

# Inteligencia Artificial

Machine Learning - (Aprendizaje Automático)

Aprendizaje Supervisado - Clasificación

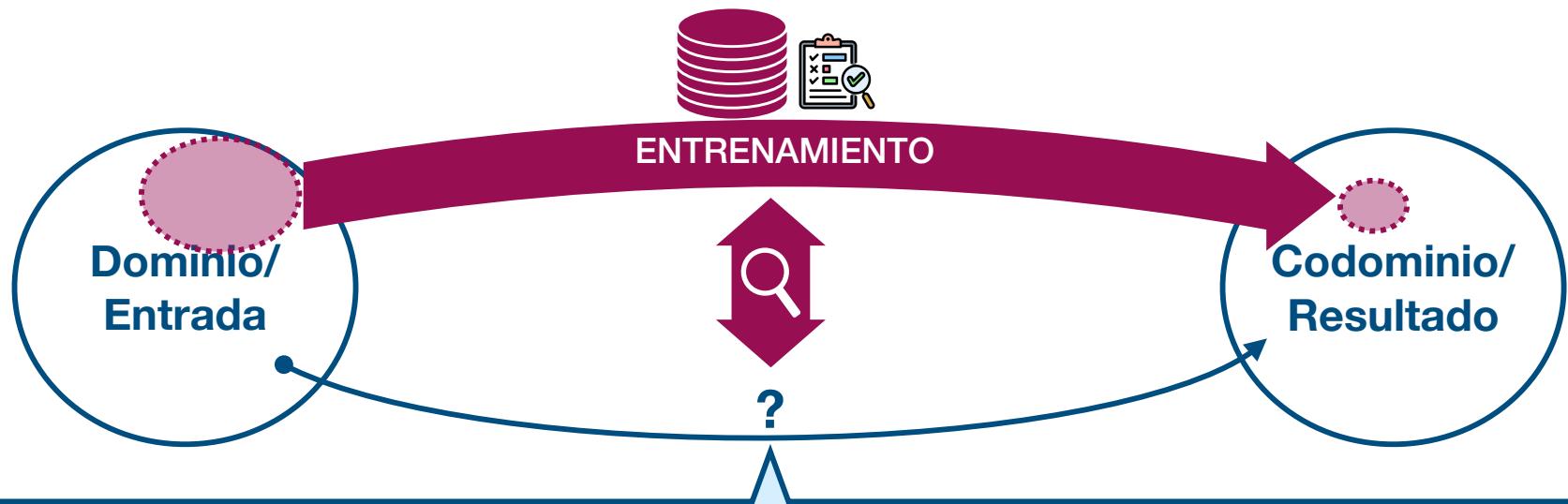


# Machine Learning - Qué esperamos como resultado

| Tipo de Aprendizaje                 | Tipo de Resultado Esperado              | Ejemplo   |
|-------------------------------------|---|---|
| <b>Aprendizaje Supervisado</b>      | Clasificación                           | Predecir categorías. Ej: detectar si un email es spam o no.             |
|                                     | Regresión                               | Predecir valores numéricos. Ej: precio de una casa.                     |
| <b>Aprendizaje No Supervisado</b>   | Agrupamiento (Clustering)               | Descubrir grupos. Ej: segmentación de clientes por comportamiento.      |
|                                     | Detección de anomalías                  | Identificar valores atípicos. Ej: fraude bancario.                      |
|                                     | Reducción de dimensionalidad            | Simplificar datos. Ej: visualización de datos de alta dimensión.        |
|                                     | Reglas de asociación                    | Encontrar relaciones frecuentes. Ej: productos comprados juntos.        |
| <b>Aprendizaje por Refuerzo</b>     | Secuencias óptimas de acciones (reglas) | Tomar decisiones en ambientes dinámicos. Ej: robots o videojuegos.      |
| <b>Aprendizaje Auto-supervisado</b> | Representaciones de datos               | Aprender estructuras internas sin etiquetas explícitas. Ej: embeddings. |
| <b>Aprendizaje Semi-supervisado</b> | Clasificación / Regresión               | Mezcla de datos etiquetados y no etiquetados. Mejora generalización.    |

# Machine Learning - Aprendizaje Supervisado

El algoritmo recibe un conjunto de entrenamiento **con las respuestas correctas** y aprende a generalizar para responder correctamente a nuevas entradas.

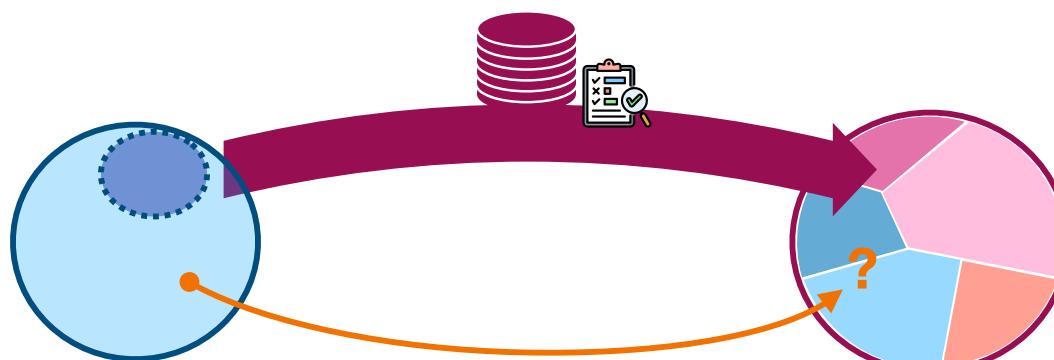


De manera general, los problemas se dividen en dos categorías, de acuerdo a la salida esperada del algoritmo:

- Problemas de **clasificación**: La salida del algoritmo es un valor tomado de un conjunto de finito de valores
- Problemas de **predicción (regresión)**: La salida es un número entero o real

# ML Supervisado - Clasificación

La clasificación es una tarea de aprendizaje supervisado en la que **un modelo aprende a asignar una etiqueta o categoría a una entrada**, basándose en ejemplos previos con etiquetas conocidas. Estas etiquetas o categorías deben ser discretas y acotadas, sino estaríamos en un problema de regresión.



