ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Machine Learning

Lecture 2: Linear regression

Petia Georgieva
(petia@ua.pt)

universidade de aveiro

LINEAR REGRESSION - outline

1. Univariate linear regression apenas um parâmetro

- Cost (loss) function - Mean Squared Error (MSE)
- Cost function convergence
- Gradient descent algorithm

2. Multivariate linear regression output; more than one feature

-Overfitting problem

3. Regularization => way to deal with overfitting

CLASSIFICATION vs REGRESSION

Classification- the model output is a label (e.g. integer numbers 0, 1, -1, etc.) o 0 as vezes costuma ser evento normal

Regression - the model output is a real number

Examples of regression problems:

- Weather forecast
- Predicting wind velocity from temperature, humidity, air pressure
- Time series prediction of stock market indices
- Predicting sales amounts of new product based on advertising expenditure
- Equalization in communication channels (IT-UA)

Standard Notations in this course

x – input vector of features, attributes

y – output vector of labels, ground truth, target

m - number of training examples

n – number of features

$h_\theta(x)$ - model (hypothesis)

θ - vector of model parameters   objetivo; optimizar os parametros

Training set: data matrix X (m rows, n columns)

	feature x_1	feature x_2	.....	feature x_n	output(label) y
Example 1	$x_1^{(1)}$			$x_n^{(1)}$	$y^{(1)}$
Example 2	$x_1^{(2)}$			$x_n^{(2)}$	$y^{(2)}$
...					
Example i	$x_1^{(i)}$			$x_n^{(i)}$	$y^{(i)}$
...					
...					
Example m	$x_1^{(m)}$			$x_n^{(m)}$	$y^{(m)}$

Supervised Learning – univariate regression

Problem: Learning to predict the housing prices (output) as a function of the living area (input, data feature)

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Supervised Learning – univariate regression

usamos os parametros para estimar o model

linear problem

$$h_\theta(x) = \theta_0 + \theta_1 x_1 = \begin{bmatrix} 1 & x_1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \vec{x}^T \vec{\theta}$$

=> in Python => np.dot(X, Theta)

$$\vec{x} = \begin{bmatrix} x_0 = 1 \\ x_1 \end{bmatrix}$$

	X_0 (extra column)	feature x_1 (living area)	output(label) y (price)
Example 1=>	1	$x^{(1)}$	$y^{(1)}$
Example 2=>	1	$x^{(2)}$	$y^{(2)}$
	1		
Example m=>	1	$x^{(m)}$	$y^{(m)}$

Mean Square Error (MSE)

Linear Model (hypothesis) =>

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

h: vector of observed values of the variable being predicted

minimzar o erro das previsoes

Cost (loss) function **=>**
(Mean Square Error)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

y: being the predicted values

m – number of training examples

Goal **=>**

$$\min_\theta J(\theta)$$

minimizar o erro mudando os parametros

Gradient descent algorithm =>
iterative algorithm; at each iteration all parameters (theta) are updated simultaneously

primeira derivativa

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

otimizar os parametros atraves das iteracoes

alpha – learning rate > 0

universidade de aveiro

Linear Regression (computing the gradient)

Gradient descent is a method for finding the minimum of a function of multiple variables.

Cost function =>

is something you want to minimize

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Cost function gradients => vector with parcial derivatives of J with respect to each parameter for one example (m=1)

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y \right)^2 \\ &= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta(x) - y \right) \\ &= \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\ &= \left(h_\theta(x) - y \right) x_j \end{aligned}$$

Gradient descent is a method for finding the minimum of a function of multiple variables.

Gradient descent enables a model to learn the gradient or direction that the model should take in order to reduce errors

Cost function gradients => for m examples

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Linear Regression – iterative gradient descent algorithm (summary)

