# Classification: Perceptron

Industrial AI Lab.

Prof. Seungchul Lee

Classification

- Where y is a discrete value
  - Develop the classification algorithm to determine which class a new input should fall into

- Start with a binary class problem
  - Later look at multiclass classification problem, although this is just an extension of binary classification

- We could use linear regression
  - Then, threshold the classifier output (i.e. anything over some value is yes, else no)
  - linear regression with thresholding seems to work

Classification

- We will learn
  - Perceptron
  - Support vector machine (SVM)
  - Logistic regression

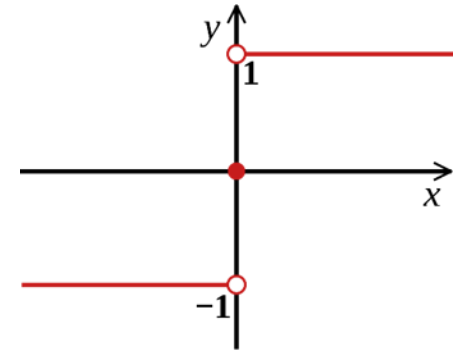- To find
  a classification boundary
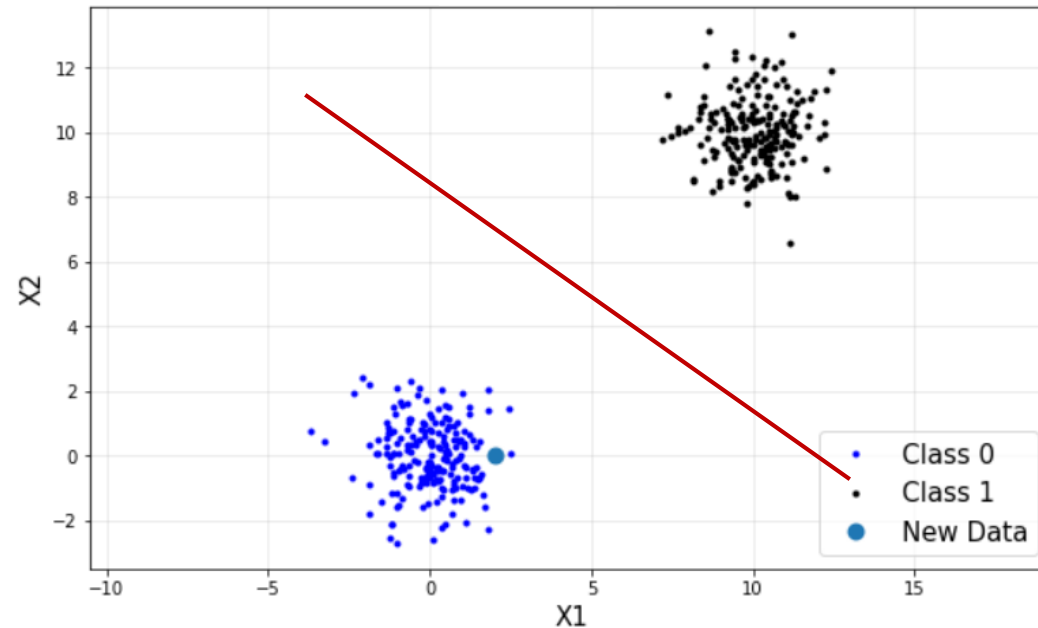
Perceptron

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

- Weights $\omega = \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_d \end{bmatrix}$

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

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

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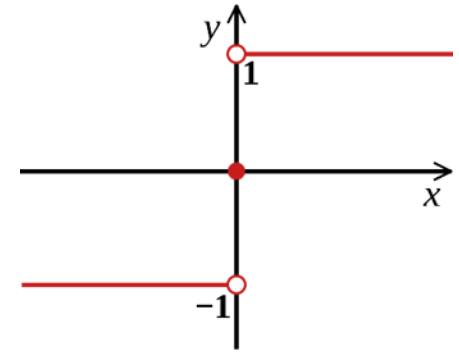
Perceptron

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

- Introduce an artificial coordinate $x_0 = 1$:

