

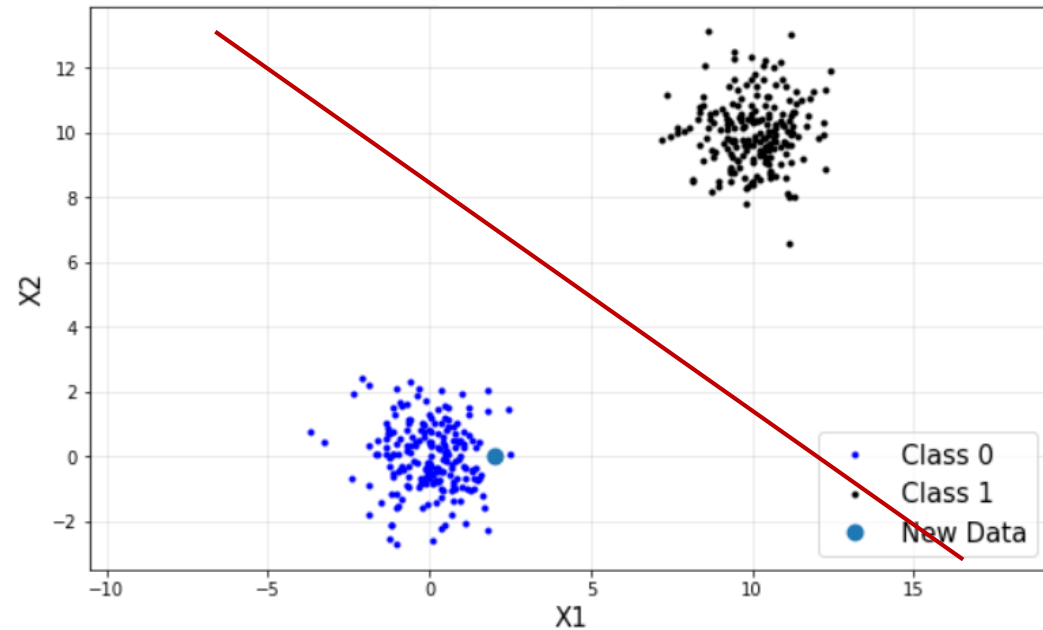


Classification: Perceptron

Prof. Seungchul Lee
Industrial AI Lab.

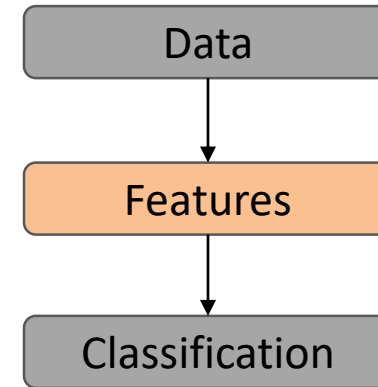
Classification

- Where y is a discrete value
 - Develop the classification algorithm to determine which class a new input should fall into
- We will learn
 - Perceptron
 - Logistic regression
- To find a classification boundary



