Logistic Regression

Industrial AI Lab.

Sooyoung Lee

Linear Classification: Logistic Regression

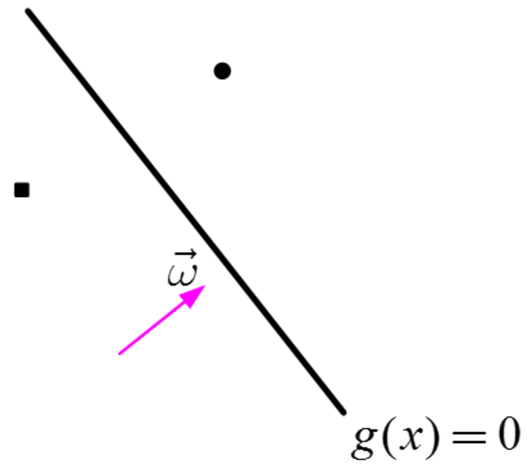
- Logistic regression is a classification algorithm
  - don't be confused

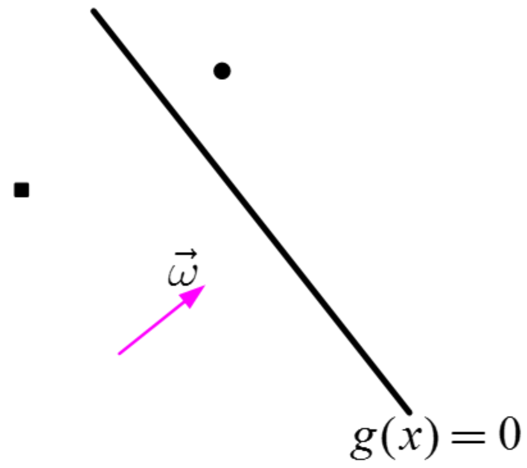
- Perceptron: make use of sign of data

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

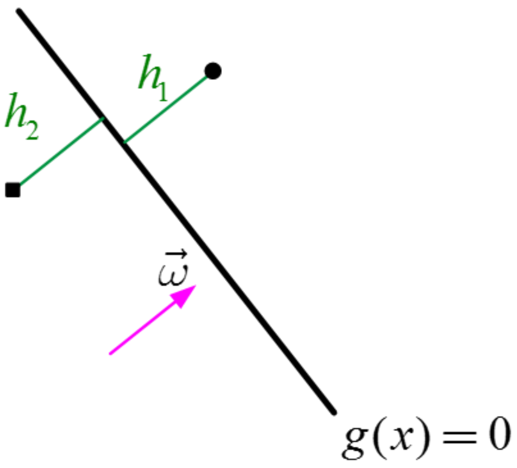
- We want to use distance information of all data points
  - logistic regression

