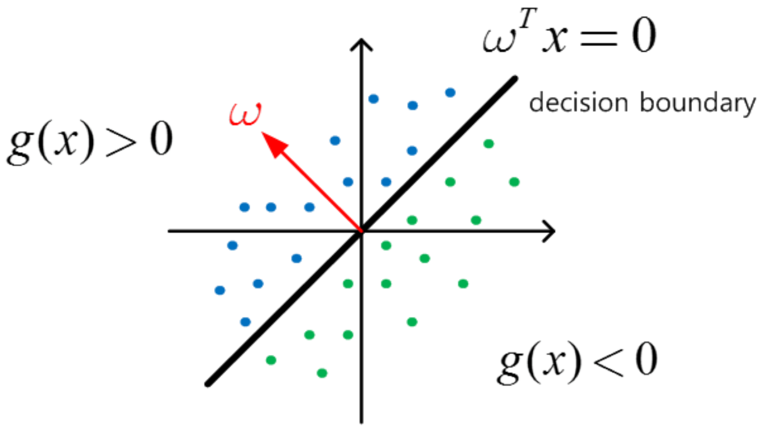
Perceptron

- Works for linearly separable data



Linearly separable data

- Hyperplane
  - Separates a D-dimensional space into two half-spaces
  - Defined by an outward pointing normal vector
  - ω is orthogonal to any vector lying on the hyperplane
  - Assume the hyperplane passes through origin, $\omega^T x = 0$ with $x_0 = 1$

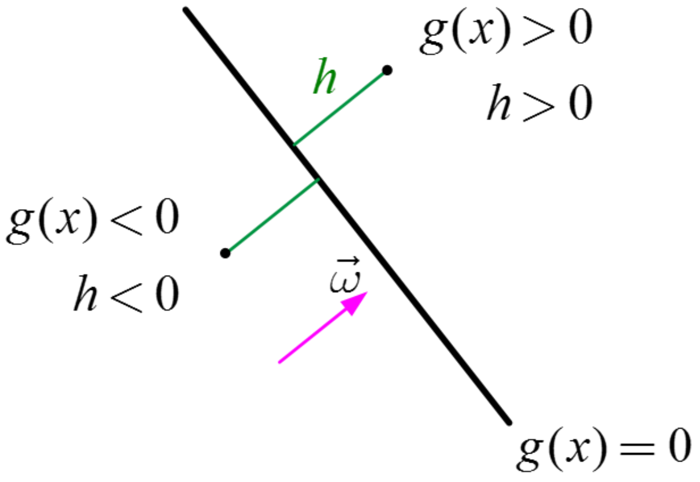


Sign

- Sign with respect to a line

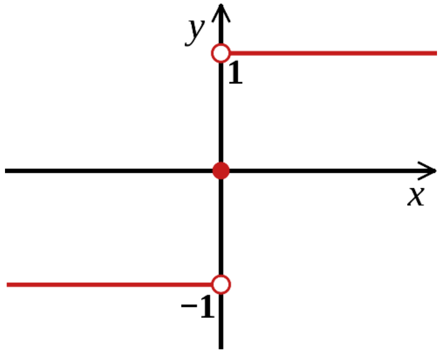
$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_1 x_1 + \omega_2 x_2 + \omega_0 = \omega^T x + \omega_0$$

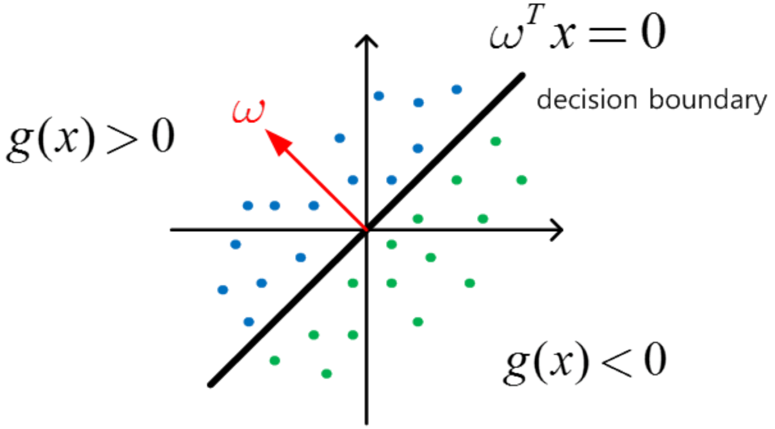
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = \omega^T x$$



How to Find ω

- All data in class 1 ($y = 1$)
  - $g(x) > 0$
- All data in class 0 ($y = -1$)
  - $g(x) < 0$





Perceptron Algorithm

- The perceptron implements

$$h(x) = \text{sign}\left(\omega^T x\right)$$

- Given the training set

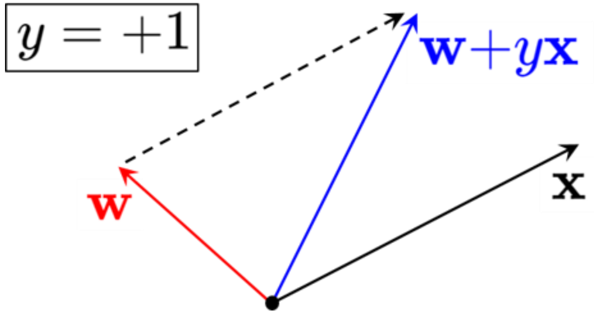
$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N) \quad \text{where } y_i \in \{-1, 1\}$$

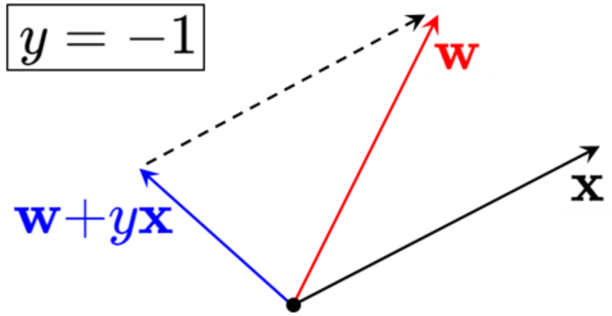
1) pick a misclassified point

$$\text{sign}\left(\omega^T x_n\right) \neq y_n$$

2) and update the weight vector

$$\omega \leftarrow \omega + y_n x_n$$





Perceptron Algorithm

- Why perceptron updates work ?