Inicialize model parameters *(e.g. θ =0)*
Repeat until J converge {

Compute Linear Regression Model =>

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Compute cost function =>

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

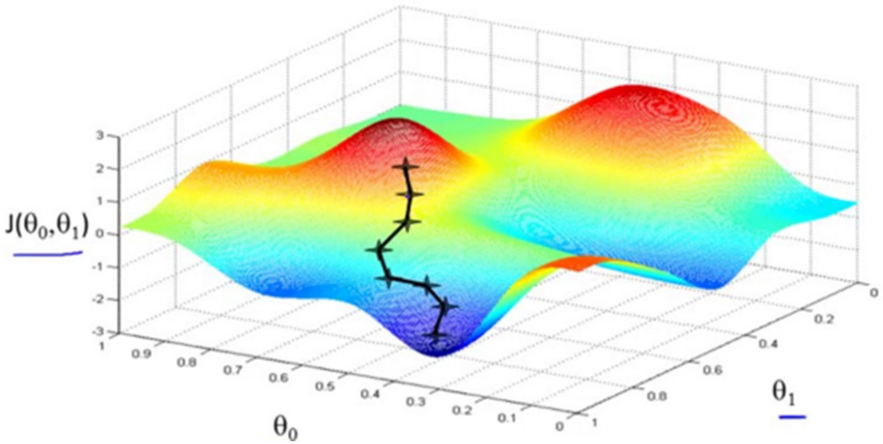
Compute cost function gradients =>

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Update parameters =>

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

}



funcao do custo

Batch/mini batch/stochastic gradient descent for parameter update

alpha -:? hyperparameter

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Batch learning (classical approach)
update parameters after all training examples have been processed, repeat several iterations until convergence

learning rate

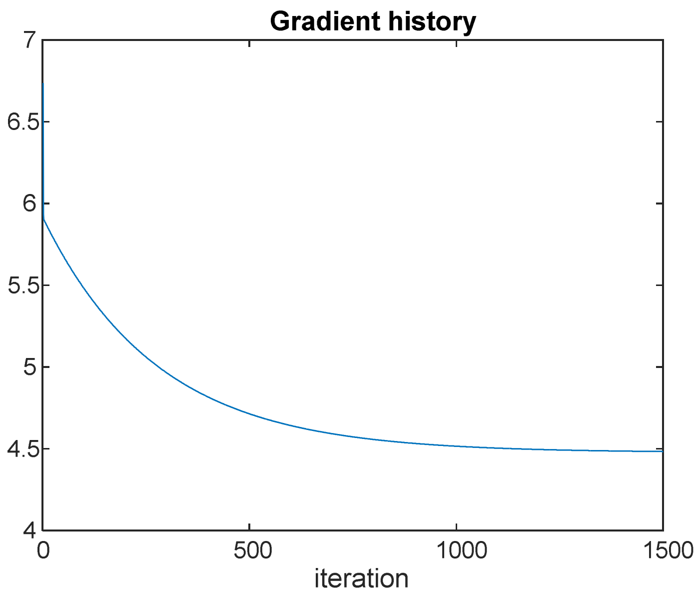
Mini batch learning (if big training data):
devide training data into small batches update parameters after each mini batch has been processed, repeat until convergence

Stochastic (incremental) learning (if small training data)
update parameters after every single training example has been processed.

universidade de aveiro

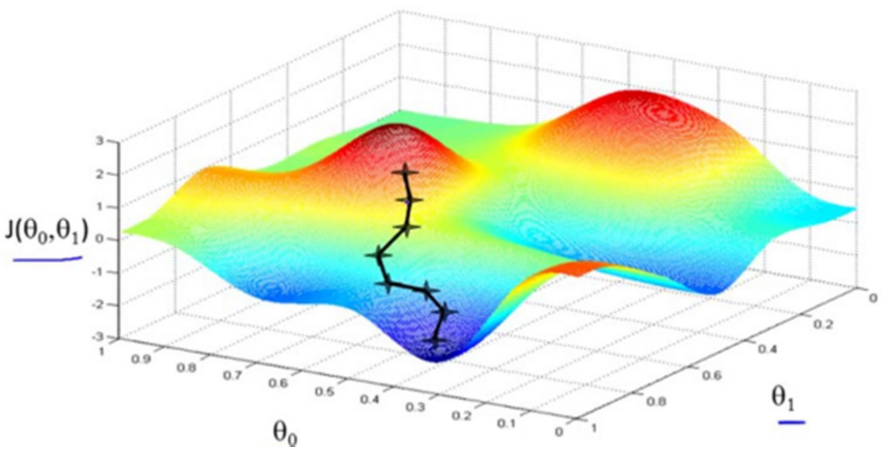
Cost function convergence

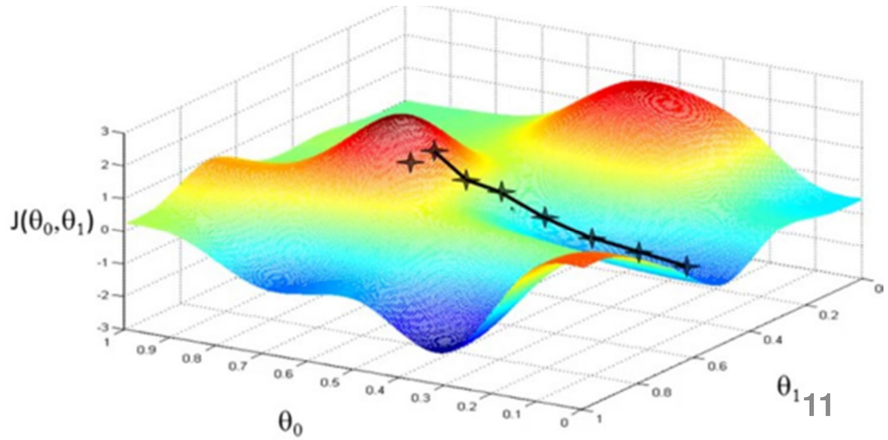
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$



