Logistic Regression

Industrial AI Lab.

Linear Classification: Logistic Regression

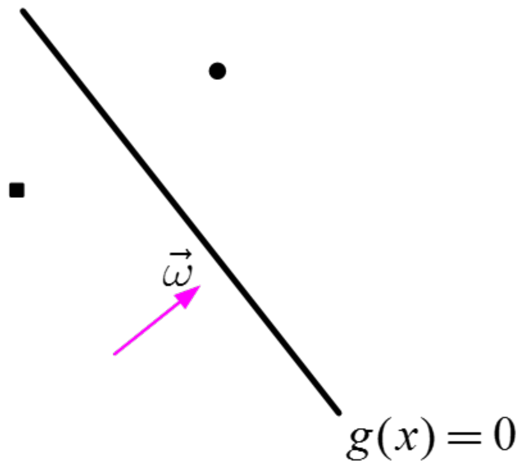
- Logistic regression is a classification algorithm
  - don't be confused

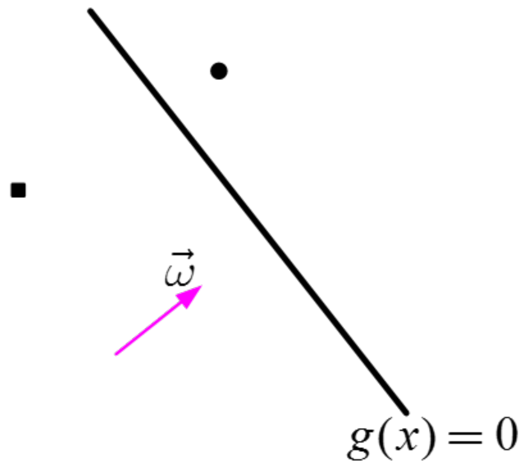
- Perceptron: make use of sign of data

- SVM: make use of margin (minimum distance)
  - Distance from a single data point

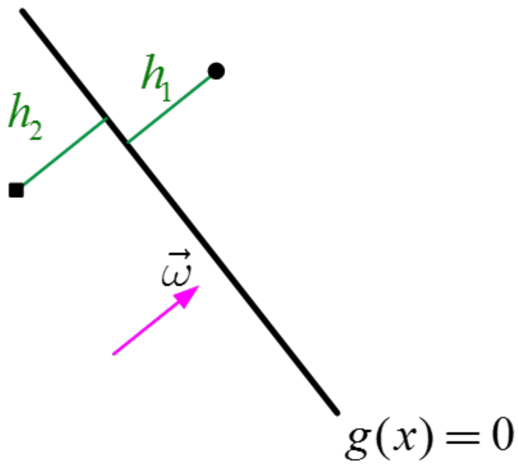
- We want to use distance information of all data points
  - logistic regression

