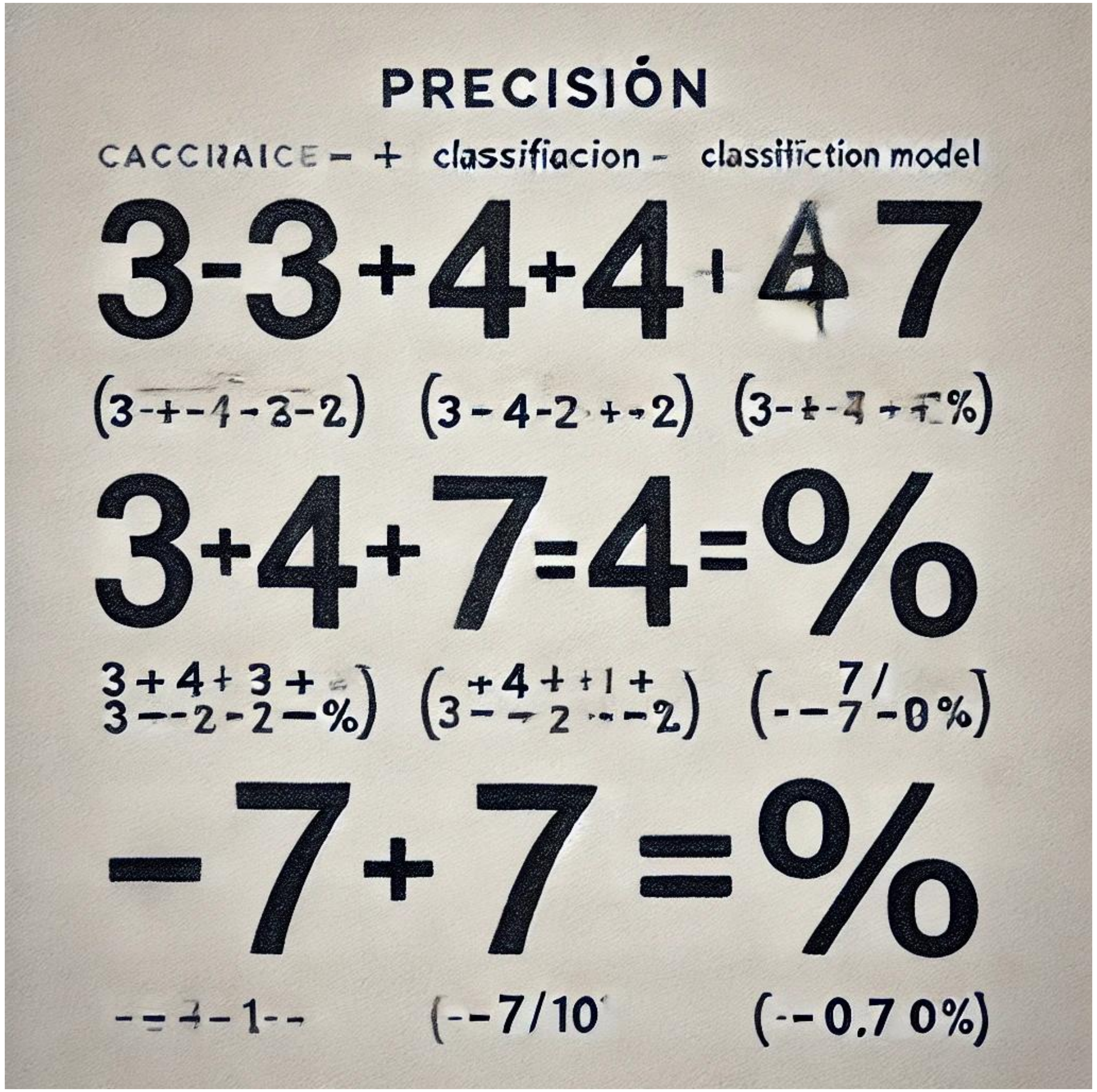
Regresión logística



Para normalizar un conjunto de valores entre 0 y 1 utilizando una "dummy library", puedes hacerlo manualmente o utilizando una librería estándar como `scikit-learn` en Python. A continuación, te muestro cómo hacerlo de ambas maneras:

1. Normalización Manual (Sin Librería)

Puedes normalizar los valores utilizando la fórmula:

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Aquí te muestro cómo hacerlo manualmente:

python Copiar código

```python
def normalize(data):
    min_val = min(data)
    max_val = max(data)
    return [(x - min_val) / (max_val - min_val) for x in data]

# Ejemplo de uso
data = [5, 10, 15, 20, 25]
normalized_data = normalize(data)
print(normalized_data)
```

2. Usando `scikit-learn` (Librería Estándar)

`scikit-learn` tiene una función `MinMaxScaler` que se usa para escalar los valores entre 0 y 1 fácilmente.

python Copiar código

```python
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Ejemplo de datos
data = np.array([[5], [10], [15], [20], [25]])

# Crear el escalador y ajustar los datos
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)

print(normalized_data)
```

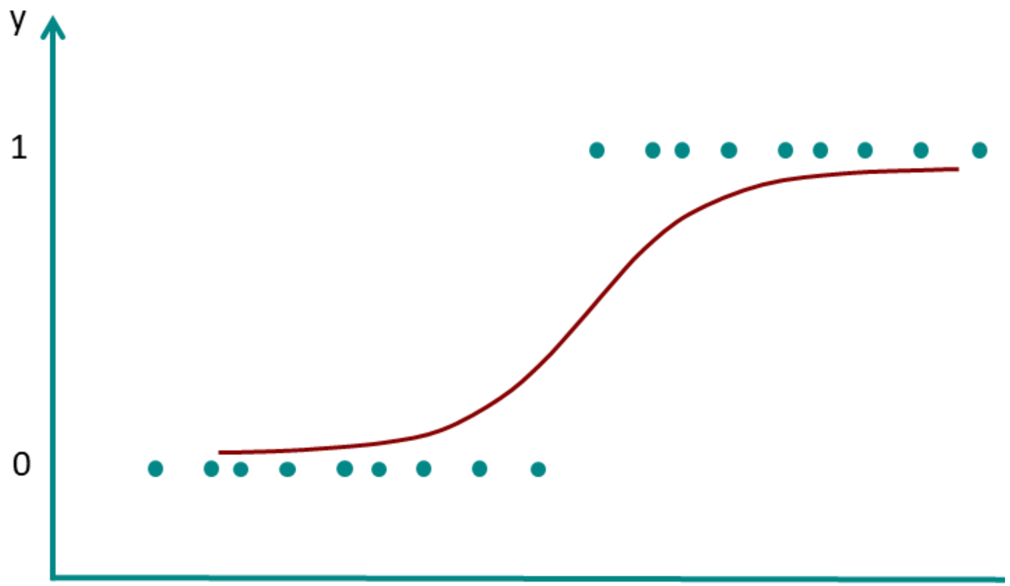
python Copiar código

```python
# Ejemplo de uso
from dummy_library import DummyScaler

data = [5, 10, 15, 20, 25]
scaler = DummyScaler()
normalized_data = scaler.fit_transform(data)

print(normalized_data)
```

- **Normalización Manual**: Realiza la normalización sin usar ninguna librería externa.
- `scikit-learn`: Usa `MinMaxScaler` para normalizar datos de manera eficiente.
- **Librería Dummy**: Simula el comportamiento de `MinMaxScaler` con una implementación personalizada básica.

Regresión logística

La regresión logística es una técnica estadística de machine learning para clasificar los registros de un conjunto de datos, basándose en los valores de los campos de entrada**. En regresión logística, usaremos una o mas variables independientes para predecir un resultado, al cual llamaremos variable dependiente**. La regresión logística es análoga a la regresión lineal pero intenta predecir un campo objetivo categórico o discreto en lugar de uno numérico.



- Predecir la probabilidad de que una persona tenga un ataque al corazón en un periodo especificado de tiempo, basado en nuestro conocimiento de la edad de la persona, sexo, e indice de masa corporal.
- Predecir la probabilidad de mortalidad en un paciente herido, o predecir si un paciente tiene una enfermedad, como la diabetes, basado en las características observadas de ese paciente, como el peso, la altura, la presión sanguínea, y el resultado de varios test de sangre, etc.
- En un contexto de marketing, podemos usarlo para predecir la probabilidad de un cliente de estar pagando o cancelando una suscripción.
- También podemos usar regresión logística para predecir la probabilidad de fallo de un proceso, sistema o producto.
- Podemos precedir la probabilidad del propietario de dejar de pagar la hipoteca.

¿Cuáles son las aplicaciones de la regresión logística?

La regresión logística tiene varias aplicaciones del mundo real en muchos sectores diferentes.

Fabricación

Las empresas de fabricación utilizan el análisis de regresión logística para estimar la probabilidad de fallo de las piezas en la maquinaria. Luego, planifican los programas de mantenimiento en función de esta estimación para minimizar los fallos futuros.

Sanidad

Los investigadores médicos planifican la atención y el tratamiento preventivos mediante la predicción de la probabilidad de enfermedad en los pacientes. Utilizan modelos de regresión logística para comparar el impacto de los antecedentes familiares o los genes en las enfermedades.

Finanzas

Las empresas financieras tienen que analizar las transacciones financieras en busca de fraudes y evaluar las solicitudes de préstamos y seguros en busca de riesgos. Estos problemas son adecuados para un modelo de regresión logística porque tienen resultados discretos, como alto riesgo o bajo riesgo y fraudulento o no fraudulento.

