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

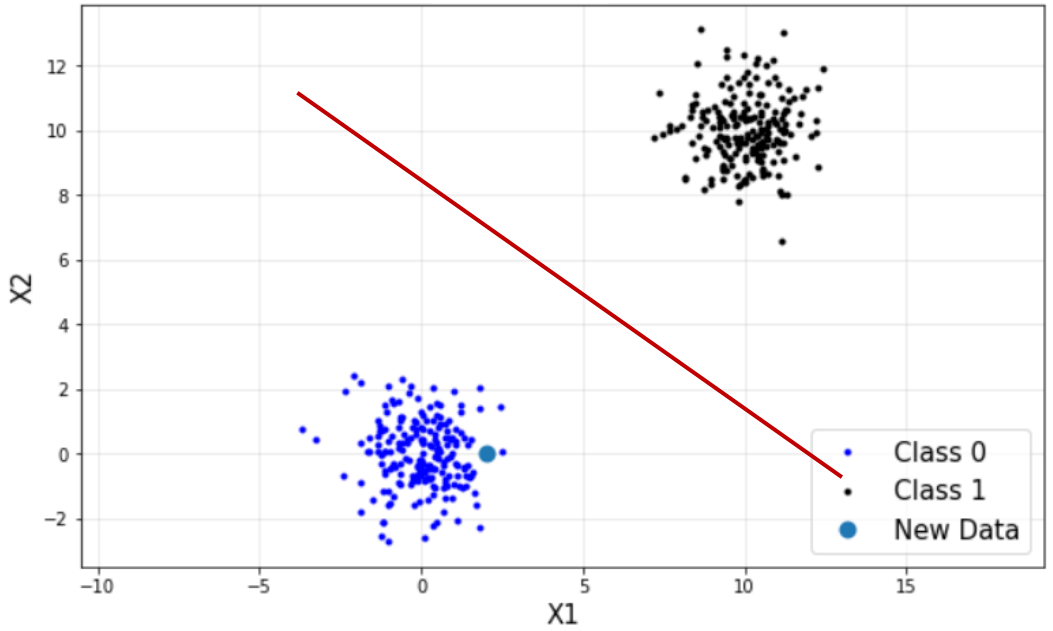
- We could use linear regression
  - Then, threshold the classifier output (i.e. anything over some value is yes, else no)
  - linear regression with thresholding seems to work

Classification

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

- To find

  a classification boundary

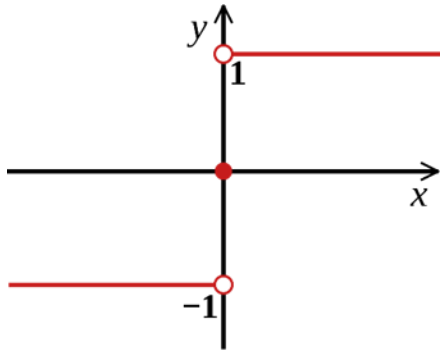


Perceptron

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

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

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



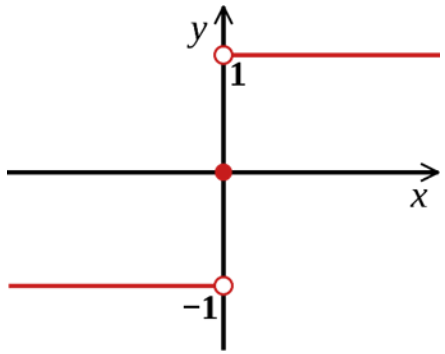
$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) - \text{threshold}\right) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) + \omega_0\right)$$

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 $x_0 = 1$:

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

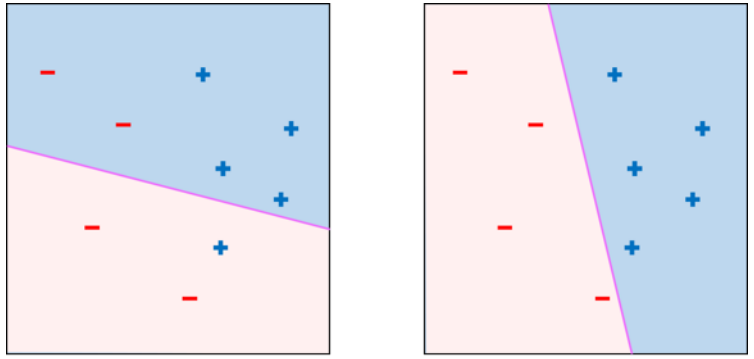


- In a vector form, the perceptron implements

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

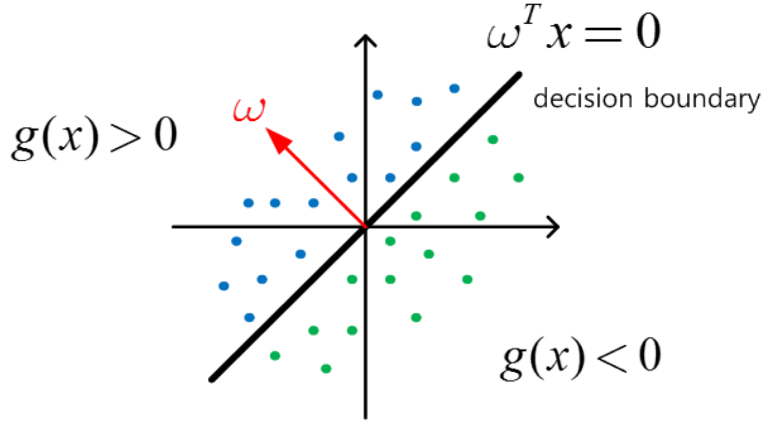
Perceptron

- Works for linearly separable data



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 $x_0 = 1$

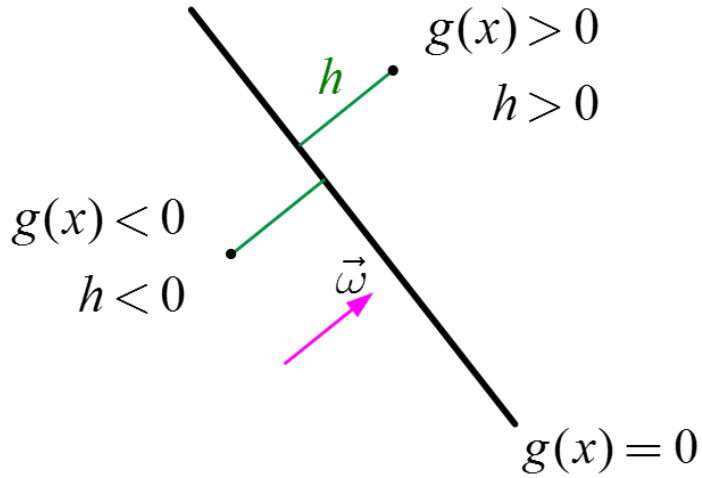


Sign

- Sign with respect to a line

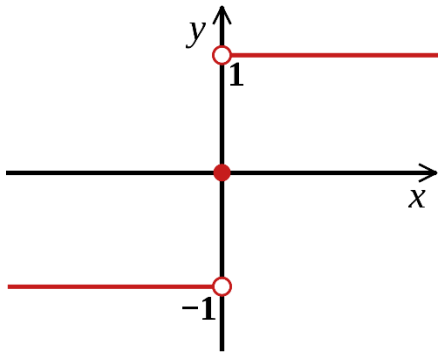
$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_1 x_1 + \omega_2 x_2 + \omega_0 = \omega^T x + \omega_0$$

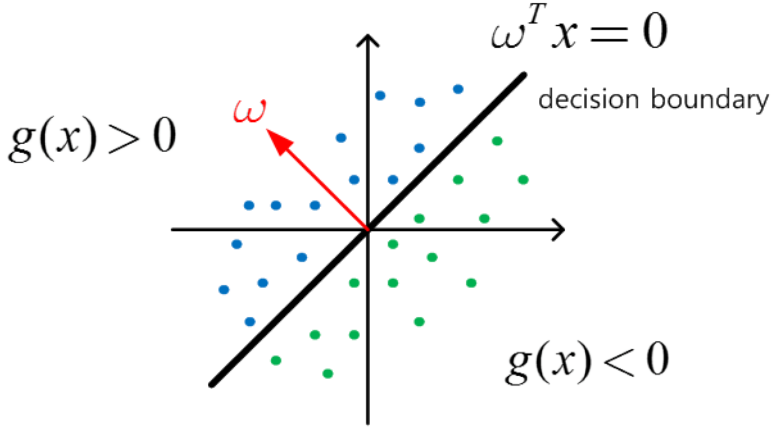
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = \omega^T x$$



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

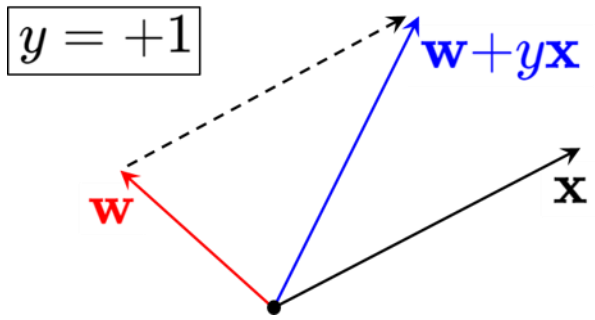
$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N) \quad \text{where } y_i \in \{-1, 1\}$$

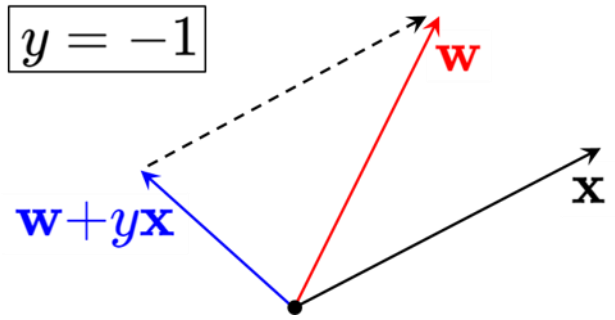
1) pick a misclassified point

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

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!

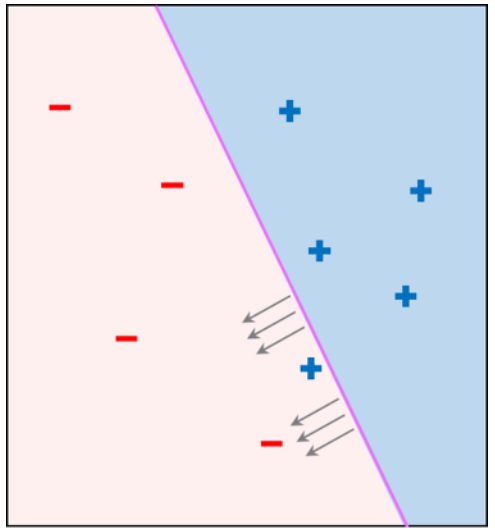
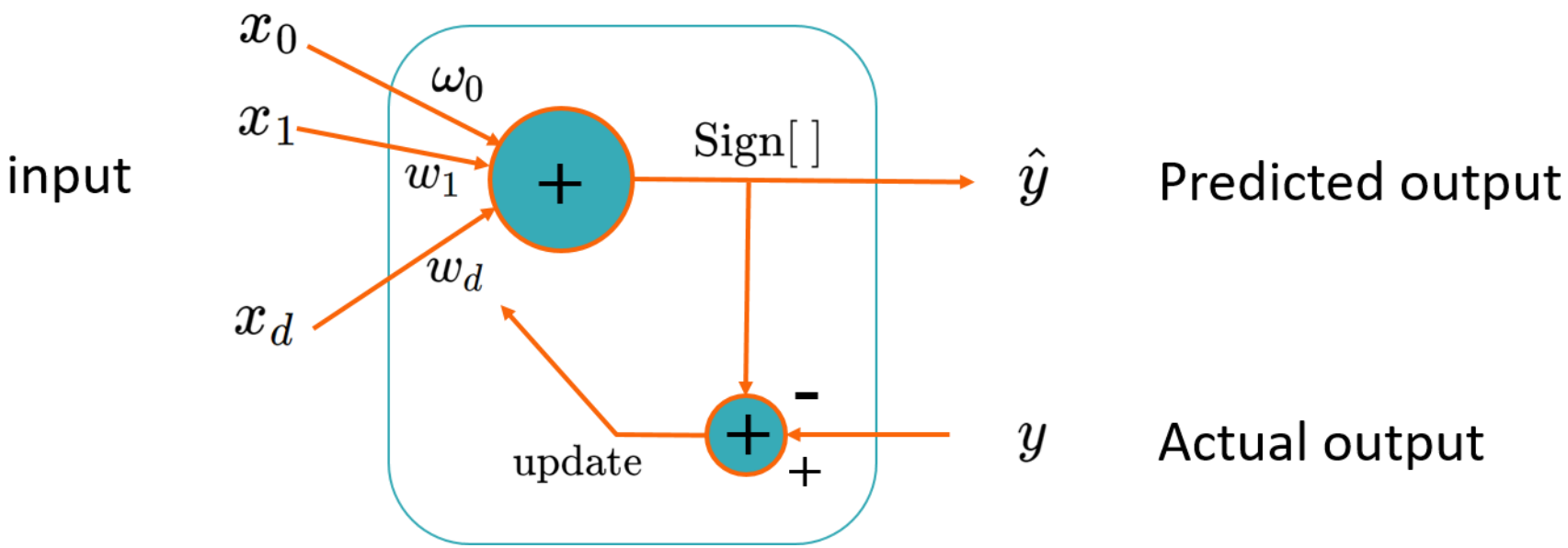


Diagram of Perceptron



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., $y_n \cdot \left(\omega^T x_n\right) > 0$