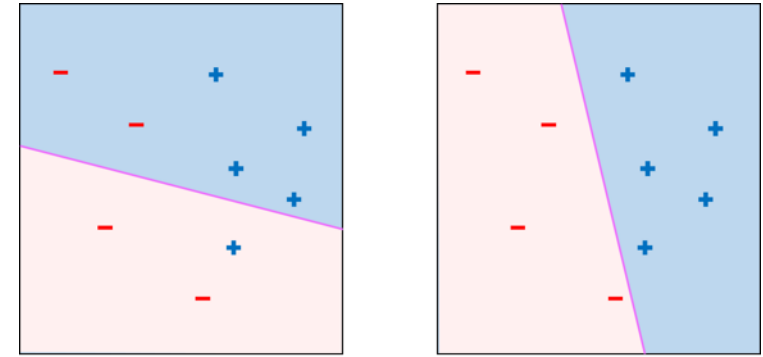$$h(x) = \text{sign}\left(\sum_{i=0}^{d} \omega_i x_i\right)$$
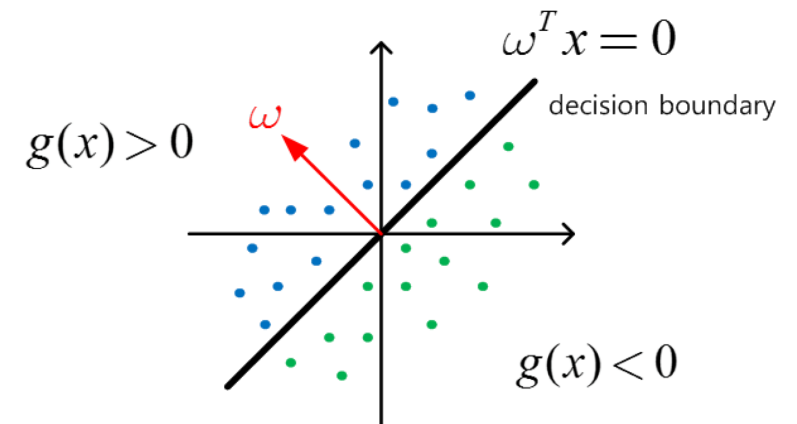
- In a vector form, the perceptron implements

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

Perceptron

- Works for linearly separable data

- 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 $x_0 = 1$
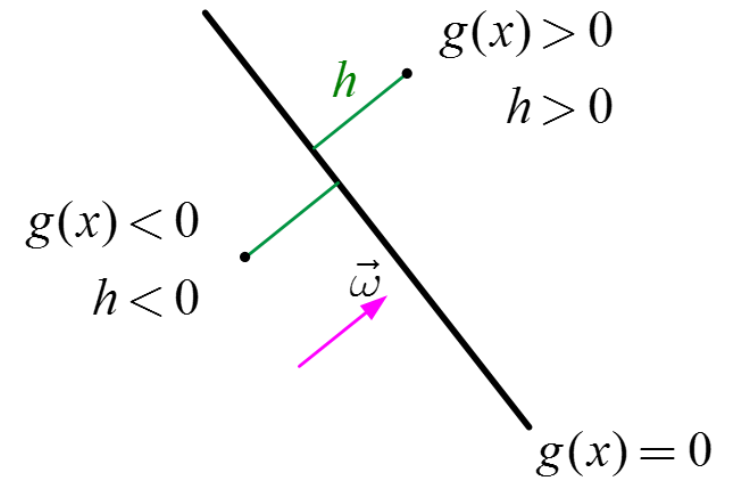

