CIS603 - Artificial Intelligence

Logistic regression

Vasileios Megalooikonomou

(some material adopted from notes by M. Hauskrecht)

Supervised learning

Data: $D = \{d_1, d_2, .., d_n\}$ **a set of *n* examples**

$d_i = < \mathbf{x}_i, y_i >$

$\mathbf{x}_i$ is input vector, and *y* is desired output (given by a teacher)

Objective: learn the mapping $f : X \rightarrow Y$

s.t. $y_i \approx f(x_i)$ for all $i = 1,.., n$

Two types of problems:

- **Regression:** Y is **continuous**
  Example: earnings, product orders → company stock price
- **Classification:** Y is **discrete**
  Example: temperature, heart rate → disease

Now: BINARY **classification problems**

Binary classification

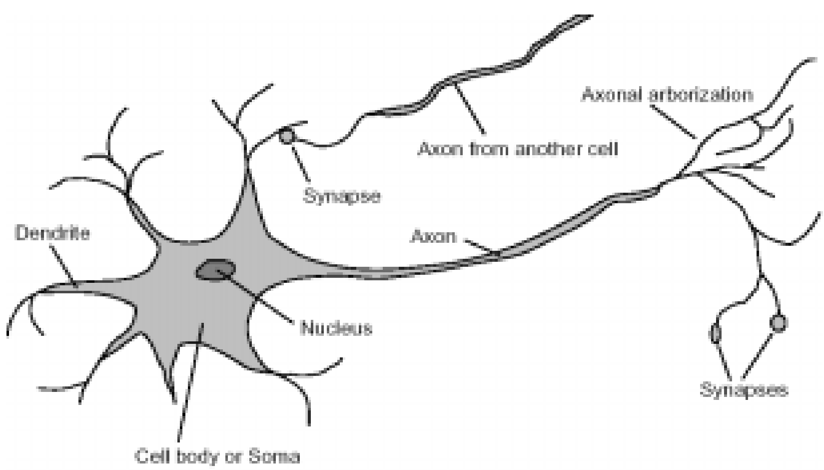
- Two classes $Y = \{0,1\}$
- Our goal is to learn to classify correctly two types of examples
  - Class 0 – labeled as 0,
  - Class 1 – labeled as 1
- We would like to learn $f : X \to \{0,1\}$

- **First step:** we need to devise a model of the function *f*

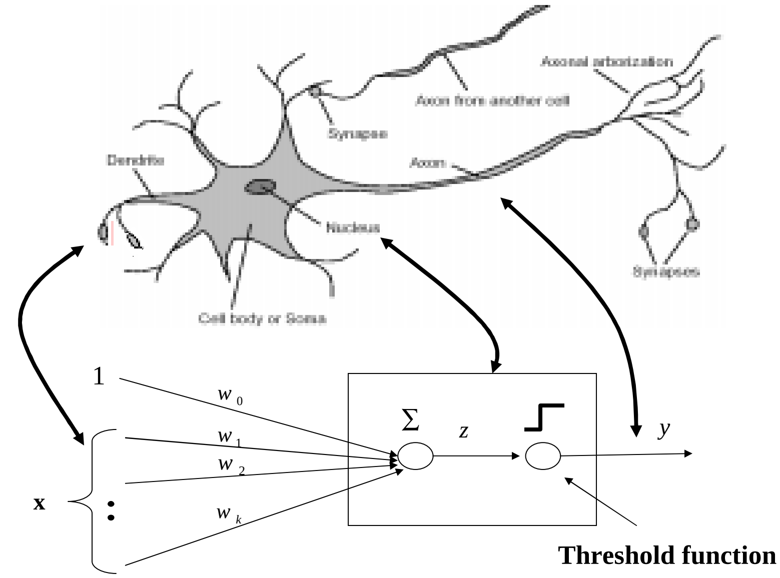
- **Inspiration:** ***neuron (nerve cells)***

