KIBERNUM
IT Academy

Módulo 5 – Aprendizaje de Máquina Supervisado

Algoritmo de Descenso de Gradiente

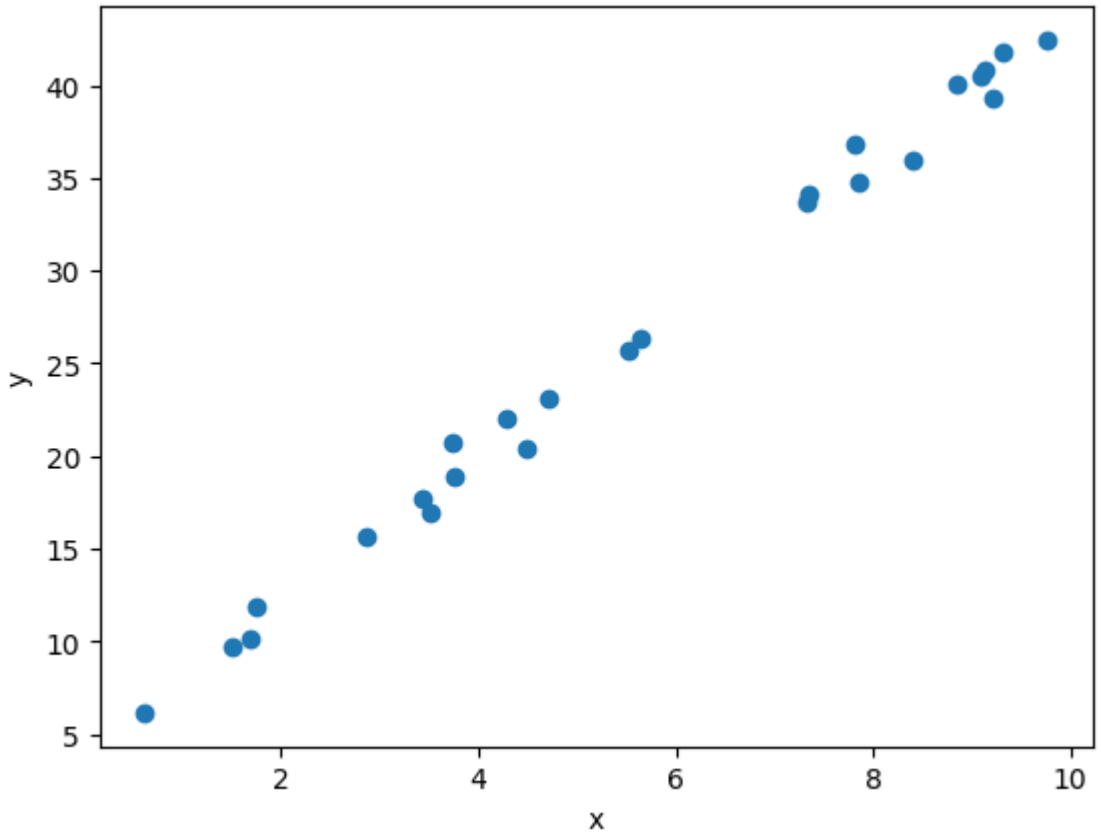
Especialización en Ciencia de Datos

De la Matemática a los Algoritmos

KIBERNUM

Problema

A partir de un set de datos, determinaremos la línea recta que mejor se ajusta.



