# Linear Classification

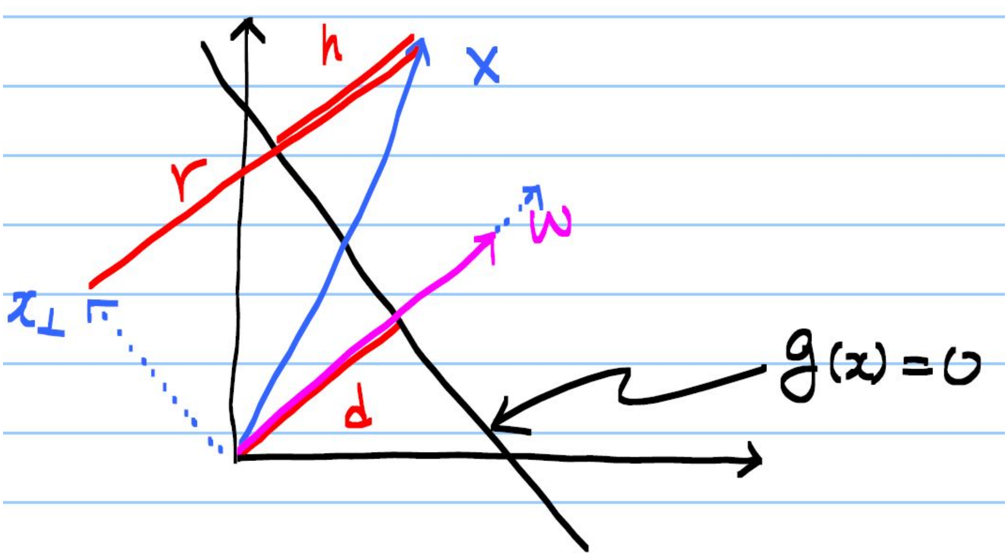
by Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents



# 1. Distance from a Line

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



- If $\vec{p}$ and $\vec{q}$ are on the decision line

$$
\begin{aligned}
g(\vec{p}) = g(\vec{q}) = 0 \implies & \omega^T \vec{p} + \omega_0 = \omega^T \vec{q} + \omega_0 = 0 \\
\implies & \omega^T (\vec{p} - \vec{q}) = 0
\end{aligned}
$$

$$
\therefore\ \omega : \text{normal to the line (orthogonal)} \implies \text{tells the direction of the line}
$$

- If $x$ is on the line and $x = d\frac{\omega}{\|\omega\|}$ (where $d$ is a normal distance from the origin to the line)

$$
\begin{aligned}
g(x) &= \omega^T x + \omega_0 = 0 \\
\implies & \omega^T d\frac{\omega}{\|\omega\|} + \omega_0 = d\frac{\omega^T\omega}{\|\omega\|} + \omega_0 = d\|\omega\| + \omega_0 = 0 \\
\therefore d &= -\frac{\omega_0}{\|\omega\|}
\end{aligned}
$$