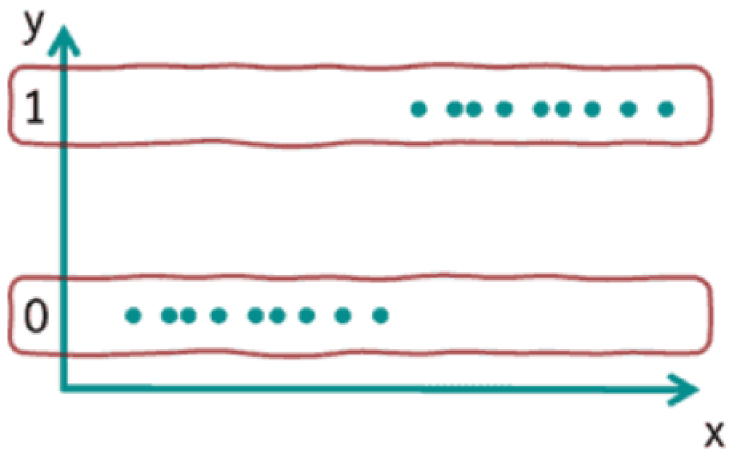
Marketing

Las herramientas de publicidad en línea utilizan el modelo de regresión logística para predecir si los usuarios harán clic en un anuncio. Como resultado, los especialistas en marketing pueden analizar las respuestas de los usuarios a diferentes palabras e imágenes y crear anuncios de alto rendimiento con los que los clientes interactuarán.

Regresión logística

La regresión logística es un **caso especial del análisis de regresión** y se utiliza cuando la variable **dependiente tiene una escala nominal**. Es el caso, por ejemplo, de la variable decisión de compra con los dos valores *compra un producto* y *no compra un producto*.

Con la regresión logística, ahora es posible explicar la variable dependiente o estimar la probabilidad de ocurrencia de las categorías de la variable.



Calcular la regresión logística

Para construir un modelo de regresión logística, se parte de la ecuación de regresión lineal.

Dependent variable

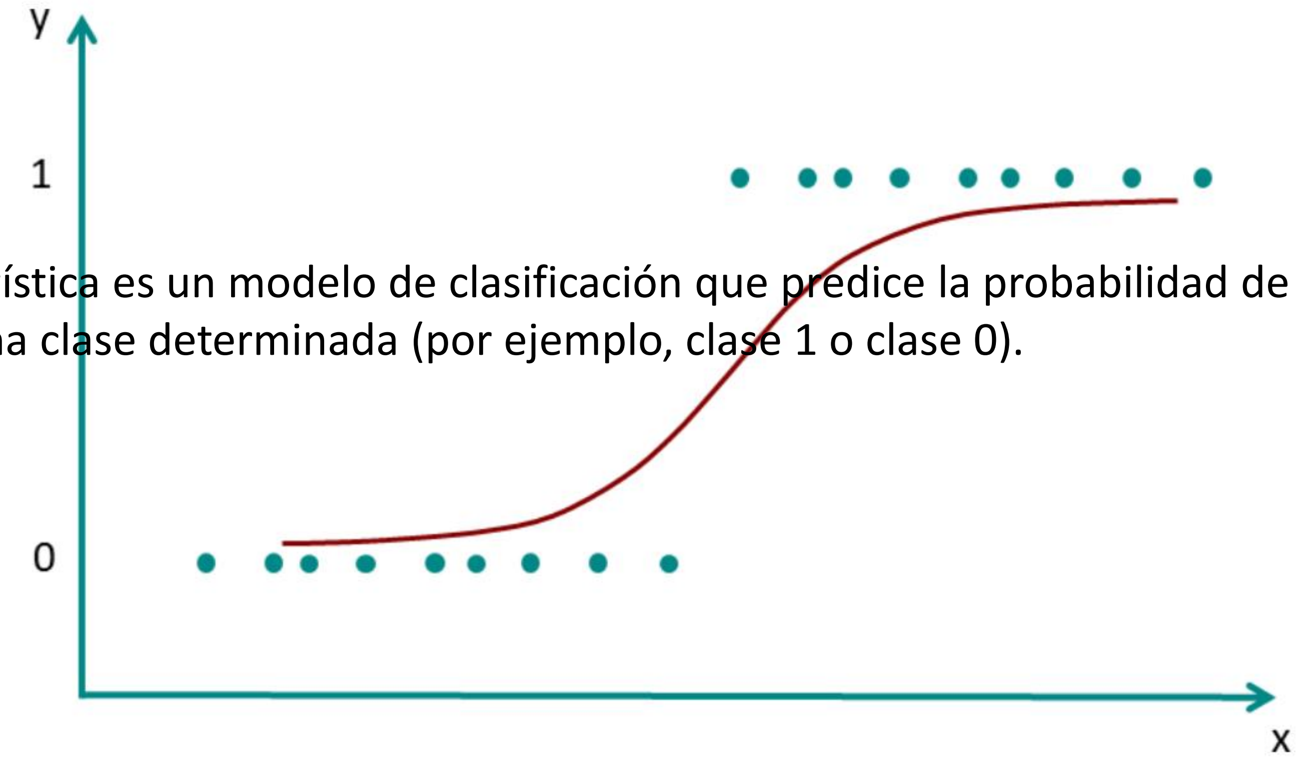
Independent variables

$$\boxed{\hat{y}} = \boxed{b_1} \cdot \boxed{x_1} + \boxed{b_2} \cdot \boxed{x_2} + \ldots + \boxed{b_k} \cdot \boxed{x_k} + \boxed{a}$$

Regression coefficients

$$\hat{y} = \boxed{b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a}$$

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-\left(\boxed{b_1 \cdot x_1 + \ldots + b_k \cdot x_k + a}\right)}}$$



La regresión logística es un modelo de clasificación que predice la probabilidad de que una observación pertenezca a una clase determinada (por ejemplo, clase 1 o clase 0).

La **función sigmoidea** es una función matemática utilizada ampliamente en regresión logística, redes neuronales y otros modelos de aprendizaje automático. Es especialmente conocida por su capacidad para transformar cualquier valor de entrada en un valor de salida que se encuentra en el rango de 0 a 1, lo que la hace ideal para modelar probabilidades.

Fórmula de la Función Sigmoidea

Problema es binario una sola ecuación
Si un cliente es tacaño , exagerado , buena paga , voy a tener una ecuación # ecuaciones= clases

La función sigmoidea estándar se define como:

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

$$P(y = 1 \mid x) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

Regresión logística Simple

Regresión logística múltiple

Donde:

- $w \cdot x$ es el producto punto entre el vector de pesos w y el vector de características x.
- b es el sesgo o intercepto del modelo.

Función Logit:

A veces, la ecuación se expresa en términos de la función logit, que es el logaritmo de las probabilidades (odds):

Probabilidad que sea blanco, bonito

$$\text{logit}(P) = \ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

Probabilidad que sea negro,feo

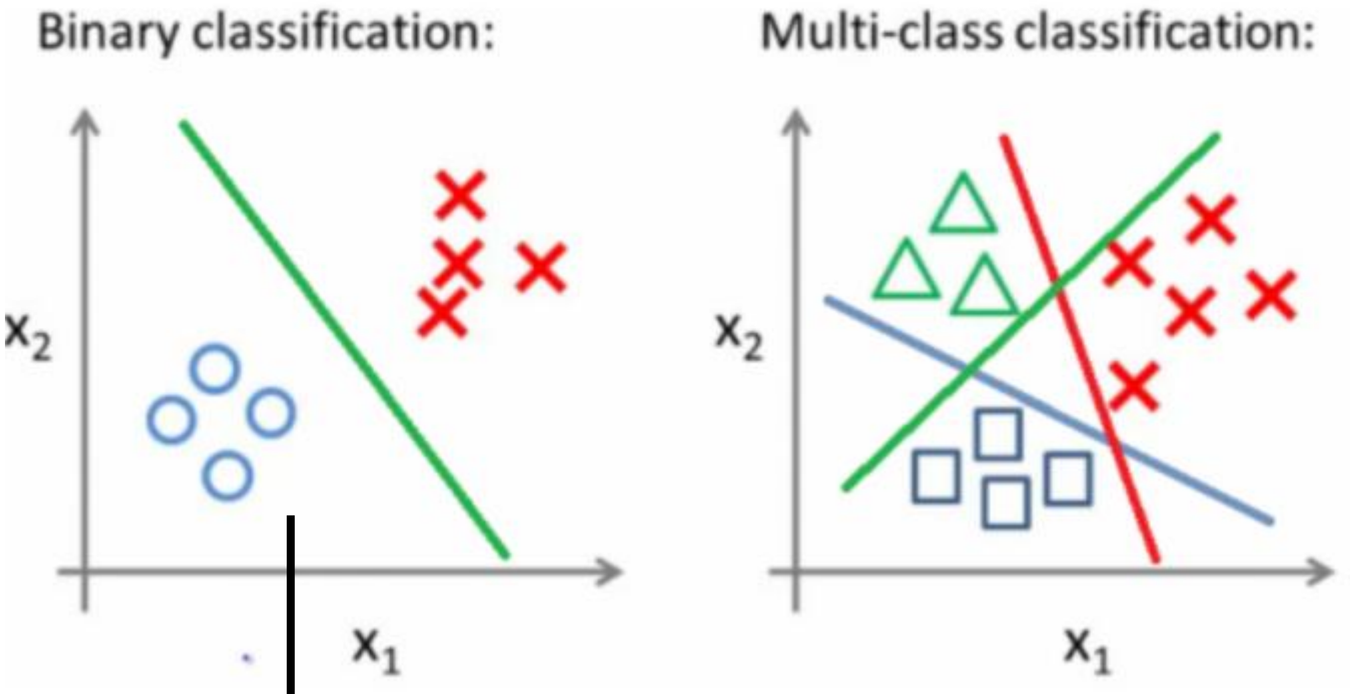
Aquí:

LOGISTICA es por que es una función LOG para volverla una lineal

- $\ln$ es el logaritmo natural.
- P es la probabilidad de que Y sea 1.
- $1 - P$ es la probabilidad de que Y sea 0.