Using Distances

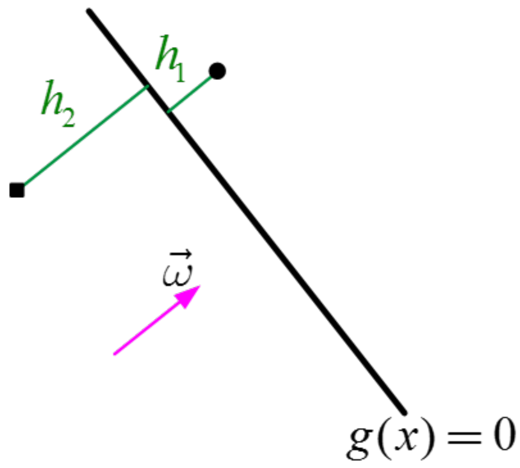




Using Distances

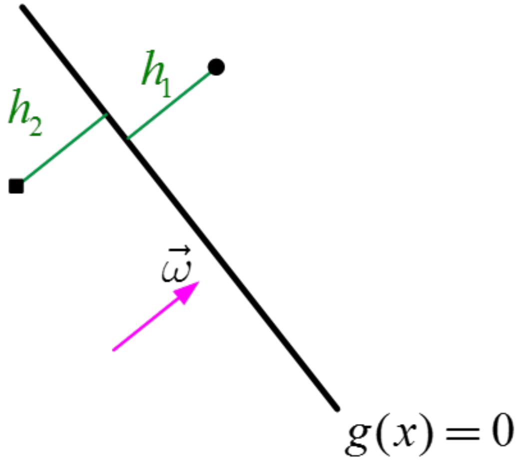


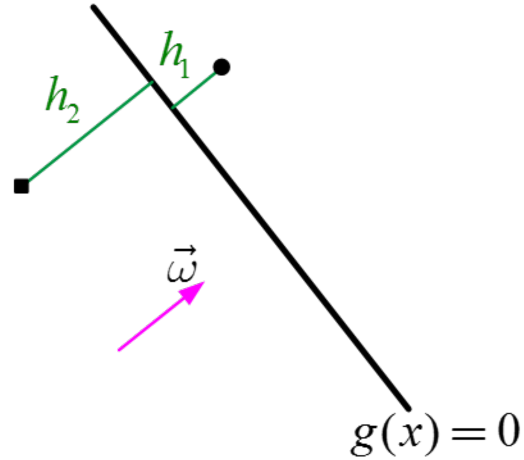
$$|h_1| + |h_2|$$



$$|h_1| + |h_2|$$

Using Distances





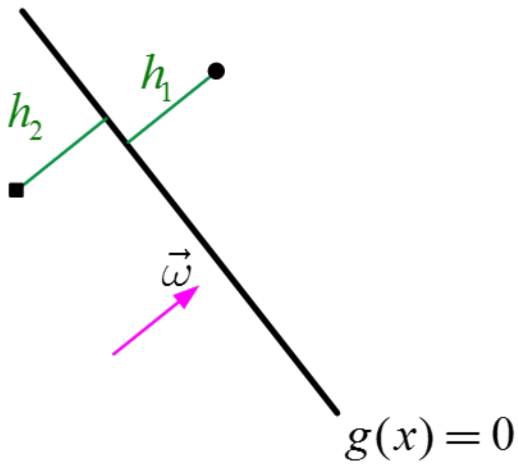
$$|h_1| + |h_2|$$

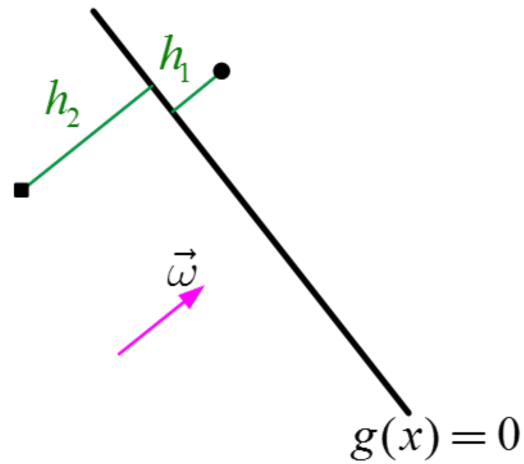
$$|h_1| \cdot |h_2|$$

$$|h_1| + |h_2|$$

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

Using Distances





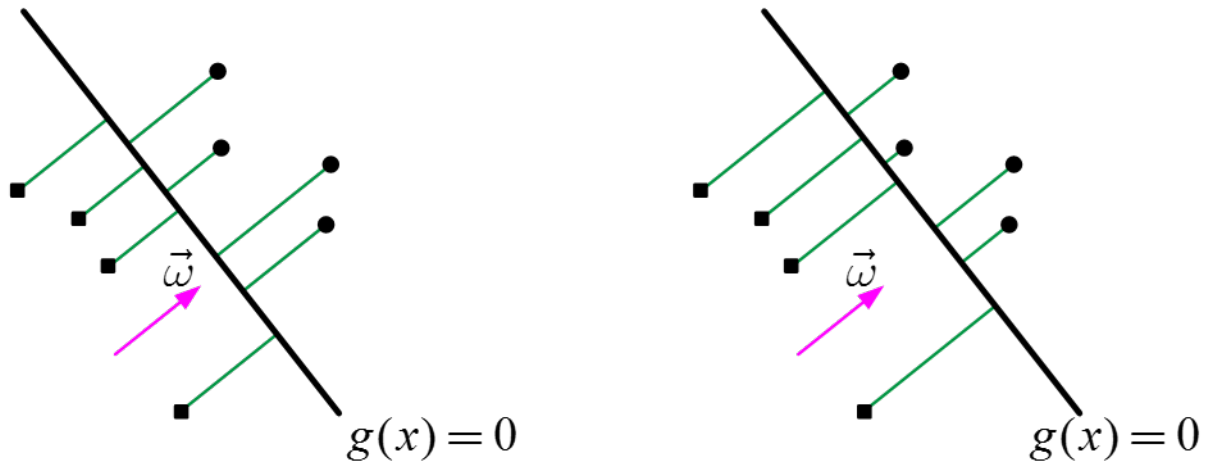
$|h_1| + |h_2|$ $\qquad$ $|h_1| + |h_2|$