Existen diferentes tipos de clasificación, algunas de ellas son:

- **Clasificación Binaria:** cuando la cantidad de características o etiquetas son dos (por ejemplo, detección de spam en el correo)
- **Clasificación Multi-clase:** cuando la cantidad de características o etiquetas son más de dos (por ejemplo, detección de frutas)
- **Clasificación Multi-etiqueta:** cuando los individuos pueden tener más de una etiqueta o característica (por ejemplo, características de una foto: (sepia/color, día/noche, ...))

# MLS Clasificación - ¿Cómo medimos la calidad?

Los modelos de aprendizaje que clasifican (**clasificadores**) pueden tener error en sus respuestas, es decir, predicen clases incorrectas para algunos individuos de la población. Existen varias métricas que ayudan a evaluar la calidad de los clasificadores. En su gran mayoría estos indicadores se alimentan de los siguientes posibles resultado de las predicciones:

| Resultados REALES     |    |    |
|-----------------------|----|----|
| Resultados del Modelo | TP | FP |
|                       | FN | TN |

- **True positives TF (Positivos Verdaderos)**: casos en los cuales la clase **C** predicha coincide con la real
- **False positives FP (Positivos Falsos)**: casos en los cuales la clase **C** predicha no coincide con la clase del individuo correspondiente
- **True negatives TN(Negativos Verdaderos)**: casos en los cuales la clase predicha no es **C**, y efectivamente los individuos correspondientes no son de clase **C**
- **False negatives FN(Negativos Falsos)**: casos en los cuales la clase predicha por el clasificador no es **C**, pero los individuos correspondientes corresponden a la clase **C**.

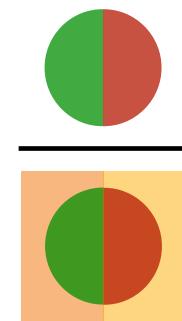
MATRIZ DE CONFUSION

# MLS Clasificación - ¿Cómo medimos la calidad?



Una métrica general es la de **exactitud** o precisión global (**Accuracy**):

$$\frac{TP + TN}{TP + TN + FP + FN}$$



Es una métrica muy simple ¿ cuántos clasifica correctamente ?, Puede ser engañosa cuando las clases están desbalanceadas

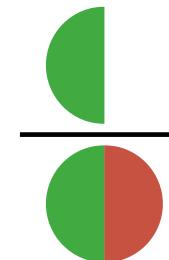
# MLS Clasificación - ¿Cómo medimos la calidad?



Dos métricas importantes son **Precision** y **Recall**:

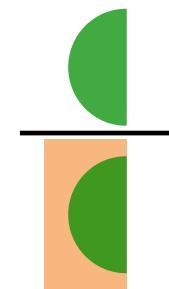
- **Precision (Precisión)**: De todos los casos que el modelo predijo como positivos, ¿cuántos realmente eran positivos? (confiabilidad)

$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$



- **Recall (Exhaustividad)**: De todos los casos que realmente eran positivos, ¿cuántos detectó el modelo correctamente? (detección)

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$



# MLS Clasificación - ¿Cómo medimos la calidad?, ejemplo

Supongamos un clasificador de correo “basura” (SPAM), llegaron 100 correos electrónicos de los cuales 20 eran basura.

Nuestro clasificador detectó 15 correos basura, pero solo 10 efectivamente lo eran. Calculemos los indicadores vistos de su comportamiento en ese escenario:

|          |          |
|----------|----------|
| TP<br>10 | FP<br>5  |
| FN<br>10 | TN<br>75 |



$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 66\%$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 50\%$$

# MLS Clasificación - ¿Cómo medimos la calidad?

Otro indicador que combina los anteriores es **F1-Score**:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- Si **precision** y **recall** son **ambos altos** entonces el **F1** será **alto**.
- Si **uno es muy bajo** entonces el **F1 baja mucho** (por la media armónica).
- Sirve cuando queremos **balancear** falsos positivos y falsos negativos.

# MLS Clasificación - ¿Cómo medimos la calidad?

| Métrica   | Fórmula   | ¿ qué indica ?                                      | ¿ cuándo usarla ?                       |
|-----------|---|---|---|
| Precision | $TP/(TP + FP)$  | ¿Qué tan confiables son las predicciones positivas? | Queremos evitar <b>falsos positivos</b> |
| Recall    | $TP/(TP + FN)$  | ¿Qué tan bien detecta los positivos reales?         | Queremos evitar <b>falsos negativos</b> |
| F1-Score  | $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$ | ¿Qué tan balanceado está el modelo?                 | Queremos equilibrio entre ambos         |

Ejemplos:

Si necesitamos un clasificador para detectar videos seguros para niños, se priorizará mostrar sólo videos seguros (alta Precision) pudiendo rechazar varios videos seguros (bajo Recall). Por otro lado, si necesitamos un clasificador para detectar sospechosos en videos de vigilancia de una tienda, se priorizará reportar siempre si cualquier acción sospechosa (alto Recall), pudiendo muchos de ellos ser falsos positivos (baja Precisión), la decisión será evaluada por el agente de vigilancia.