- Let's look at a misclassified positive example $(y_n = +1)$
  - Perceptron (wrongly) thinks $\omega_{old}^T x_n < 0$

  - Updates would be

$$\omega_{new} = \omega_{old} + y_n x_n = \omega_{old} + x_n$$

$$\omega_{new}^T x_n = (\omega_{old} + x_n)^T x_n = \omega_{old}^T x_n + x_n^T x_n$$

  - Thus $\omega_{new}^T x_n$ is less negative than $\omega_{old}^T x_n$

Iterations of Perceptron

1. Randomly assign ω

2. One iteration of the PLA (perceptron learning algorithm)

$$\omega \leftarrow \omega + yx$$

   where (x, y) is a misclassified training point

3. At iteration $i = 1, 2, 3, \cdots$, pick a misclassified point from

$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$$

4. And run a PLA iteration on it

5. That's it!

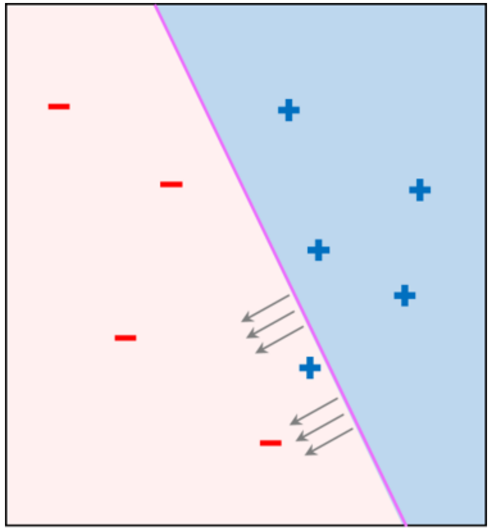
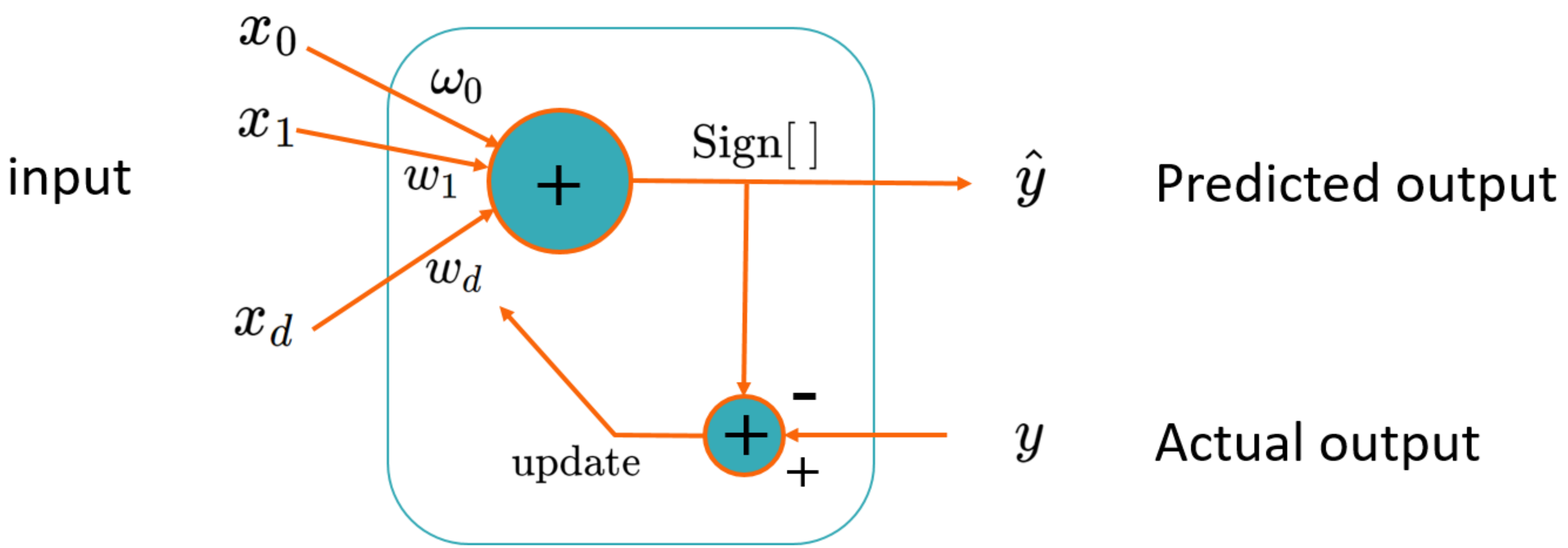


Diagram of Perceptron



Perceptron Loss Function

$$L(\omega) = \sum_{n=1}^{m} \max\left\{0, -y_n \cdot \left(\omega^T x_n\right)\right\}$$

- Loss = 0 on examples where perceptron is correct,
  i.e., $y_n \cdot (\omega^T x_n) > 0$

- Loss > 0 on examples where perceptron is misclassified,
  i.e., $y_n \cdot (\omega^T x_n) < 0$

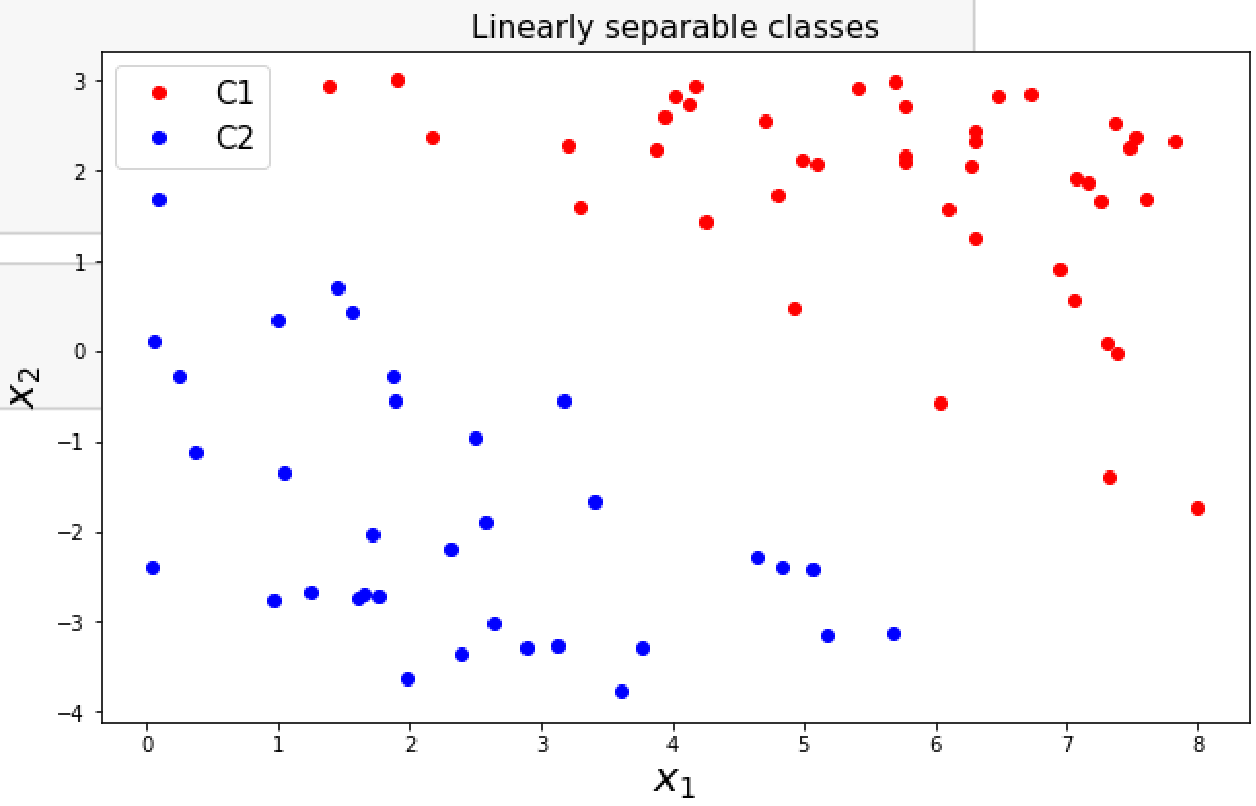
- Note:
  – $\mathrm{sign}(\omega^T x_n) \neq y_n$ is equivalent to $y_n \cdot (\omega^T x_n) < 0$