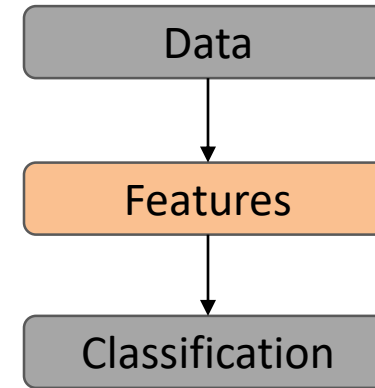
Perceptron

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

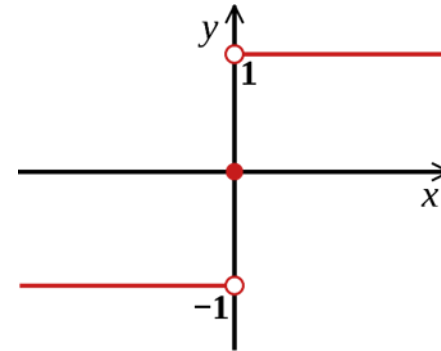


Perceptron

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



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

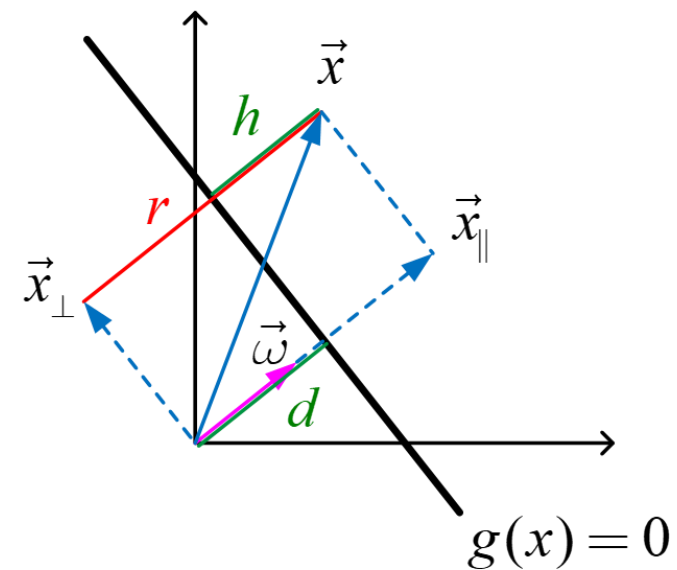
$$h(x) = \text{sign} (\omega^T x)$$

- Let's see geometrical meaning of perceptron

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

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

$\therefore \omega$: normal to the line (orthogonal)
 \Rightarrow tells the direction of the line



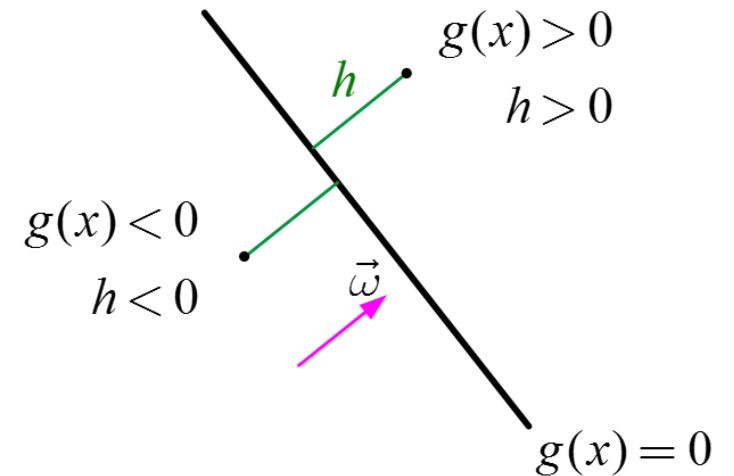
Signed Distance from a Line: h

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

- Sign with respect to a line

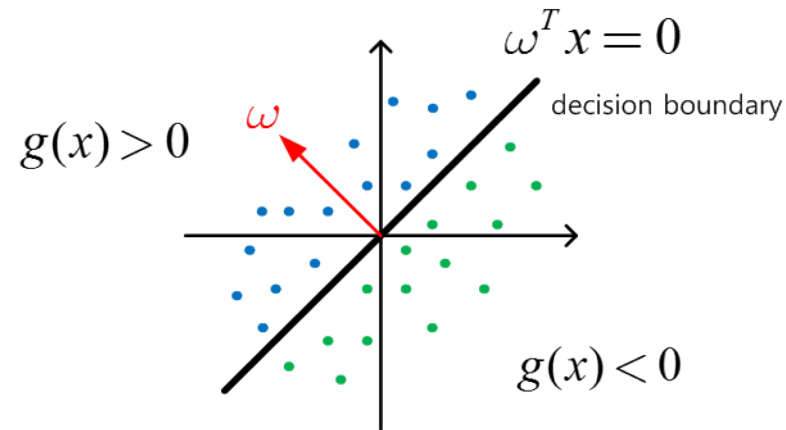
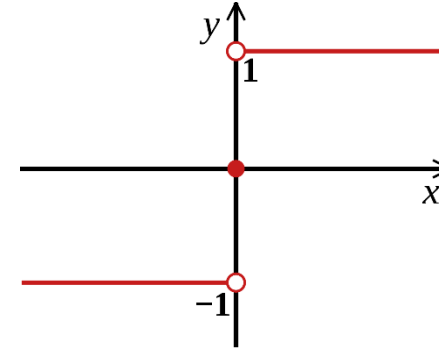
$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies g(x) = \omega_1 x_1 + \omega_2 x_2 + \omega_0 = \omega^T x + \omega_0$$

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



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

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

- Given the training set

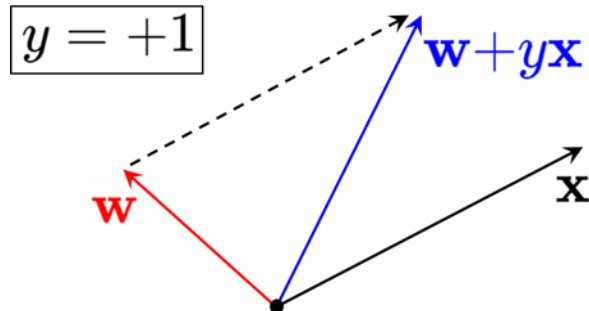
$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N) \quad \text{where } y_i \in \{-1, 1\}$$

- 1) pick a misclassified point

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

- 2) and update the weight vector

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



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 $t = 1, 2, 3, \dots$, pick a misclassified point from