# MLS Clasificación - ¿Cómo medimos la calidad?

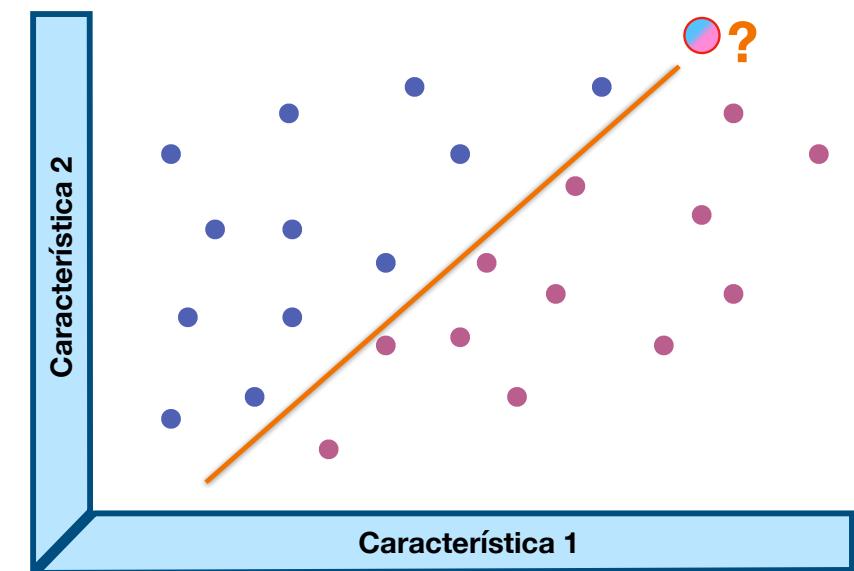
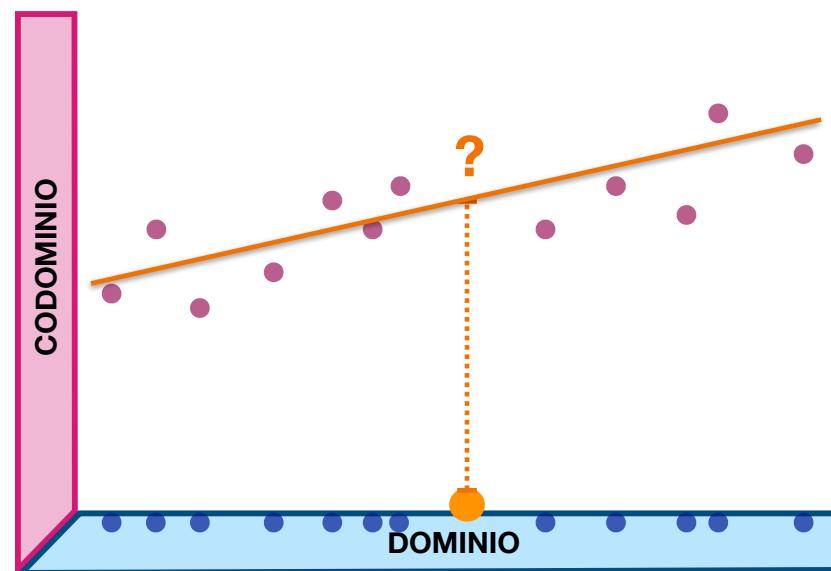
| Métrica   | Fórmula   | ¿ qué indica ?                                      | ¿ cuándo usarla ?                       |
|-----------|---|---|---|
| Precision | $TP/(TP + FP)$  | ¿Qué tan confiables son las predicciones positivas? | Queremos evitar <b>falsos positivos</b> |
| Recall    | $TP/(TP + FN)$  | ¿Qué tan bien detecta los positivos reales?         | Queremos evitar <b>falsos negativos</b> |
| F1-Score  | $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$ | ¿Qué tan balanceado está el modelo?                 | Queremos equilibrio entre ambos         |

Ejemplos:

Si necesitamos un clasificador para detectar videos seguros para niños, se priorizará mostrar sólo videos seguros (alta Precision) pudiendo rechazar varios videos seguros (bajo Recall). Por otro lado, si necesitamos un clasificador para detectar sospechosos en videos de vigilancia de una tienda, se priorizará reportar siempre si cualquier acción sospechosa (alto Recall), pudiendo muchos de ellos ser falsos positivos (baja Precisión), la decisión será evaluada por el agente de vigilancia.

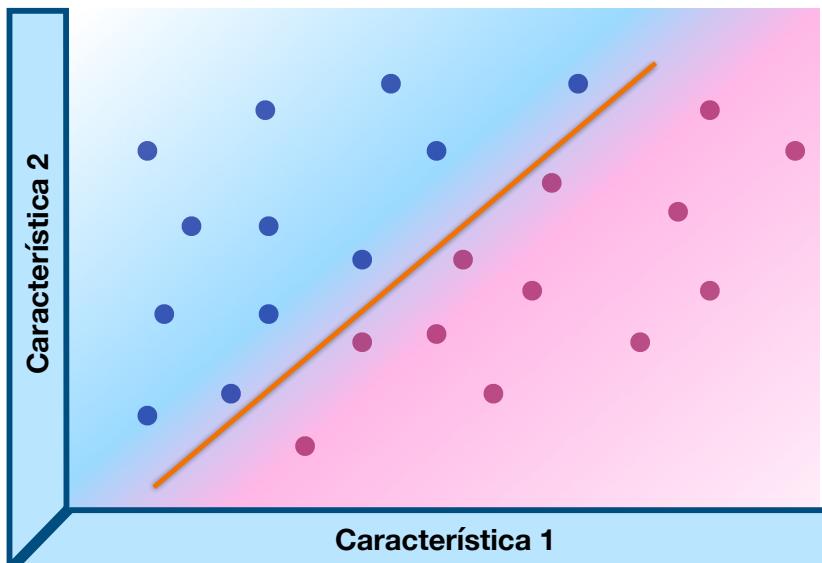
# MLS Clasificación - Regresión a Clasificación

En los problemas de clasificación, no intentamos predecir un valor específico aprendido de conjunto de datos de entrenamiento, sino predecir una clasificación de los mismos según sus características. En una clasificación binaria, las posibles respuestas son **Si** o **NO**.