KIBERNUM

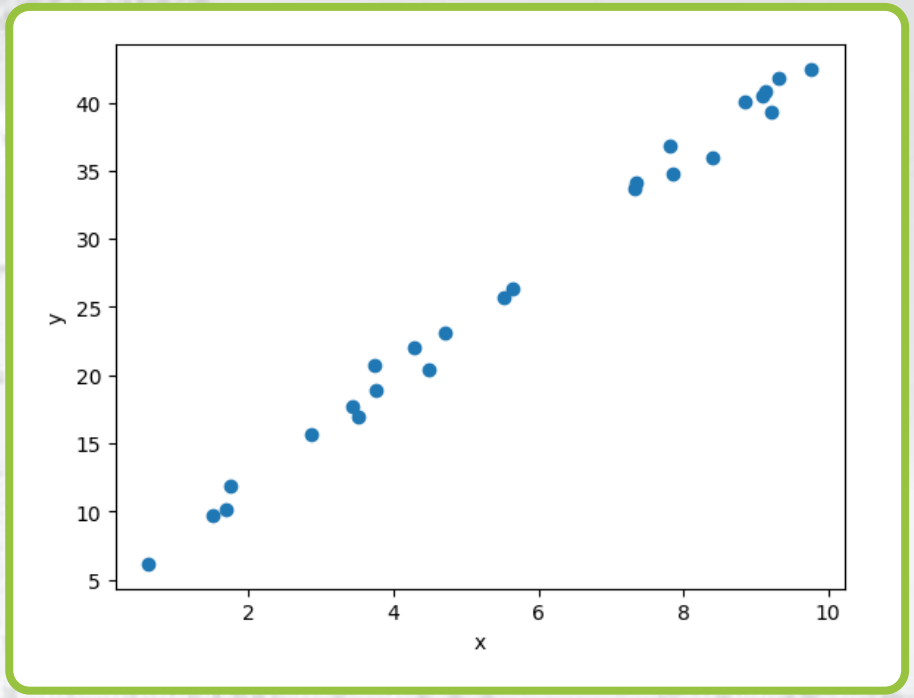
Problema

Este problema ya fue resuelto anteriormente. El problema se remite a encontrar los valores estimados para α y β que pueda proveer el mejor ajuste para los datos. En otras palabras, los valores que minimizan el siguiente problema:

$$\text{Find } \min_{\alpha,\beta} Q(\alpha,\beta), \quad \text{for } Q(\alpha,\beta) = \sum_{i=1}^{n} \hat{\varepsilon}_i^{\,2} = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2$$

Las siguientes ecuaciones permiten justamente determinar los valores estimados de α y β:

$$\begin{aligned}
\widehat{\alpha} &= \bar{y} - \widehat{\beta}\,\bar{x}, \\
\widehat{\beta} &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
&= \frac{s_{x,y}}{s_x^2} \\
&= r_{xy}\frac{s_y}{s_x}.
\end{aligned}$$



KIBERNUM

Forma Matricial

- Supongamos que tenemos n instancias de dato en un modelo lineal simple. En este caso, se sabe que la tarea consiste simplemente en determinar los valores de β_0 y β_1.

- Para efectos de visualización escribiremos la ecuación de las n instancias, puesto que de esa forma se aprecia con mayor claridad la matriz de datos.

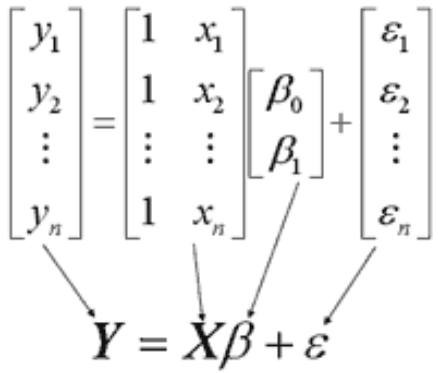
$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \qquad \text{for } i = 1, \ldots, n$$

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \epsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_2 + \epsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \epsilon_n \end{aligned}$$

KIBERNUM

Forma Matricial

- Ahora se hace más evidente la formulación matricial del problema de regresión.
- Entonces, en vez de escribir las n ecuaciones, nuestra regresión lineal simple, se reduce a esta simple notación:



$$Y = X\beta + \varepsilon$$

KIBERNUM

Forma Matricial

Con la forma matricial, podemos resolver problemas multivariados simplemente dándole las dimensiones apropiadas a las matrices.

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

$$\hat{y} = Xw$$

$$\hat{y} = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix} \qquad X = \begin{bmatrix} x_{0,0} & \cdots & x_{0,n} \\ \vdots & \ddots & \vdots \\ x_{m,0} & \cdots & x_{m,n} \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$$

Solución Matricial

Se puede demostrar que, al aplicar mínimos cuadráticos en notación matricial, se obtiene el siguiente resultado para la matriz de coeficientes.