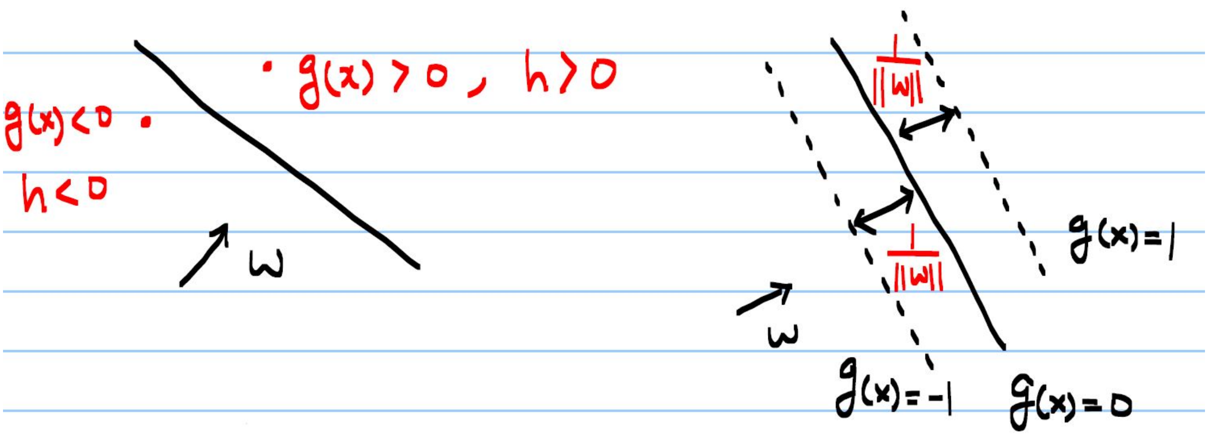
- for any vector of $x$

$$
x = x_\perp + r\frac{\omega}{\|\omega\|}
$$

$$
\omega^T x = \omega^T \left( x_\perp + r\frac{\omega}{\|\omega\|} \right) = r\frac{\omega^T\omega}{\|\omega\|} = r\|\omega\|
$$

$$
\begin{aligned}
g(x) &= \omega^T x + \omega_0 \\
&= r\|\omega\| + \omega_0 \qquad (r = d + h) \\
&= (d + h)\|\omega\| + \omega_0 \\
&= \left( -\frac{\omega_0}{\|\omega\|} + h \right) \|\omega\| + \omega_0 \\
&= h\|\omega\|
\end{aligned}
$$

$$
\therefore\ h = \frac{g(x)}{\|\omega\|} \implies \textbf{orthogonal distance from the line}
$$

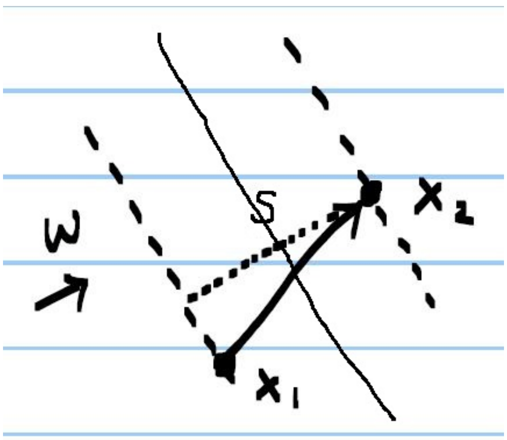


## Another method to find a distance between $g(x) = -1$ and $g(x) = 1$

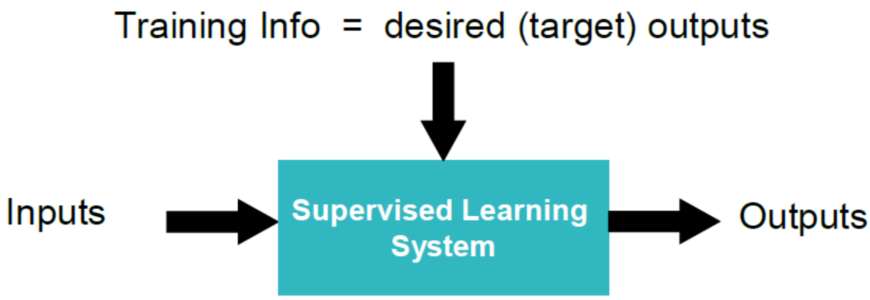
suppose $g(x_1) = -1,\ g(x_2) = 1$

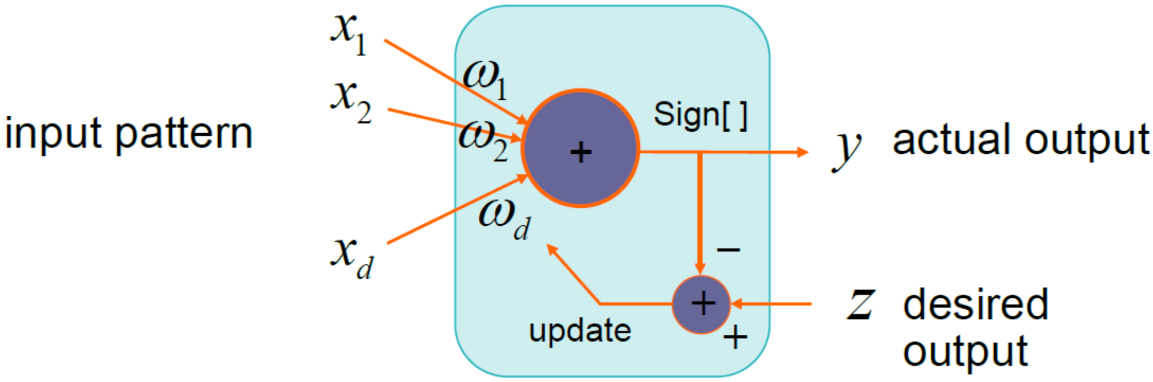
$$\begin{aligned} \omega^T x_1 + \omega_0 &= -1 \\ \omega^T x_2 + \omega_0 &= 1 \end{aligned} \implies \omega^T (x_2 - x_1) = 2$$

$$s = \left\langle \frac{\omega}{\|\omega\|}, x_2 - x_1 \right\rangle = \frac{1}{\|\omega\|}\omega^T (x_2 - x_1) = \frac{2}{\|\omega\|}$$



