Regression and Classification

by Prof. Seungchul Lee
Industrial AI Lab
http://isystems.unist.ac.kr/
POSTECH

Table of Contents

1. Regression

- $\hat{y}_i = f(x_i, \theta)$ in general
- In many cases, a linear model to predict y_i is assumed

Given $\begin{cases} x_i : \text{inputs} \\ y_i : \text{outputs} \end{cases}$, Find θ_1 and θ_2

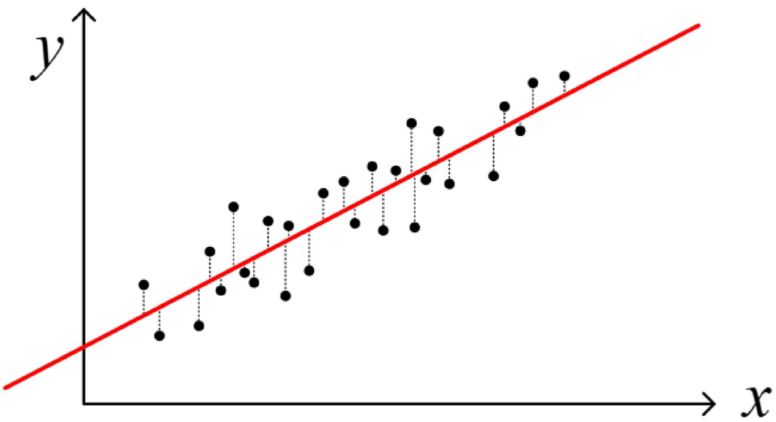
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_1 x_i + \theta_2$$

- $\hat{y}_i$: predicted output
- $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$: Model parameters

$$\hat{y}_i = f(x_i, \theta) \text{ in general}$$

- in many cases, a linear model to predict y_i used

$$\hat{y}_i = \theta_1 x_i + \theta_2 \text{ such that } \min_{\theta_1, \theta_2} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$



Linear Regression as Optimization</fotn>

- How to find model parameters, $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

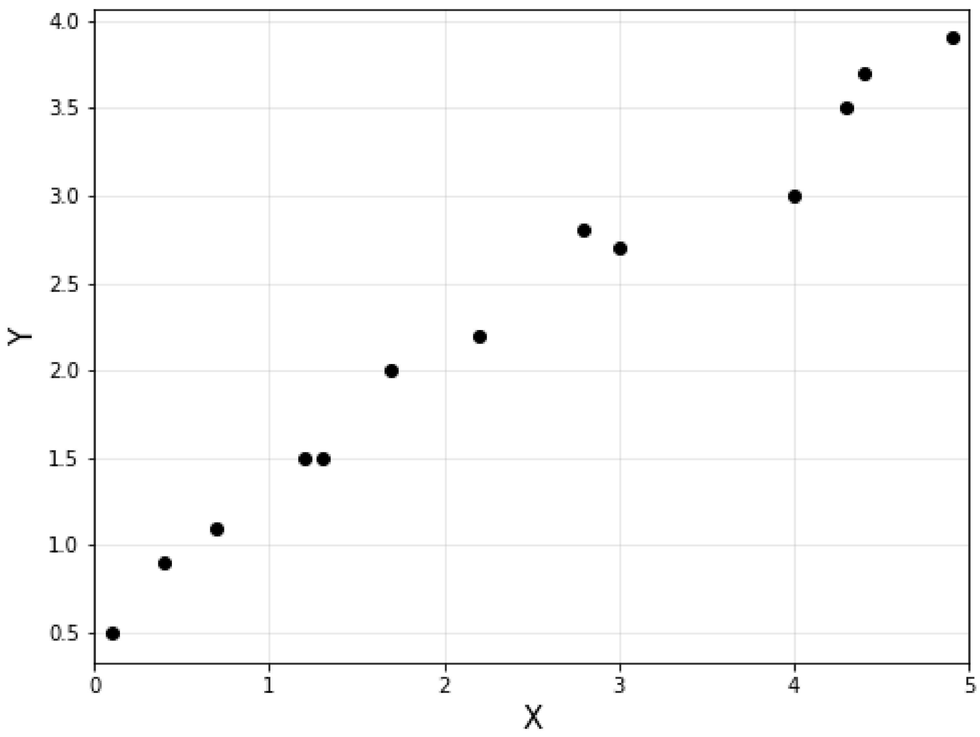
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_1 x_i + \theta_2$$

$$\hat{y}_i = \theta_1 x_i + \theta_2 \quad \text{such that} \ \min_{\theta_1, \theta_2} \sum_{\theta_1, \theta_2}^{m} (\hat{y}_i - y_i)^2$$