$$\mathrm{w} = (X^T X)^{-1} X^T Y$$

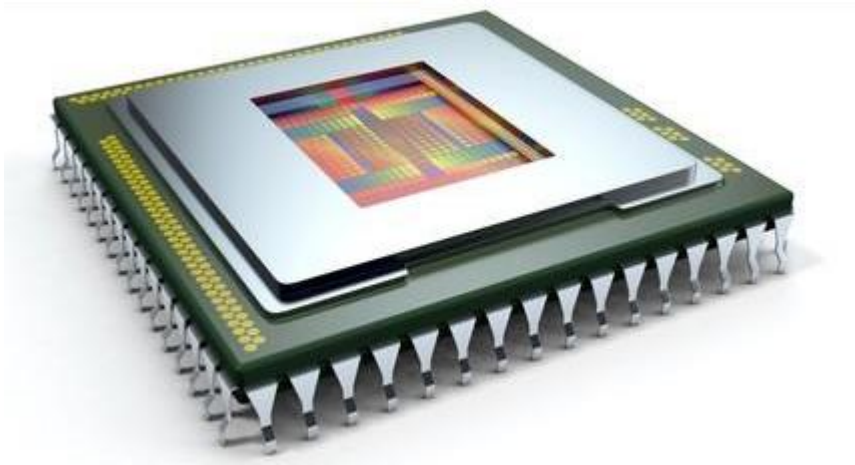
Como se puede observar, la determinación de los coeficientes **w** tiene una solución determinística, por lo tanto, se podría pensar que simplemente con operaciones matriciales que realiza un computador se podría encontrar la solución. Sin embargo, hay una limitante.

Limitantes de la Solución Matricial

A pesar de contar con una solución matricial, existe una limitante:

La inversión de una matriz es computacionalmente costosa en términos de procesamiento.

En la medida que hay una mayor cantidad de predictores y de instancias de dato, llegamos a los límites de procesamiento, lo cual hace que **este camino no sea muy viable**, pues no escala debido a la complejidad de la operación.



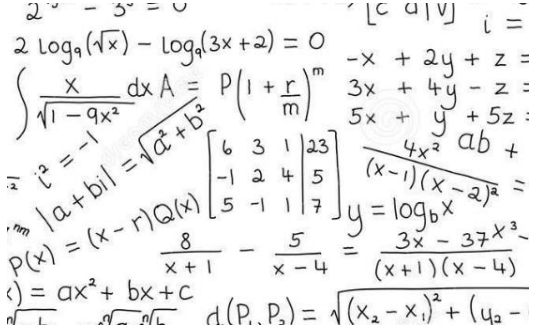
KIBERNUM

Enfoque Algorítmico

Dado el escenario planteado anteriormente, se hace necesario encontrar otras aproximaciones para llegar a la solución del volumen. En este sentido, los enfoques algorítmicos son capaces de entregarnos una buena solución a este problema. Tal vez no es la solución exacta y precisa, pero se acerca bastante y con un costo computacional escalable a grandes volúmenes de datos.

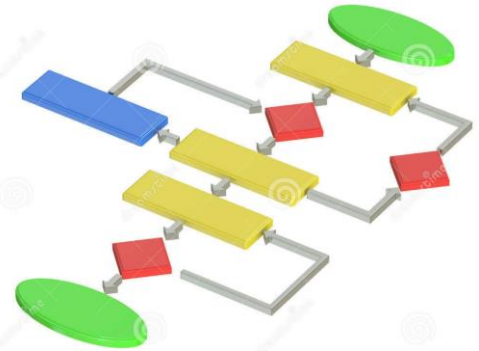
A continuación, para resolver este problema regresivo, formularemos un algoritmo llamado: **Descenso de Gradiente.**

Enfoque Matemático



Resuelve el problema de forma exacta pero **computacionalmente costoso**

Enfoque Algorítmico



Resuelve el problema con una buena aproximación, pero **computacionalmente escalable**

