Universidade do Minho
Departamento de Informática

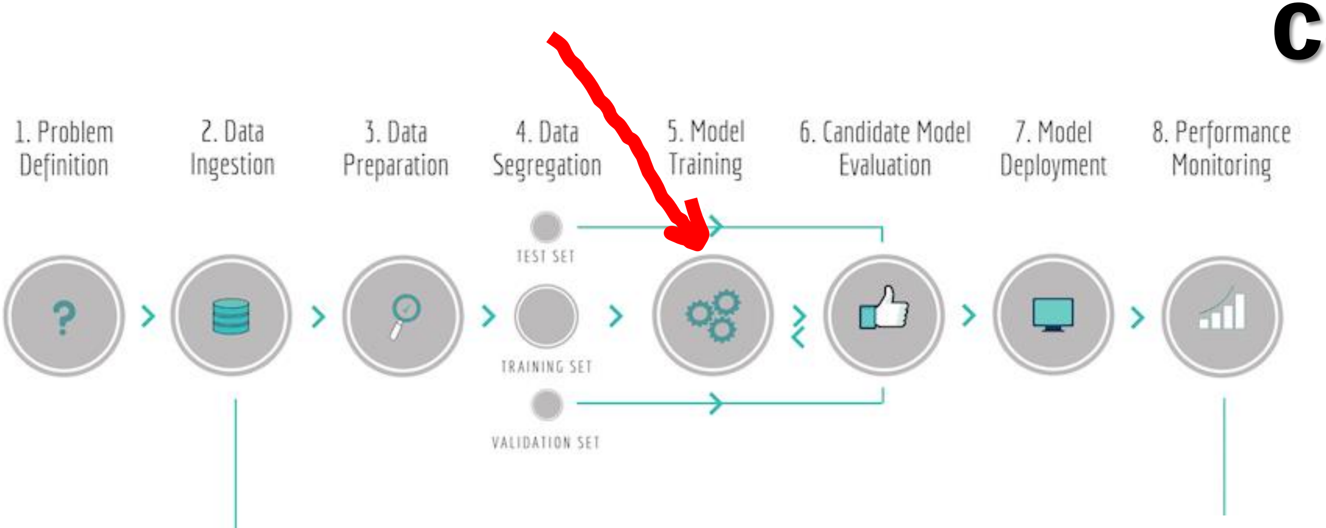
Synthetic Intelligence Lab

DADOS e APRENDIZAGEM AUTOMÁTICA

Supervised Learning
Linear and Logistic Regression

MESTRADO (integrado) EM ENGENHARIA INFORMÁTICA

Contents



Supervised Learning:

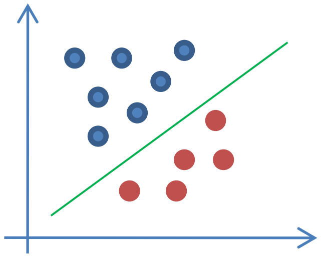
- **Linear Regression;**
- **Logistic Regression**

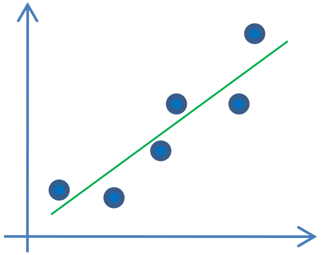
Linear models

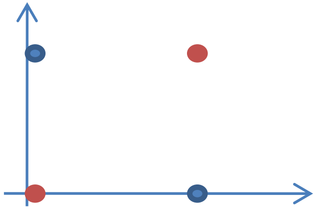
- Characterized by the simplicity of calculation and analysis
- Linearity is defined in terms of functions with the properties:

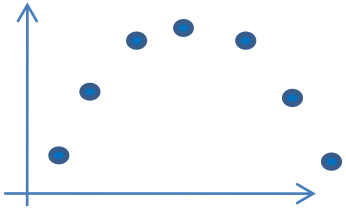
$\mathrm{f}(\mathrm{x}+\mathrm{y})=\mathrm{f}\,(\mathrm{x})+\mathrm{f}\,(\mathrm{y})$ and $\mathrm{f}(a\mathrm{x})=a\mathrm{f}(\mathrm{x})$

