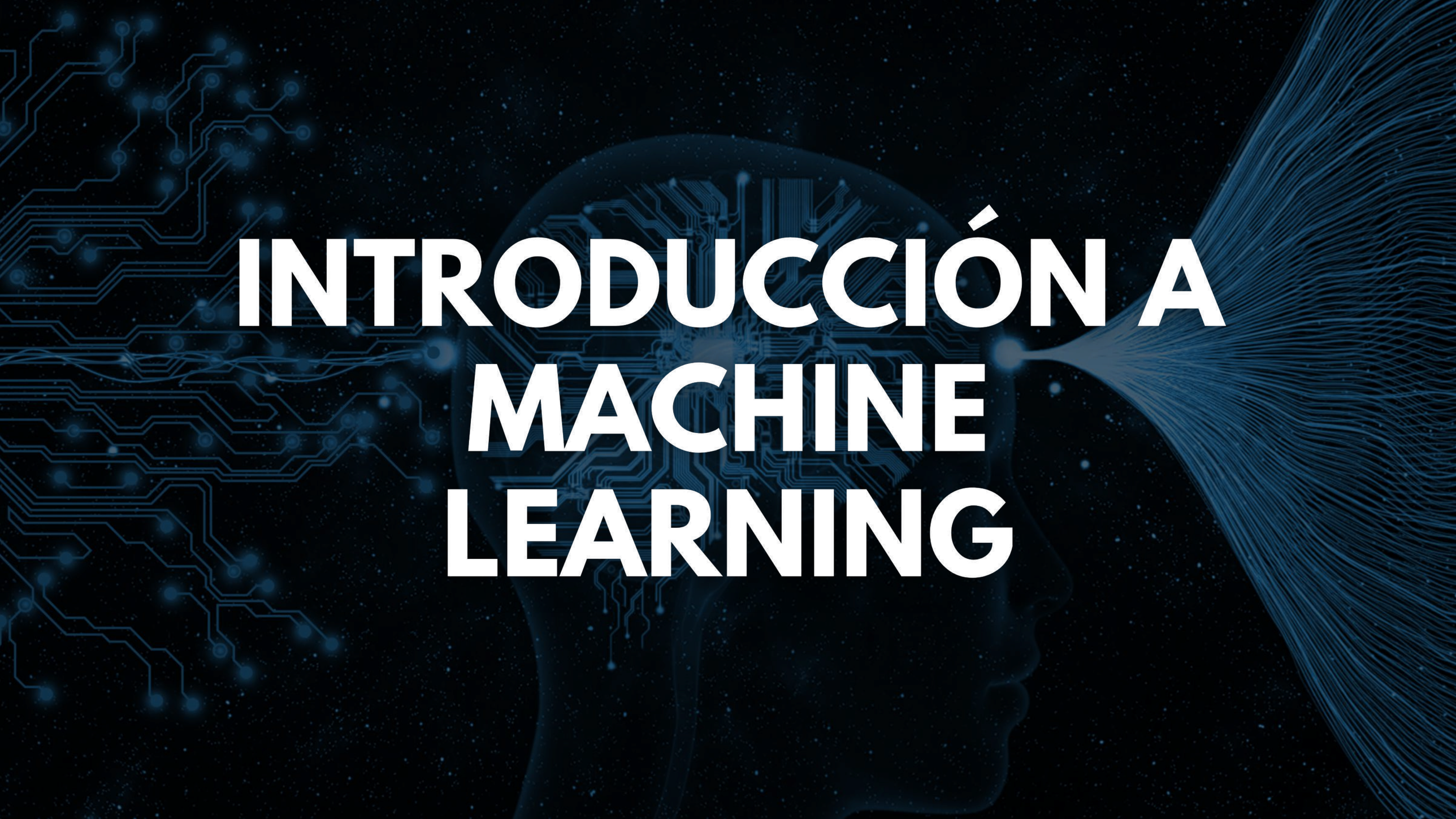
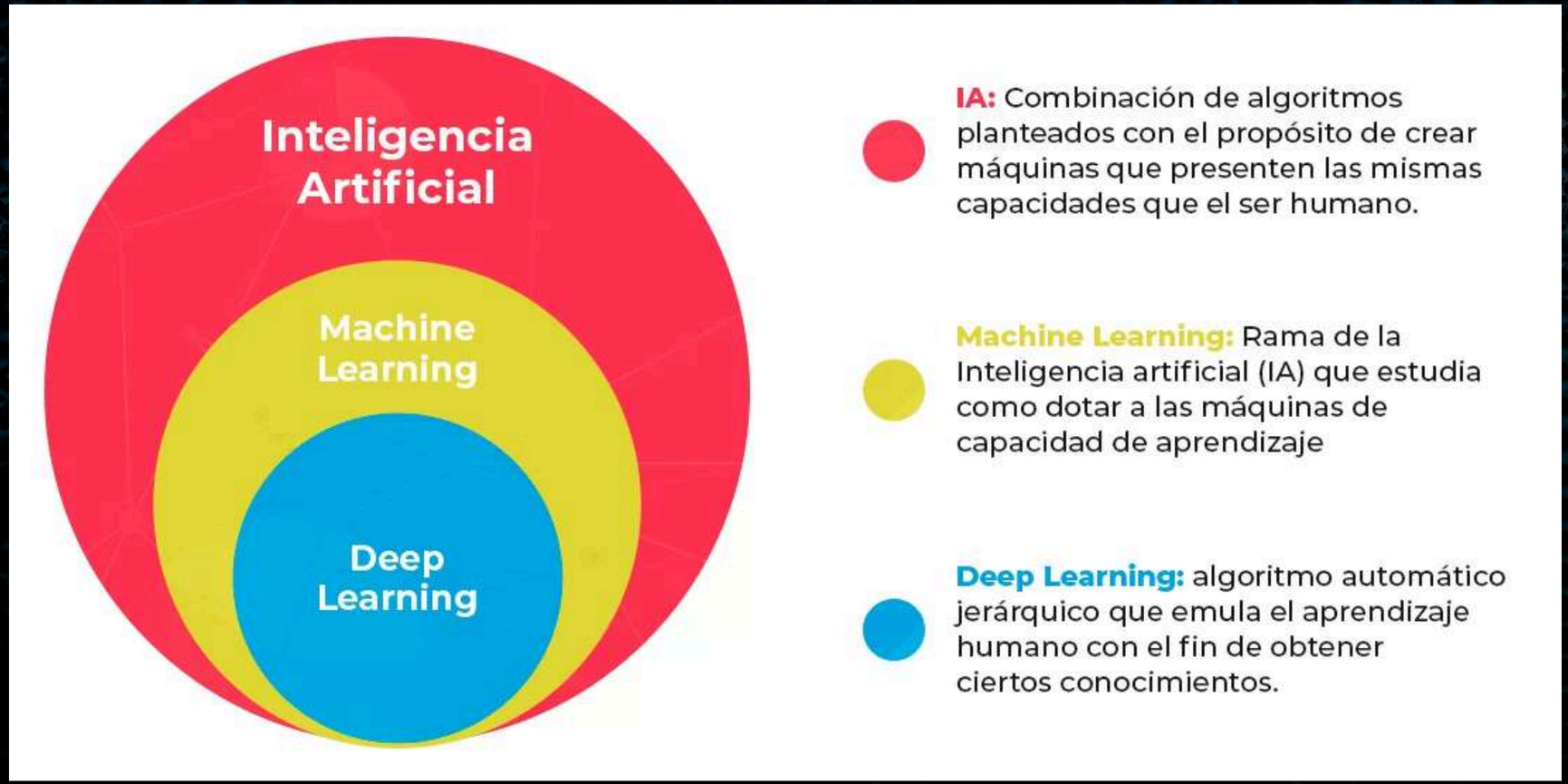
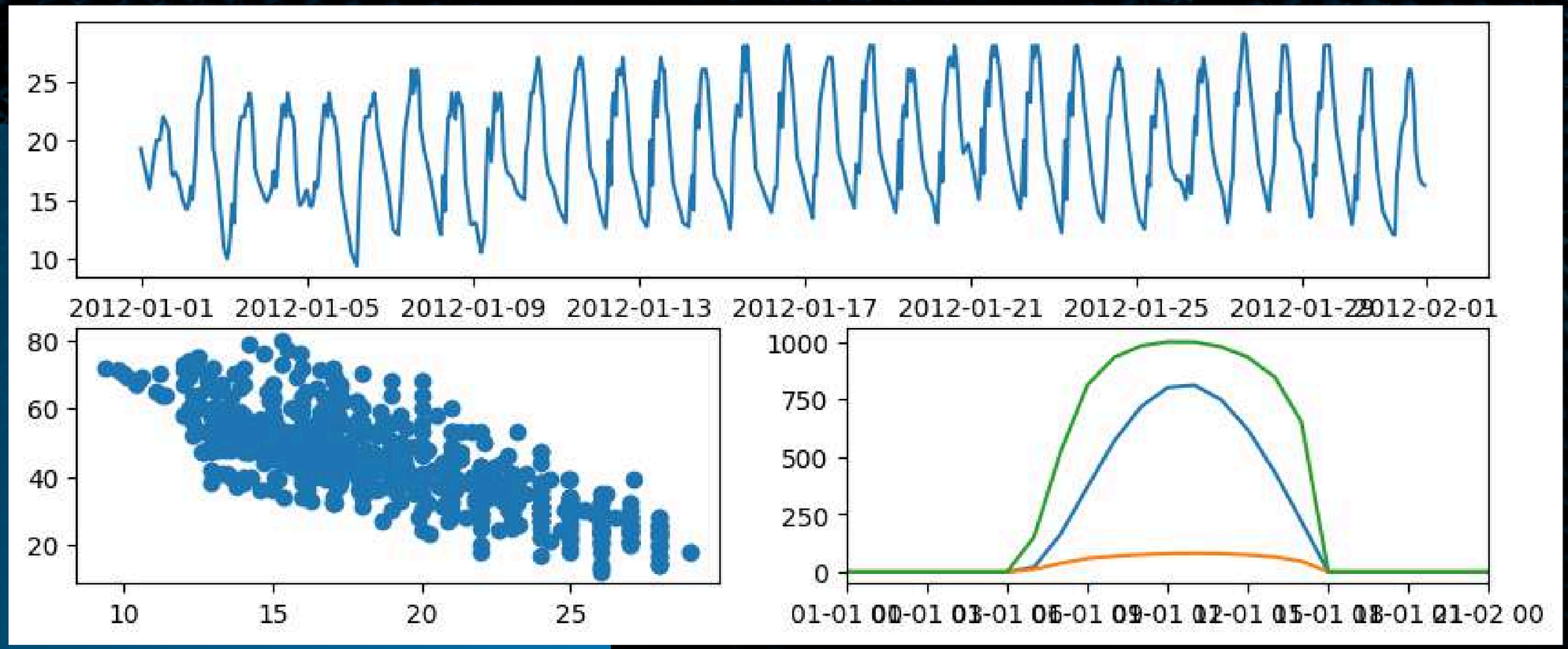

INTRODUCCIÓN A
MACHINE
LEARNING

¿Qué es el Machine Learning?

Machine Learning (ML) es un subcampo de la inteligencia artificial que permite a las máquinas aprender patrones a partir de datos sin ser programadas explícitamente para cada tarea.



Todo lo que se pueda almacenar digitalmente puede servir como dato para el Machine Learning. Al detectar patrones en esos datos, los algoritmos aprenden y mejoran su rendimiento en la ejecución de una tarea específica.



DIFERENCIAS DE UN MODELO DE ML VS TRADICIONAL

Clasificar correos como Spam a No Spam

Objetivo:

Detectar si un correo electrónico es spam (correo basura) o no spam.

Enfoque Tradicional (Programación clásica / basada en reglas)

python Copiar Editar

```python
if "gana dinero rápido" in correo and "sin esfuerzo" in correo:
    return "spam"
else:
    return "no spam"
```

Clasificar correos como Spam a No Spam

Enfoque con Machine Learning

Objetivo:
Detectar si un correo electrónico es spam o no spam.

Cómo funciona:

- **Se recopilan muchos correos etiquetados como "spam" y "no spam".**
- **Se extraen características del texto (palabras, frecuencia, etc.).**
- **Se entrena un modelo (por ejemplo, regresión logística o una red neuronal).**
- **El modelo aprende automáticamente los patrones que suelen tener los correos spam.**
- **Se usa el modelo para predecir nuevos correos.**

Clasificar correos como Spam a No Spam

Objetivo:
Detectar si un correo electrónico es spam (correo basura) o no spam.

python

Copiar Editar

```python
# Pseudocódigo
modelo = entrenar_modelo(datos_de_correos, etiquetas)
resultado = modelo.predecir(nuevo_correo)
```

Clasificar correos como Spam a No Spam

Característica	Programación Tradicional	Machine Learning
¿Aprende de los datos?	No	Sí
¿Adaptabilidad?	Requiere reprogramación	Aprende con nuevos datos
¿Precisión en tareas complejas?	Limitada	Alta con suficiente data
¿Escrito por humanos?	Reglas manuales	Aprende patrones por sí solo
¿Uso en entornos reales a gran escala?	Difícil	Muy usado (email, bancos, etc.)

Etapas principales en el desarrollo de un modelo de machine learning.

1 Recolección de datos

Recolectar información de diversas fuentes.

2 Procesamiento de datos

Se eliminan valores faltantes
Se eliminan errores.
Se meten a un formato numérico. (si es necesario)
A veces se normalizan los datos. (Para que esten en la misma escala)

Nombre	Edad	Género	Nota1	Nota2	Asistencia	Aprobado
Ana	18	F	7.0	8.0	0.9	Sí
Luis	17	M	NaN	6.5	0.8	No
Marta	19	F	6.0	7.0	0.95	Sí
Carlos	NaN	M	5.5	NaN	0.75	No

3 Selección del modelo

Regresión: predecir valores numéricos (ej. precio).
Clasificación: predecir categorías (ej. spam/no spam).
Clustering: agrupar datos similares sin etiquetas previas.

4 Entrenamiento

Aquí es donde el modelo aprende. Se le alimentan los datos y ajusta sus parámetros internos para minimizar el error entre sus predicciones y los valores reales.