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

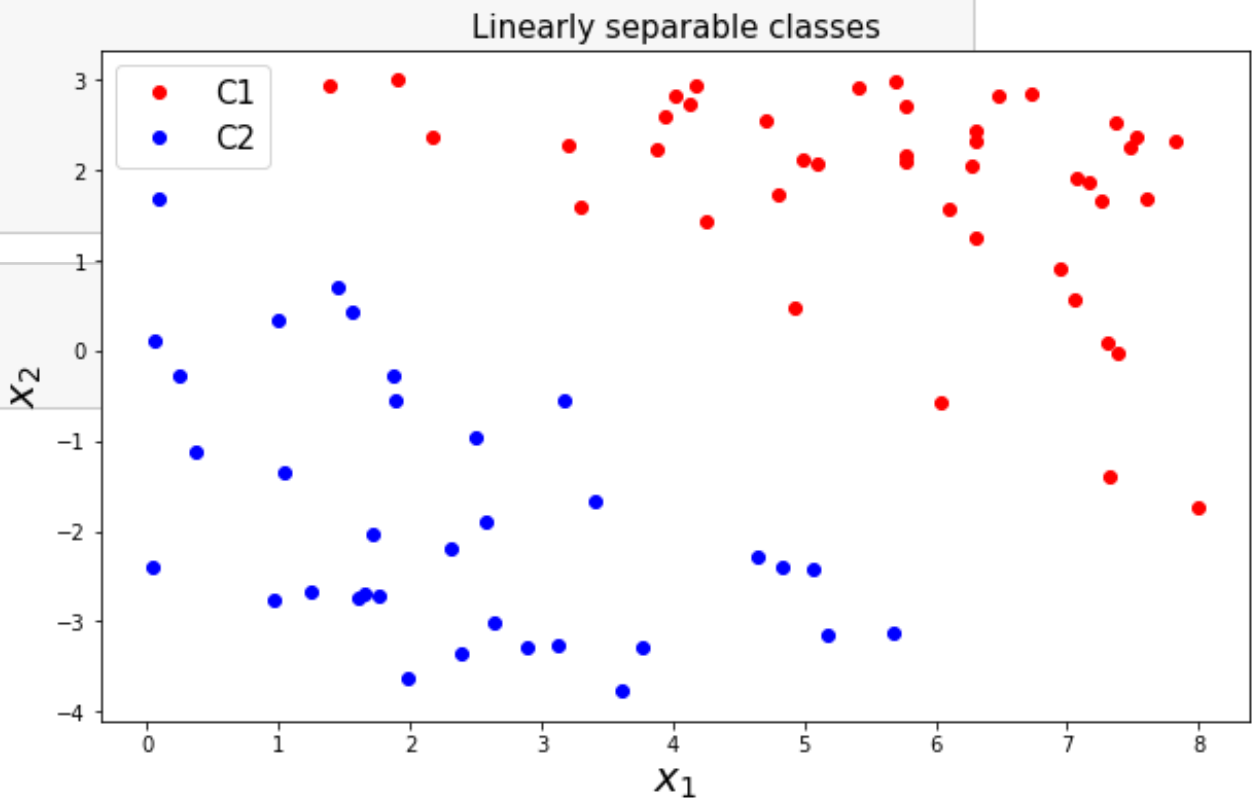
- Note:
  - $\text{sign}\left(\omega^T x_n\right) \neq y_n$ is equivalent to $y_n \cdot \left(\omega^T x_n\right) < 0$

Perceptron Algorithm in Python

```
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

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

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

```
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)
```

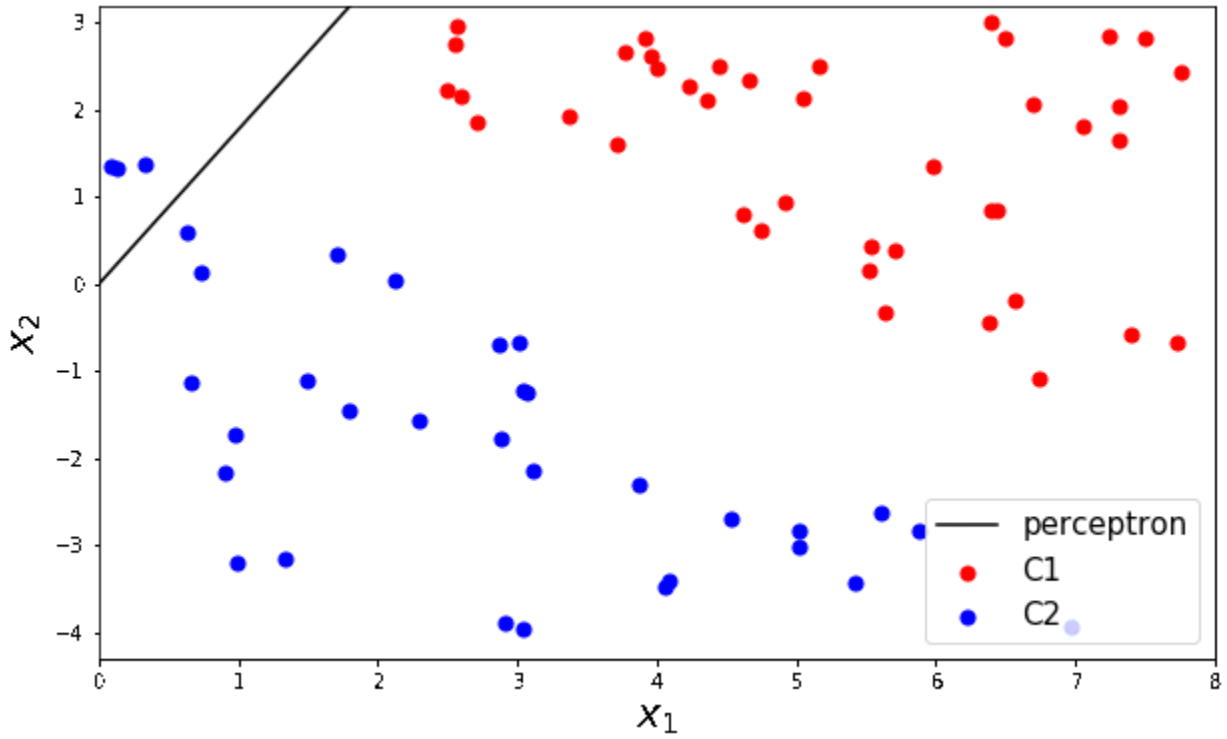
Perceptron Algorithm in Python

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

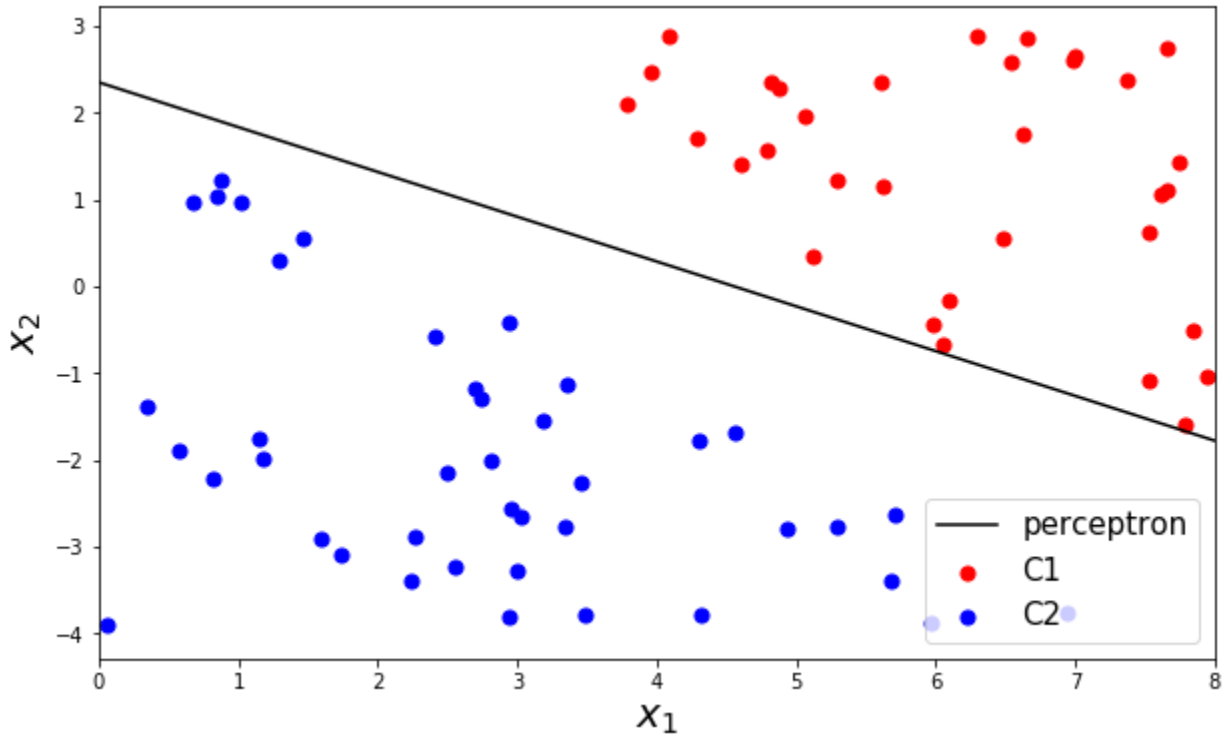
$$\Longrightarrow \quad x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\omega_0}{\omega_2}$$

```
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

Support Vector Machine

Industrial AI Lab.

Prof. Seungchul Lee

Classification (Linear)

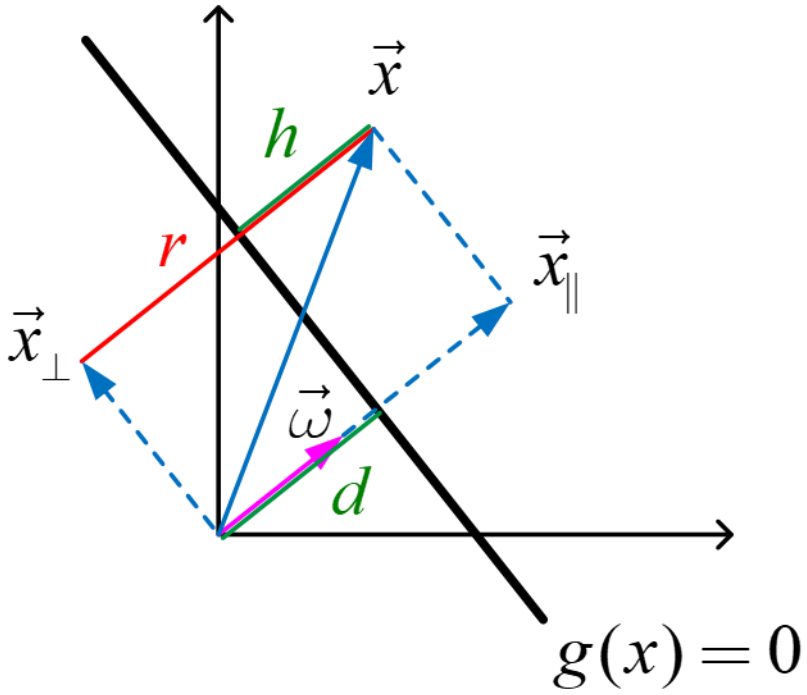
- Autonomously figure out which category (or class) an unknown item should be categorized into

- Number of categories / classes
  - Binary: 2 different classes
  - Multiclass: more than 2 classes

- Feature
  - The measurable parts that make up the unknown item (or the information you have available to categorize)

Distance from a Line

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

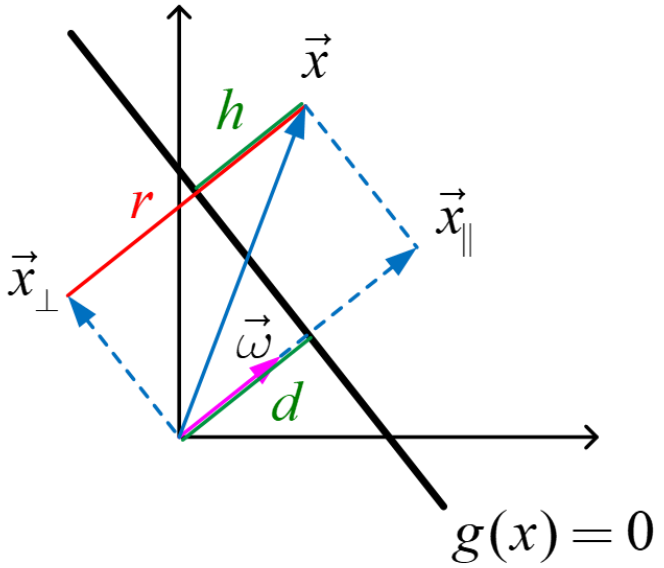


ω

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

$$g(\vec{p}) = g(\vec{q}) = 0 \quad \Rightarrow \quad \omega_0 + \omega^T \vec{p} = \omega_0 + \omega^T \vec{q} = 0$$
$$\Rightarrow \quad \omega^T (\vec{p} - \vec{q}) = 0$$

$\therefore$ ω : normal to the line (orthogonal)
$\Longrightarrow$ tells the direction of the line