- Used for **classification** (separation between classes) or **regression**
- Does not solve non-linear problems


Classification


Regression


Classification
(e.g. XOR)

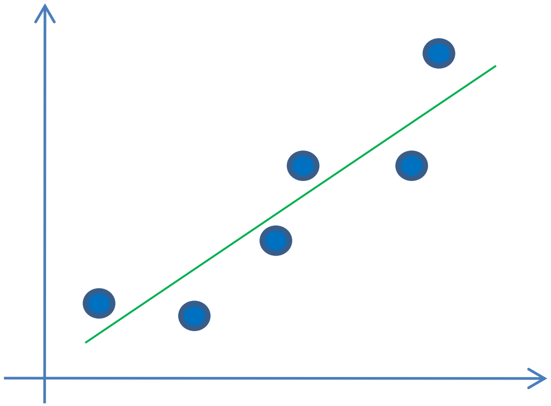

Regression

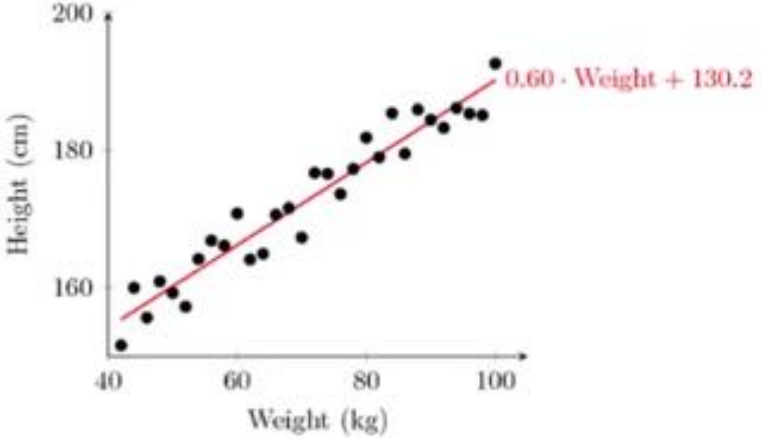
Non-linear problems

Linear Regression

Aims to predict the value of a outcome, Y, based on the value of an predictor variable, X.

- Fit a straight line into a data set of observations;
- Use this line to predict unobserved values.





Linear regression models

Represent the relationship between **input variables** $\boldsymbol{x_1}$, …, $\boldsymbol{x_n}$ (independent variables), and an **output variable y** (dependent variable).

Model prediction given by (for the **i**-th example):