# 2. Supervised Learning




# 3. Classification

- where $y$ is a discrete value
  - develop the classification algorithm to determine which class a new input should fall into
- start with binary class problems
  - 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
- We will learn
  - perceptron
  - support vector machine
  - logistic regression

# 4. Perceptron

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

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

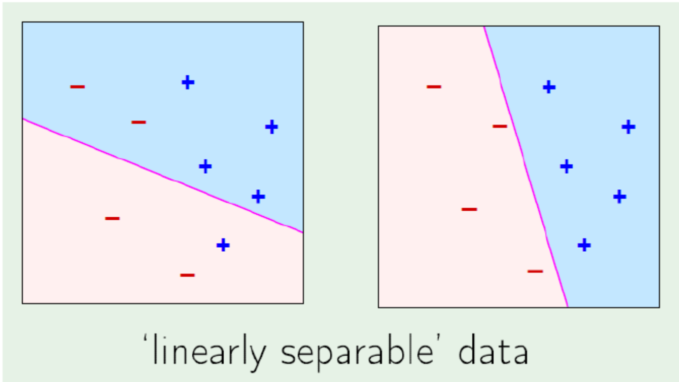
$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) - \text{threshold}\right) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) + \omega_0\right)$$

- Introduce an artificial coordinate $x_0 = 1$:

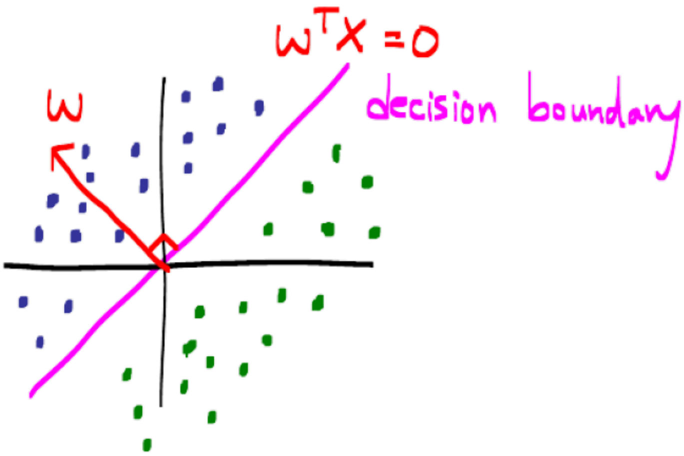
$$h(x) = \text{sign}\left(\sum_{i=0}^{d} \omega_i x_i\right)$$