Neuron

- **neuron (nerve cell) and its activities**



Neuron-based binary classification model



Binary classification

- Instead of learning the mapping to discrete values 0,1

$$f : X \rightarrow \{0,1\}$$

- It is easier to learn a probabilistic function

$$f' : X \rightarrow [0,1]$$

  - where *f'* describes the probability of a class 1 given x

$$p(y = 1 \mid \mathbf{x})$$

- Transformation to discrete class values:

If $p(y = 1 \mid \mathbf{x}) \geq 1/2$ then choose **1**
Else choose **0**

- **Logistic regression model** uses a probabilistic function

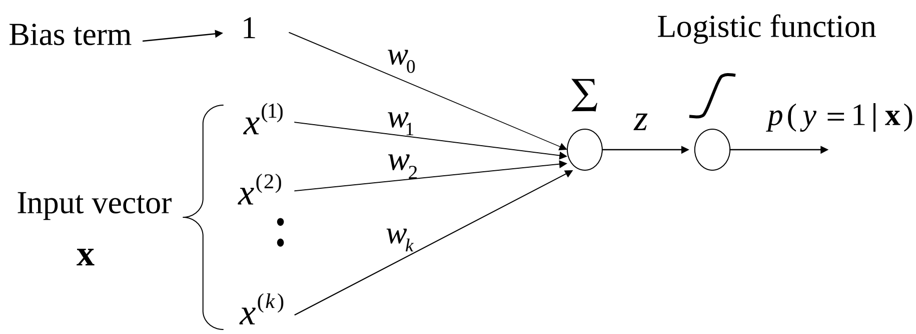
Logistic regression

- **Logistic regression:**

  $p(y=1 \mid \mathbf{x}, \mathbf{w}) = g(z) = g(w_0 + w_1 x^{(1)} + \ldots w_k x^{(k)})$

  where ***w*** are parameters of the models

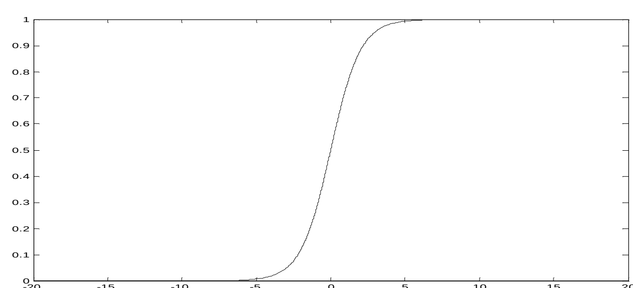
  and *g(z)* is a **logistic function** $g(z) = 1/(1+e^{-z})$



Logistic function

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

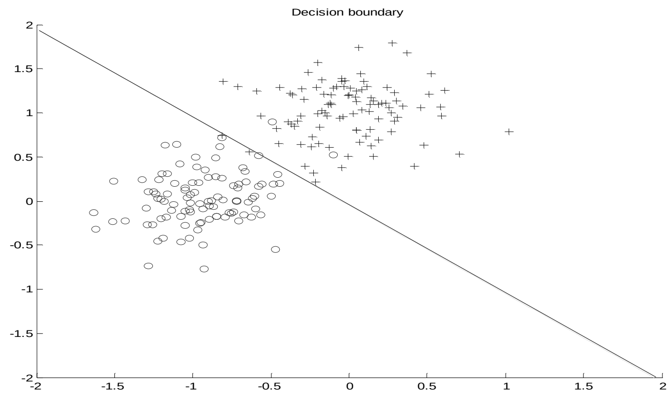
- also referred to as sigmoid function
- replaces threshold function with smooth switching
- takes a real number and outputs the number in the interval [0,1]



Logistic regression - Decision boundary

Logistic regression model defines a linear decision boundary

- Example: 2 classes (crosses and circles)