n – no. of attributes

θ – model parameters

$$\hat{y}^{(i)} = h_\theta(x_1^{(i)}, \ldots, x_n^{(i)})$$

General case: **regression** models

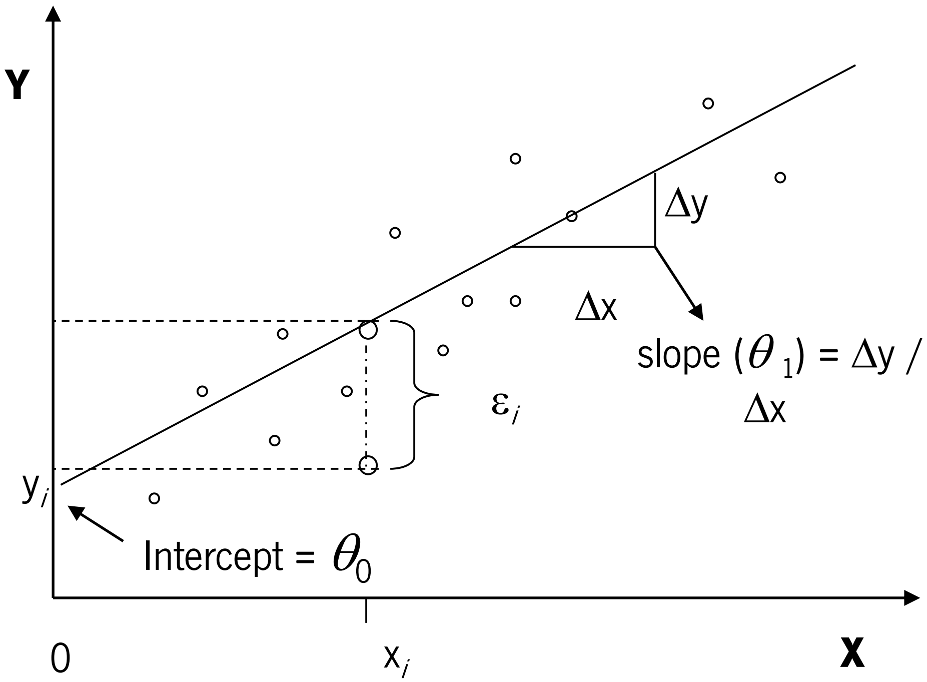
If **n=1**: **linear regression**

If **n>=2**: **multiple linear regression**

$$\hat{y}^{(i)} = h_q(x^{(i)}) = q_0 + \overset{n}{\underset{j=1}{å}} q_j x_j^{(i)}$$

θ_i – model parameters

Linear Regression



Model:

$$\hat{y}^{(i)} = h_\theta(x^{(i)}) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j^{(i)}$$

Linear Regression Models

How does it work?

Usually using “Error/loss (cost) function” and minimizing its value (minimize the squared-error between each point and the line)

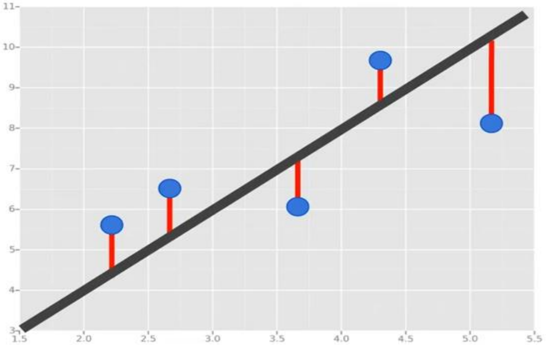
Error/loss (cost) function: mean squares errores – MSE

$$J_q = \frac{1}{2m} \overset{m}{\underset{i=1}{\text{å}}} (h_q(x^{(i)}) - y^{(i)})^2$$

J is a function of the model parameters $\theta_1, \ldots, \theta_n$

$\mathbf{h_\theta}(\mathbf{x}^{(\mathbf{i})})$ is the value predicted by the model, $\hat{\mathbf{y}}^{(\mathbf{i})}$

$\mathbf{y}^{(\mathbf{i})}$ is the real value



Objective: to identify the parameters of the model in order to **minimize** the value of **J**

Multiple linear regression

- Multiple regression is used to determine the effect of a number of independent variables, x_1, x_2, x_3 etc, on a single dependent variable, y
- The different x variables are combined in a linear way and each has its own regression coefficient (**θ**):

$$\hat{y}^{(i)} = \theta_0 + \sum_{j=1}^{n} \theta_j x_j^{(i)}$$

Model prediction given by (for the **i**-th example)

