

# Regresión simple, múltiple, y clasificación (1)

Mathieu Kessler

Departamento de Matemática Aplicada y Estadística  
Universidad Politécnica de Cartagena

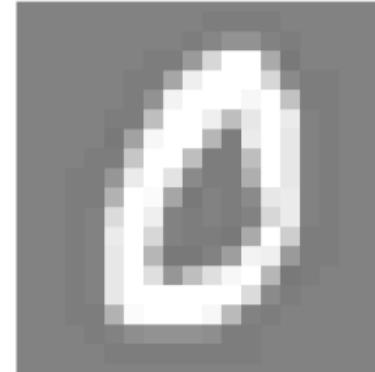
Cartagena

## Conjuntos de datos reales

- A menudo, un individuo está descrito por una serie de características (en inglés “features”), o variables.
- En base a los valores de estas características, podemos estar interesados en:
  - ① Clasificar cada individuo en una determinada categoría.
  - ② Predecir el valor de una cantidad de interés y para un individuo.

## Ejemplo: problema de clasificación

- Queremos reconocer de manera automática los códigos postales escritos a mano en sobres
- Aislamos imágenes 20x20 píxeles de cada dígito en el código postal:



- Las características son  $x_1, x_2, \dots, x_{400}$ , la intensidad de gris en cada pixel.
- Basándonos en esas características, queremos clasificar el dígito en una de las categorías "0", "1", "2", ..., "9".

## Ejemplo: problema de predicción

- Queremos predecir la demanda de energía eléctrica de un conjunto de clientes con un día de antelación.
- Las características que consideramos pueden incluir: temperatura ambiente, ¿es día festivo?, contexto económico, ¿hay huelga?
- Basándonos en esas características para el día siguiente, queremos obtener una predicción de  $y$ , la demanda eléctrica para mañana.

Tenemos un conjunto de datos del que podemos aprender:  
Para un cierto número de individuos,

- Por una parte, hemos observado sus características

$$x_1, x_2, \dots, x_k.$$

- Pero también hemos observado:
  - para el problema de clasificación, a qué categoría pertenece. Se recoge en una variable y que toma valores discretos (las categorías).
  - para el problema de predicción, el valor que toma la variable de interés y para cada uno de esos individuos.

“Learning set”

- Usaremos un algoritmo (“learning algorithm”), que aprenda del conjunto de aprendizaje, de manera que, para un nuevo individuo, con características  $x_1, \dots, x_k$  (que no tengan por qué corresponder exactamente con valores ya observados en el conjunto de aprendizaje), nos proporcione una predicción del valor de  $y$ .

Empezamos por ilustrar los conceptos con el problema de predicción

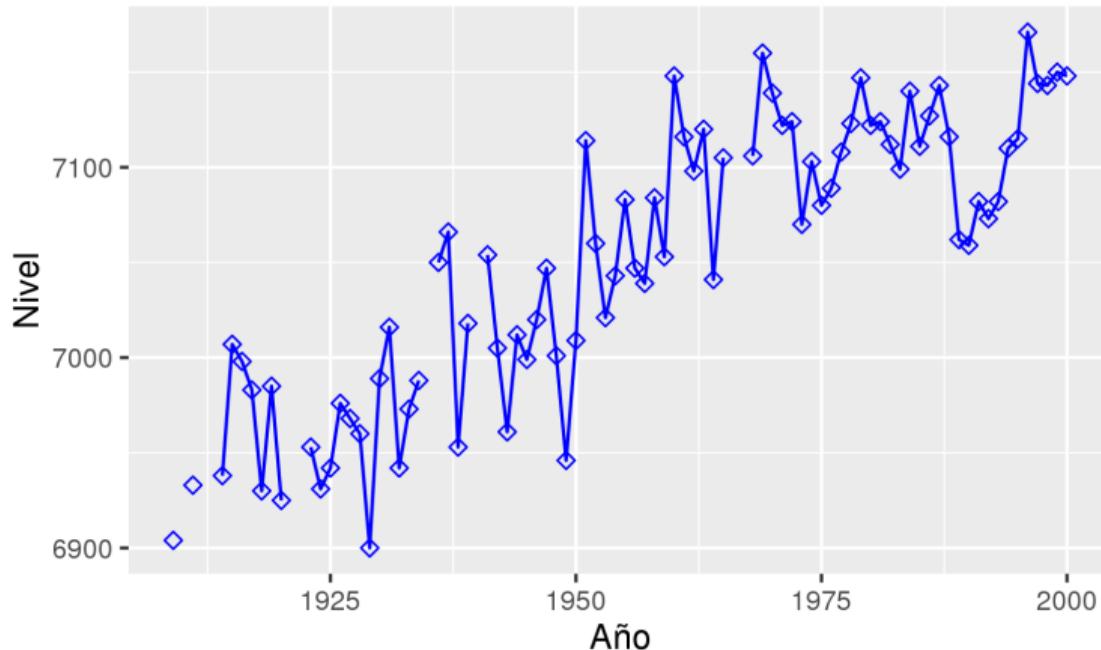
Consideraremos

- Una variable  $y$  (también llamada **respuesta**)
- $k$  variables o **características**  $x_1, x_2, \dots, x_k$ .

Buscamos aprender para predecir el valor de  $y$  en función del valor observado de  $x_1, x_2, \dots, x_k$ .

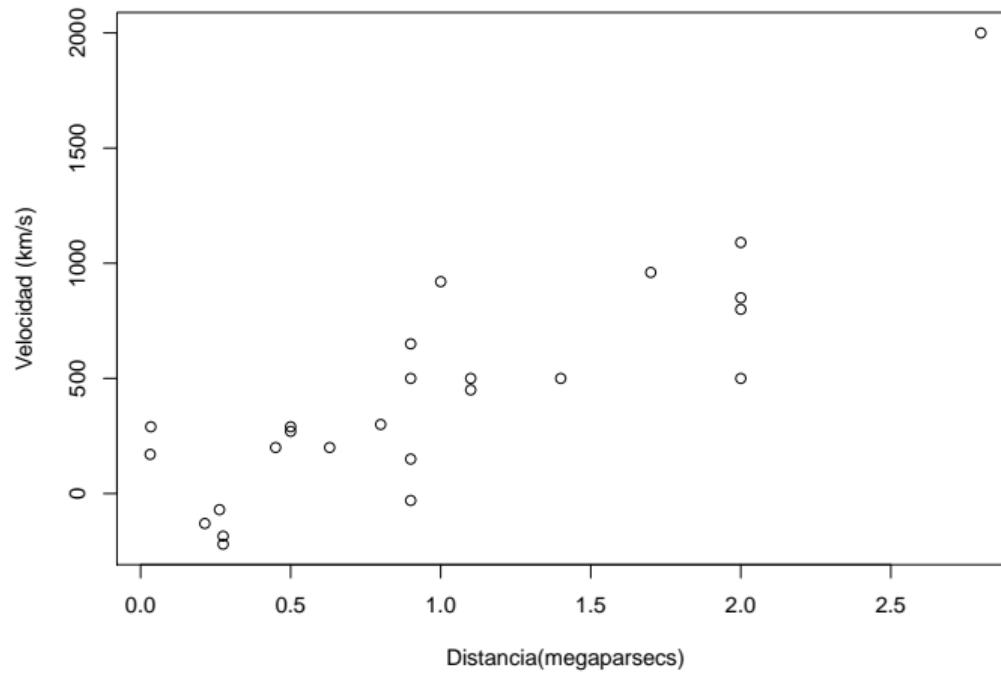
# Ejemplos de conjuntos de datos reales

Nivel máximo anual del mar en Venecia



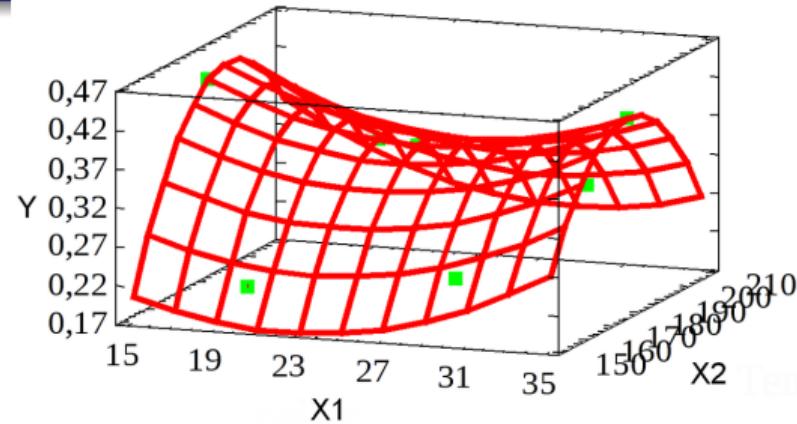
# Ejemplos de conjuntos de datos reales

Velocidad de Recesión de 24 nebulosas



# Ejemplos de conjuntos de datos reales

## Superficie de respuesta

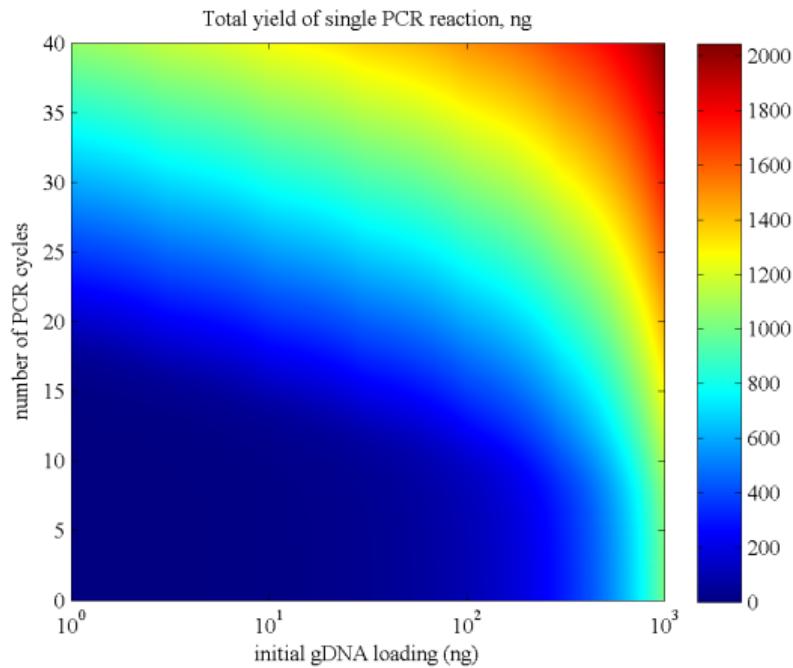


$$\begin{aligned}Y = & -5,53141 \\& -0,00740821 * X_1 \\& +0,0653744 * X_2 \\& +0,0005965 * X_1^2 \\& -0,000128269 * X_1 * X_2 \\& -0,000169138 * X_2^2\end{aligned}$$

Fuente: [https://commons.wikimedia.org/wiki/File:Estimated\\_response\\_surface3.png](https://commons.wikimedia.org/wiki/File:Estimated_response_surface3.png) Attribution: Cjp24, CC BY-SA 3.0, via Wikimedia Commons.

# Ejemplos de conjuntos de datos reales

## Rendimiento de una reacción



Fuente: [http://www.wright.edu/~oleg.paliy/Papers/PCR\\_modeling/FigureS5.png](http://www.wright.edu/~oleg.paliy/Papers/PCR_modeling/FigureS5.png)

## Los datos

Nuestro conjunto de aprendizaje:

$Y$	$X_1$	$X_2$	$\cdots$	$X_k$
$y_1$	$x_{11}$	$x_{12}$	$\cdots$	$x_{1k}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$y_n$	$x_{n1}$	$x_{n2}$	$\cdots$	$x_{nk}$

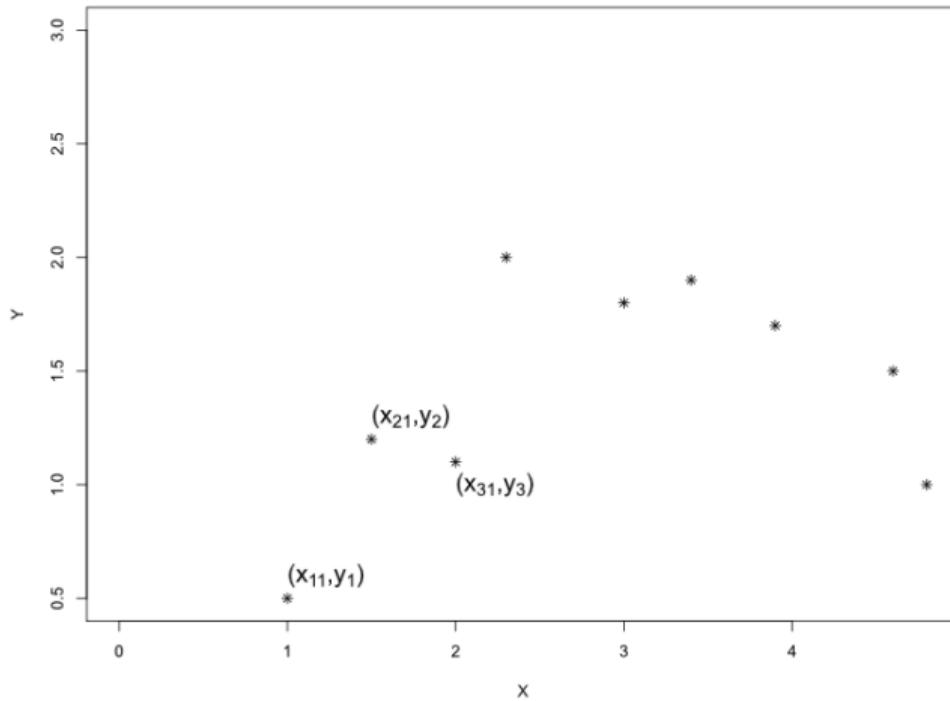
Cada fila representa un individuo, cada columna una variable o característica para ese individuo.