Regression

In [1]:

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

# data points in column vector [input, output]
x = np.array([0.1, 0.4, 0.7, 1.2, 1.3, 1.7, 2.2, 2.8, 3.0, 4.0, 4.3, 4.4, 4.9]).reshape(-1, 1)
y = np.array([0.5, 0.9, 1.1, 1.5, 1.5, 2.0, 2.2, 2.8, 2.7, 3.0, 3.5, 3.7, 3.9]).reshape(-1, 1)

# to plot
plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label="data")
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis('scaled')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```



4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Y
0
1
2
3
4
5
X

CVXPY

- Use CVXPY optimization (least-squared)
  - For convenience, we define a function that maps inputs to feature vectors, ϕ

$$\begin{aligned}
\hat{y}_i &= \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \\
&= \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \qquad\qquad \text{feature vector} \phi(x_i) = \begin{bmatrix} x_i \\ 1 \end{bmatrix} \\
&= \phi^T(x_i)\theta
\end{aligned}$$

$$\Phi = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} = \begin{bmatrix} \phi^T(x_1) \\ \phi^T(x_2) \\ \vdots \\ \phi^T(x_m) \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_m} \end{bmatrix} = \Phi\theta$$

parameter Estimation

- Model parameter estimation

$$\min_{\theta} \|\hat{y} - y\|_2 = \min_{\theta} \|A\theta - y\|_2$$

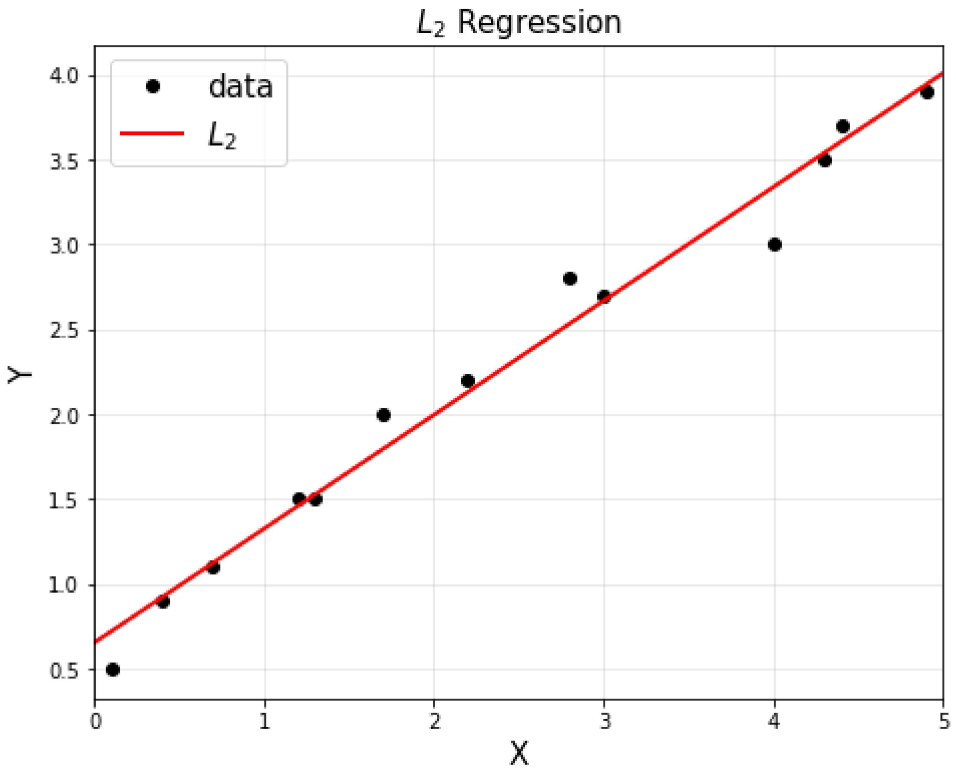
In [2]:

```
import cvxpy as cvx

m = y.shape[0]
#A = np.hstack([x, np.ones([m, 1])])
A = np.hstack([x, x**0])
A = np.asmatrix(A)

theta2 = cvx.Variable(2, 1)
obj = cvx.Minimize(cvx.norm(A*theta2-y, 2))
cvx.Problem(obj,[]).solve()

print('theta:\n', theta2.value)
```

```
theta:
 [[0.67129519]
 [0.65306531]]
```