Linear Regression (LR):

starting from different initial values of the parameters the cost function J should always converge (<u>**maybe to a local minimum !!!**</u>) if LR works properly.



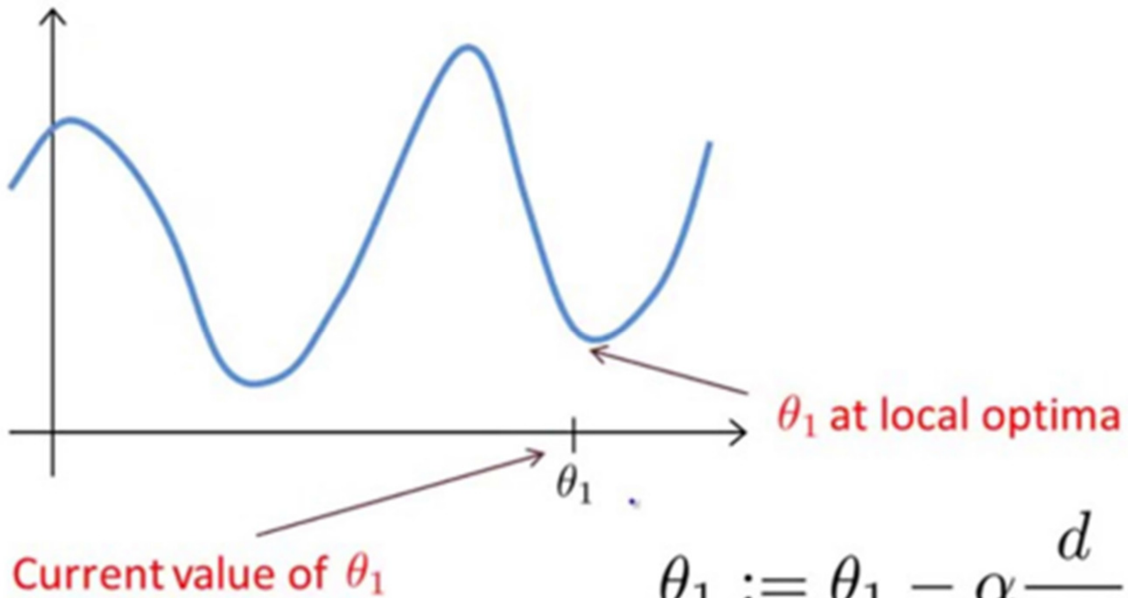


Cost function – local minimum

Suppose θ_1 is at a local optima as shown in the figure.

