CS 1571 Introduction to AI
Lecture 27

Logistic regression

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Supervised learning

Data: $D = \{D_1, D_2, .., D_n\}$ **a set of *n* examples**

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

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \cdots x_{i,d})$ is an input vector of size d

y_i is the desired output (given by a teacher)

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

s.t. $y_i \approx f(\mathbf{x}_i)$ for all $i = 1, .., n$

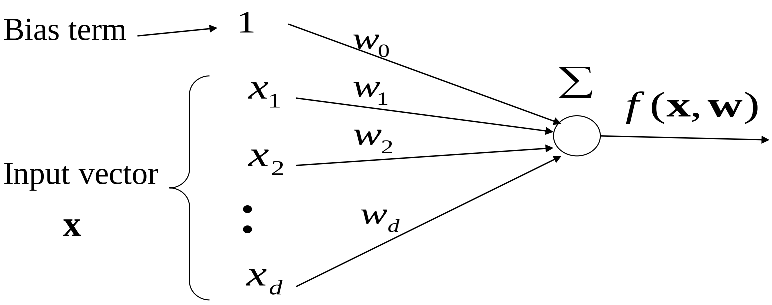
- **Regression:** Y is **continuous**
  Example: earnings, product orders $\rightarrow$ company stock price
- **Classification:** Y is **discrete**
  Example: handwritten digit in binary form $\rightarrow$ digit label

Linear regression: review

- **Function** $f : X \to Y$ is a linear combination of input components

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j$$

$w_0, w_1, \ldots w_k$ - **parameters (weights)**



Linear regression: review

- **Data:** $D_i = < \mathbf{x}_i, y_i >$
- **Function:** $\mathbf{x}_i \to f(\mathbf{x}_i)$
- We would like to have $y_i \approx f(\mathbf{x}_i)$ for all $i = 1,..,n$

- **Error function** measures how much our predictions deviate from the desired answers

Mean-squared error $J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$

- **Learning:**

  We want to find the weights minimizing the error !

Solving linear regression: review

- The optimal set of weights satisfies:

$$\nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \bar{\mathbf{0}}$$

Leads to a **system of linear equations (SLE)** with *d+1* unknowns of the form

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

$$w_0\sum_{i=1}^{n}x_{i,0}x_{i,j} + w_1\sum_{i=1}^{n}x_{i,1}x_{i,j} + \ldots + w_j\sum_{i=1}^{n}x_{i,j}x_{i,j} + \ldots + w_d\sum_{i=1}^{n}x_{i,d}x_{i,j} = \sum_{i=1}^{n}y_i x_{i,j}$$

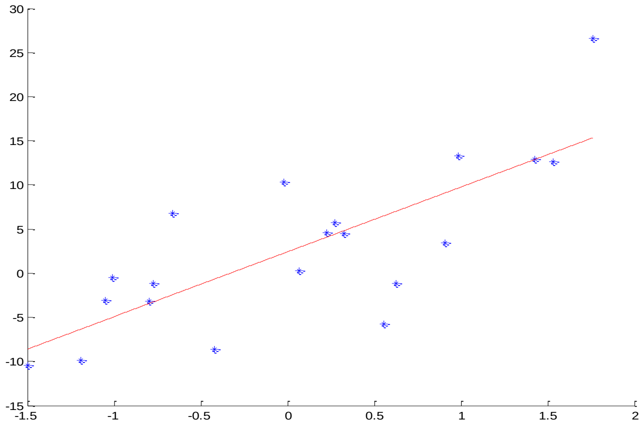
Solutions to SLE:

- e.g. matrix inversion (if the matrix is singular)

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$$

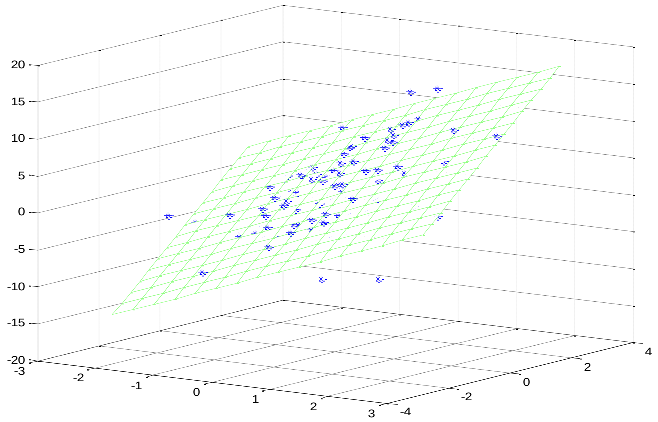
Linear regression. Example

- 1 dimensional input $\mathbf{x} = (x_1)$



Linear regression. Example.