Usaremos la notación

$$x_{i\bullet} = (x_{i0}, x_{i1}, \dots, x_{ik})^T$$

para denotar el vector de características del individuo número  $i$  (*hemos incluido  $x_{i0} = 1.$* )

# Nuestro conjunto de aprendizaje, con sólo una característica $x$ .



Decidimos explicar la relación entre  $x_1, x_2, \dots, x_k$  e  $y$  usando una determinada forma funcional

- Por ejemplo, con una característica, una recta:  $y = ax_1 + b$ .
- Por ejemplo, con dos características, un parabolóide:  
$$y = a_{00} + a_{10}x_1 + a_{01}x_2 + a_{20}x_1^2 + a_{02}x_2^2 + a_{11}x_1x_2.$$
- En general, especificamos una familia paramétrica:

$$y = f(\theta, x_0, x_1, \dots, x_k) \quad \theta = (\theta_0, \theta_1, \dots, \theta_k),$$

$\theta$  es un vector de parámetros.

A veces llamamos a la relación  $y = f(\theta, x_0, x_1, \dots, x_k)$  la “hipótesis”.

Nuestro objetivo: aprender del conjunto de aprendizaje,

es decir determinar la función de la familia que proporcione la **mejor** adecuación al conjunto de aprendizaje  $\Leftrightarrow$  Debemos aprender el valor concreto de  $\theta$  que corresponde a esa función óptima.

# Nuestro concepto de “mejor”: la función coste

Introducimos una función de coste

- Asocia para un valor concreto de  $\theta$  un valor numérico que refleje la calidad del ajuste de nuestra funcional o hipótesis al conjunto de aprendizaje.
- Es una función que denotamos por

$$\theta \mapsto J(\theta).$$

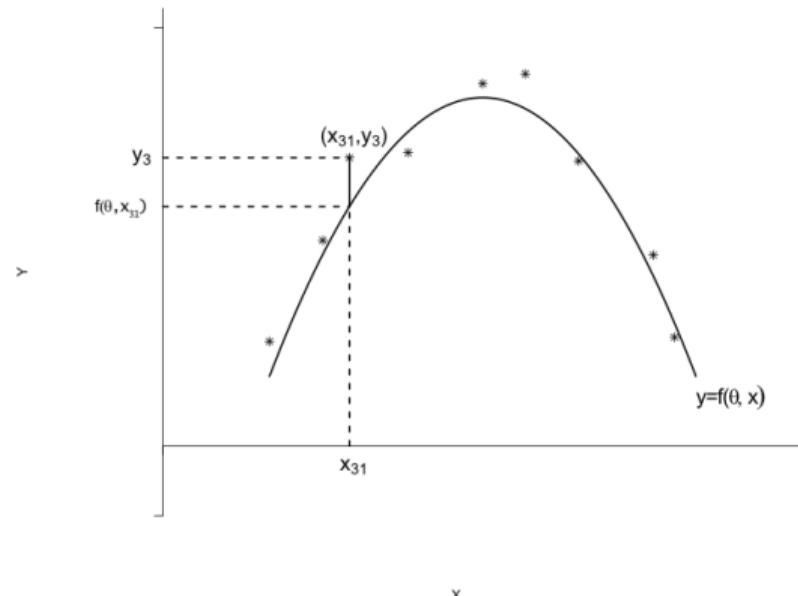
Nuestro objetivo será:

Encontrar el valor de  $\theta$  que minimiza el coste, es decir tendremos que resolver el problema de minimización:

$$\min_{\theta} J(\theta).$$

# La función de coste que consideraremos

Buscamos que sean mínimas las distancias entre los valores de  $y$  observados en el conjunto de aprendizaje, es decir  $y_1, y_2, \dots, y_n$ , y sus valores predichos por la hipótesis:  $f(\theta, x_{10}, x_{11}, \dots, x_{1k}), f(\theta, x_{20}, x_{21}, \dots, x_{2k}), \dots, f(\theta, x_{n0}, x_{n1}, \dots, x_{nk})$ .



# La función de coste

Definimos la función de coste:

$$J(\theta) = \frac{1}{n} [(y_1 - f(\theta, x_{10}, \dots, x_{1k}))^2 + \dots + (y_n - f(\theta, x_{n0}, \dots, x_{nk}))^2].$$

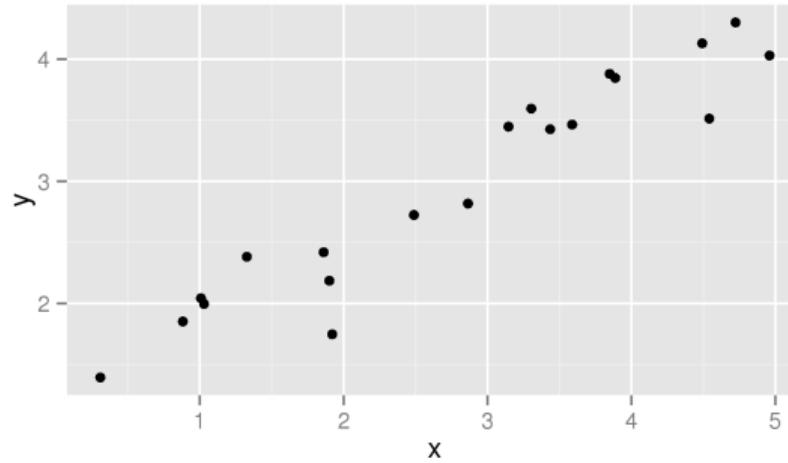
Buscamos el valor  $\hat{\theta}$  de  $\theta$  que minimiza  $\theta \mapsto J(\theta)$ .

## Nota

- En  $J(\theta)$ ,  $x_{10}, \dots, x_{nk}$  e  $y_1, \dots, y_n$  son valores fijados concretos (conjunto de aprendizaje). Sólo  $\theta$  es variable:  $\theta \mapsto J(\theta)$  es una función de  $\theta$ .
- $\theta$  es un vector.

# Un primer ejemplo

Nuestro conjunto de aprendizaje:



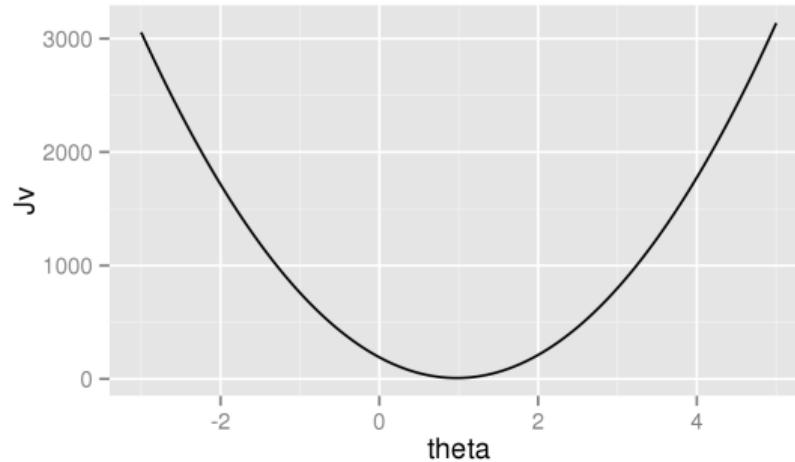
Nuestra hipótesis:

$$f(\theta, x_1) = \theta_1 x_1,$$

es decir, una recta forzada por el origen.

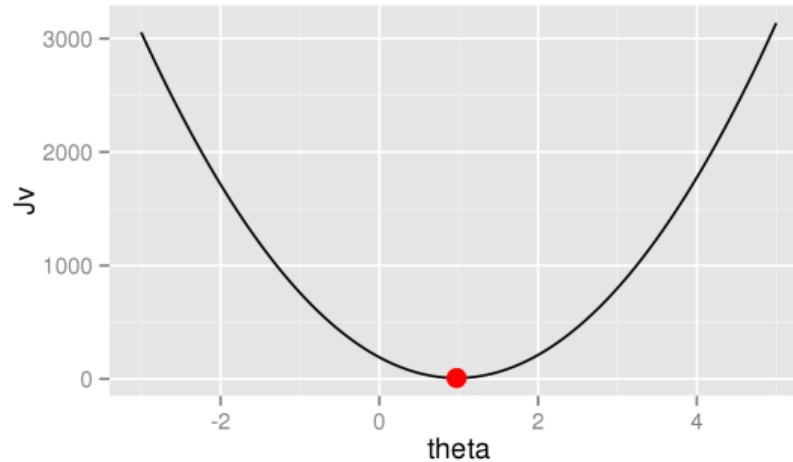
# La función de coste

Tenemos un único parámetro  $\theta = \theta_1$ , podemos representar la función de coste.



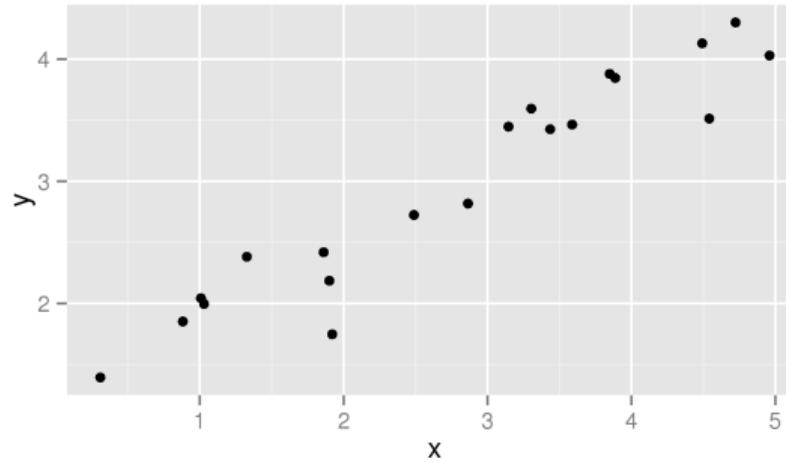
# La función de coste

Tenemos un único parámetro  $\theta = \theta_1$ , podemos representar la función de coste.  
Con su mínimo:



# Un segundo ejemplo

Mismo conjunto de aprendizaje:



Nuestra hipótesis:

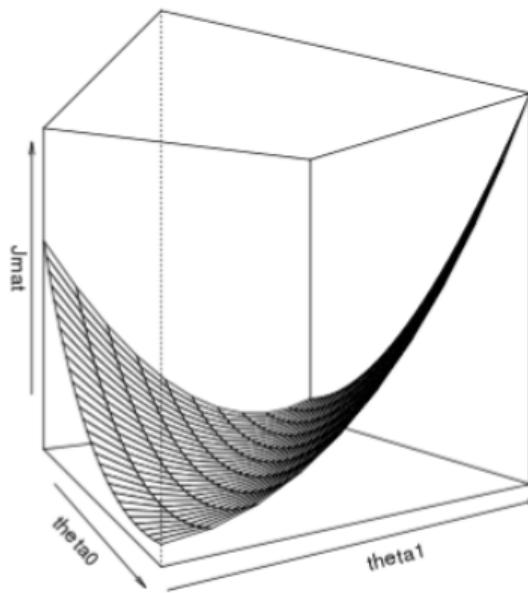
$$f(\theta, x_1) = \theta_0 + \theta_1 x_1,$$

es decir, ahora admitimos una ordenada al origen

# La función de coste

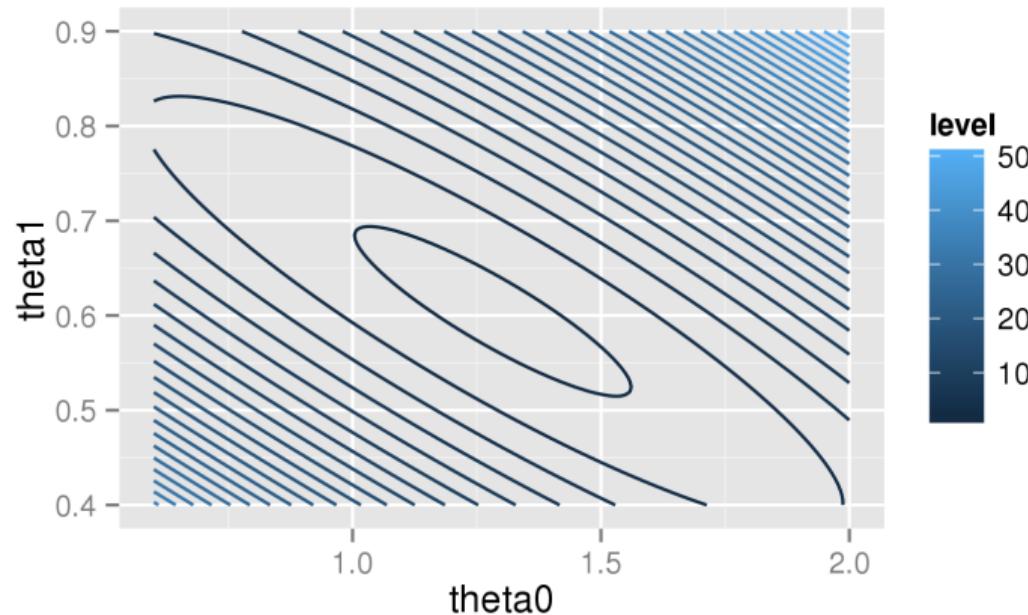
Tenemos dos parámetros: representamos la función de coste como una superficie:

Función de coste con dos parámetros



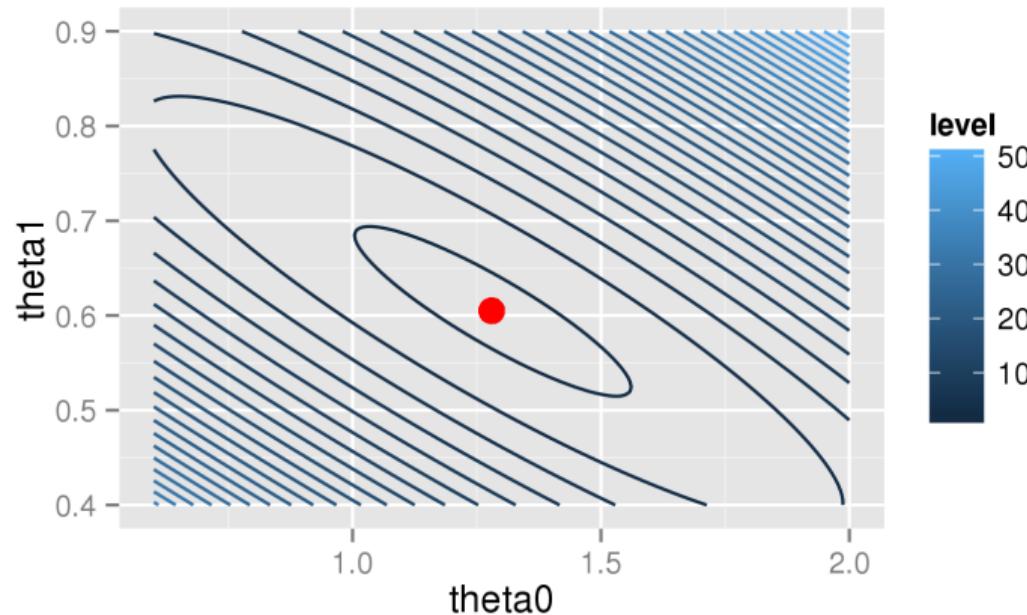
# La función de coste

O usando curvas de nivel:



# La función de coste

Situamos el mínimo:



# Procedimiento numérico de minimización: el algoritmo del gradiente (“gradient descent”)

## Idea básica

- Queremos minimizar una función  $(\theta_0, \dots, \theta_k) \mapsto J(\theta_0, \dots, \theta_k)$ , es decir encontrar  $\theta$  situado en el mínimo de la superficie (*en un espacio de  $k + 2$  dimensiones*)
- El procedimiento consistirá en:

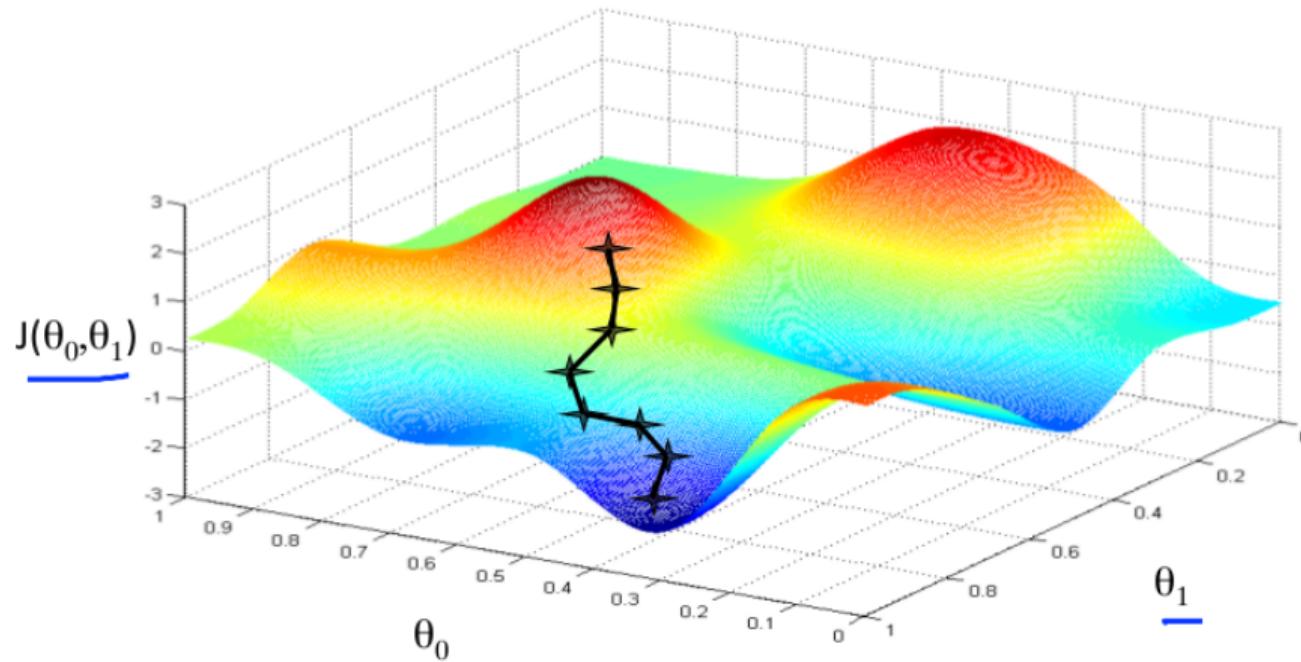
**Iniciar:** empezamos con un vector  $(\theta_0, \dots, \theta_k)$  inicial.

**Iterar:** modificamos el vector  $(\theta_0, \dots, \theta_k)$  para disminuir el valor del coste.

**Parar:** decidimos de un cierto criterio de parada.

# “Gradient descent”

(Imagen de Andrew Ng, Stanford:)



# Algoritmo de “Gradient descent”.

- Partimos de un vector inicial  $(\theta_0, \theta_1, \dots, \theta_k)$ .
- La actualización de  $\theta = (\theta_0, \theta_1, \dots, \theta_k)$  en cada iteración (“valor actual de  $\theta$ ”) aprovecha el gradiente de la función coste:

$$\left\{ \begin{array}{l} \theta_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta), \\ \theta_1 \leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta), \\ \vdots \quad \vdots \quad \vdots \\ \theta_k \leftarrow \theta_k - \alpha \frac{\partial}{\partial \theta_k} J(\theta), \end{array} \right.$$

donde  $\alpha$  es la tasa de aprendizaje (“learning rate”), un parámetro positivo que fijamos.

## Nota

La actualización en cada iteración se hace de manera simultánea para todos los componentes de  $\theta$ , es decir que el gradiente es evaluado en el mismo vector  $(\theta_0, \theta_1, \dots, \theta_k)$  en todas las líneas.

# El algoritmo del gradiente

La modificación se hace de manera simultánea, es decir,

$$\left\{ \begin{array}{l} \theta_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta), \\ \theta_1 \leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta), \\ \vdots \quad \vdots \quad \vdots \\ \theta_k \leftarrow \theta_k - \alpha \frac{\partial}{\partial \theta_k} J(\theta), \end{array} \right.$$

- En **verde**: el valor de  $\theta$  proveniente de la etapa anterior
- En **rojo**: el valor de  $\theta$  actualizado en la etapa actual.

## Algoritmo de “Gradient descent”.

- Partimos de un vector  $(\theta_0, \theta_1, \dots, \theta_k)$ .
- La actualización de  $(\theta_0, \theta_1, \dots, \theta_k)$  en cada iteración aprovecha el gradiente de la función coste:

$$\left\{ \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_k \end{pmatrix} \leftarrow \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_1 \end{pmatrix} - \alpha \nabla J(\theta), \right.$$

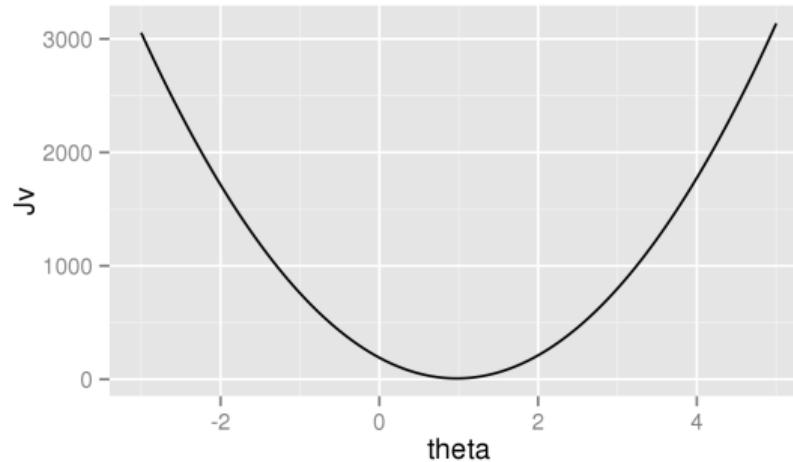
donde  $\alpha$  es la tasa de aprendizaje (“learning rate”), un parámetro positivo que fijamos.

### Nota

La actualización en cada iteración se hace de manera simultánea para  $\theta_0, \dots, \theta_k$ , es decir que el gradiente es evaluado en el mismo vector  $\theta$  en ambas líneas.

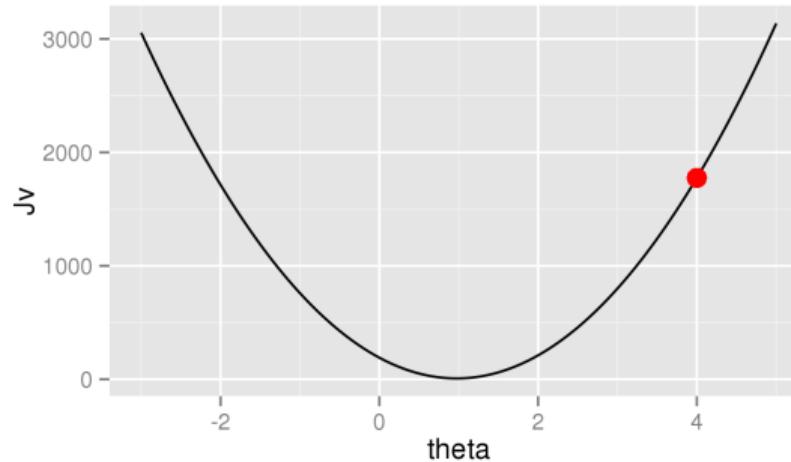
# Intuición para el algoritmo

Es más fácil verlo si tenemos sólo un parámetro  $\theta_1 \mapsto J(\theta_1)$ .



# Intuición para el algoritmo

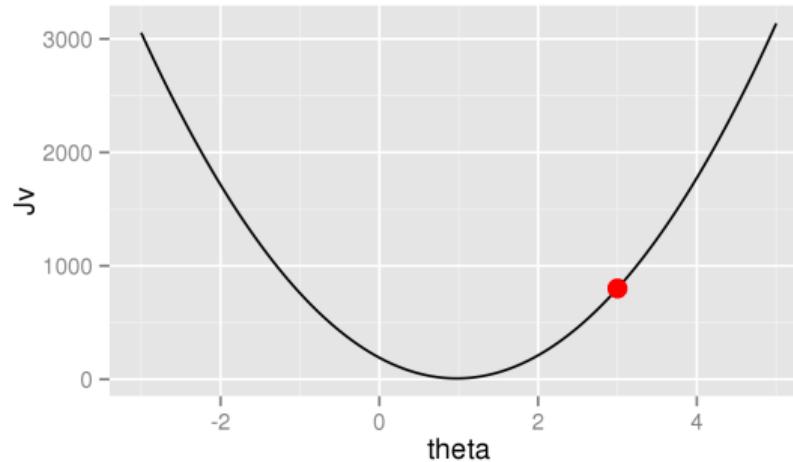
Es más fácil verlo si tenemos sólo un parámetro  $\theta_1 \mapsto J(\theta_1)$ .



