# Classification: Perceptron

**Prof. Seungchul Lee**
**Industrial AI Lab.**

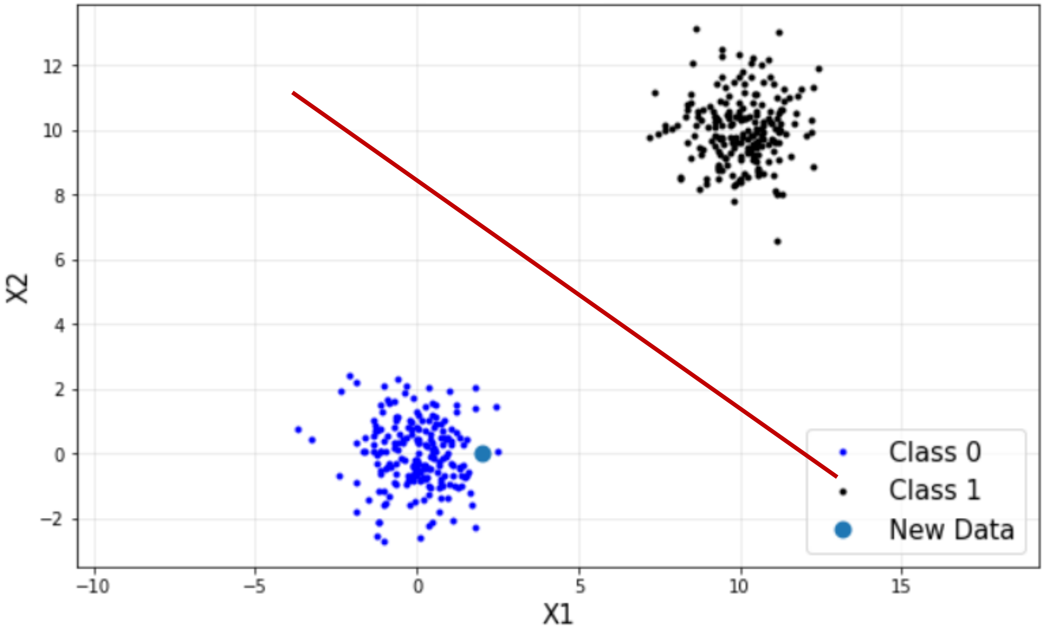
# Classification

- 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

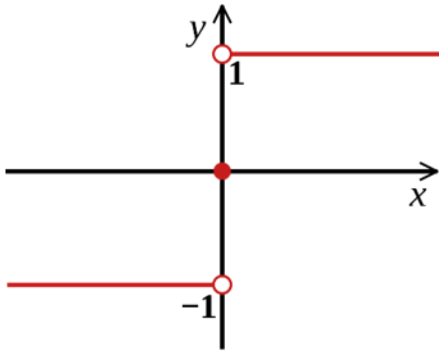


# Perceptron

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

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

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



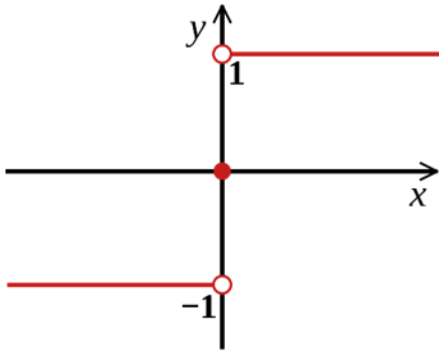

$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) - \text{threshold}\right) = \text{sign}\left(\left(\sum_{i=1}^{d} \omega_i x_i\right) + \omega_0\right)$$

# 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

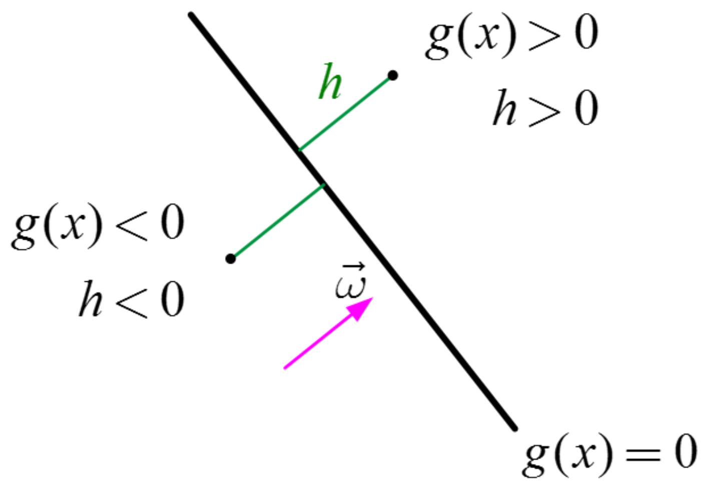
$$h(x) = \text{sign}\left(\omega^T x\right)$$

# Sign

- Sign with respect to a line

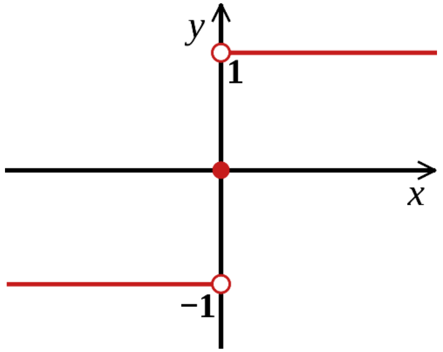
$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_1 x_1 + \omega_2 x_2 + \omega_0 = \omega^T x + \omega_0$$