El número de ecuaciones que una regresión logística genera , es igual al número de clases que yo quiero predecir.

$$P(Y=1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n)}}$$



Una clasificación binaria, voy a utilizar una sola regresión logística, una sola ecuación.

$$\boxed{\hat{y}} = \boxed{b_1}\cdot\boxed{x_1} + \boxed{b_2}\cdot\boxed{x_2} + \dots + \boxed{b_k}\cdot\boxed{x_k} + \boxed{a}$$

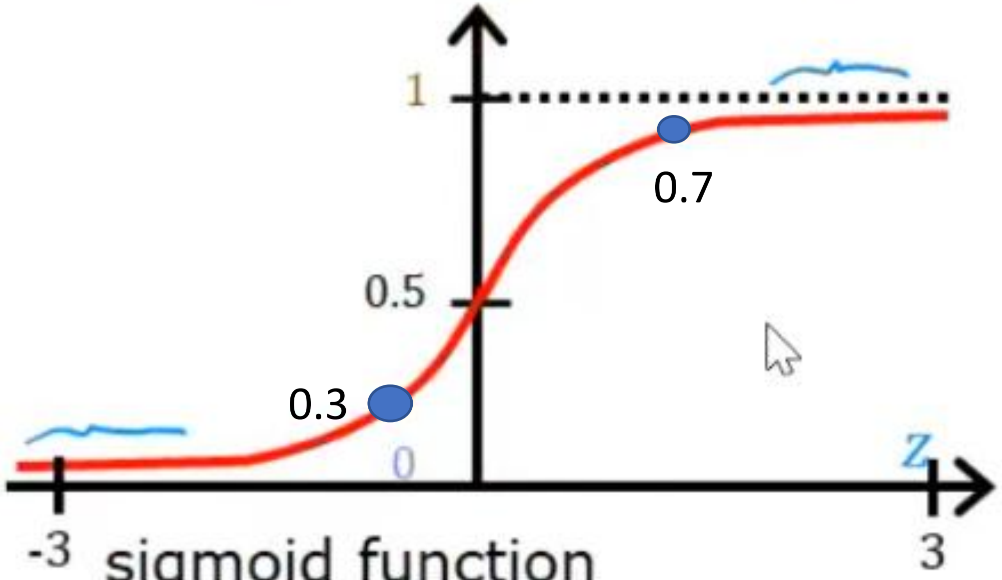
$$\boxed{\hat{y}} = \boxed{b_1}\cdot\boxed{x_1} + \boxed{b_2}\cdot\boxed{x_2} + \dots + \boxed{b_k}\cdot\boxed{x_k} + \boxed{a}$$

$$\boxed{\hat{y}} = \boxed{b_1}\cdot\boxed{x_1} + \boxed{b_2}\cdot\boxed{x_2} + \dots + \boxed{b_k}\cdot\boxed{x_k} + \boxed{a}$$

Diferentes coeficientes diferentes pesos

Want outputs between 0 and 1

Me garantiza que mi modelo va a generar entre 0 y 1



sigmoid function

logistic function

Para generar esta Función:

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

DeepLearning.AI

$$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}}$$

```python
def sigmoid(z):
    """
    Compute the sigmoid of z

    Parameters
    ----------
    z : array_like
        A scalar or numpy array of any size.

    Returns
    -------
     g : array_like
         sigmoid(z)
    """
    z = np.clip( z, -500, 500 )           # protect against overflow
    g = 1.0/(1.0+np.exp(-z))

    return g
```

Z es el parámetro de la función y se espera que sea un array de NumPy o un escalar (un solo número).

La función devuelve g, que es un array del mismo tamaño que z, con los valores transformados por la función sigmoide.

Una distribución de probabilidad, o sea, una estructura tipo vector donde él me va a decir para cada observación. Cuál es la probabilidad entre 0 y 1

Primero, se realiza un recorte de los valores de z con np.clip(z, -500, 500). Esto protege contra el desbordamiento numérico que podría ocurrir

3. Usando `scikit-learn` (`scipy.special.expit`) con Recorte (Clipping)

python Copiar código

```python
from scipy.special import expit
import numpy as np

def sigmoid(z):
    # Recortar (clipping) z al rango [-500, 500]
    z = np.clip(z, -500, 500)
    return expit(z)

# Ejemplo de uso
z = np.array([600, -600, 0, 2, -2, 500, -500])
print(sigmoid(z))
```

Explicación:

- **Recorte (`clipping`)**: Esta técnica asegura que los valores extremos de `z` no causen problemas de desbordamiento cuando se calcule la función exponencial, que es parte de la función sigmoide.

Variable objetivo lo que quiero predecir

1	0.9
0	0.7
1	0.2
0	0.1
1	0.6
1	0.8
1	0.999
0	0.0111
1	0.111

Datos de entrenamiento

0	0.1	acerto	TP
1	0.999	acerto	TP
0	0.001	acerto	TP
0	0.999	erró	TP
0	0.0001	acerto	TP
1	0.0001	Erro	FN

Variable de prueba

RECORDATORIO:
Con los datos de entrenamiento va a aprender a aproximarse a la realidad y luego, con los datos de test, va verificar la realidad con modelo.

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

"logistic regression"

Coeficientes y variables

Y en Resumen ¿ Podría usted explicarlo?

Refresher on logistic regression and decision boundary

- Recall that for logistic regression, the model is represented as

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = g(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \tag{1}$$

where $g(z)$ is known as the sigmoid function and it maps all input values to values between 0 and 1:

$$g(z) = \frac{1}{1+e^{-z}} \tag{2}$$

and $\mathbf{w} \cdot \mathbf{x}$ is the vector dot product:

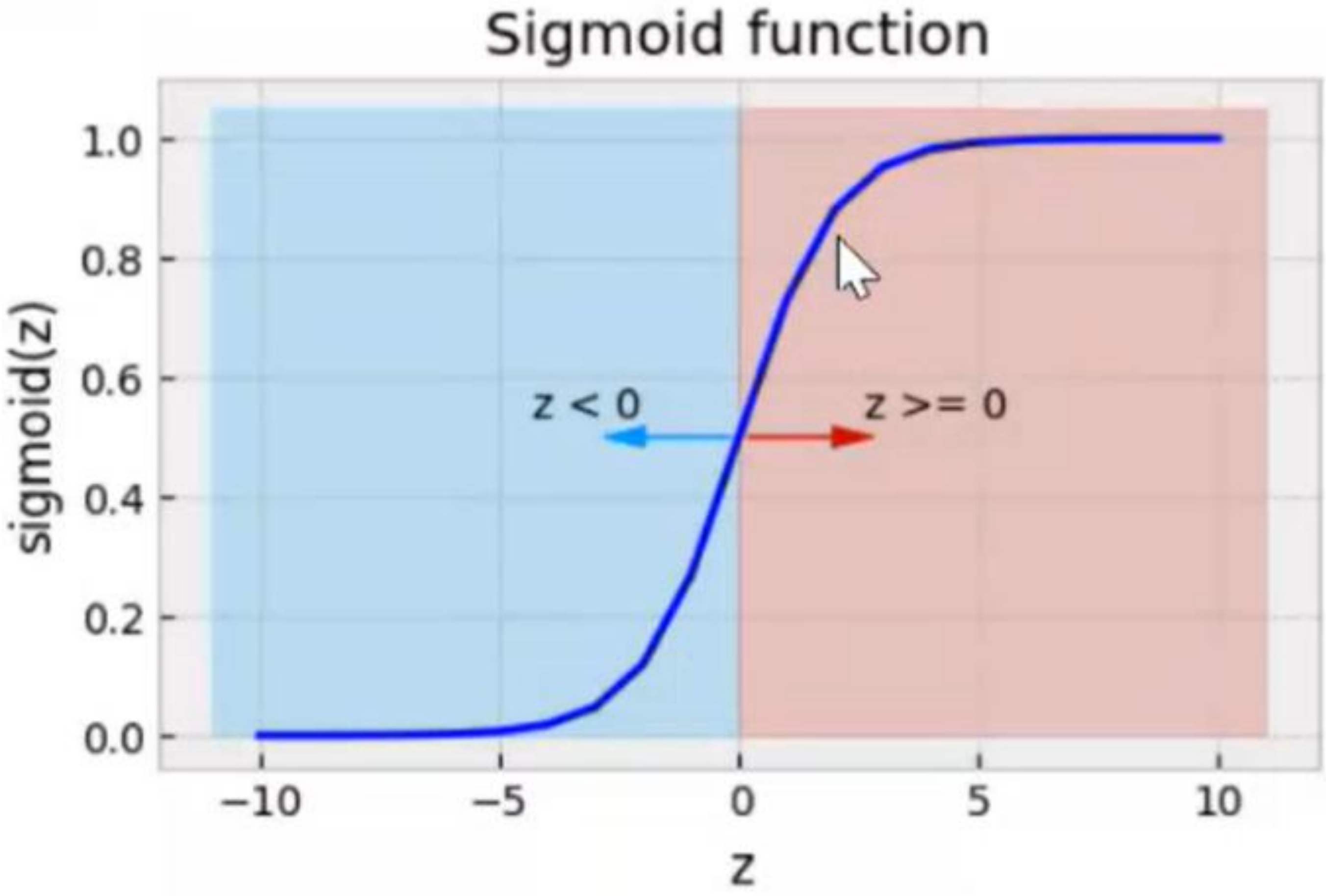
$$\mathbf{w} \cdot \mathbf{x} = w_0 x_0 + w_1 x_1$$

