

Entrenamiento de modelos

**Universidad Nacional de Colombia
2024**

Carlos Daniel jimenez

Contenido

- Modelos de regresión Lineal
- Gradiente descendiente
- Curvas de Aprendizaje
- Regularización de modelos lineales

Bibliografía Recomendada

- Linear Algebra and Optimization for Machine Learning
- Gradient Descent Algorithm – a deep dive
- Gradient Descent video
- Gradient Descent video II
- Learning Curves
- Learning Curves II

Modelos de regresión Lineal

Modelos de regresión Lineal

- Los modelos del tipo de regresión lineal tienen la siguiente forma

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

Donde :

$\theta_0 + \theta_1$ -> Son los parámetros del modelo

- La función lo que se encarga de hacer una suma ponderada de las características de entradas más la intersección

Modelos de regresión Lineal

- Los modelos del tipo de regresión lineal tienen la siguiente forma

$$\hat{y} = h_{\theta}(x) = \theta x$$

Donde

$h_{\theta}(x)$ -> Función de hipótesis

θ -> Parámetros del modelo

x -> vectores de características

- Por lo tanto el entrenamiento del modelo debe radicar en establecer parámetros para que un modelo se adapte a un conjunto de datos

Modelos de regresión Lineal

- Para lo anterior es mejor trabajar con la **función de costo** la cual se encarga de minimizar los errores del modelo.
- La función de costo debe verse de la siguiente manera (por conveniencia de la optimización)

$$MSE(X, h\theta) = \frac{1}{m} \sum (\theta_{x^{(i)}}^t - y^i)^2$$

Donde :

$\theta_{x^i}^t$ -> es el parámetro a optimizar para mejorar el modelo

Modelos de regresión Lineal

- Para encontrar los valores óptimos al modelo de regresión lineal sin necesidad de iterar se puede trabajar con la **Normal Equation**

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

Examples: $m = 4$.

	x_0	Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	y
1	1	2104	5	1	45	460
1	1	1416	3	2	40	232
1	1	1534	3	2	30	315
1	1	852	2	1	36	178

$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$ $m \times (n+1)$

$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$ m -dimensional vector

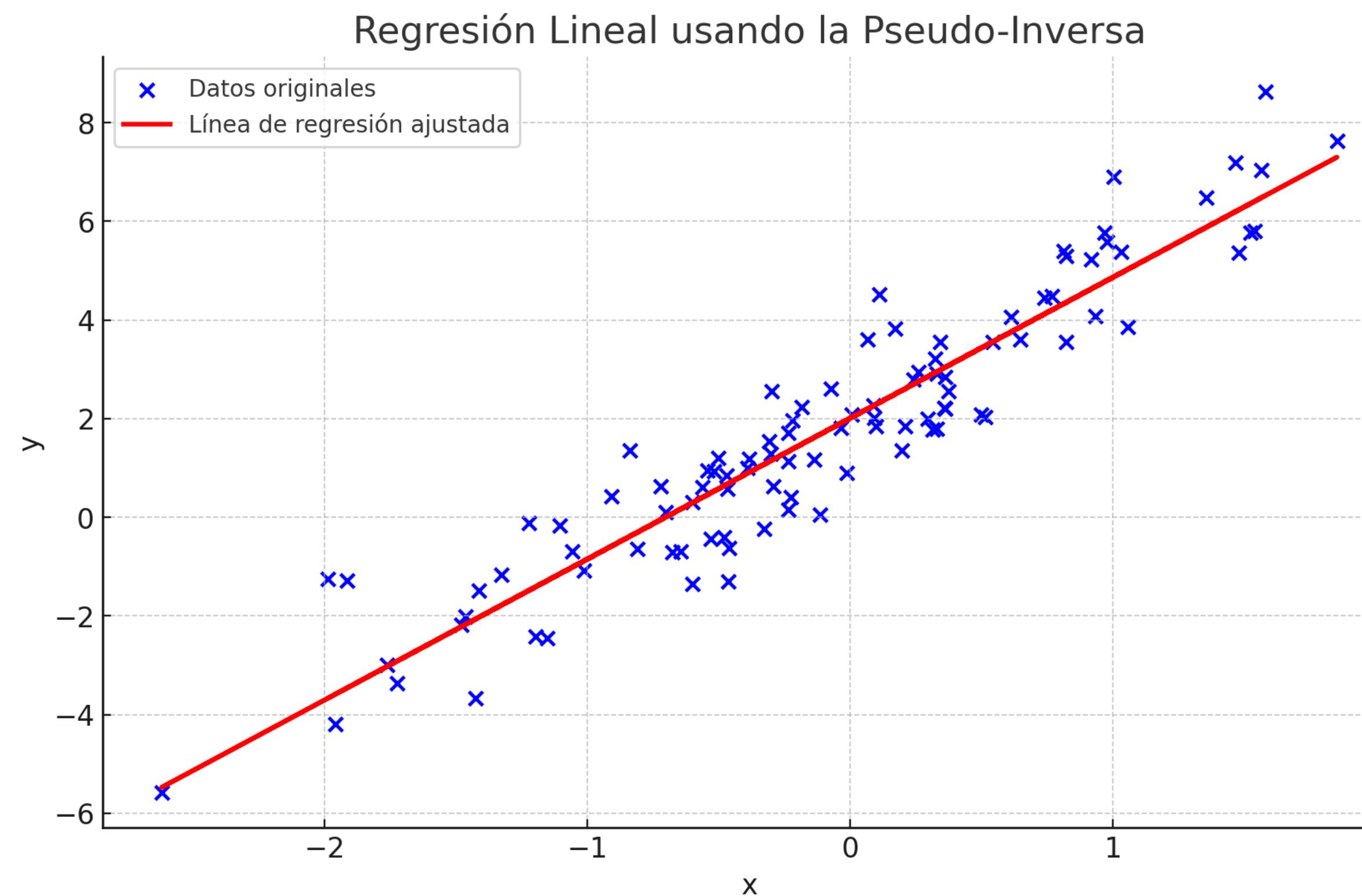
$\theta = (X^T X)^{-1} X^T y$

Ejemplo: Coursera Curso Machine Learning

Modelos de regresión Lineal

Otra manera de encontrar una solución a la optimización es usando la seudo inversa , donde se descomponen los valores en un conjunto de entrenamiento x , en una multiplicación de matrices donde a los valores pequeños se les asigna el valor cero , a los valores mayores a cero se les reemplaza con su inverso y se transpone la matriz resultante -> esto ofrece un excelente manejo a los valores atípicos.