KIBERNUM

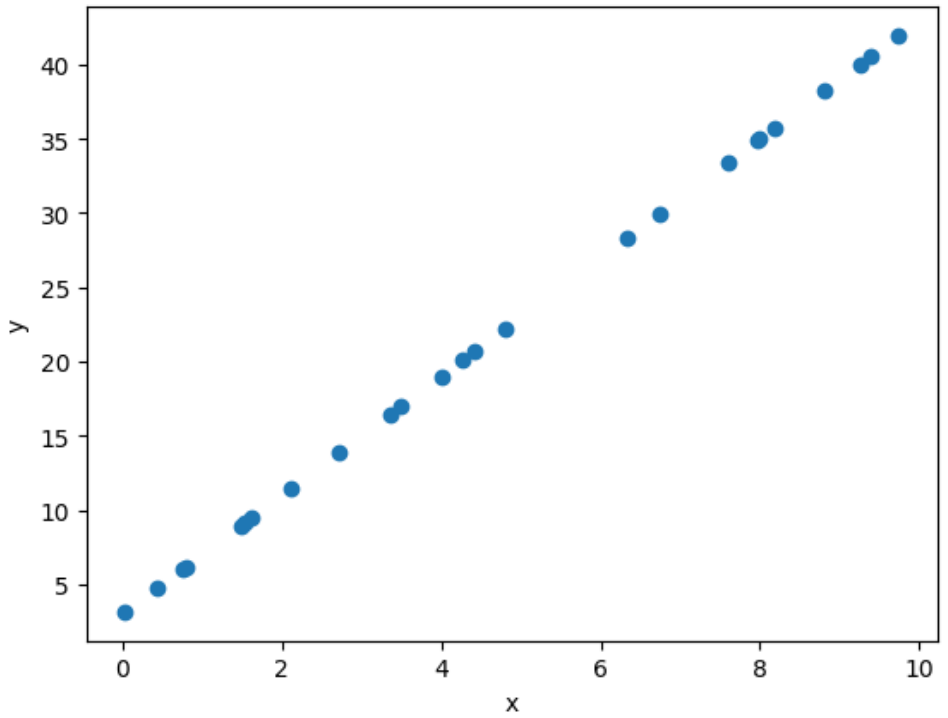
Algoritmo de Descenso de Gradiente

Hipótesis

- Para este ejemplo, consideraremos un conjunto de mediciones perfectamente lineales, los cuales fueron generados de forma artificial con una pendiente de 4 y un intercepto 3, sólo para mostrar cómo funciona el algoritmo.
- Nuestra hipótesis de trabajo es que:

$$y = \theta_0 b + \theta_1 x$$

- Por lo tanto, la tarea consiste en determinar los valores de θ_0 y θ_1 que minimizan el error cuadrático medio. Es decir, vamos a resolver un problema de minimización.



KIBERNUM

Función de Costo

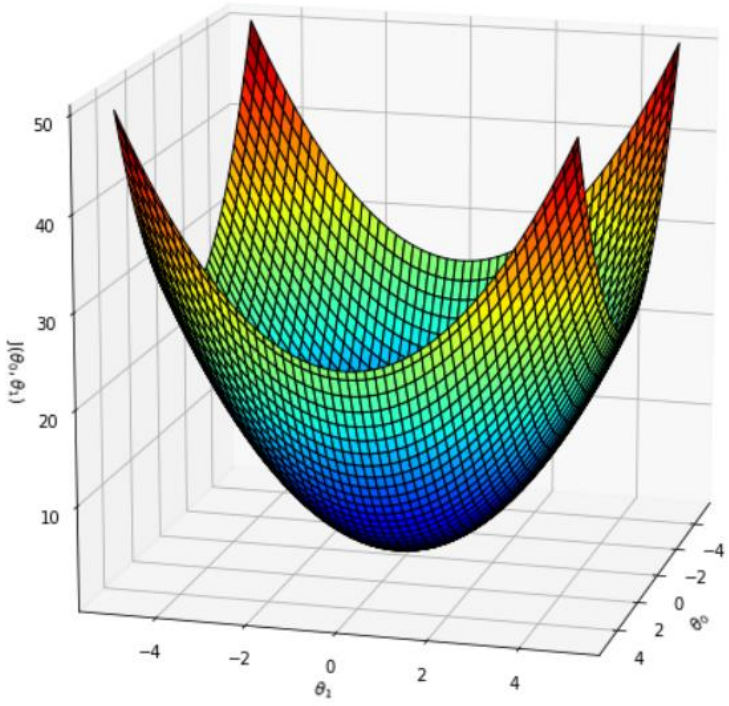
- La función de costo (también llamada Función de Pérdida), el cual denotaremos por $J(\theta)$, es una medida de error que representa el nivel de inexactitud entre las etiquetas actuales y respecto a las etiquetas predichas y_{pred}. La función de costo es elegida de forma que tenga un mínimo global único (es decir, una función convexa).