- We interpret the output of the model ($f_{\mathbf{w},b}(x)$) as the probability that $y = 1$ given $\mathbf{x}$ and parameterized by $\mathbf{w}$ and b.
  - Therefore, to get a final prediction ($y = 0$ or $y = 1$) from the logistic regression model, we can use the following heuristic -

if $f_{\mathbf{w},b}(x) >= 0.5$, predict $y = 1$

if $f_{\mathbf{w},b}(x) < 0.5$, predict $y = 0$

- Let's plot the sigmoid function to see where $g(z) >= 0.5$

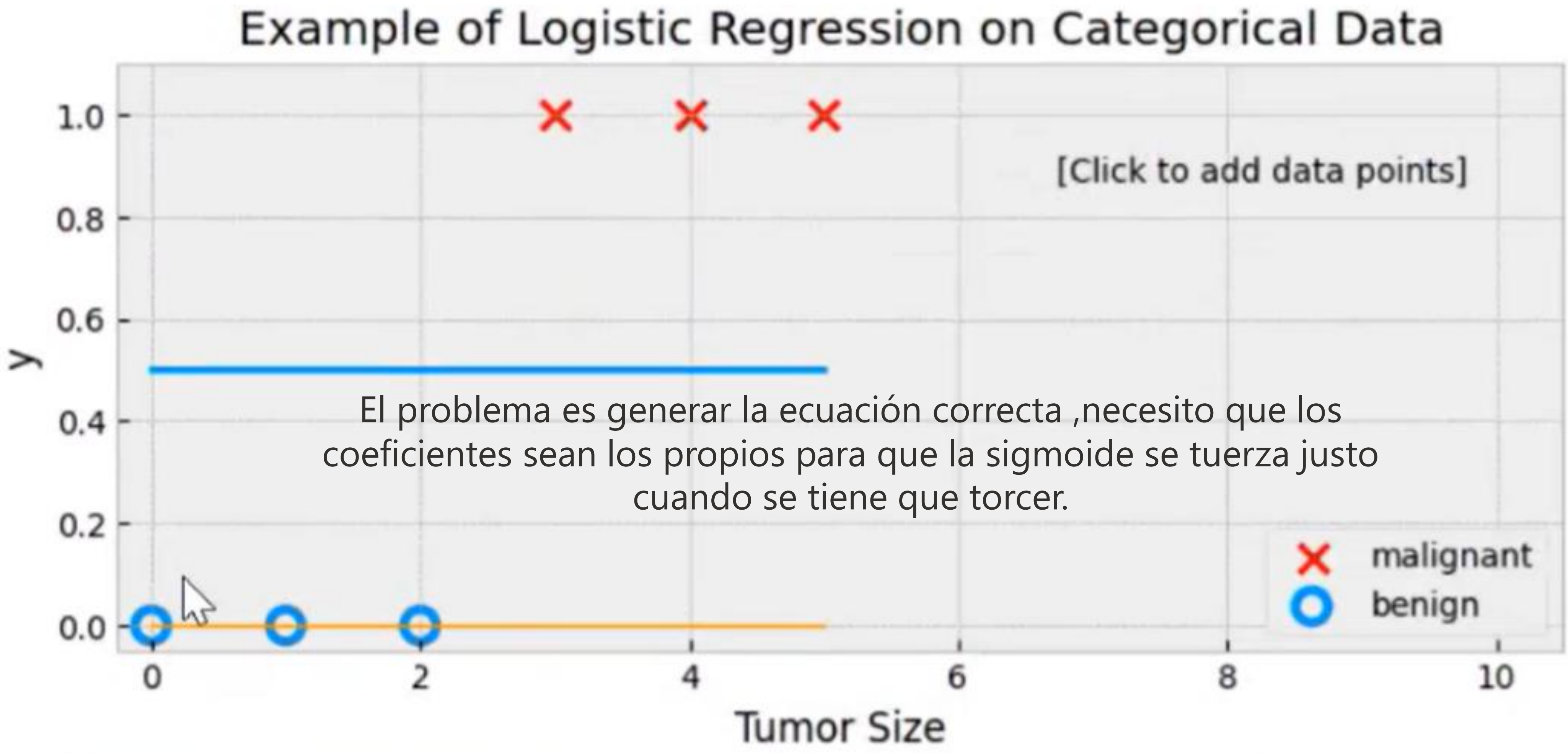


if $f_{\mathbf{w},b}(x) >= 0.5$, predict $y = 1$

if $f_{\mathbf{w},b}(x) < 0.5$, predict $y = 0$

- Let's plot the sigmoid function to see where $g(z) >= 0.5$

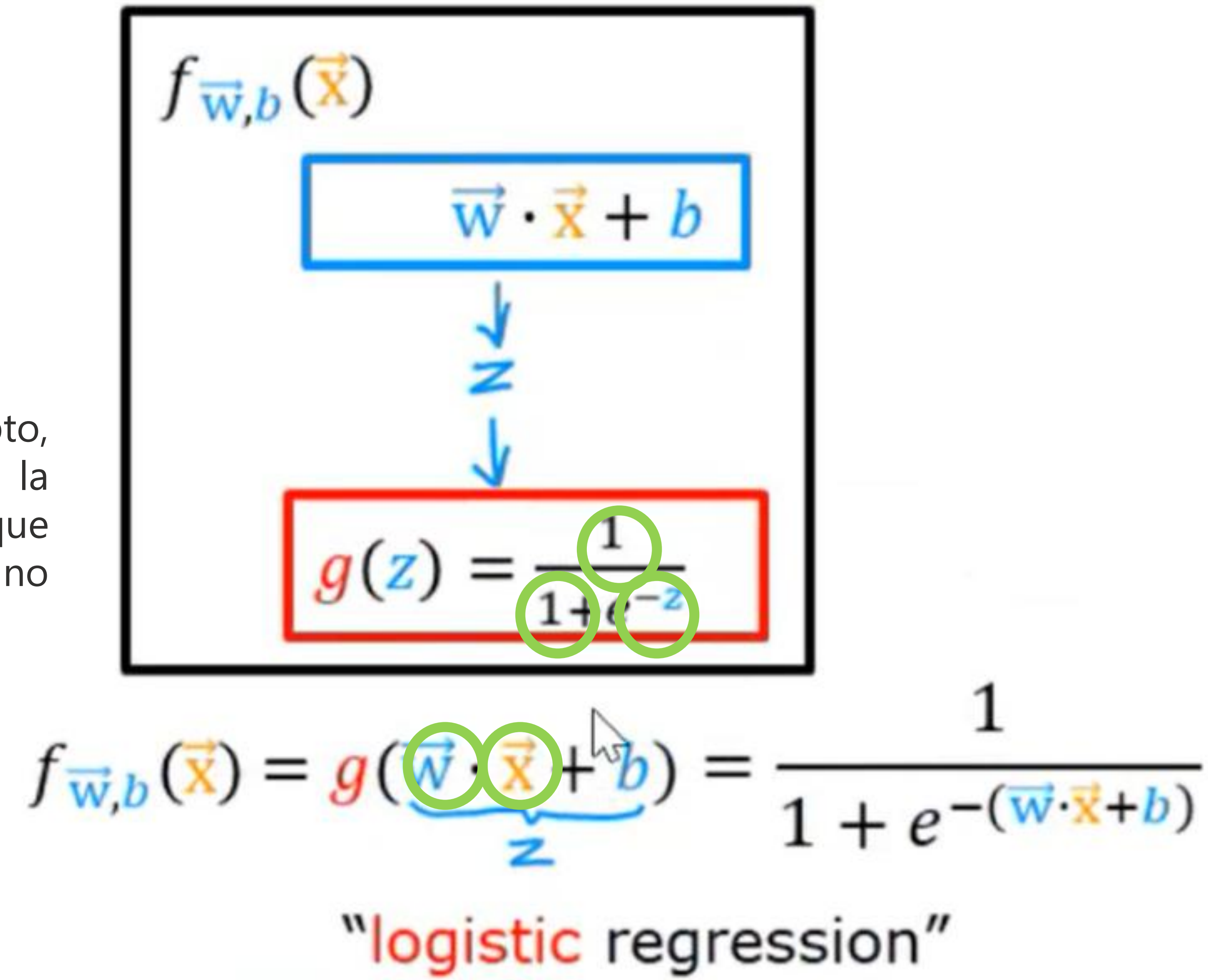
- **Eje X (Tumor Size)**: Representa el tamaño del tumor, que es la característica utilizada para predecir la clasificación del tumor.
- **Eje Y (y)**: Representa la probabilidad de que un tumor sea maligno (1) o benigno (0).



