

# CS 1571 Introduction to AI

## Machine Learning: Linear and logistic regression, decision trees

Milos Hauskrecht

[milos@cs.pitt.edu](mailto:milos@cs.pitt.edu)

5329 Sennott Square

# Supervised learning

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

$$D_i = \langle \mathbf{x}_i, y_i \rangle$$

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,d})$  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  $i = 1, \dots, n$

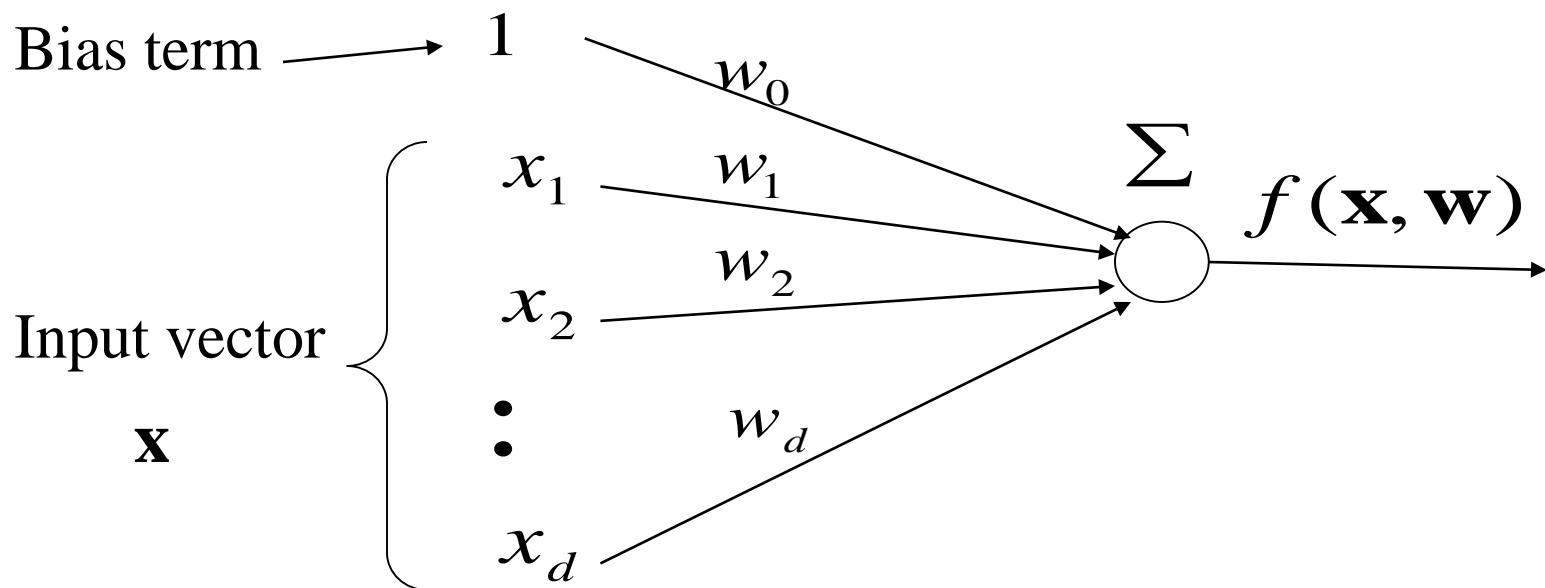
- **Regression:**
  - **Y is in  $\mathcal{R}$**
- **Classification**
  - **Y is discrete**

# Linear regression

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

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

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



# Linear regression

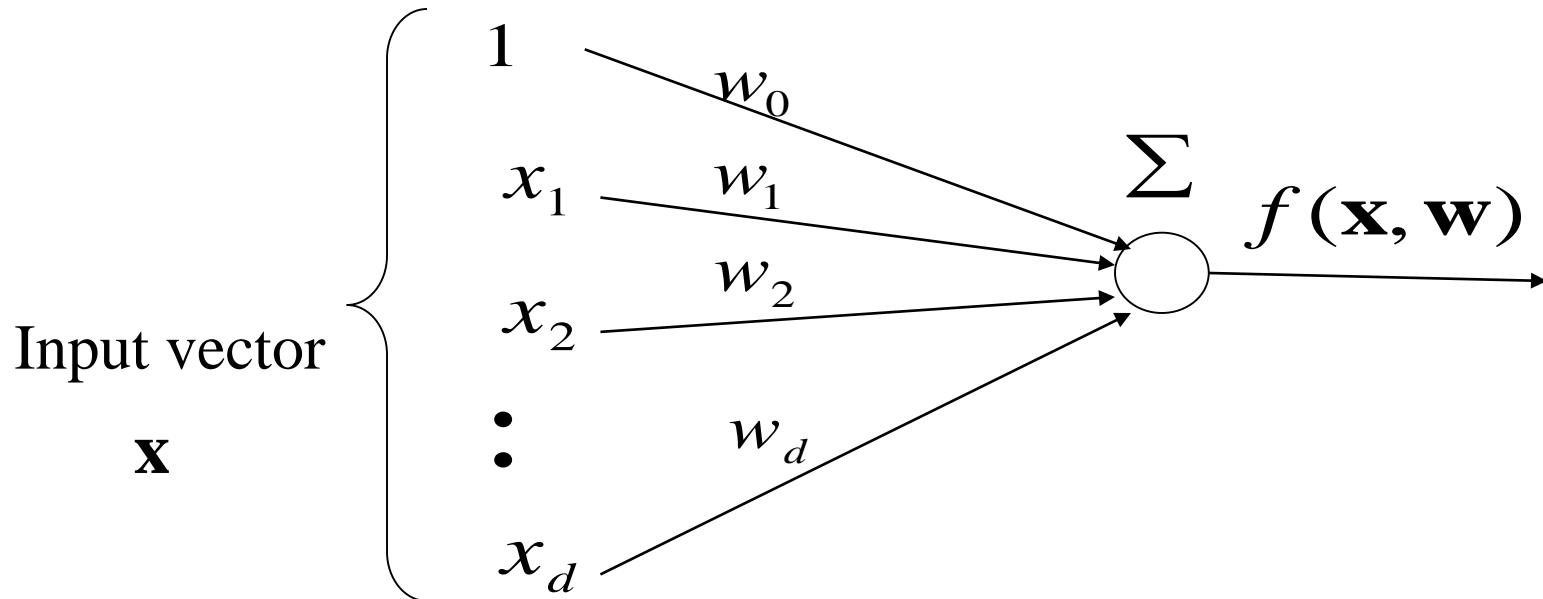
- **Shorter (vector) definition of the model**

- Include bias constant in the input vector

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

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

$w_0, w_1, \dots, w_k$  - parameters (weights)



# Linear regression. Error.

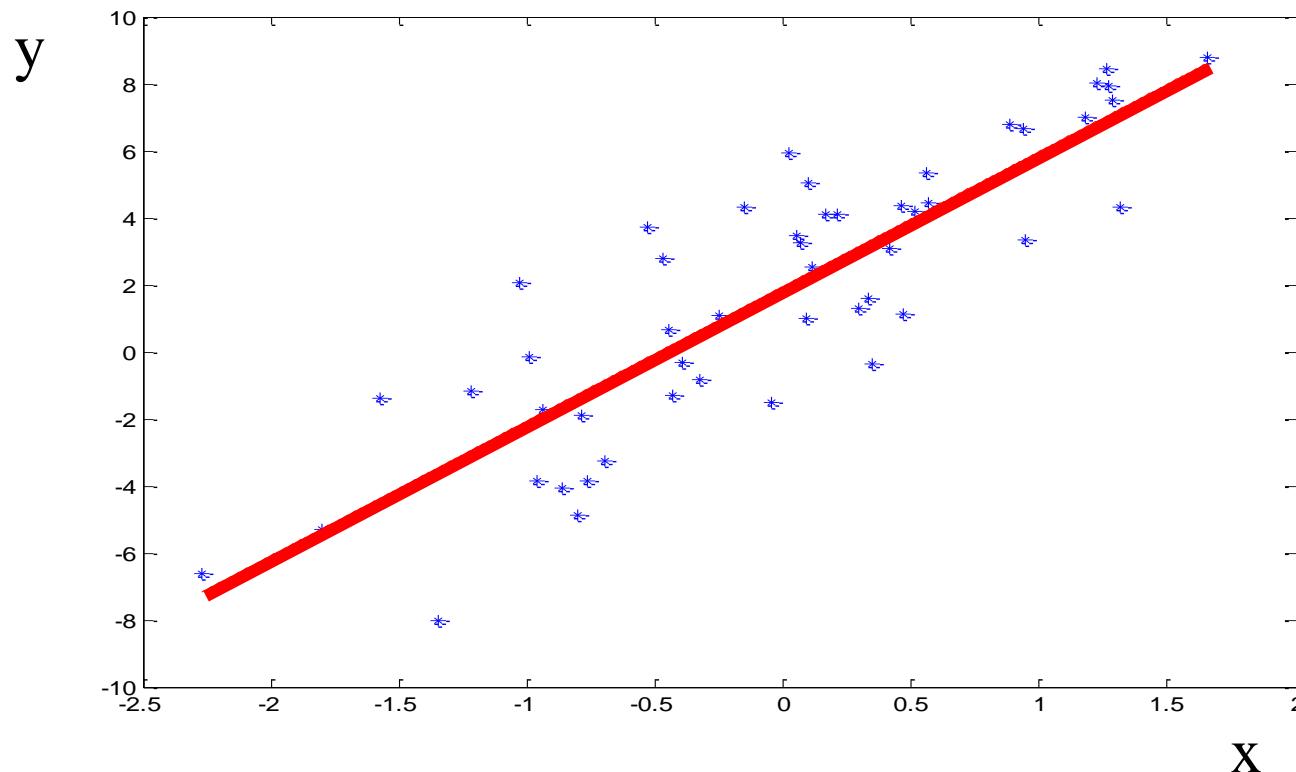
- **Data:**  $D_i = \langle \mathbf{x}_i, y_i \rangle$
- **Function:**  $\mathbf{x}_i \rightarrow f(\mathbf{x}_i)$
- We would like to have  $y_i \approx f(\mathbf{x}_i)$  for all  $i = 1, \dots, n$
- **Error function** measures how much our predictions deviate from the desired answers

**Average square 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 !**

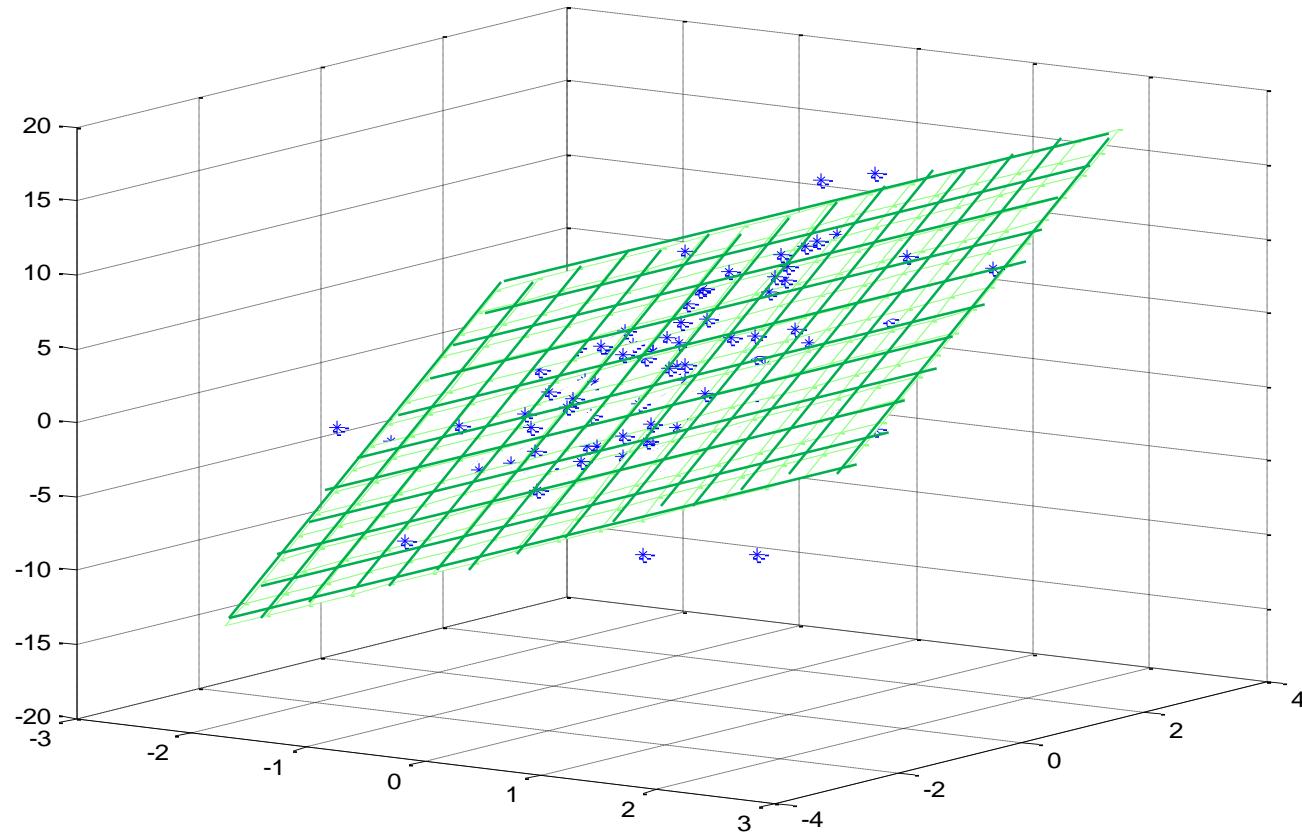
# Linear regression. Example

- 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**

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

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

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

- **Vector of derivatives:**

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

# Linear regression. Optimization.

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

$$J_n = \frac{1}{n} \sum_{i=1..n} (y_i - f(\mathbf{x}_i))^2 = \frac{1}{n} \sum_{i=1..n} (y_i - [w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)}])^2$$

- $\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \bar{\mathbf{0}}$  defines a set of equations in  $\mathbf{w}$

$$\frac{\partial}{\partial w_0} J_n(w) = -\frac{2}{n} \sum_{i=1}^n [y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)})] = 0$$

$$\frac{\partial}{\partial w_1} J_n(w) = -\frac{2}{n} \sum_{i=1}^n [y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)})] x^{(1)} = 0$$

...

$$\frac{\partial}{\partial w_j} J_n(w) = -\frac{2}{n} \sum_{i=1}^n [y_i - (w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots w_k x^{(k)})] x^{(j)} = 0$$

...