Modelos de regresión Lineal



```
import numpy as np
import matplotlib.pyplot as plt

# Generar datos de ejemplo
np.random.seed(42)
x = 2 * np.random.rand(100, 1)
y = 4 + 3 * x + np.random.randn(100, 1) # Relación lineal con ruido

X_b = np.c_[np.ones((100, 1)), x]

# Calcular los coeficientes usando la ecuación normal (pseudo-inversa)
theta_best = np.linalg.pinv(X_b).dot(y)

# Obtener los valores predichos
y_pred = X_b.dot(theta_best)

# Gráfica de los datos originales y la línea de regresión ajustada
plt.figure(figsize=(10, 6))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred, color='red', label='Línea de regresión ajustada', linewidth=2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regresión Lineal usando la Pseudo-Inversa')
plt.legend()
plt.grid(True)
plt.show()
```

Código disponible en la clase 6 : script svd

Gradiente descendiente

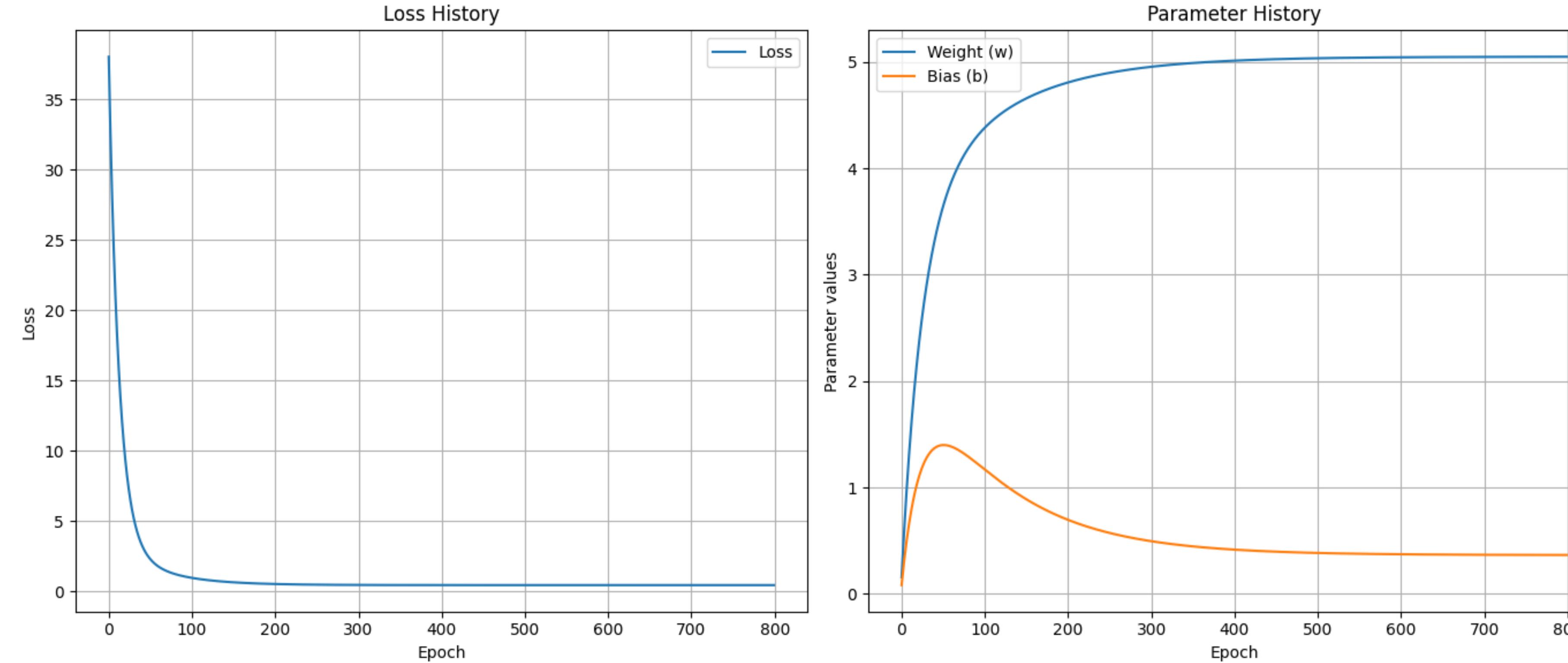
Gradiente descendiente

- Es una forma de ajustar los parámetros del modelo de forma iterativa , **minimizando la función de costo** , es super útil si se trabaja con MLflow
- Mide el gradiente local con respecto al vector de parámetros theta y busca un lugar y una forma en que este vaya descendiendo (analogía de bajar una montaña)
- Cuando el gradiente alcanza el valor cero es que ha alcanzado su mínimo global

Gradiente descendiente

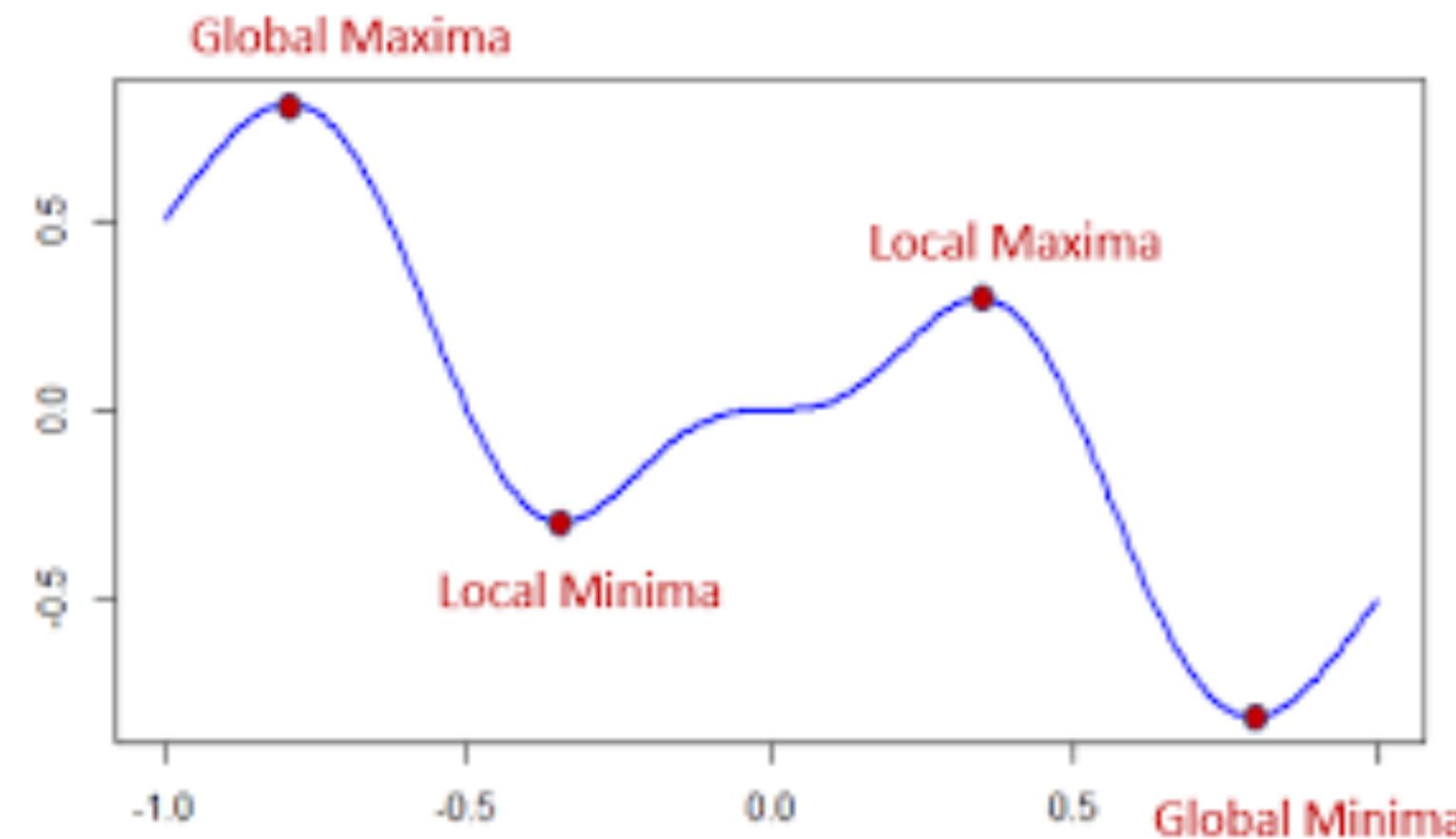
- Por si algo ... un **gradiente** es un vector que indica **la dirección y la magnitud del cambio más rápido en la función de costo** o pérdida con respecto a los parámetros del modelo.
- El gradiente empieza calculando de manera aleatoria con valores al azar, pero poco a poco va alterándolos , buscando una mejora entre los mismos y su relación con la optimización de la función de costo -> a esto se le conoce como actualización