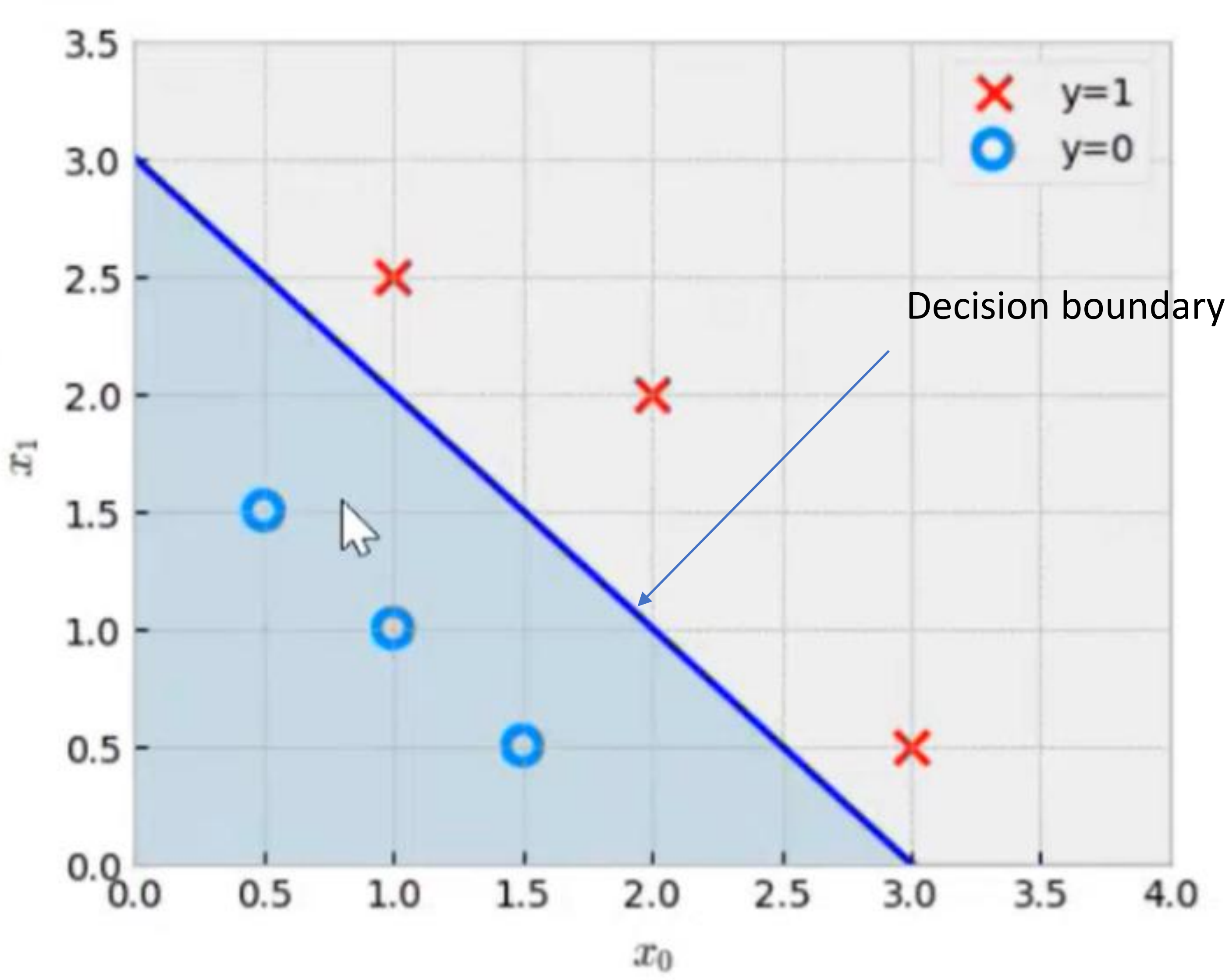
El problema es generar la ecuación correcta ,necesito que los coeficientes sean los propios para que la sigmoide se tuerza justo cuando se tiene que torcer.

Que va a modificar el modelo:

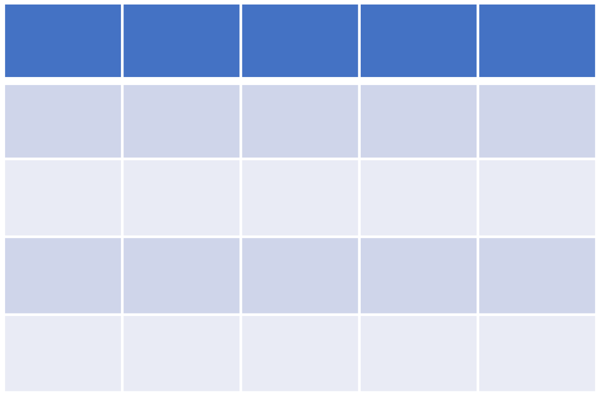
El modelo va alterar al intercepto, que es la W hasta que genere la sigmoide perfecta que se acerque siempre a decir no, cuando es no voy a decir sí cuando es sí.

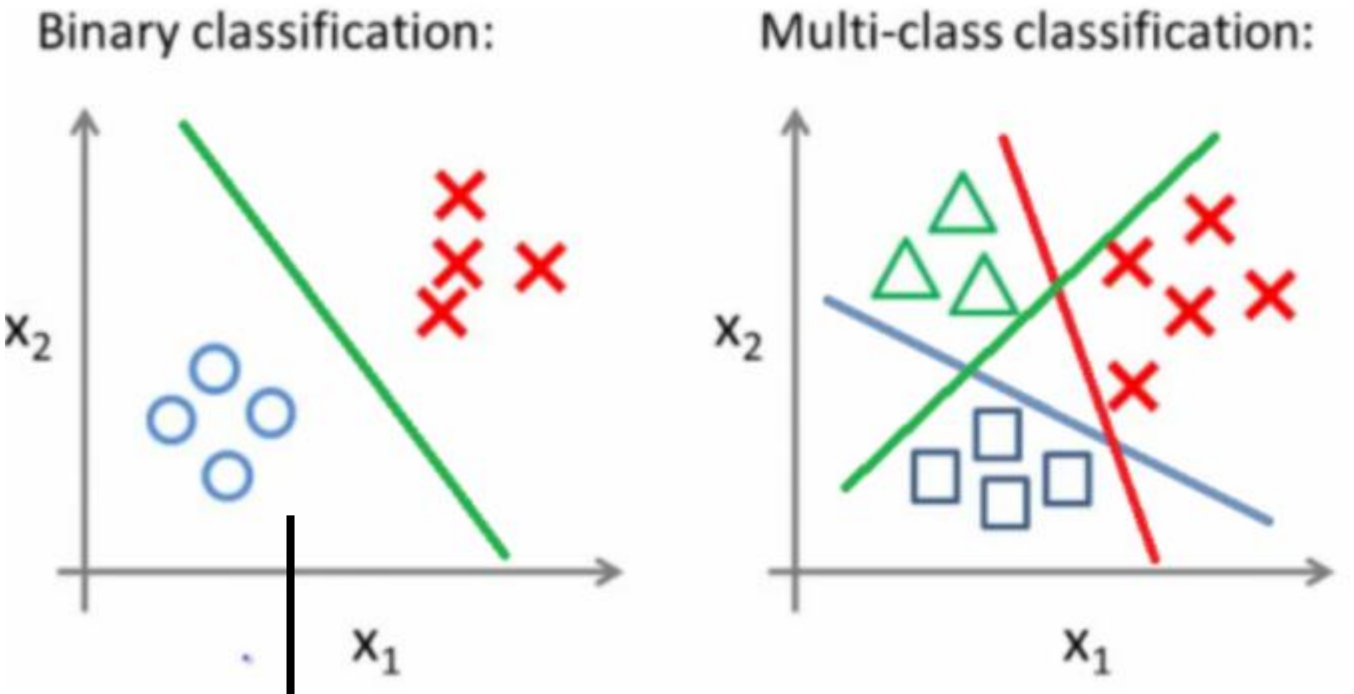


el Logaritmo natural :



5 columnas = 5D





Una clasificación binaria, voy a utilizar una sola regresión logística, una sola ecuación.

$$\hat{y} = b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a$$

$$\hat{y} = b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a$$

$$\hat{y} = b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k + a$$

Diferentes coeficientes diferentes pesos

Squared error cost

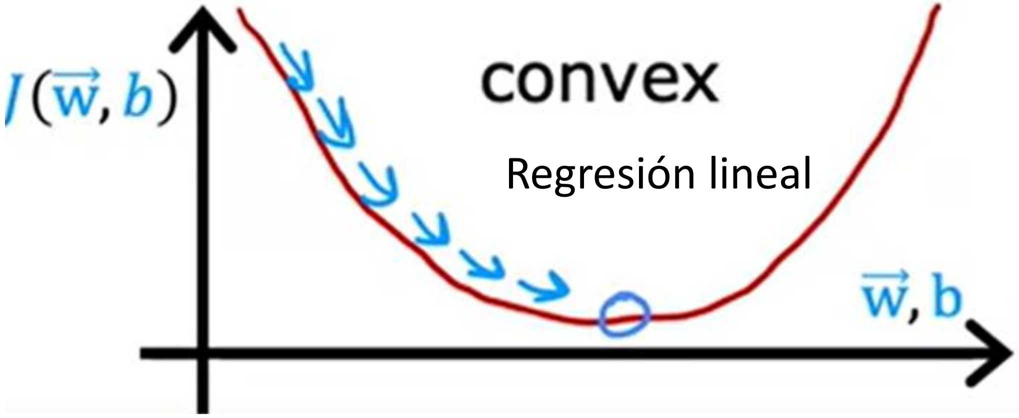
Aprox método   Datos reales

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)} \right)^2$$

Logaritmo Natural

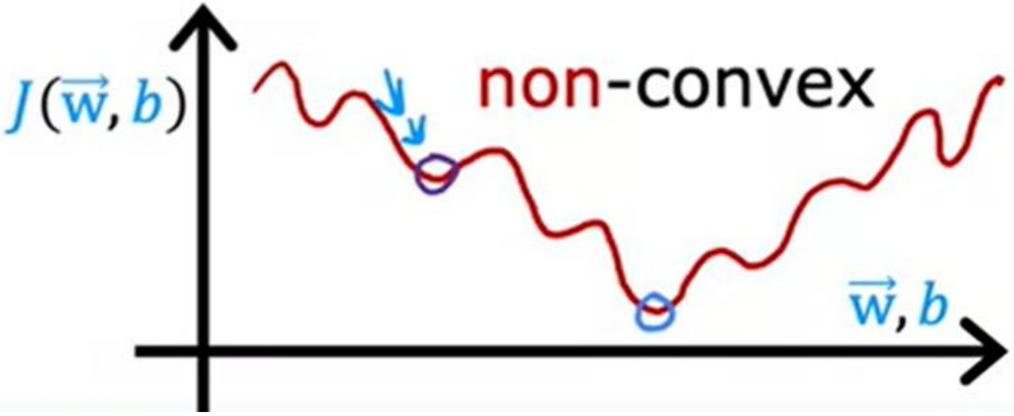
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$