Linearly separable data
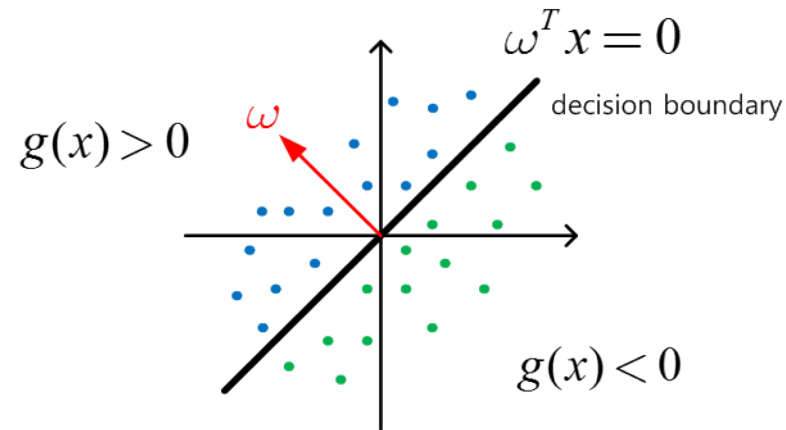
Sign

- Sign with respect to a line

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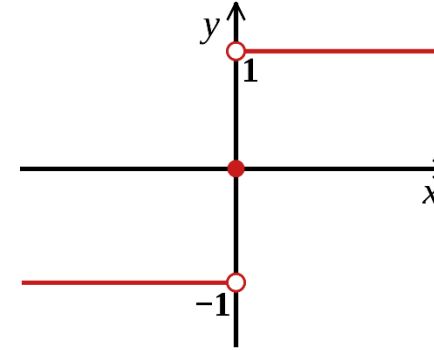
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = \omega^T x$$

$g(x) > 0$

$h > 0$

h

$g(x) < 0$

$h < 0$

$\vec{\omega}$

$g(x) = 0$

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

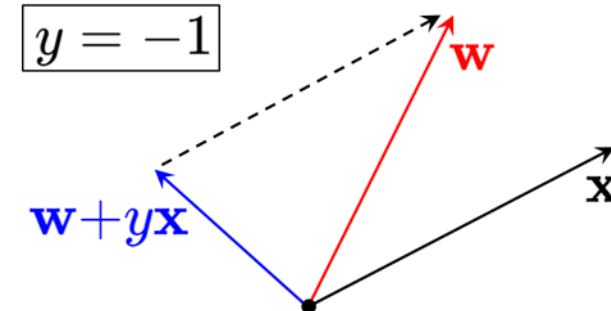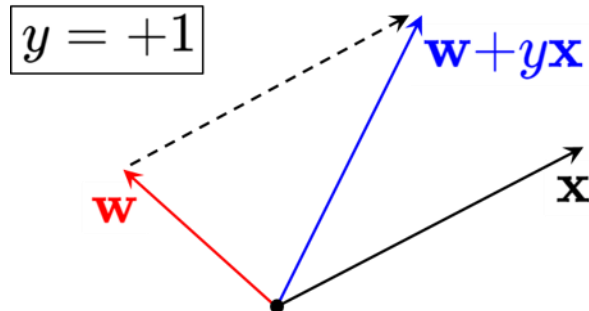$$h(x) = \text{sign}\left(\omega^T x\right)$$

- 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

$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$$
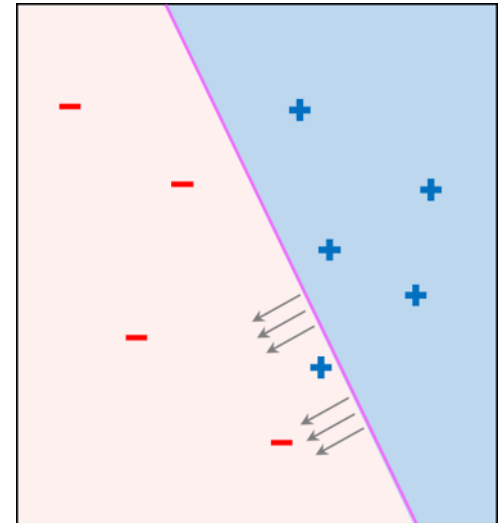
4. And run a PLA iteration on it

5. That's it!

Diagram of Perceptron

input

x_0

ω_0

x_1

w_1

w_d

x_d

Sign[ ]