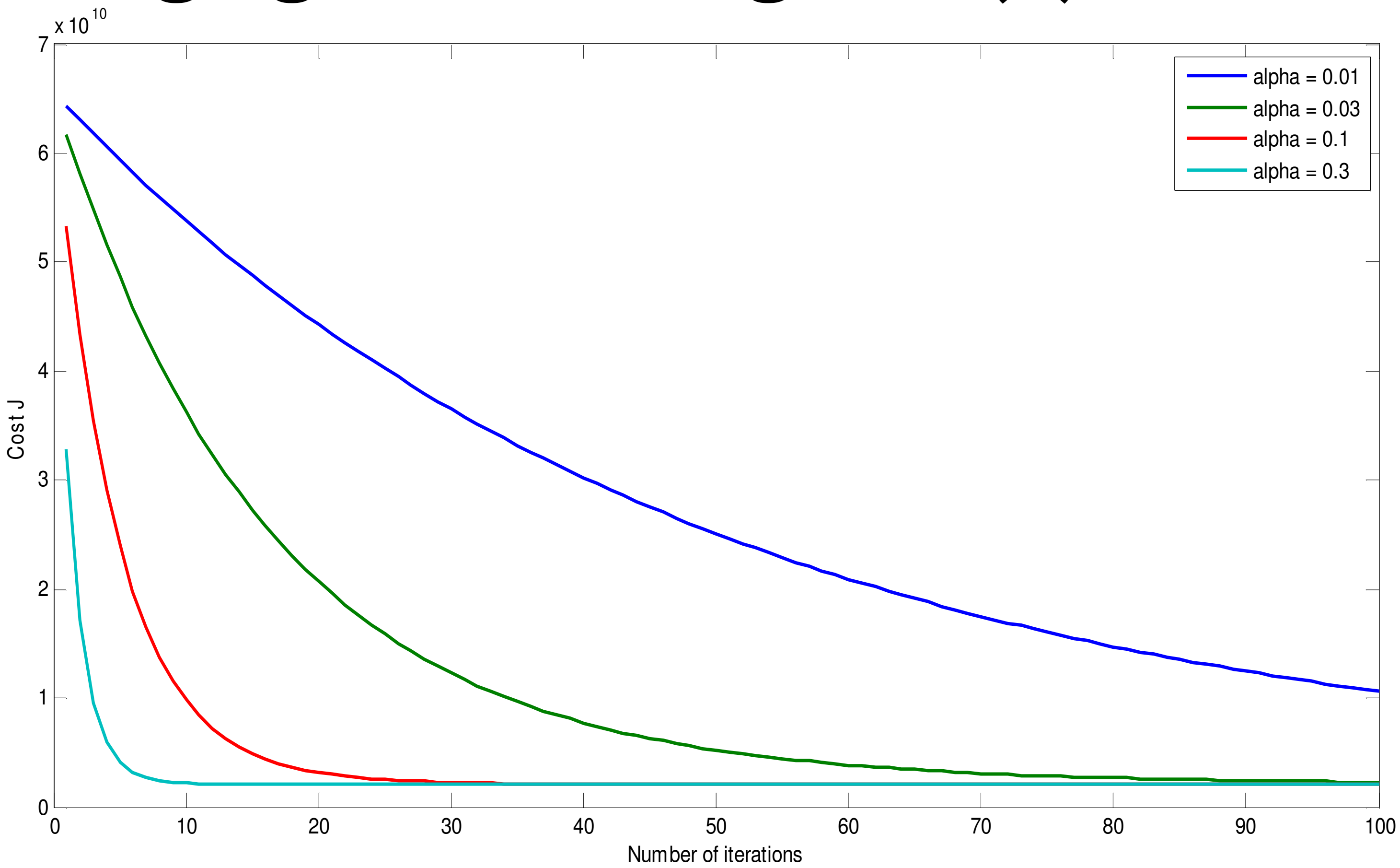
What will one step of Gradient Descent do ?

1) Leave θ_1 unchanged
2) Change θ_1 in a random direction
3) Decrease θ_1
4) Move θ_1 in direction to the global minimum of J



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Cost function convergence changing the learning rate (α) -100 iter.