n – no. of attributes

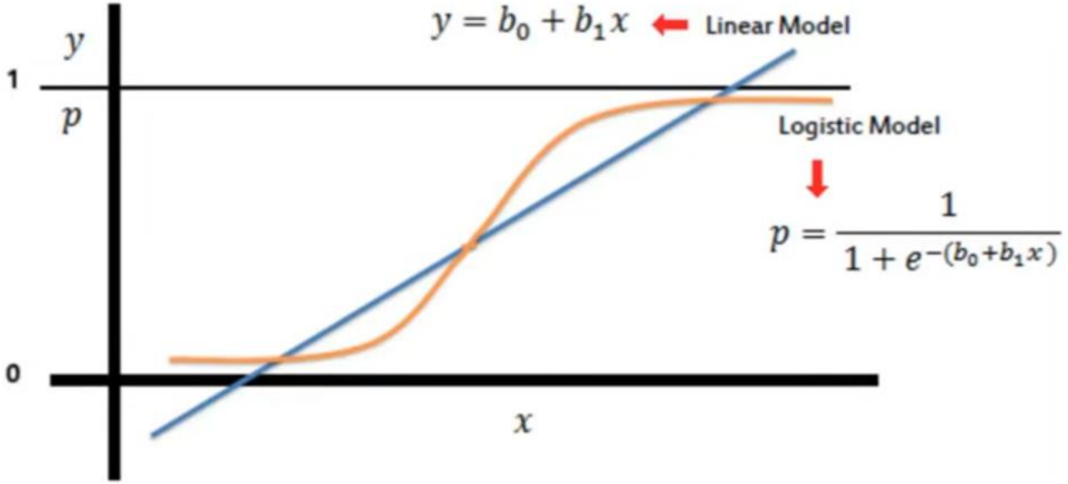
θ – model parameters

- The a parameters reflect the independent contribution of each independent variable, x, to the value of the dependent variable, y.

Logistic Regression

Discrete dependent variable: **classification** problem

Logistic regression: uses regression models for **binary classification** by interpreting the **model output** in order to extract a **class**



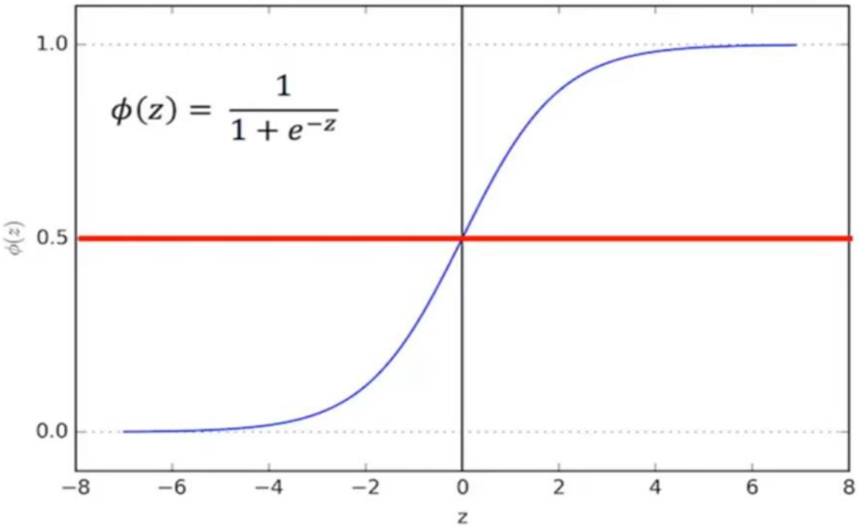
where $\frac{1}{1+e^{-\theta^T x}}$ is the sigmoid (logistic) function

The Model is given by applying the sigmoid to the linear regression function

Interpretation: ***p*** estimates the probability that **y** (output) **is equal to 1** for example **x**

Sigmoid Function

- The Sigmoid (i.e. Logistic) function takes in any value and outputs it between [0-1];
- This results in a probability from [0-1] of belonging into a class.
- We can set a threshold point at 0.5, defining:
  - ➢ Based off this probability, we assign a class
  - ➢ Predicted results below this threshold results into a class: 0
  - ➢ Predicted results above result results into a class: 1



Logistic Regression: Multiple Classes

- Logistic regression can be applied to cases with **more than two classes**
- In this case, the strategy is **to train a "binary" model for each class separately** (considering the others as a single class)
- Each model estimates the **probability** that **the example is of a given class**
- When predicting new examples, each model is applied by **choosing the class** whose **value predicted** by the model is **greater**.