- 2 dimensional input $\mathbf{x} = (x_1, x_2)$



Gradient descent solution

- There are other ways to solve the weight optimization problem in the linear regression model

$$J_n = Error(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- A simple technique:
  - **Gradient descent**

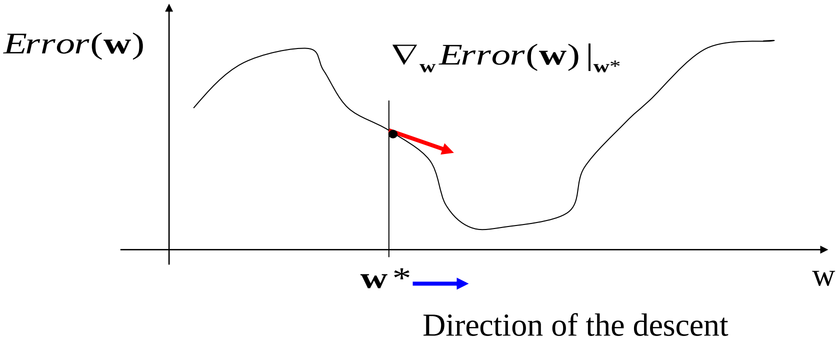
    Idea:
    - Adjust weights in the direction that improves the Error
    - The gradient tells us what is the right direction

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient descent method

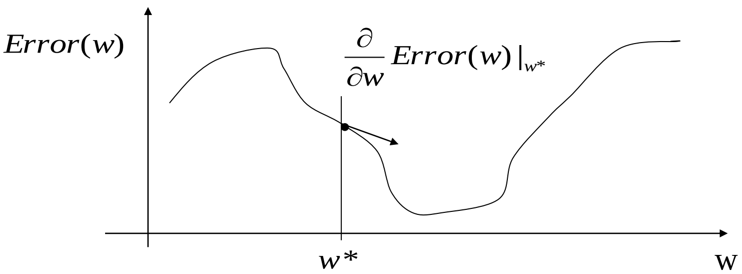
- Descend using the gradient information



- Change the value of **w** according to the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

Gradient descent method



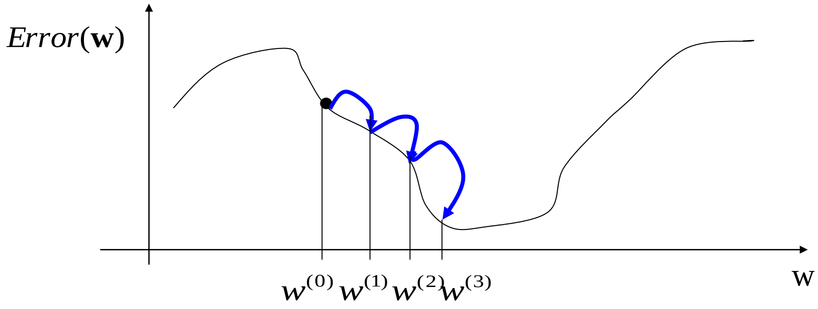
- New value of the parameter

$$w_j \leftarrow w_j^* - \alpha \frac{\partial}{\partial w_j} Error(w)|_{w^*} \qquad \text{For all j}$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient descent method

- Iteratively converge to the optimum of the Error function



Online regression algorithm

- **Batch learning:** error function defined for the whole dataset D

$$J_n = Error(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- **On-line learning:** Error function for individual datapoints.
  - Useful when we learn on datastreams

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

$$J_{\text{online}} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- **Change regression weights after every example according to the gradient of the online error:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