# MLS Clasificación - Clasificación Lineal

Los casos más simples de clasificación es la **clasificación lineal**. Donde buscamos aprender una **función lineal que nos divida** el conjunto de elementos según la clase a la que pertenece. A esta función se la denomina **decision boundary** (frontera de decisión) o separador lineal.



Un **clasificador lineal** es una recta conformada por puntos que satisfacen  $w_0 + w_1x_1 + w_2x_2 = 0$

- Puntos tales que  $w_0 + w_1x_1 + w_2x_2 > 0$  están sobre la línea y pertenecen a una categoría
- Puntos tales que  $w_0 + w_1x_1 + w_2x_2 < 0$  están debajo de la línea y pertenecen a la otra categoría

# MLS Clasificación - Clasificación Lineal

Una de las formas más simples para encontrar una función de separación (boundary decision)  $h(x) = w_0 + w_1x_1 + w_2x_2$  es mediante la regla de aprendizaje:

$$w_i \leftarrow w_i + \alpha(y - h_w(\bar{x})) * x_i$$

Tasa de aprendizaje

La regla permite **converger a una solución si los datos son linealmente separables**:

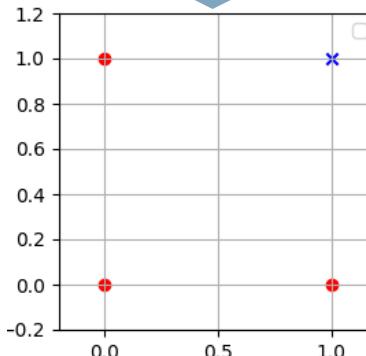
- Si el resultado es correcto,  $y = h_w(\bar{x})$ , los pesos no se modifican
- **Si  $y = 1$  y  $h_w(\bar{x}) = 0$** ,  $w_i$  se incrementa si  $x_i$  es positivo y se decrementa cuando  $x_i$  es negativo (buscamos hacer  $w_0x_0 + w_1x_1 + w_2x_2$  más grande para que  $h_w(\bar{x}) = 1$ )
- **Si  $y = 0$  y  $h_w(\bar{x}) = 1$** ,  $w_i$  se decrementa si  $x_i$  es positivo y se incrementa cuando  $x_i$  es negativo (buscamos hacer  $w_0x_0 + w_1x_1 + w_2x_2$  más chico para que  $h_w(\bar{x}) = 0$ )

# MLS Clasificación - Clasificación Lineal

Ejemplo simple:

casos a aprender

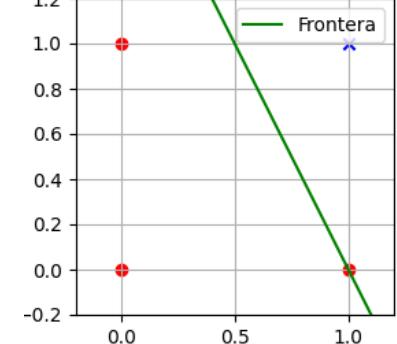
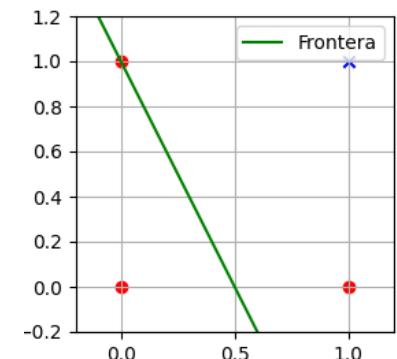
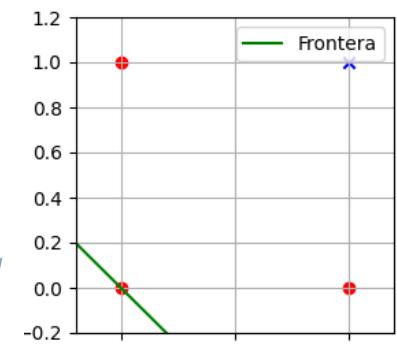
| Característica |    | clase |
|----------------|----|-------|
| x1             | x2 | y     |
| 0              | 0  | ○     |
| 0              | 1  | ○     |
| 1              | 0  | ○     |
| 1              | 1  | ✗     |