In [3]:

```
# to plot
plt.figure(figsize=(10, 6))
plt.title('$L_2$ Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label="data")

# to plot a straight line (fitted line)
xp = np.arange(0, 5, 0.01).reshape(-1, 1)
theta2 = theta2.value
yp = theta2[0,0]*xp + theta2[1,0]

plt.plot(xp, yp, 'r', linewidth=2, label="$L_2$")
plt.legend(fontsize=15)
plt.axis('scaled')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```

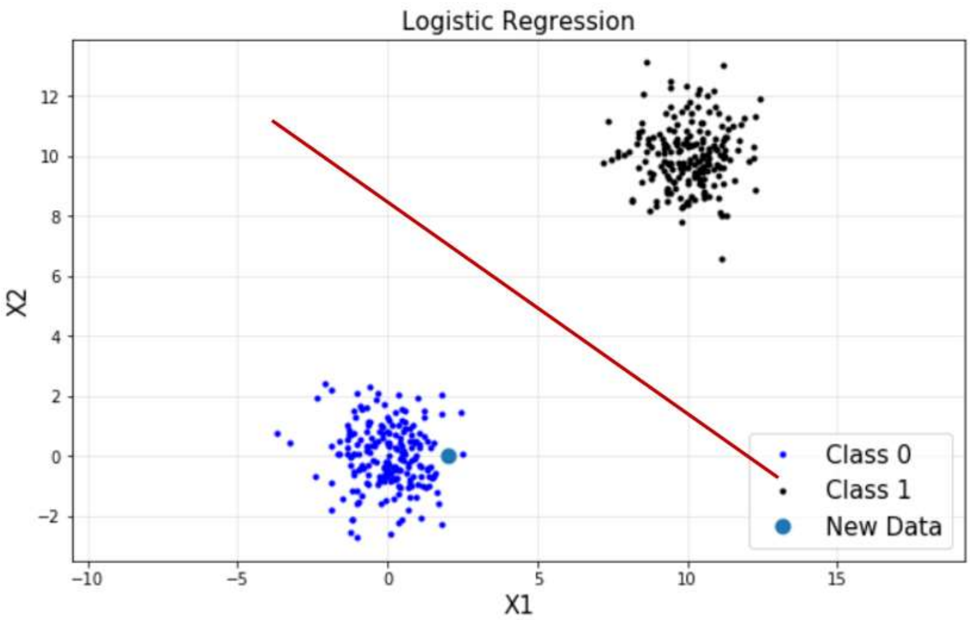


2. Classification

- Perceptron: make use of sign of data
  - discuss it later

- Logistic regression is a classification algorithm
  - don't be confused

- To find a classification boundary

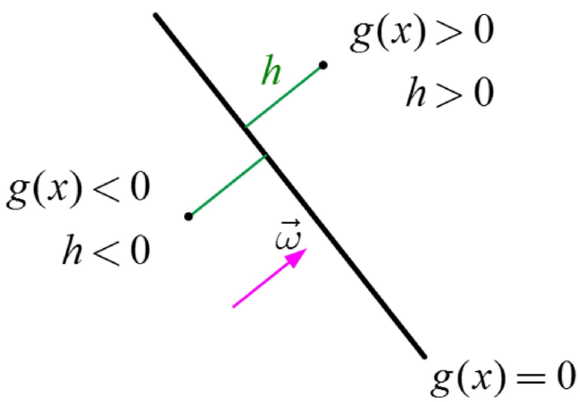


Sign

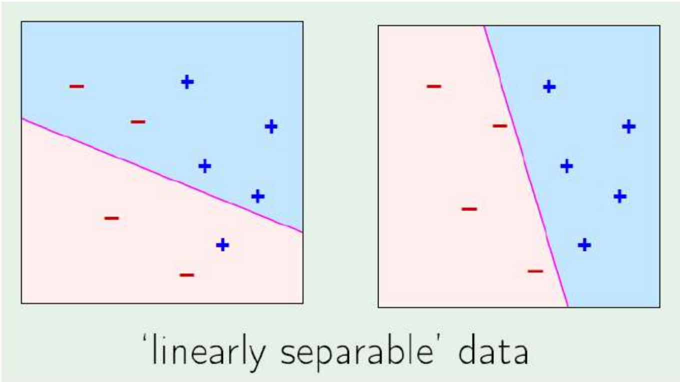
- Sign with respect to a line

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

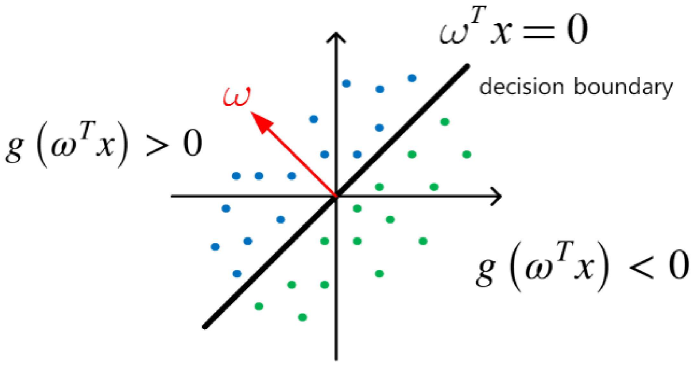