Logistic Regression: Error Function

Error function (for each example x):

$$\begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

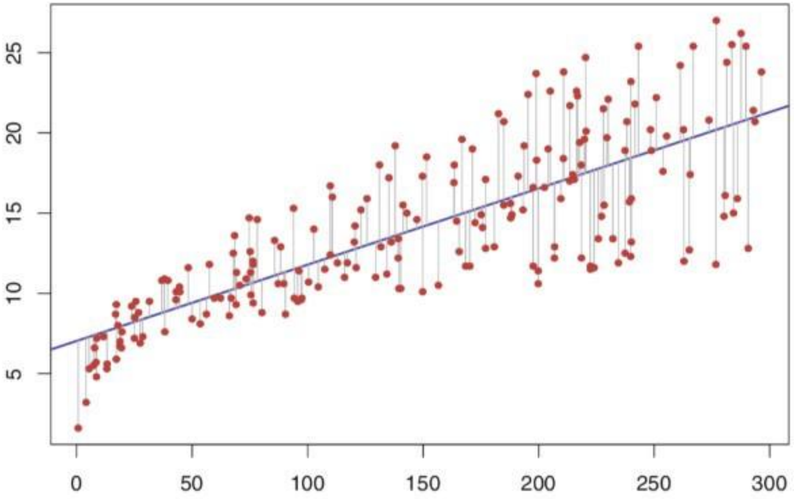
If **y = 1**:
If the prediction is **correct**: **error is zero**
Otherwise, as the prediction gets closer to 0, error **tends to infinity**.

If **y = 0**:
If prediction is correct: **error is zero**
Otherwise, as the prediction gets closer to 1, error **tends to infinity**.

Parameter Estimation: Optimization

Knowing the model structure: parameter estimation is a **numerical optimization problem** – error function minimization

In the case of linear models, the **least squares method** can be used, which minimizes the error function (square of errors) or iterative method



Parameter Estimation: Least Squares Method for Linear Regression

Analytical method to determine optimal values of parameters that minimize **J**

Algebraic method that involves solving a system of equations given by:

$$\frac{\partial}{\partial\theta_j}J(\theta)=0,\ j=1,\ldots,n$$

$$q=(X^TX)^{-1}X^Ty$$

Matricial version; **X** matrix includes examples + 1st collumn of 1’s

The **computational complexity** when training a linear regression model using the least squares method is linear with respect to the number of instances and features.

Parameter Estimation: Gradient Descent for Linear Regression

Method that depends on whether the error function is differentiable

Iterative method, which in each iteration changes the values of each of the parameters $θ_j$

For each $θ_j$ the update rule is as follows:

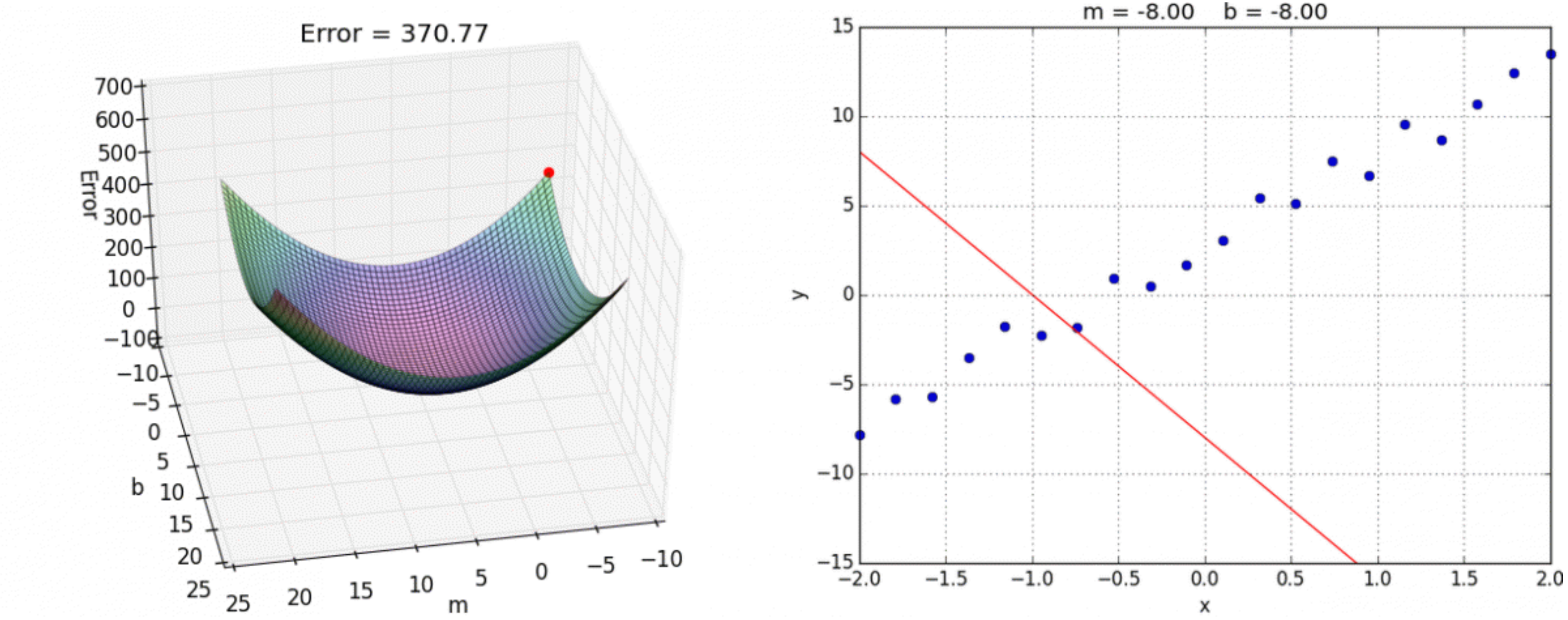
$$q_j := q_j - a \frac{¶}{¶ q_j} J(q)$$

$$q_j := q_j - a \frac{1}{m} \overset{m}{\underset{i=1}{å}} (h(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Simultaneous updates on all parameters

J is the error, h is the model and x is the vector with the attributes, m is the number of instances
The method starts the theta (parameters) with random values and improves gradually at each iteration, taking a small step at each iteration. The size of the step is called "learning rate".
To implement the gradient descent, we need to calculate how much the cost function will change if we change just a little bit of the parameters (theta). This is called the partial derivative. The derivative gives us the rate of change of a function at a given point. When we have a variation equal to zero it means we have reached the objective of the function.