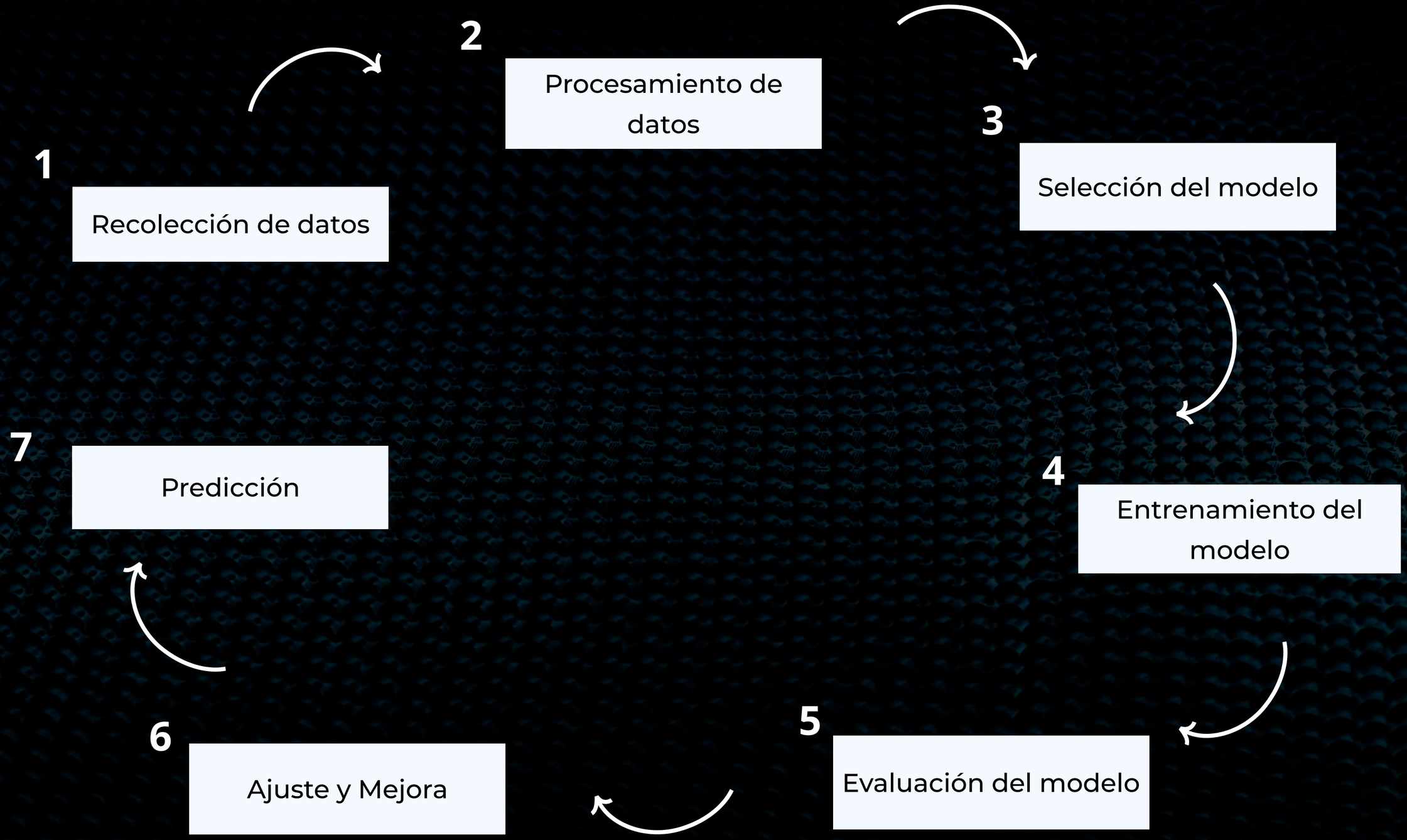
5 Evaluación del modelo

Se prueba el modelo con datos que no ha visto antes (datos de prueba) para ver qué tan bien generaliza.
Se usan métricas como:

- Precisión, recall, F1-score (para clasificación).
- Error cuadrático medio (para regresión).

6 Predicción

Una vez entrenado, el modelo puede hacer predicciones con nuevos datos.


1
Recolección de datos
2
Procesamiento de datos
3
Selección del modelo
4
Entrenamiento del modelo
5
Evaluación del modelo
6
Ajuste y Mejora
7
Predicción

Ejercicio

- Cómo tratar datos faltantes (NaN).
- Cómo convertir texto a números (codificación).
- Cómo escalar los valores para que estén en la misma escala.
- Y cómo preparar un DataFrame para alimentar a un modelo de Machine Learning.

¿Que es pandas?

Es una biblioteca de Python especializada en la manipulación y análisis de datos. Pandas también destaca en el procesamiento de datos estructurados en forma de tablas, matrices o series temporales.

¿Como funciona Pandas?

Pandas trabaja sobre DataFrames, tablas de datos bidimensionales, donde cada columna contiene los valores de una variable y cada fila contiene un conjunto de valores de cada columna.

Importación y observación del conjunto de datos

Para importar un conjunto de datos.

`pd.read_csv()`

`pd.read_excel()`

Este método devuelve el tipo de las variables, el número de columnas, el número de filas distintas de cero, el tipo de índice, el tamaño de memoria del conjunto de datos, etc.

`info()`

`describe()`

Permite obtener estadísticas descriptivas del DataFrame

Funciones principales

- pd.read_csv('archivo.csv'): Cargar datos desde un archivo CSV.
- df.head(): Ver las primeras filas de un DataFrame.
- df['columna']: Acceder a una columna.
- df.describe(): Ver estadísticas básicas.

limpieza, transformación, análisis, modelización, visualización y elaboración de informes.

¿Que es sklearn?

es una de las bibliotecas más usadas para aprendizaje automático en Python.

Dentro del módulo preprocessing, se encuentran herramientas para preparar los datos antes de entrenar un modelo

¿Que es LabelEncoder ?

Convierte etiquetas categóricas (texto) en números.

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
etiquetas = ['rojo', 'verde', 'azul']
codificadas = le.fit_transform(etiquetas)
print(codificadas)  # Resultado: [2 1 0] (el orden puede variar)
```

¿Que es StandardScaler?

Estandarizar o normalizar características numéricas, es decir, las transforma para que tengan:

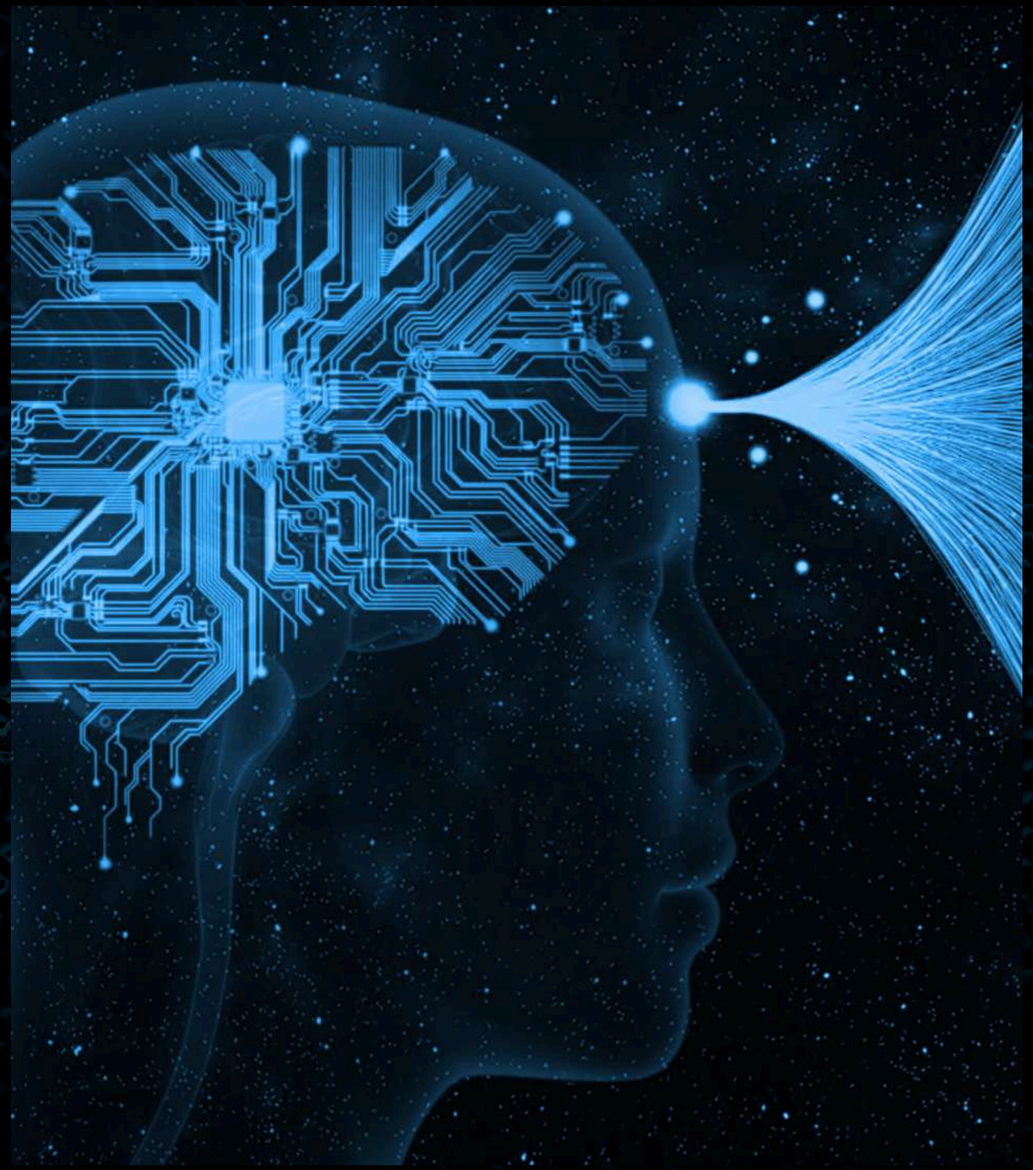
- **Media = 0**
- **Desviación estándar = 1**

Esto mejora el rendimiento de muchos algoritmos de ML (como regresión logística, SVM, redes neuronales).

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

datos = np.array([[1.0, 100.0], [2.0, 150.0], [3.0, 200.0]])
scaler = StandardScaler()
escalados = scaler.fit_transform(datos)
print(escalados)
```

Tipos principales de Machine Learning



- **Aprendizaje supervisado.**

(El modelo aprende a partir de datos etiquetados)

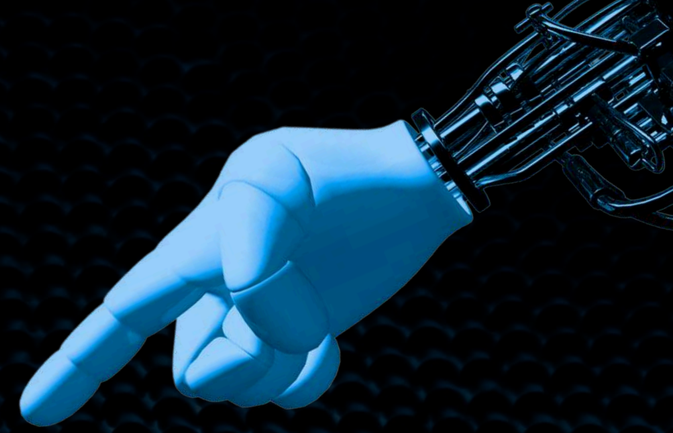
- **Aprendizaje no supervisado.**

(El modelo encuentra patrones sin etiquetas)

- **Aprendizaje por refuerzo.**

(El modelo aprende por prueba y error a través de recompensas)

Aprendizaje supervisado



El modelo aprende a partir de datos etiquetados. Es decir, cada ejemplo de entrenamiento incluye tanto las entradas (X) como la salida correcta (y).

Esto significa que cada ejemplo de entrenamiento viene con una respuesta correcta que el modelo debe aprender a predecir.

Cómo funciona el aprendizaje supervisado.

Datos etiquetados.

Una imagen de un perro con la etiqueta "perro".

Entrenamiento del modelo

El modelo analiza las características de esos ejemplos (colores, palabras, tamaño, etc.) y aprende una relación entre los datos de entrada y la etiqueta de salida.

Predicción

Una vez entrenado, el modelo puede recibir nuevos datos sin etiqueta y hacer una predicción.

- **Dada una nueva imagen, dice si es un perro o un gato.**

Entrada (X)	Etiqueta (Y)	Tarea
Imagen de un gato	"gato"	Clasificación de imágenes
Texto de una reseña	★★★★☆	Análisis de sentimiento
Medidas del cuerpo	Altura en cm	Predicción numérica (regresión)
Email	"spam" o "no spam"	Clasificación de texto

Algoritmos comunes

Algoritmo	Uso Común
Regresión Lineal	Predicción de valores continuos (ej. precios)
Regresión Logística	Clasificación binaria (ej. spam/no spam)
K-Nearest Neighbors (KNN)	Clasificación basada en similitud
Árboles de Decisión	Clasificación y regresión
Random Forest	Ensamble de árboles para mayor precisión
Support Vector Machines (SVM)	Clasificación con margen óptimo
Redes Neuronales	Reconocimiento de imágenes, voz, texto
Gradient Boosting (XGBoost, LightGBM, CatBoost)	Modelos muy potentes para Kaggle y producción

Regresión Lineal en Python usando scikit-learn

Ejercicio: Regresión Lineal en Python usando scikit-learn,

- **Objetivo:**
- **Predecir el precio de una casa en función de su superficie en m^2.**

¿Que es numpy?

Es la librería fundamental para cálculos numéricos en Python.

¿Para qué sirve?

Trabaja con arreglos (arrays) y matrices numéricas.
Mucho más rápido y eficiente que las listas de Python.
Incluye muchas funciones matemáticas y estadísticas.

```python
import numpy as np
a = np.array([1, 2, 3])
print(a * 2)  # Resultado: [2 4 6]
```

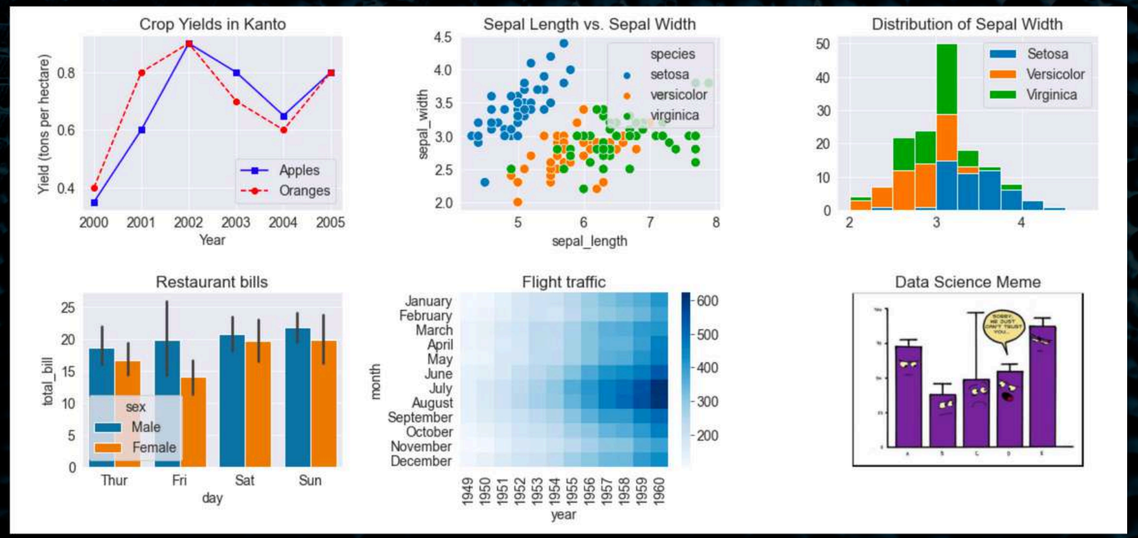
¿Que es matplotlib.pyplot?

Es una librería para crear gráficos y visualizaciones en Python.

```python
import matplotlib.pyplot as plt

x = [1, 2, 3]
y = [2, 4, 6]

plt.plot(x, y)
plt.title("Gráfico simple")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



¿Que es LineaRegression?

- LinearRegression es un modelo de regresión lineal.
- Aprende la relación entre una o más variables independientes (X) y una variable dependiente (y).

