Contenido

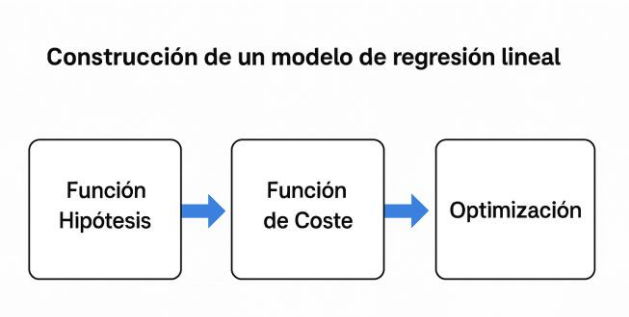
1. REGRESIÓN LINEAL.

Lo primero que vamos a ver son las características principales de la regresión lineal:

- Aprendizaje supervisado.
- Aprendizaje basado en modelo (construir función hipótesis).
- Se corresponde con un modelo lineal (función lineal).
- Realiza predicciones computando una suma ponderada de las características de entrada y sumándole una constante conocida como bias.
- Intenta predecir valores continuos.

En este punto de la unidad aprenderemos cómo construir un modelo de regresión lineal desde cero, siguiendo los pasos fundamentales que permiten entender su funcionamiento interno.

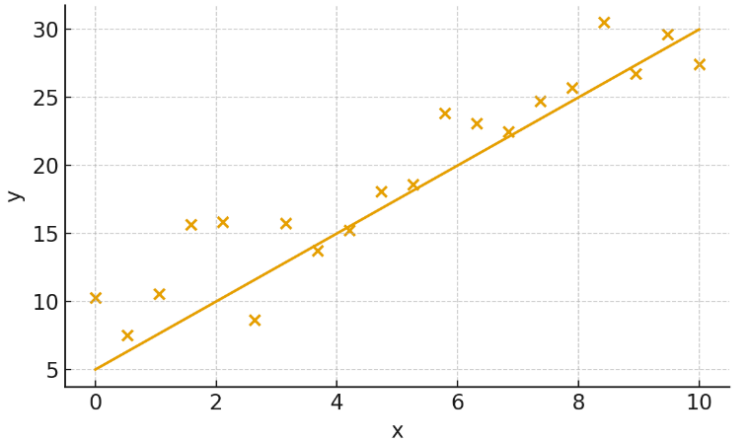
Comenzaremos definiendo la **función hipótesis**, que describe la relación entre las variables de entrada y la salida. Después, veremos **cómo se construye el modelo** y cómo evaluamos su rendimiento mediante la **función de coste**, en este caso el **Error Cuadrático Medio (MSE)**. Finalmente, aplicaremos un método de **optimización** muy utilizado en Machine Learning: el descenso por gradiente (**Gradient Descent**), que nos permitirá ajustar los parámetros del modelo para minimizar el error. Este recorrido no solo nos ayudará a comprender la teoría, sino también a sentar las bases para implementarlo en código y aplicarlo a problemas reales.



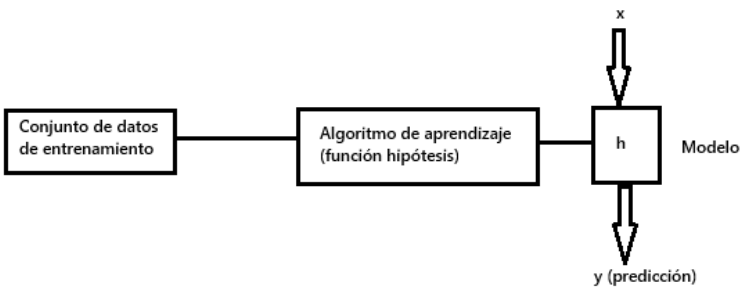
2. EJEMPLO.

Partimos de un ejemplo para poderlo ver: **incidentes pasados**

Después de recopilar los datos de incidentes pasados, junto con el número de equipos afectados y el coste asociado a cada uno, podemos observar que la distribución resultante sigue una tendencia aproximadamente lineal. Es decir, si aplicamos una función matemática lineal y tratamos de ajustarla a nuestro conjunto de datos, la recta obtenida se adapta razonablemente bien al comportamiento general que muestran las observaciones.