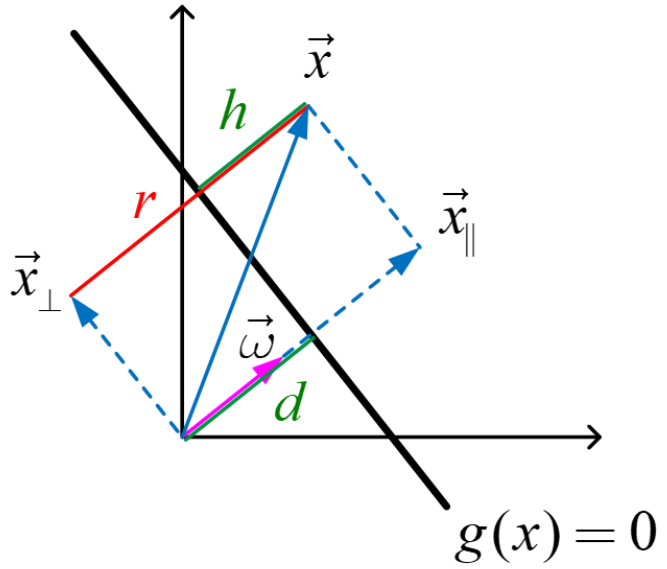
Signed Distance d

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

$$g(x) = \omega_0 + \omega^T x = 0$$

$$\Rightarrow \omega_0 + \omega^T d\frac{\omega}{\|\omega\|} = \omega_0 + d\frac{\omega^T\omega}{\|\omega\|} = \omega_0 + d\|\omega\| = 0$$

$$\therefore d = -\frac{\omega_0}{\|\omega\|}$$



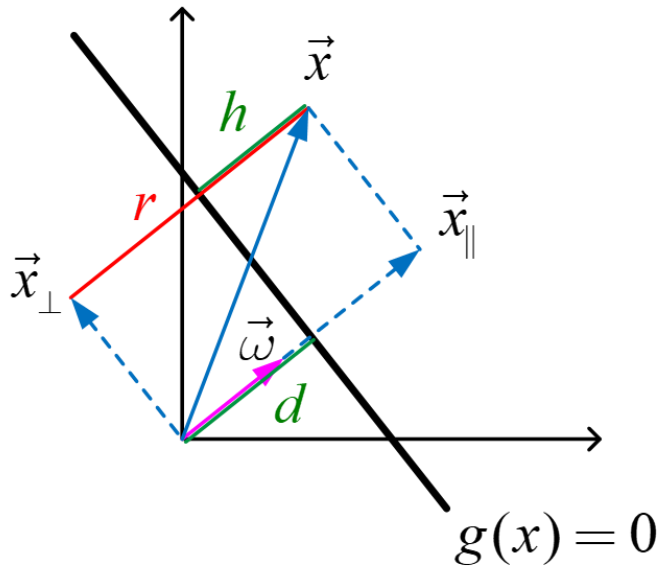
Distance from a Line: h

- for any vector of x

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

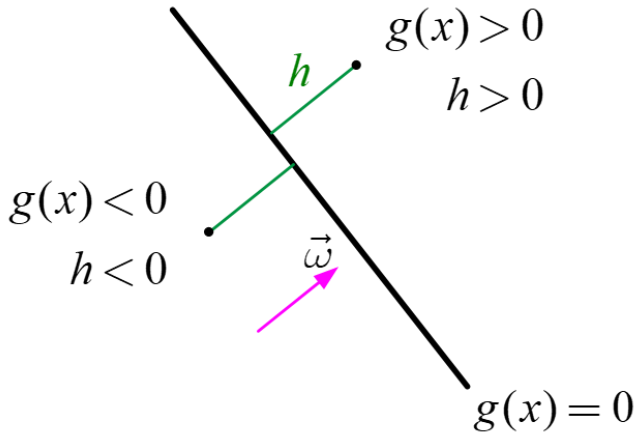
$$\omega^T x = \omega^T \left(x_\perp + r\frac{\omega}{\|\omega\|} \right) = r\frac{\omega^T\omega}{\|\omega\|} = r\|\omega\|$$

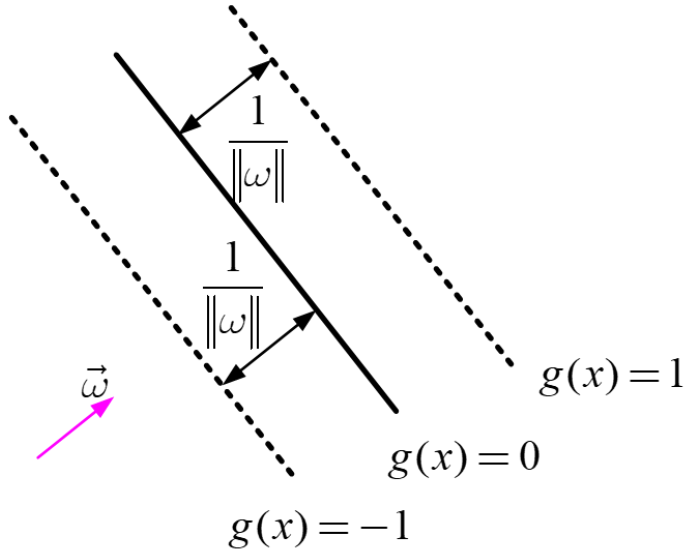
$$\begin{aligned} g(x) &= \omega_0 + \omega^T x \\ &= \omega_0 + r\|\omega\| \qquad (r = d + h) \\ &= \omega_0 + (d+h)\|\omega\| \\ &= \omega_0 + \left(-\frac{\omega_0}{\|\omega\|} + h \right) \|\omega\| \\ &= h\|\omega\| \end{aligned}$$



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

Distance from a Line: h

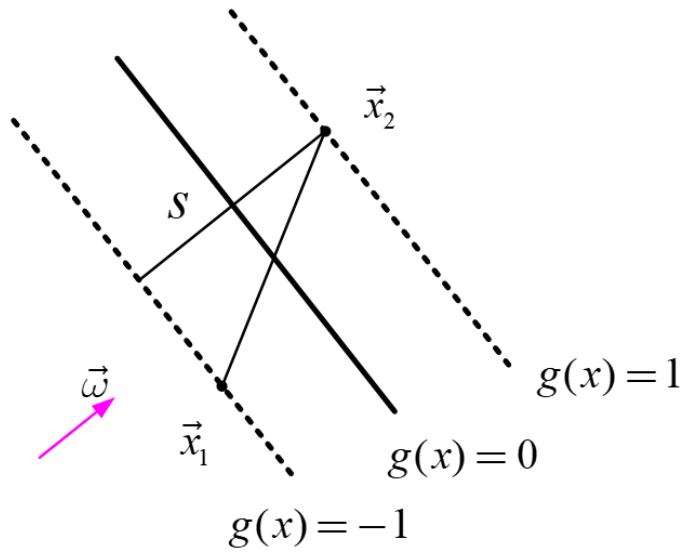




$$h = \frac{g(x)}{\|\omega\|}$$

Distance from a Line: h

- Another method to find a distance between $g(x) = 1$ and $g(x) = -1$
- Suppose $g(x_1) = -1$ and $g(x_2) = 1$



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

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

Illustrative Example

- Binary classification
  - C_1 and C_0
- Features
  - The coordinate of the unknown animal i in the zoo

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Hyperplane

- Is it possible to distinguish between C_1 and C_0 by its coordinates on a map of the zoo?

- We need to find a separating hyperplane (or a line in 2D)

$$\omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$$

$$\omega_0 + \begin{bmatrix} \omega_1 & \omega_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$\omega_0 + \omega^T x = 0$$

Decision Making

- Given:
  - Hyperplane defined by ω and ω_0
  - Animals coordinates (or features) x

- Decision making:

$$\omega_0 + \omega^T x > 0 \implies x \text{ belongs to } C_1$$
$$\omega_0 + \omega^T x < 0 \implies x \text{ belongs to } C_0$$

- Find ω and ω_0 such that x given $\omega_0 + \omega^T x = 0$

Decision Boundary or Band

- Find ω and ω_0 such that x given $\omega_0 + \omega^T x = 0$

or

- Find ω and ω_0 such that
  - $x \in C_1$ given $\omega_0 + \omega^T x > 1$ and
  - $x \in C_0$ given $\omega_0 + \omega^T x < -1$

$$\omega_0 + \omega^T x > b, \quad (b > 0)$$

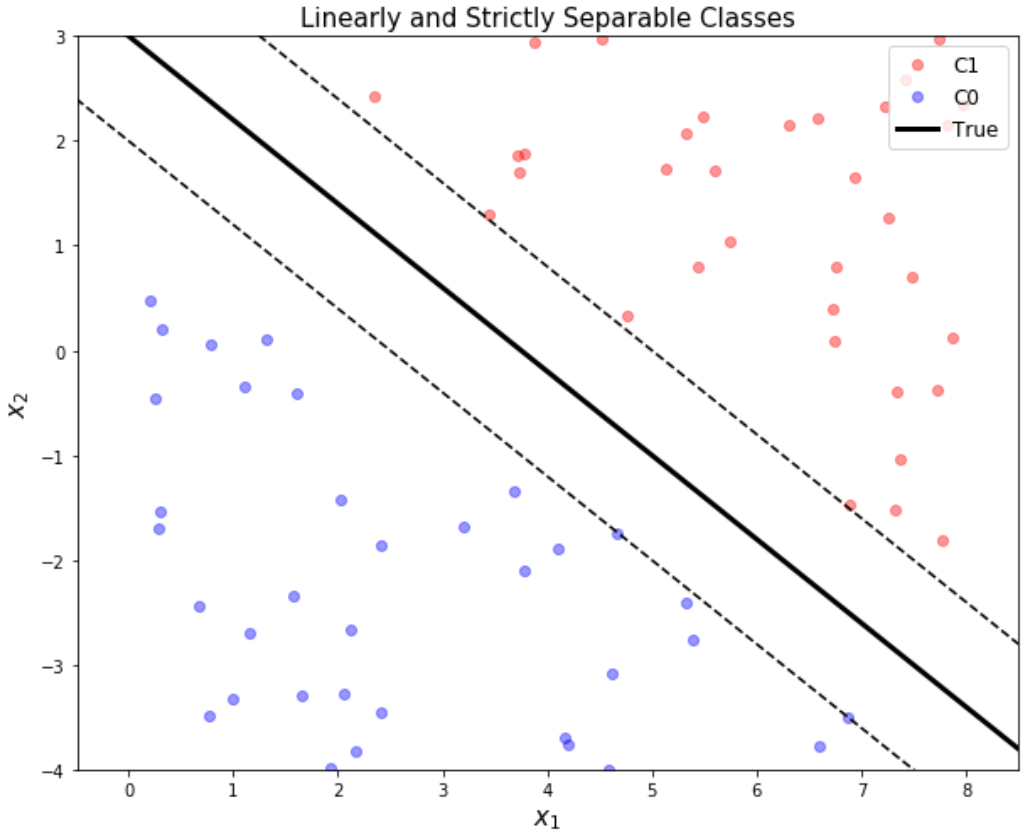
$$\iff \frac{\omega_0}{b} + \frac{\omega^T}{b} x > 1$$

$$\iff \omega_0' + \omega'^T x > 1$$

Data Generation for Classification

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

g = 0.8*x1 + x2 - 3
g1 = g - 1
g0 = g + 1

C1 = np.where(g1 >= 0)[0]
C0 = np.where(g0 < 0)[0]

xp = np.linspace(-1,9,100).reshape(-1,1)
ypt = -0.8*xp + 3
```

```
#  see how data are generated
xp = np.linspace(-1,9,100).reshape(-1,1)
ypt = -0.8*xp + 3

plt.figure(figsize=(10, 8))
plt.plot(x1[C1], x2[C1], 'ro', alpha = 0.4, label = 'C1')
plt.plot(x1[C0], x2[C0], 'bo', alpha = 0.4, label = 'C0')
plt.plot(xp, ypt, 'k', linewidth = 3, label = 'True')
plt.plot(xp, ypt-1, '--k')
plt.plot(xp, ypt+1, '--k')
plt.title('Linearly and Strictly Separable Classes', fontsize = 15)
plt.xlabel(r'$x_1$', fontsize = 15)
plt.ylabel(r'$x_2$', fontsize = 15)
plt.legend(loc = 1, fontsize = 12)
plt.axis('equal')
plt.xlim([0, 8])
plt.ylim([-4, 3])
plt.show()
```



Optimization Formulation 1

- $n\ (=2)$ features
- N belongs to C_1 in training set
- M belongs to C_0 in training set
- $m = N + M$ data points in training set

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \text{ with } \omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$$

or

$$x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \text{ with } \omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

- ω and ω_0 are the unknown variables

Optimization Formulation 1

$$\text{minimize} \quad \text{something}$$

$$\text{subject to} \quad \begin{cases} \omega_0 + \omega^T x^{(1)} \geq 1 \\ \omega_0 + \omega^T x^{(2)} \geq 1 \\ \quad \vdots \\ \omega_0 + \omega^T x^{(N)} \geq 1 \end{cases} \quad \begin{cases} \omega_0 + \omega^T x^{(N+1)} \leq -1 \\ \omega_0 + \omega^T x^{(N+2)} \leq -1 \\ \quad \vdots \\ \omega_0 + \omega^T x^{(N+M)} \leq -1 \end{cases}$$

$$\text{minimize} \quad \text{something}$$

$$\text{subject to} \quad \begin{cases} \omega^T x^{(1)} \geq 1 \\ \omega^T x^{(2)} \geq 1 \\ \quad \vdots \\ \omega^T x^{(N)} \geq 1 \end{cases} \quad \begin{cases} \omega^T x^{(N+1)} \leq -1 \\ \omega^T x^{(N+2)} \leq -1 \\ \quad \vdots \\ \omega^T x^{(N+M)} \leq -1 \end{cases}$$

CVXPY 1

$$\begin{aligned} \text{minimize} \quad & \text{something} \\ \text{subject to} \quad & X_1\omega \geq 1 \\ & X_0\omega \leq -1 \end{aligned}$$

$$X_1 = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(N)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} \end{bmatrix}$$

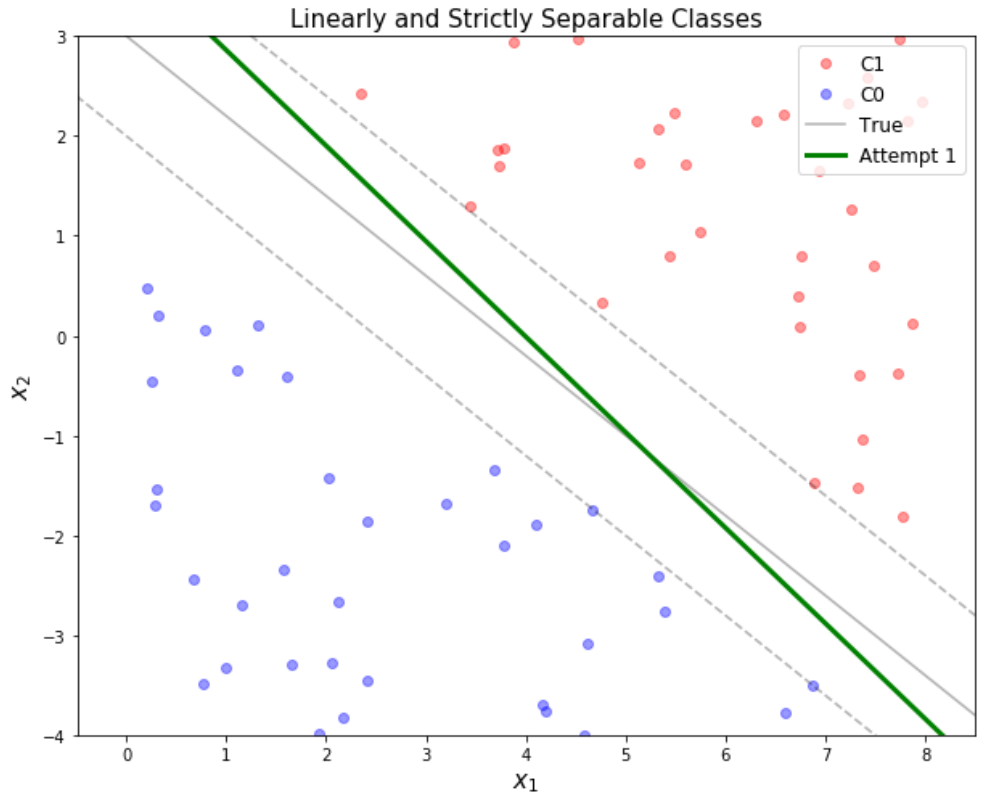
$$X_0 = \begin{bmatrix} \left(x^{(N+1)}\right)^T \\ \left(x^{(N+2)}\right)^T \\ \vdots \\ \left(x^{(N+M)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(N+1)} & x_2^{(N+1)} \\ 1 & x_1^{(N+2)} & x_2^{(N+2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N+M)} & x_2^{(N+M)} \end{bmatrix}$$