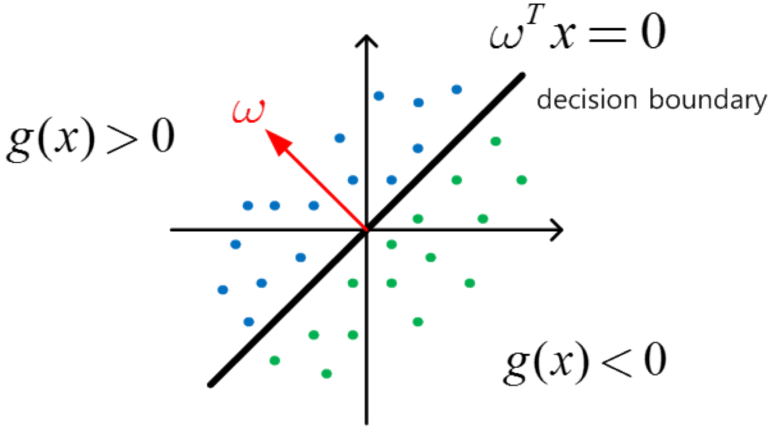
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = \omega^T x$$



# How to Find $\omega$

- 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

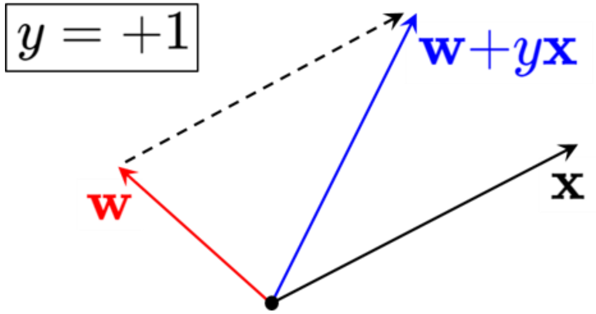
$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N) \quad \text{where } y_i \in \{-1, 1\}$$