Gradient for on-line learning

Linear model $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

On-line error $J_{online} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$

On-line algorithm: sequence of online updates

(i)-th update for the linear model: $D_i = < \mathbf{x}_i, y_i >$

Vector form:

$$\mathbf{w}^{(i)} \leftarrow \mathbf{w}^{(i-1)} - \alpha(i)\nabla_{\mathbf{w}} Error_i(\mathbf{w})|_{\mathbf{w}^{(i-1)}} = \mathbf{w}^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))\mathbf{x}_i$$

j-th weight:

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i)\frac{\partial Error_i(\mathbf{w})}{\partial w_j}|_{\mathbf{w}^{(i-1)}} = w_j^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))x_{i,j}$$

Annealed learning rate: $\alpha(i) \approx \frac{1}{i}$

- Gradually rescales changes in weights

Online regression algorithm

Online-linear-regression (*D, number of iterations*)

Initialize weights $\mathbf{w} = (w_0, w_1, w_2 \ldots w_d)$

for i=1:1*: number of iterations*

do **select** a data point $D_i = (\mathbf{x}_i, y_i)$ from D

set $\alpha = 1/i$

update weight vector

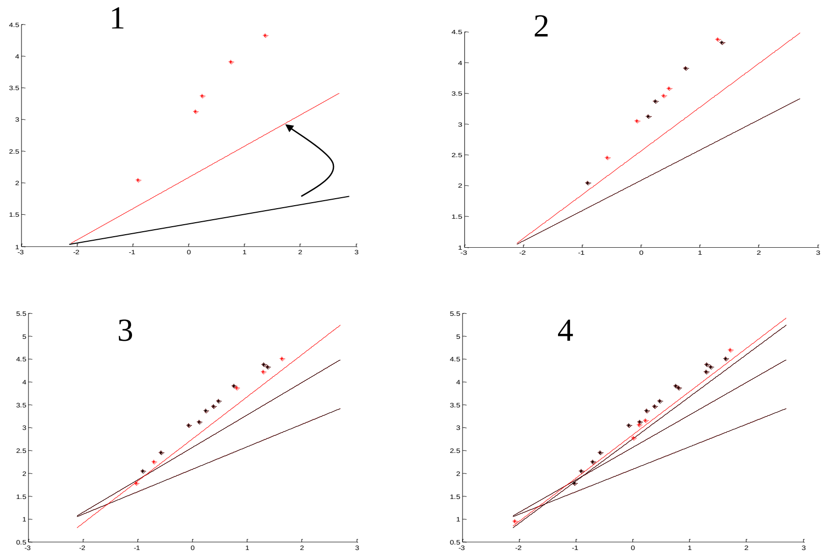
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

end for

return weights $\mathbf{w}$

- **Advantages:** very easy to implement, continuous data streams

On-line learning.   Example



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 $\rightarrow$ company stock price
- **Classification:** Y is **discrete**

  Example: temperature, heart rate $\rightarrow$ disease

Today: **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 $f : X \rightarrow \{0,1\}$
- **Zero-one error (loss) function**

$$Error_1(\mathbf{x}_i, y_i) = \begin{cases} 1 & f(\mathbf{x}_i, \mathbf{w}) \neq y_i \\ 0 & f(\mathbf{x}_i, \mathbf{w}) = y_i \end{cases}$$

- Error we would like to minimize: $E_{(x,y)}(Error_1(\mathbf{x}, y))$
- **First step:** we need to devise a model of the function

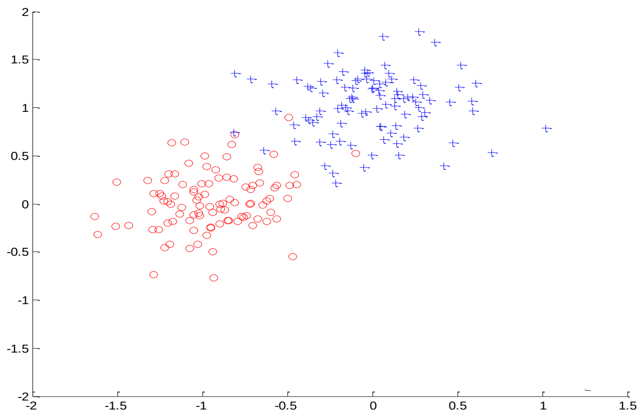
Discriminant functions

- One convenient way to represent classifiers is through
  - **Discriminant functions**
- **Works for binary and multi-way classification**

