CS 2710 Foundations of AI
Lecture 22

Logistic regression

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

Announcements

Homework 10:

- **due on Thursday, December 2, 2004**

Final exam: December 14, 2004 at 1:00pm-3:00pm

- Location: Sennott Square 5129
- Closed book
- Cumulative

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 $f : X \to Y$

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

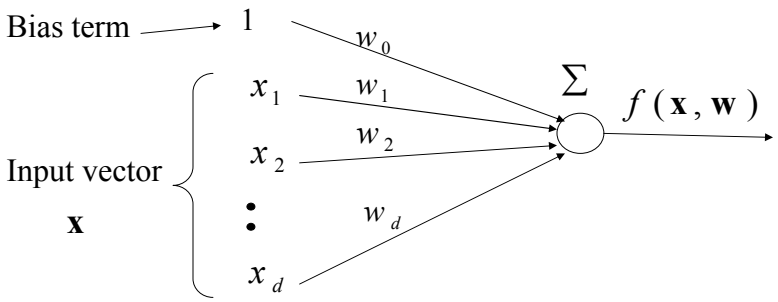
- **Regression:** Y is **continuous**
  Example: earnings, product orders → company stock price
- **Classification:** Y is **discrete**
  Example: handwritten digit  → digit label

Linear regression

- **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)**



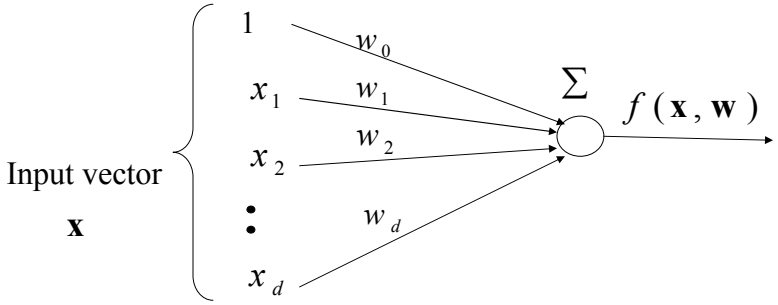
Linear regression

- **Shorter (vector) definition of the model**
  - Include bias constant in the input vector

$$\mathbf{x} = (1, x_1, x_2, \cdots x_d)$$

$$f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = \mathbf{w}^T \mathbf{x}$$

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



Linear regression. Error.

- **Data:** $D_i =< \mathbf{x}_i, y_i >$
- **Function:** $\mathbf{x}_i \rightarrow f(\mathbf{x}_i)$
- 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

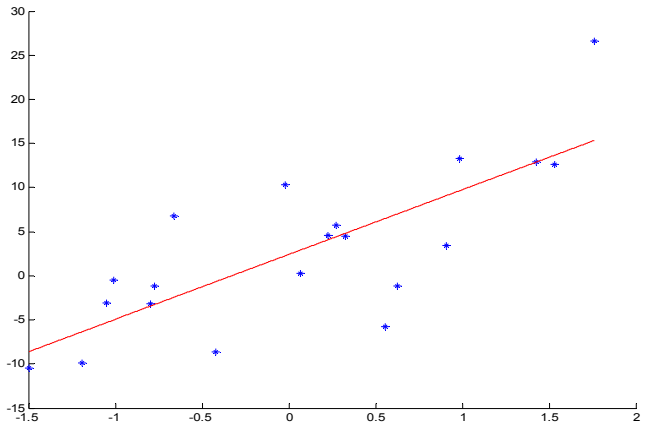
Mean-squared error $J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$

- **Learning:**

  We want to find the weights minimizing the error !

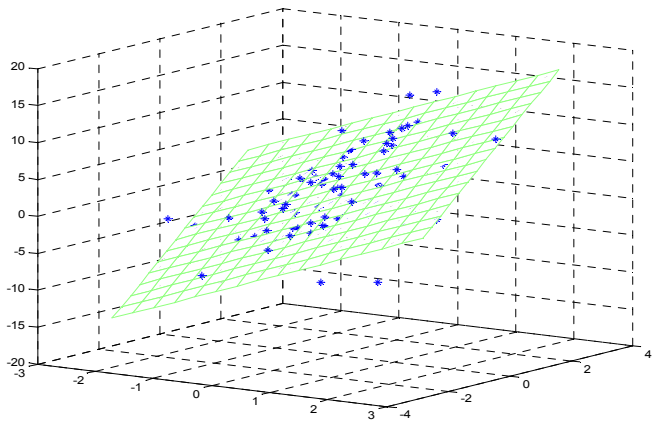
Linear regression. Example

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



Linear regression. Example.