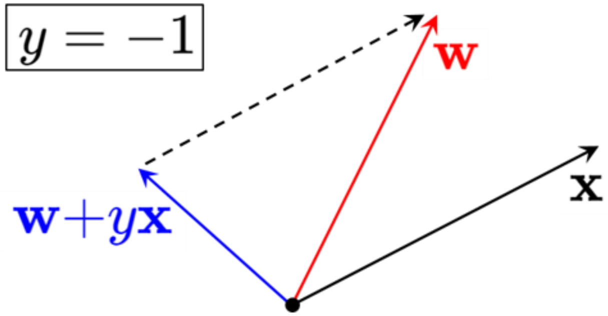
1) pick a misclassified point

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

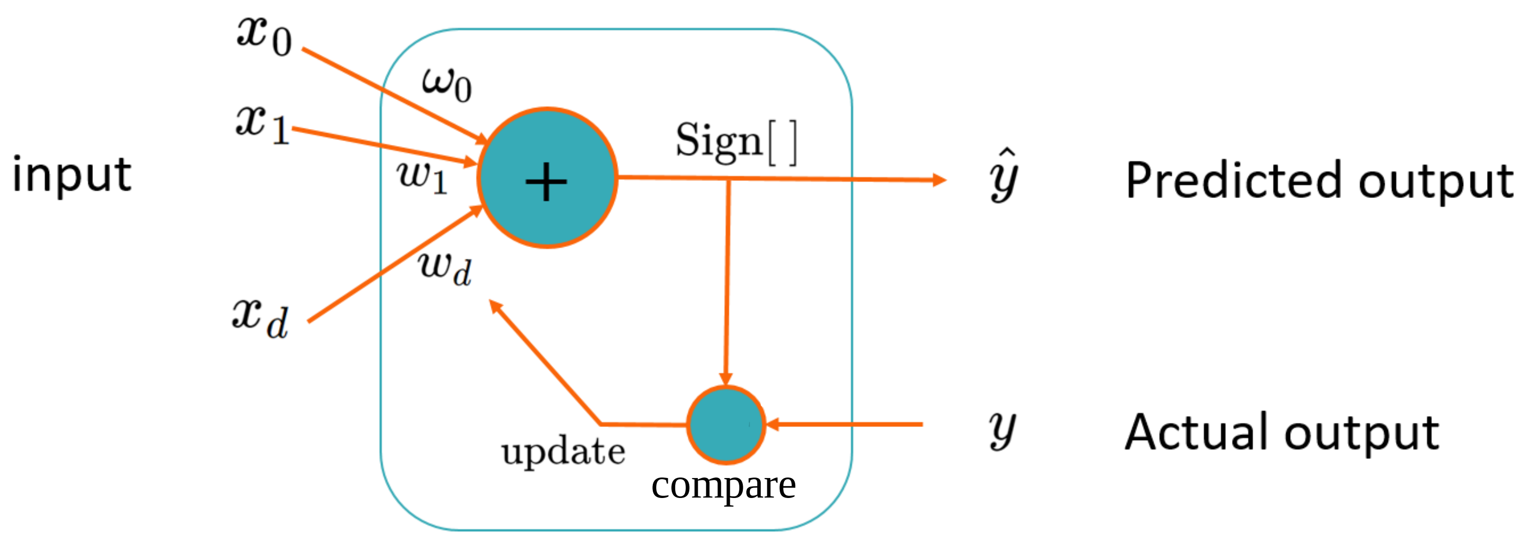
2) and update the weight vector

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



# Diagram of Perceptron



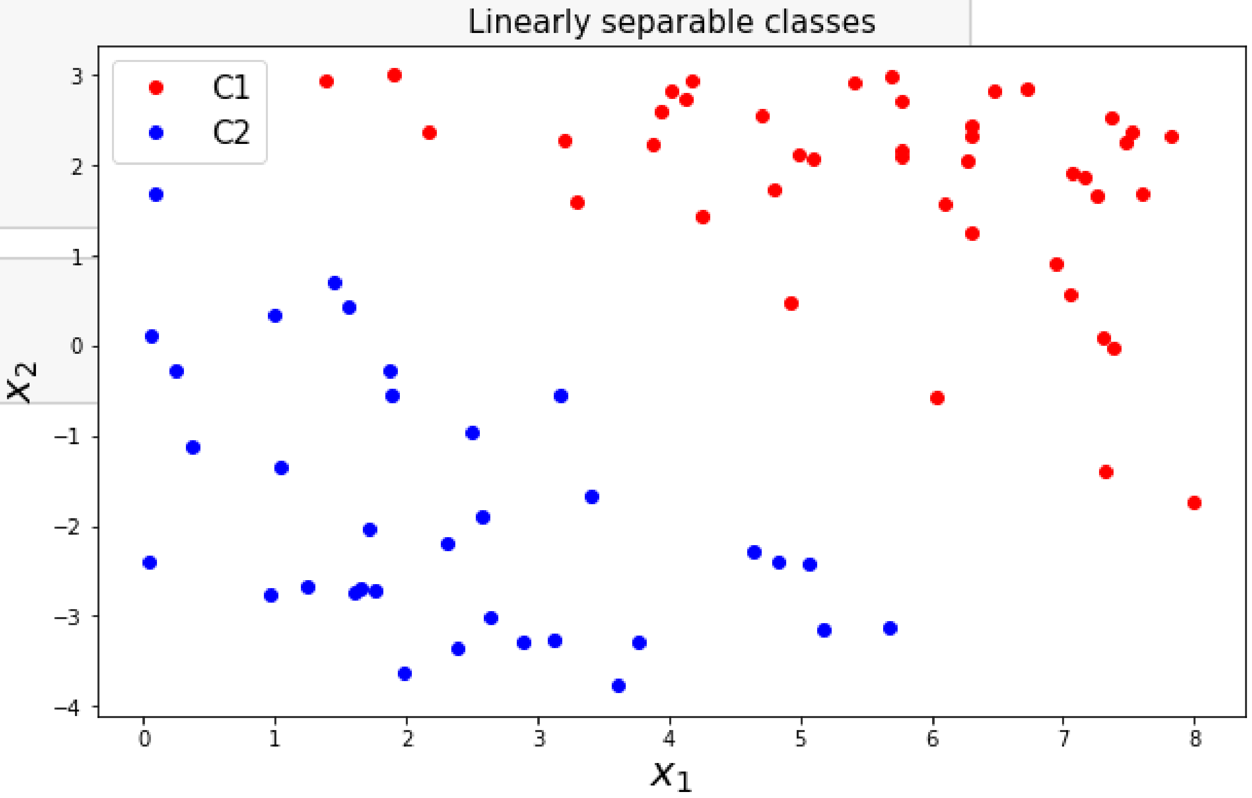
# Perceptron Algorithm in Python

```
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)
```