- **Idea:**
  - For every class $i = 0,1, \ldots k$ define a function $g_i(\mathbf{x})$ mapping $X \rightarrow \Re$
  - When the decision on input **x** should be made choose the class with the highest value of $g_i(\mathbf{x})$

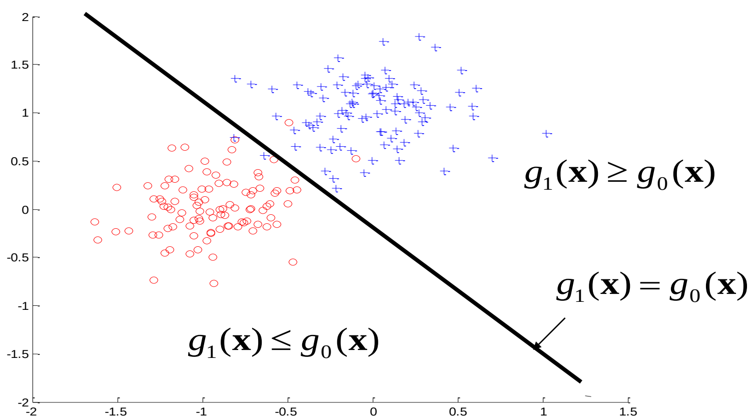
- So what happens with the input space? Assume a binary case.

Discriminant functions



Discriminant functions

- Define **decision boundary.**



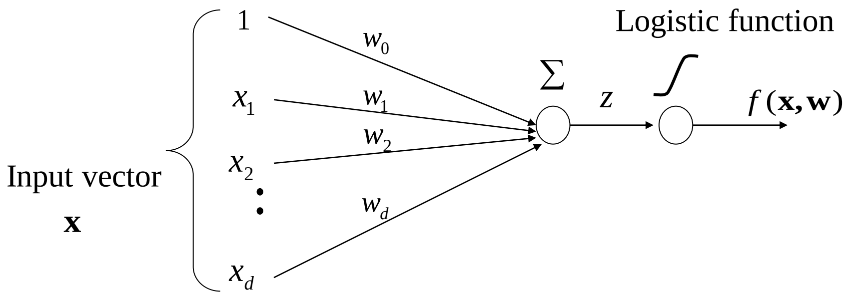
Logistic regression model

- **Defines a linear decision boundary**
- **Discriminant functions:**

$$g_1(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x}) \qquad g_0(\mathbf{x}) = 1 - g(\mathbf{w}^T\mathbf{x})$$

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

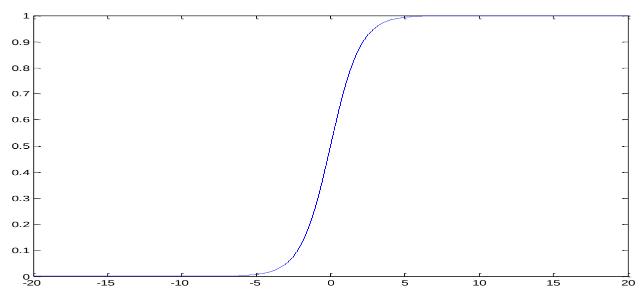
$$f(\mathbf{x},\mathbf{w}) = g_1(\mathbf{w}^T\mathbf{x}) = g(\mathbf{w}^T\mathbf{x})$$



Logistic function

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

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



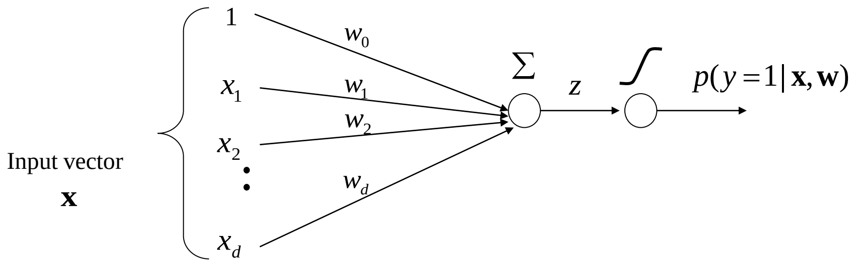
Logistic regression model

- **Discriminant functions:**

$$g_1(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x}) \qquad g_0(\mathbf{x}) = 1 - g(\mathbf{w}^T\mathbf{x})$$