$$w_i \leftarrow w_i + \alpha(y - h_w(\bar{x})) * x_i$$

learning rate: 0.8

| Caso    | Valor h(x) | Clase predicha | error | w0    | w1   | w2   |
|---------|------------|----------------|-------|-------|------|------|
| epoca 1 |            |                |       |       |      |      |
| [0 0]   | 0.00       | ✗              | -1.00 | -0.80 | 0.00 | 0.00 |
| [0 1]   | -0.80      | ○              | 0.00  | -0.80 | 0.00 | 0.00 |
| [1 0]   | -0.80      | ○              | 0.00  | -0.80 | 0.00 | 0.00 |
| [1 1]   | -0.80      | ○              | 1.00  | 0.00  | 0.80 | 0.80 |
| epoca 2 |            |                |       |       |      |      |
| [0 0]   | 0.00       | ✗              | -1.00 | -0.80 | 0.80 | 0.80 |
| [0 1]   | 0.00       | ✗              | -1.00 | -1.60 | 0.80 | 0.00 |
| [1 0]   | -0.80      | ○              | 0.00  | -1.60 | 0.80 | 0.00 |
| [1 1]   | -0.80      | ○              | 1.00  | -0.80 | 1.60 | 0.80 |
| epoca 3 |            |                |       |       |      |      |
| [0 0]   | -0.80      | ○              | 0.00  | -0.80 | 1.60 | 0.80 |
| [0 1]   | 0.00       | ✗              | -1.00 | -1.60 | 1.60 | 0.00 |
| [1 0]   | 0.00       | ✗              | -1.00 | -2.40 | 0.80 | 0.00 |
| [1 1]   | -1.60      | ○              | 1.00  | -1.60 | 1.60 | 0.80 |
| epoca 4 |            |                |       |       |      |      |
| [0 0]   | -1.60      | ○              | 0.00  | -1.60 | 1.60 | 0.80 |
| [0 1]   | -0.80      | ○              | 0.00  | -1.60 | 1.60 | 0.80 |
| [1 0]   | -0.00      | ○              | 0.00  | -1.60 | 1.60 | 0.80 |
| [1 1]   | 0.80       | ✗              | 0.00  | -1.60 | 1.60 | 0.80 |



# MLS Clasificación - Implementando un CL

```
import numpy as np
# Dataset
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([0,0,0,1]) # Clase

# Hiperparámetros
eta = .8 # learning rate
epochs = 4 # cantidad de épocas para el entrenamiento
w = np.array([0, 0]) # pesos iniciales
b = 0

# Entrenamiento
for epoch in range(epochs):
    for i in range(len(X)):
        xi = X[i]
        target = y[i]
        z = np.dot(w, xi) + b
        y_pred = step(z)
        error = target - y_pred
    # Actualización
    w = w + eta * error * xi
    b = b + eta * error
```

Conjunto de datos (características)

Clase de cada caso

Hiperparámetros

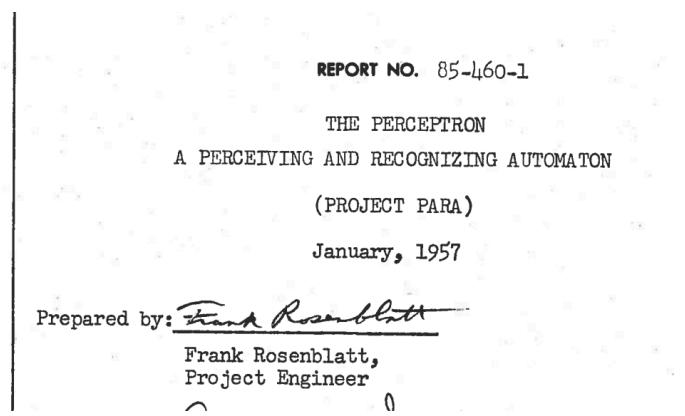
Entrenamos tantas épocas como indicamos, por cada época, ajustamos con cada caso del dataset (de entrenamiento)

Función de decisión actual

Actualización de los pesos

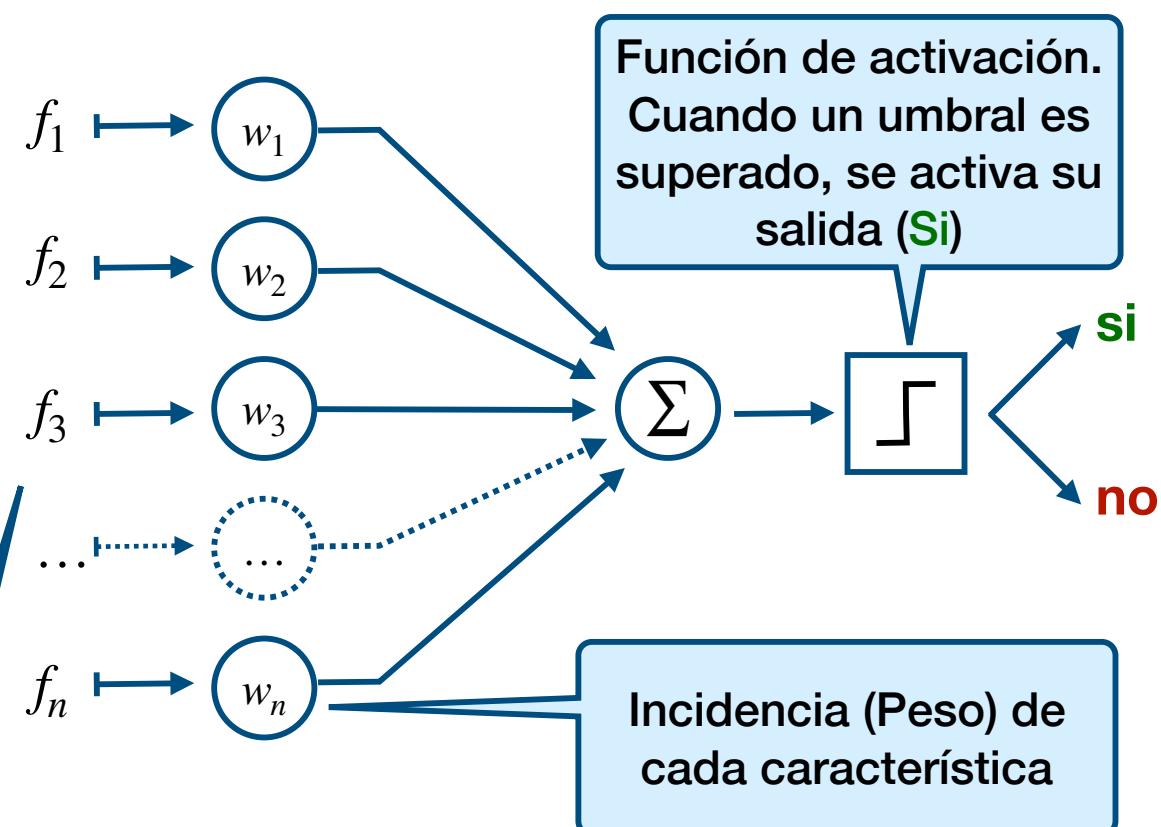
# MLS Clasificación - Clasificador binario “Perceptron”