# Solving linear regression

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

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

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

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

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

• • •

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

• • •

# Solving linear regression

- The optimal set of weights satisfies:

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

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

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

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

## Solutions to SLE:

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

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

# Gradient descent solution

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

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

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

**Idea:**

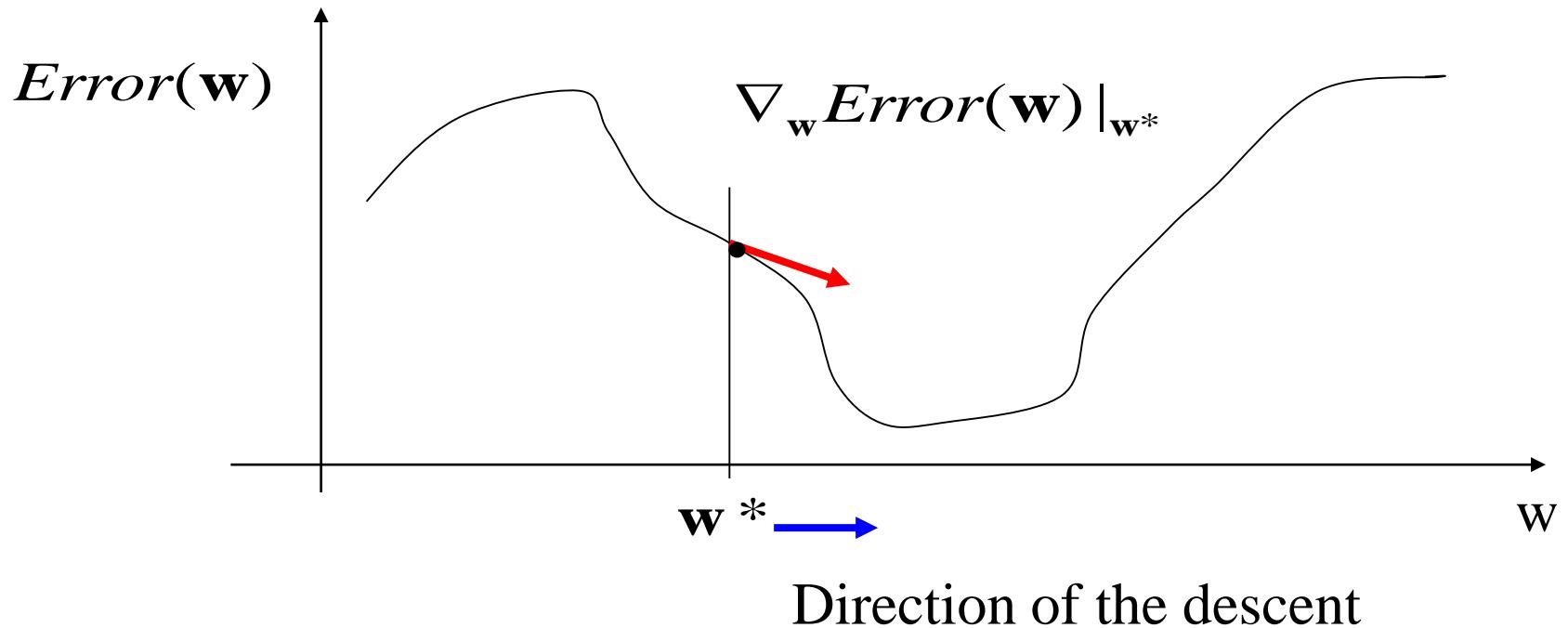
- Adjust weights in the direction that improves the Error
- The gradient tells us what is the right direction

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \text{Error}_i(\mathbf{w})$$

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

# Gradient descent method

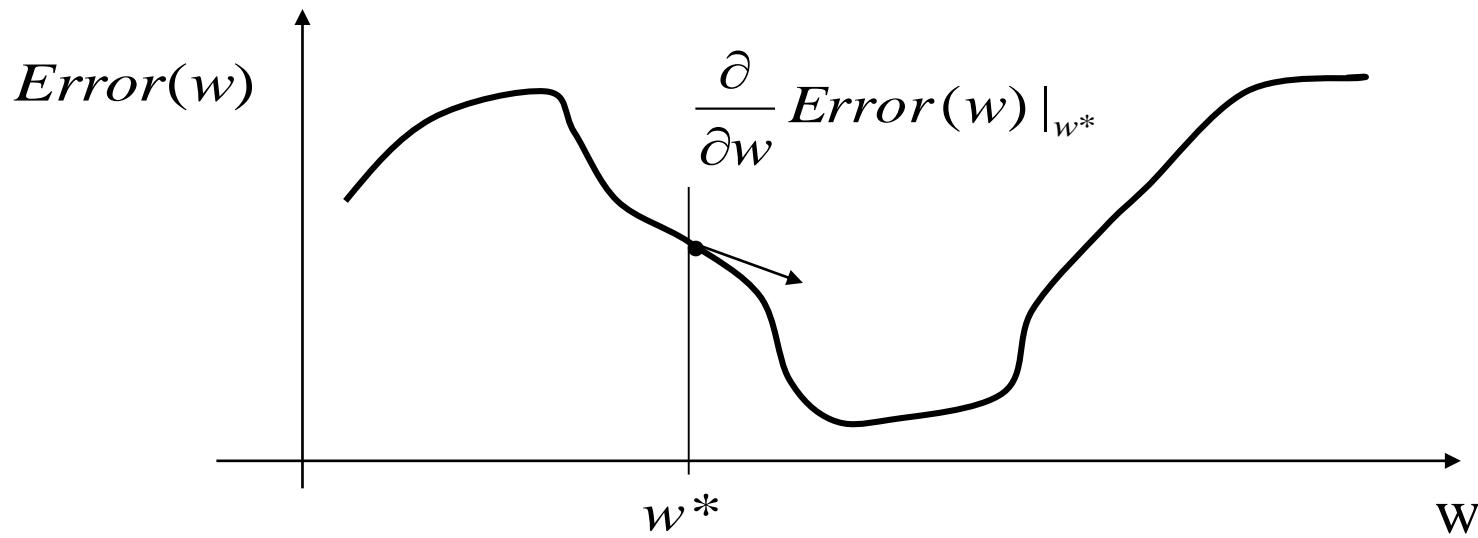
- Descend using the gradient information



- Change the value of  $\mathbf{w}$  according to the gradient

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

# Gradient descent method



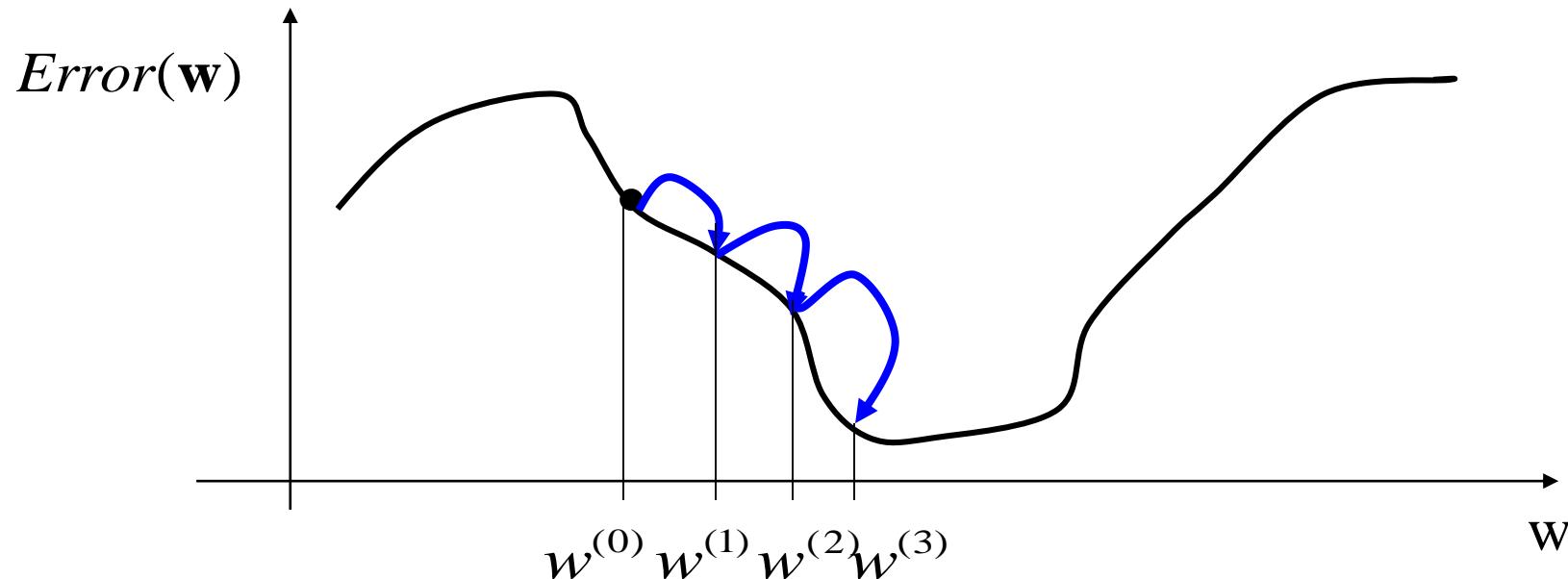
- New value of the parameter

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

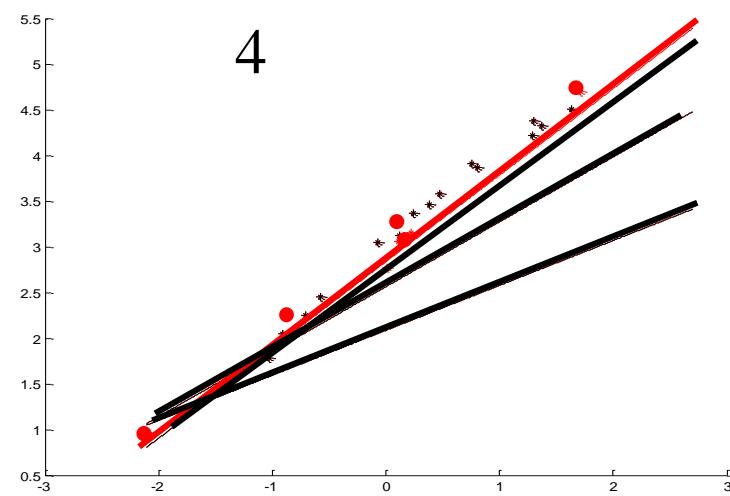
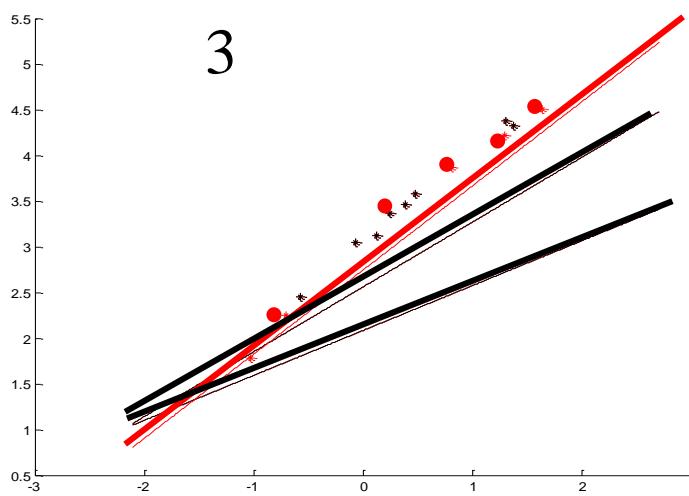
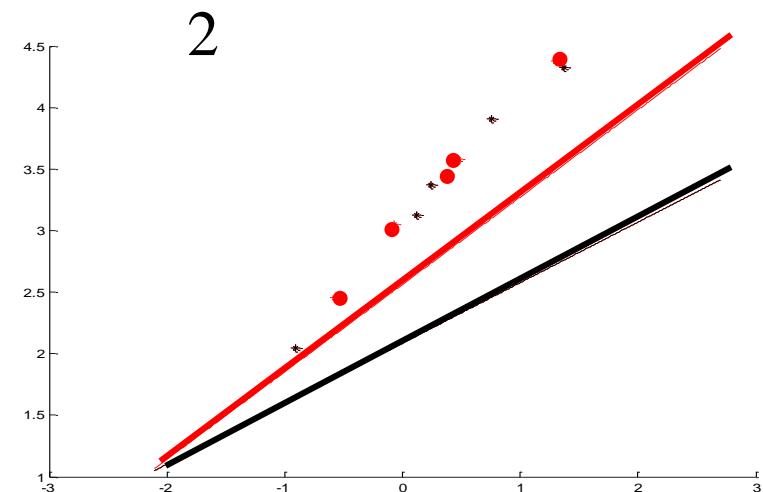
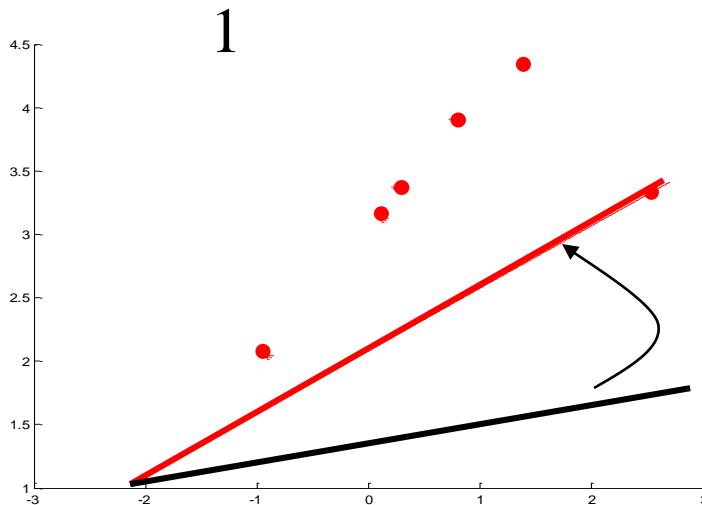
$\alpha > 0$  - a learning rate (scales the gradient changes)