- In vector form, the perceptron implements

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



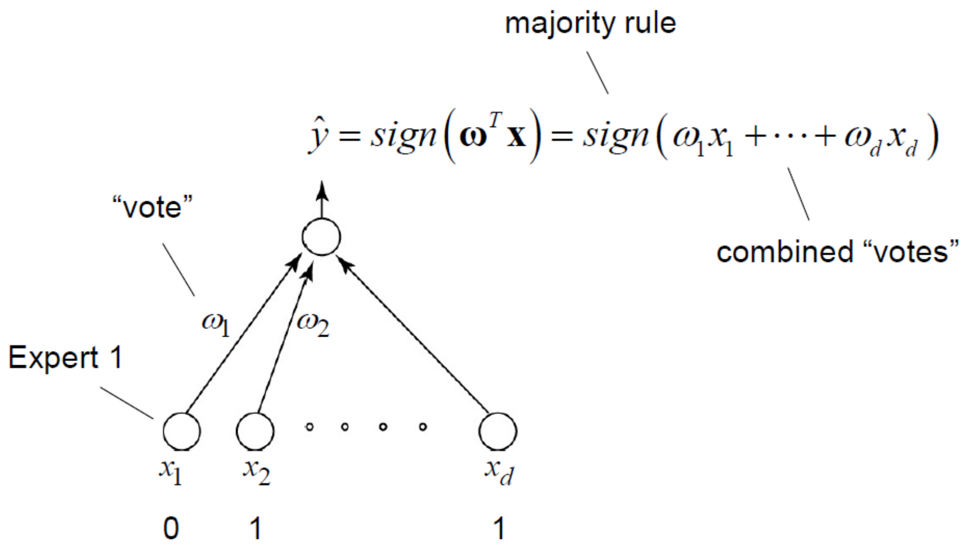

- Hyperplane
  - Separates a D-dimensional space into two half-spaces
  - Defined by an outward pointing normal vector $\omega$
  - $\omega$ is orthogonal to any vector lying on the hyperplane
  - assume the hyperplane passes through origin, $\omega^T x = 0$ with $x_0 = 1$



## 4.1. Linear Classifier

- represent the decision boundary by a hyperplane $\omega$
- The linear classifier is a way of combining expert opinion.
- In this case, each opinion is made by a binary "expert"

- Goal: to learn the hyperplane $\omega$ using the training data




## 4.2. Perceptron Algorithm

The perceptron implements

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

Given the training set

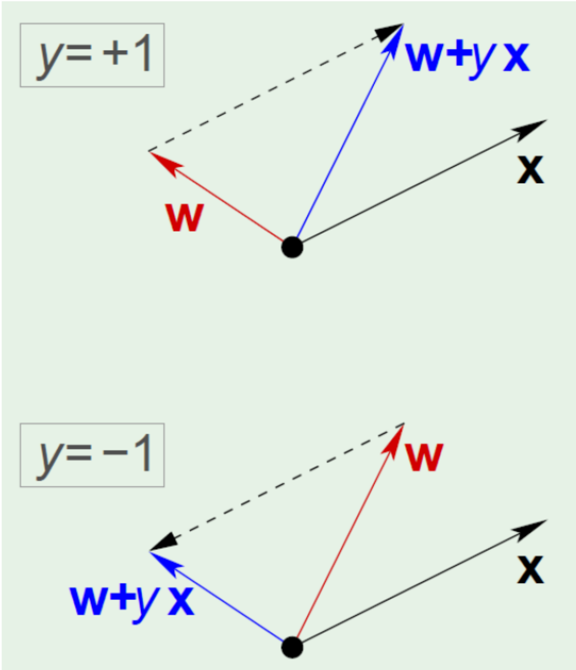
$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N) \quad \text{where } y_i \in \{-1, 1\}$$

1) pick a misclassified point

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

2) and update the weight vector

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




**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 $\omega_{old}^T x_n$

## 4.3. Iterations of Perceptron

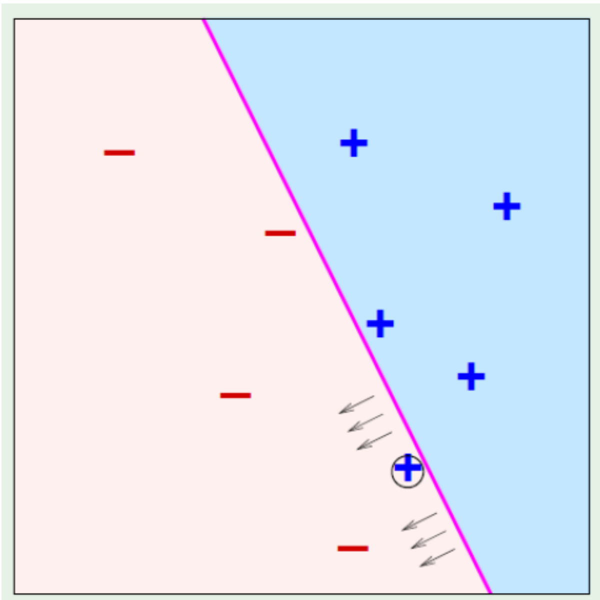
1. Randomly assign $\omega$

2. One iteration of the PLA (perceptron learning algorithm)
$$\omega \leftarrow \omega + yx$$
where $(x, y)$ is a misclassified training point

3. At iteration $t = 1, 2, 3, \cdots,$ 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!



## 4.4. 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 misclassified, *i.e.*, $y_n \cdot \left(\omega^T x_n\right) < 0$