Perceptron Algorithm in Python

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#training data gerneration
m = 100
x1 = 8*np.random.rand(m, 1)
x2 = 7*np.random.rand(m, 1) - 4

g = 0.8*x1 + x2 - 3
```

```
C1 = np.where(g >= 1)
C2 = np.where(g < -1)
print(C1)
```



Perceptron Algorithm in Python

- Unknown parameters ω

$$g(x) = \omega_0 + \omega^T x = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$$

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix} \qquad x = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

```python
X1 = np.hstack([np.ones([C1.shape[0],1]), x1[C1], x2[C1]])
X2 = np.hstack([np.ones([C2.shape[0],1]), x1[C2], x2[C2]])
X = np.vstack([X1, X2])

y = np.vstack([np.ones([C1.shape[0],1]), -np.ones([C2.shape[0],1])])

X = np.asmatrix(X)
y = np.asmatrix(y)
```

Perceptron Algorithm in Python

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

$\omega \leftarrow \omega + yx$ where (x, y) is a misclassified training point

```
w = np.ones([3,1])
w = np.asmatrix(w)

n_iter = y.shape[0]
for k in range(n_iter):
    for i in range(n_iter):
        if y[i,0] != np.sign(X[i,:]*w)[0,0]:
            w += y[i,0]*X[i,:].T

print(w)
```

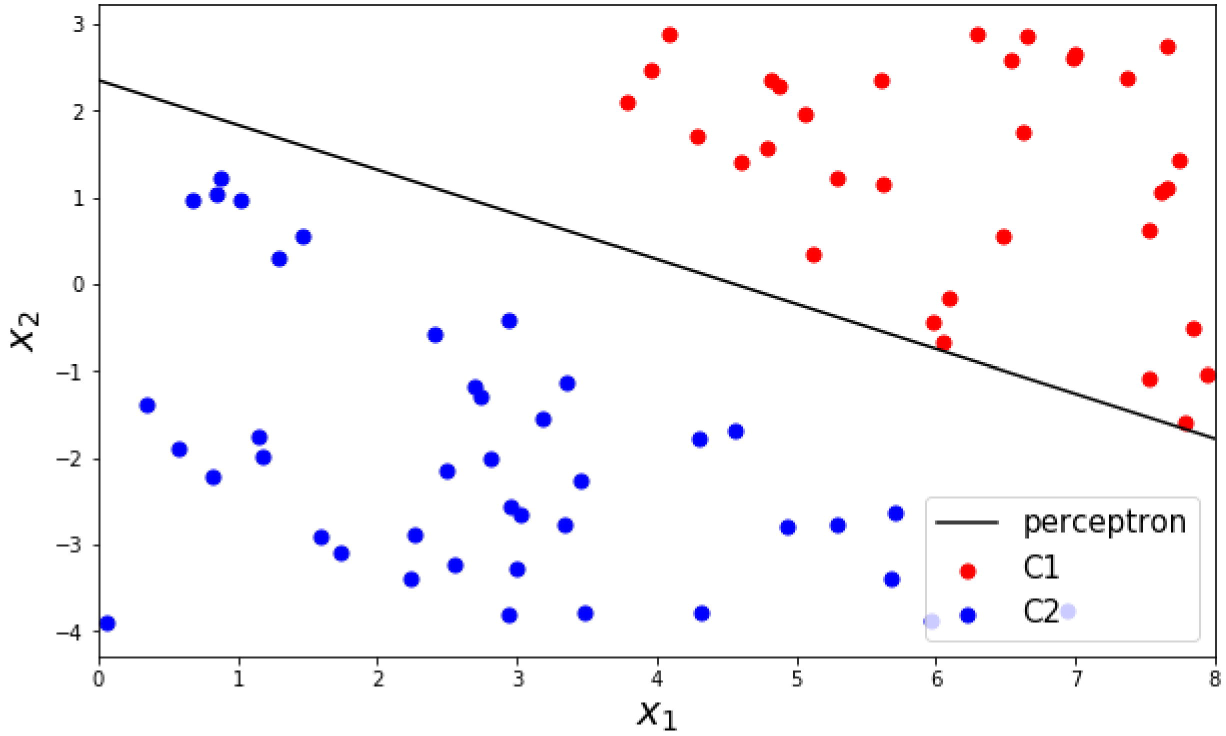
Perceptron Algorithm in Python

$$g(x) = \omega_0 + \omega^T x = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$$

$$\Longrightarrow \quad x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\omega_0}{\omega_2}$$

```python
x1p = np.linspace(0,8,100).reshape(-1,1)
x2p = - w[1,0]/w[2,0]*x1p - w[0,0]/w[2,0]

plt.figure(figsize=(10, 6))
plt.scatter(x1[C1], x2[C1], c='r', s=50, label='C1')
plt.scatter(x1[C2], x2[C2], c='b', s=50, label='C2')
plt.plot(x1p, x2p, c='k', label='perceptron')
plt.xlim([0,8])
plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.legend(loc = 1, fontsize = 15)
plt.show()
```

Perceptron Algorithm in Python



Perceptron using Scikit-Learn

```
X1 = np.hstack([x1[C1], x2[C1]])
X2 = np.hstack([x1[C2], x2[C2]])
X = np.vstack([X1, X2])