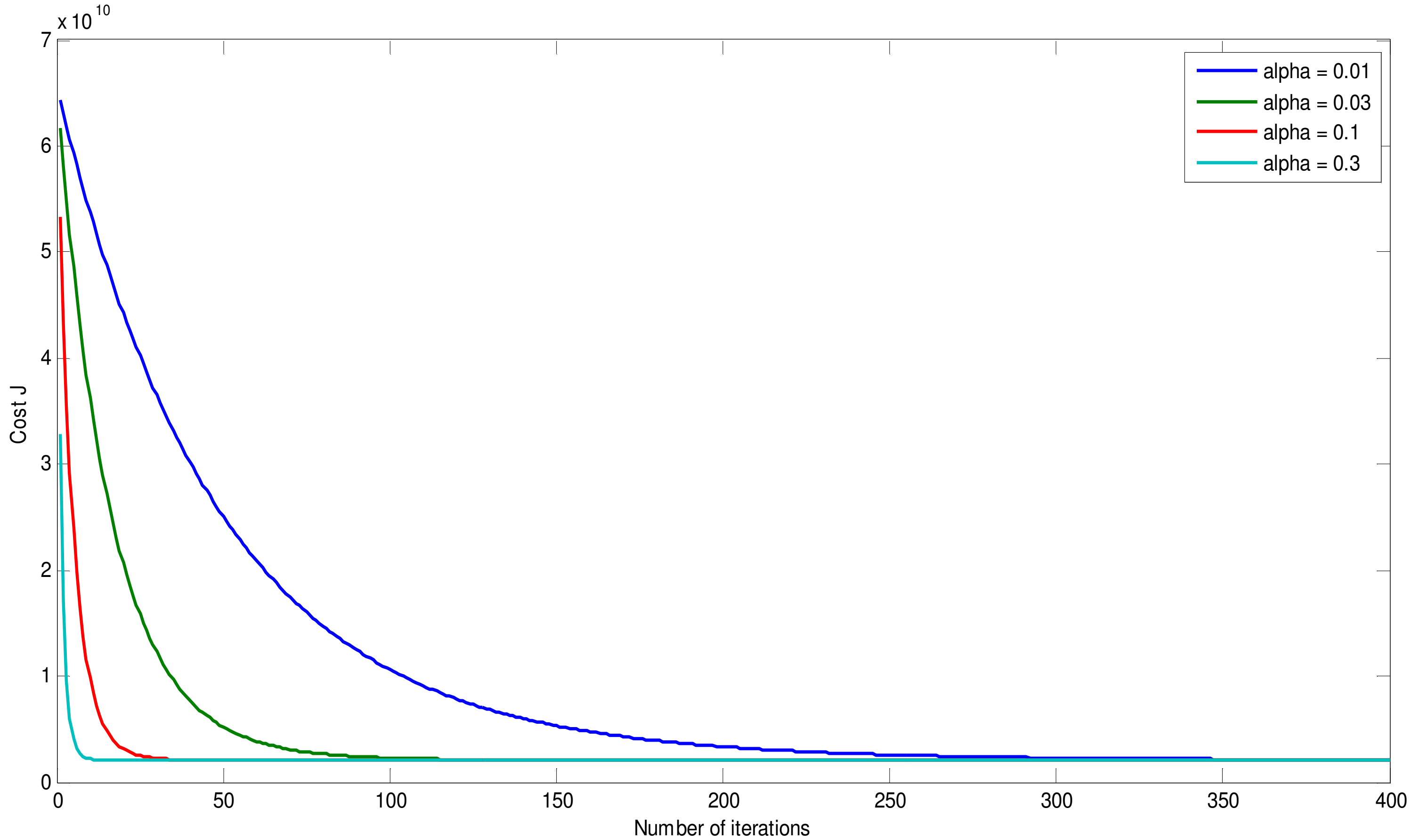


$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

If α too small : slow convergence of the cost function J (the Gradient Descent optimization can be slow)

universidade de aveiro

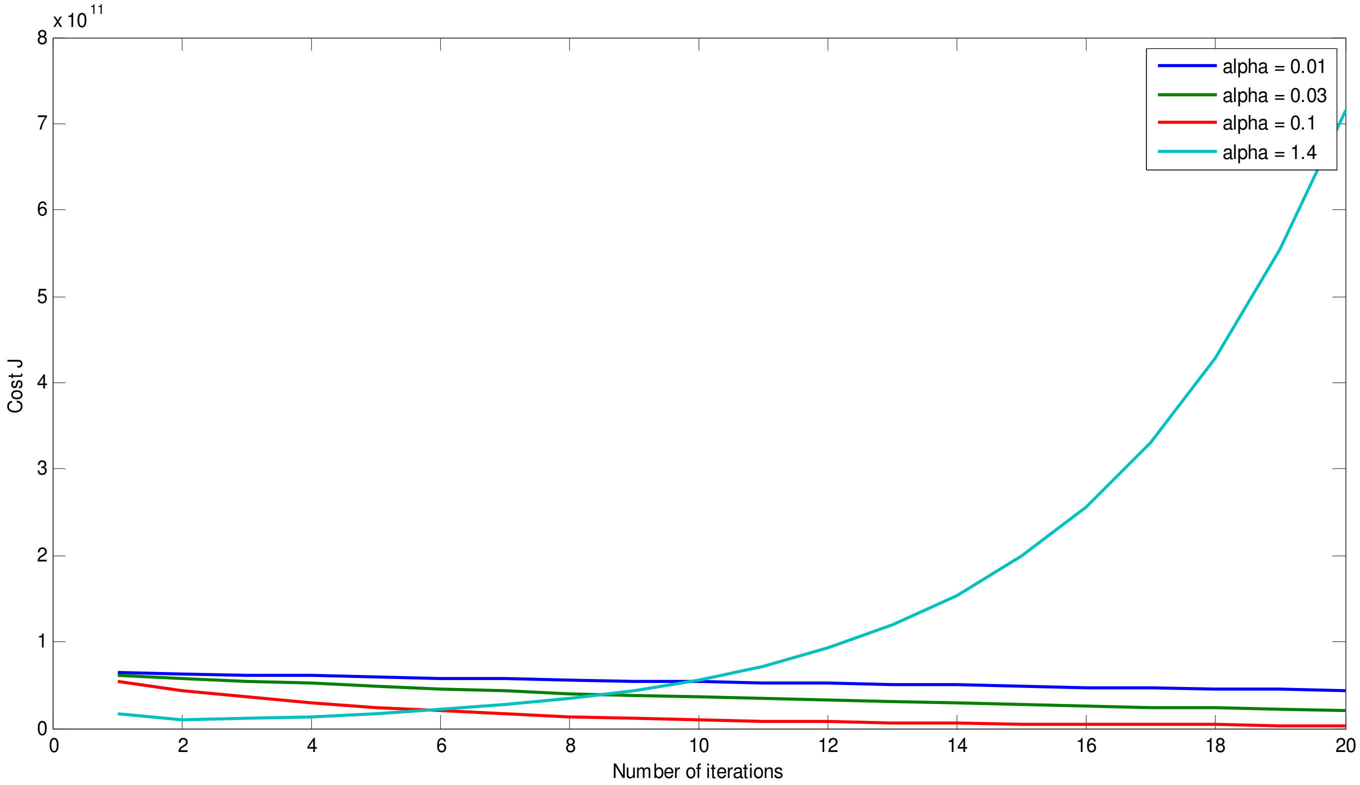
Cost function convergence changing the learning rate (α) -400 iter.