El objetivo de este algoritmo de *machine learning* es (cómo vimos en la pasada unidad), que partiendo únicamente de los ejemplos de nuestro conjunto de entrenamiento, **construir una función hipótesis (es decir, un modelo)** que se ajuste de la mejor manera posible a la tendencia real de los datos. El algoritmo se encarga automáticamente de encontrar los valores óptimos de los parámetros para que la función represente adecuadamente la relación existente en las observaciones.



3. NOTACIÓN.

Vamos a ver la notación que vamos a usar a partir de ahora:

	Nº de sistemas afectados	Gasto en euros
i1	1000	10000
i2	1500	20000
i3	500	5500
…	…	…
im	200	2500

- m= número de ejemplos de nuestro conjunto de datos.
- Una variable de entrada : x_1
- Varias variables de entrada o features: $x_1,x_2,\ldots,x_n$
- y=variables de salida o target.
- (x,y)= ejemplo de entrenamiento, conjunto de datos o dataset.
- n= número de características/variables de entrada

Ejemplo entrenamiento:

Incidente 1: i_1 (100,10000)

Incidente 3: i_3 (500,5500)

4. FUNCIÓN HIPÓTESIS.

4.1. Proceso general

Conjunto de datos → Función hipótesis → Modelo → Predicciones

Recordemos de nuevo el proceso general. El punto de partida es un **conjunto de datos**, que representa nuestra *experiencia pasada*. Con esa información alimentamos una **función matemática**, llamada **función hipótesis**, cuya misión es describir la relación entre las variables de entrada y la salida.

Esta función hipótesis incluye una serie de **parámetros que se van ajustando automáticamente** conforme el algoritmo analiza los datos del conjunto de entrenamiento. Una vez que estos parámetros han sido optimizados, obtenemos lo que denominamos **modelo**. En otras palabras, el **modelo es la función hipótesis después de haber aprendido y adaptarse a nuestros datos.**

Finalmente, será este modelo ya ajustado el que utilizaremos para **realizar predicciones** sobre nuevos casos que no estaban en el conjunto de entrenamiento.

Vamos a centrarnos en la **función hipótesis del algoritmo de regresión lineal**, es decir, la expresión matemática que utilizamos para aproximar la relación entre nuestras variables mediante una línea recta:

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

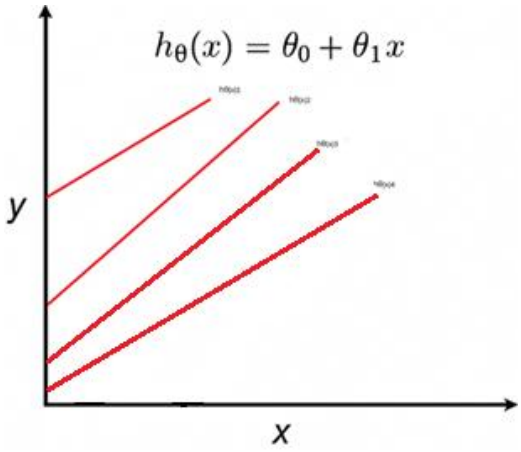
Esta función es igual a la de la recta:

$$y=mx+n$$

Si la ponemos así se ve que es igual:

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

Si la representamos gráficamente la podemos ver así:



¿Qué determina que mi función sea de una manera u otra? Es decir, ¿que sea la primera, segunda, tercera o cuarta? Lo que lo va a determinar van a ser los parámetros del modelo Θ_0 y Θ_1 respectivamente.

- Θ_0 es el punto con el que corta en el eje y.
- Θ_1 es el que va a indicar la pendiente (inclinación de la recta).

El objetivo es encontrar el **valor óptimo de " y"** que ajuste lo mejor posible esta recta a la tendencia de nuestro conjunto de datos.