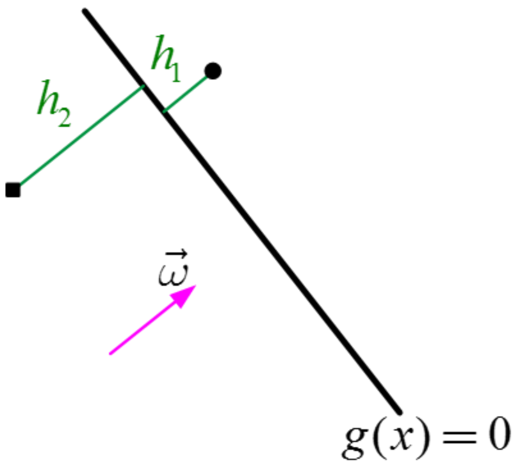
Using Distances





Using Distances

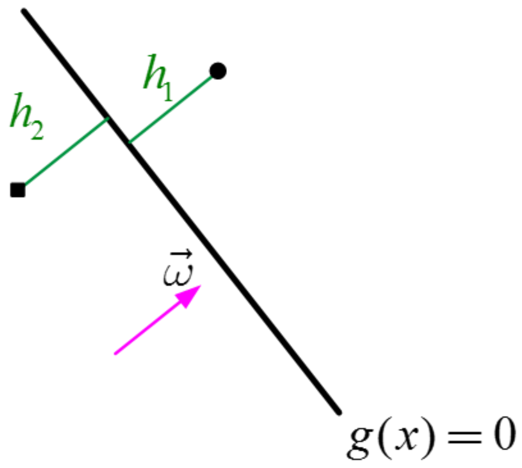


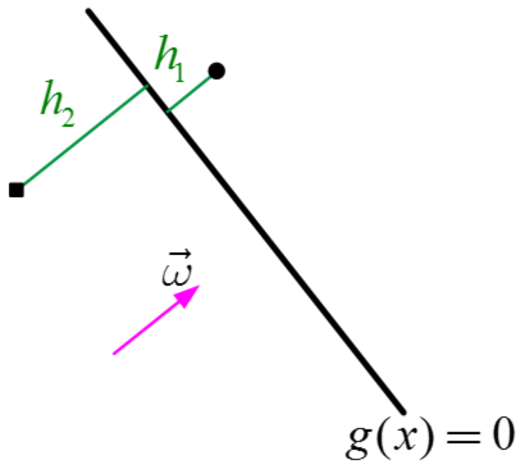


$$|h_1| + |h_2|$$

$$|h_1| + |h_2|$$

Using Distances





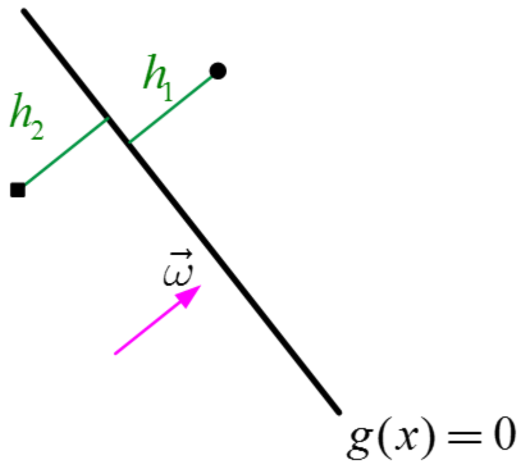
$|h_1| + |h_2|$

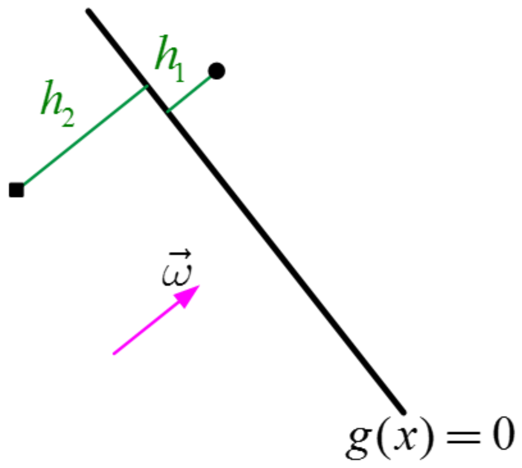
$|h_1| \cdot |h_2|$

$|h_1| + |h_2|$

$|h_1| \cdot |h_2|$

Using Distances





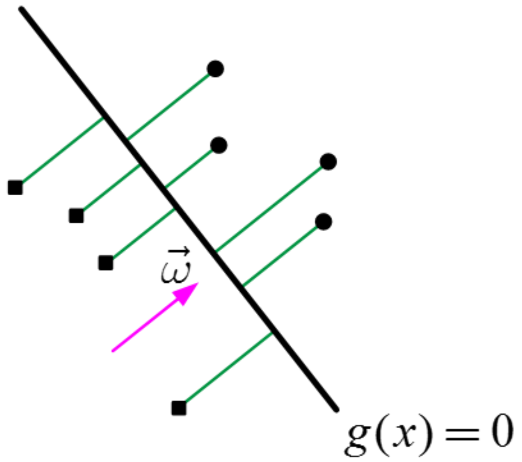
$$|h_1| + |h_2| \qquad |h_1| + |h_2|$$

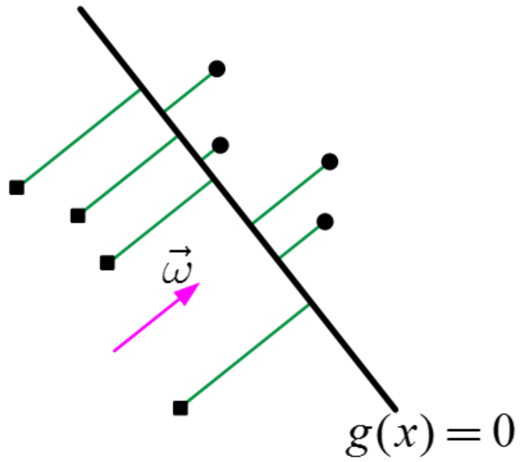
$$|h_1| \cdot |h_2| \qquad |h_1| \cdot |h_2|$$