$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$$

4. And run a PLA iteration on it
5. That's it!

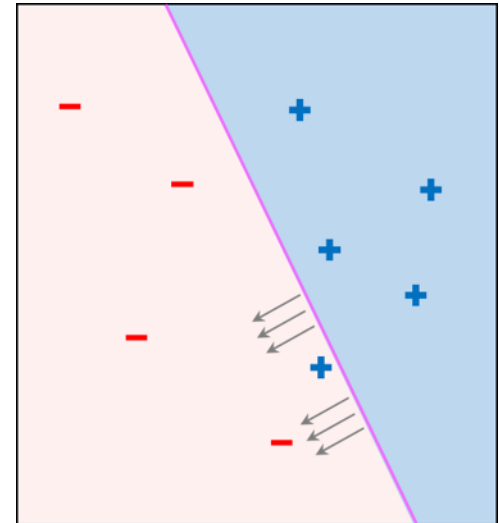
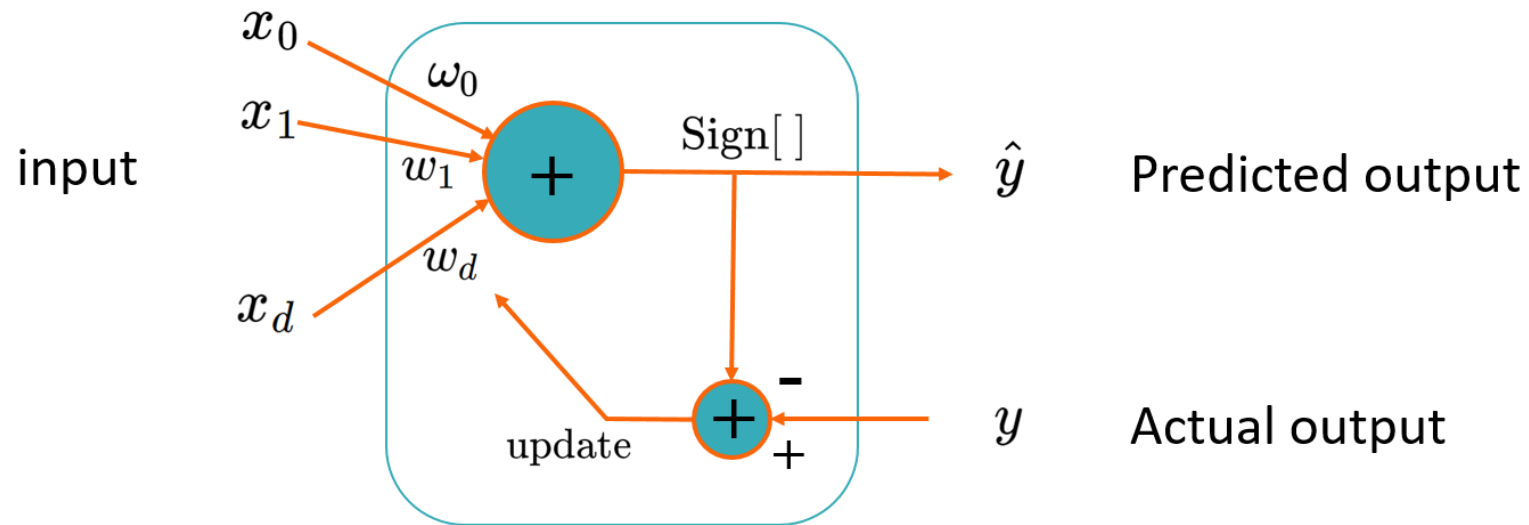