# 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))
```

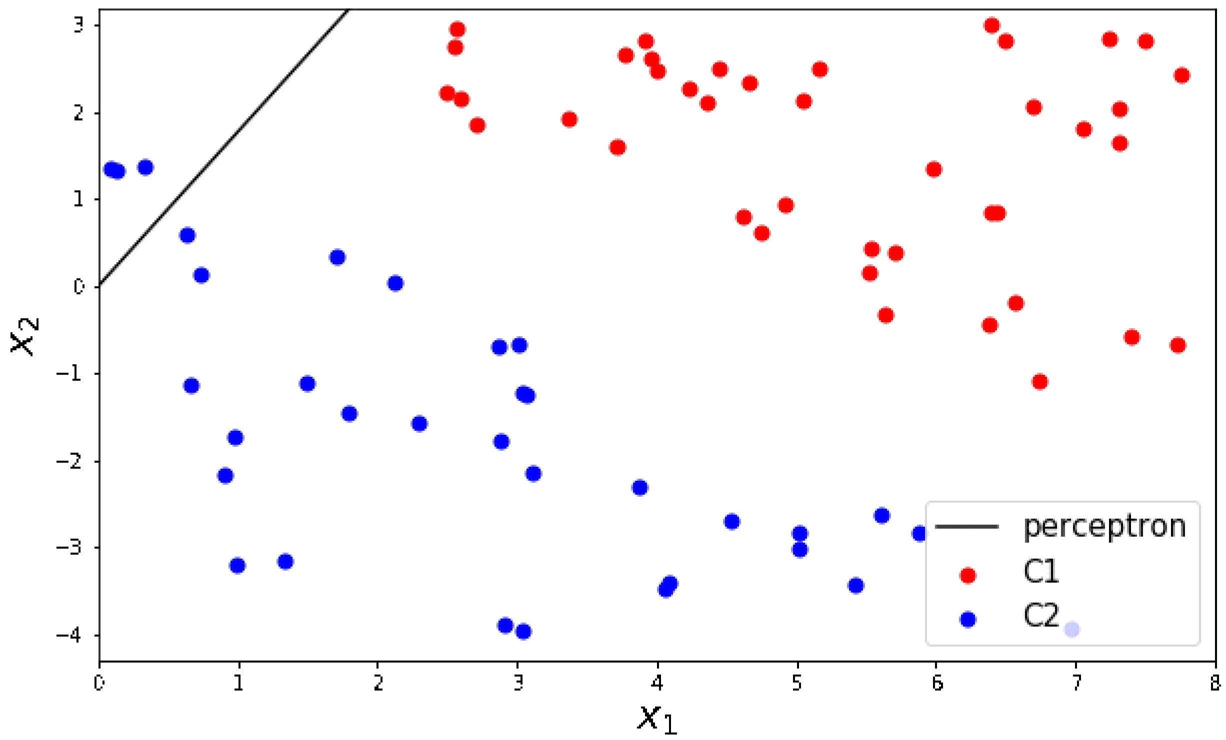
```
clf.predict([[3, -2]])
```

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

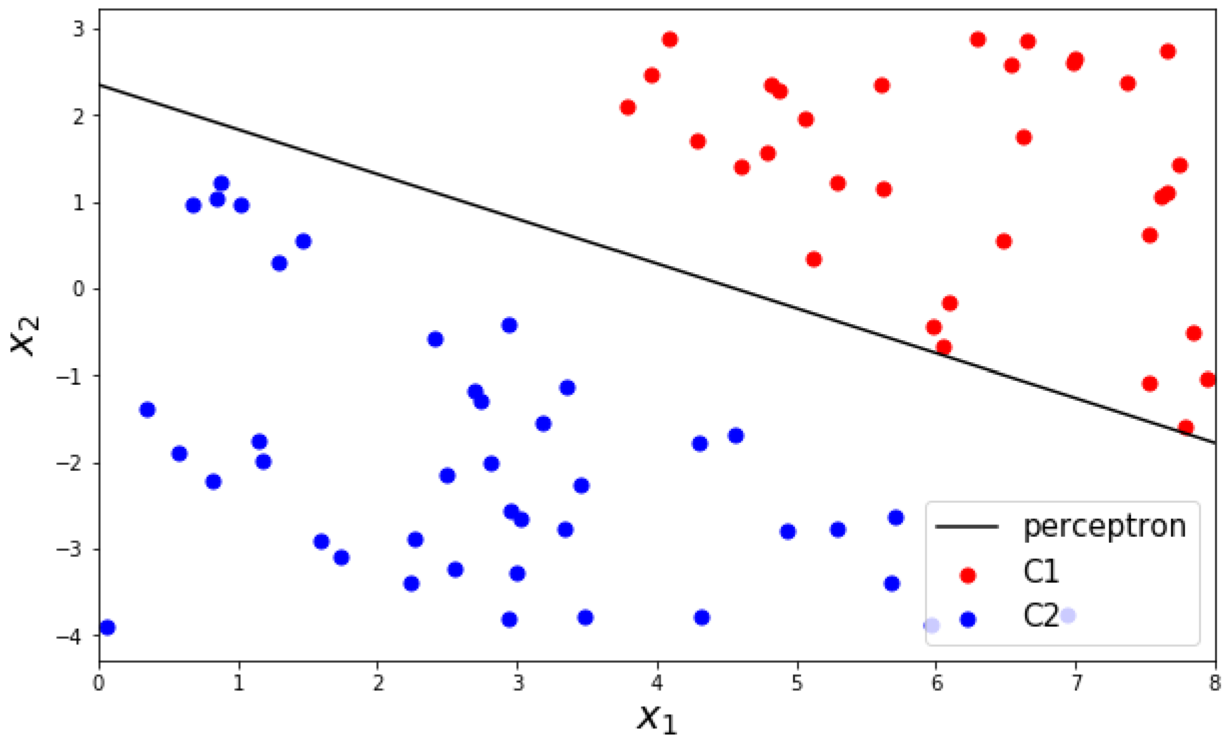
$$x = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix}$$

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

# Perceptron Algorithm in Python



# Perceptron Algorithm in Python



# The Best Hyperplane Separator?

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

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

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

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

# Logistic Regression

**Prof. Seungchul Lee**
**Industrial AI Lab.**

# Linear Classification: Logistic Regression

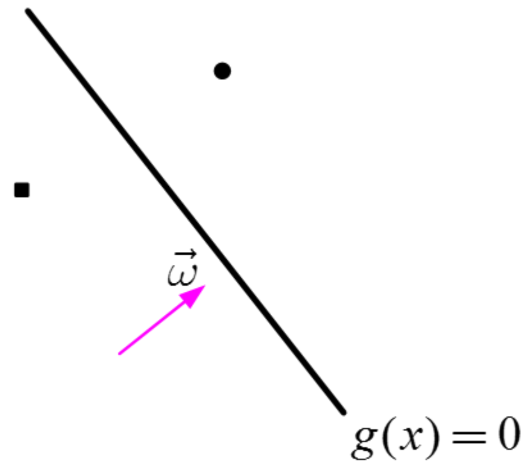
- Logistic regression is a classification algorithm
  - don't be confused