Si parto de un punto a la derecha, la pendiente (gradiente) es positiva,

# Intuición para el algoritmo

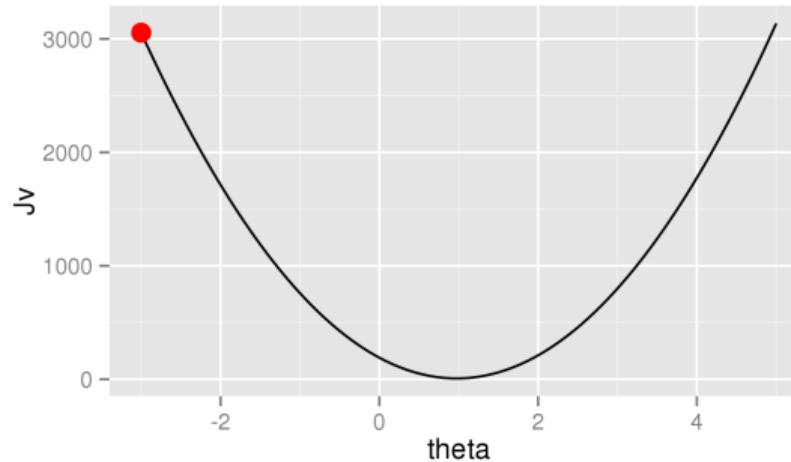
Es más fácil verlo si tenemos sólo un parámetro  $\theta_1 \mapsto J(\theta_1)$ .



Si parto de un punto a la derecha, la pendiente (gradiente) es positiva, por lo que el paso **reduce** el valor de  $\theta_1$ .

# Intuición para el algoritmo

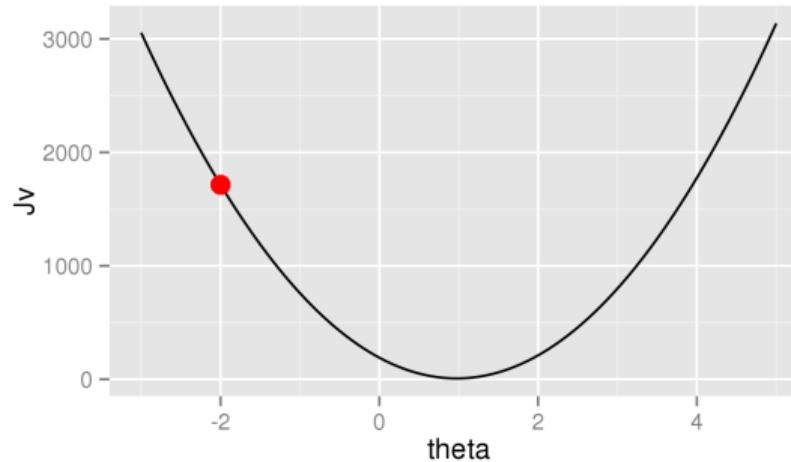
Es más fácil verlo si tenemos sólo un parámetro  $\theta_1 \mapsto J(\theta_1)$ .



Si parto de un punto a la izquierda, la pendiente (gradiente) es negativa,

# Intuición para el algoritmo

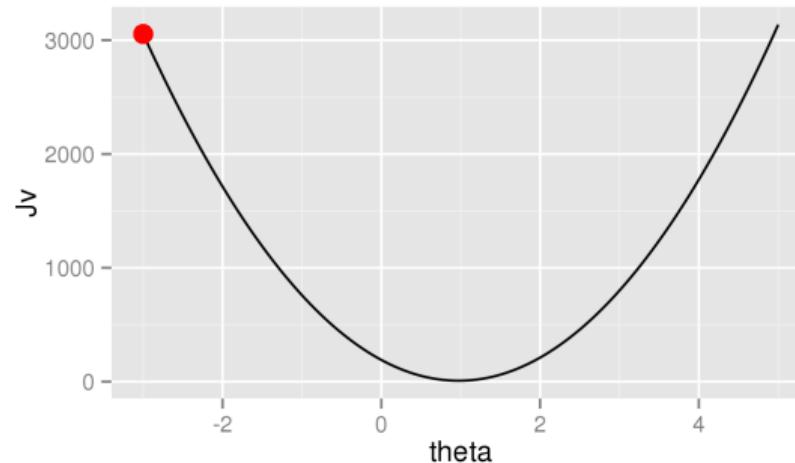
Es más fácil verlo si tenemos sólo un parámetro  $\theta_1 \mapsto J(\theta_1)$ .



Si parto de un punto a la izquierda, la pendiente (gradiente) es negativa, por lo que el paso **aumenta** el valor de  $\theta_1$ .

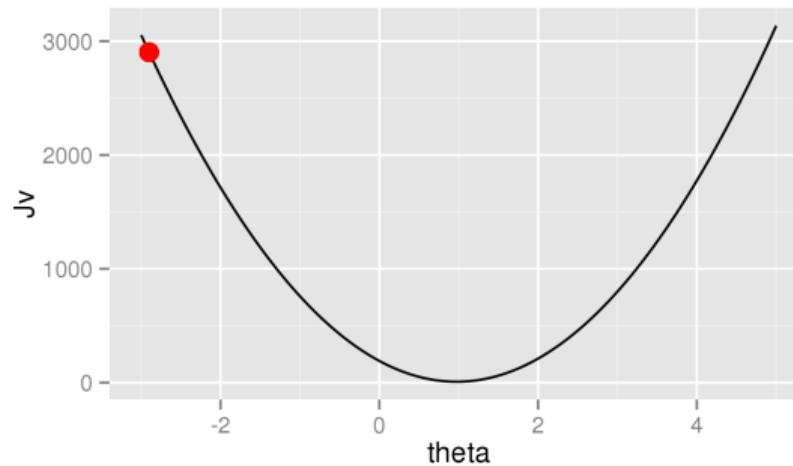
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



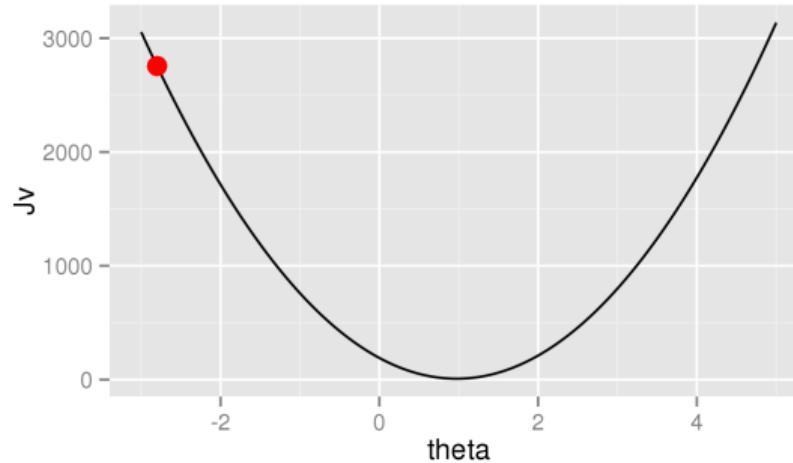
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



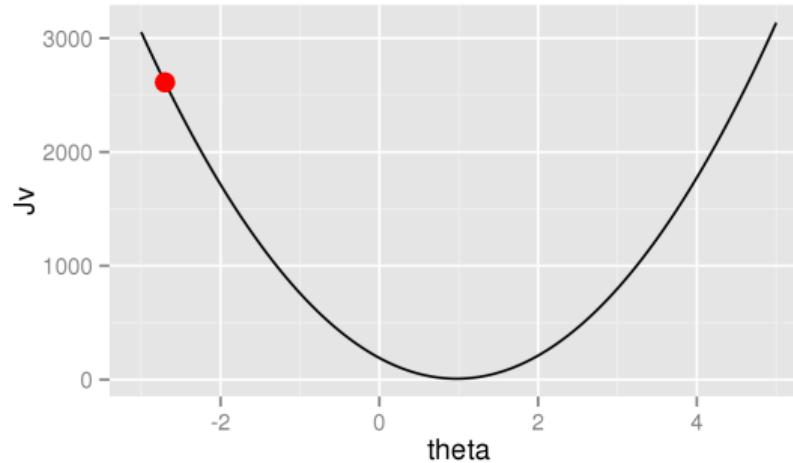
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



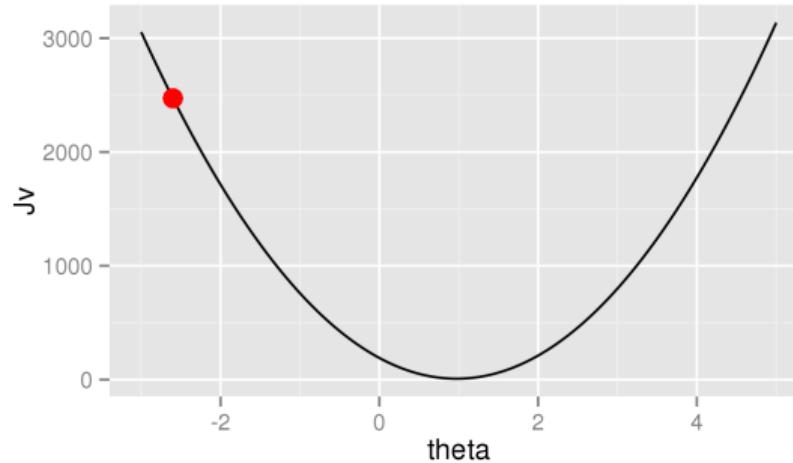
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



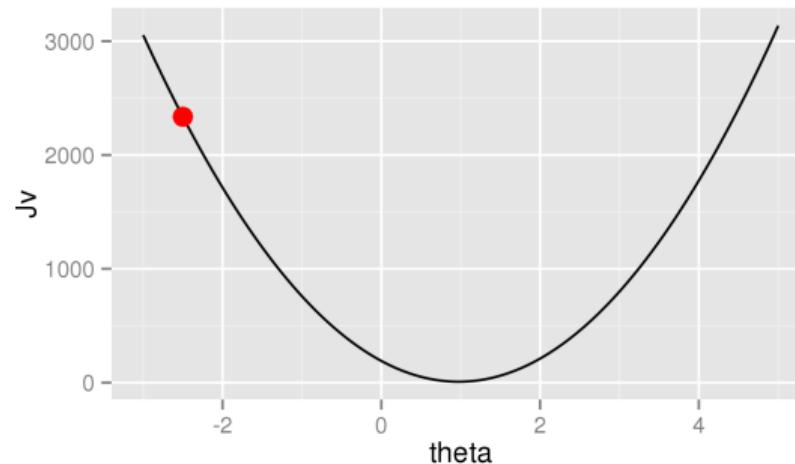
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



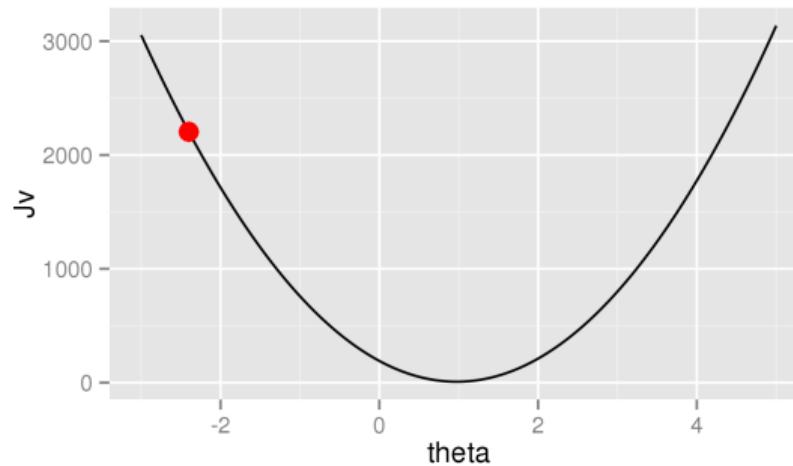
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