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

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

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

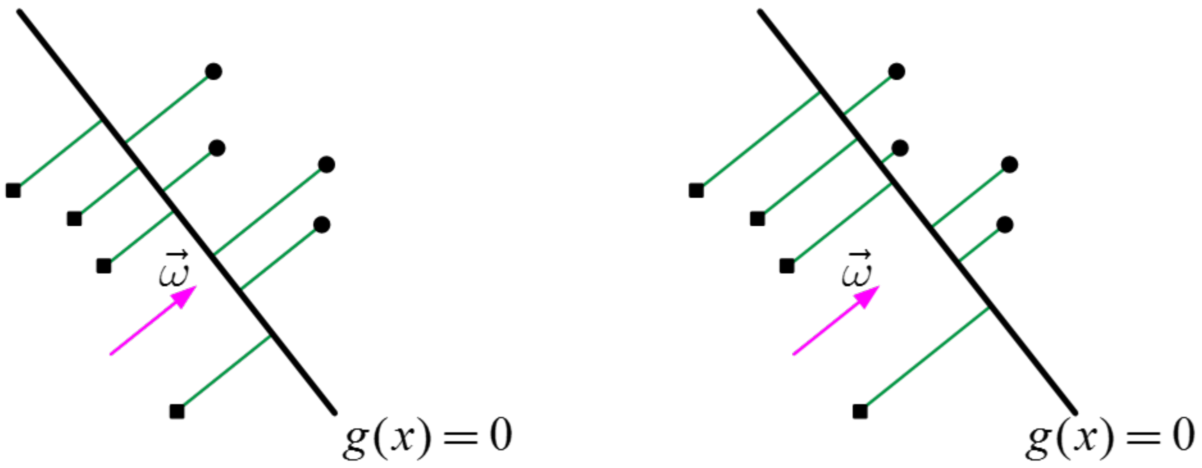


  – Inequality of arithmetic and geometric means

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

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

Using all Distances

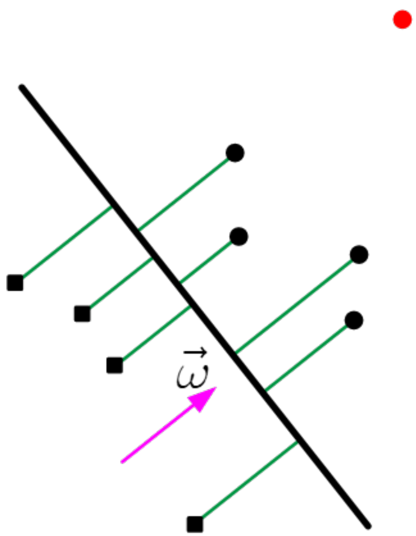


- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes

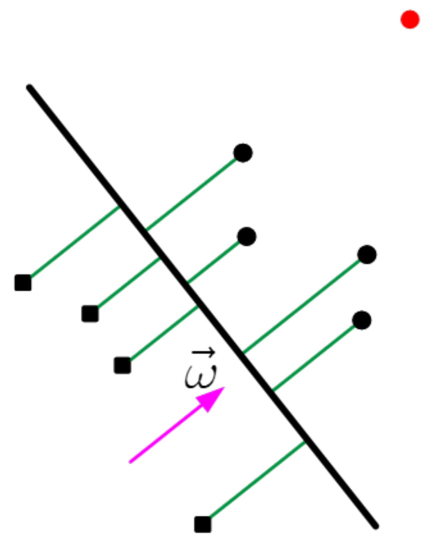
$$h = \frac{g(x)}{||\omega||} = \frac{\omega^T x}{||\omega||} \sim \omega^T x$$