Gradiente descendiente



Código disponible en el script llamado `gradient_descent.py`

Gradiente descendiente

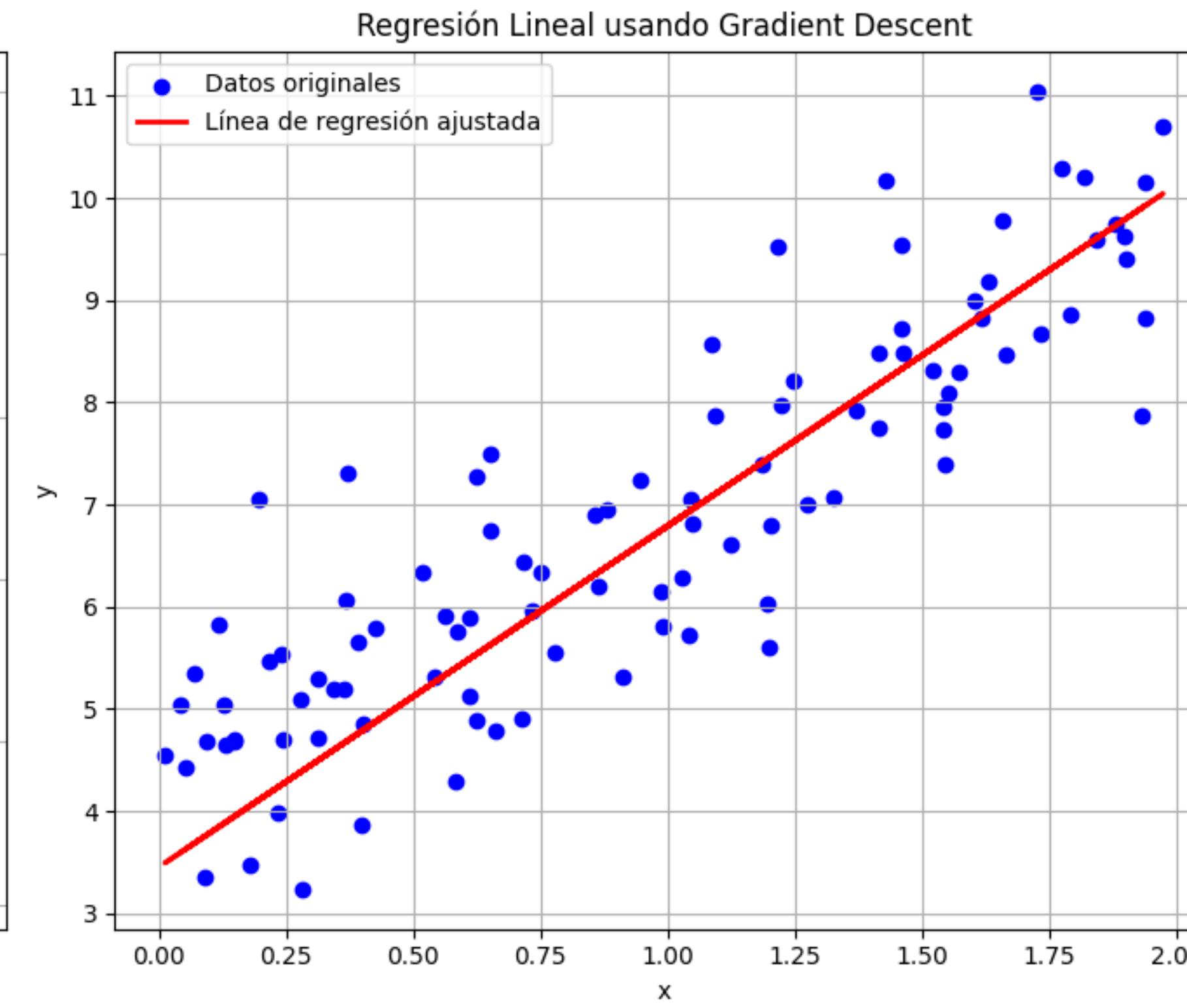
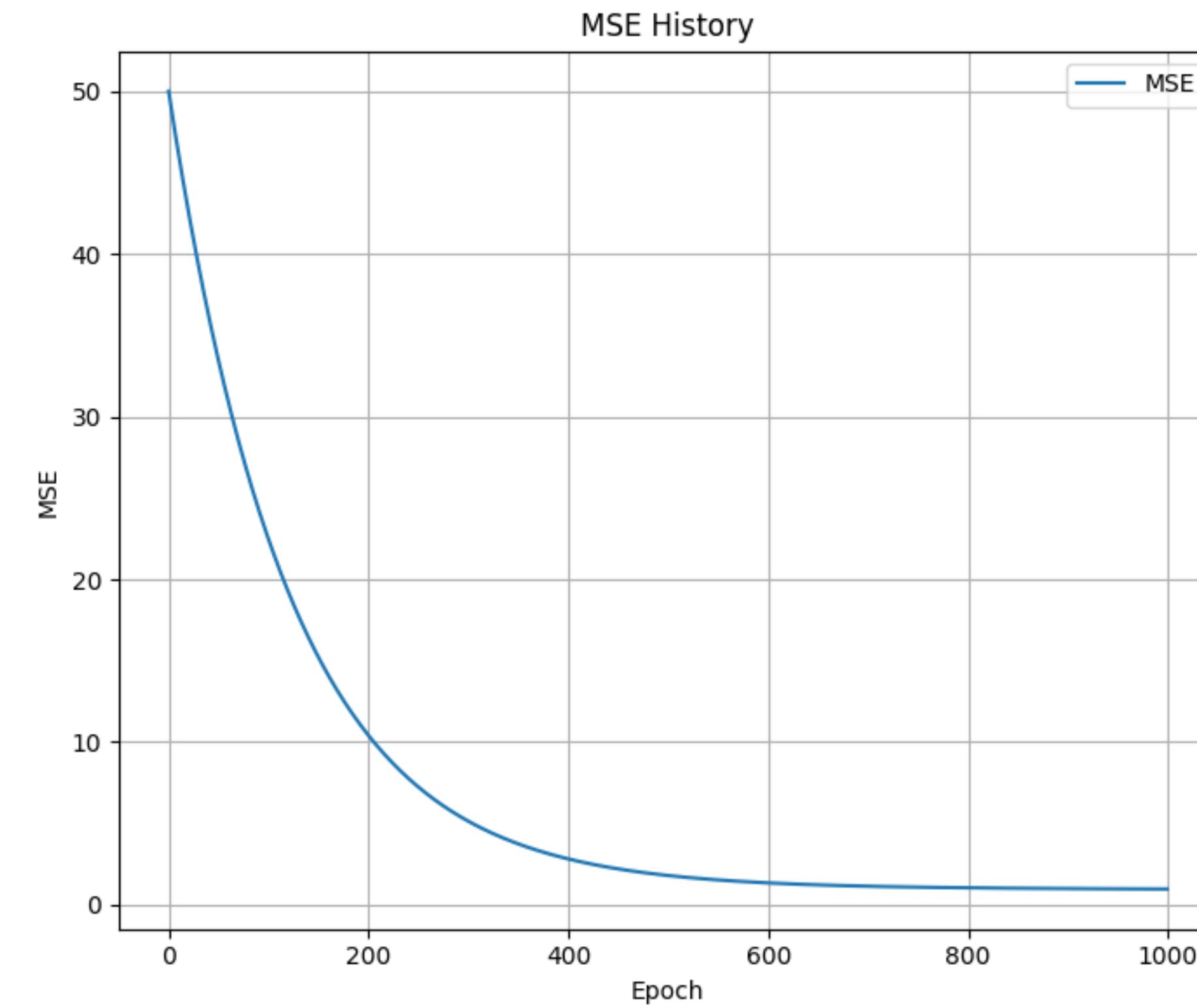