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



Linear regression. Optimization.

- We want the **weights minimizing the error**

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

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

$$\frac{\partial}{\partial w_j}J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d})x_{i,j} = 0$$

- **Vector of derivatives:**

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

Solving linear regression

$$\frac{\partial}{\partial w_j}J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d})x_{i,j} = 0$$

By rearranging the terms we get a **system of linear equations** with *d+1* unknowns

$$\mathbf{Aw} = \mathbf{b}$$

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

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

$$\bullet\bullet\bullet$$

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

$$\bullet\bullet\bullet$$

Solving linear regression

- The optimal set of weights satisfies:

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

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

$$\mathbf{Aw} = \mathbf{b}$$

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

Solution to SLE:

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

- matrix inversion

Gradient descent solution

Goal: the weight optimization in the linear regression model

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

An alternative to SLE solution:

- **Gradient descent**

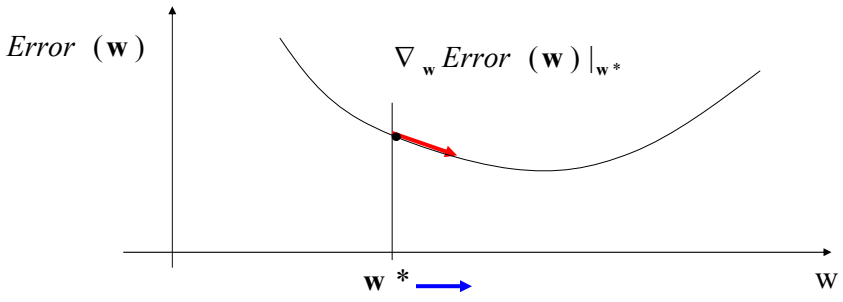
  Idea:
  - Adjust weights in the direction that improves the Error
  - The gradient tells us what is the right direction

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

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

Gradient descent method

- Descend using the gradient information

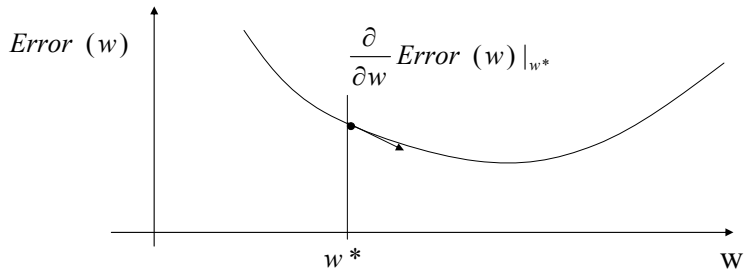


Direction of the descent

- 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



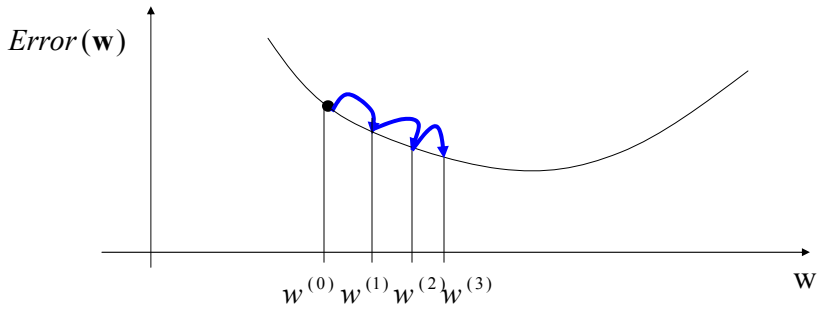
- New value of the parameter

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

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

Gradient descent method

- Iteratively converge to the optimum of the Error function



Online gradient algorithm

- The error function is defined for the whole dataset D

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

- **error for a sample** $D_i = < \mathbf{x}_i, y_i >$

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

- **Online gradient method: changes weights after every sample**
- **vector form:**

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(\mathbf{w})$$

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

$\alpha > 0$ - Learning rate that depends on the number of updates

Online gradient method

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: generates a sequence of online updates

(i)-th update step with : $D_i =< \mathbf{x}_i, y_i >$

j-th weight:

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial Error_i(\mathbf{w})}{\partial w_j} |_{\mathbf{w}^{(i-1)}}$$

$$w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))x_{i,j}$$

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

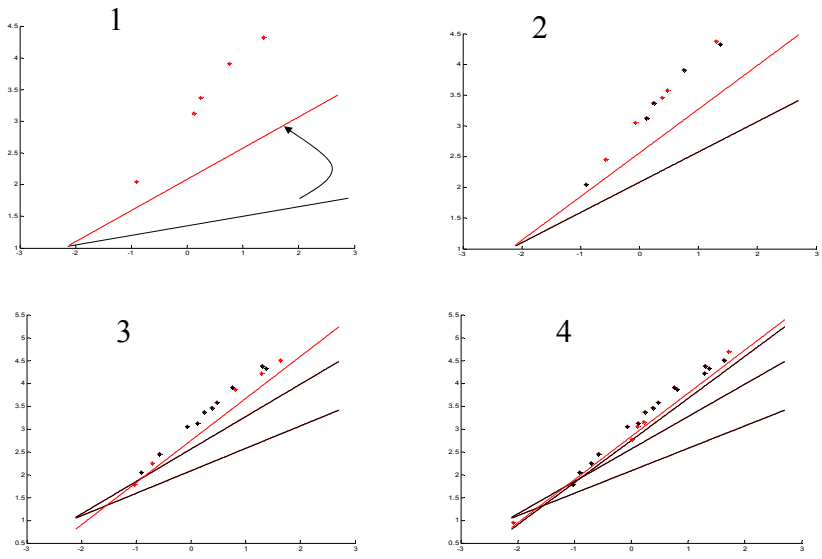
- Gradually rescales changes in weights

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 **w**

• **Advantages:** very easy to implement, continuous data streams, adapts to changes in the model over time

On-line learning. Example



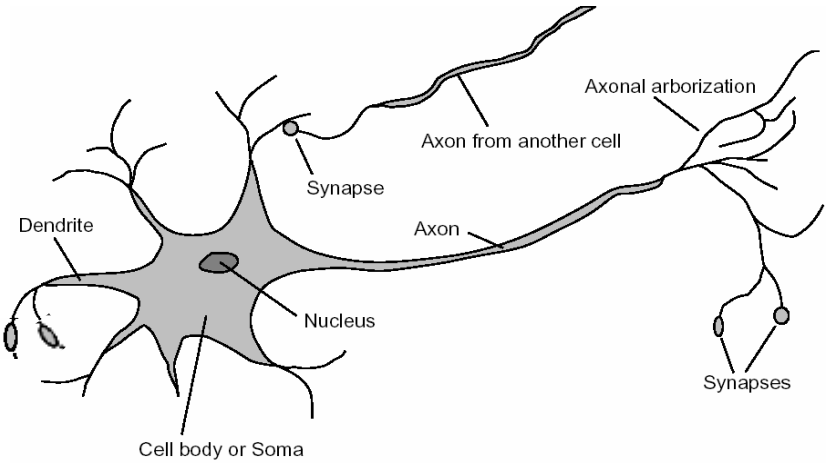
Logistic regression

Binary classification

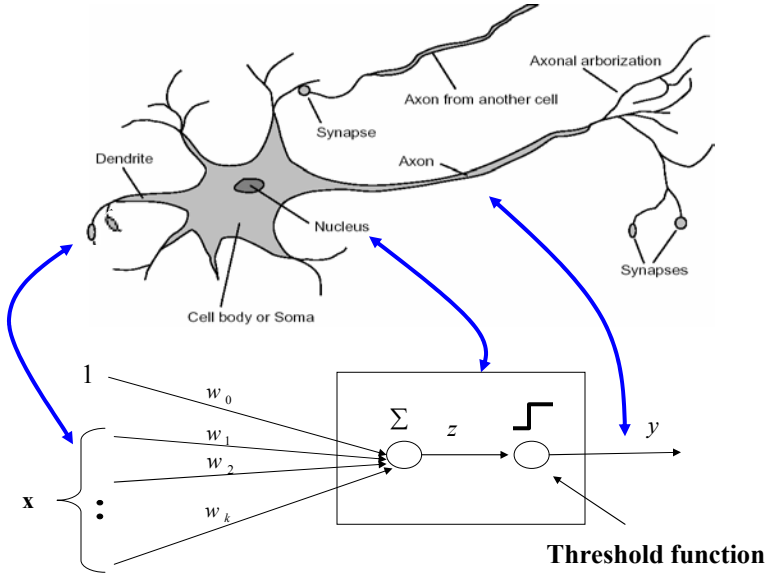
- **Two classes** $Y = \{0,1\}$
- Our goal is to learn how to correctly classify two types of examples
  - Class 0 – labeled as 0,
  - Class 1 – labeled as 1
- We would like to learn $f : X \rightarrow \{0,1\}$
- **First step:** we need to devise a model of the function f
- **Inspiration:** ***neuron (nerve cells)***