```
import cvxpy as cvx

N = C1.shape[0]
M = C0.shape[0]

X1 = np.hstack([np.ones([N,1]), x1[C1], x2[C1]])
X0 = np.hstack([np.ones([M,1]), x1[C0], x2[C0]])

X1 = np.asmatrix(X1)
X0 = np.asmatrix(X0)
```

```
w = cvx.Variable([3,1])

obj = cvx.Minimize(1)
const = [X1*w >= 1, X0*w <= -1]
prob = cvx.Problem(obj, const).solve()

w = w.value
```

CVXPY 1

$$\begin{aligned} \text{minimize} \quad & \text{something} \\ \text{subject to} \quad & X_1\omega \geq 1 \\ & X_0\omega \leq -1 \end{aligned}$$



Linear Classification: Outlier

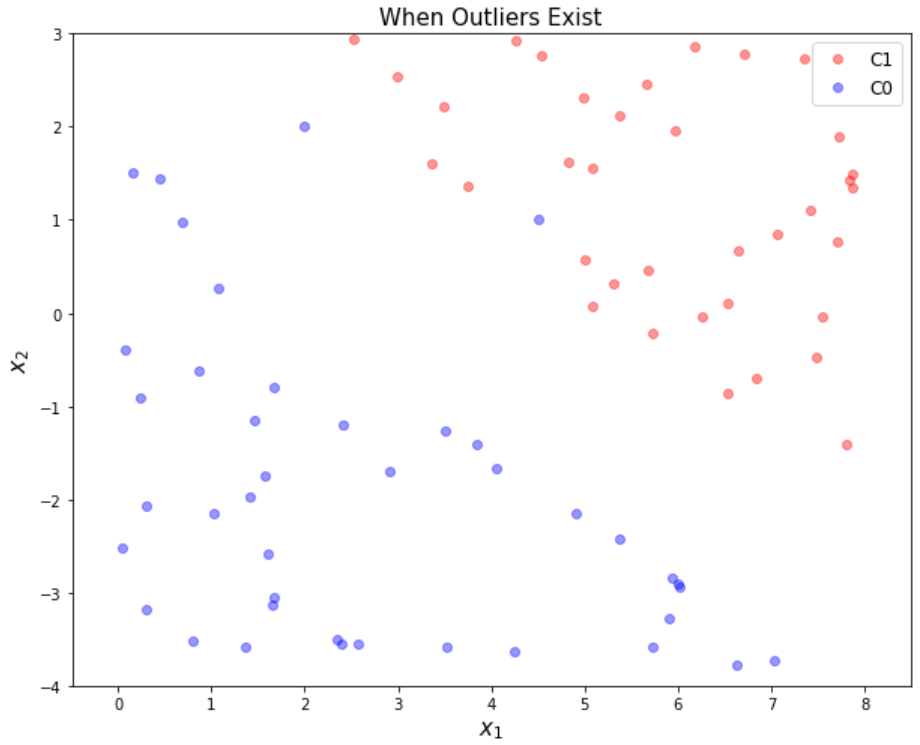
- Note that in the real world, you may have noise, errors, or outliers that do not accurately represent the actual phenomena

- Linearly non-separable case

Outliers

```
w = cvx.Variable([3,1])

obj = cvx.Minimize(1)
const = [X1*w >= 1, X0*w <= -1]
prob = cvx.Problem(obj, const).solve()

print(w.value)
```

None



- No solutions (hyperplane) exist

- We have to allow some training examples to be misclassified !
- but we want their number to be minimized

Optimization Formulation 2

- $n\ (=2)$ features
- N belongs to C_1 in training set
- M belongs to C_0 in training set
- $m = N + M$ data points in training set

$$x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \quad \text{with } \omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

$$\begin{aligned} \text{minimize} \quad & \text{something} \\ \text{subject to} \quad & X_1\omega \geq 1 \\ & X_0\omega \leq -1 \end{aligned}$$

- For the non-separable case, we relax the above constraints
- Need slack variables u and v where all are positive

Optimization Formulation 2

- The optimization problem for the non-separable case

$$\text{minimize} \quad \text{something}$$

$$\text{subject to} \quad \begin{cases} \omega^T x^{(1)} \geq 1 \\ \omega^T x^{(2)} \geq 1 \\ \quad \vdots \\ \omega^T x^{(N)} \geq 1 \end{cases} \quad \begin{cases} \omega^T x^{(N+1)} \leq -1 \\ \omega^T x^{(N+2)} \leq -1 \\ \quad \vdots \\ \omega^T x^{(N+M)} \leq -1 \end{cases}$$

$$\text{minimize} \quad \sum_{i=1}^{N} u_i + \sum_{i=1}^{M} v_i$$

$$\text{subject to} \quad \begin{cases} \omega^T x^{(1)} \geq 1 - u_1 \\ \omega^T x^{(2)} \geq 1 - u_2 \\ \quad \vdots \\ \omega^T x^{(N)} \geq 1 - u_N \end{cases} \quad \begin{cases} \omega^T x^{(N+1)} \leq -(1 - v_1) \\ \omega^T x^{(N+2)} \leq -(1 - v_2) \\ \quad \vdots \\ \omega^T x^{(N+M)} \leq -(1 - v_M) \end{cases}$$

$$\begin{cases} u \geq 0 \\ v \geq 0 \end{cases}$$

Expressed in a Matrix Form

$$X_1 = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(N)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} \end{bmatrix}$$

$$X_0 = \begin{bmatrix} \left(x^{(N+1)}\right)^T \\ \left(x^{(N+2)}\right)^T \\ \vdots \\ \left(x^{(N+M)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(N+1)} & x_2^{(N+1)} \\ 1 & x_1^{(N+2)} & x_2^{(N+2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N+M)} & x_2^{(N+M)} \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}$$

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix}$$

$$\begin{aligned} \text{minimize} \quad & 1^T u + 1^T v \\ \text{subject to} \quad & X_1 \omega \geq 1 - u \\ & X_0 \omega \leq -(1 - v) \\ & u \geq 0 \\ & v \geq 0 \end{aligned}$$

$$\text{minimize} \quad \sum_{i=1}^{N} u_i + \sum_{i=1}^{M} v_i$$

$$\text{subject to} \quad \begin{cases} \omega^T x^{(1)} \geq 1 - u_1 \\ \omega^T x^{(2)} \geq 1 - u_2 \\ \quad \vdots \\ \omega^T x^{(N)} \geq 1 - u_N \end{cases}$$

$$\begin{cases} \omega^T x^{(N+1)} \leq -(1 - v_1) \\ \omega^T x^{(N+2)} \leq -(1 - v_2) \\ \quad \vdots \\ \omega^T x^{(N+M)} \leq -(1 - v_M) \end{cases}$$

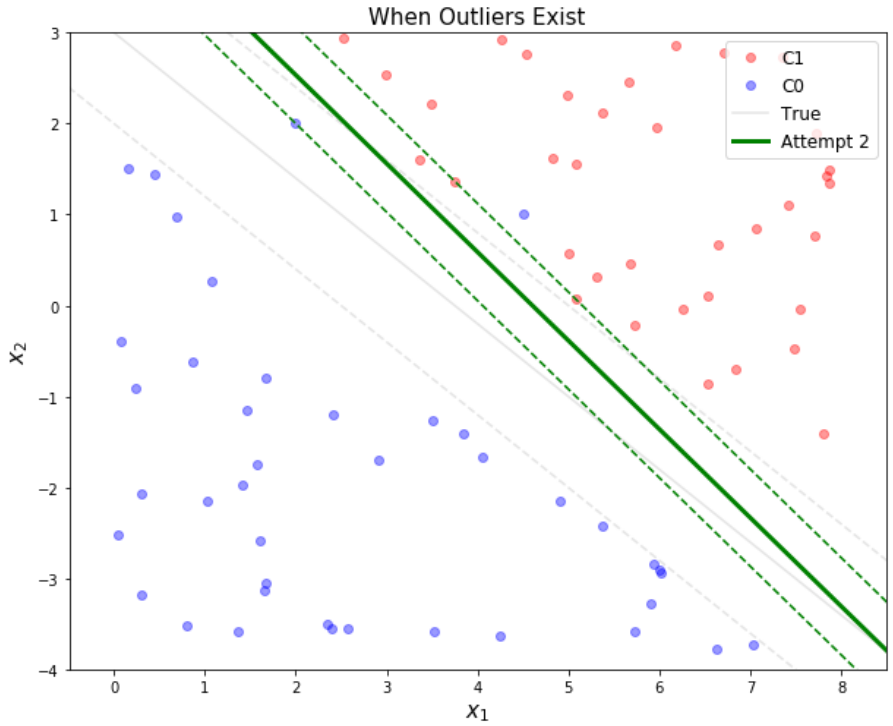
$$\begin{cases} u \geq 0 \\ v \geq 0 \end{cases}$$

CVXPY 2

$$\begin{aligned} \text{minimize} \quad & \text{something} \\ \text{subject to} \quad & X_1\omega \geq 1 \\ & X_0\omega \leq -1 \end{aligned}$$

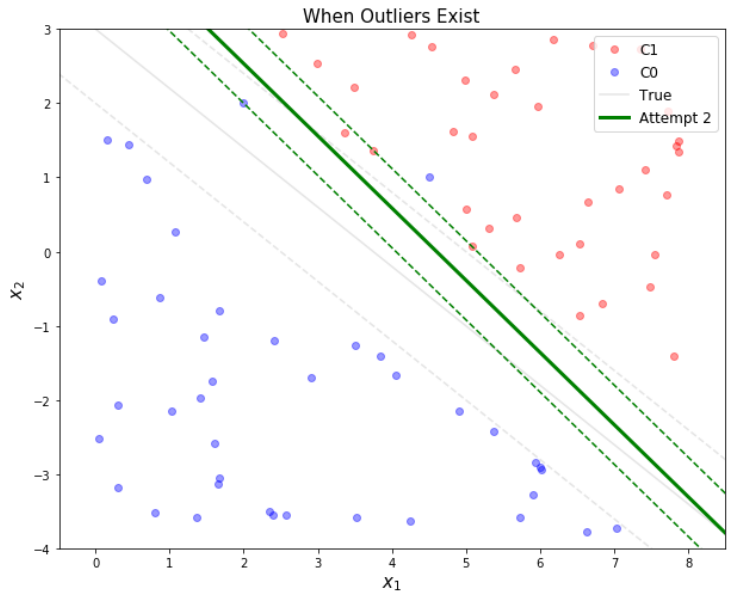
$$\begin{aligned} \text{minimize} \quad & 1^T u + 1^T v \\ \text{subject to} \quad & X_1\omega \geq 1 - u \\ & X_0\omega \leq -(1 - v) \\ & u \geq 0 \\ & v \geq 0 \end{aligned}$$