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

Using all Distances

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



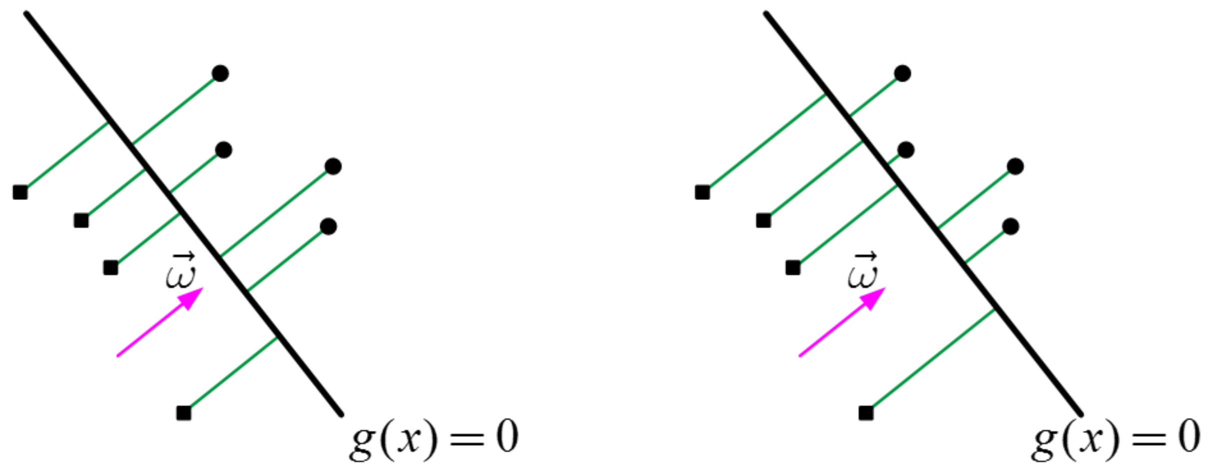


  – Inequality of arithmetic and geometric means

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

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

Using all Distances

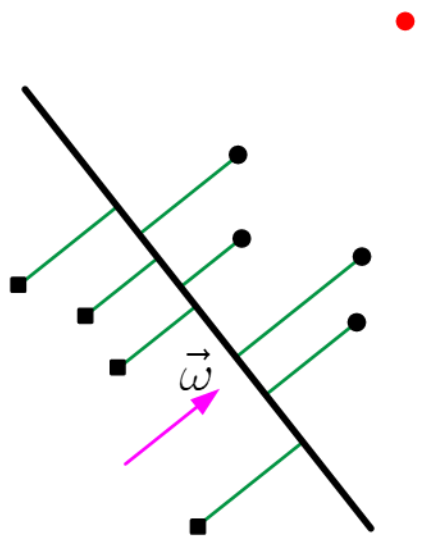


- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes

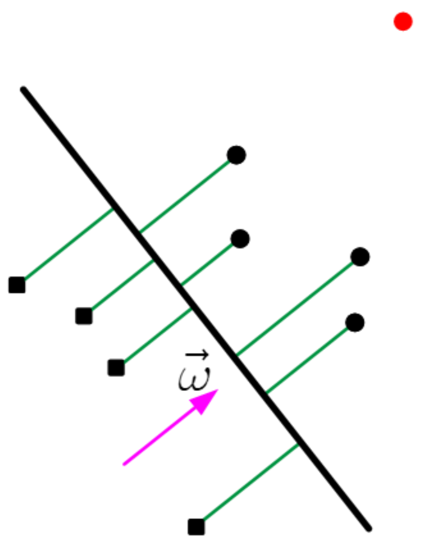
$$h = \frac{g(x)}{||\omega||} = \frac{\omega^T x}{||\omega||} \sim \omega^T x$$