+

update

+

-

+

$\hat{y}$ Predicted output

y Actual output
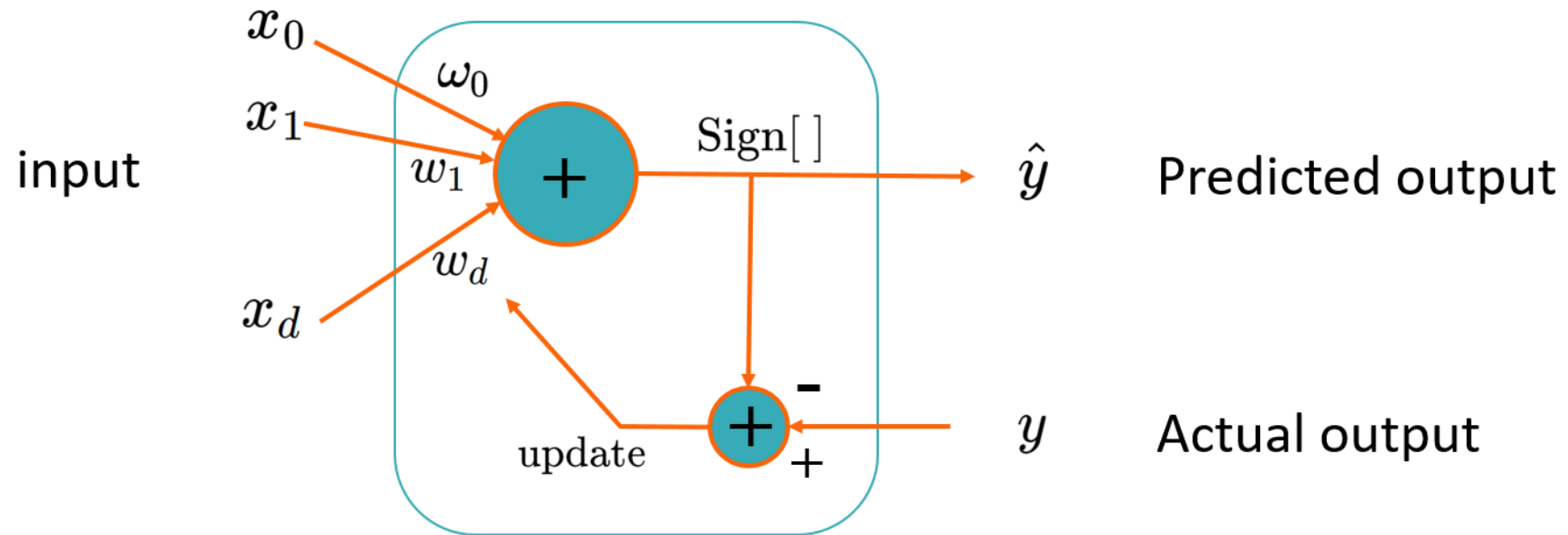
Perceptron Loss Function