Binary classification - Error

- Two classes $Y = \{0,1\}$
- Our goal is to classify correctly as many examples as possible
- Zero-one error function

$$Error\ (x_i, y_i) = \begin{cases} 1 & f(\mathbf{x}_i, \mathbf{w}) \neq y_i \\ 0 & f(\mathbf{x}_i, \mathbf{w}) = y_i \end{cases}$$

- Error we would like to minimize: $E_{(x,y)}(Error\ (x, y))$
- The error is minimized if we choose:

$$y = 1 \quad \text{if} \quad p(y = 1 \mid \mathbf{x}, \mathbf{w}) > p(y = 0 \mid \mathbf{x}, \mathbf{w})$$
$$y = 0 \quad \text{otherwise}$$

- We construct a probabilistic version of the error function based on the **likelihood of the data** $L(D, \mathbf{w}) = P(D \mid \mathbf{w})$

Inverse optimization problem $Error\ (D, \mathbf{w}) = -L(D, \mathbf{w})$

Logistic regression: parameter learning

- **Likelihood of data** $L(D,\mathbf{w}) = P(D \mid \mathbf{w}) = \prod_{i=1}^{n} \mu_i^{y_i}(1-\mu_i)^{1-y_i}$

  where $d_i =< \mathbf{x}_i, y_i >$

  $$\mu_i = p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = g(z_i) = g(w_o + \sum_{j=1}^{k} w_j x_i^{(j)})$$

  We want weights w that maximize the likelihood of data
- **Trick: maximize the log-likelihood of data instead**

  $$l(D,\mathbf{w}) = \log \prod_{i=1}^{n} \mu_i^{y_i}(1-\mu_i)^{1-y_i} = \sum_{i=1}^{n} \log \mu_i^{y_i}(1-\mu_i)^{1-y_i} =$$

  $$= \sum_{i=1}^{n} y_i \log \mu_i + (1-y_i)\log(1-\mu_i) = \sum_{i=1}^{n} -J_{\text{online}}(d_i,\mathbf{w})$$

- Rational: The optimal weights are the same for both the likelihood and the log-likelihood

Logistic regression: parameter estimation

- **Log likelihood**

  $$l(D,\mathbf{w}) = \sum_{i=1}^{n} -J_{\text{online}}(d_i,\mathbf{w}) = \sum_{i=1}^{n} y_i \log\mu_i + (1-y_i)\log(1-\mu_i)$$

- **On-line component of the log-likelihood**

  $$-J_{\text{online}}(d_i,\mathbf{w}) = y_i \log \mu_i + (1-y_i)\log(1-\mu_i)$$
- **Derivatives of the online error component (in terms of weights)**

  $$\frac{\partial}{\partial w_0} J_{\text{online}}(d_i,\mathbf{w}) = -(y_i - g(z_i))$$

  $$\vdots$$

  $$\frac{\partial}{\partial w_j} J_{\text{online}}(d_i,\mathbf{w}) = -x_i^{(j)}(y_i - g(z_i))$$

Logistic regression. Online gradient.

- We want to find the set of parameters optimizing the log-likelihood of data or minimizing the error
- **On-line learning update for weight w** $J_{online}(d_i, \mathbf{w})$

$$w \leftarrow w^* - \alpha \frac{\partial}{\partial w}[J_{online}(d_i, w)|_{w^*}]$$

- **(i+1)th update for the logistic regression** and $d =< \mathbf{x}, y >$

$$w_0^{(i+1)} \leftarrow w_0^{(i)} + \alpha(i+1)(y - g(w_0 + \sum_{u=1}^{k} w_u x^{(u)}))$$

$$\vdots$$

$$w_j^{(i+1)} \leftarrow w_j^{(i)} + \alpha(i+1)(y - g(w_0 + \sum_{u=1}^{k} w_u x^{(u)}))x^{(j)}$$

α - annealed learning rate (depends on the number of updates)