```python
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3]])
y = np.array([2, 4, 6])

modelo = LinearRegression()
modelo.fit(X, y)

print(modelo.coef_)   # Pendiente
print(modelo.intercept_)  # Intersección
```

La fórmula matemática es:

$$y = mx + b$$

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

Y: variable objetivo (lo que queremos predecir)
X1, X2, ... , Xn : variable objetivo (lo que queremos predecir)
w : coeficientes (pendientes)
b: intercepto o sesgo (bias)

¿Qué calcula internamente?

LinearRegression ajusta los coeficientes usando el método de mínimos cuadrados, que minimiza la suma de los errores cuadráticos entre los valores reales y los predichos:

$$\text{Error} = \sum (y_{\text{real}} - y_{\text{predicho}})^2$$

Regresión Logistica

Ejercicio : Regresión logística

- **Objetivo:**
- **Predecir, con 4.5 horas de estudio, hay un 70% de probabilidad de aprobar el examen”.**

Horas de estudio	¿Aprobó? (0 = No, 1 = Sí)
1	0
2	0
3	0
4	1
5	1
6	1

¿Que LogisticRegression?

Proviene de la biblioteca scikit-learn y trae la clase LogisticRegression, que sirve para clasificación (no para regresión en el sentido clásico de predecir valores continuos).

¿Que hace la regresión logistica?

En lugar de predecir un número (como el precio de una casa), predice una probabilidad de que algo ocurra.

“Con 4.5 horas de estudio, hay un 70% de probabilidad de aprobar el examen”

¿Para qué sirve?

- Clasificar correos como spam o no spam.
- Predecir si un cliente compra o no compra.
- Determinar si un paciente tiene o no tiene una enfermedad.
- Clasificación multiclase.

Función Sigmoide

Luego se usa un umbral (por defecto 0.5) para clasificar:

- Si probabilidad ≥ 0.5 calse 1 (Sí aprobó)
- Si probabilidad < 0.5 calse 0 (No aprobó)

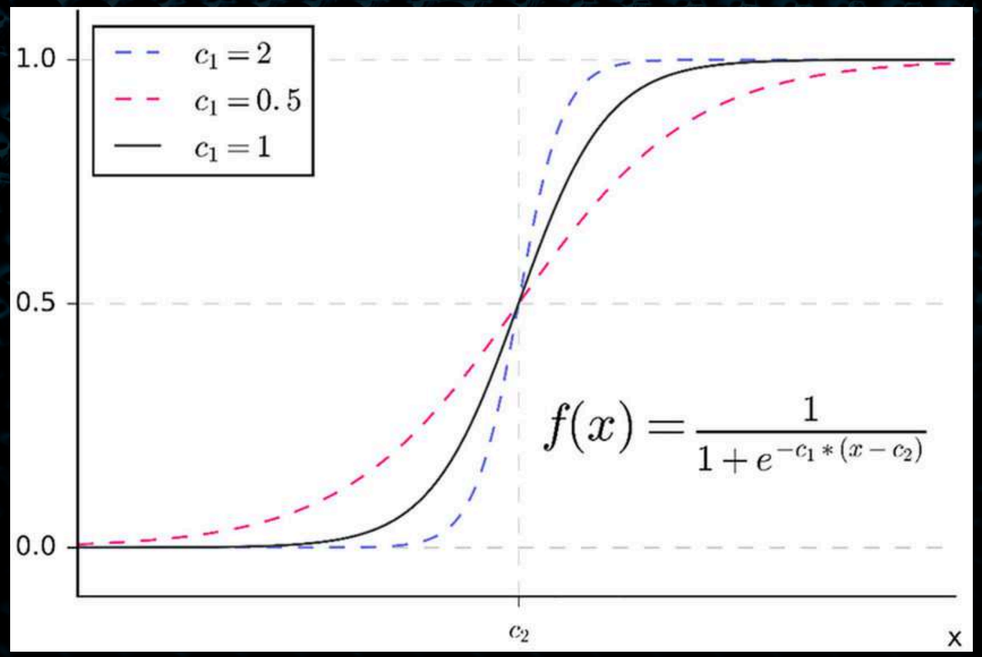
$$P(y=1) = \frac{1}{1+e^{-(mx+b)}}$$

- Esta función sigmoide, es la que da un valor entre 0 y 1.

La función sigmoide (o logística) transforma cualquier número real en un valor entre 0 y 1.

Su fórmula es:

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



Donde:

- **z es un número real (puede ser muy negativo o muy positivo)**
- **e es la constante matemática (~2.718)**

Características

- Rango: el resultado siempre está entre 0 y 1
- Si $z \to +\infty$, $S(z) \to 1$
- Si $z \to -\infty$, $S(z) \to 0$
- Forma de S: gráfica con curva en forma de "S" suave.
- Valor en 0: $S(0) = 0.5$

Interpretación en Logistic Regression

En clasificación binaria, la salida de la sigmoide se interpreta como probabilidad de pertenecer a la clase 1.

Ejemplo:

Si $S(z)=0.9S(z)$ → 90% de probabilidad de ser clase 1

Si $S(z)=0.3S(z)$ → 30% de probabilidad de ser clase 1

Luego se aplica un umbral (por defecto 0.5):

≥ 0.5 → Clase 1

< 0.5 → Clase 0

Regresión multiclase

La regresión multiclase (o más correcto: clasificación multiclase) es cuando queremos predecir más de dos categorías posibles usando un modelo como LogisticRegression.

Diferencia con la clasificación binaria

- Clasificación binaria: solo hay 2 clases posibles (ej. “spam” o “no spam”).
- Clasificación multiclase: hay 3 o más clases (ej. “perro”, “gato”, “ave”).

Cómo lo maneja LogisticRegression en scikit-learn

LogisticRegression usa estrategias para extender la lógica binaria a múltiples clases:

1. One-vs-Rest (OvR)
   - Crea un modelo por cada clase comparándola contra todas las demás.
   - Ejemplo con 3 clases (A, B, C):
   - Modelo 1: A vs (B o C)
   - Modelo 2: B vs (A o C)
   - Modelo 3: C vs (A o B)
   - Luego predice la clase con la probabilidad más alta.
2. Multinomial (Softmax)
   - Ajusta un solo modelo que calcula todas las probabilidades a la vez usando la función softmax.
   - Más eficiente y consistente para problemas con muchas clases.

Supón que tenemos un modelo para predecir el tipo de fruta según peso y color:

Peso (g)	Color (0=Verde, 1=Rojo, 2=Amarillo)	Fruta (y)
150	0	Manzana
180	1	Manzana
120	2	Plátano
130	2	Plátano
200	1	Cereza

- **Clases: ["Manzana", "Plátano", "Cereza"]**
- **El modelo asignará una probabilidad a cada fruta, y elegirá la más alta.**

K-Nearest Neighbors (KNN)

¿Que es sklearn.model_selection import train_test_split?

- Es una función que divide tu dataset en dos partes:
- Conjunto de entrenamiento (train) → para entrenar el modelo.
- Conjunto de prueba (test) → para evaluar qué tan bien aprendió el modelo con datos que no había visto antes.

¿Por qué es importante?

Porque si entrenas y pruebas con los mismos datos, el modelo podría “memorizar” en lugar de “aprender” (lo que se llama sobreajuste).

```python
from sklearn.model_selection import train_test_split

X = [[1], [2], [3], [4], [5]]
y = [0, 0, 1, 1, 1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

print("Entrenamiento:", X_train, y_train)
print("Prueba:", X_test, y_test)
```

test_size=0.4 → significa que 40% de los datos van al conjunto de prueba.

random_state=42 → asegura que la división siempre sea la misma (para reproducibilidad).

¿Que es sklearn.neighbors import KNeighborsClassifier?

- Es la clase que implementa el algoritmo K-Nearest Neighbors (KNN).
- Sirve para clasificar (o predecir) a qué categoría pertenece un nuevo dato, basándose en sus vecinos más cercanos.

Cómo funciona KNN:

- Eliges un número k (ej: 3).
- Para un nuevo punto, el modelo busca sus k vecinos más cercanos en los datos de entrenamiento.
- Hace una “votación” entre esos vecinos para decidir la clase.

```python
from sklearn.neighbors import KNeighborsClassifier

# Crear el modelo con k=3
knn = KNeighborsClassifier(n_neighbors=3)

# Entrenar el modelo
knn.fit(X_train, y_train)

# Hacer predicciones
pred = knn.predict([[4]])
print("Predicción para [4]:", pred)
```

Aquí el modelo vería cuáles son los 3 vecinos más cercanos al punto 4 y decidiría la clase.

Diferencia entre un modelo de Regresión Logística y un modelo de K-Nearest Neighbors - KNN

Regresión Logística

Tipo de modelo:

Es un modelo paramétrico → aprende una función matemática (una curva sigmoide) que separa las clases.

Cómo funciona:

- Calcula una recta o frontera en el espacio de datos.
- Usa la función sigmoide para dar probabilidades de pertenecer a una clase.
- Ejemplo: si la probabilidad > 0.5 → clase = 1, si no → clase = 0.

Regresión Logística

Características:

- Supone que los datos se pueden separar con una línea recta (o hiperplano en más dimensiones).
- Es rápido y sencillo.
- Muy interpretativo → se puede analizar el peso de cada variable.

Ejemplo visual:

- Si tus datos son puntos que forman dos grupos (uno arriba y otro abajo), la regresión logística dibuja una línea recta que los divide.

K-Nearest Neighbors (KNN)

Tipo de modelo:

- Es un modelo no paramétrico → no aprende una fórmula matemática, solo “guarda” los datos.

Cómo funciona:

- Para clasificar un nuevo punto, busca sus k vecinos más cercanos.
- La clase se decide por votación: la mayoría de esos vecinos decide.

K-Nearest Neighbors (KNN)

Características:

- No hace ninguna suposición sobre la forma de los datos.
- Puede trazar fronteras complejas (no rectas).
- Puede ser lento con muchos datos, porque tiene que calcular distancias cada vez.
- Muy sensible a la escala (peso, tamaño, etc.), por eso a veces se normalizan los datos.

Ejemplo visual:

- Si pones un punto nuevo en el gráfico, KNN mira cuáles son los vecinos más cercanos y lo clasifica según ellos.

Característica	**Regresión Logística**	**KNN**
Tipo de modelo	Paramétrico (usa una fórmula)	No paramétrico (basado en vecinos)
Frontera de decisión	Recta/lineal (sigmoide)	Puede ser muy flexible y no lineal
Velocidad de entrenamiento	Muy rápido (ajusta parámetros)	Muy rápido (solo guarda los datos)
Velocidad de predicción	Muy rápida	Lenta si hay muchos datos (calcula distancias)
Interpretación	Fácil (coeficientes)	Difícil (solo vecinos)
Sensibilidad al ruido	Más robusta	Muy sensible al ruido
Uso típico	Problemas donde hay separación más o menos lineal	Datos donde la frontera es irregular

¿Que es accuracy_score?

se usa para medir qué tan bien predice un modelo en comparación con las etiquetas reales.

Calcula la precisión (accuracy) de un modelo de clasificación.
La precisión es la proporción de predicciones correctas entre el total de predicciones.

$$\text{Accuracy} = \frac{\text{Número de predicciones correctas}}{\text{Total de predicciones}}$$

sirve para evaluar modelos de clasificación diciendo qué tan bien acierta el modelo al comparar sus predicciones contra la realidad.

Árboles de decisión

¿Que es un Árbol de Decisión?

Un árbol de decisiones es un diagrama en el que se representan las decisiones que se deben tomar, los diferentes escenarios que pueden ocurrir y todos los posibles resultados. Por lo tanto, un árbol de decisiones sirve como ayuda para tomar una decisión en el que se deben tener en cuenta varios escenarios posibles.

Se utiliza tanto para clasificación (predecir categorías) como para regresión (predecir valores numéricos).
Funciona como un árbol de preguntas: cada nodo hace una prueba sobre una característica y las ramas llevan a nuevos nodos o a una decisión final.

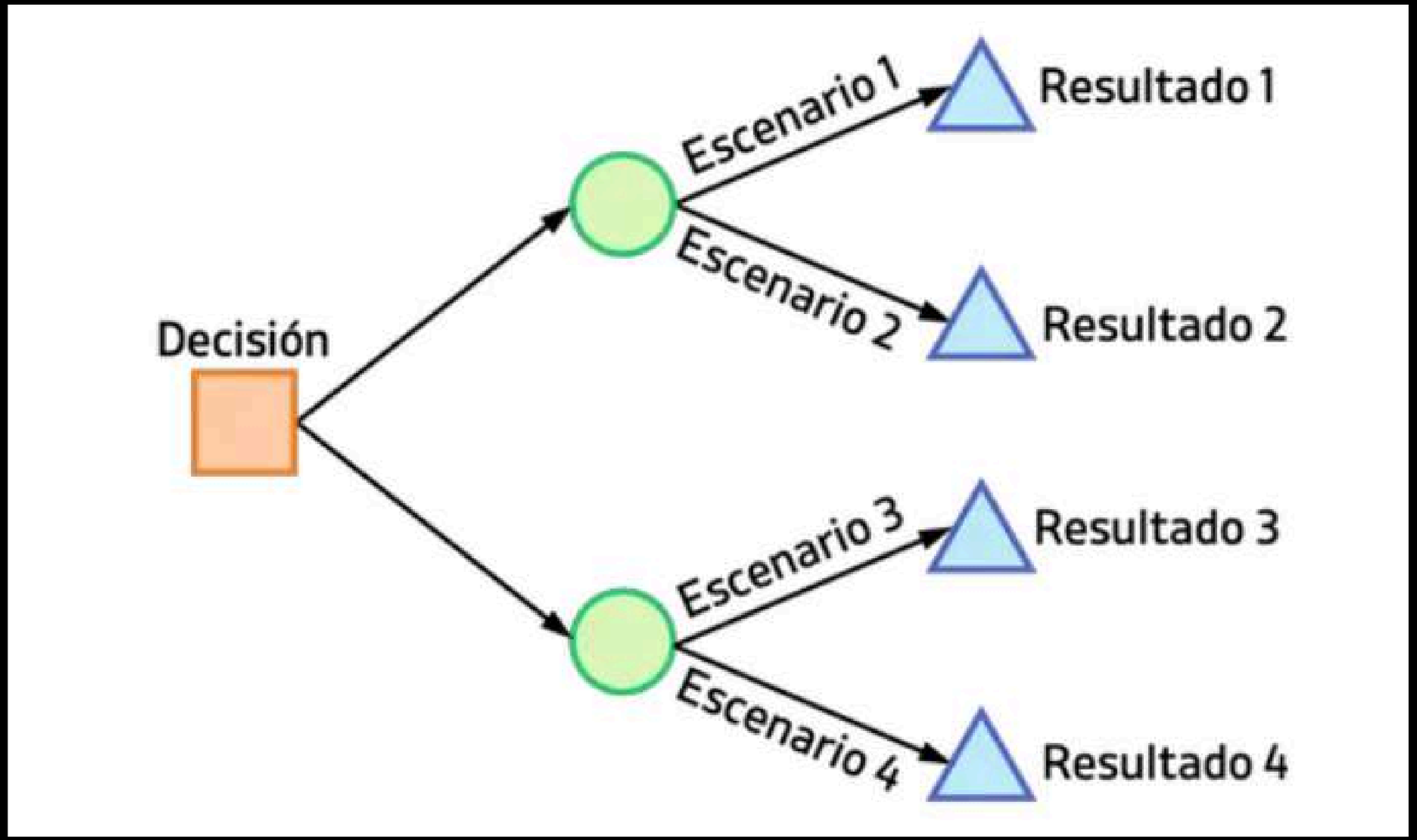
¿Que es un Árbol de Decisión?

- Nodo de decisión (□): corresponde a una decisión que se debe tomar. En el árbol de decisiones se representa como un cuadrado.

- Nodo de probabilidad (○): simboliza que pueden tener lugar varios escenarios, cada una de las ramas que sale de un nodo de probabilidad representa un escenario diferente. Se dibuja con un círculo vacío en el árbol de decisiones.

- Nodo final (△): representa un resultado, por lo tanto, se identifican fácilmente porque no sale ninguna rama de ellos. En el árbol de decisiones se representan mediante triángulos

Árbol de Decisión



Como hacer un Árbol de Decisión

- Representa la decisión principal: el primer paso para hacer un árbol de decisiones es representar la primera decisión que se debe tomar en el diagrama. Para ello, basta con dibujar un cuadrado y una flecha que salga del cuadrado por cada una de las opciones posibles que se pueden decidir.

- Añade nodos: en cada una de las ramas dibujadas en el paso anterior, expande el árbol de decisiones añadiendo nodos de decisión y de probabilidad.

Como hacer un Árbol de Decisión

- Llega hasta los resultados: ve agregando nodos de decisión y de probabilidad hasta que cada rama llegue a un nodo final o resultado. Cuando todos los caminos terminen en un resultado habrás acabado el árbol de decisiones.

- Toma una decisión: una vez ya has finalizado el árbol de decisiones, analízalo y decide qué es lo mejor que puedes hacer.

Como funciona Matematicamente

Cómo funciona matemáticamente

1. Se elige una característica y un umbral de división.
2. Se mide qué tan "buena" es esa división usando métricas como:
   - Entropía e Información Ganada (clasificación).
   - Índice Gini (clasificación).
   - Error cuadrático medio (regresión).
3. Se repite el proceso en cada nodo hasta:
   - Que se llegue a una predicción pura.
   - O que se cumpla un límite (profundidad máxima, número mínimo de muestras, etc.).

Entropía e Información Ganada (clasificación).

Ambos se usan para decidir qué atributo elegir en cada división del árbol.

La entropía mide el desorden o la impureza de un conjunto de datos.

- **Si un nodo tiene ejemplos de una sola clase → la entropía = 0 (puro).**
- **Si un nodo está 50% en cada clase → la entropía = 1 (máxima incertidumbre).**

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Donde:

- **c = número de clases.**
- **pi = proporción de ejemplos de la clase i.**

Entropía e Información Ganada (clasificación).

Supongamos un conjunto de 10 alumnos:

- **6 aprobaron (Sí)**
- **4 no aprobaron (No)**

$$H(S) = -\left(\frac{6}{10}\log_2\frac{6}{10} + \frac{4}{10}\log_2\frac{4}{10}\right)$$

$$H(S) = -(0.6 \cdot -0.737 + 0.4 \cdot -1.321)$$

$$H(S) = 0.971 \quad \text{(alto desorden)}$$

Entropía e Información Ganada (clasificación).

La ganancia de información mide cuánto reduce la entropía una división (una pregunta en el nodo).

$$IG(S, A) = H(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} H(S_v)$$

Donde:

- H(S) = entropía original del conjunto.
- A = atributo usado para dividir.
- Sv = subconjunto de ejemplos con el valor v de la característica A.

Entropía e Información Ganada (clasificación).

Dividimos por la pregunta: “¿Estudia más de 5 horas?”

- Grupo 1: ≤ 5 horas → 4 alumnos (todos reprobaron).
  - Entropía = 0 (puro).
- Grupo 2: > 5 horas → 6 alumnos (todos aprobaron).
  - Entropía = 0 (puro).

$$IG = H(S) - \left(\frac{4}{10} \cdot 0 + \frac{6}{10} \cdot 0\right)$$

$$IG = 0.971 - 0 = 0.971$$

Interpretación: esta división es perfecta, porque separa completamente a las clases (toda la entropía desaparece).

Índice Gini (clasificación).

El Índice Gini mide la impureza de un conjunto de datos (similar a la entropía).

- Si un nodo tiene ejemplos de una sola clase → Gini = 0 (puro).
- Si las clases están repartidas de forma equilibrada → Gini se acerca a 0.5 (máxima impureza en 2 clases).

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

Donde:

- c = número de clases.
- pi = proporción de ejemplos de la clase i.

Índice Gini (clasificación).

Supongamos un grupo de 10 alumnos:

- **7 aprobaron (Sí)**
- **3 no aprobaron (No)**

$$p_{Sí} = \frac{7}{10} = 0.7, \quad p_{No} = \frac{3}{10} = 0.3$$

$$Gini = 1 - (0.7^2 + 0.3^2)$$

$$Gini = 1 - (0.49 + 0.09) = 1 - 0.58 = 0.42$$

Este nodo tiene algo de impureza (mezcla de Sí y No)

Ventajas

- Los árboles de decisiones son fáciles de entender.

- Un árbol de decisiones permite visualizar de manera global todos los escenarios posibles y cuál es el resultado esperado en cada escenario.

- Este tipo de diagrama es muy eficiente, pues no se necesita invertir mucho tiempo para crearlo, sino que es rápido de hacer.

- Además, se pueden agregar nuevas ideas o escenarios al resultado, por lo que se trata de un diagrama flexible.

- Por último, el árbol de decisiones se puede combinar fácilmente con otras herramientas de decisión.

Desventajas

- Si el árbol de decisiones tiene muchos nodos de decisión o muchos escenarios posibles, puede volverse en un diagrama complejo.

- Muchas veces la probabilidad de que ocurra cada escenario no se puede determinar con exactitud, así que puede ser impreciso.

- El árbol de decisiones tan solo es una herramienta que ayuda a decidir, pero la decisión final la debe tomar alguien.

Ejercicio

Vamos a predecir si una persona aprueba un examen según sus horas de estudio y horas de sueño:

sklearn.tree import DecisionTreeClassifier, plot_tree

Parámetros más usados

- criterion: cómo medir la calidad de una división:
- "gini" (índice Gini, valor por defecto).
- "entropy" (entropía / información ganada).
- max_depth: la profundidad máxima del árbol (para evitar sobreajuste).
- min_samples_split: número mínimo de muestras necesarias para dividir un nodo.
- random_state: semilla para reproducir los resultados.

sklearn.tree import DecisionTreeClassifier, plot_tree