4.2. Regresión lineal univariable

Cuando contamos con una única característica de entrada, el algoritmo recibe el nombre de **regresión lineal univariable** (o unidimensional). En este caso, su función hipótesis se expresa de la siguiente manera:

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

4.3. Regresión lineal multivariable

Cuando tenemos más de una característica de entrada el algoritmo se denomina regresión lineal **multivaluada** y su función hipótesis es:

$$h_\theta(x)=\theta_0+\theta_1 x_{1+}\ \theta_2 x_{2+}\ \theta_3 x_{3+\ldots}\ +\theta_N x_N$$

De ahí que el algoritmo pueda interpretarse como una **suma ponderada de las características de entrada**, donde cada una se multiplica por su correspondiente parámetro del modelo (θ_1, θ_2, …, θ_n). A esta combinación lineal se le añade además un **término de sesgo** o *bias*, representado por θ_0.

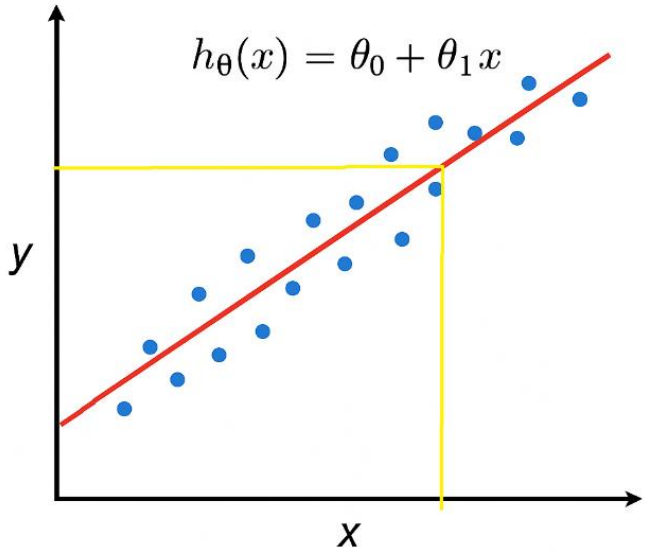
Pero entonces, ¿Cómo se ajusta esta función matemática a la tendencia de nuestro conjunto de datos? Este ajuste que es lo que denominamos entrenamiento del algoritmo que es lo siguiente que vamos a ver.

5. CONSTRUCCIÓN DEL MODELO.

Vamos a construir el modelo con una función hipótesis univariable o unidimensional.

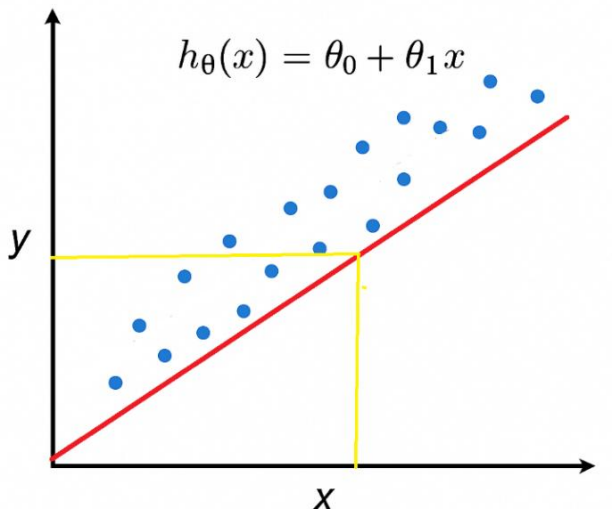
- Nuestro objetivo es encontrar los valores de $\boldsymbol{\theta_0}$ y $\boldsymbol{\theta_1}$ que permitan generar la función hipótesis que mejor se ajuste al conjunto de datos de entrenamiento *(x, y)*.
- Para ello, el algoritmo **minimiza una función de coste**, lo que nos permite obtener los parámetros $\boldsymbol{\theta_0}$ y $\boldsymbol{\theta_1}$ óptimos.

A continuación, veremos un primer ejemplo de representación de la función hipótesis óptima :



En amarillo representamos por ejemplo un numero de equipos afectados (x) y vemos como es un coste aproximado en y. Podemos comprobar que esta predicción es correcta gracias a que la función se ha adaptado correctamente a la tendencia de nuestros datos de entrenamiento.