# Gradient descent method

- Iteratively converge to the optimum of the Error function



# On-line learning. Example

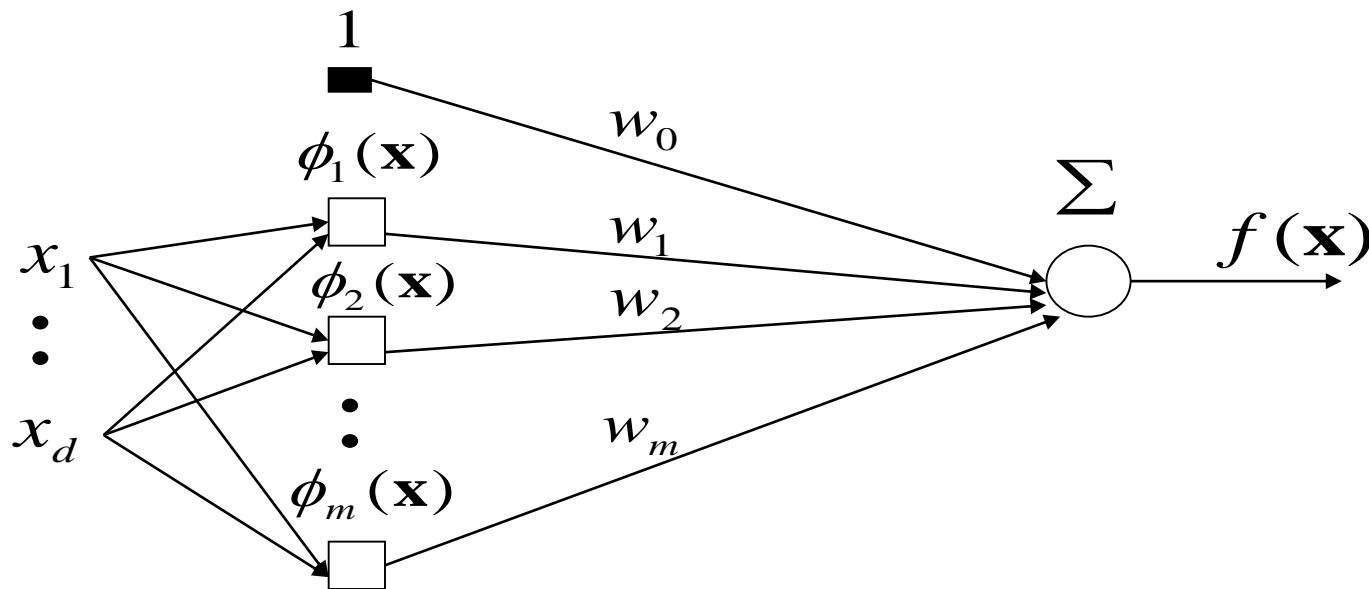


# Extensions of simple linear model

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  $\mathbf{x}$



**The same techniques as before to learn the weights**

# Extensions of the linear model

- Models linear in the parameters we want to fit

$$f(\mathbf{x}) = w_0 + \sum_{k=1}^m w_k \phi_k(\mathbf{x})$$

$w_0, w_1 \dots w_m$  - parameters

$\phi_1(\mathbf{x}), \phi_2(\mathbf{x}) \dots \phi_m(\mathbf{x})$  - **feature or basis functions**

- Basis functions examples:

– a higher order polynomial, one-dimensional input  $\mathbf{x} = (x_1)$

$$\phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3$$

– Multidimensional quadratic  $\mathbf{x} = (x_1, x_2)$

$$\phi_1(\mathbf{x}) = x_1 \quad \phi_2(\mathbf{x}) = x_1^2 \quad \phi_3(\mathbf{x}) = x_2 \quad \phi_4(\mathbf{x}) = x_2^2 \quad \phi_5(\mathbf{x}) = x_1 x_2$$

– Other types of basis functions

$$\phi_1(x) = \sin x \quad \phi_2(x) = \cos x$$

# Example. Regression with polynomials.

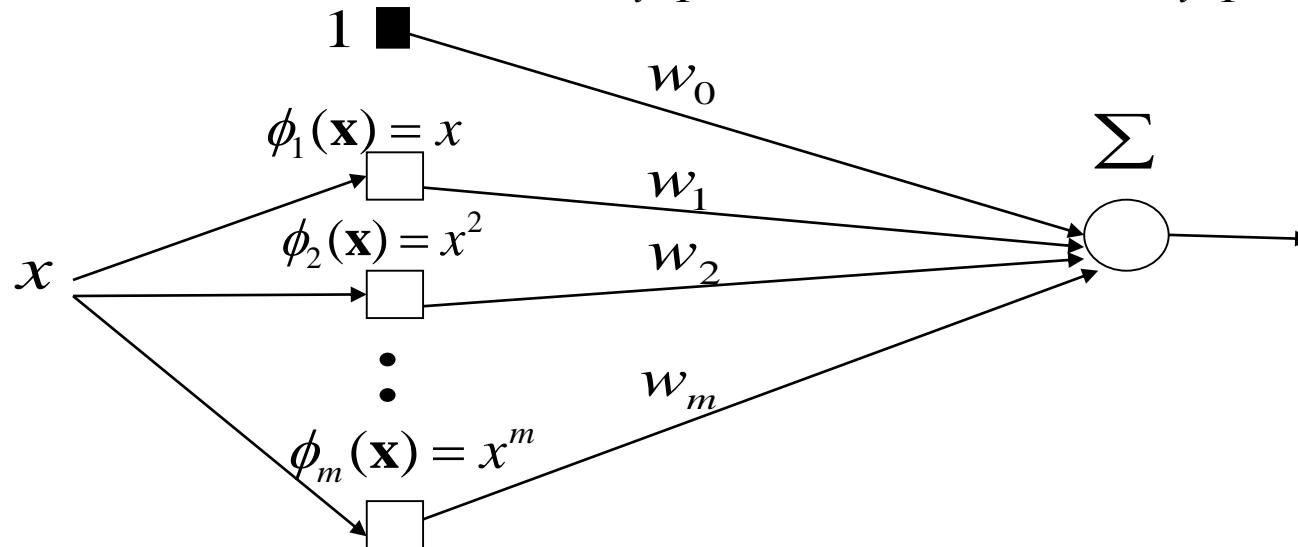
Regression with polynomials of degree m

- **Data points:** pairs of  $\langle x, y \rangle$
- **Feature functions:** m feature functions

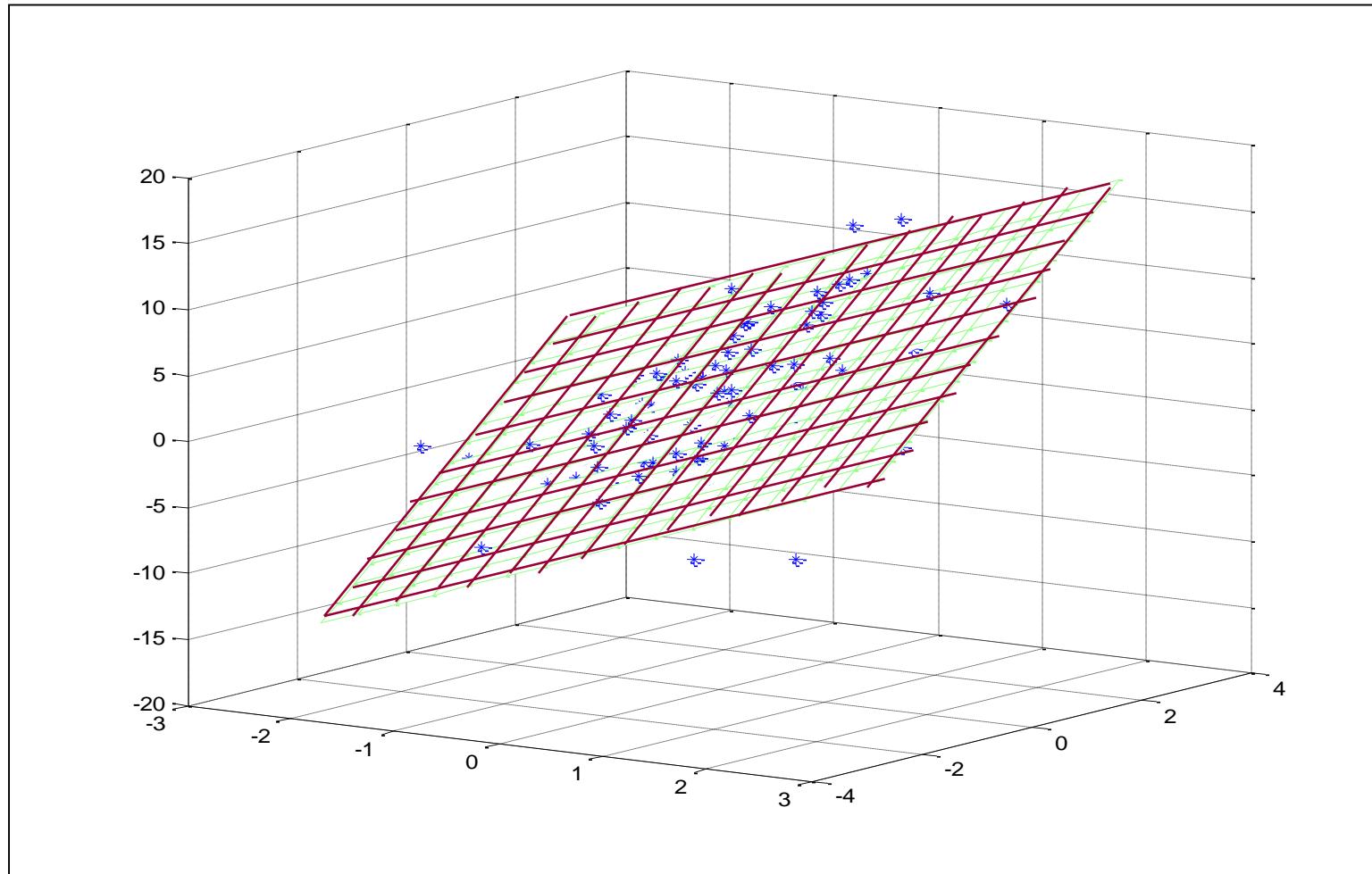
$$\phi_i(x) = x^i \quad i = 1, 2, \dots, m$$

- **Function to learn:**

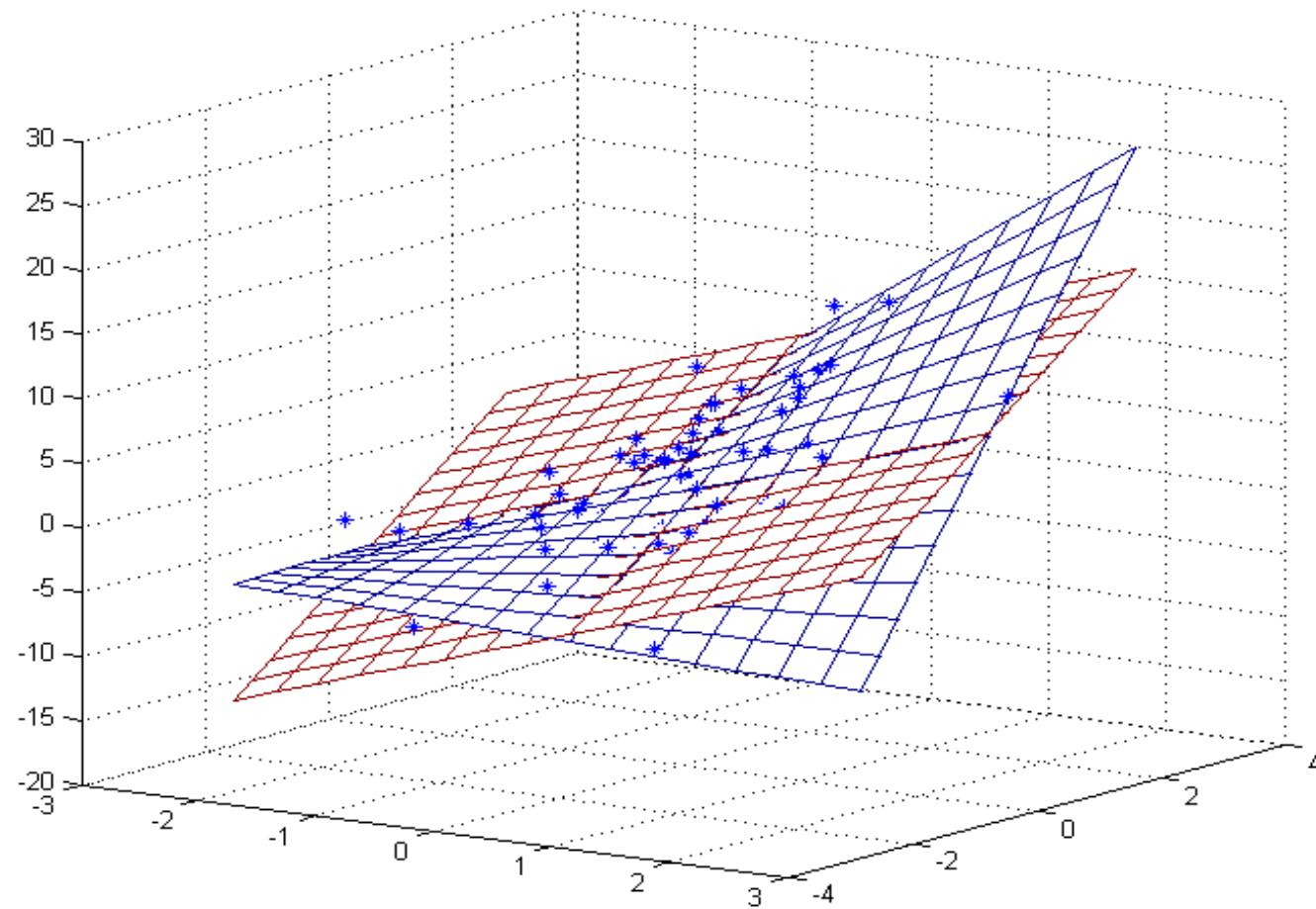
$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^m w_i \phi_i(x) = w_0 + \sum_{i=1}^m w_i x^i$$