y = np.vstack([np.ones([C1.shape[0],1]), -np.ones([C2.shape[0],1])])
```

```
from sklearn import linear_model

clf = linear_model.Perceptron(tol=1e-3)
clf.fit(X, np.ravel(y))
```

```
clf.predict([[3, -2]])
```

```
array([-1.])
```

$$x = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix}$$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

The Best Hyperplane Separator?

- Perceptron finds one of the many possible hyperplanes separating the data if one exists
- Of the many possible choices, which one is the best?

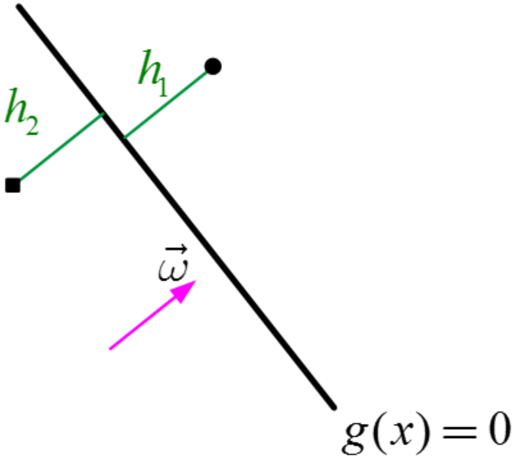
- Utilize distance information
- Intuitively we want the hyperplane having the maximum margin
- Large margin leads to good generalization on the test data
  - we will see this formally when we discuss Support Vector Machine (SVM)

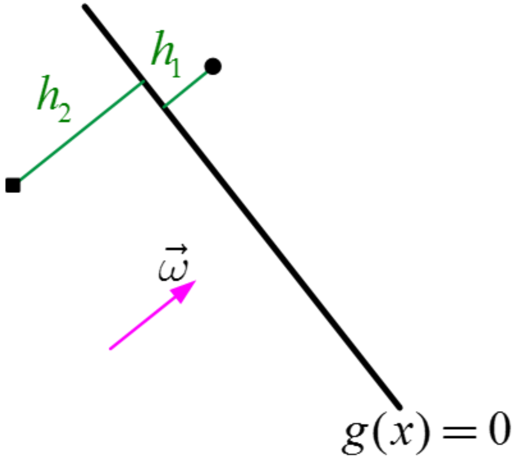
- Utilize distance information from all data samples
  - We will see this formally when we discuss the logistic regression

- Perceptron will be shown to be a basic unit for neural networks and deep learning later

Logistic Regression

Using Distances





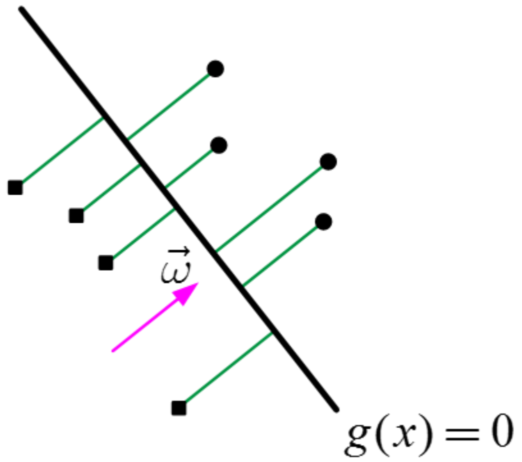
$$|h_1| + |h_2| \qquad\qquad |h_1| + |h_2|$$

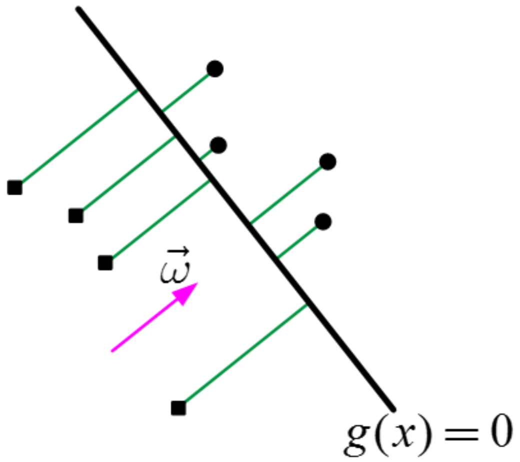
$$|h_1| \cdot |h_2| \qquad\qquad |h_1| \cdot |h_2|$$

$$\frac{|h_1| + |h_2|}{2} \geq \sqrt{|h_1| \cdot |h_2|} \qquad \text{equal iff} \quad |h_1| = |h_2|$$

Using all Distances

- basic idea: to find the decision boundary (hyperplane) of $g(x) = \omega^T x = 0$ such that maximizes $\prod_i |h_i|$ → optimization



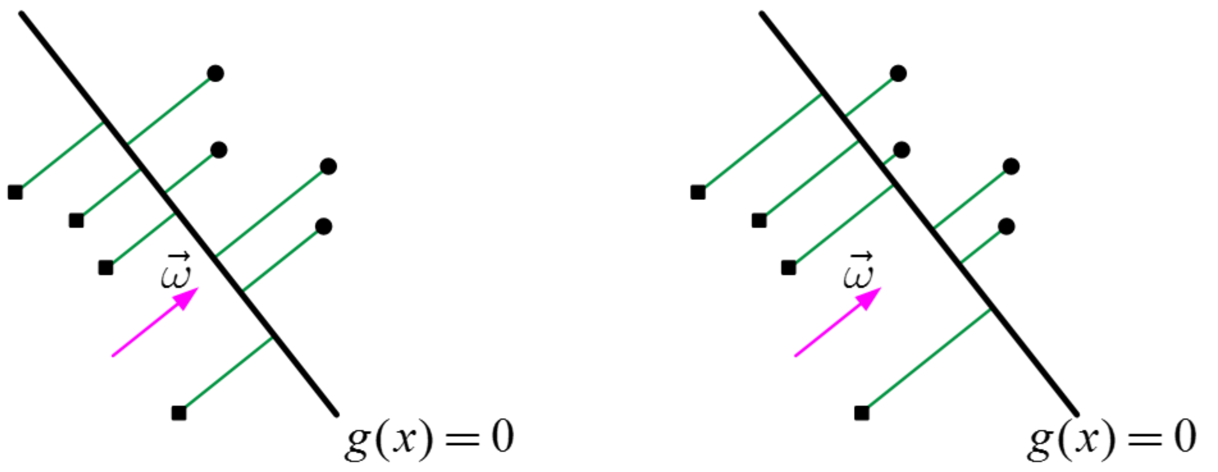


  – Inequality of arithmetic and geometric means

$$\frac{x_1 + x_2 + \cdots + x_m}{m} \geq \sqrt[m]{x_1 \cdot x_2 \ldots x_m}$$

  and that equality holds if and only if $x_1 = x_2 = \cdots = x_m$

Using all Distances

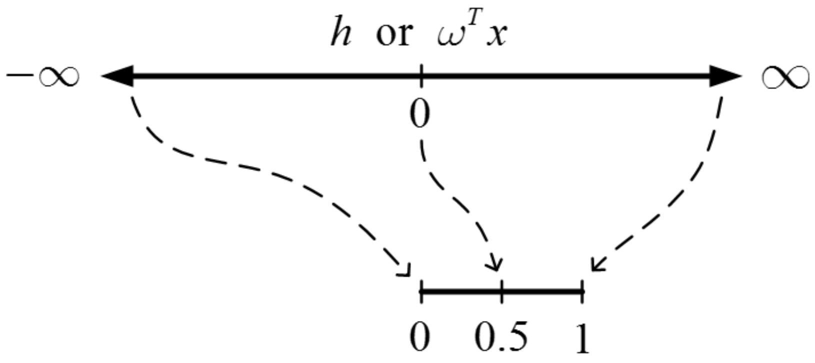


- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes

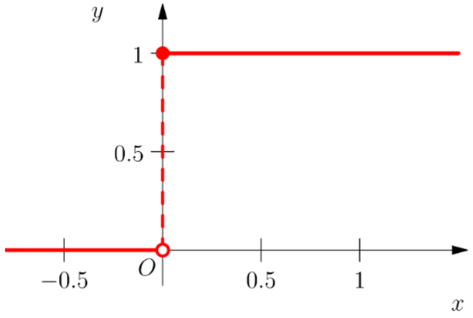
$$h = \frac{g(x)}{||\omega||} = \frac{\omega^T x}{||\omega||} \sim \omega^T x$$

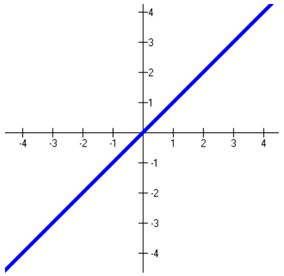
Sigmoid Function

- We link or squeeze $(-\infty, +\infty)$ to $(0, 1)$ for several reasons:

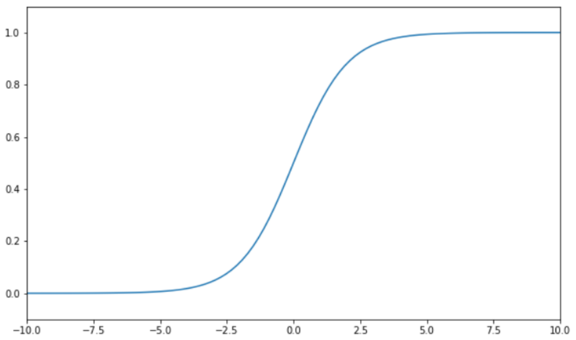


Step function





$$\sigma(z) = \frac{1}{1+e^{-z}} \implies \sigma\left(\omega^T x\right) = \frac{1}{1+e^{-\omega^T x}}$$



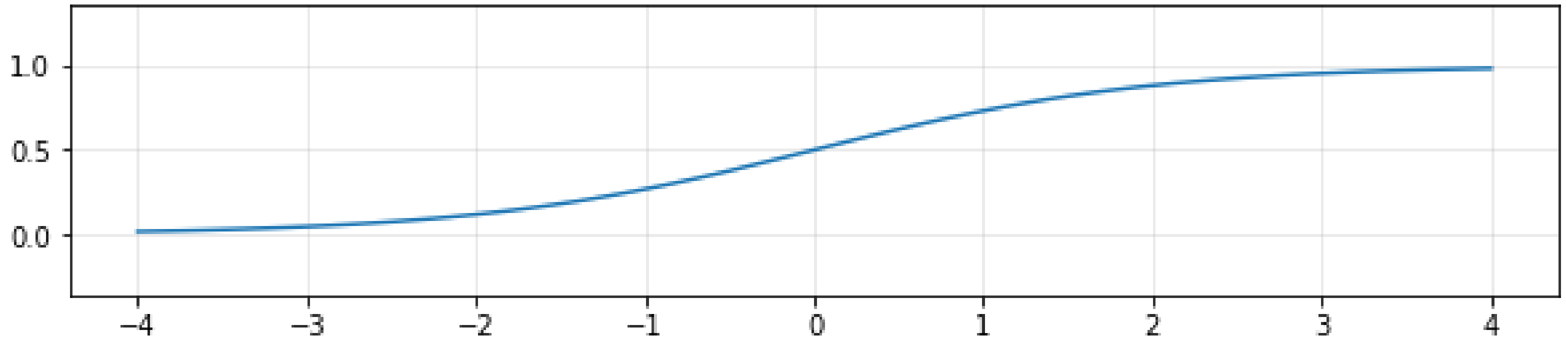
POSTECH

Sigmoid Function

- $\sigma(z)$ is the sigmoid function, or the logistic function
  - Logistic function always generates a value between 0 and 1
  - Crosses 0.5 at the origin, then flattens out