```
from sklearn.tree import DecisionTreeClassifier

# Creamos un árbol de decisión
modelo = DecisionTreeClassifier(criterion="gini", max_depth=3)

# Entrenamos con X (características) e y (etiquetas)
modelo.fit(X, y)

# Hacemos una predicción
prediccion = modelo.predict([[7, 6]])
print("Predicción:", prediccion)
```

plot_tree

Es una función que sirve para visualizar gráficamente el árbol de decisión que se entrenó.

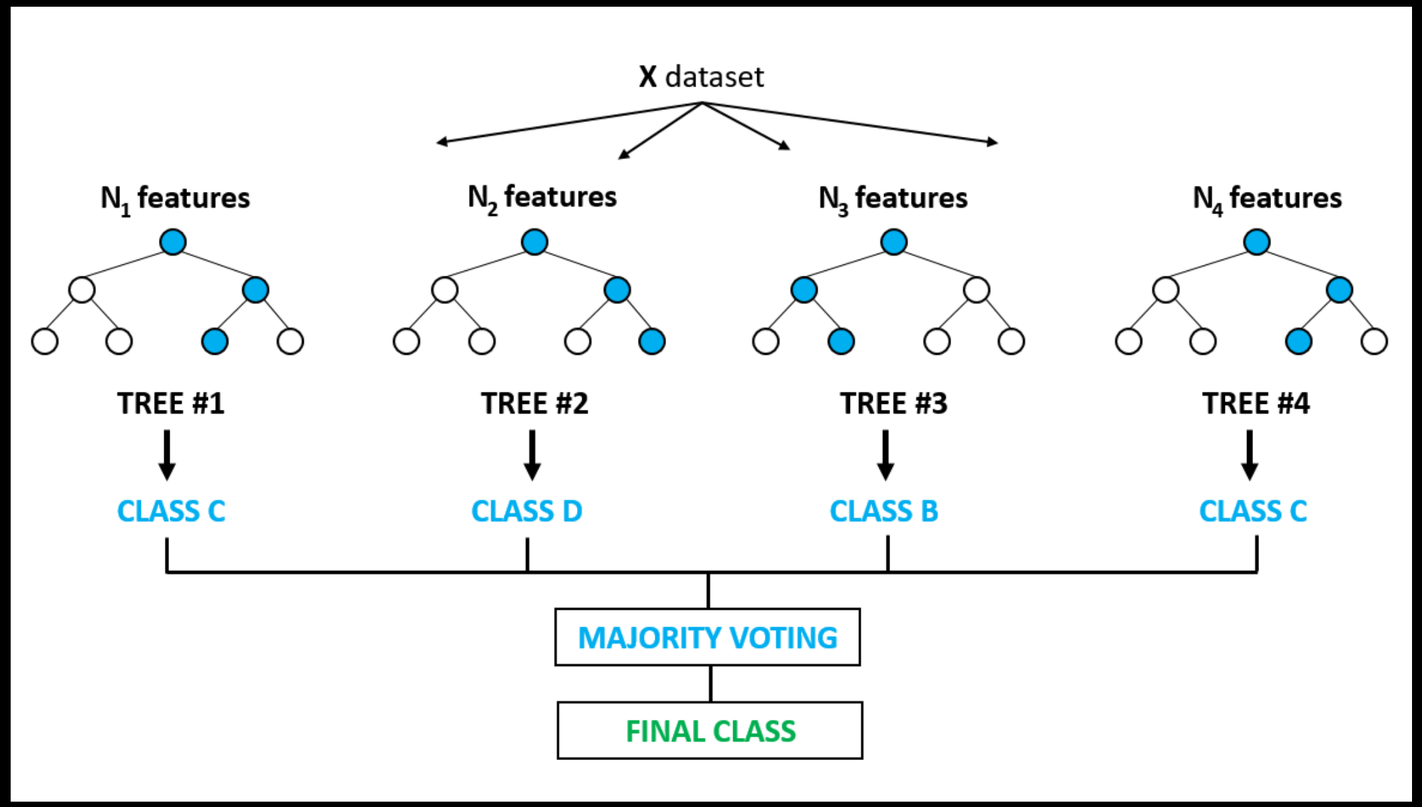
Muestra las divisiones, condiciones, y las clases que predice en cada nodo.

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plot_tree(modelo, feature_names=["Horas estudio", "Horas sueño"],
          class_names=["No aprueba", "Aprueba"], filled=True)
plt.show()
```

Random Forest

Random Forest (Bosque Aleatorio) es un algoritmo de Machine Learning que sirve tanto para clasificación como para regresión.
Se basa en la idea de combinar muchos árboles de decisión para obtener un modelo más robusto, preciso y menos propenso a errores.



¿Cómo funciona?

Construcción de múltiples árboles de decisión:

- El algoritmo crea varios árboles de decisión entrenados con diferentes subconjuntos de datos y características.
- Este proceso se llama bagging (bootstrap aggregating).

Aleatoriedad en la selección de características:

- En cada división del árbol, no se consideran todas las variables, sino un subconjunto aleatorio.
- Esto evita que los árboles sean muy parecidos entre sí.

¿Cómo funciona?

Combinación de resultados:

- **Clasificación → Cada árbol da un voto por una clase. La clase con más votos gana.**
- **Regresión → Se calcula el promedio de las predicciones de todos los árboles.**

Ventajas

- Generalmente tiene alta precisión.
- Reduce el problema de overfitting (sobreajuste) que suelen tener los árboles de decisión individuales.
- Funciona bien con muchos datos y variables.
- Da una idea de la importancia de las variables en el modelo.

Desventajas

- Puede ser más lento que un solo árbol, porque entrena muchos árboles.
- Es un modelo más difícil de interpretar (menos “explicable” que un árbol único).

las Redes Neuronales Artificiales (ANN, Artificial Neural Networks)

¿Qué es una red neuronal?

- Una red neuronal artificial es un modelo inspirado en cómo funciona el cerebro humano:
- Está formada por neuronas artificiales (también llamadas nodos o perceptrones).
- Estas neuronas se organizan en capas y están conectadas entre sí.
- Cada conexión tiene un peso que indica la importancia de la señal.

1. Capa de entrada (Input Layer)

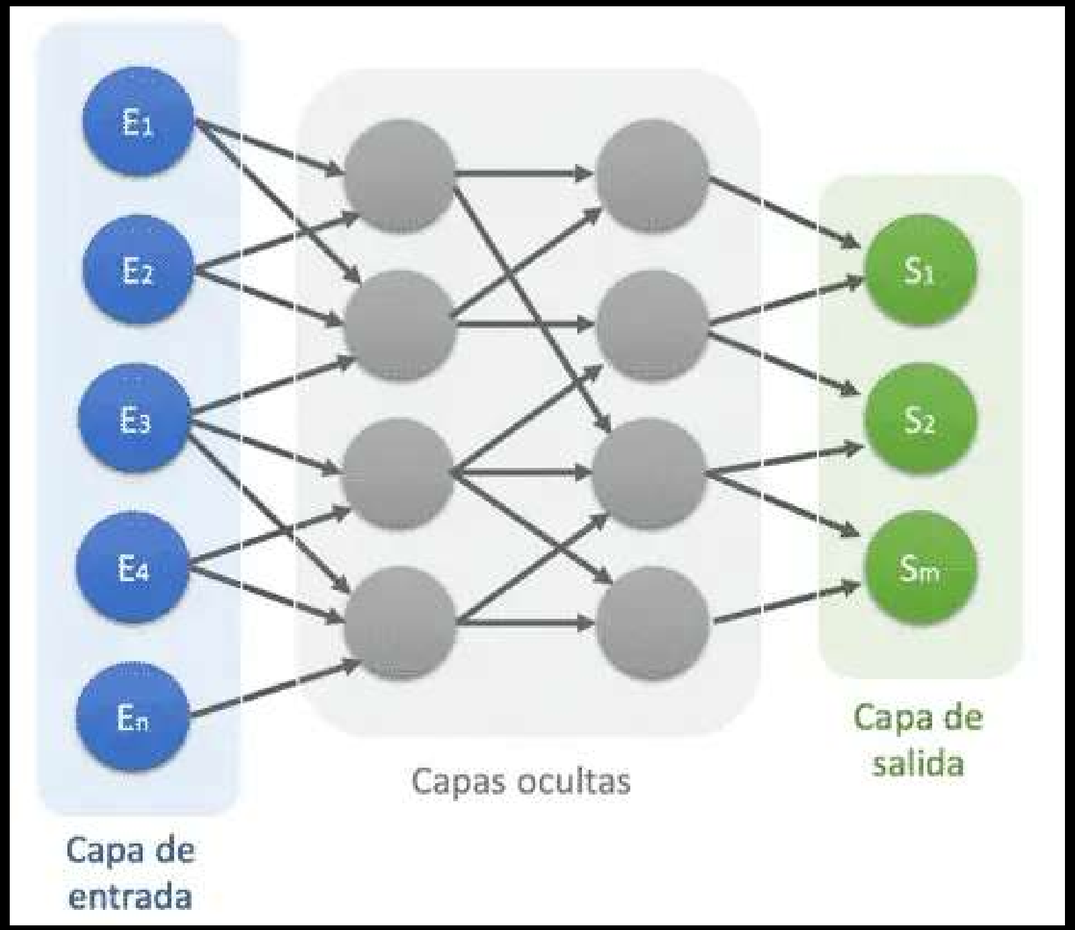
- Recibe las variables de entrada (ej. peso, tamaño, color de una fruta).