<u>The same, easy update rule as used in the linear regression !!!</u>

Online logistic regression algorithm

Online-logistic-regression (*D, number of iterations*)
  initialize weights $w_0, w_1, w_2 \ldots w_k$
  for i=1:1*: number of iterations*
    do **select** a data point d=<$\mathbf{x}$,y> from D
      set $\alpha = 1/i$
      update weights (in parallel)

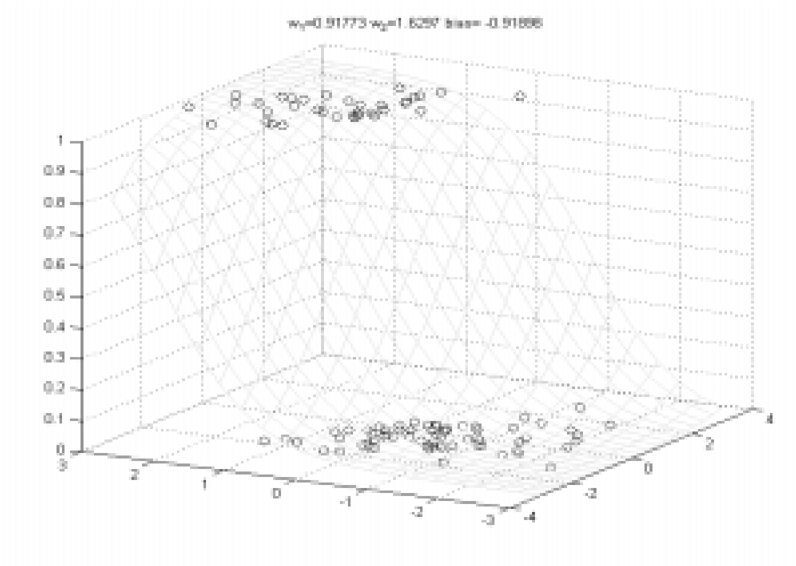
$$w_0 = w_0 + \alpha[y - p(y=1|\mathbf{x},\mathbf{w})]$$

$$\vdots$$

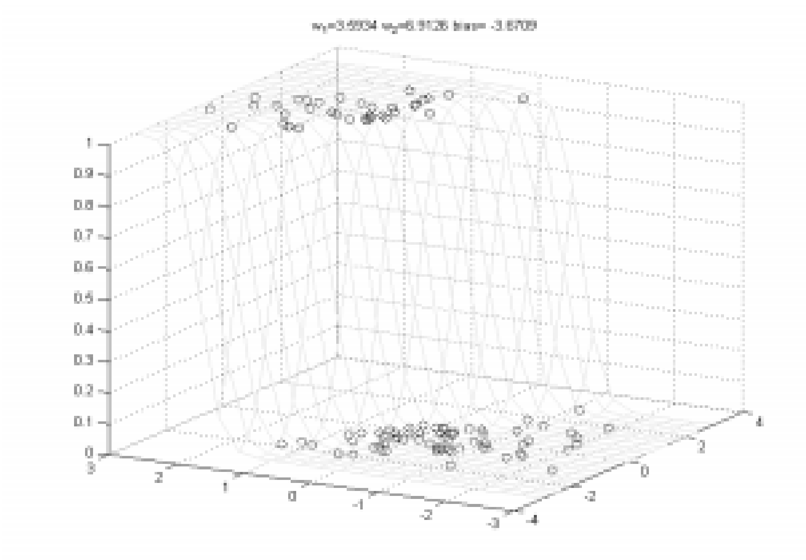
$$w_j = w_j + \alpha[y - p(y=1|\mathbf{x},\mathbf{w})]x^{(j)}$$

