

## **CIS603 - Artificial Intelligence**

# **Learning**

## **Linear regression**

**Vasileios Megalooikonomou**

---

(some material adopted from notes by M. Hauskrecht)

---

CIS603 - AI

## **Learning**

- **S u p e r v i s e d :**  
REGRESSION, Classification
  
- **U n s u p e r v i s e d :**

---

CIS603 - AI

## Supervised learning

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

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

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

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

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

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

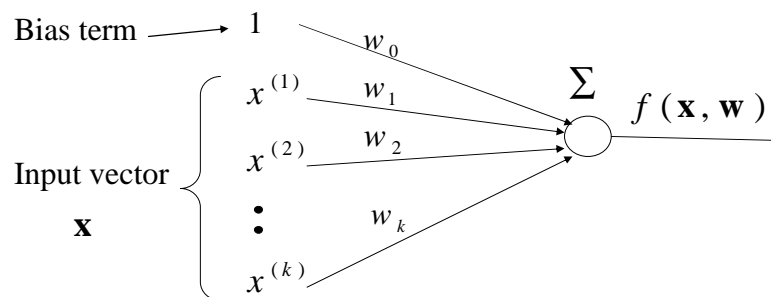
CIS603 - AI

## Linear regression

- Supervised learning:  $D = \{d_1, d_2, \dots, d_n\}$   $d_i = \langle \mathbf{x}_i, y_i \rangle$
- **Function**  $f : X \rightarrow Y$  is a linear combination of input components

$$f(\mathbf{x}) = w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \dots + w_k x^{(k)} = w_0 + \sum_{j=1}^k w_j x^{(j)}$$

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



CIS603 - AI

## 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 (loss) functions** reflects how much our prediction deviates from the correct answer, e.g.

**Least-squares error**  $Error(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2$

- **Error function for the dataset:**

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

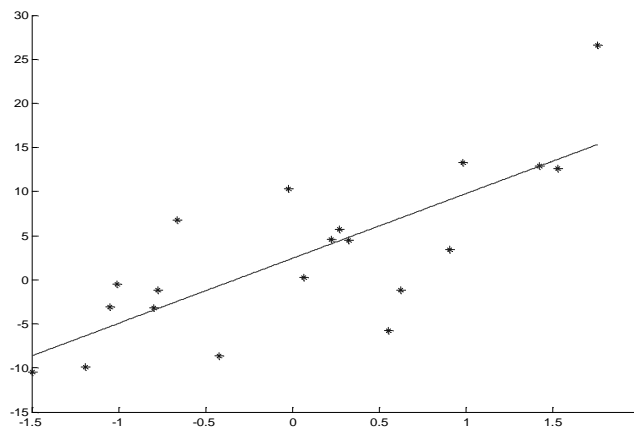
- **We want to find weights minimizing the error !**

---

CIS603 - AI

## Linear regression - Example

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

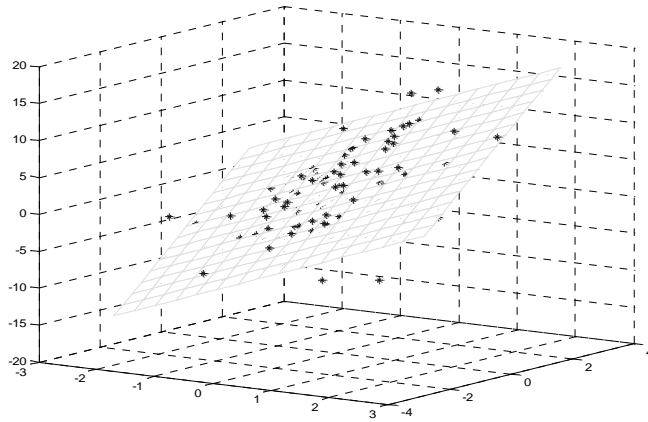


---

CIS603 - AI

## Linear regression - Example

- 2 dimensional input  $\mathbf{x} = (x^{(1)}, x^{(2)})$



CIS603 - AI

## Linear regression - Optimization

- We want to find the set of weights with the minimal error

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

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

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

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

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

CIS603 - AI

## Solving the linear regression problem

- The set of equations based on partial derivatives defines the system that is linear in unknown weights  $w_0, w_1, w_2 \dots w_k$ :
  - Can be solved using standard SLE techniques
  - Obtains the solution for all data at once
- Alternative technique for solving the same optimization problem:
  - Iterative method
  - Typically gradient-based

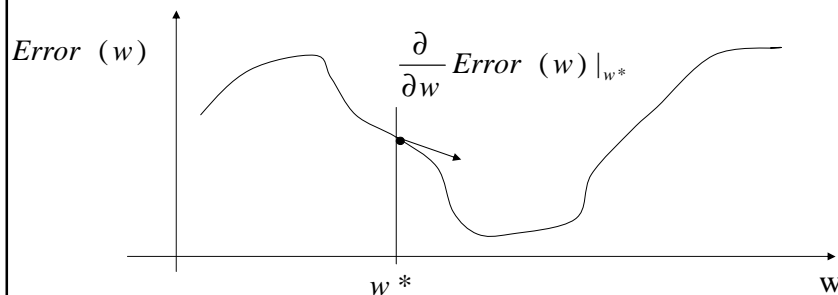
### Gradient-descent method (a special case)