Un problema con esto, son los mínimos locales

Por lo tanto es mejor trabajar con el MSE que es
Una función convexa, o sea no tiene min locales

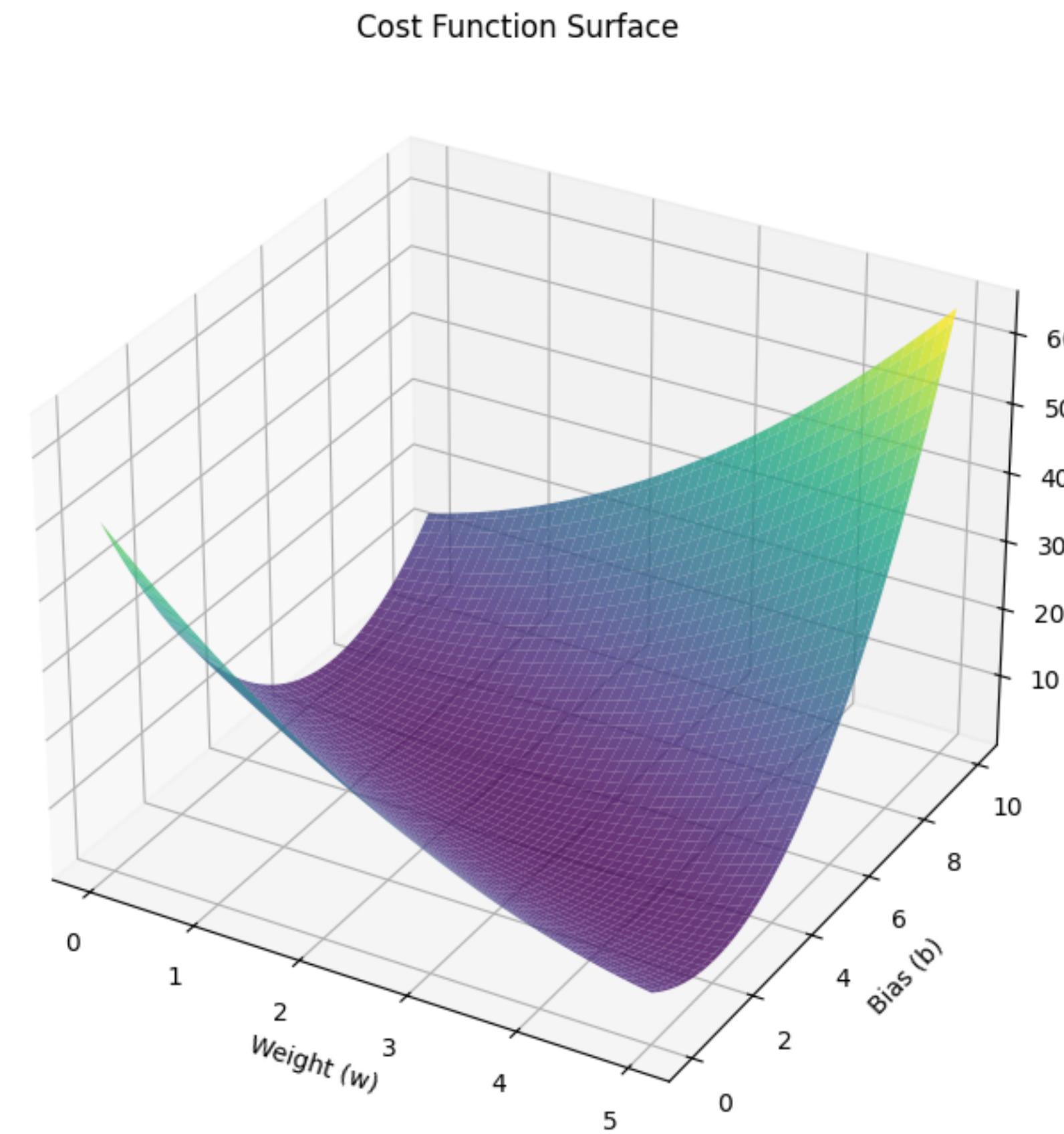
Gradiente descendiente

$$y=4+3x$$



Código script: mse.py

Gradiente descendiente



Código script: mse.py

Stochastic Gradient Descent

- SGD actualiza los parámetros del modelo usando un solo ejemplo de entrenamiento a la vez.
- Al hacer esto el algoritmo puede que encuentre un mínimo global dado un lote de datos
- Este tipo de metodología tiene un problema y es que puede que no encuentre el mínimo global dado un lote al azar de los datos, dada la falta de representación de los datos
- Una solución a esto tiene que ver con el manejo de la tasa de aprendizaje

Batch Gradient Descent

- Uso todos los datos del train para encontrar los mejores parámetros del modelo
- Esto suele ser muy costoso computacionalmente , por lo tanto es útil cuando no son muchos datos
- Para encontrar un learning rate adecuado puede ser necesario utilizar un grid search

Mini Batch Gradient Descent

- Es un mix entre SGD y batch descent
- Usa pocos datos de los batches para calcular la mejor versión de la función de costo
- Los modelos se vuelven eficientes en términos computacionales