```
# plot a sigmoid function

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

z = np.linspace(-4,4,100)
s = 1/(1 + np.exp(-z))

plt.figure(figsize=(10,2))
plt.plot(z, s)
plt.xlim([-4, 4])
plt.axis('equal')
plt.grid(alpha = 0.3)
plt.show()
```

$$\sigma(z) = \frac{1}{1+e^{-z}} \implies \sigma(\omega^T x) = \frac{1}{1+e^{-\omega^T x}}$$



Sigmoid Function

- Benefit of mapping via the logistic function
  - Monotonic: same or similar optimization solution
  - Continuous and differentiable: good for gradient descent optimization
  - Probability or confidence: can be considered as probability

$$P(y=+1 \mid x, \omega) = \frac{1}{1+e^{-\omega^T x}} \quad \in \; [0,1]$$

  - Probability that the label is $+1$

$$P(y=+1 \mid x\,; \omega)$$

  - Probability that the label is 0

$$P(y=0 \mid x\,; \omega) = 1 - P(y=+1 \mid x\,; \omega)$$

Goal: We Need to Fit ω to Data

- For a single data point (x, y) with parameters ω

$$P(y = +1 \mid x\,; \omega) = h_\omega(x) = \sigma\left(\omega^T x\right)$$
$$P(y = 0 \mid x\,; \omega) = 1 - h_\omega(x) = 1 - \sigma\left(\omega^T x\right)$$

- It can be compactly written as

$$P(y \mid x\,; \omega) = (h_\omega(x))^y (1 - h_\omega(x))^{1-y}$$

- For m training data points, the likelihood function of the parameters:

$$\begin{aligned}\mathscr{L}(\omega) &= P\left(y^{(1)}, \cdots, y^{(m)} \mid x^{(1)}, \cdots, x^{(m)}\,; \omega\right) \\ &= \prod_{i=1}^{m} P\left(y^{(i)} \mid x^{(i)}\,; \omega\right) \\ &= \prod_{i=1}^{m} \left(h_\omega\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\omega\left(x^{(i)}\right)\right)^{1-y^{(i)}} \qquad \left(\sim \prod_i |h_i|\right)\end{aligned}$$

Goal: We Need to Fit ω to Data

$$
\begin{aligned}
\mathscr{L}(\omega) &= P\left(y^{(1)}, \cdots, y^{(m)} \mid x^{(1)}, \cdots, x^{(m)} ; \omega\right) \\
&= \prod_{i=1}^{m} P\left(y^{(i)} \mid x^{(i)} ; \omega\right) \\
&= \prod_{i=1}^{m} \left(h_\omega\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\omega\left(x^{(i)}\right)\right)^{1-y^{(i)}} \qquad \left(\sim \prod_i |h_i|\right)
\end{aligned}
$$

- It would be easier to work on the log likelihood.

$$
\ell(\omega) = \log \mathscr{L}(\omega) = \sum_{i=1}^{m} y^{(i)} \log h_\omega\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\omega\left(x^{(i)}\right)\right)
$$

- The logistic regression problem can be solved as a (convex) optimization problem:

$$
\hat{\omega} = \arg\max_{\omega} \ell(\omega)
$$

- Again, it is an optimization problem

Logistic Regression using GD

Gradient Descent for Logistic Regression

- To use the gradient descent method, we need to find the derivative of it

$$\nabla \ell(\omega) = \begin{bmatrix} \frac{\partial \ell(\omega)}{\partial \omega_1} \\ \vdots \\ \frac{\partial \ell(\omega)}{\partial \omega_n} \end{bmatrix}$$

- We need to compute $\dfrac{\partial \ell(\omega)}{\partial \omega_j}$

$$\ell(\omega) = \log \mathscr{L}(\omega) = \sum_{i=1}^{m} y^{(i)} \log h_\omega\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\omega\left(x^{(i)}\right)\right)$$

Gradient Descent for Logistic Regression

$$\ell(\omega) = \log \mathscr{L}(\omega) = \sum_{i=1}^{m} y^{(i)} \log h_\omega\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\omega\left(x^{(i)}\right)\right)$$

- Think about a single data point with a single parameter ω for the simplicity.

$$\begin{aligned}
&\frac{\partial}{\partial \omega}[y \log(\sigma) + (1-y)\log(1-\sigma)] \\
&= y\frac{\sigma'}{\sigma} + (1-y)\frac{-\sigma}{1-\sigma} \\
&= \left(\frac{y}{\sigma} - \frac{1-y}{1-\sigma}\right)\sigma' \\
&= \frac{y-\sigma}{\sigma(1-\sigma)}\sigma' \\
&= \frac{y-\sigma}{\sigma(1-\sigma)}\sigma(1-\sigma)x \\
&= (y-\sigma)x
\end{aligned}$$

- For m training data points with parameters ω

$$\frac{\partial \ell(\omega)}{\partial \omega_j} = \sum_{i=1}^{m}\left(y^{(i)} - h_\omega\left(x^{(i)}\right)\right) x_j^{(i)} \overset{\text{vectorization}}{=} (y - h_\omega(x))^T x_j = x_j^T (y - h_\omega(x))$$

Gradient Descent for Logistic Regression

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \end{bmatrix}$$

$$\frac{\partial \ell(\omega)}{\partial \omega_j} = \sum_{i=1}^{m} \left(y^{(i)} - h_\omega\left(x^{(i)}\right)\right) x_j^{(i)}$$

$$\overset{\text{vectorization}}{=} \quad (y - h_\omega(x))^T x_j = x_j^T\,(y - h_\omega(x))$$

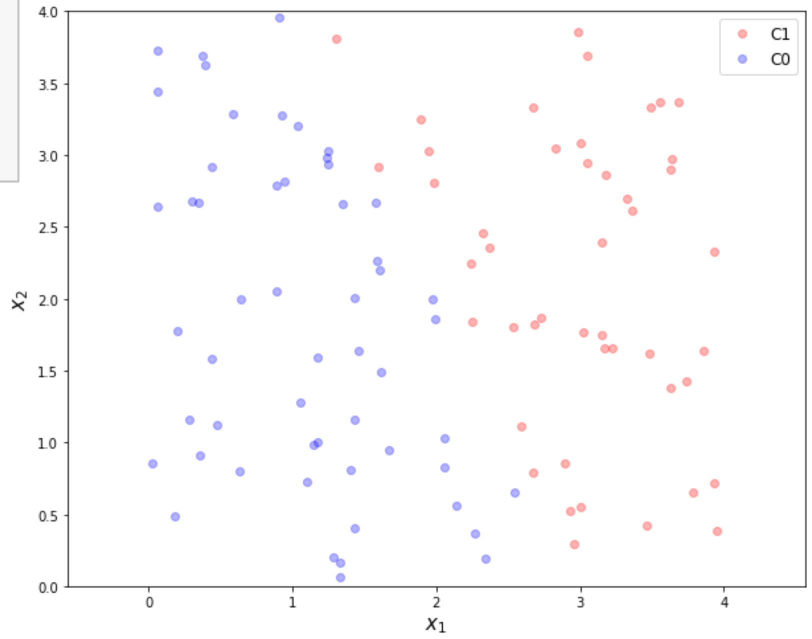
$$\nabla \ell(\omega) = \begin{bmatrix} \frac{\partial \ell(\omega)}{\partial \omega_0} \\ \frac{\partial \ell(\omega)}{\partial \omega_1} \\ \frac{\partial \ell(\omega)}{\partial \omega_2} \end{bmatrix} = X^T\,(y - h_\omega(x)) = X^T\,(y - \sigma(X\omega))$$

- Maximization problem
- Be careful on matrix shape

$$\omega \leftarrow \omega - \eta\,(-\nabla \ell(\omega))$$

Logistic Regression in Python