# Multidimensional additive model example



# Multidimensional additive model example



# 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**

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

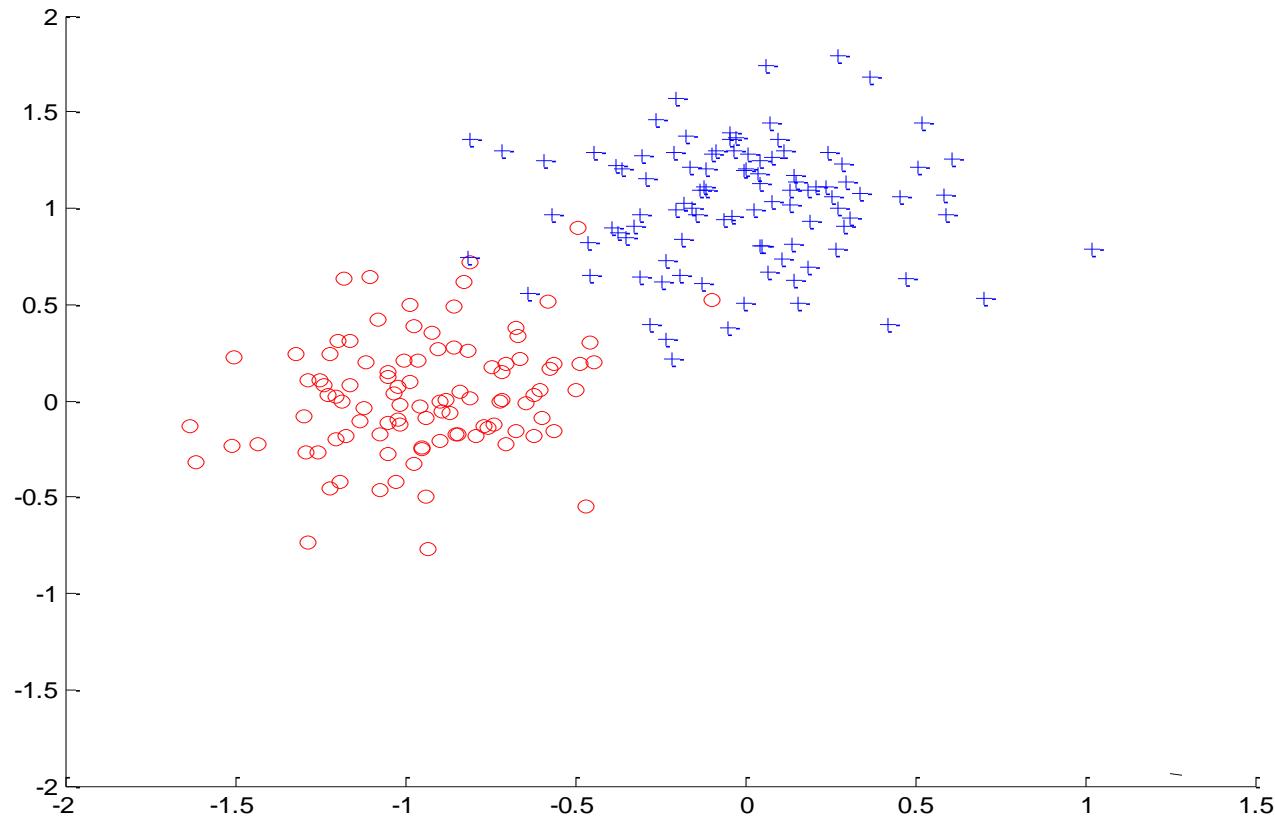
- Error we would like to minimize:  $E_{(x,y)}(Error_1(\mathbf{x}, y))$
- **First step:** we need to devise a model of the function

# Discriminant functions

- One way to represent a **classifier** is by using
  - **Discriminant functions**
- **Works for binary and multi-way classification**
- **Idea:**
  - For every class  $i = 0, 1, \dots, k$  define a function  $g_i(\mathbf{x})$  mapping  $X \rightarrow \Re$
  - When the decision on input  $\mathbf{x}$  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.

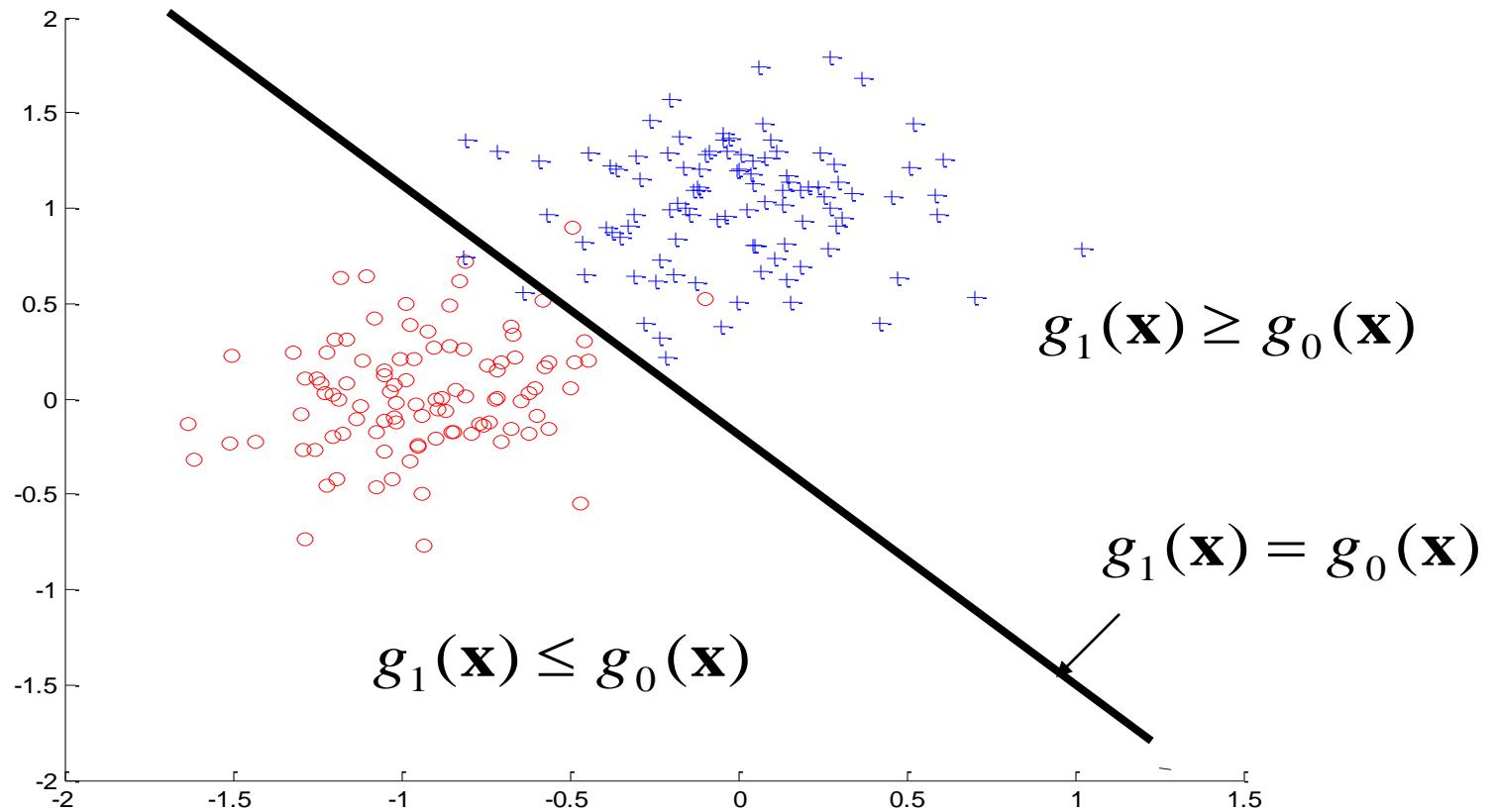
# Discriminant functions

- Example: Two classes in 2-D

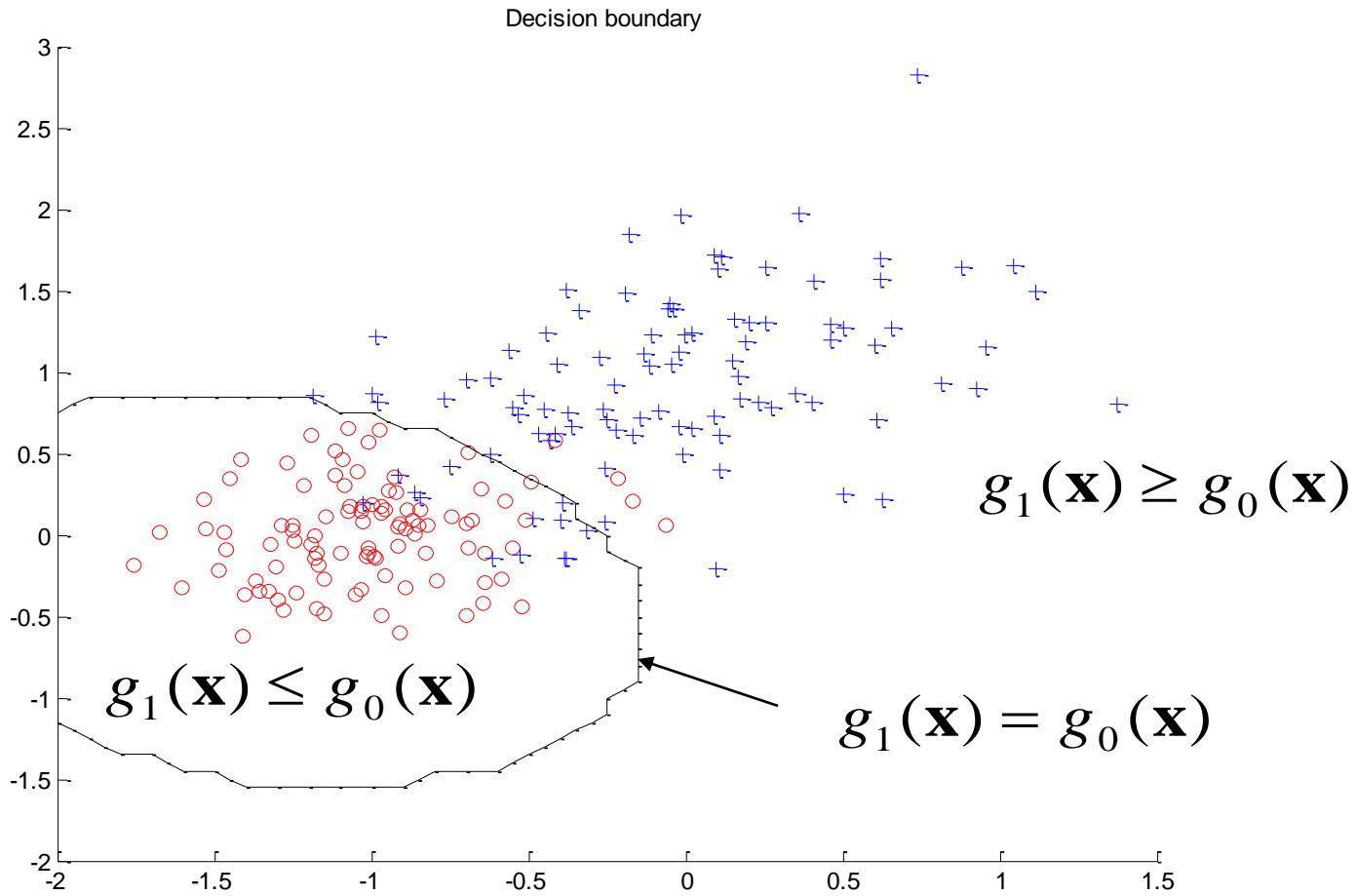


# Discriminant functions

- Discriminant functions  $g_0(\mathbf{x})$  and  $g_1(\mathbf{x})$  define the **decision boundary**



# Quadratic decision boundary



# Logistic regression model

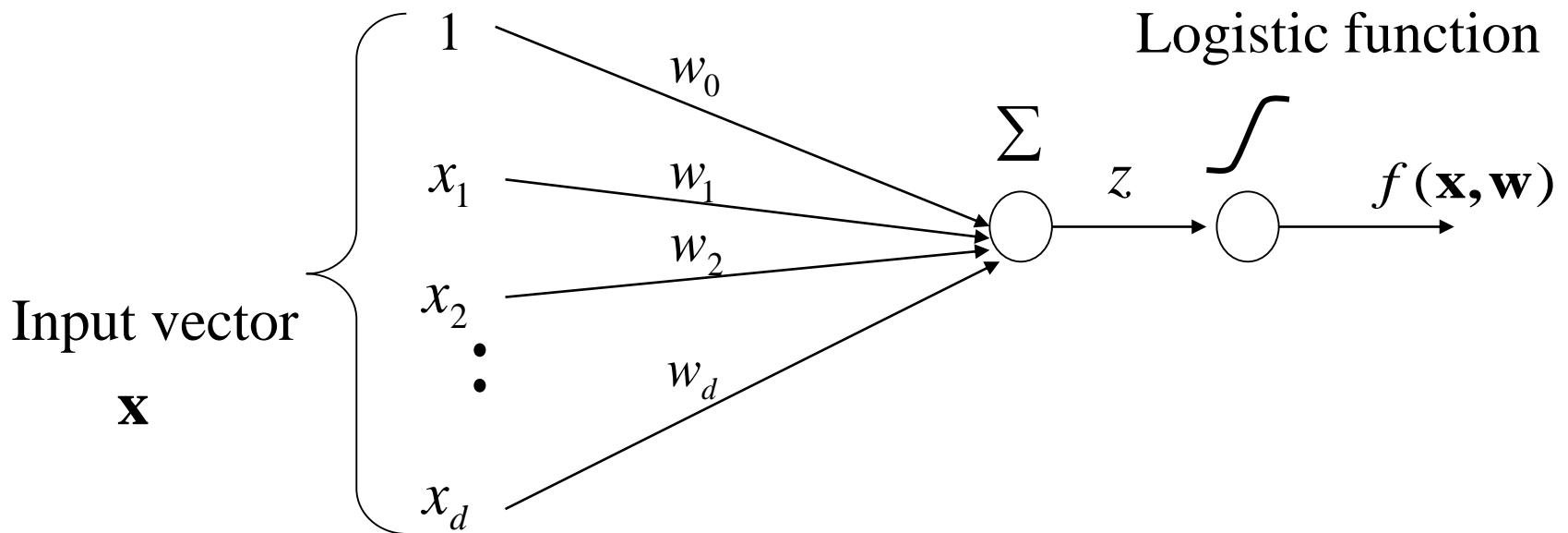
- Defines a linear decision boundary

- Discriminant functions:

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

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

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

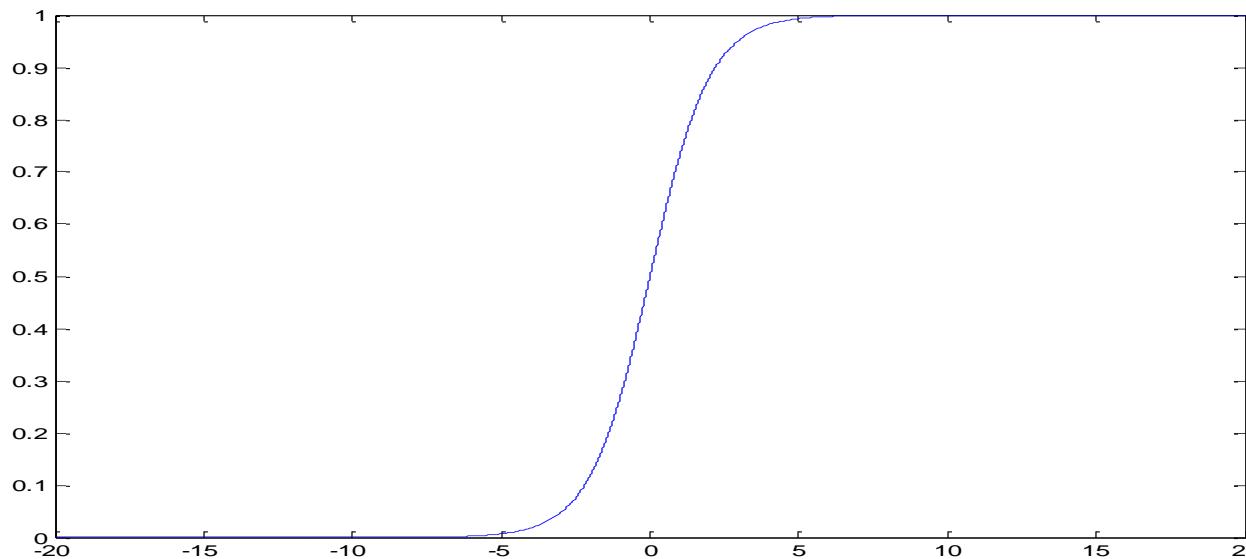


# Logistic function

function

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

- Is 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]



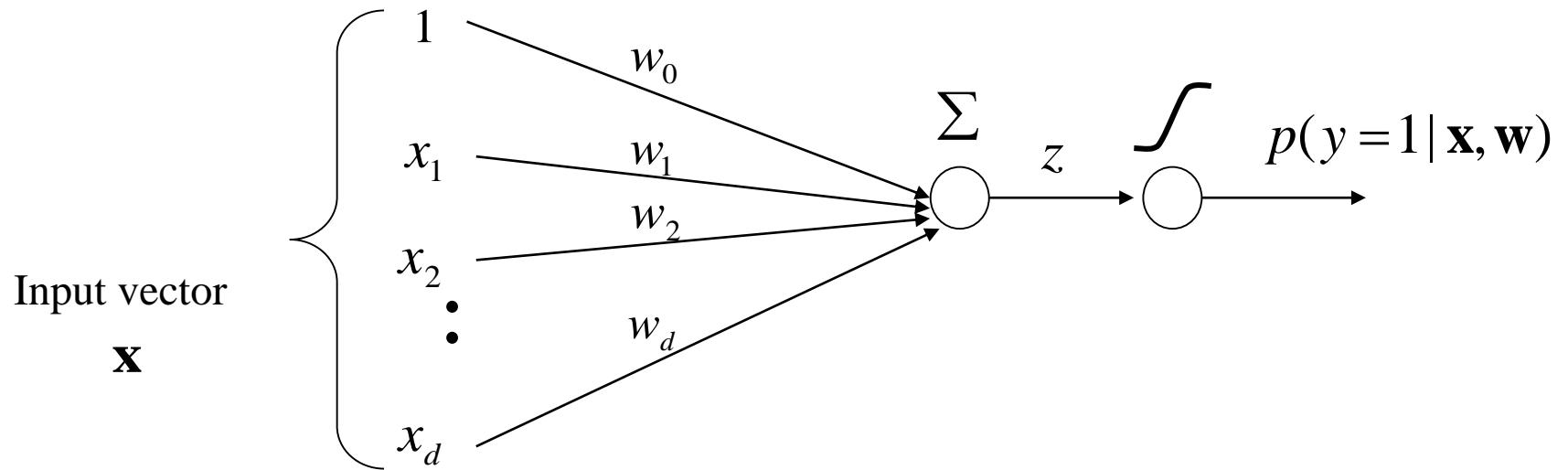
# Logistic regression model

- **Discriminant functions:**

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

- Values of discriminant functions vary in [0,1]
  - **Probabilistic interpretation**

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



# Logistic regression

- We learn **a probabilistic function**

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

- where  $f$  describes the probability of class 1 given  $\mathbf{x}$

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

**Note that:**

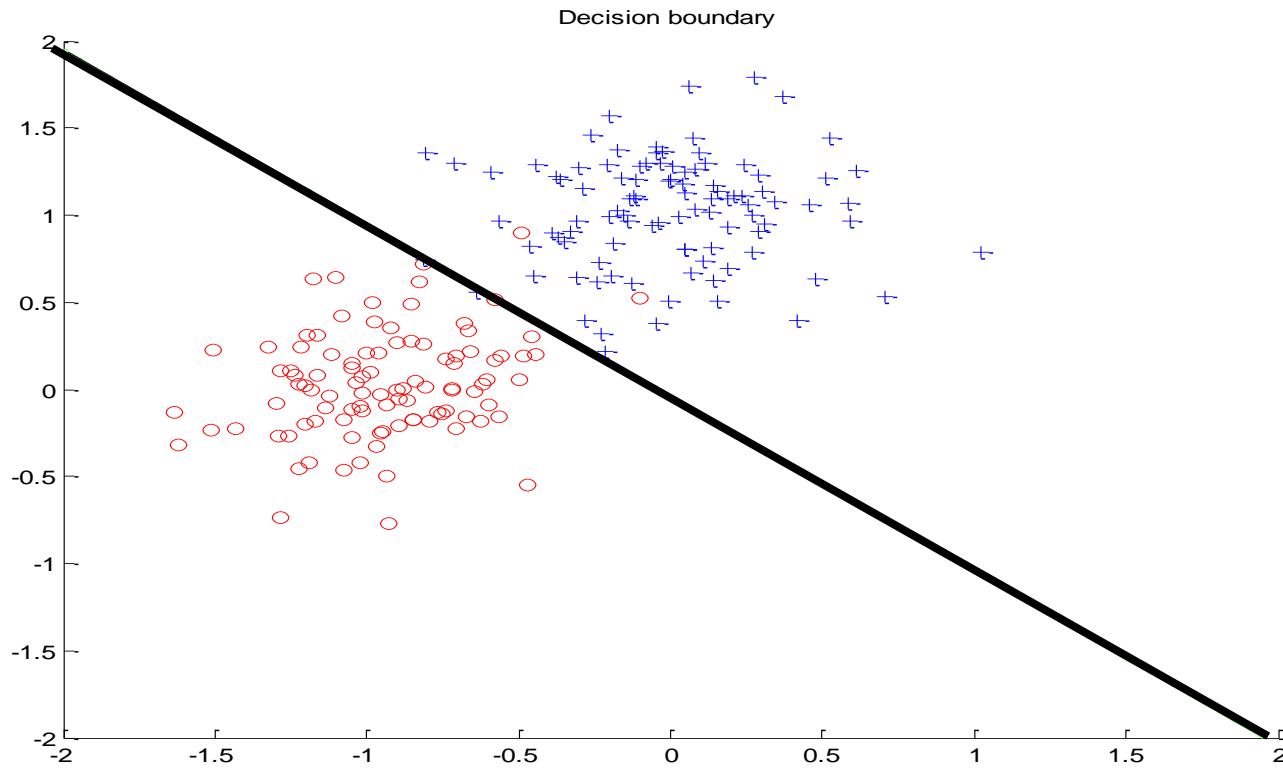
$$p(y=0 | \mathbf{x}, \mathbf{w}) = 1 - p(y=1 | \mathbf{x}, \mathbf{w})$$

- Transformation to binary class values:

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

# Logistic regression model. Decision boundary

- Logistic Regression defines a linear decision boundary  
Example: 2 classes (blue and red points)



# Logistic regression: parameter learning

## Likelihood of outputs

- Let

$$D_i = \langle \mathbf{x}_i, y_i \rangle \quad \mu_i = p(y_i = 1 | \mathbf{x}_i, \mathbf{w}) = g(z_i) = g(\mathbf{w}^T \mathbf{x})$$

- Then

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

- Find weights  $\mathbf{w}$  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

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

# 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 !!

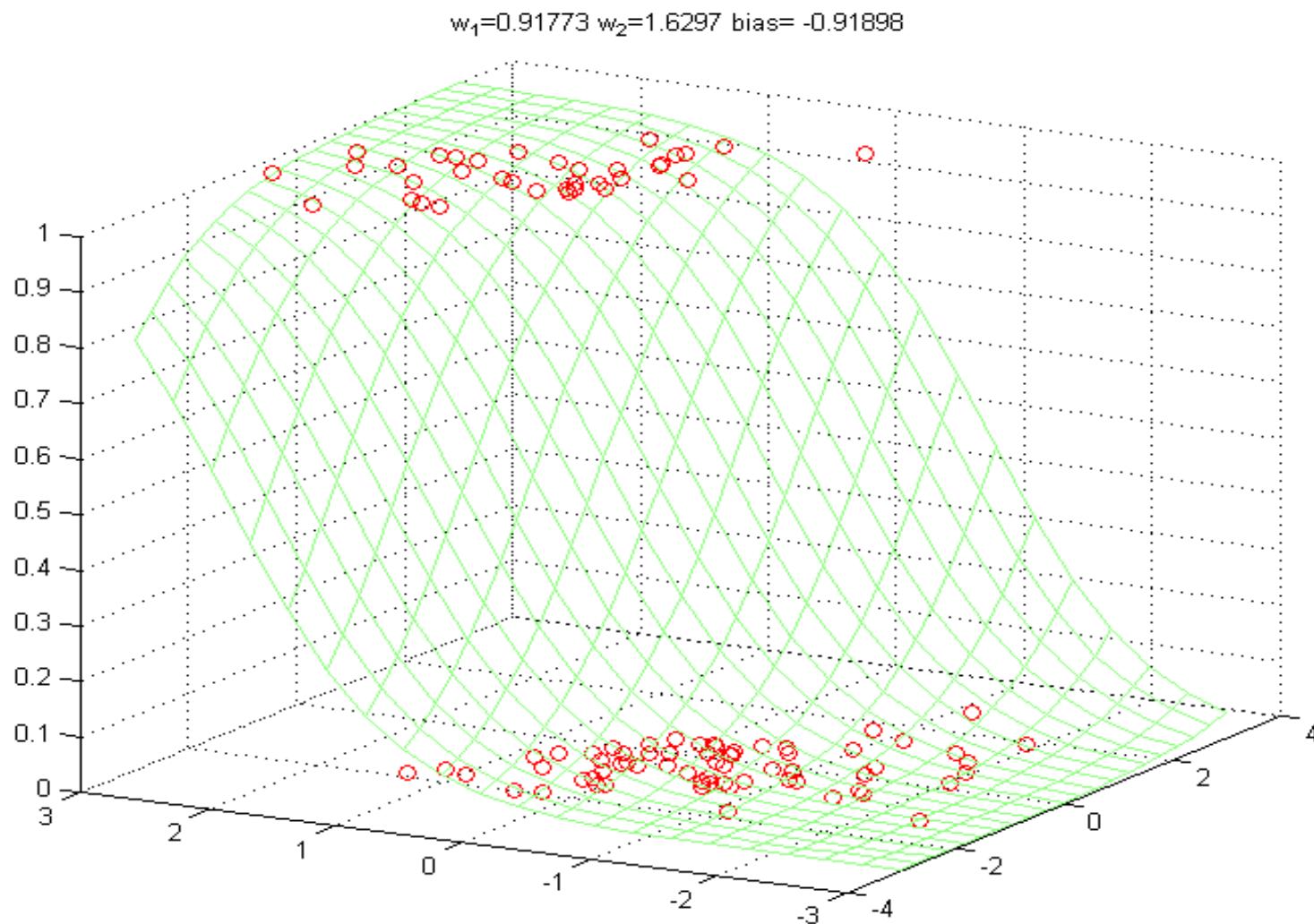
$$\nabla_{\mathbf{w}} -l(D, \mathbf{w}) = \sum_{i=1}^n -\mathbf{x}_i (y_i - g(\mathbf{w}^T \mathbf{x}_i)) = \sum_{i=1}^n -\mathbf{x}_i (y_i - f(\mathbf{w}, \mathbf{x}_i))$$