- **Where** $g(z) = 1/(1+e^{-z})$ - is a logistic function
- **Values of discriminant functions vary in [0,1]**
  - **Probabilistic interpretation**

$$f(\mathbf{x},\mathbf{w}) = p(y=1 \mid \mathbf{w},\mathbf{x}) = g_1(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x})$$



Logistic regression

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

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

- we learn **a probabilistic function**

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

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

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

Note that: $p(y=0 \mid \mathbf{x},\mathbf{w}) = 1 - p(y=1 \mid \mathbf{x},\mathbf{w})$

- Transformation to discrete class values:

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

Linear decision boundary

- Logistic regression model defines a **linear decision boundary**
- **Why?**
- **Answer:** Compare two **discriminant functions**.
- **Decision boundary:** $g_1(\mathbf{x}) = g_0(\mathbf{x})$
- For the boundary it must hold:

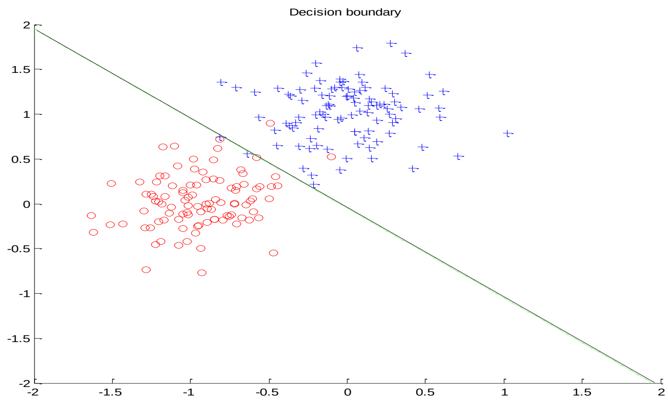
$$\log \frac{g_o(\mathbf{x})}{g_1(\mathbf{x})} = \log \frac{1 - g(\mathbf{w}^{\mathbf{T}}\mathbf{x})}{g(\mathbf{w}^{\mathbf{T}}\mathbf{x})} = 0$$

$$\log \frac{g_o(\mathbf{x})}{g_1(\mathbf{x})} = \log \frac{\dfrac{\exp -(\mathbf{w}^{\mathbf{T}}\mathbf{x})}{1+\exp -(\mathbf{w}^{\mathbf{T}}\mathbf{x})}}{\dfrac{1}{1+\exp -(\mathbf{w}^{\mathbf{T}}\mathbf{x})}} = \log \exp -(\mathbf{w}^{\mathbf{T}}\mathbf{x}) = \mathbf{w}^{\mathbf{T}}\mathbf{x} = 0$$

Logistic regression model. Decision boundary

- **LR defines a linear decision boundary**

Example: 2 classes (blue and red points)



Logistic regression: parameter learning.

Likelihood of outputs

- **Let**

$$D_i = < \mathbf{x}_i, y_i > \qquad \mu_i = p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = g(z_i) = g(\mathbf{w}^T \mathbf{x})$$

- **Then**

$$L(D, \mathbf{w}) = \prod_{i=1}^{n} P(y = y_i \mid \mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{n} \mu_i^{y_i} (1 - \mu_i)^{1 - y_i}$$

- **Find weights w that maximize the likelihood of outputs**
  - Apply the log-likelihood trick The optimal weights are the same for both the likelihood and the log-likelihood

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

$$= \sum_{i=1}^{n} y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)$$

Logistic regression: parameter learning

- **Log likelihood**

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

- **Derivatives of the loglikelihood**

Nonlinear in weights !!

$$-\frac{\partial}{\partial w_j} l(D, \mathbf{w}) = \sum_{i=1}^{n} - x_{i,j} (y_i - g(z_i))$$

$$\nabla_{\mathbf{w}} - l(D, \mathbf{w}) = \sum_{i=1}^{n} - \mathbf{x}_i (y_i - g(\mathbf{w}^T \mathbf{x}_i)) = \sum_{i=1}^{n} - \mathbf{x}_i (y_i - f(\mathbf{w}, \mathbf{x}_i))$$