Parameter Estimation: Gradient Descent for Linear Regression

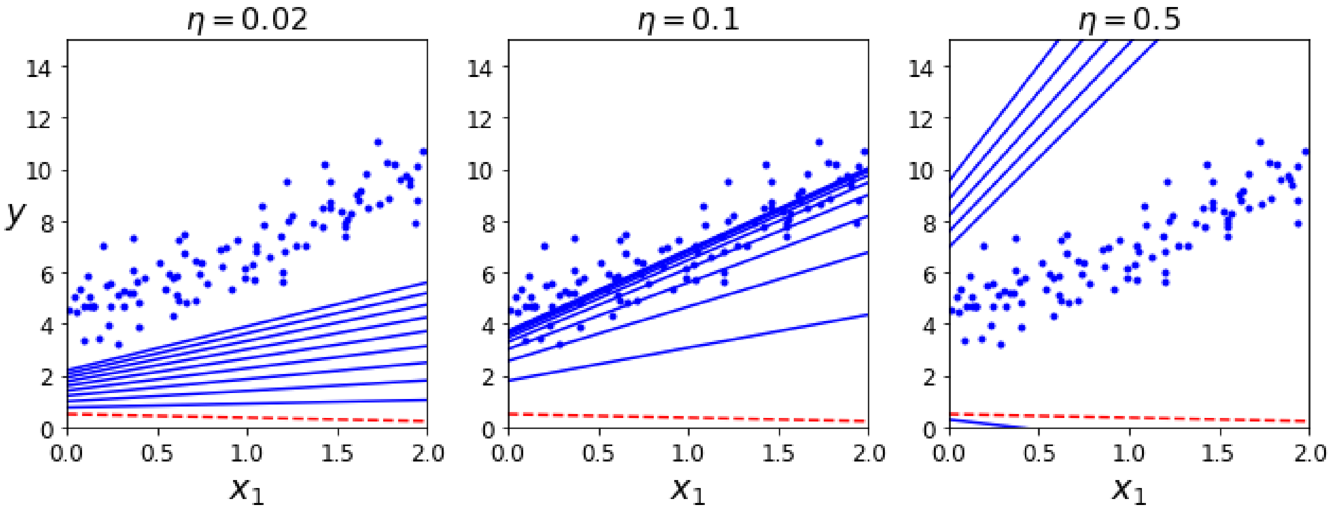


Parameter Estimation: Gradient Descent

The parameter **α** is called the **learning rate** and controls the **"speed"** of updating the parameters

Lower α values guarantee convergence but it may be slower

Higher α values can lead to **faster convergence**, but carry risks of divergence



Parameter Estimation: Gradient Descent vs. Analytic Method

Analytical method guarantees the optimal solution; **GD may not converge**

In the analytical method there are **no parameters**; **GD may take time to converge**

Analytical method can become slow with **very large n** (**n×n matrices** can become **intractable** for **n>105**)

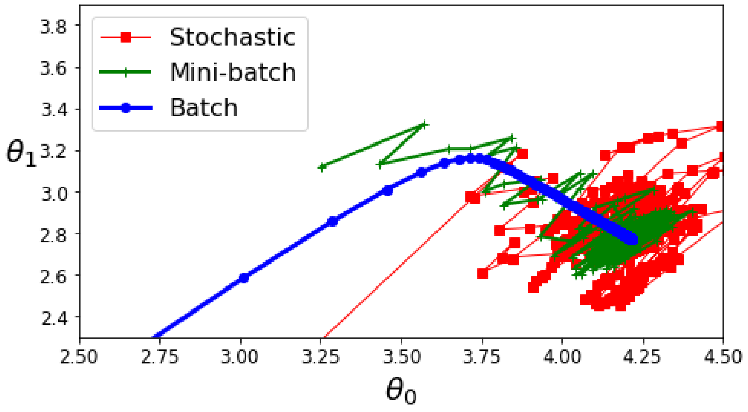
More generic GD and applicable to other types of models

Parameter Estimation: Advanced Methods

In many cases, gradient descent is **too slow to converge** to be used in practice

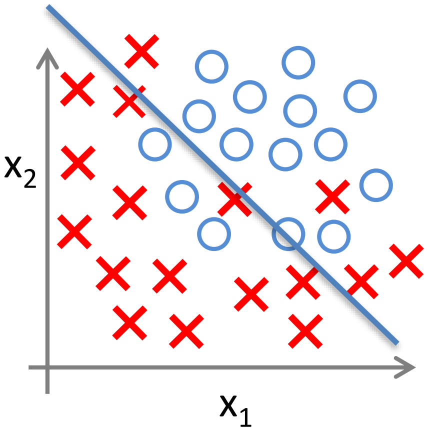
Other more advanced numerical optimization methods can be used

Python example with the **fmin** function from package **optimize** (in this case, no derivatives needed). Other alternatives available in the same package



Algorithm	Many Instances (m)	Many Attributes (n)
Square Min.	Fast	Slow
Batch GD	Lento	Fast
Stocastic GD	Fast	Fast
Mini Batch GD	Fast	Fast

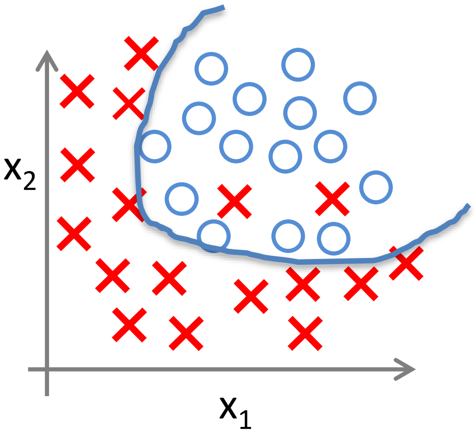
Overfitting in Logistic Regression: example