Si miramos otro caso en el que no sea así, por ejemplo:



Cuando realizamos una predicción en este ejemplo para un determinado número de equipos, el resultado obtenido no es correcto porque **no sigue la tendencia real de nuestros datos**. En los incidentes pasados que hemos recopilado, para números de equipos similares a los que queremos predecir, el **coste real** observado es considerablemente mayor que el estimado por el algoritmo. Esto significa que el modelo **no está bien ajustado** y, por tanto, **no representa adecuadamente el comportamiento de los datos**.

Entonces, ¿cómo podemos encontrar los valores óptimos de **θ_0** y **θ_1** que permiten que nuestra función hipótesis se ajuste correctamente a la tendencia de los datos? La respuesta está en una función matemática llamada **función de coste** o **función de error**, uno de los componentes fundamentales de muchos algoritmos de *machine learning*.

6. FUNCIÓN DE COSTE.

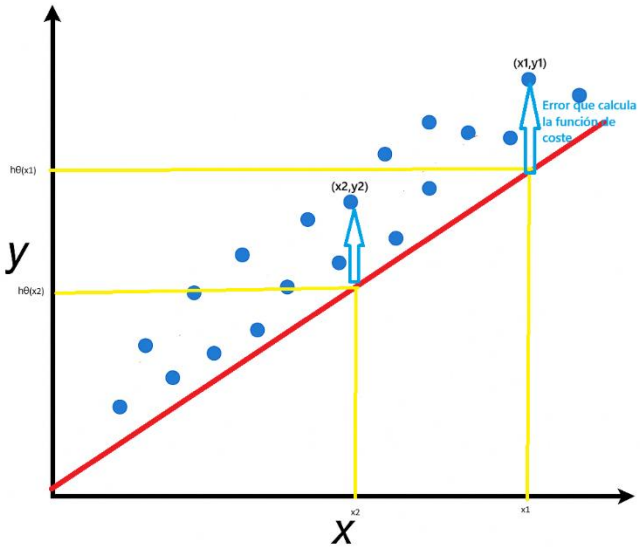
¿Qué hace la función de coste?

La función de coste **mide el error** entre las predicciones que realiza actualmente nuestro algoritmo y los valores reales de nuestro conjunto de datos. Como nuestro conjunto de entrenamiento está **etiquetado**, sabemos cuál debería ser el coste asociado a cada número de equipos afectados. Si la predicción que hace la función hipótesis se aleja de ese valor real, significa que **no está bien ajustada**.

¿Por qué necesitamos medir este error?

Porque a continuación utilizaremos otra función, llamada **función de optimización**, cuyo propósito es **modificar los parámetros θ_0 y θ_1** para que el error calculado por la función de coste se reduzca lo máximo posible. La función de optimización irá ajustando estos parámetros de forma gradual, haciendo que la función hipótesis cambie hasta que el error no pueda seguir disminuyendo.

En otras palabras: **este proceso de minimizar el error ajustando los parámetros es lo que conocemos como entrenar un algoritmo de *machine learning*.**



A un mismo algoritmo de *machine learning* se le pueden aplicar distintas **funciones de coste**, dependiendo de cómo queramos medir el error. En el caso de la **regresión lineal**, la función de coste más utilizada es el **Mean Squared Error (MSE)** o error cuadrático medio.

Para entender mejor esto, vamos a ver cómo funciona el algoritmo de regresión lineal. Primero, se **inicializan los parámetros** θ_0 y θ_1 de forma aleatoria, lo que nos da una primera función hipótesis. A continuación, necesitamos evaluar qué tan buenas (o malas) son sus predicciones, y para ello empleamos una **función de error**, que se calcula mediante la siguiente ecuación:

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

- Donde m es el número de ejemplos que tenemos en nuestro conjunto de datos de entrenamiento.
- $h_\theta(x^{(i)})$ La predicción de nuestra función hipótesis para cada uno de nuestro conjunto de datos.
- $y^{(i)}$ Menos la etiqueta que tiene ese conjunto de datos de entrenamiento (el valor que tiene).
- Y después lo elevamos al cuadrado (así nunca sale negativo).