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \quad \omega = \begin{bmatrix} 1 \\ \omega_1 \\ \omega_2 \end{bmatrix} \quad \Longrightarrow g(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = \omega^T x$$



Perceptron



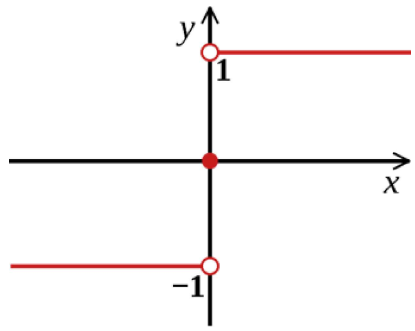
- Hyperplane
  - Separates a D-dimensional space into two half-spaces
  - Defined by an outward pointing normal vector
  - ω is orthogonal to any vector lying on the hyperplane

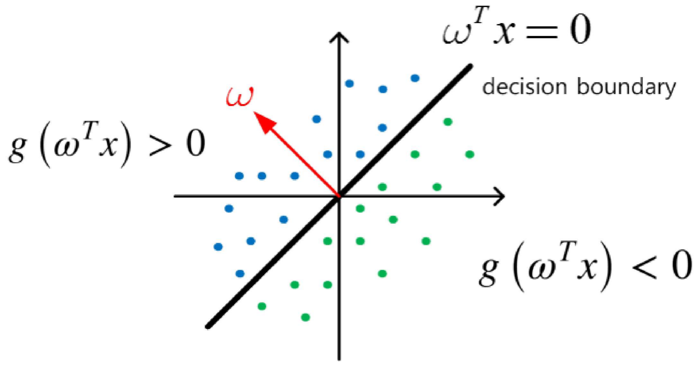


How to Find ω

- All data in class 1
  - $g(\omega^T x) > 0$

- All data in class 0
  - $g(\omega^T x) < 0$





Perceptron Algorithm

- The perceptron implements

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

- Given the training set

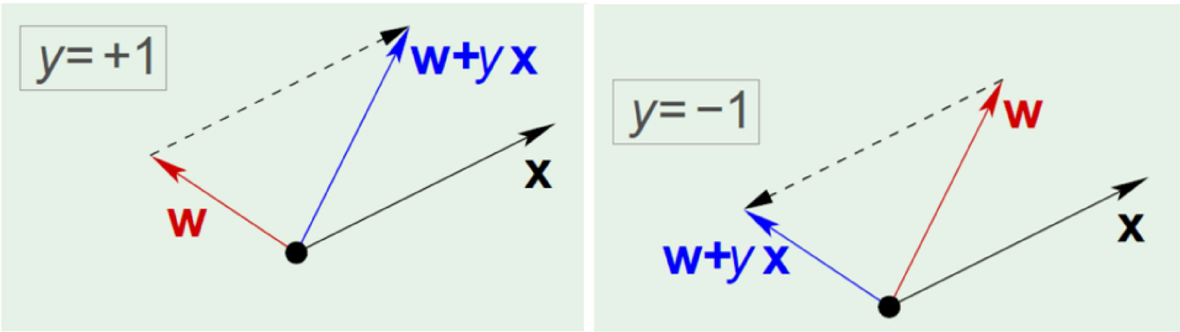
$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N) \quad \text{where } y_i \in \{-1, 1\}$$