- Idea: move the weights in the error decreasing direction

CIS603 - AI

## Gradient descent method

- Descend to the minimum of the function using the gradient information

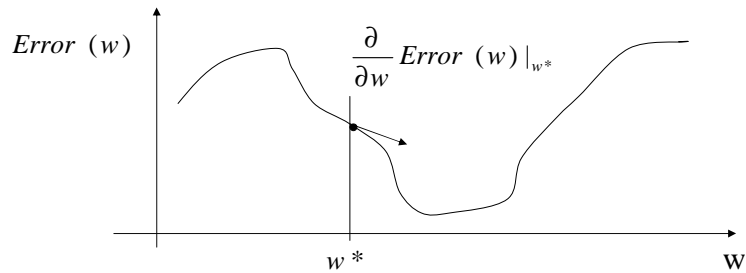


- Change the parameter value of w according to the gradient

$$w \leftarrow w^* - \alpha \frac{\partial}{\partial w} Error(w) |_{w^*}$$

CIS603 - AI

## Gradient descent method



- New value of the parameter

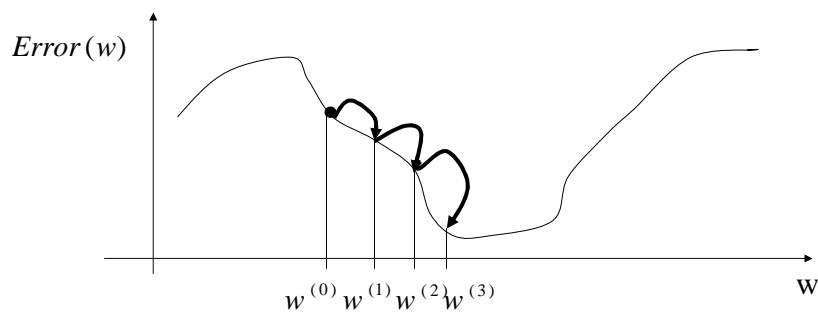
$$w \leftarrow w^* - \alpha \frac{\partial}{\partial w} Error(w) |_{w^*}$$

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

CIS603 - AI

## Gradient descent method

- To get to the function minimum repeat (iterate) the gradient based update few times



CIS603 - AI

## Online version of the gradient descent

- The error function defined for the whole dataset  $D$

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

- Instead of the error for all data points we **use error for an individual sample**

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

- Change regression weights after every sample:**

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} \text{Error}(y_i, f(\mathbf{x}_i))$$

$\alpha > 0$  - Typically annealed (depends on the number of updates)

- Advantage:** very easy to implement

CIS603 - AI

## Linear regression - On-line learning

Linear model  $f(\mathbf{x}) = w_0 + \sum_{j=1}^n w_j x^{(j)}$

Sample error  $\text{Error}(y_i, f(\mathbf{x}_i)) = \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$

**(i+1)-th update for the linear model:**

$$w_0^{(i+1)} \leftarrow w_0^{(i)} - \alpha(i+1) \frac{\partial}{\partial w_0} \text{Error}(y_i, f(\mathbf{x}_i)) \Big|_{w^{(i)}} = w_0^{(i)} + \alpha(i+1)(y_i - f(\mathbf{x}_i))$$

⋮

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

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

**Algorithm:** repeat online updates for all data points

CIS603 - AI

## Online linear regression algorithm

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

**Initialize** weights  $w_0, w_1, w_2 \dots w_k$

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

**do**    **select** a data point  $d=\langle \mathbf{x}, y \rangle$  from  $D$

**set**  $\alpha = 1/i$

**update** weights (in parallel)

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

    •

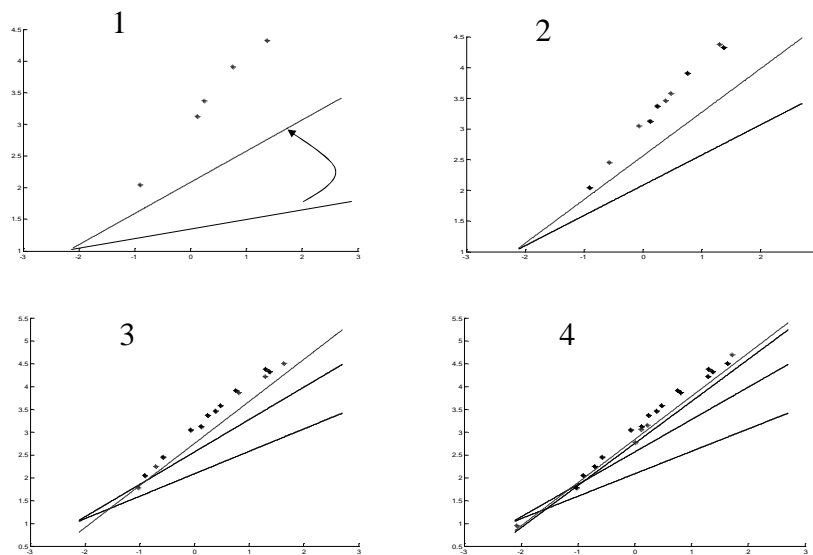
$$w_j = w_j + \alpha(y - f(\mathbf{x}, \mathbf{w}))x^{(j)}$$

**end for**

**return** weights

CIS603 - AI

## On-line learning. Example



CIS603 - AI