- **Gradient descent:**

$$\mathbf{w}^{(k)} \leftarrow \mathbf{w}^{(k-1)} - \alpha(k) \nabla_{\mathbf{w}} [-l(D, \mathbf{w})] |_{\mathbf{w}^{(k-1)}}$$

$$\mathbf{w}^{(k)} \leftarrow \mathbf{w}^{(k-1)} + \alpha(k) \sum_{i=1}^{n} [y_i - f(\mathbf{w}^{(k-1)}, \mathbf{x}_i)] \mathbf{x}_i$$

Logistic regression. Online gradient descent

- **On-line component of the loglikelihood**

$$-J_{\text{online}}(D_i, \mathbf{w}) = y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)$$

- **On-line learning update for weight w** $J_{online}(D_k, \mathbf{w})$

$$\mathbf{w}^{(k)} \leftarrow \mathbf{w}^{(k-1)} - \alpha(k) \nabla_{\mathbf{w}} [J_{online}(D_k, \mathbf{w})] |_{\mathbf{w}^{(k-1)}}$$

- **ith update for the logistic regression** and $D_k = <\mathbf{x}_k, y_k>$

$$\mathbf{w}^{(i)} \leftarrow \mathbf{w}^{(k-1)} + \alpha(k)[y_i - f(\mathbf{w}^{(k-1)}, \mathbf{x}_k)]\mathbf{x}_k$$

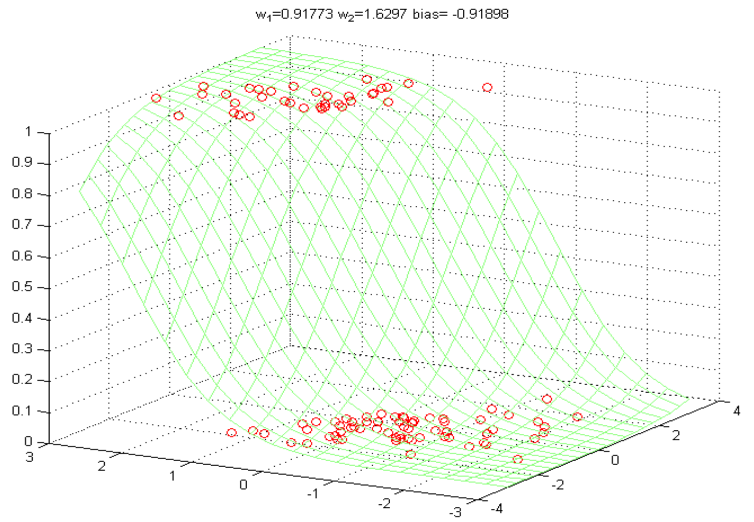
Online logistic regression algorithm

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

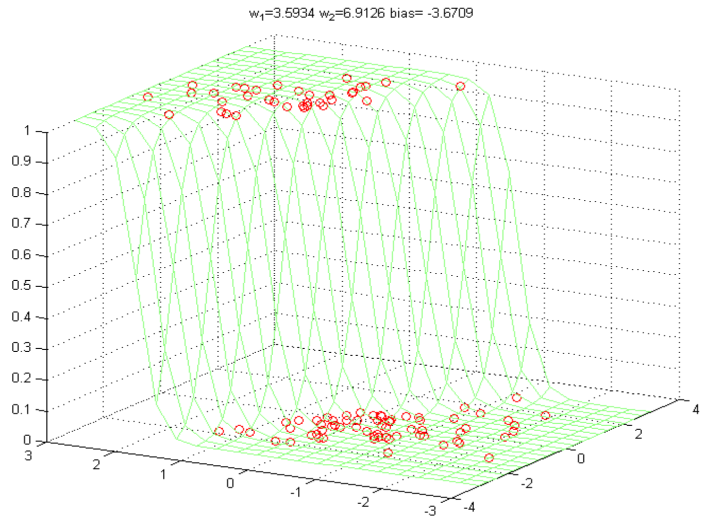
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(i)[y_i - f(\mathbf{w}, \mathbf{x}_i)]\mathbf{x}_i$$