```
1) pick a misclassified point
```

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

```
2) and update the weight vector
```

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



- 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

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

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

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

Python Example

In [9]:

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

% matplotlib inline
```

In [10]:

```
#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 [11]:

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

```
(array([ 2,  8, 10, 11, 16, 21, 22, 26, 27, 36, 38, 42, 44, 48, 53, 57, 61,
       62, 74, 79, 83, 86, 87, 89, 90, 96, 99], dtype=int64), array([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], dtype=int64))
```

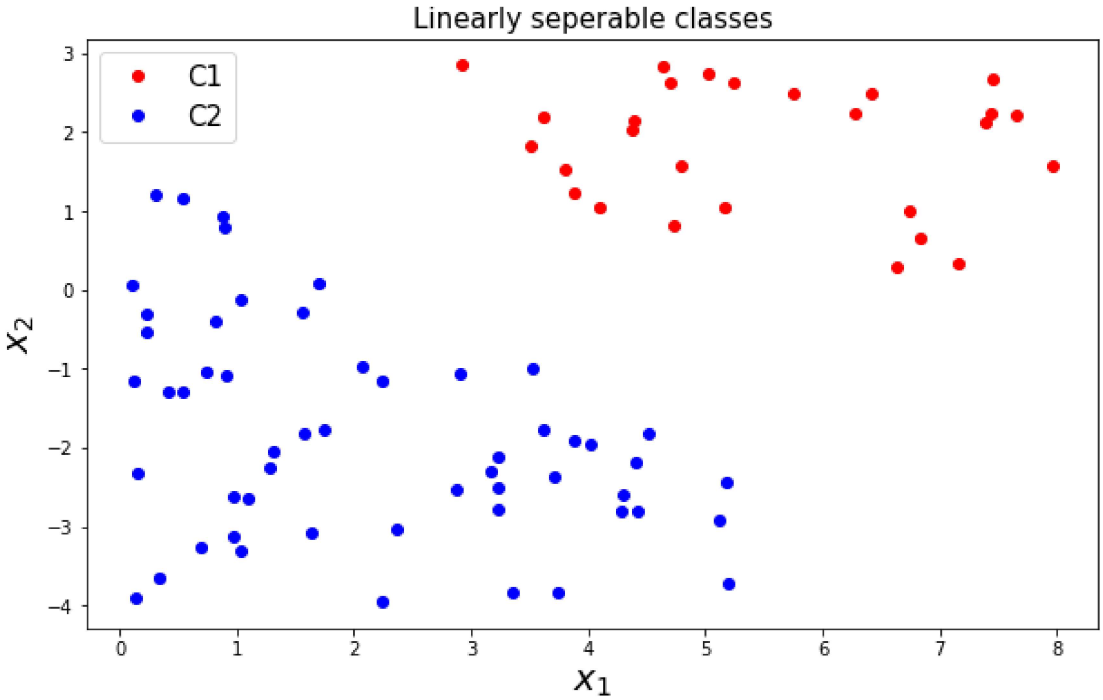
In [12]:

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

```
(27,)
(54,)
```

In [13]:

```
plt.figure(figsize=(10, 6))
plt.plot(x1[C1], x2[C1], 'ro', label='C1')
plt.plot(x1[C2], x2[C2], 'bo', 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()
```



$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \qquad x = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix}$$

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

In [15]:

```
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 [16]:

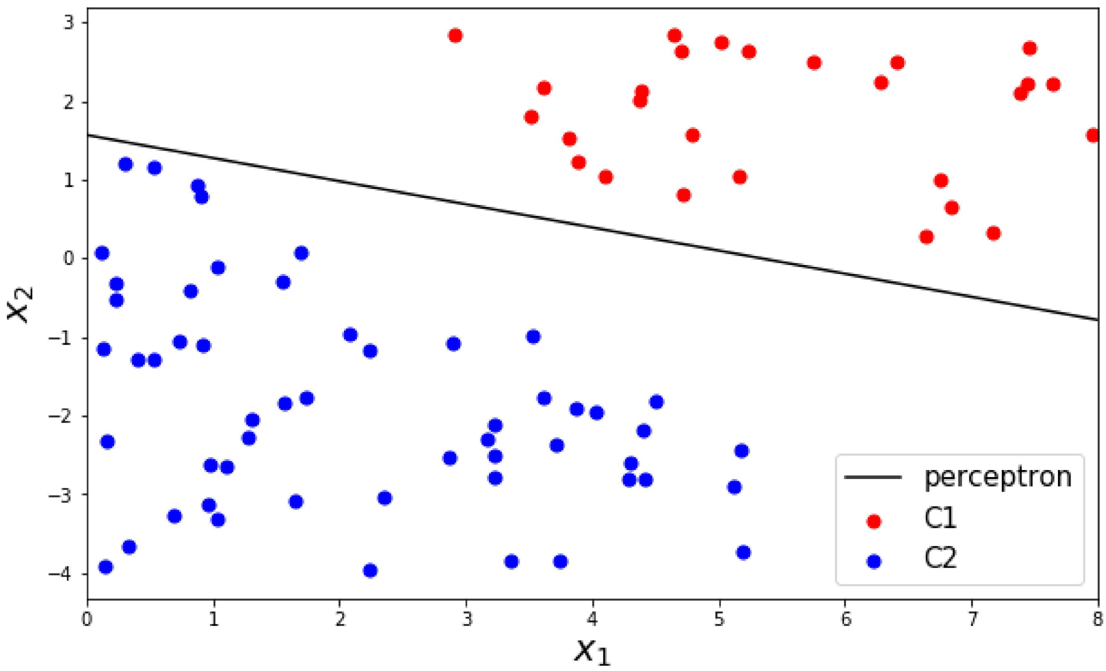
```
w = np.ones([3,1])
w = np.asmatrix(w)

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)
```

```
[[-11.        ]
 [  2.05983422]
 [  7.00620838]]
```