Polynomial Feature

- Eleva al cuadrado las características de entrenamiento para un modelo
- Esto incrementa la capacidad de un modelo para encontrar mejores relaciones entre los datos

```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, SplineTransformer

We start by defining a function that we intend to approximate and prepare plotting it.

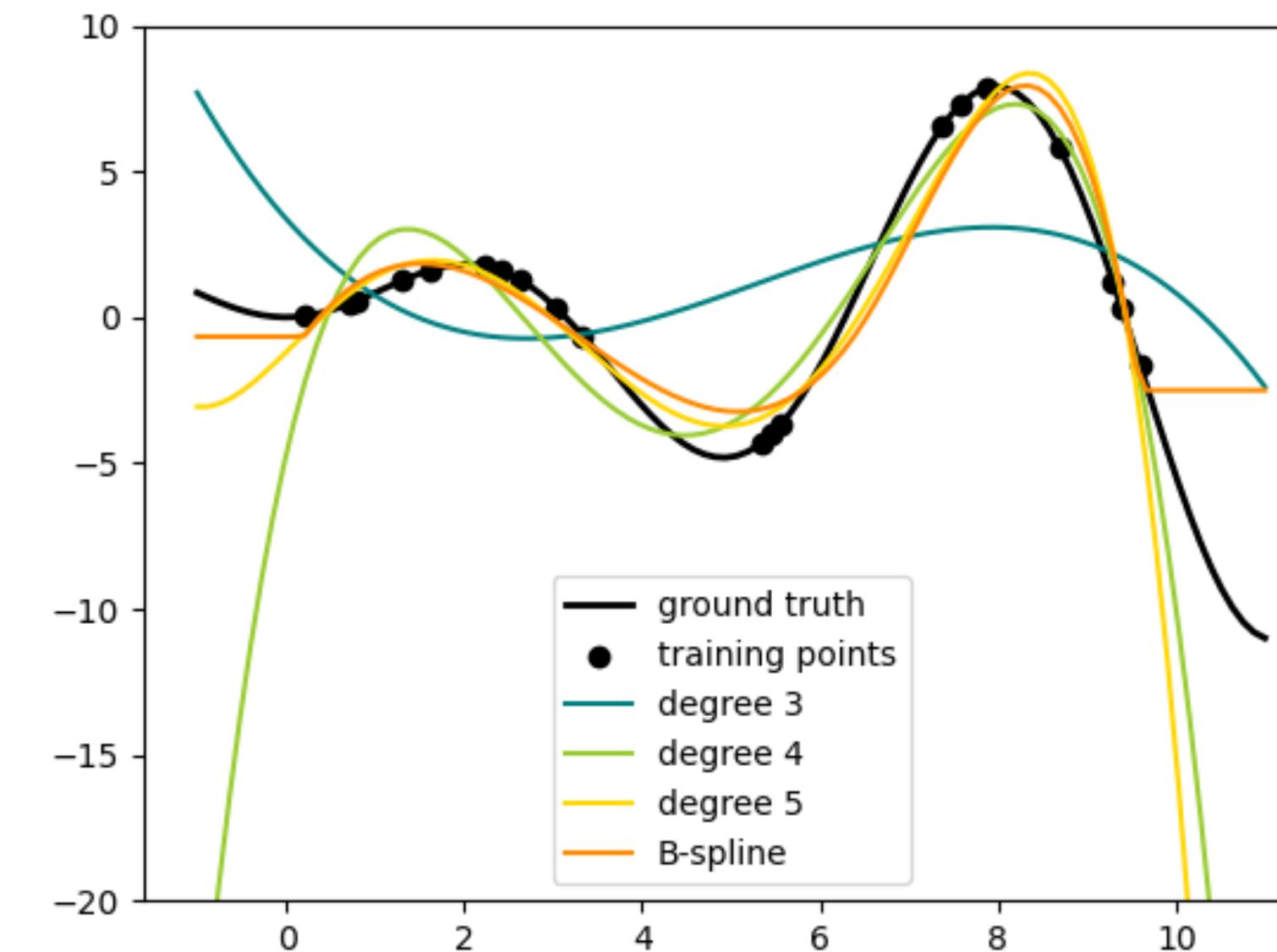
def f(x):
    """Function to be approximated by polynomial interpolation."""
    return x * np.sin(x)

# whole range we want to plot
x_plot = np.linspace(-1, 11, 100)

To make it interesting, we only give a small subset of points to train on.

x_train = np.linspace(0, 10, 100)
rng = np.random.RandomState(0)
x_train = np.sort(rng.choice(x_train, size=20, replace=False))
y_train = f(x_train)

# create 2D-array versions of these arrays to feed to transformers
X_train = x_train[:, np.newaxis]
X_plot = x_plot[:, np.newaxis]
```