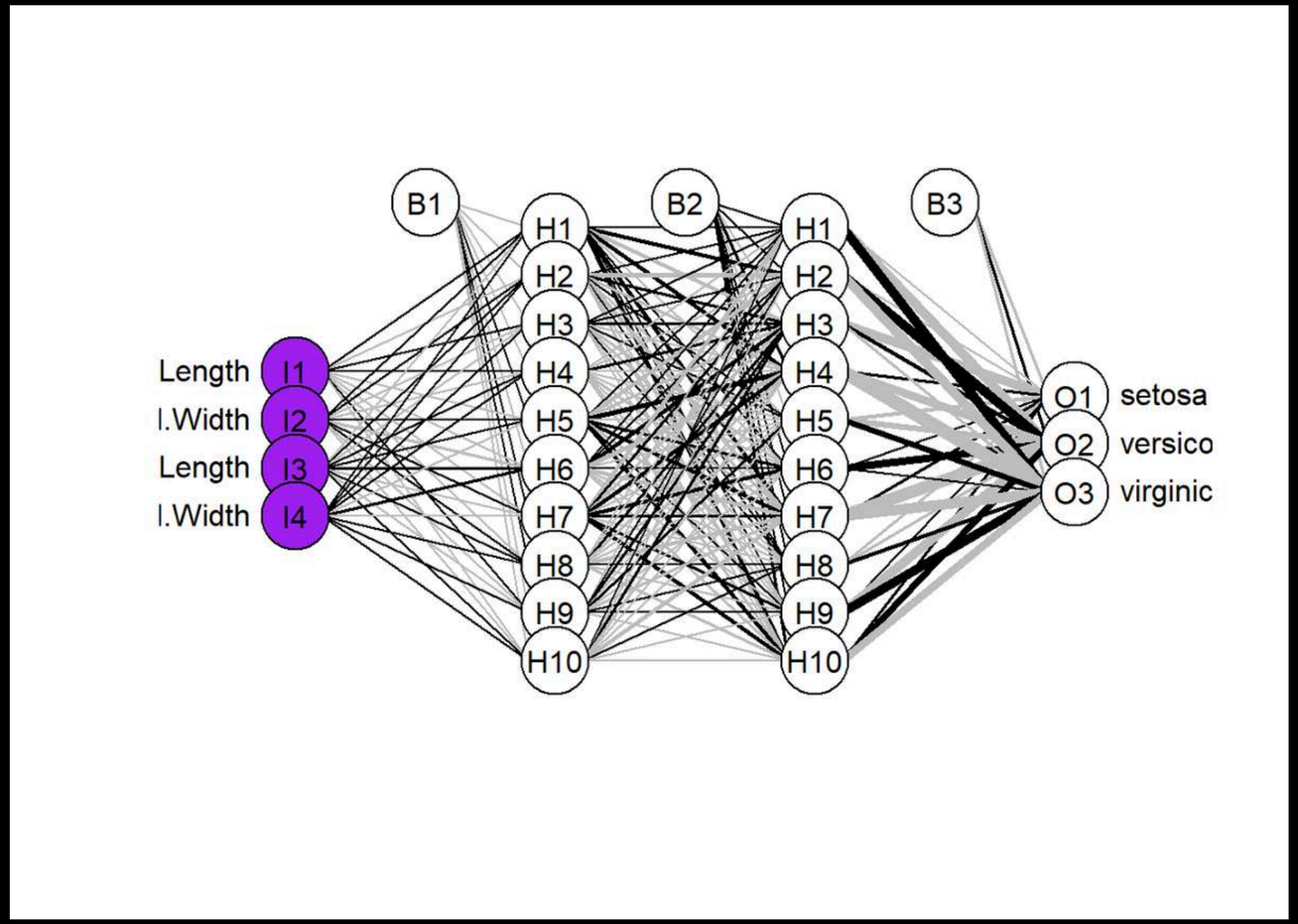
2. Capas ocultas (Hidden Layers)

- Transforman la información con operaciones matemáticas.
- Cada neurona aplica una combinación lineal de entradas + una función de activación (como sigmoide, ReLU, tanh).
- Esto permite modelar relaciones complejas y no lineales.

3. Capa de salida (Output Layer)

- Genera la predicción final.
- Ejemplo: probabilidad de que una fruta sea manzana o naranja.


E1
E2
E3
E4
En
S1
S2
Sm
Capa de entrada
Capas ocultas
Capa de salida


B1
B2
B3
Length
l.Width
Length
l.Width
I1
I2
I3
I4
H1
H2
H3
H4
H5
H6
H7
H8
H9
H10
H1
H2
H3
H4
H5
H6
H7
H8
H9
H10
O1
O2
O3
setosa
versico
virginic

Ventajas

- Capaces de aprender relaciones muy complejas.
- Funcionan muy bien en problemas grandes: visión por computadora, procesamiento de lenguaje natural, voz, etc.
- Son muy flexibles.

Desventajas

- Requieren muchos datos para entrenar bien.
- Consumen bastante poder de cómputo.
- Difíciles de interpretar (caja negra).

Los outliers o valores Atípicos

Un outlier (valor atípico) es un dato que se aleja mucho del resto de los valores en un conjunto de datos.

Son observaciones que no siguen el patrón general de la mayoría.

```
[1.60, 1.62, 1.65, 1.70, 1.68, 1.72, 1.66, 1.63, 1.61, 2.10]
```

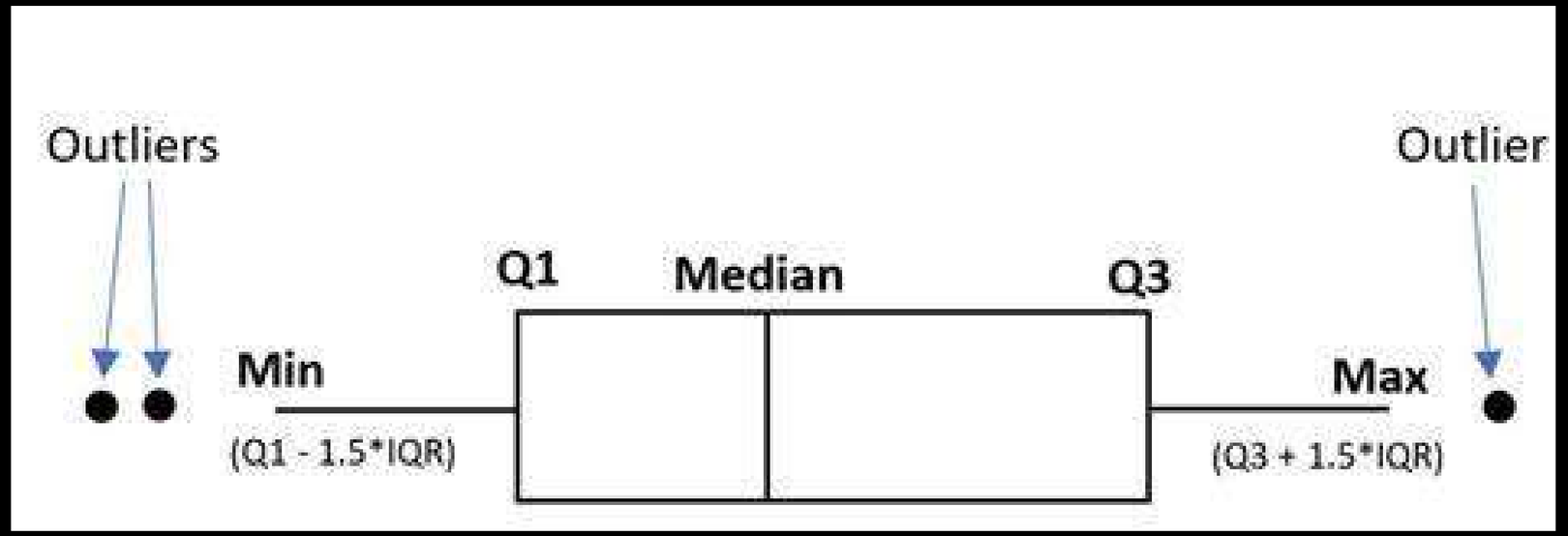
¿Por qué aparecen los outliers?

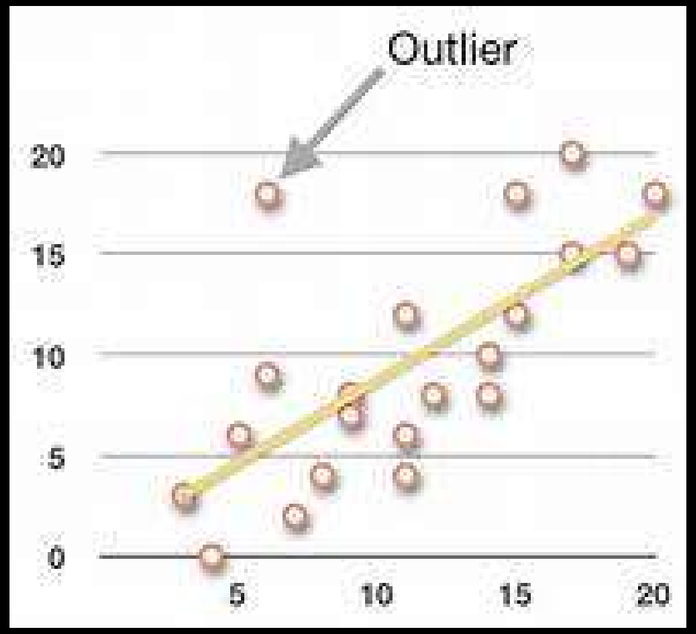
- Errores de medición o captura de datos.
- Variabilidad natural .
- Eventos inusuales.

¿Cómo detectarlos?

Existen varias formas:

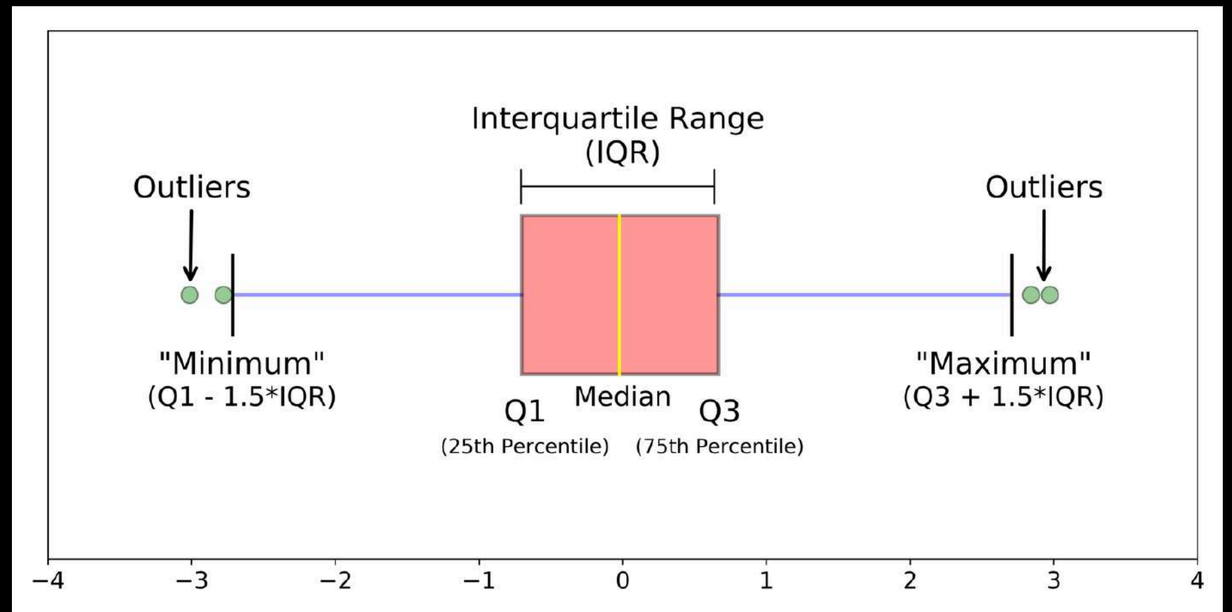
- **Visualmente (gráficos)**
- **Boxplot (diagrama de caja): los puntos alejados de los "bigotes" son outliers.**
- **Scatter plot (dispersión): se ven como puntos muy alejados.**





Medidas estadísticas

- **Regla de 1.5 × IQR (Rango intercuartílico)**
- **Si un dato está por debajo de Q1 - 1.5×IQR o por encima de Q3 + 1.5×IQR → es outlier.**
- **Z-score**
- **Si la puntuación típica (desviaciones respecto a la media) es mayor a ±3 → es outlier.**



Z-Score

Z-score mide cuántas desviaciones estándar se aleja un dato de la media:

$$Z = \frac{(x - \mu)}{\sigma}$$

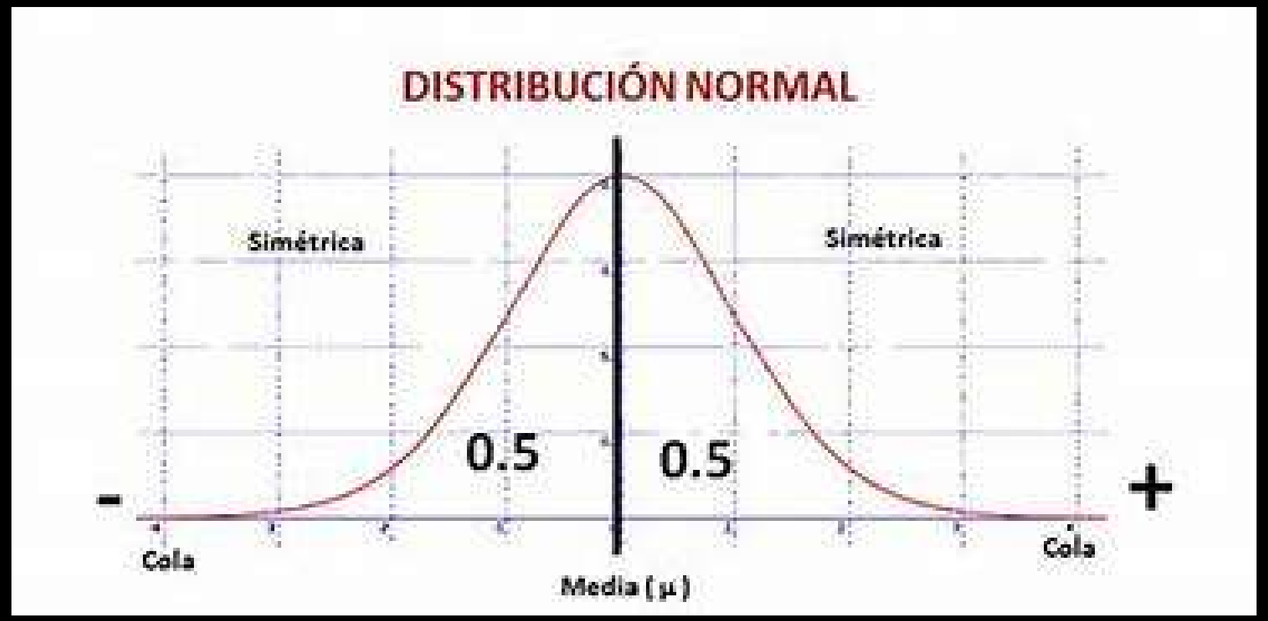
- **x = valor del dato.**
- **μ = media.**
- **σ = desviación estándar.**

Regla común:

Si Z > 3 → el dato está demasiado lejos de la media → se considera outlier.

Diferencia con IQR:

- IQR se basa en percentiles es más robusto cuando los datos no siguen distribución normal.
- Z-score funciona mejor cuando los datos son aproximadamente normales (campana de Gauss).



Método	Depende de	Mejor cuando	Ventaja	Desventaja
IQR	Percentiles (Q1, Q3)	Datos sesgados o no normales	Robusto a outliers	No mide la “distancia exacta” de un dato a la media
Z-score	Media y desviación estándar	Datos aproximadamente normales	Detecta outliers extremos	Sensible a sesgos y outliers ya presentes

¿Por qué son importantes?

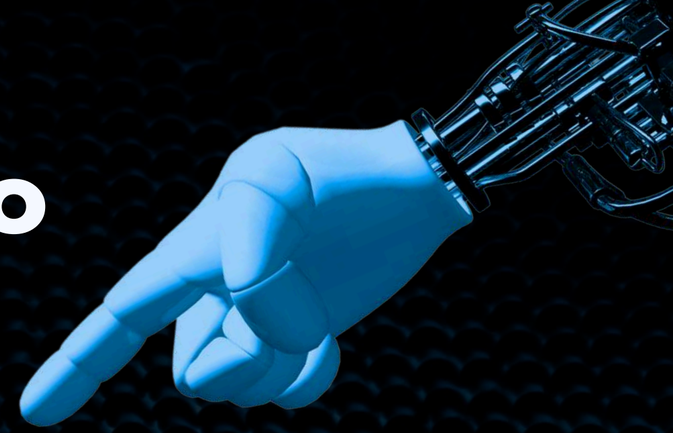
Los outliers pueden:

- Afectar al promedio (la media se mueve mucho si hay valores extremos).
- Influir en modelos de Machine Learning (pueden sesgar la predicción).
- Dar información valiosa (ej. detectar fraudes, fallas en máquinas o comportamientos anómalos).

¿Qué hacer con ellos?

- Eliminar : si sabemos que es un error.
- Corregir : si se trata de un error de digitación (ej. 210 → 1.70).
- Mantenerlos : representan casos reales importantes (ej. fraudes o clientes VIP).