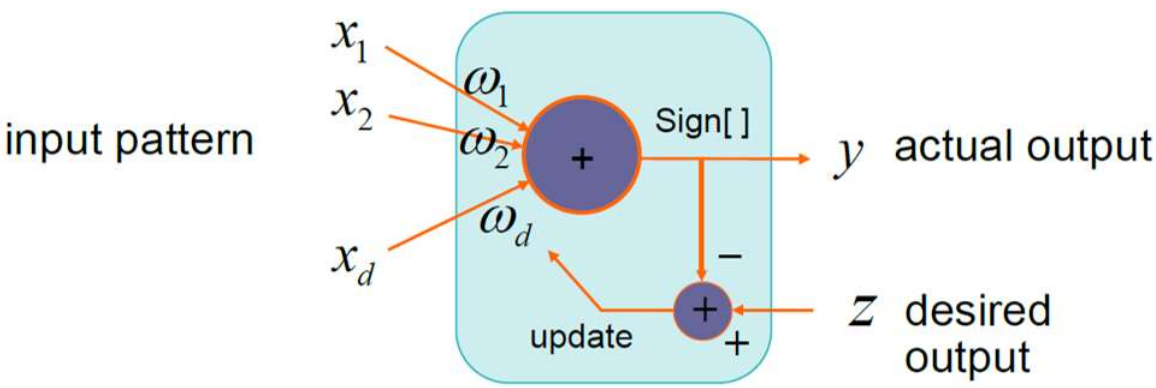
$$g(x) = \omega^T x + \omega_0 = \omega_1 x_1 + \omega_2 x_2 + \omega_0 = 0$$
$$\implies x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\omega_0}{\omega_2}$$

In [17]:

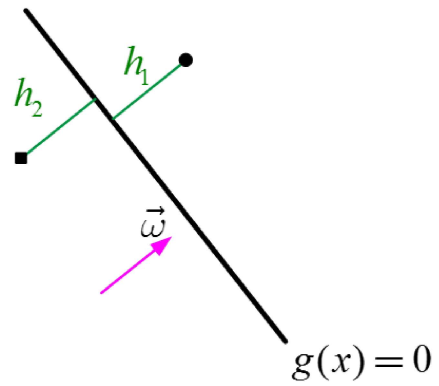
```
x1p = np.linspace(0,8,100).reshape(-1,1)
x2p = - w[1,0]/w[2,0]*x1p - w[0,0]/w[2,0]

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()
```