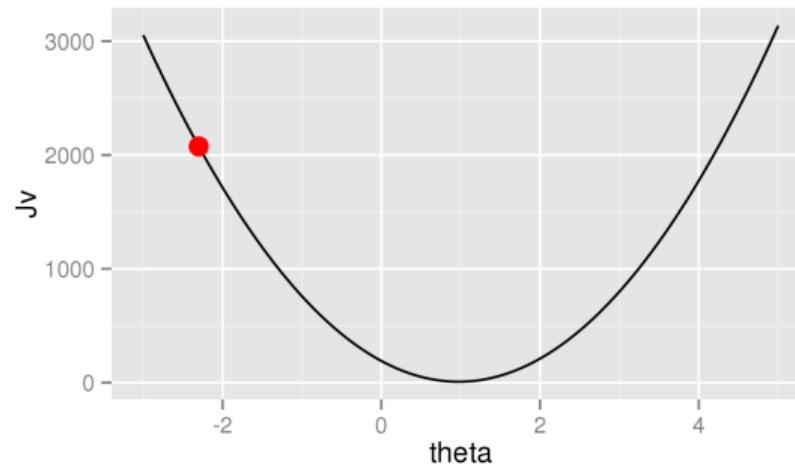
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



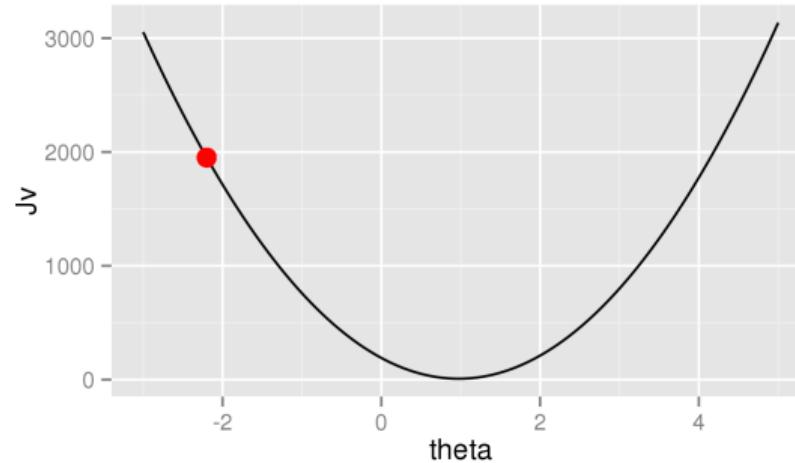
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



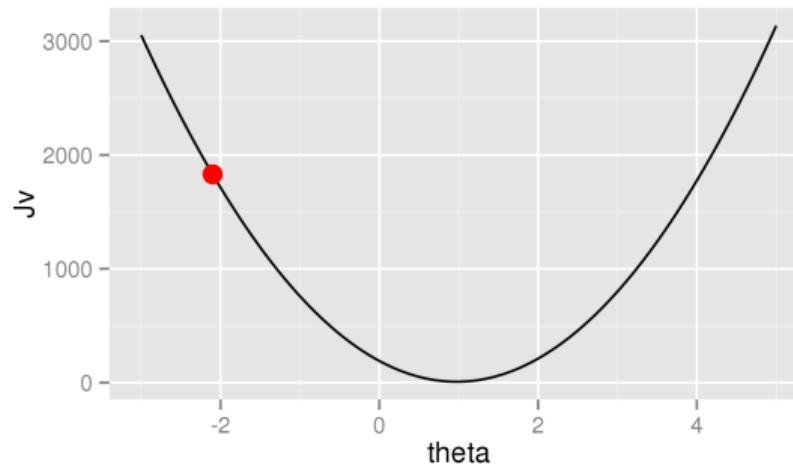
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



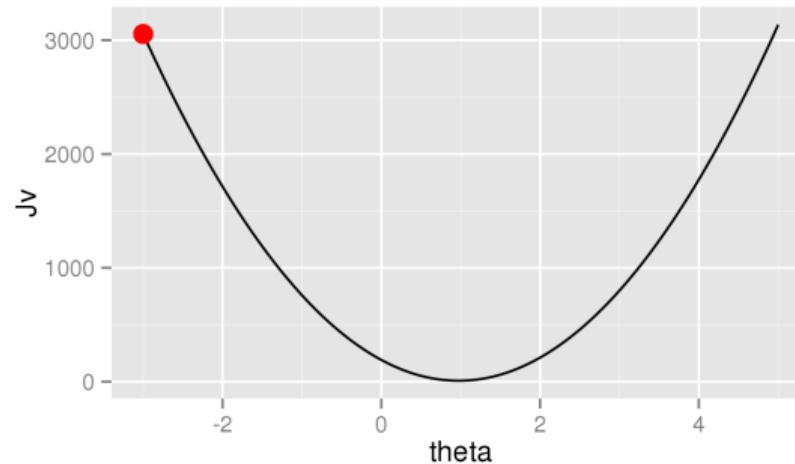
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



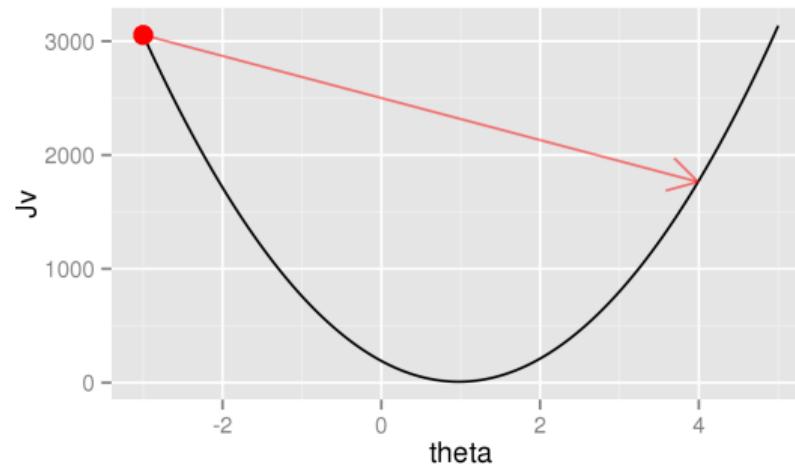
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



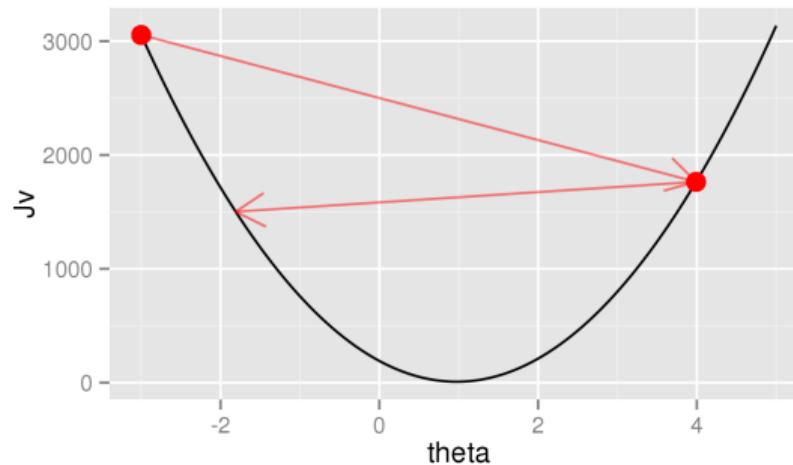
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



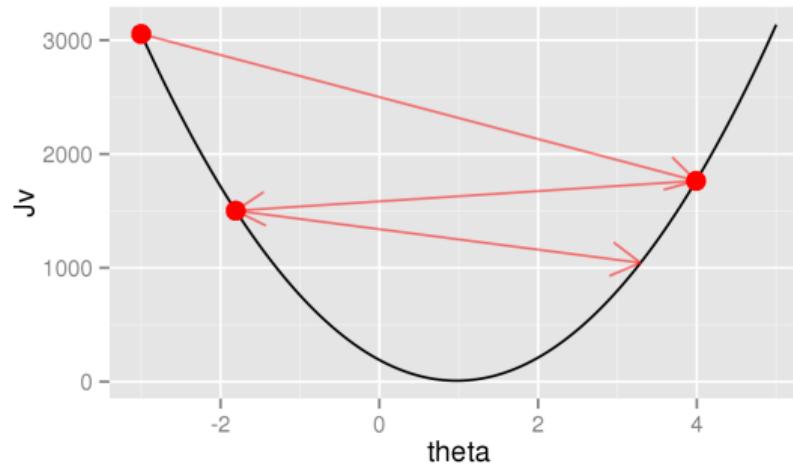
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



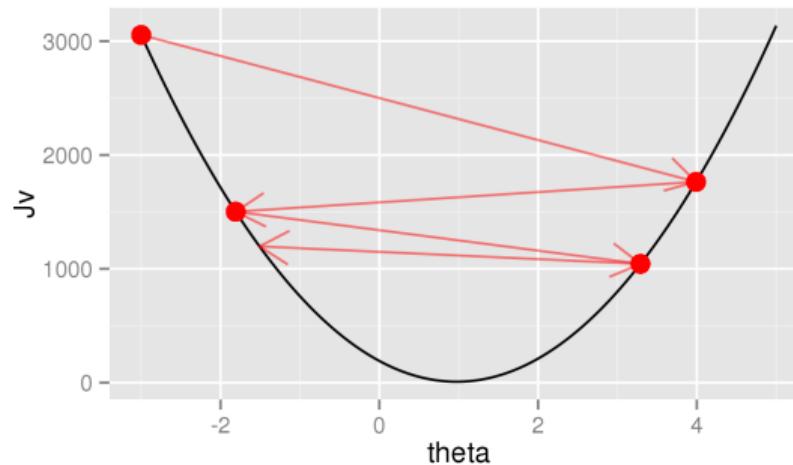
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



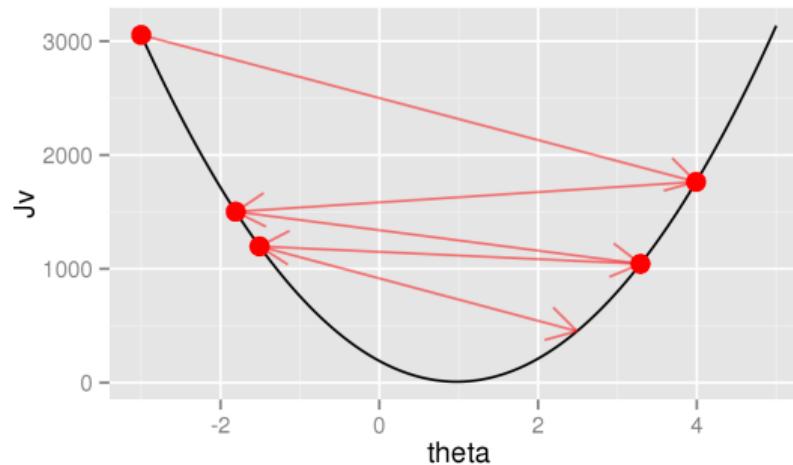
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



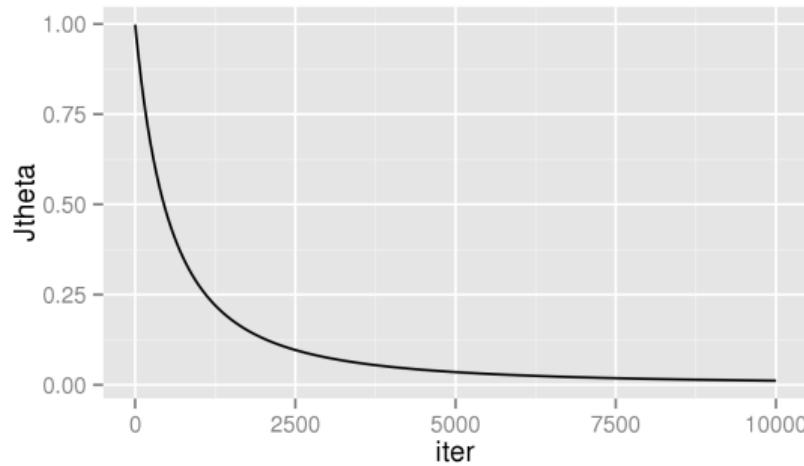
# Intuición para el algoritmo

Es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



# Intuición para el algoritmo

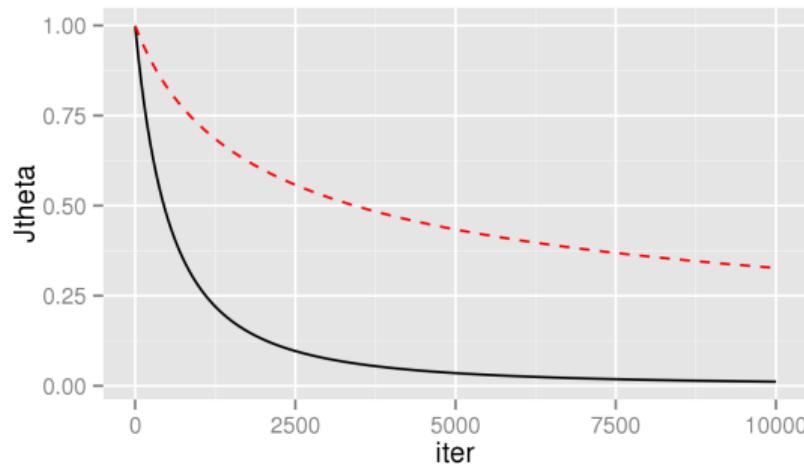
Es importante monitorizar la evolución de la función coste a medida que iteramos el algoritmo de gradiente:



Línea negra: la convergencia del algoritmo es rápida: buena elección de  $\alpha$

# Intuición para el algoritmo

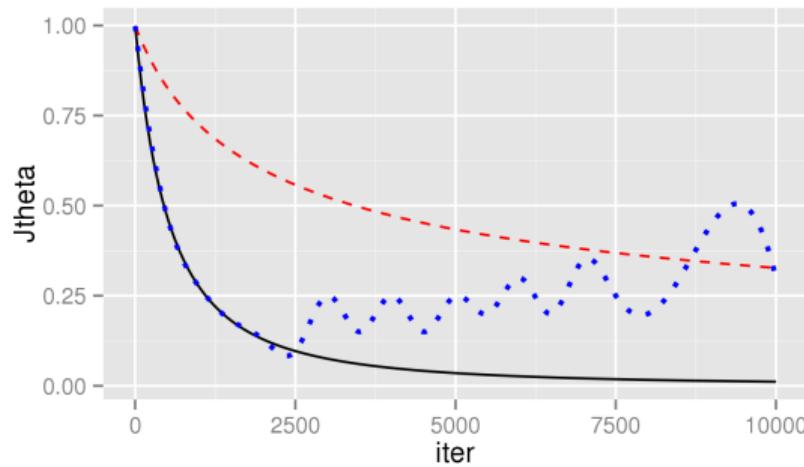
Es importante monitorizar la evolución de la función coste a medida que iteramos el algoritmo de gradiente:



Línea roja: la convergencia es muy lenta:  $\alpha$  demasiado pequeño.