```
w = cvx.Variable([3,1])
u = cvx.Variable([N,1])
v = cvx.Variable([M,1])

obj = cvx.Minimize(np.ones((1,N))*u + np.ones((1,M))*v)
const = [X1*w >= 1-u, X0*w <= -(1-v), u >= 0, v >= 0 ]
prob = cvx.Problem(obj, const).solve()

w = w.value
```



Further Improvement

- Notice that hyperplane is not as accurately represent the division due to the outlier



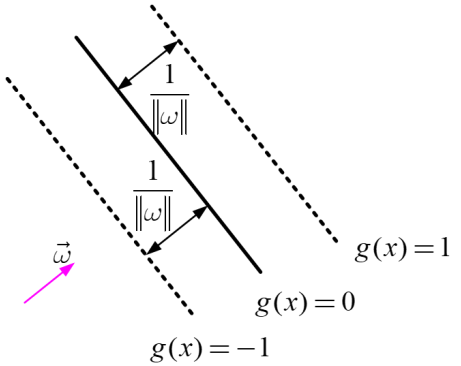
- Can we do better when there are noise data or outliers?
- Yes, but we need to look beyond linear programming

- Idea: large margin leads to good generalization on the test data

Maximize Margin

- Finally, it is Support Vector Machine (SVM)
- Distance (= margin)

$$\text{margin} = \frac{2}{\|\omega\|_2}$$



- Minimize $\|\omega\|_2$ to maximize the margin (closest samples from the decision line)

$$\text{maximize } \{\text{minimum distance}\}$$

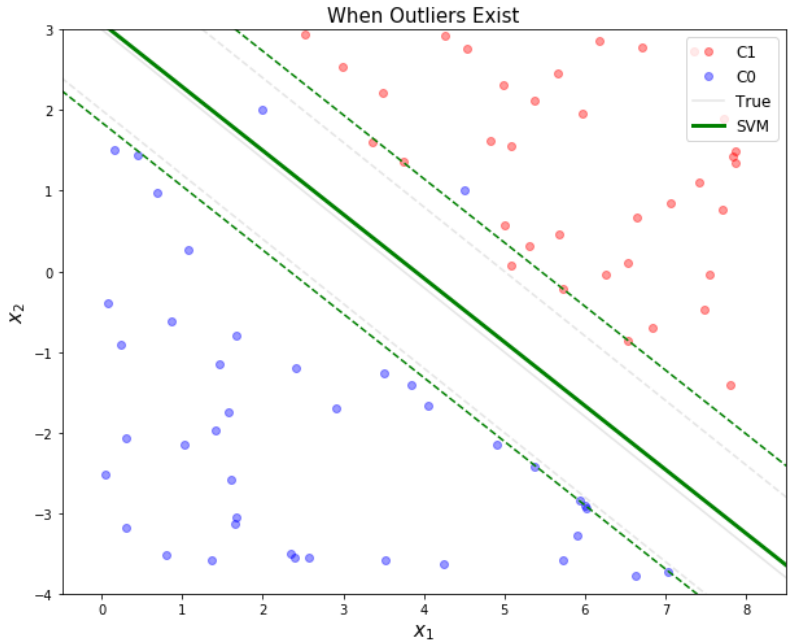
- Use gamma (γ) as a weighting between the followings:
  - Bigger margin given robustness to outliers
  - Hyperplane that has few (or no) errors

Support Vector Machine

$$
\begin{aligned}
\text{minimize} \quad & 1^T u + 1^T v \\
\text{subject to} \quad & X_1 \omega \geq 1 - u \\
& X_0 \omega \leq -(1 - v) \\
& u \geq 0 \\
& v \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\text{minimize} \quad & \|\omega\|_2 + \gamma(1^T u + 1^T v) \\
\text{subject to} \quad & X_1 \omega \geq 1 - u \\
& X_0 \omega \leq -(1 - v) \\
& u \geq 0 \\
& v \geq 0
\end{aligned}
$$

```
g = 3

w = cvx.Variable([3,1])
u = cvx.Variable([N,1])
v = cvx.Variable([M,1])

obj = cvx.Minimize(cvx.norm(w,2) + g*(np.ones((1,N))*u + np.ones((1,M))*v))
const = [X1*w >= 1-u, X0*w <= -(1-v), u >= 0, v >= 0 ]
prob = cvx.Problem(obj, const).solve()

w = w.value
```



Support Vector Machine

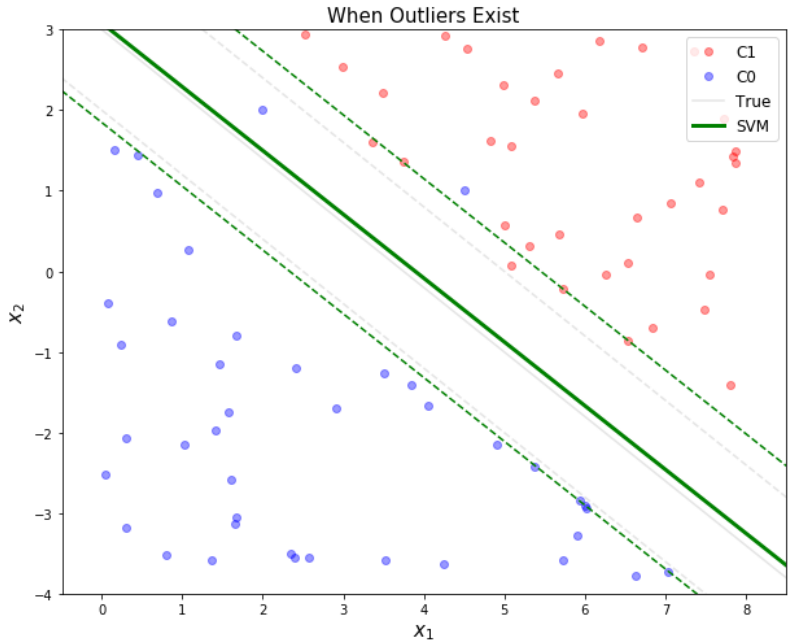
- In a more compact form

$$\begin{aligned} &\text{minimize} && \|\omega\|_2 + \gamma(1^T u + 1^T v) \\ &\text{subject to} && X_1\omega \geq 1 - u \\ &&& X_0\omega \leq -(1 - v) \\ &&& u \geq 0 \\ &&& v \geq 0 \end{aligned}$$

$$\omega^T x_n \geq 1 \text{ for } y_n = +1$$
$$\omega^T x_n \leq -1 \text{ for } y_n = -1$$
$$\iff y_n \cdot (\omega^T x_n) \geq 1$$

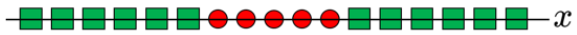
$$\begin{aligned} &\text{minimize} && \|\omega\|_2 + \gamma(1^T \xi) \\ &\text{subject to} && y_n \cdot (\omega^T x_n) \geq 1 - \xi_n \\ &&& \xi \geq 0 \end{aligned}$$

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

m = N + M

w = cvx.Variable([3,1])
d = cvx.Variable([m,1])

obj = cvx.Minimize(cvx.norm(w,2) + g*(np.ones([1,m])*d))
const = [cvx.multiply(y, X*w) >= 1-d, d >= 0]
prob = cvx.Problem(obj, const).solve()

w = w.value
```



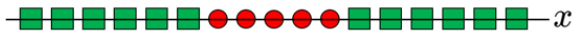
Classifying Non-linear Separable Data

- Consider the binary classification problem
  - each example represented by a single feature x
  - No linear separator exists for this data

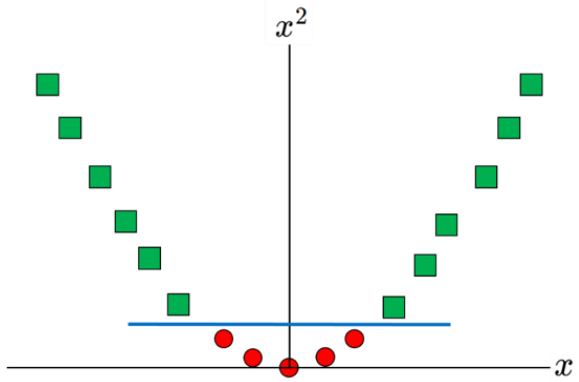


Classifying Non-linear Separable Data

- Consider the binary classification problem
  - each example represented by a single feature x
  - No linear separator exists for this data



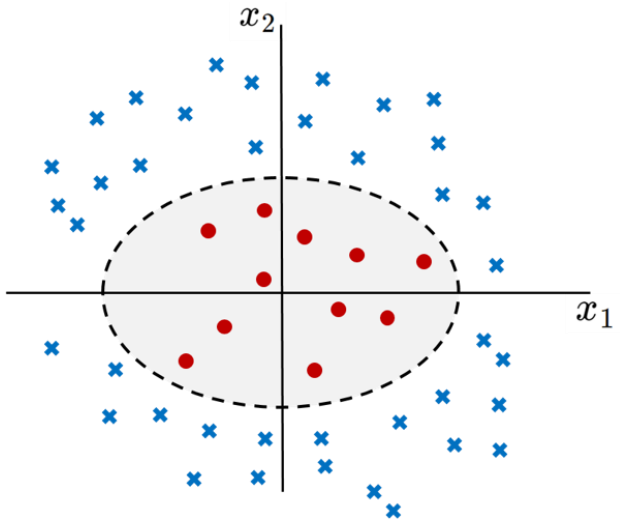
- Now map each example as $x \rightarrow \{x, x^2\}$
- Data now becomes linearly separable in the new representation

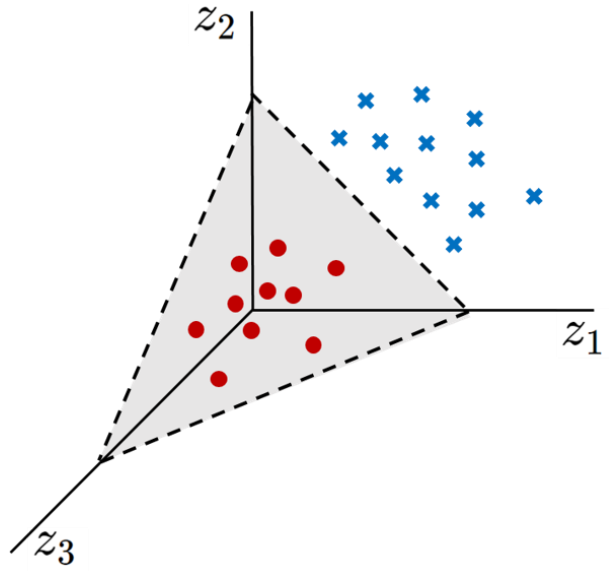


- Linear in the new representation = nonlinear in the old representation

Classifying Non-linear Separable Data

- Let's look at another example
  - Each example defined by a two features
  - No linear separator exists for this data $x = \{x_1, x_2\}$





- Now map each example as $x = \{x_1, x_2\} \rightarrow z = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$
  - Each example now has three features (derived from the old representation)
- Data now becomes linear separable in the new representation

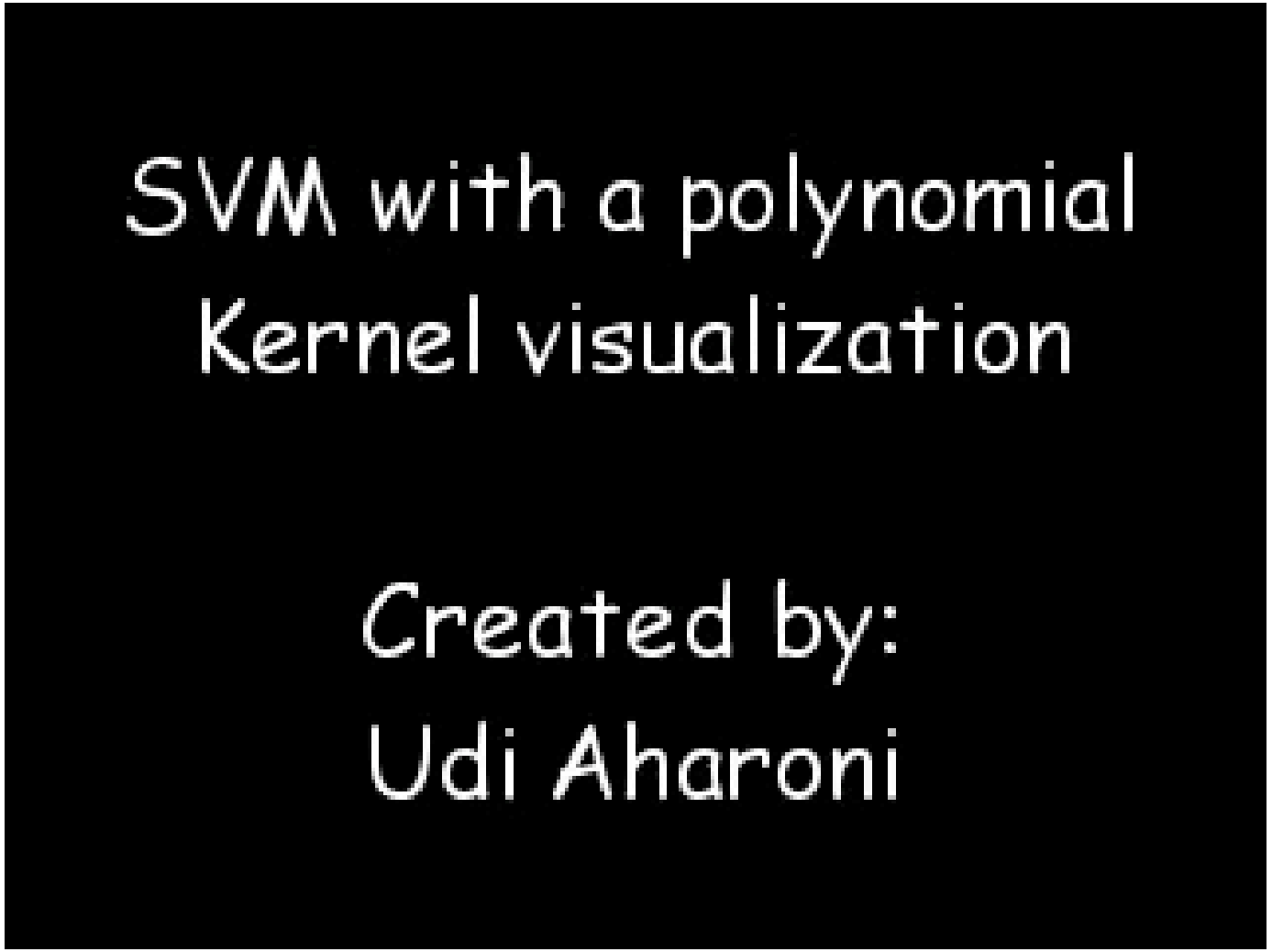
Kernel

- Often we want to capture nonlinear patterns in the data
  - nonlinear regression: input and output relationship may not be linear
  - nonlinear classification: classes may note be separable by a linear boundary