Una vez que hemos calculado el error para cada ejemplo del conjunto de entrenamiento, obtenemos un valor numérico por cada uno. El siguiente paso consiste en **sumar todos esos errores**, lo que nos da un único valor que representa el **error total** producido por la función hipótesis actual.

Posteriormente, este valor se **divide entre el número total de ejemplos** (y, en la fórmula clásica, también entre 2). Esto nos permite obtener la **media del error cuadrático**, que es precisamente lo que da nombre al **Mean Squared Error (MSE)**: se calcula el error, se eleva al cuadrado y se hace la media de todos ellos.

El resultado final es, por tanto, **un único número** que indica cuán bien o mal se está ajustando nuestra función hipótesis a los datos.

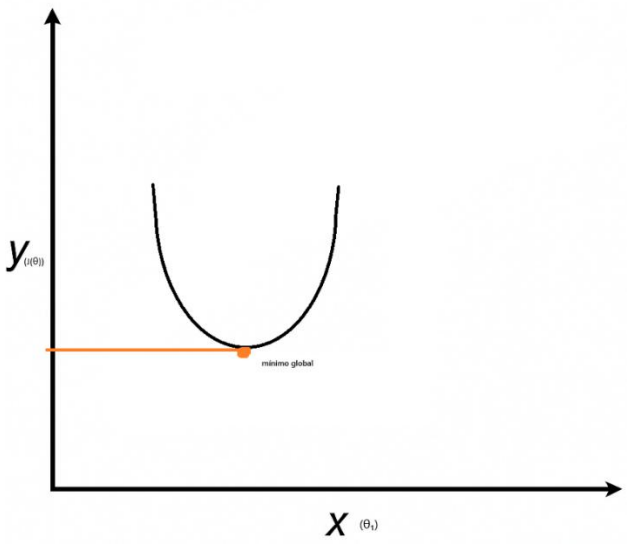
7. FUNCIÓN DE OPTIMIZACIÓN.

Una vez calculado este error, utilizamos una **función de optimización** para modificar los parámetros θ_0 y θ_1. Esta función va ajustando los parámetros de forma iterativa, haciendo que el error vaya disminuyendo hasta que ya no sea posible reducirlo más. En ese punto, habremos **minimizado la función de coste** y obtenido los valores óptimos de θ_0 y θ_1.

En resumen, este proceso permite que la función hipótesis se ajuste **lo mejor posible** a la tendencia real de nuestros datos.

Al igual que ocurre con la función de coste, podemos emplear **distintas funciones de optimización** dentro de una misma técnica de *machine learning*. Sin embargo, la función de optimización más utilizada, con mucha diferencia, es **Gradient Descent** (*descenso por gradiente*), que es la que vamos a explicar.

Para ilustrarlo, consideremos el siguiente ejemplo: supongamos que no incluimos el término θ_0 en nuestra función hipótesis. En ese caso, la **función de coste** se simplificaría de la siguiente manera:



Esta función tiene forma **convexa**, lo que significa que posee un único **mínimo global**, representado en la gráfica como el punto más bajo posible para el valor de $J(\theta)$.

Antes de avanzar, es importante comprender que la función de optimización está estrechamente relacionada con la **función de coste** utilizada por el algoritmo.

En el caso de la **regresión lineal**, la función de coste que empleamos es el **Mean Squared Error (MSE)**:

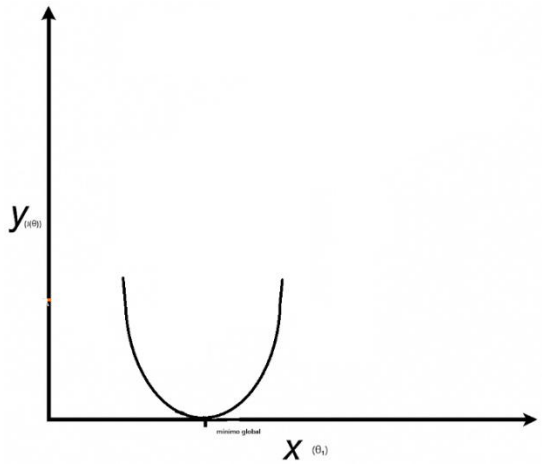
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Nuestro objetivo es **minimizar el error** de la función de coste hasta alcanzar su **mínimo global**, que es el punto donde el error no puede reducirse más.