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

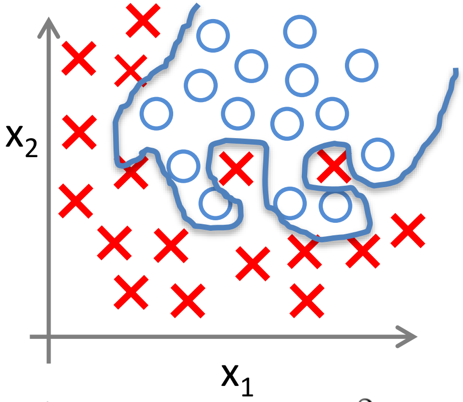
g is a sigmoid function

Underfitting:
insuficient complexity



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 \\ + \theta_3 x_1^2 + \theta_4 x_2^2 \\ + \theta_5 x_1 x_2)$$

“Adequate” complexity



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 \\ + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 \\ + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots)$$

Overfitting:
excessive complexity

Solutions for Overfitting: Functional Models

Reduce the number of attributes (coefficients) used

Select attributes “**manually**” by knowledge of the problem

Attribute selection algorithms

Regularization:

Keep all attributes but **try to reduce magnitude** of parameter values

Standardization and Normalization

Transformations in the data are often necessary for the learning algorithm to **work better**

Gradient descent algorithms may **work worse** with variables with v**ery different scales**

Several possible methods:

- Convert to **mean 0** and **standard deviation 1**
- Convert to a **[0,1]** or **[-1,1] range**, setting **minimum and maximum values**

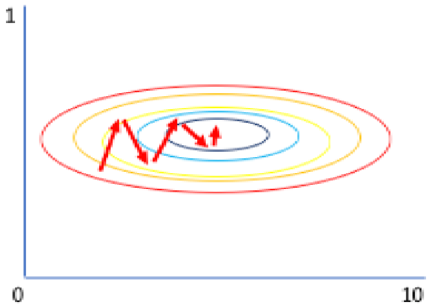
Standardization and Normalization

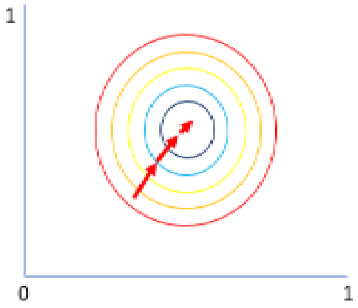
Transformations in the data are often necessary for the learning algorithm to **work better**

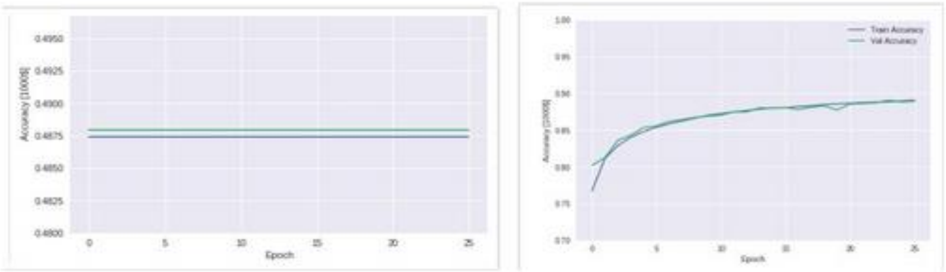
Gradient descent algorithms may **work worse** with variables with v**ery different scales**

Several possible methods:

- Convert to **mean 0** and **standard deviation 1**
- Convert to a **[0,1]** or **[-1,1] range**, setting **minimum and maximum values**







Left: Model Accuracy, without normalized data
Right: Model Accuracy with normalized data