Using all Distances with Outliers

- SVM vs. Logistic Regression



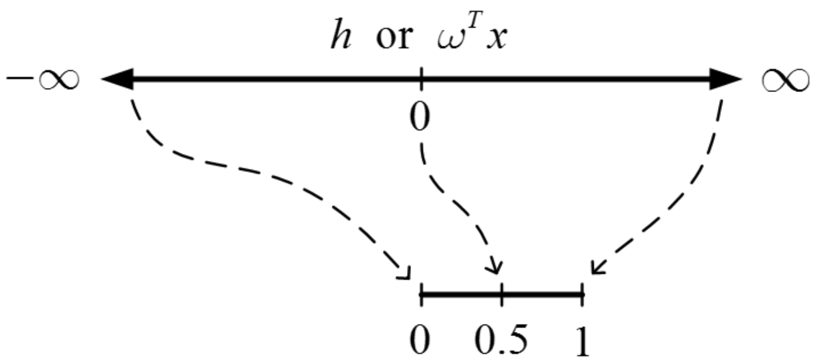
SVM



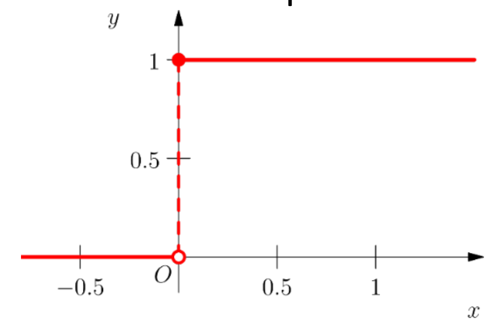
Logistic Regression

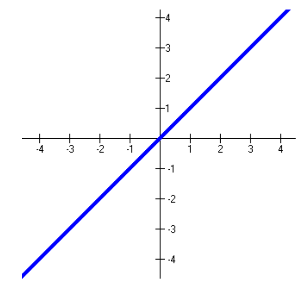
Sigmoid Function

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

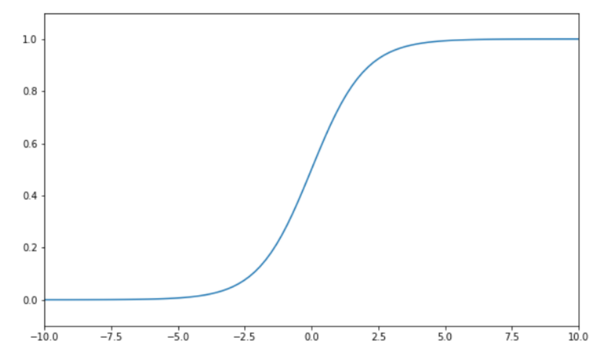


Step function





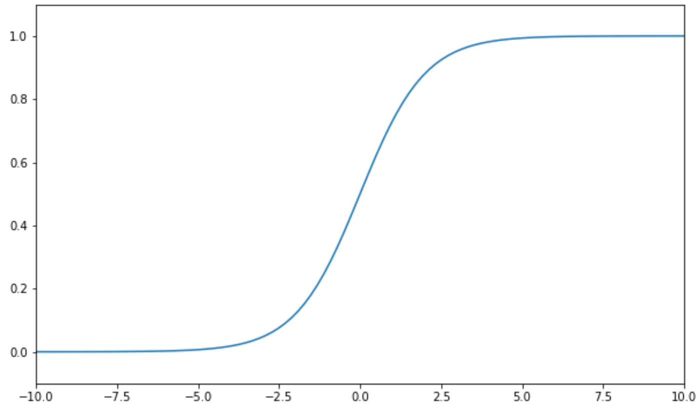
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid Function

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

%matplotlib inline
```

```
z = np.linspace(-10,10,100)
s = 1/(1+np.exp(-z))