- En el caso de una regresión lineal, una buena decisión es considerar, como función de costo, el error cuadrático medio (MSE). Esta función de costo, depende de los parámetros θ_0 y θ_1.

$$J(\theta) = MSE = \frac{1}{2m}\sum_{i=1}^{m}(y_{pred}^{(i)} - y^{(i)})^2$$

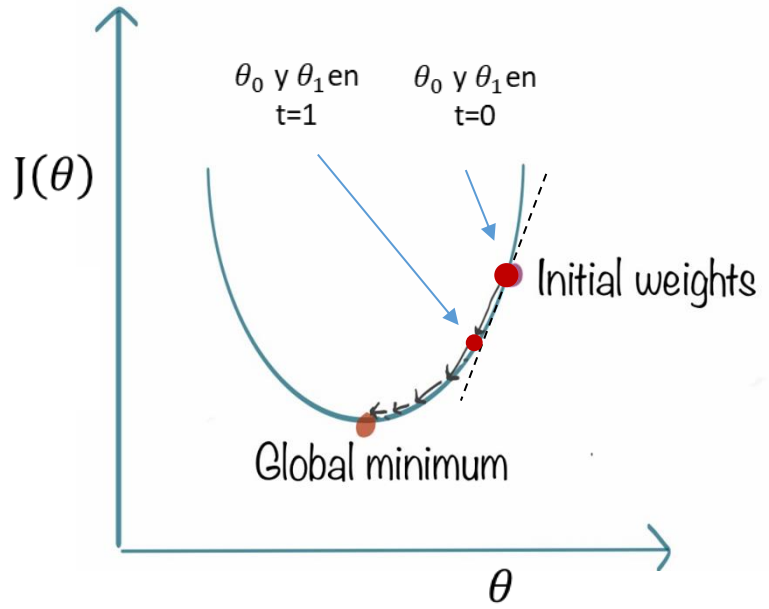
En donde,

$$y_{pred}^{(i)} = \theta_0 b + \theta_1 x^{(i)}$$



KIBERNUM

Algoritmo de Optimización



Programaremos un algoritmo de optimización para acercarnos de forma iterativa a la combinación de θ_0 y θ_1 que minimizan la función de costo $J(\theta)$.

El primer paso será definir valores iniciales para θ_0 y θ_1, los cuales serán aleatorios.

Después, calcularemos la derivada (gradiente) de $J(\theta)$y haremos un pequeño avance en sentido contrario, de forma de ir descendiendo.

El gradiente es un vector de derivadas parciales respecto a θ.

$$\nabla J(\Theta) = \frac{\partial}{\partial \Theta} J(\Theta) = \left(\frac{\partial J}{\partial \Theta_0}, \frac{\partial J}{\partial \Theta_1}\right)$$

KIBERNUM

Algoritmo de Optimización

El algoritmo consiste en calcular nuevos valores de θ_0 y θ_1 a partir de los valores anteriores, avanzando en sentido opuesto a la derivada parcial, en una proporción α .

repetir hasta(convergencia) {

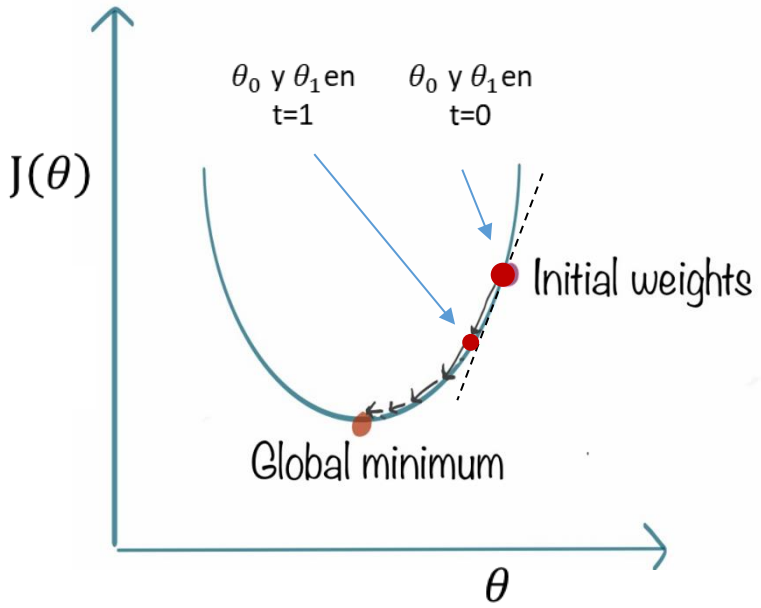
$$\Theta_0^{nuevo} := \Theta_0 - \dot{\alpha} \cdot \frac{\partial J}{\partial \Theta_0}$$

y,

$$\Theta_1^{nuevo} := \Theta_1 - \dot{\alpha} \cdot \frac{\partial J}{\partial \Theta_1}$$