Diagram of Perceptron



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

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))
```

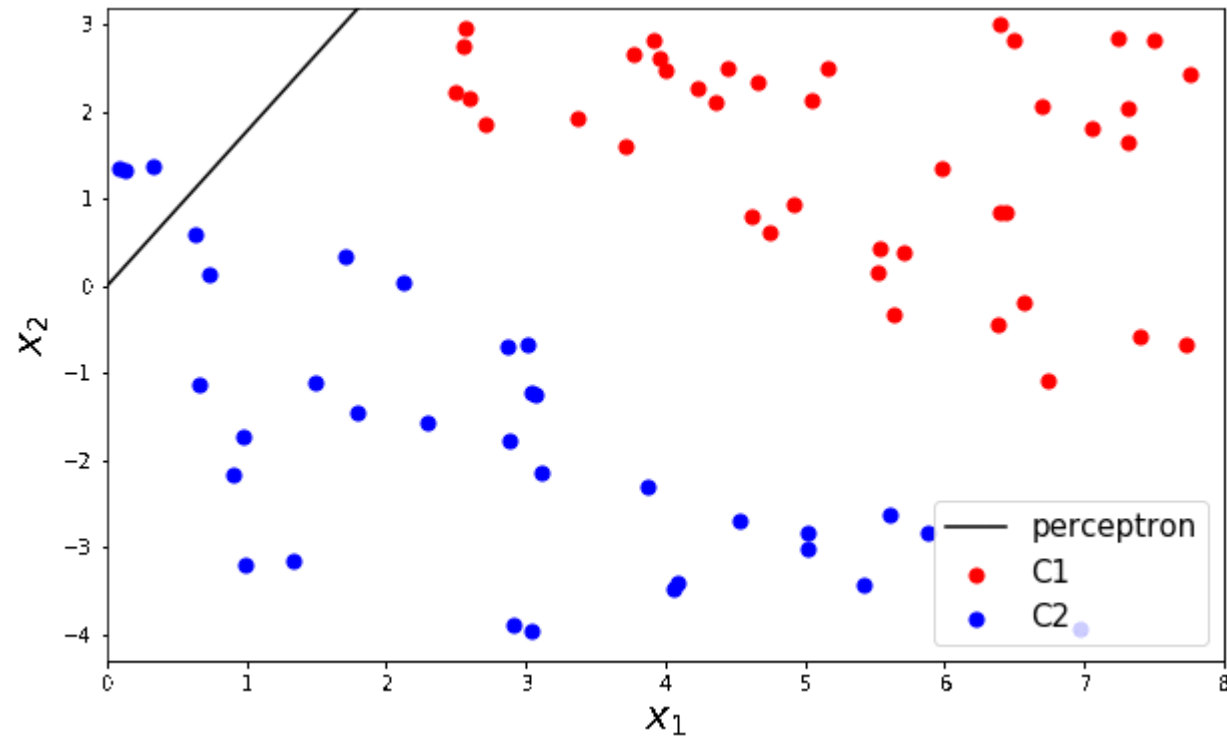
```
clf.predict([[3, -2]])
```

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

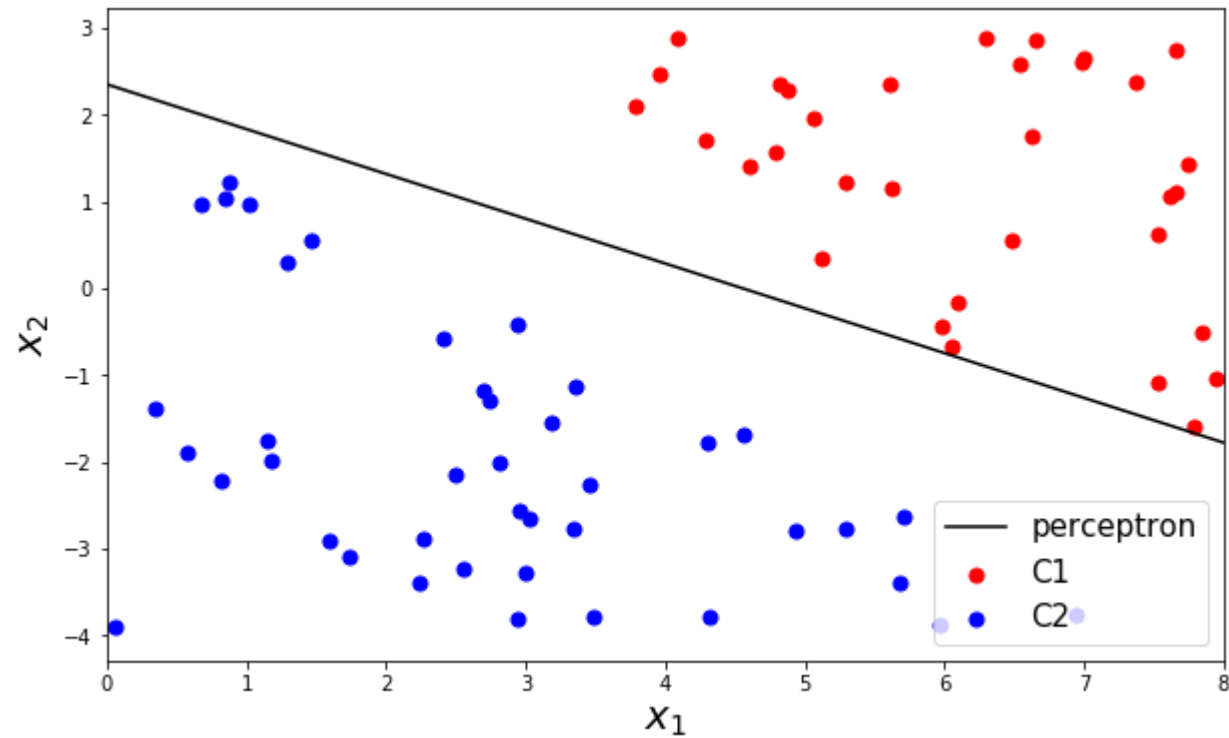
$$x = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ (x^{(3)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} \end{bmatrix}$$

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

Perceptron Algorithm in Python



Perceptron Algorithm in Python



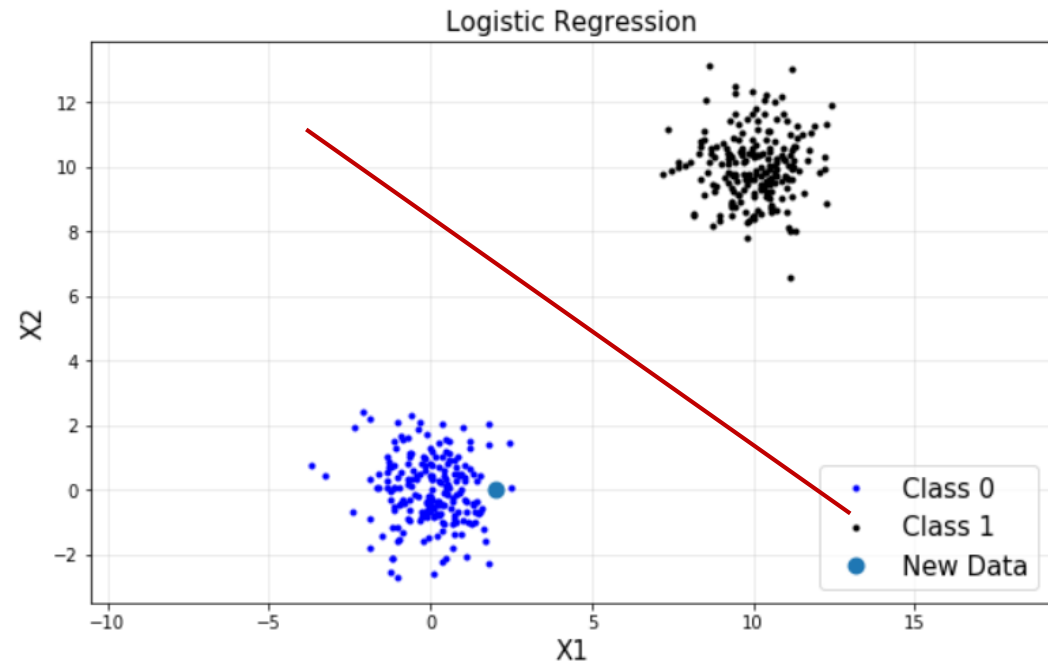
The Best Hyperplane Separator?

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

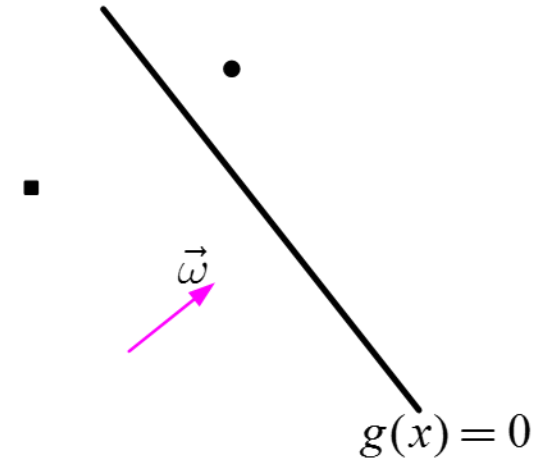
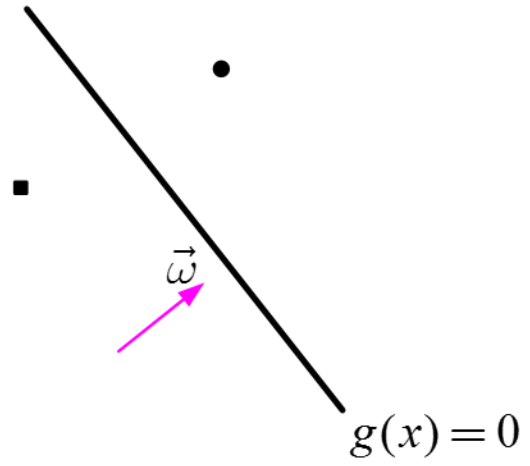
Classification: Logistic Regression

Classification: Logistic Regression

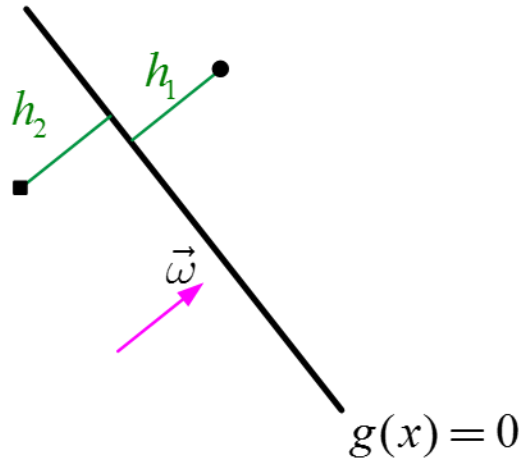