  end for
  return weights

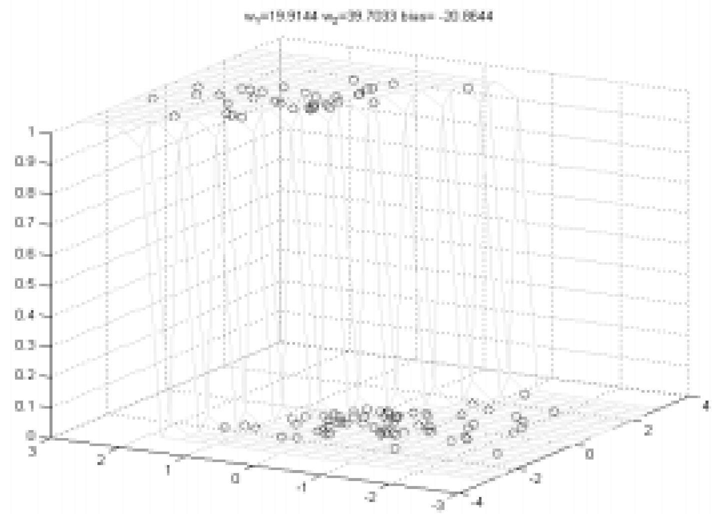
Online algorithm. Example.



Online algorithm. Example.

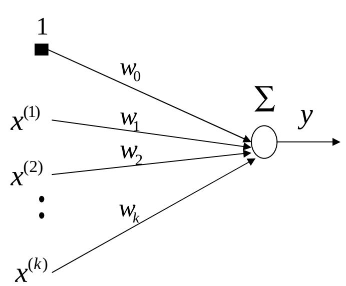


Online algorithm. Example.



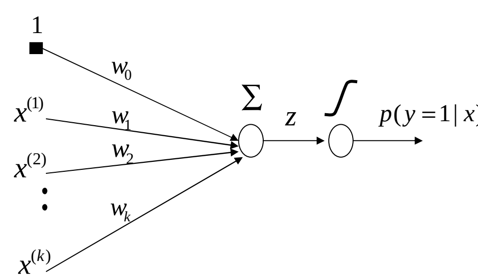
Limitations of basic linear units

Linear regression



Function linear in inputs

Logistic regression

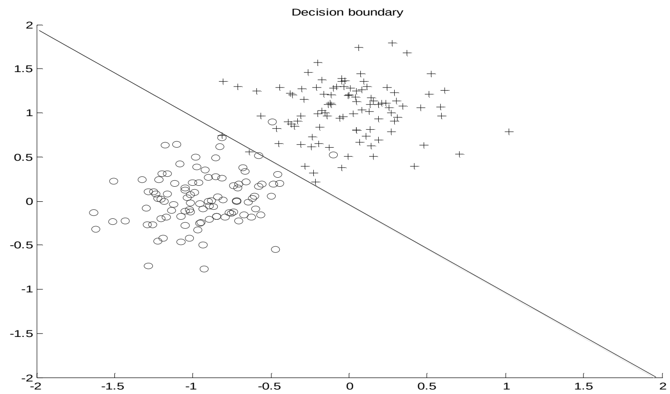


Linear decision boundary

Logistic regression - Decision boundary

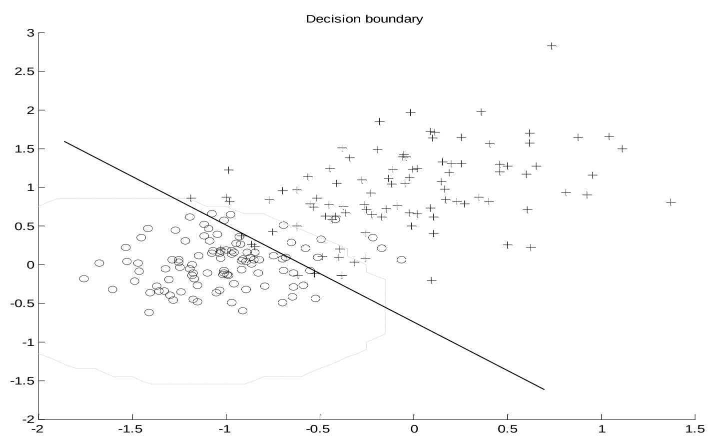
Logistic regression model defines a linear decision boundary