}

Finalmente, el algoritmo termina cuando se ha alcanzado el mínimo global, o bien, cuando ya se ha cumplido una cantidad determinada de iteraciones.



KIBERNUM

Implementación Python Gradient Descend

A continuación, se presenta una implementación del algoritmo Gradiend Descend.

Evalúa el modelo lineal a partir de valores de θ, para un determinado set de datos X, retornando y_{pred}

A partir de valores de θ, para un determinado set de datos X e y, retorna la perdida (MSE)

A partir de valores de θ, para un determinado set de datos X e y, y con un determinado learning rate, calcula los nuevos valores de θ.

X e y son arreglos numpy

```python
# implementación en python

# calcula el valor predicho y
def h(theta, X):
  h = theta[0]  + theta[1] * X
  return h;

# definimos la funcion de costo en base al mse
def cost(theta, X, y):
  m = len(y)
  mse = 1 / (2*m) * sum((h(theta, X) - y)**2)
  return mse


def gradient_descent(theta, X, y, alpha=0.01):

  # nuevo theta, actualizado
  new_theta = [0,0]

  # cantidad de instancias
  m = len(y)

  new_theta[0] = theta[0] - alpha * 1/m * sum( h(theta, X) - y )
  new_theta[1] = theta[1] - alpha * 1/m * sum( (h(theta, X) - y) * X )

  return new_theta
```

KIBERNUM

Implementación Python Gradient Descend

Ahora, definimos la parte principal del programa:

Inicialización aleatoria de los valores de θ.

Definición de learning rate

Definición de la cantidad de iteraciones

En cada iteración almacenamos el costo

```
# inicializacion aleatoria de theta
theta = [0,0]

# learning rate
alpha = 0.01

# iteraciones
epochs = 10000

costs = []
for i in range(epochs):
  theta = gradient_descent(theta, X, y, alpha)
  loss = cost(theta, X, y)
  costs.append(loss)

print("Loss:", costs[-1], "Theta:", theta)
```

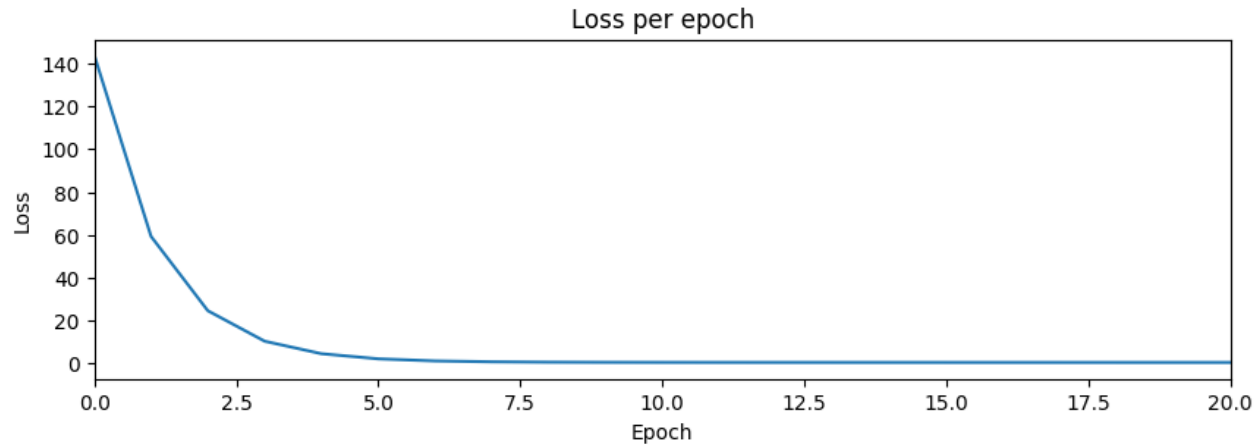
Iteración y recálculo de los valores de θ hasta cumplirse la cantidad de epochs definidas.