- Perceptron: make use of sign of data
- Logistic regression: make use of distance of data
- Logistic regression is a classification algorithm
 - don't be confused from its name
- To find a classification boundary



Using Distances



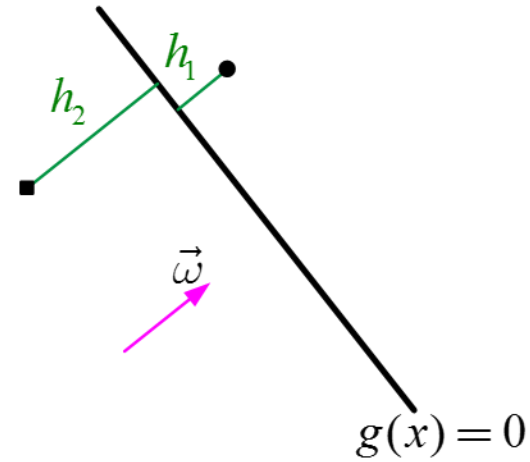
Using Distances



$$|h_1| + |h_2|$$

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

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



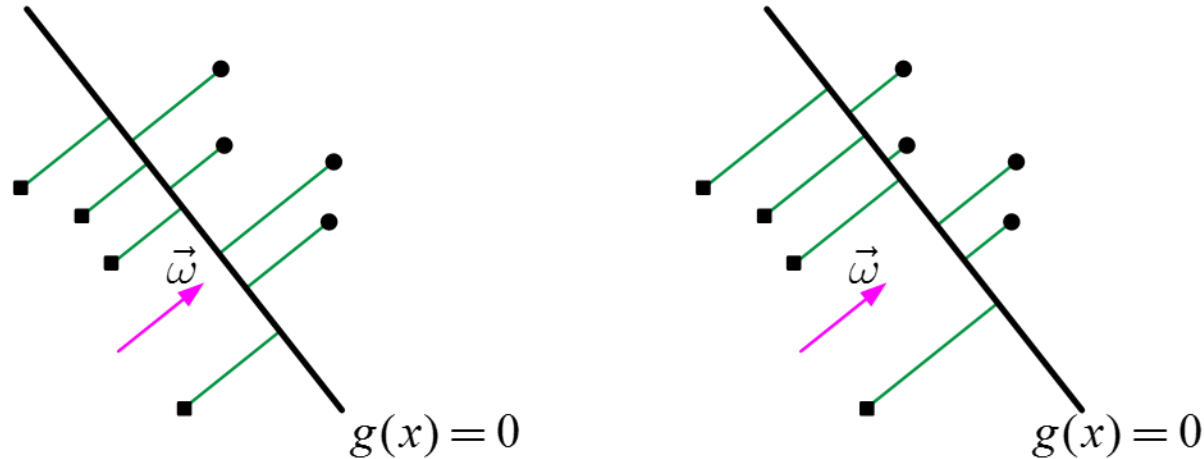
$$|h_1| + |h_2|$$

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

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

Using all Distances

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

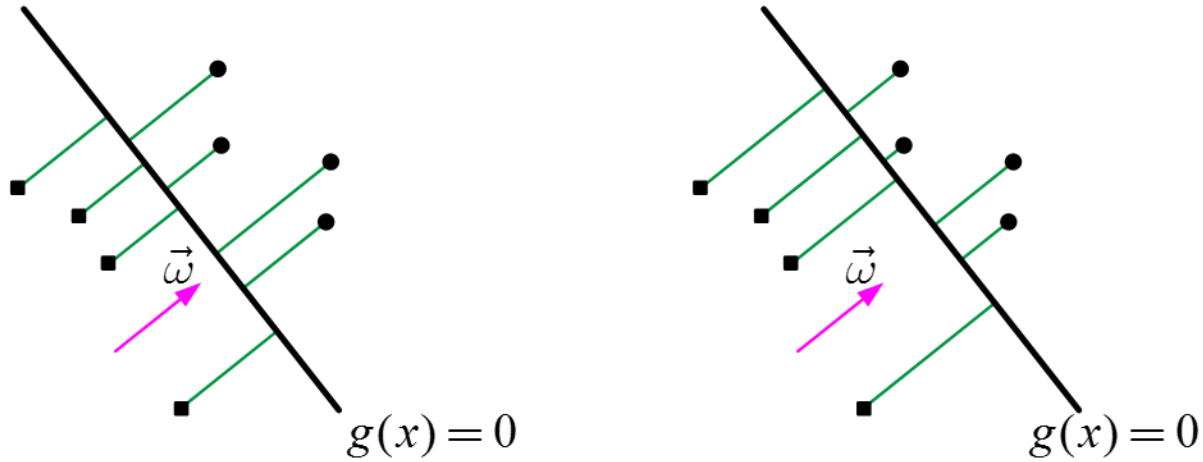


- Inequality of arithmetic and geometric means

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

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

Using all Distances

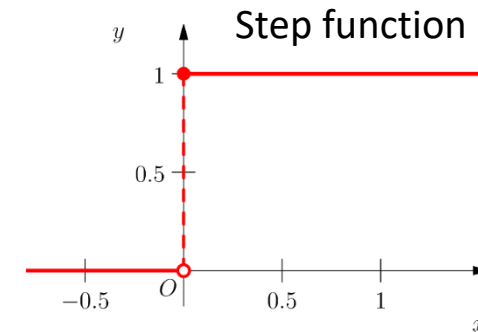