**note**: $\operatorname{sign}\left(\omega^T x_n\right) \neq y_n$ is equivalent to $y_n \cdot \left(\omega^T x_n\right) < 0$

## 4.5. 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 as well
- 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 cover Support Vector Machine

## 4.6. Python Example

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

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

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

In [1]:

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

% matplotlib inline
```

In [2]:

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

g0 = 0.8*x1 + x2 - 3
g1 = g0 - 1
g2 = g0 + 1
```

In [3]:

```
C1 = np.where(g1 >= 0)
C2 = np.where(g2 < 0)
print(C1)
```

```
(array([ 3,  7, 12, 16, 18, 19, 20, 24, 28, 31, 32, 34, 35, 37, 39, 42, 4
3,
       51, 53, 60, 63, 65, 67, 72, 79, 84, 85, 86, 87, 89, 95, 96, 97]), a
rray([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0]))
```

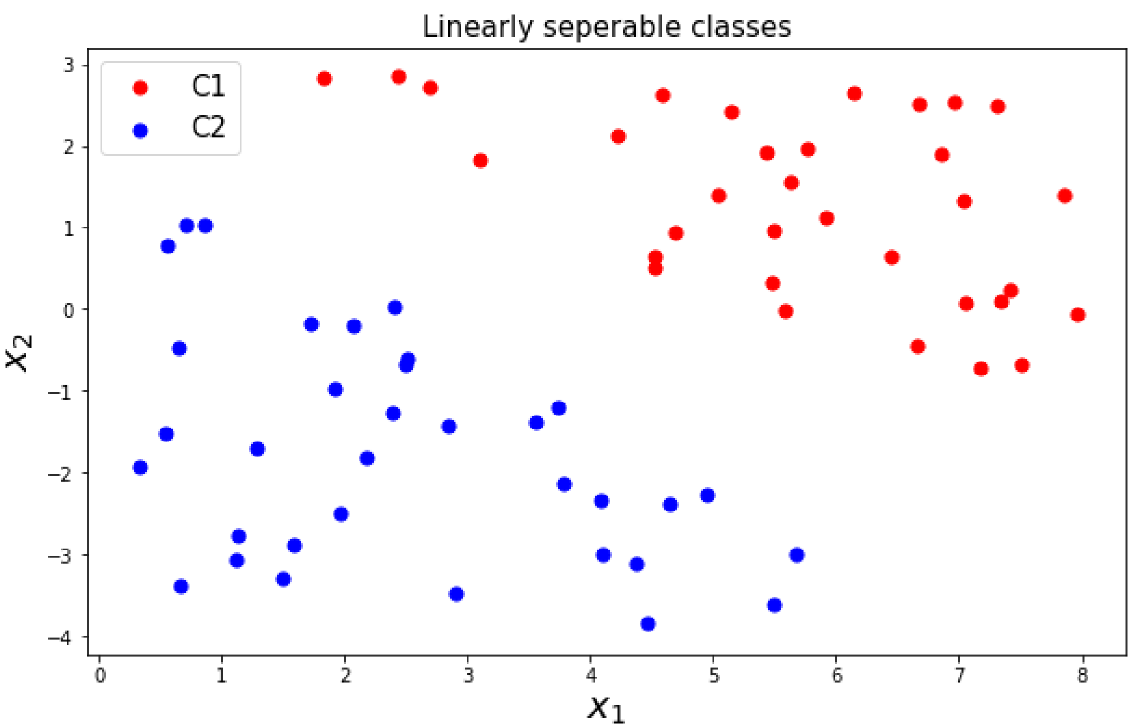
In [4]:

```
C1 = np.where(g1 >= 0)[0]
C2 = np.where(g2 < 0)[0]
print(C1.shape)
print(C2.shape)
```

```
(33,)
(34,)
```

In [5]:

```
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.title('Linearly seperable classes', fontsize=15)
plt.legend(loc='upper left', fontsize=15)
plt.xlabel(r'$x_1$', fontsize=20)
plt.ylabel(r'$x_2$', fontsize=20)
plt.show()
```