El **perceptrón** es el modelo más simple de una red neuronal artificial, propuesto por **Frank Rosenblatt en 1958**. Está inspirado por modelos simplificados del funcionamiento de neuronas biológicas. Es el precursor de las redes neuronales.



Calcula una suma ponderada por pesos, y aplica una función de activación al resultado. Si resultado es superior a un umbral, retorna 1, sino 0

Características



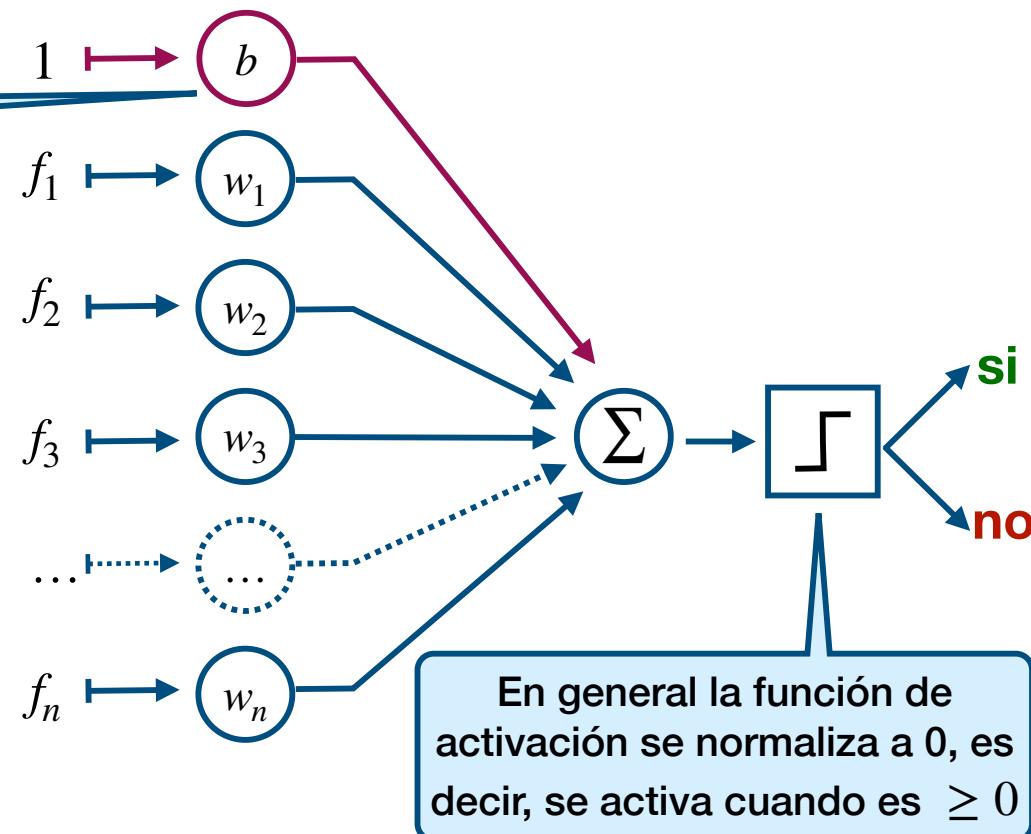
# MLS Clasificación - Clasificador binario “Perceptron”

Si se conoce el resultado esperado de la clasificación, se pueden alterar los pesos, para ajustarlos gradualmente al valor esperado. Es decir, se puede entrenar automáticamente considerando los casos conocidos (datos de entrenamiento).

El **bias** en el perceptrón es un parámetro adicional que permite desplazar la frontera de decisión y mejorar la capacidad de aprendizaje del modelo.

El proceso de aprendizaje, comienza asignando pesos aleatorios y luego se recorren todos los datos de entrenamiento ajustando el peso en base al **valor obtenido vs el esperado**, aplicando una **tasa de aprendizaje ( $\alpha$ )**