Neuron

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

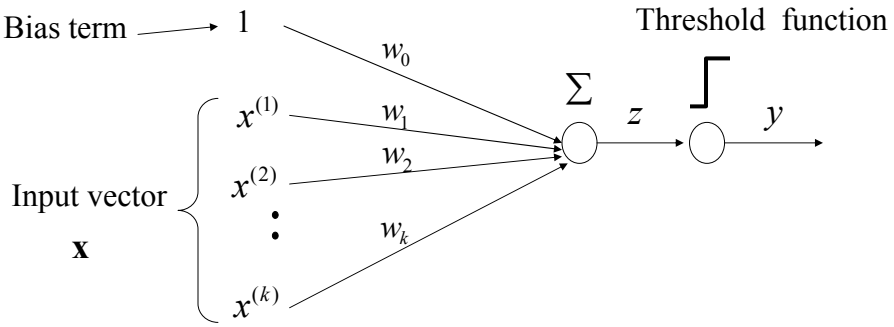


Neuron-based binary classification model



Neuron-based binary classification

- **Function we want to learn** $f : X \rightarrow \{0,1\}$



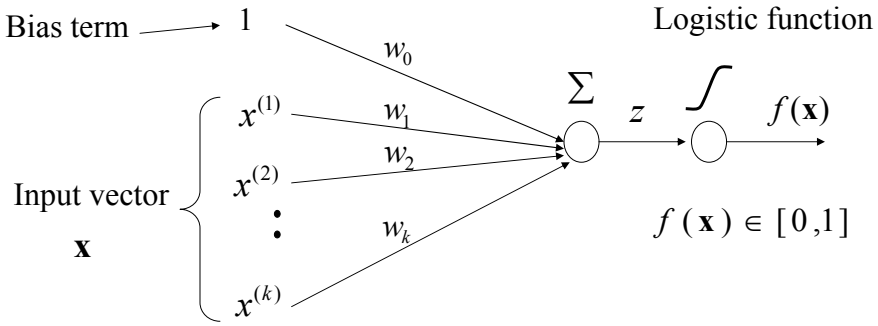
Logistic regression model

- **A function model with smooth switching:**

$$f(\mathbf{x}) = g(w_0 + w_1 x^{(1)} + \ldots w_k x^{(k)})$$

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

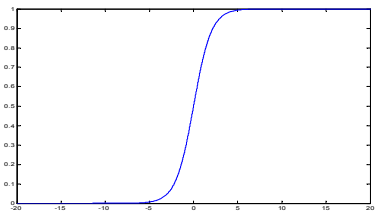
and *g(z)* is a **logistic function** $g(z) = 1/(1+e^{-z})$



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]
- Output of the logistic regression has a probabilistic interpretation

$$f(\mathbf{x}) = p(y = 1 \mid \mathbf{x})$$ - Probability of class 1

Logistic regression. Decision boundary

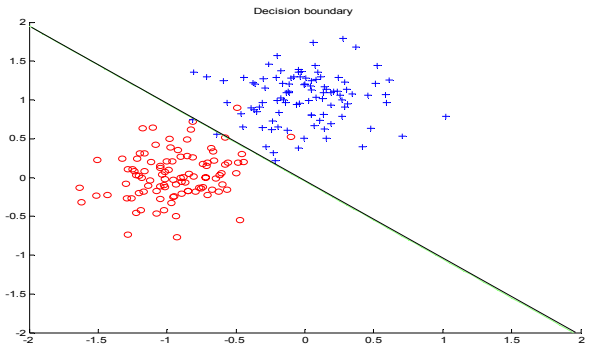
Classification with the logistic regression model:

If $f(\mathbf{x}) \geq 1/2$ then choose class **1**

Else choose class **0**

Logistic regression model defines a linear decision boundary

Example:



Optimization of weights

- **Two classes:** $Y = \{0,1\}$
- **Data:** $D = \{d_1, d_2, .., d_n\}$

  $d_i = < \mathbf{x}_i, y_i >$
- We want to find the set of weight **w** that explain the data the best
- **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:

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

$y = 0$ otherwise

Logistic regression. Parameter optimization.