$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

universidade de aveiro

Cost function convergence changing the learning rate (α)

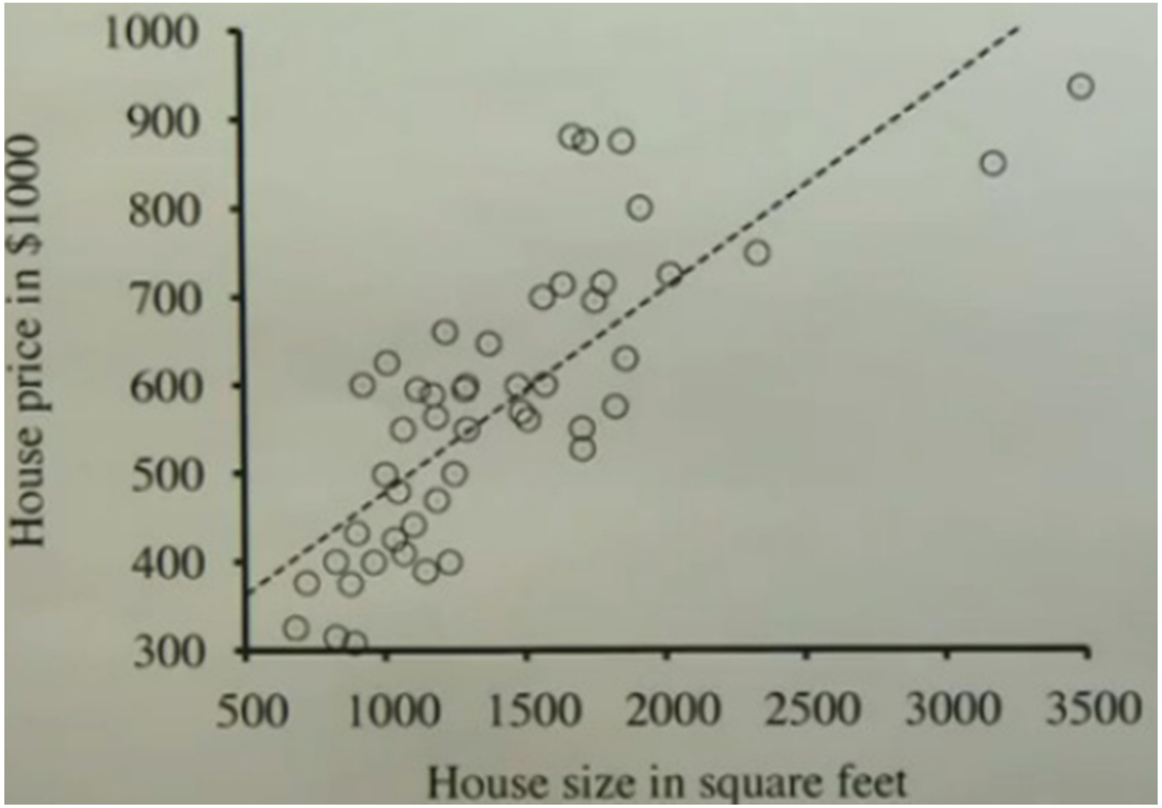


If α too large: the cost function J may no converge **(**decrease at each iteration). It may diverge **!**