plt.figure(figsize=(10,6))
plt.plot(z, s)
plt.xlim([-10, 10])
plt.ylim([-0.1, 1.1])
plt.show()
```



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

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

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

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

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

  - Often we do note care about predicting the label y
  - Rather, we want to predict the label probabilities $P(y|x,\omega)$
    - 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 our data

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

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

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

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

In Matrix Form

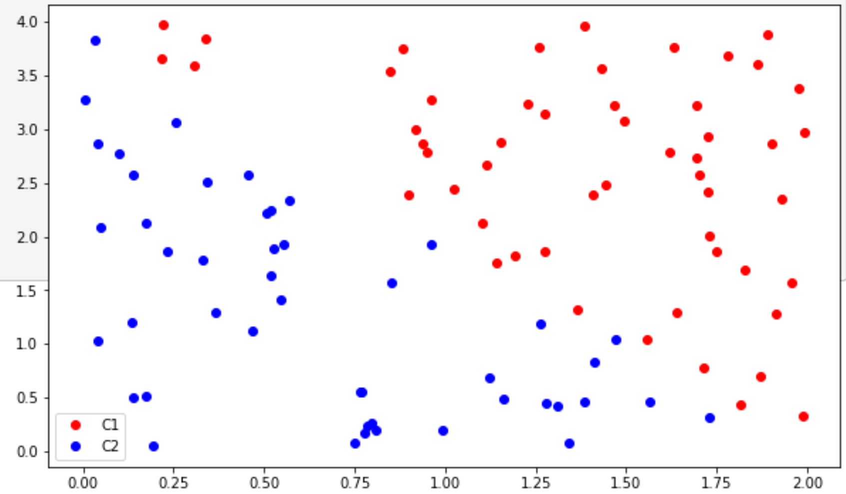
$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

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

Source: Section 7.1.1
from http://cvxr.com/cvx/examples/cvxbook/Ch07_statistical_estim/html/logistics.html

Data Generation

```
m = 100
w = np.array([[-4], [2], [1]])
X = np.hstack([np.ones([m,1]), 2*np.random.rand(m,1), 4*np.random.rand(m,1)])

w = np.asmatrix(w)
X = np.asmatrix(X)

y = (np.exp(X*w)/(1+np.exp(X*w))) > 0.5

C1 = np.where(y == True)[0]
C2 = np.where(y == False)[0]

y = np.empty([m,1])
y[C1] = 1
y[C2] = 0
y = np.asmatrix(y)

plt.figure(figsize = (10,6))
plt.plot(X[C1,1], X[C1,2], 'ro', label='C1')
plt.plot(X[C2,1], X[C2,2], 'bo', label='C2')
plt.legend()
plt.show()
```



Log Likelihood

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

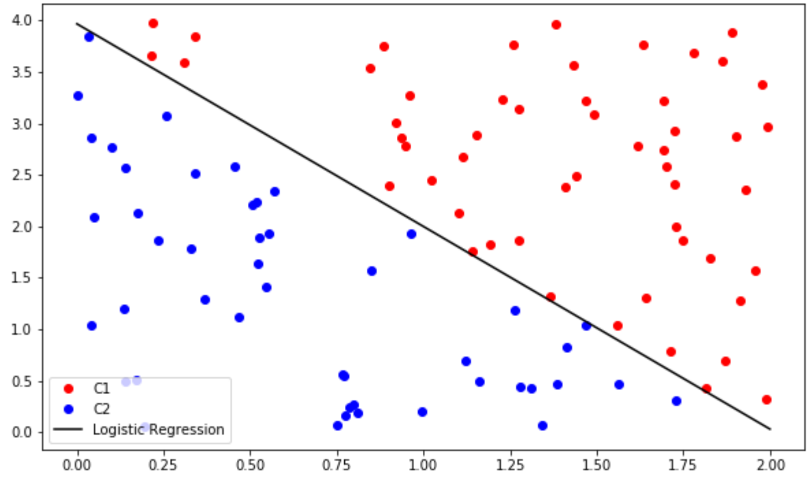
- Refer to [cvx functions]
  - scalar function: `cvx.sum_entries(x)` = $\sum_{ij} x_{ij}$
  - elementwise function: `cvx.logistic(x)` = $\log(1 + e^x)$

CVXPY

```
import cvxpy as cvx

w = cvx.Variable(3, 1)