  end for
  return weights $\mathbf{w}$

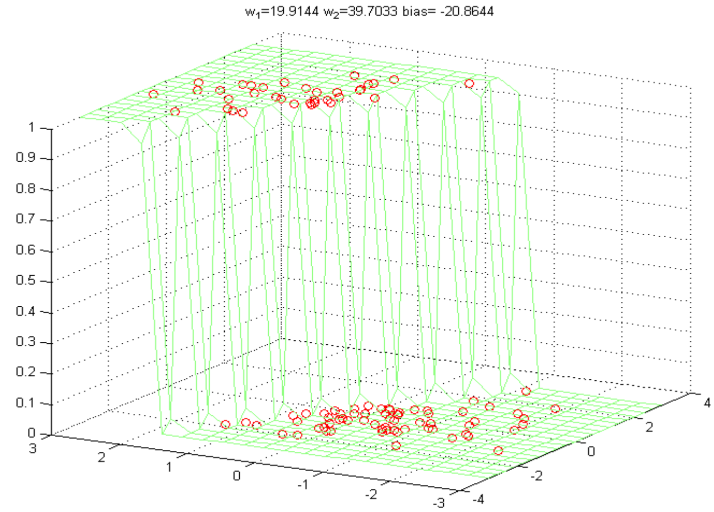
Online algorithm. Example.



Online algorithm. Example.



Online algorithm. Example.



Derivation of the gradient

- **Log likelihood** $l(D,\mathbf{w}) = \sum_{i=1}^{n} y_i \log \mu_i + (1-y_i)\log(1-\mu_i)$
- **Derivatives of the loglikelihood**

$$\frac{\partial}{\partial w_j} l(D,\mathbf{w}) = \sum_{i=1}^{n} \frac{\partial}{\partial z_i}\left[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)\right]\frac{\partial z_i}{\partial w_j}$$

$$\frac{\partial z_i}{\partial w_j} = x_{i,j}$$

Derivative of a logistic function

$$\frac{\partial g(z_i)}{\partial z_i} = g(z_i)(1-g(z_i))$$

$$\frac{\partial}{\partial z_i}\left[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)\right] = y_i \frac{1}{g(z_i)}\frac{\partial g(z_i)}{\partial z_i} + (1-y_i)\frac{-1}{1-g(z_i)}\frac{\partial g(z_i)}{\partial z_i}$$

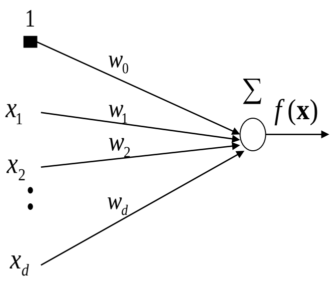
$$= y_i(1-g(z_i)) + (1-y_i)(-g(z_i)) \boxed{= y_i - g(z_i)}$$

$$\boxed{\nabla_{\mathbf{w}} l(D,\mathbf{w}) = \sum_{i=1}^{n} -\mathbf{x}_i(y_i - g(\mathbf{w}^T\mathbf{x}_i)) = \sum_{i=1}^{n} -\mathbf{x}_i(y_i - f(\mathbf{w},\mathbf{x}_i))}$$

Limitations of basic linear units

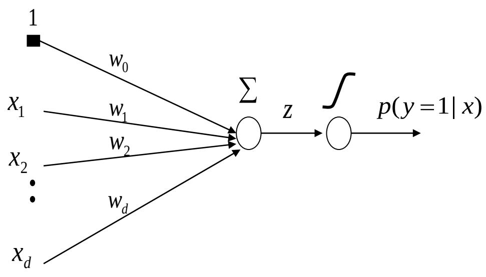
Linear regression

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$



Logistic regression

$$f(\mathbf{x}) = p(y=1 \mid \mathbf{x}, \mathbf{w}) = g(\mathbf{w}^T \mathbf{x})$$



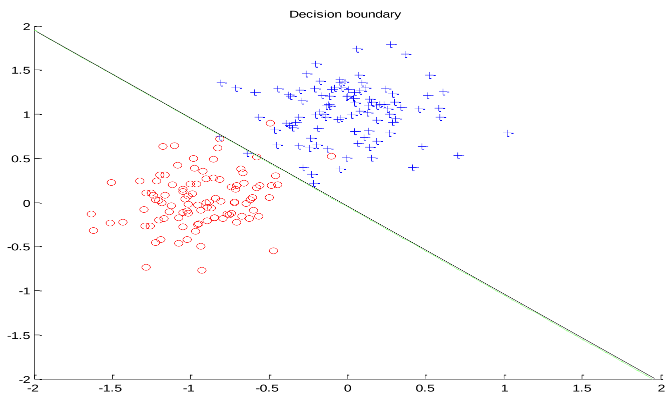
Function linear in inputs !!