- Linear models (e.g. linear regression, linear SVM) are not just rich enough
  - by mapping data to higher dimensions where it exhibits linear patterns
  - apply the linear model in the new input feature space
  - mapping = changing the feature representation

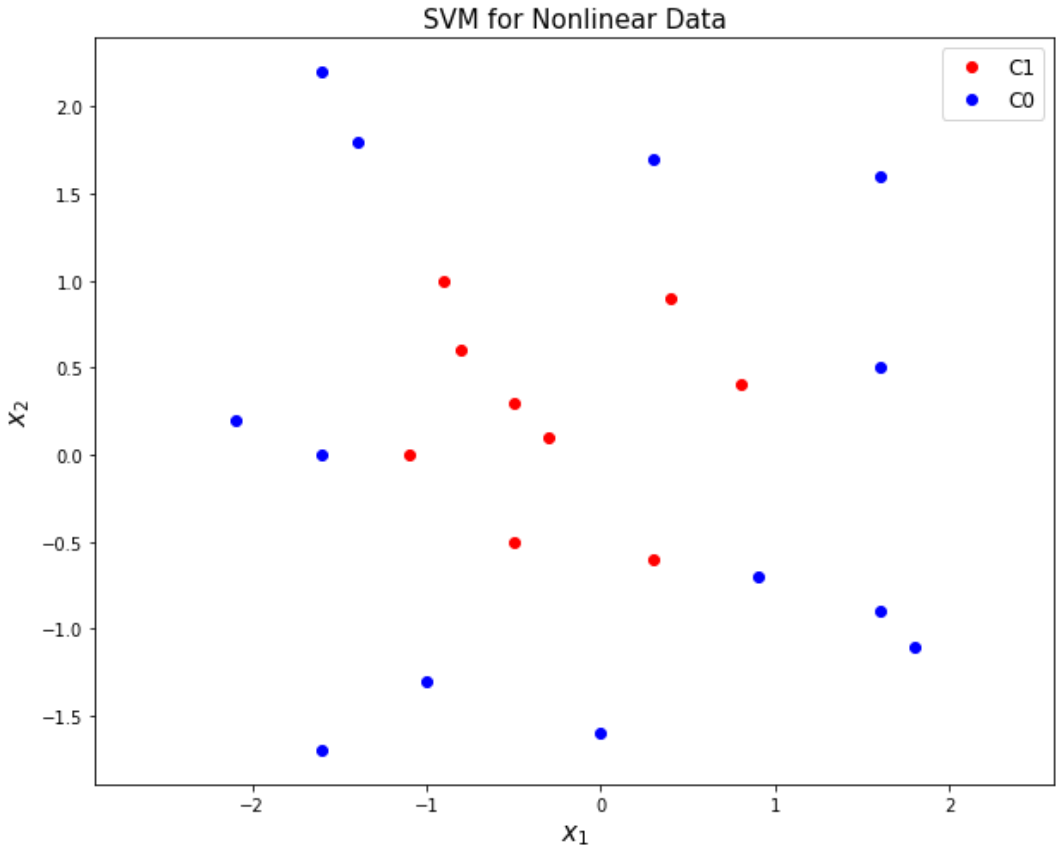
- Kernels: make linear model work in nonlinear settings

Nonlinear Classification



https://www.youtube.com/watch?v=3liCbRZPrZA

Classifying Non-linear Separable Data



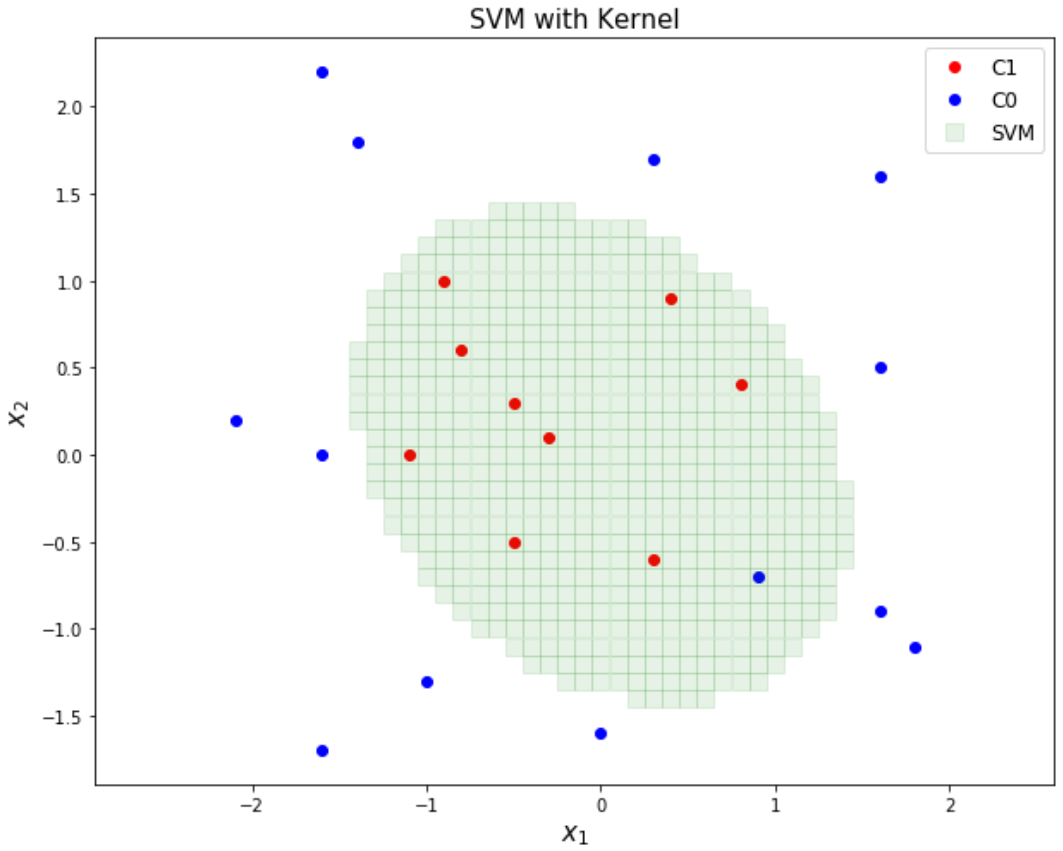
Classifying Non-linear Separable Data

```
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.square(X[:,0]), np.sqrt(2)*np.multiply(X[:,0],X[:,1]), np.square(X[:,1])])

g = 10

w = cvx.Variable([4, 1])
d = cvx.Variable([m, 1])

obj = cvx.Minimize(cvx.norm(w, 2) + g*np.ones([1,m])*d)
const = [cvx.multiply(y, Z*w) >= 1-d, d>=0]
prob = cvx.Problem(obj, const).solve()

w = w.value
print(w)
```

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

$$\begin{aligned} \text{minimize} \quad & \|\omega\|_2 + \gamma(1^T\xi) \\ \text{subject to} \quad & y_n \cdot \left(\omega^T x_n\right) \geq 1 - \xi_n \\ & \xi \geq 0 \end{aligned}$$

Classifying Non-linear Separable Data



Logistic Regression

Industrial AI Lab.

Prof. Seungchul Lee

Linear Classification: Logistic Regression

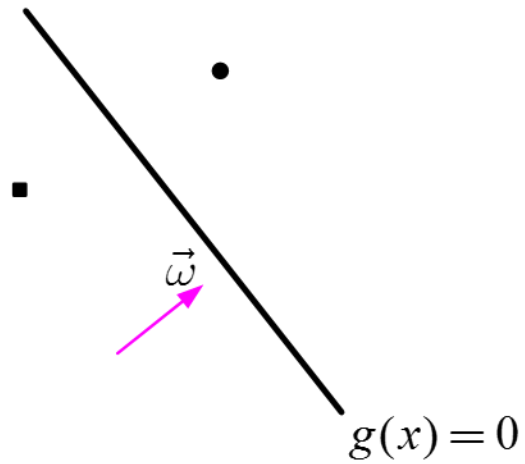
- Logistic regression is a classification algorithm
  - don't be confused

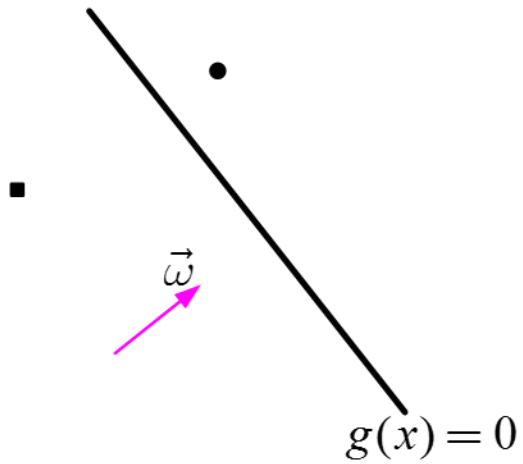
- Perceptron: make use of sign of data

- SVM: make use of margin (minimum distance)
  - Distance from two closest data points

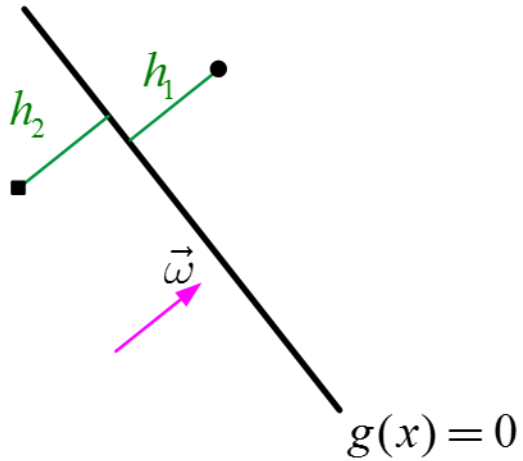
- We want to use distance information of all data points
  - logistic regression