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

w = w.value

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

plt.figure(figsize = (10,6))
plt.plot(X[C1,1], X[C1,2], 'ro', label='C1')
plt.plot(X[C2,1], X[C2,2], 'bo', label='C2')
plt.plot(xp, yp, 'k', label='Logistic Regression')
plt.legend()
plt.show()
```

$$\texttt{cvx.logistic(x)} = \log(1 + e^x)$$

$$\sum_{i=1}^{q} (\omega^T x_i) - \sum_{i=1}^{m} \log\left(1 + \exp\left(\omega^T x_i\right)\right)$$



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

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

In a More Compact Form

- MLE solution

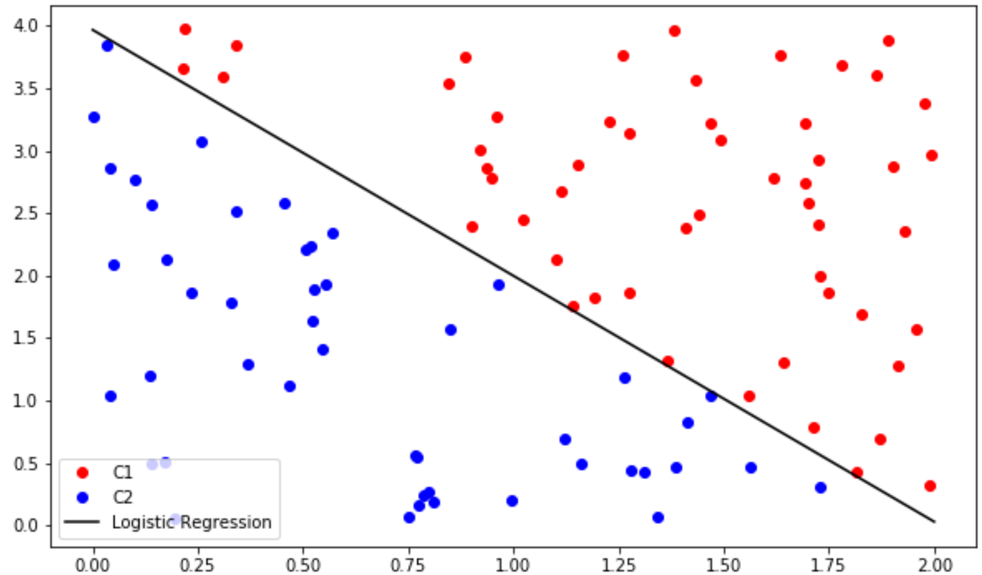
$$
\begin{aligned}
\hat{\omega} &= \arg\max_{\omega} \sum_{n=1}^{m} - \log\left(1 + \exp\left(-y_n \omega^T x_n\right)\right) \\
&= \arg\min_{\omega} \sum_{n=1}^{m} \log\left(1 + \exp\left(-y_n \omega^T x_n\right)\right)
\end{aligned}
$$

CVXPY

```
y = np.empty([m,1])
y[C1] = 1
y[C2] = -1
y = np.asmatrix(y)

w = cvx.Variable(3, 1)