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

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

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

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

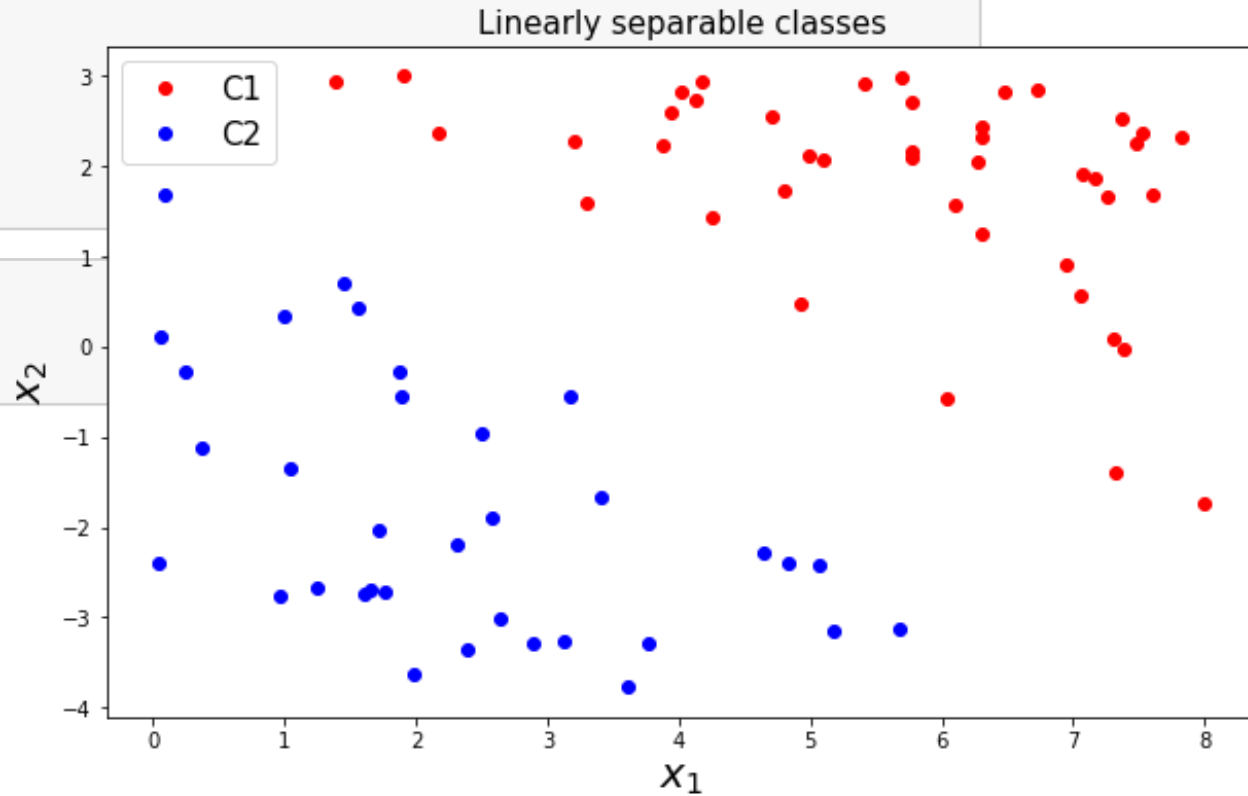# Perceptron Algorithm in Python

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

```python
#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
```

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


Linearly separable classes

Perceptron Algorithm in Python

- Unknown parameters ω

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

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

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

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

```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

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

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

```python
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)
```
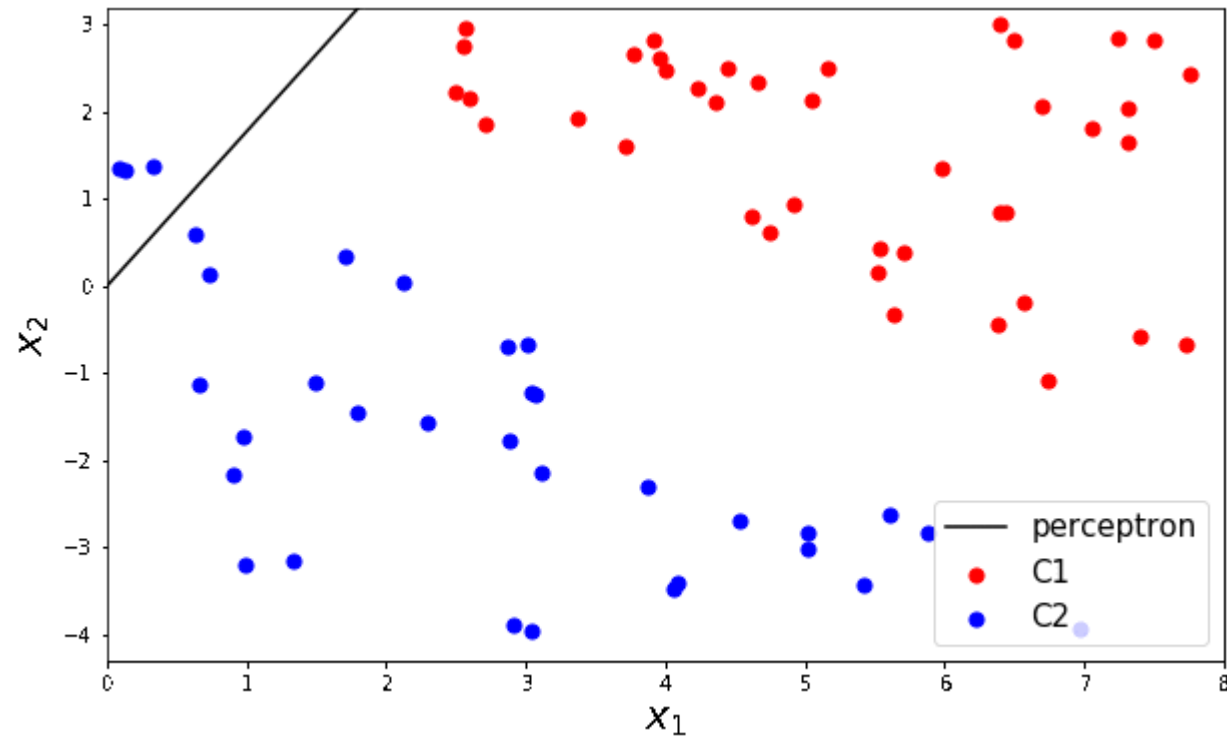
Perceptron Algorithm in Python

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

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