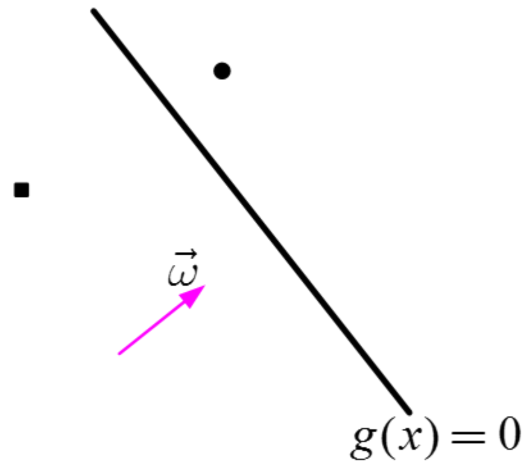
- Perceptron: make use of sign of data

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

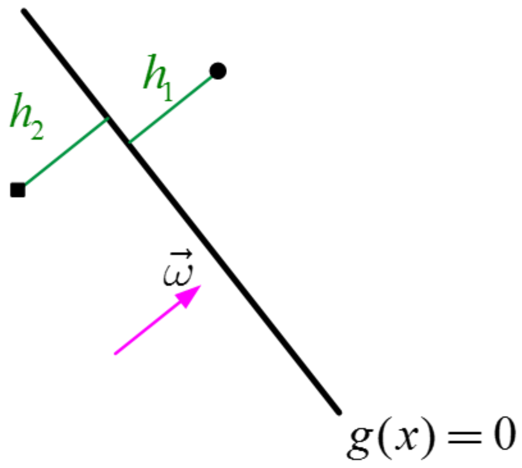
- We want to use distance information of all data points
  - logistic regression

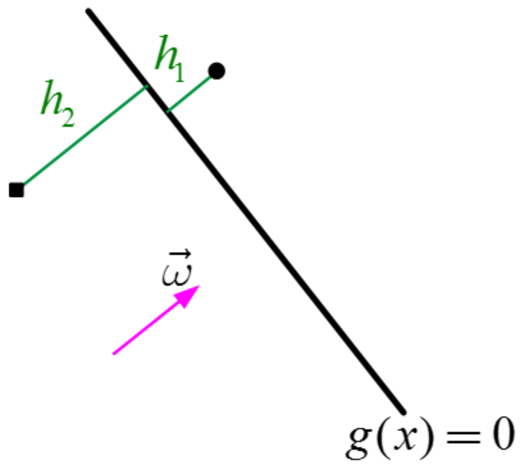
# Using Distances



# Using Distances

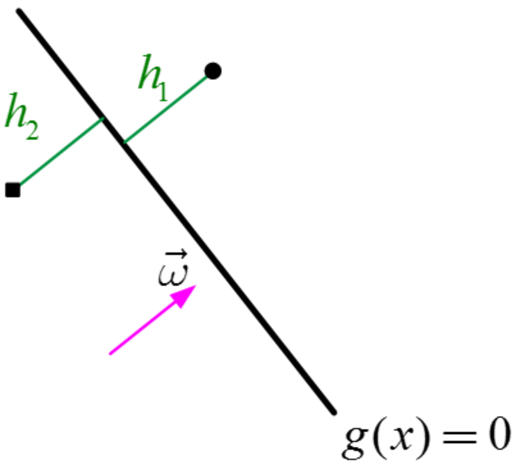


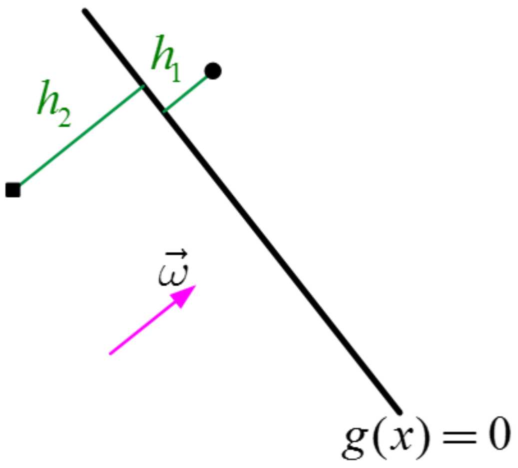




$|h_1| + |h_2|$

$|h_1| + |h_2|$

# Using Distances

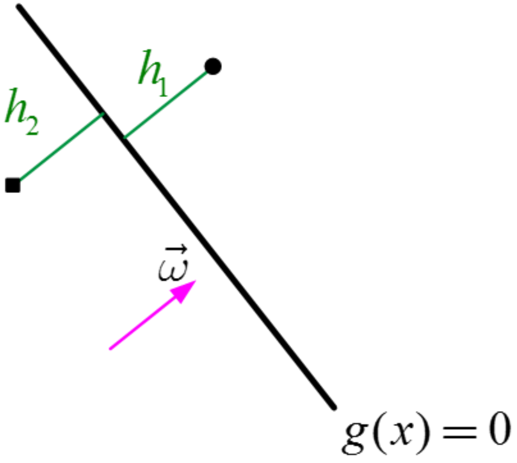


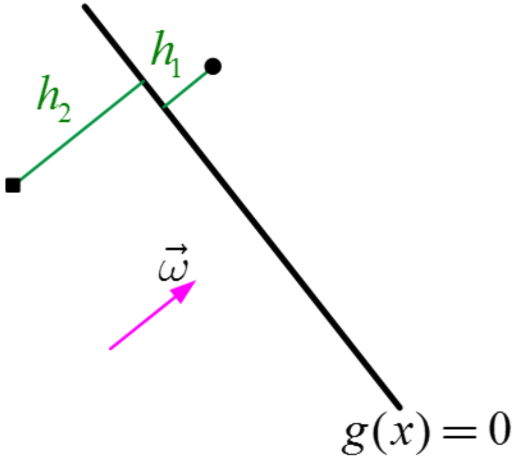




$|h_1| + |h_2|$ $\qquad$ $|h_1| + |h_2|$

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

# Using Distances




$|h_1| + |h_2|$

$|h_1| \cdot |h_2|$

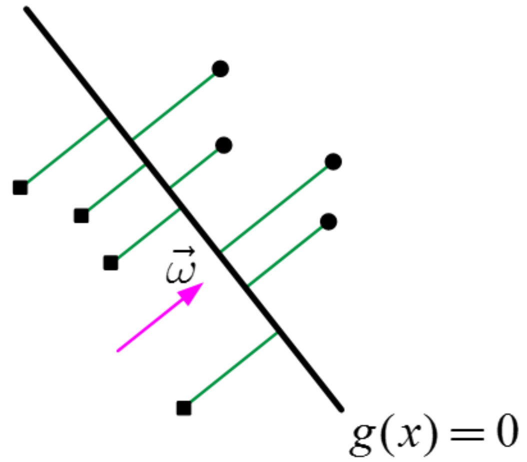
$|h_1| + |h_2|$

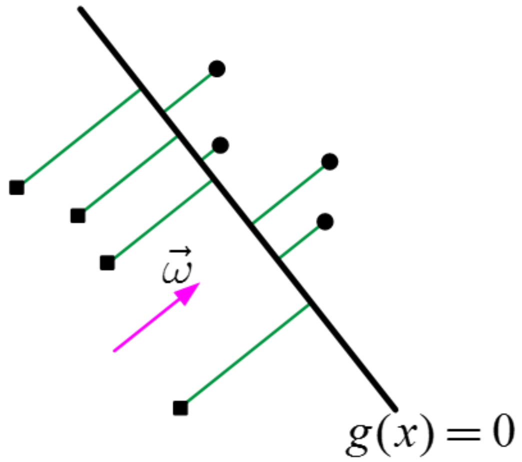
$|h_1| \cdot |h_2|$

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