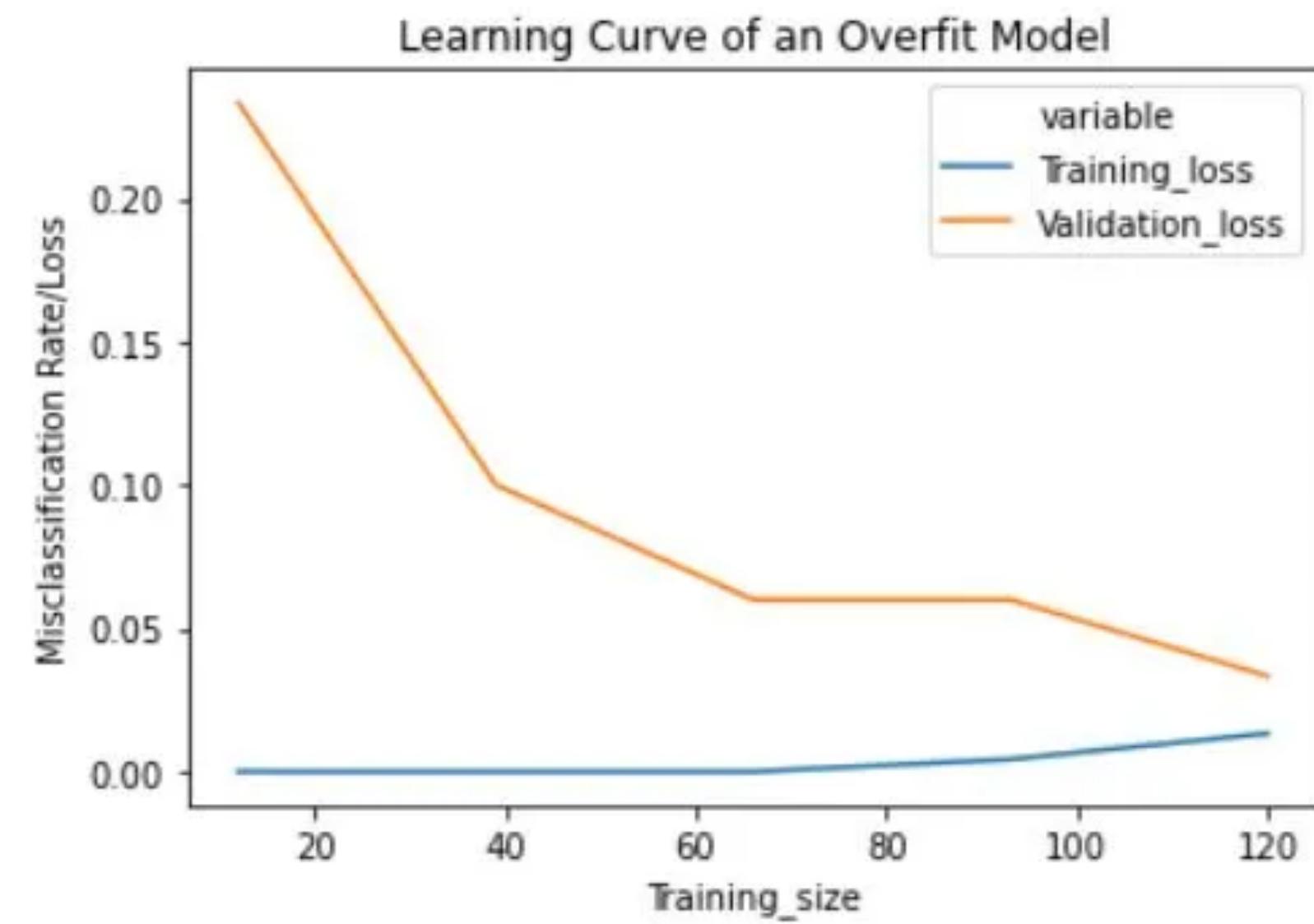
Curvas de aprendizaje

Curvas de aprendizaje

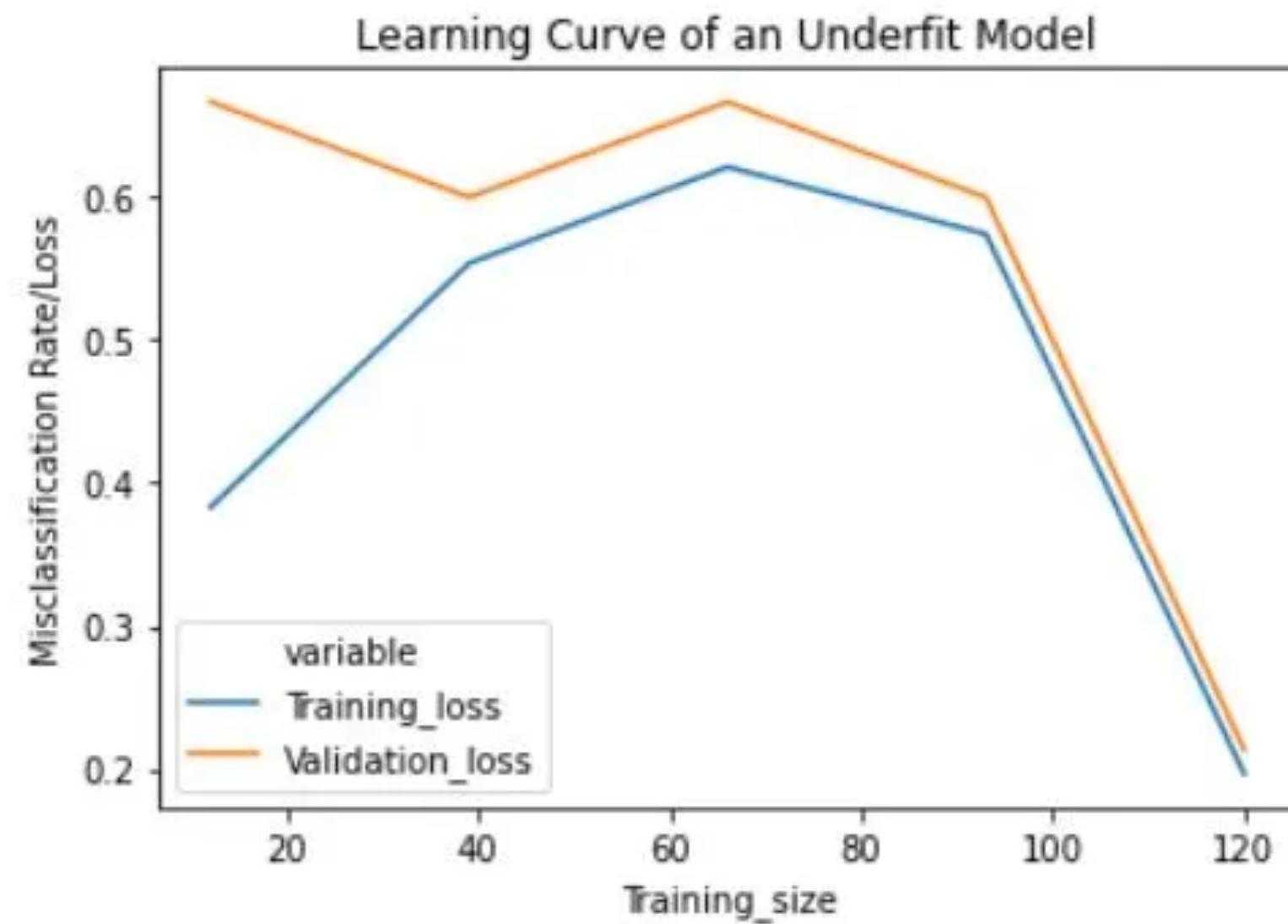
- Son utiles para saber si un modelo esta sobre ajustado o por si el contrario esta por debajo de sus capacidades
- Estas son gráficos del error de entrenamiento y el error de la validación en función de la interacción del entrenamiento.
- Underfitting : Tanto el valor del error de entrenamiento como el de validación son altos -> Esto indica que el modelo es muy simple y por lo tanto no se soluciona con más datos
- Overfitting : El error de entrenamiento es muy bajo mientras que el error de validación es alto -> este error se puede corregir con mejores y más datos

Curvas de aprendizaje

- El error de entrenamiento es muy bajo mientras que el error de validación es alto