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Linear regression model

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$



Given the house area, what is the most likely house price?
If <u>univariate linear regression</u> model is not sufficiently good model, add more features (ex. # bedrooms) => <u>multivariate regression</u>

Supervised Learning – multivariate regression

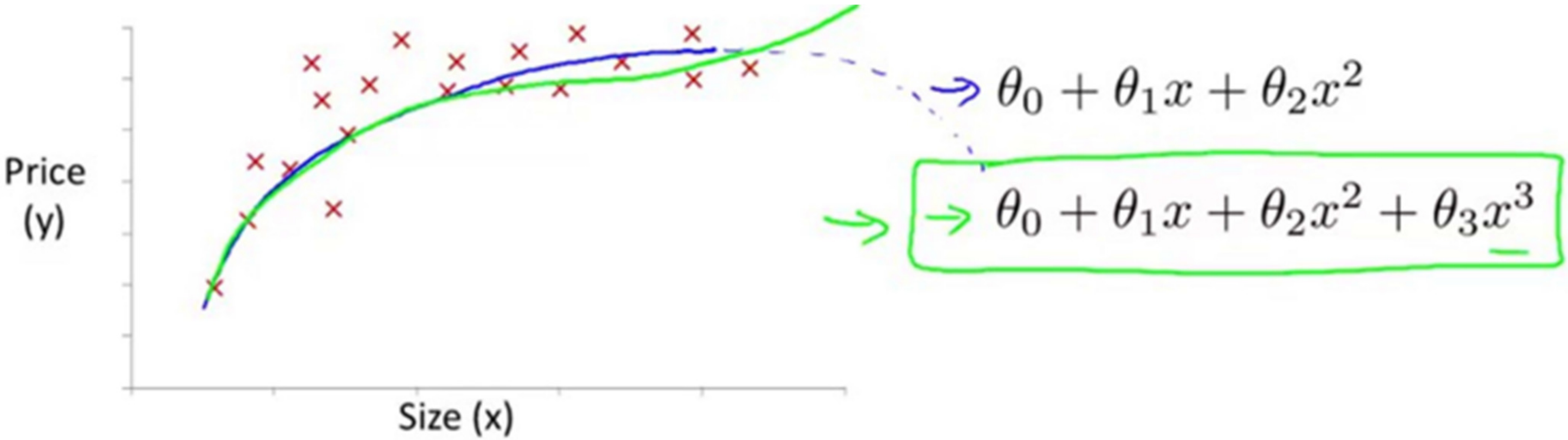
Problem: Learning to predict the housing price as a function of living area & number of bedrooms.

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = [\theta_0 \quad \theta_1 \quad \theta_2] \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \end{bmatrix} = \vec{\theta}^{\,T} \vec{x}$$

Polynomial Regression

If univariate linear regression model is not a good model, try polynomial model.
Univariate (x1=size) housing price problem transformed into multivariate (still linear !!!) regression model x=[x1=size, x2=size^2, x3=size^3]

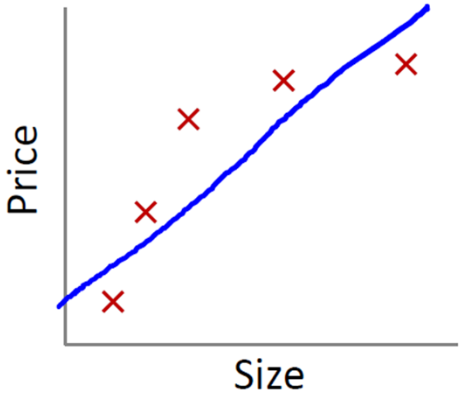


$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

Overfitting problem

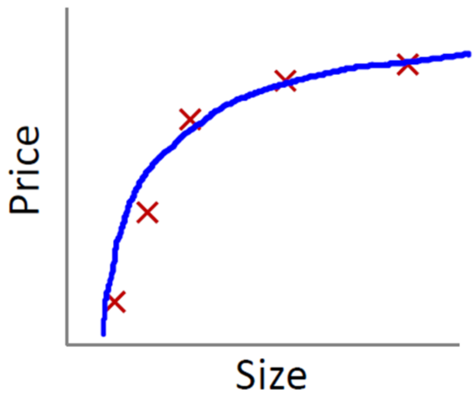
Overfitting: If we have too many features (e.g. high order polynomial model), the learned hypothesis may fit the training set very well but fail to generalize to new examples (predict prices on new examples).



underfit

(1st order polin. model)