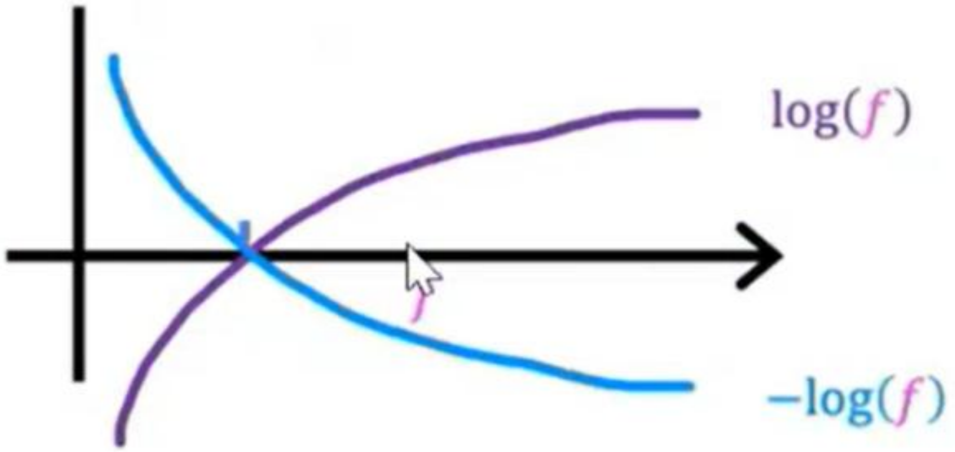
Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Predicción(entrenamiento del modelo y realidad)

Si / No tengo una sola observación Bonita/fea

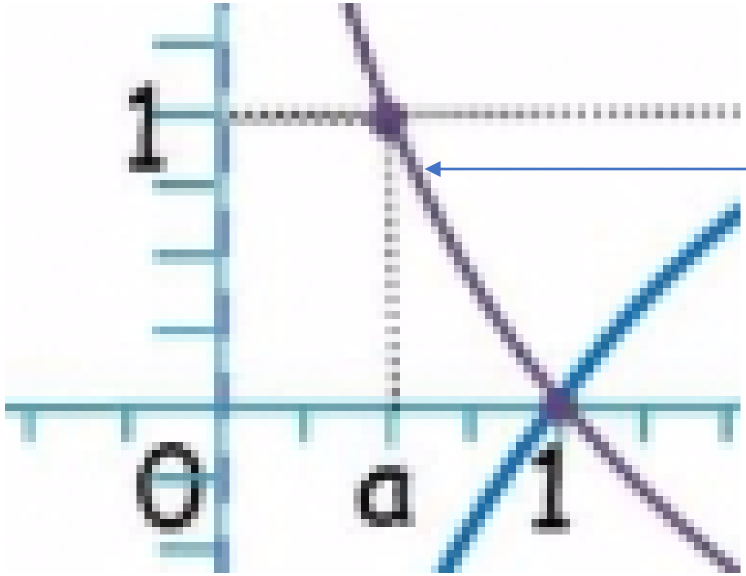
Función de pérdida : Para una predicción individual
Función de Costo para todas las observaciones de mi modelo



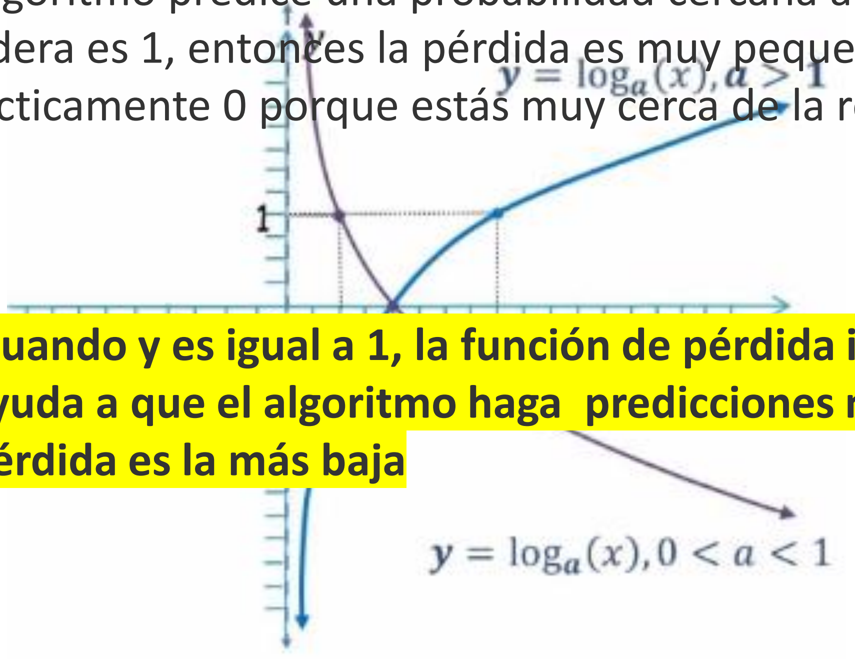
FUNCION DE PERDIDA

Para cada muestra i, la función de costo mide la discrepancia entre la predicción $f_{w,b}(\vec{x}^{(i)})$ y la etiqueta verdadera $y^{(i)}$:

$$L(f_{w,b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{w,b}(\vec{x}^{(i)})) & \text{si } y^{(i)} = 1 \\ -\log(1 - f_{w,b}(\vec{x}^{(i)})) & \text{si } y^{(i)} = 0 \end{cases}$$



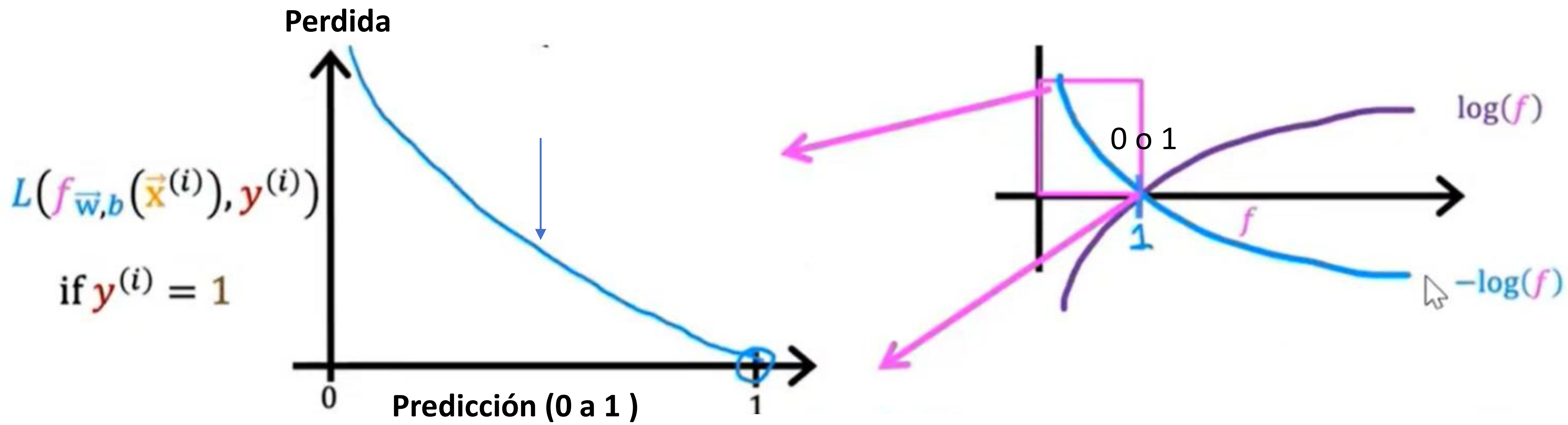
Si el algoritmo predice una probabilidad cercana a 1 y la etiqueta verdadera es 1, entonces la pérdida es muy pequeña.
Es prácticamente 0 porque estás muy cerca de la respuesta correcta.



Cuando y es igual a 1, la función de pérdida incentiva o fomenta, o ayuda a que el algoritmo haga predicciones más precisas porque la pérdida es la más baja

¿Por qué es tan útil utilizar una sigmoide en un modelo en un problema de clasificación binaria?