# Intuición para el algoritmo

Es importante monitorizar la evolución de la función coste a medida que iteramos el algoritmo de gradiente:



Línea azul: el algoritmo no converge, quizás  $\alpha$  sea demasiado grande.

# La regresión LINEAL múltiple

- Para la regresión **lineal** múltiple, tenemos  $k$  características  $x_1, x_2, \dots, x_k$  y nuestra hipótesis es

$$y = f(\theta, x_1, x_2, \dots, x_k),$$

con  $\theta = (\theta_0, \theta_1, \dots, \theta_k)$  y

$$f(\theta, x_1, x_2, \dots, x_k) = \theta_0 + \theta_1 x_1 + \dots + \theta_k x_k.$$

- Nota: la regresión lineal simple es por lo tanto un caso particular de la regresión lineal múltiple ( $k=1$ ),  $y = f_\theta(x_1)$ , con  $f_\theta(x_1) = \theta_0 + \theta_1 x_1$

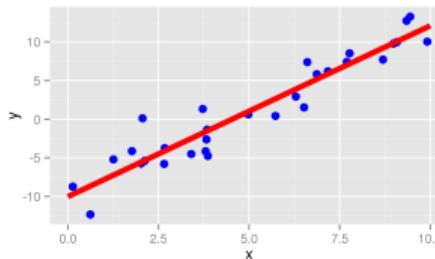
# Regresión múltiple

La regresión lineal múltiple es muy flexible y abarca muchos modelos de hipótesis de dependencia entre  $y$  y las características.

- Una característica  $x_1$ ,

$$y = \theta_0 + \theta_1 x_1$$

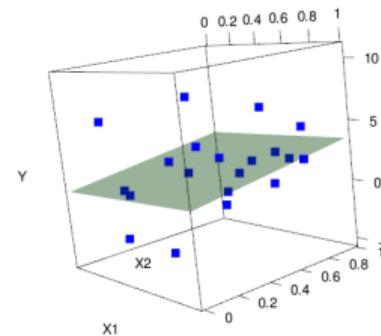
Una recta.



- Dos características  $x_1, x_2$ ,

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Un plano.



# Regresión múltiple

La regresión lineal múltiple es muy flexible y abarca muchos modelos de hipótesis de dependencia entre  $y$  y las características.

- $k$  características  $x_1, \dots, x_k$ ,

$$y = \theta_0 + \theta_1 x_1 + \dots + \theta_k x_k$$



Un hiperplano.

No podemos representarlo si  $k > 2$ ...

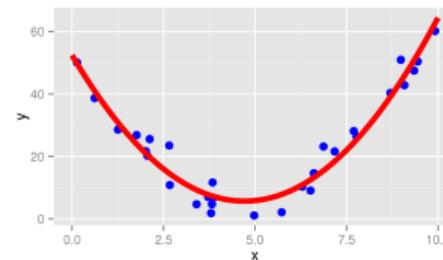
# Regresión múltiple

Podemos incluso considerar dependencias polinomiales, al introducir características ficticias, que sean los cuadrados, cubos etc.. de las características reales:

- Una característica  $x_1$ ,

$$y = \theta_0 + \theta_1 x_1 + \theta_2 (x_1)^2$$

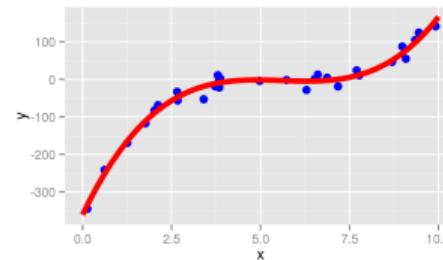
Una parábola. **Introducimos  $x_2 := (x_1)^2$ .**



- Una característica  $x_1$ ,

$$y = \theta_0 + \theta_1 x_1 + \theta_2 (x_1)^2 + \theta_3 (x_1)^3$$

Una cúbica. **Introd.**  $x_2 := (x_1)^2$ ,  $x_3 := (x_1)^3$ .



# Regresión múltiple

Podemos incluso considerar dependencias polinomiales, al introducir características ficticias, que sean los cuadrados, cubos etc.. de las características reales:

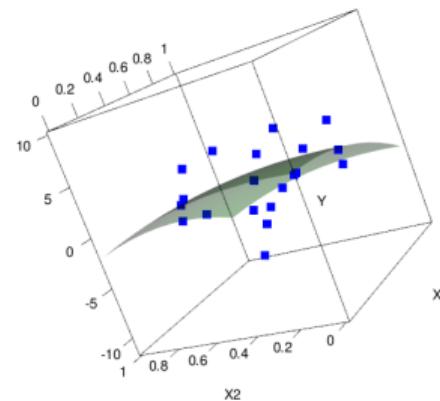
- Dos características  $x_1$  y  $x_2$ ,

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

Un paraboloide.

Introducimos

- $x_3 := (x_1)^2$ ,
- $x_4 := (x_2)^2$ ,
- $x_5 := (x_1 x_2)$



Por lo tanto, la formulación general de nuestra hipótesis será

$$f(\theta, x_1, x_2, \dots, x_k) = \theta_0 + \theta_1 x_1 + \dots + \theta_k x_k.$$

que escribiremos de manera compacta de la manera siguiente:

$$f(\theta, x) = x^T \cdot \theta,$$

donde

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_k \end{pmatrix} \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{pmatrix},$$

y hemos usado la convención  $x_0 = 1$ .

(recordar que  $T$  denota la transpuesta de un vector o matriz).

Los datos que tendremos se presentarán en la forma siguiente:

$Y$	$X_1$	$X_2$	$\cdots$	$X_k$
$y_1$	$x_{11}$	$x_{12}$	$\cdots$	$x_{1k}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$y_n$	$x_{n1}$	$x_{n2}$	$\cdots$	$x_{nk}$

Cada fila representa un individuo, cada columna una variable o característica para ese individuo.

Usaremos la notación

$$x_{i\bullet} = (x_{i0}, x_{i1}, \dots, x_{ik})^T$$

para denotar el vector de características del individuo número  $i$  (*hemos incluido  $x_{i0} = 1$ .*)

# La función de coste

La función de coste es

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\theta, x_{i\bullet}))^2 = \frac{1}{n} \sum_{i=1}^n (y_i - x_{i\bullet}^T \theta)^2$$

Si introducimos

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad \text{y la matriz} \quad \mathbf{X} = \begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1k} \\ x_{20} & x_{21} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n0} & x_{n1} & \cdots & x_{nk} \end{pmatrix}$$

# La función de coste

Tenemos que

$$\begin{pmatrix} y_1 - x_{1\bullet}^T \theta \\ y_2 - x_{2\bullet}^T \theta \\ \vdots \\ y_n - x_{n\bullet}^T \theta \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_k \\ y_n \end{pmatrix} - \begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1k} \\ x_{20} & x_{21} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n0} & x_{n1} & \cdots & x_{nk} \end{pmatrix} \cdot \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{pmatrix}$$
$$= \mathbf{y} - \mathbf{X}\theta.$$

Por lo tanto

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - x_{i\bullet}^T \theta)^2 = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\theta\|^2 = \frac{1}{n} (\mathbf{y} - \mathbf{X}\theta)^T \cdot (\mathbf{y} - \mathbf{X}\theta).$$

# El algoritmo del gradiente para la regresión lineal múltiple

Puesto que

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (\theta_0 x_{i0} + \cdots + \theta_k x_{ik} - y_i)^2,$$

es fácil obtener

$$\left\{ \begin{array}{l} \theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^n 2 (\theta_0 x_{i0} + \cdots + \theta_k x_{ik} - y_i) \cdot x_{i0} \\ \theta_1 \leftarrow \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^n 2 (\theta_0 x_{i0} + \cdots + \theta_k x_{ik} - y_i) \cdot x_{i1}, \\ \vdots \quad \vdots \quad \vdots \\ \theta_k \leftarrow \theta_k - \alpha \frac{1}{n} \sum_{i=1}^n 2 (\theta_0 x_{i0} + \cdots + \theta_k x_{ik} - y_i) \cdot x_{ik}. \end{array} \right.$$

# El algoritmo del gradiente para la regresión lineal múltiple

Puesto que

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (\theta_0 x_{i0} + \dots + \theta_k x_{ik} - y_i)^2,$$

es fácil obtener, en forma compacta:

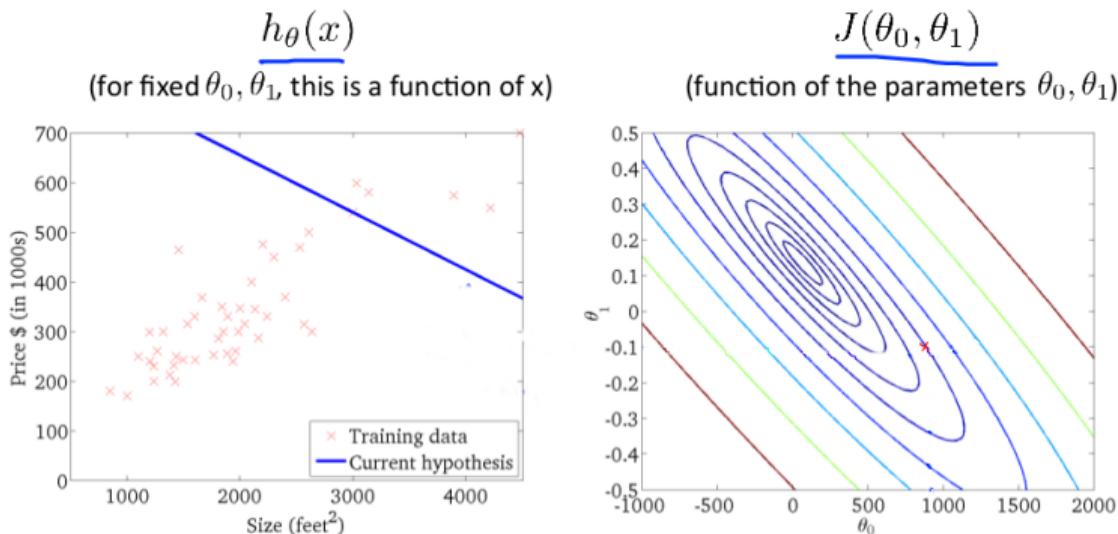
$$\theta \leftarrow \theta - \alpha \frac{2}{n} \mathbf{X}^T \cdot (\mathbf{X}\theta - \mathbf{y}).$$

Es decir, que hemos en realidad usado:

Si  $J(\theta) = \frac{1}{n} (\mathbf{X}\theta - \mathbf{y})^T \cdot (\mathbf{X}\theta - \mathbf{y})$ , entonces