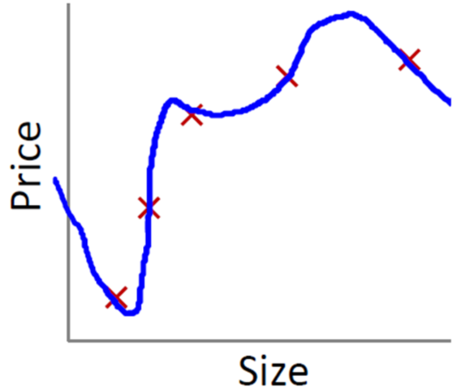
$$h_\theta(x) = \theta_0 + \theta_1 x$$



just right

(3rd order polinom. model)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$



overfit

(higher ord. polinom. Model)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_{16} x^n$$

universidade de aveiro

Overfitting problem

Overfitting: If we have too many features (x1,…x100) the learned model may fit the training data very well but fails to generalize to new examples.

$$
\begin{aligned}
x_1 &= \text{size of house} \\
x_2 &= \text{no. of bedrooms} \\
x_3 &= \text{no. of floors} \\
x_4 &= \text{age of house} \\
x_5 &= \text{average income in neighborhood} \\
x_6 &= \text{kitchen size} \\
&\vdots \\
x_{100}
\end{aligned}
$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n = \vec{\theta}^T \vec{x}$$

How to deal with overfitting problem ?

1. Reduce number of features.
— Manually select which features to keep.
— Algorithm to select the best model complexity.

2. Regularization (add extra term in cost function)
Regularization methods shrink model parameters θ towards zero to prevent overfitting by reducing the variance of the model.

2.1 Ridge Regression L2 Norm

— Reduce magnitude of θ (but never make them =0) => keep all features
— Works well when all features contributes a bit to the output y.

2.2 Lasso Regression L1 MNorm

- May shrink some of the elements of vector θ to become 0.
- Eliminate some of the features => Serve as feature selection

Regularized Linear Regression (cost function)

Unregularized cost function =>

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Regularized cost function
(add extra regularization term
don't regularize θ_0

Ridge Regression

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

segundo hyperpatrameter

$$\min_\theta J(\theta)$$

vai dizer para nao escolher muitos grandes paramtetreos. thetas

universidade de aveiro

Regularized Linear Regression (cost function gradient)

Unregularized cost function gradients =>

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Regularized cost function gradients =>

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

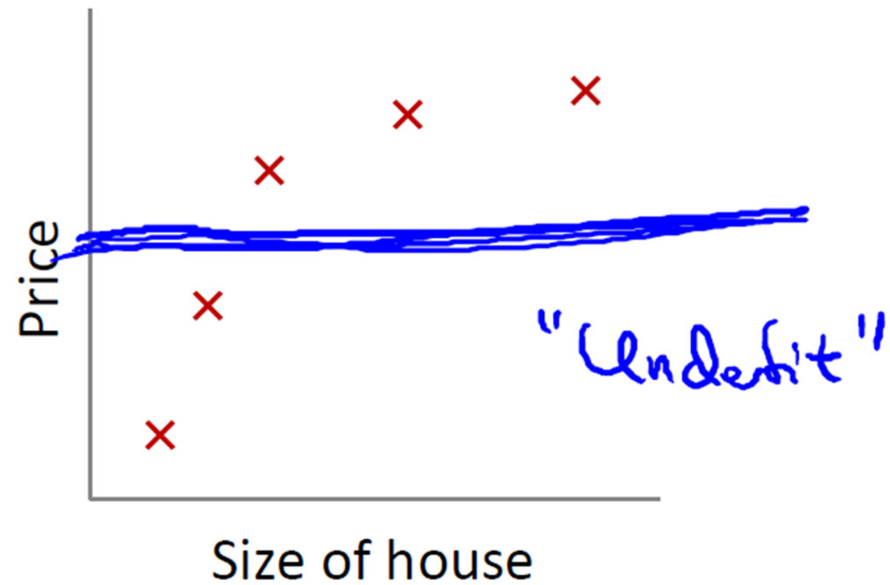
$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

universidade de aveiro

Regularized Linear Regression

What if lambda is set to an extremely large value ?

- Algorithm fails to eliminate overfitting.
- Algorithm results in under-fitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.



universidade de aveiro

Regularization: Lasso Regression

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\left|\theta_j\right|$$

Ridge Regression shrinks θ towards zero, but never equal to zero => all features are included in the model no matter how small are the coefficients.

Lasso Regression is able to shrink coefficients to exactly zero => reduces the number of features. This makes Lasso Regression useful in cases with high dimension.

Lasso Regression involves absolute values (not differentiable)=> computing is difficult => relevant algorithms available in sklearn Python library.