- We construct a probabilistic version of the **error function** based on the **likelihood of the data**

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

Independent samples (x_i ,y_i)

- likelihood measures the goodness of fit (opposite of the error)

- Log-likelihood trick.

$$l(D, \mathbf{w}) = \log L(D \mid \mathbf{w})$$

- Error is the opposite of the Log likelihood

$$Error\ (D, \mathbf{w}) = -l(D, \mathbf{w})$$

Logistic regression: parameter learning

- **Error function decomposes to online error components**

$$Error\ (D, \mathbf{w}) = \sum_{i=1}^{n} Error_i(D_i, \mathbf{w}) = -\sum_{i=1}^{n} l_i(D_i, \mathbf{w})$$

- **Derivatives of the online error component for the LR model (in terms of weights)**

$$\frac{\partial}{\partial w_0} Error_i(D_i, \mathbf{w}) = -(y_i - f(\mathbf{x}_i, \mathbf{w}))$$

$$\vdots$$

$$\frac{\partial}{\partial w_j} Error_i(D_i, \mathbf{w}) = -(y_i - f(\mathbf{x}_i, \mathbf{w}))x_{i,j}$$

Logistic regression: parameter learning.

- **Assume** $D_i =< \mathbf{x}_i, y_i >$
- **Let**

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

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

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

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

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**

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

Nonlinear in weights !!

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

- **Gradient descent:**

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

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

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

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

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

$$= y_i(1-g(z_i)) + (1-y_i)(-g(z_i)) = y_i - g(z_i)$$

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

Logistic regression. Online gradient.

- We want to optimize the online Error
- **On-line gradient update for the jth weight and ith step**

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha \frac{\partial}{\partial w_j}[Error_i(D_i,\mathbf{w})|_{w^{(i-1)}}]$$

- **(i)th update for the logistic regression** $D_i =< \mathbf{x}_i, y_i >$

J-th weight

$$w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)\left(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)})\right)x_{i,j}$$

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

The same update rule as used in the linear regression !!!

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)

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

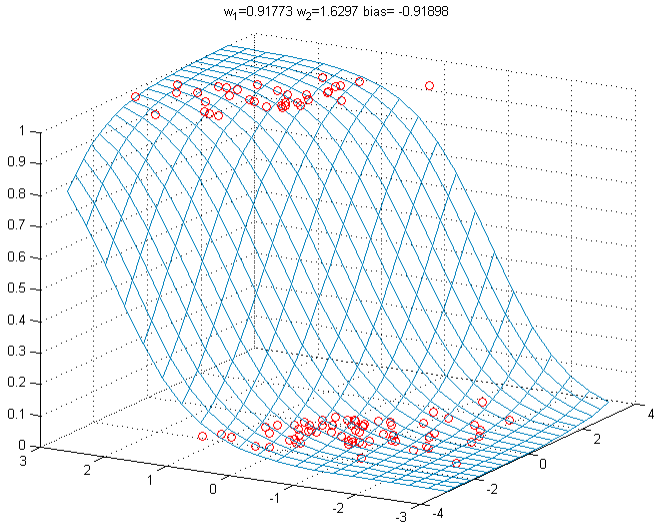
$$\vdots$$

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

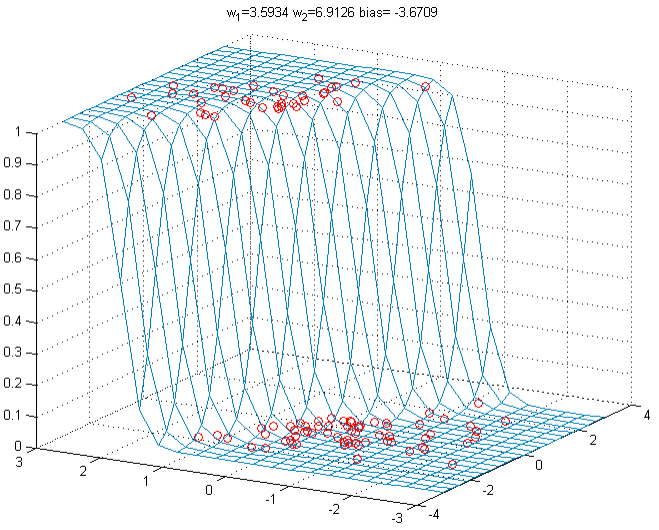
end for

return weights

Online algorithm. Example.



Online algorithm. Example.



Online algorithm. Example.