Implementación Python Gradient Descend

El resultado obtenido, después de 10000 iteraciones, para los valores θ_0 y θ_1, son los siguientes:

```
print("Loss:", costs[-1], "Theta:", theta)
```

```
Loss: 2.2883506074299733e-15 Theta: [2.9999998351530475, 4.000000025478936]
```



Nótese que nos acercamos bastante a los valores definidos arbitrariamente al inicio de esta presentación para generar el set de datos artificiales (intercepto=3, pendiente=4). En la siguiente figura, se aprecia cómo va disminuyendo el costo (mse) a medida que avanzan las iteraciones.

Resumen y Conclusiones

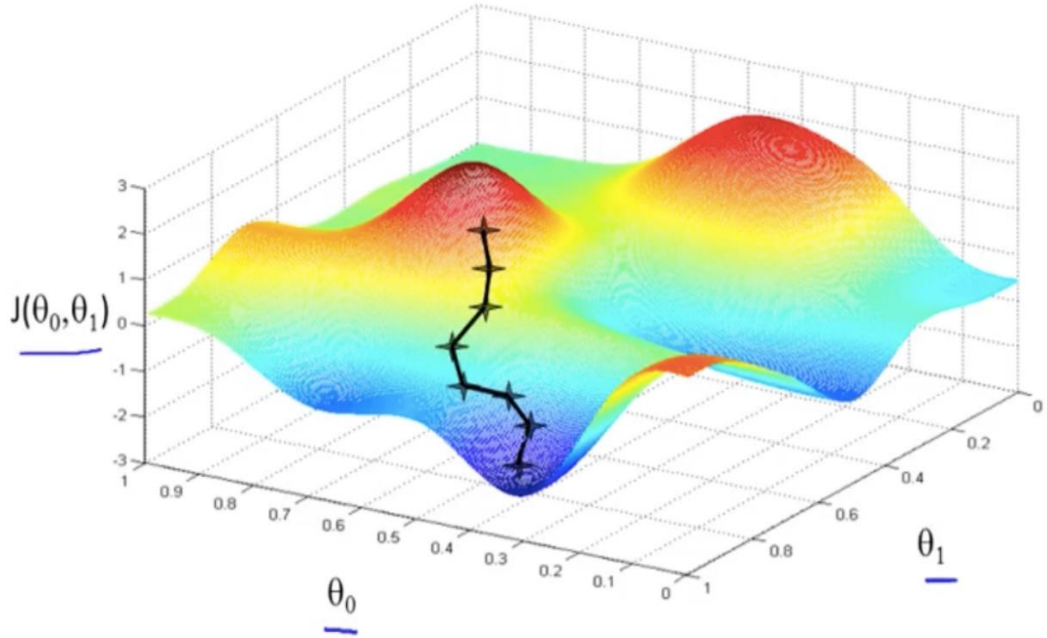
En esta presentación, hemos resuelto un problema regresivo con un enfoque algorítmico. Este mismo enfoque servirá, más adelante, para resolver otros tipos de problema (por ejemplo, problemas de clasificación) y conocer otros tipos de algoritmos (por ejemplo, redes neuronales artificiales), pues, utilizan los mismos principios.

Hoy aprendimos:

- Notación matricial para plantear problemas regresivos.
- Enfoque algorítmico para resolver tareas regresivas.
- Algoritmo Descenso de Gradiente.
- Hipótesis.
- Función de Costo.
- Learning rate.
- Epochs.

Resumen y Conclusiones

- El algoritmo de descenso de gradiente es un método de optimización utilizado para minimizar una función de costo o error en un modelo de aprendizaje automático.
- El algoritmo funciona calculando el gradiente de la función de costo con respecto a los parámetros del modelo y ajustando los parámetros en la dirección del gradiente negativo para minimizar la función de costo.
- En otras palabras, el algoritmo de descenso de gradiente busca el mínimo de una función de costo moviéndose en pequeños pasos en la dirección opuesta del gradiente de la función. El tamaño de los pasos se determina por un parámetro llamado tasa de aprendizaje, que controla qué tan grandes son los cambios en los parámetros en cada paso.