Para comprender mejor este proceso, vamos a utilizar un ejemplo con una función de error más sencilla. Partimos de la siguiente función:

$$J(x) = (x-3)^2$$

Su representación gráfica sería la siguiente:



El mínimo global nos indica el valor más bajo que puede alcanzar $J(x)$. En este caso, dado que se trata de una ecuación sencilla, podemos comprobar fácilmente que cuando $J(x) = 0$, el valor correspondiente de x es $x = 3$.

$$J(x) = 0$$

$$J(x) = (3-3)^2 = 0$$

Sin embargo, necesitamos obtener el valor mínimo de manera **programática**, y para ello utilizamos **Gradient Descent**. El proceso comienza inicializando los parámetros con valores aleatorios; por ejemplo, podemos fijar inicialmente x(que aquí representa θ_1) en **10**.

A partir de ese valor inicial, aplicamos el concepto de **gradiente**, que es equivalente a la **pendiente** de la función o, más específicamente, a su **derivada**. El gradiente nos indica en qué dirección debemos movernos para reducir el error y encontrar el valor óptimo.

Gradiente = pendiente = derivada

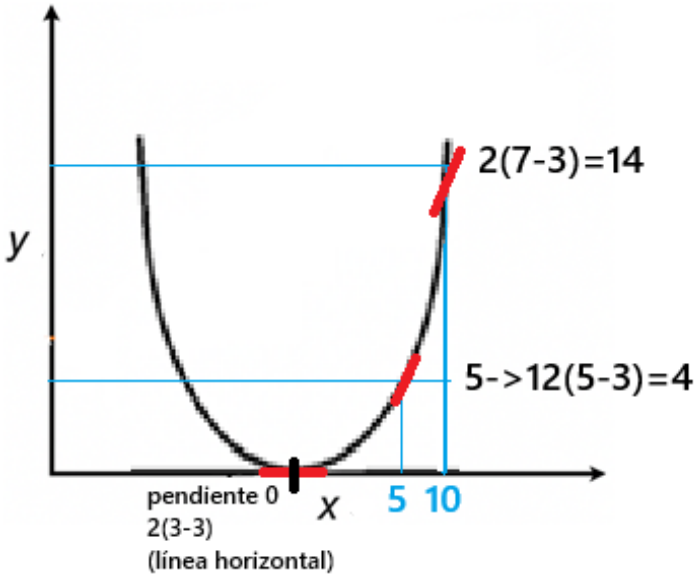
¿Cómo utilizamos el gradiente?

Calculamos la **derivada de** $J(x)$en ese punto con respecto a x. Esa derivada nos dirá si debemos aumentar o disminuir el valor de x para acercarnos al mínimo global.

A partir de ahí, vamos actualizando x paso a paso, siguiendo la dirección que marca la derivada, hasta que la función de error ya no pueda reducirse más.

$$\frac{d}{dx}J(x) = 2(x-3)$$

Si lo representamos gráficamente:



Podemos concluir que la derivada en un punto nos indica cómo cambia la función a medida que nos alejamos o nos acercamos del valor óptimo de x.

- **Cuanto más lejos estamos del mínimo global**, mayor es el valor de la derivada.
- **A medida que nos acercamos al mínimo**, la derivada disminuye hasta hacerse prácticamente cero.

Esto es fundamental, porque la derivada nos dice **en qué dirección debemos mover el valor de** x para acercarnos al mínimo global. Por tanto, la forma de conseguir que x se aproxime progresivamente al mínimo es:

$$x = x - \frac{d}{dx}J(x)$$

Gradient descent funciona básicamente así salvo una pequeña salvedad. Lo primero de todo va a repetir este proceso hasta que el valor que tiene en este caso la x (que es nuestro parámetro) no varíe:

$$x := x - \alpha \cdot \frac{d}{dx} J(x)$$

¿Qué significa esta fórmula?