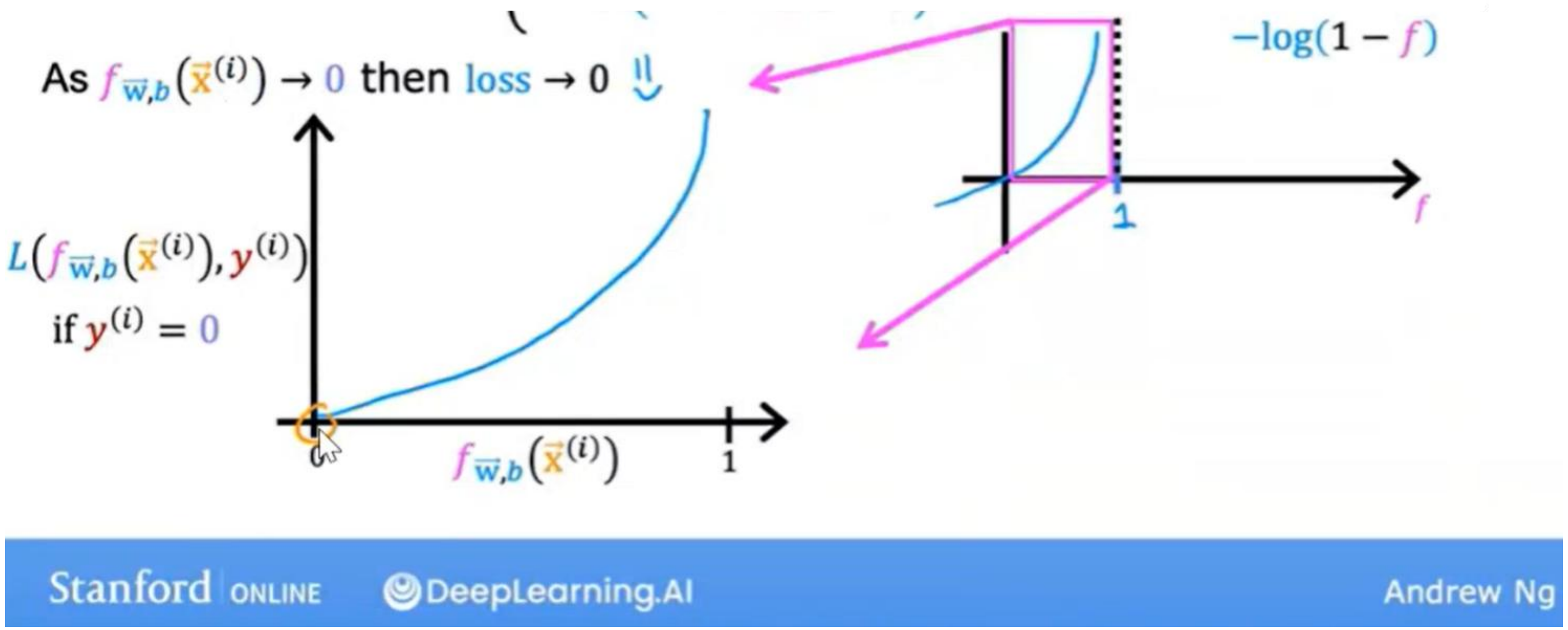
1. Perdida de ACERO es cuando coincide la realidad y la predicción
2. Perdida infinita es cuando el modelo dijo que era 0 y mentiras que 1
3. Pérdida que no es tan alta cuando él se acercó, sea que hizo una predicción intermedia



As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to \infty$

$f_{\vec{w},b}(\vec{x}^{(i)})$



Malingo o benigno

Cada que el modelo se equivoque, la pérdida tiene tiende a infinito y eso significa que el modelo será severamente castigado cada que se equivoque

Promedio de todas las observaciones individuales

Ya no es una resta entre predicción y observación , es una resta entre si es cierto y el error si me equivoco

Cost

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)}_{\text{loss}}$$

$$= \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

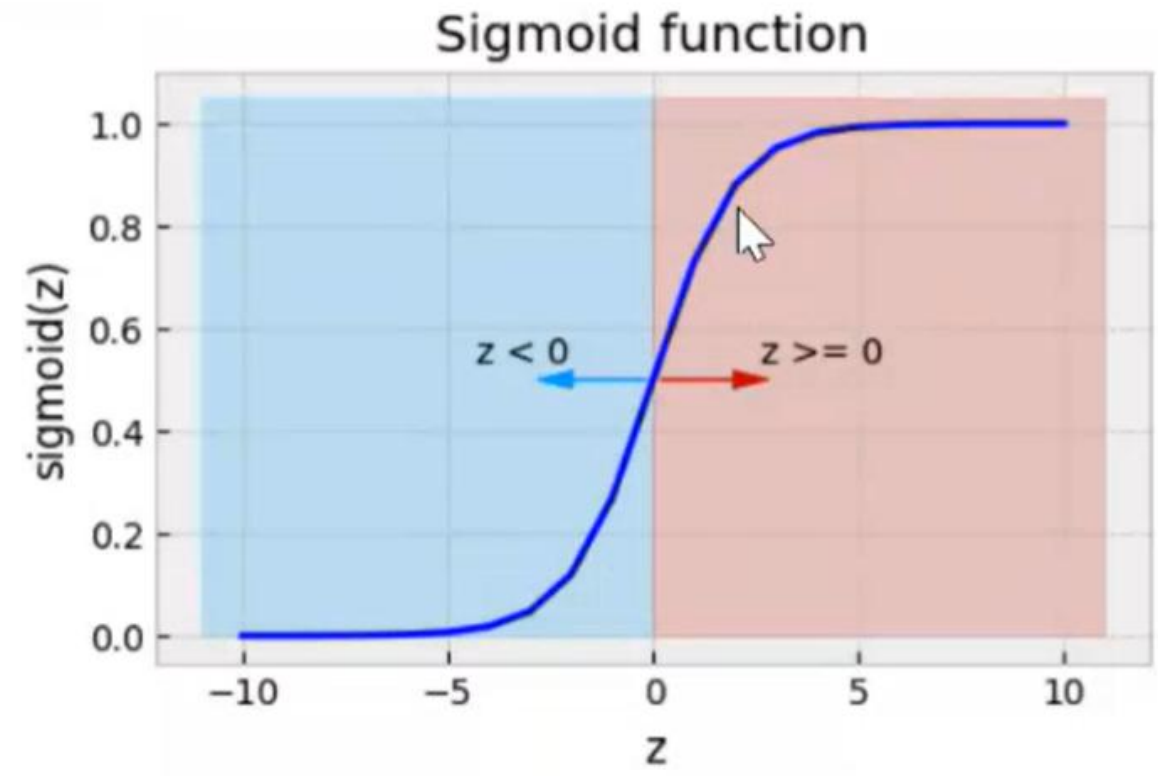
convex

↳ can reach a global minimum

find w, b that minimize cost J

El costo promedio de mi modelo sea mínimo, o sea que el modelo se equivoque, el menor número de veces que sea posible al clasificar. Intercepto hasta minimicen el costo de manera global.

Stanford ONLINE DeepLearning.AI Andrew Ng



Conclusión :

1.Deseo encontrar los coeficientes ideales que me lleven a una función de costo baja o sea, el error promedio sea mínimo es decir equivocarme el número posible de veces.

2. Para lograr ese esos coeficientes ideales, implemento un gradiente descendente, de manera que yo voy a iterar muchas veces y voy a hacer muchos ensayos con coeficientes diferentes, hasta encontrar la combinación de coeficientes que me dé la sigmoide ideal para mi modelo.

```
F/B = b(0.3) + x1 * w1 (0.001) + x2 (0.7) * w2 + x3 * w3 +  ...x4 + x5   (1/ 1
+ e(-z))

R/P = b(10.000)  + x1* w1 + x2 (0.99999) * w2 (0.9999) + x3 * w3 +  ...x4 + x5
1/1+e(-z)

A/B = b (1.5) +  x1 * w1 (0.0001) + x2 (0.2) * w2 + x3 * w3 +  ...x4 + x5   1/1
+e(-z)
```

1. **Matriz de Confusión**

- **Descripción**: Es una tabla que muestra las predicciones correctas e incorrectas realizadas por un modelo de clasificación, dividiendo los resultados en cuatro categorías:

- **Descripción**: La matriz de confusión proporciona una visión detallada de las predicciones correctas e incorrectas del modelo, desglosadas por clase.
- **Interpretación**: Permite identificar qué tipos de errores está cometiendo el modelo (FP o FN).

Vamos a ilustrar cada una de las métricas con un ejemplo práctico basado en un modelo de clasificación binaria para predecir si un correo electrónico es spam (1) o no spam (0). Supongamos que el modelo ha hecho las siguientes predicciones y tenemos las etiquetas reales:

Suposiciones:

- **Predicciones del modelo**: [1, 0, 1, 1, 0, 0, 1, 0, 1, 0]
- **Etiquetas reales (verdaderas)**: [1, 0, 1, 0, 0, 0, 1, 1, 0, 0]

A partir de estas predicciones, construimos la **matriz de confusión** y calculamos las métricas correspondientes.

1. Matriz de Confusión

Predicciones del modelo: [1, 0, 1, 1, 0, 0, 1, 0, 1, 0]

Etiquetas reales (verdaderas): [1, 0, 1, 0, 0, 0, 1, 1, 0, 0]

La matriz de confusión se vería as

	Predicho No Spam (0)	Predicho Spam (1)
Real No Spam (0)	TN = 4	FP = 1
Real Spam (1)	FN = 2	TP = 3

- **Descripción:** La matriz de confusión proporciona una visión detallada de las predicciones correctas e incorrectas del modelo, desglosadas por clase.

spam.

- **FP (False Positives):** 1 correo que no era spam fue incorrectamente predicho como spam.
- **FN (False Negatives):** 2 correos que eran spam fueron incorrectamente predichos como no spam.

2. **Precisión (Accuracy)**

La precisión es la proporción de predicciones correctas (tanto positivas como negativas) respecto al total de predicciones.

Fórmula:

$$\text{Precisión (Accuracy)} = \frac{TP + TN}{TP + TN + FP + FN}$$

Interpretación:

El 70% de las predicciones realizadas por el modelo fueron correctas.

3. **Precisión (Precision)**

La precisión mide la proporción de verdaderos positivos sobre todas las predicciones positivas.