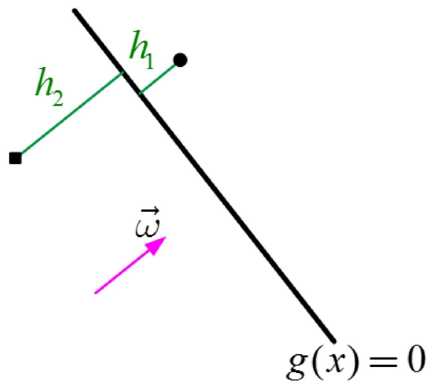


Perceptron



Using Distances



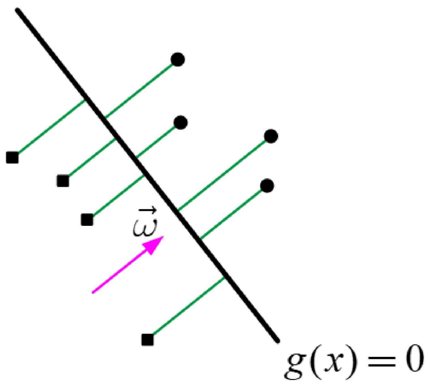


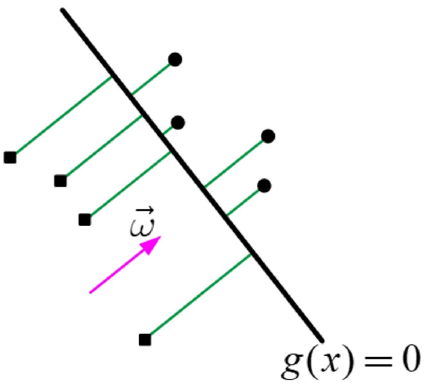
$$|h_1| + |h_2|$$
$$|h_1| \bullet |h_2|$$
$$\frac{|h_1| + |h_2|}{2} \geq \sqrt{|h_1| \bullet |h_2|}$$

$$|h_1| + |h_2|$$
$$|h_1| \bullet |h_2|$$
$$\text{equal iff } |h_1| = |h_2|$$

Using all Distances

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



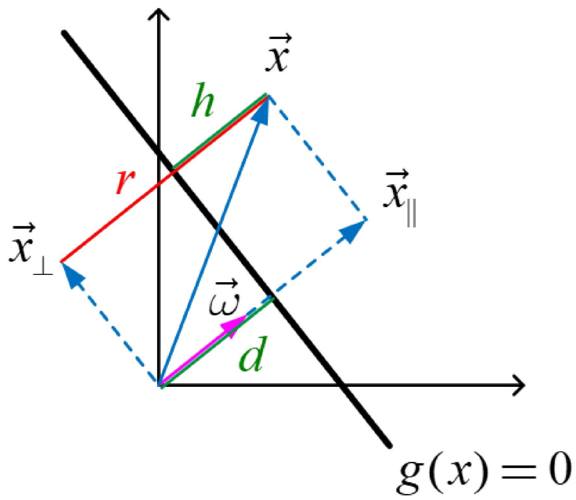


- Inequality of arithmetic and geometric means
$$\frac{x_1 + x_2 + \cdots + x_m}{m} \geq \sqrt[m]{x_1 \cdot x_2 \ldots x_m}$$
and that equality holds if and only if $x_1 = x_2 = \cdots = x_m$

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

Distance from a Line: h

- for any vector of x

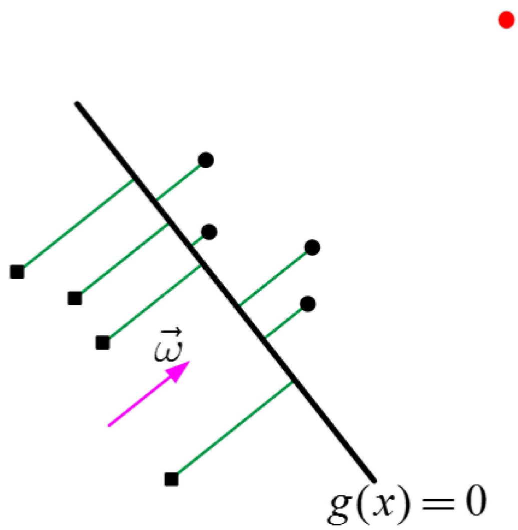


$$\therefore \quad h = \frac{g(x)}{\|\omega\|} \implies \text{orthogonal distance from the line}$$

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

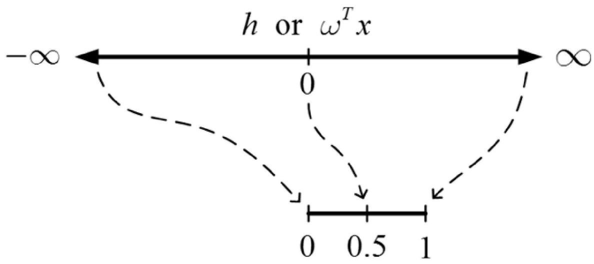
Using all Distances with Outliers

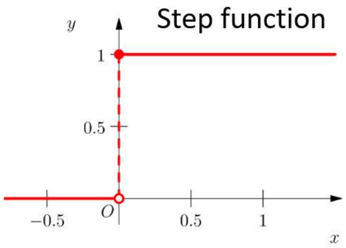
- Logistic Regression

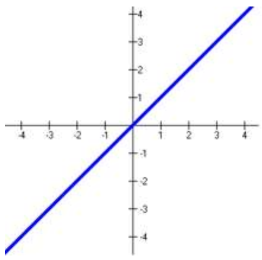


Sigmoid Function

- We link or squeeze (−∞,+∞) to (0, 1) for several reasons:



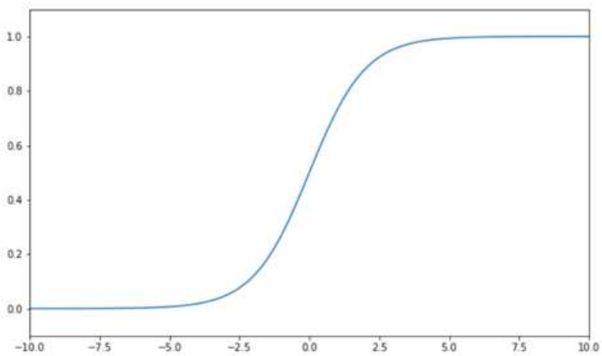




$\sigma(z)$ is the sigmoid function, or the logistic function

- logistic function always generates a value between 0 and 1
- Crosses 0.5 at the origin, then flattens out

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



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

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

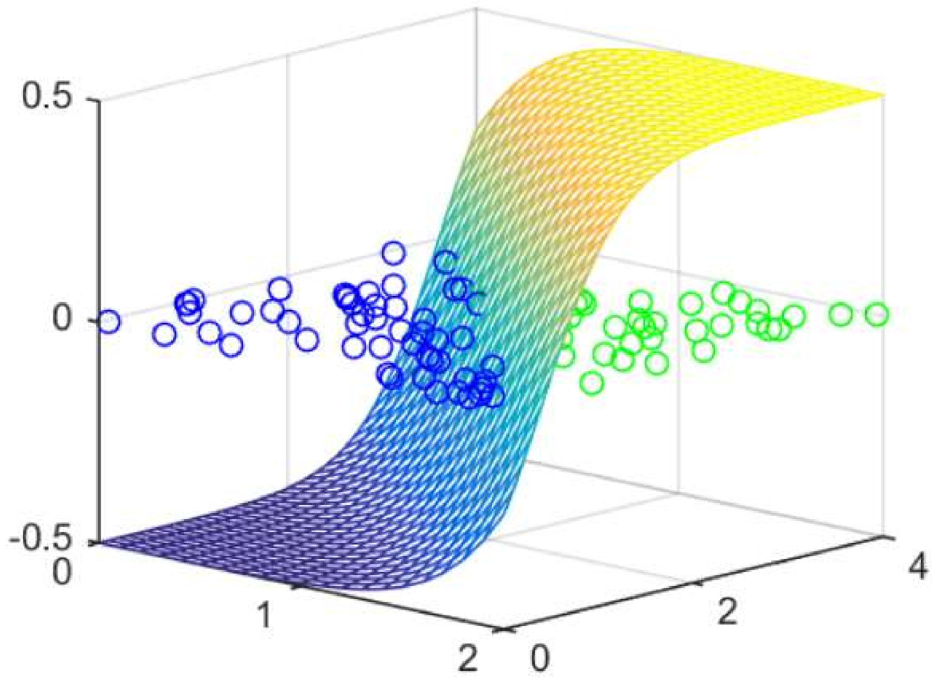
- Goal: we need to fit ω to our data

$$\max \prod_i |h_i|$$

- Again, it is an optimization problem

Logistic Regression

- Classified based on probability



3.1. Multiclass Classification: Softmax

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

- Using the soft-max function instead of the logistic function (refer to UFLDL Tutorial (http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/))
  - see them as probability

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

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

- Note: sigmoid function

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

4. Summary

- From parameter estimation of machine learning to optimization problems

Machine learning	Optimization
	Loss (or objective functions)
Regression	$\min\limits_{\theta_1,\theta_2} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$
Classification	$\ell(\omega) = \log \mathscr{L} = \log P(y \mid x,\omega) = \log \prod_{n=1}^{m} P(y_n \mid x_n,\omega) = \sum_{n=1}^{m} \log P(y_n \mid x_n,\omega)$

In [1]:

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