- $\boldsymbol{x}$
  Es el valor actual del parámetro (por ejemplo, θ_1).
- $\frac{\boldsymbol{d}}{\boldsymbol{dx}}\boldsymbol{J(x)}$
  Es la **derivada** de la función de coste en ese punto.
  Indica **en qué dirección está subiendo** la función.
- $\boldsymbol{\alpha}$
  Es la **tasa de aprendizaje (learning rate)**.
  Controla **cuánto** modificamos el valor del parámetro en cada paso.

¿Por qué restamos el gradiente?

Porque la derivada nos dice la dirección en la que la función **crece**. Para bajar hacia el mínimo global, debemos movernos **en la dirección contraria** al gradiente. Por eso restamos:

- Si la derivada es **positiva**, restar hace que x disminuya → vamos hacia el mínimo.
- Si la derivada es **negativa**, restar un número negativo hace que x aumente → también vamos hacia el mínimo.

¿Qué pasa al repetir este proceso?

Aplicamos esta regla de actualización **muchas veces**. Cada iteración acerca x un poco más al valor óptimo. A medida que nos acercamos al mínimo global:

- El gradiente se hace más pequeño.
- Los ajustes se vuelven más suaves.
- El algoritmo acaba **convergiendo** al valor óptimo de x.

Vamos a usar GD en nuestro ejemplo anterior:

Utilizamos un valor llamado **α (alpha)**, que debemos fijar de antemano. Suele tomar valores como $0.5 = \frac{1}{2}$. Este parámetro controla la **velocidad con la que el algoritmo avanza hacia el punto óptimo**.

En otras palabras, α determina el tamaño del paso que damos en cada iteración del descenso por gradiente.

$$x = x - \alpha \frac{d}{dx} J(x) = x - \alpha 2(x-3)$$

Inicializamos con x=10

$$x = 10 - \frac{1}{2} 2(10-3) = 3$$

Modificamos el valor de x=3

$$x = 3 - \frac{1}{2}(3-3) = 3$$

Si en alguna iteración observamos que el valor de x deja de cambiar (o cambia de forma prácticamente imperceptible) durante varias actualizaciones consecutivas, significa que **hemos alcanzado el punto óptimo**. Es decir, hemos encontrado el valor del parámetro que **minimiza la función de error**.

8. GRADIENT DESCENT APLICADO A MSE.

A continuación, vamos a aplicar este mismo procedimiento al cálculo de nuestra función de error MSE:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Vamos a ver por tanto la **derivación completa del MSE para regresión lineal** y cómo se obtienen las actualizaciones de θ_0 y θ_1 mediante descenso por gradiente.

1. Partimos de la función de coste MSE

Para regresión lineal univariable:

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

La función de coste MSE (sin el 1/2n opcional por simplicidad inicial) es:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Donde:

- (m): número de ejemplos del conjunto de entrenamiento
- x^i: entrada del ejemplo i
- y^i: salida real del ejemplo i

2. Queremos minimizar J

Eso significa encontrar los valores de θ_0 y θ_1 que hagan J lo más pequeño posible.

Para ello necesitamos las **derivadas parciales** de J respecto a cada parámetro:

$$\frac{d}{d\theta_0}J(\theta_0,\theta_{01}) \qquad y \qquad \frac{d}{d\theta_1}J(\theta_0,\theta_1)$$

3. Calculamos la derivada parcial respecto a θ_0

$$\frac{d}{d\theta_0}J(\theta_0,\theta_1)=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}-y^{(i)}\right)\right)$$

4. Derivada parcial respecto a θ_1

$$\frac{d}{d\theta_1}J(\theta_0,\theta_1)=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}-y^{(i)}\right)\right)x^{(i)}$$

5. Sustituyendo estas derivadas en Gradient Descent

Regla general:

$$\theta_j=\theta_j-\frac{d}{d\theta_j}J(\theta_0,\theta_1)$$

Entonces:

Actualización de θ_0:

$$\theta_0=\theta_0-\alpha\frac{1}{m}\sum_{i=1}^{nm}\left(h_\theta(x^{(i)}-y^{(i)})\right)$$

Actualización de θ_1:

$$\theta_1=\theta_1-\alpha\frac{1}{m}\sum_{i=1}^{nm}\left(h_\theta(x^{(i)}-y^{(i)})\right)x^{(i)}$$

Interpretación:

- θ_0 se ajusta mirando **el error general promedio**.
- θ_1 se ajusta mirando **el error ponderado por el valor de x**.
- Ambos parámetros se modifican **en la dirección que reduce el error**.
- Repetimos esto hasta que los valores convergen (ya no cambian).

Repetir hasta que Θ_0 y Θ_1 no varíen (valor óptimo que minimiza la función de error).

Ejemplo:

Vamos con un ejemplo numérico **paso a paso** con un conjunto de datos pequeño (m = 3), aplicando **regresión lineal + MSE + descenso por gradiente**.

1. Conjunto de datos

Imagina que tenemos esta relación (bastante lineal):

Ejemplo (i)	(x^i)	(y^i)
1	1	2
2	2	3
3	3	4

A simple vista, la relación real se parece a:

$y = x + 1$

2. Modelo y función de coste

Función hipótesis (regresión lineal univariable):

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

Función de coste (MSE):

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Con (m = 3).

3. Gradientes (derivadas parciales)

Regla de actualización (Gradient Descent):

$$\theta_j = \theta_j - \frac{d}{d\theta_j} J(\theta_0, \theta_1)$$

Vamos a tomar:

- θ_0 =0, $\theta_1 = 0$ al inicio
- Tasa de aprendizaje: α = 0.1)

4. Iteración 0,primer paso de descenso por gradiente:

a) Predicciones iniciales

Con θ_0 =0, $\theta_1 = 0$:

- Ejemplo 1: $h_\theta(1) = 0 + 0 \cdot 1 = 0$
- Ejemplo 2 $h_\theta(2) = 0 + 0 \cdot 2 = 0$
- Ejemplo 3 $h_\theta(3) = 0 + 0 \cdot 3 = 0$

b) Errores $h_\theta(x^{(i)} - y^{(i)})$

Ejemplo	(x^i)	(yi)	$h_\theta(x^{(i)})$	Error (h-y)
1	1	2	0	(-2)
2	2	3	0	(-3)
3	3	4	0	(-4)

c) Gradiente respecto a θ_0 :

$$\frac{\partial}{\partial \theta_0} J = \frac{1}{m} \sum (h - y) = \frac{1}{3}(-2 - 3 - 4) = \frac{-9}{3} = -3$$

d) Gradiente respecto a θ_1:

$$\frac{\partial}{\partial \theta_1} J = \frac{1}{3} \sum (h - y) \cdot x$$

$$= \frac{1}{3}[(-2) \cdot 1 + (-3) \cdot 2 + (-4) \cdot 3] = \frac{1}{3}(-2 - 6 - 12) = \frac{-20}{3} \approx -6{,}6667$$

e) Actualizamos parámetros

$$\theta_0^{(nuevo)} = \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0} = 0 - 0.1 \cdot (-3) = 0.3$$

$$\theta_1^{(nuevo)} = \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1} = 0 - 0.1 \cdot (-6{,}6667) \approx 0.6667$$

Después de la **primera iteración** tenemos:

- θ_{01} approx 0.3
- θ_1 approx 0.6667

La recta ya se parece un poco a (y = x + 1), pero aún no del todo.

"Si seguimos iterando, los valores seguirán ajustándose hasta que los cambios sean mínimos. En ese momento decimos que el algoritmo **ha convergido** y que hemos encontrado el modelo que mejor se adapta a nuestros datos según el criterio del MSE."

Ejercicio:

Realiza las siguientes iteración para :

Ahora usamos:

- $\Theta_0 = 0.3$
- $\Theta_1 = 0.6667$

Después de iterar con estos datos, intenta conseguir los valores para ajustar totalmente la función.