Aprendizaje no supervisado (Unsupervised Learning)



Aquí los datos no tienen etiquetas. El objetivo es descubrir patrones ocultos o estructuras internas en los datos.
El modelo no sabe cuál es la respuesta correcta, solo recibe los datos y debe descubrir patrones o estructuras ocultas por sí mismo.

Cómo funciona el aprendizaje no supervisado.

Datos no etiquetados.

Se le da al modelo un conjunto de datos, pero sin decirle qué significan.
Ejemplo:
Imágenes sin etiquetas como "gato" o "perro".

Objetivo

El modelo intenta organizar o agrupar los datos buscando similitudes, estructuras o relaciones internas.

Áreas comunes de aprendizaje no supervisado

- Clustering (agrupamiento): Agrupar datos similares. Ejemplo: Agrupar clientes con comportamientos de compra parecidos (segmentación de mercado).

- Reducción de dimensionalidad: Simplificar los datos sin perder información importante.Ejemplo: Comprimir datos de muchas variables en pocas para visualizarlos mejor (como en PCA).

- Detección de anomalías: Encontrar datos que no se parecen a los demás. Ejemplo: Detectar transacciones bancarias sospechosas.

Ejemplos de aprendizaje no supervisado

Datos de entrada (sin etiqueta)	Tarea	Resultado esperado
Clientes y sus compras	Agrupamiento (clustering)	Grupos de clientes con intereses similares
Imágenes sin nombre	Agrupamiento de imágenes	Categorías de imágenes similares
Sensores en una fábrica	Detección de anomalías	Alertas por fallas o eventos raros
Muchas variables de un problema	Reducción de dimensionalidad	Visualización más clara y simplificada

Diferencia con el aprendizaje supervisado

Característica	Supervisado	No supervisado
Datos etiquetados	Sí	No
Objetivo	Predecir salidas específicas	Descubrir estructuras ocultas
Ejemplo típico	Clasificar correos como spam	Agrupar clientes por compras

Algoritmos comunes

Algoritmo	Uso Común
K-Means	Agrupamiento (clustering)
DBSCAN	Clustering con forma de grupos irregulares
Hierarchical Clustering	Clustering jerárquico
PCA (Análisis de Componentes Principales)	Reducción de dimensionalidad
Autoencoders	Compresión y reducción de ruido (deep learning)

Ejemplo: Tienda de ropa

Supongamos que tienes datos de clientes que visitan tu tienda.
Cada cliente tiene dos características:

- Edad.
- Cantidad de dinero gastado.

Cliente	Edad	Gasto
A	20	100
B	22	120
C	35	300
D	40	280
E	65	50
F	70	40

Datos (sin etiquetas visuales)

Caso 1: Aprendizaje Supervisado

Objetivo: Predecir si un cliente comprará o no comprará.

Se usan etiquetas como:

Cliente	Edad	Gasto	¿Compró?
A	20	100	Sí
B	22	120	Sí
C	35	300	Sí
D	40	280	Sí
E	65	50	No
F	70	40	No

Caso 2: Aprendizaje No Supervisado

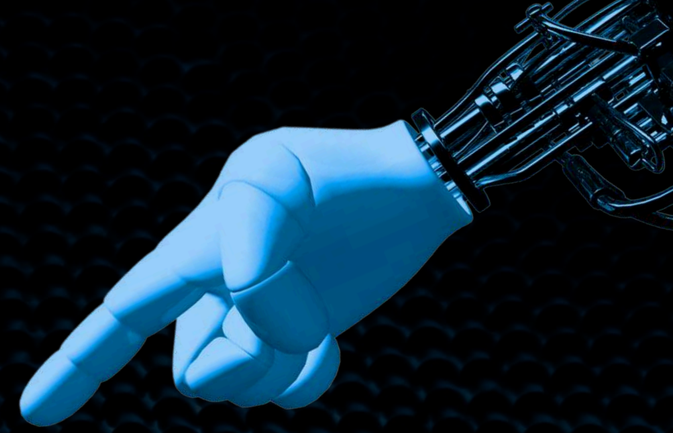
Objetivo: Descubrir grupos de clientes similares, sin saber si compraron o no.
Usamos solo esto:

Cliente	Edad	Gasto
A	20	100
B	22	120
C	35	300
D	40	280
E	65	50
F	70	40

Un algoritmo como K-means puede agruparlos así:

- 🟢 **Grupo 1: A, B → Jóvenes con poco gasto**
- 🔵 **Grupo 2: C, D → Adultos con alto gasto**
- 🔴 **Grupo 3: E, F → Mayores con bajo gasto**

Aprendizaje por refuerzo



Un agente aprende a tomar decisiones dentro de un entorno, recibiendo recompensas o castigos por sus acciones. Aprende por ensayo y error.

¿Como funciona?

- Agente: El que toma decisiones (puede ser un robot, un programa, un videojuego, etc.).
- Entorno: El lugar o situación donde actúa el agente (un juego, una fábrica, una ciudad simulada, etc.).
- Acciones: Las decisiones que puede tomar el agente (moverse, comprar, saltar, girar, etc.).

¿Como funciona?

- Recompensa: Un número que indica qué tan buena fue una acción.

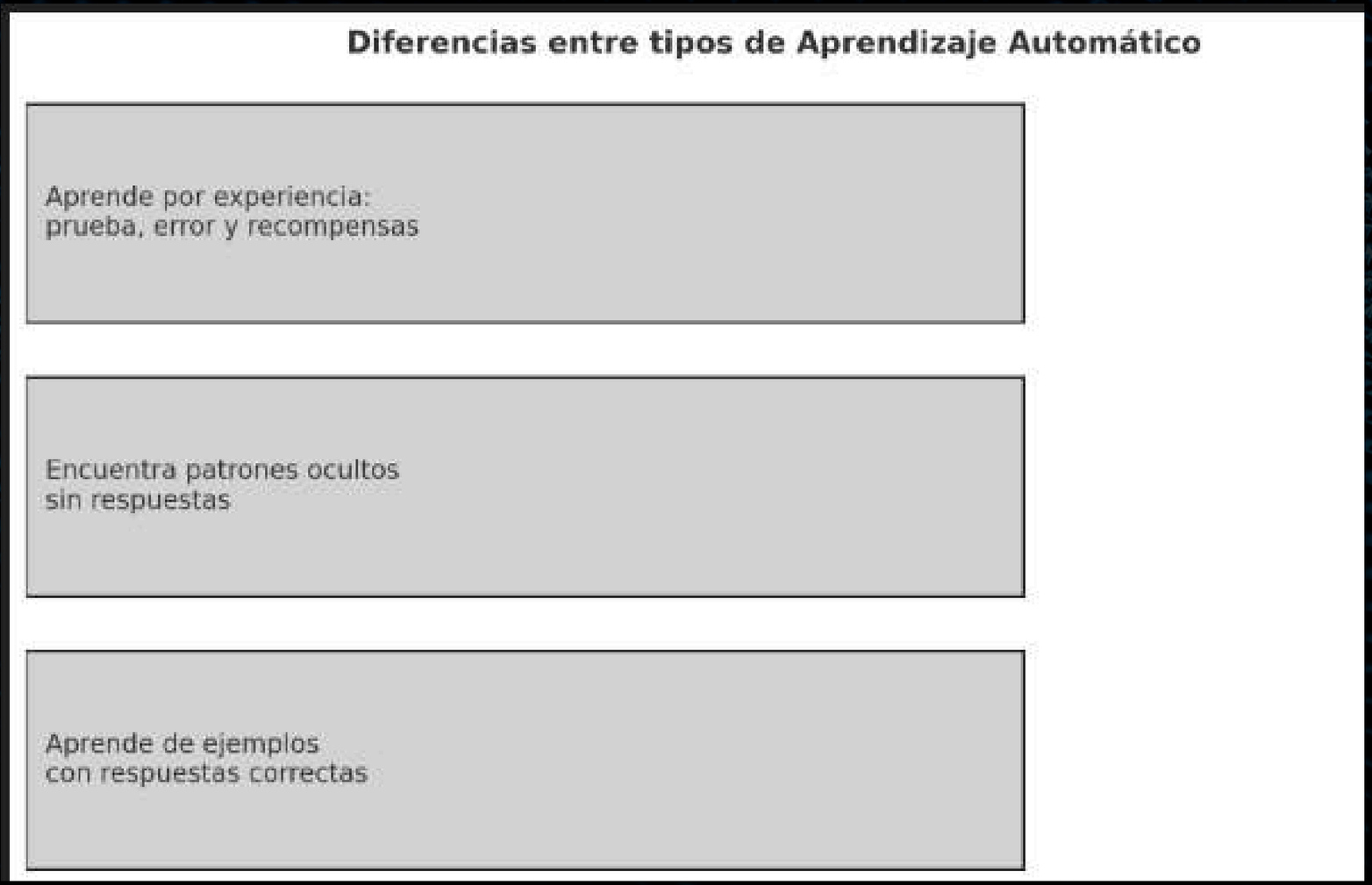
  Recompensa positiva → Buen resultado.

  Recompensa negativa (o castigo) → Mal resultado.
- Objetivo: Aprender una política (policy) que maximice la recompensa total a largo plazo.

Ejemplo: Un robot en un laberinto

- Agente: El robot.
- Entorno: El laberinto.
- Acciones: Avanzar, girar a la izquierda, girar a la derecha.
- Recompensa:
- +10 puntos si llega a la salida.
- -1 punto por cada paso que da (para que no se quede dando vueltas).
- -10 puntos si choca contra una pared.

- **El robot empieza sin saber nada y prueba diferentes movimientos. Con el tiempo, aprende el camino más rápido para salir y así maximizar su recompensa.**



Algoritmos comunes

Algoritmo	Uso Común
Q-Learning	Juegos, navegación, decisiones secuenciales
Deep Q-Networks (DQN)	Videojuegos, control robótico
Policy Gradient Methods	Control continuo, robótica
Proximal Policy Optimization (PPO)	Avanzado, usado en IA de juegos

Algoritmo	Ventajas	Desventajas	Ejemplo de uso
Regresión Lineal	Simple, interpretable, rápido	Supone relación lineal, sensible a valores atípicos	Predecir precios de casas
Regresión Logística	Buena para clasificación binaria, interpretabilidad	No funciona bien con relaciones no lineales complejas	Detección de fraude (sí/no)
KNN (K-Nearest Neighbors)	Intuitivo, sin entrenamiento complejo	Lento con grandes volúmenes de datos, sensible al ruido	Clasificación de especies según medidas
Árbol de Decisión	Interpretable, fácil de visualizar	Tiende a sobreajustar	Diagnóstico médico
Random Forest	Preciso, maneja bien datos faltantes/noise	Difícil de interpretar, más lento que un árbol simple	Clasificación de clientes

SVM (Support Vector Machine)	Efectivo en espacios de alta dimensión	Costoso en tiempo y memoria, difícil de ajustar	Clasificación de texto
Redes Neuronales	Potente para datos no estructurados (imagen, voz, texto)	Requiere muchos datos y recursos, difícil de interpretar	Reconocimiento facial, traducción automática
XGBoost / LightGBM	Muy preciso, eficiente, usado en competencias	Puede sobreajustar si no se ajusta bien	Predicción de riesgo crediticio
K-Means (Clustering)	Rápido y eficiente para agrupamiento	Necesita elegir k, no maneja bien clusters de formas irregulares	Segmentación de clientes
PCA	Reduce dimensiones, acelera otros algoritmos	Puede perder interpretabilidad	Visualización de datos de alta dimensión
Q-Learning	Aprende estrategias óptimas	Requiere muchas interacciones (simulación o entorno)	Juegos, robótica
DQN (Deep Q Network)	Aprendizaje profundo + refuerzo	Complejo de entrenar, necesita entorno simulado	Juegos (ej. jugar Atari)

Fundamentos Matemáticos y Estadísticos.

Álgebra lineal

Esencial porque los datos se representan como vectores y matrices.

- Representación de datasets como matrices.
- Operaciones con tensores (en deep learning).
- Descomposiciones .
- Producto punto y normas (medidas de distancia en KNN, *SVM*).

Cálculo diferencial e integral

Permite optimizar funciones de costo.

- Derivadas : encontrar mínimos y máximos.
- Gradiente : usado en descenso por gradiente (entrenamiento de modelos).
- Gradiente descendente estocástico (SGD) : entrenamiento de redes neuronales.

Optimización matemática

- Programación convexa : problemas donde el costo tiene forma convexa (asegura mínimos globales).
- Regularización (L1, L2) : técnicas para evitar sobreajuste.

Teoría de la información

- Entropía : mide incertidumbre.
- Información mutua : mide dependencia entre variables.
- Cross-Entropy Loss : función de pérdida usada en clasificación.

Fundamentos matemáticos

Álgebra lineal

El álgebra lineal es una de las bases matemáticas más importantes en Machine Learning (ML) porque la mayoría de los modelos y algoritmos trabajan con vectores, matrices y operaciones lineales.

Fundamentos matemáticos

Representación de los datos

- En ML, los datos casi siempre se representan en forma de vectores (características de un ejemplo) o matrices (dataset completo).
- Ejemplo: Un dataset de imágenes : cada imagen se convierte en un vector de píxeles.
- Conjunto de datos con m muestras y n características : se representa como una matriz M X N

Ejemplo: Dataset de estudiantes

Supongamos que tenemos un dataset con información de 3 estudiantes y 4 características:

- **Edad**
- **Horas de estudio por semana**
- **Promedio de calificaciones**
- **Asistencia (%)**

Estudiante	Edad	Horas estudio	Promedio	Asistencia
1	20	10	85	90
2	22	15	92	95
3	19	8	78	85

Representación como vectores (un estudiante)

Cada estudiante es un vector fila con sus características.
Por ejemplo, el estudiante 1 se representa como:

$$x^{(1)} = [20,\ 10,\ 8.5,\ 90]$$

Representación como matriz (todo el dataset)

El conjunto completo de estudiantes se representa como una matriz de dimensión. M x N: 3 x 4

$$X = \begin{bmatrix} 20 & 10 & 8.5 & 90 \\ 22 & 15 & 9.2 & 95 \\ 19 & 8 & 7.8 & 85 \end{bmatrix}$$

- m = 3 (número de estudiantes = muestras)
- n = 4 (número de características)

Transformaciones lineales

Representación de los datos

- En Machine Learning, trabajamos con datos representados como vectores (ejemplos con características).
- Una transformación lineal es una operación que toma esos vectores y los transforma mediante matrices y vectores.

Rotar una imagen : aplicar una matriz de rotación.

Escalar una característica : multiplicar por un número.

Combinar características : hacer sumas ponderadas con pesos.

La ecuación en redes neuronales

En una neurona artificial, la entrada x (vector de características) se transforma así:

$$y = W \cdot x + b$$

- x: vector de entrada (datos).
- W: matriz de pesos (qué tan importante es cada característica).
- b: sesgo (bias), un valor extra que ajusta el resultado.
- y: salida transformada.

Ejemplo con números

Imagina que queremos predecir el desempeño de un estudiante con 2 características:
x1 = horas de estudio
x2 = horas de sueño
Entonces:

$$x = \begin{bmatrix} 10 \\ 8 \end{bmatrix}$$

Ahora, supongamos que los pesos aprendidos son:

$$W = \begin{bmatrix} 0.5 & 0.2 \end{bmatrix} \quad \text{y} \quad b = 1$$

Formula

$$y = W \cdot x + b$$

Datos

$$x = \begin{bmatrix} 10 \\ 8 \end{bmatrix}$$

$$W = \begin{bmatrix} 0.5 & 0.2 \end{bmatrix} \quad \text{y} \quad b = 1$$

Calculamos

$$y = W \cdot x + b = (0.5)(10) + (0.2)(8) + 1$$

$$y = 5 + 1.6 + 1 = 7.6$$

Medidas de similitud y distancia

Cuando tenemos vectores de datos (ejemplos), necesitamos saber qué tan parecidos o diferentes son.
Esto lo hacemos con productos internos (similitud) o normas/distancias (diferencia).