Using Distances

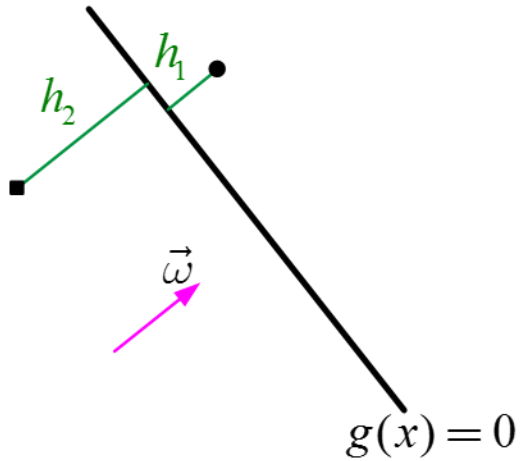




Using Distances

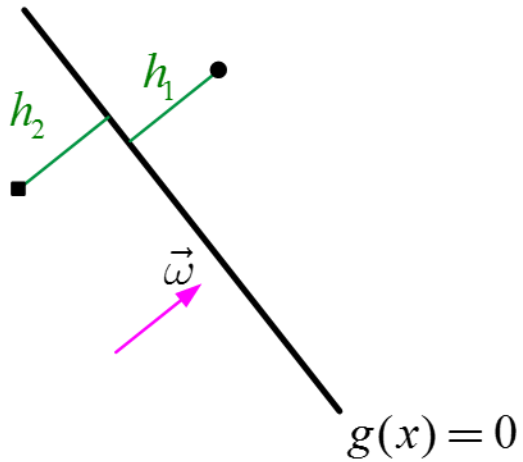


$$|h_1| + |h_2|$$



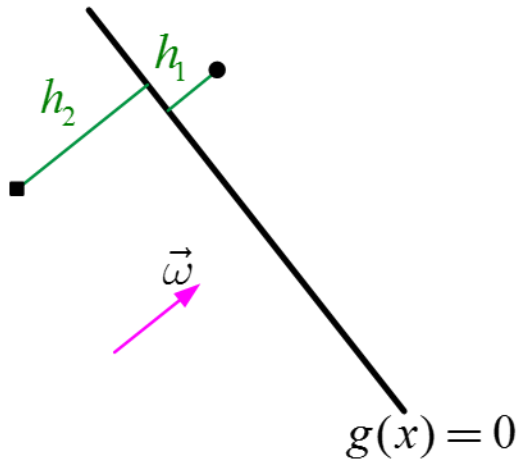
$$|h_1| + |h_2|$$

Using Distances



$$|h_1| + |h_2|$$

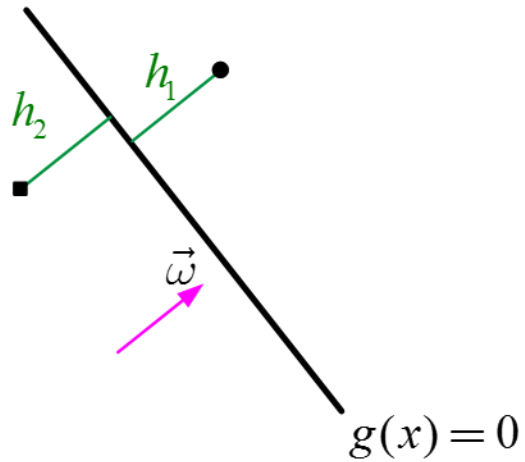
$$|h_1| \cdot |h_2|$$

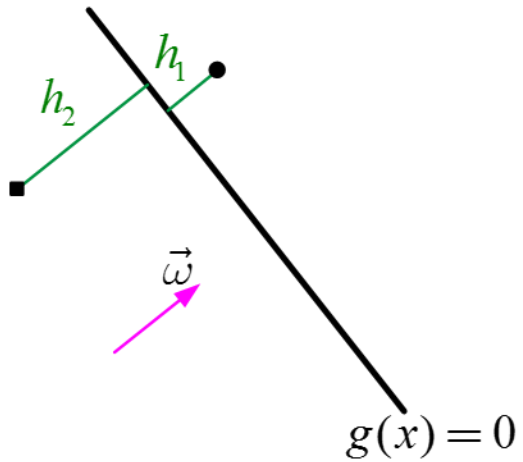


$$|h_1| + |h_2|$$

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

Using Distances





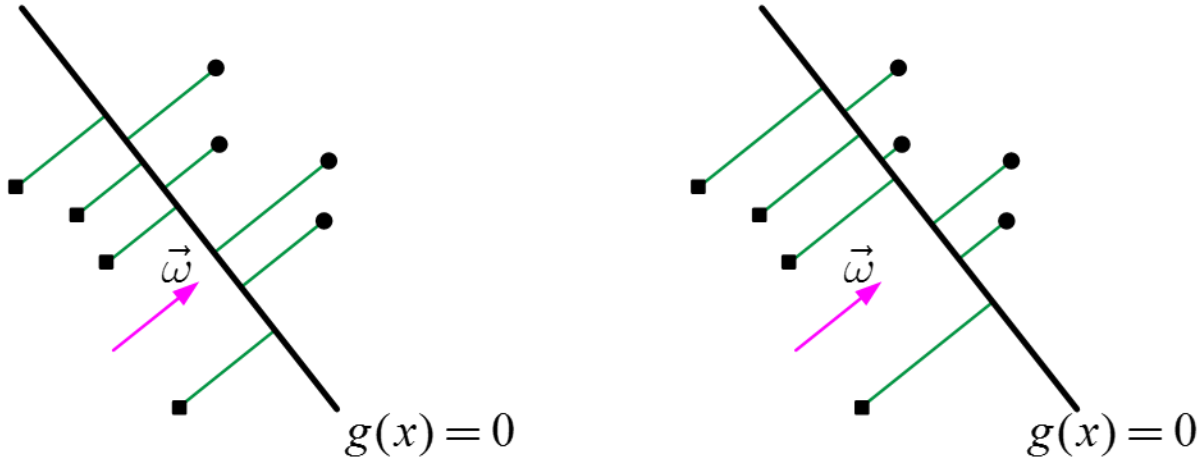
$$|h_1| + |h_2| \qquad |h_1| + |h_2|$$

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

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

Using all Distances

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

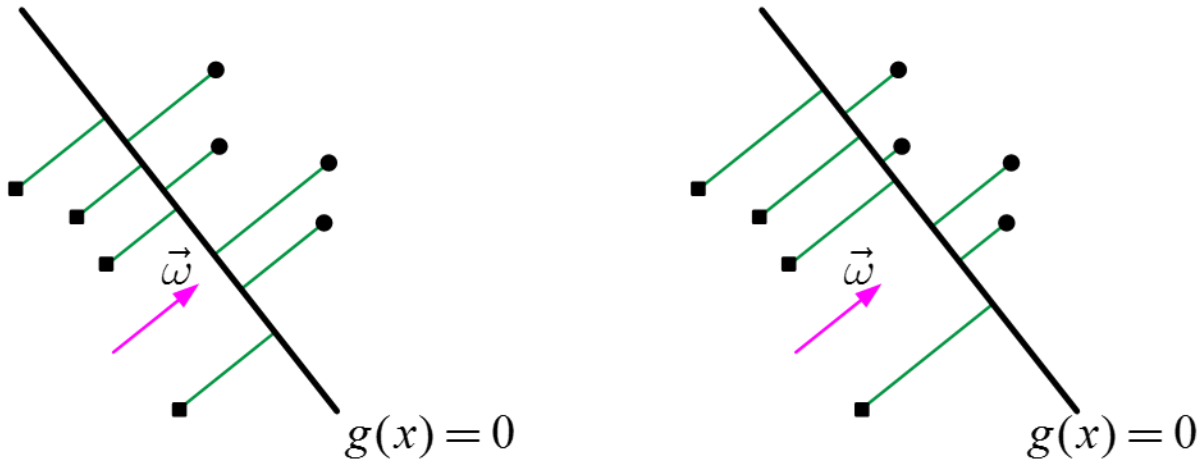


  - Inequality of arithmetic and geometric means

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

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

Using all Distances

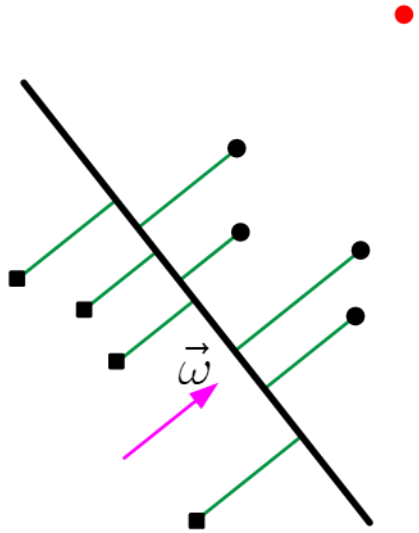


- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes

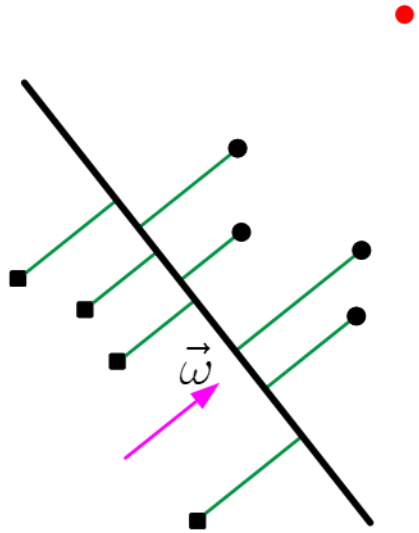
$$h = \frac{g(x)}{\|\omega\|} = \frac{\omega^T x}{\|\omega\|} \sim \omega^T x$$

Using all Distances with Outliers

- SVM vs. Logistic Regression



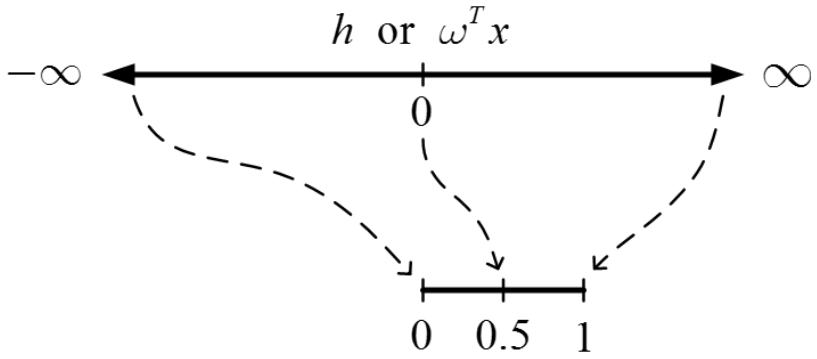
SVM



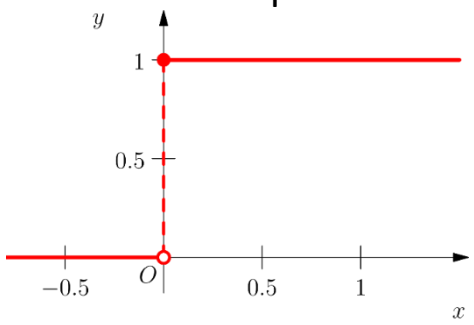
Logistic Regression

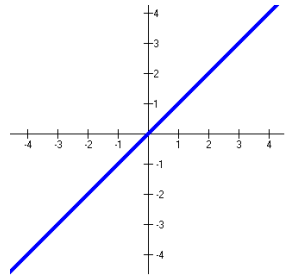
Sigmoid Function

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

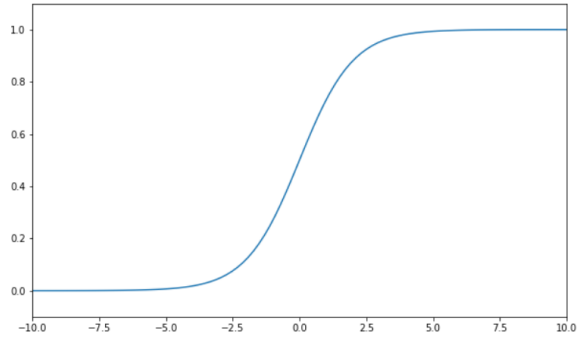


Step function





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

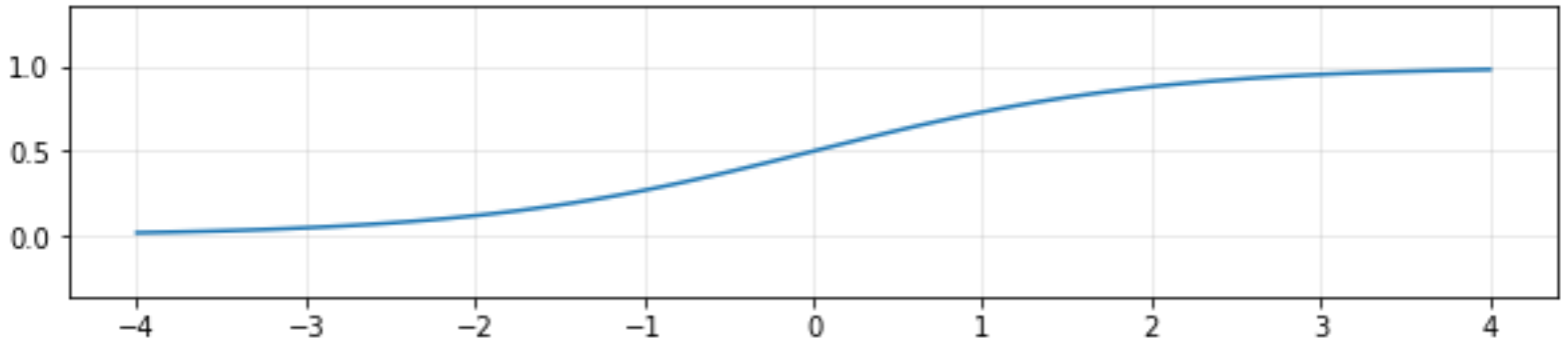


Sigmoid Function

- $\sigma(z)$ 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

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

  - Probability that the label is $+1$

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

  - Probability that the label is 0

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

Goal: We Need to Fit ω to Data

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

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

- It can be compactly written as

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

- 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

- 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 $\frac{\partial \ell(\omega)}{\partial \omega_j}$

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

Gradient Descent

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

- 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

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

Gradient Descent for Logistic Regression

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

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

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

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

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

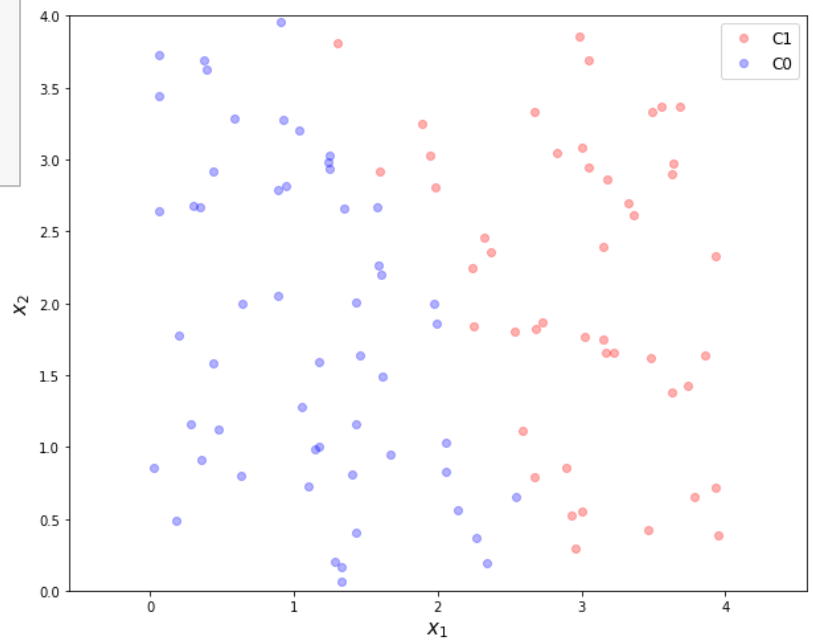
$$\nabla \ell(\omega) = \begin{bmatrix} \frac{\partial \ell(\omega)}{\partial \omega_0} \\ \frac{\partial \ell(\omega)}{\partial \omega_1} \\ \frac{\partial \ell(\omega)}{\partial \omega_2} \end{bmatrix} = X^T (y - h_\omega(x)) = X^T (y - \sigma(X\omega))$$

- Maximization problem
- Be careful on matrix shape

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

Python Implementation

```
# 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
```