Using all Distances with Outliers

- SVM vs. Logistic Regression



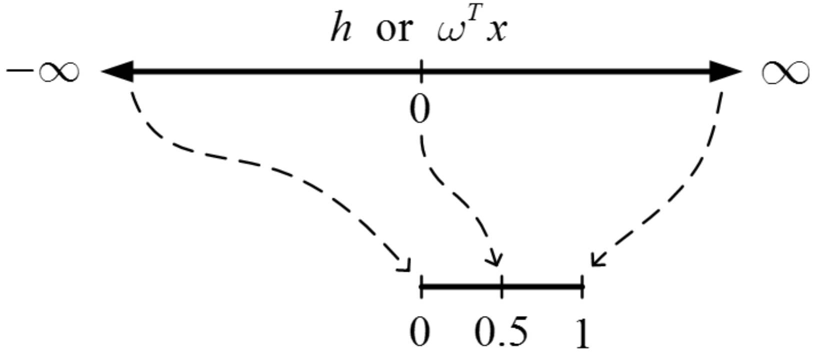
SVM



Logistic Regression

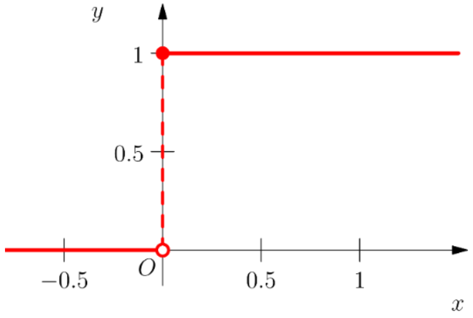
Sigmoid Function

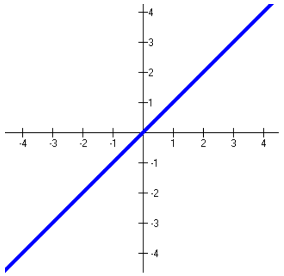
- We link or squeeze $(-\infty, +\infty)$ to $(0, 1)$ for several reasons:

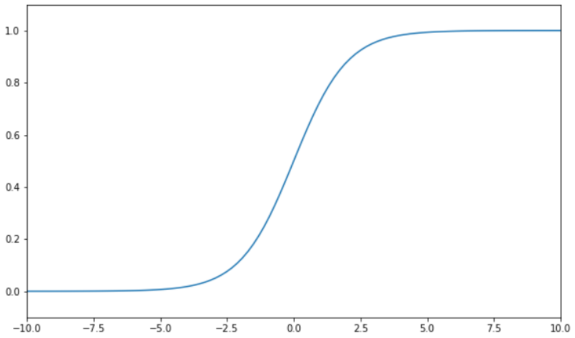


$$\sigma(z) = \frac{1}{1+e^{-z}} \implies \sigma\left(\omega^T x\right) = \frac{1}{1+e^{-\omega^T x}}$$