# Using all Distances

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



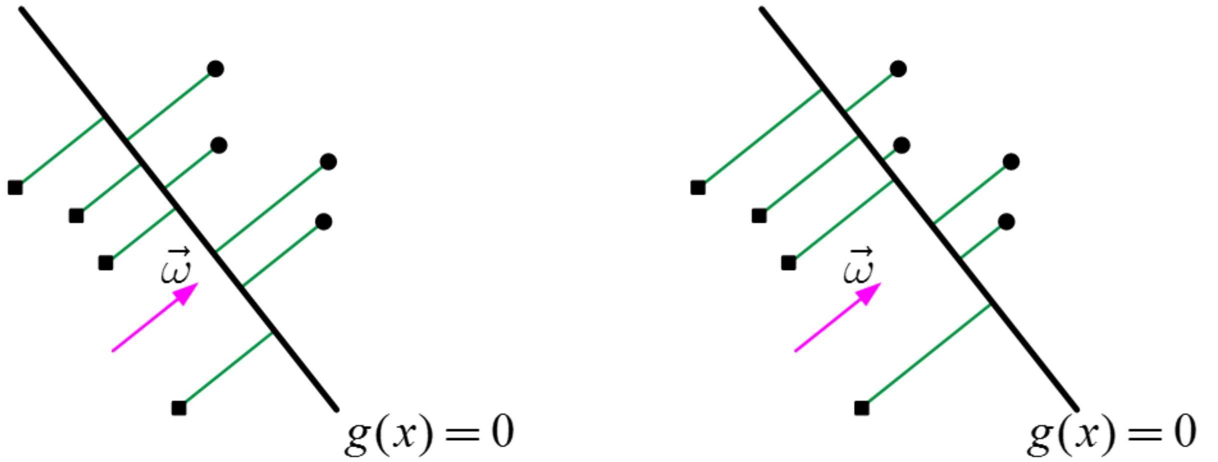




  – Inequality of arithmetic and geometric means

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

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

# Using all Distances

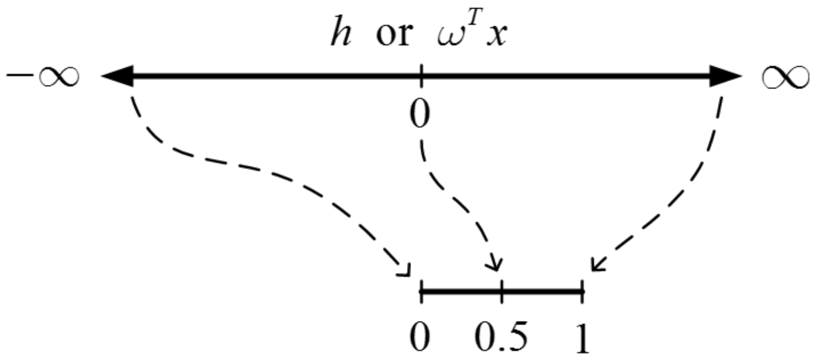



- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes

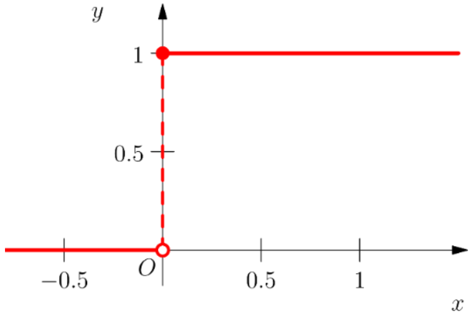
$$h = \frac{g(x)}{||\omega||} = \frac{\omega^T x}{||\omega||} \sim \omega^T x$$

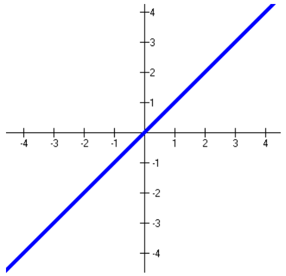
# Sigmoid Function

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

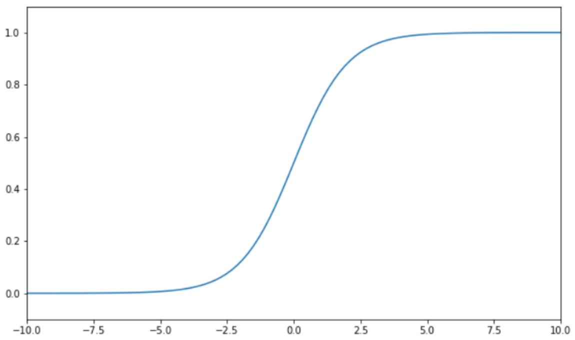



Step function



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



# 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

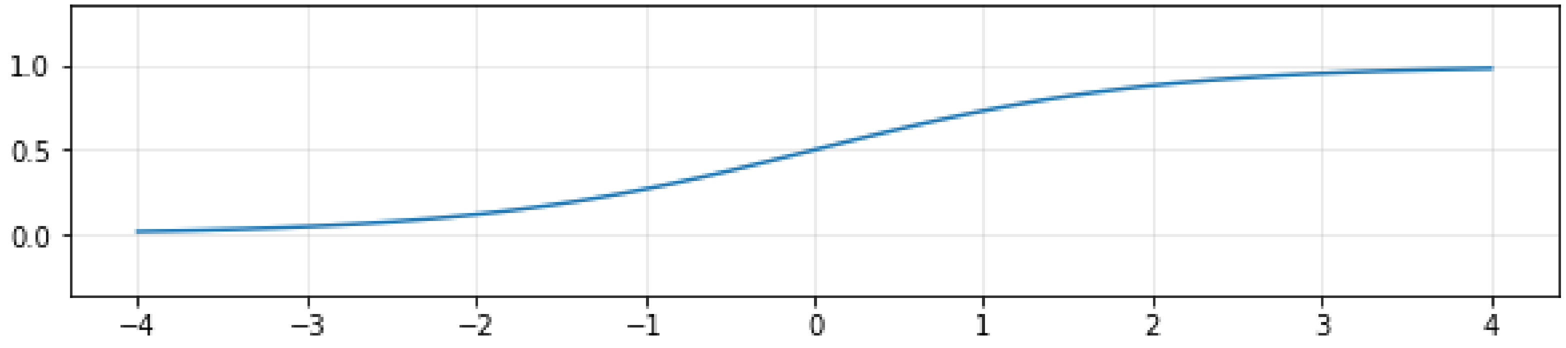
```python
# plot a sigmoid function