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

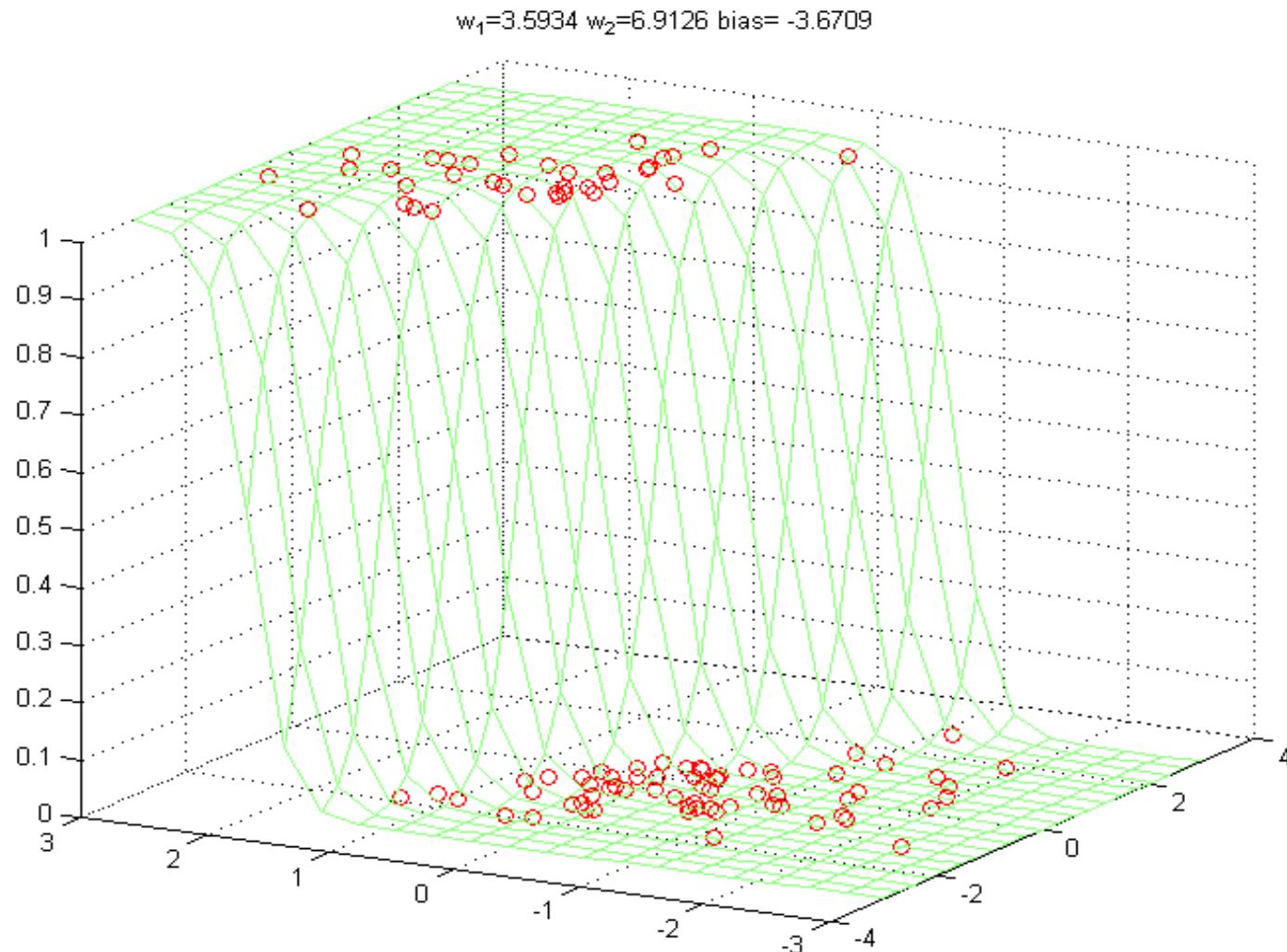
## k-th update of the weights

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

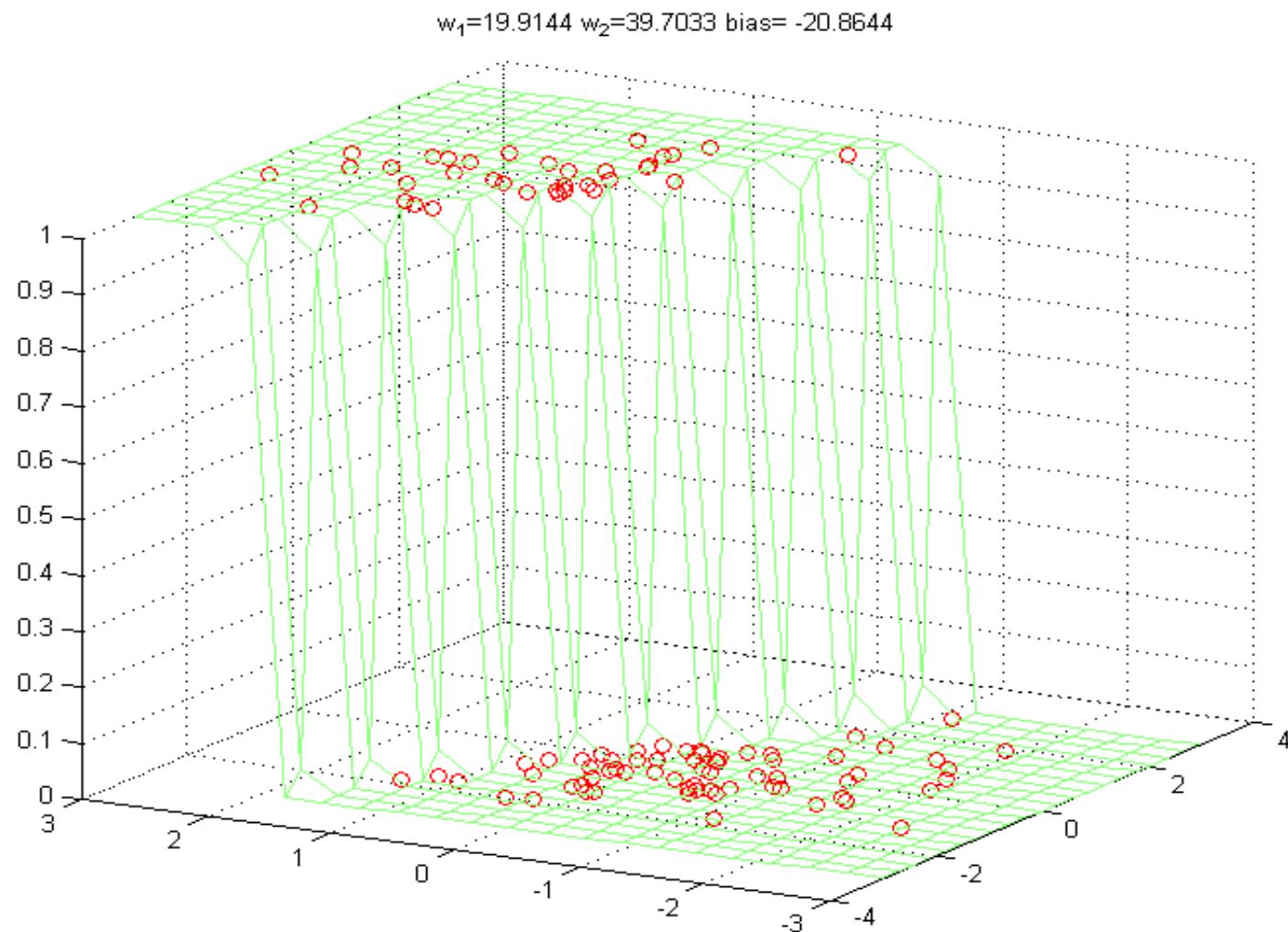
# Gradient algorithm. Example.



# Gradient algorithm. Example.

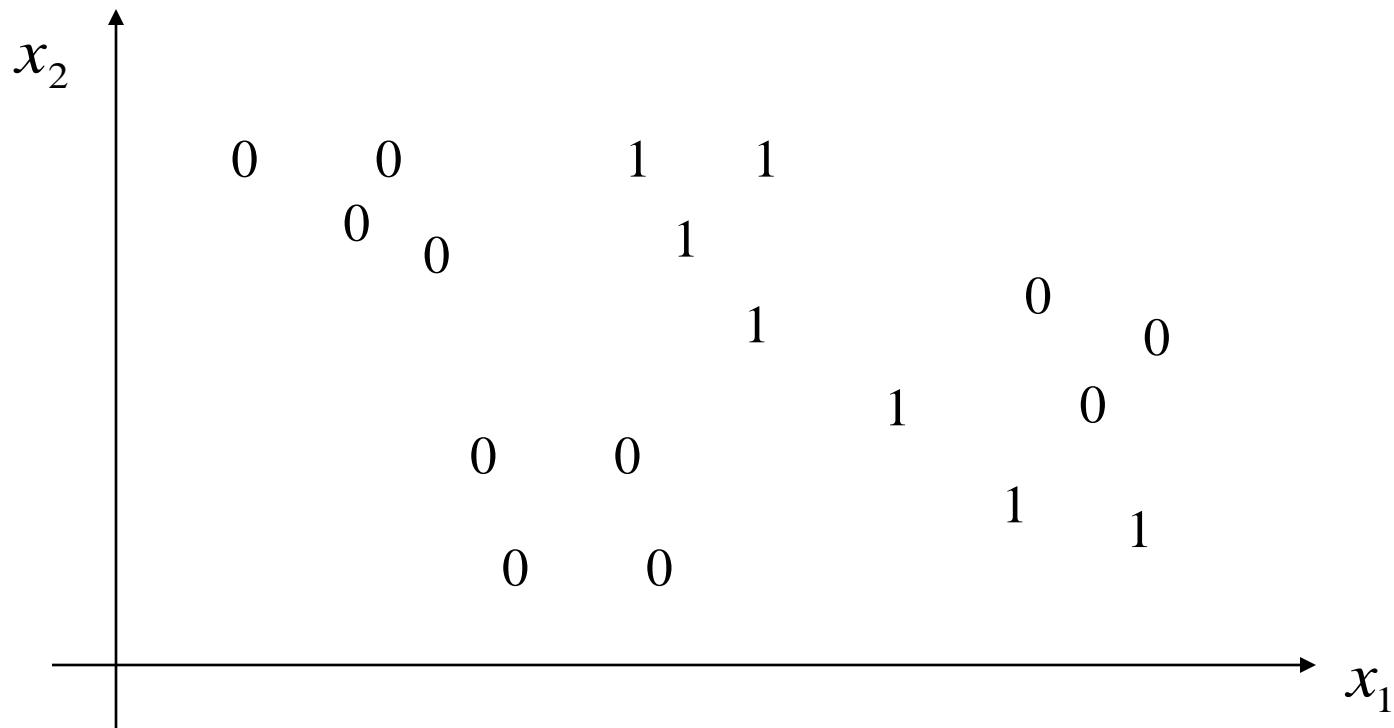


# Gradient algorithm. Example.



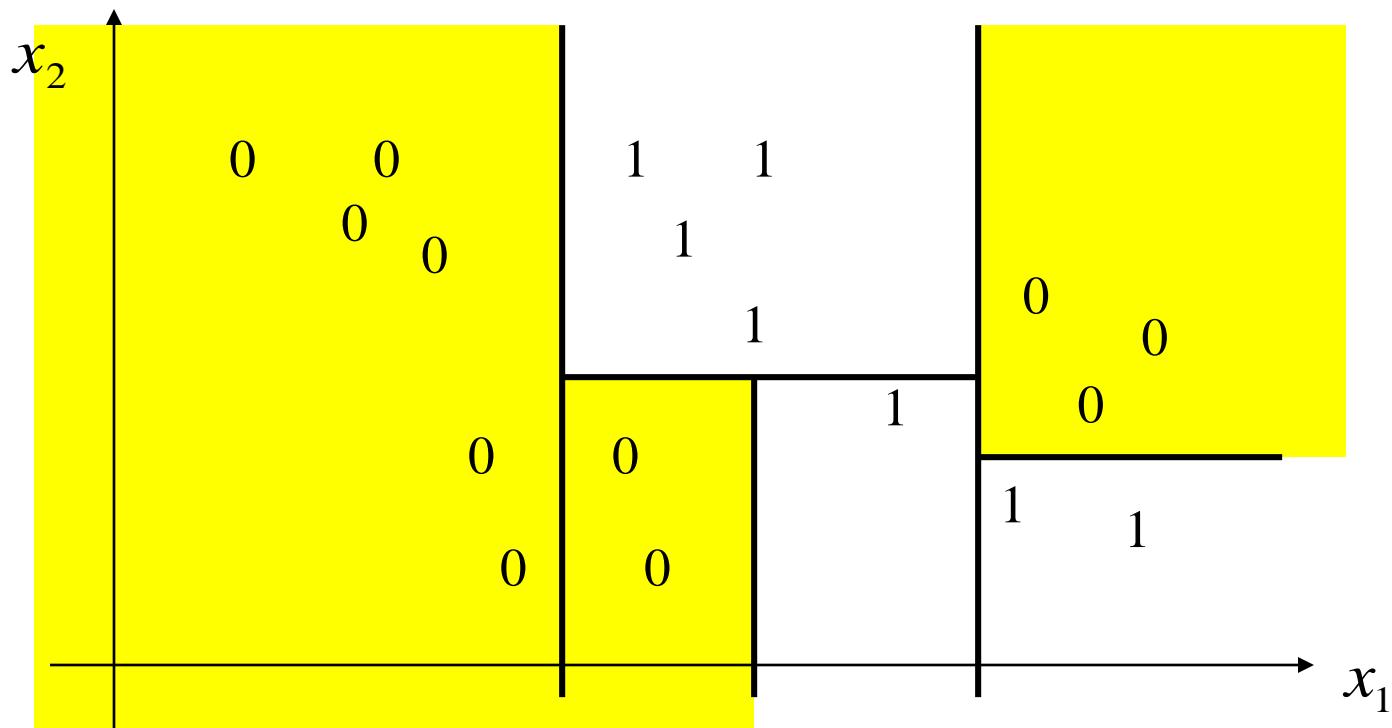
# Decision trees

- An alternative approach to classification:
  - Partition the input space to regions
  - Regress or classify independently in every region



# Decision trees

- An alternative approach to classification:
  - **Partition the input space to regions**
  - **Regress or classify independently in every region**