Step function





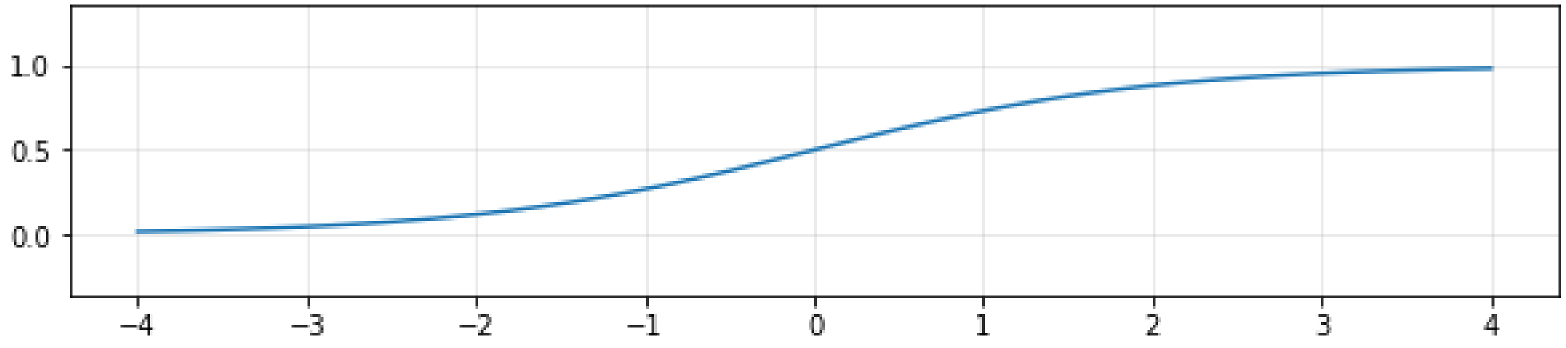


Sigmoid Function

- $\sigma(z)$ 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()
```

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



POSTECH

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

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

  - 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

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

- 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:

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

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 $\dfrac{\partial \ell(\omega)}{\partial \omega_j}$

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

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

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

Gradient Descent for Logistic Regression

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

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

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

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

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

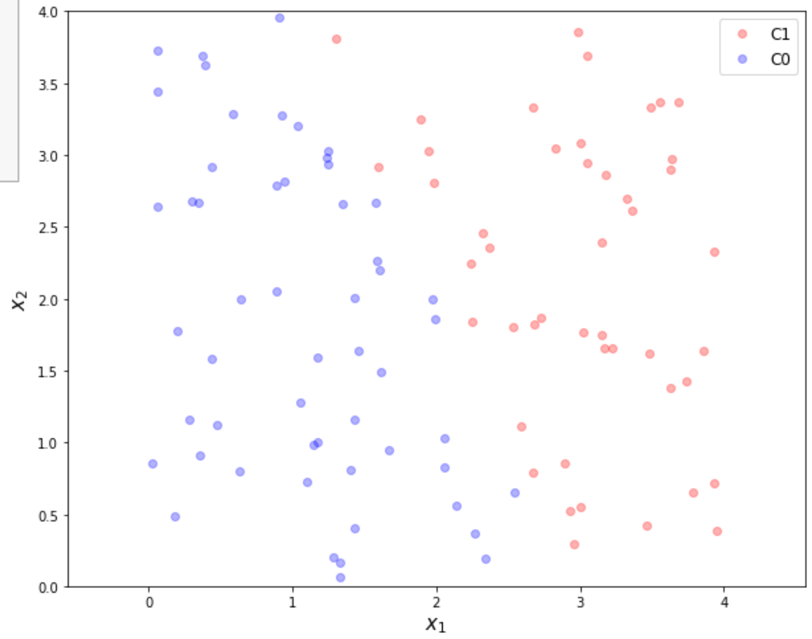
$$\nabla \ell(\omega) = \begin{bmatrix} \frac{\partial \ell(\omega)}{\partial \omega_0} \\ \frac{\partial \ell(\omega)}{\partial \omega_1} \\ \frac{\partial \ell(\omega)}{\partial \omega_2} \end{bmatrix} = X^T (y - h_\omega(x)) = X^T (y - \sigma(X\omega))$$

- Maximization problem
- Be careful on matrix shape

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

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

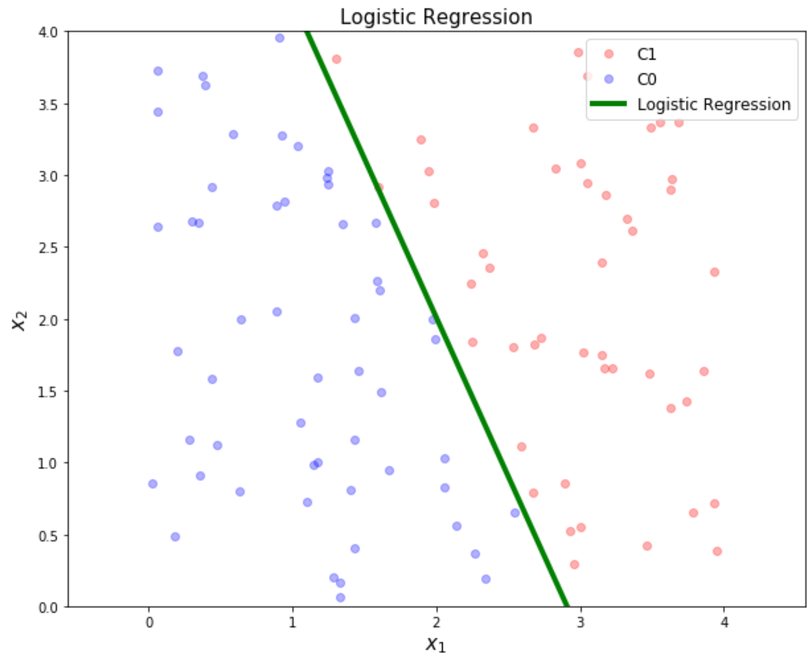
```python
# be careful with matrix shape

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

```python
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)
```