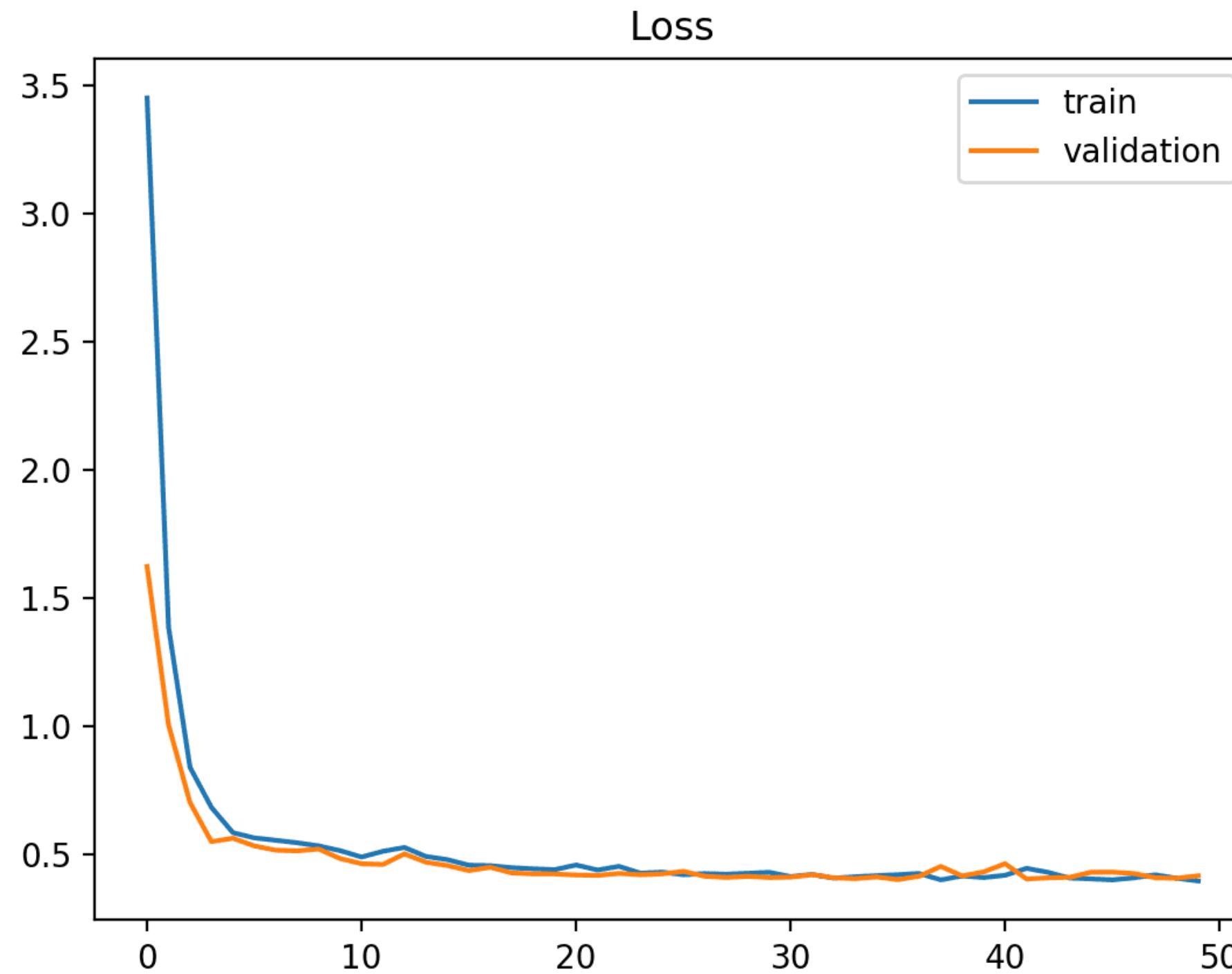


Curvas de aprendizaje



- Tanto el valor del error de entrenamiento como el de validación son altos

Curvas de aprendizaje

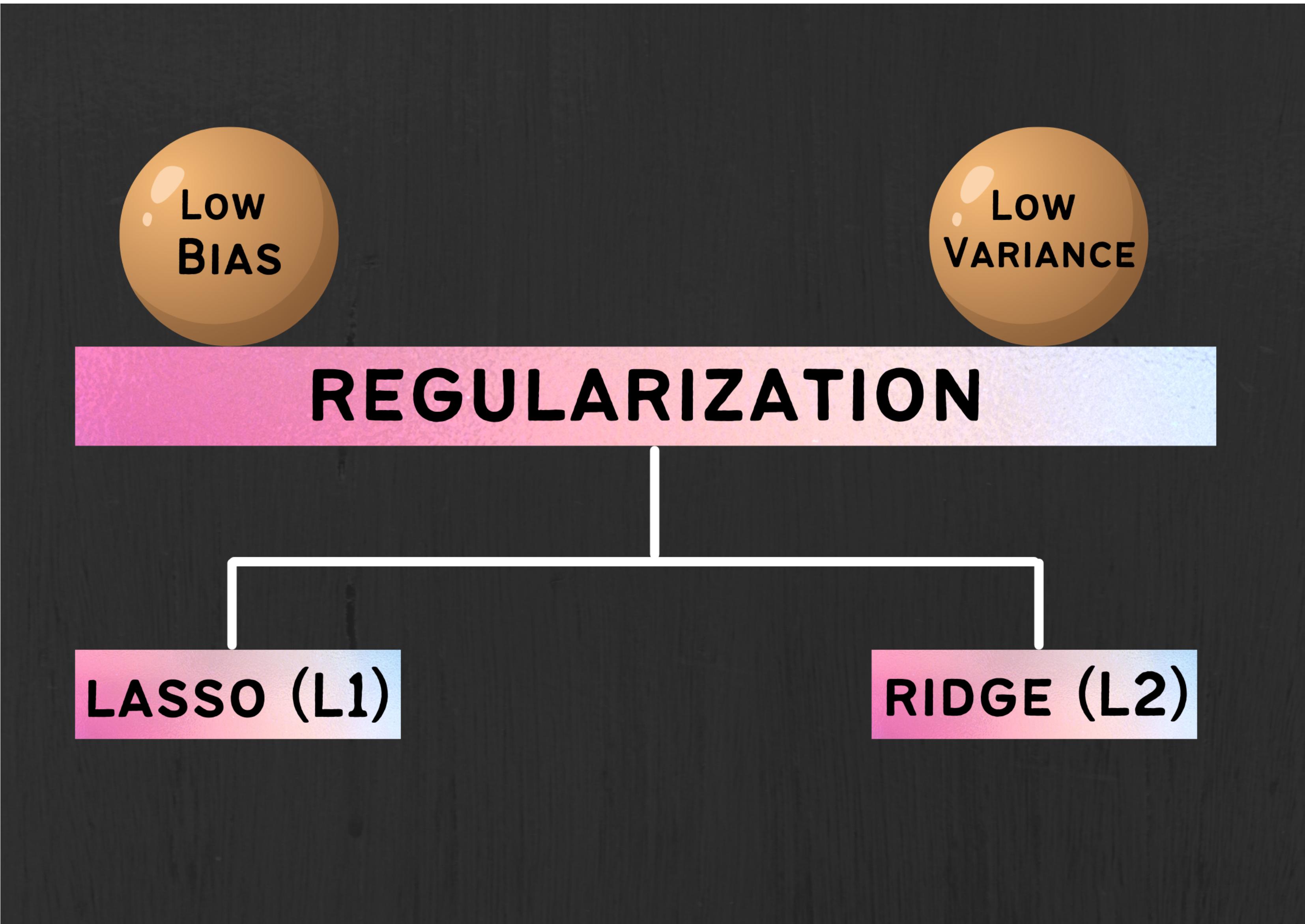


Esto sería un buen ajuste !
Un ajuste ideal para un modelo

Regularización de modelos lineales

Regularización de modelos lineales

- El término de regularización tiene que ver con restringir
- Estas restricciones ayudan a los modelos a enfrentar estos tres problemas :
 - i) Sesgo
 - ii) Varianza
 - iii) Error irreductible
- Estas restricciones tienen mucho que ver con entender los problemas de multicolinealidad



<https://www.analyticsvidhya.com/blog/2021/09/lasso-and-ridge-regularization-a-rescuer-from-overfitting/>

Regularización Ridge

- Ayuda a aislar problemas de correlaciones altas entre variables explicativas

$$J(m) = \sum_{i=0}^n (\hat{y} - y_i)^2 + \lambda (\text{slope})^2$$

Cost Function of Ridge Regression Model

- La anterior formula lo que dice es que la penalización de los parámetros puede acercarse a cero, pero nunca será cero. por lo tanto reduce la capacidad entender la varianza y con ello tambien de resolver el problema de sensibilidad de la misma y mejora la estabilidad numérica

Regularización Lasso

- Ayuda a aislar problemas de correlaciones altas entre variables explicativas

$$J(m) = \sum_{i=0}^n (\hat{y} - y_i)^2 + \lambda |\text{slope}|$$

Cost Function of LASSO Regression Model

- Lo que quiere decir esta formula es que las variables o características irrelevantes les da valor de cero. Esto es super útiles para hacer feature importance

Regularización Lasso

- Ayuda a aislar problemas de correlaciones altas entre variables explicativas

$$J(m) = \sum_{i=0}^n (\hat{y} - y_i)^2 + \lambda |\text{slope}|$$

Cost Function of LASSO Regression Model

- Lo que quiere decir esta formula es que las variables o características irrelevantes les da valor de cero. Esto es super útiles para hacer feature importance

Regularización Early Stopping

- Para el entrenamiento del modelo cuando el error de la validación es cercano a cero

Regresión logistica

- Es un problema de clasificación lineal donde la formula adquiere la siguiente forma