# Decision trees

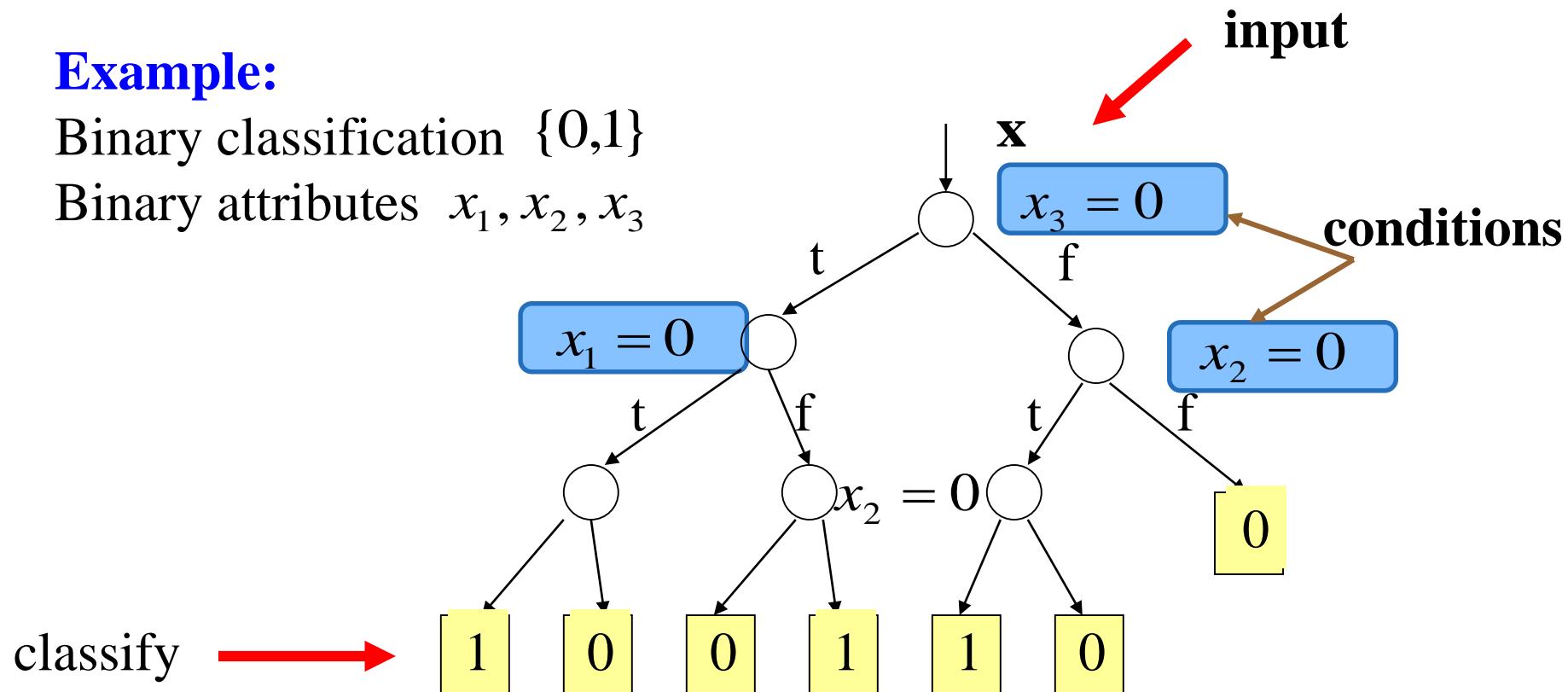
- **Decision tree model:**

- Split the space recursively according to inputs in  $\mathbf{x}$
- Classify at the bottom of the tree

**Example:**

Binary classification  $\{0,1\}$

Binary attributes  $x_1, x_2, x_3$



# Decision trees

- **Decision tree model:**

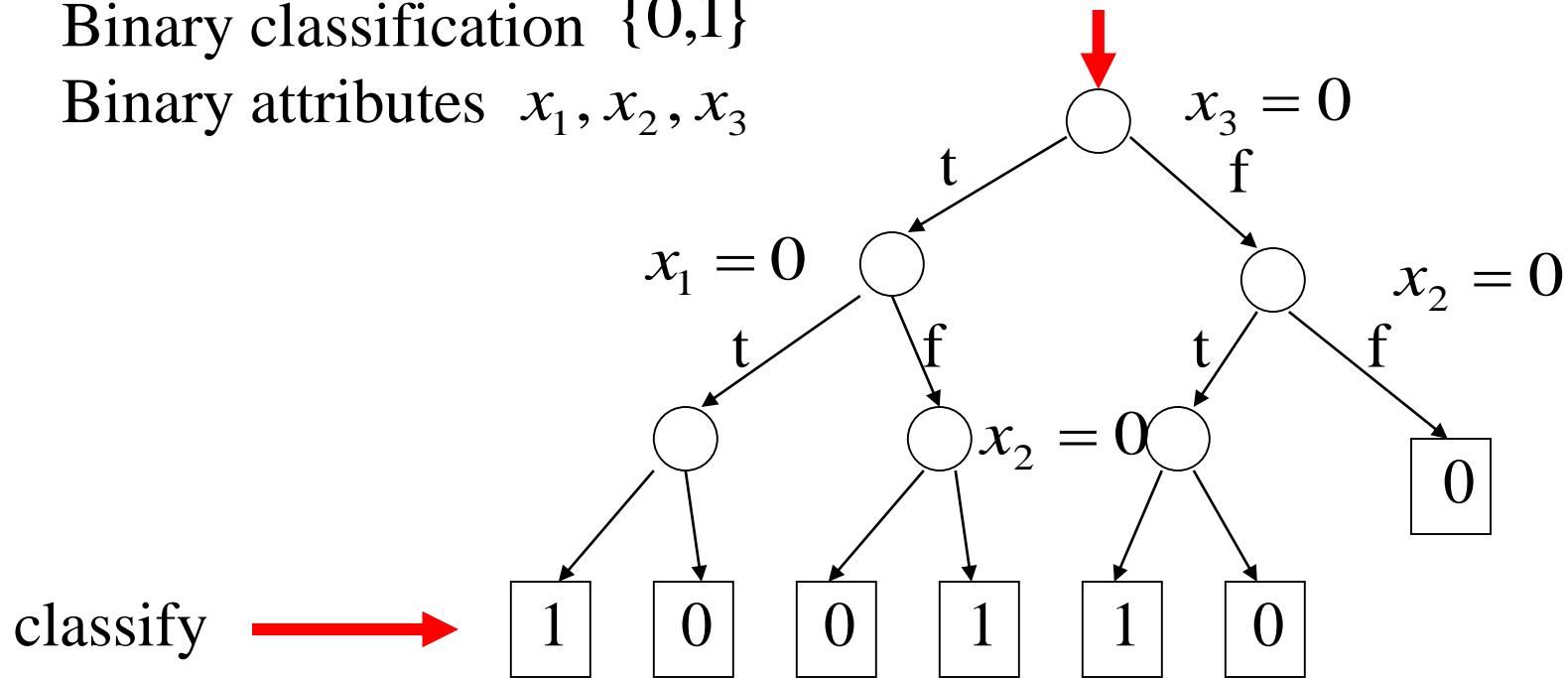
- Split the space recursively according to inputs in  $\mathbf{x}$
- Classify at the bottom of the tree

**Example:**

Binary classification  $\{0,1\}$

Binary attributes  $x_1, x_2, x_3$

$$\mathbf{x} = (x_1, x_2, x_3) = (1, 0, 0)$$



# Decision trees

- **Decision tree model:**

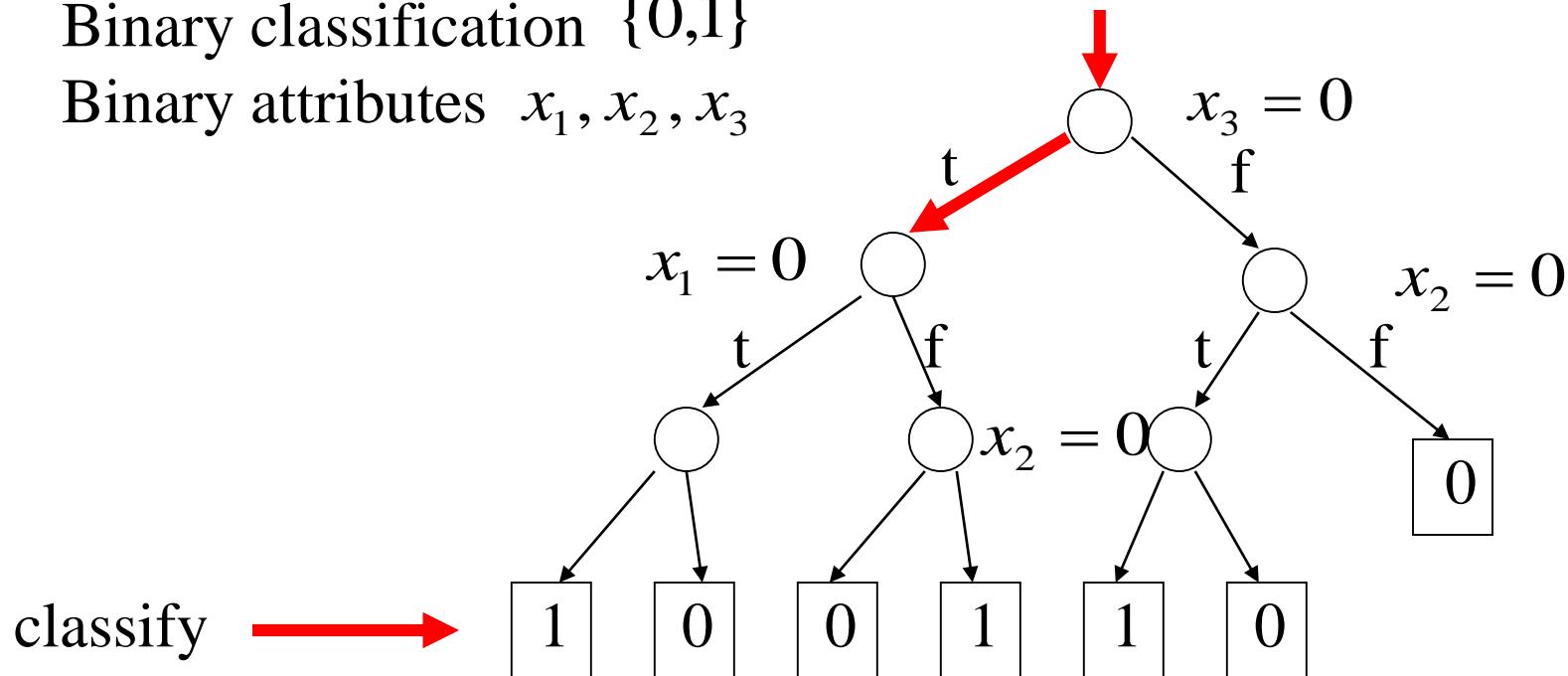
- Split the space recursively according to inputs in  $\mathbf{x}$
- Classify at the bottom of the tree

**Example:**

Binary classification  $\{0,1\}$

Binary attributes  $x_1, x_2, x_3$

$$\mathbf{x} = (x_1, x_2, x_3) = (1, 0, 0)$$



# Decision trees

- **Decision tree model:**

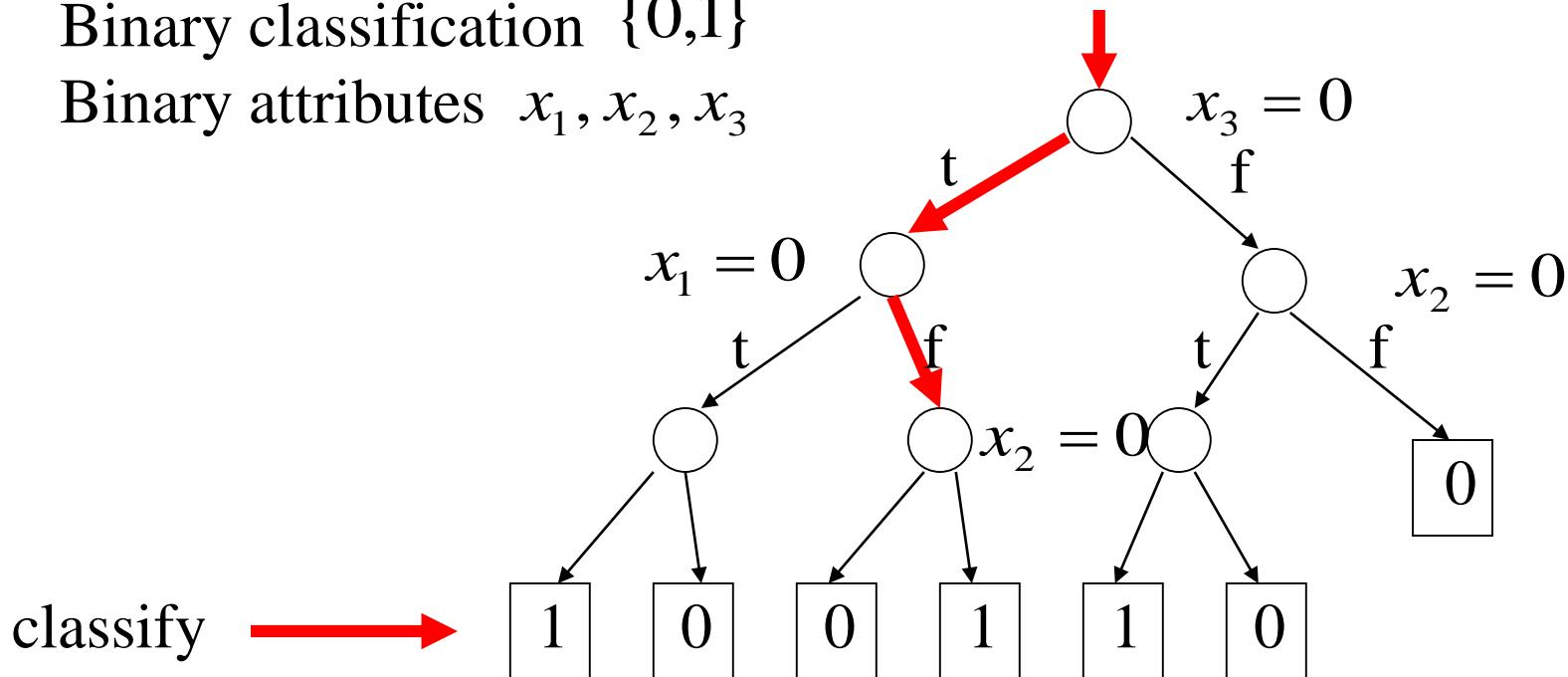
- Split the space recursively according to inputs in  $\mathbf{x}$
- Classify at the bottom of the tree