Python Implementation

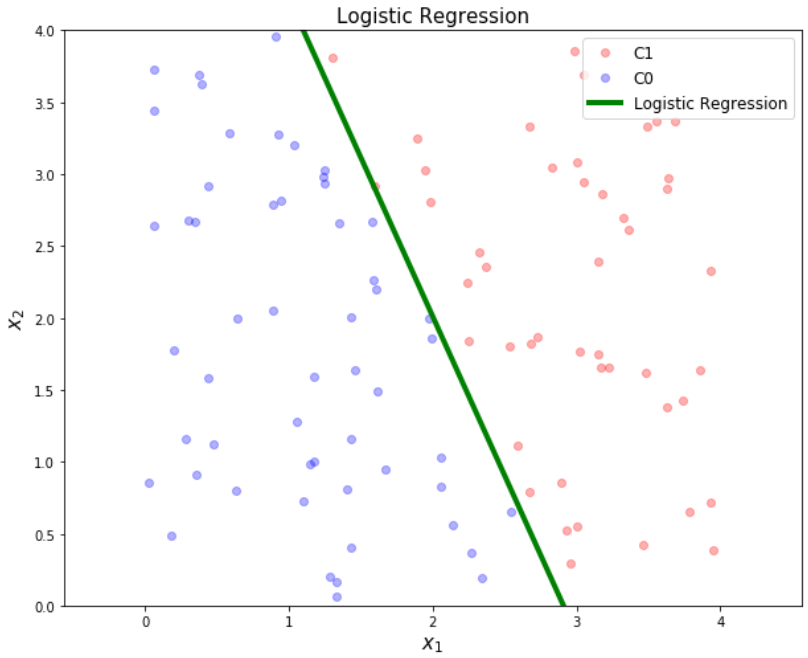
```
# be careful with matrix shape

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

```
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)
```

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

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

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



Logistic Regression using CVXPY

Probabilistic Approach (or MLE)

- Consider a random variable $y \in \{0, 1\}$

$$P(y = +1) = p, \quad P(y = 0) = 1 - p$$

where $p \in [0, 1]$, and is assumed to depend on a vector of explanatory variables $x \in \mathbb{R}^n$

- Then, the logistic model has the form

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

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

- We can re-order the training data so
  - for $x_1, \cdots, x_q$, the outcome is $y = +1$, and
  - for $x_{q+1}, \cdots, x_m$, the outcome is $y = 0$

Probabilistic Approach (or MLE)

- Likelihood function

$$\mathscr{L} = \prod_{i=1}^{q} p_i \prod_{i=q+1}^{m} (1 - p_i) \qquad \left(\sim \prod_i |h_i| \right)$$

- Log likelihood function

$$\begin{aligned} \ell(\omega) = \log \mathscr{L} &= \sum_{i=1}^{q} \log p_i + \sum_{i=q+1}^{m} \log(1 - p_i) \\ &= \sum_{i=1}^{q} \log \frac{\exp(\omega^T x_i)}{1 + \exp(\omega^T x_i)} + \sum_{i=q+1}^{m} \log \frac{1}{1 + \exp(\omega^T x_i)} \\ &= \sum_{i=1}^{q} (\omega^T x_i) - \sum_{i=1}^{m} \log(1 + \exp(\omega^T x_i)) \end{aligned}$$

- Since ℓ is a concave function of ω, the logistic regression problem can be solved as a convex optimization problem

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

CVXPY Implementation

$$\ell(\omega) = \log \mathscr{L} = \sum_{i=1}^{q} \log p_i + \sum_{i=q+1}^{m} \log(1 - p_i)$$
$$= \sum_{i=1}^{q} \log \frac{\exp(\omega^T x_i)}{1 + \exp(\omega^T x_i)} + \sum_{i=q+1}^{m} \log \frac{1}{1 + \exp(\omega^T x_i)}$$
$$= \sum_{i=1}^{q} (\omega^T x_i) - \sum_{i=1}^{m} \log(1 + \exp(\omega^T x_i))$$

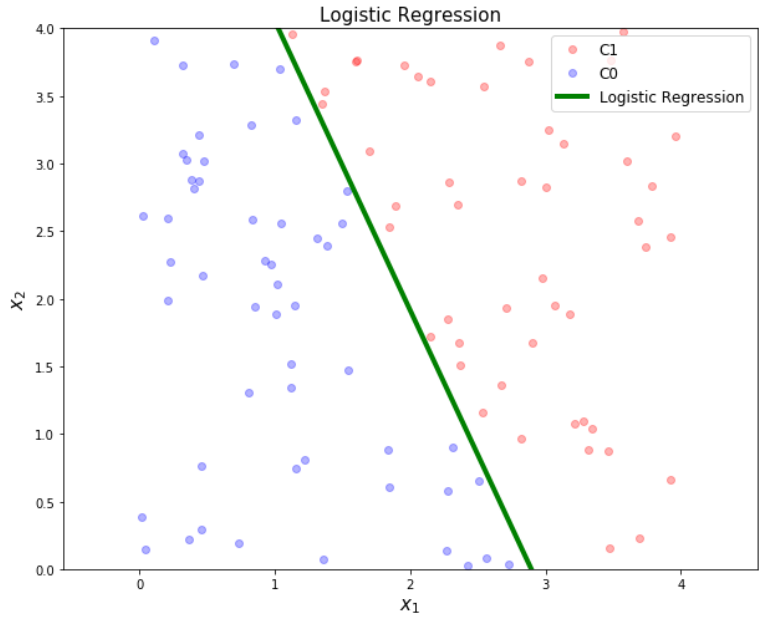
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

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

```
w = cvx.Variable([3, 1])

obj = cvx.Maximize(y.T*X*w - cvx.sum(cvx.logistic(X*w)))
prob = cvx.Problem(obj).solve()

w = w.value

xp = np.linspace(0,4,100).reshape(-1,1)
yp = - w[1,0]/w[2,0]*xp - w[0,0]/w[2,0]
```

`cvx.sum(x)` = $\sum_{ij} x_{ij}$

`cvx.logistic(x)` = $\log(1 + e^x)$



In a More Compact Form

- Change $y \in \{0, +1\} \rightarrow y \in \{-1, +1\}$ for computational convenience

  – Consider the following function

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

  – Log-likelihood

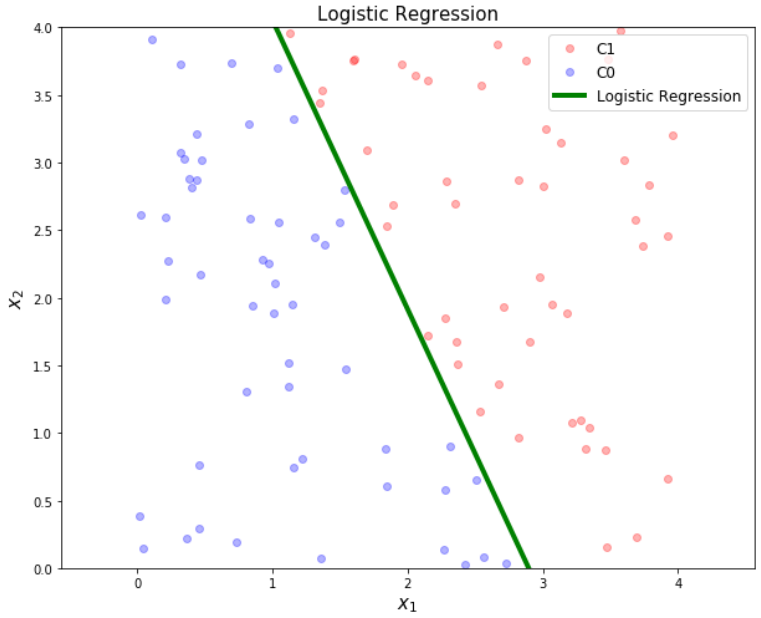
$$\begin{aligned}
\ell(\omega) = \log \mathscr{L} = \log P\left(y \mid x, \omega\right) &= \log \prod_{n=1}^{m} P\left(y_n \mid x_n, \omega\right) \\
&= \sum_{n=1}^{m} \log P\left(y_n \mid x_n, \omega\right) \\
&= \sum_{n=1}^{m} \log \frac{1}{1 + \exp(-y_n \omega^T x_n)} \\
&= \sum_{n=1}^{m} -\log\left(1 + \exp\left(-y_n \omega^T x_n\right)\right)
\end{aligned}$$

CVXPY Implementation

$$\hat{\omega} = \arg\max_{\omega} \sum_{n=1}^{m} -\log\left(1+\exp\left(-y_n \omega^T x_n\right)\right)$$

$$= \arg\min_{\omega} \sum_{n=1}^{m} \log\left(1+\exp\left(-y_n \omega^T x_n\right)\right)$$

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

w = cvx.Variable([3, 1])

obj = cvx.Minimize(cvx.sum(cvx.logistic(-cvx.multiply(y,X*w))))
prob = cvx.Problem(obj).solve()

w = w.value

xp = np.linspace(0,4,100).reshape(-1,1)
yp = - w[1,0]/w[2,0]*xp - w[0,0]/w[2,0]
```



`cvx.sum(x)` = $\sum_{ij} x_{ij}$

`cvx.logistic(x)` = $\log(1+e^x)$

Logistic Regression using Scikit-Learn

Logistic Regression using Scikit-Learn

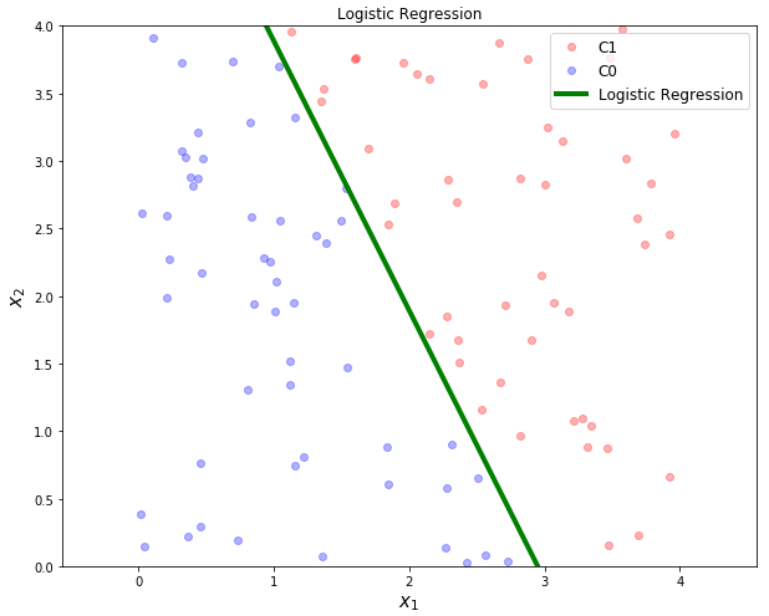
```
X = X[:,1:3]

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

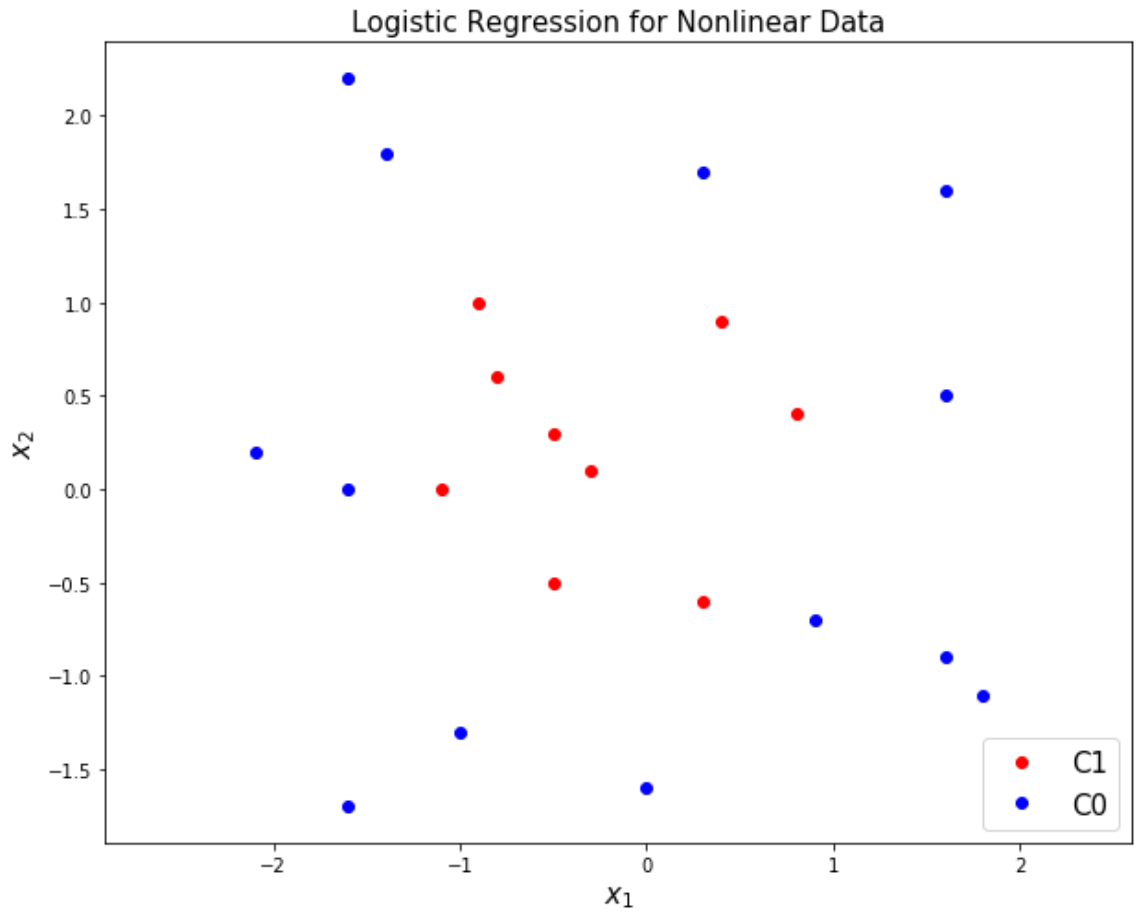
$$P\left(y = k \mid x, \omega\right) = \frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)} \in [0, 1]$$

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

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