1. Producto punto (dot product)

El producto punto entre dos vectores mide qué tan alineados están.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

- Si el resultado es grande y positivo : vectores apuntan en direcciones similares.
- Si es cero : vectores son ortogonales (no relacionados).
- Si es negativo : vectores apuntan en direcciones opuestas.

2. Norma Euclidiana (distancia L2)

Es la distancia "normal" entre dos puntos en el espacio:

$$d(a,b) = \|a-b\|_2 = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Intuición: mide qué tan lejos están los puntos.

Usos:

- **KNN (K-Nearest Neighbors): elige los vecinos más cercanos según esta distancia.**
- **Clustering (ej. K-Means): agrupa puntos que están cerca.**

Ejemplo práctico con vectores

Supongamos dos estudiantes representados por:

x1 = [10,8] → (10 horas de estudio, 8 horas de sueño)

x2 = [8,7] → (8 horas de estudio, 7 horas de sueño)

$$x_1 \cdot x_2 = (10)(8) + (8)(7) = 80 + 56 = 136$$

Distancia Euclidiana:

$$d(x_1, x_2) = \sqrt{(10-8)^2 + (8-7)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$

Interpretación:

- **Producto punto alto (136) → vectores bastante alineados, estudiantes con hábitos similares.**
- **Distancia pequeña (2.24) → están cercanos en el espacio de características.**

Conexión con ML

- **En KNN, se busca la menor distancia Euclidiana para encontrar los vecinos más cercanos.**
- **En embeddings de texto/imágenes, se usa producto punto o coseno para medir similitud.**
- **En clustering, se agrupan vectores cercanos en términos de distancia.**

Optimización en Machine Learning

Cuando entrenamos un modelo de ML (red neuronal, regresión lineal, etc.), lo que hacemos en el fondo es un problema de optimización:

$$\min_{\theta} \; J(\theta)$$

θ = parámetros del modelo (pesos W y sesgos b)
J(θ) = función de costo (qué tan mal predice el modelo).

El objetivo es ajustar los parámetros para minimizar el error.

1. Descenso de Gradiente (Gradient Descent)

La idea es simple:

- **Se calculan los gradientes de la función de costo respecto a los parámetros**

$$\nabla_\theta J(\theta)$$

Esto nos dice en qué dirección aumenta el error.

- **Se actualizan los parámetros en la dirección contraria al gradiente.**

$$\theta := \theta - \eta \cdot \nabla_\theta J(\theta)$$

η **= tasa de aprendizaje (learning rate).**

Procesamiento eficiente

Las operaciones de álgebra lineal se pueden paralelizar en GPU/TPU (usando librerías como BLAS, NumPy, TensorFlow, PyTorch).
Esto hace que entrenar modelos complejos (redes neuronales profundas) sea posible.

Cálculo diferencial e integral

El cálculo diferencial estudia tasas de cambio y es clave en el entrenamiento de modelos.

a) Derivadas para optimización
Los modelos de ML ajustan parámetros (pesos) minimizando una función de costo.
Para encontrar el mínimo, usamos derivadas.
Ejemplo: Regresión lineal : minimizar el Error Cuadrático Medio (MSE).
Se calcula la derivada de la función de error respecto a cada parámetro.

$$\frac{\partial}{\partial w}\,\mathrm{MSE}(w) = 0 \quad \Rightarrow \quad w^* = \text{parámetro óptimo}$$

b) Gradientes

- En modelos con muchos parámetros, usamos el gradiente (vector de derivadas parciales).
- Gradient Descent es el corazón del aprendizaje automático:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

donde L(w) es la función de pérdida y η es la tasa de aprendizaje.

c) Funciones de activación

En redes neuronales, las funciones como sigmoide, ReLU, tanh requieren derivadas para propagar el error hacia atrás (backpropagation).

Cálculo integral

El cálculo integral aparece en situaciones donde necesitamos acumular probabilidades, áreas o esperanzas matemáticas.

a) Probabilidad y distribuciones
Muchas funciones de probabilidad se definen con integrales.
Ejemplo: La función de densidad de la normal:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

se integra para calcular probabilidades:

$$P(a \leq X \leq b) = \int_a^b f(x)\,dx$$

b) Esperanza y varianza

La media esperada de una variable continua se calcula con integrales:

$$E[X] = \int_{-\infty}^{\infty} x f(x)\, dx$$

Lo mismo con la varianza, entropía y medidas de información.

c) Regularización y áreas bajo curvas

En aprendizaje estadístico, algunas técnicas de regularización o métricas (ej. AUC-ROC) implican integrales.

Optimización matemática

1. Optimización matemática en ML

En Machine Learning, entrenar un modelo significa resolver un problema de optimización:

$$\min_{\theta} L(\theta)$$

donde:

- theta θ = parámetros del modelo (pesos, coeficientes, etc.)
- L(θ) = función de pérdida (qué tan mal predice el modelo).

Ejemplo: en regresión lineal,

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Regularización (para evitar sobreajuste)

Cuando un modelo es demasiado flexible, aprende ruido → overfitting.
Para controlarlo, se añade un término de penalización en la función de costo.

a) Regularización L2 (Ridge)

$$L(\theta) = \text{Error}(y, \hat{y}) + \lambda \sum_j \theta_j^2$$

- Penaliza los pesos grandes.
- Tiende a repartir los valores entre todas las características.
- Útil cuando todas las variables aportan un poco.

b) Regularización L1 (Lasso)

$$L(\theta) = \text{Error}(y, \hat{y}) + \lambda \sum_j |\theta_j|$$

- Penaliza el valor absoluto → muchos coeficientes se vuelven cero.
- Hace selección de variables automática.
- Útil cuando solo unas pocas características son relevantes.

c) Elastic Net
Combina L1 y L2:

$$L(\theta) = \text{Error}(y, \hat{y}) + \lambda_1 \sum_j |\theta_j| + \lambda_2 \sum_j \theta_j^2$$

Teoría de la información

Entropía (mide incertidumbre)

Propuesta por Claude Shannon (1948).
Mide la incertidumbre promedio de una variable aleatoria.
Fórmula:

$$H(X) = -\sum_i p(x_i) \log_2 p(x_i)$$

Si un evento es seguro (p=1), la entropía es 0 (no hay incertidumbre).
Si todos los eventos son equiprobables, la entropía es máxima

En ML: la entropía se usa en árboles de decisión para medir la pureza de los nodos.

2. Información mutua (mide dependencia)

Mide cuánta información comparte una variable con otra.
Fórmula:

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

Interpretación:
Si X y Y son independientes → I(X;Y)=0.
Si X determina perfectamente a Y → la información mutua es máxima.

En ML: se usa para selección de características (elegir las variables más informativas respecto a la clase).

3. Cross-Entropy Loss (pérdida en clasificación)

- Muy usada en clasificación supervisada, especialmente en redes neuronales.
- Mide la diferencia entre la distribución real y la predicha por el modelo.

Fórmula para clasificación binaria:

$$L = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

Generalizada para multiclase:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

yi = etiqueta real (one-hot encoded).
y^i = probabilidad predicha por el modelo para la clase i.

Métricas comunes

Clasificación

Problemas donde las salidas son categorías (ej. "spam" vs "no spam").

Accuracy (Exactitud)

Es una métrica de evaluación que mide qué tan bien funciona el modelo.
Se define como la proporción de predicciones correctas entre el total de predicciones.

$$\text{Accuracy} = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}}$$

Ejemplo:

Supongamos que un modelo clasifica si un correo es spam o no spam.
De 10 correos:
El modelo acierta en 8.
Se equivoca en 2.

$$\text{Accuracy} = \frac{8}{10} = 0.8 = 80\%$$

Clasificación = predecir clases.
Accuracy = qué porcentaje de esas predicciones son correctas.

Precision (Precisión)

Se usa cuando nos interesa qué tan confiables son las predicciones positivas del modelo.
Responde a la pregunta:

De todos los casos que el modelo dijo que eran positivos, ¿cuántos realmente lo eran?

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

donde:

- TP (True Positives) = verdaderos positivos (el modelo predijo “positivo” y era correcto).
- FP (False Positives) = falsos positivos (el modelo predijo “positivo”, pero en realidad era negativo).

Ejemplo

Un modelo que detecta correos spam.
El modelo predijo 50 correos como spam.
De esos 50, 40 sí eran spam (TP).
Pero se equivocó en 10 que en realidad no eran spam (FP).

$$\text{Precision} = \frac{40}{40 + 10} = \frac{40}{50} = 0.8 = 80\%$$

- Alta precisión : cuando el modelo dice “positivo”, casi siempre tiene razón.
- Baja precisión : el modelo se equivoca seguido marcando negativos como positivos (muchos falsos positivos).

Recall (Sensibilidad o Cobertura)

- Nos dice qué tan bien el modelo detecta los positivos reales.
- Responde a la pregunta:

De todos los casos que eran positivos en realidad, ¿cuántos detectó el modelo como positivos?

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

Donde

- TP (True Positives) = verdaderos positivos (el modelo predijo “positivo” y era correcto).
- FN (False Negatives) = falsos negativos (el modelo predijo “negativo”, pero en realidad era positivo).

Ejemplo:

Volvamos al detector de correos spam.

- **En realidad había 60 correos spam (positivos reales).**
- **El modelo detectó correctamente 40 (TP).**
- **Pero dejó pasar 20 spam como si no fueran spam (FN).**

$$\text{Recall} = \frac{40}{40 + 20} = \frac{40}{60} = 0.66 = 66\%$$

- Alta Recall : el modelo encuentra casi todos los positivos (pocos falsos negativos).
- Baja Recall : el modelo deja escapar muchos positivos sin detectarlos.

Diferencia clave con Precision:
Precision = ¿de los que marqué como positivos, cuántos eran correctos?
Recall = ¿de todos los positivos reales, cuántos logré encontrar?

F1-Score

- Es una métrica que combina Precision y Recall en una sola medida.
- Se usa mucho cuando las clases están desbalanceadas (por ejemplo, 95% "no spam" y solo 5% "spam").
- Se define como la media armónica de Precision y Recall, lo que penaliza mucho si una de las dos es baja.

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

Ejemplo práctico:

Supongamos que tenemos:

Precision = 0.80 (80%)

Recall = 0.66 (66%)

Entonces:

$$F1 = 2 \cdot \frac{0.80 \cdot 0.66}{0.80 + 0.66} = 2 \cdot \frac{0.528}{1.46} \approx 0.72$$

El F1-score ≈ 72%

Interpretación:

Un modelo con F1 alto tiene buen equilibrio entre no dar muchos falsos positivos (alta Precision) y no dejar escapar muchos positivos reales (alta Recall).
Si el dataset está balanceado, la Accuracy puede ser suficiente.
Pero si hay desbalance (ejemplo: detectar fraudes, enfermedades raras, spam), el F1-score es más justo para evaluar.

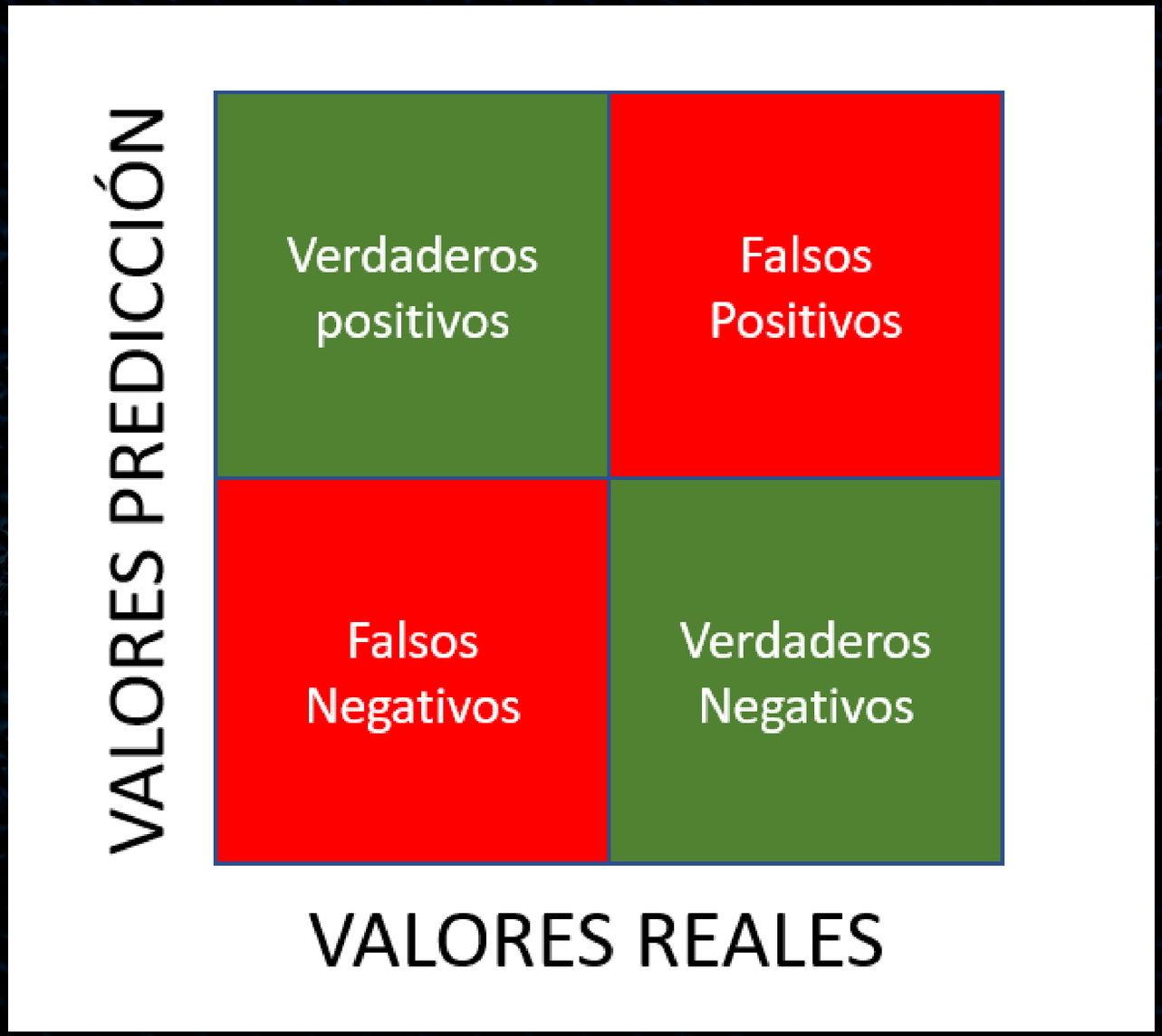
Matriz de Confusión

Es una tabla que resume los resultados de un modelo de clasificación, comparando las predicciones del modelo con las etiquetas reales.

	Predicho Positivo	**Predicho Negativo**
Real Positivo	**TP** (True Positive)	**FN** (False Negative)
Real Negativo	**FP** (False Positive)	**TN** (True Negative)

Interpretación de cada celda:

- TP (Verdaderos Positivos): el modelo predijo positivo y era positivo.
- TN (Verdaderos Negativos): el modelo predijo negativo y era negativo.
- FP (Falsos Positivos): el modelo predijo positivo, pero en realidad era negativo (“falsa alarma”).
- FN (Falsos Negativos): el modelo predijo negativo, pero en realidad era positivo (“se le escapó un positivo”).


VALORES PREDICCIÓN
Verdaderos positivos
Falsos Positivos
Falsos Negativos
Verdaderos Negativos
VALORES REALES

	Predicho Spam	Predicho No Spam
Real Spam	40 (**TP**)	10 (**FN**)
Real No Spam	5 (**FP**)	45 (**TN**)

- **Accuracy = (TP + TN) / Total = (40 + 45) / 100 = 85%**
- **Precision = TP / (TP + FP) = 40 / (40 + 5) = 88.9%**
- **Recall = TP / (TP + FN) = 40 / (40 + 10) = 80%**
- **F1-score = 2 × (Precision × Recall) / (Precision + Recall) ≈ 84.2%**

AUC-ROC (Area Under Curve – Receiver Operating Characteristic)

Es una curva que muestra el rendimiento del modelo a distintos umbrales de decisión (threshold).
En vez de dar un solo número, la curva traza la relación entre:

TPR (True Positive Rate o Recall o Sensibilidad):

$$TPR = \frac{TP}{TP + FN}$$

proporción de positivos reales que el modelo detecta.