**Example:**

Binary classification  $\{0,1\}$

Binary attributes  $x_1, x_2, x_3$

$$\mathbf{x} = (x_1, x_2, x_3) = (1, 0, 0)$$



# Decision trees

- **Decision tree model:**

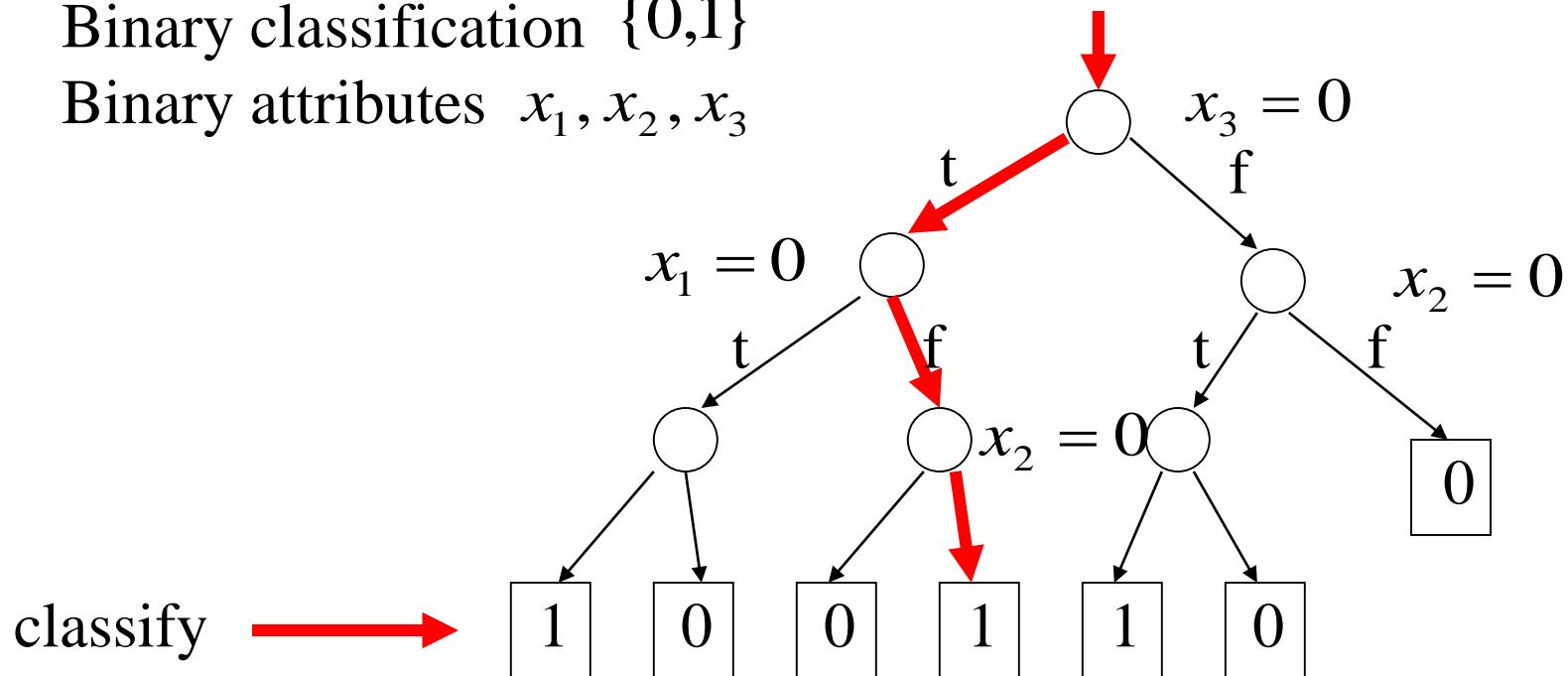
- Split the space recursively according to inputs in  $\mathbf{x}$
- Classify at the bottom of the tree

**Example:**

Binary classification  $\{0,1\}$

Binary attributes  $x_1, x_2, x_3$

$$\mathbf{x} = (x_1, x_2, x_3) = (1, 0, 0)$$

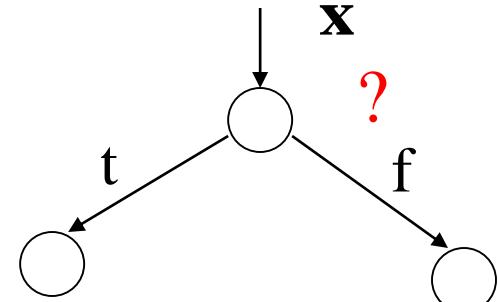


# Learning decision trees

How to construct /learn the decision tree?

- **Top-bottom algorithm:**
  - Find the best split condition (quantified based on the impurity measure)
  - Stops when no improvement possible
- **Impurity measure I:**
  - measures how well are the two classes in the training data D separated ....  $I(D)$
  - Ideally we would like to separate all 0s and 1s
- Splits: **finite or continuous value attributes**

Continuous value attributes conditions:  $x_3 \leq 0.5$



# Impurity measure

Let  $|D|$  - Total number of data entries in the training dataset

$|D_i|$  - Number of data entries classified as  $i$

$$p_i = \frac{|D_i|}{|D|} \quad \text{- ratio of instances classified as } i$$

## Impurity measure I(D)

- defines how well the classes are separated
- in general the impurity measure should satisfy:
  - Largest when data are split evenly for attribute values
  - Should be 0 when all data belong to the same class

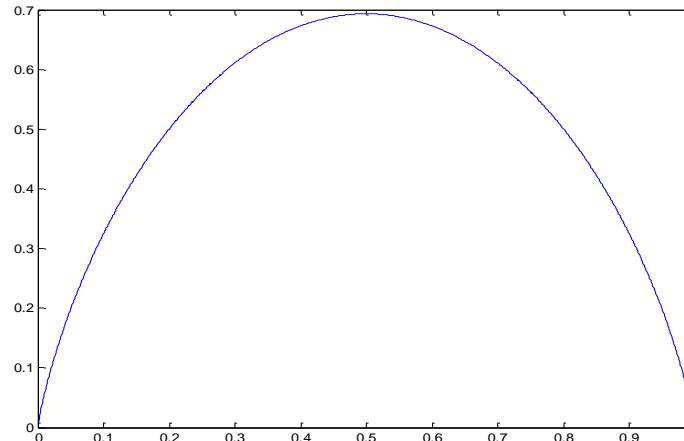
$$p_i = \frac{1}{\text{number of classes}}$$

# Impurity measures

- There are various impurity measures used in the literature
  - **Entropy based measure (Quinlan, C4.5)**

$$I(D) = Entropy(D) = -\sum_{i=1}^k p_i \log p_i$$

Example for k=2



- **Gini measure (Breiman, CART)**

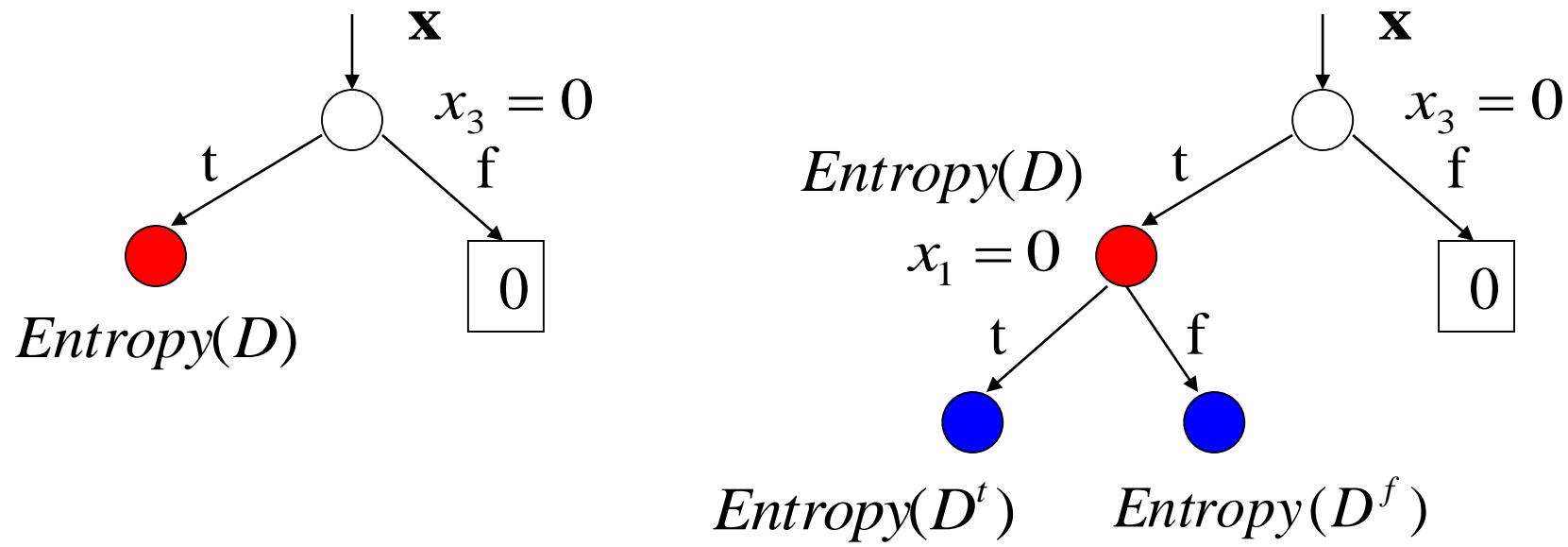
$$I(D) = Gini(D) = 1 - \sum_{i=1}^k p_i^2$$

# Impurity measures

- **Gain due to split** – expected reduction in the impurity measure (entropy example)

$$Gain(D, A) = Entropy(D) - \sum_{v \in Values(A)} \frac{|D^v|}{|D|} Entropy(D^v)$$

$|D^v|$  - a partition of  $D$  with the value of attribute  $A = v$



# Decision tree learning

- **Greedy learning algorithm:**

Repeat until no or small improvement in the purity

- Find the attribute with the highest gain
- Add the attribute to the tree and split the set accordingly

- Builds the tree in the top-down fashion

- Gradually expands the leaves of the partially built tree

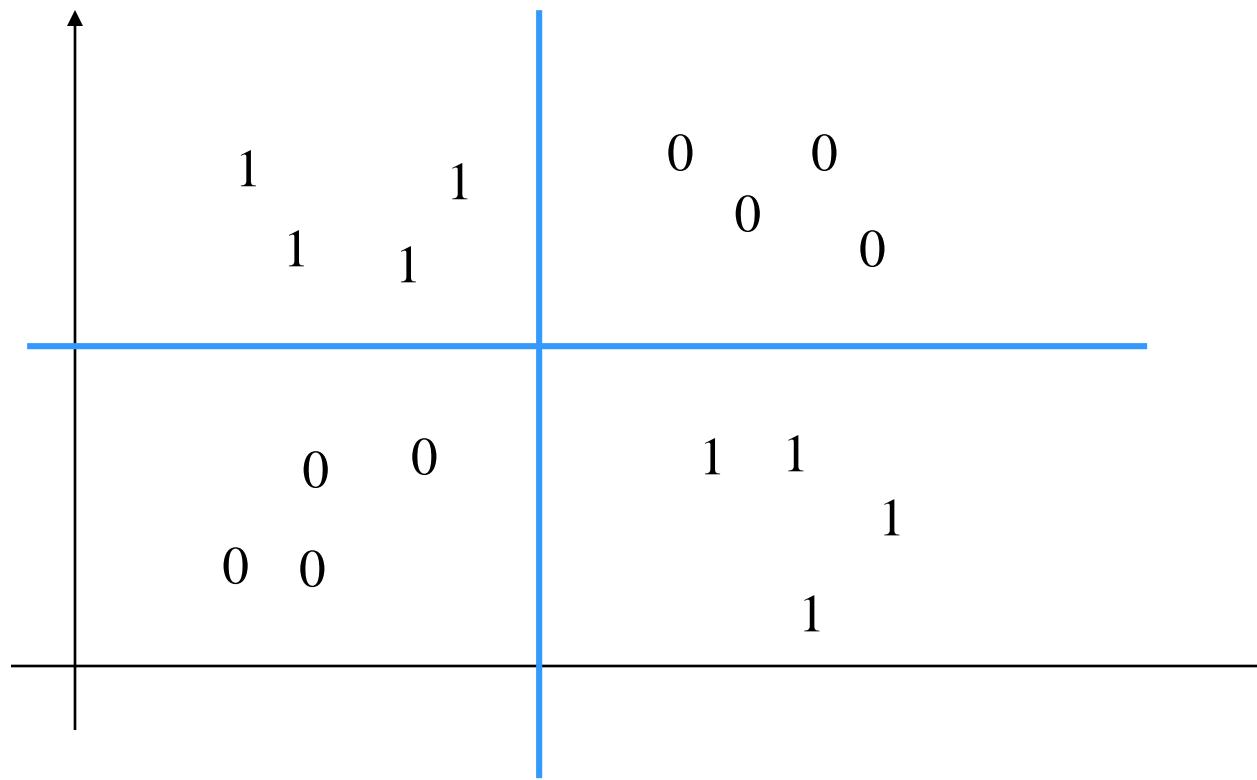
- The method is greedy

- It looks at a single attribute and gain in each step
- May fail when the combination of attributes is needed to improve the purity (parity functions)

# Decision tree learning

- **Limitations of greedy methods**

Cases in which a combination of two or more attributes improves the impurity



# Decision tree learning

By reducing the impurity measure we can grow **very large trees**

## **Problem:** Overfitting

- We may split and classify very well the training set, but we may do worse in terms of the generalization error

## **Solutions to the overfitting problem:**

- **Solution 1.**
  - Prune branches of the tree built in the first phase
  - Use validation set to test for the overfit
- **Solution 2.**
  - Test for the overfit in the tree building phase
  - Stop building the tree when performance on the validation set deteriorates

# Appendix: 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**

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

**Derivative of a logistic function**

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

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

$$\begin{aligned} \frac{\partial}{\partial z_i} [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)] &= y_i \frac{1}{g(z_i)} \frac{\partial g(z_i)}{\partial z_i} + (1 - y_i) \frac{-1}{1 - g(z_i)} \frac{\partial g(z_i)}{\partial z_i} \\ &= y_i(1 - g(z_i)) + (1 - y_i)(-g(z_i)) \end{aligned}$$

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