- Example: 2 classes (crosses and circles)



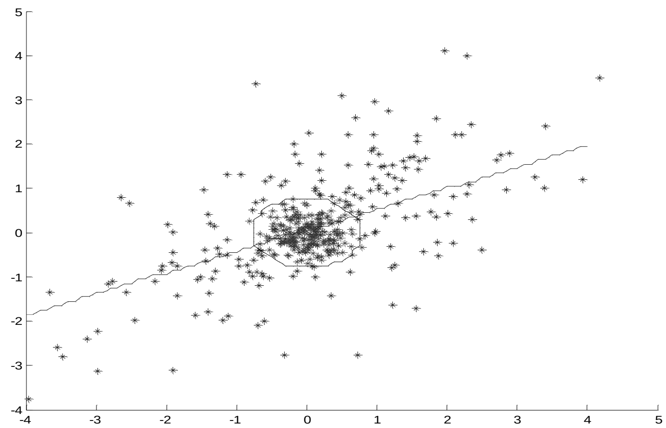
Linear decision boundary

- Example when logistic regression model is not optimal, but not that bad



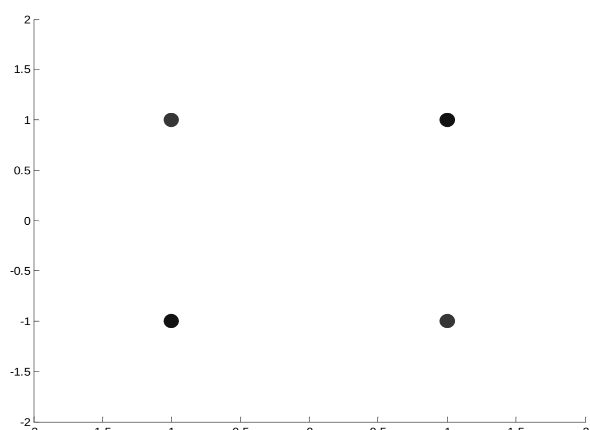
When logistic regression fails?

- Example in which the logistic regression model fails



Limitations of logistic regression.

- **parity function -** no linear decision boundary

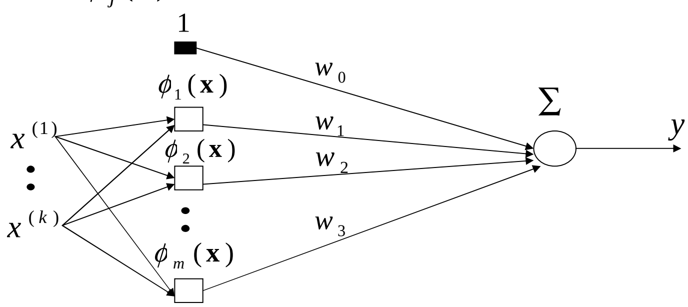


Extensions of simple linear units

Replace inputs to linear units with **feature (basis) functions** to model **nonlinearities**

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

$\phi_j(\mathbf{x})$ - an arbitrary function of **x**



The same trick can be done for the logistic regression

Extension of simple linear units

- **Example: Fitting of a polynomial of degree m**
  - **Data points: pairs of** $< x, y >$
  - **Feature functions:**

$$\phi_i(x) = x^i$$

  - **Function to learn:**

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i x^i$$

  - **On line update** for <*x*,*y*> pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x}, \mathbf{w}))$$

$$\vdots$$

$$w_i = w_i + \alpha(y - f(\mathbf{x}, \mathbf{w}))x^i$$

Multi-layered neural networks

- Alternative way to introduce nonlinearities to regression/classification models
- **Idea:** Cascade several simple neural models (based on logistic regression). Much like neuron connections.