FPR (False Positive Rate):

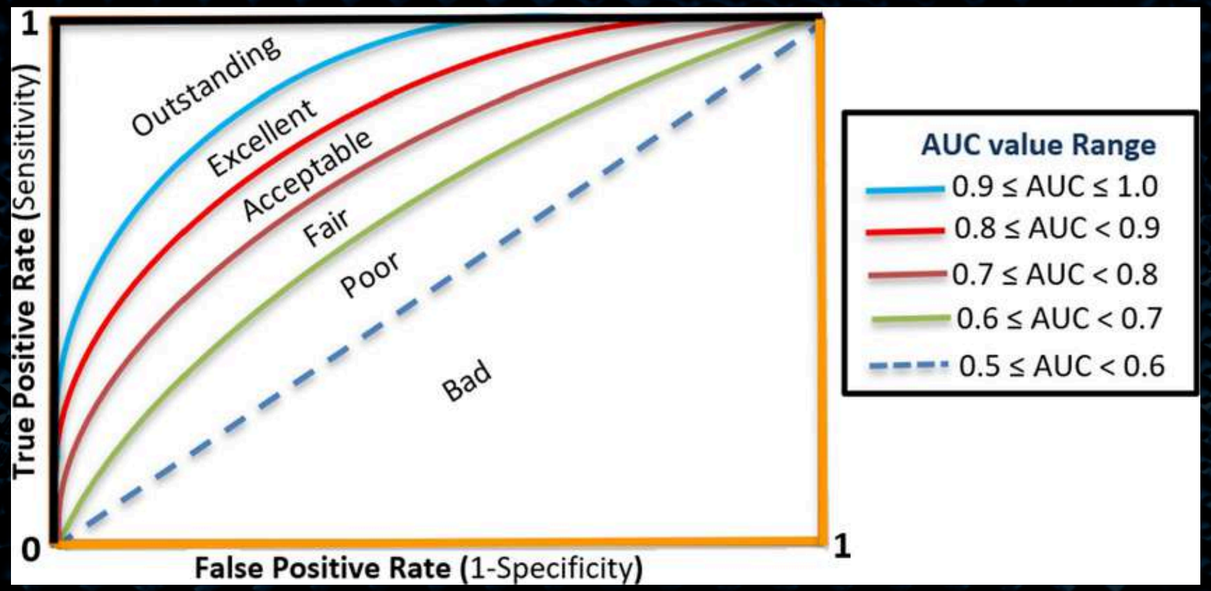
$$FPR = \frac{FP}{FP + TN}$$

proporción de negativos reales que el modelo clasificó mal como positivos.

AUC (Area Under Curve)

El AUC es el área bajo la curva ROC.

Mide qué tan bien el modelo distingue entre clases (positiva vs negativa).



- Interpretación de valores:
- AUC = 1.0 , Modelo perfecto (distingue siempre bien entre clases).
- AUC = 0.5 , Modelo inútil, como lanzar una moneda (predicción aleatoria).
- AUC < 0.5 , Peor que el azar (el modelo confunde más de lo que acierta).

Regresión

Problemas donde la salida es un número continuo (ej. predecir precio de una casa).

Error absoluto medio (MAE):
Promedio del error absoluto

$$MAE = \frac{1}{n}\sum |y_i - \hat{y}_i|$$

donde:

yi = valor real

y^i = valor predicho por el modelo

n = número de observaciones

- Es una métrica que mide qué tan lejos están las predicciones de los valores reales, en promedio.
- Se calcula tomando la diferencia entre cada valor real y su predicción, en valor absoluto, y luego sacando el promedio.

Error cuadrático medio (MSE)

Mide el promedio de los errores al cuadrado entre los valores reales y las predicciones.
Se parece al MAE, pero aquí se elevan al cuadrado las diferencias antes de promediarlas.

$$MSE = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

donde:
yi = valor real
y^i = predicción
nnn = número de datos

Raíz del Error Cuadrático Medio (RMSE – Root Mean Squared Error)

Es simplemente la raíz cuadrada del MSE.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

La ventaja principal es que vuelve la métrica a las mismas unidades que la variable objetivo, lo que hace que sea más fácil de interpretar que el MSE.

R^2 (Coeficiente de determinación)

- Mide qué proporción de la variabilidad de los datos se explica con el modelo.
- Responde a la pregunta:

¿Qué tan bien se ajusta mi modelo a los datos reales?

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

donde:
yi = valor real
y^i = predicción del modelo
$\bar{y}$ = media de los valores reales
n = número de observaciones

Interpretación:

$R^2=1$: Modelo perfecto (predicciones exactas).

$R^2=0$: El modelo no mejora respecto a simplemente usar la media.

$R^2<0$: El modelo es peor que usar la media (predicciones muy malas).

Modelo de Analisis de Sentimientos

¿Qué es el Análisis de Sentimientos?

Es una técnica de Procesamiento de Lenguaje Natural (NLP) que busca detectar la opinión o emoción detrás de un texto.

- “Me encantó la película” → Sentimiento positivo
- “La comida estaba fría y mala” → Sentimiento negativo
- “El servicio fue regular” → Sentimiento neutral

¿Cómo funciona un modelo de análisis de sentimientos?

Recolección de datos

- Se usan textos (reseñas, tweets, comentarios) previamente etiquetados con su sentimiento (positivo/negativo/neutral).

Preprocesamiento del texto

- Limpiar y normalizar el texto (quitar mayúsculas, signos, stopwords, acentos, emojis, etc.).
- Convertir palabras en representaciones numéricas (ejemplo: Bag of Words, TF-IDF o embeddings como Word2Vec, BERT).

¿Cómo funciona un modelo de análisis de sentimientos?

Entrenamiento del modelo

- Se aplica un algoritmo de machine learning o deep learning que aprende patrones.
- Modelos clásicos: Naive Bayes, SVM, Logistic Regression.
- Modelos modernos: redes neuronales recurrentes (LSTM), Transformers (BERT, RoBERTa, DistilBERT).

Predicción

- El modelo recibe un nuevo texto y predice la probabilidad de que sea positivo, negativo o neutral.

Métricas para evaluarlo

- Accuracy : qué porcentaje de predicciones fueron correctas.
- Precision y Recall : qué tan bien distingue entre clases.
- F1-score : balance entre precisión y exhaustividad.

spaCy (import spacy)

- Es una de las librerías más potentes para procesamiento de lenguaje natural.
- Permite hacer:
- Tokenización (separar texto en palabras).
- Lematización (reducir palabras a su forma base: “jugando” → “jugar”).
- Reconocimiento de entidades (personas, lugares, organizaciones).
- Análisis gramatical (dependencias sintácticas).
- Muy rápida y optimizada para proyectos en producción

string (import string)

- Biblioteca estándar de Python (no necesitas instalar nada).
- Contiene constantes útiles como:
- string.punctuation → lista de signos de puntuación.
- string.ascii_letters → letras del alfabeto en inglés.
- Útil cuando quieres limpiar texto (quitar puntos, comas, etc.).

nltk (import nltk)

- “Natural Language Toolkit”, otra librería muy usada en NLP.
- Se centra en tareas lingüísticas básicas:
- Tokenización de palabras y frases.
- Stopwords (palabras vacías como “el”, “la”, “and”).
- Stemming (reducir palabras a su raíz, ej. “running” → “run”).
- Corpus lingüísticos para entrenar modelos.

nltk (import nltk)

- “Natural Language Toolkit”, otra librería muy usada en NLP.
- Se centra en tareas lingüísticas básicas:
- Tokenización de palabras y frases.
- Stopwords (palabras vacías como “el”, “la”, “and”).
- Stemming (reducir palabras a su raíz, ej. “running” → “run”).
- Corpus lingüísticos para entrenar modelos.

Submódulos importados:

```
from nltk.tokenize import word_tokenize
```

unción para dividir texto en palabras.

```
"Hola mundo!" → ["Hola", "mundo", "!"]
```

```
from nltk.corpus import stopwords
```

contiene listas de palabras comunes sin mucho significado (ej. “de”, “la”, “the”).

re (import re)

Biblioteca estándar de Python para expresiones regulares.
Sirve para buscar y limpiar texto con patrones.

```
re.sub(r'\d+', '', 'Tengo 2 gatos y 3 perros')
# Resultado: "Tengo  gatos y  perros"
```

unidecode (from unidecode import unidecode)
Elimina acentos y caracteres especiales.

```
unidecode("canción") → "cancion"
```

gspread (import gspread)

- Sirve para conectarse y trabajar con Google Sheets desde Python.
- Puedes:
- Leer y escribir hojas de cálculo.
- Crear nuevas hojas.
- Compartir y actualizar datos en tiempo real.

pysentimiento (from pysentimiento import create_analyzer)

- Librería para análisis de sentimientos y emociones en varios idiomas (incluido español 🇪🇸).
- Utiliza modelos de deep learning preentrenados (basados en transformers).

```
analyzer = create_analyzer(task="sentiment", lang="es")
analyzer.predict("Me siento muy feliz")
# → SentimentResult(output=POS, probas={POS:0.9, NEU:0.08, NEG:0.02})
```

transformers (import transformers)

- Biblioteca de Hugging Face, la más usada para modelos modernos de NLP.
- Permite usar redes neuronales como BERT, RoBERTa, GPT, DistilBERT, etc.
- Se integra con pysentimiento, spacy y otras librerías.
- Con ella puedes:
- Cargar modelos preentrenados.
- Hacer clasificación de texto, traducción, generación de texto, etc.

BERT (Bidirectional Encoder Representations from Transformers)

¿Qué es BERT?

BERT es un modelo de lenguaje creado por Google en 2018.
Su gran innovación es que entiende el contexto de las palabras en ambas direcciones (izquierda y derecha) usando transformers.

Antes de BERT, muchos modelos leían un texto de izquierda a derecha (como un libro) o de derecha a izquierda. Eso limitaba la comprensión.

BERT en cambio es bidireccional, es decir, entiende el contexto completo de una palabra mirando antes y después de ella.

Basado en Transformers

- Usa la arquitectura Transformer (específicamente los encoders) que funcionan con atención (self-attention).
- Eso le permite darle más importancia a las palabras relevantes en una oración.

Bidireccionalidad

- Si lees “El banco está junto al río”, BERT entiende que “banco” aquí significa “asiento”, no “institución financiera”, porque ve el contexto completo.

Pre-entrenamiento masivo

BERT fue entrenado con miles de millones de palabras de Wikipedia y libros.

Para aprender, usó dos tareas principales:

- Masked Language Modeling (MLM): Oculta algunas palabras y le pide al modelo que las adivine.
- Ejemplo: “El [MASK] está ladrando” → debe predecir “perro”.

Next Sentence Prediction (NSP): Aprende si una oración sigue a otra en un texto.

Ejemplo:

- Input A: “Fui al cine ayer.”
- Input B: “Compré palomitas.” → Sí, es la siguiente.
- Input B: “El océano es muy profundo.” → No, no tiene relación.

Fine-tuning (ajuste fino)

- Una vez pre-entrenado, se puede “afinar” en tareas específicas como:
- Clasificación de sentimientos
- Preguntas y respuestas
- Reconocimiento de entidades (Nombres de personas, lugares, etc.)
- Traducción automática

RoBERTa (Robustly Optimized BERT Approach)

¿Qué es RoBERTa?

RoBERTa fue desarrollado por Facebook AI (Meta) en 2019 como una optimización de BERT.
La idea fue:

Principales diferencias con BERT

- Más datos de entrenamiento
- BERT: entrenado en Wikipedia + BookCorpus (~16GB de texto).
- RoBERTa: entrenado con 10 veces más datos (Wikipedia, CommonCrawl, noticias, etc., más de 160GB).

Más tiempo de entrenamiento y con batches más grandes

- RoBERTa se entrenó durante más pasos, con más texto en paralelo.
- Eso le permitió aprender patrones más profundos del lenguaje.

Sin Next Sentence Prediction (NSP)

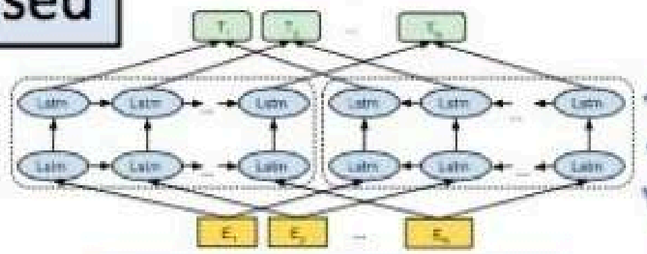
- En BERT, una de las tareas de preentrenamiento era NSP (predecir si una oración sigue a otra).
- RoBERTa descubrió que esta tarea no era necesaria y hasta podía perjudicar.
- En lugar de eso, usa solo Masked Language Modeling (MLM).

Más dinámico en el enmascarado (masking)

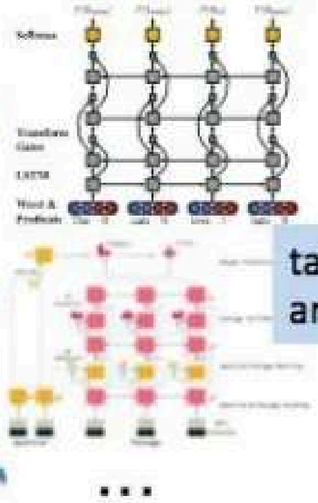
- BERT usaba el mismo patrón de palabras ocultas en cada época de entrenamiento.
- RoBERTa cambia el enmascarado en cada pasada → el modelo ve más variaciones → aprende mejor

Feature-based

ELMo



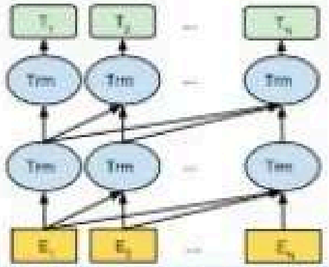
shallow concatenation of left-to-right and right-to-left



task-specific architecture

Fine-tuning

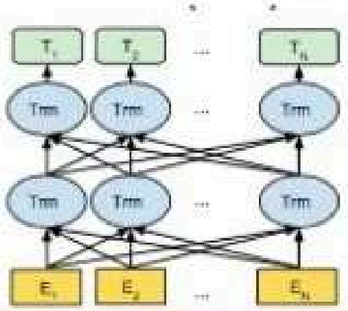
GPT



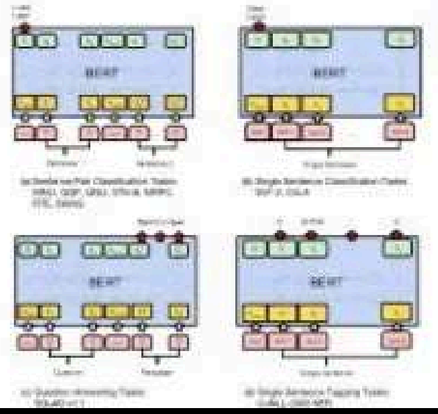
left-to-right language model

integrated architecture

BERT



bidirectional conditioning



Modelo	Tipo	¿Genera texto?	Velocidad	Precisión	Usos principales
BERT	Encoder	No	Media	Alta	Clasificación, NER, búsqueda semántica
RoBERTa	Encoder (mejora de BERT)	No	Media	Muy alta	Igual que BERT, pero más robusto
GPT	Decoder	Sí	Media-baja	Muy alta	Chatbots, generación de texto, traducción
DistilBERT	Encoder reducido	No	Rápido	Media-Alta	NLP en dispositivos, apps ligeras

Gracias por su atención