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

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

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



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

$$\begin{aligned} p &= \frac{1}{1 + e^{-\omega^T x}} = \frac{e^{\omega^T x}}{e^{\omega^T x} + 1} \\ 1 - p &= \frac{1}{e^{\omega^T x} + 1} \end{aligned}$$

- 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

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

CVXPY Implementation

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

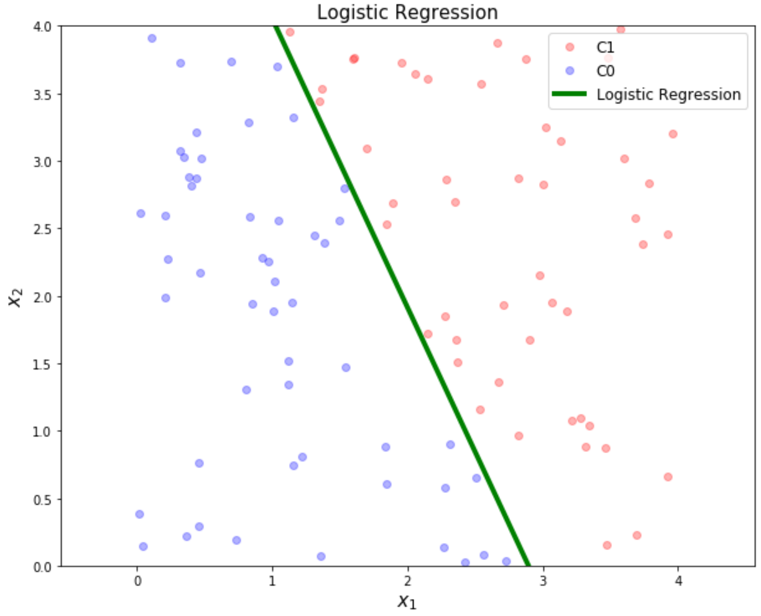
$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

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

```
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

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

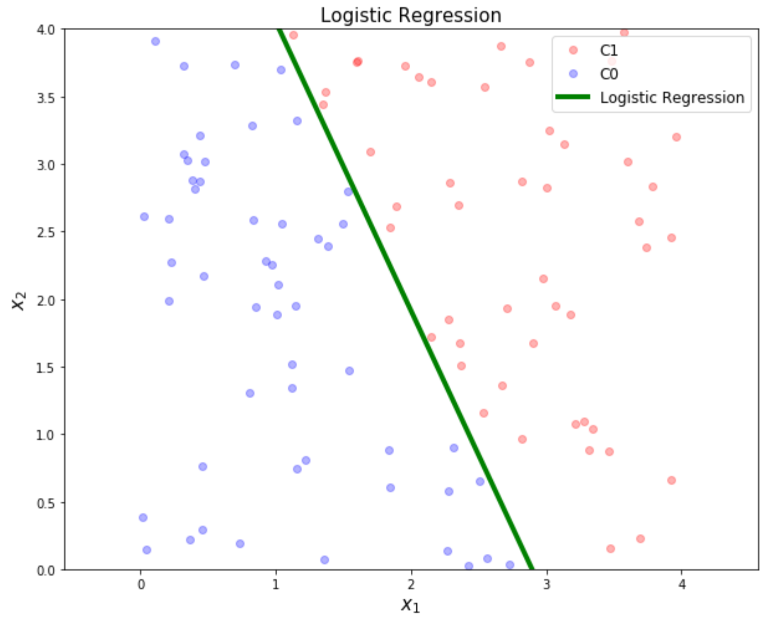
  – Log-likelihood

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

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

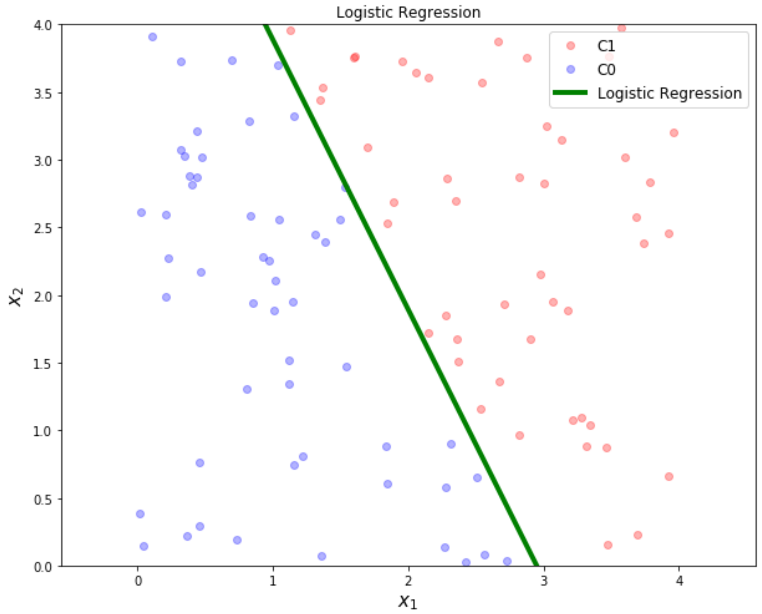
```python
X = X[:,1:3]