```
# datat generation

m = 100
w = np.array([[-6], [2], [1]])
X = np.hstack([np.ones([m,1]), 4*np.random.rand(m,1), 4*np.random.rand(m,1)])

w = np.asmatrix(w)
X = np.asmatrix(X)

y = 1/(1 + np.exp(-X*w)) > 0.5

C1 = np.where(y == True)[0]
C0 = np.where(y == False)[0]

y = np.empty([m,1])
y[C1] = 1
y[C0] = 0
```



Logistic Regression in Python

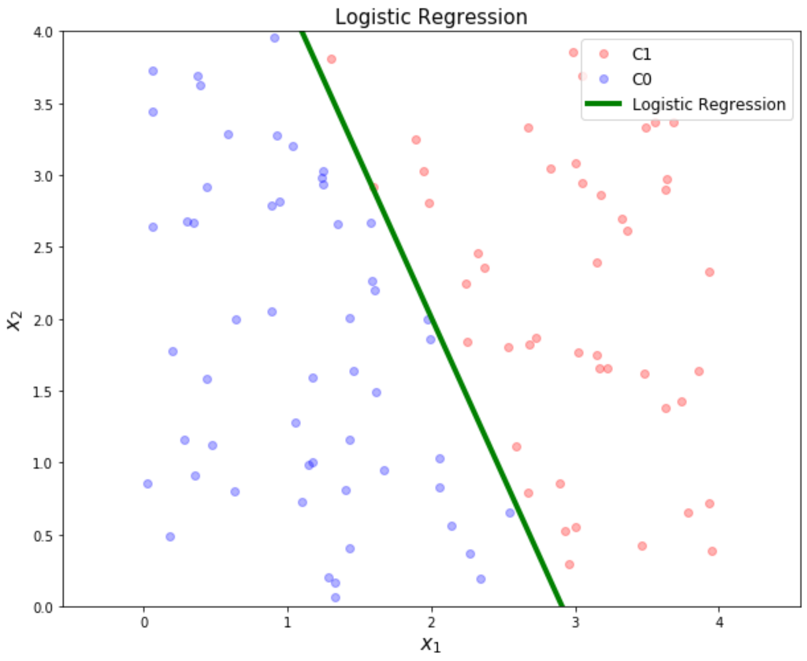
```python
# be careful with matrix shape

def h(x,w):
    return 1/(1 + np.exp(-x*w))
```

```python
alpha = 0.0001
w = np.zeros([3,1])

for i in range(1000):
    df = -X.T*(y - h(X,w))
    w = w - alpha*df

print(w)
```

$$h_\omega(x) = h(x\,;\omega) = \sigma\left(\omega^T x\right) = \frac{1}{1+e^{-\omega^T x}}$$

$$\nabla \ell(\omega) = \begin{bmatrix} \dfrac{\partial \ell(\omega)}{\partial \omega_0} \\ \dfrac{\partial \ell(\omega)}{\partial \omega_1} \\ \dfrac{\partial \ell(\omega)}{\partial \omega_2} \end{bmatrix} = X^T\left(y - h_\omega(x)\right) = X^T\left(y - \sigma(X\omega)\right)$$

$$\omega \leftarrow \omega - \eta\left(-\nabla \ell(\omega)\right)$$



Logistic Regression using Scikit-Learn

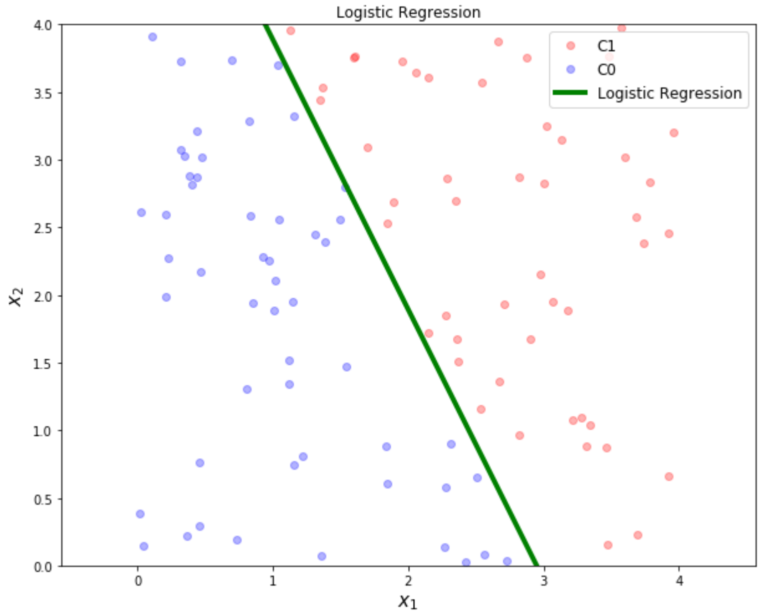
```
X = X[:,1:3]