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

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

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

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



# Sigmoid Function

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

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

  - Probability that the label is +1

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

  - Probability that the label is 0

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

# Goal: We Need to Fit $\omega$ to Data

- For a single data point $(x, y)$ with parameters $\omega$

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

- It can be compactly written as

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

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

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

# Goal: We Need to Fit $\omega$ 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 Scikit-Learn

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

X.shape
```

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

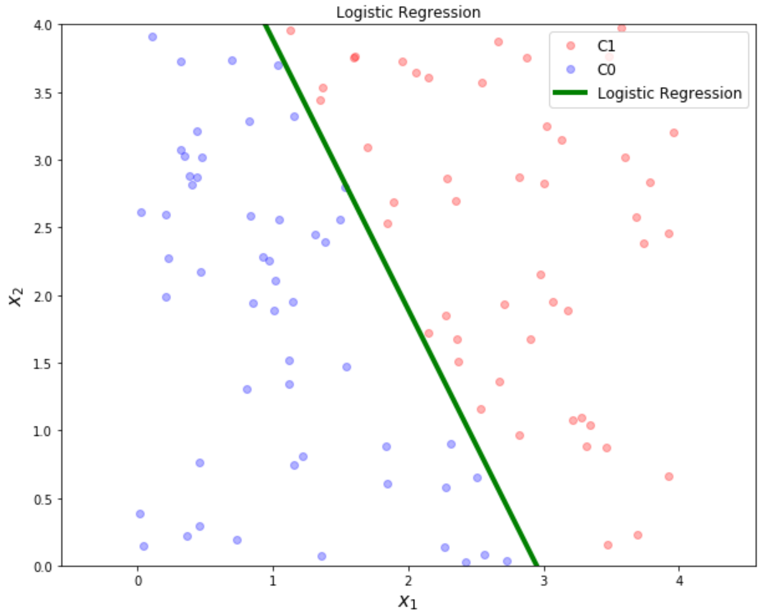
$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \left(x^{(3)}\right)^T \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots \end{bmatrix}, \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)}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

```
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

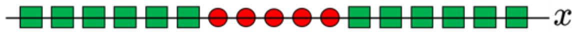
$$P\left(y=k \mid x,\omega\right)=\frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)}\in[0,1]$$