X.shape
```

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

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

```python
from sklearn import linear_model

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

```python
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

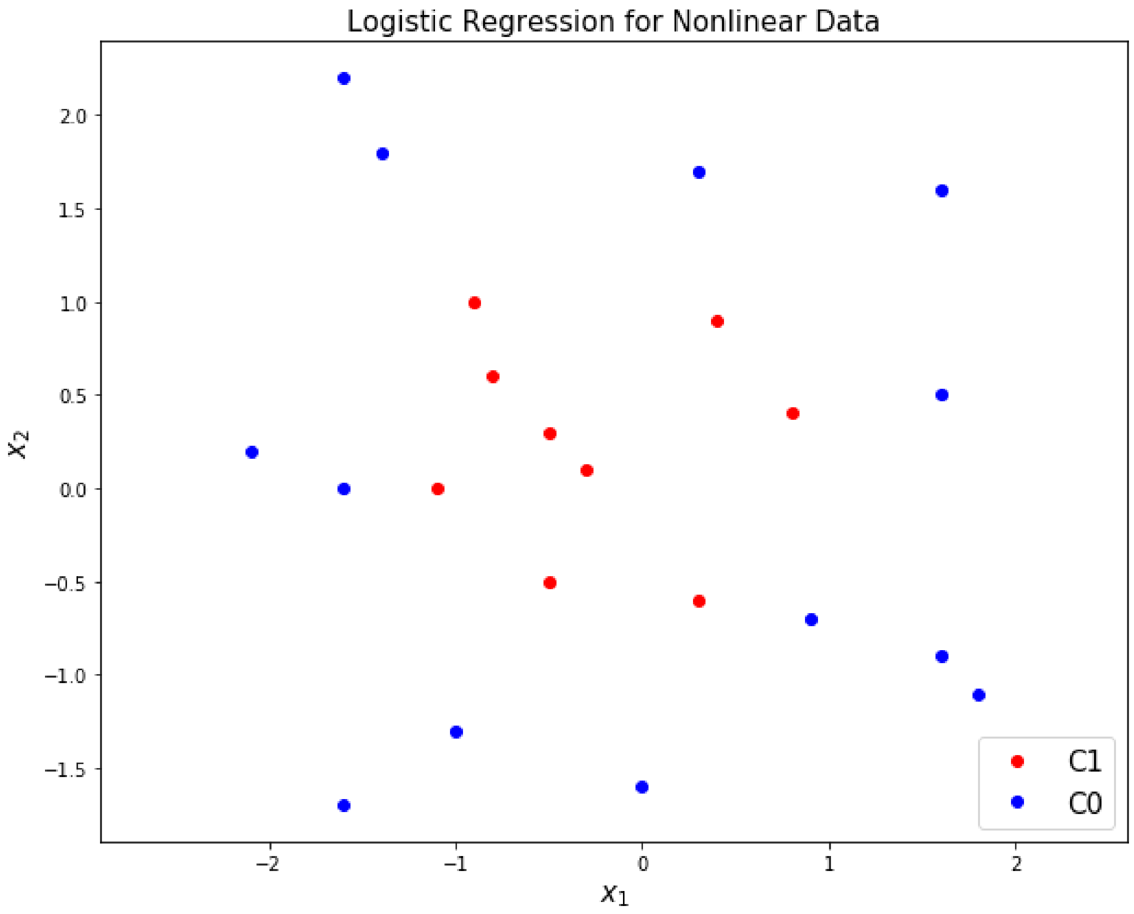
$$P(y = k \mid x, \omega) = \frac{\exp\left(\omega_k^T x\right)}{\sum_k \exp\left(\omega_k^T x\right)} \in [0, 1]$$

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

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