X.shape
```

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \qquad \omega_0, \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots \end{bmatrix}, \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \end{bmatrix}$$

```
from sklearn import linear_model

clf = linear_model.LogisticRegression(solver='lbfgs')
clf.fit(X,np.ravel(y))
```

```
w0 = clf.intercept_[0]
w1 = clf.coef_[0,0]
w2 = clf.coef_[0,1]

xp = np.linspace(0,4,100).reshape(-1,1)
yp = - w1/w2*xp - w0/w2
```



Multiclass Classification

Multiclass Classification

- Generalization to more than 2 classes is straightforward
  - one vs. all (one vs. rest)
  - one vs. one

- Using the softmax function instead of the logistic function
  - (refer to [UFLDL Tutorial]())
  - see them as probability

$$P\left(y=k \mid x, \omega\right)=\frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)} \in [0,1]$$

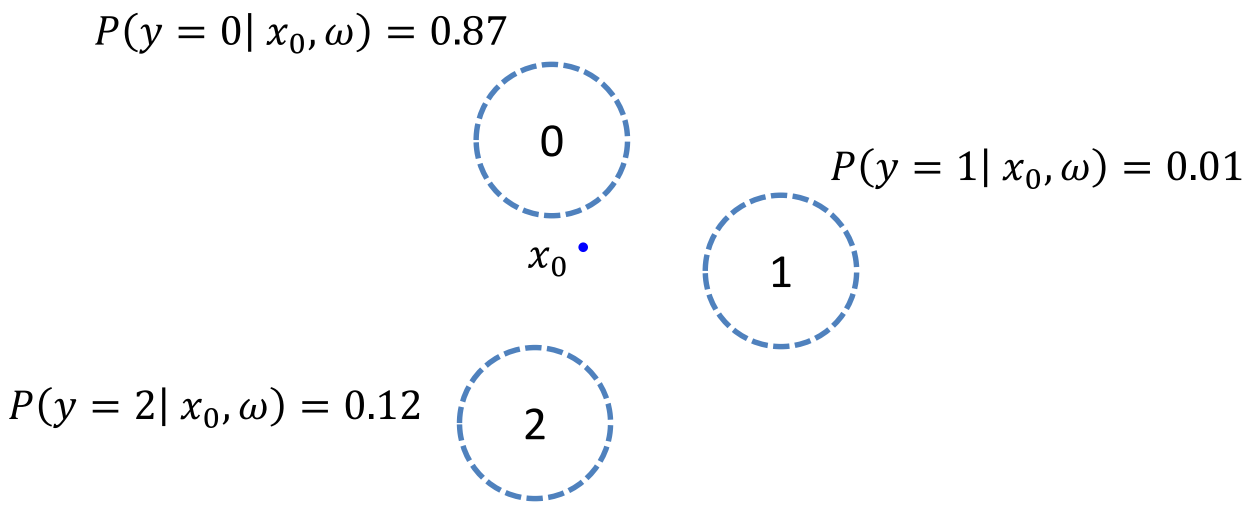
- We maintain a separator weight vector ω_k for each class k

Softmax function

- We maintain a separator weight vector ω_k for each class k
- Using the softmax function instead of the logistic function
  - see them as probability

$$P\left(y=k \mid x, \omega\right)=\frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)} \in [0,1]$$

- For example,



Softmax regression

- Logistic regression can be replaced with softmax function

$$P\left(y=+1 \mid x, \omega\right)=\frac{1}{1+e^{-\omega^{T} x}} \quad \in \ [0,1]$$

$$p=\frac{1}{1+e^{-\omega^{T} x}}=\frac{\boxed{e^{\omega^{T} x}}}{\boxed{e^{\omega^{T} x}+1}}$$

$$1-p=\frac{\boxed{1}}{\boxed{e^{\omega^{T} x}+1}}$$

$$\leftrightarrow$$

$$P\left(y=k \mid x, \omega\right)=\frac{\boxed{\exp\left(\omega_{k}^{T} x\right)}}{\boxed{\sum_{k} \exp\left(\omega_{k}^{T} x\right)}} \in [0,1]$$

Cross entropy

$$\ell(\omega)=-\sum_{i=1}^{m} y^{(i)} \log h_{\omega}\left(x^{(i)}\right)+\left(1-y^{(i)}\right) \log \left(1-h_{\omega}\left(x^{(i)}\right)\right)$$

$$\ell(\omega)=-\sum_{i=1}^{m} y^{(i)} \log s_{\omega}\left(x^{(i)}\right)$$

Logistic regression

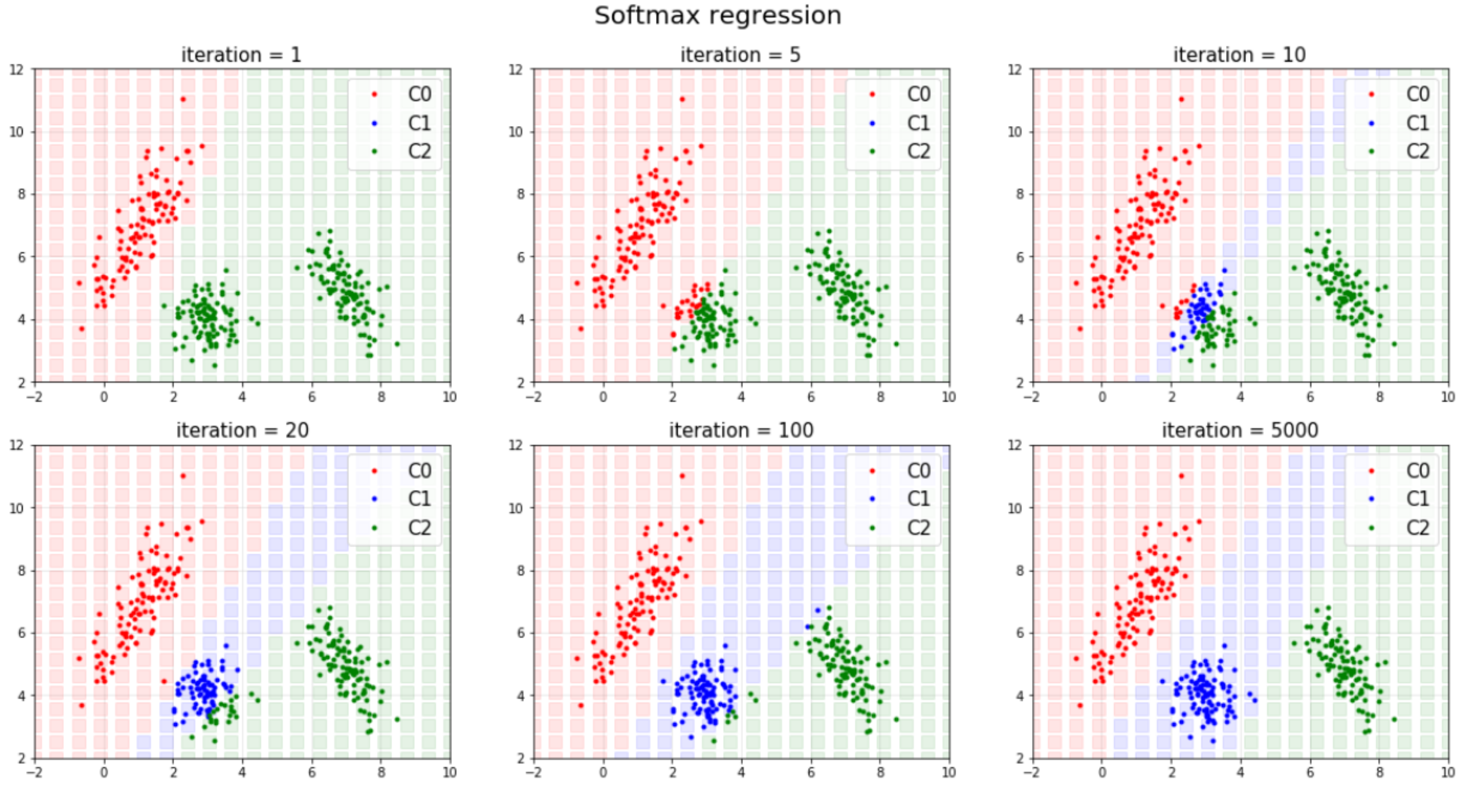
Softmax regression

Softmax regression