Fórmula:

$$\text{Precisión (Precision)} = \frac{TP}{TP + FP}$$

Interpretación:
El 75% de los correos que el modelo predijo como spam fueron realmente spam. Esta métrica es importante cuando los falsos positivos son costosos.

4. Recall (Sensibilidad o Tasa de Verdaderos Positivos)

El recall mide la proporción de verdaderos positivos identificados correctamente sobre todos los casos que son realmente positivos.

Fórmula:

$$\text{Recall (Sensibilidad)} = \frac{TP}{TP + FN}$$

Interpretación:
El modelo identificó correctamente el 60% de los correos que eran spam. Esta métrica es crucial cuando los falsos negativos son inaceptables.

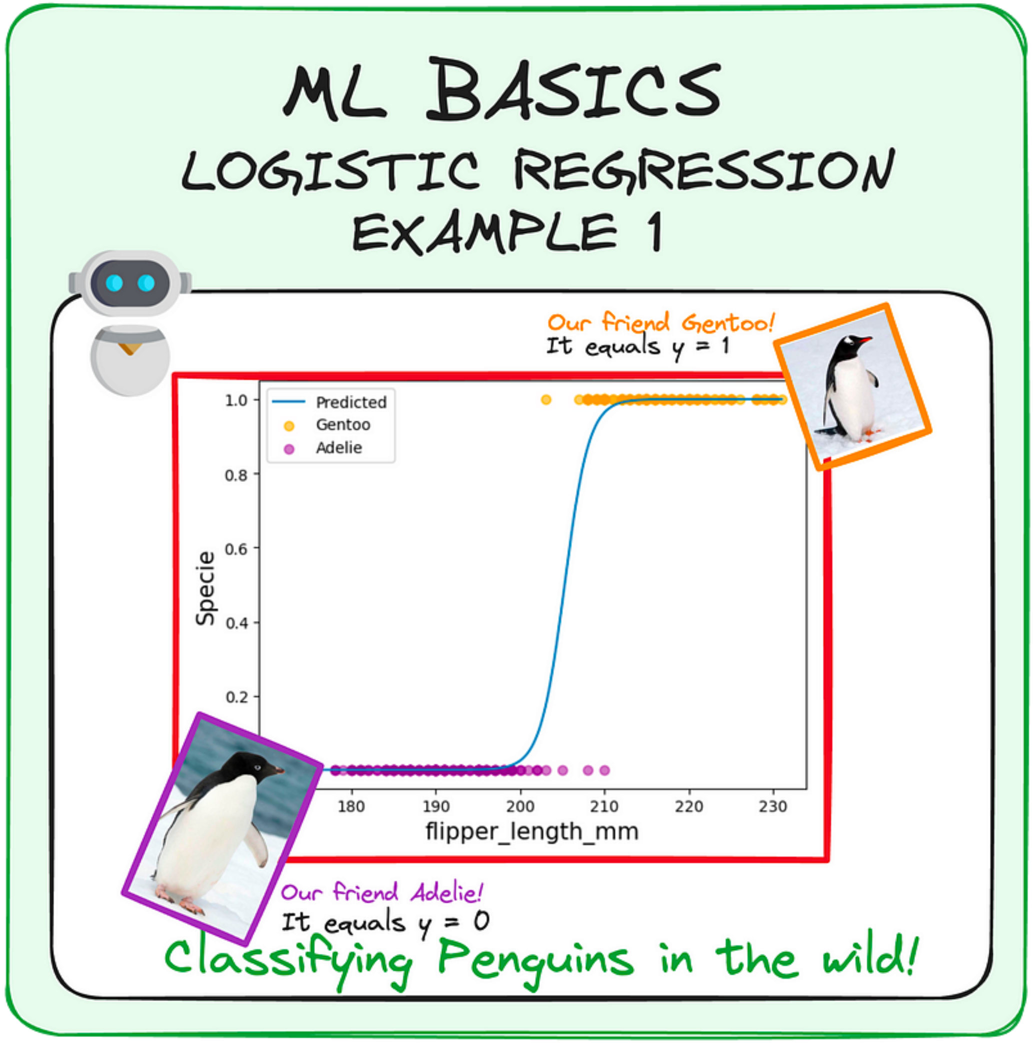
5. **F1-Score**

El F1-Score es la media armónica de la precisión y el recall. Es útil cuando necesitamos un equilibrio entre ambas métricas.

Fórmula:

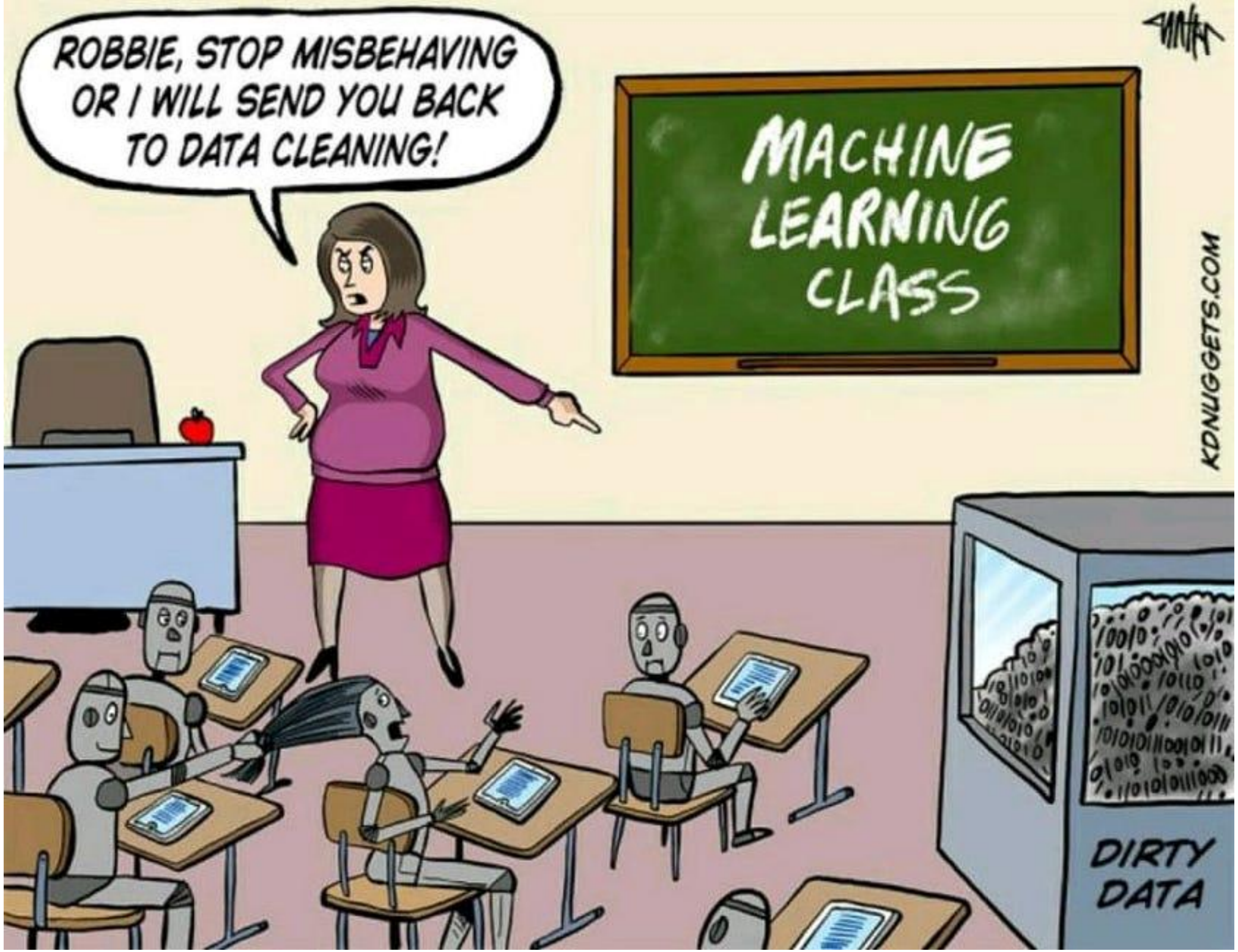
$$\text{F1-Score} = 2 \times \frac{\text{Precisión} \times \text{Recall}}{\text{Precisión} + \text{Recall}}$$

Interpretación:
El F1-Score es del 67%, lo que refleja un equilibrio entre la precisión y el recall. Es útil en escenarios donde hay un desbalance entre las clases y se desea un compromiso entre la capacidad del modelo para predecir correctamente los positivos y evitar falsos positivos.



Ejemplo en clase medico

Ejemplo de **regresión** logística


ROBBIE, STOP MISBEHAVING OR I WILL SEND YOU BACK TO DATA CLEANING!
MACHINE LEARNING CLASS
KDNUGGETS.COM
DIRTY DATA

```
    """

    m = X.shape[0]
    y = y.reshape(-1,1)              # ensure 2D
    w = w.reshape(-1,1)              # ensure 2D
    if logistic:
        if safe:  #safe from overflow
            z = X @ w + b
            cost = -(y * z) + log_1pexp(z)
            cost = np.sum(cost)/m
        else:
            f    = sigmoid(X @ w + b)
            cost = (1/m)*(np.dot(-y.T, np.log(f)) - np.dot((1-y).T, np.log(1-f)))
            cost = cost[0,0]
    else:
        f    = X @ w + b                                                      #
        cost = (1/(2*m)) * np.sum((f - y)**2)                                 #

    reg_cost = (lambda_/(2*m)) * np.sum(w**2)                                 #

    total_cost = cost + reg_cost                                              #

    return total_cost                                                         #

def compute_gradient_matrix(X, y, w, b, logistic=False, lambda_=0):
```