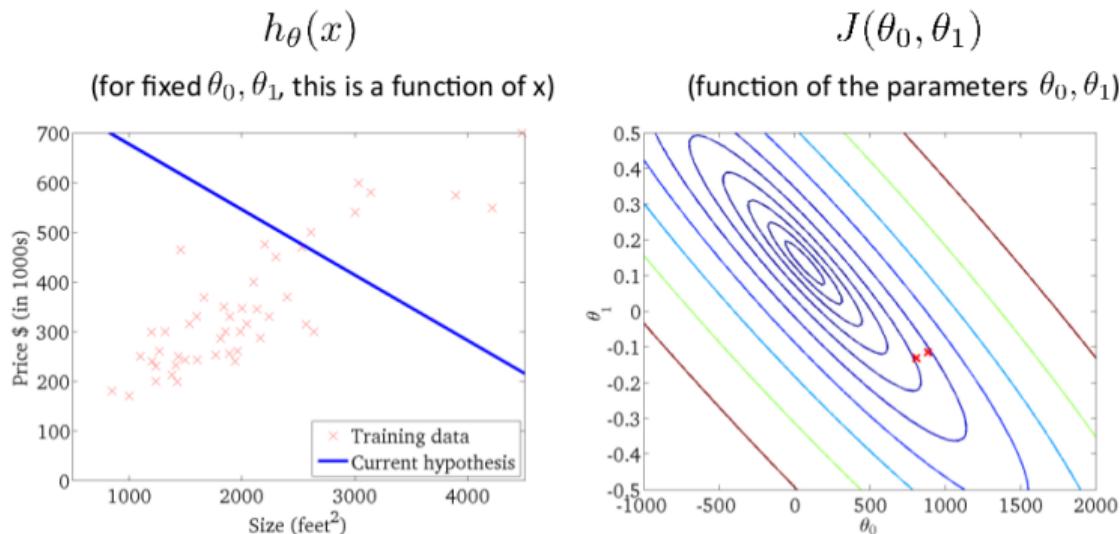
$$\nabla J(\theta) = \frac{2}{n} \mathbf{X}^T \cdot (\mathbf{X}\theta - \mathbf{y}).$$

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)



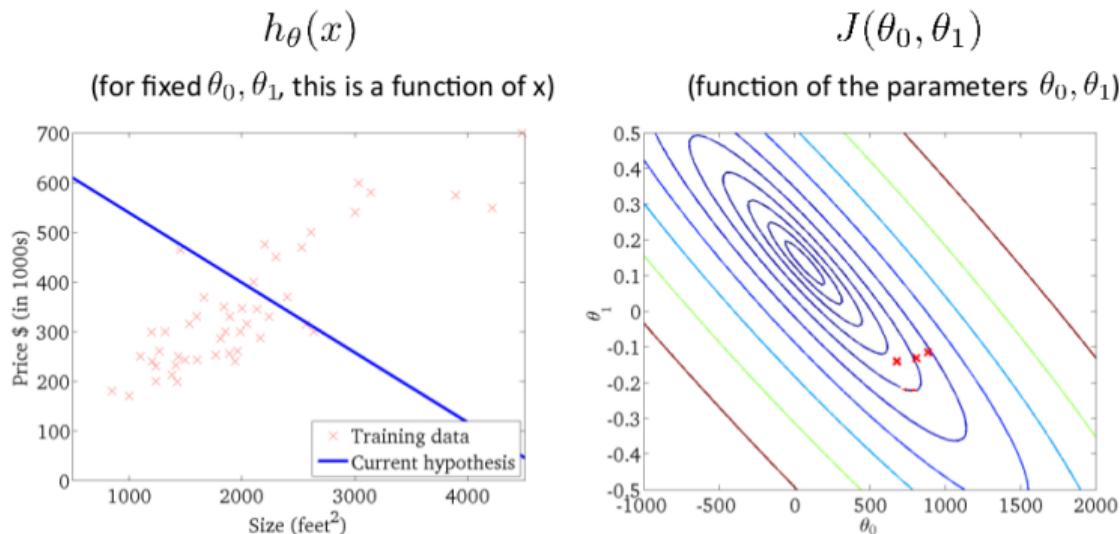
Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)



Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

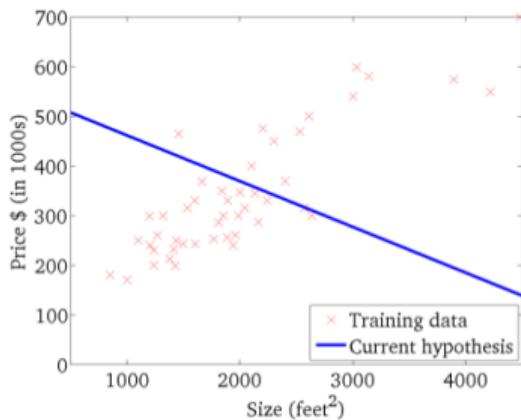


Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

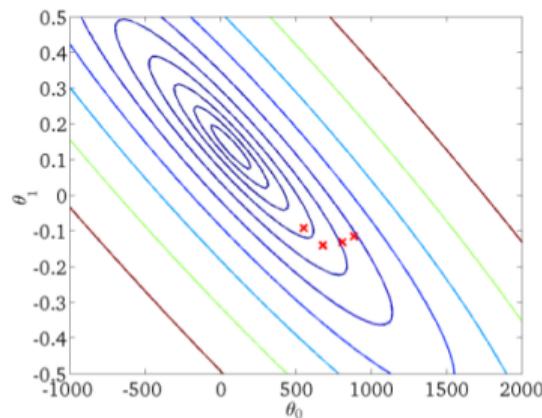
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



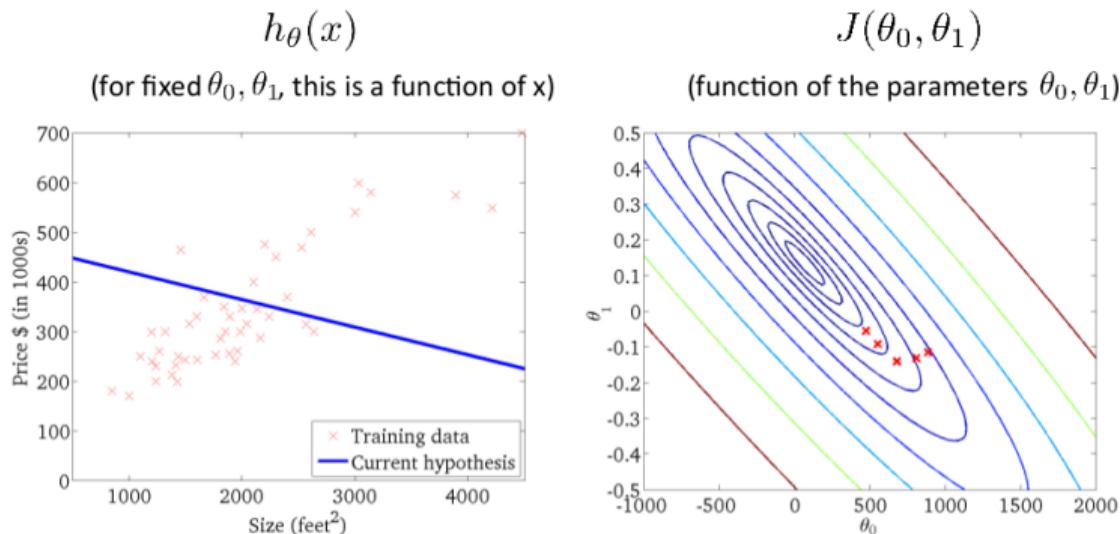
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

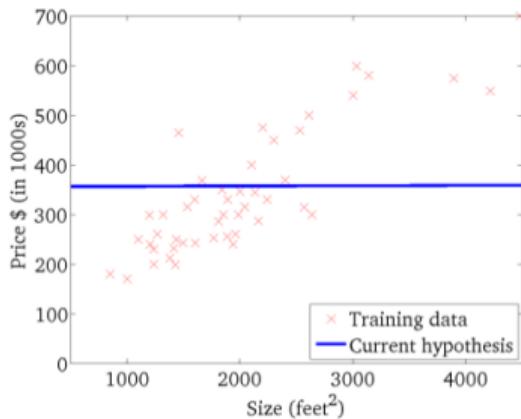


Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

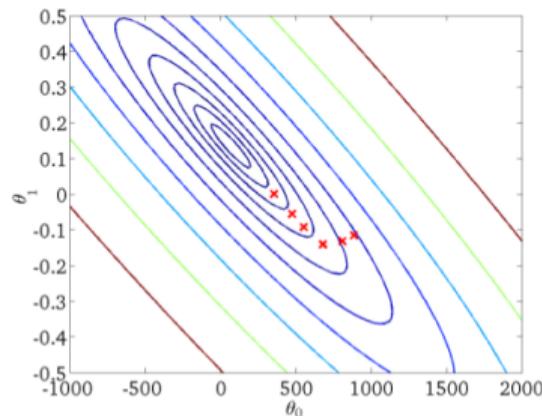
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )

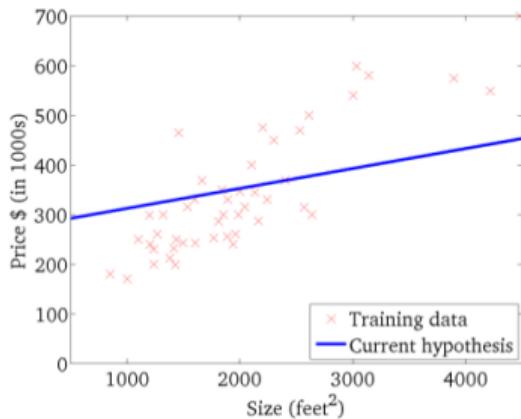


Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

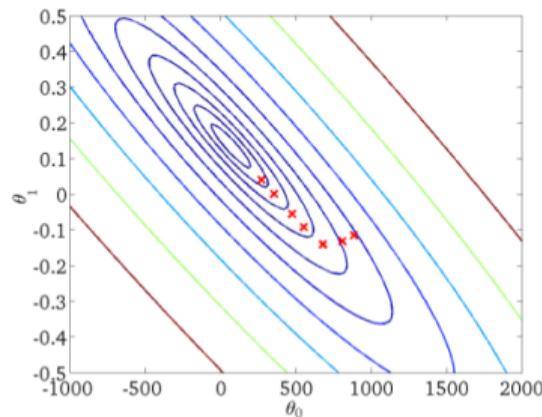
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )

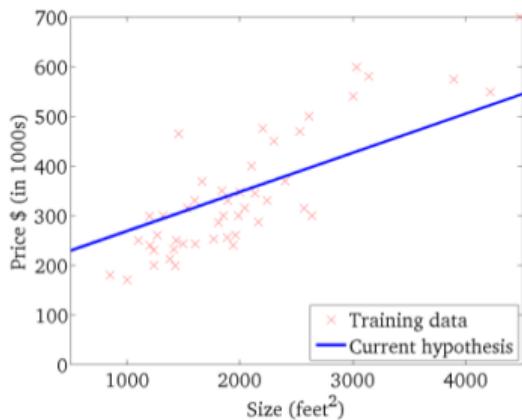


Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

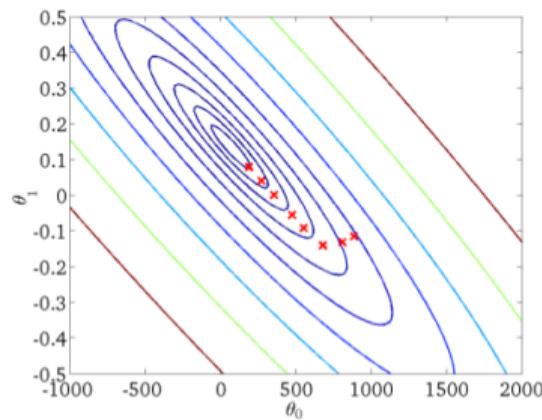
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )

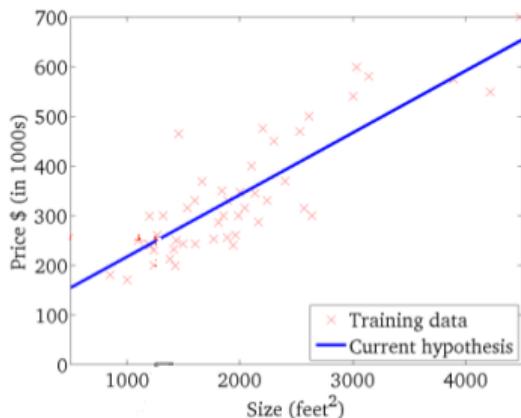


Andrew Ng

# “Gradient descent” para ajuste de una recta (Imágenes de Andrew Ng, Stanford:)

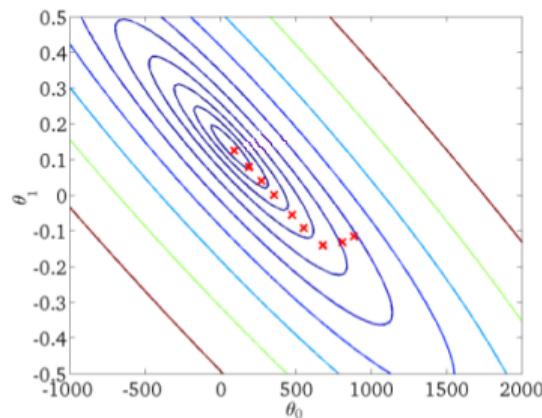
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



Andrew Ng

# Algunos aspectos de implementación práctica

En Python, usaremos el producto matricial y la transpuesta. Supongamos que tenemos los siguientes datos en un dataframe llamado df:

```
>>> import pandas as pd  
>>> df = pd.DataFrame(  
...     {  
...         'y': [2.34, 4.39, 1, 2.3],  
...         'x1': [1.04, 2, 1.9, 3.01],  
...         'x2': [0.43, 0.23, 0.9, 0.87]  
...     }  
... )  
>>> df  
  
      y      x1      x2  
0  2.34  1.04  0.43  
1  4.39  2.00  0.23  
2  1.00  1.90  0.90  
3  2.30  3.01  0.87
```

# Algunos aspectos de implementación práctica

Creamos la matriz de diseño  $X$  con la instrucción concatenate de numpy que nos permite concatenar columnas:

```
>>> import numpy as np
>>> X = np.concatenate(
...         [np.ones((df.shape[0], 1)), df.drop(columns='y').values],
...         axis=1
... )
>>> X
array([[1.    , 1.04, 0.43],
       [1.    , 2.   , 0.23],
       [1.    , 1.9  , 0.9 ],
       [1.    , 3.01, 0.87]])
```

Para implementar

$$\theta \leftarrow \theta - \frac{2}{n} \alpha \mathbf{X}^T \cdot (\mathbf{X}\theta - y).$$

basta con esta línea de código:

```
theta = (theta - 2 / y.shape[0]
         * alpha * np.matmul(X.T, np.matmul(X, theta) - y))
```

Hemos usado el operador de transposición T y el producto matricial `matmul` .