```python
def s(X, w):
    scores = np.dot(X, w)
    softmax = (np.exp(scores).T/np.sum(np.exp(scores), axis = 1)).T
    return softmax
```

$$P(y = k \mid x, \omega) = \frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)} \in [0, 1]$$

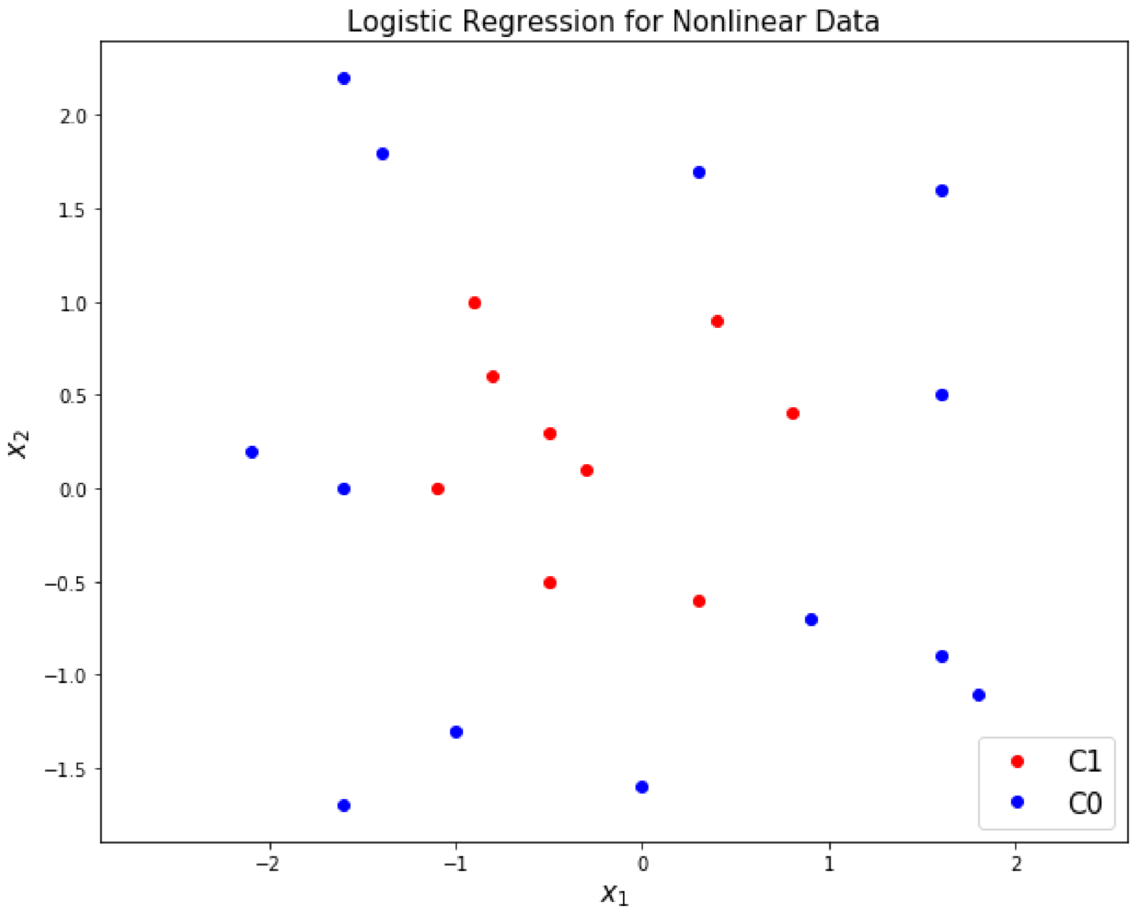
```python
alpha = 0.1
w = np.zeros([3,3])

losses = []
weights = []
for i in range(5000):
     loss  = - 1/len(X)*np.sum(Y * s(X, w))
     df = - 1/len(X)*np.dot(X.T, (Y - s(X, w)))
     w = w - alpha*df
     losses.append(loss)
     weights.append(w)
```



Non-linear Classification

Non-linear Classification

- Same idea as non-linear regression: non-linear features
  - Explicit or implicit Kernel



Explicit Kernel

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \Longrightarrow \quad z = \phi(x) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

```
N = X1.shape[0]
M = X0.shape[0]

X = np.vstack([X1, X0])
y = np.vstack([np.ones([N,1]), -np.ones([M,1])])

X = np.asmatrix(X)
y = np.asmatrix(y)

m = N + M
Z = np.hstack([np.ones([m,1]), np.sqrt(2)*X[:,0], np.sqrt(2)*X[:,1], np.square(X[:,0]),
               np.sqrt(2)*np.multiply(X[:,0], X[:,1]), np.square(X[:,1])])

w = cvx.Variable([6, 1])
obj = cvx.Minimize(cvx.sum(cvx.logistic(-cvx.multiply(y,Z*w))))
prob = cvx.Problem(obj).solve()

w = w.value
```

Non-linear Classification