$$w_i \leftarrow w_i + \alpha(y - \hat{y}) * x_i$$

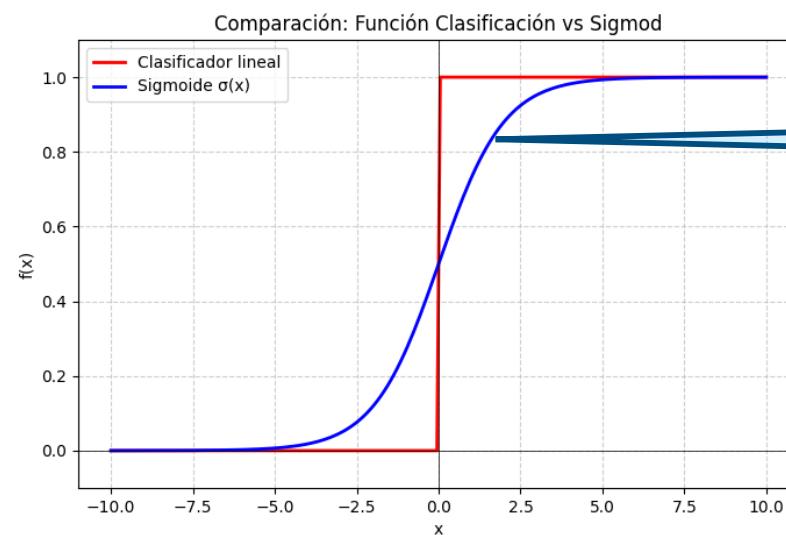


# MLS Clasificación - Logistic Regression

El **clasificador lineal** que revisamos tiene algunos inconvenientes, por un lado, dada su salida binaria, el entrenamiento puede demorar en converger, además si las clases **no son linealmente separables**, puede que nunca suceda.

La idea detrás de **Logistic Regression** es en vez de calcular si pertenece o no a una clase (discreto) es **calcular la probabilidad** de que pertenezca. Para ello utilizamos un función **Sigmoide** (en forma de “S”).

$$\text{Logistic}(z) = \frac{1}{1 + e^{-z}}$$



En la medida que más se aproxime a 0 o 1 será más confiable. Predicción cercanas a 0.5 serán poco confiables

# MLS Clasificación - Logistic Regression

Ahora que tenemos nuestra función de decisión en base a la probabilidad de que una instancia pertenezca o no a una clase, la pregunta es **qué función utilizamos como costo** para el entrenamiento.

Una de las funciones más utilizadas para Logistic Regression es **Log Loss** (pérdida logarítmica). Ésta mide cuán bien se ajustan la probabilidad predicha respecto de la clase real (valor correcto).

La pérdida logarítmica se calcula como **el promedio negativo del logaritmo de las probabilidades predichas**, teniendo en cuenta tanto las **predicciones correctas como las incorrectas**.

$$LL(\Theta) = - \frac{1}{m} \sum_{i=1}^m [y^i * \log(\sigma(i)) + (1 - y^i) * \log(1 - \sigma(i))]$$

Promedio

función sigmoid

# MLS Clasificación - Logistic Regression

Una ventaja de Logistic Regression es que la función de decisión es diferenciable, con lo cual podemos aplicar la técnica del descenso del gradiente para encontrar los parámetros que minimizan el error de predicción.

$$LL(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^i * \log(\sigma(i)) + (1 - y^i) * \log(1 - \sigma(i))]$$



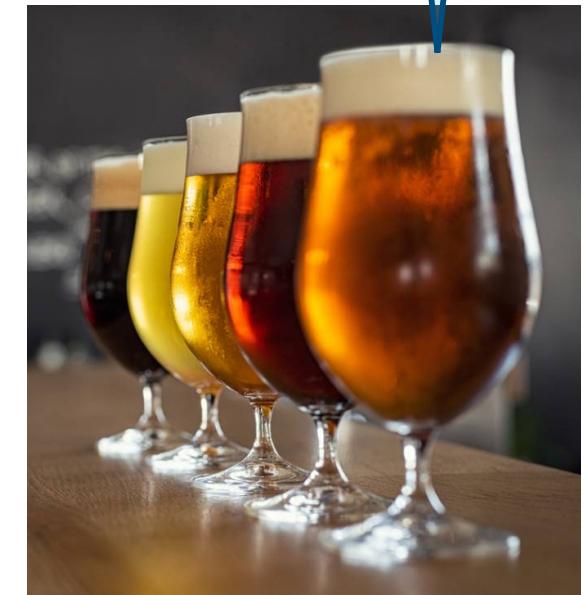
$$\frac{\delta}{\delta \theta_j} LL(\theta) = \frac{1}{m} \sum_{i=1}^m [\sigma(\theta)^T * x^i - y^i] * x_j^i$$

# MLS Clasificación - Logistic Regression

Luego podemos aplicar la técnica del **descenso del gradiente para entrenar el modelo** y obtener los parámetros que ajusten nuestra función de clasificación probabilística.

Realicemos un clasificador de estilos de cerveza (por ahora binario) teniendo en cuenta las características de su color y amargor.

Clasifiquemos entre Lagers (clara y suave) y Stouts (oscura y fuerte)



# MLS Clasificación - Logistic Regression

Implementemos la técnica

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

# Datos: (IBU, RMS) y clase
X = np.array([
    [15, 20], # Lager
    [12, 15], #...
    [28, 39],
    [21, 30],
    [45, 20], # Stout
    [40, 61],
    [42, 70] #...
])
# 0 = Lager, 1 = Stout
y = np.array([0,0,0,0,1,1,1])

# Funcion sigmoid
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```

```
# Modelo
def modelo(X, Y, learning_rate, iterations):
    X = X.T #trasponemos para realizar la multiplicación de matrices
    n = X.shape[0] #cantidad de características
    m = X.shape[1] #cantidad de casos
    W = np.zeros((n,1)) #vector de pesos para cada característica
    B = 0

    for i in range(iterations):
        Z = np.dot(W.T, X) + B #mult pesos por casos
        A = sigmoid(Z) #tenemos el vector de resultados de cada caso
        # Función de costo
        costo = -(1/m)*np.sum( Y*np.log(A) + (1-Y)*np.log(1-A))

        # Aplicación de la técnica Gradient Descent
        dW = (1/n)*np.dot(A-Y, X.T)
        dB = (1/n)*np.sum(A - Y)

        # Ajuste de pesos
        W = W - learning_rate * dW.T
        B = B - learning_rate * dB

        if(i%(iterations/10) == 0):
            print("costo luego de iteración", i, "es : ", costo)

    return W, B
```

# MLS Clasificación - Multi-clase

Qué sucede si la clasificación que deseamos utilizar estos clasificadores lineales cuando tenemos más de 2 clases en las cuales decidir la Clasificación? Por ejemplo, siguiendo el ejemplo de cervezas, si tenemos consideramos más estilos: IPA, APA, Scottish, etc.