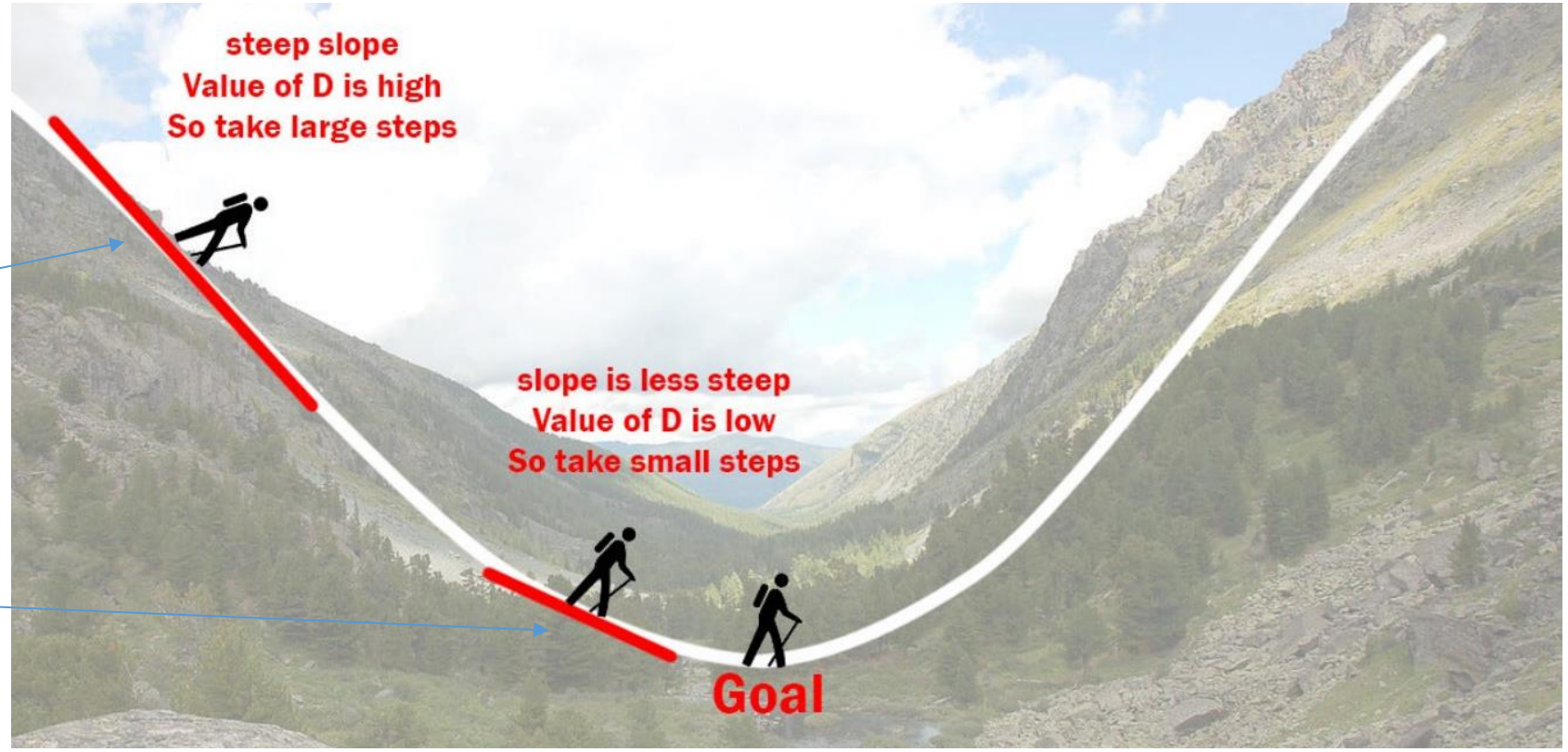
KIBERNUM

Resumen y Conclusiones

El algoritmo de descenso de gradiente es ampliamente utilizado en el aprendizaje automático para entrenar modelos de regresión y clasificación, así como en otras aplicaciones donde es necesario optimizar una función de costo.

Pendiente es mayor, por lo tanto, los pasos son más largos

Pendiente es mayor, por lo tanto, los pasos son más pequeños



KIBERNUM

Dudas y consultas

Fin de la Presentación

KIBERNUM