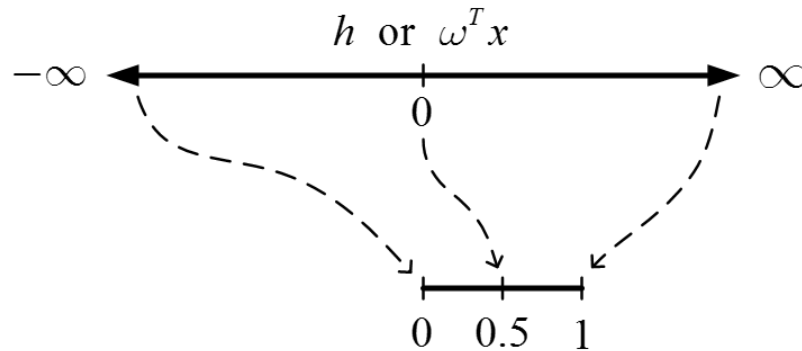


- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes

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

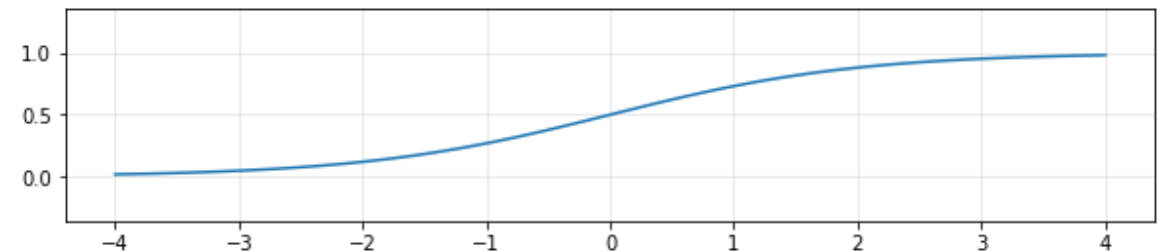
Sigmoid Function

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



- $\sigma(z)$ 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

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



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

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

- 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 ω to our Data

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

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

$$P(y = 0 \mid x; \omega) = 1 - h_{\omega}(x) = 1 - \sigma(\omega^T x)$$

- It can be compactly written as

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

Scikit-Learn for Logistic Regression