- We maintain a separator weight vector $\omega_k$ for each class $k$

# Non-linear Classification

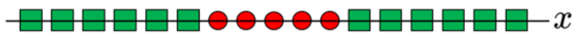
# Classifying Non-linear Separable Data

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

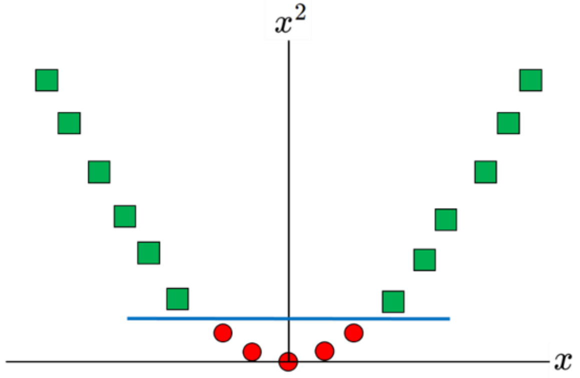


# Classifying Non-linear Separable Data

- 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

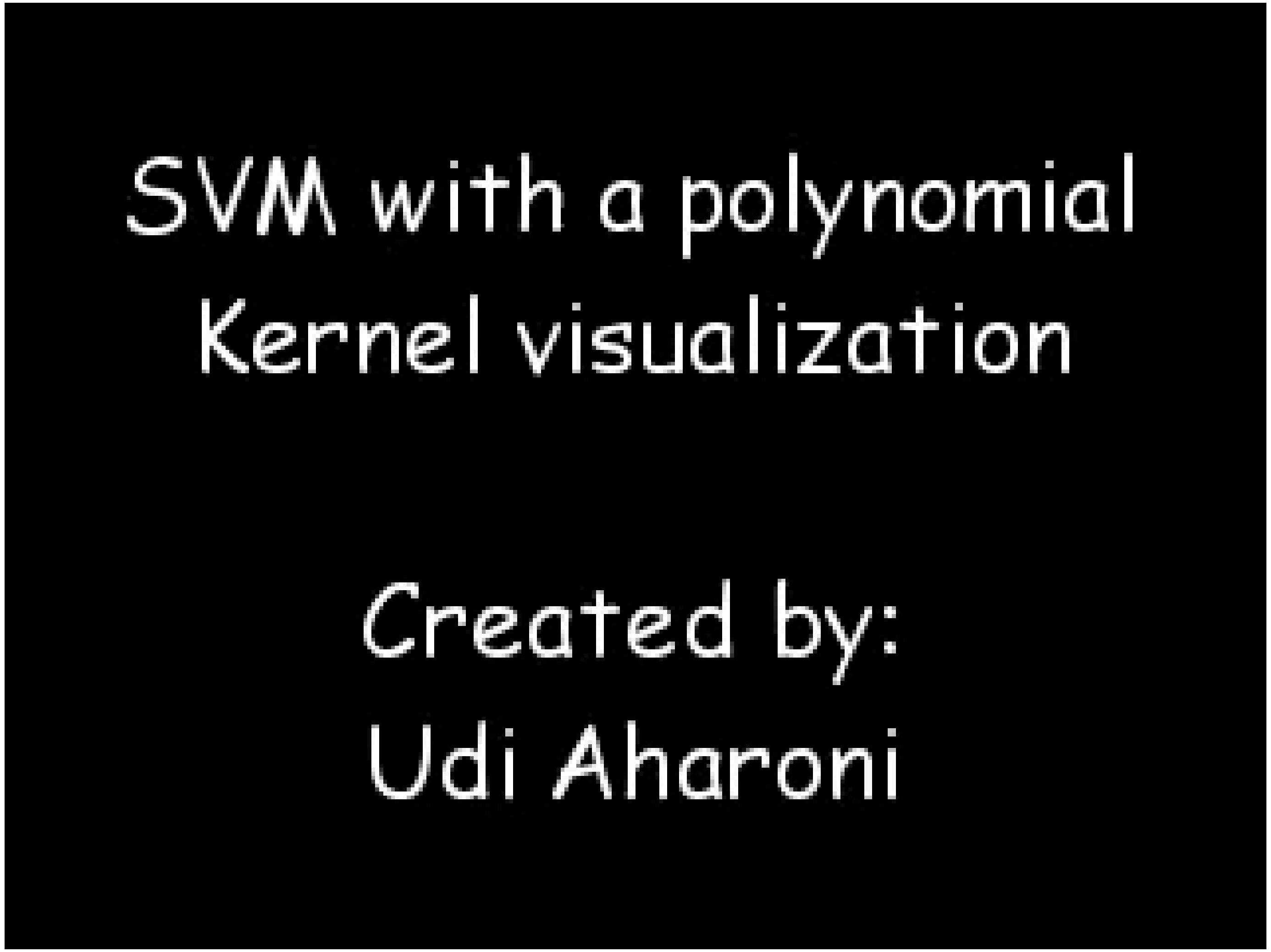
# Kernel

- 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

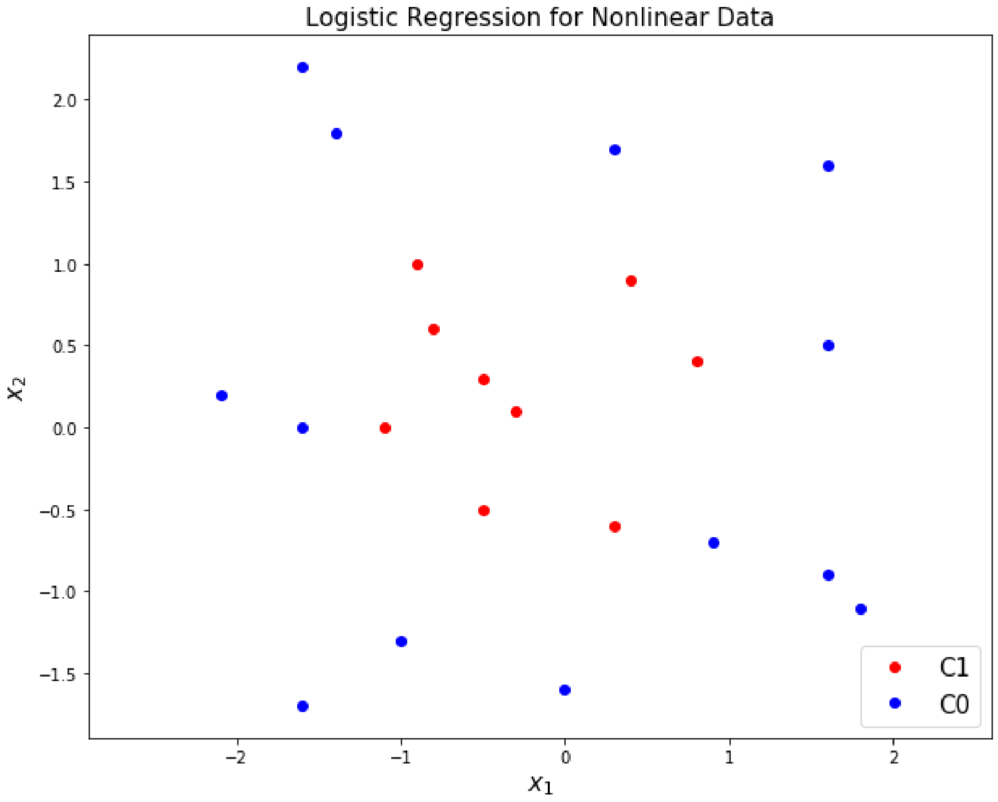
# Nonlinear Classification




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

# Non-linear Classification

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




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

# Non-linear Classification