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

w = w.value
```

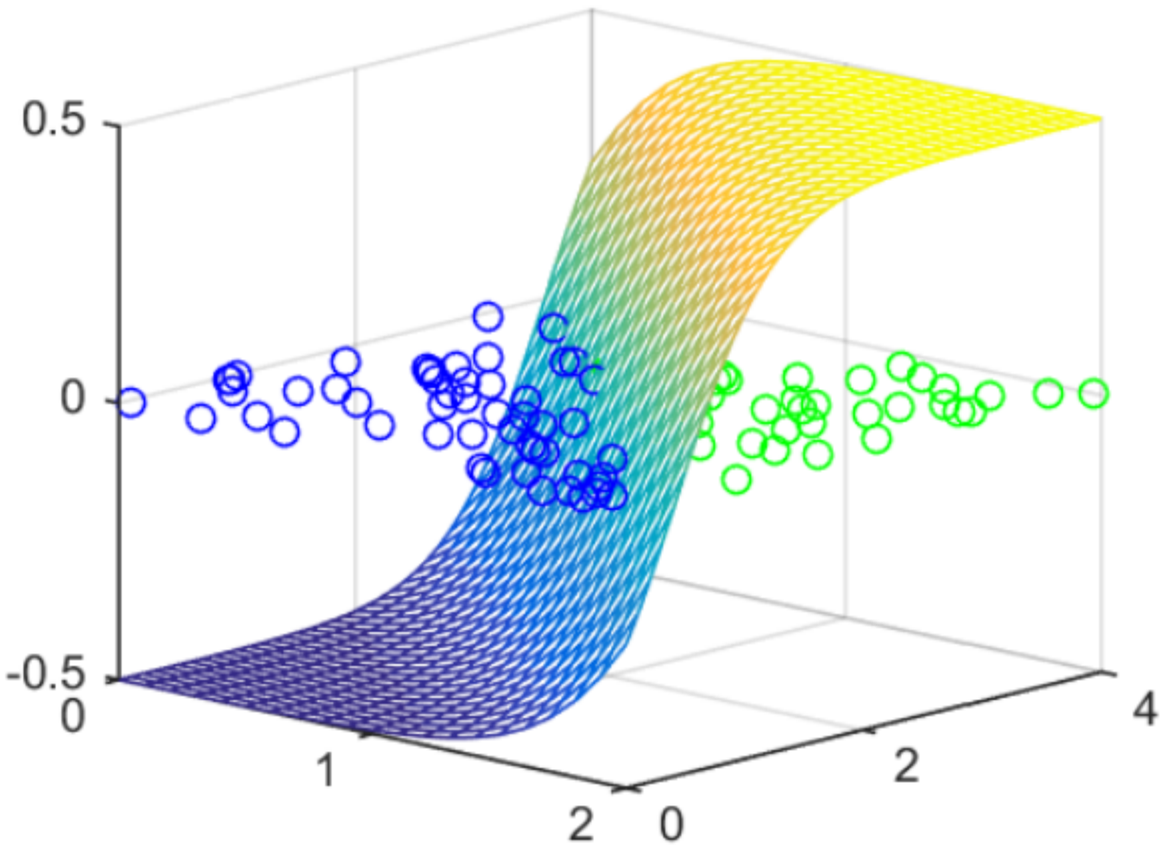
$$\texttt{cvx.logistic(x)} = \log(1 + e^x)$$

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



Logistic Regression

- Classified based on probability



Multiclass Classification

- 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)
  - see them as probability

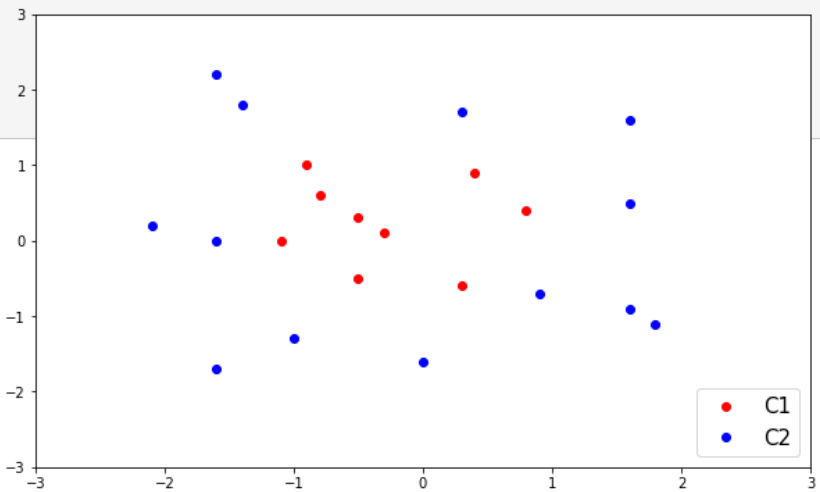
$$P\left(y=k \mid x, \omega\right)=\frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)} \in [0,1]$$

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

Non-linear Classification

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

```
X1 = np.array([[-1.1,0],[-0.3,0.1],[-0.9,1],[0.8,0.4],[0.4,0.9],[0.3,-0.6],
               [-0.5,0.3],[-0.8,0.6],[-0.5,-0.5]])

X2 = np.array([[-1,-1.3], [-1.6,2.2],[0.9,-0.7],[1.6,0.5],[1.8,-1.1],[1.6,1.6],
               [-1.6,-1.7],[-1.4,1.8],[1.6,-0.9],[0,-1.6],[0.3,1.7],[-1.6,0],[-2.1,0.2]])

X1 = np.asmatrix(X1)
X2 = np.asmatrix(X2)

plt.figure(figsize=(10, 6))
plt.plot(X1[:, 0], X1[:,1], 'ro', label='C1')
plt.plot(X2[:, 0], X2[:,1], 'bo', label='C2')
plt.axis([-3,3,-3,3])
plt.legend(loc = 4, fontsize = 15)
plt.show()
```



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 = X2.shape[0]

X = np.vstack([X1, X2])
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_entries(cvx.logistic(-cvx.mul_elemwise(y,Z*w))))
prob = cvx.Problem(obj).solve()

w = w.value
```

Non-linear Classification