```
from sklearn import linear_model

clf = linear_model.LogisticRegression(solver='lbfgs')
clf.fit(X,np.ravel(y))
```

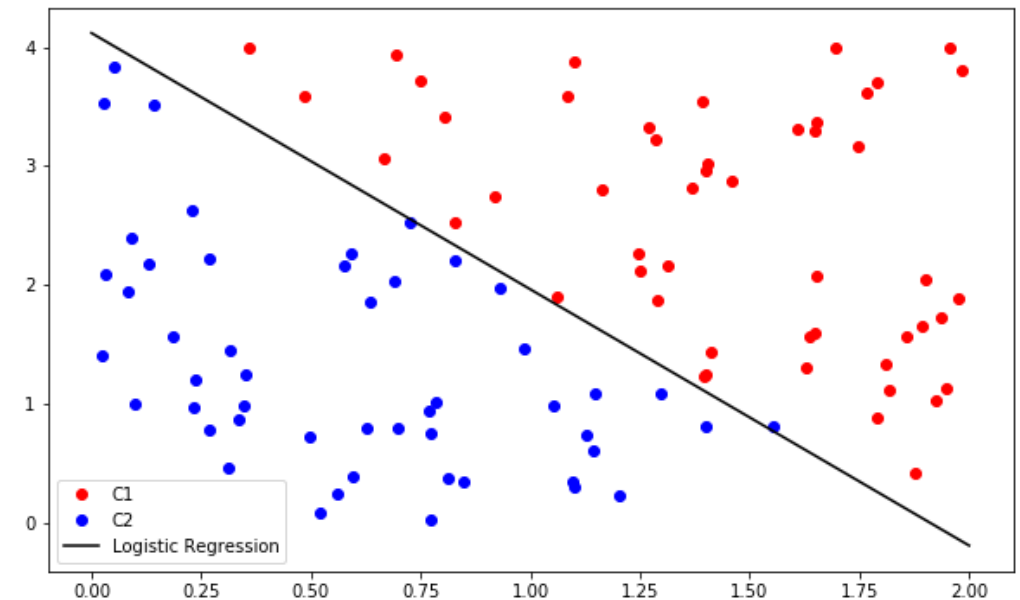
```
w1 = clf.coef_[0,0]
w2 = clf.coef_[0,1]
w0 = clf.intercept_[0]