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

In [6]:

```
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)
```

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$
$$\omega \leftarrow \omega + yx$$

where $(x, y)$ is a misclassified training point

In [7]:

```
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)
```

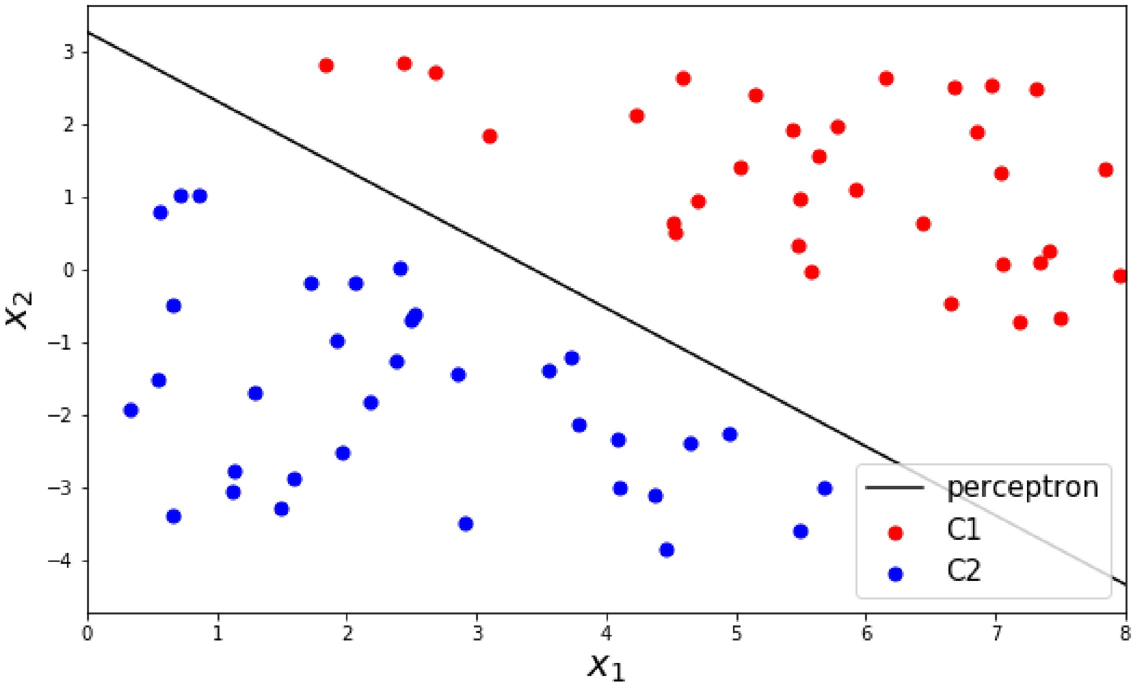
```
[[-4.        ]
 [ 1.16177769]
 [ 1.22225118]]
```

$$g(x) = \omega^T x + \omega_0 = \omega_1 x_1 + \omega_2 x_2 + \omega_0 = 0$$
$$\implies x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\omega_0}{\omega_2}$$

In [8]:

```
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 = 4, fontsize = 15)
plt.show()
```



In [9]:

```
# animation

import matplotlib.animation as animation
% matplotlib qt

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1)
plot_C1, = ax.plot(x1[C1], x2[C1], 'go', label='C1')
plot_C2, = ax.plot(x1[C2], x2[C2], 'bo', label='C2')
plot_perceptron, = ax.plot([], [], 'k', label='perceptron')

ax.set_xlim(0, 8)
ax.set_ylim(-3.5, 4.5)
ax.set_xlabel(r'$x_1$', fontsize=20)
ax.set_ylabel(r'$x_2$', fontsize=20)
ax.legend(fontsize=15, loc='upper left')

n_iter = y.shape[0]

def init():
    plot_perceptron.set_data(x1p, x2p)
    return plot_perceptron,

def animate(i):
    global w
    idx = i%n_iter
    if y[idx,0] != np.sign(X[idx,:]*w)[0,0]:
        w += y[idx,0]*X[idx,:].T
        x2p = - w[1,0]/w[2,0]*x1p - w[0,0]/w[2,0]
        plot_perceptron.set_data(x1p, x2p)
    return plot_perceptron,

w = np.ones([3,1])
x1p = np.linspace(0,8,100).reshape(-1,1)
x2p = - w[1,0]/w[2,0]*x1p - w[0,0]/w[2,0]

ani = animation.FuncAnimation(fig, animate, np.arange(0, n_iter**2), init_func=init,
                              interval=0, repeat=False)
plt.show()
```

In [10]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.
js')
```