## Algunos aspectos de implementación práctica: normalización de características

- Si trabajamos con variables que presentan órdenes de magnitud muy distintos, es muy recomendable normalizarlas para que tengan un orden de magnitud similar.
- Por ejemplo, si  $X_1$  es del orden de 10000, pero  $X_2$  del orden de 2 o 3, la superficie de la función de coste puede presentar variaciones muy bruscas que dificultan la búsqueda del mínimo.

# Algunos aspectos de implementación práctica: normalización de características

Para normalizar las variables:

Una opción consiste en restarles su media cada una, y dividirlas por su desviación típica (o por su rango, es decir la diferencia entre mínimo y máximo).

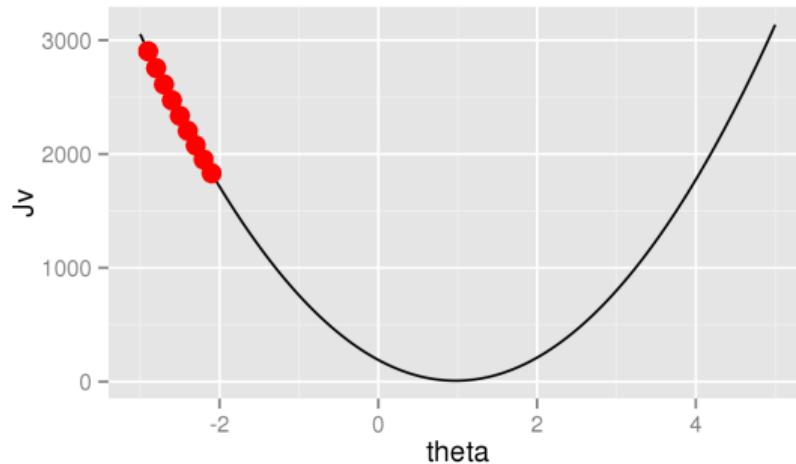
Por ejemplo, en Python, usaremos la librería de machine learning `sklearn` que en su módulo `preprocessing` incluye varios procedimientos de estanderizar o transformar los datos

```
>>> from sklearn.preprocessing import StandardScaler  
>>> scaler = StandardScaler()  
>>> scaler = scaler.fit(df.drop(columns='y').values)  
>>> columnas_normalizadas = scaler.transform(df.drop(columns='y').values)  
>>> X = np.concatenate([np.ones((df.shape[0], 1)), columnas_normalizadas], axis=1)
```

Hemos excluido la primera columna del proceso de normalización...

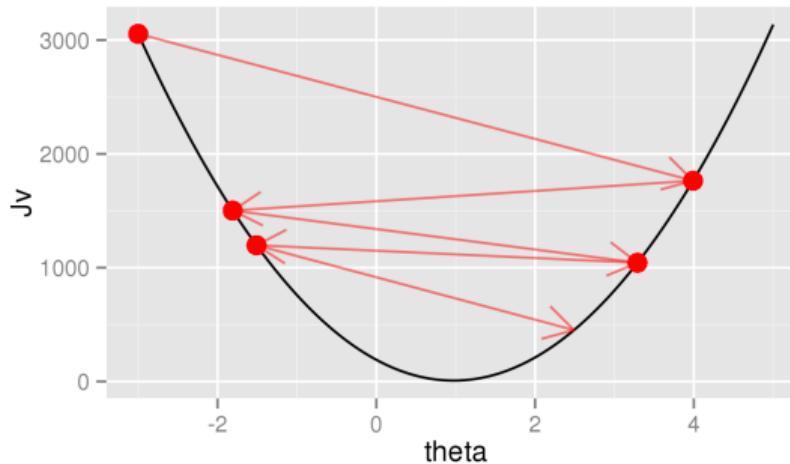
## Algunos aspectos de implementación práctica: elección de $\alpha$

Recordad que es importante la elección de  $\alpha$ . Si  $\alpha$  es demasiado pequeño, el algoritmo converge muy lentamente,



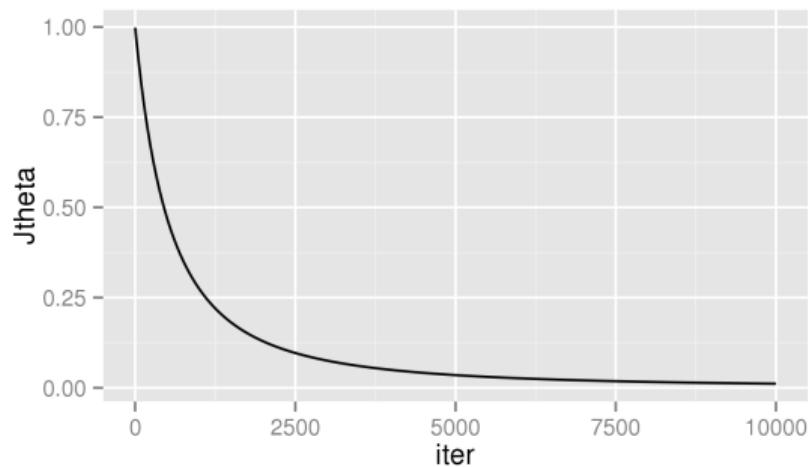
# Algunos aspectos de implementación práctica: elección de $\alpha$

Si  $\alpha$  es demasiado grande, el algoritmo oscila y puede incluso diverger,



## Algunos aspectos de implementación práctica: elección de $\alpha$

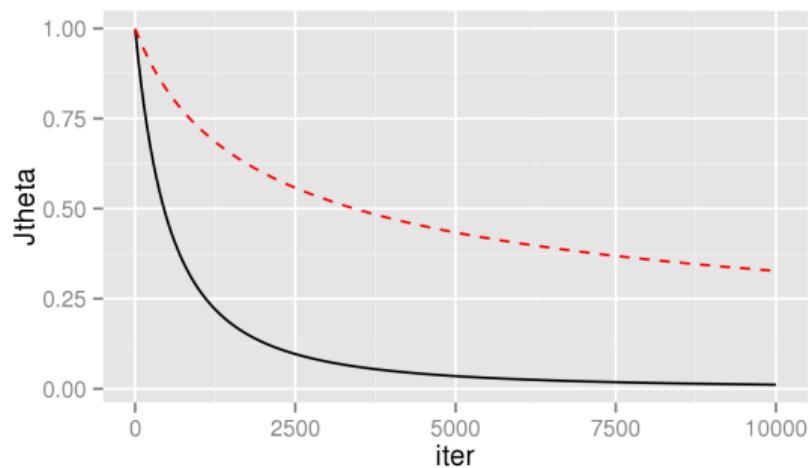
Es importante monitorizar la evolución de la función coste a medida que iteramos el algoritmo de gradiente:



Línea negra: la convergencia del algoritmo es rápida: buena elección de  $\alpha$

## Algunos aspectos de implementación práctica: elección de $\alpha$

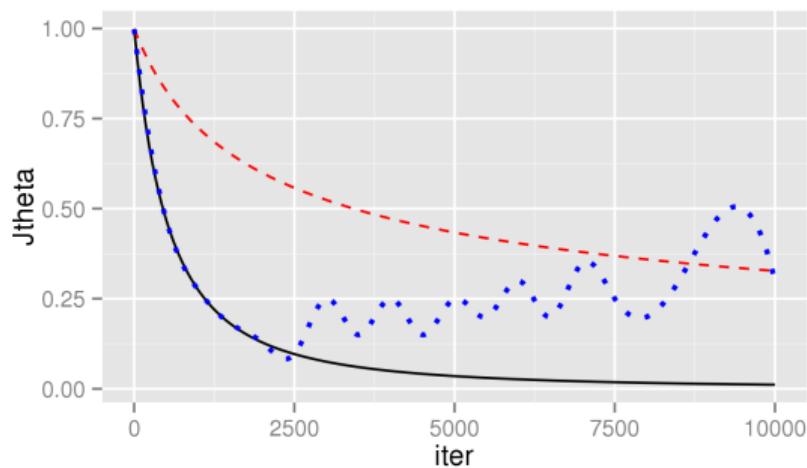
Es importante monitorizar la evolución de la función coste a medida que iteramos el algoritmo de gradiente:



Línea roja: la convergencia es muy lenta:  $\alpha$  demasiado pequeño.

## Algunos aspectos de implementación práctica: elección de $\alpha$

Es importante monitorizar la evolución de la función coste a medida que iteramos el algoritmo de gradiente:



Línea azul: el algoritmo no converge, quizás  $\alpha$  sea demasiado grande.

## Algunos aspectos de implementación práctica: elección de $\alpha$

Para un valor de  $\alpha$  elegido:

- Podemos fijar un criterio automático de parada: por ejemplo paramos si la variación en el valor de  $J$  la iteración es inferior a  $\varepsilon = 0.001$ .
- Es bueno intentar ver el perfil de decrecimiento de  $J(\theta)$  para valores muy diferentes del número de iteraciones: por ejemplo,  $\text{iter}=30$ ,  $\text{iter}=3000$  y  $\text{iter}=3000000$ .

Y además es importante

probar diferentes valores de  $\alpha$ , por ejemplo:

$$\alpha = 0.001 \curvearrowright 0.003 \curvearrowright 0.01 \curvearrowright 0.03 \curvearrowright 0.1 \curvearrowright 0.3 \curvearrowright 1\dots$$

# Ecuaciones normales para la regresión lineal

En el caso de la regresión lineal simple o múltiple, existe una expresión analítica sencilla para el mínimo de la función de coste.

Recordad que habíamos obtenido:

$$\nabla J(\theta) = \frac{2}{n} \mathbf{X}^T \cdot (\mathbf{X}\theta - \mathbf{y}).$$

Buscamos el mínimo como solución de  $\nabla J(\theta) = 0$  (*que llamamos ecuaciones normales*), y obtenemos

$$\theta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

## Nota

Además comprobamos que el Hessiano es

$$H(J) = \left( \frac{\partial^2 J}{\partial \theta_i \partial \theta_j} \right)_{1 \leq i,j \leq k} = \frac{2}{n} \mathbf{X}^T \mathbf{X},$$

que es una matriz definida positiva, por lo que se trata de un mínimo global.

Cualquier software de análisis de datos tiene implementadas estas soluciones para la regresión lineal, Python también por supuesto en la librería scikit-learn.  
Siempre se debe usar la solución analítica?

## Alg. Gradiente

- Tenemos que escoger  $\alpha$ .
- Requiere muchas iteraciones.
- Funciona incluso si  $k$  es muy grande

## Solución analítica

- No requiere escoger ningún  $\alpha$ .
- No requiere iteraciones.
- Requiere calcular  $(\mathbf{X}^T \mathbf{X})^{-1}$  que puede ser muy costoso si  $k$  es grande.

$$Y = X\theta + \varepsilon,$$

donde  $Y$  es el vector de las observaciones.

$X$ : matriz de diseño.

- Un marco bastante flexible para modelizar la respuesta promedio.
- La hipótesis de normalidad (campana de Gauss) para  $\varepsilon$  permite llevar a cabo tests de significancia.
- Las estimaciones de  $\theta$  proporcionadas por este teoria coincide con las estimaciones de mínimos cuadrados.