xp = np.linspace(0,2,100).reshape(-1,1)
yp = - w1/w2*xp - w0/w2

plt.figure(figsize = (10,6))
plt.plot(X[C1,0], X[C1,1], 'ro', label='C1')
plt.plot(X[C2,0], X[C2,1], 'bo', label='C2')
plt.plot(xp, yp, 'k', label='Logistic Regression')
plt.legend()
plt.show()
```

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

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

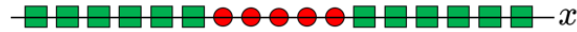


Multiclass Classification

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

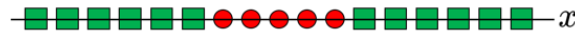
Classifying Non-linear Separable Data

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

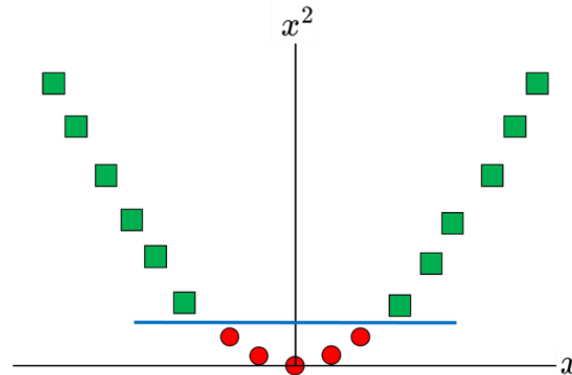


Classifying Non-linear Separable Data

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



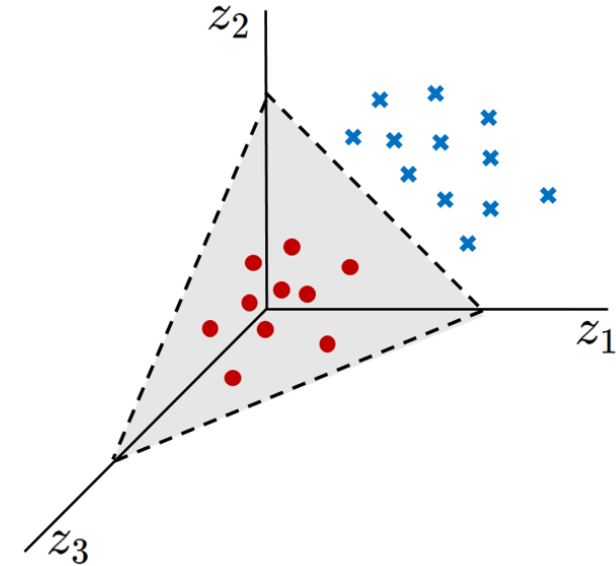
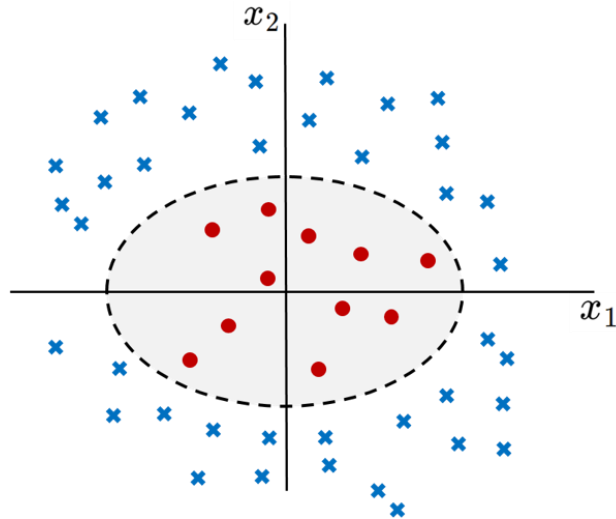
- Now map each example as $x \rightarrow \{x, x^2\}$
- Data now becomes linearly separable in the new representation



- 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 not 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

SVM with a polynomial
Kernel visualization

Created by:
Udi Aharoni