Linear decision boundary!!

Logistic regression. Decision boundary

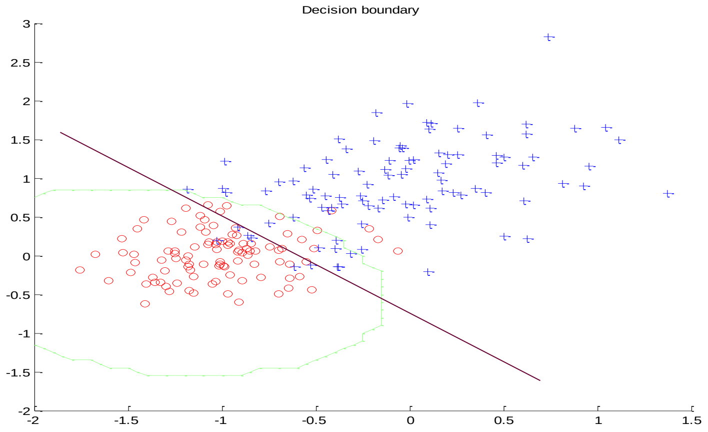
Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)



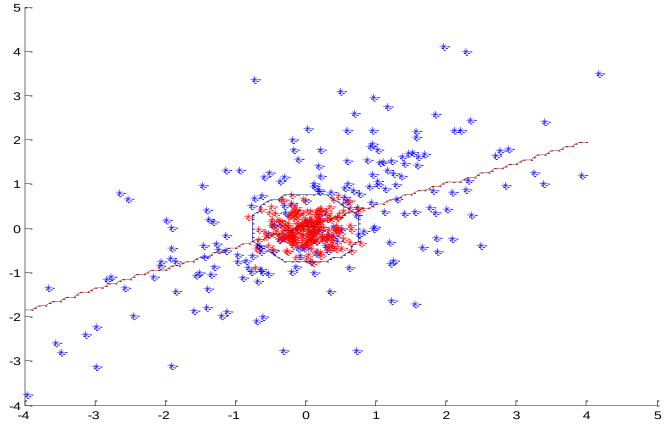
Linear decision boundary

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



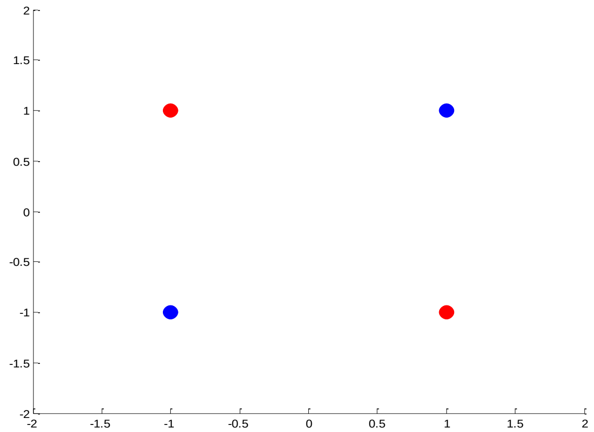
When logistic regression fails?

- Example in which the logistic regression model fails



Limitations of logistic regression.

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



Extensions of simple linear units

- use **feature (basis) functions** to model **nonlinearities**

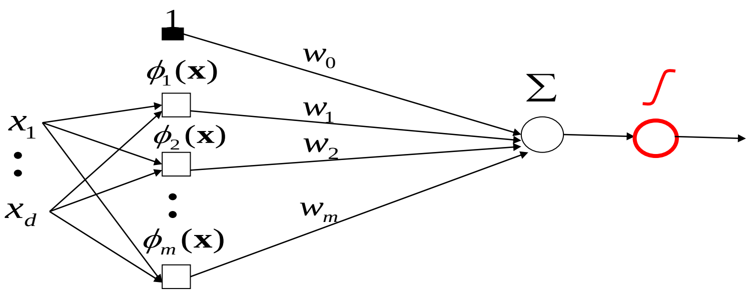
Linear regression

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

Logistic regression

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

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



The same trick can be done also 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:**

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

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

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

$$\vdots$$

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