```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 Algorithm in Python
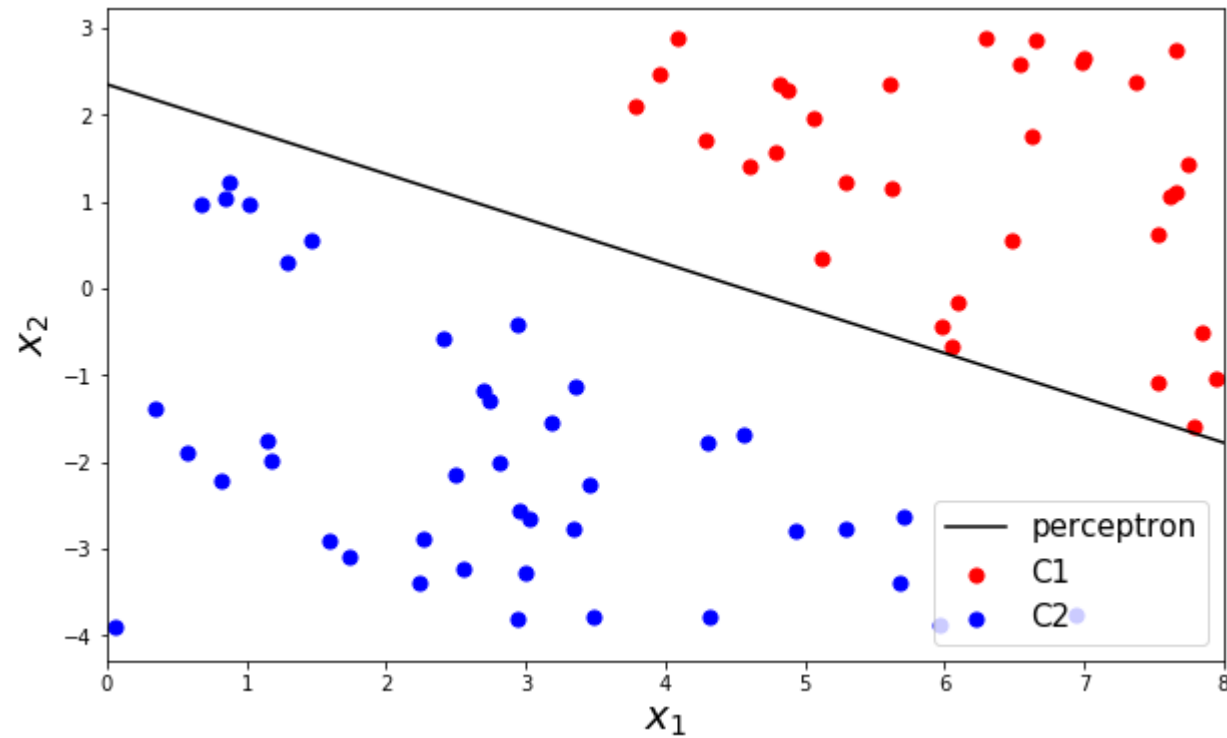
Scikit-Learn for Perceptron

```
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