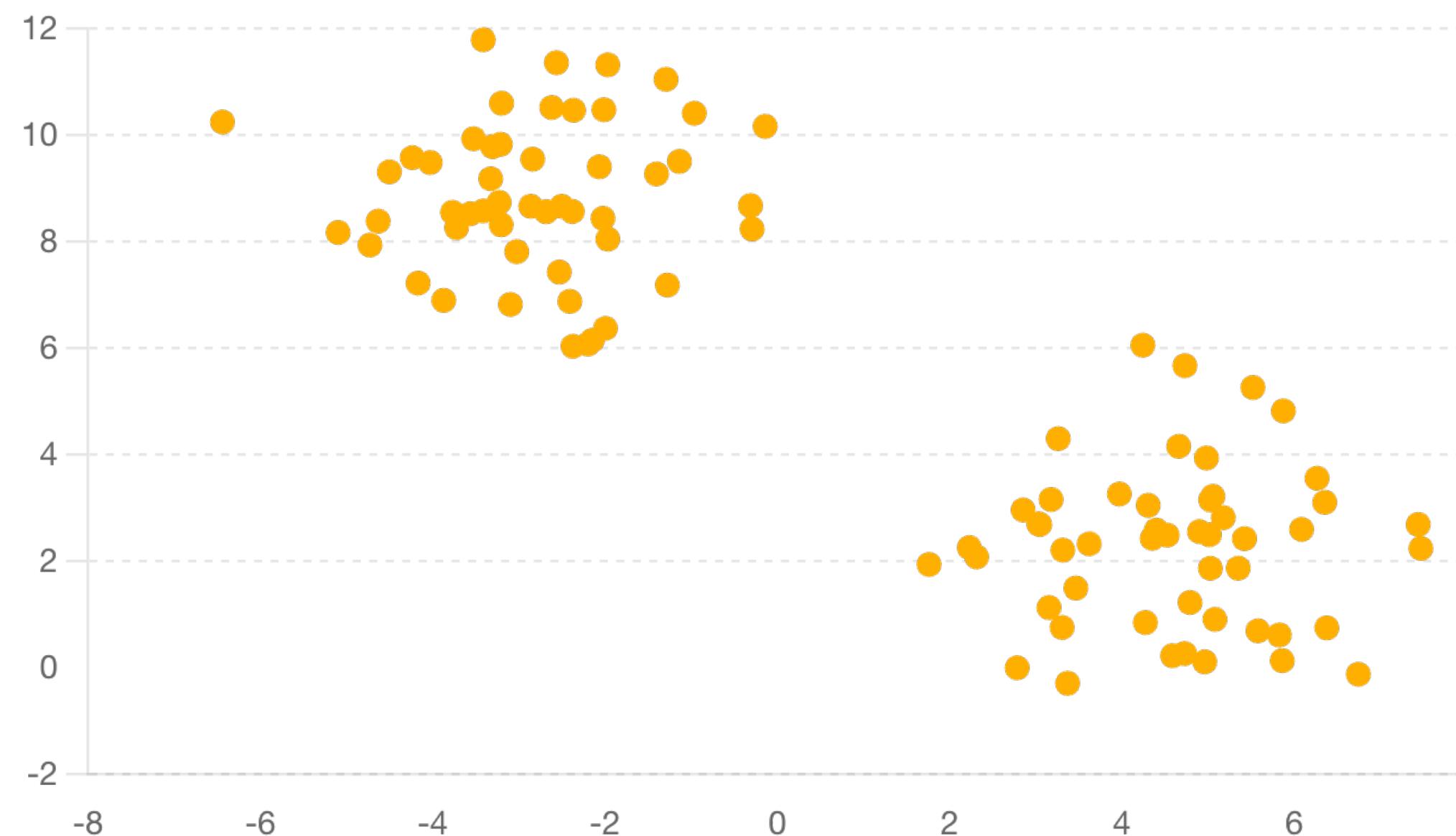
$$f(x) = \frac{1}{1 + e^{-x}}$$

Equation of Logistic Regression

- EL objetivo del entrenamiento es estimar los parámetros que mejoren al modelo desde una función sigmoide y para ello es mejor entender el Decision boundaries

Decision Boundaries

- Son las fronteras que un modelo de clasificación establece para separar diferentes clases en un espacio de características. Básicamente, estos límites determinan en qué categoría se clasifica un nuevo punto de datos.



Softmax Regression

- Es una extensión de la regresión logística que se utiliza para clasificar datos en más de dos categorías. A diferencia de la regresión logística binaria, que predice la probabilidad de pertenecer a una sola clase, la regresión Softmax calcula las probabilidades de que una observación pertenezca a cada una de las posibles clases.

$$z_i = w_i \cdot x + b_i$$

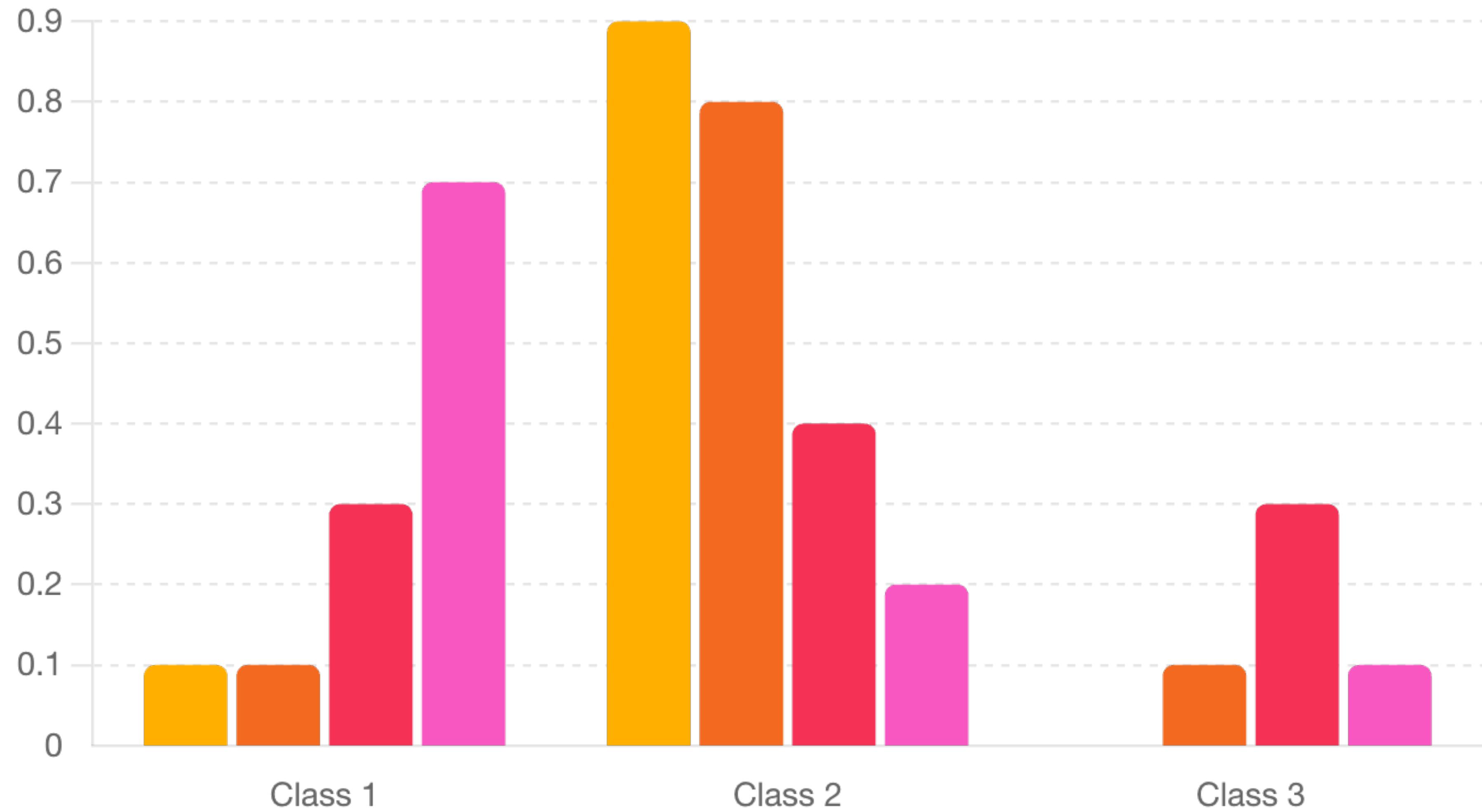
wi son los pesos asociados con la clase i

b_i es el intercepto

X es el vector de entrada

Cross entropy

- Es una función de pérdida utilizada principalmente en problemas de clasificación, especialmente cuando se usa con modelos que producen probabilidades como salida, como la regresión logística o las redes neuronales con una capa de activación Softmax.
- La entropía cruzada mide la diferencia entre dos distribuciones de probabilidad: la distribución verdadera (etiquetas verdaderas) y la distribución predicha por el modelo.



Taller en clase

- Vamos a resolver con lo que vimos en la clase pasada y esta el problema de Kaggle de Calentamiento global

deberá :

Explorar la base de datos

Diseñar un modelo donde se minimice los errores

Argumentar sus respuestas

Tendrá 1 hora para ello -> este es su examen!!!

Regularización de modelos lineales