- Enfoque **One-vs-the-Rest (OvR)**: Entrenar **un clasificador por cada clase** y seleccionamos como clase resultante aquella **con valor más alto** (mayor probabilidad)
- Enfoque **One-vs-One (OvO)**: Entrenar **un clasificador para distinguir cada par de clase**. Se genera y entranan muchos clasificadores  $N * (N - 1)/2$  para  $N$  clases y se retorna como resultado de la predicción la clase que gana más "comparaciones" (más veces vs. las restantes).

```
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier

base_clf = LogisticRegression(max_iter=200)

ovr_clf = OneVsRestClassifier(base_clf).fit(X, y)

ovo_clf = OneVsOneClassifier(base_clf).fit(X, y)
```

Utilizando la libreria SciKitLearn de Python, podemos utilizar cualquiera de estas estrategias

# MLS Clasificación - Multi-clase - Softmax

El modelo de **logística regression** puede generalizarse para manejar múltiples clases directamente, sin la necesidad de entrenar y combinar varios clasificadores binarios. Esta técnica se denomina **regresión softmax (softmax regression)** o **regresión logística multinomial (multinomial logistic regression)**:

La idea es que cuando llega una instancia  $x$  a ser clasificada:

El modelo calcula un score  $s_k(x)$  para cada clase  $k$ .

Este score es parecido a la regresión logística tradicional, es un valor lineal  $s_k(x) = \theta_k^T x$  (**notar que hay un conjuntos de pesos por cada clase**).

Luego, aplica la función **softmax** (también llamada exponencial normalizada) para

$$\text{convertir esos scores en probabilidades: } P(y = k | x) = \frac{e^{s_k(x)}}{\sum_{j=1}^K e^{s_j(x)}}$$

La idea es que todas las probabilidades quedan entre 0 y 1, y suman exactamente 1.  
La clase a predecir será la que tenga la **probabilidad más alta**.

# MLS Clasificación - Multi-clase - Softmax

Supongamos que un modelo devuelve los logits para 3 clases:

$$z = [2.0, 1.0, 0.1]$$

Aplicando Softmax, nos quedan:

$$\sigma(z_1) = \frac{e^2}{e^2 + e^1 + e^{0.1}} \approx 0.659$$

$$\sigma(z_2) = \frac{e^1}{e^2 + e^1 + e^{0.1}} \approx 0.242$$

$$\sigma(z_3) = \frac{e^{0.1}}{e^2 + e^1 + e^{0.1}} \approx 0.099$$

Podemos predecir que pertenece a la primera clase

El vector resultante para cada clase es: [0.659, 0.242, 0.099]

# MLS Clasificación - Multi-clase - Softmax

Al igual que toda técnica de aprendizaje, un aspecto clave es cómo entrenamos, es decir, cuál es la función de costo que ayuda a ajustar los parámetros de nuestro modelo para que pueda predecir con la mayor confianza y generalidad posible.

El modelo debe asignar **alta probabilidad a la clase correcta** y **baja probabilidad a las demás**. Para lograrlo se utiliza la función de **costo de entropía cruzada (cross-entropy loss)**:

$$CEL(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_{i,k} \log \hat{p}_{i,k}$$

donde:

$m$ = número de casos de aprendizaje,

$K$ = número de clases,

$y_{i,k} = 1$  si el caso  $i$  pertenece a la clase  $k$ ,  $0$  en caso contrario (one-hot encoding),

$\hat{p}_{i,k}$  = probabilidad estimada por el modelo de que el caso  $i$  sea de la clase  $k$ .

Similar a logistic pero considerando  $K$  clases

# MLS Clasificación - Multi-clase - Entropía

La **entropía** es una medida de la incertidumbre o cantidad de información contenida en una distribución de probabilidad. Fue introducida por **Claude Shannon** (1948) en su Teoría Matemática de la Comunicación.

Dada una variable aleatoria  $X$  con posibles valores  $x_1, \dots, x_k$  y probabilidades  $p_1, \dots, p_k$ :

$$H(X) = - \sum_{i=1}^k p_i \log_2(p_i)$$

Si un evento es **seguro** ( $p=1$ ), la **entropía es 0 (no hay incertidumbre)**.

Si todos los eventos son **equiprobables**, la **entropía es máxima (mucha incertidumbre)**.

# MLS Clasificación - Multi-clase - Softmax

Como la función de **costo de entropía cruzada (cross-entropy loss)** es diferenciable, para realizar el entrenamiento podemos aplicar la técnica del descenso del gradiente para ir ajustando los pesos asociados a cada clase.

$$CEL(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_{i,k} \log \hat{p}_{i,k}$$



$$\Delta_{\theta^k} CEL(\Theta) = \frac{1}{m} \sum_{i=1}^m [\hat{p}_k^i - y^i] * x^i$$

# MLS Clasificación - Multi-clase - Softmax

Podemos utilizar clasificación multiclase con SciKitLearn, utilizando la clase **LogisticRegression** y estableciendo como parámetro ***multi\_class = 'multinomial'***.

Para realizar análisis de los resultados multi-clase, también podemos crear matrices de confusión que tengan en cuenta cada clase, por ejemplo:

